

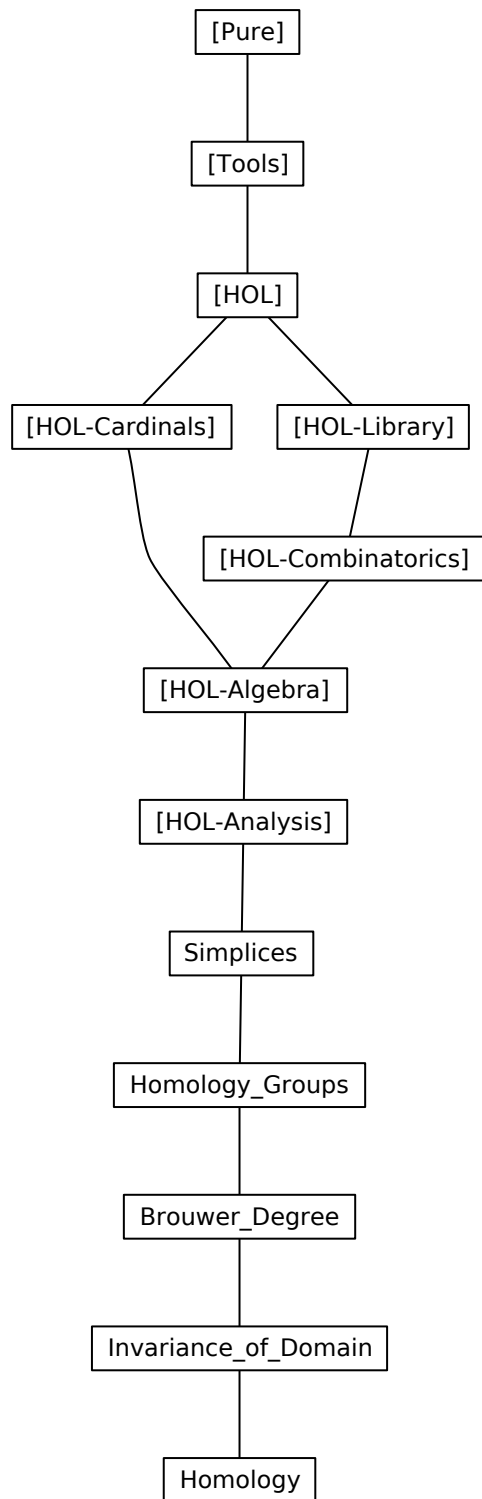
Homology

March 13, 2025

Contents

0.1	Homology, I: Simplices	4
0.1.1	Standard simplices, all of which are topological subspaces of \mathbb{R}^n	4
0.1.2	Face map	6
0.1.3	Singular simplices, forcing canonicity outside the intended domain	7
0.1.4	Singular chains	8
0.1.5	Boundary homomorphism for singular chains	9
0.1.6	Factoring out chains in a subtopology for relative homology	11
0.1.7	Relative cycles $Z_p X(S)$ where X is a topology and S a subset	11
0.1.8	Relative boundaries $B_p X(S)$, where X is a topology and S a subset.	12
0.1.9	The (relative) homology relation	14
0.1.10	Show that all boundaries are cycles, the key "chain complex" property.	16
0.1.11	Operations induced by a continuous map g between topological spaces	19
0.1.12	Homology of one-point spaces degenerates except for $p = 0$	23
0.1.13	Simplicial chains	26
0.1.14	The cone construction on simplicial simplices.	29
0.1.15	Barycentric subdivision of a linear ("simplicial") simplex's image	34
0.1.16	Singular subdivision	41
0.1.17	Excision argument that we keep doing singular subdivision	55
0.1.18	Homotopy invariance	67
0.2	Homology, II: Homology Groups	79
0.2.1	Homology Groups	79
0.2.2	Towards the Eilenberg-Steenrod axioms	94
0.2.3	Additivity axiom	106
0.2.4	Special properties of singular homology	115

0.2.5	More basic properties of homology groups, deduced from the E-S axioms	120
0.2.6	Generalize exact homology sequence to triples	130
0.3	Homology, III: Brouwer Degree	138
0.3.1	Reduced Homology	138
0.3.2	More homology properties of deformations, retracts, contractible spaces	147
0.3.3	Homology groups of spheres	156
0.3.4	Brouwer degree of a Map	169
0.4	Invariance of Domain	179
0.4.1	Degree invariance mod 2 for map between pairs . . .	179
0.4.2	General Jordan-Brouwer separation theorem and invariance of dimension	192
0.4.3	Relating two variants of Euclidean space, one within product topology.	225
0.4.4	Invariance of dimension and domain	230



0.1 Homology, I: Simplices

theory *Simplices*

imports

HOL-Analysis.Function_Metric

HOL-Analysis.Abstract_Euclidean_Space

HOL-Algebra.Free_Abelian_Groups

begin

0.1.1 Standard simplices, all of which are topological subspaces of $R^{\wedge n}$.

type_synonym 'a chain = ((nat \Rightarrow real) \Rightarrow 'a) \Rightarrow_0 int

definition *standard_simplex* :: nat \Rightarrow (nat \Rightarrow real) set **where**

standard_simplex p \equiv

$\{x. (\forall i. 0 \leq x\ i \wedge x\ i \leq 1) \wedge (\forall i > p. x\ i = 0) \wedge (\sum i \leq p. x\ i) = 1\}$

lemma *topspace_standard_simplex*:

topspace(subtopology (powertop_real UNIV) (standard_simplex p))

= standard_simplex p

by *simp*

lemma *basis_in_standard_simplex* [*simp*]:

$(\lambda j. \text{if } j = i \text{ then } 1 \text{ else } 0) \in \text{standard_simplex } p \longleftrightarrow i \leq p$

by (*auto simp: standard_simplex_def*)

lemma *nonempty_standard_simplex*: *standard_simplex p* $\neq \{\}$

using *basis_in_standard_simplex* **by** *blast*

lemma *standard_simplex_0*: *standard_simplex 0* = $\{(\lambda j. \text{if } j = 0 \text{ then } 1 \text{ else } 0)\}$

by (*auto simp: standard_simplex_def*)

lemma *standard_simplex_mono*:

assumes $p \leq q$

shows *standard_simplex p* \subseteq *standard_simplex q*

using *assms*

proof (*clarsimp simp: standard_simplex_def*)

fix $x :: \text{nat} \Rightarrow \text{real}$

assume $\forall i. 0 \leq x\ i \wedge x\ i \leq 1$ **and** $\forall i > p. x\ i = 0$ **and** $\text{sum } x \ \{..p\} = 1$

then show $\text{sum } x \ \{..q\} = 1$

using *sum.mono_neutral_left* [*of* $\{..q\}$ $\{..p\}$ x] *assms* **by** *auto*

qed

lemma *closedin_standard_simplex*:

closedin (powertop_real UNIV) (standard_simplex p)

(**is** *closedin* ?X ?S)

proof –

have *eq: standard_simplex p =*

```

      (⋂ i. {x. x ∈ topspace ?X ∧ x i ∈ {0..1}}) ∩
      (⋂ i ∈ {p<..}. {x ∈ topspace ?X. x i ∈ {0}}) ∩
      {x ∈ topspace ?X. (∑ i≤p. x i) ∈ {1}}
    by (auto simp: standard_simplex_def topspace_product_topology)
  show ?thesis
    unfolding eq
    by (rule closedin_Int closedin_Inter continuous_map_sum
        continuous_map_product_projection closedin_continuous_map_preimage
      | force | clarify)+
  qed

```

```

lemma standard_simplex_01: standard_simplex p ⊆ UNIV →E {0..1}
  using standard_simplex_def by auto

```

```

lemma compactin_standard_simplex:
  compactin (powertop_real UNIV) (standard_simplex p)
proof (rule closed_compactin)
  show compactin (powertop_real UNIV) (UNIV →E {0..1})
    by (simp add: compactin_PiE)
  show standard_simplex p ⊆ UNIV →E {0..1}
    by (simp add: standard_simplex_01)
  show closedin (powertop_real UNIV) (standard_simplex p)
    by (simp add: closedin_standard_simplex)
qed

```

```

lemma convex_standard_simplex:
  ⌊x ∈ standard_simplex p; y ∈ standard_simplex p;
    0 ≤ u; u ≤ 1⌋
  ⇒ (λi. (1 - u) * x i + u * y i) ∈ standard_simplex p
  by (simp add: standard_simplex_def sum.distrib convex_bound_le flip: sum_distrib_left)

```

```

lemma path_connectedin_standard_simplex:
  path_connectedin (powertop_real UNIV) (standard_simplex p)
proof -
  define g where g ≡ λx y::nat⇒real. λu i. (1 - u) * x i + u * y i
  have continuous_map
    (subtopology euclideanreal {0..1}) (powertop_real UNIV)
    (g x y)
  if x ∈ standard_simplex p y ∈ standard_simplex p for x y
  unfolding g_def continuous_map_componentwise
  by (force intro: continuous_intros)
  moreover
  have g x y ' {0..1} ⊆ standard_simplex p g x y 0 = x g x y 1 = y
    if x ∈ standard_simplex p y ∈ standard_simplex p for x y
  using that by (auto simp: convex_standard_simplex g_def)
  ultimately
  show ?thesis
    unfolding path_connectedin_def path_connected_space_def pathin_def
    by (metis continuous_map_in_subtopology euclidean_product_topology top_greatest

```

topspace_euclidean topspace_euclidean_subtopology)
qed

lemma *connectedin_standard_simplex*:
connectedin (powertop_real UNIV) (standard_simplex p)
by (*simp add: path_connectedin_imp_connectedin path_connectedin_standard_simplex*)

0.1.2 Face map

definition *simplicial_face* :: $\text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a::\text{comm_monoid_add}$
where

simplicial_face $k\ x \equiv \lambda i. \text{if } i < k \text{ then } x\ i \text{ else if } i = k \text{ then } 0 \text{ else } x(i - 1)$

lemma *simplicial_face_in_standard_simplex*:
assumes $1 \leq p$ $k \leq p$ $x \in \text{standard_simplex } (p - \text{Suc } 0)$
shows $(\text{simplicial_face } k\ x) \in \text{standard_simplex } p$

proof –

have $x01: \bigwedge i. 0 \leq x\ i \wedge x\ i \leq 1$ **and** $\text{sum}x: \text{sum } x \ \{..p - \text{Suc } 0\} = 1$
using *assms* **by** (*auto simp: standard_simplex_def simplicial_face_def*)

have $gg: \bigwedge g. \text{sum } g \ \{..p\} = \text{sum } g \ \{..<k\} + \text{sum } g \ \{k..p\}$
using $\langle k \leq p \rangle$ *sum.union_disjoint* [of $\{..<k\}$ $\{k..p\}$]

by (*force simp: ivl_disj_un ivl_disj_int*)

have $eq: (\sum i \leq p. \text{if } i < k \text{ then } x\ i \text{ else if } i = k \text{ then } 0 \text{ else } x\ (i - 1))$
 $= (\sum i < k. x\ i) + (\sum i \in \{k..p\}. \text{if } i = k \text{ then } 0 \text{ else } x\ (i - 1))$

by (*simp add: gg*)

consider $k \leq p - \text{Suc } 0 \mid k = p$

using $\langle k \leq p \rangle$ **by** *linarith*

then have $(\sum i \leq p. \text{if } i < k \text{ then } x\ i \text{ else if } i = k \text{ then } 0 \text{ else } x\ (i - 1)) = 1$

proof *cases*

case 1

have $[\text{simp}]: \text{Suc } (p - \text{Suc } 0) = p$

using $\langle 1 \leq p \rangle$ **by** *auto*

have $(\sum i = k..p. \text{if } i = k \text{ then } 0 \text{ else } x\ (i - 1)) = (\sum i = k+1..p. \text{if } i = k \text{ then } 0 \text{ else } x\ (i - 1))$

by (*rule sum.mono_neutral_right*) *auto*

also have $\dots = (\sum i = k+1..p. x\ (i - 1))$

by *simp*

also have $\dots = (\sum i = k..p-1. x\ i)$

using *sum.atLeastAtMost_reindex* [of $\text{Suc } k\ p-1\ \lambda i. x\ (i - \text{Suc } 0)$] 1 **by**

simp

finally have $eq2: (\sum i = k..p. \text{if } i = k \text{ then } 0 \text{ else } x\ (i - 1)) = (\sum i = k..p-1. x\ i)$.

with 1 **show** *?thesis*

by (*metis (no_types, lifting) One_nat_def eq_finite_atLeastAtMost_finite_lessThan ivl_disj_int(4) ivl_disj_un(10) sum.union_disjoint sumx*)

next

case 2

have $[\text{simp}]: (\{..p\} \cap \{x. x < p\}) = \{..p - \text{Suc } 0\}$

using *assms* **by** *auto*

```

    have (∑ i ≤ p. if i < p then x i else if i = k then 0 else x (i - 1)) = (∑ i ≤ p. if
i < p then x i else 0)
    by (rule sum.cong) (auto simp: 2)
    also have ... = sum x {..p-1}
    by (simp add: sum.If_cases)
    also have ... = 1
    by (simp add: sumx)
    finally show ?thesis
    using 2 by simp
qed
then show ?thesis
using assms by (auto simp: standard_simplex_def simplicial_face_def)
qed

```

0.1.3 Singular simplices, forcing canonicity outside the intended domain

definition *singular_simplex* :: $\text{nat} \Rightarrow 'a \text{ topology} \Rightarrow ((\text{nat} \Rightarrow \text{real}) \Rightarrow 'a) \Rightarrow \text{bool}$
where
singular_simplex p X f \equiv
 $\text{continuous_map}(\text{subtopology}(\text{powertop_real UNIV}) (\text{standard_simplex } p)) X$
 f
 $\wedge f \in \text{extensional} (\text{standard_simplex } p)$

abbreviation *singular_simplex_set* :: $\text{nat} \Rightarrow 'a \text{ topology} \Rightarrow ((\text{nat} \Rightarrow \text{real}) \Rightarrow 'a)$
set where
singular_simplex_set p X $\equiv \text{Collect} (\text{singular_simplex } p X)$

lemma *singular_simplex_empty*:
 $\text{topspace } X = \{\} \implies \neg \text{singular_simplex } p X f$
by (simp add: singular_simplex_def continuous_map_nonempty_standard_simplex)

lemma *singular_simplex_mono*:
 $\llbracket \text{singular_simplex } p (\text{subtopology } X T) f; T \subseteq S \rrbracket \implies \text{singular_simplex } p$
 $(\text{subtopology } X S) f$
by (auto simp: singular_simplex_def continuous_map_in_subtopology)

lemma *singular_simplex_subtopology*:
 $\text{singular_simplex } p (\text{subtopology } X S) f \longleftrightarrow$
 $\text{singular_simplex } p X f \wedge f' (\text{standard_simplex } p) \subseteq S$
by (auto simp: singular_simplex_def continuous_map_in_subtopology)

Singular face

definition *singular_face* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow ((\text{nat} \Rightarrow \text{real}) \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow \text{real}) \Rightarrow$
 $'a$
where *singular_face* p k f $\equiv \text{restrict} (f \circ \text{simplicial_face } k) (\text{standard_simplex}$
 $(p - \text{Suc } 0))$

lemma *singular_simplex_singular_face*:


```

    assumes  $f$ : singular_simplex  $p$   $X$   $f$  and  $1 \leq p$   $k \leq p$ 
    shows singular_simplex  $(p - \text{Suc } 0)$   $X$  (singular_face  $p$   $k$   $f$ )
  proof -
    let ?PT = (powertop_real UNIV)
    have 0: simplicial_face  $k$  'standard_simplex  $(p - \text{Suc } 0) \subseteq \text{standard\_simplex } p$ 
      using assms simplicial_face_in_standard_simplex by auto
    have 1: continuous_map (subtopology ?PT (standard_simplex  $(p - \text{Suc } 0)))$ 
      (subtopology ?PT (standard_simplex  $p$ ))
      (simplicial_face  $k$ )
    proof (clarsimp simp add: continuous_map_in_subtopology simplicial_face_in_standard_simplex
      continuous_map_componentwise 0)
      fix  $i$ 
      have continuous_map ?PT euclideanreal ( $\lambda x$ . if  $i < k$  then  $x$   $i$  else if  $i = k$ 
        then 0 else  $x$  ( $i - 1$ ))
        by (auto intro: continuous_map_product_projection)
      then show continuous_map (subtopology ?PT (standard_simplex  $(p - \text{Suc } 0)))$  euclideanreal
        ( $\lambda x$ . simplicial_face  $k$   $x$   $i$ )
        by (simp add: simplicial_face_def continuous_map_from_subtopology)
    qed
    have 2: continuous_map (subtopology ?PT (standard_simplex  $p$ ))  $X$   $f$ 
      using assms(1) singular_simplex_def by blast
    show ?thesis
      by (simp add: singular_simplex_def singular_face_def continuous_map_compose
        [OF 1 2])
  qed

```

0.1.4 Singular chains

definition *singular_chain* :: $[\text{nat}, 'a \text{ topology}, 'a \text{ chain}] \Rightarrow \text{bool}$
 where *singular_chain* p X $c \equiv \text{Poly_Mapping.keys } c \subseteq \text{singular_simplex_set } p$
 X

abbreviation *singular_chain_set* :: $[\text{nat}, 'a \text{ topology}] \Rightarrow ('a \text{ chain}) \text{ set}$
 where *singular_chain_set* p $X \equiv \text{Collect } (\text{singular_chain } p \ X)$

lemma *singular_chain_empty*:
 $\text{topspace } X = \{\} \implies \text{singular_chain } p \ X \ c \longleftrightarrow c = 0$
 by (auto simp: singular_chain_def singular_simplex_empty subset_eq poly_mapping_eqI)

lemma *singular_chain_mono*:
 $\llbracket \text{singular_chain } p \ (\text{subtopology } X \ T) \ c; \ T \subseteq S \rrbracket$
 $\implies \text{singular_chain } p \ (\text{subtopology } X \ S) \ c$
 unfolding singular_chain_def using singular_simplex_mono by blast

lemma *singular_chain_subtopology*:
 $\text{singular_chain } p \ (\text{subtopology } X \ S) \ c \longleftrightarrow$
 $\text{singular_chain } p \ X \ c \wedge (\forall f \in \text{Poly_Mapping.keys } c. f \ ' (\text{standard_simplex } p) \subseteq S)$

unfolding *singular_chain_def*
by (*fastforce simp add: singular_simplex_subtopology subset_eq*)

lemma *singular_chain_0* [*iff*]: *singular_chain p X 0*
by (*auto simp: singular_chain_def*)

lemma *singular_chain_of*:
 $singular_chain\ p\ X\ (frag_of\ c) \longleftrightarrow singular_simplex\ p\ X\ c$
by (*auto simp: singular_chain_def*)

lemma *singular_chain_cmul*:
 $singular_chain\ p\ X\ c \implies singular_chain\ p\ X\ (frag_cmul\ a\ c)$
by (*auto simp: singular_chain_def*)

lemma *singular_chain_minus*:
 $singular_chain\ p\ X\ (-c) \longleftrightarrow singular_chain\ p\ X\ c$
by (*auto simp: singular_chain_def*)

lemma *singular_chain_add*:
 $\llbracket singular_chain\ p\ X\ a; singular_chain\ p\ X\ b \rrbracket \implies singular_chain\ p\ X\ (a+b)$
unfolding *singular_chain_def*
using *keys_add* [*of a b*] **by** *blast*

lemma *singular_chain_diff*:
 $\llbracket singular_chain\ p\ X\ a; singular_chain\ p\ X\ b \rrbracket \implies singular_chain\ p\ X\ (a-b)$
unfolding *singular_chain_def*
using *keys_diff* [*of a b*] **by** *blast*

lemma *singular_chain_sum*:
 $(\bigwedge i. i \in I \implies singular_chain\ p\ X\ (f\ i)) \implies singular_chain\ p\ X\ (\sum_{i \in I} f\ i)$
unfolding *singular_chain_def*
using *keys_sum* [*of f I*] **by** *blast*

lemma *singular_chain_extend*:
 $(\bigwedge c. c \in Poly_Mapping.keys\ x \implies singular_chain\ p\ X\ (f\ c))$
 $\implies singular_chain\ p\ X\ (frag_extend\ f\ x)$
by (*simp add: frag_extend_def singular_chain_cmul singular_chain_sum*)

0.1.5 Boundary homomorphism for singular chains

definition *chain_boundary* :: *nat* \Rightarrow (*'a chain*) \Rightarrow *'a chain*
where *chain_boundary p c* \equiv
 $(if\ p = 0\ then\ 0\ else$
 $frag_extend\ (\lambda f. (\sum_{k \leq p} frag_cmul\ ((-1)^k) (frag_of (singular_face$
 $p\ k\ f))))\ c)$

lemma *singular_chain_boundary*:
assumes *singular_chain p X c*
shows *singular_chain (p - Suc 0) X (chain_boundary p c)*

```

unfolding chain_boundary_def
proof (clarsimp intro!: singular_chain_extend singular_chain_sum singular_chain_cmul)
  show  $\bigwedge d k. \llbracket 0 < p; d \in \text{Poly\_Mapping.keys } c; k \leq p \rrbracket$ 
     $\implies \text{singular\_chain } (p - \text{Suc } 0) \ X \ (\text{frag\_of } (\text{singular\_face } p \ k \ d))$ 
  using assms by (auto simp: singular_chain_def intro: singular_simplex_singular_face)
qed

```

```

lemma singular_chain_boundary_alt:
   $\text{singular\_chain } (\text{Suc } p) \ X \ c \implies \text{singular\_chain } p \ X \ (\text{chain\_boundary } (\text{Suc } p) \ c)$ 
using singular_chain_boundary by force

```

```

lemma chain_boundary_0 [simp]:  $\text{chain\_boundary } p \ 0 = 0$ 
by (simp add: chain_boundary_def)

```

```

lemma chain_boundary_cmul:
   $\text{chain\_boundary } p \ (\text{frag\_cmul } k \ c) = \text{frag\_cmul } k \ (\text{chain\_boundary } p \ c)$ 
by (auto simp: chain_boundary_def frag_extend_cmul)

```

```

lemma chain_boundary_minus:
   $\text{chain\_boundary } p \ (-c) = -(\text{chain\_boundary } p \ c)$ 
by (metis chain_boundary_cmul frag_cmul_minus_one)

```

```

lemma chain_boundary_add:
   $\text{chain\_boundary } p \ (a+b) = \text{chain\_boundary } p \ a + \text{chain\_boundary } p \ b$ 
by (simp add: chain_boundary_def frag_extend_add)

```

```

lemma chain_boundary_diff:
   $\text{chain\_boundary } p \ (a-b) = \text{chain\_boundary } p \ a - \text{chain\_boundary } p \ b$ 
using chain_boundary_add [of  $p \ a \ -b$ ]
by (simp add: chain_boundary_minus)

```

```

lemma chain_boundary_sum:
   $\text{chain\_boundary } p \ (\text{sum } g \ I) = \text{sum } (\text{chain\_boundary } p \ \circ g) \ I$ 
by (induction I rule: infinite_finite_induct) (simp_all add: chain_boundary_add)

```

```

lemma chain_boundary_sum':
   $\text{finite } I \implies \text{chain\_boundary } p \ (\text{sum}' g \ I) = \text{sum}' (\text{chain\_boundary } p \ \circ g) \ I$ 
by (induction I rule: finite_induct) (simp_all add: chain_boundary_add)

```

```

lemma chain_boundary_of:
   $\text{chain\_boundary } p \ (\text{frag\_of } f) =$ 
    (if  $p = 0$  then 0
     else  $(\sum_{k \leq p} \text{frag\_cmul } ((-1) \wedge k) \ (\text{frag\_of } (\text{singular\_face } p \ k \ f))))$ 
by (simp add: chain_boundary_def)

```

0.1.6 Factoring out chains in a subtopology for relative homology

definition *mod_subset*

where $\text{mod_subset } p \ X \equiv \{(a,b). \text{ singular_chain } p \ X \ (a - b)\}$

lemma *mod_subset_empty* [simp]:

$(a,b) \in (\text{mod_subset } p \ (\text{subtopology } X \ \{\})) \longleftrightarrow a = b$

by (simp add: mod_subset_def singular_chain_empty)

lemma *mod_subset_refl* [simp]: $(c,c) \in \text{mod_subset } p \ X$

by (auto simp: mod_subset_def)

lemma *mod_subset_cmul*:

assumes $(a,b) \in \text{mod_subset } p \ X$

shows $(\text{frag_cmul } k \ a, \text{ frag_cmul } k \ b) \in \text{mod_subset } p \ X$

using *assms*

by (simp add: mod_subset_def) (metis (no_types, lifting) add_diff_cancel diff_add_cancel frag_cmul_distrib2 singular_chain_cmul)

lemma *mod_subset_add*:

$\llbracket (c1,c2) \in \text{mod_subset } p \ X; (d1,d2) \in \text{mod_subset } p \ X \rrbracket \implies (c1+d1, c2+d2) \in \text{mod_subset } p \ X$

by (simp add: mod_subset_def add_diff_add singular_chain_add)

0.1.7 Relative cycles $Z_p X(S)$ where X is a topology and S a subset

definition *singular_relcycle* :: $\text{nat} \Rightarrow 'a \text{ topology} \Rightarrow 'a \text{ set} \Rightarrow ('a \text{ chain}) \Rightarrow \text{bool}$

where $\text{singular_relcycle } p \ X \ S \equiv$

$\lambda c. \text{ singular_chain } p \ X \ c \wedge (\text{chain_boundary } p \ c, 0) \in \text{mod_subset } (p-1) \ (\text{subtopology } X \ S)$

abbreviation *singular_relcycle_set*

where $\text{singular_relcycle_set } p \ X \ S \equiv \text{Collect } (\text{singular_relcycle } p \ X \ S)$

lemma *singular_relcycle_restrict* [simp]:

$\text{singular_relcycle } p \ X \ (\text{topspace } X \cap S) = \text{singular_relcycle } p \ X \ S$

proof –

have *eq*: $\text{subtopology } X \ (\text{topspace } X \cap S) = \text{subtopology } X \ S$

by (metis subtopology_subtopology subtopology_topospace)

show ?thesis

by (force simp: singular_relcycle_def eq)

qed

lemma *singular_relcycle*:

$\text{singular_relcycle } p \ X \ S \ c \longleftrightarrow$

$\text{singular_chain } p \ X \ c \wedge \text{singular_chain } (p-1) \ (\text{subtopology } X \ S) \ (\text{chain_boundary } p \ c)$

by (simp add: singular_relcycle_def mod_subset_def)

lemma *singular_relcycle_0* [simp]: *singular_relcycle* *p* *X* *S* 0
 by (auto simp: *singular_relcycle_def*)

lemma *singular_relcycle_cmul*:
singular_relcycle *p* *X* *S* *c* \implies *singular_relcycle* *p* *X* *S* (*frag_cmul* *k* *c*)
 by (auto simp: *singular_relcycle_def* *chain_boundary_cmul* *dest: singular_chain_cmul*
mod_subset_cmul)

lemma *singular_relcycle_minus*:
singular_relcycle *p* *X* *S* ($-c$) \longleftrightarrow *singular_relcycle* *p* *X* *S* *c*
 by (simp add: *chain_boundary_minus* *singular_chain_minus* *singular_relcycle*)

lemma *singular_relcycle_add*:
 $\llbracket \text{singular_relcycle } p \text{ } X \text{ } S \text{ } a; \text{ singular_relcycle } p \text{ } X \text{ } S \text{ } b \rrbracket$
 $\implies \text{singular_relcycle } p \text{ } X \text{ } S \text{ } (a+b)$
 by (simp add: *singular_relcycle_def* *chain_boundary_add* *mod_subset_def* *singular_chain_add*)

lemma *singular_relcycle_sum*:
 $\llbracket \bigwedge i. i \in I \implies \text{singular_relcycle } p \text{ } X \text{ } S \text{ } (f \text{ } i) \rrbracket$
 $\implies \text{singular_relcycle } p \text{ } X \text{ } S \text{ } (\text{sum } f \text{ } I)$
 by (induction *I* rule: *infinite_finite_induct*) (auto simp: *singular_relcycle_add*)

lemma *singular_relcycle_diff*:
 $\llbracket \text{singular_relcycle } p \text{ } X \text{ } S \text{ } a; \text{ singular_relcycle } p \text{ } X \text{ } S \text{ } b \rrbracket$
 $\implies \text{singular_relcycle } p \text{ } X \text{ } S \text{ } (a-b)$
 by (metis *singular_relcycle_add* *singular_relcycle_minus* *uminus_add_conv_diff*)

lemma *singular_cycle*:
singular_relcycle *p* *X* {} *c* \longleftrightarrow *singular_chain* *p* *X* *c* \wedge *chain_boundary* *p* *c* = 0
 using *mod_subset_empty* by (auto simp: *singular_relcycle_def*)

lemma *singular_cycle_mono*:
 $\llbracket \text{singular_relcycle } p \text{ } (\text{subtopology } X \text{ } T) \text{ } \{ \} \text{ } c; T \subseteq S \rrbracket$
 $\implies \text{singular_relcycle } p \text{ } (\text{subtopology } X \text{ } S) \text{ } \{ \} \text{ } c$
 by (auto simp: *singular_cycle_elim*: *singular_chain_mono*)

0.1.8 Relative boundaries B_pXS , where X is a topology and S a subset.

definition *singular_relboundary* :: *nat* \Rightarrow '*a* topology \Rightarrow '*a* set \Rightarrow ('*a* chain) \Rightarrow bool
 where
singular_relboundary *p* *X* *S* \equiv
 $\lambda c. \exists d. \text{singular_chain } (\text{Suc } p) \text{ } X \text{ } d \wedge (\text{chain_boundary } (\text{Suc } p) \text{ } d, c) \in$
 $(\text{mod_subset } p \text{ } (\text{subtopology } X \text{ } S))$

abbreviation *singular_relboundary_set* :: *nat* \Rightarrow '*a* *topology* \Rightarrow '*a* *set* \Rightarrow ('*a* *chain*) *set*

where *singular_relboundary_set* *p X S* \equiv *Collect* (*singular_relboundary* *p X S*)

lemma *singular_relboundary_restrict* [*simp*]:

singular_relboundary *p X* (*topspace* *X* \cap *S*) = *singular_relboundary* *p X S*

unfolding *singular_relboundary_def*

by (*metis* (*no_types*, *opaque_lifting*) *subtopology_subtopology* *subtopology_topspace*)

lemma *singular_relboundary_alt*:

singular_relboundary *p X S c* \longleftrightarrow

($\exists d e.$ *singular_chain* (*Suc* *p*) *X d* \wedge *singular_chain* *p* (*subtopology* *X S*) *e* \wedge *chain_boundary* (*Suc* *p*) *d* = *c* + *e*)

unfolding *singular_relboundary_def* *mod_subset_def* **by** *fastforce*

lemma *singular_relboundary*:

singular_relboundary *p X S c* \longleftrightarrow

($\exists d e.$ *singular_chain* (*Suc* *p*) *X d* \wedge *singular_chain* *p* (*subtopology* *X S*) *e* \wedge (*chain_boundary* (*Suc* *p*) *d*) + *e* = *c*)

using *singular_chain_minus*

by (*fastforce* *simp* *add*: *singular_relboundary_alt*)

lemma *singular_boundary*:

singular_relboundary *p X* {} *c* \longleftrightarrow

($\exists d.$ *singular_chain* (*Suc* *p*) *X d* \wedge *chain_boundary* (*Suc* *p*) *d* = *c*)

by (*meson* *mod_subset_empty* *singular_relboundary_def*)

lemma *singular_boundary_imp_chain*:

singular_relboundary *p X* {} *c* \implies *singular_chain* *p X c*

by (*auto* *simp*: *singular_relboundary* *singular_chain_boundary_alt* *singular_chain_empty*)

lemma *singular_boundary_mono*:

$\llbracket T \subseteq S; \textit{singular_relboundary } p (\textit{subtopology } X T) \{\} c \rrbracket$

$\implies \textit{singular_relboundary } p (\textit{subtopology } X S) \{\} c$

by (*metis* *mod_subset_empty* *singular_chain_mono* *singular_relboundary_def*)

lemma *singular_relboundary_imp_chain*:

singular_relboundary *p X S c* \implies *singular_chain* *p X c*

unfolding *singular_relboundary* *singular_chain_subtopology*

by (*blast* *intro*: *singular_chain_add* *singular_chain_boundary_alt*)

lemma *singular_chain_imp_relboundary*:

singular_chain *p* (*subtopology* *X S*) *c* \implies *singular_relboundary* *p X S c*

unfolding *singular_relboundary_def*

using *mod_subset_def* *singular_chain_minus* **by** *fastforce*

lemma *singular_relboundary_0* [*simp*]: *singular_relboundary* *p X S 0*

unfolding *singular_relboundary_def*

by (*rule_tac* *x=0* **in** *exI*) *auto*

lemma *singular_relboundary_cmul*:
 $\text{singular_relboundary } p \ X \ S \ c \implies \text{singular_relboundary } p \ X \ S \ (\text{frag_cmul } a \ c)$
unfolding *singular_relboundary_def*
by (*metis chain_boundary_cmul mod_subset_cmul singular_chain_cmul*)

lemma *singular_relboundary_minus*:
 $\text{singular_relboundary } p \ X \ S \ (-c) \longleftrightarrow \text{singular_relboundary } p \ X \ S \ c$
using *singular_relboundary_cmul*
by (*metis add.inverse_inverse frag_cmul_minus_one*)

lemma *singular_relboundary_add*:
 $\llbracket \text{singular_relboundary } p \ X \ S \ a; \text{singular_relboundary } p \ X \ S \ b \rrbracket \implies \text{singular_relboundary } p \ X \ S \ (a+b)$
unfolding *singular_relboundary_def*
by (*metis chain_boundary_add mod_subset_add singular_chain_add*)

lemma *singular_relboundary_diff*:
 $\llbracket \text{singular_relboundary } p \ X \ S \ a; \text{singular_relboundary } p \ X \ S \ b \rrbracket \implies \text{singular_relboundary } p \ X \ S \ (a-b)$
by (*metis uminus_add_conv_diff singular_relboundary_minus singular_relboundary_add*)

0.1.9 The (relative) homology relation

definition *homologous_rel* :: $[\text{nat}, 'a \text{ topology}, 'a \text{ set}, 'a \text{ chain}, 'a \text{ chain}] \Rightarrow \text{bool}$
where *homologous_rel* $p \ X \ S \equiv \lambda a \ b. \text{singular_relboundary } p \ X \ S \ (a-b)$

abbreviation *homologous_rel_set*
where *homologous_rel_set* $p \ X \ S \ a \equiv \text{Collect } (\text{homologous_rel } p \ X \ S \ a)$

lemma *homologous_rel_restrict* [*simp*]:
 $\text{homologous_rel } p \ X \ (\text{topspace } X \cap S) = \text{homologous_rel } p \ X \ S$
unfolding *homologous_rel_def* **by** (*metis singular_relboundary_restrict*)

lemma *homologous_rel_refl* [*simp*]: $\text{homologous_rel } p \ X \ S \ c \ c$
unfolding *homologous_rel_def* **by** *auto*

lemma *homologous_rel_sym*:
 $\text{homologous_rel } p \ X \ S \ a \ b = \text{homologous_rel } p \ X \ S \ b \ a$
unfolding *homologous_rel_def*
using *singular_relboundary_minus* **by** *fastforce*

lemma *homologous_rel_trans*:
assumes $\text{homologous_rel } p \ X \ S \ b \ c \ \text{homologous_rel } p \ X \ S \ a \ b$
shows $\text{homologous_rel } p \ X \ S \ a \ c$
using *homologous_rel_def*
proof –
have $\text{singular_relboundary } p \ X \ S \ (b - c)$
using *assms* **unfolding** *homologous_rel_def* **by** *blast*

```

moreover have singular_relboundary  $p\ X\ S\ (b - a)$ 
using assms by (meson homologous_rel_def homologous_rel_sym)
ultimately have singular_relboundary  $p\ X\ S\ (c - a)$ 
using singular_relboundary_diff by fastforce
then show ?thesis
by (meson homologous_rel_def homologous_rel_sym)
qed

```

```

lemma homologous_rel_eq:
  homologous_rel  $p\ X\ S\ a = \text{homologous\_rel}\ p\ X\ S\ b \longleftrightarrow$ 
  homologous_rel  $p\ X\ S\ a\ b$ 
using homologous_rel_sym homologous_rel_trans by fastforce

```

```

lemma homologous_rel_set_eq:
  homologous_rel_set  $p\ X\ S\ a = \text{homologous\_rel\_set}\ p\ X\ S\ b \longleftrightarrow$ 
  homologous_rel  $p\ X\ S\ a\ b$ 
by (metis homologous_rel_eq mem_Collect_eq)

```

```

lemma homologous_rel_singular_chain:
  homologous_rel  $p\ X\ S\ a\ b \implies (\text{singular\_chain}\ p\ X\ a \longleftrightarrow \text{singular\_chain}\ p\ X\ b)$ 
unfolding homologous_rel_def
using singular_chain_diff singular_chain_add
by (fastforce dest: singular_relboundary_imp_chain)

```

```

lemma homologous_rel_add:
   $\llbracket \text{homologous\_rel}\ p\ X\ S\ a\ a'; \text{homologous\_rel}\ p\ X\ S\ b\ b' \rrbracket$ 
   $\implies \text{homologous\_rel}\ p\ X\ S\ (a+b)\ (a'+b')$ 
unfolding homologous_rel_def
by (simp add: add_diff_add singular_relboundary_add)

```

```

lemma homologous_rel_diff:
  assumes homologous_rel  $p\ X\ S\ a\ a'$  homologous_rel  $p\ X\ S\ b\ b'$ 
  shows homologous_rel  $p\ X\ S\ (a - b)\ (a' - b')$ 
proof -
  have singular_relboundary  $p\ X\ S\ ((a - a') - (b - b'))$ 
    using assms singular_relboundary_diff unfolding homologous_rel_def by
    blast
  then show ?thesis
    by (simp add: homologous_rel_def algebra_simps)
qed

```

```

lemma homologous_rel_sum:
  assumes  $f$ : finite  $\{i \in I. f\ i \neq 0\}$  and  $g$ : finite  $\{i \in I. g\ i \neq 0\}$ 
  and  $h$ :  $\bigwedge i. i \in I \implies \text{homologous\_rel}\ p\ X\ S\ (f\ i)\ (g\ i)$ 
  shows homologous_rel  $p\ X\ S\ (\text{sum}\ f\ I)\ (\text{sum}\ g\ I)$ 
proof (cases finite  $I$ )
  case True
  let  $?L = \{i \in I. f\ i \neq 0\} \cup \{i \in I. g\ i \neq 0\}$ 
  have  $L$ : finite  $?L$   $?L \subseteq I$ 

```



```

    using f g by blast+
  have sum f I = sum f ?L
    by (rule comm_monoid_add_class.sum_mono_neutral_right [OF True]) auto
  moreover have sum g I = sum g ?L
    by (rule comm_monoid_add_class.sum_mono_neutral_right [OF True]) auto
  moreover have *: homologous_rel p X S (f i) (g i) if i ∈ ?L for i
    using h that by auto
  have homologous_rel p X S (sum f ?L) (sum g ?L)
    using L
  proof induction
    case (insert j J)
    then show ?case
      by (simp add: h homologous_rel_add)
  qed auto
  ultimately show ?thesis
    by simp
qed auto

```

lemma chain_homotopic_imp_homologous_rel:

```

  assumes
     $\bigwedge c. \text{singular\_chain } p \ X \ c \implies \text{singular\_chain } (\text{Suc } p) \ X' \ (h \ c)$ 
     $\bigwedge c. \text{singular\_chain } (p - 1) \ (\text{subtopology } X \ S) \ c \implies \text{singular\_chain } p \ (\text{subtopology } X' \ T) \ (h' \ c)$ 
     $\bigwedge c. \text{singular\_chain } p \ X \ c$ 
     $\implies (\text{chain\_boundary } (\text{Suc } p) \ (h \ c)) + (h'(\text{chain\_boundary } p \ c)) = f \ c$ 
     $- g \ c$ 
    singular_relcycle p X S c
  shows homologous_rel p X' T (f c) (g c)
proof -
  have singular_chain p (subtopology X' T) (chain_boundary (Suc p) (h c) - (f
c - g c))
    using assms
    by (metis (no_types, lifting) add_diff_cancel_left' minus_diff_eq singular_chain_minus
singular_relcycle)
  then show ?thesis
    using assms
    by (metis homologous_rel_def singular_relboundary singular_relcycle)
qed

```

0.1.10 Show that all boundaries are cycles, the key "chain complex" property.

lemma chain_boundary_boundary:

```

  assumes singular_chain p X c
  shows chain_boundary (p - Suc 0) (chain_boundary p c) = 0
proof (cases p - 1 = 0)
  case False
  then have 2 ≤ p

```

```

    by auto
  show ?thesis
    using assms
    unfolding singular_chain_def
  proof (induction rule: frag_induction)
    case (one g)
    then have ss: singular_simplex p X g
      by simp
    have eql:  $\{..p\} \times \{..p - \text{Suc } 0\} \cap \{(x, y). y < x\} = (\lambda(j, i). (\text{Suc } i, j)) \text{ ` } \{(i, j). i \leq j \wedge j \leq p - 1\}$ 
      using False
      by (auto simp: image_def) (metis One_nat_def diff_Suc_1 diff_le_mono le_refl lessE less_imp_le_nat)
    have eql:  $\{..p\} \times \{..p - \text{Suc } 0\} - \{(x, y). y < x\} = \{(i, j). i \leq j \wedge j \leq p - 1\}$ 
      by auto
    have eqf:  $\text{singular\_face } (p - \text{Suc } 0) \ i \ (\text{singular\_face } p \ (\text{Suc } j) \ g) = \text{singular\_face } (p - \text{Suc } 0) \ j \ (\text{singular\_face } p \ i \ g) \text{ if } i \leq j \wedge j \leq p - \text{Suc } 0$ 
    0 for i j
    proof (rule ext)
      fix t
      show  $\text{singular\_face } (p - \text{Suc } 0) \ i \ (\text{singular\_face } p \ (\text{Suc } j) \ g) \ t = \text{singular\_face } (p - \text{Suc } 0) \ j \ (\text{singular\_face } p \ i \ g) \ t$ 
      proof (cases t  $\in$  standard_simplex (p - 1 - 1))
        case True
        have fi:  $\text{simplicial\_face } i \ t \in \text{standard\_simplex } (p - \text{Suc } 0)$ 
          using False True simplicial_face_in_standard_simplex that by force
        have fj:  $\text{simplicial\_face } j \ t \in \text{standard\_simplex } (p - \text{Suc } 0)$ 
          by (metis False One_nat_def True simplicial_face_in_standard_simplex less_one not_less that(2))
        have eq:  $\text{simplicial\_face } (\text{Suc } j) \ (\text{simplicial\_face } i \ t) = \text{simplicial\_face } i \ (\text{simplicial\_face } j \ t)$ 
          using True that ss
          unfolding standard_simplex_def simplicial_face_def by fastforce
        show ?thesis by (simp add: singular_face_def fi fj eq)
      qed (simp add: singular_face_def)
    qed
  show ?case
  proof (cases p = 1)
    case False
    have eq0:  $\text{frag\_cmul } (-1) \ a = b \implies a + b = 0 \text{ for } a \ b$ 
      by (simp add: neg_eq_iff_add_eq_0)
    have *:  $(\sum x \leq p. \sum i \leq p - \text{Suc } 0. \text{frag\_cmul } ((-1) \wedge (x + i)) (\text{frag\_of } (\text{singular\_face } (p - \text{Suc } 0) \ i \ (\text{singular\_face } p \ x \ g)))) = 0$ 
      apply (simp add: sum.cartesian_product sum.Int_Diff [of _  $\times$  _  $\{(x, y). y < x\}$ ])
      apply (rule eq0)
    unfolding frag_cmul_sum_prod.case_distrib [of frag_cmul (-1)] frag_cmul_cmul

```

```

eql eqr
  apply (force simp: inj_on_def sum.reindex add.commute eqf intro: sum.cong)
  done
  show ?thesis
    using False by (simp add: chain_boundary_of chain_boundary_sum
chain_boundary_cmul frag_cmul_sum * flip: power_add)
  qed (simp add: chain_boundary_def)
next
  case (diff a b)
  then show ?case
    by (simp add: chain_boundary_diff)
  qed auto
qed (simp add: chain_boundary_def)

```

```

lemma chain_boundary_boundary_alt:
  singular_chain (Suc p) X c  $\implies$  chain_boundary p (chain_boundary (Suc p) c)
= 0
  using chain_boundary_boundary by force

```

```

lemma singular_relboundary_imp_relcycle:
  assumes singular_relboundary p X S c
  shows singular_relcycle p X S c
proof -
  obtain d e where d: singular_chain (Suc p) X d
    and e: singular_chain p (subtopology X S) e
    and c: c = chain_boundary (Suc p) d + e
  using assms by (auto simp: singular_relboundary singular_relcycle)
  have 1: singular_chain (p - Suc 0) (subtopology X S) (chain_boundary p
(chain_boundary (Suc p) d))
  using d chain_boundary_boundary_alt by fastforce
  have 2: singular_chain (p - Suc 0) (subtopology X S) (chain_boundary p e)
  using singular_chain p (subtopology X S) e singular_chain_boundary by
auto
  have singular_chain p X c
  using assms singular_relboundary_imp_chain by auto
  moreover have singular_chain (p - Suc 0) (subtopology X S) (chain_boundary
p c)
  by (simp add: c chain_boundary_add singular_chain_add 1 2)
  ultimately show ?thesis
  by (simp add: singular_relcycle)
qed

```

```

lemma homologous_rel_singular_relcycle_1:
  assumes homologous_rel p X S c1 c2 singular_relcycle p X S c1
  shows singular_relcycle p X S c2
  using assms
  by (metis diff_add_cancel homologous_rel_def homologous_rel_sym singular_relboundary_imp_relcycle
singular_relcycle_add)

```

```

lemma homologous_rel_singular_relcycle:
  assumes homologous_rel p X S c1 c2
  shows singular_relcycle p X S c1 = singular_relcycle p X S c2
  using assms homologous_rel_singular_relcycle_1
  using homologous_rel_sym by blast

```

0.1.11 Operations induced by a continuous map g between topological spaces

```

definition simplex_map :: nat  $\Rightarrow$  ('b  $\Rightarrow$  'a)  $\Rightarrow$  ((nat  $\Rightarrow$  real)  $\Rightarrow$  'b)  $\Rightarrow$  (nat  $\Rightarrow$  real)  $\Rightarrow$  'a
  where simplex_map p g c  $\equiv$  restrict (g  $\circ$  c) (standard_simplex p)

```

```

lemma singular_simplex_simplex_map:
   $\llbracket \textit{singular\_simplex } p \textit{ X f; continuous\_map } X \textit{ X' g} \rrbracket$ 
   $\implies \textit{singular\_simplex } p \textit{ X' (simplex\_map } p \textit{ g f)}$ 
  unfolding singular_simplex_def simplex_map_def
  by (auto simp: continuous_map_compose)

```

```

lemma simplex_map_eq:
   $\llbracket \textit{singular\_simplex } p \textit{ X c;}$ 
   $\bigwedge x. x \in \textit{topspace } X \implies f \textit{ x} = g \textit{ x} \rrbracket$ 
   $\implies \textit{simplex\_map } p \textit{ f c} = \textit{simplex\_map } p \textit{ g c}$ 
  by (auto simp: singular_simplex_def simplex_map_def continuous_map_def
Pi_iff)

```

```

lemma simplex_map_id_gen:
   $\llbracket \textit{singular\_simplex } p \textit{ X c;}$ 
   $\bigwedge x. x \in \textit{topspace } X \implies f \textit{ x} = x \rrbracket$ 
   $\implies \textit{simplex\_map } p \textit{ f c} = c$ 
  unfolding singular_simplex_def simplex_map_def continuous_map_def
  using extensional_arb by fastforce

```

```

lemma simplex_map_id [simp]:
  simplex_map p id = ( $\lambda c. \textit{restrict } c (\textit{standard\_simplex } p)$ )
  by (auto simp: simplex_map_def)

```

```

lemma simplex_map_compose:
  simplex_map p (h  $\circ$  g) = simplex_map p h  $\circ$  simplex_map p g
  unfolding simplex_map_def by force

```

```

lemma singular_face_simplex_map:
   $\llbracket 1 \leq p; k \leq p \rrbracket$ 
   $\implies \textit{singular\_face } p \textit{ k (simplex\_map } p \textit{ f c)} = \textit{simplex\_map } (p - \textit{Suc } 0) \textit{ f}$ 
  (c  $\circ$  simplicial_face k)
  unfolding simplex_map_def singular_face_def
  by (force simp: simplicial_face_in_standard_simplex)

```

lemma *singular_face_restrict* [simp]:
 assumes $p > 0$ $i \leq p$
 shows $\text{singular_face } p \ i \ (\text{restrict } f \ (\text{standard_simplex } p)) = \text{singular_face } p \ i \ f$
 by (metis *assms One_nat_def Suc_leI simplex_map_id singular_face_def singular_face_simplex_map*)

definition *chain_map* :: $\text{nat} \Rightarrow ('b \Rightarrow 'a) \Rightarrow ((\text{nat} \Rightarrow \text{real}) \Rightarrow 'b) \Rightarrow_0 \text{int} \Rightarrow 'a$
chain
 where $\text{chain_map } p \ g \ c \equiv \text{frag_extend } (\text{frag_of} \circ \text{simplex_map } p \ g) \ c$

lemma *singular_chain_chain_map*:
 $\llbracket \text{singular_chain } p \ X \ c; \text{continuous_map } X \ X' \ g \rrbracket \Longrightarrow \text{singular_chain } p \ X'$
 $(\text{chain_map } p \ g \ c)$
unfolding *chain_map_def*
by (force *simp add: singular_chain_def subset_iff*
intro!: singular_chain_extend singular_simplex_simplex_map)

lemma *chain_map_0* [simp]: $\text{chain_map } p \ g \ 0 = 0$
by (auto *simp: chain_map_def*)

lemma *chain_map_of* [simp]: $\text{chain_map } p \ g \ (\text{frag_of } f) = \text{frag_of } (\text{simplex_map } p \ g \ f)$
by (*simp add: chain_map_def*)

lemma *chain_map_cmul* [simp]:
 $\text{chain_map } p \ g \ (\text{frag_cmul } a \ c) = \text{frag_cmul } a \ (\text{chain_map } p \ g \ c)$
by (*simp add: frag_extend_cmul chain_map_def*)

lemma *chain_map_minus*: $\text{chain_map } p \ g \ (-c) = - (\text{chain_map } p \ g \ c)$
by (*simp add: frag_extend_minus chain_map_def*)

lemma *chain_map_add*:
 $\text{chain_map } p \ g \ (a+b) = \text{chain_map } p \ g \ a + \text{chain_map } p \ g \ b$
by (*simp add: frag_extend_add chain_map_def*)

lemma *chain_map_diff*:
 $\text{chain_map } p \ g \ (a-b) = \text{chain_map } p \ g \ a - \text{chain_map } p \ g \ b$
by (*simp add: frag_extend_diff chain_map_def*)

lemma *chain_map_sum*:
 $\text{finite } I \Longrightarrow \text{chain_map } p \ g \ (\text{sum } f \ I) = \text{sum } (\text{chain_map } p \ g \circ f) \ I$
by (*simp add: frag_extend_sum chain_map_def*)

lemma *chain_map_eq*:
 $\llbracket \text{singular_chain } p \ X \ c; \bigwedge x. x \in \text{topspace } X \Longrightarrow f \ x = g \ x \rrbracket$
 $\Longrightarrow \text{chain_map } p \ f \ c = \text{chain_map } p \ g \ c$
unfolding *singular_chain_def*
proof (*induction rule: frag_induction*)

```

    case (one x)
    then show ?case
    by (metis (no_types, lifting) chain_map_of mem_Collect_eq simplex_map_eq)
qed (auto simp: chain_map_diff)

```

```

lemma chain_map_id_gen:
   $\llbracket \text{singular\_chain } p \ X \ c; \bigwedge x. x \in \text{topspace } X \implies f \ x = x \rrbracket$ 
 $\implies \text{chain\_map } p \ f \ c = c$ 
unfolding singular_chain_def
by (erule frag_induction) (auto simp: chain_map_diff simplex_map_id_gen)

```

```

lemma chain_map_ident:
   $\text{singular\_chain } p \ X \ c \implies \text{chain\_map } p \ \text{id} \ c = c$ 
by (simp add: chain_map_id_gen)

```

```

lemma chain_map_id:
   $\text{chain\_map } p \ \text{id} = \text{frag\_extend } (\text{frag\_of} \circ (\lambda f. \text{restrict } f \ (\text{standard\_simplex } p)))$ 
by (auto simp: chain_map_def)

```

```

lemma chain_map_compose:
   $\text{chain\_map } p \ (h \circ g) = \text{chain\_map } p \ h \circ \text{chain\_map } p \ g$ 
proof
  show  $\text{chain\_map } p \ (h \circ g) \ c = (\text{chain\_map } p \ h \circ \text{chain\_map } p \ g) \ c$  for c
  using subset_UNIV
proof (induction c rule: frag_induction)
  case (one x)
  then show ?case
  by simp (metis (mono_tags, lifting) comp_eq_dest_lhs restrict_apply simplex_map_def)
next
  case (diff a b)
  then show ?case
  by (simp add: chain_map_diff)
qed auto
qed

```

```

lemma singular_simplex_chain_map_id:
  assumes  $\text{singular\_simplex } p \ X \ f$ 
  shows  $\text{chain\_map } p \ f \ (\text{frag\_of } (\text{restrict id } (\text{standard\_simplex } p))) = \text{frag\_of } f$ 
proof -
  have  $(\text{restrict } (f \circ \text{restrict id } (\text{standard\_simplex } p)) \ (\text{standard\_simplex } p)) = f$ 
  by (rule ext) (metis assms comp_apply extensional_arb id_apply restrict_apply singular_simplex_def)
  then show ?thesis
  by (simp add: simplex_map_def)
qed

```

```

lemma chain_boundary_chain_map:
  assumes  $\text{singular\_chain } p \ X \ c$ 

```

```

  shows chain_boundary p (chain_map p g c) = chain_map (p - Suc 0) g
(chain_boundary p c)
  using assms unfolding singular_chain_def
proof (induction c rule: frag_induction)
  case (one x)
  then have singular_face p i (simplex_map p g x) = simplex_map (p - Suc 0)
g (singular_face p i x)
  if  $0 \leq i \leq p$   $p \neq 0$  for i
  using that
  by (fastforce simp add: singular_face_def simplex_map_def simplicial_face_in_standard_simplex)
  then show ?case
  by (auto simp: chain_boundary_of_chain_map_sum)
next
  case (diff a b)
  then show ?case
  by (simp add: chain_boundary_diff chain_map_diff)
qed auto

lemma singular_relcycle_chain_map:
  assumes singular_relcycle p X S c continuous_map X X' g g ' S  $\subseteq$  T
  shows singular_relcycle p X' T (chain_map p g c)
proof -
  have continuous_map (subtopology X S) (subtopology X' T) g
  using assms
  using continuous_map_from_subtopology continuous_map_in_subtopology
topspace_subtopology by fastforce
  then show ?thesis
  using chain_boundary_chain_map [of p X c g]
  by (metis One_nat_def assms(1) assms(2) singular_chain_chain_map singular_relcycle)
qed

lemma singular_relboundary_chain_map:
  assumes singular_relboundary p X S c continuous_map X X' g g ' S  $\subseteq$  T
  shows singular_relboundary p X' T (chain_map p g c)
proof -
  obtain d e where d: singular_chain (Suc p) X d
  and e: singular_chain p (subtopology X S) e and c: c = chain_boundary (Suc
p) d + e
  using assms by (auto simp: singular_relboundary)
  have singular_chain (Suc p) X' (chain_map (Suc p) g d)
  using assms(2) d singular_chain_chain_map by blast
  moreover have singular_chain p (subtopology X' T) (chain_map p g e)
  proof -
    have  $\forall t. g ' \text{topspace } (\text{subtopology } t S) \subseteq T$ 
    by (metis assms(3) closure_of_subset_subtopology closure_of_topspace dual_order.trans
image_mono)
    then show ?thesis
    by (meson assms(2) continuous_map_from_subtopology continuous_map_in_subtopology

```

```

e singular_chain_chain_map)
qed
moreover have chain_boundary (Suc p) (chain_map (Suc p) g d) + chain_map
p g e =
    chain_map p g (chain_boundary (Suc p) d + e)
by (metis One_nat_def chain_boundary_chain_map chain_map_add d diff_Suc_1)
ultimately show ?thesis
    unfolding singular_relboundary
    using c by blast
qed

```

0.1.12 Homology of one-point spaces degenerates except for $p = 0$.

```

lemma singular_simplex_singleton:
  assumes topspace X = {a}
  shows singular_simplex p X f  $\longleftrightarrow$  f = restrict ( $\lambda x. a$ ) (standard_simplex p) (is
    ?lhs = ?rhs)
proof
  assume L: ?lhs
  then show ?rhs
  proof -
    have continuous_map (subtopology (product_topology ( $\lambda n. euclideanreal$ ) UNIV)
      (standard_simplex p)) X f
    using  $\langle$ singular_simplex p X f $\rangle$  singular_simplex_def by blast
    then have  $\bigwedge c. c \notin \text{standard\_simplex } p \vee f c = a$ 
    by (simp add: assms continuous_map_def Pi_iff)
    then show ?thesis
    by (metis (no_types) L extensional_restrict restrict_ext singular_simplex_def)
  qed
next
  assume ?rhs
  with assms show ?lhs
  by (auto simp: singular_simplex_def)
qed

```

```

lemma singular_chain_singleton:
  assumes topspace X = {a}
  shows singular_chain p X c  $\longleftrightarrow$ 
    ( $\exists b. c = \text{frag\_cmul } b (\text{frag\_of}(\text{restrict } (\lambda x. a) (\text{standard\_simplex } p)))$ )
    (is ?lhs = ?rhs)
proof
  let ?f = restrict ( $\lambda x. a$ ) (standard_simplex p)
  assume L: ?lhs
  with assms have Poly_Mapping.keys c  $\subseteq$  {?f}
  by (auto simp: singular_chain_def singular_simplex_singleton)
  then consider Poly_Mapping.keys c = {} | Poly_Mapping.keys c = {?f}
  by blast
  then show ?rhs

```



```

proof cases
  case 1
    with  $L$  show ?thesis
    by (metis frag_cmul_zero keys_eq_empty)
  next
    case 2
    then have  $\exists b. \text{frag\_extend frag\_of } c = \text{frag\_cmul } b (\text{frag\_of } (\lambda x \in \text{standard\_simplex } p. a))$ 
    by (force simp: frag_extend_def)
    then show ?thesis
    by (metis frag_expansion)
  qed
next
  assume ?rhs
  with  $assms$  show ?lhs
  by (auto simp: singular_chain_def singular_simplex_singleton)
qed

lemma chain_boundary_of_singleton:
  assumes  $tX$ :  $\text{topspace } X = \{a\}$  and  $sc$ :  $\text{singular\_chain } p \ X \ c$ 
  shows  $\text{chain\_boundary } p \ c =$ 
    (if  $p = 0 \vee \text{odd } p$  then 0
     else  $\text{frag\_extend } (\lambda f. \text{frag\_of}(\text{restrict } (\lambda x. a) (\text{standard\_simplex } (p-1))))$ )
  c)
  (is ?lhs = ?rhs)
proof (cases  $p = 0$ )
  case False
    have ?lhs =  $\text{frag\_extend } (\lambda f. \text{if odd } p \text{ then } 0 \text{ else } \text{frag\_of}(\text{restrict } (\lambda x. a) (\text{standard\_simplex } (p-1)))) \ c$ 
    proof (simp only: chain_boundary_def False if_False, rule frag_extend_eq)
      fix  $f$ 
      assume  $f \in \text{Poly\_Mapping.keys } c$ 
      with  $assms$  have  $\text{singular\_simplex } p \ X \ f$ 
      by (auto simp: singular_chain_def)
      then have  $\ast: \bigwedge k. k \leq p \implies \text{singular\_face } p \ k \ f = (\lambda x \in \text{standard\_simplex } (p-1). a)$ 
      using False singular_simplex_singular_face
      by (fastforce simp flip: singular_simplex_singleton [OF  $tX$ ])
      define  $c$  where  $c \equiv \text{frag\_of } (\lambda x \in \text{standard\_simplex } (p-1). a)$ 
      have  $(\sum_{k \leq p} \text{frag\_cmul } ((-1) \wedge k) (\text{frag\_of } (\text{singular\_face } p \ k \ f)))$ 
         $= (\sum_{k \leq p} \text{frag\_cmul } ((-1) \wedge k) \ c)$ 
      by (auto simp: c_def * intro: sum.cong)
      also have  $\dots = (\text{if odd } p \text{ then } 0 \text{ else } c)$ 
      by (induction  $p$ ) (auto simp: c_def restrict_def)
      finally show  $(\sum_{k \leq p} \text{frag\_cmul } ((-1) \wedge k) (\text{frag\_of } (\text{singular\_face } p \ k \ f)))$ 
         $= (\text{if odd } p \text{ then } 0 \text{ else } \text{frag\_of } (\lambda x \in \text{standard\_simplex } (p-1). a))$ 
      unfolding c_def .
    qed
  also have  $\dots = ?rhs$ 

```

```

    by (auto simp: False frag_extend_eq_0)
    finally show ?thesis .
qed (simp add: chain_boundary_def)

```

```

lemma singular_cycle_singleton:
  assumes topspace X = {a}
  shows singular_relcycle p X {} c  $\longleftrightarrow$  singular_chain p X c  $\wedge$  (p = 0  $\vee$  odd p  $\vee$  c = 0)
proof -
  have c = 0 if singular_chain p X c and chain_boundary p c = 0 and even p
  and p  $\neq$  0
  using that assms singular_chain_singleton [of X a p c] chain_boundary_of_singleton
  [OF assms]
  by (auto simp: frag_extend_cmul)
  moreover
  have chain_boundary p c = 0 if sc: singular_chain p X c and odd p
  by (simp add: chain_boundary_of_singleton [OF assms sc] that)
  moreover have chain_boundary 0 c = 0 if singular_chain 0 X c and p = 0
  by (simp add: chain_boundary_def)
  ultimately show ?thesis
  using assms by (auto simp: singular_cycle)
qed

```

```

lemma singular_boundary_singleton:
  assumes topspace X = {a}
  shows singular_relboundary p X {} c  $\longleftrightarrow$  singular_chain p X c  $\wedge$  (odd p  $\vee$  c = 0)
proof (cases singular_chain p X c)
  case True
  have  $\exists d. \text{singular\_chain } (\text{Suc } p) X d \wedge \text{chain\_boundary } (\text{Suc } p) d = c$ 
  if singular_chain p X c and odd p
  proof -
    obtain b where b: c = frag_cmul b (frag_of (restrict ( $\lambda x. a$ ) (standard_simplex p)))
    by (metis True assms singular_chain_singleton)
    let ?d = frag_cmul b (frag_of ( $\lambda x \in \text{standard\_simplex } (\text{Suc } p). a$ ))
    have scd: singular_chain (Suc p) X ?d
    by (metis assms singular_chain_singleton)
    moreover have chain_boundary (Suc p) ?d = c
    by (simp add: assms scd chain_boundary_of_singleton [of X a Suc p] b
    frag_extend_cmul <odd p>)
    ultimately show ?thesis
    by metis
  qed
  with True assms show ?thesis
  by (auto simp: singular_boundary_chain_boundary_of_singleton)
next

```

```

case False
with assms singular_boundary_imp_chain show ?thesis
by metis
qed

```

```

lemma singular_boundary_eq_cycle_singleton:
  assumes topspace X = {a} 1 ≤ p
  shows singular_relboundary p X {} c ⟷ singular_relcycle p X {} c (is ?lhs
= ?rhs)
proof
  show ?lhs ⟹ ?rhs
  by (simp add: singular_relboundary_imp_relcycle)
  show ?rhs ⟹ ?lhs
  by (metis assms not_one_le_zero singular_boundary_singleton singular_cycle_singleton)
qed

```

```

lemma singular_boundary_set_eq_cycle_singleton:
  assumes topspace X = {a} 1 ≤ p
  shows singular_relboundary_set p X {} = singular_relcycle_set p X {}
  using singular_boundary_eq_cycle_singleton [OF assms]
  by blast

```

0.1.13 Simplicial chains

Simplicial chains, effectively those resulting from linear maps. We still allow the map to be singular, so the name is questionable. These are intended as building-blocks for singular subdivision, rather than as a axis for 1 simplicial homology.

definition *oriented_simplex*

where $\text{oriented_simplex } p \ l \equiv (\lambda x \in \text{standard_simplex } p. \lambda i. (\sum_{j \leq p. l \ j \ i * x \ j}))$

definition *simplicial_simplex*

where

$\text{simplicial_simplex } p \ S \ f \equiv$
 $\text{singular_simplex } p \ (\text{subtopology } (\text{powertop_real UNIV}) \ S) \ f \wedge$
 $(\exists l. f = \text{oriented_simplex } p \ l)$

lemma *simplicial_simplex*:

$\text{simplicial_simplex } p \ S \ f \longleftrightarrow f \text{ ' } (\text{standard_simplex } p) \subseteq S \wedge (\exists l. f = \text{oriented_simplex } p \ l)$
(is ?lhs = ?rhs)

proof

assume R: ?rhs

have continuous_map (subtopology (powertop_real UNIV) (standard_simplex p))
 $(\text{powertop_real UNIV}) (\lambda x \ i. \sum_{j \leq p. l \ j \ i * x \ j)$ for $l :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

```

unfolding continuous_map_componentwise
  by (force intro: continuous_intros continuous_map_from_subtopology continuous_map_product_projection)
with R show ?lhs
  unfolding simplicial_simplex_def singular_simplex_subtopology
  by (auto simp add: singular_simplex_def oriented_simplex_def)
qed (simp add: simplicial_simplex_def singular_simplex_subtopology)

```

```

lemma simplicial_simplex_empty [simp]:  $\neg$  simplicial_simplex p {} f
  by (simp add: nonempty_standard_simplex simplicial_simplex)

```

```

definition simplicial_chain
  where simplicial_chain p S c  $\equiv$  Poly_Mapping.keys c  $\subseteq$  Collect (simplicial_simplex p S)

```

```

lemma simplicial_chain_0 [simp]: simplicial_chain p S 0
  by (simp add: simplicial_chain_def)

```

```

lemma simplicial_chain_of [simp]:
  simplicial_chain p S (frag_of c)  $\longleftrightarrow$  simplicial_simplex p S c
  by (simp add: simplicial_chain_def)

```

```

lemma simplicial_chain_cmul:
  simplicial_chain p S c  $\implies$  simplicial_chain p S (frag_cmul a c)
  by (auto simp: simplicial_chain_def)

```

```

lemma simplicial_chain_diff:
   $\llbracket$ simplicial_chain p S c1; simplicial_chain p S c2 $\rrbracket \implies$  simplicial_chain p S (c1 - c2)
  unfolding simplicial_chain_def by (meson UnE keys_diff subset_iff)

```

```

lemma simplicial_chain_sum:
   $(\bigwedge i. i \in I \implies \text{simplicial\_chain } p \ S \ (f \ i)) \implies \text{simplicial\_chain } p \ S \ (\text{sum } f \ I)$ 
  unfolding simplicial_chain_def
  using order_trans [OF keys_sum [of f I]]
  by (simp add: UN_least)

```

```

lemma simplicial_simplex_oriented_simplex:
  simplicial_simplex p S (oriented_simplex p l)
   $\longleftrightarrow ((\lambda x \ i. \sum_{j \leq p. l \ j \ i * x \ j}) \text{ 'standard\_simplex } p \subseteq S)$ 
  by (auto simp: simplicial_simplex oriented_simplex_def)

```

```

lemma simplicial_imp_singular_simplex:
  simplicial_simplex p S f
   $\implies$  singular_simplex p (subtopology (powertop_real UNIV) S) f
  by (simp add: simplicial_simplex_def)

```

```

lemma simplicial_imp_singular_chain:
  simplicial_chain p S c

```

```

     $\implies$  singular_chain p (subtopology (powertop_real UNIV) S) c
  unfolding simplicial_chain_def singular_chain_def
  by (auto intro: simplicial_imp_singular_simplex)

```

lemma oriented_simplex_eq:

```

  oriented_simplex p l = oriented_simplex p l'  $\longleftrightarrow$  ( $\forall i. i \leq p \longrightarrow l\ i = l'\ i$ )
  (is ?lhs = ?rhs)

```

proof

assume L: ?lhs

show ?rhs

proof clarify

fix i

assume i \leq p

let ?fi = ($\lambda j. \text{if } j = i \text{ then } 1 \text{ else } 0$)

have ($\sum j \leq p. l\ j\ k * ?fi\ j$) = ($\sum j \leq p. l'\ j\ k * ?fi\ j$) **for** k

using L $\langle i \leq p \rangle$

by (simp add: fun_eq_iff oriented_simplex_def split: if_split_asm)

with $\langle i \leq p \rangle$ **show** l i = l' i

by (simp add: if_distrib ext cong: if_cong)

qed

qed (auto simp: oriented_simplex_def)

lemma singular_face_oriented_simplex:

assumes 1 \leq p k \leq p

shows singular_face p k (oriented_simplex p l) =

oriented_simplex (p - 1) ($\lambda j. \text{if } j < k \text{ then } l\ j \text{ else } l\ (\text{Suc } j)$)

proof -

have ($\sum j \leq p. l\ j\ i * \text{simplicial_face } k\ x\ j$)

= ($\sum j \leq p - \text{Suc } 0. (\text{if } j < k \text{ then } l\ j \text{ else } l\ (\text{Suc } j))\ i * x\ j$)

if x \in standard_simplex (p - Suc 0) **for** i x

proof -

show ?thesis

unfolding simplicial_face_def

using sum.zero_middle [OF assms, **where** 'a=real, symmetric]

by (simp add: if_distrib [of $\lambda x. _ * x$] if_distrib [of $\lambda f. f\ i * _$] atLeast0AtMost

cong: if_cong)

qed

then show ?thesis

using simplicial_face_in_standard_simplex assms

by (auto simp: singular_face_def oriented_simplex_def restrict_def)

qed

lemma simplicial_simplex_singular_face:

fixes f :: (nat \Rightarrow real) \Rightarrow nat \Rightarrow real

assumes ss: simplicial_simplex p S f **and** p: 1 \leq p k \leq p

shows simplicial_simplex (p - Suc 0) S (singular_face p k f)

proof -

let ?X = subtopology (powertop_real UNIV) S

obtain m **where** l: singular_simplex p ?X (oriented_simplex p m)

```

    and feq: f = oriented_simplex p m
    using assms by (force simp: simplicial_simplex_def)
  moreover
    have singular_face p k f = oriented_simplex (p - Suc 0) ( $\lambda i$ . if  $i < k$  then  $m i$  else  $m (Suc i)$ )
    using feq p singular_face_oriented_simplex by auto
  ultimately
    show ?thesis
    using p simplicial_simplex_def singular_simplex_singular_face by blast
qed

```

```

lemma simplicial_chain_boundary:
  simplicial_chain p S c  $\implies$  simplicial_chain (p - 1) S (chain_boundary p c)
  unfolding simplicial_chain_def
proof (induction rule: frag_induction)
  case (one f)
  then have simplicial_simplex p S f
    by simp
  have simplicial_chain (p - Suc 0) S (frag_of (singular_face p i f))
    if  $0 < p i \leq p$  for  $i$ 
    using that one
    by (force simp: simplicial_simplex_def singular_simplex_singular_face singular_face_oriented_simplex)
  then have simplicial_chain (p - Suc 0) S (chain_boundary p (frag_of f))
    unfolding chain_boundary_def frag_extend_of
    by (auto intro!: simplicial_chain_cmul simplicial_chain_sum)
  then show ?case
    by (simp add: simplicial_chain_def [symmetric])
next
  case (diff a b)
  then show ?case
    by (metis chain_boundary_diff simplicial_chain_def simplicial_chain_diff)
qed auto

```

0.1.14 The cone construction on simplicial simplices.

```

consts simplex_cone :: [nat, nat  $\Rightarrow$  real, [nat  $\Rightarrow$  real, nat]  $\Rightarrow$  real, nat  $\Rightarrow$  real,
nat]  $\Rightarrow$  real
specification (simplex_cone)
  simplex_cone:
     $\bigwedge p v l$ . simplex_cone p v (oriented_simplex p l) =
      oriented_simplex (Suc p) ( $\lambda i$ . if  $i = 0$  then  $v$  else  $l(i - 1)$ )
proof -
  have *:  $\bigwedge x$ .  $\forall xv$ .  $\exists y$ . ( $\lambda l$ . oriented_simplex (Suc x)
    ( $\lambda i$ . if  $i = 0$  then  $xv$  else  $l(i - 1)$ )) =
    y  $\circ$  oriented_simplex x
  by (simp add: oriented_simplex_eq_flip: choice_iff function_factors_left)
then show ?thesis
  unfolding o_def by (metis (no_types))

```

qed

lemma *simplicial_simplex_simplex_cone*:

assumes f : *simplicial_simplex* p S f
 and T : $\bigwedge x u. \llbracket 0 \leq u; u \leq 1; x \in S \rrbracket \implies (\lambda i. (1 - u) * v\ i + u * x\ i) \in T$
 shows *simplicial_simplex* (*Suc* p) T (*simplex_cone* p v f)
 proof -
 obtain l where l : $\bigwedge x. x \in \text{standard_simplex } p \implies \text{oriented_simplex } p\ l\ x \in S$
 and f : $f = \text{oriented_simplex } p\ l$
 using f by (auto simp: *simplicial_simplex*)
 have *oriented_simplex* $p\ l\ x \in S$ if $x \in \text{standard_simplex } p$ for x
 using f that by (auto simp: *simplicial_simplex* f)
 then have S : $\bigwedge x. \llbracket \bigwedge i. 0 \leq x\ i \wedge x\ i \leq 1; \bigwedge i. i > p \implies x\ i = 0; \text{sum } x\ \{..p\} = 1 \rrbracket$
 $\implies (\lambda i. \sum_{j \leq p.} l\ j\ i * x\ j) \in S$
 by (simp add: *oriented_simplex_def* *standard_simplex_def*)
 have *oriented_simplex* (*Suc* p) ($\lambda i. \text{if } i = 0 \text{ then } v \text{ else } l\ (i - 1)$) $x \in T$
 if $x \in \text{standard_simplex } (\text{Suc } p)$ for x
 proof (simp add: *that oriented_simplex_def* *sum.atMost_Suc_shift* *del: sum.atMost_Suc*)
 have $x01$: $\bigwedge i. 0 \leq x\ i \wedge x\ i \leq 1$ and $x0$: $\bigwedge i. i > \text{Suc } p \implies x\ i = 0$ and $x1$:
 $\text{sum } x\ \{..\text{Suc } p\} = 1$
 using that by (auto simp: *oriented_simplex_def* *standard_simplex_def*)
 obtain a where $a \in S$
 using f by force
 show $(\lambda i. v\ i * x\ 0 + (\sum_{j \leq p.} l\ j\ i * x\ (\text{Suc } j))) \in T$
 proof (cases $x\ 0 = 1$)
 case True
 then have $\text{sum } x\ \{\text{Suc } 0..\text{Suc } p\} = 0$
 using $x1$ by (simp add: *atMost_atLeast0* *sum.atLeast_Suc_atMost*)
 then have $[simp]$: $x\ (\text{Suc } j) = 0$ if $j \leq p$ for j
 unfolding *sum.atLeast_Suc_atMost_Suc_shift*
 using $x01$ that by (simp add: *sum_nonneg_eq_0_iff*)
 then show ?thesis
 using T [of 0 a] $\langle a \in S \rangle$ by (auto simp: True)
 next
 case False
 then have $(\lambda i. v\ i * x\ 0 + (\sum_{j \leq p.} l\ j\ i * x\ (\text{Suc } j))) = (\lambda i. (1 - (1 - x\ 0)) * v\ i + (1 - x\ 0) * (\text{inverse } (1 - x\ 0) * (\sum_{j \leq p.} l\ j\ i * x\ (\text{Suc } j))))$
 by (force simp: *field_simps*)
 also have $\dots \in T$
 proof (rule T)
 have $x\ 0 < 1$
 by (simp add: *False less_le* $x01$)
 have xle : $x\ (\text{Suc } i) \leq (1 - x\ 0)$ for i
 proof (cases $i \leq p$)
 case True
 have $\text{sum } x\ \{0, \text{Suc } i\} \leq \text{sum } x\ \{..\text{Suc } p\}$
 by (rule *sum_mono2*) (auto simp: True $x01$)
 then show ?thesis

```

    using x1 x01 by (simp add: algebra_simps not_less)
  qed (simp add: x0 x01)
  have (λi. (∑ j≤p. l j i * (x (Suc j) * inverse (1 - x 0)))) ∈ S
  proof (rule S)
    have x 0 + (∑ j≤p. x (Suc j)) = sum x {..Suc p}
    by (metis sum.atMost_Suc_shift)
    with x1 have (∑ j≤p. x (Suc j)) = 1 - x 0
    by simp
    with False show (∑ j≤p. x (Suc j) * inverse (1 - x 0)) = 1
    by (metis add_diff_cancel_left' diff_diff_eq2 diff_zero right_inverse
sum_distrib_right)
  qed (use x01 x0 xle ⟨x 0 < 1⟩ in ⟨auto simp: field_split_simps⟩)
  then show (λi. inverse (1 - x 0) * (∑ j≤p. l j i * x (Suc j))) ∈ S
  by (simp add: field_simps sum_divide_distrib)
  qed (use x01 in auto)
  finally show ?thesis .
qed
qed
then show ?thesis
  by (auto simp: simplicial_simplex feq simplex_cone)
qed

```

definition *simplicial_cone*

where *simplicial_cone* $p\ v \equiv \text{frag_extend} (\text{frag_of} \circ \text{simplex_cone } p\ v)$

lemma *simplicial_chain_simplicial_cone*:

assumes c : *simplicial_chain* $p\ S\ c$

and T : $\bigwedge x\ u. \llbracket 0 \leq u; u \leq 1; x \in S \rrbracket \implies (\lambda i. (1 - u) * v\ i + u * x\ i) \in T$

shows *simplicial_chain* (Suc p) T (*simplicial_cone* $p\ v\ c$)

using c **unfolding** *simplicial_chain_def* *simplicial_cone_def*

proof (*induction* rule: *frag_induction*)

case (*one* x)

then show ?case

by (simp add: *T* *simplicial_simplex_simplex_cone*)

next

case (*diff* $a\ b$)

then show ?case

by (metis *frag_extend_diff* *simplicial_chain_def* *simplicial_chain_diff*)

qed *auto*

lemma *chain_boundary_simplicial_cone_of'*:

assumes $f = \text{oriented_simplex } p\ l$

shows *chain_boundary* (Suc p) (*simplicial_cone* $p\ v$ (*frag_of* f)) =

frag_of f

– (if $p = 0$ then *frag_of* ($\lambda u \in \text{standard_simplex } p. v$)

else *simplicial_cone* ($p - 1$) v (*chain_boundary* p (*frag_of* f)))

proof (*simp*, *intro* *impI* *conjI*)

assume $p = 0$


```

have eq: (oriented_simplex 0 ( $\lambda j$ . if  $j = 0$  then  $v$  else  $l\ j$ )) = ( $\lambda u \in \text{standard\_simplex } 0$ .  $v$ )
by (force simp: oriented_simplex_def standard_simplex_def)
show chain_boundary (Suc 0) (simplicial_cone 0  $v$  (frag_of  $f$ ))
  = frag_of  $f$  - frag_of ( $\lambda u \in \text{standard\_simplex } 0$ .  $v$ )
by (simp add: assms simplicial_cone_def chain_boundary_of  $\langle p = 0 \rangle$  simplex_cone_singular_face_oriented_simplex eq cong: if_cong)
next
assume  $0 < p$ 
have 0: simplex_cone ( $p - \text{Suc } 0$ )  $v$  (singular_face  $p\ x$  (oriented_simplex  $p\ l$ ))
  = oriented_simplex  $p$ 
    ( $\lambda j$ . if  $j < \text{Suc } x$ 
      then if  $j = 0$  then  $v$  else  $l\ (j - 1)$ 
      else if  $\text{Suc } j = 0$  then  $v$  else  $l\ (\text{Suc } j - 1)$ ) if  $x \leq p$  for  $x$ 
using  $\langle 0 < p \rangle$  that
by (auto simp: Suc_leI singular_face_oriented_simplex simplex_cone oriented_simplex_eq)
have 1: frag_extend (frag_of  $\circ$  simplex_cone ( $p - \text{Suc } 0$ )  $v$ )
  ( $\sum k = 0..p$ . frag_cmul  $((-1) ^ k)$  (frag_of (singular_face  $p\ k$  (oriented_simplex  $p\ l$ ))))
  = - ( $\sum k = \text{Suc } 0..p$ . frag_cmul  $((-1) ^ k)$ 
    (frag_of (singular_face ( $\text{Suc } p$ )  $k$  (simplex_cone  $p\ v$  (oriented_simplex  $p\ l$ ))))))
unfolding sum.atLeast_Suc_atMost_Suc_shift
by (auto simp: 0 simplex_cone_singular_face_oriented_simplex frag_extend_sum frag_extend_cmul simp flip: sum_negf)
moreover have 2: singular_face ( $\text{Suc } p$ ) 0 (simplex_cone  $p\ v$  (oriented_simplex  $p\ l$ ))
  = oriented_simplex  $p\ l$ 
by (simp add: simplex_cone_singular_face_oriented_simplex)
show chain_boundary (Suc  $p$ ) (simplicial_cone  $p\ v$  (frag_of  $f$ ))
  = frag_of  $f$  - simplicial_cone ( $p - \text{Suc } 0$ )  $v$  (chain_boundary  $p$  (frag_of  $f$ ))
using  $\langle p > 0 \rangle$ 
apply (simp add: assms simplicial_cone_def chain_boundary_of atMost_atLeast0 del: sum.atMost_Suc)
apply (subst sum.atLeast_Suc_atMost [of 0])
apply (simp_all add: 1 2 del: sum.atMost_Suc)
done
qed

```

lemma chain_boundary_simplicial_cone_of:

assumes simplicial_simplex $p\ S\ f$

shows chain_boundary (Suc p) (simplicial_cone $p\ v$ (frag_of f)) =
frag_of f

- (if $p = 0$ then frag_of ($\lambda u \in \text{standard_simplex } p$. v)
else simplicial_cone ($p - 1$) v (chain_boundary p (frag_of f)))

using chain_boundary_simplicial_cone_of' **assms** **unfolding** simplicial_simplex_def
by blast

```

lemma chain_boundary_simplicial_cone:
  simplicial_chain p S c
   $\implies$  chain_boundary (Suc p) (simplicial_cone p v c) =
    c - (if p = 0 then frag_extend ( $\lambda f$ . frag_of ( $\lambda u \in \text{standard\_simplex } p$ . v)) c
      else simplicial_cone (p - 1) v (chain_boundary p c))
  unfolding simplicial_chain_def
proof (induction rule: frag_induction)
  case (one x)
  then show ?case
    by (auto simp: chain_boundary_simplicial_cone_of)
qed (auto simp: chain_boundary_diff simplicial_cone_def frag_extend_diff)

```

```

lemma simplex_map_oriented_simplex:
  assumes l: simplicial_simplex p (standard_simplex q) (oriented_simplex p l)
  and g: simplicial_simplex r S g and q  $\leq$  r
  shows simplex_map p g (oriented_simplex p l) = oriented_simplex p (g  $\circ$  l)
proof -
  obtain m where geq: g = oriented_simplex r m
  using g by (auto simp: simplicial_simplex_def)
  have g ( $\lambda i$ .  $\sum_{j \leq p} l \ j \ i * x \ j$ ) i = ( $\sum_{j \leq p} g \ (l \ j) \ i * x \ j$ )
    if x  $\in$  standard_simplex p for x i
  proof -
  have ssr: ( $\lambda i$ .  $\sum_{j \leq p} l \ j \ i * x \ j$ )  $\in$  standard_simplex r
    using l that standard_simplex_mono [OF  $\langle q \leq r \rangle$ ]
  unfolding simplicial_simplex_oriented_simplex by auto
  have lss: l j  $\in$  standard_simplex r if j  $\leq$  p for j
  proof -
  have q: ( $\lambda x \ i$ .  $\sum_{j \leq p} l \ j \ i * x \ j$ ) ' standard_simplex p  $\subseteq$  standard_simplex q
    using l by (simp add: simplicial_simplex_oriented_simplex)
  let ?x = ( $\lambda i$ . if i = j then 1 else 0)
  have p: l j  $\in$  ( $\lambda x \ i$ .  $\sum_{j \leq p} l \ j \ i * x \ j$ ) ' standard_simplex p
  proof
    show l j = ( $\lambda i$ .  $\sum_{j \leq p} l \ j \ i * ?x \ j$ )
      using  $\langle j \leq p \rangle$  by (force simp: if_distrib cong: if_cong)
    show ?x  $\in$  standard_simplex p
      by (simp add: that)
  qed
  show ?thesis
    using standard_simplex_mono [OF  $\langle q \leq r \rangle$ ] q p
    by blast
  qed
  have g ( $\lambda i$ .  $\sum_{j \leq p} l \ j \ i * x \ j$ ) i = ( $\sum_{j \leq r} \sum_{n \leq p} m \ j \ i * (l \ n \ j * x \ n)$ )
    by (simp add: geq oriented_simplex_def sum_distrib_left ssr)
  also have ... = ( $\sum_{j \leq p} \sum_{n \leq r} m \ n \ i * (l \ j \ n * x \ j)$ )
    by (rule sum.swap)
  also have ... = ( $\sum_{j \leq p} g \ (l \ j) \ i * x \ j$ )
    by (simp add: geq oriented_simplex_def sum_distrib_right mult.assoc lss)
  finally show ?thesis .

