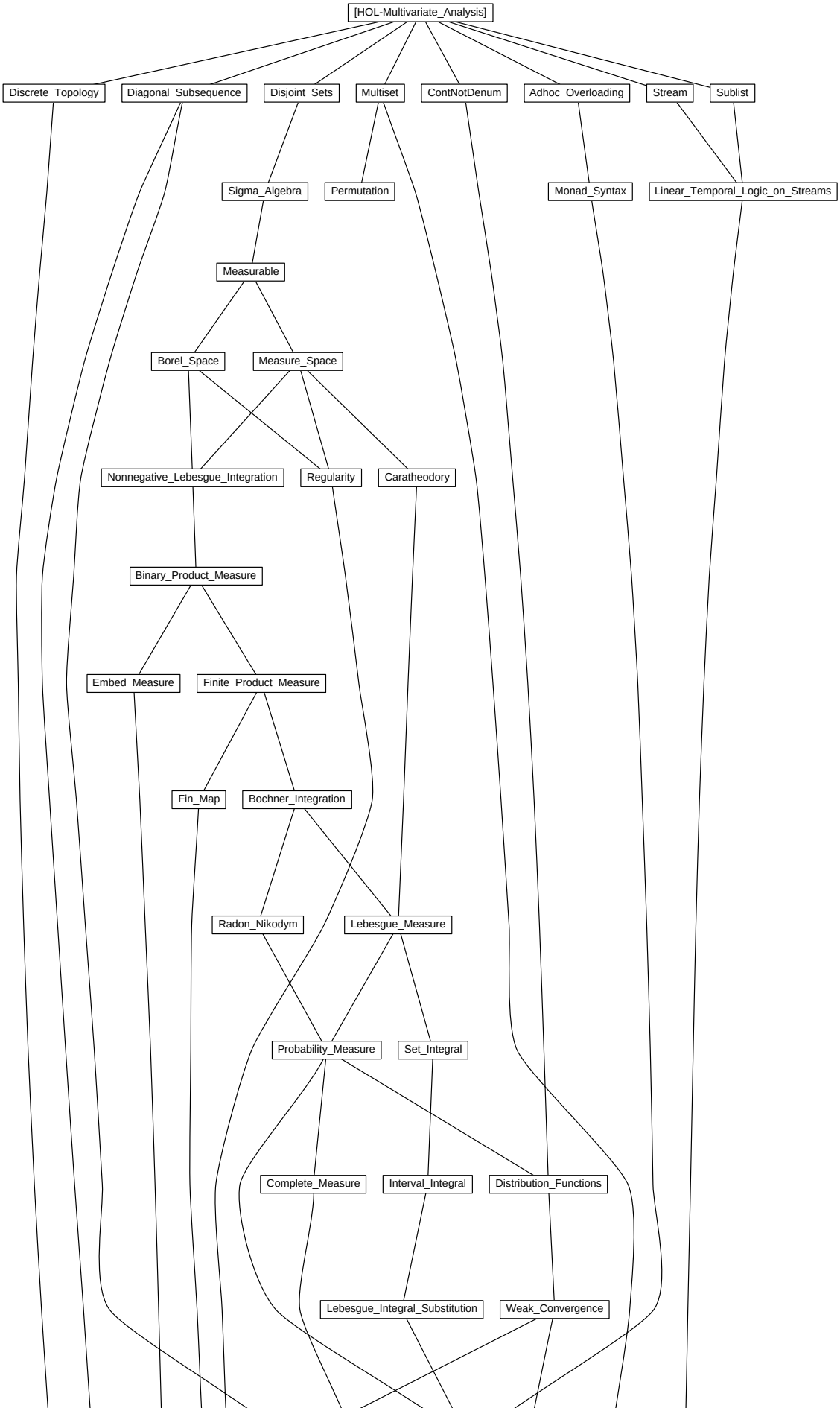
Measure and Probability Theory

April 17, 2016

Contents


[HOL-Multivariate_Analysis]
Discrete_Topology
Diagonal_Subsequence
Disjoint_Sets
Multiset
ContNotDenum
Adhoc_Overloading
Stream
Sublist
Sigma_Algebra
Permutation
Monad_Syntax
Linear_Temporal_Logic_on_Streams
Measurable
Borel_Space
Measure_Space
Nonnegative_Lebesgue_Integration
Regularity
Caratheodory
Binary_Product_Measure
Embed_Measure
Finite_Product_Measure
Fin_Map
Bochner_Integration
Radon_Nikodym
Lebesgue_Measure
Probability_Measure
Set_Integral
Complete_Measure
Interval_Integral
Distribution_Functions
Lebesgue_Integral_Substitution
Weak_Convergence

theory *Discrete-Topology*
imports ~~/*src*/*HOL*/*Multivariate-Analysis*/*Multivariate-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* ⇒ *'a discrete* ⇒ *real*
  where *dist-discrete n m* = (*if n* = *m then 0 else 1*)

definition *uniformity-discrete* :: (*'a discrete* × *'a discrete*) *filter* **where**
  (*uniformity*::(*'a discrete* × *'a discrete*) *filter*) = (*INF e*:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-discrete* :: *'a discrete set* ⇒ *bool* **where**
  (*open*::*'a discrete set* ⇒ *bool*) *U* ⟷ (∀ *x*∈*U*. *eventually* (λ(*x'*, *y*). *x'* = *x* ⟶ *y* ∈ *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*⇒*'a discrete* **assume** *Cauchy X*
  hence ∃ *n*. ∀ *m*≥*n*. *X n* = *X m*
    by (*force simp*: *dist-discrete-def Cauchy-def split*: *if-split-asm dest*:*spec*[**where** *x=1*])
  then guess *n* **..**
  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* (λ*c*::*'a discrete*. *to-nat* (*of-discrete c*))
    by (*simp add*: *inj-on-def of-discrete-inject*)
  thus ∃ *f*::*'a discrete* ⇒ *nat*. *inj f* **by** *blast*

qed

instance *discrete* :: (*countable*) *second-countable-topology*
proof
let *?B* = *range* (λ*n*::'*a discrete*. {*n*})
have ⋀*S*. *generate-topology ?B* (⋃ *x*∈*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 ∃ *B*::'*a discrete set set*. *countable B* ∧ *open* = *generate-topology B* **by** *blast*
qed

instance *discrete* :: (*countable*) *polish-space* **..**

end

1 Handling Disjoint Sets

theory *Disjoint-Sets*
imports *Main*
begin

lemma *range-subsetD*: *range f* ⊆ *B* ⟹ *f i* ∈ *B*
by *blast*

lemma *Int-Diff-disjoint*: *A* ∩ *B* ∩ (*A* − *B*) = {}
by *blast*

lemma *Int-Diff-Un*: *A* ∩ *B* ∪ (*A* − *B*) = *A*
by *blast*

lemma *mono-Un*: *mono A* ⟹ (⋃ *i*≤*n*. *A i*) = *A n*
unfolding *mono-def* **by** *auto*

1.1 Set of Disjoint Sets

abbreviation *disjoint* :: '*a set set* ⇒ *bool* **where** *disjoint* ≡ *pairwise disjnt*

lemma *disjoint-def*: *disjoint A* ⟷ (∀ *a*∈*A*. ∀ *b*∈*A*. *a* ≠ *b* ⟶ *a* ∩ *b* = {})
unfolding *pairwise-def disjnt-def* **by** *auto*

lemma *disjointI*:
(⋀*a b*. *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *a* ≠ *b* ⟹ *a* ∩ *b* = {}) ⟹ *disjoint A*
unfolding *disjoint-def* **by** *auto*

lemma *disjointD*:
disjoint A ⟹ *a* ∈ *A* ⟹ *b* ∈ *A* ⟹ *a* ≠ *b* ⟹ *a* ∩ *b* = {}

unfolding *disjoint-def* **by** *auto*

lemma *disjoint-INT*:
assumes $*$: $\bigwedge i.\ i \in I \Longrightarrow$ *disjoint* $(F\ i)$
shows *disjoint* $\{\bigcap i{\in}I.\ X\ i \mid X.\ \forall\, i{\in}I.\ X\ i \in F\ i\}$
proof (*safe intro*!: *disjointI del*: *equalityI*)
fix $A\ B :: 'a \Rightarrow 'b\ set$ **assume** $(\bigcap i{\in}I.\ A\ i) \neq (\bigcap i{\in}I.\ B\ i)$
then obtain i **where** $A\ i \neq B\ i\ i \in I$
by *auto*
moreover assume $\forall\, i{\in}I.\ A\ i \in F\ i\ \forall\, i{\in}I.\ B\ i \in F\ i$
ultimately show $(\bigcap i{\in}I.\ A\ i) \cap (\bigcap i{\in}I.\ B\ i) = \{\}$
using $*$[*OF* ‹$i{\in}I$›, *THEN disjointD*, *of A i B i*]
by (*auto simp*: *INT-Int-distrib*[*symmetric*])
qed

1.1.1 Family of Disjoint Sets

definition *disjoint-family-on* :: $('i \Rightarrow 'a\ set) \Rightarrow 'i\ set \Rightarrow bool$ **where**
disjoint-family-on A S $\longleftrightarrow$ $(\forall\, m{\in}S.\ \forall\, n{\in}S.\ m \neq n \longrightarrow A\ m \cap A\ n = \{\})$

abbreviation *disjoint-family A* $\equiv$ *disjoint-family-on A UNIV*

lemma *disjoint-family-onD*:
disjoint-family-on A I $\Longrightarrow i \in I \Longrightarrow j \in I \Longrightarrow i \neq j \Longrightarrow A\ i \cap A\ j = \{\}$
by (*auto simp*: *disjoint-family-on-def*)

lemma *disjoint-family-subset*: *disjoint-family A* $\Longrightarrow (\bigwedge x.\ B\ x \subseteq A\ x) \Longrightarrow$ *disjoint-family B*
by (*force simp add*: *disjoint-family-on-def*)

lemma *disjoint-family-on-bisimulation*:
assumes *disjoint-family-on f S*
and $\bigwedge n\ m.\ n \in S \Longrightarrow m \in S \Longrightarrow n \neq m \Longrightarrow f\ n \cap f\ m = \{\} \Longrightarrow g\ n \cap g\ m = \{\}$
shows *disjoint-family-on g S*
using *assms* **unfolding** *disjoint-family-on-def* **by** *auto*

lemma *disjoint-family-on-mono*:
$A \subseteq B \Longrightarrow$ *disjoint-family-on f B* $\Longrightarrow$ *disjoint-family-on f A*
unfolding *disjoint-family-on-def* **by** *auto*

lemma *disjoint-family-Suc*:
$(\bigwedge n.\ A\ n \subseteq A\ (Suc\ n)) \Longrightarrow$ *disjoint-family* $(\lambda i.\ A\ (Suc\ i) - A\ i)$
using *lift-Suc-mono-le*[*of A*]
by (*auto simp add*: *disjoint-family-on-def*)
(*metis insert-absorb insert-subset le-SucE le-antisym not-le-imp-less less-imp-le*)

lemma *disjoint-family-on-disjoint-image*:
disjoint-family-on A I $\Longrightarrow$ *disjoint* $(A\ `\ I)$

unfolding *disjoint-family-on-def disjoint-def* **by** *force*

lemma *disjoint-family-on-vimageI*: *disjoint-family-on F I* $\Longrightarrow$ *disjoint-family-on* ($\lambda i.\ f\ -`\ F\ i$) *I*
by (*auto simp*: *disjoint-family-on-def*)

lemma *disjoint-image-disjoint-family-on*:
assumes *d*: *disjoint* (*A* `` *I*) **and** *i*: *inj-on A I*
shows *disjoint-family-on A I*
unfolding *disjoint-family-on-def*
proof (*intro ballI impI*)
fix *n m* **assume** *nm*: $m \in I\ n \in I$ **and** $n \neq m$
with *i*[*THEN inj-onD*, *of n m*] **show** $A\ n \cap A\ m = \{\}$
by (*intro disjointD*[*OF d*]) *auto*
qed

lemma *disjoint-UN*:
assumes *F*: $\bigwedge i.\ i \in I \Longrightarrow$ *disjoint* (*F i*) **and** $*$: *disjoint-family-on* ($\lambda i.\ \bigcup F\ i$) *I*
shows *disjoint* ($\bigcup i{\in}I.\ F\ i$)
proof (*safe intro*!: *disjointI del*: *equalityI*)
fix *A B i j* **assume** $A \neq B\ A \in F\ i\ i \in I\ B \in F\ j\ j \in I$
show $A \cap B = \{\}$
proof *cases*
assume $i = j$ **with** *F*[*of i*] ‹$i \in I$› ‹$A \in F\ i$› ‹$B \in F\ j$› ‹$A \neq B$› **show** $A \cap B = \{\}$
by (*auto dest*: *disjointD*)
next
assume $i \neq j$
with $*$ ‹$i{\in}I$› ‹$j{\in}I$› **have** $(\bigcup F\ i) \cap (\bigcup F\ j) = \{\}$
by (*rule disjoint-family-onD*)
with ‹$A{\in}F\ i$› ‹$i{\in}I$› ‹$B{\in}F\ j$› ‹$j{\in}I$›
show $A \cap B = \{\}$
by *auto*
qed
qed

lemma *disjoint-union*: *disjoint C* $\Longrightarrow$ *disjoint B* $\Longrightarrow \bigcup C \cap \bigcup B = \{\} \Longrightarrow$ *disjoint* ($C \cup B$)
using *disjoint-UN*[*of* $\{C, B\}$ $\lambda x.\ x$] **by** (*auto simp add*: *disjoint-family-on-def*)

1.2 Construct Disjoint Sequences

definition *disjointed* :: (*nat* $\Rightarrow$ *'a set*) $\Rightarrow$ *nat* $\Rightarrow$ *'a set* **where**
disjointed A n = *A n* $-$ ($\bigcup i{\in}\{0..<n\}.\ A\ i$)

lemma *finite-UN-disjointed-eq*: ($\bigcup i{\in}\{0..<n\}.$ *disjointed A i*) = ($\bigcup i{\in}\{0..<n\}.\ A\ i$)
proof (*induct n*)
case *0* **show** *?case* **by** *simp*

next
case (*Suc n*)
thus *?case* **by** (*simp add*: *atLeastLessThanSuc disjointed-def*)
qed

lemma *UN-disjointed-eq*: $(\bigcup i.\ disjointed\ A\ i) = (\bigcup i.\ A\ i)$
by (*rule UN-finite2-eq* [**where** *k=0*])
(*simp add*: *finite-UN-disjointed-eq*)

lemma *less-disjoint-disjointed*: $m < n \Longrightarrow disjointed\ A\ m \cap disjointed\ A\ n = \{\}$
by (*auto simp add*: *disjointed-def*)

lemma *disjoint-family-disjointed*: *disjoint-family* (*disjointed A*)
by (*simp add*: *disjoint-family-on-def*)
(*metis neq-iff Int-commute less-disjoint-disjointed*)

lemma *disjointed-subset*: $disjointed\ A\ n \subseteq A\ n$
by (*auto simp add*: *disjointed-def*)

lemma *disjointed-0*[*simp*]: $disjointed\ A\ 0 = A\ 0$
by (*simp add*: *disjointed-def*)

lemma *disjointed-mono*: *mono A* $\Longrightarrow$ $disjointed\ A\ (Suc\ n) = A\ (Suc\ n) - A\ n$
using *mono-Un*[*of A*] **by** (*simp add*: *disjointed-def atLeastLessThanSuc-atLeastAtMost atLeast0AtMost*)

end

2 Describing measurable sets

theory *Sigma-Algebra*
imports
Complex-Main
~~/*src*/*HOL*/*Library*/*Countable-Set*
~~/*src*/*HOL*/*Library*/*FuncSet*
~~/*src*/*HOL*/*Library*/*Indicator-Function*
~~/*src*/*HOL*/*Library*/*Extended-Nonnegative-Real*
~~/*src*/*HOL*/*Library*/*Disjoint-Sets*
begin

Sigma algebras are an elementary concept in measure theory. To measure — that is to integrate — functions, we first have to measure sets. Unfortunately, when dealing with a large universe, it is often not possible to consistently assign a measure to every subset. Therefore it is necessary to define the set of measurable subsets of the universe. A sigma algebra is such a set that has three very natural and desirable properties.

2.1 Families of sets

locale *subset-class* =
  fixes $\Omega :: 'a\ set$ **and** $M :: 'a\ set\ set$
  assumes *space-closed*: $M \subseteq Pow\ \Omega$

lemma (**in** *subset-class*) *sets-into-space*: $x \in M \Longrightarrow x \subseteq \Omega$
  by (*metis PowD contra-subsetD space-closed*)

2.1.1 Semiring of sets

locale *semiring-of-sets* = *subset-class* +
  assumes *empty-sets*[*iff*]: $\{\} \in M$
  assumes *Int*[*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \cap b \in M$
  assumes *Diff-cover*:
    $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow \exists\, C \subseteq M.\ finite\ C \wedge disjoint\ C \wedge a - b = \bigcup C$

lemma (**in** *semiring-of-sets*) *finite-INT*[*intro*]:
  assumes *finite I* $I \neq \{\}$ $\bigwedge i.\ i \in I \Longrightarrow A\ i \in M$
  shows $(\bigcap i \in I.\ A\ i) \in M$
  using *assms* **by** (*induct rule*: *finite-ne-induct*) *auto*

lemma (**in** *semiring-of-sets*) *Int-space-eq1* [*simp*]: $x \in M \Longrightarrow \Omega \cap x = x$
  by (*metis Int-absorb1 sets-into-space*)

lemma (**in** *semiring-of-sets*) *Int-space-eq2* [*simp*]: $x \in M \Longrightarrow x \cap \Omega = x$
  by (*metis Int-absorb2 sets-into-space*)

lemma (**in** *semiring-of-sets*) *sets-Collect-conj*:
  assumes $\{x \in \Omega.\ P\ x\} \in M$ $\{x \in \Omega.\ Q\ x\} \in M$
  shows $\{x \in \Omega.\ Q\ x \wedge P\ x\} \in M$
proof −
  have $\{x \in \Omega.\ Q\ x \wedge P\ x\} = \{x \in \Omega.\ Q\ x\} \cap \{x \in \Omega.\ P\ x\}$
    by *auto*
  with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *semiring-of-sets*) *sets-Collect-finite-All'*:
  assumes $\bigwedge i.\ i \in S \Longrightarrow \{x \in \Omega.\ P\ i\ x\} \in M$ *finite S* $S \neq \{\}$
  shows $\{x \in \Omega.\ \forall\, i \in S.\ P\ i\ x\} \in M$
proof −
  have $\{x \in \Omega.\ \forall\, i \in S.\ P\ i\ x\} = (\bigcap i \in S.\ \{x \in \Omega.\ P\ i\ x\})$
    using ‹$S \neq \{\}$› **by** *auto*
  with *assms* **show** *?thesis* **by** *auto*
qed

locale *ring-of-sets* = *semiring-of-sets* +
  assumes *Un* [*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \cup b \in M$

lemma (**in** *ring-of-sets*) *finite-Union* [*intro*]:

finite $X \Longrightarrow X \subseteq M \Longrightarrow \bigcup X \in M$
by (*induct set*: *finite*) (*auto simp add*: *Un*)

lemma (**in** *ring-of-sets*) *finite-UN*[*intro*]:
assumes *finite* I **and** $\bigwedge i.\ i \in I \Longrightarrow A\ i \in M$
shows $(\bigcup i{\in}I.\ A\ i) \in M$
using *assms* **by** *induct auto*

lemma (**in** *ring-of-sets*) *Diff* [*intro*]:
assumes $a \in M\ b \in M$ **shows** $a - b \in M$
using *Diff-cover*[*OF assms*] **by** *auto*

lemma *ring-of-setsI*:
assumes *space-closed*: $M \subseteq Pow\ \Omega$
assumes *empty-sets*[*iff*]: $\{\} \in M$
assumes *Un*[*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \cup b \in M$
assumes *Diff*[*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a - b \in M$
shows *ring-of-sets* $\Omega\ M$
proof
fix $a\ b$ **assume** *ab*: $a \in M\ b \in M$
from *ab* **show** $\exists\, C{\subseteq}M.\ finite\ C \wedge disjoint\ C \wedge a - b = \bigcup C$
by (*intro exI*[*of* - $\{a - b\}$]) (*auto simp*: *disjoint-def*)
have $a \cap b = a - (a - b)$ **by** *auto*
also have $\ldots \in M$ **using** *ab* **by** *auto*
finally show $a \cap b \in M$ **.**
qed *fact*+

lemma *ring-of-sets-iff*: *ring-of-sets* $\Omega\ M \longleftrightarrow M \subseteq Pow\ \Omega \wedge \{\} \in M \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a \cup b \in M) \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a - b \in M)$
proof
assume *ring-of-sets* $\Omega\ M$
then interpret *ring-of-sets* $\Omega\ M$ **.**
show $M \subseteq Pow\ \Omega \wedge \{\} \in M \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a \cup b \in M) \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a - b \in M)$
using *space-closed* **by** *auto*
qed (*auto intro*!: *ring-of-setsI*)

lemma (**in** *ring-of-sets*) *insert-in-sets*:
assumes $\{x\} \in M\ A \in M$ **shows** *insert* $x\ A \in M$
proof −
have $\{x\} \cup A \in M$ **using** *assms* **by** (*rule Un*)
thus *?thesis* **by** *auto*
qed

lemma (**in** *ring-of-sets*) *sets-Collect-disj*:
assumes $\{x{\in}\Omega.\ P\ x\} \in M\ \{x{\in}\Omega.\ Q\ x\} \in M$
shows $\{x{\in}\Omega.\ Q\ x \vee P\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ Q\ x \vee P\ x\} = \{x{\in}\Omega.\ Q\ x\} \cup \{x{\in}\Omega.\ P\ x\}$

```
    by auto
  with assms show ?thesis by auto
qed

lemma (in ring-of-sets) sets-Collect-finite-Ex:
  assumes ⋀i. i ∈ S ⟹ {x∈Ω. P i x} ∈ M finite S
  shows {x∈Ω. ∃i∈S. P i x} ∈ M
proof −
  have {x∈Ω. ∃i∈S. P i x} = (⋃i∈S. {x∈Ω. P i x})
    by auto
  with assms show ?thesis by auto
qed

locale algebra = ring-of-sets +
  assumes top [iff]: Ω ∈ M

lemma (in algebra) compl-sets [intro]:
  a ∈ M ⟹ Ω − a ∈ M
  by auto

lemma algebra-iff-Un:
  algebra Ω M ⟷
    M ⊆ Pow Ω ∧
    {} ∈ M ∧
    (∀ a ∈ M. Ω − a ∈ M) ∧
    (∀ a ∈ M. ∀ b ∈ M. a ∪ b ∈ M) (is - ⟷ ?Un)
proof
  assume algebra Ω M
  then interpret algebra Ω M .
  show ?Un using sets-into-space by auto
next
  assume ?Un
  then have Ω ∈ M by auto
  interpret ring-of-sets Ω M
  proof (rule ring-of-setsI)
    show Ω: M ⊆ Pow Ω {} ∈ M
      using ‹?Un› by auto
    fix a b assume a: a ∈ M and b: b ∈ M
    then show a ∪ b ∈ M using ‹?Un› by auto
    have a − b = Ω − ((Ω − a) ∪ b)
      using Ω a b by auto
    then show a − b ∈ M
      using a b ‹?Un› by auto
  qed
  show algebra Ω M proof qed fact
qed

lemma algebra-iff-Int:
    algebra Ω M ⟷
```

$M \subseteq Pow\ \Omega$ & $\{\} \in M$ &
$(\forall\, a \in M.\ \Omega - a \in M)$ &
$(\forall\, a \in M.\ \forall\ b \in M.\ a \cap b \in M)$ (**is** - $\longleftrightarrow$ *?Int*)
proof
assume *algebra* Ω M
then interpret *algebra* Ω M **.**
show *?Int* **using** *sets-into-space* **by** *auto*
next
assume *?Int*
show *algebra* Ω M
proof (*unfold algebra-iff-Un*, *intro conjI ballI*)
show Ω: $M \subseteq Pow\ \Omega$ $\{\} \in M$
using ‹*?Int*› **by** *auto*
from ‹*?Int*› **show** $\bigwedge a.\ a \in M \Longrightarrow \Omega - a \in M$ **by** *auto*
fix a b **assume** M: $a \in M$ $b \in M$
hence $a \cup b = \Omega - ((\Omega - a) \cap (\Omega - b))$
using Ω **by** *blast*
also have $... \in M$
using M ‹*?Int*› **by** *auto*
finally show $a \cup b \in M$ **.**
qed
qed

lemma (**in** *algebra*) *sets-Collect-neg*:
assumes $\{x{\in}\Omega.\ P\ x\} \in M$
shows $\{x{\in}\Omega.\ \neg\ P\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \neg\ P\ x\} = \Omega - \{x{\in}\Omega.\ P\ x\}$ **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *algebra*) *sets-Collect-imp*:
$\{x{\in}\Omega.\ P\ x\} \in M \Longrightarrow \{x{\in}\Omega.\ Q\ x\} \in M \Longrightarrow \{x{\in}\Omega.\ Q\ x \longrightarrow P\ x\} \in M$
unfolding *imp-conv-disj* **by** (*intro sets-Collect-disj sets-Collect-neg*)

lemma (**in** *algebra*) *sets-Collect-const*:
$\{x{\in}\Omega.\ P\} \in M$
by (*cases* P) *auto*

lemma *algebra-single-set*:
$X \subseteq S \Longrightarrow$ *algebra* S { $\{\}$, X, $S - X$, S }
by (*auto simp*: *algebra-iff-Int*)

2.1.2 Restricted algebras

abbreviation (**in** *algebra*)
restricted-space $A \equiv (op \cap\ A)$ ‘ M

lemma (**in** *algebra*) *restricted-algebra*:

assumes $A \in M$ **shows** *algebra A* (*restricted-space A*)
using *assms* **by** (*auto simp*: *algebra-iff-Int*)

2.1.3 Sigma Algebras

locale *sigma-algebra* = *algebra* +
assumes *countable-nat-UN* [*intro*]: $\bigwedge A.\ \mathit{range}\ A \subseteq M \Longrightarrow (\bigcup i{::}\mathit{nat}.\ A\ i) \in M$

lemma (**in** *algebra*) *is-sigma-algebra*:
assumes *finite M*
shows *sigma-algebra* Ω *M*
proof
fix $A :: \mathit{nat} \Rightarrow 'a\ \mathit{set}$ **assume** $\mathit{range}\ A \subseteq M$
then have $(\bigcup i.\ A\ i) = (\bigcup s{\in}M \cap \mathit{range}\ A.\ s)$
by *auto*
also have $(\bigcup s{\in}M \cap \mathit{range}\ A.\ s) \in M$
using ‹*finite M*› **by** *auto*
finally show $(\bigcup i.\ A\ i) \in M$ **.**
qed

lemma *countable-UN-eq*:
fixes $A :: 'i{::}\mathit{countable} \Rightarrow 'a\ \mathit{set}$
shows $(\mathit{range}\ A \subseteq M \longrightarrow (\bigcup i.\ A\ i) \in M) \longleftrightarrow$
$(\mathit{range}\ (A \circ \mathit{from\text{-}nat}) \subseteq M \longrightarrow (\bigcup i.\ (A \circ \mathit{from\text{-}nat})\ i) \in M)$
proof −
let $?A' = A \circ \mathit{from\text{-}nat}$
have ∗: $(\bigcup i.\ ?A'\ i) = (\bigcup i.\ A\ i)$ (**is** $?l = ?r$)
proof *safe*
fix *x i* **assume** $x \in A\ i$ **thus** $x \in ?l$
by (*auto intro*!: *exI*[*of* - *to-nat i*])
next
fix *x i* **assume** $x \in ?A'\ i$ **thus** $x \in ?r$
by (*auto intro*!: *exI*[*of* - *from-nat i*])
qed
have ∗∗: $\mathit{range}\ ?A' = \mathit{range}\ A$
using *surj-from-nat*
by (*auto simp*: *image-comp* [*symmetric*] *intro*!: *imageI*)
show *?thesis* **unfolding** ∗ ∗∗ **..**
qed

lemma (**in** *sigma-algebra*) *countable-Union* [*intro*]:
assumes *countable X* $X \subseteq M$ **shows** $\bigcup X \in M$
proof *cases*
assume $X \neq \{\}$
hence $\bigcup X = (\bigcup n.\ \mathit{from\text{-}nat\text{-}into}\ X\ n)$
using *assms* **by** (*auto intro*: *from-nat-into*) (*metis from-nat-into-surj*)
also have $\ldots \in M$ **using** *assms*
by (*auto intro*!: *countable-nat-UN*) (*metis* ‹$X \neq \{\}$› *from-nat-into set-mp*)
finally show *?thesis* **.**

qed *simp*

lemma (**in** *sigma-algebra*) *countable-UN*[*intro*]:
fixes $A :: 'i::countable \Rightarrow 'a\ set$
assumes $A`X \subseteq M$
shows $(\bigcup x{\in}X.\ A\ x) \in M$
proof −
let $?A = \lambda i.\ if\ i \in X\ then\ A\ i\ else\ \{\}$
from *assms* **have** $range\ ?A \subseteq M$ **by** *auto*
with *countable-nat-UN*[*of* $?A \circ from\text{-}nat$] *countable-UN-eq*[*of* $?A\ M$]
have $(\bigcup x.\ ?A\ x) \in M$ **by** *auto*
moreover have $(\bigcup x.\ ?A\ x) = (\bigcup x{\in}X.\ A\ x)$ **by** (*auto split*: *if-split-asm*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *sigma-algebra*) *countable-UN′*:
fixes $A :: 'i \Rightarrow 'a\ set$
assumes X: $countable\ X$
assumes A: $A`X \subseteq M$
shows $(\bigcup x{\in}X.\ A\ x) \in M$
proof −
have $(\bigcup x{\in}X.\ A\ x) = (\bigcup i{\in}to\text{-}nat\text{-}on\ X\ `\ X.\ A\ (from\text{-}nat\text{-}into\ X\ i))$
using X **by** *auto*
also have $\ldots \in M$
using $A\ X$
by (*intro countable-UN*) *auto*
finally show *?thesis* **.**
qed

lemma (**in** *sigma-algebra*) *countable-UN″*:
$[\![\ countable\ X;\ \bigwedge x\ y.\ x \in X \Longrightarrow A\ x \in M\ ]\!] \Longrightarrow (\bigcup x{\in}X.\ A\ x) \in M$
by(*erule countable-UN′*)(*auto*)

lemma (**in** *sigma-algebra*) *countable-INT* [*intro*]:
fixes $A :: 'i::countable \Rightarrow 'a\ set$
assumes A: $A`X \subseteq M\ X \neq \{\}$
shows $(\bigcap i{\in}X.\ A\ i) \in M$
proof −
from A **have** $\forall i{\in}X.\ A\ i \in M$ **by** *fast*
hence $\Omega - (\bigcup i{\in}X.\ \Omega - A\ i) \in M$ **by** *blast*
moreover
have $(\bigcap i{\in}X.\ A\ i) = \Omega - (\bigcup i{\in}X.\ \Omega - A\ i)$ **using** *space-closed A*
by *blast*
ultimately show *?thesis* **by** *metis*
qed

lemma (**in** *sigma-algebra*) *countable-INT′*:
fixes $A :: 'i \Rightarrow 'a\ set$
assumes X: $countable\ X\ X \neq \{\}$

assumes *A*: $A`X \subseteq M$
shows $(\bigcap x{\in}X.\ A\ x) \in M$
proof −
have $(\bigcap x{\in}X.\ A\ x) = (\bigcap i{\in}$*to-nat-on* X ` $X.\ A$ (*from-nat-into* $X\ i$))
using X **by** *auto*
also have $\ldots \in M$
using $A\ X$
by (*intro countable-INT*) *auto*
finally show *?thesis* **.**
qed

lemma (**in** *sigma-algebra*) *countable-INT''*:
$UNIV \in M \Longrightarrow$ *countable* $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow F\ i \in M) \Longrightarrow (\bigcap i{\in}I.\ F\ i) \in M$
by (*cases* $I = \{\}$) (*auto intro*: *countable-INT'*)

lemma (**in** *sigma-algebra*) *countable*:
assumes $\bigwedge a.\ a \in A \Longrightarrow \{a\} \in M$ *countable* A
shows $A \in M$
proof −
have $(\bigcup a{\in}A.\ \{a\}) \in M$
using *assms* **by** (*intro countable-UN'*) *auto*
also have $(\bigcup a{\in}A.\ \{a\}) = A$ **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *ring-of-sets-Pow*: *ring-of-sets sp* (*Pow sp*)
by (*auto simp*: *ring-of-sets-iff*)

lemma *algebra-Pow*: *algebra sp* (*Pow sp*)
by (*auto simp*: *algebra-iff-Un*)

lemma *sigma-algebra-iff*:
sigma-algebra $\Omega\ M \longleftrightarrow$
algebra $\Omega\ M \wedge (\forall A.$ *range* $A \subseteq M \longrightarrow (\bigcup i{::}nat.\ A\ i) \in M)$
by (*simp add*: *sigma-algebra-def sigma-algebra-axioms-def*)

lemma *sigma-algebra-Pow*: *sigma-algebra sp* (*Pow sp*)
by (*auto simp*: *sigma-algebra-iff algebra-iff-Int*)

lemma (**in** *sigma-algebra*) *sets-Collect-countable-All*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$
shows $\{x{\in}\Omega.\ \forall i{::}'i{::}countable.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \forall i{::}'i{::}countable.\ P\ i\ x\} = (\bigcap i.\ \{x{\in}\Omega.\ P\ i\ x\})$ **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *sigma-algebra*) *sets-Collect-countable-Ex*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$

shows $\{x{\in}\Omega.\ \exists\, i{::}'i{::}countable.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \exists\, i{::}'i{::}countable.\ P\ i\ x\} = (\bigcup i.\ \{x{\in}\Omega.\ P\ i\ x\})$ **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *sigma-algebra*) *sets-Collect-countable-Ex'*:
assumes $\bigwedge i.\ i \in I \Longrightarrow \{x{\in}\Omega.\ P\ i\ x\} \in M$
assumes *countable I*
shows $\{x{\in}\Omega.\ \exists\, i{\in}I.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \exists\, i{\in}I.\ P\ i\ x\} = (\bigcup i{\in}I.\ \{x{\in}\Omega.\ P\ i\ x\})$ **by** *auto*
with *assms* **show** *?thesis*
by (*auto intro*!: *countable-UN'*)
qed

lemma (**in** *sigma-algebra*) *sets-Collect-countable-All'*:
assumes $\bigwedge i.\ i \in I \Longrightarrow \{x{\in}\Omega.\ P\ i\ x\} \in M$
assumes *countable I*
shows $\{x{\in}\Omega.\ \forall\, i{\in}I.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \forall\, i{\in}I.\ P\ i\ x\} = (\bigcap i{\in}I.\ \{x{\in}\Omega.\ P\ i\ x\}) \cap \Omega$ **by** *auto*
with *assms* **show** *?thesis*
by (*cases* $I = \{\}$) (*auto intro*!: *countable-INT'*)
qed

lemma (**in** *sigma-algebra*) *sets-Collect-countable-Ex1'*:
assumes $\bigwedge i.\ i \in I \Longrightarrow \{x{\in}\Omega.\ P\ i\ x\} \in M$
assumes *countable I*
shows $\{x{\in}\Omega.\ \exists!\, i{\in}I.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \exists!\, i{\in}I.\ P\ i\ x\} = \{x{\in}\Omega.\ \exists\, i{\in}I.\ P\ i\ x \wedge (\forall\, j{\in}I.\ P\ j\ x \longrightarrow i = j)\}$
by *auto*
with *assms* **show** *?thesis*
by (*auto intro*!: *sets-Collect-countable-All' sets-Collect-countable-Ex' sets-Collect-conj sets-Collect-imp sets-Collect-const*)
qed

lemmas (**in** *sigma-algebra*) *sets-Collect* =
sets-Collect-imp sets-Collect-disj sets-Collect-conj sets-Collect-neg sets-Collect-const sets-Collect-countable-All sets-Collect-countable-Ex sets-Collect-countable-All

lemma (**in** *sigma-algebra*) *sets-Collect-countable-Ball*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$
shows $\{x{\in}\Omega.\ \forall\, i{::}'i{::}countable{\in}X.\ P\ i\ x\} \in M$
unfolding *Ball-def* **by** (*intro sets-Collect assms*)

lemma (**in** *sigma-algebra*) *sets-Collect-countable-Bex*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$

shows $\{x{\in}\Omega.\ \exists\, i{::}'i{::}countable{\in}X.\ P\ i\ x\} \in M$
unfolding *Bex-def* **by** (*intro sets-Collect assms*)

lemma *sigma-algebra-single-set*:
assumes $X \subseteq S$
shows *sigma-algebra* S $\{\ \{\},\ X,\ S - X,\ S\ \}$
using *algebra.is-sigma-algebra*[*OF algebra-single-set*[*OF* ‹$X \subseteq S$›]] **by** *simp*

2.1.4 Binary Unions

definition *binary* :: $'a \Rightarrow 'a \Rightarrow nat \Rightarrow 'a$
where *binary* $a\ b = (\lambda x.\ b)(0 := a)$

lemma *range-binary-eq*: *range*(*binary* $a\ b$) = $\{a,b\}$
by (*auto simp add*: *binary-def*)

lemma *Un-range-binary*: $a \cup b = (\bigcup i{::}nat.\ binary\ a\ b\ i)$
by (*simp add*: *range-binary-eq cong del*: *strong-SUP-cong*)

lemma *Int-range-binary*: $a \cap b = (\bigcap i{::}nat.\ binary\ a\ b\ i)$
by (*simp add*: *range-binary-eq cong del*: *strong-INF-cong*)

lemma *sigma-algebra-iff2*:
sigma-algebra $\Omega\ M \longleftrightarrow$
$M \subseteq Pow\ \Omega \wedge$
$\{\} \in M \wedge (\forall s \in M.\ \Omega - s \in M) \wedge$
$(\forall A.\ range\ A \subseteq M \longrightarrow (\bigcup i{::}nat.\ A\ i) \in M)$
by (*auto simp add*: *range-binary-eq sigma-algebra-def sigma-algebra-axioms-def algebra-iff-Un Un-range-binary*)

2.1.5 Initial Sigma Algebra

Sigma algebras can naturally be created as the closure of any set of M with regard to the properties just postulated.

inductive-set *sigma-sets* :: $'a\ set \Rightarrow 'a\ set\ set \Rightarrow 'a\ set\ set$
for *sp* :: $'a\ set$ **and** A :: $'a\ set\ set$
where
Basic[*intro*, *simp*]: $a \in A \Longrightarrow a \in sigma\text{-}sets\ sp\ A$
| *Empty*: $\{\} \in sigma\text{-}sets\ sp\ A$
| *Compl*: $a \in sigma\text{-}sets\ sp\ A \Longrightarrow sp - a \in sigma\text{-}sets\ sp\ A$
| *Union*: $(\bigwedge i{::}nat.\ a\ i \in sigma\text{-}sets\ sp\ A) \Longrightarrow (\bigcup i.\ a\ i) \in sigma\text{-}sets\ sp\ A$

lemma (**in** *sigma-algebra*) *sigma-sets-subset*:
assumes *a*: $a \subseteq M$
shows *sigma-sets* $\Omega\ a \subseteq M$
proof
fix x
assume $x \in sigma\text{-}sets\ \Omega\ a$
from *this* **show** $x \in M$

by (*induct rule*: *sigma-sets.induct*, *auto*) (*metis a subsetD*)
qed

lemma *sigma-sets-into-sp*: $A \subseteq Pow\ sp \Longrightarrow x \in sigma\text{-}sets\ sp\ A \Longrightarrow x \subseteq sp$
by (*erule sigma-sets.induct*, *auto*)

lemma *sigma-algebra-sigma-sets*:
$a \subseteq Pow\ \Omega \Longrightarrow sigma\text{-}algebra\ \Omega\ (sigma\text{-}sets\ \Omega\ a)$
by (*auto simp add*: *sigma-algebra-iff2 dest*: *sigma-sets-into-sp*
intro!: *sigma-sets.Union sigma-sets.Empty sigma-sets.Compl*)

lemma *sigma-sets-least-sigma-algebra*:
assumes $A \subseteq Pow\ S$
shows $sigma\text{-}sets\ S\ A = \bigcap \{B.\ A \subseteq B \wedge sigma\text{-}algebra\ S\ B\}$
proof *safe*
fix $B\ X$ **assume** $A \subseteq B$ **and** *sa*: *sigma-algebra S B*
and X: $X \in sigma\text{-}sets\ S\ A$
from *sigma-algebra.sigma-sets-subset*[*OF sa*, *simplified*, *OF* ‹$A \subseteq B$›] X
show $X \in B$ **by** *auto*
next
fix X **assume** $X \in \bigcap \{B.\ A \subseteq B \wedge sigma\text{-}algebra\ S\ B\}$
then have [*intro*!]: $\bigwedge B.\ A \subseteq B \Longrightarrow sigma\text{-}algebra\ S\ B \Longrightarrow X \in B$
by *simp*
have $A \subseteq sigma\text{-}sets\ S\ A$ **using** *assms* **by** *auto*
moreover have *sigma-algebra S* (*sigma-sets S A*)
using *assms* **by** (*intro sigma-algebra-sigma-sets*[*of A*]) *auto*
ultimately show $X \in sigma\text{-}sets\ S\ A$ **by** *auto*
qed

lemma *sigma-sets-top*: $sp \in sigma\text{-}sets\ sp\ A$
by (*metis Diff-empty sigma-sets.Compl sigma-sets.Empty*)

lemma *sigma-sets-Un*:
$a \in sigma\text{-}sets\ sp\ A \Longrightarrow b \in sigma\text{-}sets\ sp\ A \Longrightarrow a \cup b \in sigma\text{-}sets\ sp\ A$
apply (*simp add*: *Un-range-binary range-binary-eq*)
apply (*rule Union*, *simp add*: *binary-def*)
done

lemma *sigma-sets-Inter*:
assumes *Asb*: $A \subseteq Pow\ sp$
shows $(\bigwedge i{::}nat.\ a\ i \in sigma\text{-}sets\ sp\ A) \Longrightarrow (\bigcap i.\ a\ i) \in sigma\text{-}sets\ sp\ A$
proof −
assume *ai*: $\bigwedge i{::}nat.\ a\ i \in sigma\text{-}sets\ sp\ A$
hence $\bigwedge i{::}nat.\ sp - (a\ i) \in sigma\text{-}sets\ sp\ A$
by (*rule sigma-sets.Compl*)
hence $(\bigcup i.\ sp - (a\ i)) \in sigma\text{-}sets\ sp\ A$
by (*rule sigma-sets.Union*)
hence $sp - (\bigcup i.\ sp - (a\ i)) \in sigma\text{-}sets\ sp\ A$
by (*rule sigma-sets.Compl*)

also have $sp-(\bigcup i.\ sp-(a\ i)) = sp\ Int\ (\bigcap i.\ a\ i)$
by *auto*
also have $... = (\bigcap i.\ a\ i)$ **using** *ai*
by (*blast dest*: *sigma-sets-into-sp* [*OF Asb*])
finally show *?thesis* **.**
qed

lemma *sigma-sets-INTER*:
assumes *Asb*: $A \subseteq Pow\ sp$
and *ai*: $\bigwedge i{::}nat.\ i \in S \Longrightarrow a\ i \in sigma\text{-}sets\ sp\ A$ **and** *non*: $S \neq \{\}$
shows $(\bigcap i{\in}S.\ a\ i) \in sigma\text{-}sets\ sp\ A$
proof −
from *ai* **have** $\bigwedge i.\ (if\ i{\in}S\ then\ a\ i\ else\ sp) \in sigma\text{-}sets\ sp\ A$
by (*simp add*: *sigma-sets.intros*(2−) *sigma-sets-top*)
hence $(\bigcap i.\ (if\ i{\in}S\ then\ a\ i\ else\ sp)) \in sigma\text{-}sets\ sp\ A$
by (*rule sigma-sets-Inter* [*OF Asb*])
also have $(\bigcap i.\ (if\ i{\in}S\ then\ a\ i\ else\ sp)) = (\bigcap i{\in}S.\ a\ i)$
by *auto* (*metis ai non sigma-sets-into-sp subset-empty subset-iff Asb*)+
finally show *?thesis* **.**
qed

lemma *sigma-sets-UNION*:
countable B $\Longrightarrow$ $(\bigwedge b.\ b \in B \Longrightarrow b \in sigma\text{-}sets\ X\ A) \Longrightarrow (\bigcup B) \in sigma\text{-}sets\ X\ A$
apply (*cases* $B = \{\}$)
apply (*simp add*: *sigma-sets.Empty*)
using *from-nat-into* [*of B*] *range-from-nat-into* [*of B*] *sigma-sets.Union* [*of from-nat-into B X A*]
apply *simp*
apply *auto*
apply (*metis Sup-bot-conv*(1) *Union-empty* ‹$[\![B \neq \{\};\ countable\ B]\!] \Longrightarrow range\ (from\text{-}nat\text{-}into\ B) = B$›)
done

lemma (**in** *sigma-algebra*) *sigma-sets-eq*:
$sigma\text{-}sets\ \Omega\ M = M$
proof
show $M \subseteq sigma\text{-}sets\ \Omega\ M$
by (*metis Set.subsetI sigma-sets.Basic*)
next
show $sigma\text{-}sets\ \Omega\ M \subseteq M$
by (*metis sigma-sets-subset subset-refl*)
qed

lemma *sigma-sets-eqI*:
assumes *A*: $\bigwedge a.\ a \in A \Longrightarrow a \in sigma\text{-}sets\ M\ B$
assumes *B*: $\bigwedge b.\ b \in B \Longrightarrow b \in sigma\text{-}sets\ M\ A$
shows $sigma\text{-}sets\ M\ A = sigma\text{-}sets\ M\ B$
proof (*intro set-eqI iffI*)

fix a **assume** $a \in$ *sigma-sets* M A
from *this* A **show** $a \in$ *sigma-sets* M B
by *induct* (*auto intro*!: *sigma-sets.intros*(2−) *del*: *sigma-sets.Basic*)
next
fix b **assume** $b \in$ *sigma-sets* M B
from *this* B **show** $b \in$ *sigma-sets* M A
by *induct* (*auto intro*!: *sigma-sets.intros*(2−) *del*: *sigma-sets.Basic*)
qed

lemma *sigma-sets-subseteq*: **assumes** $A \subseteq B$ **shows** *sigma-sets* X $A \subseteq$ *sigma-sets* X B
proof
fix x **assume** $x \in$ *sigma-sets* X A **then show** $x \in$ *sigma-sets* X B
by *induct* (*insert* ‹$A \subseteq B$›, *auto intro*: *sigma-sets.intros*(2−))
qed

lemma *sigma-sets-mono*: **assumes** $A \subseteq$ *sigma-sets* X B **shows** *sigma-sets* X $A \subseteq$ *sigma-sets* X B
proof
fix x **assume** $x \in$ *sigma-sets* X A **then show** $x \in$ *sigma-sets* X B
by *induct* (*insert* ‹$A \subseteq$ *sigma-sets* X B›, *auto intro*: *sigma-sets.intros*(2−))
qed

lemma *sigma-sets-mono'*: **assumes** $A \subseteq B$ **shows** *sigma-sets* X $A \subseteq$ *sigma-sets* X B
proof
fix x **assume** $x \in$ *sigma-sets* X A **then show** $x \in$ *sigma-sets* X B
by *induct* (*insert* ‹$A \subseteq B$›, *auto intro*: *sigma-sets.intros*(2−))
qed

lemma *sigma-sets-superset-generator*: $A \subseteq$ *sigma-sets* X A
by (*auto intro*: *sigma-sets.Basic*)

lemma (**in** *sigma-algebra*) *restriction-in-sets*:
fixes $A :: nat \Rightarrow 'a$ *set*
assumes $S \in M$
and $*$: *range* $A \subseteq (\lambda A.\ S \cap A)$ ‘ M (**is** - $\subseteq$ *?r*)
shows *range* $A \subseteq M$ $(\bigcup i.\ A\ i) \in (\lambda A.\ S \cap A)$ ‘ M
proof −
{ fix i **have** $A\ i \in$ *?r* **using** $*$ **by** *auto*
hence $\exists B.\ A\ i = B \cap S \wedge B \in M$ **by** *auto*
hence $A\ i \subseteq S$ $A\ i \in M$ **using** ‹$S \in M$› **by** *auto* **}**
thus *range* $A \subseteq M$ $(\bigcup i.\ A\ i) \in (\lambda A.\ S \cap A)$ ‘ M
by (*auto intro*!: *image-eqI*[*of* - - $(\bigcup i.\ A\ i)$])
qed

lemma (**in** *sigma-algebra*) *restricted-sigma-algebra*:
assumes $S \in M$
shows *sigma-algebra* S (*restricted-space* S)

```
  unfolding sigma-algebra-def sigma-algebra-axioms-def
proof safe
  show algebra S (restricted-space S) using restricted-algebra[OF assms] .
next
  fix A :: nat ⇒ 'a set assume range A ⊆ restricted-space S
  from restriction-in-sets[OF assms this[simplified]]
  show (⋃ i. A i) ∈ restricted-space S by simp
qed

lemma sigma-sets-Int:
  assumes A ∈ sigma-sets sp st A ⊆ sp
  shows op ∩ A ‘ sigma-sets sp st = sigma-sets A (op ∩ A ‘ st)
proof (intro equalityI subsetI)
  fix x assume x ∈ op ∩ A ‘ sigma-sets sp st
  then obtain y where y ∈ sigma-sets sp st x = y ∩ A by auto
  then have x ∈ sigma-sets (A ∩ sp) (op ∩ A ‘ st)
  proof (induct arbitrary: x)
    case (Compl a)
    then show ?case
      by (force intro!: sigma-sets.Compl simp: Diff-Int-distrib ac-simps)
  next
    case (Union a)
    then show ?case
      by (auto intro!: sigma-sets.Union
            simp add: UN-extend-simps simp del: UN-simps)
  qed (auto intro!: sigma-sets.intros(2−))
  then show x ∈ sigma-sets A (op ∩ A ‘ st)
    using ‹A ⊆ sp› by (simp add: Int-absorb2)
next
  fix x assume x ∈ sigma-sets A (op ∩ A ‘ st)
  then show x ∈ op ∩ A ‘ sigma-sets sp st
  proof induct
    case (Compl a)
    then obtain x where a = A ∩ x x ∈ sigma-sets sp st by auto
    then show ?case using ‹A ⊆ sp›
      by (force simp add: image-iff intro!: bexI[of - sp − x] sigma-sets.Compl)
  next
    case (Union a)
    then have ∀ i. ∃ x. x ∈ sigma-sets sp st ∧ a i = A ∩ x
      by (auto simp: image-iff Bex-def)
    from choice[OF this] guess f ..
    then show ?case
      by (auto intro!: bexI[of - (⋃ x. f x)] sigma-sets.Union
            simp add: image-iff)
  qed (auto intro!: sigma-sets.intros(2−))
qed

lemma sigma-sets-empty-eq: sigma-sets A {} = {{}, A}
proof (intro set-eqI iffI)
```

fix a **assume** $a \in$ *sigma-sets* A $\{\}$ **then show** $a \in \{\{\}, A\}$
by *induct blast+*
qed (*auto intro*: *sigma-sets.Empty sigma-sets-top*)

lemma *sigma-sets-single*[*simp*]: *sigma-sets* A $\{A\} = \{\{\}, A\}$
proof (*intro set-eqI iffI*)
fix x **assume** $x \in$ *sigma-sets* A $\{A\}$
then show $x \in \{\{\}, A\}$
by *induct blast+*
next
fix x **assume** $x \in \{\{\}, A\}$
then show $x \in$ *sigma-sets* A $\{A\}$
by (*auto intro*: *sigma-sets.Empty sigma-sets-top*)
qed

lemma *sigma-sets-sigma-sets-eq*:
$M \subseteq Pow\ S \Longrightarrow$ *sigma-sets* S (*sigma-sets* S M) = *sigma-sets* S M
by (*rule sigma-algebra.sigma-sets-eq*[*OF sigma-algebra-sigma-sets*, *of* M S]) *auto*

lemma *sigma-sets-singleton*:
assumes $X \subseteq S$
shows *sigma-sets* S $\{ X \}$ = $\{ \{\}, X, S - X, S \}$
proof −
interpret *sigma-algebra* S $\{ \{\}, X, S - X, S \}$
by (*rule sigma-algebra-single-set*) *fact*
have *sigma-sets* S $\{ X \} \subseteq$ *sigma-sets* S $\{ \{\}, X, S - X, S \}$
by (*rule sigma-sets-subseteq*) *simp*
moreover have ... = $\{ \{\}, X, S - X, S \}$
using *sigma-sets-eq* **by** *simp*
moreover
{ fix A **assume** $A \in \{ \{\}, X, S - X, S \}$
then have $A \in$ *sigma-sets* S $\{ X \}$
by (*auto intro*: *sigma-sets.intros*(2−) *sigma-sets-top*) **}**
ultimately have *sigma-sets* S $\{ X \}$ = *sigma-sets* S $\{ \{\}, X, S - X, S \}$
by (*intro antisym*) *auto*
with *sigma-sets-eq* **show** *?thesis* **by** *simp*
qed

lemma *restricted-sigma*:
assumes S: $S \in$ *sigma-sets* Ω M **and** M: $M \subseteq Pow\ \Omega$
shows *algebra.restricted-space* (*sigma-sets* Ω M) S =
sigma-sets S (*algebra.restricted-space* M S)
proof −
from S *sigma-sets-into-sp*[*OF* M]
have $S \in$ *sigma-sets* Ω M $S \subseteq \Omega$ **by** *auto*
from *sigma-sets-Int*[*OF this*]
show *?thesis* **by** *simp*
qed

lemma *sigma-sets-vimage-commute*:
assumes X: $X \in \Omega \to \Omega'$
shows $\{X -` A \cap \Omega \mid A.\ A \in$ *sigma-sets* $\Omega'\ M'\}$
$=$ *sigma-sets* $\Omega\ \{X -` A \cap \Omega \mid A.\ A \in M'\}$ (**is** *?L* $=$ *?R*)
proof
show *?L* $\subseteq$ *?R*
proof *clarify*
fix A **assume** $A \in$ *sigma-sets* $\Omega'\ M'$
then show $X -` A \cap \Omega \in$ *?R*
proof *induct*
case *Empty* **then show** *?case*
by (*auto intro*!: *sigma-sets.Empty*)
next
case (*Compl B*)
have [*simp*]: $X -` (\Omega' - B) \cap \Omega = \Omega - (X -` B \cap \Omega)$
by (*auto simp add*: *funcset-mem* [*OF X*])
with *Compl* **show** *?case*
by (*auto intro*!: *sigma-sets.Compl*)
next
case (*Union F*)
then show *?case*
by (*auto simp add*: *vimage-UN UN-extend-simps*(*4*) *simp del*: *UN-simps*
intro!: *sigma-sets.Union*)
qed *auto*
qed
show *?R* $\subseteq$ *?L*
proof *clarify*
fix A **assume** $A \in$ *?R*
then show $\exists B.\ A = X -` B \cap \Omega \wedge B \in$ *sigma-sets* $\Omega'\ M'$
proof *induct*
case (*Basic B*) **then show** *?case* **by** *auto*
next
case *Empty* **then show** *?case*
by (*auto intro*!: *sigma-sets.Empty exI*[*of* - {}])
next
case (*Compl B*)
then obtain A **where** A: $B = X -` A \cap \Omega$ $A \in$ *sigma-sets* $\Omega'\ M'$ **by** *auto*
then have [*simp*]: $\Omega - B = X -` (\Omega' - A) \cap \Omega$
by (*auto simp add*: *funcset-mem* [*OF X*])
with *A*(*2*) **show** *?case*
by (*auto intro*: *sigma-sets.Compl*)
next
case (*Union F*)
then have $\forall i.\ \exists B.\ F\ i = X -` B \cap \Omega \wedge B \in$ *sigma-sets* $\Omega'\ M'$ **by** *auto*
from *choice*[*OF this*] **guess** A **.. note** $A =$ *this*
with A **show** *?case*
by (*auto simp*: *vimage-UN*[*symmetric*] *intro*: *sigma-sets.Union*)
qed
qed

qed

lemma (**in** *ring-of-sets*) *UNION-in-sets*:
fixes *A*:: *nat* ⇒ *'a set*
assumes *A*: *range A* ⊆ *M*
shows (⋃ *i*∈{*0*..<*n*}. *A i*) ∈ *M*
proof (*induct n*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc n*)
thus *?case*
by (*simp add*: *atLeastLessThanSuc*) (*metis A Un UNIV-I image-subset-iff*)
qed

lemma (**in** *ring-of-sets*) *range-disjointed-sets*:
assumes *A*: *range A* ⊆ *M*
shows *range* (*disjointed A*) ⊆ *M*
proof (*auto simp add*: *disjointed-def*)
fix *n*
show *A n* − (⋃ *i*∈{*0*..<*n*}. *A i*) ∈ *M* **using** *UNION-in-sets*
by (*metis A Diff UNIV-I image-subset-iff*)
qed

lemma (**in** *algebra*) *range-disjointed-sets'*:
range A ⊆ *M* ⟹ *range* (*disjointed A*) ⊆ *M*
using *range-disjointed-sets* **.**

lemma *sigma-algebra-disjoint-iff*:
sigma-algebra Ω *M* ⟷ *algebra* Ω *M* ∧
(∀ *A*. *range A* ⊆ *M* ⟶ *disjoint-family A* ⟶ (⋃ *i*::*nat*. *A i*) ∈ *M*)
proof (*auto simp add*: *sigma-algebra-iff*)
fix *A* :: *nat* ⇒ *'a set*
assume *M*: *algebra* Ω *M*
and *A*: *range A* ⊆ *M*
and *UnA*: ∀ *A*. *range A* ⊆ *M* ⟶ *disjoint-family A* ⟶ (⋃ *i*::*nat*. *A i*) ∈ *M*
hence *range* (*disjointed A*) ⊆ *M* ⟶
disjoint-family (*disjointed A*) ⟶
(⋃ *i*. *disjointed A i*) ∈ *M* **by** *blast*
hence (⋃ *i*. *disjointed A i*) ∈ *M*
by (*simp add*: *algebra.range-disjointed-sets'*[*of* Ω] *M A disjoint-family-disjointed*)
thus (⋃ *i*::*nat*. *A i*) ∈ *M* **by** (*simp add*: *UN-disjointed-eq*)
qed

2.1.6 Ring generated by a semiring

definition (**in** *semiring-of-sets*)
generated-ring = { ⋃ *C* | *C*. *C* ⊆ *M* ∧ *finite C* ∧ *disjoint C* }

lemma (**in** *semiring-of-sets*) *generated-ringE*[*elim?*]:

assumes $a \in$ *generated-ring*
obtains C **where** *finite* C *disjoint* C $C \subseteq M$ $a = \bigcup C$
using *assms* **unfolding** *generated-ring-def* **by** *auto*

lemma (**in** *semiring-of-sets*) *generated-ringI*[*intro?*]:
assumes *finite* C *disjoint* C $C \subseteq M$ $a = \bigcup C$
shows $a \in$ *generated-ring*
using *assms* **unfolding** *generated-ring-def* **by** *auto*

lemma (**in** *semiring-of-sets*) *generated-ringI-Basic*:
$A \in M \Longrightarrow A \in$ *generated-ring*
by (*rule generated-ringI*[*of* $\{A\}$]) (*auto simp*: *disjoint-def*)

lemma (**in** *semiring-of-sets*) *generated-ring-disjoint-Un*[*intro*]:
assumes *a*: $a \in$ *generated-ring* **and** *b*: $b \in$ *generated-ring*
and $a \cap b = \{\}$
shows $a \cup b \in$ *generated-ring*
proof −
from *a* **guess** *Ca* **.. note** $Ca = this$
from *b* **guess** *Cb* **.. note** $Cb = this$
show *?thesis*
proof
show *disjoint* $(Ca \cup Cb)$
using ‹$a \cap b = \{\}$› Ca Cb **by** (*auto intro*!: *disjoint-union*)
qed (*insert Ca Cb*, *auto*)
qed

lemma (**in** *semiring-of-sets*) *generated-ring-empty*: $\{\} \in$ *generated-ring*
by (*auto simp*: *generated-ring-def disjoint-def*)

lemma (**in** *semiring-of-sets*) *generated-ring-disjoint-Union*:
assumes *finite* A **shows** $A \subseteq$ *generated-ring* $\Longrightarrow$ *disjoint* $A \Longrightarrow \bigcup A \in$ *generated-ring*
using *assms* **by** (*induct A*) (*auto simp*: *disjoint-def intro*!: *generated-ring-disjoint-Un generated-ring-empty*)

lemma (**in** *semiring-of-sets*) *generated-ring-disjoint-UNION*:
finite $I \Longrightarrow$ *disjoint* $(A \text{ ` } I) \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *generated-ring*$) \Longrightarrow$ *UNION* $I\ A \in$ *generated-ring*
by (*intro generated-ring-disjoint-Union*) *auto*

lemma (**in** *semiring-of-sets*) *generated-ring-Int*:
assumes *a*: $a \in$ *generated-ring* **and** *b*: $b \in$ *generated-ring*
shows $a \cap b \in$ *generated-ring*
proof −
from *a* **guess** *Ca* **.. note** $Ca = this$
from *b* **guess** *Cb* **.. note** $Cb = this$
def $C \equiv (\lambda(a,b).\ a \cap b)\text{`}\ (Ca \times Cb)$
show *?thesis*
proof

```
    show disjoint C
    proof (simp add: disjoint-def C-def, intro ballI impI)
      fix a1 b1 a2 b2 assume sets: a1 ∈ Ca b1 ∈ Cb a2 ∈ Ca b2 ∈ Cb
      assume a1 ∩ b1 ≠ a2 ∩ b2
      then have a1 ≠ a2 ∨ b1 ≠ b2 by auto
      then show (a1 ∩ b1) ∩ (a2 ∩ b2) = {}
      proof
        assume a1 ≠ a2
        with sets Ca have a1 ∩ a2 = {}
          by (auto simp: disjoint-def)
        then show ?thesis by auto
      next
        assume b1 ≠ b2
        with sets Cb have b1 ∩ b2 = {}
          by (auto simp: disjoint-def)
        then show ?thesis by auto
      qed
    qed
  qed (insert Ca Cb, auto simp: C-def)
qed

lemma (in semiring-of-sets) generated-ring-Inter:
  assumes finite A A ≠ {} shows A ⊆ generated-ring ⟹ ⋂A ∈ generated-ring
  using assms by (induct A rule: finite-ne-induct) (auto intro: generated-ring-Int)

lemma (in semiring-of-sets) generated-ring-INTER:
  finite I ⟹ I ≠ {} ⟹ (⋀i. i ∈ I ⟹ A i ∈ generated-ring) ⟹ INTER I A ∈
generated-ring
  by (intro generated-ring-Inter) auto

lemma (in semiring-of-sets) generating-ring:
  ring-of-sets Ω generated-ring
proof (rule ring-of-setsI)
  let ?R = generated-ring
  show ?R ⊆ Pow Ω
    using sets-into-space by (auto simp: generated-ring-def generated-ring-empty)
  show {} ∈ ?R by (rule generated-ring-empty)

  { fix a assume a: a ∈ ?R then guess Ca .. note Ca = this
    fix b assume b: b ∈ ?R then guess Cb .. note Cb = this

    show a - b ∈ ?R
    proof cases
      assume Cb = {} with Cb ‹a ∈ ?R› show ?thesis
        by simp
    next
      assume Cb ≠ {}
      with Ca Cb have a - b = (⋃a'∈Ca. ⋂b'∈Cb. a' - b') by auto
      also have ... ∈ ?R
```

```
    proof (intro generated-ring-INTER generated-ring-disjoint-UNION)
      fix a b assume a ∈ Ca b ∈ Cb
      with Ca Cb Diff-cover[of a b] show a − b ∈ ?R
        by (auto simp add: generated-ring-def)
          (metis DiffI Diff-eq-empty-iff empty-iff)
    next
      show disjoint ((λa'. ⋂b'∈Cb. a' − b')`Ca)
        using Ca by (auto simp add: disjoint-def ‹Cb ≠ {}›)
    next
      show finite Ca finite Cb Cb ≠ {} by fact+
    qed
    finally show a − b ∈ ?R .
  qed }
  note Diff = this

  fix a b assume sets: a ∈ ?R b ∈ ?R
  have a ∪ b = (a − b) ∪ (a ∩ b) ∪ (b − a) by auto
  also have ... ∈ ?R
    by (intro sets generated-ring-disjoint-Un generated-ring-Int Diff) auto
  finally show a ∪ b ∈ ?R .
qed

lemma (in semiring-of-sets) sigma-sets-generated-ring-eq: sigma-sets Ω generated-ring
= sigma-sets Ω M
proof
  interpret M: sigma-algebra Ω sigma-sets Ω M
    using space-closed by (rule sigma-algebra-sigma-sets)
  show sigma-sets Ω generated-ring ⊆ sigma-sets Ω M
    by (blast intro!: sigma-sets-mono elim: generated-ringE)
qed (auto intro!: generated-ringI-Basic sigma-sets-mono)
```

2.1.7 A Two-Element Series

```
definition binaryset :: 'a set ⇒ 'a set ⇒ nat ⇒ 'a set
  where binaryset A B = (λx. {})(0 := A, Suc 0 := B)

lemma range-binaryset-eq: range(binaryset A B) = {A,B,{}}
  apply (simp add: binaryset-def)
  apply (rule set-eqI)
  apply (auto simp add: image-iff)
  done

lemma UN-binaryset-eq: (⋃i. binaryset A B i) = A ∪ B
  by (simp add: range-binaryset-eq cong del: strong-SUP-cong)
```

2.1.8 Closed CDI

```
definition closed-cdi where
  closed-cdi Ω M ⟷
   M ⊆ Pow Ω &
```

$(\forall\, s \in M.\ \Omega - s \in M)$ &
$(\forall\, A.\ (range\ A \subseteq M)$ & $(A\ 0 = \{\})$ & $(\forall\, n.\ A\ n \subseteq A\ (Suc\ n)) \longrightarrow$
$(\bigcup i.\ A\ i) \in M)$ &
$(\forall\, A.\ (range\ A \subseteq M)$ & $disjoint\text{-}family\ A \longrightarrow (\bigcup i{::}nat.\ A\ i) \in M)$

inductive-set
smallest-ccdi-sets :: $'a\ set \Rightarrow 'a\ set\ set \Rightarrow 'a\ set\ set$
for $\Omega\ M$
where
Basic [*intro*]:
$a \in M \Longrightarrow a \in smallest\text{-}ccdi\text{-}sets\ \Omega\ M$
| *Compl* [*intro*]:
$a \in smallest\text{-}ccdi\text{-}sets\ \Omega\ M \Longrightarrow \Omega - a \in smallest\text{-}ccdi\text{-}sets\ \Omega\ M$
| *Inc*:
$range\ A \in Pow(smallest\text{-}ccdi\text{-}sets\ \Omega\ M) \Longrightarrow A\ 0 = \{\} \Longrightarrow (\bigwedge n.\ A\ n \subseteq A\ (Suc\ n))$
$\Longrightarrow (\bigcup i.\ A\ i) \in smallest\text{-}ccdi\text{-}sets\ \Omega\ M$
| *Disj*:
$range\ A \in Pow(smallest\text{-}ccdi\text{-}sets\ \Omega\ M) \Longrightarrow disjoint\text{-}family\ A$
$\Longrightarrow (\bigcup i{::}nat.\ A\ i) \in smallest\text{-}ccdi\text{-}sets\ \Omega\ M$

lemma (**in** *subset-class*) *smallest-closed-cdi1*: $M \subseteq smallest\text{-}ccdi\text{-}sets\ \Omega\ M$
by *auto*

lemma (**in** *subset-class*) *smallest-ccdi-sets*: $smallest\text{-}ccdi\text{-}sets\ \Omega\ M \subseteq Pow\ \Omega$
apply (*rule subsetI*)
apply (*erule smallest-ccdi-sets.induct*)
apply (*auto intro*: *range-subsetD dest*: *sets-into-space*)
done

lemma (**in** *subset-class*) *smallest-closed-cdi2*: $closed\text{-}cdi\ \Omega\ (smallest\text{-}ccdi\text{-}sets\ \Omega\ M)$
apply (*auto simp add*: *closed-cdi-def smallest-ccdi-sets*)
apply (*blast intro*: *smallest-ccdi-sets.Inc smallest-ccdi-sets.Disj*) +
done

lemma *closed-cdi-subset*: $closed\text{-}cdi\ \Omega\ M \Longrightarrow M \subseteq Pow\ \Omega$
by (*simp add*: *closed-cdi-def*)

lemma *closed-cdi-Compl*: $closed\text{-}cdi\ \Omega\ M \Longrightarrow s \in M \Longrightarrow \Omega - s \in M$
by (*simp add*: *closed-cdi-def*)

lemma *closed-cdi-Inc*:
$closed\text{-}cdi\ \Omega\ M \Longrightarrow range\ A \subseteq M \Longrightarrow A\ 0 = \{\} \Longrightarrow (!!n.\ A\ n \subseteq A\ (Suc\ n))$
$\Longrightarrow (\bigcup i.\ A\ i) \in M$
by (*simp add*: *closed-cdi-def*)

lemma *closed-cdi-Disj*:
$closed\text{-}cdi\ \Omega\ M \Longrightarrow range\ A \subseteq M \Longrightarrow disjoint\text{-}family\ A \Longrightarrow (\bigcup i{::}nat.\ A\ i) \in M$

by (*simp add*: *closed-cdi-def*)

lemma *closed-cdi-Un*:
assumes *cdi*: *closed-cdi* Ω M **and** *empty*: $\{\} \in M$
and A: $A \in M$ **and** B: $B \in M$
and *disj*: $A \cap B = \{\}$
shows $A \cup B \in M$
proof −
have *ra*: *range* (*binaryset A B*) $\subseteq M$
by (*simp add*: *range-binaryset-eq empty A B*)
have *di*: *disjoint-family* (*binaryset A B*) **using** *disj*
by (*simp add*: *disjoint-family-on-def binaryset-def Int-commute*)
from *closed-cdi-Disj* [*OF cdi ra di*]
show *?thesis*
by (*simp add*: *UN-binaryset-eq*)
qed

lemma (**in** *algebra*) *smallest-ccdi-sets-Un*:
assumes A: $A \in$ *smallest-ccdi-sets* Ω M **and** B: $B \in$ *smallest-ccdi-sets* Ω M
and *disj*: $A \cap B = \{\}$
shows $A \cup B \in$ *smallest-ccdi-sets* Ω M
proof −
have *ra*: *range* (*binaryset A B*) $\in$ *Pow* (*smallest-ccdi-sets* Ω M)
by (*simp add*: *range-binaryset-eq A B smallest-ccdi-sets.Basic*)
have *di*: *disjoint-family* (*binaryset A B*) **using** *disj*
by (*simp add*: *disjoint-family-on-def binaryset-def Int-commute*)
from *Disj* [*OF ra di*]
show *?thesis*
by (*simp add*: *UN-binaryset-eq*)
qed

lemma (**in** *algebra*) *smallest-ccdi-sets-Int1*:
assumes a: $a \in M$
shows $b \in$ *smallest-ccdi-sets* Ω $M \Longrightarrow a \cap b \in$ *smallest-ccdi-sets* Ω M
proof (*induct rule*: *smallest-ccdi-sets.induct*)
case (*Basic x*)
thus *?case*
by (*metis a Int smallest-ccdi-sets.Basic*)
next
case (*Compl x*)
have $a \cap (\Omega - x) = \Omega - ((\Omega - a) \cup (a \cap x))$
by *blast*
also have ... $\in$ *smallest-ccdi-sets* Ω M
by (*metis smallest-ccdi-sets.Compl a Compl*(2) *Diff-Int2 Diff-Int-distrib2*
Diff-disjoint Int-Diff Int-empty-right smallest-ccdi-sets-Un
smallest-ccdi-sets.Basic smallest-ccdi-sets.Compl)
finally show *?case* **.**
next
case (*Inc A*)

have *1*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) = a \cap (\bigcup i.\ A\ i)$
by *blast*
have *range* $(\lambda i.\ a \cap A\ i) \in Pow(\mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M)$ **using** *Inc*
by *blast*
moreover have $(\lambda i.\ a \cap A\ i)\ 0 = \{\}$
by (*simp add*: *Inc*)
moreover have $!!n.\ (\lambda i.\ a \cap A\ i)\ n \subseteq (\lambda i.\ a \cap A\ i)\ (Suc\ n)$ **using** *Inc*
by *blast*
ultimately have *2*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) \in \mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M$
by (*rule smallest-ccdi-sets.Inc*)
show *?case*
by (*metis 1 2*)
next
case (*Disj A*)
have *1*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) = a \cap (\bigcup i.\ A\ i)$
by *blast*
have *range* $(\lambda i.\ a \cap A\ i) \in Pow(\mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M)$ **using** *Disj*
by *blast*
moreover have *disjoint-family* $(\lambda i.\ a \cap A\ i)$ **using** *Disj*
by (*auto simp add*: *disjoint-family-on-def*)
ultimately have *2*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) \in \mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M$
by (*rule smallest-ccdi-sets.Disj*)
show *?case*
by (*metis 1 2*)
qed

lemma (**in** *algebra*) *smallest-ccdi-sets-Int*:
assumes *b*: $b \in \mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M$
shows $a \in \mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M \Longrightarrow a \cap b \in \mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M$
proof (*induct rule*: *smallest-ccdi-sets.induct*)
case (*Basic x*)
thus *?case*
by (*metis b smallest-ccdi-sets-Int1*)
next
case (*Compl x*)
have $(\Omega - x) \cap b = \Omega - (x \cap b \cup (\Omega - b))$
by *blast*
also have *...* $\in \mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M$
by (*metis Compl*(*2*) *Diff-disjoint Int-Diff Int-commute Int-empty-right b*
smallest-ccdi-sets.Compl smallest-ccdi-sets-Un)
finally show *?case* **.**
next
case (*Inc A*)
have *1*: $(\bigcup i.\ (\lambda i.\ A\ i \cap b)\ i) = (\bigcup i.\ A\ i) \cap b$
by *blast*
have *range* $(\lambda i.\ A\ i \cap b) \in Pow(\mathit{smallest\text{-}ccdi\text{-}sets}\ \Omega\ M)$ **using** *Inc*
by *blast*
moreover have $(\lambda i.\ A\ i \cap b)\ 0 = \{\}$

```
    by (simp add: Inc)
  moreover have !!n. (λi. A i ∩ b) n ⊆ (λi. A i ∩ b) (Suc n) using Inc
    by blast
  ultimately have 2: (⋃ i. (λi. A i ∩ b) i) ∈ smallest-ccdi-sets Ω M
    by (rule smallest-ccdi-sets.Inc)
  show ?case
    by (metis 1 2)
next
  case (Disj A)
  have 1: (⋃ i. (λi. A i ∩ b) i) = (⋃ i. A i) ∩ b
    by blast
  have range (λi. A i ∩ b) ∈ Pow(smallest-ccdi-sets Ω M) using Disj
    by blast
  moreover have disjoint-family (λi. A i ∩ b) using Disj
    by (auto simp add: disjoint-family-on-def)
  ultimately have 2: (⋃ i. (λi. A i ∩ b) i) ∈ smallest-ccdi-sets Ω M
    by (rule smallest-ccdi-sets.Disj)
  show ?case
    by (metis 1 2)
qed

lemma (in algebra) sigma-property-disjoint-lemma:
  assumes sbC: M ⊆ C
      and ccdi: closed-cdi Ω C
  shows sigma-sets Ω M ⊆ C
proof −
  have smallest-ccdi-sets Ω M ∈ {B . M ⊆ B ∧ sigma-algebra Ω B}
    apply (auto simp add: sigma-algebra-disjoint-iff algebra-iff-Int
          smallest-ccdi-sets-Int)
    apply (metis Union-Pow-eq Union-upper subsetD smallest-ccdi-sets)
    apply (blast intro: smallest-ccdi-sets.Disj)
    done
  hence sigma-sets (Ω) (M) ⊆ smallest-ccdi-sets Ω M
    by clarsimp
       (drule sigma-algebra.sigma-sets-subset [where a=M], auto)
  also have ... ⊆ C
    proof
      fix x
      assume x: x ∈ smallest-ccdi-sets Ω M
      thus x ∈ C
        proof (induct rule: smallest-ccdi-sets.induct)
          case (Basic x)
          thus ?case
            by (metis Basic subsetD sbC)
        next
          case (Compl x)
          thus ?case
            by (blast intro: closed-cdi-Compl [OF ccdi, simplified])
        next
```

```
      case (Inc A)
      thus ?case
        by (auto intro: closed-cdi-Inc [OF ccdi, simplified])
    next
      case (Disj A)
      thus ?case
        by (auto intro: closed-cdi-Disj [OF ccdi, simplified])
    qed
  qed
  finally show ?thesis .
qed

lemma (in algebra) sigma-property-disjoint:
  assumes sbC: M ⊆ C
    and compl: !!s. s ∈ C ∩ sigma-sets (Ω) (M) ⟹ Ω − s ∈ C
    and inc: !!A. range A ⊆ C ∩ sigma-sets (Ω) (M)
                ⟹ A 0 = {} ⟹ (!!n. A n ⊆ A (Suc n))
                ⟹ (⋃i. A i) ∈ C
    and disj: !!A. range A ⊆ C ∩ sigma-sets (Ω) (M)
                 ⟹ disjoint-family A ⟹ (⋃i::nat. A i) ∈ C
  shows sigma-sets (Ω) (M) ⊆ C
proof −
  have sigma-sets (Ω) (M) ⊆ C ∩ sigma-sets (Ω) (M)
    proof (rule sigma-property-disjoint-lemma)
      show M ⊆ C ∩ sigma-sets (Ω) (M)
        by (metis Int-greatest Set.subsetI sbC sigma-sets.Basic)
    next
      show closed-cdi Ω (C ∩ sigma-sets (Ω) (M))
        by (simp add: closed-cdi-def compl inc disj)
          (metis PowI Set.subsetI le-infI2 sigma-sets-into-sp space-closed
            IntE sigma-sets.Compl range-subsetD sigma-sets.Union)
    qed
  thus ?thesis
    by blast
qed
```

2.1.9 Dynkin systems

```
locale dynkin-system = subset-class +
  assumes space: Ω ∈ M
    and   compl[intro!]: ⋀A. A ∈ M ⟹ Ω − A ∈ M
    and   UN[intro!]: ⋀A. disjoint-family A ⟹ range A ⊆ M
                        ⟹ (⋃i::nat. A i) ∈ M

lemma (in dynkin-system) empty[intro, simp]: {} ∈ M
  using space compl[of Ω] by simp

lemma (in dynkin-system) diff:
  assumes sets: D ∈ M E ∈ M and D ⊆ E
```

shows $E - D \in M$
proof −
let *?f* = λx. *if* $x = 0$ *then* D *else if* $x = Suc\ 0$ *then* $\Omega - E$ *else* $\{\}$
have *range ?f* = $\{D, \Omega - E, \{\}\}$
by (*auto simp*: *image-iff*)
moreover have $D \cup (\Omega - E) = (\bigcup i.\ ?f\ i)$
by (*auto simp*: *image-iff split*: *if-split-asm*)
moreover
have *disjoint-family ?f* **unfolding** *disjoint-family-on-def*
using ‹$D \in M$›[*THEN sets-into-space*] ‹$D \subseteq E$› **by** *auto*
ultimately have $\Omega - (D \cup (\Omega - E)) \in M$
using *sets* **by** *auto*
also have $\Omega - (D \cup (\Omega - E)) = E - D$
using *assms sets-into-space* **by** *auto*
finally show *?thesis* **.**
qed

lemma *dynkin-systemI*:
assumes $\bigwedge A.\ A \in M \Longrightarrow A \subseteq \Omega\ \Omega \in M$
assumes $\bigwedge A.\ A \in M \Longrightarrow \Omega - A \in M$
assumes $\bigwedge A.\ disjoint\text{-}family\ A \Longrightarrow range\ A \subseteq M$
$\Longrightarrow (\bigcup i{::}nat.\ A\ i) \in M$
shows *dynkin-system* $\Omega\ M$
using *assms* **by** (*auto simp*: *dynkin-system-def dynkin-system-axioms-def subset-class-def*)

lemma *dynkin-systemI'*:
assumes *1*: $\bigwedge A.\ A \in M \Longrightarrow A \subseteq \Omega$
assumes *empty*: $\{\} \in M$
assumes *Diff*: $\bigwedge A.\ A \in M \Longrightarrow \Omega - A \in M$
assumes *2*: $\bigwedge A.\ disjoint\text{-}family\ A \Longrightarrow range\ A \subseteq M$
$\Longrightarrow (\bigcup i{::}nat.\ A\ i) \in M$
shows *dynkin-system* $\Omega\ M$
proof −
from *Diff*[*OF empty*] **have** $\Omega \in M$ **by** *auto*
from *1 this Diff 2* **show** *?thesis*
by (*intro dynkin-systemI*) *auto*
qed

lemma *dynkin-system-trivial*:
shows *dynkin-system* A ($Pow\ A$)
by (*rule dynkin-systemI*) *auto*

lemma *sigma-algebra-imp-dynkin-system*:
assumes *sigma-algebra* $\Omega\ M$ **shows** *dynkin-system* $\Omega\ M$
proof −
interpret *sigma-algebra* $\Omega\ M$ **by** *fact*
show *?thesis* **using** *sets-into-space* **by** (*fastforce intro*!: *dynkin-systemI*)
qed

2.1.10 Intersection sets systems

definition *Int-stable M* $\longleftrightarrow$ $(\forall\ a \in M.\ \forall\ b \in M.\ a \cap b \in M)$

lemma (**in** *algebra*) *Int-stable*: *Int-stable M*
unfolding *Int-stable-def* **by** *auto*

lemma *Int-stableI*:
$(\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow a \cap b \in A) \Longrightarrow$ *Int-stable A*
unfolding *Int-stable-def* **by** *auto*

lemma *Int-stableD*:
Int-stable M $\Longrightarrow a \in M \Longrightarrow b \in M \Longrightarrow a \cap b \in M$
unfolding *Int-stable-def* **by** *auto*

lemma (**in** *dynkin-system*) *sigma-algebra-eq-Int-stable*:
sigma-algebra Ω *M* $\longleftrightarrow$ *Int-stable M*
proof
assume *sigma-algebra* Ω *M* **then show** *Int-stable M*
unfolding *sigma-algebra-def* **using** *algebra.Int-stable* **by** *auto*
next
assume *Int-stable M*
show *sigma-algebra* Ω *M*
unfolding *sigma-algebra-disjoint-iff algebra-iff-Un*
proof (*intro conjI ballI allI impI*)
show $M \subseteq$ *Pow* (Ω) **using** *sets-into-space* **by** *auto*
next
fix *A B* **assume** $A \in M\ B \in M$
then have $A \cup B = \Omega - ((\Omega - A) \cap (\Omega - B))$
$\Omega - A \in M\ \Omega - B \in M$
using *sets-into-space* **by** *auto*
then show $A \cup B \in M$
using ‹*Int-stable M*› **unfolding** *Int-stable-def* **by** *auto*
qed *auto*
qed

2.1.11 Smallest Dynkin systems

definition *dynkin* **where**
dynkin Ω *M* = $(\bigcap \{D.\ dynkin\text{-}system\ \Omega\ D \wedge M \subseteq D\})$

lemma *dynkin-system-dynkin*:
assumes $M \subseteq$ *Pow* (Ω)
shows *dynkin-system* Ω (*dynkin* Ω *M*)
proof (*rule dynkin-systemI*)
fix *A* **assume** $A \in$ *dynkin* Ω *M*
moreover
{ fix *D* **assume** $A \in D$ **and** *d*: *dynkin-system* Ω *D*
then have $A \subseteq \Omega$ **by** (*auto simp*: *dynkin-system-def subset-class-def*) **}**
moreover have $\{D.\ dynkin\text{-}system\ \Omega\ D \wedge M \subseteq D\} \neq \{\}$

```
    using assms dynkin-system-trivial by fastforce
  ultimately show A ⊆ Ω
    unfolding dynkin-def using assms
    by auto
next
  show Ω ∈ dynkin Ω M
    unfolding dynkin-def using dynkin-system.space by fastforce
next
  fix A assume A ∈ dynkin Ω M
  then show Ω − A ∈ dynkin Ω M
    unfolding dynkin-def using dynkin-system.compl by force
next
  fix A :: nat ⇒ 'a set
  assume A: disjoint-family A range A ⊆ dynkin Ω M
  show (⋃ i. A i) ∈ dynkin Ω M unfolding dynkin-def
  proof (simp, safe)
    fix D assume dynkin-system Ω D M ⊆ D
    with A have (⋃ i. A i) ∈ D
      by (intro dynkin-system.UN) (auto simp: dynkin-def)
    then show (⋃ i. A i) ∈ D by auto
  qed
qed

lemma dynkin-Basic[intro]: A ∈ M ⟹ A ∈ dynkin Ω M
  unfolding dynkin-def by auto

lemma (in dynkin-system) restricted-dynkin-system:
  assumes D ∈ M
  shows dynkin-system Ω {Q. Q ⊆ Ω ∧ Q ∩ D ∈ M}
proof (rule dynkin-systemI, simp-all)
  have Ω ∩ D = D
    using ‹D ∈ M› sets-into-space by auto
  then show Ω ∩ D ∈ M
    using ‹D ∈ M› by auto
next
  fix A assume A ⊆ Ω ∧ A ∩ D ∈ M
  moreover have (Ω − A) ∩ D = (Ω − (A ∩ D)) − (Ω − D)
    by auto
  ultimately show Ω − A ⊆ Ω ∧ (Ω − A) ∩ D ∈ M
    using ‹D ∈ M› by (auto intro: diff)
next
  fix A :: nat ⇒ 'a set
  assume disjoint-family A range A ⊆ {Q. Q ⊆ Ω ∧ Q ∩ D ∈ M}
  then have ⋀i. A i ⊆ Ω disjoint-family (λi. A i ∩ D)
    range (λi. A i ∩ D) ⊆ M (⋃ x. A x) ∩ D = (⋃ x. A x ∩ D)
    by ((fastforce simp: disjoint-family-on-def)+)
  then show (⋃ x. A x) ⊆ Ω ∧ (⋃ x. A x) ∩ D ∈ M
    by (auto simp del: UN-simps)
qed
```

lemma (**in** *dynkin-system*) *dynkin-subset*:
assumes $N \subseteq M$
shows *dynkin* Ω $N \subseteq M$
proof −
have *dynkin-system* Ω M **..**
then have *dynkin-system* Ω M
using *assms* **unfolding** *dynkin-system-def dynkin-system-axioms-def subset-class-def*
by *simp*
with ‹$N \subseteq M$› **show** *?thesis* **by** (*auto simp add*: *dynkin-def*)
qed

lemma *sigma-eq-dynkin*:
assumes *sets*: $M \subseteq Pow\ \Omega$
assumes *Int-stable M*
shows *sigma-sets* Ω M = *dynkin* Ω M
proof −
have *dynkin* Ω $M \subseteq$ *sigma-sets* (Ω) (M)
using *sigma-algebra-imp-dynkin-system*
unfolding *dynkin-def sigma-sets-least-sigma-algebra*[*OF sets*] **by** *auto*
moreover
interpret *dynkin-system* Ω *dynkin* Ω M
using *dynkin-system-dynkin*[*OF sets*] **.**
have *sigma-algebra* Ω (*dynkin* Ω M)
unfolding *sigma-algebra-eq-Int-stable Int-stable-def*
proof (*intro ballI*)
fix A B **assume** $A \in$ *dynkin* Ω M $B \in$ *dynkin* Ω M
let *?D* = $\lambda E.\ \{Q.\ Q \subseteq \Omega \wedge Q \cap E \in$ *dynkin* Ω $M\}$
have $M \subseteq$ *?D B*
proof
fix E **assume** $E \in M$
then have $M \subseteq$ *?D E* $E \in$ *dynkin* Ω M
using *sets-into-space* ‹*Int-stable M*› **by** (*auto simp*: *Int-stable-def*)
then have *dynkin* Ω $M \subseteq$ *?D E*
using *restricted-dynkin-system* ‹$E \in$ *dynkin* Ω M›
by (*intro dynkin-system.dynkin-subset*) *simp-all*
then have $B \in$ *?D E*
using ‹$B \in$ *dynkin* Ω M› **by** *auto*
then have $E \cap B \in$ *dynkin* Ω M
by (*subst Int-commute*) *simp*
then show $E \in$ *?D B*
using *sets* ‹$E \in M$› **by** *auto*
qed
then have *dynkin* Ω $M \subseteq$ *?D B*
using *restricted-dynkin-system* ‹$B \in$ *dynkin* Ω M›
by (*intro dynkin-system.dynkin-subset*) *simp-all*
then show $A \cap B \in$ *dynkin* Ω M
using ‹$A \in$ *dynkin* Ω M› *sets-into-space* **by** *auto*
qed

from *sigma-algebra.sigma-sets-subset*[*OF this*, *of M*]
have *sigma-sets* (Ω) (*M*) ⊆ *dynkin* Ω *M* **by** *auto*
ultimately have *sigma-sets* (Ω) (*M*) = *dynkin* Ω *M* **by** *auto*
then show *?thesis*
by (*auto simp*: *dynkin-def*)
qed

lemma (**in** *dynkin-system*) *dynkin-idem*:
dynkin Ω *M* = *M*
proof −
have *dynkin* Ω *M* = *M*
proof
show *M* ⊆ *dynkin* Ω *M*
using *dynkin-Basic* **by** *auto*
show *dynkin* Ω *M* ⊆ *M*
by (*intro dynkin-subset*) *auto*
qed
then show *?thesis*
by (*auto simp*: *dynkin-def*)
qed

lemma (**in** *dynkin-system*) *dynkin-lemma*:
assumes *Int-stable E*
and *E*: *E* ⊆ *M M* ⊆ *sigma-sets* Ω *E*
shows *sigma-sets* Ω *E* = *M*
proof −
have *E* ⊆ *Pow* Ω
using *E sets-into-space* **by** *force*
then have ∗: *sigma-sets* Ω *E* = *dynkin* Ω *E*
using ‹*Int-stable E*› **by** (*rule sigma-eq-dynkin*)
then have *dynkin* Ω *E* = *M*
using *assms dynkin-subset*[*OF E*(*1*)] **by** *simp*
with ∗ **show** *?thesis*
using *assms* **by** (*auto simp*: *dynkin-def*)
qed

2.1.12 Induction rule for intersection-stable generators

The reason to introduce Dynkin-systems is the following induction rules for σ-algebras generated by a generator closed under intersection.

lemma *sigma-sets-induct-disjoint*[*consumes 3*, *case-names basic empty compl union*]:
assumes *Int-stable G*
and *closed*: *G* ⊆ *Pow* Ω
and *A*: *A* ∈ *sigma-sets* Ω *G*
assumes *basic*: ⋀*A*. *A* ∈ *G* ⟹ *P A*
and *empty*: *P* {}
and *compl*: ⋀*A*. *A* ∈ *sigma-sets* Ω *G* ⟹ *P A* ⟹ *P* (Ω − *A*)
and *union*: ⋀*A*. *disjoint-family A* ⟹ *range A* ⊆ *sigma-sets* Ω *G* ⟹ (⋀*i*. *P* (*A i*)) ⟹ *P* (⋃*i*::*nat*. *A i*)

shows *P A*
proof −
let *?D* = { *A* ∈ *sigma-sets* Ω *G*. *P A* }
interpret *sigma-algebra* Ω *sigma-sets* Ω *G*
using *closed* **by** (*rule sigma-algebra-sigma-sets*)
from *compl*[*OF - empty*] *closed* **have** *space*: *P* Ω **by** *simp*
interpret *dynkin-system* Ω *?D*
by *standard* (*auto dest*: *sets-into-space intro*!: *space compl union*)
have *sigma-sets* Ω *G* = *?D*
by (*rule dynkin-lemma*) (*auto simp*: *basic* ‹*Int-stable G*›)
with *A* **show** *?thesis* **by** *auto*
qed

2.2 Measure type

definition *positive* :: *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *bool* **where**
positive M μ ⟷ μ {} = *0*

definition *countably-additive* :: *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *bool* **where**
countably-additive M f ⟷ (∀ *A*. *range A* ⊆ *M* ⟶ *disjoint-family A* ⟶ (⋃ *i*. *A i*) ∈ *M* ⟶
(∑ *i*. *f* (*A i*)) = *f* (⋃ *i*. *A i*))

definition *measure-space* :: *'a set* ⇒ *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *bool* **where**
measure-space Ω *A* μ ⟷ *sigma-algebra* Ω *A* ∧ *positive A* μ ∧ *countably-additive A* μ

typedef *'a measure* = {(Ω::*'a set*, *A*, μ). (∀ *a*∈−*A*. μ *a* = *0*) ∧ *measure-space* Ω *A* μ }
proof
have *sigma-algebra UNIV* {{}, *UNIV*}
by (*auto simp*: *sigma-algebra-iff2*)
then show (*UNIV*, {{}, *UNIV*}, λ*A*. *0*) ∈ {(Ω, *A*, μ). (∀ *a*∈−*A*. μ *a* = *0*) ∧ *measure-space* Ω *A* μ}
by (*auto simp*: *measure-space-def positive-def countably-additive-def*)
qed

definition *space* :: *'a measure* ⇒ *'a set* **where**
space M = *fst* (*Rep-measure M*)

definition *sets* :: *'a measure* ⇒ *'a set set* **where**
sets M = *fst* (*snd* (*Rep-measure M*))

definition *emeasure* :: *'a measure* ⇒ *'a set* ⇒ *ennreal* **where**
emeasure M = *snd* (*snd* (*Rep-measure M*))

definition *measure* :: *'a measure* ⇒ *'a set* ⇒ *real* **where**
measure M A = *enn2real* (*emeasure M A*)

declare [[*coercion sets*]]

declare [[*coercion measure*]]

declare [[*coercion emeasure*]]

lemma *measure-space*: *measure-space* (*space M*) (*sets M*) (*emeasure M*)
by (*cases M*) (*auto simp*: *space-def sets-def emeasure-def Abs-measure-inverse*)

interpretation *sets*: *sigma-algebra space M sets M* **for** *M* :: *'a measure*
using *measure-space*[*of M*] **by** (*auto simp*: *measure-space-def*)

definition *measure-of* :: *'a set* ⇒ *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *'a measure*
where
measure-of Ω *A* μ = *Abs-measure* (Ω, *if A* ⊆ *Pow* Ω *then sigma-sets* Ω *A else* {{}, Ω},
λ*a*. *if a* ∈ *sigma-sets* Ω *A* ∧ *measure-space* Ω (*sigma-sets* Ω *A*) μ *then* μ *a else* 0)

abbreviation *sigma* Ω *A* ≡ *measure-of* Ω *A* (λ*x*. 0)

lemma *measure-space-0*: *A* ⊆ *Pow* Ω ⟹ *measure-space* Ω (*sigma-sets* Ω *A*) (λ*x*. 0)
unfolding *measure-space-def*
by (*auto intro*!: *sigma-algebra-sigma-sets simp*: *positive-def countably-additive-def*)

lemma *sigma-algebra-trivial*: *sigma-algebra* Ω {{}, Ω}
by *unfold-locales*(*fastforce intro*: *exI*[**where** *x*={{}}] *exI*[**where** *x*={Ω}])+

lemma *measure-space-0'*: *measure-space* Ω {{}, Ω} (λ*x*. 0)
by(*simp add*: *measure-space-def positive-def countably-additive-def sigma-algebra-trivial*)

lemma *measure-space-closed*:
assumes *measure-space* Ω *M* μ
shows *M* ⊆ *Pow* Ω
proof −
interpret *sigma-algebra* Ω *M* **using** *assms* **by**(*simp add*: *measure-space-def*)
show *?thesis* **by**(*rule space-closed*)
qed

lemma (**in** *ring-of-sets*) *positive-cong-eq*:
(⋀*a*. *a* ∈ *M* ⟹ μ' *a* = μ *a*) ⟹ *positive M* μ' = *positive M* μ
by (*auto simp add*: *positive-def*)

lemma (**in** *sigma-algebra*) *countably-additive-eq*:
(⋀*a*. *a* ∈ *M* ⟹ μ' *a* = μ *a*) ⟹ *countably-additive M* μ' = *countably-additive M* μ
unfolding *countably-additive-def*
by (*intro arg-cong*[**where** *f*=*All*] *ext*) (*auto simp add*: *countably-additive-def*

subset-eq)

lemma *measure-space-eq*:
assumes *closed*: $A \subseteq Pow\ \Omega$ **and** *eq*: $\bigwedge a.\ a \in sigma\text{-}sets\ \Omega\ A \Longrightarrow \mu\ a = \mu'\ a$
shows *measure-space* Ω (*sigma-sets* Ω A) μ = *measure-space* Ω (*sigma-sets* Ω A) μ'
proof −
interpret *sigma-algebra* Ω *sigma-sets* Ω A **using** *closed* **by** (*rule sigma-algebra-sigma-sets*)
from *positive-cong-eq*[*OF eq*, *of* $\lambda i.\ i$] *countably-additive-eq*[*OF eq*, *of* $\lambda i.\ i$]
show *?thesis*
by (*auto simp*: *measure-space-def*)
qed

lemma *measure-of-eq*:
assumes *closed*: $A \subseteq Pow\ \Omega$ **and** *eq*: $(\bigwedge a.\ a \in sigma\text{-}sets\ \Omega\ A \Longrightarrow \mu\ a = \mu'\ a)$
shows *measure-of* Ω A μ = *measure-of* Ω A μ'
proof −
have *measure-space* Ω (*sigma-sets* Ω A) μ = *measure-space* Ω (*sigma-sets* Ω A) μ'
using *assms* **by** (*rule measure-space-eq*)
with *eq* **show** *?thesis*
by (*auto simp add*: *measure-of-def intro*!: *arg-cong*[**where** f=*Abs-measure*])
qed

lemma
shows *space-measure-of-conv*: *space* (*measure-of* Ω A μ) = Ω (**is** *?space*)
and *sets-measure-of-conv*:
sets (*measure-of* Ω A μ) = (*if* $A \subseteq Pow\ \Omega$ *then sigma-sets* Ω A *else* $\{\{\}, \Omega\}$) (**is** *?sets*)
and *emeasure-measure-of-conv*:
emeasure (*measure-of* Ω A μ) =
($\lambda B.$ *if* $B \in sigma\text{-}sets\ \Omega\ A \wedge measure\text{-}space\ \Omega$ (*sigma-sets* Ω A) μ *then* $\mu\ B$ *else* 0) (**is** *?emeasure*)
proof −
have *?space* ∧ *?sets* ∧ *?emeasure*
proof(*cases measure-space* Ω (*sigma-sets* Ω A) μ)
case *True*
from *measure-space-closed*[*OF this*] *sigma-sets-superset-generator*[*of* A Ω]
have $A \subseteq Pow\ \Omega$ **by** *simp*
hence *measure-space* Ω (*sigma-sets* Ω A) μ = *measure-space* Ω (*sigma-sets* Ω A)
($\lambda a.$ *if* $a \in sigma\text{-}sets\ \Omega\ A$ *then* $\mu\ a$ *else* 0)
by(*rule measure-space-eq*) *auto*
with *True* ‹$A \subseteq Pow\ \Omega$› **show** *?thesis*
by(*simp add*: *measure-of-def space-def sets-def emeasure-def Abs-measure-inverse*)
next
case *False* **thus** *?thesis*
by(*cases* $A \subseteq Pow\ \Omega$)(*simp-all add*: *Abs-measure-inverse measure-of-def sets-def space-def emeasure-def measure-space-0 measure-space-0'*)

qed
thus *?space ?sets ?emeasure* **by** *simp-all*
qed

lemma [*simp*]:
assumes *A*: $A \subseteq Pow\ \Omega$
shows *sets-measure-of*: $sets\ (measure\text{-}of\ \Omega\ A\ \mu) = sigma\text{-}sets\ \Omega\ A$
and *space-measure-of*: $space\ (measure\text{-}of\ \Omega\ A\ \mu) = \Omega$
using *assms*
by(*simp-all add*: *sets-measure-of-conv space-measure-of-conv*)

lemma (**in** *sigma-algebra*) *sets-measure-of-eq*[*simp*]: $sets\ (measure\text{-}of\ \Omega\ M\ \mu) = M$
using *space-closed* **by** (*auto intro*!: *sigma-sets-eq*)

lemma (**in** *sigma-algebra*) *space-measure-of-eq*[*simp*]: $space\ (measure\text{-}of\ \Omega\ M\ \mu) = \Omega$
by (*rule space-measure-of-conv*)

lemma *measure-of-subset*: $M \subseteq Pow\ \Omega \Longrightarrow M' \subseteq M \Longrightarrow sets\ (measure\text{-}of\ \Omega\ M'\ \mu) \subseteq sets\ (measure\text{-}of\ \Omega\ M\ \mu')$
by (*auto intro*!: *sigma-sets-subseteq*)

lemma *emeasure-sigma*: $emeasure\ (sigma\ \Omega\ A) = (\lambda x.\ 0)$
unfolding *measure-of-def emeasure-def*
by (*subst Abs-measure-inverse*)
(*auto simp*: *measure-space-def positive-def countably-additive-def*
intro!: *sigma-algebra-sigma-sets sigma-algebra-trivial*)

lemma *sigma-sets-mono''*:
assumes $A \in sigma\text{-}sets\ C\ D$
assumes $B \subseteq D$
assumes $D \subseteq Pow\ C$
shows $sigma\text{-}sets\ A\ B \subseteq sigma\text{-}sets\ C\ D$
proof
fix *x* **assume** $x \in sigma\text{-}sets\ A\ B$
thus $x \in sigma\text{-}sets\ C\ D$
proof *induct*
case (*Basic a*) **with** *assms* **have** $a \in D$ **by** *auto*
thus *?case* **..**
next
case *Empty* **show** *?case* **by** (*rule sigma-sets.Empty*)
next
from *assms* **have** $A \in sets\ (sigma\ C\ D)$ **by** (*subst sets-measure-of*[*OF* ‹$D \subseteq Pow\ C$›])
moreover case (*Compl a*) **hence** $a \in sets\ (sigma\ C\ D)$ **by** (*subst sets-measure-of*[*OF* ‹$D \subseteq Pow\ C$›])
ultimately have $A - a \in sets\ (sigma\ C\ D)$ **..**
thus *?case* **by** (*subst* (*asm*) *sets-measure-of*[*OF* ‹$D \subseteq Pow\ C$›])

next
case (*Union a*)
thus *?case* **by** (*intro sigma-sets.Union*)
qed
qed

lemma *in-measure-of*[*intro*, *simp*]: $M \subseteq Pow\ \Omega \Longrightarrow A \in M \Longrightarrow A \in sets\ (measure\text{-}of\ \Omega\ M\ \mu)$
by *auto*

lemma *space-empty-iff*: $space\ N = \{\} \longleftrightarrow sets\ N = \{\{\}\}$
by (*metis Pow-empty Sup-bot-conv*(*1*) *cSup-singleton empty-iff*
sets.sigma-sets-eq sets.space-closed sigma-sets-top subset-singletonD)

