

Measure and Probability Theory

March 13, 2025

Contents

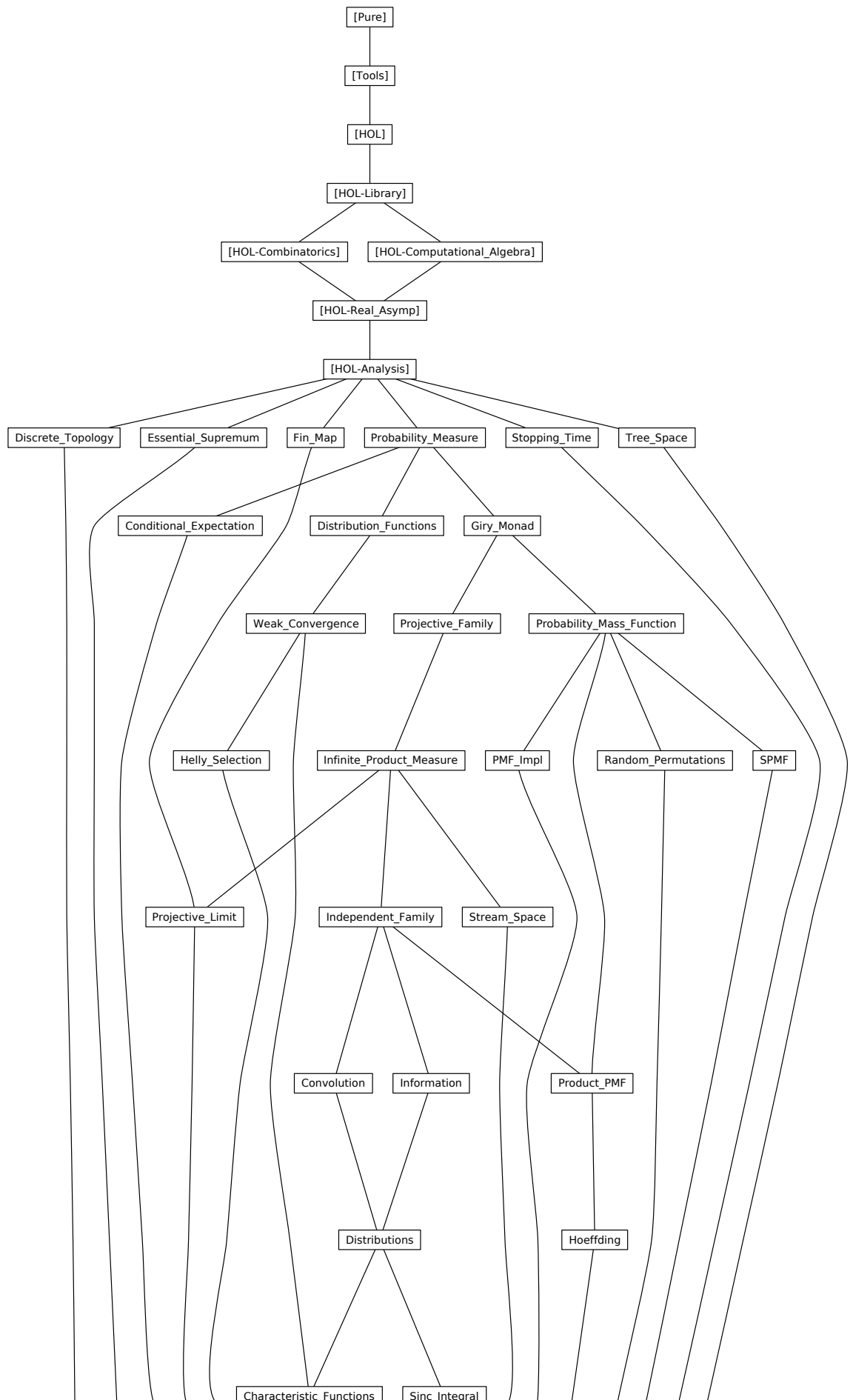
1	Probability measure	3
1.1	Introduce binder for probability	8
1.2	Distributions	15
2	Distribution Functions	32
2.1	Properties of cdf's	32
2.2	Uniqueness	36
3	Weak Convergence of Functions and Distributions	38
4	Weak Convergence of Functions	39
5	Weak Convergence of Distributions	39
6	Skorohod's theorem	39
7	The Giry monad	47
7.1	Sub-probability spaces	48
7.2	Properties of “return”	59
7.3	Join	64
7.4	Giry monad on probability spaces	82
8	Projective Family	88
9	Infinite Product Measure	103
9.1	Sequence space	109
10	Independent families of events, event sets, and random variables	114
11	Convolution Measure	145

12 Information theory	150
12.1 Information theory	150
12.2 Kullback–Leibler divergence	150
12.3 Finite Entropy	156
12.4 Mutual Information	158
12.5 Entropy	166
12.6 Conditional Mutual Information	169
12.7 Conditional Entropy	182
12.8 Equalities	186
13 Properties of Various Distributions	192
13.1 Erlang	194
13.2 Exponential distribution	200
13.3 Uniform distribution	206
13.4 Normal distribution	210
14 Characteristic Functions	224
14.1 Application of the FTC: integrating e^{ix}	225
14.2 The Characteristic Function of a Real Measure.	225
14.3 Independence	226
14.4 Approximations to e^{ix}	227
14.5 Calculation of the Characteristic Function of the Standard Distribution	235
15 Helly’s selection theorem	237
16 Integral of sinc	243
16.1 Various preparatory integrals	244
17 The sinc function, and the sine integral (Si)	246
17.1 The final theorems: boundedness and scalability	251
18 The Levy inversion theorem, and the Levy continuity theo- rem.	253
18.1 The Levy inversion theorem	253
18.2 The Levy continuity theorem	259
19 The Central Limit Theorem	265
20 Probability mass function	270
20.1 PMF as measure	271
20.2 Monad Interpretation	277
20.3 PMFs as function	286
20.4 Conditional Probabilities	295
20.5 Relator	297

20.6	Distributions	308
20.6.1	Bernoulli Distribution	308
20.6.2	Geometric Distribution	309
20.6.3	Uniform Multiset Distribution	311
20.6.4	Uniform Distribution	312
20.6.5	Poisson Distribution	315
20.6.6	Binomial Distribution	315
20.7	Negative Binomial distribution	318
20.8	PMFs from association lists	323
21	Code generation for PMFs	327
21.1	General code generation setup	327
21.2	Code abbreviations for integrals and probabilities	337
22	Finite Maps	339
22.1	Domain and Application	340
22.2	Constructor of Finite Maps	340
22.3	Product set of Finite Maps	341
22.3.1	Basic Properties of Pi'	341
22.4	Topological Space of Finite Maps	342
22.5	Metric Space of Finite Maps	345
22.6	Complete Space of Finite Maps	349
22.7	Second Countable Space of Finite Maps	351
22.8	Polish Space of Finite Maps	353
22.9	Product Measurable Space of Finite Maps	353
22.10	Isomorphism between Functions and Finite Maps	365
23	Projective Limit	369
23.1	Sequences of Finite Maps in Compact Sets	369
23.2	Daniell-Kolmogorov Theorem	371
24	Random Permutations	379
25	Discrete subprobability distribution	384
25.1	Auxiliary material	384
25.1.1	More about extended reals	384
25.1.2	More about <i>'a option</i>	385
25.1.3	A relator for sets that treats sets like predicates	387
25.1.4	Monotonicity rules	388
25.1.5	Bijections	388
25.2	Subprobability mass function	389
25.3	Support	392
25.4	Functorial structure	394
25.5	Monad operations	395

25.5.1	Return	395
25.5.2	Bind	396
25.6	Relator	399
25.7	From ' <i>a pmf</i> ' to ' <i>a spmf</i> '	402
25.8	Weight of a subprobability	403
25.9	From density to spmfs	405
25.10	Ordering on spmfs	407
25.11	CCPO structure for the flat cppo <i>ord-option</i> (=)	412
25.11.1	Admissibility of <i>rel-spmf</i>	423
25.12	Restrictions on spmfs	426
25.13	Subprobability distributions of sets	428
25.14	Losslessness	432
25.15	Scaling	434
25.16	Conditional spmfs	440
25.17	Product spmf	441
25.18	Assertions	445
25.19	Try	445
25.20	Miscellaneous	448
26	Indexed products of PMFs	449
26.1	Preliminaries	449
26.2	Definition	449
26.3	Dependent product sets with a default	452
26.4	Common PMF operations on products	455
26.5	Merging and splitting PMF products	459
26.6	Additional properties	463
26.7	Applications	467
27	Hoeffding's Lemma and Hoeffding's Inequality	468
27.1	Hoeffding's Lemma	468
27.2	Hoeffding's Inequality	473
27.3	Hoeffding's inequality for i.i.d. bounded random variables	476
27.4	Hoeffding's Inequality for the Binomial distribution	479
27.5	Tail bounds for the negative binomial distribution	481
28	Conditional Expectation	512
28.1	Restricting a measure to a sub-sigma-algebra	512
28.2	Nonnegative conditional expectation	516
28.3	Real conditional expectation	522
29	The essential supremum	545
30	Stopping times	548
30.1	Stopping Time	548

31 Filtration	549
31.1 σ -algebra of a Stopping Time	549



1 Probability measure

```
theory Probability-Measure
  imports HOL-Analysis.Analysis
begin
```

```
locale prob-space = finite-measure +
  assumes emeasure-space-1: emeasure M (space M) = 1
```

```
lemma prob-spaceI[Pure.intro!]:
  assumes *: emeasure M (space M) = 1
  shows prob-space M
  by (simp add: asms finite-measureI prob-space-axioms.intro prob-space-def)
```

```
lemma prob-space-imp-sigma-finite: prob-space M  $\implies$  sigma-finite-measure M
  unfolding prob-space-def finite-measure-def by simp
```

```
abbreviation (in prob-space) events  $\equiv$  sets M
abbreviation (in prob-space) prob  $\equiv$  measure M
abbreviation (in prob-space) random-variable  $M' X \equiv X \in$  measurable  $M M'$ 
abbreviation (in prob-space) expectation  $\equiv$  integralL M
abbreviation (in prob-space) variance  $X \equiv$  integralL M ( $\lambda x. (X x - \text{expectation } X)^2$ )
```

```
lemma (in prob-space) finite-measure [simp]: finite-measure M
  by unfold-locales
```

```
lemma (in prob-space) prob-space-distr:
  assumes f:  $f \in$  measurable  $M M'$  shows prob-space (distr M  $M' f$ )
proof (rule prob-spaceI)
  have f - 'space  $M' \cap$  space  $M =$  space  $M$  using f by (auto dest: measurable-space)
  with f show emeasure (distr M  $M' f$ ) (space (distr M  $M' f$ )) = 1
  by (auto simp: emeasure-distr emeasure-space-1)
qed
```

```
lemma prob-space-distrD:
  assumes f:  $f \in$  measurable  $M N$  and M: prob-space (distr M  $N f$ ) shows
  prob-space M
proof
  interpret M: prob-space distr M  $N f$  by fact
  have f - 'space  $N \cap$  space  $M =$  space  $M$ 
  using f[THEN measurable-space] by auto
  then show emeasure M (space M) = 1
  using M.emeasure-space-1 by (simp add: emeasure-distr[OF f])
qed
```

```
lemma (in prob-space) prob-space: prob (space M) = 1
  by (simp add: emeasure-space-1 measure-eq-emeasure-eq-enreal)
```

lemma (in prob-space) prob-le-1[simp, intro]: $\text{prob } A \leq 1$
 using bounded-measure[of A] **by** (simp add: prob-space)

lemma (in prob-space) not-empty: $\text{space } M \neq \{\}$
 using prob-space **by** auto

lemma (in prob-space) emeasure-eq-1-AE:
 $S \in \text{sets } M \implies \text{AE } x \text{ in } M. x \in S \implies \text{emeasure } M S = 1$
by (subst emeasure-eq-AE[where B=space M]) (auto simp: emeasure-space-1)

lemma (in prob-space) emeasure-le-1: $\text{emeasure } M S \leq 1$
 unfolding ennreal-1[symmetric] emeasure-eq-measure **by** (subst ennreal-le-iff)
 auto

lemma (in prob-space) emeasure-ge-1-iff: $\text{emeasure } M A \geq 1 \longleftrightarrow \text{emeasure } M A = 1$
by (rule iffI, intro antisym emeasure-le-1) simp-all

lemma (in prob-space) AE-iff-emeasure-eq-1:
 assumes [measurable]: Measurable.pred M P
 shows $(\text{AE } x \text{ in } M. P x) \longleftrightarrow \text{emeasure } M \{x \in \text{space } M. P x\} = 1$
proof –
 have *: $\{x \in \text{space } M. \neg P x\} = \text{space } M - \{x \in \text{space } M. P x\}$
by auto
 show ?thesis
by (auto simp add: ennreal-minus-eq-0 * emeasure-compl emeasure-space-1
 AE-iff-measurable[OF - refl]
 intro: antisym emeasure-le-1)
qed

lemma (in prob-space) measure-le-1: $\text{measure } M X \leq 1$
 using emeasure-space[of M X] **by** (simp add: emeasure-space-1)

lemma (in prob-space) measure-ge-1-iff: $\text{measure } M A \geq 1 \longleftrightarrow \text{measure } M A = 1$
by (auto intro!: antisym)

lemma (in prob-space) AE-I-eq-1:
 assumes $\text{emeasure } M \{x \in \text{space } M. P x\} = 1$ $\{x \in \text{space } M. P x\} \in \text{sets } M$
 shows $\text{AE } x \text{ in } M. P x$
proof (rule AE-I)
 show $\text{emeasure } M (\text{space } M - \{x \in \text{space } M. P x\}) = 0$
using assms emeasure-space-1 **by** (simp add: emeasure-compl)
qed (insert assms, auto)

lemma prob-space-restrict-space:
 $S \in \text{sets } M \implies \text{emeasure } M S = 1 \implies \text{prob-space } (\text{restrict-space } M S)$
by (intro prob-spaceI)

(simp add: emeasure-restrict-space space-restrict-space)

lemma (in prob-space) prob-compl:

assumes $A: A \in \text{events}$

shows $\text{prob} (\text{space } M - A) = 1 - \text{prob } A$

using finite-measure-compl[OF A] by (simp add: prob-space)

lemma (in prob-space) AE-in-set-eq-1:

assumes $A[\text{measurable}]: A \in \text{events}$ shows $(AE\ x\ \text{in } M. x \in A) \longleftrightarrow \text{prob } A = 1$

proof –

have *: $\{x \in \text{space } M. x \in A\} = A$

using A[THEN sets.sets-into-space] by auto

show ?thesis

by (subst AE-iff-emeasure-eq-1) (auto simp: emeasure-eq-measure *)

qed

lemma (in prob-space) AE-False: $(AE\ x\ \text{in } M. \text{False}) \longleftrightarrow \text{False}$

proof

assume $AE\ x\ \text{in } M. \text{False}$

then have $AE\ x\ \text{in } M. x \in \{\}$ by simp

then show False

by (subst (asm) AE-in-set-eq-1) auto

qed simp

lemma (in prob-space) AE-prob-1:

assumes $\text{prob } A = 1$ shows $AE\ x\ \text{in } M. x \in A$

proof –

from $\langle \text{prob } A = 1 \rangle$ have $A \in \text{events}$

by (metis measure-notin-sets zero-neq-one)

with AE-in-set-eq-1 assms show ?thesis by simp

qed

lemma (in prob-space) AE-const[simp]: $(AE\ x\ \text{in } M. P) \longleftrightarrow P$

by (cases P) (auto simp: AE-False)

lemma (in prob-space) ae-filter-bot: $\text{ae-filter } M \neq \text{bot}$

by (simp add: trivial-limit-def)

lemma (in prob-space) AE-contr:

assumes $ae: AE\ \omega\ \text{in } M. P \ \omega\ AE\ \omega\ \text{in } M. \neg P \ \omega$

shows False

proof –

from ae have $AE\ \omega\ \text{in } M. \text{False}$ by eventually-elim auto

then show False by auto

qed

lemma (in prob-space) integral-ge-const:

fixes $c :: \text{real}$

shows $\text{integrable } M f \implies (AE\ x\ \text{in } M. c \leq f\ x) \implies c \leq (\int x. f\ x\ \partial M)$
using *integral-mono-AE*[of $M\ \lambda x. c\ f$] *prob-space* **by** *simp*

lemma (**in** *prob-space*) *integral-le-const*:
fixes $c :: \text{real}$
shows $\text{integrable } M f \implies (AE\ x\ \text{in } M. f\ x \leq c) \implies (\int x. f\ x\ \partial M) \leq c$
using *integral-mono-AE*[of $M\ f\ \lambda x. c$] *prob-space* **by** *simp*

lemma (**in** *prob-space*) *nn-integral-ge-const*:
 $(AE\ x\ \text{in } M. c \leq f\ x) \implies c \leq (\int^+ x. f\ x\ \partial M)$
using *nn-integral-mono-AE*[of $\lambda x. c\ f\ M$] *emeasure-space-1*
by (*simp split: if-split-asm*)

lemma (**in** *prob-space*) *expectation-less*:
fixes $X :: - \Rightarrow \text{real}$
assumes [*simp*]: *integrable* $M\ X$
assumes *gt*: $AE\ x\ \text{in } M. X\ x < b$
shows *expectation* $X < b$
proof –
have *expectation* $X < \text{expectation } (\lambda x. b)$
using *gt emeasure-space-1*
by (*intro integral-less-AE-space*) *auto*
then show ?thesis **using** *prob-space* **by** *simp*
qed

lemma (**in** *prob-space*) *expectation-greater*:
fixes $X :: - \Rightarrow \text{real}$
assumes [*simp*]: *integrable* $M\ X$
assumes *gt*: $AE\ x\ \text{in } M. a < X\ x$
shows $a < \text{expectation } X$
proof –
have *expectation* $(\lambda x. a) < \text{expectation } X$
using *gt emeasure-space-1*
by (*intro integral-less-AE-space*) *auto*
then show ?thesis **using** *prob-space* **by** *simp*
qed

lemma (**in** *prob-space*) *jensens-inequality*:
fixes $q :: \text{real} \Rightarrow \text{real}$
assumes X : *integrable* $M\ X$ $AE\ x\ \text{in } M. X\ x \in I$
assumes I : $I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = UNIV$
assumes q : *integrable* $M\ (\lambda x. q\ (X\ x))$ *convex-on* $I\ q$
shows $q\ (\text{expectation } X) \leq \text{expectation } (\lambda x. q\ (X\ x))$
proof –
let ? $F = \lambda x. \text{Inf } ((\lambda t. (q\ x - q\ t) / (x - t))\ '(\{x<..\} \cap I))$
from $X(2)$ *AE-False* **have** $I \neq \{\}$ **by** *auto*

from I **have** *open* I **by** *auto*

```

note  $I$ 
moreover
{ assume  $I \subseteq \{a < ..\}$ 
  with  $X$  have  $a < \text{expectation } X$ 
    by (intro expectation-greater) auto }
moreover
{ assume  $I \subseteq \{.. < b\}$ 
  with  $X$  have  $\text{expectation } X < b$ 
    by (intro expectation-less) auto }
ultimately have  $\text{expectation } X \in I$ 
  by (elim disjE) (auto simp: subset-eq)
moreover
{ fix  $y$  assume  $y: y \in I$ 
  with  $q(2) \langle \text{open } I \rangle$  have  $\text{Sup } ((\lambda x. q\ x + ?F\ x * (y - x)) \text{ ` } I) = q\ y$ 
    by (auto intro!: cSup-eq-maximum convex-le-Inf-differential image-eqI [OF -
y] simp: interior-open) }
  ultimately have  $q(\text{expectation } X) = \text{Sup } ((\lambda x. q\ x + ?F\ x * (\text{expectation } X -$ 
x)) \text{ ` } I)
    by simp
  also have  $\dots \leq \text{expectation } (\lambda w. q\ (X\ w))$ 
proof (rule cSup-least)
  show  $(\lambda x. q\ x + ?F\ x * (\text{expectation } X - x)) \text{ ` } I \neq \{\}$ 
    using  $\langle I \neq \{\} \rangle$  by auto
next
  fix  $k$  assume  $k \in (\lambda x. q\ x + ?F\ x * (\text{expectation } X - x)) \text{ ` } I$ 
  then obtain  $x$ 
    where  $x: k = q\ x + (\text{INF } t \in \{x < ..\} \cap I. (q\ x - q\ t) / (x - t)) * (\text{expectation } X - x)$ 
 $x \in I ..$ 
    have  $q\ x + ?F\ x * (\text{expectation } X - x) = \text{expectation } (\lambda w. q\ x + ?F\ x * (X\ w - x))$ 
      using prob-space by (simp add: X)
    also have  $\dots \leq \text{expectation } (\lambda w. q\ (X\ w))$ 
      using  $\langle x \in I \rangle \langle \text{open } I \rangle X(2)$ 
    apply (intro integral-mono-AE Bochner-Integration.integrable-add Bochner-Integration.integrable-mult-right Bochner-Integration.integrable-diff
      integrable-const X q)
    apply (elim eventually-mono)
    apply (intro convex-le-Inf-differential)
    apply (auto simp: interior-open q)
    done
  finally show  $k \leq \text{expectation } (\lambda w. q\ (X\ w))$  using  $x$  by auto
qed
finally show  $q(\text{expectation } X) \leq \text{expectation } (\lambda x. q\ (X\ x))$  .
qed

```

lemma (*in prob-space*) *finite-borel-measurable-integrable:*

```

assumes  $f \in \text{borel-measurable } M$ 
and finite (f ` (space M))
shows integrable M f

```

```

proof –
  have simple-function  $M f$  using assms by (simp add: simple-function-borel-measurable)
  moreover have emeasure  $M \{y \in \text{space } M. f y \neq 0\} \neq \infty$  by simp
  ultimately have Bochner-Integration.simple-bochner-integrable  $M f$ 
    using Bochner-Integration.simple-bochner-integrable.simps by blast
  hence has-bochner-integral  $M f$  (Bochner-Integration.simple-bochner-integral  $M f$ )
  using has-bochner-integral-simple-bochner-integrable by auto
  thus ?thesis using integrable.simps by auto
qed

```

1.1 Introduce binder for probability

syntax

```
-prob :: ptrn  $\Rightarrow$  logic  $\Rightarrow$  logic  $\Rightarrow$  logic ( $\langle ('P'((/- \text{in } - / -)') \rangle)$ )
```

syntax-consts

```
-prob == measure
```

translations

```
 $\mathcal{P}(x \text{ in } M. P) \Rightarrow \text{CONST } \text{measure } M \{x \in \text{CONST } \text{space } M. P\}$ 
```

print-translation \langle

```

  let
    fun to-pattern (Const (const-syntax  $\langle \text{Pair} \rangle$ , -)  $\$ l \$ r$ ) =
      Syntax.const const-syntax  $\langle \text{Pair} \rangle :: \text{to-pattern } l @ \text{to-pattern } r$ 
    | to-pattern (t as (Const (syntax-const  $\langle \text{-bound} \rangle$ , -))  $\$ -$ ) = [t]

    fun mk-pattern ((t, n) :: xs) = mk-patterns n xs |>> curry list-comb t
    and mk-patterns 0 xs = ([], xs)
    | mk-patterns n xs =
      let
        val (t, xs') = mk-pattern xs
        val (ts, xs'') = mk-patterns (n - 1) xs'
      in
        (t :: ts, xs'')
    end

```

fun *unnest-tuples*

```

  (Const (syntax-const  $\langle \text{-pattern} \rangle$ , -)  $\$$ 
    t1  $\$$ 
    (t as (Const (syntax-const  $\langle \text{-pattern} \rangle$ , -)  $\$ - \$ -$ )))
  = let
    val (-  $\$ t2 \$ t3$ ) = unnest-tuples t
  in
    Syntax.const syntax-const  $\langle \text{-pattern} \rangle \$$ 
      unnest-tuples t1  $\$$ 
      (Syntax.const syntax-const  $\langle \text{-patterns} \rangle \$ t2 \$ t3$ )
  end
  | unnest-tuples pat = pat

```

```

fun tr' ctxt [sig-alg, Const (const-syntax ⟨Collect⟩, -) $ t] =
  let
    val bound-dummyT = Const (syntax-const ⟨-bound⟩, dummyT)

    fun go pattern elem
      (Const (const-syntax ⟨conj⟩, -) $
       (Const (const-syntax ⟨Set.member⟩, -) $ elem' $ (Const (const-syntax ⟨space⟩,
       -) $ sig-alg'))) $
      u)
    = let
      val - = if sig-alg aconv sig-alg' andalso to-pattern elem' = rev elem then
    () else raise Match;
      val (pat, rest) = mk-pattern (rev pattern);
      val - = case rest of [] => () | - => raise Match
    in
      Syntax.const syntax-const ⟨-prob⟩ $ unnest-tuples pat $ sig-alg $ u
    end
  | go pattern elem (Abs abs) =
    let
      val (x as (- $ tx), t) = Syntax-Trans.atomic-abs-tr' ctxt abs
    in
      go ((x, 0) :: pattern) (bound-dummyT $ tx :: elem) t
    end
  | go pattern elem (Const (const-syntax ⟨case-prod⟩, -) $ t) =
    go
      ((Syntax.const syntax-const ⟨-pattern⟩, 2) :: pattern)
      (Syntax.const const-syntax ⟨Pair⟩ :: elem)
      t
  in
    go [] [] t
  end
in
  [(const-syntax ⟨Sigma-Algebra.measure⟩, tr')]
end

```

definition

$$\text{cond-prob } M \ P \ Q = \mathcal{P}(\omega \text{ in } M. P \ \omega \wedge Q \ \omega) / \mathcal{P}(\omega \text{ in } M. Q \ \omega)$$
syntax

$$\text{-conditional-prob} :: \text{pttrn} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow \text{logic} \Rightarrow (\langle 'P'(- \text{ in } -. - \mid / -') \rangle)$$
syntax-consts

$$\text{-conditional-prob} == \text{cond-prob}$$
translations

$$\mathcal{P}(x \text{ in } M. P \mid Q) \Rightarrow \text{CONST cond-prob } M \ (\lambda x. P) \ (\lambda x. Q)$$
lemma (in prob-space) AE-E-prob:

$$\text{assumes } ae: AE \ x \text{ in } M. P \ x$$

$$\text{obtains } S \text{ where } S \subseteq \{x \in \text{space } M. P \ x\} \ S \in \text{events prob } S = 1$$

proof –

from $ae[THEN\ AE-E]$ **obtain** N
where $\{x \in space\ M. \neg P\ x\} \subseteq N$ $emeasure\ M\ N = 0$ $N \in events$ **by** $auto$
then show $thesis$
by $(intro\ that[of\ space\ M - N])$
 $(auto\ simp: prob-compl\ prob-space\ emeasure-eq-measure\ measure-nonneg)$
qed

lemma $(in\ prob-space)$ $prob-neg: \{x \in space\ M. P\ x\} \in events \implies \mathcal{P}(x\ in\ M. \neg P\ x) = 1 - \mathcal{P}(x\ in\ M. P\ x)$
by $(auto\ intro!: arg-cong[where\ f=prob]\ simp\ add: prob-compl[symmetric])$

lemma $(in\ prob-space)$ $prob-eq-AE:$
 $(AE\ x\ in\ M. P\ x \longleftrightarrow Q\ x) \implies \{x \in space\ M. P\ x\} \in events \implies \{x \in space\ M. Q\ x\} \in events \implies \mathcal{P}(x\ in\ M. P\ x) = \mathcal{P}(x\ in\ M. Q\ x)$
by $(rule\ finite-measure-eq-AE)\ auto$

lemma $(in\ prob-space)$ $prob-eq-0-AE:$
assumes $not: AE\ x\ in\ M. \neg P\ x$ **shows** $\mathcal{P}(x\ in\ M. P\ x) = 0$
proof $cases$
assume $\{x \in space\ M. P\ x\} \in events$
with $not\ have\ \mathcal{P}(x\ in\ M. P\ x) = \mathcal{P}(x\ in\ M. False)$
by $(intro\ prob-eq-AE)\ auto$
then show $?thesis$ **by** $simp$
qed $(simp\ add: measure-notin-sets)$

lemma $(in\ prob-space)$ $prob-Collect-eq-0:$
 $\{x \in space\ M. P\ x\} \in sets\ M \implies \mathcal{P}(x\ in\ M. P\ x) = 0 \longleftrightarrow (AE\ x\ in\ M. \neg P\ x)$
using $AE-iff-measurable[OF - refl, of\ M\ \lambda x. \neg P\ x]$ **by** $(simp\ add: emeasure-eq-measure\ measure-nonneg)$

lemma $(in\ prob-space)$ $prob-Collect-eq-1:$
 $\{x \in space\ M. P\ x\} \in sets\ M \implies \mathcal{P}(x\ in\ M. P\ x) = 1 \longleftrightarrow (AE\ x\ in\ M. P\ x)$
using $AE-in-set-eq-1[of\ \{x \in space\ M. P\ x\}]$ **by** $simp$

lemma $(in\ prob-space)$ $prob-eq-0:$
 $A \in sets\ M \implies prob\ A = 0 \longleftrightarrow (AE\ x\ in\ M. x \notin A)$
using $AE-iff-measurable[OF - refl, of\ M\ \lambda x. x \notin A]$
by $(auto\ simp\ add: emeasure-eq-measure\ Int-def[symmetric]\ measure-nonneg)$

lemma $(in\ prob-space)$ $prob-eq-1:$
 $A \in sets\ M \implies prob\ A = 1 \longleftrightarrow (AE\ x\ in\ M. x \in A)$
using $AE-in-set-eq-1[of\ A]$ **by** $simp$

lemma $(in\ prob-space)$ $prob-sums:$
assumes $P: \bigwedge n. \{x \in space\ M. P\ n\ x\} \in events$
assumes $Q: \{x \in space\ M. Q\ x\} \in events$
assumes $ae: AE\ x\ in\ M. (\forall n. P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists ! n. P\ n\ x))$
shows $(\lambda n. \mathcal{P}(x\ in\ M. P\ n\ x))\ sums\ \mathcal{P}(x\ in\ M. Q\ x)$

proof –

from $ae[THEN\ AE-E-prob]$ **obtain** S
where S :
 $S \subseteq \{x \in space\ M. (\forall n. P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists !n. P\ n\ x))\}$
 $S \in events$
 $prob\ S = 1$
by *auto*
then have $disj$: *disjoint-family* $(\lambda n. \{x \in space\ M. P\ n\ x\} \cap S)$
by *(auto simp: disjoint-family-on-def)*
from S **have** $ae-S$:
 $AE\ x\ in\ M. x \in \{x \in space\ M. Q\ x\} \longleftrightarrow x \in (\bigcup n. \{x \in space\ M. P\ n\ x\} \cap S)$
 $\bigwedge n. AE\ x\ in\ M. x \in \{x \in space\ M. P\ n\ x\} \longleftrightarrow x \in \{x \in space\ M. P\ n\ x\} \cap S$
using ae **by** *(auto dest!: AE-prob-1)*
from $ae-S$ **have** *:
 $\mathcal{P}(x\ in\ M. Q\ x) = prob\ (\bigcup n. \{x \in space\ M. P\ n\ x\} \cap S)$
using $P\ Q\ S$ **by** *(intro finite-measure-eq-AE) auto*
from $ae-S$ **have** **:
 $\bigwedge n. \mathcal{P}(x\ in\ M. P\ n\ x) = prob\ (\{x \in space\ M. P\ n\ x\} \cap S)$
using $P\ Q\ S$ **by** *(intro finite-measure-eq-AE) auto*
show *?thesis*
unfolding * ** **using** $S\ P\ disj$
by *(intro finite-measure-UNION) auto*
qed

lemma *(in prob-space) prob-sum*:

assumes $[simp, intro]$: *finite* I
assumes P : $\bigwedge n. n \in I \implies \{x \in space\ M. P\ n\ x\} \in events$
assumes Q : $\{x \in space\ M. Q\ x\} \in events$
assumes ae : $AE\ x\ in\ M. (\forall n \in I. P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists !n \in I. P\ n\ x))$
shows $\mathcal{P}(x\ in\ M. Q\ x) = (\sum n \in I. \mathcal{P}(x\ in\ M. P\ n\ x))$

proof –

from $ae[THEN\ AE-E-prob]$ **obtain** S
where S :
 $S \subseteq \{x \in space\ M. (\forall n \in I. P\ n\ x \longrightarrow Q\ x) \wedge (Q\ x \longrightarrow (\exists !n. n \in I \wedge P\ n\ x))\}$
 $S \in events$
 $prob\ S = 1$
by *auto*
then have $disj$: *disjoint-family-on* $(\lambda n. \{x \in space\ M. P\ n\ x\} \cap S)\ I$
by *(auto simp: disjoint-family-on-def)*
from S **have** $ae-S$:
 $AE\ x\ in\ M. x \in \{x \in space\ M. Q\ x\} \longleftrightarrow x \in (\bigcup n \in I. \{x \in space\ M. P\ n\ x\} \cap S)$
 $\bigwedge n. n \in I \implies AE\ x\ in\ M. x \in \{x \in space\ M. P\ n\ x\} \longleftrightarrow x \in \{x \in space\ M. P\ n\ x\} \cap S$
using ae **by** *(auto dest!: AE-prob-1)*
from $ae-S$ **have** *:
 $\mathcal{P}(x\ in\ M. Q\ x) = prob\ (\bigcup n \in I. \{x \in space\ M. P\ n\ x\} \cap S)$
using $P\ Q\ S$ **by** *(intro finite-measure-eq-AE) (auto intro!: sets.Int)*
from $ae-S$ **have** **:
 $\mathcal{P}(x\ in\ M. P\ n\ x) = prob\ (\{x \in space\ M. P\ n\ x\} \cap S)$

$\bigwedge n. n \in I \implies \mathcal{P}(x \text{ in } M. P \ n \ x) = \text{prob} \ (\{x \in \text{space } M. P \ n \ x\} \cap S)$
using $P \ Q \ S$ **by** (intro finite-measure-eq-AE) *auto*
show ?thesis
using $S \ P \ \text{disj}$
by (auto simp add: * ** simp del: UN-simps intro!: finite-measure-finite-Union)
qed

lemma (in prob-space) prob-EX-countable:

assumes sets: $\bigwedge i. i \in I \implies \{x \in \text{space } M. P \ i \ x\} \in \text{sets } M$ **and** I : countable I
assumes disj: $AE \ x \text{ in } M. \forall i \in I. \forall j \in I. P \ i \ x \longrightarrow P \ j \ x \longrightarrow i = j$
shows $\mathcal{P}(x \text{ in } M. \exists i \in I. P \ i \ x) = (\int^{+i}. \mathcal{P}(x \text{ in } M. P \ i \ x) \ \partial \text{count-space } I)$
proof –
let ?N = $\lambda x. \exists ! i \in I. P \ i \ x$
have ennreal $(\mathcal{P}(x \text{ in } M. \exists i \in I. P \ i \ x)) = \mathcal{P}(x \text{ in } M. (\exists i \in I. P \ i \ x \wedge ?N \ x))$
unfolding ennreal-inj[OF measure-nonneg measure-nonneg]
proof (rule prob-eq-AE)
show $AE \ x \text{ in } M. (\exists i \in I. P \ i \ x) = (\exists i \in I. P \ i \ x \wedge ?N \ x)$
using disj **by** eventually-elim blast
qed (auto intro!: sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1' $I \text{ sets}$)
also have $\mathcal{P}(x \text{ in } M. (\exists i \in I. P \ i \ x \wedge ?N \ x)) = \text{emeasure } M \ (\bigcup i \in I. \{x \in \text{space } M. P \ i \ x \wedge ?N \ x\})$
unfolding emeasure-eq-measure **by** (auto intro!: arg-cong[where $f = \text{prob}$] simp: measure-nonneg)
also have $\dots = (\int^{+i}. \text{emeasure } M \ \{x \in \text{space } M. P \ i \ x \wedge ?N \ x\} \ \partial \text{count-space } I)$
by (rule emeasure-UN-countable)
 $(\text{auto intro!: sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1' } I \text{ sets})$
 $\text{simp: disjoint-family-on-def}$
also have $\dots = (\int^{+i}. \mathcal{P}(x \text{ in } M. P \ i \ x) \ \partial \text{count-space } I)$
unfolding emeasure-eq-measure **using** disj
by (intro nn-integral-cong ennreal-inj[THEN iffD2] prob-eq-AE)
 $(\text{auto intro!: sets.sets-Collect-countable-Ex' sets.sets-Collect-conj sets.sets-Collect-countable-Ex1' } I \text{ sets measure-nonneg})$
finally show ?thesis .
qed

lemma (in prob-space) cond-prob-eq-AE:

assumes P : $AE \ x \text{ in } M. Q \ x \longrightarrow P \ x \longleftrightarrow P' \ x \ \{x \in \text{space } M. P \ x\} \in \text{events}$
 $\{x \in \text{space } M. P' \ x\} \in \text{events}$
assumes Q : $AE \ x \text{ in } M. Q \ x \longleftrightarrow Q' \ x \ \{x \in \text{space } M. Q \ x\} \in \text{events}$
 $\{x \in \text{space } M. Q' \ x\} \in \text{events}$
shows $\text{cond-prob } M \ P \ Q = \text{cond-prob } M \ P' \ Q'$
using $P \ Q$
by (auto simp: cond-prob-def intro!: arg-cong2[where $f = (/)$] prob-eq-AE sets.sets-Collect-conj)

lemma (in prob-space) joint-distribution-Times-le-fst:

random-variable $MX \ X \implies \text{random-variable } MY \ Y \implies A \in \text{sets } MX \implies B \in$

sets MY

$\implies \text{emeasure } (\text{distr } M \ (MX \otimes_M MY) \ (\lambda x. (X \ x, Y \ x))) \ (A \times B) \leq \text{emeasure } (\text{distr } M \ MX \ X) \ A$
by (auto simp: emeasure-distr measurable-pair-iff comp-def intro!: emeasure-mono measurable-sets)

lemma (in prob-space) joint-distribution-Times-le-snd:

random-variable MX X \implies random-variable MY Y $\implies A \in \text{sets } MX \implies B \in \text{sets } MY$
 $\implies \text{emeasure } (\text{distr } M \ (MX \otimes_M MY) \ (\lambda x. (X \ x, Y \ x))) \ (A \times B) \leq \text{emeasure } (\text{distr } M \ MY \ Y) \ B$
by (auto simp: emeasure-distr measurable-pair-iff comp-def intro!: emeasure-mono measurable-sets)

lemma (in prob-space) variance-eq:

fixes X :: 'a \Rightarrow real
 assumes [simp]: integrable M X
 assumes [simp]: integrable M $(\lambda x. (X \ x)^2)$
 shows variance X = expectation $(\lambda x. (X \ x)^2) - (\text{expectation } X)^2$
by (simp add: field-simps prob-space power2-diff power2-eq-square[symmetric])

lemma (in prob-space) variance-positive: $0 \leq \text{variance } (X::'a \Rightarrow \text{real})$

by (intro integral-nonneg-AE) (auto intro!: integral-nonneg-AE)

lemma (in prob-space) variance-mean-zero:

expectation X = 0 \implies variance X = expectation $(\lambda x. (X \ x)^2)$
by simp

theorem (in prob-space) Chebyshev-inequality:

assumes [measurable]: random-variable borel f
 assumes integrable M $(\lambda x. f \ x^2)$
 defines $\mu \equiv \text{expectation } f$
 assumes $a > 0$
 shows prob $\{x \in \text{space } M. |f \ x - \mu| \geq a\} \leq \text{variance } f / a^2$
 unfolding μ -def
proof (rule second-moment-method)
 have integrable: integrable M f
 using assms **by** (blast dest: square-integrable-imp-integrable)
 show integrable M $(\lambda x. (f \ x - \text{expectation } f)^2)$
 using assms integrable **unfolding** power2-eq-square ring-distribs
by (intro Bochner-Integration.integrable-diff) auto
qed (use assms in auto)

locale pair-prob-space = pair-sigma-finite M1 M2 + M1: prob-space M1 + M2:
 prob-space M2 **for** M1 M2

sublocale pair-prob-space $\subseteq P?$: prob-space M1 \otimes_M M2

proof

show $\text{emeasure } (M1 \otimes_M M2) \ (\text{space } (M1 \otimes_M M2)) = 1$

by (simp add: M2.emeasure-pair-measure-Times M1.emeasure-space-1 M2.emeasure-space-1 space-pair-measure)

qed

locale product-prob-space = product-sigma-finite M for M :: 'i \Rightarrow 'a measure +
 fixes I :: 'i set
 assumes prob-space: $\bigwedge i. \text{prob-space } (M\ i)$

sublocale product-prob-space $\subseteq M?$: prob-space M i for i
 by (rule prob-space)

locale finite-product-prob-space = finite-product-sigma-finite M I + product-prob-space
 M I for M I

sublocale finite-product-prob-space $\subseteq \text{prob-space } \Pi_M\ i \in I. M\ i$
 proof

show emeasure $(\Pi_M\ i \in I. M\ i)$ (space $(\Pi_M\ i \in I. M\ i)$) = 1

by (simp add: measure-times M.emeasure-space-1 prod.neutral-const space-PiM)

qed

lemma (in finite-product-prob-space) prob-times:

assumes X: $\bigwedge i. i \in I \implies X\ i \in \text{sets } (M\ i)$

shows prob $(\Pi_E\ i \in I. X\ i) = (\prod i \in I. M.\text{prob } i\ (X\ i))$

proof –

have ennreal (measure $(\Pi_M\ i \in I. M\ i)$ $(\Pi_E\ i \in I. X\ i)$) = emeasure $(\Pi_M\ i \in I. M\ i)$ $(\Pi_E\ i \in I. X\ i)$

using X by (simp add: emeasure-eq-measure)

also have ... = $(\prod i \in I. \text{emeasure } (M\ i)\ (X\ i))$

using measure-times X by simp

also have ... = ennreal $(\prod i \in I. \text{measure } (M\ i)\ (X\ i))$

using X by (simp add: M.emeasure-eq-measure prod-ennreal measure-nonneg)

finally show ?thesis by (simp add: measure-nonneg prod-nonneg)

qed

lemma product-prob-spaceI:

assumes $\bigwedge i. \text{prob-space } (M\ i)$

shows product-prob-space M

unfolding product-prob-space-def product-prob-space-axioms-def product-sigma-finite-def

proof safe

fix i

interpret prob-space M i

by (rule assms)

show sigma-finite-measure (M i) prob-space (M i)

by unfold-locales

qed

1.2 Distributions

definition *distributed* :: 'a measure \Rightarrow 'b measure \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow ennreal) \Rightarrow bool

where

distributed $M\ N\ X\ f \longleftrightarrow$

distr $M\ N\ X = \text{density } N\ f \wedge f \in \text{borel-measurable } N \wedge X \in \text{measurable } M\ N$

lemma

assumes *distributed* $M\ N\ X\ f$

shows *distributed-distr-eq-density*: *distr* $M\ N\ X = \text{density } N\ f$

and *distributed-measurable*: $X \in \text{measurable } M\ N$

and *distributed-borel-measurable*: $f \in \text{borel-measurable } N$

using *assms* **by** (*simp-all* *add*: *distributed-def*)

lemma

assumes D : *distributed* $M\ N\ X\ f$

shows *distributed-measurable'*[*measurable-dest*]:

$g \in \text{measurable } L\ M \implies (\lambda x. X\ (g\ x)) \in \text{measurable } L\ N$

and *distributed-borel-measurable'*[*measurable-dest*]:

$h \in \text{measurable } L\ N \implies (\lambda x. f\ (h\ x)) \in \text{borel-measurable } L$

using *distributed-measurable*[*OF D*] *distributed-borel-measurable*[*OF D*]

by *simp-all*

lemma *distributed-real-measurable*:

$(\bigwedge x. x \in \text{space } N \implies 0 \leq f\ x) \implies \text{distributed } M\ N\ X\ (\lambda x. \text{ennreal } (f\ x)) \implies f \in \text{borel-measurable } N$

by (*simp-all* *add*: *distributed-def*)

lemma *distributed-real-measurable'*:

$(\bigwedge x. x \in \text{space } N \implies 0 \leq f\ x) \implies \text{distributed } M\ N\ X\ (\lambda x. \text{ennreal } (f\ x)) \implies$

$h \in \text{measurable } L\ N \implies (\lambda x. f\ (h\ x)) \in \text{borel-measurable } L$

using *distributed-real-measurable*[*measurable*] **by** *simp*

lemma *joint-distributed-measurable1*:

distributed $M\ (S \otimes_M T)\ (\lambda x. (X\ x, Y\ x))\ f \implies h1 \in \text{measurable } N\ M \implies (\lambda x. X\ (h1\ x)) \in \text{measurable } N\ S$

by *simp*

lemma *joint-distributed-measurable2*:

distributed $M\ (S \otimes_M T)\ (\lambda x. (X\ x, Y\ x))\ f \implies h2 \in \text{measurable } N\ M \implies (\lambda x. Y\ (h2\ x)) \in \text{measurable } N\ T$

by *simp*

lemma *distributed-count-space*:

assumes X : *distributed* $M\ (\text{count-space } A)\ X\ P$ **and** a : $a \in A$ **and** A : *finite* A

shows $P\ a = \text{emeasure } M\ (X - \{a\} \cap \text{space } M)$

proof –

have $\text{emeasure } M\ (X - \{a\} \cap \text{space } M) = \text{emeasure } (\text{distr } M\ (\text{count-space } A)\ X)\ \{a\}$

```

    using X a A by (simp add: emeasure-distr)
  also have ... = emeasure (density (count-space A) P) {a}
    using X by (simp add: distributed-distr-eq-density)
  also have ... =  $\int^+ x. P\ a * \text{indicator}\ \{a\}\ x\ \partial \text{count-space}\ A$ 
    using X a by (auto simp add: emeasure-density distributed-def indicator-def
intro!: nn-integral-cong)
  also have ... = P a
    using X a by (subst nn-integral-cmult-indicator) (auto simp: distributed-def
one-ennreal-def[symmetric] AE-count-space)
  finally show ?thesis ..
qed

```

lemma *distributed-cong-density*:

```

(AE x in N. f x = g x)  $\implies$  g  $\in$  borel-measurable N  $\implies$  f  $\in$  borel-measurable N
 $\implies$ 
distributed M N X f  $\longleftrightarrow$  distributed M N X g
by (auto simp: distributed-def intro!: density-cong)

```

lemma (in prob-space) *distributed-imp-emeasure-nonzero*:

```

assumes X: distributed M MX X Px
shows emeasure MX {x  $\in$  space MX. Px x  $\neq$  0}  $\neq$  0
proof
  note Px = distributed-borel-measurable[OF X]
  interpret X: prob-space distr M MX X
    using distributed-measurable[OF X] by (rule prob-space-distr)

```

```

  assume emeasure MX {x  $\in$  space MX. Px x  $\neq$  0} = 0
  with Px have AE x in MX. Px x = 0
    by (intro AE-I[OF subset-refl]) (auto simp: borel-measurable-ennreal-iff)
  moreover
  from X.emeasure-space-1 have  $\int^+ x. Px\ x\ \partial MX = 1$ 
    unfolding distributed-distr-eq-density[OF X] using Px
    by (subst (asm) emeasure-density)
    (auto simp: borel-measurable-ennreal-iff intro!: integral-cong cong: nn-integral-cong)
  ultimately show False
    by (simp add: nn-integral-cong-AE)
qed

```

lemma *subdensity*:

```

assumes T: T  $\in$  measurable P Q
assumes f: distributed M P X f
assumes g: distributed M Q Y g
assumes Y: Y = T  $\circ$  X
shows AE x in P. g (T x) = 0  $\longrightarrow$  f x = 0
proof -
  have {x $\in$ space Q. g x = 0}  $\in$  null-sets (distr M Q (T  $\circ$  X))
    using g Y by (auto simp: null-sets-density-iff distributed-def)
  also have distr M Q (T  $\circ$  X) = distr (distr M P X) Q T
    using T f[THEN distributed-measurable] by (rule distr-distr[symmetric])

```

finally have $T - \{x \in \text{space } Q. g \ x = 0\} \cap \text{space } P \in \text{null-sets } (\text{distr } M \ P \ X)$
using T **by** $(\text{subst } (\text{asm}) \ \text{null-sets-distr-iff}) \ \text{auto}$
also have $T - \{x \in \text{space } Q. g \ x = 0\} \cap \text{space } P = \{x \in \text{space } P. g \ (T \ x) = 0\}$
using T **by** $(\text{auto dest: measurable-space})$
finally show $?thesis$
using $f \ g$ **by** $(\text{auto simp add: null-sets-density-iff distributed-def})$
qed

lemma *subdensity-real*:

fixes $g :: 'a \Rightarrow \text{real}$ **and** $f :: 'b \Rightarrow \text{real}$
assumes $T: T \in \text{measurable } P \ Q$
assumes $f: \text{distributed } M \ P \ X \ f$
assumes $g: \text{distributed } M \ Q \ Y \ g$
assumes $Y: Y = T \circ X$
shows $(AE \ x \ \text{in } P. 0 \leq g \ (T \ x)) \Longrightarrow (AE \ x \ \text{in } P. 0 \leq f \ x) \Longrightarrow AE \ x \ \text{in } P. g \ (T \ x) = 0 \longrightarrow f \ x = 0$
using *subdensity* $[OF \ T, \ \text{of } M \ X \ \lambda x. \text{ennreal } (f \ x) \ Y \ \lambda x. \text{ennreal } (g \ x)] \ \text{assms}$
by *auto*

lemma *distributed-emeasure*:

$\text{distributed } M \ N \ X \ f \Longrightarrow A \in \text{sets } N \Longrightarrow \text{emeasure } M \ (X - \{A \cap \text{space } M\}) =$
 $(\int^+ x. f \ x * \text{indicator } A \ x \ \partial N)$
by $(\text{auto simp: distributed-distr-eq-density[symmetric] emeasure-density[symmetric] emeasure-distr})$

lemma *distributed-nn-integral*:

$\text{distributed } M \ N \ X \ f \Longrightarrow g \in \text{borel-measurable } N \Longrightarrow (\int^+ x. f \ x * g \ x \ \partial N) =$
 $(\int^+ x. g \ (X \ x) \ \partial M)$
by $(\text{auto simp: distributed-distr-eq-density[symmetric] nn-integral-density[symmetric] nn-integral-distr})$

lemma *distributed-integral*:

$\text{distributed } M \ N \ X \ f \Longrightarrow g \in \text{borel-measurable } N \Longrightarrow (\bigwedge x. x \in \text{space } N \Longrightarrow 0 \leq f \ x) \Longrightarrow$
 $(\int x. f \ x * g \ x \ \partial N) = (\int x. g \ (X \ x) \ \partial M)$
supply *distributed-real-measurable* $[\text{measurable}]$
by $(\text{auto simp: distributed-distr-eq-density[symmetric] integral-real-density[symmetric] integral-distr})$

lemma *distributed-transform-integral*:

assumes $Px: \text{distributed } M \ N \ X \ Px \ \bigwedge x. x \in \text{space } N \Longrightarrow 0 \leq Px \ x$
assumes $\text{distributed } M \ P \ Y \ Py \ \bigwedge x. x \in \text{space } P \Longrightarrow 0 \leq Py \ x$
assumes $Y: Y = T \circ X$ **and** $T: T \in \text{measurable } N \ P$ **and** $f: f \in \text{borel-measurable } P$
shows $(\int x. Py \ x * f \ x \ \partial P) = (\int x. Px \ x * f \ (T \ x) \ \partial N)$
proof –
have $(\int x. Py \ x * f \ x \ \partial P) = (\int x. f \ (Y \ x) \ \partial M)$
by $(\text{rule distributed-integral}) \ \text{fact+}$
also have $\dots = (\int x. f \ (T \ (X \ x)) \ \partial M)$

```

    using Y by simp
    also have ... = (∫ x. Px x * f (T x) ∂N)
    using measurable-comp[OF T f] Px by (intro distributed-integral[symmetric])
    (auto simp: comp-def)
    finally show ?thesis .
qed

```

lemma (in prob-space) distributed-unique:

```

    assumes Px: distributed M S X Px
    assumes Py: distributed M S X Py
    shows AE x in S. Px x = Py x
proof -
    interpret X: prob-space distr M S X
    using Px by (intro prob-space-distr) simp
    have sigma-finite-measure (distr M S X) ..
    with sigma-finite-density-unique[of Px S Py ] Px Py
    show ?thesis
    by (auto simp: distributed-def)
qed

```

lemma (in prob-space) distributed-jointI:

```

    assumes sigma-finite-measure S sigma-finite-measure T
    assumes X[measurable]: X ∈ measurable M S and Y[measurable]: Y ∈ measurable M T
    assumes [measurable]: f ∈ borel-measurable (S ⊗M T) and f: AE x in S ⊗M T. 0 ≤ f x
    assumes eq: ⋀A B. A ∈ sets S ⇒ B ∈ sets T ⇒
        emeasure M {x ∈ space M. X x ∈ A ∧ Y x ∈ B} = (∫+x. (∫+y. f (x, y) *
        indicator B y ∂T) * indicator A x ∂S)
    shows distributed M (S ⊗M T) (λx. (X x, Y x)) f
    unfolding distributed-def
proof safe
    interpret S: sigma-finite-measure S by fact
    interpret T: sigma-finite-measure T by fact
    interpret ST: pair-sigma-finite S T ..

```

from ST.sigma-finite-up-in-pair-measure-generator

```

    obtain F :: nat ⇒ ('b × 'c) set
    where F: range F ⊆ {A × B | A B. A ∈ sets S ∧ B ∈ sets T} ∧ incseq F ∧
        ⋃ (range F) = space S × space T ∧ (∀ i. emeasure (S ⊗M T) (F i) ≠ ∞) ..
    let ?E = {a × b | a b. a ∈ sets S ∧ b ∈ sets T}
    let ?P = S ⊗M T
    show distr M ?P (λx. (X x, Y x)) = density ?P f (is ?L = ?R)
    proof (rule measure-eqI-generator-eq[OF Int-stable-pair-measure-generator[of S
    T]])
        show ?E ⊆ Pow (space ?P)
        using sets.space-closed[of S] sets.space-closed[of T] by (auto simp: space-pair-measure)
        show sets ?L = sigma-sets (space ?P) ?E
        by (simp add: sets-pair-measure space-pair-measure)
    qed

```

```

then show sets ?R = sigma-sets (space ?P) ?E
  by simp
next
  interpret L: prob-space ?L
    by (rule prob-space-distr) (auto intro!: measurable-Pair)
  show range F  $\subseteq$  ?E  $(\bigcup i. F i) = \text{space } ?P \wedge i. \text{emeasure } ?L (F i) \neq \infty$ 
    using F by (auto simp: space-pair-measure)
next
  fix E assume E  $\in$  ?E
  then obtain A B where E[simp]: E = A  $\times$  B
    and A[measurable]: A  $\in$  sets S and B[measurable]: B  $\in$  sets T by auto
  have emeasure ?L E = emeasure M {x  $\in$  space M. X x  $\in$  A  $\wedge$  Y x  $\in$  B}
    by (auto intro!: arg-cong[where f=emeasure M] simp add: emeasure-distr
    measurable-Pair)
  also have ... =  $(\int^+ x. (\int^+ y. (f (x, y) * \text{indicator } B y) * \text{indicator } A x \partial T)$ 
     $\partial S)$ 
    using f by (auto simp add: eq nn-integral-multc intro!: nn-integral-cong)
  also have ... = emeasure ?R E
    by (auto simp add: emeasure-density T.nn-integral-fst[symmetric]
    intro!: nn-integral-cong split: split-indicator)
  finally show emeasure ?L E = emeasure ?R E .
qed
qed (auto simp: f)

lemma (in prob-space) distributed-swap:
  assumes sigma-finite-measure S sigma-finite-measure T
  assumes Pxy: distributed M (S  $\otimes_M$  T)  $(\lambda x. (X x, Y x))$  Pxy
  shows distributed M (T  $\otimes_M$  S)  $(\lambda x. (Y x, X x))$   $(\lambda(x, y). Pxy (y, x))$ 
proof –
  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..
  interpret TS: pair-sigma-finite T S ..

  note Pxy[measurable]
  show ?thesis
    apply (subst TS.distr-pair-swap)
    unfolding distributed-def
  proof safe
    let ?D = distr (S  $\otimes_M$  T) (T  $\otimes_M$  S)  $(\lambda(x, y). (y, x))$ 
    show 1:  $(\lambda(x, y). Pxy (y, x)) \in \text{borel-measurable } ?D$ 
      by auto
    show 2: random-variable (distr (S  $\otimes_M$  T) (T  $\otimes_M$  S)  $(\lambda(x, y). (y, x)))$   $(\lambda x.$ 
     $(Y x, X x))$ 
      using Pxy by auto
    { fix A assume A: A  $\in$  sets (T  $\otimes_M$  S)
      let ?B =  $(\lambda(x, y). (y, x)) - ' A \cap \text{space } (S \otimes_M T)$ 
      from sets.sets-into-space[OF A]
      have emeasure M  $(\lambda x. (Y x, X x)) - ' A \cap \text{space } M) =$ 

```

```

    emeasure M ((λx. (X x, Y x)) - ‘ ?B ∩ space M)
  by (auto intro!: arg-cong2[where f=emeasure] simp: space-pair-measure)
also have ... = (∫+ x. Pxy x * indicator ?B x ∂(S ⊗M T))
  using Pxy A by (intro distributed-emeasure) auto
finally have emeasure M ((λx. (Y x, X x)) - ‘ A ∩ space M) =
  (∫+ x. Pxy x * indicator A (snd x, fst x) ∂(S ⊗M T))
  by (auto intro!: nn-integral-cong split: split-indicator) }
note * = this
show distr M ?D (λx. (Y x, X x)) = density ?D (λ(x, y). Pxy (y, x))
  apply (intro measure-eqI)
  apply (simp-all add: emeasure-distr[OF 2] emeasure-density[OF 1])
  apply (subst nn-integral-distr)
  apply (auto intro!: * simp: comp-def split-beta)
done
qed
qed

lemma (in prob-space) distr-marginal1:
  assumes sigma-finite-measure S sigma-finite-measure T
  assumes Pxy: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
  defines Px ≡ λx. (∫+ z. Pxy (x, z) ∂T)
  shows distributed M S X Px
  unfolding distributed-def
proof safe
  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..

  note Pxy[measurable]
  show X: X ∈ measurable M S by simp

  show borel: Px ∈ borel-measurable S
    by (auto intro!: T.nn-integral-fst simp: Px-def)

  interpret Pxy: prob-space distr M (S ⊗M T) (λx. (X x, Y x))
    by (intro prob-space-distr) simp

  show distr M S X = density S Px
  proof (rule measure-eqI)
    fix A assume A: A ∈ sets (distr M S X)
    with X measurable-space[of Y M T]
    have emeasure (distr M S X) A = emeasure (distr M (S ⊗M T) (λx. (X x,
    Y x))) (A × space T)
      by (auto simp add: emeasure-distr intro!: arg-cong[where f=emeasure M])
    also have ... = emeasure (density (S ⊗M T) Pxy) (A × space T)
      using Pxy by (simp add: distributed-def)
    also have ... = ∫+ x. ∫+ y. Pxy (x, y) * indicator (A × space T) (x, y) ∂T
      ∂S
      using A borel Pxy

```


by (simp add: emeasure-density T.nn-integral-fst[symmetric])
 also have $\dots = \int^+ x. Px\ x * \text{indicator } A\ x\ \partial S$
 proof (rule nn-integral-cong)
 fix x assume $x \in \text{space } S$
 moreover have $\text{eq: } \bigwedge y. y \in \text{space } T \implies \text{indicator } (A \times \text{space } T)\ (x, y) = \text{indicator } A\ x$
 by (auto simp: indicator-def)
 ultimately have $(\int^+ y. Pxy\ (x, y) * \text{indicator } (A \times \text{space } T)\ (x, y)\ \partial T) = (\int^+ y. Pxy\ (x, y)\ \partial T) * \text{indicator } A\ x$
 by (simp add: eq nn-integral-multc cong: nn-integral-cong)
 also have $(\int^+ y. Pxy\ (x, y)\ \partial T) = Px\ x$
 by (simp add: Px-def)
 finally show $(\int^+ y. Pxy\ (x, y) * \text{indicator } (A \times \text{space } T)\ (x, y)\ \partial T) = Px\ x * \text{indicator } A\ x$.
 qed
 finally show $\text{emeasure } (\text{distr } M\ S\ X)\ A = \text{emeasure } (\text{density } S\ Px)\ A$
 using A borel Pxy by (simp add: emeasure-density)
 qed simp
 qed

lemma (in prob-space) distr-marginal2:
 assumes S: sigma-finite-measure S and T: sigma-finite-measure T
 assumes Pxy: distributed M (S \otimes_M T) ($\lambda x. (X\ x, Y\ x)$) Pxy
 shows distributed M T Y ($\lambda y. (\int^+ x. Pxy\ (x, y)\ \partial S)$)
 using distr-marginal1[OF T S distributed-swap[OF S T]] Pxy by simp

lemma (in prob-space) distributed-marginal-eq-joint1:
 assumes T: sigma-finite-measure T
 assumes S: sigma-finite-measure S
 assumes Px: distributed M S X Px
 assumes Pxy: distributed M (S \otimes_M T) ($\lambda x. (X\ x, Y\ x)$) Pxy
 shows AE x in S. Px x = $(\int^+ y. Pxy\ (x, y)\ \partial T)$
 using Px distr-marginal1[OF S T Pxy] by (rule distributed-unique)

lemma (in prob-space) distributed-marginal-eq-joint2:
 assumes T: sigma-finite-measure T
 assumes S: sigma-finite-measure S
 assumes Py: distributed M T Y Py
 assumes Pxy: distributed M (S \otimes_M T) ($\lambda x. (X\ x, Y\ x)$) Pxy
 shows AE y in T. Py y = $(\int^+ x. Pxy\ (x, y)\ \partial S)$
 using Py distr-marginal2[OF S T Pxy] by (rule distributed-unique)

lemma (in prob-space) distributed-joint-indep':
 assumes S: sigma-finite-measure S and T: sigma-finite-measure T
 assumes X[measurable]: distributed M S X Px and Y[measurable]: distributed M T Y Py
 assumes indep: $\text{distr } M\ S\ X \otimes_M \text{distr } M\ T\ Y = \text{distr } M\ (S \otimes_M T)\ (\lambda x. (X\ x, Y\ x))$
 shows distributed M (S \otimes_M T) ($\lambda x. (X\ x, Y\ x)$) ($\lambda(x, y). Px\ x * Py\ y$)

unfolding *distributed-def*

proof *safe*

interpret S : *sigma-finite-measure* S **by** *fact*

interpret T : *sigma-finite-measure* T **by** *fact*

interpret ST : *pair-sigma-finite* S T ..

interpret X : *prob-space density* S Px

unfolding *distributed-distr-eq-density*[OF X , *symmetric*]

by (*rule prob-space-distr*) *simp*

have $sf\text{-}X$: *sigma-finite-measure* (*density* S Px) ..

interpret Y : *prob-space density* T Py

unfolding *distributed-distr-eq-density*[OF Y , *symmetric*]

by (*rule prob-space-distr*) *simp*

have $sf\text{-}Y$: *sigma-finite-measure* (*density* T Py) ..

show $distr\ M\ (S \otimes_M T)\ (\lambda x. (X\ x, Y\ x)) = density\ (S \otimes_M T)\ (\lambda(x, y). Px\ x * Py\ y)$

unfolding *indep*[*symmetric*] *distributed-distr-eq-density*[OF X] *distributed-distr-eq-density*[OF Y]

using *distributed-borel-measurable*[OF X]

using *distributed-borel-measurable*[OF Y]

by (*rule pair-measure-density*[OF - - $T\ sf\text{-}Y$])

show *random-variable* $(S \otimes_M T)\ (\lambda x. (X\ x, Y\ x))$ **by** *auto*

show $Pxy: (\lambda(x, y). Px\ x * Py\ y) \in borel\text{-}measurable\ (S \otimes_M T)$ **by** *auto*
qed

lemma *distributed-integrable*:

distributed $M\ N\ X\ f \implies g \in borel\text{-}measurable\ N \implies (\bigwedge x. x \in space\ N \implies 0 \leq f\ x) \implies$

integrable $N\ (\lambda x. f\ x * g\ x) \longleftrightarrow integrable\ M\ (\lambda x. g\ (X\ x))$

supply *distributed-real-measurable*[*measurable*]

by (*auto simp: distributed-distr-eq-density*[*symmetric*] *integrable-real-density*[*symmetric*] *integrable-distr-eq*)

lemma *distributed-transform-integrable*:

assumes Px : *distributed* $M\ N\ X\ Px \bigwedge x. x \in space\ N \implies 0 \leq Px\ x$

assumes *distributed* $M\ P\ Y\ Py \bigwedge x. x \in space\ P \implies 0 \leq Py\ x$

assumes Y : $Y = (\lambda x. T\ (X\ x))$ **and** T : $T \in measurable\ N\ P$ **and** f : $f \in borel\text{-}measurable\ P$

shows *integrable* $P\ (\lambda x. Py\ x * f\ x) \longleftrightarrow integrable\ N\ (\lambda x. Px\ x * f\ (T\ x))$

proof -

have *integrable* $P\ (\lambda x. Py\ x * f\ x) \longleftrightarrow integrable\ M\ (\lambda x. f\ (Y\ x))$

by (*rule distributed-integrable*) *fact+*

also have $\dots \longleftrightarrow integrable\ M\ (\lambda x. f\ (T\ (X\ x)))$

using Y **by** *simp*

also have $\dots \longleftrightarrow integrable\ N\ (\lambda x. Px\ x * f\ (T\ x))$

using *measurable-comp*[*OF T f*] *Px* **by** (*intro distributed-integrable*[*symmetric*])
(auto simp: comp-def)
finally show *?thesis* .
qed

lemma *distributed-integrable-var*:

fixes *X* :: '*a* \Rightarrow *real*
shows *distributed M lborel X* ($\lambda x. \text{ennreal } (f\ x)$) \implies ($\bigwedge x. 0 \leq f\ x$) \implies
integrable lborel ($\lambda x. f\ x * x$) \implies *integrable M X*
using *distributed-integrable*[*of M lborel X f* $\lambda x. x$] **by** *simp*

lemma (*in prob-space*) *distributed-variance*:

fixes *f*::*real* \Rightarrow *real*
assumes *D*: *distributed M lborel X f* **and** [*simp*]: $\bigwedge x. 0 \leq f\ x$
shows *variance X* = ($\int x. x^2 * f\ (x + \text{expectation } X)\ \partial \text{lborel}$)
proof (*subst distributed-integral*[*OF D, symmetric*])
show ($\int x. f\ x * (x - \text{expectation } X)^2\ \partial \text{lborel}$) = ($\int x. x^2 * f\ (x + \text{expectation } X)\ \partial \text{lborel}$)
by (*subst lborel-integral-real-affine*[**where** *c=1* **and** *t=expectation X*]) (*auto simp: ac-simps*)
qed *simp-all*

lemma (*in prob-space*) *variance-affine*:

fixes *f*::*real* \Rightarrow *real*
assumes [*arith*]: *b* $\neq 0$
assumes *D*[*intro*]: *distributed M lborel X f*
assumes [*simp*]: *prob-space* (*density lborel f*)
assumes *I*[*simp*]: *integrable M X*
assumes *I2*[*simp*]: *integrable M* ($\lambda x. (X\ x)^2$)
shows *variance* ($\lambda x. a + b * X\ x$) = $b^2 * \text{variance } X$
by (*subst variance-eq*)
(auto simp: power2-sum power-mult-distrib prob-space variance-eq right-diff-distrib)

definition

simple-distributed M X f \longleftrightarrow
 $(\forall x. 0 \leq f\ x) \wedge$
distributed M (*count-space* (*X'space M*)) *X* ($\lambda x. \text{ennreal } (f\ x)$) \wedge
finite (*X'space M*)

lemma *simple-distributed-nonneg*[*dest*]: *simple-distributed M X f* $\implies 0 \leq f\ x$
by (*auto simp: simple-distributed-def*)

lemma *simple-distributed*:

simple-distributed M X Px \implies *distributed M* (*count-space* (*X'space M*)) *X Px*
unfolding *simple-distributed-def* **by** *auto*

lemma *simple-distributed-finite*[*dest*]: *simple-distributed M X P* \implies *finite* (*X'space M*)

by (*simp add: simple-distributed-def*)

lemma (in prob-space) distributed-simple-function-superset:

assumes X : simple-function $M X \bigwedge x. x \in X \text{ ‘ space } M \implies P x = \text{measure } M (X - \{x\} \cap \text{space } M)$

assumes A : $X \text{ ‘ space } M \subseteq A$ finite A

defines $S \equiv \text{count-space } A$ **and** $P' \equiv (\lambda x. \text{if } x \in X \text{ ‘ space } M \text{ then } P x \text{ else } 0)$

shows distributed $M S X P'$

unfolding distributed-def

proof safe

show $(\lambda x. \text{ennreal } (P' x)) \in \text{borel-measurable } S$ **unfolding** S -def **by** simp

show $\text{distr } M S X = \text{density } S P'$

proof (rule measure-eqI-finite)

show $\text{sets } (\text{distr } M S X) = \text{Pow } A$ **sets** $(\text{density } S P') = \text{Pow } A$

using A **unfolding** S -def **by** auto

show finite A **by** fact

fix a **assume** a : $a \in A$

then have $a \notin X \text{ ‘ space } M \implies X - \{a\} \cap \text{space } M = \{\}$ **by** auto

with $A a X$ **have** $\text{emeasure } (\text{distr } M S X) \{a\} = P' a$

by (subst emeasure-distr)

(auto simp add: S -def P' -def simple-functionD emeasure-eq-measure measurable-count-space-eq2

intro!: arg-cong[where $f = \text{prob}$])

also have $\dots = (\int^+ x. \text{ennreal } (P' a) * \text{indicator } \{a\} x \partial S)$

using $A X a$

by (subst nn-integral-cmult-indicator)

(auto simp: S -def P' -def simple-distributed-def simple-functionD measure-nonneg)

also have $\dots = (\int^+ x. \text{ennreal } (P' x) * \text{indicator } \{a\} x \partial S)$

by (auto simp: indicator-def intro!: nn-integral-cong)

also have $\dots = \text{emeasure } (\text{density } S P') \{a\}$

using $a A$ **by** (intro emeasure-density[symmetric]) (auto simp: S -def)

finally show $\text{emeasure } (\text{distr } M S X) \{a\} = \text{emeasure } (\text{density } S P') \{a\}$.

qed

show random-variable $S X$

using $X(1) A$ **by** (auto simp: measurable-def simple-functionD S -def)

qed

lemma (in prob-space) simple-distributedI:

assumes X : simple-function $M X$

$\bigwedge x. 0 \leq P x$

$\bigwedge x. x \in X \text{ ‘ space } M \implies P x = \text{measure } M (X - \{x\} \cap \text{space } M)$

shows simple-distributed $M X P$

unfolding simple-distributed-def

proof (safe intro!: X)

have distributed M (count-space $(X \text{ ‘ space } M)$) X $(\lambda x. \text{ennreal } (\text{if } x \in X \text{ ‘ space } M \text{ then } P x \text{ else } 0))$

(is ?A)

using simple-functionD[OF $X(1)$] **by** (intro distributed-simple-function-superset[OF $X(1, \mathcal{P})$]) auto

also have $?A \longleftrightarrow \text{distributed } M \text{ (count-space } (X \text{ ‘ space } M)) \text{ } X \text{ (}\lambda x. \text{ ennreal } (P \text{ } x))$

by (rule distributed-cong-density) auto

finally show ...

qed (rule simple-functionD[OF X(1)])

lemma simple-distributed-joint-finite:

assumes X : simple-distributed $M \text{ (}\lambda x. (X \text{ } x, Y \text{ } x)) \text{ } Px$

shows finite $(X \text{ ‘ space } M)$ finite $(Y \text{ ‘ space } M)$

proof –

have finite $((\lambda x. (X \text{ } x, Y \text{ } x)) \text{ ‘ space } M)$

using X **by** (auto simp: simple-distributed-def simple-functionD)

then have finite $(fst \text{ ‘ } (\lambda x. (X \text{ } x, Y \text{ } x)) \text{ ‘ space } M)$ finite $(snd \text{ ‘ } (\lambda x. (X \text{ } x, Y \text{ } x)) \text{ ‘ space } M)$

by auto

then show fin: finite $(X \text{ ‘ space } M)$ finite $(Y \text{ ‘ space } M)$

by (auto simp: image-image)

qed

lemma simple-distributed-joint2-finite:

assumes X : simple-distributed $M \text{ (}\lambda x. (X \text{ } x, Y \text{ } x, Z \text{ } x)) \text{ } Px$

shows finite $(X \text{ ‘ space } M)$ finite $(Y \text{ ‘ space } M)$ finite $(Z \text{ ‘ space } M)$

proof –

have finite $((\lambda x. (X \text{ } x, Y \text{ } x, Z \text{ } x)) \text{ ‘ space } M)$

using X **by** (auto simp: simple-distributed-def simple-functionD)

then have finite $(fst \text{ ‘ } (\lambda x. (X \text{ } x, Y \text{ } x, Z \text{ } x)) \text{ ‘ space } M)$

finite $((fst \circ snd) \text{ ‘ } (\lambda x. (X \text{ } x, Y \text{ } x, Z \text{ } x)) \text{ ‘ space } M)$

finite $((snd \circ snd) \text{ ‘ } (\lambda x. (X \text{ } x, Y \text{ } x, Z \text{ } x)) \text{ ‘ space } M)$

by auto

then show fin: finite $(X \text{ ‘ space } M)$ finite $(Y \text{ ‘ space } M)$ finite $(Z \text{ ‘ space } M)$

by (auto simp: image-image)

qed

lemma simple-distributed-simple-function:

simple-distributed $M \text{ } X \text{ } Px \implies \text{simple-function } M \text{ } X$

unfolding simple-distributed-def distributed-def

by (auto simp: simple-function-def measurable-count-space-eq2)

lemma simple-distributed-measure:

simple-distributed $M \text{ } X \text{ } P \implies a \in X \text{ ‘ space } M \implies P \text{ } a = \text{measure } M \text{ (} X \text{ – ‘ } \{a\} \cap \text{space } M)$

using distributed-count-space[of $M \text{ } X \text{ ‘ space } M \text{ } X \text{ } P \text{ } a$, symmetric]

by (auto simp: simple-distributed-def measure-def)

lemma (in prob-space) simple-distributed-joint:

assumes X : simple-distributed $M \text{ (}\lambda x. (X \text{ } x, Y \text{ } x)) \text{ } Px$

defines $S \equiv \text{count-space } (X \text{ ‘ space } M) \otimes_M \text{count-space } (Y \text{ ‘ space } M)$

defines $P \equiv (\lambda x. \text{if } x \in (\lambda x. (X \text{ } x, Y \text{ } x)) \text{ ‘ space } M \text{ then } P \text{ } x \text{ else } 0)$

shows distributed $M \text{ } S \text{ (}\lambda x. (X \text{ } x, Y \text{ } x)) \text{ } P$

proof –
from *simple-distributed-joint-finite*[*OF X, simp*]
have *S-eq*: $S = \text{count-space } (X' \text{space } M \times Y' \text{space } M)$
by (*simp add: S-def pair-measure-count-space*)
show ?thesis
unfolding *S-eq P-def*
proof (*rule distributed-simple-function-superset*)
show *simple-function M* ($\lambda x. (X x, Y x)$)
using *X* **by** (*rule simple-distributed-simple-function*)
fix *x* **assume** $x \in (\lambda x. (X x, Y x))' \text{space } M$
from *simple-distributed-measure*[*OF X this*]
show $Px x = \text{prob } ((\lambda x. (X x, Y x)) -' \{x\} \cap \text{space } M)$.
qed *auto*
qed

lemma (*in prob-space*) *simple-distributed-joint2*:
assumes *X*: *simple-distributed M* ($\lambda x. (X x, Y x, Z x)$) *Px*
defines $S \equiv \text{count-space } (X' \text{space } M) \otimes_M \text{count-space } (Y' \text{space } M) \otimes_M \text{count-space } (Z' \text{space } M)$
defines $P \equiv (\lambda x. \text{if } x \in (\lambda x. (X x, Y x, Z x))' \text{space } M \text{ then } Px x \text{ else } 0)$
shows *distributed M S* ($\lambda x. (X x, Y x, Z x)$) *P*
proof –
from *simple-distributed-joint2-finite*[*OF X, simp*]
have *S-eq*: $S = \text{count-space } (X' \text{space } M \times Y' \text{space } M \times Z' \text{space } M)$
by (*simp add: S-def pair-measure-count-space*)
show ?thesis
unfolding *S-eq P-def*
proof (*rule distributed-simple-function-superset*)
show *simple-function M* ($\lambda x. (X x, Y x, Z x)$)
using *X* **by** (*rule simple-distributed-simple-function*)
fix *x* **assume** $x \in (\lambda x. (X x, Y x, Z x))' \text{space } M$
from *simple-distributed-measure*[*OF X this*]
show $Px x = \text{prob } ((\lambda x. (X x, Y x, Z x)) -' \{x\} \cap \text{space } M)$.
qed *auto*
qed

lemma (*in prob-space*) *simple-distributed-sum-space*:
assumes *X*: *simple-distributed M X f*
shows $\text{sum } f (X' \text{space } M) = 1$
proof –
from *X* **have** $\text{sum } f (X' \text{space } M) = \text{prob } (\bigcup i \in X' \text{space } M. X -' \{i\} \cap \text{space } M)$
by (*subst finite-measure-finite-Union*)
(auto simp add: disjoint-family-on-def simple-distributed-measure simple-distributed-simple-function simple-functionD)
intro!: *sum.cong arg-cong*[**where** *f=prob*])
also **have** $\dots = \text{prob } (\text{space } M)$
by (*auto intro!: arg-cong*[**where** *f=prob*])
finally **show** ?thesis
using *emeasure-space-1* **by** (*simp add: emeasure-eq-measure*)

qed

lemma (in prob-space) distributed-marginal-eq-joint-simple:

assumes Px : simple-function M X
 assumes Py : simple-distributed M Y Py
 assumes Pxy : simple-distributed M $(\lambda x. (X\ x, Y\ x))$ Pxy
 assumes y : $y \in Y$ space M
 shows $Py\ y = (\sum_{x \in X \text{ space } M}. \text{if } (x, y) \in (\lambda x. (X\ x, Y\ x)) \text{ ' space } M \text{ then } Pxy\ (x, y) \text{ else } 0)$
proof –
 note $Px = \text{simple-distributedI}[OF\ Px\ \text{measure-nonneg refl}]$
 have $AE\ y\ \text{in count-space } (Y \text{ ' space } M). \text{ennreal } (Py\ y) =$
 $\int^+ x. \text{ennreal } (\text{if } (x, y) \in (\lambda x. (X\ x, Y\ x)) \text{ ' space } M \text{ then } Pxy\ (x, y) \text{ else } 0)$
 $\partial \text{count-space } (X \text{ ' space } M)$
 using sigma-finite-measure-count-space-finite sigma-finite-measure-count-space-finite
 simple-distributed[OF Py] simple-distributed-joint[OF Pxy]
 by (rule distributed-marginal-eq-joint2)
 (auto intro: $Py\ Px$ simple-distributed-finite)
 then have $\text{ennreal } (Py\ y) =$
 $(\sum_{x \in X \text{ space } M}. \text{ennreal } (\text{if } (x, y) \in (\lambda x. (X\ x, Y\ x)) \text{ ' space } M \text{ then } Pxy\ (x, y) \text{ else } 0))$
 using $y\ Px$ [THEN simple-distributed-finite]
 by (auto simp: AE-count-space nn-integral-count-space-finite)
 also have $\dots = (\sum_{x \in X \text{ space } M}. \text{if } (x, y) \in (\lambda x. (X\ x, Y\ x)) \text{ ' space } M \text{ then } Pxy\ (x, y) \text{ else } 0)$
 using Pxy by (intro sum-ennreal) auto
 finally show ?thesis
 using simple-distributed-nonneg[OF Py] simple-distributed-nonneg[OF Pxy]
 by (subst (asm) ennreal-inj) (auto intro!: sum-nonneg)

qed

lemma distributedI-real:

fixes $f :: 'a \Rightarrow \text{real}$
 assumes gen : sets $M1 = \text{sigma-sets } (\text{space } M1)\ E$ and Int-stable E
 and A : range $A \subseteq E$ $(\bigcup_{i::\text{nat.}} A\ i) = \text{space } M1 \wedge i. \text{emeasure } (\text{distr } M\ M1\ X)$
 $(A\ i) \neq \infty$
 and X : $X \in \text{measurable } M\ M1$
 and f : $f \in \text{borel-measurable } M1$ AE x in $M1$. $0 \leq f\ x$
 and eq: $\bigwedge A. A \in E \implies \text{emeasure } M\ (X \text{ - ' } A \cap \text{space } M) = (\int^+ x. f\ x * \text{indicator } A\ x\ \partial M1)$
 shows distributed $M\ M1\ X\ f$
 unfolding distributed-def
proof (intro conjI)
 show $\text{distr } M\ M1\ X = \text{density } M1\ f$
proof (rule measure-eqI-generator-eq[where $A=A$])
 { fix A assume $A: A \in E$
 then have $A \in \text{sigma-sets } (\text{space } M1)\ E$ by auto
 then have $A \in \text{sets } M1$
 using gen by simp

```

with  $f A$   $eq[of A]$   $X$  show  $emeasure (distr M M1 X) A = emeasure (density$ 
 $M1 f) A$ 
  by ( $auto simp add: emeasure-distr emeasure-density ennreal-indicator$ 
     $intro!: nn-integral-cong split: split-indicator$ ) }
note  $eq-E = this$ 
show  $Int-stable E$  by  $fact$ 
{ fix  $e$  assume  $e \in E$ 
  then have  $e \in \sigma\text{-sets} (space M1) E$  by  $auto$ 
  then have  $e \in \text{sets } M1$  unfolding  $gen$  .
  then have  $e \subseteq space M1$  by ( $rule \text{sets.sets-into-space}$ ) }
then show  $E \subseteq Pow (space M1)$  by  $auto$ 
show  $\text{sets} (distr M M1 X) = \sigma\text{-sets} (space M1) E$ 
   $\text{sets} (density M1 (\lambda x. ennreal (f x))) = \sigma\text{-sets} (space M1) E$ 
  unfolding  $gen[symmetric]$  by  $auto$ 
qed  $fact+$ 
qed ( $insert X f, auto$ )

```

lemma $distributedI\text{-borel-atMost}$:

```

fixes  $f :: real \Rightarrow real$ 
assumes [ $measurable$ ]:  $X \in \text{borel-measurable } M$ 
and [ $measurable$ ]:  $f \in \text{borel-measurable borel}$  and  $f[simp]: AE x \text{ in } lborel. 0 \leq$ 
 $f x$ 
and  $g\text{-eq}: \bigwedge a. (\int^+ x. f x * indicator \{..a\} x \partial lborel) = ennreal (g a)$ 
and  $M\text{-eq}: \bigwedge a. emeasure M \{x \in space M. X x \leq a\} = ennreal (g a)$ 
shows  $distributed M lborel X f$ 
proof ( $rule distributedI\text{-real}$ )
show  $\text{sets} (lborel::real \text{ measure}) = \sigma\text{-sets} (space lborel) (range atMost)$ 
  by ( $simp add: borel-eq-atMost$ )
show  $Int-stable (range atMost :: real \text{ set set})$ 
  by ( $auto simp: Int-stable-def$ )
have  $image\text{-eq}: \bigwedge a. (X - ' \{..a\} \cap space M) = \{x \in space M. X x \leq a\}$  by  $auto$ 
define  $A$  where  $A i = \{.. real i\}$  for  $i :: nat$ 
then show  $range A \subseteq range atMost (\bigcup i. A i) = space lborel$ 
   $\bigwedge i. emeasure (distr M lborel X) (A i) \neq \infty$ 
  by ( $auto simp: real-arch-simple emeasure-distr image\text{-eq } M\text{-eq}$ )

fix  $A :: real \text{ set}$  assume  $A \in range atMost$ 
then obtain  $a$  where  $A = \{..a\}$  by  $auto$ 
show  $emeasure M (X - ' A \cap space M) = (\int^+ x. f x * indicator A x \partial lborel)$ 
  unfolding  $image\text{-eq } A M\text{-eq } g\text{-eq} ..$ 
qed  $auto$ 

```

lemma ($in \text{prob-space}$) $uniform-distributed-params$:

```

assumes  $X: distributed M MX X (\lambda x. indicator A x / measure MX A)$ 
shows  $A \in \text{sets } MX \text{ measure } MX A \neq 0$ 
proof –
  interpret  $X: prob-space distr M MX X$ 
  using  $distributed-measurable[OF X]$  by ( $rule prob-space-distr$ )

```



```

show measure MX A  $\neq 0$ 
proof
  assume measure MX A = 0
  with X.emeasure-space-1 X.prob-space distributed-distr-eq-density[OF X]
  show False
    by (simp add: emeasure-density zero-enreal-def[symmetric])
  qed
  with measure-notin-sets[of A MX] show A  $\in$  sets MX
    by blast
qed

lemma prob-space-uniform-measure:
  assumes A: emeasure M A  $\neq 0$  emeasure M A  $\neq \infty$ 
  shows prob-space (uniform-measure M A)
proof
  show emeasure (uniform-measure M A) (space (uniform-measure M A)) = 1
    using emeasure-uniform-measure[OF emeasure-neq-0-sets[OF A(1)], of space
M]
    using sets.sets-into-space[OF emeasure-neq-0-sets[OF A(1)]] A
    by (simp add: Int-absorb2 less-top)
qed

lemma prob-space-uniform-count-measure: finite A  $\implies$  A  $\neq \{\}$   $\implies$  prob-space
(uniform-count-measure A)
  by standard (auto simp: emeasure-uniform-count-measure space-uniform-count-measure
one-enreal-def)

lemma (in prob-space) measure-uniform-measure-eq-cond-prob:
  assumes [measurable]: Measurable.pred M P Measurable.pred M Q
  shows  $\mathcal{P}(x \text{ in } \text{uniform-measure } M \{x \in \text{space } M. Q \ x\}. P \ x) = \mathcal{P}(x \text{ in } M. P \ x \mid$ 
Q x)
proof cases
  assume Q: measure M  $\{x \in \text{space } M. Q \ x\} = 0$ 
  then have *: AE x in M.  $\neg Q \ x$ 
    by (simp add: prob-eq-0)
  then have density M ( $\lambda x. \text{indicator } \{x \in \text{space } M. Q \ x\} \ x / \text{emeasure } M \{x \in$ 
space M. Q x\}) = density M ( $\lambda x. 0$ )
    by (intro density-cong) auto
  with * show ?thesis
    unfolding uniform-measure-def
    by (simp add: emeasure-density measure-def cond-prob-def emeasure-eq-0-AE)
next
  assume Q: measure M  $\{x \in \text{space } M. Q \ x\} \neq 0$ 
  then show  $\mathcal{P}(x \text{ in } \text{uniform-measure } M \{x \in \text{space } M. Q \ x\}. P \ x) = \text{cond-prob}$ 
M P Q
    by (subst measure-uniform-measure)
    (auto simp: emeasure-eq-measure cond-prob-def measure-nonneg intro!: arg-cong[where
f=prob])
qed

```

lemma *prob-space-point-measure*:

finite S $\implies (\bigwedge s. s \in S \implies 0 \leq p\ s) \implies (\sum_{s \in S}. p\ s) = 1 \implies \text{prob-space}$
(point-measure S p)
by (*rule prob-spaceI*) (*simp add: space-point-measure emeasure-point-measure-finite*)

lemma (*in prob-space*) *distr-pair-fst*: $\text{distr } (N \otimes_M M) N\ \text{fst} = N$

proof (*intro measure-eqI*)

fix *A* **assume** *A*: $A \in \text{sets } (\text{distr } (N \otimes_M M) N\ \text{fst})$
from *A* **have** $\text{emeasure } (\text{distr } (N \otimes_M M) N\ \text{fst})\ A = \text{emeasure } (N \otimes_M M)$
 $(A \times \text{space } M)$
by (*auto simp add: emeasure-distr space-pair-measure dest: sets.sets-into-space*
intro!: arg-cong2[where f=emeasure])
with *A* **show** $\text{emeasure } (\text{distr } (N \otimes_M M) N\ \text{fst})\ A = \text{emeasure } N\ A$
by (*simp add: emeasure-pair-measure-Times emeasure-space-1*)
qed *simp*

lemma (*in product-prob-space*) *distr-reorder*:

assumes *inj-on t J* $t \in J \rightarrow K$ *finite K*
shows $\text{distr } (\text{Pi}_M\ K\ M) (\text{Pi}_M\ J\ (\lambda x. M\ (t\ x))) (\lambda \omega. \lambda n \in J. \omega\ (t\ n)) = \text{Pi}_M\ J$
 $(\lambda x. M\ (t\ x))$
proof (*rule product-sigma-finite.PiM-eqI*)
show *product-sigma-finite* $(\lambda x. M\ (t\ x))$..
have $t'J \subseteq K$ **using** *assms* **by** *auto*
then show [*simp*]: *finite J*
by (*rule finite-imageD[OF finite-subset]*) *fact+*
fix *A* **assume** *A*: $\bigwedge i. i \in J \implies A\ i \in \text{sets } (M\ (t\ i))$
moreover have $((\lambda \omega. \lambda n \in J. \omega\ (t\ n)) - ' \text{Pi}_E\ J\ A \cap \text{space } (\text{Pi}_M\ K\ M)) =$
 $(\Pi_E\ i \in K. \text{if } i \in t'J \text{ then } A\ (t\ i) \text{ else } \text{space } (M\ i))$
using *A* *A* [*THEN sets.sets-into-space*] $\langle t \in J \rightarrow K \rangle \langle \text{inj-on } t\ J \rangle$
by (*subst prod-emb-Pi[symmetric]*) (*auto simp: space-PiM PiE-iff the-inv-into-f-f*
prod-emb-def)
ultimately show $\text{distr } (\text{Pi}_M\ K\ M) (\text{Pi}_M\ J\ (\lambda x. M\ (t\ x))) (\lambda \omega. \lambda n \in J. \omega\ (t\ n))$
 $(\text{Pi}_E\ J\ A) = (\prod_{i \in J}. M\ (t\ i)\ (A\ i))$
using *assms*
apply (*subst emeasure-distr*)
apply (*auto intro!: sets-PiM-I-finite simp: Pi-iff*)
apply (*subst emeasure-PiM*)
apply (*auto simp: the-inv-into-f-f <inj-on t J> prod.reindex[OF <inj-on t J>*
if-distrib[where f=emeasure (M -)] prod.If-cases emeasure-space-1 Int-absorb1
 $\langle t'J \subseteq K \rangle$)
done
qed *simp*

lemma (*in product-prob-space*) *distr-restrict*:

$J \subseteq K \implies \text{finite } K \implies (\Pi_M\ i \in J. M\ i) = \text{distr } (\Pi_M\ i \in K. M\ i) (\Pi_M\ i \in J. M\ i)$
 $(\lambda f. \text{restrict } f\ J)$
using *distr-reorder[of $\lambda x. x\ J\ K$]* **by** (*simp add: Pi-iff subset-eq*)

lemma (in *product-prob-space*) *emeasure-prod-emb*[simp]:
assumes $L: J \subseteq L$ *finite* L **and** $X: X \in \text{sets } (Pi_M J M)$
shows $\text{emeasure } (Pi_M L M) (\text{prod-emb } L M J X) = \text{emeasure } (Pi_M J M) X$
by (*subst distr-restrict*[*OF L*])
(simp add: prod-emb-def space-PiM emeasure-distr measurable-restrict-subset L X)

lemma *emeasure-distr-restrict*:
assumes $I \subseteq K$ **and** $Q[\text{measurable-cong}]$: $\text{sets } Q = \text{sets } (Pi_M K M)$ **and**
 $A[\text{measurable}]$: $A \in \text{sets } (Pi_M I M)$
shows $\text{emeasure } (\text{distr } Q (Pi_M I M) (\lambda\omega. \text{restrict } \omega I)) A = \text{emeasure } Q$
(prod-emb K M I A)
using $\langle I \subseteq K \rangle$ *sets-eq-imp-space-eq*[*OF Q*]
by (*subst emeasure-distr*)
(auto simp: measurable-cong-sets[*OF Q*] *prod-emb-def space-PiM*[*symmetric*]
intro!: measurable-restrict)

lemma (in *prob-space*) *prob-space-completion*: *prob-space* (*completion* M)
by (*rule prob-spaceI*) (*simp add: emeasure-space-1*)

lemma *distr-PiM-finite-prob-space*:
assumes *fin*: *finite* I
assumes *product-prob-space* M
assumes *product-prob-space* M'
assumes [*measurable*]: $\bigwedge i. i \in I \implies f \in \text{measurable } (M i) (M' i)$
shows $\text{distr } (Pi_M I M) (Pi_M I M') (\text{compose } I f) = Pi_M I (\lambda i. \text{distr } (M i)$
 $(M' i) f)$
proof –
interpret M : *product-prob-space* M **by** *fact*
interpret M' : *product-prob-space* M' **by** *fact*
define N **where** $N = (\lambda i. \text{if } i \in I \text{ then } \text{distr } (M i) (M' i) f \text{ else } M' i)$
have [*intro*]: *prob-space* ($N i$) **for** i
by (*auto simp: N-def intro!: M.M.prob-space-distr M'.prob-space*)

interpret N : *product-prob-space* N
by (*intro product-prob-spaceI*) (*auto simp: N-def M'.prob-space intro: M.M.prob-space-distr*)

have $\text{distr } (Pi_M I M) (Pi_M I M') (\text{compose } I f) = Pi_M I N$

proof (*rule N.PiM-eqI*)

have $N\text{-events-eq}$: $\text{sets } (Pi_M I N) = \text{sets } (Pi_M I M')$

unfolding $N\text{-def}$ **by** (*intro sets-PiM-cong*) *auto*

also have $\dots = \text{sets } (\text{distr } (Pi_M I M) (Pi_M I M') (\text{compose } I f))$

by *simp*

finally show $\text{sets } (\text{distr } (Pi_M I M) (Pi_M I M') (\text{compose } I f)) = \text{sets } (Pi_M I N)$ **..**

fix A **assume** A : $\bigwedge i. i \in I \implies A i \in N.M.\text{events } i$

have $\text{emeasure } (\text{distr } (Pi_M I M) (Pi_M I M') (\text{compose } I f)) (Pi_E I A) =$
 $\text{emeasure } (Pi_M I M) (\text{compose } I f - ' Pi_E I A \cap \text{space } (Pi_M I M))$

```

proof (intro emeasure-distr)
  show  $\text{compose } I f \in \text{Pi}_M I M \rightarrow_M \text{Pi}_M I M'$ 
    unfolding compose-def by measurable
  show  $\text{Pi}_E I A \in \text{sets } (\text{Pi}_M I M')$ 
    unfolding N-events-eq [symmetric] by (intro sets-PiM-I-finite fin A)
qed
  also have  $\text{compose } I f - ' \text{Pi}_E I A \cap \text{space } (\text{Pi}_M I M) = \text{Pi}_E I (\lambda i. f - ' A i \cap \text{space } (M i))$ 
    using A by (auto simp: space-PiM PiE-def Pi-def extensional-def N-def compose-def)
  also have  $\text{emeasure } (\text{Pi}_M I M) (\text{Pi}_E I (\lambda i. f - ' A i \cap \text{space } (M i))) =$ 
     $(\prod_{i \in I. \text{emeasure } (M i) (f - ' A i \cap \text{space } (M i)))$ 
    using A by (intro M.emeasure-PiM fin) (auto simp: N-def)
  also have  $\dots = (\prod_{i \in I. \text{emeasure } (\text{distr } (M i) (M' i) f) (A i))$ 
    using A by (intro prod.cong emeasure-distr [symmetric]) (auto simp: N-def)
  also have  $\dots = (\prod_{i \in I. \text{emeasure } (N i) (A i))$ 
    unfolding N-def by (intro prod.cong) (auto simp: N-def)
  finally show  $\text{emeasure } (\text{distr } (\text{Pi}_M I M) (\text{Pi}_M I M') (\text{compose } I f)) (\text{Pi}_E I A) = \dots$ 
qed fact+
  also have  $\text{PiM } I N = \text{PiM } I (\lambda i. \text{distr } (M i) (M' i) f)$ 
    by (intro PiM-cong) (auto simp: N-def)
  finally show ?thesis .
qed
end

```

2 Distribution Functions

Shows that the cumulative distribution function (cdf) of a distribution (a measure on the reals) is nondecreasing and right continuous, which tends to 0 and 1 in either direction.

Conversely, every such function is the cdf of a unique distribution. This direction defines the measure in the obvious way on half-open intervals, and then applies the Caratheodory extension theorem.

```

theory Distribution-Functions
  imports Probability-Measure
begin

```

```

lemma UN-Ioc-eq-UNIV:  $(\bigcup n. \{ -\text{real } n <.. \text{real } n \}) = \text{UNIV}$ 
  by auto
  (metis le-less-trans minus-minus neg-less-iff-less not-le real-arch-simple
    of-nat-0-le-iff reals-Archimedean2)

```

2.1 Properties of cdf's

```

definition

```

cdf :: *real measure* \Rightarrow *real* \Rightarrow *real*

where

cdf *M* $\equiv \lambda x. \text{measure } M \{..x\}$

lemma *cdf-def2*: *cdf* *M* *x* = *measure* *M* $\{..x\}$

by (*simp add: cdf-def*)

locale *finite-borel-measure* = *finite-measure* *M* **for** *M* :: *real measure* +

assumes *M-is-borel*: *sets* *M* = *sets borel*

begin

lemma *sets-M[intro]*: *a* \in *sets borel* \implies *a* \in *sets* *M*

using *M-is-borel* **by** *auto*

lemma *cdf-diff-eq*:

assumes *x* < *y*

shows *cdf* *M* *y* - *cdf* *M* *x* = *measure* *M* $\{x<..y\}$

proof -

from *assms* **have** *: $\{..x\} \cup \{x<..y\} = \{..y\}$ **by** *auto*

have *measure* *M* $\{..y\}$ = *measure* *M* $\{..x\}$ + *measure* *M* $\{x<..y\}$

by (*subst finite-measure-Union [symmetric]*, *auto simp add: **)

thus *?thesis*

unfolding *cdf-def* **by** *auto*

qed

lemma *cdf-nondecreasing*: *x* \leq *y* \implies *cdf* *M* *x* \leq *cdf* *M* *y*

unfolding *cdf-def* **by** (*auto intro!: finite-measure-mono*)

lemma *borel-UNIV*: *space* *M* = *UNIV*

by (*metis in-mono sets.sets-into-space space-in-borel top-le M-is-borel*)

lemma *cdf-nonneg*: *cdf* *M* *x* \geq 0

unfolding *cdf-def* **by** (*rule measure-nonneg*)

lemma *cdf-bounded*: *cdf* *M* *x* \leq *measure* *M* (*space* *M*)

unfolding *cdf-def* **by** (*intro bounded-measure*)

lemma *cdf-lim-infty*:

$((\lambda i. \text{cdf } M (\text{real } i)) \longrightarrow \text{measure } M (\text{space } M))$

proof -

have $(\lambda i. \text{cdf } M (\text{real } i)) \longrightarrow \text{measure } M (\bigcup i::\text{nat}. \{.. \text{real } i\})$

unfolding *cdf-def* **by** (*rule finite-Lim-measure-incseq*) (*auto simp: incseq-def*)

also have $(\bigcup i::\text{nat}. \{.. \text{real } i\}) = \text{space } M$

by (*auto simp: borel-UNIV intro: real-arch-simple*)

finally show *?thesis* .

qed

lemma *cdf-lim-at-top*: $(\text{cdf } M \longrightarrow \text{measure } M (\text{space } M)) \text{ at-top}$

by (*rule tendsto-at-topI-sequentially-real*)

(simp-all add: mono-def cdf-nondecreasing cdf-lim-infity)

lemma *cdf-lim-neg-infity*: $((\lambda i. \text{cdf } M (- \text{real } i)) \longrightarrow 0)$

proof –

have $(\lambda i. \text{cdf } M (- \text{real } i)) \longrightarrow \text{measure } M (\bigcap i::\text{nat}. \{.. - \text{real } i\})$

unfolding *cdf-def* by (rule *finite-Lim-measure-decseq*) (auto simp: *decseq-def*)

also have $(\bigcap i::\text{nat}. \{.. - \text{real } i\}) = \{\}$

by auto (metis *leD le-minus-iff reals-Archimedean2*)

finally show *?thesis*

by *simp*

qed

lemma *cdf-lim-at-bot*: $(\text{cdf } M \longrightarrow 0) \text{ at-bot}$

proof –

have *: $((\lambda x :: \text{real}. - \text{cdf } M (- x)) \longrightarrow 0) \text{ at-top}$

by (intro *tendsto-at-topI-sequentially-real monoI*)

(auto simp: *cdf-nondecreasing cdf-lim-neg-infity tendsto-minus-cancel-left[symmetric]*)

from *filterlim-compose* [*OF* *, *OF filterlim-uminus-at-top-at-bot*]

show *?thesis*

unfolding *tendsto-minus-cancel-left[symmetric]* by *simp*

qed

lemma *cdf-is-right-cont*: *continuous* (at-right *a*) (cdf *M*)

unfolding *continuous-within*

proof (rule *tendsto-at-right-sequentially*[**where** *b=a + 1*])

fix *f* :: *nat* \Rightarrow *real* and *x* assume *f*: *decseq* *f f* $\longrightarrow a$

then have $(\lambda n. \text{cdf } M (f\ n)) \longrightarrow \text{measure } M (\bigcap i. \{.. f\ i\})$

using $\langle \text{decseq } f \rangle$ unfolding *cdf-def*

by (intro *finite-Lim-measure-decseq*) (auto simp: *decseq-def*)

also have $(\bigcap i. \{.. f\ i\}) = \{.. a\}$

using *decseq-ge*[*OF f*] by (auto intro: *order-trans LIMSEQ-le-const*[*OF f(2)*])

finally show $(\lambda n. \text{cdf } M (f\ n)) \longrightarrow \text{cdf } M\ a$

by (*simp add: cdf-def*)

qed *simp*

lemma *cdf-at-left*: $(\text{cdf } M \longrightarrow \text{measure } M \{.. < a\}) \text{ (at-left } a)$

proof (rule *tendsto-at-left-sequentially*[*of a - 1*])

fix *f* :: *nat* \Rightarrow *real* and *x* assume *f*: *incseq* *f f* $\longrightarrow a \wedge x. f\ x < a \wedge x. a - 1 < f\ x$

then have $(\lambda n. \text{cdf } M (f\ n)) \longrightarrow \text{measure } M (\bigcup i. \{.. f\ i\})$

using $\langle \text{incseq } f \rangle$ unfolding *cdf-def*

by (intro *finite-Lim-measure-incseq*) (auto simp: *incseq-def*)

also have $(\bigcup i. \{.. f\ i\}) = \{.. < a\}$

by (auto dest!: *order-tendstoD*(1)[*OF f(2)*] *eventually-happens*'[*OF sequentially-bot*])

intro: *less-imp-le le-less-trans f(3)*)

finally show $(\lambda n. \text{cdf } M (f\ n)) \longrightarrow \text{measure } M \{.. < a\}$

by (*simp add: cdf-def*)

qed *auto*

```

lemma isCont-cdf: isCont (cdf M) x  $\longleftrightarrow$  measure M {x} = 0
proof –
  have isCont (cdf M) x  $\longleftrightarrow$  cdf M x = measure M {..x}
    by (auto simp: continuous-at-split cdf-is-right-cont continuous-within[where
s={..< -}])
      cdf-at-left tendsto-unique[OF - cdf-at-left])
  also have cdf M x = measure M {..x}  $\longleftrightarrow$  measure M {x} = 0
    unfolding cdf-def ivl-disj-un(2)[symmetric]
    by (subst finite-measure-Union) auto
  finally show ?thesis .
qed

lemma countable-atoms: countable {x. measure M {x} > 0}
  using countable-support unfolding zero-less-measure-iff .

end

locale real-distribution = prob-space M for M :: real measure +
  assumes events-eq-borel [simp, measurable-cong]: sets M = sets borel
begin

lemma finite-borel-measure-M: finite-borel-measure M
  by standard auto

sublocale finite-borel-measure M
  by (rule finite-borel-measure-M)

lemma space-eq-univ [simp]: space M = UNIV
  using events-eq-borel[THEN sets-eq-imp-space-eq] by simp

lemma cdf-bounded-prob:  $\bigwedge x. \text{cdf } M \ x \leq 1$ 
  by (subst prob-space [symmetric], rule cdf-bounded)

lemma cdf-lim-infty-prob:  $(\lambda i. \text{cdf } M \ (\text{real } i)) \longrightarrow 1$ 
  by (subst prob-space [symmetric], rule cdf-lim-infty)

lemma cdf-lim-at-top-prob:  $(\text{cdf } M \longrightarrow 1) \text{ at-top}$ 
  by (subst prob-space [symmetric], rule cdf-lim-at-top)

lemma measurable-finite-borel [simp]:
  f  $\in$  borel-measurable borel  $\implies$  f  $\in$  borel-measurable M
  by (rule borel-measurable-subalgebra[where N=borel]) auto

end

lemma (in prob-space) real-distribution-distr [intro, simp]:
  random-variable borel X  $\implies$  real-distribution (distr M borel X)
  unfolding real-distribution-def real-distribution-axioms-def by (auto intro!: prob-space-distr)

```

2.2 Uniqueness

lemma (in *finite-borel-measure*) *emeasure-Ioc*:

assumes $a \leq b$ shows $\text{emeasure } M \{a <.. b\} = \text{cdf } M b - \text{cdf } M a$

proof –

have $\{a <.. b\} = \{..b\} - \{..a\}$

by *auto*

moreover have $\{..x\} \in \text{sets } M$ for x

using *atMost-borel[of x] M-is-borel* by *auto*

moreover note $\langle a \leq b \rangle$

ultimately show *?thesis*

by (*simp add: emeasure-eq-measure finite-measure-Diff cdf-def*)

qed

lemma *cdf-unique'*:

fixes $M1 M2$

assumes *finite-borel-measure* $M1$ and *finite-borel-measure* $M2$

assumes $\text{cdf } M1 = \text{cdf } M2$

shows $M1 = M2$

proof (*rule measure-eqI-generator-eq[where $\Omega = \text{UNIV}$]*)

fix X assume $X \in \text{range } (\lambda(a, b). \{a <.. b :: \text{real}\})$

then obtain $a b$ where $Xeq: X = \{a <.. b\}$ by *auto*

then show $\text{emeasure } M1 X = \text{emeasure } M2 X$

by (*cases a ≤ b*)

(*simp-all add: assms(1,2)[THEN finite-borel-measure.emeasure-Ioc] assms(3)*)

next

show $(\bigcup i. \{- \text{real } (i :: \text{nat}) <.. \text{real } i\}) = \text{UNIV}$

by (*rule UN-Ioc-eq-UNIV*)

qed (*auto simp: finite-borel-measure.emeasure-Ioc[OF assms(1)]*)

assms(1,2)[THEN finite-borel-measure.M-is-borel] borel-sigma-sets-Ioc

Int-stable-def)

lemma *cdf-unique*:

real-distribution $M1 \implies \text{real-distribution } M2 \implies \text{cdf } M1 = \text{cdf } M2 \implies M1 = M2$

using *cdf-unique'[of M1 M2]* by (*simp add: real-distribution.finite-borel-measure-M*)

lemma

fixes $F :: \text{real} \Rightarrow \text{real}$

assumes *nondecF* : $\bigwedge x y. x \leq y \implies F x \leq F y$

and *right-cont-F* : $\bigwedge a. \text{continuous } (\text{at-right } a) F$

and *lim-F-at-bot* : $(F \longrightarrow 0) \text{ at-bot}$

and *lim-F-at-top* : $(F \longrightarrow m) \text{ at-top}$

and $m: 0 \leq m$

shows *interval-measure-UNIV*: $\text{emeasure } (\text{interval-measure } F) \text{ UNIV} = m$

and *finite-borel-measure-interval-measure*: *finite-borel-measure* (*interval-measure* F)

proof –

let $?F = \text{interval-measure } F$

{ have $\text{ennreal } (m - 0) = (\text{SUP } i. \text{ennreal } (F (\text{real } i) - F (- \text{real } i)))$


```

    by (intro LIMSEQ-unique[OF - LIMSEQ-SUP] tendsto-ennrealI tendsto-intros
        lim-F-at-bot[THEN filterlim-compose] lim-F-at-top[THEN filter-
lim-compose]
        lim-F-at-bot[THEN filterlim-compose] filterlim-real-sequentially
        filterlim-uminus-at-top[THEN iffD1])
    (auto simp: incseq-def nondecF intro!: diff-mono)
  also have ... = (SUP i. emeasure ?F {− real i <.. real i})
    by (subst emeasure-interval-measure-Ioc) (simp-all add: nondecF right-cont-F)
  also have ... = emeasure ?F (⋃ i::nat. {− real i <.. real i})
    by (rule SUP-emeasure-incseq) (auto simp: incseq-def)
  also have (⋃ i. {− real (i::nat) <.. real i}) = space ?F
    by (simp add: UN-Ioc-eq-UNIV)
  finally have emeasure ?F (space ?F) = m
    by simp }
note * = this
then show emeasure (interval-measure F) UNIV = m
  by simp

interpret finite-measure ?F
proof
  show emeasure ?F (space ?F) ≠ ∞
    using * by simp
qed
show finite-borel-measure (interval-measure F)
  proof qed simp-all
qed

lemma real-distribution-interval-measure:
  fixes F :: real ⇒ real
  assumes nondecF : ⋀ x y. x ≤ y ⇒ F x ≤ F y and
    right-cont-F : ⋀ a. continuous (at-right a) F and
    lim-F-at-bot : (F ⟶ 0) at-bot and
    lim-F-at-top : (F ⟶ 1) at-top
  shows real-distribution (interval-measure F)
proof −
  let ?F = interval-measure F
  interpret prob-space ?F
  proof qed (use interval-measure-UNIV[OF assms] in simp)
  show ?thesis
    proof qed simp-all
qed

lemma
  fixes F :: real ⇒ real
  assumes nondecF : ⋀ x y. x ≤ y ⇒ F x ≤ F y and
    right-cont-F : ⋀ a. continuous (at-right a) F and
    lim-F-at-bot : (F ⟶ 0) at-bot
  shows emeasure-interval-measure-Iic: emeasure (interval-measure F) {.. x} = F
x

```

```

and measure-interval-measure-Iic: measure (interval-measure F) {..x} = F x
unfolding cdf-def
proof –
  have F-nonneg[simp]:  $0 \leq F\ y$  for y
  using lim-F-at-bot by (rule tendsto-upperbound) (auto simp: eventually-at-bot-linorder
nondecF intro!: exI[of - y])

  have emeasure (interval-measure F) ( $\bigcup i::nat. \{-real\ i <.. x\}$ ) = F x – ennreal
  0
  proof (intro LIMSEQ-unique[OF Lim-emeasure-incseq])
    have ( $\lambda i. F\ x - F\ (-real\ i)$ )  $\longrightarrow F\ x - 0$ 
    by (intro tendsto-intros lim-F-at-bot[THEN filterlim-compose] filterlim-real-sequentially
filterlim-uminus-at-top[THEN iffD1])
    from tendsto-ennrealI[OF this]
    show ( $\lambda i. emeasure\ (interval-measure\ F)\ \{-real\ i <.. x\}$ )  $\longrightarrow F\ x - ennreal$ 
  0

    apply (rule filterlim-cong[THEN iffD1, rotated 3])
    apply simp
    apply simp
    apply (rule eventually-sequentiallyI[where c=nat (ceiling (- x))])
    apply (simp add: emeasure-interval-measure-Ioc right-cont-F nondecF)
    done
  qed (auto simp: incseq-def)
  also have ( $\bigcup i::nat. \{-real\ i <.. x\}$ ) = {..x}
    by auto (metis minus-minus neg-less-iff-less reals-Archimedean2)
  finally show emeasure (interval-measure F) {..x} = F x
    by simp
  then show measure (interval-measure F) {..x} = F x
    by (simp add: measure-def)
qed

lemma cdf-interval-measure:
  ( $\bigwedge x\ y. x \leq y \implies F\ x \leq F\ y$ )  $\implies$  ( $\bigwedge a. continuous\ (at-right\ a)\ F$ )  $\implies$  (F  $\longrightarrow$ 
  0) at-bot  $\implies cdf\ (interval-measure\ F) = F$ 
  by (simp add: cdf-def fun-eq-iff measure-interval-measure-Iic)

end

```

3 Weak Convergence of Functions and Distributions

Properties of weak convergence of functions and measures, including the portmanteau theorem.

```

theory Weak-Convergence
  imports Distribution-Functions
begin

```

4 Weak Convergence of Functions

definition

$weak_conv :: (nat \Rightarrow (real \Rightarrow real)) \Rightarrow (real \Rightarrow real) \Rightarrow bool$

where

$weak_conv\ F\text{-seq}\ F \equiv \forall x. isCont\ F\ x \longrightarrow (\lambda n. F\text{-seq}\ n\ x) \longrightarrow F\ x$

5 Weak Convergence of Distributions

definition

$weak_conv_m :: (nat \Rightarrow real\ measure) \Rightarrow real\ measure \Rightarrow bool$

where

$weak_conv_m\ M\text{-seq}\ M \equiv weak_conv\ (\lambda n. cdf\ (M\text{-seq}\ n))\ (cdf\ M)$

6 Skorohod’s theorem

locale *right-continuous-mono* =

fixes $f :: real \Rightarrow real$ **and** $a\ b :: real$

assumes *cont*: $\bigwedge x. continuous\ (at_right\ x)\ f$

assumes *mono*: *mono* f

assumes *bot*: $(f \longrightarrow a)\ at_bot$

assumes *top*: $(f \longrightarrow b)\ at_top$

begin

abbreviation $I :: real \Rightarrow real$ **where**

$I\ \omega \equiv Inf\ \{x. \omega \leq f\ x\}$

lemma *pseudoinverse*: **assumes** $a < \omega < b$ **shows** $\omega \leq f\ x \longleftrightarrow I\ \omega \leq x$

proof

let $?F = \{x. \omega \leq f\ x\}$

obtain y **where** $f\ y < \omega$

by (*metis eventually-happens' trivial-limit-at-bot-linorder order-tendstoD(2) bot*
 $\langle a < \omega \rangle$)

with *mono* **have** *bdd*: *bdd-below* $?F$

by (*auto intro!*: *bdd-belowI[of - y]* *elim*: *mono-invE[OF - less-le-trans]*)

have *ne*: $?F \neq \{\}$

using *order-tendstoD(1)[OF top $\langle \omega < b \rangle$]*

by (*auto dest!*: *eventually-happens'[OF trivial-limit-at-top-linorder]* *intro*: *less-imp-le*)

show $\omega \leq f\ x \implies I\ \omega \leq x$

by (*auto intro!*: *cInf-lower bdd*)

{ assume $*$: $I\ \omega \leq x$

have $\omega \leq (INF\ s \in \{x. \omega \leq f\ x\}. f\ s)$

by (*rule cINF-greatest[OF ne]*) *auto*

also have $\dots = f\ (I\ \omega)$

using *continuous-at-Inf-mono[OF mono cont ne bdd]* **..**

```

    also have ... ≤ f x
    using * by (rule monoD[OF ‹mono f›])
    finally show ω ≤ f x . }
qed

```

```

lemma pseudoinverse': ∀ ω ∈ {a <..< b}. ∀ x. ω ≤ f x ⟷ I ω ≤ x
  by (intro ballI allI impI pseudoinverse) auto

```

```

lemma mono-I: mono-on {a <..< b} I
  unfolding mono-on-def by (metis order.trans order.refl pseudoinverse')

```

```

end

```

```

locale cdf-distribution = real-distribution
begin

```

```

  abbreviation C ≡ cdf M

```

```

  sublocale right-continuous-mono C 0 1
    by standard
    (auto intro: cdf-nondecreasing cdf-is-right-cont cdf-lim-at-top-prob cdf-lim-at-bot
    monoI)

```

```

lemma measurable-C[measurable]: C ∈ borel-measurable borel
  by (intro borel-measurable-mono mono)

```

```

lemma measurable-CI[measurable]: I ∈ borel-measurable (restrict-space borel {0 <..< 1})
  by (intro borel-measurable-mono-on-fnc mono-I)

```

```

lemma emeasure-distr-I: emeasure (distr (restrict-space lborel {0 <..< 1::real}) borel
I) UNIV = 1
  by (simp add: emeasure-distr space-restrict-space emeasure-restrict-space )

```

```

lemma distr-I-eq-M: distr (restrict-space lborel {0 <..< 1::real}) borel I = M (is
?I = -)

```

```

proof (intro cdf-unique ext)
  let ?Ω = restrict-space lborel {0 <..< 1}::real measure
  interpret Ω: prob-space ?Ω
  by (auto simp add: emeasure-restrict-space space-restrict-space intro!: prob-spaceI)
  show real-distribution ?I
    by auto

```

```

fix x
have cdf ?I x = measure lborel {ω ∈ {0 <..< 1}. ω ≤ C x}
  by (subst cdf-def)
  (auto simp: pseudoinverse[symmetric] measure-distr space-restrict-space mea-
sure-restrict-space
    intro!: arg-cong2[where f=measure])
also have ... = measure lborel {0 <..< C x}

```

```

    using cdf-bounded-prob[of x] AE-lborel-singleton[of C x]
    by (auto intro!: arg-cong[where f=enn2real] emeasure-eq-AE simp: measure-def)
    also have ... = C x
    by (simp add: cdf-nonneg)
    finally show cdf (distr ?Ω borel I) x = C x .
qed standard

```

end

context

```

  fixes μ :: nat ⇒ real measure
  and M :: real measure
  assumes μ: ∧n. real-distribution (μ n)
  assumes M: real-distribution M
  assumes μ-to-M: weak-conv-m μ M
begin

```

theorem Skorohod:

```

  ∃ (Ω :: real measure) (Y-seq :: nat ⇒ real ⇒ real) (Y :: real ⇒ real).
    prob-space Ω ∧
    (∀ n. Y-seq n ∈ measurable Ω borel) ∧
    (∀ n. distr Ω borel (Y-seq n) = μ n) ∧
    Y ∈ measurable Ω lborel ∧
    distr Ω borel Y = M ∧
    (∀ x ∈ space Ω. (λn. Y-seq n x) ⟶ Y x)

```

proof –

```

  interpret μ: cdf-distribution μ n for n
  unfolding cdf-distribution-def by (rule μ)
  interpret M: cdf-distribution M
  unfolding cdf-distribution-def by (rule M)

```

```

  have conv: measure M {x} = 0 ⟹ (λn. μ.C n x) ⟶ M.C x for x
  using μ-to-M M.isCont-cdf by (auto simp: weak-conv-m-def weak-conv-def)

```

```

  let ?Ω = restrict-space lborel {0<..<1} :: real measure
  have prob-space ?Ω
  by (auto simp: space-restrict-space emeasure-restrict-space intro!: prob-spaceI)
  interpret Ω: prob-space ?Ω
  by fact

```

```

  have Y-distr: distr ?Ω borel M.I = M
  by (rule M.distr-I-eq-M)

```

```

  have Y-cts-cnvt: (λn. μ.I n ω) ⟶ M.I ω
  if ω: ω ∈ {0<..<1} isCont M.I ω for ω :: real

```

```

  proof (intro limsup-le-liminf-real)
    show liminf (λn. μ.I n ω) ≥ M.I ω
    unfolding le-Liminf-iff

```

```

proof safe
  fix  $B :: \text{ereal}$  assume  $B: B < M.I \ \omega$ 
  then show  $\forall_F n$  in sequentially.  $B < \mu.I \ n \ \omega$ 
  proof (cases  $B$ )
    case (real  $r$ )
      with  $B$  have  $r: r < M.I \ \omega$ 
      by simp
      then obtain  $x$  where  $x: r < x < M.I \ \omega$  measure  $M \ \{x\} = 0$ 
      using open-minus-countable[OF  $M.\text{countable-support}$ , of  $\{r < .. < M.I \ \omega\}$ ]
by auto
  then have  $Fx\text{-less}: M.C \ x < \omega$ 
  using  $M.\text{pseudoinverse}' \ \omega$  not-less by blast

  have  $\forall_F n$  in sequentially.  $\mu.C \ n \ x < \omega$ 
  using order-tendstoD(2)[OF conv[OF  $x(3)$ ]  $Fx\text{-less}$ ] .
  then have  $\forall_F n$  in sequentially.  $x < \mu.I \ n \ \omega$ 
  by eventually-elim (insert  $\omega$   $\mu.\text{pseudoinverse}$ [symmetric], simp add:
not-le[symmetric])
  then show ?thesis
  by eventually-elim (insert  $x(1)$ , simp add: real)
qed auto
qed

have  $*$ : limsup  $(\lambda n. \mu.I \ n \ \omega) \leq M.I \ \omega'$ 
if  $\omega': 0 < \omega' \ \omega' < 1 \ \omega < \omega'$  for  $\omega' :: \text{real}$ 
proof (rule dense-ge-bounded)
  fix  $B'$  assume ereal  $(M.I \ \omega') < B' \ B' < \text{ereal} \ (M.I \ \omega' + 1)$ 
  then obtain  $B$  where  $M.I \ \omega' < B$  and [simp]:  $B' = \text{ereal} \ B$ 
  by (cases  $B'$ ) auto
  then obtain  $y$  where  $y: M.I \ \omega' < y < B$  measure  $M \ \{y\} = 0$ 
  using open-minus-countable[OF  $M.\text{countable-support}$ , of  $\{M.I \ \omega' < .. < B\}$ ]
by auto
  then have  $\omega' \leq M.C \ (M.I \ \omega')$ 
  using  $M.\text{pseudoinverse}' \ \omega'$  by (metis greaterThanLessThan-iff order-refl)
  also have  $\dots \leq M.C \ y$ 
  using  $M.\text{mono} \ y$  unfolding mono-def by auto
  finally have  $Fy\text{-gt}: \omega < M.C \ y$ 
  using  $\omega'(3)$  by simp

  have  $\forall_F n$  in sequentially.  $\omega \leq \mu.C \ n \ y$ 
  using order-tendstoD(1)[OF conv[OF  $y(3)$ ]  $Fy\text{-gt}$ ] by eventually-elim (rule
less-imp-le)
  then have 2:  $\forall_F n$  in sequentially.  $\mu.I \ n \ \omega \leq \text{ereal} \ y$ 
  by simp (subst  $\mu.\text{pseudoinverse}'$ [rule-format, OF  $\omega(1)$ , symmetric])
  then show limsup  $(\lambda n. \mu.I \ n \ \omega) \leq B'$ 
  using  $\langle y < B \rangle$ 
  by (intro Limsup-bounded[rotated]) (auto intro: le-less-trans elim: eventu-
ally-mono)
qed simp

```

```

have **: ( $M.I \longrightarrow \text{ereal } (M.I \ \omega)$ ) (at-right  $\omega$ )
  using  $\omega(2)$  by (auto intro: tendsto-within-subset simp: continuous-at)
show  $\limsup (\lambda n. \mu.I \ n \ \omega) \leq M.I \ \omega$ 
  using  $\omega$ 
  by (intro tendsto-lowerbound[OF **])
    (auto intro!: exI[of - 1] * simp: eventually-at-right[of - 1])
qed

let  $?D = \{\omega \in \{0 < .. < 1\}. \neg \text{isCont } M.I \ \omega\}$ 
have  $D\text{-countable: countable } ?D$ 
  using mono-on-ctble-discont[OF M.mono-I] by (simp add: at-within-open[of -
 $\{0 < .. < 1\}$  cong: conj-cong])
hence  $D: \text{emeasure } ?\Omega \ ?D = 0$ 
  using emeasure-lborel-countable[OF D-countable]
  by (subst emeasure-restrict-space) auto

define  $Y'$  where  $Y' \ \omega = (\text{if } \omega \in ?D \text{ then } 0 \text{ else } M.I \ \omega)$  for  $\omega$ 
have  $Y'\text{-AE: AE } \omega \text{ in } ?\Omega. Y' \ \omega = M.I \ \omega$ 
  by (rule AE-I [OF - D]) (auto simp: space-restrict-space sets-restrict-space-iff
 $Y'\text{-def}$ )

define  $Y\text{-seq'}$  where  $Y\text{-seq'} \ n \ \omega = (\text{if } \omega \in ?D \text{ then } 0 \text{ else } \mu.I \ n \ \omega)$  for  $n \ \omega$ 
have  $Y\text{-seq'-AE: } \bigwedge n. \text{AE } \omega \text{ in } ?\Omega. Y\text{-seq'} \ n \ \omega = \mu.I \ n \ \omega$ 
  by (rule AE-I [OF - D]) (auto simp: space-restrict-space sets-restrict-space-iff
 $Y\text{-seq'-def}$ )

have  $Y'\text{-cnv: } \forall \omega \in \{0 < .. < 1\}. (\lambda n. Y\text{-seq'} \ n \ \omega) \longrightarrow Y' \ \omega$ 
  by (auto simp: Y'-def Y-seq'-def Y-cts-cnvc)

have [simp]:  $Y\text{-seq'} \ n \in \text{borel-measurable } ?\Omega$  for  $n$ 
  by (rule measurable-discrete-difference[of  $\mu.I \ n$  - - ?D])
    (insert  $\mu.\text{measurable-CI}[of \ n] \ D\text{-countable}$ , auto simp: sets-restrict-space
 $Y\text{-seq'-def}$ )
moreover have  $\text{distr } ?\Omega \ \text{borel } (Y\text{-seq'} \ n) = \mu \ n$  for  $n$ 
  using  $\mu.\text{distr-I-eq-M} \ [of \ n] \ Y\text{-seq'-AE} \ [of \ n]$ 
  by (subst distr-cong-AE[where  $f = Y\text{-seq'} \ n$  and  $g = \mu.I \ n$ ], auto)
moreover have [simp]:  $Y' \in \text{borel-measurable } ?\Omega$ 
  by (rule measurable-discrete-difference[of  $M.I$  - - ?D])
    (insert  $M.\text{measurable-CI} \ D\text{-countable}$ , auto simp: sets-restrict-space  $Y'\text{-def}$ )
moreover have  $\text{distr } ?\Omega \ \text{borel } Y' = M$ 
  using  $M.\text{distr-I-eq-M} \ Y'\text{-AE}$ 
  by (subst distr-cong-AE[where  $f = Y'$  and  $g = M.I$ ], auto)
ultimately have  $\text{prob-space } ?\Omega \wedge (\forall n. Y\text{-seq'} \ n \in \text{borel-measurable } ?\Omega) \wedge$ 
  ( $\forall n. \text{distr } ?\Omega \ \text{borel } (Y\text{-seq'} \ n) = \mu \ n$ )  $\wedge Y' \in \text{measurable } ?\Omega \ \text{lborel} \wedge \text{distr } ?\Omega$ 
 $\text{borel } Y' = M \wedge$ 
  ( $\forall x \in \text{space } ?\Omega. (\lambda n. Y\text{-seq'} \ n \ x) \longrightarrow Y' \ x$ )
  using  $Y'\text{-cnv} \ \langle \text{prob-space } ?\Omega \rangle$  by (auto simp: space-restrict-space)
thus  $?thesis$  by metis

```

qed

The Portmanteau theorem, that is, the equivalence of various definitions of weak convergence.

theorem *weak-conv-imp-bdd-ae-continuous-conv:*

fixes

$f :: \text{real} \Rightarrow 'a::\{\text{banach, second-countable-topology}\}$

assumes

$\text{discont-null}: M (\{x. \neg \text{isCont } f \ x\}) = 0$ **and**

$f\text{-bdd}: \bigwedge x. \text{norm } (f \ x) \leq B$ **and**

$[\text{measurable}]: f \in \text{borel-measurable borel}$

shows

$(\lambda n. \text{integral}^L (\mu \ n) f) \longrightarrow \text{integral}^L M f$

proof –

have $0 \leq B$

using *norm-ge-zero f-bdd by (rule order-trans)*

note *Skorohod*

then obtain *Omega Y-seq Y where*

ps-Omega [simp]: prob-space Omega and

Y-seq-measurable [measurable]: $\bigwedge n. Y\text{-seq } n \in \text{borel-measurable Omega}$ and

distr-Y-seq: $\bigwedge n. \text{distr Omega borel } (Y\text{-seq } n) = \mu \ n$ and

Y-measurable [measurable]: $Y \in \text{borel-measurable Omega}$ and

distr-Y: $\text{distr Omega borel } Y = M$ and

YnY: $\bigwedge x :: \text{real}. x \in \text{space Omega} \implies (\lambda n. Y\text{-seq } n \ x) \longrightarrow Y \ x$ by force

interpret *prob-space Omega by fact*

have $*$: *emeasure Omega $(Y - \{x. \neg \text{isCont } f \ x\} \cap \text{space Omega}) = 0$*

by *(subst emeasure-distr [symmetric, where N=borel]) (auto simp: distr-Y discont-null)*

have $*$: *AE x in Omega. $(\lambda n. f (Y\text{-seq } n \ x)) \longrightarrow f (Y \ x)$*

by *(rule AE-I [OF - *]) (auto intro: isCont-tendsto-compose YnY)*

show *?thesis*

by *(auto intro!: integral-dominated-convergence[where w= $\lambda x. B$]*

*simp: f-bdd * integral-distr distr-Y-seq [symmetric] distr-Y [symmetric])*

qed

theorem *weak-conv-imp-integral-bdd-continuous-conv:*

fixes $f :: \text{real} \Rightarrow 'a::\{\text{banach, second-countable-topology}\}$

assumes

$\bigwedge x. \text{isCont } f \ x$ **and**

$\bigwedge x. \text{norm } (f \ x) \leq B$

shows

$(\lambda n. \text{integral}^L (\mu \ n) f) \longrightarrow \text{integral}^L M f$

using *assms*

by *(intro weak-conv-imp-bdd-ae-continuous-conv)*

(auto intro!: borel-measurable-continuous-onI continuous-at-imp-continuous-on)

theorem *weak-conv-imp-continuity-set-conv:*

fixes $f :: \text{real} \Rightarrow \text{real}$

assumes $[\text{measurable}]: A \in \text{sets borel}$ **and** $M (\text{frontier } A) = 0$

shows $(\lambda n. \text{measure } (\mu \ n) \ A) \longrightarrow \text{measure } M \ A$
proof –
interpret M : *real-distribution* M **by fact**
interpret μ : *real-distribution* $\mu \ n$ **for** n **by fact**

have $(\lambda n. (\int x. \text{indicator } A \ x \ \partial \mu \ n) :: \text{real}) \longrightarrow (\int x. \text{indicator } A \ x \ \partial M)$
by $(\text{intro weak-conv-imp-bdd-ae-continuous-conv}[\text{where } B=1])$
 $(\text{auto intro: assms simp: isCont-indicator})$
then show *?thesis*
by *simp*
qed

end

definition

$\text{cts-step} :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$
where
 $\text{cts-step } a \ b \ x \equiv \text{if } x \leq a \text{ then } 1 \text{ else if } x \geq b \text{ then } 0 \text{ else } (b - x) / (b - a)$

lemma *cts-step-uniformly-continuous*:

assumes $[\text{arith}]$: $a < b$
shows *uniformly-continuous-on UNIV* $(\text{cts-step } a \ b)$
unfolding *uniformly-continuous-on-def*

proof *clarsimp*

fix $e :: \text{real}$ **assume** $[\text{arith}]$: $0 < e$
let $?d = \min (e * (b - a)) (b - a)$
have $?d > 0$
by $(\text{auto simp add: field-simps})$
moreover have $\text{dist } x' \ x < ?d \implies \text{dist } (\text{cts-step } a \ b \ x') (\text{cts-step } a \ b \ x) < e$ **for**
 $x \ x'$
by $(\text{auto simp: dist-real-def divide-simps cts-step-def})$
ultimately show $\exists d > 0. \forall x \ x'. \text{dist } x' \ x < d \longrightarrow \text{dist } (\text{cts-step } a \ b \ x') (\text{cts-step } a \ b \ x) < e$
by *blast*
qed

lemma $(\text{in } \text{real-distribution})$ *integrable-cts-step*: $a < b \implies \text{integrable } M \ (\text{cts-step } a \ b)$

by $(\text{rule integrable-const-bound } [\text{of } - \ 1]) (\text{auto simp: cts-step-def}[\text{abs-def}])$

lemma $(\text{in } \text{real-distribution})$ *cdf-cts-step*:

assumes $[\text{arith}]$: $x < y$
shows $\text{cdf } M \ x \leq \text{integral}^L \ M \ (\text{cts-step } x \ y)$ **and** $\text{integral}^L \ M \ (\text{cts-step } x \ y) \leq \text{cdf } M \ y$

proof –

have $\text{cdf } M \ x = \text{integral}^L \ M \ (\text{indicator } \{..x\})$
by $(\text{simp add: cdf-def})$
also have $\dots \leq \text{expectation } (\text{cts-step } x \ y)$
by $(\text{intro integral-mono integrable-cts-step})$

```

      (auto simp: cts-step-def less-top[symmetric] split: split-indicator)
    finally show  $\text{cdf } M \ x \leq \text{expectation } (\text{cts-step } x \ y)$  .
next
  have  $\text{expectation } (\text{cts-step } x \ y) \leq \text{integral}^L M \ (\text{indicator } \{..y\})$ 
    by (intro integral-mono integrable-cts-step)
      (auto simp: cts-step-def less-top[symmetric] split: split-indicator)
  also have  $\dots = \text{cdf } M \ y$ 
    by (simp add: cdf-def)
  finally show  $\text{expectation } (\text{cts-step } x \ y) \leq \text{cdf } M \ y$  .
qed

context
  fixes  $M\text{-seq} :: \text{nat} \Rightarrow \text{real measure}$ 
  and  $M :: \text{real measure}$ 
  assumes  $\text{distr-}M\text{-seq} \ [\text{simp}]: \bigwedge n. \text{real-distribution } (M\text{-seq } n)$ 
  assumes  $\text{distr-}M \ [\text{simp}]: \text{real-distribution } M$ 
begin

theorem continuity-set-conv-imp-weak-conv:
  fixes  $f :: \text{real} \Rightarrow \text{real}$ 
  assumes *:  $\bigwedge A. A \in \text{sets borel} \implies M \ (\text{frontier } A) = 0 \implies (\lambda n. (\text{measure } (M\text{-seq } n) \ A)) \longrightarrow \text{measure } M \ A$ 
  shows  $\text{weak-conv-m } M\text{-seq } M$ 
proof -
  interpret  $\text{real-distribution } M$  by simp
  show ?thesis
    by (auto intro!: * simp: frontier-real-atMost isCont-cdf emeasure-eq-measure
        weak-conv-m-def weak-conv-def cdf-def2)
qed

theorem integral-cts-step-conv-imp-weak-conv:
  assumes  $\text{integral-conv}: \bigwedge x \ y. x < y \implies (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } x \ y)) \longrightarrow \text{integral}^L M (\text{cts-step } x \ y)$ 
  shows  $\text{weak-conv-m } M\text{-seq } M$ 
  unfolding  $\text{weak-conv-m-def weak-conv-def}$ 
proof (clarsimp)
  interpret  $\text{real-distribution } M$  by (rule distr-M)
  fix  $x$  assume  $\text{isCont } (\text{cdf } M) \ x$ 
  hence  $\text{left-cont: continuous } (\text{at-left } x) (\text{cdf } M)$ 
    unfolding  $\text{continuous-at-split ..}$ 
  { fix  $y :: \text{real}$  assume  $[\text{arith}]: x < y$ 
    have  $\text{limsup } (\lambda n. \text{cdf } (M\text{-seq } n) \ x) \leq \text{limsup } (\lambda n. \text{integral}^L (M\text{-seq } n) (\text{cts-step } x \ y))$ 
      by (auto intro!: Limsup-mono always-eventually real-distribution.cdf-cts-step)
    also have  $\dots = \text{integral}^L M (\text{cts-step } x \ y)$ 
      by (intro lim-imp-Limsup) (auto intro: integral-conv)
    also have  $\dots \leq \text{cdf } M \ y$ 
      by (simp add: cdf-cts-step)
    finally have  $\text{limsup } (\lambda n. \text{cdf } (M\text{-seq } n) \ x) \leq \text{cdf } M \ y$  .
  }

```

```

} note * = this
{ fix y :: real assume [arith]: x > y
  have cdf M y ≤ ereal (integralL M (cts-step y x))
    by (simp add: cdf-cts-step)
  also have ... = liminf (λn. integralL (M-seq n) (cts-step y x))
    by (intro lim-imp-Liminf[symmetric]) (auto intro: integral-conv)
  also have ... ≤ liminf (λn. cdf (M-seq n) x)
    by (auto intro!: Liminf-mono always-eventually real-distribution.cdf-cts-step)
  finally have liminf (λn. cdf (M-seq n) x) ≥ cdf M y .
} note ** = this

have limsup (λn. cdf (M-seq n) x) ≤ cdf M x
proof (rule tendsto-lowerbound)
  show ∀F i in at-right x. limsup (λxa. ereal (cdf (M-seq xa) x)) ≤ ereal (cdf M
i)
    by (subst eventually-at-right[of - x + 1]) (auto simp: * intro: exI [of - x + 1])
qed (insert cdf-is-right-cont, auto simp: continuous-within)
moreover have cdf M x ≤ liminf (λn. cdf (M-seq n) x)
proof (rule tendsto-upperbound)
  show ∀F i in at-left x. ereal (cdf M i) ≤ liminf (λxa. ereal (cdf (M-seq xa) x))
    by (subst eventually-at-left[of x - 1]) (auto simp: ** intro: exI [of - x - 1])
qed (insert left-cont, auto simp: continuous-within)
ultimately show (λn. cdf (M-seq n) x) ⟶ cdf M x
  by (elim limsup-le-liminf-real)
qed

theorem integral-bdd-continuous-conv-imp-weak-conv:
  assumes
    ∧f. (∧x. isCont f x) ⟹ (∧x. abs (f x) ≤ 1) ⟹ (λn. integralL (M-seq n)
f::real) ⟶ integralL M f
  shows
    weak-conv-m M-seq M
  apply (rule integral-cts-step-conv-imp-weak-conv [OF assms])
  apply (rule continuous-on-interior)
  apply (rule uniformly-continuous-imp-continuous)
  apply (rule cts-step-uniformly-continuous)
  apply (auto simp: cts-step-def)
done

end

end

```

7 The Giry monad

```

theory Giry-Monad
  imports Probability-Measure HOL-Library.Monad-Syntax
begin

```

7.1 Sub-probability spaces

locale *subprob-space* = *finite-measure* +
assumes *emeasure-space-le-1*: *emeasure* *M* (*space* *M*) ≤ 1
assumes *subprob-not-empty*: *space* *M* $\neq \{\}$

lemma *subprob-spaceI*[*Pure.intro!*]:
assumes *: *emeasure* *M* (*space* *M*) ≤ 1
assumes *space* *M* $\neq \{\}$
shows *subprob-space* *M*

proof –

interpret *finite-measure* *M*

proof

show *emeasure* *M* (*space* *M*) $\neq \infty$ **using** * **by** (*auto simp: top-unique*)

qed

show *subprob-space* *M* **by** *standard fact*+

qed

lemma (**in** *subprob-space*) *emeasure-subprob-space-less-top*: *emeasure* *M* *A* $\neq \text{top}$
by *simp*

lemma *prob-space-imp-subprob-space*:
prob-space *M* \implies *subprob-space* *M*
by (*rule subprob-spaceI*) (*simp-all add: prob-space.emeasure-space-1 prob-space.not-empty*)

lemma *subprob-space-imp-sigma-finite*: *subprob-space* *M* \implies *sigma-finite-measure* *M*
unfolding *subprob-space-def* *finite-measure-def* **by** *simp*

sublocale *prob-space* \subseteq *subprob-space*
by (*rule subprob-spaceI*) (*simp-all add: emeasure-space-1 not-empty*)

lemma *subprob-space-sigma* [*simp*]: $\Omega \neq \{\} \implies$ *subprob-space* (*sigma* Ω *X*)
by(*rule subprob-spaceI*)(*simp-all add: emeasure-sigma space-measure-of-conv*)

lemma *subprob-space-null-measure*: *space* *M* $\neq \{\} \implies$ *subprob-space* (*null-measure* *M*)
by(*simp add: null-measure-def*)

lemma (**in** *subprob-space*) *subprob-space-distr*:
assumes *f*: *f* \in *measurable* *M* *M'* **and** *space* *M'* $\neq \{\}$ **shows** *subprob-space* (*distr* *M* *M'* *f*)
proof (*rule subprob-spaceI*)
have *f* – ‘*space* *M'* \cap *space* *M* = *space* *M* **using** *f* **by** (*auto dest: measurable-space*)
with *f* **show** *emeasure* (*distr* *M* *M'* *f*) (*space* (*distr* *M* *M'* *f*)) ≤ 1
by (*auto simp: emeasure-distr emeasure-space-le-1*)
show *space* (*distr* *M* *M'* *f*) $\neq \{\}$ **by** (*simp add: assms*)
qed

lemma (in *subprob-space*) *subprob-emeasure-le-1*: *emeasure M X* ≤ 1
 by (rule *order.trans*[*OF emeasure-space emeasure-space-le-1*])

lemma (in *subprob-space*) *subprob-measure-le-1*: *measure M X* ≤ 1
 using *subprob-emeasure-le-1* [*of X*] by (simp add: *emeasure-eq-measure*)

lemma (in *subprob-space*) *nn-integral-le-const*:

assumes $0 \leq c$ AE *x* in *M*. *f x* $\leq c$

shows $(\int^+ x. f x \partial M) \leq c$

proof –

have $(\int^+ x. f x \partial M) \leq (\int^+ x. c \partial M)$

by (rule *nn-integral-mono-AE*) fact

also have $\dots \leq c * \text{emeasure } M (\text{space } M)$

using $\langle 0 \leq c \rangle$ by simp

also have $\dots \leq c * 1$ using *emeasure-space-le-1* $\langle 0 \leq c \rangle$ by (rule *mult-left-mono*)

finally show ?thesis by simp

qed

lemma *emeasure-density-distr-interval*:

fixes *h* :: *real* \Rightarrow *real* and *g* :: *real* \Rightarrow *real* and *g'* :: *real* \Rightarrow *real*

assumes [simp]: $a \leq b$

assumes *Mf*[*measurable*]: *f* \in *borel-measurable borel*

assumes *Mg*[*measurable*]: *g* \in *borel-measurable borel*

assumes *Mg'*[*measurable*]: *g'* \in *borel-measurable borel*

assumes *Mh*[*measurable*]: *h* \in *borel-measurable borel*

assumes *prob*: *subprob-space* (*density lborel f*)

assumes *nonnegf*: $\bigwedge x. f x \geq 0$

assumes *derivg*: $\bigwedge x. x \in \{a..b\} \implies (g \text{ has-real-derivative } g' x) \text{ (at } x)$

assumes *contg'*: *continuous-on* $\{a..b\}$ *g'*

assumes *mono*: *strict-mono-on* $\{a..b\}$ *g* and *inv*: $\bigwedge x. h x \in \{a..b\} \implies g (h x)$

= *x*

assumes *range*: $\{a..b\} \subseteq \text{range } h$

shows *emeasure* (*distr* (*density lborel f*) *lborel h*) $\{a..b\} =$

emeasure (*density lborel* ($\lambda x. f (g x) * g' x$)) $\{a..b\}$

proof (cases $a < b$)

assume $a < b$

from *mono* have *inj*: *inj-on* *g* $\{a..b\}$ by (rule *strict-mono-on-imp-inj-on*)

from *mono* have *mono'*: *mono-on* $\{a..b\}$ *g* by (rule *strict-mono-on-imp-mono-on*)

from *mono'* *derivg* have $\bigwedge x. x \in \{a < .. < b\} \implies g' x \geq 0$

by (rule *mono-on-imp-deriv-nonneg*) auto

from *contg'* this have *derivg-nonneg*: $\bigwedge x. x \in \{a..b\} \implies g' x \geq 0$

by (rule *continuous-ge-on-Ioo*) (simp-all add: $\langle a < b \rangle$)

from *derivg* have *contg*: *continuous-on* $\{a..b\}$ *g* by (rule *has-real-derivative-imp-continuous-on*)

have *A*: $h - ' \{a..b\} = \{g a..g b\}$

proof (intro *equalityI subsetI*)

fix *x* assume *x*: $x \in h - ' \{a..b\}$

hence *g* (*h x*) $\in \{g a..g b\}$ by (auto intro: *mono-onD*[*OF mono'*])

with *inv* and *x* show $x \in \{g a..g b\}$ by simp

```

next
  fix y assume y: y ∈ {g a..g b}
  with IVT'[OF - - - contg, of y] obtain x where x ∈ {a..b} y = g x by auto
  with range and inv show y ∈ h - ' {a..b} by auto
qed

have prob': subprob-space (distr (density lborel f) lborel h)
  by (rule subprob-space.subprob-space-distr[OF prob]) (simp-all add: Mh)
have B: emeasure (distr (density lborel f) lborel h) {a..b} =
  ∫+x. f x * indicator (h - ' {a..b}) x ∂lborel
  by (subst emeasure-distr) (simp-all add: emeasure-density Mf Mh measurable-sets-borel[OF Mh])
also note A
also have emeasure (distr (density lborel f) lborel h) {a..b} ≤ 1
  by (rule subprob-space.subprob-emeasure-le-1) (rule prob')
hence emeasure (distr (density lborel f) lborel h) {a..b} ≠ ∞ by (auto simp:
top-unique)
with assms have (∫+x. f x * indicator {g a..g b} x ∂lborel) =
  (∫+x. f (g x) * g' x * indicator {a..b} x ∂lborel)
  by (intro nn-integral-substitution-aux)
  (auto simp: derivg-nonneg A B emeasure-density mult.commute ⟨a < b⟩)
also have ... = emeasure (density lborel (λx. f (g x) * g' x)) {a..b}
  by (simp add: emeasure-density)
finally show ?thesis .
next
  assume ¬a < b
  with ⟨a ≤ b⟩ have [simp]: b = a by (simp add: not-less del: ⟨a ≤ b⟩)
  from inv and range have h - ' {a} = {g a} by auto
  thus ?thesis by (simp-all add: emeasure-distr emeasure-density measurable-sets-borel[OF
Mh])
qed

locale pair-subprob-space =
  pair-sigma-finite M1 M2 + M1: subprob-space M1 + M2: subprob-space M2 for
M1 M2

sublocale pair-subprob-space ⊆ P?: subprob-space M1 ⊗M M2
proof
  from mult-le-one[OF M1.emeasure-space-le-1 - M2.emeasure-space-le-1]
  show emeasure (M1 ⊗M M2) (space (M1 ⊗M M2)) ≤ 1
  by (simp add: M2.emeasure-pair-measure-Times space-pair-measure)
  from M1.subprob-not-empty and M2.subprob-not-empty show space (M1 ⊗M
M2) ≠ {}
  by (simp add: space-pair-measure)
qed

lemma subprob-space-null-measure-iff:
  subprob-space (null-measure M) ⟷ space M ≠ {}
  by (auto intro!: subprob-spaceI dest: subprob-space.subprob-not-empty)

```

lemma *subprob-space-restrict-space*:
assumes M : *subprob-space* M
and A : $A \cap \text{space } M \in \text{sets } M$ $A \cap \text{space } M \neq \{\}$
shows *subprob-space* (*restrict-space* M A)
proof(*rule subprob-spaceI*)
have *emeasure* (*restrict-space* M A) (*space* (*restrict-space* M A)) = *emeasure* M ($A \cap \text{space } M$)
using A **by**(*simp add: emeasure-restrict-space space-restrict-space*)
also have $\dots \leq 1$ **by**(*rule subprob-space.subprob-emeasure-le-1*)(*rule* M)
finally show *emeasure* (*restrict-space* M A) (*space* (*restrict-space* M A)) ≤ 1 .
next
show *space* (*restrict-space* M A) $\neq \{\}$
using A **by**(*simp add: space-restrict-space*)
qed

definition *subprob-algebra* :: 'a *measure* \Rightarrow 'a *measure measure* **where**
subprob-algebra $K =$
 $(\text{SUP } A \in \text{sets } K. \text{vimage-algebra } \{M. \text{subprob-space } M \wedge \text{sets } M = \text{sets } K\}$
 $(\lambda M. \text{emeasure } M A) \text{ borel})$

lemma *space-subprob-algebra*: *space* (*subprob-algebra* A) = $\{M. \text{subprob-space } M \wedge \text{sets } M = \text{sets } A\}$
by (*auto simp add: subprob-algebra-def space-Sup-eq-UN*)

lemma *subprob-algebra-cong*: *sets* $M = \text{sets } N \implies \text{subprob-algebra } M = \text{subprob-algebra } N$
by (*simp add: subprob-algebra-def*)

lemma *measurable-emeasure-subprob-algebra*[*measurable*]:
 $a \in \text{sets } A \implies (\lambda M. \text{emeasure } M a) \in \text{borel-measurable } (\text{subprob-algebra } A)$
by (*auto intro!: measurable-Sup1 measurable-vimage-algebra1 simp: subprob-algebra-def*)

lemma *measurable-measure-subprob-algebra*[*measurable*]:
 $a \in \text{sets } A \implies (\lambda M. \text{measure } M a) \in \text{borel-measurable } (\text{subprob-algebra } A)$
unfolding *measure-def* **by** *measurable*

lemma *subprob-measurableD*:
assumes N : $N \in \text{measurable } M$ (*subprob-algebra* S) **and** x : $x \in \text{space } M$
shows *space* ($N x$) = *space* S
and *sets* ($N x$) = *sets* S
and *measurable* ($N x$) $K = \text{measurable } S K$
and *measurable* K ($N x$) = *measurable* $K S$
using *measurable-space*[*OF* $N x$]
by (*auto simp: space-subprob-algebra intro!: measurable-cong-sets dest: sets-eq-imp-space-eq*)

ML \langle

fun *subprob-cong* *thm* *ctxt* = (

```

let
  val thm' = Thm.transfer' ctxt thm
  val free = thm' |> Thm.concl-of |> HOLogic.dest-Trueprop |> dest-comb |> fst
|>
  dest-comb |> snd |> strip-abs-body |> head-of |> is-Free
in
  if free then ([], Measurable.add-local-cong (thm' RS @ {thm subprob-measurableD(2)}))
  ctxt)
    else ([], ctxt)
end
handle THM - => ([], ctxt) | TERM - => ([], ctxt))
>

```

```

setup <
  Context.theory-map (Measurable.add-preprocessor subprob-cong subprob-cong)
>

```

```

context
  fixes K M N assumes K: K ∈ measurable M (subprob-algebra N)
begin

```

```

lemma subprob-space-kernel: a ∈ space M ⇒ subprob-space (K a)
using measurable-space[OF K] by (simp add: space-subprob-algebra)

```

```

lemma sets-kernel: a ∈ space M ⇒ sets (K a) = sets N
using measurable-space[OF K] by (simp add: space-subprob-algebra)

```

```

lemma measurable-emeasure-kernel[measurable]:
  A ∈ sets N ⇒ (λa. emeasure (K a) A) ∈ borel-measurable M
using measurable-compose[OF K measurable-emeasure-subprob-algebra] .

```

```

end

```

```

lemma measurable-subprob-algebra:
  (⋀ a. a ∈ space M ⇒ subprob-space (K a)) ⇒
  (⋀ a. a ∈ space M ⇒ sets (K a) = sets N) ⇒
  (⋀ A. A ∈ sets N ⇒ (λa. emeasure (K a) A) ∈ borel-measurable M) ⇒
  K ∈ measurable M (subprob-algebra N)
by (auto intro!: measurable-Sup2 measurable-vimage-algebra2 simp: subprob-algebra-def)

```

```

lemma measurable-submarkov:
  K ∈ measurable M (subprob-algebra M) ⟷
  (∀ x ∈ space M. subprob-space (K x) ∧ sets (K x) = sets M) ∧
  (∀ A ∈ sets M. (λx. emeasure (K x) A) ∈ measurable M borel)

```

```

proof
  assume (∀ x ∈ space M. subprob-space (K x) ∧ sets (K x) = sets M) ∧
  (∀ A ∈ sets M. (λx. emeasure (K x) A) ∈ borel-measurable M)
  then show K ∈ measurable M (subprob-algebra M)

```



```

  by (intro measurable-subprob-algebra) auto
next
  assume  $K \in \text{measurable } M \text{ (subprob-algebra } M)$ 
  then show  $(\forall x \in \text{space } M. \text{subprob-space } (K \ x) \wedge \text{sets } (K \ x) = \text{sets } M) \wedge$ 
     $(\forall A \in \text{sets } M. (\lambda x. \text{emeasure } (K \ x) \ A) \in \text{borel-measurable } M)$ 
  by (auto dest: subprob-space-kernel sets-kernel)
qed

```

lemma *measurable-subprob-algebra-generated*:

```

  assumes eq:  $\text{sets } N = \text{sigma-sets } \Omega \ G$  and Int-stable  $G \ G \subseteq \text{Pow } \Omega$ 
  assumes subsp:  $\bigwedge a. a \in \text{space } M \implies \text{subprob-space } (K \ a)$ 
  assumes sets:  $\bigwedge a. a \in \text{space } M \implies \text{sets } (K \ a) = \text{sets } N$ 
  assumes  $\bigwedge A. A \in G \implies (\lambda a. \text{emeasure } (K \ a) \ A) \in \text{borel-measurable } M$ 
  assumes  $\Omega: (\lambda a. \text{emeasure } (K \ a) \ \Omega) \in \text{borel-measurable } M$ 
  shows  $K \in \text{measurable } M \text{ (subprob-algebra } N)$ 
proof (rule measurable-subprob-algebra)
  fix a assume  $a \in \text{space } M$  then show  $\text{subprob-space } (K \ a) \ \text{sets } (K \ a) = \text{sets } N$ 
by fact+
next
  interpret  $G: \text{sigma-algebra } \Omega \ \text{sigma-sets } \Omega \ G$ 
  using  $\langle G \subseteq \text{Pow } \Omega \rangle$  by (rule sigma-algebra-sigma-sets)
  fix A assume  $A \in \text{sets } N$  with assms(2,3) show  $(\lambda a. \text{emeasure } (K \ a) \ A) \in$ 
    borel-measurable  $M$ 
  unfolding  $\langle \text{sets } N = \text{sigma-sets } \Omega \ G \rangle$ 
proof (induction rule: sigma-sets-induct-disjoint)
  case (basic A) then show ?case by fact
next
  case empty then show ?case by simp
next
  case (compl A)
  have  $(\lambda a. \text{emeasure } (K \ a) \ (\Omega - A)) \in \text{borel-measurable } M \longleftrightarrow$ 
     $(\lambda a. \text{emeasure } (K \ a) \ \Omega - \text{emeasure } (K \ a) \ A) \in \text{borel-measurable } M$ 
  using  $G.\text{top } G.\text{sets-into-space sets eq compl subprob-space.emeasure-subprob-space-less-top[OF}$ 
    subsp]
  by (intro measurable-cong emeasure-Diff) auto
  with compl  $\Omega$  show ?case
  by simp
next
  case (union F)
  moreover have  $(\lambda a. \text{emeasure } (K \ a) \ (\bigcup i. F \ i)) \in \text{borel-measurable } M \longleftrightarrow$ 
     $(\lambda a. \sum i. \text{emeasure } (K \ a) \ (F \ i)) \in \text{borel-measurable } M$ 
  using sets union eq
  by (intro measurable-cong suminf-emeasure[symmetric]) auto
  ultimately show ?case
  by auto
qed
qed

```

lemma *space-subprob-algebra-empty-iff*:

$space (subprob-algebra N) = \{\}$ \longleftrightarrow $space N = \{\}$
proof
 have $\bigwedge x. x \in space N \implies density N (\lambda-. 0) \in space (subprob-algebra N)$
 by (auto simp: space-subprob-algebra emeasure-density intro!: subprob-spaceI)
 then show $space (subprob-algebra N) = \{\} \implies space N = \{\}$
 by auto
next
 assume $space N = \{\}$
 hence $sets N = \{\{\}\}$ by (simp add: space-empty-iff)
 moreover have $\bigwedge M. subprob-space M \implies sets M \neq \{\{\}\}$
 by (simp add: subprob-space.subprob-not-empty space-empty-iff[symmetric])
 ultimately show $space (subprob-algebra N) = \{\}$ by (auto simp: space-subprob-algebra)
qed

lemma *nn-integral-measurable-subprob-algebra[measurable]:*

assumes $f: f \in borel-measurable N$
 shows $(\lambda M. integral^N M f) \in borel-measurable (subprob-algebra N)$ (is - $\in ?B$)
 using f
proof induct
 case (cong f g)
 moreover have $(\lambda M'. \int^+ M''. f M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. \int^+ M''. g M'' \partial M') \in ?B$
 by (intro measurable-cong nn-integral-cong cong)
 (auto simp: space-subprob-algebra dest!: sets-eq-imp-space-eq)
 ultimately show ?case by simp
next
 case (set B)
 then have $(\lambda M'. \int^+ M''. indicator B M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. emeasure M' B) \in ?B$
 by (intro measurable-cong nn-integral-indicator) (simp add: space-subprob-algebra)
 with set show ?case
 by (simp add: measurable-emeasure-subprob-algebra)
next
 case (mult f c)
 then have $(\lambda M'. \int^+ M''. c * f M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. c * \int^+ M''. f M'' \partial M') \in ?B$
 by (intro measurable-cong nn-integral-cmult) (auto simp add: space-subprob-algebra)
 with mult show ?case
 by simp
next
 case (add f g)
 then have $(\lambda M'. \int^+ M''. f M'' + g M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. (\int^+ M''. f M'' \partial M') + (\int^+ M''. g M'' \partial M')) \in ?B$
 by (intro measurable-cong nn-integral-add) (auto simp add: space-subprob-algebra)
 with add show ?case
 by (simp add: ac-simps)
next
 case (seq F)
 then have $(\lambda M'. \int^+ M''. (SUP i. F i) M'' \partial M') \in ?B \longleftrightarrow (\lambda M'. SUP i.$

$(\int^+ M''. F \text{ i } M'' \partial M') \in ?B$
unfolding *SUP-apply*
by (*intro measurable-cong nn-integral-monotone-convergence-SUP*) (*auto simp*
add: space-subprob-algebra)
with *seq show ?case*
by (*simp add: ac-simps*)
qed

lemma *measurable-distr*:
assumes [*measurable*]: $f \in \text{measurable } M \ N$
shows $(\lambda M'. \text{distr } M' \ N \ f) \in \text{measurable } (\text{subprob-algebra } M) \ (\text{subprob-algebra } N)$
proof (*cases space N = {}*)
case *False*
show *?thesis*
proof (*rule measurable-subprob-algebra*)
fix *A* **assume** *A*: $A \in \text{sets } N$
then have $(\lambda M'. \text{emeasure } (\text{distr } M' \ N \ f) \ A) \in \text{borel-measurable } (\text{subprob-algebra } M)$
 \longleftrightarrow
 $(\lambda M'. \text{emeasure } M' \ (f - 'A \cap \text{space } M)) \in \text{borel-measurable } (\text{subprob-algebra } M)$
by (*intro measurable-cong*)
(auto simp: emeasure-distr space-subprob-algebra
intro!: arg-cong2[where f=emeasure] sets-eq-imp-space-eq arg-cong2[where
f=(\cap)))
also have ...
using *A* **by** (*intro measurable-emeasure-subprob-algebra simp*)
finally show $(\lambda M'. \text{emeasure } (\text{distr } M' \ N \ f) \ A) \in \text{borel-measurable } (\text{subprob-algebra } M)$.
qed (*auto intro!: subprob-space.subprob-space-distr simp: space-subprob-algebra*
False cong: measurable-cong-sets)
qed (*use assms in <auto simp: measurable-empty-iff space-subprob-algebra-empty-iff>*)

lemma *emeasure-space-subprob-algebra[measurable]*:
 $(\lambda a. \text{emeasure } a \ (\text{space } a)) \in \text{borel-measurable } (\text{subprob-algebra } N)$
proof–
have $(\lambda a. \text{emeasure } a \ (\text{space } N)) \in \text{borel-measurable } (\text{subprob-algebra } N)$ (**is** *?f*
 $\in ?M$)
by (*rule measurable-emeasure-subprob-algebra simp*)
also have *?f* $\in ?M \longleftrightarrow (\lambda a. \text{emeasure } a \ (\text{space } a)) \in ?M$
by (*rule measurable-cong*) (*auto simp: space-subprob-algebra dest: sets-eq-imp-space-eq*)
finally show *?thesis* .
qed

lemma *integrable-measurable-subprob-algebra[measurable]*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
assumes [*measurable*]: $f \in \text{borel-measurable } N$
shows *Measurable.pred* (*subprob-algebra N*) $(\lambda M. \text{integrable } M \ f)$
proof (*rule measurable-cong[THEN iffD2]*)

show $M \in \text{space}(\text{subprob-algebra } N) \implies \text{integrable } M f \iff (\int^+ x. \text{norm}(f x) \partial M) < \infty$ **for** M
by (*auto simp: space-subprob-algebra integrable-iff-bounded*)
qed *measurable*

lemma *integral-measurable-subprob-algebra[measurable]*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second-countable-topology}\}$
assumes f [*measurable*]: $f \in \text{borel-measurable } N$
shows $(\lambda M. \text{integral}^L M f) \in \text{subprob-algebra } N \rightarrow_M \text{borel}$

proof –
from *borel-measurable-implies-sequence-metric*[*OF f, of 0*]
obtain F **where** $F: \bigwedge i. \text{simple-function } N (F i)$
 $\bigwedge x. x \in \text{space } N \implies (\lambda i. F i x) \longrightarrow f x$
 $\bigwedge i x. x \in \text{space } N \implies \text{norm}(F i x) \leq 2 * \text{norm}(f x)$
unfolding *norm-conv-dist* **by** *blast*

have [*measurable*]: $F i \in N \rightarrow_M \text{count-space UNIV}$ **for** i
using $F(1)$ **by** (*rule measurable-simple-function*)

define F' **where** [*abs-def*]:
 $F' M i = (\text{if integrable } M f \text{ then } \text{integral}^L M (F i) \text{ else } 0)$ **for** $M i$

have $(\lambda M. F' M i) \in \text{subprob-algebra } N \rightarrow_M \text{borel}$ **for** i

proof (*rule measurable-cong*[*THEN iffD2*])
fix M **assume** $M \in \text{space}(\text{subprob-algebra } N)$
then have [*simp*]: $\text{sets } M = \text{sets } N$ $\text{space } M = \text{space } N$ $\text{subprob-space } M$
by (*auto simp: space-subprob-algebra intro!: sets-eq-imp-space-eq*)
interpret *subprob-space* M **by fact**
have $F' M i = (\text{if integrable } M f \text{ then } \text{Bochner-Integration.simple-bochner-integral } M (F i) \text{ else } 0)$
using $F(1)$
by (*subst simple-bochner-integrable-eq-integral*)
(auto simp: simple-bochner-integrable.simps simple-function-def F'-def)
then show $F' M i = (\text{if integrable } M f \text{ then } \sum_{y \in F i} \text{space } N. \text{measure } M \{x \in \text{space } N. F i x = y\} *_{\mathbb{R}} y \text{ else } 0)$
unfolding *simple-bochner-integral-def* **by** *simp*
qed *measurable*

moreover

have $F' M \longrightarrow \text{integral}^L M f$ **if** $M: M \in \text{space}(\text{subprob-algebra } N)$ **for** M

proof *cases*

from M **have** [*simp*]: $\text{sets } M = \text{sets } N$ $\text{space } M = \text{space } N$
by (*auto simp: space-subprob-algebra intro!: sets-eq-imp-space-eq*)
assume *integrable* $M f$ **then show** *?thesis*
unfolding *F'-def* **using** $F(1)$ [*THEN borel-measurable-simple-function*] F
by (*auto intro!: integral-dominated-convergence*[**where** $w = \lambda x. 2 * \text{norm}(f x)$]
cong: measurable-cong-sets)

qed (*auto simp: F'-def not-integrable-integral-eq*)

ultimately show *?thesis*

by (*rule borel-measurable-LIMSEQ-metric*)

qed

lemma *measurable-pair-measure*:

assumes $f: f \in \text{measurable } M \text{ (subprob-algebra } N)$
assumes $g: g \in \text{measurable } M \text{ (subprob-algebra } L)$
shows $(\lambda x. f x \otimes_M g x) \in \text{measurable } M \text{ (subprob-algebra } (N \otimes_M L))$
proof (*rule measurable-subprob-algebra*)
 { **fix** x **assume** $x \in \text{space } M$
 with $\text{measurable-space}[OF f] \text{ measurable-space}[OF g]$
 have $fx: f x \in \text{space (subprob-algebra } N)$ **and** $gx: g x \in \text{space (subprob-algebra } L)$
 by *auto*
 interpret $F: \text{subprob-space } f x$
 using fx **by** (*simp add: space-subprob-algebra*)
 interpret $G: \text{subprob-space } g x$
 using gx **by** (*simp add: space-subprob-algebra*)

 interpret $\text{pair-subprob-space } f x g x ..$
 show $\text{subprob-space } (f x \otimes_M g x)$ **by** *unfold-locales*
 show $\text{sets-eq: sets } (f x \otimes_M g x) = \text{sets } (N \otimes_M L)$
 using $fx gx$ **by** (*simp add: space-subprob-algebra*)

 have $1: \bigwedge A B. A \in \text{sets } N \implies B \in \text{sets } L \implies \text{emeasure } (f x \otimes_M g x) (A \times B) = \text{emeasure } (f x) A * \text{emeasure } (g x) B$
 using $fx gx$ **by** (*intro G.emeasure-pair-measure-Times (auto simp: space-subprob-algebra)*)
 have $\text{emeasure } (f x \otimes_M g x) (\text{space } (f x \otimes_M g x)) =$
 $\text{emeasure } (f x) (\text{space } (f x)) * \text{emeasure } (g x) (\text{space } (g x))$
 by (*subst G.emeasure-pair-measure-Times[symmetric] (simp-all add: space-pair-measure)*)
 hence $2: \bigwedge A. A \in \text{sets } (N \otimes_M L) \implies \text{emeasure } (f x \otimes_M g x) (\text{space } N \times \text{space } L - A) =$
 $\dots - \text{emeasure } (f x \otimes_M g x) A$
 using *emeasure-compl[simplified, OF - P.emeasure-finite]*
 unfolding *sets-eq*
 unfolding *sets-eq-imp-space-eq[OF sets-eq]*
 by (*simp add: space-pair-measure G.emeasure-pair-measure-Times*)
 note $1 \ 2 \ \text{sets-eq} \}$
note $\text{Times} = \text{this}(1)$ **and** $\text{Compl} = \text{this}(2)$ **and** $\text{sets-eq} = \text{this}(3)$

fix A **assume** $A: A \in \text{sets } (N \otimes_M L)$
show $(\lambda a. \text{emeasure } (f a \otimes_M g a) A) \in \text{borel-measurable } M$
 using *Int-stable-pair-measure-generator pair-measure-closed A*
 unfolding *sets-pair-measure*
proof (*induct A rule: sigma-sets-induct-disjoint*)
 case (*basic A*) **then show** ?case
 by (*auto intro!: borel-measurable-times-ennreal simp: Times cong: measurable-cong*)
 (*auto intro!: measurable-emeasure-kernel f g*)
next

```

case (compl A)
then have A:  $A \in \text{sets } (N \otimes_M L)$ 
  by (auto simp: sets-pair-measure)
have  $(\lambda x. \text{emeasure } (f \ x) \ (\text{space } (f \ x)) * \text{emeasure } (g \ x) \ (\text{space } (g \ x)) -$ 
 $\text{emeasure } (f \ x \otimes_M g \ x) \ A) \in \text{borel-measurable } M \text{ (is } ?f \in ?M)$ 
  using compl(2) f g by measurable
thus ?case by (simp add: Compl A cong: measurable-cong)
next
case (union A)
then have  $\text{range } A \subseteq \text{sets } (N \otimes_M L) \text{ disjoint-family } A$ 
  by (auto simp: sets-pair-measure)
then have  $(\lambda a. \text{emeasure } (f \ a \otimes_M g \ a) \ (\bigcup i. A \ i)) \in \text{borel-measurable } M \longleftrightarrow$ 
 $(\lambda a. \sum i. \text{emeasure } (f \ a \otimes_M g \ a) \ (A \ i)) \in \text{borel-measurable } M$ 
  by (intro measurable-cong suminf-emeasure[symmetric])
  (auto simp: sets-eq)
also have ...
  using union by auto
finally show ?case .
qed simp
qed

lemma restrict-space-measurable:
  assumes X:  $X \neq \{\}$   $X \in \text{sets } K$ 
  assumes N:  $N \in \text{measurable } M \text{ (subprob-algebra } K)$ 
  shows  $(\lambda x. \text{restrict-space } (N \ x) \ X) \in \text{measurable } M \text{ (subprob-algebra (restrict-space } K \ X))$ 
proof (rule measurable-subprob-algebra)
  fix a assume a:  $a \in \text{space } M$ 
  from N [THEN measurable-space, OF this]
  have subprob-space  $(N \ a)$  and [simp]:  $\text{sets } (N \ a) = \text{sets } K \text{ space } (N \ a) = \text{space } K$ 
  by (auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
  then interpret subprob-space  $N \ a$ 
  by simp
  show subprob-space  $(\text{restrict-space } (N \ a) \ X)$ 
  proof
    show  $\text{space } (\text{restrict-space } (N \ a) \ X) \neq \{\}$ 
    using X by (auto simp add: space-restrict-space)
    show  $\text{emeasure } (\text{restrict-space } (N \ a) \ X) \ (\text{space } (\text{restrict-space } (N \ a) \ X)) \leq 1$ 
    using X by (simp add: emeasure-restrict-space space-restrict-space subprob-emeasure-le-1)
  qed
  show  $\text{sets } (\text{restrict-space } (N \ a) \ X) = \text{sets } (\text{restrict-space } K \ X)$ 
  by (intro sets-restrict-space-cong) fact
next
fix A assume A:  $A \in \text{sets } (\text{restrict-space } K \ X)$ 
show  $(\lambda a. \text{emeasure } (\text{restrict-space } (N \ a) \ X) \ A) \in \text{borel-measurable } M$ 
proof (subst measurable-cong)
  fix a assume  $a \in \text{space } M$ 

```

```

from  $N[THEN \text{ measurable-space, OF this}]$ 
have  $[simp]: \text{sets } (N \ a) = \text{sets } K \ \text{space } (N \ a) = \text{space } K$ 
  by  $(\text{auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq})$ 
show  $\text{emeasure } (\text{restrict-space } (N \ a) \ X) \ A = \text{emeasure } (N \ a) \ (A \cap X)$ 
  using  $X \ A$  by  $(\text{subst emeasure-restrict-space}) (\text{auto simp add: sets-restrict-space ac-simps})$ 
next
  show  $(\lambda w. \text{emeasure } (N \ w) \ (A \cap X)) \in \text{borel-measurable } M$ 
  using  $A \ X$ 
  by  $(\text{intro measurable-compose}[OF \ N \ \text{measurable-emeasure-subprob-algebra}])$ 
   $(\text{auto simp: sets-restrict-space})$ 
qed
qed

```

7.2 Properties of “return”

definition $\text{return} :: 'a \text{ measure} \Rightarrow 'a \Rightarrow 'a \text{ measure}$ **where**
 $\text{return } R \ x = \text{measure-of } (\text{space } R) \ (\text{sets } R) \ (\lambda A. \text{indicator } A \ x)$

lemma $\text{space-return}[simp]: \text{space } (\text{return } M \ x) = \text{space } M$
by $(\text{simp add: return-def})$

lemma $\text{sets-return}[simp]: \text{sets } (\text{return } M \ x) = \text{sets } M$
by $(\text{simp add: return-def})$

lemma $\text{measurable-return1}[simp]: \text{measurable } (\text{return } N \ x) \ L = \text{measurable } N \ L$
by $(\text{simp cong: measurable-cong-sets})$

lemma $\text{measurable-return2}[simp]: \text{measurable } L \ (\text{return } N \ x) = \text{measurable } L \ N$
by $(\text{simp cong: measurable-cong-sets})$

lemma $\text{return-sets-cong}: \text{sets } M = \text{sets } N \implies \text{return } M = \text{return } N$
by $(\text{auto dest: sets-eq-imp-space-eq simp: fun-eq-iff return-def})$

lemma $\text{return-cong}: \text{sets } A = \text{sets } B \implies \text{return } A \ x = \text{return } B \ x$
by $(\text{auto simp add: return-def dest: sets-eq-imp-space-eq})$

lemma $\text{emeasure-return}[simp]:$
assumes $A \in \text{sets } M$
shows $\text{emeasure } (\text{return } M \ x) \ A = \text{indicator } A \ x$
proof $(\text{rule emeasure-measure-of}[OF \ \text{return-def}])$
show $\text{sets } M \subseteq \text{Pow } (\text{space } M)$ **by** $(\text{rule sets.space-closed})$
show $\text{positive } (\text{sets } (\text{return } M \ x)) \ (\lambda A. \text{indicator } A \ x)$ **by** $(\text{simp add: positive-def})$
from assms **show** $A \in \text{sets } (\text{return } M \ x)$ **unfolding** return-def **by** simp
show $\text{countably-additive } (\text{sets } (\text{return } M \ x)) \ (\lambda A. \text{indicator } A \ x)$
by $(\text{auto intro!: countably-additiveI suminf-indicator})$
qed

lemma $\text{prob-space-return}: x \in \text{space } M \implies \text{prob-space } (\text{return } M \ x)$

by rule simp

lemma *subprob-space-return*: $x \in \text{space } M \implies \text{subprob-space } (\text{return } M x)$
 by (intro prob-space-return prob-space-imp-subprob-space)

lemma *subprob-space-return-ne*:
 assumes $\text{space } M \neq \{\}$ **shows** $\text{subprob-space } (\text{return } M x)$
 by (metis assms emeasure-return indicator-simps(2) sets.top space-return sub-
 prob-spaceI subprob-space-return zero-le)

lemma *measure-return*: **assumes** $X: X \in \text{sets } M$ **shows** $\text{measure } (\text{return } M x) X$
 $= \text{indicator } X x$
 unfolding measure-def emeasure-return[OF X , of x] by (simp split: split-indicator)

lemma *AE-return*:
 assumes [simp]: $x \in \text{space } M$ and [measurable]: $\text{Measurable.pred } M P$
 shows $(AE y \text{ in } \text{return } M x. P y) \longleftrightarrow P x$
proof –
 have $(AE y \text{ in } \text{return } M x. y \notin \{x \in \text{space } M. \neg P x\}) \longleftrightarrow P x$
 by (subst AE-iff-null-sets[symmetric]) (simp-all add: null-sets-def split: split-indicator)
 also have $(AE y \text{ in } \text{return } M x. y \notin \{x \in \text{space } M. \neg P x\}) \longleftrightarrow (AE y \text{ in } \text{return } M x. P y)$
 by (rule AE-cong) auto
 finally show ?thesis .
qed

lemma *nn-integral-return*:
 assumes $x \in \text{space } M$ $g \in \text{borel-measurable } M$
 shows $(\int^+ a. g a \partial \text{return } M x) = g x$
proof –
 interpret prob-space return $M x$ by (rule prob-space-return[OF $\langle x \in \text{space } M \rangle$])
 have $(\int^+ a. g a \partial \text{return } M x) = (\int^+ a. g x \partial \text{return } M x)$ using assms
 by (intro nn-integral-cong-AE) (auto simp: AE-return)
 also have $\dots = g x$
 using nn-integral-const[of return $M x$] emeasure-space-1 by simp
 finally show ?thesis .
qed

lemma *integral-return*:
 fixes $g :: - \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$
 assumes $x \in \text{space } M$ $g \in \text{borel-measurable } M$
 shows $(\int a. g a \partial \text{return } M x) = g x$
proof –
 interpret prob-space return $M x$ by (rule prob-space-return[OF $\langle x \in \text{space } M \rangle$])
 have $(\int a. g a \partial \text{return } M x) = (\int a. g x \partial \text{return } M x)$ using assms
 by (intro integral-cong-AE) (auto simp: AE-return)
 then show ?thesis
 using prob-space by simp
qed

lemma *return-measurable*[*measurable*]: *return* $N \in \text{measurable } N$ (*subprob-algebra* N)

by (*rule measurable-subprob-algebra*) (*auto simp: subprob-space-return*)

lemma *distr-return*:

assumes $f \in \text{measurable } M \ N$ **and** $x \in \text{space } M$

shows $\text{distr } (\text{return } M \ x) \ N \ f = \text{return } N \ (f \ x)$

using *assms* **by** (*intro measure-eqI*) (*simp-all add: indicator-def emeasure-distr*)

lemma *return-restrict-space*:

$\Omega \in \text{sets } M \implies \text{return } (\text{restrict-space } M \ \Omega) \ x = \text{restrict-space } (\text{return } M \ x) \ \Omega$

by (*auto intro!: measure-eqI simp: sets-restrict-space emeasure-restrict-space*)

lemma *measurable-distr2*:

assumes $f[\text{measurable}]$: $\text{case-prod } f \in \text{measurable } (L \otimes_M M) \ N$

assumes $g[\text{measurable}]$: $g \in \text{measurable } L$ (*subprob-algebra* M)

shows $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in \text{measurable } L$ (*subprob-algebra* N)

proof –

have $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in \text{measurable } L$ (*subprob-algebra* N)

$\longleftrightarrow (\lambda x. \text{distr } (\text{return } L \ x \otimes_M g \ x) \ N \ (\text{case-prod } f)) \in \text{measurable } L$ (*subprob-algebra* N)

proof (*rule measurable-cong*)

fix x **assume** $x: x \in \text{space } L$

have $g x: g \ x \in \text{space } (\text{subprob-algebra } M)$

using *measurable-space[OF g x]* .

then have [*simp*]: $\text{sets } (g \ x) = \text{sets } M$

by (*simp add: space-subprob-algebra*)

then have [*simp*]: $\text{space } (g \ x) = \text{space } M$

by (*rule sets-eq-imp-space-eq*)

let $?R = \text{return } L \ x$

from *measurable-compose-Pair1*[*OF x f*] **have** $f \cdot M'$: $f \ x \in \text{measurable } M \ N$

by *simp*

interpret *subprob-space* $g \ x$

using $g x$ **by** (*simp add: space-subprob-algebra*)

have *space-pair-M'*[*simp*]: $\bigwedge X. \text{space } (X \otimes_M g \ x) = \text{space } (X \otimes_M M)$

by (*simp add: space-pair-measure*)

show $\text{distr } (g \ x) \ N \ (f \ x) = \text{distr } (?R \otimes_M g \ x) \ N \ (\text{case-prod } f)$ (**is** $?l = ?r$)

proof (*rule measure-eqI*)

show $\text{sets } ?l = \text{sets } ?r$

by *simp*

next

fix A **assume** $A \in \text{sets } ?l$

then have $A[\text{measurable}]$: $A \in \text{sets } N$

by *simp*

then have $\text{emeasure } ?r \ A = \text{emeasure } (?R \otimes_M g \ x) \ ((\lambda(x, y). f \ x \ y) - ' A \cap \text{space } (?R \otimes_M g \ x))$

by (*auto simp add: emeasure-distr f-M' cong: measurable-cong-sets*)

also have $\dots = (\int^+ M''. \text{emeasure } (g \ x) \ (f \ M'' - ' A \cap \text{space } M) \ \partial ?R)$

```

    apply (subst emeasure-pair-measure-alt)
    apply (force simp add: f-M' cong: measurable-cong-sets intro!: measur-
able-sets[OF - A])
    apply (intro nn-integral-cong arg-cong[where f=emeasure (g x)])
    apply (auto simp: space-subprob-algebra space-pair-measure)
    done
  also have ... = emeasure (g x) (f x - ' A ∩ space M)
    by (subst nn-integral-return)
      (auto simp: x intro!: measurable-emeasure)
  also have ... = emeasure ?l A
    by (simp add: emeasure-distr f-M' cong: measurable-cong-sets)
  finally show emeasure ?l A = emeasure ?r A ..
qed
qed
also have ...
proof (intro measurable-compose[OF measurable-pair-measure measurable-distr])
  show return L ∈ L →M subprob-algebra L
    by (rule return-measurable)
qed measurable
finally show ?thesis .
qed

```

lemma *nn-integral-measurable-subprob-algebra2*:

```

  assumes f[measurable]: (λ(x, y). f x y) ∈ borel-measurable (M ⊗M N)
  assumes N[measurable]: L ∈ measurable M (subprob-algebra N)
  shows (λx. integralN (L x) (f x)) ∈ borel-measurable M
proof -
  note nn-integral-measurable-subprob-algebra[measurable]
  note measurable-distr2[measurable]
  have (λx. integralN (distr (L x) (M ⊗M N) (λy. (x, y))) (λ(x, y). f x y)) ∈
borel-measurable M
    by measurable
  then show (λx. integralN (L x) (f x)) ∈ borel-measurable M
    by (rule measurable-cong[THEN iffD1, rotated])
      (simp add: nn-integral-distr)
qed

```

lemma *emeasure-measurable-subprob-algebra2*:

```

  assumes A[measurable]: (SIGMA x:space M. A x) ∈ sets (M ⊗M N)
  assumes L[measurable]: L ∈ measurable M (subprob-algebra N)
  shows (λx. emeasure (L x) (A x)) ∈ borel-measurable M
proof -
  { fix x assume x ∈ space M
    then have Pair x - ' Sigma (space M) A = A x
      by auto
    with sets-Pair1[OF A, of x] have A x ∈ sets N
      by auto }
  note ** = this

```

```

have *:  $\bigwedge x. \text{fst } x \in \text{space } M \implies \text{snd } x \in A \iff x \in (\text{SIGMA } x:\text{space } M. A \ x)$ 
  by (auto simp: fun-eq-iff)
have MN: Measurable.pred (M  $\otimes_M$  N) ( $\lambda w. w \in \text{Sigma } (\text{space } M) \ A$ )
  by auto
then have ( $\lambda(x, y). \text{indicator } (A \ x) \ y::\text{ennreal}$ )  $\in \text{borel-measurable } (M \otimes_M N)$ 
  apply measurable
  by (smt (verit, best) MN measurable-cong mem-Sigma-iff prod.collapse space-pair-measure)
then have ( $\lambda x. \text{integral}^N (L \ x) (\text{indicator } (A \ x))$ )  $\in \text{borel-measurable } M$ 
  by (intro nn-integral-measurable-subprob-algebra2[where N=N] L)
then show ( $\lambda x. \text{emeasure } (L \ x) (A \ x)$ )  $\in \text{borel-measurable } M$ 
  by (smt (verit) ** L measurable-cong-simp nn-integral-indicator sets-kernel)
qed

```

```

lemma measure-measurable-subprob-algebra2:
  assumes A[measurable]:  $(\text{SIGMA } x:\text{space } M. A \ x) \in \text{sets } (M \otimes_M N)$ 
  assumes L[measurable]:  $L \in \text{measurable } M \ (\text{subprob-algebra } N)$ 
  shows ( $\lambda x. \text{measure } (L \ x) (A \ x)$ )  $\in \text{borel-measurable } M$ 
  unfolding measure-def
  by (intro borel-measurable-enn2real emeasure-measurable-subprob-algebra2[OF assms])

```

```

definition select-sets M = (SOME N. sets M = sets (subprob-algebra N))

```

```

lemma select-sets1:
  sets M = sets (subprob-algebra N)  $\implies$  sets M = sets (subprob-algebra (select-sets M))
  unfolding select-sets-def by (rule someI)

```

```

lemma sets-select-sets[simp]:
  assumes sets: sets M = sets (subprob-algebra N)
  shows sets (select-sets M) = sets N
  unfolding select-sets-def
proof (rule someI2)
  show sets M = sets (subprob-algebra N)
  by fact
next
  fix L assume sets M = sets (subprob-algebra L)
  with sets have eq: space (subprob-algebra N) = space (subprob-algebra L)
  by (intro sets-eq-imp-space-eq) simp
  show sets L = sets N
  proof cases
    assume space (subprob-algebra N) = {}
    with space-subprob-algebra-empty-iff[of N] space-subprob-algebra-empty-iff[of L]
    show ?thesis
    by (simp add: eq space-empty-iff)
  next
    assume space (subprob-algebra N)  $\neq \{\}$ 
    with eq show ?thesis

```

by (smt (verit) equals0I mem-Collect-eq space-subprob-algebra)
 qed
 qed

lemma *space-select-sets[simp]*:
 sets $M = \text{sets}(\text{subprob-algebra } N) \implies \text{space}(\text{select-sets } M) = \text{space } N$
 by (intro sets-eq-imp-space-eq sets-select-sets)

7.3 Join

definition *join* :: 'a measure \Rightarrow 'a measure **where**
 $\text{join } M = \text{measure-of}(\text{space}(\text{select-sets } M))(\text{sets}(\text{select-sets } M))(\lambda B. \int^+ M'. \text{emeasure } M' B \partial M)$

lemma
 shows *space-join[simp]*: $\text{space}(\text{join } M) = \text{space}(\text{select-sets } M)$
 and *sets-join[simp]*: $\text{sets}(\text{join } M) = \text{sets}(\text{select-sets } M)$
 by (simp-all add: join-def)

lemma *emeasure-join*:
 assumes $M[\text{simp}, \text{measurable-cong}]$: $\text{sets } M = \text{sets}(\text{subprob-algebra } N)$ and A :
 $A \in \text{sets } N$
 shows $\text{emeasure}(\text{join } M) A = (\int^+ M'. \text{emeasure } M' A \partial M)$
proof (rule *emeasure-measure-of[OF join-def]*)
 show *countably-additive* ($\text{sets}(\text{join } M)$) ($\lambda B. \int^+ M'. \text{emeasure } M' B \partial M$)
proof (rule *countably-additiveI*)
 fix $A :: \text{nat} \Rightarrow 'a \text{ set}$ **assume** A : $\text{range } A \subseteq \text{sets}(\text{join } M)$ *disjoint-family* A
 have $(\sum i. \int^+ M'. \text{emeasure } M' (A i) \partial M) = (\int^+ M'. (\sum i. \text{emeasure } M' (A i)) \partial M)$
 using A **by** (*subst nn-integral-suminf*) (*auto simp: measurable-emeasure-subprob-algebra*)
 also have $\dots = (\int^+ M'. \text{emeasure } M' (\bigcup i. A i) \partial M)$
proof (rule *nn-integral-cong*)
 fix M' **assume** $M' \in \text{space } M$
 then show $(\sum i. \text{emeasure } M' (A i)) = \text{emeasure } M' (\bigcup i. A i)$
 using A *sets-eq-imp-space-eq[OF M]* **by** (*simp add: suminf-emeasure space-subprob-algebra*)
 qed
 finally show $(\sum i. \int^+ M'. \text{emeasure } M' (A i) \partial M) = (\int^+ M'. \text{emeasure } M' (\bigcup i. A i) \partial M)$.
 qed
qed (*auto simp: A sets.space-closed positive-def*)

lemma *measurable-join*:
 $\text{join} \in \text{measurable}(\text{subprob-algebra}(\text{subprob-algebra } N))(\text{subprob-algebra } N)$
proof (*cases space N $\neq \{\}$, rule measurable-subprob-algebra*)
 fix A **assume** $A \in \text{sets } N$
 let $?B = \text{borel-measurable}(\text{subprob-algebra}(\text{subprob-algebra } N))$
 have $(\lambda M'. \text{emeasure}(\text{join } M') A) \in ?B \longleftrightarrow (\lambda M'. (\int^+ M''. \text{emeasure } M'' A \partial M')) \in ?B$

```

proof (rule measurable-cong)
  fix  $M'$  assume  $M' \in \text{space } (\text{subprob-algebra } (\text{subprob-algebra } N))$ 
  then show  $\text{emeasure } (\text{join } M') A = (\int^+ M''. \text{emeasure } M'' A \partial M')$ 
    by (intro emeasure-join) (auto simp: space-subprob-algebra  $\langle A \in \text{sets } N \rangle$ )
qed
also have  $(\lambda M'. \int^+ M''. \text{emeasure } M'' A \partial M') \in ?B$ 
  using measurable-emeasure-subprob-algebra[OF  $\langle A \in \text{sets } N \rangle$ ]
  by (rule nn-integral-measurable-subprob-algebra)
  finally show  $(\lambda M'. \text{emeasure } (\text{join } M') A) \in \text{borel-measurable } (\text{subprob-algebra } (\text{subprob-algebra } N))$  .
next
  assume [simp]:  $\text{space } N \neq \{\}$ 
  fix  $M$  assume  $M: M \in \text{space } (\text{subprob-algebra } (\text{subprob-algebra } N))$ 
  then have  $(\int^+ M'. \text{emeasure } M' (\text{space } N) \partial M) \leq (\int^+ M'. 1 \partial M)$ 
  proof (intro nn-integral-mono)
    show  $\bigwedge x. \llbracket M \in \text{space } (\text{subprob-algebra } (\text{subprob-algebra } N)); x \in \text{space } M \rrbracket$ 
       $\implies \text{emeasure } x (\text{space } N) \leq 1$ 
    by (smt (verit) mem-Collect-eq sets-eq-imp-space-eq space-subprob-algebra
      subprob-space.subprob-emeasure-le-1)
  qed
  with  $M$  show  $\text{subprob-space } (\text{join } M)$ 
  by (intro subprob-spaceI)
  (auto simp: emeasure-join space-subprob-algebra  $M \text{ dest: subprob-space.emeasure-space-le-1}$ )
next
  assume  $\neg(\text{space } N \neq \{\})$ 
  thus ?thesis by (simp add: measurable-empty-iff space-subprob-algebra-empty-iff)
qed (auto simp: space-subprob-algebra)

lemma nn-integral-join:
  assumes  $f: f \in \text{borel-measurable } N$ 
  and  $M[\text{measurable-cong}]: \text{sets } M = \text{sets } (\text{subprob-algebra } N)$ 
  shows  $(\int^+ x. f x \partial \text{join } M) = (\int^+ M'. \int^+ x. f x \partial M' \partial M)$ 
  using  $f$ 
proof induct
  case (cong  $f g$ )
    moreover have  $\text{integral}^N (\text{join } M) f = \text{integral}^N (\text{join } M) g$ 
    by (intro nn-integral-cong cong) (simp add:  $M$ )
    moreover from  $M$  have  $(\int^+ M'. \text{integral}^N M' f \partial M) = (\int^+ M'. \text{integral}^N$ 
       $M' g \partial M)$ 
    by (intro nn-integral-cong cong)
    (auto simp add: space-subprob-algebra dest!: sets-eq-imp-space-eq)
    ultimately show ?case
    by simp
  next
    case (set  $A$ )
    with  $M$  have  $(\int^+ M'. \text{integral}^N M' (\text{indicator } A) \partial M) = (\int^+ M'. \text{emeasure}$ 
       $M' A \partial M)$ 
    by (intro nn-integral-cong nn-integral-indicator)
    (auto simp: space-subprob-algebra dest!: sets-eq-imp-space-eq)

```

```

with set show ?case
  using M by (simp add: emeasure-join)
next
  case (mult f c)
  have  $(\int^+ M'. \int^+ x. c * f x \partial M' \partial M) = (\int^+ M'. c * \int^+ x. f x \partial M' \partial M)$ 
    using mult M M [THEN sets-eq-imp-space-eq]
    by (intro nn-integral-cong nn-integral-cmult) (auto simp add: space-subprob-algebra)
  also have  $\dots = c * (\int^+ M'. \int^+ x. f x \partial M' \partial M)$ 
    using nn-integral-measurable-subprob-algebra [OF mult(2)]
    by (intro nn-integral-cmult mult) (simp add: M)
  also have  $\dots = c * (\text{integral}^N (\text{join } M) f)$ 
    by (simp add: mult)
  also have  $\dots = (\int^+ x. c * f x \partial \text{join } M)$ 
    using mult(2,3) by (intro nn-integral-cmult[symmetric] mult) (simp add: M
cong: measurable-cong-sets)
  finally show ?case by simp
next
  case (add f g)
  have  $(\int^+ M'. \int^+ x. f x + g x \partial M' \partial M) = (\int^+ M'. (\int^+ x. f x \partial M') + (\int^+ x. g x \partial M') \partial M)$ 
    using add M M [THEN sets-eq-imp-space-eq]
    by (intro nn-integral-cong nn-integral-add) (auto simp add: space-subprob-algebra)
  also have  $\dots = (\int^+ M'. \int^+ x. f x \partial M' \partial M) + (\int^+ M'. \int^+ x. g x \partial M' \partial M)$ 
    using nn-integral-measurable-subprob-algebra [OF add(1)]
    using nn-integral-measurable-subprob-algebra [OF add(4)]
    by (intro nn-integral-add add) (simp-all add: M)
  also have  $\dots = (\text{integral}^N (\text{join } M) f) + (\text{integral}^N (\text{join } M) g)$ 
    by (simp add: add)
  also have  $\dots = (\int^+ x. f x + g x \partial \text{join } M)$ 
    using add by (intro nn-integral-add[symmetric] add) (simp-all add: M cong: measurable-cong-sets)
  finally show ?case by (simp add: ac-simps)
next
  case (seq F)
  have  $(\int^+ M'. \int^+ x. (\text{SUP } i. F i) x \partial M' \partial M) = (\int^+ M'. (\text{SUP } i. \int^+ x. F i x \partial M') \partial M)$ 
    using seq M M [THEN sets-eq-imp-space-eq] unfolding SUP-apply
    by (intro nn-integral-cong nn-integral-monotone-convergence-SUP)
    (auto simp add: space-subprob-algebra)
  also have  $\dots = (\text{SUP } i. \int^+ M'. \int^+ x. F i x \partial M' \partial M)$ 
    using nn-integral-measurable-subprob-algebra [OF seq(1)] seq
    by (intro nn-integral-monotone-convergence-SUP)
    (simp-all add: M incseq-nn-integral incseq-def le-fun-def nn-integral-mono)
  also have  $\dots = (\text{SUP } i. \text{integral}^N (\text{join } M) (F i))$ 
    by (simp add: seq)
  also have  $\dots = (\int^+ x. (\text{SUP } i. F i x) \partial \text{join } M)$ 
    using seq by (intro nn-integral-monotone-convergence-SUP[symmetric] seq)
    (simp-all add: M cong: measurable-cong-sets)
  finally show ?case by (simp add: ac-simps image-comp)

```

qed

lemma *measurable-join1*:

$\llbracket f \in \text{measurable } N \ K; \text{ sets } M = \text{sets } (\text{subprob-algebra } N) \rrbracket$
 $\implies f \in \text{measurable } (\text{join } M) \ K$
by(*simp add: measurable-def*)

lemma

fixes $f :: - \Rightarrow \text{real}$
assumes $f\text{-measurable}$ [*measurable*]: $f \in \text{borel-measurable } N$
and $f\text{-bounded}$: $\bigwedge x. x \in \text{space } N \implies |f\ x| \leq B$
and M [*measurable-cong*]: $\text{sets } M = \text{sets } (\text{subprob-algebra } N)$
and fin : *finite-measure* M
and $M\text{-bounded}$: $\text{AE } M' \text{ in } M. \text{emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
shows *integrable-join*: $\text{integrable } (\text{join } M) \ f$ (**is** *?integrable*)
and *integral-join*: $\text{integral}^L (\text{join } M) \ f = \int M'. \text{integral}^L M' f \ \partial M$ (**is** *?integral*)
proof(*case-tac* [*!*] $\text{space } N = \{\}$)
assume *: $\text{space } N = \{\}$
show *?integrable*
using $M * \text{by}$ (*simp add: real-integrable-def measurable-def nn-integral-empty*)
have $(\int M'. \text{integral}^L M' f \ \partial M) = (\int M'. 0 \ \partial M)$
proof(*rule Bochner-Integration.integral-cong*)
fix M'
assume $M' \in \text{space } M$
with *sets-eq-imp-space-eq*[*OF* M] **have** $\text{space } M' = \text{space } N$
by(*auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq*)
with * **show** $(\int x. f\ x \ \partial M') = 0$ **by**(*simp add: Bochner-Integration.integral-empty*)
qed *simp*
then show *?integral*
using $M * \text{by}$ (*simp add: Bochner-Integration.integral-empty*)
next
assume *: $\text{space } N \neq \{\}$

from * **have** B [*simp*]: $0 \leq B$ **by**(*auto dest: f-bounded*)

have [*measurable*]: $f \in \text{borel-measurable } (\text{join } M)$ **using** $f\text{-measurable } M$
by(*rule measurable-join1*)

{ **fix** $f\ M'$

assume [*measurable*]: $f \in \text{borel-measurable } N$
and $f\text{-bounded}$: $\bigwedge x. x \in \text{space } N \implies f\ x \leq B$
and $M' \in \text{space } M$ $\text{emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
have $\text{AE } x \text{ in } M'. \text{ennreal } (f\ x) \leq \text{ennreal } B$
proof(*rule AE-I2*)

fix x

assume $x \in \text{space } M'$

with $\langle M' \in \text{space } M \rangle$ *sets-eq-imp-space-eq*[*OF* M]

have $x \in \text{space } N$ **by**(*auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq*)
from $f\text{-bounded}$ [*OF* *this*] **show** $\text{ennreal } (f\ x) \leq \text{ennreal } B$ **by** *simp*

```

qed
then have  $(\int^+ x. \text{ennreal } (f x) \partial M') \leq (\int^+ x. \text{ennreal } B \partial M')$ 
  by(rule nn-integral-mono-AE)
also have  $\dots = \text{ennreal } B * \text{emeasure } M' (\text{space } M')$  by(simp)
also have  $\dots \leq \text{ennreal } B * \text{ennreal } B'$  by(rule mult-left-mono)(fact, simp)
also have  $\dots \leq \text{ennreal } B * \text{ennreal } |B'|$  by(rule mult-left-mono)(simp-all)
finally have  $(\int^+ x. \text{ennreal } (f x) \partial M') \leq \text{ennreal } (B * |B'|)$  by (simp add:
ennreal-mult) }
note bounded1 = this

have bounded:
 $\bigwedge f. [\![ f \in \text{borel-measurable } N; \bigwedge x. x \in \text{space } N \implies f x \leq B ]\!] \implies (\int^+ x. \text{ennreal } (f x) \partial \text{join } M) \neq \text{top}$ 
proof -
  fix f
  assume [measurable]:  $f \in \text{borel-measurable } N$ 
  and f-bounded:  $\bigwedge x. x \in \text{space } N \implies f x \leq B$ 
  have  $(\int^+ x. \text{ennreal } (f x) \partial \text{join } M) = (\int^+ M'. \int^+ x. \text{ennreal } (f x) \partial M' \partial M)$ 
    by(rule nn-integral-join[OF - M]) simp
  also have  $\dots \leq \int^+ M'. B * |B'| \partial M$ 
    using bounded1[OF  $\langle f \in \text{borel-measurable } N \rangle$  f-bounded]
    by(rule nn-integral-mono-AE[OF AE-mp[OF M-bounded AE-I2], rule-format])
  also have  $\dots = B * |B'| * \text{emeasure } M (\text{space } M)$  by simp
  also have  $\dots < \infty$ 
    using finite-measure.finite-emeasure-space[OF fin]
    by(simp add: ennreal-mult-less-top less-top)
  finally show ?thesis f by simp
qed
have f-pos:  $(\int^+ x. \text{ennreal } (f x) \partial \text{join } M) \neq \infty$ 
  and f-neg:  $(\int^+ x. \text{ennreal } (-f x) \partial \text{join } M) \neq \infty$ 
  using f-bounded by(auto del: notI intro!: bounded simp add: abs-le-iff)

show ?integrable using f-pos f-neg by(simp add: real-integrable-def)

note [measurable] = nn-integral-measurable-subprob-algebra

have int-f:  $(\int^+ x. f x \partial \text{join } M) = \int^+ M'. \int^+ x. f x \partial M' \partial M$ 
  by(simp add: nn-integral-join[OF - M])
have int-mf:  $(\int^+ x. -f x \partial \text{join } M) = (\int^+ M'. \int^+ x. -f x \partial M' \partial M)$ 
  by(simp add: nn-integral-join[OF - M])

have pos-finite:  $AE M' \text{ in } M. (\int^+ x. f x \partial M') \neq \infty$ 
  using AE-space M-bounded
proof eventually-elim
  fix M' assume  $M' \in \text{space } M \text{ emeasure } M' (\text{space } M') \leq \text{ennreal } B'$ 
  then have  $(\int^+ x. \text{ennreal } (f x) \partial M') \leq \text{ennreal } (B * |B'|)$ 
    using f-measurable by(auto intro!: bounded1 dest: f-bounded)
  then show  $(\int^+ x. \text{ennreal } (f x) \partial M') \neq \infty$ 
    by (auto simp: top-unique)

```


qed
hence $[simp]: (\int^+ M'. \text{ennreal} (\text{enn2real} (\int^+ x. f x \partial M')) \partial M) = (\int^+ M'. \int^+ x. f x \partial M' \partial M)$
by (rule nn-integral-cong-AE[OF AE-mp]) (simp add: less-top)
from $f\text{-pos}$ **have** $[simp]: \text{integrable } M (\lambda M'. \text{enn2real} (\int^+ x. f x \partial M'))$
by (simp add: int-f real-integrable-def nn-integral-0-iff-AE[THEN iffD2] en-nreal-neg enn2real-nonneg)

have $\text{neg-finite}: AE \ M' \text{ in } M. (\int^+ x. - f x \partial M') \neq \infty$
using AE-space M-bounded
proof eventually-elim
fix M' **assume** $M' \in \text{space } M \text{ emeasure } M' (\text{space } M') \leq \text{ennreal } B'$
then have $(\int^+ x. \text{ennreal} (- f x) \partial M') \leq \text{ennreal} (B * |B'|)$
using $f\text{-measurable}$ **by** (auto intro!: bounded1 dest: f-bounded)
then show $(\int^+ x. \text{ennreal} (- f x) \partial M') \neq \infty$
by (auto simp: top-unique)
qed
hence $[simp]: (\int^+ M'. \text{ennreal} (\text{enn2real} (\int^+ x. - f x \partial M')) \partial M) = (\int^+ M'. \int^+ x. - f x \partial M' \partial M)$
by (rule nn-integral-cong-AE[OF AE-mp]) (simp add: less-top)
from $f\text{-neg}$ **have** $[simp]: \text{integrable } M (\lambda M'. \text{enn2real} (\int^+ x. - f x \partial M'))$
by (simp add: int-mf real-integrable-def nn-integral-0-iff-AE[THEN iffD2] en-nreal-neg enn2real-nonneg)

have $(\int x. f x \partial \text{join } M) = \text{enn2real} (\int^+ N. \int^+ x. f x \partial N \partial M) - \text{enn2real} (\int^+ N. \int^+ x. - f x \partial N \partial M)$
unfolding real-lebesgue-integral-def[OF $\langle ?\text{integrable} \rangle$] **by** (simp add: nn-integral-join[OF - M])
also have $\dots = (\int N. \text{enn2real} (\int^+ x. f x \partial N) \partial M) - (\int N. \text{enn2real} (\int^+ x. - f x \partial N) \partial M)$
using pos-finite neg-finite **by** (subst (1 2) integral-eq-nn-integral) (auto simp: enn2real-nonneg)
also have $\dots = (\int N. \text{enn2real} (\int^+ x. f x \partial N) - \text{enn2real} (\int^+ x. - f x \partial N) \partial M)$
by simp
also have $\dots = \int M'. \int x. f x \partial M' \partial M$
proof (rule integral-cong-AE)
show $AE \ x \text{ in } M.$
 $\text{enn2real} (\int^+ x. \text{ennreal} (f x) \partial x) - \text{enn2real} (\int^+ x. \text{ennreal} (- f x) \partial x) = \text{integral}^L x f$
using AE-space M-bounded
proof eventually-elim
fix M' **assume** $M' \in \text{space } M \text{ emeasure } M' (\text{space } M') \leq B'$
then interpret subprob-space M'
by (auto simp: $M[THEN \text{sets-eq-imp-space-eq}] \text{space-subprob-algebra}$)

from $\langle M' \in \text{space } M \rangle \text{sets-eq-imp-space-eq}[OF \ M]$
have $[\text{measurable-cong}]: \text{sets } M' = \text{sets } N$ **by** (simp add: space-subprob-algebra)
hence $[simp]: \text{space } M' = \text{space } N$ **by** (rule sets-eq-imp-space-eq)

```

    have integrable M' f
      by (rule integrable-const-bound[where B=B])(auto simp add: f-bounded)
    then show enn2real ( $\int^+ x. f x \partial M'$ ) - enn2real ( $\int^+ x. - f x \partial M'$ ) =  $\int x. f x \partial M'$ 
      by (simp add: real-lebesgue-integral-def)
    qed
  qed simp-all
  finally show ?integral by simp
qed

```

lemma join-assoc:

```

  assumes M[measurable-cong]: sets M = sets (subprob-algebra (subprob-algebra N))
  shows join (distr M (subprob-algebra N) join) = join (join M)
proof (rule measure-eqI)
  fix A assume A ∈ sets (join (distr M (subprob-algebra N) join))
  then have A: A ∈ sets N by simp
  show emeasure (join (distr M (subprob-algebra N) join)) A = emeasure (join (join M)) A
    using measurable-join[of N]
    by (auto simp: M A nn-integral-distr emeasure-join measurable-emeasure-subprob-algebra
      sets-eq-imp-space-eq[OF M] space-subprob-algebra nn-integral-join[OF
- M]
      intro!: nn-integral-cong emeasure-join)
qed (simp add: M)

```

lemma join-return:

```

  assumes sets M = sets N and subprob-space M
  shows join (return (subprob-algebra N) M) = M
  by (rule measure-eqI)
    (simp-all add: emeasure-join space-subprob-algebra
      measurable-emeasure-subprob-algebra nn-integral-return assms)

```

lemma join-return':

```

  assumes sets N = sets M
  shows join (distr M (subprob-algebra N) (return N)) = M
proof (rule measure-eqI)
  fix A
  have return N ∈ measurable M (subprob-algebra N)
    using assms by auto
  moreover
  assume A ∈ sets (join (distr M (subprob-algebra N) (return N)))
  ultimately show emeasure (join (distr M (subprob-algebra N) (return N))) A
    = emeasure M A
    by (simp add: emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra
      assms)
qed (simp add: assms)

```

lemma join-distr-distr:

fixes $f :: 'a \Rightarrow 'b$ **and** $M :: 'a \text{ measure measure}$ **and** $N :: 'b \text{ measure}$
assumes $\text{sets } M = \text{sets } (\text{subprob-algebra } R)$ **and** $f \in \text{measurable } R \ N$
shows $\text{join } (\text{distr } M \ (\text{subprob-algebra } N) \ (\lambda M. \text{distr } M \ N \ f)) = \text{distr } (\text{join } M) \ N \ f$ **(is ?r = ?l)**
proof (rule *measure-eqI*)
fix A **assume** $A \in \text{sets } ?r$
hence $A\text{-in-}N: A \in \text{sets } N$ **by** *simp*

from *assms* **have** $f \in \text{measurable } (\text{join } M) \ N$
by (*simp cong: measurable-cong-sets*)
moreover from *assms* **and** $A\text{-in-}N$ **have** $f - 'A \cap \text{space } R \in \text{sets } R$
by (*intro measurable-sets*) *simp-all*
ultimately have $\text{emeasure } (\text{distr } (\text{join } M) \ N \ f) \ A = \int^+ M'. \text{emeasure } M' (f - 'A \cap \text{space } R) \ \partial M$
by (*simp-all add: A-in-N emeasure-distr emeasure-join assms*)

also have $\dots = \int^+ x. \text{emeasure } (\text{distr } x \ N \ f) \ A \ \partial M$ **using** $A\text{-in-}N$
proof (*intro nn-integral-cong, subst emeasure-distr*)
fix M' **assume** $M' \in \text{space } M$
from *assms* **have** $\text{space } M = \text{space } (\text{subprob-algebra } R)$
using *sets-eq-imp-space-eq* **by** *blast*
with $\langle M' \in \text{space } M \rangle$ **have** $[simp]: \text{sets } M' = \text{sets } R$ **using** *space-subprob-algebra*
by *blast*
show $f \in \text{measurable } M' \ N$ **by** (*simp cong: measurable-cong-sets add: assms*)
have $\text{space } M' = \text{space } R$ **by** (*rule sets-eq-imp-space-eq*) *simp*
thus $\text{emeasure } M' (f - 'A \cap \text{space } R) = \text{emeasure } M' (f - 'A \cap \text{space } M')$ **by** *simp*
qed

also have $(\lambda M. \text{distr } M \ N \ f) \in \text{measurable } M \ (\text{subprob-algebra } N)$
by (*simp cong: measurable-cong-sets add: assms measurable-distr*)
hence $(\int^+ x. \text{emeasure } (\text{distr } x \ N \ f) \ A \ \partial M) =$
 $\text{emeasure } (\text{join } (\text{distr } M \ (\text{subprob-algebra } N) \ (\lambda M. \text{distr } M \ N \ f))) \ A$
by (*simp-all add: emeasure-join assms A-in-N nn-integral-distr measurable-emeasure-subprob-algebra*)
finally show $\text{emeasure } ?r \ A = \text{emeasure } ?l \ A$ **..**
qed *simp*

definition $\text{bind} :: 'a \text{ measure} \Rightarrow ('a \Rightarrow 'b \text{ measure}) \Rightarrow 'b \text{ measure}$ **where**
 $\text{bind } M \ f = (\text{if } \text{space } M = \{\} \text{ then } \text{count-space } \{\} \text{ else}$
 $\text{join } (\text{distr } M \ (\text{subprob-algebra } (f \ (\text{SOME } x. x \in \text{space } M))) \ f))$

adhoc-overloading $\text{Monad-Syntax.bind} \rightleftharpoons \text{bind}$

lemma *bind-empty*:
 $\text{space } M = \{\} \implies \text{bind } M \ f = \text{count-space } \{\}$
by (*simp add: bind-def*)

lemma *bind-nonempty*:

$space\ M \neq \{\} \implies bind\ M\ f = join\ (distr\ M\ (subprob\text{-}algebra\ (f\ (SOME\ x.\ x \in space\ M))))\ f)$

by (*simp add: bind-def*)

lemma *sets-bind-empty*: $sets\ M = \{\} \implies sets\ (bind\ M\ f) = \{\{\}\}$

by *auto*

lemma *space-bind-empty*: $space\ M = \{\} \implies space\ (bind\ M\ f) = \{\}$

by (*simp add: bind-def*)

lemma *sets-bind[*simp, measurable-cong*]*:

assumes $f: \bigwedge x. x \in space\ M \implies sets\ (f\ x) = sets\ N$ **and** $M: space\ M \neq \{\}$

shows $sets\ (bind\ M\ f) = sets\ N$

using $f\ [of\ SOME\ x.\ x \in space\ M]$ **by** (*simp add: bind-nonempty M some-in-eq*)

lemma *space-bind[*simp*]*:

assumes $\bigwedge x. x \in space\ M \implies sets\ (f\ x) = sets\ N$ **and** $space\ M \neq \{\}$

shows $space\ (bind\ M\ f) = space\ N$

using *assms* **by** (*intro sets-eq-imp-space-eq sets-bind*)

lemma *bind-cong-All*:

assumes $\forall x \in space\ M. f\ x = g\ x$

shows $bind\ M\ f = bind\ M\ g$

proof (*cases space M = {}*)

assume $space\ M \neq \{\}$

hence $(SOME\ x.\ x \in space\ M) \in space\ M$ **by** (*rule-tac someI-ex*) *blast*

with *assms* **have** $f\ (SOME\ x.\ x \in space\ M) = g\ (SOME\ x.\ x \in space\ M)$ **by** *blast*

with $\langle space\ M \neq \{\} \rangle$ **and** *assms* **show** *?thesis* **by** (*simp add: bind-nonempty cong: distr-cong*)

qed (*simp add: bind-empty*)

lemma *bind-cong*:

$M = N \implies (\bigwedge x. x \in space\ M \implies f\ x = g\ x) \implies bind\ M\ f = bind\ N\ g$

using *bind-cong-All* [*of M f g*] **by** *auto*

lemma *bind-nonempty'*:

assumes $f \in measurable\ M\ (subprob\text{-}algebra\ N)\ x \in space\ M$

shows $bind\ M\ f = join\ (distr\ M\ (subprob\text{-}algebra\ N)\ f)$

proof –

have $join\ (distr\ M\ (subprob\text{-}algebra\ (f\ (SOME\ x.\ x \in space\ M))))\ f = join\ (distr\ M\ (subprob\text{-}algebra\ N)\ f)$

by (*metis assms someI-ex subprob-algebra-cong subprob-measurableD(2)*)

with *assms* **show** *?thesis*

by (*metis bind-nonempty empty-iff*)

qed

lemma *bind-nonempty''*:

assumes $f \in measurable\ M\ (subprob\text{-}algebra\ N)\ space\ M \neq \{\}$

shows $\text{bind } M f = \text{join } (\text{distr } M (\text{subprob-algebra } N) f)$
using *assms* **by** (*auto intro: bind-nonempty'*)

lemma *emeasure-bind*:

$\llbracket \text{space } M \neq \{\} ; f \in \text{measurable } M (\text{subprob-algebra } N); X \in \text{sets } N \rrbracket$
 $\implies \text{emeasure } (M \gg f) X = \int^+ x. \text{emeasure } (f x) X \partial M$

by (*simp add: bind-nonempty'' emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra*)

lemma *nn-integral-bind*:

assumes $f: f \in \text{borel-measurable } B$

assumes $N: N \in \text{measurable } M (\text{subprob-algebra } B)$

shows $(\int^+ x. f x \partial(M \gg N)) = (\int^+ x. \int^+ y. f y \partial N x \partial M)$

proof *cases*

assume $M: \text{space } M \neq \{\}$ **show** *?thesis*

unfolding *bind-nonempty''* [*OF* N M] *nn-integral-join* [*OF* f *sets-distr*]

by (*rule nn-integral-distr* [*OF* N])

(*simp add: f nn-integral-measurable-subprob-algebra*)

qed (*simp add: bind-empty space-empty* [*of* M] *nn-integral-count-space*)

lemma *AE-bind*:

assumes $N[\text{measurable}]: N \in \text{measurable } M (\text{subprob-algebra } B)$

assumes $P[\text{measurable}]: \text{Measurable.pred } B P$

shows $(\text{AE } x \text{ in } M \gg N. P x) \longleftrightarrow (\text{AE } x \text{ in } M. \text{AE } y \text{ in } N x. P y)$

proof *cases*

assume $M: \text{space } M = \{\}$ **show** *?thesis*

unfolding *bind-empty* [*OF* M] **unfolding** *space-empty* [*OF* M] **by** (*simp add: AE-count-space*)

next

assume $M: \text{space } M \neq \{\}$

note *sets-kernel* [*OF* N , *simp*]

have $*$: $(\int^+ x. \text{indicator } \{x. \neg P x\} x \partial(M \gg N)) = (\int^+ x. \text{indicator } \{x \in \text{space } B. \neg P x\} x \partial(M \gg N))$

by (*intro nn-integral-cong*) (*simp add: space-bind* [*OF* - M] *split: split-indicator*)

have $(\text{AE } x \text{ in } M \gg N. P x) \longleftrightarrow (\int^+ x. \text{integral}^N (N x) (\text{indicator } \{x \in \text{space } B. \neg P x\}) \partial M) = 0$

by (*simp add: AE-iff-nn-integral sets-bind* [*OF* - M] *space-bind* [*OF* - M] $*$ *nn-integral-bind* [**where** $B=B$]

del: nn-integral-indicator)

also have $\dots = (\text{AE } x \text{ in } M. \text{integral}^N (N x) (\text{indicator } \{x \in \text{space } B. \neg P x\})) = 0$

proof (*rule nn-integral-0-iff-AE*)

show $(\lambda x. \text{integral}^N (N x) (\text{indicator } \{x \in \text{space } B. \neg P x\})) \in \text{borel-measurable } M$

apply (*rule measurable-compose* [*OF* N *nn-integral-measurable-subprob-algebra*])
by *measurable*

qed

also have $\dots = (\text{AE } x \text{ in } M. \text{AE } y \text{ in } N x. P y)$

apply (*intro eventually-subst AE-I2*)

by (auto simp add: subprob-measurableD(1)[OF N] intro!: AE-iff-measurable[symmetric])
 finally show ?thesis .

qed

lemma measurable-bind':

assumes M1: $f \in \text{measurable } M \text{ (subprob-algebra } N)$ and

M2: $\text{case-prod } g \in \text{measurable } (M \otimes_M N) \text{ (subprob-algebra } R)$

shows $(\lambda x. \text{bind } (f x) (g x)) \in \text{measurable } M \text{ (subprob-algebra } R)$

proof (subst measurable-cong)

fix x assume x-in-M: $x \in \text{space } M$

with assms have space (f x) $\neq \{\}$

by (blast dest: subprob-space-kernel subprob-space.subprob-not-empty)

moreover from M2 x-in-M have $g x \in \text{measurable } (f x) \text{ (subprob-algebra } R)$

by (subst measurable-cong-sets[OF sets-kernel[OF M1 x-in-M] refl])

(auto dest: measurable-Pair2)

ultimately show $\text{bind } (f x) (g x) = \text{join } (\text{distr } (f x) \text{ (subprob-algebra } R) (g x))$

by (simp-all add: bind-nonempty')

next

show $(\lambda w. \text{join } (\text{distr } (f w) \text{ (subprob-algebra } R) (g w))) \in \text{measurable } M \text{ (subprob-algebra } R)$

apply (rule measurable-compose[OF - measurable-join])

apply (rule measurable-distr2[OF M2 M1])

done

qed

lemma measurable-bind[measurable (raw)]:

assumes M1: $f \in \text{measurable } M \text{ (subprob-algebra } N)$ and

M2: $(\lambda x. g \text{ (fst } x) \text{ (snd } x)) \in \text{measurable } (M \otimes_M N) \text{ (subprob-algebra } R)$

shows $(\lambda x. \text{bind } (f x) (g x)) \in \text{measurable } M \text{ (subprob-algebra } R)$

using assms by (auto intro: measurable-bind' simp: measurable-split-conv)

lemma measurable-bind2:

assumes $f \in \text{measurable } M \text{ (subprob-algebra } N)$ and $g \in \text{measurable } N \text{ (subprob-algebra } R)$

shows $(\lambda x. \text{bind } (f x) g) \in \text{measurable } M \text{ (subprob-algebra } R)$

using assms by (intro measurable-bind' measurable-const) auto

lemma subprob-space-bind:

assumes subprob-space M $f \in \text{measurable } M \text{ (subprob-algebra } N)$

shows subprob-space (M \ggg f)

proof (rule subprob-space-kernel[of $\lambda x. x \ggg f$])

show $(\lambda x. x \ggg f) \in \text{measurable } (\text{subprob-algebra } M) \text{ (subprob-algebra } N)$

by (rule measurable-bind, rule measurable-ident-sets, rule refl,

rule measurable-compose[OF measurable-snd assms(2)])

from assms(1) show M $\in \text{space } (\text{subprob-algebra } M)$

by (simp add: space-subprob-algebra)

qed

lemma

```

fixes  $f :: - \Rightarrow \text{real}$ 
assumes  $f\text{-measurable}$  [ $\text{measurable}$ ]:  $f \in \text{borel-measurable } K$ 
and  $f\text{-bounded}$ :  $\bigwedge x. x \in \text{space } K \implies |f\ x| \leq B$ 
and  $N$  [ $\text{measurable}$ ]:  $N \in \text{measurable } M$  ( $\text{subprob-algebra } K$ )
and  $\text{fin}$ :  $\text{finite-measure } M$ 
and  $M\text{-bounded}$ :  $\text{AE } x \text{ in } M. \text{emeasure } (N\ x) (\text{space } (N\ x)) \leq \text{ennreal } B'$ 
shows  $\text{integrable-bind}$ :  $\text{integrable } (\text{bind } M\ N) f$  (is  $? \text{integrable}$ )
and  $\text{integral-bind}$ :  $\text{integral}^L (\text{bind } M\ N) f = \int x. \text{integral}^L (N\ x) f\ \partial M$  (is
 $? \text{integral}$ )
proof( $\text{case-tac } [!]\ \text{space } M = \{\}$ )
assume [ $\text{simp}$ ]:  $\text{space } M \neq \{\}$ 
interpret  $\text{finite-measure } M$  by( $\text{rule fin}$ )

have  $\text{integrable } (\text{join } (\text{distr } M (\text{subprob-algebra } K) N)) f$ 
using  $f\text{-measurable } f\text{-bounded}$ 
by( $\text{rule integrable-join}[\text{where } B'=B'](\text{simp-all add: finite-measure-distr AE-distr-iff } M\text{-bounded})$ )
then show  $? \text{integrable}$  by( $\text{simp add: bind-nonempty}''[\text{where } N=K]$ )

have  $\text{integral}^L (\text{join } (\text{distr } M (\text{subprob-algebra } K) N)) f = \int M'. \text{integral}^L M'$ 
 $f\ \partial \text{distr } M (\text{subprob-algebra } K) N$ 
using  $f\text{-measurable } f\text{-bounded}$ 
by( $\text{rule integral-join}[\text{where } B'=B'](\text{simp-all add: finite-measure-distr AE-distr-iff } M\text{-bounded})$ )
also have  $\dots = \int x. \text{integral}^L (N\ x) f\ \partial M$ 
by( $\text{rule integral-distr}(\text{simp-all add: integral-measurable-subprob-algebra}[OF\ -])$ )
finally show  $? \text{integral}$  by( $\text{simp add: bind-nonempty}''[\text{where } N=K]$ )
qed( $\text{simp-all add: bind-def integrable-count-space lebesgue-integral-count-space-finite Bochner-Integration.integral-empty}$ )

lemma (in  $\text{prob-space}$ )  $\text{prob-space-bind}$ :
assumes  $\text{ae}$ :  $\text{AE } x \text{ in } M. \text{prob-space } (N\ x)$ 
and  $N$  [ $\text{measurable}$ ]:  $N \in \text{measurable } M$  ( $\text{subprob-algebra } S$ )
shows  $\text{prob-space } (M \gg N)$ 
proof
have  $\text{emeasure } (M \gg N) (\text{space } (M \gg N)) = (\int^+ x. \text{emeasure } (N\ x) (\text{space } (N\ x))\ \partial M)$ 
by ( $\text{subst emeasure-bind}[\text{where } N=S]$ )
 $(\text{auto simp: not-empty space-bind}[OF\ \text{sets-kernel}]\ \text{subprob-measurableD}[OF\ N]\ \text{intro!: nn-integral-cong})$ 
also have  $\dots = (\int^+ x. 1\ \partial M)$ 
using  $\text{ae}$  by ( $\text{intro nn-integral-cong-AE, eventually-elim}$ ) ( $\text{rule prob-space.emeasure-space-1}$ )
finally show  $\text{emeasure } (M \gg N) (\text{space } (M \gg N)) = 1$ 
by ( $\text{simp add: emeasure-space-1}$ )
qed

lemma (in  $\text{subprob-space}$ )  $\text{bind-in-space}$ :
 $A \in \text{measurable } M$  ( $\text{subprob-algebra } N$ )  $\implies (M \gg A) \in \text{space } (\text{subprob-algebra } N)$ 

```

by (auto simp add: space-subprob-algebra subprob-not-empty sets-kernel intro!: subprob-space-bind)
 unfold-locales

lemma (in subprob-space) measure-bind:

assumes $f: f \in \text{measurable } M \text{ (subprob-algebra } N)$ and $X: X \in \text{sets } N$
 shows $\text{measure } (M \gg f) X = \int x. \text{measure } (f x) X \partial M$

proof –

interpret $Mf: \text{subprob-space } M \gg f$
 by (rule subprob-space-bind[OF - f]) unfold-locales

{ fix x assume $x \in \text{space } M$
 from $f[THEN \text{measurable-space, OF this}]$ interpret $\text{subprob-space } f x$
 by (simp add: space-subprob-algebra)
 have $\text{emeasure } (f x) X = \text{ennreal } (\text{measure } (f x) X) \text{ measure } (f x) X \leq 1$
 by (auto simp: emeasure-eq-measure subprob-measure-le-1) }
 note this[simp]

have $\text{emeasure } (M \gg f) X = \int^+ x. \text{emeasure } (f x) X \partial M$
 using subprob-not-empty $f X$ by (rule emeasure-bind)
 also have $\dots = \int^+ x. \text{ennreal } (\text{measure } (f x) X) \partial M$
 by (intro nn-integral-cong) simp
 also have $\dots = \int x. \text{measure } (f x) X \partial M$
 by (intro nn-integral-eq-integral integrable-const-bound[where $B=1$]
 measure-measurable-subprob-algebra2[OF - f] pair-measureI X)
 (auto simp: measure-nonneg)
 finally show ?thesis
 by (simp add: $Mf.\text{emeasure-eq-measure measure-nonneg integral-nonneg}$)

qed

lemma emeasure-bind-const:

$\text{space } M \neq \{\} \implies X \in \text{sets } N \implies \text{subprob-space } N \implies$
 $\text{emeasure } (M \gg (\lambda x. N)) X = \text{emeasure } N X * \text{emeasure } M (\text{space } M)$
 by (simp add: bind-nonempty emeasure-join nn-integral-distr
 space-subprob-algebra measurable-emeasure-subprob-algebra)

lemma emeasure-bind-const':

assumes $\text{subprob-space } M \text{ subprob-space } N$
 shows $\text{emeasure } (M \gg (\lambda x. N)) X = \text{emeasure } N X * \text{emeasure } M (\text{space } M)$
 using assms
proof (case-tac $X \in \text{sets } N$)
 fix X assume $X \in \text{sets } N$
 thus $\text{emeasure } (M \gg (\lambda x. N)) X = \text{emeasure } N X * \text{emeasure } M (\text{space } M)$
 using assms
 by (subst emeasure-bind-const)
 (simp-all add: subprob-space.subprob-not-empty subprob-space.emeasure-space-le-1)
 next
 fix X assume $X \notin \text{sets } N$
 with assms show $\text{emeasure } (M \gg (\lambda x. N)) X = \text{emeasure } N X * \text{emeasure } M$

(*space M*)
 by (*simp add: sets-bind[of - - N] subprob-space.subprob-not-empty*
space-subprob-algebra emeasure-notin-sets)
 qed

lemma *emeasure-bind-const-prob-space*:
 assumes *prob-space M subprob-space N*
 shows *emeasure (M \gg ($\lambda x. N$)) X = emeasure N X*
 using *assms* by (*simp add: emeasure-bind-const' prob-space-imp-subprob-space*
prob-space.emeasure-space-1)

lemma *bind-return*:
 assumes *f \in measurable M (subprob-algebra N) and $x \in$ space M*
 shows *bind (return M x) f = f x*
 using *sets-kernel[OF assms] assms*
 by (*simp-all add: distr-return join-return subprob-space-kernel bind-nonempty'*
cong: subprob-algebra-cong)

lemma *bind-return'*:
 shows *bind M (return M) = M*
 by (*cases space M = {}*)
 (*simp-all add: bind-empty space-empty[symmetric] bind-nonempty join-return'*
cong: subprob-algebra-cong)

lemma *distr-bind*:
 assumes *N: N \in measurable M (subprob-algebra K) space M \neq {}*
 assumes *f: f \in measurable K R*
 shows *distr (M \gg N) R f = (M \gg ($\lambda x. distr (N x) R f$))*
proof –
 have *distr (join (distr M (subprob-algebra K) N)) R f =*
join (distr M (subprob-algebra R) ($\lambda x. distr (N x) R f$))
 by (*simp add: assms distr-distr[OF measurable-distr] comp-def flip: join-distr-distr*)
 with *assms* show ?thesis
 unfolding *bind-nonempty''[OF N]*
 by (*smt (verit) bind-nonempty sets-distr subprob-algebra-cong*)
 qed

lemma *bind-distr*:
 assumes *f[measurable]: f \in measurable M X*
 assumes *N[measurable]: N \in measurable X (subprob-algebra K) and space M \neq {}*
 shows *(distr M X f \gg N) = (M \gg ($\lambda x. N (f x)$))*
proof –
 have *space X \neq {} space M \neq {}*
 using *\langle space M \neq {} \rangle f[THEN measurable-space] by auto*
 then show ?thesis
 by (*simp add: bind-nonempty''[where N=K] distr-distr comp-def*)
 qed

lemma *bind-count-space-singleton*:

assumes *subprob-space* ($f\ x$)

shows *count-space* $\{x\} \ggg f = f\ x$

proof –

have $A: \bigwedge A. A \subseteq \{x\} \implies A = \{\} \vee A = \{x\}$ **by** *auto*

have *count-space* $\{x\} = \text{return } (\text{count-space } \{x\})\ x$

by (*intro measure-eqI*) (*auto dest: A*)

also have $\dots \ggg f = f\ x$

by (*subst bind-return[of - - f x]*) (*auto simp: space-subprob-algebra assms*)

finally show *?thesis* .

qed

lemma *restrict-space-bind*:

assumes $N: N \in \text{measurable } M \text{ (subprob-algebra } K)$

assumes *space* $M \neq \{\}$

assumes $X[\text{simp}]: X \in \text{sets } K\ X \neq \{\}$

shows *restrict-space* (*bind* $M\ N$) $X = \text{bind } M\ (\lambda x. \text{restrict-space } (N\ x)\ X)$

proof (*rule measure-eqI*)

note $N\text{-sets} = \text{sets-bind}[OF\ \text{sets-kernel}[OF\ N]\ \text{assms}(2),\ \text{simp}]$

note $N\text{-space} = \text{sets-eq-imp-space-eq}[OF\ N\text{-sets},\ \text{simp}]$

show *sets* (*restrict-space* (*bind* $M\ N$) X) = *sets* (*bind* $M\ (\lambda x. \text{restrict-space } (N\ x)\ X)$)

by (*simp add: sets-restrict-space assms(2) sets-bind[OF sets-kernel[OF restrict-space-measurable[OF assms(4,3,1)]]]*)

fix A **assume** $A \in \text{sets } (\text{restrict-space } (M \ggg N)\ X)$

with X **have** $A: A \in \text{sets } K\ A \subseteq X$

by (*auto simp: sets-restrict-space*)

then have *emeasure* (*restrict-space* ($M \ggg N$) X) $A = \text{emeasure } (M \ggg N)\ A$

by (*simp add: emeasure-restrict-space*)

also have $\dots = \int^+ x. \text{emeasure } (N\ x)\ A\ \partial M$

by (*metis* $\langle A \in \text{sets } K \rangle\ N\ \langle \text{space } M \neq \{\} \rangle\ \text{emeasure-bind}$)

also have $\dots = \int^+ x. \text{emeasure } (\text{restrict-space } (N\ x)\ X)\ A\ \partial M$

using A **assms** **by** (*smt (verit, best) emeasure-restrict-space nn-integral-cong sets.Int-space-eq2 subprob-measurableD(2)*)

also have $\dots = \text{emeasure } (M \ggg (\lambda x. \text{restrict-space } (N\ x)\ X))\ A$

using A **assms**

apply (*subst emeasure-bind[OF - restrict-space-measurable]*)

apply (*auto simp: sets-restrict-space*)

done

finally show *emeasure* (*restrict-space* ($M \ggg N$) X) $A = \text{emeasure } (M \ggg (\lambda x. \text{restrict-space } (N\ x)\ X))\ A$.

qed

lemma *bind-restrict-space*:

assumes $A: A \cap \text{space } M \neq \{\}\ A \cap \text{space } M \in \text{sets } M$

and $f: f \in \text{measurable } (\text{restrict-space } M\ A)\ (\text{subprob-algebra } N)$

shows *restrict-space* $M\ A \ggg f = M \ggg (\lambda x. \text{if } x \in A \text{ then } f\ x \text{ else null-measure } (f\ (\text{SOME } x. x \in A \wedge x \in \text{space } M)))$

(*is ?lhs = ?rhs is - = M \ggg ?f*)

proof –

let $?P = \lambda x. x \in A \wedge x \in \text{space } M$
let $?x = \text{Eps } ?P$
let $?c = \text{null-measure } (f ?x)$
from A **have** $?P ?x$ **by**–(*rule someI-ex, blast*)
hence $?x \in \text{space } (\text{restrict-space } M A)$ **by**(*simp add: space-restrict-space*)
with f **have** $f ?x \in \text{space } (\text{subprob-algebra } N)$ **by**(*rule measurable-space*)
hence $\text{sps: subprob-space } (f ?x)$
and $\text{sets: sets } (f ?x) = \text{sets } N$
by(*simp-all add: space-subprob-algebra*)
have $\text{space } (f ?x) \neq \{\}$ **using** sps **by**(*rule subprob-space.subprob-not-empty*)
moreover have $\text{sets } ?c = \text{sets } N$ **by**(*simp add: null-measure-def*)(*simp add: sets*)

ultimately have $c: ?c \in \text{space } (\text{subprob-algebra } N)$
by(*simp add: space-subprob-algebra subprob-space-null-measure*)
from $f A c$ **have** $f': ?f \in \text{measurable } M$ (*subprob-algebra N*)
by(*simp add: measurable-restrict-space-iff*)

from A **have** [*simp*]: $\text{space } M \neq \{\}$ **by** *blast*

have $?lhs = \text{join } (\text{distr } (\text{restrict-space } M A) (\text{subprob-algebra } N) f)$
using *assms* **by**(*simp add: space-restrict-space bind-nonempty''*)
also have $\dots = \text{join } (\text{distr } M (\text{subprob-algebra } N) ?f)$
by(*rule measure-eqI*)(*auto simp add: emeasure-join nn-integral-distr nn-integral-restrict-space*)
 $f f' A$ *intro: nn-integral-cong*
also have $\dots = ?rhs$ **using** f' **by**(*simp add: bind-nonempty''*)
finally show $?thesis$.

qed

lemma *bind-const'*: $\llbracket \text{prob-space } M; \text{subprob-space } N \rrbracket \implies M \gg= (\lambda x. N) = N$
by (*intro measure-eqI*)
(simp-all add: space-subprob-algebra prob-space.not-empty emeasure-bind-const-prob-space)

lemma *bind-return-distr*:

assumes $\text{space } M \neq \{\}$ $f \in \text{measurable } M N$
shows $\text{bind } M (\text{return } N \circ f) = \text{distr } M N f$

proof –

have $\text{bind } M (\text{return } N \circ f)$
 $= \text{join } (\text{distr } M (\text{subprob-algebra } (\text{return } N (f (\text{SOME } x. x \in \text{space } M))))$
 $(\text{return } N \circ f))$
by (*simp add: Giry-Monad.bind-def assms*)
also have $\dots = \text{join } (\text{distr } M (\text{subprob-algebra } N) (\text{return } N \circ f))$
by (*metis sets-return subprob-algebra-cong*)
also have $\dots = \text{distr } M N f$
by (*metis assms(2) distr-distr join-return' return-measurable sets-distr*)
finally show $?thesis$.

qed

lemma *bind-return-distr'*:

$\text{space } M \neq \{\} \implies f \in \text{measurable } M \ N \implies \text{bind } M \ (\lambda x. \text{return } N \ (f \ x)) = \text{distr } M \ N \ f$

using *bind-return-distr*[*of M f N*] **by** (*simp add: comp-def*)

lemma *bind-assoc*:

fixes $f :: 'a \Rightarrow 'b \text{ measure}$ **and** $g :: 'b \Rightarrow 'c \text{ measure}$

assumes $M1: f \in \text{measurable } M \ (\text{subprob-algebra } N)$ **and** $M2: g \in \text{measurable } N \ (\text{subprob-algebra } R)$

shows $\text{bind} \ (\text{bind } M \ f) \ g = \text{bind } M \ (\lambda x. \text{bind} \ (f \ x) \ g)$

proof (*cases space M = {}*)

assume [*simp*]: $\text{space } M \neq \{\}$

from *assms* **have** [*simp*]: $\text{space } N \neq \{\}$ $\text{space } R \neq \{\}$

by (*auto simp: measurable-empty-iff space-subprob-algebra-empty-iff*)

from *assms* **have** *sets-fx*[*simp*]: $\bigwedge x. x \in \text{space } M \implies \text{sets} \ (f \ x) = \text{sets} \ N$

by (*simp add: sets-kernel*)

have *ex-in*: $\bigwedge A. A \neq \{\} \implies \exists x. x \in A$ **by** *blast*

note *sets-some*[*simp*] = *sets-kernel*[*OF M1 someI-ex*[*OF ex-in*[*OF space M* $\neq \{\}$]]]]

$\text{sets-kernel}[\text{OF } M2 \text{ someI-ex}[\text{OF ex-in}[\text{OF space } N \neq \{\}]]]$

note *space-some*[*simp*] = *sets-eq-imp-space-eq*[*OF this(1)*] *sets-eq-imp-space-eq*[*OF this(2)*]

have $*$: $(\lambda x. \text{distr } x \ (\text{subprob-algebra } R) \ g) \circ f \in M \rightarrow_M \text{subprob-algebra} \ (\text{subprob-algebra } R)$

using $M1 \ M2 \text{ measurable-comp measurable-distr}$ **by** *blast*

have $\text{bind } M \ (\lambda x. \text{bind} \ (f \ x) \ g) =$

$\text{join} \ (\text{distr } M \ (\text{subprob-algebra } R) \ (\text{join} \circ (\lambda x. (\text{distr } x \ (\text{subprob-algebra } R) \ g)) \circ f))$

by (*simp add: sets-eq-imp-space-eq*[*OF sets-fx*] *bind-nonempty o-def*

cong: subprob-algebra-cong distr-cong)

also have $\text{distr } M \ (\text{subprob-algebra } R) \ (\text{join} \circ (\lambda x. (\text{distr } x \ (\text{subprob-algebra } R) \ g)) \circ f) =$

$\text{distr} \ (\text{distr} \ (\text{distr } M \ (\text{subprob-algebra } N) \ f) \ (\text{subprob-algebra} \ (\text{subprob-algebra } R)))$
 $(\lambda x. \text{distr } x \ (\text{subprob-algebra } R) \ g))$

$(\text{subprob-algebra } R) \ \text{join}$

by (*simp add: distr-distr M1 M2 measurable-distr measurable-join fun.map-comp* *)

also have $\text{join} \ \dots = \text{bind} \ (\text{bind } M \ f) \ g$

by (*simp add: join-assoc join-distr-distr M2 bind-nonempty cong: subprob-algebra-cong*)

finally show *?thesis* **..**

qed (*simp add: bind-empty*)

lemma *double-bind-assoc*:

assumes $Mg: g \in \text{measurable } N \ (\text{subprob-algebra } N')$

assumes $Mf: f \in \text{measurable } M \ (\text{subprob-algebra } M')$

assumes $Mh: \text{case-prod } h \in \text{measurable} \ (M \otimes_M M') \ N$

shows $\text{do} \ \{x \leftarrow M; y \leftarrow f \ x; g \ (h \ x \ y)\} = \text{do} \ \{x \leftarrow M; y \leftarrow f \ x; \text{return } N \ (h \ x$

$y)\} \ggg g$

proof –

have $do \{x \leftarrow M; y \leftarrow f x; return N (h x y)\} \ggg g =$
 $do \{x \leftarrow M; do \{y \leftarrow f x; return N (h x y)\} \ggg g\}$
using Mh **by** $(auto intro! : bind-assoc measurable-bind' [OF Mf] Mf Mg$
 $measurable-compose [OF - return-measurable] simp : measur-$
 $able-split-conv)$
also from Mh **have** $\bigwedge x. x \in space M \implies h x \in measurable M' N$ **by** $measurable$
hence $do \{x \leftarrow M; do \{y \leftarrow f x; return N (h x y)\} \ggg g\} =$
 $do \{x \leftarrow M; y \leftarrow f x; return N (h x y) \ggg g\}$
apply $(intro ballI bind-cong refl bind-assoc)$
apply $(subst measurable-cong-sets [OF sets-kernel [OF Mf] refl], simp)$
apply $(rule measurable-compose [OF - return-measurable], auto intro : Mg)$
done
also have $\bigwedge x. x \in space M \implies space (f x) = space M'$
by $(intro sets-eq-imp-space-eq sets-kernel [OF Mf])$
with $measurable-space [OF Mh]$
have $do \{x \leftarrow M; y \leftarrow f x; return N (h x y) \ggg g\} = do \{x \leftarrow M; y \leftarrow f x; g$
 $(h x y)\}$
by $(intro ballI bind-cong bind-return [OF Mg]) (auto simp : space-pair-measure)$
finally show $?thesis ..$
qed

lemma **(in** $prob-space$) M -in-subprob[measurable (raw)]: $M \in space (subprob-algebra M)$

by $(simp add : space-subprob-algebra) unfold-locales$

lemma **(in** $pair-prob-space$) $pair-measure-eq-bind$:

$(M1 \otimes_M M2) = (M1 \ggg (\lambda x. M2 \ggg (\lambda y. return (M1 \otimes_M M2) (x, y))))$

proof $(rule measure-eqI)$

have $ps-M2 : prob-space M2$ **by** $unfold-locales$
note $return-measurable [measurable]$
show $sets (M1 \otimes_M M2) = sets (M1 \ggg (\lambda x. M2 \ggg (\lambda y. return (M1 \otimes_M M2) (x, y))))$
by $(simp-all add : M1.not-empty M2.not-empty)$
fix A **assume** $[measurable] : A \in sets (M1 \otimes_M M2)$
show $emeasure (M1 \otimes_M M2) A = emeasure (M1 \ggg (\lambda x. M2 \ggg (\lambda y. return (M1 \otimes_M M2) (x, y)))) A$
by $(auto simp : M2.emeasure-pair-measure M1.not-empty M2.not-empty emea-$
 $sure-bind [where N=M1 \otimes_M M2]$
 $intro! : nn-integral-cong)$
qed

lemma **(in** $pair-prob-space$) $bind-rotate$:

assumes $C [measurable] : (\lambda (x, y). C x y) \in measurable (M1 \otimes_M M2) (subprob-algebra N)$

shows $(M1 \ggg (\lambda x. M2 \ggg (\lambda y. C x y))) = (M2 \ggg (\lambda y. M1 \ggg (\lambda x. C x y)))$

proof –

interpret *swap*: *pair-prob-space* *M2* *M1* **by** *unfold-locales*
note *measurable-bind*[**where** $N=M2$, *measurable*]
note *measurable-bind*[**where** $N=M1$, *measurable*]
have [*simp*]: $M1 \in \text{space } (\text{subprob-algebra } M1)$ $M2 \in \text{space } (\text{subprob-algebra } M2)$
by (*auto simp*: *space-subprob-algebra*) *unfold-locales*
have $(M1 \gg (\lambda x. M2 \gg (\lambda y. C \ x \ y))) =$
 $(M1 \gg (\lambda x. M2 \gg (\lambda y. \text{return } (M1 \otimes_M M2) \ (x, y)))) \gg (\lambda(x, y). C \ x \ y)$
by (*auto intro!*: *bind-cong simp*: *bind-return*[**where** $N=N$] *space-pair-measure*
bind-assoc[**where** $N=M1 \otimes_M M2$ **and** $R=N$])
also have $\dots = (\text{distr } (M2 \otimes_M M1) \ (M1 \otimes_M M2) \ (\lambda(x, y). (y, x))) \gg$
 $(\lambda(x, y). C \ x \ y)$
unfolding *pair-measure-eq-bind*[*symmetric*] *distr-pair-swap*[*symmetric*] ..
also have $\dots = (M2 \gg (\lambda x. M1 \gg (\lambda y. \text{return } (M2 \otimes_M M1) \ (x, y)))) \gg$
 $(\lambda(y, x). C \ x \ y)$
unfolding *swap.pair-measure-eq-bind*[*symmetric*]
by (*auto simp add*: *space-pair-measure* *M1.not-empty* *M2.not-empty* *bind-distr*[*OF*
- *C*] *intro!*: *bind-cong*)
also have $\dots = (M2 \gg (\lambda y. M1 \gg (\lambda x. C \ x \ y)))$
by (*auto intro!*: *bind-cong simp*: *bind-return*[**where** $N=N$] *space-pair-measure*
bind-assoc[**where** $N=M2 \otimes_M M1$ **and** $R=N$])
finally show *?thesis* .
qed

lemma *bind-return''*: *sets* $M = \text{sets } N \implies M \gg \text{return } N = M$
by (*cases* *space* $M = \{\}$)
(simp-all add: bind-empty space-empty[*symmetric*] *bind-nonempty* *join-return'*
cong: subprob-algebra-cong)

lemma (**in** *prob-space*) *distr-const*[*simp*]:
 $c \in \text{space } N \implies \text{distr } M \ N \ (\lambda x. c) = \text{return } N \ c$
by (*rule* *measure-eqI*) (*auto simp*: *emeasure-distr* *emeasure-space-1*)

lemma *return-count-space-eq-density*:
 $\text{return } (\text{count-space } M) \ x = \text{density } (\text{count-space } M) \ (\text{indicator } \{x\})$
by (*rule* *measure-eqI*)
(auto simp: indicator-inter-arith[*symmetric*] *emeasure-density* *split: split-indicator*)

lemma *null-measure-in-space-subprob-algebra* [*simp*]:
 $\text{null-measure } M \in \text{space } (\text{subprob-algebra } M) \longleftrightarrow \text{space } M \neq \{\}$
by(*simp add: space-subprob-algebra subprob-space-null-measure-iff*)

7.4 Giry monad on probability spaces

definition *prob-algebra* $:: 'a \text{ measure} \Rightarrow 'a \text{ measure measure}$ **where**
prob-algebra $K = \text{restrict-space } (\text{subprob-algebra } K) \ \{M. \text{prob-space } M\}$

lemma *space-prob-algebra*: $\text{space } (\text{prob-algebra } M) = \{N. \text{sets } N = \text{sets } M \wedge \text{prob-space } N\}$
unfolding *prob-algebra-def* **by** (*auto simp*: *space-subprob-algebra* *space-restrict-space*)

prob-space-imp-subprob-space)

lemma *measurable-measure-prob-algebra*[*measurable*]:

$a \in \text{sets } A \implies (\lambda M. \text{Sigma-Algebra.measure } M \ a) \in \text{prob-algebra } A \rightarrow_M \text{borel}$

unfolding *prob-algebra-def* **by** (*intro measurable-restrict-space1 measurable-measure-subprob-algebra*)

lemma *measurable-prob-algebraD*:

$f \in N \rightarrow_M \text{prob-algebra } M \implies f \in N \rightarrow_M \text{subprob-algebra } M$

unfolding *prob-algebra-def measurable-restrict-space2-iff* **by** *auto*

lemma *measure-measurable-prob-algebra2*:

$\text{Sigma } (\text{space } M) \ A \in \text{sets } (M \otimes_M N) \implies L \in M \rightarrow_M \text{prob-algebra } N \implies$

$(\lambda x. \text{Sigma-Algebra.measure } (L \ x) \ (A \ x)) \in \text{borel-measurable } M$

using *measure-measurable-subprob-algebra2*[*of M A N L*] **by** (*auto intro: measurable-prob-algebraD*)

lemma *measurable-prob-algebraI*:

$(\bigwedge x. x \in \text{space } N \implies \text{prob-space } (f \ x)) \implies f \in N \rightarrow_M \text{subprob-algebra } M \implies$

$f \in N \rightarrow_M \text{prob-algebra } M$

unfolding *prob-algebra-def* **by** (*intro measurable-restrict-space2*) *auto*

lemma *measurable-distr-prob-space*:

assumes *f*: $f \in M \rightarrow_M N$

shows $(\lambda M'. \text{distr } M' \ N \ f) \in \text{prob-algebra } M \rightarrow_M \text{prob-algebra } N$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

proof (*intro conjI measurable-restrict-space1 measurable-distr f*)

show $(\lambda M'. \text{distr } M' \ N \ f) \in \text{space } (\text{restrict-space } (\text{subprob-algebra } M)) \ (\text{Collect prob-space}) \rightarrow \text{Collect prob-space}$

using *f* **by** (*auto simp: space-restrict-space space-subprob-algebra intro!: prob-space.prob-space-distr*)

qed

lemma *measurable-return-prob-space*[*measurable*]: *return* $N \in N \rightarrow_M \text{prob-algebra } N$

by (*rule measurable-prob-algebraI*) (*auto simp: prob-space-return*)

lemma *measurable-distr-prob-space2*[*measurable (raw)*]:

assumes *f*: $g \in L \rightarrow_M \text{prob-algebra } M \ (\lambda(x, y). f \ x \ y) \in L \otimes_M M \rightarrow_M N$

shows $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in L \rightarrow_M \text{prob-algebra } N$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

proof (*intro conjI measurable-restrict-space1 measurable-distr2*[**where** $M=M$] *f measurable-prob-algebraD*)

show $(\lambda x. \text{distr } (g \ x) \ N \ (f \ x)) \in \text{space } L \rightarrow \text{Collect prob-space}$

using *f* *subprob-measurableD*[*OF measurable-prob-algebraD*[*OF f*(1)]]

by (*auto simp: measurable-restrict-space2-iff prob-algebra-def intro!: prob-space.prob-space-distr*)

qed

lemma *measurable-bind-prob-space*:

assumes *f*: $f \in M \rightarrow_M \text{prob-algebra } N$ **and** *g*: $g \in N \rightarrow_M \text{prob-algebra } R$

shows $(\lambda x. \text{bind } (f x) g) \in M \rightarrow_M \text{prob-algebra } R$
unfolding *prob-algebra-def measurable-restrict-space2-iff*
proof (*intro conjI measurable-restrict-space1 measurable-bind2[where N=N] f g measurable-prob-algebraD*)
show $(\lambda x. f x \gg g) \in \text{space } M \rightarrow \text{Collect prob-space}$
using *g f subprob-measurableD[OF measurable-prob-algebraD[OF f]]*
by (*auto simp: measurable-restrict-space2-iff prob-algebra-def intro!: prob-space.prob-space-bind[where S=R] AE-I2*)
qed

lemma *measurable-bind-prob-space2[measurable (raw)]:*
assumes *f: f ∈ M →_M prob-algebra N and g: (λ(x, y). g x y) ∈ (M ⊗_M N) →_M prob-algebra R*
shows $(\lambda x. \text{bind } (f x) (g x)) \in M \rightarrow_M \text{prob-algebra } R$
unfolding *prob-algebra-def measurable-restrict-space2-iff*
proof (*intro conjI measurable-restrict-space1 measurable-bind[where N=N] f g measurable-prob-algebraD*)
show $(\lambda x. f x \gg g x) \in \text{space } M \rightarrow \text{Collect prob-space}$
using *g f subprob-measurableD[OF measurable-prob-algebraD[OF f]]*
using *measurable-space[OF g]*
by (*auto simp: measurable-restrict-space2-iff prob-algebra-def space-pair-measure Pi-iff intro!: prob-space.prob-space-bind[where S=R] AE-I2*)
qed (*use g in simp*)

lemma *measurable-prob-algebra-generated:*
assumes *eq: sets N = sigma-sets Ω G and Int-stable G G ⊆ Pow Ω*
assumes *subsp: ∧a. a ∈ space M ⇒ prob-space (K a)*
assumes *sets: ∧a. a ∈ space M ⇒ sets (K a) = sets N*
assumes $\bigwedge A. A \in G \Rightarrow (\lambda a. \text{emeasure } (K a) A) \in \text{borel-measurable } M$
shows $K \in \text{measurable } M \text{ (prob-algebra } N)$
unfolding *measurable-restrict-space2-iff prob-algebra-def*
proof
show $K \in M \rightarrow_M \text{subprob-algebra } N$
proof (*rule measurable-subprob-algebra-generated[OF assms(1,2,3) - assms(5,6)]*)
fix *a* **assume** *a ∈ space M* **then show** *subprob-space (K a)*
using *subsp[of a]* **by** (*intro prob-space-imp-subprob-space*)
next
have $(\lambda a. \text{emeasure } (K a) \Omega) \in \text{borel-measurable } M \longleftrightarrow (\lambda a. 1::\text{ennreal}) \in \text{borel-measurable } M$
using *sets-eq-imp-space-eq[of sigma Ω G N] ‹G ⊆ Pow Ω› eq sets-eq-imp-space-eq[OF sets]*
using *prob-space.emeasure-space-1[OF subsp]*
by (*intro measurable-cong auto*)
then show $(\lambda a. \text{emeasure } (K a) \Omega) \in \text{borel-measurable } M$ **by** *simp*
qed
qed (*use subsp in auto*)

lemma *in-space-prob-algebra*:

$x \in \text{space } (\text{prob-algebra } M) \implies \text{emeasure } x \text{ (space } M) = 1$

unfolding *prob-algebra-def space-restrict-space space-subprob-algebra*

by (*auto dest!:* *prob-space.emeasure-space-1 sets-eq-imp-space-eq*)

lemma *prob-space-pair*:

assumes *prob-space* M *prob-space* N **shows** *prob-space* $(M \otimes_M N)$

by (*metis assms measurable-fst prob-space.distr-pair-fst prob-space-distrD*)

lemma *measurable-pair-prob[measurable]*:

$f \in M \rightarrow_M \text{prob-algebra } N \implies g \in M \rightarrow_M \text{prob-algebra } L \implies (\lambda x. f x \otimes_M g x) \in M \rightarrow_M \text{prob-algebra } (N \otimes_M L)$

unfolding *prob-algebra-def measurable-restrict-space2-iff*

by (*auto intro!:* *measurable-pair-measure prob-space-pair*)

lemma *emeasure-bind-prob-algebra*:

assumes $A: A \in \text{space } (\text{prob-algebra } N)$

assumes $B: B \in N \rightarrow_M \text{prob-algebra } L$

assumes $X: X \in \text{sets } L$

shows $\text{emeasure } (\text{bind } A B) X = (\int^+ x. \text{emeasure } (B x) X \partial A)$

using $A B$

by (*intro emeasure-bind[OF - - X]*)

(*auto simp: space-prob-algebra measurable-prob-algebraD cong: measurable-cong-sets intro!:* *prob-space.not-empty*)

lemma *prob-space-bind'*:

assumes $A: A \in \text{space } (\text{prob-algebra } M)$ **and** $B: B \in M \rightarrow_M \text{prob-algebra } N$
shows *prob-space* $(A \gg B)$

using *measurable-bind-prob-space*[*OF measurable-const, OF A B, THEN measurable-space, of undefined count-space UNIV*]

by (*simp add: space-prob-algebra*)

lemma *sets-bind'*:

assumes $A: A \in \text{space } (\text{prob-algebra } M)$ **and** $B: B \in M \rightarrow_M \text{prob-algebra } N$
shows *sets* $(A \gg B) = \text{sets } N$

using *measurable-bind-prob-space*[*OF measurable-const, OF A B, THEN measurable-space, of undefined count-space UNIV*]

by (*simp add: space-prob-algebra*)

lemma *bind-cong-AE'*:

assumes $M: M \in \text{space } (\text{prob-algebra } L)$

and $f: f \in L \rightarrow_M \text{prob-algebra } N$ **and** $g: g \in L \rightarrow_M \text{prob-algebra } N$

and $ae: AE x \text{ in } M. f x = g x$

shows $\text{bind } M f = \text{bind } M g$

proof (*rule measure-eqI*)

show *sets* $(M \gg f) = \text{sets } (M \gg g)$

unfolding *sets-bind'*[*OF M f*] *sets-bind'*[*OF M g*] **..**

show $A \in \text{sets } (M \gg f) \implies \text{emeasure } (M \gg f) A = \text{emeasure } (M \gg g) A$
for A

unfolding *sets-bind'*[*OF M f*]
using *emeasure-bind-prob-algebra*[*OF M f, of A*] *emeasure-bind-prob-algebra*[*OF M g, of A*] *ae*
by (*auto intro: nn-integral-cong-AE*)
qed

lemma *density-discrete*:

countable A \implies *sets N = Set.Pow A* \implies ($\bigwedge x. f\ x \geq 0$) \implies ($\bigwedge x. x \in A \implies f\ x = \text{emeasure } N\ \{x\}$) \implies
density (count-space A) f = N
by (*rule measure-eqI-countable*[*of - A*]) (*auto simp: emeasure-density*)

lemma *distr-density-discrete*:

fixes *f'*
assumes *countable A*
assumes *f' ∈ borel-measurable M*
assumes *g ∈ measurable M (count-space A)*
defines *f* $\equiv \lambda x. \int^+ t. (\text{if } g\ t = x \text{ then } 1 \text{ else } 0) * f'\ t\ \partial M$
assumes $\bigwedge x. x \in \text{space } M \implies g\ x \in A$
shows *density (count-space A) (λx. f x) = distr (density M f') (count-space A) g*
proof (*rule density-discrete*)
fix *x* **assume** *x: x ∈ A*
have *f x = ∫⁺t. indicator (g - ' {x} ∩ space M) t * f' t ∂M* (**is** - = ?*I*)
unfolding *f-def*
by (*intro nn-integral-cong*) (*simp split: split-indicator*)
also from *x* **have** *in-sets: g - ' {x} ∩ space M ∈ sets M*
by (*intro measurable-sets*[*OF assms(3)*]) *simp*
have ?*I* = *emeasure (density M f') (g - ' {x} ∩ space M)* **unfolding** *f-def*
by (*subst emeasure-density*[*OF assms(2) in-sets*], *subst mult.commute*) (*rule refl*)
also from *assms(3) x* **have** ... = *emeasure (distr (density M f') (count-space A) g) {x}*
by (*subst emeasure-distr*) *simp-all*
finally show *f x = emeasure (distr (density M f') (count-space A) g) {x}* .
qed (*use assms in auto*)

lemma *bind-cong-AE*:

assumes *M = N*
assumes *f: f ∈ measurable N (subprob-algebra B)*
assumes *g: g ∈ measurable N (subprob-algebra B)*
assumes *ae: AE x in N. f x = g x*
shows *bind M f = bind N g*
proof *cases*
assume *space N = {}* **then show** ?*thesis*
using $\langle M = N \rangle$ **by** (*simp add: bind-empty*)
next
assume *space N ≠ {}*
show ?*thesis* **unfolding** $\langle M = N \rangle$
proof (*rule measure-eqI*)

```

have *: sets (N ≫ f) = sets B
  using sets-bind[OF sets-kernel[OF f] ⟨space N ≠ {}⟩] by simp
then show sets (N ≫ f) = sets (N ≫ g)
  using sets-bind[OF sets-kernel[OF g] ⟨space N ≠ {}⟩] by auto
fix A assume A ∈ sets (N ≫ f)
then have A ∈ sets B
  unfolding * .
with ae f g ⟨space N ≠ {}⟩ show emeasure (N ≫ f) A = emeasure (N ≫
g) A
  by (subst (1 2) emeasure-bind[where N=B]) (auto intro!: nn-integral-cong-AE)
qed
qed

```

```

lemma bind-cong-simp: M = N ⟹ (⋀x. x ∈ space M = simp => f x = g x) ⟹
bind M f = bind N g
  by (auto simp: simp-implies-def intro!: bind-cong)

```

```

lemma sets-bind-measurable:
assumes f: f ∈ measurable M (subprob-algebra B)
assumes M: space M ≠ {}
shows sets (M ≫ f) = sets B
  using M by (intro sets-bind[OF sets-kernel[OF f]]) auto

```

```

lemma space-bind-measurable:
assumes f: f ∈ measurable M (subprob-algebra B)
assumes M: space M ≠ {}
shows space (M ≫ f) = space B
  using M by (intro space-bind[OF sets-kernel[OF f]]) auto

```

```

lemma bind-distr-return:
f ∈ M →M N ⟹ g ∈ N →M L ⟹ space M ≠ {} ⟹
distr M N f ≫ (λx. return L (g x)) = distr M L (λx. g (f x))
  by (subst bind-distr[OF - measurable-compose[OF - return-measurable]])
    (auto intro!: bind-return-distr)

```

```

lemma (in prob-space) AE-eq-constD:
assumes AE x in M. x = y
shows M = return M y y ∈ space M
proof -
have AE x in M. x ∈ space M
  by auto
with asms have AE x in M. y ∈ space M
  by eventually-elim auto
thus y ∈ space M
  by simp
show M = return M y
proof (rule measure-eqI)
fix X assume X: X ∈ sets M
have AE x in M. (x ∈ X) = (x ∈ (if y ∈ X then space M else {}))

```

```

    using assms by eventually-elim (use  $X \langle y \in \text{space } M \rangle$  in auto)
  hence  $\text{emeasure } M \ X = \text{emeasure } M$  (if  $y \in X$  then  $\text{space } M$  else  $\{\}$ )
    using  $X$  by (intro  $\text{emeasure-eq-AE}$ ) auto
  also have  $\dots = \text{emeasure } (\text{return } M \ y) \ X$ 
    using  $X$  by (auto simp:  $\text{emeasure-space-1}$ )
  finally show  $\text{emeasure } M \ X = \dots$  .
qed auto
qed
end

```

8 Projective Family

```

theory Projective-Family
imports Giry-Monad
begin

```

lemma *vimage-restrict-preserve-mono*:

```

  assumes  $J: J \subseteq I$ 
  and sets:  $A \subseteq (\Pi_E \ i \in J. S \ i) \ B \subseteq (\Pi_E \ i \in J. S \ i)$  and  $ne: (\Pi_E \ i \in I. S \ i) \neq \{\}$ 
  and eq:  $(\lambda x. \text{restrict } x \ J) - 'A \cap (\Pi_E \ i \in I. S \ i) \subseteq (\lambda x. \text{restrict } x \ J) - 'B \cap (\Pi_E \ i \in I. S \ i)$ 
  shows  $A \subseteq B$ 
proof (intro subsetI)
  fix  $x$  assume  $x \in A$ 
  from  $ne$  obtain  $y$  where  $y: \bigwedge i. i \in I \implies y \ i \in S \ i$  by auto
  have  $J \cap (I - J) = \{\}$  by auto
  show  $x \in B$ 
proof cases
  assume  $x: x \in (\Pi_E \ i \in J. S \ i)$ 
  have  $\text{merge } J \ (I - J) \ (x, y) \in (\lambda x. \text{restrict } x \ J) - 'A \cap (\Pi_E \ i \in I. S \ i)$ 
    using  $y \ x \langle J \subseteq I \rangle \text{PiE-cancel-merge}[of \ J \ I - J \ x \ y \ S] \langle x \in A \rangle$ 
    by (auto simp del:  $\text{PiE-cancel-merge}$  simp add:  $\text{Un-absorb1}$ )
  also have  $\dots \subseteq (\lambda x. \text{restrict } x \ J) - 'B \cap (\Pi_E \ i \in I. S \ i)$  by fact
  finally show  $x \in B$ 
    using  $y \ x \langle J \subseteq I \rangle \text{PiE-cancel-merge}[of \ J \ I - J \ x \ y \ S] \langle x \in A \rangle \text{eq}$ 
    by (auto simp del:  $\text{PiE-cancel-merge}$  simp add:  $\text{Un-absorb1}$ )
qed (insert  $\langle x \in A \rangle$  sets, auto)
qed

```

locale *projective-family* =

```

  fixes  $I :: 'i \text{ set}$  and  $P :: 'i \text{ set} \implies ('i \implies 'a) \text{ measure}$  and  $M :: 'i \implies 'a \text{ measure}$ 
  assumes  $P: \bigwedge J \ H. J \subseteq H \implies \text{finite } H \implies H \subseteq I \implies P \ J = \text{distr } (P \ H) \ (P \ M \ J \ M) \ (\lambda f. \text{restrict } f \ J)$ 
  assumes  $\text{prob-space-}P: \bigwedge J. \text{finite } J \implies J \subseteq I \implies \text{prob-space } (P \ J)$ 
begin

```

lemma *sets-P*: $\text{finite } J \implies J \subseteq I \implies \text{sets } (P \ J) = \text{sets } (PiM \ J \ M)$
by (*subst P[of J J]*) *simp-all*

lemma *space-P*: $\text{finite } J \implies J \subseteq I \implies \text{space } (P \ J) = \text{space } (PiM \ J \ M)$
using *sets-P* **by** (*rule sets-eq-imp-space-eq*)

lemma *not-empty-M*: $i \in I \implies \text{space } (M \ i) \neq \{\}$
using *prob-space-P[THEN prob-space.not-empty]* **by** (*auto simp: space-PiM space-P*)

lemma *not-empty*: $\text{space } (PiM \ I \ M) \neq \{\}$
by (*simp add: not-empty-M*)

abbreviation

$\text{emb } L \ K \equiv \text{prod-emb } L \ M \ K$

lemma *emb-preserve-mono*:

assumes $J \subseteq L \subseteq I$ **and** $\text{sets } X \in \text{sets } (PiM \ J \ M) \ Y \in \text{sets } (PiM \ J \ M)$
assumes $\text{emb } L \ J \ X \subseteq \text{emb } L \ J \ Y$
shows $X \subseteq Y$

proof (*rule vimage-restrict-preserve-mono*)

show $X \subseteq (\Pi_E \ i \in J. \text{space } (M \ i)) \ Y \subseteq (\Pi_E \ i \in J. \text{space } (M \ i))$

using *sets[THEN sets.sets-into-space]* **by** (*auto simp: space-PiM*)

show $(\Pi_E \ i \in L. \text{space } (M \ i)) \neq \{\}$

using $\langle L \subseteq I \rangle$ **by** (*auto simp add: not-empty-M space-PiM[symmetric]*)

show $(\lambda x. \text{restrict } x \ J) - ' X \cap (\Pi_E \ i \in L. \text{space } (M \ i)) \subseteq (\lambda x. \text{restrict } x \ J) - ' Y$
 $\cap (\Pi_E \ i \in L. \text{space } (M \ i))$

using $\langle \text{prod-emb } L \ M \ J \ X \subseteq \text{prod-emb } L \ M \ J \ Y \rangle$ **by** (*simp add: prod-emb-def*)

qed fact

lemma *emb-injective*:

assumes $L: J \subseteq L \subseteq I$ **and** $X: X \in \text{sets } (PiM \ J \ M)$ **and** $Y: Y \in \text{sets } (PiM \ J \ M)$

shows $\text{emb } L \ J \ X = \text{emb } L \ J \ Y \implies X = Y$

by (*intro antisym emb-preserve-mono[OF L X Y] emb-preserve-mono[OF L Y X]*) *auto*

lemma *emeasure-P*: $J \subseteq K \implies \text{finite } K \implies K \subseteq I \implies X \in \text{sets } (PiM \ J \ M)$
 $\implies P \ K (\text{emb } K \ J \ X) = P \ J \ X$

by (*auto intro!: emeasure-distr-restrict[symmetric] simp: P sets-P*)

inductive-set *generator* :: $('i \Rightarrow 'a) \text{ set set}$ **where**

$\text{finite } J \implies J \subseteq I \implies X \in \text{sets } (PiM \ J \ M) \implies \text{emb } I \ J \ X \in \text{generator}$

lemma *algebra-generator*: $\text{algebra } (\text{space } (PiM \ I \ M)) \ \text{generator}$

unfolding *algebra-iff-Int*

proof (*safe elim!: generator.cases*)

fix $J \ X$ **assume** $J: \text{finite } J \subseteq I$ **and** $X: X \in \text{sets } (PiM \ J \ M)$

from $X[THEN \text{sets.sets-into-space}] \ J$ **show** $x \in \text{emb } I \ J \ X \implies x \in \text{space } (PiM \ I \ M)$

$I\ M)$ **for** x
 by (auto simp: prod-emb-def space-PiM)

 have $emb\ I\ J\ (space\ (PiM\ J\ M) - X) \in generator$
 by (intro generator.intros J sets.Diff sets.top X)
 with J show $space\ (Pi_M\ I\ M) - emb\ I\ J\ X \in generator$
 by (simp add: space-PiM prod-emb-PiE)

 fix $K\ Y$ **assume** K : finite $K\ K \subseteq I$ **and** Y : $Y \in sets\ (PiM\ K\ M)$
 have $emb\ I\ (J \cup K)\ (emb\ (J \cup K)\ J\ X) \cap emb\ I\ (J \cup K)\ (emb\ (J \cup K)\ K\ Y)$
 $\in generator$
 unfolding prod-emb-Int[symmetric]
 by (intro generator.intros sets.Int measurable-prod-emb) (auto intro!: J K X Y)
 with J K X Y show $emb\ I\ J\ X \cap emb\ I\ K\ Y \in generator$
 by simp
 qed (force simp: generator.simps prod-emb-empty[symmetric])

interpretation generator: algebra space (PiM I M) generator
 by (rule algebra-generator)

lemma sets-PiM-generator: $sets\ (PiM\ I\ M) = sigma\ sets\ (space\ (PiM\ I\ M))\ generator$
proof (intro antisym sets.sigma-sets-subset)
 show $sets\ (PiM\ I\ M) \subseteq sigma\ sets\ (space\ (PiM\ I\ M))\ generator$
 unfolding sets-PiM-single space-PiM[symmetric]
proof (intro sigma-sets-mono', safe)
 fix $i\ A$ **assume** $i \in I$ **and** A : $A \in sets\ (M\ i)$
 then have $\{f \in space\ (Pi_M\ I\ M). f\ i \in A\} = emb\ I\ \{i\}\ (\Pi_E\ j \in \{i\}. A)$
 by (auto simp: prod-emb-def space-PiM restrict-def Pi-iff PiE-iff extensional-def)
 with $\langle i \in I \rangle\ A$ show $\{f \in space\ (Pi_M\ I\ M). f\ i \in A\} \in generator$
 by (auto intro!: generator.intros sets-PiM-I-finite)
 qed
 qed (auto elim!: generator.cases)

definition mu-G ($\langle \mu G \rangle$) **where**
 $\mu G\ A = (THE\ x. \forall J \subseteq I. finite\ J \longrightarrow (\forall X \in sets\ (Pi_M\ J\ M). A = emb\ I\ J\ X \longrightarrow x = emeasure\ (P\ J)\ X))$

definition lim :: ($'i \Rightarrow 'a$) measure **where**
 $lim = extend\ measure\ (space\ (PiM\ I\ M))\ generator\ (\lambda x. x)\ \mu G$

lemma space-lim[simp]: $space\ lim = space\ (PiM\ I\ M)$
 using generator.space-closed
 unfolding lim-def **by** (intro space-extend-measure) simp

lemma sets-lim[simp, measurable]: $sets\ lim = sets\ (PiM\ I\ M)$
 using generator.space-closed **by** (simp add: lim-def sets-PiM-generator sets-extend-measure)

lemma *mu-G-spec*:

assumes J : *finite* J $J \subseteq I$ $X \in \text{sets}$ (Pi_M J M)
shows μG ($\text{emb } I$ J X) = emeasure (P J) X
unfolding *mu-G-def*
proof (*intro the-equality allI impI ballI*)
fix K Y **assume** K : *finite* K $K \subseteq I$ $Y \in \text{sets}$ (Pi_M K M)
and [*simp*]: $\text{emb } I$ J X = $\text{emb } I$ K Y
have emeasure (P K) Y = emeasure (P ($K \cup J$)) (emb ($K \cup J$) K Y)
using K J **by** (*simp add: emeasure-P*)
also have emb ($K \cup J$) K Y = emb ($K \cup J$) J X
using K J **by** (*simp add: emb-injective[of $K \cup J$ I]*)
also have emeasure (P ($K \cup J$)) (emb ($K \cup J$) J X) = emeasure (P J) X
using K J **by** (*subst emeasure-P*) *simp-all*
finally show emeasure (P J) X = emeasure (P K) Y ..
qed (*insert J , force*)

lemma *positive-mu-G*: *positive generator* μG

proof –
show ?thesis
proof (*safe intro!: positive-def[THEN iffD2]*)
obtain J **where** *finite* J $J \subseteq I$ **by** *auto*
then have μG ($\text{emb } I$ J $\{\}$) = 0
by (*subst mu-G-spec*) *auto*
then show μG $\{\}$ = 0 **by** *simp*
qed
qed

lemma *additive-mu-G*: *additive generator* μG

proof (*safe intro!: additive-def[THEN iffD2] elim!: generator.cases*)
fix J X K Y **assume** J : *finite* J $J \subseteq I$ $X \in \text{sets}$ (Pi_M J M)
and K : *finite* K $K \subseteq I$ $Y \in \text{sets}$ (Pi_M K M)
and $\text{emb } I$ J $X \cap \text{emb } I$ K Y = $\{\}$
then have $JK\text{-disj}$: emb ($J \cup K$) J $X \cap \text{emb}$ ($J \cup K$) K Y = $\{\}$
by (*intro emb-injective[of $J \cup K$ I - $\{\}$]*) (*auto simp: sets.Int prod-emb-Int*)
have μG ($\text{emb } I$ J $X \cup \text{emb } I$ K Y) = μG ($\text{emb } I$ ($J \cup K$) (emb ($J \cup K$) J X
 $\cup \text{emb}$ ($J \cup K$) K Y))
using J K **by** *simp*
also have ... = emeasure (P ($J \cup K$)) (emb ($J \cup K$) J $X \cup \text{emb}$ ($J \cup K$) K
 Y)
using J K **by** (*simp add: mu-G-spec sets.Un del: prod-emb-Un*)
also have ... = μG ($\text{emb } I$ J X) + μG ($\text{emb } I$ K Y)
using J K $JK\text{-disj}$ **by** (*simp add: plus-emeasure[symmetric] mu-G-spec emeasure-P sets-P*)
finally show μG ($\text{emb } I$ J $X \cup \text{emb } I$ K Y) = μG ($\text{emb } I$ J X) + μG ($\text{emb } I$ K Y) .
qed

lemma *emeasure-lim*:

assumes JX : *finite* J $J \subseteq I$ $X \in \text{sets}$ (Pi_M J M)

```

assumes cont:  $\bigwedge J X. (\bigwedge i. J i \subseteq I) \implies \text{incseq } J \implies (\bigwedge i. \text{finite } (J i)) \implies (\bigwedge i. X i \in \text{sets } (\text{PiM } (J i) M)) \implies$ 
 $\text{decseq } (\lambda i. \text{emb } I (J i) (X i)) \implies 0 < (\text{INF } i. P (J i) (X i)) \implies (\bigcap i. \text{emb } I (J i) (X i)) \neq \{\}$ 
shows emeasure lim (emb I J X) = P J X
proof –
  have  $\exists \mu. (\forall s \in \text{generator}. \mu s = \mu G s) \wedge$ 
 $\text{measure-space } (\text{space } (\text{PiM } I M)) (\text{sigma-sets } (\text{space } (\text{PiM } I M)) \text{ generator}) \mu$ 
  proof (rule generator.caratheodory-empty-continuous[OF positive-mu-G additive-mu-G])
  show  $\bigwedge A. A \in \text{generator} \implies \mu G A \neq \infty$ 
  proof (clarsimp elim!: generator.cases simp: mu-G-spec del: notI)
  fix J assume finite J  $J \subseteq I$ 
  then interpret prob-space P J by (rule prob-space-P)
  show  $\bigwedge X. X \in \text{sets } (\text{PiM } J M) \implies \text{emeasure } (P J) X \neq \text{top}$ 
  by simp
qed
next
  fix A assume range A  $\subseteq \text{generator}$  and decseq A  $(\bigcap i. A i) = \{\}$ 
  then have  $\forall i. \exists J X. A i = \text{emb } I J X \wedge \text{finite } J \wedge J \subseteq I \wedge X \in \text{sets } (\text{PiM } J M)$ 
  unfolding image-subset-iff generator.simps by blast
  then obtain J X where A:  $\bigwedge i. A i = \text{emb } I (J i) (X i)$ 
  and  $\ast: \bigwedge i. \text{finite } (J i) \wedge i. J i \subseteq I \wedge i. X i \in \text{sets } (\text{PiM } (J i) M)$ 
  by metis
  have  $(\text{INF } i. P (J i) (X i)) = 0$ 
  proof (rule ccontr)
  assume INF-P:  $(\text{INF } i. P (J i) (X i)) \neq 0$ 
  have  $(\bigcap i. \text{emb } I (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i))) \neq \{\}$ 
  proof (rule cont)
  show decseq  $(\lambda i. \text{emb } I (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i)))$ 
  using  $\ast$  decseq A by (subst prod-emb-trans) (auto simp: A[abs-def])
  show  $0 < (\text{INF } i. P (\bigcup n \leq i. J n) (\text{emb } (\bigcup n \leq i. J n) (J i) (X i)))$ 
  using  $\ast$  INF-P by (subst emeasure-P) (auto simp: less-le intro!: INF-greatest)
  show incseq  $(\lambda i. \bigcup n \leq i. J n)$ 
  by (force simp: incseq-def)
qed (insert  $\ast$ , auto)
  with  $\langle (\bigcap i. A i) = \{\} \rangle \ast$  show False
  by (subst (asm) prod-emb-trans) (auto simp: A[abs-def])
qed
moreover have  $(\lambda i. P (J i) (X i)) \longrightarrow (\text{INF } i. P (J i) (X i))$ 
proof (intro LIMSEQ-INF antimonoI)
  fix x y :: nat assume  $x \leq y$ 
  have  $P (J y \cup J x) (\text{emb } (J y \cup J x) (J y) (X y)) \leq P (J y \cup J x) (\text{emb } (J y \cup J x) (J x) (X x))$ 
  using  $\langle \text{decseq } A \rangle$  [THEN decseqD, OF  $\langle x \leq y \rangle$ ]  $\ast$ 
  by (auto simp: A sets-P del: subsetI intro!: emeasure-mono  $\langle x \leq y \rangle$  emb-preserve-mono[of J y  $\cup$  J x I, where X=emb (J y  $\cup$  J x) (J y)

```



```

(X y)])
  then show  $P (J y) (X y) \leq P (J x) (X x)$ 
    using * by (simp add: emeasure-P)
  qed
  ultimately show  $(\lambda i. \mu G (A i)) \longrightarrow 0$ 
    by (auto simp: A[abs-def] mu-G-spec *)
  qed
  then obtain  $\mu$  where eq:  $\forall s \in \text{generator}. \mu s = \mu G s$ 
    and ms: measure-space (space (PiM I M)) (sets (PiM I M))  $\mu$ 
    by (metis sets-PiM-generator)
  show ?thesis
  proof (subst emeasure-extend-measure[OF lim-def])
    show  $A \in \text{generator} \implies \mu A = \mu G A$  for A
      using eq by simp
    show positive (sets lim)  $\mu$  countably-additive (sets lim)  $\mu$ 
      using ms by (auto simp add: measure-space-def)
    show  $(\lambda x. x) \text{ 'generator} \subseteq \text{Pow} (\text{space} (PiM I M))$ 
      using generator.space-closed by simp
    show  $\text{emb } I J X \in \text{generator} \implies \mu G (\text{emb } I J X) = \text{emeasure } (P J) X$ 
      using JX by (auto intro: generator.intros simp: mu-G-spec)
  qed
qed
end

```

```

sublocale product-prob-space  $\subseteq$  projective-family I  $\lambda J. PiM J M M$ 
  unfolding projective-family-def
  proof (intro conjI allI impI distr-restrict)
    show  $\bigwedge J. \text{finite } J \implies \text{prob-space } (PiM J M)$ 
      by (intro prob-spaceI) (simp add: space-PiM emeasure-PiM emeasure-space-1)
  qed auto

```

Proof due to Ionescu Tulcea.

```

locale Ionescu-Tulcea =
  fixes  $P :: \text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow 'a \text{ measure}$  and  $M :: \text{nat} \Rightarrow 'a \text{ measure}$ 
  assumes P[measurable]:  $\bigwedge i. P i \in \text{measurable} (PiM \{0..<i\} M) (\text{subprob-algebra } (M i))$ 
  assumes prob-space-P:  $\bigwedge i x. x \in \text{space} (PiM \{0..<i\} M) \implies \text{prob-space } (P i x)$ 
begin

```

```

lemma non-empty[simp]:  $\text{space } (M i) \neq \{\}$ 
proof (induction i rule: less-induct)
  case (less i)
  then obtain x where  $\bigwedge j. j < i \implies x j \in \text{space } (M j)$ 
    unfolding ex-in-conv[symmetric] by metis
  then have *:  $\text{restrict } x \{0..<i\} \in \text{space} (PiM \{0..<i\} M)$ 
    by (auto simp: space-PiM PiE-iff)
  then interpret prob-space P i (restrict x {0..<i\})

```

by (rule prob-space-P)
 show ?case
 using not-empty subprob-measurableD(1)[OF P, OF *] by simp
 qed

lemma space-PiM-not-empty[simp]: space (PiM UNIV M) \neq {}
 unfolding space-PiM-empty-iff by auto

lemma space-P: $x \in \text{space } (PiM \{0..<n\} M) \implies \text{space } (P \ n \ x) = \text{space } (M \ n)$
 by (simp add: P[THEN subprob-measurableD(1)])

lemma sets-P[measurable-cong]: $x \in \text{space } (PiM \{0..<n\} M) \implies \text{sets } (P \ n \ x) = \text{sets } (M \ n)$
 by (simp add: P[THEN subprob-measurableD(2)])

definition eP :: nat \Rightarrow (nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) measure **where**
 $eP \ n \ \omega = \text{distr } (P \ n \ \omega) \ (PiM \{0..<Suc \ n\} M) \ (\text{fun-upd } \omega \ n)$

lemma measurable-eP[measurable]:
 $eP \ n \in \text{measurable } (PiM \{0..<n\} M) \ (\text{subprob-algebra } (PiM \{0..<Suc \ n\} M))$
 by (auto simp: eP-def[abs-def] measurable-split-conv
 intro!: measurable-fun-upd[where J={0..<n}] measurable-distr2[OF - P])

lemma space-eP:
 $x \in \text{space } (PiM \{0..<n\} M) \implies \text{space } (eP \ n \ x) = \text{space } (PiM \{0..<Suc \ n\} M)$
 by (simp add: eP-def)

lemma sets-eP[measurable]:
 $x \in \text{space } (PiM \{0..<n\} M) \implies \text{sets } (eP \ n \ x) = \text{sets } (PiM \{0..<Suc \ n\} M)$
 by (simp add: eP-def)

lemma prob-space-eP: $x \in \text{space } (PiM \{0..<n\} M) \implies \text{prob-space } (eP \ n \ x)$
 unfolding eP-def
 by (intro prob-space.prob-space-distr prob-space-P measurable-fun-upd[where J={0..<n}])
 auto

lemma nn-integral-eP:
 $\omega \in \text{space } (PiM \{0..<n\} M) \implies f \in \text{borel-measurable } (PiM \{0..<Suc \ n\} M)$
 \implies
 $(\int^{+x}. f \ x \ \partial eP \ n \ \omega) = (\int^{+x}. f \ (\text{fun-upd } \omega \ n \ x) \ \partial P \ n \ \omega)$
 unfolding eP-def
 by (subst nn-integral-distr) (auto intro!: measurable-fun-upd[where J={0..<n}])
 simp: space-PiM PiE-iff)

lemma emeasure-eP:
 assumes ω [simp]: $\omega \in \text{space } (PiM \{0..<n\} M)$ and A [measurable]: $A \in \text{sets } (PiM \{0..<Suc \ n\} M)$
 shows $eP \ n \ \omega \ A = P \ n \ \omega \ ((\lambda x. \text{fun-upd } \omega \ n \ x) -' A \cap \text{space } (M \ n))$
 using nn-integral-eP[of $\omega \ n$ indicator A]

```

apply (simp add: sets-eP nn-integral-indicator[symmetric] sets-P del: nn-integral-indicator)
apply (subst nn-integral-indicator[symmetric])
using measurable-sets[OF measurable-fun-upd[OF - measurable-const[OF  $\omega$ ] measurable-id] A, of n]
apply (auto simp add: sets-P atLeastLessThanSuc space-P simp del: nn-integral-indicator
  intro!: nn-integral-cong split: split-indicator)
done

```

```

primrec C :: nat  $\Rightarrow$  nat  $\Rightarrow$  (nat  $\Rightarrow$  'a)  $\Rightarrow$  (nat  $\Rightarrow$  'a) measure where
  C n 0  $\omega$  = return (PiM {0.. $n$ } M)  $\omega$ 
| C n (Suc m)  $\omega$  = C n m  $\omega$   $\ggg$  eP (n + m)

```

```

lemma measurable-C[measurable]:
  C n m  $\in$  measurable (PiM {0.. $n$ } M) (subprob-algebra (PiM {0.. $n$  + m} M))
by (induction m) auto

```

```

lemma space-C:
   $x \in \text{space } (\text{PiM } \{0.. $n$ \} M) \implies \text{space } (C\ n\ m\ x) = \text{space } (\text{PiM } \{0.. $n$  + m\} M)$ 
by (simp add: measurable-C[THEN subprob-measurableD(1)])

```

```

lemma sets-C[measurable-cong]:
   $x \in \text{space } (\text{PiM } \{0.. $n$ \} M) \implies \text{sets } (C\ n\ m\ x) = \text{sets } (\text{PiM } \{0.. $n$  + m\} M)$ 
by (simp add: measurable-C[THEN subprob-measurableD(2)])

```

```

lemma prob-space-C:  $x \in \text{space } (\text{PiM } \{0.. $n$ \} M) \implies \text{prob-space } (C\ n\ m\ x)$ 
proof (induction m)
  case (Suc m) then show ?case
    by (auto intro!: prob-space.prob-space-bind[where S=PiM {0.. $\text{Suc } (n + m)$ } M]
      simp: space-C prob-space-eP)
qed (auto intro!: prob-space-return simp: space-PiM)

```

```

lemma split-C:
  assumes  $\omega: \omega \in \text{space } (\text{PiM } \{0.. $n$ \} M)$  shows  $(C\ n\ m\ \omega \ggg C\ (n + m)\ l) = C\ n\ (m + l)\ \omega$ 
proof (induction l)
  case 0
    with  $\omega$  show ?case
    by (simp add: bind-return-distr' prob-space-C[THEN prob-space.not-empty]
      distr-cong[OF refl sets-C[symmetric, OF  $\omega$ ]])
  next
    case (Suc l) with  $\omega$  show ?case
    by (simp add: bind-assoc[symmetric, OF - measurable-eP]) (simp add: ac-simps)
qed

```

```

lemma nn-integral-C:
  assumes  $m \leq m'$  and  $f[\text{measurable}]: f \in \text{borel-measurable } (\text{PiM } \{0.. $n+m$ \})$ 

```

M)
and $\text{nonneg: } \bigwedge x. x \in \text{space } (PiM \{0..<n+m\} M) \implies 0 \leq f x$
and $x: x \in \text{space } (PiM \{0..<n\} M)$
shows $(\int^{+x}. f x \partial C n m x) = (\int^{+x}. f (restrict x \{0..<n+m\}) \partial C n m' x)$
using $\langle m \leq m' \rangle$
proof (*induction rule: dec-induct*)
case (*step i*)
let $?E = \lambda x. f (restrict x \{0..<n + m\})$ **and** $?C = \lambda i f. \int^{+x}. f x \partial C n i x$
from $\langle m \leq i \rangle x$ **have** $?C i ?E = ?C (Suc i) ?E$
by (*auto simp: nn-integral-bind[where B=PiM {0 ..< Suc (n + i)} M] space-C nn-integral-eP*
intro!: nn-integral-cong)
(simp add: space-PiM PiE-iff nonneg prob-space.emeasure-space-1[OF
prob-space-P])
with step show $?case$ **by** (*simp del: restrict-apply*)
qed (*auto simp: space-PiM space-C[OF x] simp del: restrict-apply intro!: nn-integral-cong*)

lemma *emeasure-C*:

assumes $m \leq m'$ **and** $A[\text{measurable}]$: $A \in \text{sets } (PiM \{0..<n+m\} M)$ **and** $[simp]$:
 $x \in \text{space } (PiM \{0..<n\} M)$
shows $\text{emeasure } (C n m' x) (\text{prod-emb } \{0..<n + m'\} M \{0..<n+m\} A) =$
 $\text{emeasure } (C n m x) A$
using *assms*
by (*subst (1 2) nn-integral-indicator[symmetric]*)
(auto intro!: nn-integral-cong split: split-indicator simp del: nn-integral-indicator
simp: nn-integral-C[of m m' - n] prod-emb-iff space-PiM PiE-iff sets-C
space-C)

lemma *distr-C*:

assumes $m \leq m'$ **and** $[simp]$: $x \in \text{space } (PiM \{0..<n\} M)$
shows $C n m x = \text{distr } (C n m' x) (PiM \{0..<n+m\} M) (\lambda x. \text{restrict } x \{0..<n+m\})$
proof (*rule measure-eqI*)
fix A **assume** $A \in \text{sets } (C n m x)$
with $\langle m \leq m' \rangle$ **show** $\text{emeasure } (C n m x) A = \text{emeasure } (\text{distr } (C n m' x) (PiM \{0..<n + m\} M) (\lambda x. \text{restrict } x \{0..<n + m\})) A$
by (*subst emeasure-C[symmetric, OF $\langle m \leq m' \rangle$] (auto intro!: emeasure-distr-restrict[symmetric]*
simp: sets-C)
qed (*simp add: sets-C*)

definition *up-to* :: $\text{nat set} \Rightarrow \text{nat}$ **where**

$\text{up-to } J = (\text{LEAST } n. \forall i \geq n. i \notin J)$

lemma *up-to-less*: $\text{finite } J \implies i \in J \implies i < \text{up-to } J$

unfolding *up-to-def*

by (*rule LeastI2[of - Suc (Max J)] (auto simp: Suc-le-eq not-le[symmetric])*)

lemma *up-to-iff*: $\text{finite } J \implies \text{up-to } J \leq n \iff (\forall i \in J. i < n)$

proof *safe*

show $finite\ J \implies up\text{-}to\ J \leq n \implies i \in J \implies i < n$ **for** i
using $up\text{-}to\text{-}less[of\ J\ i]$ **by** $auto$
qed ($auto\ simp: up\text{-}to\text{-}def\ intro!: Least\text{-}le$)

lemma $up\text{-}to\text{-}iff\text{-}Ico: finite\ J \implies up\text{-}to\ J \leq n \longleftrightarrow J \subseteq \{0..<n\}$
by ($auto\ simp: up\text{-}to\text{-}iff$)

lemma $up\text{-}to: finite\ J \implies J \subseteq \{0..< up\text{-}to\ J\}$
by ($auto\ simp: up\text{-}to\text{-}less$)

lemma $up\text{-}to\text{-}mono: J \subseteq H \implies finite\ H \implies up\text{-}to\ J \leq up\text{-}to\ H$
by ($auto\ simp\ add: up\text{-}to\text{-}iff\ finite\text{-}subset\ up\text{-}to\text{-}less$)

definition $CI :: nat\ set \Rightarrow (nat \Rightarrow 'a)\ measure$ **where**
 $CI\ J = distr\ (C\ 0\ (up\text{-}to\ J)\ (\lambda x. undefined))\ (PiM\ J\ M)\ (\lambda f. restrict\ f\ J)$

sublocale $PF: projective\text{-}family\ UNIV\ CI$

unfolding $projective\text{-}family\text{-}def$

proof $safe$

show $finite\ J \implies prob\text{-}space\ (CI\ J)$ **for** J
using $up\text{-}to[of\ J]$ **unfolding** $CI\text{-}def$
by ($intro\ prob\text{-}space.\ prob\text{-}space\text{-}distr\ prob\text{-}space\text{-}C\ measurable\text{-}restrict$) $auto$
note $measurable\text{-}cong\text{-}sets[OF\ sets\text{-}C,\ simp]$
have $[simp]: J \subseteq H \implies (\lambda f. restrict\ f\ J) \in measurable\ (PiM\ H\ M)\ (PiM\ J\ M)$
for $H\ J$
by ($auto\ intro!: measurable\text{-}restrict$)

show $J \subseteq H \implies finite\ H \implies CI\ J = distr\ (CI\ H)\ (PiM\ J\ M)\ (\lambda f. restrict\ f\ J)$ **for** $J\ H$

by ($simp\ add: CI\text{-}def\ distr\text{-}C[OF\ up\text{-}to\text{-}mono[of\ J\ H]]\ up\text{-}to\ up\text{-}to\text{-}mono\ distr\text{-}distr\ comp\text{-}def$
 $inf.\ absorb2\ finite\text{-}subset$)

qed

lemma $emeasure\text{-}CI'$:

$finite\ J \implies X \in sets\ (PiM\ J\ M) \implies CI\ J\ X = C\ 0\ (up\text{-}to\ J)\ (\lambda\text{-}. undefined)$
 $(PF.\ emb\ \{0..<up\text{-}to\ J\}\ J\ X)$
unfolding $CI\text{-}def$ **using** $up\text{-}to[of\ J]$ **by** ($rule\ emeasure\text{-}distr\text{-}restrict$) ($auto\ simp: sets\text{-}C$)

lemma $emeasure\text{-}CI$:

$J \subseteq \{0..<n\} \implies X \in sets\ (PiM\ J\ M) \implies CI\ J\ X = C\ 0\ n\ (\lambda\text{-}. undefined)$
 $(PF.\ emb\ \{0..<n\}\ J\ X)$
apply ($subst\ emeasure\text{-}CI',\ simp\text{-}all\ add: finite\text{-}subset$)
apply ($subst\ emeasure\text{-}C[symmetric,\ of\ up\text{-}to\ J\ n]$)
apply ($auto\ simp: finite\text{-}subset\ up\text{-}to\text{-}iff\text{-}Ico\ up\text{-}to\text{-}less$)
apply ($subst\ prod\text{-}emb\text{-}trans$)
apply ($auto\ simp: up\text{-}to\text{-}less\ finite\text{-}subset\ up\text{-}to\text{-}iff\text{-}Ico$)
done

lemma *lim*:

```

assumes J: finite J and X:  $X \in \text{sets } (PiM \ J \ M)$ 
shows  $\text{emeasure } PF.\text{lim } (PF.\text{emb } UNIV \ J \ X) = \text{emeasure } (CI \ J) \ X$ 
proof (rule PF.emeasure-lim[OF J subset-UNIV X])
  fix J X' assume  $J[\text{simp}]: \bigwedge i. \text{finite } (J \ i) \text{ and } X'[\text{measurable}]: \bigwedge i. X' \ i \in \text{sets}$ 
  ( $PiM \ (J \ i) \ M$ )
  and dec:  $\text{decseq } (\lambda i. PF.\text{emb } UNIV \ (J \ i) \ (X' \ i))$ 
  define X where  $X \ n =$ 
    ( $\bigcap_{i \in \{i. J \ i \subseteq \{0..<n\}\}}. PF.\text{emb } \{0..<n\} \ (J \ i) \ (X' \ i)) \cap \text{space } (PiM \ \{0..<n\} \ M)$ 
  for n

  have  $\text{sets-}X[\text{measurable}]: X \ n \in \text{sets } (PiM \ \{0..<n\} \ M) \text{ for } n$ 
  by (cases  $\{i. J \ i \subseteq \{0..<n\}\} = \{\}$ )
    (simp-all add: X-def, auto intro!: sets.countable-INT' sets.Int)

  have dec-X:  $n \leq m \implies X \ m \subseteq PF.\text{emb } \{0..<m\} \ \{0..<n\} \ (X \ n) \text{ for } n \ m$ 
  unfolding X-def using ivl-subset[of 0 n 0 m]
  by (cases  $\{i. J \ i \subseteq \{0..<n\}\} = \{\}$ )
    (auto simp add: prod-emb-Int prod-emb-PiE space-PiM simp del: ivl-subset)

  have dec-X':  $PF.\text{emb } \{0..<n\} \ (J \ j) \ (X' \ j) \subseteq PF.\text{emb } \{0..<n\} \ (J \ i) \ (X' \ i)$ 
  if [simp]:  $J \ j \subseteq \{0..<n\} \ J \ j \subseteq \{0..<n\} \ i \leq j \text{ for } n \ i \ j$ 
  by (rule PF.emb-preserve-mono[of \{0..<n\} UNIV]) (auto del: subsetI intro: dec[THEN antimonoD])

  assume  $0 < (INF \ i. CI \ (J \ i) \ (X' \ i))$ 
  also have  $\dots \leq (INF \ i. C \ 0 \ i \ (\lambda x. \text{undefined}) \ (X \ i))$ 
  proof (intro INF-greatest)
    fix n
    interpret C: prob-space C 0 n ( $\lambda x. \text{undefined}$ )
    by (rule prob-space-C) simp
    show  $(INF \ i. CI \ (J \ i) \ (X' \ i)) \leq C \ 0 \ n \ (\lambda x. \text{undefined}) \ (X \ n)$ 
    proof cases
      assume  $\{i. J \ i \subseteq \{0..<n\}\} = \{\}$  with C.emeasure-space-1 show ?thesis
      by (auto simp add: X-def space-C intro!: INF-lower2[of 0] prob-space.measure-le-1 PF.prob-space-P)
    next
      assume  $\ast: \{i. J \ i \subseteq \{0..<n\}\} \neq \{\}$ 
      have  $(INF \ i. CI \ (J \ i) \ (X' \ i)) \leq$ 
        ( $INF \ i \in \{i. J \ i \subseteq \{0..<n\}\}. C \ 0 \ n \ (\lambda-. \text{undefined}) \ (PF.\text{emb } \{0..<n\} \ (J \ i) \ (X' \ i))$ )
      by (intro INF-superset-mono) (auto simp: emeasure-CI)
      also have  $\dots = C \ 0 \ n \ (\lambda-. \text{undefined}) \ (\bigcap_{i \in \{i. J \ i \subseteq \{0..<n\}\}}. (PF.\text{emb } \{0..<n\} \ (J \ i) \ (X' \ i)))$ 
      using  $\ast$  by (intro emeasure-INT-decseq-subset[symmetric]) (auto intro!: dec-X' del: subsetI simp: sets-C)
      also have  $\dots = C \ 0 \ n \ (\lambda-. \text{undefined}) \ (X \ n)$ 
      using  $\ast$  by (auto simp add: X-def INT-extend-simps)

```

```

    finally show (INF i. CI (J i) (X' i)) ≤ C 0 n (λ-. undefined) (X n) .
  qed
qed
finally have pos: 0 < (INF i. C 0 i (λx. undefined) (X i)) .
from less-INF-D[OF this, of 0] have X 0 ≠ {}
  by auto

{ fix ω n assume ω: ω ∈ space (PiM {0..<n} M)
  let ?C = λi. emeasure (C n i ω) (X (n + i))
  let ?C' = λi x. emeasure (C (Suc n) i (fun-upd ω n x)) (X (Suc n + i))
  have M: ∧i. ?C' i ∈ borel-measurable (P n ω)
    using ω[measurable, simp] measurable-fun-upd[where J={0..<n}] by mea-
surable auto

  assume 0 < (INF i. ?C i)
  also have ... ≤ (INF i. emeasure (C n (1 + i) ω) (X (n + (1 + i))))
    by (intro INF-greatest INF-lower) auto
  also have ... = (INF i. ∫+x. ?C' i x ∂P n ω)
    using ω measurable-C[of Suc n]
  apply (intro INF-cong refl)
  apply (subst split-C[symmetric, OF ω])
  apply (subst emeasure-bind[OF - - sets-X])
  apply (simp-all del: C.simps add: space-C)
  apply measurable
  apply simp
  apply (simp add: bind-return[OF measurable-eP] nn-integral-eP)
  done
  also have ... = (∫+x. (INF i. ?C' i x) ∂P n ω)
proof (rule nn-integral-monotone-convergence-INF-AE[symmetric])
  have (∫+x. ?C' 0 x ∂P n ω) ≤ (∫+x. 1 ∂P n ω)
    by (intro nn-integral-mono) (auto split: split-indicator)
  also have ... < ∞
    using prob-space-P[OF ω, THEN prob-space.emeasure-space-1] by simp
  finally show (∫+x. ?C' 0 x ∂P n ω) < ∞ .
next
show AE x in P n ω. ?C' (Suc i) x ≤ ?C' i x for i
proof (rule AE-I2)
  fix x assume x ∈ space (P n ω)
  with ω have ω': fun-upd ω n x ∈ space (PiM {0..<Suc n} M)
    by (auto simp: space-P[OF ω] space-PiM PiE-iff extensional-def)
  with ω show ?C' (Suc i) x ≤ ?C' i x
    apply (subst emeasure-C[symmetric, of i Suc i])
    apply (auto intro!: emeasure-mono[OF dec-X] del: subsetI
      simp: sets-C space-P)
    apply (subst sets-bind[OF sets-eP])
    apply (simp-all add: space-C space-P)
  done
qed
qed fact

```

```

finally have ( $\int^+ x. (INF\ i. ?C'\ i\ x)\ \partial P\ n\ \omega$ )  $\neq 0$ 
  by simp
with  $M$  have  $\exists_F\ x\ in\ ae\text{-}filter\ (P\ n\ \omega).\ 0 < (INF\ i. ?C'\ i\ x)$ 
  by (subst (asm) nn-integral-0-iff-AE)
    (auto intro!: borel-measurable-INF simp: Filter.not-eventually not-le
zero-less-iff-neq-zero)
then have  $\exists_F\ x\ in\ ae\text{-}filter\ (P\ n\ \omega). x \in space\ (M\ n) \wedge 0 < (INF\ i. ?C'\ i\ x)$ 
  by (rule frequently-mp[rotated]) (auto simp: space-P  $\omega$ )
then obtain  $x$  where  $x \in space\ (M\ n)\ 0 < (INF\ i. ?C'\ i\ x)$ 
  by (auto dest: frequently-ex)
from this(2)[THEN less-INF-D, of 0] this(2)
have  $\exists x. fun\text{-}upd\ \omega\ n\ x \in X\ (Suc\ n) \wedge 0 < (INF\ i. ?C'\ i\ x)$ 
  by (intro exI[of - x] (simp split: split-indicator-asm)) }
note step = this

let  $? \omega = \lambda \omega\ n\ x. (restrict\ \omega\ \{0..<n\})(n := x)$ 
let  $?L = \lambda \omega\ n\ r. INF\ i. emeasure\ (C\ (Suc\ n)\ i\ (? \omega\ \omega\ n\ r))\ (X\ (Suc\ n + i))$ 
have *: ( $\bigwedge i. i < n \implies ? \omega\ \omega\ i\ (\omega\ i) \in X\ (Suc\ i) \implies$ 
   $restrict\ \omega\ \{0..<n\} \in space\ (Pi_M\ \{0..<n\}\ M)$  for  $\omega\ n$ 
  using sets.sets-into-space[OF sets-X, of n]
  by (cases n) (auto simp: atLeastLessThanSuc restrict-def[of - {}])
have  $\exists \omega. \forall n. ? \omega\ \omega\ n\ (\omega\ n) \in X\ (Suc\ n) \wedge 0 < ?L\ \omega\ n\ (\omega\ n)$ 
proof (rule dependent-wellorder-choice)
  fix  $n\ \omega$  assume IH:  $\bigwedge i. i < n \implies ? \omega\ \omega\ i\ (\omega\ i) \in X\ (Suc\ i) \wedge 0 < ?L\ \omega\ i\ (\omega\ i)$ 
  show  $\exists r. ? \omega\ \omega\ n\ r \in X\ (Suc\ n) \wedge 0 < ?L\ \omega\ n\ r$ 
proof (rule step)
  show  $restrict\ \omega\ \{0..<n\} \in space\ (Pi_M\ \{0..<n\}\ M)$ 
    using IH[THEN conjunct1] by (rule *)
  show  $0 < (INF\ i. emeasure\ (C\ n\ i\ (restrict\ \omega\ \{0..<n\}))\ (X\ (n + i)))$ 
proof (cases n)
    case 0 with pos show ?thesis
      by (simp add: CI-def restrict-def)
    next
      case (Suc i) then show ?thesis
        using IH[of i, THEN conjunct2] by (simp add: atLeastLessThanSuc)
  qed
qed
qed (simp cong: restrict-cong)
then obtain  $\omega$  where  $\omega: \bigwedge n. ? \omega\ \omega\ n\ (\omega\ n) \in X\ (Suc\ n)$ 
  by auto
from this[THEN *] have  $\omega\text{-}space: \omega \in space\ (Pi_M\ UNIV\ M)$ 
  by (auto simp: space-PiM PiE-iff Ball-def)
have *:  $\omega \in PF.emb\ UNIV\ \{0..<n\}\ (X\ n)$  for  $n$ 
proof (cases n)
  case 0 with  $\omega\text{-}space\ \langle X\ 0 \neq \{\} \rangle$  sets.sets-into-space[OF sets-X, of 0] show
?thesis
    by (auto simp add: space-PiM prod-emb-def restrict-def PiE-iff)
  next

```



```

    case (Suc i) then show ?thesis
      using  $\omega$ [of i]  $\omega$ -space by (auto simp: prod-emb-def space-PiM PiE-iff atLeast-
LessThanSuc)
    qed
    have **:  $\{i. J i \subseteq \{0..<up-to (J n)\}\} \neq \{\}$  for n
      by (auto intro!: exI[of - n] up-to J)
    have  $\omega \in PF.emb UNIV (J n) (X' n)$  for n
      using *[of up-to (J n)] up-to[of J n] by (simp add: X-def prod-emb-Int prod-emb-INT[OF
**])
    then show  $(\bigcap i. PF.emb UNIV (J i) (X' i)) \neq \{\}$ 
      by auto
    qed

```

```

lemma distr-lim: assumes J[simp]: finite J shows distr PF.lim (PiM J M) ( $\lambda x.$ 
restrict x J) = CI J
  apply (rule measure-eqI)
  apply (simp add: CI-def)
  apply (simp add: emeasure-distr measurable-cong-sets[OF PF.sets-lim] lim[symmetric]
prod-emb-def space-PiM)
  done

```

end

```

lemma (in product-prob-space) emeasure-lim-emb:
  assumes *: finite J  $J \subseteq I$   $X \in sets (PiM J M)$ 
  shows emeasure lim (emb I J X) = emeasure (PiM J M) X
proof (rule emeasure-lim[OF *], goal-cases)
  case (1 J X)

  have  $\exists Q. (\forall i. sets Q = PiM (\bigcup i. J i) M \wedge distr Q (PiM (J i) M) (\lambda x. restrict$ 
 $x (J i)) = Pi_M (J i) M)$ 
  proof cases
    assume finite  $(\bigcup i. J i)$ 
    then have  $distr (PiM (\bigcup i. J i) M) (Pi_M (J i) M) (\lambda x. restrict x (J i)) =$ 
 $Pi_M (J i) M$  for i
      by (intro distr-restrict[symmetric]) auto
    then show ?thesis
      by auto
  next
    assume inf: infinite  $(\bigcup i. J i)$ 
    moreover have count: countable  $(\bigcup i. J i)$ 
      using 1(3) by (auto intro: countable-finite)
    define f where  $f = from-nat-into (\bigcup i. J i)$ 
    define t where  $t = to-nat-on (\bigcup i. J i)$ 
    have  $ft[simp]: x \in J i \implies f(t x) = x$  for x i
      unfolding f-def t-def using inf count by (intro from-nat-into-to-nat-on) auto
    have  $tf[simp]: t(f i) = i$  for i
      unfolding t-def f-def by (intro to-nat-on-from-nat-into-infinite inf count)
    have inj-t: inj-on t  $(\bigcup i. J i)$ 

```

```

    using count by (auto simp: t-def)
  then have inj-t-J: inj-on t (J i) for i
    by (rule subset-inj-on) auto
  interpret IT: Ionescu-Tulcea  $\lambda i. \omega. M (f i) \lambda i. M (f i)$ 
    by standard auto
  interpret Mf: product-prob-space  $\lambda x. M (f x) UNIV$ 
    by standard
  have C-eq-PiM: IT.C 0 n ( $\lambda. \text{undefined}$ ) = PiM {0.. $n$ } ( $\lambda x. M (f x)$ ) for n
  proof (induction n)
    case 0 then show ?case
      by (auto simp: PiM-empty intro!: measure-eqI dest!: subset-singletonD)
    next
      case (Suc n) then show ?case
        apply (auto intro!: measure-eqI simp: sets-bind[OF IT.sets-eP] emea-
          sure-bind[OF - IT.measurable-eP])
        apply (auto simp: Mf.product-nn-integral-insert nn-integral-indicator[symmetric]
          atLeastLessThanSuc IT.emeaure-eP space-PiM
            split: split-indicator simp del: nn-integral-indicator intro!:
              nn-integral-cong)
        done
      qed
    have CI-eq-PiM: IT.CI X = PiM X ( $\lambda x. M (f x)$ ) if X: finite X for X
    by (auto simp: IT.up-to-less X IT.CI-def C-eq-PiM intro!: Mf.distr-restrict[symmetric])

  let ?Q = distr IT.PF.lim (PiM ( $\bigcup i. J i$ ) M) ( $\lambda \omega. \lambda x \in \bigcup i. J i. \omega (t x)$ )
  { fix i
    have distr ?Q (PiM (J i) M) ( $\lambda x. \text{restrict } x (J i)$ ) =
      distr IT.PF.lim (PiM (J i) M) (( $\lambda \omega. \lambda n \in J i. \omega (t n)$ )  $\circ$  ( $\lambda \omega. \text{restrict } \omega (t' J$ 
i)))
    proof (subst distr-distr)
      have ( $\lambda \omega. \omega (t x)$ )  $\in$  measurable (PiM UNIV ( $\lambda x. M (f x)$ )) (M x) if x: x
 $\in J i$  for x i
      using measurable-component-singleton[of t x UNIV  $\lambda x. M (f x)$ ] unfolding
ft[OF x] by simp
      then show ( $\lambda \omega. \lambda x \in \bigcup i. J i. \omega (t x)$ )  $\in$  measurable IT.PF.lim (PiM ( $\bigcup (J$ 
‘ UNIV)) M)
      by (auto intro!: measurable-restrict simp: measurable-cong-sets[OF IT.PF.sets-lim
refl])
    qed (auto intro!: distr-cong measurable-restrict measurable-component-singleton)
    also have ... = distr (distr IT.PF.lim (PiM (t'J i) ( $\lambda x. M (f x)$ )) ( $\lambda \omega. \text{restrict } \omega (t' J i)$ )) (PiM (J i) M) ( $\lambda \omega. \lambda n \in J i. \omega (t n)$ )
    proof (intro distr-distr[symmetric])
      have ( $\lambda \omega. \omega (t x)$ )  $\in$  measurable (PiM (t'J i) ( $\lambda x. M (f x)$ )) (M x) if x: x
 $\in J i$  for x
      using measurable-component-singleton[of t x t'J i  $\lambda x. M (f x)$ ] x unfolding
ft[OF x] by auto
      then show ( $\lambda \omega. \lambda n \in J i. \omega (t n)$ )  $\in$  measurable (PiM (t ‘ J i) ( $\lambda x. M (f$ 
x))) (PiM (J i) M)
      by (auto intro!: measurable-restrict)

```

```

qed (auto intro!: measurable-restrict simp: measurable-cong-sets[OF IT.PF.sets-lim
refl])
  also have ... = distr (PiM (t‘J i) ( $\lambda x. M$  (f x))) (PiM (J i) M) ( $\lambda \omega. \lambda n \in J$ 
i.  $\omega$  (t n))
    using finite (J i) by (subst IT.distr-lim) (auto simp: CI-eq-PiM)
    also have ... = PiM (J i) M
    using Mf.distr-reorder[of t J i] by (simp add: 1 inj-t-J cong: PiM-cong)
    finally have distr ?Q (PiM (J i) M) ( $\lambda x. \text{restrict } x$  (J i)) = PiM (J i) M . }
  then show  $\exists Q. \forall i. \text{sets } Q = \text{PiM } (\bigcup i. J i) M \wedge \text{distr } Q (\text{PiM } (J i) M) (\lambda x.$ 
restrict x (J i)) = PiM (J i) M
    by (intro exI[of - ?Q]) auto
qed
then obtain Q where sets-Q: sets Q = PiM ( $\bigcup i. J i$ ) M
  and Q:  $\bigwedge i. \text{distr } Q (\text{PiM } (J i) M) (\lambda x. \text{restrict } x (J i)) = \text{PiM } (J i) M$  by
blast

from 1 interpret Q: prob-space Q
  by (intro prob-space-distrD[of  $\lambda x. \text{restrict } x (J 0) Q \text{ PiM } (J 0) M$ ]
    (auto simp: Q measurable-cong-sets[OF sets-Q]
      intro!: prob-space-P measurable-restrict measurable-component-singleton))

  have  $0 < (\text{INF } i. \text{emeasure } (\text{PiM } (J i) M) (X i))$  by fact
  also have ... =  $(\text{INF } i. \text{emeasure } Q (\text{emb } (\bigcup i. J i) (J i) (X i)))$ 
    by (simp add: emeasure-distr-restrict[OF - sets-Q 1(4), symmetric] SUP-upper
Q)
  also have ... = emeasure Q ( $\bigcap i. \text{emb } (\bigcup i. J i) (J i) (X i)$ )
  proof (rule INF-emeasure-decseq)
    from 1 show decseq ( $\lambda n. \text{emb } (\bigcup i. J i) (J n) (X n)$ )
    by (intro antimonoI emb-preserve-mono[where X=emb ( $\bigcup i. J i$ ) (J n) (X
n) and L=I and J= $\bigcup i. J i$  for n]
      measurable-prod-emb)
    (auto simp: SUP-least SUP-upper antimono-def)
  qed (insert 1, auto simp: sets-Q)
  finally have ( $\bigcap i. \text{emb } (\bigcup i. J i) (J i) (X i) \neq \{\}$ )
    by auto
  moreover have ( $\bigcap i. \text{emb } I (J i) (X i) = \{\} \implies (\bigcap i. \text{emb } (\bigcup i. J i) (J i) (X$ 
i)) = \{\})
    using 1 by (intro emb-injective[of  $\bigcup i. J i I - \{\}$ ] sets.countable-INT) (auto
simp: SUP-least SUP-upper)
  ultimately show ?case by auto
qed

end

```

9 Infinite Product Measure

```

theory Infinite-Product-Measure
  imports Probability-Measure Projective-Family
begin

```

lemma (in *product-prob-space*) *distr-PiM-restrict-finite*:
assumes *finite J J ⊆ I*
shows *distr (PiM I M) (PiM J M) (λx. restrict x J) = PiM J M*
proof (rule *PiM-eqI*)
fix *X* **assume** *X: ∧i. i ∈ J ⇒ X i ∈ sets (M i)*
{ fix J X assume J: J ≠ {} ∨ I = {} finite J J ⊆ I and X: ∧i. i ∈ J ⇒ X
i ∈ sets (M i)
have *emeasure (PiM I M) (emb I J (Pi_E J X)) = (∏_{i∈J}. M i (X i))*
proof (*subst emeasure-extend-measure-Pair[OF PiM-def, where μ'=lim], goal-cases*)
case 1 then show ?case
by (*simp add: M.emeasure-space-1 emeasure-PiM Pi-iff sets-PiM-I-finite*
emeasure-lim-emb)
next
case (2 J X)
then have *emb I J (Pi_E J X) ∈ sets (PiM I M)*
by (*intro measurable-prod-emb sets-PiM-I-finite*) *auto*
from this[THEN sets.sets-into-space] show ?case
by (*simp add: space-PiM*)
qed (*insert assms J X, simp-all del: sets-lim*
add: M.emeasure-space-1 sets-lim[symmetric] emeasure-countably-additive
emeasure-positive) }
note ** = this*

have *emeasure (PiM I M) (emb I J (Pi_E J X)) = (∏_{i∈J}. M i (X i))*
proof (*cases J ≠ {} ∨ I = {}*)
case False
then obtain i where i: J = {} i ∈ I by auto
then have *emb I {} {λx. undefined} = emb I {i} (Π_E i∈{i}. space (M i))*
by (*auto simp: space-PiM prod-emb-def*)
with i show ?thesis
by (*simp add: * M.emeasure-space-1*)
next
case True
then show ?thesis
by (*simp add: *[OF - assms X]*)
qed
with assms show *emeasure (distr (Pi_M I M) (Pi_M J M) (λx. restrict x J))*
(Pi_E J X) = (∏_{i∈J}. emeasure (M i) (X i))
by (*subst emeasure-distr-restrict[OF - refl] (auto intro!: sets-PiM-I-finite X)*)
qed (*insert assms, auto*)

lemma (in *product-prob-space*) *emeasure-PiM-emb'*:
J ⊆ I ⇒ finite J ⇒ X ∈ sets (PiM J M) ⇒ emeasure (Pi_M I M) (emb I J
X) = PiM J M X
by (*subst distr-PiM-restrict-finite[symmetric, of J]*)
(auto intro!: emeasure-distr-restrict[symmetric])

lemma (in *product-prob-space*) *emeasure-PiM-emb*:

$J \subseteq I \implies \text{finite } J \implies (\bigwedge i. i \in J \implies X i \in \text{sets } (M i)) \implies$
 $\text{emeasure } (Pi_M I M) (\text{emb } I J (Pi_E J X)) = (\prod_{i \in J. \text{emeasure } (M i) (X i)})$
by (*subst emeasure-PiM-emb'*) (*auto intro! : emeasure-PiM*)

sublocale *product-prob-space* $\subseteq P?$: *prob-space* $Pi_M I M$

proof

have *: $\text{emb } I \{ \} \{ \lambda x. \text{undefined} \} = \text{space } (Pi_M I M)$
by (*auto simp: prod-emb-def space-PiM*)
show $\text{emeasure } (Pi_M I M) (\text{space } (Pi_M I M)) = 1$
using *emeasure-PiM-emb*[*of* $\{ \} \lambda-. \{ \}$] **by** (*simp add: **)

qed

lemma *prob-space-PiM*:

assumes $M: \bigwedge i. i \in I \implies \text{prob-space } (M i)$ **shows** *prob-space* $(Pi_M I M)$

proof –

let $?M = \lambda i. \text{if } i \in I \text{ then } M i \text{ else count-space } \{ \text{undefined} \}$

interpret M' : *prob-space* $?M i$ **for** i

using M **by** (*cases* $i \in I$) (*auto intro! : prob-spaceI*)

interpret *product-prob-space* $?M I$

by *unfold-locales*

have *prob-space* $(\prod_M i \in I. ?M i)$

by *unfold-locales*

also have $(\prod_M i \in I. ?M i) = (\prod_M i \in I. M i)$

by (*intro PiM-cong*) *auto*

finally show *?thesis* .

qed

lemma (*in product-prob-space*) *emeasure-PiM-Collect*:

assumes $X: J \subseteq I \text{ finite } J \bigwedge i. i \in J \implies X i \in \text{sets } (M i)$

shows $\text{emeasure } (Pi_M I M) \{ x \in \text{space } (Pi_M I M). \forall i \in J. x i \in X i \} = (\prod_{i \in J. \text{emeasure } (M i) (X i)})$

proof –

have $\{ x \in \text{space } (Pi_M I M). \forall i \in J. x i \in X i \} = \text{emb } I J (Pi_E J X)$

unfolding *prod-emb-def* **using** *assms* **by** (*auto simp: space-PiM Pi-iff*)

with *emeasure-PiM-emb*[*OF* *assms*] **show** *?thesis* **by** *simp*

qed

lemma (*in product-prob-space*) *emeasure-PiM-Collect-single*:

assumes $X: i \in I \implies A \in \text{sets } (M i)$

shows $\text{emeasure } (Pi_M I M) \{ x \in \text{space } (Pi_M I M). x i \in A \} = \text{emeasure } (M i) A$

using *emeasure-PiM-Collect*[*of* $\{ i \} \lambda i. A$] *assms*

by *simp*

lemma (*in product-prob-space*) *measure-PiM-emb*:

assumes $J \subseteq I \text{ finite } J \bigwedge i. i \in J \implies X i \in \text{sets } (M i)$

shows $\text{measure } (Pi_M I M) (\text{emb } I J (Pi_E J X)) = (\prod_{i \in J. \text{measure } (M i) (X i)})$

using *emeasure-PiM-emb*[*OF* *assms*]

unfolding *emeasure-eq-measure* $M. \text{emeasure-eq-measure}$

by (simp add: prod-ennreal measure-nonneg prod-nonneg)

lemma sets-Collect-single':

$i \in I \implies \{x \in \text{space } (M \ i). \ P \ x\} \in \text{sets } (M \ i) \implies \{x \in \text{space } (PiM \ I \ M). \ P \ (x \ i)\} \in \text{sets } (PiM \ I \ M)$

by auto

lemma (in finite-product-prob-space) finite-measure-PiM-emb:

$(\bigwedge i. i \in I \implies A \ i \in \text{sets } (M \ i)) \implies \text{measure } (PiM \ I \ M) \ (Pi_E \ I \ A) = (\prod_{i \in I}. \text{measure } (M \ i) \ (A \ i))$

by (rule prob-times)

lemma (in product-prob-space) PiM-component:

assumes $i \in I$

shows $\text{distr } (PiM \ I \ M) \ (M \ i) \ (\lambda \omega. \ \omega \ i) = M \ i$

proof (rule measure-eqI[symmetric])

fix A assume $A \in \text{sets } (M \ i)$

moreover have $((\lambda \omega. \ \omega \ i) - 'A \cap \text{space } (PiM \ I \ M)) = \{x \in \text{space } (PiM \ I \ M). \ x \ i \in A\}$

by auto

ultimately show $\text{emeasure } (M \ i) \ A = \text{emeasure } (\text{distr } (PiM \ I \ M) \ (M \ i) \ (\lambda \omega. \ \omega \ i)) \ A$

by (auto simp: $\langle i \in I \rangle$ emeasure-distr measurable-component-singleton emeasure-PiM-Collect-single)

qed simp

lemma (in product-prob-space) PiM-eq:

assumes $M': \text{sets } M' = \text{sets } (PiM \ I \ M)$

assumes eq: $\bigwedge J \ F. \text{finite } J \implies J \subseteq I \implies (\bigwedge j. j \in J \implies F \ j \in \text{sets } (M \ j)) \implies \text{emeasure } M' \ (\text{prod-emb } I \ M \ J \ (\Pi_E \ j \in J. \ F \ j)) = (\prod_{j \in J}. \text{emeasure } (M \ j) \ (F \ j))$

shows $M' = (PiM \ I \ M)$

proof (rule measure-eqI-PiM-infinite[symmetric, OF refl M'])

show $\text{finite-measure } (PiM \ I \ M)$

by standard (simp add: P.emeasure-space-1)

qed (simp add: eq emeasure-PiM-emb)

lemma (in product-prob-space) AE-component: $i \in I \implies AE \ x \text{ in } M \ i. \ P \ x \implies AE \ x \text{ in } PiM \ I \ M. \ P \ (x \ i)$

apply (rule AE-distrD[of $\lambda \omega. \ \omega \ i \ PiM \ I \ M \ M \ i \ P$])

apply simp

apply (subst PiM-component)

apply simp-all

done

lemma emeasure-PiM-emb:

assumes $M: \bigwedge i. i \in I \implies \text{prob-space } (M \ i)$

assumes $J: J \subseteq I \text{ finite } J$ and $A: \bigwedge i. i \in J \implies A \ i \in \text{sets } (M \ i)$

shows $\text{emeasure } (PiM \ I \ M) \ (\text{prod-emb } I \ M \ J \ (Pi_E \ J \ A)) = (\prod_{i \in J}. \text{emeasure } (M \ i) \ (A \ i))$

$(M\ i)\ (A\ i))$
proof –
 let $?M = \lambda i. \text{ if } i \in I \text{ then } M\ i \text{ else count-space } \{\text{undefined}\}$
 interpret M' : prob-space $?M\ i$ **for** i
 using M **by** (cases $i \in I$) (auto intro!: prob-spaceI)
 interpret P : product-prob-space $?M\ I$
 by unfold-locales
 have $\text{emeasure } (Pi_M\ I\ M)\ (\text{prod-emb } I\ M\ J\ (Pi_E\ J\ A)) = \text{emeasure } (Pi_M\ I\ ?M)$
 $(P.\text{emb } I\ J\ (Pi_E\ J\ A))$
 by (auto simp: prod-emb-def Pi_E -iff intro!: arg-cong2[**where** $f = \text{emeasure}$]
 Pi_M -cong)
 also have $\dots = (\prod_{i \in J}. \text{emeasure } (M\ i)\ (A\ i))$
 using $J\ A$ **by** (subst $P.\text{emeasure-}Pi_M\text{-emb}[OF\ J]$) (auto intro!: prod.cong)
 finally show $?thesis$.
qed

lemma *distr-pair- Pi_M -eq- Pi_M :*

fixes $i' :: 'i$ **and** $I :: 'i$ **set and** $M :: 'i \Rightarrow 'a$ *measure*
 assumes $M: \bigwedge i. i \in I \implies \text{prob-space } (M\ i)\ \text{prob-space } (M\ i')$
 shows $\text{distr } (M\ i' \otimes_M (\prod_{i \in I}. M\ i)) (\prod_{i \in \text{insert } i'\ I}. M\ i) (\lambda(x, X). X(i' := x)) =$
 $(\prod_{i \in \text{insert } i'\ I}. M\ i) \text{ (is ?L = -)}$
proof (rule *measure-eqI- Pi_M -infinite[symmetric, OF refl]*)
 interpret M' : prob-space $M\ i'$ **by fact**
 interpret I : prob-space $(\prod_{i \in I}. M\ i)$
 using M **by** (intro prob-space- Pi_M) auto
 interpret I' : prob-space $(\prod_{i \in \text{insert } i'\ I}. M\ i)$
 using M **by** (intro prob-space- Pi_M) auto
 show *finite-measure* $(\prod_{i \in \text{insert } i'\ I}. M\ i)$
 by unfold-locales
 fix $J\ A$ **assume** J : *finite* $J\ J \subseteq \text{insert } i'\ I$ **and** $A: \bigwedge i. i \in J \implies A\ i \in \text{sets } (M\ i)$
 let $?X = \text{prod-emb } (\text{insert } i'\ I)\ M\ J\ (Pi_E\ J\ A)$
 have $Pi_M\ (\text{insert } i'\ I)\ M\ ?X = (\prod_{i \in J}. M\ i\ (A\ i))$
 using $M\ J\ A$ **by** (intro *emeasure- Pi_M -emb*) auto
 also have $\dots = M\ i' \text{ (if } i' \in J \text{ then } (A\ i') \text{ else space } (M\ i')) * (\prod_{i \in J - \{i'\}}. M\ i\ (A\ i))$
 using *prod.insert-remove[of J $\lambda i. M\ i\ (A\ i)\ i']\ J\ M'.\text{emeasure-space-1}$*
 by (cases $i' \in J$) (auto simp: *insert-absorb*)
 also have $(\prod_{i \in J - \{i'\}}. M\ i\ (A\ i)) = Pi_M\ I\ M\ (\text{prod-emb } I\ M\ (J - \{i'\})\ (Pi_E\ (J - \{i'\})\ A))$
 using $M\ J\ A$ **by** (intro *emeasure- Pi_M -emb[symmetric]*) auto
 also have $M\ i' \text{ (if } i' \in J \text{ then } (A\ i') \text{ else space } (M\ i')) * \dots =$
 $(M\ i' \otimes_M Pi_M\ I\ M) ((\text{if } i' \in J \text{ then } (A\ i') \text{ else space } (M\ i')) \times \text{prod-emb } I\ M\ (J - \{i'\})\ (Pi_E\ (J - \{i'\})\ A))$
 using $J\ A$ **by** (intro *I.emeasure-pair-measure-Times[symmetric]\ sets- Pi_M -I*) auto
 also have $((\text{if } i' \in J \text{ then } (A\ i') \text{ else space } (M\ i')) \times \text{prod-emb } I\ M\ (J - \{i'\})\ (Pi_E\ (J - \{i'\})\ A)) =$

```

    (λ(x, X). X(i' := x)) - ' ?X ∩ space (M i' ⊗M PiM I M)
    using A[of i', THEN sets.sets-into-space] unfolding set-eq-iff
    by (simp add: prod-emb-def space-pair-measure space-PiM PiE-fun-upd ac-simps
    cong: conj-cong)
    (auto simp add: Pi-iff Ball-def all-conj-distrib)
    finally show PiM (insert i' I) M ?X = ?L ?X
    using J A by (simp add: emeasure-distr)
qed simp

```

lemma *distr-PiM-reindex*:

```

    assumes M: ⋀i. i ∈ K ⟹ prob-space (M i)
    assumes f: inj-on f I f ∈ I → K
    shows distr (PiM K M) (ΠM i∈I. M (f i)) (λω. λn∈I. ω (f n)) = (ΠM i∈I. M
    (f i))
    (is distr ?K ?I ?t = ?I)
proof (rule measure-eqI-PiM-infinite[symmetric, OF refl])
  interpret prob-space ?I
  using f M by (intro prob-space-PiM) auto
  show finite-measure ?I
  by unfold-locales
  fix A J assume J: finite J J ⊆ I and A: ⋀i. i ∈ J ⟹ A i ∈ sets (M (f i))
  have [simp]: i ∈ J ⟹ the-inv-into I f (f i) = i for i
  using J f by (intro the-inv-into-f-f) auto
  have ?I (prod-emb I (λi. M (f i)) J (PiE J A)) = (Πj∈J. M (f j) (A j))
  using f J A by (intro emeasure-PiM-emb M) auto
  also have ... = (Πj∈f'J. M j (A (the-inv-into I f j)))
  using f J by (subst prod.reindex) (auto intro!: prod.cong intro: inj-on-subset
  simp: the-inv-into-f-f)
  also have ... = ?K (prod-emb K M (f'J) (ΠE j∈f'J. A (the-inv-into I f j)))
  using f J A by (intro emeasure-PiM-emb[symmetric] M) (auto simp: the-inv-into-f-f)
  also have prod-emb K M (f'J) (ΠE j∈f'J. A (the-inv-into I f j)) = ?t - ' prod-emb
  I (λi. M (f i)) J (PiE J A) ∩ space ?K
  using f J A by (auto simp: prod-emb-def space-PiM Pi-iff PiE-iff Int-absorb1)
  also have ?K ... = distr ?K ?I ?t (prod-emb I (λi. M (f i)) J (PiE J A))
  using f J A by (intro emeasure-distr[symmetric] sets-PiM-I) (auto simp: Pi-iff)
  finally show ?I (prod-emb I (λi. M (f i)) J (PiE J A)) = distr ?K ?I ?t
  (prod-emb I (λi. M (f i)) J (PiE J A)) .
qed simp

```

lemma *distr-PiM-component*:

```

    assumes M: ⋀i. i ∈ I ⟹ prob-space (M i)
    assumes i ∈ I
    shows distr (PiM I M) (M i) (λω. ω i) = M i
proof -
  have *: (λω. ω i) - ' A ∩ space (PiM I M) = prod-emb I M {i} (ΠE i'∈{i}. A)
  for A
  by (auto simp: prod-emb-def space-PiM)
  show ?thesis
  apply (intro measure-eqI)

```



```

apply (auto simp add: emeasure-distr ⟨i∈I⟩ * emeasure-PiM-emb M)
apply (subst emeasure-PiM-emb)
apply (simp-all add: M ⟨i∈I⟩)
done
qed

```

lemma *AE-PiM-component*:

```

(⋀i. i ∈ I ⟹ prob-space (M i)) ⟹ i ∈ I ⟹ AE x in M i. P x ⟹ AE x in
PiM I M. P (x i)
using AE-distrD[of λx. x i PiM I M M i]
by (subst (asm) distr-PiM-component[of I - i]) (auto intro: AE-distrD[of λx. x i
- - P])

```

lemma *decseq-emb-PiE*:

```

incseq J ⟹ decseq (λi. prod-emb I M (J i) (ΠE j∈J i. X j))
by (fastforce simp: decseq-def prod-emb-def incseq-def Pi-iff)

```

9.1 Sequence space

definition *comb-seq* :: $\text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a)$ **where**
 $\text{comb-seq } i \ \omega \ \omega' \ j = (\text{if } j < i \text{ then } \omega \ j \text{ else } \omega' (j - i))$

lemma *split-comb-seq*: $P (\text{comb-seq } i \ \omega \ \omega' \ j) \longleftrightarrow (j < i \longrightarrow P (\omega \ j)) \wedge (\forall k. j = i + k \longrightarrow P (\omega' \ k))$
by (auto simp: comb-seq-def not-less)

lemma *split-comb-seq-asm*: $P (\text{comb-seq } i \ \omega \ \omega' \ j) \longleftrightarrow \neg ((j < i \wedge \neg P (\omega \ j)) \vee (\exists k. j = i + k \wedge \neg P (\omega' \ k)))$
by (auto simp: comb-seq-def)

lemma *measurable-comb-seq*:

```

(λ(ω, ω'). comb-seq i ω ω') ∈ measurable ((ΠM i∈UNIV. M) ⊗M (ΠM i∈UNIV.
M)) (ΠM i∈UNIV. M)

```

proof (rule measurable-PiM-single)

```

show (λ(ω, ω'). comb-seq i ω ω') ∈ space ((ΠM i∈UNIV. M) ⊗M (ΠM i∈UNIV.
M)) → (UNIV →E space M)

```

by (auto simp: space-pair-measure space-PiM PiE-iff split: split-comb-seq)

fix $j :: \text{nat}$ **and** A **assume** $A: A \in \text{sets } M$

then have *: $\{\omega \in \text{space } ((\Pi_M i \in \text{UNIV}. M) \otimes_M (\Pi_M i \in \text{UNIV}. M)). \text{case-prod } (\text{comb-seq } i) \ \omega \ j \in A\} =$

$(\text{if } j < i \text{ then } \{\omega \in \text{space } (\Pi_M i \in \text{UNIV}. M). \omega \ j \in A\} \times \text{space } (\Pi_M i \in \text{UNIV}. M)$

$\text{else } \text{space } (\Pi_M i \in \text{UNIV}. M) \times \{\omega \in \text{space } (\Pi_M i \in \text{UNIV}. M). \omega (j - i) \in A\})$

by (auto simp: space-PiM space-pair-measure comb-seq-def dest: sets.sets-into-space)

show $\{\omega \in \text{space } ((\Pi_M i \in \text{UNIV}. M) \otimes_M (\Pi_M i \in \text{UNIV}. M)). \text{case-prod } (\text{comb-seq } i) \ \omega \ j \in A\} \in \text{sets } ((\Pi_M i \in \text{UNIV}. M) \otimes_M (\Pi_M i \in \text{UNIV}. M))$

unfolding * **by** (auto simp: A intro!: sets-Collect-single)

qed

lemma *measurable-comb-seq'*[*measurable (raw)*]:
assumes $f: f \in \text{measurable } N \ (\Pi_M \ i \in \text{UNIV}. M)$ **and** $g: g \in \text{measurable } N \ (\Pi_M \ i \in \text{UNIV}. M)$
shows $(\lambda x. \text{comb-seq } i \ (f \ x) \ (g \ x)) \in \text{measurable } N \ (\Pi_M \ i \in \text{UNIV}. M)$
using *measurable-compose*[*OF measurable-Pair*[*OF f g*] *measurable-comb-seq*] **by** *simp*

lemma *comb-seq-0*: *comb-seq 0 ω ω' = ω'*
by (*auto simp add: comb-seq-def*)

lemma *comb-seq-Suc*: *comb-seq (Suc n) ω ω' = comb-seq n ω (case-nat (ω n) ω')*
by (*auto simp add: comb-seq-def not-less less-Suc-eq le-imp-diff-is-add intro!: ext split: nat.split*)

lemma *comb-seq-Suc-0*[*simp*]: *comb-seq (Suc 0) ω = case-nat (ω 0)*
by (*intro ext*) (*simp add: comb-seq-Suc comb-seq-0*)

lemma *comb-seq-less*: $i < n \implies \text{comb-seq } n \ \omega \ \omega' \ i = \omega \ i$
by (*auto split: split-comb-seq*)

lemma *comb-seq-add*: *comb-seq n ω ω' (i + n) = $\omega' \ i$*
by (*auto split: nat.split split-comb-seq*)

lemma *case-nat-comb-seq*: *case-nat s' (comb-seq n ω ω') (i + n) = case-nat (case-nat s' ω n) $\omega' \ i$*
by (*auto split: nat.split split-comb-seq*)

lemma *case-nat-comb-seq'*:
 $\text{case-nat } s \ (\text{comb-seq } i \ \omega \ \omega') = \text{comb-seq } (\text{Suc } i) \ (\text{case-nat } s \ \omega) \ \omega'$
by (*auto split: split-comb-seq nat.split*)

locale *sequence-space* = *product-prob-space* $\lambda i. M \ \text{UNIV} :: \text{nat set}$ **for** M
begin

abbreviation $S \equiv \Pi_M \ i \in \text{UNIV} :: \text{nat set}. M$

lemma *infprod-in-sets*[*intro*]:
fixes $E :: \text{nat} \Rightarrow 'a \text{ set}$ **assumes** $E: \bigwedge i. E \ i \in \text{sets } M$
shows $\Pi i \ \text{UNIV} \ E \in \text{sets } S$
proof –
have $\Pi i \ \text{UNIV} \ E = (\bigcap i. \text{emb } \text{UNIV} \ \{..i\} \ (\Pi_E \ j \in \{..i\}. E \ j))$
using $E \ E[\text{THEN sets.sets-into-space}]$
by (*auto simp: prod-emb-def Pi-iff extensional-def*)
with E **show** *?thesis* **by** *auto*
qed

lemma *measure-PiM-countable*:
fixes $E :: \text{nat} \Rightarrow 'a \text{ set}$ **assumes** $E: \bigwedge i. E \ i \in \text{sets } M$

shows $(\lambda n. \prod_{i \leq n}. \text{measure } M (E i)) \longrightarrow \text{measure } S (Pi \text{ UNIV } E)$
proof –
 let $?E = \lambda n. \text{emb UNIV } \{..n\} (Pi_E \{.. n\} E)$
 have $\bigwedge n. (\prod_{i \leq n}. \text{measure } M (E i)) = \text{measure } S (?E n)$
 using E **by** (*simp add: measure-PiM-emb*)
 moreover have $Pi \text{ UNIV } E = (\bigcap n. ?E n)$
 using E $E[THEN \text{sets.sets-into-space}]$
 by (*auto simp: prod-emb-def extensional-def Pi-iff*)
 moreover have $\text{range } ?E \subseteq \text{sets } S$
 using E **by** *auto*
 moreover have $\text{decseq } ?E$
 by (*auto simp: prod-emb-def Pi-iff decseq-def*)
 ultimately show $?thesis$
 by (*simp add: finite-Lim-measure-decseq*)
qed

lemma *nat-eq-diff-eq*:
 fixes $a \ b \ c :: \text{nat}$
 shows $c \leq b \implies a = b - c \longleftrightarrow a + c = b$
by *auto*

lemma *PiM-comb-seq*:
 distr $(S \otimes_M S) S (\lambda(\omega, \omega'). \text{comb-seq } i \ \omega \ \omega') = S \text{ (is } ?D = -)$
proof (*rule PiM-eq*)
 let $?I = \text{UNIV}::\text{nat set}$ **and** $?M = \lambda n. M$
 let $\text{distr } - - ?f = ?D$

fix $J \ E$ **assume** $J: \text{finite } J \ J \subseteq ?I \ \bigwedge j. j \in J \implies E j \in \text{sets } M$
 let $?X = \text{prod-emb } ?I \ ?M \ J (\Pi_E j \in J. E j)$
 have $\bigwedge j \ x. j \in J \implies x \in E j \implies x \in \text{space } M$
 using $J(3)[THEN \text{sets.sets-into-space}]$ **by** (*auto simp: space-PiM Pi-iff sub-set-eq*)
with J **have** $?f - ' ?X \cap \text{space } (S \otimes_M S) =$
 $(\text{prod-emb } ?I \ ?M (J \cap \{..<i\}) (\Pi_E j \in J \cap \{..<i\}. E j)) \times$
 $(\text{prod-emb } ?I \ ?M (((+) i) - ' J) (\Pi_E j \in ((+) i) - ' J. E (i + j))) \text{ (is } - = ?E \times ?F)$
 by (*auto simp: space-pair-measure space-PiM prod-emb-def all-conj-distrib PiE-iff split: split-comb-seq split-comb-seq-asm*)
then have $\text{emeasure } ?D \ ?X = \text{emeasure } (S \otimes_M S) (?E \times ?F)$
 by (*subst emeasure-distr[OF measurable-comb-seq]*)
 (*auto intro!: sets-PiM-I simp: split-beta' J*)
also have $\dots = \text{emeasure } S \ ?E * \text{emeasure } S \ ?F$
 using J **by** (*intro P.emeasure-pair-measure-Times*) (*auto intro!: sets-PiM-I finite-vimageI simp: inj-on-def*)
also have $\text{emeasure } S \ ?F = (\prod_{j \in ((+) i) - ' J. \text{emeasure } M (E (i + j))})$
 using J **by** (*intro emeasure-PiM-emb*) (*simp-all add: finite-vimageI inj-on-def*)
also have $\dots = (\prod_{j \in J - (J \cap \{..<i\}). \text{emeasure } M (E j)})$
 by (*rule prod.reindex-cong [of $\lambda x. x - i$]*)
 (*auto simp: image-iff ac-simps nat-eq-diff-eq cong: conj-cong intro!: inj-onI*)

also have $\text{emeasure } S \text{ ?}E = (\prod_{j \in J \cap \{..<i\}}. \text{emeasure } M (E j))$
using J **by** (*intro emeasure-PiM-emb*) *simp-all*
also have $(\prod_{j \in J \cap \{..<i\}}. \text{emeasure } M (E j)) * (\prod_{j \in J - (J \cap \{..<i\})}. \text{emeasure } M (E j)) = (\prod_{j \in J}. \text{emeasure } M (E j))$
by (*subst mult.commute*) (*auto simp: J prod.subset-diff[symmetric]*)
finally show $\text{emeasure ?}D \text{ ?}X = (\prod_{j \in J}. \text{emeasure } M (E j))$.
qed *simp-all*

lemma *PiM-iter*:

$\text{distr } (M \otimes_M S) S (\lambda(s, \omega). \text{case-nat } s \ \omega) = S \text{ (is ?}D = \text{)}$
proof (*rule PiM-eq*)
let $?I = \text{UNIV}::\text{nat set}$ **and** $?M = \lambda n. M$
let $\text{distr} - - \text{ ?}f = ?D$

fix $J \ E$ **assume** $J: \text{finite } J \ J \subseteq ?I \ \bigwedge j. j \in J \implies E j \in \text{sets } M$
let $?X = \text{prod-emb } ?I \ ?M \ J (\prod_{E j \in J}. E j)$
have $\bigwedge j. j \in J \implies x \in E j \implies x \in \text{space } M$
using $J(3)[\text{THEN sets.sets-into-space}]$ **by** (*auto simp: space-PiM Pi-iff subset-eq*)
with J **have** $?f - ' ?X \cap \text{space } (M \otimes_M S) = (\text{if } 0 \in J \text{ then } E \ 0 \text{ else } \text{space } M)$
 \times
 $(\text{prod-emb } ?I \ ?M (\text{Suc} - ' J) (\prod_{E j \in \text{Suc} - ' J}. E (\text{Suc } j))) \text{ (is } - = ?E \times ?F)$
by (*auto simp: space-pair-measure space-PiM PiE-iff prod-emb-def all-conj-distrib split: nat.split nat.split-asm*)
then have $\text{emeasure } ?D \ ?X = \text{emeasure } (M \otimes_M S) (?E \times ?F)$
by (*subst emeasure-distr*)
 $(\text{auto intro!: sets-PiM-I simp: split-beta' } J)$
also have $\dots = \text{emeasure } M \ ?E * \text{emeasure } S \ ?F$
using J **by** (*intro P.emeasure-pair-measure-Times*) (*auto intro!: sets-PiM-I finite-vimageI*)
also have $\text{emeasure } S \ ?F = (\prod_{j \in \text{Suc} - ' J}. \text{emeasure } M (E (\text{Suc } j)))$
using J **by** (*intro emeasure-PiM-emb*) (*simp-all add: finite-vimageI*)
also have $\dots = (\prod_{j \in J - \{0\}}. \text{emeasure } M (E j))$
by (*rule prod.reindex-cong [of $\lambda x. x - 1$]*)
 $(\text{auto simp: image-iff nat-eq-diff-eq ac-simps cong: conj-cong intro!: inj-onI})$
also have $\text{emeasure } M \ ?E * (\prod_{j \in J - \{0\}}. \text{emeasure } M (E j)) = (\prod_{j \in J}. \text{emeasure } M (E j))$
 $\text{emeasure } M (E j))$
by (*auto simp: M.emeasure-space-1 prod.remove J*)
finally show $\text{emeasure } ?D \ ?X = (\prod_{j \in J}. \text{emeasure } M (E j))$.
qed *simp-all*

end

lemma *PiM-return*:

assumes *finite I*
assumes [*measurable*]: $\bigwedge i. i \in I \implies \{a \ i\} \in \text{sets } (M \ i)$
shows $\text{PiM } I (\lambda i. \text{return } (M \ i) (a \ i)) = \text{return } (\text{PiM } I \ M) (\text{restrict } a \ I)$
proof –
have [*simp*]: $a \ i \in \text{space } (M \ i)$ **if** $i \in I$ **for** i

```

using assms(2)[OF that] by (meson insert-subset sets.sets-into-space)
interpret prob-space PiM I ( $\lambda i. \text{return } (M\ i) (a\ i)$ )
by (intro prob-space-PiM prob-space-return) auto
have AE x in PiM I ( $\lambda i. \text{return } (M\ i) (a\ i)$ ).  $\forall i \in I. x\ i = \text{restrict } a\ I\ i$ 
by (intro eventually-ball-finite ballI AE-PiM-component prob-space-return assms)
    (auto simp: AE-return)
moreover have AE x in PiM I ( $\lambda i. \text{return } (M\ i) (a\ i)$ ).  $x \in \text{space } (PiM\ I\ (\lambda i. \text{return } (M\ i) (a\ i)))$ 
by simp
ultimately have AE x in PiM I ( $\lambda i. \text{return } (M\ i) (a\ i)$ ).  $x = \text{restrict } a\ I$ 
by eventually-elim (auto simp: fun-eq-iff space-PiM)
hence  $Pi_M\ I\ (\lambda i. \text{return } (M\ i) (a\ i)) = \text{return } (Pi_M\ I\ (\lambda i. \text{return } (M\ i) (a\ i)))$ 
    (restrict a I)
by (rule AE-eq-constD)
also have  $\dots = \text{return } (PiM\ I\ M) (\text{restrict } a\ I)$ 
by (intro return-cong sets-PiM-cong) auto
finally show ?thesis .
qed

```

lemma *distr-PiM-finite-prob-space'*:

```

assumes fin: finite I
assumes  $\bigwedge i. i \in I \implies \text{prob-space } (M\ i)$ 
assumes  $\bigwedge i. i \in I \implies \text{prob-space } (M'\ i)$ 
assumes [measurable]:  $\bigwedge i. i \in I \implies f \in \text{measurable } (M\ i) (M'\ i)$ 
shows  $\text{distr } (PiM\ I\ M) (PiM\ I\ M') (\text{compose } I\ f) = PiM\ I\ (\lambda i. \text{distr } (M\ i) (M'\ i) f)$ 
proof –
  define N where  $N = (\lambda i. \text{if } i \in I \text{ then } M\ i \text{ else return } (\text{count-space UNIV}))$ 
    undefined
  define N' where  $N' = (\lambda i. \text{if } i \in I \text{ then } M'\ i \text{ else return } (\text{count-space UNIV}))$ 
    undefined
  have [simp]:  $PiM\ I\ N = PiM\ I\ M\ PiM\ I\ N' = PiM\ I\ M'$ 
    by (intro PiM-cong; simp add: N-def N'-def) +

  have  $\text{distr } (PiM\ I\ N) (PiM\ I\ N') (\text{compose } I\ f) = PiM\ I\ (\lambda i. \text{distr } (N\ i) (N'\ i) f)$ 
proof (rule distr-PiM-finite-prob-space)
  show product-prob-space N
    by (rule product-prob-spaceI (auto simp: N-def intro!: prob-space-return assms))
  show product-prob-space N'
    by (rule product-prob-spaceI (auto simp: N'-def intro!: prob-space-return assms))
qed (auto simp: N-def N'-def fin)
also have  $Pi_M\ I\ (\lambda i. \text{distr } (N\ i) (N'\ i) f) = Pi_M\ I\ (\lambda i. \text{distr } (M\ i) (M'\ i) f)$ 
by (intro PiM-cong (simp-all add: N-def N'-def))
finally show ?thesis by simp
qed
end

```

10 Independent families of events, event sets, and random variables

theory *Independent-Family*
imports *Infinite-Product-Measure*
begin

definition (in *prob-space*)
 $\text{indep-sets } F \ I \longleftrightarrow (\forall i \in I. F \ i \subseteq \text{events}) \wedge$
 $(\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow (\forall A \in \text{Pi } J \ F. \text{prob } (\bigcap_{j \in J}. A \ j) = (\prod_{j \in J}. \text{prob } (A \ j))))$

definition (in *prob-space*)
 $\text{indep-set } A \ B \longleftrightarrow \text{indep-sets } (\text{case-bool } A \ B) \ \text{UNIV}$

definition (in *prob-space*)
 $\text{indep-events-def-alt: indep-events } A \ I \longleftrightarrow \text{indep-sets } (\lambda i. \{A \ i\}) \ I$

lemma (in *prob-space*) *indep-events-def*:
 $\text{indep-events } A \ I \longleftrightarrow (A \cdot I \subseteq \text{events}) \wedge$
 $(\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{prob } (\bigcap_{j \in J}. A \ j) = (\prod_{j \in J}. \text{prob } (A \ j)))$
unfolding *indep-events-def-alt indep-sets-def*
apply (*simp add: Ball-def Pi-iff image-subset-iff-funcset*)
apply (*intro conj-cong refl arg-cong[where f=All] ext imp-cong*)
apply *auto*
done

lemma (in *prob-space*) *indep-eventsI*:
 $(\bigwedge i. i \in I \implies F \ i \in \text{sets } M) \implies (\bigwedge J. J \subseteq I \implies \text{finite } J \implies J \neq \{\} \implies \text{prob } (\bigcap_{i \in J}. F \ i) = (\prod_{i \in J}. \text{prob } (F \ i))) \implies \text{indep-events } F \ I$
by (*auto simp: indep-events-def*)

definition (in *prob-space*)
 $\text{indep-event } A \ B \longleftrightarrow \text{indep-events } (\text{case-bool } A \ B) \ \text{UNIV}$

lemma (in *prob-space*) *indep-sets-cong*:
 $I = J \implies (\bigwedge i. i \in I \implies F \ i = G \ i) \implies \text{indep-sets } F \ I \longleftrightarrow \text{indep-sets } G \ J$
by (*simp add: indep-sets-def, intro conj-cong all-cong imp-cong ball-cong*) *blast+*

lemma (in *prob-space*) *indep-events-finite-index-events*:
 $\text{indep-events } F \ I \longleftrightarrow (\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-events } F \ J)$
by (*auto simp: indep-events-def*)

lemma (in *prob-space*) *indep-sets-finite-index-sets*:
 $\text{indep-sets } F \ I \longleftrightarrow (\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-sets } F \ J)$
proof (*intro iffI allI impI*)
assume *: $\forall J \subseteq I. J \neq \{\} \longrightarrow \text{finite } J \longrightarrow \text{indep-sets } F \ J$
show $\text{indep-sets } F \ I$ **unfolding** *indep-sets-def*
proof (*intro conjI ballI allI impI*)

```

fix i assume i ∈ I
with *[THEN spec, of {i}] show F i ⊆ events
  by (auto simp: indep-sets-def)
qed (insert *, auto simp: indep-sets-def)
qed (auto simp: indep-sets-def)

```

```

lemma (in prob-space) indep-sets-mono-index:
  J ⊆ I ⇒ indep-sets F I ⇒ indep-sets F J
unfolding indep-sets-def by auto

```

```

lemma (in prob-space) indep-sets-mono-sets:
  assumes indep: indep-sets F I
  assumes mono: ⋀i. i ∈ I ⇒ G i ⊆ F i
  shows indep-sets G I
proof -
  have (∀ i ∈ I. F i ⊆ events) ⇒ (∀ i ∈ I. G i ⊆ events)
    using mono by auto
  moreover have ⋀A J. J ⊆ I ⇒ A ∈ (Π j ∈ J. G j) ⇒ A ∈ (Π j ∈ J. F j)
    using mono by (auto simp: Pi-iff)
  ultimately show ?thesis
    using indep by (auto simp: indep-sets-def)
qed

```

```

lemma (in prob-space) indep-sets-mono:
  assumes indep: indep-sets F I
  assumes mono: J ⊆ I ⋀i. i ∈ J ⇒ G i ⊆ F i
  shows indep-sets G J
  apply (rule indep-sets-mono-sets)
  apply (rule indep-sets-mono-index)
  apply (fact +)
done

```

```

lemma (in prob-space) indep-setsI:
  assumes ⋀i. i ∈ I ⇒ F i ⊆ events
  and ⋀A J. J ≠ {} ⇒ J ⊆ I ⇒ finite J ⇒ (∀ j ∈ J. A j ∈ F j) ⇒ prob
    (⋂ j ∈ J. A j) = (⋂ j ∈ J. prob (A j))
  shows indep-sets F I
  using assms unfolding indep-sets-def by (auto simp: Pi-iff)

```

```

lemma (in prob-space) indep-setsD:
  assumes indep-sets F I and J ⊆ I J ≠ {} finite J ∀ j ∈ J. A j ∈ F j
  shows prob (⋂ j ∈ J. A j) = (⋂ j ∈ J. prob (A j))
  using assms unfolding indep-sets-def by auto

```

```

lemma (in prob-space) indep-setI:
  assumes ev: A ⊆ events B ⊆ events
  and indep: ⋀a b. a ∈ A ⇒ b ∈ B ⇒ prob (a ∩ b) = prob a * prob b
  shows indep-set A B
unfolding indep-set-def

```

```

proof (rule indep-setsI)
  fix  $F J$  assume  $J \neq \{\}$   $J \subseteq UNIV$ 
  and  $F: \forall j \in J. F j \in (\text{case } j \text{ of } True \Rightarrow A \mid False \Rightarrow B)$ 
  have  $J \in Pow\ UNIV$  by auto
  with  $F \langle J \neq \{\} \rangle$  indep[of  $F\ True\ F\ False$ ]
  show  $prob (\bigcap_{j \in J}. F j) = (\prod_{j \in J}. prob (F j))$ 
  unfolding UNIV-bool Pow-insert by (auto simp: ac-simps)
qed (auto split: bool.split simp: ev)

lemma (in prob-space) indep-setD:
  assumes indep: indep-set  $A\ B$  and  $ev: a \in A\ b \in B$ 
  shows  $prob (a \cap b) = prob\ a * prob\ b$ 
  using indep[unfolded indep-set-def, THEN indep-setsD, of UNIV case-bool  $a\ b$ ]
   $ev$ 
  by (simp add: ac-simps UNIV-bool)

lemma (in prob-space)
  assumes indep: indep-set  $A\ B$ 
  shows indep-setD-ev1:  $A \subseteq events$ 
  and indep-setD-ev2:  $B \subseteq events$ 
  using indep unfolding indep-set-def indep-sets-def UNIV-bool by auto

lemma (in prob-space) indep-sets-Dynkin:
  assumes indep: indep-sets  $F\ I$ 
  shows indep-sets  $(\lambda i. Dynkin\ (space\ M)\ (F\ i))\ I$ 
  (is indep-sets ?F I)
proof (subst indep-sets-finite-index-sets, intro allI impI ballI)
  fix  $J$  assume finite  $J\ J \subseteq I\ J \neq \{\}$ 
  with indep have indep-sets  $F\ J$ 
  by (subst (asm) indep-sets-finite-index-sets) auto
  { fix  $J\ K$  assume indep-sets  $F\ K$ 
    let  $?G = \lambda S\ i. \text{if } i \in S \text{ then } ?F\ i \text{ else } F\ i$ 
    assume finite  $J\ J \subseteq K$ 
    then have indep-sets  $(?G\ J)\ K$ 
    proof induct
      case (insert  $j\ J$ )
      moreover define  $G$  where  $G = ?G\ J$ 
      ultimately have  $G: indep-sets\ G\ K \wedge i. i \in K \implies G\ i \subseteq events$  and  $j \in K$ 
      by (auto simp: indep-sets-def)
      let  $?D = \{E \in events. indep-sets\ (G(j := \{E\}))\ K\}$ 
      { fix  $X$  assume  $X: X \in events$ 
        assume indep:  $\bigwedge J\ A. J \neq \{\} \implies J \subseteq K \implies finite\ J \implies j \notin J \implies (\forall i \in J. A\ i \in G\ i)$ 
         $\implies prob ((\bigcap_{i \in J}. A\ i) \cap X) = prob\ X * (\prod_{i \in J}. prob\ (A\ i))$ 
        have indep-sets  $(G(j := \{X\}))\ K$ 
        proof (rule indep-setsI)
          fix  $i$  assume  $i \in K$  then show  $(G(j := \{X\}))\ i \subseteq events$ 
          using  $G\ X$  by auto
        next

```



```

fix A J assume J: J ≠ {} J ⊆ K finite J ∀ i ∈ J. A i ∈ (G(j := {X})) i
show prob (⋂ j ∈ J. A j) = (⋂ j ∈ J. prob (A j))
proof cases
  assume j ∈ J
  with J have A j = X by auto
  show ?thesis
  proof cases
    assume J = {j} then show ?thesis by simp
  next
    assume J ≠ {j}
    have prob (⋂ i ∈ J. A i) = prob ((⋂ i ∈ J - {j}. A i) ∩ X)
    using ⟨j ∈ J⟩ ⟨A j = X⟩ by (auto intro!: arg-cong[where f=prob])
split: if-split-asm)
    also have ... = prob X * (⋂ i ∈ J - {j}. prob (A i))
  proof (rule indep)
    show J - {j} ≠ {} J - {j} ⊆ K finite (J - {j}) j ∉ J - {j}
    using J ⟨J ≠ {j}⟩ ⟨j ∈ J⟩ by auto
    show ∀ i ∈ J - {j}. A i ∈ G i
    using J by auto
  qed
  also have ... = prob (A j) * (⋂ i ∈ J - {j}. prob (A i))
  using ⟨A j = X⟩ by simp
  also have ... = (⋂ i ∈ J. prob (A i))
  unfolding prod.insert-remove[OF ⟨finite J⟩, symmetric, of λi. prob
(A i)]
  using ⟨j ∈ J⟩ by (simp add: insert-absorb)
  finally show ?thesis .
qed
next
  assume j ∉ J
  with J have ∀ i ∈ J. A i ∈ G i by (auto split: if-split-asm)
  with J show ?thesis
  by (intro indep-setsD[OF G(1)]) auto
qed
qed }
note indep-sets-insert = this
have Dynkin-system (space M) ?D
proof (rule Dynkin-systemI', simp-all cong del: indep-sets-cong, safe)
  show indep-sets (G(j := {{}))) K
  by (rule indep-sets-insert) auto
next
fix X assume X: X ∈ events and G': indep-sets (G(j := {X})) K
show indep-sets (G(j := {space M - X})) K
proof (rule indep-sets-insert)
  fix J A assume J: J ≠ {} J ⊆ K finite J j ∉ J and A: ∀ i ∈ J. A i ∈ G i
  then have A-sets: ⋀ i. i ∈ J ⇒ A i ∈ events
  using G by auto
  have prob ((⋂ j ∈ J. A j) ∩ (space M - X)) =
    prob ((⋂ j ∈ J. A j) - (⋂ i ∈ insert j J. (A(j := X)) i))

```

```

    using A-sets sets.sets-into-space[of - M] X ⟨J ≠ {}⟩
    by (auto intro!: arg-cong[where f=prob] split: if-split-asm)
  also have ... = prob (⋂ j∈J. A j) - prob (⋂ i∈insert j J. (A(j := X)) i)
    using J ⟨J ≠ {}⟩ ⟨j ∉ J⟩ A-sets X sets.sets-into-space
    by (auto intro!: finite-measure-Diff sets.finite-INT split: if-split-asm)
  finally have prob ((⋂ j∈J. A j) ∩ (space M - X)) =
    prob (⋂ j∈J. A j) - prob (⋂ i∈insert j J. (A(j := X)) i) .
  moreover {
    have prob (⋂ j∈J. A j) = (⋂ j∈J. prob (A j))
      using J A ⟨finite J⟩ by (intro indep-setsD[OF G(1)]) auto
    then have prob (⋂ j∈J. A j) = prob (space M) * (⋂ i∈J. prob (A i))
      using prob-space by simp }
  moreover {
    have prob (⋂ i∈insert j J. (A(j := X)) i) = (⋂ i∈insert j J. prob ((A(j
:= X)) i))
      using J A ⟨j ∈ K⟩ by (intro indep-setsD[OF G']) auto
    then have prob (⋂ i∈insert j J. (A(j := X)) i) = prob X * (⋂ i∈J.
prob (A i))
      using ⟨finite J⟩ ⟨j ∉ J⟩ by (auto intro!: prod.cong) }
    ultimately have prob ((⋂ j∈J. A j) ∩ (space M - X)) = (prob (space
M) - prob X) * (⋂ i∈J. prob (A i))
      by (simp add: field-simps)
    also have ... = prob (space M - X) * (⋂ i∈J. prob (A i))
      using X A by (simp add: finite-measure-compl)
    finally show prob ((⋂ j∈J. A j) ∩ (space M - X)) = prob (space M -
X) * (⋂ i∈J. prob (A i)) .
    qed (insert X, auto)
  next
    fix F :: nat ⇒ 'a set assume disj: disjoint-family F and range F ⊆ ?D
    then have F: ⋀ i. F i ∈ events ⋀ i. indep-sets (G(j:={F i})) K by auto
    show indep-sets (G(j := ⋃ k. F k)) K
    proof (rule indep-sets-insert)
      fix J A assume J: j ∉ J J ≠ {} J ⊆ K finite J and A: ∀ i∈J. A i ∈ G i
      then have A-sets: ⋀ i. i∈J ⇒ A i ∈ events
        using G by auto
      have prob ((⋂ j∈J. A j) ∩ (⋃ k. F k)) = prob (⋃ k. (⋂ i∈insert j J. (A(j
:= F k)) i))
        using ⟨J ≠ {}⟩ ⟨j ∉ J⟩ ⟨j ∈ K⟩ by (auto intro!: arg-cong[where f=prob]
split: if-split-asm)
      moreover have (λk. prob (⋂ i∈insert j J. (A(j := F k)) i)) sums prob
(⋃ k. (⋂ i∈insert j J. (A(j := F k)) i))
        proof (rule finite-measure-UNION)
          show disjoint-family (λk. ⋂ i∈insert j J. (A(j := F k)) i)
            using disj by (rule disjoint-family-on-bisimulation) auto
          show range (λk. ⋂ i∈insert j J. (A(j := F k)) i) ⊆ events
            using A-sets F ⟨finite J⟩ ⟨J ≠ {}⟩ ⟨j ∉ J⟩ by (auto intro!: sets.Int)
        qed
      moreover { fix k
        from J A ⟨j ∈ K⟩ have prob (⋂ i∈insert j J. (A(j := F k)) i) = prob

```

```

(F k) * (∏ i ∈ J. prob (A i))
  by (subst indep-setsD[OF F(2)]) (auto intro!: prod.cong split: if-split-asm)
  also have ... = prob (F k) * prob (∏ i ∈ J. A i)
    using J A ⟨j ∈ K⟩ by (subst indep-setsD[OF G(1)]) auto
  finally have prob (∏ i ∈ insert j J. (A(j := F k)) i) = prob (F k) * prob
(∏ i ∈ J. A i) . }
  ultimately have (λk. prob (F k) * prob (∏ i ∈ J. A i)) sums (prob ((∏ j ∈ J.
A j) ∩ (∪ k. F k)))
    by simp
  moreover
  have (λk. prob (F k) * prob (∏ i ∈ J. A i)) sums (prob (∪ k. F k) * prob
(∏ i ∈ J. A i))
    using disj F(1) by (intro finite-measure-UNION sums-mult2) auto
  then have (λk. prob (F k) * prob (∏ i ∈ J. A i)) sums (prob (∪ k. F k) *
(∏ i ∈ J. prob (A i)))
    using J A ⟨j ∈ K⟩ by (subst indep-setsD[OF G(1), symmetric]) auto
  ultimately
  show prob ((∏ j ∈ J. A j) ∩ (∪ k. F k)) = prob (∪ k. F k) * (∏ j ∈ J. prob
(A j))
    by (auto dest!: sums-unique)
  qed (insert F, auto)
qed (insert sets.sets-into-space, auto)
then have mono: Dynkin (space M) (G j) ⊆ {E ∈ events. indep-sets (G(j
:= {E})) K}
proof (rule Dynkin-system.Dynkin-subset, safe)
  fix X assume X ∈ G j
  then show X ∈ events using G ⟨j ∈ K⟩ by auto
  from ⟨indep-sets G K⟩
  show indep-sets (G(j := {X})) K
    by (rule indep-sets-mono-sets) (insert ⟨X ∈ G j⟩, auto)
qed
have indep-sets (G(j := ?D)) K
proof (rule indep-setsI)
  fix i assume i ∈ K then show (G(j := ?D)) i ⊆ events
    using G(2) by auto
next
fix A J assume J: J ≠ {} J ⊆ K finite J and A: ∀ i ∈ J. A i ∈ (G(j := ?D))
i
show prob (∏ j ∈ J. A j) = (∏ j ∈ J. prob (A j))
proof cases
  assume j ∈ J
  with A have indep: indep-sets (G(j := {A j})) K by auto
  from J A show ?thesis
    by (intro indep-setsD[OF indep]) auto
next
  assume j ∉ J
  with J A have ∀ i ∈ J. A i ∈ G i by (auto split: if-split-asm)
  with J show ?thesis
    by (intro indep-setsD[OF G(1)]) auto

```

```

    qed
  qed
  then have indep-sets (G(j := Dynkin (space M) (G j))) K
    by (rule indep-sets-mono-sets) (insert mono, auto)
  then show ?case
    by (rule indep-sets-mono-sets) (insert ⟨j ∈ K⟩ ⟨j ∉ J⟩, auto simp: G-def)
  qed (insert ⟨indep-sets F K⟩, simp) }
from this[OF ⟨indep-sets F J⟩ ⟨finite J⟩ subset-refl]
show indep-sets ?F J
  by (rule indep-sets-mono-sets) auto
qed

```

```

lemma (in prob-space) indep-sets-sigma:
  assumes indep: indep-sets F I
  assumes stable:  $\bigwedge i. i \in I \implies \text{Int-stable } (F i)$ 
  shows indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I
proof -
  from indep-sets-Dynkin[OF indep]
  show ?thesis
  proof (rule indep-sets-mono-sets, subst sigma-eq-Dynkin, simp-all add: stable)
    fix i assume i ∈ I
    with indep have F i ⊆ events by (auto simp: indep-sets-def)
    with sets.sets-into-space show F i ⊆ Pow (space M) by auto
  qed
qed

```

```

lemma (in prob-space) indep-sets-sigma-sets-iff:
  assumes  $\bigwedge i. i \in I \implies \text{Int-stable } (F i)$ 
  shows indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I  $\longleftrightarrow$  indep-sets F I
proof
  assume indep-sets F I then show indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I
    by (rule indep-sets-sigma) fact
  next
    assume indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) (F i)$ ) I then show indep-sets F I
      by (rule indep-sets-mono-sets) (intro subsetI sigma-sets.Basic)
qed

```

```

definition (in prob-space)
  indep-vars-def2: indep-vars M' X I  $\longleftrightarrow$ 
    ( $\forall i \in I. \text{random-variable } (M' i) (X i)$ )  $\wedge$ 
    indep-sets ( $\lambda i. \{ X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (M' i) \}$ ) I

```

```

definition (in prob-space)
  indep-var Ma A Mb B  $\longleftrightarrow$  indep-vars (case-bool Ma Mb) (case-bool A B) UNIV

```

```

lemma (in prob-space) indep-vars-def:
  indep-vars M' X I  $\longleftrightarrow$ 
    ( $\forall i \in I. \text{random-variable } (M' i) (X i)$ )  $\wedge$ 
    indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) \{ X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (M' i) \}$ ) I

```

```

i)) I
  unfolding indep-vars-def2
  apply (rule conj-cong[OF refl])
  apply (rule indep-sets-sigma-sets-iff[symmetric])
  apply (auto simp: Int-stable-def)
  apply (rule-tac x=A ∩ Aa in exI)
  apply auto
  done

lemma (in prob-space) indep-var-eq:
  indep-var S X T Y  $\longleftrightarrow$ 
    (random-variable S X  $\wedge$  random-variable T Y)  $\wedge$ 
    indep-set
      (sigma-sets (space M) { X - ' A  $\cap$  space M | A. A  $\in$  sets S })
      (sigma-sets (space M) { Y - ' A  $\cap$  space M | A. A  $\in$  sets T })
  unfolding indep-var-def indep-vars-def indep-set-def UNIV-bool
  by (intro arg-cong2[where f=( $\wedge$ )] arg-cong2[where f=indep-sets] ext)
    (auto split: bool.split)

```

```

lemma (in prob-space) indep-sets2-eq:
  indep-set A B  $\longleftrightarrow$  A  $\subseteq$  events  $\wedge$  B  $\subseteq$  events  $\wedge$  ( $\forall a \in A. \forall b \in B. \text{prob } (a \cap b) =$ 
  prob a * prob b)
  unfolding indep-set-def
  proof (intro iffI ballI conjI)
    assume indep: indep-sets (case-bool A B) UNIV
    { fix a b assume a  $\in$  A b  $\in$  B
      with indep-setsD[OF indep, of UNIV case-bool a b]
      show prob (a  $\cap$  b) = prob a * prob b
      unfolding UNIV-bool by (simp add: ac-simps) }
    from indep show A  $\subseteq$  events B  $\subseteq$  events
    unfolding indep-sets-def UNIV-bool by auto
  next
    assume *: A  $\subseteq$  events  $\wedge$  B  $\subseteq$  events  $\wedge$  ( $\forall a \in A. \forall b \in B. \text{prob } (a \cap b) = \text{prob } a *$ 
    prob b)
    show indep-sets (case-bool A B) UNIV
    proof (rule indep-setsI)
      fix i show (case i of True  $\Rightarrow$  A | False  $\Rightarrow$  B)  $\subseteq$  events
      using * by (auto split: bool.split)
    next
      fix J X assume J  $\neq$  {} J  $\subseteq$  UNIV and X:  $\forall j \in J. X j \in$  (case j of True  $\Rightarrow$  A
      | False  $\Rightarrow$  B)
      then have J = {True}  $\vee$  J = {False}  $\vee$  J = {True, False}
      by (auto simp: UNIV-bool)
      then show prob ( $\bigcap j \in J. X j$ ) = ( $\prod j \in J. \text{prob } (X j)$ )
      using X * by auto
    qed
  qed

```

```

lemma (in prob-space) indep-set-sigma-sets:

```

```

assumes indep-set  $A\ B$ 
assumes  $A$ : Int-stable  $A$  and  $B$ : Int-stable  $B$ 
shows indep-set (sigma-sets (space  $M$ )  $A$ ) (sigma-sets (space  $M$ )  $B$ )
proof –
  have indep-sets ( $\lambda i.$  sigma-sets (space  $M$ ) (case  $i$  of True  $\Rightarrow A$  | False  $\Rightarrow B$ ))
UNIV
proof (rule indep-sets-sigma)
  show indep-sets (case-bool  $A\ B$ ) UNIV
  by (rule (indep-set  $A\ B$ )[unfolded indep-set-def])
  fix  $i$  show Int-stable (case  $i$  of True  $\Rightarrow A$  | False  $\Rightarrow B$ )
  using  $A\ B$  by (cases  $i$ ) auto
qed
then show ?thesis
  unfolding indep-set-def
  by (rule indep-sets-mono-sets) (auto split: bool.split)
qed

```

```

lemma (in prob-space) indep-eventsI-indep-vars:
assumes indep: indep-vars  $N\ X\ I$ 
assumes  $P$ :  $\bigwedge i. i \in I \Rightarrow \{x \in \text{space } (N\ i). P\ i\ x\} \in \text{sets } (N\ i)$ 
shows indep-events ( $\lambda i. \{x \in \text{space } M. P\ i\ (X\ i\ x)\}$ )  $I$ 
proof –
  have indep-sets ( $\lambda i. \{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (N\ i)\}$ )  $I$ 
  using indep unfolding indep-vars-def2 by auto
  then show ?thesis
  unfolding indep-events-def-alt
  proof (rule indep-sets-mono-sets)
    fix  $i$  assume  $i \in I$ 
    then have  $\{x \in \text{space } M. P\ i\ (X\ i\ x)\} = \{X\ i - '\{x \in \text{space } (N\ i). P\ i\ x\} \cap \text{space } M\}$ 
    using indep by (auto simp: indep-vars-def dest: measurable-space)
    also have  $\dots \subseteq \{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (N\ i)\}$ 
    using  $P[OF\ \langle i \in I \rangle]$  by blast
    finally show  $\{x \in \text{space } M. P\ i\ (X\ i\ x)\} \subseteq \{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (N\ i)\}$  .
  qed
qed

```

```

lemma (in prob-space) indep-sets-collect-sigma:
fixes  $I :: 'j \Rightarrow 'i\ \text{set}$  and  $J :: 'j\ \text{set}$  and  $E :: 'i \Rightarrow 'a\ \text{set set}$ 
assumes indep: indep-sets  $E\ (\bigcup j \in J. I\ j)$ 
assumes Int-stable:  $\bigwedge i\ j. j \in J \Rightarrow i \in I\ j \Rightarrow \text{Int-stable } (E\ i)$ 
assumes disjoint: disjoint-family-on  $I\ J$ 
shows indep-sets ( $\lambda j. \text{sigma-sets } (\text{space } M) (\bigcup i \in I\ j. E\ i)$ )  $J$ 
proof –
  let ? $E = \lambda j. \{\bigcap k \in K. E'\ k \mid E'\ K. \text{finite } K \wedge K \neq \{\} \wedge K \subseteq I\ j \wedge (\forall k \in K. E'\ k \in E\ k)\}$ 

```

from indep **have** $E: \bigwedge j\ i. j \in J \Rightarrow i \in I\ j \Rightarrow E\ i \subseteq \text{events}$

```

    unfolding indep-sets-def by auto
  { fix j
    let ?S = sigma-sets (space M) (⋃ i ∈ I j. E i)
    assume j ∈ J
    from E[OF this] interpret S: sigma-algebra space M ?S
    using sets.sets-into-space[of - M] by (intro sigma-algebra-sigma-sets) auto

    have sigma-sets (space M) (⋃ i ∈ I j. E i) = sigma-sets (space M) (?E j)
  proof (rule sigma-sets-eqI)
    fix A assume A ∈ (⋃ i ∈ I j. E i)
    then obtain i where i ∈ I j A ∈ E i ..
    then show A ∈ sigma-sets (space M) (?E j)
      by (auto intro!: sigma-sets.intros(2-) exI[of - {i}] exI[of - λi. A])
  next
    fix A assume A ∈ ?E j
    then obtain E' K where finite K K ≠ {} K ⊆ I j ∧ k ∈ K ⟹ E' k ∈
E k
      and A: A = (⋂ k ∈ K. E' k)
      by auto
    then have A ∈ ?S unfolding A
      by (safe intro!: S.finite-INT) auto
    then show A ∈ sigma-sets (space M) (⋃ i ∈ I j. E i)
      by simp
  qed }
  moreover have indep-sets (λj. sigma-sets (space M) (?E j)) J
proof (rule indep-sets-sigma)
  show indep-sets ?E J
proof (intro indep-setsI)
  fix j assume j ∈ J with E show ?E j ⊆ events by (force intro!: sets.finite-INT)
next
  fix K A assume K: K ≠ {} K ⊆ J finite K
  and ∀ j ∈ K. A j ∈ ?E j
  then have ∀ j ∈ K. ∃ E' L. A j = (⋂ l ∈ L. E' l) ∧ finite L ∧ L ≠ {} ∧ L ⊆ I
j ∧ (∀ l ∈ L. E' l ∈ E l)
    by simp
  from bchoice[OF this] obtain E'
  where ∀ x ∈ K. ∃ L. A x = ⋂ (E' x ' L) ∧ finite L ∧ L ≠ {} ∧ L ⊆ I x ∧
(∀ l ∈ L. E' x l ∈ E l)
    ..
  from bchoice[OF this] obtain L
  where A: ⋀ j. j ∈ K ⟹ A j = (⋂ l ∈ L j. E' j l)
  and L: ⋀ j. j ∈ K ⟹ finite (L j) ⋀ j. j ∈ K ⟹ L j ≠ {} ⋀ j. j ∈ K ⟹ L j
⊆ I j
  and E': ⋀ j l. j ∈ K ⟹ l ∈ L j ⟹ E' j l ∈ E l
  by auto
  { fix k l j assume k ∈ K j ∈ K l ∈ L j l ∈ L k
    have k = j
    proof (rule ccontr)
      assume k ≠ j

```

```

with disjoint  $\langle K \subseteq J \rangle \langle k \in K \rangle \langle j \in K \rangle$  have  $I\ k \cap I\ j = \{\}$ 
  unfolding disjoint-family-on-def by auto
with  $L(2,3)[OF\ \langle j \in K \rangle]$   $L(2,3)[OF\ \langle k \in K \rangle]$ 
show False using  $\langle l \in L\ k \rangle \langle l \in L\ j \rangle$  by auto
qed }
note  $L\text{-inj} = \text{this}$ 

define  $k$  where  $k\ l = (SOME\ k. k \in K \wedge l \in L\ k)$  for  $l$ 
{ fix  $x\ j\ l$  assume  $*$ :  $j \in K\ l \in L\ j$ 
  have  $k\ l = j$  unfolding  $k\text{-def}$ 
  proof (rule some-equality)
    fix  $k$  assume  $k \in K \wedge l \in L\ k$ 
    with  $*$   $L\text{-inj}$  show  $k = j$  by auto
  qed (insert  $*$ , simp) }
note  $k\text{-simp}[simp] = \text{this}$ 
let  $?E' = \lambda l. E'\ (k\ l)\ l$ 
have  $prob\ (\bigcap_{j \in K}. A\ j) = prob\ (\bigcap_{l \in (\bigcup_{k \in K}. L\ k)}. ?E'\ l)$ 
  by (auto simp: A intro!: arg-cong[where  $f = prob$ ])
also have  $\dots = (\prod_{l \in (\bigcup_{k \in K}. L\ k)}. prob\ (?E'\ l))$ 
  using  $L\ K\ E'$  by (intro indep-setsD[OF indep]) (simp-all add: UN-mono)
also have  $\dots = (\prod_{j \in K}. \prod_{l \in L\ j}. prob\ (E'\ j\ l))$ 
  using  $K\ L\ L\text{-inj}$  by (subst prod.UNION-disjoint) auto
also have  $\dots = (\prod_{j \in K}. prob\ (A\ j))$ 
  using  $K\ L\ E'$  by (auto simp add: A intro!: prod.cong indep-setsD[OF indep,
symmetric]) blast
finally show  $prob\ (\bigcap_{j \in K}. A\ j) = (\prod_{j \in K}. prob\ (A\ j))$  .
qed
next
fix  $j$  assume  $j \in J$ 
show Int-stable  $(?E\ j)$ 
proof (rule Int-stableI)
  fix  $a$  assume  $a \in ?E\ j$  then obtain  $Ka\ Ea$ 
    where  $a$ :  $a = (\bigcap_{k \in Ka}. Ea\ k)$  finite  $Ka\ Ka \neq \{\}$   $Ka \subseteq I\ j \wedge k. k \in Ka \implies$ 
 $Ea\ k \in E\ k$  by auto
  fix  $b$  assume  $b \in ?E\ j$  then obtain  $Kb\ Eb$ 
    where  $b$ :  $b = (\bigcap_{k \in Kb}. Eb\ k)$  finite  $Kb\ Kb \neq \{\}$   $Kb \subseteq I\ j \wedge k. k \in Kb \implies$ 
 $Eb\ k \in E\ k$  by auto
  let  $?f = \lambda k. (if\ k \in Ka \cap Kb\ \text{then}\ Ea\ k \cap Eb\ k\ \text{else}\ if\ k \in Kb\ \text{then}\ Eb\ k\ \text{else}\ if\ k \in Ka\ \text{then}\ Ea\ k\ \text{else}\ \{\})$ 
  have  $Ka \cup Kb = (Ka \cap Kb) \cup (Kb - Ka) \cup (Ka - Kb)$ 
    by blast
  moreover have  $(\bigcap_{x \in Ka \cap Kb}. Ea\ x \cap Eb\ x) \cap$ 
 $(\bigcap_{x \in Kb - Ka}. Eb\ x) \cap (\bigcap_{x \in Ka - Kb}. Ea\ x) = (\bigcap_{k \in Ka}. Ea\ k) \cap (\bigcap_{k \in Kb.}$ 
 $Eb\ k)$ 
    by auto
  ultimately have  $(\bigcap_{k \in Ka \cup Kb}. ?f\ k) = (\bigcap_{k \in Ka}. Ea\ k) \cap (\bigcap_{k \in Kb}. Eb\ k)$ 
(is  $?lhs = ?rhs)$ 
    by (simp only: image-Un Inter-Un-distrib) simp
  then have  $a \cap b = (\bigcap_{k \in Ka \cup Kb}. ?f\ k)$ 

```



```

    by (simp only: a(1) b(1))
  with a b ⟨j ∈ J⟩ Int-stableD[OF Int-stable] show a ∩ b ∈ ?E j
    by (intro CollectI exI[of - Ka ∪ Kb] exI[of - ?f]) auto
qed
qed
ultimately show ?thesis
  by (simp cong: indep-sets-cong)
qed

lemma (in prob-space) indep-vars-restrict:
  assumes ind: indep-vars M' X I and K:  $\bigwedge j. j \in L \implies K j \subseteq I$  and J:
    disjoint-family-on K L
  shows indep-vars ( $\lambda j. PiM (K j) M'$ ) ( $\lambda j \omega. restrict (\lambda i. X i \omega) (K j)$ ) L
  unfolding indep-vars-def
proof safe
  fix j assume j ∈ L then show random-variable ( $Pi_M (K j) M'$ ) ( $\lambda \omega. \lambda i \in K j. X i \omega$ )
  using K ind by (auto simp: indep-vars-def intro!: measurable-restrict)
next
  have X:  $\bigwedge i. i \in I \implies X i \in measurable M (M' i)$ 
  using ind by (auto simp: indep-vars-def)
  let ?proj =  $\lambda j S. \{(\lambda \omega. \lambda i \in K j. X i \omega) - ' A \cap space M \mid A. A \in S\}$ 
  let ?UN =  $\lambda j. sigma-sets (space M) (\bigcup i \in K j. \{ X i - ' A \cap space M \mid A. A \in sets (M' i) \})$ 
  show indep-sets ( $\lambda i. sigma-sets (space M) (?proj i (sets (Pi_M (K i) M')))$ ) L
  proof (rule indep-sets-mono-sets)
    fix j assume j: j ∈ L
    have sigma-sets (space M) (?proj j (sets (Pi_M (K j) M'))) =
      sigma-sets (space M) (sigma-sets (space M) (?proj j (prod-algebra (K j) M')))
    using j K X [THEN measurable-space] unfolding sets-PiM
    by (subst sigma-sets-vimage-commute) (auto simp add: Pi-iff)
    also have ... = sigma-sets (space M) (?proj j (prod-algebra (K j) M'))
    by (rule sigma-sets-sigma-sets-eq) auto
    also have ... ⊆ ?UN j
  proof (rule sigma-sets-mono, safe del: disjE elim!: prod-algebraE)
    fix J E assume J: finite J J ≠ {} ∨ K j = {} J ⊆ K j and E:  $\forall i. i \in J \implies E i \in sets (M' i)$ 
    show  $(\lambda \omega. \lambda i \in K j. X i \omega) - ' prod-emb (K j) M' J (Pi_E J E) \cap space M \in ?UN j$ 
  proof cases
    assume K j = {} with J show ?thesis
      by (auto simp add: sigma-sets-empty-eq prod-emb-def)
    next
      assume K j ≠ {} with J have J ≠ {}
      by auto
      { interpret sigma-algebra space M ?UN j
        by (rule sigma-algebra-sigma-sets) auto
        have  $\bigwedge A. (\bigwedge i. i \in J \implies A i \in ?UN j) \implies \bigcap (A - ' J) \in ?UN j$ 
        using ⟨finite J⟩ ⟨J ≠ {}⟩ by (rule finite-INT) blast }
      }
    }
  end
end

```

```

note INT = this

from  $\langle J \neq \{\} \rangle J K E$  [rule-format, THEN sets.sets-into-space] j
have  $(\lambda\omega. \lambda i \in K j. X i \omega) - ' prod-emb (K j) M' J (Pi_E J E) \cap space M$ 
  =  $(\bigcap i \in J. X i - ' E i \cap space M)$ 
  apply (subst prod-emb-PiE[OF -])
  apply auto []
  apply auto []
  apply (auto simp add: Pi-iff intro!: X[THEN measurable-space])
  apply (erule-tac x=i in ballE)
  apply auto
  done
also have  $\dots \in ?UN j$ 
  apply (rule INT)
  apply (rule sigma-sets.Basic)
  using  $\langle J \subseteq K j \rangle E$ 
  apply auto
  done
finally show ?thesis .
qed
qed
finally show sigma-sets (space M) (?proj j (sets (Pi_M (K j) M')))  $\subseteq ?UN j$  .
next
  show indep-sets ?UN L
  proof (rule indep-sets-collect-sigma)
    show indep-sets  $(\lambda i. \{X i - ' A \cap space M \mid A. A \in sets (M' i)\}) (\bigcup j \in L. K j)$ 
    proof (rule indep-sets-mono-index)
      show indep-sets  $(\lambda i. \{X i - ' A \cap space M \mid A. A \in sets (M' i)\}) I$ 
      using ind unfolding indep-vars-def2 by auto
      show  $(\bigcup l \in L. K l) \subseteq I$ 
      using K by auto
    qed
  next
    fix l i assume  $l \in L i \in K l$ 
    show Int-stable  $\{X i - ' A \cap space M \mid A. A \in sets (M' i)\}$ 
    apply (auto simp: Int-stable-def)
    apply (rule-tac x=A \cap Aa in exI)
    apply auto
    done
  qed fact
qed
qed

lemma (in prob-space) indep-var-restrict:
  assumes ind: indep-vars M' X I and AB: A \cap B = \{\}  $A \subseteq I B \subseteq I$ 
  shows indep-var  $(Pi M A M') (\lambda\omega. restrict (\lambda i. X i \omega) A) (Pi M B M') (\lambda\omega. restrict (\lambda i. X i \omega) B)$ 
proof –

```

```

have *:
  case-bool (PiM A M') (PiM B M') = (λb. PiM (case-bool A B b) M')
  case-bool (λω. λi∈A. X i ω) (λω. λi∈B. X i ω) = (λb ω. λi∈case-bool A B b.
X i ω)
  by (simp-all add: fun-eq-iff split: bool.split)
show ?thesis
  unfolding indep-var-def * using AB
  by (intro indep-vars-restrict[OF ind]) (auto simp: disjoint-family-on-def split:
bool.split)
qed

```

```

lemma (in prob-space) indep-vars-subset:
  assumes indep-vars M' X I J ⊆ I
  shows indep-vars M' X J
  using assms unfolding indep-vars-def indep-sets-def
  by auto

```

```

lemma (in prob-space) indep-vars-cong:
  I = J ⟹ (⋀i. i ∈ I ⟹ X i = Y i) ⟹ (⋀i. i ∈ I ⟹ M' i = N' i) ⟹
indep-vars M' X I ⟷ indep-vars N' Y J
  unfolding indep-vars-def2 by (intro conj-cong indep-sets-cong) auto

```

```

definition (in prob-space) tail-events where
  tail-events A = (⋂ n. sigma-sets (space M) (⋃ (A ' {n..})))

```

```

lemma (in prob-space) tail-events-sets:
  assumes A: ⋀i::nat. A i ⊆ events
  shows tail-events A ⊆ events
proof
  fix X assume X: X ∈ tail-events A
  let ?A = (⋂ n. sigma-sets (space M) (⋃ (A ' {n..})))
  from X have ⋀n::nat. X ∈ sigma-sets (space M) (⋃ (A ' {n..})) by (auto simp:
tail-events-def)
  from this[of 0] have X ∈ sigma-sets (space M) (⋃ (A ' UNIV)) by simp
  then show X ∈ events
    by induct (insert A, auto)
qed

```

```

lemma (in prob-space) sigma-algebra-tail-events:
  assumes ⋀i::nat. sigma-algebra (space M) (A i)
  shows sigma-algebra (space M) (tail-events A)
  unfolding tail-events-def
proof (simp add: sigma-algebra-iff2, safe)
  let ?A = (⋂ n. sigma-sets (space M) (⋃ (A ' {n..})))
  interpret A: sigma-algebra space M A i for i by fact
  { fix X x assume X ∈ ?A x ∈ X
    then have ⋀n. X ∈ sigma-sets (space M) (⋃ (A ' {n..})) by auto
    from this[of 0] have X ∈ sigma-sets (space M) (⋃ (A ' UNIV)) by simp
    then have X ⊆ space M

```

```

    by induct (insert A.sets-into-space, auto)
  with  $\langle x \in X \rangle$  show  $x \in \text{space } M$  by auto }
{ fix  $F :: \text{nat} \Rightarrow 'a \text{ set}$  and  $n$  assume  $\text{range } F \subseteq ?A$ 
  then show  $(\bigcup (F \text{ ' } UNIV)) \in \text{sigma-sets } (\text{space } M) (\bigcup (A \text{ ' } \{n..\}))$ 
    by (intro sigma-sets.Union) auto }
qed (auto intro!: sigma-sets.Compl sigma-sets.Empty)

```

lemma (in prob-space) kolmogorov-0-1-law:

```

fixes  $A :: \text{nat} \Rightarrow 'a \text{ set set}$ 
assumes  $\bigwedge i::\text{nat}. \text{sigma-algebra } (\text{space } M) (A \ i)$ 
assumes indep: indep-sets  $A \ UNIV$ 
and  $X: X \in \text{tail-events } A$ 
shows  $\text{prob } X = 0 \vee \text{prob } X = 1$ 

```

proof –

```

have  $A: \bigwedge i. A \ i \subseteq \text{events}$ 
  using indep unfolding indep-sets-def by simp

```

```

let  $?D = \{D \in \text{events}. \text{prob } (X \cap D) = \text{prob } X * \text{prob } D\}$ 
interpret  $A: \text{sigma-algebra } \text{space } M \ A \ i$  for  $i$  by fact
interpret  $T: \text{sigma-algebra } \text{space } M \ \text{tail-events } A$ 
  by (rule sigma-algebra-tail-events) fact
have  $X \subseteq \text{space } M$  using  $T.\text{space-closed } X$  by auto

```

```

have  $X\text{-in}: X \in \text{events}$ 
  using tail-events-sets  $A \ X$  by auto

```

interpret $D: \text{Dynkin-system } \text{space } M \ ?D$

proof (rule Dynkin-systemI)

```

  fix  $D$  assume  $D \in ?D$  then show  $D \subseteq \text{space } M$ 
    using sets.sets-into-space by auto

```

next

```

  show  $\text{space } M \in ?D$ 
    using prob-space  $\langle X \subseteq \text{space } M \rangle$  by (simp add: Int-absorb2)

```

next

```

  fix  $A$  assume  $A: A \in ?D$ 
  have  $\text{prob } (X \cap (\text{space } M - A)) = \text{prob } (X - (X \cap A))$ 
    using  $\langle X \subseteq \text{space } M \rangle$  by (auto intro!: arg-cong[where  $f=\text{prob}$ ])
  also have  $\dots = \text{prob } X - \text{prob } (X \cap A)$ 
    using  $X\text{-in } A$  by (intro finite-measure-Diff) auto
  also have  $\dots = \text{prob } X * \text{prob } (\text{space } M) - \text{prob } X * \text{prob } A$ 
    using  $A \text{ prob-space}$  by auto
  also have  $\dots = \text{prob } X * \text{prob } (\text{space } M - A)$ 
    using  $X\text{-in } A \ \text{sets.sets-into-space}$ 
    by (subst finite-measure-Diff) (auto simp: field-simps)
  finally show  $\text{space } M - A \in ?D$ 
    using  $A \ \langle X \subseteq \text{space } M \rangle$  by auto

```

next

```

  fix  $F :: \text{nat} \Rightarrow 'a \text{ set}$  assume  $\text{dis: disjoint-family } F$  and  $\text{range } F \subseteq ?D$ 
  then have  $F: \text{range } F \subseteq \text{events} \bigwedge i. \text{prob } (X \cap F \ i) = \text{prob } X * \text{prob } (F \ i)$ 

```

```

    by auto
  have (λi. prob (X ∩ F i)) sums prob (⋃ i. X ∩ F i)
proof (rule finite-measure-UNION)
  show range (λi. X ∩ F i) ⊆ events
    using F X-in by auto
  show disjoint-family (λi. X ∩ F i)
    using dis by (rule disjoint-family-on-bisimulation) auto
qed
with F have (λi. prob X * prob (F i)) sums prob (X ∩ (⋃ i. F i))
  by simp
moreover have (λi. prob X * prob (F i)) sums (prob X * prob (⋃ i. F i))
  by (intro sums-mult finite-measure-UNION F dis)
ultimately have prob (X ∩ (⋃ i. F i)) = prob X * prob (⋃ i. F i)
  by (auto dest!: sums-unique)
with F show (⋃ i. F i) ∈ ?D
  by auto
qed

{ fix n
  have indep-sets (λb. sigma-sets (space M) (⋃ m∈case-bool {..n} {Suc n..} b.
A m)) UNIV
proof (rule indep-sets-collect-sigma)
  have *: (⋃ b. case b of True ⇒ {..n} | False ⇒ {Suc n..}) = UNIV (is ?U
= -)
    by (simp split: bool.split add: set-eq-iff) (metis not-less-eq-eq)
  with indep show indep-sets A ?U by simp
  show disjoint-family (case-bool {..n} {Suc n..})
    unfolding disjoint-family-on-def by (auto split: bool.split)
  fix m
  show Int-stable (A m)
    unfolding Int-stable-def using A.Int by auto
qed
also have (λb. sigma-sets (space M) (⋃ m∈case-bool {..n} {Suc n..} b. A m))
=
  case-bool (sigma-sets (space M) (⋃ m∈{..n}. A m)) (sigma-sets (space M)
(⋃ m∈{Suc n..}. A m))
    by (auto intro!: ext split: bool.split)
  finally have indep: indep-set (sigma-sets (space M) (⋃ m∈{..n}. A m)) (sigma-sets
(space M) (⋃ m∈{Suc n..}. A m))
    unfolding indep-set-def by simp

have sigma-sets (space M) (⋃ m∈{..n}. A m) ⊆ ?D
proof (simp add: subset-eq, rule)
  fix D assume D: D ∈ sigma-sets (space M) (⋃ m∈{..n}. A m)
  have X ∈ sigma-sets (space M) (⋃ m∈{Suc n..}. A m)
    using X unfolding tail-events-def by simp
  from indep-setD[OF indep D this] indep-setD-ev1[OF indep] D
  show D ∈ events ∧ prob (X ∩ D) = prob X * prob D
    by (auto simp add: ac-simps)

```

```

    qed }
  then have  $(\bigcup n. \text{sigma-sets } (\text{space } M) (\bigcup m \in \{..n\}. A \ m)) \subseteq ?D$  (is  $?A \subseteq -$ )
    by auto

  note  $\langle X \in \text{tail-events } A \rangle$ 
  also {
    have  $\bigwedge n. \text{sigma-sets } (\text{space } M) (\bigcup i \in \{n.. \}. A \ i) \subseteq \text{sigma-sets } (\text{space } M) ?A$ 
      by (intro sigma-sets-subseteq UN-mono) auto
    then have  $\text{tail-events } A \subseteq \text{sigma-sets } (\text{space } M) ?A$ 
      unfolding tail-events-def by auto }
  also have  $\text{sigma-sets } (\text{space } M) ?A = \text{Dynkin } (\text{space } M) ?A$ 
  proof (rule sigma-eq-Dynkin)
    { fix B n assume  $B \in \text{sigma-sets } (\text{space } M) (\bigcup m \in \{..n\}. A \ m)$ 
      then have  $B \subseteq \text{space } M$ 
        by induct (insert A sets.sets-into-space[of - M], auto) }
    then show  $?A \subseteq \text{Pow } (\text{space } M)$  by auto
  show Int-stable ?A
  proof (rule Int-stableI)
    fix a b assume  $a \in ?A \ b \in ?A$  then obtain n m
      where  $a: n \in \text{UNIV } a \in \text{sigma-sets } (\text{space } M) (\bigcup (A \ ' \{..n\}))$ 
      and  $b: m \in \text{UNIV } b \in \text{sigma-sets } (\text{space } M) (\bigcup (A \ ' \{..m\}))$  by auto
    interpret Amn: sigma-algebra space M sigma-sets (space M)  $(\bigcup i \in \{.. \max m \ n\}. A \ i)$ 
    using A sets.sets-into-space[of - M] by (intro sigma-algebra-sigma-sets) auto
    have  $\text{sigma-sets } (\text{space } M) (\bigcup i \in \{..n\}. A \ i) \subseteq \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max m \ n\}. A \ i)$ 
      by (intro sigma-sets-subseteq UN-mono) auto
    with a have  $a \in \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max m \ n\}. A \ i)$  by auto
    moreover
    have  $\text{sigma-sets } (\text{space } M) (\bigcup i \in \{..m\}. A \ i) \subseteq \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max m \ n\}. A \ i)$ 
      by (intro sigma-sets-subseteq UN-mono) auto
    with b have  $b \in \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max m \ n\}. A \ i)$  by auto
    ultimately have  $a \cap b \in \text{sigma-sets } (\text{space } M) (\bigcup i \in \{.. \max m \ n\}. A \ i)$ 
      using Amn.Int[of a b] by simp
    then show  $a \cap b \in (\bigcup n. \text{sigma-sets } (\text{space } M) (\bigcup i \in \{..n\}. A \ i))$  by auto
  qed
  qed
  also have  $\text{Dynkin } (\text{space } M) ?A \subseteq ?D$ 
    using  $\langle ?A \subseteq ?D \rangle$  by (auto intro!: D.Dynkin-subset)
  finally show  $?thesis$  by auto
  qed

```

lemma (in prob-space) borel-0-1-law:

```

  fixes F :: nat  $\Rightarrow$  'a set
  assumes F2: indep-events F UNIV
  shows  $\text{prob } (\bigcap n. \bigcup m \in \{n.. \}. F \ m) = 0 \vee \text{prob } (\bigcap n. \bigcup m \in \{n.. \}. F \ m) = 1$ 
  proof (rule kolmogorov-0-1-law[of  $\lambda i. \text{sigma-sets } (\text{space } M) \{ F \ i \}$ ])
    have F1:  $\text{range } F \subseteq \text{events}$ 

```

```

    using F2 by (simp add: indep-events-def subset-eq)
  { fix i show sigma-algebra (space M) (sigma-sets (space M) {F i})
    using sigma-algebra-sigma-sets[of {F i} space M] F1 sets.sets-into-space
    by auto }
show indep-sets (λi. sigma-sets (space M) {F i}) UNIV
proof (rule indep-sets-sigma)
  show indep-sets (λi. {F i}) UNIV
    unfolding indep-events-def-alt[symmetric] by fact
  fix i show Int-stable {F i}
    unfolding Int-stable-def by simp
qed
let ?Q = λn. ⋃ i∈{n..}. F i
show (⋂ n. ⋃ m∈{n..}. F m) ∈ tail-events (λi. sigma-sets (space M) {F i})
  unfolding tail-events-def
proof
  fix j
  interpret S: sigma-algebra space M sigma-sets (space M) (⋃ i∈{j..}. sigma-sets
(space M) {F i})
    using order-trans[OF F1 sets.space-closed]
  by (intro sigma-algebra-sigma-sets) (simp add: sigma-sets-singleton subset-eq)
  have (⋂ n. ?Q n) = (⋂ n∈{j..}. ?Q n)
    by (intro decseq-SucI INT-decseq-offset UN-mono) auto
  also have ... ∈ sigma-sets (space M) (⋃ i∈{j..}. sigma-sets (space M) {F i})
    using order-trans[OF F1 sets.space-closed]
  by (safe intro!: S.countable-INT S.countable-UN)
    (auto simp: sigma-sets-singleton intro!: sigma-sets.Basic bexI)
  finally show (⋂ n. ?Q n) ∈ sigma-sets (space M) (⋃ i∈{j..}. sigma-sets (space
M) {F i})
    by simp
qed
qed

```

lemma (in prob-space) borel-0-1-law-AE:

```

  fixes P :: nat ⇒ 'a ⇒ bool
  assumes indep-events (λm. {x∈space M. P m x}) UNIV (is indep-events ?P -)
  shows (AE x in M. infinite {m. P m x}) ∨ (AE x in M. finite {m. P m x})
proof -
  have [measurable]: ⋂ m. {x∈space M. P m x} ∈ sets M
    using assms by (auto simp: indep-events-def)
  have *: (⋂ n. ⋃ m∈{n..}. {x ∈ space M. P m x}) ∈ events
    by simp
  from assms have prob (⋂ n. ⋃ m∈{n..}. ?P m) = 0 ∨ prob (⋂ n. ⋃ m∈{n..}.
?P m) = 1
    by (rule borel-0-1-law)
  also have prob (⋂ n. ⋃ m∈{n..}. ?P m) = 1 ⟷ (AE x in M. infinite {m. P
m x})
    using * by (simp add: prob-eq-1)
    (simp add: Bex-def infinite-nat-iff-unbounded-le)
  also have prob (⋂ n. ⋃ m∈{n..}. ?P m) = 0 ⟷ (AE x in M. finite {m. P m

```

```

x}))
  using * by (simp add: prob-eq-0)
  (auto simp add: Ball-def finite-nat-iff-bounded not-less [symmetric])
  finally show ?thesis
  by blast
qed

lemma (in prob-space) indep-sets-finite:
  assumes I:  $I \neq \{\}$  finite I
  and F:  $\bigwedge i. i \in I \implies F\ i \subseteq \text{events} \bigwedge i. i \in I \implies \text{space } M \in F\ i$ 
  shows indep-sets F I  $\longleftrightarrow (\forall A \in \text{Pi } I\ F. \text{prob } (\bigcap j \in I. A\ j) = (\prod j \in I. \text{prob } (A\ j)))$ 
proof
  assume *: indep-sets F I
  from I show  $\forall A \in \text{Pi } I\ F. \text{prob } (\bigcap j \in I. A\ j) = (\prod j \in I. \text{prob } (A\ j))$ 
  by (intro indep-setsD[OF *] ballI) auto
next
  assume indep:  $\forall A \in \text{Pi } I\ F. \text{prob } (\bigcap j \in I. A\ j) = (\prod j \in I. \text{prob } (A\ j))$ 
  show indep-sets F I
  proof (rule indep-setsI[OF F(1)])
    fix A J assume J:  $J \neq \{\}$   $J \subseteq I$  finite J
    assume A:  $\forall j \in J. A\ j \in F\ j$ 
    let ?A =  $\lambda j. \text{if } j \in J \text{ then } A\ j \text{ else space } M$ 
    have  $\text{prob } (\bigcap j \in I. ?A\ j) = \text{prob } (\bigcap j \in J. A\ j)$ 
    using subset-trans[OF F(1) sets.space-closed] J A
    by (auto intro!: arg-cong[where f=prob] split: if-split-asm) blast
    also
    from A F have  $(\lambda j. \text{if } j \in J \text{ then } A\ j \text{ else space } M) \in \text{Pi } I\ F$  (is ?A  $\in$  -)
    by (auto split: if-split-asm)
    with indep have  $\text{prob } (\bigcap j \in I. ?A\ j) = (\prod j \in I. \text{prob } (?A\ j))$ 
    by auto
    also have  $\dots = (\prod j \in J. \text{prob } (A\ j))$ 
    unfolding if-distrib prod.If-cases[OF <finite I>]
    using prob-space <J  $\subseteq$  I> by (simp add: Int-absorb1 prod.neutral-const)
    finally show  $\text{prob } (\bigcap j \in J. A\ j) = (\prod j \in J. \text{prob } (A\ j))$  ..
  qed
qed

```

```

lemma (in prob-space) indep-vars-finite:
  fixes I :: 'i set
  assumes I:  $I \neq \{\}$  finite I
  and M':  $\bigwedge i. i \in I \implies \text{sets } (M'\ i) = \text{sigma-sets } (\text{space } (M'\ i))\ (E\ i)$ 
  and rv:  $\bigwedge i. i \in I \implies \text{random-variable } (M'\ i)\ (X\ i)$ 
  and Int-stable:  $\bigwedge i. i \in I \implies \text{Int-stable } (E\ i)$ 
  and space:  $\bigwedge i. i \in I \implies \text{space } (M'\ i) \in E\ i$  and closed:  $\bigwedge i. i \in I \implies E\ i \subseteq$ 
  Pow (space (M' i))
  shows indep-vars M' X I  $\longleftrightarrow$ 
  ( $\forall A \in (\Pi\ i \in I. E\ i). \text{prob } (\bigcap j \in I. X\ j - 'A\ j \cap \text{space } M) = (\prod j \in I. \text{prob } (X\ j$ 
  - 'A j  $\cap \text{space } M))$ )

```


proof –

```

from rv have  $X: \bigwedge i. i \in I \implies X\ i \in \text{space } M \rightarrow \text{space } (M'\ i)$ 
  unfolding measurable-def by simp

{ fix i assume  $i \in I$ 
  from closed[OF  $\langle i \in I \rangle$ ]
  have  $\text{sigma-sets } (\text{space } M) \{X\ i - ' A \cap \text{space } M \mid A. A \in \text{sets } (M'\ i)\}$ 
     $= \text{sigma-sets } (\text{space } M) \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$ 
  unfolding sigma-sets-vimage-commute[OF X, OF  $\langle i \in I \rangle$ , symmetric]  $M'$ [OF
 $\langle i \in I \rangle$ ]
  by (subst sigma-sets-sigma-sets-eq) auto }
note  $\text{sigma-sets-}X = \text{this}$ 

{ fix i assume  $i \in I$ 
  have Int-stable  $\{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$ 
  proof (rule Int-stableI)
    fix a assume  $a \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$ 
    then obtain A where  $a = X\ i - ' A \cap \text{space } M \mid A \in E\ i$  by auto
    moreover
    fix b assume  $b \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$ 
    then obtain B where  $b = X\ i - ' B \cap \text{space } M \mid B \in E\ i$  by auto
    moreover
    have  $(X\ i - ' A \cap \text{space } M) \cap (X\ i - ' B \cap \text{space } M) = X\ i - ' (A \cap B) \cap$ 
space M by auto
    moreover note Int-stable[OF  $\langle i \in I \rangle$ ]
    ultimately
    show  $a \cap b \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$ 
    by (auto simp del: vimage-Int intro!: exI[of - A \cap B] dest: Int-stableD)
  qed }
note  $\text{indep-sets-}X = \text{indep-sets-sigma-sets-iff}$ [OF this]

{ fix i assume  $i \in I$ 
  { fix A assume  $A \in E\ i$ 
    with  $M'$ [OF  $\langle i \in I \rangle$ ] have  $A \in \text{sets } (M'\ i)$  by auto
    moreover
    from rv[OF  $\langle i \in I \rangle$ ] have  $X\ i \in \text{measurable } M\ (M'\ i)$  by auto
    ultimately
    have  $X\ i - ' A \cap \text{space } M \in \text{sets } M$  by (auto intro: measurable-sets) }
  with  $X$ [OF  $\langle i \in I \rangle$ ] space[OF  $\langle i \in I \rangle$ ]
  have  $\{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\} \subseteq \text{events}$ 
     $\text{space } M \in \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}$ 
    by (auto intro!: exI[of - space (M' i)]) }
  note  $\text{indep-sets-finite-}X = \text{indep-sets-finite}$ [OF I this]

  have  $(\forall A \in \Pi\ i \in I. \{X\ i - ' A \cap \text{space } M \mid A. A \in E\ i\}. \text{prob } (\bigcap (A - ' I)) = (\prod_{j \in I}.$ 
prob (A j))) =
   $(\forall A \in \Pi\ i \in I. E\ i. \text{prob } ((\bigcap_{j \in I} X\ j - ' A\ j) \cap \text{space } M) = (\prod_{x \in I} \text{prob } (X\ x$ 
 $- ' A\ x \cap \text{space } M)))$ 
  (is ?L = ?R)

```

```

proof safe
  fix  $A$  assume  $?L$  and  $A: A \in (\prod_{i \in I}. E\ i)$ 
  from  $\langle ?L \rangle [THEN\ bspec, of\ \lambda i. X\ i - 'A\ i \cap space\ M]\ A\ \langle I \neq \{\} \rangle$ 
  show  $prob\ ((\bigcap_{j \in I}. X\ j - 'A\ j) \cap space\ M) = (\prod_{x \in I}. prob\ (X\ x - 'A\ x \cap space\ M))$ 
  by  $(auto\ simp\ add: Pi\ iff)$ 
next
  fix  $A$  assume  $?R$  and  $A: A \in (\prod_{i \in I}. \{X\ i - 'A \cap space\ M \mid A. A \in E\ i\})$ 
  from  $A$  have  $\forall i \in I. \exists B. A\ i = X\ i - 'B \cap space\ M \wedge B \in E\ i$  by auto
  from  $bchoice[OF\ this]$  obtain  $B$  where  $B: \forall i \in I. A\ i = X\ i - 'B\ i \cap space\ M$ 
   $B \in (\prod_{i \in I}. E\ i)$  by auto
  from  $\langle ?R \rangle [THEN\ bspec, OF\ B(2)]\ B(1)\ \langle I \neq \{\} \rangle$ 
  show  $prob\ (\bigcap (A - 'I)) = (\prod_{j \in I}. prob\ (A\ j))$ 
  by simp
qed
then show  $?thesis$  using  $\langle I \neq \{\} \rangle$ 
  by  $(simp\ add: rv\ indep\ vars\ def\ indep\ sets\ X\ sigma\ sets\ X\ indep\ sets\ finite\ X$ 
cong: indep\ sets\ cong)
qed

```

lemma $(in\ prob\ space)\ indep\ vars\ compose:$

```

assumes  $indep\ vars\ M'\ X\ I$ 
assumes  $rv: \bigwedge i. i \in I \implies Y\ i \in measurable\ (M'\ i)\ (N\ i)$ 
shows  $indep\ vars\ N\ (\lambda i. Y\ i \circ X\ i)\ I$ 
unfolding  $indep\ vars\ def$ 

```

proof

```

from  $rv\ \langle indep\ vars\ M'\ X\ I \rangle$ 
show  $\forall i \in I. random\ variable\ (N\ i)\ (Y\ i \circ X\ i)$ 
by  $(auto\ simp: indep\ vars\ def)$ 

```

```

have  $indep\ sets\ (\lambda i. sigma\ sets\ (space\ M)\ \{X\ i - 'A \cap space\ M \mid A. A \in sets\ (M'\ i)\})\ I$ 

```

```

using  $\langle indep\ vars\ M'\ X\ I \rangle$  by  $(simp\ add: indep\ vars\ def)$ 
then show  $indep\ sets\ (\lambda i. sigma\ sets\ (space\ M)\ \{(Y\ i \circ X\ i) - 'A \cap space\ M \mid A. A \in sets\ (N\ i)\})\ I$ 

```

proof $(rule\ indep\ sets\ mono\ sets)$

```

fix  $i$  assume  $i \in I$ 
with  $\langle indep\ vars\ M'\ X\ I \rangle$  have  $X: X\ i \in space\ M \rightarrow space\ (M'\ i)$ 
unfolding  $indep\ vars\ def\ measurable\ def$  by auto
{ fix  $A$  assume  $A \in sets\ (N\ i)$ 
  then have  $\exists B. (Y\ i \circ X\ i) - 'A \cap space\ M = X\ i - 'B \cap space\ M \wedge B \in sets\ (M'\ i)$ 
  by  $(intro\ exI[of\ -\ Y\ i - 'A \cap space\ (M'\ i)])$ 
   $(auto\ simp: vimage\ comp\ intro!: measurable\ sets\ rv\ \langle i \in I \rangle\ funcset\ mem[OF\ X])\ }$ 

```

```

then show  $sigma\ sets\ (space\ M)\ \{(Y\ i \circ X\ i) - 'A \cap space\ M \mid A. A \in sets\ (N\ i)\} \subseteq$ 

```

```

 $sigma\ sets\ (space\ M)\ \{X\ i - 'A \cap space\ M \mid A. A \in sets\ (M'\ i)\}$ 
by  $(intro\ sigma\ sets\ subseteq)\ (auto\ simp: vimage\ comp)$ 

```

qed
qed

lemma (in *prob-space*) *indep-vars-compose2*:
assumes *indep-vars* $M' X I$
assumes *rv*: $\bigwedge i. i \in I \implies Y i \in \text{measurable } (M' i) (N i)$
shows *indep-vars* $N (\lambda i x. Y i (X i x)) I$
using *indep-vars-compose* [*OF assms*] **by** (*simp add: comp-def*)

lemma (in *prob-space*) *indep-var-compose*:
assumes *indep-var* $M1 X1 M2 X2 Y1 \in \text{measurable } M1 N1 Y2 \in \text{measurable } M2 N2$
shows *indep-var* $N1 (Y1 \circ X1) N2 (Y2 \circ X2)$
proof –
have *indep-vars* (*case-bool* $N1 N2$) ($\lambda b. \text{case-bool } Y1 Y2 b \circ \text{case-bool } X1 X2 b$)
UNIV
using *assms*
by (*intro indep-vars-compose* [**where** $M' = \text{case-bool } M1 M2$])
(*auto simp: indep-var-def split: bool.split*)
also have ($\lambda b. \text{case-bool } Y1 Y2 b \circ \text{case-bool } X1 X2 b$) = *case-bool* ($Y1 \circ X1$)
($Y2 \circ X2$)
by (*simp add: fun-eq-iff split: bool.split*)
finally show ?thesis
unfolding *indep-var-def* .
qed

lemma (in *prob-space*) *indep-vars-Min*:
fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$
assumes *I*: *finite* $I i \notin I$ **and** *indep*: *indep-vars* ($\lambda -. \text{borel}$) $X (\text{insert } i I)$
shows *indep-var* *borel* ($X i$) *borel* ($\lambda \omega. \text{Min } ((\lambda i. X i \omega) 'I)$)
proof –
have *indep-var*
borel ($(\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\})$)
borel ($(\lambda f. \text{Min } (f 'I)) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I)$)
using *I* **by** (*intro indep-var-compose* [*OF indep-var-restrict* [*OF indep*]] *borel-measurable-Min*)
auto
also have $((\lambda f. f i) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) \{i\})) = X i$
by *auto*
also have $((\lambda f. \text{Min } (f 'I)) \circ (\lambda \omega. \text{restrict } (\lambda i. X i \omega) I)) = (\lambda \omega. \text{Min } ((\lambda i. X i \omega) 'I))$
ω)
by (*auto cong: rev-conj-cong*)
finally show ?thesis
unfolding *indep-var-def* .
qed

lemma (in *prob-space*) *indep-vars-sum*:
fixes $X :: 'i \Rightarrow 'a \Rightarrow \text{real}$
assumes *I*: *finite* $I i \notin I$ **and** *indep*: *indep-vars* ($\lambda -. \text{borel}$) $X (\text{insert } i I)$
shows *indep-var* *borel* ($X i$) *borel* ($\lambda \omega. \sum_{i \in I. X i \omega}$)

proof –

have *indep-var*
 borel $((\lambda f. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ \{i\}))$
 borel $((\lambda f. \sum_{i \in I}. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ I))$
 using *I* **by** $(\text{intro indep-var-compose}[OF\ indep-var-restrict[OF\ indep]])$ *auto*
also have $((\lambda f. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ \{i\})) = X\ i$
 by *auto*
also have $((\lambda f. \sum_{i \in I}. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ I)) = (\lambda \omega. \sum_{i \in I}. X\ i\ \omega)$
 by $(\text{auto cong; rev-conj-cong})$
finally show *?thesis* .
qed

lemma (*in prob-space*) *indep-vars-prod*:

fixes *X* :: '*i* \Rightarrow 'a \Rightarrow real
assumes *I*: *finite I* *i* $\notin I$ **and** *indep*: *indep-vars* $(\lambda \cdot. \text{borel})\ X\ (\text{insert } i\ I)$
shows *indep-var borel* $(X\ i)\ \text{borel } (\lambda \omega. \prod_{i \in I}. X\ i\ \omega)$

proof –

have *indep-var*
 borel $((\lambda f. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ \{i\}))$
 borel $((\lambda f. \prod_{i \in I}. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ I))$
 using *I* **by** $(\text{intro indep-var-compose}[OF\ indep-var-restrict[OF\ indep]])$ *auto*
also have $((\lambda f. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ \{i\})) = X\ i$
 by *auto*
also have $((\lambda f. \prod_{i \in I}. f\ i) \circ (\lambda \omega. \text{restrict } (\lambda i. X\ i\ \omega) \ I)) = (\lambda \omega. \prod_{i \in I}. X\ i\ \omega)$
 by $(\text{auto cong; rev-conj-cong})$
finally show *?thesis* .
qed

lemma (*in prob-space*) *indep-varsD-finite*:

assumes *X*: *indep-vars* *M'* *X* *I*
assumes *I*: *I* $\neq \{\}$ *finite I* $\bigwedge i. i \in I \implies A\ i \in \text{sets } (M'\ i)$
shows *prob* $(\bigcap_{i \in I}. X\ i - 'A\ i \cap \text{space } M) = (\prod_{i \in I}. \text{prob } (X\ i - 'A\ i \cap \text{space } M))$

proof (*rule indep-setsD*)

show *indep-sets* $(\lambda i. \text{sigma-sets } (\text{space } M) \ \{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (M'\ i)\})\ I$
 using *X* **by** $(\text{auto simp: indep-vars-def})$
show $I \subseteq I\ I \neq \{\}$ *finite I* **using** *I* **by** *auto*
show $\forall i \in I. X\ i - 'A\ i \cap \text{space } M \in \text{sigma-sets } (\text{space } M) \ \{X\ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (M'\ i)\}$
 using *I* **by** *auto*
qed

lemma (*in prob-space*) *indep-varsD*:

assumes *X*: *indep-vars* *M'* *X* *I*
assumes *I*: *J* $\neq \{\}$ *finite J* $J \subseteq I \bigwedge i. i \in J \implies A\ i \in \text{sets } (M'\ i)$
shows *prob* $(\bigcap_{i \in J}. X\ i - 'A\ i \cap \text{space } M) = (\prod_{i \in J}. \text{prob } (X\ i - 'A\ i \cap \text{space } M))$
proof (*rule indep-setsD*)

```

show indep-sets ( $\lambda i. \text{sigma-sets } (\text{space } M) \{X \ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (M' \ i)\}$ )  $I$ 
using  $X$  by (auto simp: indep-vars-def)
show  $\forall i \in J. X \ i - 'A \ i \cap \text{space } M \in \text{sigma-sets } (\text{space } M) \{X \ i - 'A \cap \text{space } M \mid A. A \in \text{sets } (M' \ i)\}$ 
using  $I$  by auto
qed fact+

```

lemma (in prob-space) indep-vars-iff-distr-eq-PiM:

```

fixes  $I :: 'i \text{ set}$  and  $X :: 'i \Rightarrow 'a \Rightarrow 'b$ 
assumes  $I \neq \{\}$ 
assumes  $rv: \bigwedge i. \text{random-variable } (M' \ i) (X \ i)$ 
shows  $\text{indep-vars } M' \ X \ I \longleftrightarrow$ 
 $\text{distr } M \ (\Pi_M \ i \in I. M' \ i) (\lambda x. \lambda i \in I. X \ i \ x) = (\Pi_M \ i \in I. \text{distr } M \ (M' \ i) (X \ i))$ 

```

proof –

```

let  $?P = \Pi_M \ i \in I. M' \ i$ 
let  $?X = \lambda x. \lambda i \in I. X \ i \ x$ 
let  $?D = \text{distr } M \ ?P \ ?X$ 
have  $X: \text{random-variable } ?P \ ?X$  by (intro measurable-restrict rv)
interpret  $D: \text{prob-space } ?D$  by (intro prob-space-distr  $X$ )

```

```

let  $?D' = \lambda i. \text{distr } M \ (M' \ i) (X \ i)$ 
let  $?P' = \Pi_M \ i \in I. \text{distr } M \ (M' \ i) (X \ i)$ 
interpret  $D': \text{prob-space } ?D' \ i$  for  $i$  by (intro prob-space-distr rv)
interpret  $P: \text{product-prob-space } ?D' \ I \ ..$ 

```

show ?thesis

proof

```

assume  $\text{indep-vars } M' \ X \ I$ 
show  $?D = ?P'$ 
proof (rule measure-eqI-generator-eq)
show  $\text{Int-stable } (\text{prod-algebra } I \ M')$ 
by (rule Int-stable-prod-algebra)
show  $\text{prod-algebra } I \ M' \subseteq \text{Pow } (\text{space } ?P)$ 
using  $\text{prod-algebra-sets-into-space}$  by (simp add: space-PiM)
show  $\text{sets } ?D = \text{sigma-sets } (\text{space } ?P) (\text{prod-algebra } I \ M')$ 
by (simp add: sets-PiM space-PiM)
show  $\text{sets } ?P' = \text{sigma-sets } (\text{space } ?P) (\text{prod-algebra } I \ M')$ 
by (simp add: sets-PiM space-PiM cong: prod-algebra-cong)
let  $?A = \lambda i. \Pi_E \ i \in I. \text{space } (M' \ i)$ 
show  $\text{range } ?A \subseteq \text{prod-algebra } I \ M' (\bigcup i. ?A \ i) = \text{space } (\Pi_M \ i \in I. M' \ i)$ 
by (auto simp: space-PiM intro!: space-in-prod-algebra cong: prod-algebra-cong)
{ fix } i show  $\text{emeasure } ?D (\Pi_E \ i \in I. \text{space } (M' \ i)) \neq \infty$  by auto }

```

next

```

fix  $E$  assume  $E: E \in \text{prod-algebra } I \ M'$ 
from  $\text{prod-algebra } E[\text{OF } E]$  obtain  $J \ Y$ 
where  $J$ :
 $E = \text{prod-emb } I \ M' \ J \ (\Pi_E \ J \ Y)$ 
 $\text{finite } J$ 