```

```

qed
then show ?thesis
  by (force simp: oriented_simplex_def simplex_map_def o_def)
qed

lemma chain_map_simplicial_cone:
  assumes g: simplicial_simplex r S g
    and c: simplicial_chain p (standard_simplex q) c
    and v: v ∈ standard_simplex q and q ≤ r
  shows chain_map (Suc p) g (simplicial_cone p v c) = simplicial_cone p (g v)
    (chain_map p g c)
proof -
  have *: simplex_map (Suc p) g (simplex_cone p v f) = simplex_cone p (g v)
    (simplex_map p g f)
  if f ∈ Poly_Mapping.keys c for f
  proof -
    have simplicial_simplex p (standard_simplex q) f
      using c that by (auto simp: simplicial_chain_def)
    then obtain m where feq: f = oriented_simplex p m
      by (auto simp: simplicial_simplex)
    have 0: simplicial_simplex p (standard_simplex q) (oriented_simplex p m)
      using ⟨simplicial_simplex p (standard_simplex q) f⟩ feq by blast
    then have 1: simplicial_simplex (Suc p) (standard_simplex q)
      (oriented_simplex (Suc p) (λi. if i = 0 then v else m (i - 1)))
      using convex_standard_simplex v
      by (simp flip: simplex_cone add: simplicial_simplex_simplex_cone)
    show ?thesis
      using simplex_map_oriented_simplex [OF 1 g ⟨q ≤ r⟩]
        simplex_map_oriented_simplex [of p q m r S g, OF 0 g ⟨q ≤ r⟩]
      by (simp add: feq oriented_simplex_eq simplex_cone)
  qed
qed
show ?thesis
  by (auto simp: chain_map_def simplicial_cone_def frag_extend_compose *
    intro: frag_extend_eq)
qed

```

0.1.15 Barycentric subdivision of a linear ("simplicial") simplex's image

definition *simplicial_vertex*

where *simplicial_vertex* i $f = f(\lambda j. \text{if } j = i \text{ then } 1 \text{ else } 0)$

lemma *simplicial_vertex_oriented_simplex*:

simplicial_vertex i (oriented_simplex p l) = (if $i \leq p$ then l i else undefined)

by (simp add: simplicial_vertex_def oriented_simplex_def if_distrib cong: if_cong)

primrec *simplicial_subdivision*

where

```

simplicial_subdivision 0 = id
| simplicial_subdivision (Suc p) =
  frag_extend
  (λf. simplicial_cone p
    (λi. (∑ j≤Suc p. simplicial_vertex j f i) / (p + 2))
    (simplicial_subdivision p (chain_boundary (Suc p) (frag_of f))))

```

lemma *simplicial_subdivision_0 [simp]:*

```

simplicial_subdivision p 0 = 0
by (induction p) auto

```

lemma *simplicial_subdivision_diff:*

```

simplicial_subdivision p (c1 - c2) = simplicial_subdivision p c1 - simplicial_subdivision
p c2
by (induction p) (auto simp: frag_extend_diff)

```

lemma *simplicial_subdivision_of:*

```

simplicial_subdivision p (frag_of f) =
  (if p = 0 then frag_of f
   else simplicial_cone (p - 1)
    (λi. (∑ j≤p. simplicial_vertex j f i) / (Suc p))
    (simplicial_subdivision (p - 1) (chain_boundary p (frag_of f))))
by (induction p) (auto simp: add commute)

```

lemma *simplicial_chain_simplicial_subdivision:*

```

simplicial_chain p S c
  ⇒ simplicial_chain p S (simplicial_subdivision p c)

```

proof (induction p arbitrary: S c)

case (Suc p)

show ?case

using Suc.premis [unfolded simplicial_chain_def]

proof (induction c rule: frag_induction)

case (one f)

then have *f: simplicial_simplex (Suc p) S f*

by auto

then have *simplicial_chain p (f ‘ standard_simplex (Suc p))*

(simplicial_subdivision p (chain_boundary (Suc p) (frag_of f)))

by (metis Suc.IH diff_Suc_1 simplicial_chain_boundary simplicial_chain_of
simplicial_simplex subsetI)

moreover

obtain *l where l: ∧x. x ∈ standard_simplex (Suc p) ⇒ (λi. (∑ j≤Suc p. l
j i * x j)) ∈ S*

and *freq: f = oriented_simplex (Suc p) l*

using *f* **by** (fastforce simp: simplicial_simplex oriented_simplex_def simp
del: sum.atMost_Suc)

have *(λi. (1 - u) * ((∑ j≤Suc p. simplicial_vertex j f i) / (real p + 2)) + u*

```

* y i) ∈ S
  if 0 ≤ u u ≤ 1 and y: y ∈ f ‘ standard_simplex (Suc p) for y u
  proof -
    obtain x where x: x ∈ standard_simplex (Suc p) and yeq: y = ori-
      ented_simplex (Suc p) l x
    using y feq by blast
    have (λi. ∑ j≤Suc p. l j i * ((if j ≤ Suc p then (1 - u) * inverse (p + 2)
+ u * x j else 0))) ∈ S
    proof (rule l)
      have inverse (2 + real p) ≤ 1 (2 + real p) * ((1 - u) * inverse (2 + real
p)) + u = 1
      by (auto simp add: field_split_simps)
      then show (λj. if j ≤ Suc p then (1 - u) * inverse (real (p + 2)) + u * x
j else 0) ∈ standard_simplex (Suc p)
      using x ⟨0 ≤ u⟩ ⟨u ≤ 1⟩
      by (simp add: sum.distrib standard_simplex_def linepath_le_1 flip:
sum_distrib_left del: sum.atMost_Suc)
    qed
    moreover have (λi. ∑ j≤Suc p. l j i * ((1 - u) * inverse (2 + real p) + u
* x j))
      = (λi. (1 - u) * (∑ j≤Suc p. l j i) / (real p + 2) + u * (∑ j≤Suc
p. l j i * x j))
    proof
      fix i
      have (∑ j≤Suc p. l j i * ((1 - u) * inverse (2 + real p) + u * x j))
        = (∑ j≤Suc p. (1 - u) * l j i / (real p + 2) + u * l j i * x j) (is ?lhs
= _)
      by (simp add: field_simps cong: sum.cong)
      also have ... = (1 - u) * (∑ j≤Suc p. l j i) / (real p + 2) + u * (∑ j≤Suc
p. l j i * x j) (is _ = ?rhs)
      by (simp add: sum_distrib_left sum.distrib sum_divide_distrib mult.assoc
del: sum.atMost_Suc)
      finally show ?lhs = ?rhs .
    qed
    ultimately show ?thesis
      using feq x yeq
      by (simp add: simplicial_vertex_oriented_simplex) (simp add: oriented_simplex_def)
    qed
    ultimately show ?case
      by (simp add: simplicial_chain_simplicial_cone)
  next
    case (diff a b)
    then show ?case
      by (metis simplicial_chain_diff simplicial_subdivision_diff)
    qed auto
  qed auto

lemma chain_boundary_simplicial_subdivision:
  simplicial_chain p S c

```

```

     $\Rightarrow$  chain_boundary p (simplicial_subdivision p c) = simplicial_subdivision (p
-1) (chain_boundary p c)
proof (induction p arbitrary: c)
  case (Suc p)
  show ?case
    using Suc.premis [unfolded simplicial_chain_def]
  proof (induction c rule: frag_induction)
    case (one f)
    then have f: simplicial_simplex (Suc p) S f
    by simp
    then have simplicial_chain p S (simplicial_subdivision p (chain_boundary
(Suc p) (frag_of f)))
    by (metis diff_Suc_1 simplicial_chain_boundary simplicial_chain_of sim-
plicial_chain_simplicial_subdivision)
    moreover have simplicial_chain p S (chain_boundary (Suc p) (frag_of f))
    using one simplicial_chain_boundary simplicial_chain_of by fastforce
    moreover have simplicial_subdivision (p - Suc 0) (chain_boundary p (chain_boundary
(Suc p) (frag_of f))) = 0
    by (metis f chain_boundary_boundary_alt simplicial_simplex_def simpli-
cial_subdivision_0 singular_chain_of)
    ultimately show ?case
    using chain_boundary_simplicial_cone Suc
    by (auto simp: chain_boundary_of_frag_extend_diff simplicial_cone_def)
  next
  case (diff a b)
  then show ?case
  by (simp add: simplicial_subdivision_diff chain_boundary_diff frag_extend_diff)
qed auto
qed auto

```

A MESS AND USED ONLY ONCE

lemma simplicial_subdivision_shrinks:

```

 $\llbracket$ simplicial_chain p S c;
 $\bigwedge f x y. \llbracket f \in \text{Poly\_Mapping.keys } c; x \in \text{standard\_simplex } p; y \in \text{stan-}$ 
 $\text{dard\_simplex } p \rrbracket \Rightarrow |f x k - f y k| \leq d;$ 
 $f \in \text{Poly\_Mapping.keys}(\text{simplicial\_subdivision } p c);$ 
 $x \in \text{standard\_simplex } p; y \in \text{standard\_simplex } p \rrbracket$ 
 $\Rightarrow |f x k - f y k| \leq (p / (\text{Suc } p)) * d$ 
proof (induction p arbitrary: d c f x y)
  case (Suc p)
  define Sigg where Sigg  $\equiv \lambda f::(\text{nat} \Rightarrow \text{real}) \Rightarrow \text{nat} \Rightarrow \text{real}. \lambda i. (\sum j \leq \text{Suc } p.$ 
 $\text{simplicial\_vertex } j f i) / \text{real } (p + 2)$ 
  define CB where CB  $\equiv \lambda f::(\text{nat} \Rightarrow \text{real}) \Rightarrow \text{nat} \Rightarrow \text{real}. \text{chain\_boundary } (\text{Suc}$ 
 $p) (\text{frag\_of } f)$ 
  have *: Poly_Mapping.keys
    (simplicial_cone p (Sigg f))
    (simplicial_subdivision p (CB f)))
     $\subseteq \{f. \forall x \in \text{standard\_simplex } (\text{Suc } p). \forall y \in \text{standard\_simplex } (\text{Suc } p).$ 
 $|f x k - f y k| \leq \text{Suc } p / (\text{real } p + 2) * d\}$  (is ?lhs  $\subseteq$  ?rhs)

```

```

    if f: f ∈ Poly_Mapping.keys c for f
  proof -
    have ssf: simplicial_simplex (Suc p) S f
      using Suc.premis(1) simplicial_chain_def that by auto
    have 2:  $\bigwedge x y. \llbracket x \in \text{standard\_simplex (Suc p)}; y \in \text{standard\_simplex (Suc p)} \rrbracket$ 
 $\implies |f x k - f y k| \leq d$ 
    by (meson Suc.premis(2) f subsetD le_Suc_eq order_refl standard_simplex_mono)
    have sub: Poly_Mapping.keys ((frag_of ∘ simplex_cone p (Sipp f)) g) ⊆ ?rhs
      if g ∈ Poly_Mapping.keys (simplicial_subdivision p (CB f)) for g
    proof -
      have 1: simplicial_chain p S (CB f)
        unfolding CB_def
        using ssf simplicial_chain_boundary simplicial_chain_of by fastforce
      have simplicial_chain (Suc p) (f ' standard_simplex(Suc p)) (frag_of f)
        by (metis simplicial_chain_of simplicial_simplex ssf subset_refl)
      then have sc_sub: Poly_Mapping.keys (CB f)
        ⊆ Collect (simplicial_simplex p (f ' standard_simplex (Suc p)))
        by (metis diff_Suc_1 simplicial_chain_boundary simplicial_chain_def
          CB_def)
      have led:  $\bigwedge h x y. \llbracket h \in \text{Poly\_Mapping.keys (CB f)}; x \in \text{standard\_simplex p}; y \in \text{standard\_simplex p} \rrbracket \implies |h x k$ 
 $- h y k| \leq d$ 
        using Suc.premis(2) f sc_sub
        by (simp add: simplicial_simplex_subset_iff image_iff) metis
      have  $\bigwedge f' x y. \llbracket f' \in \text{Poly\_Mapping.keys (simplicial\_subdivision p (CB f))}; x \in \text{standard\_simplex p}; y \in \text{standard\_simplex p} \rrbracket$ 
 $\implies |f' x k - f' y k| \leq (p / (\text{Suc p})) * d$ 
        by (blast intro: led Suc.IH [of CB f, OF 1])
      then have g:  $\bigwedge x y. \llbracket x \in \text{standard\_simplex p}; y \in \text{standard\_simplex p} \rrbracket \implies$ 
 $|g x k - g y k| \leq (p / (\text{Suc p})) * d$ 
        using that by blast
      have d ≥ 0
        using Suc.premis(2)[OF f] ⟨x ∈ standard_simplex (Suc p)⟩ by force
      have 3: simplex_cone p (Sipp f) g ∈ ?rhs
    proof -
      have simplicial_simplex p (f ' standard_simplex(Suc p)) g
        by (metis (mono_tags, opaque_lifting) sc_sub mem_Collect_eq simplicial_chain_def simplicial_chain_simplicial_subdivision subsetD that)
      then obtain m where m: g ' standard_simplex p ⊆ f ' standard_simplex (Suc p)
        and geq: g = oriented_simplex p m
        using ssf by (auto simp: simplicial_simplex)
      have m_in_gim: m i ∈ g ' standard_simplex p if i ≤ p for i
    proof
      show m i = g (λj. if j = i then 1 else 0)
        by (simp add: geq oriented_simplex_def that if_distrib cong: if_cong)
      show (λj. if j = i then 1 else 0) ∈ standard_simplex p
        by (simp add: oriented_simplex_def that)
    qed
  qed

```

```

obtain  $l$  where  $l: f \text{ ' standard\_simplex } (Suc\ p) \subseteq S$ 
and  $feq: f = oriented\_simplex\ (Suc\ p)\ l$ 
using  $ssf$  by  $(auto\ simp: simplicial\_simplex)$ 
show  $?thesis$ 
proof  $(clarsimp\ simp\ add: geq\ simp\ del: sum.atMost\_Suc)$ 
fix  $x\ y$ 
assume  $x: x \in standard\_simplex\ (Suc\ p)$  and  $y: y \in standard\_simplex$ 
 $(Suc\ p)$ 
then have  $x': (\forall i. 0 \leq x\ i \wedge x\ i \leq 1) \wedge (\forall i > Suc\ p. x\ i = 0) \wedge (\sum i \leq Suc$ 
 $p. x\ i) = 1$ 
and  $y': (\forall i. 0 \leq y\ i \wedge y\ i \leq 1) \wedge (\forall i > Suc\ p. y\ i = 0) \wedge (\sum i \leq Suc\ p.$ 
 $y\ i) = 1$ 
by  $(auto\ simp: standard\_simplex\_def)$ 
have  $|(\sum j \leq Suc\ p. (if\ j = 0\ then\ \lambda i. (\sum j \leq Suc\ p. l\ j\ i) / (2 + real\ p)\ else$ 
 $m\ (j - 1))\ k * x\ j) -$ 
 $(\sum j \leq Suc\ p. (if\ j = 0\ then\ \lambda i. (\sum j \leq Suc\ p. l\ j\ i) / (2 + real\ p)\ else$ 
 $m\ (j - 1))\ k * y\ j)|$ 
 $\leq (1 + real\ p) * d / (2 + real\ p)$ 
proof  $-$ 
have  $zero: |m\ (s - Suc\ 0)\ k - (\sum j \leq Suc\ p. l\ j\ k) / (2 + real\ p)| \leq (1$ 
 $+ real\ p) * d / (2 + real\ p)$ 
if  $0 < s$  and  $s \leq Suc\ p$  for  $s$ 
proof  $-$ 
have  $m\ (s - Suc\ 0) \in f \text{ ' standard\_simplex } (Suc\ p)$ 
using  $m\ m\_in\_gim\ that(2)$  by  $auto$ 
then obtain  $z$  where  $eq: m\ (s - Suc\ 0) = (\lambda i. \sum j \leq Suc\ p. l\ j\ i * z$ 
 $j)$  and  $z: z \in standard\_simplex\ (Suc\ p)$ 
using  $feq\ unfolding\ oriented\_simplex\_def$  by  $auto$ 
show  $?thesis$ 
unfolding  $eq$ 
proof  $(rule\ convex\_sum\_bound\_le)$ 
fix  $i$ 
assume  $i: i \in \{..Suc\ p\}$ 
then have  $[simp]: card\ (\{..Suc\ p\} - \{i\}) = Suc\ p$ 
by  $(simp\ add: card\_Suc\_Diff1)$ 
have  $(\sum j \leq Suc\ p. |l\ i\ k / (p + 2) - l\ j\ k / (p + 2)|) = (\sum j \leq Suc\ p.$ 
 $|l\ i\ k - l\ j\ k| / (p + 2))$ 
by  $(rule\ sum.cong)\ (simp\_all\ add: flip: diff\_divide\_distrib)$ 
also have  $\dots = (\sum j \in \{..Suc\ p\} - \{i\}. |l\ i\ k - l\ j\ k| / (p + 2))$ 
by  $(rule\ sum.mono\_neutral\_right)\ auto$ 
also have  $\dots \leq (1 + real\ p) * d / (p + 2)$ 
proof  $(rule\ sum\_bounded\_above\_divide)$ 
fix  $i' :: nat$ 
assume  $i': i' \in \{..Suc\ p\} - \{i\}$ 
have  $lf: l\ r \in f \text{ ' standard\_simplex } (Suc\ p)$  if  $r \leq Suc\ p$  for  $r$ 
proof
show  $l\ r = f\ (\lambda j. if\ j = r\ then\ 1\ else\ 0)$ 
using  $that$  by  $(simp\ add: feq\ oriented\_simplex\_def\ if\_distrib$ 
 $cong: if\_cong)$ 

```



```

      show (λj. if j = r then 1 else 0) ∈ standard_simplex (Suc p)
      by (auto simp: oriented_simplex_def that)
    qed
    show |l i k - l i' k| / real (p + 2) ≤ (1 + real p) * d / real (p +
2) / real (card ({..Suc p} - {i}))
      using i i' lf [of i] lf [of i'] 2
      by (auto simp: image_iff divide_simps)
    qed auto
    finally have (∑ j ≤ Suc p. |l i k / (p + 2) - l j k / (p + 2)|) ≤ (1
+ real p) * d / (p + 2) .
      then have |∑ j ≤ Suc p. l i k / (p + 2) - l j k / (p + 2)| ≤ (1 +
real p) * d / (p + 2)
        by (rule order_trans [OF sum_abs])
      then show |l i k - (∑ j ≤ Suc p. l j k) / (2 + real p)| ≤ (1 + real p)
* d / (2 + real p)
        by (simp add: sum_subtractf sum_divide_distrib del: sum.atMost_Suc)
      qed (use standard_simplex_def z in auto)
    qed
    have nonz: |m (s - Suc 0) k - m (r - Suc 0) k| ≤ (1 + real p) * d /
(2 + real p) (is ?lhs ≤ ?rhs)
      if r < s and 0 < r and r ≤ Suc p and s ≤ Suc p for r s
    proof -
      have ?lhs ≤ (p / (Suc p)) * d
        using m_in_gim [of r - Suc 0] m_in_gim [of s - Suc 0] that g by
fastforce
      also have ... ≤ ?rhs
        by (simp add: field_simps ‹0 ≤ d›)
      finally show ?thesis .
    qed
    have jj: j ≤ Suc p ∧ j' ≤ Suc p
      → |(if j' = 0 then λi. (∑ j ≤ Suc p. l j i) / (2 + real p) else m (j'
- 1)) k -
      (if j = 0 then λi. (∑ j ≤ Suc p. l j i) / (2 + real p) else m (j - 1))
k|
      ≤ (1 + real p) * d / (2 + real p) for j j'
    using ‹0 ≤ d›
      by (rule_tac a=j and b = j' in linorder_less_wlog; force simp: zero
nonz simp del: sum.atMost_Suc)
    show ?thesis
      apply (rule convex_sum_bound_le)
      using x' apply blast
      using x' apply blast
      apply (subst abs_minus_commute)
      apply (rule convex_sum_bound_le)
      using y' apply blast
      using y' apply blast
      using jj by blast
    qed
    then show |simplex_cone p (Sigm f) (oriented_simplex p m) x k -

```

```

simplicial_cone p (Simp f) (oriented_simplex p m) y k|
  ≤ (1 + real p) * d / (real p + 2)
  apply (simp add: feq Simp_def simplicial_vertex_oriented_simplex
simplicial_cone del: sum.atMost_Suc)
  apply (simp add: oriented_simplex_def algebra_simps x y del: sum.atMost_Suc)
  done
qed
qed
show ?thesis
  using Suc.IH [OF 1, where f=g] 2 3 by simp
qed
then show ?thesis
  unfolding simplicial_chain_def simplicial_cone_def
  by (simp add: order_trans [OF keys_frag_extend] sub UN_subset_iff)
qed
obtain ff where ff ∈ Poly_Mapping.keys c
  f ∈ Poly_Mapping.keys
  (simplicial_cone p
    (λi. (∑ j ≤ Suc p. simplicial_vertex j ff i) /
      (real p + 2))
    (simplicial_subdivision p (CB ff)))
  using Suc.prem3 subsetD [OF keys_frag_extend]
  by (force simp: CB_def simp del: sum.atMost_Suc)
then show ?case
  using Suc * by (simp add: add.commute Simp_def subset_iff)
qed (auto simp: standard_simplex_0)

```

0.1.16 Singular subdivision

definition *singular_subdivision*

```

where singular_subdivision p ≡
  frag_extend
  (λf. chain_map p f
    (simplicial_subdivision p
      (frag_of(restrict id (standard_simplex p)))))

```

lemma *singular_subdivision_0* [simp]: *singular_subdivision p 0 = 0*

by (simp add: singular_subdivision_def)

lemma *singular_subdivision_add*:

```

singular_subdivision p (a + b) = singular_subdivision p a + singular_subdivision
p b
by (simp add: singular_subdivision_def frag_extend_add)

```

lemma *singular_subdivision_diff*:

```

singular_subdivision p (a - b) = singular_subdivision p a - singular_subdivision
p b
by (simp add: singular_subdivision_def frag_extend_diff)

```

```

lemma simplicial_simplex_id [simp]:
  simplicial_simplex p S (restrict id (standard_simplex p))  $\longleftrightarrow$  standard_simplex
p  $\subseteq$  S
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
  by (simp add: simplicial_simplex_def singular_simplex_def continuous_map_in_subtopology
set_mp)
next
  assume R: ?rhs
  then have cm: continuous_map
    (subtopology (powertop_real UNIV) (standard_simplex p))
    (subtopology (powertop_real UNIV) S) id
  using continuous_map_from_subtopology_mono continuous_map_id by blast
  moreover have  $\exists l.$  restrict id (standard_simplex p) = oriented_simplex p l
  proof
    show restrict id (standard_simplex p) = oriented_simplex p ( $\lambda i j.$  if i = j then
1 else 0)
    by (force simp: oriented_simplex_def standard_simplex_def if_distrib [of  $\lambda u.$ 
u *  $\_$ ] cong: if_cong)
  qed
  ultimately show ?lhs
  by (simp add: simplicial_simplex_def singular_simplex_def)
qed

```

```

lemma singular_chain_singular_subdivision:
  assumes singular_chain p X c
  shows singular_chain p X (singular_subdivision p c)
  unfolding singular_subdivision_def
proof (rule singular_chain_extend)
  fix ca
  assume ca  $\in$  Poly_Mapping.keys c
  with assms have singular_simplex p X ca
  by (simp add: singular_chain_def subset_iff)
  then show singular_chain p X (chain_map p ca (simplicial_subdivision p
(frag_of (restrict id (standard_simplex p))))))
  unfolding singular_simplex_def
  by (metis order_refl simplicial_chain_of_simplicial_chain_simplicial_subdivision
simplicial_imp_singular_chain simplicial_simplex_id singular_chain_chain_map)
qed

```

```

lemma naturality_singular_subdivision:
  singular_chain p X c
   $\implies$  singular_subdivision p (chain_map p g c) = chain_map p g (singular_subdivision
p c)
  unfolding singular_chain_def
proof (induction rule: frag_induction)
  case (one f)

```

```

then have singular_simplex p X f
  by auto
have  $\llbracket \text{simplicial\_chain } p \text{ (standard\_simplex } p) \text{ } d \rrbracket$ 
   $\implies \text{chain\_map } p \text{ (simplex\_map } p \text{ } g \text{ } f) \text{ } d = \text{chain\_map } p \text{ } g \text{ (chain\_map } p \text{ } f \text{ } d)$ 
for d
  unfolding simplicial_chain_def
proof (induction rule: frag_induction)
  case (one x)
  then have simplex_map p (simplex_map p g f) x = simplex_map p g (simplex_map
p f x)
    by (force simp: simplex_map_def restrict_compose_left simplicial_simplex)
  then show ?case
    by auto
qed (auto simp: chain_map_diff)
then show ?case
  using simplicial_chain_simplicial_subdivision [of p standard_simplex p frag_of
(restrict id (standard_simplex p))]
    by (simp add: singular_subdivision_def)
next
  case (diff a b)
  then show ?case
    by (simp add: chain_map_diff singular_subdivision_diff)
qed auto

lemma simplicial_chain_chain_map:
  assumes f: simplicial_simplex q X f and c: simplicial_chain p (standard_simplex
q) c
  shows simplicial_chain p X (chain_map p f c)
  using c unfolding simplicial_chain_def
proof (induction c rule: frag_induction)
  case (one g)
  have  $\exists n. \text{simplex\_map } p \text{ (oriented\_simplex } q \text{ } l)$ 
    (oriented_simplex p m) = oriented_simplex p n
  if m: singular_simplex p
    (subtopology (powertop_real UNIV) (standard_simplex q)) (oriented_simplex
p m)
  for l m
  proof –
  have ( $\lambda i. \sum_{j \leq p. m \text{ } j \text{ } i * x \text{ } j$ )  $\in \text{standard\_simplex } q$ 
    if x  $\in \text{standard\_simplex } p$  for x
    using that m unfolding oriented_simplex_def singular_simplex_def
    by (auto simp: continuous_map_in_subtopology image_subset_iff)
  then show ?thesis
    unfolding oriented_simplex_def simplex_map_def
    apply (rule_tac x =  $\lambda j \text{ } k. (\sum_{i \leq q. l \text{ } i \text{ } k * m \text{ } j \text{ } i)$  in exI)
    apply (force simp: sum_distrib_left sum_distrib_right mult.assoc intro:
sum.swap)
    done
  qed

```

```

then show ?case
  using f one
  apply (simp add: simplicial_simplex_def)
  using singular_simplex_def singular_simplex_simplex_map by blast
next
  case (diff a b)
  then show ?case
    by (metis chain_map_diff simplicial_chain_def simplicial_chain_diff)
qed auto

lemma singular_subdivision_simplicial_simplex:
  simplicial_chain p S c
     $\implies$  singular_subdivision p c = simplicial_subdivision p c
proof (induction p arbitrary: S c)
  case 0
  then show ?case
    unfolding simplicial_chain_def
  proof (induction rule: frag_induction)
    case (one x)
    then show ?case
      using singular_simplex_chain_map_id simplicial_imp_singular_simplex
      by (fastforce simp: singular_subdivision_def simplicial_subdivision_def)
    qed (auto simp: singular_subdivision_diff)
  next
  case (Suc p)
  show ?case
    using Suc.premis unfolding simplicial_chain_def
  proof (induction rule: frag_induction)
    case (one f)
    then have ssf: simplicial_simplex (Suc p) S f
      by (auto simp: simplicial_simplex)
    then have 1: simplicial_chain p (standard_simplex (Suc p))
      (simplicial_subdivision p
        (chain_boundary (Suc p)
          (frag_of (restrict id (standard_simplex (Suc p))))))
      by (metis diff_Suc_1 order_refl simplicial_chain_boundary simplicial_chain_of
        simplicial_chain_simplicial_subdivision simplicial_simplex_id)
    have 2:  $(\lambda i. (\sum j \leq \text{Suc } p. \text{simplicial\_vertex } j \text{ (restrict id (standard\_simplex (Suc } p)) ) } i) / (\text{real } p + 2))$ 
       $\in$  standard_simplex (Suc p)
      by (simp add: simplicial_vertex_def standard_simplex_def del: sum.atMost_Suc)
    have ss_Sp:  $(\lambda i. (\text{if } i \leq \text{Suc } p \text{ then } 1 \text{ else } 0) / (\text{real } p + 2)) \in \text{standard\_simplex (Suc } p)$ 
      by (simp add: standard_simplex_def field_split_simps)
    obtain l where feq: f = oriented_simplex (Suc p) l
      using one unfolding simplicial_simplex by blast
    then have 3:  $f (\lambda i. (\sum j \leq \text{Suc } p. \text{simplicial\_vertex } j \text{ (restrict id (standard\_simplex (Suc } p)) ) } i) / (\text{real } p + 2))$ 

```

```

      = (λi. (∑ j ≤ Suc p. simplicial_vertex j f i) / (real p + 2))
    unfolding simplicial_vertex_def oriented_simplex_def
      by (simp add: ss_Sp if_distrib [of λx. _ * x] sum_divide_distrib del:
sum.atMost_Suc cong: if_cong)
    have scp: singular_chain (Suc p)
      (subtopology (powertop_real UNIV) (standard_simplex (Suc p)))
      (frag_of (restrict id (standard_simplex (Suc p))))
    by (simp add: simplicial_imp_singular_chain)
    have scps: simplicial_chain p (standard_simplex (Suc p))
      (chain_boundary (Suc p) (frag_of (restrict id (standard_simplex
(Suc p)))))
    by (metis diff_Suc_1 order_refl simplicial_chain_boundary simplicial_chain_of
simplicial_simplex_id)
    have scpf: simplicial_chain p S
      (chain_map p f
      (chain_boundary (Suc p) (frag_of (restrict id (standard_simplex
(Suc p)))))
    using scps simplicial_chain_chain_map ssf by blast
    have 4: chain_map p f
      (simplicial_subdivision p
      (chain_boundary (Suc p) (frag_of (restrict id (standard_simplex
(Suc p)))))
    = simplicial_subdivision p (chain_boundary (Suc p) (frag_of f))
  proof -
    have singular_simplex (Suc p) (subtopology (powertop_real UNIV) S) f
      using simplicial_simplex_def ssf by blast
    then have chain_map (Suc p) f (frag_of (restrict id (standard_simplex
(Suc p))))) = frag_of f
      using singular_simplex_chain_map_id by blast
    then show ?thesis
      by (metis (no_types) Suc.IH chain_boundary_chain_map diff_Suc_Suc
diff_zero
      naturality_singular_subdivision scp scpf scps simplicial_imp_singular_chain)
  qed
  show ?case
    apply (simp add: singular_subdivision_def del: sum.atMost_Suc)
    apply (simp only: ssf 1 2 3 4 chain_map_simplicial_cone [of Suc p S _ p
Suc p])
    done
  qed (auto simp: frag_extend_diff singular_subdivision_diff)
qed

```

lemma *naturality_singular_subdivision*:

```

  [[simplicial_chain p (standard_simplex q) c; simplicial_simplex q S g]]
  ⇒ simplicial_subdivision p (chain_map p g c) = chain_map p g (simplicial_subdivision
p c)
  by (metis naturality_singular_subdivision simplicial_chain_chain_map simpli-
cial_imp_singular_chain)

```

singular_subdivision_simplicial_simplex)

lemma *chain_boundary_singular_subdivision*:

```

  singular_chain p X c
     $\implies$  chain_boundary p (singular_subdivision p c) =
      singular_subdivision (p - Suc 0) (chain_boundary p c)
  unfolding singular_chain_def
  proof (induction rule: frag_induction)
    case (one f)
    then have ssf: singular_simplex p X f
      by (auto simp: singular_simplex_def)
    then have scp: simplicial_chain p (standard_simplex p) (frag_of (restrict id
      (standard_simplex p)))
      by simp
    have scp1: simplicial_chain (p - Suc 0) (standard_simplex p)
      (chain_boundary p (frag_of (restrict id (standard_simplex p))))
      using simplicial_chain_boundary by force
    have sgp1: singular_chain (p - Suc 0)
      (subtopology (powertop_real UNIV) (standard_simplex p))
      (chain_boundary p (frag_of (restrict id (standard_simplex p))))
      using scp1 simplicial_imp_singular_chain by blast
    have scpp: singular_chain p (subtopology (powertop_real UNIV) (standard_simplex
      p))
      (frag_of (restrict id (standard_simplex p)))
      using scp simplicial_imp_singular_chain by blast
    then show ?case
      unfolding singular_subdivision_def
      using chain_boundary_chain_map [of p subtopology (powertop_real UNIV)
        (standard_simplex p) _ f]
      apply (simp add: simplicial_chain_simplicial_subdivision
        simplicial_imp_singular_chain chain_boundary_simplicial_subdivision
        [OF scp]
        flip: singular_subdivision_simplicial_simplex [OF scp1] naturality_singular_subdivision
        [OF sgp1])
      by (metis (full_types) singular_subdivision_def chain_boundary_chain_map
        [OF scpp] singular_simplex_chain_map_id [OF ssf])
    qed (auto simp: singular_subdivision_def frag_extend_diff chain_boundary_diff)
  qed

```

lemma *singular_subdivision_zero*:

```

  singular_chain 0 X c  $\implies$  singular_subdivision 0 c = c
  unfolding singular_chain_def
  proof (induction rule: frag_induction)
    case (one f)
    then have restrict (f  $\circ$  restrict id (standard_simplex 0)) (standard_simplex 0)
      = f
      by (simp add: extensional_restrict restrict_compose_right singular_simplex_def)
    then show ?case
      by (auto simp: singular_subdivision_def simplex_map_def)
    qed (auto simp: singular_subdivision_def frag_extend_diff)
  qed

```

primrec *subd* **where**

$subd\ 0 = (\lambda x. 0)$
 $| subd\ (Suc\ p) =$
 $\quad frag_extend$
 $\quad (\lambda f. simplicial_cone\ (Suc\ p)\ (\lambda i. (\sum j \leq Suc\ p. simplicial_vertex\ j\ f\ i) / real$
 $\quad (Suc\ p + 1)))$
 $\quad (simplicial_subdivision\ (Suc\ p)\ (frag_of\ f) - frag_of\ f -$
 $\quad subd\ p\ (chain_boundary\ (Suc\ p)\ (frag_of\ f))))$

lemma *subd_0* [*simp*]: $subd\ p\ 0 = 0$
by (*induction p auto*)

lemma *subd_diff* [*simp*]: $subd\ p\ (c1 - c2) = subd\ p\ c1 - subd\ p\ c2$
by (*induction p (auto simp: frag_extend_diff)*)

lemma *subd_uminus* [*simp*]: $subd\ p\ (-c) = - subd\ p\ c$
by (*metis diff_0 subd_0 subd_diff*)

lemma *subd_power_uminus*: $subd\ p\ (frag_cmul\ ((-1) ^ k)\ c) = frag_cmul\ ((-1) ^ k)\ (subd\ p\ c)$

proof (*induction k*)

case 0

then show ?*case* **by** *simp*

next

case (*Suc k*)

then show ?*case*

by (*metis frag_cmul_cmul frag_cmul_minus_one power_Suc subd_uminus*)

qed

lemma *subd_power_sum*: $subd\ p\ (sum\ f\ I) = sum\ (subd\ p \circ f)\ I$

proof (*induction I rule: infinite_finite_induct*)

case (*insert i I*)

then show ?*case*

by (*metis (no_types, lifting) comp_apply diff_minus_eq_add subd_diff subd_uminus sum.insert*)

qed *auto*

lemma *subd*: $simplicial_chain\ p\ (standard_simplex\ s)\ c$

$\implies (\forall r\ g. simplicial_simplex\ s\ (standard_simplex\ r)\ g \longrightarrow chain_map\ (Suc\ p)\ g\ (subd\ p\ c) = subd\ p\ (chain_map\ p\ g\ c))$

$\wedge simplicial_chain\ (Suc\ p)\ (standard_simplex\ s)\ (subd\ p\ c)$

$\wedge (chain_boundary\ (Suc\ p)\ (subd\ p\ c)) + (subd\ (p - Suc\ 0)\ (chain_boundary\ p\ c)) = (simplicial_subdivision\ p\ c) - c$

proof (*induction p arbitrary: c*)

case (*Suc p*)

show ?*case*

using *Suc.premis* [*unfolded simplicial_chain_def*]


```

proof (induction rule: frag_induction)
  case (one f)
  then obtain l where l: ( $\lambda x i. \sum j \leq \text{Suc } p. l j i * x j$ ) ‘ standard_simplex (Suc
p)  $\subseteq$  standard_simplex s
    and feq: f = oriented_simplex (Suc p) l
  by (metis (mono_tags) mem_Collect_eq simplicial_simplex simplicial_simplex_oriented_simplex)
  have scf: simplicial_chain (Suc p) (standard_simplex s) (frag_of f)
    using one by simp
  have lss: l i  $\in$  standard_simplex s if i  $\leq$  Suc p for i
  proof –
    have ( $\lambda i'. \sum j \leq \text{Suc } p. l j i' * (\text{if } j = i \text{ then } 1 \text{ else } 0)$ )  $\in$  standard_simplex s
      using subsetD [OF l] basis_in_standard_simplex that by blast
    moreover have ( $\lambda i'. \sum j \leq \text{Suc } p. l j i' * (\text{if } j = i \text{ then } 1 \text{ else } 0)$ ) = l i
      using that by (simp add: if_distrib [of  $\lambda x. \_ * x$ ] del: sum.atMost_Suc
cong: if_cong)
    ultimately show ?thesis
      by simp
  qed
  have *: ( $\bigwedge i. i \leq n \implies l i \in \text{standard\_simplex } s$ )
     $\implies (\lambda i. (\sum j \leq n. l j i) / (\text{Suc } n)) \in \text{standard\_simplex } s$  for n
  proof (induction n)
    case (Suc n)
    let ?x =  $\lambda i. (1 - \text{inverse } (n + 2)) * ((\sum j \leq n. l j i) / (\text{Suc } n)) + \text{inverse } (n$ 
+ 2) * l (Suc n) i
    have ?x  $\in$  standard_simplex s
    proof (rule convex_standard_simplex)
      show ( $\lambda i. (\sum j \leq n. l j i) / \text{real } (\text{Suc } n)$ )  $\in$  standard_simplex s
        using Suc by simp
    qed (auto simp: lss Suc inverse_le_1_iff)
    moreover have ?x = ( $\lambda i. (\sum j \leq \text{Suc } n. l j i) / \text{real } (\text{Suc } (\text{Suc } n))$ )
      by (force simp: divide_simps)
    ultimately show ?case
      by simp
  qed auto
  have **: ( $\lambda i. (\sum j \leq \text{Suc } p. \text{simplicial\_vertex } j f i) / (2 + \text{real } p)$ )  $\in$  stan-
dard_simplex s
    using * [of Suc p] lss by (simp add: simplicial_vertex_oriented_simplex feq)
  show ?case
  proof (intro conjI impI allI)
    fix r g
    assume g: simplicial_simplex s (standard_simplex r) g
    then obtain m where geg: g = oriented_simplex s m
      using simplicial_simplex by blast
    have 1: simplicial_chain (Suc p) (standard_simplex s) (simplicial_subdivision
(Suc p) (frag_of f))
      by (metis mem_Collect_eq one.hyps simplicial_chain_of simplicial_chain_simplicial_subdivision)
    have 2: ( $\sum j \leq \text{Suc } p. \sum i \leq s. m i k * \text{simplicial\_vertex } j f i$ )
      = ( $\sum j \leq \text{Suc } p. \text{simplicial\_vertex } j$ 
        (simplex_map (Suc p) (oriented_simplex s m) f) k) for k

```

```

proof (rule sum.cong [OF refl])
  fix j
  assume j: j ∈ {..Suc p}
  have eq: simplex_map (Suc p) (oriented_simplex s m) (oriented_simplex
(Suc p) l)
    = oriented_simplex (Suc p) (oriented_simplex s m ∘ l)
  proof (rule simplex_map_oriented_simplex)
    show simplicial_simplex (Suc p) (standard_simplex s) (oriented_simplex
(Suc p) l)
      using one by (simp add: feq flip: oriented_simplex_def)
      show simplicial_simplex s (standard_simplex r) (oriented_simplex s m)
        using g by (simp add: geq)
    qed auto
  show (∑ i ≤ s. m i k * simplicial_vertex j f i)
    = simplicial_vertex j (simplex_map (Suc p) (oriented_simplex s m) f) k
    using one j
    apply (simp add: feq eq simplicial_vertex_oriented_simplex simpli-
cial_simplex_oriented_simplex image_subset_iff)
    apply (drule_tac x=(λi. if i = j then 1 else 0) in bspec)
    apply (auto simp: oriented_simplex_def lss)
    done
  qed
  have 4: chain_map (Suc p) g (subd p (chain_boundary (Suc p) (frag_of f)))
    = subd p (chain_boundary (Suc p) (frag_of (simplex_map (Suc p) g
f)))
    by (metis (no_types) One_nat_def scf Suc.IH chain_boundary_chain_map
chain_map_of_diff_Suc_Suc diff_zero g simplicial_chain_boundary simplicial_imp_singular_chain)
  show chain_map (Suc (Suc p)) g (subd (Suc p) (frag_of f)) = subd (Suc p)
(chain_map (Suc p) g (frag_of f))
    unfolding subd.simps frag_extend_of
    using g
    apply (subst chain_map_simplicial_cone [of s standard_simplex r _ Suc p
s], assumption)
    apply (metis 1 Suc.IH diff_Suc_1 scf simplicial_chain_boundary simpli-
cial_chain_diff)
    using ** apply auto[1]
    apply (rule order_refl)
    unfolding chain_map_of_frag_extend_of
    apply (rule arg_cong2 [where f = simplicial_cone (Suc p)])
    apply (simp add: geq sum_distrib_left oriented_simplex_def ** del:
sum.atMost_Suc flip: sum_divide_distrib)
    using 2 apply (simp only: oriented_simplex_def sum.swap [where A =
{..s}])
    using naturality_simplicial_subdivision scf apply (fastforce simp add: 4
chain_map_diff)
    done
  next
  have sc: simplicial_chain (Suc p) (standard_simplex s)
    (simplicial_cone p

```

```

      (λi. (∑ j ≤ Suc p. simplicial_vertex j f i) / (Suc (Suc p)))
      (simplicial_subdivision p
        (chain_boundary (Suc p) (frag_of f))))
    by (metis diff_Suc_1 nat.simps(3) simplicial_subdivision_of scf simplicial_chain_simplicial_subdivision)
    have ff: simplicial_chain (Suc p) (standard_simplex s) (subd p (chain_boundary (Suc p) (frag_of f)))
      by (metis (no_types) Suc.IH diff_Suc_1 scf simplicial_chain_boundary)
    show simplicial_chain (Suc (Suc p)) (standard_simplex s) (subd (Suc p) (frag_of f))
      using one
      unfolding subd.simps frag_extend_of
      apply (rule_tac S=standard_simplex s in simplicial_chain_simplicial_cone)
      apply (meson ff scf simplicial_chain_diff simplicial_chain_simplicial_subdivision)
      using ** convex_standard_simplex by force
      have simplicial_chain p (standard_simplex s) (chain_boundary (Suc p) (frag_of f))
      using scf simplicial_chain_boundary by fastforce
      then have chain_boundary (Suc p) (simplicial_subdivision (Suc p) (frag_of f) - frag_of f
        - subd p (chain_boundary (Suc p) (frag_of f)))
      = 0
      unfolding chain_boundary_diff
      using Suc.IH chain_boundary_boundary
      by (metis One_nat_def add_diff_cancel_left' chain_boundary_simplicial_subdivision
        diff_Suc_1 scf
        simplicial_imp_singular_chain subd_0)
      moreover have simplicial_chain (Suc p) (standard_simplex s)
        (simplicial_subdivision (Suc p) (frag_of f) - frag_of f -
        subd p (chain_boundary (Suc p) (frag_of f)))
      by (meson ff scf simplicial_chain_diff simplicial_chain_simplicial_subdivision)
      ultimately show chain_boundary (Suc (Suc p)) (subd (Suc p) (frag_of f))
        + subd (Suc p - Suc 0) (chain_boundary (Suc p) (frag_of f))
        = simplicial_subdivision (Suc p) (frag_of f) - frag_of f
      unfolding subd.simps frag_extend_of
      apply (simp add: chain_boundary_simplicial_cone)
      apply (simp add: simplicial_cone_def del: sum.atMost_Suc simplicial_subdivision.simps)
      done
    qed
  next
    case (diff a b)
    then show ?case
      apply safe
      apply (metis chain_map_diff subd_diff)
      apply (metis simplicial_chain_diff subd_diff)
      by (smt (verit, ccfv_threshold) add_diff_add chain_boundary_diff diff_add_cancel
        simplicial_subdivision_diff subd_diff)
    qed auto
  qed simp

```

lemma *chain_homotopic_simplicial_subdivision1*:

$\llbracket \text{simplicial_chain } p \text{ (standard_simplex } q) \text{ } c; \text{ simplicial_simplex } q \text{ (standard_simplex } r) \text{ } g \rrbracket$
 $\implies \text{chain_map (Suc } p) \text{ } g \text{ (subd } p \text{ } c) = \text{subd } p \text{ (chain_map } p \text{ } g \text{ } c)$
by (*simp add: subd*)

lemma *chain_homotopic_simplicial_subdivision2*:

simplicial_chain *p* (standard_simplex *q*) *c*
 $\implies \text{simplicial_chain (Suc } p) \text{ (standard_simplex } q) \text{ (subd } p \text{ } c)$
by (*simp add: subd*)

lemma *chain_homotopic_simplicial_subdivision3*:

simplicial_chain *p* (standard_simplex *q*) *c*
 $\implies \text{chain_boundary (Suc } p) \text{ (subd } p \text{ } c) = (\text{simplicial_subdivision } p \text{ } c) - c -$
 $\text{subd } (p - \text{Suc } 0) \text{ (chain_boundary } p \text{ } c)$
by (*simp add: subd algebra_simps*)

lemma *chain_homotopic_simplicial_subdivision*:

$\exists h. (\forall p. h \text{ } p \text{ } 0 = 0) \wedge$
 $(\forall p \text{ } c1 \text{ } c2. h \text{ } p \text{ (} c1 - c2) = h \text{ } p \text{ } c1 - h \text{ } p \text{ } c2) \wedge$
 $(\forall p \text{ } q \text{ } r \text{ } g \text{ } c.$
 $\quad \text{simplicial_chain } p \text{ (standard_simplex } q) \text{ } c$
 $\quad \longrightarrow \text{simplicial_simplex } q \text{ (standard_simplex } r) \text{ } g$
 $\quad \longrightarrow \text{chain_map (Suc } p) \text{ } g \text{ (} h \text{ } p \text{ } c) = h \text{ } p \text{ (chain_map } p \text{ } g \text{ } c)) \wedge$
 $(\forall p \text{ } q \text{ } c. \text{simplicial_chain } p \text{ (standard_simplex } q) \text{ } c$
 $\quad \longrightarrow \text{simplicial_chain (Suc } p) \text{ (standard_simplex } q) \text{ (} h \text{ } p \text{ } c)) \wedge$
 $(\forall p \text{ } q \text{ } c. \text{simplicial_chain } p \text{ (standard_simplex } q) \text{ } c$
 $\quad \longrightarrow \text{chain_boundary (Suc } p) \text{ (} h \text{ } p \text{ } c) + h \text{ (} p - \text{Suc } 0) \text{ (chain_boundary$
 $\text{ } p \text{ } c)$
 $\quad = (\text{simplicial_subdivision } p \text{ } c) - c)$
by (*rule_tac x=subd in exI*) (*fastforce simp: subd*)

lemma *chain_homotopic_singular_subdivision*:

obtains *h* **where**

$\bigwedge p. h \text{ } p \text{ } 0 = 0$
 $\bigwedge p \text{ } c1 \text{ } c2. h \text{ } p \text{ (} c1 - c2) = h \text{ } p \text{ } c1 - h \text{ } p \text{ } c2$
 $\bigwedge p \text{ } X \text{ } c. \text{singular_chain } p \text{ } X \text{ } c \implies \text{singular_chain (Suc } p) \text{ } X \text{ (} h \text{ } p \text{ } c)$
 $\bigwedge p \text{ } X \text{ } c. \text{singular_chain } p \text{ } X \text{ } c$
 $\implies \text{chain_boundary (Suc } p) \text{ (} h \text{ } p \text{ } c) + h \text{ (} p - \text{Suc } 0) \text{ (chain_boundary$
 $\text{ } p \text{ } c) = \text{singular_subdivision } p \text{ } c - c$

proof –

define *k* **where** $k \equiv \lambda p. \text{frag_extend } (\lambda f.:: (\text{nat} \Rightarrow \text{real}) \Rightarrow 'a. \text{chain_map (Suc } p) \text{ } f \text{ (subd } p \text{ (frag_of(restrict id (standard_simplex } p))))))$

show *?thesis*

proof

fix *p* *X* **and** *c* :: '*a* *chain*

assume *c*: *singular_chain* *p* *X* *c*

have *singular_chain* (Suc *p*) *X* (*k* *p* *c*) \wedge

```

      chain_boundary (Suc p) (k p c) + k (p - Suc 0) (chain_boundary p
c) = singular_subdivision p c - c
    using c [unfolded singular_chain_def]
  proof (induction rule: frag_induction)
    case (one f)
    let ?X = subtopology (powertop_real UNIV) (standard_simplex p)
    show ?case
    proof (simp add: k_def, intro conjI)
      show singular_chain (Suc p) X (chain_map (Suc p) f (subd p (frag_of
(restrict id (standard_simplex p)))))
      proof (rule singular_chain_chain_map)
        show singular_chain (Suc p) ?X (subd p (frag_of (restrict id (standard_simplex
p))))
        by (simp add: chain_homotopic_simplicial_subdivision2 simplicial_imp_singular_chain)
        show continuous_map ?X X f
        using one.hyps singular_simplex_def by auto
      qed
    next
    have scp: singular_chain (Suc p) ?X (subd p (frag_of (restrict id (standard_simplex
p))))
    by (simp add: chain_homotopic_simplicial_subdivision2 simplicial_imp_singular_chain)
    have feqf: frag_of (simplex_map p f (restrict id (standard_simplex p))) =
frag_of f
    using one.hyps singular_simplex_chain_map_id by auto
    have *: chain_map p f
      (subd (p - Suc 0)
      (∑ k ≤ p. frag_cmul ((-1) ^ k) (frag_of (singular_face p k id))))
    = (∑ x ≤ p. frag_cmul ((-1) ^ x)
      (chain_map p (singular_face p x f)
      (subd (p - Suc 0) (frag_of (restrict id (standard_simplex
(p - Suc 0)))))))
    (is ?lhs = ?rhs)
    if p > 0
  proof -
    have eqc: subd (p - Suc 0) (frag_of (singular_face p i id))
      = chain_map p (singular_face p i id)
      (subd (p - Suc 0) (frag_of (restrict id (standard_simplex
(p - Suc 0)))))
    if i ≤ p for i
  proof -
    have 1: simplicial_chain (p - Suc 0) (standard_simplex (p - Suc 0))
      (frag_of (restrict id (standard_simplex (p - Suc 0))))
    by simp
    have 2: simplicial_simplex (p - Suc 0) (standard_simplex p) (singular_face
p i id)
    by (metis One_nat_def Suc_leI ‹0 < p› simplicial_simplex_id
simplicial_simplex_singular_face singular_face_restrict subsetI that)
    have 3: simplex_map (p - Suc 0) (singular_face p i id) (restrict id
(standard_simplex (p - Suc 0)))

```

```

      = singular_face p i id
    by (force simp: simplex_map_def singular_face_def)
  show ?thesis
    using chain_homotopic_simplicial_subdivision1 [OF 1 2]
      that ⟨p > 0⟩ by (simp add: 3)
  qed
  have xx: simplicial_chain p (standard_simplex(p - Suc 0))
    (subd (p - Suc 0) (frag_of (restrict id (standard_simplex (p -
Suc 0)))))
    by (metis Suc_pred chain_homotopic_simplicial_subdivision2 order_refl
simplicial_chain_of simplicial_simplex_id that)
  have yy:  $\bigwedge k. k \leq p \implies$ 
    chain_map p f
    (chain_map p (singular_face p k id) h) = chain_map p (singular_face
p k f) h
    if simplicial_chain p (standard_simplex(p - Suc 0)) h for h
    using that unfolding simplicial_chain_def
  proof (induction h rule: frag_induction)
    case (one x)
    then show ?case
      using one
      apply (simp add: chain_map_of_singular_simplex_def simplicial_simplex_def, auto)
      apply (rule arg_cong [where f=frag_of])
      by (auto simp: image_subset_iff simplex_map_def simplicial_simplex
singular_face_def)

  qed (auto simp: chain_map_diff)
  have ?lhs
    = chain_map p f
      ( $\sum_{k \leq p} \text{frag\_cmul } ((-1) \wedge k)$ 
      (chain_map p (singular_face p k id)
      (subd (p - Suc 0) (frag_of (restrict id (standard_simplex
(p - Suc 0)))))
      )))
    by (simp add: subd_power_sum subd_power_uminus eqc)
  also have ... = ?rhs
    by (simp add: chain_map_sum xx yy)
  finally show ?thesis .
  qed
  have chain_map p f
    (simplicial_subdivision p (frag_of (restrict id (standard_simplex
p))))
    - subd (p - Suc 0) (chain_boundary p (frag_of (restrict id
(standard_simplex p))))
    = singular_subdivision p (frag_of f)
    - frag_extend
      ( $\lambda f. \text{chain\_map } (\text{Suc } (p - \text{Suc } 0)) f$ 
      (subd (p - Suc 0) (frag_of (restrict id (standard_simplex (p
- Suc 0)))))

```

```

      (chain_boundary p (frag_of f))
    apply (simp add: singular_subdivision_def chain_map_diff)
    apply (clarsimp simp add: chain_boundary_def)
    apply (simp add: frag_extend_sum frag_extend_cmul *)
  done
  then show chain_boundary (Suc p) (chain_map (Suc p) f (subd p (frag_of
(restrict id (standard_simplex p))))))
    + frag_extend
      (λf. chain_map (Suc (p - Suc 0)) f
        (subd (p - Suc 0) (frag_of (restrict id (standard_simplex (p
- Suc 0))))))
      (chain_boundary p (frag_of f))
      = singular_subdivision p (frag_of f) - frag_of f
    by (simp add: chain_boundary_chain_map [OF scp] chain_homotopic_simplicial_subdivision3
[where q=p] chain_map_diff feqf)
  qed
next
case (diff a b)
then show ?case
  apply (simp only: k_def singular_chain_diff chain_boundary_diff frag_extend_diff
singular_subdivision_diff)
  by (metis (no_types, lifting) add_diff_add diff_add_cancel)
  qed (auto simp: k_def)
  then show singular_chain (Suc p) X (k p c) chain_boundary (Suc p) (k p c)
+ k (p - Suc 0) (chain_boundary p c) = singular_subdivision p c - c
    by auto
  qed (auto simp: k_def frag_extend_diff)
qed

lemma homologous_rel_singular_subdivision:
  assumes singular_relcycle p X T c
  shows homologous_rel p X T (singular_subdivision p c) c
proof (cases p = 0)
case True
  with assms show ?thesis
  by (auto simp: singular_relcycle_def singular_subdivision_zero)
next
case False
  with assms show ?thesis
  unfolding homologous_rel_def singular_relboundary singular_relcycle
  by (metis One_nat_def Suc_diff_1 chain_homotopic_singular_subdivision
gr_zeroI)
qed

```

0.1.17 Excision argument that we keep doing singular subdivision

lemma *singular_subdivision_power_0* [simp]: (*singular_subdivision* $p \rightsquigarrow n$) 0 = 0
 by (*induction* n) *auto*

lemma *singular_subdivision_power_diff*:
 (*singular_subdivision* $p \rightsquigarrow n$) ($a - b$) = (*singular_subdivision* $p \rightsquigarrow n$) $a -$
 (*singular_subdivision* $p \rightsquigarrow n$) b
 by (*induction* n) (*auto simp: singular_subdivision_diff*)

lemma *iterated_singular_subdivision*:
singular_chain $p \ X \ c$
 \implies (*singular_subdivision* $p \rightsquigarrow n$) $c =$
frag_extend
 ($\lambda f.$ *chain_map* $p \ f$
 ((*simplicial_subdivision* $p \rightsquigarrow n$)
 (*frag_of*(*restrict id* (*standard_simplex* p)))))) c

proof (*induction* n *arbitrary: c*)

case 0

then show ?case

unfolding *singular_chain_def*

proof (*induction* c *rule: frag_induction*)

case (*one* f)

then have *restrict* f (*standard_simplex* p) = f

by (*simp add: extensional_restrict singular_simplex_def*)

then show ?case

by (*auto simp: simplex_map_def cong: restrict_cong*)

qed (*auto simp: frag_extend_diff*)

next

case (*Suc* n)

show ?case

using *Suc.premis* unfolding *singular_chain_def*

proof (*induction* c *rule: frag_induction*)

case (*one* f)

then have *singular_simplex* $p \ X \ f$

by *simp*

have *scp*: *simplicial_chain* p (*standard_simplex* p)

((*simplicial_subdivision* $p \rightsquigarrow n$) (*frag_of* (*restrict id* (*standard_simplex* p))))

proof (*induction* n)

case 0

then show ?case

by (*metis funpow_0 order_refl simplicial_chain_of simplicial_simplex_id*)

next

case (*Suc* n)

then show ?case

by (*simp add: simplicial_chain_simplicial_subdivision*)

qed


```

have scnp: simplicial_chain p (standard_simplex p)
  ((simplicial_subdivision p  $\sim$  n) (frag_of ( $\lambda x \in \text{standard\_simplex } p.$ 
x)))
proof (induction n)
  case 0
  then show ?case
    by (metis eq_id_iff_funpow_0 order_refl simplicial_chain_of_simpli-
cial_simplex_id)
  next
  case (Suc n)
  then show ?case
    by (simp add: simplicial_chain_simplicial_subdivision)
qed
have sff: singular_chain p X (frag_of f)
  by (simp add: singular_simplex_p_X_f singular_chain_of)
then show ?case
  using Suc.IH [OF sff] naturality_singular_subdivision [OF simplicial_imp_singular_chain
[OF scp], of f] singular_subdivision_simplicial_simplex [OF scnp]
  by (simp add: singular_chain_of_id_def del: restrict_apply)
qed (auto simp: singular_subdivision_power_diff singular_subdivision_diff frag_extend_diff)
qed

```

lemma *chain_homotopic_iterated_singular_subdivision*:

obtains *h* **where**

$$\begin{aligned}
& \bigwedge p. h \ p \ 0 = (0 :: 'a \ \text{chain}) \\
& \bigwedge p \ c1 \ c2. h \ p \ (c1 - c2) = h \ p \ c1 - h \ p \ c2 \\
& \bigwedge p \ X \ c. \text{singular_chain } p \ X \ c \implies \text{singular_chain } (\text{Suc } p) \ X \ (h \ p \ c) \\
& \bigwedge p \ X \ c. \text{singular_chain } p \ X \ c \\
& \implies \text{chain_boundary } (\text{Suc } p) \ (h \ p \ c) + h \ (p - \text{Suc } 0) \ (\text{chain_boundary} \\
& p \ c) \\
& = (\text{singular_subdivision } p \ \sim n) \ c - c
\end{aligned}$$

proof (*induction* n *arbitrary: thesis*)

case 0

show ?case

by (*rule* 0 [*of* ($\lambda p \ x. 0$)]) *auto*

next

case (*Suc* n)

then obtain *k* **where** *k*:

$$\begin{aligned}
& \bigwedge p. k \ p \ 0 = (0 :: 'a \ \text{chain}) \\
& \bigwedge p \ c1 \ c2. k \ p \ (c1 - c2) = k \ p \ c1 - k \ p \ c2 \\
& \bigwedge p \ X \ c. \text{singular_chain } p \ X \ c \implies \text{singular_chain } (\text{Suc } p) \ X \ (k \ p \ c) \\
& \bigwedge p \ X \ c. \text{singular_chain } p \ X \ c \\
& \implies \text{chain_boundary } (\text{Suc } p) \ (k \ p \ c) + k \ (p - \text{Suc } 0) \ (\text{chain_boundary} \\
& p \ c) \\
& = (\text{singular_subdivision } p \ \sim n) \ c - c
\end{aligned}$$

by *metis*

obtain *h* **where** *h*:

$\bigwedge p. h \ p \ 0 = (0 :: 'a \ \text{chain})$

```


$$\bigwedge p \ c1 \ c2. \ h \ p \ (c1 - c2) = h \ p \ c1 - h \ p \ c2$$


$$\bigwedge p \ X \ c. \ singular\_chain \ p \ X \ c \implies singular\_chain \ (Suc \ p) \ X \ (h \ p \ c)$$


$$\bigwedge p \ X \ c. \ singular\_chain \ p \ X \ c$$


$$\implies chain\_boundary \ (Suc \ p) \ (h \ p \ c) + h \ (p - Suc \ 0) \ (chain\_boundary \ p \ c) = singular\_subdivision \ p \ c - c$$

by (blast intro: chain_homotopic_singular_subdivision)
let ?h = ( $\lambda p \ c. \ singular\_subdivision \ (Suc \ p) \ (k \ p \ c) + h \ p \ c$ )
show ?case
proof (rule Suc.premis)
  fix p X and c :: 'a chain
  assume singular_chain p X c
  then show singular_chain (Suc p) X (?h p c)
    by (simp add: h k singular_chain_add singular_chain_singular_subdivision)
  next
  fix p :: nat and X :: 'a topology and c :: 'a chain
  assume sc: singular_chain p X c
  have f5: chain_boundary (Suc p) (singular_subdivision (Suc p) (k p c)) =
singular_subdivision p (chain_boundary (Suc p) (k p c))
    using chain_boundary_singular_subdivision k(3) sc by fastforce
  have [simp]: singular_subdivision (Suc (p - Suc 0)) (k (p - Suc 0) (chain_boundary
p c)) =
    singular_subdivision p (k (p - Suc 0) (chain_boundary p c))
  proof (cases p)
    case 0
    then show ?thesis
      by (simp add: k chain_boundary_def)
    qed auto
  show chain_boundary (Suc p) (?h p c) + ?h (p - Suc 0) (chain_boundary p
c) = (singular_subdivision p  $\widehat{\sim}$  Suc n) c - c
    using chain_boundary_singular_subdivision [of Suc p X]
    apply (simp add: chain_boundary_add f5 h k algebra_simps)
    by (smt (verit, del_insts) add.commute add.left_commute diff_add_cancel
h(4) k(4) sc singular_subdivision_add)
  qed (auto simp: k h singular_subdivision_diff)
qed

```

lemma llemma:

```

assumes p: standard_simplex p  $\subseteq \bigcup \mathcal{C}$ 
and C:  $\bigwedge U. U \in \mathcal{C} \implies openin \ (powertop\_real \ UNIV) \ U$ 
obtains d where 0 < d

$$\bigwedge K. \llbracket K \subseteq standard\_simplex \ p; \bigwedge x \ y \ i. \llbracket i \leq p; x \in K; y \in K \rrbracket \implies |x \ i - y \ i| \leq d \rrbracket$$


$$\implies \exists U. U \in \mathcal{C} \wedge K \subseteq U$$


```

proof –

```

have  $\exists e \ U. \ 0 < e \wedge U \in \mathcal{C} \wedge x \in U \wedge$ 
 $(\forall y. (\forall i \leq p. |y \ i - x \ i| \leq 2 * e) \wedge (\forall i > p. y \ i = 0) \longrightarrow y \in U)$ 
if x: x in standard_simplex p for x

```

proof –

```

obtain U where U: U  $\in \mathcal{C}$  x  $\in U$ 

```

```

    using  $x\ p$  by blast
  then obtain  $V$  where  $\text{fin}V$ :  $\text{finite } \{i. V\ i \neq \text{UNIV}\}$  and  $\text{open}V$ :  $\bigwedge i. \text{open}$ 
  ( $V\ i$ )
    and  $xV$ :  $x \in \text{Pi}_E\ \text{UNIV}\ V$  and  $UV$ :  $\text{Pi}_E\ \text{UNIV}\ V \subseteq U$ 
    using  $\mathcal{C}$  unfolding  $\text{openin\_product\_topology\_alt}$  by force
  have  $xVi$ :  $x\ i \in V\ i$  for  $i$ 
    using  $\text{PiE\_mem } [OF\ xV]$  by simp
  have  $\bigwedge i. \exists e > 0. \forall x'. |x' - x\ i| < e \longrightarrow x' \in V\ i$ 
    by (rule  $\text{open}V$  [unfolded  $\text{open\_real}$ ,  $\text{rule\_format}$ ,  $OF\ xVi$ ])
  then obtain  $d$  where  $d$ :  $\bigwedge i. d\ i > 0$  and  $dV$ :  $\bigwedge i\ x'. |x' - x\ i| < d\ i \implies x' \in V\ i$ 
    by metis
  define  $e$  where  $e \equiv \text{Inf } (\text{insert } 1\ (d\ ` \{i. V\ i \neq \text{UNIV}\})) / 3$ 
  have  $\text{ed3}$ :  $e \leq d\ i / 3$  if  $V\ i \neq \text{UNIV}$  for  $i$ 
    using that  $\text{fin}V$  by (auto simp:  $e\_def$  intro:  $cInf\_le\_finite$ )
  show  $\exists e\ U. 0 < e \wedge U \in \mathcal{C} \wedge x \in U \wedge$ 
    ( $\forall y. (\forall i \leq p. |y\ i - x\ i| \leq 2 * e) \wedge (\forall i > p. y\ i = 0) \longrightarrow y \in U$ )
  proof (intro  $\text{exI conjI allI impI}$ )
    show  $e > 0$ 
      using  $d\ \text{fin}V$  by (simp add:  $e\_def$   $\text{finite\_less\_Inf\_iff}$ )
    fix  $y$  assume  $y$ :  $(\forall i \leq p. |y\ i - x\ i| \leq 2 * e) \wedge (\forall i > p. y\ i = 0)$ 
    have  $y \in \text{Pi}_E\ \text{UNIV}\ V$ 
    proof
      show  $y\ i \in V\ i$  for  $i$ 
      proof (cases  $p < i$ )
        case True
        then show ?thesis
        by (metis ( $\text{mono\_tags}$ ,  $\text{lifting}$ )  $y\ x\ \text{mem\_Collect\_eq standard\_simplex\_def}$ 
 $xVi$ )
      next
        case False show ?thesis
        proof (cases  $V\ i = \text{UNIV}$ )
          case False show ?thesis
          proof (rule  $dV$ )
            have  $|y\ i - x\ i| \leq 2 * e$ 
            using  $y\ \langle \neg p < i \rangle$  by simp
            also have  $\dots < d\ i$ 
            using  $\text{ed3 } [OF\ False] \langle e > 0 \rangle$  by simp
            finally show  $|y\ i - x\ i| < d\ i$  .
          qed
        qed auto
      qed
    with  $UV$  show  $y \in U$ 
    by blast
  qed (use  $U$  in auto)
qed
then obtain  $e\ U$  where
   $eU$ :  $\bigwedge x. x \in \text{standard\_simplex } p \implies$ 

```

```

    0 < e x ∧ U x ∈ C ∧ x ∈ U x
    and UI: ∧x y. [x ∈ standard_simplex p; ∧i. i ≤ p ⇒ |y i - x i| ≤ 2 * e
x; ∧i. i > p ⇒ y i = 0]
    ⇒ y ∈ U x

  by metis
  define F where F ≡ λx. Pi_E UNIV (λi. if i ≤ p then {x i - e x .. x i + e
x} else UNIV)
  have ∀ S ∈ F ' standard_simplex p. openin (powertop_real UNIV) S
  by (simp add: F_def openin_PiE_gen)
  moreover have pF: standard_simplex p ⊆ ∪ (F ' standard_simplex p)
  by (force simp: F_def PiE_iff eU)
  ultimately have ∃ F. finite F ∧ F ⊆ F ' standard_simplex p ∧ standard_simplex
p ⊆ ∪ F
  using compactin_standard_simplex [of p]
  unfolding compactin_def by force
  then obtain S where finite S and ssp: S ⊆ standard_simplex p standard_simplex
p ⊆ ∪ (F ' S)
  unfolding ex_finite_subset_image by (auto simp: ex_finite_subset_image)
  then have S ≠ {}
  by (auto simp: nonempty_standard_simplex)
  show ?thesis
  proof
    show Inf (e ' S) > 0
    using ⟨finite S⟩ ⟨S ≠ {}⟩ ssp eU by (auto simp: finite_less_Inf_iff)
    fix k :: (nat ⇒ real) set
    assume k: k ⊆ standard_simplex p
    and kle: ∧x y i. [i ≤ p; x ∈ k; y ∈ k] ⇒ |x i - y i| ≤ Inf (e ' S)
    show ∃ U. U ∈ C ∧ k ⊆ U
    proof (cases k = {})
      case True
      then show ?thesis
      using ⟨S ≠ {}⟩ eU equals0I ssp(1) subset_eq p by auto
    next
      case False
      with k ssp obtain x a where x ∈ k x ∈ standard_simplex p
      and a: a ∈ S and Fa: x ∈ F a

      by blast
      then have le_ea: ∧i. i ≤ p ⇒ abs (x i - a i) < e a
      by (simp add: F_def PiE_iff if_distrib abs_diff_less_iff cong: if_cong)
      show ?thesis
      proof (intro exI conjI)
        show U a ∈ C
        using a eU ssp(1) by auto
        show k ⊆ U a
        proof clarify
          fix y assume y ∈ k
          with k have y: y ∈ standard_simplex p
          by blast
          show y ∈ U a

```

```

proof (rule UI)
  show  $a \in \text{standard\_simplex } p$ 
    using  $a \text{ ssp}(1)$  by auto
  fix  $i :: \text{nat}$ 
  assume  $i \leq p$ 
  then have  $|x\ i - y\ i| \leq e\ a$ 
    by (meson kle [OF  $\langle i \leq p \rangle$ ]  $a \langle \text{finite } S \rangle \langle x \in k \rangle \langle y \in k \rangle \text{cInf\_le\_finite}$ 
     $\text{finite\_imageI imageI order\_trans}$ )
  then show  $|y\ i - a\ i| \leq 2 * e\ a$ 
    using  $\text{le\_ea}$  [OF  $\langle i \leq p \rangle$ ] by linarith
  next
    fix  $i$  assume  $p < i$ 
    then show  $y\ i = 0$ 
      using  $\text{standard\_simplex\_def } y$  by auto
    qed
  qed
qed
qed
qed
qed

```

proposition *sufficient_iterated_singular_subdivision_exists:*

assumes $\mathcal{C}: \bigwedge U. U \in \mathcal{C} \implies \text{openin } X\ U$
and $X: \text{topspace } X \subseteq \bigcup \mathcal{C}$
and $p: \text{singular_chain } p\ X\ c$
obtains n **where** $\bigwedge m\ f. \llbracket n \leq m; f \in \text{Poly_Mapping.keys } ((\text{singular_subdivision } p \hat{\sim} m)\ c) \rrbracket$
 $\implies \exists V \in \mathcal{C}. f' (\text{standard_simplex } p) \subseteq V$

```

proof (cases  $c = 0$ )
  case False
    then show ?thesis
  proof (cases  $\text{topspace } X = \{\}$ )
    case True
      show ?thesis
      using  $p$  that by (force simp: singular_chain_empty True)
    next
      case False
        show ?thesis
  proof (cases  $\mathcal{C} = \{\}$ )
    case True
      then show ?thesis
      using False  $X$  by blast
    next
      case False
        have  $\exists e. 0 < e \wedge$ 
           $(\forall K. K \subseteq \text{standard\_simplex } p \longrightarrow (\forall x\ y\ i. x \in K \wedge y \in K \wedge i \leq p$ 
           $\longrightarrow |x\ i - y\ i| \leq e))$ 
           $\longrightarrow (\exists V. V \in \mathcal{C} \wedge f' K \subseteq V)$ 

```

```

    if f: f ∈ Poly_Mapping.keys c for f
  proof -
    have ssf: singular_simplex p X f
      using f p by (auto simp: singular_chain_def)
    then have fp:  $\bigwedge x. x \in \text{standard\_simplex } p \implies f x \in \text{topspace } X$ 
      by (auto simp: singular_simplex_def image_subset_iff dest: continuous_map_image_subset_topspace)
    have  $\exists T. \text{openin } (\text{powertop\_real UNIV}) T \wedge$ 
       $\text{standard\_simplex } p \cap f^{-1} V = T \cap \text{standard\_simplex } p$ 
      if V: V ∈ C for V
    proof -
      have singular_simplex p X f
        using p f unfolding singular_chain_def by blast
      then have openin (subtopology (powertop_real UNIV) (standard_simplex p))
         $\{x \in \text{standard\_simplex } p. f x \in V\}$ 
        using C [OF V ∈ C] by (simp add: singular_simplex_def continuous_map_def)
      moreover have  $\text{standard\_simplex } p \cap f^{-1} V = \{x \in \text{standard\_simplex } p. f x \in V\}$ 
        by blast
      ultimately show ?thesis
        by (simp add: openin_subtopology)
    qed
    then obtain g where gope:  $\bigwedge V. V \in C \implies \text{openin } (\text{powertop\_real UNIV}) (g V)$ 
      and geq:  $\bigwedge V. V \in C \implies \text{standard\_simplex } p \cap f^{-1} V = g V \cap \text{standard\_simplex } p$ 
      by metis
    obtain d where 0 < d
      and d:  $\bigwedge K. \llbracket K \subseteq \text{standard\_simplex } p; \bigwedge x y i. \llbracket i \leq p; x \in K; y \in K \rrbracket \implies |x i - y i| \leq d \rrbracket$ 
       $\implies \exists U. U \in g^{-1} C \wedge K \subseteq U$ 
    proof (rule llemma [of p g^{-1} C])
      show  $\text{standard\_simplex } p \subseteq \bigcup (g^{-1} C)$ 
        using geq X fp by (fastforce simp add:)
      show  $\text{openin } (\text{powertop\_real UNIV}) U$  if U ∈ g^{-1} C for U :: (nat  $\Rightarrow$  real) set
        using gope that by blast
    qed auto
    show ?thesis
      proof (rule exI, intro allI conjI impI)
        fix K :: (nat  $\Rightarrow$  real) set
        assume K: K ⊆ standard_simplex p
          and Kd:  $\forall x y i. x \in K \wedge y \in K \wedge i \leq p \longrightarrow |x i - y i| \leq d$ 
        then have  $\exists U. U \in g^{-1} C \wedge K \subseteq U$ 
          using d [OF K] by auto
        then show  $\exists V. V \in C \wedge f^{-1} K \subseteq V$ 
          using K geq by fastforce
      qed
  
```

```

qed (rule ‹d > 0›)
qed
then obtain  $\psi$  where epos:  $\forall f \in \text{Poly\_Mapping.keys } c. 0 < \psi f$ 
and e:  $\bigwedge f K. \llbracket f \in \text{Poly\_Mapping.keys } c; K \subseteq \text{standard\_simplex } p; \bigwedge x y i. x \in K \wedge y \in K \wedge i \leq p \implies |x i - y i| \leq \psi f \rrbracket \implies \exists V. V \in \mathcal{C} \wedge f ' K \subseteq V$ 

by metis
obtain d where 0 < d
and d:  $\bigwedge f K. \llbracket f \in \text{Poly\_Mapping.keys } c; K \subseteq \text{standard\_simplex } p; \bigwedge x y i. \llbracket x \in K; y \in K; i \leq p \rrbracket \implies |x i - y i| \leq d \rrbracket \implies \exists V. V \in \mathcal{C} \wedge f ' K \subseteq V$ 

proof
show Inf ( $\psi ' \text{Poly\_Mapping.keys } c$ ) > 0
by (simp add: finite_less_Inf_iff ‹c ≠ 0› epos)
fix f K
assume fK:  $f \in \text{Poly\_Mapping.keys } c \wedge K \subseteq \text{standard\_simplex } p$ 
and le:  $\bigwedge x y i. \llbracket x \in K; y \in K; i \leq p \rrbracket \implies |x i - y i| \leq \text{Inf } (\psi ' \text{Poly\_Mapping.keys } c)$ 
then have lef:  $\text{Inf } (\psi ' \text{Poly\_Mapping.keys } c) \leq \psi f$ 
by (auto intro: cInf_le_finite)
show  $\exists V. V \in \mathcal{C} \wedge f ' K \subseteq V$ 
using le lef by (blast intro: dual_order.trans e [OF fK])
qed
let ?d =  $\lambda m. (\text{simplicial\_subdivision } p \rightsquigarrow m) (\text{frag\_of } (\text{restrict id } (\text{standard\_simplex } p)))$ 
obtain n where  $n: (p / (\text{Suc } p)) \wedge n < d$ 
using real_arch_pow_inv ‹0 < d› by fastforce
show ?thesis
proof
fix m h
assume  $n \leq m$  and  $h \in \text{Poly\_Mapping.keys } ((\text{singular\_subdivision } p \rightsquigarrow m) c)$ 
then obtain f where  $f \in \text{Poly\_Mapping.keys } c \wedge h \in \text{Poly\_Mapping.keys } (\text{chain\_map } p f (\text{?d } m))$ 
using subsetD [OF keys_frag_extend] iterated_singular_subdivision [OF p, of m] by force
then obtain g where  $g: g \in \text{Poly\_Mapping.keys } (\text{?d } m)$  and heq:  $h = \text{restrict } (f \circ g) (\text{standard\_simplex } p)$ 
using keys_frag_extend by (force simp: chain_map_def simplex_map_def)
have xx:  $\text{simplicial\_chain } p (\text{standard\_simplex } p) (\text{?d } n) \wedge (\forall f \in \text{Poly\_Mapping.keys } (\text{?d } n). \forall x \in \text{standard\_simplex } p. \forall y \in \text{standard\_simplex } p. |f x i - f y i| \leq (p / (\text{Suc } p)) \wedge n)$ 
for n i
proof (induction n)
case 0
have simplicial_simplex p (standard_simplex p) ( $\lambda a \in \text{standard\_simplex } p. a$ )
by (metis eq_id_iff order_refl simplicial_simplex_id)

```

```

    moreover have ( $\forall x \in \text{standard\_simplex } p. \forall y \in \text{standard\_simplex } p. |x \ i - y \ i| \leq 1$ )
      unfolding standard_simplex_def
      by (auto simp: abs_if dest!: spec [where x=i])
    ultimately show ?case
      unfolding power_0 funpow_0 by simp
  next
    case (Suc n)
    show ?case
      unfolding power_Suc funpow.simps o_def
      proof (intro conjI ballI)
        show simplicial_chain p (standard_simplex p) (simplicial_subdivision p
(?d n))
          by (simp add: Suc simplicial_chain_simplicial_subdivision)
        show  $|f \ x \ i - f \ y \ i| \leq \text{real } p / \text{real } (\text{Suc } p) * (\text{real } p / \text{real } (\text{Suc } p)) ^ n$ 
          if  $f \in \text{Poly\_Mapping.keys } (\text{simplicial\_subdivision } p \ (\text{?d } n))$ 
          and  $x \in \text{standard\_simplex } p$  and  $y \in \text{standard\_simplex } p$  for  $f \ x \ y$ 
          using Suc that by (blast intro: simplicial_subdivision_shrinks)
        qed
      qed
      have  $g \text{ 'standard\_simplex } p \subseteq \text{standard\_simplex } p$ 
        using g xx [of m] unfolding simplicial_chain_def simplicial_simplex by
auto
      moreover
        have  $|g \ x \ i - g \ y \ i| \leq d$  if  $i \leq p$   $x \in \text{standard\_simplex } p$   $y \in \text{standard\_simplex } p$  for  $x \ y \ i$ 
        proof -
          have  $|g \ x \ i - g \ y \ i| \leq (p / (\text{Suc } p)) ^ m$ 
            using g xx [of m] that by blast
          also have  $\dots \leq (p / (\text{Suc } p)) ^ n$ 
            by (auto intro: power_decreasing [OF ⟨n ≤ m⟩])
          finally show ?thesis using n by simp
        qed
      then have  $|x \ i - y \ i| \leq d$ 
        if  $x \in g \text{ 'standard\_simplex } p$   $y \in g \text{ 'standard\_simplex } p$   $i \leq p$  for  $i$ 
        x y
        using that by blast
      ultimately show  $\exists V \in \mathcal{C}. h \text{ 'standard\_simplex } p \subseteq V$ 
        using ⟨f ∈ Poly_Mapping.keys c⟩ d [of f g 'standard_simplex p]
        by (simp add: Bex_def heq image_image)
      qed
    qed
  qed
qed force

```

lemma *small_homologous_rel_relcycle_exists:*

assumes $\mathcal{C}: \bigwedge U. U \in \mathcal{C} \implies \text{openin } X \ U$
and $X: \text{topspace } X \subseteq \bigcup \mathcal{C}$


```

    and p: singular_relcycle p X S c
  obtains c' where singular_relcycle p X S c' homologous_rel p X S c c'
     $\wedge f. f \in \text{Poly\_Mapping.keys } c' \implies \exists V \in \mathcal{C}. f \text{ ' (standard\_simplex$ 
 $p) \subseteq V$ 
  proof -
    have singular_chain p X c
      (chain_boundary p c, 0)  $\in$  (mod_subset (p - Suc 0) (subtopology X S))
    using p unfolding singular_relcycle_def by auto
    then obtain n where n:  $\wedge m f. \llbracket n \leq m; f \in \text{Poly\_Mapping.keys } ((\text{singular\_subdivision}$ 
 $p \text{ } \widehat{\sim} m) \text{ } c) \rrbracket$ 
       $\implies \exists V \in \mathcal{C}. f \text{ ' (standard\_simplex } p) \subseteq V$ 
    by (blast intro: sufficient_iterated_singular_subdivision_exists [OF C X])
    let ?c' = (singular_subdivision p  $\widehat{\sim}$  n) c
    show ?thesis
    proof
      show homologous_rel p X S c ?c'
      proof (induction n)
        case 0
        then show ?case by auto
      next
        case (Suc n)
        then show ?case
          by simp (metis homologous_rel_eq p homologous_rel_singular_subdivision
            homologous_rel_singular_relcycle)
      qed
      then show singular_relcycle p X S ?c'
        by (metis homologous_rel_singular_relcycle p)
    next
      fix f :: (nat  $\Rightarrow$  real)  $\Rightarrow$  'a
      assume f  $\in$  Poly_Mapping.keys ?c'
      then show  $\exists V \in \mathcal{C}. f \text{ ' standard\_simplex } p \subseteq V$ 
        by (rule n [OF order_refl])
    qed
  qed

lemma excised_chain_exists:
  fixes S :: 'a set
  assumes X closure_of U  $\subseteq$  X interior_of T T  $\subseteq$  S singular_chain p (subtopology
    X S) c
  obtains n d e where singular_chain p (subtopology X (S - U)) d
    singular_chain p (subtopology X T) e
    (singular_subdivision p  $\widehat{\sim}$  n) c = d + e
  proof -
    have *:  $\exists n d e. \text{singular\_chain } p \text{ (subtopology } X \text{ (S - U)) } d \wedge$ 
      singular_chain p (subtopology X T) e  $\wedge$ 
      (singular_subdivision p  $\widehat{\sim}$  n) c = d + e
    if c: singular_chain p (subtopology X S) c
      and X: X closure_of U  $\subseteq$  X interior_of T U  $\subseteq$  topspace X and S: T  $\subseteq$  S
      S  $\subseteq$  topspace X

```

```

    for  $p \ X \ c \ S$  and  $T \ U :: 'a \ set$ 
  proof -
    obtain  $n$  where  $n: \bigwedge m f. \llbracket n \leq m; f \in Poly\_Mapping.keys \ ((singular\_subdivision \ p \ \frown m) \ c) \rrbracket$ 
       $\implies \exists V \in \{S \cap X \ interior\_of \ T, S - X \ closure\_of \ U\}. f$ 
    'standard_simplex  $p \subseteq V$ 
      apply (rule sufficient_iterated_singular_subdivision_exists
        [of  $\{S \cap X \ interior\_of \ T, S - X \ closure\_of \ U\}$ ])
      using  $X \ S \ c$ 
      by (auto simp: topspace_subtopology openin_subtopology_Int2 openin_subtopology_diff_closed)
    let  $?c' = \lambda n. (singular\_subdivision \ p \ \frown n) \ c$ 
    have singular_chain  $p \ (subtopology \ X \ S) \ (?c' \ m)$  for  $m$ 
      by (induction  $m$ ) (auto simp: singular_chain_singular_subdivision  $c$ )
    then have scp: singular_chain  $p \ (subtopology \ X \ S) \ (?c' \ n)$  .

    have SS:  $Poly\_Mapping.keys \ (?c' \ n) \subseteq singular\_simplex\_set \ p \ (subtopology \ X \ (S - U))$ 
       $\cup singular\_simplex\_set \ p \ (subtopology \ X \ T)$ 
    proof (clarsimp)
      fix  $f$ 
      assume  $f: f \in Poly\_Mapping.keys \ ((singular\_subdivision \ p \ \frown n) \ c)$ 
      and non:  $\neg singular\_simplex \ p \ (subtopology \ X \ T) \ f$ 
      show singular_simplex  $p \ (subtopology \ X \ (S - U)) \ f$ 
      using  $n \ [OF \ order\_refl \ f] \ scp \ f \ non \ closure\_of\_subset \ [OF \ \langle U \subseteq topspace \ X \rangle \ interior\_of\_subset \ [of \ X \ T]]$ 
      by (fastforce simp: image_subset_iff singular_simplex_subtopology_singular_chain_def)
    qed
    show ?thesis
      unfolding singular_chain_def using frag_split  $[OF \ SS]$  by metis
  qed
  have (subtopology  $X \ (topspace \ X \cap S) = (subtopology \ X \ S)$ 
    by (metis subtopology_subtopology_subtopology_topspace)
  with assms have  $c: singular\_chain \ p \ (subtopology \ X \ (topspace \ X \cap S)) \ c$ 
    by simp
  have Xsub:  $X \ closure\_of \ (topspace \ X \cap U) \subseteq X \ interior\_of \ (topspace \ X \cap T)$ 
    using assms closure_of_restrict interior_of_restrict by fastforce
  obtain  $n \ d \ e$  where
     $d: singular\_chain \ p \ (subtopology \ X \ (topspace \ X \cap S - topspace \ X \cap U)) \ d$ 
    and  $e: singular\_chain \ p \ (subtopology \ X \ (topspace \ X \cap T)) \ e$ 
    and  $de: (singular\_subdivision \ p \ \frown n) \ c = d + e$ 
    using  $*[OF \ c \ Xsub, simplified] \ assms$  by force
  show thesis
  proof
    show singular_chain  $p \ (subtopology \ X \ (S - U)) \ d$ 
      by (metis  $d \ Diff\_Int\_distrib \ inf.cobounded2 \ singular\_chain\_mono$ )
    show singular_chain  $p \ (subtopology \ X \ T) \ e$ 
      by (metis  $e \ inf.cobounded2 \ singular\_chain\_mono$ )
    show  $(singular\_subdivision \ p \ \frown n) \ c = d + e$ 