2.2.1 Constructing simple *'a measure*

lemma *emeasure-measure-of*:
assumes *M*: $M = measure\text{-}of\ \Omega\ A\ \mu$
assumes *ms*: $A \subseteq Pow\ \Omega\ positive\ (sets\ M)\ \mu\ countably\text{-}additive\ (sets\ M)\ \mu$
assumes *X*: $X \in sets\ M$
shows $emeasure\ M\ X = \mu\ X$
proof −
interpret *sigma-algebra* Ω *sigma-sets* $\Omega\ A$ **by** (*rule sigma-algebra-sigma-sets*) *fact*
have $measure\text{-}space\ \Omega\ (sigma\text{-}sets\ \Omega\ A)\ \mu$
using *ms M* **by** (*simp add*: *measure-space-def sigma-algebra-sigma-sets*)
thus *?thesis* **using** *X ms*
by(*simp add*: *M emeasure-measure-of-conv sets-measure-of-conv*)
qed

lemma *emeasure-measure-of-sigma*:
assumes *ms*: *sigma-algebra* Ω *M positive M* μ *countably-additive M* μ
assumes *A*: $A \in M$
shows $emeasure\ (measure\text{-}of\ \Omega\ M\ \mu)\ A = \mu\ A$
proof −
interpret *sigma-algebra* Ω *M* **by** *fact*
have $measure\text{-}space\ \Omega\ (sigma\text{-}sets\ \Omega\ M)\ \mu$
using *ms sigma-sets-eq* **by** (*simp add*: *measure-space-def*)
thus *?thesis* **by**(*simp add*: *emeasure-measure-of-conv A*)
qed

lemma *measure-cases*[*cases type*: *measure*]:
obtains (*measure*) $\Omega\ A\ \mu$ **where** $x = Abs\text{-}measure\ (\Omega, A, \mu)\ \forall a \in -A.\ \mu\ a = 0\ measure\text{-}space\ \Omega\ A\ \mu$
by *atomize-elim* (*cases x*, *auto*)

lemma *sets-le-imp-space-le*: $sets\ A \subseteq sets\ B \Longrightarrow space\ A \subseteq space\ B$
by (*auto dest*: *sets.sets-into-space*)

lemma *sets-eq-imp-space-eq*: *sets M = sets M' ⟹ space M = space M'*
by (*auto intro*!: *antisym sets-le-imp-space-le*)

lemma *emeasure-notin-sets*: *A ∉ sets M ⟹ emeasure M A = 0*
by (*cases M*) (*auto simp*: *sets-def emeasure-def Abs-measure-inverse measure-space-def*)

lemma *emeasure-neq-0-sets*: *emeasure M A ≠ 0 ⟹ A ∈ sets M*
using *emeasure-notin-sets*[*of A M*] **by** *blast*

lemma *measure-notin-sets*: *A ∉ sets M ⟹ measure M A = 0*
by (*simp add*: *measure-def emeasure-notin-sets zero-ennreal.rep-eq*)

lemma *measure-eqI*:
fixes *M N* :: *'a measure*
assumes *sets M = sets N* **and** *eq*: *⋀A. A ∈ sets M ⟹ emeasure M A = emeasure N A*
shows *M = N*
proof (*cases M N rule*: *measure-cases*[*case-product measure-cases*])
case (*measure-measure Ω A μ Ω' A' μ'*)
interpret *M*: *sigma-algebra Ω A* **using** *measure-measure* **by** (*auto simp*: *measure-space-def*)
interpret *N*: *sigma-algebra Ω' A'* **using** *measure-measure* **by** (*auto simp*: *measure-space-def*)
have *A = sets M A' = sets N*
using *measure-measure* **by** (*simp-all add*: *sets-def Abs-measure-inverse*)
with ‹*sets M = sets N*› **have** *AA'*: *A = A'* **by** *simp*
moreover from *M.top N.top M.space-closed N.space-closed AA'* **have** *Ω = Ω'*
by *auto*
moreover { **fix** *B* **have** *μ B = μ' B*
proof *cases*
assume *B ∈ A*
with *eq* ‹*A = sets M*› **have** *emeasure M B = emeasure N B* **by** *simp*
with *measure-measure* **show** *μ B = μ' B*
by (*simp add*: *emeasure-def Abs-measure-inverse*)
next
assume *B ∉ A*
with ‹*A = sets M*› ‹*A' = sets N*› ‹*A = A'*› **have** *B ∉ sets M B ∉ sets N*
by *auto*
then have *emeasure M B = 0 emeasure N B = 0*
by (*simp-all add*: *emeasure-notin-sets*)
with *measure-measure* **show** *μ B = μ' B*
by (*simp add*: *emeasure-def Abs-measure-inverse*)
qed }
then have *μ = μ'* **by** *auto*
ultimately show *M = N*
by (*simp add*: *measure-measure*)
qed

lemma *sigma-eqI*:
assumes [*simp*]: *M ⊆ Pow Ω N ⊆ Pow Ω sigma-sets Ω M = sigma-sets Ω N*
shows *sigma Ω M = sigma Ω N*

by (*rule measure-eqI*) (*simp-all add*: *emeasure-sigma*)

2.2.2 Measurable functions

definition *measurable* :: *'a measure* ⇒ *'b measure* ⇒ (*'a* ⇒ *'b*) *set* (**infixr** $\rightarrow_M$ *60*) **where**
measurable A B = {*f* ∈ *space A* → *space B*. ∀ *y* ∈ *sets B*. *f* −‘ *y* ∩ *space A* ∈ *sets A*}

lemma *measurableI*:
(⋀*x*. *x* ∈ *space M* ⟹ *f x* ∈ *space N*) ⟹ (⋀*A*. *A* ∈ *sets N* ⟹ *f* −‘ *A* ∩ *space M* ∈ *sets M*) ⟹
f ∈ *measurable M N*
by (*auto simp*: *measurable-def*)

lemma *measurable-space*:
f ∈ *measurable M A* ⟹ *x* ∈ *space M* ⟹ *f x* ∈ *space A*
unfolding *measurable-def* **by** *auto*

lemma *measurable-sets*:
f ∈ *measurable M A* ⟹ *S* ∈ *sets A* ⟹ *f* −‘ *S* ∩ *space M* ∈ *sets M*
unfolding *measurable-def* **by** *auto*

lemma *measurable-sets-Collect*:
assumes *f*: *f* ∈ *measurable M N* **and** *P*: {*x*∈*space N*. *P x*} ∈ *sets N* **shows** {*x*∈*space M*. *P* (*f x*)} ∈ *sets M*
proof −
have *f* −‘ {*x* ∈ *space N*. *P x*} ∩ *space M* = {*x*∈*space M*. *P* (*f x*)}
using *measurable-space*[*OF f*] **by** *auto*
with *measurable-sets*[*OF f P*] **show** *?thesis*
by *simp*
qed

lemma *measurable-sigma-sets*:
assumes *B*: *sets N* = *sigma-sets* Ω *A* *A* ⊆ *Pow* Ω
and *f*: *f* ∈ *space M* → Ω
and *ba*: ⋀*y*. *y* ∈ *A* ⟹ (*f* −‘ *y*) ∩ *space M* ∈ *sets M*
shows *f* ∈ *measurable M N*
proof −
interpret *A*: *sigma-algebra* Ω *sigma-sets* Ω *A* **using** *B*(*2*) **by** (*rule sigma-algebra-sigma-sets*)
from *B sets.top*[*of N*] *A.top sets.space-closed*[*of N*] *A.space-closed* **have** Ω: Ω = *space N* **by** *force*

{ **fix** *X* **assume** *X* ∈ *sigma-sets* Ω *A*
then have *f* −‘ *X* ∩ *space M* ∈ *sets M* ∧ *X* ⊆ Ω
proof *induct*
case (*Basic a*) **then show** *?case*
by (*auto simp add*: *ba*) (*metis B*(*2*) *subsetD PowD*)
next

```
      case (Compl a)
      have [simp]: f −‘ Ω ∩ space M = space M
        by (auto simp add: funcset-mem [OF f])
      then show ?case
        by (auto simp add: vimage-Diff Diff-Int-distrib2 sets.compl-sets Compl)
    next
      case (Union a)
      then show ?case
        by (simp add: vimage-UN, simp only: UN-extend-simps(4)) blast
    qed auto }
  with f show ?thesis
    by (auto simp add: measurable-def B Ω)
qed

lemma measurable-measure-of:
  assumes B: N ⊆ Pow Ω
      and f: f ∈ space M → Ω
      and ba: ⋀y. y ∈ N ⟹ (f −‘ y) ∩ space M ∈ sets M
  shows f ∈ measurable M (measure-of Ω N μ)
proof −
  have sets (measure-of Ω N μ) = sigma-sets Ω N
    using B by (rule sets-measure-of)
  from this assms show ?thesis by (rule measurable-sigma-sets)
qed

lemma measurable-iff-measure-of:
  assumes N ⊆ Pow Ω f ∈ space M → Ω
  shows f ∈ measurable M (measure-of Ω N μ) ⟷ (∀ A∈N. f −‘ A ∩ space M
∈ sets M)
  by (metis assms in-measure-of measurable-measure-of assms measurable-sets)

lemma measurable-cong-sets:
  assumes sets: sets M = sets M' sets N = sets N'
  shows measurable M N = measurable M' N'
  using sets[THEN sets-eq-imp-space-eq] sets by (simp add: measurable-def)

lemma measurable-cong:
  assumes ⋀w. w ∈ space M ⟹ f w = g w
  shows f ∈ measurable M M' ⟷ g ∈ measurable M M'
  unfolding measurable-def using assms
  by (simp cong: vimage-inter-cong Pi-cong)

lemma measurable-cong':
  assumes ⋀w. w ∈ space M =simp=> f w = g w
  shows f ∈ measurable M M' ⟷ g ∈ measurable M M'
  unfolding measurable-def using assms
  by (simp cong: vimage-inter-cong Pi-cong add: simp-implies-def)

lemma measurable-cong-strong:
```

$M = N \Longrightarrow M' = N' \Longrightarrow (\bigwedge w.\ w \in space\ M \Longrightarrow f\ w = g\ w) \Longrightarrow$
$f \in measurable\ M\ M' \longleftrightarrow g \in measurable\ N\ N'$
by (*metis measurable-cong*)

lemma *measurable-compose*:
assumes *f*: $f \in measurable\ M\ N$ **and** *g*: $g \in measurable\ N\ L$
shows $(\lambda x.\ g\ (f\ x)) \in measurable\ M\ L$
proof −
have $\bigwedge A.\ (\lambda x.\ g\ (f\ x))\ -`\ A \cap space\ M = f\ -`\ (g\ -`\ A \cap space\ N) \cap space\ M$
using *measurable-space*[*OF f*] **by** *auto*
with *measurable-space*[*OF f*] *measurable-space*[*OF g*] **show** *?thesis*
by (*auto intro*: *measurable-sets*[*OF f*] *measurable-sets*[*OF g*]
simp del: *vimage-Int simp add*: *measurable-def*)
qed

lemma *measurable-comp*:
$f \in measurable\ M\ N \Longrightarrow g \in measurable\ N\ L \Longrightarrow g \circ f \in measurable\ M\ L$
using *measurable-compose*[*of f M N g L*] **by** (*simp add*: *comp-def*)

lemma *measurable-const*:
$c \in space\ M' \Longrightarrow (\lambda x.\ c) \in measurable\ M\ M'$
by (*auto simp add*: *measurable-def*)

lemma *measurable-ident*: $id \in measurable\ M\ M$
by (*auto simp add*: *measurable-def*)

lemma *measurable-id*: $(\lambda x.\ x) \in measurable\ M\ M$
by (*simp add*: *measurable-def*)

lemma *measurable-ident-sets*:
assumes *eq*: $sets\ M = sets\ M'$ **shows** $(\lambda x.\ x) \in measurable\ M\ M'$
using *measurable-ident*[*of M*]
unfolding *id-def measurable-def eq sets-eq-imp-space-eq*[*OF eq*] **.**

lemma *sets-Least*:
assumes *meas*: $\bigwedge i{::}nat.\ \{x{\in}space\ M.\ P\ i\ x\} \in M$
shows $(\lambda x.\ LEAST\ j.\ P\ j\ x)\ -`\ A \cap space\ M \in sets\ M$
proof −
{ fix *i* **have** $(\lambda x.\ LEAST\ j.\ P\ j\ x)\ -`\ \{i\} \cap space\ M \in sets\ M$
proof *cases*
assume *i*: $(LEAST\ j.\ False) = i$
have $(\lambda x.\ LEAST\ j.\ P\ j\ x)\ -`\ \{i\} \cap space\ M =$
$\{x{\in}space\ M.\ P\ i\ x\} \cap (space\ M - (\bigcup j{<}i.\ \{x{\in}space\ M.\ P\ j\ x\})) \cup (space\ M - (\bigcup i.\ \{x{\in}space\ M.\ P\ i\ x\}))$
by (*simp add*: *set-eq-iff*, *safe*)
(*insert i*, *auto dest*: *Least-le intro*: *LeastI intro*!: *Least-equality*)
with *meas* **show** *?thesis*
by (*auto intro*!: *sets.Int*)
next

```
    assume i: (LEAST j. False) ≠ i
    then have (λx. LEAST j. P j x) -` {i} ∩ space M =
      {x∈space M. P i x} ∩ (space M - (⋃j<i. {x∈space M. P j x}))
    proof (simp add: set-eq-iff, safe)
      fix x assume neq: (LEAST j. False) ≠ (LEAST j. P j x)
      have ∃j. P j x
        by (rule ccontr) (insert neq, auto)
      then show P (LEAST j. P j x) x by (rule LeastI-ex)
    qed (auto dest: Least-le intro!: Least-equality)
    with meas show ?thesis
      by auto
  qed }
 then have (⋃i∈A. (λx. LEAST j. P j x) -` {i} ∩ space M) ∈ sets M
  by (intro sets.countable-UN) auto
 moreover have (⋃i∈A. (λx. LEAST j. P j x) -` {i} ∩ space M) =
  (λx. LEAST j. P j x) -` A ∩ space M by auto
 ultimately show ?thesis by auto
qed

lemma measurable-mono1:
 M' ⊆ Pow Ω ⟹ M ⊆ M' ⟹
   measurable (measure-of Ω M μ) N ⊆ measurable (measure-of Ω M' μ') N
 using measure-of-subset[of M' Ω M] by (auto simp add: measurable-def)
```

2.2.3 Counting space

```
definition count-space :: 'a set ⇒ 'a measure where
 count-space Ω = measure-of Ω (Pow Ω) (λA. if finite A then of-nat (card A) else
∞)

lemma
 shows space-count-space[simp]: space (count-space Ω) = Ω
   and sets-count-space[simp]: sets (count-space Ω) = Pow Ω
 using sigma-sets-into-sp[of Pow Ω Ω]
 by (auto simp: count-space-def)

lemma measurable-count-space-eq1[simp]:
 f ∈ measurable (count-space A) M ⟷ f ∈ A → space M
 unfolding measurable-def by simp

lemma measurable-compose-countable':
 assumes f: ⋀i. i ∈ I ⟹ (λx. f i x) ∈ measurable M N
 and g: g ∈ measurable M (count-space I) and I: countable I
 shows (λx. f (g x) x) ∈ measurable M N
 unfolding measurable-def
proof safe
 fix x assume x ∈ space M then show f (g x) x ∈ space N
   using measurable-space[OF f] g[THEN measurable-space] by auto
next
```

fix A **assume** A: $A \in sets\ N$
have $(\lambda x.\ f\ (g\ x)\ x) -` A \cap space\ M = (\bigcup i{\in}I.\ (g -` \{i\} \cap space\ M) \cap (f\ i -` A \cap space\ M))$
using *measurable-space*[*OF g*] **by** *auto*
also have $\ldots \in sets\ M$
using *f*[*THEN measurable-sets*, *OF - A*] *g*[*THEN measurable-sets*]
by (*auto intro*!: *sets.countable-UN' I intro*: *sets.Int*[*OF measurable-sets measurable-sets*])
finally show $(\lambda x.\ f\ (g\ x)\ x) -` A \cap space\ M \in sets\ M$ **.**
qed

lemma *measurable-count-space-eq-countable*:
assumes *countable A*
shows $f \in measurable\ M\ (count\text{-}space\ A) \longleftrightarrow (f \in space\ M \to A \wedge (\forall a{\in}A.\ f -` \{a\} \cap space\ M \in sets\ M))$
proof −
{ fix X **assume** $X \subseteq A\ f \in space\ M \to A$
with ‹*countable A*› **have** $f -` X \cap space\ M = (\bigcup a{\in}X.\ f -` \{a\} \cap space\ M)$ *countable X*
by (*auto dest*: *countable-subset*)
moreover assume $\forall a{\in}A.\ f -` \{a\} \cap space\ M \in sets\ M$
ultimately have $f -` X \cap space\ M \in sets\ M$
using ‹$X \subseteq A$› **by** (*auto intro*!: *sets.countable-UN' simp del*: *UN-simps*) **}**
then show *?thesis*
unfolding *measurable-def* **by** *auto*
qed

lemma *measurable-count-space-eq2*:
finite A $\Longrightarrow f \in measurable\ M\ (count\text{-}space\ A) \longleftrightarrow (f \in space\ M \to A \wedge (\forall a{\in}A.\ f -` \{a\} \cap space\ M \in sets\ M))$
by (*intro measurable-count-space-eq-countable countable-finite*)

lemma *measurable-count-space-eq2-countable*:
fixes $f :: 'a => 'c{::}countable$
shows $f \in measurable\ M\ (count\text{-}space\ A) \longleftrightarrow (f \in space\ M \to A \wedge (\forall a{\in}A.\ f -` \{a\} \cap space\ M \in sets\ M))$
by (*intro measurable-count-space-eq-countable countableI-type*)

lemma *measurable-compose-countable*:
assumes *f*: $\bigwedge i{::}'i{::}countable.\ (\lambda x.\ f\ i\ x) \in measurable\ M\ N$ **and** *g*: $g \in measurable\ M\ (count\text{-}space\ UNIV)$
shows $(\lambda x.\ f\ (g\ x)\ x) \in measurable\ M\ N$
by (*rule measurable-compose-countable'*[*OF assms*]) *auto*

lemma *measurable-count-space-const*:
$(\lambda x.\ c) \in measurable\ M\ (count\text{-}space\ UNIV)$
by (*simp add*: *measurable-const*)

lemma *measurable-count-space*:
$f \in measurable\ (count\text{-}space\ A)\ (count\text{-}space\ UNIV)$

by *simp*

lemma *measurable-compose-rev*:
assumes *f*: $f \in$ *measurable L N* **and** *g*: $g \in$ *measurable M L*
shows $(\lambda x.\ f\ (g\ x)) \in$ *measurable M N*
using *measurable-compose*[*OF g f*] **.**

lemma *measurable-empty-iff*:
space N $= \{\} \Longrightarrow f \in$ *measurable M N* $\longleftrightarrow$ *space M* $= \{\}$
by (*auto simp add*: *measurable-def Pi-iff*)

2.2.4 Extend measure

definition *extend-measure* $\Omega\ I\ G\ \mu =$
(*if* $(\exists \mu'.\ (\forall i \in I.\ \mu'\ (G\ i) = \mu\ i) \wedge$ *measure-space* Ω (*sigma-sets* Ω (*G*‘*I*)) $\mu') \wedge \neg\ (\forall i \in I.\ \mu\ i = 0)$
then measure-of Ω (*G*‘*I*) (*SOME* $\mu'.\ (\forall i \in I.\ \mu'\ (G\ i) = \mu\ i) \wedge$ *measure-space* Ω (*sigma-sets* Ω (*G*‘*I*)) μ')
else measure-of Ω (*G*‘*I*) (λ-. *0*))

lemma *space-extend-measure*: *G* ‘ *I* $\subseteq$ *Pow* $\Omega \Longrightarrow$ *space* (*extend-measure* Ω *I G* μ) $= \Omega$
unfolding *extend-measure-def* **by** *simp*

lemma *sets-extend-measure*: *G* ‘ *I* $\subseteq$ *Pow* $\Omega \Longrightarrow$ *sets* (*extend-measure* Ω *I G* μ) $=$ *sigma-sets* Ω (*G*‘*I*)
unfolding *extend-measure-def* **by** *simp*

lemma *emeasure-extend-measure*:
assumes *M*: $M =$ *extend-measure* Ω *I G* μ
and *eq*: $\bigwedge i.\ i \in I \Longrightarrow \mu'\ (G\ i) = \mu\ i$
and *ms*: *G* ‘ *I* $\subseteq$ *Pow* Ω *positive* (*sets M*) μ' *countably-additive* (*sets M*) μ'
and $i \in I$
shows *emeasure M* (*G i*) $= \mu\ i$
proof *cases*
assume $*$: $(\forall i \in I.\ \mu\ i = 0)$
with *M* **have** *M-eq*: $M =$ *measure-of* Ω (*G*‘*I*) (λ-. *0*)
by (*simp add*: *extend-measure-def*)
from *measure-space-0*[*OF ms*(*1*)] *ms* ‹*i*∈*I*›
have *emeasure M* (*G i*) $= 0$
by (*intro emeasure-measure-of*[*OF M-eq*]) (*auto simp add*: *M measure-space-def sets-extend-measure*)
with ‹*i*∈*I*› $*$ **show** *?thesis*
by *simp*
next
def $P \equiv \lambda\mu'.\ (\forall i \in I.\ \mu'\ (G\ i) = \mu\ i) \wedge$ *measure-space* Ω (*sigma-sets* Ω (*G*‘*I*)) μ'
assume $\neg\ (\forall i \in I.\ \mu\ i = 0)$
moreover
have *measure-space* (*space M*) (*sets M*) μ'

using *ms* **unfolding** *measure-space-def* **by** *auto standard*
with *ms eq* **have** $\exists \mu'.\ P\ \mu'$
unfolding *P-def*
by (*intro exI*[*of - μ'*]) (*auto simp add*: *M space-extend-measure sets-extend-measure*)
ultimately have *M-eq*: M = *measure-of* Ω (*G'I*) (*Eps P*)
by (*simp add*: *M extend-measure-def P-def*[*symmetric*])

from ‹$\exists \mu'.\ P\ \mu'$› **have** *P*: *P* (*Eps P*) **by** (*rule someI-ex*)
show *emeasure M* (*G i*) = *μ i*
proof (*subst emeasure-measure-of*[*OF M-eq*])
have *sets-M*: *sets M* = *sigma-sets* Ω (*G'I*)
using *M-eq ms* **by** (*auto simp*: *sets-extend-measure*)
then show *G i* ∈ *sets M* **using** ‹*i* ∈ *I*› **by** *auto*
show *positive* (*sets M*) (*Eps P*) *countably-additive* (*sets M*) (*Eps P*) *Eps P* (*G i*) = *μ i*
using *P* ‹*i*∈*I*› **by** (*auto simp add*: *sets-M measure-space-def P-def*)
qed *fact*
qed

lemma *emeasure-extend-measure-Pair*:
assumes *M*: *M* = *extend-measure* Ω {(*i*, *j*). *I i j*} (λ(*i*, *j*). *G i j*) (λ(*i*, *j*). *μ i j*)
and *eq*: ⋀*i j*. *I i j* ⟹ *μ'* (*G i j*) = *μ i j*
and *ms*: ⋀*i j*. *I i j* ⟹ *G i j* ∈ *Pow* Ω *positive* (*sets M*) *μ'* *countably-additive* (*sets M*) *μ'*
and *I i j*
shows *emeasure M* (*G i j*) = *μ i j*
using *emeasure-extend-measure*[*OF M - - ms*(*2*,*3*), *of* (*i*,*j*)] *eq ms*(*1*) ‹*I i j*›
by (*auto simp*: *subset-eq*)

2.2.5 Supremum of a set of σ-algebras

definition *Sup-sigma M* = *sigma* (⋃*x*∈*M*. *space x*) (⋃*x*∈*M*. *sets x*)

syntax
-SUP-sigma :: *pttrn* ⇒ *'a set* ⇒ *'b* ⇒ *'b* ((*3*⨆$_\sigma$ *-*∈*-./ -*) [*0*, *0*, *10*] *10*)

translations
⨆$_\sigma$ *x*∈*A*. *B* == *CONST Sup-sigma* ((λ*x*. *B*) ‘ *A*)

lemma *space-Sup-sigma*: *space* (*Sup-sigma M*) = (⋃*x*∈*M*. *space x*)
unfolding *Sup-sigma-def* **by** (*rule space-measure-of*) (*auto dest*: *sets.sets-into-space*)

lemma *sets-Sup-sigma*: *sets* (*Sup-sigma M*) = *sigma-sets* (⋃*x*∈*M*. *space x*) (⋃*x*∈*M*. *sets x*)
unfolding *Sup-sigma-def* **by** (*rule sets-measure-of*) (*auto dest*: *sets.sets-into-space*)

lemma *in-Sup-sigma*: *m* ∈ *M* ⟹ *A* ∈ *sets m* ⟹ *A* ∈ *sets* (*Sup-sigma M*)
unfolding *sets-Sup-sigma* **by** *auto*

lemma *SUP-sigma-cong*:
assumes $*$: $\bigwedge i.\ i \in I \Longrightarrow sets\ (M\ i) = sets\ (N\ i)$ **shows** $sets\ (\bigsqcup_\sigma\ i{\in}I.\ M\ i) = sets\ (\bigsqcup_\sigma\ i{\in}I.\ N\ i)$
using $*$ *sets-eq-imp-space-eq*[*OF* $*$] **by** (*simp add*: *Sup-sigma-def*)

lemma *sets-Sup-in-sets*:
assumes $M \neq \{\}$
assumes $\bigwedge m.\ m \in M \Longrightarrow space\ m = space\ N$
assumes $\bigwedge m.\ m \in M \Longrightarrow sets\ m \subseteq sets\ N$
shows $sets\ (Sup\text{-}sigma\ M) \subseteq sets\ N$
proof −
have $*$: $UNION\ M\ space = space\ N$
using *assms* **by** *auto*
show *?thesis*
unfolding *sets-Sup-sigma* $*$ **using** *assms* **by** (*auto intro*!: *sets.sigma-sets-subset*)
qed

lemma *measurable-Sup-sigma1*:
assumes *m*: $m \in M$ **and** *f*: $f \in measurable\ m\ N$
and *const-space*: $\bigwedge m\ n.\ m \in M \Longrightarrow n \in M \Longrightarrow space\ m = space\ n$
shows $f \in measurable\ (Sup\text{-}sigma\ M)\ N$
proof −
have $space\ (Sup\text{-}sigma\ M) = space\ m$
using *m* **by** (*auto simp add*: *space-Sup-sigma dest*: *const-space*)
then show *?thesis*
using *m f* **unfolding** *measurable-def* **by** (*auto intro*: *in-Sup-sigma*)
qed

lemma *measurable-Sup-sigma2*:
assumes *M*: $M \neq \{\}$
assumes *f*: $\bigwedge m.\ m \in M \Longrightarrow f \in measurable\ N\ m$
shows $f \in measurable\ N\ (Sup\text{-}sigma\ M)$
unfolding *Sup-sigma-def*
proof (*rule measurable-measure-of*)
show $f \in space\ N \to UNION\ M\ space$
using *measurable-space*[*OF f*] *M* **by** *auto*
qed (*auto intro*: *measurable-sets f dest*: *sets.sets-into-space*)

lemma *Sup-sigma-sigma*:
assumes [*simp*]: $M \neq \{\}$ **and** *M*: $\bigwedge m.\ m \in M \Longrightarrow m \subseteq Pow\ \Omega$
shows $(\bigsqcup_\sigma\ m{\in}M.\ sigma\ \Omega\ m) = sigma\ \Omega\ (\bigcup M)$
proof (*rule measure-eqI*)
{ fix *a m* **assume** $a \in sigma\text{-}sets\ \Omega\ m\ m \in M$
then have $a \in sigma\text{-}sets\ \Omega\ (\bigcup M)$
by *induction* (*auto intro*: *sigma-sets.intros*) **}**
then show $sets\ (\bigsqcup_\sigma\ m{\in}M.\ sigma\ \Omega\ m) = sets\ (sigma\ \Omega\ (\bigcup M))$
apply (*simp add*: *sets-Sup-sigma space-measure-of-conv M Union-least*)
apply (*rule sigma-sets-eqI*)

apply *auto*
done
qed (*simp add*: *Sup-sigma-def emeasure-sigma*)

lemma *SUP-sigma-sigma*:
assumes *M*: $M \neq \{\}$ $\bigwedge m.\ m \in M \Longrightarrow f\ m \subseteq Pow\ \Omega$
shows $(\bigsqcup_\sigma\ m{\in}M.\ sigma\ \Omega\ (f\ m)) = sigma\ \Omega\ (\bigcup m{\in}M.\ f\ m)$
proof −
have *Sup-sigma* (*sigma* Ω ‘ *f* ‘ *M*) = *sigma* Ω (⋃(*f* ‘ *M*))
using *M* **by** (*intro Sup-sigma-sigma*) *auto*
then show *?thesis*
by (*simp add*: *image-image*)
qed

2.3 The smallest σ-algebra regarding a function

definition
vimage-algebra X f M = *sigma X* {*f* −‘ *A* ∩ *X* | *A*. *A* ∈ *sets M*}

lemma *space-vimage-algebra*[*simp*]: *space* (*vimage-algebra X f M*) = *X*
unfolding *vimage-algebra-def* **by** (*rule space-measure-of*) *auto*

lemma *sets-vimage-algebra*: *sets* (*vimage-algebra X f M*) = *sigma-sets X* {*f* −‘ *A* ∩ *X* | *A*. *A* ∈ *sets M*}
unfolding *vimage-algebra-def* **by** (*rule sets-measure-of*) *auto*

lemma *sets-vimage-algebra2*:
f ∈ *X* → *space M* ⟹ *sets* (*vimage-algebra X f M*) = {*f* −‘ *A* ∩ *X* | *A*. *A* ∈ *sets M*}
using *sigma-sets-vimage-commute*[*of f X space M sets M*]
unfolding *sets-vimage-algebra sets.sigma-sets-eq* **by** *simp*

lemma *sets-vimage-algebra-cong*: *sets M* = *sets N* ⟹ *sets* (*vimage-algebra X f M*) = *sets* (*vimage-algebra X f N*)
by (*simp add*: *sets-vimage-algebra*)

lemma *vimage-algebra-cong*:
assumes *X* = *Y*
assumes ⋀*x*. *x* ∈ *Y* ⟹ *f x* = *g x*
assumes *sets M* = *sets N*
shows *vimage-algebra X f M* = *vimage-algebra Y g N*
by (*auto simp*: *vimage-algebra-def assms intro*!: *arg-cong2*[**where** *f*=*sigma*])

lemma *in-vimage-algebra*: *A* ∈ *sets M* ⟹ *f* −‘ *A* ∩ *X* ∈ *sets* (*vimage-algebra X f M*)
by (*auto simp*: *vimage-algebra-def*)

lemma *sets-image-in-sets*:
assumes *N*: *space N* = *X*

assumes *f*: $f \in$ *measurable N M*
shows *sets* (*vimage-algebra X f M*) $\subseteq$ *sets N*
unfolding *sets-vimage-algebra N*[*symmetric*]
by (*rule sets.sigma-sets-subset*) (*auto intro*!: *measurable-sets f*)

lemma *measurable-vimage-algebra1*: $f \in X \to$ *space M* $\Longrightarrow f \in$ *measurable* (*vimage-algebra X f M*) *M*
unfolding *measurable-def* **by** (*auto intro*: *in-vimage-algebra*)

lemma *measurable-vimage-algebra2*:
assumes *g*: $g \in$ *space N* $\to X$ **and** *f*: $(\lambda x.\ f\ (g\ x)) \in$ *measurable N M*
shows $g \in$ *measurable N* (*vimage-algebra X f M*)
unfolding *vimage-algebra-def*
proof (*rule measurable-measure-of*)
fix *A* **assume** $A \in \{f\ -`\ A \cap X \mid A.\ A \in$ *sets M*$\}$
then obtain *Y* **where** *Y*: $Y \in$ *sets M* **and** *A*: $A = f\ -`\ Y \cap X$
by *auto*
then have $g\ -`\ A \cap$ *space N* $= (\lambda x.\ f\ (g\ x))\ -`\ Y \cap$ *space N*
using *g* **by** *auto*
also have $\ldots \in$ *sets N*
using *f Y* **by** (*rule measurable-sets*)
finally show $g\ -`\ A \cap$ *space N* $\in$ *sets N* **.**
qed (*insert g*, *auto*)

lemma *vimage-algebra-sigma*:
assumes *X*: $X \subseteq$ *Pow* Ω' **and** *f*: $f \in \Omega \to \Omega'$
shows *vimage-algebra* Ω *f* (*sigma* Ω' *X*) = *sigma* Ω $\{f\ -`\ A \cap \Omega \mid A.\ A \in X\ \}$
(**is** *?V* = *?S*)
proof (*rule measure-eqI*)
have Ω: $\{f\ -`\ A \cap \Omega \mid A.\ A \in X\} \subseteq$ *Pow* Ω **by** *auto*
show *sets ?V* = *sets ?S*
using *sigma-sets-vimage-commute*[*OF f*, *of X*]
by (*simp add*: *space-measure-of-conv f sets-vimage-algebra2* Ω *X*)
qed (*simp add*: *vimage-algebra-def emeasure-sigma*)

lemma *vimage-algebra-vimage-algebra-eq*:
assumes $*$: $f \in X \to Y$ $g \in Y \to$ *space M*
shows *vimage-algebra X f* (*vimage-algebra Y g M*) = *vimage-algebra X* $(\lambda x.\ g\ (f\ x))$ *M*
(**is** *?VV* = *?V*)
proof (*rule measure-eqI*)
have $(\lambda x.\ g\ (f\ x)) \in X \to$ *space M* $\bigwedge A.\ A \cap f\ -`\ Y \cap X = A \cap X$
using $*$ **by** *auto*
with $*$ **show** *sets ?VV* = *sets ?V*
by (*simp add*: *sets-vimage-algebra2 ex-simps*[*symmetric*] *vimage-comp comp-def del*: *ex-simps*)
qed (*simp add*: *vimage-algebra-def emeasure-sigma*)

lemma *sets-vimage-Sup-eq*:

assumes $*$: $M \neq \{\}$ $\bigwedge m.\ m \in M \Longrightarrow f \in X \to space\ m$
shows *sets* (*vimage-algebra X f* (*Sup-sigma M*)) = *sets* ($\bigsqcup_\sigma$ $m \in M$. *vimage-algebra X f m*)
(**is** *?IS* = *?SI*)
proof
show *?IS* ⊆ *?SI*
by (*intro sets-image-in-sets measurable-Sup-sigma2 measurable-Sup-sigma1*)
(*auto simp*: *space-Sup-sigma measurable-vimage-algebra1* $*$)
{ fix *m* **assume** $m \in M$
moreover then have $f \in X \to space\ (Sup\text{-}sigma\ M)$ $f \in X \to space\ m$
using $*$ **by** (*auto simp*: *space-Sup-sigma*)
ultimately have $f \in measurable\ (vimage\text{-}algebra\ X\ f\ (Sup\text{-}sigma\ M))\ m$
by (*auto simp add*: *measurable-def sets-vimage-algebra2 intro*: *in-Sup-sigma*)
}
then show *?SI* ⊆ *?IS*
by (*auto intro*!: *sets-image-in-sets sets-Sup-in-sets del*: *subsetI simp*: $*$)
qed

lemma *vimage-algebra-Sup-sigma*:
assumes [*simp*]: $MM \neq \{\}$ **and** $\bigwedge M.\ M \in MM \Longrightarrow f \in X \to space\ M$
shows *vimage-algebra X f* (*Sup-sigma MM*) = *Sup-sigma* (*vimage-algebra X f* ‘ *MM*)
proof (*rule measure-eqI*)
show *sets* (*vimage-algebra X f* (*Sup-sigma MM*)) = *sets* (*Sup-sigma* (*vimage-algebra X f* ‘ *MM*))
using *assms* **by** (*rule sets-vimage-Sup-eq*)
qed (*simp add*: *vimage-algebra-def Sup-sigma-def emeasure-sigma*)

2.3.1 Restricted Space Sigma Algebra

definition *restrict-space* **where**
restrict-space M Ω = *measure-of* (Ω ∩ *space M*) ((*op* ∩ Ω) ‘ *sets M*) (*emeasure M*)

lemma *space-restrict-space*: *space* (*restrict-space M* Ω) = Ω ∩ *space M*
using *sets.sets-into-space* **unfolding** *restrict-space-def* **by** (*subst space-measure-of*) *auto*

lemma *space-restrict-space2*: Ω ∈ *sets M* ⟹ *space* (*restrict-space M* Ω) = Ω
by (*simp add*: *space-restrict-space sets.sets-into-space*)

lemma *sets-restrict-space*: *sets* (*restrict-space M* Ω) = (*op* ∩ Ω) ‘ *sets M*
unfolding *restrict-space-def*
proof (*subst sets-measure-of*)
show *op* ∩ Ω ‘ *sets M* ⊆ *Pow* (Ω ∩ *space M*)
by (*auto dest*: *sets.sets-into-space*)
have *sigma-sets* (Ω ∩ *space M*) {((λ*x*. *x*) −‘ *X*) ∩ (Ω ∩ *space M*) | *X*. *X* ∈ *sets M*} =
(λ*X*. *X* ∩ (Ω ∩ *space M*)) ‘ *sets M*

by (*subst sigma-sets-vimage-commute*[*symmetric*, **where** $\Omega' = space\ M$])
(*auto simp add*: *sets.sigma-sets-eq*)
moreover have $\{((\lambda x.\ x) -` X) \cap (\Omega \cap space\ M) \mid X.\ X \in sets\ M\} = (\lambda X.\ X \cap (\Omega \cap space\ M))\ `\ sets\ M$
by *auto*
moreover have $(\lambda X.\ X \cap (\Omega \cap space\ M))\ `\ sets\ M = (op \cap \Omega)\ `\ sets\ M$
by (*intro image-cong*) (*auto dest*: *sets.sets-into-space*)
ultimately show *sigma-sets* $(\Omega \cap space\ M)\ (op \cap \Omega\ `\ sets\ M) = op \cap \Omega\ `\ sets\ M$
by *simp*
qed

lemma *restrict-space-sets-cong*:
$A = B \Longrightarrow sets\ M = sets\ N \Longrightarrow sets\ (restrict\text{-}space\ M\ A) = sets\ (restrict\text{-}space\ N\ B)$
by (*auto simp*: *sets-restrict-space*)

lemma *sets-restrict-space-count-space* :
$sets\ (restrict\text{-}space\ (count\text{-}space\ A)\ B) = sets\ (count\text{-}space\ (A \cap B))$
by(*auto simp add*: *sets-restrict-space*)

lemma *sets-restrict-UNIV*[*simp*]: $sets\ (restrict\text{-}space\ M\ UNIV) = sets\ M$
by (*auto simp add*: *sets-restrict-space*)

lemma *sets-restrict-restrict-space*:
$sets\ (restrict\text{-}space\ (restrict\text{-}space\ M\ A)\ B) = sets\ (restrict\text{-}space\ M\ (A \cap B))$
unfolding *sets-restrict-space image-comp* **by** (*intro image-cong*) *auto*

lemma *sets-restrict-space-iff*:
$\Omega \cap space\ M \in sets\ M \Longrightarrow A \in sets\ (restrict\text{-}space\ M\ \Omega) \longleftrightarrow (A \subseteq \Omega \wedge A \in sets\ M)$
proof (*subst sets-restrict-space*, *safe*)
fix A **assume** $\Omega \cap space\ M \in sets\ M$ **and** A: $A \in sets\ M$
then have $(\Omega \cap space\ M) \cap A \in sets\ M$
by *rule*
also have $(\Omega \cap space\ M) \cap A = \Omega \cap A$
using *sets.sets-into-space*[*OF A*] **by** *auto*
finally show $\Omega \cap A \in sets\ M$
by *auto*
qed *auto*

lemma *sets-restrict-space-cong*: $sets\ M = sets\ N \Longrightarrow sets\ (restrict\text{-}space\ M\ \Omega) = sets\ (restrict\text{-}space\ N\ \Omega)$
by (*simp add*: *sets-restrict-space*)

lemma *restrict-space-eq-vimage-algebra*:
$\Omega \subseteq space\ M \Longrightarrow sets\ (restrict\text{-}space\ M\ \Omega) = sets\ (vimage\text{-}algebra\ \Omega\ (\lambda x.\ x)\ M)$
unfolding *restrict-space-def*
apply (*subst sets-measure-of*)

apply (*auto simp add*: *image-subset-iff dest*: *sets.sets-into-space*) []
apply (*auto simp add*: *sets-vimage-algebra intro*!: *arg-cong2*[**where** *f*=*sigma-sets*])
done

lemma *sets-Collect-restrict-space-iff*:
assumes $S \in$ *sets M*
shows $\{x \in$*space* (*restrict-space M S*). *P x*$\} \in$ *sets* (*restrict-space M S*) $\longleftrightarrow$ $\{x \in$*space M*. $x \in S \wedge$ *P x*$\} \in$ *sets M*
proof −
have $\{x \in S.$ *P x*$\} = \{x \in$*space M*. $x \in S \wedge$ *P x*$\}$
using *sets.sets-into-space*[*OF assms*] **by** *auto*
then show *?thesis*
by (*subst sets-restrict-space-iff*) (*auto simp add*: *space-restrict-space assms*)
qed

lemma *measurable-restrict-space1*:
assumes *f*: $f \in$ *measurable M N*
shows $f \in$ *measurable* (*restrict-space M* Ω) *N*
unfolding *measurable-def*
proof (*intro CollectI conjI ballI*)
show *sp*: $f \in$ *space* (*restrict-space M* Ω) $\rightarrow$ *space N*
using *measurable-space*[*OF f*] **by** (*auto simp*: *space-restrict-space*)

fix *A* **assume** $A \in$ *sets N*
have $f -` A \cap$ *space* (*restrict-space M* $\Omega$) = $(f -` A \cap$ *space M*$) \cap (\Omega \cap$ *space M*)
by (*auto simp*: *space-restrict-space*)
also have ... $\in$ *sets* (*restrict-space M* Ω)
unfolding *sets-restrict-space*
using *measurable-sets*[*OF f* ‹*A* $\in$ *sets N*›] **by** *blast*
finally show $f -` A \cap$ *space* (*restrict-space M* Ω) $\in$ *sets* (*restrict-space M* Ω) **.**
qed

lemma *measurable-restrict-space2-iff*:
$f \in$ *measurable M* (*restrict-space N* Ω) $\longleftrightarrow$ ($f \in$ *measurable M N* $\wedge$ $f \in$ *space M* $\rightarrow \Omega$)
proof −
have $\bigwedge A.\ f \in$ *space M* $\rightarrow \Omega \Longrightarrow f -` \Omega \cap f -` A \cap$ *space M* = $f -` A \cap$ *space M*
by *auto*
then show *?thesis*
by (*auto simp*: *measurable-def space-restrict-space Pi-Int*[*symmetric*] *sets-restrict-space*)
qed

lemma *measurable-restrict-space2*:
$f \in$ *space M* $\rightarrow \Omega \Longrightarrow f \in$ *measurable M N* $\Longrightarrow f \in$ *measurable M* (*restrict-space N* Ω)
by (*simp add*: *measurable-restrict-space2-iff*)

lemma *measurable-piecewise-restrict*:
assumes *I*: *countable C*
and *X*: $\bigwedge\Omega.\ \Omega \in C \Longrightarrow \Omega \cap$ *space M* $\in$ *sets M space M* $\subseteq \bigcup C$
and *f*: $\bigwedge\Omega.\ \Omega \in C \Longrightarrow f \in$ *measurable* (*restrict-space M* Ω) *N*
shows $f \in$ *measurable M N*
proof (*rule measurableI*)
fix *x* **assume** $x \in$ *space M*
with *X* **obtain** Ω **where** $\Omega \in C\ x \in \Omega\ x \in$ *space M* **by** *auto*
then show $f\ x \in$ *space N*
by (*auto simp*: *space-restrict-space intro*: *f measurable-space*)
next
fix *A* **assume** *A*: $A \in$ *sets N*
have $f\ -`\ A \cap$ *space M* $= (\bigcup\Omega{\in}C.\ (f\ -`\ A \cap (\Omega \cap$ *space M*$)))$
using *X* **by** (*auto simp*: *subset-eq*)
also have $\ldots \in$ *sets M*
using *measurable-sets*[*OF f A*] *X I*
by (*intro sets.countable-UN'*) (*auto simp*: *sets-restrict-space-iff space-restrict-space*)
finally show $f\ -`\ A \cap$ *space M* $\in$ *sets M* **.**
qed

lemma *measurable-piecewise-restrict-iff*:
countable C $\Longrightarrow (\bigwedge\Omega.\ \Omega \in C \Longrightarrow \Omega \cap$ *space M* $\in$ *sets M*$) \Longrightarrow$ *space M* $\subseteq (\bigcup C)$
$\Longrightarrow$
$f \in$ *measurable M N* $\longleftrightarrow (\forall\,\Omega{\in}C.\ f \in$ *measurable* (*restrict-space M* Ω) *N*$)$
by (*auto intro*: *measurable-piecewise-restrict measurable-restrict-space1*)

lemma *measurable-If-restrict-space-iff*:
$\{x{\in}$*space M*$.\ P\ x\} \in$ *sets M* $\Longrightarrow$
$(\lambda x.$ *if P x then f x else g x*$) \in$ *measurable M N* $\longleftrightarrow$
$(f \in$ *measurable* (*restrict-space M* $\{x.\ P\ x\}$) *N* $\wedge\ g \in$ *measurable* (*restrict-space M* $\{x.\ \neg\ P\ x\}$) *N*$)$
by (*subst measurable-piecewise-restrict-iff*[**where** $C{=}\{\{x.\ P\ x\}, \{x.\ \neg\ P\ x\}\}$])
(*auto simp*: *Int-def sets.sets-Collect-neg space-restrict-space conj-commute*[*of* - $x \in$ *space M* **for** *x*]
cong: *measurable-cong'*)

lemma *measurable-If*:
$f \in$ *measurable M M'* $\Longrightarrow$ $g \in$ *measurable M M'* $\Longrightarrow \{x{\in}$*space M*$.\ P\ x\} \in$ *sets M* $\Longrightarrow$
$(\lambda x.$ *if P x then f x else g x*$) \in$ *measurable M M'*
unfolding *measurable-If-restrict-space-iff* **by** (*auto intro*: *measurable-restrict-space1*)

lemma *measurable-If-set*:
assumes *measure*: $f \in$ *measurable M M'* $g \in$ *measurable M M'*
assumes *P*: $A \cap$ *space M* $\in$ *sets M*
shows $(\lambda x.$ *if* $x \in A$ *then f x else g x*$) \in$ *measurable M M'*
proof (*rule measurable-If*[*OF measure*])
have $\{x \in$ *space M*$.\ x \in A\} = A \cap$ *space M* **by** *auto*
thus $\{x \in$ *space M*$.\ x \in A\} \in$ *sets M* **using** ‹$A \cap$ *space M* $\in$ *sets M*› **by** *auto*

qed

lemma *measurable-restrict-space-iff*:
 $\Omega \cap$ *space* $M \in$ *sets* $M \Longrightarrow c \in$ *space* $N \Longrightarrow$
 $f \in$ *measurable* (*restrict-space* $M\ \Omega$) $N \longleftrightarrow (\lambda x.$ *if* $x \in \Omega$ *then* $f\ x$ *else* $c) \in$ *measurable* $M\ N$
 by (*subst measurable-If-restrict-space-iff*)
 (*simp-all add*: *Int-def conj-commute measurable-const*)

lemma *restrict-space-singleton*: $\{x\} \in$ *sets* $M \Longrightarrow$ *sets* (*restrict-space* $M\ \{x\}$) = *sets* (*count-space* $\{x\}$)
 using *sets-restrict-space-iff*[*of* $\{x\}$ M]
 by (*auto simp add*: *sets-restrict-space-iff dest*!: *subset-singletonD*)

lemma *measurable-restrict-countable*:
 assumes X[*intro*]: *countable* X
 assumes *sets*[*simp*]: $\bigwedge x.\ x \in X \Longrightarrow \{x\} \in$ *sets* M
 assumes *space*[*simp*]: $\bigwedge x.\ x \in X \Longrightarrow f\ x \in$ *space* N
 assumes f: $f \in$ *measurable* (*restrict-space* M $(-\ X)$) N
 shows $f \in$ *measurable* $M\ N$
 using f *sets.countable*[*OF sets* X]
 by (*intro measurable-piecewise-restrict*[**where** $M{=}M$ **and** $C{=}\{-\ X\} \cup ((\lambda x.\ \{x\})\ `\ X)$])
 (*auto simp*: *Diff-Int-distrib2 Compl-eq-Diff-UNIV Int-insert-left sets.Diff restrict-space-singleton*
 simp del: *sets-count-space cong*: *measurable-cong-sets*)

lemma *measurable-discrete-difference*:
 assumes f: $f \in$ *measurable* $M\ N$
 assumes X: *countable* X $\bigwedge x.\ x \in X \Longrightarrow \{x\} \in$ *sets* M $\bigwedge x.\ x \in X \Longrightarrow g\ x \in$ *space* N
 assumes *eq*: $\bigwedge x.\ x \in$ *space* $M \Longrightarrow x \notin X \Longrightarrow f\ x = g\ x$
 shows $g \in$ *measurable* $M\ N$
 by (*rule measurable-restrict-countable*[*OF* X])
 (*auto simp*: *eq*[*symmetric*] *space-restrict-space cong*: *measurable-cong'* *intro*: *f measurable-restrict-space1*)

end

theory *Measurable*
 imports
 Sigma-Algebra
 ~~/*src*/*HOL*/*Library*/*Order-Continuity*
begin

2.4 Measurability prover

lemma (**in** *algebra*) *sets-Collect-finite-All*:
 assumes $\bigwedge i.\ i \in S \Longrightarrow \{x \in \Omega.\ P\ i\ x\} \in M$ *finite* S
 shows $\{x \in \Omega.\ \forall i \in S.\ P\ i\ x\} \in M$

proof −
have $\{x\in\Omega.\ \forall i\in S.\ P\ i\ x\} = (\mathit{if}\ S = \{\}\ \mathit{then}\ \Omega\ \mathit{else}\ \bigcap i\in S.\ \{x\in\Omega.\ P\ i\ x\})$
by *auto*
with *assms* **show** *?thesis* **by** (*auto intro*!: *sets-Collect-finite-All'*)
qed

abbreviation *pred M P* ≡ $P \in \mathit{measurable}\ M\ (\mathit{count\text{-}space}\ (\mathit{UNIV}::\mathit{bool\ set}))$

lemma *pred-def*: *pred M P* ⟷ $\{x\in\mathit{space}\ M.\ P\ x\} \in \mathit{sets}\ M$
proof
assume *pred M P*
then have $P -` \{\mathit{True}\} \cap \mathit{space}\ M \in \mathit{sets}\ M$
by (*auto simp*: *measurable-count-space-eq2*)
also have $P -` \{\mathit{True}\} \cap \mathit{space}\ M = \{x\in\mathit{space}\ M.\ P\ x\}$ **by** *auto*
finally show $\{x\in\mathit{space}\ M.\ P\ x\} \in \mathit{sets}\ M$ **.**
next
assume *P*: $\{x\in\mathit{space}\ M.\ P\ x\} \in \mathit{sets}\ M$
moreover
{ fix *X*
have $X \in \mathit{Pow}\ (\mathit{UNIV}::\mathit{bool\ set})$ **by** *simp*
then have $P -` X \cap \mathit{space}\ M = \{x\in\mathit{space}\ M.\ ((X = \{\mathit{True}\} \longrightarrow P\ x) \wedge (X = \{\mathit{False}\} \longrightarrow \neg\ P\ x) \wedge X \neq \{\})\}$
unfolding *UNIV-bool Pow-insert Pow-empty* **by** *auto*
then have $P -` X \cap \mathit{space}\ M \in \mathit{sets}\ M$
by (*auto intro*!: *sets.sets-Collect-neg sets.sets-Collect-imp sets.sets-Collect-conj sets.sets-Collect-const P*) **}**
then show *pred M P*
by (*auto simp*: *measurable-def*)
qed

lemma *pred-sets1*: $\{x\in\mathit{space}\ M.\ P\ x\} \in \mathit{sets}\ M \Longrightarrow f \in \mathit{measurable}\ N\ M \Longrightarrow \mathit{pred}\ N\ (\lambda x.\ P\ (f\ x))$
by (*rule measurable-compose*[**where** *f*=*f* **and** *N*=*M*]) (*auto simp*: *pred-def*)

lemma *pred-sets2*: $A \in \mathit{sets}\ N \Longrightarrow f \in \mathit{measurable}\ M\ N \Longrightarrow \mathit{pred}\ M\ (\lambda x.\ f\ x \in A)$
by (*rule measurable-compose*[**where** *f*=*f* **and** *N*=*N*]) (*auto simp*: *pred-def Int-def*[*symmetric*])

ML-file *measurable.ML*

attribute-setup *measurable* = ‹
Scan.lift (
(*Args.add* >> *K true* || *Args.del* >> *K false* || *Scan.succeed true*) −−
Scan.optional (*Args.parens* (
Scan.optional (*Args.$$$ raw* >> *K true*) *false* −−
Scan.optional (*Args.$$$ generic* >> *K Measurable.Generic*) *Measurable.Concrete*))
(*false*, *Measurable.Concrete*) >>
Measurable.measurable-thm-attr)
› *declaration of measurability theorems*

attribute-setup *measurable-dest = Measurable.dest-thm-attr*
add dest rule to measurability prover

attribute-setup *measurable-cong = Measurable.cong-thm-attr*
add congurence rules to measurability prover

method-setup *measurable =* ‹ *Scan.lift (Scan.succeed (METHOD o Measurable.measurable-tac))*
›
measurability prover

simproc-setup *measurable (A ∈ sets M | f ∈ measurable M N) =* ‹*K Measurable.simproc*›

setup ‹
Global-Theory.add-thms-dynamic (@{binding measurable}, Measurable.get-all)
›

declare
pred-sets1[*measurable-dest*]
pred-sets2[*measurable-dest*]
sets.sets-into-space[*measurable-dest*]

declare
sets.top[*measurable*]
sets.empty-sets[*measurable (raw)*]
sets.Un[*measurable (raw)*]
sets.Diff[*measurable (raw)*]

declare
measurable-count-space[*measurable (raw)*]
measurable-ident[*measurable (raw)*]
measurable-id[*measurable (raw)*]
measurable-const[*measurable (raw)*]
measurable-If[*measurable (raw)*]
measurable-comp[*measurable (raw)*]
measurable-sets[*measurable (raw)*]

declare *measurable-cong-sets*[*measurable-cong*]
declare *sets-restrict-space-cong*[*measurable-cong*]
declare *sets-restrict-UNIV*[*measurable-cong*]

lemma *predE*[*measurable (raw)*]:
pred M P ⟹ {*x*∈*space M*. *P x*} ∈ *sets M*
unfolding *pred-def* **.**

lemma *pred-intros-imp'*[*measurable (raw)*]:
(*K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *K* ⟶ *P x*)
by (*cases K*) *auto*

lemma *pred-intros-conj1'*[*measurable* (*raw*)]:
(*K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *K* ∧ *P x*)
by (*cases K*) *auto*

lemma *pred-intros-conj2'*[*measurable* (*raw*)]:
(*K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *P x* ∧ *K*)
by (*cases K*) *auto*

lemma *pred-intros-disj1'*[*measurable* (*raw*)]:
(¬ *K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *K* ∨ *P x*)
by (*cases K*) *auto*

lemma *pred-intros-disj2'*[*measurable* (*raw*)]:
(¬ *K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *P x* ∨ *K*)
by (*cases K*) *auto*

lemma *pred-intros-logic*[*measurable* (*raw*)]:
pred M (λ*x*. *x* ∈ *space M*)
pred M (λ*x*. *P x*) ⟹ *pred M* (λ*x*. ¬ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* ∧ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* ⟶ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* ∨ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* = *P x*)
pred M (λ*x*. *f x* ∈ *UNIV*)
pred M (λ*x*. *f x* ∈ {})
pred M (λ*x*. *P'* (*f x*) *x*) ⟹ *pred M* (λ*x*. *f x* ∈ {*y*. *P'* *y x*})
pred M (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ − (*B x*))
pred M (λ*x*. *f x* ∈ (*A x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*A x*) − (*B x*))
pred M (λ*x*. *f x* ∈ (*A x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*A x*) ∩ (*B x*))
pred M (λ*x*. *f x* ∈ (*A x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*A x*) ∪ (*B x*))
pred M (λ*x*. *g x* (*f x*) ∈ (*X x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*g x*) −‘ (*X x*))
by (*auto simp*: *iff-conv-conj-imp pred-def*)

lemma *pred-intros-countable*[*measurable* (*raw*)]:
fixes *P* :: *'a* ⇒ *'i* :: *countable* ⇒ *bool*
shows
(⋀*i*. *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∀ *i*. *P x i*)
(⋀*i*. *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∃ *i*. *P x i*)
by (*auto intro*!: *sets.sets-Collect-countable-All sets.sets-Collect-countable-Ex simp*: *pred-def*)

lemma *pred-intros-countable-bounded*[*measurable* (*raw*)]:
fixes *X* :: *'i* :: *countable set*
shows
(⋀*i*. *i* ∈ *X* ⟹ *pred M* (λ*x*. *x* ∈ *N x i*)) ⟹ *pred M* (λ*x*. *x* ∈ (⋂ *i*∈*X*. *N x i*))

$(\bigwedge i.\ i \in X \Longrightarrow pred\ M\ (\lambda x.\ x \in N\ x\ i)) \Longrightarrow pred\ M\ (\lambda x.\ x \in (\bigcup i{\in}X.\ N\ x\ i))$
$(\bigwedge i.\ i \in X \Longrightarrow pred\ M\ (\lambda x.\ P\ x\ i)) \Longrightarrow pred\ M\ (\lambda x.\ \forall i{\in}X.\ P\ x\ i)$
$(\bigwedge i.\ i \in X \Longrightarrow pred\ M\ (\lambda x.\ P\ x\ i)) \Longrightarrow pred\ M\ (\lambda x.\ \exists i{\in}X.\ P\ x\ i)$
by *simp-all* (*auto simp*: *Bex-def Ball-def*)

lemma *pred-intros-finite*[*measurable* (*raw*)]:
finite $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow pred\ M\ (\lambda x.\ x \in N\ x\ i)) \Longrightarrow pred\ M\ (\lambda x.\ x \in (\bigcap i{\in}I.\ N\ x\ i))$
finite $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow pred\ M\ (\lambda x.\ x \in N\ x\ i)) \Longrightarrow pred\ M\ (\lambda x.\ x \in (\bigcup i{\in}I.\ N\ x\ i))$
finite $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow pred\ M\ (\lambda x.\ P\ x\ i)) \Longrightarrow pred\ M\ (\lambda x.\ \forall i{\in}I.\ P\ x\ i)$
finite $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow pred\ M\ (\lambda x.\ P\ x\ i)) \Longrightarrow pred\ M\ (\lambda x.\ \exists i{\in}I.\ P\ x\ i)$
by (*auto intro*!: *sets.sets-Collect-finite-Ex sets.sets-Collect-finite-All simp*: *iff-conv-conj-imp pred-def*)

lemma *countable-Un-Int*[*measurable* (*raw*)]:
$(\bigwedge i :: 'i :: countable.\ i \in I \Longrightarrow N\ i \in sets\ M) \Longrightarrow (\bigcup i{\in}I.\ N\ i) \in sets\ M$
$I \neq \{\} \Longrightarrow (\bigwedge i :: 'i :: countable.\ i \in I \Longrightarrow N\ i \in sets\ M) \Longrightarrow (\bigcap i{\in}I.\ N\ i) \in sets\ M$
by *auto*

declare
finite-UN[*measurable* (*raw*)]
finite-INT[*measurable* (*raw*)]

lemma *sets-Int-pred*[*measurable* (*raw*)]:
assumes *space*: $A \cap B \subseteq space\ M$ **and** [*measurable*]: $pred\ M\ (\lambda x.\ x \in A)$ $pred\ M\ (\lambda x.\ x \in B)$
shows $A \cap B \in sets\ M$
proof −
have $\{x{\in}space\ M.\ x \in A \cap B\} \in sets\ M$ **by** *auto*
also have $\{x{\in}space\ M.\ x \in A \cap B\} = A \cap B$
using *space* **by** *auto*
finally show *?thesis* **.**
qed

lemma [*measurable* (*raw generic*)]:
assumes *f*: $f \in measurable\ M\ N$ **and** *c*: $c \in space\ N \Longrightarrow \{c\} \in sets\ N$
shows *pred-eq-const1*: $pred\ M\ (\lambda x.\ f\ x = c)$
and *pred-eq-const2*: $pred\ M\ (\lambda x.\ c = f\ x)$
proof −
show $pred\ M\ (\lambda x.\ f\ x = c)$
proof *cases*
assume $c \in space\ N$
with *measurable-sets*[*OF f c*] **show** *?thesis*
by (*auto simp*: *Int-def conj-commute pred-def*)
next
assume $c \notin space\ N$
with *f*[*THEN measurable-space*] **have** $\{x \in space\ M.\ f\ x = c\} = \{\}$ **by** *auto*

then show *?thesis* **by** (*auto simp*: *pred-def cong*: *conj-cong*)
qed
then show *pred M* (λ*x*. *c* = *f x*)
by (*simp add*: *eq-commute*)
qed

lemma *pred-count-space-const1*[*measurable* (*raw*)]:
f ∈ *measurable M* (*count-space UNIV*) ⟹ *Measurable.pred M* (λ*x*. *f x* = *c*)
by (*intro pred-eq-const1*[**where** *N*=*count-space UNIV*]) (*auto*)

lemma *pred-count-space-const2*[*measurable* (*raw*)]:
f ∈ *measurable M* (*count-space UNIV*) ⟹ *Measurable.pred M* (λ*x*. *c* = *f x*)
by (*intro pred-eq-const2*[**where** *N*=*count-space UNIV*]) (*auto*)

lemma *pred-le-const*[*measurable* (*raw generic*)]:
assumes *f*: *f* ∈ *measurable M N* **and** *c*: {.. *c*} ∈ *sets N* **shows** *pred M* (λ*x*. *f x* ≤ *c*)
using *measurable-sets*[*OF f c*]
by (*auto simp*: *Int-def conj-commute eq-commute pred-def*)

lemma *pred-const-le*[*measurable* (*raw generic*)]:
assumes *f*: *f* ∈ *measurable M N* **and** *c*: {*c* ..} ∈ *sets N* **shows** *pred M* (λ*x*. *c* ≤ *f x*)
using *measurable-sets*[*OF f c*]
by (*auto simp*: *Int-def conj-commute eq-commute pred-def*)

lemma *pred-less-const*[*measurable* (*raw generic*)]:
assumes *f*: *f* ∈ *measurable M N* **and** *c*: {..< *c*} ∈ *sets N* **shows** *pred M* (λ*x*. *f x* < *c*)
using *measurable-sets*[*OF f c*]
by (*auto simp*: *Int-def conj-commute eq-commute pred-def*)

lemma *pred-const-less*[*measurable* (*raw generic*)]:
assumes *f*: *f* ∈ *measurable M N* **and** *c*: {*c* <..} ∈ *sets N* **shows** *pred M* (λ*x*. *c* < *f x*)
using *measurable-sets*[*OF f c*]
by (*auto simp*: *Int-def conj-commute eq-commute pred-def*)

declare
sets.Int[*measurable* (*raw*)]

lemma *pred-in-If*[*measurable* (*raw*)]:
(*P* ⟹ *pred M* (λ*x*. *x* ∈ *A x*)) ⟹ (¬ *P* ⟹ *pred M* (λ*x*. *x* ∈ *B x*)) ⟹
pred M (λ*x*. *x* ∈ (*if P then A x else B x*))
by *auto*

lemma *sets-range*[*measurable-dest*]:
A ' *I* ⊆ *sets M* ⟹ *i* ∈ *I* ⟹ *A i* ∈ *sets M*
by *auto*

lemma *pred-sets-range*[*measurable-dest*]:
A ' I ⊆ *sets N* ⟹ *i* ∈ *I* ⟹ *f* ∈ *measurable M N* ⟹ *pred M* (λ*x*. *f x* ∈ *A i*)
using *pred-sets2*[*OF sets-range*] **by** *auto*

lemma *sets-All*[*measurable-dest*]:
∀ *i*. *A i* ∈ *sets* (*M i*) ⟹ *A i* ∈ *sets* (*M i*)
by *auto*

lemma *pred-sets-All*[*measurable-dest*]:
∀ *i*. *A i* ∈ *sets* (*N i*) ⟹ *f* ∈ *measurable M* (*N i*) ⟹ *pred M* (λ*x*. *f x* ∈ *A i*)
using *pred-sets2*[*OF sets-All*, *of A N f*] **by** *auto*

lemma *sets-Ball*[*measurable-dest*]:
∀ *i*∈*I*. *A i* ∈ *sets* (*M i*) ⟹ *i*∈*I* ⟹ *A i* ∈ *sets* (*M i*)
by *auto*

lemma *pred-sets-Ball*[*measurable-dest*]:
∀ *i*∈*I*. *A i* ∈ *sets* (*N i*) ⟹ *i*∈*I* ⟹ *f* ∈ *measurable M* (*N i*) ⟹ *pred M* (λ*x*. *f x* ∈ *A i*)
using *pred-sets2*[*OF sets-Ball*, *of* - - - *f*] **by** *auto*

lemma *measurable-finite*[*measurable* (*raw*)]:
fixes *S* :: *'a* ⇒ *nat set*
assumes [*measurable*]: ⋀*i*. {*x*∈*space M*. *i* ∈ *S x*} ∈ *sets M*
shows *pred M* (λ*x*. *finite* (*S x*))
unfolding *finite-nat-set-iff-bounded* **by** (*simp add*: *Ball-def*)

lemma *measurable-Least*[*measurable*]:
assumes [*measurable*]: (⋀*i*::*nat*. (λ*x*. *P i x*) ∈ *measurable M* (*count-space UNIV*))*q*
shows (λ*x*. *LEAST i*. *P i x*) ∈ *measurable M* (*count-space UNIV*)
unfolding *measurable-def* **by** (*safe intro*!: *sets-Least*) *simp-all*

lemma *measurable-Max-nat*[*measurable* (*raw*)]:
fixes *P* :: *nat* ⇒ *'a* ⇒ *bool*
assumes [*measurable*]: ⋀*i*. *Measurable.pred M* (*P i*)
shows (λ*x*. *Max* {*i*. *P i x*}) ∈ *measurable M* (*count-space UNIV*)
unfolding *measurable-count-space-eq2-countable*
proof *safe*
fix *n*

{ fix *x* **assume** ∀ *i*. ∃ *n*≥*i*. *P n x*
then have *infinite* {*i*. *P i x*}
unfolding *infinite-nat-iff-unbounded-le* **by** *auto*
then have *Max* {*i*. *P i x*} = *the None*
by (*rule Max.infinite*) **}**
note *1* = *this*

{ fix *x i j* **assume** *P i x* ∀ *n*≥*j*. ¬ *P n x*

```
    then have finite {i. P i x}
      by (auto simp: subset-eq not-le[symmetric] finite-nat-iff-bounded)
    with ‹P i x› have P (Max {i. P i x}) x i ≤ Max {i. P i x} finite {i. P i x}
      using Max-in[of {i. P i x}] by auto }
  note 2 = this

  have (λx. Max {i. P i x}) -‘ {n} ∩ space M = {x∈space M. Max {i. P i x}
= n}
    by auto
  also have ... =
    {x∈space M. if (∀ i. ∃ n≥i. P n x) then the None = n else
      if (∃ i. P i x) then P n x ∧ (∀ i>n. ¬ P i x)
      else Max {} = n}
    by (intro arg-cong[where f=Collect] ext conj-cong)
       (auto simp add: 1 2 not-le[symmetric] intro!: Max-eqI)
  also have ... ∈ sets M
    by measurable
  finally show (λx. Max {i. P i x}) -‘ {n} ∩ space M ∈ sets M .
qed simp

lemma measurable-Min-nat[measurable (raw)]:
  fixes P :: nat ⇒ 'a ⇒ bool
  assumes [measurable]: ⋀i. Measurable.pred M (P i)
  shows (λx. Min {i. P i x}) ∈ measurable M (count-space UNIV)
  unfolding measurable-count-space-eq2-countable
proof safe
  fix n

  { fix x assume ∀ i. ∃ n≥i. P n x
    then have infinite {i. P i x}
      unfolding infinite-nat-iff-unbounded-le by auto
    then have Min {i. P i x} = the None
      by (rule Min.infinite) }
  note 1 = this

  { fix x i j assume P i x ∀ n≥j. ¬ P n x
    then have finite {i. P i x}
      by (auto simp: subset-eq not-le[symmetric] finite-nat-iff-bounded)
    with ‹P i x› have P (Min {i. P i x}) x Min {i. P i x} ≤ i finite {i. P i x}
      using Min-in[of {i. P i x}] by auto }
  note 2 = this

  have (λx. Min {i. P i x}) -‘ {n} ∩ space M = {x∈space M. Min {i. P i x} =
n}
    by auto
  also have ... =
    {x∈space M. if (∀ i. ∃ n≥i. P n x) then the None = n else
      if (∃ i. P i x) then P n x ∧ (∀ i<n. ¬ P i x)
      else Min {} = n}
```

by (*intro arg-cong*[**where** *f*=*Collect*] *ext conj-cong*)
(*auto simp add*: *1 2 not-le*[*symmetric*] *intro*!: *Min-eqI*)
also have ... ∈ *sets M*
by *measurable*
finally show (λ*x*. *Min* {*i*. *P i x*}) −‘ {*n*} ∩ *space M* ∈ *sets M* **.**
qed *simp*

lemma *measurable-count-space-insert*[*measurable* (*raw*)]:
s ∈ *S* ⟹ *A* ∈ *sets* (*count-space S*) ⟹ *insert s A* ∈ *sets* (*count-space S*)
by *simp*

lemma *sets-UNIV* [*measurable* (*raw*)]: *A* ∈ *sets* (*count-space UNIV*)
by *simp*

lemma *measurable-card*[*measurable*]:
fixes *S* :: *'a* ⇒ *nat set*
assumes [*measurable*]: ⋀*i*. {*x*∈*space M*. *i* ∈ *S x*} ∈ *sets M*
shows (λ*x*. *card* (*S x*)) ∈ *measurable M* (*count-space UNIV*)
unfolding *measurable-count-space-eq2-countable*
proof *safe*
fix *n* **show** (λ*x*. *card* (*S x*)) −‘ {*n*} ∩ *space M* ∈ *sets M*
proof (*cases n*)
case *0*
then have (λ*x*. *card* (*S x*)) −‘ {*n*} ∩ *space M* = {*x*∈*space M*. *infinite* (*S x*) ∨ (∀*i*. *i* ∉ *S x*)}
by *auto*
also have ... ∈ *sets M*
by *measurable*
finally show *?thesis* **.**
next
case (*Suc i*)
then have (λ*x*. *card* (*S x*)) −‘ {*n*} ∩ *space M* =
(⋃*F*∈{*A*∈{*A*. *finite A*}. *card A* = *n*}. {*x*∈*space M*. (∀*i*. *i* ∈ *S x* ⟷ *i* ∈ *F*)})
unfolding *set-eq-iff*[*symmetric*] *Collect-bex-eq*[*symmetric*] **by** (*auto intro*: *card-ge-0-finite*)
also have ... ∈ *sets M*
by (*intro sets.countable-UN' countable-Collect countable-Collect-finite*) *auto*
finally show *?thesis* **.**
qed
qed *rule*

lemma *measurable-pred-countable*[*measurable* (*raw*)]:
assumes *countable X*
shows
(⋀*i*. *i* ∈ *X* ⟹ *Measurable.pred M* (λ*x*. *P x i*)) ⟹ *Measurable.pred M* (λ*x*. ∀*i*∈*X*. *P x i*)
(⋀*i*. *i* ∈ *X* ⟹ *Measurable.pred M* (λ*x*. *P x i*)) ⟹ *Measurable.pred M* (λ*x*. ∃*i*∈*X*. *P x i*)

unfolding *pred-def*
by (*auto intro*!: *sets.sets-Collect-countable-All'* *sets.sets-Collect-countable-Ex'* *assms*)

2.5 Measurability for (co)inductive predicates

lemma *measurable-bot*[*measurable*]: *bot* ∈ *measurable M* (*count-space UNIV*)
by (*simp add*: *bot-fun-def*)

lemma *measurable-top*[*measurable*]: *top* ∈ *measurable M* (*count-space UNIV*)
by (*simp add*: *top-fun-def*)

lemma *measurable-SUP*[*measurable*]:
fixes *F* :: *'i* ⇒ *'a* ⇒ *'b*::{*complete-lattice*, *countable*}
assumes [*simp*]: *countable I*
assumes [*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *measurable M* (*count-space UNIV*)
shows (λ*x*. *SUP i*:*I*. *F i x*) ∈ *measurable M* (*count-space UNIV*)
unfolding *measurable-count-space-eq2-countable*
proof (*safe intro*!: *UNIV-I*)
fix *a*
have (λ*x*. *SUP i*:*I*. *F i x*) −‘ {*a*} ∩ *space M* =
{*x*∈*space M*. (∀ *i*∈*I*. *F i x* ≤ *a*) ∧ (∀ *b*. (∀ *i*∈*I*. *F i x* ≤ *b*) ⟶ *a* ≤ *b*)}
unfolding *SUP-le-iff*[*symmetric*] **by** *auto*
also have ... ∈ *sets M*
by *measurable*
finally show (λ*x*. *SUP i*:*I*. *F i x*) −‘ {*a*} ∩ *space M* ∈ *sets M* .
qed

lemma *measurable-INF*[*measurable*]:
fixes *F* :: *'i* ⇒ *'a* ⇒ *'b*::{*complete-lattice*, *countable*}
assumes [*simp*]: *countable I*
assumes [*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *measurable M* (*count-space UNIV*)
shows (λ*x*. *INF i*:*I*. *F i x*) ∈ *measurable M* (*count-space UNIV*)
unfolding *measurable-count-space-eq2-countable*
proof (*safe intro*!: *UNIV-I*)
fix *a*
have (λ*x*. *INF i*:*I*. *F i x*) −‘ {*a*} ∩ *space M* =
{*x*∈*space M*. (∀ *i*∈*I*. *a* ≤ *F i x*) ∧ (∀ *b*. (∀ *i*∈*I*. *b* ≤ *F i x*) ⟶ *b* ≤ *a*)}
unfolding *le-INF-iff*[*symmetric*] **by** *auto*
also have ... ∈ *sets M*
by *measurable*
finally show (λ*x*. *INF i*:*I*. *F i x*) −‘ {*a*} ∩ *space M* ∈ *sets M* .
qed

lemma *measurable-lfp-coinduct*[*consumes 1*, *case-names continuity step*]:
fixes *F* :: (*'a* ⇒ *'b*) ⇒ (*'a* ⇒ *'b*::{*complete-lattice*, *countable*})
assumes *P M*
assumes *F*: *sup-continuous F*
assumes ∗: ⋀*M A*. *P M* ⟹ (⋀*N*. *P N* ⟹ *A* ∈ *measurable N* (*count-space*

UNIV)) ⟹ *F A* ∈ *measurable M* (*count-space UNIV*)
shows *lfp F* ∈ *measurable M* (*count-space UNIV*)
proof −
{ fix *i* **from** ‹*P M*› **have** ((*F* ^^ *i*) *bot*) ∈ *measurable M* (*count-space UNIV*)
by (*induct i arbitrary*: *M*) (*auto intro*!: ∗) **}**
then have (λ*x*. *SUP i*. (*F* ^^ *i*) *bot x*) ∈ *measurable M* (*count-space UNIV*)
by *measurable*
also have (λ*x*. *SUP i*. (*F* ^^ *i*) *bot x*) = *lfp F*
by (*subst sup-continuous-lfp*) (*auto intro*: *F*)
finally show *?thesis* **.**
qed

lemma *measurable-lfp*:
fixes *F* :: ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*b*::{*complete-lattice*, *countable*})
assumes *F*: *sup-continuous F*
assumes ∗: ⋀*A*. *A* ∈ *measurable M* (*count-space UNIV*) ⟹ *F A* ∈ *measurable M* (*count-space UNIV*)
shows *lfp F* ∈ *measurable M* (*count-space UNIV*)
by (*coinduction rule*: *measurable-lfp-coinduct*[*OF* - *F*]) (*blast intro*: ∗)

lemma *measurable-gfp-coinduct*[*consumes 1*, *case-names continuity step*]:
fixes *F* :: ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*b*::{*complete-lattice*, *countable*})
assumes *P M*
assumes *F*: *inf-continuous F*
assumes ∗: ⋀*M A*. *P M* ⟹ (⋀*N*. *P N* ⟹ *A* ∈ *measurable N* (*count-space UNIV*)) ⟹ *F A* ∈ *measurable M* (*count-space UNIV*)
shows *gfp F* ∈ *measurable M* (*count-space UNIV*)
proof −
{ fix *i* **from** ‹*P M*› **have** ((*F* ^^ *i*) *top*) ∈ *measurable M* (*count-space UNIV*)
by (*induct i arbitrary*: *M*) (*auto intro*!: ∗) **}**
then have (λ*x*. *INF i*. (*F* ^^ *i*) *top x*) ∈ *measurable M* (*count-space UNIV*)
by *measurable*
also have (λ*x*. *INF i*. (*F* ^^ *i*) *top x*) = *gfp F*
by (*subst inf-continuous-gfp*) (*auto intro*: *F*)
finally show *?thesis* **.**
qed

lemma *measurable-gfp*:
fixes *F* :: ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*b*::{*complete-lattice*, *countable*})
assumes *F*: *inf-continuous F*
assumes ∗: ⋀*A*. *A* ∈ *measurable M* (*count-space UNIV*) ⟹ *F A* ∈ *measurable M* (*count-space UNIV*)
shows *gfp F* ∈ *measurable M* (*count-space UNIV*)
by (*coinduction rule*: *measurable-gfp-coinduct*[*OF* - *F*]) (*blast intro*: ∗)

lemma *measurable-lfp2-coinduct*[*consumes 1*, *case-names continuity step*]:
fixes *F* :: ('*a* ⇒ '*c* ⇒ '*b*) ⇒ ('*a* ⇒ '*c* ⇒ '*b*::{*complete-lattice*, *countable*})
assumes *P M s*
assumes *F*: *sup-continuous F*

assumes $*$: $\bigwedge M\ A\ s.\ P\ M\ s \Longrightarrow (\bigwedge N\ t.\ P\ N\ t \Longrightarrow A\ t \in measurable\ N\ (count\text{-}space\ UNIV)) \Longrightarrow F\ A\ s \in measurable\ M\ (count\text{-}space\ UNIV)$
shows $lfp\ F\ s \in measurable\ M\ (count\text{-}space\ UNIV)$
proof −
{ fix i **from** ‹$P\ M\ s$› **have** $(\lambda x.\ (F\ \hat{}\hat{}\ i)\ bot\ s\ x) \in measurable\ M\ (count\text{-}space\ UNIV)$
by (*induct i arbitrary*: $M\ s$) (*auto intro*!: $*$) **}**
then have $(\lambda x.\ SUP\ i.\ (F\ \hat{}\hat{}\ i)\ bot\ s\ x) \in measurable\ M\ (count\text{-}space\ UNIV)$
by *measurable*
also have $(\lambda x.\ SUP\ i.\ (F\ \hat{}\hat{}\ i)\ bot\ s\ x) = lfp\ F\ s$
by (*subst sup-continuous-lfp*) (*auto simp*: F)
finally show *?thesis* **.**
qed

lemma *measurable-gfp2-coinduct*[*consumes 1*, *case-names continuity step*]:
fixes $F :: ('a \Rightarrow 'c \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c \Rightarrow 'b{::}\{complete\text{-}lattice, countable\})$
assumes $P\ M\ s$
assumes F: *inf-continuous* F
assumes $*$: $\bigwedge M\ A\ s.\ P\ M\ s \Longrightarrow (\bigwedge N\ t.\ P\ N\ t \Longrightarrow A\ t \in measurable\ N\ (count\text{-}space\ UNIV)) \Longrightarrow F\ A\ s \in measurable\ M\ (count\text{-}space\ UNIV)$
shows $gfp\ F\ s \in measurable\ M\ (count\text{-}space\ UNIV)$
proof −
{ fix i **from** ‹$P\ M\ s$› **have** $(\lambda x.\ (F\ \hat{}\hat{}\ i)\ top\ s\ x) \in measurable\ M\ (count\text{-}space\ UNIV)$
by (*induct i arbitrary*: $M\ s$) (*auto intro*!: $*$) **}**
then have $(\lambda x.\ INF\ i.\ (F\ \hat{}\hat{}\ i)\ top\ s\ x) \in measurable\ M\ (count\text{-}space\ UNIV)$
by *measurable*
also have $(\lambda x.\ INF\ i.\ (F\ \hat{}\hat{}\ i)\ top\ s\ x) = gfp\ F\ s$
by (*subst inf-continuous-gfp*) (*auto simp*: F)
finally show *?thesis* **.**
qed

lemma *measurable-enat-coinduct*:
fixes $f :: 'a \Rightarrow enat$
assumes $R\ f$
assumes $*$: $\bigwedge f.\ R\ f \Longrightarrow \exists g\ h\ i\ P.\ R\ g \wedge f = (\lambda x.\ if\ P\ x\ then\ h\ x\ else\ eSuc\ (g\ (i\ x))) \wedge$
Measurable.pred M P $\wedge$
$i \in measurable\ M\ M\ \wedge$
$h \in measurable\ M\ (count\text{-}space\ UNIV)$
shows $f \in measurable\ M\ (count\text{-}space\ UNIV)$
proof (*simp add*: *measurable-count-space-eq2-countable*, *rule*)
fix $a :: enat$
have $f -` \{a\} \cap space\ M = \{x{\in}space\ M.\ f\ x = a\}$
by *auto*
{ fix $i :: nat$
from ‹$R\ f$› **have** *Measurable.pred M* $(\lambda x.\ f\ x = enat\ i)$
proof (*induction i arbitrary*: f)
case *0*

```
      from ∗[OF this] obtain g h i P
        where f: f = (λx. if P x then h x else eSuc (g (i x))) and
          [measurable]: Measurable.pred M P i ∈ measurable M M h ∈ measurable
M (count-space UNIV)
        by auto
      have Measurable.pred M (λx. P x ∧ h x = 0)
        by measurable
      also have (λx. P x ∧ h x = 0) = (λx. f x = enat 0)
        by (auto simp: f zero-enat-def[symmetric])
      finally show ?case .
    next
      case (Suc n)
      from ∗[OF Suc.prems] obtain g h i P
        where f: f = (λx. if P x then h x else eSuc (g (i x))) and R g and
          M[measurable]: Measurable.pred M P i ∈ measurable M M h ∈ measurable
M (count-space UNIV)
        by auto
      have (λx. f x = enat (Suc n)) =
        (λx. (P x ⟶ h x = enat (Suc n)) ∧ (¬ P x ⟶ g (i x) = enat n))
        by (auto simp: f zero-enat-def[symmetric] eSuc-enat[symmetric])
      also have Measurable.pred M ...
          by (intro pred-intros-logic measurable-compose[OF M(2)] Suc ‹R g›)
measurable
      finally show ?case .
    qed
    then have f -` {enat i} ∩ space M ∈ sets M
      by (simp add: pred-def Int-def conj-commute) }
  note fin = this
  show f -` {a} ∩ space M ∈ sets M
  proof (cases a)
    case infinity
    then have f -` {a} ∩ space M = space M - (⋃n. f -` {enat n} ∩ space M)
      by auto
    also have ... ∈ sets M
      by (intro sets.Diff sets.top sets.Un sets.countable-UN) (auto intro!: fin)
    finally show ?thesis .
  qed (simp add: fin)
qed

lemma measurable-THE:
  fixes P :: 'a ⇒ 'b ⇒ bool
  assumes [measurable]: ⋀i. Measurable.pred M (P i)
  assumes I[simp]: countable I ⋀i x. x ∈ space M ⟹ P i x ⟹ i ∈ I
  assumes unique: ⋀x i j. x ∈ space M ⟹ P i x ⟹ P j x ⟹ i = j
  shows (λx. THE i. P i x) ∈ measurable M (count-space UNIV)
  unfolding measurable-def
proof safe
  fix X
  def f ≡ λx. THE i. P i x def undef ≡ THE i::'a. False
```

{ fix *i x* **assume** *x* ∈ *space M P i x* **then have** *f x* = *i*
unfolding *f-def* **using** *unique* **by** *auto* **}**
note *f-eq* = *this*
{ fix *x* **assume** *x* ∈ *space M* ∀ *i*∈*I*. ¬ *P i x*
then have ⋀*i*. ¬ *P i x*
using *I*(*2*)[*of x*] **by** *auto*
then have *f x* = *undef*
by (*auto simp*: *undef-def f-def*) **}**
then have *f* −‘ *X* ∩ *space M* = (⋃ *i*∈*I* ∩ *X*. {*x*∈*space M*. *P i x*}) ∪
(*if undef* ∈ *X then space M* − (⋃ *i*∈*I*. {*x*∈*space M*. *P i x*}) *else* {})
by (*auto dest*: *f-eq*)
also have ... ∈ *sets M*
by (*auto intro*!: *sets.Diff sets.countable-UN'*)
finally show *f* −‘ *X* ∩ *space M* ∈ *sets M* **.**
qed *simp*

lemma *measurable-Ex1*[*measurable* (*raw*)]:
assumes [*simp*]: *countable I* **and** [*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *Measurable.pred M* (*P i*)
shows *Measurable.pred M* (λ*x*. ∃!*i*∈*I*. *P i x*)
unfolding *bex1-def* **by** *measurable*

lemma *measurable-Sup-nat*[*measurable* (*raw*)]:
fixes *F* :: *'a* ⇒ *nat set*
assumes [*measurable*]: ⋀*i*. *Measurable.pred M* (λ*x*. *i* ∈ *F x*)
shows (λ*x*. *Sup* (*F x*)) ∈ *M* →*M* *count-space UNIV*
proof (*clarsimp simp add*: *measurable-count-space-eq2-countable*)
fix *a*
have *F-empty-iff*: *F x* = {} ⟷ (∀ *i*. *i* ∉ *F x*) **for** *x*
by *auto*
have *Measurable.pred M* (λ*x*. *if finite* (*F x*) *then if F x* = {} *then a* = *Max* {}
else a ∈ *F x* ∧ (∀ *j*. *j* ∈ *F x* ⟶ *j* ≤ *a*) *else a* = *the None*)
unfolding *finite-nat-set-iff-bounded Ball-def F-empty-iff* **by** *measurable*
moreover have (λ*x*. *Sup* (*F x*)) −‘ {*a*} ∩ *space M* =
{*x*∈*space M*. *if finite* (*F x*) *then if F x* = {} *then a* = *Max* {}
else a ∈ *F x* ∧ (∀ *j*. *j* ∈ *F x* ⟶ *j* ≤ *a*) *else a* = *the None*}
by (*intro set-eqI*)
(*auto simp*: *Sup-nat-def Max.infinite intro*!: *Max-in Max-eqI*)
ultimately show (λ*x*. *Sup* (*F x*)) −‘ {*a*} ∩ *space M* ∈ *sets M*
by *auto*
qed

lemma *measurable-if-split*[*measurable* (*raw*)]:
(*c* ⟹ *Measurable.pred M f*) ⟹ (¬ *c* ⟹ *Measurable.pred M g*) ⟹
Measurable.pred M (*if c then f else g*)
by *simp*

lemma *pred-restrict-space*:
assumes *S* ∈ *sets M*

shows *Measurable.pred* (*restrict-space M S*) *P* ⟷ *Measurable.pred M* (λ*x*. *x* ∈ *S* ∧ *P x*)
unfolding *pred-def sets-Collect-restrict-space-iff*[*OF assms*] **..**

lemma *measurable-predpow*[*measurable*]:
assumes *Measurable.pred M T*
assumes ⋀*Q*. *Measurable.pred M Q* ⟹ *Measurable.pred M* (*R Q*)
shows *Measurable.pred M* ((*R* ^^ *n*) *T*)
by (*induct n*) (*auto intro*: *assms*)

hide-const (**open**) *pred*

end

3 Measure spaces and their properties

theory *Measure-Space*
imports
Measurable ~~/*src*/*HOL*/*Multivariate-Analysis*/*Multivariate-Analysis*
begin

3.1 Relate extended reals and the indicator function

lemma *suminf-cmult-indicator*:
fixes *f* :: *nat* ⇒ *ennreal*
assumes *disjoint-family A x* ∈ *A i*
shows (∑ *n*. *f n* ∗ *indicator* (*A n*) *x*) = *f i*
proof −
have ∗∗: ⋀*n*. *f n* ∗ *indicator* (*A n*) *x* = (*if n* = *i then f n else* 0 :: *ennreal*)
using ‹*x* ∈ *A i*› *assms* **unfolding** *disjoint-family-on-def indicator-def* **by** *auto*
then have ⋀*n*. (∑ *j*<*n*. *f j* ∗ *indicator* (*A j*) *x*) = (*if i* < *n then f i else* 0 :: *ennreal*)
by (*auto simp*: *setsum.If-cases*)
moreover have (*SUP n*. *if i* < *n then f i else* 0) = (*f i* :: *ennreal*)
proof (*rule SUP-eqI*)
fix *y* :: *ennreal* **assume** ⋀*n*. *n* ∈ *UNIV* ⟹ (*if i* < *n then f i else* 0) ≤ *y*
from *this*[*of Suc i*] **show** *f i* ≤ *y* **by** *auto*
qed (*insert assms*, *simp add*: *zero-le*)
ultimately show *?thesis* **using** *assms*
by (*subst suminf-eq-SUP*) (*auto simp*: *indicator-def*)
qed

lemma *suminf-indicator*:
assumes *disjoint-family A*
shows (∑ *n*. *indicator* (*A n*) *x* :: *ennreal*) = *indicator* (⋃ *i*. *A i*) *x*
proof *cases*
assume ∗: *x* ∈ (⋃ *i*. *A i*)
then obtain *i* **where** *x* ∈ *A i* **by** *auto*
from *suminf-cmult-indicator*[*OF assms*(1), *OF* ‹*x* ∈ *A i*›, *of* λ*k*. 1]

show *?thesis* **using** $*$ **by** *simp*
qed *simp*

lemma *setsum-indicator-disjoint-family*:
fixes $f :: 'd \Rightarrow 'e$::*semiring-1*
assumes *d*: *disjoint-family-on A P* **and** $x \in A\ j$ **and** *finite P* **and** $j \in P$
shows $(\sum i{\in}P.\ f\ i * \mathit{indicator}\ (A\ i)\ x) = f\ j$
proof −
have $P \cap \{i.\ x \in A\ i\} = \{j\}$
using *d* ‹$x \in A\ j$› ‹$j \in P$› **unfolding** *disjoint-family-on-def*
by *auto*
thus *?thesis*
unfolding *indicator-def*
by (*simp add*: *if-distrib setsum.If-cases*[*OF* ‹*finite P*›])
qed

The type for emeasure spaces is already defined in *Sigma-Algebra*, as it is also used to represent sigma algebras (with an arbitrary emeasure).