```

```

     $J \neq \{\}$   $\vee I = \{\}$ 
     $J \subseteq I$ 
     $\bigwedge i. i \in J \implies Y i \in \text{sets } (M' i)$ 
  by auto
  from  $E$  have  $E \in \text{sets } ?P$  by (auto simp: sets-PiM)
  then have  $\text{emeasure } ?D E = \text{emeasure } M (?X - ' E \cap \text{space } M)$ 
    by (simp add: emeasure-distr  $X$ )
  also have  $?X - ' E \cap \text{space } M = (\bigcap i \in J. X i - ' Y i \cap \text{space } M)$ 
    using  $J \langle I \neq \{\} \rangle$  measurable-space[OF  $rv$ ] by (auto simp: prod-emb-def
  PiE-iff split: if-split-asm)
  also have  $\text{emeasure } M (\bigcap i \in J. X i - ' Y i \cap \text{space } M) = (\prod i \in J. \text{emeasure } M (X i - ' Y i \cap \text{space } M))$ 
    using  $\langle \text{indep-vars } M' X I \rangle J \langle I \neq \{\} \rangle$  using indep-varsD[of  $M' X I J$ ]
  by (auto simp: emeasure-eq-measure prod-ennreal measure-nonneg prod-nonneg)
  also have  $\dots = (\prod i \in J. \text{emeasure } (?D' i) (Y i))$ 
    using  $rv J$  by (simp add: emeasure-distr)
  also have  $\dots = \text{emeasure } ?P' E$ 
    using  $P.\text{emeasure-PiM-emb}$ [of  $J Y$ ]  $J$  by (simp add: prod-emb-def)
  finally show  $\text{emeasure } ?D E = \text{emeasure } ?P' E$ .
qed
next
  assume  $?D = ?P'$ 
  show indep-vars  $M' X I$  unfolding indep-vars-def
  proof (intro conjI indep-setsI ballI  $rv$ )
    fix  $i$  show sigma-sets (space  $M$ )  $\{X i - ' A \cap \text{space } M \mid A. A \in \text{sets } (M' i)\}$ 
 $\subseteq \text{events}$ 
    by (auto intro!: sets.sigma-sets-subset measurable-sets  $rv$ )
  next
    fix  $J Y'$  assume  $J: J \neq \{\} J \subseteq I$  finite  $J$ 
    assume  $Y': \forall j \in J. Y' j \in \text{sigma-sets } (\text{space } M) \{X j - ' A \cap \text{space } M \mid A. A \in \text{sets } (M' j)\}$ 
    have  $\forall j \in J. \exists Y. Y' j = X j - ' Y \cap \text{space } M \wedge Y \in \text{sets } (M' j)$ 
    proof
      fix  $j$  assume  $j \in J$ 
      from  $Y'$ [rule-format, OF this]  $rv$ [of  $j$ ]
      show  $\exists Y. Y' j = X j - ' Y \cap \text{space } M \wedge Y \in \text{sets } (M' j)$ 
      by (subst (asm) sigma-sets-vimage-commute[symmetric, of - - space (M' j)])
      (auto dest: measurable-space simp: sets.sigma-sets-eq)
    qed
  from bchoice[OF this] obtain  $Y$  where
     $Y: \bigwedge j. j \in J \implies Y' j = X j - ' Y j \cap \text{space } M \wedge j. j \in J \implies Y j \in \text{sets } (M' j)$ 
  by auto
  let  $?E = \text{prod-emb } I M' J (Pi_E J Y)$ 
  from  $Y$  have  $(\bigcap j \in J. Y' j) = ?X - ' ?E \cap \text{space } M$ 
    using  $J \langle I \neq \{\} \rangle$  measurable-space[OF  $rv$ ] by (auto simp: prod-emb-def
  PiE-iff split: if-split-asm)
  then have  $\text{emeasure } M (\bigcap j \in J. Y' j) = \text{emeasure } M (?X - ' ?E \cap \text{space } M)$ 
    by simp

```

```

also have ... = emeasure ?D ?E
  using Y J by (intro emeasure-distr[symmetric] X sets-PiM-I) auto
also have ... = emeasure ?P' ?E
  using ⟨?D = ?P'⟩ by simp
also have ... = (∏ i∈J. emeasure (?D' i) (Y i))
  using P.emeasure-PiM-emb[of J Y] J Y by (simp add: prod-emb-def)
also have ... = (∏ i∈J. emeasure M (Y' i))
  using rv J Y by (simp add: emeasure-distr)
finally have emeasure M (∏ j∈J. Y' j) = (∏ i∈J. emeasure M (Y' i)) .
then show prob (∏ j∈J. Y' j) = (∏ i∈J. prob (Y' i))
  by (auto simp: emeasure-eq-measure prod-ennreal measure-nonneg prod-nonneg)
qed
qed
qed

```

lemma (in prob-space) indep-vars-iff-distr-eq-PiM':

```

fixes I :: 'i set and X :: 'i ⇒ 'a ⇒ 'b
assumes I ≠ {}
assumes rv: ⋀i. i ∈ I ⇒ random-variable (M' i) (X i)
shows indep-vars M' X I ⟷
  distr M (ΠM i∈I. M' i) (λx. λi∈I. X i x) = (ΠM i∈I. distr M (M' i)
(X i))

```

proof –

```

from assms obtain j where j: j ∈ I
  by auto
define N' where N' = (λi. if i ∈ I then M' i else M' j)
define Y where Y = (λi. if i ∈ I then X i else X j)
have rv: random-variable (N' i) (Y i) for i
  using j by (auto simp: N'-def Y-def intro: assms)

have indep-vars M' X I = indep-vars N' Y I
  by (intro indep-vars-cong) (auto simp: N'-def Y-def)
also have ... ⟷ distr M (ΠM i∈I. N' i) (λx. λi∈I. Y i x) = (ΠM i∈I. distr
M (N' i) (Y i))
  by (intro indep-vars-iff-distr-eq-PiM rv assms)
also have (ΠM i∈I. N' i) = (ΠM i∈I. M' i)
  by (intro PiM-cong) (simp-all add: N'-def)
also have (λx. λi∈I. Y i x) = (λx. λi∈I. X i x)
  by (simp-all add: Y-def fun-eq-iff)
also have (ΠM i∈I. distr M (N' i) (Y i)) = (ΠM i∈I. distr M (M' i) (X i))
  by (intro PiM-cong distr-cong) (simp-all add: N'-def Y-def)
finally show ?thesis .
qed

```

lemma (in prob-space) indep-varD:

```

assumes indep: indep-var Ma A Mb B
assumes sets: Xa ∈ sets Ma Xb ∈ sets Mb
shows prob ((λx. (A x, B x)) - ' (Xa × Xb) ∩ space M) =
  prob (A - ' Xa ∩ space M) * prob (B - ' Xb ∩ space M)

```

proof –

have $\text{prob } ((\lambda x. (A \ x, B \ x)) - ' (Xa \times Xb) \cap \text{space } M) =$
 $\text{prob } (\bigcap i \in \text{UNIV}. (\text{case-bool } A \ B \ i - ' \text{case-bool } Xa \ Xb \ i \cap \text{space } M))$
 by (*auto intro!*: *arg-cong*[**where** $f = \text{prob}$] *simp*: *UNIV-bool*)
 also have $\dots = (\prod i \in \text{UNIV}. \text{prob } (\text{case-bool } A \ B \ i - ' \text{case-bool } Xa \ Xb \ i \cap \text{space } M))$
 using *indep unfolding indep-var-def*
 by (*rule indep-varsD*) (*auto split*: *bool.split intro*: *sets*)
 also have $\dots = \text{prob } (A - ' Xa \cap \text{space } M) * \text{prob } (B - ' Xb \cap \text{space } M)$
 unfolding *UNIV-bool* **by** *simp*
 finally show *?thesis* .
qed

lemma (*in prob-space*) *prob-indep-random-variable*:

assumes *ind*[*simp*]: *indep-var* $N \ X \ N \ Y$
 assumes [*simp*]: $A \in \text{sets } N \ B \in \text{sets } N$
 shows $\mathcal{P}(x \text{ in } M. X \ x \in A \wedge Y \ x \in B) = \mathcal{P}(x \text{ in } M. X \ x \in A) * \mathcal{P}(x \text{ in } M. Y \ x \in B)$
proof –
 have $\mathcal{P}(x \text{ in } M. (X \ x) \in A \wedge (Y \ x) \in B) = \text{prob } ((\lambda x. (X \ x, Y \ x)) - ' (A \times B) \cap \text{space } M)$
 by (*auto intro!*: *arg-cong*[**where** $f = \text{prob}$])
 also have $\dots = \text{prob } (X - ' A \cap \text{space } M) * \text{prob } (Y - ' B \cap \text{space } M)$
 by (*auto intro!*: *indep-varD*[**where** $Ma = N$ **and** $Mb = N$])
 also have $\dots = \mathcal{P}(x \text{ in } M. X \ x \in A) * \mathcal{P}(x \text{ in } M. Y \ x \in B)$
 by (*auto intro!*: *arg-cong2*[**where** $f = (*)$] *arg-cong*[**where** $f = \text{prob}$])
 finally show *?thesis* .
qed

lemma (*in prob-space*)

assumes *indep-var* $S \ X \ T \ Y$
 shows *indep-var-rv1*: *random-variable* $S \ X$
 and *indep-var-rv2*: *random-variable* $T \ Y$

proof –

have $\forall i \in \text{UNIV}. \text{random-variable } (\text{case-bool } S \ T \ i) (\text{case-bool } X \ Y \ i)$
 using *assms unfolding indep-var-def indep-vars-def* **by** *auto*
 then show *random-variable* $S \ X$ *random-variable* $T \ Y$
 unfolding *UNIV-bool* **by** *auto*
qed

lemma (*in prob-space*) *indep-var-distribution-eq*:

$\text{indep-var } S \ X \ T \ Y \longleftrightarrow \text{random-variable } S \ X \wedge \text{random-variable } T \ Y \wedge$
 $\text{distr } M \ S \ X \otimes_M \text{distr } M \ T \ Y = \text{distr } M \ (S \otimes_M T) (\lambda x. (X \ x, Y \ x))$ (**is** -
 $\longleftrightarrow - \wedge - \wedge ?S \otimes_M ?T = ?J$)

proof *safe*

assume *indep-var* $S \ X \ T \ Y$
 then show *rvs*: *random-variable* $S \ X$ *random-variable* $T \ Y$
 by (*blast dest*: *indep-var-rv1 indep-var-rv2*) +
 then have XY : *random-variable* $(S \otimes_M T) (\lambda x. (X \ x, Y \ x))$


```

    by (rule measurable-Pair)

interpret X: prob-space ?S by (rule prob-space-distr) fact
interpret Y: prob-space ?T by (rule prob-space-distr) fact
interpret XY: pair-prob-space ?S ?T ..
show ?S  $\otimes_M$  ?T = ?J
proof (rule pair-measure-eqI)
  show sigma-finite-measure ?S ..
  show sigma-finite-measure ?T ..

fix A B assume A: A  $\in$  sets ?S and B: B  $\in$  sets ?T
have emeasure ?J (A  $\times$  B) = emeasure M (( $\lambda x.$  (X x, Y x)) -‘ (A  $\times$  B)  $\cap$ 
space M)
  using A B by (intro emeasure-distr[OF XY]) auto
also have ... = emeasure M (X -‘ A  $\cap$  space M) * emeasure M (Y -‘ B  $\cap$ 
space M)
  using indep-varD[OF <indep-var S X T Y>, of A B] A B
  by (simp add: emeasure-eq-measure measure-nonneg ennreal-mult)
also have ... = emeasure ?S A * emeasure ?T B
  using rvs A B by (simp add: emeasure-distr)
finally show emeasure ?S A * emeasure ?T B = emeasure ?J (A  $\times$  B) by
simp
qed simp
next
assume rvs: random-variable S X random-variable T Y
then have XY: random-variable (S  $\otimes_M$  T) ( $\lambda x.$  (X x, Y x))
  by (rule measurable-Pair)

let ?S = distr M S X and ?T = distr M T Y
interpret X: prob-space ?S by (rule prob-space-distr) fact
interpret Y: prob-space ?T by (rule prob-space-distr) fact
interpret XY: pair-prob-space ?S ?T ..

assume ?S  $\otimes_M$  ?T = ?J

{ fix S and X
  have Int-stable {X -‘ A  $\cap$  space M | A. A  $\in$  sets S}
  proof (safe intro!: Int-stableI)
    fix A B assume A  $\in$  sets S B  $\in$  sets S
    then show  $\exists C.$  (X -‘ A  $\cap$  space M)  $\cap$  (X -‘ B  $\cap$  space M) = (X -‘ C  $\cap$ 
space M)  $\wedge$  C  $\in$  sets S
      by (intro exI[of - A  $\cap$  B]) auto
    qed }
note Int-stable = this

show indep-var S X T Y unfolding indep-var-eq
proof (intro conjI indep-set-sigma-sets Int-stable rvs)
  show indep-set {X -‘ A  $\cap$  space M | A. A  $\in$  sets S} {Y -‘ A  $\cap$  space M | A.
A  $\in$  sets T}

```

```

proof (safe intro!: indep-setI)
  { fix  $A$  assume  $A \in \text{sets } S$  then show  $X - 'A \cap \text{space } M \in \text{sets } M$ 
    using  $\langle X \in \text{measurable } M \ S \rangle$  by (auto intro: measurable-sets) }
  { fix  $A$  assume  $A \in \text{sets } T$  then show  $Y - 'A \cap \text{space } M \in \text{sets } M$ 
    using  $\langle Y \in \text{measurable } M \ T \rangle$  by (auto intro: measurable-sets) }
next
  fix  $A \ B$  assume  $ab: A \in \text{sets } S \ B \in \text{sets } T$ 
  then have  $\text{prob } ((X - 'A \cap \text{space } M) \cap (Y - 'B \cap \text{space } M)) = \text{emeasure}$ 
 $?J \ (A \times B)$ 
    using  $XY$  by (auto simp add: emeasure-distr emeasure-eq-measure mea-
sure-nonneg intro!: arg-cong[where  $f=\text{prob}$ ])
  also have  $\dots = \text{emeasure } (?S \otimes_M ?T) \ (A \times B)$ 
    unfolding  $\langle ?S \otimes_M ?T = ?J \rangle$  ..
  also have  $\dots = \text{emeasure } ?S \ A * \text{emeasure } ?T \ B$ 
    using  $ab$  by (simp add:  $Y.\text{emeasure-pair-measure-Times}$ )
  finally show  $\text{prob } ((X - 'A \cap \text{space } M) \cap (Y - 'B \cap \text{space } M)) =$ 
 $\text{prob } (X - 'A \cap \text{space } M) * \text{prob } (Y - 'B \cap \text{space } M)$ 
    using  $\text{rvs } ab$  by (simp add: emeasure-eq-measure emeasure-distr mea-
sure-nonneg ennreal-mult[symmetric])
  qed
qed
qed

```

```

lemma (in prob-space) distributed-joint-indep:
  assumes  $S$ : sigma-finite-measure  $S$  and  $T$ : sigma-finite-measure  $T$ 
  assumes  $X$ : distributed  $M \ S \ X \ P_x$  and  $Y$ : distributed  $M \ T \ Y \ P_y$ 
  assumes indep: indep-var  $S \ X \ T \ Y$ 
  shows distributed  $M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x)) \ (\lambda(x, y). P_x \ x * P_y \ y)$ 
  using indep-var-distribution-eq[of  $S \ X \ T \ Y$ ] indep
  by (intro distributed-joint-indep[OF  $S \ T \ X \ Y$ ]) auto

```

```

lemma (in prob-space) indep-vars-nn-integral:
  assumes  $I$ : finite  $I$  indep-vars  $(\lambda-. \text{borel}) \ X \ I \ \wedge i \ \omega. i \in I \implies 0 \leq X \ i \ \omega$ 
  shows  $(\int^+ \omega. (\prod_{i \in I}. X \ i \ \omega) \ \partial M) = (\prod_{i \in I}. \int^+ \omega. X \ i \ \omega \ \partial M)$ 

```

```

proof cases
  assume  $I \neq \{\}$ 
  define  $Y$  where [abs-def]:  $Y \ i \ \omega = (\text{if } i \in I \text{ then } X \ i \ \omega \text{ else } 0)$  for  $i \ \omega$ 
  { fix  $i$  have  $i \in I \implies \text{random-variable borel } (X \ i)$ 
    using  $I(2)$  by (cases  $i \in I$ ) (auto simp: indep-vars-def) }
  note  $\text{rv-}X = \text{this}$ 

```

```

  { fix  $i$  have random-variable borel  $(Y \ i)$ 
    using  $I(2)$  by (cases  $i \in I$ ) (auto simp:  $Y\text{-def } \text{rv-}X$ ) }
  note  $\text{rv-}Y = \text{this}[\text{measurable}]$ 

```

```

interpret  $Y$ : prob-space distr  $M$  borel  $(Y \ i)$  for  $i$ 
  using  $I(2)$  by (cases  $i \in I$ ) (auto intro!: prob-space-distr simp: indep-vars-def)
prob-space-return
  interpret product-sigma-finite  $\lambda i. \text{distr } M \text{ borel } (Y \ i)$ 

```

```

..

have indep-Y: indep-vars ( $\lambda i. \text{borel}$ ) Y I
  by (rule indep-vars-cong[THEN iffD1, OF - - - I(2)]) (auto simp: Y-def)

have ( $\int^{+\omega}. (\prod_{i \in I}. X \ i \ \omega) \ \partial M$ ) = ( $\int^{+\omega}. (\prod_{i \in I}. Y \ i \ \omega) \ \partial M$ )
  using I(3) by (auto intro!: nn-integral-cong prod.cong simp add: Y-def max-def)
also have ... = ( $\int^{+\omega}. (\prod_{i \in I}. \omega \ i) \ \partial \text{distr } M \ (Pi_M \ I \ (\lambda i. \text{borel})) \ (\lambda x. \lambda i \in I. Y \ i \ x)$ )
  by (subst nn-integral-distr) auto
also have ... = ( $\int^{+\omega}. (\prod_{i \in I}. \omega \ i) \ \partial Pi_M \ I \ (\lambda i. \text{distr } M \ \text{borel} \ (Y \ i))$ )
  unfolding indep-vars-iff-distr-eq-PiM[THEN iffD1, OF  $\langle I \neq \{\} \rangle$  rv-Y indep-Y]
..
also have ... = ( $\prod_{i \in I}. (\int^{+\omega}. \omega \ \partial \text{distr } M \ \text{borel} \ (Y \ i))$ )
  by (rule product-nn-integral-prod) (auto intro:  $\langle \text{finite } I \rangle$ )
also have ... = ( $\prod_{i \in I}. \int^{+\omega}. X \ i \ \omega \ \partial M$ )
  by (intro prod.cong nn-integral-cong) (auto simp: nn-integral-distr Y-def rv-X)
finally show ?thesis .
qed (simp add: emeasure-space-1)

lemma (in prob-space)
  fixes X :: 'i  $\Rightarrow$  'a  $\Rightarrow$  'b::{real-normed-field, banach, second-countable-topology}
  assumes I: finite I indep-vars ( $\lambda \cdot. \text{borel}$ ) X I  $\wedge i. i \in I \implies \text{integrable } M \ (X \ i)$ 
  shows indep-vars-lebesgue-integral: ( $\int \omega. (\prod_{i \in I}. X \ i \ \omega) \ \partial M$ ) = ( $\prod_{i \in I}. \int \omega. X \ i \ \omega \ \partial M$ ) (is ?eq)
  and indep-vars-integrable: integrable M ( $\lambda \omega. (\prod_{i \in I}. X \ i \ \omega)$ ) (is ?int)
proof (induct rule: case-split)
  assume I  $\neq \{\}$ 
  define Y where [abs-def]:  $Y \ i \ \omega = (\text{if } i \in I \text{ then } X \ i \ \omega \text{ else } 0)$  for i  $\omega$ 
  { fix i have  $i \in I \implies \text{random-variable borel } (X \ i)$ 
    using I(2) by (cases  $i \in I$ ) (auto simp: indep-vars-def) }
  note rv-X = this[measurable]

  { fix i have random-variable borel (Y i)
    using I(2) by (cases  $i \in I$ ) (auto simp: Y-def rv-X) }
  note rv-Y = this[measurable]

  { fix i have integrable M (Y i)
    using I(3) by (cases  $i \in I$ ) (auto simp: Y-def) }
  note int-Y = this

  interpret Y: prob-space distr M borel (Y i) for i
    using I(2) by (cases  $i \in I$ ) (auto intro!: prob-space-distr simp: indep-vars-def
    prob-space-return)
  interpret product-sigma-finite  $\lambda i. \text{distr } M \ \text{borel} \ (Y \ i)$ 
  ..

  have indep-Y: indep-vars ( $\lambda i. \text{borel}$ ) Y I
    by (rule indep-vars-cong[THEN iffD1, OF - - - I(2)]) (auto simp: Y-def)

```

```

have (∫ ω. (∏ i∈I. X i ω) ∂M) = (∫ ω. (∏ i∈I. Y i ω) ∂M)
  using I(β) by (simp add: Y-def)
also have ... = (∫ ω. (∏ i∈I. ω i) ∂distr M (Pi_M I (λi. borel)) (λx. λi∈I. Y i
x))
  by (subst integral-distr) auto
also have ... = (∫ ω. (∏ i∈I. ω i) ∂Pi_M I (λi. distr M borel (Y i)))
  unfolding indep-vars-iff-distr-eq-PiM[THEN iffD1, OF ⟨I ≠ {}⟩ rv-Y indep-Y]
..
also have ... = (∏ i∈I. (∫ ω. ω ∂distr M borel (Y i)))
  by (rule product-integral-prod) (auto intro: ⟨finite I⟩ simp: integrable-distr-eq
int-Y)
also have ... = (∏ i∈I. ∫ ω. X i ω ∂M)
  by (intro prod.cong integral-cong)
  (auto simp: integral-distr Y-def rv-X)
finally show ?eq .

have integrable (distr M (Pi_M I (λi. borel)) (λx. λi∈I. Y i x)) (λω. (∏ i∈I. ω
i))
  unfolding indep-vars-iff-distr-eq-PiM[THEN iffD1, OF ⟨I ≠ {}⟩ rv-Y indep-Y]
  by (intro product-integrable-prod[OF ⟨finite I⟩])
  (simp add: integrable-distr-eq int-Y)
then show ?int
  by (simp add: integrable-distr-eq Y-def)
qed (simp-all add: prob-space)

lemma (in prob-space)
  fixes X1 X2 :: 'a ⇒ 'b::{real-normed-field, banach, second-countable-topology}
  assumes indep-var borel X1 borel X2 integrable M X1 integrable M X2
  shows indep-var-lebesgue-integral: (∫ ω. X1 ω * X2 ω ∂M) = (∫ ω. X1 ω ∂M)
* (∫ ω. X2 ω ∂M) (is ?eq)
  and indep-var-integrable: integrable M (λω. X1 ω * X2 ω) (is ?int)
unfolding indep-var-def
proof -
  have *: (λω. X1 ω * X2 ω) = (λω. ∏ i∈UNIV. (case-bool X1 X2 i ω))
    by (simp add: UNIV-bool mult.commute)
  have **: (λ -. borel) = case-bool borel borel
    by (rule ext, metis (full-types) bool.simps(3) bool.simps(4))
  show ?eq
    apply (subst *)
    apply (subst indep-vars-lebesgue-integral)
    apply (auto)
    apply (subst **, subst indep-var-def [symmetric], rule assms)
    apply (simp split: bool.split add: assms)
    by (simp add: UNIV-bool mult.commute)
  show ?int
    apply (subst *)
    apply (rule indep-vars-integrable)
    apply auto

```

```

    apply (subst **, subst indep-var-def [symmetric], rule assms)
    by (simp split: bool.split add: assms)
qed

end

```

11 Convolution Measure

theory *Convolution*

imports *Independent-Family*

begin

lemma (in *finite-measure*) *sigma-finite-measure*: *sigma-finite-measure* M
 $..$

definition *convolution* :: ($'a :: \text{ordered-euclidean-space}$) *measure* \Rightarrow $'a$ *measure* \Rightarrow
 $'a$ *measure* (**infix** \star 50) **where**
 $\text{convolution } M \ N = \text{distr } (M \otimes_M N) \ \text{borel } (\lambda(x, y). x + y)$

lemma

shows *space-convolution*[*simp*]: *space* (*convolution* $M \ N$) = *space borel*
and *sets-convolution*[*simp*]: *sets* (*convolution* $M \ N$) = *sets borel*
and *measurable-convolution1*[*simp*]: *measurable* A (*convolution* $M \ N$) = *measurable* A *borel*
and *measurable-convolution2*[*simp*]: *measurable* (*convolution* $M \ N$) B = *measurable* *borel* B
by (*simp-all add: convolution-def*)

lemma *nn-integral-convolution*:

assumes *finite-measure* M *finite-measure* N
assumes [*measurable-cong*]: *sets* N = *sets borel* *sets* M = *sets borel*
assumes [*measurable*]: $f \in \text{borel-measurable borel}$
shows $(\int^{+x}. f \ x \ \partial \text{convolution } M \ N) = (\int^{+x}. \int^{+y}. f \ (x + y) \ \partial N \ \partial M)$

proof –

interpret M : *finite-measure* M **by** *fact*
interpret N : *finite-measure* N **by** *fact*
interpret *pair-sigma-finite* $M \ N$ $..$
show *?thesis*
unfolding *convolution-def*
by (*simp add: nn-integral-distr N.nn-integral-fst[symmetric]*)
qed

lemma *convolution-emeasure*:

assumes $A \in \text{sets borel}$ *finite-measure* M *finite-measure* N
assumes [*simp*]: *sets* N = *sets borel* *sets* M = *sets borel*
assumes [*simp*]: *space* M = *space* N *space* N = *space borel*
shows *emeasure* ($M \star N$) A = $\int^{+x}. (\text{emeasure } N \ \{a. a + x \in A\}) \ \partial M$
using *assms* **by** (*auto intro!: nn-integral-cong simp del: nn-integral-indicator simp: nn-integral-convolution*)

nn-integral-indicator [symmetric] *ac-simps* *split:split-indicator*)

lemma *convolution-emeasure'*:

assumes [simp]: $A \in \text{sets borel}$
assumes [simp]: *finite-measure* M *finite-measure* N
assumes [simp]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$
shows *emeasure* $(M \star N) A = \int^+ x. \int^+ y. (\text{indicator } A (x + y)) \partial N \partial M$
by (*auto simp del: nn-integral-indicator simp: nn-integral-convolution*
nn-integral-indicator[symmetric] *borel-measurable-indicator*)

lemma *convolution-finite*:

assumes [simp]: *finite-measure* M *finite-measure* N
assumes [measurable-cong]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$
shows *finite-measure* $(M \star N)$
unfolding *convolution-def*
by (*intro finite-measure-pair-measure finite-measure.finite-measure-distr*) *auto*

lemma *convolution-emeasure-3*:

assumes [simp, measurable]: $A \in \text{sets borel}$
assumes [simp]: *finite-measure* M *finite-measure* N *finite-measure* L
assumes [simp]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$ *sets* $L = \text{sets borel}$
shows *emeasure* $(L \star (M \star N)) A = \int^+ x. \int^+ y. \int^+ z. \text{indicator } A (x + y + z) \partial N \partial M \partial L$
apply (*subst nn-integral-indicator*[symmetric], *simp*)
apply (*subst nn-integral-convolution*,
auto intro!: borel-measurable-indicator borel-measurable-indicator' convolution-finite)
by (*rule nn-integral-cong*) + (*auto simp: semigroup-add-class.add.assoc*)

lemma *convolution-emeasure-3'*:

assumes [simp, measurable]: $A \in \text{sets borel}$
assumes [simp]: *finite-measure* M *finite-measure* N *finite-measure* L
assumes [measurable-cong, simp]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$ *sets* $L = \text{sets borel}$
shows *emeasure* $((L \star M) \star N) A = \int^+ x. \int^+ y. \int^+ z. \text{indicator } A (x + y + z) \partial N \partial M \partial L$
apply (*subst nn-integral-indicator*[symmetric], *simp*) +
apply (*subst nn-integral-convolution*)
apply (*simp-all add: convolution-finite*)
apply (*subst nn-integral-convolution*)
apply (*simp-all add: finite-measure.sigma-finite-measure sigma-finite-measure.borel-measurable-nn-integral*)
done

lemma *convolution-commutative*:

assumes [simp]: *finite-measure* M *finite-measure* N
assumes [measurable-cong, simp]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$
shows $(M \star N) = (N \star M)$
proof (*rule measure-eqI*)
interpret M : *finite-measure* M **by** *fact*

interpret N : *finite-measure* N **by** *fact*
interpret *pair-sigma-finite* M N ..

show *sets* $(M \star N) = \text{sets } (N \star M)$ **by** *simp*

fix A **assume** $A \in \text{sets } (M \star N)$
then have $1[\text{measurable}]: A \in \text{sets borel}$ **by** *simp*
have *emeasure* $(M \star N) A = \int^+ x. \int^+ y. \text{indicator } A (x + y) \partial N \partial M$ **by** (*auto intro!*: *convolution-emeasure'*)
also have $\dots = \int^+ x. \int^+ y. (\lambda(x,y). \text{indicator } A (x + y)) (x, y) \partial N \partial M$ **by** (*auto intro!*: *nn-integral-cong*)
also have $\dots = \int^+ y. \int^+ x. (\lambda(x,y). \text{indicator } A (x + y)) (x, y) \partial M \partial N$ **by** (*rule Fubini[symmetric]*) *simp*
also have $\dots = \text{emeasure } (N \star M) A$ **by** (*auto intro!*: *nn-integral-cong simp: add commute convolution-emeasure'*)
finally show *emeasure* $(M \star N) A = \text{emeasure } (N \star M) A$ **by** *simp*
qed

lemma *convolution-associative*:
assumes [*simp*]: *finite-measure* M *finite-measure* N *finite-measure* L
assumes [*simp*]: *sets* $N = \text{sets borel}$ *sets* $M = \text{sets borel}$ *sets* $L = \text{sets borel}$
shows $(L \star (M \star N)) = ((L \star M) \star N)$
by (*auto intro!*: *measure-eqI simp: convolution-emeasure-3 convolution-emeasure-3'*)

lemma (*in prob-space*) *sum-indep-random-variable*:
assumes *ind*: *indep-var* *borel* X *borel* Y
assumes [*simp*, *measurable*]: *random-variable* *borel* X
assumes [*simp*, *measurable*]: *random-variable* *borel* Y
shows *distr* M *borel* $(\lambda x. X x + Y x) = \text{convolution } (\text{distr } M \text{ borel } X) (\text{distr } M \text{ borel } Y)$
using *ind* **unfolding** *indep-var-distribution-eq convolution-def*
by (*auto simp: distr-distr intro! arg-cong[where f = distr M borel]*)

lemma (*in prob-space*) *sum-indep-random-variable-lborel*:
assumes *ind*: *indep-var* *borel* X *borel* Y
assumes [*simp*, *measurable*]: *random-variable* *lborel* X
assumes [*simp*, *measurable*]: *random-variable* *lborel* Y
shows *distr* M *lborel* $(\lambda x. X x + Y x) = \text{convolution } (\text{distr } M \text{ lborel } X) (\text{distr } M \text{ lborel } Y)$
using *ind* **unfolding** *indep-var-distribution-eq convolution-def*
by (*auto simp: distr-distr o-def intro! arg-cong[where f = distr M borel] cong: distr-cong*)

lemma *convolution-density*:
fixes $f g :: \text{real} \Rightarrow \text{ennreal}$
assumes [*measurable*]: $f \in \text{borel-measurable borel}$ $g \in \text{borel-measurable borel}$
assumes [*simp*]: *finite-measure* (*density* *lborel* f) *finite-measure* (*density* *lborel* g)
shows *density* *lborel* $f \star \text{density$ *lborel* $g = \text{density$ *lborel* $(\lambda x. \int^+ y. f (x - y) \star g y \partial \text{lborel})$

```

    (is ?l = ?r)
  proof (intro measure-eqI)
    fix A assume A ∈ sets ?l
    then have [measurable]: A ∈ sets borel
      by simp

  have (∫+x. f x * (∫+y. g y * indicator A (x + y) ∂lborel) ∂lborel) =
    (∫+x. (∫+y. g y * (f x * indicator A (x + y)) ∂lborel) ∂lborel)
  proof (intro nn-integral-cong-AE, eventually-elim)
    fix x
    have f x * (∫+y. g y * indicator A (x + y) ∂lborel) =
      (∫+y. f x * (g y * indicator A (x + y)) ∂lborel)
    by (intro nn-integral-cmult[symmetric]) auto
    then show f x * (∫+y. g y * indicator A (x + y) ∂lborel) =
      (∫+y. g y * (f x * indicator A (x + y)) ∂lborel)
    by (simp add: ac-simps)
  qed
  also have ... = (∫+y. (∫+x. g y * (f x * indicator A (x + y)) ∂lborel) ∂lborel)
    by (intro lborel-pair.Fubini') simp
  also have ... = (∫+y. (∫+x. f (x - y) * g y * indicator A x ∂lborel) ∂lborel)
  proof (intro nn-integral-cong-AE, eventually-elim)
    fix y
    have (∫+x. g y * (f x * indicator A (x + y)) ∂lborel) =
      g y * (∫+x. f x * indicator A (x + y) ∂lborel)
    by (intro nn-integral-cmult) auto
    also have ... = g y * (∫+x. f (x - y) * indicator A x ∂lborel)
    by (subst nn-integral-real-affine[where c=1 and t=-y])
      (auto simp add: one-ennreal-def[symmetric])
    also have ... = (∫+x. g y * (f (x - y) * indicator A x) ∂lborel)
    by (intro nn-integral-cmult[symmetric]) auto
    finally show (∫+x. g y * (f x * indicator A (x + y)) ∂lborel) =
      (∫+x. f (x - y) * g y * indicator A x ∂lborel)
    by (simp add: ac-simps)
  qed
  also have ... = (∫+x. (∫+y. f (x - y) * g y * indicator A x ∂lborel) ∂lborel)
    by (intro lborel-pair.Fubini') simp
  finally show emeasure ?l A = emeasure ?r A
    by (auto simp: convolution-emeasure' nn-integral-density emeasure-density
      nn-integral-multc)
  qed simp

lemma (in prob-space) distributed-finite-measure-density:
  distributed M N X f ⟹ finite-measure (density N f)
  using finite-measure-distr[of X N] distributed-distr-eq-density[of M N X f] by
  simp

lemma (in prob-space) distributed-convolution:
  fixes f :: real ⇒ -

```



```

fixes  $g :: \text{real} \Rightarrow -$ 
assumes  $\text{indep}: \text{indep-var borel } X \text{ borel } Y$ 
assumes  $X: \text{distributed } M \text{ lborel } X f$ 
assumes  $Y: \text{distributed } M \text{ lborel } Y g$ 
shows  $\text{distributed } M \text{ lborel } (\lambda x. X x + Y x) (\lambda x. \int^+ y. f (x - y) * g y \partial \text{lborel})$ 
unfolding  $\text{distributed-def}$ 
proof  $\text{safe}$ 
  have  $fg[\text{measurable}]: f \in \text{borel-measurable borel } g \in \text{borel-measurable borel}$ 
  using  $\text{distributed-borel-measurable}[OF X] \text{ distributed-borel-measurable}[OF Y]$ 
by  $\text{simp-all}$ 

  show  $(\lambda x. \int^+ xa. f (x - xa) * g xa \partial \text{lborel}) \in \text{borel-measurable lborel}$ 
  by  $\text{measurable}$ 

  have  $\text{distr } M \text{ borel } (\lambda x. X x + Y x) = (\text{distr } M \text{ borel } X \star \text{distr } M \text{ borel } Y)$ 
  using  $\text{distributed-measurable}[OF X] \text{ distributed-measurable}[OF Y]$ 
  by  $(\text{intro sum-indep-random-variable}) (\text{auto simp: indep})$ 
  also have  $\dots = (\text{density lborel } f \star \text{density lborel } g)$ 
  using  $\text{distributed-distr-eq-density}[OF X] \text{ distributed-distr-eq-density}[OF Y]$ 
  by  $(\text{simp cong: distr-cong})$ 
  also have  $\dots = \text{density lborel } (\lambda x. \int^+ y. f (x - y) * g y \partial \text{lborel})$ 
  proof  $(\text{rule convolution-density})$ 
    show  $\text{finite-measure } (\text{density lborel } f)$ 
    using  $X$  by  $(\text{rule distributed-finite-measure-density})$ 
    show  $\text{finite-measure } (\text{density lborel } g)$ 
    using  $Y$  by  $(\text{rule distributed-finite-measure-density})$ 
  qed fact+
  finally show  $\text{distr } M \text{ lborel } (\lambda x. X x + Y x) = \text{density lborel } (\lambda x. \int^+ y. f (x - y) * g y \partial \text{lborel})$ 
  by  $(\text{simp cong: distr-cong})$ 
  show  $\text{random-variable lborel } (\lambda x. X x + Y x)$ 
  using  $\text{distributed-measurable}[OF X] \text{ distributed-measurable}[OF Y]$  by  $\text{simp}$ 
qed

lemma  $\text{prob-space-convolution-density}$ :
  fixes  $f :: \text{real} \Rightarrow -$ 
  fixes  $g :: \text{real} \Rightarrow -$ 
  assumes  $[\text{measurable}]: f \in \text{borel-measurable borel}$ 
  assumes  $[\text{measurable}]: g \in \text{borel-measurable borel}$ 
  assumes  $\text{gt-0}[\text{simp}]: \bigwedge x. 0 \leq f x \bigwedge x. 0 \leq g x$ 
  assumes  $\text{prob-space } (\text{density lborel } f) (\text{is prob-space } ?F)$ 
  assumes  $\text{prob-space } (\text{density lborel } g) (\text{is prob-space } ?G)$ 
  shows  $\text{prob-space } (\text{density lborel } (\lambda x. \int^+ y. f (x - y) * g y \partial \text{lborel})) (\text{is prob-space } ?D)$ 
proof  $(\text{subst convolution-density}[\text{symmetric}])$ 
  interpret  $F: \text{prob-space } ?F$  by  $\text{fact}$ 
  show  $\text{finite-measure } ?F$  by  $\text{unfold-locales}$ 
  interpret  $G: \text{prob-space } ?G$  by  $\text{fact}$ 
  show  $\text{finite-measure } ?G$  by  $\text{unfold-locales}$ 

```

```

interpret FG: pair-prob-space ?F ?G ..

show prob-space (density lborel f ★ density lborel g)
  unfolding convolution-def by (rule FG.prob-space-distr) simp
qed simp-all

end

```

12 Information theory

```

theory Information
imports
  Independent-Family
begin

```

12.1 Information theory

```

locale information-space = prob-space +
  fixes b :: real assumes b-gt-1: 1 < b

```

Introduce some simplification rules for logarithm of base b .

```

lemmas log-simps = log-mult log-inverse log-divide

```

12.2 Kullback–Leibler divergence

The Kullback–Leibler divergence is also known as relative entropy or Kullback–Leibler distance.

definition

$$\text{entropy-density } b \ M \ N = \log b \circ \text{enn2real} \circ \text{RN-deriv } M \ N$$

definition

$$\text{KL-divergence } b \ M \ N = \text{integral}^L \ N \ (\text{entropy-density } b \ M \ N)$$

lemma *measurable-entropy-density*[*measurable*]: *entropy-density* $b \ M \ N \in \text{borel-measurable } M$

```

unfolding entropy-density-def by auto

```

lemma (in *sigma-finite-measure*) *KL-density*:

```

fixes f :: 'a ⇒ real

```

```

assumes 1 < b

```

```

assumes f[measurable]: f ∈ borel-measurable M and nn: AE x in M. 0 ≤ f x

```

```

shows KL-divergence b M (density M f) = (∫ x. f x * log b (f x) ∂M)

```

```

unfolding KL-divergence-def

```

proof (*subst integral-real-density*)

```

show [measurable]: entropy-density b M (density M (λx. ennreal (f x))) ∈ borel-measurable M

```

```

using f

```

```

by (auto simp: comp-def entropy-density-def)

```

```

have density M (RN-deriv M (density M f)) = density M f
  using f nn by (intro density-RN-deriv-density) auto
then have eq: AE x in M. RN-deriv M (density M f) x = f x
  using f nn by (intro density-unique) auto
have AE x in M. f x * entropy-density b M (density M (λx. ennreal (f x))) x =
  f x * log b (f x)
  using eq nn by (auto simp: entropy-density-def)
then show (∫ x. f x * entropy-density b M (density M (λx. ennreal (f x))) x
  ∂M) = (∫ x. f x * log b (f x) ∂M)
  by (intro integral-cong-AE) measurable
qed fact+

```

```

lemma (in sigma-finite-measure) KL-density-density:
  fixes f g :: 'a ⇒ real
  assumes 1 < b
  assumes f: f ∈ borel-measurable M AE x in M. 0 ≤ f x
  assumes g: g ∈ borel-measurable M AE x in M. 0 ≤ g x
  assumes ac: AE x in M. f x = 0 ⟶ g x = 0
  shows KL-divergence b (density M f) (density M g) = (∫ x. g x * log b (g x / f
x) ∂M)
proof -
  interpret Mf: sigma-finite-measure density M f
  using f by (subst sigma-finite-iff-density-finite) auto
  have KL-divergence b (density M f) (density M g) =
    KL-divergence b (density M f) (density (density M f) (λx. g x / f x))
  using f g ac by (subst density-density-divide) simp-all
  also have ... = (∫ x. (g x / f x) * log b (g x / f x) ∂density M f)
  using f g ⟨1 < b⟩ by (intro Mf.KL-density) (auto simp: AE-density)
  also have ... = (∫ x. g x * log b (g x / f x) ∂M)
  using ac f g ⟨1 < b⟩ by (subst integral-density) (auto intro!: integral-cong-AE)
  finally show ?thesis .
qed

```

```

lemma (in information-space) KL-gt-0:
  fixes D :: 'a ⇒ real
  assumes prob-space (density M D)
  assumes D: D ∈ borel-measurable M AE x in M. 0 ≤ D x
  assumes int: integrable M (λx. D x * log b (D x))
  assumes A: density M D ≠ M
  shows 0 < KL-divergence b M (density M D)
proof -
  interpret N: prob-space density M D by fact

```

```

obtain A where A ∈ sets M emeasure (density M D) A ≠ emeasure M A
  using measure-eqI[of density M D M] ⟨density M D ≠ M⟩ by auto

```

```

let ?D-set = {x ∈ space M. D x ≠ 0}
have [simp, intro]: ?D-set ∈ sets M
  using D by auto

```

```

have D-neg: ( $\int^+ x. \text{ennreal } (- D x) \partial M$ ) = 0
  using D by (subst nn-integral-0-iff-AE) (auto simp: ennreal-neg)

have ( $\int^+ x. \text{ennreal } (D x) \partial M$ ) =  $\text{emeasure } (\text{density } M D) (\text{space } M)$ 
  using D by (simp add: emeasure-density cong: nn-integral-cong)
then have D-pos: ( $\int^+ x. \text{ennreal } (D x) \partial M$ ) = 1
  using N.emeasure-space-1 by simp

have integrable M D
  using D D-pos D-neg unfolding real-integrable-def real-lebesgue-integral-def by
simp-all
then have integralL M D = 1
  using D D-pos D-neg by (simp add: real-lebesgue-integral-def)

have  $0 \leq 1 - \text{measure } M \text{ ?D-set}$ 
  using prob-le-1 by (auto simp: field-simps)
also have ... = ( $\int x. D x - \text{indicator ?D-set } x \partial M$ )
  using ⟨integrable M D⟩ ⟨integralL M D = 1⟩
  by (simp add: emeasure-eq-measure)
also have ... < ( $\int x. D x * (\ln b * \log b (D x)) \partial M$ )
proof (rule integral-less-AE)
  show integrable M ( $\lambda x. D x - \text{indicator ?D-set } x$ )
    using ⟨integrable M D⟩ by (auto simp: less-top[symmetric])
next
  from integrable-mult-left(1)[OF int, of ln b]
  show integrable M ( $\lambda x. D x * (\ln b * \log b (D x))$ )
    by (simp add: ac-simps)
next
  show  $\text{emeasure } M \{x \in \text{space } M. D x \neq 1 \wedge D x \neq 0\} \neq 0$ 
proof
  assume eq-0:  $\text{emeasure } M \{x \in \text{space } M. D x \neq 1 \wedge D x \neq 0\} = 0$ 
  then have disj:  $AE x \text{ in } M. D x = 1 \vee D x = 0$ 
    using D(1) by (auto intro!: AE-I[OF subset-refl] sets.sets-Collect)

  have  $\text{emeasure } M \{x \in \text{space } M. D x = 1\} = (\int^+ x. \text{indicator } \{x \in \text{space } M. D x = 1\} x \partial M)$ 
    using D(1) by auto
  also have ... = ( $\int^+ x. \text{ennreal } (D x) \partial M$ )
  using disj by (auto intro!: nn-integral-cong-AE simp: indicator-def one-ennreal-def)
  finally have  $AE x \text{ in } M. D x = 1$ 
    using D D-pos by (intro AE-I-eq-1) auto
  then have ( $\int^+ x. \text{indicator } A x \partial M$ ) = ( $\int^+ x. \text{ennreal } (D x) * \text{indicator } A x \partial M$ )
    by (intro nn-integral-cong-AE) (auto simp: one-ennreal-def[symmetric])
  also have ... =  $\text{density } M D A$ 
    using ⟨ $A \in \text{sets } M$ ⟩ D by (simp add: emeasure-density)
  finally show False using ⟨ $A \in \text{sets } M$ ⟩ ⟨ $\text{emeasure } (\text{density } M D) A \neq \text{emeasure } M A$ ⟩ by simp

```

```

qed
show { $x \in \text{space } M. D x \neq 1 \wedge D x \neq 0$ }  $\in \text{sets } M$ 
  using  $D(1)$  by (auto intro: sets.sets-Collect-conj)

have False
  if  $Dt: t \in \text{space } M D t \neq 1 D t \neq 0 0 \leq D t$ 
    and  $eq: D t - \text{indicator } ?D\text{-set } t = D t * (\ln b * \log b (D t))$  for  $t$ 
proof -
  have  $D t - 1 = D t - \text{indicator } ?D\text{-set } t$ 
    using  $Dt$  by simp
  also note  $eq$ 
  also have  $D t * (\ln b * \log b (D t)) = - D t * \ln (1 / D t)$ 
    using  $b\text{-gt-1 } \langle D t \neq 0 \rangle \langle 0 \leq D t \rangle$ 
    by (simp add: log-def ln-div less-le)
  finally have  $\ln (1 / D t) = 1 / D t - 1$ 
    using  $\langle D t \neq 0 \rangle$  by (auto simp: field-simps)
  from  $\ln\text{-eq-minus-one}[OF - this] \langle D t \neq 0 \rangle \langle 0 \leq D t \rangle \langle D t \neq 1 \rangle$ 
  show False by auto
qed
with  $D(2)$ 
show  $AE t \text{ in } M. t \in \{x \in \text{space } M. D x \neq 1 \wedge D x \neq 0\} \longrightarrow$ 
   $D t - \text{indicator } ?D\text{-set } t \neq D t * (\ln b * \log b (D t))$ 
  by fastforce

show  $AE t \text{ in } M. D t - \text{indicator } ?D\text{-set } t \leq D t * (\ln b * \log b (D t))$ 
  using  $D(2)$   $AE\text{-space}$ 
proof eventually-elim
  fix  $t$  assume  $t \in \text{space } M 0 \leq D t$ 
  show  $D t - \text{indicator } ?D\text{-set } t \leq D t * (\ln b * \log b (D t))$ 
  proof cases
    assume  $asm: D t \neq 0$ 
    then have  $0 < D t$  using  $\langle 0 \leq D t \rangle$  by auto
    then have  $0 < 1 / D t$  by auto
    have  $D t - \text{indicator } ?D\text{-set } t \leq - D t * (1 / D t - 1)$ 
      using  $asm \langle t \in \text{space } M \rangle$  by (simp add: field-simps)
    also have  $- D t * (1 / D t - 1) \leq - D t * \ln (1 / D t)$ 
      using  $\ln\text{-le-minus-one } \langle 0 < 1 / D t \rangle$  by (intro mult-left-mono-neg) auto
    also have  $\dots = D t * (\ln b * \log b (D t))$ 
      using  $\langle 0 < D t \rangle b\text{-gt-1}$ 
      by (simp-all add: log-def ln-div)
    finally show ?thesis by simp
  qed simp
qed
qed
also have  $\dots = (\int x. \ln b * (D x * \log b (D x)) \partial M)$ 
  by (simp add: ac-simps)
also have  $\dots = \ln b * (\int x. D x * \log b (D x) \partial M)$ 
  using  $\text{int}$  by simp
finally show ?thesis

```

using $b\text{-gt-1 } D$ by (subst $KL\text{-density}$) (auto simp: zero-less-mult-iff)
qed

lemma (in $\sigma\text{-finite-measure}$) $KL\text{-same-eq-0}$: $KL\text{-divergence } b \ M \ M = 0$
proof –

have $AE \ x \ in \ M. \ 1 = RN\text{-deriv } M \ M \ x$

proof (rule $RN\text{-deriv-unique}$)

show $density \ M \ (\lambda x. \ 1) = M$

by (simp add: $density\text{-1}$)

qed auto

then have $AE \ x \ in \ M. \ log \ b \ (enn2real \ (RN\text{-deriv } M \ M \ x)) = 0$

by (elim $AE\text{-mp}$) simp

from $integral\text{-cong-}AE[OF \ - \ - \ this]$

have $integral^L \ M \ (entropy\text{-density } b \ M \ M) = 0$

by (simp add: $entropy\text{-density-def comp-def}$)

then show $KL\text{-divergence } b \ M \ M = 0$

unfolding $KL\text{-divergence-def}$

by auto

qed

lemma (in $information\text{-space}$) $KL\text{-eq-0-iff-eq}$:

fixes $D :: 'a \Rightarrow real$

assumes $prob\text{-space } (density \ M \ D)$

assumes D : $D \in borel\text{-measurable } M \ AE \ x \ in \ M. \ 0 \leq D \ x$

assumes int : $integrable \ M \ (\lambda x. \ D \ x * log \ b \ (D \ x))$

shows $KL\text{-divergence } b \ M \ (density \ M \ D) = 0 \longleftrightarrow density \ M \ D = M$

using $KL\text{-same-eq-0}[of \ b] \ KL\text{-gt-0}[OF \ assms]$

by (auto simp: less-le)

lemma (in $information\text{-space}$) $KL\text{-eq-0-iff-eq-ac}$:

fixes $D :: 'a \Rightarrow real$

assumes $prob\text{-space } N$

assumes ac : $absolutely\text{-continuous } M \ N \ sets \ N = sets \ M$

assumes int : $integrable \ N \ (entropy\text{-density } b \ M \ N)$

shows $KL\text{-divergence } b \ M \ N = 0 \longleftrightarrow N = M$

proof –

interpret N : $prob\text{-space } N$ by fact

have $finite\text{-measure } N$ by $unfold\text{-locales}$

from $real\text{-}RN\text{-deriv}[OF \ this \ ac]$ obtain D

where D :

$random\text{-variable } borel \ D$

$AE \ x \ in \ M. \ RN\text{-deriv } M \ N \ x = ennreal \ (D \ x)$

$AE \ x \ in \ N. \ 0 < D \ x$

$\bigwedge x. \ 0 \leq D \ x$

by this auto

have $N = density \ M \ (RN\text{-deriv } M \ N)$

using ac by (rule $density\text{-}RN\text{-deriv}[symmetric]$)

also have $\dots = density \ M \ D$

```

    using D by (auto intro!: density-cong)
    finally have N: N = density M D .

from absolutely-continuous-AE[OF ac(2,1) D(2)] D b-gt-1 ac measurable-entropy-density
have integrable N ( $\lambda x. \log b (D x)$ )
  by (intro integrable-cong-AE[THEN iffD2, OF - - - int])
    (auto simp: N entropy-density-def)
with D b-gt-1 have integrable M ( $\lambda x. D x * \log b (D x)$ )
  by (subst integrable-real-density[symmetric]) (auto simp: N[symmetric] comp-def)
with  $\langle \text{prob-space } N \rangle D$  show ?thesis
  unfolding N
  by (intro KL-eq-0-iff-eq) auto
qed

lemma (in information-space) KL-nonneg:
  assumes prob-space (density M D)
  assumes D:  $D \in \text{borel-measurable } M \text{ AE } x \text{ in } M. 0 \leq D x$ 
  assumes int: integrable M ( $\lambda x. D x * \log b (D x)$ )
  shows  $0 \leq \text{KL-divergence } b M (\text{density } M D)$ 
  using KL-gt-0[OF assms] by (cases density M D = M) (auto simp: KL-same-eq-0)

lemma (in sigma-finite-measure) KL-density-density-nonneg:
  fixes f g :: 'a  $\Rightarrow$  real
  assumes 1 < b
  assumes f:  $f \in \text{borel-measurable } M \text{ AE } x \text{ in } M. 0 \leq f x \text{ prob-space } (\text{density } M f)$ 
  assumes g:  $g \in \text{borel-measurable } M \text{ AE } x \text{ in } M. 0 \leq g x \text{ prob-space } (\text{density } M g)$ 
  assumes ac:  $\text{AE } x \text{ in } M. f x = 0 \longrightarrow g x = 0$ 
  assumes int: integrable M ( $\lambda x. g x * \log b (g x / f x)$ )
  shows  $0 \leq \text{KL-divergence } b (\text{density } M f) (\text{density } M g)$ 
proof -
  interpret Mf: prob-space density M f by fact
  interpret Mf: information-space density M f b by standard fact
  have eq:  $\text{density } (\text{density } M f) (\lambda x. g x / f x) = \text{density } M g$  (is ?DD = -)
    using f g ac by (subst density-density-divide) simp-all
  have  $0 \leq \text{KL-divergence } b (\text{density } M f) (\text{density } (\text{density } M f) (\lambda x. g x / f x))$ 
  proof (rule Mf.KL-nonneg)
    show prob-space ?DD unfolding eq by fact
    from f g show  $(\lambda x. g x / f x) \in \text{borel-measurable } (\text{density } M f)$ 
      by auto
    show  $\text{AE } x \text{ in } \text{density } M f. 0 \leq g x / f x$ 
      using f g by (auto simp: AE-density)
    show integrable (density M f) ( $\lambda x. g x / f x * \log b (g x / f x)$ )
      using  $\langle 1 < b \rangle f g ac$ 
      by (subst integrable-density)
      (auto intro!: integrable-cong-AE[THEN iffD2, OF - - - int] measurable-If)
  qed
qed

```

also have ... = *KL-divergence* *b* (*density* *M* *f*) (*density* *M* *g*)
 using *f g ac* by (*subst density-density-divide*) *simp-all*
 finally show ?thesis .
 qed

12.3 Finite Entropy

definition (in *information-space*) *finite-entropy* :: 'b *measure* \Rightarrow ('a \Rightarrow 'b) \Rightarrow ('b \Rightarrow real) \Rightarrow bool

where

finite-entropy *S* *X* *f* \longleftrightarrow
distributed *M* *S* *X* *f* \wedge
integrable *S* ($\lambda x. f\ x * \log b\ (f\ x)$) \wedge
 $(\forall x \in \text{space } S. 0 \leq f\ x)$

lemma (in *information-space*) *finite-entropy-simple-function*:

assumes *X*: *simple-function* *M* *X*

shows *finite-entropy* (*count-space* (*X*'*space* *M*)) *X* ($\lambda a. \text{measure } M\ \{x \in \text{space } M. X\ x = a\}$)

unfolding *finite-entropy-def*

proof *safe*

have [*simp*]: *finite* (*X* ' *space* *M*)

using *X* **by** (*auto simp: simple-function-def*)

then show *integrable* (*count-space* (*X* ' *space* *M*))

($\lambda x. \text{prob } \{xa \in \text{space } M. X\ xa = x\} * \log b\ (\text{prob } \{xa \in \text{space } M. X\ xa = x\})$)

by (*rule integrable-count-space*)

have *d*: *distributed* *M* (*count-space* (*X* ' *space* *M*)) *X* ($\lambda x. \text{ennreal } (\text{if } x \in X'\text{space } M \text{ then } \text{prob } \{xa \in \text{space } M. X\ xa = x\} \text{ else } 0)$)

by (*rule distributed-simple-function-superset[OF X]*) (*auto intro!: arg-cong[where f=prob]*)

show *distributed* *M* (*count-space* (*X* ' *space* *M*)) *X* ($\lambda x. \text{ennreal } (\text{prob } \{xa \in \text{space } M. X\ xa = x\})$)

by (*rule distributed-cong-density[THEN iffD1, OF - - d]*) *auto*

qed (*rule measure-nonneg*)

lemma *ac-fst*:

assumes *sigma-finite-measure* *T*

shows *absolutely-continuous* *S* (*distr* (*S* \otimes_M *T*) *S* *fst*)

proof –

interpret *sigma-finite-measure* *T* **by** *fact*

{ **fix** *A* **assume** *A*: *A* \in *sets* *S* *emeasure* *S* *A* = 0

then have *fst* – ' *A* \cap *space* (*S* \otimes_M *T*) = *A* \times *space* *T*

by (*auto simp: space-pair-measure dest!: sets.sets-into-space*)

with *A* **have** *emeasure* (*S* \otimes_M *T*) (*fst* – ' *A* \cap *space* (*S* \otimes_M *T*)) = 0

by (*simp add: emeasure-pair-measure-Times*) }

then show ?thesis

unfolding *absolutely-continuous-def*

by (*metis emeasure-distr measurable-fst null-setsD1 null-setsD2 null-setsI sets-distr subsetI*)

qed

lemma *ac-snd*:

assumes *sigma-finite-measure* *T*

shows *absolutely-continuous* *T* (*distr* (*S* \otimes_M *T*) *T* *snd*)

proof –

interpret *sigma-finite-measure* *T* **by** *fact*

{ **fix** *A* **assume** *A*: *A* \in *sets* *T* *emeasure* *T* *A* = 0

then have *snd* – ‘*A* \cap *space* (*S* \otimes_M *T*) = *space* *S* \times *A*

by (*auto simp: space-pair-measure dest!: sets.sets-into-space*)

with *A* **have** *emeasure* (*S* \otimes_M *T*) (*snd* – ‘*A* \cap *space* (*S* \otimes_M *T*)) = 0

by (*simp add: emeasure-pair-measure-Times*) }

then show *?thesis*

unfolding *absolutely-continuous-def*

by (*metis emeasure-distr measurable-snd null-setsD1 null-setsD2 null-setsI sets-distr subsetI*)

qed

lemma (*in information-space*) *finite-entropy-integrable*:

finite-entropy *S* *X* *Px* \implies *integrable* *S* ($\lambda x. Px\ x * \log b\ (Px\ x)$)

unfolding *finite-entropy-def* **by** *auto*

lemma (*in information-space*) *finite-entropy-distributed*:

finite-entropy *S* *X* *Px* \implies *distributed* *M* *S* *X* *Px*

unfolding *finite-entropy-def* **by** *auto*

lemma (*in information-space*) *finite-entropy-nn*:

finite-entropy *S* *X* *Px* $\implies x \in \text{space } S \implies 0 \leq Px\ x$

by (*auto simp: finite-entropy-def*)

lemma (*in information-space*) *finite-entropy-measurable*:

finite-entropy *S* *X* *Px* $\implies Px \in S \rightarrow_M \text{borel}$

using *distributed-real-measurable*[*of S Px M X*]

finite-entropy-nn[*of S X Px*] *finite-entropy-distributed*[*of S X Px*] **by** *auto*

lemma (*in information-space*) *subdensity-finite-entropy*:

fixes *g* :: ‘*b* \Rightarrow *real* **and** *f* :: ‘*c* \Rightarrow *real*

assumes *T*: *T* \in *measurable* *P* *Q*

assumes *f*: *finite-entropy* *P* *X* *f*

assumes *g*: *finite-entropy* *Q* *Y* *g*

assumes *Y*: *Y* = *T* \circ *X*

shows *AE* *x* *in* *P*. *g* (*T* *x*) = 0 $\longrightarrow f\ x = 0$

using *subdensity*[*OF T*, *of M X* $\lambda x. ennreal\ (f\ x)\ Y\ \lambda x. ennreal\ (g\ x)$]

finite-entropy-distributed[*OF f*] *finite-entropy-distributed*[*OF g*]

finite-entropy-nn[*OF f*] *finite-entropy-nn*[*OF g*]

assms

by *auto*

lemma (*in information-space*) *finite-entropy-integrable-transform*:

$\text{finite-entropy } S \ X \ Px \implies \text{distributed } M \ T \ Y \ Py \implies (\bigwedge x. x \in \text{space } T \implies 0 \leq Py \ x) \implies$
 $X = (\lambda x. f \ (Y \ x)) \implies f \in \text{measurable } T \ S \implies \text{integrable } T \ (\lambda x. Py \ x * \log b \ (Px \ (f \ x)))$
using *distributed-transform-integrable*[of $M \ T \ Y \ Py \ S \ X \ Px \ f \ \lambda x. \log b \ (Px \ x)$]
using *distributed-real-measurable*[of $S \ Px \ M \ X$]
by (*auto simp: finite-entropy-def*)

12.4 Mutual Information

definition (*in prob-space*)

$\text{mutual-information } b \ S \ T \ X \ Y =$
 $\text{KL-divergence } b \ (\text{distr } M \ S \ X \otimes_M \text{distr } M \ T \ Y) \ (\text{distr } M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x)))$

lemma (*in information-space*) *mutual-information-indep-vars*:

fixes $S \ T \ X \ Y$
defines $P \equiv \text{distr } M \ S \ X \otimes_M \text{distr } M \ T \ Y$
defines $Q \equiv \text{distr } M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x))$
shows $\text{indep-var } S \ X \ T \ Y \longleftrightarrow$
 $(\text{random-variable } S \ X \wedge \text{random-variable } T \ Y \wedge$
 $\text{absolutely-continuous } P \ Q \wedge \text{integrable } Q \ (\text{entropy-density } b \ P \ Q) \wedge$
 $\text{mutual-information } b \ S \ T \ X \ Y = 0)$
unfolding *indep-var-distribution-eq*

proof *safe*

assume $rv[\text{measurable}]$: *random-variable* $S \ X$ *random-variable* $T \ Y$

interpret X : *prob-space* $\text{distr } M \ S \ X$

by (*rule prob-space-distr*) *fact*

interpret Y : *prob-space* $\text{distr } M \ T \ Y$

by (*rule prob-space-distr*) *fact*

interpret XY : *pair-prob-space* $\text{distr } M \ S \ X \ \text{distr } M \ T \ Y$ **by** *standard*

interpret P : *information-space* $P \ b$ **unfolding** $P\text{-def}$ **by** *standard* (*rule b-gt-1*)

interpret Q : *prob-space* Q **unfolding** $Q\text{-def}$

by (*rule prob-space-distr*) *simp*

{ **assume** $\text{distr } M \ S \ X \otimes_M \text{distr } M \ T \ Y = \text{distr } M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x))$

then have [*simp*]: $Q = P$ **unfolding** $Q\text{-def}$ $P\text{-def}$ **by** *simp*

show *ac*: *absolutely-continuous* $P \ Q$ **by** (*simp add: absolutely-continuous-def*)

then have *ed*: *entropy-density* $b \ P \ Q \in \text{borel-measurable } P$

by *simp*

have $AE \ x \ \text{in } P. \ 1 = \text{RN-deriv } P \ Q \ x$

proof (*rule P.RN-deriv-unique*)

show *density* $P \ (\lambda x. 1) = Q$

unfolding $\langle Q = P \rangle$ **by** (*intro measure-eqI*) (*auto simp: emeasure-density*)

```

qed auto
then have ae-0: AE x in P. entropy-density b P Q x = 0
  by (auto simp: entropy-density-def)
then have integrable P (entropy-density b P Q)  $\longleftrightarrow$  integrable Q ( $\lambda x. 0::\text{real}$ )
  using ed unfolding  $\langle Q = P \rangle$  by (intro integrable-cong-AE) auto
then show integrable Q (entropy-density b P Q) by simp

from ae-0 have mutual-information b S T X Y = ( $\int x. 0 \partial P$ )
  unfolding mutual-information-def KL-divergence-def P-def[symmetric] Q-def[symmetric]
 $\langle Q = P \rangle$ 
  by (intro integral-cong-AE) auto
then show mutual-information b S T X Y = 0
  by simp }

{ assume ac: absolutely-continuous P Q
  assume int: integrable Q (entropy-density b P Q)
  assume I-eq-0: mutual-information b S T X Y = 0

  have eq: Q = P
  proof (rule P.KL-eq-0-iff-eq-ac[THEN iffD1])
    show prob-space Q by unfold-locales
    show absolutely-continuous P Q by fact
    show integrable Q (entropy-density b P Q) by fact
    show sets Q = sets P by (simp add: P-def Q-def sets-pair-measure)
    show KL-divergence b P Q = 0
    using I-eq-0 unfolding mutual-information-def by (simp add: P-def Q-def)
  qed
  then show distr M S X  $\otimes_M$  distr M T Y = distr M (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ )
    unfolding P-def Q-def .. }
qed

abbreviation (in information-space)
  mutual-information-Pow ( $\langle \mathcal{I}'(-; -) \rangle$ ) where
 $\mathcal{I}(X; Y) \equiv$  mutual-information b (count-space (X'space M)) (count-space (Y'space M)) X Y

lemma (in information-space)
  fixes Pxy :: 'b  $\times$  'c  $\Rightarrow$  real and Px :: 'b  $\Rightarrow$  real and Py :: 'c  $\Rightarrow$  real
  assumes S: sigma-finite-measure S and T: sigma-finite-measure T
  assumes Fx: finite-entropy S X Px and Fy: finite-entropy T Y Py
  assumes Fxy: finite-entropy (S  $\otimes_M$  T) ( $\lambda x. (X x, Y x)$ ) Pxy
  defines f  $\equiv \lambda x. Pxy x * \log b (Pxy x / (Px (fst x) * Py (snd x)))$ 
  shows mutual-information-distr': mutual-information b S T X Y =  $\text{integral}^L (S \otimes_M T) f$  (is ?M = ?R)
  and mutual-information-nonneg':  $0 \leq$  mutual-information b S T X Y
proof -
  have Px: distributed M S X Px and Px-nn:  $\bigwedge x. x \in \text{space } S \implies 0 \leq Px x$ 
  using Fx by (auto simp: finite-entropy-def)

```

```

have Py: distributed M T Y Py and Py-nn:  $\bigwedge x. x \in \text{space } T \implies 0 \leq Py\ x$ 
  using Fy by (auto simp: finite-entropy-def)
have Pxy: distributed M (S  $\otimes_M$  T) ( $\lambda x. (X\ x, Y\ x)$ ) Pxy
  and Pxy-nn:  $\bigwedge x. x \in \text{space } (S \otimes_M T) \implies 0 \leq Pxy\ x$ 
   $\bigwedge x\ y. x \in \text{space } S \implies y \in \text{space } T \implies 0 \leq Pxy\ (x, y)$ 
  using Fxy by (auto simp: finite-entropy-def space-pair-measure)

have [measurable]: Px  $\in S \rightarrow_M \text{borel}$ 
  using Px Px-nn by (intro distributed-real-measurable)
have [measurable]: Py  $\in T \rightarrow_M \text{borel}$ 
  using Py Py-nn by (intro distributed-real-measurable)
have measurable-Pxy[measurable]: Pxy  $\in (S \otimes_M T) \rightarrow_M \text{borel}$ 
  using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

have X[measurable]: random-variable S X
  using Px by auto
have Y[measurable]: random-variable T Y
  using Py by auto
interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret ST: pair-sigma-finite S T ..
interpret X: prob-space distr M S X using X by (rule prob-space-distr)
interpret Y: prob-space distr M T Y using Y by (rule prob-space-distr)
interpret XY: pair-prob-space distr M S X distr M T Y ..
let ?P = S  $\otimes_M$  T
let ?D = distr M ?P ( $\lambda x. (X\ x, Y\ x)$ )

{ fix A assume A  $\in \text{sets } S$ 
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M S X) A = emeasure ?D (A  $\times \text{space } T$ )
  by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq1 = this
{ fix A assume A  $\in \text{sets } T$ 
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M T Y) A = emeasure ?D (space S  $\times$  A)
  by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq2 = this

have distr-eq: distr M S X  $\otimes_M$  distr M T Y = density ?P ( $\lambda x. \text{ennreal } (Px\ (\text{fst } x) * Py\ (\text{snd } x))$ )
  unfolding Px(1)[THEN distributed-distr-eq-density] Py(1)[THEN distributed-distr-eq-density]
  proof (subst pair-measure-density)
    show ( $\lambda x. \text{ennreal } (Px\ x) \in \text{borel-measurable } S\ (\lambda y. \text{ennreal } (Py\ y)) \in \text{borel-measurable } T$ )
      using Px Py by (auto simp: distributed-def)
    show sigma-finite-measure (density T Py) unfolding Py(1)[THEN distributed-distr-eq-density, symmetric] ..
    show density (S  $\otimes_M$  T) ( $\lambda(x, y). \text{ennreal } (Px\ x) * \text{ennreal } (Py\ y) = \text{density } (S \otimes_M T) (\lambda x. \text{ennreal } (Px\ (\text{fst } x) * Py\ (\text{snd } x))$ )

```

```

using  $Px$ -nn  $Py$ -nn by (auto intro!: density-cong simp: distributed-def ennreal-mult space-pair-measure)
qed fact

have  $M$ :  $?M = KL\text{-divergence } b$  (density  $?P$  ( $\lambda x. \text{ennreal } (Px \text{ (fst } x) * Py \text{ (snd } x))$ ) (density  $?P$  ( $\lambda x. \text{ennreal } (Pxy \text{ } x)$ ))
unfolding mutual-information-def distr-eq  $Pxy(1)$ [THEN distributed-distr-eq-density]
..

from  $Px$   $Py$  have  $f$ : ( $\lambda x. Px \text{ (fst } x) * Py \text{ (snd } x) \in \text{borel-measurable } ?P$ 
by (intro borel-measurable-times) (auto intro: distributed-real-measurable measurable-fst'' measurable-snd'')
have  $PxPy\text{-nonneg}$ :  $AE \ x \text{ in } ?P. 0 \leq Px \text{ (fst } x) * Py \text{ (snd } x)$ 
using  $Px$ -nn  $Py$ -nn by (auto simp: space-pair-measure)

have  $A$ : ( $AE \ x \text{ in } ?P. Px \text{ (fst } x) = 0 \longrightarrow Pxy \ x = 0$ )
by (rule subdensity-real[OF measurable-fst  $Pxy \ Px$ ]) (insert  $Px$ -nn  $Pxy$ -nn, auto simp: space-pair-measure)
moreover
have  $B$ : ( $AE \ x \text{ in } ?P. Py \text{ (snd } x) = 0 \longrightarrow Pxy \ x = 0$ )
by (rule subdensity-real[OF measurable-snd  $Pxy \ Py$ ]) (insert  $Py$ -nn  $Pxy$ -nn, auto simp: space-pair-measure)
ultimately have  $ac$ :  $AE \ x \text{ in } ?P. Px \text{ (fst } x) * Py \text{ (snd } x) = 0 \longrightarrow Pxy \ x = 0$ 
by auto

show  $?M = ?R$ 
unfolding  $M$   $f$ -def using  $Pxy$ -nn  $Px$ -nn  $Py$ -nn
by (intro  $ST.KL\text{-density-density } b\text{-gt-1 } f \ PxPy\text{-nonneg } ac$ ) (auto simp: space-pair-measure)

have  $X$ :  $X = \text{fst} \circ (\lambda x. (X \ x, Y \ x))$  and  $Y$ :  $Y = \text{snd} \circ (\lambda x. (X \ x, Y \ x))$ 
by auto

have integrable ( $S \otimes_M T$ ) ( $\lambda x. Pxy \ x * \log b \ (Pxy \ x) - Pxy \ x * \log b \ (Px \text{ (fst } x)) - Pxy \ x * \log b \ (Py \text{ (snd } x))$ )
using finite-entropy-integrable[OF  $Fxy$ ]
using finite-entropy-integrable-transform[OF  $Fx \ Pxy$ , of fst]
using finite-entropy-integrable-transform[OF  $Fy \ Pxy$ , of snd]
by (simp add:  $Pxy$ -nn)
moreover have  $f \in \text{borel-measurable } (S \otimes_M T)$ 
unfolding  $f$ -def using  $Px \ Py \ Pxy$ 
by (auto intro: distributed-real-measurable measurable-fst'' measurable-snd''
intro!: borel-measurable-times borel-measurable-log borel-measurable-divide)
ultimately have  $int$ : integrable ( $S \otimes_M T$ )  $f$ 
proof (rule integrable-cong-AE-imp)
show  $AE \ x \text{ in } S \otimes_M$ 
 $T. Pxy \ x * \log b \ (Pxy \ x) - Pxy \ x * \log b \ (Px \text{ (fst } x)) - Pxy \ x * \log b \ (Py \text{ (snd } x))$ 
 $= f \ x$ 
using  $A \ B$ 

```

```

    by (auto simp: f-def field-simps space-pair-measure log-mult log-divide)
  qed

show 0 ≤ ?M unfolding M
proof (intro ST.KL-density-density-nonneg)
  show prob-space (density (S ⊗M T) (λx. ennreal (Pxy x)))
    unfolding distributed-distr-eq-density[OF Pxy, symmetric]
    using distributed-measurable[OF Pxy] by (rule prob-space-distr)
  show prob-space (density (S ⊗M T) (λx. ennreal (Px (fst x) * Py (snd x))))
    unfolding distr-eq[symmetric] by unfold-locales
  show integrable (S ⊗M T) (λx. Pxy x * log b (Pxy x / (Px (fst x) * Py (snd x))))
    using int unfolding f-def .
  qed (insert ac, auto simp: b-gt-1 Px-nn Py-nn Pxy-nn space-pair-measure)
qed

lemma (in information-space)
  fixes Pxy :: 'b × 'c ⇒ real and Px :: 'b ⇒ real and Py :: 'c ⇒ real
  assumes sigma-finite-measure S sigma-finite-measure T
  assumes Px: distributed M S X Px and Px-nn: ⋀x. x ∈ space S ⇒ 0 ≤ Px x
    and Py: distributed M T Y Py and Py-nn: ⋀y. y ∈ space T ⇒ 0 ≤ Py y
    and Pxy: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
    and Pxy-nn: ⋀x y. x ∈ space S ⇒ y ∈ space T ⇒ 0 ≤ Pxy (x, y)
  defines f ≡ λx. Pxy x * log b (Pxy x / (Px (fst x) * Py (snd x)))
  shows mutual-information-distr: mutual-information b S T X Y = integralL (S
    ⊗M T) f (is ?M = ?R)
    and mutual-information-nonneg: integrable (S ⊗M T) f ⇒ 0 ≤ mutual-information
    b S T X Y
proof -
  have X[measurable]: random-variable S X
    using Px by (auto simp: distributed-def)
  have Y[measurable]: random-variable T Y
    using Py by (auto simp: distributed-def)
  have [measurable]: Px ∈ S →M borel
    using Px Px-nn by (intro distributed-real-measurable)
  have [measurable]: Py ∈ T →M borel
    using Py Py-nn by (intro distributed-real-measurable)
  have measurable-Pxy[measurable]: Pxy ∈ (S ⊗M T) →M borel
    using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..
  interpret X: prob-space distr M S X using X by (rule prob-space-distr)
  interpret Y: prob-space distr M T Y using Y by (rule prob-space-distr)
  interpret XY: pair-prob-space distr M S X distr M T Y ..
  let ?P = S ⊗M T
  let ?D = distr M ?P (λx. (X x, Y x))

```

```

{ fix A assume A ∈ sets S
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M S X) A = emeasure ?D (A × space T)
    by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq1 = this
{ fix A assume A ∈ sets T
  with X[THEN measurable-space] Y[THEN measurable-space]
  have emeasure (distr M T Y) A = emeasure ?D (space S × A)
    by (auto simp: emeasure-distr intro!: arg-cong[where f=emeasure M]) }
note marginal-eq2 = this

have distr-eq: distr M S X ⊗M distr M T Y = density ?P (λx. ennreal (Px (fst
x) * Py (snd x)))
  unfolding Px(1)[THEN distributed-distr-eq-density] Py(1)[THEN distributed-distr-eq-density]
  proof (subst pair-measure-density)
    show (λx. ennreal (Px x)) ∈ borel-measurable S (λy. ennreal (Py y)) ∈
borel-measurable T
    using Px Py by (auto simp: distributed-def)
    show sigma-finite-measure (density T Py) unfolding Py(1)[THEN distributed-distr-eq-density,
symmetric] ..
    show density (S ⊗M T) (λ(x, y). ennreal (Px x) * ennreal (Py y)) =
density (S ⊗M T) (λx. ennreal (Px (fst x) * Py (snd x)))
    using Px-nn Py-nn by (auto intro!: density-cong simp: distributed-def en-
nreal-mult space-pair-measure)
  qed fact

have M: ?M = KL-divergence b (density ?P (λx. ennreal (Px (fst x) * Py (snd
x)))) (density ?P (λx. ennreal (Pxy x)))
  unfolding mutual-information-def distr-eq Pxy(1)[THEN distributed-distr-eq-density]
  ..

from Px Py have f: (λx. Px (fst x) * Py (snd x)) ∈ borel-measurable ?P
  by (intro borel-measurable-times) (auto intro: distributed-real-measurable mea-
surable-fst'' measurable-snd'')
have PxPy-nonneg: AE x in ?P. 0 ≤ Px (fst x) * Py (snd x)
  using Px-nn Py-nn by (auto simp: space-pair-measure)

have (AE x in ?P. Px (fst x) = 0 → Pxy x = 0)
  by (rule subdensity-real[OF measurable-fst Pxy Px]) (insert Px-nn Pxy-nn, auto
simp: space-pair-measure)
moreover
have (AE x in ?P. Py (snd x) = 0 → Pxy x = 0)
  by (rule subdensity-real[OF measurable-snd Pxy Py]) (insert Py-nn Pxy-nn,
auto simp: space-pair-measure)
ultimately have ac: AE x in ?P. Px (fst x) * Py (snd x) = 0 → Pxy x = 0
  by auto

show ?M = ?R
  unfolding M f-def

```

```

using b-gt-1 f PxPy-nonneg ac Pxy-nn
by (intro ST.KL-density-density) (auto simp: space-pair-measure)

assume int: integrable ( $S \otimes_M T$ ) f
show  $0 \leq ?M$  unfolding M
proof (intro ST.KL-density-density-nonneg)
  show prob-space (density ( $S \otimes_M T$ ) ( $\lambda x. \text{ennreal } (Pxy\ x)$ ))
    unfolding distributed-distr-eq-density[OF Pxy, symmetric]
    using distributed-measurable[OF Pxy] by (rule prob-space-distr)
  show prob-space (density ( $S \otimes_M T$ ) ( $\lambda x. \text{ennreal } (Px\ (\text{fst } x) * Py\ (\text{snd } x))$ ))
    unfolding distr-eq[symmetric] by unfold-locales
  show integrable ( $S \otimes_M T$ ) ( $\lambda x. Pxy\ x * \log b\ (Pxy\ x / (Px\ (\text{fst } x) * Py\ (\text{snd } x)))$ )
    using int unfolding f-def .
qed (insert ac, auto simp: b-gt-1 Px-nn Py-nn Pxy-nn space-pair-measure)
qed

lemma (in information-space)
  fixes Pxy :: 'b  $\times$  'c  $\Rightarrow$  real and Px :: 'b  $\Rightarrow$  real and Py :: 'c  $\Rightarrow$  real
  assumes sigma-finite-measure S sigma-finite-measure T
  assumes Px[measurable]: distributed M S X Px and Px-nn:  $\bigwedge x. x \in \text{space } S \Rightarrow 0 \leq Px\ x$ 
  and Py[measurable]: distributed M T Y Py and Py-nn:  $\bigwedge x. x \in \text{space } T \Rightarrow 0 \leq Py\ x$ 
  and Pxy[measurable]: distributed M ( $S \otimes_M T$ ) ( $\lambda x. (X\ x, Y\ x)$ ) Pxy
  and Pxy-nn:  $\bigwedge x. x \in \text{space } (S \otimes_M T) \Rightarrow 0 \leq Pxy\ x$ 
  assumes ae: AE x in S. AE y in T. Pxy (x, y) = Px x * Py y
  shows mutual-information-eq-0: mutual-information b S T X Y = 0
proof –
  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..
  note
    distributed-real-measurable[OF Px-nn Px, measurable]
    distributed-real-measurable[OF Py-nn Py, measurable]
    distributed-real-measurable[OF Pxy-nn Pxy, measurable]

  have AE x in  $S \otimes_M T$ . Px (fst x) = 0  $\longrightarrow$  Pxy x = 0
    by (rule subdensity-real[OF measurable-fst Pxy Px]) (auto simp: Px-nn Pxy-nn space-pair-measure)
  moreover
  have AE x in  $S \otimes_M T$ . Py (snd x) = 0  $\longrightarrow$  Pxy x = 0
    by (rule subdensity-real[OF measurable-snd Pxy Py]) (auto simp: Py-nn Pxy-nn space-pair-measure)
  moreover
  have AE x in  $S \otimes_M T$ . Pxy x = Px (fst x) * Py (snd x)
    by (intro ST.AE-pair-measure) (auto simp: ae intro!: measurable-snd'' measurable-fst'')
  ultimately have AE x in  $S \otimes_M T$ . Pxy x * log b (Pxy x / (Px (fst x) * Py (snd x)))

```



```

(snd x))) = 0
  by eventually-elim simp
  then have (∫ x. Pxy x * log b (Pxy x / (Px (fst x) * Py (snd x))) ∂(S ⊗M T))
= (∫ x. 0 ∂(S ⊗M T))
  by (intro integral-cong-AE) auto
  then show ?thesis
  by (subst mutual-information-distr[OF assms(1-8)]) (auto simp add: space-pair-measure)
qed

```

```

lemma (in information-space) mutual-information-simple-distributed:
  assumes X: simple-distributed M X Px and Y: simple-distributed M Y Py
  assumes XY: simple-distributed M (λx. (X x, Y x)) Pxy
  shows  $\mathcal{I}(X ; Y) = (\sum (x, y) \in (\lambda x. (X x, Y x)) \text{ 'space } M. Pxy (x, y) * \log b (Pxy (x, y) / (Px x * Py y)))$ 
proof (subst mutual-information-distr[OF - - simple-distributed[OF X] - simple-distributed[OF Y] - simple-distributed-joint[OF XY]])
  note fin = simple-distributed-joint-finite[OF XY, simp]
  show sigma-finite-measure (count-space (X ‘ space M))
  by (simp add: sigma-finite-measure-count-space-finite)
  show sigma-finite-measure (count-space (Y ‘ space M))
  by (simp add: sigma-finite-measure-count-space-finite)
  let ?Pxy = λx. (if x ∈ (λx. (X x, Y x)) ‘ space M then Pxy x else 0)
  let ?f = λx. ?Pxy x * log b (?Pxy x / (Px (fst x) * Py (snd x)))
  have ∧x. ?f x = (if x ∈ (λx. (X x, Y x)) ‘ space M then Pxy x * log b (Pxy x / (Px (fst x) * Py (snd x))) else 0)
  by auto
  with fin show (∫ x. ?f x ∂(count-space (X ‘ space M) ⊗M count-space (Y ‘ space M))) =
    (∑ (x, y) ∈ (λx. (X x, Y x)) ‘ space M. Pxy (x, y) * log b (Pxy (x, y) / (Px x * Py y)))
  by (auto simp add: pair-measure-count-space lebesgue-integral-count-space-finite sum.If-cases split-beta'
    intro!: sum.cong)
qed (insert X Y XY, auto simp: simple-distributed-def)

```

```

lemma (in information-space)
  fixes Pxy :: 'b × 'c ⇒ real and Px :: 'b ⇒ real and Py :: 'c ⇒ real
  assumes Px: simple-distributed M X Px and Py: simple-distributed M Y Py
  assumes Pxy: simple-distributed M (λx. (X x, Y x)) Pxy
  assumes ae: ∀ x ∈ space M. Pxy (X x, Y x) = Px (X x) * Py (Y x)
  shows mutual-information-eq-0-simple:  $\mathcal{I}(X ; Y) = 0$ 
proof (subst mutual-information-simple-distributed[OF Px Py Pxy])
  have (∑ (x, y) ∈ (λx. (X x, Y x)) ‘ space M. Pxy (x, y) * log b (Pxy (x, y) / (Px x * Py y))) =
    (∑ (x, y) ∈ (λx. (X x, Y x)) ‘ space M. 0)
  by (intro sum.cong) (auto simp: ae)
  then show (∑ (x, y) ∈ (λx. (X x, Y x)) ‘ space M. Pxy (x, y) * log b (Pxy (x, y) / (Px x * Py y))) = 0 by simp
qed

```

12.5 Entropy

definition (in prob-space) *entropy* :: $real \Rightarrow 'b \text{ measure} \Rightarrow ('a \Rightarrow 'b) \Rightarrow real$ **where**
 $entropy\ b\ S\ X = -\ KL\text{-divergence}\ b\ S\ (distr\ M\ S\ X)$

abbreviation (in information-space)

entropy-Pow ($\langle \mathcal{H}'(-)' \rangle$) **where**

$\mathcal{H}(X) \equiv entropy\ b\ (count\text{-space}\ (X' \text{space}\ M))\ X$

lemma (in prob-space) *distributed-RN-deriv*:

assumes X : *distributed* $M\ S\ X\ Px$

shows $AE\ x\ in\ S.$ *RN-deriv* $S\ (density\ S\ Px)\ x = Px\ x$

proof –

have *distributed* $M\ S\ X\ (RN\text{-deriv}\ S\ (density\ S\ Px))$

by (*metis* *RN-derivI* *assms* *borel-measurable-RN-deriv* *distributed-def*)

then show *?thesis*

using *assms* *distributed-unique* **by** *blast*

qed

lemma (in information-space)

fixes $X :: 'a \Rightarrow 'b$

assumes $X[\text{measurable}]$: *distributed* $M\ MX\ X\ f$ **and** nn : $\bigwedge x. x \in \text{space}\ MX \implies 0 \leq f\ x$

shows *entropy-distr*: $entropy\ b\ MX\ X = -\ (\int x. f\ x * \log\ b\ (f\ x)\ \partial MX)$ (*is ?eq*)

proof –

note $D = \text{distributed-measurable}[OF\ X]\ \text{distributed-borel-measurable}[OF\ X]$

note $ae = \text{distributed-RN-deriv}[OF\ X]$

note *distributed-real-measurable* $[OF\ nn\ X,\ \text{measurable}]$

have *ae-eq*: $AE\ x\ in\ distr\ M\ MX\ X.$ $\log\ b\ (enn2real\ (RN\text{-deriv}\ MX\ (distr\ M\ MX\ X)\ x)) = \log\ b\ (f\ x)$

unfolding *distributed-distr-eq-density* $[OF\ X]$

using $D\ ae$ **by** (*auto simp*: *AE-density*)

have *int-eq*: $(\int x. f\ x * \log\ b\ (f\ x)\ \partial MX) = (\int x. \log\ b\ (f\ x)\ \partial distr\ M\ MX\ X)$

unfolding *distributed-distr-eq-density* $[OF\ X]$

using D

by (*subst integral-density*) (*auto simp*: nn)

show *?eq*

unfolding *entropy-def* *KL-divergence-def* *entropy-density-def* *comp-def* *int-eq* *neg-equal-iff-equal*

using *ae-eq* **by** (*intro integral-cong-AE*) (*auto simp*: nn)

qed

lemma (in information-space) *entropy-le*:

fixes $Px :: 'b \Rightarrow real$ **and** $MX :: 'b \text{ measure}$

assumes $X[\text{measurable}]$: *distributed* $M\ MX\ X\ Px$ **and** $Px\text{-nn}[simp]$: $\bigwedge x. x \in \text{space}\ MX \implies 0 \leq Px\ x$

and *fin*: $\text{emeasure}\ MX\ \{x \in \text{space}\ MX. Px\ x \neq 0\} \neq \text{top}$

```

and int: integrable  $MX$  ( $\lambda x. - Px\ x * \log b\ (Px\ x)$ )
shows entropy  $b\ MX\ X \leq \log b\ (\text{measure } MX\ \{x \in \text{space } MX. Px\ x \neq 0\})$ 
proof -
  note  $Px = \text{distributed-borel-measurable}[OF\ X]$ 
  interpret  $X$ : prob-space distr  $M\ MX\ X$ 
  using distributed-measurable[ $OF\ X$ ] by (rule prob-space-distr)

  have  $-\log b\ (\text{measure } MX\ \{x \in \text{space } MX. Px\ x \neq 0\}) =$ 
     $-\log b\ (\int x. \text{indicator } \{x \in \text{space } MX. Px\ x \neq 0\}\ x\ \partial MX)$ 
    using  $Px\ Px\text{-nn}\ \text{fin}$  by (auto simp: measure-def)
  also have  $-\log b\ (\int x. \text{indicator } \{x \in \text{space } MX. Px\ x \neq 0\}\ x\ \partial MX) = -\log$ 
 $b\ (\int x. 1 / Px\ x\ \partial \text{distr } M\ MX\ X)$ 
  proof -
    have  $\text{integral}^L\ MX\ (\text{indicator } \{x \in \text{space } MX. Px\ x \neq 0\}) = LINT\ x|MX. Px$ 
 $x *_R\ (1 / Px\ x)$ 
    by (rule Bochner-Integration.integral-cong) auto
    also have  $\dots = LINT\ x|density\ MX\ (\lambda x. \text{ennreal } (Px\ x)). 1 / Px\ x$ 
    by (rule integral-density [symmetric]) (use Px Px-nn in auto)
    finally show ?thesis
    unfolding distributed-distr-eq-density[ $OF\ X$ ] by simp
  qed
  also have  $\dots \leq (\int x. -\log b\ (1 / Px\ x)\ \partial \text{distr } M\ MX\ X)$ 
  proof (rule X.jensens-inequality[of  $\lambda x. 1 / Px\ x\ \{0 <.. \}\ 0\ 1\ \lambda x. -\log b\ x$ ])
    show  $AE\ x\ \text{in}\ \text{distr } M\ MX\ X. 1 / Px\ x \in \{0 <.. \}$ 
    unfolding distributed-distr-eq-density[ $OF\ X$ ]
    using  $Px$  by (auto simp: AE-density)
    have [simp]:  $\bigwedge x. x \in \text{space } MX \implies \text{ennreal } (\text{if } Px\ x = 0 \text{ then } 0 \text{ else } 1) =$ 
 $\text{indicator } \{x \in \text{space } MX. Px\ x \neq 0\}\ x$ 
    by (auto simp: one-ennreal-def)
    have  $(\int^+ x. \text{ennreal } (- (\text{if } Px\ x = 0 \text{ then } 0 \text{ else } 1))\ \partial MX) = (\int^+ x. 0\ \partial MX)$ 
    by (intro nn-integral-cong) (auto simp: ennreal-neg)
    then show integrable (distr  $M\ MX\ X$ ) ( $\lambda x. 1 / Px\ x$ )
    unfolding distributed-distr-eq-density[ $OF\ X$ ] using  $Px$ 
    by (auto simp: nn-integral-density real-integrable-def fin ennreal-neg en-
 $\text{nnreal-mult[symmetric]}$ 
    cong: nn-integral-cong)
    have integrable  $MX\ (\lambda x. Px\ x * \log b\ (1 / Px\ x)) =$ 
integrable  $MX\ (\lambda x. - Px\ x * \log b\ (Px\ x))$ 
    using  $Px$ 
    by (intro integrable-cong-AE) (auto simp: log-divide-pos log-recip)
    then show integrable (distr  $M\ MX\ X$ ) ( $\lambda x. -\log b\ (1 / Px\ x)$ )
    unfolding distributed-distr-eq-density[ $OF\ X$ ]
    using  $Px\ \text{int}$ 
    by (subst integrable-real-density) auto
  qed (auto simp: minus-log-convex[ $OF\ b\text{-gt-1}$ ])
  also have  $\dots = (\int x. \log b\ (Px\ x)\ \partial \text{distr } M\ MX\ X)$ 
    unfolding distributed-distr-eq-density[ $OF\ X$ ] using  $Px$ 
    by (intro integral-cong-AE) (auto simp: AE-density log-divide-pos)
  also have  $\dots = -\text{entropy } b\ MX\ X$ 

```

unfolding *distributed-distr-eq-density*[OF X] **using** *Px*
by (*subst entropy-distr*[OF X]) (*auto simp: integral-density*)
finally show *?thesis*
by *simp*
qed

lemma (*in information-space*) *entropy-le-space*:
fixes *Px* :: 'b \Rightarrow real **and** *MX* :: 'b measure
assumes *X*: *distributed M MX X Px* **and** *Px-nn*[*simp*]: $\bigwedge x. x \in \text{space } MX \Rightarrow 0 \leq Px\ x$
and *fin*: *finite-measure MX*
and *int*: *integrable MX* ($\lambda x. - Px\ x * \log b (Px\ x)$)
shows *entropy b MX X* $\leq \log b (\text{measure } MX (\text{space } MX))$
proof –
note *Px* = *distributed-borel-measurable*[OF X]
interpret *finite-measure MX* **by** *fact*
have *entropy b MX X* $\leq \log b (\text{measure } MX \{x \in \text{space } MX. Px\ x \neq 0\})$
using *int X* **by** (*intro entropy-le*) *auto*
also have $\dots \leq \log b (\text{measure } MX (\text{space } MX))$
using *Px distributed-imp-emeasure-nonzero*[OF X]
by (*intro Transcendental.log-mono*)
 $(\text{auto intro!}: \text{finite-measure-mono } b\text{-gt-1 less-le}[THEN \text{iffD2}])$
 $\text{simp: emeasure-eq-measure cong: conj-cong}$
finally show *?thesis* .
qed

lemma (*in information-space*) *entropy-uniform*:
assumes *X*: *distributed M MX X* ($\lambda x. \text{indicator } A\ x / \text{measure } MX\ A$) (*is distributed - - ?f*)
shows *entropy b MX X* $= \log b (\text{measure } MX\ A)$
proof (*subst entropy-distr*[OF X])
have [*simp*]: *emeasure MX A* $\neq \infty$
using *uniform-distributed-params*[OF X] **by** (*auto simp add: measure-def*)
have *eq*: $(\int x. \text{indicator } A\ x / \text{measure } MX\ A * \log b (\text{indicator } A\ x / \text{measure } MX\ A) \partial MX) =$
 $(\int x. (- \log b (\text{measure } MX\ A) / \text{measure } MX\ A) * \text{indicator } A\ x \partial MX)$
using *uniform-distributed-params*[OF X]
by (*intro Bochner-Integration.integral-cong*) (*auto split: split-indicator simp: log-divide-pos zero-less-measure-iff*)
show – $(\int x. \text{indicator } A\ x / \text{measure } MX\ A * \log b (\text{indicator } A\ x / \text{measure } MX\ A) \partial MX) =$
 $\log b (\text{measure } MX\ A)$
unfolding *eq* **using** *uniform-distributed-params*[OF X]
by (*subst Bochner-Integration.integral-mult-right*) (*auto simp: measure-def less-top[symmetric]*)
intro!: *integrable-real-indicator*
qed *simp*

lemma (*in information-space*) *entropy-simple-distributed*:
simple-distributed M X f $\Rightarrow \mathcal{H}(X) = - (\sum x \in X \text{space } M. f\ x * \log b (f\ x))$

by (subst entropy-distr[OF simple-distributed])
 (auto simp add: lebesgue-integral-count-space-finite)

lemma (in information-space) entropy-le-card-not-0:
 assumes X : simple-distributed M X f
 shows $\mathcal{H}(X) \leq \log b$ (card (X ‘ space $M \cap \{x. f\ x \neq 0\}$))
proof –
 let $?X = \text{count-space } (X \text{ ‘ space } M)$
 have $\mathcal{H}(X) \leq \log b$ (measure $?X$ $\{x \in \text{space } ?X. f\ x \neq 0\}$)
 by (rule entropy-le[OF simple-distributed[OF X]])
 (insert X , auto simp: simple-distributed-finite[OF X] subset-eq integrable-count-space
 emeasure-count-space)
 also have measure $?X$ $\{x \in \text{space } ?X. f\ x \neq 0\} = \text{card } (X \text{ ‘ space } M \cap \{x. f\ x$
 $\neq 0\})$
 by (simp-all add: simple-distributed-finite[OF X] subset-eq emeasure-count-space
 measure-def Int-def)
 finally show ?thesis .
qed

lemma (in information-space) entropy-le-card:
 assumes X : simple-distributed M X f
 shows $\mathcal{H}(X) \leq \log b$ (real (card (X ‘ space M)))
proof –
 let $?X = \text{count-space } (X \text{ ‘ space } M)$
 have $\mathcal{H}(X) \leq \log b$ (measure $?X$ (space $?X$))
 by (rule entropy-le-space[OF simple-distributed[OF X]])
 (insert X , auto simp: simple-distributed-finite[OF X] subset-eq integrable-count-space
 emeasure-count-space finite-measure-count-space)
 also have measure $?X$ (space $?X$) = card (X ‘ space M)
 by (simp-all add: simple-distributed-finite[OF X] subset-eq emeasure-count-space
 measure-def)
 finally show ?thesis .
qed

12.6 Conditional Mutual Information

definition (in prob-space)
 conditional-mutual-information b MX MY MZ X Y $Z \equiv$
 mutual-information b MX ($MY \otimes_M MZ$) X ($\lambda x. (Y\ x, Z\ x)$) –
 mutual-information b MX MZ X Z

abbreviation (in information-space)
 conditional-mutual-information-Pow ($\langle \mathcal{I}'(-; - | -) \rangle$) **where**
 $\mathcal{I}(X ; Y | Z) \equiv \text{conditional-mutual-information } b$
 (count-space (X ‘ space M)) (count-space (Y ‘ space M)) (count-space (Z ‘ space
 M)) X Y Z

lemma (in information-space)
 assumes S : sigma-finite-measure S and T : sigma-finite-measure T and P :

sigma-finite-measure P

```

assumes Px[measurable]: distributed M S X Px
and Px-nn[simp]:  $\bigwedge x. x \in \text{space } S \implies 0 \leq Px\ x$ 
assumes Pz[measurable]: distributed M P Z Pz
and Pz-nn[simp]:  $\bigwedge z. z \in \text{space } P \implies 0 \leq Pz\ z$ 
assumes Pyz[measurable]: distributed M (T  $\otimes_M$  P) ( $\lambda x. (Y\ x, Z\ x)$ ) Pyz
and Pyz-nn[simp]:  $\bigwedge y\ z. y \in \text{space } T \implies z \in \text{space } P \implies 0 \leq Pyz\ (y, z)$ 
assumes Pxz[measurable]: distributed M (S  $\otimes_M$  P) ( $\lambda x. (X\ x, Z\ x)$ ) Pxz
and Pxz-nn[simp]:  $\bigwedge x\ z. x \in \text{space } S \implies z \in \text{space } P \implies 0 \leq Pxz\ (x, z)$ 
assumes Pxyz[measurable]: distributed M (S  $\otimes_M$  T  $\otimes_M$  P) ( $\lambda x. (X\ x, Y\ x, Z\ x)$ ) Pxyz
and Pxyz-nn[simp]:  $\bigwedge x\ y\ z. x \in \text{space } S \implies y \in \text{space } T \implies z \in \text{space } P \implies 0 \leq Pxyz\ (x, y, z)$ 
assumes I1: integrable (S  $\otimes_M$  T  $\otimes_M$  P) ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxyz\ (x, y, z) / (Px\ x * Pyz\ (y, z)))$ )
assumes I2: integrable (S  $\otimes_M$  T  $\otimes_M$  P) ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxz\ (x, z) / (Px\ x * Pz\ z))$ )
shows conditional-mutual-information-generic-eq: conditional-mutual-information
b S T P X Y Z
= ( $\int (x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxyz\ (x, y, z) / (Pxz\ (x, z) * (Pyz\ (y, z) / Pz\ z)))\ \partial(S \otimes_M T \otimes_M P)$ ) (is ?eq)
and conditional-mutual-information-generic-nonneg:  $0 \leq \text{conditional-mutual-information}$ 
b S T P X Y Z (is ?nonneg)
proof -
have [measurable]:  $Px \in S \rightarrow_M \text{borel}$ 
using Px Px-nn by (intro distributed-real-measurable)
have [measurable]:  $Pz \in P \rightarrow_M \text{borel}$ 
using Pz Pz-nn by (intro distributed-real-measurable)
have measurable-Pyz[measurable]:  $Pyz \in (T \otimes_M P) \rightarrow_M \text{borel}$ 
using Pyz Pyz-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)
have measurable-Pxz[measurable]:  $Pxz \in (S \otimes_M P) \rightarrow_M \text{borel}$ 
using Pxz Pxz-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)
have measurable-Pxyz[measurable]:  $Pxyz \in (S \otimes_M T \otimes_M P) \rightarrow_M \text{borel}$ 
using Pxyz Pxyz-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret P: sigma-finite-measure P by fact
interpret TP: pair-sigma-finite T P ..
interpret SP: pair-sigma-finite S P ..
interpret ST: pair-sigma-finite S T ..
interpret SPT: pair-sigma-finite S  $\otimes_M$  P T ..
interpret STP: pair-sigma-finite S T  $\otimes_M$  P ..
interpret TPS: pair-sigma-finite T  $\otimes_M$  P S ..
have TP: sigma-finite-measure (T  $\otimes_M$  P) ..
have SP: sigma-finite-measure (S  $\otimes_M$  P) ..
have YZ: random-variable (T  $\otimes_M$  P) ( $\lambda x. (Y\ x, Z\ x)$ )
using Pyz by (simp add: distributed-measurable)

```

from $Pxz\ Pxyz$ **have** *distr-eq*: $distr\ M\ (S \otimes_M P)\ (\lambda x. (X\ x, Z\ x)) =$
 $distr\ (distr\ M\ (S \otimes_M T \otimes_M P)\ (\lambda x. (X\ x, Y\ x, Z\ x)))\ (S \otimes_M P)\ (\lambda(x, y,$
 $z). (x, z))$
by (*simp add: comp-def distr-distr*)

have *mutual-information* $b\ S\ P\ X\ Z =$
 $(\int x. Pxz\ x * \log b\ (Pxz\ x / (Px\ (fst\ x) * Pz\ (snd\ x))))\ \partial(S \otimes_M P)$
by (*rule mutual-information-distr[OF S P Px Px-nn Pz Pz-nn Pxz Pxz-nn]*)
also have $\dots = (\int (x,y,z). Pxyz\ (x,y,z) * \log b\ (Pxz\ (x,z) / (Px\ x * Pz\ z))\ \partial(S$
 $\otimes_M T \otimes_M P))$
using *b-gt-1 Pxz Px Pz*
by (*subst distributed-transform-integral[OF Pxyz - Pxz -, where T= $\lambda(x, y, z).$*
 $(x, z)]$)
(auto simp: split-beta' space-pair-measure)

finally have *mi-eq*:
 $mutual-information\ b\ S\ P\ X\ Z = (\int (x,y,z). Pxyz\ (x,y,z) * \log b\ (Pxz\ (x,z) /$
 $(Px\ x * Pz\ z))\ \partial(S \otimes_M T \otimes_M P))$.

have *ae1*: $AE\ x\ in\ S \otimes_M T \otimes_M P. Px\ (fst\ x) = 0 \longrightarrow Pxyz\ x = 0$
by (*intro subdensity-real[of fst, OF - Pxyz Px]*) (*auto simp: space-pair-measure*)
moreover have *ae2*: $AE\ x\ in\ S \otimes_M T \otimes_M P. Pz\ (snd\ (snd\ x)) = 0 \longrightarrow Pxyz$
 $x = 0$
by (*intro subdensity-real[of $\lambda x. snd\ (snd\ x)$, OF - Pxyz Pz]*) (*auto simp:*
space-pair-measure)
moreover have *ae3*: $AE\ x\ in\ S \otimes_M T \otimes_M P. Pxz\ (fst\ x, snd\ (snd\ x)) = 0$
 $\longrightarrow Pxyz\ x = 0$
by (*intro subdensity-real[of $\lambda x. (fst\ x, snd\ (snd\ x))$, OF - Pxyz Pxz]*) (*auto*
simp: space-pair-measure)
moreover have *ae4*: $AE\ x\ in\ S \otimes_M T \otimes_M P. Pyz\ (snd\ x) = 0 \longrightarrow Pxyz\ x$
 $= 0$
by (*intro subdensity-real[of snd, OF - Pxyz Pyz]*) (*auto simp: space-pair-measure*)
ultimately have *ae*: $AE\ x\ in\ S \otimes_M T \otimes_M P.$
 $Pxyz\ x * \log b\ (Pxyz\ x / (Px\ (fst\ x) * Pyz\ (snd\ x))) -$
 $Pxyz\ x * \log b\ (Pxz\ (fst\ x, snd\ (snd\ x)) / (Px\ (fst\ x) * Pz\ (snd\ (snd\ x)))) =$
 $Pxyz\ x * \log b\ (Pxyz\ x * Pz\ (snd\ (snd\ x)) / (Pxz\ (fst\ x, snd\ (snd\ x)) * Pyz$
 $(snd\ x)))$
using *AE-space*
proof *eventually-elim*
case (*elim x*)
show *?case*
proof *cases*
assume $Pxyz\ x \neq 0$
with *elim* **have** $0 < Px\ (fst\ x)\ 0 < Pz\ (snd\ (snd\ x))\ 0 < Pxz\ (fst\ x, snd$
 $(snd\ x))$
 $0 < Pyz\ (snd\ x)\ 0 < Pxyz\ x$
by (*auto simp: space-pair-measure less-le*)
then show *?thesis*
using *b-gt-1* **by** (*simp add: log-simps less-imp-le field-simps*)
qed *simp*

```

qed
with I1 I2 show ?eq
  unfolding conditional-mutual-information-def
  apply (subst mi-eq)
  apply (subst mutual-information-distr[OF S TP Px Px-nn Pyz - Pxyz])
  apply (auto simp: space-pair-measure)
  apply (subst Bochner-Integration.integral-diff[symmetric])
  apply (auto intro!: integral-cong-AE simp: split-beta' simp del: Bochner-Integration.integral-diff)
done

let ?P = density (S  $\otimes_M$  T  $\otimes_M$  P) Pxyz
interpret P: prob-space ?P
  unfolding distributed-distr-eq-density[OF Pxyz, symmetric]
  by (rule prob-space-distr) simp

let ?Q = density (T  $\otimes_M$  P) Pyz
interpret Q: prob-space ?Q
  unfolding distributed-distr-eq-density[OF Pyz, symmetric]
  by (rule prob-space-distr) simp

let ?f =  $\lambda(x, y, z). Pxz(x, z) * (Pyz(y, z) / Pz z) / Pxyz(x, y, z)$ 

from subdensity-real[of snd, OF - Pyz Pz - AE-I2 AE-I2]
have aeX1: AE x in T  $\otimes_M$  P. Pz (snd x) = 0  $\longrightarrow$  Pyz x = 0
  by (auto simp: comp-def space-pair-measure)
have aeX2: AE x in T  $\otimes_M$  P. 0  $\leq$  Pz (snd x)
  using Pz by (intro TP.AE-pair-measure) (auto simp: comp-def)

have aeX3: AE y in T  $\otimes_M$  P. ( $\int^+ x. ennreal (Pxz(x, snd y)) \partial S$ ) = ennreal
(Pz (snd y))
  using Pz distributed-marginal-eq-joint2[OF P S Pz Pxz]
  by (intro TP.AE-pair-measure) auto

have ( $\int^+ x. ?f x \partial ?P$ )  $\leq$  ( $\int^+ (x, y, z). Pxz(x, z) * (Pyz(y, z) / Pz z) \partial (S$ 
 $\otimes_M T \otimes_M P)$ )
  by (subst nn-integral-density)
  (auto intro!: nn-integral-mono simp: space-pair-measure ennreal-mult[symmetric])
also have ... = ( $\int^+ (y, z). (\int^+ x. ennreal (Pxz(x, z)) * ennreal (Pyz(y, z) /$ 
 $Pz z) \partial S) \partial (T \otimes_M P)$ )
  by (subst STP.nn-integral-snd[symmetric])
  (auto simp add: split-beta' ennreal-mult[symmetric] space-pair-measure intro!:
nn-integral-cong)
also have ... = ( $\int^+ x. ennreal (Pyz x) * 1 \partial T \otimes_M P$ )
proof -
  have D: ( $\int^+ x. ennreal (Pxz(x, b)) * ennreal (Pyz(a, b) / Pz b) \partial S$ ) =
ennreal (Pyz(a, b))
  if Pz b = 0  $\longrightarrow$  Pyz(a, b) = 0 a  $\in$  space T  $\wedge$  b  $\in$  space P
  ( $\int^+ x. ennreal (Pxz(x, b)) \partial S$ ) = ennreal (Pz b)
  for a b

```



```

    using that by (subst nn-integral-multc) (auto split: prod.split simp: en-
nreal-mult[symmetric])
    show ?thesis
    apply (rule nn-integral-cong-AE)
    using aeX1 aeX3
    by (force simp add: space-pair-measure D)
qed
also have ... = 1
    using Q.emeasure-space-1 distributed-distr-eq-density[OF Pyz]
    by (subst nn-integral-density[symmetric]) auto
finally have le1:  $(\int^+ x. ?f x \partial ?P) \leq 1$  .
also have ... <  $\infty$  by simp
finally have fin:  $(\int^+ x. ?f x \partial ?P) \neq \infty$  by simp

have  $(\int^+ x. \text{ennreal } (Pxyz x) * \text{ennreal } (Pxz (\text{fst } x, \text{snd } (\text{snd } x)) * Pyz (\text{snd } x) / (Pz (\text{snd } (\text{snd } x)) * Pxyz x)) \partial (S \otimes_M T \otimes_M P)) \neq 0$ 
proof
    let ?g =  $\lambda x. \text{ennreal } (Pxyz x) * (Pxz (\text{fst } x, \text{snd } (\text{snd } x)) * Pyz (\text{snd } x) / (Pz (\text{snd } (\text{snd } x)) * Pxyz x))$ 
    assume  $(\int^+ x. ?g x \partial (S \otimes_M T \otimes_M P)) = 0$ 
    then have AE x in  $S \otimes_M T \otimes_M P. ?g x = 0$ 
        by (intro nn-integral-0-iff-AE[THEN iffD1]) auto
    then have AE x in  $S \otimes_M T \otimes_M P. Pxyz x = 0$ 
        using ae2 ae3 ae4
    by (auto split: if-split-asm simp: mult-le-0-iff divide-le-0-iff space-pair-measure)
    then have  $(\int^+ x. \text{ennreal } (Pxyz x) \partial S \otimes_M T \otimes_M P) = 0$ 
        by (subst nn-integral-cong-AE[of -  $\lambda x. 0$ ]) auto
    with P.emeasure-space-1 show False
        by (subst (asm) emeasure-density) (auto cong: nn-integral-cong)
qed
then have pos:  $(\int^+ x. ?f x \partial ?P) \neq 0$ 
    by (subst nn-integral-density) (simp-all add: split-beta')

have neg:  $(\int^+ x. - ?f x \partial ?P) = 0$ 
    by (subst nn-integral-0-iff-AE) (auto simp: space-pair-measure ennreal-neg)

have I3: integrable  $(S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))))$ 
    apply (rule integrable-cong-AE[THEN iffD1, OF - - - Bochner-Integration.integrable-diff[OF I1 I2]])
    using ae
    apply (auto simp: split-beta')
    done

have  $-\log b \ 1 \leq -\log b (\text{integral}^L ?P ?f)$ 
proof (intro le-imp-neg-le Transcendental.log-mono[OF b-gt-1])
    have If: integrable ?P ?f

```

```

    unfolding real-integrable-def
  proof (intro conjI)
    from neg show  $(\int^+ x. - ?f x \partial ?P) \neq \infty$ 
      by simp
    from fin show  $(\int^+ x. ?f x \partial ?P) \neq \infty$ 
      by simp
  qed simp
  then have  $(\int^+ x. ?f x \partial ?P) = (\int x. ?f x \partial ?P)$ 
    apply (rule nn-integral-eq-integral)
    apply (auto simp: space-pair-measure ennreal-neg)
    done
  with pos le1
  show  $0 < (\int x. ?f x \partial ?P) (\int x. ?f x \partial ?P) \leq 1$ 
    by (simp-all add: one-ennreal.rep-eq zero-less-iff-neq-zero[symmetric])
  qed
  also have  $-\log b (\text{integral}^L ?P ?f) \leq (\int x. -\log b (?f x) \partial ?P)$ 
  proof (rule P.jensens-inequality[where a=0 and b=1 and I={0<..}])
    show  $AE x \text{ in } ?P. ?f x \in \{0<..\}$ 
      unfolding AE-density[OF distributed-borel-measurable[OF Pxyz]]
      using ae1 ae2 ae3 ae4
      by (auto simp: space-pair-measure less-le)
    show integrable ?P ?f
      unfolding real-integrable-def
      using fin neg by (auto simp: split-beta')
    have integrable  $(S \otimes_M T \otimes_M P) (\lambda x. Pxyz x * -\log b (?f x))$ 
      apply (rule integrable-cong-AE[THEN iffD1, OF - - - I3])
      by (auto simp: log-mult log-divide field-simps)
    then
    show integrable ?P  $(\lambda x. -\log b (?f x))$ 
      by (subst integrable-real-density) (auto simp: space-pair-measure)
  qed (auto simp: b-gt-1 minus-log-convex)
  also have  $\dots = \text{conditional-mutual-information } b S T P X Y Z$ 
    unfolding ‹?eq›
    apply (subst integral-real-density)
    apply simp
    apply (force simp: space-pair-measure)
    apply simp
    apply (intro integral-cong-AE)
    by (auto simp: field-simps log-divide)
  finally show ?nonneg
    by simp
  qed

lemma (in information-space)
  fixes Px ::  $\text{real}$ 
  assumes S: sigma-finite-measure S and T: sigma-finite-measure T and P:
    sigma-finite-measure P
  assumes Fx: finite-entropy S X Px
  assumes Fz: finite-entropy P Z Pz

```

assumes Fyz : *finite-entropy* $(T \otimes_M P) (\lambda x. (Y x, Z x)) P yz$
assumes Fxz : *finite-entropy* $(S \otimes_M P) (\lambda x. (X x, Z x)) P xz$
assumes $Fxyz$: *finite-entropy* $(S \otimes_M T \otimes_M P) (\lambda x. (X x, Y x, Z x)) P xyz$
shows *conditional-mutual-information-generic-eq'*: *conditional-mutual-information*
 $b S T P X Y Z$
 $= (\int (x, y, z). Pxyz (x, y, z) * \log b (Pxyz (x, y, z) / (Pxz (x, z) * (Pyz (y, z) / Pz z))) \partial(S \otimes_M T \otimes_M P))$ (**is** *?eq*)
and *conditional-mutual-information-generic-nonneg'*: $0 \leq$ *conditional-mutual-information*
 $b S T P X Y Z$ (**is** *?nonneg*)
proof –
note $Px = Fx[THEN \text{finite-entropy-distributed}, \text{measurable}]$
note $Pz = Fz[THEN \text{finite-entropy-distributed}, \text{measurable}]$
note $Pyz = Fyz[THEN \text{finite-entropy-distributed}, \text{measurable}]$
note $Pxz = Fxz[THEN \text{finite-entropy-distributed}, \text{measurable}]$
note $Pxyz = Fxyz[THEN \text{finite-entropy-distributed}, \text{measurable}]$

note $Px\text{-nn} = Fx[THEN \text{finite-entropy-nn}]$
note $Pz\text{-nn} = Fz[THEN \text{finite-entropy-nn}]$
note $Pyz\text{-nn} = Fyz[THEN \text{finite-entropy-nn}]$
note $Pxz\text{-nn} = Fxz[THEN \text{finite-entropy-nn}]$
note $Pxyz\text{-nn} = Fxyz[THEN \text{finite-entropy-nn}]$

note $Px' = Fx[THEN \text{finite-entropy-measurable}, \text{measurable}]$
note $Pz' = Fz[THEN \text{finite-entropy-measurable}, \text{measurable}]$
note $Pyz' = Fyz[THEN \text{finite-entropy-measurable}, \text{measurable}]$
note $Pxz' = Fxz[THEN \text{finite-entropy-measurable}, \text{measurable}]$
note $Pxyz' = Fxyz[THEN \text{finite-entropy-measurable}, \text{measurable}]$

interpret S : *sigma-finite-measure* S **by fact**
interpret T : *sigma-finite-measure* T **by fact**
interpret P : *sigma-finite-measure* P **by fact**
interpret TP : *pair-sigma-finite* $T P$..
interpret SP : *pair-sigma-finite* $S P$..
interpret ST : *pair-sigma-finite* $S T$..
interpret SPT : *pair-sigma-finite* $S \otimes_M P T$..
interpret STP : *pair-sigma-finite* $S T \otimes_M P$..
interpret TPS : *pair-sigma-finite* $T \otimes_M P S$..
have TP : *sigma-finite-measure* $(T \otimes_M P)$..
have SP : *sigma-finite-measure* $(S \otimes_M P)$..

from $Pxz Pxyz$ **have** *distr-eq*: $distr M (S \otimes_M P) (\lambda x. (X x, Z x)) =$
 $distr (distr M (S \otimes_M T \otimes_M P) (\lambda x. (X x, Y x, Z x))) (S \otimes_M P) (\lambda(x, y,$
 $z). (x, z))$
by (*simp add: distr-distr comp-def*)

have *mutual-information* $b S P X Z =$
 $(\int x. Pxz x * \log b (Pxz x / (Px (fst x) * Pz (snd x))) \partial(S \otimes_M P))$
using $Px Px\text{-nn} Pz Pz\text{-nn} Pxz Pxz\text{-nn}$
by (*rule mutual-information-distr[OF S P]*) (*auto simp: space-pair-measure*)

also have $\dots = (\int (x,y,z). Pxyz (x,y,z) * \log b (Pxz (x,z) / (Px x * Pz z)) \partial(S \otimes_M T \otimes_M P))$
using *b-gt-1 Pxz Pxz-nn Pxyz Pxyz-nn*
by (*subst distributed-transform-integral[OF Pxyz - Pxz, where $T=\lambda(x, y, z).$*
(x, z)])
(auto simp: split-beta')
finally have *mi-eq:*
*mutual-information b S P X Z = (\int (x,y,z). Pxyz (x,y,z) * \log b (Pxz (x,z) /*
*(Px x * Pz z)) \partial(S \otimes_M T \otimes_M P)) .*

have *ae1: AE x in S \otimes_M T \otimes_M P. Px (fst x) = 0 \longrightarrow Pxyz x = 0*
by (*intro subdensity-finite-entropy[of fst, OF - Fxyz Fx] auto*)
moreover have *ae2: AE x in S \otimes_M T \otimes_M P. Pz (snd (snd x)) = 0 \longrightarrow Pxyz*
x = 0
by (*intro subdensity-finite-entropy[of \lambda x. snd (snd x), OF - Fxyz Fz] auto*)
moreover have *ae3: AE x in S \otimes_M T \otimes_M P. Pxz (fst x, snd (snd x)) = 0*
\longrightarrow Pxyz x = 0
by (*intro subdensity-finite-entropy[of \lambda x. (fst x, snd (snd x)), OF - Fxyz Fxz]*
auto)
moreover have *ae4: AE x in S \otimes_M T \otimes_M P. Pyz (snd x) = 0 \longrightarrow Pxyz x*
= 0
by (*intro subdensity-finite-entropy[of snd, OF - Fxyz Fyz] auto*)
ultimately have *ae: AE x in S \otimes_M T \otimes_M P.*
*Pxyz x * \log b (Pxyz x / (Px (fst x) * Pyz (snd x))) -*
*Pxyz x * \log b (Pxz (fst x, snd (snd x)) / (Px (fst x) * Pz (snd (snd x)))) =*
*Pxyz x * \log b (Pxyz x * Pz (snd (snd x)) / (Pxz (fst x, snd (snd x)) * Pyz*
(snd x)))
using *AE-space*
proof *eventually-elim*
case (*elim x*)
show *?case*
proof *cases*
assume *Pxyz x \neq 0*
with *elim have* $0 < Px (fst x) \ 0 < Pz (snd (snd x)) \ 0 < Pxz (fst x, snd$
 $(snd x))$
 $0 < Pyz (snd x) \ 0 < Pxyz x$
using *Px-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn*
by (*auto simp: space-pair-measure less-le*)
then show *?thesis*
using *b-gt-1 by (simp add: log-simps less-imp-le field-simps)*
qed *simp*
qed

have *integrable (S \otimes_M T \otimes_M P)*
 $(\lambda x. Pxyz x * \log b (Pxyz x) - Pxyz x * \log b (Px (fst x)) - Pxyz x * \log b (Pyz$
 $(snd x)))$
using *finite-entropy-integrable[OF Fxyz]*
using *finite-entropy-integrable-transform[OF Fx Pxyz Pxyz-nn, of fst]*
using *finite-entropy-integrable-transform[OF Fyz Pxyz Pxyz-nn, of snd]*

```

by simp
moreover have ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxyz\ (x, y, z) / (Px\ x * Pyz\ (y, z))) \in \text{borel-measurable}\ (S \otimes_M T \otimes_M P)$ )
using  $Pxyz\ Px\ Pyz$  by simp
ultimately have  $I1: \text{integrable}\ (S \otimes_M T \otimes_M P)\ (\lambda(x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxyz\ (x, y, z) / (Px\ x * Pyz\ (y, z))))$ 
apply (rule integrable-cong-AE-imp)
using ae1 ae4  $Px\text{-nn}\ Pyz\text{-nn}\ Pxyz\text{-nn}$ 
by (auto simp: log-divide log-mult field-simps)
have  $\text{integrable}\ (S \otimes_M T \otimes_M P)\ (\lambda x. Pxyz\ x * \log b\ (Pxz\ (fst\ x, snd\ (snd\ x))) - Pxyz\ x * \log b\ (Px\ (fst\ x) - Pxyz\ x * \log b\ (Pz\ (snd\ (snd\ x))))$ 
using finite-entropy-integrable-transform[OF  $Fxz\ Pxyz\ Pxyz\text{-nn}$ , of  $\lambda x. (fst\ x, snd\ (snd\ x))$ ]
using finite-entropy-integrable-transform[OF  $Fx\ Pxyz\ Pxyz\text{-nn}$ , of  $fst$ ]
using finite-entropy-integrable-transform[OF  $Fz\ Pxyz\ Pxyz\text{-nn}$ , of  $snd \circ snd$ ]
by simp
moreover have ( $\lambda(x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxz\ (x, z) / (Px\ x * Pz\ z))) \in \text{borel-measurable}\ (S \otimes_M T \otimes_M P)$ 
using  $Pxyz\ Px\ Pz$ 
by auto
ultimately have  $I2: \text{integrable}\ (S \otimes_M T \otimes_M P)\ (\lambda(x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxz\ (x, z) / (Px\ x * Pz\ z)))$ 
apply (rule integrable-cong-AE-imp)
using ae1 ae2 ae3 ae4  $Px\text{-nn}\ Pz\text{-nn}\ Pxz\text{-nn}\ Pyz\text{-nn}\ Pxyz\text{-nn}$ 
apply (auto simp: field-simps log-divide log-mult)
done
from ae I1 I2 show ?eq
unfolding conditional-mutual-information-def mi-eq
apply (subst mutual-information-distr[OF  $S\ TP\ Px\ Px\text{-nn}\ Pyz\ Pyz\text{-nn}\ Pxyz\ Pxyz\text{-nn}$ ]; simp add: space-pair-measure)
apply (subst Bochner-Integration.integral-diff[symmetric])
apply (auto intro!: integral-cong-AE simp: split-beta' simp del: Bochner-Integration.integral-diff)
done

let ?P = density  $(S \otimes_M T \otimes_M P)\ Pxyz$ 
interpret P: prob-space ?P
unfolding distributed-distr-eq-density[OF  $Pxyz$ , symmetric] by (rule prob-space-distr)
simp

let ?Q = density  $(T \otimes_M P)\ Pyz$ 
interpret Q: prob-space ?Q
unfolding distributed-distr-eq-density[OF  $Pyz$ , symmetric] by (rule prob-space-distr)
simp

let ?f =  $\lambda(x, y, z). Pxz\ (x, z) * (Pyz\ (y, z) / Pz\ z) / Pxyz\ (x, y, z)$ 

from subdensity-finite-entropy[of  $snd$ , OF -  $Fyz\ Fz$ ]
have aeX1: AE  $x$  in  $T \otimes_M P. Pz\ (snd\ x) = 0 \longrightarrow Pyz\ x = 0$  by (auto simp:

```

```

comp-def)
  have aeX2: AE x in T  $\otimes_M$  P. 0  $\leq$  Pz (snd x)
    using Pz by (intro TP.AE-pair-measure) (auto intro: Pz-nn)

  have aeX3: AE y in T  $\otimes_M$  P. ( $\int^+$  x. ennreal (Pzx (x, snd y))  $\partial S$ ) = ennreal
(Pz (snd y))
    using Pz distributed-marginal-eq-joint2[OF P S Pz Pzx]
    by (intro TP.AE-pair-measure) (auto )
  have ( $\int^+$  x.  $\int$  x  $\partial ?P$ )  $\leq$  ( $\int^+$  (x, y, z). Pzx (x, z) * (Pyz (y, z) / Pz z)  $\partial(S$ 
 $\otimes_M T \otimes_M P)$ )
    using Px-nn Pz-nn Pzx-nn Pyz-nn Pxyz-nn
    by (subst nn-integral-density)
    (auto intro!: nn-integral-mono simp: space-pair-measure ennreal-mult[symmetric])
  also have ... = ( $\int^+$  (y, z).  $\int^+$  x. ennreal (Pzx (x, z)) * ennreal (Pyz (y, z) /
Pz z)  $\partial S \partial T \otimes_M P$ )
    using Px-nn Pz-nn Pzx-nn Pyz-nn Pxyz-nn
    by (subst STP.nn-integral-snd[symmetric])
    (auto simp add: split-beta' ennreal-mult[symmetric] space-pair-measure intro!:
nn-integral-cong)
  also have ... = ( $\int^+$  x. ennreal (Pyz x) * 1  $\partial T \otimes_M P$ )
  proof -
    have *: ( $\int^+$  x. ennreal (Pzx (x, b)) * ennreal (Pyz (a, b) / Pz b)  $\partial S$ ) = ennreal
(Pyz (a, b))
      if Pz b = 0  $\longrightarrow$  Pyz (a, b) = 0 0  $\leq$  Pz b a  $\in$  space T  $\wedge$  b  $\in$  space P
      ( $\int^+$  x. ennreal (Pzx (x, b))  $\partial S$ ) = ennreal (Pz b) for a b
      using Pyz-nn[of (a,b)] that
    by (subst nn-integral-multc) (auto simp: space-pair-measure ennreal-mult[symmetric])
  show ?thesis
    using aeX1 aeX2 aeX3 AE-space
    by (force simp: * space-pair-measure intro: nn-integral-cong-AE)
qed
also have ... = 1
  using Q.emmeasure-space-1 Pyz-nn distributed-distr-eq-density[OF Pyz]
  by (subst nn-integral-density[symmetric]) auto
finally have le1: ( $\int^+$  x.  $\int$  x  $\partial ?P$ )  $\leq$  1 .
also have ...  $< \infty$  by simp
finally have fin: ( $\int^+$  x.  $\int$  x  $\partial ?P$ )  $\neq \infty$  by simp

have ( $\int^+$  x.  $\int$  x  $\partial ?P$ )  $\neq 0$ 
  using Pxyz-nn
  apply (subst nn-integral-density)
  apply (simp-all add: split-beta' ennreal-mult'[symmetric] cong: nn-integral-cong)
proof
  let ?g =  $\lambda x$ . ennreal (if Pxyz x = 0 then 0 else Pzx (fst x, snd (snd x)) * Pyz
(snd x) / Pz (snd (snd x)))
  assume ( $\int^+$  x. ?g x  $\partial(S \otimes_M T \otimes_M P)$ ) = 0
  then have AE x in S  $\otimes_M T \otimes_M P$ . ?g x = 0
    by (intro nn-integral-0-iff-AE[THEN iffD1]) auto
  then have AE x in S  $\otimes_M T \otimes_M P$ . Pxyz x = 0

```

```

using ae1 ae2 ae3 ae4
by (insert Px-nn Pz-nn Pxz-nn Pyz-nn,
    auto split: if-split-asm simp: mult-le-0-iff divide-le-0-iff space-pair-measure)
then have  $(\int^+ x. \text{ennreal } (Pxyz\ x) \partial S \otimes_M T \otimes_M P) = 0$ 
    by (subst nn-integral-cong-AE[of -  $\lambda x. 0$ ]) auto
with P.emeasure-space-1 show False
    by (subst (asm) emeasure-density) (auto cong: nn-integral-cong)
qed
then have pos:  $0 < (\int^+ x. ?f\ x\ \partial ?P)$ 
    by (simp add: zero-less-iff-neq-zero)

have neg:  $(\int^+ x. -\ ?f\ x\ \partial ?P) = 0$ 
    using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
    by (intro nn-integral-0-iff-AE[THEN iffD2])
        (auto simp: split-beta' AE-density space-pair-measure intro!: AE-I2 en-
nreal-neg)

have I3: integrable  $(S \otimes_M T \otimes_M P) (\lambda(x, y, z). Pxyz\ (x, y, z) * \log b\ (Pxyz\ (x, y, z) / (Pxz\ (x, z) * (Pyz\ (y, z) / Pz\ z))))$ 
apply (rule integrable-cong-AE[THEN iffD1, OF - - - Bochner-Integration.integrable-diff[OF
I1 I2]])
    using ae
    by (auto simp: split-beta')

have  $-\log b\ 1 \leq -\log b\ (\text{integral}^L\ ?P\ ?f)$ 
proof (intro le-imp-neg-le Transcendental.log-mono[OF b-gt-1])
    have If: integrable  $?P\ ?f$ 
        using neg fin by (force simp add: real-integrable-def)
    then have  $(\int^+ x. ?f\ x\ \partial ?P) = (\int x. ?f\ x\ \partial ?P)$ 
        using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
        by (intro nn-integral-eq-integral)
        (auto simp: AE-density space-pair-measure)
    with pos le1
    show  $0 < (\int x. ?f\ x\ \partial ?P) (\int x. ?f\ x\ \partial ?P) \leq 1$ 
        by (simp-all add: )
qed
also have  $-\log b\ (\text{integral}^L\ ?P\ ?f) \leq (\int x. -\log b\ (?f\ x)\ \partial ?P)$ 
proof (rule P.jensens-inequality[where a=0 and b=1 and I={0<..}])
    show AE x in ?P.  $?f\ x \in \{0 < ..\}$ 
        unfolding AE-density[OF distributed-borel-measurable[OF Pxyz]]
        using ae1 ae2 ae3 ae4
        by (insert Pxyz-nn Pyz-nn Pz-nn Pxz-nn, auto simp: space-pair-measure
less-le)
    show integrable  $?P\ ?f$ 
        unfolding real-integrable-def
        using fin neg by (auto simp: split-beta')
have integrable  $(S \otimes_M T \otimes_M P) (\lambda x. Pxyz\ x * -\log b\ (?f\ x))$ 
apply (rule integrable-cong-AE[THEN iffD1, OF - - - I3])
    using Pz-nn Pxz-nn Pyz-nn Pxyz-nn ae2 ae3 ae4

```

```

    by (auto simp: log-divide field-simps)
  then show integrable ?P ( $\lambda x. - \log b \ (?f \ x)$ )
    using Pxyz-nn by (auto simp: integrable-real-density)
qed (auto simp: b-gt-1 minus-log-convex)
also have ... = conditional-mutual-information b S T P X Y Z
  unfolding <?eq>
  using Pz-nn Pxz-nn Pyz-nn Pxyz-nn
  apply (subst integral-real-density)
  apply simp
  apply simp
  apply simp
  apply (intro integral-cong-AE)
  using ae1 ae2 ae3 ae4
  by (auto simp: log-divide zero-less-mult-iff field-simps)
finally show ?nonneg
  by simp
qed

```

lemma (in information-space) conditional-mutual-information-eq:

```

  assumes Pz: simple-distributed M Z Pz
  assumes Pyz: simple-distributed M ( $\lambda x. (Y \ x, Z \ x)$ ) Pyz
  assumes Pxz: simple-distributed M ( $\lambda x. (X \ x, Z \ x)$ ) Pxz
  assumes Pxyz: simple-distributed M ( $\lambda x. (X \ x, Y \ x, Z \ x)$ ) Pxyz
  shows  $\mathcal{I}(X ; Y \mid Z) =$ 
    ( $\sum (x, y, z) \in (\lambda x. (X \ x, Y \ x, Z \ x))' \text{space } M. Pxyz \ (x, y, z) * \log b \ (Pxyz \ (x, y, z) / (Pxz \ (x, z) * (Pyz \ (y, z) / Pz \ z)))$ )
proof (subst conditional-mutual-information-generic-eq[OF - - - -
  simple-distributed[OF Pz] - simple-distributed-joint[OF Pyz] - simple-distributed-joint[OF
Pxz] -
  simple-distributed-joint2[OF Pxyz]])
  note simple-distributed-joint2-finite[OF Pxyz, simp]
  show sigma-finite-measure (count-space (X ‘ space M))
    by (simp add: sigma-finite-measure-count-space-finite)
  show sigma-finite-measure (count-space (Y ‘ space M))
    by (simp add: sigma-finite-measure-count-space-finite)
  show sigma-finite-measure (count-space (Z ‘ space M))
    by (simp add: sigma-finite-measure-count-space-finite)
  have count-space (X ‘ space M)  $\otimes_M$  count-space (Y ‘ space M)  $\otimes_M$  count-space
(Z ‘ space M) =
    count-space (X‘space M  $\times$  Y‘space M  $\times$  Z‘space M)
    (is ?P = ?C)
    by (simp add: pair-measure-count-space)

  let ?Px =  $\lambda x. \text{measure } M \ (X - \{x\} \cap \text{space } M)$ 
  have ( $\lambda x. (X \ x, Z \ x) \in \text{measurable } M \ (\text{count-space } (X \text{ ‘ space } M) \otimes_M \text{count-space } (Z \text{ ‘ space } M))$ )
    using simple-distributed-joint[OF Pxz] by (rule distributed-measurable)
  from measurable-comp[OF this measurable-fst]
  have random-variable (count-space (X ‘ space M)) X

```



```

  by (simp add: comp-def)
then have simple-function M X
  unfolding simple-function-def by (auto simp: measurable-count-space-eq2)
then have simple-distributed M X ?Px
  by (rule simple-distributedI) (auto simp: measure-nonneg)
then show distributed M (count-space (X ‘ space M)) X ?Px
  by (rule simple-distributed)

let ?f = (λx. if x ∈ (λx. (X x, Y x, Z x)) ‘ space M then Pxyz x else 0)
let ?g = (λx. if x ∈ (λx. (Y x, Z x)) ‘ space M then Pyz x else 0)
let ?h = (λx. if x ∈ (λx. (X x, Z x)) ‘ space M then Prx x else 0)
show
  integrable ?P (λ(x, y, z). ?f (x, y, z) * log b (?f (x, y, z) / (?Px x * ?g (y,
z))))
  integrable ?P (λ(x, y, z). ?f (x, y, z) * log b (?h (x, z) / (?Px x * Pz z)))
  by (auto intro!: integrable-count-space simp: pair-measure-count-space)
  let ?i = λx y z. ?f (x, y, z) * log b (?f (x, y, z) / (?h (x, z) * (?g (y, z) / Pz
z)))
  let ?j = λx y z. Pxyz (x, y, z) * log b (Pxyz (x, y, z) / (Prx (x, z) * (Pyz (y, z)
/ Pz z)))
  have (λ(x, y, z). ?i x y z) = (λx. if x ∈ (λx. (X x, Y x, Z x)) ‘ space M then ?j
(fst x) (fst (snd x)) (snd (snd x)) else 0)
  by (auto intro!: ext)
  then show (∫ (x, y, z). ?i x y z ∂?P) = (∑ (x, y, z) ∈ (λx. (X x, Y x, Z x)) ‘
space M. ?j x y z)
  by (auto intro!: sum.cong simp add: ⟨?P = ?C⟩ lebesgue-integral-count-space-finite
simple-distributed-finite sum.If-cases split-beta)
qed (insert Pz Pyz Prx Pxyz, auto intro: measure-nonneg)

```

lemma (in information-space) conditional-mutual-information-nonneg:

assumes X : simple-function M X **and** Y : simple-function M Y **and** Z : simple-function M Z

shows $0 \leq \mathcal{I}(X ; Y \mid Z)$

proof –

have $[simp]: \text{count-space } (X \text{ ‘ space } M) \otimes_M \text{count-space } (Y \text{ ‘ space } M) \otimes_M \text{count-space } (Z \text{ ‘ space } M) =$

$\text{count-space } (X \text{ ‘ space } M \times Y \text{ ‘ space } M \times Z \text{ ‘ space } M)$

by (simp add: pair-measure-count-space X Y Z simple-functionD)

note $sf = \text{sigma-finite-measure-count-space-finite}[OF \text{ simple-functionD}(1)]$

note $sd = \text{simple-distributedI}[OF - - refl]$

note $sp = \text{simple-function-Pair}$

show ?thesis

apply (rule conditional-mutual-information-generic-nonneg[OF $sf[OF X]$ $sf[OF Y]$ $sf[OF Z]$])

apply (force intro: simple-distributed[OF $sd[OF X]$])

apply simp

apply (force intro: simple-distributed[OF $sd[OF Z]$])

apply simp

apply (force intro: simple-distributed-joint[OF $sd[OF sp[OF Y Z]]$])

```

    apply simp
    apply (force intro: simple-distributed-joint[OF sd[OF sp[OF X Z]]])
    apply simp
    apply (fastforce intro: simple-distributed-joint2[OF sd[OF sp[OF X sp[OF
Y Z]]]])
    apply (auto intro!: integrable-count-space simp: X Y Z simple-functionD)
  done
qed

```

12.7 Conditional Entropy

definition (in *prob-space*)

conditional-entropy $b \ S \ T \ X \ Y = - \left(\int (x, y). \log b \ (enn2real \ (RN\text{-}deriv \ (S \otimes_M T) \ (distr \ M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x))) \ (x, y)) \ / \ enn2real \ (RN\text{-}deriv \ T \ (distr \ M \ T \ Y) \ y)) \ \partial \ distr \ M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x)) \right)$

abbreviation (in *information-space*)

conditional-entropy-Pow $(\langle \mathcal{H}'(- \mid -') \rangle)$ **where**
 $\mathcal{H}(X \mid Y) \equiv conditional\text{-}entropy \ b \ (count\text{-}space \ (X' \ space \ M)) \ (count\text{-}space \ (Y' \ space \ M)) \ X \ Y$

lemma (in *information-space*) *conditional-entropy-generic-eq*:

fixes $Pxy :: - \Rightarrow real$ **and** $Py :: 'c \Rightarrow real$
assumes S : *sigma-finite-measure* S **and** T : *sigma-finite-measure* T
assumes $Py[measurable]$: *distributed* $M \ T \ Y \ Py$ **and** $Py\text{-}nn[simp]$: $\bigwedge x. x \in space \ T \Longrightarrow 0 \leq Py \ x$

assumes $Pxy[measurable]$: *distributed* $M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x)) \ Pxy$
and $Pxy\text{-}nn[simp]$: $\bigwedge x \ y. x \in space \ S \Longrightarrow y \in space \ T \Longrightarrow 0 \leq Pxy \ (x, y)$
shows *conditional-entropy* $b \ S \ T \ X \ Y = - \left(\int (x, y). Pxy \ (x, y) * \log b \ (Pxy \ (x, y) \ / \ Py \ y) \ \partial (S \otimes_M T) \right)$

proof –

interpret S : *sigma-finite-measure* S **by fact**
interpret T : *sigma-finite-measure* T **by fact**
interpret ST : *pair-sigma-finite* $S \ T$..

have $[measurable]$: $Py \in T \rightarrow_M \text{borel}$
using $Py \ Py\text{-}nn$ **by** (*intro distributed-real-measurable*)
have *measurable-Pxy* $[measurable]$: $Pxy \in (S \otimes_M T) \rightarrow_M \text{borel}$
using $Pxy \ Py\text{-}nn$ **by** (*intro distributed-real-measurable*) (*auto simp: space-pair-measure*)

have $AE \ x \ in \ density \ (S \otimes_M T) \ (\lambda x. ennreal \ (Pxy \ x)). Pxy \ x = enn2real \ (RN\text{-}deriv \ (S \otimes_M T) \ (distr \ M \ (S \otimes_M T) \ (\lambda x. (X \ x, Y \ x))) \ x)$

unfolding *AE-density* $[OF \ distributed\text{-}borel\text{-}measurable, \ OF \ Pxy]$

unfolding *distributed-distr-eq-density* $[OF \ Pxy]$

using *distributed-RN-deriv* $[OF \ Pxy]$

by auto

moreover

have $AE \ x \ in \ density \ (S \otimes_M T) \ (\lambda x. ennreal \ (Pxy \ x)). Py \ (snd \ x) = enn2real$

```

(RN-deriv T (distr M T Y) (snd x))
  unfolding AE-density[OF distributed-borel-measurable, OF Pxy]
  unfolding distributed-distr-eq-density[OF Py]
  using distributed-RN-deriv[OF Py]
  by (force intro: ST.AE-pair-measure)
ultimately
  have conditional-entropy b S T X Y = - (∫ x. Pxy x * log b (Pxy x / Py (snd
x)) ∂(S ⊗M T))
  unfolding conditional-entropy-def neg-equal-iff-equal
  apply (subst integral-real-density[symmetric])
  apply (auto simp: distributed-distr-eq-density[OF Pxy] space-pair-measure
intro!: integral-cong-AE)
done
then show ?thesis by (simp add: split-beta)
qed

```

lemma (in information-space) conditional-entropy-eq-entropy:

```

fixes Px :: 'b ⇒ real and Py :: 'c ⇒ real
assumes S: sigma-finite-measure S and T: sigma-finite-measure T
assumes Py[measurable]: distributed M T Y Py
  and Py-nn[simp]: ∧x. x ∈ space T ⇒ 0 ≤ Py x
assumes Pxy[measurable]: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
  and Pxy-nn[simp]: ∧x y. x ∈ space S ⇒ y ∈ space T ⇒ 0 ≤ Pxy (x, y)
assumes I1: integrable (S ⊗M T) (λx. Pxy x * log b (Pxy x))
assumes I2: integrable (S ⊗M T) (λx. Pxy x * log b (Py (snd x)))
shows conditional-entropy b S T X Y = entropy b (S ⊗M T) (λx. (X x, Y x))
– entropy b T Y

```

proof –

```

interpret S: sigma-finite-measure S by fact
interpret T: sigma-finite-measure T by fact
interpret ST: pair-sigma-finite S T ..

```

```

have [measurable]: Py ∈ T →M borel
  using Py Py-nn by (intro distributed-real-measurable)
have measurable-Pxy[measurable]: Pxy ∈ (S ⊗M T) →M borel
  using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

```

```

have entropy b T Y = - (∫ y. Py y * log b (Py y) ∂T)
  by (rule entropy-distr[OF Py Py-nn])
also have ... = - (∫ (x,y). Pxy (x,y) * log b (Py y) ∂(S ⊗M T))
  using b-gt-1
  by (subst distributed-transform-integral[OF Pxy - Py, where T=snd])
  (auto intro!: Bochner-Integration.integral-cong simp: space-pair-measure)
finally have e-eq: entropy b T Y = - (∫ (x,y). Pxy (x,y) * log b (Py y) ∂(S
⊗M T)) .

```

```

have **: ∧x. x ∈ space (S ⊗M T) ⇒ 0 ≤ Pxy x
  by (auto simp: space-pair-measure)

```

have $ae2: AE\ x\ in\ S \otimes_M T. Py\ (snd\ x) = 0 \longrightarrow Pxy\ x = 0$
by (*intro subdensity-real[of snd, OF - Pxy Py]*)
(auto intro: measurable-Pair simp: space-pair-measure)
moreover have $ae4: AE\ x\ in\ S \otimes_M T. 0 \leq Py\ (snd\ x)$
using Py **by** (*intro ST.AE-pair-measure*) (*auto simp: comp-def intro!: measurable-snd''*)
ultimately have $AE\ x\ in\ S \otimes_M T. 0 \leq Pxy\ x \wedge 0 \leq Py\ (snd\ x) \wedge$
 $(Pxy\ x = 0 \vee (Pxy\ x \neq 0 \longrightarrow 0 < Pxy\ x \wedge 0 < Py\ (snd\ x)))$
by (*auto simp: space-pair-measure less-le*)
then have $ae: AE\ x\ in\ S \otimes_M T.$
 $Pxy\ x * \log b\ (Pxy\ x) - Pxy\ x * \log b\ (Py\ (snd\ x)) = Pxy\ x * \log b\ (Pxy\ x /$
 $Py\ (snd\ x))$
by (*auto simp: log-simps field-simps b-gt-1*)
have *conditional-entropy b S T X Y =*
 $-(\int x. Pxy\ x * \log b\ (Pxy\ x) - Pxy\ x * \log b\ (Py\ (snd\ x))\ \partial(S \otimes_M T))$
unfolding *conditional-entropy-generic-eq[OF S T Py Py-nn Pxy Pxy-nn, simplified]* *neg-equal-iff-equal*
using ae **by** (*force intro: integral-cong-AE*)
also have $\dots = -(\int x. Pxy\ x * \log b\ (Pxy\ x)\ \partial(S \otimes_M T)) - -(\int x. Pxy\ x$
 $* \log b\ (Py\ (snd\ x))\ \partial(S \otimes_M T))$
by (*simp add: Bochner-Integration.integral-diff[OF I1 I2]*)
finally show *?thesis*
using *conditional-entropy-generic-eq[OF S T Py Py-nn Pxy Pxy-nn, simplified]*
*entropy-distr[OF Pxy **, simplified]* *e-eq*
by (*simp add: split-beta'*)
qed

lemma (in information-space) conditional-entropy-eq-entropy-simple:
assumes $X: \text{simple-function } M\ X$ **and** $Y: \text{simple-function } M\ Y$
shows $\mathcal{H}(X \mid Y) = \text{entropy } b\ (\text{count-space } (X' \text{space } M) \otimes_M \text{count-space } (Y' \text{space } M))\ (\lambda x. (X\ x, Y\ x)) - \mathcal{H}(Y)$
proof –
have $\text{count-space } (X' \text{space } M) \otimes_M \text{count-space } (Y' \text{space } M) = \text{count-space } (X' \text{space } M \times Y' \text{space } M)$
(is ?P = ?C) **using** $X\ Y$ **by** (*simp add: simple-functionD pair-measure-count-space*)
show *?thesis*
by (*rule conditional-entropy-eq-entropy sigma-finite-measure-count-space-finite*
simple-functionD X Y simple-distributed simple-distributedI[OF - - refl]
simple-distributed-joint simple-function-Pair integrable-count-space
measure-nonneg)
(auto simp: <?P = ?C> measure-nonneg intro!: integrable-count-space simple-functionD X Y)
qed

lemma (in information-space) conditional-entropy-eq:
assumes $Y: \text{simple-distributed } M\ Y\ Py$
assumes $XY: \text{simple-distributed } M\ (\lambda x. (X\ x, Y\ x))\ Pxy$
shows $\mathcal{H}(X \mid Y) = -(\sum (x, y) \in (\lambda x. (X\ x, Y\ x))' \text{space } M. Pxy\ (x, y) * \log b\ (Pxy\ (x, y) / Py\ y))$

```

proof (subst conditional-entropy-generic-eq[OF - -
  simple-distributed[OF Y] - simple-distributed-joint[OF XY]])
  have finite (( $\lambda x. (X\ x, Y\ x)$ )'space M)
    using XY unfolding simple-distributed-def by auto
  from finite-imageI[OF this, of fst]
  have [simp]: finite (X'space M)
    by (simp add: image-comp comp-def)
  note Y[THEN simple-distributed-finite, simp]
  show sigma-finite-measure (count-space (X 'space M))
    by (simp add: sigma-finite-measure-count-space-finite)
  show sigma-finite-measure (count-space (Y 'space M))
    by (simp add: sigma-finite-measure-count-space-finite)
  let ?f = ( $\lambda x. \text{if } x \in (\lambda x. (X\ x, Y\ x))\ 'space\ M \text{ then } Pxy\ x \text{ else } 0$ )
  have count-space (X 'space M)  $\otimes_M$  count-space (Y 'space M) = count-space
    (X'space M  $\times$  Y'space M)
    (is ?P = ?C)
    using Y by (simp add: simple-distributed-finite pair-measure-count-space)
  have eq: ( $\lambda(x, y). ?f\ (x, y) * \log\ b\ (?f\ (x, y) / Py\ y)$ ) =
    ( $\lambda x. \text{if } x \in (\lambda x. (X\ x, Y\ x))\ 'space\ M \text{ then } Pxy\ x * \log\ b\ (Pxy\ x / Py\ (snd\ x))$ 
    else 0)
    by auto
  from Y show - ( $\int (x, y). ?f\ (x, y) * \log\ b\ (?f\ (x, y) / Py\ y)\ \partial ?P$ ) =
    - ( $\sum (x, y) \in (\lambda x. (X\ x, Y\ x))\ 'space\ M. Pxy\ (x, y) * \log\ b\ (Pxy\ (x, y) / Py\ y)$ )
    by (auto intro!: sum.cong simp add: (P = ?C) lebesgue-integral-count-space-finite
    simple-distributed-finite eq sum.If-cases split-beta)
qed (use Y XY in auto)

```

lemma (in information-space) conditional-mutual-information-eq-conditional-entropy:

assumes X: simple-function M X **and** Y: simple-function M Y

shows $\mathcal{I}(X ; X \mid Y) = \mathcal{H}(X \mid Y)$

proof –

define Py **where** $Py\ x = (\text{if } x \in Y'space\ M \text{ then measure } M\ (Y - \{x\} \cap space\ M) \text{ else } 0)$ **for** x

define Pxy **where** $Pxy\ x =$

(if $x \in (\lambda x. (X\ x, Y\ x))'space\ M$ then measure M (($\lambda x. (X\ x, Y\ x)$) - {x} \cap space M) else 0)

for x

define Pxyx **where** $Pxyx\ x =$

(if $x \in (\lambda x. (X\ x, X\ x, Y\ x))'space\ M$ then measure M (($\lambda x. (X\ x, X\ x, Y\ x)$) - {x} \cap space M) else 0)

for x

let ?M = X'space M \times X'space M \times Y'space M

note XY = simple-function-Pair[OF X Y]

note XXY = simple-function-Pair[OF X XY]

have Py: simple-distributed M Y Py

using Y **by** (rule simple-distributedI) (auto simp: Py-def measure-nonneg)

```

have Pxy: simple-distributed M (λx. (X x, Y x)) Pxy
  using XY by (rule simple-distributedI) (auto simp: Pxy-def measure-nonneg)
have Pxy: simple-distributed M (λx. (X x, X x, Y x)) Pxy
  using XXY by (rule simple-distributedI) (auto simp: Pxy-def measure-nonneg)
have eq: (λx. (X x, X x, Y x)) ‘ space M = (λ(x, y). (x, x, y)) ‘ (λx. (X x, Y
x)) ‘ space M
  by auto
have inj: ∧A. inj-on (λ(x, y). (x, x, y)) A
  by (auto simp: inj-on-def)
have Pxy-eq: ∧x y. Pxy (x, x, y) = Pxy (x, y)
  by (auto simp: Pxy-def Pxy-def intro!: arg-cong[where f=prob])
have AE x in count-space ((λx. (X x, Y x))‘space M). Py (snd x) = 0 → Pxy
x = 0
  using Py Pxy
  by (intro subdensity-real[of snd, OF - Pxy[THEN simple-distributed] Py[THEN
simple-distributed]])
  (auto intro: measurable-Pair simp: AE-count-space)
then show ?thesis
  apply (subst conditional-mutual-information-eq[OF Py Pxy Pxy Pxy])
  apply (subst conditional-entropy-eq[OF Py Pxy])
  apply (auto intro!: sum.cong simp: Pxy-eq sum-negf[symmetric] eq sum.reindex[OF
inj]
log-simps zero-less-mult-iff zero-le-mult-iff field-simps mult-less-0-iff
AE-count-space)
done
qed

```

lemma (in information-space) conditional-entropy-nonneg:

assumes X : simple-function M X **and** Y : simple-function M Y **shows** $0 \leq \mathcal{H}(X \mid Y)$

using conditional-mutual-information-eq-conditional-entropy[OF X Y] conditional-mutual-information-nonneg X X Y]

by simp

12.8 Equalities

lemma (in information-space) mutual-information-eq-entropy-conditional-entropy-distr:

fixes $Px :: 'b \Rightarrow \text{real}$ **and** $Py :: 'c \Rightarrow \text{real}$ **and** $Pxy :: ('b \times 'c) \Rightarrow \text{real}$

assumes S : sigma-finite-measure S **and** T : sigma-finite-measure T

assumes Px [measurable]: distributed M S X Px

and Px -nn[simp]: $\bigwedge x. x \in \text{space } S \implies 0 \leq Px \ x$

and Py [measurable]: distributed M T Y Py

and Py -nn[simp]: $\bigwedge x. x \in \text{space } T \implies 0 \leq Py \ x$

and Pxy [measurable]: distributed M $(S \otimes_M T)$ $(\lambda x. (X \ x, Y \ x))$ Pxy

and Pxy -nn[simp]: $\bigwedge x \ y. x \in \text{space } S \implies y \in \text{space } T \implies 0 \leq Pxy \ (x, y)$

assumes Ix : integrable($S \otimes_M T$) $(\lambda x. Pxy \ x * \log b \ (Px \ (fst \ x)))$

assumes Iy : integrable($S \otimes_M T$) $(\lambda x. Pxy \ x * \log b \ (Py \ (snd \ x)))$

assumes Ixy : integrable($S \otimes_M T$) $(\lambda x. Pxy \ x * \log b \ (Pxy \ x))$

shows mutual-information b S T X Y = entropy b S X + entropy b T Y −

```

entropy b (S  $\otimes_M$  T) ( $\lambda x. (X\ x, Y\ x)$ )
proof -
  have [measurable]: Px  $\in S \rightarrow_M$  borel
    using Px Px-nn by (intro distributed-real-measurable)
  have [measurable]: Py  $\in T \rightarrow_M$  borel
    using Py Py-nn by (intro distributed-real-measurable)
  have measurable-Pxy[measurable]: Pxy  $\in (S \otimes_M T) \rightarrow_M$  borel
    using Pxy Pxy-nn by (intro distributed-real-measurable) (auto simp: space-pair-measure)

  have X: entropy b S X = - ( $\int x. Pxy\ x * \log b\ (Px\ (fst\ x))\ \partial(S \otimes_M T)$ )
    using b-gt-1
    apply (subst entropy-distr[OF Px Px-nn], simp)
    apply (subst distributed-transform-integral[OF Pxy - Px, where T=fst])
    apply (auto intro!: integral-cong simp: space-pair-measure)
    done

  have Y: entropy b T Y = - ( $\int x. Pxy\ x * \log b\ (Py\ (snd\ x))\ \partial(S \otimes_M T)$ )
    using b-gt-1
    apply (subst entropy-distr[OF Py Py-nn], simp)
    apply (subst distributed-transform-integral[OF Pxy - Py, where T=snd])
    apply (auto intro!: integral-cong simp: space-pair-measure)
    done

  interpret S: sigma-finite-measure S by fact
  interpret T: sigma-finite-measure T by fact
  interpret ST: pair-sigma-finite S T ..
  have ST: sigma-finite-measure (S  $\otimes_M$  T) ..

  have XY: entropy b (S  $\otimes_M$  T) ( $\lambda x. (X\ x, Y\ x)$ ) = - ( $\int x. Pxy\ x * \log b\ (Pxy\ x)\ \partial(S \otimes_M T)$ )
    by (subst entropy-distr[OF Pxy]) (auto intro!: integral-cong simp: space-pair-measure)

  have AE x in S  $\otimes_M$  T. Px (fst x) = 0  $\longrightarrow$  Pxy x = 0
    by (intro subdensity-real[of fst, OF - Pxy Px]) (auto intro: measurable-Pair simp: space-pair-measure)
  moreover have AE x in S  $\otimes_M$  T. Py (snd x) = 0  $\longrightarrow$  Pxy x = 0
    by (intro subdensity-real[of snd, OF - Pxy Py]) (auto intro: measurable-Pair simp: space-pair-measure)
  moreover have AE x in S  $\otimes_M$  T. 0  $\leq$  Px (fst x)
    using Px by (intro ST.AE-pair-measure) (auto simp: comp-def intro!: measurable-fst'')
  moreover have AE x in S  $\otimes_M$  T. 0  $\leq$  Py (snd x)
    using Py by (intro ST.AE-pair-measure) (auto simp: comp-def intro!: measurable-snd'')
  ultimately have AE x in S  $\otimes_M$  T. Pxy x * log b (Pxy x) - Pxy x * log b (Px (fst x)) - Pxy x * log b (Py (snd x)) =
    Pxy x * log b (Pxy x / (Px (fst x) * Py (snd x)))
    (is AE x in -. ?f x = ?g x)
    using AE-space

```

```

proof eventually-elim
  case (elim x)
  show ?case
proof cases
  assume Pxy x ≠ 0
  with elim have 0 < Px (fst x) 0 < Py (snd x) 0 < Pxy x
    by (auto simp: space-pair-measure less-le)
  then show ?thesis
    using b-gt-1 by (simp add: log-simps less-imp-le field-simps)
qed simp
qed

have entropy b S X + entropy b T Y - entropy b (S ⊗M T) (λx. (X x, Y x))
= integralL (S ⊗M T) ?f
  unfolding X Y XY
  apply (subst Bochner-Integration.integral-diff)
  apply (intro Bochner-Integration.integrable-diff Ixy Ix Iy)+
  apply (subst Bochner-Integration.integral-diff)
  apply (intro Ixy Ix Iy)+
  apply (simp add: field-simps)
done
also have ... = integralL (S ⊗M T) ?g
  using ⟨AE x in -. ?f x = ?g x⟩ by (intro integral-cong-AE) auto
also have ... = mutual-information b S T X Y
  by (rule mutual-information-distr[OF S T Px - Py - Pxy -, symmetric])
  (auto simp: space-pair-measure)
finally show ?thesis ..
qed

lemma (in information-space) mutual-information-eq-entropy-conditional-entropy':
fixes Px :: 'b ⇒ real and Py :: 'c ⇒ real and Pxy :: ('b × 'c) ⇒ real
assumes S: sigma-finite-measure S and T: sigma-finite-measure T
assumes Px: distributed M S X Px ∧ x. x ∈ space S ⇒ 0 ≤ Px x
  and Py: distributed M T Y Py ∧ x. x ∈ space T ⇒ 0 ≤ Py x
assumes Pxy: distributed M (S ⊗M T) (λx. (X x, Y x)) Pxy
  ∧ x. x ∈ space (S ⊗M T) ⇒ 0 ≤ Pxy x
assumes Ix: integrable(S ⊗M T) (λx. Pxy x * log b (Px (fst x)))
assumes Iy: integrable(S ⊗M T) (λx. Pxy x * log b (Py (snd x)))
assumes Ixy: integrable(S ⊗M T) (λx. Pxy x * log b (Pxy x))
shows mutual-information b S T X Y = entropy b S X - conditional-entropy b
S T X Y
using
  mutual-information-eq-entropy-conditional-entropy-distr[OF S T Px Py Pxy Ix
Iy Ixy]
  conditional-entropy-eq-entropy[OF S T Py Pxy Ixy Iy]
by (simp add: space-pair-measure)

lemma (in information-space) mutual-information-eq-entropy-conditional-entropy:
assumes sf-X: simple-function M X and sf-Y: simple-function M Y

```


shows $\mathcal{I}(X ; Y) = \mathcal{H}(X) - \mathcal{H}(X \mid Y)$

proof –

have X : *simple-distributed* M X ($\lambda x. \text{measure } M (X - \{x\} \cap \text{space } M)$)

using $\text{sf-}X$ **by** (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

have Y : *simple-distributed* M Y ($\lambda x. \text{measure } M (Y - \{x\} \cap \text{space } M)$)

using $\text{sf-}Y$ **by** (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

have $\text{sf-}XY$: *simple-function* M ($\lambda x. (X\ x, Y\ x)$)

using $\text{sf-}X$ $\text{sf-}Y$ **by** (*rule simple-function-Pair*)

then have XY : *simple-distributed* M ($\lambda x. (X\ x, Y\ x)$) ($\lambda x. \text{measure } M ((\lambda x. (X\ x, Y\ x)) - \{x\} \cap \text{space } M)$)

by (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

from *simple-distributed-joint-finite*[*OF this, simp*]

have eq : $\text{count-space } (X \text{ 'space } M) \otimes_M \text{count-space } (Y \text{ 'space } M) = \text{count-space } (X \text{ 'space } M \times Y \text{ 'space } M)$

by (*simp add: pair-measure-count-space*)

have $\mathcal{I}(X ; Y) = \mathcal{H}(X) + \mathcal{H}(Y) - \text{entropy } b (\text{count-space } (X \text{ 'space } M) \otimes_M \text{count-space } (Y \text{ 'space } M))$ ($\lambda x. (X\ x, Y\ x)$)

using *sigma-finite-measure-count-space-finite*

sigma-finite-measure-count-space-finite

simple-distributed[*OF X*] *measure-nonneg* *simple-distributed*[*OF Y*] *measure-nonneg* *simple-distributed-joint*[*OF XY*]

by (*rule mutual-information-eq-entropy-conditional-entropy-distr*)

(*auto simp: eq-integrable-count-space measure-nonneg*)

then show *?thesis*

unfolding *conditional-entropy-eq-entropy-simple*[*OF sf-X sf-Y*] **by** *simp*

qed

lemma (*in information-space*) *mutual-information-nonneg-simple*:

assumes $\text{sf-}X$: *simple-function* M X **and** $\text{sf-}Y$: *simple-function* M Y

shows $0 \leq \mathcal{I}(X ; Y)$

proof –

have X : *simple-distributed* M X ($\lambda x. \text{measure } M (X - \{x\} \cap \text{space } M)$)

using $\text{sf-}X$ **by** (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

have Y : *simple-distributed* M Y ($\lambda x. \text{measure } M (Y - \{x\} \cap \text{space } M)$)

using $\text{sf-}Y$ **by** (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

have $\text{sf-}XY$: *simple-function* M ($\lambda x. (X\ x, Y\ x)$)

using $\text{sf-}X$ $\text{sf-}Y$ **by** (*rule simple-function-Pair*)

then have XY : *simple-distributed* M ($\lambda x. (X\ x, Y\ x)$) ($\lambda x. \text{measure } M ((\lambda x. (X\ x, Y\ x)) - \{x\} \cap \text{space } M)$)

by (*rule simple-distributedI*) (*auto simp: measure-nonneg*)

from *simple-distributed-joint-finite*[*OF this, simp*]

have eq : $\text{count-space } (X \text{ 'space } M) \otimes_M \text{count-space } (Y \text{ 'space } M) = \text{count-space } (X \text{ 'space } M \times Y \text{ 'space } M)$

by (*simp add: pair-measure-count-space*)

show *?thesis*

by (rule mutual-information-nonneg[OF - - simple-distributed[OF X] - simple-distributed[OF Y] - simple-distributed-joint[OF XY]])
 (simp-all add: eq integrable-count-space sigma-finite-measure-count-space-finite measure-nonneg)
 qed

lemma (in information-space) conditional-entropy-less-eq-entropy:
 assumes X: simple-function M X and Z: simple-function M Z
 shows $\mathcal{H}(X \mid Z) \leq \mathcal{H}(X)$
 proof -
 have $0 \leq \mathcal{I}(X \mid Z)$ using X Z by (rule mutual-information-nonneg-simple)
 also have $\mathcal{I}(X \mid Z) = \mathcal{H}(X) - \mathcal{H}(X \mid Z)$ using mutual-information-eq-entropy-conditional-entropy[OF assms].
 finally show ?thesis by auto
 qed

lemma (in information-space)
 fixes Px :: 'b \Rightarrow real and Py :: 'c \Rightarrow real and Pxy :: ('b \times 'c) \Rightarrow real
 assumes S: sigma-finite-measure S and T: sigma-finite-measure T
 assumes Px: finite-entropy S X Px and Py: finite-entropy T Y Py
 assumes Pxy: finite-entropy (S \otimes_M T) ($\lambda x. (X x, Y x)$) Pxy
 shows conditional-entropy b S T X Y \leq entropy b S X
 proof -
 have $0 \leq \text{mutual-information } b \ S \ T \ X \ Y$
 by (rule mutual-information-nonneg') fact+
 also have $\dots = \text{entropy } b \ S \ X - \text{conditional-entropy } b \ S \ T \ X \ Y$
 proof (intro mutual-information-eq-entropy-conditional-entropy')
 show integrable (S \otimes_M T) ($\lambda x. Pxy \ x * \log b \ (Px \ (fst \ x))$)
 integrable (S \otimes_M T) ($\lambda x. Pxy \ x * \log b \ (Py \ (snd \ x))$)
 integrable (S \otimes_M T) ($\lambda x. Pxy \ x * \log b \ (Pxy \ x)$)
 using assms
 by (auto intro!: finite-entropy-integrable finite-entropy-distributed
 finite-entropy-integrable-transform[OF Px] finite-entropy-integrable-transform[OF Py])
 intro: finite-entropy-nn
 qed (use assms Px Py Pxy finite-entropy-nn finite-entropy-distributed in auto)
 finally show ?thesis by auto
 qed

lemma (in information-space) entropy-chain-rule:
 assumes X: simple-function M X and Y: simple-function M Y
 shows $\mathcal{H}(\lambda x. (X \ x, Y \ x)) = \mathcal{H}(X) + \mathcal{H}(Y \mid X)$
 proof -
 note XY = simple-distributedI[OF simple-function-Pair[OF X Y] measure-nonneg refl]
 note YX = simple-distributedI[OF simple-function-Pair[OF Y X] measure-nonneg refl]
 note simple-distributed-joint-finite[OF this, simp]
 let ?f = $\lambda x. \text{prob } ((\lambda x. (X \ x, Y \ x)) - ' \{x\} \cap \text{space } M)$

```

let ?g = λx. prob ((λx. (Y x, X x)) - ‘ {x} ∩ space M)
let ?h = λx. if x ∈ (λx. (Y x, X x)) ‘ space M then prob ((λx. (Y x, X x)) - ‘
{x} ∩ space M) else 0
have  $\mathcal{H}(\lambda x. (X x, Y x)) = - (\sum x \in (\lambda x. (X x, Y x)) \text{ ‘ space } M. ?f x * \log b (?f x))$ 
using XY by (rule entropy-simple-distributed)
also have ... = - ( $\sum x \in (\lambda(x, y). (y, x)) \text{ ‘ } (\lambda x. (X x, Y x)) \text{ ‘ space } M. ?g x * \log b (?g x)$ )
by (subst (2) sum.reindex) (auto simp: inj-on-def intro!: sum.cong arg-cong[where
f=λA. prob A * log b (prob A)])
also have ... = - ( $\sum x \in (\lambda x. (Y x, X x)) \text{ ‘ space } M. ?h x * \log b (?h x)$ )
by (auto intro!: sum.cong)
also have ... = entropy b (count-space (Y ‘ space M)  $\otimes_M$  count-space (X ‘
space M)) (λx. (Y x, X x))
by (subst entropy-distr[OF simple-distributed-joint[OF YX]])
(auto simp: pair-measure-count-space sigma-finite-measure-count-space-finite
lebesgue-integral-count-space-finite
cong del: sum.cong-simp intro!: sum.mono-neutral-left measure-nonneg)
finally have  $\mathcal{H}(\lambda x. (X x, Y x)) = \text{entropy } b \text{ (count-space (Y ‘ space M) } \otimes_M \text{ count-space (X ‘ space M)) (λx. (Y x, X x)) .}$ 
then show ?thesis
unfolding conditional-entropy-eq-entropy-simple[OF Y X] by simp
qed

```

lemma (in information-space) entropy-partition:

```

assumes X: simple-function M X
shows  $\mathcal{H}(X) = \mathcal{H}(f \circ X) + \mathcal{H}(X|f \circ X)$ 
proof -
note fX = simple-function-compose[OF X, of f]
have eq:  $(\lambda x. ((f \circ X) x, X x)) \text{ ‘ space } M = (\lambda x. (f x, x)) \text{ ‘ } X \text{ ‘ space } M$  by auto
have inj:  $\bigwedge A. \text{inj-on } (\lambda x. (f x, x)) A$ 
by (auto simp: inj-on-def)

have  $\mathcal{H}(X) = - (\sum x \in X \text{ ‘ space } M. \text{prob } (X - \text{ ‘ } \{x\} \cap \text{space } M) * \log b (\text{prob } (X - \text{ ‘ } \{x\} \cap \text{space } M)))$ 
by (simp add: entropy-simple-distributed[OF simple-distributedI[OF X measure-nonneg refl]])
also have ... = - ( $\sum x \in (\lambda x. ((f \circ X) x, X x)) \text{ ‘ space } M. \text{prob } ((\lambda x. ((f \circ X) x, X x)) - \text{ ‘ } \{x\} \cap \text{space } M) * \log b (\text{prob } ((\lambda x. ((f \circ X) x, X x)) - \text{ ‘ } \{x\} \cap \text{space } M)))$ )
unfolding eq
apply (subst sum.reindex[OF inj])
apply (auto intro!: sum.cong arg-cong[where f=λA. prob A * log b (prob A)])
done
also have ... =  $\mathcal{H}(\lambda x. ((f \circ X) x, X x))$ 
using entropy-simple-distributed[OF simple-distributedI[OF simple-function-Pair[OF fX X] measure-nonneg refl]]
by fastforce
also have ... =  $\mathcal{H}(f \circ X) + \mathcal{H}(X|f \circ X)$ 

```

using X entropy-chain-rule by blast
 finally show ?thesis .
 qed

corollary (in information-space) entropy-data-processing:
 assumes simple-function M X shows $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$
 by (smt (verit) assms conditional-entropy-nonneg entropy-partition simple-function-compose)

corollary (in information-space) entropy-of-inj:
 assumes X : simple-function M X and inj: inj-on f (X 'space M)
 shows $\mathcal{H}(f \circ X) = \mathcal{H}(X)$
 proof (rule antisym)
 show $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$ using entropy-data-processing[OF X] .
 next
 have sf: simple-function M ($f \circ X$)
 using X by auto
 have $\mathcal{H}(X) = \mathcal{H}(\text{the-inv-into } (X\text{'space } M) f \circ (f \circ X))$
 unfolding o-assoc
 apply (subst entropy-simple-distributed[OF simple-distributedI[OF X measure-nonneg
 refl]])
 apply (subst entropy-simple-distributed[OF simple-distributedI[OF simple-function-compose[OF
 X]], where $f=\lambda x. \text{prob } (X - \{x\} \cap \text{space } M)$))
 apply (auto intro!: sum.cong arg-cong[where $f=\text{prob}$] image-eqI simp: the-inv-into-f-f[OF
 inj] comp-def measure-nonneg)
 done
 also have $\dots \leq \mathcal{H}(f \circ X)$
 using entropy-data-processing[OF sf] .
 finally show $\mathcal{H}(X) \leq \mathcal{H}(f \circ X)$.
 qed
 end

13 Properties of Various Distributions

theory Distributions

imports Convolution Information

begin

lemma (in prob-space) distributed-affine:
 fixes $f :: \text{real} \Rightarrow \text{ennreal}$
 assumes f : distributed M lborel X f
 assumes c : $c \neq 0$
 shows distributed M lborel $(\lambda x. t + c * X x)$ $(\lambda x. f ((x - t) / c) / |c|)$
 unfolding distributed-def
 proof safe
 have [measurable]: $f \in \text{borel-measurable borel}$
 using f by (simp add: distributed-def)
 have [measurable]: $X \in \text{borel-measurable } M$
 using f by (simp add: distributed-def)

```

show  $(\lambda x. f ((x - t) / c) / |c|) \in \text{borel-measurable lborel}$ 
  by simp
show random-variable lborel  $(\lambda x. t + c * X x)$ 
  by simp

have eq:  $\text{ennreal } |c| * (f x / \text{ennreal } |c|) = f x$  for  $x$ 
  using  $c$ 
  by (cases  $f x$ )
      (auto simp: divide-ennreal ennreal-mult[symmetric] ennreal-top-divide en-
nreal-mult-top)

have density lborel  $f = \text{distr } M \text{ lborel } X$ 
  using  $f$  by (simp add: distributed-def)
with  $c$  show  $\text{distr } M \text{ lborel } (\lambda x. t + c * X x) = \text{density lborel } (\lambda x. f ((x - t) /$ 
 $c) / \text{ennreal } |c|)$ 
  by (subst ( $\text{?}$ ) lborel-real-affine[where  $c=c$  and  $t=t$ ])
      (simp-all add: density-density-eq density-distr distr-distr field-simps eq cong:
distr-cong)
qed

lemma (in prob-space) distributed-affineI:
  fixes  $f :: \text{real} \Rightarrow \text{ennreal}$  and  $c :: \text{real}$ 
  assumes  $f$ : distributed  $M \text{ lborel } (\lambda x. (X x - t) / c) (\lambda x. |c| * f (x * c + t))$ 
  assumes  $c$ :  $c \neq 0$ 
  shows distributed  $M \text{ lborel } X f$ 
proof –
  have eq:  $f x * \text{ennreal } |c| / \text{ennreal } |c| = f x$  for  $x$ 
    using  $c$  by (simp add: ennreal-times-divide[symmetric])

  show ?thesis
    using distributed-affine[OF  $f c$ , where  $t=t$ ]  $c$ 
    by (simp add: field-simps eq)
qed

lemma (in prob-space) distributed-AE2:
  assumes [measurable]: distributed  $M N X f \text{ Measurable.pred } N P$ 
  shows  $(AE x \text{ in } M. P (X x)) \longleftrightarrow (AE x \text{ in } N. 0 < f x \longrightarrow P x)$ 
proof –
  have  $(AE x \text{ in } M. P (X x)) \longleftrightarrow (AE x \text{ in } \text{distr } M N X. P x)$ 
    by (simp add: AE-distr-iff)
  also have  $\dots \longleftrightarrow (AE x \text{ in } \text{density } N f. P x)$ 
    unfolding distributed-distr-eq-density[OF assms(1)] ..
  also have  $\dots \longleftrightarrow (AE x \text{ in } N. 0 < f x \longrightarrow P x)$ 
    by (rule AE-density) simp
  finally show ?thesis .
qed

```

13.1 Erlang

lemma *nn-integral-power-times-exp-Icc*:

assumes $[arith]: 0 \leq a$
shows $(\int^+ x. ennreal (x^k * exp (-x)) * indicator \{0 .. a\} x \partial lborel) =$
 $(1 - (\sum_{n \leq k}. (a^n * exp (-a)) / fact n)) * fact k$ (**is** $?I = -$)
proof –
let $?f = \lambda k x. x^k * exp (-x) / fact k$
let $?F = \lambda k x. - (\sum_{n \leq k}. (x^n * exp (-x)) / fact n)$
have $?I * (inverse (real-of-nat (fact k))) =$
 $(\int^+ x. ennreal (x^k * exp (-x)) * indicator \{0 .. a\} x * (inverse (real-of-nat$
 $(fact k))) \partial lborel)$
by (*intro nn-integral-multc[symmetric]*) *auto*
also have $\dots = (\int^+ x. ennreal (?f k x) * indicator \{0 .. a\} x \partial lborel)$
by (*intro nn-integral-cong*)
 $(simp \text{ add: field-simps ennreal-mult' [symmetric] indicator-mult-ennreal})$
also have $\dots = ennreal (?F k a - ?F k 0)$
proof (*rule nn-integral-FTC-Icc*)
fix x **assume** $x \in \{0..a\}$
show $DERIV (?F k) x :> ?f k x$
proof (*induction k*)
case 0 **show** $?case$ **by** (*auto intro!: derivative-eq-intros*)
next
case $(Suc k)$
have $DERIV (\lambda x. ?F k x - (x^{Suc k} * exp (-x)) / fact (Suc k)) x :>$
 $?f k x - ((real (Suc k) - x) * x^k * exp (-x)) / (fact (Suc k))$
by (*intro DERIV-diff Suc*)
 $(auto intro!: derivative-eq-intros simp del: fact-Suc power-Suc$
 $simp \text{ add: field-simps power-Suc [symmetric]})$
also have $(\lambda x. ?F k x - (x^{Suc k} * exp (-x)) / fact (Suc k)) = ?F (Suc k)$
by *simp*
also have $?f k x - ((real (Suc k) - x) * x^k * exp (-x)) / (fact (Suc k))$
 $= ?f (Suc k) x$
by (*auto simp: field-simps simp del: fact-Suc*)
 $(simp-all \text{ add: of-nat-Suc field-simps})$
finally show $?case$.
qed
qed *auto*
also have $\dots = ennreal (1 - (\sum_{n \leq k}. (a^n * exp (-a)) / fact n))$
by (*auto simp: power-0-left if-distrib [where f= $\lambda x. x / a$ for a] sum.If-cases*)
also have $\dots = ennreal ((1 - (\sum_{n \leq k}. (a^n * exp (-a)) / fact n)) * fact k) *$
 $ennreal (inverse (fact k))$
by (*subst ennreal-mult' [symmetric]*) (*auto intro!: arg-cong [where f=ennreal]*)
finally show $?thesis$
by (*auto simp add: mult-right-ennreal-cancel le-less*)
qed

lemma *nn-integral-power-times-exp-Ici*:

shows $(\int^+ x. ennreal (x^k * exp (-x)) * indicator \{0 ..\} x \partial lborel) = real-of-nat$
 $(fact k)$

proof (*rule LIMSEQ-unique*)
let $?X = \lambda n. \int^+ x. \text{ennreal } (x^k * \exp(-x)) * \text{indicator } \{0 \dots \text{real } n\} x \partial \text{lborel}$
show $?X \longrightarrow (\int^+ x. \text{ennreal } (x^k * \exp(-x)) * \text{indicator } \{0 \dots\} x \partial \text{lborel})$
apply (*intro nn-integral-LIMSEQ*)
apply (*auto simp: incseq-def le-fun-def eventually-sequentially*
split: split-indicator intro!: tendsto-eventually)
apply (*metis nat-ceiling-le-eq*)
done

have $((\lambda x::\text{real}. (1 - (\sum_{n \leq k}. (x^n / \exp x) / (\text{fact } n))) * \text{fact } k) \longrightarrow$
 $(1 - (\sum_{n \leq k}. 0 / (\text{fact } n))) * \text{fact } k) \text{ at-top}$
by (*intro tendsto-intros tendsto-power-div-exp-0*) *simp*
then show $?X \longrightarrow \text{real-of-nat } (\text{fact } k)$
by (*subst nn-integral-power-times-exp-Icc*)
(auto simp: exp-minus field-simps intro!: filterlim-compose[OF - filterlim-real-sequentially])
qed

definition *erlang-density* :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**
 $\text{erlang-density } k \ l \ x = (\text{if } x < 0 \text{ then } 0 \text{ else } (l^k * x^k * \exp(-l * x)) / \text{fact } k)$

definition *erlang-CDF* :: $\text{nat} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**
 $\text{erlang-CDF } k \ l \ x = (\text{if } x < 0 \text{ then } 0 \text{ else } 1 - (\sum_{n \leq k}. ((l * x)^n * \exp(-l * x)) / \text{fact } n))$

lemma *erlang-density-nonneg[simp]*: $0 \leq l \implies 0 \leq \text{erlang-density } k \ l \ x$
by (*simp add: erlang-density-def*)

lemma *borel-measurable-erlang-density[measurable]*: $\text{erlang-density } k \ l \in \text{borel-measurable borel}$
by (*auto simp add: erlang-density-def[abs-def]*)

lemma *erlang-CDF-transform*: $0 < l \implies \text{erlang-CDF } k \ l \ a = \text{erlang-CDF } k \ 1 \ (l * a)$
by (*auto simp add: erlang-CDF-def mult-less-0-iff*)

lemma *erlang-CDF-nonneg[simp]*: **assumes** $0 < l$ **shows** $0 \leq \text{erlang-CDF } k \ l \ x$
unfolding *erlang-CDF-def*
proof (*clarsimp simp: not-less*)
assume $0 \leq x$
have $(\sum_{n \leq k}. (l * x)^n * \exp(-l * x)) / \text{fact } n =$
 $\exp(-l * x) * (\sum_{n \leq k}. (l * x)^n / \text{fact } n)$
unfolding *sum-distrib-left* **by** (*intro sum.cong*) (*auto simp: field-simps*)
also have $\dots = (\sum_{n \leq k}. (l * x)^n / \text{fact } n) / \exp(l * x)$
by (*simp add: exp-minus field-simps*)
also have $\dots \leq 1$
proof (*subst divide-le-eq-1-pos*)
show $(\sum_{n \leq k}. (l * x)^n / \text{fact } n) \leq \exp(l * x)$
using $\langle 0 < l \rangle \langle 0 \leq x \rangle$ *summable-exp-generic[of l * x]*

by (auto simp: exp-def divide-inverse ac-simps intro!: sum-le-suminf)
qed simp
finally show $(\sum_{n \leq k}. (l * x) \wedge n * \exp(- (l * x)) / \text{fact } n) \leq 1$.
qed

lemma nn-integral-erlang-density:

assumes [arith]: $0 < l$
shows $(\int^+ x. \text{ennreal} (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. \ a\} \ x \ \partial \text{lborel}) = \text{erlang-CDF } k \ l \ a$
proof (cases $0 \leq a$)
case [arith]: True
have eq: $\bigwedge x. \text{indicator } \{0..a\} (x / l) = \text{indicator } \{0..a * l\} x$
by (simp add: field-simps split: split-indicator)
have $(\int^+ x. \text{ennreal} (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. \ a\} \ x \ \partial \text{lborel}) =$
 $(\int^+ x. (l / \text{fact } k) * (\text{ennreal} ((l * x) \wedge k * \exp(- (l * x))) * \text{indicator } \{0 .. a\} \ x) \ \partial \text{lborel})$
by (intro nn-integral-cong)
(auto simp: erlang-density-def power-mult-distrib ennreal-mult[symmetric] split: split-indicator)
also have $\dots = (l / \text{fact } k) * (\int^+ x. \text{ennreal} ((l * x) \wedge k * \exp(- (l * x))) * \text{indicator } \{0 .. a\} \ x \ \partial \text{lborel})$
by (intro nn-integral-cmult) auto
also have $\dots = \text{ennreal} (l / \text{fact } k) * ((1 / l) * (\int^+ x. \text{ennreal} (x \wedge k * \exp(- x)) * \text{indicator } \{0 .. l * a\} \ x \ \partial \text{lborel}))$
by (subst nn-integral-real-affine[where c=1 / l and t=0]) (auto simp: field-simps eq)
also have $\dots = (1 - (\sum_{n \leq k}. ((l * a) \wedge n * \exp(- (l * a))) / \text{fact } n))$
by (subst nn-integral-power-times-exp-Icc) (auto simp: ennreal-mult[symmetric])
also have $\dots = \text{erlang-CDF } k \ l \ a$
by (auto simp: erlang-CDF-def)
finally show ?thesis .
next
case False
then have $(\int^+ x. \text{ennreal} (\text{erlang-density } k \ l \ x) * \text{indicator } \{.. \ a\} \ x \ \partial \text{lborel}) =$
 $(\int^+ x. 0 \ \partial (\text{lborel}::\text{real measure}))$
by (intro nn-integral-cong) (auto simp: erlang-density-def)
with False show ?thesis
by (simp add: erlang-CDF-def)
qed

lemma emeasure-erlang-density:

$0 < l \implies \text{emeasure} (\text{density } \text{lborel} (\text{erlang-density } k \ l)) \{.. \ a\} = \text{erlang-CDF } k \ l \ a$
by (simp add: emeasure-density nn-integral-erlang-density)

lemma nn-integral-erlang-ith-moment:

fixes $k \ i :: \text{nat}$ and $l :: \text{real}$
assumes [arith]: $0 < l$
shows $(\int^+ x. \text{ennreal} (\text{erlang-density } k \ l \ x * x \wedge i) \ \partial \text{lborel}) = \text{fact } (k + i) /$


```

(fact k * l ^ i)
proof -
  have eq:  $\bigwedge x. \text{indicator } \{0..\} (x / l) = \text{indicator } \{0..\} x$ 
    by (simp add: field-simps split: split-indicator)
  have  $(\int^+ x. \text{ennreal } (\text{erlang-density } k \ l \ x * x^i) \ \partial \text{lborel}) =$ 
 $(\int^+ x. (l / (\text{fact } k * l^i)) * (\text{ennreal } ((l * x)^{(k+i)} * \exp(- (l * x)))) * \text{indicator } \{0$ 
 $..\} x) \ \partial \text{lborel})$ 
    by (intro nn-integral-cong)
    (auto simp: erlang-density-def power-mult-distrib power-add ennreal-mult'[symmetric]
split: split-indicator)
  also have  $\dots = (l / (\text{fact } k * l^i)) * (\int^+ x. \text{ennreal } ((l * x)^{(k+i)} * \exp(- (l * x))))$ 
 $* \text{indicator } \{0 ..\} x \ \partial \text{lborel})$ 
    by (intro nn-integral-cmult) auto
  also have  $\dots = \text{ennreal } (l / (\text{fact } k * l^i)) * ((1 / l) * (\int^+ x. \text{ennreal } (x^{(k+i)} * \exp(- x)) * \text{indicator } \{0 ..\} x \ \partial \text{lborel}))$ 
    by (subst nn-integral-real-affine[where c=1 / l and t=0]) (auto simp: field-simps eq)
  also have  $\dots = \text{fact } (k + i) / (\text{fact } k * l^i)$ 
    by (subst nn-integral-power-times-exp-Ici) (auto simp: ennreal-mult'[symmetric])
  finally show ?thesis .
qed

```

```

lemma prob-space-erlang-density:
  assumes l[arith]:  $0 < l$ 
  shows prob-space (density lborel (erlang-density k l)) (is prob-space ?D)
proof
  show emeasure ?D (space ?D) = 1
    using nn-integral-erlang-ith-moment[OF l, where k=k and i=0] by (simp
add: emeasure-density)
qed

```

```

lemma (in prob-space) erlang-distributed-le:
  assumes D: distributed M lborel X (erlang-density k l)
  assumes [simp, arith]:  $0 < l \ 0 \leq a$ 
  shows  $\mathcal{P}(x \text{ in } M. X \ x \leq a) = \text{erlang-CDF } k \ l \ a$ 
proof -
  have emeasure M  $\{x \in \text{space } M. X \ x \leq a\} = \text{emeasure } (\text{distr } M \ \text{lborel } X) \ \{..$ 
 $a\}$ 
    using distributed-measurable[OF D]
    by (subst emeasure-distr) (auto intro!: arg-cong2[where f=emeasure])
  also have  $\dots = \text{emeasure } (\text{density lborel } (\text{erlang-density } k \ l)) \ \{.. \ a\}$ 
    unfolding distributed-distr-eq-density[OF D] ..
  also have  $\dots = \text{erlang-CDF } k \ l \ a$ 
    by (auto intro!: emeasure-erlang-density)
  finally show ?thesis
    by (auto simp: emeasure-eq-measure measure-nonneg)
qed

```

```

lemma (in prob-space) erlang-distributed-gt:

```

```

assumes  $D[simp]$ : distributed  $M$  lborel  $X$  (erlang-density  $k$   $l$ )
assumes  $[arith]$ :  $0 < l$   $0 \leq a$ 
shows  $\mathcal{P}(x \text{ in } M. a < X x) = 1 - (\text{erlang-CDF } k \ l \ a)$ 
proof –
  have  $1 - (\text{erlang-CDF } k \ l \ a) = 1 - \mathcal{P}(x \text{ in } M. X x \leq a)$  by (subst erlang-distributed-le) auto
  also have  $\dots = \text{prob } (space \ M - \{x \in space \ M. X x \leq a\})$ 
    using distributed-measurable[OF D] by (auto simp: prob-compl)
  also have  $\dots = \mathcal{P}(x \text{ in } M. a < X x)$  by (auto intro!: arg-cong[where f=prob])
  simp: not-le
  finally show ?thesis by simp
qed

```

```

lemma erlang-CDF-at0: erlang-CDF  $k$   $l$   $0 = 0$ 
  by (induction k) (auto simp: erlang-CDF-def)

```

```

lemma erlang-distributedI:
  assumes  $X[measurable]$ :  $X \in \text{borel-measurable } M$  and  $[arith]$ :  $0 < l$ 
  and  $X\text{-distr}$ :  $\bigwedge a. 0 \leq a \implies \text{emeasure } M \{x \in space \ M. X x \leq a\} = \text{erlang-CDF } k \ l \ a$ 
  shows distributed  $M$  lborel  $X$  (erlang-density  $k$   $l$ )
proof (rule distributedI-borel-atMost)
  fix  $a :: \text{real}$ 
  { assume  $a \leq 0$ 
    with  $X$  have  $\text{emeasure } M \{x \in space \ M. X x \leq a\} \leq \text{emeasure } M \{x \in space \ M. X x \leq 0\}$ 
    by (intro emeasure-mono) auto
    also have  $\dots = 0$  by (auto intro!: erlang-CDF-at0 simp: X-distr[of 0])
    finally have  $\text{emeasure } M \{x \in space \ M. X x \leq a\} \leq 0$  by simp
    then have  $\text{emeasure } M \{x \in space \ M. X x \leq a\} = 0$  by simp
  }
  note  $eq-0 = \text{this}$ 

```

```

show  $(\int^+ x. \text{erlang-density } k \ l \ x * \text{indicator } \{..a\} \ x \ \partial \text{lborel}) = \text{ennreal } (\text{erlang-CDF } k \ l \ a)$ 
  using nn-integral-erlang-density[of l k a]
  by (simp add: ennreal-indicator ennreal-mult)

```

```

show  $\text{emeasure } M \{x \in space \ M. X x \leq a\} = \text{ennreal } (\text{erlang-CDF } k \ l \ a)$ 
  using  $X\text{-distr}[of a]$   $eq-0$  by (auto simp: one-ennreal-def erlang-CDF-def)
qed simp-all

```

```

lemma (in prob-space) erlang-distributed-iff:
  assumes  $[arith]$ :  $0 < l$ 
  shows distributed  $M$  lborel  $X$  (erlang-density  $k$   $l$ )  $\longleftrightarrow$ 
     $(X \in \text{borel-measurable } M \wedge 0 < l \wedge (\forall a \geq 0. \mathcal{P}(x \text{ in } M. X x \leq a) = \text{erlang-CDF } k \ l \ a))$ 
  using
    distributed-measurable[of M lborel X erlang-density k l]

```

```

    emeasure-erlang-density[of l]
    erlang-distributed-le[of X k l]
  by (auto intro!: erlang-distributedI simp: one-enreal-def emeasure-eq-measure)

lemma (in prob-space) erlang-distributed-mult-const:
  assumes erlX: distributed M lborel X (erlang-density k l)
  assumes a-pos[arith]: 0 < α 0 < l
  shows distributed M lborel (λx. α * X x) (erlang-density k (l / α))
proof (subst erlang-distributed-iff, safe)
  have [measurable]: random-variable borel X and [arith]: 0 < l
  and [simp]: ∧a. 0 ≤ a ⇒ prob {x ∈ space M. X x ≤ a} = erlang-CDF k l a
  by (insert erlX, auto simp: erlang-distributed-iff)

  show random-variable borel (λx. α * X x) 0 < l / α 0 < l / α
  by (auto simp: field-simps)

  fix a:: real assume [arith]: 0 ≤ a
  obtain b:: real where [simp, arith]: b = a / α by blast

  have [arith]: 0 ≤ b by (auto simp: divide-nonneg-pos)

  have prob {x ∈ space M. α * X x ≤ a} = prob {x ∈ space M. X x ≤ b}
  by (rule arg-cong[where f=prob]) (auto simp: field-simps)

  moreover have prob {x ∈ space M. X x ≤ b} = erlang-CDF k l b by auto
  moreover have erlang-CDF k (l / α) a = erlang-CDF k l b unfolding er-
    lang-CDF-def by auto
  ultimately show prob {x ∈ space M. α * X x ≤ a} = erlang-CDF k (l / α) a
  by fastforce
qed

lemma (in prob-space) has-bochner-integral-erlang-ith-moment:
  fixes k i :: nat and l :: real
  assumes [arith]: 0 < l and D: distributed M lborel X (erlang-density k l)
  shows has-bochner-integral M (λx. X x ^ i) (fact (k + i) / (fact k * l ^ i))
proof (rule has-bochner-integral-nn-integral)
  show AE x in M. 0 ≤ X x ^ i
  by (subst distributed-AE2[OF D]) (auto simp: erlang-density-def)
  show (∫+ x. ennreal (X x ^ i) ∂M) = ennreal (fact (k + i) / (fact k * l ^ i))
  using nn-integral-erlang-ith-moment[of l k i]
  by (subst distributed-nn-integral[symmetric, OF D]) (auto simp: ennreal-mult')
qed (insert distributed-measurable[OF D], auto)

lemma (in prob-space) erlang-ith-moment-integrable:
  0 < l ⇒ distributed M lborel X (erlang-density k l) ⇒ integrable M (λx. X x
    ^ i)
  by rule (rule has-bochner-integral-erlang-ith-moment)

lemma (in prob-space) erlang-ith-moment:

```

$0 < l \implies \text{distributed } M \text{ lborel } X \text{ (erlang-density } k \text{ } l) \implies$
 $\text{expectation } (\lambda x. X x \wedge i) = \text{fact } (k + i) / (\text{fact } k * l \wedge i)$
by (rule has-bochner-integral-integral-eq) (rule has-bochner-integral-erlang-ith-moment)

lemma (in prob-space) erlang-distributed-variance:
assumes [arith]: $0 < l$ **and** distributed $M \text{ lborel } X \text{ (erlang-density } k \text{ } l)$
shows variance $X = (k + 1) / l^2$
proof (subst variance-eq)
show integrable $M X$ integrable $M (\lambda x. (X x)^2)$
using erlang-ith-moment-integrable[OF assms, of 1] erlang-ith-moment-integrable[OF
 assms, of 2]
by auto

show expectation $(\lambda x. (X x)^2) - (\text{expectation } X)^2 = \text{real } (k + 1) / l^2$
using erlang-ith-moment[OF assms, of 1] erlang-ith-moment[OF assms, of 2]
by simp (auto simp: power2-eq-square field-simps of-nat-Suc)
qed

13.2 Exponential distribution

abbreviation exponential-density :: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**
 exponential-density \equiv erlang-density 0

lemma exponential-density-def:
 exponential-density $l \ x = (\text{if } x < 0 \text{ then } 0 \text{ else } l * \exp(-x * l))$
by (simp add: fun-eq-iff erlang-density-def)

lemma erlang-CDF-0: erlang-CDF 0 $l \ a = (\text{if } 0 \leq a \text{ then } 1 - \exp(-l * a) \text{ else } 0)$
by (simp add: erlang-CDF-def)

lemma prob-space-exponential-density: $0 < l \implies \text{prob-space (density lborel (exponential-density } l))$
by (rule prob-space-erlang-density)

lemma (in prob-space) exponential-distributedD-le:
assumes D : distributed $M \text{ lborel } X \text{ (exponential-density } l)$ **and** a : $0 \leq a$ **and** l :
 $0 < l$
shows $\mathcal{P}(x \text{ in } M. X x \leq a) = 1 - \exp(-a * l)$
using erlang-distributed-le[OF $D \ l \ a$] **by** (simp add: erlang-CDF-def)

lemma (in prob-space) exponential-distributedD-gt:
assumes D : distributed $M \text{ lborel } X \text{ (exponential-density } l)$ **and** a : $0 \leq a$ **and** l :
 $0 < l$
shows $\mathcal{P}(x \text{ in } M. a < X x) = \exp(-a * l)$
using erlang-distributed-gt[OF $D \ l \ a$] **by** (simp add: erlang-CDF-def)

lemma (in prob-space) exponential-distributed-memoryless:
assumes D : distributed $M \text{ lborel } X \text{ (exponential-density } l)$ **and** a : $0 \leq a$ **and** l :

$0 < l$ and $t: 0 \leq t$
shows $\mathcal{P}(x \text{ in } M. a + t < X x \mid a < X x) = \mathcal{P}(x \text{ in } M. t < X x)$
proof –
have $\mathcal{P}(x \text{ in } M. a + t < X x \mid a < X x) = \mathcal{P}(x \text{ in } M. a + t < X x) / \mathcal{P}(x \text{ in } M. a < X x)$
using $\langle 0 \leq t \rangle$ **by** (*auto simp: cond-prob-def intro!: arg-cong[where f=prob]*
arg-cong2[where f=(/)])
also have $\dots = \exp(-(a + t) * l) / \exp(-a * l)$
using $a \ t$ **by** (*simp add: exponential-distributedD-gt[OF D - l]*)
also have $\dots = \exp(-t * l)$
using l **by** (*auto simp: field-simps exp-add[symmetric]*)
finally show *?thesis*
using t **by** (*simp add: exponential-distributedD-gt[OF D - l]*)
qed

lemma *exponential-distributedI*:
assumes $X[\text{measurable}]: X \in \text{borel-measurable } M$ **and** $[\text{arith}]: 0 < l$
and $X\text{-distr}: \bigwedge a. 0 \leq a \implies \text{emeasure } M \{x \in \text{space } M. X x \leq a\} = 1 - \exp(-a * l)$
shows $\text{distributed } M \text{ lborel } X \text{ (exponential-density } l)$
proof (*rule erlang-distributedI*)
fix $a :: \text{real}$ **assume** $0 \leq a$ **then show** $\text{emeasure } M \{x \in \text{space } M. X x \leq a\} = \text{ennreal (erlang-CDF } 0 \ l \ a)$
using $X\text{-distr}[of \ a]$ **by** (*simp add: erlang-CDF-def ennreal-minus ennreal-1[symmetric]*
del: ennreal-1)
qed fact+

lemma (*in prob-space*) *exponential-distributed-iff*:
assumes $0 < l$
shows $\text{distributed } M \text{ lborel } X \text{ (exponential-density } l) \longleftrightarrow$
 $(X \in \text{borel-measurable } M \wedge (\forall a \geq 0. \mathcal{P}(x \text{ in } M. X x \leq a) = 1 - \exp(-a * l)))$
using *assms erlang-distributed-iff[of l X 0]* **by** (*auto simp: erlang-CDF-0*)

lemma (*in prob-space*) *exponential-distributed-expectation*:
 $0 < l \implies \text{distributed } M \text{ lborel } X \text{ (exponential-density } l) \implies \text{expectation } X = 1 / l$
using *erlang-ith-moment[of l X 0 1]* **by** *simp*

lemma *exponential-density-nonneg*: $0 < l \implies 0 \leq \text{exponential-density } l \ x$
by (*auto simp: exponential-density-def*)

lemma (*in prob-space*) *exponential-distributed-min*:
assumes $0 < l \ 0 < u$
assumes $\text{expX}: \text{distributed } M \text{ lborel } X \text{ (exponential-density } l)$
assumes $\text{expY}: \text{distributed } M \text{ lborel } Y \text{ (exponential-density } u)$
assumes *ind: indep-var borel X borel Y*
shows $\text{distributed } M \text{ lborel } (\lambda x. \min(X x) (Y x)) \text{ (exponential-density } (l + u))$
proof (*subst exponential-distributed-iff, safe*)

```

have randX: random-variable borel X
  using expX ⟨0 < l⟩ by (simp add: exponential-distributed-iff)
moreover have randY: random-variable borel Y
  using expY ⟨0 < u⟩ by (simp add: exponential-distributed-iff)
ultimately show random-variable borel (λx. min (X x) (Y x)) by auto

show 0 < l + u
  using ⟨0 < l⟩ ⟨0 < u⟩ by auto

fix a::real assume a[arith]: 0 ≤ a
have gt1[simp]:  $\mathcal{P}(x \text{ in } M. a < X x) = \exp(-a * l)$ 
  by (rule exponential-distributedD-gt[OF expX a]) fact
have gt2[simp]:  $\mathcal{P}(x \text{ in } M. a < Y x) = \exp(-a * u)$ 
  by (rule exponential-distributedD-gt[OF expY a]) fact

have  $\mathcal{P}(x \text{ in } M. a < (\min (X x) (Y x))) = \mathcal{P}(x \text{ in } M. a < (X x) \wedge a < (Y x))$ 
by (auto intro!: arg-cong[where f=prob])

also have ... =  $\mathcal{P}(x \text{ in } M. a < (X x)) * \mathcal{P}(x \text{ in } M. a < (Y x))$ 
  using prob-indep-random-variable[OF ind, of {a <..} {a <..}] by simp
also have ... =  $\exp(-a * (l + u))$  by (auto simp: field-simps mult-exp-exp)
finally have indep-prob:  $\mathcal{P}(x \text{ in } M. a < (\min (X x) (Y x))) = \exp(-a * (l + u))$  .

have  $\{x \in \text{space } M. (\min (X x) (Y x)) \leq a\} = (\text{space } M - \{x \in \text{space } M. a < (\min (X x) (Y x))\})$ 
  by auto
then have  $1 - \text{prob } \{x \in \text{space } M. a < (\min (X x) (Y x))\} = \text{prob } \{x \in \text{space } M. (\min (X x) (Y x)) \leq a\}$ 
  using randX randY by (auto simp: prob-compl)
then show  $\text{prob } \{x \in \text{space } M. (\min (X x) (Y x)) \leq a\} = 1 - \exp(-a * (l + u))$ 
  using indep-prob by auto
qed

lemma (in prob-space) exponential-distributed-Min:
  assumes finI: finite I
  assumes A:  $I \neq \{\}$ 
  assumes l:  $\bigwedge i. i \in I \implies 0 < l i$ 
  assumes expX:  $\bigwedge i. i \in I \implies \text{distributed } M \text{ lborel } (X i) (\text{exponential-density } (l i))$ 
  assumes ind: indep-vars (λi. borel) X I
  shows  $\text{distributed } M \text{ lborel } (\lambda x. \text{Min } ((\lambda i. X i x) 'I)) (\text{exponential-density } (\sum_{i \in I. l i}))$ 
  using assms
  proof (induct rule: finite-ne-induct)
    case (singleton i) then show ?case by simp
  next
    case (insert i I)

```

```

then have distributed  $M$  lborel  $(\lambda x. \min (X \ i \ x) (Min ((\lambda i. X \ i \ x) 'I)))$  (exponential-density
 $(l \ i + (\sum_{i \in I}. l \ i)))$ )
  by (intro exponential-distributed-min indep-vars-Min insert)
      (auto intro: indep-vars-subset sum-pos)
then show ?case
using insert by simp
qed

```

```

lemma (in prob-space) exponential-distributed-variance:
   $0 < l \implies$  distributed  $M$  lborel  $X$  (exponential-density  $l$ )  $\implies$  variance  $X = 1 / l^2$ 
using erlang-distributed-variance[of  $l \ X \ 0$ ] by simp

```

```

lemma nn-integral-zero':  $AE \ x \ in \ M. f \ x = 0 \implies (\int^+ x. f \ x \ \partial M) = 0$ 
by (simp cong: nn-integral-cong-AE)

```

```

lemma convolution-erlang-density:
  fixes  $k_1 \ k_2 :: nat$ 
  assumes [simp, arith]:  $0 < l$ 
  shows  $(\lambda x. \int^+ y. ennreal (erlang-density \ k_1 \ l \ (x - y)) * ennreal (erlang-density$ 
 $k_2 \ l \ y) \ \partial lborel) =$ 
   $(erlang-density (Suc \ k_1 + Suc \ k_2 - 1) \ l)$ 
  (is ?LHS = ?RHS)

```

```

proof
  fix  $x :: real$ 
  have  $x \leq 0 \vee 0 < x$ 
  by arith
  then show ?LHS  $x =$  ?RHS  $x$ 
  proof
    assume  $x \leq 0$  then show ?thesis
    apply (subst nn-integral-zero')
    apply (rule AE-I[where  $N = \{0\}$ ])
    apply (auto simp add: erlang-density-def not-less)
    done
  next
    note zero-le-mult-iff[simp] zero-le-divide-iff[simp]

```

```

  have I-eq1:  $integral^N \ lborel (erlang-density (Suc \ k_1 + Suc \ k_2 - 1) \ l) = 1$ 
    using nn-integral-erlang-ith-moment[of  $l \ Suc \ k_1 + Suc \ k_2 - 1 \ 0$ ] by (simp
del: fact-Suc)

```

```

  have 1:  $(\int^+ x. ennreal (erlang-density (Suc \ k_1 + Suc \ k_2 - 1) \ l \ x * indicator$ 
 $\{0 < ..\} \ x) \ \partial lborel) = 1$ 
    apply (subst I-eq1[symmetric])
    unfolding erlang-density-def
    by (auto intro!: nn-integral-cong split:split-indicator)

```

```

have prob-space (density lborel ?LHS)
by (intro prob-space-convolution-density)
    (auto intro!: prob-space-erlang-density erlang-density-nonneg)

```

```

then have 2:  $\text{integral}^N \text{lborel } ?LHS = 1$ 
  by (auto dest!: prob-space.emeasure-space-1 simp: emeasure-density)

let ?I = ( $\text{integral}^N \text{lborel } (\lambda y. \text{ennreal } ((1 - y)^\wedge k_1 * y^\wedge k_2 * \text{indicator } \{0..1\} y)))$ )
let ?C = ( $\text{fact } (\text{Suc } (k_1 + k_2))) / ((\text{fact } k_1) * (\text{fact } k_2))$ )
let ?s =  $\text{Suc } k_1 + \text{Suc } k_2 - 1$ 
let ?L = ( $\lambda x. \int^+ y. \text{ennreal } (\text{erlang-density } k_1 \text{ l } (x - y) * \text{erlang-density } k_2 \text{ l } y * \text{indicator } \{0..x\} y) \partial \text{lborel}$ )

{ fix x :: real assume [arith]:  $0 < x$ 
  have *:  $\bigwedge x y n. (x - y * x :: \text{real})^\wedge n = x^\wedge n * (1 - y)^\wedge n$ 
    unfolding power-mult-distrib[symmetric] by (simp add: field-simps)

  have ?LHS x = ?L x
    unfolding erlang-density-def
    by (auto intro!: nn-integral-cong simp: ennreal-mult split:split-indicator)
  also have ... = ( $\lambda x. \text{ennreal } ?C * ?I * \text{erlang-density } ?s \text{ l } x$ ) x
    apply (subst nn-integral-real-affine[where c=x and t = 0])
  apply (simp-all add: nn-integral-cmult[symmetric] nn-integral-multc[symmetric]
del: fact-Suc)
    apply (intro nn-integral-cong)
    apply (auto simp add: erlang-density-def mult-less-0-iff exp-minus field-simps
exp-diff power-add *
      ennreal-mult[symmetric]
      simp del: fact-Suc split: split-indicator)
  done
  finally have ( $\int^+ y. \text{ennreal } (\text{erlang-density } k_1 \text{ l } (x - y) * \text{erlang-density } k_2 \text{ l } y) \partial \text{lborel}$ ) =
    ( $\lambda x. \text{ennreal } ?C * ?I * \text{erlang-density } ?s \text{ l } x$ ) x
    by (simp add: ennreal-mult) }
note * = this

assume [arith]:  $0 < x$ 
have 3:  $1 = \text{integral}^N \text{lborel } (\lambda xa. ?LHS xa * \text{indicator } \{0<..\} xa)$ 
  by (subst 2[symmetric]
    (auto intro!: nn-integral-cong-AE AE-I[where N={0}]
      simp: erlang-density-def nn-integral-multc[symmetric] indicator-def
split: if-split-asm))
  also have ... =  $\text{integral}^N \text{lborel } (\lambda x. (\text{ennreal } (?C) * ?I) * ((\text{erlang-density } ?s \text{ l } x) * \text{indicator } \{0<..\} x))$ 
    by (auto intro!: nn-integral-cong simp: ennreal-mult[symmetric] * split: split-indicator)
  also have ... =  $\text{ennreal } (?C) * ?I$ 
    using 1
    by (auto simp: nn-integral-cmult)
  finally have  $\text{ennreal } (?C) * ?I = 1$  by presburger

then show ?thesis
  using * by (simp add: ennreal-mult)

```


qed
qed

lemma (in prob-space) sum-indep-erlang:
 assumes indep: indep-var borel X borel Y
 assumes [simp, arith]: $0 < l$
 assumes erlX: distributed M lborel X (erlang-density k_1 l)
 assumes erlY: distributed M lborel Y (erlang-density k_2 l)
 shows distributed M lborel $(\lambda x. X\ x + Y\ x)$ (erlang-density $(\text{Suc } k_1 + \text{Suc } k_2 - 1)$ l)
 using assms
 apply (subst convolution-erlang-density[symmetric, OF <0<l])
 apply (intro distributed-convolution)
 apply auto
 done

lemma (in prob-space) erlang-distributed-sum:
 assumes finI: finite I
 assumes A: $I \neq \{\}$
 assumes [simp, arith]: $0 < l$
 assumes expX: $\bigwedge i. i \in I \implies \text{distributed } M \text{ lborel } (X\ i) \text{ (erlang-density } (k\ i)\ l)$
 assumes ind: indep-vars $(\lambda i. \text{borel})\ X\ I$
 shows distributed M lborel $(\lambda x. \sum_{i \in I}. X\ i\ x)$ (erlang-density $((\sum_{i \in I}. \text{Suc } (k\ i)) - 1)$ l)
 using assms
 proof (induct rule: finite-ne-induct)
 case (singleton i) then show ?case by auto
 next
 case (insert $i\ I$)
 then have distributed M lborel $(\lambda x. (X\ i\ x) + (\sum_{i \in I}. X\ i\ x))$ (erlang-density $(\text{Suc } (k\ i) + \text{Suc } ((\sum_{i \in I}. \text{Suc } (k\ i)) - 1) - 1)$ l)
 by (intro sum-indep-erlang indep-vars-sum) (auto intro!: indep-vars-subset)
 also have $(\lambda x. (X\ i\ x) + (\sum_{i \in I}. X\ i\ x)) = (\lambda x. \sum_{i \in \text{insert } i\ I}. X\ i\ x)$
 using insert by auto
 also have $\text{Suc } (k\ i) + \text{Suc } ((\sum_{i \in I}. \text{Suc } (k\ i)) - 1) - 1 = (\sum_{i \in \text{insert } i\ I}. \text{Suc } (k\ i)) - 1$
 using insert by (auto intro!: Suc-pred simp: ac-simps)
 finally show ?case by fast
 qed

lemma (in prob-space) exponential-distributed-sum:
 assumes finI: finite I
 assumes A: $I \neq \{\}$
 assumes l: $0 < l$
 assumes expX: $\bigwedge i. i \in I \implies \text{distributed } M \text{ lborel } (X\ i) \text{ (exponential-density } l)$
 assumes ind: indep-vars $(\lambda i. \text{borel})\ X\ I$
 shows distributed M lborel $(\lambda x. \sum_{i \in I}. X\ i\ x)$ (erlang-density $((\text{card } I) - 1)$ l)
 using erlang-distributed-sum[OF assms] by simp

lemma (in *information-space*) *entropy-exponential*:
assumes $l[simp, arith]: 0 < l$
assumes D : distributed M lborel X (exponential-density l)
shows entropy b lborel $X = \log b$ (exp $1 / l$)
proof –
have $[simp]:$ integrable lborel (exponential-density l)
using distributed-integrable[OF D , of $\lambda\cdot. 1$] **by** *simp*

have $[simp]:$ integral ^{L} lborel (exponential-density l) = 1
using distributed-integral[OF D , of $\lambda\cdot. 1$] **by** (*simp add: prob-space*)

have $[simp]:$ integrable lborel ($\lambda x.$ exponential-density l $x * x$)
using erlang-ith-moment-integrable[OF l D , of 1] distributed-integrable[OF D , of $\lambda x. x$] **by** *simp*

have $[simp]:$ integral ^{L} lborel ($\lambda x.$ exponential-density l $x * x$) = $1 / l$
using erlang-ith-moment[OF l D , of 1] distributed-integral[OF D , of $\lambda x. x$] **by** *simp*

have entropy b lborel $X = - (\int x.$ exponential-density l $x * \log b$ (exponential-density l x) ∂ lborel)
using D **by** (*rule entropy-distr*) *simp*
also have $(\int x.$ exponential-density l $x * \log b$ (exponential-density l x) ∂ lborel)
= $(\int x.$ ($\ln l * \text{exponential-density } l \ x - l * (\text{exponential-density } l \ x * x)$) ∂ lborel)
by (*intro Bochner-Integration.integral-cong*) (*auto simp: log-def ln-mult exponential-density-def field-simps*)
also have $\dots = (\ln l - 1) / \ln b$
by *simp*
finally show ?thesis
by (*simp add: log-def ln-div*) (*simp add: field-split-simps*)
qed

13.3 Uniform distribution

lemma *uniform-distrI*:
assumes $X: X \in \text{measurable } M \ M'$
and $A: A \in \text{sets } M' \text{ emeasure } M' \ A \neq \infty \text{ emeasure } M' \ A \neq 0$
assumes *distr*: $\bigwedge B. B \in \text{sets } M' \implies \text{emeasure } M \ (X -' B \cap \text{space } M) = \text{emeasure } M' \ (A \cap B) / \text{emeasure } M' \ A$
shows $\text{distr } M \ M' \ X = \text{uniform-measure } M' \ A$
unfolding *uniform-measure-def*
proof (*intro measure-eqI*)
let ?f = $\lambda x.$ indicator A x / $\text{emeasure } M' \ A$
fix B **assume** $B: B \in \text{sets } (distr \ M \ M' \ X)$
with X **have** $\text{emeasure } M \ (X -' B \cap \text{space } M) = \text{emeasure } M' \ (A \cap B) / \text{emeasure } M' \ A$
by (*simp add: distr[of B] measurable-sets*)

```

also have ... = (1 / emeasure M' A) * emeasure M' (A ∩ B)
  by (simp add: divide-ennreal-def ac-simps)
also have ... = (∫+ x. (1 / emeasure M' A) * indicator (A ∩ B) x ∂M')
  using A B
  by (intro nn-integral-cmult-indicator[symmetric]) (auto intro!: )
also have ... = (∫+ x. ?f x * indicator B x ∂M')
  by (rule nn-integral-cong) (auto split: split-indicator)
finally show emeasure (distr M M' X) B = emeasure (density M' ?f) B
  using A B X by (auto simp add: emeasure-distr emeasure-density)
qed simp

```

lemma *uniform-distrI-borel*:

```

fixes A :: real set
assumes X[measurable]: X ∈ borel-measurable M and A: emeasure lborel A =
ennreal r 0 < r
  and [measurable]: A ∈ sets borel
assumes distr: ⋀a. emeasure M {x ∈ space M. X x ≤ a} = emeasure lborel (A ∩
{.. a}) / r
shows distributed M lborel X (λx. indicator A x / measure lborel A)
proof (rule distributedI-borel-atMost)
  let ?f = λx. 1 / r * indicator A x
  fix a
  have emeasure lborel (A ∩ {..a}) ≤ emeasure lborel A
    using A by (intro emeasure-mono) auto
  also have ... < ∞
    using A by simp
  finally have fin: emeasure lborel (A ∩ {..a}) ≠ top
    by simp
  from emeasure-eq-ennreal-measure[OF this]
  have fin-eq: emeasure lborel (A ∩ {..a}) / r = ennreal (measure lborel (A ∩
{..a}) / r)
    using A by (simp add: divide-ennreal measure-nonneg)
  then show emeasure M {x ∈ space M. X x ≤ a} = ennreal (measure lborel (A ∩
{..a}) / r)
    using distr by simp

```

```

  have (∫+ x. ennreal (indicator A x / measure lborel A * indicator {..a} x)
∂lborel) =
  (∫+ x. ennreal (1 / measure lborel A) * indicator (A ∩ {..a}) x ∂lborel)
  by (auto intro!: nn-integral-cong split: split-indicator)
also have ... = ennreal (1 / measure lborel A) * emeasure lborel (A ∩ {..a})
  using ⟨A ∈ sets borel⟩
  by (intro nn-integral-cmult-indicator) (auto simp: measure-nonneg)
also have ... = ennreal (measure lborel (A ∩ {..a}) / r)
  unfolding emeasure-eq-ennreal-measure[OF fin] using A
  by (simp add: measure-def ennreal-mult[symmetric])
finally show (∫+ x. ennreal (indicator A x / measure lborel A * indicator {..a}
x) ∂lborel) =
  ennreal (measure lborel (A ∩ {..a}) / r) .