```

by (rule de)
qed
qed

lemma *excised_relcycle_exists*:

fixes $S :: 'a \text{ set}$
assumes $X: X \text{ closure_of } U \subseteq X \text{ interior_of } T$ **and** $T \subseteq S$
and $c: \text{singular_relcycle } p \text{ (subtopology } X \text{ } S) \text{ } T \text{ } c$
obtains c' **where** $\text{singular_relcycle } p \text{ (subtopology } X \text{ } (S - U)) \text{ } (T - U) \text{ } c'$
 $\text{homologous_rel } p \text{ (subtopology } X \text{ } S) \text{ } T \text{ } c \text{ } c'$
proof –
have $[simp]: (S - U) \cap (T - U) = T - U \text{ } S \cap T = T$
using $\langle T \subseteq S \rangle$ **by** *auto*
have $scc: \text{singular_chain } p \text{ (subtopology } X \text{ } S) \text{ } c$
and $scp1: \text{singular_chain } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } T) \text{ (chain_boundary } p$
 $c)$
using c **by** $(\text{auto simp: singular_relcycle_def mod_subset_def subtopology_subtopology})$
obtain $n \text{ } d \text{ } e$ **where** $d: \text{singular_chain } p \text{ (subtopology } X \text{ } (S - U)) \text{ } d$
and $e: \text{singular_chain } p \text{ (subtopology } X \text{ } T) \text{ } e$
and $de: (\text{singular_subdivision } p \text{ } \hat{\sim} n) \text{ } c = d + e$
using *excised_chain_exists* $[OF \text{ } X \text{ } \langle T \subseteq S \rangle \text{ } scc]$.
have $scSud: \text{singular_chain } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } (S - U)) \text{ (chain_boundary } p$
 $d)$
by $(\text{simp add: singular_chain_boundary } d)$
have $scen: \text{singular_chain } p \text{ (subtopology } X \text{ } S) ((\text{singular_subdivision } p \text{ } \hat{\sim} n) \text{ } c)$
for n
by $(\text{induction } n) (\text{auto simp: singular_chain_singular_subdivision } scc)$
have $\text{singular_chain } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } T) \text{ (chain_boundary } p ((\text{singular_subdivision } p$
 $\hat{\sim} n) \text{ } c))$
proof $(\text{induction } n)$
case $(\text{Suc } n)$
then show $?case$
by $(\text{simp add: singular_chain_singular_subdivision chain_boundary_singular_subdivision } [OF \text{ } scen])$
qed $(\text{auto simp: } scp1)$
then have $\text{singular_chain } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } T) \text{ (chain_boundary } p$
 $((\text{singular_subdivision } p \text{ } \hat{\sim} n) \text{ } c - e))$
by $(\text{simp add: chain_boundary_diff singular_chain_diff singular_chain_boundary } e)$
with de **have** $scTd: \text{singular_chain } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } T) \text{ (chain_boundary } p$
 $d)$
by *simp*
show *thesis*
proof
have $\text{singular_chain } (p - \text{Suc } 0) \text{ } X \text{ (chain_boundary } p \text{ } d)$
using $scTd \text{ singular_chain_subtopology }$ **by** *blast*
with $scSud \text{ } scTd$ **have** $\text{singular_chain } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } (T - U))$
 $(\text{chain_boundary } p \text{ } d)$

```

    by (fastforce simp add: singular_chain_subtopology)
  then show singular_relcycle p (subtopology X (S - U)) (T - U) d
    by (auto simp: singular_relcycle_def mod_subset_def subtopology_subtopology
d)
  have homologous_rel p (subtopology X S) T (c-0) ((singular_subdivision p ^~
n) c - e)
  proof (rule homologous_rel_diff)
    show homologous_rel p (subtopology X S) T c ((singular_subdivision p ^~ n)
c)
    proof (induction n)
      case (Suc n)
      then show ?case
        apply simp
        by (metis c homologous_rel_eq homologous_rel_singular_relcycle_1
homologous_rel_singular_subdivision)
    qed auto
    show homologous_rel p (subtopology X S) T 0 e
      unfolding homologous_rel_def using e
      by (intro singular_relboundary_diff singular_chain_imp_relboundary; simp
add: subtopology_subtopology)
    qed
    with de show homologous_rel p (subtopology X S) T c d
      by simp
    qed
  qed

```

0.1.18 Homotopy invariance

theorem *homotopic_imp_homologous_rel_chain_maps*:
assumes *hom*: *homotopic_with* $(\lambda h. h \text{ ` } T \subseteq V) S U f g$ **and** *c*: *singular_relcycle* *p S T c*
shows *homologous_rel* *p U V* (*chain_map* *p f c*) (*chain_map* *p g c*)
proof –
note *sum.atMost_Suc* [*simp del*]
have *contf*: *continuous_map* *S U f* **and** *contg*: *continuous_map* *S U g*
using *homotopic_with_imp_continuous_maps* [*OF hom*] **by** *metis*+
obtain *h* **where** *conth*: *continuous_map* (*prod_topology* (*top_of_set* {*0..1::real*})
S) *U h*
and *h0*: $\bigwedge x. h(0, x) = f\ x$
and *h1*: $\bigwedge x. h(1, x) = g\ x$
and *hV*: $\bigwedge t\ x. [0 \leq t; t \leq 1; x \in T] \implies h(t, x) \in V$
using *hom* **by** (*fastforce simp: homotopic_with_def*)
define *vv* **where** *vv* $\equiv \lambda j\ i. \text{if } i = \text{Suc } j \text{ then } 1 \text{ else } (0::\text{real})$
define *ww* **where** *ww* $\equiv \lambda j\ i. \text{if } i=0 \vee i = \text{Suc } j \text{ then } 1 \text{ else } (0::\text{real})$
define *simp* **where** *simp* $\equiv \lambda q\ i. \text{oriented_simplex } (\text{Suc } q) (\lambda j. \text{if } j \leq i \text{ then } vv$
j **else** *ww*(*j* - 1))
define *pr* **where *pr* $\equiv \lambda q\ c. \sum_{i \leq q} \text{frag_cmul } ((-1) ^ i$
 $(\text{frag_of } (\text{simplex_map } (\text{Suc } q) (\lambda z. h(z\ 0, c(z \circ$
Suc))) (*simp* *q* i)))**

```

have ss_ss: simplicial_simplex (Suc q) ( $\{x. x \ 0 \in \{0..1\} \wedge (x \circ \text{Suc}) \in \text{standard\_simplex } q\}$ ) (simp q i)
if  $i \leq q$  for  $q \ i$ 
proof -
have ( $\sum j \leq \text{Suc } q. (\text{if } j \leq i \text{ then } vv \ j \ 0 \text{ else } ww \ (j - 1) \ 0) * x \ j$ )  $\in \{0..1\}$ 
if  $x \in \text{standard\_simplex } (\text{Suc } q)$  for  $x$ 
proof -
have ( $\sum j \leq \text{Suc } q. \text{if } j \leq i \text{ then } 0 \text{ else } x \ j$ )  $\leq \text{sum } x \ \{\dots \text{Suc } q\}$ 
using that unfolding standard_simplex_def
by (force intro!: sum_mono)
with  $\langle i \leq q \rangle$  that show ?thesis
by (simp add: vv_def ww_def standard_simplex_def if_distrib [of  $\lambda u. u *$ 
_] sum_nonneg cong: if_cong)
qed
moreover
have ( $\lambda k. \sum j \leq \text{Suc } q. (\text{if } j \leq i \text{ then } vv \ j \ k \text{ else } ww \ (j - 1) \ k) * x \ j$ )  $\circ \text{Suc} \in$ 
standard_simplex  $q$ 
if  $x \in \text{standard\_simplex } (\text{Suc } q)$  for  $x$ 
proof -
have card: ( $\{\dots q\} \cap \{k. \text{Suc } k = j\}$ ) =  $\{j-1\}$  if  $0 < j \ j \leq \text{Suc } q$  for  $j$ 
using that by auto
have eq: ( $\sum j \leq \text{Suc } q. \sum k \leq q. \text{if } j \leq i \text{ then if } k = j \text{ then } x \ j \text{ else } 0 \text{ else if } \text{Suc } k = j \text{ then } x \ j \text{ else } 0$ )
= ( $\sum j \leq \text{Suc } q. x \ j$ )
by (rule sum.cong [OF refl]) (use  $\langle i \leq q \rangle$  in  $\langle \text{simp add: sum.If_cases card} \rangle$ )
have ( $\sum j \leq \text{Suc } q. \text{if } j \leq i \text{ then if } k = j \text{ then } x \ j \text{ else } 0 \text{ else if } \text{Suc } k = j \text{ then } x \ j \text{ else } 0$ )
 $\leq \text{sum } x \ \{\dots \text{Suc } q\}$  for  $k$ 
using that unfolding standard_simplex_def
by (force intro!: sum_mono)
then show ?thesis
using  $\langle i \leq q \rangle$  that
by (simp add: vv_def ww_def standard_simplex_def if_distrib [of  $\lambda u. u *$ 
_] sum_nonneg
sum.swap [where  $A = \text{atMost } q$ ] eq cong: if_cong)
qed
ultimately show ?thesis
by (simp add: that simplicial_simplex_oriented_simplex simp_def image_subset_iff
if_distribR)
qed
obtain prism where prism:  $\bigwedge q. \text{prism } q \ 0 = 0$ 
 $\bigwedge q \ c. \text{singular\_chain } q \ S \ c \implies \text{singular\_chain } (\text{Suc } q) \ U \ (\text{prism } q \ c)$ 
 $\bigwedge q \ c. \text{singular\_chain } q \ (\text{subtopology } S \ T) \ c$ 
 $\implies \text{singular\_chain } (\text{Suc } q) \ (\text{subtopology } U \ V) \ (\text{prism } q \ c)$ 
 $\bigwedge q \ c. \text{singular\_chain } q \ S \ c$ 
 $\implies \text{chain\_boundary } (\text{Suc } q) \ (\text{prism } q \ c) =$ 
 $\text{chain\_map } q \ g \ c - \text{chain\_map } q \ f \ c - \text{prism } (q - 1)$ 
(chain_boundary  $q \ c$ )
proof

```

```

show (frag_extend ∘ pr) q 0 = 0 for q
  by (simp add: pr_def)
next
show singular_chain (Suc q) U ((frag_extend ∘ pr) q c)
  if singular_chain q S c for q c
  using that [unfolded singular_chain_def]
proof (induction c rule: frag_induction)
  case (one m)
  show ?case
  proof (simp add: pr_def, intro singular_chain_cmul singular_chain_sum)
    fix i :: nat
    assume i ∈ {..q}
    define X where X = subtopology (powertop_real UNIV) {x. x 0 ∈ {0..1}}
    ∧ (x ∘ Suc) ∈ standard_simplex q}
    show singular_chain (Suc q) U
      (frag_of (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc)))) (simp q
i)))
    unfolding singular_chain_of
  proof (rule singular_simplex_simplex_map)
    show singular_simplex (Suc q) X (simp q i)
      unfolding X_def using ⟨i ∈ {..q}⟩ simplicial_imp_singular_simplex
ss_ss by blast
    have 0: continuous_map X (top_of_set {0..1}) (λx. x 0)
      unfolding continuous_map_in_subtopology topspace_subtopology X_def
    by (auto intro: continuous_map_product_projection continuous_map_from_subtopology)
    have 1: continuous_map X S (m ∘ (λx j. x (Suc j)))
      proof (rule continuous_map_compose)
        have continuous_map (powertop_real UNIV) (powertop_real UNIV)
(λx j. x (Suc j))
        by (auto intro: continuous_map_product_projection)
        then show continuous_map X (subtopology (powertop_real UNIV)
(standard_simplex q)) (λx j. x (Suc j))
          unfolding X_def o_def
          by (auto simp: continuous_map_in_subtopology intro: continu-
ous_map_from_subtopology continuous_map_product_projection)
      qed (use one in ⟨simp add: singular_simplex_def⟩)
    show continuous_map X U (λz. h (z 0, m (z ∘ Suc)))
      apply (rule continuous_map_compose [unfolded o_def, OF _ conth])
      using 0 1 by (simp add: continuous_map_pairwise o_def)
    qed
  qed
next
case (diff a b)
then show ?case
  by (simp add: frag_extend_diff singular_chain_diff)
qed auto
next
show singular_chain (Suc q) (subtopology U V) ((frag_extend ∘ pr) q c)
  if singular_chain q (subtopology S T) c for q c

```

```

    using that [unfolded singular_chain_def]
  proof (induction c rule: frag_induction)
    case (one m)
    show ?case
    proof (simp add: pr_def, intro singular_chain_cmul singular_chain_sum)
      fix i :: nat
      assume i ∈ {..q}
      define X where X = subtopology (powertop_real UNIV) {x. x 0 ∈ {0..1}
    ∧ (x 0 Suc) ∈ standard_simplex q}
      show singular_chain (Suc q) (subtopology U V)
        (frag_of (simplex_map (Suc q) (λz. h (z 0, m (z 0 Suc)))) (simp q
    i)))
      unfolding singular_chain_of
    proof (rule singular_simplex_simplex_map)
      show singular_simplex (Suc q) X (simp q i)
        unfolding X_def using ⟨i ∈ {..q}⟩ simplicial_imp_singular_simplex
    ss_ss by blast
      have 0: continuous_map X (top_of_set {0..1}) (λx. x 0)
        unfolding continuous_map_in_subtopology topspace_subtopology X_def
      by (auto intro: continuous_map_product_projection continuous_map_from_subtopology)
      have 1: continuous_map X (subtopology S T) (m 0 (λx j. x (Suc j)))
        proof (rule continuous_map_compose)
          have continuous_map (powertop_real UNIV) (powertop_real UNIV)
    (λx j. x (Suc j))
            by (auto intro: continuous_map_product_projection)
          then show continuous_map X (subtopology (powertop_real UNIV)
    (standard_simplex q)) (λx j. x (Suc j))
            unfolding X_def o_def
              by (auto simp: continuous_map_in_subtopology intro: continu-
    ous_map_from_subtopology continuous_map_product_projection)
          show continuous_map (subtopology (powertop_real UNIV) (standard_simplex
    q)) (subtopology S T) m
            using one continuous_map_into_fulltopology by (auto simp: singu-
    lar_simplex_def)
        qed
      have continuous_map X (subtopology U V) (h 0 (λz. (z 0, m (z 0 Suc))))
        proof (rule continuous_map_compose)
          show continuous_map X (prod_topology (top_of_set {0..1::real})
    (subtopology S T)) (λz. (z 0, m (z 0 Suc)))
            using 0 1 by (simp add: continuous_map_pairwise o_def)
          have continuous_map (subtopology (prod_topology euclideanreal S)
    ({0..1} × T)) U h
            by (metis conth continuous_map_from_subtopology subtopology_Times
    subtopology_topospace)
          with hV show continuous_map (prod_topology (top_of_set {0..1::real})
    (subtopology S T)) (subtopology U V) h
            by (force simp: topspace_subtopology continuous_map_in_subtopology
    subtopology_restrict subtopology_Times)
        qed
    qed
  qed

```

```

      then show continuous_map X (subtopology U V) (λz. h (z 0, m (z ∘
Suc)))
    by (simp add: o_def)
  qed
  qed
next
  case (diff a b)
  then show ?case
    by (metis comp_apply frag_extend_diff singular_chain_diff)
  qed auto
next
  show chain_boundary (Suc q) ((frag_extend ∘ pr) q c) =
    chain_map q g c - chain_map q f c - (frag_extend ∘ pr) (q - 1)
(chain_boundary q c)
  if singular_chain q S c for q c
  using that [unfolded singular_chain_def]
  proof (induction c rule: frag_induction)
    case (one m)
    have eq2: Sigma S T = (λi. (i, i)) ‘ {i ∈ S. i ∈ T i} ∪ (Sigma S (λi. T i -
{i})) for S :: nat set and T
    by force
    have 1: (∑ (i,j) ∈ (λi. (i, i)) ‘ {i. i ≤ q ∧ i ≤ Suc q}.
      frag_cmul (((-1) ^ i) * (-1) ^ j)
      (frag_of
        (singular_face (Suc q) j
          (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q i))))))
      + (∑ (i,j) ∈ (λi. (i, i)) ‘ {i. i ≤ q}.
      frag_cmul (- ((-1) ^ i * (-1) ^ j))
      (frag_of
        (singular_face (Suc q) (Suc j)
          (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q
i))))))
      = frag_of (simplex_map q g m) - frag_of (simplex_map q f m)
    proof -
      have restrict ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q 0 ∘ simplicial_face 0))
(standard_simplex q)
      = restrict (g ∘ m) (standard_simplex q)
    proof (rule restrict_ext)
      fix x
      assume x: x ∈ standard_simplex q
      have (∑ j ≤ Suc q. if j = 0 then 0 else x (j - Suc 0)) = (∑ j ≤ q. x j)
      by (simp add: sum.atMost_Suc_shift)
      with x have simp q 0 (simplicial_face 0 x) 0 = 1
    apply (simp add: oriented_simplex_def simp_def simplicial_face_in_standard_simplex)
    apply (simp add: simplicial_face_def if_distrib ww_def standard_simplex_def
cong: if_cong)
    done
  moreover
  have (λn. if n ≤ q then x n else 0) = x

```



```

    using standard_simplex_def x by auto
  then have (λn. simp q 0 (simplicial_face 0 x) (Suc n)) = x
    unfolding oriented_simplex_def simp_def ww_def using x
    apply (simp add: simplicial_face_in_standard_simplex)
    apply (simp add: simplicial_face_def if_distrib)
    apply (simp add: if_distribR if_distrib cong: if_cong)
  done
  ultimately show ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q 0 ∘ simplicial_face
0)) x = (g ∘ m) x
    by (simp add: o_def h1)
  qed
  then have a: frag_of (singular_face (Suc q) 0 (simplex_map (Suc q) (λz.
h (z 0, m (z ∘ Suc))) (simp q 0)))
    = frag_of (simplex_map q g m)
    by (simp add: singular_face_simplex_map) (simp add: simplex_map_def)
  have restrict ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q q ∘ simplicial_face (Suc
q))) (standard_simplex q)
    = restrict (f ∘ m) (standard_simplex q)
  proof (rule restrict_ext)
    fix x
    assume x: x ∈ standard_simplex q
    then have simp q q (simplicial_face (Suc q) x) 0 = 0
      unfolding oriented_simplex_def simp_def
      by (simp add: simplicial_face_in_standard_simplex sum.atMost_Suc)
    (simp add: simplicial_face_def vv_def)
    moreover have (λn. simp q q (simplicial_face (Suc q) x) (Suc n)) = x
      unfolding oriented_simplex_def simp_def vv_def using x
      apply (simp add: simplicial_face_in_standard_simplex)
      apply (force simp: standard_simplex_def simplicial_face_def if_distribR
if_distrib [of λx. x * _] sum.atMost_Suc cong: if_cong)
    done
    ultimately show ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q q ∘ simplicial_face
(Suc q))) x = (f ∘ m) x
      by (simp add: o_def h0)
  qed
  then have b: frag_of (singular_face (Suc q) (Suc q)
    (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q q)))
    = frag_of (simplex_map q f m)
  by (simp add: singular_face_simplex_map) (simp add: simplex_map_def)
  have sfq: simplex_map q (λz. h (z 0, m (z ∘ Suc))) (simp q (Suc i) ∘
simplicial_face (Suc i))
    = simplex_map q (λz. h (z 0, m (z ∘ Suc))) (simp q i ∘ simplicial_face
(Suc i))
  if i < q for i
  unfolding simplex_map_def
  proof (rule restrict_ext)
    fix x
    assume x ∈ standard_simplex q
    then have (simp q (Suc i) ∘ simplicial_face (Suc i)) x = (simp q i ∘

```

```

simplical_face (Suc i)) x
  unfolding oriented_simplex_def simp_def simplical_face_def
  by (force intro: sum.cong)
  then show ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q (Suc i) ∘ simplical_face
(Suc i))) x
    = ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q i ∘ simplical_face (Suc i))) x
  by simp
qed
have eqq: {i. i ≤ q ∧ i ≤ Suc q} = {..q}
  by force
have qeq: {..q} = insert 0 ((λi. Suc i) ‘ {i. i < q}) {i. i ≤ q} = insert q
{i. i < q}
  using le_imp_less_Suc less_Suc_eq_0_disj by auto
show ?thesis
  using a b
  apply (simp add: sum.reindex inj_on_def eqq)
  apply (simp add: qeq sum.insert_if sum.reindex sum_negf singular_face_simplex_map
sfeq)
  done
qed
have 2: (∑ (i,j)∈(SIGMA i:{..q}. {0..min (Suc q) i} - {i}).
  frag_cmul ((-1) ^ i * (-1) ^ j)
  (frag_of
    (singular_face (Suc q) j
      (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q i)))))
+ (∑ (i,j)∈(SIGMA i:{..q}. {i..q} - {i}).
  frag_cmul (- ((-1) ^ i * (-1) ^ j))
  (frag_of
    (singular_face (Suc q) (Suc j)
      (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q i)))))
= - frag_extend (pr (q - Suc 0)) (chain_boundary q (frag_of m))
proof (cases q=0)
case True
  then show ?thesis
    by (simp add: chain_boundary_def flip: sum.Sigma)
next
case False
  have eq: {..q - Suc 0} × {..q} = Sigma {..q - Suc 0} (λi. {0..min q i})
  ∪ Sigma {..q} (λi. {i<..q})
  by force
  have I: (∑ (i,j)∈(SIGMA i:{..q}. {0..min (Suc q) i} - {i}).
    frag_cmul ((-1) ^ (i + j))
    (frag_of
      (singular_face (Suc q) j
        (simplex_map (Suc q) (λz. h (z 0, m (z ∘ Suc))) (simp q i)))))
  = (∑ (i,j)∈(SIGMA i:{..q - Suc 0}. {0..min q i}).
    frag_cmul (- ((-1) ^ (j + i)))
    (frag_of
      (simplex_map q (λz. h (z 0, singular_face q j m (z ∘ Suc)))))

```

```

      (simp (q - Suc 0) i)))
proof -
  have seq: simplex_map q (λz. h (z 0, singular_face q j m (z ∘ Suc)))
    (simp (q - Suc 0) (i - Suc 0))
    = simplex_map q (λz. h (z 0, m (z ∘ Suc))) (simp q i ∘ simplicial_face
j)
    if ij: i ≤ q j ≠ i j ≤ i for i j
    unfolding simplex_map_def
  proof (rule restrict_ext)
    fix x
    assume x: x ∈ standard_simplex q
    have i > 0
      using that by force
    then have iq: i - Suc 0 ≤ q - Suc 0
      using ⟨i ≤ q⟩ False by simp
    have q0_eq: {..Suc q} = insert 0 (Suc ‘ {..q})
      by (auto simp: image_def gr0_conv_Suc)
    have α: simp (q - Suc 0) (i - Suc 0) x 0 = simp q i (simplicial_face j
x) 0
      using False x ij
      unfolding oriented_simplex_def simp_def vv_def ww_def
      apply (simp add: simplicial_face_in_standard_simplex)
      apply (force simp: simplicial_face_def q0_eq sum.reindex intro!:
sum.cong)
      done
    have β: simplicial_face j (simp (q - Suc 0) (i - Suc 0) x ∘ Suc) = simp
q i (simplicial_face j x) ∘ Suc
      proof
        fix k
        show simplicial_face j (simp (q - Suc 0) (i - Suc 0) x ∘ Suc) k
          = (simp q i (simplicial_face j x) ∘ Suc) k
          using False x ij
          unfolding oriented_simplex_def simp_def o_def vv_def ww_def
          apply (simp add: simplicial_face_in_standard_simplex if_distribR)
          apply (simp add: simplicial_face_def if_distrib [of λu. u * _] cong:
if_cong)
          apply (intro impI conjI)
          apply (force simp: sum.atMost_Suc intro: sum.cong)
          apply (force simp: q0_eq sum.reindex intro!: sum.cong)
          done
      qed
    have simp (q - Suc 0) (i - Suc 0) x ∘ Suc ∈ standard_simplex (q -
Suc 0)
      using ss_ss [OF iq] ⟨i ≤ q⟩ False ⟨i > 0⟩
      by (simp add: image_subset_iff simplicial_simplex x)
    then show ((λz. h (z 0, singular_face q j m (z ∘ Suc))) ∘ simp (q -
Suc 0) (i - Suc 0)) x
      = ((λz. h (z 0, m (z ∘ Suc))) ∘ (simp q i ∘ simplicial_face j)) x
      by (simp add: singular_face_def α β)

```

```

    qed
    have [simp]:  $(-1::int) \wedge (i + j - \text{Suc } 0) = -((-1) \wedge (i + j))$  if  $i \neq j$ 
  for  $i\ j::nat$ 
    proof -
      have  $i + j > 0$ 
      using that by blast
      then show ?thesis
      by (metis (no_types, opaque_lifting) One_nat_def Suc_diff_1
        add.inverse_inverse mult.left_neutral mult.minus_left power_Suc)
    qed
    show ?thesis
    apply (rule sum.eq_general_inverses [where  $h = \lambda(a,b). (a-1,b)$  and
 $k = \lambda(a,b). (\text{Suc } a,b)$ ])
    using False apply (auto simp: singular_face_simplex_map seq add.commute)
    done
  qed
  have *:  $\text{singular\_face } (\text{Suc } q) (\text{Suc } j) (\text{simplex\_map } (\text{Suc } q) (\lambda z. h (z\ 0, m (z \circ \text{Suc}))) (\text{simp } q\ i)))$ 
     $= \text{simplex\_map } q (\lambda z. h (z\ 0, \text{singular\_face } q\ j\ m (z \circ \text{Suc}))) (\text{simp } (q - \text{Suc } 0)\ i)$ 
    if  $ij: i < j \leq q$  for  $i\ j$ 
  proof -
    have  $iq: i \leq q - \text{Suc } 0$ 
    using that by auto
    have  $sf\_eqh: \text{singular\_face } (\text{Suc } q) (\text{Suc } j)$ 
       $(\lambda x. \text{if } x \in \text{standard\_simplex } (\text{Suc } q)$ 
         $\text{then } ((\lambda z. h (z\ 0, m (z \circ \text{Suc}))) \circ \text{simp } q\ i)\ x \text{ else }$ 
         $\text{undefined})\ x$ 
       $= h (\text{simp } (q - \text{Suc } 0)\ i\ x\ 0,$ 
         $\text{singular\_face } q\ j\ m (\lambda xa. \text{simp } (q - \text{Suc } 0)\ i\ x (\text{Suc } xa)))$ 
    if  $x: x \in \text{standard\_simplex } q$  for  $x$ 
  proof -
    let  $?f = \lambda k. \sum_{j \leq q} \text{if } j \leq i \text{ then if } k = j \text{ then } x\ j \text{ else } 0$ 
       $\text{else if } \text{Suc } k = j \text{ then } x\ j \text{ else } 0$ 
    have  $fm: \text{simplicial\_face } (\text{Suc } j)\ x \in \text{standard\_simplex } (\text{Suc } q)$ 
    using  $ss\_ss [OF\ iq]$  that  $ij$ 
    by (simp add: simplicial_face_in_standard_simplex)
    have  $ss: ?f \in \text{standard\_simplex } (q - \text{Suc } 0)$ 
    unfolding standard_simplex_def
    proof (intro CollectI conjI impI allI)
      fix  $k$ 
      show  $0 \leq ?f\ k$ 
      using that by (simp add: sum_nonneg standard_simplex_def)
      show  $?f\ k \leq 1$ 
      using  $x\ \text{sum\_le\_included } [of\ \{..q\}\ \{..q\}\ x\ id]$ 
      by (simp add: standard_simplex_def)
      assume  $k: q - \text{Suc } 0 < k$ 
      show  $?f\ k = 0$ 
      by (rule sum.neutral) (use that  $x\ iq\ k\ \text{standard\_simplex\_def}$  in auto)
    qed
  qed

```

```

next
  have ( $\sum k \leq q - \text{Suc } 0. ?f k$ )
    = ( $\sum (k,j) \in (\{..q - \text{Suc } 0\} \times \{..q\}) \cap \{(k,j). \text{ if } j \leq i \text{ then } k = j$ 
else  $\text{Suc } k = j\}. x j$ )
  apply (simp add: sum.Sigma)
  by (rule sum.mono_neutral_cong) (auto simp: split: if_split_asm)
  also have  $\dots = \text{sum } x \{..q\}$ 
  apply (rule sum.eq_general_inverses)
  [where  $h = \lambda(k,j). \text{ if } j \leq i \wedge k=j \vee j > i \wedge \text{Suc } k = j \text{ then } j \text{ else Suc}$ 
 $q$ 
    and  $k = \lambda j. \text{ if } j \leq i \text{ then } (j,j) \text{ else } (j - \text{Suc } 0, j)$ ]
  using ij by auto
  also have  $\dots = 1$ 
  using x by (simp add: standard_simplex_def)
  finally show ( $\sum k \leq q - \text{Suc } 0. ?f k$ ) = 1
  by (simp add: standard_simplex_def)
qed
let ?g =  $\lambda k. \text{ if } k \leq i \text{ then } 0$ 
    else if  $k < \text{Suc } j \text{ then } x k$ 
    else if  $k = \text{Suc } j \text{ then } 0 \text{ else } x (k - \text{Suc } 0)$ 
  have eq:  $\{.. \text{Suc } q\} = \{..j\} \cup \{\text{Suc } j\} \cup \text{Suc}\{j < ..q\} \{..q\} = \{..j\} \cup$ 
 $\{j < ..q\}$ 
  using ij image_iff less_Suc_eq_0_disj less_Suc_eq_le
  by (force simp: image_iff)+
  then have ( $\sum k \leq \text{Suc } q. ?g k$ ) = ( $\sum k \in \{..j\} \cup \{\text{Suc } j\} \cup \text{Suc}\{j < ..q\}. ?g k$ )
 $?g k$ )
  by simp
  also have  $\dots = (\sum k \in \{..j\} \cup \text{Suc}\{j < ..q\}. ?g k)$ 
  by (rule sum.mono_neutral_right) auto
  also have  $\dots = (\sum k \in \{..j\}. ?g k) + (\sum k \in \text{Suc}\{j < ..q\}. ?g k)$ 
  by (rule sum.union_disjoint) auto
  also have  $\dots = (\sum k \in \{..j\}. ?g k) + (\sum k \in \{j < ..q\}. ?g (\text{Suc } k))$ 
  by (auto simp: sum.reindex)
  also have  $\dots = (\sum k \in \{..j\}. \text{ if } k \leq i \text{ then } 0 \text{ else } x k)$ 
    + ( $\sum k \in \{j < ..q\}. \text{ if } k \leq i \text{ then } 0 \text{ else } x k$ )
  by (intro sum.cong arg_cong2 [of concl: (+)]) (use ij in auto)
  also have  $\dots = (\sum k \leq q. \text{ if } k \leq i \text{ then } 0 \text{ else } x k)$ 
  unfolding eq by (subst sum.union_disjoint) auto
  finally have ( $\sum k \leq \text{Suc } q. ?g k$ ) = ( $\sum k \leq q. \text{ if } k \leq i \text{ then } 0 \text{ else } x k$ ) .
  then have QQ: ( $\sum l \leq \text{Suc } q. \text{ if } l \leq i \text{ then } 0 \text{ else simplicial\_face } (\text{Suc } j)$ 
 $x l$ ) = ( $\sum j \leq q. \text{ if } j \leq i \text{ then } 0 \text{ else } x j$ )
  by (simp add: simplicial_face_def cong: if_cong)
  have WW: ( $\lambda k. \sum l \leq \text{Suc } q. \text{ if } l \leq i$ 
    then if  $k = l \text{ then simplicial\_face } (\text{Suc } j) x l \text{ else } 0$ 
    else if  $\text{Suc } k = l \text{ then simplicial\_face } (\text{Suc } j) x l$ 
    else 0)
    = simplicial_face j
    ( $\lambda k. \sum j \leq q. \text{ if } j \leq i \text{ then if } k = j \text{ then } x j \text{ else } 0$ 
    else if  $\text{Suc } k = j \text{ then } x j \text{ else } 0$ )

```

```

proof -
  have *: ( $\sum l \leq q. \text{if } l \leq i \text{ then } 0 \text{ else if } \text{Suc } k = l \text{ then } x (l - \text{Suc } 0)$ 
else 0)
    = ( $\sum l \leq q. \text{if } l \leq i \text{ then if } k - \text{Suc } 0 = l \text{ then } x l \text{ else } 0 \text{ else if } k =$ 
l then x l else 0)
    (is ?lhs = ?rhs)
    if  $k \neq q$   $k > j$  for  $k$ 
proof (cases  $k \leq q$ )
  case True
    have ?lhs =  $\text{sum } (\lambda l. x (l - \text{Suc } 0)) \{ \text{Suc } k \}$  ?rhs =  $\text{sum } x \{ k \}$ 
    by (rule sum.mono_neutral_cong_right; use True ij that in auto)+
    then show ?thesis
    by simp
  next
    case False
    have ?lhs = 0 ?rhs = 0
    by (rule sum.neutral; use False ij in auto)+
    then show ?thesis
    by simp
qed
have xq:  $x q = (\sum j \leq q. \text{if } j \leq i \text{ then if } q - \text{Suc } 0 = j \text{ then } x j \text{ else } 0$ 
else if  $q = j$  then  $x j$  else 0) if  $q \neq j$ 
    using ij that
    by (force simp flip: ivl_disj_un(2) intro: sum.neutral)
show ?thesis
    using ij unfolding simplicial_face_def
    by (intro ext) (auto simp: * sum.atMost_Suc xq cong: if_cong)
qed
show ?thesis
    using False that iq
    unfolding oriented_simplex_def simp_def vv_def ww_def
    apply (simp add: if_distribR simplicial_face_def if_distrib [of  $\lambda u. u *$ 
_] o_def cong: if_cong)
    apply (simp add: singular_face_def fm ss QQ WW)
    done
qed
show ?thesis
    unfolding simplex_map_def restrict_def
    apply (simp add: simplicial_simplex_image_subset_iff o_def sf_eqh
fun_eq_iff)
    apply (simp add: singular_face_def)
    done
qed
have sgeq:  $(\text{SIGMA } i: \{..q\}. \{i..q\} - \{i\}) = (\text{SIGMA } i: \{..q\}. \{i <..q\})$ 
    by force
have II:  $(\sum (i,j) \in (\text{SIGMA } i: \{..q\}. \{i..q\} - \{i\}).$ 
frag_cmul  $(-((-1) ^ (i + j)))$ 
    (frag_of
    (singular_face (Suc q) (Suc j))

```

```

i)))) =
  (simplex_map (Suc q) (λz. h (z 0, m (z o Suc))) (simp q
  i)))) =
  (Σ (i,j) ∈ (SIGMA i: {..q}. {i <.. q}).
    frag_cmul (- ((-1) ^ (j + i)))
    (frag_of
      (simplex_map q (λz. h (z 0, singular_face q j m (z o Suc)))
      (simp (q - Suc 0) i))))
  by (force simp: * sgeq add.commute intro: sum.cong)
show ?thesis
using False
apply (simp add: chain_boundary_def frag_extend_sum frag_extend_cmul
frag_cmul_sum pr_def flip: sum_negf power_add)
  apply (subst sum.swap [where A = {..q}])
    apply (simp add: sum.cartesian_product eq sum.union_disjoint dis-
joint_iff_not_equal I II)
  done
qed
have *: [a+b = w; c+d = -z] ⇒ (a + c) + (b+d) = w-z for a b w c d z
:: 'c ⇒0 int
  by (auto simp: algebra_simps)
have eq: {..q} × {..Suc q} =
  Sigma {..q} (λi. {0..min (Suc q) i})
  ∪ Sigma {..q} (λi. {Suc i..Suc q})
  by force
show ?case
  apply (subst pr_def)
  apply (simp add: chain_boundary_sum chain_boundary_cmul)
  apply (subst chain_boundary_def)
  apply simp
  apply (simp add: frag_cmul_sum sum.cartesian_product eq sum.union_disjoint
disjoint_iff_not_equal
    sum.atLeast_Suc_atMost_Suc_shift del: sum.cl_ivl_Suc min.absorb2
min.absorb4
    flip: comm_monoid_add_class.sum.Sigma)
  apply (simp add: sum.Sigma eq2 [of _ λi. {_ i.._ i}])
    del: min.absorb2 min.absorb4)
  apply (simp add: sum.union_disjoint disjoint_iff_not_equal * [OF 1 2])
  done
next
case (diff a b)
then show ?case
  by (simp add: chain_boundary_diff frag_extend_diff chain_map_diff)
qed auto
qed
have *: singular_chain p (subtopology U V) (prism (p - Suc 0) (chain_boundary
p c))
  if singular_chain p S c singular_chain (p - Suc 0) (subtopology S T) (chain_boundary
p c)
  proof (cases p)

```

```

    case 0 then show ?thesis by (simp add: chain_boundary_def prism)
  next
    case (Suc p')
    with prism that show ?thesis by auto
  qed
  then show ?thesis
    using c
    unfolding singular_relcycle_def homologous_rel_def singular_relboundary_def
    mod_subset_def
    apply (rule_tac x=- prism p c in exI)
    by (simp add: chain_boundary_minus prism(2) prism(4) singular_chain_minus)
  qed
end

```

0.2 Homology, II: Homology Groups

```

theory Homology_Groups
  imports Simplicies HOL-Algebra.Exact_Sequence

```

```
begin
```

0.2.1 Homology Groups

Now actually connect to group theory and set up homology groups. Note that we define homomogy groups for all *integers* p , since this seems to avoid some special-case reasoning, though they are trivial for $p < 0$.

```

definition chain_group :: nat  $\Rightarrow$  'a topology  $\Rightarrow$  'a chain monoid
  where chain_group p X  $\equiv$  free_Abelian_group (singular_simplex_set p X)

```

```

lemma carrier_chain_group [simp]: carrier(chain_group p X) = singular_chain_set
p X
  by (auto simp: chain_group_def singular_chain_def free_Abelian_group_def)

```

```

lemma one_chain_group [simp]: one(chain_group p X) = 0
  by (auto simp: chain_group_def free_Abelian_group_def)

```

```

lemma mult_chain_group [simp]: monoid.mult(chain_group p X) = (+)
  by (auto simp: chain_group_def free_Abelian_group_def)

```

```

lemma m_inv_chain_group [simp]: Poly_Mapping.keys a  $\subseteq$  singular_simplex_set
p X  $\implies$  inv chain_group p X a = -a
  unfolding chain_group_def by simp

```

```

lemma group_chain_group [simp]: Group.group (chain_group p X)
  by (simp add: chain_group_def)

```

```

lemma abelian_chain_group: comm_group(chain_group p X)

```


by (simp add: free_Abelian_group_def group.group_comm_groupI [OF group_chain_group])

lemma subgroup_singular_relcycle:

subgroup (singular_relcycle_set p X S) (chain_group p X)

proof

show $x \otimes_{\text{chain_group } p \ X} y \in \text{singular_relcycle_set } p \ X \ S$

if $x \in \text{singular_relcycle_set } p \ X \ S$ and $y \in \text{singular_relcycle_set } p \ X \ S$ for x

y

using that by (simp add: singular_relcycle_add)

next

show $\text{inv}_{\text{chain_group } p \ X} x \in \text{singular_relcycle_set } p \ X \ S$

if $x \in \text{singular_relcycle_set } p \ X \ S$ for x

using that

by clarsimp (metis m_inv_chain_group singular_chain_def singular_relcycle
singular_relcycle_minus)

qed (auto simp: singular_relcycle)

definition relcycle_group :: nat \Rightarrow 'a topology \Rightarrow 'a set \Rightarrow ('a chain) monoid

where relcycle_group p X S \equiv

subgroup_generated (chain_group p X) (Collect(singular_relcycle p X S))

lemma carrier_relcycle_group [simp]:

carrier (relcycle_group p X S) = singular_relcycle_set p X S

proof –

have carrier (chain_group p X) \cap singular_relcycle_set p X S = singular_relcycle_set
p X S

using subgroup.subset subgroup_singular_relcycle by blast

moreover have generate (chain_group p X) (singular_relcycle_set p X S) \subseteq
singular_relcycle_set p X S

by (simp add: group.generate_subgroup_incl group_chain_group subgroup_singular_relcycle)

ultimately show ?thesis

by (auto simp: relcycle_group_def subgroup_generated_def generate.incl)

qed

lemma one_relcycle_group [simp]: one(relcycle_group p X S) = 0

by (simp add: relcycle_group_def)

lemma mult_relcycle_group [simp]: $(\otimes_{\text{relcycle_group } p \ X \ S}) = (+)$

by (simp add: relcycle_group_def)

lemma abelian_relcycle_group [simp]:

comm_group(relcycle_group p X S)

unfolding relcycle_group_def

by (intro group.abelian_subgroup_generated group_chain_group) (auto simp:
abelian_chain_group singular_relcycle)

lemma group_relcycle_group [simp]: group(relcycle_group p X S)

by (simp add: comm_group.axioms(2))

lemma *relcycle_group_restrict* [simp]:
 $\text{relcycle_group } p \ X \ (\text{topspace } X \cap S) = \text{relcycle_group } p \ X \ S$
by (metis *relcycle_group_def singular_relcycle_restrict*)

definition *relative_homology_group* :: $\text{int} \Rightarrow 'a \text{ topology} \Rightarrow 'a \text{ set} \Rightarrow ('a \text{ chain})$
 set monoid

where
 $\text{relative_homology_group } p \ X \ S \equiv$
 if $p < 0$ then *singleton_group undefined* else
 $(\text{relcycle_group } (\text{nat } p) \ X \ S) \text{ Mod } (\text{singular_relboundary_set } (\text{nat } p) \ X \ S)$

abbreviation *homology_group*
where $\text{homology_group } p \ X \equiv \text{relative_homology_group } p \ X \ \{\}$

lemma *relative_homology_group_restrict* [simp]:
 $\text{relative_homology_group } p \ X \ (\text{topspace } X \cap S) = \text{relative_homology_group } p \ X \ S$
by (simp add: *relative_homology_group_def*)

lemma *nontrivial_relative_homology_group*:
fixes $p::\text{nat}$
shows $\text{relative_homology_group } p \ X \ S$
 $= \text{relcycle_group } p \ X \ S \text{ Mod } \text{singular_relboundary_set } p \ X \ S$
by (simp add: *relative_homology_group_def*)

lemma *singular_relboundary_ss*:
 $\text{singular_relboundary } p \ X \ S \ x \implies \text{Poly_Mapping.keys } x \subseteq \text{singular_simplex_set } p \ X$
using *singular_chain_def singular_relboundary_imp_chain* **by** blast

lemma *trivial_relative_homology_group* [simp]:
 $p < 0 \implies \text{trivial_group}(\text{relative_homology_group } p \ X \ S)$
by (simp add: *relative_homology_group_def*)

lemma *subgroup_singular_relboundary*:
 $\text{subgroup } (\text{singular_relboundary_set } p \ X \ S) \ (\text{chain_group } p \ X)$
unfolding *chain_group_def*

proof *unfold_locales*
show $\text{singular_relboundary_set } p \ X \ S$
 $\subseteq \text{carrier } (\text{free_Abelian_group } (\text{singular_simplex_set } p \ X))$
using *singular_chain_def singular_relboundary_imp_chain* **by** fastforce

next

fix x
assume $x \in \text{singular_relboundary_set } p \ X \ S$
then show $\text{invfree_Abelian_group } (\text{singular_simplex_set } p \ X) \ x$
 $\in \text{singular_relboundary_set } p \ X \ S$
by (simp add: *singular_relboundary_ss singular_relboundary_minus*)

qed (auto simp: free_Abelian_group_def singular_relboundary_add)

lemma subgroup_singular_relboundary_relcycle:

subgroup (singular_relboundary_set p X S) (relcycle_group p X S)

unfolding relcycle_group_def

by (simp add: Collect_mono group.subgroup_of_subgroup_generated singular_relboundary_imp_relcycle subgroup_singular_relboundary)

lemma normal_subgroup_singular_relboundary_relcycle:

(singular_relboundary_set p X S) \triangleleft (relcycle_group p X S)

by (simp add: comm_group.normal_iff_subgroup subgroup_singular_relboundary_relcycle)

lemma group_relative_homology_group [simp]:

group (relative_homology_group p X S)

by (simp add: relative_homology_group_def normal.factorgroup_is_group normal_subgroup_singular_relboundary_relcycle)

lemma right_coset_singular_relboundary:

r_coset (relcycle_group p X S) (singular_relboundary_set p X S)

= ($\lambda a. \{b. \text{homologous_rel } p \ X \ S \ a \ b\}$)

using singular_relboundary_minus

by (force simp: r_coset_def homologous_rel_def relcycle_group_def subgroup_generated_def)

lemma carrier_relative_homology_group:

carrier (relative_homology_group (int p) X S)

= (homologous_rel_set p X S) 'singular_relcycle_set p X S

by (auto simp: set_eq_iff image_iff relative_homology_group_def FactGroup_def RCOSETS_def right_coset_singular_relboundary)

lemma carrier_relative_homology_group_0:

carrier (relative_homology_group 0 X S)

= (homologous_rel_set 0 X S) 'singular_relcycle_set 0 X S

using carrier_relative_homology_group [of 0 X S] **by** simp

lemma one_relative_homology_group [simp]:

one (relative_homology_group (int p) X S) = singular_relboundary_set p X S

by (simp add: relative_homology_group_def FactGroup_def)

lemma mult_relative_homology_group:

($\otimes_{\text{relative_homology_group } (int \ p) \ X \ S}$) = ($\lambda R \ S. (\bigcup r \in R. \bigcup s \in S. \{r + s\})$)

unfolding relcycle_group_def subgroup_generated_def chain_group_def free_Abelian_group_def set_mult_def relative_homology_group_def FactGroup_def

by force

lemma inv_relative_homology_group:

assumes $R \in \text{carrier } (\text{relative_homology_group } (int \ p) \ X \ S)$

shows $m_inv(\text{relative_homology_group } (int \ p) \ X \ S) \ R = u_minus \ ' \ R$

proof (rule group.inv_equality [OF group_relative_homology_group __ assms])

obtain c **where** c: $R = \text{homologous_rel_set } p \ X \ S \ c \ \text{singular_relcycle } p \ X \ S \ c$

```

    using assms by (auto simp: carrier_relative_homology_group)
  have singular_relboundary p X S (b - a)
    if a ∈ R and b ∈ R for a b
    using c that
    by clarify (metis homologous_rel_def homologous_rel_eq)
  moreover
  have x ∈ (⋃ x∈R. ⋃ y∈R. {y - x})
    if singular_relboundary p X S x for x
    using c
    by simp (metis diff_eq_eq homologous_rel_def homologous_rel_refl homolo-
gous_rel_sym that)
  ultimately
  have (⋃ x∈R. ⋃ xa∈R. {xa - x}) = singular_relboundary_set p X S
    by auto
  then show uminus ' R ⊗_relative_homology_group (int p) X S R =
    1_relative_homology_group (int p) X S
    by (auto simp: carrier_relative_homology_group mult_relative_homology_group)
  have singular_relcycle p X S (-c)
    using c by (simp add: singular_relcycle_minus)
  moreover have homologous_rel p X S c x ⟹ homologous_rel p X S (-c) (-
x) for x
    by (metis homologous_rel_def homologous_rel_sym minus_diff_eq minus_diff_minus)
  moreover have homologous_rel p X S (-c) x ⟹ x ∈ uminus ' homolo-
gous_rel_set p X S c for x
    by (clarsimp simp: image_iff) (metis add.inverse_inverse diff_0 homolo-
gous_rel_diff homologous_rel_refl)
  ultimately show uminus ' R ∈ carrier (relative_homology_group (int p) X S)
    using c by (auto simp: carrier_relative_homology_group)
qed

```

lemma homologous_rel_eq_relboundary:

```

  homologous_rel p X S c = singular_relboundary p X S
  ⟷ singular_relboundary p X S c (is ?lhs = ?rhs)

```

proof

```

  assume ?lhs
  then show ?rhs
    unfolding homologous_rel_def
    by (metis diff_zero singular_relboundary_0)
next
  assume R: ?rhs
  show ?lhs
    unfolding homologous_rel_def
    using singular_relboundary_diff R by fastforce
qed

```

lemma homologous_rel_set_eq_relboundary:

```

  homologous_rel_set p X S c = singular_relboundary_set p X S ⟷ singu-
lar_relboundary p X S c
  by (auto simp flip: homologous_rel_eq_relboundary)

```

Lift the boundary and induced maps to homology groups. We totalize both quite aggressively to the appropriate group identity in all "undefined" situations, which makes several of the properties cleaner and simpler.

lemma *homomorphism_chain_boundary*:

```

  chain_boundary p ∈ hom (relcycle_group p X S) (relcycle_group(p - Suc 0)
(subtopology X S) {})
  (is ?h ∈ hom ?G ?H)
proof (rule homI)
  show  $\bigwedge x. x \in \text{carrier } ?G \implies ?h\ x \in \text{carrier } ?H$ 
  by (auto simp: singular_relcycle_def mod_subset_def chain_boundary_boundary)
qed (simp add: relcycle_group_def subgroup_generated_def chain_boundary_add)

```

lemma *hom_boundary1*:

```

   $\exists d. \forall p\ X\ S.$ 
     $d\ p\ X\ S \in \text{hom} (\text{relative\_homology\_group } (int\ p)\ X\ S)$ 
       $(\text{homology\_group } (int\ (p - Suc\ 0))\ (\text{subtopology } X\ S))$ 
     $\wedge (\forall c. \text{singular\_relcycle } p\ X\ S\ c$ 
       $\longrightarrow d\ p\ X\ S\ (\text{homologous\_rel\_set } p\ X\ S\ c)$ 
       $= \text{homologous\_rel\_set } (p - Suc\ 0)\ (\text{subtopology } X\ S)\ \{\}\ (\text{chain\_boundary}$ 
 $p\ c))$ 
  (is  $\exists d. \forall p\ X\ S. ?\Phi\ (d\ p\ X\ S)\ p\ X\ S$ )
proof ((subst choice_iff [symmetric])+, clarify)
  fix p X and S :: 'a set
  define  $\vartheta$  where  $\vartheta \equiv r\_coset\ (\text{relcycle\_group } (p - Suc\ 0)\ (\text{subtopology } X\ S)\ \{\})$ 
     $(\text{singular\_relboundary\_set } (p - Suc\ 0)\ (\text{subtopology } X\ S)\ \{\}) \circ$ 
 $\text{chain\_boundary } p$ 
  define H where  $H \equiv \text{relative\_homology\_group } (int\ (p - Suc\ 0))\ (\text{subtopology } X\ S)\ \{\}$ 
  define J where  $J \equiv \text{relcycle\_group } (p - Suc\ 0)\ (\text{subtopology } X\ S)\ \{\}$ 

  have  $\vartheta: \vartheta \in \text{hom} (\text{relcycle\_group } p\ X\ S)\ H$ 
  unfolding  $\vartheta\_def$ 
proof (rule hom_compose)
  show  $\text{chain\_boundary } p \in \text{hom} (\text{relcycle\_group } p\ X\ S)\ J$ 
  by (simp add: J_def homomorphism_chain_boundary)
  show  $(\#>\text{relcycle\_group } (p - Suc\ 0)\ (\text{subtopology } X\ S)\ \{\})$ 
     $(\text{singular\_relboundary\_set } (p - Suc\ 0)\ (\text{subtopology } X\ S)\ \{\}) \in \text{hom } J\ H$ 
  by (simp add: H_def J_def nontrivial_relative_homology_group
    normal.r_coset_hom_Mod normal_subgroup_singular_relboundary_relcycle)
qed
  have *:  $\text{singular\_relboundary } (p - Suc\ 0)\ (\text{subtopology } X\ S)\ \{\}\ (\text{chain\_boundary } p\ c)$ 
  if  $\text{singular\_relboundary } p\ X\ S\ c$  for c
proof (cases p=0)
  case True
  then show ?thesis
  by (metis chain_boundary_def singular_relboundary_0)

```

```

next
  case False
  with that have  $\exists d. \text{singular\_chain } p \text{ (subtopology } X \text{ } S) \ d \wedge \text{chain\_boundary}$ 
     $p \ d = \text{chain\_boundary } p \ c$ 
    by (metis add.left_neutral chain_boundary_add chain_boundary_boundary_alt
      singular_relboundary)
    with that False show ?thesis
    by (auto simp: singular_boundary)
qed
have  $\vartheta\_eq: \vartheta \ x = \vartheta \ y$ 
  if  $x: x \in \text{singular\_relcycle\_set } p \ X \ S$  and  $y: y \in \text{singular\_relcycle\_set } p \ X \ S$ 
  and  $eq: \text{singular\_relboundary\_set } p \ X \ S \ \#>_{\text{relcycle\_group } p \ X \ S} x$ 
     $= \text{singular\_relboundary\_set } p \ X \ S \ \#>_{\text{relcycle\_group } p \ X \ S} y$  for  $x \ y$ 
proof -
  have  $\text{singular\_relboundary } p \ X \ S \ (x - y)$ 
  by (metis eq_homologous_rel_def homologous_rel_eq mem_Collect_eq right_coset_singular_relboundary)
  with * have  $(\text{singular\_relboundary } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } S) \ \{\}) \text{ (chain\_boundary}$ 
     $p \ (x - y))$ 
  by blast
  then show ?thesis
  unfolding  $\vartheta\_def \text{ comp\_def}$ 
  by (metis chain_boundary_diff homologous_rel_def homologous_rel_eq right_coset_singular_relboundary)
qed
obtain  $d$ 
  where  $d \in \text{hom } ((\text{relcycle\_group } p \ X \ S) \text{ Mod } (\text{singular\_relboundary\_set } p \ X \ S)) \ H$ 
  and  $d: \bigwedge u. u \in \text{singular\_relcycle\_set } p \ X \ S \implies d \ (\text{homologous\_rel\_set } p \ X \ S \ u) = \vartheta \ u$ 
  by (metis FactGroup_universal [OF  $\vartheta \text{ normal\_subgroup\_singular\_relboundary\_relcycle}$ 
     $\vartheta\_eq$ ] right_coset_singular_relboundary carrier_relcycle_group)
  then have  $d \in \text{hom } (\text{relative\_homology\_group } p \ X \ S) \ H$ 
  by (simp add: nontrivial_relative_homology_group)
  then show  $\exists d. ?\Phi \ d \ p \ X \ S$ 
  by (force simp:  $H\_def \text{ right\_coset\_singular\_relboundary } d \ \vartheta\_def$ )
qed

lemma hom_boundary2:
   $\exists d. (\forall p \ X \ S.$ 
     $(d \ p \ X \ S) \in \text{hom } (\text{relative\_homology\_group } p \ X \ S)$ 
     $(\text{homology\_group } (p - 1) \text{ (subtopology } X \text{ } S)))$ 
     $\wedge (\forall p \ X \ S \ c. \text{singular\_relcycle } p \ X \ S \ c \wedge \text{Suc } 0 \leq p$ 
       $\longrightarrow d \ p \ X \ S \ (\text{homologous\_rel\_set } p \ X \ S \ c)$ 
       $= \text{homologous\_rel\_set } (p - \text{Suc } 0) \text{ (subtopology } X \text{ } S) \ \{\}) \text{ (chain\_boundary}$ 
     $p \ c))$ 
    (is  $\exists d. ?\Phi \ d$ )
proof -
  have *:  $\exists f. \Phi(\lambda p. \text{if } p \leq 0 \text{ then } \lambda q \ r \ t. \text{undefined else } f(\text{nat } p)) \implies \exists f. \Phi \ f$  for
     $\Phi$ 
  by blast

```

```

show ?thesis
  apply (rule * [OF ex_forward [OF hom_boundary1]])
  apply (simp add: not_le relative_homology_group_def nat_diff_distrib' int_eq_iff
nat_diff_distrib flip: nat_1)
  by (simp add: hom_def singleton_group_def)
qed

lemma hom_boundary3:
   $\exists d. ((\forall p \ X \ S \ c. c \notin \text{carrier}(\text{relative\_homology\_group } p \ X \ S) \longrightarrow d \ p \ X \ S \ c = \text{one}(\text{homology\_group } (p-1) (\text{subtopology } X \ S))) \wedge$ 
 $(\forall p \ X \ S. d \ p \ X \ S \in \text{hom}(\text{relative\_homology\_group } p \ X \ S) (\text{homology\_group } (p-1) (\text{subtopology } X \ S))) \wedge$ 
 $(\forall p \ X \ S \ c. \text{singular\_relcycle } p \ X \ S \ c \wedge 1 \leq p \longrightarrow d \ p \ X \ S (\text{homologous\_rel\_set } p \ X \ S \ c) = \text{homologous\_rel\_set } (p - \text{Suc } 0) (\text{subtopology } X \ S) \{\}) (\text{chain\_boundary } p \ c)) \wedge$ 
 $(\forall p \ X \ S. d \ p \ X \ S = d \ p \ X (\text{topspace } X \cap S))) \wedge$ 
 $(\forall p \ X \ S \ c. d \ p \ X \ S \ c \in \text{carrier}(\text{homology\_group } (p-1) (\text{subtopology } X \ S)))$ 
 $\wedge$ 
 $(\forall p. p \leq 0 \longrightarrow d \ p = (\lambda q \ r \ t. \text{undefined}))$ 
  (is  $\exists x. ?P \ x \wedge ?Q \ x \wedge ?R \ x$ )
proof -
  have  $\bigwedge x. ?Q \ x \implies ?R \ x$ 
  by (erule all_forward) (force simp: relative_homology_group_def)
  moreover have  $\exists x. ?P \ x \wedge ?Q \ x$ 
  proof -
    obtain d:: [int, 'a topology, 'a set, ('a chain) set]  $\Rightarrow$  ('a chain) set
    where 1:  $\bigwedge p \ X \ S. d \ p \ X \ S \in \text{hom}(\text{relative\_homology\_group } p \ X \ S) (\text{homology\_group } (p-1) (\text{subtopology } X \ S))$ 
    and 2:  $\bigwedge n \ X \ S \ c. \text{singular\_relcycle } n \ X \ S \ c \wedge \text{Suc } 0 \leq n \implies d \ n \ X \ S (\text{homologous\_rel\_set } n \ X \ S \ c) = \text{homologous\_rel\_set } (n - \text{Suc } 0) (\text{subtopology } X \ S) \{\}$ 
    (chain_boundary n c)
    using hom_boundary2 by blast
    have 4:  $c \in \text{carrier}(\text{relative\_homology\_group } p \ X \ S) \implies d \ p \ X (\text{topspace } X \cap S) \ c \in \text{carrier}(\text{relative\_homology\_group } (p-1) (\text{subtopology } X \ S) \{\})$ 
    for  $p \ X \ S \ c$ 
    using hom_carrier [OF 1 [of p X topspace X  $\cap$  S]]
    by (simp add: image_subset_iff subtopology_restrict)
    show ?thesis
    apply (rule_tac x= $\lambda p \ X \ S \ c.$ 
      if  $c \in \text{carrier}(\text{relative\_homology\_group } p \ X \ S)$ 
      then  $d \ p \ X (\text{topspace } X \cap S) \ c$ 
      else  $\text{one}(\text{homology\_group } (p-1) (\text{subtopology } X \ S))$  in exI)
    apply (simp add: Int_left_absorb subtopology_restrict carrier_relative_homology_group.is_monoid_group.restrict_hom_iff 4 cong: if_cong)

```

```

      by (metis 1 2 homologous_rel_restrict relative_homology_group_restrict
singular_relcycle_def subtopology_restrict)
    qed
    ultimately show ?thesis
      by auto
    qed

```

```

consts hom_boundary :: [int,'a topology,'a set,'a chain set]  $\Rightarrow$  'a chain set
specification (hom_boundary)
  hom_boundary:
    (( $\forall p X S c. c \notin \text{carrier}(\text{relative\_homology\_group } p X S)$ 
 $\longrightarrow \text{hom\_boundary } p X S c = \text{one}(\text{homology\_group } (p-1) (\text{subtopology } X (S::'a \text{ set})))$ ))  $\wedge$ 
    ( $\forall p X S.$ 
      hom_boundary  $p X S \in \text{hom}(\text{relative\_homology\_group } p X S)$ 
      ( $\text{homology\_group } (p-1) (\text{subtopology } X (S::'a \text{ set})))$ )  $\wedge$ 
    ( $\forall p X S c.$ 
      singular_relcycle  $p X S c \wedge 1 \leq p$ 
 $\longrightarrow \text{hom\_boundary } p X S (\text{homologous\_rel\_set } p X S c)$ 
      = homologous_rel_set  $(p - \text{Suc } 0) (\text{subtopology } X (S::'a \text{ set})) \{ \}$ 
      ( $\text{chain\_boundary } p c$ ))  $\wedge$ 
    ( $\forall p X S. \text{hom\_boundary } p X S = \text{hom\_boundary } p X (\text{topspace } X \cap (S::'a \text{ set}))$ ))  $\wedge$ 
    ( $\forall p X S c. \text{hom\_boundary } p X S c \in \text{carrier}(\text{homology\_group } (p-1) (\text{subtopology } X (S::'a \text{ set})))$ )  $\wedge$ 
    ( $\forall p. p \leq 0 \longrightarrow \text{hom\_boundary } p = (\lambda q r. \lambda t::'a \text{ chain set. undefined})$ )
  by (fact hom_boundary3)

```

```

lemma hom_boundary_default:
   $c \notin \text{carrier}(\text{relative\_homology\_group } p X S)$ 
 $\implies \text{hom\_boundary } p X S c = \text{one}(\text{homology\_group } (p-1) (\text{subtopology } X S))$ 
  and hom_boundary_hom:  $\text{hom\_boundary } p X S \in \text{hom}(\text{relative\_homology\_group } p X S)$ 
  ( $\text{homology\_group } (p-1) (\text{subtopology } X S)$ )
  and hom_boundary_restrict [simp]:  $\text{hom\_boundary } p X (\text{topspace } X \cap S) = \text{hom\_boundary } p X S$ 
  and hom_boundary_carrier:  $\text{hom\_boundary } p X S c \in \text{carrier}(\text{homology\_group } (p-1) (\text{subtopology } X S))$ 
  and hom_boundary_trivial:  $p \leq 0 \implies \text{hom\_boundary } p = (\lambda q r t. \text{undefined})$ 
  by (metis hom_boundary)+

```

```

lemma hom_boundary_chain_boundary:
  [ $\text{singular\_relcycle } p X S c; 1 \leq p$ ]
 $\implies \text{hom\_boundary } (\text{int } p) X S (\text{homologous\_rel\_set } p X S c) =$ 
  homologous_rel_set  $(p - \text{Suc } 0) (\text{subtopology } X S) \{ \}$  ( $\text{chain\_boundary } p c$ )
  by (metis hom_boundary)+

```

```

lemma hom_chain_map:

```



```

[[continuous_map X Y f; f ' S ⊆ T]]
  ⇒ (chain_map p f) ∈ hom (relcycle_group p X S) (relcycle_group p Y T)
by (force simp: chain_map_add singular_relcycle_chain_map hom_def)

```

lemma *hom_induced1*:

```

  ∃ hom_relmap.
    (∀ p X S Y T f.
      continuous_map X Y f ∧ f ' (topspace X ∩ S) ⊆ T
      → (hom_relmap p X S Y T f) ∈ hom (relative_homology_group (int p) X
S)
      (relative_homology_group (int p) Y T)) ∧
    (∀ p X S Y T f c.
      continuous_map X Y f ∧ f ' (topspace X ∩ S) ⊆ T ∧
      singular_relcycle p X S c
      → hom_relmap p X S Y T f (homologous_rel_set p X S c) =
      homologous_rel_set p Y T (chain_map p f c))
proof -
  have ∃ y. (y ∈ hom (relative_homology_group (int p) X S) (relative_homology_group
(int p) Y T)) ∧
    (∀ c. singular_relcycle p X S c →
      y (homologous_rel_set p X S c) = homologous_rel_set p Y T
(chain_map p f c))
  if contf: continuous_map X Y f and fim: f ' (topspace X ∩ S) ⊆ T
  for p X S Y T and f :: 'a ⇒ 'b
  proof -
  let ?f = (#>relcycle_group p Y T) (singular_relboundary_set p Y T) ∘ chain_map
p f
  let ?F = λx. singular_relboundary_set p X S #>relcycle_group p X S x
  have chain_map p f ∈ hom (relcycle_group p X S) (relcycle_group p Y T)
  by (metis contf fim hom_chain_map relcycle_group_restrict)
  then have 1: ?f ∈ hom (relcycle_group p X S) (relative_homology_group (int
p) Y T)
  by (simp add: hom_compose normal.r_coset_hom_Mod normal_subgroup_singular_relboundary_relcycle
relative_homology_group_def)
  have 2: singular_relboundary_set p X S ⊆ relcycle_group p X S
  using normal_subgroup_singular_relboundary_relcycle by blast
  have 3: ?f x = ?f y
  if singular_relcycle p X S x singular_relcycle p X S y ?F x = ?F y for x y
  proof -
  have homologous_rel p X S x y
  by (metis (no_types) homologous_rel_set_eq right_coset_singular_relboundary
that(3))
  then have singular_relboundary p Y T (chain_map p f (x - y))
  using singular_relboundary_chain_map [OF _ contf fim] by (simp add:
homologous_rel_def)
  then have singular_relboundary p Y T (chain_map p f x - chain_map p f
y)
  by (simp add: chain_map_diff)

```

```

    with that
    show ?thesis
    by (metis comp_apply homologous_rel_def homologous_rel_set_eq right_coset_singular_relbounds)
  qed
  obtain g where g ∈ hom (relcycle_group p X S Mod singular_relbounds_set
    p X S)
    (relative_homology_group (int p) Y T)
    ∧ x. x ∈ singular_relcycle_set p X S ⇒ g (?F x) = ?f x
  using FactGroup_universal [OF 1 2 3, unfolded carrier_relcycle_group] by
blast
  then show ?thesis
  by (force simp: right_coset_singular_relbounds nontrivial_relative_homology_group)
  qed
  then show ?thesis
  apply (simp flip: all_conj_distrib)
  apply ((subst choice_iff [symmetric])+)
  apply metis
  done
  qed

```

lemma *hom_induced2*:

```

  ∃ hom_relmap.
    (∀ p X S Y T f.
      continuous_map X Y f ∧
      f ‘ (topspace X ∩ S) ⊆ T
      → (hom_relmap p X S Y T f) ∈ hom (relative_homology_group p X S)
        (relative_homology_group p Y T)) ∧
    (∀ p X S Y T f c.
      continuous_map X Y f ∧
      f ‘ (topspace X ∩ S) ⊆ T ∧
      singular_relcycle p X S c
      → hom_relmap p X S Y T f (homologous_rel_set p X S c) =
        homologous_rel_set p Y T (chain_map p f c)) ∧
    (∀ p. p < 0 → hom_relmap p = (λ X S Y T f c. undefined))
  (is ∃ d. ?Φ d)

```

proof –

```

  have *: ∃ f. Φ(λ p. if p < 0 then λ X S Y T f c. undefined else f(nat p)) ⇒ ∃ f.
  Φ f for Φ
  by blast
  show ?thesis
  apply (rule * [OF ex_forward [OF hom_induced1]])
  apply (simp add: not_le relative_homology_group_def nat_diff_distrib' int_eq_iff
    nat_diff_distrib flip: nat_1)
  done
  qed

```

lemma *hom_induced3*:

```

  ∃ hom_relmap.
    ((∀ p X S Y T f c.

```

```

  ~ (continuous_map X Y f ∧ f ' (topspace X ∩ S) ⊆ T ∧
    c ∈ carrier (relative_homology_group p X S))
  → hom_relmap p X S Y T f c = one (relative_homology_group p Y T)) ∧
  (∀ p X S Y T f.
    hom_relmap p X S Y T f ∈ hom (relative_homology_group p X S)
    (relative_homology_group p Y T)) ∧
  (∀ p X S Y T f c.
    continuous_map X Y f ∧ f ' (topspace X ∩ S) ⊆ T ∧ singular_relcycle p
    X S c
    → hom_relmap p X S Y T f (homologous_rel_set p X S c) =
      homologous_rel_set p Y T (chain_map p f c)) ∧
  (∀ p X S Y T.
    hom_relmap p X S Y T =
    hom_relmap p X (topspace X ∩ S) Y (topspace Y ∩ T))) ∧
  (∀ p X S Y T f c.
    hom_relmap p X S Y T f c ∈ carrier (relative_homology_group p Y T)) ∧
  (∀ p. p < 0 → hom_relmap p = (λ X S Y T f c. undefined))
  (is ∃ x. ?P x ∧ ?Q x ∧ ?R x)
proof -
  have ∧x. ?Q x ⇒ ?R x
  by (erule all_forward) (fastforce simp: relative_homology_group_def)
  moreover have ∃ x. ?P x ∧ ?Q x
  proof -
    obtain hom_relmap:: [int, 'a topology, 'a set, 'b topology, 'b set, 'a ⇒ 'b, ('a chain)
    set] ⇒ ('b chain) set
    where 1: ∧p X S Y T f. [[continuous_map X Y f; f ' (topspace X ∩ S) ⊆
    T]] ⇒
      hom_relmap p X S Y T f
      ∈ hom (relative_homology_group p X S) (relative_homology_group
    p Y T)
    and 2: ∧p X S Y T f c.
      [[continuous_map X Y f; f ' (topspace X ∩ S) ⊆ T; singular_relcycle
    p X S c]]
      ⇒
      hom_relmap (int p) X S Y T f (homologous_rel_set p X S c) =
      homologous_rel_set p Y T (chain_map p f c)
    and 3: (∀ p. p < 0 → hom_relmap p = (λ X S Y T f c. undefined))
    using hom_induced2 [where ?'a='a and ?'b='b]
    by (metis (mono_tags, lifting))
    have 4: [[continuous_map X Y f; f ' (topspace X ∩ S) ⊆ T; c ∈ carrier
    (relative_homology_group p X S)]] ⇒
      hom_relmap p X (topspace X ∩ S) Y (topspace Y ∩ T) f c
      ∈ carrier (relative_homology_group p Y T)
    for p X S Y T f c
    using hom_carrier [OF 1 [of X Y f topspace X ∩ S topspace Y ∩ T p]]
    continuous_map_image_subset_topspace by fastforce
    have inhom: (λ c. if continuous_map X Y f ∧ f ' (topspace X ∩ S) ⊆ T ∧
      c ∈ carrier (relative_homology_group p X S)
      then hom_relmap p X (topspace X ∩ S) Y (topspace Y ∩ T) f c

```

```

      else 1relative_homology_group p Y T)
    ∈ hom (relative_homology_group p X S) (relative_homology_group p Y T)
(is ?h ∈ hom ?GX ?GY)
  for p X S Y T f
  proof (rule homI)
    show  $\bigwedge x. x \in \text{carrier } ?GX \implies ?h x \in \text{carrier } ?GY$ 
      by (auto simp: 4 group.is_monoid)
    show ?h (x  $\otimes_{?GX}$  y) = ?h x  $\otimes_{?GY}$  ?h y if x ∈ carrier ?GX y ∈ carrier ?GX
  for x y
  proof (cases p < 0)
    case True
      with that show ?thesis
        by (simp add: relative_homology_group_def singleton_group_def 3)
    next
      case False
      show ?thesis
      proof (cases continuous_map X Y f)
        case True
          then have f' (topspace X  $\cap$  S)  $\subseteq$  topspace Y
            using continuous_map_image_subset_topspace by blast
          then show ?thesis
            using True False that
            using 1 [of X Y f topspace X  $\cap$  S topspace Y  $\cap$  T p]
            by (simp add: 4 continuous_map_image_subset_topspace hom_mult
not_less group.is_monoid monoid.m_closed Int_left_absorb)
          qed (simp add: group.is_monoid)
        qed
      qed
      have hrel:  $\llbracket \text{continuous\_map } X Y f; f' (\text{topspace } X \cap S) \subseteq T; \text{singular\_relcycle } p X S c \rrbracket$ 
         $\implies \text{hom\_relmap } (\text{int } p) X (\text{topspace } X \cap S) Y (\text{topspace } Y \cap T)$ 
        f (homologous_rel_set p X S c) = homologous_rel_set p Y T (chain_map
p f c)
        for p X S Y T f c
        using 2 [of X Y f topspace X  $\cap$  S topspace Y  $\cap$  T p c]
        continuous_map_image_subset_topspace by fastforce
      show ?thesis
      apply (rule_tac x= $\lambda p X S Y T f c.$ 
        if continuous_map X Y f  $\wedge$  f' (topspace X  $\cap$  S)  $\subseteq$  T  $\wedge$ 
        c ∈ carrier(relative_homology_group p X S)
        then hom_relmap p X (topspace X  $\cap$  S) Y (topspace Y  $\cap$  T) f c
        else one(relative_homology_group p Y T) in exI)
      apply (simp add: Int_left_absorb subtopology_restrict carrier_relative_homology_group
group.is_monoid group.restrict_hom_iff 4 inhom hrel cong: if_cong)
      apply (force simp: continuous_map_def intro!: ext)
      done
    qed
  ultimately show ?thesis
  by auto

```

qed

```

consts hom_induced:: [int,'a topology,'a set,'b topology,'b set,'a  $\Rightarrow$  'b,('a chain)
set]  $\Rightarrow$  ('b chain) set
specification (hom_induced)
  hom_induced:
    (( $\forall p X S Y T f c.$ 
       $\sim(\text{continuous\_map } X Y f \wedge$ 
         $f'(\text{topspace } X \cap S) \subseteq T \wedge$ 
         $c \in \text{carrier}(\text{relative\_homology\_group } p X S))$ 
       $\longrightarrow \text{hom\_induced } p X (S::'a \text{ set}) Y (T::'b \text{ set}) f c =$ 
         $\text{one}(\text{relative\_homology\_group } p Y T)) \wedge$ 
      ( $\forall p X S Y T f.$ 
         $(\text{hom\_induced } p X (S::'a \text{ set}) Y (T::'b \text{ set}) f) \in \text{hom}(\text{relative\_homology\_group}$ 
           $p X S)$ 
           $(\text{relative\_homology\_group } p Y T)) \wedge$ 
        ( $\forall p X S Y T f c.$ 
           $\text{continuous\_map } X Y f \wedge$ 
           $f'(\text{topspace } X \cap S) \subseteq T \wedge$ 
           $\text{singular\_relcycle } p X S c$ 
           $\longrightarrow \text{hom\_induced } p X (S::'a \text{ set}) Y (T::'b \text{ set}) f (\text{homologous\_rel\_set } p X$ 
             $S c) =$ 
             $\text{homologous\_rel\_set } p Y T (\text{chain\_map } p f c)) \wedge$ 
          ( $\forall p X S Y T.$ 
             $\text{hom\_induced } p X (S::'a \text{ set}) Y (T::'b \text{ set}) =$ 
             $\text{hom\_induced } p X (\text{topspace } X \cap S) Y (\text{topspace } Y \cap T))) \wedge$ 
          ( $\forall p X S Y f T c.$ 
             $\text{hom\_induced } p X (S::'a \text{ set}) Y (T::'b \text{ set}) f c \in$ 
             $\text{carrier}(\text{relative\_homology\_group } p Y T)) \wedge$ 
          ( $\forall p. p < 0 \longrightarrow \text{hom\_induced } p = (\lambda X S Y T. \lambda f::'a \Rightarrow 'b. \lambda c. \text{undefined}))$ 
        )
    )
  by (fact hom_induced3)

lemma hom_induced_default:
   $\sim(\text{continuous\_map } X Y f \wedge f'(\text{topspace } X \cap S) \subseteq T \wedge c \in \text{carrier}(\text{relative\_homology\_group}$ 
     $p X S))$ 
     $\Longrightarrow \text{hom\_induced } p X S Y T f c = \text{one}(\text{relative\_homology\_group } p Y T)$ 
  and hom_induced_hom:
     $\text{hom\_induced } p X S Y T f \in \text{hom}(\text{relative\_homology\_group } p X S) (\text{relative\_homology\_group}$ 
       $p Y T)$ 
  and hom_induced_restrict [simp]:
     $\text{hom\_induced } p X (\text{topspace } X \cap S) Y (\text{topspace } Y \cap T) = \text{hom\_induced } p X$ 
       $S Y T$ 
  and hom_induced_carrier:
     $\text{hom\_induced } p X S Y T f c \in \text{carrier}(\text{relative\_homology\_group } p Y T)$ 
  and hom_induced_trivial:  $p < 0 \Longrightarrow \text{hom\_induced } p = (\lambda X S Y T f c. \text{undefined})$ 
  by (metis hom_induced)+

lemma hom_induced_chain_map_gen:

```

$\llbracket \text{continuous_map } X \ Y \ f; f' (topspace \ X \cap S) \subseteq T; \text{singular_relcycle } p \ X \ S \ c \rrbracket$
 $\implies \text{hom_induced } p \ X \ S \ Y \ T \ f \ (\text{homologous_rel_set } p \ X \ S \ c) = \text{homologous_rel_set } p \ Y \ T \ (\text{chain_map } p \ f \ c)$
by (metis hom_induced)

lemma hom_induced_chain_map:
 $\llbracket \text{continuous_map } X \ Y \ f; f' S \subseteq T; \text{singular_relcycle } p \ X \ S \ c \rrbracket$
 $\implies \text{hom_induced } p \ X \ S \ Y \ T \ f \ (\text{homologous_rel_set } p \ X \ S \ c)$
 $= \text{homologous_rel_set } p \ Y \ T \ (\text{chain_map } p \ f \ c)$
by (meson Int_lower2 hom_induced image_subsetI image_subset_iff subset_iff)

lemma hom_induced_eq:
assumes $\bigwedge x. x \in topspace \ X \implies f \ x = g \ x$
shows $\text{hom_induced } p \ X \ S \ Y \ T \ f = \text{hom_induced } p \ X \ S \ Y \ T \ g$
proof –
consider $p < 0 \mid n$ **where** $p = \text{int } n$
by (metis int_nat_eq not_less)
then show ?thesis
proof cases
case 1
then show ?thesis
by (simp add: hom_induced_trivial)
next
case 2
have $\text{hom_induced } n \ X \ S \ Y \ T \ f \ C = \text{hom_induced } n \ X \ S \ Y \ T \ g \ C$ **for** C
proof –
have $\text{continuous_map } X \ Y \ f \wedge f' (topspace \ X \cap S) \subseteq T \wedge C \in \text{carrier}$
 $(\text{relative_homology_group } n \ X \ S)$
 $\longleftrightarrow \text{continuous_map } X \ Y \ g \wedge g' (topspace \ X \cap S) \subseteq T \wedge C \in \text{carrier}$
 $(\text{relative_homology_group } n \ X \ S)$
(is ?P = ?Q)
by (metis IntD1 assms continuous_map_eq image_cong)
then consider $\neg ?P \wedge \neg ?Q \mid ?P \wedge ?Q$
by blast
then show ?thesis
proof cases
case 1
then show ?thesis
by (simp add: hom_induced_default)
next
case 2
have $\text{homologous_rel_set } n \ Y \ T \ (\text{chain_map } n \ f \ c) = \text{homologous_rel_set}$
 $n \ Y \ T \ (\text{chain_map } n \ g \ c)$
if $\text{continuous_map } X \ Y \ f \wedge f' (topspace \ X \cap S) \subseteq T$
 $\text{continuous_map } X \ Y \ g \wedge g' (topspace \ X \cap S) \subseteq T$
 $C = \text{homologous_rel_set } n \ X \ S \ c \text{ singular_relcycle } n \ X \ S \ c$
for c
proof –

```

    have chain_map n f c = chain_map n g c
      using assms chain_map_eq singular_relcycle that by blast
    then show ?thesis
      by simp
  qed
with 2 show ?thesis
  by (auto simp: relative_homology_group_def carrier_FactGroup
    right_coset_singular_relboundary hom_induced_chain_map_gen)
qed
qed
with 2 show ?thesis
  by auto
qed
qed

```

0.2.2 Towards the Eilenberg-Steenrod axioms

First prove we get functors into abelian groups with the boundary map being a natural transformation between them, and prove Eilenberg-Steenrod axioms (we also prove additivity a bit later on if one counts that).

```

lemma abelian_relative_homology_group [simp]:
  comm_group(relative_homology_group p X S)
  by (simp add: comm_group.abelian_FactGroup relative_homology_group_def
    subgroup_singular_relboundary_relcycle)

```

```

lemma abelian_homology_group: comm_group(homology_group p X)
  by simp

```

```

lemma hom_induced_id_gen:
  assumes contf: continuous_map X X f and feq:  $\bigwedge x. x \in \text{topspace } X \implies f x = x$ 
  and c:  $c \in \text{carrier } (\text{relative\_homology\_group } p \text{ } X \text{ } S)$ 
  shows hom_induced p X S X S f c = c
proof -
  consider  $p < 0 \mid n$  where  $p = \text{int } n$ 
  by (metis int_nat_eq not_less)
  then show ?thesis
proof cases
  case 1
  with c show ?thesis
    by (simp add: hom_induced_trivial relative_homology_group_def)
next
  case 2
  have cm: chain_map n f d = d if singular_relcycle n X S d for d
    using that assms by (auto simp: chain_map_id_gen singular_relcycle)
  have f'  $(\text{topspace } X \cap S) \subseteq S$ 
    using feq by auto
  with 2 c show ?thesis

```

```

    by (auto simp: nontrivial_relative_homology_group carrier_FactGroup
        cm_right_coset_singular_relboundary hom_induced_chain_map_gen
        assms)
  qed
qed

```

lemma *hom_induced_id*:

```

  c ∈ carrier (relative_homology_group p X S) ⇒ hom_induced p X S X S id c
= c
  by (rule hom_induced_id_gen) auto

```

lemma *hom_induced_compose*:

```

  assumes continuous_map X Y f f ' S ⊆ T continuous_map Y Z g g ' T ⊆ U
  shows hom_induced p X S Z U (g ∘ f) = hom_induced p Y T Z U g ∘
hom_induced p X S Y T f

```

proof –

```

  consider (neg) p < 0 | (int) n where p = int n
  by (metis int_nat_eq not_less)
  then show ?thesis
  proof cases
    case int
    have gf: continuous_map X Z (g ∘ f)
    using assms continuous_map_compose by fastforce
    have gfim: (g ∘ f) ' S ⊆ U
    unfolding o_def using assms by blast
    have sr: ∧ a. singular_relcycle n X S a ⇒ singular_relcycle n Y T (chain_map
n f a)

```

```

    by (simp add: assms singular_relcycle_chain_map)

```

```

  show ?thesis

```

proof

fix c

```

  show hom_induced p X S Z U (g ∘ f) c = (hom_induced p Y T Z U g ∘
hom_induced p X S Y T f) c

```

```

  proof (cases c ∈ carrier(relative_homology_group p X S))

```

case True

with gfim **show** ?thesis

unfolding int

```

  by (auto simp: carrier_relative_homology_group gf gfim assms sr chain_map_compose
hom_induced_chain_map)

```

next

case False

then show ?thesis

```

  by (simp add: hom_induced_default hom_one [OF hom_induced_hom])

```

qed

qed

```

  qed (force simp: hom_induced_trivial)

```

qed


```

lemma hom_induced_compose':
  assumes continuous_map  $X\ Y\ f\ f' : S \subseteq T \text{ continuous\_map } Y\ Z\ g\ g' : T \subseteq U$ 
  shows  $\text{hom\_induced } p\ Y\ T\ Z\ U\ g\ (\text{hom\_induced } p\ X\ S\ Y\ T\ f\ x) = \text{hom\_induced}$ 
 $p\ X\ S\ Z\ U\ (g \circ f)\ x$ 
  using hom_induced_compose [OF assms] by simp

lemma naturality_hom_induced:
  assumes continuous_map  $X\ Y\ f\ f' : S \subseteq T$ 
  shows  $\text{hom\_boundary } q\ Y\ T \circ \text{hom\_induced } q\ X\ S\ Y\ T\ f$ 
 $= \text{hom\_induced } (q - 1)\ (\text{subtopology } X\ S)\ \{\}\ (\text{subtopology } Y\ T)\ \{\}\ f \circ$ 
 $\text{hom\_boundary } q\ X\ S$ 
  proof (cases  $q \leq 0$ )
    case False
    then obtain  $p$  where  $p1: p \geq \text{Suc } 0$  and  $q: q = \text{int } p$ 
    using zero_le_imp_eq_int by force
    show ?thesis
    proof
      fix  $c$ 
      show  $(\text{hom\_boundary } q\ Y\ T \circ \text{hom\_induced } q\ X\ S\ Y\ T\ f)\ c =$ 
 $(\text{hom\_induced } (q - 1)\ (\text{subtopology } X\ S)\ \{\}\ (\text{subtopology } Y\ T)\ \{\}\ f \circ$ 
 $\text{hom\_boundary } q\ X\ S)\ c$ 
      proof (cases  $c \in \text{carrier}(\text{relative\_homology\_group } p\ X\ S)$ )
        case True
        then obtain  $a$  where  $\text{ceq}: c = \text{homologous\_rel\_set } p\ X\ S\ a$  and  $a: \text{singular\_relcycle } p\ X\ S\ a$ 
        by (force simp: carrier\_relative\_homology\_group)
        then have  $\text{sr}: \text{singular\_relcycle } p\ Y\ T\ (\text{chain\_map } p\ f\ a)$ 
        using assms singular\_relcycle\_chain\_map by fastforce
        then have  $\text{sb}: \text{singular\_relcycle } (p - \text{Suc } 0)\ (\text{subtopology } X\ S)\ \{\}\ (\text{chain\_boundary } p\ a)$ 
        by (metis One_nat_def a chain\_boundary\_boundary singular\_chain_0 singular\_relcycle)
        have  $p1\_eq: \text{int } p - 1 = \text{int } (p - \text{Suc } 0)$ 
        using  $p1$  by auto
        have  $\text{cbm}: (\text{chain\_boundary } p\ (\text{chain\_map } p\ f\ a))$ 
 $= (\text{chain\_map } (p - \text{Suc } 0)\ f\ (\text{chain\_boundary } p\ a))$ 
        using  $a$  chain\_boundary\_chain\_map singular\_relcycle by blast
        have  $\text{contf}: \text{continuous\_map } (\text{subtopology } X\ S)\ (\text{subtopology } Y\ T)\ f$ 
        using assms
        by (auto simp: continuous\_map\_in\_subtopology continuous\_map\_from\_subtopology)
        show ?thesis
        unfolding  $q$  using assms  $p1\ a$ 
        by (simp add: cbm ceq contf hom\_boundary\_chain\_boundary hom\_induced\_chain\_map
 $p1\_eq\ sb\ sr$ 
 $\text{del: of\_nat\_diff}$ )
      next
      case False
      with assms show ?thesis
      unfolding  $q$  o_def using assms

```

```

    apply (simp add: hom_induced_default hom_boundary_default)
    by (metis group_relative_homology_group hom_boundary hom_induced
hom_one one_relative_homology_group)
  qed
  qed
qed (force simp: hom_induced_trivial hom_boundary_trivial)

```

lemma *homology_exactness_axiom_1*:

```

  exact_seq ([homology_group (p-1) (subtopology X S), relative_homology_group
p X S, homology_group p X],
    [hom_boundary p X S, hom_induced p X {} X S id])

```

proof –

```

  consider (neg) p < 0 | (int) n where p = int n
  by (metis int_nat_eq not_less)
  then have (hom_induced p X {} X S id) ‘ carrier (homology_group p X)
    = kernel (relative_homology_group p X S) (homology_group (p-1)
(subtopology X S))
    (hom_boundary p X S)

```

proof *cases*

case *neg*

then show *?thesis*

```

  unfolding kernel_def singleton_group_def relative_homology_group_def
  by (auto simp: hom_induced_trivial hom_boundary_trivial)