3.2 Extend binary sets

lemma *LIMSEQ-binaryset*:
assumes *f*: $f\ \{\} = 0$
shows $(\lambda n.\ \sum i{<}n.\ f\ (\mathit{binaryset}\ A\ B\ i)) \longrightarrow f\ A + f\ B$
proof −
have $(\lambda n.\ \sum i < \mathit{Suc}\ (\mathit{Suc}\ n).\ f\ (\mathit{binaryset}\ A\ B\ i)) = (\lambda n.\ f\ A + f\ B)$
proof
fix *n*
show $(\sum i < \mathit{Suc}\ (\mathit{Suc}\ n).\ f\ (\mathit{binaryset}\ A\ B\ i)) = f\ A + f\ B$
by (*induct n*) (*auto simp add*: *binaryset-def f*)
qed
moreover
have ... $\longrightarrow f\ A + f\ B$ **by** (*rule tendsto-const*)
ultimately
have $(\lambda n.\ \sum i{<}\ \mathit{Suc}\ (\mathit{Suc}\ n).\ f\ (\mathit{binaryset}\ A\ B\ i)) \longrightarrow f\ A + f\ B$
by *metis*
hence $(\lambda n.\ \sum i{<}\ n{+}2.\ f\ (\mathit{binaryset}\ A\ B\ i)) \longrightarrow f\ A + f\ B$
by *simp*
thus *?thesis* **by** (*rule LIMSEQ-offset* [**where** *k*=*2*])
qed

lemma *binaryset-sums*:
assumes *f*: $f\ \{\} = 0$
shows $(\lambda n.\ f\ (\mathit{binaryset}\ A\ B\ n))\ \mathit{sums}\ (f\ A + f\ B)$
by (*simp add*: *sums-def LIMSEQ-binaryset* [**where** *f*=*f*, *OF f*] *atLeast0LessThan*)

lemma *suminf-binaryset-eq*:
fixes $f :: 'a\ \mathit{set} \Rightarrow 'b$::{*comm-monoid-add*, *t2-space*}
shows $f\ \{\} = 0 \Longrightarrow (\sum n.\ f\ (\mathit{binaryset}\ A\ B\ n)) = f\ A + f\ B$

by (*metis binaryset-sums sums-unique*)

3.3 Properties of a premeasure μ

The definitions for *positive* and *countably-additive* should be here, by they are necessary to define *'a measure* in *Sigma-Algebra*.

definition *subadditive* **where**
subadditive M f $\longleftrightarrow$ $(\forall\, x{\in}M.\ \forall\, y{\in}M.\ x \cap y = \{\} \longrightarrow f\ (x \cup y) \leq f\, x + f\, y)$

lemma *subadditiveD*: *subadditive M f* $\Longrightarrow$ $x \cap y = \{\} \Longrightarrow x \in M \Longrightarrow y \in M \Longrightarrow$ $f\ (x \cup y) \leq f\, x + f\, y$
by (*auto simp add*: *subadditive-def*)

definition *countably-subadditive* **where**
countably-subadditive M f $\longleftrightarrow$
$(\forall\, A.\ range\ A \subseteq M \longrightarrow disjoint\text{-}family\ A \longrightarrow (\bigcup i.\ A\ i) \in M \longrightarrow (f\ (\bigcup i.\ A\ i) \leq (\sum i.\ f\ (A\ i))))$

lemma (**in** *ring-of-sets*) *countably-subadditive-subadditive*:
fixes *f* :: *'a set* $\Rightarrow$ *ennreal*
assumes *f*: *positive M f* **and** *cs*: *countably-subadditive M f*
shows *subadditive M f*
proof (*auto simp add*: *subadditive-def*)
fix *x y*
assume *x*: $x \in M$ **and** *y*: $y \in M$ **and** $x \cap y = \{\}$
hence *disjoint-family* (*binaryset x y*)
by (*auto simp add*: *disjoint-family-on-def binaryset-def*)
hence *range* (*binaryset x y*) $\subseteq M \longrightarrow$
$(\bigcup i.\ binaryset\ x\ y\ i) \in M \longrightarrow$
$f\ (\bigcup i.\ binaryset\ x\ y\ i) \leq (\sum\ n.\ f\ (binaryset\ x\ y\ n))$
using *cs* **by** (*auto simp add*: *countably-subadditive-def*)
hence $\{x,y,\{\}\} \subseteq M \longrightarrow x \cup y \in M \longrightarrow$
$f\ (x \cup y) \leq (\sum\ n.\ f\ (binaryset\ x\ y\ n))$
by (*simp add*: *range-binaryset-eq UN-binaryset-eq*)
thus $f\ (x \cup y) \leq\ f\, x + f\, y$ **using** *f x y*
by (*auto simp add*: *Un o-def suminf-binaryset-eq positive-def*)
qed

definition *additive* **where**
additive M μ $\longleftrightarrow$ $(\forall\, x{\in}M.\ \forall\, y{\in}M.\ x \cap y = \{\} \longrightarrow \mu\ (x \cup y) = \mu\ x + \mu\ y)$

definition *increasing* **where**
increasing M μ $\longleftrightarrow$ $(\forall\, x{\in}M.\ \forall\, y{\in}M.\ x \subseteq y \longrightarrow \mu\ x \leq \mu\ y)$

lemma *positiveD1*: *positive M f* $\Longrightarrow$ $f\ \{\} = 0$ **by** (*auto simp*: *positive-def*)

lemma *positiveD-empty*:
positive M f $\Longrightarrow$ $f\ \{\} = 0$
by (*auto simp add*: *positive-def*)

lemma *additiveD*:
additive M f $\Longrightarrow$ $x \cap y = \{\} \Longrightarrow x \in M \Longrightarrow y \in M \Longrightarrow f\ (x \cup y) = f\ x + f\ y$
by (*auto simp add*: *additive-def*)

lemma *increasingD*:
increasing M f $\Longrightarrow$ $x \subseteq y \Longrightarrow x{\in}M \Longrightarrow y{\in}M \Longrightarrow f\ x \leq f\ y$
by (*auto simp add*: *increasing-def*)

lemma *countably-additiveI*[*case-names countably*]:
($\bigwedge A$. *range A* $\subseteq M \Longrightarrow$ *disjoint-family A* $\Longrightarrow$ $(\bigcup i.\ A\ i) \in M \Longrightarrow (\sum i.\ f\ (A\ i)) = f\ (\bigcup i.\ A\ i)$)
$\Longrightarrow$ *countably-additive M f*
by (*simp add*: *countably-additive-def*)

lemma (**in** *ring-of-sets*) *disjointed-additive*:
assumes *f*: *positive M f* **and** *additive M f* **and** *A*: *range A* $\subseteq$ *M incseq A*
shows $(\sum i{\leq}n.\ f\ (\mathit{disjointed}\ A\ i)) = f\ (A\ n)$
proof (*induct n*)
case (*Suc n*)
then have $(\sum i{\leq}\mathit{Suc}\ n.\ f\ (\mathit{disjointed}\ A\ i)) = f\ (A\ n) + f\ (\mathit{disjointed}\ A\ (\mathit{Suc}\ n))$
by *simp*
also have ... $= f\ (A\ n \cup \mathit{disjointed}\ A\ (\mathit{Suc}\ n))$
using *A* **by** (*subst f*(*2*)[*THEN additiveD*]) (*auto simp*: *disjointed-mono*)
also have $A\ n \cup \mathit{disjointed}\ A\ (\mathit{Suc}\ n) = A\ (\mathit{Suc}\ n)$
using ‹*incseq A*› **by** (*auto dest*: *incseq-SucD simp*: *disjointed-mono*)
finally show *?case* **.**
qed *simp*

lemma (**in** *ring-of-sets*) *additive-sum*:
fixes *A*:: *'i* $\Rightarrow$ *'a set*
assumes *f*: *positive M f* **and** *ad*: *additive M f* **and** *finite S*
and *A*: *A*`*S* $\subseteq$ *M*
and *disj*: *disjoint-family-on A S*
shows $(\sum i{\in}S.\ f\ (A\ i)) = f\ (\bigcup i{\in}S.\ A\ i)$
using ‹*finite S*› *disj A*
proof *induct*
case *empty* **show** *?case* **using** *f* **by** (*simp add*: *positive-def*)
next
case (*insert s S*)
then have $A\ s \cap (\bigcup i{\in}S.\ A\ i) = \{\}$
by (*auto simp add*: *disjoint-family-on-def neq-iff*)
moreover
have $A\ s \in M$ **using** *insert* **by** *blast*
moreover have $(\bigcup i{\in}S.\ A\ i) \in M$
using *insert* ‹*finite S*› **by** *auto*
ultimately have $f\ (A\ s \cup (\bigcup i{\in}S.\ A\ i)) = f\ (A\ s) + f(\bigcup i{\in}S.\ A\ i)$
using *ad UNION-in-sets A* **by** (*auto simp add*: *additive-def*)

with *insert* **show** *?case* **using** *ad disjoint-family-on-mono*[*of S insert s S A*]
by (*auto simp add*: *additive-def subset-insertI*)
qed

lemma (**in** *ring-of-sets*) *additive-increasing*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes *posf*: *positive M f* **and** *addf*: *additive M f*
shows *increasing M f*
proof (*auto simp add*: *increasing-def*)
fix *x y*
assume *xy*: $x \in M\ y \in M\ x \subseteq y$
then have $y - x \in M$ **by** *auto*
then have $f\ x + 0 \leq f\ x + f\ (y-x)$ **by** (*intro add-left-mono zero-le*)
also have $... = f\ (x \cup (y-x))$ **using** *addf*
by (*auto simp add*: *additive-def*) (*metis Diff-disjoint Un-Diff-cancel Diff xy*(*1*,*2*))
also have $... = f\ y$
by (*metis Un-Diff-cancel Un-absorb1 xy*(*3*))
finally show $f\ x \leq f\ y$ **by** *simp*
qed

lemma (**in** *ring-of-sets*) *subadditive*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes *f*: *positive M f additive M f* **and** *A*: *A'S* $\subseteq M$ **and** *S*: *finite S*
shows $f\ (\bigcup i{\in}S.\ A\ i) \leq (\sum i{\in}S.\ f\ (A\ i))$
using *S A*
proof (*induct S*)
case *empty* **thus** *?case* **using** *f* **by** (*auto simp*: *positive-def*)
next
case (*insert x F*)
hence *in-M*: $A\ x \in M\ (\bigcup i{\in}F.\ A\ i) \in M\ (\bigcup i{\in}F.\ A\ i) - A\ x \in M$ **using** *A*
by *force+*
have *subs*: $(\bigcup i{\in}F.\ A\ i) - A\ x \subseteq (\bigcup i{\in}F.\ A\ i)$ **by** *auto*
have $(\bigcup i{\in}(insert\ x\ F).\ A\ i) = A\ x \cup ((\bigcup i{\in}F.\ A\ i) - A\ x)$ **by** *auto*
hence $f\ (\bigcup i{\in}(insert\ x\ F).\ A\ i) = f\ (A\ x \cup ((\bigcup i{\in}F.\ A\ i) - A\ x))$
by *simp*
also have $\ldots = f\ (A\ x) + f\ ((\bigcup i{\in}F.\ A\ i) - A\ x)$
using *f*(*2*) **by** (*rule additiveD*) (*insert in-M*, *auto*)
also have $\ldots \leq f\ (A\ x) + f\ (\bigcup i{\in}F.\ A\ i)$
using *additive-increasing*[*OF f*] *in-M subs* **by** (*auto simp*: *increasing-def intro*: *add-left-mono*)
also have $\ldots \leq f\ (A\ x) + (\sum i{\in}F.\ f\ (A\ i))$ **using** *insert* **by** (*auto intro*: *add-left-mono*)
finally show $f\ (\bigcup i{\in}(insert\ x\ F).\ A\ i) \leq (\sum i{\in}(insert\ x\ F).\ f\ (A\ i))$ **using** *insert* **by** *simp*
qed

lemma (**in** *ring-of-sets*) *countably-additive-additive*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes *posf*: *positive M f* **and** *ca*: *countably-additive M f*

shows *additive M f*
proof (*auto simp add*: *additive-def*)
fix *x y*
assume *x*: $x \in M$ **and** *y*: $y \in M$ **and** $x \cap y = \{\}$
hence *disjoint-family* (*binaryset x y*)
by (*auto simp add*: *disjoint-family-on-def binaryset-def*)
hence *range* (*binaryset x y*) $\subseteq M \longrightarrow$
$(\bigcup i.\ binaryset\ x\ y\ i) \in M \longrightarrow$
$f\ (\bigcup i.\ binaryset\ x\ y\ i) = (\sum\ n.\ f\ (binaryset\ x\ y\ n))$
using *ca*
by (*simp add*: *countably-additive-def*)
hence $\{x,y,\{\}\} \subseteq M \longrightarrow x \cup y \in M \longrightarrow$
$f\ (x \cup y) = (\sum n.\ f\ (binaryset\ x\ y\ n))$
by (*simp add*: *range-binaryset-eq UN-binaryset-eq*)
thus $f\ (x \cup y) = f\ x + f\ y$ **using** *posf x y*
by (*auto simp add*: *Un suminf-binaryset-eq positive-def*)
qed

lemma (**in** *algebra*) *increasing-additive-bound*:
fixes *A*:: *nat* $\Rightarrow$ *'a set* **and** *f* :: *'a set* $\Rightarrow$ *ennreal*
assumes *f*: *positive M f* **and** *ad*: *additive M f*
and *inc*: *increasing M f*
and *A*: *range A* $\subseteq M$
and *disj*: *disjoint-family A*
shows $(\sum i.\ f\ (A\ i)) \leq f\ \Omega$
proof (*safe intro*!: *suminf-le-const*)
fix *N*
note *disj-N* = *disjoint-family-on-mono*[*OF - disj*, *of* $\{..<N\}$]
have $(\sum i<N.\ f\ (A\ i)) = f\ (\bigcup i \in \{..<N\}.\ A\ i)$
using *A* **by** (*intro additive-sum* [*OF f ad - -*]) (*auto simp*: *disj-N*)
also have ... $\leq f\ \Omega$ **using** *space-closed A*
by (*intro increasingD*[*OF inc*] *finite-UN*) *auto*
finally show $(\sum i<N.\ f\ (A\ i)) \leq f\ \Omega$ **by** *simp*
qed (*insert f A*, *auto simp*: *positive-def*)

lemma (**in** *ring-of-sets*) *countably-additiveI-finite*:
fixes μ :: *'a set* $\Rightarrow$ *ennreal*
assumes *finite* Ω *positive M* μ *additive M* μ
shows *countably-additive M* μ
proof (*rule countably-additiveI*)
fix *F* :: *nat* $\Rightarrow$ *'a set* **assume** *F*: *range F* $\subseteq M$ $(\bigcup i.\ F\ i) \in M$ **and** *disj*: *disjoint-family F*

have $\forall i \in \{i.\ F\ i \neq \{\}\}.\ \exists x.\ x \in F\ i$ **by** *auto*
from *bchoice*[*OF this*] **obtain** *f* **where** *f*: $\bigwedge i.\ F\ i \neq \{\} \Longrightarrow f\ i \in F\ i$ **by** *auto*

have *inj-f*: *inj-on f* $\{i.\ F\ i \neq \{\}\}$
proof (*rule inj-onI*, *simp*)
fix *i j a b* **assume** $*$: $f\ i = f\ j\ F\ i \neq \{\}\ F\ j \neq \{\}$

```
    then have f i ∈ F i f j ∈ F j using f by force+
    with disj ∗ show i = j by (auto simp: disjoint-family-on-def)
  qed
  have finite (⋃ i. F i)
    by (metis F(2) assms(1) infinite-super sets-into-space)

  have F-subset: {i. μ (F i) ≠ 0} ⊆ {i. F i ≠ {}}
    by (auto simp: positiveD-empty[OF ‹positive M μ›])
  moreover have fin-not-empty: finite {i. F i ≠ {}}
  proof (rule finite-imageD)
    from f have f‘{i. F i ≠ {}} ⊆ (⋃ i. F i) by auto
    then show finite (f‘{i. F i ≠ {}})
      by (rule finite-subset) fact
  qed fact
  ultimately have fin-not-0: finite {i. μ (F i) ≠ 0}
    by (rule finite-subset)

  have disj-not-empty: disjoint-family-on F {i. F i ≠ {}}
    using disj by (auto simp: disjoint-family-on-def)

  from fin-not-0 have (∑ i. μ (F i)) = (∑ i | μ (F i) ≠ 0. μ (F i))
    by (rule suminf-finite) auto
  also have ... = (∑ i | F i ≠ {}. μ (F i))
    using fin-not-empty F-subset by (rule setsum.mono-neutral-left) auto
  also have ... = μ (⋃ i∈{i. F i ≠ {}}. F i)
    using ‹positive M μ› ‹additive M μ› fin-not-empty disj-not-empty F by (intro
additive-sum) auto
  also have ... = μ (⋃ i. F i)
    by (rule arg-cong[where f=μ]) auto
  finally show (∑ i. μ (F i)) = μ (⋃ i. F i) .
qed

lemma (in ring-of-sets) countably-additive-iff-continuous-from-below:
  fixes f :: 'a set ⇒ ennreal
  assumes f: positive M f additive M f
  shows countably-additive M f ⟷
    (∀ A. range A ⊆ M ⟶ incseq A ⟶ (⋃ i. A i) ∈ M ⟶ (λi. f (A i)) ⟶
f (⋃ i. A i))
  unfolding countably-additive-def
proof safe
  assume count-sum: ∀ A. range A ⊆ M ⟶ disjoint-family A ⟶ UNION UNIV
A ∈ M ⟶ (∑ i. f (A i)) = f (UNION UNIV A)
  fix A :: nat ⇒ 'a set assume A: range A ⊆ M incseq A (⋃ i. A i) ∈ M
  then have dA: range (disjointed A) ⊆ M by (auto simp: range-disjointed-sets)
  with count-sum[THEN spec, of disjointed A] A(3)
  have f-UN: (∑ i. f (disjointed A i)) = f (⋃ i. A i)
    by (auto simp: UN-disjointed-eq disjoint-family-disjointed)
  moreover have (λn. (∑ i<n. f (disjointed A i))) ⟶ (∑ i. f (disjointed A
i))
```

using *f(1)*[*unfolded positive-def*] *dA*
by (*auto intro*!: *summable-LIMSEQ summableI*)
from *LIMSEQ-Suc*[*OF this*]
have $(\lambda n.\ (\sum i{\le}n.\ f\ (\mathit{disjointed}\ A\ i))) \longrightarrow (\sum i.\ f\ (\mathit{disjointed}\ A\ i))$
unfolding *lessThan-Suc-atMost* **.**
moreover have $\bigwedge n.\ (\sum i{\le}n.\ f\ (\mathit{disjointed}\ A\ i)) = f\ (A\ n)$
using *disjointed-additive*[*OF f A(1,2)*] **.**
ultimately show $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$ **by** *simp*
next
assume *cont*: $\forall A.\ \mathit{range}\ A \subseteq M \longrightarrow \mathit{incseq}\ A \longrightarrow (\bigcup i.\ A\ i) \in M \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A* ⊆ *M disjoint-family A* $(\bigcup i.\ A\ i) \in M$
have ∗: $(\bigcup n.\ (\bigcup i{<}n.\ A\ i)) = (\bigcup i.\ A\ i)$ **by** *auto*
have $(\lambda n.\ f\ (\bigcup i{<}n.\ A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
proof (*unfold* ∗[*symmetric*], *intro cont*[*rule-format*])
show *range* $(\lambda i.\ \bigcup i{<}i.\ A\ i) \subseteq M$ $(\bigcup i.\ \bigcup i{<}i.\ A\ i) \in M$
using *A* ∗ **by** *auto*
qed (*force intro*!: *incseq-SucI*)
moreover have $\bigwedge n.\ f\ (\bigcup i{<}n.\ A\ i) = (\sum i{<}n.\ f\ (A\ i))$
using *A*
by (*intro additive-sum*[*OF f*, *of* - *A*, *symmetric*])
(*auto intro*: *disjoint-family-on-mono*[**where** *B*=*UNIV*])
ultimately
have $(\lambda i.\ f\ (A\ i))$ *sums* $f\ (\bigcup i.\ A\ i)$
unfolding *sums-def* **by** *simp*
from *sums-unique*[*OF this*]
show $(\sum i.\ f\ (A\ i)) = f\ (\bigcup i.\ A\ i)$ **by** *simp*
qed

lemma (**in** *ring-of-sets*) *continuous-from-above-iff-empty-continuous*:
fixes *f* :: *'a set* ⇒ *ennreal*
assumes *f*: *positive M f additive M f*
shows $(\forall A.\ \mathit{range}\ A \subseteq M \longrightarrow \mathit{decseq}\ A \longrightarrow (\bigcap i.\ A\ i) \in M \longrightarrow (\forall i.\ f\ (A\ i) \neq \infty) \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcap i.\ A\ i))$
$\longleftrightarrow (\forall A.\ \mathit{range}\ A \subseteq M \longrightarrow \mathit{decseq}\ A \longrightarrow (\bigcap i.\ A\ i) = \{\} \longrightarrow (\forall i.\ f\ (A\ i) \neq \infty) \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow 0)$
proof *safe*
assume *cont*: $(\forall A.\ \mathit{range}\ A \subseteq M \longrightarrow \mathit{decseq}\ A \longrightarrow (\bigcap i.\ A\ i) \in M \longrightarrow (\forall i.\ f\ (A\ i) \neq \infty) \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcap i.\ A\ i))$
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A* ⊆ *M decseq A* $(\bigcap i.\ A\ i) = \{\}$ $\forall i.\ f\ (A\ i) \neq \infty$
with *cont*[*THEN spec*, *of A*] **show** $(\lambda i.\ f\ (A\ i)) \longrightarrow 0$
using ‹*positive M f*›[*unfolded positive-def*] **by** *auto*
next
assume *cont*: $\forall A.\ \mathit{range}\ A \subseteq M \longrightarrow \mathit{decseq}\ A \longrightarrow (\bigcap i.\ A\ i) = \{\} \longrightarrow (\forall i.\ f\ (A\ i) \neq \infty) \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow 0$
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A* ⊆ *M decseq A* $(\bigcap i.\ A\ i) \in M$ $\forall i.\ f\ (A\ i) \neq \infty$

have *f-mono*: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \subseteq b \Longrightarrow f\ a \leq f\ b$
using *additive-increasing*[*OF f*] **unfolding** *increasing-def* **by** *simp*

have *decseq-fA*: *decseq* $(\lambda i.\ f\ (A\ i))$
using *A* **by** (*auto simp*: *decseq-def intro*!: *f-mono*)
have *decseq*: *decseq* $(\lambda i.\ A\ i - (\bigcap i.\ A\ i))$
using *A* **by** (*auto simp*: *decseq-def*)
then have *decseq-f*: *decseq* $(\lambda i.\ f\ (A\ i - (\bigcap i.\ A\ i)))$
using *A* **unfolding** *decseq-def* **by** (*auto intro*!: *f-mono Diff*)
have $f\ (\bigcap x.\ A\ x) \leq f\ (A\ 0)$
using *A* **by** (*auto intro*!: *f-mono*)
then have *f-Int-fin*: $f\ (\bigcap x.\ A\ x) \neq \infty$
using *A* **by** (*auto simp*: *top-unique*)
{ fix *i*
have $f\ (A\ i - (\bigcap i.\ A\ i)) \leq f\ (A\ i)$ **using** *A* **by** (*auto intro*!: *f-mono*)
then have $f\ (A\ i - (\bigcap i.\ A\ i)) \neq \infty$
using *A* **by** (*auto simp*: *top-unique*) **}**
note *f-fin* = *this*
have $(\lambda i.\ f\ (A\ i - (\bigcap i.\ A\ i))) \longrightarrow 0$
proof (*intro cont*[*rule-format*, *OF - decseq - f-fin*])
show *range* $(\lambda i.\ A\ i - (\bigcap i.\ A\ i)) \subseteq M$ $(\bigcap i.\ A\ i - (\bigcap i.\ A\ i)) = \{\}$
using *A* **by** *auto*
qed
from *INF-Lim-ereal*[*OF decseq-f this*]
have $(INF\ n.\ f\ (A\ n - (\bigcap i.\ A\ i))) = 0$ **.**
moreover have $(INF\ n.\ f\ (\bigcap i.\ A\ i)) = f\ (\bigcap i.\ A\ i)$
by *auto*
ultimately have $(INF\ n.\ f\ (A\ n - (\bigcap i.\ A\ i)) + f\ (\bigcap i.\ A\ i)) = 0 + f\ (\bigcap i.\ A\ i)$
using *A*(*4*) *f-fin f-Int-fin*
by (*subst INF-ennreal-add-const*) (*auto simp*: *decseq-f*)
moreover {
fix *n*
have $f\ (A\ n - (\bigcap i.\ A\ i)) + f\ (\bigcap i.\ A\ i) = f\ ((A\ n - (\bigcap i.\ A\ i)) \cup (\bigcap i.\ A\ i))$
using *A* **by** (*subst f*(*2*)[*THEN additiveD*]) *auto*
also have $(A\ n - (\bigcap i.\ A\ i)) \cup (\bigcap i.\ A\ i) = A\ n$
by *auto*
finally have $f\ (A\ n - (\bigcap i.\ A\ i)) + f\ (\bigcap i.\ A\ i) = f\ (A\ n)$ **. }**
ultimately have $(INF\ n.\ f\ (A\ n)) = f\ (\bigcap i.\ A\ i)$
by *simp*
with *LIMSEQ-INF*[*OF decseq-fA*]
show $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcap i.\ A\ i)$ **by** *simp*
qed

lemma (**in** *ring-of-sets*) *empty-continuous-imp-continuous-from-below*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes *f*: *positive M f additive M f* $\forall A{\in}M.\ f\ A \neq \infty$
assumes *cont*: $\forall A.$ *range* $A \subseteq M \longrightarrow$ *decseq* $A \longrightarrow (\bigcap i.\ A\ i) = \{\} \longrightarrow (\lambda i.\ f$

$(A\ i)) \longrightarrow 0$
assumes *A*: *range A* $\subseteq$ *M incseq A* $(\bigcup i.\ A\ i) \in M$
shows $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
proof −
from *A* **have** $(\lambda i.\ f\ ((\bigcup i.\ A\ i) - A\ i)) \longrightarrow 0$
by (*intro cont*[*rule-format*]) (*auto simp*: *decseq-def incseq-def*)
moreover
{ fix *i*
have $f\ ((\bigcup i.\ A\ i) - A\ i \cup A\ i) = f\ ((\bigcup i.\ A\ i) - A\ i) + f\ (A\ i)$
using *A* **by** (*intro f*(*2*)[*THEN additiveD*]) *auto*
also have $((\bigcup i.\ A\ i) - A\ i) \cup A\ i = (\bigcup i.\ A\ i)$
by *auto*
finally have $f\ ((\bigcup i.\ A\ i) - A\ i) = f\ (\bigcup i.\ A\ i) - f\ (A\ i)$
using *f*(*3*)[*rule-format*, *of A i*] *A* **by** (*auto simp*: *ennreal-add-diff-cancel subset-eq*) **}**
moreover have $\forall_F\ i$ *in sequentially*. $f\ (A\ i) \leq f\ (\bigcup i.\ A\ i)$
using *increasingD*[*OF additive-increasing*[*OF f*(*1*, *2*)], *of A* - $\bigcup i.\ A\ i$] *A*
by (*auto intro*!: *always-eventually simp*: *subset-eq*)
ultimately show $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
by (*auto intro*: *ennreal-tendsto-const-minus*)
qed

lemma (**in** *ring-of-sets*) *empty-continuous-imp-countably-additive*:
fixes *f* :: *'a set* $\Rightarrow$ *ennreal*
assumes *f*: *positive M f additive M f* **and** *fin*: $\forall A{\in}M.\ f\ A \neq \infty$
assumes *cont*: $\bigwedge A.$ *range A* $\subseteq$ *M* $\Longrightarrow$ *decseq A* $\Longrightarrow$ $(\bigcap i.\ A\ i) = \{\} \Longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow 0$
shows *countably-additive M f*
using *countably-additive-iff-continuous-from-below*[*OF f*]
using *empty-continuous-imp-continuous-from-below*[*OF f fin*] *cont*
by *blast*

3.4 Properties of *emeasure*

lemma *emeasure-positive*: *positive* (*sets M*) (*emeasure M*)
by (*cases M*) (*auto simp*: *sets-def emeasure-def Abs-measure-inverse measure-space-def*)

lemma *emeasure-empty*[*simp*, *intro*]: *emeasure M* $\{\} = 0$
using *emeasure-positive*[*of M*] **by** (*simp add*: *positive-def*)

lemma *emeasure-single-in-space*: *emeasure M* $\{x\} \neq 0 \Longrightarrow x \in$ *space M*
using *emeasure-notin-sets*[*of* $\{x\}$ *M*] **by** (*auto dest*: *sets.sets-into-space zero-less-iff-neq-zero*[*THEN iffD2*])

lemma *emeasure-countably-additive*: *countably-additive* (*sets M*) (*emeasure M*)
by (*cases M*) (*auto simp*: *sets-def emeasure-def Abs-measure-inverse measure-space-def*)

lemma *suminf-emeasure*:
range A $\subseteq$ *sets M* $\Longrightarrow$ *disjoint-family A* $\Longrightarrow$ $(\sum i.$ *emeasure M* $(A\ i)) =$ *emeasure*

$M\ (\bigcup i.\ A\ i)$
using *sets.countable-UN*[*of A UNIV M*] *emeasure-countably-additive*[*of M*]
by (*simp add*: *countably-additive-def*)

lemma *sums-emeasure*:
disjoint-family F $\Longrightarrow$ $(\bigwedge i.\ F\ i \in sets\ M) \Longrightarrow$ $(\lambda i.\ emeasure\ M\ (F\ i))$ *sums emeasure M* $(\bigcup i.\ F\ i)$
unfolding *sums-iff* **by** (*intro conjI suminf-emeasure*) *auto*

lemma *emeasure-additive*: *additive* (*sets M*) (*emeasure M*)
by (*metis sets.countably-additive-additive emeasure-positive emeasure-countably-additive*)

lemma *plus-emeasure*:
$a \in sets\ M \Longrightarrow b \in sets\ M \Longrightarrow a \cap b = \{\} \Longrightarrow emeasure\ M\ a + emeasure\ M\ b = emeasure\ M\ (a \cup b)$
using *additiveD*[*OF emeasure-additive*] **..**

lemma *setsum-emeasure*:
$F`I \subseteq sets\ M \Longrightarrow$ *disjoint-family-on F I* $\Longrightarrow$ *finite I* $\Longrightarrow$
$(\sum i \in I.\ emeasure\ M\ (F\ i)) = emeasure\ M\ (\bigcup i \in I.\ F\ i)$
by (*metis sets.additive-sum emeasure-positive emeasure-additive*)

lemma *emeasure-mono*:
$a \subseteq b \Longrightarrow b \in sets\ M \Longrightarrow emeasure\ M\ a \leq emeasure\ M\ b$
by (*metis zero-le sets.additive-increasing emeasure-additive emeasure-notin-sets emeasure-positive increasingD*)

lemma *emeasure-space*:
$emeasure\ M\ A \leq emeasure\ M\ (space\ M)$
by (*metis emeasure-mono emeasure-notin-sets sets.sets-into-space sets.top zero-le*)

lemma *emeasure-Diff*:
assumes *finite*: $emeasure\ M\ B \neq \infty$
and [*measurable*]: $A \in sets\ M\ B \in sets\ M$ **and** $B \subseteq A$
shows $emeasure\ M\ (A - B) = emeasure\ M\ A - emeasure\ M\ B$
proof −
have $(A - B) \cup B = A$ **using** ‹$B \subseteq A$› **by** *auto*
then have $emeasure\ M\ A = emeasure\ M\ ((A - B) \cup B)$ **by** *simp*
also have $\ldots = emeasure\ M\ (A - B) + emeasure\ M\ B$
by (*subst plus-emeasure*[*symmetric*]) *auto*
finally show $emeasure\ M\ (A - B) = emeasure\ M\ A - emeasure\ M\ B$
using *finite* **by** *simp*
qed

lemma *emeasure-compl*:
$s \in sets\ M \Longrightarrow emeasure\ M\ s \neq \infty \Longrightarrow emeasure\ M\ (space\ M - s) = emeasure\ M\ (space\ M) - emeasure\ M\ s$
by (*rule emeasure-Diff*) (*auto dest*: *sets.sets-into-space*)

lemma *Lim-emeasure-incseq*:
range A ⊆ sets M ⟹ incseq A ⟹ (λi. (emeasure M (A i))) ⟶ emeasure M (⋃ i. A i)
using *emeasure-countably-additive*
by (*auto simp add*: *sets.countably-additive-iff-continuous-from-below emeasure-positive emeasure-additive*)

lemma *incseq-emeasure*:
assumes *range B ⊆ sets M incseq B*
shows *incseq (λi. emeasure M (B i))*
using *assms* **by** (*auto simp*: *incseq-def intro*!: *emeasure-mono*)

lemma *SUP-emeasure-incseq*:
assumes *A*: *range A ⊆ sets M incseq A*
shows *(SUP n. emeasure M (A n)) = emeasure M (⋃ i. A i)*
using *LIMSEQ-SUP*[*OF incseq-emeasure*, *OF A*] *Lim-emeasure-incseq*[*OF A*]
by (*simp add*: *LIMSEQ-unique*)

lemma *decseq-emeasure*:
assumes *range B ⊆ sets M decseq B*
shows *decseq (λi. emeasure M (B i))*
using *assms* **by** (*auto simp*: *decseq-def intro*!: *emeasure-mono*)

lemma *INF-emeasure-decseq*:
assumes *A*: *range A ⊆ sets M* **and** *decseq A*
and *finite*: *⋀i. emeasure M (A i) ≠ ∞*
shows *(INF n. emeasure M (A n)) = emeasure M (⋂ i. A i)*
proof −
have *le-MI*: *emeasure M (⋂ i. A i) ≤ emeasure M (A 0)*
using *A* **by** (*auto intro*!: *emeasure-mono*)
hence ∗: *emeasure M (⋂ i. A i) ≠ ∞* **using** *finite*[*of 0*] **by** (*auto simp*: *top-unique*)

have *emeasure M (A 0) − (INF n. emeasure M (A n)) = (SUP n. emeasure M (A 0) − emeasure M (A n))*
by (*simp add*: *ennreal-INF-const-minus*)
also have *... = (SUP n. emeasure M (A 0 − A n))*
using *A finite* ‹*decseq A*›[*unfolded decseq-def*] **by** (*subst emeasure-Diff*) *auto*
also have *... = emeasure M (⋃ i. A 0 − A i)*
proof (*rule SUP-emeasure-incseq*)
show *range (λn. A 0 − A n) ⊆ sets M*
using *A* **by** *auto*
show *incseq (λn. A 0 − A n)*
using ‹*decseq A*› **by** (*auto simp add*: *incseq-def decseq-def*)
qed
also have *... = emeasure M (A 0) − emeasure M (⋂ i. A i)*
using *A finite* ∗ **by** (*simp*, *subst emeasure-Diff*) *auto*
finally show *?thesis*
by (*rule ennreal-minus-cancel*[*rotated 3*])
(*insert finite A*, *auto intro*: *INF-lower emeasure-mono*)

qed

lemma *emeasure-INT-decseq-subset*:
fixes $F :: nat \Rightarrow 'a\ set$
assumes I: $I \neq \{\}$ **and** F: $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow i \leq j \Longrightarrow F\ j \subseteq F\ i$
assumes F-$sets[measurable]$: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in sets\ M$
and *fin*: $\bigwedge i.\ i \in I \Longrightarrow emeasure\ M\ (F\ i) \neq \infty$
shows $emeasure\ M\ (\bigcap i{\in}I.\ F\ i) = (INF\ i{:}I.\ emeasure\ M\ (F\ i))$
proof *cases*
assume *finite I*
have $(\bigcap i{\in}I.\ F\ i) = F\ (Max\ I)$
using I ‹*finite I*› **by** (*intro antisym INF-lower INF-greatest F*) *auto*
moreover have $(INF\ i{:}I.\ emeasure\ M\ (F\ i)) = emeasure\ M\ (F\ (Max\ I))$
using I ‹*finite I*› **by** (*intro antisym INF-lower INF-greatest F emeasure-mono*)
auto
ultimately show *?thesis*
by *simp*
next
assume *infinite I*
def $L \equiv \lambda n.\ LEAST\ i.\ i \in I \wedge i \geq n$
have L: $L\ n \in I \wedge n \leq L\ n$ **for** n
unfolding *L-def*
proof (*rule LeastI-ex*)
show $\exists x.\ x \in I \wedge n \leq x$
using ‹*infinite I*› *finite-subset*[*of I* $\{..< n\}$]
by (*rule-tac ccontr*) (*auto simp*: *not-le*)
qed
have L-$eq[simp]$: $i \in I \Longrightarrow L\ i = i$ **for** i
unfolding *L-def* **by** (*intro Least-equality*) *auto*
have *L-mono*: $i \leq j \Longrightarrow L\ i \leq L\ j$ **for** $i\ j$
using $L[of\ j]$ **unfolding** *L-def* **by** (*intro Least-le*) (*auto simp*: *L-def*)

have $emeasure\ M\ (\bigcap i.\ F\ (L\ i)) = (INF\ i.\ emeasure\ M\ (F\ (L\ i)))$
proof (*intro INF-emeasure-decseq*[*symmetric*])
show $decseq\ (\lambda i.\ F\ (L\ i))$
using *L* **by** (*intro antimonoI F L-mono*) *auto*
qed (*insert L fin*, *auto*)
also have $\ldots = (INF\ i{:}I.\ emeasure\ M\ (F\ i))$
proof (*intro antisym INF-greatest*)
show $i \in I \Longrightarrow (INF\ i.\ emeasure\ M\ (F\ (L\ i))) \leq emeasure\ M\ (F\ i)$ **for** i
by (*intro INF-lower2*[*of i*]) *auto*
qed (*insert L*, *auto intro*: *INF-lower*)
also have $(\bigcap i.\ F\ (L\ i)) = (\bigcap i{\in}I.\ F\ i)$
proof (*intro antisym INF-greatest*)
show $i \in I \Longrightarrow (\bigcap i.\ F\ (L\ i)) \subseteq F\ i$ **for** i
by (*intro INF-lower2*[*of i*]) *auto*
qed (*insert L*, *auto*)
finally show *?thesis* **.**
qed

lemma *Lim-emeasure-decseq*:
assumes *A*: *range A* ⊆ *sets M decseq A* **and** *fin*: ⋀*i*. *emeasure M* (*A i*) ≠ ∞
shows (λ*i*. *emeasure M* (*A i*)) ⟶ *emeasure M* (⋂ *i*. *A i*)
using *LIMSEQ-INF*[*OF decseq-emeasure*, *OF A*]
using *INF-emeasure-decseq*[*OF A fin*] **by** *simp*

lemma *emeasure-lfp'*[*consumes 1*, *case-names cont measurable*]:
assumes *P M*
assumes *cont*: *sup-continuous F*
assumes ∗: ⋀*M A*. *P M* ⟹ (⋀*N*. *P N* ⟹ *Measurable.pred N A*) ⟹ *Measurable.pred M* (*F A*)
shows *emeasure M* {*x*∈*space M*. *lfp F x*} = (*SUP i*. *emeasure M* {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*})
proof −
have *emeasure M* {*x*∈*space M*. *lfp F x*} = *emeasure M* (⋃ *i*. {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*})
using *sup-continuous-lfp*[*OF cont*] **by** (*auto simp add*: *bot-fun-def intro*!: *arg-cong2*[**where** *f*=*emeasure*])
moreover { fix *i* **from** ‹*P M*› **have** {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*} ∈ *sets M*
by (*induct i arbitrary*: *M*) (*auto simp add*: *pred-def*[*symmetric*] *intro*: ∗) **}**
moreover have *incseq* (λ*i*. {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*})
proof (*rule incseq-SucI*)
fix *i*
have (*F* ^^ *i*) (λ*x*. *False*) ≤ (*F* ^^ (*Suc i*)) (λ*x*. *False*)
proof (*induct i*)
case *0* **show** *?case* **by** (*simp add*: *le-fun-def*)
next
case *Suc* **thus** *?case* **using** *monoD*[*OF sup-continuous-mono*[*OF cont*] *Suc*] **by** *auto*
qed
then show {*x* ∈ *space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*} ⊆ {*x* ∈ *space M*. (*F* ^^ *Suc i*) (λ*x*. *False*) *x*}
by *auto*
qed
ultimately show *?thesis*
by (*subst SUP-emeasure-incseq*) *auto*
qed

lemma *emeasure-lfp*:
assumes [*simp*]: ⋀*s*. *sets* (*M s*) = *sets N*
assumes *cont*: *sup-continuous F sup-continuous f*
assumes *meas*: ⋀*P*. *Measurable.pred N P* ⟹ *Measurable.pred N* (*F P*)
assumes *iter*: ⋀*P s*. *Measurable.pred N P* ⟹ *P* ≤ *lfp F* ⟹ *emeasure* (*M s*) {*x*∈*space N*. *F P x*} = *f* (λ*s*. *emeasure* (*M s*) {*x*∈*space N*. *P x*}) *s*
shows *emeasure* (*M s*) {*x*∈*space N*. *lfp F x*} = *lfp f s*
proof (*subst lfp-transfer-bounded*[**where** α=λ*F s*. *emeasure* (*M s*) {*x*∈*space N*. *F x*} **and** *g*=*f* **and** *f*=*F* **and** *P*=*Measurable.pred N*, *symmetric*])

fix C **assume** *incseq* C $\bigwedge i$. *Measurable.pred* N $(C\ i)$
then show $(\lambda s.$ *emeasure* $(M\ s)$ $\{x \in$ *space* $N.$ $(SUP\ i.\ C\ i)\ x\}) = (SUP\ i.\ (\lambda s.$ *emeasure* $(M\ s)$ $\{x \in$ *space* $N.\ C\ i\ x\}))$
unfolding *SUP-apply*[*abs-def*]
by (*subst SUP-emeasure-incseq*) (*auto simp*: *mono-def fun-eq-iff intro*!: *arg-cong2*[**where** *f*=*emeasure*])
qed (*auto simp add*: *iter le-fun-def SUP-apply*[*abs-def*] *intro*!: *meas cont*)

lemma *emeasure-subadditive-finite*:
finite $I \Longrightarrow A$ ' $I \subseteq$ *sets* $M \Longrightarrow$ *emeasure* M $(\bigcup i{\in}I.\ A\ i) \leq (\sum i{\in}I.$ *emeasure* M $(A\ i))$
by (*rule sets.subadditive*[*OF emeasure-positive emeasure-additive*]) *auto*

lemma *emeasure-subadditive*:
$A \in$ *sets* $M \Longrightarrow B \in$ *sets* $M \Longrightarrow$ *emeasure* M $(A \cup B) \leq$ *emeasure* M $A +$ *emeasure* M B
using *emeasure-subadditive-finite*[*of* $\{True, False\}$ $\lambda True \Rightarrow A \mid False \Rightarrow B\ M$]
by *simp*

lemma *emeasure-subadditive-countably*:
assumes *range* $f \subseteq$ *sets* M
shows *emeasure* M $(\bigcup i.\ f\ i) \leq (\sum i.$ *emeasure* M $(f\ i))$
proof −
have *emeasure* M $(\bigcup i.\ f\ i) =$ *emeasure* M $(\bigcup i.$ *disjointed* $f\ i)$
unfolding *UN-disjointed-eq* **..**
also have $\ldots = (\sum i.$ *emeasure* M $($*disjointed* $f\ i))$
using *sets.range-disjointed-sets*[*OF assms*] *suminf-emeasure*[*of disjointed f*]
by (*simp add*: *disjoint-family-disjointed comp-def*)
also have $\ldots \leq (\sum i.$ *emeasure* M $(f\ i))$
using *sets.range-disjointed-sets*[*OF assms*] *assms*
by (*auto intro*!: *suminf-le emeasure-mono disjointed-subset*)
finally show *?thesis* **.**
qed

lemma *emeasure-insert*:
assumes *sets*: $\{x\} \in$ *sets* M $A \in$ *sets* M **and** $x \notin A$
shows *emeasure* M $($*insert* $x\ A) =$ *emeasure* M $\{x\} +$ *emeasure* M A
proof −
have $\{x\} \cap A = \{\}$ **using** ‹$x \notin A$› **by** *auto*
from *plus-emeasure*[*OF sets this*] **show** *?thesis* **by** *simp*
qed

lemma *emeasure-insert-ne*:
$A \neq \{\} \Longrightarrow \{x\} \in$ *sets* $M \Longrightarrow A \in$ *sets* $M \Longrightarrow x \notin A \Longrightarrow$ *emeasure* M $($*insert* $x\ A) =$ *emeasure* M $\{x\} +$ *emeasure* M A
by (*rule emeasure-insert*)

lemma *emeasure-eq-setsum-singleton*:
assumes *finite* S $\bigwedge x.\ x \in S \Longrightarrow \{x\} \in$ *sets* M

shows *emeasure M S* = $(\sum x{\in}S.$ *emeasure M* $\{x\})$
using *setsum-emeasure*[*of* $\lambda x.$ $\{x\}$ *S M*] *assms*
by (*auto simp*: *disjoint-family-on-def subset-eq*)

lemma *setsum-emeasure-cover*:
assumes *finite S* **and** *A* ∈ *sets M* **and** *br-in-M*: *B* ' *S* ⊆ *sets M*
assumes *A*: *A* ⊆ $(\bigcup i{\in}S.$ *B i*)
assumes *disj*: *disjoint-family-on B S*
shows *emeasure M A* = $(\sum i{\in}S.$ *emeasure M* $(A \cap (B\ i)))$
proof −
have $(\sum i{\in}S.$ *emeasure M* $(A \cap (B\ i)))$ = *emeasure M* $(\bigcup i{\in}S.\ A \cap (B\ i))$
proof (*rule setsum-emeasure*)
show *disjoint-family-on* $(\lambda i.\ A \cap B\ i)$ *S*
using ‹*disjoint-family-on B S*›
unfolding *disjoint-family-on-def* **by** *auto*
qed (*insert assms*, *auto*)
also have $(\bigcup i{\in}S.\ A \cap (B\ i))$ = *A*
using *A* **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *emeasure-eq-0*:
N ∈ *sets M* ⟹ *emeasure M N* = *0* ⟹ *K* ⊆ *N* ⟹ *emeasure M K* = *0*
by (*metis emeasure-mono order-eq-iff zero-le*)

lemma *emeasure-UN-eq-0*:
assumes $\bigwedge i{::}nat.$ *emeasure M* $(N\ i)$ = *0* **and** *range N* ⊆ *sets M*
shows *emeasure M* $(\bigcup i.\ N\ i)$ = *0*
proof −
have *emeasure M* $(\bigcup i.\ N\ i) \leq 0$
using *emeasure-subadditive-countably*[*OF assms*(*2*)] *assms*(*1*) **by** *simp*
then show *?thesis*
by (*auto intro*: *antisym zero-le*)
qed

lemma *measure-eqI-finite*:
assumes [*simp*]: *sets M* = *Pow A sets N* = *Pow A* **and** *finite A*
assumes *eq*: $\bigwedge a.$ *a* ∈ *A* ⟹ *emeasure M* $\{a\}$ = *emeasure N* $\{a\}$
shows *M* = *N*
proof (*rule measure-eqI*)
fix *X* **assume** *X* ∈ *sets M*
then have *X*: *X* ⊆ *A* **by** *auto*
then have *emeasure M X* = $(\sum a{\in}X.$ *emeasure M* $\{a\})$
using ‹*finite A*› **by** (*subst emeasure-eq-setsum-singleton*) (*auto dest*: *finite-subset*)
also have ... = $(\sum a{\in}X.$ *emeasure N* $\{a\})$
using *X eq* **by** (*auto intro*!: *setsum.cong*)
also have ... = *emeasure N X*
using *X* ‹*finite A*› **by** (*subst emeasure-eq-setsum-singleton*) (*auto dest*: *finite-subset*)
finally show *emeasure M X* = *emeasure N X* **.**

qed *simp*

lemma *measure-eqI-generator-eq*:
fixes $M\ N :: 'a\ measure$ **and** $E :: 'a\ set\ set$ **and** $A :: nat \Rightarrow 'a\ set$
assumes *Int-stable E* $E \subseteq Pow\ \Omega$
and *eq*: $\bigwedge X.\ X \in E \Longrightarrow emeasure\ M\ X = emeasure\ N\ X$
and *M*: *sets M* $= sigma\text{-}sets\ \Omega\ E$
and *N*: *sets N* $= sigma\text{-}sets\ \Omega\ E$
and *A*: *range A* $\subseteq E$ $(\bigcup i.\ A\ i) = \Omega$ $\bigwedge i.\ emeasure\ M\ (A\ i) \neq \infty$
shows $M = N$
proof −
let $?\mu = emeasure\ M$ **and** $?\nu = emeasure\ N$
interpret *S*: *sigma-algebra* Ω *sigma-sets* $\Omega\ E$ **by** (*rule sigma-algebra-sigma-sets*) *fact*
have *space* $M = \Omega$
using *sets.top*[*of M*] *sets.space-closed*[*of M*] *S.top S.space-closed* ‹*sets M* = *sigma-sets* Ω *E*›
by *blast*

{ fix *F D* **assume** $F \in E$ **and** $?\mu\ F \neq \infty$
then have [*intro*]: $F \in sigma\text{-}sets\ \Omega\ E$ **by** *auto*
have $?\nu\ F \neq \infty$ **using** ‹$?\mu\ F \neq \infty$› ‹$F \in E$› *eq* **by** *simp*
assume $D \in sets\ M$
with ‹*Int-stable E*› ‹$E \subseteq Pow\ \Omega$› **have** $emeasure\ M\ (F \cap D) = emeasure\ N\ (F \cap D)$
unfolding *M*
proof (*induct rule*: *sigma-sets-induct-disjoint*)
case (*basic A*)
then have $F \cap A \in E$ **using** ‹*Int-stable E*› ‹$F \in E$› **by** (*auto simp*: *Int-stable-def*)
then show *?case* **using** *eq* **by** *auto*
next
case *empty* **then show** *?case* **by** *simp*
next
case (*compl A*)
then have $**$: $F \cap (\Omega - A) = F - (F \cap A)$
and [*intro*]: $F \cap A \in sigma\text{-}sets\ \Omega\ E$
using ‹$F \in E$› *S.sets-into-space* **by** (*auto simp*: *M*)
have $?\nu\ (F \cap A) \leq ?\nu\ F$ **by** (*auto intro*!: *emeasure-mono simp*: *M N*)
then have $?\nu\ (F \cap A) \neq \infty$ **using** ‹$?\nu\ F \neq \infty$› **by** (*auto simp*: *top-unique*)
have $?\mu\ (F \cap A) \leq ?\mu\ F$ **by** (*auto intro*!: *emeasure-mono simp*: *M N*)
then have $?\mu\ (F \cap A) \neq \infty$ **using** ‹$?\mu\ F \neq \infty$› **by** (*auto simp*: *top-unique*)
then have $?\mu\ (F \cap (\Omega - A)) = ?\mu\ F - ?\mu\ (F \cap A)$ **unfolding** $**$
using ‹$F \cap A \in sigma\text{-}sets\ \Omega\ E$› **by** (*auto intro*!: *emeasure-Diff simp*: *M N*)
also have $\ldots = ?\nu\ F - ?\nu\ (F \cap A)$ **using** *eq* ‹$F \in E$› *compl* **by** *simp*
also have $\ldots = ?\nu\ (F \cap (\Omega - A))$ **unfolding** $**$
using ‹$F \cap A \in sigma\text{-}sets\ \Omega\ E$› ‹$?\nu\ (F \cap A) \neq \infty$›
by (*auto intro*!: *emeasure-Diff*[*symmetric*] *simp*: *M N*)
finally show *?case*

```
      using ‹space M = Ω› by auto
    next
      case (union A)
      then have ?μ (⋃x. F ∩ A x) = ?ν (⋃x. F ∩ A x)
      by (subst (1 2) suminf-emeasure[symmetric]) (auto simp: disjoint-family-on-def
subset-eq M N)
      with A show ?case
        by auto
    qed }
  note ∗ = this
  show M = N
  proof (rule measure-eqI)
    show sets M = sets N
      using M N by simp
    have [simp, intro]: ⋀i. A i ∈ sets M
      using A(1) by (auto simp: subset-eq M)
    fix F assume F ∈ sets M
    let ?D = disjointed (λi. F ∩ A i)
    from ‹space M = Ω› have F-eq: F = (⋃i. ?D i)
      using ‹F ∈ sets M›[THEN sets.sets-into-space] A(2)[symmetric] by (auto
simp: UN-disjointed-eq)
    have [simp, intro]: ⋀i. ?D i ∈ sets M
      using sets.range-disjointed-sets[of λi. F ∩ A i M] ‹F ∈ sets M›
      by (auto simp: subset-eq)
    have disjoint-family ?D
      by (auto simp: disjoint-family-disjointed)
    moreover
    have (∑i. emeasure M (?D i)) = (∑i. emeasure N (?D i))
    proof (intro arg-cong[where f=suminf] ext)
      fix i
      have A i ∩ ?D i = ?D i
        by (auto simp: disjointed-def)
      then show emeasure M (?D i) = emeasure N (?D i)
        using ∗[of A i ?D i, OF - A(3)] A(1) by auto
    qed
    ultimately show emeasure M F = emeasure N F
     by (simp add: image-subset-iff ‹sets M = sets N›[symmetric] F-eq[symmetric]
suminf-emeasure)
  qed
qed

lemma measure-of-of-measure: measure-of (space M) (sets M) (emeasure M) =
M
proof (intro measure-eqI emeasure-measure-of-sigma)
  show sigma-algebra (space M) (sets M) ..
  show positive (sets M) (emeasure M)
    by (simp add: positive-def)
  show countably-additive (sets M) (emeasure M)
    by (simp add: emeasure-countably-additive)
```

qed *simp-all*

3.5 μ-null sets

definition *null-sets* :: *'a measure* ⇒ *'a set set* **where**
null-sets M = {*N*∈*sets M*. *emeasure M N* = *0*}

lemma *null-setsD1*[*dest*]: *A* ∈ *null-sets M* ⟹ *emeasure M A* = *0*
by (*simp add*: *null-sets-def*)

lemma *null-setsD2*[*dest*]: *A* ∈ *null-sets M* ⟹ *A* ∈ *sets M*
unfolding *null-sets-def* **by** *simp*

lemma *null-setsI*[*intro*]: *emeasure M A* = *0* ⟹ *A* ∈ *sets M* ⟹ *A* ∈ *null-sets M*
unfolding *null-sets-def* **by** *simp*

interpretation *null-sets*: *ring-of-sets space M null-sets M* **for** *M*
proof (*rule ring-of-setsI*)
show *null-sets M* ⊆ *Pow* (*space M*)
using *sets.sets-into-space* **by** *auto*
show {} ∈ *null-sets M*
by *auto*
fix *A B* **assume** *null-sets*: *A* ∈ *null-sets M B* ∈ *null-sets M*
then have *sets*: *A* ∈ *sets M B* ∈ *sets M*
by *auto*
then have ∗: *emeasure M* (*A* ∪ *B*) ≤ *emeasure M A* + *emeasure M B*
emeasure M (*A* − *B*) ≤ *emeasure M A*
by (*auto intro*!: *emeasure-subadditive emeasure-mono*)
then have *emeasure M B* = *0 emeasure M A* = *0*
using *null-sets* **by** *auto*
with *sets* ∗ **show** *A* − *B* ∈ *null-sets M A* ∪ *B* ∈ *null-sets M*
by (*auto intro*!: *antisym zero-le*)
qed

lemma *UN-from-nat-into*:
assumes *I*: *countable I I* ≠ {}
shows (⋃*i*∈*I*. *N i*) = (⋃*i*. *N* (*from-nat-into I i*))
proof −
have (⋃*i*∈*I*. *N i*) = ⋃(*N* ‘ *range* (*from-nat-into I*))
using *I* **by** *simp*
also have ... = (⋃*i*. (*N* ∘ *from-nat-into I*) *i*)
by *simp*
finally show *?thesis* **by** *simp*
qed

lemma *null-sets-UN'*:
assumes *countable I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *N i* ∈ *null-sets M*
shows (⋃*i*∈*I*. *N i*) ∈ *null-sets M*

proof *cases*
assume $I = \{\}$ **then show** *?thesis* **by** *simp*
next
assume $I \neq \{\}$
show *?thesis*
proof (*intro conjI CollectI null-setsI*)
show $(\bigcup i{\in}I.\ N\ i) \in$ *sets M*
using *assms* **by** (*intro sets.countable-UN'*) *auto*
have *emeasure M* $(\bigcup i{\in}I.\ N\ i) \leq (\sum n.$ *emeasure M* $(N$ (*from-nat-into I n*)))
unfolding *UN-from-nat-into*[*OF* ‹*countable I*› ‹$I \neq \{\}$›]
using *assms* ‹$I \neq \{\}$› **by** (*intro emeasure-subadditive-countably*) (*auto intro*: *from-nat-into*)
also have $(\lambda n.$ *emeasure M* $(N$ (*from-nat-into I n*))) $= (\lambda$-. *0*)
using *assms* ‹$I \neq \{\}$› **by** (*auto intro*: *from-nat-into*)
finally show *emeasure M* $(\bigcup i{\in}I.\ N\ i) =$ *0*
by (*intro antisym zero-le*) *simp*
qed
qed

lemma *null-sets-UN*[*intro*]:
$(\bigwedge i::'i::$*countable*$.\ N\ i \in$ *null-sets M*$) \Longrightarrow (\bigcup i.\ N\ i) \in$ *null-sets M*
by (*rule null-sets-UN'*) *auto*

lemma *null-set-Int1*:
assumes $B \in$ *null-sets M* $A \in$ *sets M* **shows** $A \cap B \in$ *null-sets M*
proof (*intro CollectI conjI null-setsI*)
show *emeasure M* $(A \cap B) =$ *0* **using** *assms*
by (*intro emeasure-eq-0*[*of B - A* $\cap$ *B*]) *auto*
qed (*insert assms*, *auto*)

lemma *null-set-Int2*:
assumes $B \in$ *null-sets M* $A \in$ *sets M* **shows** $B \cap A \in$ *null-sets M*
using *assms* **by** (*subst Int-commute*) (*rule null-set-Int1*)

lemma *emeasure-Diff-null-set*:
assumes $B \in$ *null-sets M* $A \in$ *sets M*
shows *emeasure M* $(A - B) =$ *emeasure M A*
proof −
have $*$: $A - B = (A - (A \cap B))$ **by** *auto*
have $A \cap B \in$ *null-sets M* **using** *assms* **by** (*rule null-set-Int1*)
then show *?thesis*
unfolding $*$ **using** *assms*
by (*subst emeasure-Diff*) *auto*
qed

lemma *null-set-Diff*:
assumes $B \in$ *null-sets M* $A \in$ *sets M* **shows** $B - A \in$ *null-sets M*
proof (*intro CollectI conjI null-setsI*)
show *emeasure M* $(B - A) =$ *0* **using** *assms* **by** (*intro emeasure-eq-0*[*of B - B*

$- A]$) *auto*
qed (*insert assms*, *auto*)

lemma *emeasure-Un-null-set*:
assumes $A \in$ *sets* M $B \in$ *null-sets* M
shows *emeasure* M $(A \cup B)$ = *emeasure* M A
proof −
have $*$: $A \cup B = A \cup (B - A)$ **by** *auto*
have $B - A \in$ *null-sets* M **using** *assms*(*2*,*1*) **by** (*rule null-set-Diff*)
then show *?thesis*
unfolding $*$ **using** *assms*
by (*subst plus-emeasure*[*symmetric*]) *auto*
qed

3.6 The almost everywhere filter (i.e. quantifier)

definition *ae-filter* :: *'a measure* ⇒ *'a filter* **where**
ae-filter M = (*INF* N:*null-sets* M. *principal* (*space* M − N))

abbreviation *almost-everywhere* :: *'a measure* ⇒ (*'a* ⇒ *bool*) ⇒ *bool* **where**
almost-everywhere M $P \equiv$ *eventually* P (*ae-filter* M)

syntax
-almost-everywhere :: *pttrn* ⇒ *'a* ⇒ *bool* ⇒ *bool* (*AE - in -. - [0,0,10] 10*)

translations
AE x *in* M. $P \rightleftharpoons$ *CONST almost-everywhere* M (λx. P)

lemma *eventually-ae-filter*: *eventually* P (*ae-filter* M) ⟷ (∃ N∈*null-sets* M. {x ∈ *space* M. ¬ P x} ⊆ N)
unfolding *ae-filter-def* **by** (*subst eventually-INF-base*) (*auto simp*: *eventually-principal subset-eq*)

lemma *AE-I'*:
$N \in$ *null-sets* M ⟹ {x∈*space* M. ¬ P x} ⊆ N ⟹ (*AE* x *in* M. P x)
unfolding *eventually-ae-filter* **by** *auto*

lemma *AE-iff-null*:
assumes {x∈*space* M. ¬ P x} ∈ *sets* M (**is** *?P* ∈ *sets* M)
shows (*AE* x *in* M. P x) ⟷ {x∈*space* M. ¬ P x} ∈ *null-sets* M
proof
assume *AE* x *in* M. P x **then obtain** N **where** N: $N \in$ *sets* M *?P* ⊆ N *emeasure* M N = *0*
unfolding *eventually-ae-filter* **by** *auto*
have *emeasure* M *?P* ≤ *emeasure* M N
using *assms* N(*1*,*2*) **by** (*auto intro*: *emeasure-mono*)
then have *emeasure* M *?P* = *0*
unfolding ‹*emeasure* M N = *0*› **by** *auto*
then show *?P* ∈ *null-sets* M **using** *assms* **by** *auto*

next
assume *?P* ∈ *null-sets M* **with** *assms* **show** *AE x in M. P x* **by** (*auto intro*: *AE-I'*)
qed

lemma *AE-iff-null-sets*:
N ∈ *sets M* ⟹ *N* ∈ *null-sets M* ⟷ (*AE x in M. x* ∉ *N*)
using *Int-absorb1*[*OF sets.sets-into-space*, *of N M*]
by (*subst AE-iff-null*) (*auto simp*: *Int-def*[*symmetric*])

lemma *AE-not-in*:
N ∈ *null-sets M* ⟹ *AE x in M. x* ∉ *N*
by (*metis AE-iff-null-sets null-setsD2*)

lemma *AE-iff-measurable*:
N ∈ *sets M* ⟹ {*x*∈*space M.* ¬ *P x*} = *N* ⟹ (*AE x in M. P x*) ⟷ *emeasure M N* = *0*
using *AE-iff-null*[*of* - *P*] **by** *auto*

lemma *AE-E*[*consumes 1*]:
assumes *AE x in M. P x*
obtains *N* **where** {*x* ∈ *space M.* ¬ *P x*} ⊆ *N emeasure M N* = *0 N* ∈ *sets M*
using *assms* **unfolding** *eventually-ae-filter* **by** *auto*

lemma *AE-E2*:
assumes *AE x in M. P x* {*x*∈*space M. P x*} ∈ *sets M*
shows *emeasure M* {*x*∈*space M.* ¬ *P x*} = *0* (**is** *emeasure M ?P* = *0*)
proof −
have {*x*∈*space M.* ¬ *P x*} = *space M* − {*x*∈*space M. P x*} **by** *auto*
with *AE-iff-null*[*of M P*] *assms* **show** *?thesis* **by** *auto*
qed

lemma *AE-I*:
assumes {*x* ∈ *space M.* ¬ *P x*} ⊆ *N emeasure M N* = *0 N* ∈ *sets M*
shows *AE x in M. P x*
using *assms* **unfolding** *eventually-ae-filter* **by** *auto*

lemma *AE-mp*[*elim*!]:
assumes *AE-P*: *AE x in M. P x* **and** *AE-imp*: *AE x in M. P x* ⟶ *Q x*
shows *AE x in M. Q x*
proof −
from *AE-P* **obtain** *A* **where** *P*: {*x*∈*space M.* ¬ *P x*} ⊆ *A*
and *A*: *A* ∈ *sets M emeasure M A* = *0*
by (*auto elim*!: *AE-E*)

from *AE-imp* **obtain** *B* **where** *imp*: {*x*∈*space M. P x* ∧ ¬ *Q x*} ⊆ *B*
and *B*: *B* ∈ *sets M emeasure M B* = *0*
by (*auto elim*!: *AE-E*)

show *?thesis*
proof (*intro AE-I*)
have *emeasure M* (*A* ∪ *B*) ≤ *0*
using *emeasure-subadditive*[*of A M B*] *A B* **by** *auto*
then show *A* ∪ *B* ∈ *sets M emeasure M* (*A* ∪ *B*) = *0*
using *A B* **by** *auto*
show {*x*∈*space M*. ¬ *Q x*} ⊆ *A* ∪ *B*
using *P imp* **by** *auto*
qed
qed

lemma
shows *AE-iffI*: *AE x in M*. *P x* ⟹ *AE x in M*. *P x* ⟷ *Q x* ⟹ *AE x in M*. *Q x*
and *AE-disjI1*: *AE x in M*. *P x* ⟹ *AE x in M*. *P x* ∨ *Q x*
and *AE-disjI2*: *AE x in M*. *Q x* ⟹ *AE x in M*. *P x* ∨ *Q x*
and *AE-conjI*: *AE x in M*. *P x* ⟹ *AE x in M*. *Q x* ⟹ *AE x in M*. *P x* ∧ *Q x*
and *AE-conj-iff*[*simp*]: (*AE x in M*. *P x* ∧ *Q x*) ⟷ (*AE x in M*. *P x*) ∧ (*AE x in M*. *Q x*)
by *auto*

lemma *AE-impI*:
(*P* ⟹ *AE x in M*. *Q x*) ⟹ *AE x in M*. *P* ⟶ *Q x*
by (*cases P*) *auto*

lemma *AE-measure*:
assumes *AE*: *AE x in M*. *P x* **and** *sets*: {*x*∈*space M*. *P x*} ∈ *sets M* (**is** *?P* ∈ *sets M*)
shows *emeasure M* {*x*∈*space M*. *P x*} = *emeasure M* (*space M*)
proof −
from *AE-E*[*OF AE*] **guess** *N* **. note** *N* = *this*
with *sets* **have** *emeasure M* (*space M*) ≤ *emeasure M* (*?P* ∪ *N*)
by (*intro emeasure-mono*) *auto*
also have ... ≤ *emeasure M ?P* + *emeasure M N*
using *sets N* **by** (*intro emeasure-subadditive*) *auto*
also have ... = *emeasure M ?P* **using** *N* **by** *simp*
finally show *emeasure M ?P* = *emeasure M* (*space M*)
using *emeasure-space*[*of M ?P*] **by** *auto*
qed

lemma *AE-space*: *AE x in M*. *x* ∈ *space M*
by (*rule AE-I*[**where** *N*={}]) *auto*

lemma *AE-I2*[*simp*, *intro*]:
(⋀*x*. *x* ∈ *space M* ⟹ *P x*) ⟹ *AE x in M*. *P x*
using *AE-space* **by** *force*

lemma *AE-Ball-mp*:
$\forall x{\in}space\ M.\ P\ x \Longrightarrow AE\ x\ in\ M.\ P\ x \longrightarrow Q\ x \Longrightarrow AE\ x\ in\ M.\ Q\ x$
by *auto*

lemma *AE-cong*[*cong*]:
$(\bigwedge x.\ x \in space\ M \Longrightarrow P\ x \longleftrightarrow Q\ x) \Longrightarrow (AE\ x\ in\ M.\ P\ x) \longleftrightarrow (AE\ x\ in\ M.\ Q\ x)$
by *auto*

lemma *AE-all-countable*:
$(AE\ x\ in\ M.\ \forall i.\ P\ i\ x) \longleftrightarrow (\forall i{::}'i{::}countable.\ AE\ x\ in\ M.\ P\ i\ x)$
proof
assume $\forall i.\ AE\ x\ in\ M.\ P\ i\ x$
from *this*[*unfolded eventually-ae-filter Bex-def*, *THEN choice*]
obtain N **where** N: $\bigwedge i.\ N\ i \in null\text{-}sets\ M\ \bigwedge i.\ \{x{\in}space\ M.\ \neg\ P\ i\ x\} \subseteq N\ i$
by *auto*
have $\{x{\in}space\ M.\ \neg\ (\forall i.\ P\ i\ x)\} \subseteq (\bigcup i.\ \{x{\in}space\ M.\ \neg\ P\ i\ x\})$ **by** *auto*
also have $\ldots \subseteq (\bigcup i.\ N\ i)$ **using** N **by** *auto*
finally have $\{x{\in}space\ M.\ \neg\ (\forall i.\ P\ i\ x)\} \subseteq (\bigcup i.\ N\ i)$ **.**
moreover from N **have** $(\bigcup i.\ N\ i) \in null\text{-}sets\ M$
by (*intro null-sets-UN*) *auto*
ultimately show $AE\ x\ in\ M.\ \forall i.\ P\ i\ x$
unfolding *eventually-ae-filter* **by** *auto*
qed *auto*

lemma *AE-ball-countable*:
assumes [*intro*]: *countable* X
shows $(AE\ x\ in\ M.\ \forall y{\in}X.\ P\ x\ y) \longleftrightarrow (\forall y{\in}X.\ AE\ x\ in\ M.\ P\ x\ y)$
proof
assume $\forall y{\in}X.\ AE\ x\ in\ M.\ P\ x\ y$
from *this*[*unfolded eventually-ae-filter Bex-def*, *THEN bchoice*]
obtain N **where** N: $\bigwedge y.\ y \in X \Longrightarrow N\ y \in null\text{-}sets\ M\ \bigwedge y.\ y \in X \Longrightarrow \{x{\in}space\ M.\ \neg\ P\ x\ y\} \subseteq N\ y$
by *auto*
have $\{x{\in}space\ M.\ \neg\ (\forall y{\in}X.\ P\ x\ y)\} \subseteq (\bigcup y{\in}X.\ \{x{\in}space\ M.\ \neg\ P\ x\ y\})$
by *auto*
also have $\ldots \subseteq (\bigcup y{\in}X.\ N\ y)$
using N **by** *auto*
finally have $\{x{\in}space\ M.\ \neg\ (\forall y{\in}X.\ P\ x\ y)\} \subseteq (\bigcup y{\in}X.\ N\ y)$ **.**
moreover from N **have** $(\bigcup y{\in}X.\ N\ y) \in null\text{-}sets\ M$
by (*intro null-sets-UN'*) *auto*
ultimately show $AE\ x\ in\ M.\ \forall y{\in}X.\ P\ x\ y$
unfolding *eventually-ae-filter* **by** *auto*
qed *auto*

lemma *AE-discrete-difference*:
assumes X: *countable* X
assumes *null*: $\bigwedge x.\ x \in X \Longrightarrow emeasure\ M\ \{x\} = 0$
assumes *sets*: $\bigwedge x.\ x \in X \Longrightarrow \{x\} \in sets\ M$

shows *AE x in M. x ∉ X*
proof −
have $(\bigcup x{\in}X.\ \{x\}) \in$ *null-sets M*
using *assms* **by** (*intro null-sets-UN'*) *auto*
from *AE-not-in*[*OF this*] **show** *AE x in M. x ∉ X*
by *auto*
qed

lemma *AE-finite-all*:
assumes *f*: *finite S* **shows** (*AE x in M.* $\forall i{\in}S.$ *P i x*) $\longleftrightarrow$ ($\forall i{\in}S.$ *AE x in M. P i x*)
using *f* **by** *induct auto*

lemma *AE-finite-allI*:
assumes *finite S*
shows ($\bigwedge s.$ *s* $\in$ *S* $\Longrightarrow$ *AE x in M. Q s x*) $\Longrightarrow$ *AE x in M.* $\forall s{\in}S.$ *Q s x*
using *AE-finite-all*[*OF* ‹*finite S*›] **by** *auto*

lemma *emeasure-mono-AE*:
assumes *imp*: *AE x in M. x* $\in$ *A* $\longrightarrow$ *x* $\in$ *B*
and *B*: *B* $\in$ *sets M*
shows *emeasure M A* $\leq$ *emeasure M B*
proof *cases*
assume *A*: *A* $\in$ *sets M*
from *imp* **obtain** *N* **where** *N*: {*x*∈*space M.* ¬ (*x* $\in$ *A* $\longrightarrow$ *x* $\in$ *B*)} $\subseteq$ *N N* $\in$ *null-sets M*
by (*auto simp*: *eventually-ae-filter*)
have *emeasure M A* = *emeasure M* (*A* − *N*)
using *N A* **by** (*subst emeasure-Diff-null-set*) *auto*
also have *emeasure M* (*A* − *N*) $\leq$ *emeasure M* (*B* − *N*)
using *N A B sets.sets-into-space* **by** (*auto intro*!: *emeasure-mono*)
also have *emeasure M* (*B* − *N*) = *emeasure M B*
using *N B* **by** (*subst emeasure-Diff-null-set*) *auto*
finally show *?thesis* **.**
qed (*simp add*: *emeasure-notin-sets*)

lemma *emeasure-eq-AE*:
assumes *iff*: *AE x in M. x* $\in$ *A* $\longleftrightarrow$ *x* $\in$ *B*
assumes *A*: *A* $\in$ *sets M* **and** *B*: *B* $\in$ *sets M*
shows *emeasure M A = emeasure M B*
using *assms* **by** (*safe intro*!: *antisym emeasure-mono-AE*) *auto*

lemma *emeasure-Collect-eq-AE*:
AE x in M. P x $\longleftrightarrow$ *Q x* $\Longrightarrow$ *Measurable.pred M Q* $\Longrightarrow$ *Measurable.pred M P* $\Longrightarrow$
emeasure M {*x*∈*space M. P x*} = *emeasure M* {*x*∈*space M. Q x*}
by (*intro emeasure-eq-AE*) *auto*

lemma *emeasure-eq-0-AE*: *AE x in M.* ¬ *P x* $\Longrightarrow$ *emeasure M* {*x*∈*space M. P*

$x\} = 0$
using *AE-iff-measurable*[*OF - refl, of M* $\lambda x.\ \neg\ P\ x$]
by (*cases* $\{x{\in}space\ M.\ P\ x\} \in sets\ M$) (*simp-all add*: *emeasure-notin-sets*)

lemma *emeasure-add-AE*:
assumes [*measurable*]: $A \in sets\ M\ B \in sets\ M\ C \in sets\ M$
assumes *1*: *AE x in M*. $x \in C \longleftrightarrow x \in A \vee x \in B$
assumes *2*: *AE x in M*. $\neg\ (x \in A \wedge x \in B)$
shows *emeasure M C* = *emeasure M A* + *emeasure M B*
proof −
have *emeasure M C* = *emeasure M* $(A \cup B)$
by (*rule emeasure-eq-AE*) (*insert 1*, *auto*)
also have ... = *emeasure M A* + *emeasure M* $(B - A)$
by (*subst plus-emeasure*) *auto*
also have *emeasure M* $(B - A)$ = *emeasure M B*
by (*rule emeasure-eq-AE*) (*insert 2*, *auto*)
finally show *?thesis* **.**
qed

3.7 σ-finite Measures

locale *sigma-finite-measure* =
fixes *M* :: *'a measure*
assumes *sigma-finite-countable*:
$\exists\ A{::}'a\ set\ set.\ countable\ A \wedge A \subseteq sets\ M \wedge (\bigcup A) = space\ M \wedge (\forall\ a{\in}A.$ *emeasure M a* $\neq \infty$)

lemma (**in** *sigma-finite-measure*) *sigma-finite*:
obtains *A* :: *nat* $\Rightarrow$ *'a set*
where *range A* $\subseteq$ *sets M* $(\bigcup i.\ A\ i)$ = *space M* $\bigwedge i.$ *emeasure M* $(A\ i) \neq \infty$
proof −
obtain *A* :: *'a set set* **where**
[*simp*]: *countable A* **and**
A: $A \subseteq sets\ M\ (\bigcup A) = space\ M\ \bigwedge a.\ a \in A \Longrightarrow$ *emeasure M a* $\neq \infty$
using *sigma-finite-countable* **by** *metis*
show *thesis*
proof *cases*
assume $A = \{\}$ **with** ‹$(\bigcup A) = space\ M$› **show** *thesis*
by (*intro that*[*of* λ-. $\{\}$]) *auto*
next
assume $A \neq \{\}$
show *thesis*
proof
show *range* (*from-nat-into A*) $\subseteq$ *sets M*
using ‹$A \neq \{\}$› *A* **by** *auto*
have $(\bigcup i.$ *from-nat-into A i*$) = \bigcup A$
using *range-from-nat-into*[*OF* ‹$A \neq \{\}$› ‹*countable A*›] **by** *auto*
with *A* **show** $(\bigcup i.$ *from-nat-into A i*$) =$ *space M*
by *auto*

```
    qed (intro A from-nat-into ‹A ≠ {}›)
  qed
qed

lemma (in sigma-finite-measure) sigma-finite-disjoint:
  obtains A :: nat ⇒ 'a set
  where range A ⊆ sets M (⋃i. A i) = space M ⋀i. emeasure M (A i) ≠ ∞
disjoint-family A
proof −
  obtain A :: nat ⇒ 'a set where
    range: range A ⊆ sets M and
    space: (⋃i. A i) = space M and
    measure: ⋀i. emeasure M (A i) ≠ ∞
    using sigma-finite by blast
  show thesis
  proof (rule that[of disjointed A])
    show range (disjointed A) ⊆ sets M
      by (rule sets.range-disjointed-sets[OF range])
    show (⋃i. disjointed A i) = space M
      and disjoint-family (disjointed A)
      using disjoint-family-disjointed UN-disjointed-eq[of A] space range
      by auto
    show emeasure M (disjointed A i) ≠ ∞ for i
    proof −
      have emeasure M (disjointed A i) ≤ emeasure M (A i)
        using range disjointed-subset[of A i] by (auto intro!: emeasure-mono)
      then show ?thesis using measure[of i] by (auto simp: top-unique)
    qed
  qed
qed

lemma (in sigma-finite-measure) sigma-finite-incseq:
  obtains A :: nat ⇒ 'a set
  where range A ⊆ sets M (⋃i. A i) = space M ⋀i. emeasure M (A i) ≠ ∞
incseq A
proof −
  obtain F :: nat ⇒ 'a set where
    F: range F ⊆ sets M (⋃i. F i) = space M ⋀i. emeasure M (F i) ≠ ∞
    using sigma-finite by blast
  show thesis
  proof (rule that[of λn. ⋃i≤n. F i])
    show range (λn. ⋃i≤n. F i) ⊆ sets M
      using F by (force simp: incseq-def)
    show (⋃n. ⋃i≤n. F i) = space M
    proof −
      from F have ⋀x. x ∈ space M ⟹ ∃i. x ∈ F i by auto
      with F show ?thesis by fastforce
    qed
    show emeasure M (⋃i≤n. F i) ≠ ∞ for n
```

```
    proof −
      have emeasure M (⋃ i≤n. F i) ≤ (∑ i≤n. emeasure M (F i))
        using F by (auto intro!: emeasure-subadditive-finite)
      also have ... < ∞
        using F by (auto simp: setsum-Pinfty less-top)
      finally show ?thesis by simp
    qed
    show incseq (λn. ⋃ i≤n. F i)
      by (force simp: incseq-def)
  qed
qed
```

3.8 Measure space induced by distribution of *op* $\rightarrow_M$-functions

```
definition distr :: 'a measure ⇒ 'b measure ⇒ ('a ⇒ 'b) ⇒ 'b measure where
  distr M N f = measure-of (space N) (sets N) (λA. emeasure M (f −‘ A ∩ space
M))

lemma
  shows sets-distr[simp, measurable-cong]: sets (distr M N f) = sets N
    and space-distr[simp]: space (distr M N f) = space N
  by (auto simp: distr-def)

lemma
  shows measurable-distr-eq1[simp]: measurable (distr Mf Nf f) Mf' = measurable
Nf Mf'
    and measurable-distr-eq2[simp]: measurable Mg' (distr Mg Ng g) = measurable
Mg' Ng
  by (auto simp: measurable-def)

lemma distr-cong:
  M = K ⟹ sets N = sets L ⟹ (⋀x. x ∈ space M ⟹ f x = g x) ⟹ distr M
N f = distr K L g
 using sets-eq-imp-space-eq[of N L] by (simp add: distr-def Int-def cong: rev-conj-cong)

lemma emeasure-distr:
  fixes f :: 'a ⇒ 'b
  assumes f: f ∈ measurable M N and A: A ∈ sets N
  shows emeasure (distr M N f) A = emeasure M (f −‘ A ∩ space M) (is - = ?μ
A)
  unfolding distr-def
proof (rule emeasure-measure-of-sigma)
  show positive (sets N) ?μ
    by (auto simp: positive-def)

  show countably-additive (sets N) ?μ
  proof (intro countably-additiveI)
    fix A :: nat ⇒ 'b set assume range A ⊆ sets N disjoint-family A
    then have A: ⋀i. A i ∈ sets N (⋃ i. A i) ∈ sets N by auto
```

```
    then have ∗: range (λi. f −‘ (A i) ∩ space M) ⊆ sets M
      using f by (auto simp: measurable-def)
    moreover have (⋃ i. f −‘ A i ∩ space M) ∈ sets M
      using ∗ by blast
    moreover have ∗∗: disjoint-family (λi. f −‘ A i ∩ space M)
      using ‹disjoint-family A› by (auto simp: disjoint-family-on-def)
    ultimately show (∑ i. ?μ (A i)) = ?μ (⋃ i. A i)
      using suminf-emeasure[OF - ∗∗] A f
      by (auto simp: comp-def vimage-UN)
  qed
  show sigma-algebra (space N) (sets N) ..
qed fact

lemma emeasure-Collect-distr:
  assumes X[measurable]: X ∈ measurable M N Measurable.pred N P
  shows emeasure (distr M N X) {x∈space N. P x} = emeasure M {x∈space M.
P (X x)}
  by (subst emeasure-distr)
     (auto intro!: arg-cong2[where f=emeasure] X(1)[THEN measurable-space])

lemma emeasure-lfp2[consumes 1, case-names cont f measurable]:
  assumes P M
  assumes cont: sup-continuous F
  assumes f: ⋀M. P M ⟹ f ∈ measurable M' M
  assumes ∗: ⋀M A. P M ⟹ (⋀N. P N ⟹ Measurable.pred N A) ⟹ Mea-
surable.pred M (F A)
  shows emeasure M' {x∈space M'. lfp F (f x)} = (SUP i. emeasure M' {x∈space
M'. (F ^^ i) (λx. False) (f x)})
proof (subst (1 2) emeasure-Collect-distr[symmetric, where X=f])
  show f ∈ measurable M' M  f ∈ measurable M' M
    using f[OF ‹P M›] by auto
  { fix i show Measurable.pred M ((F ^^ i) (λx. False))
    using ‹P M› by (induction i arbitrary: M) (auto intro!: ∗) }
  show Measurable.pred M (lfp F)
    using ‹P M› cont ∗ by (rule measurable-lfp-coinduct[of P])

  have emeasure (distr M' M f) {x ∈ space (distr M' M f). lfp F x} =
     (SUP i. emeasure (distr M' M f) {x ∈ space (distr M' M f). (F ^^ i) (λx.
False) x})
    using ‹P M›
  proof (coinduction arbitrary: M rule: emeasure-lfp')
    case (measurable A N) then have ⋀N. P N ⟹ Measurable.pred (distr M'
N f) A
      by metis
    then have ⋀N. P N ⟹ Measurable.pred N A
      by simp
    with ‹P N›[THEN ∗] show ?case
      by auto
  qed fact
```

then show *emeasure* (*distr M' M f*) {*x* ∈ *space M*. *lfp F x*} =
(*SUP i*. *emeasure* (*distr M' M f*) {*x* ∈ *space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*})
by *simp*
qed

lemma *distr-id*[*simp*]: *distr N N* (λ*x*. *x*) = *N*
by (*rule measure-eqI*) (*auto simp*: *emeasure-distr*)

lemma *measure-distr*:
f ∈ *measurable M N* ⟹ *S* ∈ *sets N* ⟹ *measure* (*distr M N f*) *S* = *measure M* (*f* −‘ *S* ∩ *space M*)
by (*simp add*: *emeasure-distr measure-def*)

lemma *distr-cong-AE*:
assumes *1*: *M* = *K sets N* = *sets L* **and**
2: (*AE x in M*. *f x* = *g x*) **and** *f* ∈ *measurable M N* **and** *g* ∈ *measurable K L*
shows *distr M N f* = *distr K L g*
proof (*rule measure-eqI*)
fix *A* **assume** *A* ∈ *sets* (*distr M N f*)
with *assms* **show** *emeasure* (*distr M N f*) *A* = *emeasure* (*distr K L g*) *A*
by (*auto simp add*: *emeasure-distr intro*!: *emeasure-eq-AE measurable-sets*)
qed (*insert 1*, *simp*)

lemma *AE-distrD*:
assumes *f*: *f* ∈ *measurable M M'*
and *AE*: *AE x in distr M M' f*. *P x*
shows *AE x in M*. *P* (*f x*)
proof −
from *AE*[*THEN AE-E*] **guess** *N* **.**
with *f* **show** *?thesis*
unfolding *eventually-ae-filter*
by (*intro bexI*[*of - f* −‘ *N* ∩ *space M*])
(*auto simp*: *emeasure-distr measurable-def*)
qed

lemma *AE-distr-iff*:
assumes *f*[*measurable*]: *f* ∈ *measurable M N* **and** *P*[*measurable*]: {*x* ∈ *space N*. *P x*} ∈ *sets N*
shows (*AE x in distr M N f*. *P x*) ⟷ (*AE x in M*. *P* (*f x*))
proof (*subst* (*1 2*) *AE-iff-measurable*[*OF - refl*])
have *f* −‘ {*x*∈*space N*. ¬ *P x*} ∩ *space M* = {*x* ∈ *space M*. ¬ *P* (*f x*)}
using *f*[*THEN measurable-space*] **by** *auto*
then show (*emeasure* (*distr M N f*) {*x* ∈ *space* (*distr M N f*). ¬ *P x*} = *0*) =
(*emeasure M* {*x* ∈ *space M*. ¬ *P* (*f x*)} = *0*)
by (*simp add*: *emeasure-distr*)
qed *auto*

lemma *null-sets-distr-iff*:
f ∈ *measurable M N* ⟹ *A* ∈ *null-sets* (*distr M N f*) ⟷ *f* −‘ *A* ∩ *space M* ∈

null-sets M ∧ A ∈ sets N
by (*auto simp add*: *null-sets-def emeasure-distr*)

lemma *distr-distr*:
g ∈ measurable N L ⟹ f ∈ measurable M N ⟹ distr (*distr M N f*) *L g* = *distr M L* (*g ∘ f*)
by (*auto simp add*: *emeasure-distr measurable-space*
intro!: *arg-cong*[**where** *f*=*emeasure M*] *measure-eqI*)

3.9 Real measure values

lemma *ring-of-finite-sets*: *ring-of-sets* (*space M*) {*A*∈*sets M*. *emeasure M A* ≠ *top*}
proof (*rule ring-of-setsI*)
show *a ∈* {*A ∈ sets M*. *emeasure M A ≠ top*} *⟹ b ∈* {*A ∈ sets M*. *emeasure M A ≠ top*} *⟹*
a ∪ b ∈ {*A ∈ sets M*. *emeasure M A ≠ top*} **for** *a b*
using *emeasure-subadditive*[*of a M b*] **by** (*auto simp*: *top-unique*)

show *a ∈* {*A ∈ sets M*. *emeasure M A ≠ top*} *⟹ b ∈* {*A ∈ sets M*. *emeasure M A ≠ top*} *⟹*
a − b ∈ {*A ∈ sets M*. *emeasure M A ≠ top*} **for** *a b*
using *emeasure-mono*[*of a − b a M*] **by** (*auto simp*: *Diff-subset top-unique*)
qed (*auto dest*: *sets.sets-into-space*)

lemma *measure-nonneg*[*simp*]: *0 ≤ measure M A*
unfolding *measure-def* **by** (*auto simp*: *enn2real-nonneg*)

lemma *zero-less-measure-iff*: *0 < measure M A ⟷ measure M A ≠ 0*
using *measure-nonneg*[*of M A*] **by** (*auto simp add*: *le-less*)

lemma *measure-le-0-iff*: *measure M X ≤ 0 ⟷ measure M X = 0*
using *measure-nonneg*[*of M X*] **by** *linarith*

lemma *measure-empty*[*simp*]: *measure M {} = 0*
unfolding *measure-def* **by** (*simp add*: *zero-ennreal.rep-eq*)

lemma *emeasure-eq-ennreal-measure*:
emeasure M A ≠ top ⟹ emeasure M A = ennreal (*measure M A*)
by (*cases emeasure M A rule*: *ennreal-cases*) (*auto simp*: *measure-def*)

lemma *measure-zero-top*: *emeasure M A = top ⟹ measure M A = 0*
by (*simp add*: *measure-def enn2ereal-top*)

lemma *measure-eq-emeasure-eq-ennreal*: *0 ≤ x ⟹ emeasure M A = ennreal x ⟹ measure M A = x*
using *emeasure-eq-ennreal-measure*[*of M A*]
by (*cases A ∈ M*) (*auto simp*: *measure-notin-sets emeasure-notin-sets*)

lemma *enn2real-plus*:$a < top \Longrightarrow b < top \Longrightarrow enn2real\ (a + b) = enn2real\ a + enn2real\ b$
by (*simp add*: *enn2real-def plus-ennreal.rep-eq real-of-ereal-add enn2ereal-nonneg less-top*
del: *real-of-ereal-enn2ereal*)

lemma *measure-Union*:
$emeasure\ M\ A \neq \infty \Longrightarrow emeasure\ M\ B \neq \infty \Longrightarrow A \in sets\ M \Longrightarrow B \in sets\ M \Longrightarrow A \cap B = \{\} \Longrightarrow$
$measure\ M\ (A \cup B) = measure\ M\ A + measure\ M\ B$
by (*simp add*: *measure-def enn2ereal-nonneg plus-emeasure*[*symmetric*] *enn2real-plus less-top*)

lemma *disjoint-family-on-insert*:
$i \notin I \Longrightarrow disjoint\text{-}family\text{-}on\ A\ (insert\ i\ I) \longleftrightarrow A\ i \cap (\bigcup i{\in}I.\ A\ i) = \{\} \wedge disjoint\text{-}family\text{-}on\ A\ I$
by (*fastforce simp*: *disjoint-family-on-def*)

lemma *measure-finite-Union*:
$finite\ S \Longrightarrow A`S \subseteq sets\ M \Longrightarrow disjoint\text{-}family\text{-}on\ A\ S \Longrightarrow (\bigwedge i.\ i \in S \Longrightarrow emeasure\ M\ (A\ i) \neq \infty) \Longrightarrow$
$measure\ M\ (\bigcup i{\in}S.\ A\ i) = (\sum i{\in}S.\ measure\ M\ (A\ i))$
by (*induction S rule*: *finite-induct*)
(*auto simp*: *disjoint-family-on-insert measure-Union setsum-emeasure*[*symmetric*] *sets.countable-UN'*[*OF countable-finite*])

lemma *measure-Diff*:
assumes *finite*: $emeasure\ M\ A \neq \infty$
and *measurable*: $A \in sets\ M\ B \in sets\ M\ B \subseteq A$
shows $measure\ M\ (A - B) = measure\ M\ A - measure\ M\ B$
proof −
have $emeasure\ M\ (A - B) \leq emeasure\ M\ A\ emeasure\ M\ B \leq emeasure\ M\ A$
using *measurable* **by** (*auto intro*!: *emeasure-mono*)
hence $measure\ M\ ((A - B) \cup B) = measure\ M\ (A - B) + measure\ M\ B$
using *measurable finite* **by** (*rule-tac measure-Union*) (*auto simp*: *top-unique*)
thus *?thesis* **using** ‹$B \subseteq A$› **by** (*auto simp*: *Un-absorb2*)
qed

lemma *measure-UNION*:
assumes *measurable*: $range\ A \subseteq sets\ M\ disjoint\text{-}family\ A$
assumes *finite*: $emeasure\ M\ (\bigcup i.\ A\ i) \neq \infty$
shows $(\lambda i.\ measure\ M\ (A\ i))\ sums\ (measure\ M\ (\bigcup i.\ A\ i))$
proof −
have $(\lambda i.\ emeasure\ M\ (A\ i))\ sums\ (emeasure\ M\ (\bigcup i.\ A\ i))$
unfolding *suminf-emeasure*[*OF measurable*, *symmetric*] **by** (*simp add*: *summable-sums*)
moreover
{ fix *i*
have $emeasure\ M\ (A\ i) \leq emeasure\ M\ (\bigcup i.\ A\ i)$
using *measurable* **by** (*auto intro*!: *emeasure-mono*)

```
    then have emeasure M (A i) = ennreal ((measure M (A i)))
      using finite by (intro emeasure-eq-ennreal-measure) (auto simp: top-unique)
}
  ultimately show ?thesis using finite
    by (subst (asm) (2) emeasure-eq-ennreal-measure)
       (simp-all add: measure-nonneg)
qed

lemma measure-subadditive:
  assumes measurable: A ∈ sets M B ∈ sets M
  and fin: emeasure M A ≠ ∞ emeasure M B ≠ ∞
  shows measure M (A ∪ B) ≤ measure M A + measure M B
proof −
  have emeasure M (A ∪ B) ≠ ∞
    using emeasure-subadditive[OF measurable] fin by (auto simp: top-unique)
  then show (measure M (A ∪ B)) ≤ (measure M A) + (measure M B)
    using emeasure-subadditive[OF measurable] fin
    apply simp
    apply (subst (asm) (2 3 4) emeasure-eq-ennreal-measure)
    apply (auto simp add: ennreal-plus[symmetric] simp del: ennreal-plus)
    done
qed

lemma measure-subadditive-finite:
  assumes A: finite I A`I ⊆ sets M and fin: ⋀i. i ∈ I ⟹ emeasure M (A i) ≠
∞
  shows measure M (⋃i∈I. A i) ≤ (∑ i∈I. measure M (A i))
proof −
  { have emeasure M (⋃i∈I. A i) ≤ (∑ i∈I. emeasure M (A i))
      using emeasure-subadditive-finite[OF A] .
    also have ... < ∞
      using fin by (simp add: less-top A)
    finally have emeasure M (⋃i∈I. A i) ≠ top by simp }
  note ∗ = this
  show ?thesis
    using emeasure-subadditive-finite[OF A] fin
    unfolding emeasure-eq-ennreal-measure[OF ∗]
   by (simp-all add: setsum-ennreal measure-nonneg setsum-nonneg emeasure-eq-ennreal-measure)
qed

lemma measure-subadditive-countably:
  assumes A: range A ⊆ sets M and fin: (∑ i. emeasure M (A i)) ≠ ∞
  shows measure M (⋃ i. A i) ≤ (∑ i. measure M (A i))
proof −
  from fin have ∗∗: ⋀i. emeasure M (A i) ≠ top
    using ennreal-suminf-lessD[of λi. emeasure M (A i)] by (simp add: less-top)
  { have emeasure M (⋃ i. A i) ≤ (∑ i. emeasure M (A i))
      using emeasure-subadditive-countably[OF A] .
    also have ... < ∞
```

using *fin* **by** (*simp add*: *less-top*)
finally have *emeasure M* (⋃ *i*. *A i*) ≠ *top* **by** *simp* **}**
then have *ennreal* (*measure M* (⋃ *i*. *A i*)) = *emeasure M* (⋃ *i*. *A i*)
by (*rule emeasure-eq-ennreal-measure*[*symmetric*])
also have ... ≤ (∑ *i*. *emeasure M* (*A i*))
using *emeasure-subadditive-countably*[*OF A*] **.**
also have ... = *ennreal* (∑ *i*. *measure M* (*A i*))
using *fin* **unfolding** *emeasure-eq-ennreal-measure*[*OF* **]
by (*subst suminf-ennreal*) (*auto simp*: **)
finally show *?thesis*
apply (*rule ennreal-le-iff*[*THEN iffD1*, *rotated*])
apply (*intro suminf-nonneg allI measure-nonneg summable-suminf-not-top*)
using *fin*
apply (*simp add*: *emeasure-eq-ennreal-measure*[*OF* **])
done
qed

lemma *measure-eq-setsum-singleton*:
finite S ⟹ (⋀*x*. *x* ∈ *S* ⟹ {*x*} ∈ *sets M*) ⟹ (⋀*x*. *x* ∈ *S* ⟹ *emeasure M* {*x*} ≠ ∞) ⟹
measure M S = (∑ *x*∈*S*. *measure M* {*x*})
using *emeasure-eq-setsum-singleton*[*of S M*]
by (*intro measure-eq-emeasure-eq-ennreal*) (*auto simp*: *setsum-nonneg emeasure-eq-ennreal-measure*)

lemma *Lim-measure-incseq*:
assumes *A*: *range A* ⊆ *sets M incseq A* **and** *fin*: *emeasure M* (⋃ *i*. *A i*) ≠ ∞
shows (λ*i*. *measure M* (*A i*)) ⟶ *measure M* (⋃ *i*. *A i*)
proof (*rule tendsto-ennrealD*)
have *ennreal* (*measure M* (⋃ *i*. *A i*)) = *emeasure M* (⋃ *i*. *A i*)
using *fin* **by** (*auto simp*: *emeasure-eq-ennreal-measure*)
moreover have *ennreal* (*measure M* (*A i*)) = *emeasure M* (*A i*) **for** *i*
using *assms emeasure-mono*[*of A - ⋃ i*. *A i M*]
by (*intro emeasure-eq-ennreal-measure*[*symmetric*]) (*auto simp*: *less-top UN-upper intro*: *le-less-trans*)
ultimately show (λ*x*. *ennreal* (*Sigma-Algebra.measure M* (*A x*))) ⟶ *ennreal* (*Sigma-Algebra.measure M* (⋃ *i*. *A i*))
using *A* **by** (*auto intro*!: *Lim-emeasure-incseq*)
qed *auto*

lemma *Lim-measure-decseq*:
assumes *A*: *range A* ⊆ *sets M decseq A* **and** *fin*: ⋀*i*. *emeasure M* (*A i*) ≠ ∞
shows (λ*n*. *measure M* (*A n*)) ⟶ *measure M* (⋂ *i*. *A i*)
proof (*rule tendsto-ennrealD*)
have *ennreal* (*measure M* (⋂ *i*. *A i*)) = *emeasure M* (⋂ *i*. *A i*)
using *fin*[*of 0*] *A emeasure-mono*[*of* ⋂ *i*. *A i A 0 M*]
by (*auto intro*!: *emeasure-eq-ennreal-measure*[*symmetric*] *simp*: *INT-lower less-top intro*: *le-less-trans*)
moreover have *ennreal* (*measure M* (*A i*)) = *emeasure M* (*A i*) **for** *i*
using *A fin*[*of i*] **by** (*intro emeasure-eq-ennreal-measure*[*symmetric*]) *auto*

ultimately show $(\lambda x.\ ennreal\ (Sigma\text{-}Algebra.measure\ M\ (A\ x))) \longrightarrow ennreal\ (Sigma\text{-}Algebra.measure\ M\ (\bigcap i.\ A\ i))$
using *fin A* **by** (*auto intro*!: *Lim-emeasure-decseq*)
qed *auto*

3.10 Measure spaces with *emeasure M* (*space M*) $< \infty$

locale *finite-measure* = *sigma-finite-measure M* **for** *M* +
assumes *finite-emeasure-space*: *emeasure M* (*space M*) $\neq$ *top*

lemma *finite-measureI*[*Pure.intro*!]:
emeasure M (*space M*) $\neq \infty \Longrightarrow$ *finite-measure M*
proof qed (*auto intro*!: *exI*[*of* - {*space M*}])

lemma (**in** *finite-measure*) *emeasure-finite*[*simp*, *intro*]: *emeasure M A* $\neq$ *top*
using *finite-emeasure-space emeasure-space*[*of M A*] **by** (*auto simp*: *top-unique*)

lemma (**in** *finite-measure*) *emeasure-eq-measure*: *emeasure M A* = *ennreal* (*measure M A*)
by (*intro emeasure-eq-ennreal-measure*) *simp*

lemma (**in** *finite-measure*) *emeasure-real*: $\exists r.\ 0 \leq r \wedge$ *emeasure M A* = *ennreal r*
using *emeasure-finite*[*of A*] **by** (*cases emeasure M A rule*: *ennreal-cases*) *auto*

lemma (**in** *finite-measure*) *bounded-measure*: *measure M A* $\leq$ *measure M* (*space M*)
using *emeasure-space*[*of M A*] *emeasure-real*[*of A*] *emeasure-real*[*of space M*] **by** (*auto simp*: *measure-def*)

lemma (**in** *finite-measure*) *finite-measure-Diff*:
assumes *sets*: *A* $\in$ *sets M B* $\in$ *sets M* **and** *B* $\subseteq$ *A*
shows *measure M* (*A* $-$ *B*) = *measure M A* $-$ *measure M B*
using *measure-Diff*[*OF* - *assms*] **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-Union*:
assumes *sets*: *A* $\in$ *sets M B* $\in$ *sets M* **and** *A* $\cap$ *B* = {}
shows *measure M* (*A* $\cup$ *B*) = *measure M A* + *measure M B*
using *measure-Union*[*OF* - - *assms*] **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-finite-Union*:
assumes *measurable*: *finite S A*‘*S* $\subseteq$ *sets M disjoint-family-on A S*
shows *measure M* ($\bigcup i \in S.\ A\ i$) = ($\sum i \in S.\ measure\ M\ (A\ i)$)
using *measure-finite-Union*[*OF assms*] **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-UNION*:
assumes *A*: *range A* $\subseteq$ *sets M disjoint-family A*
shows ($\lambda i.\ measure\ M\ (A\ i)$) *sums* (*measure M* ($\bigcup i.\ A\ i$))
using *measure-UNION*[*OF A*] **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-mono*:
assumes $A \subseteq B$ $B \in sets\ M$ **shows** $measure\ M\ A \leq measure\ M\ B$
using *emeasure-mono*[*OF assms*] *emeasure-real*[*of A*] *emeasure-real*[*of B*] **by** (*auto simp*: *measure-def*)

lemma (**in** *finite-measure*) *finite-measure-subadditive*:
assumes *m*: $A \in sets\ M$ $B \in sets\ M$
shows $measure\ M\ (A \cup B) \leq measure\ M\ A + measure\ M\ B$
using *measure-subadditive*[*OF m*] **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-subadditive-finite*:
assumes *finite I* $A`I \subseteq sets\ M$ **shows** $measure\ M\ (\bigcup i{\in}I.\ A\ i) \leq (\sum i{\in}I.\ measure\ M\ (A\ i))$
using *measure-subadditive-finite*[*OF assms*] **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-subadditive-countably*:
$range\ A \subseteq sets\ M \Longrightarrow summable\ (\lambda i.\ measure\ M\ (A\ i)) \Longrightarrow measure\ M\ (\bigcup i.\ A\ i) \leq (\sum i.\ measure\ M\ (A\ i))$
by (*rule measure-subadditive-countably*)
(*simp-all add*: *ennreal-suminf-neq-top emeasure-eq-measure*)

lemma (**in** *finite-measure*) *finite-measure-eq-setsum-singleton*:
assumes *finite S* **and** $*$: $\bigwedge x.\ x \in S \Longrightarrow \{x\} \in sets\ M$
shows $measure\ M\ S = (\sum x{\in}S.\ measure\ M\ \{x\})$
using *measure-eq-setsum-singleton*[*OF assms*] **by** *simp*

lemma (**in** *finite-measure*) *finite-Lim-measure-incseq*:
assumes *A*: $range\ A \subseteq sets\ M$ *incseq A*
shows $(\lambda i.\ measure\ M\ (A\ i)) \longrightarrow measure\ M\ (\bigcup i.\ A\ i)$
using *Lim-measure-incseq*[*OF A*] **by** *simp*

lemma (**in** *finite-measure*) *finite-Lim-measure-decseq*:
assumes *A*: $range\ A \subseteq sets\ M$ *decseq A*
shows $(\lambda n.\ measure\ M\ (A\ n)) \longrightarrow measure\ M\ (\bigcap i.\ A\ i)$
using *Lim-measure-decseq*[*OF A*] **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-compl*:
assumes *S*: $S \in sets\ M$
shows $measure\ M\ (space\ M - S) = measure\ M\ (space\ M) - measure\ M\ S$
using *measure-Diff*[*OF - sets.top S sets.sets-into-space*] *S* **by** *simp*

lemma (**in** *finite-measure*) *finite-measure-mono-AE*:
assumes *imp*: *AE x in M*. $x \in A \longrightarrow x \in B$ **and** *B*: $B \in sets\ M$
shows $measure\ M\ A \leq measure\ M\ B$
using *assms emeasure-mono-AE*[*OF imp B*]
by (*simp add*: *emeasure-eq-measure*)

lemma (**in** *finite-measure*) *finite-measure-eq-AE*:

assumes *iff*: *AE x in M*. $x \in A \longleftrightarrow x \in B$
assumes *A*: $A \in sets\ M$ **and** *B*: $B \in sets\ M$
shows *measure M A* = *measure M B*
using *assms emeasure-eq-AE*[*OF assms*] **by** (*simp add*: *emeasure-eq-measure*)

lemma (**in** *finite-measure*) *measure-increasing*: *increasing M* (*measure M*)
by (*auto intro*!: *finite-measure-mono simp*: *increasing-def*)

lemma (**in** *finite-measure*) *measure-zero-union*:
assumes $s \in sets\ M\ t \in sets\ M\ measure\ M\ t = 0$
shows *measure M* $(s \cup t)$ = *measure M s*
using *assms*
proof −
have *measure M* $(s \cup t) \leq measure\ M\ s$
using *finite-measure-subadditive*[*of s t*] *assms* **by** *auto*
moreover have *measure M* $(s \cup t) \geq measure\ M\ s$
using *assms* **by** (*blast intro*: *finite-measure-mono*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *finite-measure*) *measure-eq-compl*:
assumes $s \in sets\ M\ t \in sets\ M$
assumes *measure M* (*space M* − *s*) = *measure M* (*space M* − *t*)
shows *measure M s* = *measure M t*
using *assms finite-measure-compl* **by** *auto*

lemma (**in** *finite-measure*) *measure-eq-bigunion-image*:
assumes $range\ f \subseteq sets\ M\ range\ g \subseteq sets\ M$
assumes *disjoint-family f disjoint-family g*
assumes $\bigwedge n :: nat.\ measure\ M\ (f\ n) = measure\ M\ (g\ n)$
shows *measure M* $(\bigcup i.\ f\ i)$ = *measure M* $(\bigcup i.\ g\ i)$
using *assms*
proof −
have *a*: $(\lambda\ i.\ measure\ M\ (f\ i))\ sums\ (measure\ M\ (\bigcup i.\ f\ i))$
by (*rule finite-measure-UNION*[*OF assms*(*1*,*3*)])
have *b*: $(\lambda\ i.\ measure\ M\ (g\ i))\ sums\ (measure\ M\ (\bigcup i.\ g\ i))$
by (*rule finite-measure-UNION*[*OF assms*(*2*,*4*)])
show *?thesis* **using** *sums-unique*[*OF b*] *sums-unique*[*OF a*] *assms* **by** *simp*
qed

lemma (**in** *finite-measure*) *measure-countably-zero*:
assumes $range\ c \subseteq sets\ M$
assumes $\bigwedge i.\ measure\ M\ (c\ i) = 0$
shows *measure M* $(\bigcup i :: nat.\ c\ i) = 0$
proof (*rule antisym*)
show *measure M* $(\bigcup i :: nat.\ c\ i) \leq 0$
using *finite-measure-subadditive-countably*[*OF assms*(*1*)] **by** (*simp add*: *assms*(*2*))
qed *simp*

lemma (**in** *finite-measure*) *measure-space-inter*:
assumes *events*:$s \in$ *sets* M $t \in$ *sets* M
assumes *measure* M t = *measure* M (*space* M)
shows *measure* M $(s \cap t)$ = *measure* M s
proof −
have *measure* M ((*space* M − s) ∪ (*space* M − t)) = *measure* M (*space* M − s)
using *events assms finite-measure-compl*[*of t*] **by** (*auto intro*!: *measure-zero-union*)
also have (*space* M − s) ∪ (*space* M − t) = *space* M − $(s \cap t)$
by *blast*
finally show *measure* M $(s \cap t)$ = *measure* M s
using *events* **by** (*auto intro*!: *measure-eq-compl*[*of* $s \cap t$ s])
qed

lemma (**in** *finite-measure*) *measure-equiprobable-finite-unions*:
assumes s: *finite* s $\bigwedge x.\ x \in s \Longrightarrow \{x\} \in$ *sets* M
assumes $\bigwedge\ x\ y.\ [\![x \in s;\ y \in s]\!] \Longrightarrow$ *measure* M $\{x\}$ = *measure* M $\{y\}$
shows *measure* M s = *real* (*card* s) ∗ *measure* M $\{SOME\ x.\ x \in s\}$
proof *cases*
assume $s \neq \{\}$
then have $\exists\ x.\ x \in s$ **by** *blast*
from *someI-ex*[*OF this*] *assms*
have *prob-some*: $\bigwedge\ x.\ x \in s \Longrightarrow$ *measure* M $\{x\}$ = *measure* M $\{SOME\ y.\ y \in s\}$ **by** *blast*
have *measure* M s = $(\sum\ x \in s.$ *measure* M $\{x\})$
using *finite-measure-eq-setsum-singleton*[*OF s*] **by** *simp*
also have ... = $(\sum\ x \in s.$ *measure* M $\{SOME\ y.\ y \in s\})$ **using** *prob-some* **by** *auto*
also have ... = *real* (*card* s) ∗ *measure* M $\{(SOME\ x.\ x \in s)\}$
using *setsum-constant assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed *simp*

lemma (**in** *finite-measure*) *measure-real-sum-image-fn*:
assumes $e \in$ *sets* M
assumes $\bigwedge\ x.\ x \in s \Longrightarrow e \cap f\ x \in$ *sets* M
assumes *finite* s
assumes *disjoint*: $\bigwedge\ x\ y.\ [\![x \in s\ ;\ y \in s\ ;\ x \neq y]\!] \Longrightarrow f\ x \cap f\ y = \{\}$
assumes *upper*: *space* $M \subseteq (\bigcup i \in s.\ f\ i)$
shows *measure* M e = $(\sum\ x \in s.$ *measure* M $(e \cap f\ x))$
proof −
have $e \subseteq (\bigcup i \in s.\ f\ i)$
using ‹$e \in$ *sets* M› *sets.sets-into-space upper* **by** *blast*
then have *e*: $e = (\bigcup i \in s.\ e \cap f\ i)$
by *auto*
hence *measure* M e = *measure* M $(\bigcup i \in s.\ e \cap f\ i)$ **by** *simp*
also have ... = $(\sum\ x \in s.$ *measure* M $(e \cap f\ x))$
proof (*rule finite-measure-finite-Union*)
show *finite* s **by** *fact*

show $(\lambda i.\ e \cap f\ i)`s \subseteq sets\ M$ **using** *assms(2)* **by** *auto*
show *disjoint-family-on* $(\lambda i.\ e \cap f\ i)\ s$
using *disjoint* **by** (*auto simp*: *disjoint-family-on-def*)
qed
finally show *?thesis* **.**
qed

lemma (**in** *finite-measure*) *measure-exclude*:
assumes $A \in sets\ M\ B \in sets\ M$
assumes $measure\ M\ A = measure\ M\ (space\ M)\ A \cap B = \{\}$
shows $measure\ M\ B = 0$
using *measure-space-inter*[*of B A*] *assms* **by** (*auto simp*: *ac-simps*)
lemma (**in** *finite-measure*) *finite-measure-distr*:
assumes *f*: $f \in measurable\ M\ M'$
shows *finite-measure* (*distr M M' f*)
proof (*rule finite-measureI*)
have $f\ -`\ space\ M' \cap space\ M = space\ M$ **using** *f* **by** (*auto dest*: *measurable-space*)
with *f* **show** $emeasure\ (distr\ M\ M'\ f)\ (space\ (distr\ M\ M'\ f)) \neq \infty$ **by** (*auto simp*: *emeasure-distr*)
qed

lemma *emeasure-gfp*[*consumes 1*, *case-names cont measurable*]:
assumes *sets*[*simp*]: $\bigwedge s.\ sets\ (M\ s) = sets\ N$
assumes $\bigwedge s.\ finite\text{-}measure\ (M\ s)$
assumes *cont*: *inf-continuous F inf-continuous f*
assumes *meas*: $\bigwedge P.\ Measurable.pred\ N\ P \Longrightarrow Measurable.pred\ N\ (F\ P)$
assumes *iter*: $\bigwedge P\ s.\ Measurable.pred\ N\ P \Longrightarrow emeasure\ (M\ s)\ \{x \in space\ N.\ F\ P\ x\} = f\ (\lambda s.\ emeasure\ (M\ s)\ \{x \in space\ N.\ P\ x\})\ s$
assumes *bound*: $\bigwedge P.\ f\ P \leq f\ (\lambda s.\ emeasure\ (M\ s)\ (space\ (M\ s)))$
shows $emeasure\ (M\ s)\ \{x \in space\ N.\ gfp\ F\ x\} = gfp\ f\ s$
proof (*subst gfp-transfer-bounded*[**where** $\alpha = \lambda F\ s.\ emeasure\ (M\ s)\ \{x \in space\ N.\ F\ x\}$ **and** *g*=*f* **and** *f*=*F* **and**
P=*Measurable.pred N*, *symmetric*])
interpret *finite-measure M s* **for** *s* **by** *fact*
fix *C* **assume** *decseq C* $\bigwedge i.\ Measurable.pred\ N\ (C\ i)$
then show $(\lambda s.\ emeasure\ (M\ s)\ \{x \in space\ N.\ (INF\ i.\ C\ i)\ x\}) = (INF\ i.\ (\lambda s.\ emeasure\ (M\ s)\ \{x \in space\ N.\ C\ i\ x\}))$
unfolding *INF-apply*[*abs-def*]
by (*subst INF-emeasure-decseq*) (*auto simp*: *antimono-def fun-eq-iff intro*!: *arg-cong2*[**where** *f*=*emeasure*])
next
show $f\ x \leq (\lambda s.\ emeasure\ (M\ s)\ \{x \in space\ N.\ F\ top\ x\})$ **for** *x*
using *bound*[*of x*] *sets-eq-imp-space-eq*[*OF sets*] **by** (*simp add*: *iter*)
qed (*auto simp add*: *iter le-fun-def INF-apply*[*abs-def*] *intro*!: *meas cont*)