```

qed (*auto simp: measure-nonneg*)

lemma (*in prob-space*) *uniform-distrI-borel-atLeastAtMost*:

fixes $a\ b :: \text{real}$

assumes $X: X \in \text{borel-measurable } M$ **and** $a < b$

assumes $\text{distr}: \bigwedge t. a \leq t \implies t \leq b \implies \mathcal{P}(x \text{ in } M. X\ x \leq t) = (t - a) / (b - a)$

shows *distributed* M *lborel* X $(\lambda x. \text{indicator } \{a..b\} x / \text{measure lborel } \{a..b\})$

proof (*rule uniform-distrI-borel*)

fix t

have $t < a \vee (a \leq t \wedge t \leq b) \vee b < t$

by *auto*

then show $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} = \text{emeasure lborel } (\{a..b\} \cap \{..t\}) / (b - a)$

proof (*elim disjE conjE*)

assume $t < a$

then have $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} \leq \text{emeasure } M \{x \in \text{space } M. X\ x \leq a\}$

using X **by** (*auto intro!: emeasure-mono measurable-sets*)

also have $\dots = 0$

using $\text{distr}[of\ a] \langle a < b \rangle$ **by** (*simp add: emeasure-eq-measure*)

finally have $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} = 0$

by (*simp add: antisym measure-nonneg*)

with $\langle t < a \rangle$ **show** *?thesis* **by** *simp*

next

assume $bnds: a \leq t \wedge t \leq b$

have $\{a..b\} \cap \{..t\} = \{a..t\}$

using $bnds$ **by** *auto*

then show *?thesis* **using** $\langle a \leq t \rangle \langle a < b \rangle$

using $\text{distr}[OF\ bnds]$ **by** (*simp add: emeasure-eq-measure divide-ennreal*)

next

assume $b < t$

have $1 = \text{emeasure } M \{x \in \text{space } M. X\ x \leq b\}$

using $\text{distr}[of\ b] \langle a < b \rangle$ **by** (*simp add: one-ennreal-def emeasure-eq-measure*)

also have $\dots \leq \text{emeasure } M \{x \in \text{space } M. X\ x \leq t\}$

using $X \langle b < t \rangle$ **by** (*auto intro!: emeasure-mono measurable-sets*)

finally have $\text{emeasure } M \{x \in \text{space } M. X\ x \leq t\} = 1$

by (*simp add: antisym emeasure-eq-measure*)

with $\langle b < t \rangle \langle a < b \rangle$ **show** *?thesis* **by** (*simp add: measure-def divide-ennreal*)

qed

qed (*insert* $X \langle a < b \rangle$, *auto*)

lemma (*in prob-space*) *uniform-distributed-measure*:

fixes $a\ b :: \text{real}$

assumes $D: \text{distributed } M \text{ lborel } X (\lambda x. \text{indicator } \{a..b\} x / \text{measure lborel } \{a..b\})$

assumes $t: a \leq t \wedge t \leq b$

shows $\mathcal{P}(x \text{ in } M. X\ x \leq t) = (t - a) / (b - a)$

proof –

```

have emeasure M {x ∈ space M. X x ≤ t} = emeasure (distr M lborel X) {.. t}
  using distributed-measurable[OF D]
  by (subst emeasure-distr) (auto intro!: arg-cong2[where f=emeasure])
also have ... = (∫+ x. ennreal (1 / (b - a)) * indicator {a .. t} x ∂lborel)
  using distributed-borel-measurable[OF D] ⟨a ≤ t⟩ ⟨t ≤ b⟩
  unfolding distributed-distr-eq-density[OF D]
  by (subst emeasure-density)
    (auto intro!: nn-integral-cong simp: measure-def split: split-indicator)
also have ... = ennreal (1 / (b - a)) * (t - a)
  using ⟨a ≤ t⟩ ⟨t ≤ b⟩
  by (subst nn-integral-cmult-indicator) auto
finally show ?thesis
  using t by (simp add: emeasure-eq-measure ennreal-mult''[symmetric] mea-
sure-nonneg)
qed

```

```

lemma (in prob-space) uniform-distributed-bounds:
  fixes a b :: real
  assumes D: distributed M lborel X (λx. indicator {a .. b} x / measure lborel {a
  .. b})
  shows a < b
proof (rule ccontr)
  assume ¬ a < b
  then have {a .. b} = {} ∨ {a .. b} = {a .. a} by simp
  with uniform-distributed-params[OF D] show False
    by (auto simp: measure-def)
qed

```

```

lemma (in prob-space) uniform-distributed-iff:
  fixes a b :: real
  shows distributed M lborel X (λx. indicator {a..b} x / measure lborel {a..b}) ⟷
    (X ∈ borel-measurable M ∧ a < b ∧ (∀ t ∈ {a .. b}. P(x in M. X x ≤ t) = (t -
a) / (b - a)))
  using
    uniform-distributed-bounds[of X a b]
    uniform-distributed-measure[of X a b]
    distributed-measurable[of M lborel X]
  by (auto intro!: uniform-distrI-borel-atLeastAtMost simp del: content-real-if)

```

```

lemma (in prob-space) uniform-distributed-expectation:
  fixes a b :: real
  assumes D: distributed M lborel X (λx. indicator {a .. b} x / measure lborel {a
  .. b})
  shows expectation X = (a + b) / 2
proof (subst distributed-integral[OF D, of λx. x, symmetric])
  have a < b
    using uniform-distributed-bounds[OF D] .

```

```

have (∫ x. indicator {a .. b} x / measure lborel {a .. b} * x ∂lborel) =

```

```

  (∫ x. (x / measure lborel {a .. b}) * indicator {a .. b} x ∂lborel)
  by (intro Bochner-Integration.integral-cong) auto
  also have (∫ x. (x / measure lborel {a .. b}) * indicator {a .. b} x ∂lborel) =
(a + b) / 2
  proof (subst integral-FTC-Icc-real)
  fix x
  show DERIV (λx. x2 / (2 * measure lborel {a..b})) x :> x / measure lborel
{a..b}
  using uniform-distributed-params[OF D]
  by (auto intro!: derivative-eq-intros simp del: content-real-if)
  show isCont (λx. x / Sigma-Algebra.measure lborel {a..b}) x
  using uniform-distributed-params[OF D]
  by (auto intro!: isCont-divide)
  have *: b2 / (2 * measure lborel {a..b}) - a2 / (2 * measure lborel {a..b}) =
(b*b - a * a) / (2 * (b - a))
  using ⟨a < b⟩
  by (auto simp: measure-def power2-eq-square diff-divide-distrib[symmetric])
  show b2 / (2 * measure lborel {a..b}) - a2 / (2 * measure lborel {a..b}) = (a
+ b) / 2
  using ⟨a < b⟩
  unfolding * square-diff-square-factored by (auto simp: field-simps)
  qed (insert ⟨a < b⟩, simp)
  finally show (∫ x. indicator {a .. b} x / measure lborel {a .. b} * x ∂lborel) =
(a + b) / 2 .
  qed (auto simp: measure-nonneg)

```

lemma (in prob-space) uniform-distributed-variance:

```

  fixes a b :: real
  assumes D: distributed M lborel X (λx. indicator {a .. b} x / measure lborel {a
.. b})
  shows variance X = (b - a)2 / 12
  proof (subst distributed-variance)
  have [arith]: a < b using uniform-distributed-bounds[OF D] .
  let ?μ = expectation X let ?D = λx. indicator {a..b} (x + ?μ) / measure lborel
{a..b}
  have (∫ x. x2 * (?D x) ∂lborel) = (∫ x. x2 * (indicator {a - ?μ .. b - ?μ} x) /
measure lborel {a .. b} ∂lborel)
  by (intro Bochner-Integration.integral-cong) (auto split: split-indicator)
  also have ... = (b - a)2 / 12
  by (simp add: integral-power uniform-distributed-expectation[OF D])
  (simp add: eval-nat-numeral field-simps)
  finally show (∫ x. x2 * ?D x ∂lborel) = (b - a)2 / 12 .
  qed (auto intro: D simp del: content-real-if)

```

13.4 Normal distribution

definition normal-density :: real ⇒ real ⇒ real ⇒ real **where**

normal-density μ σ x = 1 / sqrt (2 * pi * σ²) * exp (-(x - μ)² / (2 * σ²))

abbreviation *std-normal-density* :: *real* \Rightarrow *real* **where**
std-normal-density \equiv *normal-density* 0 1

lemma *std-normal-density-def*: *std-normal-density* $x = (1 / \text{sqrt } (2 * \pi)) * \exp$
 $(- x^2 / 2)$
unfolding *normal-density-def* **by** *simp*

lemma *normal-density-nonneg*[*simp*]: $0 \leq \text{normal-density } \mu \sigma x$
by (*auto simp: normal-density-def*)

lemma *normal-density-pos*: $0 < \sigma \implies 0 < \text{normal-density } \mu \sigma x$
by (*auto simp: normal-density-def*)

lemma *borel-measurable-normal-density*[*measurable*]: *normal-density* $\mu \sigma \in \text{borel-measurable}$
borel
by (*auto simp: normal-density-def[abs-def]*)

lemma *gaussian-moment-0*:
has-bochner-integral lborel $(\lambda x. \text{indicator } \{0..\} x *_{\mathbb{R}} \exp(-x^2)) (\text{sqrt } \pi / 2)$

proof –
let $?pI = \lambda f. (\int^+ s. f(s::\text{real}) * \text{indicator } \{0..\} s \partial \text{lborel})$
let $?gauss = \lambda x. \exp(-x^2)$

let $?I = \text{indicator } \{0 <..\} :: \text{real} \Rightarrow \text{real}$
let $?ff = \lambda x s. x * \exp(-x^2 * (1 + s^2)) :: \text{real}$

have $*: ?pI ?gauss = (\int^+ x. ?gauss x * ?I x \partial \text{lborel})$
by (*intro nn-integral-cong-AE AE-I[where N={0}]*) (*auto split: split-indicator*)

have $?pI ?gauss * ?pI ?gauss = (\int^+ x. \int^+ s. ?gauss x * ?gauss s * ?I s * ?I x \partial \text{lborel} \partial \text{lborel})$

by (*auto simp: nn-integral-cmult[symmetric] nn-integral-multc[symmetric] * ennreal-mult[symmetric]*)

intro!: *nn-integral-cong split: split-indicator*)

also have $\dots = (\int^+ x. \int^+ s. ?ff x s * ?I s * ?I x \partial \text{lborel} \partial \text{lborel})$

proof (*rule nn-integral-cong, cases*)

fix $x :: \text{real}$ **assume** $x \neq 0$

then show $(\int^+ s. ?gauss x * ?gauss s * ?I s * ?I x \partial \text{lborel}) = (\int^+ s. ?ff x s * ?I s * ?I x \partial \text{lborel})$

by (*subst nn-integral-real-affine[where t=0 and c=x]*)

(*auto simp: mult-exp-exp nn-integral-cmult[symmetric] field-simps zero-less-mult-iff ennreal-mult[symmetric]*)

intro!: *nn-integral-cong split: split-indicator*)

qed *simp*

also have $\dots = \int^+ s. \int^+ x. ?ff x s * ?I s * ?I x \partial \text{lborel} \partial \text{lborel}$

by (*rule lborel-pair.Fubini'[symmetric]*) *auto*

also have $\dots = ?pI (\lambda s. ?pI (\lambda x. ?ff x s))$

by (*rule nn-integral-cong-AE*)

(*auto intro!: nn-integral-cong-AE AE-I[where N={0}] split: split-indicator-asm*)

```

also have ... = ?pI (λs. ennreal (1 / (2 * (1 + s2))))
proof (intro nn-integral-cong ennreal-mult-right-cong)
  fix s :: real show ?pI (λx. ?ff x s) = ennreal (1 / (2 * (1 + s2)))
  proof (subst nn-integral-FTC-atLeast)
    have ((λa. - (exp (- (a2 * (1 + s2))) / (2 + 2 * s2))) → (- (0 / (2 +
2 * s2)))) at-top
    apply (intro tendsto-intros filterlim-compose[OF exp-at-bot] filterlim-compose[OF
filterlim-uminus-at-bot-at-top])
    apply (subst mult.commute)
    apply (auto intro!: filterlim-tendsto-pos-mult-at-top
filterlim-at-top-mult-at-top[OF filterlim-ident filterlim-ident]
simp: add-pos-nonneg power2-eq-square add-nonneg-eq-0-iff)
  done
  then show ((λa. - (exp (- a2 - s2 * a2) / (2 + 2 * s2))) → 0) at-top
  by (simp add: field-simps)
qed (auto intro!: derivative-eq-intros simp: field-simps add-nonneg-eq-0-iff)
qed
also have ... = ennreal (pi / 4)
proof (subst nn-integral-FTC-atLeast)
  show ((λa. arctan a / 2) → (pi / 2) / 2) at-top
  by (intro tendsto-intros) (simp-all add: tendsto-arctan-at-top)
qed (auto intro!: derivative-eq-intros simp: add-nonneg-eq-0-iff field-simps power2-eq-square)
finally have ?pI ?gauss2 = pi / 4
  by (simp add: power2-eq-square)
then have ?pI ?gauss = sqrt (pi / 4)
  using power-eq-iff-eq-base[of 2 enn2real (?pI ?gauss) sqrt (pi / 4)]
  by (cases ?pI ?gauss) (auto simp: power2-eq-square ennreal-mult[symmetric]
ennreal-top-mult)
also have ?pI ?gauss = (∫+ x. indicator {0..} x *R exp (- x2) ∂lborel)
  by (intro nn-integral-cong) (simp split: split-indicator)
also have sqrt (pi / 4) = sqrt pi / 2
  by (simp add: real-sqrt-divide)
finally show ?thesis
  by (rule has-bochner-integral-nn-integral[rotated 3])
  auto
qed

```

lemma gaussian-moment-1:

*has-bochner-integral lborel (λx::real. indicator {0..} x *_R (exp (- x²) * x)) (1 / 2)*

proof –

```

have (∫+ x. indicator {0..} x *R (exp (- x2) * x) ∂lborel) =
  (∫+ x. ennreal (x * exp (- x2)) * indicator {0..} x ∂lborel)
by (intro nn-integral-cong)
  (auto simp: ac-simps split: split-indicator)

```

also have ... = ennreal (0 - (- exp (- 0²) / 2))

proof (rule nn-integral-FTC-atLeast)

have ((λx::real. - exp (- x²) / 2) → - 0 / 2) *at-top*

by (intro tendsto-divide tendsto-minus filterlim-compose[OF exp-at-bot])


```

      filterlim-compose[OF filterlim-uminus-at-bot-at-top]
      filterlim-pow-at-top filterlim-ident)
    auto
  then show  $((\lambda a::real. - \exp(-a^2) / 2) \longrightarrow 0)$  at-top
    by simp
qed (auto intro!: derivative-eq-intros)
also have  $\dots = \text{ennreal}(1 / 2)$ 
  by simp
finally show ?thesis
  by (rule has-bochner-integral-nn-integral[rotated 3])
    (auto split: split-indicator)
qed

lemma
  fixes  $k :: nat$ 
  shows gaussian-moment-even-pos:
    has-bochner-integral lborel  $(\lambda x::real. \text{indicator}\{0..\} x *_R (\exp(-x^2) * x^{\wedge}(2 * k)))$ 
       $((\sqrt{\pi} / 2) * (\text{fact}(2 * k) / (2^{\wedge}(2 * k) * \text{fact } k)))$ 
      (is ?even)
  and gaussian-moment-odd-pos:
    has-bochner-integral lborel  $(\lambda x::real. \text{indicator}\{0..\} x *_R (\exp(-x^2) * x^{\wedge}(2 * k + 1)))$ 
       $(\text{fact } k / 2)$ 
      (is ?odd)
  proof -
    let ?M =  $\lambda k x. \exp(-x^2) * x^{\wedge}k :: real$ 

    { fix  $k$  I assume Mk: has-bochner-integral lborel  $(\lambda x. \text{indicator}\{0..\} x *_R ?M k$ 
       $x)$  I
      have  $2 \neq (0::real)$ 
        by linarith
      let ?f =  $\lambda b. \int x. \text{indicator}\{0..\} x *_R ?M (k + 2) x * \text{indicator}\{..b\} x \partial \text{lborel}$ 
      have  $((\lambda b. (k + 1) / 2 * (\int x. \text{indicator}\{..b\} x *_R (\text{indicator}\{0..\} x *_R ?M$ 
         $k x) \partial \text{lborel}) - ?M (k + 1) b / 2) \longrightarrow$ 
         $(k + 1) / 2 * (\int x. \text{indicator}\{0..\} x *_R ?M k x \partial \text{lborel}) - 0 / 2)$  at-top
        (is ?tendsto)
      proof (intro tendsto-intros  $\langle 2 \neq 0 \rangle$  tendsto-integral-at-top sets-lborel Mk[THEN
        integrable.intros])
        show  $(?M (k + 1) \longrightarrow 0)$  at-top
      proof cases
        assume even k
        have  $((\lambda x. ((x^2)^{\wedge}(k \text{ div } 2 + 1) / \exp(x^2)) * (1 / x) :: real) \longrightarrow 0 * 0)$ 
          at-top
        by (intro tendsto-intros tendsto-divide-0[OF tendsto-const] filterlim-compose[OF
          tendsto-power-div-exp-0]
          filterlim-at-top-imp-at-infinity filterlim-ident filterlim-pow-at-top
          filterlim-ident)
          auto
        also have  $(\lambda x. ((x^2)^{\wedge}(k \text{ div } 2 + 1) / \exp(x^2)) * (1 / x) :: real) = ?M k$ 

```

```

+ 1)
  using ⟨even k⟩ by (auto simp: fun-eq-iff exp-minus field-simps power2-eq-square
power-mult elim: evenE)
  finally show ?thesis by simp
next
  assume odd k
  have ((λx. ((x2)∧((k - 1) div 2 + 1) / exp (x2)) :: real)  $\longrightarrow$  0) at-top
  by (intro filterlim-compose[OF tendsto-power-div-exp-0] filterlim-at-top-imp-at-infinity
filterlim-ident filterlim-pow-at-top)
    auto
  also have (λx. ((x2)∧((k - 1) div 2 + 1) / exp (x2)) :: real) = ?M (k + 1)
  using ⟨odd k⟩ by (auto simp: fun-eq-iff exp-minus field-simps power2-eq-square
power-mult elim: oddE)
  finally show ?thesis by simp
qed
qed
  also have ?tendsto  $\longleftrightarrow$  ((?f  $\longrightarrow$  (k + 1) / 2 * (∫ x. indicator {0..} x *R
?M k x ∂lborel) - 0 / 2) at-top)
  proof (intro filterlim-cong refl eventually-at-top-linorder[THEN iffD2] exI[of -
0] allI impI)
    fix b :: real assume b: 0 ≤ b
    have Suc k * (∫ x. indicator {0..b} x *R ?M k x ∂lborel) = (∫ x. indicator
{0..b} x *R (exp (- x2) * ((Suc k) * x^k)) ∂lborel)
    unfolding integral-mult-right-zero[symmetric] by (intro Bochner-Integration.integral-cong)
    auto
    also have ... = exp (- b2) * b^(Suc k) - exp (- 02) * 0^(Suc k) -
      (∫ x. indicator {0..b} x *R (- 2 * x * exp (- x2) * x^(Suc k)) ∂lborel)
    by (rule integral-by-parts')
    (auto intro!: derivative-eq-intros b
simp: diff-Suc of-nat-Suc field-simps split: nat.split)
    also have ... = exp (- b2) * b^(Suc k) - (∫ x. indicator {0..b} x *R (- 2
* (exp (- x2) * x^(k + 2))) ∂lborel)
    by (auto simp: intro!: Bochner-Integration.integral-cong)
    also have ... = exp (- b2) * b^(Suc k) + 2 * (∫ x. indicator {0..b} x *R
?M (k + 2) x ∂lborel)
    unfolding Bochner-Integration.integral-mult-right-zero [symmetric]
    by (simp del: real-scaleR-def)
    finally have Suc k * (∫ x. indicator {0..b} x *R ?M k x ∂lborel) =
      exp (- b2) * b^(Suc k) + 2 * (∫ x. indicator {0..b} x *R ?M (k + 2) x
∂lborel) .
    then show (k + 1) / 2 * (∫ x. indicator {..b} x *R (indicator {0..} x *R ?M
k x) ∂lborel) - exp (- b2) * b^(k + 1) / 2 = ?f b
    by (simp add: field-simps atLeastAtMost-def indicator-inter-arith)
  qed
  finally have int-M-at-top: ((?f  $\longrightarrow$  (k + 1) / 2 * (∫ x. indicator {0..} x *R
?M k x ∂lborel)) at-top)
  by simp
  have has-bochner-integral lborel (λx. indicator {0..} x *R ?M (k + 2) x) ((k

```

```

+ 1) / 2 * I)
  proof (rule has-bochner-integral-monotone-convergence-at-top)
    fix y :: real
    have *: (λx. indicator {0..} x *R ?M (k + 2) x * indicator {..y} x::real) =
      (λx. indicator {0..y} x *R ?M (k + 2) x)
    by rule (simp split: split-indicator)
    show integrable lborel (λx. indicator {0..} x *R (?M (k + 2) x) * indicator
      {..y} x::real)
      unfolding * by (rule borel-integrable-compact) (auto intro!: continuous-intros)
    show ((?f ⟶ (k + 1) / 2 * I) at-top)
      using int-M-at-top has-bochner-integral-integral-eq[OF Mk] by simp
    qed (auto split: split-indicator) }
  note step = this

show ?even
proof (induct k)
  case (Suc k)
  note step[OF this]
  also have (real (2 * k + 1) / 2 * (sqrt pi / 2 * ((fact (2 * k)) / ((2::real)^(2*k)
    * fact k)))) =
    sqrt pi / 2 * ((fact (2 * Suc k)) / ((2::real)^(2 * Suc k) * fact (Suc k)))
  apply (simp add: field-simps del: fact-Suc)
  apply (simp add: of-nat-mult field-simps)
  done
  finally show ?case
    by simp
qed (insert gaussian-moment-0, simp)

show ?odd
proof (induct k)
  case (Suc k)
  note step[OF this]
  also have (real (2 * k + 1 + 1) / (2::real) * ((fact k) / 2)) = (fact (Suc k))
    / 2
  by (simp add: field-simps of-nat-Suc field-split-simps del: fact-Suc) (simp add:
    field-simps)
  finally show ?case
    by simp
qed (insert gaussian-moment-1, simp)
qed

context
  fixes k :: nat and μ σ :: real assumes [arith]: 0 < σ
begin

lemma normal-moment-even:
  has-bochner-integral lborel (λx. normal-density μ σ x * (x - μ)^(2 * k)) (fact
    (2 * k) / ((2 / σ2)^k * fact k))

```

proof –

have $eq: \bigwedge x::real. x^2 \frown k = (x \frown k)^2$
by (*simp add: power-mult[symmetric] ac-simps*)

have *has-bochner-integral lborel* ($\lambda x. \exp(-x^2) * x \frown (2 * k)$)
 $(\text{sqrt } \pi * (\text{fact } (2 * k) / (2 \frown (2 * k) * \text{fact } k)))$
using *has-bochner-integral-even-function* [*OF gaussian-moment-even-pos* [**where**
 $k=k$]] **by** *simp*
then have *has-bochner-integral lborel* ($\lambda x. (\exp(-x^2) * x \frown (2 * k)) * ((2 * \sigma^2) \frown k /$
 $\text{sqrt } (2 * \pi * \sigma^2)))$
 $((\text{sqrt } \pi * (\text{fact } (2 * k) / (2 \frown (2 * k) * \text{fact } k))) * ((2 * \sigma^2) \frown k / \text{sqrt } (2 * \pi$
 $* \sigma^2)))$
by (*rule has-bochner-integral-mult-left*)
also have ($\lambda x. (\exp(-x^2) * x \frown (2 * k)) * ((2 * \sigma^2) \frown k / \text{sqrt } (2 * \pi * \sigma^2))) =$
 $(\lambda x. \exp(-((\text{sqrt } 2 * \sigma) * x)^2 / (2 * \sigma^2)) * ((\text{sqrt } 2 * \sigma) * x) \frown (2 * k) / \text{sqrt}$
 $(2 * \pi * \sigma^2)))$
by (*auto simp: fun-eq-iff field-simps real-sqrt-power[symmetric] real-sqrt-mult*
real-sqrt-divide power-mult eq)
also have $((\text{sqrt } \pi * (\text{fact } (2 * k) / (2 \frown (2 * k) * \text{fact } k))) * ((2 * \sigma^2) \frown k / \text{sqrt}$
 $(2 * \pi * \sigma^2))) =$
 $(\text{inverse } (\text{sqrt } 2 * \sigma) * ((\text{fact } (2 * k))) / ((2 / \sigma^2) \frown k * (\text{fact } k)))$
by (*auto simp: fun-eq-iff power-mult field-simps real-sqrt-power[symmetric]*
real-sqrt-mult
power2-eq-square)
finally show *?thesis*
unfolding *normal-density-def*
by (*subst lborel-has-bochner-integral-real-affine-iff* [**where** $c=\text{sqrt } 2 * \sigma$ and
 $t=\mu$]) *simp-all*
qed

lemma *normal-moment-abs-odd:*

has-bochner-integral lborel ($\lambda x. \text{normal-density } \mu \sigma x * |x - \mu| \frown (2 * k + 1)$) $(2 \frown k$
 $* \sigma \frown (2 * k + 1) * \text{fact } k * \text{sqrt } (2 / \pi))$

proof –

have *has-bochner-integral lborel* ($\lambda x::real. \text{indicator } \{0..\} x * \text{sqrt } 2 * (\exp(-x^2) * |x| \frown (2$
 $* k + 1)))$ (*fact k / 2*)
by (*rule has-bochner-integral-cong* [*THEN iffD1*, *OF* - - - *gaussian-moment-odd-pos* [*of*
 k]] *auto*)
from *has-bochner-integral-even-function* [*OF this*]
have *has-bochner-integral lborel* ($\lambda x::real. \exp(-x^2) * |x| \frown (2 * k + 1)$) (*fact k*)
by *simp*
then have *has-bochner-integral lborel* ($\lambda x. (\exp(-x^2) * |x| \frown (2 * k + 1)) * (2 \frown k$
 $* \sigma \frown (2 * k + 1) / \text{sqrt } (\pi * \sigma^2)))$
 $(\text{fact } k * (2 \frown k * \sigma \frown (2 * k + 1) / \text{sqrt } (\pi * \sigma^2)))$
by (*rule has-bochner-integral-mult-left*)
also have ($\lambda x. (\exp(-x^2) * |x| \frown (2 * k + 1)) * (2 \frown k * \sigma \frown (2 * k + 1) / \text{sqrt } (\pi$
 $* \sigma^2))) =$
 $(\lambda x. \exp(-(((\text{sqrt } 2 * \sigma) * x)^2 / (2 * \sigma^2))) * |\text{sqrt } 2 * \sigma * x| \frown (2 * k + 1)$
 $/ \text{sqrt } (2 * \pi * \sigma^2)))$

by (*simp add: field-simps abs-mult real-sqrt-power[symmetric] power-mult real-sqrt-mult*)
also have ($\text{fact } k * (2^k * \sigma^{2 * k + 1}) / \text{sqrt } (\pi * \sigma^2)) =$
 $(\text{inverse } (\text{sqrt } 2) * \text{inverse } \sigma * (2^k * (\sigma * \sigma^{2 * k})) * (\text{fact } k) * \text{sqrt } (2 /$
 $\pi))$)
by (*auto simp: fun-eq-iff power-mult field-simps real-sqrt-power[symmetric]*
real-sqrt-divide
real-sqrt-mult)
finally show ?thesis
unfolding normal-density-def
by (*subst lborel-has-bochner-integral-real-affine-iff[where c=sqrt 2 * sigma and*
t=mu])
simp-all
qed

lemma normal-moment-odd:

has-bochner-integral lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{2 * k + 1}$) 0
proof –
have *has-bochner-integral lborel* ($\lambda x. \exp(-x^2) * x^{2 * k + 1} :: \text{real}$) 0
using gaussian-moment-odd-pos **by** (*rule has-bochner-integral-odd-function*)
simp
then have *has-bochner-integral lborel* ($\lambda x. (\exp(-x^2) * x^{2 * k + 1}) * (2^k * \sigma^{2 * k}) / \text{sqrt}$
 π)
 $(0 * (2^k * \sigma^{2 * k}) / \text{sqrt } \pi))$
by (*rule has-bochner-integral-mult-left*)
also have ($\lambda x. (\exp(-x^2) * x^{2 * k + 1}) * (2^k * \sigma^{2 * k}) / \text{sqrt } \pi$) =
 $(\lambda x. \exp(-((\text{sqrt } 2 * \sigma * x)^2 / (2 * \sigma^2))) *$
 $(\text{sqrt } 2 * \sigma * x * (\text{sqrt } 2 * \sigma * x)^{2 * k}) /$
 $\text{sqrt } (2 * \pi * \sigma^2))$
unfolding real-sqrt-mult
by (*simp add: field-simps abs-mult real-sqrt-power[symmetric] power-mult fun-eq-iff*)
finally show ?thesis
unfolding normal-density-def
by (*subst lborel-has-bochner-integral-real-affine-iff[where c=sqrt 2 * sigma and*
t=mu]) *simp-all*
qed

lemma integral-normal-moment-even:

integral^L lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{2 * k}$) = $\text{fact } (2 * k) /$
 $((2 / \sigma^2)^k * \text{fact } k)$
using normal-moment-even **by** (*rule has-bochner-integral-integral-eq*)

lemma integral-normal-moment-abs-odd:

integral^L lborel ($\lambda x. \text{normal-density } \mu \sigma x * |x - \mu|^{2 * k + 1}$) = $2^k * \sigma^{2 * k + 1} * \text{fact } k * \text{sqrt } (2 / \pi)$
using normal-moment-abs-odd **by** (*rule has-bochner-integral-integral-eq*)

lemma integral-normal-moment-odd:

integral^L lborel ($\lambda x. \text{normal-density } \mu \sigma x * (x - \mu)^{2 * k + 1}$) = 0
using normal-moment-odd **by** (*rule has-bochner-integral-integral-eq*)

end

context

fixes $\sigma :: \text{real}$

assumes $\sigma\text{-pos}[\text{arith}]: 0 < \sigma$

begin

lemma *normal-moment-nz-1*: *has-bochner-integral lborel* $(\lambda x. \text{normal-density } \mu \sigma x * x) \mu$

proof –

note *normal-moment-even*[*OF* $\sigma\text{-pos}$, *of* μ 0]

note *normal-moment-odd*[*OF* $\sigma\text{-pos}$, *of* μ 0] *has-bochner-integral-mult-left*[*of* μ , *OF this*]

note *has-bochner-integral-add*[*OF this*]

then show ?thesis

by (*simp add: power2-eq-square field-simps*)

qed

lemma *integral-normal-moment-nz-1*:

integral^L *lborel* $(\lambda x. \text{normal-density } \mu \sigma x * x) = \mu$

using *normal-moment-nz-1* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integrable-normal-moment-nz-1*: *integrable lborel* $(\lambda x. \text{normal-density } \mu \sigma x * x)$

using *normal-moment-nz-1* **by** *rule*

lemma *integrable-normal-moment*: *integrable lborel* $(\lambda x. \text{normal-density } \mu \sigma x * (x - \mu) \frown k)$

proof *cases*

assume *even* k then show ?thesis

using *integrable.intros*[*OF normal-moment-even*] **by** (*auto elim: evenE*)

next

assume *odd* k then show ?thesis

using *integrable.intros*[*OF normal-moment-odd*] **by** (*auto elim: oddE*)

qed

lemma *integrable-normal-moment-abs*: *integrable lborel* $(\lambda x. \text{normal-density } \mu \sigma x * |x - \mu| \frown k)$

proof *cases*

assume *even* k then show ?thesis

using *integrable.intros*[*OF normal-moment-even*] **by** (*auto simp add: power-even-abs elim: evenE*)

next

assume *odd* k then show ?thesis

using *integrable.intros*[*OF normal-moment-abs-odd*] **by** (*auto elim: oddE*)

qed

lemma *integrable-normal-density*[simp, intro]: *integrable lborel (normal-density μ σ)*

using *integrable-normal-moment*[of μ 0] **by** *simp*

lemma *integral-normal-density*[simp]: $(\int x. \text{normal-density } \mu \sigma x \partial \text{lborel}) = 1$

using *integral-normal-moment-even*[of $\sigma \mu$ 0] **by** *simp*

lemma *prob-space-normal-density*:

prob-space (density lborel (normal-density μ σ))

proof qed (*simp add: emeasure-density nn-integral-eq-integral normal-density-nonneg*)

end

context

fixes $k :: \text{nat}$

begin

lemma *std-normal-moment-even*:

*has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k)}$) (fact (2 * k) / (2^k * fact k))*

using *normal-moment-even*[of 1 0 k] **by** *simp*

lemma *std-normal-moment-abs-odd*:

*has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * |x|^{(2 * k + 1)}$) (sqrt (2/pi) * 2^k * fact k)*

using *normal-moment-abs-odd*[of 1 0 k] **by** (*simp add: ac-simps*)

lemma *std-normal-moment-odd*:

*has-bochner-integral lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k + 1)}$) 0*

using *normal-moment-odd*[of 1 0 k] **by** *simp*

lemma *integral-std-normal-moment-even*:

*integral^L lborel ($\lambda x. \text{std-normal-density } x * x^{(2*k)}$) = fact (2 * k) / (2^k * fact k)*

using *std-normal-moment-even* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-std-normal-moment-abs-odd*:

*integral^L lborel ($\lambda x. \text{std-normal-density } x * |x|^{(2 * k + 1)}$) = sqrt (2 / pi) * 2^k * fact k*

using *std-normal-moment-abs-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-std-normal-moment-odd*:

*integral^L lborel ($\lambda x. \text{std-normal-density } x * x^{(2 * k + 1)}$) = 0*

using *std-normal-moment-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integrable-std-normal-moment-abs*: *integrable lborel ($\lambda x. \text{std-normal-density } x * |x|^k$)*

```

using integrable-normal-moment-abs[of 1 0 k] by simp

lemma integrable-std-normal-moment: integrable lborel ( $\lambda x. \text{std-normal-density } x$ 
 $\ast x^k$ )
  using integrable-normal-moment[of 1 0 k] by simp

end

lemma (in prob-space) normal-density-affine:
  assumes  $X$ : distributed  $M$  lborel  $X$  (normal-density  $\mu$   $\sigma$ )
  assumes [simp, arith]:  $0 < \sigma$   $\alpha \neq 0$ 
  shows distributed  $M$  lborel ( $\lambda x. \beta + \alpha \ast X$ ) (normal-density  $(\beta + \alpha \ast \mu)$  ( $|\alpha|$ 
 $\ast \sigma$ ))
proof –
  have eq:  $\bigwedge x. |\alpha| \ast \text{normal-density } (\beta + \alpha \ast \mu) (|\alpha| \ast \sigma) (x \ast \alpha + \beta) =$ 
    normal-density  $\mu$   $\sigma$   $x$ 
    by (simp add: normal-density-def real-sqrt-mult field-simps)
    (simp add: power2-eq-square field-simps)
  show ?thesis
    by (rule distributed-affineI[OF -  $\langle \alpha \neq 0 \rangle$ , where  $t = \beta$ ])
    (simp-all add: eq  $X$  ennreal-mult'[symmetric])
qed

lemma (in prob-space) normal-standard-normal-convert:
  assumes pos-var[simp, arith]:  $0 < \sigma$ 
  shows distributed  $M$  lborel  $X$  (normal-density  $\mu$   $\sigma$ ) = distributed  $M$  lborel ( $\lambda x.$ 
 $(X$   $x - \mu) / \sigma$ ) std-normal-density
proof auto
  assume distributed  $M$  lborel  $X$  ( $\lambda x. \text{ennreal } (\text{normal-density } \mu \sigma x)$ )
  then have distributed  $M$  lborel ( $\lambda x. -\mu / \sigma + (1/\sigma) \ast X$ ) ( $\lambda x. \text{ennreal } (\text{normal-density } (-\mu / \sigma + (1/\sigma) \ast \mu) (|1/\sigma| \ast \sigma) x)$ )
    by (rule normal-density-affine) auto

  then show distributed  $M$  lborel ( $\lambda x. (X$   $x - \mu) / \sigma$ ) ( $\lambda x. \text{ennreal } (\text{std-normal-density } x)$ )
    by (simp add: diff-divide-distrib[symmetric] field-simps)
next
  assume *: distributed  $M$  lborel ( $\lambda x. (X$   $x - \mu) / \sigma$ ) ( $\lambda x. \text{ennreal } (\text{std-normal-density } x)$ )
  have distributed  $M$  lborel ( $\lambda x. \mu + \sigma \ast ((X$   $x - \mu) / \sigma)$ ) ( $\lambda x. \text{ennreal } (\text{normal-density } \mu \sigma x)$ )
    using normal-density-affine[OF *, of  $\sigma$   $\mu$ ] by simp
  then show distributed  $M$  lborel  $X$  ( $\lambda x. \text{ennreal } (\text{normal-density } \mu \sigma x)$ ) by simp
qed

lemma conv-normal-density-zero-mean:
  assumes [simp, arith]:  $0 < \sigma$   $0 < \tau$ 
  shows ( $\lambda x. \int^+ y. \text{ennreal } (\text{normal-density } 0 \sigma (x - y) \ast \text{normal-density } 0 \tau y)$ 
 $\partial \text{lborel}$ ) =

```


$normal-density\ 0\ (\sqrt{\sigma^2 + \tau^2})\ (\text{is } ?LHS = ?RHS)$
proof –
 define $\sigma' \tau'$ where $\sigma' = \sigma^2$ and $\tau' = \tau^2$
 then have $[simp, arith]: 0 < \sigma' \ 0 < \tau'$
 by *simp-all*
 let $? \sigma = \sqrt{(\sigma' * \tau') / (\sigma' + \tau')}$
 have $\sqrt{(\sqrt{2 * \pi} * (\sigma' + \tau')) * \sqrt{2 * \pi} * (\sigma' * \tau') / (\sigma' + \tau')} =$
 $(\sqrt{2 * \pi} * \sigma') * \sqrt{2 * \pi} * \tau')$
 by (*subst power-eq-iff-eq-base[symmetric, where n=2]*)
 (*simp-all add: real-sqrt-mult[symmetric] power2-eq-square*)
 have $?LHS =$
 $(\lambda x. \int^+ y. ennreal((normal-density\ 0\ (\sqrt{\sigma' + \tau'})\ x) * normal-density\ (\tau' * x / (\sigma' + \tau'))\ ? \sigma\ y)\ \partial lborel)$
 apply (*intro ext nn-integral-cong*)
 apply (*simp add: normal-density-def \sigma'-def[symmetric] \tau'-def[symmetric] sqrt mult-exp-exp*)
 apply (*simp add: divide-simps power2-eq-square*)
 apply (*simp add: algebra-simps*)
 done

 also have ... =
 $(\lambda x. (normal-density\ 0\ (\sqrt{\sigma^2 + \tau^2})\ x) * \int^+ y. ennreal(normal-density\ (\tau^2 * x / (\sigma^2 + \tau^2))\ ? \sigma\ y)\ \partial lborel)$
 by (*subst nn-integral-cmult[symmetric]*)
 (*auto simp: \sigma'-def \tau'-def normal-density-def ennreal-mult'[symmetric]*)

 also have ... = $(\lambda x. (normal-density\ 0\ (\sqrt{\sigma^2 + \tau^2})\ x))$
 by (*subst nn-integral-eq-integral*) (*auto simp: normal-density-nonneg*)

 finally show *?thesis* by *fast*
qed

lemma *conv-std-normal-density:*

$(\lambda x. \int^+ y. ennreal\ (std-normal-density\ (x - y) * std-normal-density\ y)\ \partial lborel)$
 =
 $(normal-density\ 0\ (\sqrt{2}))$
 by (*subst conv-normal-density-zero-mean*) *simp-all*

lemma (*in prob-space*) *add-indep-normal:*

assumes *indep: indep-var borel X borel Y*
 assumes *pos-var[arith]: 0 < \sigma\ 0 < \tau*
 assumes *normalX[simp]: distributed M lborel X (normal-density \mu \sigma)*
 assumes *normalY[simp]: distributed M lborel Y (normal-density \nu \tau)*
 shows *distributed M lborel (\lambda x. X x + Y x) (normal-density (\mu + \nu) (\sqrt{\sigma^2 + \tau^2}))*
proof –
 have *ind[simp]: indep-var borel (\lambda x. - \mu + X x) borel (\lambda x. - \nu + Y x)*
proof –
 have *indep-var borel ((\lambda x. -\mu + x) o X) borel ((\lambda x. -\nu + x) o Y)*

```

    by (auto intro!: indep-var-compose assms)
    then show ?thesis by (simp add: o-def)
qed

have distributed M lborel ( $\lambda x. -\mu + 1 * X x$ ) (normal-density ( $-\mu + 1 * \mu$ )
(|1| *  $\sigma$ ))
  by(rule normal-density-affine[OF normalX pos-var(1), of 1 - $\mu$ ]) simp
then have 1[simp]: distributed M lborel ( $\lambda x. -\mu + X x$ ) (normal-density 0  $\sigma$ )
by simp

have distributed M lborel ( $\lambda x. -\nu + 1 * Y x$ ) (normal-density ( $-\nu + 1 * \nu$ )
(|1| *  $\tau$ ))
  by(rule normal-density-affine[OF normalY pos-var(2), of 1 - $\nu$ ]) simp
then have 2[simp]: distributed M lborel ( $\lambda x. -\nu + Y x$ ) (normal-density 0  $\tau$ )
by simp

have *: distributed M lborel ( $\lambda x. (-\mu + X x) + (-\nu + Y x)$ ) ( $\lambda x. ennreal$ 
(normal-density 0 ( $\sqrt{\sigma^2 + \tau^2}$ )  $x$ ))
  using distributed-convolution[OF ind 1 2] conv-normal-density-zero-mean[OF
pos-var]
  by (simp add: ennreal-mult'[symmetric] normal-density-nonneg)

have distributed M lborel ( $\lambda x. \mu + \nu + 1 * (-\mu + X x + (-\nu + Y x))$ )
  ( $\lambda x. ennreal$  (normal-density ( $\mu + \nu + 1 * 0$ ) ( $|1| * \sqrt{\sigma^2 + \tau^2}$ )  $x$ ))
  by (rule normal-density-affine[OF *, of 1  $\mu + \nu$ ]) (auto simp: add-pos-pos)

then show ?thesis by auto
qed

lemma (in prob-space) diff-indep-normal:
  assumes indep[simp]: indep-var borel X borel Y
  assumes [simp, arith]:  $0 < \sigma$   $0 < \tau$ 
  assumes normalX[simp]: distributed M lborel X (normal-density  $\mu$   $\sigma$ )
  assumes normalY[simp]: distributed M lborel Y (normal-density  $\nu$   $\tau$ )
  shows distributed M lborel ( $\lambda x. X x - Y x$ ) (normal-density ( $\mu - \nu$ ) ( $\sqrt{\sigma^2 + \tau^2}$ ))
proof -
  have distributed M lborel ( $\lambda x. 0 + -1 * Y x$ ) ( $\lambda x. ennreal$  (normal-density (0
+ -1 *  $\nu$ ) ( $|-1| * \tau$ )  $x$ ))
  by(rule normal-density-affine, auto)
  then have [simp]:distributed M lborel ( $\lambda x. -Y x$ ) ( $\lambda x. ennreal$  (normal-density
(- $\nu$ )  $\tau$   $x$ )) by simp

  have distributed M lborel ( $\lambda x. X x + (-Y x)$ ) (normal-density ( $\mu + -\nu$ ) ( $\sqrt{\sigma^2 + \tau^2}$ ))
  apply (rule add-indep-normal)
  apply (rule indep-var-compose[unfolded comp-def, of borel - borel -  $\lambda x. x - \lambda x.$ 
-  $x$ ])
  apply simp-all

```

lemma (in prob-space) normal-distributed-expectation:
assumes $\sigma[\text{arith}]$: $0 < \sigma$
assumes D : distributed M lborel X (normal-density μ σ)
shows expectation $X = \mu$
using integral-normal-moment-nz-1[OF σ , of μ] distributed-integral[OF D , of $\lambda x. x$, symmetric]
by (auto simp: field-simps)

lemma (in prob-space) normal-distributed-variance:

fixes $a\ b :: \text{real}$

assumes [simp, arith]: $0 < \sigma$

assumes D : distributed M lborel X (normal-density $\mu\ \sigma$)

shows variance $X = \sigma^2$

using integral-normal-moment-even[of $\sigma\ \mu\ 1$]

by (subst distributed-integral[OF D , symmetric])

(simp-all add: eval-nat-numeral normal-distributed-expectation[OF assms])

lemma (in prob-space) standard-normal-distributed-variance:

distributed M lborel X std-normal-density \implies variance $X = 1$

using normal-distributed-variance[of $1\ X\ 0$] by simp

lemma (in information-space) entropy-normal-density:

assumes [arith]: $0 < \sigma$

assumes D : distributed M lborel X (normal-density $\mu\ \sigma$)

shows entropy b lborel $X = \log b\ (2 * \pi * \exp 1 * \sigma^2) / 2$

proof –

have entropy b lborel $X = - (\int x. \text{normal-density } \mu\ \sigma\ x * \log b\ (\text{normal-density } \mu\ \sigma\ x)\ \partial \text{lborel})$

using D by (rule entropy-distr) simp

also have $\dots = - (\int x. \text{normal-density } \mu\ \sigma\ x * (- \ln (2 * \pi * \sigma^2) - (x - \mu)^2 / \sigma^2) / (2 * \ln b)\ \partial \text{lborel})$

by (intro arg-cong[where $f = \text{uminus}$] Bochner-Integration.integral-cong)

(auto simp: normal-density-def field-simps ln-mult log-def ln-div ln-sqrt)

also have $\dots = - (\int x. - (\text{normal-density } \mu\ \sigma\ x * (\ln (2 * \pi * \sigma^2)) + (\text{normal-density } \mu\ \sigma\ x * (x - \mu)^2) / \sigma^2) / (2 * \ln b)\ \partial \text{lborel})$

by (intro arg-cong[where $f = \text{uminus}$] Bochner-Integration.integral-cong) (auto simp: field-split-simps field-simps)

also have $\dots = (\int x. \text{normal-density } \mu\ \sigma\ x * (\ln (2 * \pi * \sigma^2)) + (\text{normal-density } \mu\ \sigma\ x * (x - \mu)^2) / \sigma^2\ \partial \text{lborel}) / (2 * \ln b)$

by (simp del: minus-add-distrib)

also have $\dots = (\ln (2 * \pi * \sigma^2) + 1) / (2 * \ln b)$

using integral-normal-moment-even[of $\sigma\ \mu\ 1$] by (simp add: integrable-normal-moment fact-numeral)

also have $\dots = \log b\ (2 * \pi * \exp 1 * \sigma^2) / 2$

by (simp add: log-def field-simps ln-mult)

finally show ?thesis .

qed

end

14 Characteristic Functions

theory Characteristic-Functions

imports Weak-Convergence Independent-Family Distributions

begin

lemma mult-min-right: $a \geq 0 \implies (a :: \text{real}) * \min b\ c = \min (a * b)\ (a * c)$

by (metis min.absorb-iff2 min-def mult-left-mono)

lemma *sequentially-even-odd*:

assumes E : eventually $(\lambda n. P (2 * n))$ sequentially and O : eventually $(\lambda n. P (2 * n + 1))$ sequentially

shows eventually P sequentially

proof –

from E obtain $n-e$ where [intro]: $\bigwedge n. n \geq n-e \implies P (2 * n)$

by (auto simp: eventually-sequentially)

moreover

from O obtain $n-o$ where [intro]: $\bigwedge n. n \geq n-o \implies P (Suc (2 * n))$

by (auto simp: eventually-sequentially)

show ?thesis

unfolding eventually-sequentially

proof (intro exI allI impI)

fix n assume $\max (2 * n-e) (2 * n-o + 1) \leq n$ then show $P n$

by (cases even n) (auto elim!: evenE oddE)

qed

qed

lemma *limseq-even-odd*:

assumes $(\lambda n. f (2 * n)) \longrightarrow (l :: 'a :: \text{topological-space})$

and $(\lambda n. f (2 * n + 1)) \longrightarrow l$

shows $f \longrightarrow l$

using assms by (auto simp: filterlim-iff intro: sequentially-even-odd)

14.1 Application of the FTC: integrating $e^i x$

abbreviation $iexp :: \text{real} \Rightarrow \text{complex}$ where

$iexp \equiv (\lambda x. exp (i * \text{complex-of-real } x))$

lemma *isCont-iexp* [simp]: *isCont* $iexp$ x

by (intro continuous-intros)

lemma *has-vector-derivative-iexp*[*derivative-intros*]:

$(iexp \text{ has-vector-derivative } i * iexp \ x) \text{ (at } x \text{ within } s)$

by (auto intro!: derivative-eq-intros simp: Re-exp Im-exp has-vector-derivative-complex-iff)

lemma *interval-integral-iexp*:

fixes $a \ b :: \text{real}$

shows $(CLBINT \ x=a..b. iexp \ x) = i * iexp \ a - i * iexp \ b$

by (subst interval-integral-FTC-finite [where $F = \lambda x. -i * iexp \ x$])

(auto intro!: derivative-eq-intros continuous-intros)

14.2 The Characteristic Function of a Real Measure.

definition

$char :: \text{real measure} \Rightarrow \text{real} \Rightarrow \text{complex}$

where $char \ M \ t \equiv CLINT \ x|M. iexp \ (t * x)$

lemma (in *real-distribution*) *char-zero*: $\text{char } M \ 0 = 1$
unfolding *char-def* **by** (*simp del: space-eq-univ add: prob-space*)

lemma (in *prob-space*) *integrable-iexp*:
assumes $f: f \in \text{borel-measurable } M \ \wedge x. \text{Im } (f \ x) = 0$
shows $\text{integrable } M \ (\lambda x. \exp (i * (f \ x)))$
proof (*intro integrable-const-bound [of - 1]*)
from f **have** $\wedge x. \text{of-real } (\text{Re } (f \ x)) = f \ x$
by (*simp add: complex-eq-iff*)
then show $\text{AE } x \text{ in } M. \text{cmod } (\exp (i * f \ x)) \leq 1$
using *norm-exp-i-times[of Re (f x) for x]* **by** *simp*
qed (*insert f, simp*)

lemma (in *real-distribution*) *cmod-char-le-1*: $\text{norm } (\text{char } M \ t) \leq 1$
proof –
have $\text{norm } (\text{char } M \ t) \leq (\int x. \text{norm } (iexp \ (t * x)) \ \partial M)$
unfolding *char-def* **by** (*intro integral-norm-bound*)
also have $\dots \leq 1$
by (*simp del: of-real-mult*)
finally show *?thesis* .
qed

lemma (in *real-distribution*) *isCont-char*: $\text{isCont } (\text{char } M) \ t$
unfolding *continuous-at-sequentially*
proof *safe*
fix X **assume** $X: X \longrightarrow t$
show $(\text{char } M \circ X) \longrightarrow \text{char } M \ t$
unfolding *comp-def char-def*
by (*rule integral-dominated-convergence[where w= $\lambda \cdot$. 1]*) (*auto intro!: tendsto-intros X*)
qed

lemma (in *real-distribution*) *char-measurable [measurable]*: $\text{char } M \in \text{borel-measurable borel}$
by (*auto intro!: borel-measurable-continuous-onI continuous-at-imp-continuous-on isCont-char*)

14.3 Independence

lemma (in *prob-space*) *char-distr-add*:
fixes $X1 \ X2 :: 'a \Rightarrow \text{real}$ **and** $t :: \text{real}$
assumes $\text{indep-var borel } X1 \ \text{borel } X2$
shows $\text{char } (\text{distr } M \ \text{borel } (\lambda \omega. X1 \ \omega + X2 \ \omega)) \ t =$
 $\text{char } (\text{distr } M \ \text{borel } X1) \ t * \text{char } (\text{distr } M \ \text{borel } X2) \ t$
proof –
from *assms* **have** *[measurable]: random-variable borel X1* **by** (*elim indep-var-rv1*)
from *assms* **have** *[measurable]: random-variable borel X2* **by** (*elim indep-var-rv2*)
have $\text{char } (\text{distr } M \ \text{borel } (\lambda \omega. X1 \ \omega + X2 \ \omega)) \ t = (\text{CLINT } x | M. iexp \ (t * (X1$

```

x + X2 x)))
  by (simp add: char-def integral-distr)
  also have ... = (CLINT x|M. iexp (t * (X1 x)) * iexp (t * (X2 x)))
    by (simp add: field-simps exp-add)
  also have ... = (CLINT x|M. iexp (t * (X1 x))) * (CLINT x|M. iexp (t * (X2
x)))
    by (intro indep-var-lebesgue-integral indep-var-compose[unfolded comp-def, OF
assms])
    (auto intro!: integrable-iexp)
  also have ... = char (distr M borel X1) t * char (distr M borel X2) t
    by (simp add: char-def integral-distr)
  finally show ?thesis .
qed

```

```

lemma (in prob-space) char-distr-sum:
  indep-vars ( $\lambda i$ . borel) X A  $\implies$ 
  char (distr M borel ( $\lambda \omega$ .  $\sum_{i \in A} X i \omega$ )) t = ( $\prod_{i \in A}$ . char (distr M borel (X
i)) t)
proof (induct A rule: infinite-finite-induct)
  case (insert x F) with indep-vars-subset[of  $\lambda$ -. borel X insert x F F] show ?case
    by (auto simp add: char-distr-add indep-vars-sum)
qed (simp-all add: char-def integral-distr prob-space del: distr-const)

```

14.4 Approximations to e^{ix}

Proofs from Billingsley, page 343.

```

lemma CLBINT-I0c-power-mirror-iexp:
  fixes x :: real and n :: nat
  defines f s m  $\equiv$  complex-of-real ((x - s) ^ m)
  shows (CLBINT s=0..x. f s n * iexp s) =
    x ^ Suc n / Suc n + (i / Suc n) * (CLBINT s=0..x. f s (Suc n) * iexp s)
proof -
  have 1:
    (( $\lambda s$ . complex-of-real(-((x - s) ^ (Suc n) / (Suc n))) * iexp s)
      has-vector-derivative complex-of-real((x - s) ^ n) * iexp s + (i * iexp s) *
      complex-of-real(-((x - s) ^ (Suc n) / (Suc n))))
    (at s within A) for s A
  by (intro derivative-eq-intros) auto

  let ?F =  $\lambda s$ . complex-of-real(-((x - s) ^ (Suc n) / (Suc n))) * iexp s
  have x ^ (Suc n) / (Suc n) = (CLBINT s=0..x. (f s n * iexp s + (i * iexp s) *
- (f s (Suc n) / (Suc n)))) (is ?LHS = ?RHS)
proof -
  have ?RHS = (CLBINT s=0..x. (f s n * iexp s + (i * iexp s) *
    complex-of-real(-((x - s) ^ (Suc n) / (Suc n))))))
    by (cases 0  $\leq$  x) (auto intro!: simp: f-def[abs-def])
  also have ... = ?F x - ?F 0
    unfolding zero-ereal-def using 1
    by (intro interval-integral-FTC-finite)

```

```

      (auto simp: f-def add-nonneg-eq-0-iff complex-eq-iff
        intro!: continuous-at-imp-continuous-on continuous-intros)
    finally show ?thesis
      by auto
  qed
show ?thesis
  unfolding ⟨?LHS = ?RHS⟩ f-def interval-lebesgue-integral-mult-right [symmetric]
  by (subst interval-lebesgue-integral-add(2) [symmetric])
    (auto intro!: interval-integrable-isCont continuous-intros simp: zero-ereal-def
      complex-eq-iff)
qed

```

```

lemma iexp-eq1:
  fixes x :: real
  defines f s m ≡ complex-of-real ((x - s) ^ m)
  shows iexp x =
    (∑ k ≤ n. (i * x) ^ k / (fact k)) + ((i ^ (Suc n)) / (fact n)) * (CLBINT s=0..x.
      (f s n) * (iexp s)) (is ?P n)
  proof (induction n)
    show ?P 0
      by (auto simp add: field-simps interval-integral-iexp f-def zero-ereal-def)
  next
    fix n assume ih: ?P n
    have **: ∧ a b :: real. a = -b ⟷ a + b = 0
      by linarith
    have *: of-nat n * of-nat (fact n) ≠ - (of-nat (fact n)::complex)
      unfolding of-nat-mult[symmetric]
      by (simp add: complex-eq-iff ** of-nat-add[symmetric] del: of-nat-mult of-nat-fact
        of-nat-add)
    show ?P (Suc n)
      unfolding sum.atMost-Suc ih f-def CLBINT-I0c-power-mirror-iexp[of - n]
      by (simp add: divide-simps add-eq-0-iff *) (simp add: field-simps)
  qed

```

```

lemma iexp-eq2:
  fixes x :: real
  defines f s m ≡ complex-of-real ((x - s) ^ m)
  shows iexp x = (∑ k ≤ Suc n. (i * x) ^ k / fact k) + i ^ Suc n / fact n * (CLBINT
    s=0..x. f s n * (iexp s - 1))
  proof -
    have isCont-f: isCont (λs. f s n) x for n x
      by (auto simp: f-def)
    let ?F = λs. complex-of-real (-((x - s) ^ (Suc n) / real (Suc n)))
    have calc1: (CLBINT s=0..x. f s n * (iexp s - 1)) =
      (CLBINT s=0..x. f s n * iexp s) - (CLBINT s=0..x. f s n)
    unfolding zero-ereal-def
    by (subst interval-lebesgue-integral-diff(2) [symmetric])
      (simp-all add: interval-integrable-isCont isCont-f field-simps)
  qed

```



```

have calc2: (CLBINT s=0..x. f s n) = x^Suc n / Suc n
  unfolding zero-ereal-def
proof (subst interval-integral-FTC-finite [where a = 0 and b = x and f = λs.
f s n and F = ?F])
  show (?F has-vector-derivative f y n) (at y within {min 0 x..max 0 x}) for y
    unfolding f-def
    by (intro has-vector-derivative-of-real)
      (auto intro!: derivative-eq-intros simp del: power-Suc simp add: add-nonneg-eq-0-iff)
qed (auto intro: continuous-at-imp-continuous-on isCont-f)

have calc3: i^(Suc (Suc n)) / (fact (Suc n)) = (i^(Suc n) / (fact n)) * (i /
(Suc n))
  by (simp add: field-simps)

show ?thesis
  unfolding iexp-eq1 [where n = Suc n and x=x] calc1 calc2 calc3 unfolding
f-def
  by (subst CLBINT-I0c-power-mirror-iexp [where n = n]) auto
qed

lemma abs-LBINT-I0c-abs-power-diff:
|LBINT s=0..x. |(x - s)^n|| = |x^(Suc n) / (Suc n)|
proof -
have |LBINT s=0..x. |(x - s)^n|| = |LBINT s=0..x. (x - s)^n|
proof cases
  assume 0 ≤ x ∨ even n
  then have (LBINT s=0..x. |(x - s)^n|) = LBINT s=0..x. (x - s)^n
    by (auto simp add: zero-ereal-def power-even-abs power-abs min-absorb1
max-absorb2
      intro!: interval-integral-cong)
  then show ?thesis by simp
next
  assume ¬(0 ≤ x ∨ even n)
  then have (LBINT s=0..x. |(x - s)^n|) = LBINT s=0..x. -((x - s)^n)
    by (auto simp add: zero-ereal-def power-abs min-absorb1 max-absorb2
      ereal-min[symmetric] ereal-max[symmetric] power-minus-odd[symmetric]
      simp del: ereal-min ereal-max intro!: interval-integral-cong)
  also have ... = - (LBINT s=0..x. (x - s)^n)
    by (subst interval-lebesgue-integral-uminus, rule refl)
  finally show ?thesis by simp
qed

also have LBINT s=0..x. (x - s)^n = x^Suc n / Suc n
proof -
let ?F = λt. - ((x - t)^(Suc n) / Suc n)
have LBINT s=0..x. (x - s)^n = ?F x - ?F 0
  unfolding zero-ereal-def
  by (intro interval-integral-FTC-finite continuous-at-imp-continuous-on
      has-real-derivative-iff-has-vector-derivative[THEN iffD1])
    (auto simp del: power-Suc intro!: derivative-eq-intros simp add: add-nonneg-eq-0-iff)

```

also have $\dots = x^{\wedge}(\text{Suc } n) / (\text{Suc } n)$ by *simp*
 finally show *?thesis* .
 qed
 finally show *?thesis* .
 qed

lemma *iexp-approx1*: $\text{cmod } (iexp\ x - (\sum k \leq n. (i * x)^{\wedge}k / \text{fact } k)) \leq |x|^{\wedge}(\text{Suc } n) / \text{fact } (\text{Suc } n)$

proof –

have $iexp\ x - (\sum k \leq n. (i * x)^{\wedge}k / \text{fact } k) =$
 $((i^{\wedge}(\text{Suc } n)) / (\text{fact } n)) * (\text{CLBINT } s=0..x. (x - s)^{\wedge}n * (iexp\ s))$ (is *?t1 = ?t2*)
 by (*subst iexp-eq1 [of - n], simp add: field-simps*)
 then have $\text{cmod } (?t1) = \text{cmod } (?t2)$
 by *simp*
 also have $\dots = (1 / \text{of-nat } (\text{fact } n)) * \text{cmod } (\text{CLBINT } s=0..x. (x - s)^{\wedge}n * (iexp\ s))$
 by (*simp add: norm-mult norm-divide norm-power*)
 also have $\dots \leq (1 / \text{of-nat } (\text{fact } n)) * |\text{LBINT } s=0..x. \text{cmod } ((x - s)^{\wedge}n * (iexp\ s))|$
 by (*intro mult-left-mono interval-integral-norm2*)
 (*auto simp: zero-ereal-def intro: interval-integrable-isCont*)
 also have $\dots \leq (1 / \text{of-nat } (\text{fact } n)) * |\text{LBINT } s=0..x. |(x - s)^{\wedge}n||$
 by (*simp add: norm-mult del: of-real-diff of-real-power*)
 also have $\dots \leq (1 / \text{of-nat } (\text{fact } n)) * |x^{\wedge}(\text{Suc } n) / (\text{Suc } n)|$
 by (*simp add: abs-LBINT-I0c-abs-power-diff*)
 also have $1 / \text{real-of-nat } (\text{fact } n::\text{nat}) * |x^{\wedge}\text{Suc } n / \text{real } (\text{Suc } n)| =$
 $|x|^{\wedge}\text{Suc } n / \text{fact } (\text{Suc } n)$
 by (*simp add: abs-mult power-abs*)
 finally show *?thesis* .
 qed

lemma *iexp-approx2*: $\text{cmod } (iexp\ x - (\sum k \leq n. (i * x)^{\wedge}k / \text{fact } k)) \leq 2 * |x|^{\wedge}n / \text{fact } n$

proof (*induction n*) — really cases

case (*Suc n*)
 have *: $\bigwedge a\ b. \text{interval-lebesgue-integrable } lborel\ a\ b\ f \implies \text{interval-lebesgue-integrable } lborel\ a\ b\ g \implies$
 $|\text{LBINT } s=a..b. f\ s| \leq |\text{LBINT } s=a..b. g\ s|$
 if $f: \bigwedge s. 0 \leq f\ s$ and $g: \bigwedge s. f\ s \leq g\ s$ for $f\ g :: - \Rightarrow \text{real}$
 using *order-trans[OF f g] f g*
 unfolding *interval-lebesgue-integral-def interval-lebesgue-integrable-def set-lebesgue-integral-def set-integrable-def*
 by (*auto simp: integral-nonneg-AE[OF AE-I2] intro!: integral-mono mult-mono*)
 have $iexp\ x - (\sum k \leq \text{Suc } n. (i * x)^{\wedge}k / \text{fact } k) =$
 $((i^{\wedge}(\text{Suc } n)) / (\text{fact } n)) * (\text{CLBINT } s=0..x. (x - s)^{\wedge}n * (iexp\ s - 1))$ (is *?t1 = ?t2*)
 unfolding *iexp-eq2 [of x n]* by (*simp add: field-simps*)

then have $cmod (?t1) = cmod (?t2)$
by *simp*
also have $\dots = (1 / (fact\ n)) * cmod (CLBINT\ s=0..x. (x - s)^{\wedge}n * (iexp\ s - 1))$
by (*simp add: norm-mult norm-divide norm-power*)
also have $\dots \leq (1 / (fact\ n)) * |LBINT\ s=0..x. cmod ((x - s)^{\wedge}n * (iexp\ s - 1))|$
by (*intro mult-left-mono interval-integral-norm2*)
(auto intro!: interval-integrable-isCont simp: zero-ereal-def)
also have $\dots = (1 / (fact\ n)) * |LBINT\ s=0..x. abs ((x - s)^{\wedge}n) * cmod((iexp\ s - 1))|$
by (*simp add: norm-mult del: of-real-diff of-real-power*)
also have $\dots \leq (1 / (fact\ n)) * |LBINT\ s=0..x. abs ((x - s)^{\wedge}n) * 2|$
by (*intro mult-left-mono * order-trans [OF norm-triangle-ineq4]*)
(auto simp: mult-ac zero-ereal-def intro!: interval-integrable-isCont)
also have $\dots = (2 / (fact\ n)) * |x^{\wedge}(Suc\ n) / (Suc\ n)|$
by (*simp add: abs-LBINT-I0c-abs-power-diff abs-mult*)
also have $2 / fact\ n * |x^{\wedge}Suc\ n / real\ (Suc\ n)| = 2 * |x|^{\wedge}Suc\ n / (fact\ (Suc\ n))$
by (*simp add: abs-mult power-abs*)
finally show *?case .*
qed (*insert norm-triangle-ineq4[of iexp x 1], simp*)

lemma (*in real-distribution*) *char-approx1*:

assumes *integrable-moments*: $\bigwedge k. k \leq n \implies integrable\ M\ (\lambda x. x^{\wedge}k)$
shows $cmod (char\ M\ t - (\sum k \leq n. ((i * t)^{\wedge}k / fact\ k) * expectation\ (\lambda x. x^{\wedge}k))) \leq$
 $(2 * |t|^{\wedge}n / fact\ n) * expectation\ (\lambda x. |x|^{\wedge}n)$ (**is** $cmod (char\ M\ t - ?t1) \leq -$)

proof –

have *integ-iexp*: $integrable\ M\ (\lambda x. iexp\ (t * x))$
by (*intro integrable-const-bound auto*)

define *c* **where** [*abs-def*]: $c\ k\ x = (i * t)^{\wedge}k / fact\ k * complex-of-real\ (x^{\wedge}k)$ **for** $k\ x$

have *integ-c*: $\bigwedge k. k \leq n \implies integrable\ M\ (\lambda x. c\ k\ x)$
unfolding *c-def* **by** (*intro integrable-mult-right integrable-of-real integrable-moments*)

have $k \leq n \implies expectation\ (c\ k) = (i * t)^{\wedge}k * (expectation\ (\lambda x. x^{\wedge}k)) / fact\ k$ **for** k

unfolding *c-def* *integral-mult-right-zero integral-complex-of-real* **by** *simp*

then have $norm\ (char\ M\ t - ?t1) = norm\ (char\ M\ t - (CLINT\ x\ |\ M. (\sum k \leq n. c\ k\ x)))$

by (*simp add: integ-c*)

also have $\dots = norm\ ((CLINT\ x\ |\ M. iexp\ (t * x) - (\sum k \leq n. c\ k\ x)))$

unfolding *char-def* **by** (*subst Bochner-Integration.integral-diff[OF integ-iexp]*)
(auto intro!: integ-c)

also have $\dots \leq expectation\ (\lambda x. cmod\ (iexp\ (t * x) - (\sum k \leq n. c\ k\ x)))$

by (*intro integral-norm-bound*)

also have $\dots \leq expectation\ (\lambda x. 2 * |t|^{\wedge}n / fact\ n * |x|^{\wedge}n)$

```

proof (rule integral-mono)
  show integrable M ( $\lambda x. \text{cmod} (\text{iexp} (t * x) - (\sum_{k \leq n}. c \ k \ x)))$ 
  by (intro integrable-norm Bochner-Integration.integrable-diff integ-iexp Bochner-Integration.integrable-sum
integ-c) simp
  show integrable M ( $\lambda x. 2 * |t| \wedge n / \text{fact } n * |x| \wedge n$ )
  unfolding power-abs[symmetric]
  by (intro integrable-mult-right integrable-abs integrable-moments) simp
  show  $\text{cmod} (\text{iexp} (t * x) - (\sum_{k \leq n}. c \ k \ x)) \leq 2 * |t| \wedge n / \text{fact } n * |x| \wedge n$ 
for x
  using iexp-approx2[of t * x n] by (auto simp add: abs-mult field-simps c-def)
qed
finally show ?thesis
  unfolding integral-mult-right-zero .
qed

```

```

lemma (in real-distribution) char-approx2:
  assumes integrable-moments:  $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. x \wedge k)$ 
  shows  $\text{cmod} (\text{char } M \ t - (\sum_{k \leq n}. ((i * t) \wedge k / \text{fact } k) * \text{expectation } (\lambda x. x \wedge k)))$ 
   $\leq$ 
   $(|t| \wedge n / \text{fact } (\text{Suc } n)) * \text{expectation } (\lambda x. \min (2 * |x| \wedge n * \text{Suc } n) (|t| * |x| \wedge \text{Suc } n))$ 
   $(\text{is } \text{cmod} (\text{char } M \ t - ?t1) \leq -)$ 

```

```

proof -
  have integ-iexp: integrable M ( $\lambda x. \text{iexp} (t * x)$ )
  by (intro integrable-const-bound) auto

  define c where [abs-def]:  $c \ k \ x = (i * t) \wedge k / \text{fact } k * \text{complex-of-real } (x \wedge k)$  for
  k x
  have integ-c:  $\bigwedge k. k \leq n \implies \text{integrable } M (\lambda x. c \ k \ x)$ 
  unfolding c-def by (intro integrable-mult-right integrable-of-real integrable-moments)

  have *:  $\min (2 * |t * x| \wedge n / \text{fact } n) (|t * x| \wedge \text{Suc } n / \text{fact } (\text{Suc } n)) =$ 
   $|t| \wedge n / \text{fact } (\text{Suc } n) * \min (2 * |x| \wedge n * \text{real } (\text{Suc } n)) (|t| * |x| \wedge (\text{Suc } n))$  for x
  apply (subst mult-min-right)
  apply simp
  apply (rule arg-cong2[where f=min])
  apply (simp-all add: field-simps abs-mult del: fact-Suc)
  apply (simp-all add: field-simps)
  done

```

```

  have  $k \leq n \implies \text{expectation } (c \ k) = (i * t) \wedge k * (\text{expectation } (\lambda x. x \wedge k)) / \text{fact } k$ 
  for k
  unfolding c-def integral-mult-right-zero integral-complex-of-real by simp
  then have norm (char M t - ?t1) = norm (char M t - (CLINT x | M.  $(\sum_{k \leq n}. c \ k \ x)$ ))
   $\leq$  norm (c k x))
  by (simp add: integ-c)
  also have  $\dots = \text{norm } ((\text{CLINT } x \mid M. \text{iexp} (t * x) - (\sum_{k \leq n}. c \ k \ x)))$ 
  unfolding char-def by (subst Bochner-Integration.integral-diff[OF integ-iexp])
  (auto intro!: integ-c)

```

```

also have ...  $\leq$  expectation ( $\lambda x. \text{cmod } (\text{iecp } (t * x) - (\sum k \leq n. c \ k \ x)))$ 
  by (rule integral-norm-bound)
also have ...  $\leq$  expectation ( $\lambda x. \min (2 * |t * x|^n / \text{fact } n) (|t * x|^{\wedge} (\text{Suc } n) / \text{fact } (\text{Suc } n)))$ 
  (is -  $\leq$  expectation ?f)
proof (rule integral-mono)
  show integrable M ( $\lambda x. \text{cmod } (\text{iecp } (t * x) - (\sum k \leq n. c \ k \ x)))$ 
  by (intro integrable-norm Bochner-Integration.integrable-diff integ-iecp Bochner-Integration.integrable-sum
integ-c) simp
  show integrable M ?f
  by (rule Bochner-Integration.integrable-bound[where  $f = \lambda x. 2 * |t * x|^n / \text{fact } n$ ])
    (auto simp: integrable-moments power-abs[symmetric] power-mult-distrib)
  show  $\text{cmod } (\text{iecp } (t * x) - (\sum k \leq n. c \ k \ x)) \leq ?f \ x$  for  $x$ 
    using iexp-approx1[of  $t * x \ n$ ] iexp-approx2[of  $t * x \ n$ ]
    by (auto simp add: abs-mult field-simps c-def intro!: min.boundedI)
qed
also have ...  $= (|t|^n / \text{fact } (\text{Suc } n)) * \text{expectation } (\lambda x. \min (2 * |x|^n * \text{Suc } n) (|t| * |x|^{\wedge} \text{Suc } n))$ 
  unfolding *
proof (rule Bochner-Integration.integral-mult-right)
  show integrable M ( $\lambda x. \min (2 * |x|^n * \text{real } (\text{Suc } n)) (|t| * |x|^{\wedge} \text{Suc } n)$ )
  by (rule Bochner-Integration.integrable-bound[where  $f = \lambda x. 2 * |x|^n * \text{real } (\text{Suc } n)$ ])
    (auto simp: integrable-moments power-abs[symmetric] power-mult-distrib)
qed
finally show ?thesis
  unfolding integral-mult-right-zero .
qed

```

lemma (in real-distribution) char-approx3:

```

fixes  $t$ 
assumes
  integrable-1: integrable M ( $\lambda x. x$ ) and
  integral-1: expectation ( $\lambda x. x$ ) = 0 and
  integrable-2: integrable M ( $\lambda x. x^2$ ) and
  integral-2: variance ( $\lambda x. x$ ) =  $\sigma^2$ 
shows  $\text{cmod } (\text{char } M \ t - (1 - t^2 * \sigma^2 / 2)) \leq$ 
   $(t^2 / 6) * \text{expectation } (\lambda x. \min (6 * x^2) (\text{abs } t * (\text{abs } x)^3))$ 
proof -
  note real-distribution.char-approx2 [of  $M \ 2 \ t$ , simplified]
  have [simp]: prob UNIV = 1 by (metis prob-space space-eq-univ)
  from integral-2 have [simp]: expectation ( $\lambda x. x * x$ ) =  $\sigma^2$ 
    by (simp add: integral-1 numeral-eq-Suc)
  have  $1: k \leq 2 \implies \text{integrable } M (\lambda x. x^k)$  for  $k$ 
    using assms by (auto simp: eval-nat-numeral le-Suc-eq)
  note char-approx1
  note  $2 = \text{char-approx1}$  [of  $2 \ t$ , OF 1, simplified]
  have  $\text{cmod } (\text{char } M \ t - (\sum k \leq 2. (i * t)^{\wedge} k * (\text{expectation } (\lambda x. x^{\wedge} k)) / (\text{fact } k)))$ 

```

```

k))) ≤
  t2 * expectation (λx. min (6 * x2) (|t| * |x| ^ 3)) / fact (3::nat)
  using char-approx2 [of 2 t, OF 1] by simp
  also have (∑ k ≤ 2. (i * t) ^ k * expectation (λx. x ^ k) / (fact k)) = 1 - t2
  * σ2 / 2
  by (simp add: complex-eq-iff numeral-eq-Suc integral-1 Re-divide Im-divide)
  also have fact 3 = 6 by (simp add: eval-nat-numeral)
  also have t2 * expectation (λx. min (6 * x2) (|t| * |x| ^ 3)) / 6 =
    t2 / 6 * expectation (λx. min (6 * x2) (|t| * |x| ^ 3)) by (simp add: field-simps)
  finally show ?thesis .
qed

```

This is a more familiar textbook formulation in terms of random variables, but we will use the previous version for the CLT.

```

lemma (in prob-space) char-approx3':
  fixes μ :: real measure and X
  assumes rv-X [simp]: random-variable borel X
    and [simp]: integrable M X integrable M (λx. (X x)^2) expectation X = 0
    and var-X: variance X = σ2
    and μ-def: μ = distr M borel X
  shows cmod (char μ t - (1 - t2 * σ2 / 2)) ≤
    (t2 / 6) * expectation (λx. min (6 * (X x)^2) (|t| * |X x|^3))
  using var-X unfolding μ-def
  apply (subst integral-distr [symmetric, OF rv-X], simp)
  apply (intro real-distribution.char-approx3)
  apply (auto simp add: integrable-distr-eq integral-distr)
  done

```

this is the formulation in the book – in terms of a random variable *with* the distribution, rather the distribution itself. I don’t know which is more useful, though in principal we can go back and forth between them.

```

lemma (in prob-space) char-approx1':
  fixes μ :: real measure and X
  assumes integrable-moments : ∧k. k ≤ n ⇒ integrable M (λx. X x ^ k)
    and rv-X[measurable]: random-variable borel X
    and μ-distr : distr M borel X = μ
  shows cmod (char μ t - (∑ k ≤ n. ((i * t) ^ k / fact k) * expectation (λx. (X
x) ^ k))) ≤
    (2 * |t| ^ n / fact n) * expectation (λx. |X x| ^ n)
  unfolding μ-distr[symmetric]
  apply (subst (1 2) integral-distr [symmetric, OF rv-X], simp, simp)
  apply (intro real-distribution.char-approx1 [of distr M borel X n t] real-distribution-distr
rv-X)
  apply (auto simp: integrable-distr-eq integrable-moments)
  done

```

14.5 Calculation of the Characteristic Function of the Standard Distribution

abbreviation

std-normal-distribution \equiv *density lborel std-normal-density*

lemma *real-dist-normal-dist: real-distribution std-normal-distribution*

using *prob-space-normal-density* **by** (*auto simp: real-distribution-def real-distribution-axioms-def*)

lemma *std-normal-distribution-even-moments:*

fixes $k :: \text{nat}$

shows (*LINT* $x \mid \text{std-normal-distribution}. x^{(2 * k)} = \text{fact } (2 * k) / (2^k * \text{fact } k)$)

and *integrable std-normal-distribution* ($\lambda x. x^{(2 * k)}$)

using *integral-std-normal-moment-even*[*of k*]

by (*subst integral-density*)

(*auto simp: normal-density-nonneg integrable-density*

intro: integrable.intros std-normal-moment-even)

lemma *integrable-std-normal-distribution-moment: integrable std-normal-distribution*

($\lambda x. x^k$)

by (*auto simp: normal-density-nonneg integrable-std-normal-moment integrable-density*)

lemma *integral-std-normal-distribution-moment-odd:*

odd k \implies *integral^L std-normal-distribution* ($\lambda x. x^k$) = 0

using *integral-std-normal-moment-odd*[*of (k - 1) div 2*]

by (*auto simp: integral-density normal-density-nonneg elim: oddE*)

lemma *std-normal-distribution-even-moments-abs:*

fixes $k :: \text{nat}$

shows (*LINT* $x \mid \text{std-normal-distribution}. |x|^{(2 * k)} = \text{fact } (2 * k) / (2^k * \text{fact } k)$)

using *integral-std-normal-moment-even*[*of k*]

by (*subst integral-density*) (*auto simp: normal-density-nonneg power-even-abs*)

lemma *std-normal-distribution-odd-moments-abs:*

fixes $k :: \text{nat}$

shows (*LINT* $x \mid \text{std-normal-distribution}. |x|^{(2 * k + 1)} = \text{sqrt } (2 / \text{pi}) * 2^k * \text{fact } k$)

using *integral-std-normal-moment-abs-odd*[*of k*]

by (*subst integral-density*) (*auto simp: normal-density-nonneg*)

theorem *char-std-normal-distribution:*

char std-normal-distribution = ($\lambda t. \text{complex-of-real } (\exp (- (t^2) / 2)))$)

proof (*intro ext LIMSEQ-unique*)

fix $t :: \text{real}$

let $?f' = \lambda k. (i * t)^k / \text{fact } k * (\text{LINT } x \mid \text{std-normal-distribution}. x^k)$

let $?f = \lambda n. (\sum k \leq n. ?f' k)$

show $?f \longrightarrow \exp (- (t^2) / 2)$

```

proof (rule limseq-even-odd)
  have (i * complex-of-real t) ^ (2 * a) / (2 ^ a * fact a) = (- ((complex-of-real
t)^2 / 2)) ^ a / fact a for a
    by (subst power-mult) (simp add: field-simps uminus-power-if power-mult)
  then have *: ?f (2 * n) = complex-of-real (∑ k < Suc n. (1 / fact k) * (-
(t^2) / 2) ^ k) for n :: nat
    unfolding of-real-sum
    by (intro sum.reindex-bij-witness-not-neutral[symmetric, where
      i=λn. n div 2 and j=λn. 2 * n and T'={i. i ≤ 2 * n ∧ odd i} and
      S'={}))
    (auto simp: integral-std-normal-distribution-moment-odd std-normal-distribution-even-moments)
  show (λn. ?f (2 * n)) ⟶ exp (-(t^2) / 2)
    unfolding * using exp-converges[where 'a=real]
    by (intro tendsto-of-real LIMSEQ-Suc) (auto simp: inverse-eq-divide sums-def
[symmetric])
  have **: ?f (2 * n + 1) = ?f (2 * n) for n
    proof -
      have ?f (2 * n + 1) = ?f (2 * n) + ?f' (2 * n + 1)
        by simp
      also have ?f' (2 * n + 1) = 0
        by (subst integral-std-normal-distribution-moment-odd) simp-all
      finally show ?f (2 * n + 1) = ?f (2 * n)
        by simp
    qed
  show (λn. ?f (2 * n + 1)) ⟶ exp (-(t^2) / 2)
    unfolding ** by fact
  qed

have **: (λn. x ^ n / fact n) ⟶ 0 for x :: real
  using summable-LIMSEQ-zero [OF summable-exp] by (auto simp add: in-
verse-eq-divide)

let ?F = λn. 2 * |t| ^ n / fact n * (LINT x|std-normal-distribution. |x| ^ n)

show ?f ⟶ char std-normal-distribution t
proof (rule metric-tendsto-imp-tendsto[OF limseq-even-odd])
  show (λn. ?F (2 * n)) ⟶ 0
    proof (rule Lim-transform-eventually)
      show ∀F n in sequentially. 2 * ((t^2 / 2) ^ n / fact n) = ?F (2 * n)
        unfolding std-normal-distribution-even-moments-abs by (simp add: power-mult
power-divide)
      qed (intro tendsto-mult-right-zero **)

  have *: ?F (2 * n + 1) = (2 * |t| * sqrt (2 / pi)) * ((2 * t^2) ^ n * fact n /
fact (2 * n + 1)) for n
    unfolding std-normal-distribution-odd-moments-abs
    by (simp add: field-simps power-mult[symmetric] power-even-abs)
  have norm ((2 * t^2) ^ n * fact n / fact (2 * n + 1)) ≤ (2 * t^2) ^ n / fact n
for n

```



```

    using mult-mono[OF - square-fact-le-2-fact, of 1 1 + 2 * real n n]
    by (auto simp add: divide-simps intro!: mult-left-mono)
  then show (λn. ?F (2 * n + 1)) ⟶ 0
    unfolding * by (intro tendsto-mult-right-zero Lim-null-comparison [OF - **
[of 2 * t2]]) auto

  show ∀F n in sequentially. dist (?f n) (char std-normal-distribution t) ≤ dist
(?F n) 0
    using real-distribution.char-approx1[OF real-dist-normal-dist integrable-std-normal-distribution-moment]
    by (auto simp: dist-norm integral-nonneg-AE norm-minus-commute)
qed
qed

end

```

15 Helly’s selection theorem

The set of bounded, monotone, right continuous functions is sequentially compact

theory *Helly-Selection*

imports *HOL-Library.Diagonal-Subsequence Weak-Convergence*
begin

lemma *minus-one-less*: $x - 1 < (x::real)$
by *simp*

theorem *Helly-selection*:

fixes $f :: nat \Rightarrow real \Rightarrow real$
assumes *rcont*: $\bigwedge n x. continuous (at-right\ x) (f\ n)$
assumes *mono*: $\bigwedge n. mono (f\ n)$
assumes *bdd*: $\bigwedge n x. |f\ n\ x| \leq M$
shows $\exists s. strict-mono (s::nat \Rightarrow nat) \wedge (\exists F. (\forall x. continuous (at-right\ x) F) \wedge$
 $mono\ F \wedge (\forall x. |F\ x| \leq M) \wedge$
 $(\forall x. continuous (at\ x) F \longrightarrow (\lambda n. f\ (s\ n)\ x) \longrightarrow F\ x))$

proof –

obtain $m :: real \Rightarrow nat$ **where** *bij-betw* m *Q UNIV*
using *countable-rat Rats-infinite* **by** (*erule countableE-infinite*)
then obtain $r :: nat \Rightarrow real$ **where** *bij*: *bij-betw* r *UNIV Q*
using *bij-betw-inv* **by** *blast*

have *dense-r*: $\bigwedge x y. x < y \implies \exists n. x < r\ n \wedge r\ n < y$
by (*metis Rats-dense-in-real bij f-the-inv-into-f bij-betw-def*)

let $?P = \lambda n. \lambda s. convergent (\lambda k. f\ (s\ k)\ (r\ n))$

interpret *nat*: *subseqs* $?P$

proof (*unfold convergent-def, unfold subseqs-def, auto*)

fix $n :: nat$ **and** $s :: nat \Rightarrow nat$ **assume** s : *strict-mono* s

have *bounded* $\{-M..M\}$

```

    using bounded-closed-interval by auto
  moreover have  $\bigwedge k. f (s k) (r n) \in \{-M..M\}$ 
    using bdd by (simp add: abs-le-iff minus-le-iff)
  ultimately have  $\exists l s'. \text{strict-mono } s' \wedge ((\lambda k. f (s k) (r n)) \circ s') \longrightarrow l$ 
    using compact-Icc compact-imp-seq-compact seq-compactE by metis
  thus  $\exists s'. \text{strict-mono } (s'::\text{nat} \Rightarrow \text{nat}) \wedge (\exists l. (\lambda k. f (s (s' k)) (r n)) \longrightarrow l)$ 
    by (auto simp: comp-def)
qed
define d where  $d = \text{nat.diagseq}$ 
have subseq:  $\text{strict-mono } d$ 
  unfolding d-def using nat.subseq-diagseq by auto
have rat-cnvt:  $?P n d$  for  $n$ 
proof -
  have Pn-seqseq:  $?P n (\text{nat.seqseq } (\text{Suc } n))$ 
    by (rule nat.seqseq-holds)
  have 1:  $(\lambda k. f ((\text{nat.seqseq } (\text{Suc } n) \circ (\lambda k. \text{nat.fold-reduce } (\text{Suc } n) k$ 
     $(\text{Suc } n + k))) k) (r n)) = (\lambda k. f (\text{nat.seqseq } (\text{Suc } n) k) (r n)) \circ$ 
     $(\lambda k. \text{nat.fold-reduce } (\text{Suc } n) k (\text{Suc } n + k))$ 
    by auto
  have 2:  $?P n (d \circ ((+) (\text{Suc } n)))$ 
    unfolding d-def nat.diagseq-seqseq 1
    by (intro convergent-subseq-convergent Pn-seqseq nat.subseq-diagonal-rest)
  then obtain L where  $\exists: (\lambda na. f (d (na + \text{Suc } n)) (r n)) \longrightarrow L$ 
    by (auto simp: add.commute dest: convergentD)
  then have  $(\lambda k. f (d k) (r n)) \longrightarrow L$ 
    by (rule LIMSEQ-offset)
  then show ?thesis
    by (auto simp: convergent-def)
qed
let ?f =  $\lambda n. \lambda k. f (d k) (r n)$ 
have lim-f:  $?f n \longrightarrow \text{lim } (?f n)$  for  $n$ 
  using rat-cnvt convergent-LIMSEQ-iff by auto
have lim-bdd:  $\text{lim } (?f n) \in \{-M..M\}$  for  $n$ 
proof -
  have closed  $\{-M..M\}$  using closed-real-atLeastAtMost by auto
  hence  $(\forall i. ?f n i \in \{-M..M\}) \wedge ?f n \longrightarrow \text{lim } (?f n) \longrightarrow \text{lim } (?f n) \in$ 
 $\{-M..M\}$ 
    unfolding closed-sequential-limits by (drule-tac  $x = \lambda k. f (d k) (r n)$  in spec)
blast
  moreover have  $\forall i. ?f n i \in \{-M..M\}$ 
    using bdd by (simp add: abs-le-iff minus-le-iff)
  ultimately show  $\text{lim } (?f n) \in \{-M..M\}$ 
    using lim-f by auto
qed
then have limset-bdd:  $\bigwedge x. \{\text{lim } (?f n) \mid n. x < r n\} \subseteq \{-M..M\}$ 
  by auto
then have bdd-below:  $\text{bdd-below } \{\text{lim } (?f n) \mid n. x < r n\}$  for  $x$ 
  by (metis (mono-tags) bdd-below-Icc bdd-below-mono)
have r-unbdd:  $\exists n. x < r n$  for  $x$ 

```

```

    using dense-r[OF less-add-one, of x] by auto
  then have nonempty:  $\{\lim (?f n) \mid n. x < r n\} \neq \{\}$  for x
    by auto

  define F where  $F x = \text{Inf } \{\lim (?f n) \mid n. x < r n\}$  for x
  have F-eq:  $\text{ereal } (F x) = (\text{INF } n \in \{n. x < r n\}. \text{ereal } (\lim (?f n)))$  for x
    unfolding F-def by (subst ereal-Inf'[OF bdd-below nonempty]) (simp add:
setcompr-eq-image image-comp)
  have mono-F: mono F
    using nonempty by (auto intro!: cInf-superset-mono simp: F-def bdd-below
mono-def)
  moreover have  $\bigwedge x. \text{continuous } (\text{at-right } x) F$ 
    unfolding continuous-within order-tendsto-iff eventually-at-right[OF less-add-one]
  proof safe
    show  $F x < u \implies \exists b > x. \forall y > x. y < b \longrightarrow F y < u$  for x u
      unfolding F-def cInf-less-iff[OF nonempty bdd-below] by auto
    next
      show  $\exists b > x. \forall y > x. y < b \longrightarrow l < F y$  if  $l < F x$  for x l
        using less-le-trans[OF l mono-F[THEN monoD, of x]] by (auto intro: less-add-one)
      qed
    moreover
      { fix x
        have  $F x \in \{-M..M\}$ 
          unfolding F-def using limset-bdd bdd-below r-unbdd by (intro closed-subset-contains-Inf)
        auto
        then have  $|F x| \leq M$  by auto }
    moreover have  $(\lambda n. f (d n) x) \longrightarrow F x$  if cts: continuous (at x) F for x
  proof (rule limsup-le-liminf-real)
    show  $\text{limsup } (\lambda n. f (d n) x) \leq F x$ 
    proof (rule tendsto-lowerbound)
      show  $(F \longrightarrow \text{ereal } (F x)) (\text{at-right } x)$ 
        using cts unfolding continuous-at-split by (auto simp: continuous-within)
      show  $\forall i \text{ in } \text{at-right } x. \text{limsup } (\lambda n. f (d n) x) \leq F i$ 
        unfolding eventually-at-right[OF less-add-one]
      proof (rule, rule, rule less-add-one, safe)
        fix y assume  $y: x < y$ 
        with dense-r obtain N where  $x < r N$   $r N < y$  by auto
        have *:  $y < r n' \implies \lim (?f N) \leq \lim (?f n')$  for n'
          using  $\langle r N < y \rangle$  by (intro LIMSEQ-le[OF lim-f lim-f]) (auto intro!:
mono[THEN monoD])
        have  $\text{limsup } (\lambda n. f (d n) x) \leq \text{limsup } (?f N)$ 
          using  $\langle x < r N \rangle$  by (auto intro!: Limsup-mono always-eventually
mono[THEN monoD])
        also have  $\dots = \lim (\lambda n. \text{ereal } (?f N n))$ 
          using rat-cnv[OF N] by (force intro!: convergent-limsup-cl simp: conver-
gent-def)
        also have  $\dots \leq F y$ 
          by (auto intro!: INF-greatest * simp: convergent-real-imp-convergent-ereal
rat-cnv F-eq)
      qed
    qed
  qed

```

```

    finally show  $\limsup (\lambda n. f (d n) x) \leq F y$  .
  qed
qed simp
show  $F x \leq \liminf (\lambda n. f (d n) x)$ 
proof (rule tendsto-upperbound)
  show  $(F \longrightarrow \text{ereal } (F x)) \text{ (at-left } x)$ 
    using cts unfolding continuous-at-split by (auto simp: continuous-within)
  show  $\forall_F i \text{ in at-left } x. F i \leq \liminf (\lambda n. f (d n) x)$ 
    unfolding eventually-at-left[OF minus-one-less]
  proof (rule, rule, rule minus-one-less, safe)
    fix  $y$  assume  $y: y < x$ 
    with dense-r obtain  $N$  where  $y < r N$   $r N < x$  by auto
    have  $F y \leq \liminf (?f N)$ 
      using  $\langle y < r N \rangle$  by (auto simp: F-eq convergent-real-imp-convergent-ereal
        rat-cnv convergent-liminf-cl intro!: INF-lower2)
    also have  $\dots \leq \liminf (\lambda n. f (d n) x)$ 
      using  $\langle r N < x \rangle$  by (auto intro!: Liminf-mono monoD[OF mono] al-
ways-eventually)
    finally show  $F y \leq \liminf (\lambda n. f (d n) x)$  .
  qed
qed simp
qed
ultimately show ?thesis using subseq by auto
qed

```

definition

```

tight :: (nat  $\Rightarrow$  real measure)  $\Rightarrow$  bool
where
  tight  $\mu \equiv (\forall n. \text{real-distribution } (\mu n)) \wedge (\forall (\varepsilon::\text{real}) > 0. \exists a b::\text{real}. a < b \wedge (\forall n.
\text{measure } (\mu n) \{a <..b\} > 1 - \varepsilon))$ 

```

theorem tight-imp-convergent-subsubsequence:

```

assumes  $\mu: \text{tight } \mu \text{ strict-mono } s$ 
shows  $\exists r M. \text{strict-mono } (r :: \text{nat} \Rightarrow \text{nat}) \wedge \text{real-distribution } M \wedge \text{weak-conv-m}$ 
 $(\mu \circ s \circ r) M$ 
proof -
  define  $f$  where  $f k = \text{cdf } (\mu (s k))$  for  $k$ 
  interpret  $\mu: \text{real-distribution } \mu k$  for  $k$ 
  using  $\mu$  unfolding tight-def by auto

  have  $rcont: \bigwedge x. \text{continuous } (\text{at-right } x) (f k)$ 
    and  $mono: \text{mono } (f k)$ 
    and  $top: (f k \longrightarrow 1) \text{ at-top}$ 
    and  $bot: (f k \longrightarrow 0) \text{ at-bot}$  for  $k$ 
  unfolding f-def mono-def
  using  $\mu.\text{cdf-nondecreasing } \mu.\text{cdf-is-right-cont } \mu.\text{cdf-lim-at-top-prob } \mu.\text{cdf-lim-at-bot}$ 

```

```

by auto
have bdd:  $|f\ k\ x| \leq 1$  for  $k\ x$ 
  by (auto simp add: abs-le-iff minus-le-iff f-def  $\mu$ .cdf-nonneg  $\mu$ .cdf-bounded-prob)

from Helly-selection[OF rcont mono bdd, of  $\lambda x. x$ ] obtain  $r\ F$ 
  where  $F$ : strict-mono  $r \wedge x. \text{continuous } (\text{at-right } x) F \text{ mono } F \wedge x. |F\ x| \leq 1$ 
  and  $\text{lim-}F$ :  $\wedge x. \text{continuous } (\text{at } x) F \implies (\lambda n. f\ (r\ n)\ x) \longrightarrow F\ x$ 
  by blast

have  $0 \leq f\ n\ x$  for  $n\ x$ 
  unfolding f-def by (rule  $\mu$ .cdf-nonneg)
have  $F$ -nonneg:  $0 \leq F\ x$  for  $x$ 
proof -
  obtain  $y$  where  $y < x$  isCont  $F\ y$ 
    using open-minus-countable[OF mono-ctble-discont[OF  $\langle \text{mono } F \rangle$ ], of  $\{.. $x\}$ ] by auto
  then have  $0 \leq F\ y$ 
    by (intro LIMSEQ-le-const[OF  $\text{lim-}F$ ]) (auto simp: f-def  $\mu$ .cdf-nonneg)
  also have  $\dots \leq F\ x$ 
    using  $\langle y < x \rangle$  by (auto intro!: monoD[OF  $\langle \text{mono } F \rangle$ ])
  finally show  $0 \leq F\ x$ .
qed

have  $Fab$ :  $\exists a\ b. (\forall x \geq b. F\ x \geq 1 - \varepsilon) \wedge (\forall x \leq a. F\ x \leq \varepsilon)$  if  $\varepsilon$ :  $0 < \varepsilon$  for  $\varepsilon$ 
proof auto
  obtain  $a'\ b'$  where  $a'b'$ :  $a' < b' \wedge k. \text{measure } (\mu\ k) \{a' <.. $b'\} > 1 - \varepsilon$ 
    using  $\varepsilon\ \mu$  by (auto simp: tight-def)
  obtain  $a$  where  $a$ :  $a < a'$  isCont  $F\ a$ 
    using open-minus-countable[OF mono-ctble-discont[OF  $\langle \text{mono } F \rangle$ ], of  $\{.. $a'\}$ ] by auto
  obtain  $b$  where  $b$ :  $b' < b$  isCont  $F\ b$ 
    using open-minus-countable[OF mono-ctble-discont[OF  $\langle \text{mono } F \rangle$ ], of  $\{b' <.. $\}$ ] by auto
  have  $a < b$ 
    using  $a\ b\ a'b'$  by simp

  let  $? \mu = \lambda k. \text{measure } (\mu\ (s\ (r\ k)))$ 
  have  $ab$ :  $? \mu\ k \{a <.. $b\} > 1 - \varepsilon$  for  $k$ 
  proof -
    have  $? \mu\ k \{a' <.. $b'\} \leq ? \mu\ k \{a <.. $b\}$ 
      using  $a\ b$  by (intro  $\mu$ .finite-measure-mono) auto
    then show  $?thesis$ 
      using  $a'b'(2)$  by (metis less-eq-real-def less-trans)
  qed

  have  $(\lambda k. ? \mu\ k \{.. $b\}) \longrightarrow F\ b$ 
    using  $b(2)\ \text{lim-}F$  unfolding f-def cdf-def o-def by auto
  then have  $1 - \varepsilon \leq F\ b$ 
  proof (rule tendsto-lowerbound, intro always-eventually allI)$$$$$$$$ 
```

```

fix k
have  $1 - \varepsilon < ?\mu \ k \ \{a <..b\}$ 
  using ab by auto
also have  $\dots \leq ?\mu \ k \ \{..b\}$ 
  by (auto intro!:  $\mu$ .finite-measure-mono)
finally show  $1 - \varepsilon \leq ?\mu \ k \ \{..b\}$ 
  by (rule less-imp-le)
qed (rule sequentially-bot)
then show  $\exists b. \forall x \geq b. 1 - \varepsilon \leq F \ x$ 
  using F unfolding mono-def by (metis order.trans)

have  $(\lambda k. ?\mu \ k \ \{..a\}) \longrightarrow F \ a$ 
  using a(2) lim-F unfolding f-def cdf-def o-def by auto
then have Fa:  $F \ a \leq \varepsilon$ 
proof (rule tendsto-upperbound, intro always-eventually allI)
  fix k
  have  $?\mu \ k \ \{..a\} + ?\mu \ k \ \{a <..b\} \leq 1$ 
    by (subst  $\mu$ .finite-measure-Union[symmetric]) auto
  then show  $?\mu \ k \ \{..a\} \leq \varepsilon$ 
    using ab[of k] by simp
  qed (rule sequentially-bot)
  then show  $\exists a. \forall x \leq a. F \ x \leq \varepsilon$ 
    using F unfolding mono-def by (metis order.trans)
qed

have  $(F \longrightarrow 1)$  at-top
proof (rule order-tendstoI)
  show  $1 < y \implies \forall_F \ x \text{ in } \textit{at-top}. F \ x < y$  for y
    using  $\langle \bigwedge x. |F \ x| \leq 1 \rangle \langle \bigwedge x. 0 \leq F \ x \rangle$  by (auto intro: le-less-trans always-eventually)
  fix y :: real assume  $y < 1$ 
  then obtain z where  $y < z < 1$ 
    using dense[of y 1] by auto
  with Fab[of  $1 - z$ ] show  $\forall_F \ x \text{ in } \textit{at-top}. y < F \ x$ 
    by (auto simp: eventually-at-top-linorder intro: less-le-trans)
qed
moreover
have  $(F \longrightarrow 0)$  at-bot
proof (rule order-tendstoI)
  show  $y < 0 \implies \forall_F \ x \text{ in } \textit{at-bot}. y < F \ x$  for y
    using  $\langle \bigwedge x. 0 \leq F \ x \rangle$  by (auto intro: less-le-trans always-eventually)
  fix y :: real assume  $0 < y$ 
  then obtain z where  $0 < z < y$ 
    using dense[of 0 y] by auto
  with Fab[of z] show  $\forall_F \ x \text{ in } \textit{at-bot}. F \ x < y$ 
    by (auto simp: eventually-at-bot-linorder intro: le-less-trans)
qed
ultimately have M: real-distribution (interval-measure F) cdf (interval-measure F) = F

```

```

    using F by (auto intro!: real-distribution-interval-measure cdf-interval-measure
simp: mono-def)
    with lim-F LIMSEQ-subseq-LIMSEQ M have weak-conv-m ( $\mu \circ s \circ r$ ) (interval-measure
F)
    by (auto simp: weak-conv-def weak-conv-m-def f-def comp-def)
    then show  $\exists r M. \text{strict-mono } (r :: \text{nat} \Rightarrow \text{nat}) \wedge (\text{real-distribution } M \wedge \text{weak-conv-m }
(\mu \circ s \circ r) M)$ 
    using F M by auto
qed

```

corollary *tight-subseq-weak-converge:*

```

    fixes  $\mu :: \text{nat} \Rightarrow \text{real measure}$  and  $M :: \text{real measure}$ 
    assumes  $\bigwedge n. \text{real-distribution } (\mu \ n) \text{ real-distribution } M$  and tight:  $\text{tight } \mu$  and
    subseq:  $\bigwedge s \nu. \text{strict-mono } s \implies \text{real-distribution } \nu \implies \text{weak-conv-m } (\mu \circ s) \nu$ 
 $\implies \text{weak-conv-m } (\mu \circ s) M$ 
    shows  $\text{weak-conv-m } \mu M$ 
proof (rule ccontr)
    define f where  $f \ n = \text{cdf } (\mu \ n)$  for n
    define F where  $F = \text{cdf } M$ 

    assume  $\neg \text{weak-conv-m } \mu M$ 
    then obtain x where  $x: \text{isCont } F \ x \neg (\lambda n. f \ n \ x) \longrightarrow F \ x$ 
    by (auto simp: weak-conv-m-def weak-conv-def f-def F-def)
    then obtain  $\varepsilon$  where  $\varepsilon > 0$  and infinite  $\{n. \neg \text{dist } (f \ n \ x) (F \ x) < \varepsilon\}$ 
    by (auto simp: tendsto-iff not-eventually INFM-iff-infinite cofinite-eq-sequentially[symmetric])
    then obtain  $s :: \text{nat} \Rightarrow \text{nat}$  where  $s: \bigwedge n. \neg \text{dist } (f \ (s \ n) \ x) (F \ x) < \varepsilon$  and
strict-mono s
    using enumerate-in-set enumerate-mono by (fastforce simp: strict-mono-def)
    then obtain  $r \ \nu$  where  $r: \text{strict-mono } r \text{ real-distribution } \nu \text{ weak-conv-m } (\mu \circ s \circ r) \ \nu$ 
    using tight-imp-convergent-subsubsequence[OF tight] by blast
    then have  $\text{weak-conv-m } (\mu \circ (s \circ r)) \ M$ 
    using  $\langle \text{strict-mono } s \rangle r$  by (intro subseq strict-mono-o) (auto simp: comp-assoc)
    then have  $(\lambda n. f \ (s \ (r \ n)) \ x) \longrightarrow F \ x$ 
    using x by (auto simp: weak-conv-m-def weak-conv-def F-def f-def)
    then show False
    using  $s \ \langle \varepsilon > 0 \rangle$  by (auto dest: tendstoD)
qed
end

```

16 Integral of sinc

```

theory Sinc-Integral
  imports Distributions
begin

```

16.1 Various preparatory integrals

Naming convention The theorem name consists of the following parts:

- Kind of integral: *has-bochner-integral* / *integrable* / *LBINT*
- Interval: Interval (0 / infinity / open / closed) (infinity / open / closed)
- Name of the occurring constants: power, exp, m (for minus), scale, sin, ...

lemma *has-bochner-integral-I0i-power-exp-m'*:

has-bochner-integral lborel ($\lambda x. x^k * \exp(-x) * \text{indicator } \{0 \dots\} x :: \text{real}$) (fact k)

using *nn-integral-power-times-exp-Ici*[of k]

by (*intro has-bochner-integral-nn-integral*)

(*auto simp: nn-integral-set-ennreal split: split-indicator*)

lemma *has-bochner-integral-I0i-power-exp-m*:

has-bochner-integral lborel ($\lambda x. x^k * \exp(-x) * \text{indicator } \{0 < \dots\} x :: \text{real}$) (fact k)

using *AE-lborel-singleton*[of 0]

by (*intro has-bochner-integral-cong-AE[THEN iffD1, OF - - - has-bochner-integral-I0i-power-exp-m']*)

(*auto split: split-indicator*)

lemma *integrable-I0i-exp-mscale*: $0 < (u :: \text{real}) \implies \text{set-integrable lborel } \{0 < \dots\} (\lambda x. \exp(-(x * u)))$

using *lborel-integrable-real-affine-iff*[of $u \lambda x. \text{indicator } \{0 < \dots\} x *_R \exp(-x)$ 0]

has-bochner-integral-I0i-power-exp-m[of 0]

by (*simp add: indicator-def zero-less-mult-iff mult-ac integrable.intros set-integrable-def*)

lemma *LBINT-I0i-exp-mscale*: $0 < (u :: \text{real}) \implies \text{LBINT } x=0..\infty. \exp(-(x * u)) = 1 / u$

using *lborel-integral-real-affine*[of $u \lambda x. \text{indicator } \{0 < \dots\} x *_R \exp(-x)$ 0]

has-bochner-integral-I0i-power-exp-m[of 0]

by (*auto simp: indicator-def zero-less-mult-iff interval-lebesgue-integral-0-infnty set-lebesgue-integral-def field-simps*

dest!: has-bochner-integral-integral-eq)

lemma *LBINT-I0c-exp-mscale-sin*:

LBINT $x=0..t. \exp(-(u * x)) * \sin x =$

$(1 / (1 + u^2)) * (1 - \exp(-(u * t)) * (u * \sin t + \cos t))$ (**is** $= ?F t$)

unfolding *zero-ereal-def*

proof (*subst interval-integral-FTC-finite*)

show ($?F \text{ has-vector-derivative } \exp(-(u * x)) * \sin x$) (at x within $\{\min 0 t \dots \max 0 t\}$) **for** x

by (*auto intro!: derivative-eq-intros*

simp: has-real-derivative-iff-has-vector-derivative[symmetric] power2-eq-square)

(*simp-all add: field-simps add-nonneg-eq-0-iff*)

qed (auto intro: continuous-at-imp-continuous-on)

lemma *LBINT-I0i-exp-mscale-sin*:

assumes $0 < x$
shows $LBINT\ u=0..\infty. |exp\ (-u * x) * sin\ x| = |sin\ x| / x$
proof (subst interval-integral-FTC-nonneg)
let $?F = \lambda u. 1 / x * (1 - exp\ (-u * x)) * |sin\ x|$
show $\bigwedge t. (?F\ has-real-derivative\ |exp\ (-t * x) * sin\ x|)\ (at\ t)$
using $\langle 0 < x \rangle$ **by** (auto intro!: derivative-eq-intros simp: abs-mult)
show $((?F \circ real-of-ereal) \longrightarrow 0)\ (at-right\ 0)$
using $\langle 0 < x \rangle$ **by** (auto simp: zero-ereal-def ereal-tendsto-simps intro!: tendsto-eq-intros)
have $*$: $((\lambda t. exp\ (-t * x)) \longrightarrow 0)\ at-top$
using $\langle 0 < x \rangle$
by (auto intro!: exp-at-bot[THEN filterlim-compose] filterlim-tendsto-pos-mult-at-top filterlim-ident
simp: filterlim-uminus-at-bot mult.commute[of - x])
show $((?F \circ real-of-ereal) \longrightarrow |sin\ x| / x)\ (at-left\ \infty)$
using $\langle 0 < x \rangle$ **unfolding** ereal-tendsto-simps
by (intro filterlim-compose[OF - *]) (auto intro!: tendsto-eq-intros filterlim-ident)
qed auto

lemma

shows integrable-inverse-1-plus-square:
set-integrable lborel (einterval $(-\infty)\ \infty$) $(\lambda x. inverse\ (1 + x^2))$
and *LBINT-inverse-1-plus-square*:
 $LBINT\ x=-\infty..\infty. inverse\ (1 + x^2) = pi$
proof –
have 1 : $-(pi / 2) < x \implies x * 2 < pi \implies (tan\ has-real-derivative\ 1 + (tan\ x)^2)\ (at\ x)\ \text{for}\ x$
using cos-gt-zero-pi[of x] **by** (subst tan-sec) (auto intro!: DERIV-tan simp: power-inverse)
have 2 : $-(pi / 2) < x \implies x * 2 < pi \implies isCont\ (\lambda x. 1 + (tan\ x)^2)\ x\ \text{for}\ x$
using cos-gt-zero-pi[of x] **by** auto
have 3 : $\llbracket -(pi / 2) < x; x * 2 < pi \rrbracket \implies isCont\ (\lambda x. inverse\ (1 + x^2))\ (tan\ x)\ \text{for}\ x$
by (rule continuous-intros | simp add: add-nonneg-eq-0-iff)+
show $LBINT\ x=-\infty..\infty. inverse\ (1 + x^2) = pi$
by (subst interval-integral-substitution-nonneg[of $-pi/2\ pi/2\ tan\ \lambda x. 1 + (tan\ x)^2$])
(auto intro: derivative-eq-intros 1 2 3 filterlim-tan-at-right
simp add: ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff
set-integrable-def)
show set-integrable lborel (einterval $(-\infty)\ \infty$) $(\lambda x. inverse\ (1 + x^2))$
by (subst interval-integral-substitution-nonneg[of $-pi/2\ pi/2\ tan\ \lambda x. 1 + (tan\ x)^2$])
(auto intro: derivative-eq-intros 1 2 3 filterlim-tan-at-right
simp add: ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff
set-integrable-def)

qed

lemma

shows *integrable-I0i-1-div-plus-square*:

interval-lebesgue-integrable lborel $0 \infty (\lambda x. 1 / (1 + x^2))$

and *LBINT-I0i-1-div-plus-square*:

LBINT $x=0..\infty. 1 / (1 + x^2) = \pi / 2$

proof –

have 1: $0 < x \implies x * 2 < \pi \implies (\tan \text{ has-real-derivative } 1 + (\tan x)^2) \text{ (at } x)$

for x

using *cos-gt-zero-pi*[of x] by (subst tan-sec) (auto intro!: *DERIV-tan simp: power-inverse*)

have 2: $0 < x \implies x * 2 < \pi \implies \text{isCont } (\lambda x. 1 + (\tan x)^2) \text{ } x \text{ for } x$

using *cos-gt-zero-pi*[of x] by auto

show *LBINT* $x=0..\infty. 1 / (1 + x^2) = \pi / 2$

by (subst *interval-integral-substitution-nonneg*[of $0 \pi/2 \tan \lambda x. 1 + (\tan x)^2$])

(auto intro: *derivative-eq-intros 1 2 tendsto-eq-intros*

simp add: ereal-tendsto-simps filterlim-tan-at-left zero-ereal-def add-nonneg-eq-0-iff

set-integrable-def)

show *interval-lebesgue-integrable lborel* $0 \infty (\lambda x. 1 / (1 + x^2))$

unfolding *interval-lebesgue-integrable-def*

by (subst *interval-integral-substitution-nonneg*[of $0 \pi/2 \tan \lambda x. 1 + (\tan x)^2$])

(auto intro: *derivative-eq-intros 1 2 tendsto-eq-intros*

simp add: ereal-tendsto-simps filterlim-tan-at-left zero-ereal-def add-nonneg-eq-0-iff

set-integrable-def)

qed

17 The sinc function, and the sine integral (Si)

abbreviation *sinc* :: *real* \Rightarrow *real* where

sinc $\equiv (\lambda x. \text{if } x = 0 \text{ then } 1 \text{ else } \sin x / x)$

lemma *sinc-at-0*: $((\lambda x. \sin x / x :: \text{real}) \longrightarrow 1) \text{ (at } 0)$

using *DERIV-sin* [of 0] by (auto simp add: *has-field-derivative-def field-has-derivative-at*)

lemma *isCont-sinc*: *isCont* *sinc* x

proof cases

assume $x = 0$ then show ?thesis

using *LIM-equal* [where $g = \lambda x. \sin x / x$ and $a=0$ and $f=\text{sinc}$ and $l=1$]

by (auto simp: *isCont-def sinc-at-0*)

next

assume $x \neq 0$ show ?thesis

by (rule *continuous-transform-within* [where $d = \text{abs } x$ and $f = \lambda x. \sin x / x$])

(auto simp add: *dist-real-def* $\langle x \neq 0 \rangle$)

qed

lemma *continuous-on-sinc*[*continuous-intros*]:

continuous-on $S \text{ } f \implies \text{continuous-on } S (\lambda x. \text{sinc } (f x))$

using *continuous-on-compose*[*of S f sinc, OF - continuous-at-imp-continuous-on*]
by (*auto simp: isCont-sinc*)

lemma *borel-measurable-sinc*[*measurable*]: *sinc* \in *borel-measurable borel*
by (*intro borel-measurable-continuous-onI continuous-at-imp-continuous-on ballI isCont-sinc*)

lemma *sinc-AE*: *AE x in lborel. sin x / x = sinc x*
by (*rule AE-I [where N = {0}], auto*)

definition *Si* :: *real* \Rightarrow *real* **where** *Si t* \equiv *LBINT x=0..t. sin x / x*

lemma *sinc-neg* [*simp*]: *sinc* ($- x$) = *sinc x*
by *auto*

lemma *Si-alt-def* : *Si t = LBINT x=0..t. sinc x*

proof *cases*

assume $0 \leq t$ **then show** *?thesis*

using *AE-lborel-singleton*[*of 0*]

by (*auto simp: Si-def intro!: interval-lebesgue-integral-cong-AE*)

next

assume $\neg 0 \leq t$ **then show** *?thesis*

unfolding *Si-def* **using** *AE-lborel-singleton*[*of 0*]

by (*subst (1 2) interval-integral-endpoints-reverse*)

(*auto simp: Si-def intro!: interval-lebesgue-integral-cong-AE*)

qed

lemma *Si-neg*:

assumes $T \geq 0$ **shows** *Si* ($- T$) = $- Si T$

proof $-$

have *LBINT x=ereal 0..T. -1 *_R sinc* ($- x$) = *LBINT y= ereal* ($- 0$)..*ereal* ($- T$). *sinc y*

by (*rule interval-integral-substitution-finite [OF assms]*)

(*auto intro: derivative-intros continuous-at-imp-continuous-on isCont-sinc*)

also have (*LBINT x=ereal 0..T. -1 *_R sinc* ($- x$)) = $-(LBINT x=ereal 0..T. sinc x)$

by (*subst sinc-neg*) (*simp-all add: interval-lebesgue-integral-uminus*)

finally have $*$: $-(LBINT x=ereal 0..T. sinc x) = LBINT y= ereal 0..ereal (- T). sinc y$

by *simp*

show *?thesis*

using *assms* **unfolding** *Si-alt-def*

by (*subst zero-ereal-def*) + (*auto simp add: * [symmetric]*)

qed

lemma *integrable-sinc'*:

interval-lebesgue-integrable lborel (*ereal 0*) (*ereal T*) ($\lambda t. sin (t * \vartheta) / t$)

proof $-$

```

have *: interval-lebesgue-integrable lborel (ereal 0) (ereal T) ( $\lambda t. \vartheta * \text{sinc } (t * \vartheta)$ )
by (intro interval-lebesgue-integrable-mult-right interval-integrable-isCont continuous-within-compose3 [OF isCont-sinc])
    auto
have 0  $\notin$  einterval (min (ereal 0) (ereal T)) (max (ereal 0) (ereal T))
by (smt (verit, best) einterval-iff max-def min-def-raw order-less-le)
then show ?thesis
by (intro interval-lebesgue-integrable-cong-AE[THEN iffD1, OF - - - *]) auto
qed

```

```

lemma DERIV-Si: (Si has-real-derivative sinc x) (at x)
proof -
have (at x within {min 0 (x - 1)..max 0 (x + 1)}) = at x
by (intro at-within-interior) auto
moreover have min 0 (x - 1)  $\leq x \leq$  max 0 (x + 1)
by auto
ultimately show ?thesis
using interval-integral-FTC2[of min 0 (x - 1) 0 max 0 (x + 1) sinc x]
by (auto simp: continuous-at-imp-continuous-on isCont-sinc Si-alt-def[abs-def]
    zero-ereal-def
    has-real-derivative-iff-has-vector-derivative
    split del: if-split)
qed

```

```

lemma isCont-Si: isCont Si x
using DERIV-Si by (rule DERIV-isCont)

```

```

lemma borel-measurable-Si[measurable]: Si  $\in$  borel-measurable borel
by (auto intro: isCont-Si continuous-at-imp-continuous-on borel-measurable-continuous-onI)

```

```

lemma Si-at-top-LBINT:
  (( $\lambda t. (LBINT x=0..\infty. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))$ ))  $\longrightarrow$ 
  0) at-top
proof -
let ?F =  $\lambda x t. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2) :: \text{real}$ 
have int: set-integrable lborel {0<..} ( $\lambda x. \exp(-x) * (x + 1) :: \text{real}$ )
unfolding distrib-left
using has-bochner-integral-I0i-power-exp-m[of 0] has-bochner-integral-I0i-power-exp-m[of 1]
by (intro set-integral-add) (auto dest!: integrable.intros simp: ac-simps set-integrable-def)

have (( $\lambda t :: \text{real}. LBINT x:\{0<..\}. ?F x t$ )  $\longrightarrow$  (LBINT x::real:\{0<..\}. 0)) at-top
unfolding set-lebesgue-integral-def
proof (rule integral-dominated-convergence-at-top[OF - - int [unfolded set-integrable-def]],
    simp-all del: abs-divide split: split-indicator)
have *: 0 < x  $\implies |x * \sin t + \cos t| / (1 + x^2) \leq (x * 1 + 1) / 1$  for x t ::

```

```

real
  by (intro frac-le abs-triangle-ineq[THEN order-trans] add-mono)
    (auto simp add: abs-mult simp del: mult-1-right intro!: mult-mono)
  then have  $1 \leq t \implies 0 < x \implies |\text{?F } x \ t| \leq \exp(-x) * (x + 1)$  for  $x \ t :: \text{real}$ 
    by (auto simp add: abs-mult times-divide-eq-right[symmetric] simp del:
times-divide-eq-right
      intro!: mult-mono)
  then show  $\forall_F i \text{ in at-top. } AE \ x \text{ in lborel. } 0 < x \longrightarrow |\text{?F } x \ i| \leq \exp(-x) * (x + 1)$ 
    by (auto intro: eventually-mono eventually-ge-at-top[of 1::real])
  show  $AE \ x \text{ in lborel. } 0 < x \longrightarrow (\text{?F } x \longrightarrow 0) \text{ at-top}$ 
  proof (intro always-eventually impI allI)
    fix  $x :: \text{real}$  assume  $0 < x$ 
    show  $((\lambda t. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2)) \longrightarrow 0) \text{ at-top}$ 
    proof (rule Lim-null-comparison)
      show  $\forall_F t \text{ in at-top. } \text{norm } (\text{?F } x \ t) \leq \exp(-(x * t)) * (x + 1)$ 
      using eventually-ge-at-top[of 1::real] *  $\langle 0 < x \rangle$ 
      by (auto simp add: abs-mult times-divide-eq-right[symmetric] simp del:
times-divide-eq-right
        intro!: mult-mono elim: eventually-mono)
    show  $((\lambda t. \exp(-(x * t)) * (x + 1)) \longrightarrow 0) \text{ at-top}$ 
    by (auto simp: filterlim-uminus-at-top [symmetric]
      intro!: filterlim-tendsto-pos-mult-at-top[OF tendsto-const]  $\langle 0 < x \rangle$ 
filterlim-ident
      tendsto-mult-left-zero filterlim-compose[OF exp-at-bot])
    qed
  qed
  qed
  then show  $((\lambda t. (LBINT \ x=0..\infty. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))) \longrightarrow 0) \text{ at-top}$ 
    by (simp add: interval-lebesgue-integral-0-infty set-lebesgue-integral-def)
  qed

lemma Si-at-top-integrable:
  assumes  $t \geq 0$ 
  shows interval-lebesgue-integrable lborel  $0 \ \infty (\lambda x. \exp(-(x * t)) * (x * \sin t + \cos t) / (1 + x^2))$ 
    using  $\langle 0 \leq t \rangle$  unfolding le-less
  proof
    assume  $0 = t$  then show ?thesis
      using integrable-I0i-1-div-plus-square by simp
    next
      assume [arith]:  $0 < t$ 
      have *:  $0 \leq a \implies 0 < x \implies a / (1 + x^2) \leq a$  for  $a \ x :: \text{real}$ 
      using zero-le-power2[of x, arith] using divide-left-mono[of 1  $1 + x^2$  a] by auto
      have set-integrable lborel  $\{0 <.. \}$   $(\lambda x. (\exp(-x) * x) * (\sin t / t) + \exp(-x) * \cos t)$ 
      using has-bochner-integral-I0i-power-exp-m[of 0] has-bochner-integral-I0i-power-exp-m[of 1]

```

```

    by (intro set-integral-add set-integrable-mult-left)
      (auto dest!: integrable.intros simp: ac-simps set-integrable-def)
  from lborel-integrable-real-affine[OF this [unfolded set-integrable-def], of t 0]
  show ?thesis
    unfolding interval-lebesgue-integral-0-infty set-integrable-def
    by (rule Bochner-Integration.integrable-bound) (auto simp: field-simps * split:
split-indicator)
qed

```

lemma *Si-at-top*: $(Si \longrightarrow pi / 2) \text{ at-top}$

proof –

```

  have †:  $\forall_F t \text{ in at-top. } pi / 2 - (LBINT u=0..\infty. \exp(-(u * t)) * (u * \sin t + \cos t) / (1 + u^2)) = Si\ t$ 
    using eventually-ge-at-top[of 0]
  proof eventually-elim
    fix t :: real assume t ≥ 0
    have Si t = LBINT x=0..t. sin x * (LBINT u=0..\infty. exp(-(u * x)))
      unfolding Si-def using ‹0 ≤ t›
      by (intro interval-integral-discrete-difference[where X={0}]) (auto simp:
LBINT-I0i-exp-mscale)
    also have ... = (LBINT x. (LBINT u=ereal 0..\infty. indicator {0 <..< t} x *R
sin x * exp(-(u * x))))
      using ‹0 ≤ t› by (simp add: zero-ereal-def interval-lebesgue-integral-le-eq
mult-ac set-lebesgue-integral-def)
    also have ... = (LBINT x. (LBINT u. indicator ({0 <..} × {0 <..< t}) (u,
x) *R (sin x * exp(-(u * x)))))
      apply (subst interval-integral-I0i)
    unfolding set-lebesgue-integral-def by (simp-all add: indicator-times ac-simps
)
    also have ... = (LBINT u. (LBINT x. indicator ({0 <..} × {0 <..< t}) (u,
x) *R (sin x * exp(-(u * x)))))
  proof (intro lborel-pair.Fubini-integral[symmetric] lborel-pair.Fubini-integrable)
    show (λ(x, y). indicator ({0 <..} × {0 <..< t}) (y, x) *R (sin x * exp(-(y
* x))))
      ∈ borel-measurable (lborel ⊗M lborel) (is ?f ∈ borel-measurable -)
    by measurable

```

```

  have AE x in lborel. indicator {0..t} x *R abs (sinc x) = (LBINT y. norm
(?f (x, y)))
    using AE-lborel-singleton[of 0] AE-lborel-singleton[of t]
  proof eventually-elim
    fix x :: real assume x: x ≠ 0 x ≠ t
    have (LBINT y. |indicator ({0 <..} × {0 <..< t}) (y, x) *R (sin x * exp(-(y
* x)))|) =
      (LBINT y. |sin x| * exp(-(y * x)) * indicator {0 <..} y * indicator
{0 <..< t} x)
      by (intro Bochner-Integration.integral-cong) (auto split: split-indicator
simp: abs-mult)
    also have ... = |sin x| * indicator {0 <..< t} x * (LBINT y=0..\infty. exp(-

```

```

(y * x)))
  by (cases x > 0)
    (auto simp: field-simps interval-lebesgue-integral-0-infty set-lebesgue-integral-def
split: split-indicator)
  also have ... = |sin x| * indicator {0 <..<t} x * (1 / x)
    by (cases x > 0) (auto simp add: LBINT-I0i-exp-mscale)
  also have ... = indicator {0..t} x *R |sinc x|
    using x by (simp add: field-simps split: split-indicator)
  finally show indicator {0..t} x *R abs (sinc x) = (LBINT y. norm (?f (x,
y)))
    by simp
qed
moreover have integrable lborel (λx. indicat-real {0..t} x *R |sinc x|)
by (auto intro!: borel-integrable-compact continuous-intros simp del: real-scaleR-def)
ultimately show integrable lborel (λx. LBINT y. norm (?f (x, y)))
  by (rule integrable-cong-AE-imp[rotated 2]) simp

have 0 < x ⇒ set-integrable lborel {0<..} (λy. sin x * exp (-(y * x))) for
x :: real
  by (intro set-integrable-mult-right integrable-I0i-exp-mscale)
  then show AE x in lborel. integrable lborel (λy. ?f (x, y))
  by (intro AE-I2) (auto simp: indicator-times set-integrable-def split: split-indicator)
qed
also have ... = (LBINT u=0..∞. (LBINT x=0..t. exp (-(u * x)) * sin x))
  using <0≤t>
  by (auto simp: interval-lebesgue-integral-def set-lebesgue-integral-def zero-ereal-def
ac-simps
split: split-indicator intro!: Bochner-Integration.integral-cong)
  also have ... = (LBINT u=0..∞. 1 / (1 + u2) - 1 / (1 + u2) * (exp (-(u
* t)) * (u * sin t + cos t)))
  by (auto simp: divide-simps LBINT-I0c-exp-mscale-sin intro!: interval-integral-cong)
  also have ... = pi / 2 - (LBINT u=0..∞. exp (-(u * t)) * (u * sin t + cos
t) / (1 + u2))
  using Si-at-top-integrable[OF <0≤t>] by (simp add: integrable-I0i-1-div-plus-square
LBINT-I0i-1-div-plus-square)
  finally show pi / 2 - (LBINT u=0..∞. exp (-(u * t)) * (u * sin t + cos t)
/ (1 + u2)) = Si t ..
qed
show ?thesis
  by (rule Lim-transform-eventually [OF - †])
    (auto intro!: tendsto-eq-intros Si-at-top-LBINT)
qed

```

17.1 The final theorems: boundedness and scalability

lemma *bounded-Si*: $\exists B. \forall T. |Si\ T| \leq B$

proof –

have *: $0 \leq y \implies dist\ x\ y < z \implies abs\ x \leq y + z$ for $x\ y\ z :: real$
 by (simp add: dist-real-def)

have *eventually* $(\lambda T. \text{dist } (Si \ T) \ (pi \ / \ 2) < 1)$ *at-top*
using *Si-at-top* **by** *(elim tendstoD) simp*
then have *eventually* $(\lambda T. 0 \leq T \wedge |Si \ T| \leq pi \ / \ 2 + 1)$ *at-top*
using *eventually-ge-at-top[of 0]* **by** *eventually-elim (insert *[of pi/2 Si - 1], auto)*
then have $\exists T. 0 \leq T \wedge (\forall t \geq T. |Si \ t| \leq pi \ / \ 2 + 1)$
by *(auto simp add: eventually-at-top-linorder)*
then obtain *T* **where** $T: 0 \leq T \wedge t. t \geq T \implies |Si \ t| \leq pi \ / \ 2 + 1$
by *auto*
moreover have $t \leq -T \implies |Si \ t| \leq pi \ / \ 2 + 1$ **for** *t*
using *T(1) T(2)[of -t] Si-neg[of -t] by simp*
moreover have $\exists M. \forall t. -T \leq t \wedge t \leq T \longrightarrow |Si \ t| \leq M$
by *(rule isCont-bounded) (auto intro!: isCont-Si continuous-intros <0 ≤ T>)*
then obtain *M* **where** $M: \bigwedge t. -T \leq t \wedge t \leq T \implies |Si \ t| \leq M$
by *auto*
ultimately show *?thesis*
by *(intro exI[of - max M (pi/2 + 1)]) (meson le-max-iff-disj linorder-not-le order-le-less)*
qed

lemma *LBINT-I0c-sin-scale-divide:*

assumes $T \geq 0$
shows $LBINT \ t=0..T. \sin \ (t * \vartheta) \ / \ t = \text{sgn } \vartheta * Si \ (T * |\vartheta|)$
proof –
{ assume $0 < \vartheta$
have $(LBINT \ t=ereal \ 0..T. \sin \ (t * \vartheta) \ / \ t) = (LBINT \ t=ereal \ 0..T. \vartheta *_R \text{sinc} \ (t * \vartheta))$
by *(rule interval-integral-discrete-difference[of {0}]) auto*
also have $\dots = (LBINT \ t=ereal \ (0 * \vartheta)..T * \vartheta. \text{sinc } t)$
apply *(rule interval-integral-substitution-finite [OF assms])*
apply *(subst mult.commute, rule DERIV-subset)*
by *(auto intro!: derivative-intros continuous-at-imp-continuous-on isCont-sinc)*
also have $\dots = (LBINT \ t=ereal \ (0 * \vartheta)..T * \vartheta. \sin \ t \ / \ t)$
by *(rule interval-integral-discrete-difference[of {0}]) auto*
finally have $(LBINT \ t=ereal \ 0..T. \sin \ (t * \vartheta) \ / \ t) = (LBINT \ t=ereal \ 0..T * \vartheta. \sin \ t \ / \ t)$
by *simp*
hence $(LBINT \ x. \text{indicator } \{0 <..< T\} \ x * \sin \ (x * \vartheta) \ / \ x) = (LBINT \ x. \text{indicator } \{0 <..< T * \vartheta\} \ x * \sin \ x \ / \ x)$
using *assms <0 < \vartheta> unfolding interval-lebesgue-integral-def einterval-eq zero-ereal-def*
by *(auto simp: ac-simps set-lebesgue-integral-def)*
} note *aux1 = this*
{ assume $0 > \vartheta$
have $[simp]: \text{integrable lborel } (\lambda x. \sin \ (x * \vartheta) * \text{indicator } \{0 <..< T\} \ x \ / \ x)$
using *integrable-sinc' [of T \vartheta] assms*
by *(simp add: interval-lebesgue-integrable-def set-integrable-def ac-simps)*
have $(LBINT \ t=ereal \ 0..T. \sin \ (t * -\vartheta) \ / \ t) = (LBINT \ t=ereal \ 0..T. -\vartheta *_R$


```

sinc (t * -∅))
  by (rule interval-integral-discrete-difference[of {0}]) auto
also have ... = (LBINT t=ereal (0 * -∅)..T * -∅. sinc t)
  apply (rule interval-integral-substitution-finite [OF assms])
  apply (subst mult.commute, rule DERIV-subset)
  by (auto intro!: derivative-intros continuous-at-imp-continuous-on isCont-sinc)
also have ... = (LBINT t=ereal (0 * -∅)..T * -∅. sin t / t)
  by (rule interval-integral-discrete-difference[of {0}]) auto
finally have (LBINT t=ereal 0..T. sin (t * -∅) / t) = (LBINT t=ereal 0..T
* -∅. sin t / t)
  by simp
hence (LBINT x. indicator {0 <..

```

18 The Levy inversion theorem, and the Levy continuity theorem.

```

theory Levy
  imports Characteristic-Functions Helly-Selection Sinc-Integral
begin

```

18.1 The Levy inversion theorem

```

lemma Levy-Inversion-aux1:
  fixes a b :: real
  assumes a ≤ b
  shows ((λt. (iexp (-(t * a)) - iexp (-(t * b))) / (i * t)) ⟶ b - a) (at 0)
    (is (?F ⟶ -) (at -))
proof -
  have 1: cmod (?F t - (b - a)) ≤ a^2 / 2 * abs t + b^2 / 2 * abs t if t ≠ 0
  for t
  proof -
    have cmod (?F t - (b - a)) = cmod (
      (iexp (-(t * a)) - (1 + i * -(t * a))) / (i * t) -
      (iexp (-(t * b)) - (1 + i * -(t * b))) / (i * t))
      (is - = cmod (?one / (i * t) - ?two / (i * t)))
    )
  qed

```

```

    using ⟨t ≠ 0⟩ by (intro arg-cong[where f=norm]) (simp add: field-simps)
  also have ... ≤ cmod (?one / (i * t)) + cmod (?two / (i * t))
    by (rule norm-triangle-ineq4)
  also have cmod (?one / (i * t)) = cmod ?one / abs t
    by (simp add: norm-divide norm-mult)
  also have cmod (?two / (i * t)) = cmod ?two / abs t
    by (simp add: norm-divide norm-mult)
  also have cmod ?one / abs t + cmod ?two / abs t ≤
    ((- (a * t))^2 / 2) / abs t + ((- (b * t))^2 / 2) / abs t
    using iexp-approx1 [of -(t * -) 1]
    by (intro add-mono divide-right-mono abs-ge-zero) (auto simp: field-simps
eval-nat-numeral)
  also have ... = a^2 / 2 * abs t + b^2 / 2 * abs t
    using ⟨t ≠ 0⟩ apply (case-tac t ≥ 0, simp add: field-simps power2-eq-square)
    using ⟨t ≠ 0⟩ by (subst (1 2) abs-of-neg, auto simp add: field-simps
power2-eq-square)
  finally show cmod (?F t - (b - a)) ≤ a^2 / 2 * abs t + b^2 / 2 * abs t .
qed
show ?thesis
  apply (rule LIM-zero-cancel)
  apply (rule tendsto-norm-zero-cancel)
  apply (rule real-LIM-sandwich-zero [OF - - 1])
  apply (auto intro!: tendsto-eq-intros)
  done
qed

```

lemma *Levy-Inversion-aux2*:

```

  fixes a b t :: real
  assumes a ≤ b and t ≠ 0
  shows cmod ((iexp (t * b) - iexp (t * a)) / (i * t)) ≤ b - a (is ?F ≤ -)
proof -
  have ?F = cmod (iexp (t * a) * (iexp (t * (b - a)) - 1) / (i * t))
    using ⟨t ≠ 0⟩ by (intro arg-cong[where f=norm]) (simp add: field-simps
exp-diff exp-minus)
  also have ... = cmod (iexp (t * (b - a)) - 1) / abs t
    unfolding norm-divide norm-mult norm-exp-i-times using ⟨t ≠ 0⟩
    by (simp add: complex-eq-iff norm-mult)
  also have ... ≤ abs (t * (b - a)) / abs t
    using iexp-approx1 [of t * (b - a) 0]
    by (intro divide-right-mono) (auto simp add: field-simps eval-nat-numeral)
  also have ... = b - a
    using assms by (auto simp add: abs-mult)
  finally show ?thesis .
qed

```

theorem (in *real-distribution*) *Levy-Inversion*:

```

  fixes a b :: real
  assumes a ≤ b

```

```

defines  $\mu \equiv \text{measure } M$  and  $\varphi \equiv \text{char } M$ 
assumes  $\mu \{a\} = 0$  and  $\mu \{b\} = 0$ 
shows  $(\lambda T. 1 / (2 * \pi i) * (\text{CLBINT } t=-T..T. (\text{iexp } (-(t * a)) - \text{iexp } (-(t * b))) / (i * t) * \varphi t))$ 
   $\longrightarrow \mu \{a <..b\}$ 
   $(\text{is } (\lambda T. 1 / (2 * \pi i) * (\text{CLBINT } t=-T..T. ?F t * \varphi t)) \longrightarrow \text{of-real } (\mu \{a <..b\}))$ 
proof -
  interpret  $P$ : pair-sigma-finite lborel  $M$  ..
  from bounded-Si obtain  $B$  where  $B\text{prop}$ :  $\bigwedge T. \text{abs } (Si \ T) \leq B$  by auto
  from  $B\text{prop}$  [of 0] have [simp]:  $B \geq 0$  by auto
  let  $?f = \lambda t \ x :: \text{real}. (\text{iexp } (t * (x - a)) - \text{iexp } (t * (x - b))) / (i * t)$ 
  { fix  $T :: \text{real}$ 
    assume  $T \geq 0$ 
    let  $?f' = \lambda(t, x). \text{indicator } \{-T <..< T\} \ t *_{\mathbb{R}} ?f \ t \ x$ 
    { fix  $x$ 
      have int: interval-lebesgue-integrable lborel (ereal 0) (ereal T)  $(\lambda t. 2 * (\sin (t * (x - w)) / t))$  for  $w$ 
      using integrable-sinc' interval-lebesgue-integrable-mult-right by blast
      have 1: complex-interval-lebesgue-integrable lborel  $u \ v$   $(\lambda t. ?f \ t \ x)$  for  $u \ v :: \text{real}$ 
      using Levy-Inversion-aux2 [of  $x - b \ x - a$ ]
      apply (simp add: interval-lebesgue-integrable-def set-integrable-def del: times-divide-eq-left)
      apply (intro integrableI-bounded-set-indicator [where  $B=b - a$ ] conjI impI)
      apply (auto intro!: AE-I [of - -  $\{0\}$ ] simp: assms)
      done
      have  $(\text{CLBINT } t. ?f' (t, x)) = (\text{CLBINT } t=-T..T. ?f \ t \ x)$ 
      using  $\langle T \geq 0 \rangle$  by (simp add: interval-lebesgue-integral-def set-lebesgue-integral-def)
      also have  $\dots = (\text{CLBINT } t=-T..(0 :: \text{real}). ?f \ t \ x) + (\text{CLBINT } t=(0 :: \text{real})..T. ?f \ t \ x)$ 
      (is - = - +  $?t$ )
      using 1 by (intro interval-integral-sum [symmetric] (simp add: min-absorb1 max-absorb2  $\langle T \geq 0 \rangle$ ))
      also have  $(\text{CLBINT } t=-T..(0 :: \text{real}). ?f \ t \ x) = (\text{CLBINT } t=(0::\text{real})..T. ?f \ (-t) \ x)$ 
      by (subst interval-integral-reflect) auto
      also have  $\dots + ?t = (\text{CLBINT } t=(0::\text{real})..T. ?f \ (-t) \ x + ?f \ t \ x)$ 
      using 1
      by (intro interval-lebesgue-integral-add (2) [symmetric] interval-integrable-mirror [THEN iffD2]) simp-all
      also have  $\dots = (\text{CLBINT } t=(0::\text{real})..T. ((\text{iexp } (t * (x - a)) - \text{iexp } (-(t * (x - a)))) - (\text{iexp } (t * (x - b)) - \text{iexp } (-(t * (x - b))))) / (i * t))$ 
      using  $\langle T \geq 0 \rangle$  by (intro interval-integral-cong) (auto simp add: field-split-simps)
      also have  $\dots = (\text{CLBINT } t=(0::\text{real})..T. \text{complex-of-real } (2 * (\sin (t * (x - a)) / t) - 2 * (\sin (t * (x - b)) / t)))$ 
      using  $\langle T \geq 0 \rangle$ 
      by (intro interval-integral-cong) (simp add: divide-simps Im-divide Re-divide)
    }
  }

```

```

Im-exp Re-exp complex-eq-iff)
  also have ... = complex-of-real (LBINT t=(0::real)..T.
    2 * (sin (t * (x - a)) / t) - 2 * (sin (t * (x - b)) / t))
    by (rule interval-lebesgue-integral-of-real)
  also have ... = complex-of-real ((LBINT t=ereal 0..ereal T. 2 * (sin (t * (x
    - a)) / t)) -
    (LBINT t=ereal 0..ereal T. 2 * (sin (t * (x - b))
    / t)))
    unfolding interval-lebesgue-integral-diff
    using int by auto
  also have ... = complex-of-real (2 * (sgn (x - a) * Si (T * abs (x - a)) -
    sgn (x - b) * Si (T * abs (x - b))))
    unfolding interval-lebesgue-integral-mult-right
    by (simp add: zero-ereal-def[symmetric] LBINT-I0c-sin-scale-divide[OF <T
    ≥ 0>])
  finally have (CLBINT t. ?f' (t, x)) =
    2 * (sgn (x - a) * Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x
    - b))) .
  } note main-eq = this
  have (CLBINT t=-T..T. ?F t * φ t) =
    (CLBINT t. (CLINT x | M. ?F t * iexp (t * x) * indicator {-T<..M M) ({- T<..M M. cmod (case x of (t,
    x) ⇒ ?f' (t, x)) ≤ b - a
      using Levy-Inversion-aux2[of x - b x - a for x] <a ≤ b>
      by (intro AE-I [of - - {0} × UNIV]) (force simp: emeasure-pair-measure-Times)+
    qed (auto split: split-indicator split-indicator-asm)
  also have ... = (CLINT x | M. (complex-of-real (2 * (sgn (x - a) *
    Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))))
    using main-eq by (intro Bochner-Integration.integral-cong, auto)
  also have ... = complex-of-real (LINT x | M. (2 * (sgn (x - a) *
    Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))))
    by (rule integral-complex-of-real)
  finally have (CLBINT t=-T..T. ?F t * φ t) =
    complex-of-real (LINT x | M. (2 * (sgn (x - a) *
    Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b))))) .
  } note main-eq2 = this

```

```

have (λT :: nat. LINT x | M. (2 * (sgn (x - a) *
  Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))) →
  (LINT x | M. 2 * pi * indicator {a<..b} x)
proof (rule integral-dominated-convergence [where w=λx. 4 * B])
  show integrable M (λx. 4 * B)
  by (rule integrable-const-bound [of - 4 * B]) auto
next
let ?S = λn::nat. λx. sgn (x - a) * Si (n * |x - a|) - sgn (x - b) * Si (n *
|x - b|)
{ fix n x
  have norm (?S n x) ≤ norm (sgn (x - a) * Si (n * |x - a|)) + norm (sgn
(x - b) * Si (n * |x - b|))
  by (rule norm-triangle-ineq4)
  also have ... ≤ B + B
  using Bprop by (intro add-mono) (auto simp: abs-mult abs-sgn-eq)
  finally have norm (2 * ?S n x) ≤ 4 * B
  by simp }
then show ∧n. AE x in M. norm (2 * ?S n x) ≤ 4 * B
  by auto
have AE x in M. x ≠ a AE x in M. x ≠ b
  using prob-eq-0[of {a}] prob-eq-0[of {b}] ⟨μ {a} = 0⟩ ⟨μ {b} = 0⟩ by (auto
simp: μ-def)
then show AE x in M. (λn. 2 * ?S n x) → 2 * pi * indicator {a<..b} x
proof eventually-elim
  fix x assume x: x ≠ a x ≠ b
  then have (λn. 2 * (sgn (x - a) * Si (|x - a| * n) - sgn (x - b) * Si (|x
- b| * n)))
    → 2 * (sgn (x - a) * (pi / 2) - sgn (x - b) * (pi / 2))
  by (intro tendsto-intros filterlim-compose[OF Si-at-top]
    filterlim-tendsto-pos-mult-at-top[OF tendsto-const] filterlim-real-sequentially)
    auto
  also have (λn. 2 * (sgn (x - a) * Si (|x - a| * n) - sgn (x - b) * Si (|x -
b| * n))) = (λn. 2 * ?S n x)
  by (auto simp: ac-simps)
  also have 2 * (sgn (x - a) * (pi / 2) - sgn (x - b) * (pi / 2)) = 2 * pi *
indicator {a<..b} x
  using x ⟨a ≤ b⟩ by (auto split: split-indicator)
  finally show (λn. 2 * ?S n x) → 2 * pi * indicator {a<..b} x .
qed
qed simp-all
also have (LINT x | M. 2 * pi * indicator {a<..b} x) = 2 * pi * μ {a<..b}
  by (simp add: μ-def)
finally have (λT. LINT x | M. (2 * (sgn (x - a) *
  Si (T * abs (x - a)) - sgn (x - b) * Si (T * abs (x - b)))) →
  2 * pi * μ {a<..b} .
with main-eq2 show ?thesis
  by (auto intro!: tendsto-eq-intros)
qed

```

theorem *Levy-uniqueness:*

fixes $M1\ M2 :: \text{real measure}$

assumes *real-distribution* $M1$ *real-distribution* $M2$ **and**

$\text{char } M1 = \text{char } M2$

shows $M1 = M2$

proof –

interpret $M1$: *real-distribution* $M1$ **by** (*rule assms*)

interpret $M2$: *real-distribution* $M2$ **by** (*rule assms*)

have *countable* ($\{x. \text{measure } M1 \{x\} \neq 0\} \cup \{x. \text{measure } M2 \{x\} \neq 0\}$)

by (*intro countable-Un* $M2.\text{countable-support}$ $M1.\text{countable-support}$)

then have *count*: *countable* ($\{x. \text{measure } M1 \{x\} \neq 0 \vee \text{measure } M2 \{x\} \neq 0\}$)

by (*simp add: Un-def*)

have $\text{cdf } M1 = \text{cdf } M2$

proof (*rule ext*)

fix x

let $?D = \lambda x. |\text{cdf } M1\ x - \text{cdf } M2\ x|$

{ **fix** $\varepsilon :: \text{real}$

assume $\varepsilon > 0$

have ($?D \longrightarrow 0$) *at-bot*

using $M1.\text{cdf-lim-at-bot}$ $M2.\text{cdf-lim-at-bot}$ **by** (*intro tendsto-eq-intros*) *auto*

then have *eventually* ($\lambda y. ?D\ y < \varepsilon / 2 \wedge y \leq x$) *at-bot*

using $\langle \varepsilon > 0 \rangle$ **by** (*simp only: tendsto-iff dist-real-def diff-0-right eventually-conj eventually-le-at-bot abs-idempotent*)

then obtain M **where** $\bigwedge y. y \leq M \implies ?D\ y < \varepsilon / 2 \wedge M \leq x$

unfolding *eventually-at-bot-linorder* **by** *auto*

with *open-minus-countable*[*OF count*, *of* $\{.. < M\}$] **obtain** a **where**

$\text{measure } M1\ \{a\} = 0 \wedge \text{measure } M2\ \{a\} = 0 \wedge a < M \wedge a \leq x$ **and** $1: ?D\ a < \varepsilon$

$/ 2$

by *auto*

have ($?D \longrightarrow ?D\ x$) (*at-right* x)

using $M1.\text{cdf-is-right-cont}$ [*of* x] $M2.\text{cdf-is-right-cont}$ [*of* x]

by (*intro tendsto-intros*) (*auto simp add: continuous-within*)

then have *eventually* ($\lambda y. |?D\ y - ?D\ x| < \varepsilon / 2$) (*at-right* x)

using $\langle \varepsilon > 0 \rangle$ **by** (*simp only: tendsto-iff dist-real-def eventually-conj eventually-at-right-less*)

then obtain N **where** $N > x \wedge \bigwedge y. x < y \implies y < N \implies |?D\ y - ?D\ x| < \varepsilon / 2$

by (*auto simp add: eventually-at-right*[*OF less-add-one*])

with *open-minus-countable*[*OF count*, *of* $\{x <.. < N\}$] **obtain** b **where** $x < b < N$

$\text{measure } M1\ \{b\} = 0 \wedge \text{measure } M2\ \{b\} = 0$ **and** $2: |?D\ b - ?D\ x| < \varepsilon / 2$

by (*auto simp: abs-minus-commute*)

from $\langle a \leq x \rangle \langle x < b \rangle$ **have** $a < b \wedge a \leq b$ **by** *auto*

from $\langle \text{char } M1 = \text{char } M2 \rangle$

$M1.\text{Levy-Inversion}$ [*OF* $\langle a \leq b \rangle \langle \text{measure } M1\ \{a\} = 0 \rangle \langle \text{measure } M1\ \{b\}$

```

= 0⟩]
  M2.Levy-Inversion [OF ⟨a ≤ b⟩ ⟨measure M2 {a} = 0⟩ ⟨measure M2 {b}
= 0⟩]
  have complex-of-real (measure M1 {a<..b}) = complex-of-real (measure M2
{a<..b})
    by (intro LIMSEQ-unique) auto
  then have ?D a = ?D b
  unfolding of-real-eq-iff M1.cdf-diff-eq [OF ⟨a < b⟩, symmetric] M2.cdf-diff-eq
[OF ⟨a < b⟩, symmetric] by simp
  then have ?D x = |( ?D b - ?D x) - ?D a|
    by simp
  also have ... ≤ |?D b - ?D x| + |?D a|
    by (rule abs-triangle-ineq4)
  also have ... ≤ ε / 2 + ε / 2
    using 1 2 by (intro add-mono) auto
  finally have ?D x ≤ ε by simp }
  then show cdf M1 x = cdf M2 x
    by (metis abs-le-zero-iff dense-ge eq-iff-diff-eq-0)
qed
thus ?thesis
  by (rule cdf-unique [OF ⟨real-distribution M1⟩ ⟨real-distribution M2⟩])
qed

```

18.2 The Levy continuity theorem

theorem *levy-continuity1*:

```

fixes M :: nat ⇒ real measure and M' :: real measure
assumes ∧n. real-distribution (M n) real-distribution M' weak-conv-m M M'
shows (λn. char (M n) t) ⟶ char M' t
unfolding char-def using assms by (rule weak-conv-imp-integral-bdd-continuous-conv)
auto

```

theorem *levy-continuity*:

```

fixes M :: nat ⇒ real measure and M' :: real measure
assumes real-distr-M : ∧n. real-distribution (M n)
  and real-distr-M': real-distribution M'
  and char-conv: ∧t. (λn. char (M n) t) ⟶ char M' t
shows weak-conv-m M M'

```

proof –

```

interpret Mn: real-distribution M n for n by fact
interpret M': real-distribution M' by fact

```

```

have *: ∧u x. u > 0 ⟹ x ≠ 0 ⟹ (CLBINT t:{-u..u}. 1 - iexp (t * x)) =
  2 * (u - sin (u * x) / x)

```

proof –

```

fix u :: real and x :: real
assume u > 0 and x ≠ 0
hence (CLBINT t:{-u..u}. 1 - iexp (t * x)) = (CLBINT t=-u..u. 1 - iexp
(t * x))

```

```

    by (subst interval-integral-Icc, auto)
    also have ... = (CLBINT t=-u..ereal 0. 1 - iexp (t * x)) + (CLBINT t=
ereal 0..u. 1 - iexp (t * x))
    using ⟨u > 0⟩
    by (subst interval-integral-sum; force simp add: interval-integrable-isCont)
    also have ... = (CLBINT t=ereal 0..u. 1 - iexp (t * -x)) + (CLBINT t=ereal
0..u. 1 - iexp (t * x))
    by (subst interval-integral-reflect, auto)
    also have ... = CLBINT xa=ereal 0..ereal u. 1 - iexp (xa * -x) + (1 - iexp
(xa * x))
    by (subst interval-lebesgue-integral-add (2) [symmetric]) (auto simp: inter-
val-integrable-isCont)
    also have ... = (LBINT t=ereal 0..u. 2 - 2 * cos (t * x))
    unfolding exp-Euler cos-of-real by (simp flip: interval-lebesgue-integral-of-real)
    also have ... = 2 * u - 2 * sin (u * x) / x
    by (subst interval-lebesgue-integral-diff)
    (auto intro!: interval-integrable-isCont
      simp: interval-lebesgue-integral-of-real integral-cos [OF ⟨x ≠ 0⟩]
      mult.commute[of - x])
    finally show (CLBINT t:{-u..u}. 1 - iexp (t * x)) = 2 * (u - sin (u * x)
/ x)
    by (simp add: field-simps)
  qed
  have main-bound:  $\bigwedge u n. u > 0 \implies \text{Re } (\text{CLBINT } t:\{-u..u\}. 1 - \text{char } (M\ n)\ t)$ 
 $\geq$ 
    u * measure (M n) {x. abs x  $\geq$  2 / u}
  proof -
    fix u :: real and n
    assume u > 0
    interpret P: pair-sigma-finite M n lborel ..

    have Mn1 [simp]: measure (M n) UNIV = 1 by (metis Mn.prob-space Mn.space-eq-univ)

    have Mn2 [simp]:  $\bigwedge x. \text{complex-integrable } (M\ n) (\lambda t. \text{exp } (i * \text{complex-of-real }
(x * t)))$ 
    by (rule Mn.integrable-const-bound [where B = 1], auto)
    have Mn3: set-integrable (M n  $\otimes_M$  lborel) (UNIV  $\times$  {- u..u}) ( $\lambda a. 1 - \text{exp }
(i * \text{complex-of-real } (\text{snd } a * \text{fst } a))$ )
    using ⟨0 < u⟩
    unfolding set-integrable-def
    by (intro integrableI-bounded-set-indicator [where B=2])
    (auto simp: lborel.emeasure-pair-measure-Times ennreal-mult-less-top not-less
top-unique
      split: split-indicator
      intro!: order-trans [OF norm-triangle-ineq4])
    have (CLBINT t:{-u..u}. 1 - char (M n) t) =
      (CLBINT t:{-u..u}. (CLINT x | M n. 1 - iexp (t * x)))
    unfolding char-def by (rule set-lebesgue-integral-cong, auto simp del: of-real-mult)
    also have ... = (CLBINT t. (CLINT x | M n. indicator {-u..u} t *R (1 -

```



```

iexp (t * x)))
  unfolding set-lebesgue-integral-def
  by (rule Bochner-Integration.integral-cong) (auto split: split-indicator)
  also have ... = (CLINT x | M n. (CLBINT t:{-u..u}. 1 - iexp (t * x)))
  using Mn3 by (subst P.Fubini-integral) (auto simp: indicator-times split-beta'
set-integrable-def set-lebesgue-integral-def)
  also have ... = (CLINT x | M n. (if x = 0 then 0 else 2 * (u - sin (u * x)
/ x)))
  using ⟨u > 0⟩ by (intro Bochner-Integration.integral-cong, auto simp add: *
simp del: of-real-mult)
  also have ... = (LINT x | M n. (if x = 0 then 0 else 2 * (u - sin (u * x) /
x)))
  by (rule integral-complex-of-real)
  finally have Re (CLBINT t:{-u..u}. 1 - char (M n) t) =
    (LINT x | M n. (if x = 0 then 0 else 2 * (u - sin (u * x) / x))) by simp
  also have ... ≥ (LINT x : {x. abs x ≥ 2 / u} | M n. u)
  proof -
    have complex-integrable (M n) (λx. CLBINT t:{-u..u}. 1 - iexp (snd (x, t)
* fst (x, t)))
      using Mn3 unfolding set-integrable-def set-lebesgue-integral-def
      by (intro P.integrable-fst) (simp add: indicator-times split-beta')
    hence complex-integrable (M n) (λx. if x = 0 then 0 else 2 * (u - sin (u *
x) / x))
      using ⟨u > 0⟩
      unfolding set-integrable-def
      by (subst Bochner-Integration.integrable-cong) (auto simp add: * simp del:
of-real-mult)
    hence **: integrable (M n) (λx. if x = 0 then 0 else 2 * (u - sin (u * x) /
x))
      unfolding complex-of-real-integrable-eq .
    have 2 * sin x ≤ x if 2 ≤ x for x :: real
      by (rule order-trans[OF - ⟨2 ≤ x⟩]) auto
    moreover have x ≤ 2 * sin x if x ≤ - 2 for x :: real
      by (rule order-trans[OF ⟨x ≤ - 2⟩]) auto
    moreover have x < 0 ⟹ x ≤ sin x for x :: real
      using sin-x-le-x[of -x] by simp
    ultimately show ?thesis
      using ⟨u > 0⟩ unfolding set-lebesgue-integral-def
      by (intro integral-mono [OF - **])
        (auto simp: divide-simps sin-x-le-x mult.commute[of u] mult-neg-pos
top-unique less-top[symmetric]
split: split-indicator)
  qed
  also (xtrans) have (LINT x : {x. abs x ≥ 2 / u} | M n. u) = u * measure (M
n) {x. abs x ≥ 2 / u}
    unfolding set-lebesgue-integral-def
    by (simp add: Mn.emmeasure-eq-measure)
  finally show Re (CLBINT t:{-u..u}. 1 - char (M n) t) ≥ u * measure (M
n) {x. abs x ≥ 2 / u} .

```

```

qed

have tight-aux:  $\bigwedge \varepsilon. \varepsilon > 0 \implies \exists a\ b. a < b \wedge (\forall n. 1 - \varepsilon < \text{measure } (M\ n) \{a <..b\})$ 
proof -
  fix  $\varepsilon :: \text{real}$ 
  assume  $\varepsilon > 0$ 
  with  $M'.\text{isCont-char } [of\ 0]$ 
  obtain  $d$  where  $d0: d > 0$  and  $\forall x'. \text{dist } x' 0 < d \implies \text{dist } (\text{char } M' x') (\text{char } M' 0) < \varepsilon/4$ 
  unfolding continuous-at-eps-delta by (metis ‹ $0 < \varepsilon$ › divide-pos-pos zero-less-numeral)
  then have  $d1: \bigwedge t. \text{abs } t < d \implies \text{cmod } (\text{char } M' t - 1) < \varepsilon / 4$ 
    by (simp add:  $M'.\text{char-zero dist-norm}$ )
  have  $1: \bigwedge x. \text{cmod } (1 - \text{char } M' x) \leq 2$ 
    by (rule order-trans [OF norm-triangle-ineq4], auto simp add:  $M'.\text{cmod-char-le-1}$ )
  then have  $2: \bigwedge u\ v. \text{complex-set-integrable lborel } \{u..v\} (\lambda x. 1 - \text{char } M' x)$ 
    unfolding set-integrable-def
    by (intro integrableI-bounded-set-indicator[where  $B=2$ ]) (auto simp: emeasure-lborel-Icc-eq)
  have  $3: \bigwedge u\ v. \text{integrable lborel } (\lambda x. \text{indicat-real } \{u..v\} x *_R \text{cmod } (1 - \text{char } M' x))$ 
    by (intro borel-integrable-compact[OF compact-Icc] continuous-at-imp-continuous-on continuous-intros ballI  $M'.\text{isCont-char continuous-intros}$ )
  have  $\text{cmod } (\text{CLBINT } t: \{-d/2..d/2\}. 1 - \text{char } M' t) \leq (\text{LBINT } t: \{-d/2..d/2\}. \text{cmod } (1 - \text{char } M' t))$ 
    unfolding set-lebesgue-integral-def
    using integral-norm-bound[of -  $\lambda x. \text{indicat-real } \{u..v\} x *_R (1 - \text{char } M' x)$ ]
  for  $u\ v$ ] by simp
  also have  $4: \dots \leq (\text{LBINT } t: \{-d/2..d/2\}. \varepsilon / 4)$ 
    unfolding set-lebesgue-integral-def
  proof (rule integral-mono [OF 3])

    show  $\text{indicat-real } \{-d/2..d/2\} x *_R \text{cmod } (1 - \text{char } M' x) \leq \text{indicat-real } \{-d/2..d/2\} x *_R (\varepsilon / 4)$ 
      if  $x \in \text{space lborel}$  for  $x$ 
    proof (cases  $x \in \{-d/2..d/2\}$ )
    case True
    show ?thesis
      using  $d0\ d1$  that True [simplified]
      by (smt (verit, best) field-sum-of-halves minus-diff-eq norm-minus-cancel indicator-pos-le scaleR-left-mono)
    qed auto
    qed (simp add: emeasure-lborel-Icc-eq)
  also from  $d0\ 4$  have  $\dots = d * \varepsilon / 4$ 
    unfolding set-lebesgue-integral-def by simp
  finally have  $\text{bound: cmod } (\text{CLBINT } t: \{-d/2..d/2\}. 1 - \text{char } M' t) \leq d * \varepsilon / 4$ 
  have  $\text{cmod } (1 - \text{char } (M\ n) x) \leq 2$  for  $n\ x$ 
    by (rule order-trans [OF norm-triangle-ineq4], auto simp add:  $Mn.\text{cmod-char-le-1}$ )

```

then have $(\lambda n. \text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) \longrightarrow (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)$
unfolding *set-lebesgue-integral-def*
apply (*intro integral-dominated-convergence* [**where** $w = \lambda x. \text{indicator } \{-d/2..d/2\} \ x \ *_R \ 2$])
apply (*auto intro! char-conv tendsto-intros*
simp: emeasure-lborel-Icc-eq
split: split-indicator)
done
hence eventually $(\lambda n. \text{cmod } ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)) < d * \varepsilon / 4)$ *sequentially*
using $d0 \ \langle \varepsilon > 0 \rangle$ **apply** (*subst (asm) tendsto-iff*)
by (*subst (asm) dist-complex-def, drule spec, erule mp, auto*)
hence $\exists N. \forall n \geq N. \text{cmod } ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)) < d * \varepsilon / 4$ **by** (*simp add: eventually-sequentially*)
then obtain N
where $\forall n \geq N. \text{cmod } ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)) < d * \varepsilon / 4$..
hence $N: \bigwedge n. n \geq N \implies \text{cmod } ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)) < d * \varepsilon / 4$ **by** *auto*
{ fix n
assume $n \geq N$
have $\text{cmod } (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) = \text{cmod } ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)) + (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)$ **by** *simp*
also have $\dots \leq \text{cmod } ((\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) - (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)) + \text{cmod}(\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } M' \ t)$
by (*rule norm-triangle-ineq*)
also have $\dots < d * \varepsilon / 4 + d * \varepsilon / 4$
by (*rule add-less-le-mono [OF N [OF $\langle n \geq N \rangle$ bound]]*)
also have $\dots = d * \varepsilon / 2$ **by** *auto*
finally have $\text{cmod } (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t) < d * \varepsilon / 2$.
hence $d * \varepsilon / 2 > \text{Re } (\text{CLBINT } t:\{-d/2..d/2\}. 1 - \text{char } (M \ n) \ t)$
by (*rule order-le-less-trans [OF complex-Re-le-cmod]*)
hence $d * \varepsilon / 2 > \text{Re } (\text{CLBINT } t:\{-(d/2)..d/2\}. 1 - \text{char } (M \ n) \ t)$ (**is -**
> ?lhs) **by** *simp*
also have $?lhs \geq (d / 2) * \text{measure } (M \ n) \ \{x. \text{abs } x \geq 2 / (d / 2)\}$
using $d0$ **by** (*intro main-bound, simp*)
finally (*xtrans*) **have** $d * \varepsilon / 2 > (d / 2) * \text{measure } (M \ n) \ \{x. \text{abs } x \geq 2 / (d / 2)\}$.
with $d0 \ \langle \varepsilon > 0 \rangle$ **have** $\varepsilon > \text{measure } (M \ n) \ \{x. \text{abs } x \geq 2 / (d / 2)\}$ **by** (*simp add: field-simps*)
hence $\varepsilon > 1 - \text{measure } (M \ n) \ (\text{UNIV} - \{x. \text{abs } x \geq 2 / (d / 2)\})$
apply (*subst Mn.borel-UNIV [symmetric]*)

```

    by (subst Mn.prob-compl, auto)
  also have UNIV = {x. abs x ≥ 2 / (d / 2)} = {x. -(4 / d) < x ∧ x < (4 / d)}
    using d0 by (simp add: set-eq-iff divide-simps abs-if) (smt (verit, best)
mult-less-0-iff)
  finally have measure (M n) {x. -(4 / d) < x ∧ x < (4 / d)} > 1 - ε
    by auto
} note 6 = this
{ fix n :: nat
  have *: (UN (k :: nat). {− real k <.. real k}) = UNIV
    by (auto, metis leI le-less-trans less-imp-le minus-less-iff reals-Archimedean2)
  have (λk. measure (M n) {− real k <.. real k}) ⟶
    measure (M n) (UN (k :: nat). {− real k <.. real k})
    by (rule Mn.finite-Lim-measure-incseq, auto simp add: incseq-def)
  hence (λk. measure (M n) {− real k <.. real k}) ⟶ 1
    using Mn.prob-space unfolding * Mn.borel-UNIV by simp
  hence eventually (λk. measure (M n) {− real k <.. real k} > 1 - ε) sequentially
    using ⟨ε > 0⟩ order-tendstoD by fastforce
} note 7 = this
{ fix n :: nat
  have eventually (λk. ∀ m < n. measure (M m) {− real k <.. real k} > 1 - ε)
sequentially
    (is ?P n)
  proof (induct n)
    case (Suc n) with 7[of n] show ?case
      by eventually-elim (auto simp add: less-Suc-eq)
    qed simp
  } note 8 = this
  from 8 [of N] have ∃ K :: nat. ∀ k ≥ K. ∀ m < N. 1 - ε <
    Sigma-Algebra.measure (M m) {− real k <.. real k}
    by (auto simp add: eventually-sequentially)
  hence ∃ K :: nat. ∀ m < N. 1 - ε < Sigma-Algebra.measure (M m) {− real
K <.. real K} by auto
  then obtain K :: nat where
    ∀ m < N. 1 - ε < Sigma-Algebra.measure (M m) {− real K <.. real K} ..
  hence K: ⋀ m. m < N ⟹ 1 - ε < Sigma-Algebra.measure (M m) {− real
K <.. real K}
    by auto
  let ?K' = max K (4 / d)
  have 1 - ε < measure (M n) {− max (real K) (4 / d) <.. max (real K) (4 /
d)} for n
  proof (cases n < N)
    case True
      then show ?thesis
        by (force intro: order-less-le-trans [OF K Mn.finite-measure-mono])
    next
      case False
      then show ?thesis
        by (force intro: order-less-le-trans [OF 6 Mn.finite-measure-mono])
  end

```

```

qed
then have  $-?K' < ?K' \wedge (\forall n. 1 - \varepsilon < \text{measure } (M \ n) \ \{-?K' < .. ?K'\})$ 
  using  $d0$  by (simp add: less-max-iff-disj minus-less-iff)
thus  $\exists a \ b. a < b \wedge (\forall n. 1 - \varepsilon < \text{measure } (M \ n) \ \{a < .. b\})$  by (intro exI)
qed
have tight: tight  $M$ 
  by (auto simp: tight-def intro: assms tight-aux)
show ?thesis
proof (rule tight-subseq-weak-converge [OF real-distr- $M$  real-distr- $M'$  tight])
  fix  $s \ \nu$ 
  assume  $s$ : strict-mono ( $s :: \text{nat} \Rightarrow \text{nat}$ )
  assume  $nu$ : weak-conv-m ( $M \circ s$ )  $\nu$ 
  assume *: real-distribution  $\nu$ 
  have 2:  $\bigwedge n. \text{real-distribution } ((M \circ s) \ n)$  unfolding comp-def by (rule assms)
  have 3:  $\bigwedge t. (\lambda n. \text{char } ((M \circ s) \ n) \ t) \longrightarrow \text{char } \nu \ t$  by (intro levy-continuity1 [OF 2 *  $nu$ ])
  have 4:  $\bigwedge t. (\lambda n. \text{char } ((M \circ s) \ n) \ t) = ((\lambda n. \text{char } (M \ n) \ t) \circ s)$  by (rule ext, simp)
  have 5:  $\bigwedge t. (\lambda n. \text{char } ((M \circ s) \ n) \ t) \longrightarrow \text{char } M' \ t$ 
    by (subst 4, rule LIMSEQ-subseq-LIMSEQ [OF -  $s$ ], rule assms)
  hence  $\text{char } \nu = \text{char } M'$  by (intro ext, intro LIMSEQ-unique [OF 3 5])
  hence  $\nu = M'$  by (rule Levy-uniqueness [OF * (real-distribution  $M'$ )])
  thus weak-conv-m ( $M \circ s$ )  $M'$ 
    by (elim subst) (rule nu)
qed
qed
end

```

19 The Central Limit Theorem

theory Central-Limit-Theorem

imports Levy

begin

theorem (in prob-space) central-limit-theorem-zero-mean:

fixes $X :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

and $\mu :: \text{real measure}$

and $\sigma :: \text{real}$

and $S :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$

assumes X -indep: indep-vars ($\lambda i. \text{borel } X \ i$) UNIV

and X -mean-0: $\bigwedge n. \text{expectation } (X \ n) = 0$

and σ -pos: $\sigma > 0$

and X -square-integrable: $\bigwedge n. \text{integrable } M \ (\lambda x. (X \ n \ x)^2)$

and X -variance: $\bigwedge n. \text{variance } (X \ n) = \sigma^2$

and X -distrib: $\bigwedge n. \text{distr } M \ \text{borel } (X \ n) = \mu$

defines $S \ n \equiv \lambda x. \sum_{i < n. X \ i \ x}$

shows weak-conv-m ($\lambda n. \text{distr } M \ \text{borel } (\lambda x. S \ n \ x / \text{sqrt } (n * \sigma^2))$) std-normal-distribution

proof –

```

let ?S' =  $\lambda n x. S\ n\ x\ /\ \text{sqrt}\ (\text{real}\ n * \sigma^2)$ 
define  $\varphi$  where  $\varphi\ n = \text{char}\ (\text{distr}\ M\ \text{borel}\ ( ?S'\ n))$  for  $n$ 
define  $\psi$  where  $\psi\ n\ t = \text{char}\ \mu\ (t\ /\ \text{sqrt}\ (\sigma^2 * n))$  for  $n\ t$ 

have  $X\text{-rv}$  [simp, measurable]: random-variable borel ( $X\ n$ ) for  $n$ 
  using  $X\text{-indep}$  unfolding indep-vars-def2 by simp
have  $X\text{-integrable}$  [simp, intro]: integrable  $M\ (X\ n)$  for  $n$ 
  by (rule square-integrable-imp-integrable[OF - X-square-integrable]) simp-all
interpret  $\mu$ : real-distribution  $\mu$ 
  by (subst  $X\text{-distrib}$  [symmetric, of 0], rule real-distribution-distr, simp)

have  $\mu\text{-integrable}$  [simp]: integrable  $\mu\ (\lambda x. x)$ 
  and  $\mu\text{-mean-integrable}$  [simp]:  $\mu.\text{expectation}\ (\lambda x. x) = 0$ 
  and  $\mu\text{-square-integrable}$  [simp]: integrable  $\mu\ (\lambda x. x^2)$ 
  and  $\mu\text{-variance}$  [simp]:  $\mu.\text{expectation}\ (\lambda x. x^2) = \sigma^2$ 
  using assms by (simp-all add:  $X\text{-distrib}$  [symmetric, of 0] integrable-distr-eq
integral-distr)

have main:  $\forall_F\ n$  in sequentially.
   $\text{cmod}\ (\varphi\ n\ t - (1 + (t^2 / 2) / n)^n) \leq$ 
   $t^2 / (6 * \sigma^2) * (\text{LINT}\ x | \mu. \min\ (6 * x^2)\ (|t / \text{sqrt}\ (\sigma^2 * n)| * |x| ^ 3))$  for  $t$ 
proof (rule eventually-sequentiallyI)
  fix  $n :: \text{nat}$ 
  assume  $n \geq \text{nat}\ (\text{ceiling}\ (t^2 / 4))$ 
  hence  $n: n \geq t^2 / 4$  by (subst nat-ceiling-le-eq [symmetric])
  let  $?t = t / \text{sqrt}\ (\sigma^2 * n)$ 

  define  $\psi'$  where  $\psi'\ n\ i = \text{char}\ (\text{distr}\ M\ \text{borel}\ (\lambda x. X\ i\ x\ /\ \text{sqrt}\ (\sigma^2 * n)))$  for
   $n\ i$ 
  have  $*$ :  $\bigwedge n\ i\ t. \psi'\ n\ i\ t = \psi\ n\ t$ 
    unfolding  $\psi\text{-def}$   $\psi'\text{-def}$  char-def
    by (subst  $X\text{-distrib}$  [symmetric]) (auto simp: integral-distr)

  have  $\varphi\ n\ t = \text{char}\ (\text{distr}\ M\ \text{borel}\ (\lambda x. \sum i < n. X\ i\ x\ /\ \text{sqrt}\ (\sigma^2 * \text{real}\ n)))\ t$ 
    by (auto simp:  $\varphi\text{-def}$   $S\text{-def}$  sum-divide-distrib ac-simps)
  also have  $\dots = (\prod i < n. \psi'\ n\ i\ t)$ 
    unfolding  $\psi'\text{-def}$ 
    apply (rule char-distr-sum)
    apply (rule indep-vars-compose2[where  $X=X$ ])
    apply (rule indep-vars-subset)
    apply (rule X-indep)
    apply auto
    done
  also have  $\dots = (\psi\ n\ t)^n$ 
    by (auto simp add:  $*$  prod-constant)
  finally have  $\varphi\text{-eq}$ :  $\varphi\ n\ t = (\psi\ n\ t)^n$  .

  have norm  $(\psi\ n\ t - (1 - ?t^2 * \sigma^2 / 2)) \leq ?t^2 / 6 * (\text{LINT}\ x | \mu. \min\ (6 *$ 

```

x^2) ($|?t| * |x| \wedge 3$)
unfolding ψ -def **by** (rule μ .char-approx3, auto)
also have $?t^2 * \sigma^2 = t^2 / n$
using σ -pos **by** (simp add: power-divide)
also have $t^2 / n / 2 = (t^2 / 2) / n$
by simp
finally have **: norm ($\psi \ n \ t - (1 + (-(t^2) / 2) / n) \wedge n$) \leq
 $?t^2 / 6 * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| \wedge 3))$ **by** simp

have norm ($\varphi \ n \ t - (\text{complex-of-real } (1 + (-(t^2) / 2) / n) \wedge n) \leq$
 $n * \text{norm } (\psi \ n \ t - (\text{complex-of-real } (1 + (-(t^2) / 2) / n)))$
using n
by (auto intro!: norm-power-diff μ .cmod-char-le-1 abs-leI
simp del: of-real-diff simp: of-real-diff[symmetric] divide-le-eq φ -eq
 ψ -def)
also have ... $\leq n * (?t^2 / 6 * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| \wedge 3)))$
by (rule mult-left-mono [OF **], simp)
also have ... $= (t^2 / (6 * \sigma^2)) * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| \wedge 3))$
using σ -pos **by** (simp add: field-simps min-absorb2)
finally show norm ($\varphi \ n \ t - (1 + (-(t^2) / 2) / n) \wedge n$) \leq
 $(t^2 / (6 * \sigma^2)) * (LINT \ x|\mu. \min (6 * x^2) (|?t| * |x| \wedge 3))$
by simp
qed

show ?thesis
proof (rule levy-continuity)
fix t
let $?t = \lambda n. t / \text{sqrt } (\sigma^2 * n)$
have $\bigwedge x. (\lambda n. \min (6 * x^2) (|t| * |x| \wedge 3 / |\text{sqrt } (\sigma^2 * \text{real } n)|)) \longrightarrow 0$
using σ -pos
by (auto simp: real-sqrt-mult min-absorb2
intro!: tendsto-min[THEN tendsto-eq-rhs] sqrt-at-top[THEN filter-
lim-compose]
filterlim-tendsto-pos-mult-at-top filterlim-at-top-imp-at-infinity
tendsto-divide-0 filterlim-real-sequentially)
then have $(\lambda n. LINT \ x|\mu. \min (6 * x^2) (|?t \ n| * |x| \wedge 3)) \longrightarrow (LINT \ x|\mu. 0)$
by (intro integral-dominated-convergence [where $w = \lambda x. 6 * x^2$]) auto
then have *: $(\lambda n. t^2 / (6 * \sigma^2)) * (LINT \ x|\mu. \min (6 * x^2) (|?t \ n| * |x| \wedge 3)) \longrightarrow 0$
by (simp only: integral-zero tendsto-mult-right-zero)

have $(\lambda n. \text{complex-of-real } ((1 + (-(t^2) / 2) / n) \wedge n)) \longrightarrow \text{complex-of-real } (\exp (-(t^2) / 2))$
by (rule isCont-tendsto-compose [OF - tendsto-exp-limit-sequentially]) auto
then have $(\lambda n. \varphi \ n \ t) \longrightarrow \text{complex-of-real } (\exp (-(t^2) / 2))$
by (rule Lim-transform) (rule Lim-null-comparison [OF main *])
then show $(\lambda n. \text{char } (\text{distr } M \text{ borel } (?S' \ n)) \ t) \longrightarrow \text{char std-normal-distribution } t$

```

    by (simp add:  $\varphi$ -def char-std-normal-distribution)
  qed (auto intro!: real-dist-normal-dist simp: S-def)
qed

theorem (in prob-space) central-limit-theorem:
  fixes  $X :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$ 
  and  $\mu :: \text{real measure}$ 
  and  $\sigma :: \text{real}$ 
  and  $S :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$ 
  assumes  $X\text{-indep}$ : indep-vars ( $\lambda i.$  borel)  $X$  UNIV
  and  $X\text{-mean}$ :  $\bigwedge n.$  expectation ( $X\ n$ ) =  $m$ 
  and  $\sigma\text{-pos}$ :  $\sigma > 0$ 
  and  $X\text{-square-integrable}$ :  $\bigwedge n.$  integrable  $M$  ( $\lambda x.$  ( $X\ n\ x$ )2)
  and  $X\text{-variance}$ :  $\bigwedge n.$  variance ( $X\ n$ ) =  $\sigma^2$ 
  and  $X\text{-distrib}$ :  $\bigwedge n.$  distr  $M$  borel ( $X\ n$ ) =  $\mu$ 
  defines  $X' i\ x \equiv X\ i\ x - m$ 
  shows weak-conv-m ( $\lambda n.$  distr  $M$  borel ( $\lambda x.$  ( $\sum i < n.$   $X' i\ x$ ) / sqrt ( $n * \sigma^2$ )))
std-normal-distribution
proof (intro central-limit-theorem-zero-mean)
  show indep-vars ( $\lambda i.$  borel)  $X'$  UNIV
  unfolding  $X'\text{-def}$ [abs-def] using  $X\text{-indep}$  by (rule indep-vars-compose2) auto
  have  $X\text{-rv}$  [simp, measurable]: random-variable borel ( $X\ n$ ) for  $n$ 
  using  $X\text{-indep}$  unfolding indep-vars-def2 by simp
  have  $X\text{-integrable}$  [simp, intro]: integrable  $M$  ( $X\ n$ ) for  $n$ 
  by (rule square-integrable-imp-integrable[OF  $X\text{-square-integrable}$ ]) simp-all
  show expectation ( $X' n$ ) = 0 for  $n$ 
  using  $X\text{-mean}$  by (auto simp:  $X'\text{-def}$ [abs-def] prob-space)
  show  $\sigma > 0$  integrable  $M$  ( $\lambda x.$  ( $X' n\ x$ )2) variance ( $X' n$ ) =  $\sigma^2$  for  $n$ 
  using  $\langle 0 < \sigma \rangle$   $X\text{-integrable}$   $X\text{-mean}$   $X\text{-square-integrable}$   $X\text{-variance}$  unfolding
 $X'\text{-def}$ 
  by (auto simp: prob-space power2-diff)
  show distr  $M$  borel ( $X' n$ ) = distr  $\mu$  borel ( $\lambda x.$   $x - m$ ) for  $n$ 
  unfolding  $X\text{-distrib}$ [of  $n$ , symmetric] using  $X\text{-integrable}$ 
  by (subst distr-distr) (auto simp:  $X'\text{-def}$ [abs-def] comp-def)
qed

end

theory Discrete-Topology
imports HOL-Analysis.Analysis
begin

Copy of discrete types with discrete topology. This space is polish.

typedef 'a discrete = UNIV::'a set
morphisms of-discrete discrete
..

instantiation discrete :: (type) metric-space

```


begin

definition *dist-discrete* :: 'a discrete \Rightarrow 'a discrete \Rightarrow real
where *dist-discrete* *n m* = (if *n* = *m* then 0 else 1)

definition *uniformity-discrete* :: ('a discrete \times 'a discrete) filter **where**
 (uniformity::('a discrete \times 'a discrete) filter) = (INF *e* \in {0 <.. ∞ }. principal {(*x*,
y). *dist* *x y* < *e*})

definition *open-discrete* :: 'a discrete set \Rightarrow bool **where**
 (open::'a discrete set \Rightarrow bool) *U* \longleftrightarrow ($\forall x \in U$. eventually ($\lambda(x', y)$. $x' = x \longrightarrow y \in U$) uniformity))

instance proof qed (auto simp: uniformity-discrete-def open-discrete-def dist-discrete-def
 intro: exI[where *x*=1])
end

lemma *open-discrete*: open (*S* :: 'a discrete set)
unfolding *open-dist* *dist-discrete-def* **by** (auto intro!: exI[of - 1 / 2])

instance *discrete* :: (type) complete-space
proof
fix *X*::nat \Rightarrow 'a discrete
assume *Cauchy* *X*
then obtain *n* **where** $\forall m \geq n$. *X* *m* = *X* *m*
by (force simp: dist-discrete-def *Cauchy-def* *split*: if-split-asm *dest*:spec[where
x=1])
thus *convergent* *X*
by (intro *convergentI*[where *L*=*X* *n*] *tendstoI* eventually-sequentiallyI[of *n*])
 (simp add: dist-discrete-def)
qed

instance *discrete* :: (countable) countable
proof
have *inj* (λc ::'a discrete. to-nat (of-discrete *c*))
by (simp add: inj-on-def of-discrete-inject)
thus $\exists f$::'a discrete \Rightarrow nat. *inj* *f* **by** blast
qed

instance *discrete* :: (countable) second-countable-topology
proof
let ?*B* = range (λn ::'a discrete. {*n*})
have $\bigwedge S$. generate-topology ?*B* ($\bigcup x \in S$. {*x*})
by (intro generate-topology-Union) (auto intro: generate-topology.intros)
then have *open* = generate-topology ?*B*
by (auto intro!: ext simp: open-discrete)
moreover have countable ?*B* **by** simp
ultimately show $\exists B$::'a discrete set set. countable *B* \wedge *open* = generate-topology
B **by** blast

qed

instance discrete :: (countable) polish-space ..

end

20 Probability mass function

theory Probability-Mass-Function

imports

Giry-Monad

HOL-Library.Multiset

begin

Conflicting notation from *HOL-Analysis.Infinite-Sum*

no-notation *Infinite-Sum.abs-summable-on* (infixr ‹abs'-summable'-on› 46)

lemma AE-emeasure-singleton:

assumes x : *emeasure* $M \{x\} \neq 0$ and ae : $AE\ x\ in\ M.$ $P\ x$ shows $P\ x$

proof –

from x have x - M : $\{x\} \in sets\ M$

by (auto intro: *emeasure-notin-sets*)

from ae obtain N where N : $\{x \in space\ M. \neg P\ x\} \subseteq N$ *emeasure* $M\ N = 0$ $N \in sets\ M$

by (auto elim: *AE-E*)

{ assume $\neg P\ x$

with x - M [*THEN sets.sets-into-space*] N have *emeasure* $M \{x\} \leq$ *emeasure* $M\ N$

by (intro *emeasure-mono*) auto

with $x\ N$ have *False*

by (auto simp:) }

then show $P\ x$ by auto

qed

lemma *AE-measure-singleton*: *measure* $M \{x\} \neq 0 \implies AE\ x\ in\ M.$ $P\ x \implies P\ x$

by (metis *AE-emeasure-singleton measure-def emeasure-empty measure-empty*)

lemma (in *finite-measure*) *AE-support-countable*:

assumes [*simp*]: *sets* $M = UNIV$

shows ($AE\ x\ in\ M.$ *measure* $M \{x\} \neq 0$) $\longleftrightarrow (\exists S.$ *countable* $S \wedge (AE\ x\ in\ M.$ $x \in S)$)

proof

assume $\exists S.$ *countable* $S \wedge (AE\ x\ in\ M.$ $x \in S)$

then obtain S where S [*intro*]: *countable* S and ae : $AE\ x\ in\ M.$ $x \in S$

by auto

then have *emeasure* $M (\bigcup_{x \in \{x \in S. \text{emeasure } M \{x\} \neq 0\}} \{x\}) =$

$(\int^+ x. \text{emeasure } M \{x\} * \text{indicator } \{x \in S. \text{emeasure } M \{x\} \neq 0\} x\ \partial \text{count-space } UNIV)$

by (*subst emeasure-UN-countable*)

```

    (auto simp: disjoint-family-on-def nn-integral-restrict-space[symmetric] re-
strict-count-space)
  also have ... = ( $\int^+ x. \text{emeasure } M \{x\} * \text{indicator } S \, x \, \partial \text{count-space } UNIV$ )
    by (auto intro!: nn-integral-cong split: split-indicator)
  also have ... =  $\text{emeasure } M (\bigcup_{x \in S. \{x\}}$ )
    by (subst emeasure-UN-countable)
    (auto simp: disjoint-family-on-def nn-integral-restrict-space[symmetric] re-
strict-count-space)
  also have ... =  $\text{emeasure } M (\text{space } M)$ 
    using ae by (intro emeasure-eq-AE) auto
  finally have  $\text{emeasure } M \{x \in \text{space } M. x \in S \wedge \text{emeasure } M \{x\} \neq 0\} =$ 
 $\text{emeasure } M (\text{space } M)$ 
    by (simp add: emeasure-single-in-space cong: rev-conj-cong)
  with finite-measure-compl[of  $\{x \in \text{space } M. x \in S \wedge \text{emeasure } M \{x\} \neq 0\}$ ]
  have AE  $x$  in  $M. x \in S \wedge \text{emeasure } M \{x\} \neq 0$ 
    by (intro AE-I[OF order-refl]) (auto simp: emeasure-eq-measure measure-nonneg
set-diff-eq cong: conj-cong)
  then show AE  $x$  in  $M. \text{measure } M \{x\} \neq 0$ 
    by (auto simp: emeasure-eq-measure)
qed (auto intro!: exI[of -  $\{x. \text{measure } M \{x\} \neq 0\}$ ] countable-support)

```

20.1 PMF as measure

```

typedef 'a pmf =  $\{M :: 'a \text{ measure. prob-space } M \wedge \text{sets } M = UNIV \wedge (AE \, x \text{ in } M. \text{measure } M \{x\} \neq 0)\}$ 
morphisms measure-pmf Abs-pmf
by (intro exI[of - uniform-measure (count-space UNIV) {undefined}])
    (auto intro!: prob-space-uniform-measure AE-uniform-measureI)

declare [[coercion measure-pmf]]

lemma prob-space-measure-pmf: prob-space (measure-pmf p)
using pmf.measure-pmf[of p] by auto

interpretation measure-pmf: prob-space measure-pmf M for M
by (rule prob-space-measure-pmf)

interpretation measure-pmf: subprob-space measure-pmf M for M
by (rule prob-space-imp-subprob-space) unfold-locales

lemma subprob-space-measure-pmf: subprob-space (measure-pmf x)
by unfold-locales

locale pmf-as-measure
begin

setup-lifting type-definition-pmf

end

```

```

context
begin

interpretation pmf-as-measure .

lemma sets-measure-pmf[simp]: sets (measure-pmf p) = UNIV
  by transfer blast

lemma sets-measure-pmf-count-space[measurable-cong]:
  sets (measure-pmf M) = sets (count-space UNIV)
  by simp

lemma space-measure-pmf[simp]: space (measure-pmf p) = UNIV
  using sets-eq-imp-space-eq[of measure-pmf p count-space UNIV] by simp

lemma measure-pmf-UNIV [simp]: measure (measure-pmf p) UNIV = 1
using measure-pmf.prob-space[of p] by simp

lemma measure-pmf-in-subprob-algebra[measurable (raw)]: measure-pmf x ∈ space
(subprob-algebra (count-space UNIV))
  by (simp add: space-subprob-algebra subprob-space-measure-pmf)

lemma measurable-pmf-measure1[simp]: measurable (M :: 'a pmf) N = UNIV →
space N
  by (auto simp: measurable-def)

lemma measurable-pmf-measure2[simp]: measurable N (M :: 'a pmf) = measurable
N (count-space UNIV)
  by (intro measurable-cong-sets simp-all)

lemma measurable-pair-restrict-pmf2:
  assumes countable A
  assumes [measurable]:  $\bigwedge y. y \in A \implies (\lambda x. f(x, y)) \in \text{measurable } M \ L$ 
  shows  $f \in \text{measurable } (M \otimes_M \text{restrict-space (measure-pmf } N) \ A) \ L$  (is  $f \in$ 
measurable ?M -)
proof –
  have [measurable-cong]: sets (restrict-space (count-space UNIV) A) = sets (count-space
A)
    by (simp add: restrict-count-space)

  show ?thesis
    by (intro measurable-compose-countable'[where f=λa b. f (fst b, a) and g=snd
and  $I=A,$ 
unfolded prod.collapse] assms)
    measurable
qed

lemma measurable-pair-restrict-pmf1:

```

assumes *countable* A
assumes [*measurable*]: $\bigwedge x. x \in A \implies (\lambda y. f(x, y)) \in \text{measurable } N \ L$
shows $f \in \text{measurable } (\text{restrict-space } (\text{measure-pmf } M) \ A \ \otimes_M \ N) \ L$
proof –
have [*measurable-cong*]: $\text{sets } (\text{restrict-space } (\text{count-space } \text{UNIV}) \ A) = \text{sets } (\text{count-space } A)$
by (*simp add: restrict-count-space*)

show ?thesis
by (*intro measurable-compose-countable'* [**where** $f = \lambda a \ b. f(a, \text{snd } b)$ **and** $g = \text{fst}$ **and** $I = A$,
unfolded prod.collapse *assms*)
measurable
qed

lift-definition *pmf* :: $'a \ \text{pmf} \Rightarrow 'a \Rightarrow \text{real}$ **is** $\lambda M \ x. \text{measure } M \ \{x\}$.

lift-definition *set-pmf* :: $'a \ \text{pmf} \Rightarrow 'a \ \text{set}$ **is** $\lambda M. \{x. \text{measure } M \ \{x\} \neq 0\}$.
declare [*coercion set-pmf*]

lemma *AE-measure-pmf*: $AE \ x \text{ in } (M :: 'a \ \text{pmf}). \ x \in M$
by *transfer simp*

lemma *emeasure-pmf-single-eq-zero-iff*:
fixes $M :: 'a \ \text{pmf}$
shows $\text{emeasure } M \ \{y\} = 0 \iff y \notin M$
unfolding *set-pmf.rep-eq* **by** (*simp add: measure-pmf.emeasure-eq-measure*)

lemma *AE-measure-pmf-iff*: $(AE \ x \text{ in } \text{measure-pmf } M. \ P \ x) \iff (\forall y \in M. \ P \ y)$
using *AE-measure-singleton[of M]* *AE-measure-pmf[of M]*
by (*auto simp: set-pmf.rep-eq*)

lemma *AE-pmfI*: $(\bigwedge y. y \in \text{set-pmf } M \implies P \ y) \implies \text{almost-everywhere } (\text{measure-pmf } M) \ P$
by (*simp add: AE-measure-pmf-iff*)

lemma *countable-set-pmf* [*simp*]: *countable* (*set-pmf* p)
by *transfer (metis prob-space.finite-measure finite-measure.countable-support)*

lemma *pmf-positive*: $x \in \text{set-pmf } p \implies 0 < \text{pmf } p \ x$
by *transfer (simp add: less-le)*

lemma *pmf-nonneg* [*simp*]: $0 \leq \text{pmf } p \ x$
by *transfer simp*

lemma *pmf-not-neg* [*simp*]: $\neg \text{pmf } p \ x < 0$
by (*simp add: not-less pmf-nonneg*)

lemma *pmf-pos* [*simp*]: $\text{pmf } p \ x \neq 0 \implies \text{pmf } p \ x > 0$

```

using pmf-nonneg[of p x] by linarith

lemma pmf-le-1: pmf p x ≤ 1
  by (simp add: pmf.rep-eq)

lemma set-pmf-not-empty: set-pmf M ≠ {}
  using AE-measure-pmf[of M] by (intro notI) simp

lemma set-pmf-iff: x ∈ set-pmf M ⟷ pmf M x ≠ 0
  by transfer simp

lemma pmf-positive-iff: 0 < pmf p x ⟷ x ∈ set-pmf p
  unfolding less-le by (simp add: set-pmf-iff)

lemma set-pmf-eq: set-pmf M = {x. pmf M x ≠ 0}
  by (auto simp: set-pmf-iff)

lemma set-pmf-eq': set-pmf p = {x. pmf p x > 0}
proof safe
  fix x assume x ∈ set-pmf p
  hence pmf p x ≠ 0 by (auto simp: set-pmf-eq)
  with pmf-nonneg[of p x] show pmf p x > 0 by simp
qed (auto simp: set-pmf-eq)

lemma emeasure-pmf-single:
  fixes M :: 'a pmf
  shows emeasure M {x} = pmf M x
  by transfer (simp add: finite-measure.emeasure-eq-measure[OF prob-space.finite-measure])

lemma measure-pmf-single: measure (measure-pmf M) {x} = pmf M x
  using emeasure-pmf-single[of M x] by (simp add: measure-pmf.emeasure-eq-measure
    pmf-nonneg measure-nonneg)

lemma emeasure-measure-pmf-finite: finite S ⟹ emeasure (measure-pmf M) S
  = (∑ s∈S. pmf M s)
  by (subst emeasure-eq-sum-singleton) (auto simp: emeasure-pmf-single pmf-nonneg)

lemma measure-measure-pmf-finite: finite S ⟹ measure (measure-pmf M) S =
  sum (pmf M) S
  using emeasure-measure-pmf-finite[of S M]
  by (simp add: measure-pmf.emeasure-eq-measure measure-nonneg sum-nonneg
    pmf-nonneg)

lemma sum-pmf-eq-1:
  assumes finite A set-pmf p ⊆ A
  shows (∑ x∈A. pmf p x) = 1
proof -
  have (∑ x∈A. pmf p x) = measure-pmf.prob p A
  by (simp add: measure-measure-pmf-finite assms)

```

also from *assms* have ... = 1
 by (subst measure-pmf.prob-eq-1) (auto simp: AE-measure-pmf-iff)
 finally show ?thesis .
 qed

lemma *nn-integral-measure-pmf-support*:

fixes $f :: 'a \Rightarrow \text{ennreal}$
 assumes f : finite A and nn : $\bigwedge x. x \in A \implies 0 \leq f\ x \bigwedge x. x \in \text{set-pmf } M \implies x \notin A \implies f\ x = 0$
 shows $(\int^+ x. f\ x\ \partial \text{measure-pmf } M) = (\sum_{x \in A}. f\ x * \text{pmf } M\ x)$
proof –
 have $(\int^+ x. f\ x\ \partial M) = (\int^+ x. f\ x * \text{indicator } A\ x\ \partial M)$
 using *nn* by (intro nn-integral-cong-AE) (auto simp: AE-measure-pmf-iff split: split-indicator)
 also have ... = $(\sum_{x \in A}. f\ x * \text{emeasure } M\ \{x\})$
 using *assms* by (intro nn-integral-indicator-finite) auto
 finally show ?thesis
 by (simp add: emeasure-measure-pmf-finite)
 qed

lemma *nn-integral-measure-pmf-finite*:

fixes $f :: 'a \Rightarrow \text{ennreal}$
 assumes f : finite (set-pmf M) and nn : $\bigwedge x. x \in \text{set-pmf } M \implies 0 \leq f\ x$
 shows $(\int^+ x. f\ x\ \partial \text{measure-pmf } M) = (\sum_{x \in \text{set-pmf } M}. f\ x * \text{pmf } M\ x)$
 using *assms* by (intro nn-integral-measure-pmf-support) auto

lemma *integrable-measure-pmf-finite*:

fixes $f :: 'a \Rightarrow 'b::\{\text{banach, second-countable-topology}\}$
 shows finite (set-pmf M) \implies integrable $M\ f$
 by (auto intro!: integrableI-bounded simp: nn-integral-measure-pmf-finite ennreal-mult-less-top)

lemma *integral-measure-pmf-real*:

assumes [*simp*]: finite A and $\bigwedge a. a \in \text{set-pmf } M \implies f\ a \neq 0 \implies a \in A$
 shows $(\int x. f\ x\ \partial \text{measure-pmf } M) = (\sum_{a \in A}. f\ a * \text{pmf } M\ a)$
proof –
 have $(\int x. f\ x\ \partial \text{measure-pmf } M) = (\int x. f\ x * \text{indicator } A\ x\ \partial \text{measure-pmf } M)$
 using *assms*(2) by (intro integral-cong-AE) (auto split: split-indicator simp: AE-measure-pmf-iff)
 also have ... = $(\sum_{a \in A}. f\ a * \text{pmf } M\ a)$
 by (subst integral-indicator-finite-real)
 (auto simp: measure-def emeasure-measure-pmf-finite pmf-nonneg)
 finally show ?thesis .
 qed

lemma *integrable-pmf*: integrable (count-space X) (pmf M)

proof –

have $(\int^+ x. \text{pmf } M\ x\ \partial \text{count-space } X) = (\int^+ x. \text{pmf } M\ x\ \partial \text{count-space } (M \cap X))$
 by (auto simp add: nn-integral-count-space-indicator set-pmf-iff intro!: nn-integral-cong)

split: split-indicator)
then have *integrable (count-space X) (pmf M) = integrable (count-space (M ∩ X)) (pmf M)*
by (*simp add: integrable-iff-bounded pmf-nonneg*)
then show *?thesis*
by (*simp add: pmf.rep-eq measure-pmf.integrable-measure disjoint-family-on-def*)
qed

lemma *integral-pmf: (∫ x. pmf M x ∂count-space X) = measure M X*
proof –
have *(∫ x. pmf M x ∂count-space X) = (∫ +x. pmf M x ∂count-space X)*
by (*simp add: pmf-nonneg integrable-pmf nn-integral-eq-integral*)
also have *... = (∫ +x. emeasure M {x} ∂count-space (X ∩ M))*
by (*auto intro!: nn-integral-cong-AE split: split-indicator*
simp: pmf.rep-eq measure-pmf.emeasure-eq-measure nn-integral-count-space-indicator
AE-count-space set-pmf-iff)
also have *... = emeasure M (X ∩ M)*
by (*rule emeasure-countable-singleton[symmetric]*) (*auto intro: countable-set-pmf*)
also have *... = emeasure M X*
by (*auto intro!: emeasure-eq-AE simp: AE-measure-pmf-iff*)
finally show *?thesis*
by (*simp add: measure-pmf.emeasure-eq-measure measure-nonneg integral-nonneg*
pmf-nonneg)
qed

lemma *integral-pmf-restrict:*
(f::'a ⇒ 'b::{banach, second-countable-topology}) ∈ borel-measurable (count-space UNIV) ⇒
(∫ x. f x ∂measure-pmf M) = (∫ x. f x ∂restrict-space M M)
by (*auto intro!: integral-cong-AE simp add: integral-restrict-space AE-measure-pmf-iff*)

lemma *emeasure-pmf: emeasure (M::'a pmf) M = 1*
proof –
have *emeasure (M::'a pmf) M = emeasure (M::'a pmf) (space M)*
by (*intro emeasure-eq-AE (simp-all add: AE-measure-pmf)*)
then show *?thesis*
using *measure-pmf.emeasure-space-1* **by** *simp*
qed

lemma *emeasure-pmf-UNIV [simp]: emeasure (measure-pmf M) UNIV = 1*
using *measure-pmf.emeasure-space-1[of M]* **by** *simp*

lemma *in-null-sets-measure-pmfI:*
A ∩ set-pmf p = {} ⇒ A ∈ null-sets (measure-pmf p)
using *emeasure-eq-0-AE[where ?P=λx. x ∈ A and M=measure-pmf p]*
by(*auto simp add: null-sets-def AE-measure-pmf-iff*)

lemma *measure-subprob: measure-pmf M ∈ space (subprob-algebra (count-space UNIV))*

by (simp add: space-subprob-algebra subprob-space-measure-pmf)

20.2 Monad Interpretation

lemma measurable-measure-pmf[measurable]:

($\lambda x. \text{measure-pmf } (M \ x) \in \text{measurable } (\text{count-space } UNIV) (\text{subprob-algebra } (\text{count-space } UNIV))$)

by (auto simp: space-subprob-algebra intro!: prob-space-imp-subprob-space) unfold-locales

lemma bind-measure-pmf-cong:

assumes $\bigwedge x. A \ x \in \text{space } (\text{subprob-algebra } N) \ \bigwedge x. B \ x \in \text{space } (\text{subprob-algebra } N)$

assumes $\bigwedge i. i \in \text{set-pmf } x \implies A \ i = B \ i$

shows $\text{bind } (\text{measure-pmf } x) \ A = \text{bind } (\text{measure-pmf } x) \ B$

proof (rule measure-eqI)

show $\text{sets } (\text{measure-pmf } x \ggg A) = \text{sets } (\text{measure-pmf } x \ggg B)$

using assms by (subst (1 2) sets-bind) (auto simp: space-subprob-algebra)

next

fix X assume $X \in \text{sets } (\text{measure-pmf } x \ggg A)$

then have $X: X \in \text{sets } N$

using assms by (subst (asm) sets-bind) (auto simp: space-subprob-algebra)

show $\text{emeasure } (\text{measure-pmf } x \ggg A) \ X = \text{emeasure } (\text{measure-pmf } x \ggg B) \ X$

using assms

by (subst (1 2) emeasure-bind[where $N=N$, $OF \ - \ X$])

(auto intro!: nn-integral-cong-AE simp: AE-measure-pmf-iff)

qed

lift-definition bind-pmf :: $'a \text{ pmf} \Rightarrow ('a \Rightarrow 'b \text{ pmf}) \Rightarrow 'b \text{ pmf}$ is bind

proof (clarify, intro conjI)

fix $f :: 'a \text{ measure}$ and $g :: 'a \Rightarrow 'b \text{ measure}$

assume prob-space f

then interpret $f: \text{prob-space } f$.

assume $\text{sets } f = UNIV$ and $ae\text{-}f: AE \ x \text{ in } f. \text{measure } f \ \{x\} \neq 0$

then have $s\text{-}f[simp]: \text{sets } f = \text{sets } (\text{count-space } UNIV)$

by simp

assume $g: \bigwedge x. \text{prob-space } (g \ x) \wedge \text{sets } (g \ x) = UNIV \wedge (AE \ y \text{ in } g \ x. \text{measure } (g \ x) \ \{y\} \neq 0)$

then have $g: \bigwedge x. \text{prob-space } (g \ x)$ and $s\text{-}g[simp]: \bigwedge x. \text{sets } (g \ x) = \text{sets } (\text{count-space } UNIV)$

and $ae\text{-}g: \bigwedge x. AE \ y \text{ in } g \ x. \text{measure } (g \ x) \ \{y\} \neq 0$

by auto

have [measurable]: $g \in \text{measurable } f \ (\text{subprob-algebra } (\text{count-space } UNIV))$

by (auto simp: measurable-def space-subprob-algebra prob-space-imp-subprob-space g)

show $\text{prob-space } (f \ggg g)$

using g by (intro $f.\text{prob-space-bind}$ [where $S=\text{count-space } UNIV$]) auto

```

then interpret fg: prob-space f  $\gg$  g .
show [simp]: sets (f  $\gg$  g) = UNIV
  using sets-eq-imp-space-eq[OF s-f]
  by (subst sets-bind[where N=count-space UNIV]) auto
show AE x in f  $\gg$  g. measure (f  $\gg$  g) {x}  $\neq$  0
  apply (simp add: fg.prob-eq-0 AE-bind[where B=count-space UNIV])
  using ae-f
  apply eventually-elim
  using ae-g
  apply eventually-elim
  apply (auto dest: AE-measure-singleton)
done
qed

```

adhoc-overloading *Monad-Syntax.bind* \equiv *bind-pmf*

```

lemma ennreal-pmf-bind: pmf (bind-pmf N f) i = ( $\int$   $^+$ x. pmf (f x) i  $\partial$ measure-pmf N)
  unfolding pmf.rep-eq bind-pmf.rep-eq
  by (auto simp: measure-pmf.measure-bind[where N=count-space UNIV] measure-subprob measure-nonneg
    intro!: nn-integral-eq-integral[symmetric] measure-pmf.integrable-const-bound[where B=1])

```

```

lemma pmf-bind: pmf (bind-pmf N f) i = ( $\int$  x. pmf (f x) i  $\partial$ measure-pmf N)
  using ennreal-pmf-bind[of N f i]
  by (subst (asm) nn-integral-eq-integral)
    (auto simp: pmf-nonneg pmf-le-1 pmf-nonneg integral-nonneg
      intro!: nn-integral-eq-integral[symmetric] measure-pmf.integrable-const-bound[where B=1])

```

```

lemma bind-pmf-const[simp]: bind-pmf M ( $\lambda$ x. c) = c
  by transfer (simp add: bind-const' prob-space-imp-subprob-space)

```

```

lemma set-bind-pmf[simp]: set-pmf (bind-pmf M N) = ( $\bigcup$   $M \in$  set-pmf M. set-pmf (N M))

```

proof –

```

  have set-pmf (bind-pmf M N) = {x. ennreal (pmf (bind-pmf M N) x)  $\neq$  0}
  by (simp add: set-pmf-eq pmf-nonneg)
  also have ... = ( $\bigcup$   $M \in$  set-pmf M. set-pmf (N M))
  unfolding ennreal-pmf-bind
  by (subst nn-integral-0-iff-AE) (auto simp: AE-measure-pmf-iff pmf-nonneg set-pmf-eq)

```

finally show ?thesis .

qed

```

lemma bind-pmf-cong [fundef-cong]:

```

```

  assumes p = q

```

```

  shows ( $\bigwedge$ x. x  $\in$  set-pmf q  $\implies$  f x = g x)  $\implies$  bind-pmf p f = bind-pmf q g