```

next

case *int*

```

  have hom_induced (int m) X {} X S id ‘ carrier (relative_homology_group
(int m) X {})
    = carrier (relative_homology_group (int m) X S) ∩
    {c. hom_boundary (int m) X S c = 1_{relative_homology_group (int m - 1) (subtopology X S) {}}}

```

for *m*

proof (*cases m*)

case *0*

```

  have hom_induced 0 X {} X S id ‘ carrier (relative_homology_group 0 X
{})
    = carrier (relative_homology_group 0 X S) (is ?lhs = ?rhs)

```

proof

show *?lhs* ⊆ *?rhs*

using *hom_induced_hom* [of 0 X {} X S id]

by (*simp add: hom_induced_hom hom_carrier*)

show *?rhs* ⊆ *?lhs*

```

  apply (clarsimp simp add: image_iff carrier_relative_homology_group [of
0, simplified] singular_relcycle)

```

```

  apply (force simp: chain_map_id_gen chain_boundary_def singular_relcycle

```

```

    hom_induced_chain_map [of concl: 0, simplified])

```

done

qed

```

with 0 show ?thesis
by (simp add: hom_boundary_trivial relative_homology_group_def [of -1]
singleton_group_def)
next
case (Suc n)
have (hom_induced (int (Suc n)) X {} X S id ◦
homologous_rel_set (Suc n) X {}) 'singular_relcycle_set (Suc n) X {}
= homologous_rel_set (Suc n) X S '
(singular_relcycle_set (Suc n) X S ∩
{c. hom_boundary (int (Suc n)) X S (homologous_rel_set (Suc n) X S
c)
= singular_relboundary_set n (subtopology X S) {}})
(is ?lhs = ?rhs)
proof -
have 1: (∧x. x ∈ A ⇒ x ∈ B ⇔ x ∈ C) ⇒ f ' (A ∩ B) = f ' (A ∩ C)
for f A B C
by blast
have 2: [∧x. x ∈ A ⇒ ∃y. y ∈ B ∧ f x = f y; ∧x. x ∈ B ⇒ ∃y. y ∈ A
∧ f x = f y]
⇒ f ' A = f ' B for f A B
by blast
have ?lhs = homologous_rel_set (Suc n) X S ' singular_relcycle_set (Suc
n) X {}
using hom_induced_chain_map chain_map_ident [of _ X] singu-
lar_relcycle
by (smt (verit) bot.extremum comp_apply continuous_map_id image_cong
image_empty mem_Collect_eq)
also have ... = homologous_rel_set (Suc n) X S '
(singular_relcycle_set (Suc n) X S ∩
{c. singular_relboundary n (subtopology X S) {} (chain_boundary
(Suc n) c)})
proof (rule 2)
fix c
assume c ∈ singular_relcycle_set (Suc n) X {}
then show ∃y. y ∈ singular_relcycle_set (Suc n) X S ∩
{c. singular_relboundary n (subtopology X S) {} (chain_boundary
(Suc n) c)} ∧
homologous_rel_set (Suc n) X S c = homologous_rel_set (Suc n)
X S y
using singular_cycle singular_relcycle by (fastforce simp: singu-
lar_boundary)
next
fix c
assume c: c ∈ singular_relcycle_set (Suc n) X S ∩
{c. singular_relboundary n (subtopology X S) {} (chain_boundary
(Suc n) c)}
then obtain d where d: singular_chain (Suc n) (subtopology X S) d
chain_boundary (Suc n) d = chain_boundary (Suc n) c
by (auto simp: singular_boundary)

```

```

      with c have c - d ∈ singular_relcycle_set (Suc n) X {}
    by (auto simp: singular_cycle_chain_boundary_diff singular_chain_subtopology
singular_relcycle singular_chain_diff)
    moreover have homologous_rel_set (Suc n) X S c = homologous_rel_set
(Suc n) X S (c - d)
    proof (simp add: homologous_rel_set_eq)
      show homologous_rel (Suc n) X S c (c - d)
    using d by (simp add: homologous_rel_def singular_chain_imp_relboundary)
    qed
    ultimately show ∃ y. y ∈ singular_relcycle_set (Suc n) X {} ∧
      homologous_rel_set (Suc n) X S c = homologous_rel_set (Suc n)
X S y
      by blast
    qed
    also have ... = ?rhs
    by (rule 1) (simp add: hom_boundary_chain_boundary homologous_rel_set_eq_relboundary
del: of_nat_Suc)
    finally show ?lhs = ?rhs .
    qed
    with Suc show ?thesis
    unfolding carrier_relative_homology_group image_comp id_def by auto
    qed
    then show ?thesis
    by (auto simp: kernel_def int)
    qed
    then show ?thesis
    using hom_boundary_hom hom_induced_hom
    by (force simp: group_hom_def group_hom_axioms_def)
  qed

```

lemma *homology_exactness_axiom_2*:

```

exact_seq ([homology_group (p-1) X, homology_group (p-1) (subtopology X
S), relative_homology_group p X S],
[hom_induced (p-1) (subtopology X S) {} X {} id, hom_boundary p
X S])

```

proof –

```

consider (neg) p ≤ 0 | (int) n where p = int (Suc n)
  by (metis linear not0_implies_Suc of_nat_0 zero_le_imp_eq_int)
then have kernel (relative_homology_group (p-1) (subtopology X S) {})
  (relative_homology_group (p-1) X {})
  (hom_induced (p-1) (subtopology X S) {} X {} id)
  = hom_boundary p X S ‘ carrier (relative_homology_group p X S)

```

proof *cases*

case *neg*

obtain *x* **where** *x* ∈ *carrier* (relative_homology_group p X S)

using *group_relative_homology_group* *group.is_monoid* **by** *blast*

with *neg* **show** *?thesis*

unfolding *kernel_def singleton_group_def* *relative_homology_group_def*

```

    by (force simp: hom_induced_trivial hom_boundary_trivial)
  next
    case int
    have hom_boundary (int (Suc n)) X S ' carrier (relative_homology_group (int
(Suc n)) X S)
      = carrier (relative_homology_group n (subtopology X S) {}) ∩
        {c. hom_induced n (subtopology X S) {} X {} id c =
          1relative_homology_group n X {}}
      (is ?lhs = ?rhs)
    proof -
      have 1: (∧x. x ∈ A ⇒ x ∈ B ⇔ x ∈ C) ⇒ f ' (A ∩ B) = f ' (A ∩ C)
    for f A B C
      by blast
      have 2: (∧x. x ∈ A ⇒ x ∈ B ⇔ x ∈ f - ' C) ⇒ f ' (A ∩ B) = f ' A ∩
C for f A B C
      by blast
      have ?lhs = homologous_rel_set n (subtopology X S) {}
        ' (chain_boundary (Suc n) ' singular_relcycle_set (Suc n) X S)
      unfolding carrier_relative_homology_group image_comp
      by (rule image_cong [OF refl]) (simp add: o_def hom_boundary_chain_boundary
del: of_nat_Suc)
      also have ... = homologous_rel_set n (subtopology X S) {} '
        (singular_relcycle_set n (subtopology X S) {}) ∩ singu-
lar_relboundary_set n X {}
      by (force simp: singular_relcycle singular_boundary chain_boundary_boundary_alt)
      also have ... = ?rhs
      unfolding carrier_relative_homology_group vimage_def
      by (intro 2) (auto simp: hom_induced_chain_map chain_map_ident ho-
mologous_rel_set_eq_relboundary singular_relcycle)
      finally show ?thesis .
    qed
    then show ?thesis
      by (auto simp: kernel_def int)
  qed
  then show ?thesis
    using hom_boundary_hom hom_induced_hom
    by (force simp: group_hom_def group_hom_axioms_def)
  qed

```

lemma homology_exactness_axiom_3:

```

  exact_seq ([relative_homology_group p X S, homology_group p X, homol-
ogy_group p (subtopology X S)],
    [hom_induced p X {} X S id, hom_induced p (subtopology X S) {} X
{} id])
proof (cases p < 0)
  case True
  then show ?thesis
    unfolding relative_homology_group_def

```

```

    by (simp add: group_hom.kernel_to_trivial_group group_hom_axioms_def
group_hom_def hom_induced_trivial)
next
  case False
  then obtain n where peg: p = int n
    by (metis int_ops(1) linorder_neqE_linordered_idom pos_int_cases)
  have hom_induced n (subtopology X S) {} X {} id '
    (homologous_rel_set n (subtopology X S) {}) '
    singular_relcycle_set n (subtopology X S) {})
    = {c ∈ homologous_rel_set n X {} ' singular_relcycle_set n X {}}.
    hom_induced n X {} X S id c = singular_relboundary_set n X S}
    (is ?lhs = ?rhs)
  proof -
    have 2: [⋀x. x ∈ A ⇒ ∃y. y ∈ B ∧ f x = f y; ⋀x. x ∈ B ⇒ ∃y. y ∈ A ∧
f x = f y]
      ⇒ f ' A = f ' B for f A B
    by blast
    have ?lhs = homologous_rel_set n X {} ' (singular_relcycle_set n (subtopology
X S) {})
    by (smt (verit) chain_map_ident continuous_map_id_subt empty_subsetI
hom_induced_chain_map image_cong image_empty image_image mem_Collect_eq
singular_relcycle)
    also have ... = homologous_rel_set n X {} ' (singular_relcycle_set n X {}
∩ singular_relboundary_set n X S)
    proof (rule 2)
      fix c
      assume c ∈ singular_relcycle_set n (subtopology X S) {}
      then show ∃y. y ∈ singular_relcycle_set n X {} ∩ singular_relboundary_set
n X S ∧
        homologous_rel_set n X {} c = homologous_rel_set n X {} y
      using singular_chain_imp_relboundary singular_cycle singular_relboundary_imp_chain
singular_relcycle by fastforce
    next
      fix c
      assume c ∈ singular_relcycle_set n X {} ∩ singular_relboundary_set n X S
      then obtain d e where c: singular_relcycle n X {} c singular_relboundary
n X S c
        and d: singular_chain n (subtopology X S) d
        and e: singular_chain (Suc n) X e chain_boundary (Suc n) e = c + d
      using singular_relboundary_alt by blast
      then have chain_boundary n (c + d) = 0
      using chain_boundary_boundary_alt by fastforce
      then have chain_boundary n c + chain_boundary n d = 0
      by (metis chain_boundary_add)
      with c have singular_relcycle n (subtopology X S) {} (- d)
      by (metis (no_types) d eq_add_iff singular_cycle singular_relcycle_minus)
      moreover have homologous_rel n X {} c (- d)
      using c
      by (metis diff_minus_eq_add e homologous_rel_def singular_boundary)

```

```

ultimately
show  $\exists y. y \in \text{singular\_relcycle\_set } n \ (\text{subtopology } X \ S) \ \{\}$   $\wedge$ 
 $\text{homologous\_rel\_set } n \ X \ \{\} \ c = \text{homologous\_rel\_set } n \ X \ \{\} \ y$ 
by (force simp: homologous_rel_set_eq)
qed
also have  $\dots = \text{homologous\_rel\_set } n \ X \ \{\}$  ‘
 $(\text{singular\_relcycle\_set } n \ X \ \{\} \cap \text{homologous\_rel\_set } n \ X \ \{\} - \{x.$ 
 $\text{hom\_induced } n \ X \ \{\} \ X \ S \ \text{id } x = \text{singular\_relboundary\_set } n \ X \ S\})$ 
by (rule 2) (auto simp: hom_induced_chain_map homologous_rel_set_eq_relboundary
chain_map_ident [of _ X] singular_cycle cong: conj_cong)
also have  $\dots = ?rhs$ 
by blast
finally show ?thesis .
qed
then have kernel (relative_homology_group p X  $\{\}$ ) (relative_homology_group
p X S) (hom_induced p X  $\{\}$  X S id)
= hom_induced p (subtopology X S)  $\{\}$  X  $\{\}$  id ‘ carrier (relative_homology_group
p (subtopology X S)  $\{\}$ )
by (simp add: kernel_def carrier_relative_homology_group peq)
then show ?thesis
by (simp add: not_less group_hom_def group_hom_axioms_def hom_induced_hom)
qed

lemma homology_dimension_axiom:
assumes  $X: \text{topspace } X = \{a\}$  and  $p \neq 0$ 
shows trivial_group(homology_group p X)
proof (cases p < 0)
case True
then show ?thesis
by simp
next
case False
then obtain n where peq:  $p = \text{int } n \ n > 0$ 
by (metis assms(2) neq0_conv nonneg_int_cases not_less of_nat_0)
have homologous_rel_set n X  $\{\}$  ‘ singular_relcycle_set n X  $\{\}$  = {singular_relcycle_set
n X  $\{\}}$ 
(is ?lhs = ?rhs)
proof
show ?lhs  $\subseteq$  ?rhs
using peq assms
by (auto simp: image_subset_iff homologous_rel_set_eq_relboundary simp
flip: singular_boundary_set_eq_cycle_singleton)
have singular_relboundary n X  $\{\}$  0
by simp
with peq assms
show ?rhs  $\subseteq$  ?lhs
by (auto simp: image_iff simp flip: homologous_rel_eq_relboundary singular_
boundary_set_eq_cycle_singleton)

```

```

qed
with peq assms show ?thesis
  unfolding trivial_group_def
  by (simp add: carrier_relative_homology_group singular_boundary_set_eq_cycle_singleton
[OF X])
qed

```

```

proposition homology_homotopy_axiom:
  assumes homotopic_with ( $\lambda h. h \text{ ' } S \subseteq T$ )  $X \ Y \ f \ g$ 
  shows hom_induced  $p \ X \ S \ Y \ T \ f = \text{hom\_induced } p \ X \ S \ Y \ T \ g$ 
proof (cases  $p < 0$ )
  case True
  then show ?thesis
    by (simp add: hom_induced_trivial)
  next
  case False
  then obtain  $n$  where peq:  $p = \text{int } n$ 
    by (metis int_nat_eq not_le)
  have cont: continuous_map  $X \ Y \ f$  continuous_map  $X \ Y \ g$ 
    using assms homotopic_with_imp_continuous_maps by blast+
  have im:  $f \text{ ' } (\text{topspace } X \cap S) \subseteq T \ g \text{ ' } (\text{topspace } X \cap S) \subseteq T$ 
    using homotopic_with_imp_property assms by blast+
  show ?thesis
  proof
    fix  $c$  show hom_induced  $p \ X \ S \ Y \ T \ f \ c = \text{hom\_induced } p \ X \ S \ Y \ T \ g \ c$ 
    proof (cases  $c \in \text{carrier}(\text{relative\_homology\_group } p \ X \ S)$ )
    case True
    then obtain  $a$  where  $a: c = \text{homologous\_rel\_set } n \ X \ S \ a \ \text{singular\_relcycle}$ 
       $n \ X \ S \ a$ 
      unfolding carrier_relative_homology_group peq by auto
      with assms homotopic_imp_homologous_rel_chain_maps show ?thesis
        by (force simp add: peq hom_induced_chain_map_gen cont im homologous_rel_set_eq)
    qed (simp add: hom_induced_default)
    qed
  qed

```

```

proposition homology_excision_axiom:
  assumes  $X \ \text{closure\_of } U \subseteq X \ \text{interior\_of } T \ T \subseteq S$ 
  shows
    hom_induced  $p \ (\text{subtopology } X \ (S - U)) \ (T - U) \ (\text{subtopology } X \ S) \ T \ \text{id}$ 
       $\in \text{iso} \ (\text{relative\_homology\_group } p \ (\text{subtopology } X \ (S - U)) \ (T - U))$ 
         $(\text{relative\_homology\_group } p \ (\text{subtopology } X \ S) \ T)$ 
proof (cases  $p < 0$ )
  case True
  then show ?thesis
    unfolding iso_def bij_betw_def relative_homology_group_def by (simp add:
hom_induced_trivial)

```



```

next
  case False
  then obtain n where peq: p = int n
    by (metis int_nat_eq not_le)
  have cont: continuous_map (subtopology X (S - U)) (subtopology X S) id
    by (simp add: closure_of_subtopology_mono continuous_map_eq_image_closure_subset)
  have TU: topspace X  $\cap$  (S - U)  $\cap$  (T - U)  $\subseteq$  T
    by auto
  show ?thesis
    proof (simp add: iso_def peq carrier_relative_homology_group bij_betw_def
      hom_induced_hom, intro conjI)
      show inj_on (hom_induced n (subtopology X (S - U)) (T - U) (subtopology
        X S) T id)
        (homologous_rel_set n (subtopology X (S - U)) (T - U) '
          singular_relcycle_set n (subtopology X (S - U)) (T - U))
      unfolding inj_on_def
      proof (clarsimp simp add: homologous_rel_set_eq)
        fix c d
        assume c: singular_relcycle n (subtopology X (S - U)) (T - U) c
          and d: singular_relcycle n (subtopology X (S - U)) (T - U) d
          and hh: hom_induced n (subtopology X (S - U)) (T - U) (subtopology X
            S) T id
          (homologous_rel_set n (subtopology X (S - U)) (T - U) c)
          = hom_induced n (subtopology X (S - U)) (T - U) (subtopology X
            S) T id
          (homologous_rel_set n (subtopology X (S - U)) (T - U) d)
        then have scc: singular_chain n (subtopology X (S - U)) c
          and scd: singular_chain n (subtopology X (S - U)) d
          using singular_relcycle by blast+
        have singular_relboundary n (subtopology X (S - U)) (T - U) c
          if srb: singular_relboundary n (subtopology X S) T c
          and src: singular_relcycle n (subtopology X (S - U)) (T - U) c for c
        proof -
          have [simp]: (S - U)  $\cap$  (T - U) = T - U  $\cap$  S  $\cap$  T = T
            using  $\langle T \subseteq S \rangle$  by blast+
          have c: singular_chain n (subtopology X (S - U)) c
            singular_chain (n - Suc 0) (subtopology X (T - U)) (chain_boundary
              n c)
          using that by (auto simp: singular_relcycle_def mod_subset_def subtopol-
            ogy_subtopology)
          obtain d e where d: singular_chain (Suc n) (subtopology X S) d
            and e: singular_chain n (subtopology X T) e
            and dce: chain_boundary (Suc n) d = c + e
          using srb by (auto simp: singular_relboundary_alt subtopology_subtopology)
          obtain m f g where f: singular_chain (Suc n) (subtopology X (S - U)) f
            and g: singular_chain (Suc n) (subtopology X T) g
            and dfg: (singular_subdivision (Suc n)  $\widehat{\sim}$  m) d = f + g
          using excised_chain_exists [OF assms d] .
          obtain h where

```

```

    h0:  $\bigwedge p. h\ p\ 0 = (0 :: 'a\ chain)$ 
    and hdiff:  $\bigwedge p\ c1\ c2. h\ p\ (c1 - c2) = h\ p\ c1 - h\ p\ c2$ 
    and hSuc:  $\bigwedge p\ X\ c. singular\_chain\ p\ X\ c \implies singular\_chain\ (Suc\ p)\ X\ (h\ p\ c)$ 
    and hchain:  $\bigwedge p\ X\ c. singular\_chain\ p\ X\ c \implies chain\_boundary\ (Suc\ p)\ (h\ p\ c) + h\ (p - Suc\ 0)\ (chain\_boundary\ p\ c)$ 
               =  $(singular\_subdivision\ p\ \sim m)\ c - c$ 
    using chain_homotopic_iterated_singular_subdivision by blast
    have hadd:  $\bigwedge p\ c1\ c2. h\ p\ (c1 + c2) = h\ p\ c1 + h\ p\ c2$ 
    by (metis add_diff_cancel diff_add_cancel hdiff)
    define c1 where  $c1 \equiv f - h\ n\ c$ 
    define c2 where  $c2 \equiv chain\_boundary\ (Suc\ n)\ (h\ n\ e) - (chain\_boundary\ (Suc\ n)\ g - e)$ 
    show ?thesis
    unfolding singular_relboundary_alt
    proof (intro exI conjI)
    show c1:  $singular\_chain\ (Suc\ n)\ (subtopology\ X\ (S - U))\ c1$ 
    by (simp add:  $\langle singular\_chain\ n\ (subtopology\ X\ (S - U))\ c \rangle\ c1\_def\ f\ hSuc\ singular\_chain\_diff$ )
    have chain_boundary (Suc n) (chain_boundary (Suc (Suc n)) (h (Suc n) d) + h n (c+e))
    = chain_boundary (Suc n) (f + g - d)
    using hchain [OF d] by (simp add: dce dfg)
    then have chain_boundary (Suc n) (h n (c + e))
    = chain_boundary (Suc n) f + chain_boundary (Suc n) g - (c + e)
    using chain_boundary_boundary_alt [of Suc n subtopology X S]
    by (simp add: chain_boundary_add chain_boundary_diff d hSuc dce)
    then have chain_boundary (Suc n) (h n c) + chain_boundary (Suc n) (h n e)
    = chain_boundary (Suc n) f + chain_boundary (Suc n) g - (c + e)
    by (simp add: chain_boundary_add hadd)
    then have *:  $chain\_boundary\ (Suc\ n)\ (f - h\ n\ c) = c + (chain\_boundary\ (Suc\ n)\ (h\ n\ e) - (chain\_boundary\ (Suc\ n)\ g - e))$ 
    by (simp add: algebra_simps chain_boundary_diff)
    then show chain_boundary (Suc n) c1 = c + c2
    unfolding c1_def c2_def
    by (simp add: algebra_simps chain_boundary_diff)
    obtain singular_chain n (subtopology X (S - U)) c2 singular_chain n (subtopology X T) c2
    using singular_chain_diff c c1 *
    unfolding c1_def c2_def
    by (metis add_diff_cancel_left' e g hSuc singular_chain_boundary_alt)
    then show singular_chain n (subtopology (subtopology X (S - U)) (T - U)) c2
    by (fastforce simp add: singular_chain_subtopology)
    qed
    qed
    then have singular_relboundary n (subtopology X S) T (c - d)  $\implies$ 

```

```

      singular_relboundary n (subtopology X (S - U)) (T - U) (c - d)
    using c d singular_relcycle_diff by metis
  with hh show homologous_rel n (subtopology X (S - U)) (T - U) c d
    apply (simp add: hom_induced_chain_map cont c d chain_map_ident [OF
scc] chain_map_ident [OF scd])
    using homologous_rel_set_eq homologous_rel_def by metis
  qed
next
have h: homologous_rel_set n (subtopology X S) T a
  ∈ (λx. homologous_rel_set n (subtopology X S) T (chain_map n id x)) '
    singular_relcycle_set n (subtopology X (S - U)) (T - U)
  if a: singular_relcycle n (subtopology X S) T a for a
proof -
  obtain c' where c': singular_relcycle n (subtopology X (S - U)) (T - U)
c'
    homologous_rel n (subtopology X S) T a c'
  using a by (blast intro: excised_relcycle_exists [OF assms])
  then have scc': singular_chain n (subtopology X S) c'
    using homologous_rel_singular_chain singular_relcycle that by blast
  then show ?thesis
    using scc' chain_map_ident [of _ subtopology X S] c' homologous_rel_set_eq
    by fastforce
  qed
have (λx. homologous_rel_set n (subtopology X S) T (chain_map n id x)) '
    singular_relcycle_set n (subtopology X (S - U)) (T - U) =
    homologous_rel_set n (subtopology X S) T '
    singular_relcycle_set n (subtopology X S) T
  by (force simp: cont h singular_relcycle_chain_map)
then
show hom_induced n (subtopology X (S - U)) (T - U) (subtopology X S) T
id '
    homologous_rel_set n (subtopology X (S - U)) (T - U) '
    singular_relcycle_set n (subtopology X (S - U)) (T - U)
    = homologous_rel_set n (subtopology X S) T ' singular_relcycle_set n
(subtopology X S) T
  by (simp add: image_comp o_def hom_induced_chain_map_gen cont TU
topspace_subtopology
    cong: image_cong_simp)
  qed
qed

```

0.2.3 Additivity axiom

Not in the original Eilenberg-Steenrod list but usually included nowadays, following Milnor's "On Axiomatic Homology Theory".

lemma *iso_chain_group_sum*:

assumes *disj*: pairwise disjoint \mathcal{U} **and** UU : $\bigcup \mathcal{U} = \text{topspace } X$
and *subs*: $\bigwedge C T. \llbracket \text{compactin } X \ C; \text{ path_connectedin } X \ C; T \in \mathcal{U}; \sim \text{ disjoint } C \ T \rrbracket \implies C \subseteq T$

```

shows ( $\lambda f. \text{sum}' f \mathcal{U}$ )  $\in \text{iso}$  ( $\text{sum\_group } \mathcal{U}$  ( $\lambda S. \text{chain\_group } p$  ( $\text{subtopology } X$ 
 $S$ ))) ( $\text{chain\_group } p$   $X$ )
proof –
  have  $\text{pw}$ :  $\text{pairwise } (\lambda i j. \text{disjnt } (\text{singular\_simplex\_set } p$  ( $\text{subtopology } X$   $i$ ))
    ( $\text{singular\_simplex\_set } p$  ( $\text{subtopology } X$   $j$ )))  $\mathcal{U}$ 

  proof
    fix  $S$   $T$ 
    assume  $S \in \mathcal{U}$   $T \in \mathcal{U}$   $S \neq T$ 
    then show  $\text{disjnt } (\text{singular\_simplex\_set } p$  ( $\text{subtopology } X$   $S$ ))
      ( $\text{singular\_simplex\_set } p$  ( $\text{subtopology } X$   $T$ ))
      using  $\text{nonempty\_standard\_simplex [of } p] \text{ disj}$ 
      by ( $\text{fastforce simp: pairwise\_def disjnt\_def singular\_simplex\_subtopology}$ 
 $\text{image\_subset\_iff}$ )
    qed
    have  $\exists S \in \mathcal{U}. \text{singular\_simplex } p$  ( $\text{subtopology } X$   $S$ )  $f$ 
    if  $f$ :  $\text{singular\_simplex } p$   $X$   $f$  for  $f$ 
    proof –
      obtain  $x$  where  $x$ :  $x \in \text{topspace } X$   $x \in f$  ‘  $\text{standard\_simplex } p$ 
      using  $f \text{ nonempty\_standard\_simplex [of } p] \text{ continuous\_map\_image\_subset\_topspace}$ 
      unfolding  $\text{singular\_simplex\_def}$  by  $\text{fastforce}$ 
      then obtain  $S$  where  $S \in \mathcal{U}$   $x \in S$ 
      using  $UU$  by  $\text{auto}$ 
      have  $f$  ‘  $\text{standard\_simplex } p \subseteq S$ 
      proof ( $\text{rule subs}$ )
        have  $\text{cont}$ :  $\text{continuous\_map } (\text{subtopology } (\text{powertop\_real } UNIV)$ 
          ( $\text{standard\_simplex } p$ ))  $X$   $f$ 
          using  $f \text{ singular\_simplex\_def}$  by  $\text{auto}$ 
          show  $\text{compactin } X$  ( $f$  ‘  $\text{standard\_simplex } p$ )
          by ( $\text{simp add: compactin\_subtopology compactin\_standard\_simplex im-}$ 
 $\text{age\_compactin [OF\_ cont]}$ )
          show  $\text{path\_connectedin } X$  ( $f$  ‘  $\text{standard\_simplex } p$ )
          by ( $\text{simp add: path\_connectedin\_subtopology path\_connectedin\_standard\_simplex}$ 
 $\text{path\_connectedin\_continuous\_map\_image [OF cont]}$ )
          have  $\text{standard\_simplex } p \neq \{\}$ 
          by ( $\text{simp add: nonempty\_standard\_simplex}$ )
          then
          show  $\neg \text{disjnt } (f$  ‘  $\text{standard\_simplex } p$ )  $S$ 
          using  $x \in S$  by ( $\text{auto simp: disjnt\_def}$ )
        qed ( $\text{auto simp: } \langle S \in \mathcal{U} \rangle$ )
      then show  $?thesis$ 
      by ( $\text{meson } \langle S \in \mathcal{U} \rangle \text{ singular\_simplex\_subtopology that}$ )
    qed
    then have  $(\bigcup i \in \mathcal{U}. \text{singular\_simplex\_set } p$  ( $\text{subtopology } X$   $i$ )) =  $\text{singular\_simplex\_set}$ 
 $p$   $X$ 
    by ( $\text{auto simp: singular\_simplex\_subtopology}$ )
    then show  $?thesis$ 
    using  $\text{iso\_free\_Abelian\_group\_sum [OF pw]}$  by ( $\text{simp add: chain\_group\_def}$ )
  qed

```

lemma *relcycle_group_0_eq_chain_group*: $\text{relcycle_group } 0 \ X \ \{\} = \text{chain_group } 0 \ X$

proof (rule *monoid.equality*)

show $\text{carrier } (\text{relcycle_group } 0 \ X \ \{\}) = \text{carrier } (\text{chain_group } 0 \ X)$

by (simp add: *Collect_mono chain_boundary_def singular_cycle subset_antisym*)

qed (simp_all add: *relcycle_group_def chain_group_def*)

proposition *iso_cycle_group_sum*:

assumes *disj*: pairwise disjoint \mathcal{U} and UU : $\bigcup \mathcal{U} = \text{topspace } X$

and *subs*: $\bigwedge C \ T. [\text{compactin } X \ C; \text{path_connectedin } X \ C; T \in \mathcal{U}; \neg \text{disjnt } C \ T] \implies C \subseteq T$

shows $(\lambda f. \text{sum}' f \ \mathcal{U}) \in \text{iso } (\text{sum_group } \mathcal{U} \ (\lambda T. \text{relcycle_group } p \ (\text{subtopology } X \ T) \ \{\}))$

$(\text{relcycle_group } p \ X \ \{\})$

proof (cases $p = 0$)

case *True*

then show *?thesis*

by (simp add: *relcycle_group_0_eq_chain_group iso_chain_group_sum [OF assms]*)

next

case *False*

let $?SG = (\text{sum_group } \mathcal{U} \ (\lambda T. \text{chain_group } p \ (\text{subtopology } X \ T)))$

let $?PI = (\Pi_E \ T \in \mathcal{U}. \text{singular_relcycle_set } p \ (\text{subtopology } X \ T) \ \{\})$

have $(\lambda f. \text{sum}' f \ \mathcal{U}) \in \text{Group.iso } (\text{subgroup_generated } ?SG \ (\text{carrier } ?SG \cap ?PI))$

$(\text{subgroup_generated } (\text{chain_group } p \ X) \ (\text{singular_relcycle_set } p \ X \ \{\}))$

proof (rule *group_hom.iso_between_subgroups*)

have $\text{iso}: (\lambda f. \text{sum}' f \ \mathcal{U}) \in \text{Group.iso } ?SG \ (\text{chain_group } p \ X)$

by (auto simp: *assms iso_chain_group_sum*)

then show *group_hom* $?SG \ (\text{chain_group } p \ X) \ (\lambda f. \text{sum}' f \ \mathcal{U})$

by (auto simp: *iso_imp_homomorphism group_hom_def group_hom_axioms_def*)

have $B: \text{sum}' f \ \mathcal{U} \in \text{singular_relcycle_set } p \ X \ \{\} \longleftrightarrow f \in (\text{carrier } ?SG \cap ?PI)$

if $f \in (\text{carrier } ?SG)$ for f

proof –

have $f: \bigwedge S. S \in \mathcal{U} \longrightarrow \text{singular_chain } p \ (\text{subtopology } X \ S) \ (f \ S)$

$f \in \text{extensional } \mathcal{U} \text{ finite } \{i \in \mathcal{U}. f \ i \neq 0\}$

using that by (auto simp: *carrier_sum_group PiE_def Pi_def*)

then have *rfin*: $\text{finite } \{S \in \mathcal{U}. \text{restrict } (\text{chain_boundary } p \circ f) \ \mathcal{U} \ S \neq 0\}$

by (auto elim: *rev_finite_subset*)

have $\text{chain_boundary } p \ ((\sum x \mid x \in \mathcal{U} \wedge f \ x \neq 0. f \ x)) = 0$

$\longleftrightarrow (\forall S \in \mathcal{U}. \text{chain_boundary } p \ (f \ S) = 0) \ (\text{is } ?cb = 0 \longleftrightarrow ?rhs)$

proof

assume $?cb = 0$

moreover have $?cb = \text{sum}' (\lambda S. \text{chain_boundary } p \ (f \ S)) \ \mathcal{U}$

unfolding *sum.G_def* using *rfin* f

by (force simp: *chain_boundary_sum intro: sum.mono_neutral_right cong: conj_cong*)

ultimately have *eq0*: $\text{sum}' (\lambda S. \text{chain_boundary } p \ (f \ S)) \ \mathcal{U} = 0$

```

    by simp
    have  $(\lambda f. \text{sum}' f \mathcal{U}) \in \text{hom} (\text{sum\_group } \mathcal{U} (\lambda S. \text{chain\_group } (p - \text{Suc } 0) (\text{subtopology } X S)))$ 
       $(\text{chain\_group } (p - \text{Suc } 0) X)$ 
    and inj:  $\text{inj\_on } (\lambda f. \text{sum}' f \mathcal{U}) (\text{carrier } (\text{sum\_group } \mathcal{U} (\lambda S. \text{chain\_group } (p - \text{Suc } 0) (\text{subtopology } X S))))$ 
    using iso_chain_group_sum [OF assms, of p-1] by (auto simp: iso_def
    bij_betw_def)
    then have eq:  $\llbracket f \in (\Pi_E i \in \mathcal{U}. \text{singular\_chain\_set } (p - \text{Suc } 0) (\text{subtopology } X i)) \rrbracket$ 
       $\text{finite } \{S \in \mathcal{U}. f S \neq 0\}; \text{sum}' f \mathcal{U} = 0; S \in \mathcal{U} \rrbracket \implies f S = 0$  for  $f S$ 
    apply (simp add: group_hom_def group_hom_axioms_def group_hom.inj_on_one_iff
    [of _ chain_group (p-1) X])
    apply (auto simp: carrier_sum_group fun_eq_iff that)
    done
  show ?rhs
  proof clarify
    fix S assume  $S \in \mathcal{U}$ 
    then show  $\text{chain\_boundary } p (f S) = 0$ 
      using eq [of restrict (chain_boundary p  $\circ$  f)  $\mathcal{U}$  S] rfin f eq0
      by (simp add: singular_chain_boundary cong: conj_cong)
    qed
  next
    assume ?rhs
    then show  $?cb = 0$ 
      by (force simp: chain_boundary_sum intro: sum.mono_neutral_right)
    qed
  moreover
    have  $(\bigwedge S. S \in \mathcal{U} \longrightarrow \text{singular\_chain } p (\text{subtopology } X S) (f S))$ 
       $\implies \text{singular\_chain } p X (\sum x \mid x \in \mathcal{U} \wedge f x \neq 0. f x)$ 
    by (metis (no_types, lifting) mem_Collect_eq singular_chain_subtopology
    singular_chain_sum)
    ultimately show ?thesis
      using f by (auto simp: carrier_sum_group sum.G_def singular_cycle
    PiE_iff)
    qed
    have  $\text{singular\_relcycle\_set } p X \{\} \subseteq \text{carrier } (\text{chain\_group } p X)$ 
      using subgroup.subset subgroup_singular_relcycle by blast
    then show  $(\lambda f. \text{sum}' f \mathcal{U}) ' (\text{carrier } ?SG \cap ?PI) = \text{singular\_relcycle\_set } p X$ 
       $\{\}$ 
    using iso B unfolding Group.iso_def
      by (smt (verit, del_insts) Int_iff bij_betw_def image_iff mem_Collect_eq
    subset_antisym subset_iff)
    qed (auto simp: assms iso_chain_group_sum)
  then show ?thesis
    by (simp add: relcycle_group_def sum_group_subgroup_generated subgroup_singular_relcycle)
  qed

```

```

proposition homology_additivity_axiom_gen:
  assumes disj: pairwise disjnt  $\mathcal{U}$  and  $UU: \bigcup \mathcal{U} = \text{topspace } X$ 
  and subs:  $\bigwedge C \ T. \llbracket \text{compactin } X \ C; \text{ path\_connectedin } X \ C; T \in \mathcal{U}; \neg \text{disjnt } C \rrbracket \implies C \subseteq T$ 
  shows  $(\lambda x. \text{gfinprod } (\text{homology\_group } p \ X)$ 
     $(\lambda V. \text{hom\_induced } p \ (\text{subtopology } X \ V) \ \{\} \ X \ \{\} \ \text{id } (x \ V)) \ \mathcal{U})$ 
     $\in \text{iso } (\text{sum\_group } \mathcal{U} \ (\lambda S. \text{homology\_group } p \ (\text{subtopology } X \ S))) \ (\text{homology\_group } p \ X)$ 
    (is  $?h \in \text{iso } ?SG \ ?HG)$ 
proof (cases  $p < 0$ )
  case True
    then have [simp]:  $\text{gfinprod } (\text{singleton\_group } \text{undefined}) \ (\lambda v. \text{undefined}) \ \mathcal{U} = \text{undefined}$ 
    by (metis  $\text{Pi\_I carrier\_singleton\_group comm\_group\_def comm\_monoid.gfinprod\_closed singletonD singleton\_abelian\_group}$ )
    show ?thesis
    using True
    apply (simp add: iso_def relative_homology_group_def hom_induced_trivial carrier_sum_group)
    apply (auto simp: singleton_group_def bij_betw_def inj_on_def fun_eq_iff)
    done
  next
  case False
    then obtain  $n$  where  $\text{peq}: p = \text{int } n$ 
    by (metis  $\text{int\_ops}(1) \text{linorder\_neqE\_linordered\_idom pos\_int\_cases}$ )
    interpret  $\text{comm\_group homology\_group } p \ X$ 
    by (rule abelian_homology_group)
    show ?thesis
    proof (simp add: iso_def bij_betw_def, intro conjI)
      show  $?h \in \text{hom } ?SG \ ?HG$ 
      by (rule hom_group_sum) (simp_all add: hom_induced_hom)
      then interpret  $\text{group\_hom } ?SG \ ?HG \ ?h$ 
      by (simp add: group_hom_def group_hom_axioms_def)
      have  $\text{carrSG}: \text{carrier } ?SG$ 
       $= (\lambda x. \lambda S \in \mathcal{U}. \text{homologous\_rel\_set } n \ (\text{subtopology } X \ S) \ \{\} \ (x \ S))$ 
       $\quad \text{'(carrier } (\text{sum\_group } \mathcal{U} \ (\lambda S. \text{relcycle\_group } n \ (\text{subtopology } X \ S) \ \{\}))) \text{' (is$ 
       $?lhs = ?rhs)$ 
      proof
        show  $?lhs \subseteq ?rhs$ 
        proof (clarsimp simp: carrier_sum_group carrier_relative_homology_group peq)
          fix  $z$ 
          assume  $z: z \in (\Pi_E \ S \in \mathcal{U}. \text{homologous\_rel\_set } n \ (\text{subtopology } X \ S) \ \{\} \ \text{'singular\_relcycle\_set } n \ (\text{subtopology } X \ S) \ \{\})$ 
          and fin:  $\text{finite } \{S \in \mathcal{U}. z \ S \neq \text{singular\_relboundary\_set } n \ (\text{subtopology } X \ S) \ \{\}\}$ 
          then obtain  $c$  where  $c: \forall S \in \mathcal{U}. \text{singular\_relcycle } n \ (\text{subtopology } X \ S) \ \{\} \ (c \ S)$ 
           $\wedge z \ S = \text{homologous\_rel\_set } n \ (\text{subtopology } X \ S) \ \{\} \ (c \ S)$ 

```

```

    by (simp add: PiE_def Pi_def image_def) metis
  let ?f =  $\lambda S \in \mathcal{U}. \text{if singular\_relboundary } n \text{ (subtopology } X \text{ } S) \text{ \{ \} } (c \text{ } S) \text{ then } 0 \text{ else } c \text{ } S$ 
  have z = ( $\lambda S \in \mathcal{U}. \text{homologous\_rel\_set } n \text{ (subtopology } X \text{ } S) \text{ \{ \} } (?f \text{ } S)$ )
    by (smt (verit) PiE_restrict c homologous_rel_eq_relboundary restrict_apply restrict_ext singular_relboundary_0 z)
  moreover have  $?f \in (\Pi_E i \in \mathcal{U}. \text{singular\_relcycle\_set } n \text{ (subtopology } X \text{ } i) \text{ \{ \} })$ 
    by (simp add: c_fun_eq_iff PiE_arb [OF z])
  moreover have finite  $\{i \in \mathcal{U}. ?f \text{ } i \neq 0\}$ 
    using z c by (intro finite_subset [OF _ fin]) auto
  ultimately
  show  $z \in (\lambda x. \lambda S \in \mathcal{U}. \text{homologous\_rel\_set } n \text{ (subtopology } X \text{ } S) \text{ \{ \} } (x \text{ } S)) \text{ '}$ 
     $\{x \in \Pi_E i \in \mathcal{U}. \text{singular\_relcycle\_set } n \text{ (subtopology } X \text{ } i) \text{ \{ \} }. \text{finite } \{i \in \mathcal{U}. x \text{ } i \neq 0\}\}$ 
    by blast
  qed
  show  $?rhs \subseteq ?lhs$ 
    by (force simp: peq_carrier_sum_group carrier_relative_homology_group homologous_rel_set_eq_relboundary elim: rev_finite_subset)
  qed
  have gf: gfinprod (homology_group p X)
    ( $\lambda V. \text{hom\_induced } n \text{ (subtopology } X \text{ } V) \text{ \{ \} } X \text{ \{ \} } id$ 
      ( $(\lambda S \in \mathcal{U}. \text{homologous\_rel\_set } n \text{ (subtopology } X \text{ } S) \text{ \{ \} } (z \text{ } S)) \text{ } V$ ))
     $\mathcal{U}$ 
    = homologous_rel_set n X  $\{ \}$  (sum' z  $\mathcal{U}$ ) (is ?lhs = ?rhs)
  if z:  $z \in \text{carrier (sum\_group } \mathcal{U} \text{ } (\lambda S. \text{relcycle\_group } n \text{ (subtopology } X \text{ } S) \text{ \{ \} })))$ 
  for z
  proof -
    have hom_pi: ( $\lambda S. \text{homologous\_rel\_set } n \text{ } X \text{ \{ \} } (z \text{ } S)) \in \mathcal{U} \rightarrow \text{carrier (homology\_group } p \text{ } X)$ 
      using z
    by (intro Pi_I) (force simp: peq_carrier_sum_group carrier_relative_homology_group singular_chain_subtopology_singular_cycle)
    have fin: finite  $\{S \in \mathcal{U}. z \text{ } S \neq 0\}$ 
      using that by (force simp: carrier_sum_group)
    have  $?lhs = \text{gfinprod (homology\_group } p \text{ } X) (\lambda S. \text{homologous\_rel\_set } n \text{ } X \text{ \{ \} } (z \text{ } S)) \mathcal{U}$ 
      proof (rule gfinprod_cong [OF refl Pi_I])
        fix i
        show  $i \in \mathcal{U} \Rightarrow \text{hom\_induced (int } n) \text{ (subtopology } X \text{ } i) \text{ \{ \} } X \text{ \{ \} } id$ 
          ( $(\lambda S \in \mathcal{U}. \text{homologous\_rel\_set } n \text{ (subtopology } X \text{ } S) \text{ \{ \} } (z \text{ } S)) \text{ } i$ )
          = homologous_rel_set n X  $\{ \}$  (z i)
          using that
        by (auto simp: peq_simp_implies_def carrier_sum_group PiE_def Pi_def chain_map_ident singular_cycle hom_induced_chain_map)
      qed (simp add: hom_induced_carrier_peq)
    also have  $\dots = \text{gfinprod (homology\_group } p \text{ } X)$ 

```



```

      (λS. homologous_rel_set n X {} (z S)) {S ∈ U. z S ≠ 0}
proof –
  have homologous_rel_set n X {} 0 = singular_relboundary_set n X {}
    by (metis homologous_rel_eq_relboundary singular_relboundary_0)
  with hom_pi peq show ?thesis
    by (intro gfinprod_mono_neutral_cong_right) auto
qed
also have ... = ?rhs
proof –
  have gfinprod (homology_group p X) (λS. homologous_rel_set n X {} (z
S)) F
    = homologous_rel_set n X {} (sum z F)
    if finite F F ⊆ {S ∈ U. z S ≠ 0} for F
    using that
  proof (induction F)
    case empty
      have 1homology_group p X = homologous_rel_set n X {} 0
      by (metis homologous_rel_eq_relboundary one_relative_homology_group
peq singular_relboundary_0)
      then show ?case
        by simp
    next
      case (insert S F)
      with z have pi: (λS. homologous_rel_set n X {} (z S)) ∈ F → carrier
(homology_group p X)
        homologous_rel_set n X {} (z S) ∈ carrier (homology_group p X)
      by (force simp: peq_carrier_sum_group carrier_relative_homology_group
singular_chain_subtopology singular_cycle)+
      have hom: homologous_rel_set n X {} (z S) ∈ carrier (homology_group
p X)
        using insert z
      by (force simp: peq_carrier_sum_group carrier_relative_homology_group
singular_chain_subtopology singular_cycle)
      show ?case
        using insert z
      proof (simp add: pi)
        have ∧x. homologous_rel n X {} (z S + sum z F) x
          ⇒ ∃ u v. homologous_rel n X {} (z S) u ∧ homologous_rel n X {} (sum
z F) v ∧ x = u + v
          by (metis (no_types, lifting) diff_add_cancel diff_diff_eq2 homolo-
gous_rel_def homologous_rel_refl)
        with insert z
        show homologous_rel_set n X {} (z S) ⊗homology_group p X homolo-
gous_rel_set n X {} (sum z F)
          = homologous_rel_set n X {} (z S + sum z F)
        using insert z by (auto simp: peq_homologous_rel_add_mult_relative_homology_group)
      qed
    qed
  with fin show ?thesis

```

```

      by (simp add: sum.G_def)
    qed
  finally show ?thesis .
  qed
  show inj_on ?h (carrier ?SG)
  proof (clarify simp add: inj_on_one_iff)
    fix x
    assume x: x ∈ carrier (sum_group U (λS. homology_group p (subtopology X S)))
    and 1: gfinprod (homology_group p X) (λV. hom_induced p (subtopology X V) {} X {} id (x V)) U
    = 1_homology_group p X
    have feq: (λS∈U. homologous_rel_set n (subtopology X S) {}) (z S))
    = (λS∈U. 1_homology_group p (subtopology X S))
    if z: z ∈ carrier (sum_group U (λS. relcycle_group n (subtopology X S) {}))
    and eq: homologous_rel_set n X {} (sum' z U) = 1_homology_group p X
  for z
  proof -
    have z ∈ (ΠE S∈U. singular_relcycle_set n (subtopology X S) {}) finite {S
    ∈ U. z S ≠ 0}
    using z by (auto simp: carrier_sum_group)
    have singular_relboundary n X {} (sum' z U)
    using eq singular_chain_imp_relboundary by (auto simp: relative_homology_group_def
    peq)
    then obtain d where scd: singular_chain (Suc n) X d and cbd: chain_boundary
    (Suc n) d = sum' z U
    by (auto simp: singular_boundary)
    have *: ∃ d. singular_chain (Suc n) (subtopology X S) d ∧ chain_boundary
    (Suc n) d = z S
    if S ∈ U for S
    proof -
      have inj': inj_on (λf. sum' f U) {x ∈ ΠE S∈U. singular_chain_set (Suc
    n) (subtopology X S). finite {S ∈ U. x S ≠ 0}}
      using iso_chain_group_sum [OF assms, of Suc n]
      by (simp add: iso_iff_mon_epi mon_def carrier_sum_group)
      obtain w where w: w ∈ (ΠE S∈U. singular_chain_set (Suc n) (subtopology
    X S))
      and finw: finite {S ∈ U. w S ≠ 0}
      and deq: d = sum' w U
      using iso_chain_group_sum [OF assms, of Suc n] scd
      by (auto simp: iso_iff_mon_epi epi_def carrier_sum_group set_eq_iff)
      with ⟨S ∈ U⟩ have scwS: singular_chain (Suc n) (subtopology X S) (w S)
      by blast
      have inj_on (λf. sum' f U) {x ∈ ΠE S∈U. singular_chain_set n
    (subtopology X S). finite {S ∈ U. x S ≠ 0}}
      using iso_chain_group_sum [OF assms, of n]
      by (simp add: iso_iff_mon_epi mon_def carrier_sum_group)
      then have (λS∈U. chain_boundary (Suc n) (w S)) = z
      proof (rule inj_onD)

```

```

      have sum' ( $\lambda S \in \mathcal{U}. \text{chain\_boundary } (\text{Suc } n) (w S)) \mathcal{U} = \text{sum}' (\text{chain\_boundary }
(\text{Suc } n) \circ w) \{S \in \mathcal{U}. w S \neq 0\}
      by (auto simp: o_def intro: sum.mono_neutral_right')
      also have ... = chain_boundary (Suc n) d
      by (auto simp: sum.G_def deq chain_boundary_sum finw intro:
finite_subset [OF _ finw] sum.mono_neutral_left)
      finally show sum' ( $\lambda S \in \mathcal{U}. \text{chain\_boundary } (\text{Suc } n) (w S)) \mathcal{U} = \text{sum}' z
\mathcal{U}
      by (simp add: cbd)
      show ( $\lambda S \in \mathcal{U}. \text{chain\_boundary } (\text{Suc } n) (w S)) \in \{x \in \Pi_E \ S \in \mathcal{U}.
singular\_chain\_set \ n \ (\text{subtopology } X \ S). \text{finite } \{S \in \mathcal{U}. x S \neq 0\}\}
      using w by (auto simp: PiE_iff singular_chain_boundary_alt cong:
rev_conj_cong intro: finite_subset [OF _ finw])
      show  $z \in \{x \in \Pi_E \ S \in \mathcal{U}. singular\_chain\_set \ n \ (\text{subtopology } X \ S). \text{finite }
\{S \in \mathcal{U}. x S \neq 0\}\}$ 
      using z by (simp_all add: carrier_sum_group PiE_iff singular_cycle)
      qed
      with  $\langle S \in \mathcal{U} \rangle \text{ scwS}$  show ?thesis
      by force
      qed
      show ?thesis
      using that *
      by (force intro!: restrict_ext simp add: singular_boundary_relative_homology_group_def
homologous_rel_set_eq_relboundary peg)
      qed
      show  $x = (\lambda S \in \mathcal{U}. \mathbf{1}_{\text{homology\_group } p \ (\text{subtopology } X \ S)})$ 
      using  $x \ 1 \ \text{carrSG} \ \text{gf}$ 
      by (auto simp: peg feq)
      qed
      show ?h 'carrier ?SG = carrier ?HG
      proof safe
        fix A
        assume  $A \in \text{carrier } (\text{homology\_group } p \ X)$ 
        then obtain y where  $y: singular\_relcycle \ n \ X \ \{\} \ y$  and  $xeq: A = \text{homolo-}
gous\_rel\_set \ n \ X \ \{\} \ y$ 
        by (auto simp: peg carrier_relative_homology_group)
        then obtain x where  $x \in \text{carrier } (\text{sum\_group } \mathcal{U} \ (\lambda T. \text{relcycle\_group } n
(\text{subtopology } X \ T) \ \{\}))$ 
        and  $y = \text{sum}' x \ \mathcal{U}$ 
        using iso_cycle_group_sum [OF assms, of n] that by (force simp: iso_iff_mon_epi
epi_def)
        then show  $A \in (\lambda x. \text{gfinprod } (\text{homology\_group } p \ X) \ (\lambda V. \text{hom\_induced } p
(\text{subtopology } X \ V) \ \{\} \ X \ \{\} \ \text{id } (x \ V)) \ \mathcal{U}) \ '
        carrier (\text{sum\_group } \mathcal{U} \ (\lambda S. \text{homology\_group } p \ (\text{subtopology } X \ S)))$ 
        apply (simp add: carrSG image_comp o_def xeq)
        apply (simp add: hom_induced_carrier peg flip: gf cong: gfinprod_cong)
        done
      qed auto
    qed
  qed$$$ 
```

qed

```

corollary homology_additivity_axiom:
  assumes disj: pairwise disjnt  $\mathcal{U}$  and  $UU: \bigcup \mathcal{U} = \text{topspace } X$ 
  and ope:  $\bigwedge v. v \in \mathcal{U} \implies \text{openin } X \ v$ 
  shows  $(\lambda x. \text{gfinprod } (\text{homology\_group } p \ X)$ 
     $(\lambda v. \text{hom\_induced } p \ (\text{subtopology } X \ v) \ \{\} \ X \ \{\} \ \text{id } (x \ v)) \ \mathcal{U})$ 
     $\in \text{iso } (\text{sum\_group } \mathcal{U} \ (\lambda S. \text{homology\_group } p \ (\text{subtopology } X \ S))) \ (\text{homology\_group } p \ X)$ 
  proof (rule homology_additivity_axiom_gen [OF disj UU])
    fix  $C \ T$ 
    assume
      compactin  $X \ C$  and
      path_connectedin  $X \ C$  and
       $T \in \mathcal{U}$  and
       $\neg \text{disjnt } C \ T$ 
    then have  $\ast: \bigwedge B. [\text{openin } X \ T; T \cap B \cap C = \{\}; C \subseteq T \cup B; \text{openin } X \ B]$ 
     $\implies B \cap C = \{\}$ 
    by (meson connectedin disjnt_def disjnt_sym path_connectedin_imp_connectedin)
    have  $C \subseteq \text{Union } \mathcal{U}$ 
    by (simp add: UU <compactin X C> compactin_subset_topspace)
    moreover have  $\bigcup (\mathcal{U} - \{T\}) \cap C = \{\}$ 
    proof (rule *)
      show  $T \cap \bigcup (\mathcal{U} - \{T\}) \cap C = \{\}$ 
      using  $\langle T \in \mathcal{U} \rangle \text{ disj disjointD}$  by fastforce
      show  $C \subseteq T \cup \bigcup (\mathcal{U} - \{T\})$ 
      using  $\langle C \subseteq \bigcup \mathcal{U} \rangle$  by fastforce
    qed (auto simp: <T ∈ U> ope)
    ultimately show  $C \subseteq T$ 
    by blast
  qed

```

0.2.4 Special properties of singular homology

In particular: the zeroth homology group is isomorphic to the free abelian group generated by the path components. So, the "coefficient group" is the integers.

```

lemma iso_integer_zeroth_homology_group_aux:
  assumes  $X$ : path_connected_space  $X$  and  $f$ : singular_simplex  $0 \ X \ f$  and  $f'$ :
singular_simplex  $0 \ X \ f'$ 
  shows homologous_rel  $0 \ X \ \{\} \ (\text{frag\_of } f) \ (\text{frag\_of } f')$ 
  proof –
    let  $?p = \lambda j. \text{if } j = 0 \text{ then } 1 \text{ else } 0$ 
    have  $f \ ?p \in \text{topspace } X \ f' \ ?p \in \text{topspace } X$ 
    using assms by (auto simp: singular_simplex_def continuous_map_def)
    then obtain  $g$  where  $g$ : pathin  $X \ g$ 
      and  $g0$ :  $g \ 0 = f \ ?p$ 
      and  $g1$ :  $g \ 1 = f' \ ?p$ 

```

```

    using assms by (force simp: path_connected_space_def)
  then have contg: continuous_map (subtopology euclideanreal {0..1}) X g
    by (simp add: pathin_def)
  have singular_chain (Suc 0) X (frag_of (restrict (g ∘ (λx. x 0)) (standard_simplex
1)))
  proof -
    have continuous_map (subtopology (powertop_real UNIV) (standard_simplex
(Suc 0)))
      euclideanreal (λx. x 0)
    by (metis (mono_tags) UNIV_I continuous_map_from_subtopology contin-
uous_map_product_projection)
    then have continuous_map (subtopology (powertop_real UNIV) (standard_simplex
(Suc 0)))
      (top_of_set {0..1}) (λx. x 0)
    unfolding continuous_map_in_subtopology g
    by (auto simp: continuous_map_in_subtopology standard_simplex_def g)
    moreover have continuous_map (top_of_set {0..1}) X g
    using contg by blast
    ultimately show ?thesis
    by (force simp: singular_chain_of chain_boundary_of singular_simplex_def
continuous_map_compose)
  qed
  moreover
  have chain_boundary (Suc 0) (frag_of (restrict (g ∘ (λx. x 0)) (standard_simplex
1))) =
    frag_of f - frag_of f'
  proof -
    have singular_face (Suc 0) 0 (g ∘ (λx. x 0)) = f
      singular_face (Suc 0) (Suc 0) (g ∘ (λx. x 0)) = f'
    using assms
    by (auto simp: singular_face_def singular_simplex_def extensional_def sim-
plical_face_def standard_simplex_0 g0 g1)
    then show ?thesis
    by (simp add: singular_chain_of chain_boundary_of)
  qed
  ultimately
  show ?thesis
  by (auto simp: homologous_rel_def singular_boundary)
qed

```

proposition *iso_integer_zeroth_homology_group:*

assumes X : path_connected_space X **and** f : singular_simplex 0 X f
shows pow (homology_group 0 X) (homologous_rel_set 0 X { $\{$ } (frag_of f))
 \in iso_integer_group (homology_group 0 X) (**is** pow ? H ? $q \in$ iso _ ? H)

proof -

have srf: singular_relcycle 0 X { $\{$ } (frag_of f)
by (simp add: chain_boundary_def f singular_chain_of singular_cycle)
then have qcarr: ? $q \in$ carrier ? H

```

    by (simp add: carrier_relative_homology_group_0)
  have 1: homologous_rel_set 0 X {} a ∈ range (λn. homologous_rel_set 0 X {}
(frag_cmulo n (frag_of f)))
    if singular_relcycle 0 X {} a for a
  proof -
    have singular_chain 0 X d ==>
      homologous_rel_set 0 X {} d ∈ range (λn. homologous_rel_set 0 X {}
(frag_cmulo n (frag_of f))) for d
      unfolding singular_chain_def
    proof (induction d rule: frag_induction)
      case zero
      then show ?case
        by (metis frag_cmulo_zero rangeI)
    next
      case (one x)
      then have ∃ i. homologous_rel_set 0 X {} (frag_cmulo i (frag_of f))
        = homologous_rel_set 0 X {} (frag_of x)
        by (metis (no_types) iso_integer_zeroth_homology_group_aux [OF X] f
frag_cmulo_one homologous_rel_eq mem_Collect_eq)
        with one show ?case
          by auto
    next
      case (diff a b)
      then obtain c d where
        homologous_rel 0 X {} (a - b) (frag_cmulo c (frag_of f) - frag_cmulo d
(frag_of f))
        using homologous_rel_diff by (fastforce simp add: homologous_rel_set_eq)
      then show ?case
        by (rule_tac x=c-d in image_eqI) (auto simp: homologous_rel_set_eq
frag_cmulo_diff_distrib)
      qed
    with that show ?thesis
      unfolding singular_relcycle_def by blast
  qed
  have 2: n = 0
  if homologous_rel_set 0 X {} (frag_cmulo n (frag_of f)) = 1relative_homology_group 0 X {}
  for n
  proof -
    have singular_chain (Suc 0) X d
      ==> frag_extend (λx. frag_of f) (chain_boundary (Suc 0) d) = 0 for d
      unfolding singular_chain_def
    proof (induction d rule: frag_induction)
      case (one x)
      then show ?case
        by (simp add: frag_extend_diff_chain_boundary_of)
    next
      case (diff a b)
      then show ?case
        by (simp add: chain_boundary_diff frag_extend_diff)
  qed

```

```

qed auto
with that show ?thesis
  by (force simp: singular_boundary relative_homology_group_def homologous_rel_set_eq relboundary frag_extend_cmul)
qed
interpret GH : group_hom integer_group ?H ( $\bigwedge ?H$ ) ?q
  by (simp add: group_hom_def group_hom_axioms_def qcarr group_hom_integer_group_pow)
  have eq: pow ?H ?q = ( $\lambda n$ . homologous_rel_set 0 X  $\{\}$ ) (frag_cmul n (frag_of f)))
proof
  fix n
  have frag_of f
     $\in$  carrier (subgroup_generated
      (free_Abelian_group (singular_simplex_set 0 X)) (singular_relcycle_set 0 X  $\{\}$ ))
    by (metis carrier_relcycle_group chain_group_def mem_Collect_eq relcycle_group_def srf)
    then have ff: frag_of f  $\bigwedge$  relcycle_group 0 X  $\{\}$  n = frag_cmul n (frag_of f)
    by (simp add: relcycle_group_def chain_group_def group.int_pow_subgroup_generated f)
    show pow ?H ?q n = homologous_rel_set 0 X  $\{\}$  (frag_cmul n (frag_of f))
    apply (rule subst [OF right_coset_singular_relboundary])
    by (simp add: ff normal.FactGroup_int_pow normal_subgroup_singular_relboundary_relcycle relative_homology_group_def srf)
qed
show ?thesis
  apply (subst GH.iso_iff)
  apply (simp add: eq)
  apply (auto simp: carrier_relative_homology_group_0 1 2)
  done
qed

```

corollary *isomorphic_integer_zeroth_homology_group*:

assumes *X*: *path_connected_space X* *topspace X* $\neq \{\}$

shows *homology_group 0 X* \cong *integer_group*

proof –

obtain *a* **where** *a*: *a* \in *topspace X*

using *assms* **by** *blast*

have *singular_simplex 0 X* (*restrict* (λx . *a*) (*standard_simplex 0*))

by (*simp add: singular_simplex_def a*)

then show *?thesis*

using *X group.iso_sym group_integer_group is_isoI iso_integer_zeroth_homology_group*

by *blast*

qed

corollary *homology_coefficients*:

topspace X = $\{a\} \implies$ *homology_group 0 X* \cong *integer_group*

using *isomorphic_integer_zeroth_homology_group_path_connectedin_topspace*
by *fastforce*

proposition *zeroth_homology_group*:

homology_group 0 X \cong *free_Abelian_group (path_components_of X)*

proof –

obtain *h* **where** *h*: *h* \in *iso (sum_group (path_components_of X) (λS . homology_group 0 (subtopology X S)))*

(homology_group 0 X)

proof (*rule that [OF homology_additivity_axiom_gen]*)

show *disjoint (path_components_of X)*

by (*simp add: pairwise_disjoint_path_components_of*)

show $\bigcup (\text{path_components_of } X) = \text{topspace } X$

by (*rule Union_path_components_of*)

next

fix *C T*

assume *path_connectedin X C T* \in *path_components_of X* \neg *disjnt C T*

then show *C* \subseteq *T*

by (*metis path_components_of_maximal_disjnt_sym*) +

qed

have *homology_group 0 X* \cong *sum_group (path_components_of X) (λS . homology_group 0 (subtopology X S))*

by (*rule group_iso_sym*) (*use h is_iso_def in auto*)

also have $\dots \cong$ *sum_group (path_components_of X) (λi . integer_group)*

proof (*rule iso_sum_groupI*)

show *homology_group 0 (subtopology X i)* \cong *integer_group* **if** *i* \in *path_components_of X* **for** *i*

by (*metis that isomorphic_integer_zeroth_homology_group_nonempty_path_components_of_path_connectedin_def path_connectedin_path_components_of_topspace_subtopology_subset*)

qed *auto*

also have $\dots \cong$ *free_Abelian_group (path_components_of X)*

using *path_connectedin_path_components_of_nonempty_path_components_of*

by (*simp add: isomorphic_sum_integer_group_path_connectedin_def*)

finally show *?thesis* .

qed

lemma *isomorphic_homology_imp_path_components*:

assumes *homology_group 0 X* \cong *homology_group 0 Y*

shows *path_components_of X* \approx *path_components_of Y*

proof –

have *free_Abelian_group (path_components_of X)* \cong *homology_group 0 X*

by (*rule group_iso_sym*) (*auto simp: zeroth_homology_group*)

also have $\dots \cong$ *homology_group 0 Y*

by (*rule assms*)

also have $\dots \cong$ *free_Abelian_group (path_components_of Y)*

by (*rule zeroth_homology_group*)

finally have *free_Abelian_group (path_components_of X)* \cong *free_Abelian_group (path_components_of Y)* .


```

then show ?thesis
  by (simp add: isomorphic_free_Abelian_groups)
qed

```

```

lemma isomorphic_homology_imp_path_connectedness:
  assumes homology_group 0 X  $\cong$  homology_group 0 Y
  shows path_connected_space X  $\longleftrightarrow$  path_connected_space Y
proof -
  obtain h where h: bij_betw h (path_components_of X) (path_components_of
    Y)
  using assms isomorphic_homology_imp_path_components_eqpoll_def by blast
  have 1: path_components_of X  $\subseteq$  {a}  $\implies$  path_components_of Y  $\subseteq$  {h a} for
    a
  using h unfolding bij_betw_def by blast
  have 2: path_components_of Y  $\subseteq$  {a}
     $\implies$  path_components_of X  $\subseteq$  {inv_into (path_components_of X) h a}
for a
  using h [THEN bij_betw_inv_into] unfolding bij_betw_def by blast
  show ?thesis
  unfolding path_connected_space_iff_components_subset_singleton
  by (blast intro: dest: 1 2)
qed

```

0.2.5 More basic properties of homology groups, deduced from the E-S axioms

```

lemma trivial_homology_group:
   $p < 0 \implies$  trivial_group(homology_group p X)
  by simp

```

```

lemma hom_induced_empty_hom:
   $(\text{hom\_induced } p \text{ X } \{\} \text{ X' } \{\} f) \in \text{hom } (\text{homology\_group } p \text{ X}) (\text{homology\_group } p \text{ X'})$ 
  by (simp add: hom_induced_hom)

```

```

lemma hom_induced_compose_empty:
   $\llbracket \text{continuous\_map } X \text{ Y } f; \text{continuous\_map } Y \text{ Z } g \rrbracket$ 
   $\implies \text{hom\_induced } p \text{ X } \{\} \text{ Z } \{\} (g \circ f) = \text{hom\_induced } p \text{ Y } \{\} \text{ Z } \{\} g \circ$ 
 $\text{hom\_induced } p \text{ X } \{\} \text{ Y } \{\} f$ 
  by (simp add: hom_induced_compose)

```

```

lemma homology_homotopy_empty:
   $\text{homotopic\_with } (\lambda h. \text{True}) \text{ X Y } f g \implies \text{hom\_induced } p \text{ X } \{\} \text{ Y } \{\} f =$ 
 $\text{hom\_induced } p \text{ X } \{\} \text{ Y } \{\} g$ 
  by (simp add: homology_homotopy_axiom)

```

```

lemma homotopy_equivalence_relative_homology_group_isomorphisms:
  assumes contf: continuous_map X Y f and fim:  $f' \text{ ` } S \subseteq T$ 

```

```

    and contg: continuous_map Y X g and gim: g ' T ⊆ S
    and gf: homotopic_with (λh. h ' S ⊆ S) X X (g ∘ f) id
    and fg: homotopic_with (λk. k ' T ⊆ T) Y Y (f ∘ g) id
    shows group_isomorphisms (relative_homology_group p X S) (relative_homology_group
p Y T)
      (hom_induced p X S Y T f) (hom_induced p Y T X S g)
    unfolding group_isomorphisms_def
  proof (intro conjI ballI)
    fix x
    assume x: x ∈ carrier (relative_homology_group p X S)
    then show hom_induced p Y T X S g (hom_induced p X S Y T f x) = x
      using homology_homotopy_axiom [OF gf, of p]
      by (simp add: contf contg fim gim hom_induced_compose' hom_induced_id)
    next
    fix y
    assume y ∈ carrier (relative_homology_group p Y T)
    then show hom_induced p X S Y T f (hom_induced p Y T X S g y) = y
      using homology_homotopy_axiom [OF fg, of p]
      by (simp add: contf contg fim gim hom_induced_compose' hom_induced_id)
  qed (auto simp: hom_induced_hom)

```

```

lemma homotopy_equivalence_relative_homology_group_isomorphism:
  assumes continuous_map X Y f and fim: f ' S ⊆ T
    and continuous_map Y X g and gim: g ' T ⊆ S
    and homotopic_with (λh. h ' S ⊆ S) X X (g ∘ f) id
    and homotopic_with (λk. k ' T ⊆ T) Y Y (f ∘ g) id
  shows (hom_induced p X S Y T f) ∈ iso (relative_homology_group p X S)
(relative_homology_group p Y T)
  using homotopy_equivalence_relative_homology_group_isomorphisms [OF assms]
group_isomorphisms_imp_iso
  by metis

```

```

lemma homotopy_equivalence_homology_group_isomorphism:
  assumes continuous_map X Y f
    and continuous_map Y X g
    and homotopic_with (λh. True) X X (g ∘ f) id
    and homotopic_with (λk. True) Y Y (f ∘ g) id
  shows (hom_induced p X {} Y {} f) ∈ iso (homology_group p X) (homology_group
p Y)
  using assms by (intro homotopy_equivalence_relative_homology_group_isomorphism)
auto

```

```

lemma homotopy_equivalent_space_imp_isomorphic_relative_homology_groups:
  assumes continuous_map X Y f and fim: f ' S ⊆ T
    and continuous_map Y X g and gim: g ' T ⊆ S
    and homotopic_with (λh. h ' S ⊆ S) X X (g ∘ f) id
    and homotopic_with (λk. k ' T ⊆ T) Y Y (f ∘ g) id
  shows relative_homology_group p X S ≅ relative_homology_group p Y T

```

```

using homotopy_equivalence_relative_homology_group_isomorphism [OF assms]
unfolding is_iso_def by blast

```

```

lemma homotopy_equivalent_space_imp_isomorphic_homology_groups:
   $X \text{ homotopy\_equivalent\_space } Y \implies \text{homology\_group } p \, X \cong \text{homology\_group } p \, Y$ 
unfolding homotopy_equivalent_space_def
by (auto intro: homotopy_equivalent_space_imp_isomorphic_relative_homology_groups)

```

```

lemma homeomorphic_space_imp_isomorphic_homology_groups:
   $X \text{ homeomorphic\_space } Y \implies \text{homology\_group } p \, X \cong \text{homology\_group } p \, Y$ 
by (simp add: homeomorphic_imp_homotopy_equivalent_space homotopy_equivalent_space_imp_isomorphic_ho

```

```

lemma trivial_relative_homology_group_gen:
  assumes continuous_map  $X$  (subtopology  $X \, S$ )  $f$ 
    homotopic_with ( $\lambda h. \text{True}$ ) (subtopology  $X \, S$ ) (subtopology  $X \, S$ )  $f \, id$ 
    homotopic_with ( $\lambda k. \text{True}$ )  $X \, X \, f \, id$ 
  shows trivial_group(relative_homology_group  $p \, X \, S$ )
proof (rule exact_seq_imp_triviality)
  show exact_seq ([homology_group ( $p-1$ )  $X$ ,
    homology_group ( $p-1$ ) (subtopology  $X \, S$ ),
    relative_homology_group  $p \, X \, S$ , homology_group  $p \, X$ , homol-
ogy_group  $p$  (subtopology  $X \, S$ )],
    [hom_induced ( $p-1$ ) (subtopology  $X \, S$ ) {}  $X$  {}  $id$ ,
    hom_boundary  $p \, X \, S$ ,
    hom_induced  $p \, X$  {}  $X \, S \, id$ ,
    hom_induced  $p$  (subtopology  $X \, S$ ) {}  $X$  {}  $id$ ])
  using homology_exactness_axiom_1 homology_exactness_axiom_2 homol-
ogy_exactness_axiom_3
  by (metis exact_seq_cons_iff)
next
  show hom_induced  $p$  (subtopology  $X \, S$ ) {}  $X$  {}  $id$ 
     $\in \text{iso} (\text{homology\_group } p \, (\text{subtopology } X \, S)) (\text{homology\_group } p \, X)$ 
    hom_induced ( $p-1$ ) (subtopology  $X \, S$ ) {}  $X$  {}  $id$ 
     $\in \text{iso} (\text{homology\_group } (p-1) \, (\text{subtopology } X \, S)) (\text{homology\_group } (p-1) \, X)$ 
  using assms
  by (auto intro: homotopy_equivalence_relative_homology_group_isomorphism)
qed

```

```

lemma trivial_relative_homology_group_topospace:
  trivial_group(relative_homology_group  $p \, X$  (topspace  $X$ ))
by (rule trivial_relative_homology_group_gen [where  $f=id$ ]) auto

```

```

lemma trivial_relative_homology_group_empty:
  topspace  $X = \{\} \implies \text{trivial\_group}(\text{relative\_homology\_group } p \, X \, S)$ 
by (metis Int_absorb2 empty_subsetI relative_homology_group_restrict trivial_relative_homology_group_topospace)

```

lemma *trivial_homology_group_empty*:

topspace X = {} \implies trivial_group(homology_group p X)
by (simp add: trivial_relative_homology_group_empty)

lemma *homeomorphic_maps_relative_homology_group_isomorphisms*:

assumes *homeomorphic_maps X Y f g* **and** *im: f ' S \subseteq T g ' T \subseteq S*
shows *group_isomorphisms (relative_homology_group p X S) (relative_homology_group p Y T)*

(hom_induced p X S Y T f) (hom_induced p Y T X S g)

proof –

have *fg: continuous_map X Y f continuous_map Y X g*
 $(\forall x \in \text{topspace } X. g (f x) = x) (\forall y \in \text{topspace } Y. f (g y) = y)$

using *assms* **by** (simp_all add: homeomorphic_maps_def)

have *group_isomorphisms*

(relative_homology_group p X (topspace X \cap S))
 (relative_homology_group p Y (topspace Y \cap T))
 (hom_induced p X (topspace X \cap S) Y (topspace Y \cap T) f)
 (hom_induced p Y (topspace Y \cap T) X (topspace X \cap S) g)

proof (rule homotopy_equivalence_relative_homology_group_isomorphisms)

show *homotopic_with* $(\lambda h. h ' (\text{topspace } X \cap S) \subseteq \text{topspace } X \cap S) X X (g \circ f) \text{ id}$

using *fg im* **by** (auto intro: homotopic_with_equal_continuous_map_compose)

next

show *homotopic_with* $(\lambda k. k ' (\text{topspace } Y \cap T) \subseteq \text{topspace } Y \cap T) Y Y (f \circ g) \text{ id}$

using *fg im* **by** (auto intro: homotopic_with_equal_continuous_map_compose)

qed (use *im fg* in $\langle \text{auto simp: continuous_map_def} \rangle$)

then show *?thesis*

by *simp*

qed

lemma *homeomorphic_map_relative_homology_iso*:

assumes *f: homeomorphic_map X Y f* **and** *S: S \subseteq topspace X f ' S = T*

shows $(\text{hom_induced } p X S Y T f) \in \text{iso } (\text{relative_homology_group } p X S)$
 $(\text{relative_homology_group } p Y T)$

proof –

obtain *g* **where** *g: homeomorphic_maps X Y f g*

using *homeomorphic_map_maps f* **by** *metis*

then have *group_isomorphisms (relative_homology_group p X S) (relative_homology_group p Y T)*

(hom_induced p X S Y T f) (hom_induced p Y T X S g)

using *S g* **by** (auto simp: homeomorphic_maps_def intro!: homeomorphic_maps_relative_homology_g

then show *?thesis*

by (rule group_isomorphisms_imp_iso)

qed

lemma *inj_on_hom_induced_section_map*:

assumes *section_map X Y f*

```

shows inj_on (hom_induced p X {} Y {} f) (carrier (homology_group p X))
proof –
  obtain g where cont: continuous_map X Y f continuous_map Y X g
    and gf:  $\bigwedge x. x \in \text{topspace } X \implies g (f x) = x$ 
    using assms by (auto simp: section_map_def retraction_maps_def)
  show ?thesis
  proof (rule inj_on_inverseI)
    fix x
    assume x: x  $\in$  carrier (homology_group p X)
    have continuous_map X X ( $\lambda x. g (f x)$ )
      by (metis (no_types, lifting) continuous_map_eq continuous_map_id gf
id_apply)
    with x show hom_induced p Y {} X {} g (hom_induced p X {} Y {} f x) =
x
    using hom_induced_compose_empty [OF cont, symmetric]
    by (metis comp_apply cont continuous_map_compose gf hom_induced_id_gen)
  qed
qed

corollary mon_hom_induced_section_map:
  assumes section_map X Y f
  shows (hom_induced p X {} Y {} f)  $\in$  mon (homology_group p X) (homology_group
p Y)
  by (simp add: hom_induced_empty_hom inj_on_hom_induced_section_map
[OF assms] mon_def)

lemma surj_hom_induced_retraction_map:
  assumes retraction_map X Y f
  shows carrier (homology_group p Y) = (hom_induced p X {} Y {} f) ‘carrier
(homology_group p X)
    (is ?lhs = ?rhs)
proof –
  obtain g where cont: continuous_map Y X g continuous_map X Y f
    and fg:  $\bigwedge x. x \in \text{topspace } Y \implies f (g x) = x$ 
    using assms by (auto simp: retraction_map_def retraction_maps_def)
  have x = hom_induced p X {} Y {} f (hom_induced p Y {} X {} g x)
    if x: x  $\in$  carrier (homology_group p Y) for x
  proof –
    have continuous_map Y Y ( $\lambda x. f (g x)$ )
      by (metis (no_types, lifting) continuous_map_eq continuous_map_id fg
id_apply)
    with x show ?thesis
    using hom_induced_compose_empty [OF cont, symmetric]
    by (metis comp_def cont continuous_map_compose fg hom_induced_id_gen)
  qed
moreover
  have (hom_induced p Y {} X {} g x)  $\in$  carrier (homology_group p X)
    if x  $\in$  carrier (homology_group p Y) for x
    by (metis hom_induced)

```

```

ultimately have ?lhs  $\subseteq$  ?rhs
  by auto
moreover have ?rhs  $\subseteq$  ?lhs
  using hom_induced_hom [of p X {} Y {} f]
  by (simp add: hom_def flip: image_subset_iff_funcset)
ultimately show ?thesis
  by auto
qed

```

```

corollary epi_hom_induced_retraction_map:
  assumes retraction_map X Y f
  shows (hom_induced p X {} Y {} f)  $\in$  epi (homology_group p X) (homology_group
p Y)
  using assms epi_iff_subset hom_induced_empty_hom surj_hom_induced_retraction_map
  by fastforce

```

```

lemma homeomorphic_map_homology_iso:
  assumes homeomorphic_map X Y f
  shows (hom_induced p X {} Y {} f)  $\in$  iso (homology_group p X) (homology_group
p Y)
  using assms by (simp add: homeomorphic_map_relative_homology_iso)

```

```

lemma inj_on_hom_induced_inclusion:
  assumes S = {}  $\vee$  S retract_of_space X
  shows inj_on (hom_induced p (subtopology X S) {} X {} id) (carrier (homology_group
p (subtopology X S)))
  using assms
proof
  assume S = {}
  then have trivial_group (homology_group p (subtopology X S))
    by (auto simp: topspace_subtopology intro: trivial_homology_group_empty)
  then show ?thesis
    by (auto simp: inj_on_def trivial_group_def)
next
  assume S retract_of_space X
  then show ?thesis
    by (simp add: retract_of_space_section_map inj_on_hom_induced_section_map)
qed

```

```

lemma trivial_homomorphism_hom_boundary_inclusion:
  assumes S = {}  $\vee$  S retract_of_space X
  shows trivial_homomorphism
    (relative_homology_group p X S) (homology_group (p-1) (subtopology
X S))
    (hom_boundary p X S)
  using exact_seq_mon_eq_triviality inj_on_hom_induced_inclusion [OF assms]

```

by (metis exact_seq_cons_iff homology_exactness_axiom_1 homology_exactness_axiom_2)

lemma *epi_hom_induced_relativization*:

assumes $S = \{\}$ \vee S retract_of_space X

shows $(\text{hom_induced } p \ X \ \{\} \ X \ S \ \text{id}) \text{ ' carrier } (\text{homology_group } p \ X) = \text{carrier}$
 $(\text{relative_homology_group } p \ X \ S)$

using exact_seq_epi_eq_triviality trivial_homomorphism_hom_boundary_inclusion

by (metis assms exact_seq_cons_iff homology_exactness_axiom_1 homology_exactness_axiom_2)

lemmas short_exact_sequence_hom_induced_inclusion = homology_exactness_axiom_3

lemma *group_isomorphisms_homology_group_prod_retract*:

assumes $S = \{\}$ \vee S retract_of_space X

obtains $\mathcal{H} \ \mathcal{K}$ **where**

subgroup \mathcal{H} (homology_group $p \ X$)

subgroup \mathcal{K} (homology_group $p \ X$)

$(\lambda(x, y). \ x \otimes_{\text{homology_group } p \ X} y)$

$\in \text{iso } (\text{DirProd } (\text{subgroup_generated } (\text{homology_group } p \ X) \ \mathcal{H}) \ (\text{subgroup_generated}$
 $(\text{homology_group } p \ X) \ \mathcal{K}))$

$(\text{homology_group } p \ X)$

$(\text{hom_induced } p \ (\text{subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id})$

$\in \text{iso } (\text{homology_group } p \ (\text{subtopology } X \ S)) \ (\text{subgroup_generated } (\text{homology_group}$
 $p \ X) \ \mathcal{H})$

$(\text{hom_induced } p \ X \ \{\} \ X \ S \ \text{id})$

$\in \text{iso } (\text{subgroup_generated } (\text{homology_group } p \ X) \ \mathcal{K}) \ (\text{relative_homology_group}$
 $p \ X \ S)$

using assms

proof

assume $S = \{\}$

show thesis

proof (rule splitting_lemma_left [OF homology_exactness_axiom_3 [of p]])

let $?f = \lambda x. \ \text{one}(\text{homology_group } p \ (\text{subtopology } X \ \{\}))$

show $?f \in \text{hom } (\text{homology_group } p \ X) \ (\text{homology_group } p \ (\text{subtopology } X \ \{\}))$

by (simp add: trivial_hom)

have $\text{tg}: \text{trivial_group } (\text{homology_group } p \ (\text{subtopology } X \ \{\}))$

by (auto simp: topspace_subtopology trivial_homology_group_empty)

then have $[\text{simp}]: \text{carrier } (\text{homology_group } p \ (\text{subtopology } X \ \{\})) = \{\text{one}$
 $(\text{homology_group } p \ (\text{subtopology } X \ \{\}))\}$

by (auto simp: trivial_group_def)

then show $?f \ (\text{hom_induced } p \ (\text{subtopology } X \ \{\}) \ \{\} \ X \ \{\} \ \text{id } x) = x$

if $x \in \text{carrier } (\text{homology_group } p \ (\text{subtopology } X \ \{\}))$ **for** x

using that **by** auto

show $\text{inj_on } (\text{hom_induced } p \ (\text{subtopology } X \ \{\}) \ \{\} \ X \ \{\} \ \text{id})$

$(\text{carrier } (\text{homology_group } p \ (\text{subtopology } X \ \{\})))$

by (meson inj_on_hom_induced_inclusion)

show $\text{hom_induced } p \ X \ \{\} \ X \ \{\} \ \text{id} \text{ ' carrier } (\text{homology_group } p \ X) = \text{carrier}$
 $(\text{homology_group } p \ X)$

by (metis epi_hom_induced_relativization)

```

next
  fix  $\mathcal{H} \ \mathcal{K}$ 
  assume *:  $\mathcal{H} \triangleleft \text{homology\_group } p \ X \ \mathcal{K} \triangleleft \text{homology\_group } p \ X$ 
   $\mathcal{H} \cap \mathcal{K} \subseteq \{1_{\text{homology\_group } p \ X}\}$ 
   $\text{hom\_induced } p \ (\text{subtopology } X \ \{\}) \ \{\} \ X \ \{\} \ \text{id}$ 
   $\in \text{Group.iso } (\text{homology\_group } p \ (\text{subtopology } X \ \{\})) \ (\text{subgroup\_generated}$ 
   $(\text{homology\_group } p \ X) \ \mathcal{H})$ 
   $\text{hom\_induced } p \ X \ \{\} \ X \ \{\} \ \text{id}$ 
   $\in \text{Group.iso } (\text{subgroup\_generated } (\text{homology\_group } p \ X) \ \mathcal{K}) \ (\text{relative\_homology\_group}$ 
   $p \ X \ \{\})$ 
   $\mathcal{H} \langle \# \rangle_{\text{homology\_group } p \ X} \ \mathcal{K} = \text{carrier } (\text{homology\_group } p \ X)$ 
  show thesis
  proof (rule that)
    show  $(\lambda(x, y). \ x \otimes_{\text{homology\_group } p \ X} y)$ 
     $\in \text{iso } (\text{subgroup\_generated } (\text{homology\_group } p \ X) \ \mathcal{H} \times \times \text{subgroup\_generated}$ 
     $(\text{homology\_group } p \ X) \ \mathcal{K})$ 
     $(\text{homology\_group } p \ X)$ 
    using * by (simp add: group_disjoint_sum.iso_group_mul_normal_def
    group_disjoint_sum_def)
    qed (use  $\langle S = \{\} \rangle$  * in  $\langle \text{auto simp: normal\_def} \rangle$ )
  qed
next
  assume  $S \text{ retract\_of\_space } X$ 
  then obtain  $r$  where  $S \subseteq \text{topspace } X$  and  $r$ :  $\text{continuous\_map } X \ (\text{subtopology}$ 
   $X \ S) \ r$ 
    and req:  $\forall x \in S. \ r \ x = x$ 
  by (auto simp: retract_of_space_def)
  show thesis
  proof (rule splitting_lemma_left [OF homology_exactness_axiom_3 [of p]])
    let  $?f = \text{hom\_induced } p \ X \ \{\} \ (\text{subtopology } X \ S) \ \{\} \ r$ 
    show  $?f \in \text{hom } (\text{homology\_group } p \ X) \ (\text{homology\_group } p \ (\text{subtopology } X \ S))$ 
    by (simp add: hom_induced_empty_hom)
    show eqx:  $?f \ (\text{hom\_induced } p \ (\text{subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id } x) = x$ 
    if  $x \in \text{carrier } (\text{homology\_group } p \ (\text{subtopology } X \ S))$  for  $x$ 
    proof -
      have  $\text{hom\_induced } p \ (\text{subtopology } X \ S) \ \{\} \ (\text{subtopology } X \ S) \ \{\} \ r \ x = x$ 
      by (metis  $\langle S \subseteq \text{topspace } X \rangle \text{continuous\_map\_from\_subtopology } \text{hom\_induced\_id\_gen}$ 
       $\text{inf.absorb\_iff2 } r \ \text{req} \ \text{that } \text{topspace\_subtopology}$ )
      then show ?thesis
      by (simp add:  $r \ \text{hom\_induced\_compose}$  [unfolded o_def fun_eq_iff, rule_format,
      symmetric])
    qed
    then show  $\text{inj\_on } (\text{hom\_induced } p \ (\text{subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id})$ 
     $(\text{carrier } (\text{homology\_group } p \ (\text{subtopology } X \ S)))$ 
    unfolding inj_on_def by metis
    show  $\text{hom\_induced } p \ X \ \{\} \ X \ S \ \text{id} \ ' \ \text{carrier } (\text{homology\_group } p \ X) = \text{carrier}$ 
     $(\text{relative\_homology\_group } p \ X \ S)$ 
    by (simp add:  $\langle S \text{ retract\_of\_space } X \rangle \text{epi\_hom\_induced\_relativization}$ )
  next