3.11 Counting space

lemma *strict-monoI-Suc*:
assumes *ord* [*simp*]: $(\bigwedge n.\ f\ n < f\ (Suc\ n))$ **shows** *strict-mono f*

```
  unfolding strict-mono-def
proof safe
  fix n m :: nat assume n < m then show f n < f m
    by (induct m) (auto simp: less-Suc-eq intro: less-trans ord)
qed

lemma emeasure-count-space:
  assumes X ⊆ A shows emeasure (count-space A) X = (if finite X then of-nat
(card X) else ∞)
    (is - = ?M X)
  unfolding count-space-def
proof (rule emeasure-measure-of-sigma)
  show X ∈ Pow A using ‹X ⊆ A› by auto
  show sigma-algebra A (Pow A) by (rule sigma-algebra-Pow)
  show positive: positive (Pow A) ?M
    by (auto simp: positive-def)
  have additive: additive (Pow A) ?M
    by (auto simp: card-Un-disjoint additive-def)

  interpret ring-of-sets A Pow A
    by (rule ring-of-setsI) auto
  show countably-additive (Pow A) ?M
    unfolding countably-additive-iff-continuous-from-below[OF positive additive]
  proof safe
    fix F :: nat ⇒ 'a set assume incseq F
    show (λi. ?M (F i)) ⟶ ?M (⋃ i. F i)
    proof cases
      assume ∃ i. ∀ j≥i. F i = F j
      then guess i .. note i = this
      { fix j from i ‹incseq F› have F j ⊆ F i
          by (cases i ≤ j) (auto simp: incseq-def) }
      then have eq: (⋃ i. F i) = F i
        by auto
      with i show ?thesis
        by (auto intro!: Lim-eventually eventually-sequentiallyI[where c=i])
    next
      assume ¬ (∃ i. ∀ j≥i. F i = F j)
      then obtain f where f: ⋀i. i ≤ f i ⋀i. F i ≠ F (f i) by metis
      then have ⋀i. F i ⊆ F (f i) using ‹incseq F› by (auto simp: incseq-def)
      with f have *: ⋀i. F i ⊂ F (f i) by auto

      have incseq (λi. ?M (F i))
        using ‹incseq F› unfolding incseq-def by (auto simp: card-mono dest:
finite-subset)
      then have (λi. ?M (F i)) ⟶ (SUP n. ?M (F n))
        by (rule LIMSEQ-SUP)

      moreover have (SUP n. ?M (F n)) = top
      proof (rule ennreal-SUP-eq-top)
```

```
      fix n :: nat show ∃ k::nat∈UNIV. of-nat n ≤ ?M (F k)
      proof (induct n)
        case (Suc n)
        then guess k .. note k = this
         moreover have finite (F k) ⟹ finite (F (f k)) ⟹ card (F k) < card
(F (f k))
          using ‹F k ⊂ F (f k)› by (simp add: psubset-card-mono)
        moreover have finite (F (f k)) ⟹ finite (F k)
          using ‹k ≤ f k› ‹incseq F› by (auto simp: incseq-def dest: finite-subset)
        ultimately show ?case
          by (auto intro!: exI[of - f k] simp del: of-nat-Suc)
      qed auto
    qed

    moreover
    have inj (λn. F ((f ^^ n) 0))
      by (intro strict-mono-imp-inj-on strict-monoI-Suc) (simp add: ∗)
    then have 1: infinite (range (λi. F ((f ^^ i) 0)))
      by (rule range-inj-infinite)
    have infinite (Pow (⋃ i. F i))
      by (rule infinite-super[OF - 1]) auto
    then have infinite (⋃ i. F i)
      by auto

    ultimately show ?thesis by auto
  qed
 qed
qed

lemma distr-bij-count-space:
 assumes f: bij-betw f A B
 shows distr (count-space A) (count-space B) f = count-space B
proof (rule measure-eqI)
 have f': f ∈ measurable (count-space A) (count-space B)
   using f unfolding Pi-def bij-betw-def by auto
 fix X assume X ∈ sets (distr (count-space A) (count-space B) f)
 then have X: X ∈ sets (count-space B) by auto
 moreover then have f -‘ X ∩ A = the-inv-into A f ‘ X
   using f by (auto simp: bij-betw-def subset-image-iff image-iff the-inv-into-f-f
intro: the-inv-into-f-f[symmetric])
 moreover have inj-on (the-inv-into A f) B
   using X f by (auto simp: bij-betw-def inj-on-the-inv-into)
 with X have inj-on (the-inv-into A f) X
   by (auto intro: subset-inj-on)
 ultimately show emeasure (distr (count-space A) (count-space B) f) X = emea-
sure (count-space B) X
   using f unfolding emeasure-distr[OF f' X]
  by (subst (1 2) emeasure-count-space) (auto simp: card-image dest: finite-imageD)
qed simp
```

lemma *emeasure-count-space-finite*[*simp*]:
  $X \subseteq A \Longrightarrow$ *finite* $X \Longrightarrow$ *emeasure* (*count-space* A) X = *of-nat* (*card* X)
  using *emeasure-count-space*[*of* X A] **by** *simp*

lemma *emeasure-count-space-infinite*[*simp*]:
  $X \subseteq A \Longrightarrow$ *infinite* $X \Longrightarrow$ *emeasure* (*count-space* A) $X = \infty$
  using *emeasure-count-space*[*of* X A] **by** *simp*

lemma *measure-count-space*: *measure* (*count-space* A) X = (*if* $X \subseteq A$ *then of-nat* (*card* X) *else* 0)
  by (*cases finite* X) (*auto simp*: *measure-notin-sets ennreal-of-nat-eq-real-of-nat measure-zero-top measure-eq-emeasure-eq-ennreal*)

lemma *emeasure-count-space-eq-0*:
  emeasure (*count-space* A) $X = 0 \longleftrightarrow (X \subseteq A \longrightarrow X = \{\})$
proof *cases*
  assume X: $X \subseteq A$
  then show *?thesis*
  proof (*intro iffI impI*)
    assume *emeasure* (*count-space* A) $X = 0$
    with X **show** $X = \{\}$
      by (*subst* (*asm*) *emeasure-count-space*) (*auto split*: *if-split-asm*)
  qed *simp*
qed (*simp add*: *emeasure-notin-sets*)

lemma *space-empty*: *space* $M = \{\} \Longrightarrow M$ = *count-space* $\{\}$
  by (*rule measure-eqI*) (*simp-all add*: *space-empty-iff*)

lemma *null-sets-count-space*: *null-sets* (*count-space* A) = $\{ \{\} \}$
  unfolding *null-sets-def* **by** (*auto simp add*: *emeasure-count-space-eq-0*)

lemma *AE-count-space*: (*AE* x *in count-space* A. $P\ x$) $\longleftrightarrow$ ($\forall x{\in}A$. $P\ x$)
  unfolding *eventually-ae-filter* **by** (*auto simp add*: *null-sets-count-space*)

lemma *sigma-finite-measure-count-space-countable*:
  assumes A: *countable* A
  shows *sigma-finite-measure* (*count-space* A)
  proof qed (*insert* A, *auto intro*!: *exI*[*of* - (λa. $\{a\}$) ` A])

lemma *sigma-finite-measure-count-space*:
  fixes A :: 'a::*countable set* **shows** *sigma-finite-measure* (*count-space* A)
  by (*rule sigma-finite-measure-count-space-countable*) *auto*

lemma *finite-measure-count-space*:
  assumes [*simp*]: *finite* A
  shows *finite-measure* (*count-space* A)
  by *rule simp*

lemma *sigma-finite-measure-count-space-finite*:
assumes *A*: *finite A* **shows** *sigma-finite-measure* (*count-space A*)
proof −
interpret *finite-measure count-space A* **using** *A* **by** (*rule finite-measure-count-space*)
show *sigma-finite-measure* (*count-space A*) **..**
qed

3.12 Measure restricted to space

lemma *emeasure-restrict-space*:
assumes Ω ∩ *space M* ∈ *sets M A* ⊆ Ω
shows *emeasure* (*restrict-space M* Ω) *A* = *emeasure M A*
proof *cases*
assume *A* ∈ *sets M*
show *?thesis*
proof (*rule emeasure-measure-of*[*OF restrict-space-def*])
show *op* ∩ Ω ' *sets M* ⊆ *Pow* (Ω ∩ *space M*) *A* ∈ *sets* (*restrict-space M* Ω)
using ‹*A* ⊆ Ω› ‹*A* ∈ *sets M*› *sets.space-closed* **by** (*auto simp*: *sets-restrict-space*)
show *positive* (*sets* (*restrict-space M* Ω)) (*emeasure M*)
by (*auto simp*: *positive-def*)
show *countably-additive* (*sets* (*restrict-space M* Ω)) (*emeasure M*)
proof (*rule countably-additiveI*)
fix *A* :: *nat* ⇒ - **assume** *range A* ⊆ *sets* (*restrict-space M* Ω) *disjoint-family A*
with *assms* **have** ⋀*i*. *A i* ∈ *sets M* ⋀*i*. *A i* ⊆ *space M disjoint-family A*
by (*fastforce simp*: *sets-restrict-space-iff*[*OF assms*(*1*)] *image-subset-iff*
dest: *sets.sets-into-space*)+
then show (∑ *i*. *emeasure M* (*A i*)) = *emeasure M* (⋃ *i*. *A i*)
by (*subst suminf-emeasure*) (*auto simp*: *disjoint-family-subset*)
qed
qed
next
assume *A* ∉ *sets M*
moreover with *assms* **have** *A* ∉ *sets* (*restrict-space M* Ω)
by (*simp add*: *sets-restrict-space-iff*)
ultimately show *?thesis*
by (*simp add*: *emeasure-notin-sets*)
qed

lemma *measure-restrict-space*:
assumes Ω ∩ *space M* ∈ *sets M A* ⊆ Ω
shows *measure* (*restrict-space M* Ω) *A* = *measure M A*
using *emeasure-restrict-space*[*OF assms*] **by** (*simp add*: *measure-def*)

lemma *AE-restrict-space-iff*:
assumes Ω ∩ *space M* ∈ *sets M*
shows (*AE x in restrict-space M* Ω. *P x*) ⟷ (*AE x in M*. *x* ∈ Ω ⟶ *P x*)
proof −
have *ex-cong*: ⋀*P Q f*. (⋀*x*. *P x* ⟹ *Q x*) ⟹ (⋀*x*. *Q x* ⟹ *P* (*f x*)) ⟹ (∃ *x*.

```
P x) ⟷ (∃ x. Q x)
    by auto
  { fix X assume X: X ∈ sets M emeasure M X = 0
    then have emeasure M (Ω ∩ space M ∩ X) ≤ emeasure M X
      by (intro emeasure-mono) auto
    then have emeasure M (Ω ∩ space M ∩ X) = 0
      using X by (auto intro!: antisym) }
  with assms show ?thesis
    unfolding eventually-ae-filter
    by (auto simp add: space-restrict-space null-sets-def sets-restrict-space-iff
                       emeasure-restrict-space cong: conj-cong
             intro!: ex-cong[where f=λX. (Ω ∩ space M) ∩ X])
qed

lemma restrict-restrict-space:
  assumes A ∩ space M ∈ sets M B ∩ space M ∈ sets M
  shows restrict-space (restrict-space M A) B = restrict-space M (A ∩ B) (is ?l
= ?r)
proof (rule measure-eqI[symmetric])
  show sets ?r = sets ?l
    unfolding sets-restrict-space image-comp by (intro image-cong) auto
next
  fix X assume X ∈ sets (restrict-space M (A ∩ B))
  then obtain Y where Y ∈ sets M X = Y ∩ A ∩ B
    by (auto simp: sets-restrict-space)
  with assms sets.Int[OF assms] show emeasure ?r X = emeasure ?l X
    by (subst (1 2) emeasure-restrict-space)
       (auto simp: space-restrict-space sets-restrict-space-iff emeasure-restrict-space
ac-simps)
qed

lemma restrict-count-space: restrict-space (count-space B) A = count-space (A ∩
B)
proof (rule measure-eqI)
  show sets (restrict-space (count-space B) A) = sets (count-space (A ∩ B))
    by (subst sets-restrict-space) auto
  moreover fix X assume X ∈ sets (restrict-space (count-space B) A)
  ultimately have X ⊆ A ∩ B by auto
 then show emeasure (restrict-space (count-space B) A) X = emeasure (count-space
(A ∩ B)) X
    by (cases finite X) (auto simp add: emeasure-restrict-space)
qed

lemma sigma-finite-measure-restrict-space:
  assumes sigma-finite-measure M
  and A: A ∈ sets M
  shows sigma-finite-measure (restrict-space M A)
proof −
  interpret sigma-finite-measure M by fact
```

from *sigma-finite-countable* **obtain** C
where C: *countable* C $C \subseteq$ *sets* M $(\bigcup C) =$ *space* M $\forall a{\in}C.$ *emeasure* M $a \neq \infty$
by *blast*
let *?C* $=$ *op* $\cap$ A ' C
from C **have** *countable ?C ?C* $\subseteq$ *sets* (*restrict-space* M A) $(\bigcup ?C) =$ *space* (*restrict-space* M A)
by(*auto simp add*: *sets-restrict-space space-restrict-space*)
moreover {
fix a
assume $a \in ?C$
then obtain a' **where** $a = A \cap a'$ $a' \in C$ **..**
then have *emeasure* (*restrict-space* M A) $a \leq$ *emeasure* M a'
using A C **by**(*auto simp add*: *emeasure-restrict-space intro*: *emeasure-mono*)
also have $\ldots < \infty$ **using** $C(4)$[*rule-format*, *of* a'] ‹$a' \in C$› **by** (*simp add*: *less-top*)
finally have *emeasure* (*restrict-space* M A) $a \neq \infty$ **by** *simp* **}**
ultimately show *?thesis*
by *unfold-locales* (*rule exI conjI*|*assumption*|*blast*)+
qed

lemma *finite-measure-restrict-space*:
assumes *finite-measure* M
and A: $A \in$ *sets* M
shows *finite-measure* (*restrict-space* M A)
proof −
interpret *finite-measure* M **by** *fact*
show *?thesis*
by(*rule finite-measureI*)(*simp add*: *emeasure-restrict-space A space-restrict-space*)
qed

lemma *restrict-distr*:
assumes [*measurable*]: $f \in$ *measurable* M N
assumes [*simp*]: $\Omega \cap$ *space* $N \in$ *sets* N **and** *restrict*: $f \in$ *space* $M \to \Omega$
shows *restrict-space* (*distr* M N f) $\Omega =$ *distr* M (*restrict-space* N Ω) f
(**is** *?l* $=$ *?r*)
proof (*rule measure-eqI*)
fix A **assume** $A \in$ *sets* (*restrict-space* (*distr* M N f) Ω)
with *restrict* **show** *emeasure ?l* $A =$ *emeasure ?r* A
by (*subst emeasure-distr*)
(*auto simp*: *sets-restrict-space-iff emeasure-restrict-space emeasure-distr*
intro!: *measurable-restrict-space2*)
qed (*simp add*: *sets-restrict-space*)

lemma *measure-eqI-restrict-generator*:
assumes E: *Int-stable* E $E \subseteq$ *Pow* Ω $\bigwedge X.\ X \in E \Longrightarrow$ *emeasure* M $X =$ *emeasure* N X
assumes *sets-eq*: *sets* $M =$ *sets* N **and** Ω: $\Omega \in$ *sets* M
assumes *sets* (*restrict-space* M Ω) $=$ *sigma-sets* Ω E

assumes *sets* (*restrict-space N* Ω) = *sigma-sets* Ω *E*
assumes *ae*: *AE x in M*. *x* ∈ Ω *AE x in N*. *x* ∈ Ω
assumes *A*: *countable A A* ≠ {} *A* ⊆ *E* ⋃ *A* = Ω ⋀*a*. *a* ∈ *A* ⟹ *emeasure M a* ≠ ∞
shows *M* = *N*
proof (*rule measure-eqI*)
fix *X* **assume** *X*: *X* ∈ *sets M*
then have *emeasure M X* = *emeasure* (*restrict-space M* Ω) (*X* ∩ Ω)
using *ae* Ω **by** (*auto simp add*: *emeasure-restrict-space intro*!: *emeasure-eq-AE*)
also have *restrict-space M* Ω = *restrict-space N* Ω
proof (*rule measure-eqI-generator-eq*)
fix *X* **assume** *X* ∈ *E*
then show *emeasure* (*restrict-space M* Ω) *X* = *emeasure* (*restrict-space N* Ω) *X*
using *E* Ω **by** (*subst* (*1 2*) *emeasure-restrict-space*) (*auto simp*: *sets-eq sets-eq*[*THEN sets-eq-imp-space-eq*])
next
show *range* (*from-nat-into A*) ⊆ *E* (⋃ *i*. *from-nat-into A i*) = Ω
using *A* **by** (*auto cong del*: *strong-SUP-cong*)
next
fix *i*
have *emeasure* (*restrict-space M* Ω) (*from-nat-into A i*) = *emeasure M* (*from-nat-into A i*)
using *A* Ω **by** (*subst emeasure-restrict-space*)
(*auto simp*: *sets-eq sets-eq*[*THEN sets-eq-imp-space-eq*] *intro*: *from-nat-into*)
with *A* **show** *emeasure* (*restrict-space M* Ω) (*from-nat-into A i*) ≠ ∞
by (*auto intro*: *from-nat-into*)
qed *fact*+
also have *emeasure* (*restrict-space N* Ω) (*X* ∩ Ω) = *emeasure N X*
using *X ae* Ω **by** (*auto simp add*: *emeasure-restrict-space sets-eq intro*!: *emeasure-eq-AE*)
finally show *emeasure M X* = *emeasure N X* **.**
qed *fact*

3.13 Null measure

definition *null-measure M* = *sigma* (*space M*) (*sets M*)

lemma *space-null-measure*[*simp*]: *space* (*null-measure M*) = *space M*
by (*simp add*: *null-measure-def*)

lemma *sets-null-measure*[*simp*, *measurable-cong*]: *sets* (*null-measure M*) = *sets M*
by (*simp add*: *null-measure-def*)

lemma *emeasure-null-measure*[*simp*]: *emeasure* (*null-measure M*) *X* = *0*
by (*cases X* ∈ *sets M*, *rule emeasure-measure-of*)
(*auto simp*: *positive-def countably-additive-def emeasure-notin-sets null-measure-def*
dest: *sets.sets-into-space*)

lemma *measure-null-measure*[*simp*]: *measure* (*null-measure M*) *X* = *0*
by (*intro measure-eq-emeasure-eq-ennreal*) *auto*

lemma *null-measure-idem* [*simp*]: *null-measure* (*null-measure M*) = *null-measure M*
by(*rule measure-eqI*) *simp-all*

3.14 Scaling a measure

definition *scale-measure* :: *ennreal* ⇒ *'a measure* ⇒ *'a measure*
where
scale-measure r M = *measure-of* (*space M*) (*sets M*) (λ*A*. *r* ∗ *emeasure M A*)

lemma *space-scale-measure*: *space* (*scale-measure r M*) = *space M*
by (*simp add*: *scale-measure-def*)

lemma *sets-scale-measure* [*simp*, *measurable-cong*]: *sets* (*scale-measure r M*) = *sets M*
by (*simp add*: *scale-measure-def*)

lemma *emeasure-scale-measure* [*simp*]:
emeasure (*scale-measure r M*) *A* = *r* ∗ *emeasure M A*
(**is** - = *?μ A*)
proof(*cases A* ∈ *sets M*)
case *True*
show *?thesis* **unfolding** *scale-measure-def*
proof(*rule emeasure-measure-of-sigma*)
show *sigma-algebra* (*space M*) (*sets M*) **..**
show *positive* (*sets M*) *?μ* **by** (*simp add*: *positive-def*)
show *countably-additive* (*sets M*) *?μ*
proof (*rule countably-additiveI*)
fix *A* :: *nat* ⇒ - **assume** ∗: *range A* ⊆ *sets M disjoint-family A*
have (∑ *i*. *?μ* (*A i*)) = *r* ∗ (∑ *i*. *emeasure M* (*A i*))
by *simp*
also have ... = *?μ* (⋃ *i*. *A i*) **using** ∗ **by**(*simp add*: *suminf-emeasure*)
finally show (∑ *i*. *?μ* (*A i*)) = *?μ* (⋃ *i*. *A i*) **.**
qed
qed(*fact True*)
qed(*simp add*: *emeasure-notin-sets*)

lemma *scale-measure-1* [*simp*]: *scale-measure 1 M* = *M*
by(*rule measure-eqI*) *simp-all*

lemma *scale-measure-0*[*simp*]: *scale-measure 0 M* = *null-measure M*
by(*rule measure-eqI*) *simp-all*

lemma *measure-scale-measure* [*simp*]: *0* ≤ *r* ⟹ *measure* (*scale-measure r M*) *A* = *r* ∗ *measure M A*
using *emeasure-scale-measure*[*of r M A*]

emeasure-eq-ennreal-measure[*of M A*]
measure-eq-emeasure-eq-ennreal[*of - scale-measure r M A*]
by (*cases emeasure* (*scale-measure r M*) *A* = *top*)
(*auto simp del*: *emeasure-scale-measure*
simp: *ennreal-top-eq-mult-iff ennreal-mult-eq-top-iff measure-zero-top ennreal-mult*[*symmetric*])

lemma *scale-scale-measure* [*simp*]:
scale-measure r (*scale-measure r′ M*) = *scale-measure* (*r* ∗ *r′*) *M*
by (*rule measure-eqI*) (*simp-all add*: *max-def mult.assoc*)

lemma *scale-null-measure* [*simp*]: *scale-measure r* (*null-measure M*) = *null-measure M*
by (*rule measure-eqI*) *simp-all*

3.15 Measures form a chain-complete partial order

instantiation *measure* :: (*type*) *order-bot*
begin

definition *bot-measure* :: *′a measure* **where**
bot-measure = *sigma* {} {{}}

lemma *space-bot*[*simp*]: *space bot* = {}
unfolding *bot-measure-def* **by** (*rule space-measure-of*) *auto*

lemma *sets-bot*[*simp*]: *sets bot* = {{}}
unfolding *bot-measure-def* **by** (*subst sets-measure-of*) *auto*

lemma *emeasure-bot*[*simp*]: *emeasure bot* = (λ*x*. *0*)
unfolding *bot-measure-def* **by** (*rule emeasure-sigma*)

inductive *less-eq-measure* :: *′a measure* ⇒ *′a measure* ⇒ *bool* **where**
sets N = *sets M* ⟹ (⋀*A*. *A* ∈ *sets M* ⟹ *emeasure M A* ≤ *emeasure N A*) ⟹ *less-eq-measure M N*
| *less-eq-measure bot N*

definition *less-measure* :: *′a measure* ⇒ *′a measure* ⇒ *bool* **where**
less-measure M N ⟷ (*M* ≤ *N* ∧ ¬ *N* ≤ *M*)

instance
proof (*standard*, *goal-cases*)
case *1* **then show** *?case*
unfolding *less-measure-def* **..**
next
case (*2 M*) **then show** *?case*
by (*intro less-eq-measure.intros*) *auto*
next
case (*3 M N L*) **then show** *?case*

```
    apply (safe elim!: less-eq-measure.cases)
    apply (simp-all add: less-eq-measure.intros)
    apply (rule less-eq-measure.intros)
    apply simp
    apply (blast intro: order-trans) []
    unfolding less-eq-measure.simps
    apply (rule disjI2)
    apply simp
    apply (rule measure-eqI)
    apply (auto intro!: antisym)
    done
next
  case (4 M N) then show ?case
    apply (safe elim!: less-eq-measure.cases intro!: measure-eqI)
    apply simp
    apply simp
    apply (blast intro: antisym)
    apply (simp)
    apply simp
    done
qed (rule less-eq-measure.intros)
end

lemma le-emeasureD: M ≤ N ⟹ emeasure M A ≤ emeasure N A
  by (cases A ∈ sets M) (auto elim!: less-eq-measure.cases simp: emeasure-notin-sets)

lemma le-sets: N ≤ M ⟹ sets N ≤ sets M
  unfolding less-eq-measure.simps by auto

instantiation measure :: (type) ccpo
begin

definition Sup-measure :: 'a measure set ⇒ 'a measure where
  Sup-measure A = measure-of (SUP a:A. space a) (SUP a:A. sets a) (SUP a:A.
emeasure a)

lemma
  assumes A: Complete-Partial-Order.chain op ≤ A and a: a ≠ bot a ∈ A
  shows space-Sup: space (Sup A) = space a
    and sets-Sup: sets (Sup A) = sets a
proof −
  have sets: (SUP a:A. sets a) = sets a
  proof (intro antisym SUP-least)
    fix a' show a' ∈ A ⟹ sets a' ⊆ sets a
      using a chainD[OF A, of a a'] by (auto elim!: less-eq-measure.cases)
  qed (insert ‹a∈A›, auto)
  have space: (SUP a:A. space a) = space a
  proof (intro antisym SUP-least)
    fix a' show a' ∈ A ⟹ space a' ⊆ space a
```

using *a chainD*[*OF A*, *of a a'*] **by** (*intro sets-le-imp-space-le*) (*auto elim*!: *less-eq-measure.cases*)
qed (*insert* ‹*a*∈*A*›, *auto*)
show *space* (*Sup A*) = *space a*
unfolding *Sup-measure-def sets space sets.space-measure-of-eq* **..**
show *sets* (*Sup A*) = *sets a*
unfolding *Sup-measure-def sets space sets.sets-measure-of-eq* **..**
qed

lemma *emeasure-Sup*:
assumes *A*: *Complete-Partial-Order.chain op* ≤ *A A* ≠ {}
assumes *X* ∈ *sets* (*Sup A*)
shows *emeasure* (*Sup A*) *X* = (*SUP a*:*A*. *emeasure a*) *X*
proof (*rule emeasure-measure-of*[*OF Sup-measure-def*])
show *countably-additive* (*sets* (*Sup A*)) (*SUP a*:*A*. *emeasure a*)
unfolding *countably-additive-def*
proof *safe*
fix *F* :: *nat* ⇒ *'a set* **assume** *F*: *range F* ⊆ *sets* (*Sup A*) *disjoint-family F*
show (∑ *i*. (*SUP a*:*A*. *emeasure a*) (*F i*)) = *SUPREMUM A emeasure* (*UNION UNIV F*)
unfolding *SUP-apply*
proof (*subst ennreal-suminf-SUP-eq-directed*)
fix *N i j* **assume** *i* ∈ *A j* ∈ *A*
with *A*(*1*)
show ∃ *k*∈*A*. ∀ *n*∈*N*. *emeasure i* (*F n*) ≤ *emeasure k* (*F n*) ∧ *emeasure j* (*F n*) ≤ *emeasure k* (*F n*)
by (*blast elim*: *chainE dest*: *le-emeasureD*)
next
show (*SUP n*:*A*. ∑ *i*. *emeasure n* (*F i*)) = (*SUP y*:*A*. *emeasure y* (*UNION UNIV F*))
proof (*intro SUP-cong refl*)
fix *a* **assume** *a* ∈ *A* **then show** (∑ *i*. *emeasure a* (*F i*)) = *emeasure a* (*UNION UNIV F*)
using *sets-Sup*[*OF A*(*1*), *of a*] *F* **by** (*cases a* = *bot*) (*auto simp*: *suminf-emeasure*)
qed
qed
qed
qed (*insert* ‹*A* ≠ {}› ‹*X* ∈ *sets* (*Sup A*)›, *auto simp*: *positive-def dest*: *sets.sets-into-space intro*: *SUP-upper2*)

instance
proof
fix *A* **and** *x* :: *'a measure* **assume** *A*: *Complete-Partial-Order.chain op* ≤ *A* **and** *x*: *x* ∈ *A*
show *x* ≤ *Sup A*
proof *cases*
assume *x* ≠ *bot*
show *?thesis*

```
    proof
      show sets (Sup A) = sets x
        using A ‹x ≠ bot› x by (rule sets-Sup)
      with x show ⋀a. a ∈ sets x ⟹ emeasure x a ≤ emeasure (Sup A) a
        by (subst emeasure-Sup[OF A]) (auto intro: SUP-upper)
    qed
  qed simp
next
  fix A and x :: 'a measure assume A: Complete-Partial-Order.chain op ≤ A and
x: ⋀z. z ∈ A ⟹ z ≤ x
  consider A = {} | A = {bot} | x where x∈A x ≠ bot
    by blast
  then show Sup A ≤ x
  proof cases
    assume A = {}
    moreover have Sup ({}::'a measure set) = bot
      by (auto simp add: Sup-measure-def sigma-sets-empty-eq intro!: measure-eqI)
    ultimately show ?thesis
      by simp
  next
    assume A = {bot}
    moreover have Sup ({bot}::'a measure set) = bot
      by (auto simp add: Sup-measure-def sigma-sets-empty-eq intro!: measure-eqI)
     ultimately show ?thesis
      by simp
  next
    fix a assume a ∈ A a ≠ bot
    then have a ≤ x x ≠ bot a ≠ bot
      using x[OF ‹a ∈ A›] by (auto simp: bot-unique)
    then have sets x = sets a
      by (auto elim: less-eq-measure.cases)

    show Sup A ≤ x
    proof (rule less-eq-measure.intros)
      show sets x = sets (Sup A)
        by (subst sets-Sup[OF A ‹a ≠ bot› ‹a ∈ A›]) fact
    next
      fix X assume X ∈ sets (Sup A)
      then have emeasure (Sup A) X ≤ (SUP a:A. emeasure a X)
        using ‹a∈A› by (subst emeasure-Sup[OF A -]) auto
      also have ... ≤ emeasure x X
        by (intro SUP-least le-emeasureD x)
      finally show emeasure (Sup A) X ≤ emeasure x X .
    qed
  qed
qed
end

lemma
```

assumes *A*: *Complete-Partial-Order.chain op* $\leq$ (*f* ' *A*) **and** *a*: *a* $\in$ *A f a* $\neq$ *bot*
shows *space-SUP*: *space* (*SUP M*:*A*. *f M*) = *space* (*f a*)
and *sets-SUP*: *sets* (*SUP M*:*A*. *f M*) = *sets* (*f a*)
by(*rule space-Sup*[*OF A a*(*2*) *imageI*[*OF a*(*1*)]] *sets-Sup*[*OF A a*(*2*) *imageI*[*OF a*(*1*)]])+

lemma *emeasure-SUP*:
assumes *A*: *Complete-Partial-Order.chain op* $\leq$ (*f* ' *A*) *A* $\neq$ {}
assumes *X* $\in$ *sets* (*SUP M*:*A*. *f M*)
shows *emeasure* (*SUP M*:*A*. *f M*) *X* = (*SUP M*:*A*. *emeasure* (*f M*)) *X*
using ‹*X* $\in$ -› **by**(*subst emeasure-Sup*[*OF A*(*1*)]; *simp add*: *A*)

end

4 Borel spaces

theory *Borel-Space*
imports
Measurable
~~/*src*/*HOL*/*Multivariate-Analysis*/*Multivariate-Analysis*
begin

lemma *sets-Collect-eventually-sequentially*[*measurable*]:
($\bigwedge$*i*. {*x*$\in$*space M*. *P x i*} $\in$ *sets M*) $\Longrightarrow$ {*x*$\in$*space M*. *eventually* (*P x*) *sequentially*} $\in$ *sets M*
unfolding *eventually-sequentially* **by** *simp*

lemma *open-Collect-less*:
fixes *f g* :: '*i*::*topological-space* $\Rightarrow$ '*a* :: {*dense-linorder*, *linorder-topology*}
assumes *continuous-on UNIV f*
assumes *continuous-on UNIV g*
shows *open* {*x*. *f x* < *g x*}
proof −
have *open* ($\bigcup$ *y*. {*x* $\in$ *UNIV*. *f x* $\in$ {..< *y*}} $\cap$ {*x* $\in$ *UNIV*. *g x* $\in$ {*y* <..}}) (**is** *open ?X*)
by (*intro open-UN ballI open-Int continuous-open-preimage assms*) *auto*
also have *?X* = {*x*. *f x* < *g x*}
by (*auto intro*: *dense*)
finally show *?thesis* **.**
qed

lemma *closed-Collect-le*:
fixes *f g* :: '*i*::*topological-space* $\Rightarrow$ '*a* :: {*dense-linorder*, *linorder-topology*}
assumes *f*: *continuous-on UNIV f*
assumes *g*: *continuous-on UNIV g*
shows *closed* {*x*. *f x* $\leq$ *g x*}
using *open-Collect-less*[*OF g f*] **unfolding** *not-less*[*symmetric*] *Collect-neg-eq open-closed* **.**

lemma *topological-basis-trivial*: *topological-basis* {*A*. *open A*}
by (*auto simp*: *topological-basis-def*)

lemma *open-prod-generated*: *open* = *generate-topology* {*A* × *B* | *A B*. *open A* ∧ *open B*}
proof −
have {*A* × *B* :: ('*a* × '*b*) *set* | *A B*. *open A* ∧ *open B*} = ((λ(*a*, *b*). *a* × *b*) ' ({*A*. *open A*} × {*A*. *open A*}))
by *auto*
then show *?thesis*
by (*auto intro*: *topological-basis-prod topological-basis-trivial topological-basis-imp-subbasis*)
qed

definition *mono-on f A* ≡ ∀ *r s*. *r* ∈ *A* ∧ *s* ∈ *A* ∧ *r* ≤ *s* ⟶ *f r* ≤ *f s*

lemma *mono-onI*:
(⋀*r s*. *r* ∈ *A* ⟹ *s* ∈ *A* ⟹ *r* ≤ *s* ⟹ *f r* ≤ *f s*) ⟹ *mono-on f A*
unfolding *mono-on-def* **by** *simp*

lemma *mono-onD*:
⟦*mono-on f A*; *r* ∈ *A*; *s* ∈ *A*; *r* ≤ *s*⟧ ⟹ *f r* ≤ *f s*
unfolding *mono-on-def* **by** *simp*

lemma *mono-imp-mono-on*: *mono f* ⟹ *mono-on f A*
unfolding *mono-def mono-on-def* **by** *auto*

lemma *mono-on-subset*: *mono-on f A* ⟹ *B* ⊆ *A* ⟹ *mono-on f B*
unfolding *mono-on-def* **by** *auto*

definition *strict-mono-on f A* ≡ ∀ *r s*. *r* ∈ *A* ∧ *s* ∈ *A* ∧ *r* < *s* ⟶ *f r* < *f s*

lemma *strict-mono-onI*:
(⋀*r s*. *r* ∈ *A* ⟹ *s* ∈ *A* ⟹ *r* < *s* ⟹ *f r* < *f s*) ⟹ *strict-mono-on f A*
unfolding *strict-mono-on-def* **by** *simp*

lemma *strict-mono-onD*:
⟦*strict-mono-on f A*; *r* ∈ *A*; *s* ∈ *A*; *r* < *s*⟧ ⟹ *f r* < *f s*
unfolding *strict-mono-on-def* **by** *simp*

lemma *mono-on-greaterD*:
assumes *mono-on g A x* ∈ *A y* ∈ *A g x* > (*g* (*y*::-::*linorder*) :: - :: *linorder*)
shows *x* > *y*
proof (*rule ccontr*)
assume ¬*x* > *y*
hence *x* ≤ *y* **by** (*simp add*: *not-less*)
from *assms*(*1*−*3*) **and** *this* **have** *g x* ≤ *g y* **by** (*rule mono-onD*)
with *assms*(*4*) **show** *False* **by** *simp*
qed

lemma *strict-mono-inv*:
fixes $f :: ('a::linorder) \Rightarrow ('b::linorder)$
assumes *strict-mono f* **and** *surj f* **and** *inv*: $\bigwedge x.\ g\ (f\ x) = x$
shows *strict-mono g*
proof
fix $x\ y :: 'b$ **assume** $x < y$
from ‹*surj f*› **obtain** $x'\ y'$ **where** [*simp*]: $x = f\ x'\ y = f\ y'$ **by** *blast*
with ‹$x < y$› **and** ‹*strict-mono f*› **have** $x' < y'$ **by** (*simp add*: *strict-mono-less*)
with *inv* **show** $g\ x < g\ y$ **by** *simp*
qed

lemma *strict-mono-on-imp-inj-on*:
assumes *strict-mono-on* $(f :: (- :: linorder) \Rightarrow (- :: preorder))\ A$
shows *inj-on f A*
proof (*rule inj-onI*)
fix $x\ y$ **assume** $x \in A\ y \in A\ f\ x = f\ y$
thus $x = y$
by (*cases x y rule*: *linorder-cases*)
(*auto dest*: *strict-mono-onD*[*OF assms*, *of x y*] *strict-mono-onD*[*OF assms*, *of y x*])
qed

lemma *strict-mono-on-leD*:
assumes *strict-mono-on* $(f :: (- :: linorder) \Rightarrow - :: preorder)\ A\ x \in A\ y \in A\ x \leq y$
shows $f\ x \leq f\ y$
proof (*insert le-less-linear*[*of y x*], *elim disjE*)
assume $x < y$
with *assms* **have** $f\ x < f\ y$ **by** (*rule-tac strict-mono-onD*[*OF assms*(*1*)]) *simp-all*
thus *?thesis* **by** (*rule less-imp-le*)
qed (*insert assms*, *simp*)

lemma *strict-mono-on-eqD*:
fixes $f :: (- :: linorder) \Rightarrow (- :: preorder)$
assumes *strict-mono-on f A* $f\ x = f\ y\ x \in A\ y \in A$
shows $y = x$
using *assms* **by** (*rule-tac linorder-cases*[*of x y*]) (*auto dest*: *strict-mono-onD*)

lemma *mono-on-imp-deriv-nonneg*:
assumes *mono*: *mono-on f A* **and** *deriv*: (*f has-real-derivative D*) (*at x*)
assumes $x \in$ *interior A*
shows $D \geq 0$
proof (*rule tendsto-le-const*)
let *?A'* $= (\lambda y.\ y - x)$ ‘ *interior A*
from *deriv* **show** $((\lambda h.\ (f\ (x + h) - f\ x)\ /\ h) \longrightarrow D)\ (at\ 0)$
by (*simp add*: *field-has-derivative-at has-field-derivative-def*)
from *mono* **have** *mono'*: *mono-on f* (*interior A*) **by** (*rule mono-on-subset*) (*rule interior-subset*)

```
  show eventually (λh. (f (x + h) − f x) / h ≥ 0) (at 0)
  proof (subst eventually-at-topological, intro exI conjI ballI impI)
    have open (interior A) by simp
    hence open (op + (−x) ‘ interior A) by (rule open-translation)
    also have (op + (−x) ‘ interior A) = ?A′ by auto
    finally show open ?A′ .
  next
    from ‹x ∈ interior A› show 0 ∈ ?A′ by auto
  next
    fix h assume h ∈ ?A′
    hence x + h ∈ interior A by auto
    with mono′ and ‹x ∈ interior A› show (f (x + h) − f x) / h ≥ 0
      by (cases h rule: linorder-cases[of - 0])
         (simp-all add: divide-nonpos-neg divide-nonneg-pos mono-onD field-simps)
  qed
qed simp

lemma strict-mono-on-imp-mono-on:
  strict-mono-on (f :: (- :: linorder) ⇒ - :: preorder) A ⟹ mono-on f A
  by (rule mono-onI, rule strict-mono-on-leD)

lemma mono-on-ctble-discont:
  fixes f :: real ⇒ real
  fixes A :: real set
  assumes mono-on f A
  shows countable {a∈A. ¬ continuous (at a within A) f}
proof −
  have mono: ⋀x y. x ∈ A ⟹ y ∈ A ⟹ x ≤ y ⟹ f x ≤ f y
    using ‹mono-on f A› by (simp add: mono-on-def)
  have ∀ a ∈ {a∈A. ¬ continuous (at a within A) f}. ∃ q :: nat × rat.
      (fst q = 0 ∧ of-rat (snd q) < f a ∧ (∀ x ∈ A. x < a ⟶ f x < of-rat (snd
q))) ∨
      (fst q = 1 ∧ of-rat (snd q) > f a ∧ (∀ x ∈ A. x > a ⟶ f x > of-rat (snd
q)))
  proof (clarsimp simp del: One-nat-def)
    fix a assume a ∈ A assume ¬ continuous (at a within A) f
    thus ∃ q1 q2.
          q1 = 0 ∧ real-of-rat q2 < f a ∧ (∀ x∈A. x < a ⟶ f x < real-of-rat q2)
∨
          q1 = 1 ∧ f a < real-of-rat q2 ∧ (∀ x∈A. a < x ⟶ real-of-rat q2 < f x)
    proof (auto simp add: continuous-within order-tendsto-iff eventually-at)
      fix l assume l < f a
      then obtain q2 where q2: l < of-rat q2 of-rat q2 < f a
        using of-rat-dense by blast
      assume ∗ [rule-format]: ∀ d>0. ∃ x∈A. x ≠ a ∧ dist x a < d ∧ ¬ l < f x
      from q2 have real-of-rat q2 < f a ∧ (∀ x∈A. x < a ⟶ f x < real-of-rat q2)
      proof auto
        fix x assume x ∈ A x < a
        with q2 ∗[of a − x] show f x < real-of-rat q2
```

```
        apply (auto simp add: dist-real-def not-less)
        apply (subgoal-tac f x ≤ f xa)
        by (auto intro: mono)
      qed
      thus ?thesis by auto
    next
      fix u assume u > f a
      then obtain q2 where q2: f a < of-rat q2 of-rat q2 < u
        using of-rat-dense by blast
      assume *[rule-format]: ∀ d>0. ∃ x∈A. x ≠ a ∧ dist x a < d ∧ ¬ u > f x
      from q2 have real-of-rat q2 > f a ∧ (∀ x∈A. x > a ⟶ f x > real-of-rat q2)
      proof auto
        fix x assume x ∈ A x > a
        with q2 *[of x − a] show f x > real-of-rat q2
          apply (auto simp add: dist-real-def)
          apply (subgoal-tac f x ≥ f xa)
          by (auto intro: mono)
      qed
      thus ?thesis by auto
    qed
  qed
  hence ∃ g :: real ⇒ nat × rat . ∀ a ∈ {a∈A. ¬ continuous (at a within A) f}.
      (fst (g a) = 0 ∧ of-rat (snd (g a)) < f a ∧ (∀ x ∈ A. x < a ⟶ f x < of-rat
(snd (g a)))) |
      (fst (g a) = 1 ∧ of-rat (snd (g a)) > f a ∧ (∀ x ∈ A. x > a ⟶ f x > of-rat
(snd (g a))))
    by (rule bchoice)
  then guess g ..
  hence g: ⋀a x. a ∈ A ⟹ ¬ continuous (at a within A) f ⟹ x ∈ A ⟹
      (fst (g a) = 0 ∧ of-rat (snd (g a)) < f a ∧ (x < a ⟶ f x < of-rat (snd (g
a)))) |
      (fst (g a) = 1 ∧ of-rat (snd (g a)) > f a ∧ (x > a ⟶ f x > of-rat (snd (g
a))))
    by auto
  have inj-on g {a∈A. ¬ continuous (at a within A) f}
  proof (auto simp add: inj-on-def)
    fix w z
    assume 1: w ∈ A and 2: ¬ continuous (at w within A) f and
           3: z ∈ A and 4: ¬ continuous (at z within A) f and
           5: g w = g z
    from g [OF 1 2 3] g [OF 3 4 1] 5
    show w = z by auto
  qed
  thus ?thesis
    by (rule countableI')
qed

lemma mono-on-ctble-discont-open:
  fixes f :: real ⇒ real
```

fixes A :: *real set*
assumes *open A mono-on f A*
shows *countable* $\{a{\in}A.\ \neg isCont\ f\ a\}$
proof −
have $\{a{\in}A.\ \neg isCont\ f\ a\} = \{a{\in}A.\ \neg(continuous\ (at\ a\ within\ A)\ f)\}$
by (*auto simp add*: *continuous-within-open* [*OF* - ‹*open A*›])
thus *?thesis*
apply (*elim ssubst*)
by (*rule mono-on-ctble-discont*, *rule assms*)
qed

lemma *mono-ctble-discont*:
fixes f :: *real* ⇒ *real*
assumes *mono f*
shows *countable* $\{a.\ \neg\ isCont\ f\ a\}$
using *assms mono-on-ctble-discont* [*of f UNIV*] **unfolding** *mono-on-def mono-def*
by *auto*

lemma *has-real-derivative-imp-continuous-on*:
assumes $\bigwedge x.\ x \in A \Longrightarrow (f\ has\text{-}real\text{-}derivative\ f'\ x)\ (at\ x)$
shows *continuous-on A f*
apply (*intro differentiable-imp-continuous-on*, *unfold differentiable-on-def*)
apply (*intro ballI Deriv.differentiableI*)
apply (*rule has-field-derivative-subset*[*OF assms*])
apply *simp-all*
done

lemma *closure-contains-Sup*:
fixes S :: *real set*
assumes $S \neq \{\}$ *bdd-above S*
shows *Sup S* ∈ *closure S*
proof−
have *Inf* (*uminus* ‘ *S*) ∈ *closure* (*uminus* ‘ *S*)
using *assms* **by** (*intro closure-contains-Inf*) *auto*
also have *Inf* (*uminus* ‘ *S*) = −*Sup S* **by** (*simp add*: *Inf-real-def*)
also have *closure* (*uminus* ‘ *S*) = *uminus* ‘ *closure S*
by (*rule sym*, *intro closure-injective-linear-image*) (*auto intro*: *linearI*)
finally show *?thesis* **by** *auto*
qed

lemma *closed-contains-Sup*:
fixes S :: *real set*
shows $S \neq \{\} \Longrightarrow$ *bdd-above S* ⟹ *closed S* ⟹ *Sup S* ∈ *S*
by (*subst closure-closed*[*symmetric*], *assumption*, *rule closure-contains-Sup*)

lemma *deriv-nonneg-imp-mono*:
assumes *deriv*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g\ has\text{-}real\text{-}derivative\ g'\ x)\ (at\ x)$
assumes *nonneg*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g'\ x \geq 0$
assumes *ab*: $a \leq b$

shows $g\ a \leq g\ b$
proof (*cases* $a < b$)
assume $a < b$
from *deriv* **have** $\forall x.\ x \geq a \wedge x \leq b \longrightarrow (g\ has\text{-}real\text{-}derivative\ g'\ x)\ (at\ x)$ **by** *simp*
from *MVT2*[*OF* ‹$a < b$› *this*] **and** *deriv*
obtain ξ **where** ξ-*ab*: $\xi > a\ \xi < b$ **and** *g-ab*: $g\ b - g\ a = (b - a) * g'\ \xi$ **by** *blast*
from ξ-*ab ab nonneg* **have** $(b - a) * g'\ \xi \geq 0$ **by** *simp*
with *g-ab* **show** *?thesis* **by** *simp*
qed (*insert ab*, *simp*)

lemma *continuous-interval-vimage-Int*:
assumes *continuous-on* $\{a::real..b\}\ g$ **and** *mono*: $\bigwedge x\ y.\ a \leq x \Longrightarrow x \leq y \Longrightarrow y \leq b \Longrightarrow g\ x \leq g\ y$
assumes $a \leq b\ (c::real) \leq d\ \{c..d\} \subseteq \{g\ a..g\ b\}$
obtains $c'\ d'$ **where** $\{a..b\} \cap g -` \{c..d\} = \{c'..d'\}\ c' \leq d'\ g\ c' = c\ g\ d' = d$
proof−
let *?A* $= \{a..b\} \cap g -` \{c..d\}$
from *IVT'*[*of g a c b*, *OF - -* ‹$a \leq b$› *assms*(*1*)] *assms*(*4*,*5*)
obtain c'' **where** c'': $c'' \in$ *?A g* $c'' = c$ **by** *auto*
from *IVT'*[*of g a d b*, *OF - -* ‹$a \leq b$› *assms*(*1*)] *assms*(*4*,*5*)
obtain d'' **where** d'': $d'' \in$ *?A g* $d'' = d$ **by** *auto*
hence [*simp*]: *?A* $\neq \{\}$ **by** *blast*

def $c' \equiv$ *Inf ?A* **and** $d' \equiv$ *Sup ?A*
have *?A* $\subseteq \{c'..d'\}$ **unfolding** c'-*def* d'-*def*
by (*intro subsetI*) (*auto intro*: *cInf-lower cSup-upper*)
moreover from *assms* **have** *closed ?A*
using *continuous-on-closed-vimage*[*of* $\{a..b\}\ g$] **by** (*subst Int-commute*) *simp*
hence $c'd'$-*in-set*: $c' \in$ *?A* $d' \in$ *?A* **unfolding** c'-*def* d'-*def*
by ((*intro closed-contains-Inf closed-contains-Sup*, *simp-all*)[])+
hence $\{c'..d'\} \subseteq$ *?A* **using** *assms*
by (*intro subsetI*)
(*auto intro*!: *order-trans*[*of c g c' g x* **for** *x*] *order-trans*[*of g x g d' d* **for** *x*]
intro!: *mono*)
moreover have $c' \leq d'$ **using** $c'd'$-*in-set*(*2*) **unfolding** c'-*def* **by** (*intro cInf-lower*) *auto*
moreover have $g\ c' \leq c\ g\ d' \geq d$
apply (*insert* c'' d'' $c'd'$-*in-set*)
apply (*subst* c''(*2*)[*symmetric*])
apply (*auto simp*: c'-*def intro*!: *mono cInf-lower* c'') []
apply (*subst* d''(*2*)[*symmetric*])
apply (*auto simp*: d'-*def intro*!: *mono cSup-upper* d'') []
done
with $c'd'$-*in-set* **have** $g\ c' = c\ g\ d' = d$ **by** *auto*
ultimately show *?thesis* **using** *that* **by** *blast*

qed

4.1 Generic Borel spaces

definition (**in** *topological-space*) *borel* :: *'a measure* **where**
borel = *sigma UNIV* {*S*. *open S*}

abbreviation *borel-measurable M* ≡ *measurable M borel*

lemma *in-borel-measurable*:
f ∈ *borel-measurable M* ⟷
(∀ *S* ∈ *sigma-sets UNIV* {*S*. *open S*}. *f* −‘ *S* ∩ *space M* ∈ *sets M*)
by (*auto simp add*: *measurable-def borel-def*)

lemma *in-borel-measurable-borel*:
f ∈ *borel-measurable M* ⟷
(∀ *S* ∈ *sets borel*.
f −‘ *S* ∩ *space M* ∈ *sets M*)
by (*auto simp add*: *measurable-def borel-def*)

lemma *space-borel*[*simp*]: *space borel* = *UNIV*
unfolding *borel-def* **by** *auto*

lemma *space-in-borel*[*measurable*]: *UNIV* ∈ *sets borel*
unfolding *borel-def* **by** *auto*

lemma *sets-borel*: *sets borel* = *sigma-sets UNIV* {*S*. *open S*}
unfolding *borel-def* **by** (*rule sets-measure-of*) *simp*

lemma *measurable-sets-borel*:
⟦*f* ∈ *measurable borel M*; *A* ∈ *sets M*⟧ ⟹ *f* −‘ *A* ∈ *sets borel*
by (*drule* (*1*) *measurable-sets*) *simp*

lemma *pred-Collect-borel*[*measurable* (*raw*)]: *Measurable.pred borel P* ⟹ {*x*. *P x*} ∈ *sets borel*
unfolding *borel-def pred-def* **by** *auto*

lemma *borel-open*[*measurable* (*raw generic*)]:
assumes *open A* **shows** *A* ∈ *sets borel*
proof −
have *A* ∈ {*S*. *open S*} **unfolding** *mem-Collect-eq* **using** *assms* **.**
thus *?thesis* **unfolding** *borel-def* **by** *auto*
qed

lemma *borel-closed*[*measurable* (*raw generic*)]:
assumes *closed A* **shows** *A* ∈ *sets borel*
proof −
have *space borel* − (− *A*) ∈ *sets borel*
using *assms* **unfolding** *closed-def* **by** (*blast intro*: *borel-open*)

thus *?thesis* **by** *simp*
qed

lemma *borel-singleton*[*measurable*]:
A ∈ *sets borel* ⟹ *insert x A* ∈ *sets* (*borel* :: *'a*::*t1-space measure*)
unfolding *insert-def* **by** (*rule sets.Un*) *auto*

lemma *borel-comp*[*measurable*]: *A* ∈ *sets borel* ⟹ − *A* ∈ *sets borel*
unfolding *Compl-eq-Diff-UNIV* **by** *simp*

lemma *borel-measurable-vimage*:
fixes *f* :: *'a* ⇒ *'x*::*t2-space*
assumes *borel*[*measurable*]: *f* ∈ *borel-measurable M*
shows *f* −‘ {*x*} ∩ *space M* ∈ *sets M*
by *simp*

lemma *borel-measurableI*:
fixes *f* :: *'a* ⇒ *'x*::*topological-space*
assumes ⋀*S*. *open S* ⟹ *f* −‘ *S* ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel-measurable M*
unfolding *borel-def*
proof (*rule measurable-measure-of*, *simp-all*)
fix *S* :: *'x set* **assume** *open S* **thus** *f* −‘ *S* ∩ *space M* ∈ *sets M*
using *assms*[*of S*] **by** *simp*
qed

lemma *borel-measurable-const*:
(λ*x*. *c*) ∈ *borel-measurable M*
by *auto*

lemma *borel-measurable-indicator*:
assumes *A*: *A* ∈ *sets M*
shows *indicator A* ∈ *borel-measurable M*
unfolding *indicator-def* [*abs-def*] **using** *A*
by (*auto intro*!: *measurable-If-set*)

lemma *borel-measurable-count-space*[*measurable* (*raw*)]:
f ∈ *borel-measurable* (*count-space S*)
unfolding *measurable-def* **by** *auto*

lemma *borel-measurable-indicator'*[*measurable* (*raw*)]:
assumes [*measurable*]: {*x*∈*space M*. *f x* ∈ *A x*} ∈ *sets M*
shows (λ*x*. *indicator* (*A x*) (*f x*)) ∈ *borel-measurable M*
unfolding *indicator-def*[*abs-def*]
by (*auto intro*!: *measurable-If*)

lemma *borel-measurable-indicator-iff*:
(*indicator A* :: *'a* ⇒ *'x*::{*t1-space*, *zero-neq-one*}) ∈ *borel-measurable M* ⟷ *A* ∩ *space M* ∈ *sets M*

(**is** *?I* ∈ *borel-measurable M* ⟷ -)
proof
assume *?I* ∈ *borel-measurable M*
then have *?I* −‘ {*1*} ∩ *space M* ∈ *sets M*
unfolding *measurable-def* **by** *auto*
also have *?I* −‘ {*1*} ∩ *space M* = *A* ∩ *space M*
unfolding *indicator-def* [*abs-def*] **by** *auto*
finally show *A* ∩ *space M* ∈ *sets M* **.**
next
assume *A* ∩ *space M* ∈ *sets M*
moreover have *?I* ∈ *borel-measurable M* ⟷
(*indicator* (*A* ∩ *space M*) :: *'a* ⇒ *'x*) ∈ *borel-measurable M*
by (*intro measurable-cong*) (*auto simp*: *indicator-def*)
ultimately show *?I* ∈ *borel-measurable M* **by** *auto*
qed

lemma *borel-measurable-subalgebra*:
assumes *sets N* ⊆ *sets M space N* = *space M f* ∈ *borel-measurable N*
shows *f* ∈ *borel-measurable M*
using *assms* **unfolding** *measurable-def* **by** *auto*

lemma *borel-measurable-restrict-space-iff-ereal*:
fixes *f* :: *'a* ⇒ *ereal*
assumes Ω[*measurable*, *simp*]: Ω ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel-measurable* (*restrict-space M* Ω) ⟷
(λ*x*. *f x* ∗ *indicator* Ω *x*) ∈ *borel-measurable M*
by (*subst measurable-restrict-space-iff*)
(*auto simp*: *indicator-def if-distrib*[**where** *f*=λ*x*. *a* ∗ *x* **for** *a*] *cong del*: *if-cong*)

lemma *borel-measurable-restrict-space-iff-ennreal*:
fixes *f* :: *'a* ⇒ *ennreal*
assumes Ω[*measurable*, *simp*]: Ω ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel-measurable* (*restrict-space M* Ω) ⟷
(λ*x*. *f x* ∗ *indicator* Ω *x*) ∈ *borel-measurable M*
by (*subst measurable-restrict-space-iff*)
(*auto simp*: *indicator-def if-distrib*[**where** *f*=λ*x*. *a* ∗ *x* **for** *a*] *cong del*: *if-cong*)

lemma *borel-measurable-restrict-space-iff*:
fixes *f* :: *'a* ⇒ *'b*::*real-normed-vector*
assumes Ω[*measurable*, *simp*]: Ω ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel-measurable* (*restrict-space M* Ω) ⟷
(λ*x*. *indicator* Ω *x* ∗$_R$ *f x*) ∈ *borel-measurable M*
by (*subst measurable-restrict-space-iff*)
(*auto simp*: *indicator-def if-distrib*[**where** *f*=λ*x*. *x* ∗$_R$ *a* **for** *a*] *ac-simps cong del*: *if-cong*)

lemma *cbox-borel*[*measurable*]: *cbox a b* ∈ *sets borel*
by (*auto intro*: *borel-closed*)

lemma *box-borel*[*measurable*]: *box a b ∈ sets borel*
by (*auto intro*: *borel-open*)

lemma *borel-compact*: *compact* (*A*::*'a*::*t2-space set*) ⟹ *A ∈ sets borel*
by (*auto intro*: *borel-closed dest*!: *compact-imp-closed*)

lemma *borel-sigma-sets-subset*:
A ⊆ sets borel ⟹ *sigma-sets UNIV A ⊆ sets borel*
using *sets.sigma-sets-subset*[*of A borel*] **by** *simp*

lemma *borel-eq-sigmaI1*:
fixes *F* :: *'i* ⇒ *'a*::*topological-space set* **and** *X* :: *'a*::*topological-space set set*
assumes *borel-eq*: *borel* = *sigma UNIV X*
assumes *X*: ⋀*x*. *x ∈ X* ⟹ *x ∈ sets* (*sigma UNIV* (*F* ‘ *A*))
assumes *F*: ⋀*i*. *i ∈ A* ⟹ *F i ∈ sets borel*
shows *borel* = *sigma UNIV* (*F* ‘ *A*)
unfolding *borel-def*
proof (*intro sigma-eqI antisym*)
have *borel-rev-eq*: *sigma-sets UNIV* {*S*::*'a set*. *open S*} = *sets borel*
unfolding *borel-def* **by** *simp*
also have ... = *sigma-sets UNIV X*
unfolding *borel-eq* **by** *simp*
also have ... ⊆ *sigma-sets UNIV* (*F*‘*A*)
using *X* **by** (*intro sigma-algebra.sigma-sets-subset*[*OF sigma-algebra-sigma-sets*])
auto
finally show *sigma-sets UNIV* {*S*. *open S*} ⊆ *sigma-sets UNIV* (*F*‘*A*) .
show *sigma-sets UNIV* (*F*‘*A*) ⊆ *sigma-sets UNIV* {*S*. *open S*}
unfolding *borel-rev-eq* **using** *F* **by** (*intro borel-sigma-sets-subset*) *auto*
qed *auto*

lemma *borel-eq-sigmaI2*:
fixes *F* :: *'i* ⇒ *'j* ⇒ *'a*::*topological-space set*
and *G* :: *'l* ⇒ *'k* ⇒ *'a*::*topological-space set*
assumes *borel-eq*: *borel* = *sigma UNIV* ((λ(*i*, *j*). *G i j*)‘*B*)
assumes *X*: ⋀*i j*. (*i*, *j*) *∈ B* ⟹ *G i j ∈ sets* (*sigma UNIV* ((λ(*i*, *j*). *F i j*) ‘ *A*))
assumes *F*: ⋀*i j*. (*i*, *j*) *∈ A* ⟹ *F i j ∈ sets borel*
shows *borel* = *sigma UNIV* ((λ(*i*, *j*). *F i j*) ‘ *A*)
using *assms*
by (*intro borel-eq-sigmaI1*[**where** *X*=(λ(*i*, *j*). *G i j*) ‘ *B* **and** *F*=(λ(*i*, *j*). *F i j*)]) *auto*

lemma *borel-eq-sigmaI3*:
fixes *F* :: *'i* ⇒ *'j* ⇒ *'a*::*topological-space set* **and** *X* :: *'a*::*topological-space set set*
assumes *borel-eq*: *borel* = *sigma UNIV X*
assumes *X*: ⋀*x*. *x ∈ X* ⟹ *x ∈ sets* (*sigma UNIV* ((λ(*i*, *j*). *F i j*) ‘ *A*))
assumes *F*: ⋀*i j*. (*i*, *j*) *∈ A* ⟹ *F i j ∈ sets borel*
shows *borel* = *sigma UNIV* ((λ(*i*, *j*). *F i j*) ‘ *A*)
using *assms* **by** (*intro borel-eq-sigmaI1*[**where** *X*=*X* **and** *F*=(λ(*i*, *j*). *F i j*)])

auto

lemma *borel-eq-sigmaI4*:
fixes $F :: 'i \Rightarrow 'a$::*topological-space set*
and $G :: 'l \Rightarrow 'k \Rightarrow 'a$::*topological-space set*
assumes *borel-eq*: $borel = sigma\ UNIV\ ((\lambda(i, j).\ G\ i\ j)\text{`}A)$
assumes X: $\bigwedge i\ j.\ (i, j) \in A \Longrightarrow G\ i\ j \in sets\ (sigma\ UNIV\ (range\ F))$
assumes F: $\bigwedge i.\ F\ i \in sets\ borel$
shows $borel = sigma\ UNIV\ (range\ F)$
using *assms* **by** (*intro borel-eq-sigmaI1*[**where** $X=(\lambda(i, j).\ G\ i\ j)\ \text{`}\ A$ **and** $F=F$])
auto

lemma *borel-eq-sigmaI5*:
fixes $F :: 'i \Rightarrow 'j \Rightarrow 'a$::*topological-space set* **and** $G :: 'l \Rightarrow 'a$::*topological-space set*
assumes *borel-eq*: $borel = sigma\ UNIV\ (range\ G)$
assumes X: $\bigwedge i.\ G\ i \in sets\ (sigma\ UNIV\ (range\ (\lambda(i, j).\ F\ i\ j)))$
assumes F: $\bigwedge i\ j.\ F\ i\ j \in sets\ borel$
shows $borel = sigma\ UNIV\ (range\ (\lambda(i, j).\ F\ i\ j))$
using *assms* **by** (*intro borel-eq-sigmaI1*[**where** $X=range\ G$ **and** $F=(\lambda(i, j).\ F\ i\ j)$]) *auto*

lemma *second-countable-borel-measurable*:
fixes $X :: 'a$::*second-countable-topology set set*
assumes *eq*: $open = generate\text{-}topology\ X$
shows $borel = sigma\ UNIV\ X$
unfolding *borel-def*
proof (*intro sigma-eqI sigma-sets-eqI*)
interpret X: *sigma-algebra UNIV sigma-sets UNIV X*
by (*rule sigma-algebra-sigma-sets*) *simp*

fix $S :: 'a\ set$ **assume** $S \in Collect\ open$
then have *generate-topology X S*
by (*auto simp*: *eq*)
then show $S \in sigma\text{-}sets\ UNIV\ X$
proof *induction*
case (*UN K*)
then have K: $\bigwedge k.\ k \in K \Longrightarrow open\ k$
unfolding *eq* **by** *auto*
from *ex-countable-basis* **obtain** $B :: 'a\ set\ set$ **where**
B: $\bigwedge b.\ b \in B \Longrightarrow open\ b\ \bigwedge X.\ open\ X \Longrightarrow \exists\, b \subseteq B.\ (\bigcup b) = X$ **and** *countable B*
by (*auto simp*: *topological-basis-def*)
from *B*(*2*)[*OF K*] **obtain** m **where** m: $\bigwedge k.\ k \in K \Longrightarrow m\ k \subseteq B\ \bigwedge k.\ k \in K \Longrightarrow (\bigcup m\ k) = k$
by *metis*
def $U \equiv (\bigcup k \in K.\ m\ k)$
with m **have** *countable U*
by (*intro countable-subset*[*OF* - ‹*countable B*›]) *auto*

```
    have ⋃ U = (⋃ A∈U. A) by simp
    also have ... = ⋃ K
      unfolding U-def UN-simps by (simp add: m)
    finally have ⋃ U = ⋃ K .

    have ∀ b∈U. ∃ k∈K. b ⊆ k
      using m by (auto simp: U-def)
    then obtain u where u: ⋀b. b ∈ U ⟹ u b ∈ K and ⋀b. b ∈ U ⟹ b ⊆ u
b
      by metis
    then have (⋃ b∈U. u b) ⊆ ⋃ K ⋃ U ⊆ (⋃ b∈U. u b)
      by auto
    then have ⋃ K = (⋃ b∈U. u b)
      unfolding ‹⋃ U = ⋃ K› by auto
    also have ... ∈ sigma-sets UNIV X
      using u UN by (intro X.countable-UN' ‹countable U›) auto
    finally show ⋃ K ∈ sigma-sets UNIV X .
  qed auto
qed (auto simp: eq intro: generate-topology.Basis)

lemma borel-eq-closed: borel = sigma UNIV (Collect closed)
  unfolding borel-def
proof (intro sigma-eqI sigma-sets-eqI, safe)
  fix x :: 'a set assume open x
  hence x = UNIV − (UNIV − x) by auto
  also have ... ∈ sigma-sets UNIV (Collect closed)
    by (force intro: sigma-sets.Compl simp: ‹open x›)
  finally show x ∈ sigma-sets UNIV (Collect closed) by simp
next
  fix x :: 'a set assume closed x
  hence x = UNIV − (UNIV − x) by auto
  also have ... ∈ sigma-sets UNIV (Collect open)
    by (force intro: sigma-sets.Compl simp: ‹closed x›)
  finally show x ∈ sigma-sets UNIV (Collect open) by simp
qed simp-all

lemma borel-eq-countable-basis:
  fixes B::'a::topological-space set set
  assumes countable B
  assumes topological-basis B
  shows borel = sigma UNIV B
  unfolding borel-def
proof (intro sigma-eqI sigma-sets-eqI, safe)
  interpret countable-basis using assms by unfold-locales
  fix X::'a set assume open X
  from open-countable-basisE[OF this] guess B' . note B' = this
  then show X ∈ sigma-sets UNIV B
    by (blast intro: sigma-sets-UNION ‹countable B› countable-subset)
next
```

fix b **assume** $b \in B$
hence *open* b **by** (*rule topological-basis-open*[*OF assms*(*2*)])
thus $b \in$ *sigma-sets UNIV* (*Collect open*) **by** *auto*
qed *simp-all*

lemma *borel-measurable-continuous-on-restrict*:
fixes f :: $'a$::*topological-space* $\Rightarrow$ $'b$::*topological-space*
assumes f: *continuous-on A f*
shows $f \in$ *borel-measurable* (*restrict-space borel A*)
proof (*rule borel-measurableI*)
fix S :: $'b$ *set* **assume** *open S*
with f **obtain** T **where** $f -` S \cap A = T \cap A$ *open T*
by (*metis continuous-on-open-invariant*)
then show $f -` S \cap$ *space* (*restrict-space borel A*) $\in$ *sets* (*restrict-space borel A*)
by (*force simp add*: *sets-restrict-space space-restrict-space*)
qed

lemma *borel-measurable-continuous-on1*: *continuous-on UNIV f* $\Longrightarrow$ $f \in$ *borel-measurable borel*
by (*drule borel-measurable-continuous-on-restrict*) *simp*

lemma *borel-measurable-continuous-on-if*:
$A \in$ *sets borel* $\Longrightarrow$ *continuous-on A f* $\Longrightarrow$ *continuous-on* $(-A)$ *g* $\Longrightarrow$
$(\lambda x.$ *if* $x \in A$ *then f x else g x*$) \in$ *borel-measurable borel*
by (*auto simp add*: *measurable-If-restrict-space-iff Collect-neg-eq*
intro!: *borel-measurable-continuous-on-restrict*)

lemma *borel-measurable-continuous-countable-exceptions*:
fixes f :: $'a$::*t1-space* $\Rightarrow$ $'b$::*topological-space*
assumes X: *countable X*
assumes *continuous-on* $(-X)$ *f*
shows $f \in$ *borel-measurable borel*
proof (*rule measurable-discrete-difference*[*OF - X*])
have $X \in$ *sets borel*
by (*rule sets.countable*[*OF - X*]) *auto*
then show $(\lambda x.$ *if* $x \in X$ *then undefined else f x*$) \in$ *borel-measurable borel*
by (*intro borel-measurable-continuous-on-if assms continuous-intros*)
qed *auto*

lemma *borel-measurable-continuous-on*:
assumes f: *continuous-on UNIV f* **and** g: $g \in$ *borel-measurable M*
shows $(\lambda x.\ f\ (g\ x)) \in$ *borel-measurable M*
using *measurable-comp*[*OF g borel-measurable-continuous-on1*[*OF f*]] **by** (*simp add*: *comp-def*)

lemma *borel-measurable-continuous-on-indicator*:
fixes *f g* :: $'a$::*topological-space* $\Rightarrow$ $'b$::*real-normed-vector*
shows $A \in$ *sets borel* $\Longrightarrow$ *continuous-on A f* $\Longrightarrow$ $(\lambda x.$ *indicator A x* $*_R$ *f x*$) \in$ *borel-measurable borel*

by (*subst borel-measurable-restrict-space-iff*[*symmetric*])
(*auto intro*: *borel-measurable-continuous-on-restrict*)

lemma *borel-measurable-Pair*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow 'b$::*second-countable-topology* **and** $g :: 'a \Rightarrow 'c$::*second-countable-topology*
assumes *f*[*measurable*]: $f \in$ *borel-measurable M*
assumes *g*[*measurable*]: $g \in$ *borel-measurable M*
shows $(\lambda x.\ (f\ x,\ g\ x)) \in$ *borel-measurable M*
proof (*subst borel-eq-countable-basis*)
let *?B* = *SOME B*::$'b$ *set set*. *countable B* $\wedge$ *topological-basis B*
let *?C* = *SOME B*::$'c$ *set set*. *countable B* $\wedge$ *topological-basis B*
let *?P* = $(\lambda(b,\ c).\ b \times c)$ ‘ (*?B* $\times$ *?C*)
show *countable ?P topological-basis ?P*
by (*auto intro*!: *countable-basis topological-basis-prod is-basis*)

show $(\lambda x.\ (f\ x,\ g\ x)) \in$ *measurable M* (*sigma UNIV ?P*)
proof (*rule measurable-measure-of*)
fix *S* **assume** $S \in$ *?P*
then obtain *b c* **where** $b \in$ *?B* $c \in$ *?C* **and** *S*: $S = b \times c$ **by** *auto*
then have *borel*: *open b open c*
by (*auto intro*: *is-basis topological-basis-open*)
have $(\lambda x.\ (f\ x,\ g\ x)) -$‘ $S \cap$ *space M* $= (f -$‘ $b \cap$ *space M*$) \cap (g -$‘ $c \cap$ *space M*$)$
unfolding *S* **by** *auto*
also have $\ldots \in$ *sets M*
using *borel* **by** *simp*
finally show $(\lambda x.\ (f\ x,\ g\ x)) -$‘ $S \cap$ *space M* $\in$ *sets M* **.**
qed *auto*
qed

lemma *borel-measurable-continuous-Pair*:
fixes $f :: 'a \Rightarrow 'b$::*second-countable-topology* **and** $g :: 'a \Rightarrow 'c$::*second-countable-topology*
assumes [*measurable*]: $f \in$ *borel-measurable M*
assumes [*measurable*]: $g \in$ *borel-measurable M*
assumes *H*: *continuous-on UNIV* $(\lambda x.\ H\ (fst\ x)\ (snd\ x))$
shows $(\lambda x.\ H\ (f\ x)\ (g\ x)) \in$ *borel-measurable M*
proof −
have *eq*: $(\lambda x.\ H\ (f\ x)\ (g\ x)) = (\lambda x.\ (\lambda x.\ H\ (fst\ x)\ (snd\ x))\ (f\ x,\ g\ x))$ **by** *auto*
show *?thesis*
unfolding *eq* **by** (*rule borel-measurable-continuous-on*[*OF H*]) *auto*
qed

4.2 Borel spaces on order topologies

lemma [*measurable*]:
fixes *a b* :: $'a$::*linorder-topology*
shows *lessThan-borel*: $\{..< a\} \in$ *sets borel*
and *greaterThan-borel*: $\{a <..\} \in$ *sets borel*
and *greaterThanLessThan-borel*: $\{a<..<b\} \in$ *sets borel*

```
    and atMost-borel: {..a} ∈ sets borel
    and atLeast-borel: {a..} ∈ sets borel
    and atLeastAtMost-borel: {a..b} ∈ sets borel
    and greaterThanAtMost-borel: {a<..b} ∈ sets borel
    and atLeastLessThan-borel: {a..<b} ∈ sets borel
  unfolding greaterThanAtMost-def atLeastLessThan-def
  by (blast intro: borel-open borel-closed open-lessThan open-greaterThan open-greaterThanLessThan
                   closed-atMost closed-atLeast closed-atLeastAtMost)+

lemma borel-Iio:
  borel = sigma UNIV (range lessThan :: 'a::{linorder-topology, second-countable-topology}
set set)
  unfolding second-countable-borel-measurable[OF open-generated-order]
proof (intro sigma-eqI sigma-sets-eqI)
  from countable-dense-setE guess D :: 'a set . note D = this

  interpret L: sigma-algebra UNIV sigma-sets UNIV (range lessThan)
    by (rule sigma-algebra-sigma-sets) simp

  fix A :: 'a set assume A ∈ range lessThan ∪ range greaterThan
  then obtain y where A = {y <..} ∨ A = {..< y}
    by blast
  then show A ∈ sigma-sets UNIV (range lessThan)
  proof
    assume A: A = {y <..}
    show ?thesis
    proof cases
      assume ∀ x>y. ∃ d. y < d ∧ d < x
      with D(2)[of {y <..< x} for x] have ∀ x>y. ∃ d∈D. y < d ∧ d < x
        by (auto simp: set-eq-iff)
      then have A = UNIV - (⋂ d∈{d∈D. y < d}. {..< d})
        by (auto simp: A) (metis less-asym)
      also have ... ∈ sigma-sets UNIV (range lessThan)
        using D(1) by (intro L.Diff L.top L.countable-INT'') auto
      finally show ?thesis .
    next
      assume ¬ (∀ x>y. ∃ d. y < d ∧ d < x)
      then obtain x where y < x  ⋀d. y < d ⟹ ¬ d < x
        by auto
      then have A = UNIV - {..< x}
        unfolding A by (auto simp: not-less[symmetric])
      also have ... ∈ sigma-sets UNIV (range lessThan)
        by auto
      finally show ?thesis .
    qed
  qed auto
qed auto

lemma borel-Ioi:
```

borel = *sigma UNIV* (*range greaterThan* :: *'a*::{*linorder-topology*, *second-countable-topology*} *set set*)
unfolding *second-countable-borel-measurable*[*OF open-generated-order*]
proof (*intro sigma-eqI sigma-sets-eqI*)
from *countable-dense-setE* **guess** *D* :: *'a set* **. note** *D* = *this*

interpret *L*: *sigma-algebra UNIV sigma-sets UNIV* (*range greaterThan*)
by (*rule sigma-algebra-sigma-sets*) *simp*

fix *A* :: *'a set* **assume** *A* ∈ *range lessThan* ∪ *range greaterThan*
then obtain *y* **where** *A* = {*y* <..} ∨ *A* = {..< *y*}
by *blast*
then show *A* ∈ *sigma-sets UNIV* (*range greaterThan*)
proof
assume *A*: *A* = {..< *y*}
show *?thesis*
proof *cases*
assume ∀ *x*<*y*. ∃ *d*. *x* < *d* ∧ *d* < *y*
with *D*(*2*)[*of* {*x* <..< *y*} **for** *x*] **have** ∀ *x*<*y*. ∃ *d*∈*D*. *x* < *d* ∧ *d* < *y*
by (*auto simp*: *set-eq-iff*)
then have *A* = *UNIV* − (⋂ *d*∈{*d*∈*D*. *d* < *y*}. {*d* <..})
by (*auto simp*: *A*) (*metis less-asym*)
also have ... ∈ *sigma-sets UNIV* (*range greaterThan*)
using *D*(*1*) **by** (*intro L.Diff L.top L.countable-INT''*) *auto*
finally show *?thesis* **.**
next
assume ¬ (∀ *x*<*y*. ∃ *d*. *x* < *d* ∧ *d* < *y*)
then obtain *x* **where** *x* < *y* ⋀*d*. *y* > *d* ⟹ *x* ≥ *d*
by (*auto simp*: *not-less*[*symmetric*])
then have *A* = *UNIV* − {*x* <..}
unfolding *A Compl-eq-Diff-UNIV*[*symmetric*] **by** *auto*
also have ... ∈ *sigma-sets UNIV* (*range greaterThan*)
by *auto*
finally show *?thesis* **.**
qed
qed *auto*
qed *auto*

lemma *borel-measurableI-less*:
fixes *f* :: *'a* ⇒ *'b*::{*linorder-topology*, *second-countable-topology*}
shows (⋀*y*. {*x*∈*space M*. *f x* < *y*} ∈ *sets M*) ⟹ *f* ∈ *borel-measurable M*
unfolding *borel-Iio*
by (*rule measurable-measure-of*) (*auto simp*: *Int-def conj-commute*)

lemma *borel-measurableI-greater*:
fixes *f* :: *'a* ⇒ *'b*::{*linorder-topology*, *second-countable-topology*}
shows (⋀*y*. {*x*∈*space M*. *y* < *f x*} ∈ *sets M*) ⟹ *f* ∈ *borel-measurable M*
unfolding *borel-Ioi*
by (*rule measurable-measure-of*) (*auto simp*: *Int-def conj-commute*)

lemma *borel-measurableI-le*:
fixes $f :: 'a \Rightarrow 'b$::{*linorder-topology*, *second-countable-topology*}
shows $(\bigwedge y.\ \{x \in space\ M.\ f\ x \leq y\} \in sets\ M) \Longrightarrow f \in borel\text{-}measurable\ M$
by (*rule borel-measurableI-greater*) (*auto simp*: *not-le*[*symmetric*])

lemma *borel-measurableI-ge*:
fixes $f :: 'a \Rightarrow 'b$::{*linorder-topology*, *second-countable-topology*}
shows $(\bigwedge y.\ \{x \in space\ M.\ y \leq f\ x\} \in sets\ M) \Longrightarrow f \in borel\text{-}measurable\ M$
by (*rule borel-measurableI-less*) (*auto simp*: *not-le*[*symmetric*])

lemma *borel-measurable-less*[*measurable*]:
fixes $f :: 'a \Rightarrow 'b$::{*second-countable-topology*, *dense-linorder*, *linorder-topology*}
assumes $f \in borel\text{-}measurable\ M$
assumes $g \in borel\text{-}measurable\ M$
shows $\{w \in space\ M.\ f\ w < g\ w\} \in sets\ M$
proof −
have $\{w \in space\ M.\ f\ w < g\ w\} = (\lambda x.\ (f\ x,\ g\ x))\ -`\ \{x.\ fst\ x < snd\ x\} \cap space\ M$
by *auto*
also have $\ldots \in sets\ M$
by (*intro measurable-sets*[*OF borel-measurable-Pair borel-open*, *OF assms open-Collect-less*] *continuous-intros*)
finally show *?thesis* **.**
qed

lemma
fixes $f :: 'a \Rightarrow 'b$::{*second-countable-topology*, *dense-linorder*, *linorder-topology*}
assumes *f*[*measurable*]: $f \in borel\text{-}measurable\ M$
assumes *g*[*measurable*]: $g \in borel\text{-}measurable\ M$
shows *borel-measurable-le*[*measurable*]: $\{w \in space\ M.\ f\ w \leq g\ w\} \in sets\ M$
and *borel-measurable-eq*[*measurable*]: $\{w \in space\ M.\ f\ w = g\ w\} \in sets\ M$
and *borel-measurable-neq*: $\{w \in space\ M.\ f\ w \neq g\ w\} \in sets\ M$
unfolding *eq-iff not-less*[*symmetric*]
by *measurable*

lemma *borel-measurable-SUP*[*measurable* (*raw*)]:
fixes $F :: - \Rightarrow - \Rightarrow -$::{*complete-linorder*, *linorder-topology*, *second-countable-topology*}
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ SUP\ i{:}I.\ F\ i\ x) \in borel\text{-}measurable\ M$
by (*rule borel-measurableI-greater*) (*simp add*: *less-SUP-iff*)

lemma *borel-measurable-INF*[*measurable* (*raw*)]:
fixes $F :: - \Rightarrow - \Rightarrow -$::{*complete-linorder*, *linorder-topology*, *second-countable-topology*}
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ INF\ i{:}I.\ F\ i\ x) \in borel\text{-}measurable\ M$
by (*rule borel-measurableI-less*) (*simp add*: *INF-less-iff*)

lemma *borel-measurable-cSUP*[*measurable* (*raw*)]:
fixes $F :: - \Rightarrow - \Rightarrow 'a::\{conditionally\text{-}complete\text{-}linorder, linorder\text{-}topology, second\text{-}countable\text{-}topology\}$
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in borel\text{-}measurable\ M$
assumes *bdd*: $\bigwedge x.\ x \in space\ M \Longrightarrow bdd\text{-}above\ ((\lambda i.\ F\ i\ x)\ `\ I)$
shows $(\lambda x.\ SUP\ i{:}I.\ F\ i\ x) \in borel\text{-}measurable\ M$
proof *cases*
assume $I = \{\}$ **then show** *?thesis*
unfolding ‹$I = \{\}$› *image-empty* **by** *simp*
next
assume $I \neq \{\}$
show *?thesis*
proof (*rule borel-measurableI-le*)
fix *y*
have $\{x \in space\ M.\ \forall i{\in}I.\ F\ i\ x \leq y\} \in sets\ M$
by *measurable*
also have $\{x \in space\ M.\ \forall i{\in}I.\ F\ i\ x \leq y\} = \{x \in space\ M.\ (SUP\ i{:}I.\ F\ i\ x) \leq y\}$
by (*simp add*: *cSUP-le-iff* ‹$I \neq \{\}$› *bdd cong*: *conj-cong*)
finally show $\{x \in space\ M.\ (SUP\ i{:}I.\ F\ i\ x) \leq\ y\} \in sets\ M$ **.**
qed
qed

lemma *borel-measurable-cINF*[*measurable* (*raw*)]:
fixes $F :: - \Rightarrow - \Rightarrow 'a::\{conditionally\text{-}complete\text{-}linorder, linorder\text{-}topology, second\text{-}countable\text{-}topology\}$
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in borel\text{-}measurable\ M$
assumes *bdd*: $\bigwedge x.\ x \in space\ M \Longrightarrow bdd\text{-}below\ ((\lambda i.\ F\ i\ x)\ `\ I)$
shows $(\lambda x.\ INF\ i{:}I.\ F\ i\ x) \in borel\text{-}measurable\ M$
proof *cases*
assume $I = \{\}$ **then show** *?thesis*
unfolding ‹$I = \{\}$› *image-empty* **by** *simp*
next
assume $I \neq \{\}$
show *?thesis*
proof (*rule borel-measurableI-ge*)
fix *y*
have $\{x \in space\ M.\ \forall i{\in}I.\ y \leq F\ i\ x\} \in sets\ M$
by *measurable*
also have $\{x \in space\ M.\ \forall i{\in}I.\ y \leq F\ i\ x\} = \{x \in space\ M.\ y \leq (INF\ i{:}I.\ F\ i\ x)\}$
by (*simp add*: *le-cINF-iff* ‹$I \neq \{\}$› *bdd cong*: *conj-cong*)
finally show $\{x \in space\ M.\ y \leq (INF\ i{:}I.\ F\ i\ x)\} \in sets\ M$ **.**
qed
qed

lemma *borel-measurable-lfp*[*consumes 1*, *case-names continuity step*]:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{complete\text{-}linorder, linorder\text{-}topology, second\text{-}countable\text{-}topology\})$

assumes *sup-continuous F*
assumes ∗: ⋀*f*. *f* ∈ *borel-measurable M* ⟹ *F f* ∈ *borel-measurable M*
shows *lfp F* ∈ *borel-measurable M*
proof −
{ fix *i* **have** ((*F* ^^ *i*) *bot*) ∈ *borel-measurable M*
by (*induct i*) (*auto intro*!: ∗) **}**
then have (λ*x*. *SUP i*. (*F* ^^ *i*) *bot x*) ∈ *borel-measurable M*
by *measurable*
also have (λ*x*. *SUP i*. (*F* ^^ *i*) *bot x*) = (*SUP i*. (*F* ^^ *i*) *bot*)
by *auto*
also have (*SUP i*. (*F* ^^ *i*) *bot*) = *lfp F*
by (*rule sup-continuous-lfp*[*symmetric*]) *fact*
finally show *?thesis* **.**
qed

lemma *borel-measurable-gfp*[*consumes 1*, *case-names continuity step*]:
fixes *F* :: ('*a* ⇒ '*b*) ⇒ ('*a* ⇒ '*b*::{*complete-linorder*, *linorder-topology*, *second-countable-topology*})
assumes *inf-continuous F*
assumes ∗: ⋀*f*. *f* ∈ *borel-measurable M* ⟹ *F f* ∈ *borel-measurable M*
shows *gfp F* ∈ *borel-measurable M*
proof −
{ fix *i* **have** ((*F* ^^ *i*) *top*) ∈ *borel-measurable M*
by (*induct i*) (*auto intro*!: ∗ *simp*: *bot-fun-def*) **}**
then have (λ*x*. *INF i*. (*F* ^^ *i*) *top x*) ∈ *borel-measurable M*
by *measurable*
also have (λ*x*. *INF i*. (*F* ^^ *i*) *top x*) = (*INF i*. (*F* ^^ *i*) *top*)
by *auto*
also have ... = *gfp F*
by (*rule inf-continuous-gfp*[*symmetric*]) *fact*
finally show *?thesis* **.**
qed

lemma *borel-measurable-max*[*measurable* (*raw*)]:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹ (λ*x*. *max* (*g x*) (*f x*) :: '*b*::{*second-countable-topology*, *linorder-topology*}) ∈ *borel-measurable M*
by (*rule borel-measurableI-less*) *simp*

lemma *borel-measurable-min*[*measurable* (*raw*)]:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹ (λ*x*. *min* (*g x*) (*f x*) :: '*b*::{*second-countable-topology*, *linorder-topology*}) ∈ *borel-measurable M*
by (*rule borel-measurableI-greater*) *simp*

lemma *borel-measurable-Min*[*measurable* (*raw*)]:
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *borel-measurable M*) ⟹ (λ*x*. *Min* ((λ*i*. *f i x*)‘*I*) :: '*b*::{*second-countable-topology*, *linorder-topology*}) ∈ *borel-measurable M*
proof (*induct I rule*: *finite-induct*)
case (*insert i I*) **then show** *?case*
by (*cases I* = {}) *auto*
qed *auto*

lemma *borel-measurable-Max*[*measurable* (*raw*)]:
finite I $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow f\ i \in$ *borel-measurable M*) $\Longrightarrow$ ($\lambda x.$ *Max* (($\lambda i.\ f\ i\ x$)`*I*) :: *'b*::{*second-countable-topology*, *linorder-topology*}) $\in$ *borel-measurable M*
proof (*induct I rule*: *finite-induct*)
case (*insert i I*) **then show** *?case*
by (*cases I* = {}) *auto*
qed *auto*

lemma *borel-measurable-sup*[*measurable* (*raw*)]:
f $\in$ *borel-measurable M* $\Longrightarrow$ *g* $\in$ *borel-measurable M* $\Longrightarrow$ ($\lambda x.$ *sup* (*g x*) (*f x*) :: *'b*::{*lattice*, *second-countable-topology*, *linorder-topology*}) $\in$ *borel-measurable M*
unfolding *sup-max* **by** *measurable*

lemma *borel-measurable-inf*[*measurable* (*raw*)]:
f $\in$ *borel-measurable M* $\Longrightarrow$ *g* $\in$ *borel-measurable M* $\Longrightarrow$ ($\lambda x.$ *inf* (*g x*) (*f x*) :: *'b*::{*lattice*, *second-countable-topology*, *linorder-topology*}) $\in$ *borel-measurable M*
unfolding *inf-min* **by** *measurable*

lemma [*measurable* (*raw*)]:
fixes *f* :: *nat* $\Rightarrow$ *'a* $\Rightarrow$ *'b*::{*complete-linorder*, *second-countable-topology*, *linorder-topology*}
assumes $\bigwedge i.\ f\ i \in$ *borel-measurable M*
shows *borel-measurable-liminf*: ($\lambda x.$ *liminf* ($\lambda i.\ f\ i\ x$)) $\in$ *borel-measurable M*
and *borel-measurable-limsup*: ($\lambda x.$ *limsup* ($\lambda i.\ f\ i\ x$)) $\in$ *borel-measurable M*
unfolding *liminf-SUP-INF limsup-INF-SUP* **using** *assms* **by** *auto*

lemma *measurable-convergent*[*measurable* (*raw*)]:
fixes *f* :: *nat* $\Rightarrow$ *'a* $\Rightarrow$ *'b*::{*complete-linorder*, *second-countable-topology*, *dense-linorder*, *linorder-topology*}
assumes [*measurable*]: $\bigwedge i.\ f\ i \in$ *borel-measurable M*
shows *Measurable.pred M* ($\lambda x.$ *convergent* ($\lambda i.\ f\ i\ x$))
unfolding *convergent-ereal* **by** *measurable*

lemma *sets-Collect-convergent*[*measurable*]:
fixes *f* :: *nat* $\Rightarrow$ *'a* $\Rightarrow$ *'b*::{*complete-linorder*, *second-countable-topology*, *dense-linorder*, *linorder-topology*}
assumes *f*[*measurable*]: $\bigwedge i.\ f\ i \in$ *borel-measurable M*
shows {x∈*space M*. *convergent* ($\lambda i.\ f\ i\ x$)} $\in$ *sets M*
by *measurable*

lemma *borel-measurable-lim*[*measurable* (*raw*)]:
fixes *f* :: *nat* $\Rightarrow$ *'a* $\Rightarrow$ *'b*::{*complete-linorder*, *second-countable-topology*, *dense-linorder*, *linorder-topology*}
assumes [*measurable*]: $\bigwedge i.\ f\ i \in$ *borel-measurable M*
shows ($\lambda x.$ *lim* ($\lambda i.\ f\ i\ x$)) $\in$ *borel-measurable M*
proof −
have $\bigwedge x.$ *lim* ($\lambda i.\ f\ i\ x$) = (*if convergent* ($\lambda i.\ f\ i\ x$) *then limsup* ($\lambda i.\ f\ i\ x$) *else* (*THE i*. *False*))
by (*simp add*: *lim-def convergent-def convergent-limsup-cl*)

then show *?thesis*
by *simp*
qed

lemma *borel-measurable-LIMSEQ-order*:
fixes $u :: nat \Rightarrow 'a \Rightarrow 'b$::{*complete-linorder*, *second-countable-topology*, *dense-linorder*, *linorder-topology*}
assumes u': $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ u\ i\ x) \longrightarrow u'\ x$
and u: $\bigwedge i.\ u\ i \in borel\text{-}measurable\ M$
shows $u' \in borel\text{-}measurable\ M$
proof −
have $\bigwedge x.\ x \in space\ M \Longrightarrow u'\ x = liminf\ (\lambda n.\ u\ n\ x)$
using u' **by** (*simp add*: *lim-imp-Liminf*[*symmetric*])
with u **show** *?thesis* **by** (*simp cong*: *measurable-cong*)
qed

4.3 Borel spaces on topological monoids

lemma *borel-measurable-add*[*measurable* (*raw*)]:
fixes $f\ g :: 'a \Rightarrow 'b$::{*second-countable-topology*, *topological-monoid-add*}
assumes f: $f \in borel\text{-}measurable\ M$
assumes g: $g \in borel\text{-}measurable\ M$
shows $(\lambda x.\ f\ x + g\ x) \in borel\text{-}measurable\ M$
using $f\ g$ **by** (*rule borel-measurable-continuous-Pair*) (*intro continuous-intros*)

lemma *borel-measurable-setsum*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow 'b$::{*second-countable-topology*, *topological-comm-monoid-add*}
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ \sum i{\in}S.\ f\ i\ x) \in borel\text{-}measurable\ M$
proof *cases*
assume *finite S*
thus *?thesis* **using** *assms* **by** *induct auto*
qed *simp*

lemma *borel-measurable-suminf-order*[*measurable* (*raw*)]:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b$::{*complete-linorder*, *second-countable-topology*, *dense-linorder*, *linorder-topology*, *topological-comm-monoid-add*}
assumes *f*[*measurable*]: $\bigwedge i.\ f\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ suminf\ (\lambda i.\ f\ i\ x)) \in borel\text{-}measurable\ M$
unfolding *suminf-def sums-def*[*abs-def*] *lim-def*[*symmetric*] **by** *simp*