```

unfolding $\langle p = q \rangle [\text{symmetric}] \text{measure-pmf-inject} [\text{symmetric}] \text{bind-pmf.rep-eq}$
by (*auto simp: AE-measure-pmf-iff Pi-iff space-subprob-algebra subprob-space-measure-pmf*
sets-bind[where N=count-space UNIV] emeasure-bind[where
N=count-space UNIV]
intro!: nn-integral-cong-AE measure-eqI)

lemma *bind-pmf-cong-simp*:
 $p = q \implies (\bigwedge x. x \in \text{set-pmf } q = \text{simp} \implies f x = g x) \implies \text{bind-pmf } p f = \text{bind-pmf}$
 $q g$
by (*simp add: simp-implies-def cong: bind-pmf-cong*)

lemma *measure-pmf-bind*: $\text{measure-pmf } (\text{bind-pmf } M f) = (\text{measure-pmf } M \gg=$
 $(\lambda x. \text{measure-pmf } (f x)))$
by *transfer simp*

lemma *nn-integral-bind-pmf[simp]*: $(\int^+ x. f x \partial \text{bind-pmf } M N) = (\int^+ x. \int^+ y. f y$
 $\partial N x \partial M)$
using *measurable-measure-pmf[of N]*
unfolding *measure-pmf-bind*
apply (*intro nn-integral-bind[where B=count-space UNIV]*)
apply *auto*
done

lemma *emeasure-bind-pmf[simp]*: $\text{emeasure } (\text{bind-pmf } M N) X = (\int^+ x. \text{emeasure}$
 $(N x) X \partial M)$
using *measurable-measure-pmf[of N]*
unfolding *measure-pmf-bind*
by (*subst emeasure-bind[where N=count-space UNIV]*) *auto*

lift-definition *return-pmf* :: $'a \Rightarrow 'a \text{ pmf}$ **is** *return (count-space UNIV)*
by (*auto intro!: prob-space-return simp: AE-return measure-return*)

lemma *bind-return-pmf*: $\text{bind-pmf } (\text{return-pmf } x) f = f x$
by *transfer*
(auto intro!: prob-space-imp-subprob-space bind-return[where N=count-space
UNIV]
simp: space-subprob-algebra)

lemma *set-return-pmf[simp]*: $\text{set-pmf } (\text{return-pmf } x) = \{x\}$
by *transfer (auto simp add: measure-return split: split-indicator)*

lemma *bind-return-pmf'*: $\text{bind-pmf } N \text{return-pmf} = N$
proof (*transfer, clarify*)
fix $N :: 'a \text{ measure}$ **assume** *sets N = UNIV* **then show** $N \gg= \text{return (count-space$
 $\text{UNIV}) = N$
by (*subst return-sets-cong[where N=N]*) (*simp-all add: bind-return'*)
qed

lemma *bind-assoc-pmf*: $\text{bind-pmf } (\text{bind-pmf } A B) C = \text{bind-pmf } A (\lambda x. \text{bind-pmf}$

(*B x*) *C*)
by *transfer*
 (*auto intro!*: *bind-assoc*[**where** *N=count-space UNIV and R=count-space UNIV*]
simp: measurable-def space-subprob-algebra prob-space-imp-subprob-space)

definition *map-pmf* *f M* = *bind-pmf* *M* ($\lambda x. \text{return-pmf } (f\ x)$)

lemma *map-bind-pmf*: *map-pmf* *f* (*bind-pmf* *M g*) = *bind-pmf* *M* ($\lambda x. \text{map-pmf } f\ (g\ x)$)
by (*simp add: map-pmf-def bind-assoc-pmf*)

lemma *bind-map-pmf*: *bind-pmf* (*map-pmf* *f M*) *g* = *bind-pmf* *M* ($\lambda x. g\ (f\ x)$)
by (*simp add: map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *map-pmf-transfer*[*transfer-rule*]:
rel-fun (=) (*rel-fun* *cr-pmf cr-pmf*) ($\lambda f\ M. \text{distr } M\ (\text{count-space } UNIV)\ f$)
map-pmf

proof –
have *rel-fun* (=) (*rel-fun* *pmf-as-measure.cr-pmf pmf-as-measure.cr-pmf*)
 ($\lambda f\ M. M \gg= (\text{return } (\text{count-space } UNIV)\ o\ f))\ \text{map-pmf}$
unfolding *map-pmf-def*[*abs-def*] *comp-def* **by** *transfer-prover*
then show ?thesis
by (*force simp: rel-fun-def cr-pmf-def bind-return-distr*)

qed

lemma *map-pmf-rep-eq*:
measure-pmf (*map-pmf* *f M*) = *distr* (*measure-pmf* *M*) (*count-space UNIV*) *f*
unfolding *map-pmf-def bind-pmf.rep-eq comp-def return-pmf.rep-eq*
using *bind-return-distr*[*of M f count-space UNIV*] **by** (*simp add: comp-def*)

lemma *map-pmf-id*[*simp*]: *map-pmf* *id* = *id*
by (*rule, transfer*) (*auto simp: emeasure-distr measurable-def intro!: measure-eqI*)

lemma *map-pmf-ident*[*simp*]: *map-pmf* ($\lambda x. x$) = ($\lambda x. x$)
using *map-pmf-id* **unfolding** *id-def* .

lemma *map-pmf-compose*: *map-pmf* (*f* \circ *g*) = *map-pmf* *f* \circ *map-pmf* *g*
by (*rule, transfer*) (*simp add: distr-distr*[*symmetric, where N=count-space UNIV*]
measurable-def)

lemma *map-pmf-comp*: *map-pmf* *f* (*map-pmf* *g M*) = *map-pmf* ($\lambda x. f\ (g\ x)$) *M*
using *map-pmf-compose*[*of f g*] **by** (*simp add: comp-def*)

lemma *map-pmf-cong*: $p = q \implies (\bigwedge x. x \in \text{set-pmf } q \implies f\ x = g\ x) \implies \text{map-pmf } f\ p = \text{map-pmf } f\ q$
unfolding *map-pmf-def* **by** (*rule bind-pmf-cong*) *auto*

lemma *pmf-set-map*: *set-pmf* \circ *map-pmf* *f* = (\cdot) *f* \circ *set-pmf*

by (auto simp add: comp-def fun-eq-iff map-pmf-def)

lemma set-map-pmf[simp]: set-pmf (map-pmf f M) = f‘set-pmf M
 using pmf-set-map[of f] by (auto simp: comp-def fun-eq-iff)

lemma emeasure-map-pmf[simp]: emeasure (map-pmf f M) X = emeasure M (f -‘ X)
 unfolding map-pmf-rep-eq by (subst emeasure-distr) auto

lemma measure-map-pmf[simp]: measure (map-pmf f M) X = measure M (f -‘ X)
 using emeasure-map-pmf[of f M X] by (simp add: measure-pmf.emeasure-eq-measure measure-nonneg)

lemma nn-integral-map-pmf[simp]: $(\int^+ x. f x \partial \text{map-pmf } g M) = (\int^+ x. f (g x) \partial M)$
 unfolding map-pmf-rep-eq by (intro nn-integral-distr) auto

lemma ennreal-pmf-map: pmf (map-pmf f p) x = $(\int^+ y. \text{indicator } (f -‘ \{x\}) y \partial \text{measure-pmf } p)$
proof (transfer fixing: f x)
 fix p :: 'b measure
 presume prob-space p
 then interpret prob-space p .
 presume sets p = UNIV
 then show ennreal (measure (distr p (count-space UNIV) f) {x}) = integral^N p (indicator (f -‘ {x}))
 by (simp add: measure-distr measurable-def emeasure-eq-measure)
qed simp-all

lemma pmf-map: pmf (map-pmf f p) x = measure p (f -‘ {x})
proof (transfer fixing: f x)
 fix p :: 'b measure
 presume prob-space p
 then interpret prob-space p .
 presume sets p = UNIV
 then show measure (distr p (count-space UNIV) f) {x} = measure p (f -‘ {x})
 by (simp add: measure-distr measurable-def emeasure-eq-measure)
qed simp-all

lemma nn-integral-pmf: $(\int^+ x. \text{pmf } p x \partial \text{count-space } A) = \text{emeasure } (\text{measure-pmf } p) A$
proof –
 have $(\int^+ x. \text{pmf } p x \partial \text{count-space } A) = (\int^+ x. \text{pmf } p x \partial \text{count-space } (A \cap \text{set-pmf } p))$
 by (auto simp add: nn-integral-count-space-indicator indicator-def set-pmf-iff intro: nn-integral-cong)
 also have $\dots = \text{emeasure } (\text{measure-pmf } p) (\bigcup x \in A \cap \text{set-pmf } p. \{x\})$
 by (subst emeasure-UN-countable) (auto simp add: emeasure-pmf-single disjoint-family-on-def)

also have $\dots = \text{emeasure } (\text{measure-pmf } p) ((\bigcup x \in A \cap \text{set-pmf } p. \{x\}) \cup \{x. x \in A \wedge x \notin \text{set-pmf } p\})$
by (*rule emeasure-Un-null-set[symmetric]*) (*auto intro: in-null-sets-measure-pmfI*)
also have $\dots = \text{emeasure } (\text{measure-pmf } p) A$
by (*auto intro: arg-cong2[where f=emeasure]*)
finally show ?thesis .
qed

lemma *integral-map-pmf[simp]*:
fixes $f :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
shows $\text{integral}^L (\text{map-pmf } g \ p) \ f = \text{integral}^L \ p \ (\lambda x. f \ (g \ x))$
by (*simp add: integral-distr map-pmf-rep-eq*)

lemma *integrable-map-pmf-eq [simp]*:
fixes $g :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
shows $\text{integrable } (\text{map-pmf } f \ p) \ g \longleftrightarrow \text{integrable } (\text{measure-pmf } p) \ (\lambda x. g \ (f \ x))$
by (*subst map-pmf-rep-eq, subst integrable-distr-eq*) *auto*

lemma *integrable-map-pmf [intro]*:
fixes $g :: 'a \Rightarrow 'b :: \{\text{banach, second-countable-topology}\}$
shows $\text{integrable } (\text{measure-pmf } p) \ (\lambda x. g \ (f \ x)) \implies \text{integrable } (\text{map-pmf } f \ p) \ g$
by (*subst integrable-map-pmf-eq*)

lemma *pmf-abs-summable [intro]*: $\text{pmf } p \text{ abs-summable-on } A$
by (*rule abs-summable-on-subset[OF - subset-UNIV]*)
(auto simp: abs-summable-on-def integrable-iff-bounded nn-integral-pmf)

lemma *measure-pmf-conv-infsetsum*: $\text{measure } (\text{measure-pmf } p) \ A = \text{infsetsum } (\text{pmf } p) \ A$
unfolding *infsetsum-def* **by** (*simp add: integral-eq-nn-integral nn-integral-pmf measure-def*)

lemma *infsetsum-pmf-eq-1*:
assumes $\text{set-pmf } p \subseteq A$
shows $\text{infsetsum } (\text{pmf } p) \ A = 1$
proof –
have $\text{infsetsum } (\text{pmf } p) \ A = \text{lebesgue-integral } (\text{count-space } \text{UNIV}) \ (\text{pmf } p)$
using *assms* **unfolding** *infsetsum-altdef set-lebesgue-integral-def*
by (*intro Bochner-Integration.integral-cong*) (*auto simp: indicator-def set-pmf-eq*)
also have $\dots = 1$
by (*subst integral-eq-nn-integral*) (*auto simp: nn-integral-pmf*)
finally show ?thesis .
qed

lemma *map-return-pmf [simp]*: $\text{map-pmf } f \ (\text{return-pmf } x) = \text{return-pmf } (f \ x)$
by *transfer (simp add: distr-return)*

lemma *map-pmf-const[simp]*: $\text{map-pmf } (\lambda \cdot. c) \ M = \text{return-pmf } c$
by *transfer (auto simp: prob-space.distr-const)*

```

lemma pmf-return [simp]: pmf (return-pmf x) y = indicator {y} x
  by transfer (simp add: measure-return)

lemma nn-integral-return-pmf[simp]:  $0 \leq f x \implies (\int^+ x. f x \partial \text{return-pmf } x) = f x$ 
  unfolding return-pmf.rep-eq by (intro nn-integral-return) auto

lemma emeasure-return-pmf[simp]: emeasure (return-pmf x) X = indicator X x
  unfolding return-pmf.rep-eq by (intro emeasure-return) auto

lemma measure-return-pmf [simp]: measure-pmf.prob (return-pmf x) A = indicator A x
proof –
  have ennreal (measure-pmf.prob (return-pmf x) A) =
    emeasure (measure-pmf (return-pmf x)) A
    by (simp add: measure-pmf.emeasure-eq-measure)
  also have ... = ennreal (indicator A x) by (simp add: ennreal-indicator)
  finally show ?thesis by simp
qed

lemma return-pmf-inj[simp]: return-pmf x = return-pmf y  $\longleftrightarrow$  x = y
  by (metis insertI1 set-return-pmf singletonD)

lemma map-pmf-eq-return-pmf-iff:
  map-pmf f p = return-pmf x  $\longleftrightarrow$  ( $\forall y \in \text{set-pmf } p. f y = x$ )
proof
  assume map-pmf f p = return-pmf x
  then have set-pmf (map-pmf f p) = set-pmf (return-pmf x) by simp
  then show  $\forall y \in \text{set-pmf } p. f y = x$  by auto
next
  assume  $\forall y \in \text{set-pmf } p. f y = x$ 
  then show map-pmf f p = return-pmf x
    unfolding map-pmf-const[symmetric, of - p] by (intro map-pmf-cong) auto
qed

definition pair-pmf A B = bind-pmf A ( $\lambda x. \text{bind-pmf } B (\lambda y. \text{return-pmf } (x, y))$ )

lemma pmf-pair: pmf (pair-pmf M N) (a, b) = pmf M a * pmf N b
  unfolding pair-pmf-def pmf-bind pmf-return
  apply (subst integral-measure-pmf-real[where A={b}])
  apply (auto simp: indicator-eq-0-iff)
  apply (subst integral-measure-pmf-real[where A={a}])
  apply (auto simp: indicator-eq-0-iff sum-nonneg-eq-0-iff pmf-nonneg)
  done

lemma set-pair-pmf[simp]: set-pmf (pair-pmf A B) = set-pmf A  $\times$  set-pmf B
  unfolding pair-pmf-def set-bind-pmf set-return-pmf by auto

lemma measure-pmf-in-subprob-space[measurable (raw)]:

```

measure-pmf $M \in \text{space } (\text{subprob-algebra } (\text{count-space } \text{UNIV}))$
by (*simp add: space-subprob-algebra intro-locales*)

lemma *nn-integral-pair-pmf'*: $(\int^+ x. f \ x \ \partial \text{pair-pmf } A \ B) = (\int^+ a. \int^+ b. f \ (a, b) \ \partial B \ \partial A)$

proof –

have $(\int^+ x. f \ x \ \partial \text{pair-pmf } A \ B) = (\int^+ x. f \ x * \text{indicator } (A \times B) \ x \ \partial \text{pair-pmf } A \ B)$

by (*auto simp: AE-measure-pmf-iff intro!: nn-integral-cong-AE*)

also have $\dots = (\int^+ a. \int^+ b. f \ (a, b) * \text{indicator } (A \times B) \ (a, b) \ \partial B \ \partial A)$

by (*simp add: pair-pmf-def*)

also have $\dots = (\int^+ a. \int^+ b. f \ (a, b) \ \partial B \ \partial A)$

by (*auto intro!: nn-integral-cong-AE simp: AE-measure-pmf-iff*)

finally show *?thesis* .

qed

lemma *bind-pair-pmf*:

assumes $M[\text{measurable}]: M \in \text{measurable } (\text{count-space } \text{UNIV} \otimes_M \text{count-space } \text{UNIV}) \ (\text{subprob-algebra } N)$

shows $\text{measure-pmf } (\text{pair-pmf } A \ B) \ggg M = (\text{measure-pmf } A \ggg (\lambda x. \text{measure-pmf } B \ggg (\lambda y. M \ (x, y))))$

(**is** *?L = ?R*)

proof (*rule measure-eqI*)

have $M'[\text{measurable}]: M \in \text{measurable } (\text{pair-pmf } A \ B) \ (\text{subprob-algebra } N)$

using $M[\text{THEN measurable-space}]$ **by** (*simp-all add: space-pair-measure*)

note *measurable-bind[where $N = \text{count-space UNIV}$, measurable]*

note *measure-pmf-in-subprob-space[simp]*

have *sets-eq-N*: $\text{sets } ?L = N$

by (*subst sets-bind[OF sets-kernel[OF M']] auto*)

show $\text{sets } ?L = \text{sets } ?R$

using *measurable-space[OF M]*

by (*simp add: sets-eq-N space-pair-measure space-subprob-algebra*)

fix X **assume** $X \in \text{sets } ?L$

then have $X[\text{measurable}]: X \in \text{sets } N$

unfolding *sets-eq-N* .

then show $\text{emeasure } ?L \ X = \text{emeasure } ?R \ X$

apply (*simp add: emeasure-bind[OF - M' X]*)

apply (*simp add: nn-integral-bind[where $B = \text{count-space UNIV}$] pair-pmf-def measure-pmf-bind[of A]*)

nn-integral-measure-pmf-finite)

apply (*subst emeasure-bind[OF - - X]*)

apply *measurable*

apply (*subst emeasure-bind[OF - - X]*)

apply *measurable*

done

qed

lemma *map-fst-pair-pmf*: *map-pmf fst (pair-pmf A B) = A*
by (*simp add: pair-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf'*)

lemma *map-snd-pair-pmf*: *map-pmf snd (pair-pmf A B) = B*
by (*simp add: pair-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf'*)

lemma *nn-integral-pmf'*:
*inj-on f A \implies ($\int^+ x. \text{pmf } p (f x) \partial \text{count-space } A$) = *emeasure* *p* (*f* ‘ *A*)*
by (*subst nn-integral-bij-count-space[where g=f and B=f‘A]*)
(auto simp: bij-betw-def nn-integral-pmf)

lemma *pmf-le-0-iff*[*simp*]: *pmf M p \leq 0 \longleftrightarrow pmf M p = 0*
using *pmf-nonneg[of M p]* **by** *arith*

lemma *min-pmf-0*[*simp*]: *min (pmf M p) 0 = 0 min 0 (pmf M p) = 0*
using *pmf-nonneg[of M p]* **by** *arith+*

lemma *pmf-eq-0-set-pmf*: *pmf M p = 0 \longleftrightarrow p \notin set-pmf M*
unfolding *set-pmf-iff* **by** *simp*

lemma *pmf-map-inj*: *inj-on f (set-pmf M) \implies x \in set-pmf M \implies pmf (map-pmf f M) (f x) = pmf M x*
by (*auto simp: pmf.rep-eq map-pmf-rep-eq measure-distr AE-measure-pmf-iff inj-onD*
intro!: measure-pmf.finite-measure-eq-AE)

lemma *pair-return-pmf* [*simp*]: *pair-pmf (return-pmf x) (return-pmf y) = return-pmf (x, y)*
by (*auto simp: pair-pmf-def bind-return-pmf*)

lemma *pmf-map-inj'*: *inj f \implies pmf (map-pmf f M) (f x) = pmf M x*
apply(*cases x \in set-pmf M*)
apply(*simp add: pmf-map-inj[OF subset-inj-on]*)
apply(*simp add: pmf-eq-0-set-pmf[symmetric]*)
apply(*auto simp add: pmf-eq-0-set-pmf dest: injD*)
done

lemma *expectation-pair-pmf-fst* [*simp*]:
fixes *f :: 'a \Rightarrow 'b::{banach, second-countable-topology}*
shows *measure-pmf.expectation (pair-pmf p q) ($\lambda x. f (fst x)$) = measure-pmf.expectation p f*
proof –
have *measure-pmf.expectation (pair-pmf p q) ($\lambda x. f (fst x)$) =*
measure-pmf.expectation (map-pmf fst (pair-pmf p q)) f **by** *simp*
also have *map-pmf fst (pair-pmf p q) = p*
by (*simp add: map-fst-pair-pmf*)
finally show *?thesis .*
qed

lemma *expectation-pair-pmf-snd* [simp]:
 fixes $f :: 'a \Rightarrow 'b :: \{\text{banach}, \text{second-countable-topology}\}$
 shows $\text{measure-pmf.expectation} (\text{pair-pmf } p \ q) (\lambda x. f (\text{snd } x)) = \text{measure-pmf.expectation } q \ f$
proof –
 have $\text{measure-pmf.expectation} (\text{pair-pmf } p \ q) (\lambda x. f (\text{snd } x)) =$
 $\text{measure-pmf.expectation} (\text{map-pmf snd} (\text{pair-pmf } p \ q)) f$ **by** *simp*
 also have $\text{map-pmf snd} (\text{pair-pmf } p \ q) = q$
by (*simp add: map-snd-pair-pmf*)
 finally **show** ?thesis .
qed

lemma *pmf-map-outside*: $x \notin f \text{ ` set-pmf } M \implies \text{pmf} (\text{map-pmf } f \ M) \ x = 0$
unfolding *pmf-eq-0-set-pmf* **by** *simp*

lemma *measurable-set-pmf*[*measurable*]: $\text{Measurable.pred} (\text{count-space } \text{UNIV}) (\lambda x. x \in \text{set-pmf } M)$
by *simp*

20.3 PMFs as function

context
 fixes $f :: 'a \Rightarrow \text{real}$
 assumes *nonneg*: $\bigwedge x. 0 \leq f \ x$
 assumes *prob*: $(\int^+ x. f \ x \ \partial \text{count-space } \text{UNIV}) = 1$
begin

lift-definition *embed-pmf* :: $'a \text{ pmf is density } (\text{count-space } \text{UNIV}) (\text{ennreal} \circ f)$
proof (*intro conjI*)

have $*[\text{simp}]: \bigwedge x \ y. \text{ennreal} (f \ y) * \text{indicator } \{x\} \ y = \text{ennreal} (f \ x) * \text{indicator } \{x\} \ y$
by (*simp split: split-indicator*)
 show $\text{AE } x \text{ in density } (\text{count-space } \text{UNIV}) (\text{ennreal} \circ f).$
 $\text{measure} (\text{density } (\text{count-space } \text{UNIV}) (\text{ennreal} \circ f)) \ \{x\} \neq 0$
by (*simp add: AE-density nonneg measure-def emeasure-density max-def*)
 show $\text{prob-space} (\text{density } (\text{count-space } \text{UNIV}) (\text{ennreal} \circ f))$
by *standard* (*simp add: emeasure-density prob*)
qed *simp*

lemma *pmf-embed-pmf*: $\text{pmf embed-pmf } x = f \ x$

proof *transfer*
 have $*[\text{simp}]: \bigwedge x \ y. \text{ennreal} (f \ y) * \text{indicator } \{x\} \ y = \text{ennreal} (f \ x) * \text{indicator } \{x\} \ y$
by (*simp split: split-indicator*)
 fix x **show** $\text{measure} (\text{density } (\text{count-space } \text{UNIV}) (\text{ennreal} \circ f)) \ \{x\} = f \ x$
by *transfer* (*simp add: measure-def emeasure-density nonneg max-def*)
qed

lemma *set-embed-pmf*: $\text{set-pmf embed-pmf} = \{x. f \ x \neq 0\}$

by(*auto simp add: set-pmf-eq pmf-embed-pmf*)

end

lemma *embed-pmf-transfer*:

rel-fun (eq-onp ($\lambda f. (\forall x. 0 \leq f x) \wedge (\int^+ x. \text{ennreal } (f x) \partial \text{count-space UNIV}) = 1$)) pmf-as-measure.cr-pmf ($\lambda f. \text{density } (\text{count-space UNIV}) (\text{ennreal } \circ f)$) embed-pmf

by (*auto simp: rel-fun-def eq-onp-def embed-pmf.transfer*)

lemma *measure-pmf-eq-density*: *measure-pmf p = density (count-space UNIV) (pmf p)*

proof (*transfer, elim conjE*)

fix *M* :: 'a *measure* **assume** [*simp*]: *sets M = UNIV* **and** *ae: AE x in M. measure M {x} ≠ 0*

assume *prob-space M* **then interpret** *prob-space M* .

show *M = density (count-space UNIV) ($\lambda x. \text{ennreal } (\text{measure } M \{x\})$)*

proof (*rule measure-eqI*)

fix *A* :: 'a *set*

have ($\int^+ x. \text{ennreal } (\text{measure } M \{x\}) * \text{indicator } A x \partial \text{count-space UNIV}$) = ($\int^+ x. \text{emeasure } M \{x\} * \text{indicator } (A \cap \{x. \text{measure } M \{x\} \neq 0\}) x \partial \text{count-space UNIV}$)

by (*auto intro!: nn-integral-cong simp: emeasure-eq-measure split: split-indicator*)

also have ... = ($\int^+ x. \text{emeasure } M \{x\} \partial \text{count-space } (A \cap \{x. \text{measure } M \{x\} \neq 0\})$)

by (*subst nn-integral-restrict-space[symmetric] (auto simp: restrict-count-space)*)

also have ... = *emeasure M* ($\bigcup x \in (A \cap \{x. \text{measure } M \{x\} \neq 0\}). \{x\}$)

by (*intro emeasure-UN-countable[symmetric] countable-Int2 countable-support (auto simp: disjoint-family-on-def)*)

also have ... = *emeasure M A*

using *ae* **by** (*intro emeasure-eq-AE*) *auto*

finally show *emeasure M A = emeasure (density (count-space UNIV) ($\lambda x. \text{ennreal } (\text{measure } M \{x\})$)) A*

using *emeasure-space-1* **by** (*simp add: emeasure-density*)

qed *simp*

qed

lemma *td-pmf-embed-pmf*:

type-definition pmf embed-pmf {f::'a \Rightarrow real. ($\forall x. 0 \leq f x$) \wedge ($\int^+ x. \text{ennreal } (f x) \partial \text{count-space UNIV}) = 1$ }

unfolding *type-definition-def*

proof *safe*

fix *p* :: 'a *pmf*

have ($\int^+ x. 1 \partial \text{measure-pmf } p$) = 1

using *measure-pmf.emeasure-space-1[of p]* **by** *simp*

then show *: ($\int^+ x. \text{ennreal } (\text{pmf } p x) \partial \text{count-space UNIV}$) = 1

by (*simp add: measure-pmf-eq-density nn-integral-density pmf-nonneg del: nn-integral-const*)

show *embed-pmf (pmf p) = p*

```

    by (intro measure-pmf-inject[THEN iffD1])
      (simp add: * embed-pmf.rep-eq pmf-nonneg measure-pmf-eq-density[of p]
comp-def)
next
  fix f :: 'a  $\Rightarrow$  real assume  $\forall x. 0 \leq f x (\int^+ x. f x \partial \text{count-space UNIV}) = 1$ 
  then show pmf (embed-pmf f) = f
    by (auto intro!: pmf-embed-pmf)
qed (rule pmf-nonneg)

end

```

```

lemma nn-integral-measure-pmf:  $(\int^+ x. f x \partial \text{measure-pmf } p) = \int^+ x. \text{ennreal}$ 
   $(\text{pmf } p \ x) * f x \partial \text{count-space UNIV}$ 
by (simp add: measure-pmf-eq-density nn-integral-density pmf-nonneg)

```

```

lemma integral-measure-pmf:
  fixes f :: 'a  $\Rightarrow$  'b::{\banach, second-countable-topology}
  assumes A: finite A
  shows  $(\bigwedge a. a \in \text{set-pmf } M \Rightarrow f a \neq 0 \Rightarrow a \in A) \Rightarrow (\text{LINT } x | M. f x) =$ 
   $(\sum_{a \in A. \text{pmf } M \ a *_{\mathbb{R}} f a)$ 
  unfolding measure-pmf-eq-density
  apply (simp add: integral-density)
  apply (subst lebesgue-integral-count-space-finite-support)
  apply (auto intro!: finite-subset[OF - \langle finite A \rangle] sum.mono-neutral-left simp:
pmf-eq-0-set-pmf)
done

```

```

lemma expectation-return-pmf [simp]:
  fixes f :: 'a  $\Rightarrow$  'b::{\banach, second-countable-topology}
  shows measure-pmf.expectation (return-pmf x) f = f x
  by (subst integral-measure-pmf[of {x}]) simp-all

```

```

lemma pmf-expectation-bind:
  fixes p :: 'a pmf and f :: 'a  $\Rightarrow$  'b pmf
  and h :: 'b  $\Rightarrow$  'c::{\banach, second-countable-topology}
  assumes finite A  $\bigwedge x. x \in A \Rightarrow \text{finite } (\text{set-pmf } (f x))$  set-pmf p  $\subseteq A$ 
  shows measure-pmf.expectation (p  $\gg$  f) h =
     $(\sum_{a \in A. \text{pmf } p \ a *_{\mathbb{R}} \text{measure-pmf.expectation } (f a) \ h)$ 

```

```

proof -
  have measure-pmf.expectation (p  $\gg$  f) h =  $(\sum_{a \in (\bigcup x \in A. \text{set-pmf } (f x)). \text{pmf}}$ 
   $(p \gg f) \ a *_{\mathbb{R}} h \ a)$ 
  using assms by (intro integral-measure-pmf) auto
  also have ... =  $(\sum_{x \in (\bigcup x \in A. \text{set-pmf } (f x)). (\sum_{a \in A. (\text{pmf } p \ a * \text{pmf } (f a)$ 
   $x) *_{\mathbb{R}} h \ x))$ 
  proof (intro sum.cong refl, goal-cases)
    case (1 x)
    thus ?case
      by (subst pmf-bind, subst integral-measure-pmf[of A])
        (insert assms, auto simp: scaleR-sum-left)

```

```

qed
also have ... = (∑ j∈A. pmf p j *R (∑ i∈(∪ x∈A. set-pmf (f x)). pmf (f j) i
*_R h i))
  by (subst sum.swap) (simp add: scaleR-sum-right)
also have ... = (∑ j∈A. pmf p j *R measure-pmf.expectation (f j) h)
proof (intro sum.cong refl, goal-cases)
  case (1 x)
  thus ?case
    by (subst integral-measure-pmf[of (∪ x∈A. set-pmf (f x))])
      (insert assms, auto simp: scaleR-sum-left)
qed
finally show ?thesis .
qed

```

```

lemma continuous-on-LINT-pmf: — This is dominated convergence!
fixes f :: 'i ⇒ 'a::topological-space ⇒ 'b::{banach, second-countable-topology}
assumes f: ⋀ i. i ∈ set-pmf M ⇒ continuous-on A (f i)
  and bnd: ⋀ a i. a ∈ A ⇒ i ∈ set-pmf M ⇒ norm (f i a) ≤ B
shows continuous-on A (λ a. LINT i|M. f i a)
proof cases
  assume finite M with f show ?thesis
    using integral-measure-pmf[OF ⟨finite M⟩]
    by (subst integral-measure-pmf[OF ⟨finite M⟩])
      (auto intro!: continuous-on-sum continuous-on-scaleR continuous-on-const)
next
  assume infinite M
  let ?f = λ i x. pmf (map-pmf (to-nat-on M) M) i *R f (from-nat-into M i) x

  show ?thesis
  proof (rule uniform-limit-theorem)
    show ∀F n in sequentially. continuous-on A (λ a. ∑ i<n. ?f i a)
    by (intro always-eventually allI continuous-on-sum continuous-on-scaleR con-
tinuous-on-const f
      from-nat-into set-pmf-not-empty)
    show uniform-limit A (λ n a. ∑ i<n. ?f i a) (λ a. LINT i|M. f i a) sequentially
    proof (subst uniform-limit-cong[where g=λ n a. ∑ i<n. ?f i a])
      fix a assume a ∈ A
      have 1: (LINT i|M. f i a) = (LINT i|map-pmf (to-nat-on M) M. f (from-nat-into
M i) a)
        by (auto intro!: integral-cong-AE AE-pmfI)
      have 2: ... = (LINT i|count-space UNIV. pmf (map-pmf (to-nat-on M) M)
i *R f (from-nat-into M i) a)
        by (simp add: measure-pmf-eq-density integral-density)
      have (λ n. ?f n a) sums (LINT i|M. f i a)
      unfolding 1 2
    proof (intro sums-integral-count-space-nat)
      have A: integrable M (λ i. f i a)
      using ⟨a∈A⟩ by (auto intro!: measure-pmf.integrable-const-bound AE-pmfI
bnd)
    end
  end
end

```

```

      have integrable (map-pmf (to-nat-on M) M) (λi. f (from-nat-into M i) a)
      by (auto simp add: map-pmf-rep-eq integrable-distr-eq intro!: AE-pmfI
integrable-cong-AE-imp[OF A])
      then show integrable (count-space UNIV) (λn. ?f n a)
      by (simp add: measure-pmf-eq-density integrable-density)
    qed
    then show (LINT i|M. f i a) = (∑ n. ?f n a)
    by (simp add: sums-unique)
  next
  show uniform-limit A (λn a. ∑ i<n. ?f i a) (λa. (∑ n. ?f n a)) sequentially
  proof (rule Weierstrass-m-test)
    fix n a assume a∈A
    then show norm (?f n a) ≤ pmf (map-pmf (to-nat-on M) M) n * B
    using bnd by (auto intro!: mult-mono simp: from-nat-into set-pmf-not-empty)
  next
    have integrable (map-pmf (to-nat-on M) M) (λn. B)
    by auto
    then show summable (λn. pmf (map-pmf (to-nat-on (set-pmf M)) M) n *
B)
    by (fastforce simp add: measure-pmf-eq-density integrable-density inte-
grable-count-space-nat-iff summable-mult2)
  qed
  qed simp
  qed simp
qed

```

lemma *continuous-on-LBINT*:

```

  fixes f :: real ⇒ real
  assumes f: ∧b. a ≤ b ⇒ set-integrable lborel {a..b} f
  shows continuous-on UNIV (λb. LBINT x:{a..b}. f x)
proof (subst set-borel-integral-eq-integral)
  { fix b :: real assume a ≤ b
    from f[OF this] have continuous-on {a..b} (λb. integral {a..b} f)
    by (intro indefinite-integral-continuous-1 set-borel-integral-eq-integral) }
  note * = this

  have continuous-on (⋃ b∈{a..}. {a <..b}) (λb. integral {a..b} f)
  proof (intro continuous-on-open-UN)
    show b ∈ {a..} ⇒ continuous-on {a<..b} (λb. integral {a..b} f) for b
    using *[of b] by (rule continuous-on-subset) auto
  qed simp
  also have (⋃ b∈{a..}. {a <..b}) = {a <....}
  by (auto simp: lt-ex gt-ex less-imp-le) (simp add: Bex-def less-imp-le gt-ex cong:
rev-conj-cong)
  finally have continuous-on {a+1 ..} (λb. integral {a..b} f)
  by (rule continuous-on-subset) auto
  moreover have continuous-on {a..a+1} (λb. integral {a..b} f)
  by (rule *) simp
  moreover

```

```

have  $x \leq a \implies \{a..x\} = (\text{if } a = x \text{ then } \{a\} \text{ else } \{\})$  for  $x$ 
  by auto
then have continuous-on  $\{..a\}$   $(\lambda b. \text{integral } \{a..b\} f)$ 
  by (subst continuous-on-cong[OF refl, where  $g = \lambda x. 0$ ]) (auto intro!: continuous-on-const)
ultimately have continuous-on  $(\{..a\} \cup \{a..a+1\} \cup \{a+1 ..\})$   $(\lambda b. \text{integral } \{a..b\} f)$ 
  by (intro continuous-on-closed-Un) auto
also have  $\{..a\} \cup \{a..a+1\} \cup \{a+1 ..\} = \text{UNIV}$ 
  by auto
finally show continuous-on UNIV  $(\lambda b. \text{integral } \{a..b\} f)$ 
  by auto
next
  show set-integrable lborel  $\{a..b\} f$  for  $b$ 
  using  $f$  by (cases  $a \leq b$ ) auto
qed

locale pmf-as-function
begin

setup-lifting td-pmf-embed-pmf

lemma set-pmf-transfer[transfer-rule]:
  assumes bi-total  $A$ 
  shows rel-fun (pcr-pmf  $A$ ) (rel-set  $A$ )  $(\lambda f. \{x. f\ x \neq 0\})$  set-pmf
  using  $\langle \text{bi-total } A \rangle$ 
  by (auto simp: pcr-pmf-def cr-pmf-def rel-fun-def rel-set-def bi-total-def Bex-def set-pmf-iff)
  metis+

end

context
begin

interpretation pmf-as-function .

lemma pmf-eqI:  $(\bigwedge i. \text{pmf } M\ i = \text{pmf } N\ i) \implies M = N$ 
  by transfer auto

lemma pmf-eq-iff:  $M = N \longleftrightarrow (\forall i. \text{pmf } M\ i = \text{pmf } N\ i)$ 
  by (auto intro: pmf-eqI)

lemma pmf-neq-exists-less:
  assumes  $M \neq N$ 
  shows  $\exists x. \text{pmf } M\ x < \text{pmf } N\ x$ 
proof (rule ccontr)
  assume  $\neg(\exists x. \text{pmf } M\ x < \text{pmf } N\ x)$ 
  hence ge:  $\text{pmf } M\ x \geq \text{pmf } N\ x$  for  $x$  by (auto simp: not-less)

```

from *assms* **obtain** x **where** $\text{pmf } M \ x \neq \text{pmf } N \ x$ **by** (*auto simp: pmf-eq-iff*)
with $\text{ge}[of \ x]$ **have** $\text{gt: pmf } M \ x > \text{pmf } N \ x$ **by** *simp*
have $1 = \text{measure} (\text{measure-pmf } M) \ UNIV$ **by** *simp*
also have $\dots = \text{measure} (\text{measure-pmf } N) \ \{x\} + \text{measure} (\text{measure-pmf } N) \ (UNIV - \{x\})$
by (*subst measure-pmf.finite-measure-Union [symmetric]*) *simp-all*
also from *gt* **have** $\text{measure} (\text{measure-pmf } N) \ \{x\} < \text{measure} (\text{measure-pmf } M) \ \{x\}$
by (*simp add: measure-pmf-single*)
also have $\text{measure} (\text{measure-pmf } N) \ (UNIV - \{x\}) \leq \text{measure} (\text{measure-pmf } M) \ (UNIV - \{x\})$
by (*subst (1 2) integral-pmf [symmetric]*)
(intro integral-mono integrable-pmf, simp-all add: ge)
also have $\text{measure} (\text{measure-pmf } M) \ \{x\} + \dots = 1$
by (*subst measure-pmf.finite-measure-Union [symmetric]*) *simp-all*
finally show *False* **by** *simp-all*
qed

lemma *bind-commute-pmf*: $\text{bind-pmf } A \ (\lambda x. \text{bind-pmf } B \ (C \ x)) = \text{bind-pmf } B \ (\lambda y. \text{bind-pmf } A \ (\lambda x. \ C \ x \ y))$

unfolding *pmf-eq-iff pmf-bind*

proof

fix i

interpret B : *prob-space restrict-space B B*

by (*intro prob-space-restrict-space measure-pmf.emmeasure-eq-1-AE*)
(auto simp: AE-measure-pmf-iff)

interpret A : *prob-space restrict-space A A*

by (*intro prob-space-restrict-space measure-pmf.emmeasure-eq-1-AE*)
(auto simp: AE-measure-pmf-iff)

interpret AB : *pair-prob-space restrict-space A A restrict-space B B*

by *unfold-locales*

have $(\int x. \int y. \text{pmf} \ (C \ x \ y) \ i \ \partial B \ \partial A) = (\int x. (\int y. \text{pmf} \ (C \ x \ y) \ i \ \partial \text{restrict-space } B \ B) \ \partial A)$

by (*rule Bochner-Integration.integral-cong*) (*auto intro!: integral-pmf-restrict*)

also have $\dots = (\int x. (\int y. \text{pmf} \ (C \ x \ y) \ i \ \partial \text{restrict-space } B \ B) \ \partial \text{restrict-space } A \ A)$

by (*intro integral-pmf-restrict B.borel-measurable-lebesgue-integral measurable-pair-restrict-pmf2 countable-set-pmf borel-measurable-count-space*)

also have $\dots = (\int y. \int x. \text{pmf} \ (C \ x \ y) \ i \ \partial \text{restrict-space } A \ A \ \partial \text{restrict-space } B \ B)$

by (*rule AB.Fubini-integral[symmetric]*)

(auto intro!: AB.integrable-const-bound[where B=1] measurable-pair-restrict-pmf2 simp: pmf-nonneg pmf-le-1 measurable-restrict-space1)

also have $\dots = (\int y. \int x. \text{pmf} \ (C \ x \ y) \ i \ \partial \text{restrict-space } A \ A \ \partial B)$

by (*intro integral-pmf-restrict[symmetric] A.borel-measurable-lebesgue-integral measurable-pair-restrict-pmf2 countable-set-pmf borel-measurable-count-space*)

also have $\dots = (\int y. \int x. \text{pmf } (C \ x \ y) \ i \ \partial A \ \partial B)$
 by (rule Bochner-Integration.integral-cong) (auto intro!: integral-pmf-restrict[symmetric])
 finally show $(\int x. \int y. \text{pmf } (C \ x \ y) \ i \ \partial B \ \partial A) = (\int y. \int x. \text{pmf } (C \ x \ y) \ i \ \partial A \ \partial B)$.
 qed

lemma pair-map-pmf1: pair-pmf (map-pmf f A) B = map-pmf (apfst f) (pair-pmf A B)

proof (safe intro!: pmf-eqI)

fix a :: 'a and b :: 'b

have [simp]: $\bigwedge c \ d. \text{indicator } (\text{apfst } f - \{(a, b)\}) \ (c, d) = \text{indicator } (f - \{a\}) \ c * (\text{indicator } \{b\} \ d :: \text{ennreal})$

by (auto split: split-indicator)

have ennreal (pmf (pair-pmf (map-pmf f A) B) (a, b)) =
 ennreal (pmf (map-pmf (apfst f) (pair-pmf A B)) (a, b))

unfolding pmf-pair ennreal-pmf-map

by (simp add: nn-integral-pair-pmf' max-def emeasure-pmf-single nn-integral-multc pmf-nonneg

emeasure-map-pmf[symmetric] ennreal-mult del: emeasure-map-pmf)

then show pmf (pair-pmf (map-pmf f A) B) (a, b) = pmf (map-pmf (apfst f) (pair-pmf A B)) (a, b)

by (simp add: pmf-nonneg)

qed

lemma pair-map-pmf2: pair-pmf A (map-pmf f B) = map-pmf (apsnd f) (pair-pmf A B)

proof (safe intro!: pmf-eqI)

fix a :: 'a and b :: 'b

have [simp]: $\bigwedge c \ d. \text{indicator } (\text{apsnd } f - \{(a, b)\}) \ (c, d) = \text{indicator } \{a\} \ c * (\text{indicator } (f - \{b\}) \ d :: \text{ennreal})$

by (auto split: split-indicator)

have ennreal (pmf (pair-pmf A (map-pmf f B)) (a, b)) =
 ennreal (pmf (map-pmf (apsnd f) (pair-pmf A B)) (a, b))

unfolding pmf-pair ennreal-pmf-map

by (simp add: nn-integral-pair-pmf' max-def emeasure-pmf-single nn-integral-cmult nn-integral-multc pmf-nonneg

emeasure-map-pmf[symmetric] ennreal-mult del: emeasure-map-pmf)

then show pmf (pair-pmf A (map-pmf f B)) (a, b) = pmf (map-pmf (apsnd f) (pair-pmf A B)) (a, b)

by (simp add: pmf-nonneg)

qed

lemma map-pair: map-pmf ($\lambda(a, b). (f \ a, g \ b)$) (pair-pmf A B) = pair-pmf (map-pmf f A) (map-pmf g B)

by (simp add: pair-map-pmf2 pair-map-pmf1 map-pmf-comp split-beta')

end

lemma *pair-return-pmf1*: $\text{pair-pmf } (\text{return-pmf } x) \ y = \text{map-pmf } (\text{Pair } x) \ y$
by(*simp add: pair-pmf-def bind-return-pmf map-pmf-def*)

lemma *pair-return-pmf2*: $\text{pair-pmf } x \ (\text{return-pmf } y) = \text{map-pmf } (\lambda x. (x, y)) \ x$
by(*simp add: pair-pmf-def bind-return-pmf map-pmf-def*)

lemma *pair-pair-pmf*: $\text{pair-pmf } (\text{pair-pmf } u \ v) \ w = \text{map-pmf } (\lambda(x, (y, z)). ((x, y), z)) \ (\text{pair-pmf } u \ (\text{pair-pmf } v \ w))$
by(*simp add: pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf*)

lemma *pair-commute-pmf*: $\text{pair-pmf } x \ y = \text{map-pmf } (\lambda(x, y). (y, x)) \ (\text{pair-pmf } y \ x)$
unfolding *pair-pmf-def* **by**(*subst bind-commute-pmf*)(*simp add: map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *set-pmf-subset-singleton*: $\text{set-pmf } p \subseteq \{x\} \longleftrightarrow p = \text{return-pmf } x$
proof(*intro iffI pmf-eqI*)
fix *i*
assume *x*: $\text{set-pmf } p \subseteq \{x\}$
hence *: $\text{set-pmf } p = \{x\}$ **using** *set-pmf-not-empty[of p]* **by** *auto*
have *ennreal* ($\text{pmf } p \ x$) = $\int^+ i. \text{indicator } \{x\} \ i \ \partial p$ **by**(*simp add: emeasure-pmf-single*)
also have $\dots = \int^+ i. 1 \ \partial p$ **by**(*rule nn-integral-cong-AE*)(*simp add: AE-measure-pmf-iff **)
also have $\dots = 1$ **by** *simp*
finally show $\text{pmf } p \ i = \text{pmf } (\text{return-pmf } x) \ i$ **using** *x*
by(*auto split: split-indicator simp add: pmf-eq-0-set-pmf*)
qed *auto*

lemma *bind-eq-return-pmf*:
 $\text{bind-pmf } p \ f = \text{return-pmf } x \longleftrightarrow (\forall y \in \text{set-pmf } p. f \ y = \text{return-pmf } x)$
 (*is ?lhs \longleftrightarrow ?rhs*)
proof(*intro iffI strip*)
fix *y*
assume *y*: $y \in \text{set-pmf } p$
assume *?lhs*
hence $\text{set-pmf } (\text{bind-pmf } p \ f) = \{x\}$ **by** *simp*
hence $(\bigcup y \in \text{set-pmf } p. \text{set-pmf } (f \ y)) = \{x\}$ **by** *simp*
hence $\text{set-pmf } (f \ y) \subseteq \{x\}$ **using** *y* **by** *auto*
thus $f \ y = \text{return-pmf } x$ **by**(*simp add: set-pmf-subset-singleton*)
next
assume *: *?rhs*
show *?lhs*
proof(*rule pmf-eqI*)
fix *i*
have *ennreal* ($\text{pmf } (\text{bind-pmf } p \ f) \ i$) = $\int^+ y. \text{ennreal } (\text{pmf } (f \ y) \ i) \ \partial p$
by (*simp add: ennreal-pmf-bind*)
also have $\dots = \int^+ y. \text{ennreal } (\text{pmf } (\text{return-pmf } x) \ i) \ \partial p$
by(*rule nn-integral-cong-AE*)(*simp add: AE-measure-pmf-iff **)

```

    also have ... = ennreal (pmf (return-pmf x) i)
    by simp
    finally show pmf (bind-pmf p f) i = pmf (return-pmf x) i
    by (simp add: pmf-nonneg)
  qed
qed

```

lemma *pmf-False-conv-True*: $\text{pmf } p \text{ False} = 1 - \text{pmf } p \text{ True}$

proof –

```

  have pmf p False + pmf p True = measure p {False} + measure p {True}
  by (simp add: measure-pmf-single)
  also have ... = measure p ({False} ∪ {True})
  by (subst measure-pmf.finite-measure-Union) simp-all
  also have {False} ∪ {True} = space p by auto
  finally show ?thesis by simp
qed

```

lemma *pmf-True-conv-False*: $\text{pmf } p \text{ True} = 1 - \text{pmf } p \text{ False}$

by (simp add: pmf-False-conv-True)

20.4 Conditional Probabilities

lemma *measure-pmf-zero-iff*: $\text{measure } (\text{measure-pmf } p) s = 0 \longleftrightarrow \text{set-pmf } p \cap s = \{\}$

by (subst measure-pmf.prob-eq-0) (auto simp: AE-measure-pmf-iff)

context

fixes $p :: 'a \text{ pmf}$ **and** $s :: 'a \text{ set}$

assumes *not-empty*: $\text{set-pmf } p \cap s \neq \{\}$

begin

interpretation *pmf-as-measure* .

lemma *emeasure-measure-pmf-not-zero*: $\text{emeasure } (\text{measure-pmf } p) s \neq 0$

proof

assume $\text{emeasure } (\text{measure-pmf } p) s = 0$

then have *AE x in measure-pmf p. x ∉ s*

by (rule AE-I[rotated]) auto

with *not-empty* **show** False

by (auto simp: AE-measure-pmf-iff)

qed

lemma *measure-measure-pmf-not-zero*: $\text{measure } (\text{measure-pmf } p) s \neq 0$

using *emeasure-measure-pmf-not-zero* **by** (simp add: measure-pmf.emeasure-eq-measure measure-nonneg)

lift-definition *cond-pmf* :: $'a \text{ pmf}$ **is**

uniform-measure (measure-pmf p) s

proof (intro conjI)

```

show prob-space (uniform-measure (measure-pmf p) s)
by (intro prob-space-uniform-measure) (auto simp: emeasure-measure-pmf-not-zero)
show  $\text{AE } x \text{ in uniform-measure (measure-pmf p) } s. \text{ measure (uniform-measure (measure-pmf p) } s) \{x\} \neq 0$ 
by (simp add: emeasure-measure-pmf-not-zero measure-measure-pmf-not-zero
  AE-uniform-measure
  AE-measure-pmf-iff set-pmf.rep-eq less-top[symmetric])
qed simp

```

```

lemma pmf-cond:  $\text{pmf cond-pmf } x = (\text{if } x \in s \text{ then pmf } p \ x / \text{measure } p \ s \text{ else } 0)$ 
by transfer (simp add: emeasure-measure-pmf-not-zero pmf.rep-eq)

```

```

lemma set-cond-pmf[simp]:  $\text{set-pmf cond-pmf} = \text{set-pmf } p \cap s$ 
by (auto simp add: set-pmf-iff pmf-cond measure-measure-pmf-not-zero split:
  if-split-asm)

```

end

```

lemma measure-pmf-posI:  $x \in \text{set-pmf } p \implies x \in A \implies \text{measure-pmf.prob } p \ A > 0$ 
using measure-measure-pmf-not-zero[of p A] by (subst zero-less-measure-iff) blast

```

```

lemma cond-map-pmf:
  assumes  $\text{set-pmf } p \cap f - ' s \neq \{\}$ 
  shows  $\text{cond-pmf (map-pmf f } p) \ s = \text{map-pmf f (cond-pmf } p \ (f - ' s))$ 
proof -
  have *:  $\text{set-pmf (map-pmf f } p) \cap s \neq \{\}$ 
  using assms by auto
  { fix } x
  have  $\text{ennreal (pmf (map-pmf f (cond-pmf } p \ (f - ' s))) } x) =$ 
     $\text{emeasure } p \ (f - ' s \cap f - ' \{x\}) / \text{emeasure } p \ (f - ' s)$ 
  unfolding ennreal-pmf-map cond-pmf.rep-eq[OF assms] by (simp add: nn-integral-uniform-measure)
  also have  $f - ' s \cap f - ' \{x\} = (\text{if } x \in s \text{ then } f - ' \{x\} \text{ else } \{\})$ 
  by auto
  also have  $\text{emeasure } p \ (\text{if } x \in s \text{ then } f - ' \{x\} \text{ else } \{\}) / \text{emeasure } p \ (f - ' s) =$ 
     $\text{ennreal (pmf (cond-pmf (map-pmf f } p) \ s) } x)$ 
  using measure-measure-pmf-not-zero[OF *]
  by (simp add: pmf-cond[OF *] ennreal-pmf-map measure-pmf.emeasure-eq-measure
    divide-ennreal pmf-nonneg measure-nonneg zero-less-measure-iff
    pmf-map)
  finally have  $\text{ennreal (pmf (cond-pmf (map-pmf f } p) \ s) } x) = \text{ennreal (pmf (map-pmf f (cond-pmf } p \ (f - ' s))) } x)$ 
  by simp }
  then show ?thesis
  by (intro pmf-eqI) (simp add: pmf-nonneg)
qed

```

```

lemma bind-cond-pmf-cancel:
  assumes [simp]:  $\bigwedge x. x \in \text{set-pmf } p \implies \text{set-pmf } q \cap \{y. R \ x \ y\} \neq \{\}$ 

```

```

assumes [simp]:  $\bigwedge y. y \in \text{set-pmf } q \implies \text{set-pmf } p \cap \{x. R \ x \ y\} \neq \{\}$ 
assumes [simp]:  $\bigwedge x \ y. x \in \text{set-pmf } p \implies y \in \text{set-pmf } q \implies R \ x \ y \implies \text{measure}$ 
 $q \ \{y. R \ x \ y\} = \text{measure } p \ \{x. R \ x \ y\}$ 
shows  $\text{bind-pmf } p \ (\lambda x. \text{cond-pmf } q \ \{y. R \ x \ y\}) = q$ 
proof (rule pmf-eqI)
  fix  $i$ 
  have  $\text{ennreal } (\text{pmf } (\text{bind-pmf } p \ (\lambda x. \text{cond-pmf } q \ \{y. R \ x \ y\})) \ i) =$ 
 $(\int^+ x. \text{ennreal } (\text{pmf } q \ i / \text{measure } p \ \{x. R \ x \ i\}) * \text{ennreal } (\text{indicator } \{x. R \ x \ i\}$ 
 $x) \ \partial p)$ 
  by (auto simp add:  $\text{ennreal-pmf-bind}$   $\text{AE-measure-pmf-iff}$   $\text{pmf-cond}$   $\text{pmf-eq-0-set-pmf}$ 
 $\text{pmf-nonneg}$   $\text{measure-nonneg}$ 
 $\text{intro!}$ :  $\text{nn-integral-cong-AE}$ )
  also have  $\dots = (\text{pmf } q \ i * \text{measure } p \ \{x. R \ x \ i\}) / \text{measure } p \ \{x. R \ x \ i\}$ 
  by (simp add:  $\text{pmf-nonneg}$   $\text{measure-nonneg}$   $\text{zero-ennreal-def[symmetric]}$   $\text{ennreal-indicator}$ 
 $\text{nn-integral-cmult}$   $\text{measure-pmf.emeasure-eq-measure}$   $\text{ennreal-mult[symmetric]}$ )
  also have  $\dots = \text{pmf } q \ i$ 
  by (cases  $\text{pmf } q \ i = 0$ )
  (simp-all add:  $\text{pmf-eq-0-set-pmf}$   $\text{measure-measure-pmf-not-zero}$   $\text{pmf-nonneg}$ )
  finally show  $\text{pmf } (\text{bind-pmf } p \ (\lambda x. \text{cond-pmf } q \ \{y. R \ x \ y\})) \ i = \text{pmf } q \ i$ 
  by (simp add:  $\text{pmf-nonneg}$ )
qed

```

20.5 Relator

```

inductive  $\text{rel-pmf} :: ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \text{ pmf} \Rightarrow 'b \text{ pmf} \Rightarrow \text{bool}$ 
for  $R \ p \ q$ 
where
   $\llbracket \bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies R \ x \ y;$ 
 $\text{map-pmf fst } pq = p; \text{map-pmf snd } pq = q \rrbracket$ 
 $\implies \text{rel-pmf } R \ p \ q$ 

```

lemma rel-pmfI :

```

assumes  $R: \text{rel-set } R \ (\text{set-pmf } p) \ (\text{set-pmf } q)$ 
assumes  $\text{eq}: \bigwedge x \ y. x \in \text{set-pmf } p \implies y \in \text{set-pmf } q \implies R \ x \ y \implies$ 
 $\text{measure } p \ \{x. R \ x \ y\} = \text{measure } q \ \{y. R \ x \ y\}$ 
shows  $\text{rel-pmf } R \ p \ q$ 
proof
  let  $?pq = \text{bind-pmf } p \ (\lambda x. \text{bind-pmf } (\text{cond-pmf } q \ \{y. R \ x \ y\}) \ (\lambda y. \text{return-pmf } (x,$ 
 $y)))$ 
  have  $\bigwedge x. x \in \text{set-pmf } p \implies \text{set-pmf } q \cap \{y. R \ x \ y\} \neq \{\}$ 
  using  $R$  by (auto simp:  $\text{rel-set-def}$ )
  then show  $\bigwedge x \ y. (x, y) \in \text{set-pmf } ?pq \implies R \ x \ y$ 
  by auto
  show  $\text{map-pmf fst } ?pq = p$ 
  by (simp add:  $\text{map-bind-pmf bind-return-pmf}'$ )

  show  $\text{map-pmf snd } ?pq = q$ 
  using  $R \ \text{eq}$ 

```

```

  apply (simp add: bind-cond-pmf-cancel map-bind-pmf bind-return-pmf')
  apply (rule bind-cond-pmf-cancel)
  apply (auto simp: rel-set-def)
  done
qed

```

lemma *rel-pmf-imp-rel-set*: $\text{rel-pmf } R \ p \ q \implies \text{rel-set } R \ (\text{set-pmf } p) \ (\text{set-pmf } q)$
 by (force simp add: rel-pmf.simps rel-set-def)

lemma *rel-pmfD-measure*:

```

  assumes rel-R: rel-pmf R p q and R:  $\bigwedge a \ b. R \ a \ b \implies R \ a \ y \longleftrightarrow R \ x \ b$ 
  assumes  $x \in \text{set-pmf } p \ y \in \text{set-pmf } q$ 
  shows  $\text{measure } p \ \{x. R \ x \ y\} = \text{measure } q \ \{y. R \ x \ y\}$ 
proof -
  from rel-R obtain pq where  $pq: \bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies R \ x \ y$ 
  and eq:  $p = \text{map-pmf fst } pq \ q = \text{map-pmf snd } pq$ 
  by (auto elim: rel-pmf.cases)
  have  $\text{measure } p \ \{x. R \ x \ y\} = \text{measure } pq \ \{x. R \ (\text{fst } x) \ y\}$ 
  by (simp add: eq map-pmf-rep-eq measure-distr)
  also have  $\dots = \text{measure } pq \ \{y. R \ x \ (\text{snd } y)\}$ 
  by (intro measure-pmf.finite-measure-eq-AE)
  (auto simp: AE-measure-pmf-iff R dest!: pq)
  also have  $\dots = \text{measure } q \ \{y. R \ x \ y\}$ 
  by (simp add: eq map-pmf-rep-eq measure-distr)
  finally show  $\text{measure } p \ \{x. R \ x \ y\} = \text{measure } q \ \{y. R \ x \ y\}$  .
qed

```

lemma *rel-pmf-measureD*:

```

  assumes rel-pmf R p q
  shows  $\text{measure } (\text{measure-pmf } p) \ A \leq \text{measure } (\text{measure-pmf } q) \ \{y. \exists x \in A. R \ x \ y\}$  (is ?lhs ≤ ?rhs)
using assms
proof cases
  fix pq
  assume R:  $\bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies R \ x \ y$ 
  and p[symmetric]:  $\text{map-pmf fst } pq = p$ 
  and q[symmetric]:  $\text{map-pmf snd } pq = q$ 
  have ?lhs =  $\text{measure } (\text{measure-pmf } pq) \ (\text{fst} - 'A)$  by (simp add: p)
  also have  $\dots \leq \text{measure } (\text{measure-pmf } pq) \ \{y. \exists x \in A. R \ x \ (\text{snd } y)\}$ 
  by (rule measure-pmf.finite-measure-mono-AE) (auto  $\lambda$  3 simp add: AE-measure-pmf-iff
  dest: R)
  also have  $\dots = ?rhs$  by (simp add: q)
  finally show ?thesis .
qed

```

lemma *rel-pmf-iff-measure*:

```

  assumes symp R transp R
  shows  $\text{rel-pmf } R \ p \ q \longleftrightarrow$ 
     $\text{rel-set } R \ (\text{set-pmf } p) \ (\text{set-pmf } q) \ \wedge$ 

```

```

  (∀ x ∈ set-pmf p. ∀ y ∈ set-pmf q. R x y ⟶ measure p {x. R x y} = measure q
  {y. R x y})
  by (safe intro!: rel-pmf-imp-rel-set rel-pmfI)
  (auto intro!: rel-pmfD-measure dest: sympD[OF ‹symp R›] transpD[OF ‹transp
  R›])

```

lemma *quotient-rel-set-disjoint*:

```

  equivp R ⟹ C ∈ UNIV // {(x, y). R x y} ⟹ rel-set R A B ⟹ A ∩ C = {}
  ⟷ B ∩ C = {}
  using in-quotient-imp-closed[of UNIV {(x, y). R x y} C]
  by (auto 0 0 simp: equivp-equiv rel-set-def set-eq-iff elim: equivpE)
  (blast dest: equivp-symp)+

```

lemma *quotientD*: $\text{equiv } X \ R \implies A \in X \ // \ R \implies x \in A \implies A = R \text{ `` } \{x\}$

by (*metis Image-singleton-iff equiv-class-eq-iff quotientE*)

lemma *rel-pmf-iff-equivp*:

```

  assumes equivp R
  shows rel-pmf R p q ⟷ (∀ C ∈ UNIV // {(x, y). R x y}. measure p C = measure
  q C)
  (is - ⟷ (∀ C ∈ - // ?R. -))

```

proof (*subst rel-pmf-iff-measure, safe*)

show *symp R transp R*

using *assms* **by** (*auto simp: equivp-reflp-symp-transp*)

next

fix *C* **assume** *C*: $C \in UNIV \ // \ ?R$ **and** *R*: *rel-set R (set-pmf p) (set-pmf q)*

assume *eq*: $\forall x \in \text{set-pmf } p. \forall y \in \text{set-pmf } q. R \ x \ y \longrightarrow \text{measure } p \ \{x. R \ x \ y\} =$
measure q {y. R x y}

show *measure p C = measure q C*

proof (*cases p ∩ C = {}*)

case *True*

then have $q \cap C = \{\}$

using *quotient-rel-set-disjoint*[*OF assms C R*] **by** *simp*

with *True* **show** *?thesis*

unfolding *measure-pmf-zero-iff*[*symmetric*] **by** *simp*

next

case *False*

then have $q \cap C \neq \{\}$

using *quotient-rel-set-disjoint*[*OF assms C R*] **by** *simp*

with *False* **obtain** *x y* **where** *in-set*: $x \in \text{set-pmf } p \ y \in \text{set-pmf } q$ **and** *in-C*:
 $x \in C \ y \in C$

by *auto*

then have *R x y*

using *in-quotient-imp-in-rel*[*of UNIV ?R C x y*] *C assms*

by (*simp add: equivp-equiv*)

with *in-set eq* **have** $\text{measure } p \ \{x. R \ x \ y\} = \text{measure } q \ \{y. R \ x \ y\}$

by *auto*

moreover have $\{y. R \ x \ y\} = C$

```

    using assms ⟨x ∈ C⟩ C quotientD[of UNIV ?R C x] by (simp add: equivp-equiv)
  moreover have {x. R x y} = C
    using assms ⟨y ∈ C⟩ C quotientD[of UNIV ?R C y] sympD[of R]
    by (auto simp add: equivp-equiv elim: equivpE)
  ultimately show ?thesis
    by auto
qed
next
assume eq: ∀ C ∈ UNIV // ?R. measure p C = measure q C
show rel-set R (set-pmf p) (set-pmf q)
  unfolding rel-set-def
proof safe
  fix x assume x: x ∈ set-pmf p
  have {y. R x y} ∈ UNIV // ?R
    by (auto simp: quotient-def)
  with eq have *: measure q {y. R x y} = measure p {y. R x y}
    by auto
  have measure q {y. R x y} ≠ 0
    using x assms unfolding * by (auto simp: measure-pmf-zero-iff set-eq-iff
dest: equivp-reflp)
  then show ∃ y ∈ set-pmf q. R x y
    unfolding measure-pmf-zero-iff by auto
next
fix y assume y: y ∈ set-pmf q
have {x. R x y} ∈ UNIV // ?R
  using assms by (auto simp: quotient-def dest: equivp-symp)
with eq have *: measure p {x. R x y} = measure q {x. R x y}
  by auto
have measure p {x. R x y} ≠ 0
  using y assms unfolding * by (auto simp: measure-pmf-zero-iff set-eq-iff
dest: equivp-reflp)
then show ∃ x ∈ set-pmf p. R x y
  unfolding measure-pmf-zero-iff by auto
qed

fix x y assume x ∈ set-pmf p y ∈ set-pmf q R x y
have {y. R x y} ∈ UNIV // ?R {x. R x y} = {y. R x y}
  using assms ⟨R x y⟩ by (auto simp: quotient-def dest: equivp-symp equivp-transp)
with eq show measure p {x. R x y} = measure q {y. R x y}
  by auto
qed

bnf pmf: 'a pmf map: map-pmf sets: set-pmf bd : card-suc natLeq rel: rel-pmf
proof -
  show map-pmf id = id by (rule map-pmf-id)
  show ∧ f g. map-pmf (f ∘ g) = map-pmf f ∘ map-pmf g by (rule map-pmf-compose)
  show ∧ f g::'a ⇒ 'b. ∧ p. (∧ x. x ∈ set-pmf p ⇒ f x = g x) ⇒ map-pmf f p =
map-pmf g p
    by (intro map-pmf-cong refl)

```



```

show  $\bigwedge f :: 'a \Rightarrow 'b. \text{set-pmf} \circ \text{map-pmf } f = (\cdot) f \circ \text{set-pmf}$ 
by (rule pmf-set-map)

show card-order (card-suc natLeq) using natLeq-card-order by (rule card-order-card-suc)
show BNF-Cardinal-Arithmetic.cinfinite (card-suc natLeq)
using natLeq-Cinfinite natLeq-card-order Cinfinite-card-suc by blast
show regularCard (card-suc natLeq) using natLeq-card-order natLeq-Cinfinite
by (rule regularCard-card-suc)

show (card-of (set-pmf p), card-suc natLeq)  $\in$  ordLess for  $p :: 's \text{ pmf}$ 
proof –
  have (card-of (set-pmf p), card-of (UNIV :: nat set))  $\in$  ordLeq
    by (rule card-of-ordLeqI[where  $f = \text{to-nat-on (set-pmf p)}$ ])
    (auto intro: countable-set-pmf)
  also have (card-of (UNIV :: nat set), natLeq)  $\in$  ordLeq
    by (metis Field-natLeq card-of-least natLeq-Well-order)
  finally show ?thesis using card-suc-greater natLeq-card-order ordLeq-ordLess-trans
by blast
qed

show  $\bigwedge R. \text{rel-pmf } R = (\lambda x y. \exists z. \text{set-pmf } z \subseteq \{(x, y). R \ x \ y\} \wedge$ 
   $\text{map-pmf fst } z = x \wedge \text{map-pmf snd } z = y)$ 
by (auto simp add: fun-eq-iff rel-pmf.simps)

show  $\text{rel-pmf } R \text{ OO } \text{rel-pmf } S \leq \text{rel-pmf } (R \text{ OO } S)$ 
for  $R :: 'a \Rightarrow 'b \Rightarrow \text{bool}$  and  $S :: 'b \Rightarrow 'c \Rightarrow \text{bool}$ 
proof –
  { fix  $p \ q \ r$ 
    assume  $pq: \text{rel-pmf } R \ p \ q$ 
    and  $qr: \text{rel-pmf } S \ q \ r$ 
    from  $pq$  obtain  $pq$  where  $pq: \bigwedge x y. (x, y) \in \text{set-pmf } pq \implies R \ x \ y$ 
    and  $p: p = \text{map-pmf fst } pq$  and  $q: q = \text{map-pmf snd } pq$  by cases auto
    from  $qr$  obtain  $qr$  where  $qr: \bigwedge y z. (y, z) \in \text{set-pmf } qr \implies S \ y \ z$ 
    and  $q': q = \text{map-pmf fst } qr$  and  $r: r = \text{map-pmf snd } qr$  by cases auto

    define  $pr$  where  $pr =$ 
       $\text{bind-pmf } pq \ (\lambda xy. \text{bind-pmf } (\text{cond-pmf } qr \ \{yz. \text{fst } yz = \text{snd } xy\})$ 
       $(\lambda yz. \text{return-pmf } (\text{fst } xy, \text{snd } yz)))$ 
    have  $pr\text{-welldefined}: \bigwedge y. y \in q \implies qr \cap \{yz. \text{fst } yz = y\} \neq \{\}$ 
    by (force simp:  $q'$ )

    have  $\text{rel-pmf } (R \text{ OO } S) \ p \ r$ 
    proof (rule rel-pmf.intros)
      fix  $x \ z$  assume  $(x, z) \in pr$ 
      then have  $\exists y. (x, y) \in pq \wedge (y, z) \in qr$ 
      by (auto simp:  $q \text{ pr-welldefined } pr\text{-def split-beta}$ )
      with  $pq \ qr$  show  $(R \text{ OO } S) \ x \ z$ 
      by blast
  }

```

```

next
  have map-pmf snd pr = map-pmf snd (bind-pmf q (λy. cond-pmf qr {yz. fst
yz = y}))
    by (simp add: pr-def q split-beta bind-map-pmf map-pmf-def[symmetric]
map-bind-pmf map-pmf-comp)
    then show map-pmf snd pr = r
      unfolding r q' bind-map-pmf by (subst (asm) bind-cond-pmf-cancel) (auto
simp: eq-commute)
      qed (simp add: pr-def map-bind-pmf split-beta map-pmf-def[symmetric] p
map-pmf-comp)
    }
  then show ?thesis
    by(auto simp add: le-fun-def)
qed
qed

```

lemma *map-pmf-idI*: $(\bigwedge x. x \in \text{set-pmf } p \implies f\ x = x) \implies \text{map-pmf } f\ p = p$
by(*simp cong: pmf.map-cong*)

lemma *rel-pmf-conj[simp]*:
 $\text{rel-pmf } (\lambda x\ y. P \wedge Q\ x\ y)\ x\ y \longleftrightarrow P \wedge \text{rel-pmf } Q\ x\ y$
 $\text{rel-pmf } (\lambda x\ y. Q\ x\ y \wedge P)\ x\ y \longleftrightarrow P \wedge \text{rel-pmf } Q\ x\ y$
using *set-pmf-not-empty* **by** (*fastforce simp: pmf.in-rel subset-eq*) +

lemma *rel-pmf-top[simp]*: $\text{rel-pmf } \text{top} = \text{top}$
by (*auto simp: pmf.in-rel[abs-def] fun-eq-iff map-fst-pair-pmf map-snd-pair-pmf
intro: exI[of - pair-pmf x y for x y]*)

lemma *rel-pmf-return-pmf1*: $\text{rel-pmf } R\ (\text{return-pmf } x)\ M \longleftrightarrow (\forall a \in M. R\ x\ a)$
proof *safe*
fix *a* **assume** $a \in M$ $\text{rel-pmf } R\ (\text{return-pmf } x)\ M$
then obtain *pq* **where** $*$: $\bigwedge a\ b. (a, b) \in \text{set-pmf } pq \implies R\ a\ b$
and *eq*: $\text{return-pmf } x = \text{map-pmf } \text{fst } pq\ M = \text{map-pmf } \text{snd } pq$
by (*force elim: rel-pmf.cases*)
moreover have $\text{set-pmf } (\text{return-pmf } x) = \{x\}$
by *simp*
with $\langle a \in M \rangle$ **have** $(x, a) \in pq$
by (*force simp: eq*)
with $*$ **show** $R\ x\ a$
by *auto*
qed (*auto intro!: rel-pmf.intros[where pq=pair-pmf (return-pmf x) M]
simp: map-fst-pair-pmf map-snd-pair-pmf*)

lemma *rel-pmf-return-pmf2*: $\text{rel-pmf } R\ M\ (\text{return-pmf } x) \longleftrightarrow (\forall a \in M. R\ a\ x)$
by (*subst pmf.rel-flip[symmetric]*) (*simp add: rel-pmf-return-pmf1*)

lemma *rel-return-pmf[simp]*: $\text{rel-pmf } R\ (\text{return-pmf } x1)\ (\text{return-pmf } x2) = R\ x1\ x2$
unfolding *rel-pmf-return-pmf2 set-return-pmf* **by** *simp*

lemma *rel-pmf-False*[simp]: *rel-pmf* ($\lambda x y. \text{False}$) $x y = \text{False}$
unfolding *pmf.in-rel fun-eq-iff* **using** *set-pmf-not-empty* **by** *fastforce*

lemma *rel-pmf-rel-prod*:

rel-pmf (*rel-prod* $R S$) (*pair-pmf* $A A'$) (*pair-pmf* $B B'$) \longleftrightarrow *rel-pmf* $R A B \wedge$
rel-pmf $S A' B'$

proof *safe*

assume *rel-pmf* (*rel-prod* $R S$) (*pair-pmf* $A A'$) (*pair-pmf* $B B'$)

then obtain pq **where** $pq: \bigwedge a b c d. ((a, c), (b, d)) \in \text{set-pmf } pq \implies R a b \wedge$
 $S c d$

and $eq: \text{map-pmf fst } pq = \text{pair-pmf } A A' \text{ map-pmf snd } pq = \text{pair-pmf } B B'$

by (*force elim: rel-pmf.cases*)

show *rel-pmf* $R A B$

proof (*rule rel-pmf.intros*)

let $?f = \lambda(a, b). (\text{fst } a, \text{fst } b)$

have [simp]: $(\lambda x. \text{fst } (?f x)) = \text{fst } o \text{fst } (\lambda x. \text{snd } (?f x)) = \text{fst } o \text{snd}$

by *auto*

show *map-pmf fst* (*map-pmf* $?f pq$) = A

by (*simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-fst-pair-pmf*)

show *map-pmf snd* (*map-pmf* $?f pq$) = B

by (*simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-fst-pair-pmf*)

fix $a b$ **assume** $(a, b) \in \text{set-pmf } (\text{map-pmf } ?f pq)$

then obtain $c d$ **where** $((a, c), (b, d)) \in \text{set-pmf } pq$

by *auto*

from $pq[OF \text{ this}]$ **show** $R a b \dots$

qed

show *rel-pmf* $S A' B'$

proof (*rule rel-pmf.intros*)

let $?f = \lambda(a, b). (\text{snd } a, \text{snd } b)$

have [simp]: $(\lambda x. \text{fst } (?f x)) = \text{snd } o \text{fst } (\lambda x. \text{snd } (?f x)) = \text{snd } o \text{snd}$

by *auto*

show *map-pmf fst* (*map-pmf* $?f pq$) = A'

by (*simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-snd-pair-pmf*)

show *map-pmf snd* (*map-pmf* $?f pq$) = B'

by (*simp add: map-pmf-comp pmf.map-comp[symmetric] eq map-snd-pair-pmf*)

fix $c d$ **assume** $(c, d) \in \text{set-pmf } (\text{map-pmf } ?f pq)$

then obtain $a b$ **where** $((a, c), (b, d)) \in \text{set-pmf } pq$

by *auto*

from $pq[OF \text{ this}]$ **show** $S c d \dots$

qed

next

assume *rel-pmf* $R A B$ *rel-pmf* $S A' B'$

then obtain $Rpq Spq$

where $Rpq: \bigwedge a b. (a, b) \in \text{set-pmf } Rpq \implies R a b$

```

    map-pmf fst Rpq = A map-pmf snd Rpq = B
  and Spq:  $\bigwedge a\ b. (a, b) \in \text{set-pmf } Spq \implies S\ a\ b$ 
    map-pmf fst Spq = A' map-pmf snd Spq = B'
  by (force elim: rel-pmf.cases)

  let ?f = ( $\lambda((a, c), (b, d)). ((a, b), (c, d))$ )
  let ?pq = map-pmf ?f (pair-pmf Rpq Spq)
  have [simp]: ( $\lambda x. \text{fst } (?f\ x) = (\lambda(a, b). (\text{fst } a, \text{fst } b))$ ) ( $\lambda x. \text{snd } (?f\ x) = (\lambda(a, b). (\text{snd } a, \text{snd } b))$ )
    by auto

  show rel-pmf (rel-prod R S) (pair-pmf A A') (pair-pmf B B')
    by (rule rel-pmf.intros[where pq=?pq])
      (auto simp: map-snd-pair-pmf map-fst-pair-pmf map-pmf-comp Rpq Spq
        map-pair)

qed

lemma rel-pmf-refl:
  assumes  $\bigwedge x. x \in \text{set-pmf } p \implies P\ x\ x$ 
  shows rel-pmf P p p
  by (rule rel-pmf.intros[where pq=map-pmf ( $\lambda x. (x, x)$ ) p])
    (auto simp add: pmf.map-comp o-def assms)

lemma rel-pmf-bij-betw:
  assumes f: bij-betw f (set-pmf p) (set-pmf q)
  and eq:  $\bigwedge x. x \in \text{set-pmf } p \implies \text{pmf } p\ x = \text{pmf } q\ (f\ x)$ 
  shows rel-pmf ( $\lambda x\ y. f\ x = y$ ) p q
proof(rule rel-pmf.intros)
  let ?pq = map-pmf ( $\lambda x. (x, f\ x)$ ) p
  show map-pmf fst ?pq = p by(simp add: pmf.map-comp o-def)

  have map-pmf f p = q
  proof(rule pmf-eqI)
    fix i
    show pmf (map-pmf f p) i = pmf q i
    proof(cases i  $\in$  set-pmf q)
    case True
    with f obtain j where i = f j j  $\in$  set-pmf p
    by(auto simp add: bij-betw-def image-iff)
    thus ?thesis using f by(simp add: bij-betw-def pmf-map-inj eq)
    next
    case False thus ?thesis
    by(subst pmf-map-outside)(auto simp add: set-pmf-iff eq[symmetric])
  qed
qed
qed
then show map-pmf snd ?pq = q by(simp add: pmf.map-comp o-def)
qed auto

context

```

begin

interpretation *pmf-as-measure* .

definition *join-pmf* $M = \text{bind-pmf } M \ (\lambda x. x)$

lemma *bind-eq-join-pmf*: $\text{bind-pmf } M f = \text{join-pmf } (\text{map-pmf } f M)$
unfolding *join-pmf-def bind-map-pmf* ..

lemma *join-eq-bind-pmf*: $\text{join-pmf } M = \text{bind-pmf } M \text{ id}$
by (*simp add: join-pmf-def id-def*)

lemma *pmf-join*: $\text{pmf } (\text{join-pmf } N) \ i = (\int M. \text{pmf } M \ i \ \partial \text{measure-pmf } N)$
unfolding *join-pmf-def pmf-bind* ..

lemma *ennreal-pmf-join*: $\text{ennreal } (\text{pmf } (\text{join-pmf } N) \ i) = (\int^+ M. \text{pmf } M \ i \ \partial \text{measure-pmf } N)$
unfolding *join-pmf-def ennreal-pmf-bind* ..

lemma *set-pmf-join-pmf[simp]*: $\text{set-pmf } (\text{join-pmf } f) = (\bigcup p \in \text{set-pmf } f. \text{set-pmf } p)$
by (*simp add: join-pmf-def*)

lemma *join-return-pmf*: $\text{join-pmf } (\text{return-pmf } M) = M$
by (*simp add: integral-return pmf-eq-iff pmf-join return-pmf.rep-eq*)

lemma *map-join-pmf*: $\text{map-pmf } f (\text{join-pmf } AA) = \text{join-pmf } (\text{map-pmf } (\text{map-pmf } f) AA)$
by (*simp add: join-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *join-map-return-pmf*: $\text{join-pmf } (\text{map-pmf } \text{return-pmf } A) = A$
by (*simp add: join-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf'*)

end

lemma *rel-pmf-joinI*:

assumes *rel-pmf* (*rel-pmf* P) $p \ q$

shows *rel-pmf* P (*join-pmf* p) (*join-pmf* q)

proof –

from *assms* **obtain** pq **where** $p: p = \text{map-pmf } \text{fst } pq$

and $q: q = \text{map-pmf } \text{snd } pq$

and $P: \bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies \text{rel-pmf } P \ x \ y$

by *cases auto*

from P **obtain** PQ

where $PQ: \bigwedge x \ y \ a \ b. \llbracket (x, y) \in \text{set-pmf } pq; (a, b) \in \text{set-pmf } (PQ \ x \ y) \rrbracket \implies$

$P \ a \ b$

and $x: \bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies \text{map-pmf } \text{fst } (PQ \ x \ y) = x$

and $y: \bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies \text{map-pmf } \text{snd } (PQ \ x \ y) = y$

by(*metis rel-pmf.simps*)

```

let ?r = bind-pmf pq ( $\lambda(x, y). PQ\ x\ y$ )
have  $\bigwedge a\ b. (a, b) \in \text{set-pmf } ?r \implies P\ a\ b$  by (auto intro: PQ)
moreover have map-pmf fst ?r = join-pmf p map-pmf snd ?r = join-pmf q
  by (simp-all add: p q x y join-pmf-def map-bind-pmf bind-map-pmf split-def
cong: bind-pmf-cong)
ultimately show ?thesis ..
qed

```

```

lemma rel-pmf-bindI:
  assumes pq: rel-pmf R p q
  and fg:  $\bigwedge x\ y. R\ x\ y \implies \text{rel-pmf } P\ (f\ x)\ (g\ y)$ 
  shows rel-pmf P (bind-pmf p f) (bind-pmf q g)
  unfolding bind-eq-join-pmf
  by (rule rel-pmf-joinI)
  (auto simp add: pmf.rel-map intro: pmf.rel-mono[THEN le-funD, THEN
le-funD, THEN le-boolD, THEN mp, OF - pq] fg)

```

Proof that *rel-pmf* preserves orders. Antisymmetry proof follows Thm. 1 in N. Saheb-Djahromi, Cpo’s of measures for nondeterminism, Theoretical Computer Science 12(1):19–37, 1980, [https://doi.org/10.1016/0304-3975\(80\)90003-1](https://doi.org/10.1016/0304-3975(80)90003-1)

```

lemma
  assumes *: rel-pmf R p q
  and refl: reflp R and trans: transp R
  shows measure-Ici: measure p {y. R x y}  $\leq$  measure q {y. R x y} (is ?thesis1)
  and measure-Ioi: measure p {y. R x y  $\wedge$   $\neg$  R y x}  $\leq$  measure q {y. R x y  $\wedge$   $\neg$ 
R y x} (is ?thesis2)
proof –
  from * obtain pq
    where pq:  $\bigwedge x\ y. (x, y) \in \text{set-pmf } pq \implies R\ x\ y$ 
    and p: p = map-pmf fst pq
    and q: q = map-pmf snd pq
    by cases auto
  show ?thesis1 ?thesis2 unfolding p q map-pmf-rep-eq using refl trans
    by(auto 4 3 simp add: measure-distr reflpD AE-measure-pmf-iff intro!: mea-
sure-pmf.finite-measure-mono-AE dest!: pq elim: transpE)
qed

```

```

lemma rel-pmf-inf:
  fixes p q :: 'a pmf
  assumes 1: rel-pmf R p q
  assumes 2: rel-pmf R q p
  and refl: reflp R and trans: transp R
  shows rel-pmf (inf R  $R^{-1-1}$ ) p q
proof (subst rel-pmf-iff-equivp, safe)
  show equivp (inf R  $R^{-1-1}$ )
    using trans refl by (auto simp: equivp-reflp-symp-transp intro: sympI transpI
reflpI dest: transpD reflpD)

```

```

fix C assume C ∈ UNIV // {(x, y). inf R R-1-1 x y}
then obtain x where C: C = {y. R x y ∧ R y x}
  by (auto elim: quotientE)

let ?R = λx y. R x y ∧ R y x
let ?μR = λy. measure q {x. ?R x y}
have measure p {y. ?R x y} = measure p ({y. R x y} - {y. R x y ∧ ¬ R y x})
  by (auto intro!: arg-cong[where f=measure p])
also have ... = measure p {y. R x y} - measure p {y. R x y ∧ ¬ R y x}
  by (rule measure-pmf.finite-measure-Diff) auto
also have measure p {y. R x y ∧ ¬ R y x} = measure q {y. R x y ∧ ¬ R y x}
  using 1 2 refl trans by (auto intro!: Orderings.antisym measure-Ioi)
also have measure p {y. R x y} = measure q {y. R x y}
  using 1 2 refl trans by (auto intro!: Orderings.antisym measure-Ici)
also have measure q {y. R x y} - measure q {y. R x y ∧ ¬ R y x} =
  measure q ({y. R x y} - {y. R x y ∧ ¬ R y x})
  by (rule measure-pmf.finite-measure-Diff[symmetric]) auto
also have ... = ?μR x
  by (auto intro!: arg-cong[where f=measure q])
finally show measure p C = measure q C
  by (simp add: C conj-commute)
qed

```

```

lemma rel-pmf-antisym:
  fixes p q :: 'a pmf
  assumes 1: rel-pmf R p q
  assumes 2: rel-pmf R q p
  and refl: reflp R and trans: transp R and antisym: antisymp R
  shows p = q
proof -
  from 1 2 refl trans have rel-pmf (inf R R-1-1) p q by (rule rel-pmf-inf)
  also have inf R R-1-1 = (=)
    using refl antisym by (auto intro!: ext simp add: reflpD dest: antisympD)
  finally show ?thesis unfolding pmf.rel-eq .
qed

```

```

lemma reflp-rel-pmf: reflp R ⇒ reflp (rel-pmf R)
  by (fact pmf.rel-reflp)

```

```

lemma antisymp-rel-pmf:
  [ reflp R; transp R; antisymp R ]
  ⇒ antisymp (rel-pmf R)
by (rule antisympI)(blast intro: rel-pmf-antisym)

```

```

lemma transp-rel-pmf:
  assumes transp R
  shows transp (rel-pmf R)
  using assms by (fact pmf.rel-transp)

```

20.6 Distributions

context

begin

interpretation *pmf-as-function* .

20.6.1 Bernoulli Distribution

lift-definition *bernoulli-pmf* :: *real* \Rightarrow *bool pmf* is

$\lambda p \ b. ((\lambda p. \text{if } b \text{ then } p \text{ else } 1 - p) \circ \text{min } 1 \circ \text{max } 0) \ p$

by (*auto simp: nn-integral-count-space-finite* [where $A = \{\text{False}, \text{True}\}$] *UNIV-bool split: split-max split-min*)

lemma *pmf-bernoulli-True*[*simp*]: $0 \leq p \Rightarrow p \leq 1 \Rightarrow \text{pmf } (\text{bernoulli-pmf } p) \text{ True} = p$

by *transfer simp*

lemma *pmf-bernoulli-False*[*simp*]: $0 \leq p \Rightarrow p \leq 1 \Rightarrow \text{pmf } (\text{bernoulli-pmf } p) \text{ False} = 1 - p$

by *transfer simp*

lemma *set-pmf-bernoulli*[*simp*]: $0 < p \Rightarrow p < 1 \Rightarrow \text{set-pmf } (\text{bernoulli-pmf } p) = \text{UNIV}$

by (*auto simp add: set-pmf-iff UNIV-bool*)

lemma *nn-integral-bernoulli-pmf*[*simp*]:

assumes [*simp*]: $0 \leq p \ p \leq 1 \ \bigwedge x. 0 \leq f \ x$

shows $(\int^+ x. f \ x \ \partial \text{bernoulli-pmf } p) = f \ \text{True} * p + f \ \text{False} * (1 - p)$

by (*subst nn-integral-measure-pmf-support* [of *UNIV*])
(*auto simp: UNIV-bool field-simps*)

lemma *integral-bernoulli-pmf*[*simp*]:

assumes [*simp*]: $0 \leq p \ p \leq 1$

shows $(\int x. f \ x \ \partial \text{bernoulli-pmf } p) = f \ \text{True} * p + f \ \text{False} * (1 - p)$

by (*subst integral-measure-pmf* [of *UNIV*]) (*auto simp: UNIV-bool*)

lemma *pmf-bernoulli-half* [*simp*]: $\text{pmf } (\text{bernoulli-pmf } (1 / 2)) \ x = 1 / 2$

by (*cases x*) *simp-all*

lemma *measure-pmf-bernoulli-half*: $\text{measure-pmf } (\text{bernoulli-pmf } (1 / 2)) = \text{uniform-count-measure } \text{UNIV}$

by (*rule measure-eqI*)

(*simp-all add: nn-integral-pmf* [*symmetric*] *emeasure-uniform-count-measure ennreal-divide-numeral* [*symmetric*])

nn-integral-count-space-finite sets-uniform-count-measure divide-ennreal-def mult-ac ennreal-of-nat-eq-real-of-nat)

20.6.2 Geometric Distribution

context

fixes $p :: \text{real}$ assumes $p[\text{arith}]: 0 < p \leq 1$
begin

lift-definition *geometric-pmf* :: $\text{nat} \rightarrow \text{pmf}$ is $\lambda n. (1 - p)^\wedge n * p$

proof

have $(\sum i. \text{ennreal } (p * (1 - p)^\wedge i)) = \text{ennreal } (p * (1 / (1 - (1 - p))))$
 by (intro *suminf-ennreal-eq sums-mult geometric-sums*) auto
 then show $(\int^+ x. \text{ennreal } ((1 - p)^\wedge x * p) \partial \text{count-space UNIV}) = 1$
 by (simp add: *nn-integral-count-space-nat field-simps*)
qed *simp*

lemma *pmf-geometric[simp]*: $\text{pmf } \text{geometric-pmf } n = (1 - p)^\wedge n * p$
 by *transfer rule*

end

lemma *geometric-pmf-1 [simp]*: $\text{geometric-pmf } 1 = \text{return-pmf } 0$
 by (intro *pmf-eqI*) (auto simp: *indicator-def*)

lemma *set-pmf-geometric*: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{geometric-pmf } p) = \text{UNIV}$
 by (auto simp: *set-pmf-iff*)

lemma *geometric-sums-times-n*:

fixes $c :: 'a :: \{\text{banach}, \text{real-normed-field}\}$
 assumes $\text{norm } c < 1$
 shows $(\lambda n. c^\wedge n * \text{of-nat } n) \text{ sums } (c / (1 - c)^2)$
proof –
 have $(\lambda n. c * z^\wedge n) \text{ sums } (c / (1 - z))$ if $\text{norm } z < 1$ for z
 using *geometric-sums sums-mult* that by *fastforce*
 moreover have $((\lambda z. c / (1 - z)) \text{ has-field-derivative } (c / (1 - c)^2)) \text{ (at } c)$
 using *assms* by (auto intro!: *derivative-eq-intros simp add: semiring-normalization-rules*)
 ultimately have $(\lambda n. \text{diffs } (\lambda n. c) n * c^\wedge n) \text{ sums } (c / (1 - c)^2)$
 using *assms* by (intro *termdiffs-sums-strong*)
 then have $(\lambda n. \text{of-nat } (\text{Suc } n) * c^\wedge (\text{Suc } n)) \text{ sums } (c / (1 - c)^2)$
 unfolding *diffs-def* by (simp add: *power-eq-if mult.assoc*)
 then show *?thesis*
 by (subst (*asm*) *sums-Suc-iff*) (auto simp add: *mult.commute*)
qed

lemma *geometric-sums-times-norm*:

fixes $c :: 'a :: \{\text{banach}, \text{real-normed-field}\}$
 assumes $\text{norm } c < 1$
 shows $(\lambda n. \text{norm } (c^\wedge n * \text{of-nat } n)) \text{ sums } (\text{norm } c / (1 - \text{norm } c)^2)$
proof –
 have $\text{norm } (c^\wedge n * \text{of-nat } n) = (\text{norm } c)^\wedge n * \text{of-nat } n$ for $n :: \text{nat}$
 by (simp add: *norm-power norm-mult*)

then show *?thesis*
using *geometric-sums-times-n*[*of norm c*] *assms*
by force
qed

lemma *integrable-real-geometric-pmf*:
assumes $p \in \{0 < .. 1\}$
shows *integrable* (*geometric-pmf* p) *real*
proof –
have *summable* ($\lambda x. p * ((1 - p) ^ x * \text{real } x)$)
using *geometric-sums-times-norm*[*of 1 - p*] *assms*
by (*intro summable-mult*) (*auto simp: sums-iff*)
hence *summable* ($\lambda x. (1 - p) ^ x * \text{real } x$)
by (*rule summable-mult-D*) (*use assms in auto*)
thus *?thesis*
unfolding *measure-pmf-eq-density* **using** *assms*
by (*subst integrable-density*)
(auto simp: integrable-count-space-nat-iff mult-ac)
qed

lemma *expectation-geometric-pmf*:
assumes $p \in \{0 < .. 1\}$
shows *measure-pmf.expectation* (*geometric-pmf* p) *real* = $(1 - p) / p$
proof –
have ($\lambda n. p * ((1 - p) ^ n * n)$) *sums* ($p * ((1 - p) / p ^ 2)$)
using *assms geometric-sums-times-n*[*of 1 - p*] **by** (*intro sums-mult*) *auto*
moreover have ($\lambda n. p * ((1 - p) ^ n * n)$) = ($\lambda n. (1 - p) ^ n * p * \text{real } n$)
by auto
ultimately have *: ($\lambda n. (1 - p) ^ n * p * \text{real } n$) *sums* $((1 - p) / p)$
using *assms sums-subst* **by** (*auto simp add: power2-eq-square*)
have *measure-pmf.expectation* (*geometric-pmf* p) *real* =
 $(\int n. \text{pmf } (\text{geometric-pmf } p) \ n * \text{real } n \ \partial \text{count-space UNIV})$
unfolding *measure-pmf-eq-density* **by** (*subst integral-density*) *auto*
also have *integrable* (*count-space UNIV*) ($\lambda n. \text{pmf } (\text{geometric-pmf } p) \ n * \text{real } n$)
using * *assms* **unfolding** *integrable-count-space-nat-iff* **by** (*simp add: sums-iff*)
hence $(\int n. \text{pmf } (\text{geometric-pmf } p) \ n * \text{real } n \ \partial \text{count-space UNIV}) = (1 - p) / p$
using * *assms* **by** (*subst integral-count-space-nat*) (*simp-all add: sums-iff*)
finally show *?thesis* **by auto**
qed

lemma *geometric-bind-pmf-unfold*:
assumes $p \in \{0 < .. 1\}$
shows *geometric-pmf* p =
 $\text{do } \{b \leftarrow \text{bernoulli-pmf } p;$
 $\text{if } b \text{ then return-pmf } 0 \text{ else map-pmf Suc } (\text{geometric-pmf } p)\}$
proof –
have *: ($\text{Suc } - ' \{i\}$) = (*if* $i = 0$ *then* $\{ \}$ *else* $\{i - 1\}$) **for** i

```

    by force
  have pmf (geometric-pmf p) i =
    pmf (bernoulli-pmf p >>=
      (λb. if b then return-pmf 0 else map-pmf Suc (geometric-pmf p)))
    i for i
  proof -
    have pmf (geometric-pmf p) i =
      (if i = 0 then p else (1 - p) * pmf (geometric-pmf p) (i - 1))
    using assms by (simp add: power-eq-if)
    also have ... = (if i = 0 then p else (1 - p) * pmf (map-pmf Suc (geometric-pmf
  p)) i)
    by (simp add: pmf-map indicator-def measure-pmf-single *)
    also have ... = measure-pmf.expectation (bernoulli-pmf p)
      (λx. pmf (if x then return-pmf 0 else map-pmf Suc (geometric-pmf p)) i)
    using assms by (auto simp add: pmf-map *)
    also have ... = pmf (bernoulli-pmf p >>=
      (λb. if b then return-pmf 0 else map-pmf Suc (geometric-pmf p)))
      i
    by (auto simp add: pmf-bind)
    finally show ?thesis .
  qed
  then show ?thesis
    using pmf-eqI by blast
  qed

```

20.6.3 Uniform Multiset Distribution

context

fixes $M :: 'a \text{ multiset}$ **assumes** $M\text{-not-empty}$: $M \neq \{\#\}$
begin

lift-definition $\text{pmf-of-multiset} :: 'a \text{ pmf}$ **is** $\lambda x. \text{count } M \ x \ / \ \text{size } M$

proof

show $(\int^+ x. \text{ennreal } (\text{real } (\text{count } M \ x) / \text{real } (\text{size } M))) \ \partial \text{count-space } UNIV = 1$

using $M\text{-not-empty}$

by (simp add: zero-less-divide-iff nn-integral-count-space nonempty-has-size
 sum-divide-distrib[symmetric])

(auto simp: size-multiset-overloaded-eq intro!: sum.cong)

qed simp

lemma $\text{pmf-of-multiset}[simp]$: $\text{pmf } \text{pmf-of-multiset } x = \text{count } M \ x \ / \ \text{size } M$
by transfer rule

lemma $\text{set-pmf-of-multiset}[simp]$: $\text{set-pmf } \text{pmf-of-multiset} = \text{set-mset } M$
by (auto simp: set-pmf-iff)

end

20.6.4 Uniform Distribution

context

fixes $S :: 'a \text{ set}$ **assumes** $S\text{-not-empty}: S \neq \{\}$ **and** $S\text{-finite}: \text{finite } S$
begin

lift-definition $\text{pmf-of-set} :: 'a \text{ pmf}$ **is** $\lambda x. \text{indicator } S \ x \ / \ \text{card } S$

proof

show $(\int^+ x. \text{ennreal } (\text{indicator } S \ x \ / \ \text{real } (\text{card } S)) \ \partial \text{count-space UNIV}) = 1$

using $S\text{-not-empty } S\text{-finite}$

by $(\text{subst nn-integral-count-space'[of } S])$

$(\text{auto simp: ennreal-of-nat-eq-real-of-nat ennreal-mult[symmetric]})$

qed simp

lemma $\text{pmf-of-set[simp]}: \text{pmf pmf-of-set } x = \text{indicator } S \ x \ / \ \text{card } S$

by transfer rule

lemma $\text{set-pmf-of-set[simp]}: \text{set-pmf pmf-of-set} = S$

using $S\text{-finite } S\text{-not-empty}$ **by** $(\text{auto simp: set-pmf-iff})$

lemma $\text{emeasure-pmf-of-set-space[simp]}: \text{emeasure pmf-of-set } S = 1$

by $(\text{rule measure-pmf.emeasure-eq-1-AE}) \ (\text{auto simp: AE-measure-pmf-iff})$

lemma $\text{nn-integral-pmf-of-set}: \text{nn-integral } (\text{measure-pmf pmf-of-set}) \ f = \text{sum } f \ S \ / \ \text{card } S$

by $(\text{subst nn-integral-measure-pmf-finite})$

$(\text{simp-all add: sum-distrib-right[symmetric] card-gt-0-iff } S\text{-not-empty } S\text{-finite}$
 $\text{divide-ennreal-def}$

$\text{divide-ennreal[symmetric] ennreal-of-nat-eq-real-of-nat[symmetric]}$
 $\text{ennreal-times-divide})$

lemma $\text{integral-pmf-of-set}: \text{integral}^L (\text{measure-pmf pmf-of-set}) \ f = \text{sum } f \ S \ / \ \text{card } S$

by $(\text{subst integral-measure-pmf[of } S]) \ (\text{auto simp: } S\text{-finite sum-divide-distrib})$

lemma $\text{emeasure-pmf-of-set}: \text{emeasure } (\text{measure-pmf pmf-of-set}) \ A = \text{card } (S \cap A) \ / \ \text{card } S$

by $(\text{subst nn-integral-indicator[symmetric], simp})$

$(\text{simp add: } S\text{-finite } S\text{-not-empty card-gt-0-iff indicator-def sum.If-cases divide-ennreal}$

$\text{ennreal-of-nat-eq-real-of-nat nn-integral-pmf-of-set})$

lemma $\text{measure-pmf-of-set}: \text{measure } (\text{measure-pmf pmf-of-set}) \ A = \text{card } (S \cap A) \ / \ \text{card } S$

using $\text{emeasure-pmf-of-set[of } A]$

by $(\text{simp add: measure-nonneg measure-pmf.emeasure-eq-measure})$

end

lemma $\text{pmf-expectation-bind-pmf-of-set}:$

```

fixes  $A :: 'a \text{ set}$  and  $f :: 'a \Rightarrow 'b \text{ pmf}$ 
and  $h :: 'b \Rightarrow 'c :: \{\text{banach, second-countable-topology}\}$ 
assumes  $A \neq \{\}$   $\text{finite } A \wedge x. x \in A \implies \text{finite } (\text{set-pmf } (f \ x))$ 
shows  $\text{measure-pmf.expectation } (\text{pmf-of-set } A \gg f) \ h =$ 
 $(\sum a \in A. \text{measure-pmf.expectation } (f \ a) \ h \ /_R \ \text{real } (\text{card } A))$ 
using assms by (subst pmf-expectation-bind[of A]) (auto simp: field-split-simps)

```

```

lemma map-pmf-of-set:
assumes  $\text{finite } A \ A \neq \{\}$ 
shows  $\text{map-pmf } f \ (\text{pmf-of-set } A) = \text{pmf-of-multiset } (\text{image-mset } f \ (\text{mset-set } A))$ 
 $(\text{is } ?lhs = ?rhs)$ 
proof (intro pmf-eqI)
fix  $x$ 
from assms have  $\text{ennreal } (\text{pmf } ?lhs \ x) = \text{ennreal } (\text{pmf } ?rhs \ x)$ 
by (subst ennreal-pmf-map)
 $(\text{simp-all add: emeasure-pmf-of-set mset-set-empty-iff count-image-mset Int-commute})$ 
thus  $\text{pmf } ?lhs \ x = \text{pmf } ?rhs \ x$  by simp
qed

```

```

lemma pmf-bind-pmf-of-set:
assumes  $A \neq \{\}$   $\text{finite } A$ 
shows  $\text{pmf } (\text{bind-pmf } (\text{pmf-of-set } A) \ f) \ x =$ 
 $(\sum xa \in A. \text{pmf } (f \ xa) \ x) \ / \ \text{real-of-nat } (\text{card } A) \ (\text{is } ?lhs = ?rhs)$ 
proof –
from assms have  $\text{card } A > 0$  by auto
with assms have  $\text{ennreal } ?lhs = \text{ennreal } ?rhs$ 
by (subst ennreal-pmf-bind)
 $(\text{simp-all add: nn-integral-pmf-of-set max-def pmf-nonneg divide-ennreal [symmetric] sum-nonneg ennreal-of-nat-eq-real-of-nat})$ 
thus ?thesis by (subst (asm) ennreal-inj) (auto intro!: sum-nonneg divide-nonneg-nonneg)
qed

```

```

lemma pmf-of-set-singleton:  $\text{pmf-of-set } \{x\} = \text{return-pmf } x$ 
by(rule pmf-eqI)(simp add: indicator-def)

```

```

lemma map-pmf-of-set-inj:
assumes  $f: \text{inj-on } f \ A$ 
and [simp]:  $A \neq \{\}$   $\text{finite } A$ 
shows  $\text{map-pmf } f \ (\text{pmf-of-set } A) = \text{pmf-of-set } (f \ ' A) \ (\text{is } ?lhs = ?rhs)$ 
proof(rule pmf-eqI)
fix  $i$ 
show  $\text{pmf } ?lhs \ i = \text{pmf } ?rhs \ i$ 
proof(cases i ∈ f ' A)
case True
then obtain  $i'$  where  $i = f \ i' \ i' \in A$  by auto
thus ?thesis using  $f$  by(simp add: card-image pmf-map-inj)

```

```

next
  case False
  hence pmf ?lhs i = 0 by (simp add: pmf-eq-0-set-pmf set-map-pmf)
  moreover have pmf ?rhs i = 0 using False by simp
  ultimately show ?thesis by simp
qed
qed

```

```

lemma map-pmf-of-set-bij-betw:
  assumes bij-betw f A B A ≠ {} finite A
  shows map-pmf f (pmf-of-set A) = pmf-of-set B
proof -
  have map-pmf f (pmf-of-set A) = pmf-of-set (f ' A)
  by (intro map-pmf-of-set-inj assms bij-betw-imp-inj-on[OF assms(1)])
  also from assms have f ' A = B by (simp add: bij-betw-def)
  finally show ?thesis .
qed

```

Choosing an element uniformly at random from the union of a disjoint family of finite non-empty sets with the same size is the same as first choosing a set from the family uniformly at random and then choosing an element from the chosen set uniformly at random.

```

lemma pmf-of-set-UN:
  assumes finite (⋃ (f ' A)) A ≠ {} ∧ x. x ∈ A ⇒ f x ≠ {}
  ∧ x. x ∈ A ⇒ card (f x) = n disjoint-family-on f A
  shows pmf-of-set (⋃ (f ' A)) = do {x ← pmf-of-set A; pmf-of-set (f x)}
  (is ?lhs = ?rhs)
proof (intro pmf-eqI)
  fix x
  from assms have [simp]: finite A
  using infinite-disjoint-family-imp-infinite-UNION[of A f] by blast
  from assms have ereal (pmf (pmf-of-set (⋃ (f ' A))) x) =
    ereal (indicator (⋃ x∈A. f x) x / real (card (⋃ x∈A. f x)))
  by (subst pmf-of-set) auto
  also from assms have card (⋃ x∈A. f x) = card A * n
  by (subst card-UN-disjoint) (auto simp: disjoint-family-on-def)
  also from assms
  have indicator (⋃ x∈A. f x) x / real ... =
    indicator (⋃ x∈A. f x) x / (n * real (card A))
  by (simp add: sum-divide-distrib [symmetric] mult-ac)
  also from assms have indicator (⋃ x∈A. f x) x = (∑ y∈A. indicator (f y) x)
  by (intro indicator-UN-disjoint) simp-all
  also from assms have ereal ((∑ y∈A. indicator (f y) x) / (real n * real (card
A))) =
    ereal (pmf ?rhs x)
  by (subst pmf-bind-pmf-of-set) (simp-all add: sum-divide-distrib)
  finally show pmf ?lhs x = pmf ?rhs x by simp
qed

```

lemma *bernoulli-pmf-half-conv-pmf-of-set*: $\text{bernoulli-pmf } (1 / 2) = \text{pmf-of-set UNIV}$
by (*rule pmf-eqI*) *simp-all*

20.6.5 Poisson Distribution

context

fixes $\text{rate} :: \text{real}$ **assumes** rate-pos : $0 < \text{rate}$

begin

lift-definition *poisson-pmf* :: nat pmf **is** $\lambda k. \text{rate}^k / \text{fact } k * \exp(-\text{rate})$

proof

have *summable*: *summable* $(\lambda x :: \text{nat}. \text{rate}^x / \text{fact } x)$ **using** *summable-exp*

by (*simp add: field-simps divide-inverse [symmetric]*)

have $(\int^+(x :: \text{nat}). \text{rate}^x / \text{fact } x * \exp(-\text{rate}) \partial \text{count-space UNIV}) =$
 $\exp(-\text{rate}) * (\int^+(x :: \text{nat}). \text{rate}^x / \text{fact } x \partial \text{count-space UNIV})$

by (*simp add: field-simps nn-integral-cmult[symmetric] ennreal-mult'[symmetric]*)

also from *rate-pos* **have** $(\int^+(x :: \text{nat}). \text{rate}^x / \text{fact } x \partial \text{count-space UNIV}) =$
 $(\sum x. \text{rate}^x / \text{fact } x)$

by (*simp-all add: nn-integral-count-space-nat suminf-ennreal summable ennreal-suminf-neq-top*)

also have $\dots = \exp \text{rate}$ **unfolding** *exp-def*

by (*simp add: field-simps divide-inverse [symmetric]*)

also have $\text{ennreal } (\exp(-\text{rate})) * \text{ennreal } (\exp \text{rate}) = 1$

by (*simp add: mult-exp-exp ennreal-mult[symmetric]*)

finally show $(\int^+ x. \text{ennreal } (\text{rate}^x / (\text{fact } x) * \exp(-\text{rate})) \partial \text{count-space UNIV}) = 1$.

qed (*simp add: rate-pos[THEN less-imp-le]*)

lemma *pmf-poisson[simp]*: $\text{pmf poisson-pmf } k = \text{rate}^k / \text{fact } k * \exp(-\text{rate})$

by *transfer rule*

lemma *set-pmf-poisson[simp]*: $\text{set-pmf poisson-pmf} = \text{UNIV}$

using *rate-pos* **by** (*auto simp: set-pmf-iff*)

end

20.6.6 Binomial Distribution

context

fixes $n :: \text{nat}$ **and** $p :: \text{real}$ **assumes** *p-nonneg*: $0 \leq p$ **and** *p-le-1*: $p \leq 1$

begin

lift-definition *binomial-pmf* :: nat pmf **is** $\lambda k. (n \text{ choose } k) * p^k * (1 - p)^{(n - k)}$

proof

have $(\int^+ k. \text{ennreal } (\text{real } (n \text{ choose } k) * p^k * (1 - p)^{(n - k})) \partial \text{count-space UNIV}) =$

$\text{ennreal } (\sum k \leq n. \text{real } (n \text{ choose } k) * p^k * (1 - p)^{(n - k}))$

using *p-le-1 p-nonneg* **by** (*subst nn-integral-count-space'*) *auto*

also have $(\sum k \leq n. \text{real } (n \text{ choose } k) * p^k * (1 - p)^{n-k}) = (p + (1 - p))^n$
by $(\text{subst binomial-ring}) (\text{simp add: atLeast0AtMost})$
finally show $(\int^+ x. \text{ennreal } (\text{real } (n \text{ choose } x) * p^x * (1 - p)^{n-x}))$
 $\partial \text{count-space UNIV} = 1$
by simp
qed $(\text{insert } p\text{-nonneg } p\text{-le-1, simp})$

lemma $\text{pmf-binomial}[\text{simp}]$: $\text{pmf binomial-pmf } k = (n \text{ choose } k) * p^k * (1 - p)^{n-k}$
by transfer rule

lemma $\text{set-pmf-binomial-eq}$: $\text{set-pmf binomial-pmf} = (\text{if } p = 0 \text{ then } \{0\} \text{ else if } p = 1 \text{ then } \{n\} \text{ else } \{..n\})$
using $p\text{-nonneg } p\text{-le-1}$ **unfolding** $\text{set-eq-iff set-pmf-iff pmf-binomial}$ **by** $(\text{auto simp: set-pmf-iff})$

end

end

lemma $\text{set-pmf-binomial-0}[\text{simp}]$: $\text{set-pmf } (\text{binomial-pmf } n \ 0) = \{0\}$
by $(\text{simp add: set-pmf-binomial-eq})$

lemma $\text{set-pmf-binomial-1}[\text{simp}]$: $\text{set-pmf } (\text{binomial-pmf } n \ 1) = \{n\}$
by $(\text{simp add: set-pmf-binomial-eq})$

lemma $\text{set-pmf-binomial}[\text{simp}]$: $0 < p \implies p < 1 \implies \text{set-pmf } (\text{binomial-pmf } n \ p) = \{..n\}$
by $(\text{simp add: set-pmf-binomial-eq})$

lemma $\text{finite-set-pmf-binomial-pmf} [\text{intro}]$: $p \in \{0..1\} \implies \text{finite } (\text{set-pmf } (\text{binomial-pmf } n \ p))$
by $(\text{subst set-pmf-binomial-eq}) \text{ auto}$

lemma $\text{expectation-binomial-pmf}'$:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$
assumes $p: p \in \{0..1\}$
shows $\text{measure-pmf.expectation } (\text{binomial-pmf } n \ p) \ f =$
 $(\sum k \leq n. (\text{real } (n \text{ choose } k) * p^k * (1 - p)^{n-k})) *_R f \ k)$
using p **by** $(\text{subst integral-measure-pmf}[\text{where } A = \{..n\}])$
 $(\text{auto simp: set-pmf-binomial-eq split: if-splits})$

lemma $\text{integrable-binomial-pmf} [\text{simp, intro}]$:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{banach, second-countable-topology}\}$
assumes $p: p \in \{0..1\}$
shows $\text{integrable } (\text{binomial-pmf } n \ p) \ f$
by $(\text{rule integrable-measure-pmf-finite}) (\text{use assms in auto})$

context includes *lifting-syntax*
begin

lemma *bind-pmf-parametric* [*transfer-rule*]:
 $(\text{rel-pmf } A \text{ ===> } (A \text{ ===> } \text{rel-pmf } B) \text{ ===> } \text{rel-pmf } B) \text{ bind-pmf bind-pmf}$
by(*blast intro: rel-pmf-bindI dest: rel-funD*)

lemma *return-pmf-parametric* [*transfer-rule*]: $(A \text{ ===> } \text{rel-pmf } A) \text{ return-pmf}$
 return-pmf
by(*rule rel-funI simp*)

end

primrec *replicate-pmf* :: $\text{nat} \Rightarrow 'a \text{ pmf} \Rightarrow 'a \text{ list pmf}$ **where**
 $\text{replicate-pmf } 0 = \text{return-pmf } []$
 $| \text{replicate-pmf } (\text{Suc } n) \text{ } p = \text{do } \{x \leftarrow p; xs \leftarrow \text{replicate-pmf } n \text{ } p; \text{return-pmf } (x \# xs)\}$

lemma *replicate-pmf-1*: $\text{replicate-pmf } 1 \text{ } p = \text{map-pmf } (\lambda x. [x]) \text{ } p$
by (*simp add: map-pmf-def bind-return-pmf*)

lemma *set-replicate-pmf*:
 $\text{set-pmf } (\text{replicate-pmf } n \text{ } p) = \{xs \in \text{lists } (\text{set-pmf } p). \text{length } xs = n\}$
by (*induction n (auto simp: length-Suc-conv)*)

lemma *replicate-pmf-distrib*:
 $\text{replicate-pmf } (m + n) \text{ } p =$
 $\text{do } \{xs \leftarrow \text{replicate-pmf } m \text{ } p; ys \leftarrow \text{replicate-pmf } n \text{ } p; \text{return-pmf } (xs @ ys)\}$
by (*induction m (simp-all add: bind-return-pmf bind-return-pmf' bind-assoc-pmf)*)

lemma *power-diff'*:
assumes $b \leq a$
shows $x \wedge (a - b) = (\text{if } x = 0 \wedge a = b \text{ then } 1 \text{ else } x \wedge a / (x :: 'a :: \text{field}) \wedge b)$
proof (*cases x = 0*)
case *True*
with *assms* **show** *?thesis* **by** (*cases a - b simp-all*)
qed (*insert assms, simp-all add: power-diff*)

lemma *binomial-pmf-Suc*:
assumes $p \in \{0..1\}$
shows $\text{binomial-pmf } (\text{Suc } n) \text{ } p =$
 $\text{do } \{b \leftarrow \text{bernoulli-pmf } p;$
 $k \leftarrow \text{binomial-pmf } n \text{ } p;$
 $\text{return-pmf } ((\text{if } b \text{ then } 1 \text{ else } 0) + k)\}$ (*is - = ?rhs*)
proof (*intro pmf-eqI*)
fix k
have $A: \text{indicator } \{\text{Suc } a\} (\text{Suc } b) = \text{indicator } \{a\} b$ **for** $a \text{ } b$
by (*simp add: indicator-def*)

```

show pmf (binomial-pmf (Suc n) p) k = pmf ?rhs k
by (cases k; cases k > n)
      (insert assms, auto simp: pmf-bind measure-pmf-single A field-split-simps
algebra-simps
      not-less less-eq-Suc-le [symmetric] power-diff')
qed

```

```

lemma binomial-pmf-0:  $p \in \{0..1\} \implies \text{binomial-pmf } 0 \ p = \text{return-pmf } 0$ 
by (rule pmf-eqI) (simp-all add: indicator-def)

```

```

lemma binomial-pmf-altdef:
  assumes  $p \in \{0..1\}$ 
  shows  $\text{binomial-pmf } n \ p = \text{map-pmf } (\text{length} \circ \text{filter id}) \ (\text{replicate-pmf } n \ (\text{bernoulli-pmf } p))$ 
by (induction n)
      (insert assms, auto simp: binomial-pmf-Suc map-pmf-def bind-return-pmf
bind-assoc-pmf
      bind-return-pmf' binomial-pmf-0 intro!: bind-pmf-cong)

```

20.7 Negative Binomial distribution

The negative binomial distribution counts the number of times a weighted coin comes up tails before having come up heads n times. In other words: how many failures do we see before seeing the n -th success?

An alternative view is that the negative binomial distribution is the sum of n i.i.d. geometric variables (this is the definition that we use).

Note that there are sometimes different conventions for this distributions in the literature; for instance, sometimes the number of *attempts* is counted instead of the number of failures. This only shifts the entire distribution by a constant number and is thus not a big difference. I think that the convention we use is the most natural one since the support of the distribution starts at 0, whereas for the other convention it starts at n .

```

primrec neg-binomial-pmf ::  $\text{nat} \Rightarrow \text{real} \Rightarrow \text{nat pmf}$  where
  neg-binomial-pmf 0 p = return-pmf 0
| neg-binomial-pmf (Suc n) p =
  map-pmf ( $\lambda(x,y). (x + y)$ ) (pair-pmf (geometric-pmf p) (neg-binomial-pmf n p))

```

```

lemma neg-binomial-pmf-Suc-0 [simp]:  $\text{neg-binomial-pmf } (\text{Suc } 0) \ p = \text{geometric-pmf } p$ 
by (auto simp: pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf bind-return-pmf')

```

```

lemmas neg-binomial-pmf-Suc [simp del] = neg-binomial-pmf.simps(2)

```

```

lemma neg-binomial-prob-1 [simp]:  $\text{neg-binomial-pmf } n \ 1 = \text{return-pmf } 0$ 
by (induction n) (simp-all add: neg-binomial-pmf-Suc)

```

We can now show the aforementioned intuition about counting the failures before the n -th success with the following recurrence:

lemma *neg-binomial-pmf-unfold*:
assumes $p: p \in \{0 < .. 1\}$
shows $\text{neg-binomial-pmf } (\text{Suc } n) \ p =$
 $\text{do } \{b \leftarrow \text{bernoulli-pmf } p;$
 $\text{if } b \text{ then } \text{neg-binomial-pmf } n \ p \text{ else } \text{map-pmf } \text{Suc } (\text{neg-binomial-pmf}$
 $(\text{Suc } n) \ p)\}$
(is - = ?rhs)
unfolding *neg-binomial-pmf-Suc*
by (*subst geometric-bind-pmf-unfold[OF p]*)
(auto simp: map-pmf-def pair-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf'
intro!: bind-pmf-cong)

Next, we show an explicit formula for the probability mass function of the negative binomial distribution:

lemma *pmf-neg-binomial*:
assumes $p: p \in \{0 < .. 1\}$
shows $\text{pmf } (\text{neg-binomial-pmf } n \ p) \ k = \text{real } ((k + n - 1) \text{ choose } k) * p ^ n * (1 - p) ^ k$
proof (*induction n arbitrary: k*)
case 0
thus *?case using assms by (auto simp: indicator-def)*
next
case (*Suc n*)
show *?case*
proof (*cases n = 0*)
case True
thus *?thesis using assms by auto*
next
case False
let $?f = \text{pmf } (\text{neg-binomial-pmf } n \ p)$
have $\text{pmf } (\text{neg-binomial-pmf } (\text{Suc } n) \ p) \ k =$
 $\text{pmf } (\text{geometric-pmf } p \gg (\lambda x. \text{map-pmf } ((+) \ x) (\text{neg-binomial-pmf } n$
 $p))) \ k$
by (*auto simp: pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf*
neg-binomial-pmf-Suc)
also have $\dots = \text{measure-pmf.expectation } (\text{geometric-pmf } p)$
 $(\lambda x. \text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) ((+) \ x - \{k\}))$
by (*simp add: pmf-bind pmf-map*)
also have $(\lambda x. (+) \ x - \{k\}) = (\lambda x. \text{if } x \leq k \text{ then } \{k - x\} \text{ else } \{\})$
by (*auto simp: fun-eq-iff*)
also have $(\lambda x. \text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) (\dots \ x)) =$
 $(\lambda x. \text{if } x \leq k \text{ then } ?f(k - x) \text{ else } 0)$
by (*auto simp: fun-eq-iff measure-pmf-single*)
also have $\text{measure-pmf.expectation } (\text{geometric-pmf } p) \dots =$
 $(\sum i \leq k. \text{pmf } (\text{neg-binomial-pmf } n \ p) (k - i) * \text{pmf } (\text{geometric-pmf}$
 $p) \ i)$
by (*subst integral-measure-pmf-real[where A = {..k}] (auto split: if-splits)*)

```

also have ... =  $p^{(n+1)} * (1-p)^k * \text{real } (\sum_{i \leq k}. (k - i + n - 1) \text{ choose } (k - i))$ 
unfolding sum-distrib-left of-nat-sum
proof (intro sum.cong refl, goal-cases)
  case (1 i)
  have  $\text{pmf } (\text{neg-binomial-pmf } n \ p) \ (k - i) * \text{pmf } (\text{geometric-pmf } p) \ i =$ 
     $\text{real } ((k - i + n - 1) \text{ choose } (k - i)) * p^{(n+1)} * ((1-p)^{(k-i)} * (1-p)^i)$ 
    using assms Suc.IH by (simp add: mult-ac)
  also have  $(1-p)^{(k-i)} * (1-p)^i = (1-p)^k$ 
    using 1 by (subst power-add [symmetric] auto)
  finally show ?case by simp
qed
also have  $(\sum_{i \leq k}. (k - i + n - 1) \text{ choose } (k - i)) = (\sum_{i \leq k}. (n - 1 + i) \text{ choose } i)$ 
    by (intro sum.reindex-bij-witness[of -  $\lambda i. k - i$   $\lambda i. k - i$ ])
    (use <n ≠ 0> in <auto simp: algebra-simps>)
also have ... =  $(n + k) \text{ choose } k$ 
    by (subst sum-choose-lower) (use <n ≠ 0> in auto)
finally show ?thesis
    by (simp add: add-ac)
qed
qed

```

lemma *gbinomial-0-left*: $0 \text{ gchoose } k = (\text{if } k = 0 \text{ then } 1 \text{ else } 0)$
by (*cases k*) *auto*

The following alternative formula highlights why it is called ‘negative binomial distribution’:

```

lemma pmf-neg-binomial':
  assumes  $p: p \in \{0 \dots 1\}$ 
  shows  $\text{pmf } (\text{neg-binomial-pmf } n \ p) \ k = (-1)^k * ((- \text{real } n) \text{ gchoose } k) * p^n * (1 - p)^k$ 
proof (cases n > 0)
  case n: True
  have  $\text{pmf } (\text{neg-binomial-pmf } n \ p) \ k = \text{real } ((k + n - 1) \text{ choose } k) * p^n * (1 - p)^k$ 
    by (rule pmf-neg-binomial fact+)
  also have  $\text{real } ((k + n - 1) \text{ choose } k) = ((\text{real } k + \text{real } n - 1) \text{ gchoose } k)$ 
    using n by (subst binomial-gbinomial) (auto simp: of-nat-diff)
  also have ... =  $(-1)^k * ((- \text{real } n) \text{ gchoose } k)$ 
    by (subst gbinomial-negated-upper) auto
  finally show ?thesis by simp
qed (auto simp: indicator-def gbinomial-0-left)

```

The cumulative distribution function of the negative binomial distribution can be expressed in terms of that of the ‘normal’ binomial distribution.

lemma *prob-neg-binomial-pmf-atMost*:

```

assumes  $p: p \in \{0 < .. 1\}$ 
shows  $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{..k\} =$ 
 $\text{measure-pmf.prob } (\text{binomial-pmf } (n + k) \ (1 - p)) \ \{..k\}$ 
proof (cases  $n = 0$ )
  case [simp]: True
    have  $\text{set-pmf } (\text{binomial-pmf } (n + k) \ (1 - p)) \subseteq \{..n+k\}$ 
    using  $p$  by (subst set-pmf-binomial-eq) auto
    hence  $\text{measure-pmf.prob } (\text{binomial-pmf } (n + k) \ (1 - p)) \ \{..k\} = 1$ 
    by (subst measure-pmf.prob-eq-1) (auto intro!: AE-pmfI)
    thus ?thesis by simp
  next
    case False
    hence  $n: n > 0$  by auto
    have  $\text{measure-pmf.prob } (\text{binomial-pmf } (n + k) \ (1 - p)) \ \{..k\} = (\sum i \leq k. \text{pmf } (\text{binomial-pmf } (n + k) \ (1 - p)) \ i)$ 
    by (intro measure-measure-pmf-finite) auto
    also have  $\dots = (\sum i \leq k. \text{real } ((n + k) \text{ choose } i) * p^{\wedge (n + k - i)} * (1 - p)^{\wedge i})$ 
    using  $p$  by (simp add: mult-ac)
    also have  $\dots = p^{\wedge n} * (\sum i \leq k. \text{real } ((n + k) \text{ choose } i) * (1 - p)^{\wedge i} * p^{\wedge (k - i)})$ 
    unfolding sum-distrib-left by (intro sum.cong) (auto simp: algebra-simps simp flip: power-add)
    also have  $(\sum i \leq k. \text{real } ((n + k) \text{ choose } i) * (1 - p)^{\wedge i} * p^{\wedge (k - i)}) =$ 
 $(\sum i \leq k. ((n + i - 1) \text{ choose } i) * (1 - p)^{\wedge i})$ 
    using gbinomial-partial-sum-poly-xpos[of  $k$  real  $n$   $1 - p$   $p$ ]  $n$ 
    by (simp add: binomial-gbinomial add-ac of-nat-diff)
    also have  $p^{\wedge n} * \dots = (\sum i \leq k. \text{pmf } (\text{neg-binomial-pmf } n \ p) \ i)$ 
    using  $p$  unfolding sum-distrib-left by (simp add: pmf-neg-binomial algebra-simps)
    also have  $\dots = \text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{..k\}$ 
    by (intro measure-measure-pmf-finite [symmetric]) auto
    finally show ?thesis ..
qed

```

lemma prob-neg-binomial-pmf-lessThan:

```

assumes  $p: p \in \{0 < .. 1\}$ 
shows  $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{..<k\} =$ 
 $\text{measure-pmf.prob } (\text{binomial-pmf } (n + k - 1) \ (1 - p)) \ \{..<k\}$ 
proof (cases  $k = 0$ )
  case False
    hence  $\{..<k\} = \{..k-1\}$ 
    by auto
    thus ?thesis
    using prob-neg-binomial-pmf-atMost[OF  $p$ , of  $n \ k - 1$ ] False by simp
qed auto

```

The expected value of the negative binomial distribution is $n(1 - p)/p$:

lemma nn-integral-neg-binomial-pmf-real:

```

assumes  $p: p \in \{0 < .. 1\}$ 
shows  $nn\text{-integral} \text{ (measure-pmf (neg-binomial-pmf } n \text{ } p)) \text{ of-nat} = \text{ennreal } (n * (1 - p) / p)$ 
proof (induction  $n$ )
  case 0
  thus ?case by auto
next
  case (Suc  $n$ )
  have  $nn\text{-integral} \text{ (measure-pmf (neg-binomial-pmf (Suc } n \text{) } p)) \text{ of-nat} =$ 
     $nn\text{-integral} \text{ (measure-pmf (geometric-pmf } p)) \text{ of-nat} +$ 
     $nn\text{-integral} \text{ (measure-pmf (neg-binomial-pmf } n \text{ } p)) \text{ of-nat}$ 
    by (simp add: neg-binomial-pmf-Suc case-prod-unfold nn-integral-add nn-integral-pair-pmf')
  also have  $nn\text{-integral} \text{ (measure-pmf (geometric-pmf } p)) \text{ of-nat} = \text{ennreal } ((1-p) / p)$ 
  unfolding ennreal-of-nat-eq-real-of-nat
  using expectation-geometric-pmf[OF  $p$ ] integrable-real-geometric-pmf[OF  $p$ ]
  by (subst nn-integral-eq-integral) auto
  also have  $nn\text{-integral} \text{ (measure-pmf (neg-binomial-pmf } n \text{ } p)) \text{ of-nat} = n * (1 - p) / p$ 
  using  $p$  by (subst Suc.IH)
  (auto simp: ennreal-of-nat-eq-real-of-nat ennreal-mult simp flip: divide-ennreal ennreal-minus)
  also have  $\text{ennreal } ((1 - p) / p) + \text{ennreal } (\text{real } n * (1 - p) / p) =$ 
     $\text{ennreal } ((1-p) / p + \text{real } n * (1 - p) / p)$ 
    by (intro ennreal-plus [symmetric] divide-nonneg-pos mult-nonneg-nonneg) (use  $p$  in auto)
  also have  $(1-p) / p + \text{real } n * (1 - p) / p = \text{real } (\text{Suc } n) * (1 - p) / p$ 
    using  $p$  by (auto simp: field-simps)
  finally show ?case
    by (simp add: ennreal-of-nat-eq-real-of-nat)
qed

```

```

lemma integrable-neg-binomial-pmf-real:
  assumes  $p: p \in \{0 < .. 1\}$ 
  shows  $\text{integrable} \text{ (measure-pmf (neg-binomial-pmf } n \text{ } p)) \text{ real}$ 
  using nn-integral-neg-binomial-pmf-real[OF  $p$ , of  $n$ ]
  by (subst integrable-iff-bounded) (auto simp flip: ennreal-of-nat-eq-real-of-nat)

```

```

lemma expectation-neg-binomial-pmf:
  assumes  $p: p \in \{0 < .. 1\}$ 
  shows  $\text{measure-pmf.expectation} \text{ (neg-binomial-pmf } n \text{ } p) \text{ real} = n * (1 - p) / p$ 
proof -
  have  $nn\text{-integral} \text{ (measure-pmf (neg-binomial-pmf } n \text{ } p)) \text{ of-nat} = \text{ennreal } (n * (1 - p) / p)$ 
    by (intro nn-integral-neg-binomial-pmf-real  $p$ )
  also have  $\text{of-nat} = (\lambda x. \text{ennreal } (\text{real } x))$ 
    by (simp add: ennreal-of-nat-eq-real-of-nat fun-eq-iff)
  finally show ?thesis
    using  $p$  by (subst (asm) nn-integral-eq-integrable) auto

```

qed

20.8 PMFs from association lists

definition *pmf-of-list* :: $('a \times \text{real}) \text{ list} \Rightarrow 'a \text{ pmf}$ **where**

pmf-of-list *xs* = *embed-pmf* $(\lambda x. \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) \text{ xs})))$

definition *pmf-of-list-wf* **where**

pmf-of-list-wf *xs* $\longleftrightarrow (\forall x \in \text{set } (\text{map } \text{snd } \text{xs}) . x \geq 0) \wedge \text{sum-list } (\text{map } \text{snd } \text{xs}) = 1$

lemma *pmf-of-list-wfI*:

$(\bigwedge x. x \in \text{set } (\text{map } \text{snd } \text{xs}) \implies x \geq 0) \implies \text{sum-list } (\text{map } \text{snd } \text{xs}) = 1 \implies \text{pmf-of-list-wf } \text{xs}$

unfolding *pmf-of-list-wf-def* **by** *simp*

context

begin

private lemma *pmf-of-list-aux*:

assumes $\bigwedge x. x \in \text{set } (\text{map } \text{snd } \text{xs}) \implies x \geq 0$

assumes $\text{sum-list } (\text{map } \text{snd } \text{xs}) = 1$

shows $(\int^+ x. \text{ennreal } (\text{sum-list } (\text{map } \text{snd } [z \leftarrow \text{xs} . \text{fst } z = x]))) \partial \text{count-space UNIV} = 1$

proof –

have $(\int^+ x. \text{ennreal } (\text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) \text{ xs})))) \partial \text{count-space UNIV} =$

$(\int^+ x. \text{ennreal } (\text{sum-list } (\text{map } (\lambda(x',p). \text{indicator } \{x'\} x * p) \text{ xs})))$

$\partial \text{count-space UNIV}$

apply (*intro nn-integral-cong ennreal-cong, subst sum-list-map-filter*)

apply (*rule arg-cong[where f = sum-list]*)

apply (*auto cong: map-cong*)

done

also have $\dots = (\sum (x',p) \leftarrow \text{xs}. (\int^+ x. \text{ennreal } (\text{indicator } \{x'\} x * p) \partial \text{count-space UNIV}))$

using *assms(1)*

proof (*induction xs*)

case (*Cons x xs*)

from *Cons.prem*s **have** $\text{snd } x \geq 0$ **by** *simp*

moreover have $b \geq 0$ **if** $(a,b) \in \text{set } \text{xs}$ **for** *a b*

using *Cons.prem*s[*of b*] **that by** *force*

ultimately have $(\int^+ y. \text{ennreal } (\sum (x', p) \leftarrow x \# \text{xs}. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV}) =$

$(\int^+ y. \text{ennreal } (\text{indicator } \{\text{fst } x\} y * \text{snd } x) +$

$\text{ennreal } (\sum (x', p) \leftarrow \text{xs}. \text{indicator } \{x'\} y * p) \partial \text{count-space UNIV})$

by (*intro nn-integral-cong, subst ennreal-plus [symmetric]*)

(*auto simp: case-prod-unfold indicator-def intro!: sum-list-nonneg*)

also have $\dots = (\int^+ y. \text{ennreal } (\text{indicator } \{\text{fst } x\} y * \text{snd } x) \partial \text{count-space UNIV}) +$

```

    (∫+ y. ennreal (∑ (x', p) ← xs. indicator {x'} y * p) ∂count-space
UNIV)
  by (intro nn-integral-add)
    (force intro!: sum-list-nonneg AE-I2 intro: Cons simp: indicator-def)+
  also have (∫+ y. ennreal (∑ (x', p) ← xs. indicator {x'} y * p) ∂count-space
UNIV) =
    (∑ (x', p) ← xs. (∫+ y. ennreal (indicator {x'} y * p) ∂count-space
UNIV))
  using Cons(1) by (intro Cons) simp-all
  finally show ?case by (simp add: case-prod-unfold)
qed simp
also have ... = (∑ (x', p) ← xs. ennreal p * (∫+ x. indicator {x'} x ∂count-space
UNIV))
  using assms(1)
  by (simp cong: map-cong only: case-prod-unfold, subst nn-integral-cmult [symmetric])
    (auto intro!: assms(1) simp: max-def times-ereal.simps [symmetric] mult-ac
ereal-indicator
      simp del: times-ereal.simps)+
  also from assms have ... = sum-list (map snd xs) by (simp add: case-prod-unfold
sum-list-ennreal)
  also have ... = 1 using assms(2) by simp
  finally show ?thesis .
qed

```

```

lemma pmf-pmf-of-list:
  assumes pmf-of-list-wf xs
  shows pmf (pmf-of-list xs) x = sum-list (map snd (filter (λz. fst z = x) xs))
  using assms pmf-of-list-aux[of xs] unfolding pmf-of-list-def pmf-of-list-wf-def
  by (subst pmf-embed-pmf) (auto intro!: sum-list-nonneg)

```

end

```

lemma set-pmf-of-list:
  assumes pmf-of-list-wf xs
  shows set-pmf (pmf-of-list xs) ⊆ set (map fst xs)
proof clarify
  fix x assume A: x ∈ set-pmf (pmf-of-list xs)
  show x ∈ set (map fst xs)
  proof (rule ccontr)
    assume x ∉ set (map fst xs)
    hence [z ← xs . fst z = x] = [] by (auto simp: filter-empty-conv)
    with A assms show False by (simp add: pmf-pmf-of-list set-pmf-eq)
  qed
qed

```

```

lemma finite-set-pmf-of-list:
  assumes pmf-of-list-wf xs
  shows finite (set-pmf (pmf-of-list xs))
  using assms by (rule finite-subset[OF set-pmf-of-list]) simp-all

```


lemma *emeasure-Int-set-pmf*:

emeasure (*measure-pmf* *p*) (*A* \cap *set-pmf* *p*) = *emeasure* (*measure-pmf* *p*) *A*
by (*rule* *emeasure-eq-AE*) (*auto simp*: *AE-measure-pmf-iff*)

lemma *measure-Int-set-pmf*:

measure (*measure-pmf* *p*) (*A* \cap *set-pmf* *p*) = *measure* (*measure-pmf* *p*) *A*
using *emeasure-Int-set-pmf*[*of* *p* *A*] **by** (*simp add*: *Sigma-Algebra.measure-def*)

lemma *measure-prob-cong-0*:

assumes $\bigwedge x. x \in A - B \implies \text{pmf } p \ x = 0$
assumes $\bigwedge x. x \in B - A \implies \text{pmf } p \ x = 0$
shows *measure* (*measure-pmf* *p*) *A* = *measure* (*measure-pmf* *p*) *B*

proof –

have *measure-pmf.prob* *p* *A* = *measure-pmf.prob* *p* (*A* \cap *set-pmf* *p*)
by (*simp add*: *measure-Int-set-pmf*)
also have *A* \cap *set-pmf* *p* = *B* \cap *set-pmf* *p*
using *assms* **by** (*auto simp*: *set-pmf-eq*)
also have *measure-pmf.prob* *p* ... = *measure-pmf.prob* *p* *B*
by (*simp add*: *measure-Int-set-pmf*)
finally show ?*thesis* .

qed

lemma *emeasure-pmf-of-list*:

assumes *pmf-of-list-wf* *xs*
shows *emeasure* (*pmf-of-list* *xs*) *A* = *ennreal* (*sum-list* (*map snd* (*filter* ($\lambda x. \text{fst } x \in A$) *xs*))))

proof –

have *emeasure* (*pmf-of-list* *xs*) *A* = *nn-integral* (*measure-pmf* (*pmf-of-list* *xs*))
(*indicator* *A*)

by *simp*

also from *assms*

have ... = ($\sum x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{ennreal } (\text{sum-list } (\text{map snd } [z \leftarrow xs . \text{fst } z = x])))$)

by (*subst nn-integral-measure-pmf-finite*) (*simp-all add*: *finite-set-pmf-of-list pmf-pmf-of-list Int-def*)

also from *assms*

have ... = *ennreal* ($\sum x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A. \text{sum-list } (\text{map snd } [z \leftarrow xs . \text{fst } z = x]))$)

by (*subst sum-ennreal*) (*auto simp*: *pmf-of-list-wf-def intro!*: *sum-list-nonneg*)

also have ... = *ennreal* ($\sum x \in \text{set-pmf } (\text{pmf-of-list } xs) \cap A.$

indicator *A* $x * \text{pmf } (\text{pmf-of-list } xs) \ x$) (*is* - = *ennreal* ?*S*)

using *assms* **by** (*intro ennreal-cong sum.cong*) (*auto simp*: *pmf-pmf-of-list*)

also have ?*S* = ($\sum x \in \text{set-pmf } (\text{pmf-of-list } xs). \text{indicator } A \ x * \text{pmf } (\text{pmf-of-list } xs) \ x$)

using *assms* **by** (*intro sum.mono-neutral-left set-pmf-of-list finite-set-pmf-of-list*)

auto

also have ... = ($\sum x \in \text{set } (\text{map fst } xs). \text{indicator } A \ x * \text{pmf } (\text{pmf-of-list } xs) \ x$)

using *assms* **by** (*intro sum.mono-neutral-left set-pmf-of-list*) (*auto simp*: *set-pmf-eq*)

```

also have ... = ( $\sum x \in \text{set } (\text{map } \text{fst } xs). \text{indicator } A \ x * \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x) \ xs)))$ )
using assms by (simp add: pmf-pmf-of-list)
also have ... = ( $\sum x \in \text{set } (\text{map } \text{fst } xs). \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda z. \text{fst } z = x \wedge x \in A) \ xs)))$ )
by (intro sum.cong) (auto simp: indicator-def)
also have ... = ( $\sum x \in \text{set } (\text{map } \text{fst } xs). (\sum xa = 0..<\text{length } xs. \text{if } \text{fst } (xs \ ! \ xa) = x \wedge x \in A \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0))$ )
by (intro sum.cong refl, subst sum-list-map-filter', subst sum-list-sum-nth) simp
also have ... = ( $\sum xa = 0..<\text{length } xs. (\sum x \in \text{set } (\text{map } \text{fst } xs). \text{if } \text{fst } (xs \ ! \ xa) = x \wedge x \in A \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0))$ )
by (rule sum.swap)
also have ... = ( $\sum xa = 0..<\text{length } xs. \text{if } \text{fst } (xs \ ! \ xa) \in A \text{ then } (\sum x \in \text{set } (\text{map } \text{fst } xs). \text{if } x = \text{fst } (xs \ ! \ xa) \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0) \text{ else } 0$ )
by (auto intro!: sum.cong sum.neutral simp del: sum.delta)
also have ... = ( $\sum xa = 0..<\text{length } xs. \text{if } \text{fst } (xs \ ! \ xa) \in A \text{ then } \text{snd } (xs \ ! \ xa) \text{ else } 0$ )
by (intro sum.cong refl) (simp-all add: sum.delta)
also have ... =  $\text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda x. \text{fst } x \in A) \ xs))$ 
by (subst sum-list-map-filter', subst sum-list-sum-nth) simp-all
finally show ?thesis .
qed

```

lemma *measure-pmf-of-list*:

```

assumes pmf-of-list-wf xs
shows  $\text{measure } (\text{pmf-of-list } xs) \ A = \text{sum-list } (\text{map } \text{snd } (\text{filter } (\lambda x. \text{fst } x \in A) \ xs))$ 
using assms unfolding pmf-of-list-wf-def Sigma-Algebra.measure-def
by (subst emeasure-pmf-of-list [OF assms], subst enn2real-ennreal) (auto intro!: sum-list-nonneg)

```

lemma *sum-list-nonneg-eq-zero-iff*:

```

fixes xs :: 'a :: linordered-ab-group-add list
shows  $(\bigwedge x. x \in \text{set } xs \implies x \geq 0) \implies \text{sum-list } xs = 0 \iff \text{set } xs \subseteq \{0\}$ 
proof (induction xs)
case (Cons x xs)
from Cons.prems have  $\text{sum-list } (x \# xs) = 0 \iff x = 0 \wedge \text{sum-list } xs = 0$ 
unfolding sum-list-simps by (subst add-nonneg-eq-0-iff) (auto intro: sum-list-nonneg)
with Cons.IH Cons.prems show ?case by simp
qed simp-all

```

lemma *sum-list-filter-nonzero*:

```

 $\text{sum-list } (\text{filter } (\lambda x. x \neq 0) \ xs) = \text{sum-list } xs$ 
by (induction xs) simp-all

```

lemma *set-pmf-of-list-eq*:

```

assumes pmf-of-list-wf  $xs \wedge x. x \in \text{snd} \text{ ` } \text{set } xs \implies x > 0$ 
shows   set-pmf (pmf-of-list  $xs$ ) = fst ` set  $xs$ 
proof
  {
    fix  $x$  assume  $A: x \in \text{fst} \text{ ` } \text{set } xs$  and  $B: x \notin \text{set-pmf} \text{ (pmf-of-list } xs)$ 
    then obtain  $y$  where  $y: (x, y) \in \text{set } xs$  by auto
    from  $B$  have  $\text{sum-list} \text{ (map snd [z←xs. fst z = x])} = 0$ 
      by (simp add: pmf-pmf-of-list[OF assms(1)] set-pmf-eq)
    moreover from  $y$  have  $y \in \text{snd} \text{ ` } \{xa \in \text{set } xs. \text{fst } xa = x\}$  by force
    ultimately have  $y = 0$  using  $\text{assms}(1)$ 
      by (subst (asm) sum-list-nonneg-eq-zero-iff) (auto simp: pmf-of-list-wf-def)
    with  $\text{assms}(2)$   $y$  have False by force
  }
  thus  $\text{fst} \text{ ` } \text{set } xs \subseteq \text{set-pmf} \text{ (pmf-of-list } xs)$  by blast
qed (insert set-pmf-of-list[OF assms(1)], simp-all)

```

lemma *pmf-of-list-remove-zeros:*

```

assumes pmf-of-list-wf  $xs$ 
defines  $xs' \equiv \text{filter} \text{ (}\lambda z. \text{snd } z \neq 0 \text{) } xs$ 
shows   pmf-of-list-wf  $xs'$  pmf-of-list  $xs' = \text{pmf-of-list } xs$ 
proof –
  have  $\text{map snd [z←xs. snd z} \neq 0] = \text{filter} \text{ (}\lambda x. x \neq 0 \text{) (map snd } xs)$ 
    by (induction xs) simp-all
  with  $\text{assms}(1)$  show wf: pmf-of-list-wf  $xs'$ 
    by (auto simp: pmf-of-list-wf-def xs'-def sum-list-filter-nonzero)
  have  $\text{sum-list} \text{ (map snd [z←xs'. fst z = i])} = \text{sum-list} \text{ (map snd [z←xs. fst z$ 
     $= i])}$  for  $i$ 
    unfolding  $xs'$ -def by (induction xs) simp-all
  with  $\text{assms}(1)$  wf show  $\text{pmf-of-list } xs' = \text{pmf-of-list } xs$ 
    by (intro pmf-eqI) (simp-all add: pmf-pmf-of-list)
qed
end

```

21 Code generation for PMFs

theory *PMF-Impl*

imports *Probability-Mass-Function HOL-Library.AList-Mapping*
begin

21.1 General code generation setup

definition *pmf-of-mapping* :: $('a, \text{real}) \text{ mapping} \Rightarrow 'a \text{ pmf}$ **where**
 $\text{pmf-of-mapping } m = \text{embed-pmf} \text{ (Mapping.lookup-default } 0 \text{ } m)$

lemma *nn-integral-lookup-default:*

```

fixes  $m :: ('a, \text{real}) \text{ mapping}$ 
assumes finite (Mapping.keys  $m$ ) All-mapping  $m \text{ (}\lambda x. x \geq 0 \text{)}$ 
shows   nn-integral (count-space UNIV)  $(\lambda k. \text{ennreal} \text{ (Mapping.lookup-default } 0$ 

```

$m\ k)) =$
 $\text{ennreal } (\sum k \in \text{Mapping.keys } m. \text{Mapping.lookup-default } 0\ m\ k)$
proof –
have $\text{nn-integral } (\text{count-space } \text{UNIV}) (\lambda k. \text{ennreal } (\text{Mapping.lookup-default } 0\ m\ k)) =$
 $(\sum x \in \text{Mapping.keys } m. \text{ennreal } (\text{Mapping.lookup-default } 0\ m\ x))$ **using**
 assms
by $(\text{subst nn-integral-count-space'} [\text{of Mapping.keys } m])$
 $(\text{auto simp: Mapping.lookup-default-def keys-is-none-rep Option.is-none-def})$
also from assms **have** $\dots = \text{ennreal } (\sum k \in \text{Mapping.keys } m. \text{Mapping.lookup-default } 0\ m\ k)$
by $(\text{intro sum-ennreal})$
 $(\text{auto simp: Mapping.lookup-default-def All-mapping-def split: option.splits})$
finally show $?thesis$.
qed

lemma pmf-of-mapping :
assumes $\text{finite } (\text{Mapping.keys } m)$ $\text{All-mapping } m$ $(\lambda p. p \geq 0)$
assumes $(\sum x \in \text{Mapping.keys } m. \text{Mapping.lookup-default } 0\ m\ x) = 1$
shows $\text{pmf } (\text{pmf-of-mapping } m) x = \text{Mapping.lookup-default } 0\ m\ x$
unfolding $\text{pmf-of-mapping-def}$
proof $(\text{intro pmf-embed-pmf})$
from assms **show** $(\int^+ x. \text{ennreal } (\text{Mapping.lookup-default } 0\ m\ x) \partial \text{count-space } \text{UNIV}) = 1$
by $(\text{subst nn-integral-lookup-default})$ (simp-all)
qed $(\text{insert assms, simp add: All-mapping-def Mapping.lookup-default-def split: option.splits})$

lemma $\text{pmf-of-set-pmf-of-mapping}$:
assumes $A \neq \{\}$ $\text{set } xs = A$ $\text{distinct } xs$
shows $\text{pmf-of-set } A = \text{pmf-of-mapping } (\text{Mapping.tabulate } xs (\lambda x. 1 / \text{real } (\text{length } xs)))$
 $(\text{is } ?lhs = ?rhs)$
by $(\text{rule pmf-eqI, subst pmf-of-mapping})$
 $(\text{insert assms, auto intro!: All-mapping-tabulate})$
 $\text{simp: Mapping.lookup-default-def lookup-tabulate distinct-card})$

lift-definition $\text{mapping-of-pmf} :: 'a\ \text{pmf} \Rightarrow ('a, \text{real})\ \text{mapping}$ **is**
 $\lambda p\ x. \text{if } \text{pmf } p\ x = 0 \text{ then None else Some } (\text{pmf } p\ x)$.

lemma $\text{lookup-default-mapping-of-pmf}$:
 $\text{Mapping.lookup-default } 0\ (\text{mapping-of-pmf } p) x = \text{pmf } p\ x$
by $(\text{simp add: mapping-of-pmf.abs-eq lookup-default-def Mapping.lookup.abs-eq})$

context
begin

interpretation pmf-as-function .

lemma *nn-integral-pmf-eq-1*: $(\int^+ x. \text{ennreal } (\text{pmf } p \ x) \ \partial \text{count-space } \text{UNIV}) = 1$
by *transfer simp-all*
end

lemma *pmf-of-mapping-mapping-of-pmf* [*code abstype*]:
 $\text{pmf-of-mapping } (\text{mapping-of-pmf } p) = p$
unfolding *pmf-of-mapping-def*
by (*rule pmf-eqI*, *subst pmf-embed-pmf*)
(insert nn-integral-pmf-eq-1 [of p],
auto simp: lookup-default-mapping-of-pmf split: option.splits)

lemma *mapping-of-pmfI*:
assumes $\bigwedge x. x \in \text{Mapping.keys } m \implies \text{Mapping.lookup } m \ x = \text{Some } (\text{pmf } p \ x)$
assumes $\text{Mapping.keys } m = \text{set-pmf } p$
shows $\text{mapping-of-pmf } p = m$
using *assms by transfer (rule ext, auto simp: set-pmf-eq)*

lemma *mapping-of-pmfI'*:
assumes $\bigwedge x. x \in \text{Mapping.keys } m \implies \text{Mapping.lookup-default } 0 \ m \ x = \text{pmf } p \ x$
assumes $\text{Mapping.keys } m = \text{set-pmf } p$
shows $\text{mapping-of-pmf } p = m$
using *assms unfolding Mapping.lookup-default-def*
by *transfer (rule ext, force simp: set-pmf-eq)*

lemma *return-pmf-code* [*code abstract*]:
 $\text{mapping-of-pmf } (\text{return-pmf } x) = \text{Mapping.update } x \ 1 \ \text{Mapping.empty}$
by (*intro mapping-of-pmfI*) (*auto simp: lookup-update'*)

lemma *pmf-of-set-code-aux*:
assumes $A \neq \{\}$ *set* $xs = A$ *distinct* xs
shows $\text{mapping-of-pmf } (\text{pmf-of-set } A) = \text{Mapping.tabulate } xs \ (\lambda x. 1 / \text{real } (\text{length } xs))$
using *assms*
by (*intro mapping-of-pmfI*, *subst pmf-of-set*)
(auto simp: lookup-tabulate distinct-card)

definition *pmf-of-set-impl* **where**
 $\text{pmf-of-set-impl } A = \text{mapping-of-pmf } (\text{pmf-of-set } A)$

lemma *pmf-of-set-impl-code-alt*:
assumes $A \neq \{\}$ *finite* A
shows $\text{pmf-of-set-impl } A =$
 $(\text{let } p = 1 / \text{real } (\text{card } A)$
 $\text{in } \text{Finite-Set.fold } (\lambda x. \text{Mapping.update } x \ p) \ \text{Mapping.empty } A)$

proof –

define p **where** $p = 1 / \text{real } (\text{card } A)$
let $?m = \text{Finite-Set.fold } (\lambda x. \text{Mapping.update } x \ p) \ \text{Mapping.empty } A$

```

interpret comp-fun-idem  $\lambda x. \text{Mapping.update } x \ p$ 
  by standard (transfer, force simp: fun-eq-iff)+
have keys: Mapping.keys ?m = A
  using assms(2) by (induction A rule: finite-induct) simp-all
have lookup: Mapping.lookup ?m x = Some p if  $x \in A$  for x
  using assms(2) that by (induction A rule: finite-induct) (auto simp: lookup-update')
from keys lookup assms show ?thesis unfolding pmf-of-set-impl-def
  by (intro mapping-of-pmfI) (simp-all add: Let-def p-def)
qed

```

```

lemma pmf-of-set-impl-code [code]:
  pmf-of-set-impl (set xs) =
    (if xs = [] then
      Code.abort (STR "pmf-of-set of empty set") ( $\lambda \cdot. \text{mapping-of-pmf } (\text{pmf-of-set } (\text{set } xs))$ )
    else let xs' = remdups xs; p = 1 / real (length xs') in
      Mapping.tabulate xs' ( $\lambda \cdot. p$ ))
  unfolding pmf-of-set-impl-def
  using pmf-of-set-code-aux[of set xs remdups xs] by (simp add: Let-def)

```

```

lemma pmf-of-set-code [code abstract]:
  mapping-of-pmf (pmf-of-set A) = pmf-of-set-impl A
  by (simp add: pmf-of-set-impl-def)

```

```

lemma pmf-of-multiset-pmf-of-mapping:
  assumes  $A \neq \{\#\}$  set xs = set-mset A distinct xs
  shows mapping-of-pmf (pmf-of-multiset A) = Mapping.tabulate xs ( $\lambda x. \text{count } A \ x \ / \ \text{real } (\text{size } A)$ )
  using assms by (intro mapping-of-pmfI) (auto simp: lookup-tabulate)

```

```

definition pmf-of-multiset-impl where
  pmf-of-multiset-impl A = mapping-of-pmf (pmf-of-multiset A)

```

```

lemma pmf-of-multiset-impl-code-alt:
  assumes  $A \neq \{\#\}$ 
  shows pmf-of-multiset-impl A =
    (let p = 1 / real (size A)
     in fold-mset ( $\lambda x. \text{Mapping.map-default } x \ 0 \ ((+) \ p)$ ) Mapping.empty A)

```

proof –

```

  define p where p = 1 / real (size A)
  interpret comp-fun-commute  $\lambda x. \text{Mapping.map-default } x \ 0 \ ((+) \ p)$ 
  unfolding Mapping.map-default-def [abs-def]
  by (standard, intro mapping-eqI ext)
  (simp-all add: o-def lookup-map-entry' lookup-default' lookup-default-def)
  let ?m = fold-mset ( $\lambda x. \text{Mapping.map-default } x \ 0 \ ((+) \ p)$ ) Mapping.empty A
  have keys: Mapping.keys ?m = set-mset A by (induction A) simp-all
  have lookup: Mapping.lookup-default 0 ?m x = real (count A x) * p for x
  by (induction A)

```

```

  (simp-all add: lookup-map-default' lookup-default-def lookup-empty ring-distrib)
  from keys lookup assms show ?thesis unfolding pmf-of-multiset-impl-def
  by (intro mapping-of-pmfI') (simp-all add: Let-def p-def)
qed

```

```

lemma pmf-of-multiset-impl-code [code]:
  pmf-of-multiset-impl (mset xs) =
    (if xs = [] then
      Code.abort (STR "pmf-of-multiset of empty multiset")
      (λ-. mapping-of-pmf (pmf-of-multiset (mset xs)))
    else let xs' = remdups xs; p = 1 / real (length xs) in
      Mapping.tabulate xs' (λx. real (count (mset xs) x) * p))
  using pmf-of-multiset-pmf-of-mapping[of mset xs remdups xs]
  by (simp add: pmf-of-multiset-impl-def)

```

```

lemma pmf-of-multiset-code [code abstract]:
  mapping-of-pmf (pmf-of-multiset A) = pmf-of-multiset-impl A
  by (simp add: pmf-of-multiset-impl-def)

```

```

lemma bernoulli-pmf-code [code abstract]:
  mapping-of-pmf (bernoulli-pmf p) =
    (if p ≤ 0 then Mapping.update False 1 Mapping.empty
     else if p ≥ 1 then Mapping.update True 1 Mapping.empty
     else Mapping.update False (1 - p) (Mapping.update True p Mapping.empty))
  by (intro mapping-of-pmfI) (auto simp: bernoulli-pmf.rep-eq lookup-update' set-pmf-eq)

```

```

lemma pmf-code [code]: pmf p x = Mapping.lookup-default 0 (mapping-of-pmf p)
x
  unfolding mapping-of-pmf-def Mapping.lookup-default-def
  by (auto split: option.splits simp: id-def Mapping.lookup.abs-eq)

```

```

lemma set-pmf-code [code]: set-pmf p = Mapping.keys (mapping-of-pmf p)
  by transfer (auto simp: dom-def set-pmf-eq)

```

```

lemma keys-mapping-of-pmf [simp]: Mapping.keys (mapping-of-pmf p) = set-pmf
p
  by transfer (auto simp: dom-def set-pmf-eq)

```

```

definition fold-combine-plus where
  fold-combine-plus = comm-monoid-set.F (Mapping.combine ((+) :: real ⇒ -))
  Mapping.empty

```

```

context
begin

```

interpretation *fold-combine-plus*: *combine-mapping-abel-semigroup* $(+)$:: *real* \Rightarrow

-
by *unfold-locales* (*simp-all* *add*: *add-ac*)

qualified lemma *lookup-default-fold-combine-plus*:

fixes $A :: 'b$ *set* **and** $f :: 'b \Rightarrow ('a, \text{real})$ *mapping*

assumes *finite A*

shows $\text{Mapping.lookup-default } 0 \ (\text{fold-combine-plus } f \ A) \ x =$
 $(\sum_{y \in A}. \text{Mapping.lookup-default } 0 \ (f \ y) \ x)$

unfolding *fold-combine-plus-def* **using** *assms*

by (*induction A* *rule*: *finite-induct*)

(*simp-all* *add*: *lookup-default-empty lookup-default-neutral-combine*)

qualified lemma *keys-fold-combine-plus*:

finite A $\Rightarrow \text{Mapping.keys} \ (\text{fold-combine-plus } f \ A) = (\bigcup_{x \in A}. \text{Mapping.keys} \ (f \ x))$

by (*simp* *add*: *fold-combine-plus-def fold-combine-plus.keys-fold-combine*)

qualified lemma *fold-combine-plus-code* [*code*]:

fold-combine-plus g (*set xs*) = *foldr* $(\lambda x. \text{Mapping.combine} \ (+) \ (g \ x)) \ (\text{remdups } xs) \ \text{Mapping.empty}$

by (*simp* *add*: *fold-combine-plus-def fold-combine-plus.fold-combine-code*)

private lemma *lookup-default-0-map-values*:

assumes $f \ x \ 0 = 0$

shows $\text{Mapping.lookup-default } 0 \ (\text{Mapping.map-values } f \ m) \ x = f \ x \ (\text{Mapping.lookup-default } 0 \ m \ x)$

unfolding *Mapping.lookup-default-def*

using *assms* **by** *transfer* (*auto split*: *option.splits*)

qualified lemma *mapping-of-bind-pmf*:

assumes *finite* (*set-pmf p*)

shows *mapping-of-pmf* (*bind-pmf p f*) =

fold-combine-plus $(\lambda x. \text{Mapping.map-values} \ (\lambda \cdot. \ (*) \ (\text{pmf } p \ x))$
 $(\text{mapping-of-pmf} \ (f \ x))) \ (\text{set-pmf } p)$

using *assms*

by (*intro mapping-of-pmfI'*)

(*auto simp*: *keys-fold-combine-plus lookup-default-fold-combine-plus*
pmf-bind integral-measure-pmf lookup-default-0-map-values
lookup-default-mapping-of-pmf mult-ac)

lift-definition *bind-pmf-aux* :: $'a \text{ pmf} \Rightarrow ('a \Rightarrow 'b \text{ pmf}) \Rightarrow 'a \text{ set} \Rightarrow ('b, \text{real})$
mapping is

$\lambda(p :: 'a \text{ pmf}) \ (f :: 'a \Rightarrow 'b \text{ pmf}) \ (A :: 'a \text{ set}) \ (x :: 'b).$

if $x \in (\bigcup_{y \in A}. \text{set-pmf} \ (f \ y))$ *then*

Some (*measure-pmf.expectation p* $(\lambda y. \text{indicator } A \ y * \text{pmf} \ (f \ y) \ x)$)

else None .

lemma *keys-bind-pmf-aux* [*simp*]:

Mapping.keys (*bind-pmf-aux* *p f A*) = ($\bigcup x \in A. \text{set-pmf } (f x)$)
by *transfer* (*auto split: if-splits*)

lemma *lookup-default-bind-pmf-aux*:

Mapping.lookup-default 0 (*bind-pmf-aux* *p f A*) *x* =
 (if *x* $\in (\bigcup y \in A. \text{set-pmf } (f y))$ then
 measure-pmf.expectation *p* ($\lambda y. \text{indicator } A y * \text{pmf } (f y) x$) else 0)
unfolding *lookup-default-def* **by** *transfer' simp-all*

lemma *lookup-default-bind-pmf-aux' [simp]*:

Mapping.lookup-default 0 (*bind-pmf-aux* *p f (set-pmf p)*) *x* = *pmf* (*bind-pmf* *p f*)
x
unfolding *lookup-default-def*
by *transfer* (*auto simp: pmf-bind AE-measure-pmf-iff set-pmf-eq*
 intro!: integral-cong-AE integral-eq-zero-AE)

lemma *bind-pmf-aux-correct*:

mapping-of-pmf (*bind-pmf* *p f*) = *bind-pmf-aux* *p f (set-pmf p)*
by (*intro mapping-of-pmfI'*) *simp-all*

lemma *bind-pmf-aux-code-aux*:

assumes *finite A*
shows *bind-pmf-aux* *p f A* =
 fold-combine-plus ($\lambda x. \text{Mapping.map-values } (\lambda-. (*) (\text{pmf } p x))$
 (*mapping-of-pmf* (*f x*))) *A* (**is** ?lhs = ?rhs)

proof (*intro mapping-eqI'* [where *d* = 0])

fix *x* **assume** *x* $\in \text{Mapping.keys } ?lhs$

then obtain *y* **where** *y*: *y* $\in A$ *x* $\in \text{set-pmf } (f y)$ **by** *auto*

hence *Mapping.lookup-default* 0 ?lhs *x* =

measure-pmf.expectation *p* ($\lambda y. \text{indicator } A y * \text{pmf } (f y) x$)

by (*auto simp: lookup-default-bind-pmf-aux*)

also from *assms* **have** ... = ($\sum y \in A. \text{pmf } p y * \text{pmf } (f y) x$)

by (*subst integral-measure-pmf [of A]*)

 (*auto simp: set-pmf-eq indicator-def mult-ac split: if-splits*)

also from *assms* **have** ... = *Mapping.lookup-default* 0 ?rhs *x*

by (*simp add: lookup-default-fold-combine-plus lookup-default-0-map-values*
 lookup-default-mapping-of-pmf)

finally show *Mapping.lookup-default* 0 ?lhs *x* = *Mapping.lookup-default* 0 ?rhs
x .

qed (*insert assms, simp-all add: keys-fold-combine-plus*)

lemma *bind-pmf-aux-code [code]*:

bind-pmf-aux *p f (set xs)* =
 fold-combine-plus ($\lambda x. \text{Mapping.map-values } (\lambda-. (*) (\text{pmf } p x))$
 (*mapping-of-pmf* (*f x*))) (*set xs*)
by (*rule bind-pmf-aux-code-aux*) *simp-all*

lemmas *bind-pmf-code [code abstract]* = *bind-pmf-aux-correct*

end

hide-const (**open**) *fold-combine-plus*

lift-definition *cond-pmf-impl* :: 'a pmf \Rightarrow 'a set \Rightarrow ('a, real) mapping option **is**
 $\lambda p A.$ if $A \cap \text{set-pmf } p = \{\}$ then None else
 Some ($\lambda x.$ if $x \in A \cap \text{set-pmf } p$ then Some (pmf p x / measure-pmf.prob p A)
 else None) .

lemma *cond-pmf-impl-code-alt*:

assumes *finite A*

shows *cond-pmf-impl* p $A =$ (

let $C = A \cap \text{set-pmf } p$;

prob = ($\sum x \in C. \text{pmf } p$ x)

in if prob = 0 then

None

else

Some (Mapping.map-values ($\lambda y. y$ / prob)

(Mapping.filter ($\lambda k -. k \in C$) (mapping-of-pmf p))))

proof –

define C **where** $C = A \cap \text{set-pmf } p$

define *prob* **where** *prob* = ($\sum x \in C. \text{pmf } p$ x)

also note *C-def*

also from *assms* **have** ($\sum x \in A \cap \text{set-pmf } p. \text{pmf } p$ x) = ($\sum x \in A. \text{pmf } p$ x)

by (*intro sum.mono-neutral-left*) (*auto simp: set-pmf-eq*)

finally have *prob1*: *prob* = ($\sum x \in A. \text{pmf } p$ x) .

hence *prob2*: *prob* = measure-pmf.prob p A

using *assms* **by** (*subst measure-measure-pmf-finite*) *simp-all*

have *prob3*: *prob* = 0 $\longleftrightarrow A \cap \text{set-pmf } p = \{\}$

by (*subst prob1, subst sum-nonneg-eq-0-iff*) (*auto simp: set-pmf-eq assms*)

from *assms* **have** *prob4*: *prob* = measure-pmf.prob p C

unfolding *prob-def* **by** (*intro measure-measure-pmf-finite [symmetric]*) (*simp-all*

add: C-def)

show *?thesis*

proof (*cases prob = 0*)

case *True*

hence $A \cap \text{set-pmf } p = \{\}$ **by** (*subst (asm) prob3*)

with *True* **show** *?thesis* **by** (*simp add: Let-def prob-def C-def cond-pmf-impl.abs-eq*)

next

case *False*

hence $A: C \neq \{\}$ **unfolding** *C-def* **by** (*subst (asm) prob3*) *auto*

with *prob3* **have** *prob-nz*: *prob* $\neq 0$ **by** (*auto simp: C-def*)

fix x

have *cond-pmf-impl* p $A =$

Some (mapping.Mapping ($\lambda x.$ if $x \in C$ then

Some (pmf p x / measure-pmf.prob p C) else None))

(**is** - = Some ?m)

```

    using A prob2 prob4 unfolding C-def by transfer (auto simp: fun-eq-iff)
  also have ?m = Mapping.map-values ( $\lambda y. y / \text{prob}$ )
    (Mapping.filter ( $\lambda k. k \in C$ ) (mapping-of-pmf p))
    using prob-nz prob4 assms unfolding C-def
    by transfer (auto simp: fun-eq-iff set-pmf-eq)
  finally show ?thesis using False by (simp add: Let-def prob-def C-def)
qed
qed

```

```

lemma cond-pmf-impl-code [code]:
  cond-pmf-impl p (set xs) = (
    let C = set xs  $\cap$  set-pmf p;
    prob = ( $\sum_{x \in C} \text{pmf } p \ x$ )
    in if prob = 0 then
      None
    else
      Some (Mapping.map-values ( $\lambda y. y / \text{prob}$ )
        (Mapping.filter ( $\lambda k. k \in C$ ) (mapping-of-pmf p))))
  by (rule cond-pmf-impl-code-alt) simp-all

```

```

lemma cond-pmf-code [code abstract]:
  mapping-of-pmf (cond-pmf p A) =
    (case cond-pmf-impl p A of
      None  $\Rightarrow$  Code.abort (STR "cond-pmf with set of probability 0")
      ( $\lambda. \text{mapping-of-pmf } (\text{cond-pmf } p \ A)$ )
      | Some m  $\Rightarrow$  m)
proof (cases cond-pmf-impl p A)
  case (Some m)
  hence A: set-pmf p  $\cap$  A  $\neq$  {} by transfer (auto split: if-splits)
  from Some have B: Mapping.keys m = set-pmf (cond-pmf p A)
    by (subst set-cond-pmf[OF A], transfer) (auto split: if-splits)
  with Some A have mapping-of-pmf (cond-pmf p A) = m
    by (intro mapping-of-pmfI[OF B], transfer) (auto split: if-splits simp: pmf-cond)
  with Some show ?thesis by simp
qed simp-all

```

```

lemma binomial-pmf-code [code abstract]:
  mapping-of-pmf (binomial-pmf n p) = (
    if p < 0  $\vee$  p > 1 then
      Code.abort (STR "binomial-pmf with invalid probability")
      ( $\lambda. \text{mapping-of-pmf } (\text{binomial-pmf } n \ p)$ )
    else if p = 0 then Mapping.update 0 1 Mapping.empty
    else if p = 1 then Mapping.update n 1 Mapping.empty
    else Mapping.tabulate [0.. $\text{Suc } n$ ] ( $\lambda k. \text{real } (n \text{ choose } k) * p^k * (1 - p)^{n-k}$ ))
  by (cases p < 0  $\vee$  p > 1)
    (simp, intro mapping-of-pmfI,
      auto simp: lookup-update' lookup-empty set-pmf-binomial-eq lookup-tabulate

```

split: if-splits)

lemma *pred-pmf-code* [code]:
 $\text{pred-pmf } P \text{ } p = (\forall x \in \text{set-pmf } p. P \text{ } x)$
by (*auto simp: pred-pmf-def*)

lemma *mapping-of-pmf-pmf-of-list*:
assumes $\bigwedge x. x \in \text{snd } ' \text{set } xs \implies x > 0$ *sum-list* ($\text{map snd } xs$) = 1
shows $\text{mapping-of-pmf } (\text{pmf-of-list } xs) =$
 $\text{Mapping.tabulate } (\text{remdups } (\text{map fst } xs))$
 $(\lambda x. \text{sum-list } (\text{map snd } (\text{filter } (\lambda z. \text{fst } z = x) xs)))$
proof –
from *assms* **have** *wf: pmf-of-list-wf xs* **by** (*intro pmf-of-list-wfI*) *force*
with *assms* **have** $\text{set-pmf } (\text{pmf-of-list } xs) = \text{fst } ' \text{set } xs$
by (*intro set-pmf-of-list-eq*) *auto*
with *wf* **show** *?thesis*
by (*intro mapping-of-pmfI*) (*auto simp: lookup-tabulate pmf-pmf-of-list*)
qed

lemma *mapping-of-pmf-pmf-of-list'*:
assumes *pmf-of-list-wf xs*
defines $xs' \equiv \text{filter } (\lambda z. \text{snd } z \neq 0) \text{ } xs$
shows $\text{mapping-of-pmf } (\text{pmf-of-list } xs) =$
 $\text{Mapping.tabulate } (\text{remdups } (\text{map fst } xs'))$
 $(\lambda x. \text{sum-list } (\text{map snd } (\text{filter } (\lambda z. \text{fst } z = x) xs')))$ (*is - = ?rhs*)
proof –
have *wf: pmf-of-list-wf xs'* **unfolding** *xs'-def* **by** (*rule pmf-of-list-remove-zeros*)
fact
have *pos: $\forall x \in \text{snd } ' \text{set } xs'. x > 0$* **using** *assms(1)* **unfolding** *xs'-def*
by (*force simp: pmf-of-list-wf-def*)
from *assms* **have** $\text{pmf-of-list } xs = \text{pmf-of-list } xs'$
unfolding *xs'-def* **by** (*subst pmf-of-list-remove-zeros*) *simp-all*
also from *wf pos* **have** $\text{mapping-of-pmf } \dots = ?rhs$
by (*intro mapping-of-pmf-pmf-of-list*) (*auto simp: pmf-of-list-wf-def*)
finally show *?thesis* .
qed

lemma *pmf-of-list-wf-code* [code]:
 $\text{pmf-of-list-wf } xs \longleftrightarrow \text{list-all } (\lambda z. \text{snd } z \geq 0) \text{ } xs \wedge \text{sum-list } (\text{map snd } xs) = 1$
by (*auto simp add: pmf-of-list-wf-def list-all-def*)

lemma *pmf-of-list-code* [code abstract]:
 $\text{mapping-of-pmf } (\text{pmf-of-list } xs) =$ (
if *pmf-of-list-wf xs* **then**
 $\text{let } xs' = \text{filter } (\lambda z. \text{snd } z \neq 0) \text{ } xs$
in $\text{Mapping.tabulate } (\text{remdups } (\text{map fst } xs'))$
 $(\lambda x. \text{sum-list } (\text{map snd } (\text{filter } (\lambda z. \text{fst } z = x) xs')))$

```

    else
      Code.abort (STR "Invalid list for pmf-of-list") (λ-. mapping-of-pmf (pmf-of-list
xs)))
  using mapping-of-pmf-pmf-of-list'[of xs] by (simp add: Let-def)

lemma mapping-of-pmf-eq-iff [simp]:
  mapping-of-pmf p = mapping-of-pmf q  $\longleftrightarrow$  p = (q :: 'a pmf)
proof (transfer, intro iffI pmf-eqI)
  fix p q :: 'a pmf and x :: 'a
  assume (λx. if pmf p x = 0 then None else Some (pmf p x)) =
    (λx. if pmf q x = 0 then None else Some (pmf q x))
  hence (if pmf p x = 0 then None else Some (pmf p x)) =
    (if pmf q x = 0 then None else Some (pmf q x)) for x
  by (simp add: fun-eq-iff)
  from this[of x] show pmf p x = pmf q x by (auto split: if-splits)
qed (simp-all cong: if-cong)

```

21.2 Code abbreviations for integrals and probabilities

Integrals and probabilities are defined for general measures, so we cannot give any code equations directly. We can, however, specialise these constants them to PMFs, give code equations for these specialised constants, and tell the code generator to unfold the original constants to the specialised ones whenever possible.

definition *pmf-integral* **where**

pmf-integral p f = *lebesgue-integral* (measure-pmf p) (f :: - \Rightarrow real)

definition *pmf-set-integral* **where**

pmf-set-integral p f A = *lebesgue-integral* (measure-pmf p) (λx. indicator A x * f x :: real)

definition *pmf-prob* **where**

pmf-prob p A = *measure-pmf.prob* p A

lemma *pmf-prob-compl*: *pmf-prob* p (¬A) = 1 - *pmf-prob* p A

using *measure-pmf.prob-compl*[of A p] **by** (simp add: *pmf-prob-def Compl-eq-Diff-UNIV*)

lemma *pmf-integral-pmf-set-integral* [code]:

pmf-integral p f = *pmf-set-integral* p f (set-pmf p)

unfolding *pmf-integral-def pmf-set-integral-def*

by (intro *integral-cong-AE*) (simp-all add: *AE-measure-pmf-iff*)

lemma *pmf-prob-pmf-set-integral*:

pmf-prob p A = *pmf-set-integral* p (λ-. 1) A

by (simp add: *pmf-prob-def pmf-set-integral-def*)

lemma *pmf-set-integral-code-alt-finite*:

finite A \implies *pmf-set-integral* p f A = ($\sum_{x \in A} \text{pmf } p \ x * f \ x$)

unfolding *pmf-set-integral-def*
by (*subst integral-measure-pmf[of A]*) (*auto simp: indicator-def mult-ac split: if-splits*)

lemma *pmf-set-integral-code* [*code*]:
 $\text{pmf-set-integral } p \ f \ (\text{set } xs) = (\sum_{x \in \text{set } xs} \text{pmf } p \ x * f \ x)$
by (*rule pmf-set-integral-code-alt-finite*) *simp-all*

lemma *pmf-prob-code-alt-finite*:
 $\text{finite } A \implies \text{pmf-prob } p \ A = (\sum_{x \in A} \text{pmf } p \ x)$
by (*simp add: pmf-prob-pmf-set-integral pmf-set-integral-code-alt-finite*)

lemma *pmf-prob-code* [*code*]:
 $\text{pmf-prob } p \ (\text{set } xs) = (\sum_{x \in \text{set } xs} \text{pmf } p \ x)$
 $\text{pmf-prob } p \ (\text{List.coset } xs) = 1 - (\sum_{x \in \text{set } xs} \text{pmf } p \ x)$
by (*simp-all add: pmf-prob-code-alt-finite pmf-prob-compl*)

lemma *pmf-prob-code-unfold* [*code-abbrev*]: $\text{pmf-prob } p = \text{measure-pmf.prob } p$
by (*intro ext*) (*simp add: pmf-prob-def*)

lemma *pmf-integral-code-unfold* [*code-abbrev*]: $\text{pmf-integral } p = \text{measure-pmf.expectation } p$
by (*intro ext*) (*simp add: pmf-integral-def*)

definition *pmf-of-alist* $xs = \text{embed-pmf } (\lambda x. \text{case map-of } xs \ x \text{ of Some } p \Rightarrow p \mid \text{None} \Rightarrow 0)$

lemma *pmf-of-mapping-Mapping* [*code-post*]:
 $\text{pmf-of-mapping } (\text{Mapping } xs) = \text{pmf-of-alist } xs$
unfolding *pmf-of-mapping-def Mapping.lookup-default-def [abs-def] pmf-of-alist-def*
by *transfer simp-all*

instantiation *pmf* :: (*equal*) *equal*
begin

definition *equal-pmf* $p \ q = (\text{mapping-of-pmf } p = \text{mapping-of-pmf } (q :: 'a \text{ pmf}))$

instance **by** *standard* (*simp add: equal-pmf-def*)
end

definition *single* :: '*a* \Rightarrow '*a* *multiset* **where**
 $\text{single } s = \{\#s\# \}$

instantiation *pmf* :: (*random*) *random*
begin

context
includes *state-combinator-syntax* **and** *term-syntax*
begin

definition

$$\begin{aligned} \text{pmfify} &:: ('b::\text{typerep multiset} \times (\text{unit} \Rightarrow \text{Code-Evaluation.term})) \Rightarrow \\ &\quad 'b \times (\text{unit} \Rightarrow \text{Code-Evaluation.term}) \Rightarrow \\ &\quad 'b \text{ pmf} \times (\text{unit} \Rightarrow \text{Code-Evaluation.term}) \text{ where} \\ [\text{code-unfold}]: \text{pmfify } A \ x = \\ &\quad \text{Code-Evaluation.valtermify pmf-of-multiset } \{\cdot\} \\ &\quad (\text{Code-Evaluation.valtermify } (+) \ \{\cdot\} \ A \ \{\cdot\}) \\ &\quad (\text{Code-Evaluation.valtermify single } \{\cdot\} \ x) \end{aligned}$$

definition

$$\begin{aligned} \text{Quickcheck-Random.random } i = \\ \text{Quickcheck-Random.random } i \circ \rightarrow (\lambda A. \\ \text{Quickcheck-Random.random } i \circ \rightarrow (\lambda x. \text{Pair } (\text{pmfify } A \ x))) \end{aligned}$$

instance ..

end

end

instantiation *pmf* :: (*full-exhaustive*) *full-exhaustive*
begin

definition *full-exhaustive-pmf* :: (*a pmf* \times (*unit* \Rightarrow *term*)) \Rightarrow (*bool* \times *term list*)
option \Rightarrow *natural* \Rightarrow (*bool* \times *term list*) *option*
where

$$\begin{aligned} \text{full-exhaustive-pmf } f \ i = \\ \text{Quickcheck-Exhaustive.full-exhaustive } (\lambda A. \\ \text{Quickcheck-Exhaustive.full-exhaustive } (\lambda x. f \ (\text{pmfify } A \ x)) \ i) \ i \end{aligned}$$

instance ..

end

end

22 Finite Maps

theory *Fin-Map*
imports *HOL-Analysis.Finite-Product-Measure* *HOL-Library.Finite-Map*
begin

The *fmap* type can be instantiated to *polish-space*, needed for the proof of projective limit. *extensional* functions are used for the representation in order to stay close to the developments of (finite) products Pi_E and their sigma-algebra Pi_M .

type-notation *fmap* ($\langle \langle notation = \langle infix fmap \rangle \rangle - \Rightarrow_F / - \rangle \rangle$ [22, 21] 21)

unbundle *fmap.lifting*

22.1 Domain and Application

lift-definition *domain*::($'i \Rightarrow_F 'a$) \Rightarrow $'i$ set is *dom* .

lemma *finite-domain*[*simp*, *intro*]: *finite* (*domain* *P*)
by *transfer simp*

lift-definition *proj* :: ($'i \Rightarrow_F 'a$) \Rightarrow $'i \Rightarrow 'a$
($\langle \langle indent = 1 \text{ notation} = \langle infix proj \rangle \rangle '(-)'_F \rangle \rangle$ [0] 1000) is
 $\lambda f x$. if $x \in \text{dom } f$ then the $(f x)$ else *undefined* .

declare [[*coercion proj*]]

lemma *extensional-proj*[*simp*, *intro*]: (*P*)_F \in *extensional* (*domain* *P*)
by *transfer (auto simp: extensional-def)*

lemma *proj-undefined*[*simp*, *intro*]: $i \notin \text{domain } P \implies P i = \text{undefined}$
using *extensional-proj*[*of P*] **unfolding** *extensional-def* **by** *auto*

lemma *finmap-eq-iff*: $P = Q \iff (\text{domain } P = \text{domain } Q \wedge (\forall i \in \text{domain } P. P i = Q i))$
apply *transfer*
apply (*safe intro!*: *ext*)
subgoal for *P Q x*
by (*cases* $x \in \text{dom } P$; *cases* $P x$) (*auto dest!*: *bspec*[**where** $x=x$])
done

22.2 Constructor of Finite Maps

lift-definition *finmap-of*:: $'i$ set \Rightarrow ($'i \Rightarrow 'a$) \Rightarrow ($'i \Rightarrow_F 'a$) is
 $\lambda I f x$. if $x \in I \wedge \text{finite } I$ then *Some* $(f x)$ else *None*
by (*simp add: dom-def*)

lemma *proj-finmap-of*[*simp*]:
assumes *finite inds*
shows (*finmap-of* *inds f*)_F = *restrict f inds*
using *assms*
by *transfer force*

lemma *domain-finmap-of*[*simp*]:
assumes *finite inds*

shows $\text{domain } (\text{finmap-of inds } f) = \text{inds}$
using *assms*
by *transfer (auto split: if-splits)*

lemma *finmap-of-eq-iff[simp]*:
assumes *finite i finite j*
shows $\text{finmap-of } i \text{ } m = \text{finmap-of } j \text{ } n \longleftrightarrow i = j \wedge (\forall k \in i. m \text{ } k = n \text{ } k)$
using *assms* **by** *(auto simp: finmap-eq-iff)*

lemma *finmap-of-inj-on-extensional-finite*:
assumes *finite K*
assumes $S \subseteq \text{extensional } K$
shows *inj-on (finmap-of K) S*
proof *(rule inj-onI)*
fix $x \ y :: 'a \Rightarrow 'b$
assume $\text{finmap-of } K \ x = \text{finmap-of } K \ y$
hence $(\text{finmap-of } K \ x)_F = (\text{finmap-of } K \ y)_F$ **by** *simp*
moreover
assume $x \in S \ y \in S$ **hence** $x \in \text{extensional } K \ y \in \text{extensional } K$ **using** *assms*
by *auto*
ultimately
show $x = y$ **using** *assms* **by** *(simp add: extensional-restrict)*
qed

22.3 Product set of Finite Maps

This is Pi for Finite Maps, most of this is copied

definition $Pi' :: 'i \text{ set} \Rightarrow ('i \Rightarrow 'a \text{ set}) \Rightarrow ('i \Rightarrow_F 'a) \text{ set}$ **where**
 $Pi' \ I \ A = \{ P. \text{domain } P = I \wedge (\forall i. i \in I \longrightarrow (P)_F \ i \in A \ i) \}$

syntax

$-Pi' :: [\text{pttrn}, 'a \text{ set}, 'b \text{ set}] \Rightarrow ('a \Rightarrow 'b) \text{ set}$
 $(\langle \langle \text{indent}=3 \text{ notation}=\langle \text{binder } \Pi' \rangle \Pi'' \text{ } -\in- / - \rangle \rangle \quad 10)$

syntax-consts

$-Pi' == Pi'$

translations

$\Pi' \ x \in A. B == \text{CONST } Pi' \ A \ (\lambda x. B)$

22.3.1 Basic Properties of Pi'

lemma $Pi'-I[\text{intro!}]$: $\text{domain } f = A \Longrightarrow (\bigwedge x. x \in A \Longrightarrow f \ x \in B \ x) \Longrightarrow f \in Pi' \ A \ B$

by *(simp add: Pi'-def)*

lemma $Pi'-I[\text{simp}]$: $\text{domain } f = A \Longrightarrow (\bigwedge x. x \in A \longrightarrow f \ x \in B \ x) \Longrightarrow f \in Pi' \ A \ B$

by *(simp add: Pi'-def)*

lemma $Pi'-\text{mem}$: $f \in Pi' \ A \ B \Longrightarrow x \in A \Longrightarrow f \ x \in B \ x$

```

by (simp add: Pi'-def)

lemma Pi'-iff:  $f \in \text{Pi}' I X \longleftrightarrow \text{domain } f = I \wedge (\forall i \in I. f i \in X i)$ 
unfolding Pi'-def by auto

lemma Pi'E [elim]:
 $f \in \text{Pi}' A B \implies (f x \in B x \implies \text{domain } f = A \implies Q) \implies (x \notin A \implies Q) \implies Q$ 
by(auto simp: Pi'-def)

lemma in-Pi'-cong:
 $\text{domain } f = \text{domain } g \implies (\bigwedge w. w \in A \implies f w = g w) \implies f \in \text{Pi}' A B \longleftrightarrow g \in \text{Pi}' A B$ 
by (auto simp: Pi'-def)

lemma Pi'-eq-empty[simp]:
assumes finite A shows  $(\text{Pi}' A B) = \{\} \longleftrightarrow (\exists x \in A. B x = \{\})$ 
using assms
apply (simp add: Pi'-def, auto)
apply (drule-tac  $x = \text{finmap-of } A (\lambda u. \text{SOME } y. y \in B u)$  in spec, auto)
apply (cut-tac  $P = \%y. y \in B i$  in some-eq-ex, auto)
done

lemma Pi'-mono:  $(\bigwedge x. x \in A \implies B x \subseteq C x) \implies \text{Pi}' A B \subseteq \text{Pi}' A C$ 
by (auto simp: Pi'-def)

lemma Pi-Pi':  $\text{finite } A \implies (\text{Pi}_E A B) = \text{proj } \text{'Pi}' A B$ 
apply (auto simp: Pi'-def Pi-def extensional-def)
apply (rule-tac  $x = \text{finmap-of } A (\text{restrict } x A)$  in image-eqI)
apply auto
done

```

22.4 Topological Space of Finite Maps

```

instantiation fmap :: (type, topological-space) topological-space
begin

```

```

definition open-fmap :: ('a  $\Rightarrow_F$  'b) set  $\Rightarrow$  bool where
  [code del]: open-fmap = generate-topology {Pi' a b | a b.  $\forall i \in a. \text{open } (b i)$ }

```

```

lemma open-Pi'I:  $(\bigwedge i. i \in I \implies \text{open } (A i)) \implies \text{open } (\text{Pi}' I A)$ 
by (auto intro: generate-topology.Basis simp: open-fmap-def)

```

```

instance using topological-space-generate-topology
by intro-classes (auto simp: open-fmap-def class.topological-space-def)

```

```

end

```

```

lemma open-restricted-space:

```

```

  shows open {m. P (domain m)}
proof -
  have {m. P (domain m)} = ( $\bigcup i \in \text{Collect } P. \{m. \text{domain } m = i\}$ ) by auto
  also have open ...
proof (rule, safe, cases)
  fix i::'a set
  assume finite i
  hence {m. domain m = i} = Pi' i ( $\lambda-. \text{UNIV}$ ) by (auto simp: Pi'-def)
  also have open ... by (auto intro: open-Pi'I simp: finite i)
  finally show open {m. domain m = i} .
next
  fix i::'a set
  assume  $\neg$  finite i hence {m. domain m = i} = {} by auto
  also have open ... by simp
  finally show open {m. domain m = i} .
qed
finally show ?thesis .
qed

lemma closed-restricted-space:
  shows closed {m. P (domain m)}
  using open-restricted-space[of  $\lambda x. \neg P x$ ]
  unfolding closed-def by (rule back-subst) auto

lemma tendsto-proj:  $((\lambda x. x) \longrightarrow a) F \Longrightarrow ((\lambda x. (x)_F i) \longrightarrow (a)_F i) F$ 
  unfolding tendsto-def
proof safe
  fix S::'b set
  let ?S = Pi' (domain a) ( $\lambda x. \text{if } x = i \text{ then } S \text{ else UNIV}$ )
  assume open S hence open ?S by (auto intro!: open-Pi'I)
  moreover assume  $\forall S. \text{open } S \longrightarrow a \in S \longrightarrow \text{eventually } (\lambda x. x \in S) F \wedge i \in S$ 
  ultimately have eventually  $(\lambda x. x \in ?S) F$  by auto
  thus eventually  $(\lambda x. (x)_F i \in S) F$ 
    by eventually-elim (insert  $\langle a i \in S \rangle$ , force simp: Pi'-iff split: if-split-asm)
qed

lemma continuous-proj:
  shows continuous-on s  $(\lambda x. (x)_F i)$ 
  unfolding continuous-on-def by (safe intro!: tendsto-proj tendsto-ident-at)

instance fmap :: (type, first-countable-topology) first-countable-topology
proof
  fix x::'a $\Rightarrow_F$ 'b
  have  $\forall i. \exists A. \text{countable } A \wedge (\forall a \in A. x i \in a) \wedge (\forall a \in A. \text{open } a) \wedge$ 
     $(\forall S. \text{open } S \wedge x i \in S \longrightarrow (\exists a \in A. a \subseteq S)) \wedge (\forall a b. a \in A \longrightarrow b \in A \longrightarrow a$ 
 $\cap b \in A) \text{ (is } \forall i. ?th i)$ 
  proof
    fix i from first-countable-basis-Int-stableE[of x i]
    obtain A where

```

```

    countable A
     $\bigwedge C. C \in A \implies (x)_F i \in C$ 
     $\bigwedge C. C \in A \implies \text{open } C$ 
     $\bigwedge S. \text{open } S \implies (x)_F i \in S \implies \exists A \in A. A \subseteq S$ 
     $\bigwedge C D. C \in A \implies D \in A \implies C \cap D \in A$ 
    by auto
    thus ?th i by (intro exI[where x=A]) simp
  qed
  then obtain A
    where A:  $\forall i. \text{countable } (A i) \wedge \text{Ball } (A i) ((\in) ((x)_F i)) \wedge \text{Ball } (A i) \text{open} \wedge$ 
       $(\forall S. \text{open } S \wedge (x)_F i \in S \longrightarrow (\exists a \in A i. a \subseteq S)) \wedge (\forall a b. a \in A i \longrightarrow b \in$ 
 $A i \longrightarrow a \cap b \in A i)$ 
    by (auto simp: choice-iff)
    hence open-sub:  $\bigwedge i S. i \in \text{domain } x \implies \text{open } (S i) \implies x i \in (S i) \implies (\exists a \in A i.$ 
 $a \subseteq (S i))$  by auto
    have A-notempty:  $\bigwedge i. i \in \text{domain } x \implies A i \neq \{\}$  using open-sub[of -  $\lambda i. \text{UNIV}$ ]
    by auto
    let ?A =  $(\lambda f. \text{Pi}' (\text{domain } x) f) ' (\text{Pi}_E (\text{domain } x) A)$ 
    show  $\exists A :: \text{nat} \Rightarrow ('a \Rightarrow_F 'b) \text{ set}. (\forall i. x \in (A i) \wedge \text{open } (A i)) \wedge (\forall S. \text{open } S \wedge$ 
 $x \in S \longrightarrow (\exists i. A i \subseteq S))$ 
    proof (rule first-countableI[of ?A], safe)
      show countable ?A using A by (simp add: countable-PiE)
    next
      fix S:: $('a \Rightarrow_F 'b) \text{ set}$  assume open S  $x \in S$ 
      thus  $\exists a \in ?A. a \subseteq S$  unfolding open-fmap-def
      proof (induct rule: generate-topology.induct)
        case UNIV thus ?case by (auto simp add: ex-in-conv PiE-eq-empty-iff
        A-notempty)
      next
        case (Int a b)
        then obtain f g where
           $f \in \text{Pi}_E (\text{domain } x) A \text{ Pi}' (\text{domain } x) f \subseteq a \text{ } g \in \text{Pi}_E (\text{domain } x) A \text{ Pi}'$ 
 $(\text{domain } x) g \subseteq b$ 
        by auto
        thus ?case using A
        by (auto simp: Pi'-iff PiE-iff extensional-def Int-stable-def
        intro!: bexI[where x= $\lambda i. f i \cap g i$ ])
      next
        case (UN B)
        then obtain b where  $x \in b \text{ } b \in B$  by auto
        hence  $\exists a \in ?A. a \subseteq b$  using UN by simp
        thus ?case using  $\langle b \in B \rangle$  by (metis Sup-upper2)
      next
        case (Basis s)
        then obtain a b where  $xs: x \in \text{Pi}' a b \text{ } s = \text{Pi}' a b \wedge i. i \in a \implies \text{open } (b i)$ 
        by auto
        have  $\forall i. \exists a. (i \in \text{domain } x \wedge \text{open } (b i) \wedge (x)_F i \in b i) \longrightarrow (a \in A i \wedge a \subseteq$ 
 $b i)$ 
        using open-sub[of - b] by auto

```

```

then obtain  $b'$ 
  where  $\bigwedge i. i \in \text{domain } x \implies \text{open } (b \ i) \implies (x)_F \ i \in b \ i \implies (b' \ i \in A \ i \wedge$ 
 $b' \ i \subseteq b \ i)$ 
  unfolding choice-iff by auto
  with  $xs$  have  $\bigwedge i. i \in a \implies (b' \ i \in A \ i \wedge b' \ i \subseteq b \ i) \ Pi' \ a \ b' \subseteq \Pi' \ a \ b$ 
  by (auto simp: Pi'-iff intro!: Pi'-mono)
  thus ?case using  $xs$ 
  by (intro bexI[where  $x = \Pi' \ a \ b'$ ])
  (auto simp: Pi'-iff intro!: image-eqI[where  $x = \text{restrict } b' \ (\text{domain } x)$ ])
qed
qed (insert A, auto simp: PiE-iff intro!: open-Pi'I)
qed

```

22.5 Metric Space of Finite Maps

```

instantiation fmap :: (type, metric-space) dist
begin

```

```

definition dist-fmap where

```

```

  dist P Q = Max (range ( $\lambda i. \text{dist } ((P)_F \ i) ((Q)_F \ i)$ )) + (if  $\text{domain } P = \text{domain } Q$  then 0 else 1)

```

```

instance ..
end

```

```

instantiation fmap :: (type, metric-space) uniformity-dist
begin

```

```

definition [code del]:

```

```

  (uniformity :: ( $(\ 'a, \ 'b) \text{fmap} \times (\ 'a \Rightarrow_F \ 'b)$ ) filter) =
  (INF  $e \in \{0 \dots\}. \text{principal } \{(x, y). \text{dist } x \ y < e\}$ )

```

```

instance
  by standard (rule uniformity-fmap-def)
end

```

```

declare uniformity-Absort[where  $\ 'a = (\ 'a \Rightarrow_F \ 'b :: \text{metric-space})$ , code]

```

```

instantiation fmap :: (type, metric-space) metric-space
begin

```

```

lemma finite-proj-image':  $x \notin \text{domain } P \implies \text{finite } ((P)_F \ 'S)$ 
  by (rule finite-subset[of - proj P ' (domain P  $\cap$  S  $\cup$  {x})]) auto

```

```

lemma finite-proj-image: finite  $((P)_F \ 'S)$ 
  by (cases  $\exists x. x \notin \text{domain } P$ ) (auto intro: finite-proj-image' finite-subset[where
 $B = \text{domain } P]$ )

```

```

lemma finite-proj-diag: finite  $((\lambda i. d \ ((P)_F \ i) ((Q)_F \ i)) \ 'S)$ 

```

proof –
 have $(\lambda i. d ((P)_F i) ((Q)_F i)) \text{ ‘ } S = (\lambda(i, j). d i j) \text{ ‘ } ((\lambda i. ((P)_F i, (Q)_F i)) \text{ ‘ } S)$ **by** *auto*
 moreover have $((\lambda i. ((P)_F i, (Q)_F i)) \text{ ‘ } S) \subseteq (\lambda i. (P)_F i) \text{ ‘ } S \times (\lambda i. (Q)_F i) \text{ ‘ } S$ **by** *auto*
 moreover have *finite* ... **using** *finite-proj-image[of P S] finite-proj-image[of Q S]*
by (*intro finite-cartesian-product simp-all*)
 ultimately show *?thesis* **by** (*simp add: finite-subset*)
qed

lemma *dist-le-1-imp-domain-eq*:
 shows $\text{dist } P \ Q < 1 \implies \text{domain } P = \text{domain } Q$
by (*simp add: dist-fmap-def finite-proj-diag split: if-split-asm*)

lemma *dist-proj*:
 shows $\text{dist } ((x)_F i) ((y)_F i) \leq \text{dist } x \ y$
proof –
 have $\text{dist } (x \ i) (y \ i) \leq \text{Max } (\text{range } (\lambda i. \text{dist } (x \ i) (y \ i)))$
by (*simp add: Max-ge-iff finite-proj-diag*)
 also have $\dots \leq \text{dist } x \ y$ **by** (*simp add: dist-fmap-def*)
 finally show *?thesis* .
qed

lemma *dist-finmap-lessI*:
 assumes $\text{domain } P = \text{domain } Q$
 assumes $0 < e$
 assumes $\bigwedge i. i \in \text{domain } P \implies \text{dist } (P \ i) (Q \ i) < e$
 shows $\text{dist } P \ Q < e$
proof –
 have $\text{dist } P \ Q = \text{Max } (\text{range } (\lambda i. \text{dist } (P \ i) (Q \ i)))$
using *assms* **by** (*simp add: dist-fmap-def finite-proj-diag*)
 also have $\dots < e$
proof (*subst Max-less-iff, safe*)
 fix i
 show $\text{dist } ((P)_F i) ((Q)_F i) < e$ **using** *assms*
by (*cases i ∈ domain P simp-all*)
qed (*simp add: finite-proj-diag*)
 finally show *?thesis* .
qed

instance
proof
 fix $S :: ('a \Rightarrow_F 'b) \text{ set}$
 have $*$: $\text{open } S = (\forall x \in S. \exists e > 0. \forall y. \text{dist } y \ x < e \longrightarrow y \in S)$ (**is** - = *?od*)
proof
 assume *open S*
 thus *?od*
unfolding *open-fmap-def*

```

proof (induct rule: generate-topology.induct)
  case UNIV thus ?case by (auto intro: zero-less-one)
next
  case (Int a b)
  show ?case
  proof safe
    fix x assume x:  $x \in a \ x \in b$ 
    with Int obtain e1 e2 where
       $e1 > 0 \ \forall y. \text{dist } y \ x < e1 \longrightarrow y \in a \ e2 > 0 \ \forall y. \text{dist } y \ x < e2 \longrightarrow y \in b$  by
force
    thus  $\exists e > 0. \ \forall y. \text{dist } y \ x < e \longrightarrow y \in a \cap b$ 
    by (auto intro!: exI[where  $x = \min \ e1 \ e2$ ])
  qed
next
  case (UN K)
  show ?case
  proof safe
    fix x X assume  $x \in X$  and X:  $X \in K$ 
    with UN obtain e where  $e > 0 \ \bigwedge y. \text{dist } y \ x < e \longrightarrow y \in X$  by force
    with X show  $\exists e > 0. \ \forall y. \text{dist } y \ x < e \longrightarrow y \in \bigcup K$  by auto
  qed
next
  case (Basis s) then obtain a b where  $s = \text{Pi}' \ a \ b$  and  $b: \bigwedge i. i \in a \implies$ 
open (b i) by auto
  show ?case
  proof safe
    fix x assume  $x \in s$ 
    hence [simp]: finite a and a-dom:  $a = \text{domain } x$  using s by (auto simp:
Pi'-iff)
    obtain es where  $es: \forall i \in a. \ es \ i > 0 \wedge (\forall y. \text{dist } y \ (\text{proj } x \ i) < es \ i \longrightarrow y$ 
 $\in b \ i)$ 
    using  $b \ \langle x \in s \rangle$  by atomize-elim (intro bchoice, auto simp: open-dist s)
    hence in-b:  $\bigwedge i \ y. \ i \in a \implies \text{dist } y \ (\text{proj } x \ i) < es \ i \implies y \in b \ i$  by auto
    show  $\exists e > 0. \ \forall y. \text{dist } y \ x < e \longrightarrow y \in s$ 
    proof (cases, rule, safe)
      assume  $a \neq \{\}$ 
      show  $0 < \min 1 \ (\text{Min } (es \ 'a))$  using es by (auto simp: \langle a \neq \{\} \rangle)
      fix y assume d:  $\text{dist } y \ x < \min 1 \ (\text{Min } (es \ 'a))$ 
      show  $y \in s$  unfolding s
    proof
      show  $\text{domain } y = a$  using  $d \ s \ \langle a \neq \{\} \rangle$  by (auto simp: dist-le-1-imp-domain-eq
a-dom)
      fix i assume  $i: i \in a$ 
      hence  $\text{dist } ((y)_F \ i) \ ((x)_F \ i) < es \ i$  using d
      by (auto simp: dist-fmap-def \langle a \neq \{\} \rangle intro! le-less-trans[OF dist-proj])
      with i show  $y \ i \in b \ i$  by (rule in-b)
    qed
  next
  assume  $\neg a \neq \{\}$ 

```

```

      thus  $\exists e > 0. \forall y. \text{dist } y \ x < e \longrightarrow y \in s$ 
      using  $s \langle x \in s \rangle$  by (auto simp:  $Pi'$ -def  $\text{dist-le-1-imp-domain-eq}$  intro!)
    exI[where  $x=1$ ])
  qed
  qed
  qed
next
  assume  $\forall x \in S. \exists e > 0. \forall y. \text{dist } y \ x < e \longrightarrow y \in S$ 
  then obtain  $e$  where  $e\text{-pos}: \bigwedge x. x \in S \Longrightarrow e \ x > 0$  and
     $e\text{-in}: \bigwedge x \ y. x \in S \Longrightarrow \text{dist } y \ x < e \ x \Longrightarrow y \in S$ 
  unfolding  $b\text{choice-iff}$ 
  by auto
  have  $S\text{-eq}: S = \bigcup \{Pi' \ a \ b \mid a \ b. \exists x \in S. \text{domain } x = a \wedge b = (\lambda i. \text{ball } (x \ i) (e \ x))\}$ 
  proof safe
    fix  $x$  assume  $x \in S$ 
    thus  $x \in \bigcup \{Pi' \ a \ b \mid a \ b. \exists x \in S. \text{domain } x = a \wedge b = (\lambda i. \text{ball } (x \ i) (e \ x))\}$ 
      using  $e\text{-pos}$  by (auto intro!: exI[where  $x=Pi' \ (\text{domain } x) \ (\lambda i. \text{ball } (x \ i) (e \ x))$ ])
  next
    fix  $x \ y$ 
    assume  $y \in S$ 
    moreover
      assume  $x \in (\Pi' \ i \in \text{domain } y. \text{ball } (y \ i) (e \ y))$ 
    hence  $\text{dist } x \ y < e \ y$  using  $e\text{-pos} \langle y \in S \rangle$ 
      by (auto simp:  $\text{dist-fmap-def } Pi'\text{-iff finite-proj-diag dist-commute}$ )
    ultimately show  $x \in S$  by (rule  $e\text{-in}$ )
  qed
  also have open ...
    unfolding  $\text{open-fmap-def}$ 
    by (intro  $\text{generate-topology.UN}$ ) (auto intro:  $\text{generate-topology.Basis}$ )
  finally show open  $S$  .
qed
show open  $S = (\forall x \in S. \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in S)$ 
  unfolding *  $\text{eventually-uniformity-metric}$ 
  by (simp del:  $\text{split-paired-All}$  add:  $\text{dist-fmap-def dist-commute eq-commute}$ )
next
  fix  $P \ Q::'a \Rightarrow_F 'b$ 
  have  $\text{Max-eq-iff}: \bigwedge A \ m. \text{finite } A \Longrightarrow A \neq \{\} \Longrightarrow (\text{Max } A = m) = (m \in A \wedge (\forall a \in A. a \leq m))$ 
    by (auto intro:  $\text{Max-in Max-eqI}$ )
  show  $\text{dist } P \ Q = 0 \longleftrightarrow P = Q$ 
    by (auto simp:  $\text{finmap-eq-iff dist-fmap-def Max-ge-iff finite-proj-diag Max-eq-iff}$ 
       $\text{add-nonneg-eq-0-iff}$ 
      intro!:  $\text{Max-eqI image-eqI}$ [where  $x=\text{undefined}$ ])
next
  fix  $P \ Q \ R::'a \Rightarrow_F 'b$ 
  let  $?dists = \lambda P \ Q \ i. \text{dist } ((P)_F \ i) ((Q)_F \ i)$ 
  let  $?dpr = ?dists \ P \ Q$  and  $?dqr = ?dists \ Q \ R$ 

```



```

let ?dom =  $\lambda P Q$ . (if domain  $P = \text{domain } Q$  then 0 else 1::real)
have dist  $P Q = \text{Max } (\text{range } ?dpq) + ?dom P Q$ 
  by (simp add: dist-fmap-def)
also obtain  $t$  where  $t \in \text{range } ?dpq$   $t = \text{Max } (\text{range } ?dpq)$  by (simp add:
finite-proj-diag)
then obtain  $i$  where  $\text{Max } (\text{range } ?dpq) = ?dpq i$  by auto
also have  $?dpq i \leq ?dpr i + ?dqr i$  by (rule dist-triangle2)
also have  $?dpr i \leq \text{Max } (\text{range } ?dpr)$  by (simp add: finite-proj-diag)
also have  $?dqr i \leq \text{Max } (\text{range } ?dqr)$  by (simp add: finite-proj-diag)
also have  $?dom P Q \leq ?dom P R + ?dom Q R$  by simp
finally show dist  $P Q \leq \text{dist } P R + \text{dist } Q R$  by (simp add: dist-fmap-def
ac-simps)
qed

end

```

22.6 Complete Space of Finite Maps

lemma tendsto-finmap:

```

fixes  $f::\text{nat} \Rightarrow ('i \Rightarrow_F ('a::\text{metric-space}))$ 
assumes ind-f:  $\bigwedge n. \text{domain } (f n) = \text{domain } g$ 
assumes proj-g:  $\bigwedge i. i \in \text{domain } g \implies (\lambda n. (f n) i) \longrightarrow g i$ 
shows  $f \longrightarrow g$ 
unfolding tendsto-iff
proof safe
fix  $e::\text{real}$  assume  $0 < e$ 
let ?dists =  $\lambda x i. \text{dist } ((f x)_F i) ((g)_F i)$ 
have eventually  $(\lambda x. \forall i \in \text{domain } g. ?dists x i < e)$  sequentially
  using finite-domain[of  $g$ ] proj-g
proof induct
case (insert  $i G$ )
  with  $\langle 0 < e \rangle$  have eventually  $(\lambda x. ?dists x i < e)$  sequentially by (auto simp
add: tendsto-iff)
  moreover
  from insert have eventually  $(\lambda x. \forall i \in G. \text{dist } ((f x)_F i) ((g)_F i) < e)$  sequentially
by simp
  ultimately show ?case by eventually-elim auto
qed simp
thus eventually  $(\lambda x. \text{dist } (f x) g < e)$  sequentially
  by eventually-elim (auto simp add: dist-fmap-def finite-proj-diag ind-f  $\langle 0 < e \rangle$ )
qed

```

instance fmap :: (type, complete-space) complete-space

proof

```

fix  $P::\text{nat} \Rightarrow 'a \Rightarrow_F 'b$ 
assume Cauchy  $P$ 
then obtain  $Nd$  where  $Nd: \bigwedge n. n \geq Nd \implies \text{dist } (P n) (P Nd) < 1$ 
  by (force simp: Cauchy-altdef2)
define  $d$  where  $d = \text{domain } (P Nd)$ 

```

```

with Nd have dim:  $\bigwedge n. n \geq Nd \implies \text{domain } (P\ n) = d$  using dist-le-1-imp-domain-eq
by auto
  have [simp]: finite d unfolding d-def by simp
  define p where p i n = P n i for i n
  define q where q i = lim (p i) for i
  define Q where Q = finmap-of d q
  have q:  $\bigwedge i. i \in d \implies q\ i = Q\ i$  by (auto simp add: Q-def Abs-fmap-inverse)
  {
    fix i assume i  $\in d$ 
    have Cauchy (p i) unfolding Cauchy-altdef2 p-def
    proof safe
      fix e::real assume 0 < e
      with  $\langle \text{Cauchy } P \rangle$  obtain N where N:  $\bigwedge n. n \geq N \implies \text{dist } (P\ n) (P\ N) < \min\ e\ 1$ 
      by (force simp: Cauchy-altdef2 min-def)
      hence  $\bigwedge n. n \geq N \implies \text{domain } (P\ n) = \text{domain } (P\ N)$  using dist-le-1-imp-domain-eq
    by auto
    with dim have dim:  $\bigwedge n. n \geq N \implies \text{domain } (P\ n) = d$  by (metis nat-le-linear)
    show  $\exists N. \forall n \geq N. \text{dist } ((P\ n)\ i) ((P\ N)\ i) < e$ 
    proof (safe intro!: exI[where x=N])
      fix n assume N  $\leq n$  have N  $\leq N$  by simp
      have  $\text{dist } ((P\ n)\ i) ((P\ N)\ i) \leq \text{dist } (P\ n) (P\ N)$ 
      using dim[OF  $\langle N \leq n \rangle$ ] dim[OF  $\langle N \leq N \rangle$ ]  $\langle i \in d \rangle$ 
      by (auto intro!: dist-proj)
      also have ... < e using N[OF  $\langle N \leq n \rangle$ ] by simp
      finally show  $\text{dist } ((P\ n)\ i) ((P\ N)\ i) < e$  .
    qed
  }
  qed
  hence convergent (p i) by (metis Cauchy-convergent-iff)
  hence p i  $\longrightarrow$  q i unfolding q-def convergent-def by (metis limI)
} note p = this
have P  $\longrightarrow$  Q
proof (rule metric-LIMSEQ-I)
  fix e::real assume 0 < e
  have  $\exists ni. \forall i \in d. \forall n \geq ni\ i. \text{dist } (p\ i\ n) (q\ i) < e$ 
  proof (safe intro!: bchoice)
    fix i assume i  $\in d$ 
    from p[OF  $\langle i \in d \rangle$ , THEN metric-LIMSEQ-D, OF  $\langle 0 < e \rangle$ ]
    show  $\exists no. \forall n \geq no. \text{dist } (p\ i\ n) (q\ i) < e$  .
  qed
  then obtain ni where ni:  $\forall i \in d. \forall n \geq ni\ i. \text{dist } (p\ i\ n) (q\ i) < e$  ..
  define N where N = max Nd (Max (ni ‘ d))
  show  $\exists N. \forall n \geq N. \text{dist } (P\ n) Q < e$ 
  proof (safe intro!: exI[where x=N])
    fix n assume N  $\leq n$ 
    hence dom:  $\text{domain } (P\ n) = d$   $\text{domain } Q = d$   $\text{domain } (P\ n) = \text{domain } Q$ 
    using dim by (simp-all add: N-def Q-def dim-def Abs-fmap-inverse)
    show  $\text{dist } (P\ n) Q < e$ 
    proof (rule dist-finmap-lessI[OF dom(3)  $\langle 0 < e \rangle$ ])

```

```

    fix i
    assume i ∈ domain (P n)
    hence ni i ≤ Max (ni ‘ d) using dom by simp
    also have ... ≤ N by (simp add: N-def)
    finally show dist ((P n)F i) ((Q)F i) < e using ni ⟨i ∈ domain (P n)⟩ ⟨N
≤ n⟩ dom
    by (auto simp: p-def q N-def less-imp-le)
  qed
qed
qed
thus convergent P by (auto simp: convergent-def)
qed