```



```

fix  $\mathcal{H} \ \mathcal{K}$ 
assume *:  $\mathcal{H} \triangleleft \text{homology\_group } p \ X \ \mathcal{K} \triangleleft \text{homology\_group } p \ X$ 
 $\mathcal{H} \cap \mathcal{K} \subseteq \{1_{\text{homology\_group } p \ X}\}$ 
 $\mathcal{H} <\#> \text{homology\_group } p \ X \ \mathcal{K} = \text{carrier } (\text{homology\_group } p \ X)$ 
 $\text{hom\_induced } p \ (\text{subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id}$ 
 $\in \text{Group.iso } (\text{homology\_group } p \ (\text{subtopology } X \ S)) \ (\text{subgroup\_generated}$ 
 $(\text{homology\_group } p \ X) \ \mathcal{H})$ 
 $\text{hom\_induced } p \ X \ \{\} \ X \ S \ \text{id}$ 
 $\in \text{Group.iso } (\text{subgroup\_generated } (\text{homology\_group } p \ X) \ \mathcal{K}) \ (\text{relative\_homology\_group}$ 
 $p \ X \ S)$ 
show thesis
proof (rule that)
  show  $(\lambda(x, y). x \otimes_{\text{homology\_group } p \ X} y)$ 
 $\in \text{iso } (\text{subgroup\_generated } (\text{homology\_group } p \ X) \ \mathcal{H} \times \times \text{subgroup\_generated}$ 
 $(\text{homology\_group } p \ X) \ \mathcal{K})$ 
 $(\text{homology\_group } p \ X)$ 
  using *
  by (simp add: group_disjoint_sum.iso_group_mul normal_def group_disjoint_sum_def)
qed (use * in <auto simp: normal_def>)
qed
qed

```

```

lemma isomorphic_group_homology_group_prod_retract:
assumes  $S = \{\} \vee S \text{ retract\_of\_space } X$ 
shows  $\text{homology\_group } p \ X \cong \text{homology\_group } p \ (\text{subtopology } X \ S) \times \times \text{rela-}$ 
 $\text{tive\_homology\_group } p \ X \ S$ 
 $(\text{is } ?lhs \cong ?rhs)$ 
proof –
  obtain  $\mathcal{H} \ \mathcal{K}$  where
     $\text{subgroup } \mathcal{H} \ (\text{homology\_group } p \ X)$ 
     $\text{subgroup } \mathcal{K} \ (\text{homology\_group } p \ X)$ 
  and  $1: (\lambda(x, y). x \otimes_{\text{homology\_group } p \ X} y)$ 
 $\in \text{iso } (\text{DirProd } (\text{subgroup\_generated } (\text{homology\_group } p \ X) \ \mathcal{H}) \ (\text{subgroup\_generated}$ 
 $(\text{homology\_group } p \ X) \ \mathcal{K}))$ 
 $(\text{homology\_group } p \ X)$ 
 $(\text{hom\_induced } p \ (\text{subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id})$ 
 $\in \text{iso } (\text{homology\_group } p \ (\text{subtopology } X \ S)) \ (\text{subgroup\_generated } (\text{homology\_group}$ 
 $p \ X) \ \mathcal{H})$ 
 $(\text{hom\_induced } p \ X \ \{\} \ X \ S \ \text{id})$ 
 $\in \text{iso } (\text{subgroup\_generated } (\text{homology\_group } p \ X) \ \mathcal{K}) \ (\text{relative\_homology\_group}$ 
 $p \ X \ S)$ 
  using group_isomorphisms_homology_group_prod_retract [OF assms] by blast
  have  $?lhs \cong \text{subgroup\_generated } (\text{homology\_group } p \ X) \ \mathcal{H} \times \times \text{subgroup\_generated}$ 
 $(\text{homology\_group } p \ X) \ \mathcal{K}$ 
  by (meson DirProd_group 1 abelian_homology_group_comm_group_def group.abelian_subgroup_generated
 $\text{group.iso\_sym is\_isoI}$ )
  also have  $\dots \cong ?rhs$ 

```

by (meson 1(2) 1(3) abelian_homology_group comm_group_def group.DirProd_iso_trans
 group.abelian_subgroup_generated group.iso_sym is_isoI)
 finally show ?thesis .
 qed

lemma homology_additivity_explicit:

assumes openin X S openin X T disjnt S T **and** SUT: $S \cup T = \text{topspace } X$

shows $(\lambda(a,b).(\text{hom_induced } p (\text{subtopology } X S) \{\} X \{\} \text{id } a)$

$\otimes \text{homology_group } p X$

$(\text{hom_induced } p (\text{subtopology } X T) \{\} X \{\} \text{id } b))$

$\in \text{iso } (\text{DirProd } (\text{homology_group } p (\text{subtopology } X S)) (\text{homology_group } p$
 $(\text{subtopology } X T)))$
 $(\text{homology_group } p X)$

proof –

have closedin X S closedin X T

using assms Un_commute disjnt_sym

by (metis Diff_cancel Diff_triv Un_Diff disjnt_def openin_closedin_eq sup_bot.right_neutral)+

with $\langle \text{openin } X S \rangle \langle \text{openin } X T \rangle$ **have** SS: $X \text{ closure_of } S \subseteq X \text{ interior_of } S$

and TT: $X \text{ closure_of } T \subseteq X \text{ interior_of } T$

by (simp_all add: closure_of_closedin interior_of_openin)

have [simp]: $S \cup T - T = S \ S \cup T - S = T$

using $\langle \text{disjnt } S T \rangle$

by (auto simp: Diff_triv Un_Diff disjnt_def)

let ?f = $\text{hom_induced } p X \{\} X T \text{id}$

let ?g = $\text{hom_induced } p X \{\} X S \text{id}$

let ?h = $\text{hom_induced } p (\text{subtopology } X S) \{\} X T \text{id}$

let ?i = $\text{hom_induced } p (\text{subtopology } X S) \{\} X \{\} \text{id}$

let ?j = $\text{hom_induced } p (\text{subtopology } X T) \{\} X \{\} \text{id}$

let ?k = $\text{hom_induced } p (\text{subtopology } X T) \{\} X S \text{id}$

let ?A = $\text{homology_group } p (\text{subtopology } X S)$

let ?B = $\text{homology_group } p (\text{subtopology } X T)$

let ?C = $\text{relative_homology_group } p X T$

let ?D = $\text{relative_homology_group } p X S$

let ?G = $\text{homology_group } p X$

have h: $?h \in \text{iso } ?A ?C$ **and** k: $?k \in \text{iso } ?B ?D$

using homology_excision_axiom [OF TT, of $S \cup T$ p]

using homology_excision_axiom [OF SS, of $S \cup T$ p]

by auto (simp_all add: SUT)

have 1: $\bigwedge x. (\text{hom_induced } p X \{\} X T \text{id} \circ \text{hom_induced } p (\text{subtopology } X S)$
 $\{\} X \{\} \text{id}) x$

$= \text{hom_induced } p (\text{subtopology } X S) \{\} X T \text{id } x$

by (simp flip: hom_induced_compose)

have 2: $\bigwedge x. (\text{hom_induced } p X \{\} X S \text{id} \circ \text{hom_induced } p (\text{subtopology } X T)$
 $\{\} X \{\} \text{id}) x$

$= \text{hom_induced } p (\text{subtopology } X T) \{\} X S \text{id } x$

by (simp flip: hom_induced_compose)

show ?thesis

using exact_sequence_sum_lemma

```

      [OF abelian_homology_group h k homology_exactness_axiom_3 homol-
ogy_exactness_axiom_3] 1 2
    by auto
  qed

```

0.2.6 Generalize exact homology sequence to triples

definition *hom_relboundary* :: $[int, 'a \text{ topology}, 'a \text{ set}, 'a \text{ set}, 'a \text{ chain set}] \Rightarrow 'a \text{ chain set}$

where

```

hom_relboundary p X S T =
  hom_induced (p-1) (subtopology X S) {} (subtopology X S) T id ∘
  hom_boundary p X S

```

lemma *group_homomorphism_hom_relboundary*:

```

  hom_relboundary p X S T
  ∈ hom (relative_homology_group p X S) (relative_homology_group (p-1) (subtopology
X S) T)
  unfolding hom_relboundary_def
  proof (rule hom_compose)
    show hom_boundary p X S ∈ hom (relative_homology_group p X S) (homology_group(p-1)
(subtopology X S))
    by (simp add: hom_boundary_hom)
    show hom_induced (p-1) (subtopology X S) {} (subtopology X S) T id
      ∈ hom (homology_group(p-1) (subtopology X S)) (relative_homology_group
(p-1) (subtopology X S) T)
    by (simp add: hom_induced_hom)
  qed

```

lemma *hom_relboundary*:

```

  hom_relboundary p X S T c ∈ carrier (relative_homology_group (p-1) (subtopology
X S) T)
  by (simp add: hom_relboundary_def hom_induced_carrier)

```

lemma *hom_relboundary_empty*: $\text{hom_relboundary } p \text{ } X \text{ } S \text{ } \{\} = \text{hom_boundary } p \text{ } X \text{ } S$

```

  by (simp add: ext hom_boundary_carrier hom_induced_id hom_relboundary_def)

```

lemma *naturality_hom_induced_relboundary*:

```

  assumes continuous_map X Y f f ' S ⊆ U f ' T ⊆ V
  shows hom_relboundary p Y U V ∘
    hom_induced p X S Y (U) f =
    hom_induced (p-1) (subtopology X S) T (subtopology Y U) V f ∘
    hom_relboundary p X S T
  proof -
    have [simp]: continuous_map (subtopology X S) (subtopology Y U) f
    using assms continuous_map_from_subtopology continuous_map_in_subtopology
topspace_subtopology by fastforce
    have hom_induced (p-1) (subtopology Y U) {} (subtopology Y U) V id ∘

```

```

      hom_induced (p-1) (subtopology X S) {} (subtopology Y U) {} f
    = hom_induced (p-1) (subtopology X S) T (subtopology Y U) V f ∘
      hom_induced (p-1) (subtopology X S) {} (subtopology X S) T id
    using assms by (simp flip: hom_induced_compose)
  with assms show ?thesis
  by (metis (no_types, lifting) fun.map_comp hom_relboundary_def naturality_hom_induced)
qed

```

proposition *homology_exactness_triple_1*:

```

  assumes  $T \subseteq S$ 
  shows exact_seq ([relative_homology_group (p-1) (subtopology X S) T,
    relative_homology_group p X S,
    relative_homology_group p X T],
    [hom_relboundary p X S T, hom_induced p X T X S id])
  (is exact_seq ([?G1, ?G2, ?G3], [?h1, ?h2]))
proof -
  have iTS:  $id \restriction T \subseteq S$  and [simp]:  $S \cap T = T$ 
  using assms by auto
  have ?h2  $B \in \text{kernel } ?G2 \text{ } ?G1 \text{ } ?h1$  for B
  proof -
    have hom_boundary p X T  $B \in \text{carrier } (\text{relative\_homology\_group } (p-1) (\text{subtopology } X \text{ } T) \text{ } \{\})$ 
    by (metis (no_types) hom_boundary)
    then have *: hom_induced (p-1) (subtopology X S) {} (subtopology X S) T id
    id
      (hom_induced (p-1) (subtopology X T) {} (subtopology X S) {} id
      (hom_boundary p X T B))
    = 1 ?G1
    using homology_exactness_axiom_3 [of p-1 subtopology X S T]
    by (auto simp: subtopology_subtopology kernel_def)
    show ?thesis
    using naturality_hom_induced [OF continuous_map_id iTS]
    by (smt (verit, best) * comp_apply hom_induced_carrier hom_relboundary_def kernel_def mem_Collect_eq)
  qed
  moreover have  $B \in ?h2 \restriction \text{carrier } ?G3$  if  $B \in \text{kernel } ?G2 \text{ } ?G1 \text{ } ?h1$  for B
  proof -
    have Bcarr:  $B \in \text{carrier } ?G2$ 
    and Beq:  $?h1 \restriction B = 1 ?G1$ 
    using that by (auto simp: kernel_def)
    have  $\exists A' \in \text{carrier } (\text{homology\_group } (p-1) (\text{subtopology } X \text{ } T)). \text{hom\_induced } (p-1) (\text{subtopology } X \text{ } T) \text{ } \{\} (\text{subtopology } X \text{ } S) \text{ } \{\} id \text{ } A' = A$ 
    if  $A \in \text{carrier } (\text{homology\_group } (p-1) (\text{subtopology } X \text{ } S))$ 
    hom_induced (p-1) (subtopology X S) {} (subtopology X S) T id A =
    1 ?G1 for A
    using homology_exactness_axiom_3 [of p-1 subtopology X S T] that
    by (simp add: kernel_def subtopology_subtopology image_iff set_eq_iff)
  meson

```

```

then obtain C where Ccarr:  $C \in \text{carrier } (\text{homology\_group } (p-1) (\text{subtopology } X T))$ 
and Ceq:  $\text{hom\_induced } (p-1) (\text{subtopology } X T) \{ \} (\text{subtopology } X S) \{ \} \text{ id}$ 
 $C = \text{hom\_boundary } p X S B$ 
using Beq by (simp add: hom_relboundary_def) (metis hom_boundary_carrier)
let ?hi_XT =  $\text{hom\_induced } (p-1) (\text{subtopology } X T) \{ \} X \{ \} \text{ id}$ 
have ?hi_XT
  =  $\text{hom\_induced } (p-1) (\text{subtopology } X S) \{ \} X \{ \} \text{ id}$ 
     $\circ (\text{hom\_induced } (p-1) (\text{subtopology } X T) \{ \} (\text{subtopology } X S) \{ \} \text{ id})$ 
by (metis assms comp_id continuous_map_id_subt hom_induced_compose_empty
inf.absorb_iff2 subtopology_subtopology)
then have ?hi_XT C
  =  $\text{hom\_induced } (p-1) (\text{subtopology } X S) \{ \} X \{ \} \text{ id } (\text{hom\_boundary } p X S B)$ 
by (simp add: Ceq)
also have eq:  $\dots = \mathbf{1}_{\text{homology\_group } (p-1) X}$ 
using homology_exactness_axiom_2 [of p X S] Bcarr by (auto simp: kernel_def)
finally have ?hi_XT C =  $\mathbf{1}_{\text{homology\_group } (p-1) X} \cdot$ 
then obtain D where Dcarr:  $D \in \text{carrier } ?G3$  and Deq:  $\text{hom\_boundary } p X T D = C$ 
using homology_exactness_axiom_2 [of p X T] Ccarr
by (auto simp: kernel_def image_iff set_eq_iff) meson
interpret hb:  $\text{group\_hom } ?G2 \text{ homology\_group } (p-1) (\text{subtopology } X S)$ 
 $\text{hom\_boundary } p X S$ 
using hom_boundary_hom_group_hom_axioms_def group_hom_def by fast-force
let ?A =  $B \otimes_{?G2} \text{inv}_{?G2} ?h2 D$ 
have  $\exists A' \in \text{carrier } (\text{homology\_group } p X). \text{hom\_induced } p X \{ \} X S \text{ id } A' = A$ 
if  $A \in \text{carrier } ?G2$ 
 $\text{hom\_boundary } p X S A = \text{one } (\text{homology\_group } (p-1) (\text{subtopology } X S))$ 
for A
using that homology_exactness_axiom_1 [of p X S]
by (simp add: kernel_def subtopology_subtopology image_iff set_eq_iff) meson
moreover
have  $?A \in \text{carrier } ?G2$ 
by (simp add: Bcarr abelian_relative_homology_group comm_groupE(1) hom_induced_carrier)
moreover have  $\text{hom\_boundary } p X S (?h2 D) = \text{hom\_boundary } p X S B$ 
by (metis (mono_tags, lifting) Ceq Deq comp_eq_dest continuous_map_id
iTS naturality_hom_induced)
then have  $\text{hom\_boundary } p X S ?A = \text{one } (\text{homology\_group } (p-1) (\text{subtopology } X S))$ 
by (simp add: hom_induced_carrier Bcarr)
ultimately obtain W where Wcarr:  $W \in \text{carrier } (\text{homology\_group } p X)$ 
and Weg:  $\text{hom\_induced } p X \{ \} X S \text{ id } W = ?A$ 
by blast

```

```

let ?W = D ⊗?G3 hom_induced p X {} X T id W
show ?thesis
proof
  interpret comm_group ?G2
  by (rule abelian_relative_homology_group)
  have hom_induced p X T X S id (hom_induced p X {} X T id W) =
hom_induced p X {} X S id W
  by (simp add: assms hom_induced_compose')
  then have B = (?h2 ∘ hom_induced p X {} X T id) W ⊗?G2 ?h2 D
  by (simp add: Bcarr Weq hb.G.m_assoc hom_induced_carrier)
  then show B = ?h2 ?W
  by (metis hom_mult [OF hom_induced_hom] Dcarr comp_apply hom_induced_carrier
m_comm)
  show ?W ∈ carrier ?G3
  by (simp add: Dcarr comm_groupE(1) hom_induced_carrier)
qed
qed
ultimately show ?thesis
  by (auto simp: group_hom_def group_hom_axioms_def hom_induced_hom
group_homomorphism_hom_relboundary)
qed

```

proposition *homology_exactness_triple_2:*

```

assumes T ⊆ S
shows exact_seq ([relative_homology_group(p-1) X T,
                  relative_homology_group(p-1) (subtopology X S) T,
                  relative_homology_group p X S],
[hom_induced (p-1) (subtopology X S) T X T id, hom_relboundary
p X S T])
(is exact_seq ([?G1, ?G2, ?G3], [?h1, ?h2]))
proof -
  let ?H2 = homology_group (p-1) (subtopology X S)
  have iTS: id ' T ⊆ S and [simp]: S ∩ T = T
  using assms by auto
  have ?h2 C ∈ kernel ?G2 ?G1 ?h1 for C
  proof -
    have ?h1 (?h2 C)
    = (hom_induced (p-1) X {} X T id ∘ hom_induced (p-1) (subtopology X
S) {} X {} id ∘ hom_boundary p X S) C
    unfolding hom_relboundary_def
    by (metis (no_types, lifting) comp_apply continuous_map_id continuous_map_id_subt
empty_subsetI hom_induced_compose id_apply image_empty image_id order_refl)
    also have ... = 1 ?G1
  proof -
    have *: hom_boundary p X S C ∈ carrier ?H2
    by (simp add: hom_boundary_carrier)
    moreover have hom_boundary p X S C ∈ hom_boundary p X S ' carrier
?G3

```

```

    using homology_exactness_axiom_2 [of p X S] *
    apply (simp add: kernel_def set_eq_iff)
    by (metis group_relative_homology_group hom_boundary_default hom_one
image_eqI)
    ultimately
    have 1: hom_induced (p-1) (subtopology X S) {} X {} id (hom_boundary
p X S C)
        = 1_homology_group (p-1) X
    using homology_exactness_axiom_2 [of p X S] by (simp add: kernel_def)
blast
    show ?thesis
    by (simp add: 1 hom_one [OF hom_induced_hom])
qed
finally have ?h1 (?h2 C) = 1 ?G1 .
then show ?thesis
    by (simp add: kernel_def hom_relboundary_def hom_induced_carrier)
qed
moreover have x ∈ ?h2 ‘ carrier ?G3 if x ∈ kernel ?G2 ?G1 ?h1 for x
proof -
    let ?homX = hom_induced (p-1) (subtopology X S) {} X {} id
    let ?homXS = hom_induced (p-1) (subtopology X S) {} (subtopology X S) T
id
    have x ∈ carrier (relative_homology_group (p-1) (subtopology X S) T)
    using that by (simp add: kernel_def)
    moreover
    have hom_boundary (p-1) X T ∘ hom_induced (p-1) (subtopology X S) T
X T id = hom_boundary (p-1) (subtopology X S) T
    by (metis Int_lower2 ⟨S ∩ T = T⟩ continuous_map_id_subt hom_relboundary_def
hom_relboundary_empty id_apply image_id naturality_hom_induced subtopology_subtopology)
    then have hom_boundary (p-1) (subtopology X S) T x = 1_homology_group (p - 2) (subtopology (subtopology X S)
    using naturality_hom_induced [of subtopology X S X id T T p-1] that
    hom_one [OF hom_boundary_hom group_relative_homology_group group_relative_homology_group,
of p-1 X T]
    by (smt (verit) assms comp_apply inf.absorb_iff2 kernel_def mem_Collect_eq
subtopology_subtopology)
    ultimately
    obtain y where ycarr: y ∈ carrier ?H2
    and yeq: ?homXS y = x
    using homology_exactness_axiom_1 [of p-1 subtopology X S T]
    by (simp add: kernel_def image_def set_eq_iff) meson
    have ?homX y ∈ carrier (homology_group (p-1) X)
    by (simp add: hom_induced_carrier)
    moreover
    have (hom_induced (p-1) X {} X T id ∘ ?homX) y = 1_relative_homology_group (p-1) X T
    using that
    apply (simp add: kernel_def flip: hom_induced_compose)
    using hom_induced_compose [of subtopology X S subtopology X S id {} T X
id T p-1] yeq
    by auto

```

```

then have hom_induced (p-1) X {} X T id (?homX y) = 1relative_homology_group (p-1) X T
  by simp
ultimately obtain z where zcarr: z ∈ carrier (homology_group (p-1)
(subtopology X T))
  and zeq: hom_induced (p-1) (subtopology X T) {} X {} id z = ?homX
y
  using homology_exactness_axiom_3 [of p-1 X T]
  by (auto simp: kernel_def dest!: equalityD1 [of Collect _])
have *:  $\bigwedge t. \llbracket t \in \text{carrier } ?H2; \text{hom\_induced (p-1) (subtopology X S) } \{\} X \{\} \text{ id } t = 1_{\text{homology\_group (p-1) X}} \rrbracket$ 
 $\implies t \in \text{hom\_boundary } p X S \text{ 'carrier' } ?G3$ 
  using homology_exactness_axiom_2 [of p X S]
  by (auto simp: kernel_def dest!: equalityD1 [of Collect _])
interpret comm_group ?H2
  by (rule abelian_relative_homology_group)
interpret gh: group_hom ?H2 homology_group (p-1) X hom_induced (p-1)
(subtopology X S) {} X {} id
  by (meson group_hom_axioms_def group_hom_def group_relative_homology_group
hom_induced)
let ?yz = y  $\otimes_{?H2}$  inv ?H2 hom_induced (p-1) (subtopology X T) {} (subtopology
X S) {} id z
  have yzcarr: ?yz ∈ carrier ?H2
  by (simp add: hom_induced_carrier ycarr)
  have hom_induced (p-1) (subtopology X S) {} X {} id y =
    hom_induced (p-1) (subtopology X S) {} X {} id
    (hom_induced (p-1) (subtopology X T) {} (subtopology X S) {} id z)
  by (metis assms continuous_map_id_subt hom_induced_compose_empty
inf.absorb_iff2 o_apply o_id subtopyology_subtopyology zeq)
  then have yzeq: hom_induced (p-1) (subtopology X S) {} X {} id ?yz =
1homology_group (p-1) X
  by (simp add: hom_induced_carrier ycarr gh.inv_solve_right')
  obtain w where wcarr: w ∈ carrier ?G3 and weq: hom_boundary p X S w =
?yz
  using * [OF yzcarr yzeq] by blast
interpret gh2: group_hom ?H2 ?G2 ?homXS
  by (simp add: group_hom_axioms_def group_hom_def hom_induced_hom)
  have ?homXS (hom_induced (p-1) (subtopology X T) {} (subtopology X S)
{} id z)
    = 1relative_homology_group (p-1) (subtopology X S) T
  using homology_exactness_axiom_3 [of p-1 subtopyology X S T] zcarr
  by (auto simp: kernel_def subtopyology_subtopyology)
then show ?thesis
  apply (rule_tac x=w in image_eqI)
  apply (simp_all add: hom_relboundary_def weq wcarr)
  by (metis gh2.hom_inv gh2.hom_mult gh2.inv_one gh2.r_one group.inv_closed
group_l_invI hom_induced_carrier l_inv_ex ycarr yeq)
qed
ultimately show ?thesis
  by (auto simp: group_hom_axioms_def group_hom_def group_homomorphism_hom_relboundary

```


hom_induced_hom)
qed

proposition homology_exactness_triple_3:

assumes $T \subseteq S$

shows exact_seq ([relative_homology_group p X S,
relative_homology_group p X T,
relative_homology_group p (subtopology X S) T],
[hom_induced p X T X S id, hom_induced p (subtopology X S) T
X T id])

(is exact_seq ([?G1, ?G2, ?G3], [?h1, ?h2]))

proof –

have iTS: id ‘ $T \subseteq S$ and [simp]: $S \cap T = T$

using assms by auto

have 1: ?h2 x ∈ kernel ?G2 ?G1 ?h1 for x

proof –

have ?h1 (?h2 x)

= (hom_induced p (subtopology X S) S X S id ◦

hom_induced p (subtopology X S) T (subtopology X S) S id) x

by (metis comp_eq_dest_lhs continuous_map_id continuous_map_id_subt
hom_induced_compose iTS id_apply image_subsetI)

also have ... = 1_{relative_homology_group p X S}

proof –

have trivial_group (relative_homology_group p (subtopology X S) S)

using trivial_relative_homology_group_topspace [of p subtopology X S]

by (metis inf_right_idem relative_homology_group_restrict_topspace_subtopology)

then have 1: hom_induced p (subtopology X S) T (subtopology X S) S id x

= 1_{relative_homology_group p (subtopology X S) S}

using hom_induced_carrier by (fastforce simp add: trivial_group_def)

show ?thesis

by (simp add: 1 hom_one [OF hom_induced_hom])

qed

finally have ?h1 (?h2 x) = 1_{relative_homology_group p X S} ·

then show ?thesis

by (simp add: hom_induced_carrier kernel_def)

qed

moreover have x ∈ ?h2 ‘ carrier ?G3 if x: x ∈ kernel ?G2 ?G1 ?h1 for x

proof –

have xcarr: x ∈ carrier ?G2

using that by (auto simp: kernel_def)

interpret G2: comm_group ?G2

by (rule abelian_relative_homology_group)

let ?b = hom_boundary p X T x

have bcarr: ?b ∈ carrier(homology_group(p-1) (subtopology X T))

by (simp add: hom_boundary_carrier)

have hom_boundary p X S (hom_induced p X T X S id x)

= hom_induced (p-1) (subtopology X T) {} (subtopology X S) {} id

(hom_boundary p X T x)

using naturality_hom_induced [of X X id T S p] by (simp add: assms o_def)

```

meson
  with bcarr have hom_boundary p X T x ∈ hom_boundary p (subtopology X
S) T ‘ carrier ?G3
    using homology_exactness_axiom_2 [of p subtopology X S T] x
    apply (simp add: kernel_def set_eq_iff subtopology_subtopology)
    by (metis group_relative_homology_group hom_boundary_hom hom_one
set_eq_iff)
  then obtain u where ucarr: u ∈ carrier ?G3
    and ueq: hom_boundary p X T x = hom_boundary p (subtopology X S)
T u
    by (auto simp: kernel_def set_eq_iff subtopology_subtopology hom_boundary_carrier)
  define y where y = x ⊗?G2 inv?G2 ?h2 u
  have ycarr: y ∈ carrier ?G2
    using x by (simp add: y_def kernel_def hom_induced_carrier)
  interpret hb: group_hom ?G2 homology_group (p-1) (subtopology X T)
hom_boundary p X T
    by (simp add: group_hom_axioms_def group_hom_def hom_boundary_hom)
  have yyy: hom_boundary p X T y = 1homology_group (p-1) (subtopology X T)
  apply (simp add: y_def bcarr xcarr hom_induced_carrier hom_boundary_carrier
hb.inv_solve_right')
    using naturality_hom_induced [of concl: p X T subtopology X S T id]
  by (smt (verit, best) ⟨S ∩ T = T⟩ bcarr comp_eq_dest continuous_map_id_subt
hom_induced_id_id_apply
    image_subset_iff subtopology_subtopology ueq)
  then have y ∈ hom_induced p X {} X T id ‘ carrier (homology_group p X)
    using homology_exactness_axiom_1 [of p X T] x ycarr by (auto simp:
kernel_def)
  then obtain z where zcarr: z ∈ carrier (homology_group p X)
    and zeq: hom_induced p X {} X T id z = y
    by auto
  interpret gh1: group_hom ?G2 ?G1 ?h1
  by (meson group_hom_axioms_def group_hom_def group_relative_homology_group
hom_induced)

  have hom_induced p X {} X S id z = (hom_induced p X T X S id ∘
hom_induced p X {} X T id) z
    by (simp add: assms flip: hom_induced_compose)
  also have ... = 1relative_homology_group p X S
    using x 1 by (simp add: kernel_def zeq y_def)
  finally have hom_induced p X {} X S id z = 1relative_homology_group p X S ·
  then have z ∈ hom_induced p (subtopology X S) {} X {} id ‘
    carrier (homology_group p (subtopology X S))
    using homology_exactness_axiom_3 [of p X S] zcarr by (auto simp: ker-
nel_def)
  then obtain w where wcarr: w ∈ carrier (homology_group p (subtopology X
S))
    and weq: hom_induced p (subtopology X S) {} X {} id w = z
    by blast
  let ?u = hom_induced p (subtopology X S) {} (subtopology X S) T id w ⊗?G3

```

0.3 Homology, III: Brouwer Degree

begin

$$\begin{aligned} \text{definition } \text{reduced_homology_group} &:: \text{int} \Rightarrow 'a \text{ topology} \Rightarrow 'a \text{ chain set monoid} \\ \text{where } \text{reduced_homology_group } p \ X &\equiv \\ &\text{subgroup_generated } (\text{homology_group } p \ X) \\ &\quad (\text{kernel } (\text{homology_group } p \ X) \ (\text{homology_group } p \ (\text{discrete_topology} \\ &\quad \{()\})) \\ &\quad (\text{hom_induced } p \ X \ \{\} \ (\text{discrete_topology } \{()\}) \ \{\} \ (\lambda x. ())) \end{aligned}$$

lemma *group_reduced_homology_group* [*simp*]: *group* (*reduced_homology_group* p X)
by (*simp add: reduced_homology_group_def group.group subgroup generated*)

```

lemma carrier_reduced_homology_group:
  carrier (reduced_homology_group p X) =

```

```

    kernel (homology_group p X) (homology_group p (discrete_topology {}))
      (hom_induced p X {} (discrete_topology {}) {} (λx. ()))
    (is _ = kernel ?G ?H ?h)
  proof -
    interpret subgroup kernel ?G ?H ?h ?G
    by (simp add: hom_induced_empty_hom group_hom_axioms_def group_hom_def
      group_hom.subgroup_kernel)
    show ?thesis
      unfolding reduced_homology_group_def
      using carrier_subgroup_generated_subgroup by blast
  qed

```

```

lemma carrier_reduced_homology_group_subset:
  carrier (reduced_homology_group p X) ⊆ carrier (homology_group p X)
  by (simp add: group.carrier_subgroup_generated_subset reduced_homology_group_def)

```

```

lemma un_reduced_homology_group:
  assumes p ≠ 0
  shows reduced_homology_group p X = homology_group p X
  proof -
    have (kernel (homology_group p X) (homology_group p (discrete_topology {})))
      (hom_induced p X {} (discrete_topology {}) {} (λx. ()))
      = carrier (homology_group p X)
    proof (rule group_hom.kernel_to_trivial_group)
      show group_hom (homology_group p X) (homology_group p (discrete_topology
        {}))
        (hom_induced p X {} (discrete_topology {}) {} (λx. ()))
      by (auto simp: hom_induced_empty_hom group_hom_def group_hom_axioms_def)
      show trivial_group (homology_group p (discrete_topology {}))
      by (simp add: homology_dimension_axiom [OF _ assms])
    qed
    then show ?thesis
      by (simp add: reduced_homology_group_def group.subgroup_generated_group_carrier)
  qed

```

```

lemma trivial_reduced_homology_group:
  p < 0 ⟹ trivial_group (reduced_homology_group p X)
  by (simp add: trivial_homology_group un_reduced_homology_group)

```

```

lemma hom_induced_reduced_hom:
  (hom_induced p X {} Y {} f) ∈ hom (reduced_homology_group p X) (reduced_homology_group
    p Y)
  proof (cases continuous_map X Y f)
    case True
      have eq: continuous_map X Y f
        ⟹ hom_induced p X {} (discrete_topology {}) {} (λx. ())
        = (hom_induced p Y {} (discrete_topology {}) {} (λx. ())) ∘ hom_induced
          p X {} Y {} f
      by (simp flip: hom_induced_compose_empty)

```

```

interpret subgroup_kernel (homology_group p X)
  (homology_group p (discrete_topology {}))
  (hom_induced p X {} (discrete_topology {}) {} (λx. ()))
  homology_group p X
by (meson group_hom.subgroup_kernel group_hom_axioms_def group_hom_def
group_relative_homology_group hom_induced)
have sb: hom_induced p X {} Y {} f ' carrier (homology_group p X) ⊆ carrier
(homology_group p Y)
using hom_induced_carrier by blast
show ?thesis
using True
unfolding reduced_homology_group_def
apply (simp add: hom_into_subgroup_eq group_hom.subgroup_kernel hom_induced_empty_hom
group_hom_from_subgroup_generated group_hom_def group_hom_axioms_def)
unfolding kernel_def using eq sb by auto
next
case False
then have hom_induced p X {} Y {} f = (λc. one(reduced_homology_group p
Y))
by (force simp: hom_induced_default reduced_homology_group_def)
then show ?thesis
by (simp add: trivial_hom)
qed

```

lemma hom_induced_reduced:

```

  c ∈ carrier(reduced_homology_group p X)
    ⇒ hom_induced p X {} Y {} f c ∈ carrier(reduced_homology_group p Y)
by (meson hom_in_carrier hom_induced_reduced_hom)

```

lemma hom_boundary_reduced_hom:

```

  hom_boundary p X S
  ∈ hom (relative_homology_group p X S) (reduced_homology_group (p-1) (subtopology
X S))

```

proof –

```

have *: continuous_map X (discrete_topology {}) (λx. ()) (λx. ()) ' S ⊆ {}
by auto
interpret group_hom relative_homology_group p (discrete_topology {}) {}
  homology_group (p-1) (discrete_topology {})
  hom_boundary p (discrete_topology {}) {}
apply (clarsimp simp: group_hom_def group_hom_axioms_def)
by (metis UNIV_unit hom_boundary_hom subtopology_UNIV)
have hom_boundary p X S '
  carrier (relative_homology_group p X S)
  ⊆ kernel (homology_group (p-1) (subtopology X S))
  (homology_group (p-1) (discrete_topology {}))
  (hom_induced (p-1) (subtopology X S) {})
  (discrete_topology {}) {} (λx. ())
proof (clarsimp simp add: kernel_def hom_boundary_carrier)

```

```

fix c
assume c:  $c \in \text{carrier } (\text{relative\_homology\_group } p \ X \ S)$ 
have triv:  $\text{trivial\_group } (\text{relative\_homology\_group } p \ (\text{discrete\_topology } \{\}\))$ 
 $\{\}\}$ 
by (metis topspace_discrete_topology trivial_relative_homology_group_topspace)
have hom_boundary p ( $\text{discrete\_topology } \{\}\}$   $\{\}$ 
  ( $\text{hom\_induced } p \ X \ S \ (\text{discrete\_topology } \{\}\}) \ \{\}\} \ (\lambda x. \ ()) \ c$ 
  =  $1_{\text{homology\_group } (p - 1) \ (\text{discrete\_topology } \{\}\)}$ 
by (metis hom_induced_carrier local.hom_one singletonD triv_trivial_group_def)
then show  $\text{hom\_induced } (p - 1) \ (\text{subtopology } X \ S) \ \{\} \ (\text{discrete\_topology } \{\}\})$ 
 $\{\} \ (\lambda x. \ ()) \ (\text{hom\_boundary } p \ X \ S \ c) =$ 
 $1_{\text{homology\_group } (p - 1) \ (\text{discrete\_topology } \{\}\)}$ 
using naturality_hom_induced [OF *, of p, symmetric] by (simp add: o_def
fun_eq_iff)
qed
then show ?thesis
by (simp add: reduced_homology_group_def hom_boundary_hom hom_into_subgroup)
qed

```

```

lemma homotopy_equivalence_reduced_homology_group_isomorphisms:
assumes contf:  $\text{continuous\_map } X \ Y \ f$  and contg:  $\text{continuous\_map } Y \ X \ g$ 
and gf:  $\text{homotopic\_with } (\lambda h. \ \text{True}) \ X \ X \ (g \circ f) \ \text{id}$ 
and fg:  $\text{homotopic\_with } (\lambda k. \ \text{True}) \ Y \ Y \ (f \circ g) \ \text{id}$ 
shows  $\text{group\_isomorphisms } (\text{reduced\_homology\_group } p \ X) \ (\text{reduced\_homology\_group } p \ Y)$ 
 $(\text{hom\_induced } p \ X \ \{\} \ Y \ \{\} \ f) \ (\text{hom\_induced } p \ Y \ \{\} \ X$ 
 $\{\} \ g)$ 
proof (simp add: hom_induced_reduced_hom_group_isomorphisms_def, intro conjI
ballI)
fix a
assume  $a \in \text{carrier } (\text{reduced\_homology\_group } p \ X)$ 
then have  $(\text{hom\_induced } p \ Y \ \{\} \ X \ \{\} \ g \circ \text{hom\_induced } p \ X \ \{\} \ Y \ \{\} \ f) \ a = a$ 
apply (simp add: contf contg flip: hom_induced_compose)
using carrier_reduced_homology_group_subset gf hom_induced_id homology_homotopy_empty by fastforce
then show  $\text{hom\_induced } p \ Y \ \{\} \ X \ \{\} \ g \ (\text{hom\_induced } p \ X \ \{\} \ Y \ \{\} \ f \ a) = a$ 
by simp
next
fix b
assume  $b \in \text{carrier } (\text{reduced\_homology\_group } p \ Y)$ 
then have  $(\text{hom\_induced } p \ X \ \{\} \ Y \ \{\} \ f \circ \text{hom\_induced } p \ Y \ \{\} \ X \ \{\} \ g) \ b = b$ 
apply (simp add: contf contg flip: hom_induced_compose)
using carrier_reduced_homology_group_subset fg hom_induced_id homology_homotopy_empty by fastforce
then show  $\text{hom\_induced } p \ X \ \{\} \ Y \ \{\} \ f \ (\text{hom\_induced } p \ Y \ \{\} \ X \ \{\} \ g \ b) = b$ 
by (simp add: carrier_reduced_homology_group)
qed

```

lemma *homotopy_equivalence_reduced_homology_group_isomorphism*:
assumes *continuous_map* $X\ Y\ f$ *continuous_map* $Y\ X\ g$
and *homotopic_with* $(\lambda h. \text{True})\ X\ X\ (g \circ f)$ *id* *homotopic_with* $(\lambda k. \text{True})\ Y\ Y\ (f \circ g)$ *id*
shows $(\text{hom_induced } p\ X\ \{\}\ Y\ \{\}\ f)$
 $\in \text{iso } (\text{reduced_homology_group } p\ X)\ (\text{reduced_homology_group } p\ Y)$
proof (*rule group_isomorphisms_imp_iso*)
show *group_isomorphisms* $(\text{reduced_homology_group } p\ X)\ (\text{reduced_homology_group } p\ Y)$
 $(\text{hom_induced } p\ X\ \{\}\ Y\ \{\}\ f)\ (\text{hom_induced } p\ Y\ \{\}\ X\ \{\}\ g)$
by (*simp add: assms homotopy_equivalence_reduced_homology_group_isomorphisms*)
qed

lemma *homotopy_equivalent_space_imp_isomorphic_reduced_homology_groups*:
 $X\ \text{homotopy_equivalent_space } Y$
 $\implies \text{reduced_homology_group } p\ X \cong \text{reduced_homology_group } p\ Y$
unfolding *homotopy_equivalent_space_def*
using *homotopy_equivalence_reduced_homology_group_isomorphism is_isoI* **by**
blast

lemma *homeomorphic_space_imp_isomorphic_reduced_homology_groups*:
 $X\ \text{homeomorphic_space } Y \implies \text{reduced_homology_group } p\ X \cong \text{reduced_homology_group } p\ Y$
by (*simp add: homeomorphic_imp_homotopy_equivalent_space homotopy_equivalent_space_imp_isomorphic_reduced_homology_group*)

lemma *trivial_reduced_homology_group_empty*:
 $\text{topspace } X = \{\} \implies \text{trivial_group } (\text{reduced_homology_group } p\ X)$
by (*metis carrier_reduced_homology_group_subset group.trivial_group_alt group_reduced_homology_group_trivial_group_def trivial_homology_group_empty*)

lemma *homology_dimension_reduced*:
assumes $\text{topspace } X = \{a\}$
shows *trivial_group* $(\text{reduced_homology_group } p\ X)$
proof –
have *iso*: $(\text{hom_induced } p\ X\ \{\}\ (\text{discrete_topology } \{\{\}\})\ \{\}\ (\lambda x. ()))$
 $\in \text{iso } (\text{homology_group } p\ X)\ (\text{homology_group } p\ (\text{discrete_topology } \{\{\}\}))$
apply (*rule homeomorphic_map_homology_iso*)
apply (*force simp: homeomorphic_map_maps homeomorphic_maps_def assms*)
done
show *?thesis*
unfolding *reduced_homology_group_def*
by (*rule group.trivial_group_subgroup_generated*) (*use iso in <auto simp: iso_kernel_image>*)
qed

lemma *trivial_reduced_homology_group_contractible_space*:
 $\text{contractible_space } X \implies \text{trivial_group } (\text{reduced_homology_group } p\ X)$
apply (*simp add: contractible_eq_homotopy_equivalent_singleton_subtopology*)

```

apply (auto simp: trivial_reduced_homology_group_empty)
using isomorphic_group_triviality
by (metis (full_types) group_reduced_homology_group homology_dimension_reduced
homotopy_equivalent_space_imp_isomorphic_reduced_homology_groups path_connectedin_def
path_connectedin_singleton topspace_subtopology_subset)

```

lemma *image_reduced_homology_group*:

```

assumes topspace  $X \cap S \neq \{\}$ 
shows  $\text{hom\_induced } p \ X \ \{\} \ X \ S \ \text{id} \ \text{'carrier (reduced\_homology\_group } p \ X)$ 
 $\quad = \text{hom\_induced } p \ X \ \{\} \ X \ S \ \text{id} \ \text{'carrier (homology\_group } p \ X)$ 
(is  $?h \ \text{'carrier } ?G = ?h \ \text{'carrier } ?H)$ 
proof –
obtain  $a$  where  $a: a \in \text{topspace } X$  and  $a \in S$ 
using assms by blast
have  $[\text{simp}]: A \cap \{x \in A. P \ x\} = \{x \in A. P \ x\}$  for  $A \ P$ 
by blast
interpret comm_group homology_group  $p \ X$ 
by (rule abelian_relative_homology_group)
have  $*$ :  $\exists x'. ?h \ y = ?h \ x' \wedge$ 
 $\quad x' \in \text{carrier } ?H \wedge$ 
 $\quad \text{hom\_induced } p \ X \ \{\} \ (\text{discrete\_topology } \{\{\}\}) \ \{\} \ (\lambda x. ()) \ x'$ 
 $\quad = \mathbf{1}_{\text{homology\_group } p \ (\text{discrete\_topology } \{\{\}\})}$ 
if  $y \in \text{carrier } ?H$  for  $y$ 
proof –
let  $?f = \text{hom\_induced } p \ (\text{discrete\_topology } \{\{\}\}) \ \{\} \ X \ \{\} \ (\lambda x. a)$ 
let  $?g = \text{hom\_induced } p \ X \ \{\} \ (\text{discrete\_topology } \{\{\}\}) \ \{\} \ (\lambda x. ())$ 
have  $\text{bcarr}: ?f \ ( ?g \ y) \in \text{carrier } ?H$ 
by (simp add: hom_induced_carrier)
interpret gh1:
 $\text{group\_hom\_relative\_homology\_group } p \ X \ S \ \text{relative\_homology\_group } p$ 
 $\ (\text{discrete\_topology } \{\{\}\}) \ \{\{\}\}$ 
 $\quad \text{hom\_induced } p \ X \ S \ (\text{discrete\_topology } \{\{\}\}) \ \{\{\}\} \ (\lambda x. ())$ 
by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
interpret gh2:
 $\text{group\_hom\_relative\_homology\_group } p \ (\text{discrete\_topology } \{\{\}\}) \ \{\{\}\} \ \text{rela-}$ 
 $\text{tive\_homology\_group } p \ X \ S$ 
 $\quad \text{hom\_induced } p \ (\text{discrete\_topology } \{\{\}\}) \ \{\{\}\} \ X \ S \ (\lambda x. a)$ 
by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
interpret gh3:
 $\text{group\_hom\_homology\_group } p \ X \ \text{relative\_homology\_group } p \ X \ S \ ?h$ 
by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
interpret gh4:
 $\text{group\_hom\_homology\_group } p \ X \ \text{homology\_group } p \ (\text{discrete\_topology } \{\{\}\})$ 
 $\quad ?g$ 
by (meson group_hom_axioms_def group_hom_def hom_induced_hom

```



```

group_relative_homology_group)
interpret gh5:
  group_hom homology_group p (discrete_topology {}) homology_group p X
    ?f
  by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
interpret gh6:
  group_hom homology_group p (discrete_topology {}) relative_homology_group
p (discrete_topology {}) {}
    hom_induced p (discrete_topology {}) {} (discrete_topology {})
{} id
  by (meson group_hom_axioms_def group_hom_def hom_induced_hom
group_relative_homology_group)
show ?thesis
proof (intro exI conjI)
  have (?h ∘ ?f ∘ ?g) y
    = (hom_induced p (discrete_topology {}) {} X S (λx. a) ∘
    hom_induced p (discrete_topology {}) {} (discrete_topology {}) {})
id ∘ ?g) y
  by (simp add: a ⟨a ∈ S⟩ flip: hom_induced_compose)
  also have ... = 1relative_homology_group p X S
    using trivial_relative_homology_group_topospace [of p discrete_topology
{}]
  apply simp
  by (metis (full_types) empty_iff gh1.H.one_closed gh1.H.trivial_group
gh2.hom_one hom_induced_carrier insert_iff)
  finally have ?h (?f (?g y)) = 1relative_homology_group p X S
  by simp
  then show ?h y = ?h (y ⊗?H inv?H ?f (?g y))
  by (simp add: that hom_induced_carrier)
  show (y ⊗?H inv?H ?f (?g y)) ∈ carrier (homology_group p X)
  by (simp add: hom_induced_carrier that)
  have *: (?g ∘ hom_induced p X {} X {} (λx. a)) y = hom_induced p X {}
(discrete_topology {}) {} (λa. ()) y
  by (simp add: a ⟨a ∈ S⟩ flip: hom_induced_compose)
  have ?g (y ⊗?H inv?H (?f ∘ ?g) y)
    = 1homology_group p (discrete_topology {})
  by (simp add: a ⟨a ∈ S⟩ that hom_induced_carrier flip: hom_induced_compose
* [unfolded o_def])
  then show ?g (y ⊗?H inv?H ?f (?g y))
    = 1homology_group p (discrete_topology {})
  by simp
qed
qed
show ?thesis
  apply (auto simp: reduced_homology_group_def carrier_subgroup_generated
kernel_def image_iff)
  apply (metis (no_types, lifting) generate_in_carrier mem_Collect_eq subsetI)
  apply (force simp: dest: * intro: generate.incl)

```

done
qed

lemma *homology_exactness_reduced_1*:
assumes *topspace* $X \cap S \neq \{\}$
shows *exact_seq*([*reduced_homology_group*($p - 1$) (*subtopology* $X S$),
relative_homology_group $p X S$,
reduced_homology_group $p X$],
[*hom_boundary* $p X S$, *hom_induced* $p X \{\}$ $X S id$])
(is *exact_seq* ([$?G1, ?G2, ?G3$], [$?h1, ?h2$]))
proof –
have *: $?h2$ ‘ *carrier* (*homology_group* $p X$)
= *kernel* $?G2$ (*homology_group* ($p - 1$) (*subtopology* $X S$)) $?h1$
using *homology_exactness_axiom_1* [of $p X S$] **by** *simp*
have *gh*: *group_hom* $?G3 ?G2 ?h2$
by (*simp add*: *reduced_homology_group_def* *group_hom_def* *group_hom_axioms_def*
group.group_subgroup_generated *group.hom_from_subgroup_generated* *hom_induced_hom*)
show $?thesis$
apply (*simp add*: *hom_boundary_reduced_hom* *gh* * *image_reduced_homology_group*
[*OF assms*])
apply (*simp add*: *kernel_def one_reduced_homology_group*)
done
qed

lemma *homology_exactness_reduced_2*:
exact_seq([*reduced_homology_group*($p - 1$) X ,
reduced_homology_group($p - 1$) (*subtopology* $X S$),
relative_homology_group $p X S$],
[*hom_induced* ($p - 1$) (*subtopology* $X S$) $\{\}$ $X \{\}$ *id*, *hom_boundary*
 $p X S$])
(is *exact_seq* ([$?G1, ?G2, ?G3$], [$?h1, ?h2$]))
using *homology_exactness_axiom_2* [of $p X S$]
apply (*simp add*: *group_hom_axioms_def* *group_hom_def* *hom_boundary_reduced_hom*
hom_induced_reduced_hom)
apply (*simp add*: *reduced_homology_group_def* *group_hom.subgroup_kernel* *group_hom_axioms_def*
group_hom_def *hom_induced_hom*)
using *hom_boundary_reduced_hom* [of $p X S$]
apply (*auto simp*: *image_def set_eq_iff*)
by (*metis* *carrier_reduced_homology_group* *hom_in_carrier set_eq_iff*)

lemma *homology_exactness_reduced_3*:
exact_seq([*relative_homology_group* $p X S$,
reduced_homology_group $p X$,
reduced_homology_group p (*subtopology* $X S$)],
[*hom_induced* $p X \{\}$ $X S id$, *hom_induced* p (*subtopology* $X S$) $\{\}$ X
 $\{\}$ *id*])

```

  (is_exact_seq ([?G1,?G2,?G3], [?h1,?h2]))
proof -
  have kernel ?G2 ?G1 ?h1 =
    ?h2 'carrier ?G3
  proof -
    obtain U where U:
      (hom_induced p (subtopology X S) {} X {} id) 'carrier ?G3 ⊆ U
      (hom_induced p (subtopology X S) {} X {} id) 'carrier ?G3
      ⊆ (hom_induced p (subtopology X S) {} X {} id) 'carrier (homology_group
p (subtopology X S))
      U ∩ kernel (homology_group p X) ?G1 (hom_induced p X {} X S id)
      = kernel ?G2 ?G1 (hom_induced p X {} X S id)
      U ∩ (hom_induced p (subtopology X S) {} X {} id) 'carrier (homology_group
p (subtopology X S))
      ⊆ (hom_induced p (subtopology X S) {} X {} id) 'carrier ?G3
    proof
      show ?h2 'carrier ?G3 ⊆ carrier ?G2
        by (simp add: hom_induced_reduced_image_subset_iff)
      show ?h2 'carrier ?G3 ⊆ ?h2 'carrier (homology_group p (subtopology X
S))
        by (meson carrier_reduced_homology_group_subset_image_mono)
      have subgroup (kernel (homology_group p X) (homology_group p (discrete_topology
{})))
        (hom_induced p X {} (discrete_topology {}) {} (λx. ()))
        (homology_group p X)
        by (simp add: group.normal_invE(1) group_hom.normal_kernel_group_hom_axioms_def
group_hom_def hom_induced_empty_hom)
      then show carrier ?G2 ∩ kernel (homology_group p X) ?G1 ?h1 = kernel
?G2 ?G1 ?h1
        unfolding carrier_reduced_homology_group
        by (auto simp: reduced_homology_group_def)
      show carrier ?G2 ∩ ?h2 'carrier (homology_group p (subtopology X S))
        ⊆ ?h2 'carrier ?G3
        by (force simp: carrier_reduced_homology_group_kernel_def hom_induced_compose')
    qed
  qed
  with homology_exactness_axiom_3 [of p X S] show ?thesis
    by (fastforce simp add:)
qed
then show ?thesis
  apply (simp add: group_hom_axioms_def group_hom_def hom_boundary_reduced_hom
hom_induced_reduced_hom)
  apply (simp add: group_hom_from_subgroup_generated hom_induced_hom
reduced_homology_group_def)
  done
qed

```

0.3.2 More homology properties of deformations, retracts, contractible spaces

lemma *iso_relative_homology_of_contractible:*

$\llbracket \text{contractible_space } X; \text{topspace } X \cap S \neq \{\} \rrbracket$
 $\implies \text{hom_boundary } p \ X \ S$
 $\in \text{iso } (\text{relative_homology_group } p \ X \ S) \ (\text{reduced_homology_group}(p - 1) \ (\text{subtopology } X \ S))$

using *very_short_exact_sequence*

$[\text{of reduced_homology_group } (p - 1) \ X$
 $\text{reduced_homology_group } (p - 1) \ (\text{subtopology } X \ S)$
 $\text{relative_homology_group } p \ X \ S$
 $\text{reduced_homology_group } p \ X$
 $\text{hom_induced } (p - 1) \ (\text{subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id}$
 $\text{hom_boundary } p \ X \ S$
 $\text{hom_induced } p \ X \ \{\} \ X \ S \ \text{id}]$

by (*meson exact_seq_cons_iff homology_exactness_reduced_1 homology_exactness_reduced_2 trivial_reduced_homology_group_contractible_space*)

lemma *isomorphic_group_relative_homology_of_contractible:*

$\llbracket \text{contractible_space } X; \text{topspace } X \cap S \neq \{\} \rrbracket$
 $\implies \text{relative_homology_group } p \ X \ S \cong$
 $\text{reduced_homology_group}(p - 1) \ (\text{subtopology } X \ S)$
by (*meson iso_relative_homology_of_contractible is_isoI*)

lemma *isomorphic_group_reduced_homology_of_contractible:*

$\llbracket \text{contractible_space } X; \text{topspace } X \cap S \neq \{\} \rrbracket$
 $\implies \text{reduced_homology_group } p \ (\text{subtopology } X \ S) \cong \text{relative_homology_group}(p + 1) \ X \ S$

by (*metis add.commute add_diff_cancel_left' group.iso_sym group_relative_homology_group isomorphic_group_relative_homology_of_contractible*)

lemma *iso_reduced_homology_by_contractible:*

$\llbracket \text{contractible_space}(\text{subtopology } X \ S); \text{topspace } X \cap S \neq \{\} \rrbracket$
 $\implies (\text{hom_induced } p \ X \ \{\} \ X \ S \ \text{id}) \in \text{iso } (\text{reduced_homology_group } p \ X)$
 $(\text{relative_homology_group } p \ X \ S)$

using *very_short_exact_sequence*

$[\text{of reduced_homology_group } (p - 1) \ (\text{subtopology } X \ S)$
 $\text{relative_homology_group } p \ X \ S$
 $\text{reduced_homology_group } p \ X$
 $\text{reduced_homology_group } p \ (\text{subtopology } X \ S)$
 $\text{hom_boundary } p \ X \ S$
 $\text{hom_induced } p \ X \ \{\} \ X \ S \ \text{id}$
 $\text{hom_induced } p \ (\text{subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id}]$

by (*meson exact_seq_cons_iff homology_exactness_reduced_1 homology_exactness_reduced_3 trivial_reduced_homology_group_contractible_space*)

lemma *isomorphic_reduced_homology_by_contractible:*

$\llbracket \text{contractible_space}(\text{subtopology } X \ S); \text{topspace } X \cap S \neq \{\} \rrbracket$
 $\implies \text{reduced_homology_group } p \ X \cong \text{relative_homology_group } p \ X \ S$

using is_isoI iso_reduced_homology_by_contractible by blast

lemma isomorphic_relative_homology_by_contractible:

$\llbracket \text{contractible_space}(\text{subtopology } X \ S); \text{topspace } X \cap S \neq \{\} \rrbracket$

$\implies \text{relative_homology_group } p \ X \ S \cong \text{reduced_homology_group } p \ X$

using group.iso_sym group_reduced_homology_group isomorphic_reduced_homology_by_contractible by blast

lemma isomorphic_reduced_homology_by_singleton:

$a \in \text{topspace } X \implies \text{reduced_homology_group } p \ X \cong \text{relative_homology_group } p \ X \ (\{a\})$

by (simp add: contractible_space_subtopology_singleton isomorphic_reduced_homology_by_contractible)

lemma isomorphic_relative_homology_by_singleton:

$a \in \text{topspace } X \implies \text{relative_homology_group } p \ X \ (\{a\}) \cong \text{reduced_homology_group } p \ X$

by (simp add: group.iso_sym isomorphic_reduced_homology_by_singleton)

lemma reduced_homology_group_pair:

assumes $t1_space \ X$ and $a: a \in \text{topspace } X$ and $b: b \in \text{topspace } X$ and $a \neq b$

shows $\text{reduced_homology_group } p \ (\text{subtopology } X \ \{a, b\}) \cong \text{homology_group } p \ (\text{subtopology } X \ \{a\})$

(is ?lhs \cong ?rhs)

proof –

have $?lhs \cong \text{relative_homology_group } p \ (\text{subtopology } X \ \{a, b\}) \ \{b\}$

by (simp add: b isomorphic_reduced_homology_by_singleton topspace_subtopology)

also have $\dots \cong ?rhs$

proof –

have $\text{sub: subtopology } X \ \{a, b\} \ \text{closure_of } \{b\} \subseteq \text{subtopology } X \ \{a, b\} \ \text{interior_of } \{b\}$

by (simp add: assms t1_space_subtopology closure_of_singleton subtopology_eq_discrete_topology_finite discrete_topology_closure_of)

show ?thesis

using homology_excision_axiom [OF sub, of $\{a, b\}$ p]

by (simp add: assms(4) group.iso_sym is_isoI subtopology_subtopology)

qed

finally show ?thesis .

qed

lemma deformation_retraction_relative_homology_group_isomorphisms:

$\llbracket \text{retraction_maps } X \ Y \ r \ s; r \ ' U \subseteq V; s \ ' V \subseteq U; \text{homotopic_with } (\lambda h. h \ ' U \subseteq U) \ X \ X \ (s \circ r) \ id \rrbracket$

$\implies \text{group_isomorphisms } (\text{relative_homology_group } p \ X \ U) \ (\text{relative_homology_group } p \ Y \ V)$

$(\text{hom_induced } p \ X \ U \ Y \ V \ r) \ (\text{hom_induced } p \ Y \ V \ X \ U \ s)$

apply (simp add: retraction_maps_def)

apply (rule homotopy_equivalence_relative_homology_group_isomorphisms)

apply (auto simp: image_subset_iff continuous_map_compose homotopic_with_equal)

done

lemma *deformation_retract_relative_homology_group_isomorphisms:*

$\llbracket \text{retraction_maps } X \ Y \ r \ \text{id}; \ V \subseteq U; \ r \text{ ' } U \subseteq V; \text{homotopic_with } (\lambda h. \ h \text{ ' } U \subseteq U) \ X \ X \ r \ \text{id} \rrbracket$

$\implies \text{group_isomorphisms } (\text{relative_homology_group } p \ X \ U) \ (\text{relative_homology_group } p \ Y \ V)$

$(\text{hom_induced } p \ X \ U \ Y \ V \ r) \ (\text{hom_induced } p \ Y \ V \ X \ U \ \text{id})$

by (*simp add: deformation_retraction_relative_homology_group_isomorphisms*)

lemma *deformation_retract_relative_homology_group_isomorphism:*

$\llbracket \text{retraction_maps } X \ Y \ r \ \text{id}; \ V \subseteq U; \ r \text{ ' } U \subseteq V; \text{homotopic_with } (\lambda h. \ h \text{ ' } U \subseteq U) \ X \ X \ r \ \text{id} \rrbracket$

$\implies (\text{hom_induced } p \ X \ U \ Y \ V \ r) \in \text{iso } (\text{relative_homology_group } p \ X \ U) \ (\text{relative_homology_group } p \ Y \ V)$

by (*metis deformation_retract_relative_homology_group_isomorphisms group_isomorphisms_imp_is*)

lemma *deformation_retract_relative_homology_group_isomorphism_id:*

$\llbracket \text{retraction_maps } X \ Y \ r \ \text{id}; \ V \subseteq U; \ r \text{ ' } U \subseteq V; \text{homotopic_with } (\lambda h. \ h \text{ ' } U \subseteq U) \ X \ X \ r \ \text{id} \rrbracket$

$\implies (\text{hom_induced } p \ Y \ V \ X \ U \ \text{id}) \in \text{iso } (\text{relative_homology_group } p \ Y \ V) \ (\text{relative_homology_group } p \ X \ U)$

by (*metis deformation_retract_relative_homology_group_isomorphisms group_isomorphisms_imp_is group_isomorphisms_sym*)

lemma *deformation_retraction_imp_isomorphic_relative_homology_groups:*

$\llbracket \text{retraction_maps } X \ Y \ r \ s; \ r \text{ ' } U \subseteq V; \ s \text{ ' } V \subseteq U; \text{homotopic_with } (\lambda h. \ h \text{ ' } U \subseteq U) \ X \ X \ (s \circ r) \ \text{id} \rrbracket$

$\implies \text{relative_homology_group } p \ X \ U \cong \text{relative_homology_group } p \ Y \ V$

by (*blast intro: is_isoI group_isomorphisms_imp_iso deformation_retraction_relative_homology_group*)

lemma *deformation_retraction_imp_isomorphic_homology_groups:*

$\llbracket \text{retraction_maps } X \ Y \ r \ s; \text{homotopic_with } (\lambda h. \ \text{True}) \ X \ X \ (s \circ r) \ \text{id} \rrbracket$

$\implies \text{homology_group } p \ X \cong \text{homology_group } p \ Y$

by (*simp add: deformation_retraction_imp_homotopy_equivalent_space homotopy_equivalent_space_imp_isomorphic_homology_groups*)

lemma *deformation_retract_imp_isomorphic_relative_homology_groups:*

$\llbracket \text{retraction_maps } X \ X' \ r \ \text{id}; \ V \subseteq U; \ r \text{ ' } U \subseteq V; \text{homotopic_with } (\lambda h. \ h \text{ ' } U \subseteq U) \ X \ X \ r \ \text{id} \rrbracket$

$\implies \text{relative_homology_group } p \ X \ U \cong \text{relative_homology_group } p \ X' \ V$

by (*simp add: deformation_retraction_imp_isomorphic_relative_homology_groups*)

lemma *deformation_retract_imp_isomorphic_homology_groups:*

$\llbracket \text{retraction_maps } X \ X' \ r \ \text{id}; \text{homotopic_with } (\lambda h. \ \text{True}) \ X \ X \ r \ \text{id} \rrbracket$

$\implies \text{homology_group } p \ X \cong \text{homology_group } p \ X'$

by (*simp add: deformation_retraction_imp_isomorphic_homology_groups*)

lemma *epi_hom_induced_inclusion*:

assumes *homotopic_with* $(\lambda x. \text{True})$ X X *id* f **and** $f' : (\text{topspace } X) \subseteq S$
shows $(\text{hom_induced } p \text{ (subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id})$
 $\in \text{epi } (\text{homology_group } p \text{ (subtopology } X \ S)) \ (\text{homology_group } p \ X)$
proof (rule *epi_right_invertible*)
show $\text{hom_induced } p \text{ (subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id}$
 $\in \text{hom } (\text{homology_group } p \text{ (subtopology } X \ S)) \ (\text{homology_group } p \ X)$
by (simp add: *hom_induced_empty_hom*)
show $\text{hom_induced } p \ X \ \{\} \ \text{(subtopology } X \ S) \ \{\} \ f$
 $\in \text{carrier } (\text{homology_group } p \ X) \rightarrow \text{carrier } (\text{homology_group } p \ \text{(subtopology } X \ S))$
by (simp add: *hom_induced_carrier*)
fix x
assume $x \in \text{carrier } (\text{homology_group } p \ X)$
then show $\text{hom_induced } p \ \text{(subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id} \ (\text{hom_induced } p \ X \ \{\} \ \text{(subtopology } X \ S) \ \{\} \ f \ x) = x$
by (metis *assms continuous_map_id_subt continuous_map_in_subtopology hom_induced_compose' hom_induced_id homology_homotopy_empty homotopic_with_imp_continuous_maps image_empty order_refl*)
qed

lemma *trivial_homomorphism_hom_induced_relativization*:

assumes *homotopic_with* $(\lambda x. \text{True})$ X X *id* f **and** $f' : (\text{topspace } X) \subseteq S$
shows *trivial_homomorphism* $(\text{homology_group } p \ X) \ (\text{relative_homology_group } p \ X \ S)$
 $(\text{hom_induced } p \ X \ \{\} \ X \ S \ \text{id})$
proof –
have $(\text{hom_induced } p \ \text{(subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id})$
 $\in \text{epi } (\text{homology_group } p \ \text{(subtopology } X \ S)) \ (\text{homology_group } p \ X)$
by (metis *assms epi_hom_induced_inclusion*)
then show ?thesis
using *homology_exactness_axiom_3* [of $p \ X \ S$] *homology_exactness_axiom_1* [of $p \ X \ S$]
by (simp add: *epi_def group.trivial_homomorphism_image group_hom.trivial_hom_iff*)
qed

lemma *mon_hom_boundary_inclusion*:

assumes *homotopic_with* $(\lambda x. \text{True})$ X X *id* f **and** $f' : (\text{topspace } X) \subseteq S$
shows $(\text{hom_boundary } p \ X \ S) \in \text{mon}$
 $(\text{relative_homology_group } p \ X \ S) \ (\text{homology_group } (p - 1) \ \text{(subtopology } X \ S))$
proof –
have $(\text{hom_induced } p \ \text{(subtopology } X \ S) \ \{\} \ X \ \{\} \ \text{id})$
 $\in \text{epi } (\text{homology_group } p \ \text{(subtopology } X \ S)) \ (\text{homology_group } p \ X)$
by (metis *assms epi_hom_induced_inclusion*)
then show ?thesis

```

using homology_exactness_axiom_3 [of p X S] homology_exactness_axiom_1
[of p X S]
apply (simp add: mon_def epi_def hom_boundary_hom)
by (metis (no_types, opaque_lifting) group_hom.trivial_hom_iff group_hom.trivial_ker_imp_inj
group_hom_axioms_def group_hom_def group_relative_homology_group hom_boundary_hom)
qed

```

```

lemma short_exact_sequence_hom_induced_relativization:
assumes homotopic_with ( $\lambda x.$  True) X X id f and f ' ( $\text{topspace } X$ )  $\subseteq S$ 
shows short_exact_sequence (homology_group (p-1) X) (homology_group (p-1)
(subtopology X S)) (relative_homology_group p X S)
(hom_induced (p-1) (subtopology X S) {} X {} id) (hom_boundary
p X S)
unfolding short_exact_sequence_iff
by (intro conjI homology_exactness_axiom_2 epi_hom_induced_inclusion [OF
assms] mon_hom_boundary_inclusion [OF assms])

```

```

lemma group_isomorphisms_homology_group_prod_deformation:
fixes p::int
assumes homotopic_with ( $\lambda x.$  True) X X id f and f ' ( $\text{topspace } X$ )  $\subseteq S$ 
obtains H K where
subgroup H (homology_group p (subtopology X S))
subgroup K (homology_group p (subtopology X S))
( $\lambda(x, y).$   $x \otimes_{\text{homology\_group } p \text{ (subtopology } X \text{ S)}} y$ )
 $\in \text{Group.iso (subgroup\_generated (homology\_group } p \text{ (subtopology } X \text{ S)) } K)$ 
 $H \times \times$ 
subgroup_generated (homology_group p (subtopology X S)) K)
(homology_group p (subtopology X S))
hom_boundary (p + 1) X S
 $\in \text{Group.iso (relative\_homology\_group (p + 1) X S)}$ 
(subgroup_generated (homology_group p (subtopology X S)) H)
hom_induced p (subtopology X S) {} X {} id
 $\in \text{Group.iso}$ 
(subgroup_generated (homology_group p (subtopology X S)) K)
(homology_group p X)

```

```

proof -
let ?rhs = relative_homology_group (p + 1) X S
let ?pXS = homology_group p (subtopology X S)
let ?pX = homology_group p X
let ?hb = hom_boundary (p + 1) X S
let ?hi = hom_induced p (subtopology X S) {} X {} id
have x: short_exact_sequence (?pX) ?pXS ?rhs ?hi ?hb
using short_exact_sequence_hom_induced_relativization [OF assms, of p +
1] by simp
have conf: continuous_map X (subtopology X S) f
by (meson assms continuous_map_in_subtopology homotopic_with_imp_continuous_maps)
obtain H K where HK:  $H \triangleleft ?pXS$  subgroup K  $?pXS \cap K \subseteq \{one \ ?pXS\}$ 
set_mult ?pXS H K = carrier ?pXS

```



```

and iso: ?hb ∈ iso ?rhs (subgroup_generated ?pXS H) ?hi ∈ iso (subgroup_generated
?pXS K) ?pX
  apply (rule splitting_lemma_right [OF x, where g' = hom_induced p X {}]
(subtopology X S) {} f])
  apply (simp add: hom_induced_empty_hom)
  apply (simp add: contf_hom_induced_compose')
  apply (metis (full_types) assms(1) hom_induced_id homology_homotopy_empty)
  apply blast
  done
show ?thesis
proof
  show subgroup H ?pXS
    using HK(1) normal_imp_subgroup by blast
  then show (λ(x, y). x ⊗ ?pXS y)
    ∈ Group.iso (subgroup_generated (?pXS) H ×× subgroup_generated (?pXS)
K) (?pXS)
    by (meson HK abelian_relative_homology_group group_disjoint_sum.iso_group_mul
group_disjoint_sum_def group_relative_homology_group)
  show subgroup K ?pXS
    by (rule HK)
  show hom_boundary (p + 1) X S ∈ Group.iso ?rhs (subgroup_generated
(?pXS) H)
    using iso_int_ops(4) by presburger
  show hom_induced p (subtopology X S) {} X {} id ∈ Group.iso (subgroup_generated
(?pXS) K) (?pX)
    by (simp add: iso(2))
  qed
qed

lemma iso_homology_group_prod_deformation:
assumes homotopic_with (λx. True) X X id f and f ' (topspace X) ⊆ S
shows homology_group p (subtopology X S)
  ≅ DirProd (homology_group p X) (relative_homology_group(p + 1) X S)
  (is ?G ≅ DirProd ?H ?R)
proof –
  obtain H K where HK:
    (λ(x, y). x ⊗ ?G y)
    ∈ Group.iso (subgroup_generated (?G) H ×× subgroup_generated (?G) K)
(?G)
    hom_boundary (p + 1) X S ∈ Group.iso (?R) (subgroup_generated (?G) H)
    hom_induced p (subtopology X S) {} X {} id ∈ Group.iso (subgroup_generated
(?G) K) (?H)
    by (blast intro: group_isomorphisms_homology_group_prod_deformation [OF
assms])
  have ?G ≅ DirProd (subgroup_generated (?G) H) (subgroup_generated (?G)
K)
    by (meson DirProd_group HK(1) group.group_subgroup_generated group.iso_sym
group_relative_homology_group is_isoI)
  also have ... ≅ DirProd ?R ?H

```

```

    by (meson HK group.DirProd_iso_trans group.group_subgroup_generated group.iso_sym
group_relative_homology_group is_isoI)
    also have ...  $\cong$  DirProd ?H ?R
    by (simp add: DirProd_commute_iso)
    finally show ?thesis .
qed

```

```

lemma iso_homology_contractible_space_subtopology1:
  assumes contractible_space X S  $\subseteq$  topspace X S  $\neq$  {}
  shows homology_group 0 (subtopology X S)  $\cong$  DirProd integer_group (relative_homology_group(1)
X S)
proof -
  obtain f where homotopic_with ( $\lambda x$ . True) X X id f and f' (topspace X)  $\subseteq$  S
  using assms contractible_space_alt by fastforce
  then have homology_group 0 (subtopology X S)  $\cong$  homology_group 0 X  $\times \times$ 
relative_homology_group 1 X S
  using iso_homology_group_prod_deformation [of X _ S 0] by auto
  also have ...  $\cong$  integer_group  $\times \times$  relative_homology_group 1 X S
  using assms contractible_imp_path_connected_space group.DirProd_iso_trans
group_relative_homology_group iso_refl isomorphic_integer_zeroth_homology_group
by blast
  finally show ?thesis .
qed

```

```

lemma iso_homology_contractible_space_subtopology2:
  [[contractible_space X; S  $\subseteq$  topspace X; p  $\neq$  0; S  $\neq$  {}]]
   $\implies$  homology_group p (subtopology X S)  $\cong$  relative_homology_group (p + 1)
X S
  by (metis (no_types, opaque_lifting) add.commute isomorphic_group_reduced_homology_of_contractible
topspace_subtopology topspace_subtopology_subset un_reduced_homology_group)

```

```

lemma trivial_relative_homology_group_contractible_spaces:
  [[contractible_space X; contractible_space(subtopology X S); topspace X  $\cap$  S  $\neq$ 
{}]]
   $\implies$  trivial_group(relative_homology_group p X S)
  using group_reduced_homology_group group_relative_homology_group isomor-
phic_group_triviality isomorphic_relative_homology_by_contractible trivial_reduced_homology_group
by blast

```

```

lemma trivial_relative_homology_group_alt:
  assumes contf: continuous_map X (subtopology X S) f and hom: homotopic_with
( $\lambda k$ . k' S  $\subseteq$  S) X X f id
  shows trivial_group (relative_homology_group p X S)
proof (rule trivial_relative_homology_group_gen [OF contf])
  show homotopic_with ( $\lambda h$ . True) (subtopology X S) (subtopology X S) f id
  using hom unfolding homotopic_with_def
  apply (rule ex_forward)

```

```

apply (auto simp: prod_topology_subtopology continuous_map_in_subtopology
continuous_map_from_subtopology image_subset_iff topspace_subtopology)
done
show homotopic_with ( $\lambda k. \text{True}$ )  $X X f id$ 
using assms by (force simp: homotopic_with_def)
qed

```

lemma iso_hom_induced_relativization_contractible:

```

assumes contractible_space(subtopology  $X S$ ) contractible_space(subtopology  $X T$ )  $T \subseteq S$   $\text{topspace } X \cap T \neq \{\}$ 
shows ( $\text{hom\_induced } p X T X S id$ )  $\in$  iso ( $\text{relative\_homology\_group } p X T$ )
( $\text{relative\_homology\_group } p X S$ )
proof (rule very_short_exact_sequence)
show exact_seq
  ([ $\text{relative\_homology\_group}(p - 1)(\text{subtopology } X S) T, \text{relative\_homology\_group}$ 
 $p X S, \text{relative\_homology\_group } p X T, \text{relative\_homology\_group } p(\text{subtopology}$ 
 $X S) T$ ],
  [ $\text{hom\_relboundary } p X S T, \text{hom\_induced } p X T X S id, \text{hom\_induced } p$ 
( $\text{subtopology } X S) T X T id$ ])
using homology_exactness_triple_1 [ $OF \langle T \subseteq S \rangle$ ] homology_exactness_triple_3
[ $OF \langle T \subseteq S \rangle$ ]
by fastforce
show trivial_group ( $\text{relative\_homology\_group } p(\text{subtopology } X S) T$ ) trivial_group
( $\text{relative\_homology\_group}(p - 1)(\text{subtopology } X S) T$ )
using assms
by (force simp: inf.absorb_iff2 subtopology_subtopology topspace_subtopology
intro!: trivial_relative_homology_group_contractible_spaces)+
qed

```

corollary isomorphic_relative_homology_groups_relativization_contractible:

```

assumes contractible_space(subtopology  $X S$ ) contractible_space(subtopology  $X T$ )  $T \subseteq S$   $\text{topspace } X \cap T \neq \{\}$ 
shows  $\text{relative\_homology\_group } p X T \cong \text{relative\_homology\_group } p X S$ 
by (rule is_isoI) (rule iso_hom_induced_relativization_contractible [ $OF$  assms])

```

lemma iso_hom_induced_inclusion_contractible:

```

assumes contractible_space  $X$  contractible_space(subtopology  $X S$ )  $T \subseteq S$   $\text{topspace } X \cap S \neq \{\}$ 
shows ( $\text{hom\_induced } p(\text{subtopology } X S) T X T id$ )
 $\in$  iso ( $\text{relative\_homology\_group } p(\text{subtopology } X S) T$ ) ( $\text{relative\_homology\_group}$ 
 $p X T$ )
proof (rule very_short_exact_sequence)
show exact_seq
  ([ $\text{relative\_homology\_group } p X S, \text{relative\_homology\_group } p X T,$ 
 $\text{relative\_homology\_group } p(\text{subtopology } X S) T, \text{relative\_homology\_group}$ 
 $(p+1) X S$ ],
  [ $\text{hom\_induced } p X T X S id, \text{hom\_induced } p(\text{subtopology } X S) T X T id,$ 
 $\text{hom\_relboundary } (p+1) X S T$ ])

```

```

using homology_exactness_triple_2 [OF  $\langle T \subseteq S \rangle$ ] homology_exactness_triple_3
[OF  $\langle T \subseteq S \rangle$ ]
by (metis add_diff_cancel_left' diff_add_cancel exact_seq_cons_iff)
show trivial_group (relative_homology_group (p+1) X S) trivial_group (relative_homology_group
p X S)
using assms
by (auto simp: subtopology_subtopology topspace_subtopology intro!: trivial_relative_homology_group)
qed

```

corollary *isomorphic_relative_homology_groups_inclusion_contractible:*
assumes contractible_space X contractible_space(subtopology X S) $T \subseteq S$ topspace
 $X \cap S \neq \{\}$
shows relative_homology_group p (subtopology X S) $T \cong$ relative_homology_group
p X T
by (rule is_isoI) (rule iso_hom_induced_inclusion_contractible [OF assms])

lemma *iso_hom_relboundary_contractible:*
assumes contractible_space X contractible_space(subtopology X T) $T \subseteq S$ topspace
 $X \cap T \neq \{\}$
shows hom_relboundary p X S T
 \in iso (relative_homology_group p X S) (relative_homology_group (p - 1)
(subtopology X S) T)
proof (rule very_short_exact_sequence)
show exact_seq
([relative_homology_group (p - 1) X T, relative_homology_group (p - 1)
(subtopology X S) T, relative_homology_group p X S, relative_homology_group p
X T],
[hom_induced (p - 1) (subtopology X S) T X T id, hom_relboundary p X
S T, hom_induced p X T X S id])
using homology_exactness_triple_1 [OF $\langle T \subseteq S \rangle$] homology_exactness_triple_2
[OF $\langle T \subseteq S \rangle$] **by** simp
show trivial_group (relative_homology_group p X T) trivial_group (relative_homology_group
(p - 1) X T)
using assms
by (auto simp: subtopology_subtopology topspace_subtopology intro!: trivial_relative_homology_group)
qed

corollary *isomorphic_relative_homology_groups_relboundary_contractible:*
assumes contractible_space X contractible_space(subtopology X T) $T \subseteq S$ topspace
 $X \cap T \neq \{\}$
shows relative_homology_group p X S \cong relative_homology_group (p - 1)
(subtopology X S) T
by (rule is_isoI) (rule iso_hom_relboundary_contractible [OF assms])

lemma *isomorphic_relative_contractible_space_imp_homology_groups:*
assumes contractible_space X contractible_space Y $S \subseteq$ topspace X $T \subseteq$ topspace
Y
and ST: $S = \{\} \longleftrightarrow T = \{\}$
and iso: $\bigwedge p.$ relative_homology_group p X S \cong relative_homology_group p Y

```

T
  shows homology_group p (subtopology X S)  $\cong$  homology_group p (subtopology Y
T)
proof (cases T = {})
  case True
    have homology_group p (subtopology X {})  $\cong$  homology_group p (subtopology Y
{})
    by (simp add: homeomorphic_empty_space_eq homeomorphic_space_imp_isomorphic_homology_groups)
    then show ?thesis
      using ST True by blast
  next
    case False
    show ?thesis
    proof (cases p = 0)
      case True
        have homology_group p (subtopology X S)  $\cong$  integer_group  $\times \times$  relative_homology_group
1 X S
        using assms True  $\langle T \neq \{\} \rangle$ 
        by (simp add: iso_homology_contractible_space_subtopology1)
        also have ...  $\cong$  integer_group  $\times \times$  relative_homology_group 1 Y T
        by (simp add: assms group.DirProd_iso_trans iso_refl)
        also have ...  $\cong$  homology_group p (subtopology Y T)
        by (simp add: True  $\langle T \neq \{\} \rangle$  assms group.iso_sym iso_homology_contractible_space_subtopology1)
        finally show ?thesis .
      next
        case False
        have homology_group p (subtopology X S)  $\cong$  relative_homology_group (p+1)
X S
        using assms False  $\langle T \neq \{\} \rangle$ 
        by (simp add: iso_homology_contractible_space_subtopology2)
        also have ...  $\cong$  relative_homology_group (p+1) Y T
        by (simp add: assms)
        also have ...  $\cong$  homology_group p (subtopology Y T)
        by (simp add: False  $\langle T \neq \{\} \rangle$  assms group.iso_sym iso_homology_contractible_space_subtopology2)
        finally show ?thesis .
    qed
  qed
qed

```

0.3.3 Homology groups of spheres

lemma *iso_reduced_homology_group_lower_hemisphere:*

```

  assumes  $k \leq n$ 
  shows hom_induced p (nsphere n) {} (nsphere n) {x. x k  $\leq$  0} id
     $\in$  iso (reduced_homology_group p (nsphere n)) (relative_homology_group p
(nsphere n) {x. x k  $\leq$  0})
proof (rule iso_reduced_homology_by_contractible)
  show contractible_space (subtopology (nsphere n) {x. x k  $\leq$  0})
    by (simp add: assms contractible_space_lower_hemisphere)
  have ( $\lambda i.$  if  $i = k$  then  $-1$  else  $0$ )  $\in$  topspace (nsphere n)  $\cap$  {x. x k  $\leq$  0}

```

```

    using assms by (simp add: nsphere if_distrib [of  $\lambda x. x^2$ ] cong: if_cong)
  then show  $\text{topspace } (\text{nsphere } n) \cap \{x. x \cdot k \leq 0\} \neq \{\}$ 
    by blast
qed

```

```

lemma topspace_nsphere_1:
  assumes  $x \in \text{topspace } (\text{nsphere } n)$  shows  $(x \cdot k)^2 \leq 1$ 
proof (cases  $k \leq n$ )
  case True
  have  $(\sum i \in \{..n\} - \{k\}. (x \cdot i)^2) = (\sum i \leq n. (x \cdot i)^2) - (x \cdot k)^2$ 
    using  $\langle k \leq n \rangle$  by (simp add: sum_diff)
  then show ?thesis
    using assms
    apply (simp add: nsphere)
    by (metis diff_ge_0_iff_ge sum_nonneg zero_le_power2)
  next
  case False
  then show ?thesis
    using assms by (simp add: nsphere)
qed

```

```

lemma topspace_nsphere_1_eq_0:
  fixes  $x :: \text{nat} \Rightarrow \text{real}$ 
  assumes  $x: x \in \text{topspace } (\text{nsphere } n)$  and  $xk: (x \cdot k)^2 = 1$  and  $i \neq k$ 
  shows  $x \cdot i = 0$ 
proof (cases  $i \leq n$ )
  case True
  have  $k \leq n$ 
    using  $x$ 
    by (simp add: nsphere) (metis not_less xk zero_neq_one zero_power2)
  have  $(\sum i \in \{..n\} - \{k\}. (x \cdot i)^2) = (\sum i \leq n. (x \cdot i)^2) - (x \cdot k)^2$ 
    using  $\langle k \leq n \rangle$  by (simp add: sum_diff)
  also have  $\dots = 0$ 
    using assms by (simp add: nsphere)
  finally have  $\forall i \in \{..n\} - \{k\}. (x \cdot i)^2 = 0$ 
    by (simp add: sum_nonneg_eq_0_iff)
  then show ?thesis
    using True  $\langle i \neq k \rangle$  by auto
  next
  case False
  with  $x$  show ?thesis
    by (simp add: nsphere)
qed

```

proposition *iso_relative_homology_group_upper_hemisphere*:
 $(\text{hom_induced } p (\text{subtopology } (\text{nsphere } n) \{x. x \cdot k \geq 0\}) \{x. x \cdot k = 0\} (\text{nsphere } n) \{x. x \cdot k \leq 0\} \text{ id})$

```

    ∈ iso (relative_homology_group p (subtopology (nsphere n) {x. x k ≥ 0}) {x. x
k = 0})
    (relative_homology_group p (nsphere n) {x. x k ≤ 0}) (is ?h ∈ iso ?G ?H)
proof –
    have topspace (nsphere n) ∩ {x. x k < - 1 / 2} ⊆ {x ∈ topspace (nsphere n).
x k ∈ {y. y ≤ - 1 / 2}}
    by force
    moreover have closedin (nsphere n) {x ∈ topspace (nsphere n). x k ∈ {y. y ≤
- 1 / 2}}
    apply (rule closedin_continuous_map_preimage [OF continuous_map_nsphere_projection])
    using closed_Collect_le [of id λx::real. -1/2] apply simp
    done
    ultimately have nsphere n closure_of {x. x k < -1/2} ⊆ {x ∈ topspace
(nsphere n). x k ∈ {y. y ≤ -1/2}}
    by (metis (no_types, lifting) closure_of_eq closure_of_mono closure_of_restrict)
    also have ... ⊆ {x ∈ topspace (nsphere n). x k ∈ {y. y < 0}}
    by force
    also have ... ⊆ nsphere n interior_of {x. x k ≤ 0}
    proof (rule interior_of_maximal)
    show {x ∈ topspace (nsphere n). x k ∈ {y. y < 0}} ⊆ {x. x k ≤ 0}
    by force
    show openin (nsphere n) {x ∈ topspace (nsphere n). x k ∈ {y. y < 0}}
    apply (rule openin_continuous_map_preimage [OF continuous_map_nsphere_projection])
    using open_Collect_less [of id λx::real. 0] apply simp
    done
    qed
    finally have nn: nsphere n closure_of {x. x k < -1/2} ⊆ nsphere n interior_of
{x. x k ≤ 0} .
    have [simp]: {x::nat⇒real. x k ≤ 0} - {x. x k < - (1/2)} = {x. -1/2 ≤ x k
∧ x k ≤ 0}
    UNIV - {x::nat⇒real. x k < a} = {x. a ≤ x k} for a
    by auto
    let ?T01 = top_of_set {0..1::real}
    let ?X12 = subtopology (nsphere n) {x. -1/2 ≤ x k}
    have 1: hom_induced p ?X12 {x. -1/2 ≤ x k ∧ x k ≤ 0} (nsphere n) {x. x k
≤ 0} id
    ∈ iso (relative_homology_group p ?X12 {x. -1/2 ≤ x k ∧ x k ≤ 0})
    ?H
    using homology_excision_axiom [OF nn subset_UNIV, of p] by simp
    define h where h ≡ λ(T,x). let y = max (x k) (-T) in
    (λi. if i = k then y else sqrt(1 - y ^ 2) / sqrt(1 - x k ^
2) * x i)
    have h: h(T,x) = x if 0 ≤ T T ≤ 1 (∑ i≤n. (x i)^2) = 1 and 0: ∀ i>n. x i =
0 -T ≤ x k for T x
    using that by (force simp: nsphere h_def Let_def max_def intro!: topspace_nsphere_1_eq_0)
    have continuous_map (prod_topology ?T01 ?X12) euclideanreal (λx. h x i) for
i
    proof –
    show ?thesis