4.4 Borel spaces on Euclidean spaces

lemma *borel-measurable-inner*[*measurable* (*raw*)]:
fixes $f\ g :: 'a \Rightarrow 'b$::{*second-countable-topology*, *real-inner*}
assumes $f \in borel\text{-}measurable\ M$
assumes $g \in borel\text{-}measurable\ M$
shows $(\lambda x.\ f\ x \cdot g\ x) \in borel\text{-}measurable\ M$
using *assms*
by (*rule borel-measurable-continuous-Pair*) (*intro continuous-intros*)

notation
eucl-less (**infix** $<e$ *50*)

lemma *box-oc*: $\{x.\ a <e\ x \wedge x \leq b\} = \{x.\ a <e\ x\} \cap \{..b\}$
and *box-co*: $\{x.\ a \leq x \wedge x <e\ b\} = \{a..\} \cap \{x.\ x <e\ b\}$
by *auto*

lemma *eucl-ivals*[*measurable*]:
fixes *a b* :: *'a*::*ordered-euclidean-space*
shows $\{x.\ x <e\ a\} \in$ *sets borel*
and $\{x.\ a <e\ x\} \in$ *sets borel*
and $\{..a\} \in$ *sets borel*
and $\{a..\} \in$ *sets borel*
and $\{a..b\} \in$ *sets borel*
and $\{x.\ a <e\ x \wedge x \leq b\} \in$ *sets borel*
and $\{x.\ a \leq x \wedge\ x <e\ b\} \in$ *sets borel*
unfolding *box-oc box-co*
by (*auto intro*: *borel-open borel-closed*)

lemma
fixes *i* :: *'a*::{*second-countable-topology*, *real-inner*}
shows *hafspace-less-borel*: $\{x.\ a < x \cdot i\} \in$ *sets borel*
and *hafspace-greater-borel*: $\{x.\ x \cdot i < a\} \in$ *sets borel*
and *hafspace-less-eq-borel*: $\{x.\ a \leq x \cdot i\} \in$ *sets borel*
and *hafspace-greater-eq-borel*: $\{x.\ x \cdot i \leq a\} \in$ *sets borel*
by *simp-all*

lemma *borel-eq-box*:
borel = *sigma UNIV* (*range* (λ (*a*, *b*). *box a b* :: *'a* :: *euclidean-space set*))
(**is** - = *?SIGMA*)
proof (*rule borel-eq-sigmaI1*[*OF borel-def*])
fix *M* :: *'a set* **assume** *M* $\in \{S.$ *open S*$\}$
then have *open M* **by** *simp*
show *M* $\in$ *?SIGMA*
apply (*subst open-UNION-box*[*OF* ‹*open M*›])
apply (*safe intro*!: *sets.countable-UN' countable-PiE countable-Collect*)
apply (*auto intro*: *countable-rat*)
done
qed (*auto simp*: *box-def*)

lemma *halfspace-gt-in-halfspace*:
assumes *i*: *i* $\in$ *A*
shows $\{x$::*'a*. *a* $< x \cdot i\} \in$
sigma-sets UNIV ((λ (*a*, *i*). $\{x$::*'a*::*euclidean-space*. $x \cdot i < a\}$) ‘ (*UNIV* $\times$ *A*))
(**is** *?set* $\in$ *?SIGMA*)
proof −
interpret *sigma-algebra UNIV ?SIGMA*

```
    by (intro sigma-algebra-sigma-sets) simp-all
  have *: ?set = (⋃n. UNIV - {x::'a. x · i < a + 1 / real (Suc n)})
  proof (safe, simp-all add: not-less del: of-nat-Suc)
    fix x :: 'a assume a < x · i
    with reals-Archimedean[of x · i - a]
    obtain n where a + 1 / real (Suc n) < x · i
      by (auto simp: field-simps)
    then show ∃n. a + 1 / real (Suc n) ≤ x · i
      by (blast intro: less-imp-le)
  next
    fix x n
    have a < a + 1 / real (Suc n) by auto
    also assume ... ≤ x
    finally show a < x .
  qed
  show ?set ∈ ?SIGMA unfolding *
    by (auto intro!: Diff sigma-sets-Inter i)
qed

lemma borel-eq-halfspace-less:
  borel = sigma UNIV ((λ(a, i). {x::'a::euclidean-space. x · i < a}) ' (UNIV ×
Basis))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI2[OF borel-eq-box])
  fix a b :: 'a
  have box a b = {x∈space ?SIGMA. ∀i∈Basis. a · i < x · i ∧ x · i < b · i}
    by (auto simp: box-def)
  also have ... ∈ sets ?SIGMA
   by (intro sets.sets-Collect-conj sets.sets-Collect-finite-All sets.sets-Collect-const)
      (auto intro!: halfspace-gt-in-halfspace countable-PiE countable-rat)
  finally show box a b ∈ sets ?SIGMA .
qed auto

lemma borel-eq-halfspace-le:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean-space. x · i ≤ a}) ' (UNIV ×
Basis))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI2[OF borel-eq-halfspace-less])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have *: {x::'a. x·i < a} = (⋃n. {x. x·i ≤ a - 1/real (Suc n)})
  proof (safe, simp-all del: of-nat-Suc)
    fix x::'a assume *: x·i < a
    with reals-Archimedean[of a - x·i]
    obtain n where x · i < a - 1 / (real (Suc n))
      by (auto simp: field-simps)
    then show ∃n. x · i ≤ a - 1 / (real (Suc n))
      by (blast intro: less-imp-le)
  next
```

```
    fix x::'a and n
    assume x•i ≤ a − 1 / real (Suc n)
    also have ... < a by auto
    finally show x•i < a .
  qed
  show {x. x•i < a} ∈ ?SIGMA unfolding ∗
    by (intro sets.countable-UN) (auto intro: i)
qed auto

lemma borel-eq-halfspace-ge:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean-space. a ≤ x • i}) ‘ (UNIV ×
Basis))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI2[OF borel-eq-halfspace-less])
  fix a :: real and i :: 'a assume i: (a, i) ∈ UNIV × Basis
  have ∗: {x::'a. x•i < a} = space ?SIGMA − {x::'a. a ≤ x•i} by auto
  show {x. x•i < a} ∈ ?SIGMA unfolding ∗
    using i by (intro sets.compl-sets) auto
qed auto

lemma borel-eq-halfspace-greater:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean-space. a < x • i}) ‘ (UNIV ×
Basis))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI2[OF borel-eq-halfspace-le])
  fix a :: real and i :: 'a assume (a, i) ∈ (UNIV × Basis)
  then have i: i ∈ Basis by auto
  have ∗: {x::'a. x•i ≤ a} = space ?SIGMA − {x::'a. a < x•i} by auto
  show {x. x•i ≤ a} ∈ ?SIGMA unfolding ∗
    by (intro sets.compl-sets) (auto intro: i)
qed auto

lemma borel-eq-atMost:
  borel = sigma UNIV (range (λa. {..a::'a::ordered-euclidean-space}))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI4[OF borel-eq-halfspace-le])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i ∈ Basis by auto
  then have ∗: {x::'a. x•i ≤ a} = (⋃ k::nat. {.. (∑ n∈Basis. (if n = i then a else
real k)∗R n)})
  proof (safe, simp-all add: eucl-le[where 'a='a] split: if-split-asm)
    fix x :: 'a
    from real-arch-simple[of Max ((λi. x•i) ‘Basis)] guess k::nat ..
    then have ⋀i. i ∈ Basis ⟹ x•i ≤ real k
      by (subst (asm) Max-le-iff) auto
    then show ∃ k::nat. ∀ ia∈Basis. ia ≠ i ⟶ x • ia ≤ real k
      by (auto intro!: exI[of - k])
  qed
  show {x. x•i ≤ a} ∈ ?SIGMA unfolding ∗
```

```
    by (intro sets.countable-UN) auto
qed auto

lemma borel-eq-greaterThan:
  borel = sigma UNIV (range (λa::'a::ordered-euclidean-space. {x. a <e x}))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI4[OF borel-eq-halfspace-le])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have {x::'a. x•i ≤ a} = UNIV - {x::'a. a < x•i} by auto
  also have *: {x::'a. a < x•i} =
      (⋃k::nat. {x. (∑ n∈Basis. (if n = i then a else -real k) *R n) <e x}) using
i
  proof (safe, simp-all add: eucl-less-def split: if-split-asm)
    fix x :: 'a
    from reals-Archimedean2[of Max ((λi. -x•i)`Basis)]
    guess k::nat .. note k = this
    { fix i :: 'a assume i ∈ Basis
      then have -x•i < real k
        using k by (subst (asm) Max-less-iff) auto
      then have - real k < x•i by simp }
    then show ∃k::nat. ∀ia∈Basis. ia ≠ i ⟶ -real k < x • ia
      by (auto intro!: exI[of - k])
  qed
  finally show {x. x•i ≤ a} ∈ ?SIGMA
    apply (simp only:)
    apply (intro sets.countable-UN sets.Diff)
    apply (auto intro: sigma-sets-top)
    done
qed auto

lemma borel-eq-lessThan:
  borel = sigma UNIV (range (λa::'a::ordered-euclidean-space. {x. x <e a}))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI4[OF borel-eq-halfspace-ge])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have {x::'a. a ≤ x•i} = UNIV - {x::'a. x•i < a} by auto
  also have *: {x::'a. x•i < a} = (⋃k::nat. {x. x <e (∑ n∈Basis. (if n = i then
a else real k) *R n)}) using ‹i∈ Basis›
  proof (safe, simp-all add: eucl-less-def split: if-split-asm)
    fix x :: 'a
    from reals-Archimedean2[of Max ((λi. x•i)`Basis)]
    guess k::nat .. note k = this
    { fix i :: 'a assume i ∈ Basis
      then have x•i < real k
        using k by (subst (asm) Max-less-iff) auto
      then have x•i < real k by simp }
    then show ∃k::nat. ∀ia∈Basis. ia ≠ i ⟶ x • ia < real k
```

```
      by (auto intro!: exI[of - k])
  qed
  finally show {x. a ≤ x·i} ∈ ?SIGMA
    apply (simp only:)
    apply (intro sets.countable-UN sets.Diff)
    apply (auto intro: sigma-sets-top )
    done
qed auto

lemma borel-eq-atLeastAtMost:
  borel = sigma UNIV (range (λ(a,b). {a..b} ::'a::ordered-euclidean-space set))
  (is - = ?SIGMA)
proof (rule borel-eq-sigmaI5[OF borel-eq-atMost])
  fix a::'a
  have *: {..a} = (⋃ n::nat. {- real n *R One .. a})
  proof (safe, simp-all add: eucl-le[where 'a='a])
    fix x :: 'a
    from real-arch-simple[of Max ((λi. - x·i) `Basis)]
    guess k::nat .. note k = this
    { fix i :: 'a assume i ∈ Basis
      with k have - x·i ≤ real k
        by (subst (asm) Max-le-iff) (auto simp: field-simps)
      then have - real k ≤ x·i by simp }
    then show ∃ n::nat. ∀ i∈Basis. - real n ≤ x · i
      by (auto intro!: exI[of - k])
  qed
  show {..a} ∈ ?SIGMA unfolding *
    by (intro sets.countable-UN)
       (auto intro!: sigma-sets-top)
qed auto

lemma borel-set-induct[consumes 1, case-names empty interval compl union]:
  assumes A ∈ sets borel
  assumes empty: P {} and int: ⋀a b. a ≤ b ⟹ P {a..b} and compl: ⋀A. A ∈
sets borel ⟹ P A ⟹ P (-A) and
        un: ⋀f. disjoint-family f ⟹ (⋀i. f i ∈ sets borel) ⟹ (⋀i. P (f i)) ⟹
P (⋃ i::nat. f i)
  shows P (A::real set)
proof-
  let ?G = range (λ(a,b). {a..b::real})
  have Int-stable ?G ?G ⊆ Pow UNIV A ∈ sigma-sets UNIV ?G
    using assms(1) by (auto simp add: borel-eq-atLeastAtMost Int-stable-def)
  thus ?thesis
  proof (induction rule: sigma-sets-induct-disjoint)
    case (union f)
    from union.hyps(2) have ⋀i. f i ∈ sets borel by (auto simp: borel-eq-atLeastAtMost)
      with union show ?case by (auto intro: un)
  next
    case (basic A)
```

then obtain *a b* **where** $A = \{a .. b\}$ **by** *auto*
then show *?case*
by (*cases* $a \leq b$) (*auto intro*: *int empty*)
qed (*auto intro*: *empty compl simp*: *Compl-eq-Diff-UNIV*[*symmetric*] *borel-eq-atLeastAtMost*)
qed

lemma *borel-sigma-sets-Ioc*: *borel* = *sigma UNIV* (*range* (λ(*a*, *b*). {*a* <.. *b*::*real*}))
proof (*rule borel-eq-sigmaI5*[*OF borel-eq-atMost*])
fix *i* :: *real*
have {..*i*} = ($\bigcup$*j*::*nat*. {−*j* <.. *i*})
by (*auto simp*: *minus-less-iff reals-Archimedean2*)
also have ... ∈ *sets* (*sigma UNIV* (*range* (λ(*i*, *j*). {*i*<..*j*})))
by (*intro sets.countable-nat-UN*) *auto*
finally show {..*i*} ∈ *sets* (*sigma UNIV* (*range* (λ(*i*, *j*). {*i*<..*j*}))) .
qed *simp*

lemma *eucl-lessThan*: {*x*::*real*. *x* <*e* *a*} = *lessThan a*
by (*simp add*: *eucl-less-def lessThan-def*)

lemma *borel-eq-atLeastLessThan*:
borel = *sigma UNIV* (*range* (λ(*a*, *b*). {*a* ..< *b* :: *real*})) (**is** - = *?SIGMA*)
proof (*rule borel-eq-sigmaI5*[*OF borel-eq-lessThan*])
have *move-uminus*: $\bigwedge x\ y$::*real*. $-x \leq y \longleftrightarrow -y \leq x$ **by** *auto*
fix *x* :: *real*
have {..<*x*} = ($\bigcup$*i*::*nat*. {−*real i* ..< *x*})
by (*auto simp*: *move-uminus real-arch-simple*)
then show {*y*. *y* <*e* *x*} ∈ *?SIGMA*
by (*auto intro*: *sigma-sets.intros*(*2*−) *simp*: *eucl-lessThan*)
qed *auto*

lemma *borel-measurable-halfspacesI*:
fixes *f* :: *'a* ⇒ *'c*::*euclidean-space*
assumes *F*: *borel* = *sigma UNIV* (*F* ‘ (*UNIV* × *Basis*))
and *S-eq*: $\bigwedge a\ i$. *S a i* = *f* −‘ *F* (*a*,*i*) ∩ *space M*
shows *f* ∈ *borel-measurable M* = (∀ *i*∈*Basis*. ∀ *a*::*real*. *S a i* ∈ *sets M*)
proof *safe*
fix *a* :: *real* **and** *i* :: *'b* **assume** *i*: *i* ∈ *Basis* **and** *f*: *f* ∈ *borel-measurable M*
then show *S a i* ∈ *sets M* **unfolding** *assms*
by (*auto intro*!: *measurable-sets simp*: *assms*(*1*))
next
assume *a*: ∀ *i*∈*Basis*. ∀ *a*. *S a i* ∈ *sets M*
then show *f* ∈ *borel-measurable M*
by (*auto intro*!: *measurable-measure-of simp*: *S-eq F*)
qed

lemma *borel-measurable-iff-halfspace-le*:
fixes *f* :: *'a* ⇒ *'c*::*euclidean-space*
shows *f* ∈ *borel-measurable M* = (∀ *i*∈*Basis*. ∀ *a*. {*w* ∈ *space M*. *f w* • *i* ≤ *a*}
∈ *sets M*)

by (*rule borel-measurable-halfspacesI*[*OF borel-eq-halfspace-le*]) *auto*

lemma *borel-measurable-iff-halfspace-less*:
fixes $f :: 'a \Rightarrow 'c$::*euclidean-space*
shows $f \in$ *borel-measurable* $M \longleftrightarrow (\forall\, i{\in}Basis.\ \forall\, a.\ \{w \in space\ M.\ f\ w \cdot i < a\} \in sets\ M)$
by (*rule borel-measurable-halfspacesI*[*OF borel-eq-halfspace-less*]) *auto*

lemma *borel-measurable-iff-halfspace-ge*:
fixes $f :: 'a \Rightarrow 'c$::*euclidean-space*
shows $f \in$ *borel-measurable* $M = (\forall\, i{\in}Basis.\ \forall\, a.\ \{w \in space\ M.\ a \leq f\ w \cdot i\} \in sets\ M)$
by (*rule borel-measurable-halfspacesI*[*OF borel-eq-halfspace-ge*]) *auto*

lemma *borel-measurable-iff-halfspace-greater*:
fixes $f :: 'a \Rightarrow 'c$::*euclidean-space*
shows $f \in$ *borel-measurable* $M \longleftrightarrow (\forall\, i{\in}Basis.\ \forall\, a.\ \{w \in space\ M.\ a < f\ w \cdot i\} \in sets\ M)$
by (*rule borel-measurable-halfspacesI*[*OF borel-eq-halfspace-greater*]) *auto*

lemma *borel-measurable-iff-le*:
$(f::'a \Rightarrow real) \in$ *borel-measurable* $M = (\forall\, a.\ \{w \in space\ M.\ f\ w \leq a\} \in sets\ M)$
using *borel-measurable-iff-halfspace-le*[**where** *'c=real*] **by** *simp*

lemma *borel-measurable-iff-less*:
$(f::'a \Rightarrow real) \in$ *borel-measurable* $M = (\forall\, a.\ \{w \in space\ M.\ f\ w < a\} \in sets\ M)$
using *borel-measurable-iff-halfspace-less*[**where** *'c=real*] **by** *simp*

lemma *borel-measurable-iff-ge*:
$(f::'a \Rightarrow real) \in$ *borel-measurable* $M = (\forall\, a.\ \{w \in space\ M.\ a \leq f\ w\} \in sets\ M)$
using *borel-measurable-iff-halfspace-ge*[**where** *'c=real*]
by *simp*

lemma *borel-measurable-iff-greater*:
$(f::'a \Rightarrow real) \in$ *borel-measurable* $M = (\forall\, a.\ \{w \in space\ M.\ a < f\ w\} \in sets\ M)$
using *borel-measurable-iff-halfspace-greater*[**where** *'c=real*] **by** *simp*

lemma *borel-measurable-euclidean-space*:
fixes $f :: 'a \Rightarrow 'c$::*euclidean-space*
shows $f \in$ *borel-measurable* $M \longleftrightarrow (\forall\, i{\in}Basis.\ (\lambda x.\ f\ x \cdot i) \in borel\text{-}measurable\ M)$
proof *safe*
assume f: $\forall\, i{\in}Basis.\ (\lambda x.\ f\ x \cdot i) \in$ *borel-measurable* M
then show $f \in$ *borel-measurable* M
by (*subst borel-measurable-iff-halfspace-le*) *auto*
qed *auto*

4.5 Borel measurable operators

lemma *borel-measurable-norm*[*measurable*]: *norm* ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-sgn* [*measurable*]: (*sgn*::*'a*::*real-normed-vector* ⇒ *'a*) ∈ *borel-measurable borel*
by (*rule borel-measurable-continuous-countable-exceptions*[**where** *X*={*0*}])
(*auto intro*!: *continuous-on-sgn continuous-on-id*)

lemma *borel-measurable-uminus*[*measurable* (*raw*)]:
fixes *g* :: *'a* ⇒ *'b*::{*second-countable-topology*, *real-normed-vector*}
assumes *g*: *g* ∈ *borel-measurable M*
shows (λ*x*. − *g x*) ∈ *borel-measurable M*
by (*rule borel-measurable-continuous-on*[*OF - g*]) (*intro continuous-intros*)

lemma *borel-measurable-diff*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *'b*::{*second-countable-topology*, *real-normed-vector*}
assumes *f*: *f* ∈ *borel-measurable M*
assumes *g*: *g* ∈ *borel-measurable M*
shows (λ*x*. *f x* − *g x*) ∈ *borel-measurable M*
using *borel-measurable-add* [*of f M* − *g*] *assms* **by** (*simp add*: *fun-Compl-def*)

lemma *borel-measurable-times*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *'b*::{*second-countable-topology*, *real-normed-algebra*}
assumes *f*: *f* ∈ *borel-measurable M*
assumes *g*: *g* ∈ *borel-measurable M*
shows (λ*x*. *f x* ∗ *g x*) ∈ *borel-measurable M*
using *f g* **by** (*rule borel-measurable-continuous-Pair*) (*intro continuous-intros*)

lemma *borel-measurable-setprod*[*measurable* (*raw*)]:
fixes *f* :: *'c* ⇒ *'a* ⇒ *'b*::{*second-countable-topology*, *real-normed-field*}
assumes ⋀*i*. *i* ∈ *S* ⟹ *f i* ∈ *borel-measurable M*
shows (λ*x*. ∏ *i*∈*S*. *f i x*) ∈ *borel-measurable M*
proof *cases*
assume *finite S*
thus *?thesis* **using** *assms* **by** *induct auto*
qed *simp*

lemma *borel-measurable-dist*[*measurable* (*raw*)]:
fixes *g f* :: *'a* ⇒ *'b*::{*second-countable-topology*, *metric-space*}
assumes *f*: *f* ∈ *borel-measurable M*
assumes *g*: *g* ∈ *borel-measurable M*
shows (λ*x*. *dist* (*f x*) (*g x*)) ∈ *borel-measurable M*
using *f g* **by** (*rule borel-measurable-continuous-Pair*) (*intro continuous-intros*)

lemma *borel-measurable-scaleR*[*measurable* (*raw*)]:
fixes *g* :: *'a* ⇒ *'b*::{*second-countable-topology*, *real-normed-vector*}
assumes *f*: *f* ∈ *borel-measurable M*
assumes *g*: *g* ∈ *borel-measurable M*

shows $(\lambda x.\ f\ x *_R g\ x) \in$ *borel-measurable M*
using *f g* **by** (*rule borel-measurable-continuous-Pair*) (*intro continuous-intros*)

lemma *affine-borel-measurable-vector*:
fixes $f :: 'a \Rightarrow 'x$::*real-normed-vector*
assumes $f \in$ *borel-measurable M*
shows $(\lambda x.\ a + b *_R f\ x) \in$ *borel-measurable M*
proof (*rule borel-measurableI*)
fix $S :: 'x$ *set* **assume** *open S*
show $(\lambda x.\ a + b *_R f\ x) -` S \cap$ *space M* $\in$ *sets M*
proof *cases*
assume $b \neq 0$
with ‹*open S*› **have** *open* $((\lambda x.\ (-\ a + x) /_R b) ` S)$ (**is** *open ?S*)
using *open-affinity* [*of S inverse b − a* $/_R$ *b*]
by (*auto simp*: *algebra-simps*)
hence *?S* $\in$ *sets borel* **by** *auto*
moreover
from ‹$b \neq 0$› **have** $(\lambda x.\ a + b *_R f\ x) -` S = f -` ?S$
apply *auto* **by** (*rule-tac x*=*a* + *b* $*_R$ *f x* **in** *image-eqI*, *simp-all*)
ultimately show *?thesis* **using** *assms* **unfolding** *in-borel-measurable-borel*
by *auto*
qed *simp*
qed

lemma *borel-measurable-const-scaleR*[*measurable* (*raw*)]:
f $\in$ *borel-measurable M* $\Longrightarrow$ $(\lambda x.\ b *_R f\ x :: 'a$::*real-normed-vector*$) \in$ *borel-measurable M*
using *affine-borel-measurable-vector*[*of f M 0 b*] **by** *simp*

lemma *borel-measurable-const-add*[*measurable* (*raw*)]:
f $\in$ *borel-measurable M* $\Longrightarrow$ $(\lambda x.\ a + f\ x :: 'a$::*real-normed-vector*$) \in$ *borel-measurable M*
using *affine-borel-measurable-vector*[*of f M a 1*] **by** *simp*

lemma *borel-measurable-inverse*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-div-algebra*
assumes *f*: $f \in$ *borel-measurable M*
shows $(\lambda x.\ inverse\ (f\ x)) \in$ *borel-measurable M*
apply (*rule measurable-compose*[*OF f*])
apply (*rule borel-measurable-continuous-countable-exceptions*[*of* $\{0\}$])
apply (*auto intro*!: *continuous-on-inverse continuous-on-id*)
done

lemma *borel-measurable-divide*[*measurable* (*raw*)]:
f $\in$ *borel-measurable M* $\Longrightarrow$ *g* $\in$ *borel-measurable M* $\Longrightarrow$
$(\lambda x.\ f\ x\ /\ g\ x :: 'b :: \{$*second-countable-topology*, *real-normed-div-algebra*$\})$ $\in$ *borel-measurable M*
by (*simp add*: *divide-inverse*)

lemma *borel-measurable-abs*[*measurable* (*raw*)]:
f ∈ *borel-measurable M* ⟹ (λ*x*. |*f x* :: *real*|) ∈ *borel-measurable M*
unfolding *abs-real-def* **by** *simp*

lemma *borel-measurable-nth*[*measurable* (*raw*)]:
(λ*x*::*real*^'*n*. *x* $ *i*) ∈ *borel-measurable borel*
by (*simp add*: *cart-eq-inner-axis*)

lemma *convex-measurable*:
fixes *A* :: '*a* :: *euclidean-space set*
shows *X* ∈ *borel-measurable M* ⟹ *X* ' *space M* ⊆ *A* ⟹ *open A* ⟹ *convex-on A q* ⟹
(λ*x*. *q* (*X x*)) ∈ *borel-measurable M*
by (*rule measurable-compose*[**where** *f*=*X* **and** *N*=*restrict-space borel A*])
(*auto intro*!: *borel-measurable-continuous-on-restrict convex-on-continuous measurable-restrict-space2*)

lemma *borel-measurable-ln*[*measurable* (*raw*)]:
assumes *f*: *f* ∈ *borel-measurable M*
shows (λ*x*. *ln* (*f x* :: *real*)) ∈ *borel-measurable M*
apply (*rule measurable-compose*[*OF f*])
apply (*rule borel-measurable-continuous-countable-exceptions*[*of* {*0*}])
apply (*auto intro*!: *continuous-on-ln continuous-on-id*)
done

lemma *borel-measurable-log*[*measurable* (*raw*)]:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹ (λ*x*. *log* (*g x*) (*f x*)) ∈ *borel-measurable M*
unfolding *log-def* **by** *auto*

lemma *borel-measurable-exp*[*measurable*]:
(*exp*::'*a*::{*real-normed-field*,*banach*}⇒'*a*) ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-at-imp-continuous-on ballI isCont-exp*)

lemma *measurable-real-floor*[*measurable*]:
(*floor* :: *real* ⇒ *int*) ∈ *measurable borel* (*count-space UNIV*)
proof −
have ⋀*a x*. ⌊*x*⌋ = *a* ⟷ (*real-of-int a* ≤ *x* ∧ *x* < *real-of-int* (*a* + *1*))
by (*auto intro*: *floor-eq2*)
then show *?thesis*
by (*auto simp*: *vimage-def measurable-count-space-eq2-countable*)
qed

lemma *measurable-real-ceiling*[*measurable*]:
(*ceiling* :: *real* ⇒ *int*) ∈ *measurable borel* (*count-space UNIV*)
unfolding *ceiling-def*[*abs-def*] **by** *simp*

lemma *borel-measurable-real-floor*: (λ*x*::*real*. *real-of-int* ⌊*x*⌋) ∈ *borel-measurable borel*

by *simp*

lemma *borel-measurable-root* [*measurable*]: *root n* ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-sqrt* [*measurable*]: *sqrt* ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-power* [*measurable* (*raw*)]:
fixes *f* :: - ⇒ *'b*::{*power*,*real-normed-algebra*}
assumes *f*: *f* ∈ *borel-measurable M*
shows (λ*x*. (*f x*) ^ *n*) ∈ *borel-measurable M*
by (*intro borel-measurable-continuous-on* [*OF - f*] *continuous-intros*)

lemma *borel-measurable-Re* [*measurable*]: *Re* ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-Im* [*measurable*]: *Im* ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-of-real* [*measurable*]: (*of-real* :: - ⇒ (-::*real-normed-algebra*))
∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-sin* [*measurable*]: (*sin* :: - ⇒ (-::{*real-normed-field*,*banach*}))
∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-cos* [*measurable*]: (*cos* :: - ⇒ (-::{*real-normed-field*,*banach*}))
∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-arctan* [*measurable*]: *arctan* ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 continuous-intros*)

lemma *borel-measurable-complex-iff*:
f ∈ *borel-measurable M* ⟷
(λ*x*. *Re* (*f x*)) ∈ *borel-measurable M* ∧ (λ*x*. *Im* (*f x*)) ∈ *borel-measurable M*
apply *auto*
apply (*subst fun-complex-eq*)
apply (*intro borel-measurable-add*)
apply *auto*
done

4.6 Borel space on the extended reals

lemma *borel-measurable-ereal*[*measurable* (*raw*)]:
assumes *f*: *f* ∈ *borel-measurable M* **shows** (λ*x*. *ereal* (*f x*)) ∈ *borel-measurable M*

using *continuous-on-ereal f* **by** (*rule borel-measurable-continuous-on*) (*rule continuous-on-id*)

lemma *borel-measurable-real-of-ereal*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow ereal$
assumes *f*: $f \in$ *borel-measurable M*
shows $(\lambda x.$ *real-of-ereal* $(f\,x)) \in$ *borel-measurable M*
apply (*rule measurable-compose*[*OF f*])
apply (*rule borel-measurable-continuous-countable-exceptions*[*of* $\{\infty, -\infty\}$])
apply (*auto intro*: *continuous-on-real simp*: *Compl-eq-Diff-UNIV*)
done

lemma *borel-measurable-ereal-cases*:
fixes $f :: 'a \Rightarrow ereal$
assumes *f*: $f \in$ *borel-measurable M*
assumes *H*: $(\lambda x.$ *H* $($*ereal* $($*real-of-ereal* $(f\,x))))) \in$ *borel-measurable M*
shows $(\lambda x.$ *H* $(f\,x)) \in$ *borel-measurable M*
proof −
let *?F* $= \lambda x.$ *if* $f\,x = \infty$ *then H* ∞ *else if* $f\,x = -\infty$ *then H* $(-\infty)$ *else H* (*ereal* (*real-of-ereal* $(f\,x))$)
{ fix *x* **have** *H* $(f\,x) =$ *?F x* **by** (*cases f x*) *auto* **}**
with *f H* **show** *?thesis* **by** *simp*
qed

lemma
fixes $f :: 'a \Rightarrow ereal$ **assumes** *f*[*measurable*]: $f \in$ *borel-measurable M*
shows *borel-measurable-ereal-abs*[*measurable*(*raw*)]: $(\lambda x.\ |f\,x|) \in$ *borel-measurable M*
and *borel-measurable-ereal-inverse*[*measurable*(*raw*)]: $(\lambda x.$ *inverse* $(f\,x) ::$ *ereal*$) \in$ *borel-measurable M*
and *borel-measurable-uminus-ereal*[*measurable*(*raw*)]: $(\lambda x. -f\,x ::$ *ereal*$) \in$ *borel-measurable M*
by (*auto simp del*: *abs-real-of-ereal simp*: *borel-measurable-ereal-cases*[*OF f*] *measurable-If*)

lemma *borel-measurable-uminus-eq-ereal*[*simp*]:
$(\lambda x. -f\,x ::$ *ereal*$) \in$ *borel-measurable M* $\longleftrightarrow$ $f \in$ *borel-measurable M* (**is** *?l* $=$ *?r*)
proof
assume *?l* **from** *borel-measurable-uminus-ereal*[*OF this*] **show** *?r* **by** *simp*
qed *auto*

lemma *set-Collect-ereal2*:
fixes $f\,g :: 'a \Rightarrow ereal$
assumes *f*: $f \in$ *borel-measurable M*
assumes *g*: $g \in$ *borel-measurable M*
assumes *H*: $\{x \in$ *space M*. *H* $($*ereal* $($*real-of-ereal* $(f\,x)))$ $($*ereal* $($*real-of-ereal* $(g\,x)))\} \in$ *sets M*
$\{x \in$ *space borel*. *H* $(-\infty)$ $($*ereal* $x)\} \in$ *sets borel*
$\{x \in$ *space borel*. *H* (∞) $($*ereal* $x)\} \in$ *sets borel*
$\{x \in$ *space borel*. *H* $($*ereal* $x)$ $(-\infty)\} \in$ *sets borel*

$\{x \in \mathit{space}\ \mathit{borel}.\ H\ (\mathit{ereal}\ x)\ (\infty)\} \in \mathit{sets}\ \mathit{borel}$
shows $\{x \in \mathit{space}\ M.\ H\ (f\ x)\ (g\ x)\} \in \mathit{sets}\ M$
proof −
let $?G = \lambda y\ x.\ \mathit{if}\ g\ x = \infty\ \mathit{then}\ H\ y\ \infty\ \mathit{else}\ \mathit{if}\ g\ x = -\infty\ \mathit{then}\ H\ y\ (-\infty)\ \mathit{else}\ H\ y\ (\mathit{ereal}\ (\mathit{real\text{-}of\text{-}ereal}\ (g\ x)))$
let $?F = \lambda x.\ \mathit{if}\ f\ x = \infty\ \mathit{then}\ ?G\ \infty\ x\ \mathit{else}\ \mathit{if}\ f\ x = -\infty\ \mathit{then}\ ?G\ (-\infty)\ x\ \mathit{else}\ ?G\ (\mathit{ereal}\ (\mathit{real\text{-}of\text{-}ereal}\ (f\ x)))\ x$
{ fix x **have** $H\ (f\ x)\ (g\ x) = ?F\ x$ **by** $(\mathit{cases}\ f\ x\ g\ x\ \mathit{rule}\text{:}\ \mathit{ereal2\text{-}cases})\ \mathit{auto}$ **}**
note $* = \mathit{this}$
from *assms* **show** *?thesis*
by $(\mathit{subst}\ *)\ (\mathit{simp}\ \mathit{del}\text{:}\ \mathit{space\text{-}borel}\ \mathit{split}\ \mathit{del}\text{:}\ \mathit{if\text{-}split})$
qed

lemma *borel-measurable-ereal-iff*:
shows $(\lambda x.\ \mathit{ereal}\ (f\ x)) \in \mathit{borel\text{-}measurable}\ M \longleftrightarrow f \in \mathit{borel\text{-}measurable}\ M$
proof
assume $(\lambda x.\ \mathit{ereal}\ (f\ x)) \in \mathit{borel\text{-}measurable}\ M$
from $\mathit{borel\text{-}measurable\text{-}real\text{-}of\text{-}ereal}[\mathit{OF}\ \mathit{this}]$
show $f \in \mathit{borel\text{-}measurable}\ M$ **by** *auto*
qed *auto*

lemma $\mathit{borel\text{-}measurable\text{-}erealD}[\mathit{measurable\text{-}dest}]$:
$(\lambda x.\ \mathit{ereal}\ (f\ x)) \in \mathit{borel\text{-}measurable}\ M \Longrightarrow g \in \mathit{measurable}\ N\ M \Longrightarrow (\lambda x.\ f\ (g\ x)) \in \mathit{borel\text{-}measurable}\ N$
unfolding *borel-measurable-ereal-iff* **by** *simp*

lemma *borel-measurable-ereal-iff-real*:
fixes $f :: 'a \Rightarrow \mathit{ereal}$
shows $f \in \mathit{borel\text{-}measurable}\ M \longleftrightarrow$
$((\lambda x.\ \mathit{real\text{-}of\text{-}ereal}\ (f\ x)) \in \mathit{borel\text{-}measurable}\ M \wedge f -` \{\infty\} \cap \mathit{space}\ M \in \mathit{sets}\ M \wedge f -` \{-\infty\} \cap \mathit{space}\ M \in \mathit{sets}\ M)$
proof *safe*
assume $*$: $(\lambda x.\ \mathit{real\text{-}of\text{-}ereal}\ (f\ x)) \in \mathit{borel\text{-}measurable}\ M\ f -` \{\infty\} \cap \mathit{space}\ M \in \mathit{sets}\ M\ f -` \{-\infty\} \cap \mathit{space}\ M \in \mathit{sets}\ M$
have $f -` \{\infty\} \cap \mathit{space}\ M = \{x \in \mathit{space}\ M.\ f\ x = \infty\}\ f -` \{-\infty\} \cap \mathit{space}\ M = \{x \in \mathit{space}\ M.\ f\ x = -\infty\}$ **by** *auto*
with $*$ **have** $**$: $\{x \in \mathit{space}\ M.\ f\ x = \infty\} \in \mathit{sets}\ M\ \{x \in \mathit{space}\ M.\ f\ x = -\infty\} \in \mathit{sets}\ M$ **by** *simp-all*
let $?f = \lambda x.\ \mathit{if}\ f\ x = \infty\ \mathit{then}\ \infty\ \mathit{else}\ \mathit{if}\ f\ x = -\infty\ \mathit{then}\ -\infty\ \mathit{else}\ \mathit{ereal}\ (\mathit{real\text{-}of\text{-}ereal}\ (f\ x))$
have $?f \in \mathit{borel\text{-}measurable}\ M$ **using** $* **$ **by** $(\mathit{intro}\ \mathit{measurable\text{-}If})\ \mathit{auto}$
also have $?f = f$ **by** $(\mathit{auto}\ \mathit{simp}\text{:}\ \mathit{fun\text{-}eq\text{-}iff}\ \mathit{ereal\text{-}real})$
finally show $f \in \mathit{borel\text{-}measurable}\ M$ **.**
qed *simp-all*

lemma *borel-measurable-ereal-iff-Iio*:
$(f :: 'a \Rightarrow \mathit{ereal}) \in \mathit{borel\text{-}measurable}\ M \longleftrightarrow (\forall a.\ f -` \{..< a\} \cap \mathit{space}\ M \in \mathit{sets}\ M)$
by $(\mathit{auto}\ \mathit{simp}\text{:}\ \mathit{borel\text{-}Iio}\ \mathit{measurable\text{-}iff\text{-}measure\text{-}of})$

lemma *borel-measurable-ereal-iff-Ioi*:
(*f*::'*a* ⇒ *ereal*) ∈ *borel-measurable M* ⟷ (∀ *a*. *f* −‘ {*a* <..} ∩ *space M* ∈ *sets M*)
by (*auto simp*: *borel-Ioi measurable-iff-measure-of*)

lemma *vimage-sets-compl-iff*:
f −‘ *A* ∩ *space M* ∈ *sets M* ⟷ *f* −‘ (− *A*) ∩ *space M* ∈ *sets M*
proof −
{ fix *A* **assume** *f* −‘ *A* ∩ *space M* ∈ *sets M*
moreover have *f* −‘ (− *A*) ∩ *space M* = *space M* − *f* −‘ *A* ∩ *space M* **by** *auto*
ultimately have *f* −‘ (− *A*) ∩ *space M* ∈ *sets M* **by** *auto* **}**
from *this*[*of A*] *this*[*of* −*A*] **show** *?thesis*
by (*metis double-complement*)
qed

lemma *borel-measurable-iff-Iic-ereal*:
(*f*::'*a*⇒*ereal*) ∈ *borel-measurable M* ⟷ (∀ *a*. *f* −‘ {..*a*} ∩ *space M* ∈ *sets M*)
unfolding *borel-measurable-ereal-iff-Ioi vimage-sets-compl-iff* [**where** *A*={*a* <..}
for *a*] **by** *simp*

lemma *borel-measurable-iff-Ici-ereal*:
(*f*::'*a* ⇒ *ereal*) ∈ *borel-measurable M* ⟷ (∀ *a*. *f* −‘ {*a*..} ∩ *space M* ∈ *sets M*)
unfolding *borel-measurable-ereal-iff-Iio vimage-sets-compl-iff* [**where** *A*={..< *a*}
for *a*] **by** *simp*

lemma *borel-measurable-ereal2*:
fixes *f g* :: '*a* ⇒ *ereal*
assumes *f*: *f* ∈ *borel-measurable M*
assumes *g*: *g* ∈ *borel-measurable M*
assumes *H*: (λ*x*. *H* (*ereal* (*real-of-ereal* (*f x*))) (*ereal* (*real-of-ereal* (*g x*)))) ∈ *borel-measurable M*
(λ*x*. *H* (−∞) (*ereal* (*real-of-ereal* (*g x*)))) ∈ *borel-measurable M*
(λ*x*. *H* (∞) (*ereal* (*real-of-ereal* (*g x*)))) ∈ *borel-measurable M*
(λ*x*. *H* (*ereal* (*real-of-ereal* (*f x*))) (−∞)) ∈ *borel-measurable M*
(λ*x*. *H* (*ereal* (*real-of-ereal* (*f x*))) (∞)) ∈ *borel-measurable M*
shows (λ*x*. *H* (*f x*) (*g x*)) ∈ *borel-measurable M*
proof −
let *?G* = λ*y x*. *if g x* = ∞ *then H y* ∞ *else if g x* = − ∞ *then H y* (−∞) *else H y* (*ereal* (*real-of-ereal* (*g x*)))
let *?F* = λ*x*. *if f x* = ∞ *then ?G* ∞ *x else if f x* = − ∞ *then ?G* (−∞) *x else ?G* (*ereal* (*real-of-ereal* (*f x*))) *x*
{ fix *x* **have** *H* (*f x*) (*g x*) = *?F x* **by** (*cases f x g x rule*: *ereal2-cases*) *auto* **}**
note ∗ = *this*
from *assms* **show** *?thesis* **unfolding** ∗ **by** *simp*
qed

lemma [*measurable*(*raw*)]:

fixes $f :: 'a \Rightarrow ereal$
assumes [*measurable*]: $f \in borel\text{-}measurable\ M\ g \in borel\text{-}measurable\ M$
shows *borel-measurable-ereal-add*: $(\lambda x.\ f\ x + g\ x) \in borel\text{-}measurable\ M$
and *borel-measurable-ereal-times*: $(\lambda x.\ f\ x * g\ x) \in borel\text{-}measurable\ M$
by (*simp-all add*: *borel-measurable-ereal2*)

lemma [*measurable*(*raw*)]:
fixes $f\ g :: 'a \Rightarrow ereal$
assumes $f \in borel\text{-}measurable\ M$
assumes $g \in borel\text{-}measurable\ M$
shows *borel-measurable-ereal-diff*: $(\lambda x.\ f\ x - g\ x) \in borel\text{-}measurable\ M$
and *borel-measurable-ereal-divide*: $(\lambda x.\ f\ x\ /\ g\ x) \in borel\text{-}measurable\ M$
using *assms* **by** (*simp-all add*: *minus-ereal-def divide-ereal-def*)

lemma *borel-measurable-ereal-setsum*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow ereal$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ \sum i{\in}S.\ f\ i\ x) \in borel\text{-}measurable\ M$
using *assms* **by** (*induction S rule*: *infinite-finite-induct*) *auto*

lemma *borel-measurable-ereal-setprod*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow ereal$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in borel\text{-}measurable\ M$
using *assms* **by** (*induction S rule*: *infinite-finite-induct*) *auto*

lemma *borel-measurable-extreal-suminf*[*measurable* (*raw*)]:
fixes $f :: nat \Rightarrow 'a \Rightarrow ereal$
assumes [*measurable*]: $\bigwedge i.\ f\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ (\sum i.\ f\ i\ x)) \in borel\text{-}measurable\ M$
unfolding *suminf-def sums-def*[*abs-def*] *lim-def*[*symmetric*] **by** *simp*

4.7 Borel space on the extended non-negative reals

ennreal is a topological monoid, so no rules for plus are required, also all order statements are usually done on type classes.

lemma *measurable-enn2ereal*[*measurable*]: $enn2ereal \in borel \rightarrow_M borel$
by (*intro borel-measurable-continuous-on1 continuous-on-enn2ereal*)

lemma *measurable-e2ennreal*[*measurable*]: $e2ennreal \in borel \rightarrow_M borel$
by (*intro borel-measurable-continuous-on1 continuous-on-e2ennreal*)

lemma *borel-measurable-enn2real*[*measurable* (*raw*)]:
$f \in M \rightarrow_M borel \Longrightarrow (\lambda x.\ enn2real\ (f\ x)) \in M \rightarrow_M borel$
unfolding *enn2real-def*[*abs-def*] **by** *measurable*

definition [*simp*]: $is\text{-}borel\ f\ M \longleftrightarrow f \in borel\text{-}measurable\ M$

lemma *is-borel-transfer*[*transfer-rule*]: *rel-fun* (*rel-fun op* = *pcr-ennreal*) *op* =

is-borel is-borel
unfolding *is-borel-def*[*abs-def*]
proof (*safe intro*!: *rel-funI ext dest*!: *rel-fun-eq-pcr-ennreal*[*THEN iffD1*])
fix *f* **and** *M* :: *'a measure*
show *f* ∈ *borel-measurable M* **if** *f*: *enn2ereal* ∘ *f* ∈ *borel-measurable M*
using *measurable-compose*[*OF f measurable-e2ennreal*] **by** *simp*
qed *simp*

context
includes *ennreal.lifting*
begin

lemma *measurable-ennreal*[*measurable*]: *ennreal* ∈ *borel* →$_M$ *borel*
unfolding *is-borel-def*[*symmetric*]
by *transfer simp*

lemma *borel-measurable-ennreal-iff*[*simp*]:
assumes [*simp*]: ⋀*x*. *x* ∈ *space M* ⟹ *0* ≤ *f x*
shows (λ*x*. *ennreal* (*f x*)) ∈ *M* →$_M$ *borel* ⟷ *f* ∈ *M* →$_M$ *borel*
proof *safe*
assume (λ*x*. *ennreal* (*f x*)) ∈ *M* →$_M$ *borel*
then have (λ*x*. *enn2real* (*ennreal* (*f x*))) ∈ *M* →$_M$ *borel*
by *measurable*
then show *f* ∈ *M* →$_M$ *borel*
by (*rule measurable-cong*[*THEN iffD1*, *rotated*]) *auto*
qed *measurable*

lemma *borel-measurable-times-ennreal*[*measurable* (*raw*)]:
fixes *f g* :: *'a* ⇒ *ennreal*
shows *f* ∈ *M* →$_M$ *borel* ⟹ *g* ∈ *M* →$_M$ *borel* ⟹ (λ*x*. *f x* ∗ *g x*) ∈ *M* →$_M$ *borel*
unfolding *is-borel-def*[*symmetric*] **by** *transfer simp*

lemma *borel-measurable-inverse-ennreal*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *ennreal*
shows *f* ∈ *M* →$_M$ *borel* ⟹ (λ*x*. *inverse* (*f x*)) ∈ *M* →$_M$ *borel*
unfolding *is-borel-def*[*symmetric*] **by** *transfer simp*

lemma *borel-measurable-divide-ennreal*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *ennreal*
shows *f* ∈ *M* →$_M$ *borel* ⟹ *g* ∈ *M* →$_M$ *borel* ⟹ (λ*x*. *f x* / *g x*) ∈ *M* →$_M$ *borel*
unfolding *divide-ennreal-def* **by** *simp*

lemma *borel-measurable-minus-ennreal*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *ennreal*
shows *f* ∈ *M* →$_M$ *borel* ⟹ *g* ∈ *M* →$_M$ *borel* ⟹ (λ*x*. *f x* − *g x*) ∈ *M* →$_M$ *borel*
unfolding *is-borel-def*[*symmetric*] **by** *transfer simp*

lemma *borel-measurable-setprod-ennreal*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow ennreal$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in borel\text{-}measurable\ M$
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in borel\text{-}measurable\ M$
using *assms* **by** (*induction S rule*: *infinite-finite-induct*) *auto*

end

hide-const (**open**) *is-borel*

4.8 LIMSEQ is borel measurable

lemma *borel-measurable-LIMSEQ-real*:
fixes $u :: nat \Rightarrow 'a \Rightarrow real$
assumes u': $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ u\ i\ x) \longrightarrow u'\ x$
and u: $\bigwedge i.\ u\ i \in borel\text{-}measurable\ M$
shows $u' \in borel\text{-}measurable\ M$
proof −
have $\bigwedge x.\ x \in space\ M \Longrightarrow liminf\ (\lambda n.\ ereal\ (u\ n\ x)) = ereal\ (u'\ x)$
using u' **by** (*simp add*: *lim-imp-Liminf*)
moreover from u **have** $(\lambda x.\ liminf\ (\lambda n.\ ereal\ (u\ n\ x))) \in borel\text{-}measurable\ M$
by *auto*
ultimately show *?thesis* **by** (*simp cong*: *measurable-cong add*: *borel-measurable-ereal-iff*)
qed

lemma *borel-measurable-LIMSEQ-metric*:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b :: metric\text{-}space$
assumes [*measurable*]: $\bigwedge i.\ f\ i \in borel\text{-}measurable\ M$
assumes *lim*: $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ f\ i\ x) \longrightarrow g\ x$
shows $g \in borel\text{-}measurable\ M$
unfolding *borel-eq-closed*
proof (*safe intro*!: *measurable-measure-of*)
fix $A :: 'b\ set$ **assume** *closed A*

have [*measurable*]: $(\lambda x.\ infdist\ (g\ x)\ A) \in borel\text{-}measurable\ M$
proof (*rule borel-measurable-LIMSEQ-real*)
show $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ infdist\ (f\ i\ x)\ A) \longrightarrow infdist\ (g\ x)\ A$
by (*intro tendsto-infdist lim*)
show $\bigwedge i.\ (\lambda x.\ infdist\ (f\ i\ x)\ A) \in borel\text{-}measurable\ M$
by (*intro borel-measurable-continuous-on*[**where** $f{=}\lambda x.\ infdist\ x\ A$]
continuous-at-imp-continuous-on ballI continuous-infdist continuous-ident)
auto
qed

show $g -` A \cap space\ M \in sets\ M$
proof *cases*
assume $A \neq \{\}$
then have $\bigwedge x.\ infdist\ x\ A = 0 \longleftrightarrow x \in A$

```
    using ‹closed A› by (simp add: in-closed-iff-infdist-zero)
  then have g -‘ A ∩ space M = {x∈space M. infdist (g x) A = 0}
    by auto
  also have ... ∈ sets M
    by measurable
  finally show ?thesis .
 qed simp
qed auto

lemma sets-Collect-Cauchy[measurable]:
 fixes f :: nat ⇒ 'a => 'b::{metric-space, second-countable-topology}
 assumes f[measurable]: ⋀i. f i ∈ borel-measurable M
 shows {x∈space M. Cauchy (λi. f i x)} ∈ sets M
 unfolding metric-Cauchy-iff2 using f by auto

lemma borel-measurable-lim-metric[measurable (raw)]:
 fixes f :: nat ⇒ 'a ⇒ 'b::{banach, second-countable-topology}
 assumes f[measurable]: ⋀i. f i ∈ borel-measurable M
 shows (λx. lim (λi. f i x)) ∈ borel-measurable M
proof -
 def u' ≡ λx. lim (λi. if Cauchy (λi. f i x) then f i x else 0)
 then have ∗: ⋀x. lim (λi. f i x) = (if Cauchy (λi. f i x) then u' x else (THE x.
False))
   by (auto simp: lim-def convergent-eq-cauchy[symmetric])
 have u' ∈ borel-measurable M
 proof (rule borel-measurable-LIMSEQ-metric)
   fix x
   have convergent (λi. if Cauchy (λi. f i x) then f i x else 0)
     by (cases Cauchy (λi. f i x))
        (auto simp add: convergent-eq-cauchy[symmetric] convergent-def)
   then show (λi. if Cauchy (λi. f i x) then f i x else 0) ⟶ u' x
     unfolding u'-def
     by (rule convergent-LIMSEQ-iff[THEN iffD1])
 qed measurable
 then show ?thesis
   unfolding ∗ by measurable
qed

lemma borel-measurable-suminf[measurable (raw)]:
 fixes f :: nat ⇒ 'a ⇒ 'b::{banach, second-countable-topology}
 assumes f[measurable]: ⋀i. f i ∈ borel-measurable M
 shows (λx. suminf (λi. f i x)) ∈ borel-measurable M
 unfolding suminf-def sums-def[abs-def] lim-def[symmetric] by simp


lemma isCont-borel:
 fixes f :: 'b::metric-space ⇒ 'a::metric-space
 shows {x. isCont f x} ∈ sets borel
proof -
```

let *?I* = λj. *inverse*(*real* (*Suc j*))

{ fix *x*
have *isCont f x* = ($\forall i.\ \exists j.\ \forall y\ z.$ *dist x y* < *?I j* $\wedge$ *dist x z* < *?I j* $\longrightarrow$ *dist* (*f y*) (*f z*) $\leq$ *?I i*)
unfolding *continuous-at-eps-delta*
proof *safe*
fix *i* **assume** $\forall e{>}0.\ \exists d{>}0.\ \forall y.$ *dist y x* < *d* $\longrightarrow$ *dist* (*f y*) (*f x*) < *e*
moreover have *0* < *?I i / 2*
by *simp*
ultimately obtain *d* **where** *d*: *0* < *d* $\bigwedge y.$ *dist x y* < *d* $\Longrightarrow$ *dist* (*f y*) (*f x*) < *?I i / 2*
by (*metis dist-commute*)
then obtain *j* **where** *j*: *?I j* < *d*
by (*metis reals-Archimedean*)

show $\exists j.\ \forall y\ z.$ *dist x y* < *?I j* $\wedge$ *dist x z* < *?I j* $\longrightarrow$ *dist* (*f y*) (*f z*) $\leq$ *?I i*
proof (*safe intro*!: *exI*[**where** *x*=*j*])
fix *y z* **assume** $*$: *dist x y* < *?I j dist x z* < *?I j*
have *dist* (*f y*) (*f z*) $\leq$ *dist* (*f y*) (*f x*) + *dist* (*f z*) (*f x*)
by (*rule dist-triangle2*)
also have ... < *?I i / 2* + *?I i / 2*
by (*intro add-strict-mono d less-trans*[*OF - j*] $*$)
also have ... $\leq$ *?I i*
by (*simp add*: *field-simps of-nat-Suc*)
finally show *dist* (*f y*) (*f z*) $\leq$ *?I i*
by *simp*
qed
next
fix *e*::*real* **assume** *0* < *e*
then obtain *n* **where** *n*: *?I n* < *e*
by (*metis reals-Archimedean*)
assume $\forall i.\ \exists j.\ \forall y\ z.$ *dist x y* < *?I j* $\wedge$ *dist x z* < *?I j* $\longrightarrow$ *dist* (*f y*) (*f z*) $\leq$ *?I i*
from *this*[*THEN spec*, *of Suc n*]
obtain *j* **where** *j*: $\bigwedge y\ z.$ *dist x y* < *?I j* $\Longrightarrow$ *dist x z* < *?I j* $\Longrightarrow$ *dist* (*f y*) (*f z*) $\leq$ *?I* (*Suc n*)
by *auto*

show $\exists d{>}0.\ \forall y.$ *dist y x* < *d* $\longrightarrow$ *dist* (*f y*) (*f x*) < *e*
proof (*safe intro*!: *exI*[*of - ?I j*])
fix *y* **assume** *dist y x* < *?I j*
then have *dist* (*f y*) (*f x*) $\leq$ *?I* (*Suc n*)
by (*intro j*) (*auto simp*: *dist-commute*)
also have *?I* (*Suc n*) < *?I n*
by *simp*
also note *n*
finally show *dist* (*f y*) (*f x*) < *e* **.**
qed *simp*

```
    qed }
  note ∗ = this

  have ∗∗: ⋀e y. open {x. dist x y < e}
    using open-ball by (simp-all add: ball-def dist-commute)

  have {x∈space borel. isCont f x} ∈ sets borel
    unfolding ∗
    apply (intro sets.sets-Collect-countable-All sets.sets-Collect-countable-Ex)
    apply (simp add: Collect-all-eq)
    apply (intro borel-closed closed-INT ballI closed-Collect-imp open-Collect-conj
∗∗)
    apply auto
    done
  then show ?thesis
    by simp
qed

lemma isCont-borel-pred[measurable]:
  fixes f :: 'b::metric-space ⇒ 'a::metric-space
  shows Measurable.pred borel (isCont f)
  unfolding pred-def by (simp add: isCont-borel)

lemma is-real-interval:
  assumes S: is-interval S
  shows ∃ a b::real. S = {} ∨ S = UNIV ∨ S = {..<b} ∨ S = {..b} ∨ S = {a<..}
∨ S = {a..} ∨
    S = {a<..<b} ∨ S = {a<..b} ∨ S = {a..<b} ∨ S = {a..b}
  using S unfolding is-interval-1 by (blast intro: interval-cases)

lemma real-interval-borel-measurable:
  assumes is-interval (S::real set)
  shows S ∈ sets borel
proof −
  from assms is-real-interval have ∃ a b::real. S = {} ∨ S = UNIV ∨ S = {..<b}
∨ S = {..b} ∨
    S = {a<..} ∨ S = {a..} ∨ S = {a<..<b} ∨ S = {a<..b} ∨ S = {a..<b} ∨ S
= {a..b} by auto
  then guess a ..
  then guess b ..
  thus ?thesis
    by auto
qed

lemma borel-measurable-mono-on-fnc:
  fixes f :: real ⇒ real and A :: real set
  assumes mono-on f A
  shows f ∈ borel-measurable (restrict-space borel A)
  apply (rule measurable-restrict-countable[OF mono-on-ctble-discont[OF assms]])
```

apply (*auto intro*!: *image-eqI*[**where** *x*={*x*} **for** *x*] *simp*: *sets-restrict-space*)
apply (*auto simp add*: *sets-restrict-restrict-space continuous-on-eq-continuous-within*
cong: *measurable-cong-sets*
intro!: *borel-measurable-continuous-on-restrict intro*: *continuous-within-subset*)
done

lemma *borel-measurable-mono*:
fixes *f* :: *real* ⇒ *real*
shows *mono f* ⟹ *f* ∈ *borel-measurable borel*
using *borel-measurable-mono-on-fnc*[*of f UNIV*] **by** (*simp add*: *mono-def mono-on-def*)

no-notation
eucl-less (**infix** <*e* 50)

end

5 Lebesgue Integration for Nonnegative Functions

theory *Nonnegative-Lebesgue-Integration*
imports *Measure-Space Borel-Space*
begin

5.1 Simple function

Our simple functions are not restricted to nonnegative real numbers. Instead they are just functions with a finite range and are measurable when singleton sets are measurable.

definition *simple-function M g* ⟷
finite (*g* ‘ *space M*) ∧
(∀ *x* ∈ *g* ‘ *space M*. *g* −‘ {*x*} ∩ *space M* ∈ *sets M*)

lemma *simple-functionD*:
assumes *simple-function M g*
shows *finite* (*g* ‘ *space M*) **and** *g* −‘ *X* ∩ *space M* ∈ *sets M*
proof −
show *finite* (*g* ‘ *space M*)
using *assms* **unfolding** *simple-function-def* **by** *auto*
have *g* −‘ *X* ∩ *space M* = *g* −‘ (*X* ∩ *g*‘*space M*) ∩ *space M* **by** *auto*
also have ... = (⋃ *x*∈*X* ∩ *g*‘*space M*. *g*−‘{*x*} ∩ *space M*) **by** *auto*
finally show *g* −‘ *X* ∩ *space M* ∈ *sets M* **using** *assms*
by (*auto simp del*: *UN-simps simp*: *simple-function-def*)
qed

lemma *measurable-simple-function*[*measurable-dest*]:
simple-function M f ⟹ *f* ∈ *measurable M* (*count-space UNIV*)
unfolding *simple-function-def measurable-def*
proof *safe*

fix A **assume** *finite* $(f$ ‘ *space* $M)$ $\forall x{\in}f$ ‘ *space* $M.$ $f -$‘ $\{x\} \cap$ *space* $M \in$ *sets* M
then have $(\bigcup x{\in}f$ ‘ *space* $M.$ *if* $x \in A$ *then* $f -$‘ $\{x\} \cap$ *space* M *else* $\{\}) \in$ *sets* M
by (*intro sets.finite-UN*) *auto*
also have $(\bigcup x{\in}f$ ‘ *space* $M.$ *if* $x \in A$ *then* $f -$‘ $\{x\} \cap$ *space* M *else* $\{\}) = f -$‘ $A \cap$ *space* M
by (*auto split*: *if-split-asm*)
finally show $f -$‘ $A \cap$ *space* $M \in$ *sets* M **.**
qed *simp*

lemma *borel-measurable-simple-function*:
simple-function M $f \Longrightarrow f \in$ *borel-measurable* M
by (*auto dest*!: *measurable-simple-function simp*: *measurable-def*)

lemma *simple-function-measurable2*[*intro*]:
assumes *simple-function* M f *simple-function* M g
shows $f -$‘ $A \cap g -$‘ $B \cap$ *space* $M \in$ *sets* M
proof −
have $f -$‘ $A \cap g -$‘ $B \cap$ *space* $M = (f -$‘ $A \cap$ *space* $M) \cap (g -$‘ $B \cap$ *space* $M)$
by *auto*
then show *?thesis* **using** *assms*[*THEN simple-functionD*(*2*)] **by** *auto*
qed

lemma *simple-function-indicator-representation*:
fixes $f ::'a \Rightarrow$ *ennreal*
assumes f: *simple-function* M f **and** x: $x \in$ *space* M
shows f $x = (\sum y \in f$ ‘ *space* $M.$ $y *$ *indicator* $(f -$‘ $\{y\} \cap$ *space* $M)$ $x)$
(**is** *?l* = *?r*)
proof −
have *?r* $= (\sum y \in f$ ‘ *space* $M.$
(*if* $y = f$ x *then* $y *$ *indicator* $(f -$‘ $\{y\} \cap$ *space* $M)$ x *else* *0*))
by (*auto intro*!: *setsum.cong*)
also have ... = f $x *$ *indicator* $(f -$‘ $\{f$ $x\} \cap$ *space* $M)$ x
using *assms* **by** (*auto dest*: *simple-functionD simp*: *setsum.delta*)
also have ... = f x **using** x **by** (*auto simp*: *indicator-def*)
finally show *?thesis* **by** *auto*
qed

lemma *simple-function-notspace*:
simple-function M $(\lambda x.$ h $x *$ *indicator* $(-$ *space* $M)$ x::*ennreal*) (**is** *simple-function* M *?h*)
proof −
have *?h* ‘ *space* $M \subseteq \{0\}$ **unfolding** *indicator-def* **by** *auto*
hence [*simp*, *intro*]: *finite* (*?h* ‘ *space* M) **by** (*auto intro*: *finite-subset*)
have *?h* $-$‘ $\{0\} \cap$ *space* $M =$ *space* M **by** *auto*
thus *?thesis* **unfolding** *simple-function-def* **by** *auto*
qed

lemma *simple-function-cong*:
assumes $\bigwedge t.\ t \in space\ M \Longrightarrow f\ t = g\ t$
shows *simple-function M f* $\longleftrightarrow$ *simple-function M g*
proof −
have $\bigwedge x.\ f\ -`\ \{x\} \cap space\ M = g\ -`\ \{x\} \cap space\ M$
using *assms* **by** *auto*
with *assms* **show** *?thesis*
by (*simp add*: *simple-function-def cong*: *image-cong*)
qed

lemma *simple-function-cong-algebra*:
assumes *sets N* = *sets M space N* = *space M*
shows *simple-function M f* $\longleftrightarrow$ *simple-function N f*
unfolding *simple-function-def assms* **..**

lemma *simple-function-borel-measurable*:
fixes $f :: 'a \Rightarrow 'x$::{*t2-space*}
assumes $f \in$ *borel-measurable M* **and** *finite* (f ‘ *space M*)
shows *simple-function M f*
using *assms* **unfolding** *simple-function-def*
by (*auto intro*: *borel-measurable-vimage*)

lemma *simple-function-iff-borel-measurable*:
fixes $f :: 'a \Rightarrow 'x$::{*t2-space*}
shows *simple-function M f* $\longleftrightarrow$ *finite* (f ‘ *space M*) $\wedge$ $f \in$ *borel-measurable M*
by (*metis borel-measurable-simple-function simple-functionD*(*1*) *simple-function-borel-measurable*)

lemma *simple-function-eq-measurable*:
simple-function M f $\longleftrightarrow$ *finite* (*f*‘*space M*) $\wedge$ $f \in$ *measurable M* (*count-space UNIV*)
using *measurable-simple-function*[*of M f*] **by** (*fastforce simp*: *simple-function-def*)

lemma *simple-function-const*[*intro*, *simp*]:
simple-function M ($\lambda x.\ c$)
by (*auto intro*: *finite-subset simp*: *simple-function-def*)
lemma *simple-function-compose*[*intro*, *simp*]:
assumes *simple-function M f*
shows *simple-function M* ($g \circ f$)
unfolding *simple-function-def*
proof *safe*
show *finite* (($g \circ f$) ‘ *space M*)
using *assms* **unfolding** *simple-function-def* **by** (*auto simp*: *image-comp* [*symmetric*])
next
fix x **assume** $x \in$ *space M*
let *?G* = $g\ -`\ \{g\ (f\ x)\} \cap$ (*f*‘*space M*)
have $*$: ($g \circ f$) $-`\ \{(g \circ f)\ x\} \cap$ *space M* =
($\bigcup x \in ?G.\ f\ -`\ \{x\} \cap$ *space M*) **by** *auto*
show ($g \circ f$) $-`\ \{(g \circ f)\ x\} \cap$ *space M* $\in$ *sets M*
using *assms* **unfolding** *simple-function-def* $*$

by (*rule-tac sets.finite-UN*) *auto*
qed

lemma *simple-function-indicator*[*intro*, *simp*]:
assumes $A \in$ *sets M*
shows *simple-function M* (*indicator A*)
proof −
have *indicator A* ‘ *space M* $\subseteq \{0, 1\}$ (**is** *?S* $\subseteq$ -)
by (*auto simp*: *indicator-def*)
hence *finite ?S* **by** (*rule finite-subset*) *simp*
moreover have $-$ *A* $\cap$ *space M* = *space M* − *A* **by** *auto*
ultimately show *?thesis* **unfolding** *simple-function-def*
using *assms* **by** (*auto simp*: *indicator-def* [*abs-def*])
qed

lemma *simple-function-Pair*[*intro*, *simp*]:
assumes *simple-function M f*
assumes *simple-function M g*
shows *simple-function M* (λx. (*f x*, *g x*)) (**is** *simple-function M ?p*)
unfolding *simple-function-def*
proof *safe*
show *finite* (*?p* ‘ *space M*)
using *assms* **unfolding** *simple-function-def*
by (*rule-tac finite-subset*[*of - f‘space M* × *g‘space M*]) *auto*
next
fix *x* **assume** *x* ∈ *space M*
have (λx. (*f x*, *g x*)) −‘ {(*f x*, *g x*)} ∩ *space M* =
(*f* −‘ {*f x*} ∩ *space M*) ∩ (*g* −‘ {*g x*} ∩ *space M*)
by *auto*
with ‹*x* ∈ *space M*› **show** (λx. (*f x*, *g x*)) −‘ {(*f x*, *g x*)} ∩ *space M* ∈ *sets M*
using *assms* **unfolding** *simple-function-def* **by** *auto*
qed

lemma *simple-function-compose1*:
assumes *simple-function M f*
shows *simple-function M* (λx. *g* (*f x*))
using *simple-function-compose*[*OF assms*, *of g*]
by (*simp add*: *comp-def*)

lemma *simple-function-compose2*:
assumes *simple-function M f* **and** *simple-function M g*
shows *simple-function M* (λx. *h* (*f x*) (*g x*))
proof −
have *simple-function M* (($\lambda(x, y)$. *h x y*) ∘ (λx. (*f x*, *g x*)))
using *assms* **by** *auto*
thus *?thesis* **by** (*simp-all add*: *comp-def*)
qed

lemmas *simple-function-add*[*intro*, *simp*] = *simple-function-compose2*[**where** *h*=*op*

+]
and *simple-function-diff*[*intro*, *simp*] = *simple-function-compose2*[**where** *h*=*op*
−]
and *simple-function-uminus*[*intro*, *simp*] = *simple-function-compose*[**where** *g*=*uminus*]
and *simple-function-mult*[*intro*, *simp*] = *simple-function-compose2*[**where** *h*=*op*
∗]
and *simple-function-div*[*intro*, *simp*] = *simple-function-compose2*[**where** *h*=*op*
/]
and *simple-function-inverse*[*intro*, *simp*] = *simple-function-compose*[**where** *g*=*inverse*]
and *simple-function-max*[*intro*, *simp*] = *simple-function-compose2*[**where** *h*=*max*]

lemma *simple-function-setsum*[*intro*, *simp*]:
assumes ⋀*i*. *i* ∈ *P* ⟹ *simple-function M* (*f i*)
shows *simple-function M* (λ*x*. ∑ *i*∈*P*. *f i x*)
proof *cases*
assume *finite P* **from** *this assms* **show** *?thesis* **by** *induct auto*
qed *auto*

lemma *simple-function-ennreal*[*intro*, *simp*]:
fixes *f g* :: *'a* ⇒ *real* **assumes** *sf*: *simple-function M f*
shows *simple-function M* (λ*x*. *ennreal* (*f x*))
by (*rule simple-function-compose1*[*OF sf*])

lemma *simple-function-real-of-nat*[*intro*, *simp*]:
fixes *f g* :: *'a* ⇒ *nat* **assumes** *sf*: *simple-function M f*
shows *simple-function M* (λ*x*. *real* (*f x*))
by (*rule simple-function-compose1*[*OF sf*])

lemma *borel-measurable-implies-simple-function-sequence*:
fixes *u* :: *'a* ⇒ *ennreal*
assumes *u*[*measurable*]: *u* ∈ *borel-measurable M*
shows ∃*f*. *incseq f* ∧ (∀ *i*. (∀ *x*. *f i x* < *top*) ∧ *simple-function M* (*f i*)) ∧ *u* =
(*SUP i*. *f i*)
proof −
def *f* ≡ λ*i x*. *real-of-int* (*floor* (*enn2real* (*min i* (*u x*)) ∗ *2^i*)) / *2^i*

have [*simp*]: *0* ≤ *f i x* **for** *i x*
by (*auto simp*: *f-def intro*!: *divide-nonneg-nonneg mult-nonneg-nonneg enn2real-nonneg*)

have ∗: *2^n* ∗ *real-of-int x* = *real-of-int* (*2^n* ∗ *x*) **for** *n x*
by *simp*

have *real-of-int* ⌊*real i* ∗ *2* ^ *i*⌋ = *real-of-int* ⌊*i* ∗ *2* ^ *i*⌋ **for** *i*
by (*intro arg-cong*[**where** *f*=*real-of-int*]) *simp*
then have [*simp*]: *real-of-int* ⌊*real i* ∗ *2* ^ *i*⌋ = *i* ∗ *2* ^ *i* **for** *i*
unfolding *floor-of-nat* **by** *simp*

have *incseq f*
proof (*intro monoI le-funI*)

```
  fix m n :: nat and x assume m ≤ n
  moreover
  { fix d :: nat
    have ⌊2^d::real⌋ * ⌊2^m * enn2real (min (of-nat m) (u x))⌋ ≤
      ⌊2^d * (2^m * enn2real (min (of-nat m) (u x)))⌋
      by (rule le-mult-floor) (auto simp: enn2real-nonneg)
   also have ... ≤ ⌊2^d * (2^m * enn2real (min (of-nat d + of-nat m) (u x)))⌋
      by (intro floor-mono mult-mono enn2real-mono min.mono)
         (auto simp: enn2real-nonneg min-less-iff-disj of-nat-less-top)
    finally have f m x ≤ f (m + d) x
      unfolding f-def
      by (auto simp: field-simps power-add * simp del: of-int-mult) }
  ultimately show f m x ≤ f n x
    by (auto simp add: le-iff-add)
qed
then have inc-f: incseq (λi. ennreal (f i x)) for x
  by (auto simp: incseq-def le-fun-def)
then have incseq (λi x. ennreal (f i x))
  by (auto simp: incseq-def le-fun-def)
moreover
have simple-function M (f i) for i
proof (rule simple-function-borel-measurable)
  have ⌊enn2real (min (of-nat i) (u x)) * 2 ^ i⌋ ≤ ⌊int i * 2 ^ i⌋ for x
    by (cases u x rule: ennreal-cases)
       (auto split: split-min intro!: floor-mono)
  then have f i ‘ space M ⊆ (λn. real-of-int n / 2^i) ‘ {0 .. of-nat i * 2^i}
    unfolding floor-of-int by (auto simp: f-def enn2real-nonneg intro!: imageI)
  then show finite (f i ‘ space M)
    by (rule finite-subset) auto
  show f i ∈ borel-measurable M
    unfolding f-def enn2real-def by measurable
qed
moreover
{ fix x
  have (SUP i. ennreal (f i x)) = u x
  proof (cases u x rule: ennreal-cases)
    case top then show ?thesis
    by (simp add: f-def inf-min[symmetric] ennreal-of-nat-eq-real-of-nat[symmetric]
                  ennreal-SUP-of-nat-eq-top)
  next
    case (real r)
    obtain n where r ≤ of-nat n using real-arch-simple by auto
    then have min-eq-r: ∀F x in sequentially. min (real x) r = r
      by (auto simp: eventually-sequentially intro!: exI[of - n] split: split-min)

    have (λi. real-of-int ⌊min (real i) r * 2^i⌋ / 2^i) ⟶ r
    proof (rule tendsto-sandwich)
      show (λn. r - (1/2)^n) ⟶ r
        by (auto intro!: tendsto-eq-intros LIMSEQ-power-zero)
```

```
      show ∀F n in sequentially. real-of-int ⌊min (real n) r * 2 ^ n⌋ / 2 ^ n ≤ r
        using min-eq-r by eventually-elim (auto simp: field-simps)
       have *: r * (2 ^ n * 2 ^ n) ≤ 2^n + 2^n * real-of-int ⌊r * 2 ^ n⌋ for n
         using real-of-int-floor-ge-diff-one[of r * 2^n, THEN mult-left-mono, of
2^n]
        by (auto simp: field-simps)
       show ∀F n in sequentially. r − (1/2) ^n ≤ real-of-int ⌊min (real n) r * 2
^ n⌋ / 2 ^ n
         using min-eq-r by eventually-elim (insert *, auto simp: field-simps)
     qed auto
     then have (λi. ennreal (f i x)) ⟶ ennreal r
       by (simp add: real f-def ennreal-of-nat-eq-real-of-nat min-ennreal)
     from LIMSEQ-unique[OF LIMSEQ-SUP[OF inc-f] this]
     show ?thesis
       by (simp add: real)
   qed }
 ultimately show ?thesis
   by (intro exI[of - λi x. ennreal (f i x)]) auto
qed

lemma borel-measurable-implies-simple-function-sequence':
 fixes u :: 'a ⇒ ennreal
 assumes u: u ∈ borel-measurable M
 obtains f where
   ⋀i. simple-function M (f i) incseq f ⋀i x. f i x < top ⋀x. (SUP i. f i x) = u x
 using borel-measurable-implies-simple-function-sequence[OF u] by (auto simp:
fun-eq-iff) blast

lemma simple-function-induct[consumes 1, case-names cong set mult add, induct
set: simple-function]:
 fixes u :: 'a ⇒ ennreal
 assumes u: simple-function M u
 assumes cong: ⋀f g. simple-function M f ⟹ simple-function M g ⟹ (AE x
in M. f x = g x) ⟹ P f ⟹ P g
 assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
 assumes mult: ⋀u c. P u ⟹ P (λx. c * u x)
 assumes add: ⋀u v. P u ⟹ P v ⟹ P (λx. v x + u x)
 shows P u
proof (rule cong)
 from AE-space show AE x in M. (∑ y∈u ' space M. y * indicator (u −' {y}
∩ space M) x) = u x
 proof eventually-elim
   fix x assume x: x ∈ space M
   from simple-function-indicator-representation[OF u x]
   show (∑ y∈u ' space M. y * indicator (u −' {y} ∩ space M) x) = u x ..
 qed
next
 from u have finite (u ' space M)
   unfolding simple-function-def by auto
```

```
  then show P (λx. ∑y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x)
  proof induct
    case empty show ?case
      using set[of {}] by (simp add: indicator-def[abs-def])
  qed (auto intro!: add mult set simple-functionD u)
next
  show simple-function M (λx. (∑y∈u ‘ space M. y * indicator (u -‘ {y} ∩
space M) x))
    apply (subst simple-function-cong)
    apply (rule simple-function-indicator-representation[symmetric])
    apply (auto intro: u)
    done
qed fact

lemma simple-function-induct-nn[consumes 1, case-names cong set mult add]:
  fixes u :: 'a ⇒ ennreal
  assumes u: simple-function M u
  assumes cong: ⋀f g. simple-function M f ⟹ simple-function M g ⟹ (⋀x. x
∈ space M ⟹ f x = g x) ⟹ P f ⟹ P g
  assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
  assumes mult: ⋀u c. simple-function M u ⟹ P u ⟹ P (λx. c * u x)
  assumes add: ⋀u v. simple-function M u ⟹ P u ⟹ simple-function M v ⟹
(⋀x. x ∈ space M ⟹ u x = 0 ∨ v x = 0) ⟹ P v ⟹ P (λx. v x + u x)
  shows P u
proof -
  show ?thesis
  proof (rule cong)
    fix x assume x: x ∈ space M
    from simple-function-indicator-representation[OF u x]
    show (∑y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x) = u x ..
  next
    show simple-function M (λx. (∑y∈u ‘ space M. y * indicator (u -‘ {y} ∩
space M) x))
      apply (subst simple-function-cong)
      apply (rule simple-function-indicator-representation[symmetric])
      apply (auto intro: u)
      done
  next
    from u have finite (u ‘ space M)
      unfolding simple-function-def by auto
    then show P (λx. ∑y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x)
    proof induct
      case empty show ?case
        using set[of {}] by (simp add: indicator-def[abs-def])
    next
      case (insert x S)
      { fix z have (∑y∈S. y * indicator (u -‘ {y} ∩ space M) z) = 0 ∨
          x * indicator (u -‘ {x} ∩ space M) z = 0
          using insert by (subst setsum-eq-0-iff) (auto simp: indicator-def) }
```

note *disj* = *this*
from *insert* **show** *?case*
by (*auto intro*!: *add mult set simple-functionD u simple-function-setsum disj*)
qed
qed *fact*
qed

lemma *borel-measurable-induct*[*consumes 1*, *case-names cong set mult add seq*, *induct set*: *borel-measurable*]:
fixes *u* :: *'a* ⇒ *ennreal*
assumes *u*: *u* ∈ *borel-measurable M*
assumes *cong*: ⋀*f g*. *f* ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹ (⋀*x*. *x* ∈ *space M* ⟹ *f x* = *g x*) ⟹ *P g* ⟹ *P f*
assumes *set*: ⋀*A*. *A* ∈ *sets M* ⟹ *P* (*indicator A*)
assumes *mult'*: ⋀*u c*. *c* < *top* ⟹ *u* ∈ *borel-measurable M* ⟹ (⋀*x*. *x* ∈ *space M* ⟹ *u x* < *top*) ⟹ *P u* ⟹ *P* (λ*x*. *c* ∗ *u x*)
assumes *add*: ⋀*u v*. *u* ∈ *borel-measurable M* ⟹ (⋀*x*. *x* ∈ *space M* ⟹ *u x* < *top*) ⟹ *P u* ⟹ *v* ∈ *borel-measurable M* ⟹ (⋀*x*. *x* ∈ *space M* ⟹ *v x* < *top*) ⟹ (⋀*x*. *x* ∈ *space M* ⟹ *u x* = *0* ∨ *v x* = *0*) ⟹ *P v* ⟹ *P* (λ*x*. *v x* + *u x*)
assumes *seq*: ⋀*U*. (⋀*i*. *U i* ∈ *borel-measurable M*) ⟹ (⋀*i x*. *x* ∈ *space M* ⟹ *U i x* < *top*) ⟹ (⋀*i*. *P* (*U i*)) ⟹ *incseq U* ⟹ *u* = (*SUP i*. *U i*) ⟹ *P* (*SUP i*. *U i*)
shows *P u*
using *u*
proof (*induct rule*: *borel-measurable-implies-simple-function-sequence'*)
fix *U* **assume** *U*: ⋀*i*. *simple-function M* (*U i*) *incseq U* ⋀*i x*. *U i x* < *top* **and** *sup*: ⋀*x*. (*SUP i*. *U i x*) = *u x*
have *u-eq*: *u* = (*SUP i*. *U i*)
using *u sup* **by** *auto*

have *not-inf*: ⋀*x i*. *x* ∈ *space M* ⟹ *U i x* < *top*
using *U* **by** (*auto simp*: *image-iff eq-commute*)

from *U* **have** ⋀*i*. *U i* ∈ *borel-measurable M*
by (*simp add*: *borel-measurable-simple-function*)

show *P u*
unfolding *u-eq*
proof (*rule seq*)
fix *i* **show** *P* (*U i*)
using ‹*simple-function M* (*U i*)› *not-inf*[*of - i*]
proof (*induct rule*: *simple-function-induct-nn*)
case (*mult u c*)
show *?case*
proof *cases*
assume *c* = *0* ∨ *space M* = {} ∨ (∀*x*∈*space M*. *u x* = *0*)
with *mult*(*1*) **show** *?thesis*
by (*intro cong*[*of* λ*x*. *c* ∗ *u x indicator* {}] *set*)

```
        (auto dest!: borel-measurable-simple-function)
    next
      assume ¬ (c = 0 ∨ space M = {} ∨ (∀x∈space M. u x = 0))
      then obtain x where space M ≠ {} and x: x ∈ space M u x ≠ 0 c ≠ 0
        by auto
      with mult(3)[of x] have c < top
        by (auto simp: ennreal-mult-less-top)
      then have u-fin: x' ∈ space M ⟹ u x' < top for x'
        using mult(3)[of x'] ‹c ≠ 0› by (auto simp: ennreal-mult-less-top)
      then have P u
        by (rule mult)
      with u-fin ‹c < top› mult(1) show ?thesis
        by (intro mult') (auto dest!: borel-measurable-simple-function)
    qed
  qed (auto intro: cong intro!: set add dest!: borel-measurable-simple-function)
 qed fact+
qed

lemma simple-function-If-set:
  assumes sf: simple-function M f simple-function M g and A: A ∩ space M ∈
sets M
  shows simple-function M (λx. if x ∈ A then f x else g x) (is simple-function M
?IF)
proof −
  def F ≡ λx. f −‘ {x} ∩ space M and G ≡ λx. g −‘ {x} ∩ space M
  show ?thesis unfolding simple-function-def
  proof safe
    have ?IF ‘ space M ⊆ f ‘ space M ∪ g ‘ space M by auto
    from finite-subset[OF this] assms
    show finite (?IF ‘ space M) unfolding simple-function-def by auto
  next
    fix x assume x ∈ space M
    then have ∗: ?IF −‘ {?IF x} ∩ space M = (if x ∈ A
      then ((F (f x) ∩ (A ∩ space M)) ∪ (G (f x) − (G (f x) ∩ (A ∩ space M))))
      else ((F (g x) ∩ (A ∩ space M)) ∪ (G (g x) − (G (g x) ∩ (A ∩ space M)))))
     using sets.sets-into-space[OF A] by (auto split: if-split-asm simp: G-def F-def)
    have [intro]: ⋀x. F x ∈ sets M ⋀x. G x ∈ sets M
      unfolding F-def G-def using sf[THEN simple-functionD(2)] by auto
    show ?IF −‘ {?IF x} ∩ space M ∈ sets M unfolding ∗ using A by auto
  qed
qed

lemma simple-function-If:
  assumes sf: simple-function M f simple-function M g and P: {x∈space M. P
x} ∈ sets M
  shows simple-function M (λx. if P x then f x else g x)
proof −
  have {x∈space M. P x} = {x. P x} ∩ space M by auto
  with simple-function-If-set[OF sf, of {x. P x}] P show ?thesis by simp
```

qed

lemma *simple-function-subalgebra*:
assumes *simple-function N f*
and *N-subalgebra*: *sets N* ⊆ *sets M space N* = *space M*
shows *simple-function M f*
using *assms* **unfolding** *simple-function-def* **by** *auto*

lemma *simple-function-comp*:
assumes *T*: *T* ∈ *measurable M M′*
and *f*: *simple-function M′ f*
shows *simple-function M* (λ*x*. *f* (*T x*))
proof (*intro simple-function-def*[*THEN iffD2*] *conjI ballI*)
have (λ*x*. *f* (*T x*)) ‘ *space M* ⊆ *f* ‘ *space M′*
using *T* **unfolding** *measurable-def* **by** *auto*
then show *finite* ((λ*x*. *f* (*T x*)) ‘ *space M*)
using *f* **unfolding** *simple-function-def* **by** (*auto intro*: *finite-subset*)
fix *i* **assume** *i*: *i* ∈ (λ*x*. *f* (*T x*)) ‘ *space M*
then have *i* ∈ *f* ‘ *space M′*
using *T* **unfolding** *measurable-def* **by** *auto*
then have *f* −‘ {*i*} ∩ *space M′* ∈ *sets M′*
using *f* **unfolding** *simple-function-def* **by** *auto*
then have *T* −‘ (*f* −‘ {*i*} ∩ *space M′*) ∩ *space M* ∈ *sets M*
using *T* **unfolding** *measurable-def* **by** *auto*
also have *T* −‘ (*f* −‘ {*i*} ∩ *space M′*) ∩ *space M* = (λ*x*. *f* (*T x*)) −‘ {*i*} ∩ *space M*
using *T* **unfolding** *measurable-def* **by** *auto*
finally show (λ*x*. *f* (*T x*)) −‘ {*i*} ∩ *space M* ∈ *sets M* .
qed

5.2 Simple integral

definition *simple-integral* :: *′a measure* ⇒ (*′a* ⇒ *ennreal*) ⇒ *ennreal* (*integral*^{S})
where
integral^S *M f* = (∑ *x* ∈ *f* ‘ *space M*. *x* ∗ *emeasure M* (*f* −‘ {*x*} ∩ *space M*))

syntax
-simple-integral :: *pttrn* ⇒ *ennreal* ⇒ *′a measure* ⇒ *ennreal* (∫^{S} *-*. *-* ∂*-* [*60*,*61*] *110*)

translations
∫^{S} *x*. *f* ∂*M* == *CONST simple-integral M* (%*x*. *f*)

lemma *simple-integral-cong*:
assumes ⋀*t*. *t* ∈ *space M* ⟹ *f t* = *g t*
shows *integral*^{S} *M f* = *integral*^{S} *M g*
proof −
have *f* ‘ *space M* = *g* ‘ *space M*
⋀*x*. *f* −‘ {*x*} ∩ *space M* = *g* −‘ {*x*} ∩ *space M*

```
    using assms by (auto intro!: image-eqI)
  thus ?thesis unfolding simple-integral-def by simp
qed

lemma simple-integral-const[simp]:
  (∫⁺x. c ∂M) = c * (emeasure M) (space M)
proof (cases space M = {})
  case True thus ?thesis unfolding simple-integral-def by simp
next
  case False hence (λx. c) ` space M = {c} by auto
  thus ?thesis unfolding simple-integral-def by simp
qed

lemma simple-function-partition:
  assumes f: simple-function M f and g: simple-function M g
  assumes sub: ⋀x y. x ∈ space M ⟹ y ∈ space M ⟹ g x = g y ⟹ f x = f y
  assumes v: ⋀x. x ∈ space M ⟹ f x = v (g x)
  shows integral^S M f = (∑ y∈g ` space M. v y * emeasure M {x∈space M. g x
= y})
    (is - = ?r)
proof −
  from f g have [simp]: finite (f`space M) finite (g`space M)
    by (auto simp: simple-function-def)
  from f g have [measurable]: f ∈ measurable M (count-space UNIV) g ∈ mea-
surable M (count-space UNIV)
    by (auto intro: measurable-simple-function)

  { fix y assume y ∈ space M
    then have f ` space M ∩ {i. ∃x∈space M. i = f x ∧ g y = g x} = {v (g y)}
      by (auto cong: sub simp: v[symmetric]) }
  note eq = this

  have integral^S M f =
    (∑ y∈f`space M. y * (∑ z∈g`space M.
      if ∃x∈space M. y = f x ∧ z = g x then emeasure M {x∈space M. g x = z}
else 0))
    unfolding simple-integral-def
  proof (safe intro!: setsum.cong ennreal-mult-left-cong)
    fix y assume y: y ∈ space M f y ≠ 0
    have [simp]: g ` space M ∩ {z. ∃x∈space M. f y = f x ∧ z = g x} =
        {z. ∃x∈space M. f y = f x ∧ z = g x}
      by auto
    have eq:(⋃ i∈{z. ∃x∈space M. f y = f x ∧ z = g x}. {x ∈ space M. g x = i})
=
        f −` {f y} ∩ space M
      by (auto simp: eq-commute cong: sub rev-conj-cong)
    have finite (g`space M) by simp
    then have finite {z. ∃x∈space M. f y = f x ∧ z = g x}
      by (rule rev-finite-subset) auto
```

then show *emeasure M* (*f* −‘ {*f y*} ∩ *space M*) =
(∑ *z*∈*g* ‘ *space M*. *if* ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x then emeasure M* {*x* ∈ *space M*. *g x* = *z*} *else 0*)
apply (*simp add*: *setsum.If-cases*)
apply (*subst setsum-emeasure*)
apply (*auto simp*: *disjoint-family-on-def eq*)
done
qed
also have ... = (∑ *y*∈*f*‘*space M*. (∑ *z*∈*g*‘*space M*.
if ∃ *x*∈*space M*. *y* = *f x* ∧ *z* = *g x then y* ∗ *emeasure M* {*x*∈*space M*. *g x* = *z*} *else 0*))
by (*auto intro*!: *setsum.cong simp*: *setsum-right-distrib*)
also have ... = *?r*
by (*subst setsum.commute*)
(*auto intro*!: *setsum.cong simp*: *setsum.If-cases scaleR-setsum-right*[*symmetric*] *eq*)
finally show *integral*^{S} *M f* = *?r* **.**
qed

lemma *simple-integral-add*[*simp*]:
assumes *f*: *simple-function M f* **and** ⋀*x*. *0* ≤ *f x* **and** *g*: *simple-function M g* **and** ⋀*x*. *0* ≤ *g x*
shows (∫^{S}*x*. *f x* + *g x* ∂*M*) = *integral*^{S} *M f* + *integral*^{S} *M g*
proof −
have (∫^{S}*x*. *f x* + *g x* ∂*M*) =
(∑ *y*∈(λ*x*. (*f x*, *g x*))‘*space M*. (*fst y* + *snd y*) ∗ *emeasure M* {*x*∈*space M*. (*f x*, *g x*) = *y*})
by (*intro simple-function-partition*) (*auto intro*: *f g*)
also have ... = (∑ *y*∈(λ*x*. (*f x*, *g x*))‘*space M*. *fst y* ∗ *emeasure M* {*x*∈*space M*. (*f x*, *g x*) = *y*}) +
(∑ *y*∈(λ*x*. (*f x*, *g x*))‘*space M*. *snd y* ∗ *emeasure M* {*x*∈*space M*. (*f x*, *g x*) = *y*})
using *assms*(*2*,*4*) **by** (*auto intro*!: *setsum.cong distrib-right simp*: *setsum.distrib*[*symmetric*])
also have (∑ *y*∈(λ*x*. (*f x*, *g x*))‘*space M*. *fst y* ∗ *emeasure M* {*x*∈*space M*. (*f x*, *g x*) = *y*}) = (∫^{S}*x*. *f x* ∂*M*)
by (*intro simple-function-partition*[*symmetric*]) (*auto intro*: *f g*)
also have (∑ *y*∈(λ*x*. (*f x*, *g x*))‘*space M*. *snd y* ∗ *emeasure M* {*x*∈*space M*. (*f x*, *g x*) = *y*}) = (∫^{S}*x*. *g x* ∂*M*)
by (*intro simple-function-partition*[*symmetric*]) (*auto intro*: *f g*)
finally show *?thesis* **.**
qed

lemma *simple-integral-setsum*[*simp*]:
assumes ⋀*i x*. *i* ∈ *P* ⟹ *0* ≤ *f i x*
assumes ⋀*i*. *i* ∈ *P* ⟹ *simple-function M* (*f i*)
shows (∫^{S}*x*. (∑ *i*∈*P*. *f i x*) ∂*M*) = (∑ *i*∈*P*. *integral*^{S} *M* (*f i*))
proof *cases*
assume *finite P*
from *this assms* **show** *?thesis*

by *induct* (*auto simp*: *simple-function-setsum simple-integral-add setsum-nonneg*)
qed *auto*

lemma *simple-integral-mult*[*simp*]:
assumes *f*: *simple-function M f*
shows $(\int^S x.\ c * f\ x\ \partial M) = c * integral^S\ M\ f$
proof −
have $(\int^S x.\ c * f\ x\ \partial M) = (\sum y \in f\ `\ space\ M.\ (c * y) * emeasure\ M\ \{x \in space\ M.\ f\ x = y\})$
using *f* **by** (*intro simple-function-partition*) *auto*
also have $\ldots = c * integral^S\ M\ f$
using *f* **unfolding** *simple-integral-def*
by (*subst setsum-right-distrib*) (*auto simp*: *mult.assoc Int-def conj-commute*)
finally show *?thesis* **.**
qed

lemma *simple-integral-mono-AE*:
assumes *f*[*measurable*]: *simple-function M f* **and** *g*[*measurable*]: *simple-function M g*
and *mono*: *AE x in M. f x* ≤ *g x*
shows $integral^S\ M\ f \leq integral^S\ M\ g$
proof −
let *?μ* = *λP. emeasure M* {*x*∈*space M. P x*}
have $integral^S\ M\ f = (\sum y \in (\lambda x.\ (f\ x,\ g\ x))\ `space\ M.\ fst\ y * ?\mu\ (\lambda x.\ (f\ x,\ g\ x) = y))$
using *f g* **by** (*intro simple-function-partition*) *auto*
also have $\ldots \leq (\sum y \in (\lambda x.\ (f\ x,\ g\ x))\ `space\ M.\ snd\ y * ?\mu\ (\lambda x.\ (f\ x,\ g\ x) = y))$
proof (*clarsimp intro*!: *setsum-mono*)
fix *x* **assume** *x* ∈ *space M*
let *?M* = *?μ* (*λy. f y* = *f x* ∧ *g y* = *g x*)
show *f x* ∗ *?M* ≤ *g x* ∗ *?M*
proof *cases*
assume *?M* ≠ *0*
then have *0* < *?M*
by (*simp add*: *less-le*)
also have ... ≤ *?μ* (*λy. f x* ≤ *g x*)
using *mono* **by** (*intro emeasure-mono-AE*) *auto*
finally have ¬ ¬ *f x* ≤ *g x*
by (*intro notI*) *auto*
then show *?thesis*
by (*intro mult-right-mono*) *auto*
qed *simp*
qed
also have ... = $integral^S\ M\ g$
using *f g* **by** (*intro simple-function-partition*[*symmetric*]) *auto*
finally show *?thesis* **.**
qed

lemma *simple-integral-mono*:
assumes *simple-function M f* **and** *simple-function M g*
and *mono*: $\bigwedge x.\ x \in space\ M \Longrightarrow f\ x \leq g\ x$
shows $integral^S\ M\ f \leq integral^S\ M\ g$
using *assms* **by** (*intro simple-integral-mono-AE*) *auto*

lemma *simple-integral-cong-AE*:
assumes *simple-function M f* **and** *simple-function M g*
and *AE x in M. f x = g x*
shows $integral^S\ M\ f = integral^S\ M\ g$
using *assms* **by** (*auto simp*: *eq-iff intro*!: *simple-integral-mono-AE*)

lemma *simple-integral-cong'*:
assumes *sf*: *simple-function M f simple-function M g*
and *mea*: (*emeasure M*) {*x*∈*space M. f x* ≠ *g x*} = *0*
shows $integral^S\ M\ f = integral^S\ M\ g$
proof (*intro simple-integral-cong-AE sf AE-I*)
show (*emeasure M*) {*x*∈*space M. f x* ≠ *g x*} = *0* **by** *fact*
show {*x* ∈ *space M. f x* ≠ *g x*} ∈ *sets M*
using *sf*[*THEN borel-measurable-simple-function*] **by** *auto*
qed *simp*

lemma *simple-integral-indicator*:
assumes *A*: *A* ∈ *sets M*
assumes *f*: *simple-function M f*
shows (∫^{S}*x. f x * indicator A x ∂M*) =
(∑ *x* ∈ *f* ‘ *space M. x * emeasure M* (*f* −‘ {*x*} ∩ *space M* ∩ *A*))
proof −
have *eq*: (λ*x*. (*f x*, *indicator A x*)) ‘ *space M* ∩ {*x*. *snd x* = *1*} = (λ*x*. (*f x*, *1*::*ennreal*))‘*A*
using *A*[*THEN sets.sets-into-space*] **by** (*auto simp*: *indicator-def image-iff split*: *if-split-asm*)
have *eq2*: ⋀*x*. *f x* ∉ *f* ‘ *A* ⟹ *f* −‘ {*f x*} ∩ *space M* ∩ *A* = {}
by (*auto simp*: *image-iff*)

have (∫^{S}*x. f x * indicator A x ∂M*) =
(∑ *y*∈(λ*x*. (*f x*, *indicator A x*))‘*space M*. (*fst y* * *snd y*) * *emeasure M* {*x*∈*space M*. (*f x*, *indicator A x*) = *y*})
using *assms* **by** (*intro simple-function-partition*) *auto*
also have ... = (∑ *y*∈(λ*x*. (*f x*, *indicator A x*::*ennreal*))‘*space M*.
*if snd y = 1 then fst y * emeasure M* (*f* −‘ {*fst y*} ∩ *space M* ∩ *A*) *else 0*)
by (*auto simp*: *indicator-def split*: *if-split-asm intro*!: *arg-cong2*[**where** *f*=*op* *] *arg-cong2*[**where** *f*=*emeasure*] *setsum.cong*)
also have ... = (∑ *y*∈(λ*x*. (*f x*, *1*::*ennreal*))‘*A*. *fst y * emeasure M* (*f* −‘ {*fst y*} ∩ *space M* ∩ *A*))
using *assms* **by** (*subst setsum.If-cases*) (*auto intro*!: *simple-functionD*(*1*) *simp*: *eq*)
also have ... = (∑ *y*∈*fst*‘(λ*x*. (*f x*, *1*::*ennreal*))‘*A*. *y * emeasure M* (*f* −‘ {*y*} ∩ *space M* ∩ *A*))

by (*subst setsum.reindex* [*of fst*]) (*auto simp*: *inj-on-def*)
also have ... = ($\sum x \in f$ ‘ *space M*. *x* ∗ *emeasure M* (*f* −‘ {*x*} ∩ *space M* ∩ *A*))
using *A*[*THEN sets.sets-into-space*]
by (*intro setsum.mono-neutral-cong-left simple-functionD f*) (*auto simp*: *image-comp comp-def eq2*)
finally show *?thesis* **.**
qed

lemma *simple-integral-indicator-only*[*simp*]:
assumes *A* ∈ *sets M*
shows $integral^S$ *M* (*indicator A*) = *emeasure M A*
using *simple-integral-indicator*[*OF assms*, *of* λ*x*. *1*] *sets.sets-into-space*[*OF assms*]
by (*simp-all add*: *image-constant-conv Int-absorb1 split*: *if-split-asm*)

lemma *simple-integral-null-set*:
assumes *simple-function M u* ⋀*x*. *0* ≤ *u x* **and** *N* ∈ *null-sets M*
shows ($\int^S x$. *u x* ∗ *indicator N x* ∂*M*) = *0*
proof −
have *AE x in M*. *indicator N x* = (*0* :: *ennreal*)
using ‹*N* ∈ *null-sets M*› **by** (*auto simp*: *indicator-def intro*!: *AE-I*[*of* - - *N*])
then have ($\int^S x$. *u x* ∗ *indicator N x* ∂*M*) = ($\int^S x$. *0* ∂*M*)
using *assms* **apply** (*intro simple-integral-cong-AE*) **by** *auto*
then show *?thesis* **by** *simp*
qed

lemma *simple-integral-cong-AE-mult-indicator*:
assumes *sf*: *simple-function M f* **and** *eq*: *AE x in M*. *x* ∈ *S* **and** *S* ∈ *sets M*
shows $integral^S$ *M f* = ($\int^S x$. *f x* ∗ *indicator S x* ∂*M*)
using *assms* **by** (*intro simple-integral-cong-AE*) *auto*

lemma *simple-integral-cmult-indicator*:
assumes *A*: *A* ∈ *sets M*
shows ($\int^S x$. *c* ∗ *indicator A x* ∂*M*) = *c* ∗ *emeasure M A*
using *simple-integral-mult*[*OF simple-function-indicator*[*OF A*]]
unfolding *simple-integral-indicator-only*[*OF A*] **by** *simp*

lemma *simple-integral-nonneg*:
assumes *f*: *simple-function M f* **and** *ae*: *AE x in M*. *0* ≤ *f x*
shows *0* ≤ $integral^S$ *M f*
proof −
have $integral^S$ *M* (λ*x*. *0*) ≤ $integral^S$ *M f*
using *simple-integral-mono-AE*[*OF* - *f ae*] **by** *auto*
then show *?thesis* **by** *simp*
qed

5.3 Integral on nonnegative functions

definition *nn-integral* :: *'a measure* ⇒ (*'a* ⇒ *ennreal*) ⇒ *ennreal* (*integral*^{N})
where
integral^N *M f* = (*SUP g* : {*g*. *simple-function M g* ∧ *g* ≤ *f*}. *integral*^{S} *M g*)

syntax
-nn-integral :: *pttrn* ⇒ *ennreal* ⇒ *'a measure* ⇒ *ennreal* (∫$^+$((*2* -./ -)/ ∂-) [*60*,*61*] *110*)

translations
∫$^+$*x*. *f* ∂*M* == *CONST nn-integral M* (λ*x*. *f*)

lemma *nn-integral-def-finite*:
integral^N *M f* = (*SUP g* : {*g*. *simple-function M g* ∧ *g* ≤ *f* ∧ (∀ *x*. *g x* < *top*)}. *integral*^{S} *M g*)
(**is** - = *SUPREMUM ?A ?f*)
unfolding *nn-integral-def*
proof (*safe intro*!: *antisym SUP-least*)
fix *g* **assume** *g*[*measurable*]: *simple-function M g g* ≤ *f*

show *integral*^{S} *M g* ≤ *SUPREMUM ?A ?f*
proof *cases*
assume *ae*: *AE x in M*. *g x* ≠ *top*
let *?G* = {*x* ∈ *space M*. *g x* ≠ *top*}
have *integral*^{S} *M g* = *integral*^{S} *M* (λ*x*. *g x* ∗ *indicator ?G x*)
proof (*rule simple-integral-cong-AE*)
show *AE x in M*. *g x* = *g x* ∗ *indicator ?G x*
using *ae AE-space* **by** *eventually-elim auto*
qed (*insert g*, *auto*)
also have ... ≤ *SUPREMUM ?A ?f*
using *g* **by** (*intro SUP-upper*) (*auto simp*: *le-fun-def less-top split*: *split-indicator*)
finally show *?thesis* **.**
next
assume *nAE*: ¬ (*AE x in M*. *g x* ≠ *top*)
then have *emeasure M* {*x*∈*space M*. *g x* = *top*} ≠ *0* (**is** *emeasure M ?G* ≠ *0*)
by (*subst* (*asm*) *AE-iff-measurable*[*OF* - *refl*]) *auto*
then have *top* = (*SUP n*. (∫^{S}*x*. *of-nat n* ∗ *indicator ?G x* ∂*M*))
by (*simp add*: *ennreal-SUP-of-nat-eq-top ennreal-top-eq-mult-iff SUP-mult-right-ennreal*[*symmetric*])
also have ... ≤ *SUPREMUM ?A ?f*
using *g*
by (*safe intro*!: *SUP-least SUP-upper*)
(*auto simp*: *le-fun-def of-nat-less-top top-unique*[*symmetric*] *split*: *split-indicator*
intro: *order-trans*[*of* - *g x f x* **for** *x*, *OF order-trans*[*of* - *top*]])
finally show *?thesis*
by (*simp add*: *top-unique del*: *SUP-eq-top-iff Sup-eq-top-iff*)
qed
qed (*auto intro*: *SUP-upper*)

lemma *nn-integral-mono-AE*:
assumes *ae*: $AE\ x\ in\ M.\ u\ x \leq v\ x$ **shows** $integral^N\ M\ u \leq integral^N\ M\ v$
unfolding *nn-integral-def*
proof (*safe intro*!: *SUP-mono*)
fix n **assume** n: *simple-function* $M\ n\ n \leq u$
from $ae[THEN\ AE\text{-}E]$ **guess** N **. note** $N = this$
then have *ae-N*: $AE\ x\ in\ M.\ x \notin N$ **by** (*auto intro*: *AE-not-in*)
let $?n = \lambda x.\ n\ x * indicator\ (space\ M - N)\ x$
have $AE\ x\ in\ M.\ n\ x \leq ?n\ x$ *simple-function* $M\ ?n$
using $n\ N$ *ae-N* **by** *auto*
moreover
{ fix x **have** $?n\ x \leq v\ x$
proof *cases*
assume x: $x \in space\ M - N$
with N **have** $u\ x \leq v\ x$ **by** *auto*
with $n(2)[THEN\ le\text{-}funD,\ of\ x]\ x$ **show** *?thesis*
by (*auto simp*: *max-def split*: *if-split-asm*)
qed *simp* **}**
then have $?n \leq v$ **by** (*auto simp*: *le-funI*)
moreover have $integral^S\ M\ n \leq integral^S\ M\ ?n$
using *ae-N N n* **by** (*auto intro*!: *simple-integral-mono-AE*)
ultimately show $\exists m \in \{g.\ simple\text{-}function\ M\ g \wedge g \leq v\}.\ integral^S\ M\ n \leq integral^S\ M\ m$
by *force*
qed

lemma *nn-integral-mono*:
$(\bigwedge x.\ x \in space\ M \Longrightarrow u\ x \leq v\ x) \Longrightarrow integral^N\ M\ u \leq integral^N\ M\ v$
by (*auto intro*: *nn-integral-mono-AE*)

lemma *mono-nn-integral*: $mono\ F \Longrightarrow mono\ (\lambda x.\ integral^N\ M\ (F\ x))$
by (*auto simp add*: *mono-def le-fun-def intro*!: *nn-integral-mono*)

lemma *nn-integral-cong-AE*:
$AE\ x\ in\ M.\ u\ x = v\ x \Longrightarrow integral^N\ M\ u = integral^N\ M\ v$
by (*auto simp*: *eq-iff intro*!: *nn-integral-mono-AE*)

lemma *nn-integral-cong*:
$(\bigwedge x.\ x \in space\ M \Longrightarrow u\ x = v\ x) \Longrightarrow integral^N\ M\ u = integral^N\ M\ v$
by (*auto intro*: *nn-integral-cong-AE*)

lemma *nn-integral-cong-simp*:
$(\bigwedge x.\ x \in space\ M =simp=> u\ x = v\ x) \Longrightarrow integral^N\ M\ u = integral^N\ M\ v$
by (*auto intro*: *nn-integral-cong simp*: *simp-implies-def*)

lemma *nn-integral-cong-strong*:
$M = N \Longrightarrow (\bigwedge x.\ x \in space\ M \Longrightarrow u\ x = v\ x) \Longrightarrow integral^N\ M\ u = integral^N\ N\ v$
by (*auto intro*: *nn-integral-cong*)

lemma *incseq-nn-integral*:
assumes *incseq f* **shows** *incseq* ($\lambda i.$ $integral^N$ M $(f\ i)$)
proof −
have $\bigwedge i\ x.\ f\ i\ x \leq f\ (Suc\ i)\ x$
using *assms* **by** (*auto dest*!: *incseq-SucD simp*: *le-fun-def*)
then show *?thesis*
by (*auto intro*!: *incseq-SucI nn-integral-mono*)
qed

lemma *nn-integral-eq-simple-integral*:
assumes *f*: *simple-function M f* **shows** $integral^N\ M\ f = integral^S\ M\ f$
proof −
let *?f* = $\lambda x.\ f\ x * indicator\ (space\ M)\ x$
have *f'*: *simple-function M ?f* **using** *f* **by** *auto*
have $integral^N\ M\ ?f \leq integral^S\ M\ ?f$ **using** *f'*
by (*force intro*!: *SUP-least simple-integral-mono simp*: *le-fun-def nn-integral-def*)
moreover have $integral^S\ M\ ?f \leq integral^N\ M\ ?f$
unfolding *nn-integral-def*
using *f'* **by** (*auto intro*!: *SUP-upper*)
ultimately show *?thesis*
by (*simp cong*: *nn-integral-cong simple-integral-cong*)
qed