```

22.7 Second Countable Space of Finite Maps

instantiation fmap :: (countable, second-countable-topology) second-countable-topology
begin

definition basis-proj::'b set set
 where basis-proj = (SOME B. countable B ∧ topological-basis B)

lemma countable-basis-proj: countable basis-proj **and** basis-proj: topological-basis
 basis-proj
 unfolding basis-proj-def by (intro is-basis countable-basis)+

definition basis-finmap::('a ⇒_F 'b) set set
 where basis-finmap = {Pi' I S | I S. finite I ∧ (∀ i ∈ I. S i ∈ basis-proj)}

lemma in-basis-finmapI:
 assumes finite I assumes $\bigwedge i. i \in I \implies S i \in \text{basis-proj}$
 shows Pi' I S ∈ basis-finmap
 using assms unfolding basis-finmap-def by auto

lemma basis-finmap-eq:
 assumes basis-proj ≠ {}
 shows basis-finmap = (λf. Pi' (domain f) (λi. from-nat-into basis-proj ((f)_F i))) ‘
 (UNIV::('a ⇒_F nat) set) (is - = ?f ‘ -)
 unfolding basis-finmap-def

proof safe
 fix I::'a set **and** S::'a ⇒ 'b set
 assume finite I ∀ i ∈ I. S i ∈ basis-proj
 hence Pi' I S = ?f (finmap-of I (λx. to-nat-on basis-proj (S x)))
 by (force simp: Pi'-def countable-basis-proj)
 thus Pi' I S ∈ range ?f by simp

next
 fix x **and** f::'a ⇒_F nat
 show ∃ I S. (Π' i ∈ domain f. from-nat-into basis-proj ((f)_F i)) = Pi' I S ∧
 finite I ∧ (∀ i ∈ I. S i ∈ basis-proj)

using *assms* **by** (*auto intro: from-nat-into*)
qed

lemma *basis-finmap-eq-empty*: $\text{basis-proj} = \{\} \implies \text{basis-finmap} = \{Pi' \{\} \text{ undefined}\}$
by (*auto simp: Pi'-iff basis-finmap-def*)

lemma *countable-basis-finmap*: *countable basis-finmap*
by (*cases basis-proj = \{\}*) (*auto simp: basis-finmap-eq basis-finmap-eq-empty*)

lemma *finmap-topological-basis*:
topological-basis basis-finmap

proof (*subst topological-basis-iff, safe*)

fix *B'* **assume** $B' \in \text{basis-finmap}$

thus *open B'*

by (*auto intro!: open-Pi'I topological-basis-open[OF basis-proj]*)

simp: topological-basis-def basis-finmap-def Let-def)

next

fix $O'::('a \Rightarrow_F 'b)$ *set* **and** *x*

assume $O': \text{open } O' \ x \in O'$

then obtain *a* **where** *a*:

$x \in Pi'(\text{domain } x) \ a \ Pi'(\text{domain } x) \ a \subseteq O' \wedge i. i \in \text{domain } x \implies \text{open } (a \ i)$

unfolding *open-fmap-def*

proof (*atomize-elim, induct rule: generate-topology.induct*)

case (*Int a b*)

let $?p = \lambda a \ f. x \in Pi'(\text{domain } x) \ f \wedge Pi'(\text{domain } x) \ f \subseteq a \wedge (\forall i. i \in \text{domain } x \longrightarrow \text{open } (f \ i))$

x $\longrightarrow \text{open } (f \ i))$

from *Int* **obtain** *f g* **where** $?p \ a \ f \ ?p \ b \ g$ **by** *auto*

thus $?case$ **by** (*force intro!: exI[where $x = \lambda i. f \ i \cap g \ i$] simp: Pi'-def*)

next

case (*UN k*)

then obtain *kk a* **where** $x \in kk \ kk \in k \ x \in Pi'(\text{domain } x) \ a \ Pi'(\text{domain } x)$

$a \subseteq kk$

$\wedge i. i \in \text{domain } x \implies \text{open } (a \ i)$

by *force*

thus $?case$ **by** *blast*

qed (*auto simp: Pi'-def*)

have $\exists B.$

$(\forall i \in \text{domain } x. x \ i \in B \ i \wedge B \ i \subseteq a \ i \wedge B \ i \in \text{basis-proj})$

proof (*rule bchoice, safe*)

fix *i* **assume** $i \in \text{domain } x$

hence $\text{open } (a \ i) \ x \ i \in a \ i$ **using** *a* **by** *auto*

from *topological-basisE[OF basis-proj this]* **obtain** *b'*

where $b' \in \text{basis-proj } (x)_F \ i \in b' \ b' \subseteq a \ i$

by *blast*

thus $\exists y. x \ i \in y \wedge y \subseteq a \ i \wedge y \in \text{basis-proj}$ **by** *auto*

qed

then obtain *B* **where** $B: \forall i \in \text{domain } x. (x)_F \ i \in B \ i \wedge B \ i \subseteq a \ i \wedge B \ i \in \text{basis-proj}$

```

  by auto
  define B' where B' = Pi' (domain x) (λi. (B i)::'b set)
  have B' ⊆ Pi' (domain x) a using B by (auto intro!: Pi'-mono simp: B'-def)
  also note ⟨... ⊆ O'⟩
  finally show ∃ B'∈basis-finmap. x ∈ B' ∧ B' ⊆ O' using B
    by (auto intro!: bexI[where x=B'] Pi'-mono in-basis-finmapI simp: B'-def)
qed

```

```

lemma range-enum-basis-finmap-imp-open:
  assumes x ∈ basis-finmap
  shows open x
  using finmap-topological-basis assms by (auto simp: topological-basis-def)

```

```

instance proof qed (blast intro: finmap-topological-basis countable-basis-finmap
topological-basis-imp-subbasis)

```

end

22.8 Polish Space of Finite Maps

```

instance fmap :: (countable, polish-space) polish-space proof qed

```

22.9 Product Measurable Space of Finite Maps

```

definition PiF I M ≡
  sigma (⋃ J ∈ I. (Π' j∈J. space (M j))) { (Π' j∈J. X j) | X J. J ∈ I ∧ X ∈ (Π
j∈J. sets (M j)) }

```

abbreviation

$Pi_F I M \equiv PiF I M$

syntax

-PiF :: pttrn ⇒ 'i set ⇒ 'a measure ⇒ ('i => 'a) measure
 (⟨⟨indent=3 notation=⟨binder Π_F⟩Π_F -∈-./ -⟩ 10⟩

syntax-consts

-PiF == PiF

translations

$\Pi_F x \in I. M == CONST PiF I (\%x. M)$

```

lemma PiF-gen-subset: { (Π' j∈J. X j) | X J. J ∈ I ∧ X ∈ (Π j∈J. sets (M j)) }
⊆

```

$Pow (\bigcup J \in I. (\Pi' j \in J. space (M j)))$
 by (auto simp: Pi'-def) (blast dest: sets.sets-into-space)

```

lemma space-PiF: space (PiF I M) = (⋃ J ∈ I. (Π' j∈J. space (M j)))
  unfolding PiF-def using PiF-gen-subset by (rule space-measure-of)

```

lemma sets-PiF:

$sets (PiF I M) = sigma-sets (\bigcup J \in I. (\Pi' j \in J. space (M j)))$
 $\{ (\Pi' j \in J. X j) | X J. J \in I \wedge X \in (\Pi j \in J. sets (M j)) \}$

unfolding *PiF-def* **using** *PiF-gen-subset* **by** (*rule sets-measure-of*)

lemma *sets-PiF-singleton*:

sets (*PiF* {*I*} *M*) = *sigma-sets* ($\Pi' j \in I. \text{space } (M j)$)
 $\{(\Pi' j \in I. X j) \mid X. X \in (\Pi j \in I. \text{sets } (M j))\}$

unfolding *sets-PiF* **by** *simp*

lemma *in-sets-PiFI*:

assumes $X = (Pi' J S) \ J \in I \ \bigwedge i. i \in J \implies S i \in \text{sets } (M i)$

shows $X \in \text{sets } (PiF I M)$

unfolding *sets-PiF*

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

lemma *product-in-sets-PiFI*:

assumes $J \in I \ \bigwedge i. i \in J \implies S i \in \text{sets } (M i)$

shows $(Pi' J S) \in \text{sets } (PiF I M)$

unfolding *sets-PiF*

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

lemma *singleton-space-subset-in-sets*:

fixes *J*

assumes $J \in I$

assumes *finite J*

shows $\text{space } (PiF \{J\} M) \in \text{sets } (PiF I M)$

using *assms*

by (*intro in-sets-PiFI*[**where** $J=J$ **and** $S=\lambda i. \text{space } (M i)$])
 (*auto simp: product-def space-PiF*)

lemma *singleton-subspace-set-in-sets*:

assumes *A*: $A \in \text{sets } (PiF \{J\} M)$

assumes *finite J*

assumes $J \in I$

shows $A \in \text{sets } (PiF I M)$

using *A*[*unfolded sets-PiF*]

apply (*induct A*)

unfolding *sets-PiF*[*symmetric*] **unfolding** *space-PiF*[*symmetric*]

using *assms*

by (*auto intro: in-sets-PiFI intro!: singleton-space-subset-in-sets*)

lemma *finite-measurable-singletonI*:

assumes *finite I*

assumes $\bigwedge J. J \in I \implies \text{finite } J$

assumes *MN*: $\bigwedge J. J \in I \implies A \in \text{measurable } (PiF \{J\} M) \ N$

shows $A \in \text{measurable } (PiF I M) \ N$

unfolding *measurable-def*

proof *safe*

fix *y* **assume** $y \in \text{sets } N$

have $A -' y \cap \text{space } (PiF I M) = (\bigcup J \in I. A -' y \cap \text{space } (PiF \{J\} M))$

by (*auto simp: space-PiF*)

```

also have ...  $\in$  sets (PiF I M)
proof (rule sets.finite-UN)
  show finite I by fact
  fix J assume  $J \in I$ 
  with assms have finite J by simp
  show  $A - 'y \cap \text{space } (PiF \{J\} M) \in \text{sets } (PiF I M)$ 
    by (rule singleton-subspace-set-in-sets[OF measurable-sets[OF assms( $\beta$ )]])
fact+
qed
finally show  $A - 'y \cap \text{space } (PiF I M) \in \text{sets } (PiF I M)$  .
next
  fix x assume  $x \in \text{space } (PiF I M)$  thus  $A x \in \text{space } N$ 
    using MN[of domain x]
    by (auto simp: space-PiF measurable-space Pi'-def)
qed

```

lemma *countable-finite-comprehension:*

```

fixes f :: 'a::countable set  $\Rightarrow$  -
assumes  $\bigwedge s. P s \implies \text{finite } s$ 
assumes  $\bigwedge s. P s \implies f s \in \text{sets } M$ 
shows  $\bigcup \{f s \mid s. P s\} \in \text{sets } M$ 
proof -
  have  $\bigcup \{f s \mid s. P s\} = (\bigcup n::\text{nat. let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$ 
proof safe
  fix x X s assume *:  $x \in f s P s$ 
  with assms obtain l where  $s = \text{set } l$  using finite-list by blast
  with * show  $x \in (\bigcup n. \text{let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$  using
     $\langle P s \rangle$ 
    by (auto intro!: exI[where  $x = \text{to-nat } l$ ])
  next
  fix x n assume  $x \in (\text{let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$ 
  thus  $x \in \bigcup \{f s \mid s. P s\}$  using assms by (auto simp: Let-def split: if-split-asm)
qed
hence  $\bigcup \{f s \mid s. P s\} = (\bigcup n. \text{let } s = \text{set } (\text{from-nat } n) \text{ in if } P s \text{ then } f s \text{ else } \{\})$ 
by simp
also have ...  $\in \text{sets } M$  using assms by (auto simp: Let-def)
finally show ?thesis .
qed

```

lemma *space-subset-in-sets:*

```

fixes J :: 'a::countable set set
assumes  $J \subseteq I$ 
assumes  $\bigwedge j. j \in J \implies \text{finite } j$ 
shows  $\text{space } (PiF J M) \in \text{sets } (PiF I M)$ 
proof -
  have  $\text{space } (PiF J M) = \bigcup \{\text{space } (PiF \{j\} M) \mid j. j \in J\}$ 
  unfolding space-PiF by blast
  also have ...  $\in \text{sets } (PiF I M)$  using assms

```

by (intro countable-finite-comprehension) (auto simp: singleton-space-subset-in-sets)
 finally show ?thesis .
 qed

lemma *subspace-set-in-sets*:
 fixes $J::'a::\text{countable set set}$
 assumes $A: A \in \text{sets } (\text{PiF } J \ M)$
 assumes $J \subseteq I$
 assumes $\bigwedge j. j \in J \implies \text{finite } j$
 shows $A \in \text{sets } (\text{PiF } I \ M)$
 using $A[\text{unfolded sets-PiF}]$
 apply (induct A)
 unfolding $\text{sets-PiF}[\text{symmetric}]$ $\text{space-PiF}[\text{symmetric}]$
 using *assms*
 by (auto intro: in-sets-PiFI intro!: space-subset-in-sets)

lemma *countable-measurable-PiFI*:
 fixes $I::'a::\text{countable set set}$
 assumes $MN: \bigwedge J. J \in I \implies \text{finite } J \implies A \in \text{measurable } (\text{PiF } \{J\} \ M) \ N$
 shows $A \in \text{measurable } (\text{PiF } I \ M) \ N$
 unfolding *measurable-def*
proof *safe*
 fix y assume $y \in \text{sets } N$
 have $A -' y = (\bigcup \{A -' y \cap \{x. \text{domain } x = J\} \mid J. \text{finite } J\})$ **by** *auto*
 { fix $x::'a \Rightarrow_F 'b$
 from *finite-list[of domain x]* **obtain** xs **where** $\text{set } xs = \text{domain } x$ **by** *auto*
 hence $\exists n. \text{domain } x = \text{set } (\text{from-nat } n)$
by (intro *exI[where x=to-nat xs]*) *auto* }
 hence $A -' y \cap \text{space } (\text{PiF } I \ M) = (\bigcup n. A -' y \cap \text{space } (\text{PiF } (\{\text{set } (\text{from-nat } n)\} \cap I) \ M))$
by (auto simp: *space-PiF Pi'-def*)
 also have $\dots \in \text{sets } (\text{PiF } I \ M)$
 apply (intro *sets.Int sets.countable-nat-UN subsetI, safe*)
 apply (case-tac *set (from-nat i) \in I*)
 apply *simp-all*
 apply (rule *singleton-subspace-set-in-sets[OF measurable-sets[OF MN]]*)
 using *assms* $\langle y \in \text{sets } N \rangle$
 apply (auto simp: *space-PiF*)
 done
 finally show $A -' y \cap \text{space } (\text{PiF } I \ M) \in \text{sets } (\text{PiF } I \ M)$.
next
 fix x assume $x \in \text{space } (\text{PiF } I \ M)$ **thus** $A \ x \in \text{space } N$
 using $MN[\text{of domain } x]$ **by** (auto simp: *space-PiF measurable-space Pi'-def*)
 qed

lemma *measurable-PiF*:
 assumes $f: \bigwedge x. x \in \text{space } N \implies \text{domain } (f \ x) \in I \wedge (\forall i \in \text{domain } (f \ x). (f \ x) \ i \in \text{space } (M \ i))$
 assumes $S: \bigwedge J \ S. J \in I \implies (\bigwedge i. i \in J \implies S \ i \in \text{sets } (M \ i)) \implies$


```

  f - ‘ (Pi' J S) ∩ space N ∈ sets N
shows f ∈ measurable N (PiF I M)
unfolding PiF-def
using PiF-gen-subset
apply (rule measurable-measure-of)
using f apply force
apply (insert S, auto)
done

lemma restrict-sets-measurable:
  assumes A: A ∈ sets (PiF I M) and J ⊆ I
  shows A ∩ {m. domain m ∈ J} ∈ sets (PiF J M)
  using A[unfolded sets-PiF]
proof (induct A)
  case (Basic a)
  then obtain K S where S: a = Pi' K S K ∈ I (∀ i ∈ K. S i ∈ sets (M i))
    by auto
  show ?case
  proof cases
    assume K ∈ J
    hence a ∩ {m. domain m ∈ J} ∈ {Pi' K X | X K. K ∈ J ∧ X ∈ (Π j ∈ K. sets (M j))} using S
      by (auto intro!: exI[where x=K] exI[where x=S] simp: Pi'-def)
    also have ... ⊆ sets (PiF J M) unfolding sets-PiF by auto
    finally show ?thesis .
  next
    assume K ∉ J
    hence a ∩ {m. domain m ∈ J} = {} using S by (auto simp: Pi'-def)
    also have ... ∈ sets (PiF J M) by simp
    finally show ?thesis .
  qed
next
case (Union a)
have ⋃ (a ‘ UNIV) ∩ {m. domain m ∈ J} = (⋃ i. (a i ∩ {m. domain m ∈ J}))
  by simp
also have ... ∈ sets (PiF J M) using Union by (intro sets.countable-nat-UN)
auto
finally show ?case .
next
case (Compl a)
have (space (PiF I M) - a) ∩ {m. domain m ∈ J} = (space (PiF J M) - (a ∩ {m. domain m ∈ J}))
  using ⟨J ⊆ I⟩ by (auto simp: space-PiF Pi'-def)
also have ... ∈ sets (PiF J M) using Compl by auto
finally show ?case by (simp add: space-PiF)
qed simp

```

lemma measurable-finmap-of:

assumes f: $\bigwedge i. (\exists x \in \text{space } N. i \in J x) \implies (\lambda x. f x i) \in \text{measurable } N (M i)$

```

assumes  $J: \bigwedge x. x \in \text{space } N \implies J x \in I \bigwedge x. x \in \text{space } N \implies \text{finite } (J x)$ 
assumes  $JN: \bigwedge S. \{x. J x = S\} \cap \text{space } N \in \text{sets } N$ 
shows  $(\lambda x. \text{finmap-of } (J x) (f x)) \in \text{measurable } N \text{ (PiF } I M)$ 
proof (rule measurable-PiF)
  fix  $x$  assume  $x \in \text{space } N$ 
  with  $J[\text{of } x] \text{ measurable-space}[OF f]$ 
  show  $\text{domain } (\text{finmap-of } (J x) (f x)) \in I \wedge$ 
     $(\forall i \in \text{domain } (\text{finmap-of } (J x) (f x)). (\text{finmap-of } (J x) (f x)) i \in \text{space } (M$ 
 $i))$ 
    by auto
  next
    fix  $K S$  assume  $K \in I$  and  $*$ :  $\bigwedge i. i \in K \implies S i \in \text{sets } (M i)$ 
    with  $J$  have  $\text{eq}: (\lambda x. \text{finmap-of } (J x) (f x)) - ' \text{Pi}' K S \cap \text{space } N =$ 
       $(\text{if } \exists x \in \text{space } N. K = J x \wedge \text{finite } K \text{ then if } K = \{\} \text{ then } \{x \in \text{space } N. J x$ 
 $= K\}$ 
       $\text{else } (\bigcap i \in K. (\lambda x. f x i) - ' S i \cap \{x \in \text{space } N. J x = K\}) \text{ else } \{\})$ 
      by (auto simp: Pi'-def)
    have  $r: \{x \in \text{space } N. J x = K\} = \text{space } N \cap (\{x. J x = K\} \cap \text{space } N)$  by
      auto
    show  $(\lambda x. \text{finmap-of } (J x) (f x)) - ' \text{Pi}' K S \cap \text{space } N \in \text{sets } N$ 
      unfolding eq r
      apply (simp del: INT-simps add: )
      apply (intro conjI impI sets.finite-INT JN sets.Int[OF sets.top])
      apply simp apply assumption
      apply (subst Int-assoc[symmetric])
      apply (rule sets.Int)
      apply (intro measurable-sets[OF f] *) apply force apply assumption
      apply (intro JN)
      done
  qed

```

```

lemma measurable-PiM-finmap-of:
  assumes finite J
  shows  $\text{finmap-of } J \in \text{measurable } (Pi_M J M) \text{ (PiF } \{J\} M)$ 
  apply (rule measurable-finmap-of)
  apply (rule measurable-component-singleton)
  apply simp
  apply rule
  apply (rule  $\langle \text{finite } J \rangle$ )
  apply simp
  done

```

```

lemma proj-measurable-singleton:
  assumes  $A \in \text{sets } (M i)$ 
  shows  $(\lambda x. (x)_F i) - ' A \cap \text{space } (PiF \{I\} M) \in \text{sets } (PiF \{I\} M)$ 
proof cases
  assume  $i \in I$ 
  hence  $(\lambda x. (x)_F i) - ' A \cap \text{space } (PiF \{I\} M) =$ 
     $Pi' I (\lambda x. \text{if } x = i \text{ then } A \text{ else } \text{space } (M x))$ 

```

```

    using sets.sets-into-space[OF ]  $\langle A \in \text{sets } (M \ i) \rangle$  assms
    by (auto simp: space-PiF Pi'-def)
  thus ?thesis using assms  $\langle A \in \text{sets } (M \ i) \rangle$ 
    by (intro in-sets-PiFI) auto
next
  assume  $i \notin I$ 
  hence  $(\lambda x. (x)_F \ i) - 'A \cap \text{space } (PiF \ \{I\} \ M) =$ 
     $(\text{if } \text{undefined} \in A \text{ then } \text{space } (PiF \ \{I\} \ M) \text{ else } \{\})$  by (auto simp: space-PiF
Pi'-def)
  thus ?thesis by simp
qed

```

lemma *measurable-proj-singleton*:

```

  assumes  $i \in I$ 
  shows  $(\lambda x. (x)_F \ i) \in \text{measurable } (PiF \ \{I\} \ M) \ (M \ i)$ 
  by (unfold measurable-def, intro CollectI conjI ballI proj-measurable-singleton
assms)
    (insert  $\langle i \in I \rangle$ , auto simp: space-PiF)

```

lemma *measurable-proj-countable*:

```

  fixes  $I :: 'a :: \text{countable set set}$ 
  assumes  $y \in \text{space } (M \ i)$ 
  shows  $(\lambda x. \text{if } i \in \text{domain } x \text{ then } (x)_F \ i \text{ else } y) \in \text{measurable } (PiF \ I \ M) \ (M \ i)$ 
proof (rule countable-measurable-PiFI)
  fix  $J$  assume  $J \in I$  finite  $J$ 
  show  $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) \in \text{measurable } (PiF \ \{J\} \ M) \ (M \ i)$ 
    unfolding measurable-def
  proof safe
    fix  $z$  assume  $z \in \text{sets } (M \ i)$ 
    have  $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) - 'z \cap \text{space } (PiF \ \{J\} \ M) =$ 
       $(\lambda x. \text{if } i \in J \text{ then } (x)_F \ i \text{ else } y) - 'z \cap \text{space } (PiF \ \{J\} \ M)$ 
      by (auto simp: space-PiF Pi'-def)
    also have  $\dots \in \text{sets } (PiF \ \{J\} \ M)$  using  $\langle z \in \text{sets } (M \ i) \rangle \langle \text{finite } J \rangle$ 
      by (cases  $i \in J$ ) (auto intro!: measurable-sets[OF measurable-proj-singleton])
    finally show  $(\lambda x. \text{if } i \in \text{domain } x \text{ then } x \ i \text{ else } y) - 'z \cap \text{space } (PiF \ \{J\} \ M) \in$ 
       $\text{sets } (PiF \ \{J\} \ M)$  .
  qed (insert  $\langle y \in \text{space } (M \ i) \rangle$ , auto simp: space-PiF Pi'-def)
qed

```

lemma *measurable-restrict-proj*:

```

  assumes  $J \in I$  finite  $J$ 
  shows  $\text{finmap-of } J \in \text{measurable } (PiM \ J \ M) \ (PiF \ I \ M)$ 
  using assms
  by (intro measurable-finmap-of measurable-component-singleton) auto

```

lemma *measurable-proj-PiM*:

```

  fixes  $J \ K :: 'a :: \text{countable set}$  and  $I :: 'a \text{ set set}$ 
  assumes finite  $J \ J \in I$ 
  assumes  $x \in \text{space } (PiM \ J \ M)$ 

```

```

shows  $\text{proj} \in \text{measurable } (PiF \{J\} M) (PiM J M)$ 
proof (rule measurable-PiM-single)
  show  $\text{proj} \in \text{space } (PiF \{J\} M) \rightarrow (\Pi_E i \in J. \text{space } (M i))$ 
  using assms by (auto simp add: space-PiM space-PiF extensional-def sets-PiF
Pi'-def)
next
  fix  $A$  assume  $A: i \in J \ A \in \text{sets } (M i)$ 
  show  $\{\omega \in \text{space } (PiF \{J\} M). (\omega)_F i \in A\} \in \text{sets } (PiF \{J\} M)$ 
  proof
    have  $\{\omega \in \text{space } (PiF \{J\} M). (\omega)_F i \in A\} =$ 
       $(\lambda \omega. (\omega)_F i) - ' A \cap \text{space } (PiF \{J\} M)$  by auto
    also have  $\dots \in \text{sets } (PiF \{J\} M)$ 
    using assms by (auto intro: measurable-sets[OF measurable-proj-singleton]
simp: space-PiM)
    finally show ?thesis .
  qed simp
qed

```

```

lemma space-PiF-singleton-eq-product:
  assumes finite I
  shows  $\text{space } (PiF \{I\} M) = (\Pi' i \in I. \text{space } (M i))$ 
  by (auto simp: product-def space-PiF assms)

```

adapted from $\text{sets } (PiM ?I ?M) = \text{sigma-sets } (\Pi_E i \in ?I. \text{space } (?M i)) \{ \{f \in \Pi_E i \in ?I. \text{space } (?M i). f i \in A\} \mid i A. i \in ?I \wedge A \in \text{sets } (?M i) \}$

```

lemma sets-PiF-single:
  assumes finite I I ≠ {}
  shows  $\text{sets } (PiF \{I\} M) =$ 
     $\text{sigma-sets } (\Pi' i \in I. \text{space } (M i))$ 
     $\{ \{f \in \Pi' i \in I. \text{space } (M i). f i \in A\} \mid i A. i \in I \wedge A \in \text{sets } (M i) \}$ 
    (is - = sigma-sets ?Ω ?R)
  unfolding sets-PiF-singleton
proof (rule sigma-sets-eqI)
  interpret  $R: \text{sigma-algebra } ?Ω \text{ sigma-sets } ?Ω ?R$  by (rule sigma-algebra-sigma-sets)
auto
  fix  $A$  assume  $A \in \{Pi' I X \mid X. X \in (\Pi j \in I. \text{sets } (M j))\}$ 
  then obtain  $X$  where  $A = Pi' I X \ X \in (\Pi j \in I. \text{sets } (M j))$  by auto
  show  $A \in \text{sigma-sets } ?Ω ?R$ 
  proof -
    from  $\langle I \neq \{\} \rangle X$  have  $A = (\bigcap j \in I. \{f \in \text{space } (PiF \{I\} M). f j \in X j\})$ 
    using sets.sets-into-space
    by (auto simp: space-PiF product-def) blast
    also have  $\dots \in \text{sigma-sets } ?Ω ?R$ 
    using  $X \langle I \neq \{\} \rangle$  assms by (intro R.finite-INT) (auto simp: space-PiF)
    finally show  $A \in \text{sigma-sets } ?Ω ?R$  .
  qed
next
  fix  $A$  assume  $A \in ?R$ 
  then obtain  $i B$  where  $A: A = \{f \in \Pi' i \in I. \text{space } (M i). f i \in B\} \ i \in I \ B \in$ 

```

```

sets (M i)
  by auto
  then have A = ( $\Pi' j \in I. \text{if } j = i \text{ then } B \text{ else space } (M j)$ )
    using sets.sets-into-space[OF A(3)]
    apply (auto simp: Pi'-iff split: if-split-asm)
    apply blast
  done
  also have ...  $\in$  sigma-sets ? $\Omega$  {Pi' I X |X. X  $\in$  ( $\Pi j \in I. \text{sets } (M j)$ )}
    using A
    by (intro sigma-sets.Basic )
      (auto intro: exI[where x= $\lambda j. \text{if } j = i \text{ then } B \text{ else space } (M j)$ ])
  finally show A  $\in$  sigma-sets ? $\Omega$  {Pi' I X |X. X  $\in$  ( $\Pi j \in I. \text{sets } (M j)$ )}.
qed

```

adapted from ($\bigwedge i. i \in ?I \implies ?A \ i = ?B \ i$) $\implies Pi_E \ ?I \ ?A = Pi_E \ ?I \ ?B$

lemma *Pi'-cong*:
 assumes *finite I*
 assumes $\bigwedge i. i \in I \implies f \ i = g \ i$
 shows $Pi' \ I \ f = Pi' \ I \ g$
using *assms* **by** (auto simp: Pi'-def)

adapted from $\llbracket \text{finite } ?I; \bigwedge i \ n \ m. \llbracket i \in ?I; n \leq m \rrbracket \implies ?A \ n \ i \subseteq ?A \ m \ i \rrbracket$
 $\implies (\bigcup_n Pi \ ?I \ (?A \ n)) = (\Pi \ i \in ?I. \bigcup_n ?A \ n \ i)$

lemma *Pi'-UN*:
 fixes $A :: \text{nat} \Rightarrow 'i \Rightarrow 'a \ \text{set}$
 assumes *finite I*
 assumes *mono*: $\bigwedge i \ n \ m. i \in I \implies n \leq m \implies A \ n \ i \subseteq A \ m \ i$
 shows $(\bigcup n. Pi' \ I \ (A \ n)) = Pi' \ I \ (\lambda i. \bigcup n. A \ n \ i)$
proof (intro set-eqI iffI)
 fix f **assume** $f \in Pi' \ I \ (\lambda i. \bigcup n. A \ n \ i)$
then have $\forall i \in I. \exists n. f \ i \in A \ n \ i$ **domain** $f = I$ **by** (auto simp: $\langle \text{finite } I \rangle$ Pi'-def)
from bchoice[OF this(1)] **obtain** n **where** $n: \bigwedge i. i \in I \implies f \ i \in (A \ n \ i) \ i$ **by**
 auto
obtain k **where** $k: \bigwedge i. i \in I \implies n \ i \leq k$
using $\langle \text{finite } I \rangle$ finite-nat-set-iff-bounded-le[of $n \ i$] **by** auto
have $f \in Pi' \ I \ (\lambda i. A \ k \ i)$
proof
 fix i **assume** $i \in I$
from *mono*[OF this, of $n \ i \ k$] k [OF this] n [OF this] $\langle \text{domain } f = I \rangle \langle i \in I \rangle$
show $f \ i \in A \ k \ i$ **by** (auto simp: $\langle \text{finite } I \rangle$)
qed (simp add: $\langle \text{domain } f = I \rangle \langle \text{finite } I \rangle$)
then show $f \in (\bigcup n. Pi' \ I \ (A \ n))$ **by** auto
qed (auto simp: Pi'-def $\langle \text{finite } I \rangle$)

adapted from $\llbracket \bigwedge i. i \in ?I \implies \exists S \subseteq ?E \ i. \text{countable } S \wedge ?\Omega \ i = \bigcup S; \bigwedge i. i \in ?I \implies ?E \ i \subseteq \text{Pow } (? \Omega \ i); \bigwedge j. j \in ?J \implies \text{finite } j; \bigcup ?J = ?I \rrbracket \implies$
 $\text{sets } (Pi_M \ ?I \ (\lambda i. \text{sigma } (? \Omega \ i) \ (?E \ i))) = \text{sets } (\text{sigma } (Pi_E \ ?I \ ? \Omega) \ \{\{f \in Pi_E \ ?I \ ? \Omega. \forall i \in j. f \ i \in A \ i\} \mid A \ j. j \in ?J \wedge A \in Pi \ j \ ?E\})$

lemma *sigma-fprod-algebra-sigma-eq*:

```

fixes  $E :: 'i \Rightarrow 'a \text{ set set}$  and  $S :: 'i \Rightarrow \text{nat} \Rightarrow 'a \text{ set}$ 
assumes  $[simp]: \text{finite } I \Rightarrow I \neq \{\}$ 
  and  $S\text{-union}: \bigwedge i. i \in I \implies (\bigcup j. S\ i\ j) = \text{space } (M\ i)$ 
  and  $S\text{-in-}E: \bigwedge i. i \in I \implies \text{range } (S\ i) \subseteq E\ i$ 
assumes  $E\text{-closed}: \bigwedge i. i \in I \implies E\ i \subseteq \text{Pow } (\text{space } (M\ i))$ 
  and  $E\text{-generates}: \bigwedge i. i \in I \implies \text{sets } (M\ i) = \text{sigma-sets } (\text{space } (M\ i))\ (E\ i)$ 
defines  $P == \{ Pi' \ I\ F \mid F. \forall i \in I. F\ i \in E\ i \}$ 
shows  $\text{sets } (PiF\ \{I\}\ M) = \text{sigma-sets } (\text{space } (PiF\ \{I\}\ M))\ P$ 
proof
  let  $?P = \text{sigma } (\text{space } (PiF\ \{I\}\ M))\ P$ 
  from  $\langle \text{finite } I \rangle [THEN\ ex\text{-bij-betw-finite-nat}]$  obtain  $T$  where  $\text{bij-betw } T\ I\ \{0..<\text{card } I\} ..$ 
  then have  $T: \bigwedge i. i \in I \implies T\ i < \text{card } I \wedge i. i \in I \implies \text{the-inv-into } I\ T\ (T\ i) = i$ 
    by  $(\text{auto simp add: bij-betw-def set-eq-iff image-iff the-inv-into-f-f simp del: } \langle \text{finite } I \rangle)$ 
  have  $P\text{-closed}: P \subseteq \text{Pow } (\text{space } (PiF\ \{I\}\ M))$ 
    using  $E\text{-closed}$  by  $(\text{auto simp: space-PiF } P\text{-def } Pi'\text{-iff subset-eq})$ 
  then have  $\text{space-}P: \text{space } ?P = (\Pi' i \in I. \text{space } (M\ i))$ 
    by  $(\text{simp add: space-PiF})$ 
  have  $\text{sets } (PiF\ \{I\}\ M) =$ 
     $\text{sigma-sets } (\text{space } ?P)\ \{\{f \in \Pi' i \in I. \text{space } (M\ i). f\ i \in A\} \mid i \in I \wedge A \in \text{sets } (M\ i)\}$ 
    using  $\text{sets-PiF-single}[of\ I\ M]$  by  $(\text{simp add: space-}P)$ 
  also have  $\dots \subseteq \text{sets } (\text{sigma } (\text{space } (PiF\ \{I\}\ M))\ P)$ 
proof  $(\text{safe intro!}: \text{sets.sigma-sets-subset})$ 
    fix  $i\ A$  assume  $i \in I$  and  $A: A \in \text{sets } (M\ i)$ 
    have  $(\lambda x. (x)_F\ i) \in \text{measurable } ?P\ (\text{sigma } (\text{space } (M\ i))\ (E\ i))$ 
proof  $(\text{subst measurable-iff-measure-of})$ 
    show  $E\ i \subseteq \text{Pow } (\text{space } (M\ i))$  using  $\langle i \in I \rangle$  by  $\text{fact}$ 
    from  $\text{space-}P\ \langle i \in I \rangle$  show  $(\lambda x. (x)_F\ i) \in \text{space } ?P \rightarrow \text{space } (M\ i)$ 
    by  $\text{auto}$ 
    show  $\forall A \in E\ i. (\lambda x. (x)_F\ i) - 'A \cap \text{space } ?P \in \text{sets } ?P$ 
proof
    fix  $A$  assume  $A: A \in E\ i$ 
    from  $T$  have  $xs. \text{length } xs = \text{card } I$ 
     $\wedge (\forall j \in I. (g)_F\ j \in (\text{if } i = j \text{ then } A \text{ else } S\ j\ (xs\ !\ T\ j)))$ 
    if  $\text{domain } g = I$  and  $\forall j \in I. (g)_F\ j \in (\text{if } i = j \text{ then } A \text{ else } S\ j\ (f\ j))$  for  $g\ f$ 
    using  $\text{that}$  by  $(\text{auto intro!}: exI\ [of\ \text{-map } (\lambda n. f\ (\text{the-inv-into } I\ T\ n))])$ 
    from  $A$  have  $(\lambda x. (x)_F\ i) - 'A \cap \text{space } ?P = (\Pi' j \in I. \text{if } i = j \text{ then } A \text{ else } \text{space } (M\ j))$ 
    using  $E\text{-closed } \langle i \in I \rangle$  by  $(\text{auto simp: space-}P\ Pi'\text{-iff subset-eq split: if-split-asm})$ 
    also have  $\dots = (\Pi' j \in I. \bigcup n. \text{if } i = j \text{ then } A \text{ else } S\ j\ n)$ 
    by  $(\text{intro } Pi'\text{-cong})\ (\text{simp-all add: } S\text{-union})$ 
    also have  $\dots = \{g. \text{domain } g = I \wedge (\exists f. \forall j \in I. (g)_F\ j \in (\text{if } i = j \text{ then } A \text{ else } S\ j\ (f\ j)))\}$ 
    by  $(\text{rule set-eqI})\ (\text{simp del: if-image-distrib add: } Pi'\text{-iff bchoice-iff})$ 

```

```

    also have ... = ( $\bigcup xs \in \{xs. \text{length } xs = \text{card } I\}. \Pi' j \in I. \text{if } i = j \text{ then } A \text{ else } S j (xs ! T j)$ )
    using * by (auto simp add: Pi'-iff split del: if-split)
    also have ...  $\in \text{sets } ?P$ 
    proof (safe intro!: sets.countable-UN)
      fix xs show ( $\Pi' j \in I. \text{if } i = j \text{ then } A \text{ else } S j (xs ! T j)$ )  $\in \text{sets } ?P$ 
      using A S-in-E
      by (simp add: P-closed)
      (auto simp: P-def subset-eq intro!: exI[of -  $\lambda j. \text{if } i = j \text{ then } A \text{ else } S j (xs ! T j)$ ])
    qed
    finally show ( $\lambda x. (x)_F i$ )  $- 'A \cap \text{space } ?P \in \text{sets } ?P$ 
    using P-closed by simp
  qed
qed
from measurable-sets[OF this, of A] A  $\langle i \in I \rangle$  E-closed
have ( $\lambda x. (x)_F i$ )  $- 'A \cap \text{space } ?P \in \text{sets } ?P$ 
by (simp add: E-generates)
also have ( $\lambda x. (x)_F i$ )  $- 'A \cap \text{space } ?P = \{f \in \Pi' i \in I. \text{space } (M i). f i \in A\}$ 
using P-closed by (auto simp: space-PiF)
finally show ...  $\in \text{sets } ?P$  .
qed
finally show sets (PiF {I} M)  $\subseteq \text{sigma-sets } (\text{space } (\text{PiF } \{I\} M)) P$ 
by (simp add: P-closed)
show sigma-sets (space (PiF {I} M)) P  $\subseteq \text{sets } (\text{PiF } \{I\} M)$ 
using  $\langle \text{finite } I \rangle \langle I \neq \{\} \rangle$ 
by (auto intro!: sets.sigma-sets-subset product-in-sets-PiFI simp: E-generates
P-def)
qed

```

lemma *product-open-generates-sets-PiF-single:*

```

  assumes I  $\neq \{\}$ 
  assumes [simp]: finite I
  shows sets (PiF {I} ( $\lambda -. \text{borel}::'b::\text{second-countable-topology measure}$ )) =
    sigma-sets (space (PiF {I} ( $\lambda -. \text{borel}$ ))) {Pi' I F |F. ( $\forall i \in I. F i \in \text{Collect open}$ )}
  proof -
    from open-countable-basisE[OF open-UNIV] obtain S::'b set set
      where S:  $S \subseteq (\text{SOME } B. \text{countable } B \wedge \text{topological-basis } B) \text{ UNIV} = \bigcup S$ 
      by auto
    show ?thesis
    proof (rule sigma-fprod-algebra-sigma-eq)
      show finite I by simp
      show I  $\neq \{\}$  by fact
      define S' where S' = from-nat-into S
      show ( $\bigcup j. S' j$ ) = space borel
      using S
      apply (auto simp add: from-nat-into countable-basis-proj S'-def basis-proj-def)
      apply (metis (lifting, mono-tags) UNIV-I UnionE basis-proj-def count-

```

```

able-basis-proj countable-subset from-nat-into-surj)
  done
  show range  $S' \subseteq \text{Collect open}$ 
  using  $S$ 
  apply (auto simp add: from-nat-into countable-basis-proj  $S'$ -def)
  apply (metis UNIV-not-empty Union-empty from-nat-into subsetD topologi-
cal-basis-open[OF basis-proj] basis-proj-def)
  done
  show  $\text{Collect open} \subseteq \text{Pow (space borel)}$  by simp
  show  $\text{sets borel} = \text{sigma-sets (space borel) (Collect open)}$ 
    by (simp add: borel-def)
qed
qed

lemma finmap-UNIV[simp]:  $(\bigcup_{J \in \text{Collect finite.}} \Pi' j \in J. \text{UNIV}) = \text{UNIV}$  by
auto

lemma borel-eq-PiF-borel:
  shows  $(\text{borel} :: ('i :: \text{countable} \Rightarrow_F 'a :: \text{polish-space}) \text{measure}) =$ 
     $\text{PiF (Collect finite) } (\lambda \cdot. \text{borel} :: 'a \text{ measure})$ 
  unfolding borel-def PiF-def
proof (rule measure-eqI, clarsimp, rule sigma-sets-eqI)
  fix  $a :: ('i \Rightarrow_F 'a) \text{ set}$  assume  $a \in \text{Collect open}$  hence open  $a$  by simp
  then obtain  $B'$  where  $B': B' \subseteq \text{basis-finmap } a = \bigcup B'$ 
    using finmap-topological-basis by (force simp add: topological-basis-def)
  have  $a \in \text{sigma UNIV } \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV}$ 
     $(\text{Collect open})\}$ 
    unfolding  $\langle a = \bigcup B' \rangle$ 
  proof (rule sets.countable-Union)
    from  $B'$  countable-basis-finmap show countable  $B'$  by (metis countable-subset)
  next
    show  $B' \subseteq \text{sets (sigma UNIV } \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV}$ 
     $(\text{Collect open})\})$  (is -
     $\subseteq \text{sets ?s}$ )
    proof
      fix  $x$  assume  $x \in B'$  with  $B'$  have  $x \in \text{basis-finmap}$  by auto
      then obtain  $J X$  where  $x = Pi' J X$  finite  $J X \in J \rightarrow \text{sigma-sets UNIV}$ 
    (Collect open)
      by (auto simp: basis-finmap-def topological-basis-open[OF basis-proj])
      thus  $x \in \text{sets ?s}$  by auto
    qed
  qed
  thus  $a \in \text{sigma-sets UNIV } \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV}$ 
     $(\text{Collect open})\}$ 
    by simp
next
  fix  $b :: ('i \Rightarrow_F 'a) \text{ set}$ 
  assume  $b \in \{Pi' J X \mid X J. \text{finite } J \wedge X \in J \rightarrow \text{sigma-sets UNIV (Collect open)}\}$ 
  hence  $b': b \in \text{sets (PiF (Collect finite) } (\lambda \cdot. \text{borel}))$  by (auto simp: sets-PiF

```



```

borel-def)
  let ?b =  $\lambda J. b \cap \{x. \text{domain } x = J\}$ 
  have  $b = \bigcup ((\lambda J. ?b J) \text{ `Collect finite})$  by auto
  also have  $\dots \in \text{sets borel}$ 
  proof (rule sets.countable-Union, safe)
    fix J::'i set assume finite J
    { assume ef:  $J = \{\}$ 
      have  $?b J \in \text{sets borel}$ 
      proof cases
        assume  $?b J \neq \{\}$ 
        then obtain f where  $f \in b$  domain  $f = \{\}$  using ef by auto
        hence  $?b J = \{f\}$  using  $\langle J = \{\} \rangle$ 
          by (auto simp: finmap-eq-iff)
        also have  $\{f\} \in \text{sets borel}$  by simp
        finally show ?thesis .
      qed simp
    } moreover {
      assume  $J \neq (\{\}::'i \text{ set})$ 
      have  $(?b J) = b \cap \{m. \text{domain } m \in \{J\}\}$  by auto
      also have  $\dots \in \text{sets (PiF } \{J\} (\lambda-. \text{ borel}))$ 
        using b' by (rule restrict-sets-measurable) (auto simp:  $\langle \text{finite } J \rangle$ )
      also have  $\dots = \text{sigma-sets (space (PiF } \{J\} (\lambda-. \text{ borel}))}$ 
         $\{Pi' (J) F \mid F. (\forall j \in J. F j \in \text{Collect open})\}$ 
        (is - = sigma-sets - ?P)
      by (rule product-open-generates-sets-PiF-single[OF  $\langle J \neq \{\} \rangle \langle \text{finite } J \rangle$ ])
      also have  $\dots \subseteq \text{sigma-sets UNIV (Collect open)}$ 
        by (intro sigma-sets-mono'') (auto intro!: open-Pi'I simp: space-PiF)
      finally have  $(?b J) \in \text{sets borel}$  by (simp add: borel-def)
    } ultimately show  $(?b J) \in \text{sets borel}$  by blast
  qed (simp add: countable-Collect-finite)
  finally show  $b \in \text{sigma-sets UNIV (Collect open)}$  by (simp add: borel-def)
qed (simp add: emeasure-sigma borel-def PiF-def)

```

22.10 Isomorphism between Functions and Finite Maps

lemma *measurable-finmap-compose:*

shows $(\lambda m. \text{compose } J m f) \in \text{measurable (PiM } (f \text{ ` } J) (\lambda-. M)) (\text{PiM } J (\lambda-. M))$

unfolding *compose-def* **by** *measurable*

lemma *measurable-compose-inv:*

assumes *inj*: $\bigwedge j. j \in J \implies f' (f j) = j$

shows $(\lambda m. \text{compose } (f \text{ ` } J) m f') \in \text{measurable (PiM } J (\lambda-. M)) (\text{PiM } (f \text{ ` } J) (\lambda-. M))$

unfolding *compose-def* **by** (rule *measurable-restrict*) (auto simp: *inj*)

locale *function-to-finmap* =

fixes $J::'a \text{ set}$ **and** $f :: 'a \Rightarrow 'b::\text{countable}$ **and** f'

assumes [*simp*]: *finite J*

assumes *inv*: $i \in J \implies f' (f i) = i$
begin

to measure finmaps

definition $fm = (finmap-of (f \text{ ' } J)) \circ (\lambda g. compose (f \text{ ' } J) g f')$

lemma *domain-fm[simp]*: $domain (fm x) = f \text{ ' } J$
unfolding *fm-def* **by** *simp*

lemma *fm-restrict[simp]*: $fm (restrict y J) = fm y$
unfolding *fm-def* **by** (*auto simp: compose-def inv intro: restrict-ext*)

lemma *fm-product*:

assumes $\bigwedge i. space (M i) = UNIV$
shows $fm - \text{' } Pi' (f \text{ ' } J) S \cap space (Pi_M J M) = (\Pi_E j \in J. S (f j))$
using *assms*
by (*auto simp: inv fm-def compose-def space-PiM Pi'-def*)

lemma *fm-measurable*:

assumes $f \text{ ' } J \in N$
shows $fm \in measurable (Pi_M J (\lambda-. M)) (Pi_F N (\lambda-. M))$
unfolding *fm-def*
proof (*rule measurable-comp, rule measurable-compose-inv*)
show $finmap-of (f \text{ ' } J) \in measurable (Pi_M (f \text{ ' } J) (\lambda-. M)) (Pi_F N (\lambda-. M))$
using *assms* **by** (*intro measurable-finmap-of measurable-component-singleton*)
auto
qed (*simp-all add: inv*)

lemma *proj-fm*:

assumes $x \in J$
shows $fm m (f x) = m x$
using *assms* **by** (*auto simp: fm-def compose-def o-def inv*)

lemma *inj-on-compose-f'*: $inj-on (\lambda g. compose (f \text{ ' } J) g f') (extensional J)$

proof (*rule inj-on-inverseI*)
fix $x::'a \Rightarrow 'c$ **assume** $x \in extensional J$
thus $(\lambda x. compose J x f) (compose (f \text{ ' } J) x f') = x$
by (*auto simp: compose-def inv extensional-def*)
qed

lemma *inj-on-fm*:

assumes $\bigwedge i. space (M i) = UNIV$
shows $inj-on fm (space (Pi_M J M))$
using *assms*
apply (*auto simp: fm-def space-PiM PiE-def*)
apply (*rule comp-inj-on*)
apply (*rule inj-on-compose-f'*)
apply (*rule finmap-of-inj-on-extensional-finite*)
apply *simp*

apply (*auto*)
done

to measure functions

definition $mf = (\lambda g. \text{compose } J \ g \ f) \ o \ \text{proj}$

lemma *mf-fm*:

assumes $x \in \text{space } (Pi_M \ J \ (\lambda-. \ M))$

shows $mf \ (fm \ x) = x$

proof –

have $mf \ (fm \ x) \in \text{extensional } J$

by (*auto simp: mf-def extensional-def compose-def*)

moreover

have $x \in \text{extensional } J$ **using** *assms sets.sets-into-space*

by (*force simp: space-PiM PiE-def*)

moreover

{ fix i **assume** $i \in J$

hence $mf \ (fm \ x) \ i = x \ i$

by (*auto simp: inv mf-def compose-def fm-def*)

}

ultimately

show *?thesis* **by** (*rule extensionalityI*)

qed

lemma *mf-measurable*:

assumes $\text{space } M = \text{UNIV}$

shows $mf \in \text{measurable } (PiF \ \{f \ ' \ J\} \ (\lambda-. \ M)) \ (PiM \ J \ (\lambda-. \ M))$

unfolding *mf-def*

proof (*rule measurable-comp, rule measurable-proj-PiM*)

show $(\lambda g. \text{compose } J \ g \ f) \in \text{measurable } (Pi_M \ (f \ ' \ J) \ (\lambda x. \ M)) \ (Pi_M \ J \ (\lambda-. \ M))$

by (*rule measurable-finmap-compose*)

qed (*auto simp add: space-PiM extensional-def assms*)

lemma *fm-image-measurable*:

assumes $\text{space } M = \text{UNIV}$

assumes $X \in \text{sets } (Pi_M \ J \ (\lambda-. \ M))$

shows $fm \ ' \ X \in \text{sets } (PiF \ \{f \ ' \ J\} \ (\lambda-. \ M))$

proof –

have $fm \ ' \ X = (mf) \ - \ ' \ X \cap \text{space } (PiF \ \{f \ ' \ J\} \ (\lambda-. \ M))$

proof *safe*

fix x **assume** $x \in X$

with *mf-fm[of x] sets.sets-into-space[OF assms(2)]* **show** $fm \ x \in mf \ - \ ' \ X$ **by**

auto

show $fm \ x \in \text{space } (PiF \ \{f \ ' \ J\} \ (\lambda-. \ M))$ **by** (*simp add: space-PiF assms*)

next

fix $y \ x$

assume $x: mf \ y \in X$

assume $y: y \in \text{space } (PiF \ \{f \ ' \ J\} \ (\lambda-. \ M))$

thus $y \in fm \ ' \ X$

```

    by (intro image-eqI[OF - x], unfold finmap-eq-iff)
      (auto simp: space-PiF fm-def mf-def compose-def inv Pi'-def)
  qed
  also have ... ∈ sets (PiF {f ‘ J} (λ-. M))
    using assms
    by (intro measurable-sets[OF mf-measurable]) auto
  finally show ?thesis .
  qed

```

lemma *fm-image-measurable-finite*:

```

  assumes space M = UNIV
  assumes X ∈ sets (PiM J (λ-. M::'c measure))
  shows fm ‘ X ∈ sets (PiF (Collect finite) (λ-. M::'c measure))
  using fm-image-measurable[OF assms]
  by (rule subspace-set-in-sets) (auto simp: finite-subset)

```

measure on finmaps

definition *mapmeasure* $M\ N = \text{distr } M\ (\text{PiF } (\text{Collect finite})\ N)\ (\text{fm})$

lemma *sets-mapmeasure[simp]*: $\text{sets } (\text{mapmeasure } M\ N) = \text{sets } (\text{PiF } (\text{Collect finite})\ N)$

unfolding *mapmeasure-def* **by** *simp*

lemma *space-mapmeasure[simp]*: $\text{space } (\text{mapmeasure } M\ N) = \text{space } (\text{PiF } (\text{Collect finite})\ N)$

unfolding *mapmeasure-def* **by** *simp*

lemma *mapmeasure-PiF*:

```

  assumes s1: space M = space (PiM J (λ-. N))
  assumes s2: sets M = sets (PiM J (λ-. N))
  assumes space N = UNIV
  assumes X ∈ sets (PiF (Collect finite) (λ-. N))
  shows emeasure (mapmeasure M (λ-. N)) X = emeasure M ((fm -‘ X ∩ extensional J))
  using assms
  by (auto simp: measurable-cong-sets[OF s2 refl] mapmeasure-def emeasure-distr
    fm-measurable space-PiM PiE-def)

```

lemma *mapmeasure-PiM*:

```

  fixes N::'c measure
  assumes s1: space M = space (PiM J (λ-. N))
  assumes s2: sets M = (PiM J (λ-. N))
  assumes N: space N = UNIV
  assumes X: X ∈ sets M
  shows emeasure M X = emeasure (mapmeasure M (λ-. N)) (fm ‘ X)
  unfolding mapmeasure-def
  proof (subst emeasure-distr, subst measurable-cong-sets[OF s2 refl], rule fm-measurable)
    have X ⊆ space (PiM J (λ-. N)) using assms by (simp add: sets.sets-into-space)
    from assms inj-on-fm[of λ-. N] subsetD[OF this] have fm -‘ fm ‘ X ∩ space

```

```

( $Pi_M J (\lambda \cdot N) = X$ )
  by (auto simp: vimage-image-eq inj-on-def)
  thus  $emeasure M X = emeasure M (fm - ' fm ' X \cap space M)$  using  $s1$ 
  by simp
  show  $fm ' X \in sets (PiF (Collect finite) (\lambda \cdot N))$ 
    by (rule fm-image-measurable-finite[OF  $N X[simplified s2]$ ])
qed simp

end

end

```

23 Projective Limit

```

theory Projective-Limit
imports
  Fin-Map
  Infinite-Product-Measure
  HOL-Library.Diagonal-Subsequence
begin

```

23.1 Sequences of Finite Maps in Compact Sets

```

locale finmap-seqs-into-compact =
  fixes  $K::nat \Rightarrow (nat \Rightarrow_F 'a::metric-space) set$  and  $f::nat \Rightarrow (nat \Rightarrow_F 'a)$  and
   $M$ 
  assumes compact:  $\bigwedge n. compact (K n)$ 
  assumes f-in-K:  $\bigwedge n. K n \neq \{\}$ 
  assumes domain-K:  $\bigwedge n. k \in K n \implies domain k = domain (f n)$ 
  assumes proj-in-K:
     $\bigwedge t n m. m \geq n \implies t \in domain (f n) \implies (f m)_F t \in (\lambda k. (k)_F t) ' K n$ 
begin

lemma proj-in-K':  $(\exists n. \forall m \geq n. (f m)_F t \in (\lambda k. (k)_F t) ' K n)$ 
  using proj-in-K f-in-K
proof cases
  obtain  $k$  where  $k \in K (Suc 0)$  using f-in-K by auto
  assume  $\forall n. t \notin domain (f n)$ 
  thus ?thesis
    by (auto intro!: exI[where  $x=1$ ] image-eqI[OF  $\langle k \in K (Suc 0) \rangle$ ]
      simp: domain-K[OF  $\langle k \in K (Suc 0) \rangle$ ])
qed blast

lemma proj-in-KE:
  obtains  $n$  where  $\bigwedge m. m \geq n \implies (f m)_F t \in (\lambda k. (k)_F t) ' K n$ 
  using proj-in-K' by blast

lemma compact-projset:
  shows  $compact ((\lambda k. (k)_F i) ' K n)$ 

```

```

using continuous-proj compact by (rule compact-continuous-image)

end

lemma compactE':
  fixes  $S :: 'a :: \text{metric-space set}$ 
  assumes  $\text{compact } S \ \forall n \geq m. f \ n \in S$ 
  obtains  $l \ r$  where  $l \in S \ \text{strict-mono } (r :: \text{nat} \Rightarrow \text{nat}) \ ((f \circ r) \longrightarrow l) \ \text{sequentially}$ 
proof atomize-elim
  have  $\text{strict-mono } ((+) \ m)$  by (simp add: strict-mono-def)
  have  $\forall n. (f \circ (\lambda i. m + i)) \ n \in S$  using assms by auto
  from seq-compactE[OF  $\langle \text{compact } S \rangle$  [unfolded compact-eq-seq-compact-metric]] this
  obtain  $l \ r$  where  $l \in S \ \text{strict-mono } r \ (f \circ (+) \ m \circ r) \longrightarrow l$  by blast
  hence  $l \in S \ \text{strict-mono } ((\lambda i. m + i) \circ r) \wedge (f \circ ((\lambda i. m + i) \circ r)) \longrightarrow l$ 
    using strict-mono-o[OF  $\langle \text{strict-mono } ((+) \ m) \rangle \langle \text{strict-mono } r \rangle$ ] by (auto simp:
o-def)
  thus  $\exists l \ r. l \in S \wedge \text{strict-mono } r \wedge (f \circ r) \longrightarrow l$  by blast
qed

sublocale finmap-seqs-into-compact  $\subseteq$  subseqs  $\lambda n \ s. (\exists l. (\lambda i. ((f \circ s) \ i)_F \ n) \longrightarrow l)$ 
proof
  fix  $n$  and  $s :: \text{nat} \Rightarrow \text{nat}$ 
  assume strict-mono s
  from proj-in-KE[of n] obtain  $n0$  where  $n0: \bigwedge m. n0 \leq m \implies (f \ m)_F \ n \in (\lambda k. (k)_F \ n) \ 'K \ n0$ 
    by blast
  have  $\forall i \geq n0. ((f \circ s) \ i)_F \ n \in (\lambda k. (k)_F \ n) \ 'K \ n0$ 
  proof safe
    fix  $i$  assume  $n0 \leq i$ 
    also have  $\dots \leq s \ i$  by (rule seq-suble) fact
    finally have  $n0 \leq s \ i$  .
    with  $n0$  show  $((f \circ s) \ i)_F \ n \in (\lambda k. (k)_F \ n) \ 'K \ n0$ 
      by auto
  qed
  then obtain  $ls \ rs$ 
    where  $ls \in (\lambda k. (k)_F \ n) \ 'K \ n0 \ \text{strict-mono } rs \ ((\lambda i. ((f \circ s) \ i)_F \ n) \circ rs) \longrightarrow ls$ 
    by (rule compactE'[OF compact-projset])
  thus  $\exists r'. \text{strict-mono } r' \wedge (\exists l. (\lambda i. ((f \circ (s \circ r')) \ i)_F \ n) \longrightarrow l)$  by (auto simp: o-def)
qed

lemma (in finmap-seqs-into-compact) diagonal-tendsto:  $\exists l. (\lambda i. (f \ (diagseq \ i))_F \ n) \longrightarrow l$ 
proof –
  obtain  $l$  where  $(\lambda i. ((f \circ (diagseq \ o \ (+) \ (Suc \ n))) \ i)_F \ n) \longrightarrow l$ 
  proof (atomize-elim, rule diagseq-holds)
    fix  $r \ s \ n$ 

```

```

assume strict-mono ( $r :: \text{nat} \Rightarrow \text{nat}$ )
assume  $\exists l. (\lambda i. ((f \circ s) i)_F n) \longrightarrow l$ 
then obtain  $l$  where  $((\lambda i. (f i)_F n) \circ s) \longrightarrow l$ 
  by (auto simp: o-def)
hence  $((\lambda i. (f i)_F n) \circ s \circ r) \longrightarrow l$  using  $\langle \text{strict-mono } r \rangle$ 
  by (rule LIMSEQ-subseq-LIMSEQ)
thus  $\exists l. (\lambda i. ((f \circ (s \circ r)) i)_F n) \longrightarrow l$  by (auto simp add: o-def)
qed
hence  $(\lambda i. ((f (\text{diagseq } (i + \text{Suc } n))))_F n) \longrightarrow l$  by (simp add: ac-simps)
hence  $(\lambda i. (f (\text{diagseq } i))_F n) \longrightarrow l$  by (rule LIMSEQ-offset)
thus ?thesis ..
qed

```

23.2 Daniell-Kolmogorov Theorem

Existence of Projective Limit

locale *polish-projective* = *projective-family* I P $\lambda\cdot$. *borel*::*'a*::*polish-space measure*
for I ::*'i set* **and** P
begin

lemma *emeasure-lim-emb*:

assumes $X: J \subseteq I$ *finite* J $X \in \text{sets } (\Pi_M i \in J. \text{borel})$
shows $\lim (\text{emb } I J X) = P J X$

proof (*rule emeasure-lim*)

write $\mu\text{-}G$ ($\langle \mu G \rangle$)

interpret *generator*: *algebra space* $(\Pi_M I (\lambda i. \text{borel}))$ *generator*
by (*rule algebra-generator*)

fix J **and** $B :: \text{nat} \Rightarrow ('i \Rightarrow 'a)$ *set*

assume $J: \bigwedge n. \text{finite } (J n) \bigwedge n. J n \subseteq I \bigwedge n. B n \in \text{sets } (\Pi_M i \in J n. \text{borel})$
incseq J

and $B: \text{decseq } (\lambda n. \text{emb } I (J n) (B n))$

and $0 < (\text{INF } i. P (J i) (B i))$ (**is** $0 < ?a$)

moreover have $?a \leq 1$

using J **by** (*auto intro!*: *INF-lower2*[*of* 0] *prob-space-P*[*THEN prob-space.measure-le-1*])

ultimately obtain r **where** $r: ?a = \text{ennreal } r$ $0 < r$ $r \leq 1$

by (*cases* $?a$) (*auto simp: top-unique*)

define Z **where** $Z n = \text{emb } I (J n) (B n)$ **for** n

have $Z\text{-mono}: n \leq m \implies Z m \subseteq Z n$ **for** $n m$

unfolding $Z\text{-def}$ **using** B [*THEN antimonoD*, *of* $n m$] .

have $J\text{-mono}: \bigwedge n m. n \leq m \implies J n \subseteq J m$

using $\langle \text{incseq } J \rangle$ **by** (*force simp: incseq-def*)

note $[simp] = \langle \bigwedge n. \text{finite } (J n) \rangle$

interpret *prob-space* $P (J i)$ **for** i **using** J *prob-space-P* **by** *simp*

have $P\text{-eq}[simp]$:

sets $(P (J i)) = \text{sets } (\Pi_M i \in J i. \text{borel})$ *space* $(P (J i)) = \text{space } (\Pi_M i \in J i. \text{borel})$ **for** i

using J **by** (*auto simp: sets-P space-P*)

```

have  $Z\ i \in \text{generator}$  for  $i$ 
  unfolding  $Z\text{-def}$  by ( $\text{auto intro!}; \text{generator.intros } J$ )

have  $\text{countable-UN-}J$ :  $\text{countable } (\bigcup n. J\ n)$  by ( $\text{simp add: countable-finite}$ )
define  $Utn$  where  $Utn = \text{to-nat-on } (\bigcup n. J\ n)$ 
interpret  $\text{function-to-finmap } J\ n\ Utn\ \text{from-nat-into } (\bigcup n. J\ n)$  for  $n$ 
  by  $\text{unfold-locales } (\text{auto simp: } Utn\text{-def intro: from-nat-into-to-nat-on}[OF\ \text{countable-UN-}J])$ 
have  $\text{inj-on-}Utn$ :  $\text{inj-on } Utn\ (\bigcup n. J\ n)$ 
  unfolding  $Utn\text{-def}$  using  $\text{countable-UN-}J$  by ( $\text{rule inj-on-to-nat-on}$ )
hence  $\text{inj-on-}Utn\text{-}J$ :  $\bigwedge n. \text{inj-on } Utn\ (J\ n)$  by ( $\text{rule subset-inj-on}$ )  $\text{auto}$ 
define  $P'$  where  $P'\ n = \text{mapmeasure } n\ (P\ (J\ n))\ (\lambda\cdot. \text{borel})$  for  $n$ 
interpret  $P'$ :  $\text{prob-space } P'\ n$  for  $n$ 
  unfolding  $P'\text{-def}$   $\text{mapmeasure-def}$  using  $J$ 
  by ( $\text{auto intro!}; \text{prob-space-distr fm-measurable simp: measurable-cong-sets}[OF\ \text{sets-}P])$ 

let  $?SUP = \lambda n. \text{SUP } K \in \{K. K \subseteq \text{fm } n\ ' (B\ n) \wedge \text{compact } K\}. \text{emeasure } (P'\ n)\ K$ 
{ fix  $n$ 
  have  $\text{emeasure } (P\ (J\ n))\ (B\ n) = \text{emeasure } (P'\ n)\ (\text{fm } n\ ' (B\ n))$ 
    using  $J$  by ( $\text{auto simp: } P'\text{-def mapmeasure-PiM space-}P\ \text{sets-}P$ )
  also
  have  $\dots = ?SUP\ n$ 
  proof ( $\text{rule inner-regular}$ )
    show  $\text{sets } (P'\ n) = \text{sets borel}$  by ( $\text{simp add: borel-eq-PiF-borel } P'\text{-def}$ )
  next
    show  $\text{fm } n\ ' B\ n \in \text{sets borel}$ 
    unfolding  $\text{borel-eq-PiF-borel}$  by ( $\text{auto simp: } P'\text{-def fm-image-measurable-finite sets-}P\ J(\mathfrak{J})$ )
  qed simp
  finally have  $*$ :  $\text{emeasure } (P\ (J\ n))\ (B\ n) = ?SUP\ n$  .
  have  $?SUP\ n \neq \infty$ 
    unfolding  $*[\text{symmetric}]$  by  $\text{simp}$ 
  note  $*$   $\text{this}$ 
} note  $R = \text{this}$ 
have  $\forall n. \exists K. \text{emeasure } (P\ (J\ n))\ (B\ n) - \text{emeasure } (P'\ n)\ K \leq 2\ \text{powr } (-n)$ 
 $*$   $?a \wedge \text{compact } K \wedge K \subseteq \text{fm } n\ ' B\ n$ 
proof
  fix  $n$  show  $\exists K. \text{emeasure } (P\ (J\ n))\ (B\ n) - \text{emeasure } (P'\ n)\ K \leq \text{ennreal } (2\ \text{powr } - \text{real } n) * ?a \wedge$ 
     $\text{compact } K \wedge K \subseteq \text{fm } n\ ' B\ n$ 
    unfolding  $R[\text{of } n]$ 
    proof ( $\text{rule ccontr}$ )
      assume  $H$ :  $\nexists K'. ?SUP\ n - \text{emeasure } (P'\ n)\ K' \leq \text{ennreal } (2\ \text{powr } - \text{real } n) * ?a \wedge$ 
         $\text{compact } K' \wedge K' \subseteq \text{fm } n\ ' B\ n$ 
      have  $?SUP\ n + 0 < ?SUP\ n + 2\ \text{powr } (-n) * ?a$ 

```



```

    using R[of n] unfolding ennreal-add-left-cancel-less ennreal-zero-less-mult-iff
    by (auto intro: ‹0 < ?a›)
    also have ... = (SUP K ∈ {K. K ⊆ fm n ‘ B n ∧ compact K}. emeasure (P'
n) K + 2 powr (−n) * ?a)
    by (rule ennreal-SUP-add-left[symmetric]) auto
    also have ... ≤ ?SUP n
    proof (intro SUP-least)
      fix K assume K ∈ {K. K ⊆ fm n ‘ B n ∧ compact K}
      with H have 2 powr (−n) * ?a < ?SUP n − emeasure (P' n) K
      by auto
      then show emeasure (P' n) K + (2 powr (−n)) * ?a ≤ ?SUP n
      by (subst (asm) less-diff-eq-ennreal) (auto simp: less-top[symmetric])
R(1)[symmetric] ac-simps)
    qed
    finally show False by simp
  qed
qed
then obtain K' where K':
  ∧n. emeasure (P (J n)) (B n) − emeasure (P' n) (K' n) ≤ ennreal (2 powr −
real n) * ?a
  ∧n. compact (K' n) ∧n. K' n ⊆ fm n ‘ B n
  unfolding choice-iff by blast
define K where K n = fm n − ‘ K' n ∩ space (Pi_M (J n) (λ-. borel)) for n
have K-sets: ∧n. K n ∈ sets (Pi_M (J n) (λ-. borel))
  unfolding K-def
  using compact-imp-closed[OF ‹compact (K' -)›]
  by (intro measurable-sets[OF fm-measurable, of - Collect finite])
    (auto simp: borel-eq-PiF-borel[symmetric])
have K-B: ∧n. K n ⊆ B n
proof
  fix x n assume x ∈ K n
  then have fm-in: fm n x ∈ fm n ‘ B n
    using K' by (force simp: K-def)
  show x ∈ B n
    using ‹x ∈ K n› K-sets sets.sets-into-space J(1,2,3)[of n] inj-on-image-mem-iff[OF
inj-on-fm]
    by (metis (no-types) Int-iff K-def fm-in space-borel)
  qed
define Z' where Z' n = emb I (J n) (K n) for n
have Z': ∧n. Z' n ⊆ Z n
  unfolding Z'-def Z-def
proof (rule prod-emb-mono, safe)
  fix n x assume x ∈ K n
  hence fm n x ∈ K' n x ∈ space (Pi_M (J n) (λ-. borel))
    by (simp-all add: K-def space-P)
  note this(1)
  also have K' n ⊆ fm n ‘ B n by (simp add: K')
  finally have fm n x ∈ fm n ‘ B n .
  thus x ∈ B n

```

```

proof safe
  fix  $y$  assume  $y: y \in B\ n$ 
  hence  $y \in \text{space } (Pi_M\ (J\ n)\ (\lambda\cdot. \text{borel}))$  using  $J\ \text{sets.sets-into-space}[of\ B\ n\ P\ (J\ n)]$ 
  by  $(\text{auto simp add: space-}P\ \text{sets-}P)$ 
  assume  $fm\ n\ x = fm\ n\ y$ 
  note  $\text{inj-onD}[OF\ \text{inj-on-fm}[OF\ \text{space-borel}],$ 
     $OF\ \langle fm\ n\ x = fm\ n\ y \rangle\ \langle x \in \text{space} \rightarrow \langle y \in \text{space} \rightarrow \rangle]$ 
  with  $y$  show  $x \in B\ n$  by simp
qed
qed
have  $\bigwedge n. Z'\ n \in \text{generator}$  using  $J\ K'(\mathcal{Q})$  unfolding  $Z'\text{-def}$ 
  by  $(\text{auto intro!: generator.intros measurable-sets}[OF\ fm\text{-measurable}[of\ -\ \text{Collect}\ \text{finite}]])$ 
   $\text{simp: } K\text{-def borel-eq-PiF-borel[symmetric] compact-imp-closed)$ 
define  $Y$  where  $Y\ n = (\bigcap_{i \in \{1..n\}}. Z'\ i)$  for  $n$ 
hence  $\bigwedge n\ k. Y\ (n + k) \subseteq Y\ n$  by  $(\text{induct-tac } k)\ (\text{auto simp: } Y\text{-def})$ 
hence  $Y\text{-mono: } \bigwedge n\ m. n \leq m \implies Y\ m \subseteq Y\ n$  by  $(\text{auto simp: le-iff-add})$ 
have  $Y\text{-}Z': \bigwedge n. n \geq 1 \implies Y\ n \subseteq Z'\ n$  by  $(\text{auto simp: } Y\text{-def})$ 
hence  $Y\text{-}Z: \bigwedge n. n \geq 1 \implies Y\ n \subseteq Z\ n$  using  $Z'$  by auto

have  $Y\text{-notempty: } \bigwedge n. n \geq 1 \implies (Y\ n) \neq \{\}$ 
proof  $-$ 
  fix  $n::nat$  assume  $n \geq 1$  hence  $Y\ n \subseteq Z\ n$  by fact
  have  $Y\ n = (\bigcap_{i \in \{1..n\}}. \text{emb } I\ (J\ n)\ (\text{emb } (J\ n)\ (J\ i)\ (K\ i)))$  using  $J\ J\text{-mono}$ 
    by  $(\text{auto simp: } Y\text{-def } Z'\text{-def})$ 
  also have  $\dots = \text{prod-emb } I\ (\lambda\cdot. \text{borel})\ (J\ n)\ (\bigcap_{i \in \{1..n\}}. \text{emb } (J\ n)\ (J\ i)\ (K\ i))$ 
    using  $\langle n \geq 1 \rangle$ 
    by  $(\text{subst prod-emb-INT})$  auto
  finally
  have  $Y\text{-emb:}$ 
     $Y\ n = \text{prod-emb } I\ (\lambda\cdot. \text{borel})\ (J\ n)\ (\bigcap_{i \in \{1..n\}}. \text{prod-emb } (J\ n)\ (\lambda\cdot. \text{borel})\ (J\ i)\ (K\ i))$  .
  hence  $Y\ n \in \text{generator}$  using  $J\ J\text{-mono } K\text{-sets } \langle n \geq 1 \rangle$ 
    by  $(\text{auto simp del: prod-emb-INT intro!: generator.intros})$ 
  have  $*$ :  $\mu G\ (Z\ n) = P\ (J\ n)\ (B\ n)$ 
    unfolding  $Z\text{-def}$  using  $J$  by  $(\text{intro mu-G-spec})$  auto
  then have  $\mu G\ (Z\ n) \neq \infty$  by auto
  note  $*$ 
  moreover have  $*$ :  $\mu G\ (Y\ n) = P\ (J\ n)\ (\bigcap_{i \in \{Suc\ 0..n\}}. \text{prod-emb } (J\ n)\ (\lambda\cdot. \text{borel})\ (J\ i)\ (K\ i))$ 
    unfolding  $Y\text{-emb}$  using  $J\ J\text{-mono } K\text{-sets } \langle n \geq 1 \rangle$  by  $(\text{subst mu-G-spec})$  auto
  then have  $\mu G\ (Y\ n) \neq \infty$  by auto
  note  $*$ 
  moreover have  $\mu G\ (Z\ n - Y\ n) =$ 
     $P\ (J\ n)\ (B\ n - (\bigcap_{i \in \{Suc\ 0..n\}}. \text{prod-emb } (J\ n)\ (\lambda\cdot. \text{borel})\ (J\ i)\ (K\ i)))$ 
    unfolding  $Z\text{-def } Y\text{-emb prod-emb-Diff[symmetric]}$  using  $J\ J\text{-mono } K\text{-sets } \langle n \geq 1 \rangle$ 

```

by (subst mu-G-spec) (auto intro!: sets.Diff)
 ultimately
 have $\mu G (Z\ n) - \mu G (Y\ n) = \mu G (Z\ n - Y\ n)$
 using $J\ J\text{-mono}\ K\text{-sets}\ \langle n \geq 1 \rangle$
 by (simp only: emeasure-eq-measure Z-def)
 (auto dest!: bspec[**where** $x=n$] intro!: measure-Diff[symmetric] subsetD[OF
 K-B]
 intro!: arg-cong[**where** $f=ennreal$]
 simp: extensional-restrict emeasure-eq-measure prod-emb-iff sets-P
 space-P
 ennreal-minus measure-nonneg)
 also have subs: $Z\ n - Y\ n \subseteq (\bigcup_{i \in \{1..n\}} (Z\ i - Z'\ i))$
 using $\langle n \geq 1 \rangle$ unfolding Y-def UN-extend-simps(7) by (intro UN-mono
 Diff-mono Z-mono order-refl) auto
 have $Z\ n - Y\ n \in \text{generator } (\bigcup_{i \in \{1..n\}} (Z\ i - Z'\ i)) \in \text{generator}$
 using $\langle Z' - \in \text{generator} \rangle \langle Z - \in \text{generator} \rangle \langle Y - \in \text{generator} \rangle$ by auto
 hence $\mu G (Z\ n - Y\ n) \leq \mu G (\bigcup_{i \in \{1..n\}} (Z\ i - Z'\ i))$
 using subs generator.additive-increasing[OF positive-mu-G additive-mu-G]
 unfolding increasing-def by auto
 also have $\dots \leq (\sum_{i \in \{1..n\}} \mu G (Z\ i - Z'\ i))$ using $\langle Z - \in \text{generator} \rangle \langle Z' - \in \text{generator} \rangle$
 by (intro generator.subadditive[OF positive-mu-G additive-mu-G]) auto
 also have $\dots \leq (\sum_{i \in \{1..n\}} 2^{\text{powr } -\text{real } i} * ?a)$
 proof (rule sum-mono)
 fix i assume $i \in \{1..n\}$ hence $i \leq n$ by simp
 have $\mu G (Z\ i - Z'\ i) = \mu G (\text{prod-emb } I\ (\lambda\ -. \text{ borel})\ (J\ i)\ (B\ i - K\ i))$
 unfolding Z'-def Z-def by simp
 also have $\dots = P\ (J\ i)\ (B\ i - K\ i)$
 using J K-sets by (subst mu-G-spec) auto
 also have $\dots = P\ (J\ i)\ (B\ i) - P\ (J\ i)\ (K\ i)$
 using K-sets J $\langle K - \subseteq B \rangle$ by (simp add: emeasure-Diff)
 also have $\dots = P\ (J\ i)\ (B\ i) - P'\ i\ (K'\ i)$
 unfolding K-def P'-def
 by (auto simp: mapmeasure-PiF borel-eq-PiF-borel[symmetric]
 compact-imp-closed[OF $\langle \text{compact } (K'\ -) \rangle$] space-PiM PiE-def)
 also have $\dots \leq \text{ennreal } (2^{\text{powr } -\text{real } i}) * ?a$ using $K'(1)[\text{of } i]$.
 finally show $\mu G (Z\ i - Z'\ i) \leq (2^{\text{powr } -\text{real } i}) * ?a$.
 qed
 also have $\dots = \text{ennreal } ((\sum_{i \in \{1..n\}} (2^{\text{powr } -\text{enn2real } i})) * \text{enn2real } ?a)$
 using r by (simp add: sum-distrib-right ennreal-mult[symmetric])
 also have $\dots < \text{ennreal } (1 * \text{enn2real } ?a)$
 proof (intro ennreal-lessI mult-strict-right-mono)
 have $(\sum_{i = 1..n} 2^{\text{powr } -\text{real } i}) = (\sum_{i = 1..<\text{Suc } n} (1/2)^i)$
 by (rule sum.cong) (auto simp: powr-realpow powr-divide power-divide
 powr-minus-divide)
 also have $\{1..<\text{Suc } n\} = \{..<\text{Suc } n\} - \{0\}$ by auto
 also have $\text{sum } ((\bigwedge) (1 / 2::\text{real})) (\{..<\text{Suc } n\} - \{0\}) =$
 $\text{sum } ((\bigwedge) (1 / 2)) (\{..<\text{Suc } n\}) - 1$ by (auto simp: sum-diff1)
 also have $\dots < 1$ by (subst geometric-sum) auto

finally show $(\sum i = 1..n. 2 \text{ powr} - \text{enn2real } i) < 1$ by simp
 qed (auto simp: r enn2real-positive-iff)
 also have $\dots = ?a$ by (auto simp: r)
 also have $\dots \leq \mu G (Z \ n)$
 using J by (auto intro: INF-lower simp: Z-def mu-G-spec)
 finally have $\mu G (Z \ n) - \mu G (Y \ n) < \mu G (Z \ n)$.
 hence R: $\mu G (Z \ n) < \mu G (Z \ n) + \mu G (Y \ n)$
 using $\langle \mu G (Y \ n) \neq \infty \rangle$ by (auto simp: zero-less-iff-neq-zero)
 then have $\mu G (Y \ n) > 0$
 by simp
 thus $Y \ n \neq \{\}$ using positive-mu-G by (auto simp add: positive-def)
 qed
 hence $\forall n \in \{1..\}. \exists y. y \in Y \ n$ by auto
 then obtain y where $y: \bigwedge n. n \geq 1 \implies y \ n \in Y \ n$ unfolding bchoice-iff by
 force
 {
 fix t and n m::nat
 assume $1 \leq n \ n \leq m$ hence $1 \leq m$ by simp
 from Y-mono[OF $\langle m \geq n \rangle$] y[OF $\langle 1 \leq m \rangle$] have $y \ m \in Y \ n$ by auto
 also have $\dots \subseteq Z' \ n$ using Y-Z'[OF $\langle 1 \leq n \rangle$] .
 finally
 have $\text{fm } n (\text{restrict } (y \ m) (J \ n)) \in K' \ n$
 unfolding Z'-def K-def prod-emb-iff by (simp add: Z'-def K-def prod-emb-iff)
 moreover have $\text{finmap-of } (J \ n) (\text{restrict } (y \ m) (J \ n)) = \text{finmap-of } (J \ n) (y$
 m)
 using J by (simp add: fm-def)
 ultimately have $\text{fm } n (y \ m) \in K' \ n$ by simp
 } note fm-in-K' = this
 interpret finmap-seqs-into-compact $\lambda n. K' (Suc \ n) \ \lambda k. \text{fm } (Suc \ k) (y \ (Suc \ k))$
 borel
 proof
 fix n show compact $(K' \ n)$ by fact
 next
 fix n
 from Y-mono[of n Suc n] y[of Suc n] have $y \ (Suc \ n) \in Y \ (Suc \ n)$ by auto
 also have $\dots \subseteq Z' (Suc \ n)$ using Y-Z' by auto
 finally
 have $\text{fm } (Suc \ n) (\text{restrict } (y \ (Suc \ n)) (J \ (Suc \ n))) \in K' (Suc \ n)$
 unfolding Z'-def K-def prod-emb-iff by (simp add: Z'-def K-def prod-emb-iff)
 thus $K' (Suc \ n) \neq \{\}$ by auto
 fix k
 assume $k \in K' (Suc \ n)$
 with K'[of Suc n] sets.sets-into-space have $k \in \text{fm } (Suc \ n) \text{ ` } B \ (Suc \ n)$ by
 auto
 then obtain b where $k = \text{fm } (Suc \ n) \ b$ by auto
 thus $\text{domain } k = \text{domain } (\text{fm } (Suc \ n) (y \ (Suc \ n)))$
 by (simp-all add: fm-def)
 next
 fix t and n m::nat

```

assume  $n \leq m$  hence  $Suc\ n \leq Suc\ m$  by simp
assume  $t \in domain\ (fm\ (Suc\ n)\ (y\ (Suc\ n)))$ 
then obtain  $j$  where  $j: t = Utn\ j\ j \in J\ (Suc\ n)$  by auto
hence  $j \in J\ (Suc\ m)$  using  $J\text{-mono}[OF\ \langle Suc\ n \leq Suc\ m \rangle]$  by auto
have  $img: fm\ (Suc\ n)\ (y\ (Suc\ m)) \in K'\ (Suc\ n)$  using  $\langle n \leq m \rangle$ 
by (intro fm-in-K') simp-all
show  $(fm\ (Suc\ m)\ (y\ (Suc\ m)))_F\ t \in (\lambda k. (k)_F\ t)\ 'K'\ (Suc\ n)$ 
apply (rule image-eqI[OF - img])
using  $\langle j \in J\ (Suc\ n) \rangle\ \langle j \in J\ (Suc\ m) \rangle$ 
unfolding  $j$  by (subst proj-fm, auto)+
qed
have  $\forall t. \exists z. (\lambda i. (fm\ (Suc\ (diagseq\ i))\ (y\ (Suc\ (diagseq\ i))))_F\ t) \longrightarrow z$ 
using diagonal-tendsto ..
then obtain  $z$  where  $z:$ 
 $\bigwedge t. (\lambda i. (fm\ (Suc\ (diagseq\ i))\ (y\ (Suc\ (diagseq\ i))))_F\ t) \longrightarrow z\ t$ 
unfolding choice-iff by blast
{
  fix  $n :: nat$  assume  $n \geq 1$ 
  have  $\bigwedge i. domain\ (fm\ n\ (y\ (Suc\ (diagseq\ i)))) = domain\ (finmap-of\ (Utn\ 'J\ n)\ z)$ 
  by simp
  moreover
  {
    fix  $t$ 
    assume  $t: t \in domain\ (finmap-of\ (Utn\ 'J\ n)\ z)$ 
    hence  $t \in Utn\ 'J\ n$  by simp
    then obtain  $j$  where  $j: t = Utn\ j\ j \in J\ n$  by auto
    have  $(\lambda i. (fm\ n\ (y\ (Suc\ (diagseq\ i))))_F\ t) \longrightarrow z\ t$ 
    apply (subst (2) tendsto-iff, subst eventually-sequentially)
    proof safe
      fix  $e :: real$  assume  $0 < e$ 
      { fix  $i$  and  $x :: 'i \Rightarrow 'a$  assume  $i: i \geq n$ 
        assume  $t \in domain\ (fm\ n\ x)$ 
        hence  $t \in domain\ (fm\ i\ x)$  using  $J\text{-mono}[OF\ \langle i \geq n \rangle]$  by auto
        with  $i$  have  $(fm\ i\ x)_F\ t = (fm\ n\ x)_F\ t$ 
        using  $j$  by (auto simp: proj-fm dest!: inj-onD[OF inj-on-Utn])
      } note index-shift = this
      have  $I: \bigwedge i. i \geq n \implies Suc\ (diagseq\ i) \geq n$ 
      apply (rule le-SucI)
      apply (rule order-trans) apply simp
      apply (rule seq-suble[OF subseq-diagseq])
      done
    from  $z$ 
    have  $\exists N. \forall i \geq N. dist\ ((fm\ (Suc\ (diagseq\ i))\ (y\ (Suc\ (diagseq\ i))))_F\ t)\ (z\ t) < e$ 
    unfolding tendsto-iff eventually-sequentially using  $\langle 0 < e \rangle$  by auto
    then obtain  $N$  where  $N: \bigwedge i. i \geq N \implies$ 
 $dist\ ((fm\ (Suc\ (diagseq\ i))\ (y\ (Suc\ (diagseq\ i))))_F\ t)\ (z\ t) < e$  by auto
    show  $\exists N. \forall na \geq N. dist\ ((fm\ n\ (y\ (Suc\ (diagseq\ na))))_F\ t)\ (z\ t) < e$ 

```

```

proof (rule exI[where  $x = \max N\ n$ ], safe)
  fix na assume  $\max N\ n \leq na$ 
  hence  $\text{dist } ((\text{fm } n\ (y\ (\text{Suc } (\text{diagseq } na))))_F\ t)\ (z\ t) =$ 
     $\text{dist } ((\text{fm } (\text{Suc } (\text{diagseq } na))\ (y\ (\text{Suc } (\text{diagseq } na))))_F\ t)\ (z\ t)$  using  $t$ 
    by (subst index-shift[OF I]) auto
  also have  $\dots < e$  using  $\langle \max N\ n \leq na \rangle$  by (intro N) simp
  finally show  $\text{dist } ((\text{fm } n\ (y\ (\text{Suc } (\text{diagseq } na))))_F\ t)\ (z\ t) < e$  .
qed
qed
  hence  $(\lambda i. (\text{fm } n\ (y\ (\text{Suc } (\text{diagseq } i))))_F\ t) \longrightarrow (\text{finmap-of } (Utn\ 'J\ n)\ z)_F\ t$ 
    by (simp add: tendsto-intros)
} ultimately
  have  $(\lambda i. \text{fm } n\ (y\ (\text{Suc } (\text{diagseq } i)))) \longrightarrow \text{finmap-of } (Utn\ 'J\ n)\ z$ 
    by (rule tendsto-finmap)
  hence  $((\lambda i. \text{fm } n\ (y\ (\text{Suc } (\text{diagseq } i))))\ o\ (\lambda i. i + n)) \longrightarrow \text{finmap-of } (Utn\ 'J\ n)\ z$ 
    by (rule LIMSEQ-subseq-LIMSEQ) (simp add: strict-mono-def)
moreover
  have  $(\forall i. ((\lambda i. \text{fm } n\ (y\ (\text{Suc } (\text{diagseq } i))))\ o\ (\lambda i. i + n))\ i \in K'\ n)$ 
    apply (auto simp add: o-def intro!: fm-in-K'  $\langle 1 \leq n \rangle$  le-SucI)
    apply (rule le-trans)
    apply (rule le-add2)
    using seq-suble[OF subseq-diagseq]
    apply auto
  done
moreover
from  $\langle \text{compact } (K'\ n) \rangle$  have  $\text{closed } (K'\ n)$  by (rule compact-imp-closed)
ultimately
have  $\text{finmap-of } (Utn\ 'J\ n)\ z \in K'\ n$ 
    unfolding closed-sequential-limits by blast
also have  $\text{finmap-of } (Utn\ 'J\ n)\ z = \text{fm } n\ (\lambda i. z\ (Utn\ i))$ 
    unfolding finmap-eq-iff
proof clarsimp
  fix i assume  $i \in J\ n$ 
  hence  $\text{from-nat-into } (\bigcup n. J\ n)\ (Utn\ i) = i$ 
    unfolding Utn-def
    by (subst from-nat-into-to-nat-on[OF countable-UN-J]) auto
  with i show  $z\ (Utn\ i) = (\text{fm } n\ (\lambda i. z\ (Utn\ i)))_F\ (Utn\ i)$ 
    by (simp add: finmap-eq-iff fm-def compose-def)
qed
finally have  $\text{fm } n\ (\lambda i. z\ (Utn\ i)) \in K'\ n$  .
moreover
let  $?J = \bigcup n. J\ n$ 
have  $(?J \cap J\ n) = J\ n$  by auto
ultimately have  $\text{restrict } (\lambda i. z\ (Utn\ i))\ (?J \cap J\ n) \in K\ n$ 
    unfolding K-def by (auto simp: space-P space-PiM)
hence  $\text{restrict } (\lambda i. z\ (Utn\ i))\ ?J \in Z'\ n$  unfolding Z'-def
    using J by (auto simp: prod-emb-def PiE-def extensional-def)

```

```

    also have ...  $\subseteq Z\ n$  using  $Z'$  by simp
    finally have restrict  $(\lambda i. z\ (Utn\ i))\ ?J \in Z\ n$  .
  } note in-Z = this
  hence  $(\bigcap_{i \in \{1.. \}}. Z\ i) \neq \{\}$  by auto
  thus  $(\bigcap i. Z\ i) \neq \{\}$ 
  using INT-decseq-offset[OF antimonoI[OF Z-mono]] by simp
qed fact+

```

lemma *measure-lim-emb*:

```

   $J \subseteq I \implies \text{finite } J \implies X \in \text{sets } (\Pi_M\ i \in J. \text{borel}) \implies \text{measure lim } (\text{emb } I\ J\ X)$ 
= measure  $(P\ J)\ X$ 
  unfolding measure-def by (subst emeasure-lim-emb) auto

```

end

```

hide-const (open)  $PiF$ 
hide-const (open)  $Pi_F$ 
hide-const (open)  $Pi'$ 
hide-const (open) finmap-of
hide-const (open) proj
hide-const (open) domain
hide-const (open) basis-finmap

```

sublocale *polish-projective* $\subseteq P$: *prob-space lim*

proof

```

  have *:  $\text{emb } I\ \{\} \ \{\lambda x. \text{undefined}\} = \text{space } (\Pi_M\ i \in I. \text{borel})$ 
  by (auto simp: prod-emb-def space-PiM)
  interpret prob-space  $P\ \{\}$ 
  using prob-space-P by simp
  show emeasure lim  $(\text{space lim}) = 1$ 
  using emeasure-lim-emb[of  $\{\} \ \{\lambda x. \text{undefined}\}$ ] emeasure-space-1
  by (simp add: * PiM-empty space-P)
qed

```

locale *polish-product-prob-space* =

product-prob-space $\lambda -. \text{borel}::('a::\text{polish-space})\ \text{measure } I\ \text{for } I::'i\ \text{set}$

sublocale *polish-product-prob-space* $\subseteq P$: *polish-projective* $I\ \lambda J. PiM\ J\ (\lambda -. \text{borel}::('a)\ \text{measure})$

..

```

lemma (in polish-product-prob-space) limP-eq-PiM:  $\text{lim} = PiM\ I\ (\lambda -. \text{borel})$ 
  by (rule PiM-eq) (auto simp: emeasure-PiM emeasure-lim-emb)

```

end

24 Random Permutations

theory *Random-Permutations*

```

imports
  HOL-Combinatorics.Multiset-Permutations
  Probability-Mass-Function
begin

```

Choosing a set permutation (i.e. a distinct list with the same elements as the set) uniformly at random is the same as first choosing the first element of the list and then choosing the rest of the list as a permutation of the remaining set.

lemma *random-permutation-of-set*:

```

assumes finite A A ≠ {}
shows pmf-of-set (permutations-of-set A) =
  do {
    x ← pmf-of-set A;
    xs ← pmf-of-set (permutations-of-set (A - {x}));
    return-pmf (x#xs)
  } (is ?lhs = ?rhs)

```

proof –

```

from assms have permutations-of-set A = (⋃ x∈A. (#) x ‘ permutations-of-set
(A - {x}))
  by (simp add: permutations-of-set-nonempty)
also from assms have pmf-of-set ... = ?rhs
  by (subst pmf-of-set-UN[where n = fact (card A - 1)])
    (auto simp: card-image disjoint-family-on-def map-pmf-def [symmetric]
map-pmf-of-set-inj)
finally show ?thesis .
qed

```

A generic fold function that takes a function, an initial state, and a set and chooses a random order in which it then traverses the set in the same fashion as a left fold over a list. We first give a recursive definition.

function *fold-random-permutation* :: ('a ⇒ 'b ⇒ 'b) ⇒ 'b ⇒ 'a set ⇒ 'b pmf
where

```

  fold-random-permutation f x {} = return-pmf x
| ¬finite A ⇒ fold-random-permutation f x A = return-pmf x
| finite A ⇒ A ≠ {} ⇒
  fold-random-permutation f x A =
    pmf-of-set A ≫ (λa. fold-random-permutation f (f a x) (A - {a}))
by simp-all fastforce

```

termination proof (relation Wellfounded.measure (λ(-, -, A). card A))

```

fix A :: 'a set and f :: 'a ⇒ 'b ⇒ 'b and x :: 'b and y :: 'a
assume A: finite A A ≠ {} y ∈ set-pmf (pmf-of-set A)
then have card A > 0 by (simp add: card-gt-0-iff)
with A show ((f, f y x, A - {y}), f, x, A) ∈ Wellfounded.measure (λ(-, -, A).
card A)
  by simp
qed simp-all

```

We can now show that the above recursive definition is equivalent to choosing

a random set permutation and folding over it (in any direction).

lemma *fold-random-permutation-foldl*:

assumes *finite A*

shows *fold-random-permutation f x A =*

map-pmf (foldl ($\lambda x y. f y x$) x) (pmf-of-set (permutations-of-set A))

using *assms*

proof (*induction f x A rule: fold-random-permutation.induct [case-names empty infinite remove]*)

case (*remove A f x*)

from *remove*

have *fold-random-permutation f x A =*

pmf-of-set A \gg ($\lambda a. fold-random-permutation f (f a x) (A - \{a\})$) **by**

simp

also from *remove*

have *... = pmf-of-set A \gg ($\lambda a. map-pmf (foldl (\lambda x y. f y x) x$*

(map-pmf ((#) a) (pmf-of-set (permutations-of-set (A - {a}))))

by (*intro bind-pmf-cong*) (*simp-all add: pmf.map-comp o-def*)

also from *remove* **have** *... = map-pmf (foldl ($\lambda x y. f y x$) x) (pmf-of-set (permutations-of-set A))*

by (*simp-all add: random-permutation-of-set map-bind-pmf map-pmf-def [symmetric]*)

finally show *?case .*

qed (*simp-all add: pmf-of-set-singleton*)

lemma *fold-random-permutation-foldr*:

assumes *finite A*

shows *fold-random-permutation f x A =*

map-pmf ($\lambda xs. foldr f xs x$) (pmf-of-set (permutations-of-set A))

proof –

have *fold-random-permutation f x A =*

map-pmf (foldl ($\lambda x y. f y x$) x \circ rev) (pmf-of-set (permutations-of-set A))

using *assms* **by** (*subst fold-random-permutation-foldl [OF assms]*)

(simp-all add: pmf.map-comp [symmetric] map-pmf-of-set-inj)

also have *foldl ($\lambda x y. f y x$) x \circ rev = ($\lambda xs. foldr f xs x$)*

by (*intro ext*) (*simp add: foldl-conv-foldr*)

finally show *?thesis .*

qed

lemma *fold-random-permutation-fold*:

assumes *finite A*

shows *fold-random-permutation f x A =*

map-pmf ($\lambda xs. fold f xs x$) (pmf-of-set (permutations-of-set A))

by (*subst fold-random-permutation-foldl [OF assms], intro map-pmf-cong*)

(simp-all add: foldl-conv-fold)

lemma *fold-random-permutation-code* [*code*]:

fold-random-permutation f x (set xs) =

map-pmf (foldl ($\lambda x y. f y x$) x) (pmf-of-set (permutations-of-set (set xs)))

by (*simp add: fold-random-permutation-foldl*)

We now introduce a slightly generalised version of the above fold operation that does not simply return the result in the end, but applies a monadic bind to it. This may seem somewhat arbitrary, but it is a common use case, e.g. in the Social Decision Scheme of Random Serial Dictatorship, where voters narrow down a set of possible winners in a random order and the winner is chosen from the remaining set uniformly at random.

```

function fold-bind-random-permutation
  :: ('a ⇒ 'b ⇒ 'b) ⇒ ('b ⇒ 'c pmf) ⇒ 'b ⇒ 'a set ⇒ 'c pmf where
    fold-bind-random-permutation f g x {} = g x
| ¬finite A ⇒⇒ fold-bind-random-permutation f g x A = g x
| finite A ⇒⇒ A ≠ {} ⇒⇒
    fold-bind-random-permutation f g x A =
      pmf-of-set A ≫ (λa. fold-bind-random-permutation f g (f a x) (A - {a}))
by simp-all fastforce
termination proof (relation Wellfounded.measure (λ(-,-,-,A). card A))
  fix A :: 'a set and f :: 'a ⇒ 'b ⇒ 'b and x :: 'b
    and y :: 'a and g :: 'b ⇒ 'c pmf
  assume A: finite A A ≠ {} y ∈ set-pmf (pmf-of-set A)
  then have card A > 0 by (simp add: card-gt-0-iff)
  with A show ((f, g, f y x, A - {y}), f, g, x, A) ∈ Wellfounded.measure (λ(-, -, -, A). card A)
  by simp
qed simp-all

```

We now show that the recursive definition is equivalent to a random fold followed by a monadic bind.

```

lemma fold-bind-random-permutation-altdef [code]:
  fold-bind-random-permutation f g x A = fold-random-permutation f x A ≫ g
proof (induction f x A rule: fold-random-permutation.induct [case-names empty
infinite remove])
  case (remove A f x)
  from remove have pmf-of-set A ≫ (λa. fold-bind-random-permutation f g (f a
x) (A - {a})) =
    pmf-of-set A ≫ (λa. fold-random-permutation f (f a x) (A -
{a}) ≫ g)
  by (intro bind-pmf-cong) simp-all
  with remove show ?case by (simp add: bind-return-pmf bind-assoc-pmf)
qed (simp-all add: bind-return-pmf)

```

We can now derive the following nice monadic representations of the combined fold-and-bind:

```

lemma fold-bind-random-permutation-foldl:
  assumes finite A
  shows fold-bind-random-permutation f g x A =
    do {xs ← pmf-of-set (permutations-of-set A); g (foldl (λx y. f y x) x xs)}
  using assms by (simp add: fold-bind-random-permutation-altdef bind-assoc-pmf
fold-random-permutation-foldl bind-return-pmf map-pmf-def)

```

lemma *fold-bind-random-permutation-foldr*:

assumes *finite A*

shows $\text{fold-bind-random-permutation } f \ g \ x \ A =$

$\text{do } \{xs \leftarrow \text{pmf-of-set } (\text{permutations-of-set } A); g \ (\text{foldr } f \ xs \ x)\}$

using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-foldr *bind-return-pmf* *map-pmf-def*)

lemma *fold-bind-random-permutation-fold*:

assumes *finite A*

shows $\text{fold-bind-random-permutation } f \ g \ x \ A =$

$\text{do } \{xs \leftarrow \text{pmf-of-set } (\text{permutations-of-set } A); g \ (\text{fold } f \ xs \ x)\}$

using *assms* **by** (*simp* *add*: *fold-bind-random-permutation-altdef* *bind-assoc-pmf*
fold-random-permutation-fold *bind-return-pmf* *map-pmf-def*)

The following useful lemma allows us to swap partitioning a set w.r.t. a predicate and drawing a random permutation of that set.

lemma *partition-random-permutations*:

assumes *finite A*

shows $\text{map-pmf } (\text{partition } P) \ (\text{pmf-of-set } (\text{permutations-of-set } A)) =$

$\text{pair-pmf } (\text{pmf-of-set } (\text{permutations-of-set } \{x \in A. P \ x\}))$

$(\text{pmf-of-set } (\text{permutations-of-set } \{x \in A. \neg P \ x\})) \ (\text{is } ?lhs = ?rhs)$

proof (*rule* *pmf-eqI*, *clarify*, *goal-cases*)

case (*1 xs ys*)

show *?case*

proof (*cases* $xs \in \text{permutations-of-set } \{x \in A. P \ x\} \wedge ys \in \text{permutations-of-set } \{x \in A. \neg P \ x\}$)

case *True*

let *?n1* = *card* $\{x \in A. P \ x\}$ **and** *?n2* = *card* $\{x \in A. \neg P \ x\}$

have *card-eq*: *card* *A* = *?n1* + *?n2*

proof –

have *?n1* + *?n2* = *card* $(\{x \in A. P \ x\} \cup \{x \in A. \neg P \ x\})$

using *assms* **by** (*intro* *card-Un-disjoint* [*symmetric*]) *auto*

also have $\{x \in A. P \ x\} \cup \{x \in A. \neg P \ x\} = A$ **by** *blast*

finally show *?thesis* ..

qed

from *True* **have** *lengths* [*simp*]: *length* *xs* = *?n1* *length* *ys* = *?n2*

by (*auto* *intro!*: *length-finite-permutations-of-set*)

have *pmf* *?lhs* (*xs*, *ys*) =

$\text{real } (\text{card } (\text{permutations-of-set } A \cap \text{partition } P - \{(xs, ys)\})) / \text{fact}$

(*card* *A*)

using *assms* **by** (*auto* *simp*: *pmf-map* *measure-pmf-of-set*)

also have *partition* *P* – $\{(xs, ys)\} = \text{shuffles } xs \ ys$

using *True* **by** (*intro* *inv-image-partition*) (*auto* *simp*: *permutations-of-set-def*)

also have *permutations-of-set* $A \cap \text{shuffles } xs \ ys = \text{shuffles } xs \ ys$

using *True* *distinct-disjoint-shuffles*[*of* *xs* *ys*]

by (*auto* *simp*: *permutations-of-set-def* *dest*: *set-shuffles*)

also have *card* $(\text{shuffles } xs \ ys) = \text{length } xs + \text{length } ys$ *choose* *length* *xs*

using *True* **by** (*intro* *card-disjoint-shuffles*) (*auto* *simp*: *permutations-of-set-def*)

```

    also have  $\text{length } xs + \text{length } ys = \text{card } A$  by (simp add: card-eq)
    also have  $\text{real } (\text{card } A \text{ choose length } xs) = \text{fact } (\text{card } A) / (\text{fact } ?n1 * \text{fact } (\text{card } A - ?n1))$ 
      by (subst binomial-fact) (auto intro!: card-mono assms)
    also have  $\dots / \text{fact } (\text{card } A) = 1 / (\text{fact } ?n1 * \text{fact } ?n2)$ 
      by (simp add: field-split-simps card-eq)
    also have  $\dots = \text{pmf } ?rhs \ (xs, ys)$  using True assms by (simp add: pmf-pair)
    finally show ?thesis .
  next
    case False
    hence *:  $xs \notin \text{permutations-of-set } \{x \in A. P \ x\} \vee ys \notin \text{permutations-of-set } \{x \in A. \neg P \ x\}$  by blast
    hence eq:  $\text{permutations-of-set } A \cap (\text{partition } P - \{(xs, ys)\}) = \{\}$ 
      by (auto simp: o-def permutations-of-set-def)
    from * show ?thesis
      by (elim disjE) (insert assms eq, simp-all add: pmf-pair pmf-map measure-pmf-of-set)
  qed
qed
end

```

25 Discrete subprobability distribution

```

theory SPMF imports
  Probability-Mass-Function
  HOL-Library.Complete-Partial-Order2
  HOL-Library.Rewrite
begin

```

25.1 Auxiliary material

```

lemma cSUP-singleton [simp]:  $(\text{SUP } x \in \{x\}. f \ x :: - :: \text{conditionally-complete-lattice}) = f \ x$ 
  by (metis cSup-singleton image-empty image-insert)

```

25.1.1 More about extended reals

```

lemma [simp]:
  shows  $\text{ennreal-max-0}: \text{ennreal } (\max 0 \ x) = \text{ennreal } x$ 
    and  $\text{ennreal-max-0'}: \text{ennreal } (\max x \ 0) = \text{ennreal } x$ 
  by (simp-all add: max-def ennreal-eq-0-iff)

```

```

lemma e2ennreal-0 [simp]:  $e2ennreal \ 0 = 0$ 
  by (simp add: zero-ennreal-def)

```

```

lemma enn2real-bot [simp]:  $\text{enn2real } \perp = 0$ 
  by (simp add: bot-ennreal-def)

```

lemma *continuous-at-ennreal*[*continuous-intros*]: *continuous F f* \implies *continuous F*
 $(\lambda x. \text{ennreal } (f x))$

unfolding *continuous-def* **by** *auto*

lemma *ennreal-Sup*:

assumes *: $(\text{SUP } a \in A. \text{ennreal } a) \neq \top$

and $A \neq \{\}$

shows $\text{ennreal } (\text{Sup } A) = (\text{SUP } a \in A. \text{ennreal } a)$

proof (*rule continuous-at-Sup-mono*)

obtain r **where** $r: \text{ennreal } r = (\text{SUP } a \in A. \text{ennreal } a) \ r \geq 0$

using * **by**(*cases* $(\text{SUP } a \in A. \text{ennreal } a)$) *simp-all*

then show *bdd-above A*

by(*auto intro!*: *SUP-upper bdd-aboveI*[*of - r*] *simp add: ennreal-le-iff*[*symmetric*])

qed (*auto simp: mono-def continuous-at-imp-continuous-at-within continuous-at-ennreal ennreal-leI assms*)

lemma *ennreal-SUP*:

$\llbracket (\text{SUP } a \in A. \text{ennreal } (f a)) \neq \top; A \neq \{\} \rrbracket \implies \text{ennreal } (\text{SUP } a \in A. f a) = (\text{SUP } a \in A. \text{ennreal } (f a))$

using *ennreal-Sup*[*of f ' A*] **by** (*auto simp: image-comp*)

lemma *ennreal-lt-0*: $x < 0 \implies \text{ennreal } x = 0$

by(*simp add: ennreal-eq-0-iff*)

25.1.2 More about 'a option

lemma *None-in-map-option-image* [*simp*]: $\text{None} \in \text{map-option } f \text{ 'A} \longleftrightarrow \text{None} \in A$

by *auto*

lemma *Some-in-map-option-image* [*simp*]: $\text{Some } x \in \text{map-option } f \text{ 'A} \longleftrightarrow (\exists y. x = f y \wedge \text{Some } y \in A)$

by (*smt* (*verit, best*) *imageE imageI map-option-eq-Some*)

lemma *case-option-collapse*: $\text{case-option } x (\lambda \cdot. x) = (\lambda \cdot. x)$

by(*simp add: fun-eq-iff split: option.split*)

lemma *case-option-id*: $\text{case-option } \text{None } \text{Some} = \text{id}$

by(*rule ext*)(*simp split: option.split*)

inductive *ord-option* :: $('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \text{ option} \Rightarrow 'b \text{ option} \Rightarrow \text{bool}$

for *ord* :: $'a \Rightarrow 'b \Rightarrow \text{bool}$

where

None: *ord-option ord None x*

| *Some*: *ord x y* \implies *ord-option ord (Some x) (Some y)*

inductive-simps *ord-option-simps* [*simp*]:

ord-option ord None x

ord-option ord x None

ord-option ord (Some x) (Some y)
ord-option ord (Some x) None

inductive-simps *ord-option-eq-simps* [simp]:

ord-option (=) None y
ord-option (=) (Some x) y

lemma *ord-option-reflI*: $(\bigwedge y. y \in \text{set-option } x \implies \text{ord } y \ y) \implies \text{ord-option ord } x$

by(cases x) simp-all

lemma *reflp-ord-option*: *reflp ord* \implies *reflp (ord-option ord)*

by(simp add: reflp-def ord-option-reflI)

lemma *ord-option-trans*:

$\llbracket \text{ord-option ord } x \ y; \text{ord-option ord } y \ z; \bigwedge a \ b \ c. \llbracket a \in \text{set-option } x; b \in \text{set-option } y; c \in \text{set-option } z; \text{ord } a \ b; \text{ord } b \ c \rrbracket \implies \text{ord } a \ c \rrbracket$
 $\implies \text{ord-option ord } x \ z$
by(auto elim!: ord-option.cases)

lemma *transp-ord-option*: *transp ord* \implies *transp (ord-option ord)*

unfolding *transp-def* **by**(blast intro: ord-option-trans)

lemma *antisymp-ord-option*: *antisymp ord* \implies *antisymp (ord-option ord)*

by(auto intro!: antisympI elim!: ord-option.cases dest: antisymD)

lemma *ord-option-chainD*:

Complete-Partial-Order.chain (ord-option ord) Y
 $\implies \text{Complete-Partial-Order.chain ord } \{x. \text{Some } x \in Y\}$
by(rule chainI)(auto dest: chainD)

definition *lub-option* :: $('a \text{ set} \Rightarrow 'b) \Rightarrow 'a \text{ option set} \Rightarrow 'b \text{ option}$

where *lub-option lub Y* = (if $Y \subseteq \{\text{None}\}$ then None else Some (lub $\{x. \text{Some } x \in Y\}$))

lemma *map-lub-option*: *map-option f (lub-option lub Y)* = *lub-option (f \circ lub) Y*

by(simp add: lub-option-def)

lemma *lub-option-upper*:

assumes *Complete-Partial-Order.chain (ord-option ord) Y* $x \in Y$
and *lub-upper*: $\bigwedge Y \ x. \llbracket \text{Complete-Partial-Order.chain ord } Y; x \in Y \rrbracket \implies \text{ord } x \ (\text{lub } Y)$
shows *ord-option ord x (lub-option lub Y)*
using *assms(1-2)*
by(cases x)(auto simp: lub-option-def intro: lub-upper[OF ord-option-chainD])

lemma *lub-option-least*:

assumes *Y: Complete-Partial-Order.chain (ord-option ord) Y*

and *upper*: $\bigwedge x. x \in Y \implies \text{ord-option ord } x \ y$
assumes *lub-least*: $\bigwedge Y y. \llbracket \text{Complete-Partial-Order.chain ord } Y; \bigwedge x. x \in Y \implies \text{ord } x \ y \rrbracket \implies \text{ord (lub } Y) \ y$
shows *ord-option ord (lub-option lub Y) y*
using *Y*
by(*cases y*)(*auto 4 3 simp add: lub-option-def intro: lub-least[OF ord-option-chainD]*
dest: upper)

lemma *lub-map-option*: *lub-option lub (map-option f ‘ Y) = lub-option (lub o (‘ f)) Y*
proof –
have $\bigwedge u y. \llbracket \text{Some } u \in Y; y \in Y \rrbracket \implies \{f \ y \mid y. \text{Some } y \in Y\} = f \ ‘ \{x. \text{Some } x \in Y\}$
by *blast*
then show *?thesis*
by (*auto simp: lub-option-def*)
qed

lemma *ord-option-mono*: $\llbracket \text{ord-option } A \ x \ y; \bigwedge x y. A \ x \ y \implies B \ x \ y \rrbracket \implies \text{ord-option } B \ x \ y$
by(*auto elim: ord-option.cases*)

lemma *ord-option-mono' [mono]*:
 $(\bigwedge x y. A \ x \ y \longrightarrow B \ x \ y) \implies \text{ord-option } A \ x \ y \longrightarrow \text{ord-option } B \ x \ y$
by(*blast intro: ord-option-mono*)

lemma *ord-option-compp*: *ord-option (A OO B) = ord-option A OO ord-option B*
by(*auto simp: fun-eq-iff elim!: ord-option.cases intro: ord-option.intros*)

lemma *ord-option-inf*: *inf (ord-option A) (ord-option B) = ord-option (inf A B)*
(is ?lhs = ?rhs)
proof(*rule antisym*)
show *?lhs ≤ ?rhs* **by**(*auto elim!: ord-option.cases*)
qed(*auto elim: ord-option-mono*)

lemma *ord-option-map2*: *ord-option ord x (map-option f y) = ord-option (λx y. ord x (f y)) x y*
by(*auto elim: ord-option.cases*)

lemma *ord-option-map1*: *ord-option ord (map-option f x) y = ord-option (λx y. ord (f x) y) x y*
by(*auto elim: ord-option.cases*)

lemma *option-ord-Some1-iff*: *option-ord (Some x) y \longleftrightarrow y = Some x*
by(*auto simp: flat-ord-def*)

25.1.3 A relator for sets that treats sets like predicates

context *includes lifting-syntax*

begin

definition *rel-pred* :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a set \Rightarrow 'b set \Rightarrow bool
where *rel-pred* R A B = (R \implies (=)) ($\lambda x. x \in A$) ($\lambda y. y \in B$)

lemma *rel-predI*: (R \implies (=)) ($\lambda x. x \in A$) ($\lambda y. y \in B$) \implies *rel-pred* R A B
by(*simp add: rel-pred-def*)

lemma *rel-predD*: $\llbracket \text{rel-pred } R \ A \ B; R \ x \ y \rrbracket \implies x \in A \longleftrightarrow y \in B$
by(*simp add: rel-pred-def rel-fun-def*)

lemma *Collect-parametric*: ((A \implies (=)) \implies *rel-pred* A) *Collect Collect*
 — Declare this rule as *transfer-rule* only locally because it blows up the search space for *transfer* (in combination with *Collect-transfer*)
by(*simp add: rel-funI rel-predI*)

end

25.1.4 Monotonicity rules

lemma *monotone-gfp-eadd1*: *monotone* (\geq) (\geq) ($\lambda x. x + y :: \text{enat}$)
by(*auto intro!: monotoneI*)

lemma *monotone-gfp-eadd2*: *monotone* (\geq) (\geq) ($\lambda y. x + y :: \text{enat}$)
by(*auto intro!: monotoneI*)

lemma *mono2mono-gfp-eadd*[*THEN* *gfp.mono2mono2*, *cont-intro*, *simp*]:
shows *monotone-eadd*: *monotone* (*rel-prod* (\geq) (\geq)) (\geq) ($\lambda(x, y). x + y :: \text{enat}$)
by(*simp add: monotone-gfp-eadd1 monotone-gfp-eadd2*)

lemma *eadd-gfp-partial-function-mono* [*partial-function-mono*]:
 $\llbracket \text{monotone } (\text{fun-ord } (\geq)) (\geq) f; \text{monotone } (\text{fun-ord } (\geq)) (\geq) g \rrbracket$
 $\implies \text{monotone } (\text{fun-ord } (\geq)) (\geq) (\lambda x. f \ x + g \ x :: \text{enat})$
by(*rule mono2mono-gfp-eadd*)

lemma *mono2mono-ereal*[*THEN* *lfp.mono2mono*]:
shows *monotone-ereal*: *monotone* (\leq) (\leq) *ereal*
by(*rule monotoneI*) *simp*

lemma *mono2mono-ennreal*[*THEN* *lfp.mono2mono*]:
shows *monotone-ennreal*: *monotone* (\leq) (\leq) *ennreal*
by(*rule monotoneI*)(*simp add: ennreal-leI*)

25.1.5 Bijections

lemma *bi-unique-rel-set-bij-betw*:
assumes *unique*: *bi-unique* R
and *rel*: *rel-set* R A B
shows $\exists f. \text{bij-betw } f \ A \ B \wedge (\forall x \in A. R \ x \ (f \ x))$
proof —

from *assms* **obtain** *f* **where** $f: \bigwedge x. x \in A \implies R\ x\ (f\ x)$ **and** $B: \bigwedge x. x \in A \implies f\ x \in B$
by (*metis bi-unique-rel-set-lemma image-eqI*)
have *inj-on* *f* *A*
by (*metis (no-types, lifting) bi-unique-def f inj-on-def unique*)
moreover **have** $f\ 'A = B$ **using** *rel*
by (*smt (verit) bi-unique-def bi-unique-rel-set-lemma f image-cong unique*)
ultimately **have** *bij-betw* *f* *A* *B* **unfolding** *bij-betw-def* ..
thus *?thesis* **using** *f* **by** *blast*
qed

lemma *bij-betw-rel-setD*: $bij\ betw\ f\ A\ B \implies rel\ set\ (\lambda x\ y. y = f\ x)\ A\ B$
by(*rule rel-setI*)(*auto dest: bij-betwE bij-betw-imp-surj-on[symmetric]*)

25.2 Subprobability mass function

type-synonym *'a* *spmf* = *'a* *option* *pmf*
translations (*type*) *'a* *spmf* \leftarrow (*type*) *'a* *option* *pmf*

definition *measure-spmf* :: *'a* *spmf* \Rightarrow *'a* *measure*
where *measure-spmf* *p* = *distr* (*restrict-space* (*measure-pmf* *p*) (*range* *Some*))
(*count-space* *UNIV*) *the*

abbreviation *spmf* :: *'a* *spmf* \Rightarrow *'a* \Rightarrow *real*
where *spmf* *p* *x* \equiv *pmf* *p* (*Some* *x*)

lemma *space-measure-spmf*: *space* (*measure-spmf* *p*) = *UNIV*
by(*simp add: measure-spmf-def*)

lemma *sets-measure-spmf* [*simp*, *measurable-cong*]: *sets* (*measure-spmf* *p*) = *sets*
(*count-space* *UNIV*)
by(*simp add: measure-spmf-def*)

lemma *measure-spmf-not-bot* [*simp*]: *measure-spmf* *p* $\neq \perp$
by (*metis empty-not-UNIV space-bot space-measure-spmf*)

lemma *measurable-the-measure-pmf-Some* [*measurable*, *simp*]:
 $the \in measurable\ (restrict\ space\ (measure\ pmf\ p)\ (range\ Some))\ (count\ space\ UNIV)$
by(*auto simp: measurable-def sets-restrict-space space-restrict-space integral-restrict-space*)

lemma *measurable-spmf-measure1* [*simp*]: *measurable* (*measure-spmf* *M*) *N* = *UNIV*
 \rightarrow *space* *N*
by(*auto simp: measurable-def space-measure-spmf*)

lemma *measurable-spmf-measure2* [*simp*]: *measurable* *N* (*measure-spmf* *M*) = *measurable* *N* (*count-space* *UNIV*)
by(*intro measurable-cong-sets simp-all*)

lemma *subprob-space-measure-spmf* [*simp*, *intro!*]: *subprob-space* (*measure-spmf* *p*)
proof
show *emeasure* (*measure-spmf* *p*) (*space* (*measure-spmf* *p*)) ≤ 1
by(*simp* *add*: *measure-spmf-def* *emeasure-distr* *emeasure-restrict-space* *space-restrict-space*
measure-pmf.measure-le-1)
qed(*simp* *add*: *space-measure-spmf*)

interpretation *measure-spmf*: *subprob-space* *measure-spmf* *p* **for** *p*
by(*rule* *subprob-space-measure-spmf*)

lemma *finite-measure-spmf* [*simp*]: *finite-measure* (*measure-spmf* *p*)
by *unfold-locales*

lemma *spmf-conv-measure-spmf*: *spmf* *p* *x* = *measure* (*measure-spmf* *p*) {*x*}
by(*auto* *simp*: *measure-spmf-def* *measure-distr* *measure-restrict-space* *pmf.rep-eq*
space-restrict-space *intro*: *arg-cong2*[**where** *f*=*measure*])

lemma *emeasure-measure-spmf-conv-measure-pmf*:
emeasure (*measure-spmf* *p*) *A* = *emeasure* (*measure-pmf* *p*) (*Some* ‘*A*)
by(*auto* *simp*: *measure-spmf-def* *emeasure-distr* *emeasure-restrict-space* *space-restrict-space*
intro: *arg-cong2*[**where** *f*=*emeasure*])

lemma *measure-measure-spmf-conv-measure-pmf*:
measure (*measure-spmf* *p*) *A* = *measure* (*measure-pmf* *p*) (*Some* ‘*A*)
using *emeasure-measure-spmf-conv-measure-pmf*[*of* *p* *A*]
by(*simp* *add*: *measure-spmf.emeasure-eq-measure* *measure-pmf.emeasure-eq-measure*)

lemma *emeasure-spmf-map-pmf-Some* [*simp*]:
emeasure (*measure-spmf* (*map-pmf* *Some* *p*)) *A* = *emeasure* (*measure-pmf* *p*) *A*
by(*auto* *simp*: *measure-spmf-def* *emeasure-distr* *emeasure-restrict-space* *space-restrict-space*
intro: *arg-cong2*[**where** *f*=*emeasure*])

lemma *measure-spmf-map-pmf-Some* [*simp*]:
measure (*measure-spmf* (*map-pmf* *Some* *p*)) *A* = *measure* (*measure-pmf* *p*) *A*
using *emeasure-spmf-map-pmf-Some*[*of* *p* *A*] **by**(*simp* *add*: *measure-spmf.emeasure-eq-measure*
measure-pmf.emeasure-eq-measure)

lemma *nn-integral-measure-spmf*: ($\int^+ x. f\ x\ \partial\text{measure-spmf}\ p$) = $\int^+ x. \text{ennreal}$
(*spmf* *p* *x*) * *f* *x* $\partial\text{count-space UNIV}$
(**is** ?*lhs* = ?*rhs*)

proof –

have ?*lhs* = $\int^+ x. \text{pmf}\ p\ x * f\ (\text{the}\ x)\ \partial\text{count-space} (\text{range}\ \text{Some})$
by(*simp* *add*: *measure-spmf-def* *nn-integral-distr* *nn-integral-restrict-space* *nn-integral-measure-pmf*
nn-integral-count-space-indicator *ac-simps*
flip: *times-ereal.simps* [*symmetric*])
also have $\dots = \int^+ x. \text{ennreal}\ (\text{spmf}\ p\ (\text{the}\ x)) * f\ (\text{the}\ x)\ \partial\text{count-space} (\text{range}\ \text{Some})$
by(*rule* *nn-integral-cong*) *auto*
also have $\dots = \int^+ x. \text{spmf}\ p\ (\text{the}\ (\text{Some}\ x)) * f\ (\text{the}\ (\text{Some}\ x))\ \partial\text{count-space}$

UNIV

by(rule nn-integral-bij-count-space[symmetric])(simp add: bij-betw-def)
 also have $\dots = ?rhs$ by simp
 finally show ?thesis .
 qed

lemma integral-measure-spmf:

assumes integrable (measure-spmf p) f
 shows $\int x. f x \partial \text{measure-spmf } p = \int x. \text{spm}f\ p\ x * f\ x \partial \text{count-space UNIV}$
proof –
 have integrable (count-space UNIV) $(\lambda x. \text{spm}f\ p\ x * f\ x)$
 using assms by(simp add: integrable-iff-bounded nn-integral-measure-spmf abs-mult
 ennreal-mult'')
 then show ?thesis using assms
 by(simp add: real-lebesgue-integral-def nn-integral-measure-spmf ennreal-mult'[symmetric])
 qed

lemma emeasure-spmf-single: $\text{emeasure (measure-spmf } p) \{x\} = \text{spm}f\ p\ x$
 by(simp add: measure-spmf.emeasure-eq-measure spmf-conv-measure-spmf)

lemma measurable-measure-spmf[measurable]:

$(\lambda x. \text{measure-spmf } (M\ x)) \in \text{measurable (count-space UNIV) (subprob-algebra (count-space UNIV))}$
 by (auto simp: space-subprob-algebra)

lemma nn-integral-measure-spmf-conv-measure-pmf:

assumes [measurable]: $f \in \text{borel-measurable (count-space UNIV)}$
 shows $\text{nn-integral (measure-spmf } p) f = \text{nn-integral (restrict-space (measure-pmf } p) (\text{range Some})) (f \circ \text{the})$
 by(simp add: measure-spmf-def nn-integral-distr o-def)

lemma measure-spmf-in-space-subprob-algebra [simp]:

$\text{measure-spmf } p \in \text{space (subprob-algebra (count-space UNIV))}$
 by(simp add: space-subprob-algebra)

lemma nn-integral-spmf-neq-top: $(\int^+ x. \text{spm}f\ p\ x \partial \text{count-space UNIV}) \neq \top$

using nn-integral-measure-spmf[where $f=\lambda-. 1$, of p, symmetric]
 by simp

lemma SUP-spmf-neq-top': $(\text{SUP } p \in Y. \text{ennreal (spm}f\ p\ x)) \neq \top$

by (metis SUP-least ennreal-le-1 ennreal-one-neq-top neq-top-trans pmf-le-1)

lemma SUP-spmf-neq-top: $(\text{SUP } i. \text{ennreal (spm}f\ (Y\ i)\ x)) \neq \top$

by (meson SUP-eq-top-iff ennreal-le-1 ennreal-one-less-top linorder-not-le pmf-le-1)

lemma SUP-emeasure-spmf-neq-top: $(\text{SUP } p \in Y. \text{emeasure (measure-spmf } p) A) \neq \top$

by (metis ennreal-one-less-top less-SUP-iff linorder-not-le measure-spmf.subprob-emeasure-le-1)

25.3 Support

definition *set-spmf* :: 'a spmf \Rightarrow 'a set
 where *set-spmf* *p* = *set-pmf* *p* \gg *set-option*

lemma *set-spmf-rep-eq*: *set-spmf* *p* = {*x*. *measure* (*measure-spmf* *p*) {*x*} \neq 0}

proof –

have $\bigwedge x :: 'a. \text{the } -' \{x\} \cap \text{range } \text{Some} = \{\text{Some } x\}$ **by** *auto*

then **show** *?thesis*

unfolding *set-spmf-def* *measure-spmf-def*

by(*auto simp: set-pmf.rep-eq measure-distr measure-restrict-space space-restrict-space*)

qed

lemma *in-set-spmf*: $x \in \text{set-spmf } p \longleftrightarrow \text{Some } x \in \text{set-pmf } p$

by(*simp add: set-spmf-def*)

lemma *AE-measure-spmf-iff* [*simp*]: $(\text{AE } x \text{ in } \text{measure-spmf } p. P \ x) \longleftrightarrow (\forall x \in \text{set-spmf } p. P \ x)$

unfolding *set-spmf-def* *measure-spmf-def*

by(*force simp: AE-distr-iff AE-restrict-space-iff AE-measure-pmf-iff cong del: AE-cong*)

lemma *spm-f-eq-0-set-spmf*: $\text{spm-f } p \ x = 0 \longleftrightarrow x \notin \text{set-spmf } p$

by(*auto simp: pmf-eq-0-set-pmf set-spmf-def*)

lemma *in-set-spmf-iff-spmf*: $x \in \text{set-spmf } p \longleftrightarrow \text{spm-f } p \ x \neq 0$

by(*auto simp: set-spmf-def set-pmf-iff*)

lemma *set-spmf-return-pmf-None* [*simp*]: $\text{set-spmf } (\text{return-pmf } \text{None}) = \{\}$

by(*auto simp: set-spmf-def*)

lemma *countable-set-spmf* [*simp*]: *countable* (*set-spmf* *p*)

by(*simp add: set-spmf-def bind-UNION*)

lemma *spm-f-eqI*:

assumes $\bigwedge i. \text{spm-f } p \ i = \text{spm-f } q \ i$

shows $p = q$

proof(*rule pmf-eqI*)

fix *i*

show $\text{pmf } p \ i = \text{pmf } q \ i$

proof(*cases i*)

case (*Some i'*)

thus *?thesis* **by**(*simp add: assms*)

next

case *None*

have $\text{ennreal } (\text{pmf } p \ i) = \text{measure } (\text{measure-pmf } p) \ \{i\}$ **by**(*simp add: pmf-def*)

also have $\{i\} = \text{space } (\text{measure-pmf } p) - \text{range } \text{Some}$

by(*auto simp: None intro: ccontr*)

also have $\text{measure } (\text{measure-pmf } p) \ \dots = \text{ennreal } 1 - \text{measure } (\text{measure-pmf } p) \ (\text{range } \text{Some})$

by(simp add: measure-pmf.prob-compl ennreal-minus[symmetric] del: space-measure-pmf)
also have range Some = $(\bigcup x \in \text{set-spmf } p. \{ \text{Some } x \}) \cup \text{Some } '(- \text{ set-spmf } p)$
by auto
also have measure (measure-pmf p) ... = measure (measure-pmf p) $(\bigcup x \in \text{set-spmf } p. \{ \text{Some } x \})$
by(rule measure-pmf.measure-zero-union)(auto simp: measure-pmf.prob-eq-0 AE-measure-pmf-iff in-set-spmf-iff-spmf set-pmf-iff)
also have ennreal ... = $\int^+ x. \text{measure (measure-pmf } p) \{ \text{Some } x \} \partial \text{count-space (set-spmf } p)$
unfolding measure-pmf.emmeasure-eq-measure[symmetric]
by(simp-all add: emmeasure-UN-countable disjoint-family-on-def)
also have ... = $\int^+ x. \text{spmfs } p \ x \ \partial \text{count-space (set-spmf } p)$ **by**(simp add: pmf-def)
also have ... = $\int^+ x. \text{spmfs } q \ x \ \partial \text{count-space (set-spmf } p)$ **by**(simp add: assms)
also have set-spmf p = set-spmf q **by**(auto simp: in-set-spmf-iff-spmf assms)
also have $(\int^+ x. \text{spmfs } q \ x \ \partial \text{count-space (set-spmf } q)) = \int^+ x. \text{measure (measure-pmf } q) \{ \text{Some } x \} \partial \text{count-space (set-spmf } q)$
by(simp add: pmf-def)
also have ... = measure (measure-pmf q) $(\bigcup x \in \text{set-spmf } q. \{ \text{Some } x \})$
unfolding measure-pmf.emmeasure-eq-measure[symmetric]
by(simp-all add: emmeasure-UN-countable disjoint-family-on-def)
also have ... = measure (measure-pmf q) $((\bigcup x \in \text{set-spmf } q. \{ \text{Some } x \}) \cup \text{Some } '(- \text{ set-spmf } q))$
by(rule ennreal-cong measure-pmf.measure-zero-union[symmetric])+(auto simp: measure-pmf.prob-eq-0 AE-measure-pmf-iff in-set-spmf-iff-spmf set-pmf-iff)
also have $((\bigcup x \in \text{set-spmf } q. \{ \text{Some } x \}) \cup \text{Some } '(- \text{ set-spmf } q)) = \text{range Some}$ **by** auto
also have ennreal 1 - measure (measure-pmf q) ... = measure (measure-pmf q) $(\text{space (measure-pmf } q) - \text{range Some})$
by(simp add: one-ereal-def measure-pmf.prob-compl ennreal-minus[symmetric] del: space-measure-pmf)
also have space (measure-pmf q) - range Some = {i}
by(auto simp: None intro: ccontr)
also have measure (measure-pmf q) ... = pmf q i **by**(simp add: pmf-def)
finally show ?thesis **by** simp
qed
qed

lemma integral-measure-spmf-restrict:

fixes f :: 'a \Rightarrow 'b :: {banach, second-countable-topology}
shows $(\int x. f \ x \ \partial \text{measure-spmf } M) = (\int x. f \ x \ \partial \text{restrict-space (measure-spmf } M) \text{ (set-spmf } M))$
by(auto intro!: integral-cong-AE simp add: integral-restrict-space)

lemma nn-integral-measure-spmf':

$(\int^+ x. f \ x \ \partial \text{measure-spmf } p) = \int^+ x. \text{ennreal (spmfs } p \ x) * f \ x \ \partial \text{count-space (set-spmf } p)$
by(auto simp: nn-integral-measure-spmf nn-integral-count-space-indicator in-set-spmf-iff-spmf intro!: nn-integral-cong split: split-indicator)

25.4 Functorial structure

abbreviation *map-spmf* :: (*'a* \Rightarrow *'b*) \Rightarrow *'a* *spmf* \Rightarrow *'b* *spmf*
where *map-spmf* *f* \equiv *map-pmf* (*map-option* *f*)

context **begin**

local-setup \langle *Local-Theory.map-background-naming* (*Name-Space.mandatory-path* *spmf*) \rangle

lemma *map-comp*: *map-spmf* *f* (*map-spmf* *g* *p*) = *map-spmf* (*f* \circ *g*) *p*
by(*simp* *add*: *pmf.map-comp* *o-def* *option.map-comp*)

lemma *map-id0*: *map-spmf* *id* = *id*
by(*simp* *add*: *pmf.map-id* *option.map-id0*)

lemma *map-id* [*simp*]: *map-spmf* *id* *p* = *p*
by(*simp* *add*: *map-id0*)

lemma *map-ident* [*simp*]: *map-spmf* ($\lambda x. x$) *p* = *p*
by(*simp* *add*: *id-def*[*symmetric*])

end

lemma *set-map-spmf* [*simp*]: *set-spmf* (*map-spmf* *f* *p*) = *f* ‘ *set-spmf* *p*
by(*simp* *add*: *set-spmf-def* *image-bind* *bind-image* *o-def* *Option.option.set-map*)

lemma *map-spmf-cong*:
 $\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q \implies f x = g x \rrbracket \implies \text{map-spmf } f p = \text{map-spmf } g q$
by(*auto* *intro*: *pmf.map-cong* *option.map-cong* *simp* *add*: *in-set-spmf*)

lemma *map-spmf-cong-simp*:
 $\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q =_{\text{simp}} f x = g x \rrbracket$
 $\implies \text{map-spmf } f p = \text{map-spmf } g q$
unfolding *simp-implies-def* **by**(*rule* *map-spmf-cong*)

lemma *map-spmf-idI*: ($\bigwedge x. x \in \text{set-spmf } p \implies f x = x$) $\implies \text{map-spmf } f p = p$
by(*rule* *map-pmf-idI* *map-option-idI*)+(*simp* *add*: *in-set-spmf*)

lemma *emeasure-map-spmf*:
 $\text{emeasure } (\text{measure-spmf } (\text{map-spmf } f p)) A = \text{emeasure } (\text{measure-spmf } p) (f \text{ - ' } A)$
by(*auto* *simp*: *measure-spmf-def* *emeasure-distr* *measurable-restrict-space1* *space-restrict-space* *emeasure-restrict-space* *intro*: *arg-cong2*[**where** *f*=*emeasure*])

lemma *measure-map-spmf*: $\text{measure } (\text{measure-spmf } (\text{map-spmf } f p)) A = \text{measure } (\text{measure-spmf } p) (f \text{ - ' } A)$
using *emeasure-map-spmf*[*of f p A*] **by**(*simp* *add*: *measure-spmf.emeasure-eq-measure*)

lemma *measure-map-spmf-conv-distr*:
 $\text{measure-spmf } (\text{map-spmf } f p) = \text{distr } (\text{measure-spmf } p) (\text{count-space UNIV}) f$

by(rule *measure-eqI*)(simp-all add: *emeasure-map-spmf emeasure-distr*)

lemma *spmf-map-pmf-Some* [simp]: *spmf (map-pmf Some p) i = pmf p i*
by(simp add: *pmf-map-inj'*)

lemma *spmf-map-inj*: $\llbracket \text{inj-on } f \text{ (set-spmf } M); x \in \text{set-spmf } M \rrbracket \implies \text{spmf (map-spmf } f \text{ } M) (f \text{ } x) = \text{spmf } M \text{ } x$
by (smt (verit) *elem-set in-set-spmf inj-on-def option.inj-map-strong option.map(2) pmf-map-inj*)

lemma *spmf-map-inj'*: *inj f \implies spmf (map-spmf f M) (f x) = spmf M x*
by(subst *option.map(2)[symmetric, where f=f]*)(rule *pmf-map-inj'[OF option.inj-map]*)

lemma *spmf-map-outside*: *x \notin f ‘ set-spmf M \implies spmf (map-spmf f M) x = 0*
unfolding *spmf-eq-0-set-spmf* **by** *simp*

lemma *ennreal-spmf-map*: *ennreal (spmf (map-spmf f p) x) = emeasure (measure-spmf p) (f - ‘ {x})*
by (*metis emeasure-map-spmf emeasure-spmf-single*)

lemma *spmf-map*: *spmf (map-spmf f p) x = measure (measure-spmf p) (f - ‘ {x})*
using *ennreal-spmf-map[of f p x]* **by**(simp add: *measure-spmf.emeasure-eq-measure*)

lemma *ennreal-spmf-map-conv-nn-integral*:
ennreal (spmf (map-spmf f p) x) = integral^N (measure-spmf p) (indicator (f - ‘ {x}))
by (*simp add: ennreal-spmf-map*)

25.5 Monad operations

25.5.1 Return

abbreviation *return-spmf* :: '*a* \Rightarrow '*a* *spmf*
where *return-spmf x \equiv return-pmf (Some x)*

lemma *pmf-return-spmf*: *pmf (return-spmf x) y = indicator {y} (Some x)*
by(fact *pmf-return*)

lemma *measure-spmf-return-spmf*: *measure-spmf (return-spmf x) = Giry-Monad.return (count-space UNIV) x*
by(rule *measure-eqI*)(simp-all add: *measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space indicator-def*)

lemma *measure-spmf-return-pmf-None* [simp]: *measure-spmf (return-pmf None) = null-measure (count-space UNIV)*
by (*simp add: emeasure-measure-spmf-conv-measure-pmf measure-eqI*)

lemma *set-return-spmf* [simp]: *set-spmf (return-spmf x) = {x}*
by(auto simp: *set-spmf-def*)

25.5.2 Bind

definition $\text{bind-spmf} :: 'a \text{ spmf} \Rightarrow ('a \Rightarrow 'b \text{ spmf}) \Rightarrow 'b \text{ spmf}$

where $\text{bind-spmf } x \ f = \text{bind-pmf } x \ (\lambda a. \text{case } a \text{ of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } a' \Rightarrow f \ a')$

adhoc-overloading $\text{Monad-Syntax.bind} \equiv \text{bind-spmf}$

lemma $\text{return-None-bind-spmf} \ [simp]: \text{return-pmf } \text{None} \ggg (f :: 'a \Rightarrow -) = \text{return-pmf } \text{None}$

by $(\text{simp add: bind-spmf-def bind-return-pmf})$

lemma $\text{return-bind-spmf} \ [simp]: \text{return-spmf } x \ggg f = f \ x$

by $(\text{simp add: bind-spmf-def bind-return-pmf})$

lemma $\text{bind-return-spmf} \ [simp]: x \ggg \text{return-spmf} = x$

proof –

have $\bigwedge a :: 'a \text{ option. } (\text{case } a \text{ of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } a' \Rightarrow \text{return-spmf } a') = \text{return-pmf } a$

by $(\text{simp split: option.split})$

then show $?thesis$

by $(\text{simp add: bind-spmf-def bind-return-pmf})$

qed

lemma $\text{bind-spmf-assoc} \ [simp]:$

fixes $x :: 'a \text{ spmf}$ **and** $f :: 'a \Rightarrow 'b \text{ spmf}$ **and** $g :: 'b \Rightarrow 'c \text{ spmf}$

shows $(x \ggg f) \ggg g = x \ggg (\lambda y. f \ y \ggg g)$

unfolding bind-spmf-def

by $(\text{smt (verit, best) bind-assoc-pmf bind-pmf-cong bind-return-pmf option.case-eq-if})$

lemma $\text{pmf-bind-spmf-None}: \text{pmf } (p \ggg f) \ \text{None} = \text{pmf } p \ \text{None} + \int x. \text{pmf } (f \ x) \ \text{None} \ \partial \text{measure-spmf } p$

(is $?lhs = ?rhs$ **)**

proof –

let $?f = \lambda x. \text{pmf } (\text{case } x \text{ of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } x \Rightarrow f \ x) \ \text{None}$

have $?lhs = \int x. ?f \ x \ \partial \text{measure-pmf } p$

by $(\text{simp add: bind-spmf-def pmf-bind})$

also have $\dots = \int x. ?f \ \text{None} * \text{indicator } \{\text{None}\} \ x + ?f \ x * \text{indicator } (\text{range } \text{Some}) \ x \ \partial \text{measure-pmf } p$

by $(\text{rule Bochner-Integration.integral-cong})(\text{auto simp: indicator-def})$

also have $\dots = (\int x. ?f \ \text{None} * \text{indicator } \{\text{None}\} \ x \ \partial \text{measure-pmf } p) + (\int x. ?f \ x * \text{indicator } (\text{range } \text{Some}) \ x \ \partial \text{measure-pmf } p)$

by $(\text{rule Bochner-Integration.integral-add})(\text{auto 4 3 intro: integrable-real-mult-indicator measure-pmf.integrable-const-bound[where B=1] simp add: AE-measure-pmf-iff pmf-le-1})$

also have $\dots = \text{pmf } p \ \text{None} + \int x. \text{indicator } (\text{range } \text{Some}) \ x * \text{pmf } (f \ (the \ x)) \ \text{None} \ \partial \text{measure-pmf } p$

by $(\text{auto simp: measure-measure-pmf-finite indicator-eq-0-iff intro!: Bochner-Integration.integral-cong})$

also have $\dots = ?rhs$

unfolding measure-spmf-def

by(subst integral-distr)(auto simp: integral-restrict-space)
 finally show ?thesis .
 qed

lemma *spmf-bind*: $\text{spmf } (p \gg f) y = \int x. \text{spmf } (f x) y \partial \text{measure-spmf } p$
proof –
 have $\bigwedge x. \text{spmf } (\text{case } x \text{ of } \text{None} \Rightarrow \text{return-pmf None} \mid \text{Some } x \Rightarrow f x) y =$
 $\text{indicat-real } (\text{range } \text{Some}) x * \text{spmf } (f (\text{the } x)) y$
 by (simp add: split: option.split)
 then show ?thesis
 by (simp add: measure-spmf-def integral-distr bind-spmf-def pmf-bind inte-
 gral-restrict-space)
 qed

lemma *ennreal-spmf-bind*: $\text{ennreal } (\text{spmf } (p \gg f) x) = \int^+ y. \text{spmf } (f y) x \partial \text{mea-}$
 $\text{sure-spmf } p$
proof –
 have $\bigwedge y. \text{ennreal } (\text{spmf } (\text{case } y \text{ of } \text{None} \Rightarrow \text{return-pmf None} \mid \text{Some } x \Rightarrow f x)$
 $x) =$
 $\text{ennreal } (\text{spmf } (f (\text{the } y)) x) * \text{indicator } (\text{range } \text{Some}) y$
 by (simp add: split: option.split)
 then show ?thesis
 by (simp add: bind-spmf-def ennreal-pmf-bind nn-integral-measure-spmf-conv-measure-pmf
 nn-integral-restrict-space)
 qed

lemma *measure-spmf-bind-pmf*: $\text{measure-spmf } (p \gg f) = \text{measure-pmf } p \gg$
 $\text{measure-spmf } \circ f$
 (is ?lhs = ?rhs)
proof(rule measure-eqI)
 show sets ?lhs = sets ?rhs
 by (simp add: Giry-Monad.bind-def)
 next
 fix $A :: 'a \text{ set}$
 have $\text{emeasure } ?lhs A = \int^+ x. \text{emeasure } (\text{measure-spmf } (f x)) A \partial \text{measure-pmf}$
 p
 by (simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space
 bind-spmf-def)
 also have $\dots = \text{emeasure } ?rhs A$
 by (simp add: emeasure-bind[where $N = \text{count-space UNIV}$] space-measure-spmf
 space-subprob-algebra)
 finally show $\text{emeasure } ?lhs A = \text{emeasure } ?rhs A$.
 qed

lemma *measure-spmf-bind*: $\text{measure-spmf } (p \gg f) = \text{measure-spmf } p \gg \text{mea-}$
 $\text{sure-spmf } \circ f$
 (is ?lhs = ?rhs)
proof(rule measure-eqI)
 show sets ?lhs = sets ?rhs

```

  by(simp add: sets-bind[where N=count-space UNIV] space-measure-spmf)
next
  fix A :: 'a set
  let ?A = the - ' A ∩ range Some
  have emeasure ?lhs A =  $\int^+ x. \text{emeasure (measure-pmf (case x of None } \Rightarrow$ 
return-pmf None | Some x  $\Rightarrow f x))$  ?A  $\partial$ measure-pmf p
  by(simp add: measure-spmf-def emeasure-distr space-restrict-space emeasure-restrict-space
bind-spmf-def)
  also have ... =  $\int^+ x. \text{emeasure (measure-pmf (f (the x)))}$  ?A * indicator
(range Some) x  $\partial$ measure-pmf p
  by(rule nn-integral-cong)(auto split: option.split simp add: indicator-def)
  also have ... =  $\int^+ x. \text{emeasure (measure-spmf (f x))}$  A  $\partial$ measure-spmf p
  by(simp add: measure-spmf-def nn-integral-distr nn-integral-restrict-space emea-
sure-distr space-restrict-space emeasure-restrict-space)
  also have ... = emeasure ?rhs A
  by(simp add: emeasure-bind[where N=count-space UNIV] space-measure-spmf
space-subprob-algebra)
  finally show emeasure ?lhs A = emeasure ?rhs A .
qed

```

lemma map-spmf-bind-spmf: $\text{map-spmf } f (\text{bind-spmf } p \ g) = \text{bind-spmf } p (\text{map-spmf } f \circ g)$

```

  by(auto simp: bind-spmf-def map-bind-pmf fun-eq-iff split: option.split intro:
arg-cong2[where f=bind-pmf])

```

lemma bind-map-spmf: $\text{map-spmf } f \ p \ggg g = p \ggg g \circ f$

```

  by(simp add: bind-spmf-def bind-map-pmf o-def cong del: option.case-cong-weak)

```

lemma spmf-bind-leI:

```

  assumes  $\bigwedge y. y \in \text{set-spmf } p \implies \text{spmfmf (f y) } x \leq r$ 

```

```

  and  $0 \leq r$ 

```

```

  shows  $\text{spmfmf (bind-spmf } p \ f) \ x \leq r$ 

```

proof –

```

  have ennreal (spmfmf (bind-spmf p f) x) =  $\int^+ y. \text{spmfmf (f y) } x \ \partial$ measure-spmf p

```

```

  by(rule ennreal-spmf-bind)

```

```

  also have ...  $\leq \int^+ y. r \ \partial$ measure-spmf p

```

```

  by(rule nn-integral-mono-AE)(simp add: assms)

```

```

  also have ...  $\leq r$ 

```

```

  using assms measure-spmf.emeasure-space-le-1

```

```

  by(auto simp: measure-spmf.emeasure-eq-measure intro!: mult-left-le)

```

```

  finally show ?thesis using assms(2) by(simp)

```

qed

lemma map-spmf-conv-bind-spmf: $\text{map-spmf } f \ p = (p \ggg (\lambda x. \text{return-spmf (f x)}))$

```

  by(simp add: map-pmf-def bind-spmf-def)(rule bind-pmf-cong, simp-all split: op-
tion.split)

```

lemma bind-spmf-cong:

```

 $\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q \implies f x = g x \rrbracket \implies \text{bind-spmf } p \ f = \text{bind-spmf } q \ g$ 

```

by(*auto simp: bind-spmf-def in-set-spmf intro: bind-pmf-cong option.case-cong*)

lemma *bind-spmf-cong-simp*:

$\llbracket p = q; \bigwedge x. x \in \text{set-spmf } q \Rightarrow \text{simp} \Rightarrow f x = g x \rrbracket$

$\Rightarrow \text{bind-spmf } p f = \text{bind-spmf } q g$

by(*simp add: simp-implies-def cong: bind-spmf-cong*)

lemma *set-bind-spmf*: $\text{set-spmf } (M \ggg f) = \text{set-spmf } M \ggg (\text{set-spmf } \circ f)$

by(*auto simp: set-spmf-def bind-spmf-def bind-UNION split: option.splits*)

lemma *bind-spmf-const-return-None* [*simp*]: $\text{bind-spmf } p (\lambda \cdot. \text{return-pmf } \text{None}) = \text{return-pmf } \text{None}$

by(*simp add: bind-spmf-def case-option-collapse*)

lemma *bind-commute-spmf*:

$\text{bind-spmf } p (\lambda x. \text{bind-spmf } q (f x)) = \text{bind-spmf } q (\lambda y. \text{bind-spmf } p (\lambda x. f x y))$

(*is ?lhs = ?rhs*)

proof –

let *?f* = $\lambda x y. \text{case } x \text{ of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } a \Rightarrow (\text{case } y \text{ of } \text{None} \Rightarrow \text{return-pmf } \text{None} \mid \text{Some } b \Rightarrow f a b)$

have *?lhs* = $p \ggg (\lambda x. q \ggg ?f x)$

unfolding *bind-spmf-def* **by**(*rule bind-pmf-cong[OF refl](simp split: option.split)*)

also have $\dots = q \ggg (\lambda y. p \ggg (\lambda x. ?f x y))$ **by**(*rule bind-commute-pmf*)

also have $\dots = ?rhs$ **unfolding** *bind-spmf-def*

by(*rule bind-pmf-cong[OF refl](auto split: option.split, metis bind-spmf-const-return-None bind-spmf-def)*)

finally show *?thesis* .

qed

25.6 Relator

abbreviation *rel-spmf* :: $('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow 'a \text{ spmf} \Rightarrow 'b \text{ spmf} \Rightarrow \text{bool}$

where *rel-spmf* *R* $\equiv \text{rel-pmf } (\text{rel-option } R)$

lemma *rel-spmf-mono*:

$\llbracket \text{rel-spmf } A f g; \bigwedge x y. A x y \Rightarrow B x y \rrbracket \Rightarrow \text{rel-spmf } B f g$

by (*metis option.rel-sel pmf.rel-mono-strong*)

lemma *rel-spmf-mono-strong*:

$\llbracket \text{rel-spmf } A f g; \bigwedge x y. \llbracket A x y; x \in \text{set-spmf } f; y \in \text{set-spmf } g \rrbracket \Rightarrow B x y \rrbracket \Rightarrow \text{rel-spmf } B f g$

by (*metis elem-set in-set-spmf option.rel-mono-strong pmf.rel-mono-strong*)

lemma *rel-spmf-reflI*: $(\bigwedge x. x \in \text{set-spmf } p \Rightarrow P x x) \Rightarrow \text{rel-spmf } P p p$

by (*metis (mono-tags, lifting) option.rel-eq pmf.rel-eq rel-spmf-mono-strong*)

lemma *rel-spmfI* [*intro?*]:

$\llbracket \bigwedge x y. (x, y) \in \text{set-spmf } pq \Rightarrow P x y; \text{map-spmf } \text{fst } pq = p; \text{map-spmf } \text{snd } pq = q \rrbracket$

$\Rightarrow \text{rel-spmf } P \ p \ q$
by(rule *rel-pmf.intros*[**where** $pq = \text{map-pmf } (\lambda x. \text{ case } x \text{ of } \text{None} \Rightarrow (\text{None}, \text{None}) \mid \text{Some } (a, b) \Rightarrow (\text{Some } a, \text{Some } b)) \ pq$])
 (auto simp: *pmf.map-comp o-def in-set-spmf split: option.splits intro: pmf.map-cong*)

lemma *rel-spmfE* [*elim?*, *consumes 1*, *case-names rel-spmf*]:

assumes *rel-spmf* $P \ p \ q$
obtains pq **where**
 $\bigwedge x \ y. (x, y) \in \text{set-spmf } pq \Rightarrow P \ x \ y$
 $p = \text{map-spmf fst } pq$
 $q = \text{map-spmf snd } pq$
using *assms*
proof(cases rule: *rel-pmf.cases*[*consumes 1*, *case-names rel-pmf*])
case (*rel-pmf* pq)
let $?pq = \text{map-pmf } (\lambda(a, b). \text{ case } (a, b) \text{ of } (\text{Some } x, \text{Some } y) \Rightarrow \text{Some } (x, y) \mid - \Rightarrow \text{None}) \ pq$
have $\bigwedge x \ y. (x, y) \in \text{set-spmf } ?pq \Rightarrow P \ x \ y$
by(auto simp: *in-set-spmf split: option.split-asm dest: rel-pmf(1)*)
moreover
have $\bigwedge x. (x, \text{None}) \in \text{set-pmf } pq \Rightarrow x = \text{None}$ **by**(auto dest!: *rel-pmf(1)*)
then have $p = \text{map-spmf fst } ?pq$ **using** *rel-pmf(2)*
by(auto simp: *pmf.map-comp split-beta intro!: pmf.map-cong split: option.split*)
moreover
have $\bigwedge y. (\text{None}, y) \in \text{set-pmf } pq \Rightarrow y = \text{None}$ **by**(auto dest!: *rel-pmf(1)*)
then have $q = \text{map-spmf snd } ?pq$ **using** *rel-pmf(3)*
by(auto simp: *pmf.map-comp split-beta intro!: pmf.map-cong split: option.split*)
ultimately show thesis ..
qed

lemma *rel-spmf-simps*:

$\text{rel-spmf } R \ p \ q \longleftrightarrow (\exists pq. (\forall (x, y) \in \text{set-spmf } pq. R \ x \ y) \wedge \text{map-spmf fst } pq = p \wedge \text{map-spmf snd } pq = q)$
by(auto intro: *rel-spmfI elim!: rel-spmfE*)

lemma *spm-rel-map*:

shows *spm-rel-map1*: $\bigwedge R \ f \ x. \text{rel-spmf } R \ (\text{map-spmf } f \ x) = \text{rel-spmf } (\lambda x. R \ (f \ x)) \ x$
and *spm-rel-map2*: $\bigwedge R \ x \ g \ y. \text{rel-spmf } R \ x \ (\text{map-spmf } g \ y) = \text{rel-spmf } (\lambda x \ y. R \ x \ (g \ y)) \ x \ y$
by(simp-all add: *fun-eq-iff pmf.rel-map option.rel-map[abs-def]*)

lemma *spm-rel-conversep*: $\text{rel-spmf } R^{-1-1} = (\text{rel-spmf } R)^{-1-1}$

by(simp add: *option.rel-conversep pmf.rel-conversep*)

lemma *spm-rel-eq*: $\text{rel-spmf } (=) = (=)$

by(simp add: *pmf.rel-eq option.rel-eq*)

context includes *lifting-syntax*

begin

lemma *bind-spmf-parametric* [*transfer-rule*]:
 $(rel\text{-}spm\text{-}f\ A\ ==\!>\ (A\ ==\!>\ rel\text{-}spm\text{-}f\ B)\ ==\!>\ rel\text{-}spm\text{-}f\ B)\ bind\text{-}spm\text{-}f\ bind\text{-}spm\text{-}f$
unfolding *bind-spmf-def*[*abs-def*] **by** *transfer-prover*

lemma *return-spmf-parametric*: $(A\ ==\!>\ rel\text{-}spm\text{-}f\ A)\ return\text{-}spm\text{-}f\ return\text{-}spm\text{-}f$
by *transfer-prover*

lemma *map-spmf-parametric*: $((A\ ==\!>\ B)\ ==\!>\ rel\text{-}spm\text{-}f\ A\ ==\!>\ rel\text{-}spm\text{-}f\ B)\ map\text{-}spm\text{-}f\ map\text{-}spm\text{-}f$
by *transfer-prover*

lemma *rel-spmf-parametric*:
 $((A\ ==\!>\ B\ ==\!>\ (=))\ ==\!>\ rel\text{-}spm\text{-}f\ A\ ==\!>\ rel\text{-}spm\text{-}f\ B\ ==\!>\ (=))$
 $rel\text{-}spm\text{-}f\ rel\text{-}spm\text{-}f$
by *transfer-prover*

lemma *set-spmf-parametric* [*transfer-rule*]:
 $(rel\text{-}spm\text{-}f\ A\ ==\!>\ rel\text{-}set\ A)\ set\text{-}spm\text{-}f\ set\text{-}spm\text{-}f$
unfolding *set-spmf-def*[*abs-def*] **by** *transfer-prover*

lemma *return-spmf-None-parametric*:
 $(rel\text{-}spm\text{-}f\ A)\ (return\text{-}pmf\ None)\ (return\text{-}pmf\ None)$
by *simp*

end

lemma *rel-spmf-bindI*:
 $\llbracket rel\text{-}spm\text{-}f\ R\ p\ q;\ \bigwedge x\ y.\ R\ x\ y\ \Longrightarrow\ rel\text{-}spm\text{-}f\ P\ (f\ x)\ (g\ y)\ \rrbracket$
 $\Longrightarrow\ rel\text{-}spm\text{-}f\ P\ (p\ \gg\ f)\ (q\ \gg\ g)$
by(*fact bind-spmf-parametric*[*THEN rel-funD, THEN rel-funD, OF - rel-funI*])

lemma *rel-spmf-bind-reflI*:
 $(\bigwedge x.\ x \in set\text{-}spm\text{-}f\ p\ \Longrightarrow\ rel\text{-}spm\text{-}f\ P\ (f\ x)\ (g\ x))\ \Longrightarrow\ rel\text{-}spm\text{-}f\ P\ (p\ \gg\ f)\ (p\ \gg\ g)$
by(*rule rel-spmf-bindI*[**where** $R=\lambda x\ y.\ x = y \wedge x \in set\text{-}spm\text{-}f\ p$])(*auto intro: rel-spmf-reflI*)

lemma *rel-pmf-return-pmfI*: $P\ x\ y\ \Longrightarrow\ rel\text{-}pmf\ P\ (return\text{-}pmf\ x)\ (return\text{-}pmf\ y)$
by *simp*

context includes *lifting-syntax*
begin

We do not yet have a relator for '*a measure*', so we combine *Sigma-Algebra.measure* and *measure-pmf*

lemma *measure-pmf-parametric*:
 $(rel\text{-}pmf\ A\ ==\!>\ rel\text{-}pred\ A\ ==\!>\ (=))\ (\lambda p.\ measure\ (measure\text{-}pmf\ p))\ (\lambda q.\ measure\ (measure\text{-}pmf\ q))$

```

proof(rule rel-funI)+
  fix p q X Y
  assume rel-pmf A p q and rel-pred A X Y
  from this(1) obtain pq where A:  $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies A \ x \ y$ 
    and p: p = map-pmf fst pq and q: q = map-pmf snd pq by cases auto
  show measure p X = measure q Y unfolding p q measure-map-pmf
    by(rule measure-pmf.finite-measure-eq-AE)(auto simp: AE-measure-pmf-iff dest!:
A rel-predD[OF rel-pred - - -])
qed

```

lemma measure-spmf-parametric:

(rel-spmf A ==> rel-pred A ==> (=)) ($\lambda p. \text{measure } (\text{measure-spmf } p)$) ($\lambda q. \text{measure } (\text{measure-spmf } q)$)

proof –

have $\bigwedge x \ y \ x a \ y a. \text{rel-pred } A \ x a \ y a \implies \text{rel-pred } (\text{rel-option } A) \ (\text{Some } 'x a) \ (\text{Some } 'y a)$

by(auto simp: rel-pred-def rel-fun-def elim: option.rel-cases)

then show ?thesis

unfolding measure-measure-spmf-conv-measure-pmf[abs-def]

by (intro rel-funI) (force elim!: measure-pmf-parametric[THEN rel-funD, THEN rel-funD])

qed

end

25.7 From 'a pmf to 'a spmf

definition spmf-of-pmf :: 'a pmf \Rightarrow 'a spmf

where spmf-of-pmf = map-pmf Some

lemma set-spmf-spmf-of-pmf [simp]: set-spmf (spmf-of-pmf p) = set-pmf p

by(auto simp: spmf-of-pmf-def set-spmf-def bind-image o-def)

lemma spmf-spmf-of-pmf [simp]: spmf (spmf-of-pmf p) x = pmf p x

by(simp add: spmf-of-pmf-def)

lemma pmf-spmf-of-pmf-None [simp]: pmf (spmf-of-pmf p) None = 0

using ennreal-pmf-map[of Some p None] **by**(simp add: spmf-of-pmf-def)

lemma emeasure-spmf-of-pmf [simp]: emeasure (measure-spmf (spmf-of-pmf p))

A = emeasure (measure-pmf p) A

by(simp add: emeasure-measure-spmf-conv-measure-pmf spmf-of-pmf-def inj-vimage-image-eq)

lemma measure-spmf-spmf-of-pmf [simp]: measure-spmf (spmf-of-pmf p) = measure-pmf p

by(rule measure-eqI) simp-all

lemma map-spmf-of-pmf [simp]: map-spmf f (spmf-of-pmf p) = spmf-of-pmf (map-pmf f p)

by(*simp add: spmf-of-pmf-def pmf.map-comp o-def*)

lemma *rel-spmf-spmf-of-pmf* [*simp*]: *rel-spmf* *R* (*spmf-of-pmf* *p*) (*spmf-of-pmf* *q*)
 $= \text{rel-pmf } R \text{ } p \text{ } q$
by(*simp add: spmf-of-pmf-def pmf.rel-map*)

lemma *spmf-of-pmf-return-pmf* [*simp*]: *spmf-of-pmf* (*return-pmf* *x*) = *return-spmf* *x*
by(*simp add: spmf-of-pmf-def*)

lemma *bind-spmf-of-pmf* [*simp*]: *bind-spmf* (*spmf-of-pmf* *p*) *f* = *bind-pmf* *p* *f*
by(*simp add: spmf-of-pmf-def bind-spmf-def bind-map-pmf*)

lemma *set-spmf-bind-pmf*: *set-spmf* (*bind-pmf* *p* *f*) = *Set.bind* (*set-pmf* *p*) (*set-spmf* $\circ f$)
unfolding *bind-spmf-of-pmf*[*symmetric*] **by**(*subst set-bind-spmf*) *simp*

lemma *spmf-of-pmf-bind*: *spmf-of-pmf* (*bind-pmf* *p* *f*) = *bind-pmf* *p* ($\lambda x. \text{spmf-of-pmf } f \text{ } x$)
by(*simp add: spmf-of-pmf-def map-bind-pmf*)

lemma *bind-pmf-return-spmf*: $p \gg (\lambda x. \text{return-spmf } f \text{ } x) = \text{spmf-of-pmf } (\text{map-pmf } f \text{ } p)$
by(*simp add: map-pmf-def spmf-of-pmf-bind*)

25.8 Weight of a subprobability

abbreviation *weight-spmf* :: '*a* *spmf* \Rightarrow *real*'
where *weight-spmf* *p* $\equiv \text{measure } (\text{measure-spmf } p) (\text{space } (\text{measure-spmf } p))$

lemma *weight-spmf-def*: *weight-spmf* *p* = *measure* (*measure-spmf* *p*) *UNIV*
by(*simp add: space-measure-spmf*)

lemma *weight-spmf-le-1*: *weight-spmf* *p* ≤ 1
by(*rule measure-spmf.subprob-measure-le-1*)

lemma *weight-return-spmf* [*simp*]: *weight-spmf* (*return-spmf* *x*) = 1
by(*simp add: measure-spmf-return-spmf measure-return*)

lemma *weight-return-pmf-None* [*simp*]: *weight-spmf* (*return-pmf* *None*) = 0
by(*simp*)

lemma *weight-map-spmf* [*simp*]: *weight-spmf* (*map-spmf* *f* *p*) = *weight-spmf* *p*
by(*simp add: weight-spmf-def measure-map-spmf*)

lemma *weight-spmf-of-pmf* [*simp*]: *weight-spmf* (*spmf-of-pmf* *p*) = 1
by *simp*

lemma *weight-spmf-nonneg*: *weight-spmf* *p* ≥ 0

by(*fact measure-nonneg*)

lemma (*in finite-measure*) *integrable-weight-spmf* [*simp*]:

($\lambda x. \text{weight-spmf } (f \ x) \in \text{borel-measurable } M \implies \text{integrable } M \ (\lambda x. \text{weight-spmf } (f \ x))$)

by(*rule integrable-const-bound*[**where** $B=1$])(*simp-all add: weight-spmf-nonneg weight-spmf-le-1*)

lemma *weight-spmf-eq-nn-integral-spmf*: $\text{weight-spmf } p = \int^+ x. \text{spmfp } p \ x \ \partial \text{count-space UNIV}$

by (*metis NO-MATCH-def measure-spmf.emeasure-eq-measure nn-integral-count-space-indicator nn-integral-indicator nn-integral-measure-spmf sets-UNIV sets-measure-spmf space-measure-spmf*)

lemma *weight-spmf-eq-nn-integral-support*:

$\text{weight-spmf } p = \int^+ x. \text{spmfp } p \ x \ \partial \text{count-space } (\text{set-spmf } p)$

unfolding *weight-spmf-eq-nn-integral-spmf*

by(*auto simp: nn-integral-count-space-indicator in-set-spmf-iff-spmf intro!: nn-integral-cong split: split-indicator*)

lemma *pmf-None-eq-weight-spmf*: $\text{pmf } p \ \text{None} = 1 - \text{weight-spmf } p$

proof –

have $\text{weight-spmf } p = \int^+ x. \text{spmfp } p \ x \ \partial \text{count-space UNIV}$ **by**(*rule weight-spmf-eq-nn-integral-spmf*)

also have $\dots = \int^+ x. \text{ennreal } (\text{pmf } p \ x) * \text{indicator } (\text{range } \text{Some}) \ x \ \partial \text{count-space UNIV}$

by(*simp add: nn-integral-count-space-indicator[symmetric] embed-measure-count-space[symmetric] nn-integral-embed-measure measurable-embed-measure1*)

also have $\dots + \text{pmf } p \ \text{None} = \int^+ x. \text{ennreal } (\text{pmf } p \ x) * \text{indicator } (\text{range } \text{Some}) \ x + \text{ennreal } (\text{pmf } p \ \text{None}) * \text{indicator } \{\text{None}\} \ x \ \partial \text{count-space UNIV}$

by(*subst nn-integral-add*)(*simp-all add: max-def*)

also have $\dots = \int^+ x. \text{pmf } p \ x \ \partial \text{count-space UNIV}$

by(*rule nn-integral-cong*)(*auto split: split-indicator*)

also have $\dots = 1$ **by** (*simp add: nn-integral-pmf*)

finally show *?thesis* **by**(*simp add: ennreal-plus[symmetric] del: ennreal-plus*)

qed

lemma *weight-spmf-conv-pmf-None*: $\text{weight-spmf } p = 1 - \text{pmf } p \ \text{None}$

by(*simp add: pmf-None-eq-weight-spmf*)

lemma *weight-spmf-lt-0*: $\neg \text{weight-spmf } p < 0$

by(*simp add: not-less weight-spmf-nonneg*)

lemma *spmfp-le-weight*: $\text{spmfp } p \ x \leq \text{weight-spmf } p$

by (*simp add: measure-spmf.bounded-measure spmf-conv-measure-spmf*)

lemma *weight-spmf-eq-0*: $\text{weight-spmf } p = 0 \iff p = \text{return-pmf } \text{None}$

by (*metis measure-le-0-iff measure-spmf.bounded-measure spmf-conv-measure-spmf spmf-eqI weight-return-pmf-None*)

lemma *weight-bind-spmf*: $\text{weight-spmf } (x \ggg f) = \text{lebesgue-integral } (\text{measure-spmf}$

x) (*weight-spmf* $\circ f$)
unfolding *weight-spmf-def*
by (*simp add: measure-spmf-bind o-def measure-spmf.measure-bind* [**where** $N = \text{count-space UNIV}$])

lemma *rel-spmf-weightD*: $\text{rel-spmf } A \ p \ q \implies \text{weight-spmf } p = \text{weight-spmf } q$
by (*erule rel-spmfE*) *simp*

lemma *rel-spmf-bij-betw*:

assumes f : *bij-betw* f (*set-spmf* p) (*set-spmf* q)
and eq : $\bigwedge x. x \in \text{set-spmf } p \implies \text{spm} f \ p \ x = \text{spm} f \ q \ (f \ x)$
shows *rel-spmf* $(\lambda x \ y. f \ x = y) \ p \ q$

proof –

let $?f = \text{map-option } f$

have weq : $\text{ennreal } (\text{weight-spmf } p) = \text{ennreal } (\text{weight-spmf } q)$
unfolding *weight-spmf-eq-nn-integral-support*
by (*subst nn-integral-bij-count-space* [*OF* f , *symmetric*]) (*rule nn-integral-cong-AE*,
simp add: eq AE-count-space)
then have $\text{None} \in \text{set-pmf } p \longleftrightarrow \text{None} \in \text{set-pmf } q$
by (*simp add: pmf-None-eq-weight-spmf set-pmf-iff*)
with f **have** *bij-betw* (*map-option* f) (*set-pmf* p) (*set-pmf* q)
apply (*auto simp: bij-betw-def in-set-spmf inj-on-def intro: option.expand split: option.split*)
apply (*rename-tac* [$!$] x)
apply (*case-tac* [$!$] x)
apply (*auto iff: in-set-spmf*)
done

then have *rel-pmf* $(\lambda x \ y. ?f \ x = y) \ p \ q$
proof (*rule rel-pmf-bij-betw*)
show $\text{pmf } p \ x = \text{pmf } q \ (\text{map-option } f \ x)$ **if** $x \in \text{set-pmf } p$ **for** x
proof (*cases* x)
case None
then show $?thesis$
by (*metis ennreal-inj measure-nonneg option.map-disc-iff pmf-None-eq-weight-spmf weq*)
qed (*use eq in-set-spmf that in force*)
qed
thus $?thesis$
by (*smt (verit, ccv-SIG) None-eq-map-option-iff option.map-sel option.rel-sel pmf.rel-mono-strong*)
qed

25.9 From density to spmfs

context $\text{fixes } f :: 'a \Rightarrow \text{real}$ **begin**

definition *embed-spmf* $:: 'a \text{ spmf}$

where *embed-spmf* $= \text{embed-pmf } (\lambda x. \text{case } x \text{ of } \text{None} \Rightarrow 1 - \text{enn2real } (\int^+ x.$

$\text{ennreal } (f \ x) \ \partial \text{count-space } \text{UNIV}) \mid \text{Some } x' \Rightarrow \max 0 \ (f \ x')$

context

assumes $\text{prob}: (\int^+ x. \text{ennreal } (f \ x) \ \partial \text{count-space } \text{UNIV}) \leq 1$
begin

lemma *nn-integral-embed-spmf-eq-1*:

$(\int^+ x. \text{ennreal } (\text{case } x \text{ of } \text{None} \Rightarrow 1 - \text{enn2real } (\int^+ x. \text{ennreal } (f \ x) \ \partial \text{count-space } \text{UNIV}) \mid \text{Some } x' \Rightarrow \max 0 \ (f \ x')) \ \partial \text{count-space } \text{UNIV}) = 1$
 $(\text{is } ?lhs = - \text{ is } (\int^+ x. ?f \ x \ \partial ?M) = -)$

proof –

have $?lhs = \int^+ x. ?f \ x * \text{indicator } \{\text{None}\} \ x + ?f \ x * \text{indicator } (\text{range } \text{Some})$
 $x \ \partial ?M$

by $(\text{rule } \text{nn-integral-cong})(\text{auto } \text{split}: \text{split-indicator})$

also have $\dots = (1 - \text{enn2real } (\int^+ x. \text{ennreal } (f \ x) \ \partial \text{count-space } \text{UNIV})) +$
 $\int^+ x. ?f \ x * \text{indicator } (\text{range } \text{Some}) \ x \ \partial ?M$

(is $- = ?\text{None} + ?\text{Some}$)

by $(\text{subst } \text{nn-integral-add})(\text{simp-all } \text{add}: \text{AE-count-space max-def le-diff-eq real-le-ereal-iff}$
 $\text{one-ereal-def}[\text{symmetric}] \ \text{prob } \text{split}: \text{option.split})$

also have $?Some = \int^+ x. ?f \ x \ \partial \text{count-space } (\text{range } \text{Some})$

by $(\text{simp } \text{add}: \text{nn-integral-count-space-indicator})$

also have $\text{count-space } (\text{range } \text{Some}) = \text{embed-measure } (\text{count-space } \text{UNIV}) \ \text{Some}$

by $(\text{simp } \text{add}: \text{embed-measure-count-space})$

also have $(\int^+ x. ?f \ x \ \partial \dots) = \int^+ x. \text{ennreal } (f \ x) \ \partial \text{count-space } \text{UNIV}$

by $(\text{subst } \text{nn-integral-embed-measure})(\text{simp-all } \text{add}: \text{measurable-embed-measure1})$

also have $?None + \dots = 1$ **using** *prob*

by $(\text{auto } \text{simp}: \text{ennreal-minus}[\text{symmetric}] \ \text{ennreal-1}[\text{symmetric}] \ \text{ennreal-enn2real-if}$
 $\text{top-unique } \text{simp } \text{del}: \text{ennreal-1})(\text{simp } \text{add}: \text{diff-add-self-ennreal})$

finally show $?thesis$.

qed

lemma *pmf-embed-spmf-None*: $\text{pmf } \text{embed-spmf } \text{None} = 1 - \text{enn2real } (\int^+ x. \text{ennreal } (f \ x) \ \partial \text{count-space } \text{UNIV})$

unfolding *embed-spmf-def*

by $(\text{smt } (\text{verit}, \text{del-insts}) \ \text{enn2real-leI } \text{ennreal-1} \ \text{nn-integral-cong} \ \text{nn-integral-embed-spmf-eq-1}$
 $\text{option.case-eq-if } \text{pmf-embed-pmf } \text{prob})$

lemma *pmf-embed-spmf [simp]*: $\text{pmf } \text{embed-spmf } x = \max 0 \ (f \ x)$

unfolding *embed-spmf-def*

by $(\text{smt } (\text{verit}, \text{best}) \ \text{enn2real-leI } \text{ennreal-1} \ \text{nn-integral-cong} \ \text{nn-integral-embed-spmf-eq-1}$
 $\text{option.case-eq-if } \text{option.simps}(5) \ \text{pmf-embed-pmf } \text{prob})$

end

end

lemma *embed-spmf-K-0[simp]*: $\text{embed-spmf } (\lambda-. 0) = \text{return-pmf } \text{None}$

by $(\text{rule } \text{pmf-eqI})(\text{simp } \text{add}: \text{zero-ereal-def}[\text{symmetric}])$

25.10 Ordering on spmfs

rel-pmf does not preserve a ccpo structure. Counterexample by Saheb-Djahromi: Take prefix order over *bool llist* and the set *range* ($\lambda n :: \text{nat. uniform } (l\text{list-}n \ n)$) where *llist- n* is the set of all *llists* of length n and *uniform* returns a uniform distribution over the given set. The set forms a chain in *ord-pmf lprefix*, but it has not an upper bound. Any upper bound may contain only infinite lists in its support because otherwise it is not greater than the $n+1$ -st element in the chain where n is the length of the finite list. Moreover its support must contain all infinite lists, because otherwise there is a finite list all of whose finite extensions are not in the support - a contradiction to the upper bound property. Hence, the support is uncountable, but pmf's only have countable support.

However, if all chains in the ccpo are finite, then it should preserve the ccpo structure.

abbreviation *ord-spmf* :: ($'a \Rightarrow 'a \Rightarrow \text{bool}$) $\Rightarrow 'a \text{ spmf} \Rightarrow 'a \text{ spmf} \Rightarrow \text{bool}$
where *ord-spmf ord* $\equiv \text{rel-pmf } (\text{ord-option } \text{ord})$

locale *ord-spmf-syntax* **begin**

notation *ord-spmf* (**infix** $\langle \sqsubseteq_1 \rangle$ 60)

end

lemma *ord-spmf-map-spmf1*: *ord-spmf* *R* (*map-spmf* *f* *p*) = *ord-spmf* ($\lambda x. R \ (f \ x)$)
p

by(*simp add: pmf.rel-map[abs-def] ord-option-map1[abs-def]*)

lemma *ord-spmf-map-spmf2*: *ord-spmf* *R* *p* (*map-spmf* *f* *q*) = *ord-spmf* ($\lambda x \ y. R \ x \ (f \ y)$) *p* *q*

by(*simp add: pmf.rel-map ord-option-map2*)

lemma *ord-spmf-map-spmf12*: *ord-spmf* *R* (*map-spmf* *f* *p*) (*map-spmf* *f* *q*) = *ord-spmf* ($\lambda x \ y. R \ (f \ x) \ (f \ y)$) *p* *q*

by(*simp add: pmf.rel-map ord-option-map1[abs-def] ord-option-map2*)

lemmas *ord-spmf-map-spmf* = *ord-spmf-map-spmf1 ord-spmf-map-spmf2 ord-spmf-map-spmf12*

context **fixes** *ord* :: $'a \Rightarrow 'a \Rightarrow \text{bool}$ (**structure**) **begin**

interpretation *ord-spmf-syntax* .

lemma *ord-spmfI*:

$\llbracket \bigwedge x \ y. (x, y) \in \text{set-spmf } pq \implies \text{ord } x \ y; \text{map-spmf } \text{fst } pq = p; \text{map-spmf } \text{snd } pq = q \rrbracket$

$\implies p \sqsubseteq q$

by(*rule rel-pmf.intros[where pq=map-pmf ($\lambda x. \text{case } x \text{ of None} \Rightarrow (\text{None}, \text{None})$) | Some (a, b) \Rightarrow (Some a , Some b)) pq]*)

(*auto simp: pmf.map-comp o-def in-set-spmf split: option.splits intro: pmf.map-cong*)

lemma *ord-spmf-None* [*simp*]: *return-pmf None* \sqsubseteq *x*
by(*rule rel-pmf.intros*[**where** *pq=map-pmf (Pair None) x*])(*auto simp: pmf.map-comp o-def*)

lemma *ord-spmf-reflI*: $(\bigwedge x. x \in \text{set-spmf } p \implies \text{ord } x \ x) \implies p \sqsubseteq p$
by (*metis elem-set in-set-spmf ord-option-reflI pmf.rel-refl-strong*)

lemma *rel-spmf-inf*:
assumes $p \sqsubseteq q$
and $q \sqsubseteq p$
and *refl*: *reflp ord*
and *trans*: *transp ord*
shows *rel-spmf (inf ord ord⁻¹⁻¹) p q*
proof –
from $\langle p \sqsubseteq q \rangle \langle q \sqsubseteq p \rangle$
have *rel-pmf (inf (ord-option ord) (ord-option ord)⁻¹⁻¹) p q*
using *local.refl local.trans reflp-ord-option rel-pmf-inf transp-ord-option* **by** *blast*
also have $\text{inf (ord-option ord) (ord-option ord)}^{-1-1} = \text{rel-option (inf ord ord}^{-1-1})$
by(*auto simp: fun-eq-iff elim: ord-option.cases option.rel-cases*)
finally show *?thesis* .
qed

end

lemma *ord-spmf-return-spmf2*: *ord-spmf R p (return-spmf y)* \longleftrightarrow $(\forall x \in \text{set-spmf } p. R \ x \ y)$
by(*auto simp: rel-pmf-return-pmf2 in-set-spmf ord-option.simps intro: ccontr*)

lemma *ord-spmf-mono*: $\llbracket \text{ord-spmf } A \ p \ q; \bigwedge x \ y. A \ x \ y \implies B \ x \ y \rrbracket \implies \text{ord-spmf } B \ p \ q$
by(*erule pmf.rel-mono-strong*)(*erule ord-option-mono*)

lemma *ord-spmf-compp*: *ord-spmf (A OO B)* = *ord-spmf A OO ord-spmf B*
by(*simp add: ord-option-compp pmf.rel-compp*)

lemma *ord-spmf-bindI*:
assumes *pq: ord-spmf R p q*
and *fg*: $\bigwedge x \ y. R \ x \ y \implies \text{ord-spmf } P \ (f \ x) \ (g \ y)$
shows *ord-spmf P (p \gg f) (q \gg g)*
unfolding *bind-spmf-def* **using** *pq*
by(*rule rel-pmf-bindI*)(*auto split: option.split intro: fg*)

lemma *ord-spmf-bind-reflI*:
 $(\bigwedge x. x \in \text{set-spmf } p \implies \text{ord-spmf } R \ (f \ x) \ (g \ x)) \implies \text{ord-spmf } R \ (p \gg f) \ (p \gg g)$
by(*rule ord-spmf-bindI*[**where** $R = \lambda x \ y. x = y \wedge x \in \text{set-spmf } p$])(*auto intro: ord-spmf-reflI*)

lemma *ord-pmf-increaseI*:

```

assumes le:  $\bigwedge x. \text{pmf } p \ x \leq \text{pmf } q \ x$ 
and refl:  $\bigwedge x. x \in \text{set-spmf } p \implies R \ x \ x$ 
shows ord-spmf  $R \ p \ q$ 
proof(rule rel-pmf.intros)
  define pq where pq = embed-pmf
    ( $\lambda(x, y). \text{case } x \text{ of } \text{Some } x' \Rightarrow (\text{case } y \text{ of } \text{Some } y' \Rightarrow \text{if } x' = y' \text{ then } \text{pmf } p \ x' \\ \text{else } 0 \mid \text{None} \Rightarrow 0) \\ \mid \text{None} \Rightarrow (\text{case } y \text{ of } \text{None} \Rightarrow \text{pmf } q \ \text{None} \mid \text{Some } y' \Rightarrow \text{pmf } q \ y' - \text{pmf } p \ y')$ )
  (is - = embed-pmf ?f)
  have nonneg:  $\bigwedge xy. ?f \ xy \geq 0$ 
  by(clarsimp simp add: le field-simps split: option.split)
  have integral:  $(\int^+ xy. ?f \ xy \ \partial \text{count-space UNIV}) = 1$  (is nn-integral ?M - = -)
  proof -
    have  $(\int^+ xy. ?f \ xy \ \partial \text{count-space UNIV}) =$ 
       $\int^+ xy. \text{ennreal } (?f \ xy) * \text{indicator } \{(None, None)\} \ xy +$ 
       $\text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (None, \text{Some } x))) \ xy +$ 
       $\text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (\text{Some } x, \text{Some } x))) \ xy \ \partial ?M$ 
    by(rule nn-integral-cong)(auto split: split-indicator option.splits if-split-asm)
    also have ... =  $(\int^+ xy. ?f \ xy * \text{indicator } \{(None, None)\} \ xy \ \partial ?M) +$ 
       $(\int^+ xy. \text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (None, \text{Some } x))) \ xy \ \partial ?M)$ 
    +
       $(\int^+ xy. \text{ennreal } (?f \ xy) * \text{indicator } (\text{range } (\lambda x. (\text{Some } x, \text{Some } x))) \ xy \ \partial ?M)$ 
    (is - = ?None + ?Some2 + ?Some)
    by(subst nn-integral-add)(simp-all add: nn-integral-add AE-count-space le-diff-eq
le split: option.split)
    also have ?None = pmf q None by simp
    also have ?Some2 =  $\int^+ x. \text{ennreal } (\text{pmf } q \ x) - \text{pmf } p \ x \ \partial \text{count-space UNIV}$ 
    by(simp add: nn-integral-count-space-indicator[symmetric] embed-measure-count-space[symmetric]
inj-on-def nn-integral-embed-measure measurable-embed-measure1 ennreal-minus)
    also have ... =  $(\int^+ x. \text{pmf } q \ x \ \partial \text{count-space UNIV}) - (\int^+ x. \text{pmf } p \ x$ 
 $\partial \text{count-space UNIV})$ 
    (is - = ?Some2' - ?Some2'')
    by(subst nn-integral-diff)(simp-all add: le nn-integral-spmf-neq-top)
    also have ?Some =  $\int^+ x. \text{pmf } p \ x \ \partial \text{count-space UNIV}$ 
    by(simp add: nn-integral-count-space-indicator[symmetric] embed-measure-count-space[symmetric]
inj-on-def nn-integral-embed-measure measurable-embed-measure1)
    also have pmf q None + (?Some2' - ?Some2'') + ... = pmf q None +
?Some2'
    by(auto simp: diff-add-self-ennreal le intro!: nn-integral-mono)
    also have ... =  $\int^+ x. \text{ennreal } (\text{pmf } q \ x) * \text{indicator } \{None\} \ x + \text{ennreal } (\text{pmf}$ 
 $q \ x) * \text{indicator } (\text{range } \text{Some}) \ x \ \partial \text{count-space UNIV}$ 
    by(subst nn-integral-add)(simp-all add: nn-integral-count-space-indicator[symmetric]
embed-measure-count-space[symmetric] nn-integral-embed-measure measurable-embed-measure1)
    also have ... =  $\int^+ x. \text{pmf } q \ x \ \partial \text{count-space UNIV}$ 
    by(rule nn-integral-cong)(auto split: split-indicator)
    also have ... = 1
    by(simp add: nn-integral-pmf)
    finally show ?thesis .