```

```

proof (rule continuous_map_eq)
  show continuous_map (prod_topology ?T01 ?X12)
    euclideanreal ( $\lambda(T, x). \text{if } 0 \leq x \text{ k then } x \text{ i else } h(T, x) \text{ i}$ )
  unfolding case_prod_unfold
  proof (rule continuous_map_cases_le)
    show continuous_map (prod_topology ?T01 ?X12) euclideanreal ( $\lambda x. \text{snd } x$ 
k)
      apply (subst continuous_map_of_snd [unfolded o_def])
      by (simp add: continuous_map_from_subtopology continuous_map_nsphere_projection)
    next
      show continuous_map (subtopology (prod_topology ?T01 ?X12) {p ∈ topspace
(prod_topology ?T01 ?X12).  $0 \leq \text{snd } p \text{ k}$ })
        euclideanreal ( $\lambda x. \text{snd } x \text{ i}$ )
        apply (rule continuous_map_from_subtopology)
        apply (subst continuous_map_of_snd [unfolded o_def])
        by (simp add: continuous_map_from_subtopology continuous_map_nsphere_projection)
      next
        note fst = continuous_map_into_fulltopology [OF continuous_map_subtopology_fst]
        have snd: continuous_map (subtopology (prod_topology ?T01 (subtopology
(nsphere n) T)) S) euclideanreal ( $\lambda x. \text{snd } x \text{ k}$ ) for k S T
          apply (simp add: nsphere)
          apply (rule continuous_map_from_subtopology)
          apply (subst continuous_map_of_snd [unfolded o_def])
          using continuous_map_from_subtopology continuous_map_nsphere_projection
        nsphere by fastforce
        show continuous_map (subtopology (prod_topology ?T01 ?X12) {p ∈ topspace
(prod_topology ?T01 ?X12).  $\text{snd } p \text{ k} \leq 0$ })
          euclideanreal ( $\lambda x. h(\text{fst } x, \text{snd } x) \text{ i}$ )
          apply (simp add: h_def case_prod_unfold Let_def)
          apply (intro conjI impI fst snd continuous_intros)
          apply (auto simp: nsphere power2_eq_1_iff)
          done
        qed (auto simp: nsphere h)
      qed (auto simp: nsphere h)
    qed
  moreover
    have h ' ( $\{0..1\} \times (\text{topspace } (\text{nsphere } n) \cap \{x. -(1/2) \leq x \text{ k}\})$ )
       $\subseteq \{x. (\sum_{i \leq n}. (x \text{ i})^2) = 1 \wedge (\forall i > n. x \text{ i} = 0)\}$ 
    proof -
      have ( $\sum_{i \leq n}. (h(T, x) \text{ i})^2 = 1$ )
        if x:  $x \in \text{topspace } (\text{nsphere } n)$  and xk:  $-(1/2) \leq x \text{ k}$  and T:  $0 \leq T \text{ T} \leq 1$ 
      for T x
      proof (cases  $-T \leq x \text{ k}$ )
        case True
          then show ?thesis
            using that by (auto simp: nsphere h)
        next
          case False
            with  $x \text{ } 0 \leq T \text{ } have$   $k \leq n$ 

```



```

    apply (simp add: nsphere)
    by (metis neg_le_0_iff_le not_le)
  have  $1 - (x\ k)^2 \geq 0$ 
    using topspace_nsphere_1 x by auto
  with False T <k ≤ n>
  have  $(\sum_{i \leq n. (h\ (T,x)\ i)^2}) = T^2 + (1 - T^2) * (\sum_{i \in \{..n\} - \{k\}. (x\ i)^2} / (1 - (x\ k)^2))$ 
    unfolding h_def Let_def max_def
    by (simp add: not_le square_le_1 power_mult_distrib power_divide
if_distrib [of  $\lambda x. x^2$ ]
sum.delta_remove sum_distrib_left)
  also have ... = 1
    using x False xk <0 ≤ T>
    by (simp add: nsphere sum_diff not_le <k ≤ n> power2_eq_1_iff flip:
sum_divide_distrib)
  finally show ?thesis .
qed
moreover
have h (T,x) i = 0
  if  $x \in \text{topspace } (\text{nsphere } n) - (1/2) \leq x\ k$  and  $n < i\ 0 \leq T\ T \leq 1$ 
  for T x i
proof (cases  $-T \leq x\ k$ )
case False
then show ?thesis
  using that by (auto simp: nsphere h_def Let_def not_le max_def)
qed (use that in <auto simp: nsphere h>)
ultimately show ?thesis
  by auto
qed
ultimately
have cmh: continuous_map (prod_topology ?T01 ?X12) (nsphere n) h
  by (subst (2) nsphere) (simp add: continuous_map_in_subtopology continuous_map_componentwise_UNIV)
  have hom_induced p (subtopology (nsphere n) {x. 0 ≤ x k})
    (topspace (subtopology (nsphere n) {x. 0 ≤ x k}) ∩ {x. x k = 0}) ?X12
    (topspace ?X12 ∩ {x. - 1/2 ≤ x k ∧ x k ≤ 0}) id
    ∈ iso (relative_homology_group p (subtopology (nsphere n) {x. 0 ≤ x k})
    (topspace (subtopology (nsphere n) {x. 0 ≤ x k}) ∩ {x. x k = 0}))
    (relative_homology_group p ?X12 (topspace ?X12 ∩ {x. - 1/2 ≤ x k ∧ x k ≤ 0}))
  proof (rule deformation_retract_relative_homology_group_isomorphism_id)
    show retraction_maps ?X12 (subtopology (nsphere n) {x. 0 ≤ x k}) (h ∘ (λx. (0,x))) id
      unfolding retraction_maps_def
    proof (intro conjI ballI)
      show continuous_map ?X12 (subtopology (nsphere n) {x. 0 ≤ x k}) (h ∘ Pair 0)
        apply (simp add: continuous_map_in_subtopology)

```

```

apply (intro conjI continuous_map_compose [OF cmh] continuous_intros)
apply (auto simp: h_def Let_def)
done
show continuous_map (subtopology (nsphere n) {x. 0 ≤ x k}) ?X12 id
by (simp add: continuous_map_in_subtopology) (auto simp: nsphere)
qed (simp add: nsphere h)
next
have h0:  $\bigwedge x. \llbracket x \in \text{topspace } (\text{nsphere } n); - (1/2) \leq x k; x k \leq 0 \rrbracket \implies h$ 
  (0, x) k = 0
by (simp add: h_def Let_def)
show (h ∘ (λx. (0, x))) ‘ (topspace ?X12 ∩ {x. - 1 / 2 ≤ x k ∧ x k ≤ 0})
  ⊆ topspace (subtopology (nsphere n) {x. 0 ≤ x k}) ∩ {x. x k = 0}
apply (auto simp: h0)
apply (rule subsetD [OF continuous_map_image_subset_topspace [OF cmh]])
apply (force simp: nsphere)
done
have hin:  $\bigwedge t x. \llbracket x \in \text{topspace } (\text{nsphere } n); - (1/2) \leq x k; 0 \leq t; t \leq 1 \rrbracket \implies$ 
  h (t, x) ∈ topspace (nsphere n)
apply (rule subsetD [OF continuous_map_image_subset_topspace [OF cmh]])
apply (force simp: nsphere)
done
have h1:  $\bigwedge x. \llbracket x \in \text{topspace } (\text{nsphere } n); - (1/2) \leq x k \rrbracket \implies h (1, x) = x$ 
by (simp add: h nsphere)
have continuous_map (prod_topology ?T01 ?X12) (nsphere n) h
using cmh by force
then show homotopic_with
  (λh. h ‘ (topspace ?X12 ∩ {x. - 1 / 2 ≤ x k ∧ x k ≤ 0}) ⊆ topspace
  ?X12 ∩ {x. - 1 / 2 ≤ x k ∧ x k ≤ 0})
  ?X12 ?X12 (h ∘ (λx. (0, x))) id
apply (subst homotopic_with, force)
apply (rule_tac x=h in exI)
apply (auto simp: hin h1 continuous_map_in_subtopology)
apply (auto simp: h_def Let_def max_def)
done
qed auto
then have 2: hom_induced p (subtopology (nsphere n) {x. 0 ≤ x k}) {x. x k =
  0}
  ?X12 {x. - 1/2 ≤ x k ∧ x k ≤ 0} id
  ∈ Group.iso
  (relative_homology_group p (subtopology (nsphere n) {x. 0 ≤ x k})
  {x. x k = 0})
  (relative_homology_group p ?X12 {x. - 1/2 ≤ x k ∧ x k ≤ 0})
by (metis hom_induced_restrict relative_homology_group_restrict topspace_subtopology)
show ?thesis
using iso_set_trans [OF 2 1]
by (simp add: subset_iff continuous_map_in_subtopology flip: hom_induced_compose)
qed

```

corollary *iso_upper_hemisphere_reduced_homology_group:*

(*hom_boundary* (1 + p) (subtopology (nsphere (Suc n)) {x. x(Suc n) ≥ 0}) {x. x(Suc n) = 0})
 ∈ iso (relative_homology_group (1 + p) (subtopology (nsphere (Suc n)) {x. x(Suc n) ≥ 0}) {x. x(Suc n) = 0})
 (reduced_homology_group p (nsphere n))

proof –

have {x. 0 ≤ x (Suc n)} ∩ {x. x (Suc n) = 0} = {x. x (Suc n) = (0::real)}
by *auto*
then have n: nsphere n = subtopology (subtopology (nsphere (Suc n)) {x. x(Suc n) ≥ 0}) {x. x(Suc n) = 0}
by (simp add: subtopology_nsphere_equator subtopology_subtopology)
have ne: (λi. if i = n then 1 else 0) ∈ topspace (subtopology (nsphere (Suc n)) {x. 0 ≤ x (Suc n)}) ∩ {x. x (Suc n) = 0}
by (simp add: nsphere if_distrib [of λx. x ^ 2] cong: if_cong)
show ?thesis
unfolding n
apply (rule iso_relative_homology_of_contractible [where p = 1 + p, simplified])
using contractible_space_upper_hemisphere ne **apply** blast+
done
qed

corollary *iso_reduced_homology_group_upper_hemisphere:*

assumes k ≤ n
shows hom_induced p (nsphere n) {} (nsphere n) {x. x k ≥ 0} id
 ∈ iso (reduced_homology_group p (nsphere n)) (relative_homology_group p (nsphere n) {x. x k ≥ 0})
proof (rule iso_reduced_homology_by_contractible [OF contractible_space_upper_hemisphere [OF assms]])
have (λi. if i = k then 1 else 0) ∈ topspace (nsphere n) ∩ {x. 0 ≤ x k}
using assms **by** (simp add: nsphere if_distrib [of λx. x ^ 2] cong: if_cong)
then show topspace (nsphere n) ∩ {x. 0 ≤ x k} ≠ {}
by blast
qed

lemma *iso_relative_homology_group_lower_hemisphere:*

hom_induced p (subtopology (nsphere n) {x. x k ≤ 0}) {x. x k = 0} (nsphere n) {x. x k ≥ 0} id
 ∈ iso (relative_homology_group p (subtopology (nsphere n) {x. x k ≤ 0}) {x. x k = 0})
 (relative_homology_group p (nsphere n) {x. x k ≥ 0}) (is ?k ∈ iso ?G ?H)

proof –

define r **where** r ≡ λx i. if i = k then -x i else (x i::real)
then have [simp]: r ∘ r = id
by force
have cmr: continuous_map (subtopology (nsphere n) S) (nsphere n) r **for** S
using continuous_map_nsphere_reflection [of n k]

```

    by (simp add: continuous_map_from_subtopology r_def)
  let ?f = hom_induced p (subtopology (nsphere n) {x. x k ≤ 0}) {x. x k = 0}
    (subtopology (nsphere n) {x. x k ≥ 0}) {x. x k = 0} r
  let ?g = hom_induced p (subtopology (nsphere n) {x. x k ≥ 0}) {x. x k = 0}
    (nsphere n) {x. x k ≤ 0} id
  let ?h = hom_induced p (nsphere n) {x. x k ≤ 0} (nsphere n) {x. x k ≥ 0} r
  obtain f h where
    f: f ∈ iso ?G (relative_homology_group p (subtopology (nsphere n) {x. x k
    ≥ 0}) {x. x k = 0})
    and h: h ∈ iso (relative_homology_group p (nsphere n) {x. x k ≤ 0}) ?H
    and eq: h ∘ ?g ∘ f = ?k
  proof
    have hmr: homeomorphic_map (nsphere n) (nsphere n) r
      unfolding homeomorphic_map_maps
      by (metis ⟨r ∘ r = id⟩ cmr homeomorphic_maps_involution pointfree_idE
    subtopology_topspace)
    then have hmrs: homeomorphic_map (subtopology (nsphere n) {x. x k ≤ 0})
    (subtopology (nsphere n) {x. x k ≥ 0}) r
      by (simp add: homeomorphic_map_subtopologies_alt r_def)
    have rimeq: r ‘ (topspace (subtopology (nsphere n) {x. x k ≤ 0}) ∩ {x. x k =
    0})
      = topspace (subtopology (nsphere n) {x. 0 ≤ x k}) ∩ {x. x k = 0}
    using continuous_map_eq_topcontinuous_at continuous_map_nsphere_reflection
    topcontinuous_at_atin
      by (fastforce simp: r_def Pi_iff)
    show ?f ∈ iso ?G (relative_homology_group p (subtopology (nsphere n) {x. x
    k ≥ 0}) {x. x k = 0})
      using homeomorphic_map_relative_homology_iso [OF hmrs Int_lower1
    rimeq]
      by (metis hom_induced_restrict relative_homology_group_restrict)
    have rimeq: r ‘ (topspace (nsphere n) ∩ {x. x k ≤ 0}) = topspace (nsphere n)
    ∩ {x. 0 ≤ x k}
      by (metis hmrs homeomorphic_imp_surjective_map topspace_subtopology)
    show ?h ∈ Group.iso (relative_homology_group p (nsphere n) {x. x k ≤ 0})
    ?H
      using homeomorphic_map_relative_homology_iso [OF hmr Int_lower1 rimeq]
  by simp
  have [simp]: ∧x. x k = 0 ⇒ r x k = 0
    by (auto simp: r_def)
  have ?h ∘ ?g ∘ ?f
    = hom_induced p (subtopology (nsphere n) {x. 0 ≤ x k}) {x. x k = 0}
    (nsphere n) {x. 0 ≤ x k} r ∘
      hom_induced p (subtopology (nsphere n) {x. x k ≤ 0}) {x. x k = 0}
    (subtopology (nsphere n) {x. 0 ≤ x k}) {x. x k = 0} r
    apply (subst hom_induced_compose [symmetric])
    using continuous_map_nsphere_reflection apply (force simp: r_def) +
    done
  also have ... = ?k
    apply (subst hom_induced_compose [symmetric])

```

```

    apply (simp_all add: image_subset_iff cmr)
  using hmrs homeomorphic_imp_continuous_map apply blast
done
finally show ?h ◦ ?g ◦ ?f = ?k .
qed
with iso_relative_homology_group_upper_hemisphere [of p n k]
have h ◦ hom_induced p (subtopology (nsphere n) {f. 0 ≤ f k}) {f. f k = 0}
(nsphere n) {f. f k ≤ 0} id ◦ f
∈ Group.iso ?G (relative_homology_group p (nsphere n) {f. 0 ≤ f k})
using f h iso_set_trans by blast
then show ?thesis
by (simp add: eq)
qed

```

lemma *iso_lower_hemisphere_reduced_homology_group*:

```

  hom_boundary (1 + p) (subtopology (nsphere (Suc n)) {x. x(Suc n) ≤ 0}) {x.
x(Suc n) = 0}
  ∈ iso (relative_homology_group (1 + p) (subtopology (nsphere (Suc n)) {x. x(Suc
n) ≤ 0}))
    {x. x(Suc n) = 0})
    (reduced_homology_group p (nsphere n))

```

proof –

```

  have {x. (∑ i≤n. (x i)2) = 1 ∧ (∀ i>n. x i = 0)} =
    ({x. (∑ i≤n. (x i)2) + (x (Suc n))2 = 1 ∧ (∀ i>Suc n. x i = 0)} ∩ {x. x
(Suc n) ≤ 0}) ∩
    {x. x (Suc n) = (0::real)})
  by (force simp: dest: Suc_lessI)
  then have n: nsphere n = subtopology (subtopology (nsphere (Suc n)) {x. x(Suc
n) ≤ 0}) {x. x(Suc n) = 0}
  by (simp add: nsphere_subtopology_subtopology)
  have ne: (λi. if i = n then 1 else 0) ∈ topspace (subtopology (nsphere (Suc n))
{x. x (Suc n) ≤ 0}) ∩ {x. x (Suc n) = 0}
  by (simp add: nsphere_if_distrib [of λx. x ^ 2] cong: if_cong)
  show ?thesis
  unfolding n
  apply (rule iso_relative_homology_of_contractible [where p = 1 + p, sim-
plified])
  using contractible_space_lower_hemisphere ne apply blast+
done
qed

```

lemma *isomorphism_sym*:

```

[[f ∈ iso G1 G2; ∧ x. x ∈ carrier G1 ⇒ r'(f x) = f(r x);
  ∧ x. x ∈ carrier G1 ⇒ r x ∈ carrier G1; group G1; group G2]]
⇒ ∃ f ∈ iso G2 G1. ∀ x ∈ carrier G2. r(f x) = f(r' x)
apply (clarsimp simp add: group.iso_iff_group_isomorphisms Bex_def)
by (metis (full_types) group_isomorphisms_def group_isomorphisms_sym hom_in_carrier)

```

lemma *isomorphism_trans*:

$\llbracket \exists f \in \text{iso } G1 \ G2. \forall x \in \text{carrier } G1. r2(f\ x) = f(r1\ x); \exists f \in \text{iso } G2 \ G3. \forall x \in \text{carrier } G2. r3(f\ x) = f(r2\ x) \rrbracket$
 $\implies \exists f \in \text{iso } G1 \ G3. \forall x \in \text{carrier } G1. r3(f\ x) = f(r1\ x)$
apply *clarify*
apply (*rename_tac* *g f*)
apply (*rule_tac* $x=f \circ g$ **in** *boxI*)
apply (*metis iso_iff_comp_apply_hom_in_carrier*)
using *iso_set_trans* **by** *blast*

lemma *reduced_homology_group_nsphere_step*:

$\exists f \in \text{reduced_homology_group } p \ (\text{nsphere } n)$
 $(\text{reduced_homology_group } (1 + p) \ (\text{nsphere } (\text{Suc } n)))$.
 $\forall c \in \text{carrier}(\text{reduced_homology_group } p \ (\text{nsphere } n))$.
 $\text{hom_induced } (1 + p) \ (\text{nsphere } (\text{Suc } n)) \ \{\} \ (\text{nsphere } (\text{Suc } n)) \ \{\}$
 $(\lambda x\ i. \text{ if } i = 0 \text{ then } -x\ i \text{ else } x\ i) \ (f\ c)$
 $= f \ (\text{hom_induced } p \ (\text{nsphere } n) \ \{\} \ (\text{nsphere } n) \ \{\} \ (\lambda x\ i. \text{ if } i = 0 \text{ then } -x\ i \text{ else } x\ i) \ c)$

proof –

define *r* **where** $r \equiv \lambda x::\text{nat} \Rightarrow \text{real}. \lambda i. \text{ if } i = 0 \text{ then } -x\ i \text{ else } x\ i$
have *cmr*: *continuous_map* (*nsphere* *n*) (*nsphere* *n*) *r* **for** *n*
unfolding *r_def* **by** (*rule continuous_map_nsphere_reflection*)
have *rsub*: $r \restriction \{x. 0 \leq x \ (\text{Suc } n)\} \subseteq \{x. 0 \leq x \ (\text{Suc } n)\} \ r \restriction \{x. x \ (\text{Suc } n) \leq 0\}$
 $\subseteq \{x. x \ (\text{Suc } n) \leq 0\} \ r \restriction \{x. x \ (\text{Suc } n) = 0\} \subseteq \{x. x \ (\text{Suc } n) = 0\}$
by (*force simp: r_def*)
let *?sub* = *subtopology* (*nsphere* (*Suc* *n*)) $\{x. x \ (\text{Suc } n) \geq 0\}$
let *?G2* = *relative_homology_group* ($1 + p$) *?sub* $\{x. x \ (\text{Suc } n) = 0\}$
let *?r2* = *hom_induced* ($1 + p$) *?sub* $\{x. x \ (\text{Suc } n) = 0\}$ *?sub* $\{x. x \ (\text{Suc } n) = 0\}$ *r*
let *?j* = $\lambda p\ n. \text{ hom_induced } p \ (\text{nsphere } n) \ \{\} \ (\text{nsphere } n) \ \{\} \ r$
show *?thesis*
unfolding *r_def* [*symmetric*]
proof (*rule isomorphism_trans*)
let *?f* = *hom_boundary* ($1 + p$) *?sub* $\{x. x \ (\text{Suc } n) = 0\}$
show $\exists f \in \text{Group.iso } (\text{reduced_homology_group } p \ (\text{nsphere } n)) \ ?G2$.
 $\forall c \in \text{carrier } (\text{reduced_homology_group } p \ (\text{nsphere } n)). \ ?r2 \ (f\ c) = f \ (?j\ p$

n c)

proof (*rule isomorphism_sym*)
show $\exists f \in \text{Group.iso } ?G2 \ (\text{reduced_homology_group } p \ (\text{nsphere } n))$
using *iso_upper_hemisphere_reduced_homology_group*
by (*metis add commute*)
next
fix *c*
assume $c \in \text{carrier } ?G2$
have *cmrs*: *continuous_map* *?sub* *?sub* *r*
by (*metis (mono_tags, lifting) IntE cmr continuous_map_from_subtopology continuous_map_in_subtopology image_subset_iff rsub(1) topspace_subtopology*)
have *hom_induced* *p* (*nsphere* *n*) $\{\} \ (\text{nsphere } n) \ \{\} \ r \circ \text{hom_boundary } (1 + p) \ ?sub \ \{x. x \ (\text{Suc } n) = 0\}$

```

    = hom_boundary (1 + p) ?sub {x. x (Suc n) = 0} ∘
      hom_induced (1 + p) ?sub {x. x (Suc n) = 0} ?sub {x. x (Suc n) = 0}
  r
    using naturality_hom_induced [OF cmrs rsub(3), symmetric, of 1+p,
simplified]
    by (simp add: subtopology_subtopology subtopology_nsphere_equator flip:
Collect_conj_eq cong: rev_conj_cong)
    then show ?j p n (?f c) = ?f (hom_induced (1 + p) ?sub {x. x (Suc n) =
0} ?sub {x. x (Suc n) = 0} r c)
      by (metis comp_def)
    next
      fix c
      assume c ∈ carrier ?G2
      show hom_induced (1 + p) ?sub {x. x (Suc n) = 0} ?sub {x. x (Suc n) =
0} r c ∈ carrier ?G2
        using hom_induced_carrier by blast
      qed auto
    next
      let ?H2 = relative_homology_group (1 + p) (nsphere (Suc n)) {x. x (Suc n)
≤ 0}
      let ?s2 = hom_induced (1 + p) (nsphere (Suc n)) {x. x (Suc n) ≤ 0} (nsphere
(Suc n)) {x. x (Suc n) ≤ 0} r
      show ∃ f ∈ Group.iso ?G2 (reduced_homology_group (1 + p) (nsphere (Suc
n))). ∀ c ∈ carrier ?G2. ?j (1 + p) (Suc n) (f c)
        = f (?r2 c)
      proof (rule isomorphism_trans)
        show ∃ f ∈ Group.iso ?G2 ?H2.
          ∀ c ∈ carrier ?G2.
            ?s2 (f c) = f (hom_induced (1 + p) ?sub {x. x (Suc n) = 0} ?sub
{x. x (Suc n) = 0} r c)
        proof (intro ballI bexI)
          fix c
          assume c ∈ carrier (relative_homology_group (1 + p) ?sub {x. x (Suc n)
= 0})
          show ?s2 (hom_induced (1 + p) ?sub {x. x (Suc n) = 0} (nsphere (Suc
n)) {x. x (Suc n) ≤ 0} id c)
            = hom_induced (1 + p) ?sub {x. x (Suc n) = 0} (nsphere (Suc n)) {x.
x (Suc n) ≤ 0} id (?r2 c)
          apply (simp add: rsub hom_induced_compose' Collect_mono_iff cmr)
          apply (subst hom_induced_compose')
          apply (simp_all add: continuous_map_in_subtopology continu-
ous_map_from_subtopology [OF cmr] rsub)
          apply (auto simp: r_def)
        done
      proof (simp add: iso_relative_homology_group_upper_hemisphere)
    next
      let ?h = hom_induced (1 + p) (nsphere (Suc n)) {} (nsphere (Suc n)) {x.
x (Suc n) ≤ 0} id
      show ∃ f ∈ Group.iso ?H2 (reduced_homology_group (1 + p) (nsphere (Suc

```

```

n))).
  ∀ c ∈ carrier ?H2. ?j (1 + p) (Suc n) (f c) = f (?s2 c)
proof (rule isomorphism_sym)
  show ?h ∈ Group.iso (reduced_homology_group (1 + p) (nsphere (Suc n)))
    (relative_homology_group (1 + p) (nsphere (Suc n)) {x. x (Suc n) ≤
0})
    using iso_reduced_homology_group_lower_hemisphere by blast
next
  fix c
  assume c ∈ carrier (reduced_homology_group (1 + p) (nsphere (Suc n)))
  show ?s2 (?h c) = ?h (?j (1 + p) (Suc n) c)
    by (simp add: hom_induced_compose' cmr rsub)
next
  fix c
  assume c ∈ carrier (reduced_homology_group (1 + p) (nsphere (Suc n)))
  then show hom_induced (1 + p) (nsphere (Suc n)) {} (nsphere (Suc n))
{} r c
    ∈ carrier (reduced_homology_group (1 + p) (nsphere (Suc n)))
    by (simp add: hom_induced_reduced)
qed auto
qed
qed
qed

```

```

lemma reduced_homology_group_nsphere_aux:
  if p = int n then reduced_homology_group n (nsphere n) ≅ integer_group
    else trivial_group (reduced_homology_group p (nsphere n))
proof (induction n arbitrary: p)
  case 0
  let ?a = λi::nat. if i = 0 then 1 else (0::real)
  let ?b = λi::nat. if i = 0 then -1 else (0::real)
  have st: subtopology (powertop_real UNIV) {?a, ?b} = nsphere 0
  proof -
    have {?a, ?b} = {x. (x 0)2 = 1 ∧ (∀ i > 0. x i = 0)}
    using power2_eq_iff by fastforce
    then show ?thesis
    by (simp add: nsphere)
  qed
  have *: reduced_homology_group p (subtopology (powertop_real UNIV) {?a,
?b}) ≅
    homology_group p (subtopology (powertop_real UNIV) {?a})
  apply (rule reduced_homology_group_pair)
  apply (simp_all add: fun_eq_iff)
  apply (simp add: open_fun_def separation_t1 t1_space_def)
  done
  have reduced_homology_group 0 (nsphere 0) ≅ integer_group if p=0
  proof -
    have reduced_homology_group 0 (nsphere 0) ≅ homology_group 0 (top_of_set

```



```

{ ?a } if p=0
  by (metis * euclidean_product_topology st that)
  also have ...  $\cong$  integer_group
  by (simp add: homology_coefficients)
  finally show ?thesis
  using that by blast
qed
moreover have trivial_group (reduced_homology_group p (nsphere 0)) if p $\neq$ 0
  using * that homology_dimension_axiom [of subtopology (powertop_real UNIV)
{ ?a } ?a p]
  using isomorphic_group_triviality st by force
ultimately show ?case
  by auto
next
case (Suc n)
  have eq: reduced_homology_group (int n) (nsphere n)  $\cong$  integer_group if p-1
= n
  by (simp add: Suc.IH)
  have neg: trivial_group (reduced_homology_group (p-1) (nsphere n)) if p-1  $\neq$ 
n
  by (simp add: Suc.IH that)
  have iso: reduced_homology_group p (nsphere (Suc n))  $\cong$  reduced_homology_group
(p-1) (nsphere n)
  using reduced_homology_group_nsphere_step [of p-1 n] group.iso_sym [OF
_ is_isoI] group_reduced_homology_group
  by fastforce
  then show ?case
  using eq iso_trans iso isomorphic_group_triviality neg
  by (metis (no_types, opaque_lifting) add.commute add_left_cancel diff_add_cancel
group_reduced_homology_group of_nat_Suc)
qed

```

lemma *reduced_homology_group_nsphere:*
 $\text{reduced_homology_group } n \text{ (nsphere } n) \cong \text{integer_group}$
 $p \neq n \implies \text{trivial_group}(\text{reduced_homology_group } p \text{ (nsphere } n))$
using *reduced_homology_group_nsphere_aux* **by** *auto*

lemma *cyclic_reduced_homology_group_nsphere:*
 $\text{cyclic_group}(\text{reduced_homology_group } p \text{ (nsphere } n))$
by (metis *reduced_homology_group_nsphere trivial_imp_cyclic_group cyclic_integer_group*
group_integer_group group_reduced_homology_group isomorphic_group_cyclicity)

lemma *trivial_reduced_homology_group_nsphere:*
 $\text{trivial_group}(\text{reduced_homology_group } p \text{ (nsphere } n)) \longleftrightarrow (p \neq n)$
using *group_integer_group isomorphic_group_triviality nontrivial_integer_group*
reduced_homology_group_nsphere(1) reduced_homology_group_nsphere(2) trivial_group_def **by** *blast*

```

lemma non_contractible_space_nsphere:  $\neg$  (contractible_space (nsphere n))
proof (clarsimp simp add: contractible_eq_homotopy_equivalent_singleton_subtopology)
  fix a :: nat  $\Rightarrow$  real
  assume a: a  $\in$  topspace (nsphere n)
    and he: nsphere n homotopy_equivalent_space subtopology (nsphere n) {a}
  have trivial_group (reduced_homology_group (int n) (subtopology (nsphere n) {a}))
  by (simp add: a_homology_dimension_reduced [where a=a])
  then show False
  using isomorphic_group_triviality [OF homotopy_equivalent_space_imp_isomorphic_reduced_homology_group]
  by (simp add: trivial_reduced_homology_group_nsphere)
qed

```

0.3.4 Brouwer degree of a Map

```

definition Brouwer_degree2 :: nat  $\Rightarrow$  ((nat  $\Rightarrow$  real)  $\Rightarrow$  nat  $\Rightarrow$  real)  $\Rightarrow$  int
  where
    Brouwer_degree2 p f  $\equiv$ 
      @d::int.  $\forall x \in \text{carrier}(\text{reduced\_homology\_group } p \text{ (nsphere } p))$ .
        hom_induced p (nsphere p) {} (nsphere p) {} f x = pow (reduced_homology_group
p (nsphere p)) x d

```

```

lemma Brouwer_degree2_eq:
  ( $\bigwedge x. x \in \text{topspace}(\text{nsphere } p) \implies f x = g x \implies \text{Brouwer\_degree2 } p f =$ 
  Brouwer_degree2 p g
  unfolding Brouwer_degree2_def Ball_def
  apply (intro Eps_cong all_cong)
  by (metis (mono_tags, lifting) hom_induced_eq)

```

```

lemma Brouwer_degree2:
  assumes x  $\in$  carrier (reduced_homology_group p (nsphere p))
  shows hom_induced p (nsphere p) {} (nsphere p) {} f x
    = pow (reduced_homology_group p (nsphere p)) x (Brouwer_degree2 p f)
    (is ?h x = pow ?G x _)
proof (cases continuous_map (nsphere p) (nsphere p) f)
  case True
    interpret group ?G
    by simp
    interpret group_hom ?G ?G ?h
    using hom_induced_reduced_hom_group_hom_axioms_def group_hom_def
  is_group by blast
  obtain a where a: a  $\in$  carrier ?G
    and aeq: subgroup_generated ?G {a} = ?G
    using cyclic_reduced_homology_group_nsphere [of p p] by (auto simp: cyclic_group_def)
  then have carra: carrier (subgroup_generated ?G {a}) = range ( $\lambda n::\text{int. pow}$ 
  ?G a n)
    using carrier_subgroup_generated_by_singleton by blast
  moreover have ?h a  $\in$  carrier (subgroup_generated ?G {a})

```

```

    by (simp add: a aeq hom_induced_reduced)
  ultimately obtain d::int where d: ?h a = pow ?G a d
    by auto
  have *: hom_induced (int p) (nsphere p) {} (nsphere p) {} f x = x [^]?G d
    if x: x ∈ carrier ?G for x
  proof -
    obtain n::int where xeq: x = pow ?G a n
      using carra x aeq by auto
    show ?thesis
      by (simp add: xeq a d hom_int_pow int_pow_pow mult.commute)
  qed
  show ?thesis
    unfolding Brouwer_degree2_def
    apply (rule someI2 [where a=d])
    using assms * apply blast+
  done
next
case False
show ?thesis
  unfolding Brouwer_degree2_def
  by (rule someI2 [where a=0]) (simp_all add: hom_induced_default False
one_reduced_homology_group assms)
qed

```

lemma *Brouwer_degree2_iff*:

```

  assumes f: continuous_map (nsphere p) (nsphere p) f
    and x: x ∈ carrier(reduced_homology_group p (nsphere p))
  shows (hom_induced (int p) (nsphere p) {} (nsphere p) {} f x =
    x [^]_reduced_homology_group (int p) (nsphere p) d)
    ↔ (x = 1_reduced_homology_group (int p) (nsphere p) ∨ Brouwer_degree2 p f
= d)
    (is (?h x = x [^]?G d) ↔ _)
  proof -
    interpret group ?G
    by simp
    obtain a where a: a ∈ carrier ?G
      and aeq: subgroup_generated ?G {a} = ?G
    using cyclic_reduced_homology_group_nsphere [of p p] by (auto simp: cyclic_group_def)
    then obtain i::int where i: x = (a [^]?G i)
      using carrier_subgroup_generated_by_singleton x by fastforce
    then have a [^]?G i ∈ carrier ?G
      using x by blast
    have [simp]: ord a = 0
      by (simp add: a aeq iso_finite [OF reduced_homology_group_nsphere(1)] flip:
infinite_cyclic_subgroup_order)
    show ?thesis
      by (auto simp: Brouwer_degree2 int_pow_eq_id x i a int_pow_pow int_pow_eq)
  qed

```

qed

lemma *Brouwer_degree2_unique*:
assumes *f*: *continuous_map* (*nsphere* *p*) (*nsphere* *p*) *f*
and *hi*: $\bigwedge x. x \in \text{carrier}(\text{reduced_homology_group } p \text{ } (\text{nsphere } p))$
 $\implies \text{hom_induced } p \text{ } (\text{nsphere } p) \text{ } \{\} \text{ } (\text{nsphere } p) \text{ } \{\} f x = \text{pow}$
 $(\text{reduced_homology_group } p \text{ } (\text{nsphere } p)) x d$
(is $\bigwedge x. x \in \text{carrier } ?G \implies ?h x = _)$
shows *Brouwer_degree2* *p* *f* = *d*
proof –
obtain *a* **where** *a*: *a* $\in \text{carrier } ?G$
and *aeq*: *subgroup_generated* $?G \{a\} = ?G$
using *cyclic_reduced_homology_group_nsphere* [of *p* *p*] **by** (*auto simp: cyclic_group_def*)
show *?thesis*
using *hi* [OF *a*]
apply (*simp add: Brouwer_degree2* *a*)
by (*metis Brouwer_degree2_iff a aeq f group.trivial_group_subgroup_generated*
group_reduced_homology_group subsetI trivial_reduced_homology_group_nsphere)
qed

lemma *Brouwer_degree2_unique_generator*:
assumes *f*: *continuous_map* (*nsphere* *p*) (*nsphere* *p*) *f*
and *eq*: *subgroup_generated* (*reduced_homology_group* *p* (*nsphere* *p*)) $\{a\}$
 $= \text{reduced_homology_group } p \text{ } (\text{nsphere } p)$
and *hi*: *hom_induced* *p* (*nsphere* *p*) $\{\}$ (*nsphere* *p*) $\{\}$ *f* *a* = *pow* (*reduced_homology_group*
p (*nsphere* *p*)) *a* *d*
(is $?h a = \text{pow } ?G a _)$
shows *Brouwer_degree2* *p* *f* = *d*
proof (*cases a* $\in \text{carrier } ?G$)
case *True*
then show *?thesis*
by (*metis Brouwer_degree2_iff hi eq f group.trivial_group_subgroup_generated*
group_reduced_homology_group
 $\text{subset_singleton_iff trivial_reduced_homology_group_nsphere}$)
next
case *False*
then show *?thesis*
using *trivial_reduced_homology_group_nsphere* [of *p* *p*]
by (*metis group.trivial_group_subgroup_generated_eq disjoint_insert(1) eq*
group_reduced_homology_group inf_bot_right subset_singleton_iff)
qed

lemma *Brouwer_degree2_homotopic*:
assumes *homotopic_with* $(\lambda x. \text{True})$ (*nsphere* *p*) (*nsphere* *p*) *f* *g*
shows *Brouwer_degree2* *p* *f* = *Brouwer_degree2* *p* *g*
proof –
have *continuous_map* (*nsphere* *p*) (*nsphere* *p*) *f*
using *homotopic_with_imp_continuous_maps* [OF *assms*] **by** *auto*

```

show ?thesis
  using Brouwer_degree2_def assms homology_homotopy_empty by fastforce
qed

```

```

lemma Brouwer_degree2_id [simp]: Brouwer_degree2 p id = 1
proof (rule Brouwer_degree2_unique)
  fix x
  assume x:  $x \in \text{carrier } (\text{reduced\_homology\_group } (int\ p) \ (nsphere\ p))$ 
  then have  $x \in \text{carrier } (\text{homology\_group } (int\ p) \ (nsphere\ p))$ 
  using carrier_reduced_homology_group_subset by blast
  then show  $\text{hom\_induced } (int\ p) \ (nsphere\ p) \ \{\} \ (nsphere\ p) \ \{\} \ id\ x =$ 
     $x \ [\bigwedge]_{\text{reduced\_homology\_group } (int\ p) \ (nsphere\ p) \ (1::int)}$ 
  by (simp add: hom_induced_id group.int_pow_1 x)
qed auto

```

```

lemma Brouwer_degree2_compose:
  assumes f: continuous_map (nsphere p) (nsphere p) f and g: continuous_map
    (nsphere p) (nsphere p) g
  shows Brouwer_degree2 p (g ∘ f) = Brouwer_degree2 p g * Brouwer_degree2 p
    f
proof (rule Brouwer_degree2_unique)
  show continuous_map (nsphere p) (nsphere p) (g ∘ f)
  by (meson continuous_map_compose f g)
next
  fix x
  assume x:  $x \in \text{carrier } (\text{reduced\_homology\_group } (int\ p) \ (nsphere\ p))$ 
  have  $\text{hom\_induced } (int\ p) \ (nsphere\ p) \ \{\} \ (nsphere\ p) \ \{\} \ (g \circ f) =$ 
     $\text{hom\_induced } (int\ p) \ (nsphere\ p) \ \{\} \ (nsphere\ p) \ \{\} \ g \circ$ 
     $\text{hom\_induced } (int\ p) \ (nsphere\ p) \ \{\} \ (nsphere\ p) \ \{\} \ f$ 
  by (blast intro: hom_induced_compose [OF f g])
  with x show  $\text{hom\_induced } (int\ p) \ (nsphere\ p) \ \{\} \ (nsphere\ p) \ \{\} \ (g \circ f) \ x =$ 
     $x \ [\bigwedge]_{\text{reduced\_homology\_group } (int\ p) \ (nsphere\ p) \ (Brouwer\_degree2\ p\ g\ * \ Brouwer\_degree2\ p\ f)}$ 
  by (simp add: mult.commute hom_induced_reduced flip: Brouwer_degree2
    group.int_pow_pow)
qed

```

```

lemma Brouwer_degree2_homotopy_equivalence:
  assumes f: continuous_map (nsphere p) (nsphere p) f and g: continuous_map
    (nsphere p) (nsphere p) g
  and hom: homotopic_with ( $\lambda x. \text{True}$ ) (nsphere p) (nsphere p) (f ∘ g) id
  obtains  $|\text{Brouwer\_degree2 } p\ f| = 1 \ |\text{Brouwer\_degree2 } p\ g| = 1 \ \text{Brouwer\_degree2}$ 
     $p\ g = \text{Brouwer\_degree2 } p\ f$ 
  using Brouwer_degree2_homotopic [OF hom] Brouwer_degree2_compose f g
    zmult_eq_1_iff by auto

```

```

lemma Brouwer_degree2_homeomorphic_maps:
  assumes homeomorphic_maps (nsphere p) (nsphere p) f g
  obtains  $|\text{Brouwer\_degree2 } p\ f| = 1 \ |\text{Brouwer\_degree2 } p\ g| = 1 \ \text{Brouwer\_degree2}$ 

```

```

p g = Brouwer_degree2 p f
  using assms
  by (auto simp: homeomorphic_maps_def homotopic_with_equal continuous_map_compose
intro: Brouwer_degree2_homotopy_equivalence)

```

```

lemma Brouwer_degree2_retraction_map:
  assumes retraction_map (nsphere p) (nsphere p) f
  shows |Brouwer_degree2 p f| = 1
proof -
  obtain g where g: retraction_maps (nsphere p) (nsphere p) f g
    using assms by (auto simp: retraction_map_def)
  show ?thesis
proof (rule Brouwer_degree2_homotopy_equivalence)
  show homotopic_with ( $\lambda x. \text{True}$ ) (nsphere p) (nsphere p) (f  $\circ$  g) id
    using g apply (auto simp: retraction_maps_def)
    by (simp add: homotopic_with_equal continuous_map_compose)
  show continuous_map (nsphere p) (nsphere p) f continuous_map (nsphere p)
    (nsphere p) g
    using g retraction_maps_def by blast+
qed
qed

```

```

lemma Brouwer_degree2_section_map:
  assumes section_map (nsphere p) (nsphere p) f
  shows |Brouwer_degree2 p f| = 1
proof -
  obtain g where g: retraction_maps (nsphere p) (nsphere p) g f
    using assms by (auto simp: section_map_def)
  show ?thesis
proof (rule Brouwer_degree2_homotopy_equivalence)
  show homotopic_with ( $\lambda x. \text{True}$ ) (nsphere p) (nsphere p) (g  $\circ$  f) id
    using g apply (auto simp: retraction_maps_def)
    by (simp add: homotopic_with_equal continuous_map_compose)
  show continuous_map (nsphere p) (nsphere p) g continuous_map (nsphere p)
    (nsphere p) f
    using g retraction_maps_def by blast+
qed
qed

```

```

lemma Brouwer_degree2_homeomorphic_map:
  homeomorphic_map (nsphere p) (nsphere p) f  $\implies$  |Brouwer_degree2 p f| = 1
  using Brouwer_degree2_retraction_map section_and_retraction_eq_homeomorphic_map
  by blast

```

```

lemma Brouwer_degree2_nullhomotopic:
  assumes homotopic_with ( $\lambda x. \text{True}$ ) (nsphere p) (nsphere p) f ( $\lambda x. a$ )
  shows Brouwer_degree2 p f = 0

```

proof –

```

have contf: continuous_map (nsphere p) (nsphere p) f
and contc: continuous_map (nsphere p) (nsphere p) ( $\lambda x. a$ )
using homotopic_with_imp_continuous_maps [OF assms] by metis+
have Brouwer_degree2 p f = Brouwer_degree2 p ( $\lambda x. a$ )
using Brouwer_degree2_homotopic [OF assms] .
moreover
let ?G = reduced_homology_group (int p) (nsphere p)
interpret group ?G
by simp
have Brouwer_degree2 p ( $\lambda x. a$ ) = 0
proof (rule Brouwer_degree2_unique [OF contc])
fix c
assume c: c ∈ carrier ?G
have continuous_map (nsphere p) (subtopology (nsphere p) {a}) ( $\lambda f. a$ )
using contc continuous_map_in_subtopology by blast
then have he: hom_induced p (nsphere p) {} (nsphere p) {} ( $\lambda x. a$ )
      = hom_induced p (subtopology (nsphere p) {a}) {} (nsphere p) {} id
o
      hom_induced p (nsphere p) {} (subtopology (nsphere p) {a}) {}
( $\lambda x. a$ )
by (metis continuous_map_id_subt hom_induced_compose id_comp image_empty order_refl)
have 1: hom_induced p (nsphere p) {} (subtopology (nsphere p) {a}) {} ( $\lambda x. a$ )
c =
      1 reduced_homology_group (int p) (subtopology (nsphere p) {a})
using c trivial_reduced_homology_group_contractible_space [of subtopology
(nsphere p) {a} p]
by (simp add: hom_induced_reduced_contractible_space_subtopology_singleton
trivial_group_subset group.trivial_group_subset subset_iff)
show hom_induced (int p) (nsphere p) {} (nsphere p) {} ( $\lambda x. a$ ) c =
      c [ $\lceil$ ] ?G (0::int)
apply (simp add: he 1)
using hom_induced_reduced_hom_group_hom.hom_one_group_hom_axioms_def
group_hom_def group_reduced_homology_group by blast
qed
ultimately show ?thesis
by metis
qed

```

lemma *Brouwer_degree2_const*: *Brouwer_degree2* *p* ($\lambda x. a$) = 0

proof (*cases continuous_map*(*nsphere* *p*) (*nsphere* *p*) ($\lambda x. a$))

case *True*

then show *?thesis*

by (*auto intro: Brouwer_degree2_nullhomotopic* [**where** *a=a*])

next

case *False*

let *?G* = *reduced_homology_group* (*int* *p*) (*nsphere* *p*)

```

let ?H = homology_group (int p) (nsphere p)
interpret group ?G
  by simp
have eq1: 1 ?H = 1 ?G
  by (simp add: one_reduced_homology_group)
have *:  $\forall x \in \text{carrier } ?G. \text{hom\_induced } (int p) (nsphere p) \{ \} (nsphere p) \{ \} (\lambda x.$ 
a)  $x = 1 ?H$ 
  by (metis False hom_induced_default one_relative_homology_group)
obtain c where c:  $c \in \text{carrier } ?G$  and ceq:  $\text{subgroup\_generated } ?G \{c\} = ?G$ 
  using cyclic_reduced_homology_group_nsphere [of p p] by (force simp: cyclic_group_def)
have [simp]:  $\text{ord } c = 0$ 
  by (simp add: c ceq iso_finite [OF reduced_homology_group_nsphere(1)] flip:
infinite_cyclic_subgroup_order)
show ?thesis
  unfolding Brouwer_degree2_def
proof (rule some_equality)
  fix d :: int
  assume  $\forall x \in \text{carrier } ?G. \text{hom\_induced } (int p) (nsphere p) \{ \} (nsphere p) \{ \}$ 
  ( $\lambda x. a$ )  $x = x [\bigwedge] ?G d$ 
  then have c  $[\bigwedge] ?G d = 1 ?H$ 
  using * c by blast
  then have int ( $\text{ord } c$ ) dvd d
  using c eq1 int_pow_eq_id by auto
  then show  $d = 0$ 
  by (simp add: * del: one_relative_homology_group)
qed (use * eq1 in force)
qed

```

corollary *Brouwer_degree2_nonsurjective:*

```

[[continuous_map (nsphere p) (nsphere p) f; f ' topspace (nsphere p)  $\neq$  topspace
(nsphere p)]]
 $\implies \text{Brouwer\_degree2 } p f = 0$ 
by (meson Brouwer_degree2_nullhomotopic nullhomotopic_nonsurjective_sphere_map)

```

proposition *Brouwer_degree2_reflection:*

```

Brouwer_degree2 p ( $\lambda x i. \text{if } i = 0 \text{ then } -x i \text{ else } x i$ ) = -1 (is Brouwer_degree2
_ ?r = -1)

```

proof (*induction p*)

case 0

```
let ?G = homology_group 0 (nsphere 0)
```

```
let ?D = homology_group 0 (discrete_topology {()})
```

```
interpret group ?G
```

```
by simp
```

```
define r where  $r \equiv \lambda x :: \text{nat} \Rightarrow \text{real}. \lambda i. \text{if } i = 0 \text{ then } -x i \text{ else } x i$ 
```

```
then have [simp]:  $r \circ r = \text{id}$ 
```

```
by force
```

```
have cmr: continuous_map (nsphere 0) (nsphere 0) r
```



```

  by (simp add: r_def continuous_map_nsphere_reflection)
  have *: hom_induced 0 (nsphere 0) {} (nsphere 0) {} r c = inv ?G c
  if c ∈ carrier(reduced_homology_group 0 (nsphere 0)) for c
  proof -
    have c: c ∈ carrier ?G
    and ceq: hom_induced 0 (nsphere 0) {} (discrete_topology {}) {} (λx. ())
  c = 1 ?D
    using that by (auto simp: carrier_reduced_homology_group kernel_def)
    define pp::nat⇒real where pp ≡ λi. if i = 0 then 1 else 0
    define nn::nat⇒real where nn ≡ λi. if i = 0 then -1 else 0
    have topn0: topspace(nsphere 0) = {pp,nn}
    by (auto simp: nsphere pp_def nn_def fun_eq_iff power2_eq_1_iff split:
if_split_asm)
    have t1_space (nsphere 0)
    unfolding nsphere
    apply (rule t1_space_subtopology)
    by (metis (full_types) open_fun_def t1_space t1_space_def)
    then have dtn0: discrete_topology {pp,nn} = nsphere 0
    using finite_t1_space_imp_discrete_topology [OF topn0] by auto
    have pp ≠ nn
    by (auto simp: pp_def nn_def fun_eq_iff)
    have [simp]: r pp = nn r nn = pp
    by (auto simp: r_def pp_def nn_def fun_eq_iff)
    have iso: (λ(a,b). hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere
0) {} id a
      ⊗ ?G hom_induced 0 (subtopology (nsphere 0) {nn}) {} (nsphere 0)
      {} id b)
      ∈ iso (homology_group 0 (subtopology (nsphere 0) {pp}) ×× homol-
ogy_group 0 (subtopology (nsphere 0) {nn}))
      ?G (is ?f ∈ iso (?P ×× ?N) ?G)
    apply (rule homology_additivity_explicit)
    using dtn0 ⟨pp ≠ nn⟩ by (auto simp: discrete_topology_unique)
    then have fim: ?f ' carrier(?P ×× ?N) = carrier ?G
    by (simp add: iso_def bij_betw_def)
    obtain d d' where d: d ∈ carrier ?P and d': d' ∈ carrier ?N and eqc: ?f(d,d')
    = c
    using c by (force simp flip: fim)
    let ?h = λxx. hom_induced 0 (subtopology (nsphere 0) {xx}) {} (discrete_topology
{()}) {} (λx. ())
    have retraction_map (subtopology (nsphere 0) {pp}) (subtopology (nsphere 0)
{nn}) r
    apply (simp add: retraction_map_def retraction_maps_def continuous_map_in_subtopology
continuous_map_from_subtopology cmr image_subset_iff)
    apply (rule_tac x=r in exI)
    apply (force simp: retraction_map_def retraction_maps_def continuous_map_in_subtopology
continuous_map_from_subtopology cmr)
    done
    then have carrier ?N = (hom_induced 0 (subtopology (nsphere 0) {pp}) {}
(subtopology (nsphere 0) {nn}) {} r) ' carrier ?P

```

```

    by (rule surj_hom_induced_retraction_map)
  then obtain e where e: e ∈ carrier ?P and eqd': hom_induced 0 (subtopology
    (nsphere 0) {pp}) {} (subtopology (nsphere 0) {nn}) {} r e = d'
    using d' by auto
  have section_map (subtopology (nsphere 0) {pp}) (discrete_topology {}) (λx.
    ())
    by (force simp: section_map_def retraction_maps_def topn0)
  then have ?h pp ∈ mon ?P ?D
    by (rule mon_hom_induced_section_map)
  then have one: x = one ?P
    if ?h pp x = 1 ?D x ∈ carrier ?P for x
    using that by (simp add: mon_iff_hom_one)
  interpret hpd: group_hom ?P ?D ?h pp
    using hom_induced_empty_hom by (simp add: hom_induced_empty_hom
    group_hom_axioms_def group_hom_def)
  interpret hgd: group_hom ?G ?D hom_induced 0 (nsphere 0) {} (discrete_topology
    {}) {} (λx. ())
    using hom_induced_empty_hom by (simp add: hom_induced_empty_hom
    group_hom_axioms_def group_hom_def)
  interpret hpg: group_hom ?P ?G hom_induced 0 (subtopology (nsphere 0)
    {pp}) {} (nsphere 0) {} r
    using hom_induced_empty_hom by (simp add: hom_induced_empty_hom
    group_hom_axioms_def group_hom_def)
  interpret hgg: group_hom ?G ?G hom_induced 0 (nsphere 0) {} (nsphere 0)
    {} r
    using hom_induced_empty_hom by (simp add: hom_induced_empty_hom
    group_hom_axioms_def group_hom_def)
  have ?h pp d =
    (hom_induced 0 (nsphere 0) {} (discrete_topology {})) {} (λx. ())
    o hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere 0) {} id d
    by (simp flip: hom_induced_compose_empty)
  moreover
    have ?h pp = ?h nn o hom_induced 0 (subtopology (nsphere 0) {pp}) {}
    (subtopology (nsphere 0) {nn}) {} r
    by (simp add: cmr_continuous_map_from_subtopology_continuous_map_in_subtopology
    image_subset_iff flip: hom_induced_compose_empty)
  then have ?h pp e =
    (hom_induced 0 (nsphere 0) {} (discrete_topology {})) {} (λx. ())
    o hom_induced 0 (subtopology (nsphere 0) {nn}) {} (nsphere 0) {}
    id) d'
    by (simp flip: hom_induced_compose_empty eqd')
  ultimately have ?h pp (d ⊗?P e) = hom_induced 0 (nsphere 0) {} (discrete_topology
    {}) {} (λx. ()) (?f(d, d'))
    by (simp add: d e hom_induced_carrier)
  then have ?h pp (d ⊗?P e) = 1 ?D
    using ceq eqc by simp
  then have inv_p: inv ?P d = e
    by (metis (no_types, lifting) Group.group_def d e group.inv_equality group.r_inv
    group_relative_homology_group one monoid.m_closed)

```

```

    have cmr_pn: continuous_map (subtopology (nsphere 0) {pp}) (subtopology
(nsphere 0) {nn}) r
    by (simp add: cmr continuous_map_from_subtopology continuous_map_in_subtopology
image_subset_iff)
    then have hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere 0) {}
(id ∘ r) =
        hom_induced 0 (subtopology (nsphere 0) {nn}) {} (nsphere 0) {} id ∘
        hom_induced 0 (subtopology (nsphere 0) {pp}) {} (subtopology (nsphere
0) {nn}) {} r
    using hom_induced_compose_empty continuous_map_id_subt by blast
    then have inv?G hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere
0) {} r d =
        hom_induced 0 (subtopology (nsphere 0) {nn}) {} (nsphere 0) {}
id d'
    apply (simp add: flip: inv_p eqd')
    using d hpg.hom_inv by auto
    then have c: c = (hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere
0) {} id d)
        ⊗?G inv?G (hom_induced 0 (subtopology (nsphere 0) {pp}) {}
(nsphere 0) {} r d)
    by (simp flip: eqc)
    have hom_induced 0 (nsphere 0) {} (nsphere 0) {} r ∘
        hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere 0) {} id =
        hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere 0) {} r
    by (metis cmr comp_id continuous_map_id_subt hom_induced_compose_empty)
    moreover
    have hom_induced 0 (nsphere 0) {} (nsphere 0) {} r ∘
        hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere 0) {} r =
        hom_induced 0 (subtopology (nsphere 0) {pp}) {} (nsphere 0) {} id
    by (metis ⟨r ∘ r = id⟩ cmr continuous_map_from_subtopology hom_induced_compose_empty)
    ultimately show ?thesis
    by (metis inv_p c comp_def d e hgg.hom_inv hgg.hom_mult hom_induced_carrier
hpd.G.inv_inv hpg.hom_inv_inv_mult_group)
    qed
    show ?case
    unfolding r_def [symmetric]
    using Brouwer_degree2_unique [OF cmr]
    by (auto simp: * group.int_pow_neg group.int_pow_1 reduced_homology_group_def
intro!: Brouwer_degree2_unique [OF cmr])
next
case (Suc p)
let ?G = reduced_homology_group (int p) (nsphere p)
let ?G1 = reduced_homology_group (1 + int p) (nsphere (Suc p))
obtain f g where fg: group_isomorphisms ?G ?G1 f g
and *: ∀ c ∈ carrier ?G.
        hom_induced (1 + int p) (nsphere (Suc p)) {} (nsphere (Suc p)) {} ?r (f
c) =
        f (hom_induced p (nsphere p) {} (nsphere p) {} ?r c)
    using reduced_homology_group_nsphere_step

```

```

    by (meson group.iso_iff_group_isomorphisms group_reduced_homology_group)
  then have eq: carrier ?G1 = f ' carrier ?G
    by (fastforce simp add: iso_iff_dest: group_isomorphisms_imp_iso)
  interpret group_hom ?G ?G1 f
    by (meson fg group_hom_axioms_def group_hom_def group_isomorphisms_def
group_reduced_homology_group)
  have homf: f ∈ hom ?G ?G1
    using fg group_isomorphisms_def by blast
  have hom_induced (1 + int p) (nsphere (Suc p)) {} (nsphere (Suc p)) {} ?r (f
y) = f y [↑]?G1 (-1::int)
    if y ∈ carrier ?G for y
    by (simp add: that * Brouwer_degree2 Suc hom_int_pow)
  then show ?case
    by (fastforce simp: eq intro: Brouwer_degree2_unique [OF continuous_map_nsphere_reflection])
qed

end

```

0.4 Invariance of Domain

theory *Invariance_of_Domain*

imports *Brouwer_Degree HOL-Analysis.Continuous_Extension HOL-Analysis.Homeomorphism*

begin

0.4.1 Degree invariance mod 2 for map between pairs

theorem *Borsuk_odd_mapping_degree_step:*

assumes *cmf: continuous_map (nsphere n) (nsphere n) f*
and *f: $\bigwedge x. x \in \text{topspace}(\text{nsphere } n) \implies (f \circ (\lambda x i. -x i)) x = ((\lambda x i. -x i) \circ f) x$*

and *fim: $f \text{ ' } (\text{topspace}(\text{nsphere}(n - \text{Suc } 0))) \subseteq \text{topspace}(\text{nsphere}(n - \text{Suc } 0))$*

shows *even (Brouwer_degree2 n f - Brouwer_degree2 (n - Suc 0) f)*

proof (*cases n = 0*)

case *False*

define *neg* **where** *neg* $\equiv \lambda x::\text{nat} \Rightarrow \text{real}. \lambda i. -x i$

define *upper* **where** *upper* $\equiv \lambda n. \{x::\text{nat} \Rightarrow \text{real}. x n \geq 0\}$

define *lower* **where** *lower* $\equiv \lambda n. \{x::\text{nat} \Rightarrow \text{real}. x n \leq 0\}$

define *equator* **where** *equator* $\equiv \lambda n. \{x::\text{nat} \Rightarrow \text{real}. x n = 0\}$

define *usphere* **where** *usphere* $\equiv \lambda n. \text{subtopology } (\text{nsphere } n) (\text{upper } n)$

define *lsphere* **where** *lsphere* $\equiv \lambda n. \text{subtopology } (\text{nsphere } n) (\text{lower } n)$

have [*simp*]: *neg x i = -x i* **for** *x i*

by (*force simp: neg_def*)

have *equator_upper: equator n \subseteq upper n*

by (*force simp: equator_def upper_def*)

have *upper_usphere: subtopology (nsphere n) (upper n) = usphere n*

by (*simp add: usphere_def*)

let *?rhgn* = *relative_homology_group n (nsphere n)*

let *?hi_ee* = *hom_induced n (nsphere n) (equator n) (nsphere n) (equator n)*

```

interpret GE: comm_group ?rhgn (equator n)
by simp
interpret HB: group_hom ?rhgn (equator n)
      homology_group (int n - 1) (subtopology (nsphere n) (equator
n))
      hom_boundary n (nsphere n) (equator n)
by (simp add: group_hom_axioms_def group_hom_def hom_boundary_hom)
interpret HIU: group_hom ?rhgn (equator n)
      ?rhgn (upper n)
      hom_induced n (nsphere n) (equator n) (nsphere n) (upper
n) id
by (simp add: group_hom_axioms_def group_hom_def hom_induced_hom)
have subt_eq: subtopology (nsphere n) {x. x n = 0} = nsphere (n - Suc 0)
by (metis False Suc_pred le_zero_eq not_le subtopology_nsphere_equator)
then have equ: subtopology (nsphere n) (equator n) = nsphere(n - Suc 0)
      subtopology (lsphere n) (equator n) = nsphere(n - Suc 0)
      subtopology (usphere n) (equator n) = nsphere(n - Suc 0)
using False by (auto simp: lsphere_def usphere_def equator_def lower_def up-
per_def subtopology_subtopology simp flip: Collect_conj_eq cong: rev_conj_cong)
have cmr: continuous_map (nsphere(n - Suc 0)) (nsphere(n - Suc 0)) f
by (metis (no_types, lifting) IntE cmf fim continuous_map_from_subtopology
continuous_map_in_subtopology equ(1) image_subset_iff topspace_subtopology)

have f x n = 0 if x ∈ topspace (nsphere n) x n = 0 for x
proof -
have x ∈ topspace (nsphere (n - Suc 0))
by (simp add: that topspace_nsphere_minus1)
moreover have topspace (nsphere n) ∩ {f. f n = 0} = topspace (nsphere (n
- Suc 0))
by (metis subt_eq topspace_subtopology)
ultimately show ?thesis
using fim by auto
qed
then have fimeq: f ‘ (topspace (nsphere n) ∩ equator n) ⊆ topspace (nsphere n)
∩ equator n
using fim cmf by (auto simp: equator_def continuous_map_def image_subset_iff)
have ∧k. continuous_map (powertop_real UNIV) euclideanreal (λx. - x k)
by (metis UNIV_I continuous_map_product_projection continuous_map_minus)
then have cm_neg: continuous_map (nsphere m) (nsphere m) neg for m
by (force simp: nsphere continuous_map_in_subtopology neg_def continu-
ous_map_componentwise_UNIV intro: continuous_map_from_subtopology)
then have cm_neg_lu: continuous_map (lsphere n) (usphere n) neg
by (auto simp: lsphere_def usphere_def lower_def upper_def continuous_map_from_subtopology
continuous_map_in_subtopology)
have neg_in_top_iff: neg x ∈ topspace(nsphere m) ↔ x ∈ topspace(nsphere
m) for m x
by (simp add: nsphere_def neg_def topspace_Euclidean_space)
obtain z where zcarr: z ∈ carrier (reduced_homology_group (int n - 1)
(nsphere (n - Suc 0)))

```

```

    and zeq: subgroup_generated (reduced_homology_group (int n - 1) (nsphere
(n - Suc 0))) {z}
      = reduced_homology_group (int n - 1) (nsphere (n - Suc 0))
    using cyclic_reduced_homology_group_nsphere [of int n - 1 n - Suc 0] by
(auto simp: cyclic_group_def)
    have hom_boundary n (subtopology (nsphere n) {x. x n ≤ 0}) {x. x n = 0}
      ∈ Group.iso (relative_homology_group n
        (subtopology (nsphere n) {x. x n ≤ 0}) {x. x n = 0})
        (reduced_homology_group (int n - 1) (nsphere (n - Suc 0)))
    using iso_lower_hemisphere_reduced_homology_group [of int n - 1 n - Suc
0] False by simp
    then obtain gp where g: group_isomorphisms
      (relative_homology_group n (subtopology (nsphere n) {x. x n
≤ 0}) {x. x n = 0})
      (reduced_homology_group (int n - 1) (nsphere (n - Suc 0)))
      (hom_boundary n (subtopology (nsphere n) {x. x n ≤ 0}) {x.
x n = 0})
      gp
    by (auto simp: group_iso_iff_group_isomorphisms)
    then interpret gp: group_hom reduced_homology_group (int n - 1) (nsphere
(n - Suc 0))
      relative_homology_group n (subtopology (nsphere n) {x. x n ≤ 0}) {x. x n =
0} gp
    by (simp add: group_hom_axioms_def group_hom_def group_isomorphisms_def)
    obtain zp where zpcarr: zp ∈ carrier (relative_homology_group n (lsphere n)
(equator n))
      and zp_z: hom_boundary n (lsphere n) (equator n) zp = z
      and zp_sg: subgroup_generated (relative_homology_group n (lsphere n) (equator
n)) {zp}
      = relative_homology_group n (lsphere n) (equator n)
    proof
      show gp z ∈ carrier (relative_homology_group n (lsphere n) (equator n))
        hom_boundary n (lsphere n) (equator n) (gp z) = z
      using g zcarr by (auto simp: lsphere_def equator_def lower_def group_isomorphisms_def)
      have giso: gp ∈ Group.iso (reduced_homology_group (int n - 1) (nsphere (n
- Suc 0)))
        (relative_homology_group n (subtopology (nsphere n) {x. x n ≤
0}) {x. x n = 0})
      by (metis (mono_tags, lifting) g group_isomorphisms_imp_iso group_isomorphisms_sym)
      show subgroup_generated (relative_homology_group n (lsphere n) (equator n))
{gp z} =
        relative_homology_group n (lsphere n) (equator n)
      apply (rule monoid.equality)
      using giso gp.subgroup_generated_by_image [of {z}] zcarr
      by (auto simp: lsphere_def equator_def lower_def zeq gp.iso_iff)
    qed
    have hb_iso: hom_boundary n (subtopology (nsphere n) {x. x n ≥ 0}) {x. x n
= 0}
      ∈ iso (relative_homology_group n (subtopology (nsphere n) {x. x n ≥

```

```

0}) {x. x n = 0})
      (reduced_homology_group (int n - 1) (nsphere (n - Suc 0)))
    using iso_upper_hemisphere_reduced_homology_group [of int n - 1 n - Suc
0] False by simp
    then obtain gn where g: group_isomorphisms
      (relative_homology_group n (subtopology (nsphere n) {x. x n
≥ 0}) {x. x n = 0})
      (reduced_homology_group (int n - 1) (nsphere (n - Suc 0)))
      (hom_boundary n (subtopology (nsphere n) {x. x n ≥ 0}) {x.
x n = 0})
      gn
    by (auto simp: group_iso_iff_group_isomorphisms)
    then interpret gn: group_hom reduced_homology_group (int n - 1) (nsphere
(n - Suc 0))
      relative_homology_group n (subtopology (nsphere n) {x. x n ≥ 0}) {x. x n =
0} gn
    by (simp add: group_hom_axioms_def group_hom_def group_isomorphisms_def)
    obtain zn where zncarr: zn ∈ carrier(relative_homology_group n (usphere n)
(equator n))
      and zn_z: hom_boundary n (usphere n) (equator n) zn = z
      and zn_sg: subgroup_generated (relative_homology_group n (usphere n) (equator
n)) {zn}
      = relative_homology_group n (usphere n) (equator n)
    proof
      show gn z ∈ carrier (relative_homology_group n (usphere n) (equator n))
        hom_boundary n (usphere n) (equator n) (gn z) = z
      using g zcarr by (auto simp: usphere_def equator_def upper_def group_isomorphisms_def)
      have giso: gn ∈ Group.iso (reduced_homology_group (int n - 1) (nsphere (n
- Suc 0)))
        (relative_homology_group n (subtopology (nsphere n) {x. x n ≥
0}) {x. x n = 0})
      by (metis (mono_tags, lifting) g group_isomorphisms_imp_iso group_isomorphisms_sym)
      show subgroup_generated (relative_homology_group n (usphere n) (equator n))
{gn z} =
        relative_homology_group n (usphere n) (equator n)
      apply (rule monoid.equality)
      using giso gn.subgroup_generated_by_image [of {z}] zcarr
      by (auto simp: usphere_def equator_def upper_def zeq gn.iso_iff)
    qed
    let ?hi_lu = hom_induced n (lsphere n) (equator n) (nsphere n) (upper n) id
    interpret gh_lu: group_hom relative_homology_group n (lsphere n) (equator n)
?rhgn (upper n) ?hi_lu
      by (simp add: group_hom_axioms_def group_hom_def hom_induced_hom)
    interpret gh_eef: group_hom ?rhgn (equator n) ?rhgn (equator n) ?hi_ee f
      by (simp add: group_hom_axioms_def group_hom_def hom_induced_hom)
    define wp where wp ≡ ?hi_lu zp
    then have wpcarr: wp ∈ carrier(?rhgn (upper n))
      by (simp add: hom_induced_carrier)
    have hom_induced n (nsphere n) {} (nsphere n) {x. x n ≥ 0} id

```

```

    ∈ iso (reduced_homology_group n (nsphere n))
      (?rhgn {x. x n ≥ 0})
  using iso_reduced_homology_group_upper_hemisphere [of n n n] by auto
  then have carrier(?rhgn {x. x n ≥ 0})
    ⊆ (hom_induced n (nsphere n) {}) (nsphere n) {x. x n ≥ 0} id
      'carrier(reduced_homology_group n (nsphere n))
  by (simp add: iso_iff)
  then obtain vp where vpcarr: vp ∈ carrier(reduced_homology_group n (nsphere
n))
    and eqvp: hom_induced n (nsphere n) {} (nsphere n) (upper n) id vp = vp
  using vpcarr by (auto simp: upper_def)
  define wn where wn ≡ hom_induced n (usphere n) (equator n) (nsphere n)
(lower n) id zn
  then have uncarr: wn ∈ carrier(?rhgn (lower n))
    by (simp add: hom_induced_carrier)
  have hom_induced n (nsphere n) {} (nsphere n) {x. x n ≤ 0} id
    ∈ iso (reduced_homology_group n (nsphere n))
      (?rhgn {x. x n ≤ 0})
  using iso_reduced_homology_group_lower_hemisphere [of n n n] by auto
  then have carrier(?rhgn {x. x n ≤ 0})
    ⊆ (hom_induced n (nsphere n) {}) (nsphere n) {x. x n ≤ 0} id
      'carrier(reduced_homology_group n (nsphere n))
  by (simp add: iso_iff)
  then obtain vn where vpcarr: vn ∈ carrier(reduced_homology_group n (nsphere
n))
    and eqvp: hom_induced n (nsphere n) {} (nsphere n) (lower n) id vn = vn
  using uncarr by (auto simp: lower_def)
  define up where up ≡ hom_induced n (lsphere n) (equator n) (nsphere n)
(equator n) id zp
  then have upcarr: up ∈ carrier(?rhgn (equator n))
    by (simp add: hom_induced_carrier)
  define un where un ≡ hom_induced n (usphere n) (equator n) (nsphere n)
(equator n) id zn
  then have uncarr: un ∈ carrier(?rhgn (equator n))
    by (simp add: hom_induced_carrier)
  have *: (λ(x, y).
    hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id x
    ⊗ ?rhgn (equator n)
    hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id y)
    ∈ Group.iso
      (relative_homology_group n (lsphere n) (equator n) × ×
        relative_homology_group n (usphere n) (equator n))
      (?rhgn (equator n))
  proof (rule conjunct1 [OF exact_sequence_sum_lemma [OF abelian_relative_homology_group]])
    show hom_induced n (lsphere n) (equator n) (nsphere n) (upper n) id
      ∈ Group.iso (relative_homology_group n (lsphere n) (equator n))
        (?rhgn (upper n))
    apply (simp add: lsphere_def usphere_def equator_def lower_def upper_def)
    using iso_relative_homology_group_lower_hemisphere by blast

```



```

show hom_induced n (usphere n) (equator n) (nsphere n) (lower n) id
  ∈ Group.iso (relative_homology_group n (usphere n) (equator n))
    (?rhgn (lower n))
apply (simp_all add: lsphere_def usphere_def equator_def lower_def up-
per_def)
using iso_relative_homology_group_upper_hemisphere by blast+
show exact_seq
  ([?rhgn (lower n),
    ?rhgn (equator n),
    relative_homology_group n (lsphere n) (equator n)],
   [hom_induced n (nsphere n) (equator n) (nsphere n) (lower n) id,
    hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id])
unfolding lsphere_def usphere_def equator_def lower_def upper_def
by (rule homology_exactness_triple_3) force
show exact_seq
  ([?rhgn (upper n),
    ?rhgn (equator n),
    relative_homology_group n (usphere n) (equator n)],
   [hom_induced n (nsphere n) (equator n) (nsphere n) (upper n) id,
    hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id])
unfolding lsphere_def usphere_def equator_def lower_def upper_def
by (rule homology_exactness_triple_3) force
next
fix x
assume x ∈ carrier (relative_homology_group n (lsphere n) (equator n))
show hom_induced n (nsphere n) (equator n) (nsphere n) (upper n) id
  (hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id x) =
  hom_induced n (lsphere n) (equator n) (nsphere n) (upper n) id x
by (simp add: hom_induced_compose' subset_iff lsphere_def usphere_def
equator_def lower_def upper_def)
next
fix x
assume x ∈ carrier (relative_homology_group n (usphere n) (equator n))
show hom_induced n (nsphere n) (equator n) (nsphere n) (lower n) id
  (hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id x) =
  hom_induced n (usphere n) (equator n) (nsphere n) (lower n) id x
by (simp add: hom_induced_compose' subset_iff lsphere_def usphere_def
equator_def lower_def upper_def)
qed
then have sb: carrier (?rhgn (equator n))
  ⊆ (λ(x, y).
    hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id x
    ⊗ ?rhgn (equator n)
    hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id y)
    ' carrier (relative_homology_group n (lsphere n) (equator n) × ×
      relative_homology_group n (usphere n) (equator n))
by (simp add: iso_iff)
obtain a b::int
where up_ab: ?hi_ee f up

```

```

      = up [⌈] ?rhgn (equator n)  $a \otimes ?rhgn (equator n)$  un [⌈] ?rhgn (equator n)  $b$ 
proof -
  have hiupcarr: ?hi_ee f up ∈ carrier (?rhgn (equator n))
  by (simp add: hom_induced_carrier)
  obtain u v where u: u ∈ carrier (relative_homology_group n (lsphere n)
(equator n))
  and v: v ∈ carrier (relative_homology_group n (usphere n) (equator n))
  and eq: ?hi_ee f up =
    hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id u
     $\otimes ?rhgn (equator n)$ 
    hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id v
  using subsetD [OF sb hiupcarr] by auto
  have u ∈ carrier (subgroup_generated (relative_homology_group n (lsphere n)
(equator n)) {zp})
  by (simp_all add: u zp_sg)
  then obtain a::int where a: u = zp [⌈]relative_homology_group n (lsphere n) (equator n)
a
  by (metis group.carrier_subgroup_generated_by_singleton group_relative_homology_group
rangeE zpcarr)
  have ae: hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id
    (pow (relative_homology_group n (lsphere n) (equator n)) zp a)
    = pow (?rhgn (equator n)) (hom_induced n (lsphere n) (equator n) (nsphere
n) (equator n) id zp) a
  by (meson group_hom.hom_int_pow group_hom_axioms_def group_hom_def
group_relative_homology_group hom_induced zpcarr)
  have v ∈ carrier (subgroup_generated (relative_homology_group n (usphere n)
(equator n)) {zn})
  by (simp_all add: v zn_sg)
  then obtain b::int where b: v = zn [⌈]relative_homology_group n (usphere n) (equator n)
b
  by (metis group.carrier_subgroup_generated_by_singleton group_relative_homology_group
rangeE zncarr)
  have be: hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id
    (zn [⌈]relative_homology_group n (usphere n) (equator n)  $b$ )
    = hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id
    zn [⌈]relative_homology_group n (nsphere n) (equator n)  $b$ 
  by (meson group_hom.hom_int_pow group_hom_axioms_def group_hom_def
group_relative_homology_group hom_induced zncarr)
  show thesis
proof
  show ?hi_ee f up
    = up [⌈] ?rhgn (equator n)  $a \otimes ?rhgn (equator n)$  un [⌈] ?rhgn (equator n)  $b$ 
  using a ae b be eq local.up_def un_def by auto
qed
qed
have (hom_boundary n (nsphere n) (equator n)
   $\circ$  hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) id) zp = z
using zp_z equ apply (simp add: lsphere_def naturality_hom_induced)

```

```

    by (metis hom_boundary_carrier hom_induced_id)
  then have up_z: hom_boundary n (nsphere n) (equator n) up = z
    by (simp add: up_def)
  have (hom_boundary n (nsphere n) (equator n)
    ◦ hom_induced n (usphere n) (equator n) (nsphere n) (equator n) id) zn = z
    using zn_z equ apply (simp add: usphere_def naturality_hom_induced)
  by (metis hom_boundary_carrier hom_induced_id)
  then have un_z: hom_boundary n (nsphere n) (equator n) un = z
    by (simp add: un_def)
  have Bd_ab: Brouwer_degree2 (n - Suc 0) f = a + b
  proof (rule Brouwer_degree2_unique_generator; use False int_ops in simp_all)
    show continuous_map (nsphere (n - Suc 0)) (nsphere (n - Suc 0)) f
      using cmr by auto
    show subgroup_generated (reduced_homology_group (int n - 1) (nsphere (n
- Suc 0))) {z} =
      reduced_homology_group (int n - 1) (nsphere (n - Suc 0))
    using zeq by blast
    have (hom_induced (int n - 1) (nsphere (n - Suc 0)) {}) (nsphere (n - Suc
0)) {} f
      ◦ hom_boundary n (nsphere n) (equator n)) up
      = (hom_boundary n (nsphere n) (equator n) ◦
        ?hi_ee f) up
    using naturality_hom_induced [OF cmf fimeq, of n, symmetric]
    by (simp add: subtopology_restrict equ_fun_eq_iff)
    also have ... = hom_boundary n (nsphere n) (equator n)
      (up [⋂relative_homology_group n (nsphere n) (equator n)
        a ⊗ relative_homology_group n (nsphere n) (equator n)
        un [⋂relative_homology_group n (nsphere n) (equator n) b])
    by (simp add: o_def up_ab)
    also have ... = z [⋂reduced_homology_group (int n - 1) (nsphere (n - Suc 0))
(a + b)]
    using zcarr
    apply (simp add: HB.hom_int_pow reduced_homology_group_def group.int_pow_subgroup_generated
upcarr uncarr)
    by (metis equ(1) group.int_pow_mult group_relative_homology_group hom_boundary_carrier
un_z up_z)
    finally show hom_induced (int n - 1) (nsphere (n - Suc 0)) {} (nsphere (n
- Suc 0)) {} f z =
      z [⋂reduced_homology_group (int n - 1) (nsphere (n - Suc 0)) (a + b)]
    by (simp add: up_z)
  qed
  define u where u ≡ up ⊗ ?rhgn (equator n) inv ?rhgn (equator n) un
  have ucarr: u ∈ carrier (?rhgn (equator n))
    by (simp add: u_def uncarr upcarr)
  then have u [⋂?rhgn (equator n) Brouwer_degree2 n f = u [⋂?rhgn (equator n)
(a - b)]
    ↔ (GE.ord u) dvd a - b - Brouwer_degree2 n f
  by (simp add: GE.int_pow_eq)

```

```

moreover
have  $GE.ord\ u = 0$ 
proof (clarsimp simp add: GE.ord_eq_0 ucarr)
  fix  $d :: nat$ 
  assume  $0 < d$ 
  and  $u \in ?rhgn\ (equator\ n)\ d = singular\_relboundary\_set\ n\ (nsphere\ n)$ 
  (equator n)
  then have  $hom\_induced\ n\ (nsphere\ n)\ (equator\ n)\ (nsphere\ n)\ (upper\ n)\ id\ u$ 
   $\in ?rhgn\ (upper\ n)\ d$ 
   $= \mathbf{1}_{?rhgn\ (upper\ n)}$ 
  by (metis HIU.hom_one HIU.hom_nat_pow one_relative_homology_group
ucarr)
  moreover
  have  $?hi\_lu$ 
   $= hom\_induced\ n\ (nsphere\ n)\ (equator\ n)\ (nsphere\ n)\ (upper\ n)\ id \circ$ 
   $hom\_induced\ n\ (lsphere\ n)\ (equator\ n)\ (nsphere\ n)\ (equator\ n)\ id$ 
  by (simp add: lsphere_def image_subset_iff equator_upper_flip: hom_induced_compose)
  then have  $p: wp = hom\_induced\ n\ (nsphere\ n)\ (equator\ n)\ (nsphere\ n)\ (upper$ 
 $n)\ id\ up$ 
  by (simp add: local.up_def wp_def)
  have  $n: hom\_induced\ n\ (nsphere\ n)\ (equator\ n)\ (nsphere\ n)\ (upper\ n)\ id\ un =$ 
 $\mathbf{1}_{?rhgn\ (upper\ n)}$ 
  using homology_exactness_triple_3 [OF equator_upper, of n nsphere n]
  using un_def zncarr by (auto simp: upper_usphere kernel_def)
  have  $hom\_induced\ n\ (nsphere\ n)\ (equator\ n)\ (nsphere\ n)\ (upper\ n)\ id\ u = wp$ 
  unfolding u_def
  using  $p\ n\ HIU.inv\_one\ HIU.r\_one\ uncarr\ upcarr$  by auto
  ultimately have  $(wp \in ?rhgn\ (upper\ n)\ d) = \mathbf{1}_{?rhgn\ (upper\ n)}$ 
  by simp
moreover have infinite (carrier (subgroup_generated (?rhgn (upper n)) {wp}))
proof –
  have  $?rhgn\ (upper\ n) \cong reduced\_homology\_group\ n\ (nsphere\ n)$ 
  unfolding upper_def
  using iso_reduced_homology_group_upper_hemisphere [of n n n]
  by (blast intro: group.iso_sym group_reduced_homology_group is_isoI)
  also have  $\dots \cong integer\_group$ 
  by (simp add: reduced_homology_group_nsphere)
  finally have iso: ?rhgn (upper n)  $\cong$  integer_group .
  have  $carrier\ (subgroup\_generated\ (?rhgn\ (upper\ n))\ \{wp\}) = carrier\ (?rhgn$ 
 $(upper\ n))$ 
  using gh_lu.subgroup_generated_by_image [of {zp}] zpcarr HIU.carrier_subgroup_generated_sub
gh_lu.iso_iff iso_relative_homology_group_lower_hemisphere zp_sg
  by (auto simp: lower_def lsphere_def upper_def equator_def wp_def)
  then show ?thesis
  using infinite_UNIV_int iso_finite [OF iso] by simp
qed
ultimately show False
  using HIU.finite_cyclic_subgroup <0 < d> wpcarr by blast
qed

```

```

ultimately have iff:  $u \lceil \lceil ?rhgn \text{ (equator } n) \text{ Brouwer\_degree2 } n \text{ } f = u \lceil \lceil ?rhgn \text{ (equator } n) \text{ (} a - b \text{)}$ 
 $\longleftrightarrow \text{Brouwer\_degree2 } n \text{ } f = a - b$ 
  by auto
  have  $u \lceil \lceil ?rhgn \text{ (equator } n) \text{ Brouwer\_degree2 } n \text{ } f = ?hi\_ee \text{ } f \text{ } u$ 
  proof -
    have  $ne: \text{topspace } (nsphere \text{ } n) \cap \text{equator } n \neq \{\}$ 
    using False equator_def in_topspace_nsphere by fastforce
    have  $eq1: \text{hom\_boundary } n \text{ (nsphere } n) \text{ (equator } n) \text{ } u$ 
       $= 1_{\text{reduced\_homology\_group } (int \text{ } n - 1) \text{ (subtopology } (nsphere \text{ } n) \text{ (equator } n))}$ 
    using one_reduced_homology_group u_def un_z uncarr up_z upcarr by force
    then have  $uhom: u \in \text{hom\_induced } n \text{ (nsphere } n) \{\} \text{ (nsphere } n) \text{ (equator } n)$ 
  id '
       $\text{carrier } (\text{reduced\_homology\_group } (int \text{ } n) \text{ (nsphere } n))$ 
    using homology_exactness_reduced_1 [OF ne, of n] eq1 ucarr by (auto simp:
kernel_def)
    then obtain  $v$  where  $vcarr: v \in \text{carrier } (\text{reduced\_homology\_group } (int \text{ } n) \text{ (nsphere } n))$ 
      and  $ueq: u = \text{hom\_induced } n \text{ (nsphere } n) \{\} \text{ (nsphere } n) \text{ (equator } n)$ 
  id  $v$ 
    by blast
    interpret  $GH\_hi: \text{group\_hom } \text{homology\_group } n \text{ (nsphere } n) \text{ } ?rhgn \text{ (equator } n)$ 
       $\text{hom\_induced } n \text{ (nsphere } n) \{\} \text{ (nsphere } n) \text{ (equator } n) \text{ } id$ 
    by (simp add: group_hom_axioms_def group_hom_def hom_induced_hom)
    have  $poweq: \text{pow } (\text{homology\_group } n \text{ (nsphere } n)) \text{ } x \text{ } i = \text{pow } (\text{reduced\_homology\_group } n \text{ (nsphere } n)) \text{ } x \text{ } i$ 
    for  $x$  and  $i::int$ 
    by (simp add: False un_reduced_homology_group)
    have  $vcarr': v \in \text{carrier } (\text{homology\_group } n \text{ (nsphere } n))$ 
    using carrier_reduced_homology_group_subset vcarr by blast
    have  $u \lceil \lceil ?rhgn \text{ (equator } n) \text{ Brouwer\_degree2 } n \text{ } f$ 
       $= \text{hom\_induced } n \text{ (nsphere } n) \{\} \text{ (nsphere } n) \text{ (equator } n) \text{ } f \text{ } v$ 
    using vcarr vcarr'
    by (simp add: ueq poweq hom_induced_compose' cmf flip: GH_hi.hom_int_pow
Brouwer_degree2)
    also have  $\dots = \text{hom\_induced } n \text{ (nsphere } n) \text{ (topspace } (nsphere \text{ } n) \cap \text{equator } n) \text{ (nsphere } n) \text{ (equator } n) \text{ } f$ 
       $(\text{hom\_induced } n \text{ (nsphere } n) \{\} \text{ (nsphere } n) \text{ (topspace } (nsphere \text{ } n) \cap \text{equator } n) \text{ } id \text{ } v)$ 
    using fimeq by (simp add: hom_induced_compose' cmf)
    also have  $\dots = ?hi\_ee \text{ } f \text{ } u$ 
    by (metis hom_induced inf.left_idem ueq)
    finally show ?thesis .
  qed
  moreover
  interpret  $gh\_een: \text{group\_hom } ?rhgn \text{ (equator } n) \text{ } ?rhgn \text{ (equator } n) \text{ } ?hi\_ee \text{ } neg$ 
    by (simp add: group_hom_axioms_def group_hom_def hom_induced_hom)
  have  $hi\_up\_eq\_un: ?hi\_ee \text{ } neg \text{ } up = un \lceil \lceil ?rhgn \text{ (equator } n) \text{ Brouwer\_degree2}$ 