Beppo-Levi monotone convergence theorem

lemma *nn-integral-monotone-convergence-SUP*:
assumes *f*: *incseq f* **and** [*measurable*]: $\bigwedge i.\ f\ i \in borel\text{-}measurable\ M$
shows $(\int^+ x.\ (SUP\ i.\ f\ i\ x)\ \partial M) = (SUP\ i.\ integral^N\ M\ (f\ i))$
proof (*rule antisym*)
show $(\int^+ x.\ (SUP\ i.\ f\ i\ x)\ \partial M) \leq (SUP\ i.\ (\int^+ x.\ f\ i\ x\ \partial M))$
unfolding *nn-integral-def-finite*[*of* - $\lambda x.\ SUP\ i.\ f\ i\ x$]
proof (*safe intro*!: *SUP-least*)
fix *u* **assume** *sf-u*[*simp*]: *simple-function M u* **and**
u: $u \leq (\lambda x.\ SUP\ i.\ f\ i\ x)$ **and** *u-range*: $\forall x.\ u\ x < top$
note *sf-u*[*THEN borel-measurable-simple-function*, *measurable*]
show $integral^S\ M\ u \leq (SUP\ j.\ \int^+ x.\ f\ j\ x\ \partial M)$
proof (*rule ennreal-approx-unit*)
fix *a* :: *ennreal* **assume** *a* < *1*
let *?au* = $\lambda x.\ a * u\ x$

let *?B* = $\lambda c\ i.\ \{x \in space\ M.\ ?au\ x = c \wedge c \leq f\ i\ x\}$
have $integral^S\ M\ ?au = (\sum c \in ?au`space\ M.\ c * (SUP\ i.\ emeasure\ M\ (?B\ c\ i)))$
unfolding *simple-integral-def*
proof (*intro setsum.cong ennreal-mult-left-cong refl*)
fix *c* **assume** $c \in ?au\ `\ space\ M\ c \neq 0$
{ fix *x'* **assume** *x'*: $x' \in space\ M\ ?au\ x' = c$
with ‹$c \neq 0$› *u-range* **have** $?au\ x' < 1 * u\ x'$
by (*intro ennreal-mult-strict-right-mono* ‹$a < 1$›) (*auto simp*: *less-le*)

```
        also have ... ≤ (SUP i. f i x')
          using u by (auto simp: le-fun-def)
        finally have ∃i. ?au x' ≤ f i x'
          by (auto simp: less-SUP-iff intro: less-imp-le) }
      then have *: ?au -` {c} ∩ space M = (⋃i. ?B c i)
        by auto
      show emeasure M (?au -` {c} ∩ space M) = (SUP i. emeasure M (?B c
i))
        unfolding * using f
        by (intro SUP-emeasure-incseq[symmetric])
           (auto simp: incseq-def le-fun-def intro: order-trans)
    qed
    also have ... = (SUP i. ∑c∈?au`space M. c * emeasure M (?B c i))
      unfolding SUP-mult-left-ennreal using f
      by (intro ennreal-SUP-setsum[symmetric])
         (auto intro!: mult-mono emeasure-mono simp: incseq-def le-fun-def intro:
order-trans)
    also have ... ≤ (SUP i. integral^N M (f i))
    proof (intro SUP-subset-mono order-refl)
      fix i
      have (∑c∈?au`space M. c * emeasure M (?B c i)) =
        (∫^S x. (a * u x) * indicator {x∈space M. a * u x ≤ f i x} x ∂M)
        by (subst simple-integral-indicator)
           (auto intro!: setsum.cong ennreal-mult-left-cong arg-cong2[where
f=emeasure])
      also have ... = (∫^+ x. (a * u x) * indicator {x∈space M. a * u x ≤ f i x}
x ∂M)
        by (rule nn-integral-eq-simple-integral[symmetric]) simp
      also have ... ≤ (∫^+ x. f i x ∂M)
        by (intro nn-integral-mono) (auto split: split-indicator)
      finally show (∑c∈?au`space M. c * emeasure M (?B c i)) ≤ (∫^+ x. f i x
∂M) .
    qed
    finally show a * integral^S M u ≤ (SUP i. integral^N M (f i))
      by simp
  qed
 qed
qed (auto intro!: SUP-least SUP-upper nn-integral-mono)

lemma sup-continuous-nn-integral[order-continuous-intros]:
  assumes f: ⋀y. sup-continuous (f y)
  assumes [measurable]: ⋀x. (λy. f y x) ∈ borel-measurable M
  shows sup-continuous (λx. (∫^+ y. f y x ∂M))
  unfolding sup-continuous-def
proof safe
  fix C :: nat ⇒ 'b assume C: incseq C
   with sup-continuous-mono[OF f] show (∫^+ y. f y (SUPREMUM UNIV C)
∂M) = (SUP i. ∫^+ y. f y (C i) ∂M)
    unfolding sup-continuousD[OF f C]
```

by (*subst nn-integral-monotone-convergence-SUP*) (*auto simp*: *mono-def le-fun-def*)
qed

lemma *nn-integral-monotone-convergence-SUP-AE*:
assumes *f*: $\bigwedge i.$ *AE x in M. f i x* $\leq$ *f* (*Suc i*) *x* $\bigwedge i.$ *f i* $\in$ *borel-measurable M*
shows ($\int^+$ *x.* (*SUP i. f i x*) *∂M*) = (*SUP i. integral*^{N} *M* (*f i*))
proof −
from *f* **have** *AE x in M.* $\forall i.$ *f i x* $\leq$ *f* (*Suc i*) *x*
by (*simp add*: *AE-all-countable*)
from *this*[*THEN AE-E*] **guess** *N* **. note** *N* = *this*
let *?f* = λi *x. if x* $\in$ *space M* − *N then f i x else 0*
have *f-eq*: *AE x in M.* $\forall i.$ *?f i x* = *f i x* **using** *N* **by** (*auto intro*!: *AE-I*[*of - - N*])
then have ($\int^+$ *x.* (*SUP i. f i x*) *∂M*) = ($\int^+$ *x.* (*SUP i. ?f i x*) *∂M*)
by (*auto intro*!: *nn-integral-cong-AE*)
also have ... = (*SUP i.* ($\int^+$ *x. ?f i x ∂M*))
proof (*rule nn-integral-monotone-convergence-SUP*)
show *incseq ?f* **using** *N*(*1*) **by** (*force intro*!: *incseq-SucI le-funI*)
{ fix *i* **show** ($\lambda x.$ *if x* $\in$ *space M* − *N then f i x else 0*) $\in$ *borel-measurable M*
using *f N*(*3*) **by** (*intro measurable-If-set*) *auto* **}**
qed
also have ... = (*SUP i.* ($\int^+$ *x. f i x ∂M*))
using *f-eq* **by** (*force intro*!: *arg-cong*[**where** *f*=*SUPREMUM UNIV*] *nn-integral-cong-AE ext*)
finally show *?thesis* **.**
qed

lemma *nn-integral-monotone-convergence-simple*:
incseq f $\Longrightarrow$ ($\bigwedge i.$ *simple-function M* (*f i*)) $\Longrightarrow$ (*SUP i.* $\int^S$ *x. f i x ∂M*) = ($\int^+ x.$ (*SUP i. f i x*) *∂M*)
using *assms nn-integral-monotone-convergence-SUP*[*of f M*]
by (*simp add*: *nn-integral-eq-simple-integral*[*symmetric*] *borel-measurable-simple-function*)

lemma *SUP-simple-integral-sequences*:
assumes *f*: *incseq f* $\bigwedge i.$ *simple-function M* (*f i*)
and *g*: *incseq g* $\bigwedge i.$ *simple-function M* (*g i*)
and *eq*: *AE x in M.* (*SUP i. f i x*) = (*SUP i. g i x*)
shows (*SUP i. integral*^{S} *M* (*f i*)) = (*SUP i. integral*^{S} *M* (*g i*))
(**is** *SUPREMUM - ?F* = *SUPREMUM - ?G*)
proof −
have (*SUP i. integral*^{S} *M* (*f i*)) = ($\int^+ x.$ (*SUP i. f i x*) *∂M*)
using *f* **by** (*rule nn-integral-monotone-convergence-simple*)
also have ... = ($\int^+ x.$ (*SUP i. g i x*) *∂M*)
unfolding *eq*[*THEN nn-integral-cong-AE*] **..**
also have ... = (*SUP i. ?G i*)
using *g* **by** (*rule nn-integral-monotone-convergence-simple*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *nn-integral-const*[*simp*]: $(\int^+ x.\ c\ \partial M) = c * \mathit{emeasure}\ M\ (\mathit{space}\ M)$
by (*subst nn-integral-eq-simple-integral*) *auto*

lemma *nn-integral-linear*:
assumes *f*: $f \in \mathit{borel\text{-}measurable}\ M$ **and** *g*: $g \in \mathit{borel\text{-}measurable}\ M$
shows $(\int^+ x.\ a * f\ x + g\ x\ \partial M) = a * \mathit{integral}^N\ M\ f + \mathit{integral}^N\ M\ g$
(**is** $\mathit{integral}^N\ M\ ?L = -$)
proof −
from *borel-measurable-implies-simple-function-sequence'*[*OF f*(*1*)] **guess** *u* **.**
note *u* = *nn-integral-monotone-convergence-simple*[*OF this*(*2*,*1*)] *this*
from *borel-measurable-implies-simple-function-sequence'*[*OF g*(*1*)] **guess** *v* **.**
note *v* = *nn-integral-monotone-convergence-simple*[*OF this*(*2*,*1*)] *this*
let $?L' = \lambda i\ x.\ a * u\ i\ x + v\ i\ x$

have $?L \in \mathit{borel\text{-}measurable}\ M$ **using** *assms* **by** *auto*
from *borel-measurable-implies-simple-function-sequence'*[*OF this*] **guess** *l* **.**
note *l* = *nn-integral-monotone-convergence-simple*[*OF this*(*2*,*1*)] *this*

have *inc*: *incseq* $(\lambda i.\ a * \mathit{integral}^S\ M\ (u\ i))$ *incseq* $(\lambda i.\ \mathit{integral}^S\ M\ (v\ i))$
using *u v* **by** (*auto simp*: *incseq-Suc-iff le-fun-def intro*!: *add-mono mult-left-mono simple-integral-mono*)

have *l'*: $(\mathit{SUP}\ i.\ \mathit{integral}^S\ M\ (l\ i)) = (\mathit{SUP}\ i.\ \mathit{integral}^S\ M\ (?L'\ i))$
proof (*rule SUP-simple-integral-sequences*[*OF l*(*3*,*2*)])
show *incseq* $?L'$ $\bigwedge i.$ *simple-function M* $(?L'\ i)$
using *u v* **unfolding** *incseq-Suc-iff le-fun-def*
by (*auto intro*!: *add-mono mult-left-mono*)
{ fix *x*
have $(\mathit{SUP}\ i.\ a * u\ i\ x + v\ i\ x) = a * (\mathit{SUP}\ i.\ u\ i\ x) + (\mathit{SUP}\ i.\ v\ i\ x)$
using *u*(*3*) *v*(*3*) *u*(*4*)[*of - x*] *v*(*4*)[*of - x*] **unfolding** *SUP-mult-left-ennreal*
by (*auto intro*!: *ennreal-SUP-add simp*: *incseq-Suc-iff le-fun-def add-mono mult-left-mono*) **}**
then show *AE x in M*. $(\mathit{SUP}\ i.\ l\ i\ x) = (\mathit{SUP}\ i.\ ?L'\ i\ x)$
unfolding *l*(*5*) **using** *u*(*5*) *v*(*5*) **by** (*intro AE-I2*) *auto*
qed
also have ... $= (\mathit{SUP}\ i.\ a * \mathit{integral}^S\ M\ (u\ i) + \mathit{integral}^S\ M\ (v\ i))$
using *u*(*2*) *v*(*2*) **by** *auto*
finally show *?thesis*
unfolding *l*(*5*)[*symmetric*] *l*(*1*)[*symmetric*]
by (*simp add*: *ennreal-SUP-add*[*OF inc*] *v u SUP-mult-left-ennreal*[*symmetric*])
qed

lemma *nn-integral-cmult*: $f \in \mathit{borel\text{-}measurable}\ M \Longrightarrow (\int^+ x.\ c * f\ x\ \partial M) = c * \mathit{integral}^N\ M\ f$
using *nn-integral-linear*[*of f M* $\lambda x.\ 0$ *c*] **by** *simp*

lemma *nn-integral-multc*: $f \in \mathit{borel\text{-}measurable}\ M \Longrightarrow (\int^+ x.\ f\ x * c\ \partial M) = \mathit{integral}^N\ M\ f * c$
unfolding *mult.commute*[*of - c*] *nn-integral-cmult*[*OF assms*] **by** *simp*

lemma *nn-integral-divide*: $f \in$ *borel-measurable* $M \Longrightarrow (\int^+ x.\ f\ x\ /\ c\ \partial M) = (\int^+ x.\ f\ x\ \partial M)\ /\ c$
unfolding *divide-ennreal-def* **by** (*rule nn-integral-multc*)

lemma *nn-integral-indicator*[*simp*]: $A \in$ *sets* $M \Longrightarrow (\int^+ x.$ *indicator* $A\ x \partial M) =$ (*emeasure* M) A
by (*subst nn-integral-eq-simple-integral*) (*auto simp*: *simple-integral-indicator*)

lemma *nn-integral-cmult-indicator*: $A \in$ *sets* $M \Longrightarrow (\int^+ x.\ c * $ *indicator* $A\ x\ \partial M) = c *$ *emeasure* $M\ A$
by (*subst nn-integral-eq-simple-integral*)
(*auto simp*: *simple-function-indicator simple-integral-indicator*)

lemma *nn-integral-indicator'*:
assumes [*measurable*]: $A \cap$ *space* $M \in$ *sets* M
shows $(\int^+ x.$ *indicator* $A\ x\ \partial M) =$ *emeasure* $M\ (A \cap$ *space* $M)$
proof −
have $(\int^+ x.$ *indicator* $A\ x\ \partial M) = (\int^+ x.$ *indicator* $(A \cap$ *space* $M)\ x\ \partial M)$
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also have $\ldots =$ *emeasure* $M\ (A \cap$ *space* $M)$
by *simp*
finally show *?thesis* **.**
qed

lemma *nn-integral-indicator-singleton*[*simp*]:
assumes [*measurable*]: $\{y\} \in$ *sets* M **shows** $(\int^+ x.\ f\ x * $ *indicator* $\{y\}\ x\ \partial M) = f\ y *$ *emeasure* $M\ \{y\}$
proof −
have $(\int^+ x.\ f\ x * $ *indicator* $\{y\}\ x\ \partial M) = (\int^+ x.\ f\ y * $ *indicator* $\{y\}\ x\ \partial M)$
by (*auto intro*!: *nn-integral-cong split*: *split-indicator*)
then show *?thesis*
by (*simp add*: *nn-integral-cmult*)
qed

lemma *nn-integral-set-ennreal*:
$(\int^+ x.$ *ennreal* $(f\ x) *$ *indicator* $A\ x\ \partial M) = (\int^+ x.$ *ennreal* $(f\ x * $ *indicator* $A\ x)$ $\partial M)$
by (*rule nn-integral-cong*) (*simp split*: *split-indicator*)

lemma *nn-integral-indicator-singleton'*[*simp*]:
assumes [*measurable*]: $\{y\} \in$ *sets* M
shows $(\int^+ x.$ *ennreal* $(f\ x * $ *indicator* $\{y\}\ x)\ \partial M) = f\ y *$ *emeasure* $M\ \{y\}$
by (*subst nn-integral-set-ennreal*[*symmetric*]) (*simp add*: *nn-integral-indicator-singleton*)

lemma *nn-integral-add*:
$f \in$ *borel-measurable* $M \Longrightarrow g \in$ *borel-measurable* $M \Longrightarrow (\int^+ x.\ f\ x + g\ x\ \partial M)$ $=$ *integral*^{N} $M\ f +$ *integral*^{N} $M\ g$
using *nn-integral-linear*[*of f M g 1*] **by** *simp*

lemma *nn-integral-setsum*:
(⋀*i*. *i* ∈ *P* ⟹ *f i* ∈ *borel-measurable M*) ⟹ (∫⁺ *x*. (∑ *i*∈*P*. *f i x*) ∂*M*) = (∑ *i*∈*P*. *integral*^{N} *M* (*f i*))
by (*induction P rule*: *infinite-finite-induct*) (*auto simp*: *nn-integral-add*)

lemma *nn-integral-suminf*:
assumes *f*: ⋀*i*. *f i* ∈ *borel-measurable M*
shows (∫⁺ *x*. (∑ *i*. *f i x*) ∂*M*) = (∑ *i*. *integral*^{N} *M* (*f i*))
proof −
have *all-pos*: *AE x in M*. ∀ *i*. 0 ≤ *f i x*
using *assms* **by** (*auto simp*: *AE-all-countable*)
have (∑ *i*. *integral*^{N} *M* (*f i*)) = (*SUP n*. ∑ *i*<*n*. *integral*^{N} *M* (*f i*))
by (*rule suminf-eq-SUP*)
also have ... = (*SUP n*. ∫⁺*x*. (∑ *i*<*n*. *f i x*) ∂*M*)
unfolding *nn-integral-setsum*[*OF f*] **..**
also have ... = ∫⁺*x*. (*SUP n*. ∑ *i*<*n*. *f i x*) ∂*M* **using** *f all-pos*
by (*intro nn-integral-monotone-convergence-SUP-AE*[*symmetric*])
(*elim AE-mp*, *auto simp*: *setsum-nonneg simp del*: *setsum-lessThan-Suc intro*!: *AE-I2 setsum-mono3*)
also have ... = ∫⁺*x*. (∑ *i*. *f i x*) ∂*M* **using** *all-pos*
by (*intro nn-integral-cong-AE*) (*auto simp*: *suminf-eq-SUP*)
finally show *?thesis* **by** *simp*
qed

lemma *nn-integral-bound-simple-function*:
assumes *bnd*: ⋀*x*. *x* ∈ *space M* ⟹ *f x* < ∞
assumes *f*[*measurable*]: *simple-function M f*
assumes *supp*: *emeasure M* {*x*∈*space M*. *f x* ≠ 0} < ∞
shows *nn-integral M f* < ∞
proof *cases*
assume *space M* = {}
then have *nn-integral M f* = (∫⁺*x*. 0 ∂*M*)
by (*intro nn-integral-cong*) *auto*
then show *?thesis* **by** *simp*
next
assume *space M* ≠ {}
with *simple-functionD*(*1*)[*OF f*] *bnd* **have** *bnd*: 0 ≤ *Max* (*f*‘*space M*) ∧ *Max* (*f*‘*space M*) < ∞
by (*subst Max-less-iff*) (*auto simp*: *Max-ge-iff*)

have *nn-integral M f* ≤ (∫⁺*x*. *Max* (*f*‘*space M*) ∗ *indicator* {*x*∈*space M*. *f x* ≠ 0} *x* ∂*M*)
proof (*rule nn-integral-mono*)
fix *x* **assume** *x* ∈ *space M*
with *f* **show** *f x* ≤ *Max* (*f* ‘ *space M*) ∗ *indicator* {*x* ∈ *space M*. *f x* ≠ 0} *x*
by (*auto split*: *split-indicator intro*!: *Max-ge simple-functionD*)
qed
also have ... < ∞

using *bnd supp* **by** (*subst nn-integral-cmult*) (*auto simp*: *ennreal-mult-less-top*)
finally show *?thesis* **.**
qed

lemma *nn-integral-Markov-inequality*:
assumes *u*: $u \in$ *borel-measurable* M **and** $A \in$ *sets* M
shows (*emeasure* M) ($\{x \in$ *space* M. $1 \leq c * u\ x\} \cap A$) $\leq c * (\int^+ x.\ u\ x *$ *indicator* $A\ x\ \partial M$)
(**is** (*emeasure* M) *?A* $\leq$ - $*$ *?PI*)
proof −
have *?A* $\in$ *sets* M
using ‹$A \in$ *sets* M› *u* **by** *auto*
hence (*emeasure* M) *?A* $= (\int^+ x.$ *indicator ?A* $x\ \partial M$)
using *nn-integral-indicator* **by** *simp*
also have ... $\leq (\int^+ x.\ c * (u\ x *$ *indicator* $A\ x)\ \partial M$)
using *u* **by** (*auto intro*!: *nn-integral-mono-AE simp*: *indicator-def*)
also have ... $= c * (\int^+ x.\ u\ x *$ *indicator* $A\ x\ \partial M$)
using *assms* **by** (*auto intro*!: *nn-integral-cmult*)
finally show *?thesis* **.**
qed

lemma *nn-integral-noteq-infinite*:
assumes *g*: $g \in$ *borel-measurable* M **and** $integral^N\ M\ g \neq \infty$
shows *AE* x *in* M. $g\ x \neq \infty$
proof (*rule ccontr*)
assume *c*: $\neg$ (*AE* x *in* M. $g\ x \neq \infty$)
have (*emeasure* M) $\{x \in$ *space* M. $g\ x = \infty\} \neq 0$
using *c g* **by** (*auto simp add*: *AE-iff-null*)
then have $0 <$ (*emeasure* M) $\{x \in$ *space* M. $g\ x = \infty\}$
by (*auto simp*: *zero-less-iff-neq-zero*)
then have $\infty = \infty *$ (*emeasure* M) $\{x \in$ *space* M. $g\ x = \infty\}$
by (*auto simp*: *ennreal-top-eq-mult-iff*)
also have ... $\leq (\int^+ x.\ \infty *$ *indicator* $\{x \in$ *space* M. $g\ x = \infty\}\ x\ \partial M$)
using *g* **by** (*subst nn-integral-cmult-indicator*) *auto*
also have ... $\leq integral^N\ M\ g$
using *assms* **by** (*auto intro*!: *nn-integral-mono-AE simp*: *indicator-def*)
finally show *False*
using ‹$integral^N\ M\ g \neq \infty$› **by** (*auto simp*: *top-unique*)
qed

lemma *nn-integral-PInf*:
assumes *f*: $f \in$ *borel-measurable* M **and** *not-Inf*: $integral^N\ M\ f \neq \infty$
shows *emeasure* M ($f -$‘ $\{\infty\} \cap$ *space* M) $= 0$
proof −
have $\infty *$ *emeasure* M ($f -$‘ $\{\infty\} \cap$ *space* M) $= (\int^+ x.\ \infty *$ *indicator* ($f -$‘ $\{\infty\} \cap$ *space* M) $x\ \partial M$)
using *f* **by** (*subst nn-integral-cmult-indicator*) (*auto simp*: *measurable-sets*)
also have ... $\leq integral^N\ M\ f$
by (*auto intro*!: *nn-integral-mono simp*: *indicator-def*)

finally have $\infty * (emeasure\ M)\ (f\ -`\ \{\infty\} \cap space\ M) \leq integral^N\ M\ f$
by *simp*
then show *?thesis*
using *assms* **by** (*auto simp*: *ennreal-top-mult top-unique split*: *if-split-asm*)
qed

lemma *simple-integral-PInf*:
$simple\text{-}function\ M\ f \Longrightarrow integral^S\ M\ f \neq \infty \Longrightarrow emeasure\ M\ (f\ -`\ \{\infty\} \cap space\ M) = 0$
by (*rule nn-integral-PInf*) (*auto simp*: *nn-integral-eq-simple-integral borel-measurable-simple-function*)

lemma *nn-integral-PInf-AE*:
assumes $f \in borel\text{-}measurable\ M\ integral^N\ M\ f \neq \infty$ **shows** $AE\ x\ in\ M.\ f\ x \neq \infty$
proof (*rule AE-I*)
show $(emeasure\ M)\ (f\ -`\ \{\infty\} \cap space\ M) = 0$
by (*rule nn-integral-PInf*[*OF assms*])
show $f\ -`\ \{\infty\} \cap space\ M \in sets\ M$
using *assms* **by** (*auto intro*: *borel-measurable-vimage*)
qed *auto*

lemma *nn-integral-diff*:
assumes *f*: $f \in borel\text{-}measurable\ M$
and *g*: $g \in borel\text{-}measurable\ M$
and *fin*: $integral^N\ M\ g \neq \infty$
and *mono*: $AE\ x\ in\ M.\ g\ x \leq f\ x$
shows $(\int^+ x.\ f\ x - g\ x\ \partial M) = integral^N\ M\ f - integral^N\ M\ g$
proof −
have *diff*: $(\lambda x.\ f\ x - g\ x) \in borel\text{-}measurable\ M$
using *assms* **by** *auto*
have $AE\ x\ in\ M.\ f\ x = f\ x - g\ x + g\ x$
using *diff-add-cancel-ennreal mono nn-integral-noteq-infinite*[*OF g fin*] *assms*
by *auto*
then have $**$: $integral^N\ M\ f = (\int^+ x.\ f\ x - g\ x\ \partial M) + integral^N\ M\ g$
unfolding *nn-integral-add*[*OF diff g*, *symmetric*]
by (*rule nn-integral-cong-AE*)
show *?thesis* **unfolding** $**$
using *fin*
by (*cases rule*: *ennreal2-cases*[*of* $\int^+ x.\ f\ x - g\ x\ \partial M\ integral^N\ M\ g$]) *auto*
qed

lemma *nn-integral-mult-bounded-inf*:
assumes *f*: $f \in borel\text{-}measurable\ M\ (\int^+ x.\ f\ x\ \partial M) < \infty$ **and** *c*: $c \neq \infty$ **and** *ae*: $AE\ x\ in\ M.\ g\ x \leq c * f\ x$
shows $(\int^+ x.\ g\ x\ \partial M) < \infty$
proof −
have $(\int^+ x.\ g\ x\ \partial M) \leq (\int^+ x.\ c * f\ x\ \partial M)$
by (*intro nn-integral-mono-AE ae*)
also have $(\int^+ x.\ c * f\ x\ \partial M) < \infty$

using *c f* **by** (*subst nn-integral-cmult*) (*auto simp*: *ennreal-mult-less-top top-unique not-less*)
finally show *?thesis* **.**
qed

Fatou's lemma: convergence theorem on limes inferior

lemma *nn-integral-monotone-convergence-INF-AE'*:
assumes *f*: $\bigwedge i.$ *AE x in M. f* (*Suc i*) *x* $\leq$ *f i x* **and** [*measurable*]: $\bigwedge i.$ *f i* $\in$ *borel-measurable M*
and $*$: $(\int^+ x.\ f\ 0\ x\ \partial M) < \infty$
shows $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = (INF\ i.\ integral^N\ M\ (f\ i))$
proof (*rule ennreal-minus-cancel*)
have $integral^N\ M\ (f\ 0) - (\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = (\int^+ x.\ f\ 0\ x - (INF\ i.\ f\ i\ x)\ \partial M)$
proof (*rule nn-integral-diff*[*symmetric*])
have $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) \leq (\int^+ x.\ f\ 0\ x\ \partial M)$
by (*intro nn-integral-mono INF-lower*) *simp*
with $*$ **show** $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) \neq \infty$
by *simp*
qed (*auto intro*: *INF-lower*)
also have $\ldots = (\int^+ x.\ (SUP\ i.\ f\ 0\ x - f\ i\ x)\ \partial M)$
by (*simp add*: *ennreal-INF-const-minus*)
also have $\ldots = (SUP\ i.\ (\int^+ x.\ f\ 0\ x - f\ i\ x\ \partial M))$
proof (*intro nn-integral-monotone-convergence-SUP-AE*)
show *AE x in M. f 0 x* $-$ *f i x* $\leq$ *f 0 x* $-$ *f* (*Suc i*) *x* **for** *i*
using *f*[*of i*] **by** *eventually-elim* (*auto simp*: *ennreal-mono-minus*)
qed *simp*
also have $\ldots = (SUP\ i.\ nn\text{-}integral\ M\ (f\ 0) - (\int^+ x.\ f\ i\ x\ \partial M))$
proof (*subst nn-integral-diff*[*symmetric*])
fix *i*
have *dec*: *AE x in M.* $\forall i.$ *f* (*Suc i*) *x* $\leq$ *f i x*
unfolding *AE-all-countable* **using** *f* **by** *auto*
then show *AE x in M. f i x* $\leq$ *f 0 x*
using *dec* **by** *eventually-elim* (*auto intro*: *lift-Suc-antimono-le*[*of* $\lambda i.$ *f i x 0 i* **for** *x*])
then have $(\int^+ x.\ f\ i\ x\ \partial M) \leq (\int^+ x.\ f\ 0\ x\ \partial M)$
by (*rule nn-integral-mono-AE*)
with $*$ **show** $(\int^+ x.\ f\ i\ x\ \partial M) \neq \infty$
by *simp*
qed (*insert f*, *auto simp*: *decseq-def le-fun-def*)
finally show $integral^N\ M\ (f\ 0) - (\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = integral^N\ M\ (f\ 0) - (INF\ i.\ \int^+ x.\ f\ i\ x\ \partial M)$
by (*simp add*: *ennreal-INF-const-minus*)
qed (*insert* $*$, *auto intro*!: *nn-integral-mono intro*: *INF-lower*)

lemma *nn-integral-monotone-convergence-INF-AE*:
fixes *f* :: *nat* $\Rightarrow$ *'a* $\Rightarrow$ *ennreal*
assumes *f*: $\bigwedge i.$ *AE x in M. f* (*Suc i*) *x* $\leq$ *f i x*
and [*measurable*]: $\bigwedge i.$ *f i* $\in$ *borel-measurable M*

and *fin*: $(\int^+ x.\ f\ i\ x\ \partial M) < \infty$
shows $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = (INF\ i.\ integral^N\ M\ (f\ i))$
proof −
{ fix $f :: nat \Rightarrow ennreal$ **and** j **assume** *decseq f*
then have $(INF\ i.\ f\ i) = (INF\ i.\ f\ (i + j))$
apply (*intro INF-eq*)
apply (*rule-tac x=i* **in** *bexI*)
apply (*auto simp*: *decseq-def le-fun-def*)
done }
note *INF-shift* = *this*
have *mono*: *AE x in M*. $\forall i.\ f\ (Suc\ i)\ x \leq f\ i\ x$
using *f* **by** (*auto simp*: *AE-all-countable*)
then have *AE x in M*. $(INF\ i.\ f\ i\ x) = (INF\ n.\ f\ (n + i)\ x)$
by *eventually-elim* (*auto intro*!: *decseq-SucI INF-shift*)
then have $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = (\int^+ x.\ (INF\ n.\ f\ (n + i)\ x)\ \partial M)$
by (*rule nn-integral-cong-AE*)
also have $\ldots = (INF\ n.\ (\int^+ x.\ f\ (n + i)\ x\ \partial M))$
by (*rule nn-integral-monotone-convergence-INF-AE'*) (*insert assms*, *auto*)
also have $\ldots = (INF\ n.\ (\int^+ x.\ f\ n\ x\ \partial M))$
by (*intro INF-shift*[*symmetric*] *decseq-SucI nn-integral-mono-AE f*)
finally show *?thesis* **.**
qed

lemma *nn-integral-monotone-convergence-INF-decseq*:
assumes *f*: *decseq f* **and** $*$: $\bigwedge i.\ f\ i \in borel\text{-}measurable\ M\ (\int^+ x.\ f\ i\ x\ \partial M) < \infty$
shows $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = (INF\ i.\ integral^N\ M\ (f\ i))$
using *nn-integral-monotone-convergence-INF-AE*[*of f M i*, *OF - ∗*] *f* **by** (*auto simp*: *decseq-Suc-iff le-fun-def*)

lemma *nn-integral-liminf*:
fixes $u :: nat \Rightarrow 'a \Rightarrow ennreal$
assumes *u*: $\bigwedge i.\ u\ i \in borel\text{-}measurable\ M$
shows $(\int^+ x.\ liminf\ (\lambda n.\ u\ n\ x)\ \partial M) \leq liminf\ (\lambda n.\ integral^N\ M\ (u\ n))$
proof −
have $(\int^+ x.\ liminf\ (\lambda n.\ u\ n\ x)\ \partial M) = (SUP\ n.\ \int^+ x.\ (INF\ i{:}\{n..\}.\ u\ i\ x)\ \partial M)$
unfolding *liminf-SUP-INF* **using** *u*
by (*intro nn-integral-monotone-convergence-SUP-AE*)
(*auto intro*!: *AE-I2 intro*: *INF-greatest INF-superset-mono*)
also have $\ldots \leq liminf\ (\lambda n.\ integral^N\ M\ (u\ n))$
by (*auto simp*: *liminf-SUP-INF intro*!: *SUP-mono INF-greatest nn-integral-mono INF-lower*)
finally show *?thesis* **.**
qed

lemma *nn-integral-limsup*:
fixes $u :: nat \Rightarrow 'a \Rightarrow ennreal$
assumes [*measurable*]: $\bigwedge i.\ u\ i \in borel\text{-}measurable\ M\ w \in borel\text{-}measurable\ M$
assumes *bounds*: $\bigwedge i.\ AE\ x\ in\ M.\ u\ i\ x \leq w\ x$ **and** *w*: $(\int^+ x.\ w\ x\ \partial M) < \infty$

shows *limsup* $(\lambda n.\ integral^N\ M\ (u\ n)) \leq (\int^+ x.\ limsup\ (\lambda n.\ u\ n\ x)\ \partial M)$
proof −
have *bnd*: *AE x in M*. $\forall i.\ u\ i\ x \leq w\ x$
using *bounds* **by** (*auto simp*: *AE-all-countable*)
then have $(\int^+ x.\ (SUP\ n.\ u\ n\ x)\ \partial M) \leq (\int^+ x.\ w\ x\ \partial M)$
by (*auto intro*!: *nn-integral-mono-AE elim*: *eventually-mono intro*: *SUP-least*)
then have $(\int^+ x.\ limsup\ (\lambda n.\ u\ n\ x)\ \partial M) = (INF\ n.\ \int^+ x.\ (SUP\ i{:}\{n..\}.\ u\ i\ x)\ \partial M)$
unfolding *limsup-INF-SUP* **using** *bnd w*
by (*intro nn-integral-monotone-convergence-INF-AE'*)
(*auto intro*!: *AE-I2 intro*: *SUP-least SUP-subset-mono*)
also have $\ldots \geq limsup\ (\lambda n.\ integral^N\ M\ (u\ n))$
by (*auto simp*: *limsup-INF-SUP intro*!: *INF-mono SUP-least exI nn-integral-mono SUP-upper*)
finally (*xtrans*) **show** *?thesis* **.**
qed

lemma *nn-integral-LIMSEQ*:
assumes *f*: *incseq f* $\bigwedge i.\ f\ i \in borel\text{-}measurable\ M$
and *u*: $\bigwedge x.\ (\lambda i.\ f\ i\ x) \longrightarrow u\ x$
shows $(\lambda n.\ integral^N\ M\ (f\ n)) \longrightarrow integral^N\ M\ u$
proof −
have $(\lambda n.\ integral^N\ M\ (f\ n)) \longrightarrow (SUP\ n.\ integral^N\ M\ (f\ n))$
using *f* **by** (*intro LIMSEQ-SUP*[*of* $\lambda n.\ integral^N\ M\ (f\ n)$] *incseq-nn-integral*)
also have $(SUP\ n.\ integral^N\ M\ (f\ n)) = integral^N\ M\ (\lambda x.\ SUP\ n.\ f\ n\ x)$
using *f* **by** (*intro nn-integral-monotone-convergence-SUP*[*symmetric*])
also have $integral^N\ M\ (\lambda x.\ SUP\ n.\ f\ n\ x) = integral^N\ M\ (\lambda x.\ u\ x)$
using *f* **by** (*subst LIMSEQ-SUP*[*THEN LIMSEQ-unique*, *OF - u*]) (*auto simp*: *incseq-def le-fun-def*)
finally show *?thesis* **.**
qed

lemma *nn-integral-dominated-convergence*:
assumes [*measurable*]:
$\bigwedge i.\ u\ i \in borel\text{-}measurable\ M\ u' \in borel\text{-}measurable\ M\ w \in borel\text{-}measurable\ M$
and *bound*: $\bigwedge j.\ AE\ x\ in\ M.\ u\ j\ x \leq w\ x$
and *w*: $(\int^+ x.\ w\ x\ \partial M) < \infty$
and *u'*: *AE x in M*. $(\lambda i.\ u\ i\ x) \longrightarrow u'\ x$
shows $(\lambda i.\ (\int^+ x.\ u\ i\ x\ \partial M)) \longrightarrow (\int^+ x.\ u'\ x\ \partial M)$
proof −
have $limsup\ (\lambda n.\ integral^N\ M\ (u\ n)) \leq (\int^+ x.\ limsup\ (\lambda n.\ u\ n\ x)\ \partial M)$
by (*intro nn-integral-limsup*[*OF - - bound w*]) *auto*
moreover have $(\int^+ x.\ limsup\ (\lambda n.\ u\ n\ x)\ \partial M) = (\int^+ x.\ u'\ x\ \partial M)$
using *u'* **by** (*intro nn-integral-cong-AE*, *eventually-elim*) (*metis tendsto-iff-Liminf-eq-Limsup sequentially-bot*)
moreover have $(\int^+ x.\ liminf\ (\lambda n.\ u\ n\ x)\ \partial M) = (\int^+ x.\ u'\ x\ \partial M)$
using *u'* **by** (*intro nn-integral-cong-AE*, *eventually-elim*) (*metis tendsto-iff-Liminf-eq-Limsup sequentially-bot*)

moreover have $(\int^+ x.\ liminf\ (\lambda n.\ u\ n\ x)\ \partial M) \leq liminf\ (\lambda n.\ integral^N\ M\ (u\ n))$
by (*intro nn-integral-liminf*) *auto*
moreover have $liminf\ (\lambda n.\ integral^N\ M\ (u\ n)) \leq limsup\ (\lambda n.\ integral^N\ M\ (u\ n))$
by (*intro Liminf-le-Limsup sequentially-bot*)
ultimately show *?thesis*
by (*intro Liminf-eq-Limsup*) *auto*
qed

lemma *inf-continuous-nn-integral*[*order-continuous-intros*]:
assumes *f*: $\bigwedge y.$ *inf-continuous* (*f y*)
assumes [*measurable*]: $\bigwedge x.\ (\lambda y.\ f\ y\ x) \in$ *borel-measurable M*
assumes *bnd*: $\bigwedge x.\ (\int^+ y.\ f\ y\ x\ \partial M) \neq \infty$
shows *inf-continuous* $(\lambda x.\ (\int^+ y.\ f\ y\ x\ \partial M))$
unfolding *inf-continuous-def*
proof *safe*
fix *C* :: *nat* $\Rightarrow$ *'b* **assume** *C*: *decseq C*
then show $(\int^+ y.\ f\ y\ (INFIMUM\ UNIV\ C)\ \partial M) = (INF\ i.\ \int^+ y.\ f\ y\ (C\ i)\ \partial M)$
using *inf-continuous-mono*[*OF f*] *bnd*
by (*auto simp add*: *inf-continuousD*[*OF f C*] *fun-eq-iff antimono-def mono-def le-fun-def less-top*
intro!: *nn-integral-monotone-convergence-INF-decseq*)
qed

lemma *nn-integral-null-set*:
assumes $N \in$ *null-sets M* **shows** $(\int^+ x.\ u\ x * indicator\ N\ x\ \partial M) = 0$
proof −
have $(\int^+ x.\ u\ x * indicator\ N\ x\ \partial M) = (\int^+ x.\ 0\ \partial M)$
proof (*intro nn-integral-cong-AE AE-I*)
show $\{x \in space\ M.\ u\ x * indicator\ N\ x \neq 0\} \subseteq N$
by (*auto simp*: *indicator-def*)
show (*emeasure M*) $N = 0$ $N \in sets\ M$
using *assms* **by** *auto*
qed
then show *?thesis* **by** *simp*
qed

lemma *nn-integral-0-iff*:
assumes *u*: $u \in$ *borel-measurable M*
shows $integral^N\ M\ u = 0 \longleftrightarrow emeasure\ M\ \{x \in space\ M.\ u\ x \neq 0\} = 0$
(**is** - $\longleftrightarrow$ (*emeasure M*) *?A* = *0*)
proof −
have *u-eq*: $(\int^+ x.\ u\ x * indicator\ ?A\ x\ \partial M) = integral^N\ M\ u$
by (*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
show *?thesis*
proof
assume (*emeasure M*) *?A* = *0*

```
    with nn-integral-null-set[of ?A M u] u
    show integral^N M u = 0 by (simp add: u-eq null-sets-def)
  next
    assume *: integral^N M u = 0
    let ?M = λn. {x ∈ space M. 1 ≤ real (n::nat) * u x}
    have 0 = (SUP n. (emeasure M) (?M n ∩ ?A))
    proof −
      { fix n :: nat
        from nn-integral-Markov-inequality[OF u, of ?A of-nat n] u
        have (emeasure M) (?M n ∩ ?A) ≤ 0
          by (simp add: ennreal-of-nat-eq-real-of-nat u-eq *)
        moreover have 0 ≤ (emeasure M) (?M n ∩ ?A) using u by auto
        ultimately have (emeasure M) (?M n ∩ ?A) = 0 by auto }
      thus ?thesis by simp
    qed
    also have ... = (emeasure M) (⋃ n. ?M n ∩ ?A)
    proof (safe intro!: SUP-emeasure-incseq)
      fix n show ?M n ∩ ?A ∈ sets M
        using u by (auto intro!: sets.Int)
    next
      show incseq (λn. {x ∈ space M. 1 ≤ real n * u x} ∩ {x ∈ space M. u x ≠
0})
      proof (safe intro!: incseq-SucI)
        fix n :: nat and x
        assume *: 1 ≤ real n * u x
        also have real n * u x ≤ real (Suc n) * u x
          by (auto intro!: mult-right-mono)
        finally show 1 ≤ real (Suc n) * u x by auto
      qed
    qed
    also have ... = (emeasure M) {x∈space M. 0 < u x}
    proof (safe intro!: arg-cong[where f=(emeasure M)])
      fix x assume 0 < u x and [simp, intro]: x ∈ space M
      show x ∈ (⋃ n. ?M n ∩ ?A)
      proof (cases u x rule: ennreal-cases)
        case (real r) with ‹0 < u x› have 0 < r by auto
        obtain j :: nat where 1 / r ≤ real j using real-arch-simple ..
        hence 1 / r * r ≤ real j * r unfolding mult-le-cancel-right using ‹0 < r›
by auto
        hence 1 ≤ real j * r using real ‹0 < r› by auto
        thus ?thesis using ‹0 < r› real
       by (auto simp: ennreal-of-nat-eq-real-of-nat ennreal-1[symmetric] ennreal-mult[symmetric]
                simp del: ennreal-1)
      qed (insert ‹0 < u x›, auto simp: ennreal-mult-top)
    qed (auto simp: zero-less-iff-neq-zero)
    finally show emeasure M ?A = 0
      by (simp add: zero-less-iff-neq-zero)
  qed
qed
```

lemma *nn-integral-0-iff-AE*:
assumes *u*: $u \in$ *borel-measurable M*
shows $integral^N\ M\ u = 0 \longleftrightarrow (AE\ x\ in\ M.\ u\ x = 0)$
proof −
have *sets*: $\{x \in space\ M.\ u\ x \neq 0\} \in sets\ M$
using *u* **by** *auto*
show $integral^N\ M\ u = 0 \longleftrightarrow (AE\ x\ in\ M.\ u\ x = 0)$
using *nn-integral-0-iff*[*of u*] *AE-iff-null*[*OF sets*] *u* **by** *auto*
qed

lemma *AE-iff-nn-integral*:
$\{x \in space\ M.\ P\ x\} \in sets\ M \Longrightarrow (AE\ x\ in\ M.\ P\ x) \longleftrightarrow integral^N\ M\ (indicator\ \{x.\ \neg\ P\ x\}) = 0$
by (*subst nn-integral-0-iff-AE*) (*auto simp*: *indicator-def*[*abs-def*])

lemma *nn-integral-less*:
assumes [*measurable*]: $f \in$ *borel-measurable M* $g \in$ *borel-measurable M*
assumes *f*: $(\int^+ x.\ f\ x\ \partial M) \neq \infty$
assumes *ord*: $AE\ x\ in\ M.\ f\ x \leq g\ x\ \neg\ (AE\ x\ in\ M.\ g\ x \leq f\ x)$
shows $(\int^+ x.\ f\ x\ \partial M) < (\int^+ x.\ g\ x\ \partial M)$
proof −
have $0 < (\int^+ x.\ g\ x - f\ x\ \partial M)$
proof (*intro order-le-neq-trans notI*)
assume $0 = (\int^+ x.\ g\ x - f\ x\ \partial M)$
then have $AE\ x\ in\ M.\ g\ x - f\ x = 0$
using *nn-integral-0-iff-AE*[*of* $\lambda x.\ g\ x - f\ x\ M$] **by** *simp*
with *ord*(*1*) **have** $AE\ x\ in\ M.\ g\ x \leq f\ x$
by *eventually-elim* (*auto simp*: *ennreal-minus-eq-0*)
with *ord* **show** *False*
by *simp*
qed *simp*
also have $\ldots = (\int^+ x.\ g\ x\ \partial M) - (\int^+ x.\ f\ x\ \partial M)$
using *f* **by** (*subst nn-integral-diff*) (*auto simp*: *ord*)
finally show *?thesis*
using *f* **by** (*auto dest*!: *ennreal-minus-pos-iff*[*rotated*] *simp*: *less-top*)
qed

lemma *nn-integral-subalgebra*:
assumes *f*: $f \in$ *borel-measurable N*
and *N*: *sets N* $\subseteq$ *sets M space N* = *space M* $\bigwedge A.\ A \in sets\ N \Longrightarrow emeasure\ N\ A = emeasure\ M\ A$
shows $integral^N\ N\ f = integral^N\ M\ f$
proof −
have [*simp*]: $\bigwedge f :: 'a \Rightarrow ennreal.\ f \in$ *borel-measurable N* $\Longrightarrow f \in$ *borel-measurable M*
using *N* **by** (*auto simp*: *measurable-def*)
have [*simp*]: $\bigwedge P.\ (AE\ x\ in\ N.\ P\ x) \Longrightarrow (AE\ x\ in\ M.\ P\ x)$
using *N* **by** (*auto simp add*: *eventually-ae-filter null-sets-def subset-eq*)

have [*simp*]: $\bigwedge A.\ A \in sets\ N \Longrightarrow A \in sets\ M$
using N **by** *auto*
from f **show** *?thesis*
apply *induct*
apply (*simp-all add*: *nn-integral-add nn-integral-cmult nn-integral-monotone-convergence-SUP* N)
apply (*auto intro*!: *nn-integral-cong cong*: *nn-integral-cong simp*: $N(2)$[*symmetric*])
done
qed

lemma *nn-integral-nat-function*:
fixes $f :: 'a \Rightarrow nat$
assumes $f \in measurable\ M\ (count\text{-}space\ UNIV)$
shows $(\int^+ x.\ of\text{-}nat\ (f\ x)\ \partial M) = (\sum t.\ emeasure\ M\ \{x \in space\ M.\ t < f\ x\})$
proof −
def $F \equiv \lambda i.\ \{x \in space\ M.\ i < f\ x\}$
with *assms* **have** [*measurable*]: $\bigwedge i.\ F\ i \in sets\ M$
by *auto*

{ fix x **assume** $x \in space\ M$
have $(\lambda i.\ if\ i < f\ x\ then\ 1\ else\ 0)\ sums\ (of\text{-}nat\ (f\ x)::real)$
using *sums-If-finite*[*of* $\lambda i.\ i < f\ x\ \lambda\text{-}.\ 1::real$] **by** *simp*
then have $(\lambda i.\ ennreal\ (if\ i < f\ x\ then\ 1\ else\ 0))\ sums\ of\text{-}nat(f\ x)$
unfolding *ennreal-of-nat-eq-real-of-nat*
by (*subst sums-ennreal*) *auto*
moreover have $\bigwedge i.\ ennreal\ (if\ i < f\ x\ then\ 1\ else\ 0) = indicator\ (F\ i)\ x$
using ‹$x \in space\ M$› **by** (*simp add*: *one-ennreal-def F-def*)
ultimately have $of\text{-}nat\ (f\ x) = (\sum i.\ indicator\ (F\ i)\ x :: ennreal)$
by (*simp add*: *sums-iff*) **}**
then have $(\int^+ x.\ of\text{-}nat\ (f\ x)\ \partial M) = (\int^+ x.\ (\sum i.\ indicator\ (F\ i)\ x)\ \partial M)$
by (*simp cong*: *nn-integral-cong*)
also have $\ldots = (\sum i.\ emeasure\ M\ (F\ i))$
by (*simp add*: *nn-integral-suminf*)
finally show *?thesis*
by (*simp add*: *F-def*)
qed

lemma *nn-integral-lfp*:
assumes *sets*[*simp*]: $\bigwedge s.\ sets\ (M\ s) = sets\ N$
assumes f: *sup-continuous f*
assumes g: *sup-continuous g*
assumes *meas*: $\bigwedge F.\ F \in borel\text{-}measurable\ N \Longrightarrow f\ F \in borel\text{-}measurable\ N$
assumes *step*: $\bigwedge F\ s.\ F \in borel\text{-}measurable\ N \Longrightarrow integral^N\ (M\ s)\ (f\ F) = g\ (\lambda s.\ integral^N\ (M\ s)\ F)\ s$
shows $(\int^+ \omega.\ lfp\ f\ \omega\ \partial M\ s) = lfp\ g\ s$
proof (*subst lfp-transfer-bounded*[**where** α=$\lambda F\ s.\ \int^+ x.\ F\ x\ \partial M\ s$ **and** g=g **and** f=f **and** P=$\lambda f.\ f \in borel\text{-}measurable\ N$, *symmetric*])
fix $C :: nat \Rightarrow 'b \Rightarrow ennreal$ **assume** *incseq* C $\bigwedge i.\ C\ i \in borel\text{-}measurable\ N$
then show $(\lambda s.\ \int^+ x.\ (SUP\ i.\ C\ i)\ x\ \partial M\ s) = (SUP\ i.\ (\lambda s.\ \int^+ x.\ C\ i\ x\ \partial M\ s))$

unfolding *SUP-apply*[*abs-def*]
by (*subst nn-integral-monotone-convergence-SUP*)
(*auto simp*: *mono-def fun-eq-iff intro*!: *arg-cong2*[**where** *f*=*emeasure*] *cong*: *measurable-cong-sets*)
qed (*auto simp add*: *step le-fun-def SUP-apply*[*abs-def*] *bot-fun-def bot-ennreal intro*!: *meas f g*)

lemma *nn-integral-gfp*:
assumes *sets*[*simp*]: $\bigwedge s.\ sets\ (M\ s) = sets\ N$
assumes *f*: *inf-continuous f* **and** *g*: *inf-continuous g*
assumes *meas*: $\bigwedge F.\ F \in borel\text{-}measurable\ N \Longrightarrow f\ F \in borel\text{-}measurable\ N$
assumes *bound*: $\bigwedge F\ s.\ F \in borel\text{-}measurable\ N \Longrightarrow (\int^+ x.\ f\ F\ x\ \partial M\ s) < \infty$
assumes *non-zero*: $\bigwedge s.\ emeasure\ (M\ s)\ (space\ (M\ s)) \neq 0$
assumes *step*: $\bigwedge F\ s.\ F \in borel\text{-}measurable\ N \Longrightarrow integral^N\ (M\ s)\ (f\ F) = g\ (\lambda s.\ integral^N\ (M\ s)\ F)\ s$
shows $(\int^+ \omega.\ gfp\ f\ \omega\ \partial M\ s) = gfp\ g\ s$
proof (*subst gfp-transfer-bounded*[**where** α=$\lambda F\ s.\ \int^+ x.\ F\ x\ \partial M\ s$ **and** *g*=*g* **and** *f*=*f*
and P=$\lambda F.\ F \in borel\text{-}measurable\ N \wedge (\forall s.\ (\int^+ x.\ F\ x\ \partial M\ s) < \infty)$, *symmetric*])
fix *C* :: *nat* ⇒ *'b* ⇒ *ennreal* **assume** *decseq C* $\bigwedge i.\ C\ i \in borel\text{-}measurable\ N \wedge (\forall s.\ integral^N\ (M\ s)\ (C\ i) < \infty)$
then show $(\lambda s.\ \int^+ x.\ (INF\ i.\ C\ i)\ x\ \partial M\ s) = (INF\ i.\ (\lambda s.\ \int^+ x.\ C\ i\ x\ \partial M\ s))$
unfolding *INF-apply*[*abs-def*]
by (*subst nn-integral-monotone-convergence-INF-decseq*)
(*auto simp*: *mono-def fun-eq-iff intro*!: *arg-cong2*[**where** *f*=*emeasure*] *cong*: *measurable-cong-sets*)
next
show $\bigwedge x.\ g\ x \leq (\lambda s.\ integral^N\ (M\ s)\ (f\ top))$
by (*subst step*)
(*auto simp add*: *top-fun-def less-le non-zero le-fun-def ennreal-top-mult*
cong del: *if-cong intro*!: *monoD*[*OF inf-continuous-mono*[*OF g*], *THEN le-funD*])
next
fix *C* **assume** $\bigwedge i{::}nat.\ C\ i \in borel\text{-}measurable\ N \wedge (\forall s.\ integral^N\ (M\ s)\ (C\ i) < \infty)\ decseq\ C$
with *bound* **show** $INFIMUM\ UNIV\ C \in borel\text{-}measurable\ N \wedge (\forall s.\ integral^N\ (M\ s)\ (INFIMUM\ UNIV\ C) < \infty)$
unfolding *INF-apply*[*abs-def*]
by (*subst nn-integral-monotone-convergence-INF-decseq*)
(*auto simp*: *INF-less-iff cong*: *measurable-cong-sets intro*!: *borel-measurable-INF*)
next
show $\bigwedge x.\ x \in borel\text{-}measurable\ N \wedge (\forall s.\ integral^N\ (M\ s)\ x < \infty) \Longrightarrow (\lambda s.\ integral^N\ (M\ s)\ (f\ x)) = g\ (\lambda s.\ integral^N\ (M\ s)\ x)$
by (*subst step*) *auto*
qed (*insert bound*, *auto simp add*: *le-fun-def INF-apply*[*abs-def*] *top-fun-def intro*!: *meas f g*)

5.4 Integral under concrete measures

lemma *nn-integral-empty*:
assumes *space M = {}*
shows *nn-integral M f = 0*
proof −
have *(∫⁺ x. f x ∂M) = (∫⁺ x. 0 ∂M)*
by(*rule nn-integral-cong*)(*simp add*: *assms*)
thus *?thesis* **by** *simp*
qed

5.4.1 Distributions

lemma *nn-integral-distr*:
assumes *T*: *T ∈ measurable M M'* **and** *f*: *f ∈ borel-measurable (distr M M' T)*
shows *integral^N (distr M M' T) f = (∫⁺ x. f (T x) ∂M)*
using *f*
proof *induct*
case (*cong f g*)
with *T* **show** *?case*
apply (*subst nn-integral-cong*[*of - f g*])
apply *simp*
apply (*subst nn-integral-cong*[*of - λx. f (T x) λx. g (T x)*])
apply (*simp add*: *measurable-def Pi-iff*)
apply *simp*
done
next
case (*set A*)
then have *eq*: *⋀x. x ∈ space M ⟹ indicator A (T x) = indicator (T −‘ A ∩ space M) x*
by (*auto simp*: *indicator-def*)
from *set T* **show** *?case*
by (*subst nn-integral-cong*[*OF eq*])
(*auto simp add*: *emeasure-distr intro*!: *nn-integral-indicator*[*symmetric*] *measurable-sets*)
qed (*simp-all add*: *measurable-compose*[*OF T*] *T nn-integral-cmult nn-integral-add nn-integral-monotone-convergence-SUP le-fun-def incseq-def*)

5.4.2 Counting space

lemma *simple-function-count-space*[*simp*]:
simple-function (count-space A) f ⟷ finite (f ‘ A)
unfolding *simple-function-def* **by** *simp*

lemma *nn-integral-count-space*:
assumes *A*: *finite {a∈A. 0 < f a}*
shows *integral^N (count-space A) f = (∑ a|a∈A ∧ 0 < f a. f a)*
proof −
have ∗: *(∫⁺x. max 0 (f x) ∂count-space A) =*
(∫⁺ x. (∑ a|a∈A ∧ 0 < f a. f a ∗ indicator {a} x) ∂count-space A)

by (*auto intro*!: *nn-integral-cong*
simp add: *indicator-def if-distrib setsum.If-cases*[*OF A*] *max-def le-less*)
also have ... = ($\sum$ *a*|*a*∈*A* ∧ *0* < *f a*. ∫$^+$ *x*. *f a* ∗ *indicator* {*a*} *x* ∂*count-space A*)
by (*subst nn-integral-setsum*) (*simp-all add*: *AE-count-space less-imp-le*)
also have ... = ($\sum$ *a*|*a*∈*A* ∧ *0* < *f a*. *f a*)
by (*auto intro*!: *setsum.cong simp*: *one-ennreal-def*[*symmetric*] *max-def*)
finally show *?thesis* **by** (*simp add*: *max.absorb2*)
qed

lemma *nn-integral-count-space-finite*:
finite A ⟹ (∫$^+$*x*. *f x* ∂*count-space A*) = ($\sum$ *a*∈*A*. *f a*)
by (*auto intro*!: *setsum.mono-neutral-left simp*: *nn-integral-count-space less-le*)

lemma *nn-integral-count-space′*:
assumes *finite A* ⋀*x*. *x* ∈ *B* ⟹ *x* ∉ *A* ⟹ *f x* = *0 A* ⊆ *B*
shows (∫$^+$*x*. *f x* ∂*count-space B*) = ($\sum$ *x*∈*A*. *f x*)
proof −
have (∫$^+$*x*. *f x* ∂*count-space B*) = ($\sum$ *a* | *a* ∈ *B* ∧ *0* < *f a*. *f a*)
using *assms*(*2*,*3*)
by (*intro nn-integral-count-space finite-subset*[*OF* - ‹*finite A*›]) (*auto simp*: *less-le*)
also have ... = ($\sum$ *a*∈*A*. *f a*)
using *assms* **by** (*intro setsum.mono-neutral-cong-left*) (*auto simp*: *less-le*)
finally show *?thesis* **.**
qed

lemma *nn-integral-bij-count-space*:
assumes *g*: *bij-betw g A B*
shows (∫$^+$*x*. *f* (*g x*) ∂*count-space A*) = (∫$^+$*x*. *f x* ∂*count-space B*)
using *g*[*THEN bij-betw-imp-funcset*]
by (*subst distr-bij-count-space*[*OF g*, *symmetric*])
(*auto intro*!: *nn-integral-distr*[*symmetric*])

lemma *nn-integral-indicator-finite*:
fixes *f* :: *′a* ⇒ *ennreal*
assumes *f*: *finite A* **and** [*measurable*]: ⋀*a*. *a* ∈ *A* ⟹ {*a*} ∈ *sets M*
shows (∫$^+$*x*. *f x* ∗ *indicator A x* ∂*M*) = ($\sum$ *x*∈*A*. *f x* ∗ *emeasure M* {*x*})
proof −
from *f* **have** (∫$^+$*x*. *f x* ∗ *indicator A x* ∂*M*) = (∫$^+$*x*. ($\sum$ *a*∈*A*. *f a* ∗ *indicator* {*a*} *x*) ∂*M*)
by (*intro nn-integral-cong*) (*auto simp*: *indicator-def if-distrib*[**where** *f*=λ*a*. *x* ∗ *a* **for** *x*] *setsum.If-cases*)
also have ... = ($\sum$ *a*∈*A*. *f a* ∗ *emeasure M* {*a*})
by (*subst nn-integral-setsum*) *auto*
finally show *?thesis* **.**
qed

lemma *nn-integral-count-space-nat*:

fixes $f :: nat \Rightarrow ennreal$
shows $(\int^+ i.\ f\ i\ \partial count\text{-}space\ UNIV) = (\sum i.\ f\ i)$
proof −
have $(\int^+ i.\ f\ i\ \partial count\text{-}space\ UNIV) =$
$(\int^+ i.\ (\sum j.\ f\ j * indicator\ \{j\}\ i)\ \partial count\text{-}space\ UNIV)$
proof (*intro nn-integral-cong*)
fix i
have $f\ i = (\sum j \in \{i\}.\ f\ j * indicator\ \{j\}\ i)$
by *simp*
also have $\ldots = (\sum j.\ f\ j * indicator\ \{j\}\ i)$
by (*rule suminf-finite*[*symmetric*]) *auto*
finally show $f\ i = (\sum j.\ f\ j * indicator\ \{j\}\ i)$ **.**
qed
also have $\ldots = (\sum j.\ (\int^+ i.\ f\ j * indicator\ \{j\}\ i\ \partial count\text{-}space\ UNIV))$
by (*rule nn-integral-suminf*) *auto*
finally show *?thesis*
by *simp*
qed

lemma *nn-integral-enat-function*:
assumes *f*: $f \in measurable\ M\ (count\text{-}space\ UNIV)$
shows $(\int^+ x.\ ennreal\text{-}of\text{-}enat\ (f\ x)\ \partial M) = (\sum t.\ emeasure\ M\ \{x \in space\ M.\ t < f\ x\})$
proof −
def $F \equiv \lambda i{::}nat.\ \{x \in space\ M.\ i < f\ x\}$
with *assms* **have** [*measurable*]: $\bigwedge i.\ F\ i \in sets\ M$
by *auto*

{ fix x **assume** $x \in space\ M$
have $(\lambda i{::}nat.\ if\ i < f\ x\ then\ 1\ else\ 0)\ sums\ ennreal\text{-}of\text{-}enat\ (f\ x)$
using *sums-If-finite*[*of* $\lambda r.\ r < f\ x\ \lambda\text{-}.\ 1 :: ennreal$]
by (*cases* $f\ x$) (*simp-all add*: *sums-def of-nat-tendsto-top-ennreal*)
also have $(\lambda i.\ (if\ i < f\ x\ then\ 1\ else\ 0)) = (\lambda i.\ indicator\ (F\ i)\ x)$
using ‹$x \in space\ M$› **by** (*simp add*: *one-ennreal-def F-def fun-eq-iff*)
finally have $ennreal\text{-}of\text{-}enat\ (f\ x) = (\sum i.\ indicator\ (F\ i)\ x)$
by (*simp add*: *sums-iff*) **}**
then have $(\int^+ x.\ ennreal\text{-}of\text{-}enat\ (f\ x)\ \partial M) = (\int^+ x.\ (\sum i.\ indicator\ (F\ i)\ x)\ \partial M)$
by (*simp cong*: *nn-integral-cong*)
also have $\ldots = (\sum i.\ emeasure\ M\ (F\ i))$
by (*simp add*: *nn-integral-suminf*)
finally show *?thesis*
by (*simp add*: *F-def*)
qed

lemma *nn-integral-count-space-nn-integral*:
fixes $f :: 'i \Rightarrow 'a \Rightarrow ennreal$
assumes *countable I* **and** [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in borel\text{-}measurable\ M$
shows $(\int^+ x.\ \int^+ i.\ f\ i\ x\ \partial count\text{-}space\ I\ \partial M) = (\int^+ i.\ \int^+ x.\ f\ i\ x\ \partial M\ \partial count\text{-}space$

I)
proof *cases*
assume *finite I* **then show** *?thesis*
by (*simp add*: *nn-integral-count-space-finite nn-integral-setsum*)
next
assume *infinite I*
then have [*simp*]: $I \neq \{\}$
by *auto*
note $* = $ *bij-betw-from-nat-into*[*OF* ‹*countable I*› ‹*infinite I*›]
have $**$: $\bigwedge f.\ (\bigwedge i.\ 0 \leq f\ i) \Longrightarrow (\int^+ i.\ f\ i\ \partial$*count-space I*$) = (\sum n.\ f$ (*from-nat-into I n*))
by (*simp add*: *nn-integral-bij-count-space*[*symmetric*, *OF* $*$] *nn-integral-count-space-nat*)
show *?thesis*
by (*simp add*: $**$ *nn-integral-suminf from-nat-into*)
qed

lemma *emeasure-UN-countable*:
assumes *sets*[*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow X\ i \in$ *sets M* **and** *I*[*simp*]: *countable I*
assumes *disj*: *disjoint-family-on X I*
shows *emeasure M* (*UNION I X*) $= (\int^+ i.$ *emeasure M* (*X i*) ∂*count-space I*)
proof −
have *eq*: $\bigwedge x.$ *indicator* (*UNION I X*) *x* $= \int^+ i.$ *indicator* (*X i*) *x* ∂*count-space I*
proof *cases*
fix *x* **assume** *x*: $x \in$ *UNION I X*
then obtain *j* **where** *j*: $x \in X\ j\ j \in I$
by *auto*
with *disj* **have** $\bigwedge i.\ i \in I \Longrightarrow$ *indicator* (*X i*) *x* = (*indicator* $\{j\}$ *i*::*ennreal*)
by (*auto simp*: *disjoint-family-on-def split*: *split-indicator*)
with *x j* **show** *?thesis x*
by (*simp cong*: *nn-integral-cong-simp*)
qed (*auto simp*: *nn-integral-0-iff-AE*)

note *sets.countable-UN'*[*unfolded subset-eq*, *measurable*]
have *emeasure M* (*UNION I X*) $= (\int^+ x.$ *indicator* (*UNION I X*) *x* ∂M)
by *simp*
also have $\ldots = (\int^+ i.\ \int^+ x.$ *indicator* (*X i*) *x* $\partial M\ \partial$*count-space I*)
by (*simp add*: *eq nn-integral-count-space-nn-integral*)
finally show *?thesis*
by (*simp cong*: *nn-integral-cong-simp*)
qed

lemma *emeasure-countable-singleton*:
assumes *sets*: $\bigwedge x.\ x \in X \Longrightarrow \{x\} \in$ *sets M* **and** *X*: *countable X*
shows *emeasure M X* $= (\int^+ x.$ *emeasure M* $\{x\}$ ∂*count-space X*)
proof −
have *emeasure M* $(\bigcup i \in X.\ \{i\}) = (\int^+ x.$ *emeasure M* $\{x\}$ ∂*count-space X*)
using *assms* **by** (*intro emeasure-UN-countable*) (*auto simp*: *disjoint-family-on-def*)
also have $(\bigcup i \in X.\ \{i\}) = X$ **by** *auto*

finally show *?thesis* **.**
qed

lemma *measure-eqI-countable*:
assumes [*simp*]: *sets M = Pow A sets N = Pow A* **and** *A*: *countable A*
assumes *eq*: $\bigwedge a.\ a \in A \Longrightarrow$ *emeasure M {a} = emeasure N {a}*
shows *M = N*
proof (*rule measure-eqI*)
fix *X* **assume** *X ∈ sets M*
then have *X*: *X ⊆ A* **by** *auto*
moreover with *A* **have** *countable X* **by** (*auto dest*: *countable-subset*)
ultimately have
emeasure M X = (∫⁺a. emeasure M {a} ∂count-space X)
emeasure N X = (∫⁺a. emeasure N {a} ∂count-space X)
by (*auto intro*!: *emeasure-countable-singleton*)
moreover have *(∫⁺a. emeasure M {a} ∂count-space X) = (∫⁺a. emeasure N {a} ∂count-space X)*
using *X* **by** (*intro nn-integral-cong eq*) *auto*
ultimately show *emeasure M X = emeasure N X*
by *simp*
qed *simp*

lemma *measure-eqI-countable-AE*:
assumes [*simp*]: *sets M = UNIV sets N = UNIV*
assumes *ae*: *AE x in M. x ∈ Ω AE x in N. x ∈ Ω* **and** [*simp*]: *countable Ω*
assumes *eq*: $\bigwedge x.\ x \in \Omega \Longrightarrow$ *emeasure M {x} = emeasure N {x}*
shows *M = N*
proof (*rule measure-eqI*)
fix *A*
have *emeasure N A = emeasure N {x∈Ω. x ∈ A}*
using *ae* **by** (*intro emeasure-eq-AE*) *auto*
also have *... = (∫⁺x. emeasure N {x} ∂count-space {x∈Ω. x ∈ A})*
by (*intro emeasure-countable-singleton*) *auto*
also have *... = (∫⁺x. emeasure M {x} ∂count-space {x∈Ω. x ∈ A})*
by (*intro nn-integral-cong eq*[*symmetric*]) *auto*
also have *... = emeasure M {x∈Ω. x ∈ A}*
by (*intro emeasure-countable-singleton*[*symmetric*]) *auto*
also have *... = emeasure M A*
using *ae* **by** (*intro emeasure-eq-AE*) *auto*
finally show *emeasure M A = emeasure N A* **..**
qed *simp*

lemma *nn-integral-monotone-convergence-SUP-nat*:
fixes *f* :: *'a ⇒ nat ⇒ ennreal*
assumes *chain*: *Complete-Partial-Order.chain op ≤ (f ‘ Y)*
and *nonempty*: *Y ≠ {}*
shows *(∫⁺ x. (SUP i:Y. f i x) ∂count-space UNIV) = (SUP i:Y. (∫⁺ x. f i x ∂count-space UNIV))*
(**is** *?lhs = ?rhs* **is** *integral^N ?M - = -*)

proof (*rule order-class.order.antisym*)
show *?rhs* ≤ *?lhs*
by (*auto intro*!: *SUP-least SUP-upper nn-integral-mono*)
next
have ∃ *g*. *incseq g* ∧ *range g* ⊆ (λ*i*. *f i x*) ‘ *Y* ∧ (*SUP i*:*Y*. *f i x*) = (*SUP i*. *g i*) **for** *x*
by (*rule ennreal-Sup-countable-SUP*) (*simp add*: *nonempty*)
then obtain *g* **where** *incseq*: ⋀*x*. *incseq* (*g x*)
and *range*: ⋀*x*. *range* (*g x*) ⊆ (λ*i*. *f i x*) ‘ *Y*
and *sup*: ⋀*x*. (*SUP i*:*Y*. *f i x*) = (*SUP i*. *g x i*) **by** *moura*
from *incseq* **have** *incseq'*: *incseq* (λ*i x*. *g x i*)
by(*blast intro*: *incseq-SucI le-funI dest*: *incseq-SucD*)

have *?lhs* = ∫⁺ *x*. (*SUP i*. *g x i*) ∂*?M* **by**(*simp add*: *sup*)
also have ... = (*SUP i*. ∫⁺ *x*. *g x i* ∂*?M*) **using** *incseq'*
by(*rule nn-integral-monotone-convergence-SUP*) *simp*
also have ... ≤ (*SUP i*:*Y*. ∫⁺ *x*. *f i x* ∂*?M*)
proof(*rule SUP-least*)
fix *n*
have ⋀*x*. ∃ *i*. *g x n* = *f i x* ∧ *i* ∈ *Y* **using** *range* **by** *blast*
then obtain *I* **where** *I*: ⋀*x*. *g x n* = *f* (*I x*) *x* ⋀*x*. *I x* ∈ *Y* **by** *moura*

have (∫⁺ *x*. *g x n* ∂*count-space UNIV*) = (∑ *x*. *g x n*)
by(*rule nn-integral-count-space-nat*)
also have ... = (*SUP m*. ∑ *x*<*m*. *g x n*)
by(*rule suminf-eq-SUP*)
also have ... ≤ (*SUP i*:*Y*. ∫⁺ *x*. *f i x* ∂*?M*)
proof(*rule SUP-mono*)
fix *m*
show ∃ *m'*∈*Y*. (∑ *x*<*m*. *g x n*) ≤ (∫⁺ *x*. *f m' x* ∂*?M*)
proof(*cases m* > *0*)
case *False*
thus *?thesis* **using** *nonempty* **by** *auto*
next
case *True*
let *?Y* = *I* ‘ {..<*m*}
have *f* ‘ *?Y* ⊆ *f* ‘ *Y* **using** *I* **by** *auto*
with *chain* **have** *chain'*: *Complete-Partial-Order.chain op* ≤ (*f* ‘ *?Y*) **by**(*rule chain-subset*)
hence *Sup* (*f* ‘ *?Y*) ∈ *f* ‘ *?Y*
by(*rule ccpo-class.in-chain-finite*)(*auto simp add*: *True lessThan-empty-iff*)
then obtain *m'* **where** *m'* < *m* **and** *m'*: (*SUP i*:*?Y*. *f i*) = *f* (*I m'*) **by** *auto*
have *I m'* ∈ *Y* **using** *I* **by** *blast*
have (∑ *x*<*m*. *g x n*) ≤ (∑ *x*<*m*. *f* (*I m'*) *x*)
proof(*rule setsum-mono*)
fix *x*
assume *x* ∈ {..<*m*}
hence *x* < *m* **by** *simp*

```
        have g x n = f (I x) x by(simp add: I)
        also have ... ≤ (SUP i:?Y. f i) x unfolding Sup-fun-def image-image
          using ‹x ∈ {..<m}› by (rule Sup-upper [OF imageI])
        also have ... = f (I m') x unfolding m' by simp
        finally show g x n ≤ f (I m') x .
      qed
      also have ... ≤ (SUP m. (∑x<m. f (I m') x))
        by(rule SUP-upper) simp
      also have ... = (∑x. f (I m') x)
        by(rule suminf-eq-SUP[symmetric])
      also have ... = (∫+ x. f (I m') x ∂?M)
        by(rule nn-integral-count-space-nat[symmetric])
      finally show ?thesis using ‹I m' ∈ Y› by blast
    qed
  qed
  finally show (∫+ x. g x n ∂count-space UNIV) ≤ ... .
 qed
 finally show ?lhs ≤ ?rhs .
qed

lemma power-series-tendsto-at-left:
 assumes nonneg: ⋀i. 0 ≤ f i and summable: ⋀z. 0 ≤ z ⟹ z < 1 ⟹ summable
(λn. f n * z^n)
  shows ((λz. ennreal (∑n. f n * z^n)) ⟶ (∑n. ennreal (f n))) (at-left
(1::real))
proof (intro tendsto-at-left-sequentially)
 show 0 < (1::real) by simp
 fix S :: nat ⇒ real assume S: ⋀n. S n < 1 ⋀n. 0 < S n S ⟶ 1 incseq S
 then have S-nonneg: ⋀i. 0 ≤ S i by (auto intro: less-imp-le)

 have (λi. (∫+n. f n * S i^n ∂count-space UNIV)) ⟶ (∫+n. ennreal (f n)
∂count-space UNIV)
 proof (rule nn-integral-LIMSEQ)
  show incseq (λi n. ennreal (f n * S i^n))
   using S by (auto intro!: mult-mono power-mono nonneg ennreal-leI
                simp: incseq-def le-fun-def less-imp-le)
  fix n have (λi. ennreal (f n * S i^n)) ⟶ ennreal (f n * 1^n)
   by (intro tendsto-intros tendsto-ennrealI S)
  then show (λi. ennreal (f n * S i^n)) ⟶ ennreal (f n)
   by simp
 qed (auto simp: S-nonneg intro!: mult-nonneg-nonneg nonneg)
 also have (λi. (∫+n. f n * S i^n ∂count-space UNIV)) = (λi. ∑n. f n * S i^n)
  by (subst nn-integral-count-space-nat)
    (intro ext suminf-ennreal2 mult-nonneg-nonneg nonneg S-nonneg
          zero-le-power summable S)+
 also have (∫+n. ennreal (f n) ∂count-space UNIV) = (∑n. ennreal (f n))
  by (simp add: nn-integral-count-space-nat nonneg)
 finally show (λn. ennreal (∑na. f na * S n ^ na)) ⟶ (∑n. ennreal (f n))
.
```

qed

5.4.3 Measures with Restricted Space

lemma *simple-function-restrict-space-ennreal*:
fixes $f :: 'a \Rightarrow ennreal$
assumes $\Omega \cap space\ M \in sets\ M$
shows *simple-function* (*restrict-space* $M\ \Omega$) $f \longleftrightarrow$ *simple-function* M ($\lambda x.\ f\ x * indicator\ \Omega\ x$)
proof −
{ assume *finite* (f ‘ *space* (*restrict-space* $M\ \Omega$))
then have *finite* (f ‘ *space* (*restrict-space* $M\ \Omega$) $\cup\ \{0\}$) **by** *simp*
then have *finite* (($\lambda x.\ f\ x * indicator\ \Omega\ x$) ‘ *space* M)
by (*rule rev-finite-subset*) (*auto split*: *split-indicator simp*: *space-restrict-space*)
}
moreover
{ assume *finite* (($\lambda x.\ f\ x * indicator\ \Omega\ x$) ‘ *space* M)
then have *finite* (f ‘ *space* (*restrict-space* $M\ \Omega$))
by (*rule rev-finite-subset*) (*auto split*: *split-indicator simp*: *space-restrict-space*)
}
ultimately show *?thesis*
unfolding
simple-function-iff-borel-measurable borel-measurable-restrict-space-iff-ennreal[*OF assms*]
by *auto*
qed

lemma *simple-function-restrict-space*:
fixes $f :: 'a \Rightarrow 'b$::*real-normed-vector*
assumes $\Omega \cap space\ M \in sets\ M$
shows *simple-function* (*restrict-space* $M\ \Omega$) $f \longleftrightarrow$ *simple-function* M ($\lambda x.\ indicator\ \Omega\ x *_R f\ x$)
proof −
{ assume *finite* (f ‘ *space* (*restrict-space* $M\ \Omega$))
then have *finite* (f ‘ *space* (*restrict-space* $M\ \Omega$) $\cup\ \{0\}$) **by** *simp*
then have *finite* (($\lambda x.\ indicator\ \Omega\ x *_R f\ x$) ‘ *space* M)
by (*rule rev-finite-subset*) (*auto split*: *split-indicator simp*: *space-restrict-space*)
}
moreover
{ assume *finite* (($\lambda x.\ indicator\ \Omega\ x *_R f\ x$) ‘ *space* M)
then have *finite* (f ‘ *space* (*restrict-space* $M\ \Omega$))
by (*rule rev-finite-subset*) (*auto split*: *split-indicator simp*: *space-restrict-space*)
}
ultimately show *?thesis*
unfolding *simple-function-iff-borel-measurable*
borel-measurable-restrict-space-iff[*OF assms*]
by *auto*
qed

lemma *simple-integral-restrict-space*:
assumes Ω: $\Omega \cap$ *space* $M \in$ *sets* M *simple-function* (*restrict-space* M Ω) f
shows *simple-integral* (*restrict-space* M Ω) f = *simple-integral* M ($\lambda x.$ f x $*$ *indicator* Ω x)
using *simple-function-restrict-space-ennreal*[*THEN iffD1*, *OF* Ω, *THEN simple-functionD*(*1*)]
by (*auto simp add*: *space-restrict-space emeasure-restrict-space*[*OF* Ω(*1*)] *le-infI2 simple-integral-def*
split: *split-indicator split-indicator-asm*
intro!: *setsum.mono-neutral-cong-left ennreal-mult-left-cong arg-cong2*[**where** *f*=*emeasure*])

lemma *nn-integral-restrict-space*:
assumes Ω[*simp*]: $\Omega \cap$ *space* $M \in$ *sets* M
shows *nn-integral* (*restrict-space* M Ω) f = *nn-integral* M ($\lambda x.$ f x $*$ *indicator* Ω x)
proof −
let *?R* = *restrict-space* M Ω **and** *?X* = $\lambda M\, f.$ {$s.$ *simple-function* M s $\wedge$ $s \le f$ $\wedge$ ($\forall x.$ s x < *top*)}
have *integral*^{S} *?R* ‘ *?X ?R f* = *integral*^{S} M ‘ *?X* M ($\lambda x.$ f x $*$ *indicator* Ω x)
proof (*safe intro*!: *image-eqI*)
fix s **assume** s: *simple-function ?R* s $s \le f$ $\forall x.$ s x < *top*
from s **show** *integral*^{S} (*restrict-space* M Ω) s = *integral*^{S} M ($\lambda x.$ s x $*$ *indicator* Ω x)
by (*intro simple-integral-restrict-space*) *auto*
from s **show** *simple-function* M ($\lambda x.$ s x $*$ *indicator* Ω x)
by (*simp add*: *simple-function-restrict-space-ennreal*)
from s **show** ($\lambda x.$ s x $*$ *indicator* Ω x) $\le$ ($\lambda x.$ f x $*$ *indicator* Ω x)
$\bigwedge x.$ s x $*$ *indicator* Ω x < *top*
by (*auto split*: *split-indicator simp*: *le-fun-def image-subset-iff*)
next
fix s **assume** s: *simple-function* M s $s \le$ ($\lambda x.$ f x $*$ *indicator* Ω x) $\forall x.$ s x < *top*
then have *simple-function* M ($\lambda x.$ s x $*$ *indicator* ($\Omega \cap$ *space* M) x) (**is** *?s′*)
by (*intro simple-function-mult simple-function-indicator*) *auto*
also have *?s′* $\longleftrightarrow$ *simple-function* M ($\lambda x.$ s x $*$ *indicator* Ω x)
by (*rule simple-function-cong*) (*auto split*: *split-indicator*)
finally show *sf*: *simple-function* (*restrict-space* M Ω) s
by (*simp add*: *simple-function-restrict-space-ennreal*)

from s **have** *s-eq*: s = ($\lambda x.$ s x $*$ *indicator* Ω x)
by (*auto simp add*: *fun-eq-iff le-fun-def image-subset-iff*
split: *split-indicator split-indicator-asm*
intro: *antisym*)

show *integral*^{S} M s = *integral*^{S} (*restrict-space* M Ω) s
by (*subst s-eq*) (*rule simple-integral-restrict-space*[*symmetric*, *OF* Ω *sf*])
show $\bigwedge x.$ s x < *top*
using s **by** (*auto simp*: *image-subset-iff*)
from s **show** $s \le f$

by (*subst s-eq*) (*auto simp*: *image-subset-iff le-fun-def split*: *split-indicator split-indicator-asm*)
qed
then show *?thesis*
unfolding *nn-integral-def-finite* **by** (*simp cong del*: *strong-SUP-cong*)
qed

lemma *nn-integral-count-space-indicator*:
assumes *NO-MATCH* (*UNIV*::*'a set*) (*X*::*'a set*)
shows $(\int^+x.\ f\ x\ \partial count\text{-}space\ X) = (\int^+x.\ f\ x * indicator\ X\ x\ \partial count\text{-}space\ UNIV)$
by (*simp add*: *nn-integral-restrict-space*[*symmetric*] *restrict-count-space*)

lemma *nn-integral-count-space-eq*:
$(\bigwedge x.\ x \in A - B \Longrightarrow f\ x = 0) \Longrightarrow (\bigwedge x.\ x \in B - A \Longrightarrow f\ x = 0) \Longrightarrow$
$(\int^+x.\ f\ x\ \partial count\text{-}space\ A) = (\int^+x.\ f\ x\ \partial count\text{-}space\ B)$
by (*auto simp*: *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)

lemma *nn-integral-ge-point*:
assumes $x \in A$
shows $p\ x \leq \int^+ x.\ p\ x\ \partial count\text{-}space\ A$
proof −
from *assms* **have** $p\ x \leq \int^+ x.\ p\ x\ \partial count\text{-}space\ \{x\}$
by(*auto simp add*: *nn-integral-count-space-finite max-def*)
also have $\ldots = \int^+ x'.\ p\ x' * indicator\ \{x\}\ x'\ \partial count\text{-}space\ A$
using *assms* **by**(*auto simp add*: *nn-integral-count-space-indicator indicator-def intro*!: *nn-integral-cong*)
also have $\ldots \leq \int^+ x.\ p\ x\ \partial count\text{-}space\ A$
by(*rule nn-integral-mono*)(*simp add*: *indicator-def*)
finally show *?thesis* **.**
qed

5.4.4 Measure spaces with an associated density

definition *density* :: *'a measure* ⇒ (*'a* ⇒ *ennreal*) ⇒ *'a measure* **where**
$density\ M\ f = measure\text{-}of\ (space\ M)\ (sets\ M)\ (\lambda A.\ \int^+ x.\ f\ x * indicator\ A\ x\ \partial M)$

lemma
shows *sets-density*[*simp*, *measurable-cong*]: $sets\ (density\ M\ f) = sets\ M$
and *space-density*[*simp*]: $space\ (density\ M\ f) = space\ M$
by (*auto simp*: *density-def*)

lemma *space-density-imp*[*measurable-dest*]:
$\bigwedge x\ M\ f.\ x \in space\ (density\ M\ f) \Longrightarrow x \in space\ M$ **by** *auto*

lemma

shows *measurable-density-eq1*[*simp*]: $g \in$ *measurable* (*density Mg f*) *Mg'* $\longleftrightarrow g \in$ *measurable Mg Mg'*
and *measurable-density-eq2*[*simp*]: $h \in$ *measurable Mh* (*density Mh' f*) $\longleftrightarrow h \in$ *measurable Mh Mh'*
and *simple-function-density-eq*[*simp*]: *simple-function* (*density Mu f*) *u* $\longleftrightarrow$ *simple-function Mu u*
unfolding *measurable-def simple-function-def* **by** *simp-all*

lemma *density-cong*: $f \in$ *borel-measurable M* $\Longrightarrow f' \in$ *borel-measurable M* $\Longrightarrow$ (*AE x in M*. *f x* = *f' x*) $\Longrightarrow$ *density M f* = *density M f'*
unfolding *density-def* **by** (*auto intro*!: *measure-of-eq nn-integral-cong-AE sets.space-closed*)

lemma *emeasure-density*:
assumes *f*[*measurable*]: $f \in$ *borel-measurable M* **and** *A*[*measurable*]: $A \in$ *sets M*
shows *emeasure* (*density M f*) *A* = ($\int^+ x$. *f x* $*$ *indicator A x* ∂M)
(**is** - = *?μ A*)
unfolding *density-def*
proof (*rule emeasure-measure-of-sigma*)
show *sigma-algebra* (*space M*) (*sets M*) **..**
show *positive* (*sets M*) *?μ*
using *f* **by** (*auto simp*: *positive-def*)
show *countably-additive* (*sets M*) *?μ*
proof (*intro countably-additiveI*)
fix *A* :: *nat* $\Rightarrow$ *'a set* **assume** *range A* $\subseteq$ *sets M*
then have $\bigwedge i$. *A i* $\in$ *sets M* **by** *auto*
then have $*$: $\bigwedge i$. (λx. *f x* $*$ *indicator* (*A i*) *x*) $\in$ *borel-measurable M*
by *auto*
assume *disj*: *disjoint-family A*
then have ($\sum n$. *?μ* (*A n*)) = ($\int^+ x$. ($\sum n$. *f x* $*$ *indicator* (*A n*) *x*) ∂M)
using *f* $*$ **by** (*subst nn-integral-suminf*) *auto*
also have ($\int^+ x$. ($\sum n$. *f x* $*$ *indicator* (*A n*) *x*) ∂M) = ($\int^+ x$. *f x* $*$ ($\sum n$. *indicator* (*A n*) *x*) ∂M)
using *f* **by** (*auto intro*!: *ennreal-suminf-cmult nn-integral-cong-AE*)
also have ... = ($\int^+ x$. *f x* $*$ *indicator* ($\bigcup n$. *A n*) *x* ∂M)
unfolding *suminf-indicator*[*OF disj*] **..**
finally show ($\sum i$. $\int^+ x$. *f x* $*$ *indicator* (*A i*) *x* ∂M) = $\int^+ x$. *f x* $*$ *indicator* ($\bigcup i$. *A i*) *x* ∂M **.**
qed
qed *fact*

lemma *null-sets-density-iff*:
assumes *f*: $f \in$ *borel-measurable M*
shows $A \in$ *null-sets* (*density M f*) $\longleftrightarrow A \in$ *sets M* $\wedge$ (*AE x in M*. $x \in A \longrightarrow$ *f x* = *0*)
proof −
{ **assume** $A \in$ *sets M*
have ($\int^+ x$. *f x* $*$ *indicator A x* ∂M) = *0* $\longleftrightarrow$ *emeasure M* {$x \in$ *space M*. *f x* $*$ *indicator A x* $\neq$ *0*} = *0*
using *f* ‹*A* $\in$ *sets M*› **by** (*intro nn-integral-0-iff*) *auto*

also have ... $\longleftrightarrow$ (*AE x in M. f x * indicator A x = 0*)
using *f* ‹*A ∈ sets M*› **by** (*intro AE-iff-measurable*[*OF - refl, symmetric*]) *auto*
also have (*AE x in M. f x * indicator A x = 0*) $\longleftrightarrow$ (*AE x in M. x ∈ A* $\longrightarrow$ *f x ≤ 0*)
by (*auto simp add*: *indicator-def max-def split*: *if-split-asm*)
finally have ($\int^+ x.$ *f x * indicator A x ∂M*) *= 0* $\longleftrightarrow$ (*AE x in M. x ∈ A* $\longrightarrow$ *f x ≤ 0*) **. }**
with *f* **show** *?thesis*
by (*simp add*: *null-sets-def emeasure-density cong*: *conj-cong*)
qed

lemma *AE-density*:
assumes *f*: *f ∈ borel-measurable M*
shows (*AE x in density M f. P x*) $\longleftrightarrow$ (*AE x in M. 0 < f x* $\longrightarrow$ *P x*)
proof
assume *AE x in density M f. P x*
with *f* **obtain** *N* **where** {*x ∈ space M.* ¬ *P x*} ⊆ *N N ∈ sets M* **and** *ae*: *AE x in M. x ∈ N* $\longrightarrow$ *f x = 0*
by (*auto simp*: *eventually-ae-filter null-sets-density-iff*)
then have *AE x in M. x ∉ N* $\longrightarrow$ *P x* **by** *auto*
with *ae* **show** *AE x in M. 0 < f x* $\longrightarrow$ *P x*
by (*rule eventually-elim2*) *auto*
next
fix *N* **assume** *ae*: *AE x in M. 0 < f x* $\longrightarrow$ *P x*
then obtain *N* **where** {*x ∈ space M.* ¬ (*0 < f x* $\longrightarrow$ *P x*)} ⊆ *N N ∈ null-sets M*
by (*auto simp*: *eventually-ae-filter*)
then have ∗: {*x ∈ space* (*density M f*). ¬ *P x*} ⊆ *N* ∪ {*x*∈*space M. f x = 0*} *N* ∪ {*x*∈*space M. f x = 0*} *∈ sets M* **and** *ae2*: *AE x in M. x ∉ N*
using *f* **by** (*auto simp*: *subset-eq zero-less-iff-neq-zero intro*!: *AE-not-in*)
show *AE x in density M f. P x*
using *ae2*
unfolding *eventually-ae-filter*[*of - density M f*] *Bex-def null-sets-density-iff*[*OF f*]
by (*intro exI*[*of - N* ∪ {*x*∈*space M. f x = 0*}] *conjI* ∗) (*auto elim*: *eventually-elim2*)
qed

lemma *nn-integral-density*:
assumes *f*: *f ∈ borel-measurable M*
assumes *g*: *g ∈ borel-measurable M*
shows *integral*^{N} (*density M f*) *g* = ($\int^+ x.$ *f x * g x ∂M*)
using *g* **proof** *induct*
case (*cong u v*)
then show *?case*
apply (*subst nn-integral-cong*[*OF cong*(*3*)])
apply (*simp-all cong*: *nn-integral-cong*)
done
next
case (*set A*) **then show** *?case*

by (*simp add*: *emeasure-density f*)
next
case (*mult u c*)
moreover have $\bigwedge x.\ f\ x * (c * u\ x) = c * (f\ x * u\ x)$ **by** (*simp add*: *field-simps*)
ultimately show *?case*
using *f* **by** (*simp add*: *nn-integral-cmult*)
next
case (*add u v*)
then have $\bigwedge x.\ f\ x * (v\ x + u\ x) = f\ x * v\ x + f\ x * u\ x$
by (*simp add*: *distrib-left*)
with *add f* **show** *?case*
by (*auto simp add*: *nn-integral-add intro*!: *nn-integral-add*[*symmetric*])
next
case (*seq U*)
have *eq*: *AE x in M*. $f\ x * (SUP\ i.\ U\ i\ x) = (SUP\ i.\ f\ x * U\ i\ x)$
by *eventually-elim* (*simp add*: *SUP-mult-left-ennreal seq*)
from *seq f* **show** *?case*
apply (*simp add*: *nn-integral-monotone-convergence-SUP*)
apply (*subst nn-integral-cong-AE*[*OF eq*])
apply (*subst nn-integral-monotone-convergence-SUP-AE*)
apply (*auto simp*: *incseq-def le-fun-def intro*!: *mult-left-mono*)
done
qed

lemma *density-distr*:
assumes [*measurable*]: $f \in$ *borel-measurable N X* $\in$ *measurable M N*
shows *density* (*distr M N X*) *f* = *distr* (*density M* ($\lambda x.\ f\ (X\ x)$)) *N X*
by (*intro measure-eqI*)
(*auto simp add*: *emeasure-density nn-integral-distr emeasure-distr*
split: *split-indicator intro*!: *nn-integral-cong*)

lemma *emeasure-restricted*:
assumes *S*: $S \in$ *sets M* **and** *X*: $X \in$ *sets M*
shows *emeasure* (*density M* (*indicator S*)) *X* = *emeasure M* ($S \cap X$)
proof −
have *emeasure* (*density M* (*indicator S*)) *X* = ($\int^+ x.$ *indicator S x* * *indicator X x* ∂M)
using *S X* **by** (*simp add*: *emeasure-density*)
also have ... = ($\int^+ x.$ *indicator* ($S \cap X$) *x* ∂M)
by (*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
also have ... = *emeasure M* ($S \cap X$)
using *S X* **by** (*simp add*: *sets.Int*)
finally show *?thesis* **.**
qed

lemma *measure-restricted*:
$S \in$ *sets M* $\Longrightarrow$ $X \in$ *sets M* $\Longrightarrow$ *measure* (*density M* (*indicator S*)) *X* = *measure M* ($S \cap X$)
by (*simp add*: *emeasure-restricted measure-def*)

lemma (**in** *finite-measure*) *finite-measure-restricted*:
S ∈ *sets M* ⟹ *finite-measure* (*density M* (*indicator S*))
by *standard* (*simp add*: *emeasure-restricted*)

lemma *emeasure-density-const*:
A ∈ *sets M* ⟹ *emeasure* (*density M* (λ-. *c*)) *A* = *c* ∗ *emeasure M A*
by (*auto simp*: *nn-integral-cmult-indicator emeasure-density*)

lemma *measure-density-const*:
A ∈ *sets M* ⟹ *c* ≠ ∞ ⟹ *measure* (*density M* (λ-. *c*)) *A* = *enn2real c* ∗ *measure M A*
by (*auto simp*: *emeasure-density-const measure-def enn2real-mult*)

lemma *density-density-eq*:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹
density (*density M f*) *g* = *density M* (λ*x*. *f x* ∗ *g x*)
by (*auto intro*!: *measure-eqI simp*: *emeasure-density nn-integral-density ac-simps*)

lemma *distr-density-distr*:
assumes *T*: *T* ∈ *measurable M M'* **and** *T'*: *T'* ∈ *measurable M' M*
and *inv*: ∀ *x*∈*space M*. *T'* (*T x*) = *x*
assumes *f*: *f* ∈ *borel-measurable M'*
shows *distr* (*density* (*distr M M' T*) *f*) *M T'* = *density M* (*f* ∘ *T*) (**is** *?R* = *?L*)
proof (*rule measure-eqI*)
fix *A* **assume** *A*: *A* ∈ *sets ?R*
{ fix *x* **assume** *x* ∈ *space M*
with *sets.sets-into-space*[*OF A*]
have *indicator* (*T'* −‘ *A* ∩ *space M'*) (*T x*) = (*indicator A x* :: *ennreal*)
using *T inv* **by** (*auto simp*: *indicator-def measurable-space*) **}**
with *A T T' f* **show** *emeasure ?R A* = *emeasure ?L A*
by (*simp add*: *measurable-comp emeasure-density emeasure-distr*
nn-integral-distr measurable-sets cong: *nn-integral-cong*)
qed *simp*

lemma *density-density-divide*:
fixes *f g* :: *'a* ⇒ *real*
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 *density* (*density M f*) (λ*x*. *g x* / *f x*) = *density M g*
proof −
have *density M g* = *density M* (λ*x*. *ennreal* (*f x*) ∗ *ennreal* (*g x* / *f x*))
using *f g ac* **by** (*auto intro*!: *density-cong measurable-If simp*: *ennreal-mult*[*symmetric*])
then show *?thesis*
using *f g* **by** (*subst density-density-eq*) *auto*
qed

lemma *density-1*: *density M* (λ-. *1*) = *M*
by (*intro measure-eqI*) (*auto simp*: *emeasure-density*)

lemma *emeasure-density-add*:
assumes *X*: *X* ∈ *sets M*
assumes *Mf*[*measurable*]: *f* ∈ *borel-measurable M*
assumes *Mg*[*measurable*]: *g* ∈ *borel-measurable M*
shows *emeasure* (*density M f*) *X* + *emeasure* (*density M g*) *X* =
emeasure (*density M* (λx. *f x* + *g x*)) *X*
using *assms*
apply (*subst* (*1 2 3*) *emeasure-density*, *simp-all*) []
apply (*subst nn-integral-add*[*symmetric*], *simp-all*) []
apply (*intro nn-integral-cong*, *simp split*: *split-indicator*)
done