```

```

qed
note  $f = \text{nonneg integral}$ 

{ fix  $x\ y$ 
  assume  $(x, y) \in \text{set-pmf } pq$ 
  hence  $?f\ (x, y) \neq 0$  unfolding  $pq\text{-def}$  by  $(\text{simp add: set-embed-pmf}[OF\ f])$ 
  then show  $\text{ord-option } R\ x\ y$ 
  by  $(\text{simp add: spmf-eq-0-set-spmf refl split: option.split-asm if-split-asm})$  }

have  $\text{weight-le: weight-spmf } p \leq \text{weight-spmf } q$ 
by  $(\text{subst ennreal-le-iff[symmetric]})(\text{auto simp: weight-spmf-eq-nn-integral-spmf}$ 
 $\text{intro!: nn-integral-mono le})$ 

show  $\text{map-pmf fst } pq = p$ 
proof  $(\text{rule pmf-eqI})$ 
  fix  $i :: 'a\ \text{option}$ 
  have  $bi: \text{bij-betw } (Pair\ i)\ UNIV\ (\text{fst} - \{i\})$ 
  by  $(\text{auto simp: bij-betw-def inj-on-def})$ 
  have  $\text{ennreal } (\text{pmf } (\text{map-pmf fst } pq)\ i) = (\int^+ y. \text{pmf } pq\ (i, y)\ \partial \text{count-space}$ 
 $UNIV)$ 
  unfolding  $pq\text{-def ennreal-pmf-map}$ 
  apply  $(\text{simp add: embed-pmf.rep-eq}[OF\ f]\ \text{o-def emeasure-density flip: nn-integral-count-space-indicator})$ 
  by  $(\text{smt (verit, best) nn-integral-bij-count-space [OF bi] integral nn-integral-cong}$ 
 $\text{nonneg pmf-embed-pmf})$ 
  also have  $\dots = \text{pmf } p\ i$ 
  proof  $(\text{cases } i)$ 
    case  $(Some\ x)$ 
    have  $(\int^+ y. \text{pmf } pq\ (Some\ x, y)\ \partial \text{count-space } UNIV) = \int^+ y. \text{pmf } p\ (Some\ x)$ 
 $* \text{indicator } \{Some\ x\}\ y\ \partial \text{count-space } UNIV$ 
    by  $(\text{rule nn-integral-cong})(\text{simp add: pq-def pmf-embed-pmf}[OF\ f]\ \text{split:}$ 
 $\text{option.split})$ 
    then show  $?thesis$  using  $Some$  by  $\text{simp}$ 
  next
    case  $None$ 
    have  $(\int^+ y. \text{pmf } pq\ (None, y)\ \partial \text{count-space } UNIV) =$ 
 $(\int^+ y. \text{ennreal } (\text{pmf } pq\ (None, Some\ (the\ y))) * \text{indicator } (\text{range } Some)$ 
 $y +$ 
 $\text{ennreal } (\text{pmf } pq\ (None, None)) * \text{indicator } \{None\}\ y\ \partial \text{count-space}$ 
 $UNIV)$ 
    by  $(\text{rule nn-integral-cong})(\text{auto split: split-indicator})$ 
    also have  $\dots = (\int^+ y. \text{ennreal } (\text{pmf } pq\ (None, Some\ (the\ y)))\ \partial \text{count-space}$ 
 $(\text{range } Some)) + \text{pmf } pq\ (None, None)$ 
    by  $(\text{subst nn-integral-add})(\text{simp-all add: nn-integral-count-space-indicator})$ 
    also have  $\dots = (\int^+ y. \text{ennreal } (\text{spmfs } q\ y) - \text{ennreal } (\text{spmfs } p\ y)\ \partial \text{count-space}$ 
 $UNIV) + \text{pmf } q\ None$ 
    by  $(\text{simp add: pq-def pmf-embed-pmf}[OF\ f]\ \text{embed-measure-count-space[symmetric]}$ 
 $\text{nn-integral-embed-measure measurable-embed-measure1 ennreal-minus})$ 
    also have  $(\int^+ y. \text{ennreal } (\text{spmfs } q\ y) - \text{ennreal } (\text{spmfs } p\ y)\ \partial \text{count-space}$ 
 $UNIV) =$ 

```

```

    (∫+ y. spmf q y ∂count-space UNIV) - (∫+ y. spmf p y ∂count-space
UNIV)
  by(subst nn-integral-diff)(simp-all add: AE-count-space le nn-integral-spmf-neq-top
split: split-indicator)
  also have ... = pmf p None - pmf q None
  by(simp add: pmf-None-eq-weight-spmf weight-spmf-eq-nn-integral-spmf[symmetric]
ennreal-minus)
  also have ... = ennreal (pmf p None) - ennreal (pmf q None) by(simp add:
ennreal-minus)
  finally show ?thesis using None weight-le
  by(auto simp: diff-add-self-ennreal pmf-None-eq-weight-spmf intro: en-
nreal-leI)
qed
finally show pmf (map-pmf fst pq) i = pmf p i by simp
qed

show map-pmf snd pq = q
proof(rule pmf-eqI)
  fix i :: 'a option
  have bi: bij-betw (λx. (x, i)) UNIV (snd - ' {i})
  by (auto simp: bij-betw-def inj-on-def)
  have ennreal (pmf (map-pmf snd pq) i) = (∫+ x. pmf pq (x, i) ∂count-space
UNIV)
  unfolding pq-def ennreal-pmf-map
  apply(simp add: embed-pmf.rep-eq[OF f] o-def emeasure-density nn-integral-count-space-indicator[symmetr
by (smt (verit, best) nn-integral-bij-count-space [OF bi] integral nn-integral-cong
nonneg pmf-embed-pmf)
  also have ... = ennreal (pmf q i)
  proof(cases i)
    case None
    have (∫+ x. pmf pq (x, None) ∂count-space UNIV) = ∫+ x. pmf q None *
indicator {None :: 'a option} x ∂count-space UNIV
    by(rule nn-integral-cong)(simp add: pq-def pmf-embed-pmf[OF f] split:
option.split)
    then show ?thesis using None by simp
  next
    case (Some y)
    have (∫+ x. pmf pq (x, Some y) ∂count-space UNIV) =
      (∫+ x. ennreal (pmf pq (x, Some y)) * indicator (range Some) x +
      ennreal (pmf pq (None, Some y)) * indicator {None} x ∂count-space
UNIV)
    by(rule nn-integral-cong)(auto split: split-indicator)
    also have ... = (∫+ x. ennreal (pmf pq (x, Some y)) * indicator (range
Some) x ∂count-space UNIV) + pmf pq (None, Some y)
    by(subst nn-integral-add)(simp-all)
    also have ... = (∫+ x. ennreal (spmf p y) * indicator {Some y} x ∂count-space
UNIV) + (spmf q y - spmf p y)
    by(auto simp: pq-def pmf-embed-pmf[OF f] one-ereal-def[symmetric] simp del:
nn-integral-indicator-singleton intro!: arg-cong2[where f=(+)] nn-integral-cong split:

```

```

option.split)
  also have ... = spmf q y by (simp add: ennreal-minus[symmetric] le)
  finally show ?thesis using Some by simp
qed
finally show pmf (map-pmf snd pq) i = pmf q i by simp
qed
qed

lemma ord-spmf-eq-leD:
  assumes ord-spmf (=) p q
  shows spmf p x ≤ spmf q x
proof (cases x ∈ set-spmf p)
  case False
  thus ?thesis by (simp add: in-set-spmf-iff-spmf)
next
  case True
  from assms obtain pq
  where pq:  $\bigwedge x y. (x, y) \in \text{set-pmf } pq \implies \text{ord-option } (=) x y$ 
  and p:  $p = \text{map-pmf fst } pq$ 
  and q:  $q = \text{map-pmf snd } pq$  by cases auto
  have ennreal (spmf p x) =  $\text{integral}^N pq (\text{indicator } (\text{fst} - ' \{ \text{Some } x \} ))$ 
  using p by (simp add: ennreal-pmf-map)
  also have ... =  $\text{integral}^N pq (\text{indicator } \{ (\text{Some } x, \text{Some } x) \} )$ 
  by (rule nn-integral-cong-AE) (auto simp: AE-measure-pmf-iff split: split-indicator
dest: pq)
  also have ... ≤  $\text{integral}^N pq (\text{indicator } (\text{snd} - ' \{ \text{Some } x \} ))$ 
  by (rule nn-integral-mono) simp
  also have ... = ennreal (spmf q x) using q by (simp add: ennreal-pmf-map)
  finally show ?thesis by simp
qed

lemma ord-spmf-eqD-set-spmf: ord-spmf (=) p q  $\implies \text{set-spmf } p \subseteq \text{set-spmf } q$ 
  by (metis ord-spmf-eq-leD pmf-le-0-iff spmf-eq-0-set-spmf subsetI)

lemma ord-spmf-eqD-emeasure:
  ord-spmf (=) p q  $\implies \text{emeasure } (\text{measure-spmf } p) A \leq \text{emeasure } (\text{measure-spmf } q) A$ 
  by (auto intro!: nn-integral-mono split: split-indicator dest: ord-spmf-eq-leD simp
add: nn-integral-measure-spmf nn-integral-indicator[symmetric])

lemma ord-spmf-eqD-measure-spmf: ord-spmf (=) p q  $\implies \text{measure-spmf } p \leq \text{measure-spmf } q$ 
  by (subst le-measure) (auto simp: ord-spmf-eqD-emeasure)

```

25.11 CCPO structure for the flat ccpo *ord-option* (=)

context fixes $Y :: 'a \text{ spmf set}$ begin

definition *lub-spmf* :: $'a \text{ spmf}$

where $\text{lub-spmf} = \text{embed-spmf } (\lambda x. \text{enn2real } (\text{SUP } p \in Y. \text{ennreal } (\text{spm} p x)))$
 — We go through ennreal to have a sensible definition even if Y is empty.

lemma lub-spmf-empty [simp]: $\text{SPMF.lub-spmf } \{\} = \text{return-pmf None}$
by(simp add: $\text{SPMF.lub-spmf-def bot-ereal-def}$)

context assumes chain : $\text{Complete-Partial-Order.chain } (\text{ord-spmf } (=)) \ Y$
begin

lemma $\text{chain-ord-spmf-eqD}$: $\text{Complete-Partial-Order.chain } (\leq) ((\lambda p x. \text{ennreal } (\text{spm} p x)) \ ' Y)$

(is $\text{Complete-Partial-Order.chain } - (\text{?f } ' -)$)

proof(rule chainI)

fix $f g$

assume $f \in \text{?f } ' Y \ g \in \text{?f } ' Y$

then obtain $p \ q$ **where** $f: f = \text{?f } p \ p \in Y$ **and** $g: g = \text{?f } q \ q \in Y$ **by** blast

from $\text{chain } \langle p \in Y \rangle \langle q \in Y \rangle$ **have** $\text{ord-spmf } (=) \ p \ q \vee \text{ord-spmf } (=) \ q \ p$ **by**(rule chainD)

thus $f \leq g \vee g \leq f$

by ($\text{metis ennreal-leI f(1) g(1) le-funI ord-spmf-eq-leD}$)

qed

lemma $\text{ord-spmf-eq-pmf-None-eq}$:

assumes le : $\text{ord-spmf } (=) \ p \ q$

and None : $\text{pmf } p \ \text{None} = \text{pmf } q \ \text{None}$

shows $p = q$

proof(rule $\text{spm} \text{-eqI}$)

fix i

from le **have** le' : $\bigwedge x. \text{spm} p \ x \leq \text{spm} q \ x$ **by**(rule ord-spmf-eq-leD)

have $(\int^+ x. \text{ennreal } (\text{spm} q \ x) - \text{spm} p \ x \ \partial \text{count-space UNIV}) =$

$(\int^+ x. \text{spm} q \ x \ \partial \text{count-space UNIV}) - (\int^+ x. \text{spm} p \ x \ \partial \text{count-space UNIV})$

by(subst nn-integral-diff)(simp-all add: $\text{AE-count-space le' nn-integral-spmf-neq-top}$)

also have $\dots = (1 - \text{pmf } q \ \text{None}) - (1 - \text{pmf } p \ \text{None})$ **unfolding** $\text{pmf-None-eq-weight-spmf}$

by(simp add: $\text{weight-spmf-eq-nn-integral-spmf[symmetric] ennreal-minus}$)

also have $\dots = 0$ **using** None **by** simp

finally have $\bigwedge x. \text{spm} q \ x \leq \text{spm} p \ x$

by(simp add: $\text{nn-integral-0-iff-AE AE-count-space ennreal-minus ennreal-eq-0-iff}$)

with le' **show** $\text{spm} p \ i = \text{spm} q \ i$ **by**(rule antisym)

qed

lemma $\text{ord-spmf-eqD-pmf-None}$:

assumes $\text{ord-spmf } (=) \ x \ y$

shows $\text{pmf } x \ \text{None} \geq \text{pmf } y \ \text{None}$

using assms

apply cases

apply($\text{clarsimp simp only: ennreal-le-iff[symmetric, OF pmf-nonneg] ennreal-pmf-map}$)

apply($\text{fastforce simp: AE-measure-pmf-iff intro!: nn-integral-mono-AE}$)

done

Chains on $'a \ \text{spm}$ maintain countable support. Thanks to Johannes Hölzl

for the proof idea.

lemma *spmf-chain-countable*: *countable* ($\bigcup p \in Y. \text{set-spmf } p$)

proof(*cases* $Y = \{\}$)

case $Y: \text{False}$

show *?thesis*

proof(*cases* $\exists x \in Y. \forall y \in Y. \text{ord-spmf } (=) y x$)

case *True*

then obtain x **where** $x: x \in Y$ **and** $\text{upper}: \bigwedge y. y \in Y \implies \text{ord-spmf } (=) y x$

by *blast*

hence $(\bigcup x \in Y. \text{set-spmf } x) \subseteq \text{set-spmf } x$ **by**(*auto dest: ord-spmf-eqD-set-spmf*)

thus *?thesis* **by**(*rule countable-subset*) *simp*

next

case *False*

define $N :: 'a \text{ option pmf} \Rightarrow \text{real}$ **where** $N p = \text{pmf } p \text{ None for } p$

have *N-less-imp-le-spmf*: $\llbracket x \in Y; y \in Y; N y < N x \rrbracket \implies \text{ord-spmf } (=) x y$

for $x y$

using *chainD[OF chain, of x y] ord-spmf-eqD-pmf-None[of x y] ord-spmf-eqD-pmf-None[of y x]*

by (*auto simp: N-def*)

have *N-eq-imp-eq*: $\llbracket x \in Y; y \in Y; N y = N x \rrbracket \implies x = y$ **for** $x y$

using *chainD[OF chain, of x y] by(auto simp: N-def dest: ord-spmf-eq-pmf-None-eq)*

have *NC*: $N \text{ ' } Y \neq \{\}$ *bdd-below* ($N \text{ ' } Y$)

using $\langle Y \neq \{\} \rangle$ **by**(*auto intro!: bdd-belowI[of - 0] simp: N-def*)

have *NC-less*: $\text{Inf } (N \text{ ' } Y) < N x$ **if** $x \in Y$ **for** x **unfolding** *cInf-less-iff[OF NC]*

proof(*rule ccontr*)

assume $**$: $\neg (\exists y \in N \text{ ' } Y. y < N x)$

{ fix y

assume $y \in Y$

with $**$ **consider** $N x < N y \mid N x = N y$ **by**(*auto simp: not-less le-less*)

hence $\text{ord-spmf } (=) y x$ **using** $\langle y \in Y \rangle \langle x \in Y \rangle$

by *cases(auto dest: N-less-imp-le-spmf N-eq-imp-eq intro: ord-spmf-refl)* **}**

with *False* $\langle x \in Y \rangle$ **show** *False* **by** *blast*

qed

from *NC* **have** $\text{Inf } (N \text{ ' } Y) \in \text{closure } (N \text{ ' } Y)$ **by** (*intro closure-contains-Inf*)

then obtain X' **where** $\bigwedge n. X' n \in N \text{ ' } Y$ **and** $X': X' \longrightarrow \text{Inf } (N \text{ ' } Y)$

unfolding *closure-sequential* **by** *auto*

then obtain X **where** $X: \bigwedge n. X n \in Y$ **and** $X' = (\lambda n. N (X n))$ **unfolding** *image-iff Bex-def* **by** *metis*

with X' **have** $\text{seq: } (\lambda n. N (X n)) \longrightarrow \text{Inf } (N \text{ ' } Y)$ **by** *simp*

have $(\bigcup x \in Y. \text{set-spmf } x) \subseteq (\bigcup n. \text{set-spmf } (X n))$

proof(*rule UN-least*)

fix x

assume $x \in Y$

from *order-tendstoD(2)[OF seq NC-less[OF $\langle x \in Y \rangle$]]*

```

    obtain  $i$  where  $N (X i) < N x$  by (auto simp: eventually-sequentially)
    thus  $\text{set-spmf } x \subseteq (\bigcup n. \text{set-spmf } (X n))$  using  $X \langle x \in Y \rangle$ 
    by (blast dest: N-less-imp-le-spmf ord-spmf-eqD-set-spmf)
  qed
  thus ?thesis by (rule countable-subset) simp
qed
qed simp

lemma lub-spmf-subprob:  $(\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmfmf } p x)) \partial \text{count-space UNIV}) \leq 1$ 
proof (cases  $Y = \{\}$ )
  case True
    thus ?thesis by (simp add: bot-ennreal)
  next
    case False
    let  $?B = \bigcup p \in Y. \text{set-spmf } p$ 
    have countable: countable ?B by (rule spmf-chain-countable)

    have  $(\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmfmf } p x)) \partial \text{count-space UNIV}) =$ 
       $(\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmfmf } p x) * \text{indicator } ?B x) \partial \text{count-space UNIV})$ 
      by (intro nn-integral-cong arg-cong [of - - Sup]) (auto split: split-indicator simp
add: spmf-eq-0-set-spmf)
    also have  $\dots = (\int^+ x. (\text{SUP } p \in Y. \text{ennreal } (\text{spmfmf } p x)) \partial \text{count-space } ?B)$ 
      unfolding ennreal-indicator[symmetric] using False
      by (subst SUP-mult-right-ennreal[symmetric]) (simp add: ennreal-indicator nn-integral-count-space-indicator)
    also have  $\dots = (\text{SUP } p \in Y. \int^+ x. \text{spmfmf } p x \partial \text{count-space } ?B)$  using False -
countable
      by (rule nn-integral-monotone-convergence-SUP-countable) (rule chain-ord-spmf-eqD)
    also have  $\dots \leq 1$ 
    proof (rule SUP-least)
      fix  $p$ 
      assume  $p \in Y$ 
      have  $(\int^+ x. \text{spmfmf } p x \partial \text{count-space } ?B) = \int^+ x. \text{ennreal } (\text{spmfmf } p x) * \text{indicator}$ 
       $?B x \partial \text{count-space UNIV}$ 
      by (simp add: nn-integral-count-space-indicator)
    also have  $\dots = \int^+ x. \text{spmfmf } p x \partial \text{count-space UNIV}$ 
    by (rule nn-integral-cong) (auto split: split-indicator simp add: spmf-eq-0-set-spmf
 $\langle p \in Y \rangle$ )
    also have  $\dots \leq 1$ 
    by (simp add: weight-spmf-eq-nn-integral-spmf[symmetric] weight-spmf-le-1)
    finally show  $(\int^+ x. \text{spmfmf } p x \partial \text{count-space } ?B) \leq 1$  .
  qed
  finally show ?thesis .
qed

```

lemma *spmfmf-lub-spmfmf*:

```

  assumes  $Y \neq \{\}$ 
  shows  $\text{spmfmf lub-spmfmf } x = (\text{SUP } p \in Y. \text{spmfmf } p x)$ 
proof -

```

from *assms* **obtain** p **where** $p \in Y$ **by** *auto*
have $\text{spmf lub-spmf } x = \max 0 (\text{enn2real } (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x)))$
unfolding *lub-spmf-def*
by(*rule* *spmf-embed-spmf*)(*simp* *del*: *SUP-eq-top-iff* *Sup-eq-top-iff* *add*: *ennreal-enn2real-if* *SUP-spmf-neq-top'* *lub-spmf-subprob*)
also have $\dots = \text{enn2real } (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x))$
by(*rule* *max-absorb2*)(*simp*)
also have $\dots = \text{enn2real } (\text{ennreal } (\text{SUP } p \in Y. \text{spmf } p \ x))$ **using** *assms*
by(*subst* *ennreal-SUP[symmetric]*)(*simp-all* *add*: *SUP-spmf-neq-top'* *del*: *SUP-eq-top-iff* *Sup-eq-top-iff*)
also have $0 \leq (\bigsqcup p \in Y. \text{spmf } p \ x)$ **using** *assms*
by(*auto* *intro!*: *cSUP-upper2* *bdd-aboveI*[**where** $M=1$] *simp* *add*: *pmf-le-1*)
then have $\text{enn2real } (\text{ennreal } (\text{SUP } p \in Y. \text{spmf } p \ x)) = (\text{SUP } p \in Y. \text{spmf } p \ x)$
by(*rule* *enn2real-ennreal*)
finally show *?thesis* .
qed

lemma *ennreal-spmf-lub-spmf*: $Y \neq \{\}$ $\implies \text{ennreal } (\text{spmf lub-spmf } x) = (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x))$
by (*metis* *SUP-spmf-neq-top'* *ennreal-SUP* *spmf-lub-spmf*)

lemma *lub-spmf-upper*:
assumes $p: p \in Y$
shows $\text{ord-spmf } (=) \ p \ \text{lub-spmf}$
proof(*rule* *ord-pmf-increaseI*)
fix x
from p **have** [*simp*]: $Y \neq \{\}$ **by** *auto*
from p **have** $\text{ennreal } (\text{spmf } p \ x) \leq (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x))$ **by**(*rule* *SUP-upper*)
also have $\dots = \text{ennreal } (\text{spmf lub-spmf } x)$ **using** p
by(*subst* *spmf-lub-spmf*)(*auto* *simp*: *ennreal-SUP* *SUP-spmf-neq-top'* *simp* *del*: *SUP-eq-top-iff* *Sup-eq-top-iff*)
finally show $\text{spmf } p \ x \leq \text{spmf lub-spmf } x$ **by** *simp*
qed *simp*

lemma *lub-spmf-least*:
assumes $z: \bigwedge x. x \in Y \implies \text{ord-spmf } (=) \ x \ z$
shows $\text{ord-spmf } (=) \ \text{lub-spmf } z$
proof(*cases* $Y = \{\}$)
case *nonempty*: *False*
show *?thesis*
proof(*rule* *ord-pmf-increaseI*)
fix x
from *nonempty* **obtain** p **where** $p: p \in Y$ **by** *auto*
have $\text{ennreal } (\text{spmf lub-spmf } x) = (\text{SUP } p \in Y. \text{ennreal } (\text{spmf } p \ x))$
by(*subst* *spmf-lub-spmf*)(*auto* *simp*: *ennreal-SUP* *SUP-spmf-neq-top'* *nonempty* *simp* *del*: *SUP-eq-top-iff* *Sup-eq-top-iff*)
also have $\dots \leq \text{ennreal } (\text{spmf } z \ x)$ **by**(*rule* *SUP-least*)(*simp* *add*: *ord-spmf-eq-leD* z)

finally show $\text{spmf lub-spmf } x \leq \text{spmf } z \text{ } x$ **by** *simp*
qed *simp*
qed *simp*

lemma *set-lub-spmf*: $\text{set-spmf lub-spmf} = (\bigcup p \in Y. \text{set-spmf } p)$ (**is** $?lhs = ?rhs$)

proof(*cases* $Y = \{\}$)

case [*simp*]: *False*

show $?thesis$

proof(*rule set-eqI*)

fix x

have $x \in ?lhs \longleftrightarrow \text{ennreal } (\text{spmf lub-spmf } x) > 0$

by(*simp-all add: in-set-spmf-iff-spmf less-le*)

also have $\dots \longleftrightarrow (\exists p \in Y. \text{ennreal } (\text{spmf } p \text{ } x) > 0)$

by(*simp add: ennreal-spmf-lub-spmf less-SUP-iff*)

also have $\dots \longleftrightarrow x \in ?rhs$

by(*auto simp: in-set-spmf-iff-spmf less-le*)

finally show $x \in ?lhs \longleftrightarrow x \in ?rhs$.

qed

qed *simp*

lemma *emeasure-lub-spmf*:

assumes $Y: Y \neq \{\}$

shows $\text{emeasure } (\text{measure-spmf lub-spmf}) \text{ } A = (\text{SUP } y \in Y. \text{emeasure } (\text{measure-spmf } y) \text{ } A)$

(**is** $?lhs = ?rhs$)

proof –

let $?M = \text{count-space } (\text{set-spmf lub-spmf})$

have $?lhs = \int^+ x. \text{ennreal } (\text{spmf lub-spmf } x) * \text{indicator } A \text{ } x \text{ } \partial ?M$

by(*auto simp: nn-integral-indicator[symmetric] nn-integral-measure-spmf'*)

also have $\dots = \int^+ x. (\text{SUP } y \in Y. \text{ennreal } (\text{spmf } y \text{ } x) * \text{indicator } A \text{ } x) \text{ } \partial ?M$

unfolding *ennreal-indicator[symmetric]*

by(*simp add: spmf-lub-spmf assms ennreal-SUP[OF SUP-spmf-neq-top'] SUP-mult-right-ennreal*)

also from *assms* **have** $\dots = (\text{SUP } y \in Y. \int^+ x. \text{ennreal } (\text{spmf } y \text{ } x) * \text{indicator } A \text{ } x \text{ } \partial ?M)$

proof(*rule nn-integral-monotone-convergence-SUP-countable*)

have $(\lambda i \text{ } x. \text{ennreal } (\text{spmf } i \text{ } x) * \text{indicator } A \text{ } x) \text{ } ' Y = (\lambda f \text{ } x. f \text{ } x * \text{indicator } A \text{ } x) \text{ } ' (\lambda p \text{ } x. \text{ennreal } (\text{spmf } p \text{ } x)) \text{ } ' Y$

by(*simp add: image-image*)

also have *Complete-Partial-Order.chain* $(\leq) \dots$ **using** *chain-ord-spmf-eqD*

by(*rule chain-imageI*)(*auto simp: le-fun-def split: split-indicator*)

finally show *Complete-Partial-Order.chain* $(\leq) ((\lambda i \text{ } x. \text{ennreal } (\text{spmf } i \text{ } x) * \text{indicator } A \text{ } x) \text{ } ' Y)$.

qed *simp*

also have $\dots = (\text{SUP } y \in Y. \int^+ x. \text{ennreal } (\text{spmf } y \text{ } x) * \text{indicator } A \text{ } x \text{ } \partial \text{count-space UNIV})$

by(*auto simp: nn-integral-count-space-indicator set-lub-spmf spmf-eq-0-set-spmf split: split-indicator intro!: arg-cong [of - - Sup] image-cong nn-integral-cong*)

also have $\dots = ?rhs$

by(*auto simp: nn-integral-indicator[symmetric] nn-integral-measure-spmf*)

finally show *?thesis* .
qed

lemma *measure-lub-spmf*:
assumes $Y: Y \neq \{\}$
shows $\text{measure } (\text{measure-spmf } \text{lub-spmf}) A = (\text{SUP } y \in Y. \text{measure } (\text{measure-spmf } y) A)$ (**is** *?lhs = ?rhs*)
proof –
have *ennreal ?lhs = ennreal ?rhs*
using *emeasure-lub-spmf[OF assms] SUP-emeasure-spmf-neq-top[of A Y] Y*
unfolding *measure-spmf.emeasure-eq-measure* **by**(*subst ennreal-SUP*)
moreover have $0 \leq ?rhs$ **using** *Y*
by(*auto intro!: cSUP-upper2 bdd-aboveI[where M=1] measure-spmf.subprob-measure-le-1*)
ultimately show *?thesis* **by**(*simp*)
qed

lemma *weight-lub-spmf*:
assumes $Y: Y \neq \{\}$
shows $\text{weight-spmf } \text{lub-spmf} = (\text{SUP } y \in Y. \text{weight-spmf } y)$
by (*smt (verit, best) SUP-cong assms measure-lub-spmf space-measure-spmf*)

lemma *measure-spmf-lub-spmf*:
assumes $Y: Y \neq \{\}$
shows $\text{measure-spmf } \text{lub-spmf} = (\text{SUP } p \in Y. \text{measure-spmf } p)$ (**is** *?lhs = ?rhs*)
proof(*rule measure-eqI*)
from *assms* **obtain** p **where** $p: p \in Y$ **by** *auto*
from *chain* **have** *chain'*: *Complete-Partial-Order.chain* (\leq) (*measure-spmf* ‘ Y)
by(*rule chain-imageI*)(*rule ord-spmf-eqD-measure-spmf*)
show *sets ?lhs = sets ?rhs*
using *Y* **by** (*subst sets-SUP*) *auto*
show *emeasure ?lhs A = emeasure ?rhs A* **for** A
using *chain' Y p* **by** (*subst emeasure-SUP-chain*) (*auto simp: emeasure-lub-spmf*)
qed

end

end

lemma *partial-function-definitions-spmf*: *partial-function-definitions* (*ord-spmf* (=))
lub-spmf
(**is** *partial-function-definitions ?R -*)
proof
fix x **show** $?R x x$ **by**(*simp add: ord-spmf-refl*)
next
fix $x y z$
assume $?R x y \ ?R y z$
with *transp-ord-option[OF transp-equality]* **show** $?R x z$ **by**(*rule transp-rel-pmf[THEN transpD]*)
next

```

fix  $x\ y$ 
assume  $?R\ x\ y\ ?R\ y\ x$ 
thus  $x = y$ 
  by(rule rel-pmf-antisym)(simp-all add: reflp-ord-option transp-ord-option anti-symp-ord-option)
next
  fix  $Y\ x$ 
  assume Complete-Partial-Order.chain  $?R\ Y\ x \in Y$ 
  then show  $?R\ x\ (\text{lub-spmf } Y)$ 
    by(rule lub-spmf-upper)
next
  fix  $Y\ z$ 
  assume Complete-Partial-Order.chain  $?R\ Y\ \bigwedge x. x \in Y \implies ?R\ x\ z$ 
  then show  $?R\ (\text{lub-spmf } Y)\ z$ 
    by(cases  $Y = \{\}$ )(simp-all add: lub-spmf-least)
qed

lemma ccpo-spmf: class.ccpo lub-spmf (ord-spmf (=)) (mk-less (ord-spmf (=)))
  by(metis ccpo partial-function-definitions-spmf)

interpretation spmf: partial-function-definitions ord-spmf (=) lub-spmf
  rewrites lub-spmf  $\{\} \equiv \text{return-pmf } \text{None}$ 
  by(rule partial-function-definitions-spmf) simp

declaration  $\langle \text{Partial-Function.init } \text{spmf } \textbf{term} \langle \text{spmf.fixp-fun} \rangle$ 
   $\textbf{term} \langle \text{spmf.mono-body} \rangle @ \{\text{thm } \text{spmf.fixp-rule-uc}\} @ \{\text{thm } \text{spmf.fixp-induct-uc}\}$ 
   $\text{NONE} \rangle$ 

declare spmf.leq-refl[simp]
declare admissible-leI[OF ccpo-spmf, cont-intro]

abbreviation mono-spmf  $\equiv \text{monotone } (\text{fun-ord } (\text{ord-spmf } (=))) (\text{ord-spmf } (=))$ 

lemma lub-spmf-const [simp]: lub-spmf  $\{p\} = p$ 
  by(rule spmf-eqI)(simp add: spmf-lub-spmf[OF ccpo.chain-singleton[OF ccpo-spmf]])

lemma bind-spmf-mono':
  assumes  $fg: \text{ord-spmf } (=)\ f\ g$ 
  and  $hk: \bigwedge x :: 'a. \text{ord-spmf } (=)\ (h\ x)\ (k\ x)$ 
  shows  $\text{ord-spmf } (=)\ (f \gg h)\ (g \gg k)$ 
  unfolding bind-spmf-def using assms(1)
  by(rule rel-pmf-bindI)(auto split: option.split simp add: hk)

lemma bind-spmf-mono [partial-function-mono]:
  assumes  $mf: \text{mono-spmf } B$  and  $mg: \bigwedge y. \text{mono-spmf } (\lambda f. C\ y\ f)$ 
  shows  $\text{mono-spmf } (\lambda f. \text{bind-spmf } (B\ f)\ (\lambda y. C\ y\ f))$ 
proof (rule monotoneI)
  fix  $f\ g :: 'a \Rightarrow 'b\ \text{spmf}$ 
  assume  $fg: \text{fun-ord } (\text{ord-spmf } (=))\ f\ g$ 

```

with mf **have** $\text{ord-spmf} (=) (B f) (B g)$ **by** (*rule monotoneD*[*of - - f g*])
moreover from mg **have** $\bigwedge y'. \text{ord-spmf} (=) (C y' f) (C y' g)$
by (*rule monotoneD*) (*rule fg*)
ultimately show $\text{ord-spmf} (=) (\text{bind-spmf} (B f) (\lambda y. C y f)) (\text{bind-spmf} (B g) (\lambda y'. C y' g))$
by(*rule bind-spmf-mono'*)
qed

lemma *monotone-bind-spmf1*: *monotone* ($\text{ord-spmf} (=)$) ($\text{ord-spmf} (=)$) ($\lambda y. \text{bind-spmf} y g$)
by(*rule monotoneI*)(*simp add: bind-spmf-mono' ord-spmf-reflI*)

lemma *monotone-bind-spmf2*:
assumes $g: \bigwedge x. \text{monotone ord} (\text{ord-spmf} (=)) (\lambda y. g y x)$
shows *monotone ord* ($\text{ord-spmf} (=)$) ($\lambda y. \text{bind-spmf} p (g y)$)
by(*rule monotoneI*)(*auto intro: bind-spmf-mono' monotoneD[OF g] ord-spmf-reflI*)

lemma *bind-lub-spmf*:
assumes *chain*: *Complete-Partial-Order.chain* ($\text{ord-spmf} (=)$) Y
shows $\text{bind-spmf} (\text{lub-spmf } Y) f = \text{lub-spmf} ((\lambda p. \text{bind-spmf } p f) ' Y)$ (*is ?lhs = ?rhs*)
proof(*cases* $Y = \{\}$)
case $Y: \text{False}$
show *?thesis*
proof(*rule spmf-eqI*)
fix i
have *chain'*: *Complete-Partial-Order.chain* (\leq) ($(\lambda p x. \text{ennreal} (\text{spmf } p x * \text{spmf} (f x) i)) ' Y$)
using *chain* **by**(*rule chain-imageI*)(*auto simp: le-fun-def dest: ord-spmf-eq-leD intro: mult-right-mono*)
have *chain''*: *Complete-Partial-Order.chain* ($\text{ord-spmf} (=)$) ($(\lambda p. p \ggg f) ' Y$)
using *chain* **by**(*rule chain-imageI*)(*auto intro!: monotoneI bind-spmf-mono' ord-spmf-reflI*)
let $?M = \text{count-space} (\text{set-spmf} (\text{lub-spmf } Y))$
have $\text{ennreal} (\text{spmf } ?lhs i) = \int^+ x. \text{ennreal} (\text{spmf} (\text{lub-spmf } Y) x) * \text{ennreal} (\text{spmf} (f x) i) \partial ?M$
by(*auto simp: ennreal-spmf-lub-spmf ennreal-spmf-bind nn-integral-measure-spmf'*)
also have $\dots = \int^+ x. (\text{SUP } p \in Y. \text{ennreal} (\text{spmf } p x * \text{spmf} (f x) i)) \partial ?M$
by(*subst ennreal-spmf-lub-spmf[OF chain Y](subst SUP-mult-right-ennreal, simp-all add: ennreal-mult Y)*)
also have $\dots = (\text{SUP } p \in Y. \int^+ x. \text{ennreal} (\text{spmf } p x * \text{spmf} (f x) i) \partial ?M)$
using Y *chain'* **by**(*rule nn-integral-monotone-convergence-SUP-countable*)
simp
also have $\dots = (\text{SUP } p \in Y. \text{ennreal} (\text{spmf} (\text{bind-spmf } p f) i))$
by(*auto simp: ennreal-spmf-bind nn-integral-measure-spmf nn-integral-count-space-indicator set-lub-spmf[OF chain] in-set-spmf-iff-spmf ennreal-mult intro!: arg-cong [of - - Sup] image-cong nn-integral-cong split: split-indicator*)
also have $\dots = \text{ennreal} (\text{spmf } ?rhs i)$ **using** *chain''* **by**(*simp add: ennreal-spmf-lub-spmf Y image-comp*)

finally show $\text{spmf } ?\text{lhs } i = \text{spmf } ?\text{rhs } i$ **by** *simp*
qed
qed *simp*

lemma *map-lub-spmf*:

Complete-Partial-Order.chain (ord-spmf (=)) Y
 $\implies \text{map-spmf } f \text{ (lub-spmf } Y) = \text{lub-spmf (map-spmf } f \text{ ‘ } Y)$
unfolding *map-spmf-conv-bind-spmf[abs-def]* **by**(*simp add: bind-lub-spmf o-def*)

lemma *mcont-bind-spmf1*: *mcont lub-spmf (ord-spmf (=)) lub-spmf (ord-spmf (=))* $(\lambda y. \text{bind-spmf } y \text{ } f)$
using *monotone-bind-spmf1*
by(*intro contI mcontI*) (*auto simp: bind-lub-spmf*)

lemma *bind-lub-spmf2*:

assumes *chain*: *Complete-Partial-Order.chain ord Y*
and $g: \bigwedge y. \text{monotone ord (ord-spmf (=)) (g } y)$
shows $\text{bind-spmf } x \text{ (}\lambda y. \text{lub-spmf (g } y \text{ ‘ } Y)) = \text{lub-spmf ((}\lambda p. \text{bind-spmf } x \text{ (}\lambda y. \text{g } y \text{ } p)) \text{ ‘ } Y)$
(is $?\text{lhs} = ?\text{rhs}$ **)**
proof(*cases Y = {}*)
case *Y: False*
show *?thesis*
proof(*rule spmf-eqI*)
fix *i*
have *chain'*: $\bigwedge y. \text{Complete-Partial-Order.chain (ord-spmf (=)) (g } y \text{ ‘ } Y)$
using *chain g[THEN monotoneD]* **by**(*rule chain-imageI*)
have *chain''*: *Complete-Partial-Order.chain (\leq) ((}\lambda p y. \text{ennreal (spmf } x \text{ } y * \text{spmf (g } y \text{ } p) } i)) \text{ ‘ } Y)
using *chain* **by**(*rule chain-imageI*)(*auto simp: le-fun-def dest: ord-spmf-eq-leD monotoneD[OF g] intro!: mult-left-mono*)
have *chain'''*: *Complete-Partial-Order.chain (ord-spmf (=)) ((}\lambda p. \text{bind-spmf } x \text{ (}\lambda y. \text{g } y \text{ } p)) \text{ ‘ } Y)
using *chain* **by**(*rule chain-imageI*)(*rule monotone-bind-spmf2[OF g, THEN monotoneD]*)**

have $\text{ennreal (spmf } ?\text{lhs } i) = \int^+ y. (\text{SUP } p \in Y. \text{ennreal (spmf } x \text{ } y * \text{spmf (g } y \text{ } p) } i)) \text{ } \partial \text{count-space (set-spmf } x)$
by(*simp add: ennreal-spmf-bind ennreal-spmf-lub-spmf[OF chain'] Y nn-integral-measure-spmf' SUP-mult-left-ennreal ennreal-mult image-comp*)
also have $\dots = (\text{SUP } p \in Y. \int^+ y. \text{ennreal (spmf } x \text{ } y * \text{spmf (g } y \text{ } p) } i)) \text{ } \partial \text{count-space (set-spmf } x)$
unfolding *nn-integral-measure-spmf'* **using** *Y chain''*
by(*rule nn-integral-monotone-convergence-SUP-countable*) *simp*
also have $\dots = (\text{SUP } p \in Y. \text{ennreal (spmf (bind-spmf } x \text{ (}\lambda y. \text{g } y \text{ } p)) } i))$
by(*simp add: ennreal-spmf-bind nn-integral-measure-spmf' ennreal-mult*)
also have $\dots = \text{ennreal (spmf } ?\text{rhs } i)$ **using** *chain'''*
by(*auto simp: ennreal-spmf-lub-spmf Y image-comp*)
finally show $\text{spmf } ?\text{lhs } i = \text{spmf } ?\text{rhs } i$ **by** *simp*

qed
qed simp

lemma *mcont-bind-spmf* [*cont-intro*]:
assumes *f*: *mcont luba orda lub-spmf (ord-spmf (=)) f*
and *g*: $\bigwedge y. \text{mcont luba orda lub-spmf (ord-spmf (=)) (g y)}$
shows *mcont luba orda lub-spmf (ord-spmf (=))* ($\lambda x. \text{bind-spmf (f x) (\lambda y. g y x)}$)
proof(*rule spmf.mcont2mcont'[OF - - f]*)
fix *z*
show *mcont lub-spmf (ord-spmf (=)) lub-spmf (ord-spmf (=))* ($\lambda x. \text{bind-spmf x (\lambda y. g y z)}$)
by(*rule mcont-bind-spmf1*)
next
fix *x*
let *?f* = $\lambda z. \text{bind-spmf x (\lambda y. g y z)}$
have *monotone orda (ord-spmf (=)) ?f* **using** *mcont-mono[OF g]* **by**(*rule monotone-bind-spmf2*)
moreover have *cont luba orda lub-spmf (ord-spmf (=)) ?f*
proof(*rule contI*)
fix *Y*
assume *chain: Complete-Partial-Order.chain orda Y* **and** *Y: Y ≠ {}*
have *bind-spmf x (\lambda y. g y (luba Y)) = bind-spmf x (\lambda y. lub-spmf (g y ' Y))*
by(*rule bind-spmf-cong*)(*simp-all add: mcont-contD[OF g chain Y]*)
also have $\dots = \text{lub-spmf ((\lambda p. x \gg (\lambda y. g y p)) ' Y)}$ **using** *chain*
by(*rule bind-lub-spmf2*)(*rule mcont-mono[OF g]*)
finally show *bind-spmf x (\lambda y. g y (luba Y)) = \dots*
qed
ultimately show *mcont luba orda lub-spmf (ord-spmf (=)) ?f* **by**(*rule mcontI*)
qed

lemma *bind-pmf-mono* [*partial-function-mono*]:
 $(\bigwedge y. \text{mono-spmf } (\lambda f. C y f)) \implies \text{mono-spmf } (\lambda f. \text{bind-pmf p } (\lambda x. C x f))$
using *bind-spmf-mono[of \lambda-. spmf-of-pmf p C]* **by** *simp*

lemma *map-spmf-mono* [*partial-function-mono*]: *mono-spmf B* $\implies \text{mono-spmf } (\lambda g. \text{map-spmf f (B g)})$
unfolding *map-spmf-conv-bind-spmf* **by**(*rule bind-spmf-mono*) *simp-all*

lemma *mcont-map-spmf* [*cont-intro*]:
mcont luba orda lub-spmf (ord-spmf (=)) g
 $\implies \text{mcont luba orda lub-spmf (ord-spmf (=)) } (\lambda x. \text{map-spmf f (g x)})$
unfolding *map-spmf-conv-bind-spmf* **by**(*rule mcont-bind-spmf*) *simp-all*

lemma *monotone-set-spmf*: *monotone (ord-spmf (=))* (\subseteq) *set-spmf*
by(*rule monotoneI*)(*rule ord-spmf-eqD-set-spmf*)

lemma *cont-set-spmf*: *cont lub-spmf (ord-spmf (=)) Union* (\subseteq) *set-spmf*
by(*rule contI*)(*subst set-lub-spmf; simp*)

lemma *mcont2mcont-set-spmf* [*THEN mcont2mcont, cont-intro*]:
shows *mcont-set-spmf*: *mcont lub-spmf (ord-spmf (=)) Union (\subseteq) set-spmf*
by(*rule mcontI monotone-set-spmf cont-set-spmf*)+

lemma *monotone-spmf*: *monotone (ord-spmf (=)) (\leq) ($\lambda p. \text{spm}f\ p\ x$)*
by(*rule monotoneI*)(*simp add: ord-spmf-eq-leD*)

lemma *cont-spmf*: *cont lub-spmf (ord-spmf (=)) Sup (\leq) ($\lambda p. \text{spm}f\ p\ x$)*
by(*rule contI*)(*simp add: spmf-lub-spmf*)

lemma *mcont-spmf*: *mcont lub-spmf (ord-spmf (=)) Sup (\leq) ($\lambda p. \text{spm}f\ p\ x$)*
by(*metis mcontI monotone-spmf cont-spmf*)

lemma *cont-ennreal-spmf*: *cont lub-spmf (ord-spmf (=)) Sup (\leq) ($\lambda p. \text{ennreal}(\text{spm}f\ p\ x)$)*
by(*rule contI*)(*simp add: ennreal-spmf-lub-spmf*)

lemma *mcont2mcont-ennreal-spmf* [*THEN mcont2mcont, cont-intro*]:
shows *mcont-ennreal-spmf*: *mcont lub-spmf (ord-spmf (=)) Sup (\leq) ($\lambda p. \text{ennreal}(\text{spm}f\ p\ x)$)*
by(*metis mcontI mono2mono-ennreal monotone-spmf cont-ennreal-spmf*)

lemma *nn-integral-map-spmf* [*simp*]: *nn-integral (measure-spmf (map-spmf f p))*
g = nn-integral (measure-spmf p) (g \circ f)
by(*force simp: measure-spmf-def nn-integral-distr nn-integral-restrict-space intro: nn-integral-cong split: split-indicator*)

25.11.1 Admissibility of *rel-spmf*

lemma *rel-spmf-measureD*:
assumes *rel-spmf R p q*
shows *measure (measure-spmf p) A \leq measure (measure-spmf q) {y. $\exists x \in A. R\ x\ y$ }* (**is** ?lhs \leq ?rhs)
proof –
have ?lhs = *measure (measure-pmf p) (Some ‘A)* **by**(*simp add: measure-measure-spmf-conv-measure-pmf*)
also have ... \leq *measure (measure-pmf q) {y. $\exists x \in \text{Some ‘A}. \text{rel-option } R\ x\ y$ }*
using *assms* **by**(*rule rel-pmf-measureD*)
also have ... = ?rhs **unfolding** *measure-measure-spmf-conv-measure-pmf*
by(*rule arg-cong2[where f=measure]*)(*auto simp: option-rel-Some1*)
finally show ?thesis .
qed

locale *rel-spmf-characterisation* =
assumes *rel-pmf-measureI*:
 $\bigwedge (R :: 'a\ \text{option} \Rightarrow 'b\ \text{option} \Rightarrow \text{bool})\ p\ q.$
 $(\bigwedge A. \text{measure}(\text{measure-pmf } p)\ A \leq \text{measure}(\text{measure-pmf } q)\ \{y. \exists x \in A. R\ x\ y\})$
 $\implies \text{rel-pmf } R\ p\ q$

— This assumption is shown to hold in general in the AFP entry *MFMC-Countable*.
begin

context fixes $R :: 'a \Rightarrow 'b \Rightarrow \text{bool}$ **begin**

lemma *rel-spmf-measureI*:

assumes *eq1*: $\bigwedge A. \text{measure} (\text{measure-spmf } p) A \leq \text{measure} (\text{measure-spmf } q) \{y. \exists x \in A. R \ x \ y\}$
assumes *eq2*: $\text{weight-spmf } q \leq \text{weight-spmf } p$
shows *rel-spmf* $R \ p \ q$
proof(*rule rel-pmf-measureI*)
fix $A :: 'a \text{ option set}$
define A' **where** $A' = \text{the } ' (A \cap \text{range } \text{Some})$
define A'' **where** $A'' = A \cap \{None\}$
have $A = \text{Some } ' A' \cup A''$ **Some** $' A' \cap A'' = \{\}$
unfolding A' -def A'' -def **by**(*auto simp: image-iff*)
have $\text{measure} (\text{measure-pmf } p) A = \text{measure} (\text{measure-pmf } p) (\text{Some } ' A') + \text{measure} (\text{measure-pmf } p) A''$
by(*simp add: A measure-pmf.finite-measure-Union*)
also have $\text{measure} (\text{measure-pmf } p) (\text{Some } ' A') = \text{measure} (\text{measure-spmf } p) A'$
by(*simp add: measure-measure-spmf-conv-measure-pmf*)
also have $\dots \leq \text{measure} (\text{measure-spmf } q) \{y. \exists x \in A'. R \ x \ y\}$ **by**(*rule eq1*)
also (*ord-eq-le-trans[OF - add-right-mono]*)
have $\dots = \text{measure} (\text{measure-pmf } q) \{y. \exists x \in A'. \text{rel-option } R (\text{Some } x) \ y\}$
unfolding *measure-measure-spmf-conv-measure-pmf*
by(*rule arg-cong2[where f=measure](auto simp: A'-def option-rel-Some1)*)
also
{ have $\text{weight-spmf } p \leq \text{measure} (\text{measure-spmf } q) \{y. \exists x. R \ x \ y\}$
using *eq1*[*of UNIV*] **unfolding** *weight-spmf-def* **by** *simp*
also have $\dots \leq \text{weight-spmf } q$ **unfolding** *weight-spmf-def*
by(*rule measure-spmf.finite-measure-mono*) *simp-all*
finally have $\text{weight-spmf } p = \text{weight-spmf } q$ **using** *eq2* **by** *simp* }
then have $\text{measure} (\text{measure-pmf } p) A'' = \text{measure} (\text{measure-pmf } q) (\text{if } None \in A \text{ then } \{None\} \text{ else } \{\})$
unfolding A'' -def **by**(*simp add: pmf-None-eq-weight-spmf measure-pmf-single*)
also have $\text{measure} (\text{measure-pmf } q) \{y. \exists x \in A'. \text{rel-option } R (\text{Some } x) \ y\} + \dots$
 $= \text{measure} (\text{measure-pmf } q) \{y. \exists x \in A. \text{rel-option } R \ x \ y\}$
by(*subst measure-pmf.finite-measure-Union[symmetric]*)
(auto 4 3 intro!: arg-cong2[where f=measure] simp add: option-rel-Some1 option-rel-Some2 A'-def intro: rev-beI elim: option.rel-cases)
finally show $\text{measure} (\text{measure-pmf } p) A \leq \dots$.
qed

lemma *admissible-rel-spmf*:

ccpo.admissible (*prod-lub lub-spmf lub-spmf*) (*rel-prod* (*ord-spmf* (*=*)) (*ord-spmf* (*=*))) (*case-prod* (*rel-spmf* R))
(is cppo.admissible ?lub ?ord ?P)
proof(*rule cppo.admissibleI*)
fix Y

```

assume chain: Complete-Partial-Order.chain ?ord Y
and Y: Y ≠ {}
and R: ∀ (p, q) ∈ Y. rel-spmf R p q
from R have R: ∧ p q. (p, q) ∈ Y ⇒ rel-spmf R p q by auto
have chain1: Complete-Partial-Order.chain (ord-spmf (=)) (fst ‘ Y)
and chain2: Complete-Partial-Order.chain (ord-spmf (=)) (snd ‘ Y)
using chain by(rule chain-imageI; clarsimp)+
from Y have Y1: fst ‘ Y ≠ {} and Y2: snd ‘ Y ≠ {} by auto

have rel-spmf R (lub-spmf (fst ‘ Y)) (lub-spmf (snd ‘ Y))
proof(rule rel-spmf-measureI)
  show weight-spmf (lub-spmf (snd ‘ Y)) ≤ weight-spmf (lub-spmf (fst ‘ Y))
  by(auto simp: weight-lub-spmf chain1 chain2 Y rel-spmf-weightD[OF R, sym-
metric] intro!: cSUP-least intro: cSUP-upper2[OF bdd-aboveI2[OF weight-spmf-le-1]])

  fix A
  have measure (measure-spmf (lub-spmf (fst ‘ Y))) A = (SUP y∈fst ‘ Y. measure
(measure-spmf y) A)
  using chain1 Y1 by(rule measure-lub-spmf)
  also have ... ≤ (SUP y∈snd ‘ Y. measure (measure-spmf y) {y. ∃ x∈A. R x
y}) using Y1
  by(rule cSUP-least)(auto intro!: cSUP-upper2[OF bdd-aboveI2[OF measure-spmf.subprob-measure-le-1]]
rel-spmf-measureD R)
  also have ... = measure (measure-spmf (lub-spmf (snd ‘ Y))) {y. ∃ x∈A. R x
y}
  using chain2 Y2 by(rule measure-lub-spmf[symmetric])
  finally show measure (measure-spmf (lub-spmf (fst ‘ Y))) A ≤ ... .
qed
then show ?P (?lub Y) by(simp add: prod-lub-def)
qed

lemma admissible-rel-spmf-mcont [cont-intro]:
  [| mcont lub ord lub-spmf (ord-spmf (=)) f; mcont lub ord lub-spmf (ord-spmf
(=)) g |]
  ⇒ ccpo.admissible lub ord (λx. rel-spmf R (f x) (g x))
  by(rule admissible-subst[OF admissible-rel-spmf, where f=λx. (f x, g x), sim-
plified])(rule mcont-Pair)

context includes lifting-syntax
begin

lemma fixp-spmf-parametric':
  assumes f: ∧ x. monotone (ord-spmf (=)) (ord-spmf (=)) F
  and g: ∧ x. monotone (ord-spmf (=)) (ord-spmf (=)) G
  and param: (rel-spmf R ==> rel-spmf R) F G
  shows (rel-spmf R) (ccpo.fixp lub-spmf (ord-spmf (=)) F) (ccpo.fixp lub-spmf
(ord-spmf (=)) G)
  by(rule parallel-fixp-induct[OF ccpo-spmf ccpo-spmf - f g])(auto intro: param[THEN
rel-funD])

```

```

lemma fixp-spmf-parametric:
  assumes f:  $\bigwedge x. \text{mono-spmf } (\lambda f. F f x)$ 
  and g:  $\bigwedge x. \text{mono-spmf } (\lambda f. G f x)$ 
  and param:  $((A \text{====> rel-spmf } R) \text{====> } A \text{====> rel-spmf } R) F G$ 
  shows  $(A \text{====> rel-spmf } R) (\text{spm.f. fixp-fun } F) (\text{spm.f. fixp-fun } G)$ 
using f g
proof(rule parallel-fixp-induct-1-1[OF partial-function-definitions-spmf partial-function-definitions-spmf
- - reflexive reflexive, where  $P=(A \text{====> rel-spmf } R)$ ])
  show ccpo.admissible (prod-lub (fun-lub lub-spmf) (fun-lub lub-spmf)) (rel-prod
(fun-ord (ord-spmf (=))) (fun-ord (ord-spmf (=))))  $(\lambda x. (A \text{====> rel-spmf } R)$ 
(fst x) (snd x))
    unfolding rel-fun-def
    by(fastforce intro: admissible-all admissible-imp admissible-rel-spmf-mcont)
  show  $(A \text{====> rel-spmf } R) (\lambda-. \text{lub-spmf } \{\}) (\lambda-. \text{lub-spmf } \{\})$ 
    by auto
  show  $(A \text{====> rel-spmf } R) (F f) (G g)$  if  $(A \text{====> rel-spmf } R) f g$  for f g
    using that by(rule rel-funD[OF param])
qed

end

end

end

```

25.12 Restrictions on spmfs

definition restrict-spmf :: 'a spmf \Rightarrow 'a set \Rightarrow 'a spmf (**infixl** \lhd 110)
where $p \lhd A = \text{map-pmf } (\lambda x. x \gg= (\lambda y. \text{if } y \in A \text{ then Some } y \text{ else None})) p$

lemma set-restrict-spmf [simp]: $\text{set-spmf } (p \lhd A) = \text{set-spmf } p \cap A$
by(fastforce simp: restrict-spmf-def set-spmf-def split: bind-splits if-split-asm)

lemma restrict-map-spmf: $\text{map-spmf } f p \lhd A = \text{map-spmf } f (p \lhd (f^{-1} A))$
by(simp add: restrict-spmf-def pmf.map-comp o-def map-option-bind bind-map-option
if-distrib cong del: if-weak-cong)

lemma restrict-restrict-spmf [simp]: $p \lhd A \lhd B = p \lhd (A \cap B)$
by(auto simp: restrict-spmf-def pmf.map-comp o-def intro!: pmf.map-cong bind-option-cong)

lemma restrict-spmf-empty [simp]: $p \lhd \{\} = \text{return-pmf None}$
by(simp add: restrict-spmf-def)

lemma restrict-spmf-UNIV [simp]: $p \lhd \text{UNIV} = p$
by(simp add: restrict-spmf-def)

lemma spmf-restrict-spmf-outside [simp]: $x \notin A \Longrightarrow \text{spm.f } (p \lhd A) x = 0$
by(simp add: spmf-eq-0-set-spmf)

lemma *emeasure-restrict-spmf* [simp]: $\text{emeasure } (\text{measure-spmf } (p \restriction A)) \ X = \text{emeasure } (\text{measure-spmf } p) \ (X \cap A)$
proof –
have $(\lambda x. x \gg (\lambda y. \text{if } y \in A \text{ then } \text{Some } y \text{ else } \text{None})) - ' \text{the } - ' X \cap$
 $(\lambda x. x \gg (\lambda y. \text{if } y \in A \text{ then } \text{Some } y \text{ else } \text{None})) - ' \text{range } \text{Some} =$
 $\text{the } - ' X \cap \text{the } - ' A \cap \text{range } \text{Some}$
by(*auto split: bind-splits if-split-asm*)
then show ?thesis
by (*simp add: restrict-spmf-def measure-spmf-def emeasure-distr emeasure-restrict-space*)
qed

lemma *measure-restrict-spmf* [simp]:
 $\text{measure } (\text{measure-spmf } (p \restriction A)) \ X = \text{measure } (\text{measure-spmf } p) \ (X \cap A)$
using *emeasure-restrict-spmf*[*of p A X*]
by(*simp only: measure-spmf.emeasure-eq-measure ennreal-inj measure-nonneg*)

lemma *spm-f-restrict-spmf*: $\text{spm-f } (p \restriction A) \ x = (\text{if } x \in A \text{ then } \text{spm-f } p \ x \text{ else } 0)$
by(*simp add: spm-f-conv-measure-spmf*)

lemma *spm-f-restrict-spmf-inside* [simp]: $x \in A \implies \text{spm-f } (p \restriction A) \ x = \text{spm-f } p \ x$
by(*simp add: spm-f-restrict-spmf*)

lemma *pmf-restrict-spmf-None*: $\text{pmf } (p \restriction A) \ \text{None} = \text{pmf } p \ \text{None} + \text{measure } (\text{measure-spmf } p) \ (- A)$
proof –
have [simp]: $\text{None} \notin \text{Some } '(- A)$ **by** *auto*
have $(\lambda x. x \gg (\lambda y. \text{if } y \in A \text{ then } \text{Some } y \text{ else } \text{None})) - ' \{\text{None}\} = \{\text{None}\} \cup$
 $(\text{Some } '(- A))$
by(*auto split: bind-splits if-split-asm*)
then show ?thesis **unfolding** *ereal.inject[symmetric]*
by(*simp add: restrict-spmf-def ennreal-pmf-map emeasure-pmf-single del: ereal.inject*)
 $(\text{simp add: pmf.rep-eq measure-pmf.finite-measure-Union[symmetric] mea-}$
 $\text{sure-measure-spmf-conv-measure-pmf measure-pmf.emeasure-eq-measure})$
qed

lemma *restrict-spmf-trivial*: $(\bigwedge x. x \in \text{set-spmf } p \implies x \in A) \implies p \restriction A = p$
by(*rule spm-f-eqI*)(*auto simp: spm-f-restrict-spmf spm-f-eq-0-set-spmf*)

lemma *restrict-spmf-trivial'*: $\text{set-spmf } p \subseteq A \implies p \restriction A = p$
by(*rule restrict-spmf-trivial*) *blast*

lemma *restrict-return-spmf*: $\text{return-spmf } x \restriction A = (\text{if } x \in A \text{ then } \text{return-spmf } x \text{ else } \text{return-pmf } \text{None})$
by(*simp add: restrict-spmf-def*)

lemma *restrict-return-spmf-inside* [simp]: $x \in A \implies \text{return-spmf } x \restriction A = \text{return-spmf } x$
by(*simp add: restrict-return-spmf*)

lemma *restrict-return-spmf-outside* [simp]: $x \notin A \implies \text{return-spmf } x \upharpoonright A = \text{return-pmf None}$

by(simp add: restrict-return-spmf)

lemma *restrict-spmf-return-pmf-None* [simp]: $\text{return-pmf None} \upharpoonright A = \text{return-pmf None}$

by(simp add: restrict-spmf-def)

lemma *restrict-bind-pmf*: $\text{bind-pmf } p \ g \upharpoonright A = p \gg (\lambda x. g \ x \upharpoonright A)$

by(simp add: restrict-spmf-def map-bind-pmf o-def)

lemma *restrict-bind-spmf*: $\text{bind-spmf } p \ g \upharpoonright A = p \gg (\lambda x. g \ x \upharpoonright A)$

by(auto simp: bind-spmf-def restrict-bind-pmf cong del: option.case-cong-weak cong: option.case-cong intro!: bind-pmf-cong split: option.split)

lemma *bind-restrict-pmf*: $\text{bind-pmf } (p \upharpoonright A) \ g = p \gg (\lambda x. \text{if } x \in \text{Some } A \text{ then } g \ x \text{ else } g \ \text{None})$

by(auto simp: restrict-spmf-def bind-map-pmf fun-eq-iff split: bind-split intro: arg-cong2[where f=bind-pmf])

lemma *bind-restrict-spmf*: $\text{bind-spmf } (p \upharpoonright A) \ g = p \gg (\lambda x. \text{if } x \in A \text{ then } g \ x \text{ else } \text{return-pmf None})$

by(auto simp: bind-spmf-def bind-restrict-pmf fun-eq-iff intro: arg-cong2[where f=bind-pmf] split: option.split)

lemma *spmf-map-restrict*: $\text{spmf } (\text{map-spmf } \text{fst } (p \upharpoonright (\text{snd} - \{y\}))) \ x = \text{spmf } p \ (x, y)$

by(subst spmf-map)(auto intro: arg-cong2[where f=measure] simp add: spmf-conv-measure-spmf)

lemma *measure-eqI-restrict-spmf*:

assumes *rel-spmf* R (*restrict-spmf* $p \ A$) (*restrict-spmf* $q \ B$)

shows $\text{measure } (\text{measure-spmf } p) \ A = \text{measure } (\text{measure-spmf } q) \ B$

proof –

from *assms* **have** $\text{weight-spmf } (\text{restrict-spmf } p \ A) = \text{weight-spmf } (\text{restrict-spmf } q \ B)$ **by**(rule *rel-spmf-weightD*)

thus ?thesis **by**(simp add: weight-spmf-def)

qed

25.13 Subprobability distributions of sets

definition *spmf-of-set* :: 'a set \Rightarrow 'a spmf

where

$\text{spmf-of-set } A = (\text{if } \text{finite } A \wedge A \neq \{\} \text{ then } \text{spmf-of-pmf } (\text{pmf-of-set } A) \text{ else } \text{return-pmf None})$

lemma *spmf-of-set*: $\text{spmf } (\text{spmf-of-set } A) \ x = \text{indicator } A \ x / \text{card } A$

by(auto simp: spmf-of-set-def)

lemma *pmf-spmf-of-set-None* [simp]: $\text{pmf } (\text{spmof-of-set } A) \text{ None} = \text{indicator } \{A.\text{infinite } A \vee A = \{\}\} A$

by(simp add: spmf-of-set-def)

lemma *set-spmf-of-set*: $\text{set-spmf } (\text{spmof-of-set } A) = (\text{if finite } A \text{ then } A \text{ else } \{\})$

by(simp add: spmf-of-set-def)

lemma *set-spmf-of-set-finite* [simp]: $\text{finite } A \implies \text{set-spmf } (\text{spmof-of-set } A) = A$

by(simp add: set-spmf-of-set)

lemma *spmof-of-set-singleton*: $\text{spmof-of-set } \{x\} = \text{return-spmf } x$

by(simp add: spmf-of-set-def pmf-of-set-singleton)

lemma *map-spmf-of-set-inj-on* [simp]:

$\text{inj-on } f A \implies \text{map-spmf } f (\text{spmof-of-set } A) = \text{spmof-of-set } (f ` A)$

by(auto simp: spmf-of-set-def map-pmf-of-set-inj dest: finite-imageD)

lemma *spmof-of-pmf-pmf-of-set* [simp]:

$\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{spmof-of-pmf } (\text{pmf-of-set } A) = \text{spmof-of-set } A$

by(simp add: spmf-of-set-def)

lemma *weight-spmf-of-set*:

$\text{weight-spmf } (\text{spmof-of-set } A) = (\text{if finite } A \wedge A \neq \{\} \text{ then } 1 \text{ else } 0)$

by(auto simp only: spmf-of-set-def weight-spmf-of-pmf weight-return-pmf-None split: if-split)

lemma *weight-spmf-of-set-finite* [simp]: $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{weight-spmf } (\text{spmof-of-set } A) = 1$

by(simp add: weight-spmf-of-set)

lemma *weight-spmf-of-set-infinite* [simp]: $\text{infinite } A \implies \text{weight-spmf } (\text{spmof-of-set } A) = 0$

by(simp add: weight-spmf-of-set)

lemma *measure-spmf-spmf-of-set*:

$\text{measure-spmf } (\text{spmof-of-set } A) = (\text{if finite } A \wedge A \neq \{\} \text{ then } \text{measure-pmf } (\text{pmf-of-set } A) \text{ else } \text{null-measure } (\text{count-space } \text{UNIV}))$

by(simp add: spmf-of-set-def del: spmf-of-pmf-pmf-of-set)

lemma *emeasure-spmf-of-set*:

$\text{emeasure } (\text{measure-spmf } (\text{spmof-of-set } S)) A = \text{card } (S \cap A) / \text{card } S$

by(auto simp: measure-spmf-spmf-of-set emeasure-pmf-of-set)

lemma *measure-spmf-of-set*:

$\text{measure } (\text{measure-spmf } (\text{spmof-of-set } S)) A = \text{card } (S \cap A) / \text{card } S$

by(auto simp: measure-spmf-spmf-of-set measure-pmf-of-set)

lemma *nn-integral-spmf-of-set*: $\text{nn-integral } (\text{measure-spmf } (\text{spmof-of-set } A)) f = \text{sum } f A / \text{card } A$

by(cases finite A)(auto simp: spmf-of-set-def nn-integral-pmf-of-set card-gt-0-iff
simp del: spmf-of-pmf-pmf-of-set)

lemma integral-spmf-of-set: $\text{integral}^L (\text{measure-spmf } (\text{spmof-of-set } A)) f = \text{sum } f$
 $A / \text{card } A$

by (metis card.infinite div-0 division-ring-divide-zero integral-null-measure inte-
gral-pmf-of-set measure-spmf-spmf-of-set of-nat-0 sum.empty)

notepad begin — pmf-of-set is not fully parametric.

define $R :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where** $R \ x \ y \longleftrightarrow (x \neq 0 \longrightarrow y = 0)$ **for** $x \ y$

define $A :: \text{nat set}$ **where** $A = \{0, 1\}$

define $B :: \text{nat set}$ **where** $B = \{0, 1, 2\}$

have rel-set R A B **unfolding** R-def[abs-def] A-def B-def rel-set-def **by** auto

have $\neg \text{rel-pmf } R \ (\text{pmf-of-set } A) \ (\text{pmf-of-set } B)$

proof

assume rel-pmf R (pmf-of-set A) (pmf-of-set B)

then obtain pq **where** pq: $\bigwedge x \ y. (x, y) \in \text{set-pmf } pq \implies R \ x \ y$

and 1: map-pmf fst pq = pmf-of-set A

and 2: map-pmf snd pq = pmf-of-set B

by cases auto

have pmf (pmf-of-set B) 1 = 1 / 3 **by**(simp add: B-def)

have pmf (pmf-of-set B) 2 = 1 / 3 **by**(simp add: B-def)

have 2 / 3 = pmf (pmf-of-set B) 1 + pmf (pmf-of-set B) 2 **by**(simp add:
B-def)

also have ... = measure (measure-pmf (pmf-of-set B)) ($\{1\} \cup \{2\}$)

by(subst measure-pmf.finite-measure-Union)(simp-all add: measure-pmf-single)

also have ... = emeasure (measure-pmf pq) (snd - ‘ $\{2, 1\}$)

unfolding 2[symmetric] measure-pmf.emmeasure-eq-measure[symmetric] **by**(simp)

also have ... = emeasure (measure-pmf pq) $\{(0, 2), (0, 1)\}$

by(rule emeasure-eq-AE)(auto simp: AE-measure-pmf-iff R-def dest!: pq)

also have ... \leq emeasure (measure-pmf pq) (fst - ‘ $\{0\}$)

by(rule emeasure-mono) auto

also have ... = emeasure (measure-pmf (pmf-of-set A)) $\{0\}$

unfolding 1[symmetric] **by** simp

also have ... = pmf (pmf-of-set A) 0

by(simp add: measure-pmf-single measure-pmf.emmeasure-eq-measure)

also have pmf (pmf-of-set A) 0 = 1 / 2 **by**(simp add: A-def)

finally show False **by**(subst (asm) ennreal-le-iff; simp)

qed

end

lemma rel-pmf-of-set-bij:

assumes f: bij-betw f A B

and A: $A \neq \{\}$ finite A

and B: $B \neq \{\}$ finite B

and R: $\bigwedge x. x \in A \implies R \ x \ (f \ x)$

shows rel-pmf R (pmf-of-set A) (pmf-of-set B)

proof(rule pmf.rel-mono-strong)

```

define AB where AB = ( $\lambda x. (x, f\ x)$ ) ‘ A
define R' where R' x y  $\longleftrightarrow (x, y) \in AB$  for x y
have  $(x, y) \in AB$  if  $(x, y) \in \text{set-pmf } (\text{pmf-of-set } AB)$  for x y
  using that by(auto simp: AB-def A)
moreover have  $\text{map-pmf fst } (\text{pmf-of-set } AB) = \text{pmf-of-set } A$ 
  by(simp add: AB-def map-pmf-of-set-inj[symmetric] inj-on-def A pmf.map-comp
o-def)
moreover
from f have [simp]:  $\text{inj-on } f\ A$  by(rule bij-betw-imp-inj-on)
from f have [simp]:  $f\ ‘\ A = B$  by(rule bij-betw-imp-surj-on)
have  $\text{map-pmf snd } (\text{pmf-of-set } AB) = \text{pmf-of-set } B$ 
  by(simp add: AB-def map-pmf-of-set-inj[symmetric] inj-on-def A pmf.map-comp
o-def)
  (simp add: map-pmf-of-set-inj A)
ultimately show  $\text{rel-pmf } (\lambda x\ y. (x, y) \in AB) (\text{pmf-of-set } A) (\text{pmf-of-set } B) ..$ 
qed(auto intro: R)

```

```

lemma rel-spmf-of-set-bij:
  assumes f: bij-betw f A B
  and R:  $\bigwedge x. x \in A \implies R\ x\ (f\ x)$ 
  shows  $\text{rel-spmf } R\ (\text{spm-f-of-set } A)\ (\text{spm-f-of-set } B)$ 
proof –
  obtain finite A  $\longleftrightarrow \text{finite } B$   $A = \{\}$   $\longleftrightarrow B = \{\}$ 
    using bij-betw-empty1 bij-betw-empty2 bij-betw-finite f by blast
  then show ?thesis
    using assms
    by (metis rel-pmf-of-set-bij rel-spmf-spmf-of-pmf return-spmf-None-parametric
spm-f-of-set-def)
qed

```

```

context includes lifting-syntax
begin

```

```

lemma rel-spmf-of-set:
  assumes bi-unique R
  shows  $(\text{rel-set } R \implies \text{rel-spmf } R)\ \text{spm-f-of-set } \text{spm-f-of-set}$ 
proof
  fix A B
  assume R: rel-set R A B
  with assms obtain f where bij-betw f A B and f:  $\bigwedge x. x \in A \implies R\ x\ (f\ x)$ 
  by(auto dest: bi-unique-rel-set-bij-betw)
  then show  $\text{rel-spmf } R\ (\text{spm-f-of-set } A)\ (\text{spm-f-of-set } B)$ 
    by(rule rel-spmf-of-set-bij)
qed

```

```

end

```

```

lemma map-mem-spmf-of-set:
  assumes finite B  $B \neq \{\}$ 

```

shows $\text{map-spmf } (\lambda x. x \in A) \text{ (spmof-of-set } B) = \text{spmof-of-pmf (bernoulli-pmf (card } (A \cap B) / \text{card } B))$
(is ?lhs = ?rhs)
proof(rule *spmof-eqI*)
fix i
have $\text{ennreal (spmof ?lhs } i) = \text{card } (B \cap (\lambda x. x \in A) - \{i\}) / (\text{card } B)$
by(subst *ennreal-spmf-map*)(simp add: *measure-spmf-spmof-of-set assms emeasure-pmf-of-set*)
also have $\dots = (\text{if } i \text{ then card } (B \cap A) / \text{card } B \text{ else card } (B - A) / \text{card } B)$
by(auto intro: *arg-cong*[**where** $f = \text{card}$])
also have $\dots = (\text{if } i \text{ then card } (B \cap A) / \text{card } B \text{ else (card } B - \text{card } (B \cap A)) / \text{card } B)$
by(auto simp: *card-Diff-subset-Int assms*)
also have $\dots = \text{ennreal (spmof ?rhs } i)$
by(simp add: *assms card-gt-0-iff field-simps card-mono Int-commute of-nat-diff*)
finally show $\text{spmof ?lhs } i = \text{spmof ?rhs } i$ **by** simp
qed

abbreviation $\text{coin-spmf} :: \text{bool} \text{ spmf}$
where $\text{coin-spmf} \equiv \text{spmof-of-set UNIV}$

lemma *map-eq-const-coin-spmf*: $\text{map-spmf } ((=) \ c) \ \text{coin-spmf} = \text{coin-spmf}$
proof –
have $\text{inj } ((\longleftrightarrow) \ c) \ \text{range } ((\longleftrightarrow) \ c) = \text{UNIV}$ **by**(auto intro: *inj-onI*)
then show *?thesis* **by** simp
qed

lemma *bind-coin-spmf-eq-const*: $\text{coin-spmf} \gg (\lambda x :: \text{bool}. \text{return-spmf } (b = x)) = \text{coin-spmf}$
using *map-eq-const-coin-spmf* **unfolding** *map-spmf-conv-bind-spmf* **by** simp

lemma *bind-coin-spmf-eq-const'*: $\text{coin-spmf} \gg (\lambda x :: \text{bool}. \text{return-spmf } (x = b)) = \text{coin-spmf}$
by(rewrite in $=$ \sqsubset *bind-coin-spmf-eq-const*[*symmetric*, of b])(auto intro: *bind-spmf-cong*)

25.14 Losslessness

definition $\text{lossless-spmf} :: 'a \text{ spmf} \Rightarrow \text{bool}$
where $\text{lossless-spmf } p \longleftrightarrow \text{weight-spmf } p = 1$

lemma *lossless-iff-pmf-None*: $\text{lossless-spmf } p \longleftrightarrow \text{pmf } p \ \text{None} = 0$
by(simp add: *lossless-spmf-def pmf-None-eq-weight-spmf*)

lemma *lossless-return-spmf [iff]*: $\text{lossless-spmf } (\text{return-spmf } x)$
by(simp add: *lossless-iff-pmf-None*)

lemma *lossless-return-pmf-None [iff]*: $\neg \text{lossless-spmf } (\text{return-pmf } \text{None})$
by(simp add: *lossless-iff-pmf-None*)

lemma *lossless-map-spmf* [simp]: $\text{lossless-spmf } (\text{map-spmf } f \ p) \longleftrightarrow \text{lossless-spmf } p$

by(*auto simp: lossless-iff-pmf-None pmf-eq-0-set-pmf*)

lemma *lossless-bind-spmf* [simp]:

$\text{lossless-spmf } (p \gg f) \longleftrightarrow \text{lossless-spmf } p \wedge (\forall x \in \text{set-spmf } p. \text{lossless-spmf } (f \ x))$

by(*simp add: lossless-iff-pmf-None pmf-bind-spmf-None add-nonneg-eq-0-iff integral-nonneg-AE integral-nonneg-eq-0-iff-AE measure-spmf.integrable-const-bound[where B=1] pmf-le-1*)

lemma *lossless-weight-spmfD*: $\text{lossless-spmf } p \implies \text{weight-spmf } p = 1$

by(*simp add: lossless-spmf-def*)

lemma *lossless-iff-set-pmf-None*:

$\text{lossless-spmf } p \longleftrightarrow \text{None} \notin \text{set-pmf } p$

by (*simp add: lossless-iff-pmf-None pmf-eq-0-set-pmf*)

lemma *lossless-spmf-of-set* [simp]: $\text{lossless-spmf } (\text{spm-f-of-set } A) \longleftrightarrow \text{finite } A \wedge A \neq \{\}$

by(*auto simp: lossless-spmf-def weight-spmf-of-set*)

lemma *lossless-spmf-spmf-of-spmf* [simp]: $\text{lossless-spmf } (\text{spm-f-of-pmf } p)$

by(*simp add: lossless-spmf-def*)

lemma *lossless-spmf-bind-pmf* [simp]:

$\text{lossless-spmf } (\text{bind-pmf } p \ f) \longleftrightarrow (\forall x \in \text{set-pmf } p. \text{lossless-spmf } (f \ x))$

by(*simp add: lossless-iff-pmf-None pmf-bind integral-nonneg-AE integral-nonneg-eq-0-iff-AE measure-pmf.integrable-const-bound[where B=1] AE-measure-pmf-iff pmf-le-1*)

lemma *lossless-spmf-conv-spmf-of-pmf*: $\text{lossless-spmf } p \longleftrightarrow (\exists p'. p = \text{spm-f-of-pmf } p')$

proof

assume *lossless-spmf* *p*

hence *: $\bigwedge y. y \in \text{set-pmf } p \implies \exists x. y = \text{Some } x$

by(*case-tac y*)(*simp-all add: lossless-iff-set-pmf-None*)

let ?*p* = *map-pmf the p*

have *p* = *spm-f-of-pmf* ?*p*

proof(*rule spm-f-eqI*)

fix *i*

have *ennreal* (*pmf* (*map-pmf the p*) *i*) = $\int^+ x. \text{indicator } (\text{the } - ' \{i\}) \ x \ \partial p$

by(*simp add: ennreal-pmf-map*)

also have ... = $\int^+ x. \text{indicator } \{i\} \ x \ \partial \text{measure-spmf } p$ **unfolding** *measure-spmf-def*

by(*subst nn-integral-distr*)(*auto simp: nn-integral-restrict-space AE-measure-pmf-iff simp del: nn-integral-indicator intro!: nn-integral-cong-AE split: split-indicator dest!: **)

also have ... = *spm-f* *p i* **by**(*simp add: emeasure-spmf-single*)

finally show $\text{spmf } p \ i = \text{spmf } (\text{spmf-of-pmf } ?p) \ i$ **by** *simp*
qed
thus $\exists p'. p = \text{spmf-of-pmf } p' ..$
qed *auto*

lemma *spmf-False-conv-True: lossless-spmf p \implies spmf p False = 1 - spmf p True*
by(*clarsimp simp add: lossless-spmf-conv-spmf-of-pmf pmf-False-conv-True*)

lemma *spmf-True-conv-False: lossless-spmf p \implies spmf p True = 1 - spmf p False*
by(*simp add: spmf-False-conv-True*)

lemma *bind-eq-return-spmf:*
 $\text{bind-spmf } p \ f = \text{return-spmf } x \iff (\forall y \in \text{set-spmf } p. f \ y = \text{return-spmf } x) \wedge$
 $\text{lossless-spmf } p$
apply (*simp add: bind-spmf-def bind-eq-return-pmf split: option.split*)
by (*metis in-set-spmf lossless-iff-set-pmf-None not-None-eq*)

lemma *rel-spmf-return-spmf2:*
 $\text{rel-spmf } R \ p \ (\text{return-spmf } x) \iff \text{lossless-spmf } p \wedge (\forall a \in \text{set-spmf } p. R \ a \ x)$
apply (*simp add: lossless-iff-set-pmf-None rel-pmf-return-pmf2 option-rel-Some2 in-set-spmf*)
by (*metis in-set-spmf not-None-eq option.sel*)

lemma *rel-spmf-return-spmf1:*
 $\text{rel-spmf } R \ (\text{return-spmf } x) \ p \iff \text{lossless-spmf } p \wedge (\forall a \in \text{set-spmf } p. R \ x \ a)$
using *rel-spmf-return-spmf2 [of R^{-1-1}]* **by**(*simp add: spmf-rel-conversep*)

lemma *rel-spmf-bindI1:*
assumes $f: \bigwedge x. x \in \text{set-spmf } p \implies \text{rel-spmf } R \ (f \ x) \ q$
and $p: \text{lossless-spmf } p$
shows $\text{rel-spmf } R \ (\text{bind-spmf } p \ f) \ q$
proof –
fix $x :: 'a$
have $\text{rel-spmf } R \ (\text{bind-spmf } p \ f) \ (\text{bind-spmf } (\text{return-spmf } x) \ (\lambda -. q))$
by(*rule rel-spmf-bindI [where $R = \lambda x -. x \in \text{set-spmf } p$]*)(*simp-all add: rel-spmf-return-spmf2*
 $p \ f$)
then show *?thesis* **by** *simp*
qed

lemma *rel-spmf-bindI2:*
 $\llbracket \bigwedge x. x \in \text{set-spmf } q \implies \text{rel-spmf } R \ p \ (f \ x); \text{lossless-spmf } q \rrbracket$
 $\implies \text{rel-spmf } R \ p \ (\text{bind-spmf } q \ f)$
using *rel-spmf-bindI1 [of $q \ \text{conversep } R \ f \ p$]* **by**(*simp add: spmf-rel-conversep*)

25.15 Scaling

definition *scale-spmf* :: $\text{real} \Rightarrow 'a \ \text{spmf} \Rightarrow 'a \ \text{spmf}$

where

$\text{scale-spmf } r \ p = \text{embed-spmf } (\lambda x. \min (\text{inverse } (\text{weight-spmf } p)) (\max \ 0 \ r) \ *)$

$\text{spmf } p \ x)$

lemma *scale-spmf-le-1*:

$(\int^+ x. \min (\text{inverse } (\text{weight-spmf } p)) (\max 0 \ r) * \text{spmf } p \ x \ \partial \text{count-space } \text{UNIV})$
 ≤ 1 (**is** $?lhs \leq -$)

proof –

have $?lhs = \min (\text{inverse } (\text{weight-spmf } p)) (\max 0 \ r) * \int^+ x. \text{spmf } p \ x \ \partial \text{count-space } \text{UNIV}$

by(*subst nn-integral-cmult[symmetric]*)(*simp-all add: weight-spmf-nonneg max-def min-def ennreal-mult*)

also have $\dots \leq 1$ **unfolding** *weight-spmf-eq-nn-integral-spmf[symmetric]*

by(*simp add: min-def max-def weight-spmf-nonneg order.strict-iff-order field-simps ennreal-mult[symmetric]*)

finally show $?thesis$.

qed

lemma *spmf-scale-spmf*: $\text{spmf } (\text{scale-spmf } r \ p) \ x = \max 0 \ (\min (\text{inverse } (\text{weight-spmf } p)) \ r) * \text{spmf } p \ x$ (**is** $?lhs = ?rhs$)

unfolding *scale-spmf-def*

apply(*subst spmf-embed-spmf[OF scale-spmf-le-1]*)

apply(*simp add: max-def min-def measure-le-0-iff field-simps weight-spmf-nonneg not-le order.strict-iff-order*)

apply(*metis antisym-conv order-trans weight-spmf-nonneg zero-le-mult-iff zero-le-one*)

done

lemma *real-inverse-le-1-iff*: **fixes** $x :: \text{real}$

shows $\llbracket 0 \leq x; x \leq 1 \rrbracket \implies 1 / x \leq 1 \longleftrightarrow x = 1 \vee x = 0$

by *auto*

lemma *spmf-scale-spmf'*: $r \leq 1 \implies \text{spmf } (\text{scale-spmf } r \ p) \ x = \max 0 \ r * \text{spmf } p \ x$

using *real-inverse-le-1-iff[OF weight-spmf-nonneg weight-spmf-le-1, of p]*

by(*auto simp: spmf-scale-spmf max-def min-def field-simps*)(*metis pmf-le-0-iff spmf-le-weight*)

lemma *scale-spmf-neg*: $r \leq 0 \implies \text{scale-spmf } r \ p = \text{return-pmf } \text{None}$

by(*rule spmf-eqI*)(*simp add: spmf-scale-spmf' max-def*)

lemma *scale-spmf-return-None* [*simp*]: $\text{scale-spmf } r \ (\text{return-pmf } \text{None}) = \text{return-pmf } \text{None}$

by(*rule spmf-eqI*)(*simp add: spmf-scale-spmf*)

lemma *scale-spmf-conv-bind-bernoulli*:

assumes $r \leq 1$

shows $\text{scale-spmf } r \ p = \text{bind-pmf } (\text{bernoulli-pmf } r) \ (\lambda b. \text{if } b \text{ then } p \text{ else } \text{return-pmf } \text{None})$ (**is** $?lhs = ?rhs$)

proof(*rule spmf-eqI*)

fix x

have $\llbracket \text{weight-spmf } p = 0 \rrbracket \implies \text{spmf } p \ x = 0$

by (metis pmf-le-0-iff spmf-le-weight)
 moreover have $\llbracket \text{weight-spmf } p \neq 0; 1 / \text{weight-spmf } p < 1 \rrbracket \implies \text{weight-spmf } p = 1$
 by (smt (verit) divide-less-eq-1 measure-spmf.subprob-measure-le-1 weight-spmf-lt-0)
 ultimately have $\text{ennreal } (\text{spmf } ?\text{lhs } x) = \text{ennreal } (\text{spmf } ?\text{rhs } x)$
 using assms
 unfolding spmf-scale-spmf ennreal-pmf-bind nn-integral-measure-pmf UNIV-bool bernoulli-pmf.rep-eq
 by (auto simp: nn-integral-count-space-finite max-def min-def field-simps real-inverse-le-1-iff[OF weight-spmf-nonneg weight-spmf-le-1] ennreal-mult[symmetric])
 thus $\text{spmf } ?\text{lhs } x = \text{spmf } ?\text{rhs } x$ by simp
 qed

lemma *nn-integral-spmf*: $(\int^+ x. \text{spmf } p \ x \ \partial \text{count-space } A) = \text{emeasure } (\text{measure-spmf } p) \ A$
proof –
 have *bij-betw* Some *A* (the $- ' A \cap \text{range } \text{Some}$)
 by (auto simp: bij-betw-def)
 then show ?thesis
 by (metis bij-betw-def emeasure-measure-spmf-conv-measure-pmf nn-integral-pmf')
 qed

lemma *measure-spmf-scale-spmf*: $\text{measure-spmf } (\text{scale-spmf } r \ p) = \text{scale-measure } (\min (\text{inverse } (\text{weight-spmf } p)) \ r) \ (\text{measure-spmf } p)$
 by (rule measure-eqI; simp add: spmf-scale-spmf ennreal-mult' flip: nn-integral-spmf nn-integral-cmult)

lemma *measure-spmf-scale-spmf'*:
 assumes $r \leq 1$
 shows $\text{measure-spmf } (\text{scale-spmf } r \ p) = \text{scale-measure } r \ (\text{measure-spmf } p)$
proof (cases *weight-spmf* $p > 0$)
 case True
 with assms show ?thesis
 by (simp add: measure-spmf-scale-spmf field-simps weight-spmf-le-1 mult-le-one)
 next
 case False
 then show ?thesis
 by (simp add: order-less-le weight-spmf-eq-0)
 qed

lemma *scale-spmf-1* [*simp*]: $\text{scale-spmf } 1 \ p = p$
 by (simp add: spmf-eqI spmf-scale-spmf')

lemma *scale-spmf-0* [*simp*]: $\text{scale-spmf } 0 \ p = \text{return-pmf } \text{None}$
 by (simp add: scale-spmf-neg)

lemma *bind-scale-spmf*:
 assumes $r: r \leq 1$
 shows $\text{bind-spmf } (\text{scale-spmf } r \ p) \ f = \text{bind-spmf } p \ (\lambda x. \text{scale-spmf } r \ (f \ x))$


```

  (is ?lhs = ?rhs)
proof(rule spmf-eqI)
  fix x
  have ennreal (spmf ?lhs x) = ennreal (spmf ?rhs x)
    using r
  by(simp add: ennreal-spmf-bind measure-spmf-scale-spmf' nn-integral-scale-measure
spmf-scale-spmf'
ennreal-mult nn-integral-cmult)
  thus spmf ?lhs x = spmf ?rhs x by simp
qed

```

```

lemma scale-bind-spmf:
  assumes r ≤ 1
  shows scale-spmf r (bind-spmf p f) = bind-spmf p (λx. scale-spmf r (f x))
  (is ?lhs = ?rhs)
proof(rule spmf-eqI)
  fix x
  have ennreal (spmf ?lhs x) = ennreal (spmf ?rhs x) using assms
    unfolding spmf-scale-spmf'[OF assms]
  by(simp add: ennreal-mult ennreal-spmf-bind spmf-scale-spmf' nn-integral-cmult
max-def min-def)
  thus spmf ?lhs x = spmf ?rhs x by simp
qed

```

```

lemma bind-spmf-const: bind-spmf p (λx. q) = scale-spmf (weight-spmf p) q (is
?lhs = ?rhs)
proof(rule spmf-eqI)
  fix x
  have ennreal (spmf ?lhs x) = ennreal (spmf ?rhs x)
    using measure-spmf.subprob-measure-le-1[of p space (measure-spmf p)]
  by(subst ennreal-spmf-bind)(simp add: spmf-scale-spmf' weight-spmf-le-1 en-
nreal-mult mult.commute max-def min-def measure-spmf.emmeasure-eq-measure)
  thus spmf ?lhs x = spmf ?rhs x by simp
qed

```

```

lemma map-scale-spmf: map-spmf f (scale-spmf r p) = scale-spmf r (map-spmf f
p) (is ?lhs = ?rhs)
proof(rule spmf-eqI)
  fix i
  show spmf ?lhs i = spmf ?rhs i unfolding spmf-scale-spmf
    by(subst (1 2) spmf-map)(auto simp: measure-spmf-scale-spmf max-def min-def
ennreal-lt-0)
qed

```

```

lemma set-scale-spmf: set-spmf (scale-spmf r p) = (if r > 0 then set-spmf p else
{})
  apply(auto simp: in-set-spmf-iff-spmf spmf-scale-spmf)
  apply(simp add: min-def weight-spmf-eq-0 split: if-split-asm)
  done

```

lemma *set-scale-spmf'* [simp]: $0 < r \implies \text{set-spmf } (\text{scale-spmf } r \ p) = \text{set-spmf } p$
by(simp add: set-scale-spmf)

lemma *rel-spmf-scaleI*:
assumes $r > 0 \implies \text{rel-spmf } A \ p \ q$
shows $\text{rel-spmf } A \ (\text{scale-spmf } r \ p) \ (\text{scale-spmf } r \ q)$
proof(cases $r > 0$)
case True
from assms[OF True] **show** ?thesis
by(rule rel-spmfE)(auto simp: map-scale-spmf[symmetric] spmf-rel-map True
intro: rel-spmf-reflI)
qed(simp add: not-less scale-spmf-neg)

lemma *weight-scale-spmf*: $\text{weight-spmf } (\text{scale-spmf } r \ p) = \min 1 \ (\max 0 \ r \ * \ \text{weight-spmf } p)$
proof –
have $\llbracket 1 / \text{weight-spmf } p \leq r; \text{ennreal } r \ * \ \text{ennreal } (\text{weight-spmf } p) < 1 \rrbracket \implies$
 $\text{weight-spmf } p = 0$
by (smt (verit) ennreal-less-one-iff ennreal-mult'' measure-le-0-iff mult-imp-less-div-pos)
moreover
have $\llbracket r < 1 / \text{weight-spmf } p; 1 \leq \text{ennreal } r \ * \ \text{ennreal } (\text{weight-spmf } p) \rrbracket \implies$
 $\text{weight-spmf } p = 0$
by (smt (verit, ccfv-threshold) ennreal-ge-1 ennreal-mult'' mult-imp-div-pos-le
weight-spmf-lt-0)
ultimately
have $\text{ennreal } (\text{weight-spmf } (\text{scale-spmf } r \ p)) = \min 1 \ (\max 0 \ r \ * \ \text{ennreal } (\text{weight-spmf } p))$
done
thus ?thesis
by(auto simp: min-def max-def ennreal-mult[symmetric] split: if-split-asm)
qed

lemma *weight-scale-spmf'* [simp]:
 $\llbracket 0 \leq r; r \leq 1 \rrbracket \implies \text{weight-spmf } (\text{scale-spmf } r \ p) = r \ * \ \text{weight-spmf } p$
by(simp add: weight-scale-spmf max-def min-def)(metis antisym-conv mult-left-le
order-trans weight-spmf-le-1)

lemma *pmf-scale-spmf-None*:
 $\text{pmf } (\text{scale-spmf } k \ p) \ \text{None} = 1 - \min 1 \ (\max 0 \ k \ * \ (1 - \text{pmf } p \ \text{None}))$
unfolding pmf-None-eq-weight-spmf **by**(simp add: weight-scale-spmf)

lemma *scale-scale-spmf*:
 $\text{scale-spmf } r \ (\text{scale-spmf } r' \ p) = \text{scale-spmf } (r \ * \ \max 0 \ (\min (\text{inverse } (\text{weight-spmf } p)) \ r')) \ p$

```

  (is ?lhs = ?rhs)
proof(cases weight-spmf p > 0)
  case False
  thus ?thesis
    by (simp add: weight-spmf-eq-0 zero-less-measure-iff)
next
  case True
  show ?thesis
  proof(rule spmf-eqI)
    fix i
    have *: max 0 (min (1 / weight-spmf p) r') * max 0 (min (1 / min 1
      (weight-spmf p * max 0 r')) r) =
      max 0 (min (1 / weight-spmf p) (r * max 0 (min (1 / weight-spmf p) r')))
    using True
    by (simp add: max-def) (auto simp: min-def field-simps zero-le-mult-iff)
  show spmf ?lhs i = spmf ?rhs i
    by (simp add: spmf-scale-spmf) (metis * inverse-eq-divide mult.commute
      weight-scale-spmf)
  qed
qed

lemma scale-scale-spmf' [simp]:
  assumes 0 ≤ r r ≤ 1 0 ≤ r' r' ≤ 1
  shows scale-spmf r (scale-spmf r' p) = scale-spmf (r * r') p
proof(cases weight-spmf p > 0)
  case True
  with assms have r' = 1 if 1 ≤ r' * weight-spmf p
    by (smt (verit, best) measure-spmf.subprob-measure-le-1 mult-eq-1 mult-le-one
      that)
  with assms True show ?thesis
    by (smt (verit, best) eq-divide-imp measure-le-0-iff mult.assoc mult-nonneg-nonneg
      scale-scale-spmf weight-scale-spmf')
next
  case False
  with assms show ?thesis
    by (simp add: weight-spmf-eq-0 zero-less-measure-iff)
qed

lemma scale-spmf-eq-same: scale-spmf r p = p ⟷ weight-spmf p = 0 ∨ r = 1
  ∨ r ≥ 1 ∧ weight-spmf p = 1
  (is ?lhs ⟷ ?rhs)
proof
  assume ?lhs
  hence weight-spmf (scale-spmf r p) = weight-spmf p by simp
  hence *: min 1 (max 0 r * weight-spmf p) = weight-spmf p by(simp add:
    weight-scale-spmf)
  hence **: weight-spmf p = 0 ∨ r ≥ 1 by(auto simp: min-def max-def split:
    if-split-asm)
  show ?rhs

```

```

proof(cases weight-spmf p = 0)
  case False
  with ** have  $r \geq 1$ 
    by simp
  with * False have  $r = 1 \vee \text{weight-spmf } p = 1$ 
    by(simp add: max-def min-def not-le split: if-split-asm)
  with  $\langle r \geq 1 \rangle$  show ?thesis
    by simp
qed simp
next
  show  $\text{weight-spmf } p = 0 \vee r = 1 \vee 1 \leq r \wedge \text{weight-spmf } p = 1 \implies \text{scale-spmf } r \text{ } p = p$ 
    by (smt (verit) div-by-1 inverse-eq-divide inverse-positive-iff-positive scale-scale-spmf scale-spmf-1)
qed

```

lemma map-const-spmf-of-set:

```

 $\llbracket \text{finite } A; A \neq \{\} \rrbracket \implies \text{map-spmf } (\lambda \cdot. c) (\text{spm-f-of-set } A) = \text{return-spmf } c$ 
by(simp add: map-spmf-conv-bind-spmf bind-spmf-const)

```

25.16 Conditional spmfs

lemma set-pmf-Int-Some: $\text{set-pmf } p \cap \text{Some } 'A = \{\} \longleftrightarrow \text{set-spmf } p \cap A = \{\}$
by(auto simp: in-set-spmf)

lemma measure-spmf-zero-iff: $\text{measure } (\text{measure-spmf } p) A = 0 \longleftrightarrow \text{set-spmf } p \cap A = \{\}$
unfolding measure-measure-spmf-conv-measure-pmf **by**(simp add: measure-pmf-zero-iff set-pmf-Int-Some)

definition cond-spmf :: $'a \text{ spmf} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ spmf}$

where $\text{cond-spmf } p A = (\text{if } \text{set-spmf } p \cap A = \{\} \text{ then return-pmf None else cond-pmf } p (\text{Some } 'A))$

lemma set-cond-spmf [simp]: $\text{set-spmf } (\text{cond-spmf } p A) = \text{set-spmf } p \cap A$

by(auto \llcorner \llcorner simp add: cond-spmf-def in-set-spmf iff: set-cond-pmf[THEN set-eq-iff[THEN iffD1], THEN spec, rotated])

lemma cond-map-spmf [simp]: $\text{cond-spmf } (\text{map-spmf } f) p A = \text{map-spmf } f (\text{cond-spmf } p (f - 'A))$

proof –

have $\text{map-option } f - ' \text{Some } 'A = \text{Some } 'f - 'A$ **by** auto
moreover have $\text{set-pmf } p \cap \text{map-option } f - ' \text{Some } 'A \neq \{\}$ **if** $\text{Some } x \in \text{set-pmf } p$
 $f x \in A$ **for** x

using that **by** auto

ultimately show ?thesis **by**(auto simp: cond-spmf-def in-set-spmf cond-map-pmf)

qed

lemma spmf-cond-spmf [simp]:

$\text{spmf } (\text{cond-spmf } p \ A) \ x = (\text{if } x \in A \text{ then } \text{spmf } p \ x / \text{measure } (\text{measure-spmf } p) \ A \text{ else } 0)$

by(*auto simp: cond-spmf-def pmf-cond set-pmf-Int-Some[symmetric] measure-measure-spmf-conv-measure-pmf measure-pmf-zero-iff*)

lemma *bind-eq-return-pmf-None:*

$\text{bind-spmf } p \ f = \text{return-pmf } \text{None} \longleftrightarrow (\forall x \in \text{set-spmf } p. f \ x = \text{return-pmf } \text{None})$

by(*auto simp: bind-spmf-def bind-eq-return-pmf in-set-spmf split: option.splits*)

lemma *return-pmf-None-eq-bind:*

$\text{return-pmf } \text{None} = \text{bind-spmf } p \ f \longleftrightarrow (\forall x \in \text{set-spmf } p. f \ x = \text{return-pmf } \text{None})$

using *bind-eq-return-pmf-None[of p f]* **by** *auto*

25.17 Product spmf

definition *pair-spmf* :: $'a \text{ spmf} \Rightarrow 'b \text{ spmf} \Rightarrow ('a \times 'b) \text{ spmf}$

where $\text{pair-spmf } p \ q = \text{bind-pmf } (\text{pair-pmf } p \ q) \ (\lambda xy. \text{case } xy \text{ of } (\text{Some } x, \text{Some } y) \Rightarrow \text{return-spmf } (x, y) \mid - \Rightarrow \text{return-pmf } \text{None})$

lemma *map-fst-pair-spmf [simp]:* $\text{map-spmf } \text{fst} \ (\text{pair-spmf } p \ q) = \text{scale-spmf } (\text{weight-spmf } q) \ p$

unfolding *bind-spmf-const[symmetric]*

apply(*simp add: pair-spmf-def map-bind-pmf pair-pmf-def bind-assoc-pmf option.case-distrib*)

apply(*subst bind-commute-pmf*)

apply(*force intro!: bind-pmf-cong[OF refl] simp add: bind-return-pmf bind-spmf-def bind-return-pmf' case-option-collapse*

option.case-distrib[where h=map-spmf -] option.case-distrib[symmetric] case-option-id split: option.split cong: option.case-cong)

done

lemma *map-snd-pair-spmf [simp]:* $\text{map-spmf } \text{snd} \ (\text{pair-spmf } p \ q) = \text{scale-spmf } (\text{weight-spmf } p) \ q$

unfolding *bind-spmf-const[symmetric]*

apply(*simp add: pair-spmf-def map-bind-pmf pair-pmf-def bind-assoc-pmf option.case-distrib*

cong del: option.case-cong-weak)

apply(*auto intro!: bind-pmf-cong[OF refl] simp add: bind-return-pmf bind-spmf-def bind-return-pmf' case-option-collapse*

option.case-distrib[where h=map-spmf -] option.case-distrib[symmetric] case-option-id split: option.split cong del: option.case-cong-weak)

done

lemma *set-pair-spmf [simp]:* $\text{set-spmf } (\text{pair-spmf } p \ q) = \text{set-spmf } p \times \text{set-spmf } q$

by(*force simp add: pair-spmf-def set-spmf-bind-pmf bind-UNION in-set-spmf split: option.splits*)

lemma *spmf-pair [simp]:* $\text{spmf } (\text{pair-spmf } p \ q) \ (x, y) = \text{spmf } p \ x * \text{spmf } q \ y$ (*is ?lhs = ?rhs*)

proof –

have $\text{ennreal } ?\text{lhs} = \int^+ a. \int^+ b. \text{indicator } \{(x, y)\} (a, b) \partial \text{measure-spmf } q$
 $\partial \text{measure-spmf } p$
unfolding $\text{measure-spmf-def pair-spmf-def ennreal-pmf-bind nn-integral-pair-pmf'}$
by $(\text{auto simp: zero-ereal-def[symmetric] nn-integral-distr nn-integral-restrict-space}$
 $\text{nn-integral-multc[symmetric] intro!: nn-integral-cong split: option.split split-indicator})$
also have $\dots = \int^+ a. (\int^+ b. \text{indicator } \{y\} b \partial \text{measure-spmf } q) * \text{indicator } \{x\}$
 $a \partial \text{measure-spmf } p$
by $(\text{subst nn-integral-multc[symmetric]}) (\text{auto intro!: nn-integral-cong split: split-indicator})$
also have $\dots = \text{ennreal } ?\text{rhs}$ **by** $(\text{simp add: emeasure-spmf-single max-def en-$
 $\text{nreal-mult mult.commute})$
finally show $?thesis$ **by** simp
qed

lemma $\text{pair-map-spmf2: pair-spmf } p (\text{map-spmf } f q) = \text{map-spmf } (\text{apsnd } f) (\text{pair-spmf } p q)$
unfolding $\text{pair-spmf-def pair-map-pmf2 bind-map-pmf map-bind-pmf}$
by $(\text{intro bind-pmf-cong refl}) (\text{auto split: option.split})$

lemma $\text{pair-map-spmf1: pair-spmf } (\text{map-spmf } f p) q = \text{map-spmf } (\text{apfst } f) (\text{pair-spmf } p q)$
unfolding $\text{pair-spmf-def pair-map-pmf1 bind-map-pmf map-bind-pmf}$
by $(\text{intro bind-pmf-cong refl}) (\text{auto split: option.split})$

lemma $\text{pair-map-spmf: pair-spmf } (\text{map-spmf } f p) (\text{map-spmf } g q) = \text{map-spmf } (\text{map-prod } f g) (\text{pair-spmf } p q)$
unfolding $\text{pair-map-spmf2 pair-map-spmf1 spmf.map-comp}$
by $(\text{simp add: apfst-def apsnd-def o-def prod.map-comp})$

lemma $\text{pair-spmf-alt-def: pair-spmf } p q = \text{bind-spmf } p (\lambda x. \text{bind-spmf } q (\lambda y. \text{return-spmf } (x, y)))$
unfolding $\text{pair-spmf-def pair-pmf-def bind-spmf-def bind-assoc-pmf bind-return-pmf}$
by $(\text{intro bind-pmf-cong refl}) (\text{auto split: option.split})$

lemma $\text{weight-pair-spmf [simp]: weight-spmf } (\text{pair-spmf } p q) = \text{weight-spmf } p * \text{weight-spmf } q$
unfolding pair-spmf-alt-def **by** $(\text{simp add: weight-bind-spmf o-def})$

lemma pair-scale-spmf1:
 $r \leq 1 \implies \text{pair-spmf } (\text{scale-spmf } r p) q = \text{scale-spmf } r (\text{pair-spmf } p q)$
by $(\text{simp add: pair-spmf-alt-def scale-bind-spmf bind-scale-spmf})$

lemma pair-scale-spmf2:
 $r \leq 1 \implies \text{pair-spmf } p (\text{scale-spmf } r q) = \text{scale-spmf } r (\text{pair-spmf } p q)$
by $(\text{simp add: pair-spmf-alt-def scale-bind-spmf bind-scale-spmf})$

lemma $\text{pair-spmf-return-None1 [simp]: pair-spmf } (\text{return-pmf } \text{None}) p = \text{return-pmf } \text{None}$
by $(\text{rule spmf-eqI})(\text{clarsimp})$

lemma *pair-spmf-return-None2* [simp]: *pair-spmf* *p* (*return-pmf* *None*) = *return-pmf* *None*
by(*rule* *spmf-eqI*)(*clarsimp*)

lemma *pair-spmf-return-spmf1*: *pair-spmf* (*return-spmf* *x*) *q* = *map-spmf* (*Pair* *x*) *q*
by(*rule* *spmf-eqI*)(*auto* *split*: *split-indicator* *simp* *add*: *spmf-map-inj'* *inj-on-def* *intro*: *spmf-map-outside*)

lemma *pair-spmf-return-spmf2*: *pair-spmf* *p* (*return-spmf* *y*) = *map-spmf* ($\lambda x. (x, y)$) *p*
by(*rule* *spmf-eqI*)(*auto* *split*: *split-indicator* *simp* *add*: *inj-on-def* *intro*!: *spmf-map-outside* *spmf-map-inj'*[*symmetric*])

lemma *pair-spmf-return-spmf* [simp]: *pair-spmf* (*return-spmf* *x*) (*return-spmf* *y*) = *return-spmf* (*x*, *y*)
by(*simp* *add*: *pair-spmf-return-spmf1*)

lemma *rel-pair-spmf-prod*:
rel-spmf (*rel-prod* *A* *B*) (*pair-spmf* *p* *q*) (*pair-spmf* *p'* *q'*) \longleftrightarrow
rel-spmf *A* (*scale-spmf* (*weight-spmf* *q*) *p*) (*scale-spmf* (*weight-spmf* *q'*) *p'*) \wedge
rel-spmf *B* (*scale-spmf* (*weight-spmf* *p*) *q*) (*scale-spmf* (*weight-spmf* *p'*) *q'*)
(*is* *?lhs* \longleftrightarrow *?rhs* *is* - \longleftrightarrow *?A* \wedge *?B* *is* - \longleftrightarrow *rel-spmf* - *?p* *?p'* \wedge *rel-spmf* - *?q* *?q'*)

proof(*intro* *iffI* *conjI*)

assume *?rhs*

then obtain *pq* *pq'* **where** *p*: *map-spmf* *fst* *pq* = *?p* **and** *p'*: *map-spmf* *snd* *pq* = *?p'*

and *q*: *map-spmf* *fst* *pq'* = *?q* **and** *q'*: *map-spmf* *snd* *pq'* = *?q'*

and *: $\bigwedge x x'. (x, x') \in \text{set-spmf } pq \implies A x x'$

and **: $\bigwedge y y'. (y, y') \in \text{set-spmf } pq' \implies B y y'$ **by**(*auto* *elim*!: *rel-spmfE*)

let *?f* = $\lambda((x, x'), (y, y')). ((x, y), (x', y'))$

let *?r* = $1 / (\text{weight-spmf } p * \text{weight-spmf } q)$

let *?pq* = *scale-spmf* *?r* (*map-spmf* *?f* (*pair-spmf* *pq* *pq'*))

{ **fix** *p* :: '*x* *spmf* **and** *q* :: '*y* *spmf*

assume *weight-spmf* *q* $\neq 0$

and *weight-spmf* *p* $\neq 0$

and $1 / (\text{weight-spmf } p * \text{weight-spmf } q) \leq \text{weight-spmf } p * \text{weight-spmf } q$

hence $1 \leq (\text{weight-spmf } p * \text{weight-spmf } q) * (\text{weight-spmf } p * \text{weight-spmf } q)$

by(*simp* *add*: *pos-divide-le-eq* *order.strict-iff-order* *weight-spmf-nonneg*)

moreover have (*weight-spmf* *p* * *weight-spmf* *q*) * (*weight-spmf* *p* * *weight-spmf* *q*) $\leq (1 * 1) * (1 * 1)$

by(*intro* *mult-mono*)(*simp-all* *add*: *weight-spmf-nonneg* *weight-spmf-le-1*)

ultimately have (*weight-spmf* *p* * *weight-spmf* *q*) * (*weight-spmf* *p* * *weight-spmf* *q*) = 1 **by** *simp*

hence *: *weight-spmf* *p* * *weight-spmf* *q* = 1

by(*metis* *antisym-conv* *less-le* *mult-less-cancel-left1* *weight-pair-spmf* *weight-spmf-le-1*)

```

weight-spmf-nonneg)
  hence **: weight-spmf p = 1 by (metis antisym-conv mult-left-le weight-spmf-le-1
weight-spmf-nonneg)
  moreover from * ** have weight-spmf q = 1 by simp
  moreover note calculation }
note full = this

show ?lhs
proof
  have [simp]: fst ∘ ?f = map-prod fst fst by (simp add: fun-eq-iff)
  have map-spmf fst ?pq = scale-spmf ?r (pair-spmf ?p ?q)
    by (simp add: pair-map-spmf[symmetric] p q map-scale-spmf spmf.map-comp)
  also have ... = pair-spmf p q using full[of p q]
  by (simp add: pair-scale-spmf1 pair-scale-spmf2 weight-spmf-le-1 weight-spmf-nonneg)
    (auto simp: scale-scale-spmf max-def min-def field-simps weight-spmf-nonneg
weight-spmf-eq-0)
  finally show map-spmf fst ?pq = ... .

  have [simp]: snd ∘ ?f = map-prod snd snd by (simp add: fun-eq-iff)
  from ‹?rhs› have eq: weight-spmf p * weight-spmf q = weight-spmf p' *
weight-spmf q'
  by (auto dest!: rel-spmf-weightD simp add: weight-spmf-le-1 weight-spmf-nonneg)

  have map-spmf snd ?pq = scale-spmf ?r (pair-spmf ?p' ?q')
  by (simp add: pair-map-spmf[symmetric] p' q' map-scale-spmf spmf.map-comp)
  also have ... = pair-spmf p' q' using full[of p' q'] eq
  by (simp add: pair-scale-spmf1 pair-scale-spmf2 weight-spmf-le-1 weight-spmf-nonneg)
    (auto simp: scale-scale-spmf max-def min-def field-simps weight-spmf-nonneg
weight-spmf-eq-0)
  finally show map-spmf snd ?pq = ... .
qed(auto simp: set-scale-spmf split: if-split-asm dest: * ** )
next
assume ?lhs
then obtain pq where pq: map-spmf fst pq = pair-spmf p q
  and pq': map-spmf snd pq = pair-spmf p' q'
  and *:  $\bigwedge x y x' y'. ((x, y), (x', y')) \in \text{set-spmf } pq \implies A x x' \wedge B y y'$ 
  by (auto elim: rel-spmfE)

show ?A
proof
  let ?f = ( $\lambda((x, y), (x', y')). (x, x')$ )
  let ?pq = map-spmf ?f pq
  have [simp]: fst ∘ ?f = fst ∘ fst by (simp add: split-def o-def)
  show map-spmf fst ?pq = scale-spmf (weight-spmf q) p using pq
    by (simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])

  have [simp]: snd ∘ ?f = fst ∘ snd by (simp add: split-def o-def)
  show map-spmf snd ?pq = scale-spmf (weight-spmf q') p' using pq'
    by (simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])

```



```

qed(auto dest: * )

show ?B
proof
  let ?f = ( $\lambda((x, y), (x', y')). (y, y')$ )
  let ?pq = map-spmf ?f pq
  have [simp]: fst  $\circ$  ?f = snd  $\circ$  fst by(simp add: split-def o-def)
  show map-spmf fst ?pq = scale-spmf (weight-spmf p) q using pq
    by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])

  have [simp]: snd  $\circ$  ?f = snd  $\circ$  snd by(simp add: split-def o-def)
  show map-spmf snd ?pq = scale-spmf (weight-spmf p') q' using pq'
    by(simp add: spmf.map-comp)(simp add: spmf.map-comp[symmetric])
qed(auto dest: * )
qed

```

```

lemma pair-pair-spmf:
  pair-spmf (pair-spmf p q) r = map-spmf ( $\lambda(x, (y, z)). ((x, y), z)$ ) (pair-spmf p
(pair-spmf q r))
  by(simp add: pair-spmf-alt-def map-spmf-conv-bind-spmf)

```

```

lemma pair-commute-spmf:
  pair-spmf p q = map-spmf ( $\lambda(y, x). (x, y)$ ) (pair-spmf q p)
  unfolding pair-spmf-alt-def by(subst bind-commute-spmf)(simp add: map-spmf-conv-bind-spmf)

```

25.18 Assertions

```

definition assert-spmf :: bool  $\Rightarrow$  unit spmf
  where assert-spmf b = (if b then return-spmf () else return-pmf None)

```

```

lemma assert-spmf-simps [simp]:
  assert-spmf True = return-spmf ()
  assert-spmf False = return-pmf None
  by(simp-all add: assert-spmf-def)

```

```

lemma in-set-assert-spmf [simp]:  $x \in \text{set-spmf } (\text{assert-spmf } p) \longleftrightarrow p$ 
  by(cases p) simp-all

```

```

lemma set-spmf-assert-spmf-eq-empty [simp]:  $\text{set-spmf } (\text{assert-spmf } b) = \{\} \longleftrightarrow$ 
 $\neg b$ 
  by auto

```

```

lemma lossless-assert-spmf [iff]:  $\text{lossless-spmf } (\text{assert-spmf } b) \longleftrightarrow b$ 
  by(cases b) simp-all

```

25.19 Try

```

definition try-spmf :: 'a spmf  $\Rightarrow$  'a spmf  $\Rightarrow$  'a spmf
  ( $\langle \langle \text{open-block notation} = \langle \text{mixfix try-spmf} \rangle \rangle \text{TRY - ELSE -} \rangle [0, 60] 59$ )

```

where $TRY\ p\ ELSE\ q = bind_pmf\ p\ (\lambda x. case\ x\ of\ None \Rightarrow q \mid Some\ y \Rightarrow return_spmf\ y)$

lemma *try-spmf-lossless* [simp]:

assumes *lossless-spmf* p

shows $TRY\ p\ ELSE\ q = p$

proof –

have $TRY\ p\ ELSE\ q = bind_pmf\ p\ return_pmf$ **unfolding** *try-spmf-def* **using** *assms*

by(*auto simp: lossless-iff-set-pmf-None split: option.split intro: bind-pmf-cong*)

thus *?thesis* **by**(*simp add: bind-return-pmf'*)

qed

lemma *try-spmf-return-spmf1*: $TRY\ return_spmf\ x\ ELSE\ q = return_spmf\ x$

by *simp*

lemma *try-spmf-return-None* [simp]: $TRY\ return_pmf\ None\ ELSE\ q = q$

by(*simp add: try-spmf-def bind-return-pmf*)

lemma *try-spmf-return-pmf-None2* [simp]: $TRY\ p\ ELSE\ return_pmf\ None = p$

by(*simp add: try-spmf-def option.case-distrib[symmetric] bind-return-pmf' case-option-id*)

lemma *map-try-spmf*: $map_spmf\ f\ (try_spmf\ p\ q) = try_spmf\ (map_spmf\ f\ p)$
(*map-spmf* $f\ q$)

by(*simp add: try-spmf-def map-bind-pmf bind-map-pmf option.case-distrib* **where** $h = map_spmf\ f$) *o-def cong del: option.case-cong-weak*)

lemma *try-spmf-bind-pmf*: $TRY\ (bind_pmf\ p\ f)\ ELSE\ q = bind_pmf\ p\ (\lambda x. TRY\ (f\ x)\ ELSE\ q)$

by(*simp add: try-spmf-def bind-assoc-pmf*)

lemma *try-spmf-bind-spmf-lossless*:

$lossless_spmf\ p \Longrightarrow TRY\ (bind_spmf\ p\ f)\ ELSE\ q = bind_spmf\ p\ (\lambda x. TRY\ (f\ x)\ ELSE\ q)$

by (*metis (mono-tags, lifting) bind-spmf-of-pmf lossless-spmf-conv-spmf-of-pmf try-spmf-bind-pmf*)

lemma *try-spmf-bind-out*:

$lossless_spmf\ p \Longrightarrow bind_spmf\ p\ (\lambda x. TRY\ (f\ x)\ ELSE\ q) = TRY\ (bind_spmf\ p\ f)\ ELSE\ q$

by(*simp add: try-spmf-bind-spmf-lossless*)

lemma *lossless-try-spmf* [simp]:

$lossless_spmf\ (TRY\ p\ ELSE\ q) \longleftrightarrow lossless_spmf\ p \vee lossless_spmf\ q$

by(*auto simp: try-spmf-def in-set-spmf lossless-iff-set-pmf-None split: option.splits*)

context *includes lifting-syntax*

begin

lemma *try-spmf-parametric* [*transfer-rule*]:

(*rel-spmf* *A* \implies *rel-spmf* *A* \implies *rel-spmf* *A*) *try-spmf* *try-spmf*

unfolding *try-spmf-def*[*abs-def*] **by** *transfer-prover*

end

lemma *try-spmf-cong*:

$\llbracket p = p'; \neg \text{lossless-spmf } p' \implies q = q' \rrbracket \implies \text{TRY } p \text{ ELSE } q = \text{TRY } p' \text{ ELSE } q'$

unfolding *try-spmf-def*

by(*rule* *bind-pmf-cong*)(*auto split: option.split simp add: lossless-iff-set-pmf-None*)

lemma *rel-spmf-try-spmf*:

$\llbracket \text{rel-spmf } R \text{ } p \text{ } p'; \neg \text{lossless-spmf } p' \implies \text{rel-spmf } R \text{ } q \text{ } q' \rrbracket$

$\implies \text{rel-spmf } R \text{ } (\text{TRY } p \text{ ELSE } q) \text{ } (\text{TRY } p' \text{ ELSE } q')$

unfolding *try-spmf-def*

apply(*rule* *rel-pmf-bindI*[**where** *R*= $\lambda x y. \text{rel-option } R \text{ } x \text{ } y \wedge x \in \text{set-pmf } p \wedge y \in \text{set-pmf } p'$])

apply (*simp add: pmf.rel-mono-strong*)

apply(*auto split: option.split simp add: lossless-iff-set-pmf-None*)

done

lemma *spmf-try-spmf*:

spmf (*TRY* *p* *ELSE* *q*) *x* = *spmf* *p* *x* + *pmf* *p* *None* * *spmf* *q* *x*

proof –

have *ennreal* (*spmf* (*TRY* *p* *ELSE* *q*) *x*) = $\int^+ y. \text{ennreal } (\text{spmf } q \text{ } x) * \text{indicator } \{\text{None}\} y + \text{indicator } \{\text{Some } x\} y \partial \text{measure-pmf } p$

unfolding *try-spmf-def* *ennreal-pmf-bind* **by**(*rule* *nn-integral-cong*)(*simp split: option.split split-indicator*)

also have $\dots = (\int^+ y. \text{ennreal } (\text{spmf } q \text{ } x) * \text{indicator } \{\text{None}\} y \partial \text{measure-pmf } p) + \int^+ y. \text{indicator } \{\text{Some } x\} y \partial \text{measure-pmf } p$

by(*simp add: nn-integral-add*)

also have $\dots = \text{ennreal } (\text{spmf } q \text{ } x) * \text{pmf } p \text{ } \text{None} + \text{spmf } p \text{ } x$ **by**(*simp add: emeasure-pmf-single*)

finally show *?thesis* **by**(*simp flip: ennreal-plus ennreal-mult*)

qed

lemma *try-scale-spmf-same* [*simp*]: *lossless-spmf* *p* $\implies \text{TRY } \text{scale-spmf } k \text{ } p \text{ ELSE } p = p$

by(*rule* *spmf-eqI*)(*auto simp: spmf-try-spmf spmf-scale-spmf pmf-scale-spmf-None lossless-iff-pmf-None weight-spmf-conv-pmf-None min-def max-def field-simps*)

lemma *pmf-try-spmf-None* [*simp*]: *pmf* (*TRY* *p* *ELSE* *q*) *None* = *pmf* *p* *None* * *pmf* *q* *None* (**is** *?lhs* = *?rhs*)

proof –

have *?lhs* = $\int x. \text{pmf } q \text{ } \text{None} * \text{indicator } \{\text{None}\} x \partial \text{measure-pmf } p$

unfolding *try-spmf-def* *pmf-bind* **by**(*rule* *Bochner-Integration.integral-cong*)(*simp-all split: option.split*)

also have $\dots = \text{?rhs}$ **by**(*simp add: measure-pmf-single*)

finally show *?thesis* .

qed

lemma *try-bind-spmf-lossless2*:

lossless-spmf $q \implies \text{TRY} (\text{bind-spmf } p \ f) \ \text{ELSE } q = \text{TRY} (p \gg (\lambda x. \text{TRY} (f \ x) \ \text{ELSE } q)) \ \text{ELSE } q$
by(*rule* *spm-f-eqI*)(*simp* *add*: *spm-f-try-spmf* *pmf-bind-spmf-None* *spm-f-bind* *field-simps* *measure-spmf.integrable-const-bound*[**where** $B=1$] *pmf-le-1* *lossless-iff-pmf-None*)

lemma *try-bind-spmf-lossless2'*:

fixes $f :: 'a \Rightarrow 'b \text{ spmf}$ **shows**
 $\llbracket \text{NO-MATCH } (\lambda x :: 'a. \text{try-spmf } (g \ x :: 'b \text{ spmf}) \ (h \ x)) \ f; \text{lossless-spmf } q \rrbracket$
 $\implies \text{TRY} (\text{bind-spmf } p \ f) \ \text{ELSE } q = \text{TRY} (p \gg (\lambda x :: 'a. \text{TRY} (f \ x) \ \text{ELSE } q))$
 $\text{ELSE } q$
by(*rule* *try-bind-spmf-lossless2*)

lemma *try-bind-assert-spmf*:

$\text{TRY} (\text{assert-spmf } b \gg f) \ \text{ELSE } q = (\text{if } b \text{ then } \text{TRY} (f \ ()) \ \text{ELSE } q \text{ else } q)$
by *simp*

25.20 Miscellaneous

lemma *assumes* *rel-spmf* $(\lambda x \ y. \text{bad1 } x = \text{bad2 } y \wedge (\neg \text{bad2 } y \longrightarrow A \ x \longleftrightarrow B \ y)) \ p \ q$ (**is** *rel-spmf* $?A \ - \ -$)

shows *fundamental-lemma-bad*: $\text{measure} (\text{measure-spmf } p) \ \{x. \text{bad1 } x\} = \text{measure} (\text{measure-spmf } q) \ \{y. \text{bad2 } y\}$ (**is** $?bad$)

and *fundamental-lemma*: $|\text{measure} (\text{measure-spmf } p) \ \{x. A \ x\} - \text{measure} (\text{measure-spmf } q) \ \{y. B \ y\}| \leq$

$\text{measure} (\text{measure-spmf } p) \ \{x. \text{bad1 } x\}$ (**is** $?fundamental$)

proof –

have *good*: *rel-fun* $?A (=) (\lambda x. A \ x \wedge \neg \text{bad1 } x) (\lambda y. B \ y \wedge \neg \text{bad2 } y)$ **by**(*auto* *simp*: *rel-fun-def*)

from *assms* **have** *1*: $\text{measure} (\text{measure-spmf } p) \ \{x. A \ x \wedge \neg \text{bad1 } x\} = \text{measure} (\text{measure-spmf } q) \ \{y. B \ y \wedge \neg \text{bad2 } y\}$

by(*rule* *measure-spmf-parametric*[*THEN* *rel-funD*, *THEN* *rel-funD*])(*rule* *Collect-parametric*[*THEN* *rel-funD*, *OF* *good*])

have *bad*: *rel-fun* $?A (=) \text{bad1 } \text{bad2}$ **by**(*simp* *add*: *rel-fun-def*)

show *2*: $?bad$ **using** *assms*

by(*rule* *measure-spmf-parametric*[*THEN* *rel-funD*, *THEN* *rel-funD*])(*rule* *Collect-parametric*[*THEN* *rel-funD*, *OF* *bad*])

let $? \mu p = \text{measure} (\text{measure-spmf } p)$ **and** $? \mu q = \text{measure} (\text{measure-spmf } q)$

have $\{x. A \ x \wedge \text{bad1 } x\} \cup \{x. A \ x \wedge \neg \text{bad1 } x\} = \{x. A \ x\}$

and $\{y. B \ y \wedge \text{bad2 } y\} \cup \{y. B \ y \wedge \neg \text{bad2 } y\} = \{y. B \ y\}$ **by** *auto*

then have $|\ ? \mu p \ \{x. A \ x\} - ? \mu q \ \{x. B \ x\} | = |\ ? \mu p \ (\{x. A \ x \wedge \text{bad1 } x\} \cup \{x. A \ x \wedge \neg \text{bad1 } x\}) - ? \mu q \ (\{y. B \ y \wedge \text{bad2 } y\} \cup \{y. B \ y \wedge \neg \text{bad2 } y\}) |$

by *simp*

also have $\dots = |\ ? \mu p \ \{x. A \ x \wedge \text{bad1 } x\} + ? \mu p \ \{x. A \ x \wedge \neg \text{bad1 } x\} - ? \mu q \ \{y. B \ y \wedge \text{bad2 } y\} - ? \mu q \ \{y. B \ y \wedge \neg \text{bad2 } y\} |$

```

  by(subst (1 2) measure-Union)(auto)
  also have ... = |?μp {x. A x ∧ bad1 x} - ?μq {y. B y ∧ bad2 y}| using 1 by
simp
  also have ... ≤ max (?μp {x. A x ∧ bad1 x}) (?μq {y. B y ∧ bad2 y})
  by(rule abs-leI)(auto simp: max-def not-le, simp-all only: add-increasing mea-
sure-nonneg mult-2)
  also have ... ≤ max (?μp {x. bad1 x}) (?μq {y. bad2 y})
  by(rule max.mono; rule measure-spmf.finite-measure-mono; auto)
  also note 2[symmetric]
  finally show ?fundamental by simp
qed

end

```

26 Indexed products of PMFs

theory *Product-PMF*

imports *Probability-Mass-Function Independent-Family*
begin

Conflicting notation from *HOL–Analysis.Infinite-Sum*

no-notation *Infinite-Sum.abs-summable-on* (**infixr** ‹abs'-summable'-on› 46)

26.1 Preliminaries

lemma *pmf-expectation-eq-infsetsum*: *measure-pmf.expectation p f = infsetsum*
*(λx. pmf p x * f x) UNIV*
unfolding *infsetsum-def measure-pmf-eq-density* **by** (*subst integral-density*) *simp-all*

lemma *measure-pmf-prob-product*:

assumes *countable A countable B*
shows *measure-pmf.prob (pair-pmf M N) (A × B) = measure-pmf.prob M A **
measure-pmf.prob N B

proof –

have *measure-pmf.prob (pair-pmf M N) (A × B) = (∑_{a, b ∈ A × B.} pmf M*
*a * pmf N b)*

by (*auto intro!: infsetsum-cong simp add: measure-pmf-conv-infsetsum pmf-pair*)

also have ... = *measure-pmf.prob M A * measure-pmf.prob N B*

using *assms* **by** (*subst infsetsum-product*) (*auto simp add: measure-pmf-conv-infsetsum*)

finally show *?thesis*

by *simp*

qed

26.2 Definition

In analogy to Pi_M , we define an indexed product of PMFs. In the literature, this is typically called taking a vector of independent random variables. Note that the components do not have to be identically distributed.

The operation takes an explicit index set A and a function f that maps each element from A to a PMF and defines the product measure $\bigotimes_{i \in A} f(i)$, which is represented as a $(\text{'a} \Rightarrow \text{'b}) \text{ pmf}$.

Note that unlike Pi_M , this only works for *finite* index sets. It could be extended to countable sets and beyond, but the construction becomes somewhat more involved.

definition $Pi\text{-pmf} :: \text{'a set} \Rightarrow \text{'b} \Rightarrow (\text{'a} \Rightarrow \text{'b pmf}) \Rightarrow (\text{'a} \Rightarrow \text{'b}) \text{ pmf}$ **where**
 $Pi\text{-pmf } A \text{ dflt } p =$
 $\text{embed-pmf } (\lambda f. \text{ if } (\forall x. x \notin A \longrightarrow f x = \text{dflt}) \text{ then } \prod_{x \in A}. \text{ pmf } (p x) (f x)$
 $\text{ else } 0)$

A technical subtlety that needs to be addressed is this: Intuitively, the functions in the support of a product distribution have domain A . However, since HOL is a total logic, these functions must still return *some* value for inputs outside A . The product measure Pi_M simply lets these functions return *undefined* in these cases. We chose a different solution here, which is to supply a default value *dflt* that is returned in these cases.

As one possible application, one could model the result of n different independent coin tosses as $Pi\text{-pmf } \{0..<n\} \text{ False } (\lambda-. \text{ bernoulli-pmf } (1 / 2))$. This returns a function of type $\text{nat} \Rightarrow \text{bool}$ that maps every natural number below n to the result of the corresponding coin toss, and every other natural number to *False*.

lemma pmf-Pi :

assumes $A: \text{finite } A$

shows $\text{pmf } (Pi\text{-pmf } A \text{ dflt } p) f =$

$(\text{ if } (\forall x. x \notin A \longrightarrow f x = \text{dflt}) \text{ then } \prod_{x \in A}. \text{ pmf } (p x) (f x) \text{ else } 0)$

unfolding $Pi\text{-pmf-def}$

proof (*rule pmf-embed-pmf, goal-cases*)

case 2

define S **where** $S = \{f. \forall x. x \notin A \longrightarrow f x = \text{dflt}\}$

define B **where** $B = (\lambda x. \text{ set-pmf } (p x))$

have *neutral-left*: $(\prod_{x \in A}. \text{ pmf } (p x) (f x)) = 0$

if $f \in PiE A B - (\lambda f. \text{ restrict } f A) \text{ ' } S$ **for** f

proof –

have *restrict* $(\lambda x. \text{ if } x \in A \text{ then } f x \text{ else dflt}) A \in (\lambda f. \text{ restrict } f A) \text{ ' } S$

by (*intro imageI*) (*auto simp: S-def*)

also have *restrict* $(\lambda x. \text{ if } x \in A \text{ then } f x \text{ else dflt}) A = f$

using *that* **by** (*auto simp: PiE-def Pi-def extensional-def fun-eq-iff*)

finally show *?thesis* **using** *that* **by** *blast*

qed

have *neutral-right*: $(\prod_{x \in A}. \text{ pmf } (p x) (f x)) = 0$

if $f \in (\lambda f. \text{ restrict } f A) \text{ ' } S - PiE A B$ **for** f

proof –

from *that* **obtain** f' **where** $f': f = \text{ restrict } f' A \text{ } f' \in S$ **by** *auto*

moreover from *this* **and** *that* **have** *restrict* $f' A \notin PiE A B$ **by** *simp*

then obtain x where $x \in A$ $\text{pmf } (p \ x) \ (f' \ x) = 0$ by $(\text{auto simp: B-def set-pmf-eq})$

with f' and A show $?thesis$ by auto

qed

have $(\lambda f. \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x)) \text{ abs-summable-on } \text{PiE } A \ B$

by $(\text{intro abs-summable-on-prod-PiE } A) \ (\text{auto simp: B-def})$

also have $?this \longleftrightarrow (\lambda f. \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x)) \text{ abs-summable-on } (\lambda f. \text{restrict } f \ A) \ ' S$

by $(\text{intro abs-summable-on-cong-neutral neutral-left neutral-right}) \ \text{auto}$

also have $\dots \longleftrightarrow (\lambda f. \prod_{x \in A}. \text{pmf } (p \ x) \ (\text{restrict } f \ A \ x)) \text{ abs-summable-on } S$

by $(\text{rule abs-summable-on-reindex-iff [symmetric]}) \ (\text{force simp: inj-on-def fun-eq-iff S-def})$

also have $\dots \longleftrightarrow (\lambda f. \text{if } \forall x. x \notin A \longrightarrow f \ x = \text{dflt} \text{ then } \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x) \text{ else } 0)$

$\text{abs-summable-on UNIV}$

by $(\text{intro abs-summable-on-cong-neutral}) \ (\text{auto simp: S-def})$

finally have $\text{summable: } \dots$

have $1 = (\prod_{x \in A}. 1 :: \text{real})$ by simp

also have $(\prod_{x \in A}. 1) = (\prod_{x \in A}. \sum_{a y \in B \ x}. \text{pmf } (p \ x) \ y)$

unfolding B-def by $(\text{subst infsetsum-pmf-eq-1}) \ \text{auto}$

also have $(\prod_{x \in A}. \sum_{a y \in B \ x}. \text{pmf } (p \ x) \ y) = (\sum_{a f \in \text{PiE } A \ B}. \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x))$

by $(\text{intro infsetsum-prod-PiE [symmetric] } A) \ (\text{auto simp: B-def})$

also have $\dots = (\sum_{a f \in (\lambda f. \text{restrict } f \ A) \ ' S}. \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x))$ using A

by $(\text{intro infsetsum-cong-neutral neutral-left neutral-right refl})$

also have $\dots = (\sum_{a f \in S}. \prod_{x \in A}. \text{pmf } (p \ x) \ (\text{restrict } f \ A \ x))$

by $(\text{rule infsetsum-reindex}) \ (\text{force simp: inj-on-def fun-eq-iff S-def})$

also have $\dots = (\sum_{a f \in S}. \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x))$

by $(\text{intro infsetsum-cong}) \ (\text{auto simp: S-def})$

also have $\dots = (\sum_{a f. \text{if } \forall x. x \notin A \longrightarrow f \ x = \text{dflt} \text{ then } \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x) \text{ else } 0)$

by $(\text{intro infsetsum-cong-neutral}) \ (\text{auto simp: S-def})$

also have $\text{ennreal } \dots = (\int^+ f. \text{ennreal } (\text{if } \forall x. x \notin A \longrightarrow f \ x = \text{dflt}$

$\text{then } \prod_{x \in A}. \text{pmf } (p \ x) \ (f \ x) \text{ else } 0) \ \partial \text{count-space UNIV})$

by $(\text{intro nn-integral-conv-infsetsum [symmetric] summable}) \ (\text{auto simp: prod-nonneg})$

finally show $?case$ by simp

qed $(\text{auto simp: prod-nonneg})$

lemma Pi-pmf-cong :

assumes $A = A' \ \text{dflt} = \text{dflt}' \ \bigwedge x. x \in A \implies f \ x = f' \ x$

shows $\text{Pi-pmf } A \ \text{dflt } f = \text{Pi-pmf } A' \ \text{dflt}' \ f'$

proof –

have $(\lambda fa. \text{if } \forall x. x \notin A \longrightarrow fa \ x = \text{dflt} \text{ then } \prod_{x \in A}. \text{pmf } (f \ x) \ (fa \ x) \text{ else } 0) =$

$(\lambda f. \text{if } \forall x. x \notin A' \longrightarrow f \ x = \text{dflt}' \text{ then } \prod_{x \in A'}. \text{pmf } (f' \ x) \ (f \ x) \text{ else } 0)$

using assms by $(\text{intro ext}) \ (\text{auto intro!: prod.cong})$

thus $?thesis$

by $(\text{simp only: Pi-pmf-def})$

qed

lemma *pmf-Pi'*:

assumes *finite A* $\bigwedge x. x \notin A \implies f\ x = \text{dflt}$
shows $\text{pmf } (\text{Pi-pmf } A\ \text{dflt } p)\ f = (\prod_{x \in A}. \text{pmf } (p\ x)\ (f\ x))$
using *assms* **by** (*subst pmf-Pi*) *auto*

lemma *pmf-Pi-outside*:

assumes *finite A* $\exists x. x \notin A \wedge f\ x \neq \text{dflt}$
shows $\text{pmf } (\text{Pi-pmf } A\ \text{dflt } p)\ f = 0$
using *assms* **by** (*subst pmf-Pi*) *auto*

lemma *pmf-Pi-empty* [*simp*]: $\text{Pi-pmf } \{\} \text{dflt } p = \text{return-pmf } (\lambda-. \text{dflt})$

by (*intro pmf-eqI*, *subst pmf-Pi*) (*auto simp: indicator-def*)

lemma *set-Pi-pmf-subset*: $\text{finite } A \implies \text{set-pmf } (\text{Pi-pmf } A\ \text{dflt } p) \subseteq \{f. \forall x. x \notin A \longrightarrow f\ x = \text{dflt}\}$

by (*auto simp: set-pmf-eq pmf-Pi*)

26.3 Dependent product sets with a default

The following describes a dependent product of sets where the functions are required to return the default value *dflt* outside their domain, in analogy to *PiE*, which uses *undefined*.

definition *PiE-dflt*

where $\text{PiE-dflt } A\ \text{dflt } B = \{f. \forall x. (x \in A \longrightarrow f\ x \in B\ x) \wedge (x \notin A \longrightarrow f\ x = \text{dflt})\}$

lemma *restrict-PiE-dflt*: $(\lambda h. \text{restrict } h\ A) \text{ ' PiE-dflt } A\ \text{dflt } B = \text{PiE } A\ B$

proof (*intro equalityI subsetI*)

fix *h* **assume** $h \in (\lambda h. \text{restrict } h\ A) \text{ ' PiE-dflt } A\ \text{dflt } B$

thus $h \in \text{PiE } A\ B$

by (*auto simp: PiE-dflt-def*)

next

fix *h* **assume** $h: h \in \text{PiE } A\ B$

hence $\text{restrict } (\lambda x. \text{if } x \in A \text{ then } h\ x \text{ else dflt})\ A \in (\lambda h. \text{restrict } h\ A) \text{ ' PiE-dflt } A\ \text{dflt } B$

by (*intro imageI*) (*auto simp: PiE-def extensional-def PiE-dflt-def*)

also have $\text{restrict } (\lambda x. \text{if } x \in A \text{ then } h\ x \text{ else dflt})\ A = h$

using *h* **by** (*auto simp: fun-eq-iff*)

finally show $h \in (\lambda h. \text{restrict } h\ A) \text{ ' PiE-dflt } A\ \text{dflt } B$.

qed

lemma *dflt-image-PiE*: $(\lambda h\ x. \text{if } x \in A \text{ then } h\ x \text{ else dflt}) \text{ ' PiE } A\ B = \text{PiE-dflt } A\ \text{dflt } B$

(*is ?f ' ?X = ?Y*)

proof (*intro equalityI subsetI*)

fix *h* **assume** $h \in ?f \text{ ' ?X}$

thus $h \in ?Y$

by (auto simp: PiE-dflt-def PiE-def)
 next
 fix h assume h: $h \in ?Y$
 hence $?f \text{ (restrict } h \text{ } A) \in ?f \text{ ' } ?X$
 by (intro imageI) (auto simp: PiE-def extensional-def PiE-dflt-def)
 also have $?f \text{ (restrict } h \text{ } A) = h$
 using h by (auto simp: fun-eq-iff PiE-dflt-def)
 finally show $h \in ?f \text{ ' } ?X$.
 qed

lemma finite-PiE-dflt [intro]:
 assumes finite A $\bigwedge x. x \in A \implies \text{finite } (B \text{ } x)$
 shows finite (PiE-dflt A d B)
 proof –
 have PiE-dflt A d B = $(\lambda f x. \text{if } x \in A \text{ then } f \text{ } x \text{ else } d) \text{ ' } \text{PiE } A \text{ } B$
 by (rule dflt-image-PiE [symmetric])
 also have finite ...
 by (intro finite-imageI finite-PiE assms)
 finally show ?thesis .
 qed

lemma card-PiE-dflt:
 assumes finite A $\bigwedge x. x \in A \implies \text{finite } (B \text{ } x)$
 shows card (PiE-dflt A d B) = $(\prod x \in A. \text{card } (B \text{ } x))$
 proof –
 from assms have $(\prod x \in A. \text{card } (B \text{ } x)) = \text{card } (\text{PiE } A \text{ } B)$
 by (intro card-PiE [symmetric]) auto
 also have PiE A B = $(\lambda f. \text{restrict } f \text{ } A) \text{ ' } \text{PiE-dflt } A \text{ } d \text{ } B$
 by (rule restrict-PiE-dflt [symmetric])
 also have card ... = card (PiE-dflt A d B)
 by (intro card-image) (force simp: inj-on-def restrict-def fun-eq-iff PiE-dflt-def)
 finally show ?thesis ..
 qed

lemma PiE-dflt-empty-iff [simp]: $\text{PiE-dflt } A \text{ } d \text{ } B = \{\}$ $\longleftrightarrow (\exists x \in A. B \text{ } x = \{\})$
 by (simp add: dflt-image-PiE [symmetric] PiE-eq-empty-iff)

lemma set-Pi-pmf-subset':
 assumes finite A
 shows $\text{set-pmf } (\text{Pi-pmf } A \text{ } d \text{ } p) \subseteq \text{PiE-dflt } A \text{ } d \text{ } (\text{set-pmf } \circ p)$
 using assms by (auto simp: set-pmf-eq pmf-Pi PiE-dflt-def)

lemma set-Pi-pmf:
 assumes finite A
 shows $\text{set-pmf } (\text{Pi-pmf } A \text{ } d \text{ } p) = \text{PiE-dflt } A \text{ } d \text{ } (\text{set-pmf } \circ p)$
 proof (rule equalityI)
 show $\text{PiE-dflt } A \text{ } d \text{ } (\text{set-pmf } \circ p) \subseteq \text{set-pmf } (\text{Pi-pmf } A \text{ } d \text{ } p)$
 proof safe
 fix f assume f: $f \in \text{PiE-dflt } A \text{ } d \text{ } (\text{set-pmf } \circ p)$

hence $\text{pmf } (Pi\text{-pmf } A \text{ dflt } p) f = (\prod_{x \in A}. \text{pmf } (p \ x) (f \ x))$
 using *assms* by (*auto simp: pmf-Pi PiE-dflt-def*)
 also have $\dots > 0$
 using *f* by (*intro prod-pos*) (*auto simp: PiE-dflt-def set-pmf-eq*)
 finally show $f \in \text{set-pmf } (Pi\text{-pmf } A \text{ dflt } p)$
 by (*auto simp: set-pmf-eq*)
 qed
 qed (*use set-Pi-pmf-subset'[OF assms, of dflt p] in auto*)

The probability of an independent combination of events is precisely the product of the probabilities of each individual event.

lemma *measure-Pi-pmf-PiE-dflt*:

assumes [*simp*]: *finite A*
 shows $\text{measure-pmf.prob } (Pi\text{-pmf } A \text{ dflt } p) (PiE\text{-dflt } A \text{ dflt } B) =$
 $(\prod_{x \in A}. \text{measure-pmf.prob } (p \ x) (B \ x))$
 proof –
 define $B' \text{ where } B' = (\lambda x. B \ x \cap \text{set-pmf } (p \ x))$
 have $\text{measure-pmf.prob } (Pi\text{-pmf } A \text{ dflt } p) (PiE\text{-dflt } A \text{ dflt } B) =$
 $(\sum_{a \ h \in PiE\text{-dflt } A \text{ dflt } B}. \text{pmf } (Pi\text{-pmf } A \text{ dflt } p) h)$
 by (*rule measure-pmf-conv-infsetsum*)
 also have $\dots = (\sum_{a \ h \in PiE\text{-dflt } A \text{ dflt } B}. \prod_{x \in A}. \text{pmf } (p \ x) (h \ x))$
 by (*intro infsetsum-cong, subst pmf-Pi'*) (*auto simp: PiE-dflt-def*)
 also have $\dots = (\sum_{a \ h \in (\lambda h. \text{restrict } h \ A) \ ' \ PiE\text{-dflt } A \text{ dflt } B}. \prod_{x \in A}. \text{pmf } (p \ x) (h \ x))$
 by (*subst infsetsum-reindex*) (*force simp: inj-on-def PiE-dflt-def fun-eq-iff*) +
 also have $(\lambda h. \text{restrict } h \ A) \ ' \ PiE\text{-dflt } A \text{ dflt } B = PiE \ A \ B$
 by (*rule restrict-PiE-dflt*)
 also have $(\sum_{a \ h \in PiE \ A \ B}. \prod_{x \in A}. \text{pmf } (p \ x) (h \ x)) = (\sum_{a \ h \in PiE \ A \ B'}. \prod_{x \in A}. \text{pmf } (p \ x) (h \ x))$
 by (*intro infsetsum-cong-neutral*) (*auto simp: B'-def set-pmf-eq*)
 also have $(\sum_{a \ h \in PiE \ A \ B'}. \prod_{x \in A}. \text{pmf } (p \ x) (h \ x)) = (\prod_{x \in A}. \text{infsetsum } (\text{pmf } (p \ x)) (B' \ x))$
 by (*intro infsetsum-prod-PiE*) (*auto simp: B'-def*)
 also have $\dots = (\prod_{x \in A}. \text{infsetsum } (\text{pmf } (p \ x)) (B \ x))$
 by (*intro prod.cong infsetsum-cong-neutral*) (*auto simp: B'-def set-pmf-eq*)
 also have $\dots = (\prod_{x \in A}. \text{measure-pmf.prob } (p \ x) (B \ x))$
 by (*subst measure-pmf-conv-infsetsum*) (*rule refl*)
 finally show *?thesis* .
 qed

lemma *measure-Pi-pmf-Pi*:

fixes $t :: \text{nat}$
 assumes [*simp*]: *finite A*
 shows $\text{measure-pmf.prob } (Pi\text{-pmf } A \text{ dflt } p) (Pi \ A \ B) =$
 $(\prod_{x \in A}. \text{measure-pmf.prob } (p \ x) (B \ x))$ (*is ?lhs = ?rhs*)
 proof –
 have $?lhs = \text{measure-pmf.prob } (Pi\text{-pmf } A \text{ dflt } p) (PiE\text{-dflt } A \text{ dflt } B)$
 by (*intro measure-prob-cong-0*)
 (*auto simp: PiE-dflt-def PiE-def intro!: pmf-Pi-outside*) +

```

also have ... = ?rhs
using assms by (simp add: measure-Pi-pmf-PiE-dflt)
finally show ?thesis
by simp
qed

```

26.4 Common PMF operations on products

Pi-pmf distributes over the ‘bind’ operation in the Giry monad:

lemma *Pi-pmf-bind*:

```

assumes finite A
shows Pi-pmf A d ( $\lambda x. \text{bind-pmf } (p \ x) \ (q \ x) =$ 
 $\text{do } \{f \leftarrow \text{Pi-pmf } A \ d' \ p; \text{Pi-pmf } A \ d \ (\lambda x. q \ x \ (f \ x))\}$  (is ?lhs = ?rhs)
proof (rule pmf-eqI, goal-cases)
case (1 f)
show ?case
proof (cases  $\exists x \in A. f \ x \neq d$ )
case False
define B where B = ( $\lambda x. \text{set-pmf } (p \ x)$ )
have [simp]: countable (B x) for x by (auto simp: B-def)

{
  fix x :: 'a
  have ( $\lambda a. \text{pmf } (p \ x) \ a \ * \ 1$ ) abs-summable-on B x
    by (simp add: pmf-abs-summable)
  moreover have norm ( $\text{pmf } (p \ x) \ a \ * \ 1$ )  $\geq \text{norm } (\text{pmf } (p \ x) \ a \ * \ \text{pmf } (q \ x$ 
a) (f x)) for a
    unfolding norm-mult by (intro mult-left-mono) (auto simp: pmf-le-1)
    ultimately have ( $\lambda a. \text{pmf } (p \ x) \ a \ * \ \text{pmf } (q \ x \ a) \ (f \ x)$ ) abs-summable-on B x
      by (rule abs-summable-on-comparison-test)
} note summable = this

have  $\text{pmf } ?rhs \ f = (\sum a g. \text{pmf } (\text{Pi-pmf } A \ d' \ p) \ g \ * \ (\prod x \in A. \text{pmf } (q \ x \ (g \ x))$ 
(f x)))
by (subst pmf-bind, subst pmf-Pi')
  (insert assms False, simp-all add: pmf-expectation-eq-infsetsum)
also have ... = ( $\sum a g \in \text{PiE-dflt } A \ d' \ B. \text{pmf } (\text{Pi-pmf } A \ d' \ p) \ g \ * \ (\prod x \in A. \text{pmf } (q \ x \ (g \ x)) \ (f \ x))$ )
unfolding B-def
using assms by (intro infsetsum-cong-neutral) (auto simp: pmf-Pi PiE-dflt-def set-pmf-eq)
also have ... = ( $\sum a g \in \text{PiE-dflt } A \ d' \ B. (\prod x \in A. \text{pmf } (p \ x) \ (g \ x) \ * \ \text{pmf } (q \ x \ (g \ x)) \ (f \ x))$ )
using assms by (intro infsetsum-cong) (auto simp: pmf-Pi PiE-dflt-def prod.distrib)
also have ... = ( $\sum a g \in (\lambda g. \text{restrict } g \ A) \ ' \ \text{PiE-dflt } A \ d' \ B. (\prod x \in A. \text{pmf } (p \ x) \ (g \ x) \ * \ \text{pmf } (q \ x \ (g \ x)) \ (f \ x))$ )
by (subst infsetsum-reindex) (force simp: PiE-dflt-def inj-on-def fun-eq-iff)+
also have ( $\lambda g. \text{restrict } g \ A$ ) ' PiE-dflt A d' B = PiE A B

```

```

    by (rule restrict-PiE-dflt)
  also have  $(\sum_{a \in B} (\prod_{x \in A} \text{pmf } (p \ x) (g \ x) * \text{pmf } (q \ x \ (g \ x)) (f \ x))) =$ 
     $(\prod_{x \in A} \sum_{a \in B} \text{pmf } (p \ x) a * \text{pmf } (q \ x \ a) (f \ x))$ 
    using assms summable by (subst infsetsum-prod-PiE) simp-all
  also have  $\dots = (\prod_{x \in A} \sum_a \text{pmf } (p \ x) a * \text{pmf } (q \ x \ a) (f \ x))$ 
    by (intro prod.cong infsetsum-cong-neutral) (auto simp: B-def set-pmf-eq)
  also have  $\dots = \text{pmf } ?lhs \ f$ 
    using False assms by (subst pmf-Pi') (simp-all add: pmf-bind pmf-expectation-eq-infsetsum)
  finally show ?thesis ..
next
case True
have  $\text{pmf } ?rhs \ f =$ 
     $\text{measure-pmf.expectation } (Pi\text{-pmf } A \ d' \ p) (\lambda x. \text{pmf } (Pi\text{-pmf } A \ d \ (\lambda xa. \ q$ 
 $xa \ (x \ xa))) \ f)$ 
    using assms by (simp add: pmf-bind)
  also have  $\dots = \text{measure-pmf.expectation } (Pi\text{-pmf } A \ d' \ p) (\lambda x. \ 0)$ 
    using assms True by (intro Bochner-Integration.integral-cong pmf-Pi-outside)
auto
  also have  $\dots = \text{pmf } ?lhs \ f$ 
    using assms True by (subst pmf-Pi-outside) auto
  finally show ?thesis ..
qed
qed

```

lemma *Pi-pmf-return-pmf* [*simp*]:
assumes *finite A*
shows $Pi\text{-pmf } A \ dflt \ (\lambda x. \text{return-pmf } (f \ x)) = \text{return-pmf } (\lambda x. \text{if } x \in A \text{ then } f \ x \text{ else } dflt)$
using *assms* by (intro *pmf-eqI*) (auto *simp: pmf-Pi simp: indicator-def split: if-splits*)

Analogously any componentwise mapping can be pulled outside the product:

lemma *Pi-pmf-map*:
assumes [*simp*]: *finite A* **and** $f \ dflt = dflt'$
shows $Pi\text{-pmf } A \ dflt' \ (\lambda x. \text{map-pmf } f \ (g \ x)) = \text{map-pmf } (\lambda h. \ f \circ h) (Pi\text{-pmf } A \ dflt \ g)$
proof –
 have $Pi\text{-pmf } A \ dflt' \ (\lambda x. \text{map-pmf } f \ (g \ x)) =$
 $Pi\text{-pmf } A \ dflt' \ (\lambda x. \ g \ x \gg (\lambda x. \text{return-pmf } (f \ x)))$
 using *assms* by (*simp add: map-pmf-def Pi-pmf-bind*)
 also have $\dots = Pi\text{-pmf } A \ dflt \ g \gg (\lambda h. \text{return-pmf } (\lambda x. \text{if } x \in A \text{ then } f \ (h \ x) \text{ else } dflt'))$
 by (subst *Pi-pmf-bind*[**where** $d' = dflt$]) auto
 also have $\dots = \text{map-pmf } (\lambda h. \ f \circ h) (Pi\text{-pmf } A \ dflt \ g)$
 unfolding *map-pmf-def* using *set-Pi-pmf-subset'*[*of A dflt g*]
 by (intro *bind-pmf-cong refl arg-cong*[*of - - return-pmf*])
 (auto *dest: simp: fun-eq-iff PiE-dflt-def assms*(2))
 finally show ?thesis .
qed

We can exchange the default value in a product of PMFs like this:

lemma *Pi-pmf-default-swap*:

assumes *finite A*

shows $\text{map-pmf } (\lambda f x. \text{if } x \in A \text{ then } f x \text{ else } \text{dflt}') \text{ (Pi-pmf } A \text{ dflt } p) =$
 $\text{Pi-pmf } A \text{ dflt}' p \text{ (is ?lhs = ?rhs)}$

proof (*rule pmf-eqI, goal-cases*)

case (*1 f*)

let $?B = (\lambda f x. \text{if } x \in A \text{ then } f x \text{ else } \text{dflt}') - \{f\} \cap \text{PiE-dflt } A \text{ dflt } (\lambda -. \text{UNIV})$

show *?case*

proof (*cases $\exists x \in A. f x \neq \text{dflt}'$*)

case *False*

let $?f' = \lambda x. \text{if } x \in A \text{ then } f x \text{ else } \text{dflt}$

from *False* **have** $\text{pmf } ?\text{lhs } f = \text{measure-pmf.prob (Pi-pmf } A \text{ dflt } p) \text{ ?B}$

using *assms unfolding pmf-map*

by (*intro measure-prob-cong-0*) (*auto simp: PiE-dflt-def pmf-Pi-outside*)

also from *False* **have** $?B = \{?f'\}$

by (*auto simp: fun-eq-iff PiE-dflt-def*)

also have $\text{measure-pmf.prob (Pi-pmf } A \text{ dflt } p) \{?f'\} = \text{pmf (Pi-pmf } A \text{ dflt } p)$

?f'

by (*simp add: measure-pmf-single*)

also have $\dots = \text{pmf } ?\text{rhs } f$

using *False assms* **by** (*subst (1 2) pmf-Pi*) *auto*

finally show *?thesis* .

next

case *True*

have $\text{pmf } ?\text{lhs } f = \text{measure-pmf.prob (Pi-pmf } A \text{ dflt } p) \text{ ?B}$

using *assms unfolding pmf-map*

by (*intro measure-prob-cong-0*) (*auto simp: PiE-dflt-def pmf-Pi-outside*)

also from *True* **have** $?B = \{\}$ **by** *auto*

also have $\text{measure-pmf.prob (Pi-pmf } A \text{ dflt } p) \dots = 0$

by *simp*

also have $0 = \text{pmf } ?\text{rhs } f$

using *True assms* **by** (*intro pmf-Pi-outside [symmetric]*) *auto*

finally show *?thesis* .

qed

qed

The following rule allows reindexing the product:

lemma *Pi-pmf-bij-betw*:

assumes *finite A bij-betw h A B $\bigwedge x. x \notin A \implies h x \notin B$*

shows $\text{Pi-pmf } A \text{ dflt } (\lambda -. f) = \text{map-pmf } (\lambda g. g \circ h) \text{ (Pi-pmf } B \text{ dflt } (\lambda -. f))$
(is ?lhs = ?rhs)

proof –

have *B: finite B*

using *assms bij-betw-finite* **by** *auto*

have $\text{pmf } ?\text{lhs } g = \text{pmf } ?\text{rhs } g$ **for** *g*

proof (*cases $\forall a. a \notin A \longrightarrow g a = \text{dflt}$*)

case *True*

define *h'* **where** $h' = \text{the-inv-into } A \text{ } h$

```

have h': h' (h x) = x if x ∈ A for x
unfolding h'-def using that assms by (auto simp add: bij-betw-def the-inv-into-f-f)
have h: h (h' x) = x if x ∈ B for x
  unfolding h'-def using that assms f-the-inv-into-f-bij-betw by fastforce
have pmf ?rhs g = measure-pmf.prob (Pi-pmf B dflt (λ-. f)) ((λg. g ∘ h) - '
{g})
  unfolding pmf-map by simp
also have ... = measure-pmf.prob (Pi-pmf B dflt (λ-. f))
  (((λg. g ∘ h) - ' {g}) ∩ PiE-dflt B dflt (λ-. UNIV))
using B by (intro measure-prob-cong-0) (auto simp: PiE-dflt-def pmf-Pi-outside)
also have ... = pmf (Pi-pmf B dflt (λ-. f)) (λx. if x ∈ B then g (h' x) else
dflt)
proof -
  have (if h x ∈ B then g (h' (h x)) else dflt) = g x for x
    using h' assms True by (cases x ∈ A) (auto simp add: bij-betwE)
  then have (λg. g ∘ h) - ' {g} ∩ PiE-dflt B dflt (λ-. UNIV) =
    {(λx. if x ∈ B then g (h' x) else dflt)}
    using assms h' h True unfolding PiE-dflt-def by auto
  then show ?thesis
    by (simp add: measure-pmf-single)
qed
also have ... = pmf (Pi-pmf A dflt (λ-. f)) g
  using B assms True h'-def
  by (auto simp add: pmf-Pi intro!: prod.reindex-bij-betw bij-betw-the-inv-into)
finally show ?thesis
  by simp
next
case False
have pmf ?rhs g = infsetsum (pmf (Pi-pmf B dflt (λ-. f))) ((λg. g ∘ h) - ' {g})
  using assms by (auto simp add: measure-pmf-conv-infsetsum pmf-map)
also have ... = infsetsum (λ-. 0) ((λg x. g (h x)) - ' {g})
  using B False assms by (intro infsetsum-cong pmf-Pi-outside) fastforce+
also have ... = 0
  by simp
finally show ?thesis
  using assms False by (auto simp add: pmf-Pi pmf-map)
qed
then show ?thesis
  by (rule pmf-eqI)
qed

```

A product of uniform random choices is again a uniform distribution.

lemma *Pi-pmf-of-set*:

```

assumes finite A ∧ x. x ∈ A ⇒ finite (B x) ∧ x. x ∈ A ⇒ B x ≠ {}
shows Pi-pmf A d (λx. pmf-of-set (B x)) = pmf-of-set (PiE-dflt A d B) (is
?lhs = ?rhs)
proof (rule pmf-eqI, goal-cases)
  case (1 f)
  show ?case

```

```

proof (cases  $\exists x. x \notin A \wedge f x \neq d$ )
  case True
    hence  $\text{pmf } ?\text{lhs } f = 0$ 
    using assms by (intro pmf-Pi-outside) (auto simp: PiE-dflt-def)
    also from True have  $f \notin \text{PiE-dflt } A \text{ } d \text{ } B$ 
    by (auto simp: PiE-dflt-def)
    hence  $0 = \text{pmf } ?\text{rhs } f$ 
    using assms by (subst pmf-of-set) auto
    finally show ?thesis .
  next
    case False
    hence  $\text{pmf } ?\text{lhs } f = (\prod_{x \in A. \text{pmf } (\text{pmf-of-set } (B \text{ } x)) (f \text{ } x)) (f \text{ } x)$ 
    using assms by (subst pmf-Pi') auto
    also have  $\dots = (\prod_{x \in A. \text{indicator } (B \text{ } x) (f \text{ } x) / \text{real } (\text{card } (B \text{ } x)))$ 
    by (intro prod.cong refl, subst pmf-of-set) (use assms False in auto)
    also have  $\dots = (\prod_{x \in A. \text{indicator } (B \text{ } x) (f \text{ } x)) / \text{real } (\prod_{x \in A. \text{card } (B \text{ } x)})$ 
    by (subst prod-dividef) simp-all
    also have  $(\prod_{x \in A. \text{indicator } (B \text{ } x) (f \text{ } x) :: \text{real}) = \text{indicator } (\text{PiE-dflt } A \text{ } d \text{ } B) f$ 
    using assms False by (auto simp: indicator-def PiE-dflt-def)
    also have  $(\prod_{x \in A. \text{card } (B \text{ } x)}) = \text{card } (\text{PiE-dflt } A \text{ } d \text{ } B)$ 
    using assms by (intro card-PiE-dflt [symmetric]) auto
    also have  $\text{indicator } (\text{PiE-dflt } A \text{ } d \text{ } B) f / \dots = \text{pmf } ?\text{rhs } f$ 
    using assms by (intro pmf-of-set [symmetric]) auto
    finally show ?thesis .
qed
qed

```

26.5 Merging and splitting PMF products

The following lemma shows that we can add a single PMF to a product:

lemma *Pi-pmf-insert*:

```

assumes finite  $A \text{ } x \notin A$ 
shows  $\text{Pi-pmf } (\text{insert } x \text{ } A) \text{ dflt } p = \text{map-pmf } (\lambda(y, f). f(x := y)) (\text{pair-pmf } (p \text{ } x) (\text{Pi-pmf } A \text{ dflt } p))$ 
proof (intro pmf-eqI)
  fix  $f$ 
  let  $?M = \text{pair-pmf } (p \text{ } x) (\text{Pi-pmf } A \text{ dflt } p)$ 
  have  $\text{pmf } (\text{map-pmf } (\lambda(y, f). f(x := y)) ?M) f =$ 
     $\text{measure-pmf.prob } ?M ((\lambda(y, f). f(x := y)) -' \{f\})$ 
  by (subst pmf-map) auto
  also have  $((\lambda(y, f). f(x := y)) -' \{f\}) = (\bigcup y'. \{(f \text{ } x, f(x := y'))\})$ 
  by (auto simp: fun-upd-def fun-eq-iff)
  also have  $\text{measure-pmf.prob } ?M \dots = \text{measure-pmf.prob } ?M \{(f \text{ } x, f(x := \text{dflt}))\}$ 
  using assms by (intro measure-prob-cong-0) (auto simp: pmf-pair pmf-Pi split: if-splits)
  also have  $\dots = \text{pmf } (p \text{ } x) (f \text{ } x) * \text{pmf } (\text{Pi-pmf } A \text{ dflt } p) (f(x := \text{dflt}))$ 
  by (simp add: measure-pmf-single pmf-pair pmf-Pi)
  also have  $\dots = \text{pmf } (\text{Pi-pmf } (\text{insert } x \text{ } A) \text{ dflt } p) f$ 
  proof (cases  $\forall y. y \notin \text{insert } x \text{ } A \longrightarrow f \text{ } y = \text{dflt}$ )

```

```

case True
with assms have  $\text{pmf } (p \ x) \ (f \ x) * \text{pmf } (Pi\text{-pmf } A \ \text{dflt } p) \ (f(x := \text{dflt})) =$ 
 $\text{pmf } (p \ x) \ (f \ x) * (\prod_{xa \in A}. \text{pmf } (p \ xa) \ ((f(x := \text{dflt})) \ xa))$ 
by (subst pmf-Pi') auto
also have  $(\prod_{xa \in A}. \text{pmf } (p \ xa) \ ((f(x := \text{dflt})) \ xa)) = (\prod_{xa \in A}. \text{pmf } (p \ xa) \ (f$ 
 $\text{xa}))$ 
using assms by (intro prod.cong) auto
also have  $\text{pmf } (p \ x) \ (f \ x) * \dots = \text{pmf } (Pi\text{-pmf } (\text{insert } x \ A) \ \text{dflt } p) \ f$ 
using assms True by (subst pmf-Pi') auto
finally show ?thesis .
qed (insert assms, auto simp: pmf-Pi)
finally show  $\dots = \text{pmf } (\text{map-pmf } (\lambda(y, f). f(x := y)) \ ?M) \ f \ ..$ 
qed

```

```

lemma Pi-pmf-insert':
assumes finite A  $x \notin A$ 
shows  $Pi\text{-pmf } (\text{insert } x \ A) \ \text{dflt } p =$ 
 $\text{do } \{y \leftarrow p \ x; f \leftarrow Pi\text{-pmf } A \ \text{dflt } p; \text{return-pmf } (f(x := y))\}$ 
using assms
by (subst Pi-pmf-insert)
 $(\text{auto simp add: map-pmf-def pair-pmf-def case-prod-beta' bind-return-pmf}$ 
 $\text{bind-assoc-pmf})$ 

```

```

lemma Pi-pmf-singleton:
 $Pi\text{-pmf } \{x\} \ \text{dflt } p = \text{map-pmf } (\lambda a \ b. \text{if } b = x \text{ then } a \text{ else } \text{dflt}) \ (p \ x)$ 
proof –
have  $Pi\text{-pmf } \{x\} \ \text{dflt } p = \text{map-pmf } (\text{fun-upd } (\lambda-. \ \text{dflt}) \ x) \ (p \ x)$ 
by (subst Pi-pmf-insert) (simp-all add: pair-return-pmf2 pmf.map-comp o-def)
also have  $\text{fun-upd } (\lambda-. \ \text{dflt}) \ x = (\lambda z \ y. \text{if } y = x \text{ then } z \text{ else } \text{dflt})$ 
by (simp add: fun-upd-def fun-eq-iff)
finally show ?thesis .
qed

```

Projecting a product of PMFs onto a component yields the expected result:

```

lemma Pi-pmf-component:
assumes finite A
shows  $\text{map-pmf } (\lambda f. f \ x) \ (Pi\text{-pmf } A \ \text{dflt } p) = (\text{if } x \in A \text{ then } p \ x \text{ else } \text{return-pmf}$ 
 $\text{dflt})$ 
proof (cases x ∈ A)
case True
define  $A'$  where  $A' = A - \{x\}$ 
from assms and True have  $A': A = \text{insert } x \ A'$ 
by (auto simp: A'-def)
from assms have  $\text{map-pmf } (\lambda f. f \ x) \ (Pi\text{-pmf } A \ \text{dflt } p) = p \ x$  unfolding  $A'$ 
by (subst Pi-pmf-insert)
 $(\text{auto simp: A'-def pmf.map-comp o-def case-prod-unfold map-fst-pair-pmf})$ 
with True show ?thesis by simp
next
case False

```



```

have map-pmf ( $\lambda f. f\ x$ ) ( $Pi\text{-}pmf\ A\ dflt\ p$ ) = map-pmf ( $\lambda -. dflt$ ) ( $Pi\text{-}pmf\ A\ dflt\ p$ )
using assms False set-Pi-pmf-subset[of A dflt p]
by (intro pmf.map-cong refl) (auto simp: set-pmf-eq pmf-Pi-outside)
with False show ?thesis by simp
qed

```

We can take merge two PMF products on disjoint sets like this:

```

lemma Pi-pmf-union:
  assumes finite A finite B A  $\cap$  B = {}
  shows  $Pi\text{-}pmf\ (A \cup B)\ dflt\ p =$ 
     $map\text{-}pmf\ (\lambda(f,g)\ x.\ if\ x \in A\ then\ f\ x\ else\ g\ x)$ 
     $(pair\text{-}pmf\ (Pi\text{-}pmf\ A\ dflt\ p)\ (Pi\text{-}pmf\ B\ dflt\ p))\ (is\ - =\ map\text{-}pmf\ (?h\ A)$ 
     $(?q\ A))$ 
  using assms(1,3)
proof (induction rule: finite-induct)
  case (insert x A)
  have  $map\text{-}pmf\ (?h\ (insert\ x\ A))\ (?q\ (insert\ x\ A)) =$ 
     $do\ \{v \leftarrow p\ x; (f, g) \leftarrow pair\text{-}pmf\ (Pi\text{-}pmf\ A\ dflt\ p)\ (Pi\text{-}pmf\ B\ dflt\ p);$ 
     $return\text{-}pmf\ (\lambda y.\ if\ y \in insert\ x\ A\ then\ (f(x := v))\ y\ else\ g\ y)\}$ 
  by (subst Pi-pmf-insert)
    (insert insert.hyps insert.premis,
    simp-all add: pair-pmf-def map-bind-pmf bind-map-pmf bind-assoc-pmf
bind-return-pmf)
  also have  $\dots = do\ \{v \leftarrow p\ x; (f, g) \leftarrow ?q\ A; return\text{-}pmf\ ((?h\ A\ (f,g))(x := v))\}$ 
  by (intro bind-pmf-cong refl) (auto simp: fun-eq-iff)
  also have  $\dots = do\ \{v \leftarrow p\ x; f \leftarrow map\text{-}pmf\ (?h\ A)\ (?q\ A); return\text{-}pmf\ (f(x :=$ 
     $v))\}$ 
  by (simp add: bind-map-pmf map-bind-pmf case-prod-unfold cong: if-cong)
  also have  $\dots = do\ \{v \leftarrow p\ x; f \leftarrow Pi\text{-}pmf\ (A \cup B)\ dflt\ p; return\text{-}pmf\ (f(x :=$ 
     $v))\}$ 
  using insert.hyps and insert.premis by (intro bind-pmf-cong insert.IH [symmetric]
refl) auto
  also have  $\dots = Pi\text{-}pmf\ (insert\ x\ (A \cup B))\ dflt\ p$ 
  by (subst Pi-pmf-insert)
    (insert assms insert.hyps insert.premis, auto simp: pair-pmf-def map-bind-pmf)
  also have  $insert\ x\ (A \cup B) = insert\ x\ A \cup B$ 
  by simp
  finally show ?case ..
qed (simp-all add: case-prod-unfold map-snd-pair-pmf)

```

We can also project a product to a subset of the indices by mapping all the other indices to the default value:

```

lemma Pi-pmf-subset:
  assumes finite A A'  $\subseteq$  A
  shows  $Pi\text{-}pmf\ A'\ dflt\ p = map\text{-}pmf\ (\lambda f\ x.\ if\ x \in A'\ then\ f\ x\ else\ dflt)\ (Pi\text{-}pmf\ A\ dflt\ p)$ 
proof –
  let ?P =  $pair\text{-}pmf\ (Pi\text{-}pmf\ A'\ dflt\ p)\ (Pi\text{-}pmf\ (A - A')\ dflt\ p)$ 

```

```

from assms have [simp]: finite  $A'$ 
  by (blast dest: finite-subset)
from assms have  $A = A' \cup (A - A')$ 
  by blast
also have  $Pi\text{-}pmf \dots dflt\ p = map\text{-}pmf\ (\lambda(f,g)\ x.\ if\ x \in A' \text{ then } f\ x \text{ else } g\ x)\ ?P$ 
  using assms by (intro Pi-pmf-union) auto
also have  $map\text{-}pmf\ (\lambda f\ x.\ if\ x \in A' \text{ then } f\ x \text{ else } dflt)\ \dots = map\text{-}pmf\ fst\ ?P$ 
  unfolding map-pmf-comp o-def case-prod-unfold
  using set-Pi-pmf-subset[of A' dflt p] by (intro map-pmf-cong refl) (auto simp:
fun-eq-iff)
also have  $\dots = Pi\text{-}pmf\ A'\ dflt\ p$ 
  by (simp add: map-fst-pair-pmf)
finally show ?thesis ..
qed

```

```

lemma Pi-pmf-subset':
  fixes  $f :: 'a \Rightarrow 'b\ pmf$ 
  assumes  $finite\ A\ B \subseteq A \wedge x. x \in A - B \implies f\ x = return\text{-}pmf\ dflt$ 
  shows  $Pi\text{-}pmf\ A\ dflt\ f = Pi\text{-}pmf\ B\ dflt\ f$ 
proof –
  have  $Pi\text{-}pmf\ (B \cup (A - B))\ dflt\ f =$ 
     $map\text{-}pmf\ (\lambda(f, g)\ x.\ if\ x \in B \text{ then } f\ x \text{ else } g\ x)$ 
     $(pair\text{-}pmf\ (Pi\text{-}pmf\ B\ dflt\ f)\ (Pi\text{-}pmf\ (A - B)\ dflt\ f))$ 
  using assms by (intro Pi-pmf-union) (auto dest: finite-subset)
  also have  $Pi\text{-}pmf\ (A - B)\ dflt\ f = Pi\text{-}pmf\ (A - B)\ dflt\ (\lambda\_. return\text{-}pmf\ dflt)$ 
  using assms by (intro Pi-pmf-cong) auto
  also have  $\dots = return\text{-}pmf\ (\lambda\_. dflt)$ 
  using assms by simp
  also have  $map\text{-}pmf\ (\lambda(f, g)\ x.\ if\ x \in B \text{ then } f\ x \text{ else } g\ x)$ 
     $(pair\text{-}pmf\ (Pi\text{-}pmf\ B\ dflt\ f)\ (return\text{-}pmf\ (\lambda\_. dflt))) =$ 
     $map\text{-}pmf\ (\lambda f\ x.\ if\ x \in B \text{ then } f\ x \text{ else } dflt)\ (Pi\text{-}pmf\ B\ dflt\ f)$ 
  by (simp add: map-pmf-def pair-pmf-def bind-assoc-pmf bind-return-pmf bind-return-pmf')
  also have  $\dots = Pi\text{-}pmf\ B\ dflt\ f$ 
  using assms by (intro Pi-pmf-default-swap) (auto dest: finite-subset)
  also have  $B \cup (A - B) = A$ 
  using assms by auto
  finally show ?thesis .
qed

```

```

lemma Pi-pmf-if-set:
  assumes finite  $A$ 
  shows  $Pi\text{-}pmf\ A\ dflt\ (\lambda x.\ if\ b\ x \text{ then } f\ x \text{ else } return\text{-}pmf\ dflt) =$ 
     $Pi\text{-}pmf\ \{x \in A.\ b\ x\}\ dflt\ f$ 
proof –
  have  $Pi\text{-}pmf\ A\ dflt\ (\lambda x.\ if\ b\ x \text{ then } f\ x \text{ else } return\text{-}pmf\ dflt) =$ 
     $Pi\text{-}pmf\ \{x \in A.\ b\ x\}\ dflt\ (\lambda x.\ if\ b\ x \text{ then } f\ x \text{ else } return\text{-}pmf\ dflt)$ 
  using assms by (intro Pi-pmf-subset') auto
  also have  $\dots = Pi\text{-}pmf\ \{x \in A.\ b\ x\}\ dflt\ f$ 
  by (intro Pi-pmf-cong) auto

```

finally show *?thesis* .
qed

lemma *Pi-pmf-if-set'*:
assumes *finite A*
shows $Pi\text{-}pmf\ A\ dflt\ (\lambda x. \text{if } b\ x \text{ then return-pmf } dflt \text{ else } f\ x) =$
 $Pi\text{-}pmf\ \{x \in A. \neg b\ x\}\ dflt\ f$
proof –
have $Pi\text{-}pmf\ A\ dflt\ (\lambda x. \text{if } b\ x \text{ then return-pmf } dflt \text{ else } f\ x) =$
 $Pi\text{-}pmf\ \{x \in A. \neg b\ x\}\ dflt\ (\lambda x. \text{if } b\ x \text{ then return-pmf } dflt \text{ else } f\ x)$
using *assms* **by** (*intro Pi-pmf-subset'*) *auto*
also have $\dots = Pi\text{-}pmf\ \{x \in A. \neg b\ x\}\ dflt\ f$
by (*intro Pi-pmf-cong*) *auto*
finally show *?thesis* .
qed

Lastly, we can delete a single component from a product:

lemma *Pi-pmf-remove*:
assumes *finite A*
shows $Pi\text{-}pmf\ (A - \{x\})\ dflt\ p = map\text{-}pmf\ (\lambda f. f(x := dflt))\ (Pi\text{-}pmf\ A\ dflt\ p)$
proof –
have $Pi\text{-}pmf\ (A - \{x\})\ dflt\ p =$
 $map\text{-}pmf\ (\lambda f\ xa. \text{if } xa \in A - \{x\} \text{ then } f\ xa \text{ else } dflt)\ (Pi\text{-}pmf\ A\ dflt\ p)$
using *assms* **by** (*intro Pi-pmf-subset*) *auto*
also have $\dots = map\text{-}pmf\ (\lambda f. f(x := dflt))\ (Pi\text{-}pmf\ A\ dflt\ p)$
using *set-Pi-pmf-subset[of A dflt p]* *assms*
by (*intro map-pmf-cong refl*) (*auto simp: fun-eq-iff*)
finally show *?thesis* .
qed

26.6 Additional properties

lemma *nn-integral-prod-Pi-pmf*:
assumes *finite A*
shows $nn\text{-}integral\ (Pi\text{-}pmf\ A\ dflt\ p)\ (\lambda y. \prod_{x \in A. f\ x\ (y\ x)}) = (\prod_{x \in A. nn\text{-}integral\ (p\ x)\ (f\ x)})$
using *assms*
proof (*induction rule: finite-induct*)
case (*insert x A*)
have $nn\text{-}integral\ (Pi\text{-}pmf\ (insert\ x\ A)\ dflt\ p)\ (\lambda y. \prod_{z \in insert\ x\ A. f\ z\ (y\ z)}) =$
 $(\int^+ a. \int^+ b. f\ x\ a * (\prod_{z \in A. f\ z\ (\text{if } z = x \text{ then } a \text{ else } b\ z))\ \partial Pi\text{-}pmf\ A\ dflt\ p\ \partial p\ x)$
using *insert* **by** (*auto simp: Pi-pmf-insert case-prod-unfold nn-integral-pair-pmf'*
cong: if-cong)
also have $(\lambda a\ b. \prod_{z \in A. f\ z\ (\text{if } z = x \text{ then } a \text{ else } b\ z)}) = (\lambda a\ b. \prod_{z \in A. f\ z\ (b\ z)})$
by (*intro ext prod.cong*) (*use insert.hyps in auto*)
also have $(\int^+ a. \int^+ b. f\ x\ a * (\prod_{z \in A. f\ z\ (b\ z)})\ \partial Pi\text{-}pmf\ A\ dflt\ p\ \partial p\ x) =$

$(\int^+ y. f x y \partial(p x)) * (\int^+ y. (\prod z \in A. f z (y z)) \partial(Pi\text{-}pmf A \text{ dflt } p))$
 by (simp add: nn-integral-multc nn-integral-cmult)
 also have $(\int^+ y. (\prod z \in A. f z (y z)) \partial(Pi\text{-}pmf A \text{ dflt } p)) = (\prod x \in A. \text{nn-integral } (p x) (f x))$
 by (rule insert.IH)
 also have $(\int^+ y. f x y \partial(p x)) * \dots = (\prod x \in \text{insert } x A. \text{nn-integral } (p x) (f x))$
 using insert.hyps by simp
 finally show ?case .
 qed auto

lemma *integrable-prod-Pi-pmf*:

fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c :: \{\text{real-normed-field, second-countable-topology, banach}\}$
 assumes $\text{finite } A$ and $\bigwedge x. x \in A \implies \text{integrable } (\text{measure-pmf } (p x)) (f x)$
 shows $\text{integrable } (\text{measure-pmf } (Pi\text{-}pmf A \text{ dflt } p)) (\lambda h. \prod x \in A. f x (h x))$
 proof (intro integrableI-bounded)
 have $(\int^+ x. \text{ennreal } (\text{norm } (\prod xa \in A. f xa (x xa))) \partial \text{measure-pmf } (Pi\text{-}pmf A \text{ dflt } p)) =$
 $(\int^+ x. (\prod y \in A. \text{ennreal } (\text{norm } (f y (x y)))) \partial \text{measure-pmf } (Pi\text{-}pmf A \text{ dflt } p))$
 by (simp flip: prod-norm prod-ennreal)
 also have $\dots = (\prod x \in A. \int^+ a. \text{ennreal } (\text{norm } (f x a)) \partial \text{measure-pmf } (p x))$
 by (intro nn-integral-prod-Pi-pmf) fact
 also have $(\int^+ a. \text{ennreal } (\text{norm } (f i a)) \partial \text{measure-pmf } (p i)) \neq \text{top}$ if $i: i \in A$
 for i
 using assms(2)[OF i] by (simp add: integrable-iff-bounded)
 hence $(\prod x \in A. \int^+ a. \text{ennreal } (\text{norm } (f x a)) \partial \text{measure-pmf } (p x)) \neq \text{top}$
 by (subst ennreal-prod-eq-top) auto
 finally show $(\int^+ x. \text{ennreal } (\text{norm } (\prod xa \in A. f xa (x xa))) \partial \text{measure-pmf } (Pi\text{-}pmf A \text{ dflt } p)) < \infty$
 by (simp add: top.not-eq-extremum)
 qed auto

lemma *expectation-prod-Pi-pmf*:

fixes $f :: - \Rightarrow - \Rightarrow \text{real}$
 assumes $\text{finite } A$
 assumes $\bigwedge x. x \in A \implies \text{integrable } (\text{measure-pmf } (p x)) (f x)$
 assumes $\bigwedge x y. x \in A \implies y \in \text{set-pmf } (p x) \implies f x y \geq 0$
 shows $\text{measure-pmf.expectation } (Pi\text{-}pmf A \text{ dflt } p) (\lambda y. \prod x \in A. f x (y x)) =$
 $(\prod x \in A. \text{measure-pmf.expectation } (p x) (\lambda v. f x v))$
 proof –
 have $\text{nonneg: measure-pmf.expectation } (p x) (f x) \geq 0$ if $x \in A$ for x
 using that by (intro Bochner-Integration.integral-nonneg-AE AE-pmfI assms)
 have $\text{nonneg': } 0 \leq \text{measure-pmf.expectation } (Pi\text{-}pmf A \text{ dflt } p) (\lambda y. \prod x \in A. f x (y x))$
 by (intro Bochner-Integration.integral-nonneg-AE AE-pmfI assms prod-nonneg)
 (use assms in (auto simp: set-Pi-pmf PiE-dflt-def))

 have $\text{ennreal } (\text{measure-pmf.expectation } (Pi\text{-}pmf A \text{ dflt } p) (\lambda y. \prod x \in A. f x (y x)))$
 =

```

      nn-integral (Pi-pmf A dflt p) (λy. ennreal (∏ x∈A. f x (y x))) using assms
    by (intro nn-integral-eq-integral [symmetric] assms integrable-prod-Pi-pmf)
      (auto simp: AE-measure-pmf-iff set-Pi-pmf PiE-dflt-def prod-nonneg)
  also have ... = nn-integral (Pi-pmf A dflt p) (λy. (∏ x∈A. ennreal (f x (y x))))
    by (intro nn-integral-cong-AE AE-pmfI prod-ennreal [symmetric])
      (use assms(1) in ⟨auto simp: set-Pi-pmf PiE-dflt-def intro!: assms(3)⟩)
  also have ... = (∏ x∈A. ∫+ a. ennreal (f x a) ∂measure-pmf (p x))
    by (rule nn-integral-prod-Pi-pmf) fact+
  also have ... = (∏ x∈A. ennreal (measure-pmf.expectation (p x) (f x)))
    by (intro prod.cong nn-integral-eq-integral assms AE-pmfI) auto
  also have ... = ennreal (∏ x∈A. measure-pmf.expectation (p x) (f x))
    by (intro prod-ennreal nonneg)
  finally show ?thesis
    using nonneg nonneg' by (subst (asm) ennreal-inj) (auto intro!: prod-nonneg)
qed

```

```

lemma indep-vars-Pi-pmf:
  assumes fin: finite I
  shows prob-space.indep-vars (measure-pmf (Pi-pmf I dflt p))
    (λ-. count-space UNIV) (λx f. f x) I
proof (cases I = {})
  case True
  show ?thesis
    by (subst prob-space.indep-vars-def [OF measure-pmf.prob-space-axioms],
        subst prob-space.indep-sets-def [OF measure-pmf.prob-space-axioms]) (simp-all
  add: True)
next
  case [simp]: False
  show ?thesis
  proof (subst prob-space.indep-vars-iff-distr-eq-PiM')
    show distr (measure-pmf (Pi-pmf I dflt p)) (PiM I (λi. count-space UNIV))
      (λx. restrict x I) =
        PiM I (λi. distr (measure-pmf (Pi-pmf I dflt p)) (count-space UNIV) (λf.
  f i))
  proof (rule product-sigma-finite.PiM-eqI, goal-cases)
    case 1
    interpret product-prob-space λi. distr (measure-pmf (Pi-pmf I dflt p))
      (count-space UNIV) (λf. f i)
    by (intro product-prob-spaceI prob-space.prob-space-distr measure-pmf.prob-space-axioms)
      simp-all
    show ?case by unfold-locales
  next
    case 3
    have sets (PiM I (λi. distr (measure-pmf (Pi-pmf I dflt p)) (count-space
  UNIV) (λf. f i))) =
      sets (PiM I (λ-. count-space UNIV))
    by (intro sets-PiM-cong) simp-all
    thus ?case by simp
  next

```

```

case (4 A)
have  $Pi_E I A \in sets (Pi_M I (\lambda i. count-space UNIV))$ 
using 4 by (intro sets-PiM-I-finite fin) auto
hence  $emeasure (distr (measure-pmf (Pi-pmf I dflt p)) (Pi_M I (\lambda i. count-space UNIV)))$ 
   $(\lambda x. restrict x I) (Pi_E I A) =$ 
   $emeasure (measure-pmf (Pi-pmf I dflt p)) ((\lambda x. restrict x I) - ' Pi_E I A)$ 
using 4 by (subst emeasure-distr) (auto simp: space-PiM)
also have  $\dots = emeasure (measure-pmf (Pi-pmf I dflt p)) (PiE-dflt I dflt A)$ 
by (intro emeasure-eq-AE AE-pmfI) (auto simp: PiE-dflt-def set-Pi-pmf fin)
also have  $\dots = (\prod_{i \in I. emeasure (measure-pmf (p i)) (A i))$ 
by (simp add: measure-pmf.emeasure-eq-measure measure-Pi-pmf-PiE-dflt
  fin prod-ennreal)
also have  $\dots = (\prod_{i \in I. emeasure (measure-pmf (map-pmf (\lambda f. f i) (Pi-pmf I dflt p))) (A i))$ 
by (intro prod.cong refl, subst Pi-pmf-component) (auto simp: fin)
finally show ?case
by (simp add: map-pmf-rep-eq)
qed fact+
qed (simp-all add: measure-pmf.prob-space-axioms)
qed

```

lemma

```

fixes  $h :: 'a :: comm-monoid-add \Rightarrow 'b :: \{banach, second-countable-topology\}$ 
assumes fin: finite I
assumes integrable:  $\bigwedge i. i \in I \implies integrable (measure-pmf (D i)) h$ 
shows integrable-sum-Pi-pmf:  $integrable (Pi-pmf I dflt D) (\lambda g. \sum_{i \in I. h (g i))$ 
and expectation-sum-Pi-pmf:
   $measure-pmf.expectation (Pi-pmf I dflt D) (\lambda g. \sum_{i \in I. h (g i)) =$ 
   $(\sum_{i \in I. measure-pmf.expectation (D i) h)$ 
proof –
have integrable':  $integrable (Pi-pmf I dflt D) (\lambda g. h (g i))$  if  $i: i \in I$  for  $i$ 
proof –
have integrable (D i) h
using i by (rule assms)
also have  $D i = map-pmf (\lambda g. g i) (Pi-pmf I dflt D)$ 
by (subst Pi-pmf-component) (use fin i in auto)
finally show  $integrable (measure-pmf (Pi-pmf I dflt D)) (\lambda x. h (x i))$ 
by simp
qed
thus  $integrable (Pi-pmf I dflt D) (\lambda g. \sum_{i \in I. h (g i))$ 
by (intro Bochner-Integration.integrable-sum)

have  $measure-pmf.expectation (Pi-pmf I dflt D) (\lambda x. \sum_{i \in I. h (x i)) =$ 
   $(\sum_{i \in I. measure-pmf.expectation (map-pmf (\lambda x. x i) (Pi-pmf I dflt D)) h)$ 
using integrable' by (subst Bochner-Integration.integral-sum) auto
also have  $\dots = (\sum_{i \in I. measure-pmf.expectation (D i) h)$ 
by (intro sum.cong refl, subst Pi-pmf-component) (use fin in auto)

```

finally show $\text{measure-pmf.expectation } (\text{Pi-pmf } I \text{ dflt } D) (\lambda g. \sum_{i \in I}. h (g \ i)) =$
 $(\sum_{i \in I}. \text{measure-pmf.expectation } (D \ i) h) .$
qed

26.7 Applications

Choosing a subset of a set uniformly at random is equivalent to tossing a fair coin independently for each element and collecting all the elements that came up heads.

lemma *pmf-of-set-Pow-conv-bernoulli*:
assumes *finite* ($A :: 'a \text{ set}$)
shows $\text{map-pmf } (\lambda b. \{x \in A. b \ x\}) (\text{Pi-pmf } A \ P \ (\lambda-. \text{bernoulli-pmf } (1/2))) =$
 $\text{pmf-of-set } (\text{Pow } A)$
proof –
have $\text{Pi-pmf } A \ P \ (\lambda-. \text{bernoulli-pmf } (1/2)) = \text{pmf-of-set } (\text{PiE-dflt } A \ P \ (\lambda x. \text{UNIV}))$
using *assms* **by** (*simp add: bernoulli-pmf-half-conv-pmf-of-set Pi-pmf-of-set*)
also have $\text{map-pmf } (\lambda b. \{x \in A. b \ x\}) \dots = \text{pmf-of-set } (\text{Pow } A)$
proof –
have $\text{bij-betw } (\lambda b. \{x \in A. b \ x\}) (\text{PiE-dflt } A \ P \ (\lambda-. \text{UNIV})) (\text{Pow } A)$
by (*rule bij-betwI[of - - $\lambda B \ b. \text{if } b \in A \text{ then } b \in B \text{ else } P$] (auto simp add: PiE-dflt-def)*)
then show *?thesis*
using *assms* **by** (*intro map-pmf-of-set-bij-betw*) *auto*
qed
finally show *?thesis*
by *simp*
qed

A binomial distribution can be seen as the number of successes in n independent coin tosses.

lemma *binomial-pmf-altdef'*:
fixes $A :: 'a \text{ set}$
assumes *finite* A **and** $\text{card } A = n$ **and** $p: p \in \{0..1\}$
shows $\text{binomial-pmf } n \ p =$
 $\text{map-pmf } (\lambda f. \text{card } \{x \in A. f \ x\}) (\text{Pi-pmf } A \ \text{dflt } (\lambda-. \text{bernoulli-pmf } p))$ (*is*
?lhs = ?rhs)
proof –
from *assms* **have** $?lhs = \text{binomial-pmf } (\text{card } A) \ p$
by *simp*
also have $\dots = ?rhs$
using *assms(1)*
proof (*induction rule: finite-induct*)
case *empty*
with p **show** *?case* **by** (*simp add: binomial-pmf-0*)
next
case (*insert* $x \ A$)
from *insert.hyps* **have** $\text{card } (\text{insert } x \ A) = \text{Suc } (\text{card } A)$

```

    by simp
  also have binomial-pmf ... p = do {
    b ← bernoulli-pmf p;
    f ← Pi-pmf A dflt (λ-. bernoulli-pmf p);
    return-pmf ((if b then 1 else 0) + card {y ∈ A. f y})
  }
  using p by (simp add: binomial-pmf-Suc insert.IH bind-map-pmf)
  also have ... = do {
    b ← bernoulli-pmf p;
    f ← Pi-pmf A dflt (λ-. bernoulli-pmf p);
    return-pmf (card {y ∈ insert x A. (f(x := b)) y})
  }
  proof (intro bind-pmf-cong refl, goal-cases)
    case (1 b f)
    have (if b then 1 else 0) + card {y ∈ A. f y} = card ((if b then {x} else {}) ∪
    {y ∈ A. f y})
      using insert.hyps by auto
    also have (if b then {x} else {}) ∪ {y ∈ A. f y} = {y ∈ insert x A. (f(x := b))
    y}
      using insert.hyps by auto
    finally show ?case by simp
  qed
  also have ... = map-pmf (λf. card {y ∈ insert x A. f y})
    (Pi-pmf (insert x A) dflt (λ-. bernoulli-pmf p))
    using insert.hyps by (subst Pi-pmf-insert) (simp-all add: pair-pmf-def map-bind-pmf)
  finally show ?case .
  qed
  finally show ?thesis .
  qed
end

```

27 Hoeffding’s Lemma and Hoeffding’s Inequality

```

theory Hoeffding
  imports Product-PMF Independent-Family
begin

```

Hoeffding’s inequality shows that a sum of bounded independent random variables is concentrated around its mean, with an exponential decay of the tail probabilities.

27.1 Hoeffding’s Lemma

```

lemma convex-on-exp:
  fixes l :: real
  assumes l ≥ 0
  shows convex-on UNIV (λx. exp(l*x))

```



```

using assms
by (intro convex-on-realI [where  $f' = \lambda x. l * \exp (l * x)$ ])
    (auto intro!: derivative-eq-intros mult-left-mono)

lemma mult-const-minus-self-real-le:
  fixes  $x :: \text{real}$ 
  shows  $x * (c - x) \leq c^2 / 4$ 
proof -
  have  $x * (c - x) = -(x - c / 2)^2 + c^2 / 4$ 
    by (simp add: field-simps power2-eq-square)
  also have  $\dots \leq 0 + c^2 / 4$ 
    by (intro add-mono) auto
  finally show ?thesis by simp
qed

lemma Hoeffdings-lemma-aux:
  fixes  $h\ p :: \text{real}$ 
  assumes  $h \geq 0$  and  $p \geq 0$ 
  defines  $L \equiv (\lambda h. -h * p + \ln (1 + p * (\exp h - 1)))$ 
  shows  $L\ h \leq h^2 / 8$ 
proof (cases  $h = 0$ )
  case False
  hence  $h: h > 0$ 
    using  $\langle h \geq 0 \rangle$  by simp
  define  $L'$  where  $L' = (\lambda h. -p + p * \exp h / (1 + p * (\exp h - 1)))$ 
  define  $L''$  where  $L'' = (\lambda h. -(p^2) * \exp h * \exp h / (1 + p * (\exp h - 1))^2 +$ 
     $p * \exp h / (1 + p * (\exp h - 1)))$ 
  define  $L_s$  where  $L_s = (\lambda n. [L, L', L''] ! n)$ 

  have [simp]:  $L\ 0 = 0\ L'\ 0 = 0$ 
    by (auto simp: L-def L'-def)

  have  $L'$ : ( $L$  has-real-derivative  $L'\ x$ ) (at  $x$ ) if  $x \in \{0..h\}$  for  $x$ 
  proof -
    have  $1 + p * (\exp x - 1) > 0$ 
      using  $\langle p \geq 0 \rangle$  that by (intro add-pos-nonneg mult-nonneg-nonneg) auto
    thus ?thesis
    unfolding  $L$ -def  $L'$ -def by (auto intro!: derivative-eq-intros)
  qed

  have  $L''$ : ( $L'$  has-real-derivative  $L''\ x$ ) (at  $x$ ) if  $x \in \{0..h\}$  for  $x$ 
  proof -
    have  $1 + p * (\exp x - 1) > 0$ 
      using  $\langle p \geq 0 \rangle$  that by (intro add-pos-nonneg mult-nonneg-nonneg) auto
    show ?thesis
    unfolding  $L'$ -def  $L''$ -def
      by (insert  $*$ , (rule derivative-eq-intros refl | simp)+) (auto simp: divide-simps;
algebra)
  qed

```

```

have diff:  $\forall m\ t. m < 2 \wedge 0 \leq t \wedge t \leq h \longrightarrow (Ls\ m\ \text{has-real-derivative}\ Ls\ (Suc\ m)\ t)\ (at\ t)$ 
using L' L'' by (auto simp: Ls-def nth-Cons split: nat.splits)
from Taylor[of 2 Ls L 0 h 0 h, OF - - diff]
obtain t where t:  $t \in \{0 < .. < h\}$   $L\ h = L''\ t * h^2 / 2$ 
using  $\langle h > 0 \rangle$  by (auto simp: Ls-def lessThan-nat-numeral)
define u where  $u = p * \exp\ t / (1 + p * (\exp\ t - 1))$ 

have L'' t =  $u * (1 - u)$ 
by (simp add: L''-def u-def divide-simps; algebra)
also have  $\dots \leq 1 / 4$ 
using mult-const-minus-self-real-le[of u 1] by simp
finally have  $L''\ t \leq 1 / 4$  .

note t(2)
also have  $L''\ t * h^2 / 2 \leq (1 / 4) * h^2 / 2$ 
using  $\langle L''\ t \leq 1 / 4 \rangle$  by (intro mult-right-mono divide-right-mono) auto
finally show  $L\ h \leq h^2 / 8$  by simp
qed (auto simp: L-def)

```

```

locale interval-bounded-random-variable = prob-space +
  fixes f :: 'a  $\Rightarrow$  real and a b :: real
  assumes random-variable [measurable]: random-variable borel f
  assumes AE-in-interval: AE x in M. f x  $\in \{a..b\}$ 
begin

```

```

lemma integrable [intro]: integrable M f
proof (rule integrable-const-bound)
  show AE x in M. norm (f x)  $\leq \max |a| |b|$ 
  by (intro eventually-mono[OF AE-in-interval]) auto
qed (fact random-variable)

```

We first show Hoeffding’s lemma for distributions whose expectation is 0. The general case will easily follow from this later.

```

lemma Hoeffdings-lemma-nn-integral-0:
  assumes l > 0 and E0: expectation f = 0
  shows nn-integral M ( $\lambda x. \exp\ (l * f\ x)$ )  $\leq \text{ennreal}\ (\exp\ (l^2 * (b - a)^2 / 8))$ 
proof (cases AE x in M. f x = 0)
  case True
  hence nn-integral M ( $\lambda x. \exp\ (l * f\ x)$ ) = nn-integral M ( $\lambda x. \text{ennreal}\ 1$ )
  by (intro nn-integral-cong-AE) auto
  also have  $\dots = \text{ennreal}\ (\text{expectation}\ (\lambda -. 1))$ 
  by (intro nn-integral-eq-integral) auto
  finally show ?thesis by (simp add: prob-space)
next
  case False
  have a < 0

```

```

proof (rule ccontr)
  assume a:  $\neg(a < 0)$ 
  have AE x in M.  $f x = 0$ 
  proof (subst integral-nonneg-eq-0-iff-AE [symmetric])
    show AE x in M.  $f x \geq 0$ 
    using AE-in-interval by eventually-elim (use a in auto)
  qed (use E0 in ⟨auto simp: id-def integrable⟩)
  with False show False by contradiction
qed

have b > 0
proof (rule ccontr)
  assume b:  $\neg(b > 0)$ 
  have AE x in M.  $-f x = 0$ 
  proof (subst integral-nonneg-eq-0-iff-AE [symmetric])
    show AE x in M.  $-f x \geq 0$ 
    using AE-in-interval by eventually-elim (use b in auto)
  qed (use E0 in ⟨auto simp: id-def integrable⟩)
  with False show False by simp
qed

have a < b
  using ⟨a < 0⟩ ⟨b > 0⟩ by linarith

define p where  $p = -a / (b - a)$ 
define L where  $L = (\lambda t. -t * p + \ln (1 - p + p * \exp t))$ 
define z where  $z = l * (b - a)$ 
have z > 0
  unfolding z-def using ⟨a < b⟩ ⟨l > 0⟩ by auto
have p > 0
  using ⟨a < 0⟩ ⟨a < b⟩ unfolding p-def by (intro divide-pos-pos) auto

have  $(\int^{+x}. \exp (l * f x) \partial M) \leq$ 
   $(\int^{+x}. (b - f x) / (b - a) * \exp (l * a) + (f x - a) / (b - a) * \exp (l * b)$ 
 $\partial M)$ 
proof (intro nn-integral-mono-AE eventually-mono[OF AE-in-interval] ennreal-leI)
  fix x assume x:  $f x \in \{a..b\}$ 
  define y where  $y = (b - f x) / (b - a)$ 
  have y:  $y \in \{0..1\}$ 
  using x ⟨a < b⟩ by (auto simp: y-def)
  have conv: convex-on UNIV  $(\lambda x. \exp(l*x))$ 
  using ⟨l > 0⟩ by (intro convex-on-exp) auto
  have  $\exp (l * ((1 - y) *_{\mathbb{R}} b + y *_{\mathbb{R}} a)) \leq (1 - y) * \exp (l * b) + y * \exp (l$ 
   $* a)$ 
  using y ⟨l > 0⟩ by (intro convex-onD[OF convex-on-exp]) auto
  also have  $(1 - y) *_{\mathbb{R}} b + y *_{\mathbb{R}} a = f x$ 
  using ⟨a < b⟩ by (simp add: y-def divide-simps) (simp add: algebra-simps)?
  also have  $1 - y = (f x - a) / (b - a)$ 
  using ⟨a < b⟩ by (simp add: field-simps y-def)

```

finally show $\exp(l * f x) \leq (b - f x) / (b - a) * \exp(l * a) + (f x - a) / (b - a) * \exp(l * b)$
by (*simp add: y-def*)
qed
also have $\dots = (\int^+ x. \text{ennreal } (b - f x) * \exp(l * a) / (b - a) + \text{ennreal } (f x - a) * \exp(l * b) / (b - a) \partial M)$
using $\langle a < 0 \rangle \langle b > 0 \rangle$
by (*intro nn-integral-cong-AE eventually-mono[OF AE-in-interval]*)
(simp add: ennreal-plus ennreal-mult flip: divide-ennreal)
also have $\dots = ((\int^+ x. \text{ennreal } (b - f x) \partial M) * \text{ennreal } (\exp(l * a)) + (\int^+ x. \text{ennreal } (f x - a) \partial M) * \text{ennreal } (\exp(l * b))) / \text{ennreal } (b - a)$
by (*simp add: nn-integral-add nn-integral-divide nn-integral-multc add-divide-distrib-ennreal*)
also have $(\int^+ x. \text{ennreal } (b - f x) \partial M) = \text{ennreal } (\text{expectation } (\lambda x. b - f x))$
by (*intro nn-integral-eq-integral Bochner-Integration.integrable-diff eventually-mono[OF AE-in-interval] integrable-const integrable*) *auto*
also have $\text{expectation } (\lambda x. b - f x) = b$
using *assms by* (*subst Bochner-Integration.integral-diff*) (*auto simp: prob-space*)
also have $(\int^+ x. \text{ennreal } (f x - a) \partial M) = \text{ennreal } (\text{expectation } (\lambda x. f x - a))$
by (*intro nn-integral-eq-integral Bochner-Integration.integrable-diff eventually-mono[OF AE-in-interval] integrable-const integrable*) *auto*
also have $\text{expectation } (\lambda x. f x - a) = (-a)$
using *assms by* (*subst Bochner-Integration.integral-diff*) (*auto simp: prob-space*)
also have $(\text{ennreal } b * (\exp(l * a)) + \text{ennreal } (-a) * (\exp(l * b))) / (b - a) = \text{ennreal } (b * \exp(l * a) - a * \exp(l * b)) / \text{ennreal } (b - a)$
using $\langle a < 0 \rangle \langle b > 0 \rangle$
by (*simp flip: ennreal-mult ennreal-plus add: mult-nonpos-nonneg divide-ennreal mult-mono*)
also have $b * \exp(l * a) - a * \exp(l * b) = \exp(L z) * (b - a)$
proof –
have *pos: 1 - p + p * exp z > 0*
proof –
have $\exp z > 1$ **using** $\langle l > 0 \rangle$ **and** $\langle a < b \rangle$
by (*subst one-less-exp-iff*) (*auto simp: z-def intro!: mult-pos-pos*)
hence $(\exp z - 1) * p \geq 0$
unfolding *p-def* **using** $\langle a < 0 \rangle$ **and** $\langle a < b \rangle$
by (*intro mult-nonneg-nonneg divide-nonneg-pos*) *auto*
thus *?thesis*
by (*simp add: algebra-simps*)
qed
have $\exp(L z) * (b - a) = \exp(-z * p) * (1 - p + p * \exp z) * (b - a)$
using *pos by* (*simp add: exp-add L-def exp-diff exp-minus divide-simps*)
also have $\dots = b * \exp(l * a) - a * \exp(l * b)$ **using** $\langle a < b \rangle$
by (*simp add: p-def z-def divide-simps*) (*simp add: exp-diff algebra-simps*)?
finally show *?thesis by simp*
qed
also have $\text{ennreal } (\exp(L z) * (b - a)) / \text{ennreal } (b - a) = \text{ennreal } (\exp(L z))$
using $\langle a < b \rangle$ **by** (*simp add: divide-ennreal*)

```

also have  $L\ z = -z * p + \ln (1 + p * (\exp z - 1))$ 
  by (simp add: L-def algebra-simps)
also have  $\dots \leq z^2 / 8$ 
  unfolding L-def by (rule Hoeffdings-lemma-aux[where  $p = p$ ]) (use  $\langle z > 0 \rangle$ 
 $\langle p > 0 \rangle$  in simp-all)
  hence  $\text{ennreal } (\exp (-z * p + \ln (1 + p * (\exp z - 1)))) \leq \text{ennreal } (\exp (z^2 / 8))$ 
  by (intro ennreal-leI) auto
finally show ?thesis
  by (simp add: z-def power-mult-distrib)
qed

```

```

context
begin

```

```

interpretation shift: interval-bounded-random-variable  $M\ \lambda x. f\ x - \mu\ a - \mu\ b - \mu$ 
rewrites  $b - \mu - (a - \mu) \equiv b - a$ 
by unfold-locales (auto intro!: eventually-mono[OF AE-in-interval])

```

```

lemma expectation-shift: expectation  $(\lambda x. f\ x - \text{expectation } f) = 0$ 
by (subst Bochner-Integration.integral-diff) (auto simp: integrable prob-space)

```

```

lemmas Hoeffdings-lemma-nn-integral = shift.Hoeffdings-lemma-nn-integral-0[OF
- expectation-shift]

```

```

end

```

```

end

```

27.2 Hoeffding’s Inequality

Consider n independent real random variables X_1, \dots, X_n that each almost surely lie in a compact interval $[a_i, b_i]$. Hoeffding’s inequality states that the distribution of the sum of the X_i is tightly concentrated around the sum of the expected values: the probability of it being above or below the sum of the expected values by more than some ε decreases exponentially with ε .

```

locale indep-interval-bounded-random-variables = prob-space +
  fixes  $I :: 'b\ \text{set}$  and  $X :: 'b \Rightarrow 'a \Rightarrow \text{real}$ 
  fixes  $a\ b :: 'b \Rightarrow \text{real}$ 
  assumes fin: finite I
  assumes indep: indep-vars  $(\lambda -. \text{borel})\ X\ I$ 
  assumes AE-in-interval:  $\bigwedge i. i \in I \implies \text{AE } x \text{ in } M. X\ i\ x \in \{a\ i..b\ i\}$ 
begin

```

```

lemma random-variable [measurable]:
  assumes  $i: i \in I$ 
  shows random-variable borel  $(X\ i)$ 

```

```

using i indep unfolding indep-vars-def by blast

lemma bounded-random-variable [intro]:
  assumes i: i ∈ I
  shows interval-bounded-random-variable M (X i) (a i) (b i)
  by unfold-locales (use AE-in-interval[OF i] i in auto)

end

locale Hoeffding-ineq = indep-interval-bounded-random-variables +
  fixes μ :: real
  defines μ ≡ (∑ i∈I. expectation (X i))
begin

theorem Hoeffding-ineq-ge:
  assumes ε ≥ 0
  assumes (∑ i∈I. (b i - a i)2) > 0
  shows prob {x∈space M. (∑ i∈I. X i x) ≥ μ + ε} ≤ exp (-2 * ε2 / (∑ i∈I. (b i - a i)2))
proof (cases ε = 0)
  case [simp]: True
  have prob {x∈space M. (∑ i∈I. X i x) ≥ μ + ε} ≤ 1
  by simp
  thus ?thesis by simp
next
  case False
  with ⟨ε ≥ 0⟩ have ε: ε > 0
  by auto

  define d where d = (∑ i∈I. (b i - a i)2)
  define l :: real where l = 4 * ε / d
  have d: d > 0
  using assms by (simp add: d-def)
  have l: l > 0
  using ε d by (simp add: l-def)
  define μ' where μ' = (λi. expectation (X i))

  have {x∈space M. (∑ i∈I. X i x) ≥ μ + ε} = {x∈space M. (∑ i∈I. X i x) - μ ≥ ε}
  by (simp add: algebra-simps)
  hence ennreal (prob {x∈space M. (∑ i∈I. X i x) ≥ μ + ε}) = emeasure M ...
  by (simp add: emeasure-eq-measure)
  also have ... ≤ ennreal (exp (-l*ε)) * (∫+x∈space M. exp (l * ((∑ i∈I. X i x) - μ)) ∂M)
  by (intro Chernoff-ineq-nn-integral-ge l) auto
  also have (λx. (∑ i∈I. X i x) - μ) = (λx. (∑ i∈I. X i x - μ' i))
  by (simp add: μ-def sum-subtractf μ'-def)
  also have (∫+x∈space M. exp (l * ((∑ i∈I. X i x - μ' i))) ∂M) =

```

$(\int^+ x. (\prod_{i \in I}. \text{ennreal } (\exp (l * (X \ i \ x - \mu' \ i)))) \ \partial M)$
by (*intro nn-integral-cong*)
 $(\text{simp-all add: sum-distrib-left ring-distrib exp-diff exp-sum fin prod-ennreal})$
also have $\dots = (\prod_{i \in I}. \int^+ x. \text{ennreal } (\exp (l * (X \ i \ x - \mu' \ i)))) \ \partial M$
by (*intro indep-vars-nn-integral fin indep-vars-compose2[OF indep]*) *auto*
also have $\text{ennreal } (\exp (-l * \varepsilon)) * \dots \leq$
 $\text{ennreal } (\exp (-l * \varepsilon)) * (\prod_{i \in I}. \text{ennreal } (\exp (l^2 * (b \ i - a \ i)^2 / 8)))$
proof (*intro mult-left-mono prod-mono-ennreal*)
fix i **assume** $i: i \in I$
from i **interpret** *interval-bounded-random-variable* $M \ X \ i \ a \ i \ b \ i \ ..$
show $(\int^+ x. \text{ennreal } (\exp (l * (X \ i \ x - \mu' \ i)))) \ \partial M \leq \text{ennreal } (\exp (l^2 * (b \ i - a \ i)^2 / 8))$
unfolding μ' -def **by** (*rule Hoeffdings-lemma-nn-integral*) *fact+*
qed *auto*
also have $\dots = \text{ennreal } (\exp (-l * \varepsilon)) * (\prod_{i \in I}. \exp (l^2 * (b \ i - a \ i)^2 / 8))$
by (*simp add: prod-ennreal prod-nonneg flip: ennreal-mult*)
also have $\exp (-l * \varepsilon) * (\prod_{i \in I}. \exp (l^2 * (b \ i - a \ i)^2 / 8)) = \exp (d * l^2 / 8 - l * \varepsilon)$
by (*simp add: exp-diff exp-minus sum-divide-distrib sum-distrib-left sum-distrib-right exp-sum fin divide-simps mult-ac d-def*)
also have $d * l^2 / 8 - l * \varepsilon = -2 * \varepsilon^2 / d$
using d **by** (*simp add: l-def field-simps power2-eq-square*)
finally show *?thesis*
by (*subst (asm) ennreal-le-iff*) (*simp-all add: d-def*)
qed

corollary *Hoeffding-ineq-le:*

assumes $\varepsilon: \varepsilon \geq 0$
assumes $(\sum_{i \in I}. (b \ i - a \ i)^2) > 0$
shows $\text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \leq \mu - \varepsilon\} \leq \exp (-2 * \varepsilon^2 / (\sum_{i \in I}. (b \ i - a \ i)^2))$
proof –
interpret *flip: Hoeffding-ineq* $M \ I \ \lambda i \ x. -X \ i \ x \ \lambda i. -b \ i \ \lambda i. -a \ i - \mu$
proof *unfold-locales*
fix i **assume** $i \in I$
then interpret *interval-bounded-random-variable* $M \ X \ i \ a \ i \ b \ i \ ..$
show $AE \ x \ \text{in } M. -X \ i \ x \in \{-b \ i .. -a \ i\}$
by (*intro eventually-mono[OF AE-in-interval]*) *auto*
qed (*auto simp: fin μ -def sum-negf intro: indep-vars-compose2[OF indep]*)

have $\text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \leq \mu - \varepsilon\} = \text{prob } \{x \in \text{space } M. (\sum_{i \in I}. -X \ i \ x) \geq -\mu + \varepsilon\}$
by (*simp add: sum-negf algebra-simps*)
also have $\dots \leq \exp (-2 * \varepsilon^2 / (\sum_{i \in I}. (b \ i - a \ i)^2))$
using *flip.Hoeffding-ineq-ge*[*OF* ε] *assms*(2) **by** *simp*
finally show *?thesis* .
qed

corollary *Hoeffding-ineq-abs-ge:*

```

assumes  $\varepsilon: \varepsilon \geq 0$ 
assumes  $(\sum_{i \in I}. (b \ i - a \ i)^2) > 0$ 
shows  $\text{prob } \{x \in \text{space } M. |(\sum_{i \in I}. X \ i \ x) - \mu| \geq \varepsilon\} \leq 2 * \exp (-2 * \varepsilon^2 /$ 
 $(\sum_{i \in I}. (b \ i - a \ i)^2))$ 
proof -
  have  $\{x \in \text{space } M. |(\sum_{i \in I}. X \ i \ x) - \mu| \geq \varepsilon\} =$ 
 $\{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \geq \mu + \varepsilon\} \cup \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \leq \mu$ 
 $- \varepsilon\}$ 
  by auto
  also have  $\text{prob } \dots \leq \text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \geq \mu + \varepsilon\} +$ 
 $\text{prob } \{x \in \text{space } M. (\sum_{i \in I}. X \ i \ x) \leq \mu - \varepsilon\}$ 
  by (intro measure-Un-le) auto
  also have  $\dots \leq \exp (-2 * \varepsilon^2 / (\sum_{i \in I}. (b \ i - a \ i)^2)) + \exp (-2 * \varepsilon^2 / (\sum_{i \in I}. (b \ i - a \ i)^2))$ 
  by (intro add-mono Hoeffding-ineq-ge Hoeffding-ineq-le assms)
  finally show ?thesis by simp
qed

end

```

27.3 Hoeffding’s inequality for i.i.d. bounded random variables

If we have n even identically-distributed random variables, the statement of Hoeffding’s lemma simplifies a bit more: it shows that the probability that the average of the X_i is more than ε above the expected value is no greater than $e^{\frac{-2n\varepsilon^2}{(b-a)^2}}$.