```

```

(n - Suc 0) neg
proof -
  have ?hi_ee neg (hom_induced n (lsphere n) (equator n) (nsphere n) (equator
n) id zp)
    = hom_induced n (lsphere n) (equator n) (nsphere n) (equator n) (neg ∘
id) zp
  by (intro hom_induced_compose') (auto simp: lsphere_def equator_def cm_neg)
  also have ... = hom_induced n (usphere n) (equator n) (nsphere n) (equator
n) id
    (hom_induced n (lsphere n) (equator n) (usphere n) (equator n) neg zp)
  by (subst hom_induced_compose' [OF cm_neg_lu]) (auto simp: usphere_def
equator_def)
  also have hom_induced n (lsphere n) (equator n) (usphere n) (equator n) neg
zp
    = zn [⋂]relative_homology_group n (usphere n) (equator n) Brouwer_degree2
(n - Suc 0) neg
  proof -
    let ?hb = hom_boundary n (usphere n) (equator n)
    have eq: subtopology (nsphere n) {x. x n ≥ 0} = usphere n ∧ {x. x n = 0}
= equator n
    by (auto simp: usphere_def upper_def equator_def)
    with hb_iso have inj: inj_on (?hb) (carrier (relative_homology_group n
(usphere n) (equator n)))
    by (simp add: iso_iff)
    interpret hb_hom: group_hom relative_homology_group n (usphere n)
(equator n)
    reduced_homology_group (int n - 1) (nsphere (n -
Suc 0))
    ?hb
  using hb_iso iso_iff eq group_hom_axioms_def group_hom_def by fastforce
  show ?thesis
  proof (rule inj_onD [OF inj])
    have *: hom_induced (int n - 1) (nsphere (n - Suc 0)) {} (nsphere (n -
Suc 0)) {} neg z
    = z [⋂]homology_group (int n - 1) (nsphere (n - Suc 0)) Brouwer_degree2
(n - Suc 0) neg
    using Brouwer_degree2 [of z n - Suc 0 neg] False zcarr
  by (simp add: int_ops group.int_pow_subgroup_generated reduced_homology_group_def)
  have ?hb ∘
    hom_induced n (lsphere n) (equator n) (usphere n) (equator n) neg
    = hom_induced (int n - 1) (nsphere (n - Suc 0)) {} (nsphere (n -
Suc 0)) {} neg ∘
    hom_boundary n (lsphere n) (equator n)
  apply (subst naturality_hom_induced [OF cm_neg_lu])
  apply (force simp: equator_def neg_def)
  by (simp add: equ)
  then have ?hb
    (hom_induced n (lsphere n) (equator n) (usphere n) (equator n)
neg zp)

```

```

    = (z [⋀]homology_group (int n - 1) (nsphere (n - Suc 0)) Brouwer_degree2
      (n - Suc 0) neg)
    by (metis * comp_apply zp_z)
    also have ... = ?hb (zn [⋀]relative_homology_group n (usphere n) (equator n)
      Brouwer_degree2 (n - Suc 0) neg)
    by (metis group.int_pow_subgroup_generated group_relative_homology_group
      hb_hom.hom_int_pow_reduced_homology_group_def zcarr zn_z zncarr)
    finally show ?hb (hom_induced n (lsphere n) (equator n) (usphere n)
      (equator n) neg zp) =
      ?hb (zn [⋀]relative_homology_group n (usphere n) (equator n)
        Brouwer_degree2 (n - Suc 0) neg) by simp
    qed (auto simp: hom_induced_carrier group.int_pow_closed zncarr)
  qed
  finally show ?thesis
  by (metis (no_types, lifting) group_hom.hom_int_pow group_hom_axioms_def
    group_hom_def group_relative_homology_group hom_induced local.up_def un_def
    zncarr)
  qed
  have continuous_map (nsphere (n - Suc 0)) (nsphere (n - Suc 0)) neg
    using cm_neg by blast
  then have homeomorphic_map (nsphere (n - Suc 0)) (nsphere (n - Suc 0))
    neg
    apply (auto simp: homeomorphic_map_maps homeomorphic_maps_def)
    apply (rule_tac x=neg in exI, auto)
    done
  then have Brouwer_degree2_21: Brouwer_degree2 (n - Suc 0) neg ^ 2 = 1
    using Brouwer_degree2_homeomorphic_map power2_eq_1_iff by force
  have hi_un_eq_up: ?hi_ee neg un = up [⋀]?rhgn (equator n) Brouwer_degree2
    (n - Suc 0) neg (is ?f un = ?y)
  proof -
    have [simp]: neg ∘ neg = id
      by force
    have ?f (?f ?y) = ?y
      apply (subst hom_induced_compose' [OF cm_neg __ cm_neg])
      apply (force simp: equator_def)
      apply (simp add: upcarr hom_induced_id_gen)
      done
    moreover have ?f ?y = un
      using upcarr apply (simp only: gh_ee.hom_int_pow hi_up_eq_un)
    by (metis (no_types, lifting) Brouwer_degree2_21 GE.group_l_invI GE.l_inv_ex
      group.int_pow_1 group.int_pow_pow power2_eq_1_iff uncarr zmult_eq_1_iff)
    ultimately show ?f un = ?y
      by simp
  qed
  have ?hi_ee f un = un [⋀]?rhgn (equator n) a ⊗ ?rhgn (equator n) up [⋀]?rhgn (equator n)
  b
  proof -
    let ?TE = topspace (nsphere n) ∩ equator n

```

```

have fneg: (f ∘ neg) x = (neg ∘ f) x if x ∈ topspace (nsphere n) for x
using f [OF that] by (force simp: neg_def)
have neg_im: neg ‘ (topspace (nsphere n) ∩ equator n) ⊆ topspace (nsphere n)
∩ equator n
by (metis cm_neg continuous_map_image_subset_topspace equ(1) topspace_subtopology)
have 1: hom_induced n (nsphere n) ?TE (nsphere n) ?TE f ∘ hom_induced n
(nsphere n) ?TE (nsphere n) ?TE neg
= hom_induced n (nsphere n) ?TE (nsphere n) ?TE neg ∘ hom_induced
n (nsphere n) ?TE (nsphere n) ?TE f
using neg_im fimeq cm_neg cmf
apply (simp add: flip: hom_induced_compose del: hom_induced_restrict)
using fneg by (auto intro: hom_induced_eq)
have (un [⌈]?rhgn (equator n) a) ⊗ ?rhgn (equator n) (up [⌈]?rhgn (equator n) b)
= un [⌈]?rhgn (equator n) (Brouwer_degree2 (n - 1) neg * a * Brouwer_degree2
(n - 1) neg)
⊗ ?rhgn (equator n)
up [⌈]?rhgn (equator n) (Brouwer_degree2 (n - 1) neg * b * Brouwer_degree2
(n - 1) neg)
proof -
have Brouwer_degree2 (n - Suc 0) neg = 1 ∨ Brouwer_degree2 (n - Suc
0) neg = - 1
using Brouwer_degree2_21 power2_eq_1_iff by blast
then show ?thesis
by fastforce
qed
also have ... = ((un [⌈]?rhgn (equator n) Brouwer_degree2 (n - 1) neg)
[⌈]?rhgn (equator n) a ⊗ ?rhgn (equator n)
(up [⌈]?rhgn (equator n) Brouwer_degree2 (n - 1) neg) [⌈]?rhgn (equator n)
b) [⌈]?rhgn (equator n)
Brouwer_degree2 (n - 1) neg
by (simp add: GE.int_pow_distrib GE.int_pow_pow uncarr upcarr)
also have ... = ?hi_ee neg (?hi_ee f up) [⌈]?rhgn (equator n) Brouwer_degree2
(n - Suc 0) neg
by (simp add: gh_een.hom_int_pow hi_un_eq_up hi_up_eq_un uncarr
up_ab upcarr)
finally have 2: (un [⌈]?rhgn (equator n) a) ⊗ ?rhgn (equator n) (up [⌈]?rhgn (equator n)
b)
= ?hi_ee neg (?hi_ee f up) [⌈]?rhgn (equator n) Brouwer_degree2 (n -
Suc 0) neg .
have un = ?hi_ee neg up [⌈]?rhgn (equator n) Brouwer_degree2 (n - Suc 0)
neg
by (metis (no_types, opaque_lifting) Brouwer_degree2_21 GE.int_pow_1
GE.int_pow_pow hi_up_eq_un power2_eq_1_iff uncarr zmult_eq_1_iff)
moreover have ?hi_ee f ((?hi_ee neg up) [⌈]?rhgn (equator n) (Brouwer_degree2
(n - Suc 0) neg))
= un [⌈]?rhgn (equator n) a ⊗ ?rhgn (equator n) up [⌈]?rhgn (equator n)
b

```



```

    using 1 2 by (simp add: hom_induced_carrier gh_eef.hom_int_pow fun_eq_iff)
    ultimately show ?thesis
      by blast
  qed
  then have ?hi_ee f u = u [↑]?rhgn (equator n) (a - b)
    by (simp add: u_def upcarr uncarr up_ab GE.int_pow_diff GE.m_ac GE.int_pow_distrib
    GE.int_pow_inv GE.inv_mult_group)
  ultimately
  have Brouwer_degree2 n f = a - b
    using iff by blast
  with Bd_ab show ?thesis
    by simp
qed simp

```

0.4.2 General Jordan-Brouwer separation theorem and invariance of dimension

proposition *relative_homology_group_Euclidean_complement_step:*

```

  assumes closedin (Euclidean_space n) S
  shows relative_homology_group p (Euclidean_space n) (topspace(Euclidean_space
n) - S)
    ≅ relative_homology_group (p + k) (Euclidean_space (n+k)) (topspace(Euclidean_space
(n+k)) - S)
proof -
  have *: relative_homology_group p (Euclidean_space n) (topspace(Euclidean_space
n) - S)
    ≅ relative_homology_group (p + 1) (Euclidean_space (Suc n)) (topspace(Euclidean_space
(Suc n)) - {x ∈ S. x n = 0})
    (is ?lhs ≅ ?rhs)
    if clo: closedin (Euclidean_space (Suc n)) S and cong:  $\bigwedge x y. \llbracket x \in S; \bigwedge i. i \neq n \implies x i = y i \rrbracket \implies y \in S$ 
    for p n S
  proof -
    have Ssub:  $S \subseteq \text{topspace}(\text{Euclidean\_space}(\text{Suc } n))$ 
      by (meson clo closedin_def)
    define lo where lo  $\equiv \{x \in \text{topspace}(\text{Euclidean\_space}(\text{Suc } n)). x n < (\text{if } x \in S \text{ then } 0 \text{ else } 1)\}$ 
    define hi where hi  $\equiv \{x \in \text{topspace}(\text{Euclidean\_space}(\text{Suc } n)). x n > (\text{if } x \in S \text{ then } 0 \text{ else } -1)\}$ 
    have lo_hi_Int:  $lo \cap hi = \{x \in \text{topspace}(\text{Euclidean\_space}(\text{Suc } n)) - S. x n \in \{-1 < .. < 1\}\}$ 
      by (auto simp: hi_def lo_def)
    have lo_hi_Un:  $lo \cup hi = \text{topspace}(\text{Euclidean\_space}(\text{Suc } n)) - \{x \in S. x n = 0\}$ 
      by (auto simp: hi_def lo_def)
    define ret where ret  $\equiv \lambda c::\text{real}. \lambda x i. \text{if } i = n \text{ then } c \text{ else } x i$ 
    have cm_ret: continuous_map (powertop_real UNIV) (powertop_real UNIV)
      (ret t) for t
      by (auto simp: ret_def continuous_map_componentwise_UNIV intro: con-

```

```

tinuous_map_product_projection)
  let ?ST = λt. subtopology (Euclidean_space (Suc n)) {x. x n = t}
  define squashable where
    squashable ≡ λt S. ∀ x t'. x ∈ S ∧ (x n ≤ t' ∧ t' ≤ t ∨ t ≤ t' ∧ t' ≤ x n)
  → ret t' x ∈ S
  have squashable: squashable t (topspace (Euclidean_space (Suc n))) for t
    by (simp add: squashable_def topspace_Euclidean_space ret_def)
  have squashableD: [squashable t S; x ∈ S; x n ≤ t' ∧ t' ≤ t ∨ t ≤ t' ∧ t' ≤ x
n] ⇒ ret t' x ∈ S for x t' t S
    by (auto simp: squashable_def)
  have squashable 1 hi
    by (force simp: squashable_def hi_def ret_def topspace_Euclidean_space
intro: cong)
  have squashable t UNIV for t
    by (force simp: squashable_def hi_def ret_def topspace_Euclidean_space
intro: cong)
  have squashable_0_lohi: squashable 0 (lo ∩ hi)
    using Ssub
  by (auto simp: squashable_def hi_def lo_def ret_def topspace_Euclidean_space
intro: cong)
  have rm_ret: retraction_maps (subtopology (Euclidean_space (Suc n)) U)
    (subtopology (Euclidean_space (Suc n)) {x. x ∈ U ∧ x
n = t})
    (ret t) id
  if squashable t U for t U
  unfolding retraction_maps_def
proof (intro conjI ballI)
  show continuous_map (subtopology (Euclidean_space (Suc n)) U)
    (subtopology (Euclidean_space (Suc n)) {x ∈ U. x n = t}) (ret t)
  apply (simp add: cm_ret continuous_map_in_subtopology continuous_map_from_subtopology
Euclidean_space_def)
  using that by (fastforce simp: squashable_def ret_def)
next
  show continuous_map (subtopology (Euclidean_space (Suc n)) {x ∈ U. x n
= t})
    (subtopology (Euclidean_space (Suc n)) U) id
  using continuous_map_in_subtopology by fastforce
  show ret t (id x) = x
  if x ∈ topspace (subtopology (Euclidean_space (Suc n)) {x ∈ U. x n = t})
for x
  using that by (simp add: topspace_Euclidean_space ret_def fun_eq_iff)
qed
  have cm_snd: continuous_map (prod_topology (top_of_set {0..1}) (subtopology
(powertop_real UNIV) S))
    euclideanreal (λx. snd x k) for k::nat and S
  using continuous_map_componentwise_UNIV continuous_map_into_fulltopology
continuous_map_snd by fastforce
  have cm_fstsnd: continuous_map (prod_topology (top_of_set {0..1}) (subtopology
(powertop_real UNIV) S))

```

```

    euclideanreal ( $\lambda x. \text{fst } x * \text{snd } x \ k$ ) for  $k::\text{nat}$  and  $S$ 
  by (intro continuous_intros continuous_map_into_fulltopology [OF continuous_mapfst] cm_snd)
  have hw_sub: homotopic_with ( $\lambda k. k \text{ ' } V \subseteq V$ ) (subtopology (Euclidean_space (Suc n)) U)
    (subtopology (Euclidean_space (Suc n)) U) (ret t) id
  if squashable t U squashable t V for U V t
  unfolding homotopic_with_def
  proof (intro exI conjI allI ballI)
    let ?h =  $\lambda(z,x). \text{ret } ((1 - z) * t + z * x \ n)$  x
    show ( $\lambda x. ?h \ (u, x)$ ) '  $V \subseteq V$  if  $u \in \{0..1\}$  for u
      using that
      by clarsimp (metis squashableD [OF ‹squashable t V›] convex_bound_le
        diff_ge_0_iff_ge eq_diff_eq' le_cases less_eq_real_def segment_bound_lemma)
    have 1: ?h ' ( $\{0..1\} \times (\{x. \forall i \geq \text{Suc } n. x \ i = 0\} \cap U)$ )  $\subseteq U$ 
      by clarsimp (metis squashableD [OF ‹squashable t U›] convex_bound_le
        diff_ge_0_iff_ge eq_diff_eq' le_cases less_eq_real_def segment_bound_lemma)
    show continuous_map (prod_topology (top_of_set  $\{0..1\}$ ) (subtopology
      (Euclidean_space (Suc n)) U))
      (subtopology (Euclidean_space (Suc n)) U) ?h
    apply (simp add: continuous_map_in_subtopology Euclidean_space_def
      subtopology_subtopology 1)
    apply (auto simp: case_prod_unfold ret_def continuous_map_componentwise_UNIV)
    apply (intro continuous_map_into_fulltopology [OF continuous_mapfst]
      cm_snd continuous_intros)
    by (auto simp: cm_snd)
  qed (auto simp: ret_def)
  have cs_hi: contractible_space (subtopology (Euclidean_space (Suc n)) hi)
  proof -
    have homotopic_with ( $\lambda x. \text{True}$ ) (?ST 1) (?ST 1) id ( $\lambda x. (\lambda i. \text{if } i = n \text{ then } 1 \text{ else } 0)$ )
      apply (subst homotopic_with_sym)
      apply (simp add: homotopic_with)
      apply (rule_tac x=( $\lambda(z,x). i. \text{if } i=n \text{ then } 1 \text{ else } z * x \ i$ ) in exI)
      apply (auto simp: Euclidean_space_def subtopology_subtopology continuous_map_in_subtopology
        case_prod_unfold continuous_map_componentwise_UNIV cmfstsnd)
    done
    then have contractible_space (?ST 1)
      unfolding contractible_space_def by metis
    moreover have ?thesis = contractible_space (?ST 1)
      proof (intro deformation_retract_imp_homotopy_equivalent_space homotopy_equivalent_space_contractibility)
        have  $\{x. \forall i \geq \text{Suc } n. x \ i = 0\} \cap \{x \in \text{hi}. x \ n = 1\} = \{x. \forall i \geq \text{Suc } n. x \ i = 0\} \cap \{x. x \ n = 1\}$ 
          by (auto simp: hi_def topspace_Euclidean_space)
        then have eq: subtopology (Euclidean_space (Suc n))  $\{x. x \in \text{hi} \wedge x \ n = 1\} = ?ST \ 1$ 
          by (simp add: Euclidean_space_def subtopology_subtopology)
      qed
  end

```

```

      show homotopic_with (λx. True) (subtopology (Euclidean_space (Suc n))
hi) (subtopology (Euclidean_space (Suc n)) hi) (ret 1) id
      using hw_sub [OF ‹squashable 1 hi› ‹squashable 1 UNIV›] eq by simp
      show retraction_maps (subtopology (Euclidean_space (Suc n)) hi) (?ST 1)
(ret 1) id
      using rm_ret [OF ‹squashable 1 hi›] eq by simp
    qed
  ultimately show ?thesis by metis
qed
have ?lhs ≅ relative_homology_group p (Euclidean_space (Suc n)) (lo ∩ hi)
proof (rule group.iso_sym [OF _ deformation_retract_imp_isomorphic_relative_homology_groups])
  have {x. ∀ i ≥ Suc n. x i = 0} ∩ {x. x n = 0} = {x. ∀ i ≥ n. x i = (0::real)}
  by auto (metis le_less_Suc_eq not_le)
  then have ?ST 0 = Euclidean_space n
  by (simp add: Euclidean_space_def subtopology_subtopology)
  then show retraction_maps (Euclidean_space (Suc n)) (Euclidean_space n)
(ret 0) id
  using rm_ret [OF ‹squashable 0 UNIV›] by auto
  then have ret 0 x ∈ topspace (Euclidean_space n)
  if x ∈ topspace (Euclidean_space (Suc n)) - 1 < x n x n < 1 for x
  using that by (metis continuous_map_image_subset_topspace image_subset_iff
retraction_maps_def)
  then show (ret 0) ‘ (lo ∩ hi) ⊆ topspace (Euclidean_space n) - S
  by (auto simp: local.cong ret_def hi_def lo_def)
  show homotopic_with (λh. h ‘ (lo ∩ hi) ⊆ lo ∩ hi) (Euclidean_space (Suc
n)) (Euclidean_space (Suc n)) (ret 0) id
  using hw_sub [OF squashable squashable_0_lohi] by simp
  qed (auto simp: lo_def hi_def Euclidean_space_def)
  also have ... ≅ relative_homology_group p (subtopology (Euclidean_space
(Suc n)) hi) (lo ∩ hi)
  proof (rule group.iso_sym [OF _ isomorphic_relative_homology_groups_inclusion_contractible])
    show contractible_space (subtopology (Euclidean_space (Suc n)) hi)
    by (simp add: cs_hi)
    show topspace (Euclidean_space (Suc n)) ∩ hi ≠ {}
    apply (simp add: hi_def topspace_Euclidean_space_set_eq_iff)
    apply (rule_tac x=λi. if i = n then 1 else 0 in exI, auto)
    done
  qed auto
  also have ... ≅ relative_homology_group p (subtopology (Euclidean_space
(Suc n)) (lo ∪ hi)) lo
  proof -
    have oo: openin (Euclidean_space (Suc n)) {x ∈ topspace (Euclidean_space
(Suc n)). x n ∈ A}
    if open A for A
    proof (rule openin_continuous_map_preimage)
      show continuous_map (Euclidean_space (Suc n)) euclideanreal (λx. x n)
      proof -
        have ∀ n f. continuous_map (product_topology f UNIV) (f (n::nat)) (λf.
f n::real)

```

```

    by (simp add: continuous_map_product_projection)
  then show ?thesis
    using Euclidean_space_def continuous_map_from_subtopology
    by (metis (mono_tags))
qed
qed (auto intro: that)
have openin (Euclidean_space (Suc n)) lo
  apply (simp add: openin_subopen [of _ lo])
  apply (simp add: lo_def, safe)
  apply (force intro: oo [of lessThan 0, simplified] open_Collect_less)
  apply (rule_tac x={x ∈ topspace (Euclidean_space (Suc n)). x n < 1}
    ∩ (topspace (Euclidean_space (Suc n)) - S) in exI)
using clo apply (force intro: oo [of lessThan 1, simplified] open_Collect_less)
done
moreover have openin (Euclidean_space (Suc n)) hi
  apply (simp add: openin_subopen [of _ hi])
  apply (simp add: hi_def, safe)
  apply (force intro: oo [of greaterThan 0, simplified] open_Collect_less)
  apply (rule_tac x={x ∈ topspace (Euclidean_space (Suc n)). x n > -1}
    ∩ (topspace (Euclidean_space (Suc n)) - S) in exI)
using clo apply (force intro: oo [of greaterThan (-1), simplified] open_Collect_less)
done
ultimately
have *: subtopology (Euclidean_space (Suc n)) (lo ∪ hi) closure_of
  (topspace (subtopology (Euclidean_space (Suc n)) (lo ∪ hi)) - hi)
  ⊆ subtopology (Euclidean_space (Suc n)) (lo ∪ hi) interior_of lo
  by (metis (no_types, lifting) Diff_idemp Diff_subset_conv Un_commute
    Un_upper2 closure_of_interior_of_interior_of_closure_of_interior_of_complement
    interior_of_eq lo_hi Un openin Un openin_open_subtopology topspace_subtopology_subset)
  have eq: ((lo ∪ hi) ∩ (lo ∪ hi - (topspace (Euclidean_space (Suc n)) ∩ (lo
    ∪ hi) - hi))) = hi
    (lo - (topspace (Euclidean_space (Suc n)) ∩ (lo ∪ hi) - hi)) = lo ∩ hi
  by (auto simp: lo_def hi_def Euclidean_space_def)
show ?thesis
  using homology_excision_axiom [OF *, of lo ∪ hi p]
  by (force simp: subtopology_subtopology eq is_iso_def)
qed
also have ... ≅ relative_homology_group (p + 1 - 1) (subtopology (Euclidean_space
(Suc n)) (lo ∪ hi)) lo
  by simp
also have ... ≅ relative_homology_group (p + 1) (Euclidean_space (Suc n))
(lo ∪ hi)
proof (rule group.iso_sym [OF isomorphic_relative_homology_groups_relboundary_contractible])
  have proj: continuous_map (powertop_real UNIV) euclideanreal (λf. f n)
    by (metis UNIV_I continuous_map_product_projection)
  have hilo: ∧x. x ∈ hi ⟹ (λi. if i = n then - x i else x i) ∈ lo
    ∧x. x ∈ lo ⟹ (λi. if i = n then - x i else x i) ∈ hi
    using local.cong
  by (auto simp: hi_def lo_def topspace_Euclidean_space split: if_split_asm)

```

```

have subtopology (Euclidean_space (Suc n)) hi homeomorphic_space subtopol-
ogy (Euclidean_space (Suc n)) lo
  unfolding homeomorphic_space_def
  apply (rule_tac x= $\lambda x i$ . if  $i = n$  then  $-(x i)$  else  $x i$  in exI)+
  using proj
  apply (auto simp: homeomorphic_maps_def Euclidean_space_def continu-
ous_map_in_subtopology
          hilo continuous_map_componentwise_UNIV continu-
ous_map_from_subtopology continuous_map_minus
          intro: continuous_map_from_subtopology continuous_map_product_projection)
  done
then have contractible_space(subtopology (Euclidean_space(Suc n)) hi)
   $\longleftrightarrow$  contractible_space (subtopology (Euclidean_space (Suc n)) lo)
  by (rule homeomorphic_space_contractibility)
then show contractible_space (subtopology (Euclidean_space (Suc n)) lo)
  using cs_hi by auto
show topspace (Euclidean_space (Suc n))  $\cap$  lo  $\neq \{\}$ 
  apply (simp add: lo_def Euclidean_space_def set_eq_iff)
  apply (rule_tac x= $\lambda i$ . if  $i = n$  then  $-1$  else  $0$  in exI, auto)
  done
qed auto
also have ...  $\cong$  ?rhs
  by (simp flip: lo_hi_Un)
finally show ?thesis .
qed
show ?thesis
proof (induction k)
  case (Suc m)
  with assms obtain T where cloT: closedin (powertop_real UNIV) T
    and SeqT:  $S = T \cap \{x. \forall i \geq n. x i = 0\}$ 
  by (auto simp: Euclidean_space_def closedin_subtopology)
  then have closedin (Euclidean_space (m + n)) S
  apply (simp add: Euclidean_space_def closedin_subtopology)
  apply (rule_tac x= $T \cap \text{topspace}(\text{Euclidean\_space } n)$  in exI)
  using closedin_Euclidean_space topspace_Euclidean_space by force
  moreover have relative_homology_group p (Euclidean_space n) (topspace
(Euclidean_space n) - S)
     $\cong$  relative_homology_group (p + 1) (Euclidean_space (Suc n))
(topspace (Euclidean_space (Suc n)) - S)
  if closedin (Euclidean_space n) S for p n
  proof -
    define S' where  $S' \equiv \{x \in \text{topspace}(\text{Euclidean\_space}(Suc\ n)). (\lambda i. \text{if } i < n$ 
then  $x\ i$  else  $0\}) \in S\}$ 
    have Ssub_n:  $S \subseteq \text{topspace}(\text{Euclidean\_space } n)$ 
    by (meson that closedin_def)
    have relative_homology_group p (Euclidean_space n) (topspace(Euclidean_space
n) - S')
       $\cong$  relative_homology_group (p + 1) (Euclidean_space (Suc n)) (topspace(Euclidean_space
(Suc n)) -  $\{x \in S'. x\ n = 0\}$ )

```

```

proof (rule *)
  have cm: continuous_map (powertop_real UNIV) euclideanreal ( $\lambda f. f u$ )
for u
    by (metis UNIV_I continuous_map_product_projection)
  have continuous_map (subtopology (powertop_real UNIV)  $\{x. \forall i > n. x i = 0\}$ ) euclideanreal
    ( $\lambda x. \text{if } k \leq n \text{ then } x k \text{ else } 0$ ) for k
    by (simp add: continuous_map_from_subtopology [OF cm])
  moreover have  $\forall i \geq n. (\text{if } i < n \text{ then } x i \text{ else } 0) = 0$ 
    if  $x \in \text{topspace (subtopology (powertop_real UNIV) } \{x. \forall i > n. x i = 0\})$ 
for x
    using that by simp
  ultimately have continuous_map (Euclidean_space (Suc n)) (Euclidean_space n) ( $\lambda x i. \text{if } i < n \text{ then } x i \text{ else } 0$ )
    by (simp add: Euclidean_space_def continuous_map_in_subtopology
      continuous_map_componentwise_UNIV
      continuous_map_from_subtopology [OF cm] image_subset_iff)
  then show closedin (Euclidean_space (Suc n)) S'
  unfolding S'_def using that by (rule closedin_continuous_map_preimage)
next
  fix x y
  assume xy:  $\bigwedge i. i \neq n \implies x i = y i \text{ } x \in S'$ 
  then have ( $\lambda i. \text{if } i < n \text{ then } x i \text{ else } 0$ ) = ( $\lambda i. \text{if } i < n \text{ then } y i \text{ else } 0$ )
    by (simp add: S'_def Euclidean_space_def fun_eq_iff)
  with xy show  $y \in S'$ 
    by (simp add: S'_def Euclidean_space_def)
qed
moreover
  have abs_eq: ( $\lambda i. \text{if } i < n \text{ then } x i \text{ else } 0$ ) = x if  $\bigwedge i. i \geq n \implies x i = 0$  for
     $x :: \text{nat} \Rightarrow \text{real}$  and n
    using that by auto
  then have  $\text{topspace (Euclidean\_space } n) - S' = \text{topspace (Euclidean\_space } n) - S$ 
    by (simp add: S'_def Euclidean_space_def set_eq_iff cong: conj_cong)
  moreover
  have  $\text{topspace (Euclidean\_space (Suc } n)) - \{x \in S'. x n = 0\} = \text{topspace (Euclidean\_space (Suc } n)) - S$ 
    using Ssub_n
  apply (auto simp: S'_def subset_iff Euclidean_space_def set_eq_iff abs_eq
    cong: conj_cong)
    by (metis abs_eq le_antisym not_less_eq_eq)
  ultimately show ?thesis
    by simp
qed
ultimately have relative_homology_group (p + m)(Euclidean_space (m + n))(topspace (Euclidean_space (m + n)) - S)
   $\cong$  relative_homology_group (p + m + 1) (Euclidean_space (Suc (m + n)))(topspace (Euclidean_space (Suc (m + n))) - S)
    by (metis  $\langle \text{closedin (Euclidean\_space (m + n)) } S \rangle$ )

```

```

    then show ?case
      using Suc.IH iso_trans by (force simp: algebra_simps)
    qed (simp add: iso_refl)
  qed

lemma iso_Euclidean_complements_lemma1:
  assumes S: closedin (Euclidean_space m) S and cmf: continuous_map(subtopology
    (Euclidean_space m) S) (Euclidean_space n) f
  obtains g where continuous_map (Euclidean_space m) (Euclidean_space n) g
     $\wedge x. x \in S \implies g\ x = f\ x$ 
proof -
  have cont: continuous_on (topspace (Euclidean_space m)  $\cap$  S) ( $\lambda x. f\ x$ ) for i
    by (metis (no_types) continuous_on_product_then_coordinatewise
      cm_Euclidean_space_iff_continuous_on cmf topspace_subtopology)
  have f' : (topspace (Euclidean_space m)  $\cap$  S)  $\subseteq$  topspace (Euclidean_space n)
    using cmf continuous_map_image_subset_topspace by fastforce
  then
    have  $\exists g. \text{continuous\_on } (\text{topspace } (\text{Euclidean\_space } m))\ g \wedge (\forall x \in S. g\ x = f\ x)$ 
  x i for i
    using S Tietze_unbounded [OF cont [of i]]
    by (metis closedin_Euclidean_space_iff closedin_closed_Int topspace_subtopology
      topspace_subtopology_subset)
  then obtain g where cmg:  $\wedge i. \text{continuous\_map } (\text{Euclidean\_space } m)\ \text{euclidean\_real } (g\ i)$ 
    and gf:  $\wedge i\ x. x \in S \implies g\ i\ x = f\ x\ i$ 
    unfolding continuous_map_Euclidean_space_iff by metis
  let ?GG =  $\lambda x\ i. \text{if } i < n \text{ then } g\ i\ x \text{ else } 0$ 
  show thesis
  proof
    show continuous_map (Euclidean_space m) (Euclidean_space n) ?GG
      unfolding Euclidean_space_def [of n]
      by (auto simp: continuous_map_in_subtopology continuous_map_componentwise
        cmg)
    show ?GG x = f x if x  $\in$  S for x
    proof -
      have S  $\subseteq$  topspace (Euclidean_space m)
        by (meson S closedin_def)
      then have f x  $\in$  topspace (Euclidean_space n)
        using cmf that unfolding continuous_map_def topspace_subtopology by
        blast
      then show ?thesis
        by (force simp: topspace_Euclidean_space gf that)
    qed
  qed
qed

```

```

lemma iso_Euclidean_complements_lemma2:
  assumes S: closedin (Euclidean_space m) S

```



```

    and T: closedin (Euclidean_space n) T
    and hom: homeomorphic_map (subtopology (Euclidean_space m) S) (subtopology
(Euclidean_space n) T) f
    obtains g where homeomorphic_map (prod_topology (Euclidean_space m) (Euclidean_space
n))
                                (prod_topology (Euclidean_space n) (Euclidean_space
m)) g
                                 $\bigwedge x. x \in S \implies g(x, (\lambda i. 0)) = (f x, (\lambda i. 0))$ 
  proof -
    obtain g where cmf: continuous_map (subtopology (Euclidean_space m) S)
(subtopology (Euclidean_space n) T) f
    and cmg: continuous_map (subtopology (Euclidean_space n) T) (subtopology
(Euclidean_space m) S) g
    and gf:  $\bigwedge x. x \in S \implies g(f x) = x$ 
    and fg:  $\bigwedge y. y \in T \implies f(g y) = y$ 
    using hom S T closedin_subset unfolding homeomorphic_map_maps home-
omorphic_maps_def
    by fastforce
    obtain f' where cmf': continuous_map (Euclidean_space m) (Euclidean_space
n) f'
    and f'f:  $\bigwedge x. x \in S \implies f' x = f x$ 
    using iso_Euclidean_complements_lemma1 S cmf continuous_map_into_fulltopology
by metis
    obtain g' where cmg': continuous_map (Euclidean_space n) (Euclidean_space
m) g'
    and g'g:  $\bigwedge x. x \in T \implies g' x = g x$ 
    using iso_Euclidean_complements_lemma1 T cmg continuous_map_into_fulltopology
by metis
    define p where p  $\equiv \lambda(x,y). (x, (\lambda i. y i + f' x i))$ 
    define p' where p'  $\equiv \lambda(x,y). (x, (\lambda i. y i - f' x i))$ 
    define q where q  $\equiv \lambda(x,y). (x, (\lambda i. y i + g' x i))$ 
    define q' where q'  $\equiv \lambda(x,y). (x, (\lambda i. y i - g' x i))$ 
    have homeomorphic_maps (prod_topology (Euclidean_space m) (Euclidean_space
n))
                                (prod_topology (Euclidean_space m) (Euclidean_space n))
                                p p'
                                homeomorphic_maps (prod_topology (Euclidean_space n) (Euclidean_space
m))
                                (prod_topology (Euclidean_space n) (Euclidean_space m))
                                q q'
                                homeomorphic_maps (prod_topology (Euclidean_space m) (Euclidean_space
n))
                                (prod_topology (Euclidean_space n) (Euclidean_space m))
                                ( $\lambda(x,y). (y,x)$ ) ( $\lambda(x,y). (y,x)$ )
    apply (simp_all add: p_def p'_def q_def q'_def homeomorphic_maps_def
continuous_map_pairwise)
    apply (force simp: case_prod_unfold continuous_map_of_fst [unfolded o_def]
cmf' cmg' intro: continuous_intros)+
    done

```

then have *homeomorphic_maps* (*prod_topology* (*Euclidean_space* *n*) (*Euclidean_space* *n*))

(*prod_topology* (*Euclidean_space* *n*) (*Euclidean_space* *m*))
 (*q'* \circ ($\lambda(x,y). (y,x)$) \circ *p*) (*p'* \circ ($(\lambda(x,y). (y,x)) \circ$ *q*))

using *homeomorphic_maps_compose* *homeomorphic_maps_sym* **by** (*metis* (*no_types*, *lifting*))

moreover

have $\bigwedge x. x \in S \implies (q' \circ (\lambda(x,y). (y,x)) \circ p) (x, \lambda i. 0) = (f\ x, \lambda i. 0)$

apply (*simp* *add*: *q'_def* *p_def* *f'f*)

apply (*simp* *add*: *fun_eq_iff*)

by (*metis* *S T closedin_subset* *g'g gf hom* *homeomorphic_imp_surjective_map* *image_eqI* *topspace_subtopology_subset*)

ultimately

show *thesis*

using *homeomorphic_map_maps* **that** **by** *blast*

qed

proposition *isomorphic_relative_homology_groups_Euclidean_complements*:

assumes *S*: *closedin* (*Euclidean_space* *n*) *S* **and** *T*: *closedin* (*Euclidean_space* *n*) *T*

and *hom*: (*subtopology* (*Euclidean_space* *n*) *S*) *homeomorphic_space* (*subtopology* (*Euclidean_space* *n*) *T*)

shows *relative_homology_group* *p* (*Euclidean_space* *n*) (*topspace*(*Euclidean_space* *n*) - *S*)

\cong *relative_homology_group* *p* (*Euclidean_space* *n*) (*topspace*(*Euclidean_space* *n*) - *T*)

proof -

have *subST*: *S* \subseteq *topspace*(*Euclidean_space* *n*) *T* \subseteq *topspace*(*Euclidean_space* *n*)

by (*meson* *S T closedin_def*)+

have *relative_homology_group* *p* (*Euclidean_space* *n*) (*topspace* (*Euclidean_space* *n*) - *S*)

\cong *relative_homology_group* (*p* + *int* *n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) - *S*)

using *relative_homology_group_Euclidean_complement_step* [*OF S*] **by** *blast*

moreover have *relative_homology_group* *p* (*Euclidean_space* *n*) (*topspace* (*Euclidean_space* *n*) - *T*)

\cong *relative_homology_group* (*p* + *int* *n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) - *T*)

using *relative_homology_group_Euclidean_complement_step* [*OF T*] **by** *blast*

moreover have *relative_homology_group* (*p* + *int* *n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) - *S*)

\cong *relative_homology_group* (*p* + *int* *n*) (*Euclidean_space* (*n* + *n*)) (*topspace* (*Euclidean_space* (*n* + *n*)) - *T*)

proof -

obtain *f* **where** *f*: *homeomorphic_map* (*subtopology* (*Euclidean_space* *n*) *S*)
 (*subtopology* (*Euclidean_space* *n*) *T*) *f*

using *hom_unfolding* *homeomorphic_space* **by** *blast*

```

obtain  $g$  where  $g$ : homeomorphic_map (prod_topology (Euclidean_space  $n$ )
(Euclidean_space  $n$ ))
(prod_topology (Euclidean_space  $n$ ) (Euclidean_space
 $n$ ))  $g$ 
and  $gf$ :  $\bigwedge x. x \in S \implies g(x, (\lambda i. 0)) = (f\ x, (\lambda i. 0))$ 
using S T f iso_Euclidean_complements_lemma2 by blast
define  $h$  where  $h \equiv \lambda x::nat \Rightarrow real. ((\lambda i. \text{if } i < n \text{ then } x\ i \text{ else } 0), (\lambda j. \text{if } j < n \text{ then } x(n + j) \text{ else } 0))$ 
define  $k$  where  $k \equiv \lambda(x,y) i. \text{if } i < 2 * n \text{ then if } i < n \text{ then } x\ i \text{ else } y(i - n) \text{ else } (0::real)$ 
have  $hk$ : homeomorphic_maps (Euclidean_space( $2 * n$ )) (prod_topology (Euclidean_space
 $n$ ) (Euclidean_space  $n$ ))  $h\ k$ 
unfolding homeomorphic_maps_def
proof safe
show continuous_map (Euclidean_space ( $2 * n$ ))
(prod_topology (Euclidean_space  $n$ ) (Euclidean_space  $n$ ))  $h$ 
apply (simp add: h_def continuous_map_pairwise o_def continuous_map_componentwise_Euclidean_space)
unfolding Euclidean_space_def
by (metis (mono_tags) UNIV_I continuous_map_from_subtopology continuous_map_product_projection)
have continuous_map (prod_topology (Euclidean_space  $n$ ) (Euclidean_space
 $n$ )) eclideanreal ( $\lambda p. \text{fst } p\ i$ ) for  $i$ 
using Euclidean_space_def continuous_map_into_fulltopology continuous_map_fst by fastforce
moreover
have continuous_map (prod_topology (Euclidean_space  $n$ ) (Euclidean_space
 $n$ )) eclideanreal ( $\lambda p. \text{snd } p\ (i - n)$ ) for  $i$ 
using Euclidean_space_def continuous_map_into_fulltopology continuous_map_snd by fastforce
ultimately
show continuous_map (prod_topology (Euclidean_space  $n$ ) (Euclidean_space
 $n$ ))
(Euclidean_space ( $2 * n$ ))  $k$ 
by (simp add: k_def continuous_map_pairwise o_def continuous_map_componentwise_Euclidean_space case_prod_unfold)
qed (auto simp: k_def h_def fun_eq_iff topspace_Euclidean_space)
define  $kgh$  where  $kgh \equiv k \circ g \circ h$ 
let  $?i = \text{hom\_induced } (p + n) \text{ (Euclidean\_space}(2 * n)) \text{ (topspace(Euclidean\_space}(2 * n)) - S)$ 
(Euclidean_space( $2 * n$ )) (topspace(Euclidean_space( $2 * n$ )) -  $T$ )  $kgh$ 
have  $?i \in \text{iso}$  (relative_homology_group ( $p + \text{int } n$ ) (Euclidean_space ( $2 * n$ ))
(topspace (Euclidean_space ( $2 * n$ )) -  $S$ ))
(relative_homology_group ( $p + \text{int } n$ ) (Euclidean_space ( $2 * n$ ))
(topspace (Euclidean_space ( $2 * n$ )) -  $T$ ))
proof (rule homeomorphic_map_relative_homology_iso)
show  $hm$ : homeomorphic_map (Euclidean_space ( $2 * n$ )) (Euclidean_space
( $2 * n$ ))  $kgh$ 
unfolding kgh_def by (meson  $hk\ g\ \text{homeomorphic\_map\_maps homeomor-$ 

```

```

phic_maps_compose homeomorphic_maps_sym)
  have Teq:  $T = f \circ S$ 
  using f homeomorphic_imp_surjective_map subST(1) subST(2) topspace_subtopology_subset
by blast
  have khf:  $\bigwedge x. x \in S \implies k(h(f x)) = f x$ 
  by (metis (no_types, lifting) Teq hk homeomorphic_maps_def image_subset_iff
le_add1 mult_2 subST(2) subsetD subset_Euclidean_space)
  have gh:  $g(h x) = h(f x)$  if  $x \in S$  for  $x$ 
  proof -
    have [simp]:  $(\lambda i. \text{if } i < n \text{ then } x \text{ else } 0) = x$ 
    using subST(1) that topspace_Euclidean_space by (auto simp: fun_eq_iff)
    have  $f x \in \text{topspace}(\text{Euclidean\_space } n)$ 
    using Teq subST(2) that by blast
    moreover have  $(\lambda j. \text{if } j < n \text{ then } x (n + j) \text{ else } 0) = (\lambda j. 0::\text{real})$ 
    using Euclidean_space_def subST(1) that by force
    ultimately show ?thesis
    by (simp add: topspace_Euclidean_space h_def gf ⟨ $x \in S$ ⟩ fun_eq_iff)
  qed
  have *:  $\llbracket S \subseteq U; T \subseteq U; kgh \circ U = U; \text{inj\_on } kgh \text{ } U; kgh \circ S = T \rrbracket \implies kgh$ 
  ‘  $(U - S) = U - T$  for  $U$ 
  unfolding inj_on_def set_eq_iff by blast
  show  $kgh \circ (\text{topspace}(\text{Euclidean\_space } (2 * n)) - S) = \text{topspace}(\text{Euclidean\_space } (2 * n)) - T$ 
  proof (rule *)
    show  $kgh \circ \text{topspace}(\text{Euclidean\_space } (2 * n)) = \text{topspace}(\text{Euclidean\_space } (2 * n))$ 
    by (simp add: hm homeomorphic_imp_surjective_map)
    show  $\text{inj\_on } kgh (\text{topspace}(\text{Euclidean\_space } (2 * n)))$ 
    using hm homeomorphic_map_def by auto
    show  $kgh \circ S = T$ 
    by (simp add: Teq kgh_def gh khf)
  qed (use subST topspace_Euclidean_space in ⟨fastforce+⟩)
qed auto
then show ?thesis
  by (simp add: is_isoI mult_2)
qed
ultimately show ?thesis
  by (meson group.iso_sym iso_trans group_relative_homology_group)
qed

lemma lemma_iod:
  assumes  $S \subseteq T$   $S \neq \{\}$  and Tsub:  $T \subseteq \text{topspace}(\text{Euclidean\_space } n)$ 
  and S:  $\bigwedge a \ b \ u. \llbracket a \in S; b \in T; 0 < u; u < 1 \rrbracket \implies (\lambda i. (1 - u) * a + u * b \ i) \in S$ 
  shows path_connectedin  $(\text{Euclidean\_space } n) \ T$ 
  proof -
    obtain  $a$  where  $a \in S$ 
    using assms by blast
    have path_component_of  $(\text{subtopology}(\text{Euclidean\_space } n) \ T) \ a \ b$  if  $b \in T$  for

```

```

b
  unfolding path_component_of_def
proof (intro exI conjI)
  have [simp]:  $\forall i \geq n. a\ i = 0$ 
    using Tsub  $\langle a \in S \rangle$  assms(1) topspace_Euclidean_space by auto
  have [simp]:  $\forall i \geq n. b\ i = 0$ 
    using Tsub that topspace_Euclidean_space by auto
  have inT:  $(\lambda i. (1 - x) * a\ i + x * b\ i) \in T$  if  $0 \leq x \leq 1$  for  $x$ 
  proof (cases  $x = 0 \vee x = 1$ )
    case True
    with  $\langle a \in S \rangle \langle b \in T \rangle \langle S \subseteq T \rangle$  show ?thesis
    by force
  next
    case False
    then show ?thesis
    using subsetD [OF  $\langle S \subseteq T \rangle S$ ]  $\langle a \in S \rangle \langle b \in T \rangle$  that by auto
  qed
  have continuous_on  $\{0..1\}$   $(\lambda x. (1 - x) * a\ k + x * b\ k)$  for  $k$ 
    by (intro continuous_intros)
  then show pathin (subtopology (Euclidean_space n) T)  $(\lambda t\ i. (1 - t) * a\ i +$ 
 $t * b\ i)$ 
    apply (simp add: Euclidean_space_def subtopology_subtopology pathin_subtopology)
    apply (simp add: pathin_def continuous_map_componentwise_UNIV inT)
    done
  qed auto
  then have path_connected_space (subtopology (Euclidean_space n) T)
    by (metis Tsub path_component_of_equiv path_connected_space_iff_path_component
topspace_subtopology_subset)
  then show ?thesis
    by (simp add: Tsub path_connectedin_def)
qed

```

lemma invariance_of_dimension_closedin_Euclidean_space:

```

  assumes closedin (Euclidean_space n) S
  shows subtopology (Euclidean_space n) S homeomorphic_space Euclidean_space
n
 $\longleftrightarrow S = \text{topspace}(\text{Euclidean\_space } n)$ 
(is ?lhs = ?rhs)

```

proof

```

  assume L: ?lhs
  have Ssub:  $S \subseteq \text{topspace}(\text{Euclidean\_space } n)$ 
    by (meson assms closedin_def)
  moreover have False if  $a \notin S$  and  $a \in \text{topspace}(\text{Euclidean\_space } n)$  for  $a$ 
  proof -
    have cl_n: closedin (Euclidean_space (Suc n)) (topspace(Euclidean_space n))
      using Euclidean_space_def closedin_Euclidean_space closedin_subtopology
    by fastforce
    then have sub: subtopology (Euclidean_space(Suc n)) (topspace(Euclidean_space

```

```

n)) = Euclidean_space n
  by (metis (no_types, lifting) Euclidean_space_def closedin_subset subtopol-
ogy_subtopology tospace_Euclidean_space tospace_subtopology tospace_subtopology_subset)
  then have cl_S: closedin (Euclidean_space (Suc n)) S
    using cl_n assms closedin_closed_subtopology by fastforce
  have sub_SucS: subtopology (Euclidean_space (Suc n)) S = subtopology (Euclidean_space
n) S
  by (metis Ssub sub subtopology_subtopology tospace_subtopology tospace_subtopology_subset)
  have non0: {y.  $\exists x::\text{nat} \Rightarrow \text{real}. (\forall i \geq \text{Suc } n. x\ i = 0) \wedge (\exists i \geq n. x\ i \neq 0) \wedge y =$ 
x n} = -{0}
  proof safe
    show False if  $\forall i \geq \text{Suc } n. f\ i = 0 \ 0 = f\ n \ n \leq i \ f\ i \neq 0$  for  $f::\text{nat} \Rightarrow \text{real}$  and  $i$ 
      by (metis that le_antisym not_less_eq_eq)
    show  $\exists f::\text{nat} \Rightarrow \text{real}. (\forall i \geq \text{Suc } n. f\ i = 0) \wedge (\exists i \geq n. f\ i \neq 0) \wedge a = f\ n$  if  $a$ 
 $\neq 0$  for  $a$ 
      by (rule_tac x=( $\lambda i. 0$ )( $n := a$ ) in exI) (force simp: that)
  qed
  have homology_group 0 (subtopology (Euclidean_space (Suc n)) (topspace
(Euclidean_space (Suc n)) - S))
     $\cong$  homology_group 0 (subtopology (Euclidean_space (Suc n)) (topspace
(Euclidean_space (Suc n)) - topspace (Euclidean_space n)))
  proof (rule isomorphic_relative_contractible_space_imp_homology_groups)
    show (topspace (Euclidean_space (Suc n)) - S = {}) =
      (topspace (Euclidean_space (Suc n)) - topspace (Euclidean_space n) =
{})
    using cl_n closedin_subset that by auto
  next
    fix p
    show relative_homology_group p (Euclidean_space (Suc n))
      (topspace (Euclidean_space (Suc n)) - S)  $\cong$ 
      relative_homology_group p (Euclidean_space (Suc n))
      (topspace (Euclidean_space (Suc n)) - topspace (Euclidean_space n))
    by (simp add: L sub_SucS cl_S cl_n isomorphic_relative_homology_groups_Euclidean_complemen
sub)
  qed (auto simp: L)
  moreover
    have continuous_map (powertop_real UNIV) euclideanreal ( $\lambda x. x\ n$ )
      by (metis (no_types) UNIV_I continuous_map_product_projection)
    then have cm: continuous_map (subtopology (Euclidean_space (Suc n)) (topspace
(Euclidean_space (Suc n)) - topspace (Euclidean_space n)))
      euclideanreal ( $\lambda x. x\ n$ )
      by (simp add: Euclidean_space_def continuous_map_from_subtopology)
    have False if path_connected_space
      (subtopology (Euclidean_space (Suc n))
      (topspace (Euclidean_space (Suc n)) - topspace (Euclidean_space
n)))
    using path_connectedin_continuous_map_image [OF cm that [unfolded
path_connectedin_tospace [symmetric]]]
      bounded_path_connected_Compl_real [of {0}]

```

```

    by (simp add: topspace_Euclidean_space image_def Bex_def non0 flip:
path_connectedin_topspace)
  moreover
  have eq:  $T = T \cap \{x. x \ n \leq 0\} \cup T \cap \{x. x \ n \geq 0\}$  for  $T :: (\text{nat} \Rightarrow \text{real}) \text{ set}$ 
    by auto
  have path_connectedin (Euclidean_space (Suc n)) (topspace (Euclidean_space
(Suc n)) - S)
  proof (subst eq, rule path_connectedin_Un)
    have topspace(Euclidean_space(Suc n))  $\cap \{x. x \ n = 0\} = \text{topspace}(\text{Euclidean\_space}$ 
n)
    apply (auto simp: topspace_Euclidean_space)
    by (metis Suc_leI inf.absorb_iff2 inf.orderE leI)
  let ?S = topspace(Euclidean_space(Suc n))  $\cap \{x. x \ n < 0\}$ 
  show path_connectedin (Euclidean_space (Suc n))
    ((topspace (Euclidean_space (Suc n)) - S)  $\cap \{x. x \ n \leq 0\}$ )
  proof (rule lemma_iod)
    show ?S  $\subseteq (\text{topspace}(\text{Euclidean\_space}(\text{Suc } n)) - S) \cap \{x. x \ n \leq 0\}$ 
      using Ssub topspace_Euclidean_space by auto
    show ?S  $\neq \{\}$ 
      apply (simp add: topspace_Euclidean_space set_eq_iff)
      apply (rule_tac x=( $\lambda i. 0$ )( $n := -1$ ) in exI)
      apply auto
      done
    fix a b and u::real
    assume
      a  $\in ?S$  0 < u u < 1
      b  $\in (\text{topspace}(\text{Euclidean\_space}(\text{Suc } n)) - S) \cap \{x. x \ n \leq 0\}$ 
    then show ( $\lambda i. (1 - u) * a \ i + u * b \ i$ )  $\in ?S$ 
      by (simp add: topspace_Euclidean_space add_neg_nonpos less_eq_real_def
mult_less_0_iff)
    qed (simp add: topspace_Euclidean_space subset_iff)
  let ?T = topspace(Euclidean_space(Suc n))  $\cap \{x. x \ n > 0\}$ 
  show path_connectedin (Euclidean_space (Suc n))
    ((topspace (Euclidean_space (Suc n)) - S)  $\cap \{x. 0 \leq x \ n\}$ )
  proof (rule lemma_iod)
    show ?T  $\subseteq (\text{topspace}(\text{Euclidean\_space}(\text{Suc } n)) - S) \cap \{x. 0 \leq x \ n\}$ 
      using Ssub topspace_Euclidean_space by auto
    show ?T  $\neq \{\}$ 
      apply (simp add: topspace_Euclidean_space set_eq_iff)
      apply (rule_tac x=( $\lambda i. 0$ )( $n := 1$ ) in exI)
      apply auto
      done
    fix a b and u::real
    assume a  $\in ?T$  0 < u u < 1 b  $\in (\text{topspace}(\text{Euclidean\_space}(\text{Suc } n)) -$ 
S)  $\cap \{x. 0 \leq x \ n\}$ 
    then show ( $\lambda i. (1 - u) * a \ i + u * b \ i$ )  $\in ?T$ 
      by (simp add: topspace_Euclidean_space add_pos_nonneg)
    qed (simp add: topspace_Euclidean_space subset_iff)
  show (topspace (Euclidean_space (Suc n)) - S)  $\cap \{x. x \ n \leq 0\} \cap$ 

```

```

      ((topspace (Euclidean_space (Suc n)) - S) ∩ {x. 0 ≤ x n}) ≠ {}
    using that
    apply (auto simp: Set.set_eq_iff topspace_Euclidean_space)
    by (metis Suc_leD order_refl)
  qed
  then have path_connected_space (subtopology (Euclidean_space (Suc n))
    (topspace (Euclidean_space (Suc n)) - S))
  apply (simp add: path_connectedin_subtopology_flip: path_connectedin_topspace)
  by (metis Int_Diff inf_idem)
  ultimately
  show ?thesis
    using isomorphic_homology_imp_path_connectedness by blast
  qed
  ultimately show ?rhs
    by blast
  qed (simp add: homeomorphic_space_refl)

```

lemma *isomorphic_homology_groups_Euclidean_complements:*

```

  assumes closedin (Euclidean_space n) S closedin (Euclidean_space n) T
    (subtopology (Euclidean_space n) S) homeomorphic_space (subtopology
    (Euclidean_space n) T)
  shows homology_group p (subtopology (Euclidean_space n) (topspace (Euclidean_space
    n) - S))
    ≅ homology_group p (subtopology (Euclidean_space n) (topspace (Euclidean_space
    n) - T))
  proof (rule isomorphic_relative_contractible_space_imp_homology_groups)
    show topspace (Euclidean_space n) - S ⊆ topspace (Euclidean_space n)
    using assms homeomorphic_space_sym invariance_of_dimension_closedin_Euclidean_space
    subtopology_superset by fastforce
    show topspace (Euclidean_space n) - T ⊆ topspace (Euclidean_space n)
    using assms invariance_of_dimension_closedin_Euclidean_space subtopol-
    ogy_superset by force
    show (topspace (Euclidean_space n) - S = {}) = (topspace (Euclidean_space
    n) - T = {})
    by (metis Diff_eq_empty_iff assms closedin_subset homeomorphic_space_sym
    invariance_of_dimension_closedin_Euclidean_space subset_antisym subtopology_topspace)
    show relative_homology_group p (Euclidean_space n) (topspace (Euclidean_space
    n) - S) ≅
      relative_homology_group p (Euclidean_space n) (topspace (Euclidean_space
    n) - T) for p
    using assms isomorphic_relative_homology_groups_Euclidean_complements
  by blast
  qed auto

```

lemma *eqpoll_path_components_Euclidean_complements:*

```

  assumes closedin (Euclidean_space n) S closedin (Euclidean_space n) T
    (subtopology (Euclidean_space n) S) homeomorphic_space (subtopology

```


$(Euclidean_space\ n)\ T)$
shows $path_components_of$
 $(subtopology\ (Euclidean_space\ n)$
 $(topspace(Euclidean_space\ n) - S))$
 $\approx path_components_of$
 $(subtopology\ (Euclidean_space\ n)$
 $(topspace(Euclidean_space\ n) - T))$
by (*simp add: assms isomorphic_homology_groups_Euclidean_complements isomorphic_homology_imp_path_components*)

lemma *path_connectedin_Euclidean_complements:*

assumes $closedin\ (Euclidean_space\ n)\ S\ closedin\ (Euclidean_space\ n)\ T$
 $(subtopology\ (Euclidean_space\ n)\ S)\ homeomorphic_space\ (subtopology$
 $(Euclidean_space\ n)\ T)$
shows $path_connectedin\ (Euclidean_space\ n)\ (topspace(Euclidean_space\ n) - S)$
 $\longleftrightarrow path_connectedin\ (Euclidean_space\ n)\ (topspace(Euclidean_space\ n)$
 $- T)$
by (*meson Diff_subset assms isomorphic_homology_groups_Euclidean_complements isomorphic_homology_imp_path_connectedness path_connectedin_def*)

lemma *eqpoll_connected_components_Euclidean_complements:*

assumes $S: closedin\ (Euclidean_space\ n)\ S$ **and** $T: closedin\ (Euclidean_space$
 $n)\ T$
and $ST: (subtopology\ (Euclidean_space\ n)\ S)\ homeomorphic_space\ (subtopology$
 $(Euclidean_space\ n)\ T)$
shows $connected_components_of$
 $(subtopology\ (Euclidean_space\ n)$
 $(topspace(Euclidean_space\ n) - S))$
 $\approx connected_components_of$
 $(subtopology\ (Euclidean_space\ n)$
 $(topspace(Euclidean_space\ n) - T))$
using *eqpoll_path_components_Euclidean_complements [OF assms]*
by (*metis S T closedin_def locally_path_connected_Euclidean_space locally_path_connected_space_open_subset path_components_eq_connected_components_of*)

lemma *connected_in_Euclidean_complements:*

assumes $closedin\ (Euclidean_space\ n)\ S\ closedin\ (Euclidean_space\ n)\ T$
 $(subtopology\ (Euclidean_space\ n)\ S)\ homeomorphic_space\ (subtopology$
 $(Euclidean_space\ n)\ T)$
shows $connectedin\ (Euclidean_space\ n)\ (topspace(Euclidean_space\ n) - S)$
 $\longleftrightarrow connectedin\ (Euclidean_space\ n)\ (topspace(Euclidean_space\ n) - T)$
apply (*simp add: connectedin_def connected_space_iff_components_subset_singleton subset_singleton_iff_lepoll*)
using *eqpoll_connected_components_Euclidean_complements [OF assms]*
by (*meson eqpoll_sym lepoll_trans1*)

theorem *invariance_of_dimension_Euclidean_space:*

$Euclidean_space\ m\ homeomorphic_space\ Euclidean_space\ n \longleftrightarrow m = n$
proof (cases m n rule: linorder_cases)
 case less
 then have *: $topspace\ (Euclidean_space\ m) \subseteq topspace\ (Euclidean_space\ n)$
 by (meson le_cases not_le subset_Euclidean_space)
 then have $Euclidean_space\ m = subtopology\ (Euclidean_space\ n)\ (topspace(Euclidean_space\ m))$
 by (simp add: Euclidean_space_def inf.absorb_iff2 subtopology_subtopology)
 then show ?thesis
 by (metis (no_types, lifting) * Euclidean_space_def closedin_Euclidean_space
 closedin_closed_subtopology eq_iff invariance_of_dimension_closedin_Euclidean_space
 subset_Euclidean_space topspace_Euclidean_space)
 next
 case equal
 then show ?thesis
 by (simp add: homeomorphic_space_refl)
 next
 case greater
 then have *: $topspace\ (Euclidean_space\ n) \subseteq topspace\ (Euclidean_space\ m)$
 by (meson le_cases not_le subset_Euclidean_space)
 then have $Euclidean_space\ n = subtopology\ (Euclidean_space\ m)\ (topspace(Euclidean_space\ n))$
 by (simp add: Euclidean_space_def inf.absorb_iff2 subtopology_subtopology)
 then show ?thesis
 by (metis (no_types, lifting) * Euclidean_space_def closedin_Euclidean_space
 closedin_closed_subtopology eq_iff homeomorphic_space_sym invariance_of_dimension_closedin_Euclidean_space
 subset_Euclidean_space topspace_Euclidean_space)
 qed

lemma biglemma:

assumes $n \neq 0$ **and** $S: compactin\ (Euclidean_space\ n)\ S$
 and $cmh: continuous_map\ (subtopology\ (Euclidean_space\ n)\ S)\ (Euclidean_space\ n)\ h$
 and $inj_on\ h\ S$
 shows $path_connectedin\ (Euclidean_space\ n)\ (topspace(Euclidean_space\ n) - h\ 'S)$
 $\longleftrightarrow path_connectedin\ (Euclidean_space\ n)\ (topspace(Euclidean_space\ n) - S)$
proof (rule path_connectedin_Euclidean_complements)
 have $hS_sub: h\ 'S \subseteq topspace(Euclidean_space\ n)$
 by (metis (no_types) S cmh compactin_subspace continuous_map_image_subset_topspace
 topspace_subtopology_subset)
 show $clo_S: closedin\ (Euclidean_space\ n)\ S$
 using assms **by** (simp add: continuous_map_in_subtopology Hausdorff_Euclidean_space
 compactin_imp_closedin)
 show $clo_hS: closedin\ (Euclidean_space\ n)\ (h\ 'S)$
 using Hausdorff_Euclidean_space S cmh compactin_absolute compactin_imp_closedin

```

image_compactin by blast
have homeomorphic_map (subtopology (Euclidean_space n) S) (subtopology (Euclidean_space
n) (h ` S)) h
proof (rule continuous_imp_homeomorphic_map)
  show compact_space (subtopology (Euclidean_space n) S)
    by (simp add: S compact_space_subtopology)
  show Hausdorff_space (subtopology (Euclidean_space n) (h ` S))
    using hS_sub
    by (simp add: Hausdorff_Euclidean_space Hausdorff_space_subtopology)
  show continuous_map (subtopology (Euclidean_space n) S) (subtopology (Euclidean_space
n) (h ` S)) h
    using cmh continuous_map_in_subtopology by fastforce
  show h ` topspace (subtopology (Euclidean_space n) S) = topspace (subtopology
(Euclidean_space n) (h ` S))
    using clo_hS clo_S closedin_subset by auto
  show inj_on h (topspace (subtopology (Euclidean_space n) S))
    by (metis <inj_on h S> clo_S closedin_def topspace_subtopology_subset)
qed
then show subtopology (Euclidean_space n) (h ` S) homeomorphic_space subtopol-
ogy (Euclidean_space n) S
  using homeomorphic_space homeomorphic_space_sym by blast
qed

```

lemma lemmaIOD:

```

assumes
   $\exists T. T \in U \wedge c \subseteq T \exists T. T \in U \wedge d \subseteq T \bigcup U = c \cup d \wedge T. T \in U \implies T \neq \{\}$ 
pairwise_disjnt U  $\sim (\exists T. U \subseteq \{T\})$ 
shows c  $\in U$ 
using assms
apply safe
subgoal for C' D'
proof (cases C'=D')
  show c  $\in U$ 
    if UU:  $\bigcup U = c \cup d$ 
    and U:  $\bigwedge T. T \in U \implies T \neq \{\}$  disjoint U and  $\nexists T. U \subseteq \{T\}$  c  $\subseteq C'$  D'
 $\in U$  d  $\subseteq D'$  C' = D'
  proof -
    have c  $\cup d = D'$ 
    using Union_upper sup_mono UU that(5) that(6) that(7) that(8) by auto
    then have  $\bigcup U = D'$ 
    by (simp add: UU)
    with U have U = {D'}
    by (metis (no_types, lifting) disjnt_Union1 disjnt_self_iff_empty insertCI
pairwiseD subset_iff that(4) that(6))
    then show ?thesis
    using that(4) by auto
  qed

```

```

show c ∈ U
  if  $\bigcup U = c \cup d$  disjoint U C' ∈ U c ⊆ C'D' ∈ U d ⊆ D' C' ≠ D'
proof -
  have C' ∩ D' = {}
    using ⟨disjoint U⟩ ⟨C' ∈ U⟩ ⟨D' ∈ U⟩ ⟨C' ≠ D'⟩ unfolding disjnt_iff
pairwise_def
  by blast
  then show ?thesis
    using subset_antisym that(1) ⟨C' ∈ U⟩ ⟨c ⊆ C'⟩ ⟨d ⊆ D'⟩ by fastforce
qed
qed
done

theorem invariance_of_domain_Euclidean_space:
  assumes U: openin (Euclidean_space n) U
  and cmf: continuous_map (subtopology (Euclidean_space n) U) (Euclidean_space
n) f
  and inj_on f U
  shows openin (Euclidean_space n) (f ' U) (is openin ?E (f ' U))
proof (cases n = 0)
  case True
  have [simp]: Euclidean_space 0 = discrete_topology {λi. 0}
  by (auto simp: subtopology_eq_discrete_topology_sing topspace_Euclidean_space)
  show ?thesis
    using cmf True U by auto
next
  case False
  define enorm where enorm ≡ λx. sqrt(∑ i < n. x i ^ 2)
  have enorm_if [simp]: enorm (λi. if i = k then d else 0) = (if k < n then |d|
else 0) for k d
    using ⟨n ≠ 0⟩ by (auto simp: enorm_def power2_eq_square if_distrib [of λx.
x * _] cong: if_cong)
  define zero::nat⇒real where zero ≡ λi. 0
  have zero_in [simp]: zero ∈ topspace ?E
  using False by (simp add: zero_def topspace_Euclidean_space)
  have enorm_eq_0 [simp]: enorm x = 0 ↔ x = zero
  if x ∈ topspace(Euclidean_space n) for x
  using that unfolding zero_def enorm_def
  apply (simp add: sum_nonneg_eq_0_iff fun_eq_iff topspace_Euclidean_space)
  using le_less_linear by blast
  have [simp]: enorm zero = 0
  by (simp add: zero_def enorm_def)
  have cm_enorm: continuous_map ?E euclideanreal enorm
  unfolding enorm_def
proof (intro continuous_intros)
  show continuous_map ?E euclideanreal (λx. x i)

```

```

    if  $i \in \{..<n\}$  for  $i$ 
    using that by (auto simp: Euclidean_space_def intro: continuous_map_product_projection
continuous_map_from_subtopology)
qed auto
have enorm_ge0:  $0 \leq \text{enorm } x$  for  $x$ 
by (auto simp: enorm_def sum_nonneg)
have le_enorm:  $|x \ i| \leq \text{enorm } x$  if  $i < n$  for  $i \ x$ 
proof -
  have  $|x \ i| \leq \text{sqrt } (\sum_{k \in \{i\}} (x \ k)^2)$ 
  by auto
  also have  $\dots \leq \text{sqrt } (\sum_{k < n} (x \ k)^2)$ 
  by (rule real_sqrt_le_mono [OF sum_mono2]) (use that in auto)
  finally show ?thesis
  by (simp add: enorm_def)
qed
define B where  $B \equiv \lambda r. \{x \in \text{topspace } ?E. \text{enorm } x < r\}$ 
define C where  $C \equiv \lambda r. \{x \in \text{topspace } ?E. \text{enorm } x \leq r\}$ 
define S where  $S \equiv \lambda r. \{x \in \text{topspace } ?E. \text{enorm } x = r\}$ 
have BC:  $B \ r \subseteq C \ r$  and SC:  $S \ r \subseteq C \ r$  and disjSB:  $\text{disjnt } (S \ r) (B \ r)$  and
eqC:  $B \ r \cup S \ r = C \ r$  for  $r$ 
by (auto simp: B_def C_def S_def disjnt_def)
consider  $n = 1 \mid n \geq 2$ 
using False by linarith
then have **:  $\text{openin } ?E \ (h \ ` \ (B \ r))$ 
if  $r > 0$  and cmh:  $\text{continuous\_map}(\text{subtopology } ?E \ (C \ r)) \ ?E \ h$  and injh:
 $\text{inj\_on } h \ (C \ r)$  for  $r \ h$ 
proof cases
case 1
define e ::  $[\text{real}, \text{nat}] \Rightarrow \text{real}$  where  $e \equiv \lambda x \ i. \text{if } i = 0 \text{ then } x \text{ else } 0$ 
define e' ::  $(\text{nat} \Rightarrow \text{real}) \Rightarrow \text{real}$  where  $e' \equiv \lambda x. x \ 0$ 
have continuous_map euclidean euclideanreal  $(\lambda f. f \ (0::\text{nat}))$ 
by auto
then have continuous_map (subtopology (powertop_real UNIV)  $\{f. \forall n \geq \text{Suc } 0. f \ n = 0\}$ ) euclideanreal  $(\lambda f. f \ 0)$ 
by (metis (mono_tags) continuous_map_from_subtopology euclidean_product_topology)
then have hom_ee':  $\text{homeomorphic\_maps euclideanreal } (\text{Euclidean\_space } 1)$ 
e e'
by (auto simp: homeomorphic_maps_def e_def e'_def continuous_map_in_subtopology
Euclidean_space_def)
have eBr:  $e \ ` \ \{-r < .. < r\} = B \ r$ 
unfolding B_def e_def C_def
by (force simp: 1 topspace_Euclidean_space enorm_def power2_eq_square
if_distrib [of  $\lambda x. x * \_$ ] cong: if_cong)
have in_Cr:  $\bigwedge x. \llbracket -r < x; x < r \rrbracket \Longrightarrow (\lambda i. \text{if } i = 0 \text{ then } x \text{ else } 0) \in C \ r$ 
using  $\langle n \neq 0 \rangle$  by (auto simp: C_def topspace_Euclidean_space)
have inj:  $\text{inj\_on } (e' \circ h \circ e) \ \{-r < .. < r\}$ 
proof (clarsimp simp: inj_on_def e_def e'_def)
show  $(x::\text{real}) = y$ 
if  $f: h \ (\lambda i. \text{if } i = 0 \text{ then } x \text{ else } 0) \ 0 = h \ (\lambda i. \text{if } i = 0 \text{ then } y \text{ else } 0) \ 0$ 

```

```

    and  $-r < x < r$   $-r < y < r$ 
  for  $x\ y :: \text{real}$ 
  proof -
    have  $x: (\lambda i. \text{if } i = 0 \text{ then } x \text{ else } 0) \in C\ r$  and  $y: (\lambda i. \text{if } i = 0 \text{ then } y \text{ else } 0) \in C\ r$ 
    by (blast intro: inj_onD [OF <inj_on h (C r)>] that in_Cr)+
    have continuous_map (subtopology (Euclidean_space (Suc 0)) (C r))
      (Euclidean_space (Suc 0)) h
    using cmh by (simp add: 1)
    then have  $h' (\{x. \forall i \geq \text{Suc } 0. x\ i = 0\} \cap C\ r) \subseteq \{x. \forall i \geq \text{Suc } 0. x\ i = 0\}$ 
    by (force simp: Euclidean_space_def subtopology_subtopology continuous_map_def)
    have  $h (\lambda i. \text{if } i = 0 \text{ then } x \text{ else } 0)\ j = h (\lambda i. \text{if } i = 0 \text{ then } y \text{ else } 0)\ j$  for  $j$ 
    proof (cases j)
      case (Suc j')
      have  $h' (\{x. \forall i \geq \text{Suc } 0. x\ i = 0\} \cap C\ r) \subseteq \{x. \forall i \geq \text{Suc } 0. x\ i = 0\}$ 
      using continuous_map_image_subset_topspace [OF cmh]
      by (simp add: 1 Euclidean_space_def subtopology_subtopology)
      with Suc f x y show ?thesis
      by (simp add: 1 image_subset_iff)
    qed (use f in blast)
    then have  $(\lambda i. \text{if } i = 0 \text{ then } x \text{ else } 0) = (\lambda i::\text{nat}. \text{if } i = 0 \text{ then } y \text{ else } 0)$ 
    by (blast intro: inj_onD [OF <inj_on h (C r)>] that in_Cr)
    then show ?thesis
    by (simp add: fun_eq_iff) presburger
  qed
  qed
  have hom_e': homeomorphic_map (Euclidean_space 1) euclideanreal e'
  using hom_ee' homeomorphic_maps_map by blast
  have openin (Euclidean_space n) (h' e' { $-r <..})
  unfolding 1
  proof (subst homeomorphic_map_openness [OF hom_e', symmetric])
    show hesub:  $h' e' \{ -r <..
    using 1 C_def <math>\bigwedge r. B\ r \subseteq C\ r</math> cmh continuous_map_image_subset_topspace
    eBr by fastforce
    have cont: continuous_on { $-r <..} (e' o h o e)
    proof (intro continuous_on_compose)
      have  $\bigwedge i. \text{continuous\_on } \{ -r <..
      by (auto simp: continuous_on_topological)
      then show continuous_on { $-r <..} e
      by (force simp: e_def intro: continuous_on_coordinatewise_then_product)
      have subCr:  $e' \{ -r <..
      by (auto simp: eBr <math>\bigwedge r. B\ r \subseteq C\ r</math>) (auto simp: B_def)
      with cmh show continuous_on (e' { $-r <..}) h
    by (meson cm_Euclidean_space_iff_continuous_on_continuous_on_subset)
    have continuous_on (topspace ?E) e'
    by (metis 1 continuous_map_Euclidean_space_iff hom_ee' homeomorphic_maps_def)
    then show continuous_on (h' e' { $-r <..}) e'$$$$$$$$ 
```

```

      using hesub by (simp add: 1 e'_def continuous_on_subset)
    qed
    show openin euclideanreal (e' ' h ' e ' {- r <..\bigwedge r. \text{closedin } (\text{Euclidean\_space } n) (C\ r)
    unfolding C_def
    by (rule closedin_continuous_map_preimage [OF cm_enorm, of concl: {...}],
simplified)
  have cloS:  $\bigwedge r. \text{closedin } (\text{Euclidean\_space } n) (S\ r)$ 
    unfolding S_def
    by (rule closedin_continuous_map_preimage [OF cm_enorm, of concl: {...}],
simplified)
  have C_subset:  $C\ r \subseteq \text{UNIV} \rightarrow_E \{- |r|..|r|\}$ 
    using le_enorm  $\langle r > 0 \rangle$ 
    apply (auto simp: C_def topspace_Euclidean_space abs_le_iff)
    apply (metis add.inverse_neutral le_cases less_minus_iff not_le order_trans)
    by (metis enorm_ge0 not_le order_trans)
  have compactinC:  $\text{compactin } (\text{Euclidean\_space } n) (C\ r)$ 
    unfolding Euclidean_space_def compactin_subtopology
  proof
    show compactin (powertop_real UNIV) (C r)
    proof (rule closed_compactin [OF C_subset])
      show closedin (powertop_real UNIV) (C r)
      by (metis Euclidean_space_def cloC closedin_Euclidean_space closedin_closed_subtopology
topspace_Euclidean_space)
    qed (simp add: compactin_PiE)
    qed (auto simp: C_def topspace_Euclidean_space)
  have compactinS:  $\text{compactin } (\text{Euclidean\_space } n) (S\ r)$ 
    unfolding Euclidean_space_def compactin_subtopology
  proof
    show compactin (powertop_real UNIV) (S r)
    proof (rule closed_compactin)
      show  $S\ r \subseteq \text{UNIV} \rightarrow_E \{- |r|..|r|\}$ 
      using C_subset  $\langle \bigwedge r. S\ r \subseteq C\ r \rangle$  by blast
      show closedin (powertop_real UNIV) (S r)
      by (metis Euclidean_space_def cloS closedin_Euclidean_space closedin_closed_subtopology
topspace_Euclidean_space)
    qed (simp add: compactin_PiE)
    qed (auto simp: S_def topspace_Euclidean_space)
  have h_if_B:  $\bigwedge y. y \in B\ r \implies h\ y \in \text{topspace } ?E$ 
    using B_def  $\langle \bigwedge r. B\ r \cup S\ r = C\ r \rangle$  cmh continuous_map_image_subset_topspace
by fastforce
  have com_hSr:  $\text{compactin } (\text{Euclidean\_space } n) (h\ ' S\ r)$ 

```

```

    by (meson ⟨ $\bigwedge r. S \ r \subseteq C \ r$ ⟩ cmh compactinS compactin_subtopology image_compactin)
  have ope_comp_hSr: openin (Euclidean_space n) (topspace (Euclidean_space n) - h ' S r)
  proof (rule openin_diff)
    show closedin (Euclidean_space n) (h ' S r)
    using Hausdorff_Euclidean_space com_hSr compactin_imp_closedin by blast
  qed auto
  have h_pcs: h ' (B r) ∈ path_components_of (subtopology ?E (topspace ?E - h ' (S r)))
  proof (rule lemmaIOD)
    have pc_interval: path_connectedin (Euclidean_space n) {x ∈ topspace (Euclidean_space n). enorm x ∈ T}
    if T: is_interval T for T
  proof -
    define mul :: [real, nat ⇒ real, nat] ⇒ real where mul ≡ λa x i. a * x i
    let ?neg = mul (-1)
    have neg_neg [simp]: ?neg (?neg x) = x for x
    by (simp add: mul_def)
    have enorm_mul [simp]: enorm (mul a x) = abs a * enorm x for a x
    by (simp add: enorm_def mul_def power_mult_distrib) (metis real_sqrt_abs real_sqrt_mult sum_distrib_left)
    have mul_in_top: mul a x ∈ topspace ?E
    if x ∈ topspace ?E for a x
    using mul_def that topspace_Euclidean_space by auto
    have neg_in_S: ?neg x ∈ S r
    if x ∈ S r for x r
    using that topspace_Euclidean_space S_def by simp (simp add: mul_def)
    have *: path_connectedin ?E (S d)
    if d ≥ 0 for d
  proof (cases d = 0)
    let ?ES = subtopology ?E (S d)
    case False
    then have d > 0
    using that by linarith
    moreover have path_connected_space ?ES
    unfolding path_connected_space_iff_path_component
  proof clarify
    have **: path_component_of ?ES x y
    if x: x ∈ topspace ?ES and y: y ∈ topspace ?ES x ≠ ?neg y for x y
  proof -
    show ?thesis
    unfolding path_component_of_def pathin_def S_def
  proof (intro exI conjI)
    let ?g = (λx. mul (d / enorm x) x) ∘ (λt i. (1 - t) * x i + t * y i)
    show continuous_map (top_of_set {0::real..1}) (subtopology ?E {x ∈ topspace ?E. enorm x = d}) ?g
    proof (rule continuous_map_compose)

```



```

let ?Y = subtopology ?E (- {zero})
have **: False
  if eq0:  $\bigwedge j. (1 - r) * x\ j + r * y\ j = 0$ 
    and ne:  $x\ i \neq -y\ i$ 
    and d:  $enorm\ x = d\ enorm\ y = d$ 
    and r:  $0 \leq r \leq 1$ 
  for i r
proof -
  have mul (1-r) x = ?neg (mul r y)
    using eq0 by (simp add: mul_def fun_eq_iff algebra_simps)
  then have enorm (mul (1-r) x) = enorm (?neg (mul r y))
    by metis
  with r have (1-r) * enorm x = r * enorm y
    by simp
  then have r12:  $r = 1/2$ 
    using <d ≠ 0> d by auto
  show ?thesis
    using ne eq0 [of i] unfolding r12 by (simp add: algebra_simps)
qed
show continuous_map (top_of_set {0..1}) ?Y ( $\lambda t\ i. (1 - t) * x\ i$ 
+ t * y i)
  using x y
    unfolding continuous_map_componentwise_UNIV Euclidean_space_def continuous_map_in_subtopology
  apply (intro conjI allI continuous_intros)
  apply (auto simp: zero_def mul_def S_def Euclidean_space_def fun_eq_iff)
  using ** by blast
  have cm_enorm': continuous_map (subtopology (powertop_real UNIV) A) euclideanreal enorm for A
    unfolding enorm_def by (intro continuous_intros) auto
  have continuous_map ?Y (subtopology ?E {x. enorm x = d}) ( $\lambda x. mul\ (d / enorm\ x)\ x$ )
    unfolding continuous_map_in_subtopology
  proof (intro conjI)
    show continuous_map ?Y (Euclidean_space n) ( $\lambda x. mul\ (d / enorm\ x)\ x$ )
      unfolding continuous_map_in_subtopology Euclidean_space_def mul_def zero_def subtopology_subtopology continuous_map_componentwise_UNIV
    proof (intro conjI allI cm_enorm' continuous_intros)
      show enorm x ≠ 0
        if x ∈ topspace (subtopology (powertop_real UNIV) ({x.  $\forall i \geq n. x\ i = 0$ } ∩ - { $\lambda i. 0$ })) for x
          using that by simp (metis abs_le_zero_iff le_enorm not_less)
      qed auto
    qed (use <d > 0> enorm_ge0 in auto)
  moreover have subtopology ?E {x ∈ topspace ?E. enorm x = d}
    = subtopology ?E {x. enorm x = d}
    by (simp add: subtopology_restrict Collect_conj_eq)

```

```

      ultimately show continuous_map ?Y (subtopology (Euclidean_space
n) {x ∈ topspace (Euclidean_space n). enorm x = d}) (λx. mul (d / enorm x) x)
      by metis
    qed
    show ?g (0::real) = x ?g (1::real) = y
      using that by (auto simp: S_def zero_def mul_def fun_eq_iff)
    qed
  qed
  obtain a b where a: a ∈ topspace ?ES and b: b ∈ topspace ?ES
    and a ≠ b and negab: ?neg a ≠ b
  proof
    let ?v = λj i::nat. if i = j then d else 0
    show ?v 0 ∈ topspace (subtopology ?E (S d)) ?v 1 ∈ topspace (subtopology
?E (S d))
      using ⟨n ≥ 2⟩ ⟨d ≥ 0⟩ by (auto simp: S_def topspace_Euclidean_space)
      show ?v 0 ≠ ?v 1 ?neg (?v 0) ≠ (?v 1)
        using ⟨d > 0⟩ by (auto simp: mul_def fun_eq_iff)
    qed
    show path_component_of ?ES x y
      if x: x ∈ topspace ?ES and y: y ∈ topspace ?ES
      for x y
    proof -
      have path_component_of ?ES x (?neg x)
    proof -
      have path_component_of ?ES x a
        by (metis (no_types, opaque_lifting) ** a b ⟨a ≠ b⟩ negab
path_component_of_trans path_component_of_sym x)
      moreover
      have pa_ab: path_component_of ?ES a b using ** a b negab neg_neg
by blast
      then have path_component_of ?ES a (?neg x)
      by (metis ** ⟨a ≠ b⟩ cloS closedin_def neg_in_S path_component_of_equiv
topspace_subtopology_subset x)
      ultimately show ?thesis
        by (meson path_component_of_trans)
    qed
    then show ?thesis
      using ** x y by force
    qed
  qed
  ultimately show ?thesis
    by (simp add: cloS closedin_subset path_connectedin_def)
  qed (simp add: S_def cong: conj_cong)
  have path_component_of (subtopology ?E {x ∈ topspace ?E. enorm x ∈
T}) x y
    if enorm x = a x ∈ topspace ?E enorm x ∈ T enorm y = b y ∈ topspace
?E enorm y ∈ T
    for x y a b
    using that

```

```

proof (induction a b arbitrary: x y rule: linorder_less_wlog)
  case (less a b)
  then have  $a \geq 0$ 
    using enorm_ge0 by blast
  with less.hyps have  $b > 0$ 
    by linarith
  show ?case
  proof (rule path_component_of_trans)
    have  $y'_{ts}: \text{mul } (a / b) \ y \in \text{topspace } ?E$ 
      using  $\langle y \in \text{topspace } ?E \rangle$  mul_in_top by blast
    moreover have enorm (mul (a / b) y) = a
      unfolding enorm_mul using  $\langle 0 < b \rangle \langle 0 \leq a \rangle$  less.prem1 by simp
    ultimately have  $y'_{ts}: \text{mul } (a / b) \ y \in S \ a$ 
      using S_def by blast
    have  $x \in S \ a$ 
      using S_def less.prem1 by blast
    with  $\langle x \in \text{topspace } ?E \rangle \ y'_{ts} \ y'_{ts}$ 
    have path_component_of (subtopology ?E (S a)) x (mul (a / b) y)
      by (metis * [OF  $\langle a \geq 0 \rangle$ ] path_connected_space_iff_path_component
path_connected_in_def topspace_subtopology_subset)
    moreover
    have  $\{f \in \text{topspace } ?E. \text{enorm } f = a\} \subseteq \{f \in \text{topspace } ?E. \text{enorm } f \in$ 
T}
      using  $\langle \text{enorm } x = a \rangle \langle \text{enorm } x \in T \rangle$  by force
    ultimately
    show path_component_of (subtopology ?E  $\{x. x \in \text{topspace } ?E \wedge$ 
enorm x  $\in T\}$ ) x (mul (a / b) y)
      by (simp add: S_def path_component_of_mono)
    have pathin ?E  $(\lambda t. \text{mul } (((1 - t) * b + t * a) / b) \ y)$ 
      using  $\langle b > 0 \rangle \langle y \in \text{topspace } ?E \rangle$ 
      unfolding pathin_def Euclidean_space_def mul_def continuous_map_in_subtopology continuous_map_componentwise_UNIV
      by (intro allI conjI continuous_intros) auto
    moreover have mul  $(((1 - t) * b + t * a) / b) \ y \in \text{topspace } ?E$ 
      if  $t \in \{0..1\}$  for t
      using  $\langle y \in \text{topspace } ?E \rangle$  mul_in_top by blast
    moreover have enorm  $(\text{mul } (((1 - t) * b + t * a) / b) \ y) \in T$ 
      if  $t \in \{0..1\}$  for t
    proof -
      have  $a \in T \ b \in T$ 
        using less.prem1 by auto
      then have  $|(1 - t) * b + t * a| \in T$ 
        proof (rule mem_is_interval_1_I [OF T])
          show  $a \leq |(1 - t) * b + t * a|$ 
            using that  $\langle a \geq 0 \rangle$  less.hyps segment_bound_lemma by auto
          show  $|(1 - t) * b + t * a| \leq b$ 
            using that  $\langle a \geq 0 \rangle$  less.hyps by (auto intro: convex_bound_le)
        qed
      then show ?thesis