5.4.5 Point measure

definition *point-measure* :: *'a set* ⇒ (*'a* ⇒ *ennreal*) ⇒ *'a measure* **where**
point-measure A f = *density* (*count-space A*) *f*

lemma
shows *space-point-measure*: *space* (*point-measure A f*) = *A*
and *sets-point-measure*: *sets* (*point-measure A f*) = *Pow A*
by (*auto simp*: *point-measure-def*)

lemma *sets-point-measure-count-space*[*measurable-cong*]: *sets* (*point-measure A f*)
= *sets* (*count-space A*)
by (*simp add*: *sets-point-measure*)

lemma *measurable-point-measure-eq1*[*simp*]:
g ∈ *measurable* (*point-measure A f*) *M* ⟷ *g* ∈ *A* → *space M*
unfolding *point-measure-def* **by** *simp*

lemma *measurable-point-measure-eq2-finite*[*simp*]:
finite A ⟹
g ∈ *measurable M* (*point-measure A f*) ⟷
(*g* ∈ *space M* → *A* ∧ (∀ *a*∈*A*. *g* −‘ {*a*} ∩ *space M* ∈ *sets M*))
unfolding *point-measure-def* **by** (*simp add*: *measurable-count-space-eq2*)

lemma *simple-function-point-measure*[*simp*]:
simple-function (*point-measure A f*) *g* ⟷ *finite* (*g* ‘ *A*)
by (*simp add*: *point-measure-def*)

lemma *emeasure-point-measure*:
assumes *A*: *finite* {*a*∈*X*. *0* < *f a*} *X* ⊆ *A*
shows *emeasure* (*point-measure A f*) *X* = (∑ *a*|*a*∈*X* ∧ *0* < *f a*. *f a*)
proof −
have {*a*. (*a* ∈ *X* ⟶ *a* ∈ *A* ∧ *0* < *f a*) ∧ *a* ∈ *X*} = {*a*∈*X*. *0* < *f a*}
using ‹*X* ⊆ *A*› **by** *auto*

with *A* **show** *?thesis*
by (*simp add*: *emeasure-density nn-integral-count-space point-measure-def indicator-def*)
qed

lemma *emeasure-point-measure-finite*:
finite A $\Longrightarrow$ *X* $\subseteq$ *A* $\Longrightarrow$ *emeasure* (*point-measure A f*) *X* = ($\sum$ *a*∈*X*. *f a*)
by (*subst emeasure-point-measure*) (*auto dest*: *finite-subset intro*!: *setsum.mono-neutral-left simp*: *less-le*)

lemma *emeasure-point-measure-finite2*:
X $\subseteq$ *A* $\Longrightarrow$ *finite X* $\Longrightarrow$ *emeasure* (*point-measure A f*) *X* = ($\sum$ *a*∈*X*. *f a*)
by (*subst emeasure-point-measure*)
(*auto dest*: *finite-subset intro*!: *setsum.mono-neutral-left simp*: *less-le*)

lemma *null-sets-point-measure-iff*:
X ∈ *null-sets* (*point-measure A f*) $\longleftrightarrow$ *X* $\subseteq$ *A* ∧ (∀ *x*∈*X*. *f x* = *0*)
by (*auto simp*: *AE-count-space null-sets-density-iff point-measure-def*)

lemma *AE-point-measure*:
(*AE x in point-measure A f*. *P x*) $\longleftrightarrow$ (∀ *x*∈*A*. *0* < *f x* $\longrightarrow$ *P x*)
unfolding *point-measure-def*
by (*subst AE-density*) (*auto simp*: *AE-density AE-count-space point-measure-def*)

lemma *nn-integral-point-measure*:
finite {*a*∈*A*. *0* < *f a* ∧ *0* < *g a*} $\Longrightarrow$
integral^N (*point-measure A f*) *g* = ($\sum$ *a*|*a*∈*A* ∧ *0* < *f a* ∧ *0* < *g a*. *f a* ∗ *g a*)
unfolding *point-measure-def*
by (*subst nn-integral-density*)
(*simp-all add*: *nn-integral-density nn-integral-count-space ennreal-zero-less-mult-iff*)

lemma *nn-integral-point-measure-finite*:
finite A $\Longrightarrow$ *integral*^{N} (*point-measure A f*) *g* = ($\sum$ *a*∈*A*. *f a* ∗ *g a*)
by (*subst nn-integral-point-measure*) (*auto intro*!: *setsum.mono-neutral-left simp*: *less-le*)

5.4.6 Uniform measure

definition *uniform-measure M A* = *density M* (λ*x*. *indicator A x* / *emeasure M A*)

lemma
shows *sets-uniform-measure*[*simp*, *measurable-cong*]: *sets* (*uniform-measure M A*) = *sets M*
and *space-uniform-measure*[*simp*]: *space* (*uniform-measure M A*) = *space M*
by (*auto simp*: *uniform-measure-def*)

lemma *emeasure-uniform-measure*[*simp*]:
assumes *A*: *A* ∈ *sets M* **and** *B*: *B* ∈ *sets M*
shows *emeasure* (*uniform-measure M A*) *B* = *emeasure M* (*A* ∩ *B*) / *emeasure*

M A
proof −
from *A B* **have** *emeasure* (*uniform-measure M A*) *B* = (∫⁺*x*. (*1* / *emeasure M A*) ∗ *indicator* (*A* ∩ *B*) *x* ∂*M*)
by (*auto simp add*: *uniform-measure-def emeasure-density divide-ennreal-def split*: *split-indicator intro*!: *nn-integral-cong*)
also have ... = *emeasure M* (*A* ∩ *B*) / *emeasure M A*
using *A B*
by (*subst nn-integral-cmult-indicator*) (*simp-all add*: *sets.Int divide-ennreal-def mult.commute*)
finally show *?thesis* **.**
qed

lemma *measure-uniform-measure*[*simp*]:
assumes *A*: *emeasure M A* ≠ *0 emeasure M A* ≠ ∞ **and** *B*: *B* ∈ *sets M*
shows *measure* (*uniform-measure M A*) *B* = *measure M* (*A* ∩ *B*) / *measure M A*
using *emeasure-uniform-measure*[*OF emeasure-neq-0-sets*[*OF A*(*1*)] *B*] *A*
by (*cases emeasure M A emeasure M* (*A* ∩ *B*) *rule*: *ennreal2-cases*) (*simp-all add*: *measure-def divide-ennreal top-ennreal.rep-eq top-ereal-def ennreal-top-divide*)

lemma *AE-uniform-measureI*:
A ∈ *sets M* ⟹ (*AE x in M*. *x* ∈ *A* ⟶ *P x*) ⟹ (*AE x in uniform-measure M A*. *P x*)
unfolding *uniform-measure-def* **by** (*auto simp*: *AE-density divide-ennreal-def*)

lemma *emeasure-uniform-measure-1*:
emeasure M S ≠ *0* ⟹ *emeasure M S* ≠ ∞ ⟹ *emeasure* (*uniform-measure M S*) *S* = *1*
by (*subst emeasure-uniform-measure*) (*simp-all add*: *emeasure-neq-0-sets emeasure-eq-ennreal-measure divide-ennreal zero-less-iff-neq-zero*[*symmetric*])

lemma *nn-integral-uniform-measure*:
assumes *f*[*measurable*]: *f* ∈ *borel-measurable M* **and** *S*[*measurable*]: *S* ∈ *sets M*
shows (∫⁺*x*. *f x* ∂*uniform-measure M S*) = (∫⁺*x*. *f x* ∗ *indicator S x* ∂*M*) / *emeasure M S*
proof −
{ assume *emeasure M S* = ∞
then have *?thesis*
by (*simp add*: *uniform-measure-def nn-integral-density f*) **}**
moreover
{ assume [*simp*]: *emeasure M S* = *0*
then have *ae*: *AE x in M*. *x* ∉ *S*
using *sets.sets-into-space*[*OF S*]
by (*subst AE-iff-measurable*[*OF - refl*]) (*simp-all add*: *subset-eq cong*: *rev-conj-cong*)
from *ae* **have** (∫⁺ *x*. *indicator S x* / *0* ∗ *f x* ∂*M*) = *0*
by (*subst nn-integral-0-iff-AE*) *auto*

```
    moreover from ae have (∫⁺ x. f x * indicator S x ∂M) = 0
      by (subst nn-integral-0-iff-AE) auto
    ultimately have ?thesis
      by (simp add: uniform-measure-def nn-integral-density f) }
  moreover have emeasure M S ≠ 0 ⟹ emeasure M S ≠ ∞ ⟹ ?thesis
    unfolding uniform-measure-def
    by (subst nn-integral-density)
     (auto simp: ennreal-times-divide f nn-integral-divide[symmetric] mult.commute)
  ultimately show ?thesis by blast
qed

lemma AE-uniform-measure:
  assumes emeasure M A ≠ 0 emeasure M A < ∞
  shows (AE x in uniform-measure M A. P x) ⟷ (AE x in M. x ∈ A ⟶ P x)
proof −
  have A ∈ sets M
    using ‹emeasure M A ≠ 0› by (metis emeasure-notin-sets)
  moreover have ⋀x. 0 < indicator A x / emeasure M A ⟷ x ∈ A
    using assms
   by (cases emeasure M A) (auto split: split-indicator simp: ennreal-zero-less-divide)
  ultimately show ?thesis
    unfolding uniform-measure-def by (simp add: AE-density)
qed
```

5.4.7 Null measure

```
lemma null-measure-eq-density: null-measure M = density M (λ-. 0)
  by (intro measure-eqI) (simp-all add: emeasure-density)

lemma nn-integral-null-measure[simp]: (∫⁺x. f x ∂null-measure M) = 0
  by (auto simp add: nn-integral-def simple-integral-def SUP-constant bot-ennreal-def
le-fun-def
          intro!: exI[of - λx. 0])

lemma density-null-measure[simp]: density (null-measure M) f = null-measure M
proof (intro measure-eqI)
  fix A show emeasure (density (null-measure M) f) A = emeasure (null-measure
M) A
    by (simp add: density-def) (simp only: null-measure-def[symmetric] emeasure-null-measure)
qed simp
```

5.4.8 Uniform count measure

```
definition uniform-count-measure A = point-measure A (λx. 1 / card A)

lemma
  shows space-uniform-count-measure: space (uniform-count-measure A) = A
    and sets-uniform-count-measure: sets (uniform-count-measure A) = Pow A
      unfolding uniform-count-measure-def by (auto simp: space-point-measure
sets-point-measure)
```

lemma *sets-uniform-count-measure-count-space*[*measurable-cong*]:
sets (*uniform-count-measure A*) = *sets* (*count-space A*)
by (*simp add*: *sets-uniform-count-measure*)

lemma *emeasure-uniform-count-measure*:
finite A ⟹ *X* ⊆ *A* ⟹ *emeasure* (*uniform-count-measure A*) *X* = *card X* / *card A*
by (*simp add*: *emeasure-point-measure-finite uniform-count-measure-def divide-inverse ennreal-mult*
ennreal-of-nat-eq-real-of-nat)

lemma *measure-uniform-count-measure*:
finite A ⟹ *X* ⊆ *A* ⟹ *measure* (*uniform-count-measure A*) *X* = *card X* / *card A*
by (*simp add*: *emeasure-point-measure-finite uniform-count-measure-def measure-def enn2real-mult*)

lemma *space-uniform-count-measure-empty-iff* [*simp*]:
space (*uniform-count-measure X*) = {} ⟷ *X* = {}
by(*simp add*: *space-uniform-count-measure*)

lemma *sets-uniform-count-measure-eq-UNIV* [*simp*]:
sets (*uniform-count-measure UNIV*) = *UNIV* ⟷ *True*
UNIV = *sets* (*uniform-count-measure UNIV*) ⟷ *True*
by(*simp-all add*: *sets-uniform-count-measure*)

5.4.9 Scaled measure

lemma *nn-integral-scale-measure*:
assumes *f*: *f* ∈ *borel-measurable M*
shows *nn-integral* (*scale-measure r M*) *f* = *r* ∗ *nn-integral M f*
using *f*
proof *induction*
case (*cong f g*)
thus *?case*
by(*simp add*: *cong.hyps space-scale-measure cong*: *nn-integral-cong-simp*)
next
case (*mult f c*)
thus *?case*
by(*simp add*: *nn-integral-cmult max-def mult.assoc mult.left-commute*)
next
case (*add f g*)
thus *?case*
by(*simp add*: *nn-integral-add distrib-left*)
next
case (*seq U*)
thus *?case*
by(*simp add*: *nn-integral-monotone-convergence-SUP SUP-mult-left-ennreal*)

qed *simp*

end

6 Binary product measures

theory *Binary-Product-Measure*
imports *Nonnegative-Lebesgue-Integration*
begin

lemma *Pair-vimage-times*[*simp*]: *Pair x −‘ (A × B) = (if x ∈ A then B else {})*
by *auto*

lemma *rev-Pair-vimage-times*[*simp*]: *(λx. (x, y)) −‘ (A × B) = (if y ∈ B then A else {})*
by *auto*

6.1 Binary products

definition *pair-measure* (**infixr** $\bigotimes_M$ *80*) **where**
A $\bigotimes_M$ *B = measure-of (space A × space B)*
{a × b | a b. a ∈ sets A ∧ b ∈ sets B}
(λX. ∫⁺x. (∫⁺y. indicator X (x,y) ∂B) ∂A)

lemma *pair-measure-closed*: *{a × b | a b. a ∈ sets A ∧ b ∈ sets B} ⊆ Pow (space A × space B)*
using *sets.space-closed*[*of A*] *sets.space-closed*[*of B*] **by** *auto*

lemma *space-pair-measure*:
space (A $\bigotimes_M$ *B) = space A × space B*
unfolding *pair-measure-def* **using** *pair-measure-closed*[*of A B*]
by (*rule space-measure-of*)

lemma *SIGMA-Collect-eq*: *(SIGMA x:space M. {y∈space N. P x y}) = {x∈space (M* $\bigotimes_M$ *N). P (fst x) (snd x)}*
by (*auto simp*: *space-pair-measure*)

lemma *sets-pair-measure*:
sets (A $\bigotimes_M$ *B) = sigma-sets (space A × space B) {a × b | a b. a ∈ sets A ∧ b ∈ sets B}*
unfolding *pair-measure-def* **using** *pair-measure-closed*[*of A B*]
by (*rule sets-measure-of*)

lemma *sets-pair-in-sets*:
assumes *N*: *space A × space B = space N*
assumes *⋀a b. a ∈ sets A ⟹ b ∈ sets B ⟹ a × b ∈ sets N*
shows *sets (A* $\bigotimes_M$ *B) ⊆ sets N*
using *assms* **by** (*auto intro*!: *sets.sigma-sets-subset simp*: *sets-pair-measure N*)

lemma *sets-pair-measure-cong*[*measurable-cong*, *cong*]:
sets M1 = sets M1' ⟹ sets M2 = sets M2' ⟹ sets (M1 ⨂M M2) = sets (M1' ⨂M M2')
unfolding *sets-pair-measure* **by** (*simp cong*: *sets-eq-imp-space-eq*)

lemma *pair-measureI*[*intro*, *simp*, *measurable*]:
x ∈ sets A ⟹ y ∈ sets B ⟹ x × y ∈ sets (A ⨂M B)
by (*auto simp*: *sets-pair-measure*)

lemma *sets-Pair*: *{x} ∈ sets M1 ⟹ {y} ∈ sets M2 ⟹ {(x, y)} ∈ sets (M1 ⨂M M2)*
using *pair-measureI*[*of {x} M1 {y} M2*] **by** *simp*

lemma *measurable-pair-measureI*:
assumes *1*: *f ∈ space M → space M1 × space M2*
assumes *2*: *⋀A B. A ∈ sets M1 ⟹ B ∈ sets M2 ⟹ f −‘ (A × B) ∩ space M ∈ sets M*
shows *f ∈ measurable M (M1 ⨂M M2)*
unfolding *pair-measure-def* **using** *1 2*
by (*intro measurable-measure-of*) (*auto dest*: *sets.sets-into-space*)

lemma *measurable-split-replace*[*measurable (raw)*]:
(λx. f x (fst (g x)) (snd (g x))) ∈ measurable M N ⟹ (λx. case-prod (f x) (g x)) ∈ measurable M N
unfolding *split-beta'* **.**

lemma *measurable-Pair*[*measurable (raw)*]:
assumes *f*: *f ∈ measurable M M1* **and** *g*: *g ∈ measurable M M2*
shows *(λx. (f x, g x)) ∈ measurable M (M1 ⨂M M2)*
proof (*rule measurable-pair-measureI*)
show *(λx. (f x, g x)) ∈ space M → space M1 × space M2*
using *f g* **by** (*auto simp*: *measurable-def*)
fix *A B* **assume** *∗*: *A ∈ sets M1 B ∈ sets M2*
have *(λx. (f x, g x)) −‘ (A × B) ∩ space M = (f −‘ A ∩ space M) ∩ (g −‘ B ∩ space M)*
by *auto*
also have *... ∈ sets M*
by (*rule sets.Int*) (*auto intro*!: *measurable-sets ∗ f g*)
finally show *(λx. (f x, g x)) −‘ (A × B) ∩ space M ∈ sets M* **.**
qed

lemma *measurable-fst*[*intro*!, *simp*, *measurable*]: *fst ∈ measurable (M1 ⨂M M2) M1*
by (*auto simp*: *fst-vimage-eq-Times space-pair-measure sets.sets-into-space times-Int-times measurable-def*)

lemma *measurable-snd*[*intro*!, *simp*, *measurable*]: *snd ∈ measurable (M1 ⨂M M2) M2*
by (*auto simp*: *snd-vimage-eq-Times space-pair-measure sets.sets-into-space times-Int-times*

measurable-def)

lemma *measurable-Pair-compose-split*[*measurable-dest*]:
assumes *f*: *case-prod f* ∈ *measurable* (*M1* ⨂$_M$ *M2*) *N*
assumes *g*: *g* ∈ *measurable M M1* **and** *h*: *h* ∈ *measurable M M2*
shows (λ*x*. *f* (*g x*) (*h x*)) ∈ *measurable M N*
using *measurable-compose*[*OF measurable-Pair f*, *OF g h*] **by** *simp*

lemma *measurable-Pair1-compose*[*measurable-dest*]:
assumes *f*: (λ*x*. (*f x*, *g x*)) ∈ *measurable M* (*M1* ⨂$_M$ *M2*)
assumes [*measurable*]: *h* ∈ *measurable N M*
shows (λ*x*. *f* (*h x*)) ∈ *measurable N M1*
using *measurable-compose*[*OF f measurable-fst*] **by** *simp*

lemma *measurable-Pair2-compose*[*measurable-dest*]:
assumes *f*: (λ*x*. (*f x*, *g x*)) ∈ *measurable M* (*M1* ⨂$_M$ *M2*)
assumes [*measurable*]: *h* ∈ *measurable N M*
shows (λ*x*. *g* (*h x*)) ∈ *measurable N M2*
using *measurable-compose*[*OF f measurable-snd*] **by** *simp*

lemma *measurable-pair*:
assumes (*fst* ∘ *f*) ∈ *measurable M M1* (*snd* ∘ *f*) ∈ *measurable M M2*
shows *f* ∈ *measurable M* (*M1* ⨂$_M$ *M2*)
using *measurable-Pair*[*OF assms*] **by** *simp*

lemma
assumes *f*[*measurable*]: *f* ∈ *measurable M* (*N* ⨂$_M$ *P*)
shows *measurable-fst'*: (λ*x*. *fst* (*f x*)) ∈ *measurable M N*
and *measurable-snd'*: (λ*x*. *snd* (*f x*)) ∈ *measurable M P*
by *simp-all*

lemma
assumes *f*[*measurable*]: *f* ∈ *measurable M N*
shows *measurable-fst''*: (λ*x*. *f* (*fst x*)) ∈ *measurable* (*M* ⨂$_M$ *P*) *N*
and *measurable-snd''*: (λ*x*. *f* (*snd x*)) ∈ *measurable* (*P* ⨂$_M$ *M*) *N*
by *simp-all*

lemma *sets-pair-eq-sets-fst-snd*:
sets (*A* ⨂$_M$ *B*) = *sets* (*Sup-sigma* {*vimage-algebra* (*space A* × *space B*) *fst A*, *vimage-algebra* (*space A* × *space B*) *snd B*})
(**is** *?P* = *sets* (*Sup-sigma* {*?fst*, *?snd*}))
proof −
{ fix *a b* **assume** *ab*: *a* ∈ *sets A b* ∈ *sets B*
then have *a* × *b* = (*fst* −‘ *a* ∩ (*space A* × *space B*)) ∩ (*snd* −‘ *b* ∩ (*space A* × *space B*))
by (*auto dest*: *sets.sets-into-space*)
also have ... ∈ *sets* (*Sup-sigma* {*?fst*, *?snd*})
using *ab* **by** (*auto intro*: *in-Sup-sigma in-vimage-algebra*)
finally have *a* × *b* ∈ *sets* (*Sup-sigma* {*?fst*, *?snd*}) **. }**

moreover have *sets ?fst* $\subseteq$ *sets* ($A \bigotimes_M B$)
by (*rule sets-image-in-sets*) (*auto simp*: *space-pair-measure*[*symmetric*])
moreover have *sets ?snd* $\subseteq$ *sets* ($A \bigotimes_M B$)
by (*rule sets-image-in-sets*) (*auto simp*: *space-pair-measure*)
ultimately show *?thesis*
by (*intro antisym*[*of sets A* **for** *A*] *sets-Sup-in-sets sets-pair-in-sets*)
(*auto simp add*: *space-Sup-sigma space-pair-measure*)
qed

lemma *measurable-pair-iff*:
$f \in$ *measurable M* ($M1 \bigotimes_M M2$) $\longleftrightarrow$ ($fst \circ f$) $\in$ *measurable M M1* $\wedge$ ($snd \circ f$) $\in$ *measurable M M2*
by (*auto intro*: *measurable-pair*[*of f M M1 M2*])

lemma *measurable-split-conv*:
($\lambda(x, y).\ f\ x\ y$) $\in$ *measurable A B* $\longleftrightarrow$ ($\lambda x.\ f\ (fst\ x)\ (snd\ x)$) $\in$ *measurable A B*
by (*intro arg-cong2*[**where** *f*=*op* $\in$]) *auto*

lemma *measurable-pair-swap'*: ($\lambda(x,y).\ (y, x)$) $\in$ *measurable* ($M1 \bigotimes_M M2$) ($M2 \bigotimes_M M1$)
by (*auto intro*!: *measurable-Pair simp*: *measurable-split-conv*)

lemma *measurable-pair-swap*:
assumes *f*: $f \in$ *measurable* ($M1 \bigotimes_M M2$) *M* **shows** ($\lambda(x,y).\ f\ (y, x)$) $\in$ *measurable* ($M2 \bigotimes_M M1$) *M*
using *measurable-comp*[*OF measurable-Pair f*] **by** (*auto simp*: *measurable-split-conv comp-def*)

lemma *measurable-pair-swap-iff*:
$f \in$ *measurable* ($M2 \bigotimes_M M1$) *M* $\longleftrightarrow$ ($\lambda(x,y).\ f\ (y,x)$) $\in$ *measurable* ($M1 \bigotimes_M M2$) *M*
by (*auto dest*: *measurable-pair-swap*)

lemma *measurable-Pair1'*: $x \in$ *space M1* $\Longrightarrow$ *Pair x* $\in$ *measurable M2* ($M1 \bigotimes_M M2$)
by *simp*

lemma *sets-Pair1*[*measurable* (*raw*)]:
assumes *A*: $A \in$ *sets* ($M1 \bigotimes_M M2$) **shows** *Pair x* $-`$ *A* $\in$ *sets M2*
proof $-$
have *Pair x* $-`$ *A* = (*if x* $\in$ *space M1 then Pair x* $-`$ *A* $\cap$ *space M2 else* $\{\}$)
using *A*[*THEN sets.sets-into-space*] **by** (*auto simp*: *space-pair-measure*)
also have ... $\in$ *sets M2*
using *A* **by** (*auto simp add*: *measurable-Pair1' intro*!: *measurable-sets split*: *if-split-asm*)
finally show *?thesis* **.**
qed

lemma *measurable-Pair2'*: $y \in$ *space M2* $\Longrightarrow$ ($\lambda x.\ (x, y)$) $\in$ *measurable M1* (*M1*

$\bigotimes_M$ *M2*)
by (*auto intro*!: *measurable-Pair*)

lemma *sets-Pair2*: **assumes** *A*: *A* ∈ *sets* (*M1* $\bigotimes_M$ *M2*) **shows** (λ*x*. (*x*, *y*)) −‘ *A* ∈ *sets M1*
proof −
have (λ*x*. (*x*, *y*)) −‘ *A* = (*if y* ∈ *space M2 then* (λ*x*. (*x*, *y*)) −‘ *A* ∩ *space M1 else* {})
using *A*[*THEN sets.sets-into-space*] **by** (*auto simp*: *space-pair-measure*)
also have ... ∈ *sets M1*
using *A* **by** (*auto simp add*: *measurable-Pair2′ intro*!: *measurable-sets split*: *if-split-asm*)
finally show *?thesis* **.**
qed

lemma *measurable-Pair2*:
assumes *f*: *f* ∈ *measurable* (*M1* $\bigotimes_M$ *M2*) *M* **and** *x*: *x* ∈ *space M1*
shows (λ*y*. *f* (*x*, *y*)) ∈ *measurable M2 M*
using *measurable-comp*[*OF measurable-Pair1′ f*, *OF x*]
by (*simp add*: *comp-def*)

lemma *measurable-Pair1*:
assumes *f*: *f* ∈ *measurable* (*M1* $\bigotimes_M$ *M2*) *M* **and** *y*: *y* ∈ *space M2*
shows (λ*x*. *f* (*x*, *y*)) ∈ *measurable M1 M*
using *measurable-comp*[*OF measurable-Pair2′ f*, *OF y*]
by (*simp add*: *comp-def*)

lemma *Int-stable-pair-measure-generator*: *Int-stable* {*a* × *b* | *a b*. *a* ∈ *sets A* ∧ *b* ∈ *sets B*}
unfolding *Int-stable-def*
by *safe* (*auto simp add*: *times-Int-times*)

lemma (**in** *finite-measure*) *finite-measure-cut-measurable*:
assumes [*measurable*]: *Q* ∈ *sets* (*N* $\bigotimes_M$ *M*)
shows (λ*x*. *emeasure M* (*Pair x* −‘ *Q*)) ∈ *borel-measurable N*
(**is** *?s Q* ∈ -)
using *Int-stable-pair-measure-generator pair-measure-closed assms*
unfolding *sets-pair-measure*
proof (*induct rule*: *sigma-sets-induct-disjoint*)
case (*compl A*)
with *sets.sets-into-space* **have** ⋀*x*. *emeasure M* (*Pair x* −‘ ((*space N* × *space M*) − *A*)) =
(*if x* ∈ *space N then emeasure M* (*space M*) − *?s A x else 0*)
unfolding *sets-pair-measure*[*symmetric*]
by (*auto intro*!: *emeasure-compl simp*: *vimage-Diff sets-Pair1*)
with *compl sets.top* **show** *?case*
by (*auto intro*!: *measurable-If simp*: *space-pair-measure*)
next
case (*union F*)

then have $\bigwedge x.$ *emeasure* M (*Pair* x $-`$ $(\bigcup i.$ F $i)) = (\sum i.$ *?s* $(F$ $i)$ $x)$
by (*simp add*: *suminf-emeasure disjoint-family-on-vimageI subset-eq vimage-UN sets-pair-measure*[*symmetric*])
with *union* **show** *?case*
unfolding *sets-pair-measure*[*symmetric*] **by** *simp*
qed (*auto simp add*: *if-distrib Int-def*[*symmetric*] *intro*!: *measurable-If*)

lemma (**in** *sigma-finite-measure*) *measurable-emeasure-Pair*:
assumes Q: $Q \in$ *sets* $(N \bigotimes_M M)$ **shows** $(\lambda x.$ *emeasure* M (*Pair* x $-`$ $Q)) \in$ *borel-measurable* N (**is** *?s* $Q \in$ -)
proof −
from *sigma-finite-disjoint* **guess** F **. note** $F =$ *this*
then have *F-sets*: $\bigwedge i.$ F $i \in$ *sets* M **by** *auto*
let *?C* $= \lambda x$ $i.$ F $i \cap$ *Pair* x $-`$ Q
{ fix i
have [*simp*]: *space* $N \times$ F $i \cap$ *space* $N \times$ *space* $M =$ *space* $N \times$ F i
using F *sets.sets-into-space* **by** *auto*
let *?R* $=$ *density* M (*indicator* $(F$ $i))$
have *finite-measure ?R*
using F **by** (*intro finite-measureI*) (*auto simp*: *emeasure-restricted subset-eq*)
then have $(\lambda x.$ *emeasure ?R* (*Pair* x $-`$ (*space* $N \times$ *space ?R* $\cap$ Q))) $\in$ *borel-measurable* N
by (*rule finite-measure.finite-measure-cut-measurable*) (*auto intro*: Q)
moreover have $\bigwedge x.$ *emeasure ?R* (*Pair* x $-`$ (*space* $N \times$ *space ?R* $\cap$ Q))
$=$ *emeasure* M (F $i \cap$ *Pair* x $-`$ (*space* $N \times$ *space ?R* $\cap$ Q))
using Q *F-sets* **by** (*intro emeasure-restricted*) (*auto intro*: *sets-Pair1*)
moreover have $\bigwedge x.$ F $i \cap$ *Pair* x $-`$ (*space* $N \times$ *space ?R* $\cap$ $Q) =$ *?C* x i
using *sets.sets-into-space*[*OF* Q] **by** (*auto simp*: *space-pair-measure*)
ultimately have $(\lambda x.$ *emeasure* M (*?C* x $i)) \in$ *borel-measurable* N
by *simp* **}**
moreover
{ fix x
have $(\sum i.$ *emeasure* M (*?C* x $i)) =$ *emeasure* M $(\bigcup i.$ *?C* x $i)$
proof (*intro suminf-emeasure*)
show *range* (*?C* x) $\subseteq$ *sets* M
using F ‹$Q \in$ *sets* $(N \bigotimes_M M)$› **by** (*auto intro*!: *sets-Pair1*)
have *disjoint-family* F **using** F **by** *auto*
show *disjoint-family* (*?C* x)
by (*rule disjoint-family-on-bisimulation*[*OF* ‹*disjoint-family* F›]) *auto*
qed
also have $(\bigcup i.$ *?C* x $i) =$ *Pair* x $-`$ Q
using F *sets.sets-into-space*[*OF* ‹$Q \in$ *sets* $(N \bigotimes_M M)$›]
by (*auto simp*: *space-pair-measure*)
finally have *emeasure* M (*Pair* x $-`$ $Q) = (\sum i.$ *emeasure* M (*?C* x $i))$
by *simp* **}**
ultimately show *?thesis* **using** ‹$Q \in$ *sets* $(N \bigotimes_M M)$› F-*sets*
by *auto*
qed

lemma (**in** *sigma-finite-measure*) *measurable-emeasure*[*measurable* (*raw*)]:
assumes *space*: $\bigwedge x.\ x \in space\ N \Longrightarrow A\ x \subseteq space\ M$
assumes *A*: $\{x \in space\ (N \bigotimes_M M).\ snd\ x \in A\ (fst\ x)\} \in sets\ (N \bigotimes_M M)$
shows $(\lambda x.\ emeasure\ M\ (A\ x)) \in borel\text{-}measurable\ N$
proof −
from *space* **have** $\bigwedge x.\ x \in space\ N \Longrightarrow Pair\ x\ -`\ \{x \in space\ (N \bigotimes_M M).\ snd\ x \in A\ (fst\ x)\} = A\ x$
by (*auto simp*: *space-pair-measure*)
with *measurable-emeasure-Pair*[*OF A*] **show** *?thesis*
by (*auto cong*: *measurable-cong*)
qed

lemma (**in** *sigma-finite-measure*) *emeasure-pair-measure*:
assumes $X \in sets\ (N \bigotimes_M M)$
shows $emeasure\ (N \bigotimes_M M)\ X = (\int^+ x.\ \int^+ y.\ indicator\ X\ (x, y)\ \partial M\ \partial N)$
(**is** - = $?\mu\ X$)
proof (*rule emeasure-measure-of*[*OF pair-measure-def*])
show *positive* $(sets\ (N \bigotimes_M M))\ ?\mu$
by (*auto simp*: *positive-def*)
have *eq*[*simp*]: $\bigwedge A\ x\ y.\ indicator\ A\ (x, y) = indicator\ (Pair\ x\ -`\ A)\ y$
by (*auto simp*: *indicator-def*)
show *countably-additive* $(sets\ (N \bigotimes_M M))\ ?\mu$
proof (*rule countably-additiveI*)
fix $F :: nat \Rightarrow ('b \times 'a)\ set$ **assume** *F*: $range\ F \subseteq sets\ (N \bigotimes_M M)$ *disjoint-family F*
from *F* **have** $*$: $\bigwedge i.\ F\ i \in sets\ (N \bigotimes_M M)$ **by** *auto*
moreover have $\bigwedge x.\ disjoint\text{-}family\ (\lambda i.\ Pair\ x\ -`\ F\ i)$
by (*intro disjoint-family-on-bisimulation*[*OF F*(*2*)]) *auto*
moreover have $\bigwedge x.\ range\ (\lambda i.\ Pair\ x\ -`\ F\ i) \subseteq sets\ M$
using *F* **by** (*auto simp*: *sets-Pair1*)
ultimately show $(\sum n.\ ?\mu\ (F\ n)) = ?\mu\ (\bigcup i.\ F\ i)$
by (*auto simp add*: *nn-integral-suminf*[*symmetric*] *vimage-UN suminf-emeasure*
intro!: *nn-integral-cong nn-integral-indicator*[*symmetric*])
qed
show $\{a \times b \mid a\ b.\ a \in sets\ N \wedge b \in sets\ M\} \subseteq Pow\ (space\ N \times space\ M)$
using *sets.space-closed*[*of N*] *sets.space-closed*[*of M*] **by** *auto*
qed *fact*

lemma (**in** *sigma-finite-measure*) *emeasure-pair-measure-alt*:
assumes *X*: $X \in sets\ (N \bigotimes_M M)$
shows $emeasure\ (N \bigotimes_M M)\ X = (\int^+ x.\ emeasure\ M\ (Pair\ x\ -`\ X)\ \partial N)$
proof −
have [*simp*]: $\bigwedge x\ y.\ indicator\ X\ (x, y) = indicator\ (Pair\ x\ -`\ X)\ y$
by (*auto simp*: *indicator-def*)
show *?thesis*
using *X* **by** (*auto intro*!: *nn-integral-cong simp*: *emeasure-pair-measure sets-Pair1*)
qed

lemma (**in** *sigma-finite-measure*) *emeasure-pair-measure-Times*:

assumes *A*: $A \in sets\ N$ **and** *B*: $B \in sets\ M$
shows $emeasure\ (N \bigotimes_M M)\ (A \times B) = emeasure\ N\ A * emeasure\ M\ B$
proof −
have $emeasure\ (N \bigotimes_M M)\ (A \times B) = (\int^+ x.\ emeasure\ M\ B * indicator\ A\ x\ \partial N)$
using *A B* **by** (*auto intro*!: *nn-integral-cong simp*: *emeasure-pair-measure-alt*)
also have $\ldots = emeasure\ M\ B * emeasure\ N\ A$
using *A* **by** (*simp add*: *nn-integral-cmult-indicator*)
finally show *?thesis*
by (*simp add*: *ac-simps*)
qed

6.2 Binary products of σ-finite emeasure spaces

locale *pair-sigma-finite* = *M1?*: *sigma-finite-measure M1* + *M2?*: *sigma-finite-measure M2*
for *M1* :: *'a measure* **and** *M2* :: *'b measure*

lemma (**in** *pair-sigma-finite*) *measurable-emeasure-Pair1*:
$Q \in sets\ (M1 \bigotimes_M M2) \Longrightarrow (\lambda x.\ emeasure\ M2\ (Pair\ x\ -`\ Q)) \in borel\text{-}measurable\ M1$
using *M2.measurable-emeasure-Pair* **.**

lemma (**in** *pair-sigma-finite*) *measurable-emeasure-Pair2*:
assumes *Q*: $Q \in sets\ (M1 \bigotimes_M M2)$ **shows** $(\lambda y.\ emeasure\ M1\ ((\lambda x.\ (x,\ y))\ -`\ Q)) \in borel\text{-}measurable\ M2$
proof −
have $(\lambda(x,\ y).\ (y,\ x))\ -`\ Q \cap space\ (M2 \bigotimes_M M1) \in sets\ (M2 \bigotimes_M M1)$
using *Q measurable-pair-swap'* **by** (*auto intro*: *measurable-sets*)
note *M1.measurable-emeasure-Pair*[*OF this*]
moreover have $\bigwedge y.\ Pair\ y\ -`\ ((\lambda(x,\ y).\ (y,\ x))\ -`\ Q \cap space\ (M2 \bigotimes_M M1)) = (\lambda x.\ (x,\ y))\ -`\ Q$
using *Q*[*THEN sets.sets-into-space*] **by** (*auto simp*: *space-pair-measure*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *pair-sigma-finite*) *sigma-finite-up-in-pair-measure-generator*:
defines $E \equiv \{A \times B \mid A\ B.\ A \in sets\ M1 \wedge B \in sets\ M2\}$
shows $\exists F{::}nat \Rightarrow ('a \times 'b)\ set.\ range\ F \subseteq E \wedge incseq\ F \wedge (\bigcup i.\ F\ i) = space\ M1 \times space\ M2 \wedge$
$(\forall i.\ emeasure\ (M1 \bigotimes_M M2)\ (F\ i) \neq \infty)$
proof −
from *M1.sigma-finite-incseq* **guess** *F1* **. note** *F1* = *this*
from *M2.sigma-finite-incseq* **guess** *F2* **. note** *F2* = *this*
from *F1 F2* **have** *space*: $space\ M1 = (\bigcup i.\ F1\ i)\ space\ M2 = (\bigcup i.\ F2\ i)$ **by** *auto*
let *?F* = $\lambda i.\ F1\ i \times F2\ i$
show *?thesis*
proof (*intro exI*[*of* - *?F*] *conjI allI*)

show *range ?F* $\subseteq$ *E* **using** *F1 F2* **by** (*auto simp*: *E-def*) (*metis range-subsetD*)
next
have *space M1* $\times$ *space M2* $\subseteq$ ($\bigcup$ *i*. *?F i*)
proof (*intro subsetI*)
fix *x* **assume** *x* $\in$ *space M1* $\times$ *space M2*
then obtain *i j* **where** *fst x* $\in$ *F1 i snd x* $\in$ *F2 j*
by (*auto simp*: *space*)
then have *fst x* $\in$ *F1* (*max i j*) *snd x* $\in$ *F2* (*max j i*)
using ‹*incseq F1*› ‹*incseq F2*› **unfolding** *incseq-def*
by (*force split*: *split-max*)+
then have (*fst x*, *snd x*) $\in$ *F1* (*max i j*) $\times$ *F2* (*max i j*)
by (*intro SigmaI*) (*auto simp add*: *max.commute*)
then show *x* $\in$ ($\bigcup$ *i*. *?F i*) **by** *auto*
qed
then show ($\bigcup$ *i*. *?F i*) = *space M1* $\times$ *space M2*
using *space* **by** (*auto simp*: *space*)
next
fix *i* **show** *incseq* (λ*i*. *F1 i* $\times$ *F2 i*)
using ‹*incseq F1*› ‹*incseq F2*› **unfolding** *incseq-Suc-iff* **by** *auto*
next
fix *i*
from *F1 F2* **have** *F1 i* $\in$ *sets M1 F2 i* $\in$ *sets M2* **by** *auto*
with *F1 F2* **show** *emeasure* (*M1* $\bigotimes_M$ *M2*) (*F1 i* $\times$ *F2 i*) $\neq \infty$
by (*auto simp add*: *emeasure-pair-measure-Times ennreal-mult-eq-top-iff*)
qed
qed

sublocale *pair-sigma-finite* $\subseteq$ *P?*: *sigma-finite-measure M1* $\bigotimes_M$ *M2*
proof
from *M1.sigma-finite-countable* **guess** *F1* **..**
moreover from *M2.sigma-finite-countable* **guess** *F2* **..**
ultimately show
$\exists$ *A*. *countable A* $\wedge$ *A* $\subseteq$ *sets* (*M1* $\bigotimes_M$ *M2*) $\wedge$ $\bigcup$ *A* = *space* (*M1* $\bigotimes_M$ *M2*) $\wedge$ ($\forall$ *a*$\in$*A*. *emeasure* (*M1* $\bigotimes_M$ *M2*) *a* $\neq \infty$)
by (*intro exI*[*of* - (λ(*a*, *b*). *a* $\times$ *b*) ‘ (*F1* $\times$ *F2*)] *conjI*)
(*auto simp*: *M2.emeasure-pair-measure-Times space-pair-measure set-eq-iff subset-eq ennreal-mult-eq-top-iff*)
qed

lemma *sigma-finite-pair-measure*:
assumes *A*: *sigma-finite-measure A* **and** *B*: *sigma-finite-measure B*
shows *sigma-finite-measure* (*A* $\bigotimes_M$ *B*)
proof −
interpret *A*: *sigma-finite-measure A* **by** *fact*
interpret *B*: *sigma-finite-measure B* **by** *fact*
interpret *AB*: *pair-sigma-finite A B* **..**
show *?thesis* **..**
qed

lemma *sets-pair-swap*:
assumes $A \in sets\ (M1 \bigotimes_M M2)$
shows $(\lambda(x, y).\ (y, x)) -` A \cap space\ (M2 \bigotimes_M M1) \in sets\ (M2 \bigotimes_M M1)$
using *measurable-pair-swap'* *assms* **by** (*rule measurable-sets*)

lemma (**in** *pair-sigma-finite*) *distr-pair-swap*:
$M1 \bigotimes_M M2 = distr\ (M2 \bigotimes_M M1)\ (M1 \bigotimes_M M2)\ (\lambda(x, y).\ (y, x))$ (**is** *?P* = *?D*)
proof −
from *sigma-finite-up-in-pair-measure-generator* **guess** $F :: nat \Rightarrow ('a \times 'b)\ set$
.. note $F = this$
let $?E = \{a \times b \mid a\ b.\ a \in sets\ M1 \wedge b \in sets\ M2\}$
show *?thesis*
proof (*rule measure-eqI-generator-eq*[*OF Int-stable-pair-measure-generator*[*of M1 M2*]])
show $?E \subseteq Pow\ (space\ ?P)$
using *sets.space-closed*[*of M1*] *sets.space-closed*[*of M2*] **by** (*auto simp*: *space-pair-measure*)
show $sets\ ?P = sigma\text{-}sets\ (space\ ?P)\ ?E$
by (*simp add*: *sets-pair-measure space-pair-measure*)
then show $sets\ ?D = sigma\text{-}sets\ (space\ ?P)\ ?E$
by *simp*
next
show $range\ F \subseteq ?E\ (\bigcup i.\ F\ i) = space\ ?P \bigwedge i.\ emeasure\ ?P\ (F\ i) \neq \infty$
using *F* **by** (*auto simp*: *space-pair-measure*)
next
fix *X* **assume** $X \in ?E$
then obtain *A B* **where** *X*[*simp*]: $X = A \times B$ **and** *A*: $A \in sets\ M1$ **and** *B*: $B \in sets\ M2$ **by** *auto*
have $(\lambda(y, x).\ (x, y)) -` X \cap space\ (M2 \bigotimes_M M1) = B \times A$
using *sets.sets-into-space*[*OF A*] *sets.sets-into-space*[*OF B*] **by** (*auto simp*: *space-pair-measure*)
with *A B* **show** $emeasure\ (M1 \bigotimes_M M2)\ X = emeasure\ ?D\ X$
by (*simp add*: *M2.emeasure-pair-measure-Times M1.emeasure-pair-measure-Times emeasure-distr*
measurable-pair-swap' *ac-simps*)
qed
qed

lemma (**in** *pair-sigma-finite*) *emeasure-pair-measure-alt2*:
assumes *A*: $A \in sets\ (M1 \bigotimes_M M2)$
shows $emeasure\ (M1 \bigotimes_M M2)\ A = (\int^+ y.\ emeasure\ M1\ ((\lambda x.\ (x, y)) -` A)\ \partial M2)$
(**is** - = $?\nu\ A$)
proof −
have [*simp*]: $\bigwedge y.\ (Pair\ y -` ((\lambda(x, y).\ (y, x)) -` A \cap space\ (M2 \bigotimes_M M1))) = (\lambda x.\ (x, y)) -` A$
using *sets.sets-into-space*[*OF A*] **by** (*auto simp*: *space-pair-measure*)
show *?thesis* **using** *A*
by (*subst distr-pair-swap*)

```
    (simp-all del: vimage-Int add: measurable-sets[OF measurable-pair-swap']
        M1.emeasure-pair-measure-alt emeasure-distr[OF measurable-pair-swap'
A])
qed

lemma (in pair-sigma-finite) AE-pair:
  assumes AE x in (M1 ⨂M M2). Q x
  shows AE x in M1. (AE y in M2. Q (x, y))
proof −
  obtain N where N: N ∈ sets (M1 ⨂M M2) emeasure (M1 ⨂M M2) N = 0
{x∈space (M1 ⨂M M2). ¬ Q x} ⊆ N
    using assms unfolding eventually-ae-filter by auto
  show ?thesis
  proof (rule AE-I)
    from N measurable-emeasure-Pair1[OF ‹N ∈ sets (M1 ⨂M M2)›]
    show emeasure M1 {x∈space M1. emeasure M2 (Pair x −‘ N) ≠ 0} = 0
      by (auto simp: M2.emeasure-pair-measure-alt nn-integral-0-iff)
    show {x ∈ space M1. emeasure M2 (Pair x −‘ N) ≠ 0} ∈ sets M1
    by (intro borel-measurable-eq measurable-emeasure-Pair1 N sets.sets-Collect-neg
N) simp
    { fix x assume x ∈ space M1 emeasure M2 (Pair x −‘ N) = 0
      have AE y in M2. Q (x, y)
      proof (rule AE-I)
        show emeasure M2 (Pair x −‘ N) = 0 by fact
        show Pair x −‘ N ∈ sets M2 using N(1) by (rule sets-Pair1)
        show {y ∈ space M2. ¬ Q (x, y)} ⊆ Pair x −‘ N
          using N ‹x ∈ space M1› unfolding space-pair-measure by auto
      qed }
    then show {x ∈ space M1. ¬ (AE y in M2. Q (x, y))} ⊆ {x ∈ space M1.
emeasure M2 (Pair x −‘ N) ≠ 0}
      by auto
  qed
qed

lemma (in pair-sigma-finite) AE-pair-measure:
  assumes {x∈space (M1 ⨂M M2). P x} ∈ sets (M1 ⨂M M2)
  assumes ae: AE x in M1. AE y in M2. P (x, y)
  shows AE x in M1 ⨂M M2. P x
proof (subst AE-iff-measurable[OF - refl])
  show {x∈space (M1 ⨂M M2). ¬ P x} ∈ sets (M1 ⨂M M2)
    by (rule sets.sets-Collect) fact
  then have emeasure (M1 ⨂M M2) {x ∈ space (M1 ⨂M M2). ¬ P x} =
    (∫+ x. ∫+ y. indicator {x ∈ space (M1 ⨂M M2). ¬ P x} (x, y) ∂M2 ∂M1)
    by (simp add: M2.emeasure-pair-measure)
  also have ... = (∫+ x. ∫+ y. 0 ∂M2 ∂M1)
    using ae
    apply (safe intro!: nn-integral-cong-AE)
    apply (intro AE-I2)
    apply (safe intro!: nn-integral-cong-AE)
```

apply *auto*
done
finally show *emeasure* (*M1* $\bigotimes_M$ *M2*) {*x* ∈ *space* (*M1* $\bigotimes_M$ *M2*). ¬ *P x*} = *0*
by *simp*
qed

lemma (**in** *pair-sigma-finite*) *AE-pair-iff*:
{*x*∈*space* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*)} ∈ *sets* (*M1* $\bigotimes_M$ *M2*) ⟹
(*AE x in M1*. *AE y in M2*. *P x y*) ⟷ (*AE x in* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*))
using *AE-pair*[*of* λ*x*. *P* (*fst x*) (*snd x*)] *AE-pair-measure*[*of* λ*x*. *P* (*fst x*) (*snd x*)] **by** *auto*

lemma (**in** *pair-sigma-finite*) *AE-commute*:
assumes *P*: {*x*∈*space* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*)} ∈ *sets* (*M1* $\bigotimes_M$ *M2*)
shows (*AE x in M1*. *AE y in M2*. *P x y*) ⟷ (*AE y in M2*. *AE x in M1*. *P x y*)
proof −
interpret *Q*: *pair-sigma-finite M2 M1* **..**
have [*simp*]: ⋀*x*. (*fst* (*case x of* (*x*, *y*) ⇒ (*y*, *x*))) = *snd x* ⋀*x*. (*snd* (*case x of* (*x*, *y*) ⇒ (*y*, *x*))) = *fst x*
by *auto*
have {*x* ∈ *space* (*M2* $\bigotimes_M$ *M1*). *P* (*snd x*) (*fst x*)} =
(λ(*x*, *y*). (*y*, *x*)) −‘ {*x* ∈ *space* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*)} ∩ *space* (*M2* $\bigotimes_M$ *M1*)
by (*auto simp*: *space-pair-measure*)
also have ... ∈ *sets* (*M2* $\bigotimes_M$ *M1*)
by (*intro sets-pair-swap P*)
finally show *?thesis*
apply (*subst AE-pair-iff*[*OF P*])
apply (*subst distr-pair-swap*)
apply (*subst AE-distr-iff*[*OF measurable-pair-swap′ P*])
apply (*subst Q.AE-pair-iff*)
apply *simp-all*
done
qed

6.3 Fubinis theorem

lemma *measurable-compose-Pair1*:
x ∈ *space M1* ⟹ *g* ∈ *measurable* (*M1* $\bigotimes_M$ *M2*) *L* ⟹ (λ*y*. *g* (*x*, *y*)) ∈ *measurable M2 L*
by *simp*

lemma (**in** *sigma-finite-measure*) *borel-measurable-nn-integral-fst*:
assumes *f*: *f* ∈ *borel-measurable* (*M1* $\bigotimes_M$ *M*)
shows (λ*x*. $\int^+$ *y*. *f* (*x*, *y*) ∂*M*) ∈ *borel-measurable M1*
using *f* **proof** *induct*
case (*cong u v*)

then have $\bigwedge w\ x.\ w \in$ *space M1* $\Longrightarrow x \in$ *space M* $\Longrightarrow u\ (w,\ x) = v\ (w,\ x)$
by (*auto simp*: *space-pair-measure*)
show *?case*
apply (*subst measurable-cong*)
apply (*rule nn-integral-cong*)
apply *fact+*
done
next
case (*set Q*)
have [*simp*]: $\bigwedge x\ y.$ *indicator Q* $(x,\ y) =$ *indicator* (*Pair x* $-$‘ *Q*) *y*
by (*auto simp*: *indicator-def*)
have $\bigwedge x.\ x \in$ *space M1* $\Longrightarrow$ *emeasure M* (*Pair x* $-$‘ *Q*) $= \int^+ y.$ *indicator Q* $(x,\ y)\ \partial M$
by (*simp add*: *sets-Pair1*[*OF set*])
from *this measurable-emeasure-Pair*[*OF set*] **show** *?case*
by (*rule measurable-cong*[*THEN iffD1*])
qed (*simp-all add*: *nn-integral-add nn-integral-cmult measurable-compose-Pair1*
nn-integral-monotone-convergence-SUP incseq-def le-fun-def
cong: *measurable-cong*)

lemma (**in** *sigma-finite-measure*) *nn-integral-fst*:
assumes *f*: *f* $\in$ *borel-measurable* (*M1* $\bigotimes_M$ *M*)
shows $(\int^+ x.\ \int^+ y.\ f\ (x,\ y)\ \partial M\ \partial M1) =$ *integral*^{N} (*M1* $\bigotimes_M$ *M*) *f* (**is** *?I f* = -)
using *f* **proof** *induct*
case (*cong u v*)
then have *?I u* = *?I v*
by (*intro nn-integral-cong*) (*auto simp*: *space-pair-measure*)
with *cong* **show** *?case*
by (*simp cong*: *nn-integral-cong*)
qed (*simp-all add*: *emeasure-pair-measure nn-integral-cmult nn-integral-add*
nn-integral-monotone-convergence-SUP measurable-compose-Pair1
borel-measurable-nn-integral-fst nn-integral-mono incseq-def le-fun-def
cong: *nn-integral-cong*)

lemma (**in** *sigma-finite-measure*) *borel-measurable-nn-integral*[*measurable* (*raw*)]:
case-prod f $\in$ *borel-measurable* (*N* $\bigotimes_M$ *M*) $\Longrightarrow$ $(\lambda x.\ \int^+ y.\ f\ x\ y\ \partial M) \in$ *borel-measurable N*
using *borel-measurable-nn-integral-fst*[*of case-prod f N*] **by** *simp*

lemma (**in** *pair-sigma-finite*) *nn-integral-snd*:
assumes *f*[*measurable*]: *f* $\in$ *borel-measurable* (*M1* $\bigotimes_M$ *M2*)
shows $(\int^+ y.\ (\int^+ x.\ f\ (x,\ y)\ \partial M1)\ \partial M2) =$ *integral*^{N} (*M1* $\bigotimes_M$ *M2*) *f*
proof $-$
note *measurable-pair-swap*[*OF f*]
from *M1.nn-integral-fst*[*OF this*]
have $(\int^+ y.\ (\int^+ x.\ f\ (x,\ y)\ \partial M1)\ \partial M2) = (\int^+ (x,\ y).\ f\ (y,\ x)\ \partial(M2 \bigotimes_M M1))$
by *simp*

also have $(\int^+ (x, y).\ f\ (y, x)\ \partial(M2 \bigotimes_M M1)) = integral^N\ (M1 \bigotimes_M M2)\ f$
by (*subst distr-pair-swap*) (*auto simp add*: *nn-integral-distr intro*!: *nn-integral-cong*)
finally show *?thesis* **.**
qed

lemma (**in** *pair-sigma-finite*) *Fubini*:
assumes *f*: $f \in$ *borel-measurable* $(M1 \bigotimes_M M2)$
shows $(\int^+ y.\ (\int^+ x.\ f\ (x, y)\ \partial M1)\ \partial M2) = (\int^+ x.\ (\int^+ y.\ f\ (x, y)\ \partial M2)\ \partial M1)$
unfolding *nn-integral-snd*[*OF assms*] *M2.nn-integral-fst*[*OF assms*] **..**

lemma (**in** *pair-sigma-finite*) *Fubini'*:
assumes *f*: *case-prod f* $\in$ *borel-measurable* $(M1 \bigotimes_M M2)$
shows $(\int^+ y.\ (\int^+ x.\ f\ x\ y\ \partial M1)\ \partial M2) = (\int^+ x.\ (\int^+ y.\ f\ x\ y\ \partial M2)\ \partial M1)$
using *Fubini*[*OF f*] **by** *simp*

6.4 Products on counting spaces, densities and distributions

lemma *sigma-prod*:
assumes *X-cover*: $\exists E{\subseteq}A.$ *countable* $E \wedge X = \bigcup E$ **and** *A*: $A \subseteq Pow\ X$
assumes *Y-cover*: $\exists E{\subseteq}B.$ *countable* $E \wedge Y = \bigcup E$ **and** *B*: $B \subseteq Pow\ Y$
shows *sigma X A* $\bigotimes_M$ *sigma Y B* $=$ *sigma* $(X \times Y)$ $\{a \times b \mid a\ b.\ a \in A \wedge b \in B\}$
(**is** *?P* $=$ *?S*)
proof (*rule measure-eqI*)
have [*simp*]: *snd* $\in X \times Y \to Y$ *fst* $\in X \times Y \to X$
by *auto*
let *?XY* $= \{\{fst -` a \cap X \times Y \mid a.\ a \in A\}, \{snd -` b \cap X \times Y \mid b.\ b \in B\}\}$
have *sets ?P* $=$
sets $(\bigsqcup_\sigma xy{\in}?XY.\ sigma\ (X \times Y)\ xy)$
by (*simp add*: *vimage-algebra-sigma sets-pair-eq-sets-fst-snd A B*)
also have $\ldots$ $=$ *sets* $(sigma\ (X \times Y)\ (\bigcup ?XY))$
by (*intro Sup-sigma-sigma arg-cong*[**where** *f*=*sets*]) *auto*
also have $\ldots$ $=$ *sets ?S*
proof (*intro arg-cong*[**where** *f*=*sets*] *sigma-eqI sigma-sets-eqI*)
show $\bigcup ?XY \subseteq Pow\ (X \times Y)\ \{a \times b \mid a\ b.\ a \in A \wedge b \in B\} \subseteq Pow\ (X \times Y)$
using *A B* **by** *auto*
next
interpret *XY*: *sigma-algebra* $X \times Y$ *sigma-sets* $(X \times Y)$ $\{a \times b \mid a\ b.\ a \in A \wedge b \in B\}$
using *A B* **by** (*intro sigma-algebra-sigma-sets*) *auto*
fix *Z* **assume** $Z \in \bigcup ?XY$
then show $Z \in$ *sigma-sets* $(X \times Y)$ $\{a \times b \mid a\ b.\ a \in A \wedge b \in B\}$
proof *safe*
fix *a* **assume** $a \in A$
from *Y-cover* **obtain** *E* **where** *E*: $E \subseteq B$ *countable E* **and** $Y = \bigcup E$
by *auto*
with ‹$a \in A$› *A* **have** *eq*: *fst* $-` a \cap X \times Y = (\bigcup e{\in}E.\ a \times e)$
by *auto*

```
    show fst −‘ a ∩ X × Y ∈ sigma-sets (X × Y) {a × b |a b. a ∈ A ∧ b ∈ B}
      using ‹a ∈ A› E unfolding eq by (auto intro!: XY.countable-UN')
   next
    fix b assume b ∈ B
    from X-cover obtain E where E: E ⊆ A countable E and X = ⋃ E
      by auto
    with ‹b ∈ B› B have eq: snd −‘ b ∩ X × Y = (⋃ e∈E. e × b)
      by auto
    show snd −‘ b ∩ X × Y ∈ sigma-sets (X × Y) {a × b |a b. a ∈ A ∧ b ∈
B}
      using ‹b ∈ B› E unfolding eq by (auto intro!: XY.countable-UN')
   qed
  next
   fix Z assume Z ∈ {a × b |a b. a ∈ A ∧ b ∈ B}
   then obtain a b where Z = a × b and ab: a ∈ A b ∈ B
    by auto
   then have Z: Z = (fst −‘ a ∩ X × Y) ∩ (snd −‘ b ∩ X × Y)
    using A B by auto
   interpret XY: sigma-algebra X × Y sigma-sets (X × Y) (⋃ ?XY)
    by (intro sigma-algebra-sigma-sets) auto
   show Z ∈ sigma-sets (X × Y) (⋃ ?XY)
    unfolding Z by (rule XY.Int) (blast intro: ab)+
  qed
  finally show sets ?P = sets ?S .
next
  interpret finite-measure sigma X A for X A
   proof qed (simp add: emeasure-sigma)
  fix A assume A ∈ sets ?P then show emeasure ?P A = emeasure ?S A
   by (simp add: emeasure-pair-measure-alt emeasure-sigma)
qed

lemma sigma-sets-pair-measure-generator-finite:
  assumes finite A and finite B
  shows sigma-sets (A × B) { a × b | a b. a ⊆ A ∧ b ⊆ B} = Pow (A × B)
  (is sigma-sets ?prod ?sets = -)
proof safe
  have fin: finite (A × B) using assms by (rule finite-cartesian-product)
  fix x assume subset: x ⊆ A × B
  hence finite x using fin by (rule finite-subset)
  from this subset show x ∈ sigma-sets ?prod ?sets
  proof (induct x)
   case empty show ?case by (rule sigma-sets.Empty)
  next
   case (insert a x)
   hence {a} ∈ sigma-sets ?prod ?sets by auto
   moreover have x ∈ sigma-sets ?prod ?sets using insert by auto
  ultimately show ?case unfolding insert-is-Un[of a x] by (rule sigma-sets-Un)
  qed
next
```

fix *x a b*
assume *x* ∈ *sigma-sets ?prod ?sets* **and** (*a*, *b*) ∈ *x*
from *sigma-sets-into-sp*[*OF - this*(*1*)] *this*(*2*)
show *a* ∈ *A* **and** *b* ∈ *B* **by** *auto*
qed

lemma *borel-prod*:
(*borel* ⨂$_M$ *borel*) = (*borel* :: (′*a*::*second-countable-topology* × ′*b*::*second-countable-topology*) *measure*)
(**is** *?P* = *?B*)
proof −
have *?B* = *sigma UNIV* {*A* × *B* | *A B*. *open A* ∧ *open B*}
by (*rule second-countable-borel-measurable*[*OF open-prod-generated*])
also have ... = *?P*
unfolding *borel-def*
by (*subst sigma-prod*) (*auto intro*!: *exI*[*of - {UNIV}*])
finally show *?thesis* **..**
qed

lemma *pair-measure-count-space*:
assumes *A*: *finite A* **and** *B*: *finite B*
shows *count-space A* ⨂$_M$ *count-space B* = *count-space* (*A* × *B*) (**is** *?P* = *?C*)
proof (*rule measure-eqI*)
interpret *A*: *finite-measure count-space A* **by** (*rule finite-measure-count-space*) *fact*
interpret *B*: *finite-measure count-space B* **by** (*rule finite-measure-count-space*) *fact*
interpret *P*: *pair-sigma-finite count-space A count-space B* **..**
show *eq*: *sets ?P* = *sets ?C*
by (*simp add*: *sets-pair-measure sigma-sets-pair-measure-generator-finite A B*)
fix *X* **assume** *X*: *X* ∈ *sets ?P*
with *eq* **have** *X-subset*: *X* ⊆ *A* × *B* **by** *simp*
with *A B* **have** *fin-Pair*: ⋀*x*. *finite* (*Pair x* −‘ *X*)
by (*intro finite-subset*[*OF - B*]) *auto*
have *fin-X*: *finite X* **using** *X-subset* **by** (*rule finite-subset*) (*auto simp*: *A B*)
have *pos-card*: (*0*::*ennreal*) < *of-nat* (*card* (*Pair x* −‘ *X*)) ⟷ *Pair x* −‘ *X* ≠ {} **for** *x*
by (*auto simp*: *card-eq-0-iff fin-Pair*) *blast*

show *emeasure ?P X* = *emeasure ?C X*
using *X-subset A fin-Pair fin-X*
apply (*subst B.emeasure-pair-measure-alt*[*OF X*])
apply (*subst emeasure-count-space*)
apply (*auto simp add*: *emeasure-count-space nn-integral-count-space*
pos-card of-nat-setsum[*symmetric*] *card-SigmaI*[*symmetric*]
simp del: *of-nat-setsum card-SigmaI*
intro!: *arg-cong*[**where** *f*=*card*])
done
qed

lemma *emeasure-prod-count-space*:
assumes *A*: $A \in$ *sets* (*count-space UNIV* $\bigotimes_M$ *M*) (**is** $A \in$ *sets* (*?A* $\bigotimes_M$ *?B*))
shows *emeasure* (*?A* $\bigotimes_M$ *?B*) *A* = $(\int^+ x.\ \int^+ y.$ *indicator A* (*x*, *y*) ∂*?B* ∂*?A*)
by (*rule emeasure-measure-of*[*OF pair-measure-def*])
(*auto simp*: *countably-additive-def positive-def suminf-indicator A*
nn-integral-suminf[*symmetric*] *dest*: *sets.sets-into-space*)

lemma *emeasure-prod-count-space-single*[*simp*]: *emeasure* (*count-space UNIV* $\bigotimes_M$ *count-space UNIV*) {*x*} = *1*
proof −
have [*simp*]: $\bigwedge$*a b x y. indicator* {(*a*, *b*)} (*x*, *y*) = (*indicator* {*a*} *x* ∗ *indicator* {*b*} *y*::*ennreal*)
by (*auto split*: *split-indicator*)
show *?thesis*
by (*cases x*) (*auto simp*: *emeasure-prod-count-space nn-integral-cmult sets-Pair*)
qed

lemma *emeasure-count-space-prod-eq*:
fixes *A* :: ('*a* × '*b*) *set*
assumes *A*: $A \in$ *sets* (*count-space UNIV* $\bigotimes_M$ *count-space UNIV*) (**is** $A \in$ *sets* (*?A* $\bigotimes_M$ *?B*))
shows *emeasure* (*?A* $\bigotimes_M$ *?B*) *A* = *emeasure* (*count-space UNIV*) *A*
proof −
{ fix *A* :: ('*a* × '*b*) *set* **assume** *countable A*
then have *emeasure* (*?A* $\bigotimes_M$ *?B*) ($\bigcup$ *a*∈*A*. {*a*}) = $(\int^+ a.$ *emeasure* (*?A* $\bigotimes_M$ *?B*) {*a*} ∂*count-space A*)
by (*intro emeasure-UN-countable*) (*auto simp*: *sets-Pair disjoint-family-on-def*)
also have ... = $(\int^+ a.$ *indicator A a* ∂*count-space UNIV*)
by (*subst nn-integral-count-space-indicator*) *auto*
finally have *emeasure* (*?A* $\bigotimes_M$ *?B*) *A* = *emeasure* (*count-space UNIV*) *A*
by *simp* **}**
note ∗ = *this*

show *?thesis*
proof *cases*
assume *finite A* **then show** *?thesis*
by (*intro* ∗ *countable-finite*)
next
assume *infinite A*
then obtain *C* **where** *countable C* **and** *infinite C* **and** *C* ⊆ *A*
by (*auto dest*: *infinite-countable-subset'*)
with *A* **have** *emeasure* (*?A* $\bigotimes_M$ *?B*) *C* ≤ *emeasure* (*?A* $\bigotimes_M$ *?B*) *A*
by (*intro emeasure-mono*) *auto*
also have *emeasure* (*?A* $\bigotimes_M$ *?B*) *C* = *emeasure* (*count-space UNIV*) *C*
using ‹*countable C*› **by** (*rule* ∗)
finally show *?thesis*
using ‹*infinite C*› ‹*infinite A*› **by** (*simp add*: *top-unique*)

qed
qed

lemma *nn-integral-count-space-prod-eq*:
nn-integral (*count-space UNIV* $\bigotimes_M$ *count-space UNIV*) *f* = *nn-integral* (*count-space UNIV*) *f*
(**is** *nn-integral ?P f* = -)
proof *cases*
assume *cntbl*: *countable* {*x*. *f x* ≠ *0*}
have [*simp*]: ⋀*x*. *card* ({*x*} ∩ {*x*. *f x* ≠ *0*}) = (*indicator* {*x*. *f x* ≠ *0*} *x*::*ennreal*)
by (*auto split*: *split-indicator*)
have [*measurable*]: ⋀*y*. (λ*x*. *indicator* {*y*} *x*) ∈ *borel-measurable ?P*
by (*rule measurable-discrete-difference*[*of* λ*x*. *0* - *borel* {*y*} λ*x*. *indicator* {*y*} *x* **for** *y*])
(*auto intro*: *sets-Pair*)

have (∫⁺*x*. *f x* ∂*?P*) = (∫⁺*x*. ∫⁺*x'*. *f x* ∗ *indicator* {*x*} *x'* ∂*count-space* {*x*. *f x* ≠ *0*} ∂*?P*)
by (*auto simp add*: *nn-integral-cmult nn-integral-indicator' intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = (∫⁺*x*. ∫⁺*x'*. *f x'* ∗ *indicator* {*x'*} *x* ∂*count-space* {*x*. *f x* ≠ *0*} ∂*?P*)
by (*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = (∫⁺*x'*. ∫⁺*x*. *f x'* ∗ *indicator* {*x'*} *x* ∂*?P* ∂*count-space* {*x*. *f x* ≠ *0*})
by (*intro nn-integral-count-space-nn-integral cntbl*) *auto*
also have ... = (∫⁺*x'*. *f x'* ∂*count-space* {*x*. *f x* ≠ *0*})
by (*intro nn-integral-cong*) (*auto simp*: *nn-integral-cmult sets-Pair*)
finally show *?thesis*
by (*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
next
{ fix *x* **assume** *f x* ≠ *0*
then have (∃ *r*≥*0*. *0* < *r* ∧ *f x* = *ennreal r*) ∨ *f x* = ∞
by (*cases f x rule*: *ennreal-cases*) (*auto simp*: *less-le*)
then have ∃ *n*. *ennreal* (*1* / *real* (*Suc n*)) ≤ *f x*
by (*auto elim*!: *nat-approx-posE intro*!: *less-imp-le*) **}**
note ∗ = *this*

assume *cntbl*: *uncountable* {*x*. *f x* ≠ *0*}
also have {*x*. *f x* ≠ *0*} = (⋃*n*. {*x*. *1*/*Suc n* ≤ *f x*})
using ∗ **by** *auto*
finally obtain *n* **where** *infinite* {*x*. *1*/*Suc n* ≤ *f x*}
by (*meson countableI-type countable-UN uncountable-infinite*)
then obtain *C* **where** *C*: *C* ⊆ {*x*. *1*/*Suc n* ≤ *f x*} **and** *countable C infinite C*
by (*metis infinite-countable-subset'*)

have [*measurable*]: *C* ∈ *sets ?P*
using *sets.countable*[*OF* - ‹*countable C*›, *of ?P*] **by** (*auto simp*: *sets-Pair*)

have $(\int^+ x.\ ennreal\ (1/Suc\ n) * indicator\ C\ x\ \partial ?P) \leq nn\text{-}integral\ ?P\ f$
using C **by** (*intro nn-integral-mono*) (*auto split*: *split-indicator simp*: *zero-ereal-def*[*symmetric*])
moreover have $(\int^+ x.\ ennreal\ (1/Suc\ n) * indicator\ C\ x\ \partial ?P) = \infty$
using ‹*infinite C*› **by** (*simp add*: *nn-integral-cmult emeasure-count-space-prod-eq ennreal-mult-top*)
moreover have $(\int^+ x.\ ennreal\ (1/Suc\ n) * indicator\ C\ x\ \partial count\text{-}space\ UNIV) \leq nn\text{-}integral\ (count\text{-}space\ UNIV)\ f$
using C **by** (*intro nn-integral-mono*) (*auto split*: *split-indicator simp*: *zero-ereal-def*[*symmetric*])
moreover have $(\int^+ x.\ ennreal\ (1/Suc\ n) * indicator\ C\ x\ \partial count\text{-}space\ UNIV) = \infty$
using ‹*infinite C*› **by** (*simp add*: *nn-integral-cmult ennreal-mult-top*)
ultimately show *?thesis*
by (*simp add*: *top-unique*)
qed

lemma *pair-measure-density*:
assumes f: $f \in borel\text{-}measurable\ M1$
assumes g: $g \in borel\text{-}measurable\ M2$
assumes *sigma-finite-measure M2 sigma-finite-measure* (*density M2 g*)
shows $density\ M1\ f \bigotimes_M density\ M2\ g = density\ (M1 \bigotimes_M M2)\ (\lambda(x,y).\ f\ x * g\ y)$ (**is** *?L* = *?R*)
proof (*rule measure-eqI*)
interpret *M2*: *sigma-finite-measure M2* **by** *fact*
interpret *D2*: *sigma-finite-measure density M2 g* **by** *fact*

fix A **assume** A: $A \in sets\ ?L$
with $f\ g$ **have** $(\int^+ x.\ f\ x * \int^+ y.\ g\ y * indicator\ A\ (x, y)\ \partial M2\ \partial M1) = (\int^+ x.\ \int^+ y.\ f\ x * g\ y * indicator\ A\ (x, y)\ \partial M2\ \partial M1)$
by (*intro nn-integral-cong-AE*)
(*auto simp add*: *nn-integral-cmult*[*symmetric*] *ac-simps*)
with $A\ f\ g$ **show** *emeasure ?L A* = *emeasure ?R A*
by (*simp add*: *D2.emeasure-pair-measure emeasure-density nn-integral-density M2.nn-integral-fst*[*symmetric*] *cong*: *nn-integral-cong*)
qed *simp*

lemma *sigma-finite-measure-distr*:
assumes *sigma-finite-measure* (*distr M N f*) **and** f: $f \in measurable\ M\ N$
shows *sigma-finite-measure M*
proof −
interpret *sigma-finite-measure distr M N f* **by** *fact*
from *sigma-finite-countable* **guess** A **..** **note** A = *this*
show *?thesis*
proof
show $\exists A.\ countable\ A \wedge A \subseteq sets\ M \wedge \bigcup A = space\ M \wedge (\forall a{\in}A.\ emeasure\ M\ a \neq \infty)$
using $A\ f$
by (*intro exI*[*of* - $(\lambda a.\ f\ -`\ a \cap space\ M)$ `` A])

(*auto simp*: *emeasure-distr set-eq-iff subset-eq intro*: *measurable-space*)
qed
qed

lemma *pair-measure-distr*:
assumes *f*: $f \in$ *measurable M S* **and** *g*: $g \in$ *measurable N T*
assumes *sigma-finite-measure* (*distr N T g*)
shows *distr M S f* $\bigotimes_M$ *distr N T g* = *distr* (*M* $\bigotimes_M$ *N*) (*S* $\bigotimes_M$ *T*) ($\lambda(x, y)$. (*f x*, *g y*)) (**is** *?P* = *?D*)
proof (*rule measure-eqI*)
interpret *T*: *sigma-finite-measure distr N T g* **by** *fact*
interpret *N*: *sigma-finite-measure N* **by** (*rule sigma-finite-measure-distr*) *fact+*

fix *A* **assume** *A*: *A* $\in$ *sets ?P*
with *f g* **show** *emeasure ?P A* = *emeasure ?D A*
by (*auto simp add*: *N.emeasure-pair-measure-alt space-pair-measure emeasure-distr T.emeasure-pair-measure-alt nn-integral-distr intro*!: *nn-integral-cong arg-cong*[**where** *f*=*emeasure N*])
qed *simp*

lemma *pair-measure-eqI*:
assumes *sigma-finite-measure M1 sigma-finite-measure M2*
assumes *sets*: *sets* (*M1* $\bigotimes_M$ *M2*) = *sets M*
assumes *emeasure*: $\bigwedge A\ B.\ A \in$ *sets M1* $\Longrightarrow$ $B \in$ *sets M2* $\Longrightarrow$ *emeasure M1 A* $*$ *emeasure M2 B* = *emeasure M* ($A \times B$)
shows *M1* $\bigotimes_M$ *M2* = *M*
proof −
interpret *M1*: *sigma-finite-measure M1* **by** *fact*
interpret *M2*: *sigma-finite-measure M2* **by** *fact*
interpret *pair-sigma-finite M1 M2* **..**
from *sigma-finite-up-in-pair-measure-generator* **guess** *F* :: *nat* $\Rightarrow$ ('*a* $\times$ '*b*) *set* **.. note** *F* = *this*
let *?E* = $\{a \times b \mid a\ b.\ a \in$ *sets M1* $\wedge$ $b \in$ *sets M2*$\}$
let *?P* = *M1* $\bigotimes_M$ *M2*
show *?thesis*
proof (*rule measure-eqI-generator-eq*[*OF Int-stable-pair-measure-generator*[*of M1 M2*]])
show *?E* $\subseteq$ *Pow* (*space ?P*)
using *sets.space-closed*[*of M1*] *sets.space-closed*[*of M2*] **by** (*auto simp*: *space-pair-measure*)
show *sets ?P* = *sigma-sets* (*space ?P*) *?E*
by (*simp add*: *sets-pair-measure space-pair-measure*)
then show *sets M* = *sigma-sets* (*space ?P*) *?E*
using *sets*[*symmetric*] **by** *simp*
next
show *range F* $\subseteq$ *?E* ($\bigcup i.\ F\ i$) = *space ?P* $\bigwedge i.$ *emeasure ?P* (*F i*) $\neq \infty$
using *F* **by** (*auto simp*: *space-pair-measure*)
next
fix *X* **assume** *X* $\in$ *?E*
then obtain *A B* **where** *X*[*simp*]: *X* = *A* $\times$ *B* **and** *A*: *A* $\in$ *sets M1* **and** *B*:

B ∈ *sets M2* **by** *auto*
then have *emeasure ?P X = emeasure M1 A * emeasure M2 B*
by (*simp add*: *M2.emeasure-pair-measure-Times*)
also have ... = *emeasure M* (*A* × *B*)
using *A B emeasure* **by** *auto*
finally show *emeasure ?P X = emeasure M X*
by *simp*
qed
qed

lemma *sets-pair-countable*:
assumes *countable S1 countable S2*
assumes *M*: *sets M = Pow S1* **and** *N*: *sets N = Pow S2*
shows *sets* (*M* ⨂$_M$ *N*) = *Pow* (*S1* × *S2*)
proof *auto*
fix *x a b* **assume** *x*: *x* ∈ *sets* (*M* ⨂$_M$ *N*) (*a*, *b*) ∈ *x*
from *sets.sets-into-space*[*OF x*(*1*)] *x*(*2*)
sets-eq-imp-space-eq[*of N count-space S2*] *sets-eq-imp-space-eq*[*of M count-space S1*] *M N*
show *a* ∈ *S1 b* ∈ *S2*
by (*auto simp*: *space-pair-measure*)
next
fix *X* **assume** *X*: *X* ⊆ *S1* × *S2*
then have *countable X*
by (*metis countable-subset* ‹*countable S1*› ‹*countable S2*› *countable-SIGMA*)
have *X* = (⋃(*a*, *b*)∈*X*. {*a*} × {*b*}) **by** *auto*
also have ... ∈ *sets* (*M* ⨂$_M$ *N*)
using *X*
by (*safe intro*!: *sets.countable-UN'* ‹*countable X*› *subsetI pair-measureI*) (*auto simp*: *M N*)
finally show *X* ∈ *sets* (*M* ⨂$_M$ *N*) **.**
qed

lemma *pair-measure-countable*:
assumes *countable S1 countable S2*
shows *count-space S1* ⨂$_M$ *count-space S2* = *count-space* (*S1* × *S2*)
proof (*rule pair-measure-eqI*)
show *sigma-finite-measure* (*count-space S1*) *sigma-finite-measure* (*count-space S2*)
using *assms* **by** (*auto intro*!: *sigma-finite-measure-count-space-countable*)
show *sets* (*count-space S1* ⨂$_M$ *count-space S2*) = *sets* (*count-space* (*S1* × *S2*))
by (*subst sets-pair-countable*[*OF assms*]) *auto*
next
fix *A B* **assume** *A* ∈ *sets* (*count-space S1*) *B* ∈ *sets* (*count-space S2*)
then show *emeasure* (*count-space S1*) *A * emeasure* (*count-space S2*) *B* =
emeasure (*count-space* (*S1* × *S2*)) (*A* × *B*)
by (*subst* (*1 2 3*) *emeasure-count-space*) (*auto simp*: *finite-cartesian-product-iff ennreal-mult-top ennreal-top-mult*)
qed

lemma *nn-integral-fst-count-space*:
(∫⁺ x. ∫⁺ y. f (x, y) ∂count-space UNIV ∂count-space UNIV) = integral^N (count-space UNIV) f
(**is** *?lhs* = *?rhs*)
proof(*cases*)
assume ∗: *countable* {*xy*. *f xy* ≠ 0}
let *?A* = *fst* ‘ {*xy*. *f xy* ≠ 0}
let *?B* = *snd* ‘ {*xy*. *f xy* ≠ 0}
from ∗ **have** [*simp*]: *countable ?A countable ?B* **by**(*rule countable-image*)+
have *?lhs* = (∫⁺ *x*. ∫⁺ *y*. *f* (*x*, *y*) ∂*count-space UNIV* ∂*count-space ?A*)
by(*rule nn-integral-count-space-eq*)
(*auto simp add*: *nn-integral-0-iff-AE AE-count-space not-le intro*: *rev-image-eqI*)
also have ... = (∫⁺ *x*. ∫⁺ *y*. *f* (*x*, *y*) ∂*count-space ?B* ∂*count-space ?A*)
by(*intro nn-integral-count-space-eq nn-integral-cong*)(*auto intro*: *rev-image-eqI*)
also have ... = (∫⁺ *xy*. *f xy* ∂*count-space* (*?A* × *?B*))
by(*subst sigma-finite-measure.nn-integral-fst*)
(*simp-all add*: *sigma-finite-measure-count-space-countable pair-measure-countable*)
also have ... = *?rhs*
by(*rule nn-integral-count-space-eq*)(*auto intro*: *rev-image-eqI*)
finally show *?thesis* **.**
next
{ fix *xy* **assume** *f xy* ≠ *0*
then have (∃ *r*≥*0*. *0* < *r* ∧ *f xy* = *ennreal r*) ∨ *f xy* = ∞
by (*cases f xy rule*: *ennreal-cases*) (*auto simp*: *less-le*)
then have ∃ *n*. *ennreal* (*1* / *real* (*Suc n*)) ≤ *f xy*
by (*auto elim*!: *nat-approx-posE intro*!: *less-imp-le*) **}**
note ∗ = *this*

assume *cntbl*: *uncountable* {*xy*. *f xy* ≠ *0*}
also have {*xy*. *f xy* ≠ *0*} = (⋃ *n*. {*xy*. *1*/*Suc n* ≤ *f xy*})
using ∗ **by** *auto*
finally obtain *n* **where** *infinite* {*xy*. *1*/*Suc n* ≤ *f xy*}
by (*meson countableI-type countable-UN uncountable-infinite*)
then obtain *C* **where** *C*: *C* ⊆ {*xy*. *1*/*Suc n* ≤ *f xy*} **and** *countable C infinite C*
by (*metis infinite-countable-subset*′)

have ∞ = (∫⁺ *xy*. *ennreal* (*1* / *Suc n*) ∗ *indicator C xy* ∂*count-space UNIV*)
using ‹*infinite C*› **by**(*simp add*: *nn-integral-cmult ennreal-mult-top*)
also have ... ≤ *?rhs* **using** *C*
by(*intro nn-integral-mono*)(*auto split*: *split-indicator*)
finally have *?rhs* = ∞ **by** (*simp add*: *top-unique*)
moreover have *?lhs* = ∞
proof(*cases finite* (*fst* ‘ *C*))
case *True*
then obtain *x C*′ **where** *x*: *x* ∈ *fst* ‘ *C*
and *C*′: *C*′ = *fst* −‘ {*x*} ∩ *C*
and *infinite C*′

using ‹*infinite C*› **by**(*auto elim*!: *inf-img-fin-domE'*)
from *x C C'* **have** ∗∗: $C' \subseteq \{xy.\ 1\ /\ Suc\ n \leq f\ xy\}$ **by** *auto*

from *C'* ‹*infinite C'*› **have** *infinite* (*snd* ‘ *C'*)
by(*auto dest*!: *finite-imageD simp add*: *inj-on-def*)
then have $\infty = (\int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ (snd\ `\ C')\ y\ \partial count\text{-}space\ UNIV)$
by(*simp add*: *nn-integral-cmult ennreal-mult-top*)
also have ... $= (\int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ (x,\ y)\ \partial count\text{-}space\ UNIV)$
by(*rule nn-integral-cong*)(*force split*: *split-indicator intro*: *rev-image-eqI simp add*: *C'*)
also have ... $= (\int^+ x'.\ (\int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ (x,\ y)\ \partial count\text{-}space\ UNIV) * indicator\ \{x\}\ x'\ \partial count\text{-}space\ UNIV)$
by(*simp add*: *one-ereal-def*[*symmetric*])
also have ... $\leq (\int^+ x.\ \int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ (x,\ y)\ \partial count\text{-}space\ UNIV\ \partial count\text{-}space\ UNIV)$
by(*rule nn-integral-mono*)(*simp split*: *split-indicator*)
also have ... $\leq$ *?lhs* **using** ∗∗
by(*intro nn-integral-mono*)(*auto split*: *split-indicator*)
finally show *?thesis* **by** (*simp add*: *top-unique*)
next
case *False*
def $C' \equiv fst\ `\ C$
have $\infty = \int^+ x.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ x\ \partial count\text{-}space\ UNIV$
using *C'-def False* **by**(*simp add*: *nn-integral-cmult ennreal-mult-top*)
also have ... $= \int^+ x.\ \int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ x * indicator\ \{SOME\ y.\ (x,\ y) \in C\}\ y\ \partial count\text{-}space\ UNIV\ \partial count\text{-}space\ UNIV$
by(*auto simp add*: *one-ereal-def*[*symmetric*] *max-def intro*: *nn-integral-cong*)
also have ... $\leq \int^+ x.\ \int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C\ (x,\ y)\ \partial count\text{-}space\ UNIV\ \partial count\text{-}space\ UNIV$
by(*intro nn-integral-mono*)(*auto simp add*: *C'-def split*: *split-indicator intro*: *someI*)
also have ... $\leq$ *?lhs* **using** *C*
by(*intro nn-integral-mono*)(*auto split*: *split-indicator*)
finally show *?thesis* **by** (*simp add*: *top-unique*)
qed
ultimately show *?thesis* **by** *simp*
qed

lemma *nn-integral-snd-count-space*:
$(\int^+ y.\ \int^+ x.\ f\ (x,\ y)\ \partial count\text{-}space\ UNIV\ \partial count\text{-}space\ UNIV) = integral^N\ (count\text{-}space\ UNIV)\ f$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* $= (\int^+ y.\ \int^+ x.\ (\lambda(y,\ x).\ f\ (x,\ y))\ (y,\ x)\ \partial count\text{-}space\ UNIV\ \partial count\text{-}space\ UNIV)$
by(*simp*)
also have ... $= \int^+ yx.\ (\lambda(y,\ x).\ f\ (x,\ y))\ yx\ \partial count\text{-}space\ UNIV$

by(*rule nn-integral-fst-count-space*)
also have ... = $\int^+ xy.\ f\ xy\ \partial$*count-space* ((λ(*x*, *y*). (*y*, *x*)) ‘ *UNIV*)
by(*subst nn-integral-bij-count-space*[*OF inj-on-imp-bij-betw*, *symmetric*])
(*simp-all add*: *inj-on-def split-def*)
also have ... = *?rhs* **by**(*rule nn-integral-count-space-eq*) *auto*
finally show *?thesis* **.**
qed

lemma *measurable-pair-measure-countable1*:
assumes *countable A*
and [*measurable*]: $\bigwedge x.\ x \in A \Longrightarrow (\lambda y.\ f\ (x, y)) \in$ *measurable N K*
shows $f \in$ *measurable* (*count-space A* $\bigotimes_M$ *N*) *K*
using - - *assms*(*1*)
by(*rule measurable-compose-countable'*[**where** *f*=λ*a b*. *f* (*a*, *snd b*) **and** *g*=*fst* **and** *I*=*A*, *simplified*])*simp-all*

6.5 Product of Borel spaces

lemma *borel-Times*:
fixes *A* :: *'a*::*topological-space set* **and** *B* :: *'b*::*topological-space set*
assumes *A*: *A* ∈ *sets borel* **and** *B*: *B* ∈ *sets borel*
shows *A* × *B* ∈ *sets borel*
proof −
have *A* × *B* = (*A*×*UNIV*) ∩ (*UNIV* × *B*)
by *auto*
moreover
{ have *A* ∈ *sigma-sets UNIV* {*S*. *open S*} **using** *A* **by** (*simp add*: *sets-borel*)
then have *A*×*UNIV* ∈ *sets borel*
proof (*induct A*)
case (*Basic S*) **then show** *?case*
by (*auto intro*!: *borel-open open-Times*)
next
case (*Compl A*)
moreover have ∗: (*UNIV* − *A*) × *UNIV* = *UNIV* − (*A* × *UNIV*)
by *auto*
ultimately show *?case*
unfolding ∗ **by** *auto*
next
case (*Union A*)
moreover have ∗: (*UNION UNIV A*) × *UNIV* = *UNION UNIV* (λ*i*. *A i* × *UNIV*)
by *auto*
ultimately show *?case*
unfolding ∗ **by** *auto*
qed *simp* **}**
moreover
{ have *B* ∈ *sigma-sets UNIV* {*S*. *open S*} **using** *B* **by** (*simp add*: *sets-borel*)
then have *UNIV*×*B* ∈ *sets borel*
proof (*induct B*)

case (*Basic S*) **then show** *?case*
by (*auto intro*!: *borel-open open-Times*)
next
case (*Compl B*)
moreover have ∗: *UNIV* × (*UNIV* − *B*) = *UNIV* − (*UNIV* × *B*)
by *auto*
ultimately show *?case*
unfolding ∗ **by** *auto*
next
case (*Union B*)
moreover have ∗: *UNIV* × (*UNION UNIV B*) = *UNION UNIV* (λ*i*. *UNIV* × *B i*)
by *auto*
ultimately show *?case*
unfolding ∗ **by** *auto*
qed *simp* **}**
ultimately show *?thesis*
by *auto*
qed

lemma *finite-measure-pair-measure*:
assumes *finite-measure M finite-measure N*
shows *finite-measure* (*N* ⨂$_M$ *M*)
proof (*rule finite-measureI*)
interpret *M*: *finite-measure M* **by** *fact*
interpret *N*: *finite-measure N* **by** *fact*
show *emeasure* (*N* ⨂$_M$ *M*) (*space* (*N* ⨂$_M$ *M*)) ≠ ∞
by (*auto simp*: *space-pair-measure M.emeasure-pair-measure-Times ennreal-mult-eq-top-iff*)
qed

end

7 Finite product measures

theory *Finite-Product-Measure*
imports *Binary-Product-Measure*
begin

lemma *PiE-choice*: (∃*f*∈*PiE I F*. ∀*i*∈*I*. *P i* (*f i*)) ⟷ (∀*i*∈*I*. ∃*x*∈*F i*. *P i x*)
by (*auto simp*: *Bex-def PiE-iff Ball-def dest*!: *choice-iff'*[*THEN iffD1*])
(*force intro*: *exI*[*of* - *restrict f I* **for** *f*])

lemma *case-prod-const*: (λ(*i*, *j*). *c*) = (λ-. *c*)
by *auto*

7.0.1 More about Function restricted by *extensional*

definition
merge I J = (λ(*x*, *y*) *i*. *if i* ∈ *I then x i else if i* ∈ *J then y i else undefined*)

lemma *merge-apply*[*simp*]:
$I \cap J = \{\} \Longrightarrow i \in I \Longrightarrow merge\ I\ J\ (x,\ y)\ i = x\ i$
$I \cap J = \{\} \Longrightarrow i \in J \Longrightarrow merge\ I\ J\ (x,\ y)\ i = y\ i$
$J \cap I = \{\} \Longrightarrow i \in I \Longrightarrow merge\ I\ J\ (x,\ y)\ i = x\ i$
$J \cap I = \{\} \Longrightarrow i \in J \Longrightarrow merge\ I\ J\ (x,\ y)\ i = y\ i$
$i \notin I \Longrightarrow i \notin J \Longrightarrow merge\ I\ J\ (x,\ y)\ i = undefined$
unfolding *merge-def* **by** *auto*

lemma *merge-commute*:
$I \cap J = \{\} \Longrightarrow merge\ I\ J\ (x,\ y) = merge\ J\ I\ (y,\ x)$
by (*force simp*: *merge-def*)

lemma *Pi-cancel-merge-range*[*simp*]:
$I \cap J = \{\} \Longrightarrow x \in Pi\ I\ (merge\ I\ J\ (A,\ B)) \longleftrightarrow x \in Pi\ I\ A$
$I \cap J = \{\} \Longrightarrow x \in Pi\ I\ (merge\ J\ I\ (B,\ A)) \longleftrightarrow x \in Pi\ I\ A$
$J \cap I = \{\} \Longrightarrow x \in Pi\ I\ (merge\ I\ J\ (A,\ B)) \longleftrightarrow x \in Pi\ I\ A$
$J \cap I = \{\} \Longrightarrow x \in Pi\ I\ (merge\ J\ I\ (B,\ A)) \longleftrightarrow x \in Pi\ I\ A$
by (*auto simp*: *Pi-def*)

lemma *Pi-cancel-merge*[*simp*]:
$I \cap J = \{\} \Longrightarrow merge\ I\ J\ (x,\ y) \in Pi\ I\ B \longleftrightarrow x \in Pi\ I\ B$
$J \cap I = \{\} \Longrightarrow merge\ I\ J\ (x,\ y) \in Pi\ I\ B \longleftrightarrow x \in Pi\ I\ B$
$I \cap J = \{\} \Longrightarrow merge\ I\ J\ (x,\ y) \in Pi\ J\ B \longleftrightarrow y \in Pi\ J\ B$
$J \cap I = \{\} \Longrightarrow merge\ I\ J\ (x,\ y) \in Pi\ J\ B \longleftrightarrow y \in Pi\ J\ B$
by (*auto simp*: *Pi-def*)

lemma *extensional-merge*[*simp*]: $merge\ I\ J\ (x,\ y) \in extensional\ (I \cup J)$
by (*auto simp*: *extensional-def*)

lemma *restrict-merge*[*simp*]:
$I \cap J = \{\} \Longrightarrow restrict\ (merge\ I\ J\ (x,\ y))\ I = restrict\ x\ I$
$I \cap J = \{\} \Longrightarrow restrict\ (merge\ I\ J\ (x,\ y))\ J = restrict\ y\ J$
$J \cap I = \{\} \Longrightarrow restrict\ (merge\ I\ J\ (x,\ y))\ I = restrict\ x\ I$
$J \cap I = \{\} \Longrightarrow restrict\ (merge\ I\ J\ (x,\ y))\ J = restrict\ y\ J$
by (*auto simp*: *restrict-def*)

lemma *split-merge*: $P\ (merge\ I\ J\ (x,y)\ i) \longleftrightarrow (i \in I \longrightarrow P\ (x\ i)) \wedge (i \in J - I \longrightarrow P\ (y\ i)) \wedge (i \notin I \cup J \longrightarrow P\ undefined)$
unfolding *merge-def* **by** *auto*

lemma *PiE-cancel-merge*[*simp*]:
$I \cap J = \{\} \Longrightarrow$
$merge\ I\ J\ (x,\ y) \in PiE\ (I \cup J)\ B \longleftrightarrow x \in Pi\ I\ B \wedge y \in Pi\ J\ B$
by (*auto simp*: *PiE-def restrict-Pi-cancel*)

lemma *merge-singleton*[*simp*]: $i \notin I \Longrightarrow merge\ I\ \{i\}\ (x,y) = restrict\ (x(i := y\ i))\ (insert\ i\ I)$
unfolding *merge-def* **by** (*auto simp*: *fun-eq-iff*)

lemma *extensional-merge-sub*: $I \cup J \subseteq K \Longrightarrow$ *merge I J* $(x, y) \in$ *extensional K*
unfolding *merge-def extensional-def* **by** *auto*

lemma *merge-restrict*[*simp*]:
merge I J (*restrict x I*, *y*) = *merge I J* (*x*, *y*)
merge I J (*x*, *restrict y J*) = *merge I J* (*x*, *y*)
unfolding *merge-def* **by** *auto*

lemma *merge-x-x-eq-restrict*[*simp*]:
merge I J (*x*, *x*) = *restrict x* (*I* ∪ *J*)
unfolding *merge-def* **by** *auto*

lemma *injective-vimage-restrict*:
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*: $(\lambda x.\ \textit{restrict}\ x\ J) -` A \cap (\Pi_E\ i{\in}I.\ S\ i) = (\lambda x.\ \textit{restrict}\ x\ J) -` B \cap (\Pi_E\ i{\in}I.\ S\ i)$
shows $A = B$
proof (*intro set-eqI*)
fix *x*
from *ne* **obtain** *y* **where** *y*: $\bigwedge i.\ i \in I \Longrightarrow y\ i \in S\ i$ **by** *auto*
have $J \cap (I - J) = \{\}$ **by** *auto*
show $x \in A \longleftrightarrow x \in B$
proof *cases*
assume *x*: $x \in (\Pi_E\ i{\in}J.\ S\ i)$
have $x \in A \longleftrightarrow \textit{merge}\ J\ (I - J)\ (x,y) \in (\lambda x.\ \textit{restrict}\ x\ J) -` A \cap (\Pi_E\ i{\in}I.\ S\ i)$
using *y x* ‹$J \subseteq I$› *PiE-cancel-merge*[*of J I* − *J x y S*]
by (*auto simp del*: *PiE-cancel-merge simp add*: *Un-absorb1*)
then show $x \in A \longleftrightarrow x \in B$
using *y x* ‹$J \subseteq I$› *PiE-cancel-merge*[*of J I* − *J x y S*]
by (*auto simp del*: *PiE-cancel-merge simp add*: *Un-absorb1 eq*)
qed (*insert sets*, *auto*)
qed

lemma *restrict-vimage*:
$I \cap J = \{\} \Longrightarrow$
$(\lambda x.\ (\textit{restrict}\ x\ I, \textit{restrict}\ x\ J)) -` (\textit{Pi}_E\ I\ E \times \textit{Pi}_E\ J\ F) = \textit{Pi}\ (I \cup J)\ (\textit{merge}\ I\ J\ (E, F))$
by (*auto simp*: *restrict-Pi-cancel PiE-def*)

lemma *merge-vimage*:
$I \cap J = \{\} \Longrightarrow \textit{merge}\ I\ J -` \textit{Pi}_E\ (I \cup J)\ E = \textit{Pi}\ I\ E \times \textit{Pi}\ J\ E$
by (*auto simp*: *restrict-Pi-cancel PiE-def*)