This essentially gives us a more concrete version of the weak law of large numbers: the law states that the probability vanishes for $n \rightarrow \infty$ for any $\varepsilon > 0$. Unlike Hoeffding’s inequality, it does not assume the variables to have bounded support, but it does not provide concrete bounds.

```

locale iid-interval-bounded-random-variables = prob-space +
  fixes  $I :: 'b \text{ set}$  and  $X :: 'b \Rightarrow 'a \Rightarrow \text{real}$  and  $Y :: 'a \Rightarrow \text{real}$ 
  fixes  $a \ b :: \text{real}$ 
  assumes fin: finite I
  assumes indep: indep-vars ( $\lambda \cdot$ . borel) X I
  assumes distr-X:  $i \in I \implies \text{distr } M \text{ borel } (X \ i) = \text{distr } M \text{ borel } Y$ 
  assumes rv-Y [measurable]: random-variable borel Y
  assumes AE-in-interval:  $AE \ x \text{ in } M. Y \ x \in \{a..b\}$ 
begin

lemma random-variable [measurable]:
  assumes  $i: i \in I$ 
  shows random-variable borel (X i)
  using  $i \text{ indep}$  unfolding indep-vars-def by blast

sublocale  $X: \text{indep-interval-bounded-random-variables } M \ I \ X \ \lambda \cdot. a \ \lambda \cdot. b$ 

```



```

proof
  fix  $i$  assume  $i: i \in I$ 
  have  $AE\ x\ in\ M.\ Y\ x \in \{a..b\}$ 
    by (fact AE-in-interval)
  also have  $?this \longleftrightarrow (AE\ x\ in\ distr\ M\ borel\ Y.\ x \in \{a..b\})$ 
    by (subst AE-distr-iff) auto
  also have  $distr\ M\ borel\ Y = distr\ M\ borel\ (X\ i)$ 
    using  $i$  by (simp add: distr-X)
  also have  $(AE\ x\ in\ \dots\ x \in \{a..b\}) \longleftrightarrow (AE\ x\ in\ M.\ X\ i\ x \in \{a..b\})$ 
    using  $i$  by (subst AE-distr-iff) auto
  finally show  $AE\ x\ in\ M.\ X\ i\ x \in \{a..b\}$  .
qed (simp-all add: fn indep)

lemma expectation-X [simp]:
  assumes  $i: i \in I$ 
  shows  $expectation\ (X\ i) = expectation\ Y$ 
proof –
  have  $expectation\ (X\ i) = lebesgue-integral\ (distr\ M\ borel\ (X\ i))\ (\lambda x.\ x)$ 
    using  $i$  by (intro integral-distr [symmetric]) auto
  also have  $distr\ M\ borel\ (X\ i) = distr\ M\ borel\ Y$ 
    using  $i$  by (rule distr-X)
  also have  $lebesgue-integral\ \dots\ (\lambda x.\ x) = expectation\ Y$ 
    by (rule integral-distr) auto
  finally show  $expectation\ (X\ i) = expectation\ Y$  .
qed

end

```

```

locale Hoeffding-ineq-iid = iid-interval-bounded-random-variables +
  fixes  $\mu :: real$ 
  defines  $\mu \equiv expectation\ Y$ 
begin

```

```

sublocale  $X: Hoeffding-ineq\ M\ I\ X\ \lambda\cdot.\ a\ \lambda\cdot.\ b\ real\ (card\ I) * \mu$ 
  by unfold-locales (simp-all add:  $\mu$ -def)

```

```

corollary
  assumes  $\varepsilon: \varepsilon \geq 0$ 
  assumes  $a < b\ I \neq \{\}$ 
  defines  $n \equiv card\ I$ 
  shows Hoeffding-ineq-ge:
     $prob\ \{x \in space\ M.\ (\sum_{i \in I} X\ i\ x) \geq n * \mu + \varepsilon\} \leq$ 
       $exp\ (-2 * \varepsilon^2 / (n * (b - a)^2))\ (\mathbf{is}\ ?le)$ 
  and Hoeffding-ineq-le:
     $prob\ \{x \in space\ M.\ (\sum_{i \in I} X\ i\ x) \leq n * \mu - \varepsilon\} \leq$ 
       $exp\ (-2 * \varepsilon^2 / (n * (b - a)^2))\ (\mathbf{is}\ ?ge)$ 
  and Hoeffding-ineq-abs-ge:
     $prob\ \{x \in space\ M.\ |(\sum_{i \in I} X\ i\ x) - n * \mu| \geq \varepsilon\} \leq$ 

```

```

      2 * exp (-2 * ε2 / (n * (b - a)2)) (is ?abs-ge)
proof -
  have pos: (∑ i∈I. (b - a)2) > 0
    using ‹a < b› ‹I ≠ {}› fin by (intro sum-pos) auto
  show ?le
    using X.Hoeffding-ineq-ge[OF ε pos] by (simp add: n-def)
  show ?ge
    using X.Hoeffding-ineq-le[OF ε pos] by (simp add: n-def)
  show ?abs-ge
    using X.Hoeffding-ineq-abs-ge[OF ε pos] by (simp add: n-def)
qed

lemma
  assumes ε: ε ≥ 0
  assumes a < b I ≠ {}
  defines n ≡ card I
  shows Hoeffding-ineq-ge':
    prob {x∈space M. (∑ i∈I. X i x) / n ≥ μ + ε} ≤
      exp (-2 * n * ε2 / (b - a)2) (is ?ge)
  and Hoeffding-ineq-le':
    prob {x∈space M. (∑ i∈I. X i x) / n ≤ μ - ε} ≤
      exp (-2 * n * ε2 / (b - a)2) (is ?le)
  and Hoeffding-ineq-abs-ge':
    prob {x∈space M. |(∑ i∈I. X i x) / n - μ| ≥ ε} ≤
      2 * exp (-2 * n * ε2 / (b - a)2) (is ?abs-ge)

proof -
  have n > 0
    using assms fin by (auto simp: field-simps)
  have ε': ε * n ≥ 0
    using ‹n > 0› ‹ε ≥ 0› by auto
  have eq: - (2 * (ε * real n)2 / (real (card I) * (b - a)2)) =
    - (2 * real n * ε2 / (b - a)2)
    using ‹n > 0› by (simp add: power2-eq-square divide-simps n-def)

  have {x∈space M. (∑ i∈I. X i x) / n ≥ μ + ε} =
    {x∈space M. (∑ i∈I. X i x) ≥ μ * n + ε * n}
    using ‹n > 0› by (intro Collect-cong conj-cong refl) (auto simp: field-simps)
  with Hoeffding-ineq-ge[OF ε' ‹a < b› ‹I ≠ {}›] ‹n > 0› eq show ?ge
    by (simp add: n-def mult-ac)

  have {x∈space M. (∑ i∈I. X i x) / n ≤ μ - ε} =
    {x∈space M. (∑ i∈I. X i x) ≤ μ * n - ε * n}
    using ‹n > 0› by (intro Collect-cong conj-cong refl) (auto simp: field-simps)
  with Hoeffding-ineq-le[OF ε' ‹a < b› ‹I ≠ {}›] ‹n > 0› eq show ?le
    by (simp add: n-def mult-ac)

  have {x∈space M. |(∑ i∈I. X i x) / n - μ| ≥ ε} =
    {x∈space M. |(∑ i∈I. X i x) - μ * n| ≥ ε * n}
    using ‹n > 0› by (intro Collect-cong conj-cong refl) (auto simp: field-simps)

```

```

with Hoeffding-ineq-abs-ge[OF  $\varepsilon' \langle a < b \rangle \langle I \neq \{\} \rangle \langle n > 0 \rangle$  eq show ?abs-ge
  by (simp add: n-def mult-ac)
qed

end

```

27.4 Hoeffding’s Inequality for the Binomial distribution

We can now apply Hoeffding’s inequality to the Binomial distribution, which can be seen as the sum of n i.i.d. coin flips (the support of each of which is contained in $[0, 1]$).

```

locale binomial-distribution =
  fixes  $n :: \text{nat}$  and  $p :: \text{real}$ 
  assumes  $p: p \in \{0..1\}$ 
begin

```

```

context
  fixes  $\text{coins} :: (\text{nat} \Rightarrow \text{bool}) \text{ pmf}$  and  $\mu$ 
  assumes  $n: n > 0$ 
  defines  $\text{coins} \equiv \text{Pi-pmf } \{..<n\} \text{ False } (\lambda-. \text{bernoulli-pmf } p)$ 
begin

```

```

lemma coins-component:
  assumes  $i: i < n$ 
  shows  $\text{distr } (\text{measure-pmf coins}) \text{ borel } (\lambda f. \text{if } f \ i \ \text{then } 1 \ \text{else } 0) =$ 
     $\text{distr } (\text{measure-pmf } (\text{bernoulli-pmf } p)) \text{ borel } (\lambda b. \text{if } b \ \text{then } 1 \ \text{else } 0)$ 
proof –
  have  $\text{distr } (\text{measure-pmf coins}) \text{ borel } (\lambda f. \text{if } f \ i \ \text{then } 1 \ \text{else } 0) =$ 
     $\text{distr } (\text{measure-pmf } (\text{map-pmf } (\lambda f. f \ i) \ \text{coins})) \text{ borel } (\lambda b. \text{if } b \ \text{then } 1 \ \text{else } 0)$ 
  unfolding map-pmf-rep-eq by (subst distr-distr) (auto simp: o-def)
  also have  $\text{map-pmf } (\lambda f. f \ i) \ \text{coins} = \text{bernoulli-pmf } p$ 
  unfolding coins-def using  $i$  by (subst Pi-pmf-component) auto
  finally show ?thesis
  unfolding map-pmf-rep-eq .
qed

```

```

lemma prob-binomial-pmf-conv-coins:
   $\text{measure-pmf.prob } (\text{binomial-pmf } n \ p) \ \{x. P \ (\text{real } x)\} =$ 
     $\text{measure-pmf.prob coins } \{x. P \ (\sum i < n. \text{if } x \ i \ \text{then } 1 \ \text{else } 0)\}$ 
proof –
  have  $\text{eq1}: (\sum i < n. \text{if } x \ i \ \text{then } 1 \ \text{else } 0) = \text{real } (\text{card } \{i \in \{..<n\}. x \ i\})$  for  $x$ 
  proof –
    have  $(\sum i < n. \text{if } x \ i \ \text{then } 1 \ \text{else } (0::\text{real})) = (\sum i \in \{i \in \{..<n\}. x \ i\}. 1)$ 
    by (intro sum.mono-neutral-cong-right) auto
    thus ?thesis by simp
  qed
  have  $\text{eq2}: \text{binomial-pmf } n \ p = \text{map-pmf } (\lambda v. \text{card } \{i \in \{..<n\}. v \ i\}) \ \text{coins}$ 
  unfolding coins-def by (rule binomial-pmf-altdef') (use p in auto)
  show ?thesis

```

by (subst eq2) (simp-all add: eq1)
qed

interpretation *Hoeffding-ineq-iid*

coins $\{..<n\}$ $\lambda i f.$ if $f i$ then 1 else 0 $\lambda f.$ if $f 0$ then 1 else 0 0 1 p

proof *unfold-locales*

show *prob-space.indep-vars* (measure-pmf coins) ($\lambda.$ borel) ($\lambda i f.$ if $f i$ then 1 else 0) $\{..<n\}$

unfolding coins-def

by (intro *prob-space.indep-vars-compose2*[OF - *indep-vars-Pi-pmf*])
(auto simp: *measure-pmf.prob-space-axioms*)

next

have *measure-pmf.expectation* coins ($\lambda f.$ if $f 0$ then 1 else 0 :: real) =
measure-pmf.expectation (map-pmf ($\lambda f.$ f 0) coins) ($\lambda b.$ if b then 1 else 0 :: real)

by (simp add: coins-def)

also have *map-pmf* ($\lambda f.$ f 0) coins = *bernoulli-pmf* p

using n by (simp add: coins-def *Pi-pmf-component*)

also have *measure-pmf.expectation* ... ($\lambda b.$ if b then 1 else 0) = p

using p by simp

finally show $p \equiv$ *measure-pmf.expectation* coins ($\lambda f.$ if $f 0$ then 1 else 0) by
simp

qed (auto simp: coins-component)

corollary

fixes $\varepsilon ::$ real

assumes $\varepsilon: \varepsilon \geq 0$

shows *prob-ge: measure-pmf.prob* (*binomial-pmf* $n p$) $\{x. x \geq n * p + \varepsilon\} \leq \exp$
 $(-2 * \varepsilon^2 / n)$

and *prob-le: measure-pmf.prob* (*binomial-pmf* $n p$) $\{x. x \leq n * p - \varepsilon\} \leq \exp$
 $(-2 * \varepsilon^2 / n)$

and *prob-abs-ge:*

measure-pmf.prob (*binomial-pmf* $n p$) $\{x. |x - n * p| \geq \varepsilon\} \leq 2 * \exp (-2$
 $* \varepsilon^2 / n)$

proof –

have [*simp*]: $\{..<n\} \neq \{\}$

using n by auto

show *measure-pmf.prob* (*binomial-pmf* $n p$) $\{x. x \geq n * p + \varepsilon\} \leq \exp (-2 * \varepsilon^2$
 $/ n)$

using *Hoeffding-ineq-ge*[of ε] by (subst *prob-binomial-pmf-conv-coins*) (use
assms in *simp-all*)

show *measure-pmf.prob* (*binomial-pmf* $n p$) $\{x. x \leq n * p - \varepsilon\} \leq \exp (-2 * \varepsilon^2$
 $/ n)$

using *Hoeffding-ineq-le*[of ε] by (subst *prob-binomial-pmf-conv-coins*) (use
assms in *simp-all*)

show *measure-pmf.prob* (*binomial-pmf* $n p$) $\{x. |x - n * p| \geq \varepsilon\} \leq 2 * \exp (-2$
 $* \varepsilon^2 / n)$

using *Hoeffding-ineq-abs-ge*[of ε]

by (subst *prob-binomial-pmf-conv-coins*) (use assms in *simp-all*)

qed

corollary

```

  fixes  $\varepsilon :: \text{real}$ 
  assumes  $\varepsilon: \varepsilon \geq 0$ 
  shows  $\text{prob-ge}'$ :  $\text{measure-pmf.prob (binomial-pmf } n \text{ } p) \{x. x / n \geq p + \varepsilon\} \leq \exp$ 
     $(-2 * n * \varepsilon^2)$ 
    and  $\text{prob-le}'$ :  $\text{measure-pmf.prob (binomial-pmf } n \text{ } p) \{x. x / n \leq p - \varepsilon\} \leq \exp$ 
     $(-2 * n * \varepsilon^2)$ 
    and  $\text{prob-abs-ge}'$ :
       $\text{measure-pmf.prob (binomial-pmf } n \text{ } p) \{x. |x / n - p| \geq \varepsilon\} \leq 2 * \exp (-2$ 
     $* n * \varepsilon^2)$ 
  proof -
    have  $[\text{simp}]: \{..<n\} \neq \{\}$ 
    using  $n$  by auto
    show  $\text{measure-pmf.prob (binomial-pmf } n \text{ } p) \{x. x / n \geq p + \varepsilon\} \leq \exp (-2 * n$ 
     $* \varepsilon^2)$ 
    using  $\text{Hoeffding-ineq-ge}'[\text{of } \varepsilon]$  by (subst  $\text{prob-binomial-pmf-conv-coins}$ ) (use
     $\text{assms in simp-all}$ )
    show  $\text{measure-pmf.prob (binomial-pmf } n \text{ } p) \{x. x / n \leq p - \varepsilon\} \leq \exp (-2 * n$ 
     $* \varepsilon^2)$ 
    using  $\text{Hoeffding-ineq-le}'[\text{of } \varepsilon]$  by (subst  $\text{prob-binomial-pmf-conv-coins}$ ) (use
     $\text{assms in simp-all}$ )
    show  $\text{measure-pmf.prob (binomial-pmf } n \text{ } p) \{x. |x / n - p| \geq \varepsilon\} \leq 2 * \exp (-2$ 
     $* n * \varepsilon^2)$ 
    using  $\text{Hoeffding-ineq-abs-ge}'[\text{of } \varepsilon]$ 
    by (subst  $\text{prob-binomial-pmf-conv-coins}$ ) (use  $\text{assms in simp-all}$ )
  qed

```

end

end

27.5 Tail bounds for the negative binomial distribution

Since the tail probabilities of a negative Binomial distribution are equal to the tail probabilities of some Binomial distribution, we can obtain tail bounds for the negative Binomial distribution through the Hoeffding tail bounds for the Binomial distribution.

context

```

  fixes  $p \ q :: \text{real}$ 
  assumes  $p: p \in \{0 < .. < 1\}$ 
  defines  $q \equiv 1 - p$ 
  begin

```

lemma $\text{prob-neg-binomial-pmf-ge-bound}$:

```

  fixes  $n :: \text{nat}$  and  $k :: \text{real}$ 
  defines  $\mu \equiv \text{real } n * q / p$ 

```

```

assumes  $k: k \geq 0$ 
shows  $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{x. \text{ real } x \geq \mu + k\}$ 
 $\leq \exp (- 2 * p \wedge 3 * k^2 / (n + p * k))$ 
proof -
  consider  $n = 0 \mid p = 1 \mid n > 0 \ p \neq 1$ 
    by blast
  thus ?thesis
proof cases
    assume [simp]:  $n = 0$ 
    show ?thesis using  $k$ 
      by (simp add: indicator-def  $\mu$ -def)
  next
    assume [simp]:  $p = 1$ 
    show ?thesis using  $k$ 
      by (auto simp add: indicator-def  $\mu$ -def  $q$ -def)
  next
    assume  $n: n > 0$  and  $p \neq 1$ 
    from  $\langle p \neq 1 \rangle$  and  $p$  have  $p: p \in \{0 < .. < 1\}$ 
      by auto
    from  $p$  have  $q: q \in \{0 < .. < 1\}$ 
      by (auto simp:  $q$ -def)

    define  $k1$  where  $k1 = \mu + k$ 
    have  $k1: k1 \geq \mu$ 
      using  $k$  by (simp add:  $k1$ -def)
    have  $k1 > 0$ 
      by (rule less-le-trans[ $OF - k1$ ]) (use  $p \ n$  in  $\langle \text{auto } \text{simp}: q\text{-def } \mu\text{-def} \rangle$ )

    define  $k1'$  where  $k1' = \text{nat } (\text{ceiling } k1)$ 
    have  $\mu \geq 0$  using  $p$ 
      by (auto simp:  $\mu$ -def  $q$ -def)
    have  $\neg(x < k1') \longleftrightarrow \text{real } x \geq k1$  for  $x$ 
      unfolding  $k1'$ -def by linarith
    hence  $\text{eq}: \text{UNIV} - \{.. < k1'\} = \{x. x \geq k1\}$ 
      by auto
    hence  $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{n. n \geq k1\} =$ 
 $1 - \text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{.. < k1'\}$ 
      using  $\text{measure-pmf.prob-compl}$ [of  $\{.. < k1'\}$   $\text{neg-binomial-pmf } n \ p$ ] by simp
    also have  $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{.. < k1'\} =$ 
 $\text{measure-pmf.prob } (\text{binomial-pmf } (n + k1' - 1) \ q) \ \{.. < k1'\}$ 
      unfolding  $q$ -def using  $p$  by (intro prob-neg-binomial-pmf-lessThan) auto
    also have  $1 - \dots = \text{measure-pmf.prob } (\text{binomial-pmf } (n + k1' - 1) \ q) \ \{n.$ 
 $n \geq k1\}$ 
      using  $\text{measure-pmf.prob-compl}$ [of  $\{.. < k1'\}$   $\text{binomial-pmf } (n + k1' - 1) \ q$ ]
    eq by simp
    also have  $\{x. \text{ real } x \geq k1\} = \{x. x \geq \text{real } (n + k1' - 1) * q + (k1 - \text{real } (n$ 
 $+ k1' - 1) * q)\}$ 
      by simp
    also have  $\text{measure-pmf.prob } (\text{binomial-pmf } (n + k1' - 1) \ q) \ \dots \leq$ 

```

```

      exp  $(-2 * (k1 - \text{real } (n + k1' - 1) * q)^2 / \text{real } (n + k1' - 1))$ 
proof (rule binomial-distribution.prob-ge)
  show binomial-distribution q
    by unfold-locales (use q in auto)
next
  show  $n + k1' - 1 > 0$ 
    using  $\langle k1 > 0 \rangle$  n unfolding k1'-def by linarith
next
  have  $\text{real } (n + \text{nat } \lceil k1 \rceil - 1) \leq \text{real } n + k1$ 
    using  $\langle k1 > 0 \rangle$  by linarith
  hence  $\text{real } (n + k1' - 1) * q \leq (\text{real } n + k1) * q$ 
    unfolding k1'-def by (intro mult-right-mono) (use p in  $\langle \text{simp-all add: } q\text{-def} \rangle$ )
  also have  $\dots \leq k1$ 
    using k1 p by (simp add: q-def field-simps  $\mu$ -def)
  finally show  $0 \leq k1 - \text{real } (n + k1' - 1) * q$ 
    by simp
qed
also have  $\{x. \text{real } (n + k1' - 1) * q + (k1 - \text{real } (n + k1' - 1) * q) \leq \text{real } x\} = \{x. \text{real } x \geq k1\}$ 
  by simp
also have  $\exp(-2 * (k1 - \text{real } (n + k1' - 1) * q)^2 / \text{real } (n + k1' - 1)) \leq \exp(-2 * (k1 - (n + k1) * q)^2 / (n + k1))$ 
proof -
  have  $\text{real } (n + k1' - \text{Suc } 0) \leq \text{real } n + k1$ 
    unfolding k1'-def using  $\langle k1 > 0 \rangle$  by linarith
  moreover have  $(\text{real } n + k1) * q \leq k1$ 
    using k1 p by (auto simp: q-def field-simps  $\mu$ -def)
  moreover have  $1 < n + k1'$ 
    using n  $\langle k1 > 0 \rangle$  unfolding k1'-def by linarith
  ultimately have  $2 * (k1 - \text{real } (n + k1' - 1) * q)^2 / \text{real } (n + k1' - 1) \geq 2 * (k1 - (n + k1) * q)^2 / (n + k1)$ 
    by (intro frac-le mult-left-mono power-mono mult-nonneg-nonneg mult-right-mono diff-mono)
    (use q in simp-all)
  thus ?thesis
    by simp
qed
also have  $\dots = \exp(-2 * (p * k1 - q * n)^2 / (k1 + n))$ 
  by (simp add: q-def algebra-simps)
also have  $-2 * (p * k1 - q * n)^2 = -2 * p^2 * (k1 - \mu)^2$ 
  using p by (auto simp: field-simps  $\mu$ -def)
also have  $k1 - \mu = k$ 
  by (simp add: k1-def  $\mu$ -def)
also note k1-def
also have  $\mu + k + \text{real } n = \text{real } n / p + k$ 
  using p by (simp add:  $\mu$ -def q-def field-simps)
also have  $-2 * p^2 * k^2 / (\text{real } n / p + k) = -2 * p^3 * k^2 / (p * k + n)$ 
  using p by (simp add: field-simps power3-eq-cube power2-eq-square)

```

```

    finally show ?thesis by (simp add: add-ac)
qed
qed

lemma prob-neg-binomial-pmf-le-bound:
  fixes n :: nat and k :: real
  defines  $\mu \equiv \text{real } n * q / p$ 
  assumes k:  $k \geq 0$ 
  shows  $\text{measure-pmf.prob } (\text{neg-binomial-pmf } n \ p) \ \{x. \text{ real } x \leq \mu - k\}$ 
     $\leq \exp (-2 * p ^ 3 * k^2 / (n - p * k))$ 
proof -
  consider  $n = 0 \mid p = 1 \mid k > \mu \mid n > 0 \ p \neq 1 \ k \leq \mu$ 
  by force
  thus ?thesis
  proof cases
    assume [simp]:  $n = 0$ 
    show ?thesis using k
    by (simp add: indicator-def  $\mu$ -def)
  next
    assume [simp]:  $p = 1$ 
    show ?thesis using k
    by (auto simp add: indicator-def  $\mu$ -def  $q$ -def)
  next
    assume  $k > \mu$ 
    hence  $\{x. \text{ real } x \leq \mu - k\} = \{\}$ 
    by auto
    thus ?thesis by simp
  next
    assume  $n: n > 0$  and  $p \neq 1$  and  $k \leq \mu$ 
    from  $\langle p \neq 1 \rangle$  and  $p$  have  $p: p \in \{0 < .. < 1\}$ 
    by auto
    from  $p$  have  $q: q \in \{0 < .. < 1\}$ 
    by (auto simp:  $q$ -def)

    define  $f :: \text{real} \Rightarrow \text{real}$  where  $f = (\lambda x. (p * x - q * n)^2 / (x + n))$ 
    have  $f\text{-mono}: f \ x \geq f \ y$  if  $x \geq 0 \ y \leq n * q / p \ x \leq y$  for  $x \ y :: \text{real}$ 
    using that(3)
  proof (rule DERIV-nonpos-imp-nonincreasing)
    fix  $t$  assume  $t: t \geq x \ t \leq y$ 
    have  $x > -n$ 
    using  $n \ \langle x \geq 0 \rangle$  by linarith
    hence  $(f \text{ has-field-derivative } ((p * t - q * n) * (n * (1 + p) + p * t) / (n +$ 
 $t) ^ 2)) \ (at \ t)$ 
    unfolding  $f$ -def using  $t$ 
  by (auto intro!: derivative-eq-intros simp: algebra-simps  $q$ -def power2-eq-square)
  moreover {
    have  $p * t \leq p * y$ 
    using  $p$  by (intro mult-left-mono  $t$ ) auto
    also have  $p * y \leq q * n$ 

```



```

    using ⟨y ≤ n * q / p⟩ p by (simp add: field-simps)
    finally have p * t ≤ q * n .
  }
  hence (p * t - q * n) * (n * (1 + p) + p * t) / (n + t) ^ 2 ≤ 0
    using p ⟨x ≥ 0⟩ t
    by (intro mult-nonpos-nonneg divide-nonpos-nonneg add-nonneg-nonneg
mult-nonneg-nonneg) auto
  ultimately show ∃ y. (f has-real-derivative y) (at t) ∧ y ≤ 0
    by blast
qed

define k1 where k1 = μ - k
have k1: k1 ≤ real n * q / p
  using assms by (simp add: μ-def k1-def)
have k1 ≥ 0
  using k ⟨k ≤ μ⟩ by (simp add: μ-def k1-def)

define k1' where k1' = nat (floor k1)
have μ ≥ 0 using p
  by (auto simp: μ-def q-def)
have (x ≤ k1') ⟷ real x ≤ k1 for x
  unfolding k1'-def not-less using ⟨k1 ≥ 0⟩ by linarith
hence eq: {n. n ≤ k1} = {..k1'}
  by auto
hence measure-pmf.prob (neg-binomial-pmf n p) {n. n ≤ k1} =
  measure-pmf.prob (neg-binomial-pmf n p) {..k1'}
  by simp
also have measure-pmf.prob (neg-binomial-pmf n p) {..k1'} =
  measure-pmf.prob (binomial-pmf (n + k1') q) {..k1'}
  unfolding q-def using p by (intro prob-neg-binomial-pmf-atMost) auto
also note eq [symmetric]
also have {x. real x ≤ k1} = {x. x ≤ real (n + k1') * q - (real (n + k1') *
q - real k1')}
  using eq by auto
also have measure-pmf.prob (binomial-pmf (n + k1') q) ... ≤
  exp (-2 * (real (n + k1') * q - real k1')² / real (n + k1'))
proof (rule binomial-distribution.prob-le)
  show binomial-distribution q
    by unfold-locales (use q in auto)
next
  show n + k1' > 0
    using ⟨k1 ≥ 0⟩ n unfolding k1'-def by linarith
next
  have p * k1' ≤ p * k1
    using p ⟨k1 ≥ 0⟩ by (intro mult-left-mono) (auto simp: k1'-def)
  also have ... ≤ q * n
    using k1 p by (simp add: field-simps)
  finally show 0 ≤ real (n + k1') * q - real k1'
    by (simp add: algebra-simps q-def)

```

```

qed
also have {x. real x ≤ real (n + k1') * q - (real (n + k1') * q - k1')} =
{..k1'}
  by auto
also have real (n + k1') * q - k1' = -(p * k1' - q * n)
  by (simp add: q-def algebra-simps)
also have ... ^ 2 = (p * k1' - q * n) ^ 2
  by algebra
also have - 2 * (p * real k1' - q * real n)^2 / real (n + k1') = -2 * f (real
k1')
  by (simp add: f-def)
also have f (real k1') ≥ f k1
  by (rule f-mono) (use ⟨k1 ≥ 0⟩ k1 in ⟨auto simp: k1'-def⟩)
hence exp (-2 * f (real k1')) ≤ exp (-2 * f k1)
  by simp
also have ... = exp (-2 * (p * k1 - q * n)^2 / (k1 + n))
  by (simp add: f-def)

also have -2 * (p * k1 - q * n)^2 = -2 * p^2 * (k1 - μ)^2
  using p by (auto simp: field-simps μ-def)
also have (k1 - μ) ^ 2 = k ^ 2
  by (simp add: k1-def μ-def)
also note k1-def
also have μ - k + real n = real n / p - k
  using p by (simp add: μ-def q-def field-simps)
also have - 2 * p^2 * k^2 / (real n / p - k) = - 2 * p ^ 3 * k^2 / (n - p * k)
  using p by (simp add: field-simps power3-eq-cube power2-eq-square)
also have {..k1'} = {x. real x ≤ μ - k}
  using eq by (simp add: k1-def)
finally show ?thesis .
qed
qed

```

Due to the function $\exp(-l/x)$ being concave for $x \geq \frac{l}{2}$, the above two bounds can be combined into the following one for moderate values of k . (cf. <https://math.stackexchange.com/questions/1565559>)

lemma *prob-neg-binomial-pmf-abs-ge-bound*:

```

fixes n :: nat and k :: real
defines μ ≡ real n * q / p
assumes k ≥ 0 and n-ge: n ≥ p * k * (p^2 * k + 1)
shows measure-pmf.prob (neg-binomial-pmf n p) {x. |real x - μ| ≥ k} ≤
  2 * exp (-2 * p ^ 3 * k ^ 2 / n)
proof (cases k = 0)
case False
with ⟨k ≥ 0⟩ have k: k > 0
  by auto
define l :: real where l = 2 * p ^ 3 * k ^ 2
have l: l > 0
  using p k by (auto simp: l-def)

```

```

define  $f :: \text{real} \Rightarrow \text{real}$  where  $f = (\lambda x. \exp (-l / x))$ 
define  $f'$  where  $f' = (\lambda x. -l * \exp (-l / x) / x ^ 2)$ 

have  $f'$ -mono:  $f' x \leq f' y$  if  $x \geq l / 2$   $x \leq y$  for  $x y :: \text{real}$ 
  using that(2)
proof (rule DERIV-nonneg-imp-nondecreasing)
  fix  $t$  assume  $t: x \leq t$   $t \leq y$ 
  have  $t > 0$ 
    using that  $l t$  by auto
  have ( $f'$  has-field-derivative  $(l * (2 * t - l) / (\exp (l / t) * t ^ 4)))$  (at  $t$ )
    unfolding  $f'$ -def using  $t$  that  $\langle t > 0 \rangle$ 
    by (auto intro!: derivative-eq-intros simp: field-simps exp-minus simp flip:
power-Suc)
  moreover have  $l * (2 * t - l) / (\exp (l / t) * t ^ 4) \geq 0$ 
    using that  $t l$  by (intro divide-nonneg-pos mult-nonneg-nonneg) auto
  ultimately show  $\exists y. (f' \text{ has-real-derivative } y) (at\ t) \wedge 0 \leq y$  by blast
qed

have convex: convex-on  $\{l/2..\}$   $(\lambda x. -f\ x)$  unfolding  $f$ -def
proof (intro convex-on-realI[where  $f' = f'$ ])
  show  $((\lambda x. - \exp (-l / x)) \text{ has-real-derivative } f' x) (at\ x)$  if  $x \in \{l/2..\}$  for  $x$ 
    using that  $l$ 
    by (auto intro!: derivative-eq-intros simp:  $f'$ -def power2-eq-square algebra-simps)
qed (use  $l$  in  $\langle \text{auto intro!: } f'\text{-mono} \rangle$ )

have eq:  $\{x. |\text{real } x - \mu| \geq k\} = \{x. \text{real } x \leq \mu - k\} \cup \{x. \text{real } x \geq \mu + k\}$ 
  by auto
have measure-pmf.prob (neg-binomial-pmf  $n\ p$ )  $\{x. |\text{real } x - \mu| \geq k\} \leq$ 
  measure-pmf.prob (neg-binomial-pmf  $n\ p$ )  $\{x. \text{real } x \leq \mu - k\} +$ 
  measure-pmf.prob (neg-binomial-pmf  $n\ p$ )  $\{x. \text{real } x \geq \mu + k\}$ 
  by (subst eq, rule measure-Un-le) auto
also have  $\dots \leq \exp (-2 * p ^ 3 * k^2 / (n - p * k)) + \exp (-2 * p ^ 3 * k^2 /$ 
 $(n + p * k))$ 
  unfolding  $\mu$ -def
  by (intro prob-neg-binomial-pmf-le-bound prob-neg-binomial-pmf-ge-bound add-mono
 $\langle k \geq 0 \rangle$ )
also have  $\dots = 2 * (1/2 * f (n - p * k) + 1/2 * f (n + p * k))$ 
  by (simp add:  $f$ -def  $l$ -def)
also have  $1/2 * f (n - p * k) + 1/2 * f (n + p * k) \leq f (1/2 * (n - p * k)$ 
 $+ 1/2 * (n + p * k))$ 
proof -
  let  $?x = n - p * k$  and  $?y = n + p * k$ 
  have  $le1: l / 2 \leq ?x$  using  $n$ -ge
    by (simp add:  $l$ -def power2-eq-square power3-eq-cube algebra-simps)
  also have  $\dots \leq ?y$ 
    using  $p\ k$  by simp
  finally have  $le2: l / 2 \leq ?y$  .
  have  $-f ((1 - 1 / 2) * _R\ ?x + (1 / 2) * _R\ ?y) \leq (1 - 1 / 2) * -f\ ?x + 1 /$ 
 $2 * -f\ ?y$ 

```

```

    using le1 le2 by (intro convex-onD[OF convex]) auto
    thus ?thesis by simp
qed
also have  $1/2 * (n - p * k) + 1/2 * (n + p * k) = n$ 
  by (simp add: algebra-simps)
also have  $2 * f\ n = 2 * \exp\ (-l / n)$ 
  by (simp add: f-def)
finally show ?thesis
  by (simp add: l-def)
qed auto

end

end

```

theory *Stream-Space*

imports

Infinite-Product-Measure

HOL-Library.Stream

HOL-Library.Linear-Temporal-Logic-on-Streams

begin

lemma *stream-eq-Stream-iff*: $s = x \#\# t \longleftrightarrow (\text{shd } s = x \wedge \text{stl } s = t)$
 by (cases s) simp

lemma *Stream-snth*: $(x \#\# s) !! n = (\text{case } n \text{ of } 0 \Rightarrow x \mid \text{Suc } n \Rightarrow s !! n)$
 by (cases n) simp-all

definition *to-stream* :: $(\text{nat} \Rightarrow 'a) \Rightarrow 'a \text{ stream}$ **where**
to-stream $X = \text{smap } X \text{ nats}$

lemma *to-stream-nat-case*: $\text{to-stream } (\text{case-nat } x \ X) = x \#\# \text{to-stream } X$
unfolding *to-stream-def*
 by (subst siterate.ctr) (simp add: smap-siterate[symmetric] stream.map-comp comp-def)

lemma *to-stream-in-streams*: $\text{to-stream } X \in \text{streams } S \longleftrightarrow (\forall n. X\ n \in S)$
 by (simp add: to-stream-def streams-iff-snth)

definition *stream-space* :: $'a \text{ measure} \Rightarrow 'a \text{ stream measure}$ **where**
stream-space $M =$
 $\text{distr } (\Pi_M i \in \text{UNIV}. M) (\text{vimage-algebra } (\text{streams } (\text{space } M)) \text{ snth } (\Pi_M i \in \text{UNIV}. M)) \text{ to-stream}$

lemma *space-stream-space*: $\text{space } (\text{stream-space } M) = \text{streams } (\text{space } M)$
 by (simp add: stream-space-def)

lemma *streams-stream-space[intro]*: $\text{streams } (\text{space } M) \in \text{sets } (\text{stream-space } M)$

using *sets.top*[of *stream-space M*] **by** (*simp add: space-stream-space*)

lemma *stream-space-Stream*:

$x \#\# \omega \in \text{space } (\text{stream-space } M) \longleftrightarrow x \in \text{space } M \wedge \omega \in \text{space } (\text{stream-space } M)$

by (*simp add: space-stream-space streams-Stream*)

lemma *stream-space-eq-distr*: $\text{stream-space } M = \text{distr } (\Pi_M i \in \text{UNIV}. M) (\text{stream-space } M) \text{ to-stream}$

unfolding *stream-space-def* **by** (*rule distr-cong*) *auto*

lemma *sets-stream-space-cong*[*measurable-cong*]:

$\text{sets } M = \text{sets } N \implies \text{sets } (\text{stream-space } M) = \text{sets } (\text{stream-space } N)$

using *sets-eq-imp-space-eq*[of *M N*] **by** (*simp add: stream-space-def vimage-algebra-def cong: sets-PiM-cong*)

lemma *measurable-snth-PiM*: $(\lambda \omega. n. \omega !! n) \in \text{measurable } (\text{stream-space } M) (\Pi_M i \in \text{UNIV}. M)$

by (*auto intro!: measurable-vimage-algebra1*

simp: space-PiM streams-iff-sset sset-range image-subset-iff stream-space-def)

lemma *measurable-snth*[*measurable*]: $(\lambda \omega. \omega !! n) \in \text{measurable } (\text{stream-space } M) M$

using *measurable-snth-PiM measurable-component-singleton* **by** (*rule measurable-compose*) *simp*

lemma *measurable-shd*[*measurable*]: $\text{shd} \in \text{measurable } (\text{stream-space } M) M$

using *measurable-snth*[of 0] **by** *simp*

lemma *measurable-stream-space2*:

assumes *f-snth*: $\bigwedge n. (\lambda x. f x !! n) \in \text{measurable } N M$

shows $f \in \text{measurable } N (\text{stream-space } M)$

unfolding *stream-space-def measurable-distr-eq2*

proof (*rule measurable-vimage-algebra2*)

show $f \in \text{space } N \rightarrow \text{streams } (\text{space } M)$

using *f-snth*[*THEN measurable-space*] **by** (*auto simp add: streams-iff-sset sset-range*)

show $(\lambda x. (!!)(f x)) \in \text{measurable } N (\Pi_M i \in \text{UNIV}. M)$

proof (*rule measurable-PiM-single'*)

show $(\lambda x. (!!)(f x)) \in \text{space } N \rightarrow \text{UNIV} \rightarrow_E \text{space } M$

using *f-snth*[*THEN measurable-space*] **by** *auto*

qed (*rule f-snth*)

qed

lemma *measurable-stream-coinduct*[*consumes 1, case-names shd stl, coinduct set: measurable*]:

assumes *F f*

assumes *h*: $\bigwedge f. F f \implies (\lambda x. \text{shd } (f x)) \in \text{measurable } N M$

assumes *t*: $\bigwedge f. F f \implies F (\lambda x. \text{stl } (f x))$

shows $f \in \text{measurable } N (\text{stream-space } M)$

proof (rule measurable-stream-space2)
fix n **show** $(\lambda x. f\ x \ \text{!!}\ n) \in \text{measurable } N\ M$
using $\langle F\ f \rangle$ **by** (induction n arbitrary: f) (auto intro: $h\ t$)
qed

lemma measurable-sdrop[measurable]: $\text{sdrop } n \in \text{measurable } (\text{stream-space } M) (\text{stream-space } M)$
by (rule measurable-stream-space2) (simp add: sdrop-snth)

lemma measurable-stl[measurable]: $(\lambda \omega. \text{stl } \omega) \in \text{measurable } (\text{stream-space } M) (\text{stream-space } M)$
by (rule measurable-stream-space2) (simp del: snth.simps add: snth.simps[symmetric])

lemma measurable-to-stream[measurable]: $\text{to-stream} \in \text{measurable } (\prod_M i \in \text{UNIV}. M) (\text{stream-space } M)$
by (rule measurable-stream-space2) (simp add: to-stream-def)

lemma measurable-Stream[measurable (raw)]:
assumes $f[\text{measurable}]: f \in \text{measurable } N\ M$
assumes $g[\text{measurable}]: g \in \text{measurable } N\ (\text{stream-space } M)$
shows $(\lambda x. f\ x \ \#\# \ g\ x) \in \text{measurable } N\ (\text{stream-space } M)$
by (rule measurable-stream-space2) (simp add: Stream-snth)

lemma measurable-smap[measurable]:
assumes $X[\text{measurable}]: X \in \text{measurable } N\ M$
shows $\text{smap } X \in \text{measurable } (\text{stream-space } N) (\text{stream-space } M)$
by (rule measurable-stream-space2) simp

lemma measurable-stake[measurable]:
 $\text{stake } i \in \text{measurable } (\text{stream-space } (\text{count-space } \text{UNIV})) (\text{count-space } (\text{UNIV} :: 'a::\text{countable list set}))$
by (induct i) auto

lemma measurable-shift[measurable]:
assumes $f: f \in \text{measurable } N\ (\text{stream-space } M)$
assumes $[\text{measurable}]: g \in \text{measurable } N\ (\text{stream-space } M)$
shows $(\lambda x. \text{stake } n\ (f\ x) \ @- \ g\ x) \in \text{measurable } N\ (\text{stream-space } M)$
using f **by** (induction n arbitrary: f) simp-all

lemma measurable-case-stream-replace[measurable (raw)]:
 $(\lambda x. f\ x\ (\text{shd } (g\ x))\ (\text{stl } (g\ x))) \in \text{measurable } M\ N \implies (\lambda x. \text{case-stream } (f\ x)\ (g\ x)) \in \text{measurable } M\ N$
unfolding stream.case-eq-if .

lemma measurable-ev-at[measurable]:
assumes $[\text{measurable}]: \text{Measurable.pred } (\text{stream-space } M)\ P$
shows $\text{Measurable.pred } (\text{stream-space } M)\ (\text{ev-at } P\ n)$
by (induction n) auto

lemma *measurable-alw*[*measurable*]:

Measurable.pred (stream-space M) P \implies Measurable.pred (stream-space M) (alw P)

unfolding *alw-def*

by (*coinduction rule: measurable-gfp-coinduct*) (*auto simp: inf-continuous-def*)

lemma *measurable-ev*[*measurable*]:

Measurable.pred (stream-space M) P \implies Measurable.pred (stream-space M) (ev P)

unfolding *ev-def*

by (*coinduction rule: measurable-lfp-coinduct*) (*auto simp: sup-continuous-def*)

lemma *measurable-until*:

assumes [*measurable*]: *Measurable.pred (stream-space M) φ Measurable.pred (stream-space M) ψ*

shows *Measurable.pred (stream-space M) (φ until ψ)*

unfolding *UNTIL-def*

by (*coinduction rule: measurable-gfp-coinduct*) (*simp-all add: inf-continuous-def fun-eq-iff*)

lemma *measurable-holds* [*measurable*]: *Measurable.pred M P \implies Measurable.pred (stream-space M) (holds P)*

unfolding *holds.simps[abs-def]*

by (*rule measurable-compose[OF measurable-shd]*) *simp*

lemma *measurable-hld*[*measurable*]: **assumes** [*measurable*]: *t \in sets M* **shows** *Measurable.pred (stream-space M) (HLD t)*

unfolding *HLD-def* **by** *measurable*

lemma *measurable-nxt*[*measurable (raw)*]:

Measurable.pred (stream-space M) P \implies Measurable.pred (stream-space M) (nxt P)

unfolding *nxt.simps[abs-def]* **by** *simp*

lemma *measurable-suntil*[*measurable*]:

assumes [*measurable*]: *Measurable.pred (stream-space M) Q Measurable.pred (stream-space M) P*

shows *Measurable.pred (stream-space M) (Q suntil P)*

unfolding *suntil-def* **by** (*coinduction rule: measurable-lfp-coinduct*) (*auto simp: sup-continuous-def*)

lemma *measurable-szip*:

($\lambda(\omega 1, \omega 2). \text{szip } \omega 1 \ \omega 2 \in \text{measurable (stream-space } M \otimes_M \text{ stream-space } N)$

(stream-space (M \otimes_M N))

proof (*rule measurable-stream-space2*)

fix *n*

have ($\lambda x. (\text{case } x \text{ of } (\omega 1, \omega 2) \Rightarrow \text{szip } \omega 1 \ \omega 2) !! n = (\lambda(\omega 1, \omega 2). (\omega 1 !! n, \omega 2 !! n))$)

by *auto*

also have $\dots \in \text{measurable } (\text{stream-space } M \otimes_M \text{stream-space } N) (M \otimes_M N)$
 by *measurable*
 finally show $(\lambda x. (\text{case } x \text{ of } (\omega 1, \omega 2) \Rightarrow \text{szip } \omega 1 \ \omega 2) !! n) \in \text{measurable}$
 $(\text{stream-space } M \otimes_M \text{stream-space } N) (M \otimes_M N)$
 .
 qed

lemma (in *prob-space*) *prob-space-stream-space*: *prob-space* (*stream-space* M)
 proof –
 interpret *product-prob-space* $\lambda\cdot. M \text{ UNIV } ..$
 show ?thesis
 by (subst *stream-space-eq-distr*) (auto intro!: $P.\text{prob-space-distr}$)
 qed

lemma (in *prob-space*) *nn-integral-stream-space*:
 assumes [*measurable*]: $f \in \text{borel-measurable } (\text{stream-space } M)$
 shows $(\int^+ X. f X \ \partial \text{stream-space } M) = (\int^+ x. (\int^+ X. f (x \#\# X) \ \partial \text{stream-space } M) \ \partial M)$
 proof –
 interpret S : *sequence-space* $M ..$
 interpret P : *pair-sigma-finite* $M \ \Pi_M \ i::\text{nat} \in \text{UNIV}. M ..$
 have $(\int^+ X. f X \ \partial \text{stream-space } M) = (\int^+ X. f (\text{to-stream } X) \ \partial S.S)$
 by (subst *stream-space-eq-distr*) (simp add: *nn-integral-distr*)
 also have $\dots = (\int^+ X. f (\text{to-stream } ((\lambda(s, \omega). \text{case-nat } s \ \omega) X)) \ \partial(M \otimes_M S.S))$
 by (subst $S.\text{PiM-iter}[\text{symmetric}]$) (simp add: *nn-integral-distr*)
 also have $\dots = (\int^+ x. \int^+ X. f (\text{to-stream } ((\lambda(s, \omega). \text{case-nat } s \ \omega) (x, X))) \ \partial S.S \ \partial M)$
 by (subst $S.\text{nn-integral-fst}$) (simp-all)
 also have $\dots = (\int^+ x. \int^+ X. f (x \#\# \text{to-stream } X) \ \partial S.S \ \partial M)$
 by (auto intro!: *nn-integral-cong* simp: *to-stream-nat-case*)
 also have $\dots = (\int^+ x. \int^+ X. f (x \#\# X) \ \partial \text{stream-space } M \ \partial M)$
 by (subst *stream-space-eq-distr*)
 (simp add: *nn-integral-distr* cong: *nn-integral-cong*)
 finally show ?thesis .
 qed

lemma (in *prob-space*) *emeasure-stream-space*:
 assumes $X[\text{measurable}]$: $X \in \text{sets } (\text{stream-space } M)$
 shows *emeasure* (*stream-space* M) $X = (\int^+ t. \text{emeasure } (\text{stream-space } M) \{x \in \text{space } (\text{stream-space } M). t \#\# x \in X\} \ \partial M)$
 proof –
 have eq: $\bigwedge x \ xs. xs \in \text{space } (\text{stream-space } M) \implies x \in \text{space } M \implies$
 $\text{indicator } X (x \#\# xs) = \text{indicator } \{xs \in \text{space } (\text{stream-space } M). x \#\# xs \in X\} xs$
 by (auto split: *split-indicator*)
 show ?thesis
 using *nn-integral-stream-space*[of *indicator* X]


```

    apply (auto intro!: nn-integral-cong)
    apply (subst nn-integral-cong)
    apply (rule eq)
    apply simp-all
    done
qed

lemma (in prob-space) prob-stream-space:
  assumes  $P[\text{measurable}]: \{x \in \text{space } (\text{stream-space } M). P\ x\} \in \text{sets } (\text{stream-space } M)$ 
  shows  $\mathcal{P}(x \text{ in stream-space } M. P\ x) = (\int^+ t. \mathcal{P}(x \text{ in stream-space } M. P\ (t \text{ ## } x)) \partial M)$ 
  proof -
    interpret  $S: \text{prob-space stream-space } M$ 
    by (rule prob-space-stream-space)
    show ?thesis
      unfolding  $S.\text{emeasure-eq-measure}[\text{symmetric}]$ 
      by (subst  $\text{emeasure-stream-space}$ ) (auto simp:  $\text{stream-space-Stream intro!: nn-integral-cong}$ )
  qed

lemma (in prob-space) AE-stream-space:
  assumes  $[\text{measurable}]: \text{Measurable.pred } (\text{stream-space } M) P$ 
  shows  $(AE\ X \text{ in stream-space } M. P\ X) = (AE\ x \text{ in } M. AE\ X \text{ in stream-space } M. P\ (x \text{ ## } X))$ 
  proof -
    interpret  $\text{stream}: \text{prob-space stream-space } M$ 
    by (rule prob-space-stream-space)

    have  $\text{eq}: \bigwedge x\ X. \text{indicator } \{x. \neg P\ x\} (x \text{ ## } X) = \text{indicator } \{X. \neg P\ (x \text{ ## } X)\} X$ 
    by (auto split: split-indicator)
    show ?thesis
      apply (subst  $AE\text{-iff-}nn\text{-integral, simp}$ )
      apply (subst  $nn\text{-integral-stream-space, simp}$ )
      apply (subst eq)
      apply (subst  $nn\text{-integral-0-iff-AE, simp}$ )
      apply (simp add:  $AE\text{-iff-}nn\text{-integral}[\text{symmetric}]$ )
      done
  qed

lemma (in prob-space) AE-stream-all:
  assumes  $[\text{measurable}]: \text{Measurable.pred } M\ P$  and  $P: AE\ x \text{ in } M. P\ x$ 
  shows  $AE\ x \text{ in stream-space } M. \text{stream-all } P\ x$ 
  proof -
    { fix  $n$  have  $AE\ x \text{ in stream-space } M. P\ (x \text{ !! } n)$ 
      proof (induct  $n$ )
        case 0 with  $P$  show ?case
          by (subst  $AE\text{-stream-space}$ ) (auto elim!:  $\text{eventually-mono}$ )
      next

```

```

      case (Suc n) then show ?case
      by (subst AE-stream-space) auto
    qed }
  then show ?thesis
  unfolding stream-all-def by (simp add: AE-all-countable)
qed

```

lemma *streams-sets*:

```

  assumes X[measurable]: X ∈ sets M shows streams X ∈ sets (stream-space M)
proof -
  have streams X = {x ∈ space (stream-space M). x ∈ streams X}
  using streams-mono[OF - sets.sets-into-space[OF X]] by (auto simp: space-stream-space)
  also have ... = {x ∈ space (stream-space M). gfp (λp x. shd x ∈ X ∧ p (stl x))
x}
  apply (simp add: set-eq-iff streams-def streamsp-def)
  apply (intro allI conj-cong refl arg-cong2[where f=gfp] ext)
  apply (case-tac xa)
  apply auto
  done
  also have ... ∈ sets (stream-space M)
  apply (intro predE)
  apply (coinduction rule: measurable-gfp-coinduct)
  apply (auto simp: inf-continuous-def)
  done
  finally show ?thesis .
qed

```

lemma *sets-stream-space-in-sets*:

```

  assumes space: space N = streams (space M)
  assumes sets: ⋀i. (λx. x !! i) ∈ measurable N M
  shows sets (stream-space M) ⊆ sets N
  unfolding stream-space-def sets-distr
  by (auto intro!: sets-image-in-sets measurable-Sup2 measurable-vimage-algebra2
del: subsetI equalityI
simp add: sets-PiM-eq-proj snth-in space sets cong: measurable-cong-sets)

```

lemma *sets-stream-space-eq*: sets (stream-space M) =

```

  sets (SUP i ∈ UNIV. vimage-algebra (streams (space M)) (λs. s !! i) M)
by (auto intro!: sets-stream-space-in-sets sets-Sup-in-sets sets-image-in-sets
measurable-Sup1 snth-in measurable-vimage-algebra1 del: subsetI
simp: space-Sup-eq-UN space-stream-space)

```

lemma *sets-restrict-stream-space*:

```

  assumes S[measurable]: S ∈ sets M
  shows sets (restrict-space (stream-space M) (streams S)) = sets (stream-space
(restrict-space M S))
  using S[THEN sets.sets-into-space]
  apply (subst restrict-space-eq-vimage-algebra)
  apply (simp add: space-stream-space streams-mono2)

```

```

apply (subst vimage-algebra-cong[OF refl refl sets-stream-space-eq])
apply (subst sets-stream-space-eq)
apply (subst sets-vimage-Sup-eq[where  $Y = \text{streams } (\text{space } M)$ ])
apply simp
apply auto []
apply (auto intro: streams-mono) []
apply auto []
apply (simp add: image-image space-restrict-space)
apply (simp add: vimage-algebra-cong[OF refl refl restrict-space-eq-vimage-algebra])
apply (subst (1 2) vimage-algebra-vimage-algebra-eq)
apply (auto simp: streams-mono snth-in )
done

```

primrec $sstart :: 'a \text{ set} \Rightarrow 'a \text{ list} \Rightarrow 'a \text{ stream set}$ **where**

```

   $sstart \ S \ [] = \text{streams } S$ 
| [simp del]:  $sstart \ S \ (x \# \ xs) = (\#\#) \ x \ ' \ sstart \ S \ xs$ 

```

lemma $in\text{-}sstart[simp]: s \in sstart \ S \ (x \# \ xs) \longleftrightarrow shd \ s = x \wedge stl \ s \in sstart \ S \ xs$
by (cases s) (auto simp: $sstart.simps(2)$)

lemma $sstart\text{-}in\text{-}streams: xs \in \text{lists } S \Longrightarrow sstart \ S \ xs \subseteq \text{streams } S$
by (induction xs) (auto simp: $sstart.simps(2)$)

lemma $sstart\text{-}eq: x \in \text{streams } S \Longrightarrow x \in sstart \ S \ xs = (\forall i < \text{length } xs. x !! i = xs !! i)$
by (induction xs arbitrary: x) (auto simp: $nth\text{-}Cons \text{ streams}\text{-}stl \text{ split: nat.splits}$)

lemma $sstart\text{-}sets: sstart \ S \ xs \in \text{sets } (\text{stream-space } (\text{count-space } UNIV))$

proof (induction xs)

case ($Cons \ x \ xs$)

note $Cons[measurable]$

have $sstart \ S \ (x \# \ xs) =$

$\{s \in \text{space } (\text{stream-space } (\text{count-space } UNIV)). shd \ s = x \wedge stl \ s \in sstart \ S \ xs\}$

by (simp add: $set\text{-}eq\text{-}iff \ \text{space}\text{-}\text{stream}\text{-}\text{space}$)

also have $\dots \in \text{sets } (\text{stream-space } (\text{count-space } UNIV))$

by $measurable$

finally show $?case$.

qed (simp add: $\text{streams}\text{-}\text{sets}$)

lemma $\sigma\text{-sets}\text{-}singletons:$

assumes $\text{countable } S$

shows $\sigma\text{-sets } S \ ((\lambda s. \{s\})'S) = \text{Pow } S$

proof safe

interpret $\sigma\text{-algebra } S \ \sigma\text{-sets } S \ ((\lambda s. \{s\})'S)$

by (rule $\sigma\text{-algebra}\text{-}\sigma\text{-sets}$) auto

fix A **assume** $A \subseteq S$

with assms **have** $(\bigcup a \in A. \{a\}) \in \sigma\text{-sets } S \ ((\lambda s. \{s\})'S)$

by ($\text{intro countable}\text{-}UN'$) ($\text{auto dest: countable}\text{-}\text{subset}$)

then show $A \in \sigma\text{-sets } S \ ((\lambda s. \{s\})'S)$

by simp
qed (auto dest: sigma-sets-into-sp[rotated])

lemma sets-count-space-eq-sigma:

countable $S \implies \text{sets } (\text{count-space } S) = \text{sets } (\text{sigma } S ((\lambda s. \{s\})'S))$

by (subst sets-measure-of) (auto simp: sigma-sets-singletons)

lemma sets-stream-space-sstart:

assumes $S[\text{simp}]$: countable S

shows $\text{sets } (\text{stream-space } (\text{count-space } S)) = \text{sets } (\text{sigma } (\text{streams } S) (\text{sstart } S' \text{lists } S \cup \{\{\}\}))$

proof

have $[\text{simp}]: \text{sstart } S \text{ ' lists } S \subseteq \text{Pow } (\text{streams } S)$

by (simp add: image-subset-iff sstart-in-streams)

let $?S = \text{sigma } (\text{streams } S) (\text{sstart } S \text{ ' lists } S \cup \{\{\}\})$

{ fix i a assume $a \in S$

{ fix x have $(x !! i = a \wedge x \in \text{streams } S) \longleftrightarrow (\exists xs \in \text{lists } S. \text{length } xs = i \wedge x \in \text{sstart } S (xs @ [a]))$

proof (induction i arbitrary: x)

case (Suc i) from this[of stl x] show ?case

by (simp add: length-Suc-conv Bex-def ex-simps[symmetric] del: ex-simps)
(metis stream.collapse streams-Stream)

qed (insert $\langle a \in S \rangle$, auto intro: streams-stl in-streams) }

then have $(\lambda x. x !! i) - ' \{a\} \cap \text{streams } S = (\bigcup xs \in \{xs \in \text{lists } S. \text{length } xs = i\}. \text{sstart } S (xs @ [a]))$

by (auto simp add: set-eq-iff)

also have $\dots \in \text{sets } ?S$

using $\langle a \in S \rangle$ by (intro sets.countable-UN') (auto intro!: sigma-sets.Basic image-eqI)

finally have $(\lambda x. x !! i) - ' \{a\} \cap \text{streams } S \in \text{sets } ?S . \}$

then show $\text{sets } (\text{stream-space } (\text{count-space } S)) \subseteq \text{sets } (\text{sigma } (\text{streams } S) (\text{sstart } S' \text{lists } S \cup \{\{\}\}))$

by (intro sets-stream-space-in-sets) (auto simp: measurable-count-space-eq-countable snth-in)

have $\text{sigma-sets } (\text{space } (\text{stream-space } (\text{count-space } S))) (\text{sstart } S' \text{lists } S \cup \{\{\}\}) \subseteq \text{sets } (\text{stream-space } (\text{count-space } S))$

proof (safe intro!: sets.sigma-sets-subset)

fix xs assume $\forall x \in \text{set } xs. x \in S$

then have $\text{sstart } S xs = \{x \in \text{space } (\text{stream-space } (\text{count-space } S)). \forall i < \text{length } xs. x !! i = xs ! i\}$

by (induction xs)

(auto simp: space-stream-space nth-Cons split: nat.split intro: in-streams streams-stl)

also have $\dots \in \text{sets } (\text{stream-space } (\text{count-space } S))$

by measurable

finally show $\text{sstart } S xs \in \text{sets } (\text{stream-space } (\text{count-space } S)) .$

qed

then show $\text{sets } (\text{sigma } (\text{streams } S) (\text{sstart } S \text{ 'lists } S \cup \{\{\}\})) \subseteq \text{sets } (\text{stream-space } (\text{count-space } S))$
by (*simp add: space-stream-space*)
qed

lemma *Int-stable-sstart*: $\text{Int-stable } (\text{sstart } S \text{ 'lists } S \cup \{\{\}\})$

proof –

{ fix $xs\ ys$ **assume** $xs \in \text{lists } S\ ys \in \text{lists } S$
then have $\text{sstart } S\ xs \cap \text{sstart } S\ ys \in \text{sstart } S \text{ 'lists } S \cup \{\{\}\}$
proof (*induction xs ys rule: list-induct2'*)
case ($_4\ x\ xs\ y\ ys$)
show *?case*
proof *cases*
assume $x = y$
then have $\text{sstart } S\ (x \# xs) \cap \text{sstart } S\ (y \# ys) = (\# \#)\ x \text{ ' } (\text{sstart } S\ xs \cap \text{sstart } S\ ys)$
by (*auto simp: image-iff intro!: stream.collapse[symmetric]*)
also have $\dots \in \text{sstart } S \text{ 'lists } S \cup \{\{\}\}$
using $_4$ **by** (*auto simp: sstart.simps(2)[symmetric] del: in-listsD*)
finally show *?case* .
qed *auto*
qed (*simp-all add: sstart-in-streams inf.absorb1 inf.absorb2 image-eqI* [**where** $x = []$]) }
then show *?thesis*
by (*auto simp: Int-stable-def*)
qed

lemma *stream-space-eq-sstart*:

assumes $S[\text{simp}]$: *countable* S
assumes P : *prob-space* M *prob-space* N
assumes ae : $AE\ x\ \text{in } M. x \in \text{streams } S\ AE\ x\ \text{in } N. x \in \text{streams } S$
assumes $\text{sets-}M$: $\text{sets } M = \text{sets } (\text{stream-space } (\text{count-space } UNIV))$
assumes $\text{sets-}N$: $\text{sets } N = \text{sets } (\text{stream-space } (\text{count-space } UNIV))$
assumes $*$: $\bigwedge xs. xs \neq [] \implies xs \in \text{lists } S \implies \text{emeasure } M (\text{sstart } S\ xs) = \text{emeasure } N (\text{sstart } S\ xs)$
shows $M = N$
proof (*rule measure-eqI-restrict-generator[OF Int-stable-sstart]*)
have $[\text{simp}]$: $\text{sstart } S \text{ 'lists } S \subseteq \text{Pow } (\text{streams } S)$
by (*simp add: image-subset-iff sstart-in-streams*)

interpret M : *prob-space* M **by** *fact*

show $\text{sstart } S \text{ 'lists } S \cup \{\{\}\} \subseteq \text{Pow } (\text{streams } S)$
by (*auto dest: sstart-in-streams del: in-listsD*)

{ fix $M :: \text{'a stream measure}$ **assume** M : $\text{sets } M = \text{sets } (\text{stream-space } (\text{count-space } UNIV))$

have $\text{sets } (\text{restrict-space } M (\text{streams } S)) = \text{sigma-sets } (\text{streams } S) (\text{sstart } S \text{ 'lists } S \cup \{\{\}\})$

```

    by (subst sets-restrict-space-cong[OF M])
      (simp add: sets-restrict-stream-space restrict-count-space sets-stream-space-sstart)
  }
  from this[OF sets-M] this[OF sets-N]
  show sets (restrict-space M (streams S)) = sigma-sets (streams S) (sstart S ‘
lists S ∪ {{{}}})
    sets (restrict-space N (streams S)) = sigma-sets (streams S) (sstart S ‘ lists
S ∪ {{{}}})
  by auto
  show {streams S} ⊆ sstart S ‘ lists S ∪ {{{}}
    ∪ {streams S} = streams S ∧ s. s ∈ {streams S} ⇒ emeasure M s ≠ ∞
  using M.emeasure-space-1 space-stream-space[of count-space S] sets-eq-imp-space-eq[OF
sets-M]
  by (auto simp add: image-eqI[where x=[]])
  show sets M = sets N
  by (simp add: sets-M sets-N)
next
fix X assume X ∈ sstart S ‘ lists S ∪ {{{}}
then obtain xs where X = {} ∨ (xs ∈ lists S ∧ X = sstart S xs)
  by auto
moreover have emeasure M (streams S) = 1
  using ae by (intro prob-space.emeasure-eq-1-AE[OF P(1)]) (auto simp: sets-M
streams-sets)
moreover have emeasure N (streams S) = 1
  using ae by (intro prob-space.emeasure-eq-1-AE[OF P(2)]) (auto simp: sets-N
streams-sets)
ultimately show emeasure M X = emeasure N X
  using P[THEN prob-space.emeasure-space-1]
  by (cases xs = []) (auto simp: * space-stream-space del: in-listsD)
qed (auto simp: * ae sets-M del: in-listsD intro!: streams-sets)

lemma sets-sstart[measurable]: sstart Ω xs ∈ sets (stream-space (count-space UNIV))
proof (induction xs)
case (Cons x xs)
note this[measurable]
have sstart Ω (x # xs) = {ω ∈ space (stream-space (count-space UNIV)). ω ∈
sstart Ω (x # xs)}
  by (auto simp: space-stream-space)
also have ... ∈ sets (stream-space (count-space UNIV))
  unfolding in-sstart by measurable
finally show ?case .
qed (auto intro!: streams-sets)

primrec scylinder :: 'a set ⇒ 'a set list ⇒ 'a stream set
where
  scylinder S [] = streams S
| scylinder S (A # As) = {ω ∈ streams S. shd ω ∈ A ∧ stl ω ∈ scylinder S As}

lemma scylinder-streams: scylinder S xs ⊆ streams S

```

```

by (induction xs) auto

lemma sets-scyylinder:  $(\forall x \in \text{set } xs. x \in \text{sets } S) \implies \text{scylinder } (\text{space } S) \text{ } xs \in \text{sets}$ 
  (stream-space S)
  by (induction xs) (auto simp: space-stream-space[symmetric])

lemma stream-space-eq-scyylinder:
  assumes P: prob-space M prob-space N
  assumes Int-stable G and S: sets S = sets (sigma (space S) G)
    and C: countable C  $C \subseteq G \cup C = \text{space } S$  and G:  $G \subseteq \text{Pow } (\text{space } S)$ 
  assumes sets-M: sets M = sets (stream-space S)
  assumes sets-N: sets N = sets (stream-space S)
  assumes *:  $\bigwedge xs. xs \neq [] \implies xs \in \text{lists } G \implies \text{emeasure } M (\text{scylinder } (\text{space } S) xs) = \text{emeasure } N (\text{scylinder } (\text{space } S) xs)$ 
  shows M = N
proof (rule measure-eqI-generator-eq)
  interpret M: prob-space M by fact
  interpret N: prob-space N by fact

  let ?G = scylinder (space S) ‘ lists G
  show sc-Pow: ?G  $\subseteq \text{Pow } (\text{streams } (\text{space } S))$ 
    using scylinder-streams by auto

  have sets (stream-space S) = sets (sigma (streams (space S)) ?G)
    (is ?S = sets ?R)
  proof (rule antisym)
    let ?V =  $\lambda i. \text{vimage-algebra } (\text{streams } (\text{space } S)) (\lambda s. s !! i) S$ 
    show ?S  $\subseteq \text{sets } ?R$ 
      unfolding sets-stream-space-eq
    proof (safe intro!: sets-Sup-in-sets del: subsetI equalityI)
      fix i :: nat
      show space (?V i) = space ?R
        using scylinder-streams by (subst space-measure-of) auto
      { fix A assume A  $\in G$ 
        then have scylinder (space S) (replicate i (space S) @ [A]) =  $(\lambda s. s !! i)$ 
        – ‘ A  $\cap \text{streams } (\text{space } S)$ 
          by (induction i) (auto simp add: streams-shd streams-stl cong: conj-cong)
          also have scylinder (space S) (replicate i (space S) @ [A]) =  $(\bigcup xs \in \{xs \in \text{lists } C. \text{length } xs = i\}. \text{scylinder } (\text{space } S) (xs @ [A]))$ 
            apply (induction i)
            apply auto []
            apply (simp add: length-Suc-conv set-eq-iff ex-simps(1,2)[symmetric] cong: conj-cong del: ex-simps(1,2))
            apply rule
            subgoal for i x
              apply (cases x)
              apply (subst (2) C(3)[symmetric])
              apply (simp del: ex-simps(1,2) add: ex-simps(1,2)[symmetric] ac-simps Bex-def)

```

```

    apply auto
  done
done
done
finally have (λs. s !! i) - ‘ A ∩ streams (space S) = (⋃ xs ∈ {xs ∈ lists C.
length xs = i}. scylinder (space S) (xs @ [A]))
..
also have ... ∈ ?R
using C(2) ⟨A ∈ G⟩
by (intro sets.countable-UN' countable-Collect countable-lists C)
(auto intro!: in-measure-of[OF sc-Pow] imageI)
finally have (λs. s !! i) - ‘ A ∩ streams (space S) ∈ ?R . }
then show sets (?V i) ⊆ ?R
apply (subst vimage-algebra-cong[OF refl refl S])
apply (subst vimage-algebra-sigma[OF G])
apply (simp add: streams-iff-snth) []
apply (subst sigma-le-sets)
apply auto
done
qed
have G ⊆ sets S
unfolding S using G by auto
with C(2) show sets ?R ⊆ ?S
unfolding sigma-le-sets[OF sc-Pow] by (auto intro!: sets-scylinder)
qed
then show sets M = sigma-sets (streams (space S)) (scylinder (space S) ‘ lists
G)
sets N = sigma-sets (streams (space S)) (scylinder (space S) ‘ lists G)
unfolding sets-M sets-N by (simp-all add: sc-Pow)

show Int-stable ?G
proof (rule Int-stableI-image)
fix xs ys assume xs ∈ lists G ys ∈ lists G
then show ∃ zs ∈ lists G. scylinder (space S) xs ∩ scylinder (space S) ys =
scylinder (space S) zs
proof (induction xs arbitrary: ys)
case Nil then show ?case
by (auto simp add: Int-absorb1 scylinder-streams)
next
case xs: (Cons x xs)
show ?case
proof (cases ys)
case Nil with xs.hyps show ?thesis
by (auto simp add: Int-absorb2 scylinder-streams intro!: bexI[of - x # xs])
next
case ys: (Cons y ys')
with xs.IH[of ys'] xs.prem obtain zs where
zs ∈ lists G and eq: scylinder (space S) xs ∩ scylinder (space S) ys' =
scylinder (space S) zs
by auto

```



```

show ?thesis
proof (intro beXI[of - (x ∩ y)#zs])
  show x ∩ y # zs ∈ lists G
  using ⟨zs∈lists G⟩ ⟨x∈G⟩ ⟨ys∈lists G⟩ ys ⟨Int-stable G⟩ [THEN Int-stableD,
of x y] by auto
  show scylinder (space S) (x # xs) ∩ scylinder (space S) ys = scylinder
(space S) (x ∩ y # zs)
  by (auto simp add: eq[symmetric] ys)
qed
qed
qed
qed

show range (λ::nat. streams (space S)) ⊆ scylinder (space S) ‘ lists G
(⋃ i. streams (space S)) = streams (space S)
emeasure M (streams (space S)) ≠ ∞
by (auto intro!: image-eqI[of - - []])

fix X assume X ∈ scylinder (space S) ‘ lists G
then obtain xs where xs: xs ∈ lists G and eq: X = scylinder (space S) xs
by auto
then show emeasure M X = emeasure N X
proof (cases xs = [])
  assume xs = [] then show ?thesis
    unfolding eq
    using sets-M[THEN sets-eq-imp-space-eq] sets-N[THEN sets-eq-imp-space-eq]
M.emeasure-space-1 N.emeasure-space-1
    by (simp add: space-stream-space[symmetric])
  next
    assume xs ≠ [] with xs show ?thesis
    unfolding eq by (intro *)
qed
qed

lemma stream-space-coinduct:
fixes R :: 'a stream measure ⇒ 'a stream measure ⇒ bool
assumes R A B
assumes R: ⋀ A B. R A B ⇒ ∃ K∈space (prob-algebra M).
∃ A'∈M →M prob-algebra (stream-space M). ∃ B'∈M →M prob-algebra (stream-space
M).
(AE y in K. R (A' y) (B' y) ∨ A' y = B' y) ∧
A = do { y ← K; ω ← A' y; return (stream-space M) (y ## ω) } ∧
B = do { y ← K; ω ← B' y; return (stream-space M) (y ## ω) }
shows A = B
proof (rule stream-space-eq-scylinder)
let ?step = λK L. do { y ← K; ω ← L y; return (stream-space M) (y ## ω) }
{ fix K A A' assume K: K ∈ space (prob-algebra M)
and A'[measurable]: A' ∈ M →M prob-algebra (stream-space M) and A-eq:
A = ?step K A'

```

```

have ps: prob-space A
  unfolding A-eq by (rule prob-space-bind'[OF K]) measurable
have sets A = sets (stream-space M)
  unfolding A-eq by (rule sets-bind'[OF K]) measurable
note ps this }
note ** = this

{ fix A B assume R A B
  obtain K A' B' where K: K ∈ space (prob-algebra M)
    and A': A' ∈ M →M prob-algebra (stream-space M) A = ?step K A'
    and B': B' ∈ M →M prob-algebra (stream-space M) B = ?step K B'
    using R[OF ‹R A B›] by blast
  have prob-space A prob-space B sets A = sets (stream-space M) sets B = sets
    (stream-space M)
    using **[OF K A'] **[OF K B'] by auto }
note R-D = this

show prob-space A prob-space B sets A = sets (stream-space M) sets B = sets
  (stream-space M)
  using R-D[OF ‹R A B›] by auto

show Int-stable (sets M) sets M = sets (sigma (space M) (sets M)) countable
  {space M}
  {space M} ⊆ sets M ∪ {space M} = space M sets M ⊆ Pow (space M)
  using sets.space-closed[of M] by (auto simp: Int-stable-def)

{ fix A As L K assume K[measurable]: K ∈ space (prob-algebra M) and A: A
  ∈ sets M As ∈ lists (sets M)
  and L[measurable]: L ∈ M →M prob-algebra (stream-space M)
  from A have [measurable]: ∀ x ∈ set (A # As). x ∈ sets M ∀ x ∈ set As. x ∈ sets
  M
  by auto
  have [simp]: space K = space M sets K = sets M
    using K by (auto simp: space-prob-algebra intro!: sets-eq-imp-space-eq)
  have [simp]: x ∈ space M ⇒ sets (L x) = sets (stream-space M) for x
    using measurable-space[OF L] by (auto simp: space-prob-algebra)
  note sets-scyylinder[measurable]
  have *: indicator (scylinder (space M) (A # As)) (x ## ω) =
    (indicator A x * indicator (scylinder (space M) As) ω :: ennreal) for ω x
  using scylinder-streams[of space M As] ‹A ∈ sets M› [THEN sets.sets-into-space]
  by (auto split: split-indicator)
  have emeasure (?step K L) (scylinder (space M) (A # As)) = (∫+ y. L y
    (scylinder (space M) As) * indicator A y ∂K)
    apply (subst emeasure-bind-prob-algebra[OF K])
    apply measurable
    apply (rule nn-integral-cong)
    apply (subst emeasure-bind-prob-algebra[OF L [THEN measurable-space]])
    apply (simp-all add: ac-simps * nn-integral-cmult-indicator del: scylin-
      der.simps)

```

```

    apply measurable
  done }
note emeasure-step = this

fix Xs assume Xs ∈ lists (sets M)
from this ⟨R A B⟩ show emeasure A (scylinder (space M) Xs) = emeasure B
(scylinder (space M) Xs)
proof (induction Xs arbitrary: A B)
  case (Cons X Xs)
  obtain K A' B' where K: K ∈ space (prob-algebra M)
    and A'[measurable]: A' ∈ M →M prob-algebra (stream-space M) and A: A =
    ?step K A'
    and B'[measurable]: B' ∈ M →M prob-algebra (stream-space M) and B: B =
    ?step K B'
    and AE-R: AE x in K. R (A' x) (B' x) ∨ A' x = B' x
    using R[OF ⟨R A B⟩] by blast

  show ?case
    unfolding A B emeasure-step[OF K Cons.hyps A'] emeasure-step[OF K
    Cons.hyps B']
    apply (rule nn-integral-cong-AE)
    using AE-R by eventually-elim (auto simp add: Cons.IH)
  next
  case Nil
  note R-D[OF this]
  from this(1,2)[THEN prob-space.emeasure-space-1] this(3,4)[THEN sets-eq-imp-space-eq]
  show ?case
    by (simp add: space-stream-space)
qed
qed
end

```

theory *Tree-Space*

imports *HOL–Analysis.Analysis HOL–Library.Tree*

begin

lemma *countable-lfp*:

assumes *step*: $\bigwedge Y. \text{countable } Y \implies \text{countable } (F Y)$

and *cont*: *Order-Continuity.sup-continuous* *F*

shows *countable* (*lfp* *F*)

by(*subst sup-continuous-lfp*[*OF cont*])(*simp add: countable-funpow*[*OF step*])

lemma *countable-lfp-apply*:

assumes *step*: $\bigwedge Y x. (\bigwedge x. \text{countable } (Y x)) \implies \text{countable } (F Y x)$

and *cont*: *Order-Continuity.sup-continuous* *F*

shows *countable* (*lfp* *F* *x*)

proof –

```

{ fix n
  have  $\bigwedge x. \text{countable } ((F \rightsquigarrow n) \text{ bot } x)$ 
  by (induct n)(auto intro: step) }
thus ?thesis using cont by (simp add: sup-continuous-lfp)
qed

```

```

inductive-set trees :: 'a set  $\Rightarrow$  'a tree set for S :: 'a set where
  [intro!]: Leaf  $\in$  trees S
| l  $\in$  trees S  $\Rightarrow$  r  $\in$  trees S  $\Rightarrow$  v  $\in$  S  $\Rightarrow$  Node l v r  $\in$  trees S

```

```

lemma Node-in-trees-iff[simp]: Node l v r  $\in$  trees S  $\longleftrightarrow$  (l  $\in$  trees S  $\wedge$  v  $\in$  S  $\wedge$ 
r  $\in$  trees S)
  by (subst trees.simps) auto

```

```

lemma trees-sub-lfp: trees S  $\subseteq$  lfp ( $\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{ \text{Node } l \ v \ r \})))$ )
proof
  have mono: mono ( $\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{ \text{Node } l \ v \ r \})))$ )
  by (auto simp: mono-def)
  fix t assume t  $\in$  trees S then show t  $\in$  lfp ( $\lambda T. T \cup \{\text{Leaf}\} \cup (\bigcup l \in T. (\bigcup v \in S. (\bigcup r \in T. \{ \text{Node } l \ v \ r \})))$ )
  proof induction
    case 1 then show ?case
      by (subst lfp-unfold[OF mono]) auto
    next
      case 2 then show ?case
        by (subst lfp-unfold[OF mono]) auto
  qed
qed

```

```

lemma countable-trees: countable A  $\Rightarrow$  countable (trees A)
proof (intro countable-subset[OF trees-sub-lfp] countable-lfp
  sup-continuous-sup sup-continuous-const sup-continuous-id)
  show sup-continuous ( $\lambda T. (\bigcup l \in T. \bigcup v \in A. \bigcup r \in T. \{ \langle l, v, r \rangle \})$ )
  unfolding sup-continuous-def
  proof (intro allI impI equalityI subsetI, goal-cases)
    case (1 M t)
    then obtain i j :: nat and l x r where t = Node l x r x  $\in$  A l  $\in$  M i r  $\in$  M j
    by auto
    hence l  $\in$  M (max i j) r  $\in$  M (max i j)
    using incseqD[OF  $\langle \text{incseq } M \rangle$ , of i max i j] incseqD[OF  $\langle \text{incseq } M \rangle$ , of j max i j] by auto
    with  $\langle t = \text{Node } l \ x \ r \rangle$  and  $\langle x \in A \rangle$  show ?case by auto
  qed auto
qed auto

```

```

lemma trees-UNIV[simp]: trees UNIV = UNIV
proof -
  have t  $\in$  trees UNIV for t :: 'a tree

```

```

    by (induction t) (auto intro: trees.intros(2))
  then show ?thesis by auto
qed

```

```

instance tree :: (countable) countable

```

```

proof

```

```

  have countable (UNIV :: 'a tree set)

```

```

    by (subst trees-UNIV[symmetric]) (intro countable-trees[OF countableI-type])

```

```

  then show  $\exists$  to-nat::'a tree  $\Rightarrow$  nat. inj to-nat

```

```

    by (auto simp: countable-def)

```

```

qed

```

```

lemma map-in-trees[intro]: ( $\bigwedge x. x \in \text{set-tree } t \Rightarrow f x \in S$ )  $\Rightarrow$  map-tree f t  $\in$ 
trees S

```

```

  by (induction t) (auto intro: trees.intros(2))

```

```

primrec trees-cyl :: 'a set tree  $\Rightarrow$  'a tree set where

```

```

  trees-cyl Leaf = {Leaf}

```

```

| trees-cyl (Node l v r) = ( $\bigcup l' \in \text{trees-cyl } l. (\bigcup v' \in v. (\bigcup r' \in \text{trees-cyl } r. \{\text{Node } l' v' r'\}))$ )

```

```

definition tree-sigma :: 'a measure  $\Rightarrow$  'a tree measure

```

```

where

```

```

  tree-sigma M = sigma (trees (space M)) (trees-cyl ' trees (sets M))

```

```

lemma Node-in-trees-cyl: Node l' v' r'  $\in$  trees-cyl t  $\longleftrightarrow$ 

```

```

  ( $\exists l v r. t = \text{Node } l v r \wedge l' \in \text{trees-cyl } l \wedge r' \in \text{trees-cyl } r \wedge v' \in v$ )

```

```

  by (cases t) auto

```

```

lemma trees-cyl-sub-trees:

```

```

  assumes t  $\in$  trees A A  $\subseteq$  Pow B shows trees-cyl t  $\subseteq$  trees B

```

```

  using assms(1)

```

```

proof induction

```

```

  case (2 l v r) with  $\langle A \subseteq \text{Pow } B \rangle$  show ?case

```

```

    by (auto intro!: trees.intros(2))

```

```

qed auto

```

```

lemma trees-cyl-sets-in-space: trees-cyl ' trees (sets M)  $\subseteq$  Pow (trees (space M))

```

```

  using trees-cyl-sub-trees[OF - sets.space-closed, of - M] by auto

```

```

lemma space-tree-sigma: space (tree-sigma M) = trees (space M)

```

```

  unfolding tree-sigma-def by (rule space-measure-of-conv)

```

```

lemma sets-tree-sigma-eq: sets (tree-sigma M) = sigma-sets (trees (space M))
(trees-cyl ' trees (sets M))

```

```

  unfolding tree-sigma-def by (rule sets-measure-of) (rule trees-cyl-sets-in-space)

```

```

lemma Leaf-in-space-tree-sigma [measurable, simp, intro]: Leaf  $\in$  space (tree-sigma M)

```

```

by (auto simp: space-tree-sigma)

lemma Leaf-in-tree-sigma [measurable, simp, intro]: {Leaf} ∈ sets (tree-sigma M)
  unfolding sets-tree-sigma-eq
  by (rule sigma-sets.Basic) (auto intro: trees.intros(2) image-eqI[where x=Leaf])

lemma trees-cyl-map-treeI: t ∈ trees-cyl (map-tree (λx. A) t) if *: t ∈ trees A
  using * by induction auto

lemma trees-cyl-map-in-sets:
  (λx. x ∈ set-tree t ⇒ f x ∈ sets M) ⇒ trees-cyl (map-tree f t) ∈ sets (tree-sigma M)
  by (subst sets-tree-sigma-eq) auto

lemma Node-in-tree-sigma:
  assumes L: X ∈ sets (M ⊗M (tree-sigma M ⊗M tree-sigma M))
  shows {Node l v r | l v r. (v, l, r) ∈ X} ∈ sets (tree-sigma M)
proof -
  let ?E = λs::unit tree. trees-cyl (map-tree (λ-. space M) s)
  have 1: countable (range ?E)
    by (intro countable-image countableI-type)
  have 2: trees-cyl ‘ trees (sets M) ⊆ Pow (space (tree-sigma M))
    using trees-cyl-sets-in-space[of M] by (simp add: space-tree-sigma)
  have 3: sets (tree-sigma M) = sigma-sets (space (tree-sigma M)) (trees-cyl ‘ trees (sets M))
    unfolding sets-tree-sigma-eq by (simp add: space-tree-sigma)
  have 4: (⋃ s. ?E s) = space (tree-sigma M)
  proof (safe; clarsimp simp: space-tree-sigma)
    fix t s assume t ∈ trees-cyl (map-tree (λ-. unit. space M) s)
    then show t ∈ trees (space M)
      by (induction s arbitrary: t) auto
  next
    fix t assume t ∈ trees (space M)
    then show ∃ t'. t ∈ ?E t'
      by (intro exI[of - map-tree (λ-. ()) t])
        (auto simp: tree.map-comp comp-def intro: trees-cyl-map-treeI)
  qed
  have 5: range ?E ⊆ trees-cyl ‘ trees (sets M) by auto
  let ?P = {A × B | A B. A ∈ trees-cyl ‘ trees (sets M) ∧ B ∈ trees-cyl ‘ trees (sets M)}
  have P: sets (tree-sigma M ⊗M tree-sigma M) = sets (sigma (space (tree-sigma M) × space (tree-sigma M)) ?P)
    by (rule sets-pair-eq[OF 2 3 1 5 4 2 3 1 5 4])

  have sets (M ⊗M (tree-sigma M ⊗M tree-sigma M)) =
    sets (sigma (space M × space (tree-sigma M ⊗M tree-sigma M)) {A × BC |
  A BC. A ∈ sets M ∧ BC ∈ ?P})
  proof (rule sets-pair-eq)
    show sets M ⊆ Pow (space M) sets M = sigma-sets (space M) (sets M)

```

```

    by (auto simp: sets.sigma-sets-eq sets.space-closed)
  show countable {space M} {space M}  $\subseteq$  sets M  $\cup$  {space M} = space M
    by auto
  show ?P  $\subseteq$  Pow (space (tree-sigma M  $\otimes_M$  tree-sigma M))
    using trees-cyl-sets-in-space[of M]
    by (auto simp: space-pair-measure space-tree-sigma subset-eq)
  then show sets (tree-sigma M  $\otimes_M$  tree-sigma M) =
    sigma-sets (space (tree-sigma M  $\otimes_M$  tree-sigma M)) ?P
    by (subst P, subst sets-measure-of) (auto simp: space-tree-sigma space-pair-measure)
  show countable (( $\lambda(a, b). a \times b$ ) ‘ (range ?E  $\times$  range ?E))
    by (intro countable-image countable-SIGMA countableI-type)
  show (( $\lambda(a, b). a \times b$ ) ‘ (range ?E  $\times$  range ?E))  $\subseteq$  ?P
    by auto
qed (insert 4, auto simp: space-pair-measure space-tree-sigma set-eq-iff)
also have ... = sigma-sets (space M  $\times$  trees (space M)  $\times$  trees (space M))
  {A  $\times$  BC | A BC. A  $\in$  sets M  $\wedge$  BC  $\in$  {A  $\times$  B | A B.
    A  $\in$  trees-cyl ‘ trees (sets M)  $\wedge$  B  $\in$  trees-cyl ‘ trees (sets M)}}
  (is - = sigma-sets ?X ?Y) using sets.space-closed[of M] trees-cyl-sub-trees[of -
sets M space M]
    by (subst sets-measure-of)
    (auto simp: space-pair-measure space-tree-sigma)
  also have ?Y = {A  $\times$  trees-cyl B  $\times$  trees-cyl C | A B C. A  $\in$  sets M  $\wedge$ 
    B  $\in$  trees (sets M)  $\wedge$  C  $\in$  trees (sets M)} by blast
  finally have X  $\in$  sigma-sets (space M  $\times$  trees (space M)  $\times$  trees (space M))
    {A  $\times$  trees-cyl B  $\times$  trees-cyl C | A B C. A  $\in$  sets M  $\wedge$  B  $\in$  trees (sets M)  $\wedge$ 
C  $\in$  trees (sets M)}
    using assms by blast
  then show ?thesis
proof induction
  case (Basic A')
  then obtain A B C where A' = A  $\times$  trees-cyl B  $\times$  trees-cyl C
    and *: A  $\in$  sets M B  $\in$  trees (sets M) C  $\in$  trees (sets M)
    by auto
  then have {Node l v r | l v r. (v, l, r)  $\in$  A'} = trees-cyl (Node B A C)
    by auto
  then show ?case
  by (auto simp del: trees-cyl.simps simp: sets-tree-sigma-eq intro!: sigma-sets.Basic
*)
next
  case Empty show ?case by auto
next
  case (Compl A)
  have {Node l v r | l v r. (v, l, r)  $\in$  space M  $\times$  trees (space M)  $\times$  trees (space
M) - A} =
    (space (tree-sigma M) - {Node l v r | l v r. (v, l, r)  $\in$  A}) - {Leaf}
    by (auto simp: space-tree-sigma elim: trees.cases)
  also have ...  $\in$  sets (tree-sigma M)
    by (intro sets.Diff Compl) auto
  finally show ?case .

```

```

next
  case (Union I)
  have *: {Node l v r | l v r. (v, l, r) ∈ ⋃ (I ‘ UNIV)} =
    (⋃ i. {Node l v r | l v r. (v, l, r) ∈ I i}) by auto
  show ?case unfolding * using Union(2) by (intro sets.countable-UN) auto
qed
qed

lemma measurable-left[measurable]: left ∈ tree-sigma M →M tree-sigma M
proof (rule measurableI)
  show t ∈ space (tree-sigma M) ⇒ left t ∈ space (tree-sigma M) for t
    by (cases t) (auto simp: space-tree-sigma)
  fix A assume A: A ∈ sets (tree-sigma M)
  from sets.sets-into-space[OF this]
  have *: left -‘ A ∩ space (tree-sigma M) =
    (if Leaf ∈ A then {Leaf} else {}) ∪
    {Node a v r | a v r. (v, a, r) ∈ space M × A × space (tree-sigma M)}
    by (auto simp: space-tree-sigma elim: trees.cases)
  show left -‘ A ∩ space (tree-sigma M) ∈ sets (tree-sigma M)
    unfolding * using A by (intro sets.Un Node-in-tree-sigma pair-measureI) auto
qed

lemma measurable-right[measurable]: right ∈ tree-sigma M →M tree-sigma M
proof (rule measurableI)
  show t ∈ space (tree-sigma M) ⇒ right t ∈ space (tree-sigma M) for t
    by (cases t) (auto simp: space-tree-sigma)
  fix A assume A: A ∈ sets (tree-sigma M)
  from sets.sets-into-space[OF this]
  have *: right -‘ A ∩ space (tree-sigma M) =
    (if Leaf ∈ A then {Leaf} else {}) ∪
    {Node l v a | l v a. (v, l, a) ∈ space M × space (tree-sigma M) × A}
    by (auto simp: space-tree-sigma elim: trees.cases)
  show right -‘ A ∩ space (tree-sigma M) ∈ sets (tree-sigma M)
    unfolding * using A by (intro sets.Un Node-in-tree-sigma pair-measureI) auto
qed

lemma measurable-value': value ∈ restrict-space (tree-sigma M) (-{Leaf}) →M M
proof (rule measurableI)
  show t ∈ space (restrict-space (tree-sigma M) (- {Leaf})) ⇒ value t ∈ space M for t
    by (cases t) (auto simp: space-restrict-space space-tree-sigma)
  fix A assume A: A ∈ sets M
  from sets.sets-into-space[OF this]
  have value -‘ A ∩ space (restrict-space (tree-sigma M) (- {Leaf})) =
    {Node l a r | l a r. (a, l, r) ∈ A × space (tree-sigma M) × space (tree-sigma M)}
    by (auto simp: space-tree-sigma space-restrict-space elim: trees.cases)
  also have ... ∈ sets (tree-sigma M)

```



```

using  $A$  by (intro sets.Un Node-in-tree-sigma pair-measureI) auto
finally show value - ‘  $A \cap \text{space } (\text{restrict-space } (\text{tree-sigma } M) (- \{ \text{Leaf} \})) \in$ 
  sets (restrict-space (tree-sigma  $M$ ) (- {Leaf}))
by (auto simp: sets-restrict-space-iff space-restrict-space)
qed

```

```

lemma measurable-value[measurable (raw)]:
  assumes  $f \in X \rightarrow_M \text{tree-sigma } M$ 
  and  $\bigwedge x. x \in \text{space } X \implies f\ x \neq \text{Leaf}$ 
  shows  $(\lambda \omega. \text{value } (f\ \omega)) \in X \rightarrow_M M$ 
proof -
  from assms have  $f \in X \rightarrow_M \text{restrict-space } (\text{tree-sigma } M) (- \{ \text{Leaf} \})$ 
  by (intro measurable-restrict-space2) auto
  from this and measurable-value' show ?thesis by (rule measurable-compose)
qed

```

```

lemma measurable-Node [measurable]:
   $(\lambda(l,x,r). \text{Node } l\ x\ r) \in \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \rightarrow_M \text{tree-sigma } M$ 
proof (rule measurable-sigma-sets)
  show sets (tree-sigma  $M$ ) = sigma-sets (trees (space  $M$ )) (trees-cyl ‘ trees (sets  $M$ ))
  by (simp add: sets-tree-sigma-eq)
  show trees-cyl ‘ trees (sets  $M$ )  $\subseteq$  Pow (trees (space  $M$ ))
  by (rule trees-cyl-sets-in-space)
  show  $(\lambda(l, x, r). \langle l, x, r \rangle) \in \text{space } (\text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M)$ 
   $\rightarrow$  trees (space  $M$ )
  by (auto simp: space-pair-measure space-tree-sigma)
  fix  $A$  assume  $t: A \in \text{trees-cyl } \text{'trees (sets } M\text{'})}$ 
  then obtain  $t$  where  $t: t \in \text{trees (sets } M\text{'}) } A = \text{trees-cyl } t$  by auto
  show  $(\lambda(l, x, r). \langle l, x, r \rangle) - \text{' } A \cap$ 
    space (tree-sigma  $M \otimes_M M \otimes_M \text{tree-sigma } M$ )
     $\in$  sets (tree-sigma  $M \otimes_M M \otimes_M \text{tree-sigma } M$ )
  proof (cases  $t$ )
  case Leaf
  have  $(\lambda(l, x, r). \langle l, x, r \rangle) - \text{' } \{ \text{Leaf} :: \text{'a tree} \} = \{ \}$  by auto
  with Leaf show ?thesis using  $t$  by simp
  next
  case (Node  $l\ B\ r$ )
  hence  $(\lambda(l, x, r). \langle l, x, r \rangle) - \text{' } A \cap \text{space } (\text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M) =$ 
    trees-cyl  $l \times B \times \text{trees-cyl } r$ 
  using  $t$  and Node and trees-cyl-sub-trees[of - sets  $M$  space  $M$ ]
  by (auto simp: space-pair-measure space-tree-sigma
    dest: sets.sets-into-space[of -  $M$ ])
  thus ?thesis using  $t$  and Node
  by (auto intro!: pair-measureI simp: sets-tree-sigma-eq)
qed

```

qed

lemma *measurable-Node'* [*measurable (raw)*]:
assumes [*measurable*]: $l \in B \rightarrow_M \text{tree-sigma } A$
assumes [*measurable*]: $x \in B \rightarrow_M A$
assumes [*measurable*]: $r \in B \rightarrow_M \text{tree-sigma } A$
shows $(\lambda y. \text{Node } (l y) (x y) (r y)) \in B \rightarrow_M \text{tree-sigma } A$
proof –
have $(\lambda y. \text{Node } (l y) (x y) (r y)) = (\lambda(a,b,c). \text{Node } a b c) \circ (\lambda y. (l y, x y, r y))$
by (*simp add: o-def*)
also have $\dots \in B \rightarrow_M \text{tree-sigma } A$
by (*intro measurable-comp[OF - measurable-Node] simp-all*)
finally show *?thesis* .
 qed

lemma *measurable-rec-tree*[*measurable (raw)*]:
assumes $t: t \in B \rightarrow_M \text{tree-sigma } M$
assumes $l: l \in B \rightarrow_M A$
assumes $n: (\lambda(x, l, v, r, al, ar). n x l v r al ar) \in$
 $(B \otimes_M \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \otimes_M A \otimes_M A) \rightarrow_M A$ (**is**
 $?N \in ?M \rightarrow_M A$)
shows $(\lambda x. \text{rec-tree } (l x) (n x) (t x)) \in B \rightarrow_M A$
proof (*rule measurable-piecewise-restrict*)
let $?C = \lambda t. \lambda s::\text{unit tree}. t - ' \text{trees-cyl } (\text{map-tree } (\lambda-. \text{space } M) s)$
show *countable* (*range* ($?C t$)) **by** (*intro countable-image countableI-type*)
show $\text{space } B \subseteq (\bigcup s. ?C t s)$
proof (*safe; clarsimp*)
fix x **assume** $x: x \in \text{space } B$ **have** $t x \in \text{trees } (\text{space } M)$
using t [*THEN measurable-space, OF x*] **by** (*simp add: space-tree-sigma*)
then show $\exists xa::\text{unit tree}. t x \in \text{trees-cyl } (\text{map-tree } (\lambda-. \text{space } M) xa)$
by (*intro exI[of - map-tree (\lambda-. ()) (t x)]*)
 $(\text{simp add: tree.map-comp comp-def trees-cyl-map-treeI})$
 qed
fix Ω **assume** $\Omega \in \text{range } (?C t)$
then obtain $s::\text{unit tree}$ **where** $\Omega: \Omega = ?C t s$ **by** *auto*
then show $\Omega \cap \text{space } B \in \text{sets } B$
by (*safe intro!: measurable-sets[OF t] trees-cyl-map-in-sets*)
show $(\lambda x. \text{rec-tree } (l x) (n x) (t x)) \in \text{restrict-space } B \Omega \rightarrow_M A$
unfolding Ω **using** t
proof (*induction s arbitrary: t*)
case *Leaf*
show *?case*
proof (*rule measurable-cong[THEN iffD2]*)
fix ω **assume** $\omega \in \text{space } (\text{restrict-space } B (?C t \text{Leaf}))$
then show $\text{rec-tree } (l \omega) (n \omega) (t \omega) = l \omega$
by (*auto simp: space-restrict-space*)
next
show $l \in \text{restrict-space } B (?C t \text{Leaf}) \rightarrow_M A$
using l **by** (*rule measurable-restrict-space1*)

```

qed
next
case (Node ls u rs)
let ?F = λω. ?N (ω, left (t ω), value (t ω), right (t ω),
  rec-tree (l ω) (n ω) (left (t ω)), rec-tree (l ω) (n ω) (right (t ω)))
show ?case
proof (rule measurable-cong[THEN iffD2])
  fix ω assume ω ∈ space (restrict-space B (?C t (Node ls u rs)))
  then show rec-tree (l ω) (n ω) (t ω) = ?F ω
    by (auto simp: space-restrict-space)
next
show ?F ∈ (restrict-space B (?C t (Node ls u rs))) →M A
  apply (intro measurable-compose[OF - n] measurable-Pair[rotated])
  subgoal
    apply (rule measurable-restrict-mono[OF Node(2)])
    apply (rule measurable-compose[OF Node(3) measurable-right])
    by auto
  subgoal
    apply (rule measurable-restrict-mono[OF Node(1)])
    apply (rule measurable-compose[OF Node(3) measurable-left])
    by auto
  subgoal
    by (rule measurable-restrict-space1)
    (rule measurable-compose[OF Node(3) measurable-right])
  subgoal
    apply (rule measurable-compose[OF - measurable-value'])
    apply (rule measurable-restrict-space3[OF Node(3)])
    by auto
  subgoal
    by (rule measurable-restrict-space1)
    (rule measurable-compose[OF Node(3) measurable-left])
  by (rule measurable-restrict-space1) auto
qed
qed
qed

lemma measurable-case-tree [measurable (raw)]:
  assumes t ∈ B →M tree-sigma M
  assumes l ∈ B →M A
  assumes (λ(x, l, v, r). n x l v r)
    ∈ B ⊗M tree-sigma M ⊗M M ⊗M tree-sigma M →M A
  shows (λx. case-tree (l x) (n x) (t x)) ∈ B →M (A :: 'a measure)
proof -
  define n' where n' = (λx l v r (-::'a) (-::'a). n x l v r)
  have (λx. case-tree (l x) (n x) (t x)) = (λx. rec-tree (l x) (n' x) (t x))
    (is - = (λx. rec-tree - (?n' x) -)) by (rule ext) (auto split: tree.splits simp:
n'-def)
  also have ... ∈ B →M A
  proof (rule measurable-rec-tree)

```

```

    have ( $\lambda(x, l, v, r, al, ar). n' x l v r al ar$ ) =
      ( $\lambda(x, l, v, r). n x l v r$ )  $\circ$  ( $\lambda(x, l, v, r, al, ar). (x, l, v, r)$ )
    by (simp add: n'-def o-def case-prod-unfold)
    also have  $\dots \in B \otimes_M \text{tree-sigma } M \otimes_M M \otimes_M \text{tree-sigma } M \otimes_M A$ 
     $\otimes_M A \rightarrow_M A$ 
    using assms( $\beta$ ) by measurable
    finally show ( $\lambda(x, l, v, r, al, ar). n' x l v r al ar$ )  $\in \dots$  .
  qed (insert assms, simp-all)
  finally show ?thesis .
qed

hide-const (open) left
hide-const (open) right

end

```

28 Conditional Expectation

```

theory Conditional-Expectation
imports Probability-Measure
begin

```

28.1 Restricting a measure to a sub-sigma-algebra

```

definition subalgebra::'a measure  $\Rightarrow$  'a measure  $\Rightarrow$  bool where
  subalgebra  $M F = ((\text{space } F = \text{space } M) \wedge (\text{sets } F \subseteq \text{sets } M))$ 

```

```

lemma sub-measure-space:
  assumes  $i: \text{subalgebra } M F$ 
  shows measure-space ( $\text{space } M$ ) ( $\text{sets } F$ ) (emeasure  $M$ )
proof -
  have sigma-algebra ( $\text{space } M$ ) ( $\text{sets } F$ )
  by (metis  $i$  measure-space measure-space-def subalgebra-def)
  moreover have positive ( $\text{sets } F$ ) (emeasure  $M$ )
  using Sigma-Algebra.positive-def by auto
  moreover have countably-additive ( $\text{sets } F$ ) (emeasure  $M$ )
  by (meson countably-additive-def emeasure-countably-additive  $i$  order-trans sub-
algebra-def subsetCE)
  ultimately show ?thesis unfolding measure-space-def by simp
qed

```

```

definition restr-to-subalg::'a measure  $\Rightarrow$  'a measure  $\Rightarrow$  'a measure where
  restr-to-subalg  $M F = \text{measure-of } (\text{space } M) (\text{sets } F) (\text{emeasure } M)$ 

```

```

lemma space-restr-to-subalg:
   $\text{space } (\text{restr-to-subalg } M F) = \text{space } M$ 
unfolding restr-to-subalg-def by (simp add: space-measure-of-conv)

```

```

lemma sets-restr-to-subalg [measurable-cong]:

```

assumes *subalgebra* $M F$
shows $\text{sets } (\text{restr-to-subalg } M F) = \text{sets } F$
unfolding *restr-to-subalg-def* **by** (*metis sets.sets-measure-of-eq assms subalgebra-def*)

lemma *emeasure-restr-to-subalg*:

assumes *subalgebra* $M F$
 $A \in \text{sets } F$
shows $\text{emeasure } (\text{restr-to-subalg } M F) A = \text{emeasure } M A$
unfolding *restr-to-subalg-def*
by (*metis assms subalgebra-def emeasure-measure-of-conv sub-measure-space sets.sigma-sets-eq*)

lemma *null-sets-restr-to-subalg*:

assumes *subalgebra* $M F$
shows $\text{null-sets } (\text{restr-to-subalg } M F) = \text{null-sets } M \cap \text{sets } F$
proof
have $x \in \text{null-sets } M \cap \text{sets } F$ **if** $x \in \text{null-sets } (\text{restr-to-subalg } M F)$ **for** x
by (*metis that Int-iff assms emeasure-restr-to-subalg null-setsD1 null-setsD2 null-setsI sets-restr-to-subalg subalgebra-def subsetD*)
then show $\text{null-sets } (\text{restr-to-subalg } M F) \subseteq \text{null-sets } M \cap \text{sets } F$ **by** *auto*
next
have $x \in \text{null-sets } (\text{restr-to-subalg } M F)$ **if** $x \in \text{null-sets } M \cap \text{sets } F$ **for** x
by (*metis that Int-iff assms null-setsD1 null-setsI sets-restr-to-subalg emeasure-restr-to-subalg[OF assms]*)
then show $\text{null-sets } M \cap \text{sets } F \subseteq \text{null-sets } (\text{restr-to-subalg } M F)$ **by** *auto*
qed

lemma *AE-restr-to-subalg*:

assumes *subalgebra* $M F$
 $AE x \text{ in } (\text{restr-to-subalg } M F). P x$
shows $AE x \text{ in } M. P x$
proof –
obtain A **where** $A: \bigwedge x. x \in \text{space } (\text{restr-to-subalg } M F) - A \implies P x$ $A \in \text{null-sets } (\text{restr-to-subalg } M F)$
using *AE-E3[OF assms(2)]* **by** *auto*
then have $A \in \text{null-sets } M$ **using** *null-sets-restr-to-subalg[OF assms(1)]* **by** *auto*
moreover have $\bigwedge x. x \in \text{space } M - A \implies P x$
using *space-restr-to-subalg A(1)* **by** *fastforce*
ultimately show *?thesis*
unfolding *eventually-ae-filter* **by** *auto*
qed

lemma *AE-restr-to-subalg2*:

assumes *subalgebra* $M F$
 $AE x \text{ in } M. P x$ **and** $[measurable]: P \in \text{measurable } F$ (*count-space UNIV*)
shows $AE x \text{ in } (\text{restr-to-subalg } M F). P x$
proof –
define U **where** $U = \{x \in \text{space } M. \neg(P x)\}$
then have $U = \{x \in \text{space } F. \neg(P x)\}$ **using** *assms(1)* **by** (*simp add:*

```

subalgebra-def)
  then have  $U \in \text{sets } F$  by simp
  then have  $U \in \text{sets } M$  using assms(1) by (meson subalgebra-def subsetD)
  then have  $U \in \text{null-sets } M$  unfolding U-def using assms(2) using AE-iff-measurable
by blast
  then have  $U \in \text{null-sets } (\text{restr-to-subalg } M F)$  using null-sets-restr-to-subalg[OF
assms(1)]  $\langle U \in \text{sets } F \rangle$  by auto
  then show ?thesis using * by (metis (no-types, lifting) Collect-mono U-def
eventually-ae-filter space-restr-to-subalg)
qed

```

```

lemma prob-space-restr-to-subalg:
  assumes subalgebra M F
           prob-space M
  shows prob-space (restr-to-subalg M F)
by (metis (no-types, lifting) assms(1) assms(2) emeasure-restr-to-subalg prob-space.emeasure-space-1
prob-spaceI
sets.top space-restr-to-subalg subalgebra-def)

```

```

lemma finite-measure-restr-to-subalg:
  assumes subalgebra M F
           finite-measure M
  shows finite-measure (restr-to-subalg M F)
by (metis (no-types, lifting) assms emeasure-restr-to-subalg finite-measure.finite-emeasure-space
finite-measureI sets.top space-restr-to-subalg subalgebra-def infinity-enreal-def)

```

```

lemma measurable-in-subalg:
  assumes subalgebra M F
            $f \in \text{measurable } F N$ 
  shows  $f \in \text{measurable } (\text{restr-to-subalg } M F) N$ 
by (metis measurable-cong-sets assms(2) sets-restr-to-subalg[OF assms(1)])

```

```

lemma measurable-in-subalg':
  assumes subalgebra M F
            $f \in \text{measurable } (\text{restr-to-subalg } M F) N$ 
  shows  $f \in \text{measurable } F N$ 
by (metis measurable-cong-sets assms(2) sets-restr-to-subalg[OF assms(1)])

```

```

lemma measurable-from-subalg:
  assumes subalgebra M F
            $f \in \text{measurable } F N$ 
  shows  $f \in \text{measurable } M N$ 
using assms unfolding measurable-def subalgebra-def by auto

```

The following is the direct transposition of `nn_integral_subalgebra` (from `Nonnegative_Lebesgue_Integration`) in the current notations, with the removal of the useless assumption $f \geq 0$.

```

lemma nn-integral-subalgebra2:
  assumes subalgebra M F and [measurable]:  $f \in \text{borel-measurable } F$ 

```

```

shows  $(\int^+ x. f x \partial(\text{restr-to-subalg } M F)) = (\int^+ x. f x \partial M)$ 
proof (rule nn-integral-subalgebra)
  show  $f \in \text{borel-measurable } (\text{restr-to-subalg } M F)$ 
    by (rule measurable-in-subalg[OF assms(1)]) simp
  show  $\text{sets } (\text{restr-to-subalg } M F) \subseteq \text{sets } M$  by (metis sets-restr-to-subalg[OF
  assms(1)] assms(1) subalgebra-def)
  fix  $A$  assume  $A \in \text{sets } (\text{restr-to-subalg } M F)$ 
  then show  $\text{emeasure } (\text{restr-to-subalg } M F) A = \text{emeasure } M A$ 
    by (metis sets-restr-to-subalg[OF assms(1)] emeasure-restr-to-subalg[OF assms(1)])
qed (auto simp add: assms space-restr-to-subalg sets-restr-to-subalg[OF assms(1)])

```

The following is the direct transposition of `integral_subalgebra` (from `Bochner_Integration`) in the current notations.

```

lemma integral-subalgebra2:
  fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second-countable-topology}\}$ 
  assumes subalgebra  $M F$  and
    [measurable]:  $f \in \text{borel-measurable } F$ 
  shows  $(\int x. f x \partial(\text{restr-to-subalg } M F)) = (\int x. f x \partial M)$ 
by (rule integral-subalgebra,
  metis measurable-in-subalg[OF assms(1)] assms(2),
  auto simp add: assms space-restr-to-subalg sets-restr-to-subalg emeasure-restr-to-subalg,
  meson assms(1) subalgebra-def subset-eq)

```

```

lemma integrable-from-subalg:
  fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second-countable-topology}\}$ 
  assumes subalgebra  $M F$ 
    integrable  $(\text{restr-to-subalg } M F) f$ 
  shows integrable  $M f$ 
proof (rule integrableI-bounded)
  have [measurable]:  $f \in \text{borel-measurable } F$  using assms by auto
  then show  $f \in \text{borel-measurable } M$  using assms(1) measurable-from-subalg by
  blast

```

```

  have  $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) = (\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial(\text{restr-to-subalg }
  M F))$ 
    by (rule nn-integral-subalgebra2[symmetric], auto simp add: assms)
  also have  $\dots < \infty$  using integrable-iff-bounded assms by auto
  finally show  $(\int^+ x. \text{ennreal } (\text{norm } (f x)) \partial M) < \infty$  by simp
qed

```

```

lemma integrable-in-subalg:
  fixes  $f :: 'a \Rightarrow 'b::\{\text{banach}, \text{second-countable-topology}\}$ 
  assumes [measurable]: subalgebra  $M F$ 
     $f \in \text{borel-measurable } F$ 
    integrable  $M f$ 
  shows integrable  $(\text{restr-to-subalg } M F) f$ 
proof (rule integrableI-bounded)
  show  $f \in \text{borel-measurable } (\text{restr-to-subalg } M F)$  using assms(2) assms(1) by
  auto

```

have $(\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial(\text{restr-to-subalg } M \ F)) = (\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial M)$
by $(\text{rule nn-integral-subalgebra2, auto simp add: assms})$
also have $\dots < \infty$ **using** $\text{integrable-iff-bounded assms}$ **by** auto
finally show $(\int^+ x. \text{ennreal } (\text{norm } (f \ x)) \ \partial(\text{restr-to-subalg } M \ F)) < \infty$ **by** simp
qed

28.2 Nonnegative conditional expectation

The conditional expectation of a function f , on a measure space M , with respect to a sub sigma algebra F , should be a function g which is F -measurable whose integral on any F -set coincides with the integral of f . Such a function is uniquely defined almost everywhere. The most direct construction is to use the measure $f dM$, restrict it to the sigma-algebra F , and apply the Radon-Nikodym theorem to write it as $g dM|_F$ for some F -measurable function g . Another classical construction for L^2 functions is done by orthogonal projection on F -measurable functions, and then extending by density to L^1 . The Radon-Nikodym point of view avoids the L^2 machinery, and works for all positive functions.

In this paragraph, we develop the definition and basic properties for nonnegative functions, as the basics of the general case. As in the definition of integrals, the nonnegative case is done with ennreal-valued functions, without any integrability assumption.

definition $\text{nn-cond-exp} :: 'a \text{ measure} \Rightarrow 'a \text{ measure} \Rightarrow ('a \Rightarrow \text{ennreal}) \Rightarrow ('a \Rightarrow \text{ennreal})$

where

$\text{nn-cond-exp } M \ F \ f =$
 $(\text{if } f \in \text{borel-measurable } M \ \wedge \ \text{subalgebra } M \ F$
 $\text{then RN-deriv } (\text{restr-to-subalg } M \ F) \ (\text{restr-to-subalg } (\text{density } M \ f) \ F)$
 $\text{else } (\lambda \cdot. 0))$

lemma

shows $\text{borel-measurable-nn-cond-exp} [\text{measurable}]: \text{nn-cond-exp } M \ F \ f \in \text{borel-measurable } F$

and $\text{borel-measurable-nn-cond-exp2} [\text{measurable}]: \text{nn-cond-exp } M \ F \ f \in \text{borel-measurable } M$

by $(\text{simp-all add: nn-cond-exp-def})$

$(\text{metis borel-measurable-RN-deriv borel-measurable-subalgebra sets-restr-to-subalg space-restr-to-subalg subalgebra-def})$

The good setting for conditional expectations is the situation where the subalgebra F gives rise to a sigma-finite measure space. To see what goes wrong if it is not sigma-finite, think of \mathbb{R} with the trivial sigma-algebra $\{\emptyset, \mathbb{R}\}$. In this case, conditional expectations have to be constant functions, so they have integral 0 or ∞ . This means that a positive integrable function can have no meaningful conditional expectation.


```

locale sigma-finite-subalgebra =
  fixes  $M F :: 'a \text{ measure}$ 
  assumes subalg: subalgebra  $M F$ 
  and sigma-fin-subalg: sigma-finite-measure (restr-to-subalg  $M F$ )

lemma sigma-finite-subalgebra-is-sigma-finite:
  assumes sigma-finite-subalgebra  $M F$ 
  shows sigma-finite-measure  $M$ 
proof
  have subalg: subalgebra  $M F$ 
  and sigma-fin-subalg: sigma-finite-measure (restr-to-subalg  $M F$ )
  using assms unfolding sigma-finite-subalgebra-def by auto
  obtain  $A$  where  $Ap$ : countable  $A \wedge A \subseteq \text{sets } (\text{restr-to-subalg } M F) \wedge \bigcup A =$ 
space (restr-to-subalg  $M F$ )  $\wedge (\forall a \in A. \text{emeasure } (\text{restr-to-subalg } M F) a \neq \infty)$ 
  using sigma-finite-measure.sigma-finite-countable[OF sigma-fin-subalg] by fast-
force
  have  $A \subseteq \text{sets } F$  using  $Ap$  sets-restr-to-subalg[OF subalg] by fastforce
  then have  $A \subseteq \text{sets } M$  using subalg subalgebra-def by force
  moreover have  $\bigcup A = \text{space } M$  using  $Ap$  space-restr-to-subalg by simp
  moreover have  $\forall a \in A. \text{emeasure } M a \neq \infty$  by (metis subsetD emeasure-restr-to-subalg[OF
subalg]  $\langle A \subseteq \text{sets } F \rangle Ap$ )
  ultimately show  $\exists A. \text{countable } A \wedge A \subseteq \text{sets } M \wedge \bigcup A = \text{space } M \wedge (\forall a \in A. \text{emeasure } M a \neq \infty)$ 
using  $Ap$  by auto
qed

sublocale sigma-finite-subalgebra  $\subseteq$  sigma-finite-measure
using sigma-finite-subalgebra-is-sigma-finite sigma-finite-subalgebra-axioms by blast

Conditional expectations are very often used in probability spaces. This is
a special case of the previous one, as we prove now.

locale finite-measure-subalgebra = finite-measure +
  fixes  $F :: 'a \text{ measure}$ 
  assumes subalg: subalgebra  $M F$ 

lemma finite-measure-subalgebra-is-sigma-finite:
  assumes finite-measure-subalgebra  $M F$ 
  shows sigma-finite-subalgebra  $M F$ 
proof –
  interpret finite-measure-subalgebra  $M F$  using assms by simp
  have finite-measure (restr-to-subalg  $M F$ )
  using finite-measure-restr-to-subalg subalg finite-emeasure-space finite-measureI
unfolding infinity-ennreal-def by blast
  then have sigma-finite-measure (restr-to-subalg  $M F$ )
  unfolding finite-measure-def by simp
  then show sigma-finite-subalgebra  $M F$  unfolding sigma-finite-subalgebra-def
using subalg by simp
qed

sublocale finite-measure-subalgebra  $\subseteq$  sigma-finite-subalgebra

```

proof –

have *finite-measure* (*restr-to-subalg* M F)
 using *finite-measure-restr-to-subalg* *subalg* *finite-measure-space* *finite-measureI*
unfolding *infinity-enreal-def* **by** *blast*
 then have *sigma-finite-measure* (*restr-to-subalg* M F)
 unfolding *finite-measure-def* **by** *simp*
 then show *sigma-finite-subalgebra* M F **unfolding** *sigma-finite-subalgebra-def*
using *subalg* **by** *simp*
qed

context *sigma-finite-subalgebra*
begin

The next lemma is arguably the most fundamental property of conditional expectation: when computing an expectation against an F -measurable function, it is equivalent to work with a function or with its F -conditional expectation.

This property (even for bounded test functions) characterizes conditional expectations, as the second lemma below shows. From this point on, we will only work with it, and forget completely about the definition using Radon-Nikodym derivatives.

lemma *nn-cond-exp-intg*:

assumes [*measurable*]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$
 shows $(\int^+ x. f \, x * \text{nn-cond-exp } M \, F \, g \, x \, \partial M) = (\int^+ x. f \, x * g \, x \, \partial M)$
proof –
 have [*measurable*]: $f \in \text{borel-measurable } M$
 by (*meson* *assms* *subalg* *borel-measurable-subalgebra* *subalgebra-def*)
 have *ac*: *absolutely-continuous* (*restr-to-subalg* M F) (*restr-to-subalg* (*density* M g) F)
 unfolding *absolutely-continuous-def*
 proof –
 have *null-sets* (*restr-to-subalg* M F) = *null-sets* $M \cap \text{sets } F$ **by** (*rule* *null-sets-restr-to-subalg*[*OF* *subalg*])
 moreover have *null-sets* $M \subseteq \text{null-sets } (\text{density } M \, g)$
 by (*rule* *absolutely-continuousI-density*[*unfolded* *absolutely-continuous-def*])
 auto
 ultimately have *null-sets* (*restr-to-subalg* M F) $\subseteq \text{null-sets } (\text{density } M \, g) \cap \text{sets } F$ **by** *auto*
 moreover have *null-sets* (*density* $M \, g$) $\cap \text{sets } F = \text{null-sets } (\text{restr-to-subalg } (\text{density } M \, g) \, F)$
 by (*rule* *null-sets-restr-to-subalg*[*symmetric*]) (*metis* *subalg* *sets-density* *space-density* *subalgebra-def*)
 ultimately show *null-sets* (*restr-to-subalg* M F) $\subseteq \text{null-sets } (\text{restr-to-subalg } (\text{density } M \, g) \, F)$ **by** *auto*
 qed

have $(\int^+ x. f \, x * \text{nn-cond-exp } M \, F \, g \, x \, \partial M) = (\int^+ x. f \, x * \text{nn-cond-exp } M \, F \, g \, x \, \partial(\text{restr-to-subalg } M \, F))$

```

    by (rule nn-integral-subalgebra2[symmetric]) (simp-all add: assms subalg)
    also have ... = ( $\int^+ x. f x * RN\text{-deriv (restr-to-subalg M F) (restr-to-subalg (density M g) F) x \partial(restr-to-subalg M F))$ )
    unfolding nn-cond-exp-def using assms subalg by simp
    also have ... = ( $\int^+ x. RN\text{-deriv (restr-to-subalg M F) (restr-to-subalg (density M g) F) x * f x \partial(restr-to-subalg M F)$ )
    by (simp add: mult.commute)
    also have ... = ( $\int^+ x. f x \partial(restr-to-subalg (density M g) F)$ )
    proof (rule sigma-finite-measure.RN-deriv-nn-integral[symmetric])
      show sets (restr-to-subalg (density M g) F) = sets (restr-to-subalg M F)
      by (metis subalg restr-to-subalg-def sets.sets-measure-of-eq space-density subalgebra-def)
    qed (auto simp add: assms measurable-restrict ac measurable-in-subalg subalg sigma-fin-subalg)
    also have ... = ( $\int^+ x. f x \partial(density M g)$ )
    by (metis nn-integral-subalgebra2 subalg assms(1) sets-density space-density subalgebra-def)
    also have ... = ( $\int^+ x. g x * f x \partial M$ )
    by (rule nn-integral-density) (simp-all add: assms)
    also have ... = ( $\int^+ x. f x * g x \partial M$ )
    by (simp add: mult.commute)
    finally show ?thesis by simp
  qed

```

lemma *nn-cond-exp-charact*:

```

  assumes  $\bigwedge A. A \in \text{sets } F \implies (\int^+ x \in A. f x \partial M) = (\int^+ x \in A. g x \partial M)$  and
    [measurable]:  $f \in \text{borel-measurable } M \text{ } g \in \text{borel-measurable } F$ 
  shows  $\text{AE } x \text{ in } M. g x = \text{nn-cond-exp } M F f x$ 
  proof -
    let ?MF = restr-to-subalg M F
    {
      fix A assume  $A \in \text{sets } ?MF$ 
      then have [measurable]:  $A \in \text{sets } F$  using sets-restr-to-subalg[OF subalg] by
    simp
    have  $(\int^+ x \in A. g x \partial ?MF) = (\int^+ x \in A. g x \partial M)$ 
    by (simp add: nn-integral-subalgebra2 subalg)
    also have ... =  $(\int^+ x \in A. f x \partial M)$  using assms(1) by simp
    also have ... =  $(\int^+ x. \text{indicator } A x * f x \partial M)$  by (simp add: mult.commute)
    also have ... =  $(\int^+ x. \text{indicator } A x * \text{nn-cond-exp } M F f x \partial M)$ 
    by (rule nn-cond-exp-intg[symmetric]) (auto simp add: assms)
    also have ... =  $(\int^+ x \in A. \text{nn-cond-exp } M F f x \partial M)$  by (simp add:
  mult.commute)
    also have ... =  $(\int^+ x \in A. \text{nn-cond-exp } M F f x \partial ?MF)$ 
    by (simp add: nn-integral-subalgebra2 subalg)
    finally have  $(\int^+ x \in A. g x \partial ?MF) = (\int^+ x \in A. \text{nn-cond-exp } M F f x \partial ?MF)$  by simp
    } note * = this
  have  $\text{AE } x \text{ in } ?MF. g x = \text{nn-cond-exp } M F f x$ 
  by (rule sigma-finite-measure.density-unique2)

```

(*auto simp add: assms subalg sigma-fin-subalg AE-restr-to-subalg2 **)
then show ?thesis **using** AE-restr-to-subalg[OF subalg] **by** simp
qed

lemma nn-cond-exp-F-meas:

assumes $f \in \text{borel-measurable } F$
shows $AE\ x\ in\ M. f\ x = nn\text{-}cond\text{-}exp\ M\ F\ f\ x$
by (rule nn-cond-exp-charact) (*auto simp add: assms measurable-from-subalg[OF subalg]*)

lemma nn-cond-exp-prod:

assumes [measurable]: $f \in \text{borel-measurable } F\ g \in \text{borel-measurable } M$
shows $AE\ x\ in\ M. f\ x * nn\text{-}cond\text{-}exp\ M\ F\ g\ x = nn\text{-}cond\text{-}exp\ M\ F\ (\lambda x. f\ x * g\ x)\ x$
proof (rule nn-cond-exp-charact)
have [measurable]: $f \in \text{borel-measurable } M$ **by** (rule measurable-from-subalg[OF subalg assms(1)])
show $(\lambda x. f\ x * g\ x) \in \text{borel-measurable } M$ **by** measurable

fix A **assume** $A \in \text{sets } F$

then have [measurable]: $(\lambda x. f\ x * \text{indicator } A\ x) \in \text{borel-measurable } F$ **by** measurable

have $(\int^{+x \in A. (f\ x * g\ x)\ \partial M}) = \int^{+x. (f\ x * \text{indicator } A\ x) * g\ x\ \partial M}$
by (simp add: mult.commute mult.left-commute)

also have $\dots = \int^{+x. (f\ x * \text{indicator } A\ x) * nn\text{-}cond\text{-}exp\ M\ F\ g\ x\ \partial M}$
by (rule nn-cond-exp-intg[symmetric]) (*auto simp add: assms*)

also have $\dots = (\int^{+x \in A. (f\ x * nn\text{-}cond\text{-}exp\ M\ F\ g\ x)\ \partial M})$
by (simp add: mult.commute mult.left-commute)

finally show $(\int^{+x \in A. (f\ x * g\ x)\ \partial M}) = (\int^{+x \in A. (f\ x * nn\text{-}cond\text{-}exp\ M\ F\ g\ x)\ \partial M})$ **by** simp

qed (*auto simp add: assms*)

lemma nn-cond-exp-sum:

assumes [measurable]: $f \in \text{borel-measurable } M\ g \in \text{borel-measurable } M$
shows $AE\ x\ in\ M. nn\text{-}cond\text{-}exp\ M\ F\ f\ x + nn\text{-}cond\text{-}exp\ M\ F\ g\ x = nn\text{-}cond\text{-}exp\ M\ F\ (\lambda x. f\ x + g\ x)\ x$

proof (rule nn-cond-exp-charact)

fix A **assume** [measurable]: $A \in \text{sets } F$

then have $A \in \text{sets } M$ **by** (meson subalg subalgebra-def subsetD)

have $(\int^{+x \in A. (nn\text{-}cond\text{-}exp\ M\ F\ f\ x + nn\text{-}cond\text{-}exp\ M\ F\ g\ x)\ \partial M}) = (\int^{+x \in A. nn\text{-}cond\text{-}exp\ M\ F\ f\ x\ \partial M}) + (\int^{+x \in A. nn\text{-}cond\text{-}exp\ M\ F\ g\ x\ \partial M})$

by (rule nn-set-integral-add) (*auto simp add: assms $\langle A \in \text{sets } M \rangle$*)

also have $\dots = (\int^{+x. \text{indicator } A\ x * nn\text{-}cond\text{-}exp\ M\ F\ f\ x\ \partial M}) + (\int^{+x. \text{indicator } A\ x * nn\text{-}cond\text{-}exp\ M\ F\ g\ x\ \partial M})$

by (metis (no-types, lifting) mult.commute nn-integral-cong)

also have $\dots = (\int^{+x. \text{indicator } A\ x * f\ x\ \partial M}) + (\int^{+x. \text{indicator } A\ x * g\ x\ \partial M})$

by (simp add: nn-cond-exp-intg)

also have $\dots = (\int^{+x \in A. f\ x\ \partial M}) + (\int^{+x \in A. g\ x\ \partial M})$

by (metis (no-types, lifting) mult.commute nn-integral-cong)

also have ... = $(\int^{+x \in A. (f\ x + g\ x) \partial M})$
by (rule nn-set-integral-add[symmetric]) (auto simp add: assms $\langle A \in \text{sets } M \rangle$)
finally show $(\int^{+x \in A. (f\ x + g\ x) \partial M}) = (\int^{+x \in A. (nn\text{-cond-exp } M\ F\ f\ x + nn\text{-cond-exp } M\ F\ g\ x) \partial M})$
by simp
qed (auto simp add: assms)

lemma nn-cond-exp-cong:

assumes $AE\ x\ in\ M. f\ x = g\ x$
and [measurable]: $f \in \text{borel-measurable } M\ g \in \text{borel-measurable } M$
shows $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ f\ x = nn\text{-cond-exp } M\ F\ g\ x$
proof (rule nn-cond-exp-charact)
fix A **assume** [measurable]: $A \in \text{sets } F$
have $(\int^{+x \in A. nn\text{-cond-exp } M\ F\ f\ x\ \partial M}) = \int^{+x. \text{indicator } A\ x * nn\text{-cond-exp } M\ F\ f\ x\ \partial M}$
by (simp add: mult.commute)
also have ... = $\int^{+x. \text{indicator } A\ x * f\ x\ \partial M}$ **by** (simp add: nn-cond-exp-intg assms)
also have ... = $(\int^{+x \in A. f\ x\ \partial M})$ **by** (simp add: mult.commute)
also have ... = $(\int^{+x \in A. g\ x\ \partial M})$ **by** (rule nn-set-integral-cong[OF assms(1)])
finally show $(\int^{+x \in A. g\ x\ \partial M}) = (\int^{+x \in A. nn\text{-cond-exp } M\ F\ f\ x\ \partial M})$ **by** simp
qed (auto simp add: assms)

lemma nn-cond-exp-mono:

assumes $AE\ x\ in\ M. f\ x \leq g\ x$
and [measurable]: $f \in \text{borel-measurable } M\ g \in \text{borel-measurable } M$
shows $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ f\ x \leq nn\text{-cond-exp } M\ F\ g\ x$
proof –
define h **where** $h = (\lambda x. g\ x - f\ x)$
have [measurable]: $h \in \text{borel-measurable } M$ **unfolding** $h\text{-def}$ **by** simp
have *: $AE\ x\ in\ M. g\ x = f\ x + h\ x$ **unfolding** $h\text{-def}$ **using** assms(1) **by** (auto simp: ennreal-ineq-diff-add)
have $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ g\ x = nn\text{-cond-exp } M\ F\ (\lambda x. f\ x + h\ x)\ x$
by (rule nn-cond-exp-cong) (auto simp add: * assms)
moreover have $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ f\ x + nn\text{-cond-exp } M\ F\ h\ x = nn\text{-cond-exp } M\ F\ (\lambda x. f\ x + h\ x)\ x$
by (rule nn-cond-exp-sum) (auto simp add: assms)
ultimately have $AE\ x\ in\ M. nn\text{-cond-exp } M\ F\ g\ x = nn\text{-cond-exp } M\ F\ f\ x + nn\text{-cond-exp } M\ F\ h\ x$ **by** auto
then show ?thesis **by** force
qed

lemma nested-subalg-is-sigma-finite:

assumes subalgebra $M\ G$ subalgebra $G\ F$
shows sigma-finite-subalgebra $M\ G$
unfolding sigma-finite-subalgebra-def
proof (auto simp add: assms)
have $\exists A. \text{countable } A \wedge A \subseteq \text{sets } (restr\text{-to-subalg } M\ F) \wedge \bigcup A = \text{space } (restr\text{-to-subalg } M\ F) \wedge (\forall a \in A. \text{emeasure } (restr\text{-to-subalg } M\ F)\ a \neq \infty)$

```

using sigma-fin-subalg sigma-finite-measure-def by auto
then obtain A where A:countable A  $\wedge$  A  $\subseteq$  sets (restr-to-subalg M F)  $\wedge$   $\bigcup A$ 
= space (restr-to-subalg M F)  $\wedge$  ( $\forall a \in A. \text{emeasure (restr-to-subalg M F) } a \neq \infty$ )
by auto
have sets F  $\subseteq$  sets M
by (meson assms order-trans subalgebra-def)
then have countable A  $\wedge$  A  $\subseteq$  sets (restr-to-subalg M G)  $\wedge$   $\bigcup A$  = space
(restr-to-subalg M F)  $\wedge$  ( $\forall a \in A. \text{emeasure (restr-to-subalg M G) } a \neq \infty$ )
by (metis (no-types) A assms basic-trans-rules(31) emeasure-restr-to-subalg
order-trans sets-restr-to-subalg subalgebra-def)
then show sigma-finite-measure (restr-to-subalg M G)
by (metis sigma-finite-measure.intro space-restr-to-subalg)
qed

```

```

lemma nn-cond-exp-nested-subalg:
assumes subalgebra M G subalgebra G F
and [measurable]: f  $\in$  borel-measurable M
shows AE x in M. nn-cond-exp M F f x = nn-cond-exp M F (nn-cond-exp M G
f) x
proof (rule nn-cond-exp-charact, auto)
interpret G: sigma-finite-subalgebra M G by (rule nested-subalg-is-sigma-finite[OF
assms(1) assms(2)])
fix A assume [measurable]: A  $\in$  sets F
then have [measurable]: A  $\in$  sets G using assms(2) by (meson subsetD subal-
gebra-def)

have set-nn-integral M A (nn-cond-exp M G f) = ( $\int^+ x. \text{indicator } A \ x \ *$ 
nn-cond-exp M G f x  $\partial M$ )
by (metis (no-types) mult.commute)
also have ... = ( $\int^+ x. \text{indicator } A \ x \ * \ f \ x \ \partial M$ ) by (rule G.nn-cond-exp-intg,
auto simp add: assms)
also have ... = ( $\int^+ x. \text{indicator } A \ x \ * \ \text{nn-cond-exp M F f } x \ \partial M$ ) by (rule
nn-cond-exp-intg[symmetric], auto simp add: assms)
also have ... = set-nn-integral M A (nn-cond-exp M F f) by (metis (no-types)
mult.commute)
finally show set-nn-integral M A (nn-cond-exp M G f) = set-nn-integral M A
(nn-cond-exp M F f).
qed

end

```

28.3 Real conditional expectation

Once conditional expectations of positive functions are defined, the definition for real-valued functions follows readily, by taking the difference of positive and negative parts. One could also define a conditional expectation of vector-space valued functions, as in `Bochner_Integral`, but since the real-valued case is the most important, and quicker to formalize, I con-

centrate on it. (It is also essential for the case of the most general Pettis integral.)

definition *real-cond-exp* :: 'a measure \Rightarrow 'a measure \Rightarrow ('a \Rightarrow real) \Rightarrow ('a \Rightarrow real)
where
real-cond-exp $M F f =$
 $(\lambda x. \text{enn2real}(\text{nn-cond-exp } M F (\lambda x. \text{ennreal } (f x)) x) - \text{enn2real}(\text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-f x)) x))$

lemma

shows *borel-measurable-cond-exp* [*measurable*]: *real-cond-exp* $M F f \in \text{borel-measurable } F$

and *borel-measurable-cond-exp2* [*measurable*]: *real-cond-exp* $M F f \in \text{borel-measurable } M$

unfolding *real-cond-exp-def* **by** *auto*

context *sigma-finite-subalgebra*

begin

lemma *real-cond-exp-abs*:

assumes [*measurable*]: $f \in \text{borel-measurable } M$

shows $\text{AE } x \text{ in } M. \text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (\text{abs}(f x))) x$

proof –

define *fp* **where** $fp = (\lambda x. \text{ennreal } (f x))$

define *fm* **where** $fm = (\lambda x. \text{ennreal } (-f x))$

have [*measurable*]: $fp \in \text{borel-measurable } M$ $fm \in \text{borel-measurable } M$ **unfolding** *fp-def fm-def* **by** *auto*

have $eq: \bigwedge x. \text{ennreal } |f x| = fp x + fm x$ **unfolding** *fp-def fm-def* **by** (*simp add: abs-real-def ennreal-neg*)

{
fix x **assume** $H: \text{nn-cond-exp } M F fp x + \text{nn-cond-exp } M F fm x = \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$
have $|\text{real-cond-exp } M F f x| \leq |\text{enn2real}(\text{nn-cond-exp } M F fp x)| + |\text{enn2real}(\text{nn-cond-exp } M F fm x)|$
unfolding *real-cond-exp-def fp-def fm-def* **by** (*auto intro: abs-triangle-ineq4 simp del: enn2real-nonneg*)
from *ennreal-leI[OF this]*
have $\text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F fp x + \text{nn-cond-exp } M F fm x$
by *simp (metis add.commute ennreal-enn2real le-iff-add not-le top-unique)*
also have $\dots = \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$ **using** H **by** *simp*
finally have $\text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$
by *simp*
}
moreover have $\text{AE } x \text{ in } M. \text{nn-cond-exp } M F fp x + \text{nn-cond-exp } M F fm x = \text{nn-cond-exp } M F (\lambda x. fp x + fm x) x$
by (*rule nn-cond-exp-sum*) (*auto simp add: fp-def fm-def*)
ultimately have $\text{AE } x \text{ in } M. \text{abs}(\text{real-cond-exp } M F f x) \leq \text{nn-cond-exp } M F$

```

( $\lambda x. fp\ x + fm\ x$ )  $x$ 
  by auto
  then show ?thesis using eq by simp
qed

```

The next lemma shows that the conditional expectation is an F -measurable function whose average against an F -measurable function f coincides with the average of the original function against f . It is obtained as a consequence of the same property for the positive conditional expectation, taking the difference of the positive and the negative part. The proof is given first assuming $f \geq 0$ for simplicity, and then extended to the general case in the subsequent lemma. The idea of the proof is essentially trivial, but the implementation is slightly tedious as one should check all the integrability properties of the different parts, and go back and forth between positive integral and signed integrals, and between real-valued functions and ennreal-valued functions.

Once this lemma is available, we will use it to characterize the conditional expectation, and never come back to the original technical definition, as we did in the case of the nonnegative conditional expectation.

lemma *real-cond-exp-intg-fpos*:

```

assumes integrable  $M$  ( $\lambda x. f\ x * g\ x$ ) and f-pos[simp]:  $\bigwedge x. f\ x \geq 0$  and
  [measurable]:  $f \in \text{borel-measurable } F$   $g \in \text{borel-measurable } M$ 
shows integrable  $M$  ( $\lambda x. f\ x * \text{real-cond-exp } M\ F\ g\ x$ )
  ( $\int x. f\ x * \text{real-cond-exp } M\ F\ g\ x\ \partial M$ ) = ( $\int x. f\ x * g\ x\ \partial M$ )
proof –
  have [measurable]:  $f \in \text{borel-measurable } M$  by (rule measurable-from-subalg[OF subalg assms(3)])
  define gp where  $gp = (\lambda x. \text{ennreal } (g\ x))$ 
  define gm where  $gm = (\lambda x. \text{ennreal } (-g\ x))$ 
  have [measurable]:  $gp \in \text{borel-measurable } M$   $gm \in \text{borel-measurable } M$  unfolding
gp-def gm-def by auto
  define h where  $h = (\lambda x. \text{ennreal}(\text{abs}(g\ x)))$ 
  have hgpgm:  $\bigwedge x. h\ x = gp\ x + gm\ x$  unfolding gp-def gm-def h-def by (simp add: abs-real-def ennreal-neg)
  have [measurable]:  $h \in \text{borel-measurable } M$  unfolding h-def by simp
  have pos[simp]:  $\bigwedge x. h\ x \geq 0 \ \bigwedge x. gp\ x \geq 0 \ \bigwedge x. gm\ x \geq 0$  unfolding h-def gp-def gm-def by simp-all
  have gp-real:  $\bigwedge x. \text{enn2real}(gp\ x) = \max(g\ x)\ 0$ 
    unfolding gp-def by (simp add: max-def ennreal-neg)
  have gm-real:  $\bigwedge x. \text{enn2real}(gm\ x) = \max(-g\ x)\ 0$ 
    unfolding gm-def by (simp add: max-def ennreal-neg)

  have ( $\int^+ x. \text{norm}(f\ x * \max(g\ x)\ 0)\ \partial M$ )  $\leq$  ( $\int^+ x. \text{norm}(f\ x * g\ x)\ \partial M$ )
    by (simp add: nn-integral-mono)
  also have ...  $< \infty$  using assms(1) by (simp add: integrable-iff-bounded)
  finally have ( $\int^+ x. \text{norm}(f\ x * \max(g\ x)\ 0)\ \partial M$ )  $< \infty$  by simp
  then have int1: integrable  $M$  ( $\lambda x. f\ x * \max(g\ x)\ 0$ ) by (simp add: inte-

```


grableI-bounded)

have $(\int^+ x. \text{norm}(f x * \max(-g x) 0) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$
by (*simp add: nn-integral-mono*)
also have $\dots < \infty$ **using** *assms(1)* **by** (*simp add: integrable-iff-bounded*)
finally have $(\int^+ x. \text{norm}(f x * \max(-g x) 0) \partial M) < \infty$ **by** *simp*
then have *int2: integrable M* $(\lambda x. f x * \max(-g x) 0)$ **by** (*simp add: integrableI-bounded*)

have $(\int^+ x. f x * \text{nn-cond-exp } M F h x \partial M) = (\int^+ x. f x * h x \partial M)$
by (*rule nn-cond-exp-intg auto*)
also have $\dots = \int^+ x. \text{ennreal}(f x * \max(g x) 0 + f x * \max(-g x) 0) \partial M$
unfolding *h-def*
by (*intro nn-integral-cong*)(*auto simp: ennreal-mult[symmetric] abs-mult split: split-max*)
also have $\dots < \infty$
using *Bochner-Integration.integrable-add[OF int1 int2, THEN integrableD(2)]*
by (*auto simp add: less-top*)
finally have $\ast: (\int^+ x. f x * \text{nn-cond-exp } M F h x \partial M) < \infty$ **by** *simp*

have $(\int^+ x. \text{norm}(f x * \text{real-cond-exp } M F g x) \partial M) = (\int^+ x. f x * \text{abs}(\text{real-cond-exp } M F g x) \partial M)$
by (*simp add: abs-mult*)
also have $\dots \leq (\int^+ x. f x * \text{nn-cond-exp } M F h x \partial M)$
proof (*rule nn-integral-mono-AE*)
 $\{$
fix *x* **assume** $\ast: \text{abs}(\text{real-cond-exp } M F g x) \leq \text{nn-cond-exp } M F h x$
have $\text{ennreal}(f x * |\text{real-cond-exp } M F g x|) = f x * \text{ennreal}(|\text{real-cond-exp } M F g x|)$
by (*simp add: ennreal-mult*)
also have $\dots \leq f x * \text{nn-cond-exp } M F h x$
using \ast **by** (*auto intro!: mult-left-mono*)
finally have $\text{ennreal}(f x * |\text{real-cond-exp } M F g x|) \leq f x * \text{nn-cond-exp } M F h x$
by *simp*
 $\}$
then show *AE x in M. ennreal (f x * |real-cond-exp M F g x|) ≤ f x * nn-cond-exp M F h x*
using *real-cond-exp-abs[OF assms(4)] h-def* **by** *auto*
qed
finally have $\ast: (\int^+ x. \text{norm}(f x * \text{real-cond-exp } M F g x) \partial M) < \infty$ **using** \ast
by *auto*
show *integrable M* $(\lambda x. f x * \text{real-cond-exp } M F g x)$
using \ast **by** (*intro integrableI-bounded*) *auto*

have $(\int^+ x. f x * \text{nn-cond-exp } M F g x \partial M) \leq (\int^+ x. f x * \text{nn-cond-exp } M F h x \partial M)$
proof (*rule nn-integral-mono-AE*)

```

have AE x in M. nn-cond-exp M F gp x ≤ nn-cond-exp M F h x
  by (rule nn-cond-exp-mono) (auto simp add: hgpgm)
then show AE x in M. f x * nn-cond-exp M F gp x ≤ f x * nn-cond-exp M F
h x
  by (auto simp: mult-left-mono)
qed
then have a: (∫+ x. f x * nn-cond-exp M F gp x ∂M) < ∞
  using * by auto
have ennreal(norm(f x * enn2real(nn-cond-exp M F gp x))) ≤ f x * nn-cond-exp
M F gp x for x
  by (auto simp add: ennreal-mult intro!: mult-left-mono)
  (metis enn2real-ennreal enn2real-nonneg le-cases le-ennreal-iff)
then have (∫+ x. norm(f x * enn2real(nn-cond-exp M F gp x)) ∂M) ≤ (∫+ x. f
x * nn-cond-exp M F gp x ∂M)
  by (simp add: nn-integral-mono)
then have (∫+ x. norm(f x * enn2real(nn-cond-exp M F gp x)) ∂M) < ∞ using
a by auto
then have gp-int: integrable M (λx. f x * enn2real(nn-cond-exp M F gp x)) by
(simp add: integrableI-bounded)
have gp-fin: AE x in M. f x * nn-cond-exp M F gp x ≠ ∞
  apply (rule nn-integral-PInf-AE) using a by auto

have (∫ x. f x * enn2real(nn-cond-exp M F gp x) ∂M) = enn2real (∫+ x. f x *
enn2real(nn-cond-exp M F gp x) ∂M)
  by (rule integral-eq-nn-integral) auto
also have ... = enn2real(∫+ x. ennreal(f x * enn2real(gp x)) ∂M)
proof -
{
  fix x assume f x * nn-cond-exp M F gp x ≠ ∞
  then have ennreal (f x * enn2real (nn-cond-exp M F gp x)) = ennreal (f x
* nn-cond-exp M F gp x
    by (auto simp add: ennreal-mult ennreal-mult-eq-top-iff less-top intro!:
ennreal-mult-left-cong)
}
then have AE x in M. ennreal (f x * enn2real (nn-cond-exp M F gp x)) =
ennreal (f x) * nn-cond-exp M F gp x
  using gp-fin by auto
then have (∫+ x. f x * enn2real(nn-cond-exp M F gp x) ∂M) = (∫+ x. f x *
nn-cond-exp M F gp x ∂M)
  by (rule nn-integral-cong-AE)
also have ... = (∫+ x. f x * gp x ∂M)
  by (rule nn-cond-exp-intg) (auto simp add: gp-def)
also have ... = (∫+ x. ennreal(f x * enn2real(gp x)) ∂M)
  by (rule nn-integral-cong-AE) (auto simp: ennreal-mult gp-def)
finally have (∫+ x. f x * enn2real(nn-cond-exp M F gp x) ∂M) = (∫+ x.
ennreal(f x * enn2real(gp x)) ∂M) by simp
then show ?thesis by simp
qed
also have ... = (∫ x. f x * enn2real(gp x) ∂M)

```

```

    by (rule integral-eq-nn-integral[symmetric]) (auto simp add: gp-def)
    finally have gp-expr:  $(\int x. f x * enn2real(nn-cond-exp M F gp x) \partial M) = (\int x. f x * enn2real(gp x) \partial M)$  by simp

    have  $(\int^+ x. f x * nn-cond-exp M F gm x \partial M) \leq (\int^+ x. f x * nn-cond-exp M F h x \partial M)$ 
    proof (rule nn-integral-mono-AE)
      have AE x in M.  $nn-cond-exp M F gm x \leq nn-cond-exp M F h x$ 
      by (rule nn-cond-exp-mono) (auto simp add: hpggm)
      then show AE x in M.  $f x * nn-cond-exp M F gm x \leq f x * nn-cond-exp M F h x$ 
    proof
      by (auto simp: mult-left-mono)
    qed
    then have a:  $(\int^+ x. f x * nn-cond-exp M F gm x \partial M) < \infty$ 
    using * by auto
    have  $\bigwedge x. ennreal(norm(f x * enn2real(nn-cond-exp M F gm x))) \leq f x * nn-cond-exp M F gm x$ 
    by (auto simp add: ennreal-mult intro!: mult-left-mono)
    (metis enn2real-ennreal enn2real-nonneg le-cases le-ennreal-iff)
    then have  $(\int^+ x. norm(f x * enn2real(nn-cond-exp M F gm x)) \partial M) \leq (\int^+ x. f x * nn-cond-exp M F gm x \partial M)$ 
    by (simp add: nn-integral-mono)
    then have  $(\int^+ x. norm(f x * enn2real(nn-cond-exp M F gm x)) \partial M) < \infty$  using a by auto
    then have gm-int: integrable M  $(\lambda x. f x * enn2real(nn-cond-exp M F gm x))$  by (simp add: integrableI-bounded)
    have gm-fin: AE x in M.  $f x * nn-cond-exp M F gm x \neq \infty$ 
    apply (rule nn-integral-PInf-AE) using a by auto

    have  $(\int x. f x * enn2real(nn-cond-exp M F gm x) \partial M) = enn2real(\int^+ x. f x * enn2real(nn-cond-exp M F gm x) \partial M)$ 
    by (rule integral-eq-nn-integral) auto
    also have  $\dots = enn2real(\int^+ x. ennreal(f x * enn2real(gm x)) \partial M)$ 
    proof -
      {
        fix x assume  $f x * nn-cond-exp M F gm x \neq \infty$ 
        then have  $ennreal(f x * enn2real(nn-cond-exp M F gm x)) = ennreal(f x * nn-cond-exp M F gm x)$ 
        by (auto simp add: ennreal-mult ennreal-mult-eq-top-iff less-top intro!: ennreal-mult-left-cong)
      }
      then have AE x in M.  $ennreal(f x * enn2real(nn-cond-exp M F gm x)) = ennreal(f x * nn-cond-exp M F gm x)$ 
      using gm-fin by auto
      then have  $(\int^+ x. f x * enn2real(nn-cond-exp M F gm x) \partial M) = (\int^+ x. f x * nn-cond-exp M F gm x \partial M)$ 
      by (rule nn-integral-cong-AE)
      also have  $\dots = (\int^+ x. f x * gm x \partial M)$ 

```

by (rule nn-cond-exp-intg) (auto simp add: gm-def)
 also have $\dots = (\int^+ x. \text{ennreal}(f x * \text{enn2real}(g x)) \partial M)$
 by (rule nn-integral-cong-AE) (auto simp: ennreal-mult gm-def)
 finally have $(\int^+ x. f x * \text{enn2real}(\text{nn-cond-exp } M F g x) \partial M) = (\int^+ x. \text{ennreal}(f x * \text{enn2real}(g x)) \partial M)$ by simp
 then show ?thesis by simp
 qed
 also have $\dots = (\int x. f x * \text{enn2real}(g x) \partial M)$
 by (rule integral-eq-nn-integral[symmetric]) (auto simp add: gm-def)
 finally have gm-expr: $(\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F g x) \partial M) = (\int x. f x * \text{enn2real}(g x) \partial M)$ by simp

have $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F g x) - f x * \text{enn2real}(\text{nn-cond-exp } M F g x) \partial M)$
 unfolding real-cond-exp-def gp-def gm-def by (simp add: right-diff-distrib)
 also have $\dots = (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F g x) \partial M) - (\int x. f x * \text{enn2real}(\text{nn-cond-exp } M F g x) \partial M)$
 by (rule Bochner-Integration.integral-diff) (simp-all add: gp-int gm-int)
 also have $\dots = (\int x. f x * \text{enn2real}(g x) \partial M) - (\int x. f x * \text{enn2real}(g x) \partial M)$
 using gp-expr gm-expr by simp
 also have $\dots = (\int x. f x * \max(g x) 0 \partial M) - (\int x. f x * \max(-g x) 0 \partial M)$
 using gp-real gm-real by simp
 also have $\dots = (\int x. f x * \max(g x) 0 - f x * \max(-g x) 0 \partial M)$
 by (rule Bochner-Integration.integral-diff[symmetric]) (simp-all add: int1 int2)
 also have $\dots = (\int x. f x * g x \partial M)$
 by (metis (mono-tags, opaque-lifting) diff-0 diff-zero eq-iff max.cobounded2 max-def minus-minus neg-le-0-iff-le right-diff-distrib)
 finally show $(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$
 by simp
 qed

lemma real-cond-exp-intg:

assumes integrable M $(\lambda x. f x * g x)$ and

[measurable]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$

shows integrable M $(\lambda x. f x * \text{real-cond-exp } M F g x)$

$(\int x. f x * \text{real-cond-exp } M F g x \partial M) = (\int x. f x * g x \partial M)$

proof –

have [measurable]: $f \in \text{borel-measurable } M$ by (rule measurable-from-subalg[OF subalg assms(2)])

define fp where $fp = (\lambda x. \max(f x) 0)$

define fm where $fm = (\lambda x. \max(-f x) 0)$

have [measurable]: $fp \in \text{borel-measurable } M$ $fm \in \text{borel-measurable } M$

unfolding fp-def fm-def by simp-all

have [measurable]: $fp \in \text{borel-measurable } F$ $fm \in \text{borel-measurable } F$

unfolding fp-def fm-def by simp-all

have $(\int^+ x. \text{norm}(fp x * g x) \partial M) \leq (\int^+ x. \text{norm}(f x * g x) \partial M)$

by (simp add: fp-def nn-integral-mono)
 also have $\dots < \infty$ using assms(1) by (simp add: integrable-iff-bounded)
 finally have $(\int^+ x. \text{norm}(fp\ x * g\ x)\ \partial M) < \infty$ by simp
 then have $intp: \text{integrable}\ M\ (\lambda x. fp\ x * g\ x)$ by (simp add: integrableI-bounded)
 moreover have $\bigwedge x. fp\ x \geq 0$ unfolding fp-def by simp
 ultimately have $Rp: \text{integrable}\ M\ (\lambda x. fp\ x * \text{real-cond-exp}\ M\ F\ g\ x)$
 $(\int x. fp\ x * \text{real-cond-exp}\ M\ F\ g\ x\ \partial M) = (\int x. fp\ x * g\ x\ \partial M)$
 using real-cond-exp-intg-fpos by auto

 have $(\int^+ x. \text{norm}(fm\ x * g\ x)\ \partial M) \leq (\int^+ x. \text{norm}(f\ x * g\ x)\ \partial M)$
 by (simp add: fm-def nn-integral-mono)
 also have $\dots < \infty$ using assms(1) by (simp add: integrable-iff-bounded)
 finally have $(\int^+ x. \text{norm}(fm\ x * g\ x)\ \partial M) < \infty$ by simp
 then have $intm: \text{integrable}\ M\ (\lambda x. fm\ x * g\ x)$ by (simp add: integrableI-bounded)
 moreover have $\bigwedge x. fm\ x \geq 0$ unfolding fm-def by simp
 ultimately have $Rm: \text{integrable}\ M\ (\lambda x. fm\ x * \text{real-cond-exp}\ M\ F\ g\ x)$
 $(\int x. fm\ x * \text{real-cond-exp}\ M\ F\ g\ x\ \partial M) = (\int x. fm\ x * g\ x\ \partial M)$
 using real-cond-exp-intg-fpos by auto

 have $\text{integrable}\ M\ (\lambda x. fp\ x * \text{real-cond-exp}\ M\ F\ g\ x - fm\ x * \text{real-cond-exp}\ M\ F\ g\ x)$
 using Rp(1) Rm(1) integrable-diff by simp
 moreover have $*$: $\bigwedge x. f\ x * \text{real-cond-exp}\ M\ F\ g\ x = fp\ x * \text{real-cond-exp}\ M\ F\ g\ x - fm\ x * \text{real-cond-exp}\ M\ F\ g\ x$
 unfolding fp-def fm-def by (simp add: max-def)
 ultimately show $\text{integrable}\ M\ (\lambda x. f\ x * \text{real-cond-exp}\ M\ F\ g\ x)$
 by simp

 have $(\int x. f\ x * \text{real-cond-exp}\ M\ F\ g\ x\ \partial M) = (\int x. fp\ x * \text{real-cond-exp}\ M\ F\ g\ x - fm\ x * \text{real-cond-exp}\ M\ F\ g\ x\ \partial M)$
 using $*$ by simp
 also have $\dots = (\int x. fp\ x * \text{real-cond-exp}\ M\ F\ g\ x\ \partial M) - (\int x. fm\ x * \text{real-cond-exp}\ M\ F\ g\ x\ \partial M)$
 using Rp(1) Rm(1) by simp
 also have $\dots = (\int x. fp\ x * g\ x\ \partial M) - (\int x. fm\ x * g\ x\ \partial M)$
 using Rp(2) Rm(2) by simp
 also have $\dots = (\int x. fp\ x * g\ x - fm\ x * g\ x\ \partial M)$
 using intm intp by simp
 also have $\dots = (\int x. f\ x * g\ x\ \partial M)$
 unfolding fp-def fm-def by (metis (no-types, opaque-lifting) diff-0 diff-zero max commute
 max-def minus-minus mult commute neg-le-iff-le right-diff-distrib)
 finally show $(\int x. f\ x * \text{real-cond-exp}\ M\ F\ g\ x\ \partial M) = (\int x. f\ x * g\ x\ \partial M)$ by
 simp
 qed

lemma real-cond-exp-intA:

assumes [measurable]: $\text{integrable}\ M\ f\ A \in \text{sets}\ F$

shows $(\int x \in A. f\ x\ \partial M) = (\int x \in A. \text{real-cond-exp}\ M\ F\ f\ x\ \partial M)$

proof –

have $A \in \text{sets } M$ **by** (*meson* *assms*(2) *subalg* *subalgebra-def* *subsetD*)
 have *integrable* M ($\lambda x. \text{indicator } A \ x * f \ x$) **using** *integrable-mult-indicator*[*OF*
 $\langle A \in \text{sets } M \rangle \text{ assms}(1)$] **by** *auto*
 then show *?thesis* **using** *real-cond-exp-intg*(2)[**where** *?f* = *indicator* A **and** *?g*
 = f , *symmetric*]
 unfolding *set-lebesgue-integral-def* **by** *auto*
qed

lemma *real-cond-exp-int* [*intro*]:

assumes *integrable* $M \ f$
 shows *integrable* M (*real-cond-exp* $M \ F \ f$) ($\int x. \text{real-cond-exp } M \ F \ f \ x \ \partial M$) =
 ($\int x. f \ x \ \partial M$)
using *real-cond-exp-intg*[**where** *?f* = $\lambda x. 1$ **and** *?g* = f] *assms* **by** *auto*

lemma *real-cond-exp-charact*:

assumes $\bigwedge A. A \in \text{sets } F \implies (\int x \in A. f \ x \ \partial M) = (\int x \in A. g \ x \ \partial M)$
 and [*measurable*]: *integrable* $M \ f$ *integrable* $M \ g$
 $g \in \text{borel-measurable } F$
 shows $AE \ x \text{ in } M. \text{real-cond-exp } M \ F \ f \ x = g \ x$

proof –

let *?MF* = *restr-to-subalg* $M \ F$
 have $AE \ x \text{ in } ?MF. \text{real-cond-exp } M \ F \ f \ x = g \ x$
 proof (*rule* *AE-symmetric*[*OF* *density-unique-real*])
 fix A **assume** $A \in \text{sets } ?MF$
 then have [*measurable*]: $A \in \text{sets } F$ **using** *sets-restr-to-subalg*[*OF* *subalg*] **by**

simp

then have a [*measurable*]: $A \in \text{sets } M$ **by** (*meson* *subalg* *subalgebra-def* *subsetD*)
 have ($\int x \in A. g \ x \ \partial ?MF$) = ($\int x \in A. g \ x \ \partial M$)
 unfolding *set-lebesgue-integral-def* **by** (*simp* *add*: *integral-subalgebra2* *subalg*)
 also have ... = ($\int x \in A. f \ x \ \partial M$) **using** *assms*(1) **by** *simp*
 also have ... = ($\int x. \text{indicator } A \ x * f \ x \ \partial M$) **by** (*simp* *add*: *mult.commute*
set-lebesgue-integral-def)
 also have ... = ($\int x. \text{indicator } A \ x * \text{real-cond-exp } M \ F \ f \ x \ \partial M$)
 apply (*rule* *real-cond-exp-intg*(2)[*symmetric*]) **using** *integrable-mult-indicator*[*OF*
 $a \text{ assms}(2)$] **by** (*auto* *simp* *add*: *assms*)
 also have ... = ($\int x \in A. \text{real-cond-exp } M \ F \ f \ x \ \partial M$) **by** (*simp* *add*: *mult.commute*
set-lebesgue-integral-def)
 also have ... = ($\int x \in A. \text{real-cond-exp } M \ F \ f \ x \ \partial ?MF$)
 by (*simp* *add*: *integral-subalgebra2* *subalg* *set-lebesgue-integral-def*)
 finally show ($\int x \in A. g \ x \ \partial ?MF$) = ($\int x \in A. \text{real-cond-exp } M \ F \ f \ x \ \partial ?MF$)

by *simp*

next

have *integrable* M (*real-cond-exp* $M \ F \ f$) **by** (*rule* *real-cond-exp-int*(1)[*OF*
assms(2)])
 then show *integrable* *?MF* (*real-cond-exp* $M \ F \ f$) **by** (*metis* *borel-measurable-cond-exp*
integrable-in-subalg[*OF* *subalg*])
 show *integrable* (*restr-to-subalg* $M \ F$) g **by** (*simp* *add*: *assms*(3) *integrable-in-subalg*[*OF*
subalg])

qed
then show *?thesis* **using** *AE-restr-to-subalg[OF subalg]* **by** *auto*
qed

lemma *real-cond-exp-F-meas* [intro, simp]:
assumes *integrable M f*
 $f \in \text{borel-measurable } F$
shows *AE x in M. real-cond-exp M F f x = f x*
by (rule *real-cond-exp-charact*, *auto simp add: assms measurable-from-subalg[OF subalg]*)

lemma *real-cond-exp-mult*:
assumes [*measurable*]: $f \in \text{borel-measurable } F$ $g \in \text{borel-measurable } M$ *integrable M* $(\lambda x. f x * g x)$
shows *AE x in M. real-cond-exp M F* $(\lambda x. f x * g x) x = f x * \text{real-cond-exp } M F g x$
proof (rule *real-cond-exp-charact*)
fix *A* **assume** $A \in \text{sets } F$
then have [*measurable*]: $(\lambda x. f x * \text{indicator } A x) \in \text{borel-measurable } F$ **by** *measurable*
have [*measurable*]: $A \in \text{sets } M$ **using** *subalg* **by** (meson $\langle A \in \text{sets } F \rangle$ *subalgebra-def subsetD*)
have $(\int_{x \in A. (f x * g x) \partial M}) = \int x. (f x * \text{indicator } A x) * g x \partial M$
by (*simp add: mult.commute mult.left-commute set-lebesgue-integral-def*)
also have $\dots = \int x. (f x * \text{indicator } A x) * \text{real-cond-exp } M F g x \partial M$
apply (rule *real-cond-exp-intg(2)[symmetric]*, *auto simp add: assms*)
using *integrable-mult-indicator[OF $\langle A \in \text{sets } M \rangle$ assms(3)]* **by** (*simp add: mult.commute mult.left-commute*)
also have $\dots = (\int_{x \in A. (f x * \text{real-cond-exp } M F g x) \partial M})$
by (*simp add: mult.commute mult.left-commute set-lebesgue-integral-def*)
finally show $(\int_{x \in A. (f x * g x) \partial M}) = (\int_{x \in A. (f x * \text{real-cond-exp } M F g x) \partial M})$ **by** *simp*
qed (*auto simp add: real-cond-exp-intg(1) assms*)

lemma *real-cond-exp-add* [intro]:
assumes [*measurable*]: *integrable M f* *integrable M g*
shows *AE x in M. real-cond-exp M F* $(\lambda x. f x + g x) x = \text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x$
proof (rule *real-cond-exp-charact*)
have *integrable M* $(\text{real-cond-exp } M F f)$ *integrable M* $(\text{real-cond-exp } M F g)$
using *real-cond-exp-int(1) assms* **by** *auto*
then show *integrable M* $(\lambda x. \text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x)$
by *auto*

fix *A* **assume** [*measurable*]: $A \in \text{sets } F$
then have $A \in \text{sets } M$ **by** (meson *subalg subalgebra-def subsetD*)
have *intAf*: *integrable M* $(\lambda x. \text{indicator } A x * f x)$
using *integrable-mult-indicator[OF $\langle A \in \text{sets } M \rangle$ assms(1)]* **by** *auto*
have *intAg*: *integrable M* $(\lambda x. \text{indicator } A x * g x)$

```

using integrable-mult-indicator[OF ‹ $A \in \text{sets } M$ › assms(2)] by auto

have  $(\int x \in A. (\text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x) \partial M) = (\int x \in A. \text{real-cond-exp } M F f x \partial M) + (\int x \in A. \text{real-cond-exp } M F g x \partial M)$ 
apply (rule set-integral-add, auto simp add: assms set-integrable-def)
using integrable-mult-indicator[OF ‹ $A \in \text{sets } M$ › real-cond-exp-int(1)[OF assms(1)]]
      integrable-mult-indicator[OF ‹ $A \in \text{sets } M$ › real-cond-exp-int(1)[OF
assms(2)]] by simp-all
also have  $\dots = (\int x. \text{indicator } A x * \text{real-cond-exp } M F f x \partial M) + (\int x. \text{indicator } A x * \text{real-cond-exp } M F g x \partial M)$ 
unfolding set-lebesgue-integral-def by auto
also have  $\dots = (\int x. \text{indicator } A x * f x \partial M) + (\int x. \text{indicator } A x * g x \partial M)$ 
using real-cond-exp-intg(2) assms ‹ $A \in \text{sets } F$ › intAf intAg by auto
also have  $\dots = (\int x \in A. f x \partial M) + (\int x \in A. g x \partial M)$ 
unfolding set-lebesgue-integral-def by auto
also have  $\dots = (\int x \in A. (f x + g x) \partial M)$ 
by (rule set-integral-add(2)[symmetric]) (auto simp add: assms set-integrable-def
‹ $A \in \text{sets } M$ › intAf intAg)
finally show  $(\int x \in A. (f x + g x) \partial M) = (\int x \in A. (\text{real-cond-exp } M F f x + \text{real-cond-exp } M F g x) \partial M)$ 
by simp
qed (auto simp add: assms)

```

lemma *real-cond-exp-cong*:

```

assumes ae:  $AE x \text{ in } M. f x = g x$  and [measurable]:  $f \in \text{borel-measurable } M$ 
 $g \in \text{borel-measurable } M$ 
shows  $AE x \text{ in } M. \text{real-cond-exp } M F f x = \text{real-cond-exp } M F g x$ 
proof –
have  $AE x \text{ in } M. \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (f x)) x = \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (g x)) x$ 
apply (rule nn-cond-exp-cong) using assms by auto
moreover have  $AE x \text{ in } M. \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-f x)) x = \text{nn-cond-exp } M F (\lambda x. \text{ennreal } (-g x)) x$ 
apply (rule nn-cond-exp-cong) using assms by auto
ultimately show  $AE x \text{ in } M. \text{real-cond-exp } M F f x = \text{real-cond-exp } M F g x$ 
unfolding real-cond-exp-def by auto
qed

```

lemma *real-cond-exp-cmult* [intro, simp]:

```

fixes c::real
assumes integrable  $M f$ 
shows  $AE x \text{ in } M. \text{real-cond-exp } M F (\lambda x. c * f x) x = c * \text{real-cond-exp } M F f x$ 
by (rule real-cond-exp-mult[where ?f =  $\lambda x. c$  and ?g = f], auto simp add: assms
borel-measurable-integrable)

```

lemma *real-cond-exp-cdiv* [intro, simp]:

```

fixes c::real
assumes integrable  $M f$ 

```


shows $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ (\lambda x.\ f\ x\ /\ c)\ x = real\ cond\ exp\ M\ F\ f\ x\ /\ c$

using $real\ cond\ exp\ cmult[of\ -\ 1\ /\ c,\ OF\ assms]$ **by** $(auto\ simp\ add:\ field\ split_simps)$

lemma $real\ cond\ exp\ diff\ [intro,\ simp]:$

assumes $[measurable]:\ integrable\ M\ f\ integrable\ M\ g$

shows $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ (\lambda x.\ f\ x - g\ x)\ x = real\ cond\ exp\ M\ F\ f\ x - real\ cond\ exp\ M\ F\ g\ x$

proof –

have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ (\lambda x.\ f\ x + (-\ g\ x))\ x = real\ cond\ exp\ M\ F\ f\ x + real\ cond\ exp\ M\ F\ (\lambda x.\ -g\ x)\ x$

using $real\ cond\ exp\ add[where\ ?f = f\ and\ ?g = \lambda x.\ -g\ x]$ **assms** **by** $auto$

moreover **have** $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ (\lambda x.\ -g\ x)\ x = -\ real\ cond\ exp\ M\ F\ g\ x$

using $real\ cond\ exp\ cmult[where\ ?f = g\ and\ ?c = -1]$ **assms** (2) **by** $auto$

ultimately **show** $?thesis$ **by** $auto$

qed

lemma $real\ cond\ exp\ pos\ [intro]:$

assumes $AE\ x\ in\ M.\ f\ x \geq 0$ **and** $[measurable]:\ f \in borel\ measurable\ M$

shows $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ f\ x \geq 0$

proof –

define g **where** $g = (\lambda x.\ max\ (f\ x)\ 0)$

have $AE\ x\ in\ M.\ f\ x = g\ x$ **using** $assms\ g\ def$ **by** $auto$

then **have** $*: AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ f\ x = real\ cond\ exp\ M\ F\ g\ x$ **using** $real\ cond\ exp\ cong\ g\ def$ **by** $auto$

have $\bigwedge x.\ g\ x \geq 0$ **unfolding** $g\ def$ **by** $simp$

then **have** $(\lambda x.\ ennreal(-g\ x)) = (\lambda x.\ 0)$

by $(simp\ add:\ ennreal\ neg)$

moreover **have** $AE\ x\ in\ M.\ 0 = nn\ cond\ exp\ M\ F\ (\lambda x.\ 0)\ x$

by $(rule\ nn\ cond\ exp\ F\ meas,\ auto)$

ultimately **have** $AE\ x\ in\ M.\ nn\ cond\ exp\ M\ F\ (\lambda x.\ ennreal(-g\ x))\ x = 0$

by $simp$

then **have** $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ g\ x = enn2real(nn\ cond\ exp\ M\ F\ (\lambda x.\ ennreal\ (g\ x))\ x)$

unfolding $real\ cond\ exp\ def$ **by** $auto$

then **have** $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ g\ x \geq 0$ **by** $auto$

then **show** $?thesis$ **using** $*$ **by** $auto$

qed

lemma $real\ cond\ exp\ mono:$

assumes $AE\ x\ in\ M.\ f\ x \leq g\ x$ **and** $[measurable]:\ integrable\ M\ f\ integrable\ M\ g$

shows $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ f\ x \leq real\ cond\ exp\ M\ F\ g\ x$

proof –

have $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ g\ x - real\ cond\ exp\ M\ F\ f\ x = real\ cond\ exp\ M\ F\ (\lambda x.\ g\ x - f\ x)\ x$

by $(rule\ AE\ symmetric[OF\ real\ cond\ exp\ diff],\ auto\ simp\ add:\ assms)$

moreover **have** $AE\ x\ in\ M.\ real\ cond\ exp\ M\ F\ (\lambda x.\ g\ x - f\ x)\ x \geq 0$

```

    by (rule real-cond-exp-pos, auto simp add: assms(1))
    ultimately have  $AE\ x\ in\ M. real-cond-exp\ M\ F\ g\ x - real-cond-exp\ M\ F\ f\ x \geq 0$  by auto
    then show ?thesis by auto
qed

```

```

lemma (in  $\mathcal{M}$ ) measurable-P-restriction [measurable (raw)]:
  assumes [measurable]:  $Measurable.pred\ M\ P\ A \in sets\ M$ 
  shows  $\{x \in A. P\ x\} \in sets\ M$ 
proof -
  have  $A \subseteq space\ M$  using sets.sets-into-space[OF assms(2)].
  then have  $\{x \in A. P\ x\} = A \cap \{x \in space\ M. P\ x\}$  by blast
  then show ?thesis by auto
qed

```

```

lemma real-cond-exp-gr-c:
  assumes [measurable]:  $integrable\ M\ f$ 
  and  $AE\ x\ in\ M. f\ x > c$ 
  shows  $AE\ x\ in\ M. real-cond-exp\ M\ F\ f\ x > c$ 
proof -
  define  $X$  where  $X = \{x \in space\ M. real-cond-exp\ M\ F\ f\ x \leq c\}$ 
  have [measurable]:  $X \in sets\ M$ 
  unfolding X-def apply measurable by (metis sets.top subalg subalgebra-def)
  then have [measurable]:  $X \in sets\ M$  using sets-restr-to-subalg subalg subalgebra-def by blast
  have  $emeasure\ M\ X = 0$ 
  proof (rule ccontr)
    assume  $\neg(emeasure\ M\ X) = 0$ 
    have  $emeasure\ (restr-to-subalg\ M\ F)\ X = emeasure\ M\ X$ 
      by (simp add: emeasure-restr-to-subalg subalg)
    then have  $emeasure\ (restr-to-subalg\ M\ F)\ X > 0$ 
      using  $\langle \neg(emeasure\ M\ X) = 0 \rangle\ gr-zeroI$  by auto
    then obtain  $A$  where  $A \in sets\ (restr-to-subalg\ M\ F)\ A \subseteq X$ 
       $emeasure\ (restr-to-subalg\ M\ F)\ A > 0$ 
       $emeasure\ (restr-to-subalg\ M\ F)\ A < \infty$ 
    using sigma-fin-subalg by (metis emeasure-notin-sets ennreal-0 infinity-ennreal-def le-less-linear neq-top-trans not-gr-zero order-refl sigma-finite-measure.approx-PInf-emeasure-with-finite)
    then have [measurable]:  $A \in sets\ F$  using subalg sets-restr-to-subalg by blast
    then have [measurable]:  $A \in sets\ M$  using sets-restr-to-subalg subalg subalgebra-def by blast
    have  $Ic: set-integrable\ M\ A\ (\lambda x. c)$ 
      unfolding set-integrable-def
      using  $\langle emeasure\ (restr-to-subalg\ M\ F)\ A < \infty \rangle\ emeasure-restr-to-subalg\ subalg$ 
    by fastforce
    have  $If: set-integrable\ M\ A\ f$ 
      unfolding set-integrable-def
      by (rule integrable-mult-indicator, auto simp add:  $\langle integrable\ M\ f \rangle$ )
    have  $AE\ x\ in\ M. indicator\ A\ x * c = indicator\ A\ x * f\ x$ 
    proof (rule integral-ineq-eq-0-then-AE)

```

```

have ( $\int x \in A. c \, \partial M$ ) = ( $\int x \in A. f \, x \, \partial M$ )
proof (rule antisym)
  show ( $\int x \in A. c \, \partial M$ )  $\leq$  ( $\int x \in A. f \, x \, \partial M$ )
    apply (rule set-integral-mono-AE) using Ic If assms(2) by auto
  have ( $\int x \in A. f \, x \, \partial M$ ) = ( $\int x \in A. \text{real-cond-exp } M \, F \, f \, x \, \partial M$ )
    by (rule real-cond-exp-intA, auto simp add:  $\langle \text{integrable } M \, f \rangle$ )
  also have ...  $\leq$  ( $\int x \in A. c \, \partial M$ )
    apply (rule set-integral-mono)
    unfolding set-integrable-def
    apply (rule integrable-mult-indicator, simp, simp add: real-cond-exp-int(1)[OF
 $\langle \text{integrable } M \, f \rangle$ ])
    using Ic X-def  $\langle A \subseteq X \rangle$  by (auto simp: set-integrable-def)
  finally show ( $\int x \in A. f \, x \, \partial M$ )  $\leq$  ( $\int x \in A. c \, \partial M$ ) by simp
qed
then have measure  $M \, A * c = \text{LINT } x | M. \text{indicat-real } A \, x * f \, x$ 
  by (auto simp: set-lebesgue-integral-def)
then show  $\text{LINT } x | M. \text{indicat-real } A \, x * c = \text{LINT } x | M. \text{indicat-real } A \, x * f \, x$ 
  by auto
show AE  $x$  in  $M. \text{indicat-real } A \, x * c \leq \text{indicat-real } A \, x * f \, x$ 
  using AE unfolding indicator-def by auto
qed (use Ic If in  $\langle \text{auto simp: set-integrable-def} \rangle$ )
then have AE  $x \in A$  in  $M. c = f \, x$  by auto
then have AE  $x \in A$  in  $M. \text{False}$  using assms(2) by auto
have  $A \in \text{null-sets } M$  unfolding ae-filter-def by (meson AE-iff-null-sets  $\langle A \in \text{sets } M \rangle \langle \text{AE } x \in A \text{ in } M. \text{False} \rangle$ )
then show False using  $\langle \text{emeasure } (\text{restr-to-subalg } M \, F) \, A > 0 \rangle$ 
  by (simp add: emeasure-restr-to-subalg null-setsD1 subalg)
qed
then show ?thesis using AE-iff-null-sets[OF  $\langle X \in \text{sets } M \rangle$ ] unfolding X-def
  by auto
qed

lemma real-cond-exp-less-c:
  assumes [measurable]: integrable  $M \, f$ 
  and AE  $x$  in  $M. f \, x < c$ 
  shows AE  $x$  in  $M. \text{real-cond-exp } M \, F \, f \, x < c$ 
proof -
  have AE  $x$  in  $M. \text{real-cond-exp } M \, F \, f \, x = -\text{real-cond-exp } M \, F \, (\lambda x. -f \, x) \, x$ 
    using real-cond-exp-cmult[OF  $\langle \text{integrable } M \, f \rangle$ , of  $-1$ ] by auto
  moreover have AE  $x$  in  $M. \text{real-cond-exp } M \, F \, (\lambda x. -f \, x) \, x > -c$ 
    apply (rule real-cond-exp-gr-c) using assms by auto
  ultimately show ?thesis by auto
qed

lemma real-cond-exp-ge-c:
  assumes [measurable]: integrable  $M \, f$ 
  and AE  $x$  in  $M. f \, x \geq c$ 
  shows AE  $x$  in  $M. \text{real-cond-exp } M \, F \, f \, x \geq c$ 

```

proof –
obtain $u::\text{nat} \Rightarrow \text{real}$ **where** $u: \bigwedge n. u\ n < c \longrightarrow c$
using *approx-from-below-dense-linorder*[*of* $c-1\ c$] **by** *auto*
have $*$: $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ f\ x > u\ n$ **for** $n::\text{nat}$
apply (*rule* *real-cond-exp-gr-c*) **using** *assms* $\langle u\ n < c \rangle$ **by** *auto*
have $AE\ x\ \text{in}\ M. \forall n. \text{real-cond-exp}\ M\ F\ f\ x > u\ n$
by (*subst* *AE-all-countable*, *auto simp add: **)
moreover **have** $\text{real-cond-exp}\ M\ F\ f\ x \geq c$ **if** $\forall n. \text{real-cond-exp}\ M\ F\ f\ x > u\ n$
for x
proof –
have $\text{real-cond-exp}\ M\ F\ f\ x \geq u\ n$ **for** n **using** *that less-imp-le* **by** *auto*
then show *?thesis* **using** $u(2)$ *LIMSEQ-le-const2* **by** *metis*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *real-cond-exp-le-c*:
assumes [*measurable*]: *integrable* $M\ f$
and $AE\ x\ \text{in}\ M. f\ x \leq c$
shows $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ f\ x \leq c$
proof –
have $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ f\ x = -\text{real-cond-exp}\ M\ F\ (\lambda x. -f\ x)\ x$
using *real-cond-exp-cmult*[*OF* $\langle \text{integrable}\ M\ f \rangle$, *of* -1] **by** *auto*
moreover **have** $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. -f\ x)\ x \geq -c$
apply (*rule* *real-cond-exp-ge-c*) **using** *assms* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *real-cond-exp-mono-strict*:
assumes $AE\ x\ \text{in}\ M. f\ x < g\ x$ **and** [*measurable*]: *integrable* $M\ f$ *integrable* $M\ g$
shows $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ f\ x < \text{real-cond-exp}\ M\ F\ g\ x$
proof –
have $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ g\ x - \text{real-cond-exp}\ M\ F\ f\ x = \text{real-cond-exp}\ M\ F\ (\lambda x. g\ x - f\ x)\ x$
by (*rule* *AE-symmetric*[*OF* *real-cond-exp-diff*], *auto simp add: assms*)
moreover **have** $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. g\ x - f\ x)\ x > 0$
by (*rule* *real-cond-exp-gr-c*, *auto simp add: assms*)
ultimately have $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ g\ x - \text{real-cond-exp}\ M\ F\ f\ x > 0$ **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *real-cond-exp-nested-subalg* [*intro*, *simp*]:
assumes *subalgebra* $M\ G$ *subalgebra* $G\ F$
and [*measurable*]: *integrable* $M\ f$
shows $AE\ x\ \text{in}\ M. \text{real-cond-exp}\ M\ F\ (\text{real-cond-exp}\ M\ G\ f)\ x = \text{real-cond-exp}\ M\ F\ f\ x$
proof (*rule* *real-cond-exp-charact*)
interpret G : *sigma-finite-subalgebra* $M\ G$ **by** (*rule* *nested-subalg-is-sigma-finite*[*OF*

```

assms(1) assms(2)])
show integrable M (real-cond-exp M G f) by (auto simp add: assms G.real-cond-exp-int(1))

fix A assume [measurable]: A ∈ sets F
then have [measurable]: A ∈ sets G using assms(2) by (meson subsetD subalgebra-def)
have set-lebesgue-integral M A (real-cond-exp M G f) = set-lebesgue-integral M A f
  by (rule G.real-cond-exp-intA[symmetric], auto simp add: assms(3))
also have ... = set-lebesgue-integral M A (real-cond-exp M F f)
  by (rule real-cond-exp-intA, auto simp add: assms(3))
finally show set-lebesgue-integral M A (real-cond-exp M G f) = set-lebesgue-integral M A (real-cond-exp M F f) by auto
qed (auto simp add: assms real-cond-exp-int(1))

lemma real-cond-exp-sum [intro, simp]:
  fixes f::'b ⇒ 'a ⇒ real
  assumes [measurable]: ∧i. integrable M (f i)
  shows AE x in M. real-cond-exp M F (λx. ∑ i∈I. f i x) x = (∑ i∈I. real-cond-exp M F (f i) x)
proof (rule real-cond-exp-charact)
  fix A assume [measurable]: A ∈ sets F
  then have A-meas [measurable]: A ∈ sets M by (meson subsetD subalg subalgebra-def)
  have *: integrable M (λx. indicator A x * f i x) for i
    using integrable-mult-indicator[OF ‹A ∈ sets M› assms(1)] by auto
  have **: integrable M (λx. indicator A x * real-cond-exp M F (f i) x) for i
    using integrable-mult-indicator[OF ‹A ∈ sets M› real-cond-exp-int(1)[OF assms(1)]]
  by auto
  have inti: (∫ x. indicator A x * f i x ∂M) = (∫ x. indicator A x * real-cond-exp M F (f i) x ∂M) for i
    by (rule real-cond-exp-intg(2)[symmetric], auto simp add: *)

  have (∫ x∈A. (∑ i∈I. f i x) ∂M) = (∫ x. (∑ i∈I. indicator A x * f i x) ∂M)
    by (simp add: sum-distrib-left set-lebesgue-integral-def)
  also have ... = (∑ i∈I. (∫ x. indicator A x * f i x ∂M))
    by (rule Bochner-Integration.integral-sum, simp add: *)
  also have ... = (∑ i∈I. (∫ x. indicator A x * real-cond-exp M F (f i) x ∂M))
    using inti by auto
  also have ... = (∫ x. (∑ i∈I. indicator A x * real-cond-exp M F (f i) x) ∂M)
    by (rule Bochner-Integration.integral-sum[symmetric], simp add: **)
  also have ... = (∫ x∈A. (∑ i∈I. real-cond-exp M F (f i) x) ∂M)
    by (simp add: sum-distrib-left set-lebesgue-integral-def)
  finally show (∫ x∈A. (∑ i∈I. f i x) ∂M) = (∫ x∈A. (∑ i∈I. real-cond-exp M F (f i) x) ∂M) by auto
qed (auto simp add: assms real-cond-exp-int(1)[OF assms(1)])

```

Jensen’s inequality, describing the behavior of the integral under a convex function, admits a version for the conditional expectation, as follows.

theorem *real-cond-exp-jensens-inequality*:

fixes $q :: \text{real} \Rightarrow \text{real}$
assumes X : *integrable* M X AE x *in* M . X $x \in I$
assumes I : $I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = UNIV$
assumes q : *integrable* M $(\lambda x. q (X x))$ *convex-on* I q $q \in \text{borel-measurable borel}$
shows AE x *in* M . *real-cond-exp* M F X $x \in I$
 AE x *in* M . q (*real-cond-exp* M F X x) \leq *real-cond-exp* M F $(\lambda x. q (X x))$ x
proof –
have *open* I **using** I **by** *auto*
then have *interior* $I = I$ **by** (*simp add: interior-eq*)
have [*measurable*]: $I \in \text{sets borel}$ **using** I **by** *auto*
define ϕ **where** $\phi = (\lambda x. \text{Inf } ((\lambda t. (q x - q t) / (x - t)) ' (\{x <..\} \cap I)))$
have **: $q (X x) \geq q$ (*real-cond-exp* M F X x) + ϕ (*real-cond-exp* M F X x) *
 $(X x - \text{real-cond-exp } M F X x)$
if $X x \in I$ *real-cond-exp* M F X $x \in I$ **for** x
unfolding ϕ -*def* **apply** (*rule convex-le-Inf-differential*)
using $\langle \text{convex-on } I \ q \rangle$ *that* $\langle \text{interior } I = I \rangle$ **by** *auto*

It is not clear that the function ϕ is measurable. We replace it by a version which is better behaved.

define ψ **where** $\psi = (\lambda x. \phi x * \text{indicator } I x)$
have A : $\psi y = \phi y$ **if** $y \in I$ **for** y **unfolding** ψ -*def* *indicator-def* **using** *that*
by *auto*
have *: $q (X x) \geq q$ (*real-cond-exp* M F X x) + ψ (*real-cond-exp* M F X x) *
 $(X x - \text{real-cond-exp } M F X x)$
if $X x \in I$ *real-cond-exp* M F X $x \in I$ **for** x
unfolding $A[OF \ \langle \text{real-cond-exp } M F X x \in I \rangle]$ **using** ** *that* **by** *auto*

note I
moreover have AE x *in* M . *real-cond-exp* M F X $x > a$ **if** $I \subseteq \{a <..\}$ **for** a
apply (*rule real-cond-exp-gr-c*) **using** X *that* **by** *auto*
moreover have AE x *in* M . *real-cond-exp* M F X $x < b$ **if** $I \subseteq \{..< b\}$ **for** b
apply (*rule real-cond-exp-less-c*) **using** X *that* **by** *auto*
ultimately show AE x *in* M . *real-cond-exp* M F X $x \in I$
by (*elim disjE*) (*auto simp: subset-eq*)
then have *main-ineq*: AE x *in* M . $q (X x) \geq q$ (*real-cond-exp* M F X x) + ψ
 $(\text{real-cond-exp } M F X x) * (X x - \text{real-cond-exp } M F X x)$
using * $X(2)$ **by** *auto*

Then, one wants to take the conditional expectation of this inequality. On the left, one gets the conditional expectation of $q \circ X$. On the right, the last term vanishes, and one is left with q of the conditional expectation, as desired. Unfortunately, this argument only works if $\psi \cdot X$ and $q(E(X|F))$ are integrable, and there is no reason why this should be true. The trick is to multiply by a F -measurable function which is small enough to make everything integrable.

obtain $f :: 'a \Rightarrow \text{real}$ **where** [*measurable*]: $f \in \text{borel-measurable (restr-to-subalg } M F)$

```

      integrable (restr-to-subalg M F) f
    and f:  $\bigwedge x. f\ x > 0 \ \bigwedge x. f\ x \leq 1$ 
  using sigma-finite-measure.obtain-positive-integrable-function[OF sigma-fin-subalg]
by metis
  then have [measurable]:  $f \in \text{borel-measurable } F$  by (simp add: subalg)
  then have [measurable]:  $f \in \text{borel-measurable } M$  using measurable-from-subalg[OF
subalg] by blast
  define g where  $g = (\lambda x. f\ x / (1 + |\text{psi } (\text{real-cond-exp } M\ F\ X\ x)| + |q\ (\text{real-cond-exp } M\ F\ X\ x)|))$ 
  define G where  $G = (\lambda x. g\ x * \text{psi } (\text{real-cond-exp } M\ F\ X\ x))$ 
  have g:  $g\ x > 0 \ g\ x \leq 1$  for x unfolding G-def g-def using f[of x] by (auto
simp add: abs-mult)
  have G:  $|G\ x| \leq 1$  for x unfolding G-def g-def using f[of x]
  proof (auto simp add: abs-mult)
    have  $f\ x * |\text{psi } (\text{real-cond-exp } M\ F\ X\ x)| \leq 1 * |\text{psi } (\text{real-cond-exp } M\ F\ X\ x)|$ 
    apply (rule mult-mono) using f[of x] by auto
    also have  $\dots \leq 1 + |\text{psi } (\text{real-cond-exp } M\ F\ X\ x)| + |q\ (\text{real-cond-exp } M\ F\ X\ x)|$ 
    by auto
    finally show  $f\ x * |\text{psi } (\text{real-cond-exp } M\ F\ X\ x)| \leq 1 + |\text{psi } (\text{real-cond-exp } M\ F\ X\ x)| + |q\ (\text{real-cond-exp } M\ F\ X\ x)|$ 
    by simp
  qed
  have AE x in M.  $g\ x * q\ (X\ x) \geq g\ x * (q\ (\text{real-cond-exp } M\ F\ X\ x) + \text{psi } (\text{real-cond-exp } M\ F\ X\ x) * (X\ x - \text{real-cond-exp } M\ F\ X\ x))$ 
  using main-ineq g by (auto simp add: divide-simps)
  then have main-G: AE x in M.  $g\ x * q\ (X\ x) \geq g\ x * q\ (\text{real-cond-exp } M\ F\ X\ x) + G\ x * (X\ x - \text{real-cond-exp } M\ F\ X\ x)$ 
  unfolding G-def by (auto simp add: algebra-simps)

```

To proceed, we need to know that ψ is measurable.

```

  have phi-mono:  $\text{phi } x \leq \text{phi } y$  if  $x \leq y$   $x \in I$   $y \in I$  for x y
  proof (cases  $x < y$ )
    case True
      have  $q\ x + \text{phi } x * (y - x) \leq q\ y$ 
      unfolding phi-def apply (rule convex-le-Inf-differential) using convex-on I
q that interior I = I by auto
      then have  $\text{phi } x \leq (q\ x - q\ y) / (x - y)$ 
      using that  $\langle x < y \rangle$  by (auto simp add: field-split-simps algebra-simps)
      moreover have  $(q\ x - q\ y) / (x - y) \leq \text{phi } y$ 
      unfolding phi-def proof (rule cInf-greatest, auto)
        fix t assume  $t \in I$   $y < t$ 
        have  $(q\ x - q\ y) / (x - y) \leq (q\ x - q\ t) / (x - t)$ 
        apply (rule convex-on-slope-le[OF q(2)]) using  $\langle y < t \rangle \langle x < y \rangle \langle t \in I \rangle \langle x \in I \rangle$  by auto
        also have  $\dots \leq (q\ y - q\ t) / (y - t)$ 
        apply (rule convex-on-slope-le[OF q(2)]) using  $\langle y < t \rangle \langle x < y \rangle \langle t \in I \rangle \langle x \in I \rangle$  by auto
        finally show  $(q\ x - q\ y) / (x - y) \leq (q\ y - q\ t) / (y - t)$  by simp
      next

```

```

    obtain  $e$  where  $0 < e$  ball  $y \in I$  using  $\langle \text{open } I \rangle \langle y \in I \rangle \text{openE}$  by blast
    then have  $y + e/2 \in \{y < ..\} \cap I$  by (auto simp: dist-real-def)
    then show  $\{y < ..\} \cap I = \{\} \implies \text{False}$  by auto
qed
ultimately show  $\phi x \leq \phi y$  by auto
next
case False
then have  $x = y$  using  $\langle x \leq y \rangle$  by auto
then show ?thesis by auto
qed
have [measurable]:  $\psi \in \text{borel-measurable borel}$ 
  by (rule borel-measurable-piecewise-mono[of  $\{I, -I\}$ ])
  (auto simp add:  $\psi\text{-def}$  indicator-def  $\phi\text{-mono}$  intro: mono-onI)
have [measurable]:  $q \in \text{borel-measurable borel}$  using  $q$  by simp

have [measurable]:  $X \in \text{borel-measurable } M$ 
  real-cond-exp  $M F X \in \text{borel-measurable } F$ 
   $g \in \text{borel-measurable } F$   $g \in \text{borel-measurable } M$ 
   $G \in \text{borel-measurable } F$   $G \in \text{borel-measurable } M$ 
  using  $X$  measurable-from-subalg[OF subalg] unfolding  $G\text{-def}$   $g\text{-def}$  by auto
have int1: integrable (restr-to-subalg  $M F$ )  $(\lambda x. g x * q (\text{real-cond-exp } M F X x))$ 
  apply (rule Bochner-Integration.integrable-bound[of -  $f$ ], auto simp add: subalg
  integrable (restr-to-subalg  $M F$ )  $f$ )
  unfolding  $g\text{-def}$  by (auto simp add: field-split-simps abs-mult algebra-simps)
have int2: integrable  $M$   $(\lambda x. G x * (X x - \text{real-cond-exp } M F X x))$ 
  apply (rule Bochner-Integration.integrable-bound[of -  $\lambda x. |X x| + |\text{real-cond-exp } M F X x|$ ])
  apply (auto intro!: Bochner-Integration.integrable-add integrable-abs real-cond-exp-int
  integrable  $M X$  AE-I2)
  using  $G$  unfolding abs-mult by (meson abs-ge-zero abs-triangle-ineq4 dual-order.trans
  mult-left-le-one-le)
have int3: integrable  $M$   $(\lambda x. g x * q (X x))$ 
  apply (rule Bochner-Integration.integrable-bound[of -  $\lambda x. q(X x)$ ], auto simp
  add:  $q(1)$  abs-mult)
  using  $g$  by (simp add: less-imp-le mult-left-le-one-le)

```

Taking the conditional expectation of the main convexity inequality `main_G`, we get the following.

```

have AE  $x$  in  $M$ . real-cond-exp  $M F$   $(\lambda x. g x * q (X x)) x \geq \text{real-cond-exp } M F$ 
 $(\lambda x. g x * q (\text{real-cond-exp } M F X x) + G x * (X x - \text{real-cond-exp } M F X x)) x$ 
  apply (rule real-cond-exp-mono[OF main-G])
  apply (rule Bochner-Integration.integrable-add[OF integrable-from-subalg[OF
  subalg int1]])
  using int2 int3 by auto

```

This reduces to the desired inequality thanks to the properties of conditional expectation, i.e., the conditional expectation of an F -measurable function is this function, and one can multiply an F -measurable function outside of conditional expectations. Since all these equalities only hold almost ev-

erywhere, we formulate them separately, and then combine all of them to simplify the above equation, again almost everywhere.

moreover have $AE\ x\ in\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. g\ x * q\ (X\ x))\ x = g\ x * \text{real-cond-exp}\ M\ F\ (\lambda x. q\ (X\ x))\ x$
by (rule *real-cond-exp-mult*, auto simp add: *int3*)
moreover have $AE\ x\ in\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. g\ x * q\ (\text{real-cond-exp}\ M\ F\ X\ x) + G\ x * (X\ x - \text{real-cond-exp}\ M\ F\ X\ x))\ x$
 $= \text{real-cond-exp}\ M\ F\ (\lambda x. g\ x * q\ (\text{real-cond-exp}\ M\ F\ X\ x))\ x + \text{real-cond-exp}\ M\ F\ (\lambda x. G\ x * (X\ x - \text{real-cond-exp}\ M\ F\ X\ x))\ x$
by (rule *real-cond-exp-add*, auto simp add: *integrable-from-subalg*[*OF subalg int1*] *int2*)
moreover have $AE\ x\ in\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. g\ x * q\ (\text{real-cond-exp}\ M\ F\ X\ x))\ x = g\ x * q\ (\text{real-cond-exp}\ M\ F\ X\ x)$
by (rule *real-cond-exp-F-meas*, auto simp add: *integrable-from-subalg*[*OF subalg int1*])
moreover have $AE\ x\ in\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. G\ x * (X\ x - \text{real-cond-exp}\ M\ F\ X\ x))\ x = G\ x * \text{real-cond-exp}\ M\ F\ (\lambda x. (X\ x - \text{real-cond-exp}\ M\ F\ X\ x))\ x$
by (rule *real-cond-exp-mult*, auto simp add: *int2*)
moreover have $AE\ x\ in\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. (X\ x - \text{real-cond-exp}\ M\ F\ X\ x))\ x = \text{real-cond-exp}\ M\ F\ X\ x - \text{real-cond-exp}\ M\ F\ (\lambda x. \text{real-cond-exp}\ M\ F\ X\ x)\ x$
by (rule *real-cond-exp-diff*, auto intro!: *real-cond-exp-int* $\langle \text{integrable}\ M\ X \rangle$)
moreover have $AE\ x\ in\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. \text{real-cond-exp}\ M\ F\ X\ x)\ x = \text{real-cond-exp}\ M\ F\ X\ x$
by (rule *real-cond-exp-F-meas*, auto intro!: *real-cond-exp-int* $\langle \text{integrable}\ M\ X \rangle$)
ultimately have $AE\ x\ in\ M. g\ x * \text{real-cond-exp}\ M\ F\ (\lambda x. q\ (X\ x))\ x \geq g\ x * q\ (\text{real-cond-exp}\ M\ F\ X\ x)$
by *auto*
then show $AE\ x\ in\ M. \text{real-cond-exp}\ M\ F\ (\lambda x. q\ (X\ x))\ x \geq q\ (\text{real-cond-exp}\ M\ F\ X\ x)$
using *g(1)* **by** (auto simp add: *field-split-simps*)
qed

Jensen’s inequality does not imply that $q(E(X|F))$ is integrable, as it only proves an upper bound for it. Indeed, this is not true in general, as the following counterexample shows:

on $[1, \infty)$ with Lebesgue measure, let F be the sigma-algebra generated by the intervals $[n, n+1)$ for integer n . Let $q(x) = -\sqrt{x}$ for $x \geq 0$. Define X which is equal to $1/n$ over $[n, n+1/n)$ and 2^{-n} on $[n+1/n, n+1)$. Then X is integrable as $\sum 1/n^2 < \infty$, and $q(X)$ is integrable as $\sum 1/n^{3/2} < \infty$. On the other hand, $E(X|F)$ is essentially equal to $1/n^2$ on $[n, n+1)$ (we neglect the term 2^{-n} , we only put it there because X should take its values in $I = (0, \infty)$). Hence, $q(E(X|F))$ is equal to $-1/n$ on $[n, n+1)$, hence it is not integrable.

However, this counterexample is essentially the only situation where this function is not integrable, as shown by the next lemma.

lemma *integrable-convex-cond-exp*:

```

fixes  $q :: \text{real} \Rightarrow \text{real}$ 
assumes  $X: \text{integrable } M \ X \ AE \ x \text{ in } M. \ X \ x \in I$ 
assumes  $I: I = \{a <..< b\} \vee I = \{a <..\} \vee I = \{..< b\} \vee I = UNIV$ 
assumes  $q: \text{integrable } M \ (\lambda x. q \ (X \ x)) \text{ convex-on } I \ q \ q \in \text{borel-measurable borel}$ 
assumes  $H: \text{emeasure } M \ (\text{space } M) = \infty \implies 0 \in I$ 
shows  $\text{integrable } M \ (\lambda x. q \ (\text{real-cond-exp } M \ F \ X \ x))$ 
proof –
  have  $[\text{measurable}]: (\lambda x. q \ (\text{real-cond-exp } M \ F \ X \ x)) \in \text{borel-measurable } M$ 
     $q \in \text{borel-measurable borel}$ 
     $X \in \text{borel-measurable } M$ 
  using  $X(1) \ q(3)$  by auto
  have open  $I$  using  $I$  by auto
  then have interior  $I = I$  by (simp add: interior-eq)

  consider  $\text{emeasure } M \ (\text{space } M) = 0 \mid \text{emeasure } M \ (\text{space } M) > 0 \wedge \text{emeasure}$ 
 $M \ (\text{space } M) < \infty \mid \text{emeasure } M \ (\text{space } M) = \infty$ 
  by (metis infinity-ennreal-def not-gr-zero top.not-eq-extremum)
  then show ?thesis
  proof (cases)
    case 1
      show ?thesis by (subst integrable-cong-AE[of - -  $\lambda x. 0$ ], auto intro: emea-
sure-0-AE[OF 1])
    next
      case 2
        interpret finite-measure  $M$  using 2 by (auto intro!: finite-measureI)

        have  $I \neq \{\}$ 
        using  $\langle AE \ x \text{ in } M. \ X \ x \in I \rangle$  2 eventually-mono integral-less-AE-space by
fastforce
        then obtain  $z$  where  $z \in I$  by auto

        define  $A$  where  $A = \text{Inf } ((\lambda t. (q \ z - q \ t) / (z - t)) \ ' (\{z <..\} \cap I))$ 
        have  $q \ y \geq q \ z + A * (y - z)$  if  $y \in I$  for  $y$  unfolding  $A\text{-def}$  apply (rule
convex-le-Inf-differential)
        using  $\langle z \in I \rangle \langle y \in I \rangle \langle \text{interior } I = I \rangle$   $q(2)$  by auto
        then have  $AE \ x \text{ in } M. \ q \ (\text{real-cond-exp } M \ F \ X \ x) \geq q \ z + A * (\text{real-cond-exp}$ 
 $M \ F \ X \ x - z)$ 
        using real-cond-exp-jensens-inequality(1)[OF X I q] by auto
        moreover have  $AE \ x \text{ in } M. \ q \ (\text{real-cond-exp } M \ F \ X \ x) \leq \text{real-cond-exp } M \ F$ 
 $(\lambda x. q \ (X \ x)) \ x$ 
        using real-cond-exp-jensens-inequality(2)[OF X I q] by auto
        moreover have  $|a| \leq |b| + |c|$  if  $b \leq a \wedge a \leq c$  for  $a \ b \ c :: \text{real}$ 
        using that by auto
        ultimately have  $*$ :  $AE \ x \text{ in } M. \ |q \ (\text{real-cond-exp } M \ F \ X \ x)|$ 
 $\leq |\text{real-cond-exp } M \ F \ (\lambda x. q \ (X \ x)) \ x| + |q \ z + A * (\text{real-cond-exp } M \ F \ X$ 
 $x - z)|$ 
        by auto

        show  $\text{integrable } M \ (\lambda x. q \ (\text{real-cond-exp } M \ F \ X \ x))$ 

```

```

apply (rule Bochner-Integration.integrable-bound[of -  $\lambda x. |real-cond-exp M F$ 
( $\lambda x. q (X x)$ )  $x| + |q z + A * (real-cond-exp M F X x - z)|$ ])
apply (auto intro!: Bochner-Integration.integrable-add integrable-abs inte-
grable-mult-right Bochner-Integration.integrable-diff real-cond-exp-int(1))
using  $X(1) q(1) * \text{by auto}$ 
next
case 3
then have  $0 \in I$  using  $H \text{ finite-measure.finite-emeasure-space}$  by auto
have  $q(0) = 0$ 
proof (rule ccontr)
assume *:  $\neg(q(0) = 0)$ 
define  $e$  where  $e = |q(0)| / 2$ 
then have  $e > 0$  using * by auto
have continuous (at 0)  $q$ 
using  $q(2) \langle 0 \in I \rangle \langle \text{open } I \rangle \langle \text{interior } I = I \rangle \text{ continuous-on-interior con-}$ 
 $\text{vex-on-continuous}$  by blast
then obtain  $d$  where  $d: d > 0 \wedge \forall y. |y - 0| < d \implies |q y - q 0| < e$  using
 $\langle e > 0 \rangle$ 
by (metis continuous-at-real-range real-norm-def)
then have *:  $|q(y)| > e$  if  $|y| < d$  for  $y$ 
proof -
have  $|q 0| \leq |q 0 - q y| + |q y|$  by auto
also have  $\dots < e + |q y|$  using  $d(2)$  that by force
finally have  $|q y| > |q 0| - e$  by auto
then show ?thesis unfolding  $e\text{-def}$  by simp
qed
have  $\text{emeasure } M \{x \in \text{space } M. |X x| < d\} \leq \text{emeasure } M (\{x \in \text{space } M.$ 
 $1 \leq \text{ennreal}(1/e) * |q(X x)|\})$ 
by (rule  $\text{emeasure-mono}$ , auto simp add: *  $\langle e > 0 \rangle \text{ less-imp-le ennreal-mult''[symmetric]}$ )
also have  $\dots \leq (1/e) * (\int^+ x. \text{ennreal}(|q(X x)|) * \text{indicator } (\text{space } M) x \partial M)$ 
by (rule  $\text{nn-integral-Markov-inequality}$ , auto)
also have  $\dots = (1/e) * (\int^+ x. \text{ennreal}(|q(X x)|) \partial M)$  by auto
also have  $\dots = (1/e) * \text{ennreal}(\int x. |q(X x)| \partial M)$ 
using  $\text{nn-integral-eq-integral}[OF \text{ integrable-abs}[OF q(1)]]$  by auto
also have  $\dots < \infty$ 
by (simp add:  $\text{ennreal-mult-less-top}$ )
finally have  $A: \text{emeasure } M \{x \in \text{space } M. |X x| < d\} < \infty$  by simp

have  $\{x \in \text{space } M. |X x| \geq d\} = \{x \in \text{space } M. 1 \leq \text{ennreal}(1/d) * |X x|\}$ 
 $\cap \text{space } M$ 
by (auto simp add:  $\langle d > 0 \rangle \text{ ennreal-mult''[symmetric]}$ )
then have  $\text{emeasure } M \{x \in \text{space } M. |X x| \geq d\} = \text{emeasure } M (\{x \in \text{space}$ 
 $M. 1 \leq \text{ennreal}(1/d) * |X x|\})$ 
by auto
also have  $\dots \leq (1/d) * (\int^+ x. \text{ennreal}(|X x|) * \text{indicator } (\text{space } M) x \partial M)$ 
by (rule  $\text{nn-integral-Markov-inequality}$ , auto)
also have  $\dots = (1/d) * (\int^+ x. \text{ennreal}(|X x|) \partial M)$  by auto
also have  $\dots = (1/d) * \text{ennreal}(\int x. |X x| \partial M)$ 
using  $\text{nn-integral-eq-integral}[OF \text{ integrable-abs}[OF X(1)]]$  by auto

```

```

also have ... <  $\infty$ 
  by (simp add: ennreal-mult-less-top)
finally have  $B$ : emeasure  $M$   $\{x \in \text{space } M. |X x| \geq d\} < \infty$  by simp

  have space  $M = \{x \in \text{space } M. |X x| < d\} \cup \{x \in \text{space } M. |X x| \geq d\}$  by
auto
  then have emeasure  $M$  (space  $M$ ) = emeasure  $M$  ( $\{x \in \text{space } M. |X x| < d\}$ 
 $\cup \{x \in \text{space } M. |X x| \geq d\}$ )
    by simp
  also have ...  $\leq$  emeasure  $M$   $\{x \in \text{space } M. |X x| < d\} +$  emeasure  $M$   $\{x \in$ 
space  $M. |X x| \geq d\}$ 
    by (auto intro!: emeasure-subadditive)
  also have ... <  $\infty$  using  $A$   $B$  by auto
  finally show False using  $\langle$ emeasure  $M$  (space  $M$ ) =  $\infty$  $\rangle$  by auto
qed

define  $A$  where  $A = \text{Inf } ((\lambda t. (q\ 0 - q\ t) / (0 - t)) \text{ ` } (\{0 < ..\} \cap I))$ 
have  $q\ y \geq q\ 0 + A * (y - 0)$  if  $y \in I$  for  $y$  unfolding  $A$ -def apply (rule
convex-le-Inf-differential)
  using  $\langle 0 \in I \rangle \langle y \in I \rangle \langle \text{interior } I = I \rangle q(2)$  by auto
then have  $q\ y \geq A * y$  if  $y \in I$  for  $y$  using  $\langle q\ 0 = 0 \rangle$  that by auto
then have  $AE\ x$  in  $M. q\ (\text{real-cond-exp } M\ F\ X\ x) \geq A * \text{real-cond-exp } M\ F\ X$ 
 $x$ 
  using real-cond-exp-jensens-inequality(1)[OF  $X\ I\ q$ ] by auto
moreover have  $AE\ x$  in  $M. q\ (\text{real-cond-exp } M\ F\ X\ x) \leq \text{real-cond-exp } M\ F$ 
 $(\lambda x. q\ (X\ x))\ x$ 
  using real-cond-exp-jensens-inequality(2)[OF  $X\ I\ q$ ] by auto
moreover have  $|a| \leq |b| + |c|$  if  $b \leq a \wedge a \leq c$  for  $a\ b\ c :: \text{real}$ 
  using that by auto
ultimately have *:  $AE\ x$  in  $M. |q\ (\text{real-cond-exp } M\ F\ X\ x)|$ 
 $\leq |\text{real-cond-exp } M\ F\ (\lambda x. q\ (X\ x))\ x| + |A * \text{real-cond-exp } M\ F\ X\ x|$ 
by auto

show integrable  $M\ (\lambda x. q\ (\text{real-cond-exp } M\ F\ X\ x))$ 
  apply (rule Bochner-Integration.integrable-bound[of -  $\lambda x. |\text{real-cond-exp } M\ F$ 
 $(\lambda x. q\ (X\ x))\ x| + |A * \text{real-cond-exp } M\ F\ X\ x|$ ])
  apply (auto intro!: Bochner-Integration.integrable-add integrable-abs inte-
grable-mult-right Bochner-Integration.integrable-diff real-cond-exp-int(1))
  using  $X(1)\ q(1) *$  by auto
qed
qed

end

end

```

```

theory Essential-Supremum
imports HOL-Analysis.Analysis

```

begin

lemma *ae-filter-eq-bot-iff*: $ae\text{-}filter\ M = bot \longleftrightarrow emeasure\ M\ (space\ M) = 0$
 by (*simp add: AE-iff-measurable trivial-limit-def*)

29 The essential supremum

In this paragraph, we define the essential supremum and give its basic properties. The essential supremum of a function is its maximum value if one is allowed to throw away a set of measure 0. It is convenient to define it to be infinity for non-measurable functions, as it allows for neater statements in general. This is a prerequisite to define the space L^∞ .

definition *esssup*:: $'a\ measure \Rightarrow ('a \Rightarrow 'b::\{second\text{-}countable\text{-}topology, dense\text{-}linorder, linorder\text{-}topology, complete\text{-}linorder\}) \Rightarrow 'b$

where $esssup\ M\ f = (if\ f \in borel\text{-}measurable\ M\ then\ Limsup\ (ae\text{-}filter\ M)\ f\ else\ top)$

lemma *esssup-non-measurable*: $f \notin M \rightarrow_M borel \implies esssup\ M\ f = top$
 by (*simp add: esssup-def*)

lemma *esssup-eq-AE*:

assumes $f: f \in M \rightarrow_M borel$ **shows** $esssup\ M\ f = Inf\ \{z. AE\ x\ in\ M. f\ x \leq z\}$

unfolding *esssup-def if-P[OF f] Limsup-def*

proof (*intro antisym INF-greatest Inf-greatest; clarsimp*)

fix y **assume** $AE\ x\ in\ M. f\ x \leq y$

then have $(\lambda x. f\ x \leq y) \in \{P. AE\ x\ in\ M. P\ x\}$

by *simp*

then show $(INF\ P \in \{P. AE\ x\ in\ M. P\ x\}. SUP\ x \in Collect\ P. f\ x) \leq y$

by (*rule INF-lower2*) (*auto intro: SUP-least*)

next

fix P **assume** $P: AE\ x\ in\ M. P\ x$

show $Inf\ \{z. AE\ x\ in\ M. f\ x \leq z\} \leq (SUP\ x \in Collect\ P. f\ x)$

proof (*rule Inf-lower; clarsimp*)

show $AE\ x\ in\ M. f\ x \leq (SUP\ x \in Collect\ P. f\ x)$

using P **by** (*auto elim: eventually-mono simp: SUP-upper*)

qed

qed

lemma *esssup-eq*: $f \in M \rightarrow_M borel \implies esssup\ M\ f = Inf\ \{z. emeasure\ M\ \{x \in space\ M. f\ x > z\} = 0\}$

by (*auto simp add: esssup-eq-AE not-less[symmetric] AE-iff-measurable[OF - refl] intro!: arg-cong[where f=Inf]*)

lemma *esssup-zero-measure*:

$emeasure\ M\ \{x \in space\ M. f\ x > esssup\ M\ f\} = 0$

proof (*cases esssup M f = top*)

case *True*

then show *?thesis* **by** *auto*

```

next
  case False
  then have f[measurable]:  $f \in M \rightarrow_M \text{borel}$  unfolding esssup-def by meson
  have  $\text{esssup } M f < \text{top}$  using False by (auto simp: less-top)
  have *:  $\{x \in \text{space } M. f x > z\} \in \text{null-sets } M$  if  $z > \text{esssup } M f$  for  $z$ 
  proof -
    have  $\exists w. w < z \wedge \text{emeasure } M \{x \in \text{space } M. f x > w\} = 0$ 
      using  $\langle z > \text{esssup } M f \rangle$  by (auto simp: esssup-eq Inf-less-iff)
    then obtain  $w$  where  $w < z$   $\text{emeasure } M \{x \in \text{space } M. f x > w\} = 0$  by
      auto
    then have  $a: \{x \in \text{space } M. f x > w\} \in \text{null-sets } M$  by auto
    have  $b: \{x \in \text{space } M. f x > z\} \subseteq \{x \in \text{space } M. f x > w\}$  using  $\langle w < z \rangle$  by
      auto
    show ?thesis using null-sets-subset[OF  $a - b$ ] by simp
  qed
  obtain  $u::\text{nat} \Rightarrow 'b$  where  $u: \bigwedge n. u n > \text{esssup } M f u \longrightarrow \text{esssup } M f$ 
    using approx-from-above-dense-linorder[OF  $\langle \text{esssup } M f < \text{top} \rangle$ ] by auto
  have  $\{x \in \text{space } M. f x > \text{esssup } M f\} = (\bigcup n. \{x \in \text{space } M. f x > u n\})$ 
    using  $u$  apply auto
  apply (metis (mono-tags, lifting) order-tendsto-iff eventually-mono LIMSEQ-unique)
    using less-imp-le less-le-trans by blast
  also have  $\dots \in \text{null-sets } M$ 
    using  $*(\text{OF } u(1))$  by auto
  finally show ?thesis by auto
qed

lemma esssup-AE:  $\text{AE } x \text{ in } M. f x \leq \text{esssup } M f$ 
proof (cases  $f \in M \rightarrow_M \text{borel}$ )
  case True then show ?thesis
    by (intro AE-I[OF esssup-zero-measure[of  $- f$ ]]) auto
qed (simp add: esssup-non-measurable)

lemma esssup-pos-measure:
   $f \in \text{borel-measurable } M \implies z < \text{esssup } M f \implies \text{emeasure } M \{x \in \text{space } M. f x > z\} > 0$ 
  using Inf-less-iff mem-Collect-eq not-gr-zero by (force simp: esssup-eq)

lemma esssup-I [intro]:  $f \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x \leq c \implies \text{esssup } M f \leq c$ 
  unfolding esssup-def by (simp add: Limsup-bounded)

lemma esssup-AE-mono:  $f \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x \leq g x \implies \text{esssup } M f \leq \text{esssup } M g$ 
  by (auto simp: esssup-def Limsup-mono)

lemma esssup-mono:  $f \in \text{borel-measurable } M \implies (\bigwedge x. f x \leq g x) \implies \text{esssup } M f \leq \text{esssup } M g$ 
  by (rule esssup-AE-mono) auto

```

lemma *esssup-AE-cong*:

$f \in \text{borel-measurable } M \implies g \in \text{borel-measurable } M \implies \text{AE } x \text{ in } M. f x = g x$
 $\implies \text{esssup } M f = \text{esssup } M g$
by (*auto simp: esssup-def intro!: Limsup-eq*)

lemma *esssup-const*: $\text{emeasure } M (\text{space } M) \neq 0 \implies \text{esssup } M (\lambda x. c) = c$
by (*simp add: esssup-def Limsup-const ae-filter-eq-bot-iff*)

lemma *esssup-cmult*: **assumes** $c > (0::\text{real})$ **shows** $\text{esssup } M (\lambda x. c * f x::\text{ereal}) = c * \text{esssup } M f$

proof –

have $(\lambda x. \text{ereal } c * f x) \in M \rightarrow_M \text{borel} \implies f \in M \rightarrow_M \text{borel}$

proof (*subst measurable-cong*)

fix ω **show** $f \omega = \text{ereal } (1/c) * (\text{ereal } c * f \omega)$

using $\langle 0 < c \rangle$ **by** (*cases f ω auto*)

qed *auto*

then have $(\lambda x. \text{ereal } c * f x) \in M \rightarrow_M \text{borel} \longleftrightarrow f \in M \rightarrow_M \text{borel}$

by(*safe intro!: borel-measurable-ereal-times borel-measurable-const*)

with $\langle 0 < c \rangle$ **show** *?thesis*

by (*cases ae-filter M = bot*)

(*auto simp: esssup-def bot-ereal-def top-ereal-def Limsup-ereal-mult-left*)

qed

lemma *esssup-add*:

$\text{esssup } M (\lambda x. f x + g x::\text{ereal}) \leq \text{esssup } M f + \text{esssup } M g$

proof (*cases f $\in \text{borel-measurable } M \wedge g \in \text{borel-measurable } M$*)

case *True*

then have [*measurable*]: $(\lambda x. f x + g x) \in \text{borel-measurable } M$ **by** *auto*

have $f x + g x \leq \text{esssup } M f + \text{esssup } M g$ **if** $f x \leq \text{esssup } M f$ $g x \leq \text{esssup } M g$
for x

using *that add-mono* **by** *auto*

then have $\text{AE } x \text{ in } M. f x + g x \leq \text{esssup } M f + \text{esssup } M g$

using *esssup-AE[of f M] esssup-AE[of g M]* **by** *auto*

then show *?thesis* **using** *esssup-I* **by** *auto*

next

case *False*

then have $\text{esssup } M f + \text{esssup } M g = \infty$ **unfolding** *esssup-def top-ereal-def*
by *auto*

then show *?thesis* **by** *auto*

qed

lemma *esssup-zero-space*:

$\text{emeasure } M (\text{space } M) = 0 \implies f \in \text{borel-measurable } M \implies \text{esssup } M f = (-\infty::\text{ereal})$

by (*simp add: esssup-def ae-filter-eq-bot-iff[symmetric] bot-ereal-def*)

end

30 Stopping times

```
theory Stopping-Time
  imports HOL-Analysis.Analysis
begin
```

30.1 Stopping Time

This is also called strong stopping time. Then stopping time is T with alternative is $T x < t$ measurable.

definition *stopping-time* :: $('t::\text{linorder} \Rightarrow 'a \text{ measure}) \Rightarrow ('a \Rightarrow 't) \Rightarrow \text{bool}$
where

stopping-time $F T = (\forall t. \text{Measurable.pred } (F t) (\lambda x. T x \leq t))$

lemma *stopping-time-cong*: $(\bigwedge t x. x \in \text{space } (F t) \implies T x = S x) \implies \text{stopping-time } F T = \text{stopping-time } F S$

unfolding *stopping-time-def* **by** (intro arg-cog[**where** $f = \text{All}$] ext measurable-cog) simp

lemma *stopping-timeD*: $\text{stopping-time } F T \implies \text{Measurable.pred } (F t) (\lambda x. T x \leq t)$

by (auto simp: *stopping-time-def*)

lemma *stopping-timeD2*: $\text{stopping-time } F T \implies \text{Measurable.pred } (F t) (\lambda x. t < T x)$

unfolding *not-le[symmetric]* **by** (auto intro: *stopping-timeD* *Measurable.pred-intros-logic*)

lemma *stopping-timeI*[intro?]: $(\bigwedge t. \text{Measurable.pred } (F t) (\lambda x. T x \leq t)) \implies \text{stopping-time } F T$

by (auto simp: *stopping-time-def*)

lemma *measurable-stopping-time*:

fixes $T :: 'a \Rightarrow 't::\{\text{linorder-topology, second-countable-topology}\}$

assumes T : *stopping-time* $F T$

and M : $\bigwedge t. \text{sets } (F t) \subseteq \text{sets } M \wedge t. \text{space } (F t) = \text{space } M$

shows $T \in M \rightarrow_M \text{borel}$

proof (rule *borel-measurableI-le*)

show $\{x \in \text{space } M. T x \leq t\} \in \text{sets } M$ **for** t

using *stopping-timeD*[OF T] M **by** (auto simp: *Measurable.pred-def*)

qed

lemma *stopping-time-const*: $\text{stopping-time } F (\lambda x. c)$

by (auto simp: *stopping-time-def*)

lemma *stopping-time-min*:

$\text{stopping-time } F T \implies \text{stopping-time } F S \implies \text{stopping-time } F (\lambda x. \min (T x) (S x))$

by (auto simp: *stopping-time-def* *min-le-iff-disj* intro!: *pred-intros-logic*)

lemma *stopping-time-max*:

stopping-time F $T \implies$ *stopping-time* F $S \implies$ *stopping-time* F $(\lambda x. \max (T x) (S x))$
by (*auto simp: stopping-time-def intro!: pred-intros-logic*)

31 Filtration

locale *filtration* =

fixes $\Omega :: 'a \text{ set}$ **and** $F :: 't :: \{\text{linorder-topology, second-countable-topology}\} \Rightarrow 'a \text{ measure}$

assumes *space-F*: $\bigwedge i. \text{space } (F i) = \Omega$

assumes *sets-F-mono*: $\bigwedge i j. i \leq j \implies \text{sets } (F i) \leq \text{sets } (F j)$

begin

31.1 σ -algebra of a Stopping Time

definition *pre-sigma* :: $('a \Rightarrow 't) \Rightarrow 'a \text{ measure}$

where

pre-sigma $T = \text{sigma } \Omega \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$

lemma *space-pre-sigma*: $\text{space } (\text{pre-sigma } T) = \Omega$

unfolding *pre-sigma-def* **using** *sets.space-closed*[*of F -*]

by (*intro space-measure-of*) (*auto simp: space-F*)

lemma *measure-pre-sigma*[*simp*]: $\text{emeasure } (\text{pre-sigma } T) = (\lambda-. 0)$

by (*simp add: pre-sigma-def emeasure-sigma*)

lemma *sigma-algebra-pre-sigma*:

assumes T : *stopping-time* F T

shows *sigma-algebra* $\Omega \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$

unfolding *sigma-algebra-iff2*

proof (*intro sigma-algebra-iff2*[*THEN iffD2*] *conjI ballI allI impI CollectI*)

show $\{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\} \subseteq \text{Pow } \Omega$

using *sets.space-closed*[*of F -*] **by** (*auto simp: space-F*)

next

fix A t **assume** $A \in \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$

then have $\{\omega \in \text{space } (F t). T \omega \leq t\} - \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)$

using T *stopping-timeD*[*measurable*] **by** *auto*

also have $\{\omega \in \text{space } (F t). T \omega \leq t\} - \{\omega \in A. T \omega \leq t\} = \{\omega \in \Omega - A. T \omega \leq t\}$

by (*auto simp: space-F*)

finally show $\{\omega \in \Omega - A. T \omega \leq t\} \in \text{sets } (F t)$.

next

fix $AA :: \text{nat} \Rightarrow 'a \text{ set}$ **and** t **assume** $\text{range } AA \subseteq \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$

then have $(\bigcup i. \{\omega \in AA i. T \omega \leq t\}) \in \text{sets } (F t)$ **for** t

by *auto*

also have $(\bigcup i. \{\omega \in AA i. T \omega \leq t\}) = \{\omega \in \bigcup (AA \text{ ' UNIV}). T \omega \leq t\}$

by *auto*

finally show $\{\omega \in \bigcup (AA \text{ ‘ } UNIV). T \omega \leq t\} \in \text{sets } (F t)$.
qed auto

lemma sets-pre-sigma: *stopping-time* $F T \implies \text{sets } (\text{pre-sigma } T) = \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$
unfolding pre-sigma-def by (rule sigma-algebra.sets-measure-of-eq[OF sigma-algebra-pre-sigma])

lemma sets-pre-sigmaI: *stopping-time* $F T \implies (\bigwedge t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)) \implies A \in \text{sets } (\text{pre-sigma } T)$
unfolding sets-pre-sigma by auto

lemma pred-pre-sigmaI:
assumes T : *stopping-time* $F T$
shows $(\bigwedge t. \text{Measurable.pred } (F t) (\lambda \omega. P \omega \wedge T \omega \leq t)) \implies \text{Measurable.pred } (\text{pre-sigma } T) P$
unfolding pred-def space-F space-pre-sigma by (intro sets-pre-sigmaI[OF T])
simp

lemma sets-pre-sigmaD: *stopping-time* $F T \implies A \in \text{sets } (\text{pre-sigma } T) \implies \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)$
unfolding sets-pre-sigma by auto

lemma stopping-time-le-const: *stopping-time* $F T \implies s \leq t \implies \text{Measurable.pred } (F t) (\lambda \omega. T \omega \leq s)$
using stopping-timeD[of $F T$] sets-F-mono[of - t] by (auto simp: pred-def space-F)

lemma measurable-stopping-time-pre-sigma:
assumes T : *stopping-time* $F T$ **shows** $T \in \text{pre-sigma } T \rightarrow_M \text{borel}$
proof (intro borel-measurableI-le sets-pre-sigmaI[OF T])
fix $t t'$
have $\{\omega \in \text{space } (F (\min t' t)). T \omega \leq \min t' t\} \in \text{sets } (F (\min t' t))$
using T **unfolding pred-def[symmetric] by** (rule stopping-timeD)
also have $\dots \subseteq \text{sets } (F t)$
by (rule sets-F-mono) **simp**
finally show $\{\omega \in \{x \in \text{space } (\text{pre-sigma } T). T x \leq t'\}. T \omega \leq t\} \in \text{sets } (F t)$
by (simp add: space-pre-sigma space-F)
qed

lemma mono-pre-sigma:
assumes T : *stopping-time* $F T$ **and** S : *stopping-time* $F S$
and le : $\bigwedge \omega. \omega \in \Omega \implies T \omega \leq S \omega$
shows $\text{sets } (\text{pre-sigma } T) \subseteq \text{sets } (\text{pre-sigma } S)$
unfolding sets-pre-sigma[OF S] sets-pre-sigma[OF T]
proof safe
interpret sigma-algebra $\Omega \{A. \forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)\}$
using T **by** (rule sigma-algebra-pre-sigma)
fix $A t$ **assume** A : $\forall t. \{\omega \in A. T \omega \leq t\} \in \text{sets } (F t)$
then have $A \subseteq \Omega$
using sets-into-space by auto

```

from  $A$  have  $\{\omega \in A. T \omega \leq t\} \cap \{\omega \in \text{space } (F t). S \omega \leq t\} \in \text{sets } (F t)$ 
  using stopping-timeD[OF S] by (auto simp: pred-def)
also have  $\{\omega \in A. T \omega \leq t\} \cap \{\omega \in \text{space } (F t). S \omega \leq t\} = \{\omega \in A. S \omega \leq t\}$ 
  using  $\langle A \subseteq \Omega \rangle \text{sets-into-space[of A]}$  le by (auto simp: space-F intro: order-trans)
finally show  $\{\omega \in A. S \omega \leq t\} \in \text{sets } (F t)$ 
  by auto
qed

```

lemma *stopping-time-less-const:*

assumes T : *stopping-time* $F T$ **shows** $\text{Measurable.pred } (F t) (\lambda \omega. T \omega < t)$

proof –

obtain $D :: 't \text{ set}$

where D : *countable* $D \wedge X. \text{open } X \implies X \neq \{\} \implies \exists d \in D. d \in X$

using *countable-dense-setE* **by auto**

show *?thesis*

proof cases

assume $*$: $\forall t' < t. \exists t''. t' < t'' \wedge t'' < t$

{ fix t' **assume** $t' < t$

with $*$ **have** $\{t' <..< t\} \neq \{\}$

by *fastforce*

with $D(2)[OF - \text{this}]$

have $\exists d \in D. t' < d \wedge d < t$

by auto }

note $** = \text{this}$

show *?thesis*

proof (*rule measurable-cong[THEN iffD2]*)

show $T \omega < t \longleftrightarrow (\exists r \in \{r \in D. r < t\}. T \omega \leq r)$ **for** ω

by (*auto dest: ** intro: less-imp-le*)

show $\text{Measurable.pred } (F t) (\lambda \omega. \exists r \in \{r \in D. r < t\}. T \omega \leq r)$

by (*intro measurable-pred-countable stopping-time-le-const[OF T] countable-Collect D*) **auto**

qed

next

assume $\neg (\forall t' < t. \exists t''. t' < t'' \wedge t'' < t)$

then obtain t' **where** $t': t' < t \wedge t''. t'' < t \implies t'' \leq t'$

by (*auto simp: not-less[symmetric]*)

show *?thesis*

proof (*rule measurable-cong[THEN iffD2]*)

show $T \omega < t \longleftrightarrow T \omega \leq t'$ **for** ω

using t' **by auto**

show $\text{Measurable.pred } (F t) (\lambda \omega. T \omega \leq t')$

using $\langle t' < t \rangle$ **by** (*intro stopping-time-le-const[OF T]*) **auto**

qed

qed

qed

lemma *stopping-time-eq-const:* *stopping-time* $F T \implies \text{Measurable.pred } (F t) (\lambda \omega. T \omega = t)$

unfolding *eq-iff* **using** *stopping-time-less-const*[*of T t*]
by (*intro pred-intros-logic stopping-time-le-const*) (*auto simp: not-less[symmetric]*)
)

lemma *stopping-time-less*:

assumes *T*: *stopping-time F T* **and** *S*: *stopping-time F S*
shows *Measurable.pred* (*pre-sigma T*) ($\lambda\omega. T \omega < S \omega$)
proof (*rule pred-pre-sigmaI[OF T]*)
fix *t*
obtain *D* :: '*t* set
where [*simp*]: *countable D* **and** *semidense-D*: $\bigwedge x y. x < y \implies (\exists b \in D. x \leq b \wedge b < y)$
using *countable-separating-set-linorder2* **by** *auto*
show *Measurable.pred* (*F t*) ($\lambda\omega. T \omega < S \omega \wedge T \omega \leq t$)
proof (*rule measurable-cong[THEN iffD2]*)
let *?f* = $\lambda\omega. \text{if } T \omega = t \text{ then } \neg S \omega \leq t \text{ else } \exists s \in \{s \in D. s \leq t\}. T \omega \leq s \wedge \neg (S \omega \leq s)$
{ **fix** ω **assume** $T \omega \leq t$ $T \omega \neq t$ $T \omega < S \omega$
then have $T \omega < \min t (S \omega)$
by *auto*
then obtain *r* **where** $r \in D$ $T \omega \leq r$ $r < \min t (S \omega)$
by (*metis semidense-D*)
then have $\exists s \in \{s \in D. s \leq t\}. T \omega \leq s \wedge s < S \omega$
by *auto* **}**
then show $(T \omega < S \omega \wedge T \omega \leq t) = ?f \omega$ **for** ω
by (*auto simp: not-le*)
show *Measurable.pred* (*F t*) *?f*
by (*intro pred-intros-logic measurable-If measurable-pred-countable countable-Collect*
stopping-time-le-const predE stopping-time-eq-const T S)
auto
qed
qed
end

lemma *stopping-time-SUP-enat*:

fixes *T* :: *nat* \Rightarrow (*a* \Rightarrow *enat*)
shows $(\bigwedge i. \text{stopping-time } F (T i)) \implies \text{stopping-time } F (SUP i. T i)$
unfolding *stopping-time-def SUP-apply SUP-le-iff* **by** (*auto intro!: pred-intros-countable*)

lemma *less-eSuc-iff*: $a < eSuc b \longleftrightarrow (a \leq b \wedge a \neq \infty)$

by (*cases a*) *auto*

lemma *stopping-time-Inf-enat*:

fixes *F* :: *enat* \Rightarrow '*a* *measure*
assumes *F*: *filtration* Ω *F*
assumes *P*: $\bigwedge i. \text{Measurable.pred } (F i) (P i)$
shows *stopping-time F* ($\lambda\omega. \text{Inf } \{i. P i \omega\}$)

```

proof (rule stopping-timeI, cases)
  fix  $t :: \text{enat}$  assume  $t = \infty$  then show  $\text{Measurable.pred } (F \ t) \ (\lambda \omega. \text{Inf } \{i. P \ i \ \omega\} \leq t)$ 
    by auto
next
  fix  $t :: \text{enat}$  assume  $t \neq \infty$ 
  moreover
  { fix  $i \ \omega$  assume  $\text{Inf } \{i. P \ i \ \omega\} \leq t$ 
    with  $\langle t \neq \infty \rangle$  have  $(\exists i \leq t. P \ i \ \omega)$ 
    unfolding Inf-le-iff by (cases  $t$ ) (auto elim!: allE[of - eSuc t] simp: less-eSuc-iff)
  }
  ultimately have  $*$ :  $\bigwedge \omega. \text{Inf } \{i. P \ i \ \omega\} \leq t \longleftrightarrow (\exists i \in \{..t\}. P \ i \ \omega)$ 
    by (auto intro!: Inf-lower2)
  show  $\text{Measurable.pred } (F \ t) \ (\lambda \omega. \text{Inf } \{i. P \ i \ \omega\} \leq t)$ 
    unfolding  $*$  using filtration.sets-F-mono[OF F, of - t]  $P$ 
    by (intro pred-intros-countable-bounded) (auto simp: pred-def filtration.space-F[OF F])
qed

lemma stopping-time-Inf-nat:
  fixes  $F :: \text{nat} \Rightarrow 'a \text{ measure}$ 
  assumes  $F$ : filtration  $\Omega \ F$ 
  assumes  $P$ :  $\bigwedge i. \text{Measurable.pred } (F \ i) \ (P \ i)$  and  $wf$ :  $\bigwedge i \ \omega. \omega \in \Omega \implies \exists n. P \ n \ \omega$ 
  shows stopping-time  $F \ (\lambda \omega. \text{Inf } \{i. P \ i \ \omega\})$ 
  unfolding stopping-time-def
proof (intro allI, subst measurable-cong)
  fix  $t \ \omega$  assume  $\omega \in \text{space } (F \ t)$ 
  then have  $\omega \in \Omega$ 
    using filtration.space-F[OF F] by auto
  from  $wf$  [OF this] have  $((\text{LEAST } n. P \ n \ \omega) \leq t) = (\exists i \leq t. P \ i \ \omega)$ 
    by (rule LeastI2-wellorder-ex) auto
  then show  $(\text{Inf } \{i. P \ i \ \omega\} \leq t) = (\exists i \in \{..t\}. P \ i \ \omega)$ 
    by (simp add: Inf-nat-def Bex-def)
next
  fix  $t$  from  $P$  show  $\text{Measurable.pred } (F \ t) \ (\lambda w. \exists i \in \{..t\}. P \ i \ w)$ 
    using filtration.sets-F-mono[OF F, of - t]
    by (intro pred-intros-countable-bounded) (auto simp: pred-def filtration.space-F[OF F])
qed

end

```

theory *Probability*

imports

Central-Limit-Theorem

Discrete-Topology

PMF-Impl

*Projective-Limit**Random-Permutations**SPMF**Product-PMF**Hoeffding**Stream-Space**Tree-Space**Conditional-Expectation**Essential-Supremum**Stopping-Time***begin****end**