```

```

      unfolding enorm_mul ⟨enorm y = b⟩ using that ⟨b > 0⟩ by simp
    qed
    ultimately have pa: pathin (subtopology ?E {x ∈ topspace ?E. enorm
x ∈ T})
      (λt. mul (((1 - t) * b + t * a) / b) y)
      by (auto simp: pathin_subtopology)
    have ex_pathin: ∃ g. pathin (subtopology ?E {x ∈ topspace ?E. enorm
x ∈ T}) g ∧
      g 0 = y ∧ g 1 = mul (a / b) y
      apply (rule_tac x=λt. mul (((1 - t) * b + t * a) / b) y in exI)
      using ⟨b > 0⟩ pa by (auto simp: mul_def)
      show path_component_of (subtopology ?E {x. x ∈ topspace ?E ∧
enorm x ∈ T}) (mul (a / b) y) y
      by (rule path_component_of_sym) (simp add: path_component_of_def
ex_pathin)
    qed
  next
    case (refl a)
    then have pc: path_component_of (subtopology ?E (S (enorm u))) u v
      if u ∈ topspace ?E ∩ S (enorm x) v ∈ topspace ?E ∩ S (enorm u) for
u v
      using * [of a] enorm_ge0 that
    by (auto simp: path_connectedin_def path_connected_space_iff_path_component
S_def)
    have sub: {u ∈ topspace ?E. enorm u = enorm x} ⊆ {u ∈ topspace ?E.
enorm u ∈ T}
      using ⟨enorm x ∈ T⟩ by auto
    show ?case
      using pc [of x y] refl by (auto simp: S_def path_component_of_mono
[OF _ sub])
  next
    case (sym a b)
    then show ?case
      by (blast intro: path_component_of_sym)
    qed
  then show ?thesis
    by (simp add: path_connectedin_def path_connected_space_iff_path_component)
  qed
  have h ‘ S r ⊆ topspace ?E
    by (meson SC cmh compact_imp_compactin_subtopology compactinS com-
pactin_subset_topspace image_compactin)
  moreover
  have ¬ compact_space ?E
    by (metis compact_Euclidean_space ⟨n ≠ 0⟩)
  then have ¬ compactin ?E (topspace ?E)
    by (simp add: compact_space_def topspace_Euclidean_space)
  then have h ‘ S r ≠ topspace ?E
    using com_hSr by auto
  ultimately have top_hSr_ne: topspace (subtopology ?E (topspace ?E - h ‘

```

```

S r)) ≠ {}
  by auto
  show pc1: ∃ T. T ∈ path_components_of (subtopology ?E (topspace ?E - h
    ' S r)) ∧ h ' B r ⊆ T
  proof (rule exists_path_component_of_superset [OF _ top_hSr_ne])
    have path_connectedin ?E (h ' B r)
    proof (rule path_connectedin_continuous_map_image)
      show continuous_map (subtopology ?E (C r)) ?E h
      by (simp add: cmh)
      have path_connectedin ?E (B r)
      using pc_interval[of {.. $r$ }] is_interval_convex_1 unfolding B_def
    by auto
    then show path_connectedin (subtopology ?E (C r)) (B r)
    by (simp add: path_connectedin_subtopology BC)
  qed
  moreover have h ' B r ⊆ topspace ?E - h ' S r
  apply (auto simp: h_if_B)
  by (metis BC SC disjSB disjnt_iff inj_onD [OF injh] subsetD)
  ultimately show path_connectedin (subtopology ?E (topspace ?E - h ' S
    r)) (h ' B r)
  by (simp add: path_connectedin_subtopology)
  qed metis
  show ∃ T. T ∈ path_components_of (subtopology ?E (topspace ?E - h ' S
    r)) ∧ topspace ?E - h ' (C r) ⊆ T
  proof (rule exists_path_component_of_superset [OF _ top_hSr_ne])
    have eq: topspace ?E - {x ∈ topspace ?E. enorm x ≤ r} = {x ∈ topspace
      ?E. r < enorm x}
    by auto
    have path_connectedin ?E (topspace ?E - C r)
    using pc_interval[of {r<..}] is_interval_convex_1 unfolding C_def eq
  by auto
  then have path_connectedin ?E (topspace ?E - h ' C r)
  by (metis biglemma [OF ⟨n ≠ 0⟩ compactinC cmh injh])
  then show path_connectedin (subtopology ?E (topspace ?E - h ' S r))
    (topspace ?E - h ' C r)
  by (simp add: Diff_mono SC image_mono path_connectedin_subtopology)
  qed metis
  have topspace ?E ∩ (topspace ?E - h ' S r) = h ' B r ∪ (topspace ?E - h '
    C r)
    (is ?lhs = ?rhs)
  proof
    show ?lhs ⊆ ?rhs
    using ⟨ $\bigwedge r. B\ r \cup S\ r = C\ r$ ⟩ by auto
    have h ' B r ∩ h ' S r = {}
    by (metis Diff_triv ⟨ $\bigwedge r. B\ r \cup S\ r = C\ r$ ⟩ ⟨ $\bigwedge r. \text{disjnt } (S\ r) (B\ r)$ ⟩
      disjnt_def inf_commute inj_on_Un injh)
    then show ?rhs ⊆ ?lhs
    using path_components_of_subset pc1 ⟨ $\bigwedge r. B\ r \cup S\ r = C\ r$ ⟩
      by (fastforce simp add: h_if_B)
  qed

```

```

    then show  $\bigcup (\text{path\_components\_of } (\text{subtopology } ?E (\text{topspace } ?E - h \text{ ' } S
r))) = h \text{ ' } B \text{ } r \cup (\text{topspace } ?E - h \text{ ' } (C \text{ } r))$ 
      by (simp add: Union_path_components_of)
    show  $T \neq \{\}$ 
    if  $T \in \text{path\_components\_of } (\text{subtopology } ?E (\text{topspace } ?E - h \text{ ' } S \text{ } r))$  for  $T$ 
      using that by (simp add: nonempty_path_components_of)
    show disjoint (path_components_of (subtopology ?E (topspace ?E - h ' S
r)))
      by (simp add: pairwise_disjoint_path_components_of)
    have  $\neg \text{path\_connectedin } ?E (\text{topspace } ?E - h \text{ ' } S \text{ } r)$ 
    proof (subst biglemma [OF  $\langle n \neq 0 \rangle$  compactinS])
      show continuous_map (subtopology ?E (S r)) ?E h
    by (metis Un_commute Un_upper1 cmh continuous_map_from_subtopology_mono
eqC)
    show inj_on h (S r)
      using SC inj_on_subset injh by blast
    show  $\neg \text{path\_connectedin } ?E (\text{topspace } ?E - S \text{ } r)$ 
    proof
      have  $\text{topspace } ?E - S \text{ } r = \{x \in \text{topspace } ?E. \text{enorm } x \neq r\}$ 
      by (auto simp: S_def)
      moreover have  $\text{enorm } \{x \in \text{topspace } ?E. \text{enorm } x \neq r\} = \{0..\} - \{r\}$ 
      proof
        have  $\exists x. x \in \text{topspace } ?E \wedge \text{enorm } x \neq r \wedge d = \text{enorm } x$ 
        if  $d \neq r \text{ } d \geq 0$  for  $d$ 
      proof (intro exI conjI)
        show  $(\lambda i. \text{if } i = 0 \text{ then } d \text{ else } 0) \in \text{topspace } ?E$ 
        using  $\langle n \neq 0 \rangle$  by (auto simp: Euclidean_space_def)
        show  $\text{enorm } (\lambda i. \text{if } i = 0 \text{ then } d \text{ else } 0) \neq r \text{ } d = \text{enorm } (\lambda i. \text{if } i = 0
then d \text{ else } 0)$ 
          using  $\langle n \neq 0 \rangle$  that by simp_all
        qed
      then show  $\{0..\} - \{r\} \subseteq \text{enorm } \{x \in \text{topspace } ?E. \text{enorm } x \neq r\}$ 
      by (auto simp: image_def)
    qed (auto simp: enorm_ge0)
    ultimately have  $\text{non\_r}: \text{enorm } (\text{topspace } ?E - S \text{ } r) = \{0..\} - \{r\}$ 
    by simp
    have  $\exists x \geq 0. x \neq r \wedge r \leq x$ 
    by (metis gt_ex le_cases not_le order_trans)
    then have  $\neg \text{is\_interval } (\{0..\} - \{r\})$ 
    unfolding is_interval_1
    using  $\langle r > 0 \rangle$  by (auto simp: Bex_def)
    then show False
    if path_connectedin ?E (topspace ?E - S r)
      using path_connectedin_continuous_map_image [OF cm_enorm that]
by (simp add: is_interval_path_connected_1 non_r)
    qed
  qed
  then have  $\neg \text{path\_connected\_space } (\text{subtopology } ?E (\text{topspace } ?E - h \text{ ' } S \text{ } r))$ 
    by (simp add: path_connectedin_def)

```

```

    then show  $\nexists T. \text{path\_components\_of } (\text{subtopology } ?E (\text{topspace } ?E - h \text{ ` } S
r)) \subseteq \{T\}$ 
      by (simp add: path_components_of_subset_singleton)
    qed
  moreover have openin ?E A
  if A  $\in \text{path\_components\_of } (\text{subtopology } ?E (\text{topspace } ?E - h \text{ ` } (S r)))$  for
A
    using locally_path_connected_Euclidean_space [of n] that ope_comp_hSr
    by (simp add: locally_path_connected_space_open_path_components)
  ultimately show ?thesis by metis
qed
have  $\exists T. \text{openin } ?E T \wedge f x \in T \wedge T \subseteq f \text{ ` } U$ 
  if  $x \in U$  for  $x$ 
proof -
  have  $x: x \in \text{topspace } ?E$ 
  by (meson U in_mono openin_subset that)
  obtain V where V:  $\text{openin } (\text{powertop\_real UNIV}) V$  and Ueq:  $U = V \cap \{x. \forall i \geq n. x i = 0\}$ 
  using U by (auto simp: openin_subtopology Euclidean_space_def)
  with  $\langle x \in U \rangle$  have  $x \in V$  by blast
  then obtain T where Tfn:  $\text{finite } \{i. T i \neq \text{UNIV}\}$  and Topen:  $\bigwedge i. \text{open } (T i)$ 
  and Tx:  $x \in \text{Pi}_E \text{ UNIV } T$  and TV:  $\text{Pi}_E \text{ UNIV } T \subseteq V$ 
  using V by (force simp: openin_product_topology_alt)
  have  $\exists e > 0. \forall x'. |x' - x i| < e \longrightarrow x' \in T i$  for  $i$ 
  using Topen [of i] Tx by (auto simp: open_real)
  then obtain  $\beta$  where B0:  $\bigwedge i. \beta i > 0$  and BT:  $\bigwedge i x'. |x' - x i| < \beta i \implies x' \in T i$ 
  by metis
  define r where  $r \equiv \text{Min } (\text{insert } 1 (\beta \text{ ` } \{i. T i \neq \text{UNIV}\}))$ 
  have  $r > 0$ 
  by (simp add: B0 Tfn r_def)
  have inU:  $y \in U$ 
  if  $y: y \in \text{topspace } ?E$  and  $yxr: \bigwedge i. i < n \implies |y i - x i| < r$  for  $y$ 
proof -
  have  $y i \in T i$  for  $i$ 
  proof (cases  $T i = \text{UNIV}$ )
    show  $y i \in T i$  if  $T i \neq \text{UNIV}$ 
    proof (cases  $i < n$ )
      case True
      then show ?thesis
      using yxr [OF True] that by (simp add: r_def BT Tfn)
    next
      case False
      then show ?thesis
      using B0 Ueq  $\langle x \in U \rangle$   $\text{topspace\_Euclidean\_space } y$  by (force intro: BT)
    qed
  qed auto
  with TV have  $y \in V$  by auto

```

```

then show ?thesis
  using that by (auto simp: Ueq topspace_Euclidean_space)
qed
have xinU:  $(\lambda i. x\ i + y\ i) \in U$  if  $y \in C(r/2)$  for  $y$ 
proof (rule inU)
  have  $y: y \in \text{topspace } ?E$ 
  using C_def that by blast
  show  $(\lambda i. x\ i + y\ i) \in \text{topspace } ?E$ 
  using  $x\ y$  by (simp add: topspace_Euclidean_space)
  have  $\text{enorm } y \leq r/2$ 
  using that by (simp add: C_def)
  then show  $|x\ i + y\ i - x\ i| < r$  if  $i < n$  for  $i$ 
  using  $\text{le\_enorm enorm\_ge0}$  that  $\langle 0 < r \rangle \text{ leI order\_trans}$  by fastforce
qed
show ?thesis
proof (intro exI conjI)
  show openin ?E  $((f \circ (\lambda y\ i. x\ i + y\ i)) \text{ ` } B\ (r/2))$ 
  proof (rule **)
    have continuous_map (subtopology ?E  $(C(r/2))$ ) (subtopology ?E  $U$ )  $(\lambda y\ i. x\ i + y\ i)$ 
    by (auto simp: xinU continuous_map_in_subtopology
      intro!: continuous_intros continuous_map_Euclidean_space_add x)
    then show continuous_map (subtopology ?E  $(C(r/2))$ ) ?E  $(f \circ (\lambda y\ i. x\ i + y\ i))$ 
    by (rule continuous_map_compose) (simp add: cmf)
  show inj_on  $(f \circ (\lambda y\ i. x\ i + y\ i))$   $(C(r/2))$ 
  proof (clarsimp simp add: inj_on_def C_def topspace_Euclidean_space
    simp del: divide_const_simps)
    show  $y' = y$ 
    if  $ey: \text{enorm } y \leq r / 2$  and  $ey': \text{enorm } y' \leq r / 2$ 
    and  $y0: \forall i \geq n. y\ i = 0$  and  $y'0: \forall i \geq n. y'\ i = 0$ 
    and  $f eq: f\ (\lambda i. x\ i + y'\ i) = f\ (\lambda i. x\ i + y\ i)$ 
    for  $y'\ y :: \text{nat} \Rightarrow \text{real}$ 
  proof -
    have  $(\lambda i. x\ i + y\ i) \in U$ 
    proof (rule inU)
      show  $(\lambda i. x\ i + y\ i) \in \text{topspace } ?E$ 
      using topspace_Euclidean_space  $x\ y0$  by auto
      show  $|x\ i + y\ i - x\ i| < r$  if  $i < n$  for  $i$ 
      using  $ey\ \text{le\_enorm [of _ } y\text{ ] } \langle r > 0 \rangle$  that by fastforce
    qed
    moreover have  $(\lambda i. x\ i + y'\ i) \in U$ 
    proof (rule inU)
      show  $(\lambda i. x\ i + y'\ i) \in \text{topspace } ?E$ 
      using topspace_Euclidean_space  $x\ y'0$  by auto
      show  $|x\ i + y'\ i - x\ i| < r$  if  $i < n$  for  $i$ 
      using  $ey'\ \text{le\_enorm [of _ } y'\text{ ] } \langle r > 0 \rangle$  that by fastforce
    qed
    ultimately have  $(\lambda i. x\ i + y'\ i) = (\lambda i. x\ i + y\ i)$ 

```



```

      using feq by (meson ⟨inj_on f U⟩ inj_on_def)
    then show ?thesis
      by (auto simp: fun_eq_iff)
  qed
qed
qed (simp add: ⟨0 < r⟩)
have x ∈ (λy i. x i + y i) ‘ B (r / 2)
proof
  show x = (λi. x i + zero i)
    by (simp add: zero_def)
  qed (auto simp: B_def ⟨r > 0⟩)
  then show f x ∈ (f ∘ (λy i. x i + y i)) ‘ B (r/2)
    by (metis image_comp image_eqI)
  show (f ∘ (λy i. x i + y i)) ‘ B (r/2) ⊆ f ‘ U
    using ⟨∧r. B r ⊆ C r⟩ x in U by fastforce
  qed
qed
then show ?thesis
  using openin_subopen by force
qed

```

corollary *invariance_of_domain_Euclidean_space_embedding_map:*

```

  assumes openin (Euclidean_space n) U
  and cmf: continuous_map(subtopology (Euclidean_space n) U) (Euclidean_space
n) f
  and inj_on f U
  shows embedding_map(subtopology (Euclidean_space n) U) (Euclidean_space
n) f
proof (rule injective_open_imp_embedding_map [OF cmf])
  show open_map (subtopology (Euclidean_space n) U) (Euclidean_space n) f
    unfolding open_map_def
    by (meson assms continuous_map_from_subtopology_mono inj_on_subset
invariance_of_domain_Euclidean_space openin_imp_subset openin_trans_full)
  show inj_on f (topspace (subtopology (Euclidean_space n) U))
    using assms openin_subset topspace_subtopology_subset by fastforce
qed

```

corollary *invariance_of_domain_Euclidean_space_gen:*

```

  assumes n ≤ m and U: openin (Euclidean_space m) U
  and cmf: continuous_map(subtopology (Euclidean_space m) U) (Euclidean_space
n) f
  and inj_on f U
  shows openin (Euclidean_space n) (f ‘ U)
proof –
  have *: Euclidean_space n = subtopology (Euclidean_space m) (topspace(Euclidean_space
n))
  by (metis Euclidean_space_def ⟨n ≤ m⟩ inf.absorb_iff2 subset_Euclidean_space
subtopology_subtopology topspace_Euclidean_space)

```

moreover have $U \subseteq \text{topspace } (\text{subtopology } (\text{Euclidean_space } m) \ U)$
by $(\text{metis } U \text{ inf.absorb_iff2 } \text{openin_subset } \text{openin_subtopology } \text{openin_topspace})$
ultimately show $?thesis$
by $(\text{metis } (\text{no_types}) \ U \ \langle \text{inj_on } f \ U \rangle \ \text{cmf } \text{continuous_map_in_subtopology } \text{inf.absorb_iff2}$
 $\text{inf.orderE } \text{invariance_of_domain_Euclidean_space } \text{openin_imp_subset}$
 $\text{openin_subtopology } \text{openin_topspace})$
qed

corollary invariance_of_domain_Euclidean_space_embedding_map_gen:
assumes $n \leq m$ **and** $U: \text{openin } (\text{Euclidean_space } m) \ U$
and $\text{cmf}: \text{continuous_map}(\text{subtopology } (\text{Euclidean_space } m) \ U) \ (\text{Euclidean_space } n) \ f$
and $\text{inj_on } f \ U$
shows $\text{embedding_map}(\text{subtopology } (\text{Euclidean_space } m) \ U) \ (\text{Euclidean_space } n) \ f$
proof $(\text{rule } \text{injective_open_imp_embedding_map } [OF \ \text{cmf}])$
show $\text{open_map } (\text{subtopology } (\text{Euclidean_space } m) \ U) \ (\text{Euclidean_space } n) \ f$
by $(\text{meson } U \ \langle n \leq m \rangle \ \langle \text{inj_on } f \ U \rangle \ \text{cmf } \text{continuous_map_from_subtopology_mono}$
 $\text{invariance_of_domain_Euclidean_space_gen } \text{open_map_def } \text{openin_open_subtopology}$
 $\text{subset_inj_on})$
show $\text{inj_on } f \ (\text{topspace } (\text{subtopology } (\text{Euclidean_space } m) \ U))$
using $\text{assms } \text{openin_subset } \text{topspace_subtopology_subset}$ **by** fastforce
qed

0.4.3 Relating two variants of Euclidean space, one within product topology.

proposition homeomorphic_maps_Euclidean_space_euclidean_gen_OLD:
fixes $B :: 'n::\text{euclidean_space set}$
assumes $\text{finite } B$ $\text{independent } B$ **and** $\text{orth}: \text{pairwise orthogonal } B$ **and** $n: \text{card } B = n$
obtains $f \ g$ **where** $\text{homeomorphic_maps } (\text{Euclidean_space } n) \ (\text{top_of_set } (\text{span } B)) \ f \ g$
proof –
note $\text{representation_basis } [OF \ \langle \text{independent } B \rangle, \ \text{simp}]$
obtain b **where** $\text{injb}: \text{inj_on } b \ \{..<n\}$ **and** $\text{beg}: b \ \{..<n\} = B$
using $\text{finite_imp_nat_seg_image_inj_on } [OF \ \langle \text{finite } B \rangle]$
by $(\text{metis } n \ \text{card_Collect_less_nat_card_image_lessThan_def})$
then have $\text{biB}: \bigwedge i. i < n \implies b \ i \in B$
by force
have $\text{repr}: \bigwedge v. v \in \text{span } B \implies (\sum i < n. \text{representation } B \ v \ (b \ i) \ *_{\mathbb{R}} \ b \ i) = v$
using $\text{real_vector.sum_representation_eq } [OF \ \langle \text{independent } B \rangle \ _ \ \langle \text{finite } B \rangle]$
by $(\text{metis } (\text{no_types}, \ \text{lifting}) \ \text{injb } \text{beg } \text{order_refl } \text{sum.reindex_cong})$
let $?f = \lambda x. \sum i < n. x \ i \ *_{\mathbb{R}} \ b \ i$
let $?g = \lambda v \ i. \text{if } i < n \text{ then } \text{representation } B \ v \ (b \ i) \ \text{else } 0$
show thesis
proof
show $\text{homeomorphic_maps } (\text{Euclidean_space } n) \ (\text{top_of_set } (\text{span } B)) \ ?f \ ?g$

```

    unfolding homeomorphic_maps_def
  proof (intro conjI)
    have *: continuous_map euclidean (top_of_set (span B)) ?f
      by (metis (mono_tags) biB continuous_map_span_sum lessThan_iff)
    show continuous_map (Euclidean_space n) (top_of_set (span B)) ?f
      unfolding Euclidean_space_def
    by (rule continuous_map_from_subtopology) (simp add: euclidean_product_topology
*)
    show continuous_map (top_of_set (span B)) (Euclidean_space n) ?g
      unfolding Euclidean_space_def
    by (auto simp: continuous_map_in_subtopology continuous_map_componentwise_UNIV
continuous_on_representation ⟨independent B⟩ biB orth pairwise_orthogonal_imp_finite)
    have [simp]:  $\bigwedge x i. i < n \implies x i *_{\mathbb{R}} b i \in \text{span } B$ 
      by (simp add: biB span_base span_scale)
    have representation_B (?f x) (b j) = x j
      if 0:  $\forall i \geq n. x i = (0::\text{real})$  and  $j < n$  for  $x j$ 
    proof -
      have representation_B (?f x) (b j) =  $(\sum i < n. \text{representation } B (x i *_{\mathbb{R}} b i)$ 
(b j))
      by (subst real_vector.representation_sum) (auto simp add: ⟨independent
B⟩)
      also have ... =  $(\sum i < n. x i * \text{representation } B (b i) (b j))$ 
      by (simp add: assms(2) biB representation_scale span_base)
      also have ... =  $(\sum i < n. \text{if } b j = b i \text{ then } x i \text{ else } 0)$ 
      by (simp add: biB if_distrib cong: if_cong)
      also have ... =  $x j$ 
      using that inj_on_eq_iff [OF injb] by auto
      finally show ?thesis .
    qed
    then show  $\forall x \in \text{topspace } (Euclidean\_space\ n). ?g (?f\ x) = x$ 
      by (auto simp: Euclidean_space_def)
    show  $\forall y \in \text{topspace } (top\_of\_set\ (span\ B)). ?f (?g\ y) = y$ 
      using repr by (auto simp: Euclidean_space_def)
  qed
qed
qed
qed

proposition homeomorphic_maps_Euclidean_space_euclidean_gen:
  fixes B :: 'n::euclidean_space set
  assumes independent B and orth: pairwise_orthogonal B and n: card B = n
  and 1:  $\bigwedge u. u \in B \implies \text{norm } u = 1$ 
  obtains f g where homeomorphic_maps (Euclidean_space n) (top_of_set (span
B)) f g
  and  $\bigwedge x. x \in \text{topspace } (Euclidean\_space\ n) \implies (\text{norm } (f\ x))^2 = (\sum i < n. (x
i)^2)$ 
  proof -
    note representation_basis [OF ⟨independent B⟩, simp]
    have finite B
    using ⟨independent B⟩ finiteI_independent by metis

```

```

obtain  $b$  where  $\text{injb: inj\_on } b \{..<n\}$  and  $\text{beg: } b \{..<n\} = B$ 
  using  $\text{finite\_imp\_nat\_seg\_image\_inj\_on } [OF \langle \text{finite } B \rangle]$ 
  by ( $\text{metis } n \text{ card\_Collect\_less\_nat card\_image lessThan\_def}$ )
then have  $\text{biB: } \bigwedge i. i < n \implies b \ i \in B$ 
  by force
have  $0 \notin B$ 
  using  $\langle \text{independent } B \rangle \text{ dependent\_zero}$  by blast
have  $[\text{simp}]: b \ i \cdot b \ j = (\text{if } j = i \text{ then } 1 \text{ else } 0)$ 
  if  $i < n \ j < n$  for  $i \ j$ 
proof ( $\text{cases } i = j$ )
  case True
    with 1 that show ?thesis
    by ( $\text{auto simp: norm\_eq\_sqrt\_inner biB}$ )
  next
    case False
    then have  $b \ i \neq b \ j$ 
    by ( $\text{meson inj\_onD injb lessThan\_iff that}$ )
    then show ?thesis
    using orth by ( $\text{auto simp: orthogonal\_def pairwise\_def norm\_eq\_sqrt\_inner}$ 
that  $\text{biB}$ )
  qed
have  $[\text{simp}]: \bigwedge x \ i. i < n \implies x \ i *_{\mathbb{R}} b \ i \in \text{span } B$ 
  by ( $\text{simp add: biB span\_base span\_scale}$ )
have repr:  $\bigwedge v. v \in \text{span } B \implies (\sum i < n. \text{representation } B \ v \ (b \ i) *_{\mathbb{R}} b \ i) = v$ 
  using  $\text{real\_vector.sum\_representation\_eq } [OF \langle \text{independent } B \rangle \text{ } \langle \text{finite } B \rangle]$ 
  by ( $\text{metis (no\_types, lifting) injb beq order\_refl sum.reindex\_cong}$ )
  define  $f$  where  $f \equiv \lambda x. \sum i < n. x \ i *_{\mathbb{R}} b \ i$ 
  define  $g$  where  $g \equiv \lambda v \ i. \text{if } i < n \text{ then representation } B \ v \ (b \ i) \text{ else } 0$ 
show ?thesis
proof
  show  $\text{homeomorphic\_maps } (\text{Euclidean\_space } n) (\text{top\_of\_set } (\text{span } B)) \ f \ g$ 
  unfolding  $\text{homeomorphic\_maps\_def}$ 
proof ( $\text{intro conjI}$ )
    have  $*$ :  $\text{continuous\_map euclidean } (\text{top\_of\_set } (\text{span } B)) \ f$ 
    unfolding  $f\_def$ 
    by ( $\text{rule continuous\_map\_span\_sum}$ ) ( $\text{use biB } \langle 0 \notin B \rangle$  in auto)
    show  $\text{continuous\_map } (\text{Euclidean\_space } n) (\text{top\_of\_set } (\text{span } B)) \ f$ 
    unfolding  $\text{Euclidean\_space\_def}$ 
    by ( $\text{rule continuous\_map\_from\_subtopology}$ ) ( $\text{simp add: euclidean\_product\_topology}$ 
*)
    show  $\text{continuous\_map } (\text{top\_of\_set } (\text{span } B)) \ (\text{Euclidean\_space } n) \ g$ 
    unfolding  $\text{Euclidean\_space\_def } g\_def$ 
    by ( $\text{auto simp: continuous\_map\_in\_subtopology continuous\_map\_componentwise\_UNIV}$ 
continuous_on_representation  $\langle \text{independent } B \rangle$   $\text{biB orth pairwise\_orthogonal\_imp\_finite}$ )
    have  $\text{representation } B \ (f \ x) \ (b \ j) = x \ j$ 
    if  $0: \forall i \geq n. x \ i = (0::\text{real})$  and  $j < n$  for  $x \ j$ 
    proof –
      have  $\text{representation } B \ (f \ x) \ (b \ j) = (\sum i < n. \text{representation } B \ (x \ i *_{\mathbb{R}} b \ i)$ 
 $(b \ j))$ 

```

```

    unfolding f_def
    by (subst real_vector.representation_sum) (auto simp add: ⟨independent
B⟩)
  also have ... = (∑ i<n. x i * representation B (b i) (b j))
    by (simp add: ⟨independent B⟩ biB representation_scale span_base)
  also have ... = (∑ i<n. if b j = b i then x i else 0)
    by (simp add: biB if_distrib cong: if_cong)
  also have ... = x j
    using that inj_on_eq_iff [OF injb] by auto
  finally show ?thesis .
qed
then show ∀ x∈topspace (Euclidean_space n). g (f x) = x
  by (auto simp: Euclidean_space_def f_def g_def)
show ∀ y∈topspace (top_of_set (span B)). f (g y) = y
  using repr by (auto simp: Euclidean_space_def f_def g_def)
qed
show normeq: (norm (f x))2 = (∑ i<n. (x i)2) if x ∈ topspace (Euclidean_space
n) for x
  unfolding f_def dot_square_norm [symmetric]
  by (simp add: power2_eq_square inner_sum_left inner_sum_right if_distrib
biB cong: if_cong)
qed
qed

corollary homeomorphic_maps_Euclidean_space_euclidean:
  obtains f :: (nat ⇒ real) ⇒ 'n::euclidean_space and g
  where homeomorphic_maps (Euclidean_space (DIM('n))) euclidean f g
  by (force intro: homeomorphic_maps_Euclidean_space_euclidean_gen [OF in-
dependent_Basis orthogonal_Basis refl norm_Basis])

lemma homeomorphic_maps_nsphere_euclidean_sphere:
  fixes B :: 'n::euclidean_space set
  assumes B: independent B and orth: pairwise orthogonal B and n: card B = n
  and n ≠ 0
  and 1: ⋀ u. u ∈ B ⇒ norm u = 1
  obtains f :: (nat ⇒ real) ⇒ 'n::euclidean_space and g
  where homeomorphic_maps (nsphere(n - 1)) (top_of_set (sphere 0 1 ∩ span
B)) f g
proof -
  have finite B
  using ⟨independent B⟩ finiteI_independent by metis
  obtain f g where fg: homeomorphic_maps (Euclidean_space n) (top_of_set
(span B)) f g
  and normf: ⋀ x. x ∈ topspace (Euclidean_space n) ⇒ (norm (f x))2 = (∑ i<n.
(x i)2)
  using homeomorphic_maps_Euclidean_space_euclidean_gen [OF B orth n 1]
  by blast
  obtain b where injb: inj_on b {..

```

```

    by (metis n card_Collect_less_nat card_image lessThan_def)
  then have biB:  $\bigwedge i. i < n \implies b\ i \in B$ 
    by force
  have [simp]:  $\bigwedge i. i < n \implies b\ i \neq 0$ 
    using ‹independent B› biB dependent_zero by fastforce
  have [simp]:  $b\ i \cdot b\ j = (if\ j = i\ then\ (norm\ (b\ i))^2\ else\ 0)$ 
    if  $i < n\ j < n$  for  $i\ j$ 
  proof (cases  $i = j$ )
    case False
    then have  $b\ i \neq b\ j$ 
      by (meson inj_onD injb lessThan_iff that)
    then show ?thesis
      using orth by (auto simp: orthogonal_def pairwise_def norm_eq_sqrt_inner
that biB)
  qed (auto simp: norm_eq_sqrt_inner)
  have [simp]:  $Suc\ (n - Suc\ 0) = n$ 
    using Suc_pred ‹ $n \neq 0$ › by blast
  then have [simp]:  $\{.. $card\ B - Suc\ 0\} = \{.. $card\ B\}$ 
    using n by fastforce
  show thesis
  proof
    have 1:  $norm\ (f\ x) = 1$ 
      if  $(\sum i < card\ B. (x\ i)^2) = (1::real)$   $x \in topspace\ (Euclidean\_space\ n)$  for  $x$ 
    proof -
      have  $norm\ (f\ x)^2 = 1$ 
        using normf that by (simp add: n)
      with that show ?thesis
        by (simp add: power2_eq_imp_eq)
    qed
    have homeomorphic_maps (nsphere  $(n - 1)$ ) (top_of_set (span  $B \cap sphere\ 0$ 
1)) f g
      unfolding nsphere_def subtopology_subtopology [symmetric]
      proof (rule homeomorphic_maps_subtopologies_alt)
        show homeomorphic_maps (Euclidean_space (Suc  $(n - 1)$ )) (top_of_set (span
B)) f g
          using fg by (force simp add: )
        show  $f\ ' (topspace\ (Euclidean\_space\ (Suc\ (n - 1))) \cap \{x. (\sum i \leq n - 1. (x\ i)^2) = 1\}) \subseteq sphere\ 0\ 1$ 
          using n by (auto simp: image_subset_iff Euclidean_space_def 1)
        have  $(\sum i \leq n - Suc\ 0. (g\ u\ i)^2) = 1$ 
          if  $u \in span\ B$  and  $norm\ (u::'n) = 1$  for  $u$ 
        proof -
          obtain  $v$  where [simp]:  $u = f\ v\ v \in topspace\ (Euclidean\_space\ n)$ 
            using fg unfolding homeomorphic_maps_map subset_iff
            by (metis ‹ $u \in span\ B$ › homeomorphic_imp_surjective_map image_eqI
topspace_euclidean_subtopology)
          then have [simp]:  $g\ (f\ v) = v$ 
            by (meson fg homeomorphic_maps_map)
          have fv21:  $norm\ (f\ v)^2 = 1$$$ 
```

```

    using that by simp
  show ?thesis
    using that normf fv21 ‹ $v \in \text{topspace}(\text{Euclidean\_space } n)$ ›  $n$  by force
qed
  then show  $g^{-1}(\text{topspace}(\text{top\_of\_set}(\text{span } B)) \cap \text{sphere } 0 \ 1) \subseteq \{x. (\sum_{i \leq n} 1 \cdot (x \ i)^2) = 1\}$ 
    by auto
qed
  then show  $\text{homeomorphic\_maps}(\text{nsphere}(n - 1))(\text{top\_of\_set}(\text{sphere } 0 \ 1 \cap \text{span } B)) \ f \ g$ 
    by (simp add: inf_commute)
qed
qed

```

0.4.4 Invariance of dimension and domain

```

lemma homeomorphic_maps_iff_homeomorphism [simp]:
  homeomorphic_maps (top_of_set S) (top_of_set T)  $f \ g \longleftrightarrow$  homeomorphism
  S T f g
  unfolding homeomorphic_maps_def homeomorphism_def by force

```

```

lemma homeomorphic_space_iff_homeomorphic [simp]:
  (top_of_set S) homeomorphic_space (top_of_set T)  $\longleftrightarrow$  S homeomorphic T
  by (simp add: homeomorphic_def homeomorphic_space_def)

```

```

lemma homeomorphic_subspace_Euclidean_space:
  fixes S :: 'a::euclidean_space set
  assumes subspace S
  shows top_of_set S homeomorphic_space Euclidean_space n  $\longleftrightarrow$  dim S = n

```

proof –

```

  obtain B where B:  $B \subseteq S$  independent B span B = S card B = dim S
    and orth: pairwise orthogonal B and 1:  $\bigwedge x. x \in B \implies \text{norm } x = 1$ 
    by (metis assms orthonormal_basis_subspace)
  then have finite B
    by (simp add: pairwise_orthogonal_imp_finite)
  have top_of_set S homeomorphic_space top_of_set (span B)
    unfolding homeomorphic_space_iff_homeomorphic
    by (auto simp: assms B intro: homeomorphic_subspaces)
  also have ... homeomorphic_space Euclidean_space (dim S)
    unfolding homeomorphic_space_def
    using homeomorphic_maps_Euclidean_space_euclidean_gen [OF ‹independent B› orth] homeomorphic_maps_sym 1 B
    by metis
  finally have top_of_set S homeomorphic_space Euclidean_space (dim S) .
  then show ?thesis
    using homeomorphic_space_sym homeomorphic_space_trans invariance_of_dimension_Euclidean_space
  by blast
qed

```

```

lemma homeomorphic_subspace_Euclidean_space_dim:
  fixes  $S :: 'a::euclidean\_space\ set$ 
  assumes subspace S
  shows top_of_set S homeomorphic_space Euclidean_space (dim S)
  by (simp add: homeomorphic_subspace_Euclidean_space assms)

lemma homeomorphic_subspaces_eq:
  fixes  $S\ T :: 'a::euclidean\_space\ set$ 
  assumes subspace S subspace T
  shows  $S\ homeomorphic\ T \longleftrightarrow dim\ S = dim\ T$ 
proof
  show  $dim\ S = dim\ T$ 
    if S homeomorphic T
  proof –
    have Euclidean_space (dim S) homeomorphic_space top_of_set S
    using  $\langle subspace\ S \rangle\ homeomorphic\_space\_sym\ homeomorphic\_subspace\_Euclidean\_space\_dim$ 
by blast
    also have  $\dots\ homeomorphic\_space\ top\_of\_set\ T$ 
    by (simp add: that)
    also have  $\dots\ homeomorphic\_space\ Euclidean\_space\ (dim\ T)$ 
    by (simp add: homeomorphic_subspace_Euclidean_space assms)
    finally have Euclidean_space (dim S) homeomorphic_space Euclidean_space (dim T) .
    then show ?thesis
    by (simp add: invariance_of_dimension_Euclidean_space)
  qed
next
  show S homeomorphic T
    if  $dim\ S = dim\ T$ 
    by (metis that assms homeomorphic_subspaces)
qed

lemma homeomorphic_affine_Euclidean_space:
  assumes affine S
  shows  $top\_of\_set\ S\ homeomorphic\_space\ Euclidean\_space\ n \longleftrightarrow aff\_dim\ S = n$ 
  (is  $?X\ homeomorphic\_space\ ?E \longleftrightarrow aff\_dim\ S = n$ )
proof (cases S = {})
  case True
    with assms show ?thesis
    using homeomorphic_empty_space nontrivial_Euclidean_space by fastforce
  next
    case False
    then obtain a where  $a \in S$ 
    by force
    have  $(?X\ homeomorphic\_space\ ?E)$ 
       $= (top\_of\_set\ (image\ (\lambda x. -a + x)\ S)\ homeomorphic\_space\ ?E)$ 
    proof
      show  $top\_of\_set\ ((+)\ (-\ a)\ `S)\ homeomorphic\_space\ ?E$ 

```



```

    if ?X homeomorphic_space ?E
    using that
  by (meson homeomorphic_space_iff_homeomorphic homeomorphic_space_sym
homeomorphic_space_trans homeomorphic_translation)
  show ?X homeomorphic_space ?E
    if top_of_set ((+) (- a) ' S) homeomorphic_space ?E
    using that
  by (meson homeomorphic_space_iff_homeomorphic homeomorphic_space_trans
homeomorphic_translation)
qed
also have ...  $\longleftrightarrow$  aff_dim S = n
  by (metis <a ∈ S> aff_dim_eq_dim affine_diffs_subspace affine_hull_eq assms
homeomorphic_subspace_Euclidean_space of_nat_eq_iff)
  finally show ?thesis .
qed

corollary invariance_of_domain_subspaces:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes ope: openin (top_of_set U) S
    and subspace U subspace V and VU: dim V  $\leq$  dim U
    and conf: continuous_on S f and fim: f ' S  $\subseteq$  V
    and inj: inj_on f S
  shows openin (top_of_set V) (f ' S)
proof -
  have S  $\subseteq$  U
  using openin_imp_subset [OF ope] .
  have Uhom: top_of_set U homeomorphic_space Euclidean_space (dim U)
  and Vhom: top_of_set V homeomorphic_space Euclidean_space (dim V)
  by (simp_all add: assms homeomorphic_subspace_Euclidean_space_dim)
  then obtain  $\varphi$   $\varphi'$  where hom: homeomorphic_maps (top_of_set U) (Euclidean_space
(dim U))  $\varphi$   $\varphi'$ 
  by (auto simp: homeomorphic_space_def)
  obtain  $\psi$   $\psi'$  where  $\psi$ : homeomorphic_map (top_of_set V) (Euclidean_space
(dim V))  $\psi$ 
    and  $\psi'\psi$ :  $\forall x \in V. \psi'(\psi x) = x$ 
  using Vhom by (auto simp: homeomorphic_space_def homeomorphic_maps_map)
  have (( $\psi \circ f \circ \varphi'$ )  $\circ \varphi$ ) ' S = ( $\psi \circ f$ ) ' S
  proof (rule image_cong [OF refl])
    show ( $\psi \circ f \circ \varphi' \circ \varphi$ ) x = ( $\psi \circ f$ ) x if x ∈ S for x
    using that unfolding o_def
  by (metis <S  $\subseteq$  U> hom homeomorphic_maps_map in_mono topspace_euclidean_subtopology)
qed
moreover
have openin (Euclidean_space (dim V)) (( $\psi \circ f \circ \varphi'$ ) '  $\varphi$  ' S)
proof (rule invariance_of_domain_Euclidean_space_gen [OF VU])
  show openin (Euclidean_space (dim U)) ( $\varphi$  ' S)
    using homeomorphic_map_openness_eq hom homeomorphic_maps_map ope
by blast

```

```

    show continuous_map (subtopology (Euclidean_space (dim U)) (φ ' S)) (Euclidean_space
(dim V)) (ψ ∘ f ∘ φ')
  proof (intro continuous_map_compose)
    have continuous_on ({x. ∀ i ≥ dim U. x i = 0} ∩ φ ' S) φ'
      if continuous_on {x. ∀ i ≥ dim U. x i = 0} φ'
      using that by (force elim: continuous_on_subset)
    moreover have φ' ' ({x. ∀ i ≥ dim U. x i = 0} ∩ φ ' S) ⊆ S
      if ∀ x ∈ U. φ' (φ x) = x
      using that ⟨S ⊆ U⟩ by fastforce
    ultimately show continuous_map (subtopology (Euclidean_space (dim U))
(φ ' S)) (top_of_set S) φ'
      using hom_unfolding homeomorphic_maps_def
    by (simp add: Euclidean_space_def subtopology_subtopology euclidean_product_topology)
    show continuous_map (top_of_set S) (top_of_set V) f
      by (simp add: contf fim)
    show continuous_map (top_of_set V) (Euclidean_space (dim V)) ψ
      by (simp add: ψ homeomorphic_imp_continuous_map)
  qed
  show inj_on (ψ ∘ f ∘ φ') (φ ' S)
    using injf hom
    unfolding inj_on_def homeomorphic_maps_map
    by simp (metis ⟨S ⊆ U⟩ ψ'ψ fim imageI subsetD)
  qed
  ultimately have openin (Euclidean_space (dim V)) (ψ ' f ' S)
    by (simp add: image_comp)
  then show ?thesis
    by (simp add: fim homeomorphic_map_openness_eq [OF ψ])
  qed

lemma invariance_of_domain:
  fixes f :: 'a ⇒ 'a::euclidean_space
  assumes continuous_on S f open S inj_on f S shows open(f ' S)
  using invariance_of_domain_subspaces [of UNIV S UNIV] assms by (force
simp add: )

corollary invariance_of_dimension_subspaces:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
    and subspace U subspace V
    and contf: continuous_on S f and fim: f ' S ⊆ V
    and injf: inj_on f S and S ≠ {}
  shows dim U ≤ dim V
proof -
  have False if dim V < dim U
  proof -
    obtain T where subspace T T ⊆ U dim T = dim V
      using choose_subspace_of_subspace [of dim V U]
    by (metis ⟨dim V < dim U⟩ assms(2) order.strict_implies_order span_eq_iff)
    then have V homeomorphic T

```

```

    by (simp add: ⟨subspace V⟩ homeomorphic_subspaces)
  then obtain h k where homhk: homeomorphism V T h k
    using homeomorphic_def by blast
  have continuous_on S (h ∘ f)
    by (meson contf continuous_on_compose continuous_on_subset fim homeo-
morphisms_cont1 homhk)
  moreover have (h ∘ f) ' S ⊆ U
    using ⟨T ⊆ U⟩ fim homeomorphism_image1 homhk by fastforce
  moreover have inj_on (h ∘ f) S
    apply (clarisimp simp: inj_on_def)
    by (metis fim homeomorphism_apply1 homhk image_subset_iff inj_onD injf)
  ultimately have ope_hf: openin (top_of_set U) ((h ∘ f) ' S)
    using invariance_of_domain_subspaces [OF ope ⟨subspace U⟩ ⟨subspace U⟩]
  by blast
  have (h ∘ f) ' S ⊆ T
    using fim homeomorphism_image1 homhk by fastforce
  then have dim ((h ∘ f) ' S) ≤ dim T
    by (rule dim_subset)
  also have dim ((h ∘ f) ' S) = dim U
    using ⟨S ≠ {}⟩ ⟨subspace U⟩
    by (blast intro: dim_openin ope_hf)
  finally show False
    using ⟨dim V < dim U⟩ ⟨dim T = dim V⟩ by simp
qed
then show ?thesis
  using not_less by blast
qed

corollary invariance_of_domain_affine_sets:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
    and aff: affine U affine V aff_dim V ≤ aff_dim U
    and contf: continuous_on S f and fim: f ' S ⊆ V
    and injf: inj_on f S
  shows openin (top_of_set V) (f ' S)
proof (cases S = {})
case False
  obtain a b where a ∈ S a ∈ U b ∈ V
    using False fim ope openin_contains_cball by fastforce
  have openin (top_of_set ((+) (- b) ' V)) (((+) (- b) ∘ f ∘ (+) a) ' (+) (- a)
' S)
  proof (rule invariance_of_domain_subspaces)
    show openin (top_of_set ((+) (- a) ' U)) ((+) (- a) ' S)
      by (metis ope homeomorphism_imp_open_map homeomorphism_translation_
translation_galois)
    show subspace ((+) (- a) ' U)
      by (simp add: ⟨a ∈ U⟩ affine_diffs_subspace_subtract ⟨affine U⟩ cong:
image_cong_simp)
    show subspace ((+) (- b) ' V)

```

```

    by (simp add: ⟨b ∈ V⟩ affine_diffs_subspace_subtract ⟨affine V⟩ cong:
image_cong_simp)
  show dim ((+) (- b) ' V) ≤ dim ((+) (- a) ' U)
  by (metis ⟨a ∈ U⟩ ⟨b ∈ V⟩ aff_dim_eq_dim affine_hull_eq aff_of_nat_le_iff)
  show continuous_on ((+) (- a) ' S) ((+) (- b) ∘ f ∘ (+) a)
  by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
  show ((+) (- b) ∘ f ∘ (+) a) ' (+) (- a) ' S ⊆ (+) (- b) ' V
  using fim by auto
  show inj_on ((+) (- b) ∘ f ∘ (+) a) ((+) (- a) ' S)
  by (auto simp: inj_on_def) (meson inj_onD injf)
qed
then show ?thesis
  by (metis (no_types, lifting) homeomorphism_imp_open_map homeomor-
phism_translation image_comp translation_galois)
qed auto

corollary invariance_of_dimension_affine_sets:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
  and aff: affine U affine V
  and contf: continuous_on S f and fim: f ' S ⊆ V
  and injf: inj_on f S and S ≠ {}
  shows aff_dim U ≤ aff_dim V
proof -
  obtain a b where a ∈ S a ∈ U b ∈ V
  using ⟨S ≠ {}⟩ fim ope openin_contains_cball by fastforce
  have dim ((+) (- a) ' U) ≤ dim ((+) (- b) ' V)
  proof (rule invariance_of_dimension_subspaces)
    show openin (top_of_set ((+) (- a) ' U)) ((+) (- a) ' S)
    by (metis ope homeomorphism_imp_open_map homeomorphism_translation
translation_galois)
    show subspace ((+) (- a) ' U)
    by (simp add: ⟨a ∈ U⟩ affine_diffs_subspace_subtract ⟨affine U⟩ cong:
image_cong_simp)
    show subspace ((+) (- b) ' V)
    by (simp add: ⟨b ∈ V⟩ affine_diffs_subspace_subtract ⟨affine V⟩ cong:
image_cong_simp)
    show continuous_on ((+) (- a) ' S) ((+) (- b) ∘ f ∘ (+) a)
    by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
    show ((+) (- b) ∘ f ∘ (+) a) ' (+) (- a) ' S ⊆ (+) (- b) ' V
    using fim by auto
    show inj_on ((+) (- b) ∘ f ∘ (+) a) ((+) (- a) ' S)
    by (auto simp: inj_on_def) (meson inj_onD injf)
  qed (use ⟨S ≠ {}⟩ in auto)
then show ?thesis
  by (metis ⟨a ∈ U⟩ ⟨b ∈ V⟩ aff_dim_eq_dim affine_hull_eq aff_of_nat_le_iff)
qed

```

corollary *invariance_of_dimension:*

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *contf*: *continuous_on* S f **and** *open* S
and *injf*: *inj_on* f S **and** $S \neq \{\}$
shows $\text{DIM}('a) \leq \text{DIM}('b)$
using *invariance_of_dimension_subspaces* [*of UNIV* S *UNIV* f] *assms*
by *auto*

corollary *continuous_injective_image_subspace_dim_le:*

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *subspace* S *subspace* T
and *contf*: *continuous_on* S f **and** *fim*: $f \restriction S \subseteq T$
and *injf*: *inj_on* f S
shows $\dim S \leq \dim T$
apply (*rule invariance_of_dimension_subspaces* [*of* S S f])
using *assms* **by** (*auto simp: subspace_affine*)

lemma *invariance_of_dimension_convex_domain:*

fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *convex* S
and *contf*: *continuous_on* S f **and** *fim*: $f \restriction S \subseteq \text{affine hull } T$
and *injf*: *inj_on* f S
shows $\text{aff_dim } S \leq \text{aff_dim } T$

proof (*cases* $S = \{\}$)

case *True*

then show *?thesis* **by** (*simp add: aff_dim_geq*)

next

case *False*

have $\text{aff_dim } (\text{affine hull } S) \leq \text{aff_dim } (\text{affine hull } T)$

proof (*rule invariance_of_dimension_affine_sets*)

show *openin* (*top_of_set* (*affine hull* S)) (*rel_interior* S)

by (*simp add: openin_rel_interior*)

show *continuous_on* (*rel_interior* S) f

using *contf* *continuous_on_subset* *rel_interior_subset* **by** *blast*

show $f \restriction \text{rel_interior } S \subseteq \text{affine hull } T$

using *fim* *rel_interior_subset* **by** *blast*

show *inj_on* f (*rel_interior* S)

using *inj_on_subset* *injf* *rel_interior_subset* **by** *blast*

show *rel_interior* $S \neq \{\}$

by (*simp add: False_convex_S*) *rel_interior_eq_empty*)

qed *auto*

then show *?thesis*

by *simp*

qed

lemma *homeomorphic_convex_sets_le:*

assumes *convex* S S *homeomorphic* T

shows $\text{aff_dim } S \leq \text{aff_dim } T$

```

proof –
  obtain  $h\ k$  where  $homhk$ : homeomorphism  $S\ T\ h\ k$ 
    using homeomorphic_def assms by blast
  show ?thesis
  proof (rule invariance_of_dimension_convex_domain [OF  $\langle convex\ S \rangle$ ])
    show continuous_on  $S\ h$ 
      using homeomorphism_def  $homhk$  by blast
    show  $h\ ' S \subseteq \text{affine hull } T$ 
      by (metis homeomorphism_def  $homhk$  hull_subset)
    show inj_on  $h\ S$ 
      by (meson homeomorphism_apply1  $homhk$  inj_on_inverseI)
  qed
qed

lemma homeomorphic_convex_sets:
  assumes convex  $S\ convex\ T\ S\ \text{homeomorphic}\ T$ 
  shows aff_dim  $S = \text{aff\_dim } T$ 
  by (meson assms dual_order.antisym homeomorphic_convex_sets_le homeomorphic_sym)

lemma homeomorphic_convex_compact_sets_eq:
  assumes convex  $S\ compact\ S\ convex\ T\ compact\ T$ 
  shows  $S\ \text{homeomorphic}\ T \longleftrightarrow \text{aff\_dim } S = \text{aff\_dim } T$ 
  by (meson assms homeomorphic_convex_compact_sets homeomorphic_convex_sets)

lemma invariance_of_domain_gen:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes open  $S\ continuous\_on\ S\ f\ inj\_on\ f\ S\ DIM('b) \leq DIM('a)$ 
  shows open ( $f\ ' S$ )
  using invariance_of_domain_subspaces [of  $UNIV\ S\ UNIV\ f$ ] assms by auto

lemma injective_into_1d_imp_open_map_UNIV:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow \text{real}$ 
  assumes open  $T\ continuous\_on\ S\ f\ inj\_on\ f\ S\ T \subseteq S$ 
  shows open ( $f\ ' T$ )
  apply (rule invariance_of_domain_gen [OF  $\langle open\ T \rangle$ ])
  using assms apply (auto simp: elim: continuous_on_subset subset_inj_on)
  done

lemma continuous_on_inverse_open:
  fixes  $f :: 'a::\text{euclidean\_space} \Rightarrow 'b::\text{euclidean\_space}$ 
  assumes open  $S\ continuous\_on\ S\ f\ DIM('b) \leq DIM('a)$  and  $gf: \bigwedge x. x \in S \implies g(f\ x) = x$ 
  shows continuous_on ( $f\ ' S$ )  $g$ 
proof (clarsimp simp add: continuous_openin_preimage_eq)
  fix  $T :: 'a\ set$ 
  assume open  $T$ 
  have  $eq: f\ ' S \cap g\ -' T = f\ ' (S \cap T)$ 
  by (auto simp: gf)

```

```

have openin (top_of_set (f ` S)) (f ` (S ∩ T))
proof (rule open_openin_trans [OF invariance_of_domain_gen])
  show inj_on f S
    using inj_on_inverseI gf by auto
  show open (f ` (S ∩ T))
    by (meson ⟨inj_on f S⟩ ⟨open T⟩ assms(1-3) continuous_on_subset inf_le1
inj_on_subset invariance_of_domain_gen open_Int)
qed (use assms in auto)
then show openin (top_of_set (f ` S)) (f ` S ∩ g - ` T)
  by (simp add: eq)
qed

```

```

lemma invariance_of_domain_homeomorphism:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S
  obtains g where homeomorphism S (f ` S) f g
proof
  show homeomorphism S (f ` S) f (inv_into S f)
    by (simp add: assms continuous_on_inverse_open homeomorphism_def)
qed

```

```

corollary invariance_of_domain_homeomorphic:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S
  shows S homeomorphic (f ` S)
  using invariance_of_domain_homeomorphism [OF assms]
  by (meson homeomorphic_def)

```

```

lemma continuous_image_subset_interior:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes continuous_on S f inj_on f S DIM('b) ≤ DIM('a)
  shows f ` (interior S) ⊆ interior(f ` S)
proof (rule interior_maximal)
  show f ` interior S ⊆ f ` S
    by (simp add: image_mono interior_subset)
  show open (f ` interior S)
    using assms
    by (auto simp: subset_inj_on interior_subset continuous_on_subset invariance_of_domain_gen)
qed

```

```

lemma homeomorphic_interiors_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T and dimeq: DIM('a) = DIM('b)
  shows (interior S) homeomorphic (interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g

```

```

assume  $S: \forall x \in S. f x \in T \wedge g (f x) = x$  and  $T: \forall y \in T. g y \in S \wedge f (g y) = y$ 
and  $\text{contf}: \text{continuous\_on } S f$  and  $\text{contg}: \text{continuous\_on } T g$ 
then have  $fST: f ' S = T$  and  $gTS: g ' T = S$  and  $\text{inj\_on } f S \text{ inj\_on } g T$ 
by (auto simp: inj_on_def intro: rev_image_eqI) metis+
have  $\text{fim}: f ' \text{interior } S \subseteq \text{interior } T$ 
using  $\text{continuous\_image\_subset\_interior } [OF \text{ contf } \langle \text{inj\_on } f S \rangle]$  dimeq fST
by simp
have  $\text{gim}: g ' \text{interior } T \subseteq \text{interior } S$ 
using  $\text{continuous\_image\_subset\_interior } [OF \text{ contg } \langle \text{inj\_on } g T \rangle]$  dimeq gTS
by simp
show  $\text{homeomorphism } (\text{interior } S) (\text{interior } T) f g$ 
unfolding  $\text{homeomorphism\_def}$ 
proof (intro conjI ballI)
show  $\bigwedge x. x \in \text{interior } S \implies g (f x) = x$ 
by (meson  $\langle \forall x \in S. f x \in T \wedge g (f x) = x \rangle \text{subsetD interior\_subset}$ )
have  $\text{interior } T \subseteq f ' \text{interior } S$ 
proof
fix  $x$  assume  $x \in \text{interior } T$ 
then have  $g x \in \text{interior } S$ 
using  $\text{gim}$  by blast
then show  $x \in f ' \text{interior } S$ 
by (metis  $T \langle x \in \text{interior } T \rangle \text{image\_iff interior\_subset subsetCE}$ )
qed
then show  $f ' \text{interior } S = \text{interior } T$ 
using  $\text{fim}$  by blast
show  $\text{continuous\_on } (\text{interior } S) f$ 
by (metis  $\text{interior\_subset continuous\_on\_subset contf}$ )
show  $\bigwedge y. y \in \text{interior } T \implies f (g y) = y$ 
by (meson  $T \text{subsetD interior\_subset}$ )
have  $\text{interior } S \subseteq g ' \text{interior } T$ 
proof
fix  $x$  assume  $x \in \text{interior } S$ 
then have  $f x \in \text{interior } T$ 
using  $\text{fim}$  by blast
then show  $x \in g ' \text{interior } T$ 
by (metis  $S \langle x \in \text{interior } S \rangle \text{image\_iff interior\_subset subsetCE}$ )
qed
then show  $g ' \text{interior } T = \text{interior } S$ 
using  $\text{gim}$  by blast
show  $\text{continuous\_on } (\text{interior } T) g$ 
by (metis  $\text{interior\_subset continuous\_on\_subset contg}$ )
qed
qed

```

lemma $\text{homeomorphic_open_imp_same_dimension}$:

```

fixes  $S :: 'a::\text{euclidean\_space set}$  and  $T :: 'b::\text{euclidean\_space set}$ 
assumes  $S \text{homeomorphic } T \text{ open } S \ S \neq \{\} \text{ open } T \ T \neq \{\}$ 
shows  $\text{DIM}('a) = \text{DIM}('b)$ 
using assms

```



```

    apply (simp add: homeomorphic_minimal)
    apply (rule order_antisym; metis inj_onI invariance_of_dimension)
    done

proposition homeomorphic_interiors:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $S\ homeomorphic\ T$  interior  $S = \{\}$   $\longleftrightarrow$  interior  $T = \{\}$ 
  shows (interior  $S$ ) homeomorphic (interior  $T$ )
proof (cases interior  $T = \{\}$ )
  case True
    with assms show ?thesis by auto
  next
  case False
    then have  $DIM('a) = DIM('b)$ 
      using assms
      apply (simp add: homeomorphic_minimal)
      apply (rule order_antisym; metis continuous_on_subset inj_onI inj_on_subset
interior_subset invariance_of_dimension open_interior)
      done
    then show ?thesis
      by (rule homeomorphic_interiors_same_dimension [OF  $\langle S\ homeomorphic\ T \rangle$ ])
  qed

lemma homeomorphic_frontiers_same_dimension:
  fixes  $S :: 'a::euclidean\_space\ set$  and  $T :: 'b::euclidean\_space\ set$ 
  assumes  $S\ homeomorphic\ T$  closed  $S$  closed  $T$  and dimeq:  $DIM('a) = DIM('b)$ 
  shows (frontier  $S$ ) homeomorphic (frontier  $T$ )
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix  $f\ g$ 
  assume  $S: \forall x \in S. f\ x \in T \wedge g\ (f\ x) = x$  and  $T: \forall y \in T. g\ y \in S \wedge f\ (g\ y) = y$ 
    and contf: continuous_on  $S\ f$  and contg: continuous_on  $T\ g$ 
  then have fST:  $f\ 'S = T$  and gTS:  $g\ 'T = S$  and inj_on  $f\ S$  inj_on  $g\ T$ 
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have  $g\ 'interior\ T \subseteq interior\ S$ 
    using continuous_image_subset_interior [OF contg  $\langle inj\_on\ g\ T \rangle$ ] dimeq gTS
  by simp
  then have fim:  $f\ 'frontier\ S \subseteq frontier\ T$ 
    apply (simp add: frontier_def)
    using continuous_image_subset_interior assms(2) assms(3)  $S$  by auto
  have  $f\ 'interior\ S \subseteq interior\ T$ 
    using continuous_image_subset_interior [OF contf  $\langle inj\_on\ f\ S \rangle$ ] dimeq fST
  by simp
  then have gim:  $g\ 'frontier\ T \subseteq frontier\ S$ 
    apply (simp add: frontier_def)
    using continuous_image_subset_interior  $T$  assms(2) assms(3) by auto
  show homeomorphism (frontier  $S$ ) (frontier  $T$ )  $f\ g$ 
    unfolding homeomorphism_def

```

```

proof (intro conjI ballI)
  show  $gf: \bigwedge x. x \in \text{frontier } S \implies g(f\ x) = x$ 
    by (simp add: S assms(2) frontier_def)
  show  $fg: \bigwedge y. y \in \text{frontier } T \implies f(g\ y) = y$ 
    by (simp add: T assms(3) frontier_def)
  have  $\text{frontier } T \subseteq f\ ` \text{frontier } S$ 
proof
  fix  $x$  assume  $x \in \text{frontier } T$ 
  then have  $g\ x \in \text{frontier } S$ 
    using  $gim$  by blast
  then show  $x \in f\ ` \text{frontier } S$ 
    by (metis fg  $\langle x \in \text{frontier } T \rangle$  imageI)
qed
then show  $f\ ` \text{frontier } S = \text{frontier } T$ 
  using  $fim$  by blast
show continuous_on (frontier S) f
  by (metis Diff_subset assms(2) closure_eq contf continuous_on_subset frontier_def)
have  $\text{frontier } S \subseteq g\ ` \text{frontier } T$ 
proof
  fix  $x$  assume  $x \in \text{frontier } S$ 
  then have  $f\ x \in \text{frontier } T$ 
    using  $fim$  by blast
  then show  $x \in g\ ` \text{frontier } T$ 
    by (metis gf  $\langle x \in \text{frontier } S \rangle$  imageI)
qed
then show  $g\ ` \text{frontier } T = \text{frontier } S$ 
  using  $gim$  by blast
show continuous_on (frontier T) g
  by (metis Diff_subset assms(3) closure_closed contg continuous_on_subset frontier_def)
qed
qed

lemma homeomorphic_frontiers:
  fixes  $S :: 'a::\text{euclidean\_space set}$  and  $T :: 'b::\text{euclidean\_space set}$ 
  assumes  $S$  homeomorphic  $T$  closed  $S$  closed  $T$ 
     $\text{interior } S = \{\} \longleftrightarrow \text{interior } T = \{\}$ 
  shows (frontier  $S$ ) homeomorphic (frontier  $T$ )
proof (cases interior  $T = \{\}$ )
  case True
  then show ?thesis
    by (metis Diff_empty assms closure_eq frontier_def)
next
  case False
  show ?thesis
    apply (rule homeomorphic_frontiers_same_dimension)
    apply (simp_all add: assms)
    using False assms homeomorphic_interiors homeomorphic_open_imp_same_dimension

```

by blast
qed

lemma *continuous_image_subset_rel_interior*:
fixes $f :: 'a::\text{euclidean_space} \Rightarrow 'b::\text{euclidean_space}$
assumes *contf*: *continuous_on* S f **and** *injf*: *inj_on* f S **and** *fim*: $f \text{ ` } S \subseteq T$
and *TS*: *aff_dim* $T \leq \text{aff_dim } S$
shows $f \text{ ` } (\text{rel_interior } S) \subseteq \text{rel_interior}(f \text{ ` } S)$
proof (*rule rel_interior_maximal*)
show $f \text{ ` } \text{rel_interior } S \subseteq f \text{ ` } S$
by(*simp add: image_mono rel_interior_subset*)
show *openin* (*top_of_set* (*affine hull* $f \text{ ` } S$)) ($f \text{ ` } \text{rel_interior } S$)
proof (*rule invariance_of_domain_affine_sets*)
show *openin* (*top_of_set* (*affine hull* S)) (*rel_interior* S)
by (*simp add: openin_rel_interior*)
show *aff_dim* (*affine hull* $f \text{ ` } S$) $\leq \text{aff_dim}$ (*affine hull* S)
by (*metis aff_dim_affine_hull aff_dim_subset fim TS order_trans*)
show $f \text{ ` } \text{rel_interior } S \subseteq \text{affine hull } f \text{ ` } S$
by (*meson* $\langle f \text{ ` } \text{rel_interior } S \subseteq f \text{ ` } S \rangle \text{ hull_subset order_trans}$)
show *continuous_on* (*rel_interior* S) f
using *contf continuous_on_subset rel_interior_subset* **by** blast
show *inj_on* f (*rel_interior* S)
using *inj_on_subset injf rel_interior_subset* **by** blast
qed *auto*
qed

lemma *homeomorphic_rel_interiors_same_dimension*:
fixes $S :: 'a::\text{euclidean_space set}$ **and** $T :: 'b::\text{euclidean_space set}$
assumes S *homeomorphic* T **and** *aff*: *aff_dim* $S = \text{aff_dim } T$
shows (*rel_interior* S) *homeomorphic* (*rel_interior* T)
using *assms* [*unfolded homeomorphic_minimal*]
unfolding *homeomorphic_def*
proof (*clarify elim!: ex_forward*)
fix $f g$
assume $S: \forall x \in S. f x \in T \wedge g (f x) = x$ **and** $T: \forall y \in T. g y \in S \wedge f (g y) = y$
and *contf*: *continuous_on* S f **and** *contg*: *continuous_on* T g
then have *fST*: $f \text{ ` } S = T$ **and** *gTS*: $g \text{ ` } T = S$ **and** *inj_on* f S *inj_on* g T
by (*auto simp: inj_on_def intro: rev_image_eqI*) *metis* +
have *fim*: $f \text{ ` } \text{rel_interior } S \subseteq \text{rel_interior } T$
by (*metis* $\langle \text{inj_on } f \text{ ` } S \rangle \text{ aff contf continuous_image_subset_rel_interior fST order_refl}$)
have *gim*: $g \text{ ` } \text{rel_interior } T \subseteq \text{rel_interior } S$
by (*metis* $\langle \text{inj_on } g \text{ ` } T \rangle \text{ aff contg continuous_image_subset_rel_interior gTS order_refl}$)
show *homeomorphism* (*rel_interior* S) (*rel_interior* T) $f g$
unfolding *homeomorphism_def*
proof (*intro conjI ballI*)
show *gf*: $\bigwedge x. x \in \text{rel_interior } S \implies g (f x) = x$
using S *rel_interior_subset* **by** blast

```

show fg:  $\bigwedge y. y \in \text{rel\_interior } T \implies f (g y) = y$ 
  using T mem_rel_interior_ball by blast
have rel_interior T  $\subseteq$  f ' rel_interior S
proof
  fix x assume x  $\in$  rel_interior T
  then have g x  $\in$  rel_interior S
    using gim by blast
  then show x  $\in$  f ' rel_interior S
    by (metis fg  $\langle x \in \text{rel\_interior } T \rangle$  imageI)
qed
moreover have f ' rel_interior S  $\subseteq$  rel_interior T
  by (metis  $\langle \text{inj\_on } f \text{ } S \rangle$  aff contf continuous_image_subset_rel_interior fST
order_refl)
ultimately show f ' rel_interior S = rel_interior T
  by blast
show continuous_on (rel_interior S) f
  using contf continuous_on_subset rel_interior_subset by blast
have rel_interior S  $\subseteq$  g ' rel_interior T
proof
  fix x assume x  $\in$  rel_interior S
  then have f x  $\in$  rel_interior T
    using fim by blast
  then show x  $\in$  g ' rel_interior T
    by (metis gf  $\langle x \in \text{rel\_interior } S \rangle$  imageI)
qed
then show g ' rel_interior T = rel_interior S
  using gim by blast
show continuous_on (rel_interior T) g
  using contg contg continuous_on_subset rel_interior_subset by blast
qed
qed

lemma homeomorphic_rel_interiors:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T rel_interior S = {}  $\longleftrightarrow$  rel_interior T = {}
  shows (rel_interior S) homeomorphic (rel_interior T)
proof (cases rel_interior T = {})
case True
  with assms show ?thesis by auto
next
case False
  obtain f g
  where S:  $\forall x \in S. f x \in T \wedge g (f x) = x$  and T:  $\forall y \in T. g y \in S \wedge f (g y) = y$ 
  and contf: continuous_on S f and contg: continuous_on T g
  using assms [unfolded homeomorphic_minimal] by auto
have aff_dim (affine hull S)  $\leq$  aff_dim (affine hull T)
  apply (rule invariance_of_dimension_affine_sets [of _ rel_interior S _ f])
  apply (simp_all add: openin_rel_interior False assms)
  using contf continuous_on_subset rel_interior_subset apply blast

```

```

    apply (meson S hull_subset image_subsetI rel_interior_subset rev_subsetD)
    apply (metis S inj_on_inverseI inj_on_subset rel_interior_subset)
  done
moreover have aff_dim (affine hull T) ≤ aff_dim (affine hull S)
  apply (rule invariance_of_dimension_affine_sets [of _ rel_interior T _ g])
    apply (simp_all add: openin_rel_interior False assms)
  using contg_continuous_on_subset rel_interior_subset apply blast
    apply (meson T hull_subset image_subsetI rel_interior_subset rev_subsetD)
    apply (metis T inj_on_inverseI inj_on_subset rel_interior_subset)
  done
ultimately have aff_dim S = aff_dim T by force
then show ?thesis
  by (rule homeomorphic_rel_interiors_same_dimension [OF ⟨S homeomorphic
T⟩])
qed

```

```

lemma homeomorphic_rel_boundaries_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T and aff: aff_dim S = aff_dim T
  shows (S - rel_interior S) homeomorphic (T - rel_interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g
  assume S: ∀ x ∈ S. f x ∈ T ∧ g (f x) = x and T: ∀ y ∈ T. g y ∈ S ∧ f (g y) = y
    and contf: continuous_on S f and contg: continuous_on T g
  then have fST: f ' S = T and gTS: g ' T = S and inj_on f S inj_on g T
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have fim: f ' rel_interior S ⊆ rel_interior T
    by (metis ⟨inj_on f S⟩ aff contf continuous_image_subset_rel_interior fST
order_refl)
  have gim: g ' rel_interior T ⊆ rel_interior S
    by (metis ⟨inj_on g T⟩ aff contg continuous_image_subset_rel_interior gTS
order_refl)
  show homeomorphism (S - rel_interior S) (T - rel_interior T) f g
    unfolding homeomorphism_def
  proof (intro conjI ballI)
    show gf: ∧ x. x ∈ S - rel_interior S ⇒ g (f x) = x
      using S rel_interior_subset by blast
    show fg: ∧ y. y ∈ T - rel_interior T ⇒ f (g y) = y
      using T mem_rel_interior_ball by blast
    show f '(S - rel_interior S) = T - rel_interior T
      using S fST fim gim by auto
    show continuous_on (S - rel_interior S) f
      using contf continuous_on_subset rel_interior_subset by blast
    show g '(T - rel_interior T) = S - rel_interior S
      using T gTS gim fim by auto
    show continuous_on (T - rel_interior T) g

```

```

    using contg continuous_on_subset rel_interior_subset by blast
qed
qed

lemma homeomorphic_rel_boundaries:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T rel_interior S = {}  $\longleftrightarrow$  rel_interior T = {}
  shows (S - rel_interior S) homeomorphic (T - rel_interior T)
proof (cases rel_interior T = {})
  case True
  with assms show ?thesis by auto
next
  case False
  obtain f g
  where S:  $\forall x \in S. f x \in T \wedge g (f x) = x$  and T:  $\forall y \in T. g y \in S \wedge f (g y) = y$ 
  and conf: continuous_on S f and contg: continuous_on T g
  using assms [unfolded homeomorphic_minimal] by auto
  have aff_dim (affine hull S)  $\leq$  aff_dim (affine hull T)
  apply (rule invariance_of_dimension_affine_sets [of _ rel_interior S _ f])
  apply (simp_all add: openin_rel_interior False assms)
  using contf continuous_on_subset rel_interior_subset apply blast
  apply (meson S hull_subset image_subsetI rel_interior_subset rev_subsetD)
  apply (metis S inj_on_inverseI inj_on_subset rel_interior_subset)
  done
  moreover have aff_dim (affine hull T)  $\leq$  aff_dim (affine hull S)
  apply (rule invariance_of_dimension_affine_sets [of _ rel_interior T _ g])
  apply (simp_all add: openin_rel_interior False assms)
  using contg continuous_on_subset rel_interior_subset apply blast
  apply (meson T hull_subset image_subsetI rel_interior_subset rev_subsetD)
  apply (metis T inj_on_inverseI inj_on_subset rel_interior_subset)
  done
  ultimately have aff_dim S = aff_dim T by force
  then show ?thesis
  by (rule homeomorphic_rel_boundaries_same_dimension [OF  $\langle S \text{ homeomorphic } T \rangle$ ])
qed

proposition uniformly_continuous_homeomorphism_UNIV_trivial:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'a
  assumes conf: uniformly_continuous_on S f and hom: homeomorphism S
  UNIV f g
  shows S = UNIV
proof (cases S = {})
  case True
  then show ?thesis
  by (metis UNIV_I hom empty_iff homeomorphism_def image_eqI)
next
  case False
  have inj g

```

```

    by (metis UNIV_I hom homeomorphism_apply2 injI)
  then have open (g ' UNIV)
    by (blast intro: invariance_of_domain hom homeomorphism_cont2)
  then have open S
    using hom homeomorphism_image2 by blast
  moreover have complete S
    unfolding complete_def
  proof clarify
    fix  $\sigma$ 
    assume  $\sigma: \forall n. \sigma\ n \in S$  and Cauchy  $\sigma$ 
    have Cauchy (f o  $\sigma$ )
      using uniformly_continuous_imp_Cauchy_continuous (Cauchy  $\sigma$ )  $\sigma$  contf
      unfolding Cauchy_continuous_on_def by blast
    then obtain l where (f o  $\sigma$ )  $\longrightarrow$  l
      by (auto simp: convergent_eq_Cauchy [symmetric])
    show  $\exists l \in S. \sigma \longrightarrow l$ 
  proof
    show g l  $\in S$ 
      using hom homeomorphism_image2 by blast
    have (g o (f o  $\sigma$ ))  $\longrightarrow$  g l
      by (meson UNIV_I (f o  $\sigma$ )  $\longrightarrow$  l) continuous_on_sequentially hom
      homeomorphism_cont2)
    then show  $\sigma \longrightarrow g\ l$ 
  proof -
    have  $\forall n. \sigma\ n = (g \circ (f \circ \sigma))\ n$ 
      by (metis (no_types)  $\sigma$  comp_eq_dest_lhs hom homeomorphism_apply1)
    then show ?thesis
      by (metis (no_types) LIMSEQ_iff (g o (f o  $\sigma$ ))  $\longrightarrow$  g l)
  qed
qed
qed
then have closed S
  by (simp add: complete_eq_closed)
ultimately show ?thesis
  using clopen [of S] False by simp
qed

proposition invariance_of_domain_sphere_affine_set_gen:
  fixes f :: 'a::euclidean_space  $\Rightarrow$  'b::euclidean_space
  assumes contf: continuous_on S f and injf: inj_on f S and fim: f ' S  $\subseteq$  T
    and U: bounded U convex U
    and affine T and affTU: aff_dim T < aff_dim U
    and ope: openin (top_of_set (rel_frontier U)) S
  shows openin (top_of_set T) (f ' S)
proof (cases rel_frontier U = {})
  case True
    then show ?thesis
      using ope openin_subset by force
  next

```

```

case False
obtain b c where b: b ∈ rel_frontier U and c: c ∈ rel_frontier U and b ≠ c
  using ⟨bounded U⟩ rel_frontier_not_sing [of U] subset_singletonD False by
fastforce
obtain V :: 'a set where affine V and affV: aff_dim V = aff_dim U - 1
proof (rule choose_affine_subset [OF affine_UNIV])
  show - 1 ≤ aff_dim U - 1
  by (metis aff_dim_empty aff_dim_geq aff_dim_negative_iff affTU diff_0
diff_right_mono not_le)
  show aff_dim U - 1 ≤ aff_dim (UNIV::'a set)
  by (metis aff_dim_UNIV aff_dim_le_DIM le_cases not_le zle_diff1_eq)
qed auto
have SU: S ⊆ rel_frontier U
  using ope openin_imp_subset by auto
have homb: rel_frontier U - {b} homeomorphic V
and homc: rel_frontier U - {c} homeomorphic V
  using homeomorphic_punctured_sphere_affine_gen [of U _ V]
  by (simp_all add: ⟨affine V⟩ affV U b c)
then obtain g h j k
  where gh: homeomorphism (rel_frontier U - {b}) V g h
  and jk: homeomorphism (rel_frontier U - {c}) V j k
  by (auto simp: homeomorphic_def)
with SU have hgsub: (h ' g ' (S - {b})) ⊆ S and kjsub: (k ' j ' (S - {c})) ⊆ S
  by (simp_all add: homeomorphism_def subset_eq)
have [simp]: aff_dim T ≤ aff_dim V
  by (simp add: affTU affV)
have openin (top_of_set T) ((f ∘ h) ' g ' (S - {b}))
proof (rule invariance_of_domain_affine_sets [OF _ ⟨affine V⟩])
  show openin (top_of_set V) (g ' (S - {b}))
  apply (rule homeomorphism_imp_open_map [OF gh])
  by (meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans
order_refl)
  show continuous_on (g ' (S - {b})) (f ∘ h)
  apply (rule continuous_on_compose)
  apply (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
gh set_eq_subset)
  using conf continuous_on_subset hgsub by blast
  show inj_on (f ∘ h) (g ' (S - {b}))
  using kjsub
  apply (clarsimp simp add: inj_on_def)
  by (metis SU b homeomorphism_def inj_onD injf insert_Diff insert_iff gh
rev_subsetD)
  show (f ∘ h) ' g ' (S - {b}) ⊆ T
  by (metis fim image_comp image_mono hgsub subset_trans)
qed (auto simp: assms)
moreover
have openin (top_of_set T) ((f ∘ k) ' j ' (S - {c}))
proof (rule invariance_of_domain_affine_sets [OF _ ⟨affine V⟩])
  show openin (top_of_set V) (j ' (S - {c}))

```



```

    apply (rule homeomorphism_imp_open_map [OF jk])
  by (meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans
order_refl)
  show continuous_on (j ' (S - {c})) (f ∘ k)
    apply (rule continuous_on_compose)
    apply (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
jk set_eq_subset)
    using contf continuous_on_subset kjsub by blast
  show inj_on (f ∘ k) (j ' (S - {c}))
    using kjsub
    apply (clarsimp simp add: inj_on_def)
    by (metis SU c homeomorphism_def inj_onD injf insert_Diff insert_iff jk
rev_subsetD)
  show (f ∘ k) ' j ' (S - {c}) ⊆ T
    by (metis fim image_comp image_mono kjsub subset_trans)
qed (auto simp: assms)
ultimately have openin (top_of_set T) ((f ∘ h) ' g ' (S - {b}) ∪ ((f ∘ k) ' j
' (S - {c})))
  by (rule openin_Un)
moreover have (f ∘ h) ' g ' (S - {b}) = f ' (S - {b})
proof -
  have h ' g ' (S - {b}) = (S - {b})
  proof
    show h ' g ' (S - {b}) ⊆ S - {b}
      using homeomorphism_apply1 [OF gh] SU
      by (fastforce simp add: image_iff image_subset_iff)
    show S - {b} ⊆ h ' g ' (S - {b})
      apply clarify
      by (metis SU subsetD homeomorphism_apply1 [OF gh] image_iff mem-
ber_remove remove_def)
  qed
  then show ?thesis
    by (metis image_comp)
qed
moreover have (f ∘ k) ' j ' (S - {c}) = f ' (S - {c})
proof -
  have k ' j ' (S - {c}) = (S - {c})
  proof
    show k ' j ' (S - {c}) ⊆ S - {c}
      using homeomorphism_apply1 [OF jk] SU
      by (fastforce simp add: image_iff image_subset_iff)
    show S - {c} ⊆ k ' j ' (S - {c})
      apply clarify
      by (metis SU subsetD homeomorphism_apply1 [OF jk] image_iff mem-
ber_remove remove_def)
  qed
  then show ?thesis
    by (metis image_comp)
qed

```

```

    moreover have  $f^{-1}(S - \{b\}) \cup f^{-1}(S - \{c\}) = f^{-1}(S)$ 
    using  $\langle b \neq c \rangle$  by blast
    ultimately show ?thesis
    by simp
qed

```

```

lemma invariance_of_domain_sphere_affine_set:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $\text{contf: continuous\_on } S \text{ f}$  and  $\text{injf: inj\_on } f \text{ } S$  and  $\text{fim: } f^{-1} S \subseteq T$ 
    and  $r \neq 0$  affine  $T$  and  $\text{affTU: aff\_dim } T < \text{DIM}('a)$ 
    and  $\text{oep: openin (top\_of\_set (sphere a r)) } S$ 
  shows  $\text{openin (top\_of\_set } T) (f^{-1} S)$ 
proof (cases  $\text{sphere a r} = \{\}$ )
  case True
  then show ?thesis
    using ope openin_subset by force
next
  case False
  show ?thesis
  proof (rule invariance_of_domain_sphere_affine_set_gen [OF contf injf fim
    bounded_cball convex_cball  $\langle \text{affine } T \rangle$ ])
    show  $\text{aff\_dim } T < \text{aff\_dim (cball a r)}$ 
    by (metis False affTU aff_dim_cball assms(4) linorder_cases sphere_empty)
    show  $\text{openin (top\_of\_set (rel\_frontier (cball a r))) } S$ 
    by (simp add:  $\langle r \neq 0 \rangle$  ope)
  qed
qed

```

```

lemma no_embedding_sphere_lowdim:
  fixes  $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$ 
  assumes  $\text{contf: continuous\_on (sphere a r) } f$  and  $\text{injf: inj\_on } f \text{ (sphere a r)}$ 
  and  $r > 0$ 
  shows  $\text{DIM}('a) \leq \text{DIM}('b)$ 
proof -
  have False if  $\text{DIM}('a) > \text{DIM}('b)$ 
  proof -
    have compact  $(f^{-1} \text{ sphere a r})$ 
    using compact_continuous_image
    by (simp add: compact_continuous_image contf)
    then have  $\neg \text{open } (f^{-1} \text{ sphere a r})$ 
    using compact_open
    by (metis assms(3) image_is_empty not_less_iff_gr_or_eq sphere_eq_empty)
    then show False
    using invariance_of_domain_sphere_affine_set [OF contf injf subset_UNIV]
     $\langle r > 0 \rangle$ 
    by (metis aff_dim_UNIV affine_UNIV less_irrefl of_nat_less_iff open_openin
    openin_subtopology_self subtopology_UNIV that)
  qed
  then show ?thesis

```

```

    using not_less by blast
qed

lemma empty_interior_lowdim_gen:
  fixes S :: 'N::euclidean_space set and T :: 'M::euclidean_space set
  assumes dim: DIM('M) < DIM('N) and ST: S homeomorphic T
  shows interior S = {}
proof -
  obtain h :: 'M  $\Rightarrow$  'N where linear h  $\wedge$  x. norm(h x) = norm x
  by (rule isometry_subset_subspace [OF subspace_UNIV subspace_UNIV, where
    ?'a = 'M and ?'b = 'N])
    (use dim in auto)
  then have inj h
    by (metis linear_inj_iff_eq_0 norm_eq_zero)
  then have h ' T homeomorphic T
    using <linear h> homeomorphic_sym linear_homeomorphic_image by blast
  then have interior (h ' T) homeomorphic interior S
    using homeomorphic_interiors_same_dimension
    by (metis ST homeomorphic_sym homeomorphic_trans)
  moreover
  have interior (range h) = {}
    by (simp add: <inj h> <linear h> dim dim_image_eq empty_interior_lowdim)
  then have interior (h ' T) = {}
    by (metis image_mono interior_mono subset_empty top_greatest)
  ultimately show ?thesis
    by simp
qed

```

```

lemma empty_interior_lowdim_gen_le:
  fixes S :: 'N::euclidean_space set and T :: 'M::euclidean_space set
  assumes DIM('M)  $\leq$  DIM('N) interior T = {} S homeomorphic T
  shows interior S = {}
  by (metis assms empty_interior_lowdim_gen homeomorphic_empty(1) homeo-
    morphic_interiors_same_dimension less_le)

```

```

lemma homeomorphic_affine_sets_eq:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes affine S affine T
  shows S homeomorphic T  $\longleftrightarrow$  aff_dim S = aff_dim T
proof (cases S = {}  $\vee$  T = {})
case True
  then show ?thesis
    using assms homeomorphic_affine_sets by force
next
case False
  then obtain a b where a  $\in$  S b  $\in$  T
    by blast
  then have subspace ((+) (- a) ' S) subspace ((+) (- b) ' T)
    using affine_diffs_subspace assms by blast+

```

```

    then show ?thesis
      by (metis affine_imp_convex assms homeomorphic_affine_sets homeomor-
        phic_convex_sets)
    qed

```

```

lemma homeomorphic_hyperplanes_eq:
  fixes a :: 'M::euclidean_space and c :: 'N::euclidean_space
  assumes a ≠ 0 c ≠ 0
  shows ({x. a • x = b} homeomorphic {x. c • x = d} ⟷ DIM('M) = DIM('N))
(is ?lhs = ?rhs)
proof -
  have (DIM('M) - Suc 0 = DIM('N) - Suc 0) ⟷ (DIM('M) = DIM('N))
    by auto (metis DIM_positive Suc_pred)
  then show ?thesis
    using assms by (simp add: homeomorphic_affine_sets_eq affine_hyperplane)
qed

```

```

end
theory Homology
  imports Invariance_of_Domain
begin

end

```