7.1 Finite product spaces

7.1.1 Products

definition *prod-emb* **where**
prod-emb I M K X = (λ*x*. *restrict x K*) −‘ *X* ∩ (*PIE i*:*I*. *space* (*M i*))

lemma *prod-emb-iff*:
f ∈ *prod-emb I M K X* ⟷ *f* ∈ *extensional I* ∧ (*restrict f K* ∈ *X*) ∧ (∀ *i*∈*I*. *f i* ∈ *space* (*M i*))
unfolding *prod-emb-def PiE-def* **by** *auto*

lemma
shows *prod-emb-empty*[*simp*]: *prod-emb M L K* {} = {}
and *prod-emb-Un*[*simp*]: *prod-emb M L K* (*A* ∪ *B*) = *prod-emb M L K A* ∪ *prod-emb M L K B*
and *prod-emb-Int*: *prod-emb M L K* (*A* ∩ *B*) = *prod-emb M L K A* ∩ *prod-emb M L K B*
and *prod-emb-UN*[*simp*]: *prod-emb M L K* (⋃ *i*∈*I*. *F i*) = (⋃ *i*∈*I*. *prod-emb M L K* (*F i*))
and *prod-emb-INT*[*simp*]: *I* ≠ {} ⟹ *prod-emb M L K* (⋂ *i*∈*I*. *F i*) = (⋂ *i*∈*I*. *prod-emb M L K* (*F i*))
and *prod-emb-Diff*[*simp*]: *prod-emb M L K* (*A* − *B*) = *prod-emb M L K A* − *prod-emb M L K B*
by (*auto simp*: *prod-emb-def*)

lemma *prod-emb-PiE*: *J* ⊆ *I* ⟹ (⋀*i*. *i* ∈ *J* ⟹ *E i* ⊆ *space* (*M i*)) ⟹
prod-emb I M J (Π$_E$ *i*∈*J*. *E i*) = (Π$_E$ *i*∈*I*. *if i* ∈ *J then E i else space* (*M i*))
by (*force simp*: *prod-emb-def PiE-iff if-split-mem2*)

lemma *prod-emb-PiE-same-index*[*simp*]:
(⋀*i*. *i* ∈ *I* ⟹ *E i* ⊆ *space* (*M i*)) ⟹ *prod-emb I M I* (*Pi*$_E$ *I E*) = *Pi*$_E$ *I E*
by (*auto simp*: *prod-emb-def PiE-iff*)

lemma *prod-emb-trans*[*simp*]:
J ⊆ *K* ⟹ *K* ⊆ *L* ⟹ *prod-emb L M K* (*prod-emb K M J X*) = *prod-emb L M J X*
by (*auto simp add*: *Int-absorb1 prod-emb-def PiE-def*)

lemma *prod-emb-Pi*:
assumes *X* ∈ (Π *j*∈*J*. *sets* (*M j*)) *J* ⊆ *K*
shows *prod-emb K M J* (*Pi*$_E$ *J X*) = (Π$_E$ *i*∈*K*. *if i* ∈ *J then X i else space* (*M i*))
using *assms sets.space-closed*
by (*auto simp*: *prod-emb-def PiE-iff split*: *if-split-asm*) *blast*+

lemma *prod-emb-id*:
B ⊆ (Π$_E$ *i*∈*L*. *space* (*M i*)) ⟹ *prod-emb L M L B* = *B*
by (*auto simp*: *prod-emb-def subset-eq extensional-restrict*)

lemma *prod-emb-mono*:
F ⊆ *G* ⟹ *prod-emb A M B F* ⊆ *prod-emb A M B G*
by (*auto simp*: *prod-emb-def*)

definition *PiM* :: *'i set* ⇒ (*'i* ⇒ *'a measure*) ⇒ (*'i* ⇒ *'a*) *measure* **where**
PiM I M = *extend-measure* (Π$_E$ *i*∈*I*. *space* (*M i*))
{(*J*, *X*). (*J* ≠ {} ∨ *I* = {}) ∧ *finite J* ∧ *J* ⊆ *I* ∧ *X* ∈ (Π *j*∈*J*. *sets* (*M j*))}
(λ(*J*, *X*). *prod-emb I M J* (Π$_E$ *j*∈*J*. *X j*))
(λ(*J*, *X*). ∏*j*∈*J* ∪ {*i*∈*I*. *emeasure* (*M i*) (*space* (*M i*)) ≠ *1*}. *if j* ∈ *J then emeasure* (*M j*) (*X j*) *else emeasure* (*M j*) (*space* (*M j*)))

definition *prod-algebra* :: *'i set* ⇒ (*'i* ⇒ *'a measure*) ⇒ (*'i* ⇒ *'a*) *set set* **where**
prod-algebra I M = (λ(*J*, *X*). *prod-emb I M J* (Π$_E$ *j*∈*J*. *X j*)) ‘
{(*J*, *X*). (*J* ≠ {} ∨ *I* = {}) ∧ *finite J* ∧ *J* ⊆ *I* ∧ *X* ∈ (Π *j*∈*J*. *sets* (*M j*))}

abbreviation
Pi$_M$ I M ≡ *PiM I M*

syntax
-PiM :: *pttrn* ⇒ *'i set* ⇒ *'a measure* ⇒ (*'i* => *'a*) *measure* ((*3*Π$_M$ *-*∈*-*./ *-*) *10*)
translations
Π$_M$ *x*∈*I*. *M* == *CONST PiM I* (%*x*. *M*)

lemma *extend-measure-cong*:
assumes Ω = Ω′ *I* = *I*′ *G* = *G*′ ⋀*i*. *i* ∈ *I*′ ⟹ μ *i* = μ′ *i*
shows *extend-measure* Ω *I G* μ = *extend-measure* Ω′ *I*′ *G*′ μ′
unfolding *extend-measure-def* **by** (*auto simp add*: *assms*)

lemma *Pi-cong-sets*:
⟦*I* = *J*; ⋀*x*. *x* ∈ *I* ⟹ *M x* = *N x*⟧ ⟹ *Pi I M* = *Pi J N*
unfolding *Pi-def* **by** *auto*

lemma *PiM-cong*:
assumes *I* = *J* ⋀*x*. *x* ∈ *I* ⟹ *M x* = *N x*
shows *PiM I M* = *PiM J N*
unfolding *PiM-def*
proof (*rule extend-measure-cong*, *goal-cases*)
case *1*
show *?case* **using** *assms*
by (*subst assms*(*1*), *intro PiE-cong*[*of J* λ*i*. *space* (*M i*) λ*i*. *space* (*N i*)])
simp-all
next
case *2*
have ⋀*K*. *K* ⊆ *J* ⟹ (Π *j*∈*K*. *sets* (*M j*)) = (Π *j*∈*K*. *sets* (*N j*))
using *assms* **by** (*intro Pi-cong-sets*) *auto*
thus *?case* **by** (*auto simp*: *assms*)
next
case *3*
show *?case* **using** *assms*

by (*intro ext*) (*auto simp*: *prod-emb-def dest*: *PiE-mem*)
next
case (*4 x*)
thus *?case* **using** *assms*
by (*auto intro*!: *setprod.cong split*: *if-split-asm*)
qed

lemma *prod-algebra-sets-into-space*:
prod-algebra I M ⊆ *Pow* (Π$_E$ *i*∈*I*. *space* (*M i*))
by (*auto simp*: *prod-emb-def prod-algebra-def*)

lemma *prod-algebra-eq-finite*:
assumes *I*: *finite I*
shows *prod-algebra I M* = {(Π$_E$ *i*∈*I*. *X i*) |*X*. *X* ∈ (Π *j*∈*I*. *sets* (*M j*))} (**is** *?L* = *?R*)
proof (*intro iffI set-eqI*)
fix *A* **assume** *A* ∈ *?L*
then obtain *J E* **where** *J*: *J* ≠ {} ∨ *I* = {} *finite J J* ⊆ *I* ∀ *i*∈*J*. *E i* ∈ *sets* (*M i*)
and *A*: *A* = *prod-emb I M J* (*PIE j*:*J*. *E j*)
by (*auto simp*: *prod-algebra-def*)
let *?A* = Π$_E$ *i*∈*I*. *if i* ∈ *J then E i else space* (*M i*)
have *A*: *A* = *?A*
unfolding *A* **using** *J* **by** (*intro prod-emb-PiE sets.sets-into-space*) *auto*
show *A* ∈ *?R* **unfolding** *A* **using** *J sets.top*
by (*intro CollectI exI*[*of* - λ*i*. *if i* ∈ *J then E i else space* (*M i*)]) *simp*
next
fix *A* **assume** *A* ∈ *?R*
then obtain *X* **where** *A*: *A* = (Π$_E$ *i*∈*I*. *X i*) **and** *X*: *X* ∈ (Π *j*∈*I*. *sets* (*M j*))
by *auto*
then have *A*: *A* = *prod-emb I M I* (Π$_E$ *i*∈*I*. *X i*)
by (*simp add*: *prod-emb-PiE-same-index*[*OF sets.sets-into-space*] *Pi-iff*)
from *X I* **show** *A* ∈ *?L* **unfolding** *A*
by (*auto simp*: *prod-algebra-def*)
qed

lemma *prod-algebraI*:
finite J ⟹ (*J* ≠ {} ∨ *I* = {}) ⟹ *J* ⊆ *I* ⟹ (⋀*i*. *i* ∈ *J* ⟹ *E i* ∈ *sets* (*M i*))
⟹ *prod-emb I M J* (*PIE j*:*J*. *E j*) ∈ *prod-algebra I M*
by (*auto simp*: *prod-algebra-def*)

lemma *prod-algebraI-finite*:
finite I ⟹ (∀ *i*∈*I*. *E i* ∈ *sets* (*M i*)) ⟹ (*Pi$_E$ I E*) ∈ *prod-algebra I M*
using *prod-algebraI*[*of I I E M*] *prod-emb-PiE-same-index*[*of I E M*, *OF sets.sets-into-space*]
by *simp*

lemma *Int-stable-PiE*: *Int-stable* {*Pi$_E$ J E* | *E*. ∀ *i*∈*I*. *E i* ∈ *sets* (*M i*)}
proof (*safe intro*!: *Int-stableI*)

fix E F **assume** $\forall i{\in}I.\ E\ i \in sets\ (M\ i)\ \forall i{\in}I.\ F\ i \in sets\ (M\ i)$
then show $\exists G.\ Pi_E\ J\ E \cap Pi_E\ J\ F = Pi_E\ J\ G \wedge (\forall i{\in}I.\ G\ i \in sets\ (M\ i))$
by (*auto intro*!: *exI*[*of* - $\lambda i.\ E\ i \cap F\ i$] *simp*: *PiE-Int*)
qed

lemma *prod-algebraE*:
assumes *A*: $A \in prod\text{-}algebra\ I\ M$
obtains $J\ E$ **where** $A = prod\text{-}emb\ I\ M\ J\ (PIE\ j{:}J.\ E\ j)$
finite J $J \neq \{\} \vee I = \{\}$ $J \subseteq I$ $\bigwedge i.\ i \in J \Longrightarrow E\ i \in sets\ (M\ i)$
using A **by** (*auto simp*: *prod-algebra-def*)

lemma *prod-algebraE-all*:
assumes *A*: $A \in prod\text{-}algebra\ I\ M$
obtains E **where** $A = Pi_E\ I\ E\ E \in (\Pi\ i{\in}I.\ sets\ (M\ i))$
proof −
from A **obtain** $E\ J$ **where** *A*: $A = prod\text{-}emb\ I\ M\ J\ (Pi_E\ J\ E)$
and *J*: $J \subseteq I$ **and** *E*: $E \in (\Pi\ i{\in}J.\ sets\ (M\ i))$
by (*auto simp*: *prod-algebra-def*)
from E **have** $\bigwedge i.\ i \in J \Longrightarrow E\ i \subseteq space\ (M\ i)$
using *sets.sets-into-space* **by** *auto*
then have $A = (\Pi_E\ i{\in}I.\ if\ i{\in}J\ then\ E\ i\ else\ space\ (M\ i))$
using A J **by** (*auto simp*: *prod-emb-PiE*)
moreover have $(\lambda i.\ if\ i{\in}J\ then\ E\ i\ else\ space\ (M\ i)) \in (\Pi\ i{\in}I.\ sets\ (M\ i))$
using *sets.top* E **by** *auto*
ultimately show *?thesis* **using** *that* **by** *auto*
qed

lemma *Int-stable-prod-algebra*: $Int\text{-}stable\ (prod\text{-}algebra\ I\ M)$
proof (*unfold Int-stable-def*, *safe*)
fix A **assume** $A \in prod\text{-}algebra\ I\ M$
from *prod-algebraE*[*OF this*] **guess** $J\ E$ **. note** $A = this$
fix B **assume** $B \in prod\text{-}algebra\ I\ M$
from *prod-algebraE*[*OF this*] **guess** $K\ F$ **. note** $B = this$
have $A \cap B = prod\text{-}emb\ I\ M\ (J \cup K)\ (\Pi_E\ i{\in}J \cup K.\ (if\ i \in J\ then\ E\ i\ else\ space\ (M\ i)) \cap$
$(if\ i \in K\ then\ F\ i\ else\ space\ (M\ i)))$
unfolding A B **using** *A*(2,3,4) *A*(5)[*THEN sets.sets-into-space*] *B*(2,3,4)
B(5)[*THEN sets.sets-into-space*]
apply (*subst* (1 2 3) *prod-emb-PiE*)
apply (*simp-all add*: *subset-eq PiE-Int*)
apply *blast*
apply (*intro PiE-cong*)
apply *auto*
done
also have $\ldots \in prod\text{-}algebra\ I\ M$
using A B **by** (*auto intro*!: *prod-algebraI*)
finally show $A \cap B \in prod\text{-}algebra\ I\ M$ **.**
qed

lemma *prod-algebra-mono*:
assumes *space*: $\bigwedge i.\ i \in I \Longrightarrow space\ (E\ i) = space\ (F\ i)$
assumes *sets*: $\bigwedge i.\ i \in I \Longrightarrow sets\ (E\ i) \subseteq sets\ (F\ i)$
shows *prod-algebra I E* $\subseteq$ *prod-algebra I F*
proof
fix *A* **assume** $A \in$ *prod-algebra I E*
then obtain *J G* **where** *J*: $J \neq \{\} \vee I = \{\}$ *finite J* $J \subseteq I$
and *A*: $A =$ *prod-emb I E J* $(\Pi_E\ i \in J.\ G\ i)$
and *G*: $\bigwedge i.\ i \in J \Longrightarrow G\ i \in sets\ (E\ i)$
by (*auto simp*: *prod-algebra-def*)
moreover
from *space* **have** $(\Pi_E\ i \in I.\ space\ (E\ i)) = (\Pi_E\ i \in I.\ space\ (F\ i))$
by (*rule PiE-cong*)
with *A* **have** $A =$ *prod-emb I F J* $(\Pi_E\ i \in J.\ G\ i)$
by (*simp add*: *prod-emb-def*)
moreover
from *sets G J* **have** $\bigwedge i.\ i \in J \Longrightarrow G\ i \in sets\ (F\ i)$
by *auto*
ultimately show $A \in$ *prod-algebra I F*
apply (*simp add*: *prod-algebra-def image-iff*)
apply (*intro exI*[*of* - *J*] *exI*[*of* - *G*] *conjI*)
apply *auto*
done
qed

lemma *prod-algebra-cong*:
assumes $I = J$ **and** *sets*: $(\bigwedge i.\ i \in I \Longrightarrow sets\ (M\ i) = sets\ (N\ i))$
shows *prod-algebra I M* = *prod-algebra J N*
proof −
have *space*: $\bigwedge i.\ i \in I \Longrightarrow space\ (M\ i) = space\ (N\ i)$
using *sets-eq-imp-space-eq*[*OF sets*] **by** *auto*
with *sets* **show** *?thesis* **unfolding** ‹$I = J$› **by** (*intro antisym prod-algebra-mono*) *auto*
qed

lemma *space-in-prod-algebra*:
$(\Pi_E\ i \in I.\ space\ (M\ i)) \in$ *prod-algebra I M*
proof *cases*
assume $I = \{\}$ **then show** *?thesis*
by (*auto simp add*: *prod-algebra-def image-iff prod-emb-def*)
next
assume $I \neq \{\}$
then obtain *i* **where** $i \in I$ **by** *auto*
then have $(\Pi_E\ i \in I.\ space\ (M\ i)) =$ *prod-emb I M* $\{i\}$ $(\Pi_E\ i \in \{i\}.\ space\ (M\ i))$
by (*auto simp*: *prod-emb-def*)
also have $\ldots \in$ *prod-algebra I M*
using ‹$i \in I$› **by** (*intro prod-algebraI*) *auto*
finally show *?thesis* **.**
qed

lemma *space-PiM*: *space* (Π_M *i*∈*I*. *M i*) = (Π_E *i*∈*I*. *space* (*M i*))
using *prod-algebra-sets-into-space* **unfolding** *PiM-def prod-algebra-def* **by** (*intro space-extend-measure*) *simp*

lemma *prod-emb-subset-PiM*[*simp*]: *prod-emb I M K X* ⊆ *space* (*PiM I M*)
by (*auto simp*: *prod-emb-def space-PiM*)

lemma *space-PiM-empty-iff*[*simp*]: *space* (*PiM I M*) = {} ⟷ (∃ *i*∈*I*. *space* (*M i*) = {})
by (*auto simp*: *space-PiM PiE-eq-empty-iff*)

lemma *undefined-in-PiM-empty*[*simp*]: (λ*x*. *undefined*) ∈ *space* (*PiM* {} *M*)
by (*auto simp*: *space-PiM*)

lemma *sets-PiM*: *sets* (Π_M *i*∈*I*. *M i*) = *sigma-sets* (Π_E *i*∈*I*. *space* (*M i*)) (*prod-algebra I M*)
using *prod-algebra-sets-into-space* **unfolding** *PiM-def prod-algebra-def* **by** (*intro sets-extend-measure*) *simp*

lemma *sets-PiM-single*: *sets* (*PiM I M*) =
sigma-sets (Π_E *i*∈*I*. *space* (*M i*)) {{*f*∈Π_E *i*∈*I*. *space* (*M i*). *f i* ∈ *A*} | *i A*. *i* ∈ *I* ∧ *A* ∈ *sets* (*M i*)}
(**is** - = *sigma-sets* ?Ω ?*R*)
unfolding *sets-PiM*
proof (*rule sigma-sets-eqI*)
interpret *R*: *sigma-algebra* ?Ω *sigma-sets* ?Ω ?*R* **by** (*rule sigma-algebra-sigma-sets*) *auto*
fix *A* **assume** *A* ∈ *prod-algebra I M*
from *prod-algebraE*[*OF this*] **guess** *J X* **. note** *X* = *this*
show *A* ∈ *sigma-sets* ?Ω ?*R*
proof *cases*
assume *I* = {}
with *X* **have** *A* = {λ*x*. *undefined*} **by** (*auto simp*: *prod-emb-def*)
with ‹*I* = {}› **show** *?thesis* **by** (*auto intro*!: *sigma-sets-top*)
next
assume *I* ≠ {}
with *X* **have** *A* = (⋂*j*∈*J*. {*f*∈(Π_E *i*∈*I*. *space* (*M i*)). *f j* ∈ *X j*})
by (*auto simp*: *prod-emb-def*)
also have ... ∈ *sigma-sets* ?Ω ?*R*
using *X* ‹*I* ≠ {}› **by** (*intro R.finite-INT sigma-sets.Basic*) *auto*
finally show *A* ∈ *sigma-sets* ?Ω ?*R* **.**
qed
next
fix *A* **assume** *A* ∈ ?*R*
then obtain *i B* **where** *A*: *A* = {*f*∈Π_E *i*∈*I*. *space* (*M i*). *f i* ∈ *B*} *i* ∈ *I B* ∈ *sets* (*M i*)
by *auto*
then have *A* = *prod-emb I M* {*i*} (Π_E *i*∈{*i*}. *B*)

by (*auto simp*: *prod-emb-def*)
also have ... ∈ *sigma-sets ?Ω* (*prod-algebra I M*)
using *A* **by** (*intro sigma-sets.Basic prod-algebraI*) *auto*
finally show *A* ∈ *sigma-sets ?Ω* (*prod-algebra I M*) **.**
qed

lemma *sets-PiM-eq-proj*:
I ≠ {} ⟹ *sets* (*PiM I M*) = *sets* (⨆$_\sigma$ *i*∈*I*. *vimage-algebra* (Π$_E$ *i*∈*I*. *space* (*M i*)) (λ*x*. *x i*) (*M i*))
apply (*simp add*: *sets-PiM-single sets-Sup-sigma*)
apply (*subst SUP-cong*[*OF refl*])
apply (*rule sets-vimage-algebra2*)
apply *auto* []
apply (*auto intro*!: *arg-cong2*[**where** *f*=*sigma-sets*])
done

lemma
shows *space-PiM-empty*: *space* (*Pi*$_M$ {} *M*) = {λ*k*. *undefined*}
and *sets-PiM-empty*: *sets* (*Pi*$_M$ {} *M*) = { {}, {λ*k*. *undefined*} }
by (*simp-all add*: *space-PiM sets-PiM-single image-constant sigma-sets-empty-eq*)

lemma *sets-PiM-sigma*:
assumes Ω-*cover*: ⋀*i*. *i* ∈ *I* ⟹ ∃ *S*⊆*E i*. *countable S* ∧ Ω *i* = ⋃ *S*
assumes *E*: ⋀*i*. *i* ∈ *I* ⟹ *E i* ⊆ *Pow* (Ω *i*)
assumes *J*: ⋀*j*. *j* ∈ *J* ⟹ *finite j* ⋃ *J* = *I*
defines *P* ≡ {{*f*∈(Π$_E$ *i*∈*I*. Ω *i*). ∀ *i*∈*j*. *f i* ∈ *A i*} | *A j*. *j* ∈ *J* ∧ *A* ∈ *Pi j E*}
shows *sets* (Π$_M$ *i*∈*I*. *sigma* (Ω *i*) (*E i*)) = *sets* (*sigma* (Π$_E$ *i*∈*I*. Ω *i*) *P*)
proof *cases*
assume *I* = {}
with ‹⋃ *J* = *I*› **have** *P* = {{λ-. *undefined*}} ∨ *P* = {}
by (*auto simp*: *P-def*)
with ‹*I* = {}› **show** *?thesis*
by (*auto simp add*: *sets-PiM-empty sigma-sets-empty-eq*)
next
let *?F* = λ*i*. {(λ*x*. *x i*) −‘ *A* ∩ *Pi*$_E$ *I* Ω |*A*. *A* ∈ *E i*}
assume *I* ≠ {}
then have *sets* (*Pi*$_M$ *I* (λ*i*. *sigma* (Ω *i*) (*E i*))) =
sets (⨆$_\sigma$ *i*∈*I*. *vimage-algebra* (Π$_E$ *i*∈*I*. Ω *i*) (λ*x*. *x i*) (*sigma* (Ω *i*) (*E i*)))
by (*subst sets-PiM-eq-proj*) (*auto simp*: *space-measure-of-conv*)
also have ... = *sets* (⨆$_\sigma$ *i*∈*I*. *sigma* (*Pi*$_E$ *I* Ω) (*?F i*))
using *E* **by** (*intro SUP-sigma-cong arg-cong*[**where** *f*=*sets*] *vimage-algebra-sigma*) *auto*
also have ... = *sets* (*sigma* (*Pi*$_E$ *I* Ω) (⋃ *i*∈*I*. *?F i*))
using ‹*I* ≠ {}› **by** (*intro arg-cong*[**where** *f*=*sets*] *SUP-sigma-sigma*) *auto*
also have ... = *sets* (*sigma* (*Pi*$_E$ *I* Ω) *P*)
proof (*intro arg-cong*[**where** *f*=*sets*] *sigma-eqI sigma-sets-eqI*)
show (⋃ *i*∈*I*. *?F i*) ⊆ *Pow* (*Pi*$_E$ *I* Ω) *P* ⊆ *Pow* (*Pi*$_E$ *I* Ω)
by (*auto simp*: *P-def*)
next

interpret *P*: *sigma-algebra* $\Pi_E\ i{\in}I.\ \Omega\ i$ *sigma-sets* $(\Pi_E\ i{\in}I.\ \Omega\ i)\ P$
by (*auto intro*!: *sigma-algebra-sigma-sets simp*: *P-def*)

fix *Z* **assume** $Z \in (\bigcup i{\in}I.\ ?F\ i)$
then obtain *i A* **where** *i*: $i \in I\ A \in E\ i$ **and** *Z-def*: $Z = (\lambda x.\ x\ i) -` A \cap Pi_E\ I\ \Omega$
by *auto*
from ‹$i \in I$› *J* **obtain** *j* **where** *j*: $i \in j\ j \in J\ j \subseteq I$ *finite j*
by *auto*
obtain *S* **where** *S*: $\bigwedge i.\ i \in j \Longrightarrow S\ i \subseteq E\ i\ \bigwedge i.\ i \in j \Longrightarrow$ *countable* $(S\ i)$ $\bigwedge i.\ i \in j \Longrightarrow \Omega\ i = \bigcup(S\ i)$
by (*metis subset-eq* Ω*-cover* ‹$j \subseteq I$›)
def $A' \equiv \lambda n.\ n(i := A)$
then have *A′-i*: $\bigwedge n.\ A'\ n\ i = A$
by *simp*
{ fix *n* **assume** $n \in Pi_E\ (j - \{i\})\ S$
then have $A'\ n \in Pi\ j\ E$
unfolding *PiE-def Pi-def* **using** *S*(*1*) **by** (*auto simp*: *A′-def* ‹$A \in E\ i$›)
with ‹$j \in J$› **have** $\{f \in Pi_E\ I\ \Omega.\ \forall i{\in}j.\ f\ i \in A'\ n\ i\} \in P$
by (*auto simp*: *P-def*) **}**
note *A′-in-P* = *this*

{ fix *x* **assume** $x\ i \in A\ x \in Pi_E\ I\ \Omega$
with *S*(*3*) ‹$j \subseteq I$› **have** $\forall i{\in}j.\ \exists s{\in}S\ i.\ x\ i \in s$
by (*auto simp*: *PiE-def Pi-def*)
then obtain *s* **where** *s*: $\bigwedge i.\ i \in j \Longrightarrow s\ i \in S\ i\ \bigwedge i.\ i \in j \Longrightarrow x\ i \in s\ i$
by *metis*
with ‹$x\ i \in A$› **have** $\exists n{\in}PiE\ (j{-}\{i\})\ S.\ \forall i{\in}j.\ x\ i \in A'\ n\ i$
by (*intro bexI*[*of* - *restrict* $(s(i := A))\ (j{-}\{i\})$]) (*auto simp*: *A′-def split*: *if-splits*) **}**
then have $Z = (\bigcup n{\in}PiE\ (j{-}\{i\})\ S.\ \{f{\in}(\Pi_E\ i{\in}I.\ \Omega\ i).\ \forall i{\in}j.\ f\ i \in A'\ n\ i\})$
unfolding *Z-def*
by (*auto simp add*: *set-eq-iff ball-conj-distrib* ‹$i{\in}j$› *A′-i dest*: *bspec*[*OF* - ‹$i{\in}j$›] *cong*: *conj-cong*)
also have $\ldots \in$ *sigma-sets* $(\Pi_E\ i{\in}I.\ \Omega\ i)\ P$
using ‹*finite j*› *S*(*2*)
by (*intro P.countable-UN′ countable-PiE*) (*simp-all add*: *image-subset-iff A′-in-P*)
finally show $Z \in$ *sigma-sets* $(\Pi_E\ i{\in}I.\ \Omega\ i)\ P$ **.**
next
interpret *F*: *sigma-algebra* $\Pi_E\ i{\in}I.\ \Omega\ i$ *sigma-sets* $(\Pi_E\ i{\in}I.\ \Omega\ i)$ $(\bigcup i{\in}I.\ ?F\ i)$
by (*auto intro*!: *sigma-algebra-sigma-sets*)

fix *b* **assume** $b \in P$
then obtain *A j* **where** *b*: $b = \{f{\in}(\Pi_E\ i{\in}I.\ \Omega\ i).\ \forall i{\in}j.\ f\ i \in A\ i\}\ j \in J\ A \in Pi\ j\ E$
by (*auto simp*: *P-def*)
show $b \in$ *sigma-sets* $(Pi_E\ I\ \Omega)$ $(\bigcup i{\in}I.\ ?F\ i)$

proof *cases*
assume $j = \{\}$
with b **have** $b = (\Pi_E\ i{\in}I.\ \Omega\ i)$
by *auto*
then show *?thesis*
by *blast*
next
assume $j \neq \{\}$
with J $b(2,3)$ **have** *eq*: $b = (\bigcap i{\in}j.\ ((\lambda x.\ x\ i)\ -`\ A\ i \cap Pi_E\ I\ \Omega))$
unfolding $b(1)$
by (*auto simp*: *PiE-def Pi-def*)
show *?thesis*
unfolding *eq* **using** ‹$A \in Pi\ j\ E$› ‹$j \in J$› $J(2)$
by (*intro F.finite-INT J* ‹$j \in J$› ‹$j \neq \{\}$› *sigma-sets.Basic*) *blast*
qed
qed
finally show *?thesis* **.**
qed

lemma *sets-PiM-in-sets*:
assumes *space*: *space* $N = (\Pi_E\ i{\in}I.\ space\ (M\ i))$
assumes *sets*: $\bigwedge i\ A.\ i \in I \Longrightarrow A \in sets\ (M\ i) \Longrightarrow \{x{\in}space\ N.\ x\ i \in A\} \in sets\ N$
shows *sets* $(\Pi_M\ i \in I.\ M\ i) \subseteq sets\ N$
unfolding *sets-PiM-single space*[*symmetric*]
by (*intro sets.sigma-sets-subset subsetI*) (*auto intro*: *sets*)

lemma *sets-PiM-cong*[*measurable-cong*]:
assumes $I = J$ $\bigwedge i.\ i \in J \Longrightarrow sets\ (M\ i) = sets\ (N\ i)$ **shows** *sets* $(PiM\ I\ M) = sets\ (PiM\ J\ N)$
using *assms sets-eq-imp-space-eq*[*OF assms*(*2*)] **by** (*simp add*: *sets-PiM-single cong*: *PiE-cong conj-cong*)

lemma *sets-PiM-I*:
assumes *finite* J $J \subseteq I$ $\forall i{\in}J.\ E\ i \in sets\ (M\ i)$
shows *prod-emb I M J* $(PIE\ j{:}J.\ E\ j) \in sets\ (\Pi_M\ i{\in}I.\ M\ i)$
proof *cases*
assume $J = \{\}$
then have *prod-emb I M J* $(PIE\ j{:}J.\ E\ j) = (PIE\ j{:}I.\ space\ (M\ j))$
by (*auto simp*: *prod-emb-def*)
then show *?thesis*
by (*auto simp add*: *sets-PiM intro*!: *sigma-sets-top*)
next
assume $J \neq \{\}$ **with** *assms* **show** *?thesis*
by (*force simp add*: *sets-PiM prod-algebra-def*)
qed

lemma *measurable-PiM*:
assumes *space*: $f \in space\ N \to (\Pi_E\ i{\in}I.\ space\ (M\ i))$

assumes *sets*: $\bigwedge X\ J.\ J \neq \{\} \vee I = \{\} \Longrightarrow$ *finite* $J \Longrightarrow J \subseteq I \Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow X\ i \in$ *sets* $(M\ i)) \Longrightarrow$
$f\ -`$ *prod-emb* $I\ M\ J\ (Pi_E\ J\ X) \cap$ *space* $N \in$ *sets* N
shows $f \in$ *measurable* $N\ (PiM\ I\ M)$
using *sets-PiM prod-algebra-sets-into-space space*
proof (*rule measurable-sigma-sets*)
fix A **assume** $A \in$ *prod-algebra* $I\ M$
from *prod-algebraE*[*OF this*] **guess** $J\ X$ **.**
with *sets*[*of* $J\ X$] **show** $f\ -`\ A \cap$ *space* $N \in$ *sets* N **by** *auto*
qed

lemma *measurable-PiM-Collect*:
assumes *space*: $f \in$ *space* $N \to (\Pi_E\ i \in I.$ *space* $(M\ i))$
assumes *sets*: $\bigwedge X\ J.\ J \neq \{\} \vee I = \{\} \Longrightarrow$ *finite* $J \Longrightarrow J \subseteq I \Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow X\ i \in$ *sets* $(M\ i)) \Longrightarrow$
$\{\omega \in$ *space* $N.\ \forall i \in J.\ f\ \omega\ i \in X\ i\} \in$ *sets* N
shows $f \in$ *measurable* $N\ (PiM\ I\ M)$
using *sets-PiM prod-algebra-sets-into-space space*
proof (*rule measurable-sigma-sets*)
fix A **assume** $A \in$ *prod-algebra* $I\ M$
from *prod-algebraE*[*OF this*] **guess** $J\ X$ **. note** $X = this$
then have $f\ -`\ A \cap$ *space* $N = \{\omega \in$ *space* $N.\ \forall i \in J.\ f\ \omega\ i \in X\ i\}$
using *space* **by** (*auto simp*: *prod-emb-def del*: *PiE-I*)
also have $\ldots \in$ *sets* N **using** $X(3,2,4,5)$ **by** (*rule sets*)
finally show $f\ -`\ A \cap$ *space* $N \in$ *sets* N **.**
qed

lemma *measurable-PiM-single*:
assumes *space*: $f \in$ *space* $N \to (\Pi_E\ i \in I.$ *space* $(M\ i))$
assumes *sets*: $\bigwedge A\ i.\ i \in I \Longrightarrow A \in$ *sets* $(M\ i) \Longrightarrow \{\omega \in$ *space* $N.\ f\ \omega\ i \in A\} \in$ *sets* N
shows $f \in$ *measurable* $N\ (PiM\ I\ M)$
using *sets-PiM-single*
proof (*rule measurable-sigma-sets*)
fix A **assume** $A \in \{\{f \in \Pi_E\ i \in I.$ *space* $(M\ i).\ f\ i \in A\}\ |i\ A.\ i \in I \wedge A \in$ *sets* $(M\ i)\}$
then obtain $B\ i$ **where** $A = \{f \in \Pi_E\ i \in I.$ *space* $(M\ i).\ f\ i \in B\}$ **and** B: $i \in I$ $B \in$ *sets* $(M\ i)$
by *auto*
with *space* **have** $f\ -`\ A \cap$ *space* $N = \{\omega \in$ *space* $N.\ f\ \omega\ i \in B\}$ **by** *auto*
also have $\ldots \in$ *sets* N **using** B **by** (*rule sets*)
finally show $f\ -`\ A \cap$ *space* $N \in$ *sets* N **.**
qed (*auto simp*: *space*)

lemma *measurable-PiM-single'*:
assumes f: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in$ *measurable* $N\ (M\ i)$
and $(\lambda\omega\ i.\ f\ i\ \omega) \in$ *space* $N \to (\Pi_E\ i \in I.$ *space* $(M\ i))$
shows $(\lambda\omega\ i.\ f\ i\ \omega) \in$ *measurable* $N\ (Pi_M\ I\ M)$
proof (*rule measurable-PiM-single*)

fix A i **assume** A: $i \in I$ $A \in sets\ (M\ i)$
then have $\{\omega \in space\ N.\ f\ i\ \omega \in A\} = f\ i\ -`\ A \cap space\ N$
by *auto*
then show $\{\omega \in space\ N.\ f\ i\ \omega \in A\} \in sets\ N$
using A f **by** (*auto intro*!: *measurable-sets*)
qed *fact*

lemma *sets-PiM-I-finite*[*measurable*]:
assumes *finite* I **and** *sets*: $(\bigwedge i.\ i \in I \Longrightarrow E\ i \in sets\ (M\ i))$
shows $(PIE\ j{:}I.\ E\ j) \in sets\ (\Pi_M\ i{\in}I.\ M\ i)$
using *sets-PiM-I*[*of I I E M*] *sets.sets-into-space*[*OF sets*] ‹*finite I*› *sets* **by** *auto*

lemma *measurable-component-singleton*[*measurable* (*raw*)]:
assumes $i \in I$ **shows** $(\lambda x.\ x\ i) \in measurable\ (Pi_M\ I\ M)\ (M\ i)$
proof (*unfold measurable-def*, *intro CollectI conjI ballI*)
fix A **assume** $A \in sets\ (M\ i)$
then have $(\lambda x.\ x\ i)\ -`\ A \cap space\ (Pi_M\ I\ M) = prod\text{-}emb\ I\ M\ \{i\}\ (\Pi_E\ j{\in}\{i\}.\ A)$
using *sets.sets-into-space* ‹$i \in I$›
by (*fastforce dest*: *Pi-mem simp*: *prod-emb-def space-PiM split*: *if-split-asm*)
then show $(\lambda x.\ x\ i)\ -`\ A \cap space\ (Pi_M\ I\ M) \in sets\ (Pi_M\ I\ M)$
using ‹$A \in sets\ (M\ i)$› ‹$i \in I$› **by** (*auto intro*!: *sets-PiM-I*)
qed (*insert* ‹$i \in I$›, *auto simp*: *space-PiM*)

lemma *measurable-component-singleton'*[*measurable-dest*]:
assumes f: $f \in measurable\ N\ (Pi_M\ I\ M)$
assumes g: $g \in measurable\ L\ N$
assumes i: $i \in I$
shows $(\lambda x.\ (f\ (g\ x))\ i) \in measurable\ L\ (M\ i)$
using *measurable-compose*[*OF measurable-compose*[*OF g f*] *measurable-component-singleton*, *OF i*] **.**

lemma *measurable-PiM-component-rev*:
$i \in I \Longrightarrow f \in measurable\ (M\ i)\ N \Longrightarrow (\lambda x.\ f\ (x\ i)) \in measurable\ (PiM\ I\ M)\ N$
by *simp*

lemma *measurable-case-nat*[*measurable* (*raw*)]:
assumes [*measurable* (*raw*)]: $i = 0 \Longrightarrow f \in measurable\ M\ N$
$\bigwedge j.\ i = Suc\ j \Longrightarrow (\lambda x.\ g\ x\ j) \in measurable\ M\ N$
shows $(\lambda x.\ case\text{-}nat\ (f\ x)\ (g\ x)\ i) \in measurable\ M\ N$
by (*cases i*) *simp-all*

lemma *measurable-case-nat'*[*measurable* (*raw*)]:
assumes *fg*[*measurable*]: $f \in measurable\ N\ M\ g \in measurable\ N\ (\Pi_M\ i{\in}UNIV.\ M)$
shows $(\lambda x.\ case\text{-}nat\ (f\ x)\ (g\ x)) \in measurable\ N\ (\Pi_M\ i{\in}UNIV.\ M)$
using *fg*[*THEN measurable-space*]
by (*auto intro*!: *measurable-PiM-single' simp add*: *space-PiM PiE-iff split*: *nat.split*)

lemma *measurable-add-dim*[*measurable*]:
(λ(*f*, *y*). *f*(*i* := *y*)) ∈ *measurable* (*Pi*$_M$ *I M* ⨂$_M$ *M i*) (*Pi*$_M$ (*insert i I*) *M*)
(**is** *?f* ∈ *measurable ?P ?I*)
proof (*rule measurable-PiM-single*)
fix *j A* **assume** *j*: *j* ∈ *insert i I* **and** *A*: *A* ∈ *sets* (*M j*)
have {ω ∈ *space ?P*. (λ(*f*, *x*). *fun-upd f i x*) ω *j* ∈ *A*} =
(*if j* = *i then space* (*Pi*$_M$ *I M*) × *A else* ((λ*x*. *x j*) ∘ *fst*) −‘ *A* ∩ *space ?P*)
using *sets.sets-into-space*[*OF A*] **by** (*auto simp add*: *space-pair-measure space-PiM*)
also have ... ∈ *sets ?P*
using *A j*
by (*auto intro*!: *measurable-sets*[*OF measurable-comp*, *OF - measurable-component-singleton*])
finally show {ω ∈ *space ?P*. *case-prod* (λ*f*. *fun-upd f i*) ω *j* ∈ *A*} ∈ *sets ?P* .
qed (*auto simp*: *space-pair-measure space-PiM PiE-def*)

lemma *measurable-fun-upd*:
assumes *I*: *I* = *J* ∪ {*i*}
assumes *f*[*measurable*]: *f* ∈ *measurable N* (*PiM J M*)
assumes *h*[*measurable*]: *h* ∈ *measurable N* (*M i*)
shows (λ*x*. (*f x*) (*i* := *h x*)) ∈ *measurable N* (*PiM I M*)
proof (*intro measurable-PiM-single'*)
fix *j* **assume** *j* ∈ *I* **then show** (λω. ((*f* ω)(*i* := *h* ω)) *j*) ∈ *measurable N* (*M j*)
unfolding *I* **by** (*cases j* = *i*) *auto*
next
show (λ*x*. (*f x*)(*i* := *h x*)) ∈ *space N* → (Π$_E$ *i*∈*I*. *space* (*M i*))
using *I f*[*THEN measurable-space*] *h*[*THEN measurable-space*]
by (*auto simp*: *space-PiM PiE-iff extensional-def*)
qed

lemma *measurable-component-update*:
x ∈ *space* (*Pi*$_M$ *I M*) ⟹ *i* ∉ *I* ⟹ (λ*v*. *x*(*i* := *v*)) ∈ *measurable* (*M i*) (*Pi*$_M$ (*insert i I*) *M*)
by *simp*

lemma *measurable-merge*[*measurable*]:
merge I J ∈ *measurable* (*Pi*$_M$ *I M* ⨂$_M$ *Pi*$_M$ *J M*) (*Pi*$_M$ (*I* ∪ *J*) *M*)
(**is** *?f* ∈ *measurable ?P ?U*)
proof (*rule measurable-PiM-single*)
fix *i A* **assume** *A*: *A* ∈ *sets* (*M i*) *i* ∈ *I* ∪ *J*
then have {ω ∈ *space ?P*. *merge I J* ω *i* ∈ *A*} =
(*if i* ∈ *I then* ((λ*x*. *x i*) ∘ *fst*) −‘ *A* ∩ *space ?P else* ((λ*x*. *x i*) ∘ *snd*) −‘ *A* ∩ *space ?P*)
by (*auto simp*: *merge-def*)
also have ... ∈ *sets ?P*
using *A*
by (*auto intro*!: *measurable-sets*[*OF measurable-comp*, *OF - measurable-component-singleton*])
finally show {ω ∈ *space ?P*. *merge I J* ω *i* ∈ *A*} ∈ *sets ?P* .
qed (*auto simp*: *space-pair-measure space-PiM PiE-iff merge-def extensional-def*)

lemma *measurable-restrict*[*measurable* (*raw*)]:

assumes X: $\bigwedge i.\ i \in I \Longrightarrow X\ i \in \mathit{measurable}\ N\ (M\ i)$
shows $(\lambda x.\ \lambda i{\in}I.\ X\ i\ x) \in \mathit{measurable}\ N\ (\mathit{Pi}_M\ I\ M)$
proof (*rule measurable-PiM-single*)
fix $A\ i$ **assume** A: $i \in I\ A \in \mathit{sets}\ (M\ i)$
then have $\{\omega \in \mathit{space}\ N.\ (\lambda i{\in}I.\ X\ i\ \omega)\ i \in A\} = X\ i\ -`\ A \cap \mathit{space}\ N$
by *auto*
then show $\{\omega \in \mathit{space}\ N.\ (\lambda i{\in}I.\ X\ i\ \omega)\ i \in A\} \in \mathit{sets}\ N$
using $A\ X$ **by** (*auto intro*!: *measurable-sets*)
qed (*insert* X, *auto simp add*: *PiE-def dest*: *measurable-space*)

lemma *measurable-abs-UNIV*:
$(\bigwedge n.\ (\lambda\omega.\ f\ n\ \omega) \in \mathit{measurable}\ M\ (N\ n)) \Longrightarrow (\lambda\omega\ n.\ f\ n\ \omega) \in \mathit{measurable}\ M\ (\mathit{PiM}\ \mathit{UNIV}\ N)$
by (*intro measurable-PiM-single*) (*auto dest*: *measurable-space*)

lemma *measurable-restrict-subset*: $J \subseteq L \Longrightarrow (\lambda f.\ \mathit{restrict}\ f\ J) \in \mathit{measurable}\ (\mathit{Pi}_M\ L\ M)\ (\mathit{Pi}_M\ J\ M)$
by (*intro measurable-restrict measurable-component-singleton*) *auto*

lemma *measurable-restrict-subset'*:
assumes $J \subseteq L\ \bigwedge x.\ x \in J \Longrightarrow \mathit{sets}\ (M\ x) = \mathit{sets}\ (N\ x)$
shows $(\lambda f.\ \mathit{restrict}\ f\ J) \in \mathit{measurable}\ (\mathit{Pi}_M\ L\ M)\ (\mathit{Pi}_M\ J\ N)$
proof−
from *assms*(*1*) **have** $(\lambda f.\ \mathit{restrict}\ f\ J) \in \mathit{measurable}\ (\mathit{Pi}_M\ L\ M)\ (\mathit{Pi}_M\ J\ M)$
by (*rule measurable-restrict-subset*)
also from *assms*(*2*) **have** $\mathit{measurable}\ (\mathit{Pi}_M\ L\ M)\ (\mathit{Pi}_M\ J\ M) = \mathit{measurable}\ (\mathit{Pi}_M\ L\ M)\ (\mathit{Pi}_M\ J\ N)$
by (*intro sets-PiM-cong measurable-cong-sets*) *simp-all*
finally show *?thesis* **.**
qed

lemma *measurable-prod-emb*[*intro*, *simp*]:
$J \subseteq L \Longrightarrow X \in \mathit{sets}\ (\mathit{Pi}_M\ J\ M) \Longrightarrow \mathit{prod\text{-}emb}\ L\ M\ J\ X \in \mathit{sets}\ (\mathit{Pi}_M\ L\ M)$
unfolding *prod-emb-def space-PiM*[*symmetric*]
by (*auto intro*!: *measurable-sets measurable-restrict measurable-component-singleton*)

lemma *merge-in-prod-emb*:
assumes $y \in \mathit{space}\ (\mathit{PiM}\ I\ M)\ x \in X$ **and** X: $X \in \mathit{sets}\ (\mathit{Pi}_M\ J\ M)$ **and** $J \subseteq I$
shows $\mathit{merge}\ J\ I\ (x, y) \in \mathit{prod\text{-}emb}\ I\ M\ J\ X$
using *assms sets.sets-into-space*[*OF* X]
by (*simp add*: *merge-def prod-emb-def subset-eq space-PiM PiE-def extensional-restrict Pi-iff*
cong: *if-cong restrict-cong*)
(*simp add*: *extensional-def*)

lemma *prod-emb-eq-emptyD*:
assumes J: $J \subseteq I$ **and** *ne*: $\mathit{space}\ (\mathit{PiM}\ I\ M) \neq \{\}$ **and** X: $X \in \mathit{sets}\ (\mathit{Pi}_M\ J\ M)$
and $*$: $\mathit{prod\text{-}emb}\ I\ M\ J\ X = \{\}$

shows $X = \{\}$
proof *safe*
fix x **assume** $x \in X$
obtain ω **where** $\omega \in space\ (PiM\ I\ M)$
using *ne* **by** *blast*
from *merge-in-prod-emb*[*OF this* ‹$x \in X$› $X\ J$] $*$ **show** $x \in \{\}$ **by** *auto*
qed

lemma *sets-in-Pi-aux*:
$finite\ I \Longrightarrow (\bigwedge j.\ j \in I \Longrightarrow \{x \in space\ (M\ j).\ x \in F\ j\} \in sets\ (M\ j)) \Longrightarrow$
$\{x \in space\ (PiM\ I\ M).\ x \in Pi\ I\ F\} \in sets\ (PiM\ I\ M)$
by (*simp add*: *subset-eq Pi-iff*)

lemma *sets-in-Pi*[*measurable* (*raw*)]:
$finite\ I \Longrightarrow f \in measurable\ N\ (PiM\ I\ M) \Longrightarrow$
$(\bigwedge j.\ j \in I \Longrightarrow \{x \in space\ (M\ j).\ x \in F\ j\} \in sets\ (M\ j)) \Longrightarrow$
$Measurable.pred\ N\ (\lambda x.\ f\ x \in Pi\ I\ F)$
unfolding *pred-def*
by (*rule measurable-sets-Collect*[*of f N PiM I M*, *OF - sets-in-Pi-aux*]) *auto*

lemma *sets-in-extensional-aux*:
$\{x \in space\ (PiM\ I\ M).\ x \in extensional\ I\} \in sets\ (PiM\ I\ M)$
proof −
have $\{x \in space\ (PiM\ I\ M).\ x \in extensional\ I\} = space\ (PiM\ I\ M)$
by (*auto simp add*: *extensional-def space-PiM*)
then show *?thesis* **by** *simp*
qed

lemma *sets-in-extensional*[*measurable* (*raw*)]:
$f \in measurable\ N\ (PiM\ I\ M) \Longrightarrow Measurable.pred\ N\ (\lambda x.\ f\ x \in extensional\ I)$
unfolding *pred-def*
by (*rule measurable-sets-Collect*[*of f N PiM I M*, *OF - sets-in-extensional-aux*])
auto

lemma *sets-PiM-I-countable*:
assumes *I*: *countable I* **and** *E*: $\bigwedge i.\ i \in I \Longrightarrow E\ i \in sets\ (M\ i)$ **shows** $Pi_E\ I\ E \in sets\ (Pi_M\ I\ M)$
proof *cases*
assume $I \neq \{\}$
then have $PiE\ I\ E = (\bigcap i \in I.\ prod\text{-}emb\ I\ M\ \{i\}\ (PiE\ \{i\}\ E))$
using *E*[*THEN sets.sets-into-space*] **by** (*auto simp*: *PiE-iff prod-emb-def fun-eq-iff*)
also have $\ldots \in sets\ (PiM\ I\ M)$
using *I* ‹$I \neq \{\}$› **by** (*safe intro*!: *sets.countable-INT'' measurable-prod-emb sets-PiM-I-finite E*)
finally show *?thesis* **.**
qed (*simp add*: *sets-PiM-empty*)

lemma *sets-PiM-D-countable*:
assumes *A*: $A \in PiM\ I\ M$

shows $\exists J{\subseteq}I.\ \exists X{\in}PiM\ J\ M.\ countable\ J \wedge A = prod\text{-}emb\ I\ M\ J\ X$
using $A[unfolded\ sets\text{-}PiM\text{-}single]$
proof *induction*
case $(Basic\ A)$
then obtain $i\ X$ **where** $*$: $i \in I\ X \in sets\ (M\ i)$ **and** $A = \{f \in \Pi_E\ i{\in}I.\ space\ (M\ i).\ f\ i \in X\}$
by *auto*
then have A: $A = prod\text{-}emb\ I\ M\ \{i\}\ (\Pi_E\ \text{-}{\in}\{i\}.\ X)$
by $(auto\ simp{:}\ prod\text{-}emb\text{-}def)$
then show *?case*
by $(intro\ exI[of\ \text{-}\ \{i\}]\ conjI\ bexI[of\ \text{-}\ \Pi_E\ \text{-}{\in}\{i\}.\ X])$
$(auto\ intro{:}\ countable\text{-}finite * sets\text{-}PiM\text{-}I\text{-}finite)$
next
case *Empty* **then show** *?case*
by $(intro\ exI[of\ \text{-}\ \{\}]\ conjI\ bexI[of\ \text{-}\ \{\}])\ auto$
next
case $(Compl\ A)$
then obtain $J\ X$ **where** $J \subseteq I\ X \in sets\ (Pi_M\ J\ M)\ countable\ J\ A = prod\text{-}emb\ I\ M\ J\ X$
by *auto*
then show *?case*
by $(intro\ exI[of\ \text{-}\ J]\ bexI[of\ \text{-}\ space\ (PiM\ J\ M) - X]\ conjI)$
$(auto\ simp\ add{:}\ space\text{-}PiM\ prod\text{-}emb\text{-}PiE\ intro!{:}\ sets\text{-}PiM\text{-}I\text{-}countable)$
next
case $(Union\ K)$
obtain $J\ X$ **where** J: $\bigwedge i.\ J\ i \subseteq I\ \bigwedge i.\ countable\ (J\ i)$ **and** X: $\bigwedge i.\ X\ i \in sets\ (Pi_M\ (J\ i)\ M)$
and K: $\bigwedge i.\ K\ i = prod\text{-}emb\ I\ M\ (J\ i)\ (X\ i)$
by $(metis\ Union.IH)$
show *?case*
proof $(intro\ exI[of\ \text{-}\ \bigcup i.\ J\ i]\ bexI[of\ \text{-}\ \bigcup i.\ prod\text{-}emb\ (\bigcup i.\ J\ i)\ M\ (J\ i)\ (X\ i)]\ conjI)$
show $(\bigcup i.\ J\ i) \subseteq I\ countable\ (\bigcup i.\ J\ i)$ **using** J **by** *auto*
with J **show** $UNION\ UNIV\ K = prod\text{-}emb\ I\ M\ (\bigcup i.\ J\ i)\ (\bigcup i.\ prod\text{-}emb\ (\bigcup i.\ J\ i)\ M\ (J\ i)\ (X\ i))$
by $(simp\ add{:}\ K[abs\text{-}def]\ SUP\text{-}upper)$
qed$(auto\ intro{:}\ X)$
qed

lemma *measure-eqI-PiM-finite*:
assumes [*simp*]: *finite I sets P = PiM I M sets Q = PiM I M*
assumes *eq*: $\bigwedge A.\ (\bigwedge i.\ i \in I \Longrightarrow A\ i \in sets\ (M\ i)) \Longrightarrow P\ (Pi_E\ I\ A) = Q\ (Pi_E\ I\ A)$
assumes A: *range A* $\subseteq$ *prod-algebra I M* $(\bigcup i.\ A\ i) = space\ (PiM\ I\ M)\ \bigwedge i{::}nat.\ P\ (A\ i) \neq \infty$
shows $P = Q$
proof $(rule\ measure\text{-}eqI\text{-}generator\text{-}eq[OF\ Int\text{-}stable\text{-}prod\text{-}algebra\ prod\text{-}algebra\text{-}sets\text{-}into\text{-}space])$
show *range A* $\subseteq$ *prod-algebra I M* $(\bigcup i.\ A\ i) = (\Pi_E\ i{\in}I.\ space\ (M\ i))\ \bigwedge i.\ P\ (A\ i) \neq \infty$

unfolding *space-PiM*[*symmetric*] **by** *fact+*
fix *X* **assume** *X* ∈ *prod-algebra I M*
then obtain *J E* **where** *X*: *X* = *prod-emb I M J* (*PIE j*:*J*. *E j*)
and *J*: *finite J J* ⊆ *I* ⋀*j*. *j* ∈ *J* ⟹ *E j* ∈ *sets* (*M j*)
by (*force elim*!: *prod-algebraE*)
then show *emeasure P X* = *emeasure Q X*
unfolding *X* **by** (*subst* (*1 2*) *prod-emb-Pi*) (*auto simp*: *eq*)
qed (*simp-all add*: *sets-PiM*)

lemma *measure-eqI-PiM-infinite*:
assumes [*simp*]: *sets P* = *PiM I M sets Q* = *PiM I M*
assumes *eq*: ⋀*A J*. *finite J* ⟹ *J* ⊆ *I* ⟹ (⋀*i*. *i* ∈ *J* ⟹ *A i* ∈ *sets* (*M i*))
⟹
P (*prod-emb I M J* (*Pi*$_E$ *J A*)) = *Q* (*prod-emb I M J* (*Pi*$_E$ *J A*))
assumes *A*: *finite-measure P*
shows *P* = *Q*
proof (*rule measure-eqI-generator-eq*[*OF Int-stable-prod-algebra prod-algebra-sets-into-space*])
interpret *finite-measure P* **by** *fact*
def *i* ≡ *SOME i*. *i* ∈ *I*
have *i*: *I* ≠ {} ⟹ *i* ∈ *I*
unfolding *i-def* **by** (*rule someI-ex*) *auto*
def *A* ≡ λ*n*::*nat*. *if I* = {} *then prod-emb I M* {} (Π$_E$ *i*∈{}. {}) *else prod-emb*
I M {*i*} (Π$_E$ *i*∈{*i*}. *space* (*M i*))
then show *range A* ⊆ *prod-algebra I M*
using *prod-algebraI*[*of* {} *I* λ*i*. *space* (*M i*) *M*] **by** (*auto intro*!: *prod-algebraI*
i)
have ⋀*i*. *A i* = *space* (*PiM I M*)
by (*auto simp*: *prod-emb-def space-PiM PiE-iff A-def i ex-in-conv*[*symmetric*]
exI)
then show (⋃ *i*. *A i*) = (Π$_E$ *i*∈*I*. *space* (*M i*)) ⋀*i*. *emeasure P* (*A i*) ≠ ∞
by (*auto simp*: *space-PiM*)
next
fix *X* **assume** *X*: *X* ∈ *prod-algebra I M*
then obtain *J E* **where** *X*: *X* = *prod-emb I M J* (*PIE j*:*J*. *E j*)
and *J*: *finite J J* ⊆ *I* ⋀*j*. *j* ∈ *J* ⟹ *E j* ∈ *sets* (*M j*)
by (*force elim*!: *prod-algebraE*)
then show *emeasure P X* = *emeasure Q X*
by (*auto intro*!: *eq*)
qed (*auto simp*: *sets-PiM*)

locale *product-sigma-finite* =
fixes *M* :: *'i* ⇒ *'a measure*
assumes *sigma-finite-measures*: ⋀*i*. *sigma-finite-measure* (*M i*)

sublocale *product-sigma-finite* ⊆ *M?*: *sigma-finite-measure M i* **for** *i*
by (*rule sigma-finite-measures*)

locale *finite-product-sigma-finite* = *product-sigma-finite M* **for** *M* :: *'i* ⇒ *'a mea-*
sure +

fixes $I :: 'i\ set$
assumes *finite-index*: *finite I*

lemma (**in** *finite-product-sigma-finite*) *sigma-finite-pairs*:
$\exists F::'i \Rightarrow nat \Rightarrow 'a\ set.$
$(\forall i{\in}I.\ range\ (F\ i) \subseteq sets\ (M\ i)) \wedge$
$(\forall k.\ \forall i{\in}I.\ emeasure\ (M\ i)\ (F\ i\ k) \neq \infty) \wedge incseq\ (\lambda k.\ \Pi_E\ i{\in}I.\ F\ i\ k) \wedge$
$(\bigcup k.\ \Pi_E\ i{\in}I.\ F\ i\ k) = space\ (PiM\ I\ M)$
proof −
have $\forall i::'i.\ \exists F::nat \Rightarrow 'a\ set.\ range\ F \subseteq sets\ (M\ i) \wedge incseq\ F \wedge (\bigcup i.\ F\ i) = space\ (M\ i) \wedge (\forall k.\ emeasure\ (M\ i)\ (F\ k) \neq \infty)$
using *M.sigma-finite-incseq* **by** *metis*
from *choice*[*OF this*] **guess** $F :: 'i \Rightarrow nat \Rightarrow 'a\ set$ **..**
then have *F*: $\bigwedge i.\ range\ (F\ i) \subseteq sets\ (M\ i)\ \bigwedge i.\ incseq\ (F\ i)\ \bigwedge i.\ (\bigcup j.\ F\ i\ j) = space\ (M\ i)\ \bigwedge i\ k.\ emeasure\ (M\ i)\ (F\ i\ k) \neq \infty$
by *auto*
let $?F = \lambda k.\ \Pi_E\ i{\in}I.\ F\ i\ k$
note *space-PiM*[*simp*]
show *?thesis*
proof (*intro exI*[*of - F*] *conjI allI incseq-SucI set-eqI iffI ballI*)
fix *i* **show** *range* (*F i*) ⊆ *sets* (*M i*) **by** *fact*
next
fix *i k* **show** *emeasure* (*M i*) (*F i k*) ≠ ∞ **by** *fact*
next
fix *x* **assume** *x* ∈ (⋃ *i*. *?F i*) **with** *F*(*1*) **show** *x* ∈ *space* (*PiM I M*)
by (*auto simp*: *PiE-def dest*!: *sets.sets-into-space*)
next
fix *f* **assume** *f* ∈ *space* (*PiM I M*)
with *Pi-UN*[*OF finite-index*, *of* λ*k i*. *F i k*] *F*
show *f* ∈ (⋃ *i*. *?F i*) **by** (*auto simp*: *incseq-def PiE-def*)
next
fix *i* **show** *?F i* ⊆ *?F* (*Suc i*)
using ‹⋀*i*. *incseq* (*F i*)›[*THEN incseq-SucD*] **by** *auto*
qed
qed

lemma *emeasure-PiM-empty*[*simp*]: *emeasure* (*PiM* {} *M*) {λ-. *undefined*} = *1*
proof −
let *?μ* = λ*A*. *if A* = {} *then 0 else* (*1*::*ennreal*)
have *emeasure* (*Pi*$_M$ {} *M*) (*prod-emb* {} *M* {} (Π$_E$ *i*∈{}. {})) = *1*
proof (*subst emeasure-extend-measure-Pair*[*OF PiM-def*])
show *positive* (*PiM* {} *M*) *?μ*
by (*auto simp*: *positive-def*)
show *countably-additive* (*PiM* {} *M*) *?μ*
by (*rule sets.countably-additiveI-finite*)
(*auto simp*: *additive-def positive-def sets-PiM-empty space-PiM-empty intro*!:
)
qed (*auto simp*: *prod-emb-def*)
also have (*prod-emb* {} *M* {} (Π$_E$ *i*∈{}. {})) = {λ-. *undefined*}

by (*auto simp*: *prod-emb-def*)
finally show *?thesis*
by *simp*
qed

lemma *PiM-empty*: *PiM* {} *M* = *count-space* {λ-. *undefined*}
by (*rule measure-eqI*) (*auto simp add*: *sets-PiM-empty*)

lemma (**in** *product-sigma-finite*) *emeasure-PiM*:
finite I ⟹ (⋀*i*. *i*∈*I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ *emeasure* (*PiM I M*) (*Pi*$_E$ *I A*) = (∏ *i*∈*I*. *emeasure* (*M i*) (*A i*))
proof (*induct I arbitrary*: *A rule*: *finite-induct*)
case (*insert i I*)
interpret *finite-product-sigma-finite M I* **by** *standard fact*
have *finite* (*insert i I*) **using** ‹*finite I*› **by** *auto*
interpret *I'*: *finite-product-sigma-finite M insert i I* **by** *standard fact*
let *?h* = (λ(*f*, *y*). *f*(*i* := *y*))

let *?P* = *distr* (*Pi*$_M$ *I M* ⨂$_M$ *M i*) (*Pi*$_M$ (*insert i I*) *M*) *?h*
let *?μ* = *emeasure ?P*
let *?I* = {*j* ∈ *insert i I*. *emeasure* (*M j*) (*space* (*M j*)) ≠ *1*}
let *?f* = λ*J E j*. *if j* ∈ *J then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*))

have *emeasure* (*Pi*$_M$ (*insert i I*) *M*) (*prod-emb* (*insert i I*) *M* (*insert i I*) (*Pi*$_E$ (*insert i I*) *A*)) =
(∏ *i*∈*insert i I*. *emeasure* (*M i*) (*A i*))
proof (*subst emeasure-extend-measure-Pair*[*OF PiM-def*])
fix *J E* **assume** (*J* ≠ {} ∨ *insert i I* = {}) ∧ *finite J* ∧ *J* ⊆ *insert i I* ∧ *E* ∈ (Π *j*∈*J*. *sets* (*M j*))
then have *J*: *J* ≠ {} *finite J J* ⊆ *insert i I* **and** *E*: ∀*j*∈*J*. *E j* ∈ *sets* (*M j*) **by** *auto*
let *?p* = *prod-emb* (*insert i I*) *M J* (*Pi*$_E$ *J E*)
let *?p'* = *prod-emb I M* (*J* − {*i*}) (Π$_E$ *j*∈*J*−{*i*}. *E j*)
have *?μ ?p* =
emeasure (*Pi*$_M$ *I M* ⨂$_M$ (*M i*)) (*?h* −‘ *?p* ∩ *space* (*Pi*$_M$ *I M* ⨂$_M$ *M i*))
by (*intro emeasure-distr measurable-add-dim sets-PiM-I*) *fact*+
also have *?h* −‘ *?p* ∩ *space* (*Pi*$_M$ *I M* ⨂$_M$ *M i*) = *?p'* × (*if i* ∈ *J then E i else space* (*M i*))
using *J E*[*rule-format*, *THEN sets.sets-into-space*]
by (*force simp*: *space-pair-measure space-PiM prod-emb-iff PiE-def Pi-iff split*: *if-split-asm*)
also have *emeasure* (*Pi*$_M$ *I M* ⨂$_M$ (*M i*)) (*?p'* × (*if i* ∈ *J then E i else space* (*M i*))) =
emeasure (*Pi*$_M$ *I M*) *?p'* ∗ *emeasure* (*M i*) (*if i* ∈ *J then* (*E i*) *else space* (*M i*))
using *J E* **by** (*intro M.emeasure-pair-measure-Times sets-PiM-I*) *auto*
also have *?p'* = (Π$_E$ *j*∈*I*. *if j* ∈ *J*−{*i*} *then E j else space* (*M j*))
using *J E*[*rule-format*, *THEN sets.sets-into-space*]

by (*auto simp*: *prod-emb-iff PiE-def Pi-iff split*: *if-split-asm*) *blast+*
also have *emeasure* (*Pi*$_M$ *I M*) (Π$_E$ *j*∈*I*. *if j* ∈ *J*−{*i*} *then E j else space* (*M j*)) =
(∏ *j*∈*I*. *if j* ∈ *J*−{*i*} *then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*)))
using *E* **by** (*subst insert*) (*auto intro*!: *setprod.cong*)
also have (∏*j*∈*I*. *if j* ∈ *J* − {*i*} *then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*))) ∗
emeasure (*M i*) (*if i* ∈ *J then E i else space* (*M i*)) = (∏*j*∈*insert i I*. *?f J E j*)
using *insert* **by** (*auto simp*: *mult.commute intro*!: *arg-cong2*[**where** *f*=*op* ∗] *setprod.cong*)
also have ... = (∏*j*∈*J* ∪ *?I*. *?f J E j*)
using *insert*(*1*,*2*) *J E* **by** (*intro setprod.mono-neutral-right*) *auto*
finally show *?μ ?p* = ... **.**

show *prod-emb* (*insert i I*) *M J* (*Pi*$_E$ *J E*) ∈ *Pow* (Π$_E$ *i*∈*insert i I*. *space* (*M i*))
using *J E*[*rule-format*, *THEN sets.sets-into-space*] **by** (*auto simp*: *prod-emb-iff PiE-def*)
next
show *positive* (*sets* (*Pi*$_M$ (*insert i I*) *M*)) *?μ countably-additive* (*sets* (*Pi*$_M$ (*insert i I*) *M*)) *?μ*
using *emeasure-positive*[*of ?P*] *emeasure-countably-additive*[*of ?P*] **by** *simp-all*
next
show (*insert i I* ≠ {} ∨ *insert i I* = {}) ∧ *finite* (*insert i I*) ∧
insert i I ⊆ *insert i I* ∧ *A* ∈ (Π *j*∈*insert i I*. *sets* (*M j*))
using *insert* **by** *auto*
qed (*auto intro*!: *setprod.cong*)
with *insert* **show** *?case*
by (*subst* (*asm*) *prod-emb-PiE-same-index*) (*auto intro*!: *sets.sets-into-space*)
qed *simp*

lemma (**in** *product-sigma-finite*) *PiM-eqI*:
assumes *I*[*simp*]: *finite I* **and** *P*: *sets P* = *PiM I M*
assumes *eq*: ⋀*A*. (⋀*i*. *i* ∈ *I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ *P* (*Pi*$_E$ *I A*) = (∏ *i*∈*I*. *emeasure* (*M i*) (*A i*))
shows *P* = *PiM I M*
proof −
interpret *finite-product-sigma-finite M I*
proof qed *fact*
from *sigma-finite-pairs* **guess** *C* **.. note** *C* = *this*
show *?thesis*
proof (*rule measure-eqI-PiM-finite*[*OF I refl P*, *symmetric*])
show (⋀*i*. *i* ∈ *I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ (*Pi*$_M$ *I M*) (*Pi*$_E$ *I A*) = *P* (*Pi*$_E$ *I A*) **for** *A*
by (*simp add*: *eq emeasure-PiM*)
def *A* ≡ λ*n*. Π$_E$ *i*∈*I*. *C i n*
with *C* **show** *range A* ⊆ *prod-algebra I M* ⋀*i*. *emeasure* (*Pi*$_M$ *I M*) (*A i*) ≠

∞ $(\bigcup i.\ A\ i) = space\ (PiM\ I\ M)$
by (*auto intro*!: *prod-algebraI-finite simp*: *emeasure-PiM subset-eq ennreal-setprod-eq-top*)
qed
qed

lemma (**in** *product-sigma-finite*) *sigma-finite*:
assumes *finite I*
shows *sigma-finite-measure* (*PiM I M*)
proof
interpret *finite-product-sigma-finite M I* **by** *standard fact*

obtain *F* **where** *F*: $\bigwedge j.$ *countable* (*F j*) $\bigwedge j\ f.\ f \in F\ j \Longrightarrow f \in sets\ (M\ j)$
$\bigwedge j\ f.\ f \in F\ j \Longrightarrow emeasure\ (M\ j)\ f \neq \infty$ **and**
in-space: $\bigwedge j.\ space\ (M\ j) = (\bigcup F\ j)$
using *sigma-finite-countable* **by** (*metis subset-eq*)
moreover have $(\bigcup (PiE\ I\ `\ PiE\ I\ F)) = space\ (Pi_M\ I\ M)$
using *in-space* **by** (*auto simp*: *space-PiM PiE-iff intro*!: *PiE-choice*[*THEN iffD2*])
ultimately show $\exists A.$ *countable* $A \wedge A \subseteq sets\ (Pi_M\ I\ M) \wedge \bigcup A = space\ (Pi_M\ I\ M) \wedge (\forall a{\in}A.\ emeasure\ (Pi_M\ I\ M)\ a \neq \infty)$
by (*intro exI*[*of* - *PiE I ` PiE I F*])
(*auto intro*!: *countable-PiE sets-PiM-I-finite*
simp: *PiE-iff emeasure-PiM finite-index ennreal-setprod-eq-top*)
qed

sublocale *finite-product-sigma-finite* $\subseteq$ *sigma-finite-measure* $Pi_M\ I\ M$
using *sigma-finite*[*OF finite-index*] **.**

lemma (**in** *finite-product-sigma-finite*) *measure-times*:
$(\bigwedge i.\ i \in I \Longrightarrow A\ i \in sets\ (M\ i)) \Longrightarrow emeasure\ (Pi_M\ I\ M)\ (Pi_E\ I\ A) = (\prod i{\in}I.\ emeasure\ (M\ i)\ (A\ i))$
using *emeasure-PiM*[*OF finite-index*] **by** *auto*

lemma (**in** *product-sigma-finite*) *nn-integral-empty*:
$0 \leq f\ (\lambda k.\ undefined) \Longrightarrow integral^N\ (Pi_M\ \{\}\ M)\ f = f\ (\lambda k.\ undefined)$
by (*simp add*: *PiM-empty nn-integral-count-space-finite max.absorb2*)

lemma (**in** *product-sigma-finite*) *distr-merge*:
assumes *IJ*[*simp*]: $I \cap J = \{\}$ **and** *fin*: *finite I finite J*
shows $distr\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)\ (Pi_M\ (I \cup J)\ M)\ (merge\ I\ J) = Pi_M\ (I \cup J)\ M$
(**is** *?D* = *?P*)
proof (*rule PiM-eqI*)
interpret *I*: *finite-product-sigma-finite M I* **by** *standard fact*
interpret *J*: *finite-product-sigma-finite M J* **by** *standard fact*
fix *A* **assume** *A*: $\bigwedge i.\ i \in I \cup J \Longrightarrow A\ i \in sets\ (M\ i)$
have $*$: $(merge\ I\ J\ -`\ Pi_E\ (I \cup J)\ A \cap space\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)) = PiE\ I\ A \times PiE\ J\ A$
using *A*[*THEN sets.sets-into-space*] **by** (*auto simp*: *space-PiM space-pair-measure*)

from *A fin* **show** *emeasure* (*distr* (*Pi*$_M$ *I M* ⨂$_M$ *Pi*$_M$ *J M*) (*Pi*$_M$ (*I* ∪ *J*) *M*) (*merge I J*)) (*Pi*$_E$ (*I* ∪ *J*) *A*) =
(∏ *i*∈*I* ∪ *J*. *emeasure* (*M i*) (*A i*))
by (*subst emeasure-distr*)
(*auto simp*: ∗ *J.emeasure-pair-measure-Times I.measure-times J.measure-times setprod.union-disjoint*)
qed (*insert fin*, *simp-all*)

lemma (**in** *product-sigma-finite*) *product-nn-integral-fold*:
assumes *IJ*: *I* ∩ *J* = {} *finite I finite J*
and *f*[*measurable*]: *f* ∈ *borel-measurable* (*Pi*$_M$ (*I* ∪ *J*) *M*)
shows *integral*^{N} (*Pi*$_M$ (*I* ∪ *J*) *M*) *f* =
(∫$^+$ *x*. (∫$^+$ *y*. *f* (*merge I J* (*x*, *y*)) ∂(*Pi*$_M$ *J M*)) ∂(*Pi*$_M$ *I M*))
proof −
interpret *I*: *finite-product-sigma-finite M I* **by** *standard fact*
interpret *J*: *finite-product-sigma-finite M J* **by** *standard fact*
interpret *P*: *pair-sigma-finite Pi*$_M$ *I M Pi*$_M$ *J M* **by** *standard*
have *P-borel*: (λ*x*. *f* (*merge I J x*)) ∈ *borel-measurable* (*Pi*$_M$ *I M* ⨂$_M$ *Pi*$_M$ *J M*)
using *measurable-comp*[*OF measurable-merge f*] **by** (*simp add*: *comp-def*)
show *?thesis*
apply (*subst distr-merge*[*OF IJ*, *symmetric*])
apply (*subst nn-integral-distr*[*OF measurable-merge*])
apply *measurable* []
apply (*subst J.nn-integral-fst*[*symmetric*, *OF P-borel*])
apply *simp*
done
qed

lemma (**in** *product-sigma-finite*) *distr-singleton*:
distr (*Pi*$_M$ {*i*} *M*) (*M i*) (λ*x*. *x i*) = *M i* (**is** *?D* = -)
proof (*intro measure-eqI*[*symmetric*])
interpret *I*: *finite-product-sigma-finite M* {*i*} **by** *standard simp*
fix *A* **assume** *A*: *A* ∈ *sets* (*M i*)
then have (λ*x*. *x i*) −‘ *A* ∩ *space* (*Pi*$_M$ {*i*} *M*) = (Π$_E$ *i*∈{*i*}. *A*)
using *sets.sets-into-space* **by** (*auto simp*: *space-PiM*)
then show *emeasure* (*M i*) *A* = *emeasure ?D A*
using *A I.measure-times*[*of* λ-. *A*]
by (*simp add*: *emeasure-distr measurable-component-singleton*)
qed *simp*

lemma (**in** *product-sigma-finite*) *product-nn-integral-singleton*:
assumes *f*: *f* ∈ *borel-measurable* (*M i*)
shows *integral*^{N} (*Pi*$_M$ {*i*} *M*) (λ*x*. *f* (*x i*)) = *integral*^{N} (*M i*) *f*
proof −
interpret *I*: *finite-product-sigma-finite M* {*i*} **by** *standard simp*
from *f* **show** *?thesis*
apply (*subst distr-singleton*[*symmetric*])
apply (*subst nn-integral-distr*[*OF measurable-component-singleton*])

apply *simp-all*
done
qed

lemma (**in** *product-sigma-finite*) *product-nn-integral-insert*:
assumes *I*[*simp*]: *finite I i* $\notin$ *I*
and *f*: *f* $\in$ *borel-measurable* (*Pi*$_M$ (*insert i I*) *M*)
shows *integral*^{N} (*Pi*$_M$ (*insert i I*) *M*) *f* = ($\int^+$ *x*. ($\int^+$ *y*. *f* (*x*(*i* := *y*)) ∂(*M i*)) ∂(*Pi*$_M$ *I M*))
proof −
interpret *I*: *finite-product-sigma-finite M I* **by** *standard auto*
interpret *i*: *finite-product-sigma-finite M* {*i*} **by** *standard auto*
have *IJ*: *I* $\cap$ {*i*} = {} **and** *insert*: *I* $\cup$ {*i*} = *insert i I*
using *f* **by** *auto*
show *?thesis*
unfolding *product-nn-integral-fold*[*OF IJ*, *unfolded insert*, *OF I*(*1*) *i.finite-index f*]
proof (*rule nn-integral-cong*, *subst product-nn-integral-singleton*[*symmetric*])
fix *x* **assume** *x*: *x* $\in$ *space* (*Pi*$_M$ *I M*)
let *?f* = λ*y*. *f* (*x*(*i* := *y*))
show *?f* $\in$ *borel-measurable* (*M i*)
using *measurable-comp*[*OF measurable-component-update f*, *OF x* ‹*i* $\notin$ *I*›]
unfolding *comp-def* .
show ($\int^+$ *y*. *f* (*merge I* {*i*} (*x*, *y*)) ∂*Pi*$_M$ {*i*} *M*) = ($\int^+$ *y*. *f* (*x*(*i* := *y i*)) ∂*Pi*$_M$ {*i*} *M*)
using *x*
by (*auto intro*!: *nn-integral-cong arg-cong*[**where** *f*=*f*]
simp add: *space-PiM extensional-def PiE-def*)
qed
qed

lemma (**in** *product-sigma-finite*) *product-nn-integral-insert-rev*:
assumes *I*[*simp*]: *finite I i* $\notin$ *I*
and [*measurable*]: *f* $\in$ *borel-measurable* (*Pi*$_M$ (*insert i I*) *M*)
shows *integral*^{N} (*Pi*$_M$ (*insert i I*) *M*) *f* = ($\int^+$ *y*. ($\int^+$ *x*. *f* (*x*(*i* := *y*)) ∂(*Pi*$_M$ *I M*)) ∂(*M i*))
apply (*subst product-nn-integral-insert*[*OF assms*])
apply (*rule pair-sigma-finite.Fubini'*)
apply *intro-locales* []
apply (*rule sigma-finite*[*OF I*(*1*)])
apply *measurable*
done

lemma (**in** *product-sigma-finite*) *product-nn-integral-setprod*:
assumes *finite I* $\bigwedge$*i*. *i* $\in$ *I* $\Longrightarrow$ *f i* $\in$ *borel-measurable* (*M i*)
shows ($\int^+$ *x*. ($\prod$ *i*$\in$*I*. *f i* (*x i*)) ∂*Pi*$_M$ *I M*) = ($\prod$ *i*$\in$*I*. *integral*^{N} (*M i*) (*f i*))
using *assms* **proof** (*induction I*)
case (*insert i I*)
note *insert.prems*[*measurable*]

note ‹*finite I*›[*intro*, *simp*]
interpret *I*: *finite-product-sigma-finite M I* **by** *standard auto*
have ∗: ⋀*x y*. (∏ *j*∈*I*. *f j* (*if j* = *i then y else x j*)) = (∏ *j*∈*I*. *f j* (*x j*))
using *insert* **by** (*auto intro*!: *setprod.cong*)
have *prod*: ⋀*J*. *J* ⊆ *insert i I* ⟹ (λ*x*. (∏ *i*∈*J*. *f i* (*x i*))) ∈ *borel-measurable* (*Pi*$_M$ *J M*)
using *sets.sets-into-space insert*
by (*intro borel-measurable-setprod-ennreal*
measurable-comp[*OF measurable-component-singleton*, *unfolded comp-def*])
auto
then show *?case*
apply (*simp add*: *product-nn-integral-insert*[*OF insert*(*1*,*2*)])
apply (*simp add*: *insert*(*2*−) ∗ *nn-integral-multc*)
apply (*subst nn-integral-cmult*)
apply (*auto simp add*: *insert*(*2*−))
done
qed (*simp add*: *space-PiM*)

lemma (**in** *product-sigma-finite*) *product-nn-integral-pair*:
assumes [*measurable*]: *case-prod f* ∈ *borel-measurable* (*M x* ⨂$_M$ *M y*)
assumes *xy*: *x* ≠ *y*
shows (∫$^+$σ. *f* (σ *x*) (σ *y*) ∂*PiM* {*x*, *y*} *M*) = (∫$^+$*z*. *f* (*fst z*) (*snd z*) ∂(*M x* ⨂$_M$ *M y*))
proof−
interpret *psm*: *pair-sigma-finite M x M y*
unfolding *pair-sigma-finite-def* **using** *sigma-finite-measures* **by** *simp-all*
have {*x*, *y*} = {*y*, *x*} **by** *auto*
also have (∫$^+$σ. *f* (σ *x*) (σ *y*) ∂*PiM* {*y*, *x*} *M*) = (∫$^+$*y*. ∫$^+$σ. *f* (σ *x*) *y* ∂*PiM* {*x*} *M* ∂*M y*)
using *xy* **by** (*subst product-nn-integral-insert-rev*) *simp-all*
also have ... = (∫$^+$*y*. ∫$^+$*x*. *f x y* ∂*M x* ∂*M y*)
by (*intro nn-integral-cong*, *subst product-nn-integral-singleton*) *simp-all*
also have ... = (∫$^+$*z*. *f* (*fst z*) (*snd z*) ∂(*M x* ⨂$_M$ *M y*))
by (*subst psm.nn-integral-snd*[*symmetric*]) *simp-all*
finally show *?thesis* **.**
qed

lemma (**in** *product-sigma-finite*) *distr-component*:
distr (*M i*) (*Pi*$_M$ {*i*} *M*) (λ*x*. λ*i*∈{*i*}. *x*) = *Pi*$_M$ {*i*} *M* (**is** *?D* = *?P*)
proof (*intro PiM-eqI*)
fix *A* **assume** ⋀*ia*. *ia* ∈ {*i*} ⟹ *A ia* ∈ *sets* (*M ia*)
moreover then have (λ*x*. λ*i*∈{*i*}. *x*) −‘ *Pi*$_E$ {*i*} *A* ∩ *space* (*M i*) = *A i*
by (*auto dest*: *sets.sets-into-space*)
ultimately show *emeasure* (*distr* (*M i*) (*Pi*$_M$ {*i*} *M*) (λ*x*. λ*i*∈{*i*}. *x*)) (*Pi*$_E$ {*i*} *A*) = (∏ *i*∈{*i*}. *emeasure* (*M i*) (*A i*))
by (*subst emeasure-distr*) (*auto intro*!: *sets-PiM-I-finite measurable-restrict*)
qed *simp-all*

lemma (**in** *product-sigma-finite*)

assumes *IJ*: $I \cap J = \{\}$ *finite I finite J* **and** *A*: $A \in sets\ (Pi_M\ (I \cup J)\ M)$
shows *emeasure-fold-integral*:
$emeasure\ (Pi_M\ (I \cup J)\ M)\ A = (\int^+ x.\ emeasure\ (Pi_M\ J\ M)\ ((\lambda y.\ merge\ I\ J\ (x,\ y)) -` A \cap space\ (Pi_M\ J\ M))\ \partial Pi_M\ I\ M)$ (**is** *?I*)
and *emeasure-fold-measurable*:
$(\lambda x.\ emeasure\ (Pi_M\ J\ M)\ ((\lambda y.\ merge\ I\ J\ (x,\ y)) -` A \cap space\ (Pi_M\ J\ M)))$
$\in borel\text{-}measurable\ (Pi_M\ I\ M)$ (**is** *?B*)
proof −
interpret *I*: *finite-product-sigma-finite M I* **by** *standard fact*
interpret *J*: *finite-product-sigma-finite M J* **by** *standard fact*
interpret *IJ*: *pair-sigma-finite* $Pi_M\ I\ M\ Pi_M\ J\ M$ **..**
have *merge*: $merge\ I\ J -` A \cap space\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M) \in sets\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)$
by (*intro measurable-sets*[*OF - A*] *measurable-merge assms*)

show *?I*
apply (*subst distr-merge*[*symmetric*, *OF IJ*])
apply (*subst emeasure-distr*[*OF measurable-merge A*])
apply (*subst J.emeasure-pair-measure-alt*[*OF merge*])
apply (*auto intro*!: *nn-integral-cong arg-cong2*[**where** *f*=*emeasure*] *simp*: *space-pair-measure*)
done

show *?B*
using *IJ.measurable-emeasure-Pair1*[*OF merge*]
by (*simp add*: *vimage-comp comp-def space-pair-measure cong*: *measurable-cong*)
qed

lemma *sets-Collect-single*:
$i \in I \Longrightarrow A \in sets\ (M\ i) \Longrightarrow \{\ x \in space\ (Pi_M\ I\ M).\ x\ i \in A\ \} \in sets\ (Pi_M\ I\ M)$
by *simp*

lemma *pair-measure-eq-distr-PiM*:
fixes *M1* :: *'a measure* **and** *M2* :: *'a measure*
assumes *sigma-finite-measure M1 sigma-finite-measure M2*
shows $(M1 \bigotimes_M M2) = distr\ (Pi_M\ UNIV\ (case\text{-}bool\ M1\ M2))\ (M1 \bigotimes_M M2)\ (\lambda x.\ (x\ True,\ x\ False))$
(**is** *?P* = *?D*)
proof (*rule pair-measure-eqI*[*OF assms*])
interpret *B*: *product-sigma-finite case-bool M1 M2*
unfolding *product-sigma-finite-def* **using** *assms* **by** (*auto split*: *bool.split*)
let $?B = Pi_M\ UNIV\ (case\text{-}bool\ M1\ M2)$

have [*simp*]: $fst \circ (\lambda x.\ (x\ True,\ x\ False)) = (\lambda x.\ x\ True)\ snd \circ (\lambda x.\ (x\ True,\ x\ False)) = (\lambda x.\ x\ False)$
by *auto*
fix *A B* **assume** *A*: $A \in sets\ M1$ **and** *B*: $B \in sets\ M2$
have $emeasure\ M1\ A * emeasure\ M2\ B = (\prod\ i \in UNIV.\ emeasure\ (case\text{-}bool\ M1\ M2\ i)\ (case\text{-}bool\ A\ B\ i))$

by (*simp add*: *UNIV-bool ac-simps*)
also have ... = *emeasure ?B* (*Pi*$_E$ *UNIV* (*case-bool A B*))
using *A B* **by** (*subst B.emeasure-PiM*) (*auto split*: *bool.split*)
also have *Pi*$_E$ *UNIV* (*case-bool A B*) = (λ*x*. (*x True*, *x False*)) −‘ (*A* × *B*) ∩ *space ?B*
using *A*[*THEN sets.sets-into-space*] *B*[*THEN sets.sets-into-space*]
by (*auto simp*: *PiE-iff all-bool-eq space-PiM split*: *bool.split*)
finally show *emeasure M1 A* ∗ *emeasure M2 B* = *emeasure ?D* (*A* × *B*)
using *A B*
measurable-component-singleton[*of True UNIV case-bool M1 M2*]
measurable-component-singleton[*of False UNIV case-bool M1 M2*]
by (*subst emeasure-distr*) (*auto simp*: *measurable-pair-iff*)
qed *simp*

end

8 Bochner Integration for Vector-Valued Functions

theory *Bochner-Integration*
imports *Finite-Product-Measure*
begin

In the following development of the Bochner integral we use second countable topologies instead of separable spaces. A second countable topology is also separable.

lemma *borel-measurable-implies-sequence-metric*:
fixes *f* :: *'a* ⇒ *'b* :: {*metric-space*, *second-countable-topology*}
assumes [*measurable*]: *f* ∈ *borel-measurable M*
shows ∃ *F*. (∀ *i*. *simple-function M* (*F i*)) ∧ (∀ *x*∈*space M*. (λ*i*. *F i x*) ⟶ *f x*) ∧
(∀ *i*. ∀ *x*∈*space M*. *dist* (*F i x*) *z* ≤ *2* ∗ *dist* (*f x*) *z*)
proof −
obtain *D* :: *'b set* **where** *countable D* **and** *D*: ⋀*X*. *open X* ⟹ *X* ≠ {} ⟹ ∃ *d*∈*D*. *d* ∈ *X*
by (*erule countable-dense-setE*)

def *e* ≡ *from-nat-into D*
{ fix *n x*
obtain *d* **where** *d* ∈ *D* **and** *d*: *d* ∈ *ball x* (*1* / *Suc n*)
using *D*[*of ball x* (*1* / *Suc n*)] **by** *auto*
from ‹*d* ∈ *D*› *D*[*of UNIV*] ‹*countable D*› **obtain** *i* **where** *d* = *e i*
unfolding *e-def* **by** (*auto dest*: *from-nat-into-surj*)
with *d* **have** ∃ *i*. *dist x* (*e i*) < *1* / *Suc n*
by *auto* **}**
note *e* = *this*

def *A* ≡ λ*m n*. {*x*∈*space M*. *dist* (*f x*) (*e n*) < *1* / (*Suc m*) ∧ *1* / (*Suc m*) ≤ *dist* (*f x*) *z*}

def $B \equiv \lambda m.\ disjointed\ (A\ m)$

def $m \equiv \lambda N\ x.\ Max\ \{m{::}nat.\ m \leq N \wedge x \in (\bigcup n{\leq}N.\ B\ m\ n)\}$
def $F \equiv \lambda N{::}nat.\ \lambda x.\ if\ (\exists\, m{\leq}N.\ x \in (\bigcup n{\leq}N.\ B\ m\ n)) \wedge (\exists\, n{\leq}N.\ x \in B\ (m\ N\ x)\ n)$
then $e\ (LEAST\ n.\ x \in B\ (m\ N\ x)\ n)$ *else* z

have *B-imp-A*[*intro*, *simp*]: $\bigwedge x\ m\ n.\ x \in B\ m\ n \Longrightarrow x \in A\ m\ n$
using *disjointed-subset*[*of A m* **for** *m*] **unfolding** *B-def* **by** *auto*

{ fix *m*
have $\bigwedge n.\ A\ m\ n \in sets\ M$
by (*auto simp*: *A-def*)
then have $\bigwedge n.\ B\ m\ n \in sets\ M$
using *sets.range-disjointed-sets*[*of A m M*] **by** (*auto simp*: *B-def*) **}**
note *this*[*measurable*]

{ fix *N i x* **assume** $\exists\, m{\leq}N.\ x \in (\bigcup n{\leq}N.\ B\ m\ n)$
then have $m\ N\ x \in \{m{::}nat.\ m \leq N \wedge x \in (\bigcup n{\leq}N.\ B\ m\ n)\}$
unfolding *m-def* **by** (*intro Max-in*) *auto*
then have $m\ N\ x \leq N\ \exists\, n{\leq}N.\ x \in B\ (m\ N\ x)\ n$
by *auto* **}**
note $m = this$

{ fix *j N i x* **assume** $j \leq N\ i \leq N\ x \in B\ j\ i$
then have $j \leq m\ N\ x$
unfolding *m-def* **by** (*intro Max-ge*) *auto* **}**
note *m-upper* $= this$

show *?thesis*
unfolding *simple-function-def*
proof (*safe intro*!: *exI*[*of* - *F*])
have [*measurable*]: $\bigwedge i.\ F\ i \in borel\text{-}measurable\ M$
unfolding *F-def m-def* **by** *measurable*
show $\bigwedge x\ i.\ F\ i -` \{x\} \cap space\ M \in sets\ M$
by *measurable*

{ fix *i*
{ fix *n x* **assume** $x \in B\ (m\ i\ x)\ n$
then have $(LEAST\ n.\ x \in B\ (m\ i\ x)\ n) \leq n$
by (*intro Least-le*)
also assume $n \leq i$
finally have $(LEAST\ n.\ x \in B\ (m\ i\ x)\ n) \leq i$ **. }**
then have $F\ i\ `\ space\ M \subseteq \{z\} \cup e\ `\ \{..\ i\}$
by (*auto simp*: *F-def*)
then show *finite* $(F\ i\ `\ space\ M)$
by (*rule finite-subset*) *auto* **}**

{ fix *N i n x* **assume** $i \leq N\ n \leq N\ x \in B\ i\ n$

```
      then have 1: ∃ m≤N. x ∈ (⋃ n≤N. B m n) by auto
      from m[OF this] obtain n where n: m N x ≤ N n ≤ N x ∈ B (m N x) n
by auto
      moreover
      def L ≡ LEAST n. x ∈ B (m N x) n
      have dist (f x) (e L) < 1 / Suc (m N x)
      proof −
        have x ∈ B (m N x) L
          using n(3) unfolding L-def by (rule LeastI)
        then have x ∈ A (m N x) L
          by auto
        then show ?thesis
          unfolding A-def by simp
      qed
      ultimately have dist (f x) (F N x) < 1 / Suc (m N x)
        by (auto simp add: F-def L-def) }
    note ∗ = this

    fix x assume x ∈ space M
    show (λi. F i x) ⟶ f x
    proof cases
      assume f x = z
      then have ⋀i n. x ∉ A i n
        unfolding A-def by auto
      then have ⋀i. F i x = z
        by (auto simp: F-def)
      then show ?thesis
        using ‹f x = z› by auto
    next
      assume f x ≠ z

      show ?thesis
      proof (rule tendstoI)
        fix e :: real assume 0 < e
        with ‹f x ≠ z› obtain n where 1 / Suc n < e 1 / Suc n < dist (f x) z
          by (metis dist-nz order-less-trans neq-iff nat-approx-posE)
        with ‹x∈space M› ‹f x ≠ z› have x ∈ (⋃ i. B n i)
          unfolding A-def B-def UN-disjointed-eq using e by auto
        then obtain i where i: x ∈ B n i by auto

        show eventually (λi. dist (F i x) (f x) < e) sequentially
          using eventually-ge-at-top[of max n i]
        proof eventually-elim
          fix j assume j: max n i ≤ j
          with i have dist (f x) (F j x) < 1 / Suc (m j x)
            by (intro ∗[OF - - i]) auto
          also have ... ≤ 1 / Suc n
            using j m-upper[OF - - i]
            by (auto simp: field-simps)
```

```
        also note ‹1 / Suc n < e›
        finally show dist (F j x) (f x) < e
          by (simp add: less-imp-le dist-commute)
      qed
    qed
  qed
  fix i
  { fix n m assume x ∈ A n m
    then have dist (e m) (f x) + dist (f x) z ≤ 2 * dist (f x) z
      unfolding A-def by (auto simp: dist-commute)
    also have dist (e m) z ≤ dist (e m) (f x) + dist (f x) z
      by (rule dist-triangle)
    finally (xtrans) have dist (e m) z ≤ 2 * dist (f x) z . }
  then show dist (F i x) z ≤ 2 * dist (f x) z
    unfolding F-def
    apply auto
    apply (rule LeastI2)
    apply auto
    done
 qed
qed

lemma
 fixes f :: 'a ⇒ 'b::semiring-1 assumes finite A
 shows setsum-mult-indicator[simp]: (∑ x ∈ A. f x * indicator (B x) (g x)) =
(∑ x∈{x∈A. g x ∈ B x}. f x)
  and setsum-indicator-mult[simp]: (∑ x ∈ A. indicator (B x) (g x) * f x) =
(∑ x∈{x∈A. g x ∈ B x}. f x)
 unfolding indicator-def
 using assms by (auto intro!: setsum.mono-neutral-cong-right split: if-split-asm)

lemma borel-measurable-induct-real[consumes 2, case-names set mult add seq]:
 fixes P :: ('a ⇒ real) ⇒ bool
 assumes u: u ∈ borel-measurable M ⋀x. 0 ≤ u x
 assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
 assumes mult: ⋀u c. 0 ≤ c ⟹ u ∈ borel-measurable M ⟹ (⋀x. 0 ≤ u x)
⟹ P u ⟹ P (λx. c * u x)
 assumes add: ⋀u v. u ∈ borel-measurable M ⟹ (⋀x. 0 ≤ u x) ⟹ P u ⟹ v
∈ borel-measurable M ⟹ (⋀x. 0 ≤ v x) ⟹ (⋀x. x ∈ space M ⟹ u x = 0 ∨
v x = 0) ⟹ P v ⟹ P (λx. v x + u x)
 assumes seq: ⋀U. (⋀i. U i ∈ borel-measurable M) ⟹ (⋀i x. 0 ≤ U i x) ⟹
(⋀i. P (U i)) ⟹ incseq U ⟹ (⋀x. x ∈ space M ⟹ (λi. U i x) ⟶ u x)
⟹ P u
 shows P u
proof −
 have (λx. ennreal (u x)) ∈ borel-measurable M using u by auto
 from borel-measurable-implies-simple-function-sequence'[OF this]
 obtain U where U: ⋀i. simple-function M (U i) incseq U ⋀i x. U i x < top
and
```

sup: $\bigwedge x.\ (SUP\ i.\ U\ i\ x) = ennreal\ (u\ x)$
by *blast*

def $U' \equiv \lambda i\ x.\ indicator\ (space\ M)\ x * enn2real\ (U\ i\ x)$
then have *U'-sf*[*measurable*]: $\bigwedge i.\ simple\text{-}function\ M\ (U'\ i)$
using *U* **by** (*auto intro*!: *simple-function-compose1*[**where** *g*=*enn2real*])

show *P u*
proof (*rule seq*)
show *U'*: $U'\ i \in borel\text{-}measurable\ M\ \bigwedge x.\ 0 \le U'\ i\ x$ **for** *i*
using *U* **by** (*auto*
intro: *borel-measurable-simple-function*
intro!: *borel-measurable-enn2real borel-measurable-times*
simp: *U'-def zero-le-mult-iff enn2real-nonneg*)
show *incseq U'*
using *U*(*2*,*3*)
by (*auto simp*: *incseq-def le-fun-def image-iff eq-commute U'-def indicator-def enn2real-mono*)

fix *x* **assume** *x*: $x \in space\ M$
have $(\lambda i.\ U\ i\ x) \longrightarrow (SUP\ i.\ U\ i\ x)$
using *U*(*2*) **by** (*intro LIMSEQ-SUP*) (*auto simp*: *incseq-def le-fun-def*)
moreover have $(\lambda i.\ U\ i\ x) = (\lambda i.\ ennreal\ (U'\ i\ x))$
using *x U*(*3*) **by** (*auto simp*: *fun-eq-iff U'-def image-iff eq-commute*)
moreover have $(SUP\ i.\ U\ i\ x) = ennreal\ (u\ x)$
using *sup u*(*2*) **by** (*simp add*: *max-def*)
ultimately show $(\lambda i.\ U'\ i\ x) \longrightarrow u\ x$
using *u U'* **by** *simp*
next
fix *i*
have $U'\ i\ `\ space\ M \subseteq enn2real\ `\ (U\ i\ `\ space\ M)\ finite\ (U\ i\ `\ space\ M)$
unfolding *U'-def* **using** *U*(*1*) **by** (*auto dest*: *simple-functionD*)
then have *fin*: $finite\ (U'\ i\ `\ space\ M)$
by (*metis finite-subset finite-imageI*)
moreover have $\bigwedge z.\ \{y.\ U'\ i\ z = y \wedge y \in U'\ i\ `\ space\ M \wedge z \in space\ M\} = (if\ z \in space\ M\ then\ \{U'\ i\ z\}\ else\ \{\})$
by *auto*
ultimately have *U'*: $(\lambda z.\ \sum y \in U'\ i `space\ M.\ y * indicator\ \{x \in space\ M.\ U'\ i\ x = y\}\ z) = U'\ i$
by (*simp add*: *U'-def fun-eq-iff*)
have $\bigwedge x.\ x \in U'\ i\ `\ space\ M \Longrightarrow 0 \le x$
by (*auto simp*: *U'-def enn2real-nonneg*)
with *fin* **have** *P* $(\lambda z.\ \sum y \in U'\ i `space\ M.\ y * indicator\ \{x \in space\ M.\ U'\ i\ x = y\}\ z)$
proof *induct*
case *empty* **from** *set*[*of* {}] **show** *?case*
by (*simp add*: *indicator-def*[*abs-def*])
next
case (*insert x F*)

then show *?case*
by (*auto intro*!: *add mult set setsum-nonneg split*: *split-indicator split-indicator-asm simp del*: *setsum-mult-indicator simp*: *setsum-nonneg-eq-0-iff*)
qed
with U' **show** $P\ (U'\ i)$ **by** *simp*
qed
qed

lemma *scaleR-cong-right*:
fixes $x :: 'a :: real\text{-}vector$
shows $(x \neq 0 \Longrightarrow r = p) \Longrightarrow r *_R x = p *_R x$
by (*cases* $x = 0$) *auto*

inductive *simple-bochner-integrable* :: $'a\ measure \Rightarrow ('a \Rightarrow 'b{::}real\text{-}vector) \Rightarrow bool$
for $M\ f$ **where**
simple-function $M\ f \Longrightarrow$ *emeasure* $M\ \{y{\in}space\ M.\ f\ y \neq 0\} \neq \infty \Longrightarrow$
simple-bochner-integrable $M\ f$

lemma *simple-bochner-integrable-compose2*:
assumes *p-0*: $p\ 0\ 0 = 0$
shows *simple-bochner-integrable* $M\ f \Longrightarrow$ *simple-bochner-integrable* $M\ g \Longrightarrow$
simple-bochner-integrable $M\ (\lambda x.\ p\ (f\ x)\ (g\ x))$
proof (*safe intro*!: *simple-bochner-integrable.intros elim*!: *simple-bochner-integrable.cases del*: *notI*)
assume *sf*: *simple-function* $M\ f$ *simple-function* $M\ g$
then show *simple-function* $M\ (\lambda x.\ p\ (f\ x)\ (g\ x))$
by (*rule simple-function-compose2*)

from *sf* **have** [*measurable*]:
$f \in measurable\ M\ (count\text{-}space\ UNIV)$
$g \in measurable\ M\ (count\text{-}space\ UNIV)$
by (*auto intro*: *measurable-simple-function*)

assume *fin*: *emeasure* $M\ \{y \in space\ M.\ f\ y \neq 0\} \neq \infty$ *emeasure* $M\ \{y \in space\ M.\ g\ y \neq 0\} \neq \infty$

have *emeasure* $M\ \{x{\in}space\ M.\ p\ (f\ x)\ (g\ x) \neq 0\} \leq$
emeasure $M\ (\{x{\in}space\ M.\ f\ x \neq 0\} \cup \{x{\in}space\ M.\ g\ x \neq 0\})$
by (*intro emeasure-mono*) (*auto simp*: *p-0*)
also have $\ldots \leq$ *emeasure* $M\ \{x{\in}space\ M.\ f\ x \neq 0\} +$ *emeasure* $M\ \{x{\in}space\ M.\ g\ x \neq 0\}$
by (*intro emeasure-subadditive*) *auto*
finally show *emeasure* $M\ \{y \in space\ M.\ p\ (f\ y)\ (g\ y) \neq 0\} \neq \infty$
using *fin* **by** (*auto simp*: *top-unique*)
qed

lemma *simple-function-finite-support*:
assumes *f*: *simple-function* $M\ f$ **and** *fin*: $(\int^+ x.\ f\ x\ \partial M) < \infty$ **and** *nn*: $\bigwedge x.\ 0 \leq f\ x$

shows *emeasure M* {*x*∈*space M*. *f x* ≠ *0*} ≠ ∞
proof *cases*
from *f* **have** *meas*[*measurable*]: *f* ∈ *borel-measurable M*
by (*rule borel-measurable-simple-function*)

assume *non-empty*: ∃ *x*∈*space M*. *f x* ≠ *0*

def *m* ≡ *Min* (*f*'*space M* − {*0*})
have *m* ∈ *f*'*space M* − {*0*}
unfolding *m-def* **using** *f non-empty* **by** (*intro Min-in*) (*auto simp*: *simple-function-def*)
then have *m*: *0* < *m*
using *nn* **by** (*auto simp*: *less-le*)

from *m* **have** *m* ∗ *emeasure M* {*x*∈*space M*. *0* ≠ *f x*} =
(∫⁺*x*. *m* ∗ *indicator* {*x*∈*space M*. *0* ≠ *f x*} *x* ∂*M*)
using *f* **by** (*intro nn-integral-cmult-indicator*[*symmetric*]) *auto*
also have ... ≤ (∫⁺*x*. *f x* ∂*M*)
using *AE-space*
proof (*intro nn-integral-mono-AE*, *eventually-elim*)
fix *x* **assume** *x* ∈ *space M*
with *nn* **show** *m* ∗ *indicator* {*x* ∈ *space M*. *0* ≠ *f x*} *x* ≤ *f x*
using *f* **by** (*auto split*: *split-indicator simp*: *simple-function-def m-def*)
qed
also note ‹... < ∞›
finally show *?thesis*
using *m* **by** (*auto simp*: *ennreal-mult-less-top*)
next
assume ¬ (∃ *x*∈*space M*. *f x* ≠ *0*)
with *nn* **have** ∗: {*x*∈*space M*. *f x* ≠ *0*} = {}
by *auto*
show *?thesis* **unfolding** ∗ **by** *simp*
qed

lemma *simple-bochner-integrableI-bounded*:
assumes *f*: *simple-function M f* **and** *fin*: (∫⁺*x*. *norm* (*f x*) ∂*M*) < ∞
shows *simple-bochner-integrable M f*
proof
have *emeasure M* {*y* ∈ *space M*. *ennreal* (*norm* (*f y*)) ≠ *0*} ≠ ∞
proof (*rule simple-function-finite-support*)
show *simple-function M* (λ*x*. *ennreal* (*norm* (*f x*)))
using *f* **by** (*rule simple-function-compose1*)
show (∫⁺ *y*. *ennreal* (*norm* (*f y*)) ∂*M*) < ∞ **by** *fact*
qed *simp*
then show *emeasure M* {*y* ∈ *space M*. *f y* ≠ *0*} ≠ ∞ **by** *simp*
qed *fact*

definition *simple-bochner-integral* :: *'a measure* ⇒ (*'a* ⇒ *'b*::*real-vector*) ⇒ *'b*
where
simple-bochner-integral M f = (∑ *y*∈*f*'*space M*. *measure M* {*x*∈*space M*. *f x* =

$y\}$ $*_R$ y)

lemma *simple-bochner-integral-partition*:
assumes *f*: *simple-bochner-integrable M f* **and** *g*: *simple-function M g*
assumes *sub*: $\bigwedge x\ y.\ x \in space\ M \Longrightarrow y \in space\ M \Longrightarrow g\ x = g\ y \Longrightarrow f\ x = f\ y$
assumes *v*: $\bigwedge x.\ x \in space\ M \Longrightarrow f\ x = v\ (g\ x)$
shows *simple-bochner-integral M f* = ($\sum y \in g$ ' *space M*. *measure M* {*x*∈*space M*. *g x* = *y*} $*_R$ *v y*)
(**is** - = *?r*)
proof −
from *f g* **have** [*simp*]: *finite* (*f*'*space M*) *finite* (*g*'*space M*)
by (*auto simp*: *simple-function-def elim*: *simple-bochner-integrable.cases*)

from *f* **have** [*measurable*]: *f* ∈ *measurable M* (*count-space UNIV*)
by (*auto intro*: *measurable-simple-function elim*: *simple-bochner-integrable.cases*)

from *g* **have** [*measurable*]: *g* ∈ *measurable M* (*count-space UNIV*)
by (*auto intro*: *measurable-simple-function elim*: *simple-bochner-integrable.cases*)

{ fix *y* **assume** *y* ∈ *space M*
then have *f* ' *space M* ∩ {*i*. ∃ *x*∈*space M*. *i* = *f x* ∧ *g y* = *g x*} = {*v* (*g y*)}
by (*auto cong*: *sub simp*: *v*[*symmetric*]) **}**
note *eq* = *this*

have *simple-bochner-integral M f* =
($\sum y \in f$'*space M*. ($\sum z \in g$'*space M*.
if ∃ *x*∈*space M*. *y* = *f x* ∧ *z* = *g x then measure M* {*x*∈*space M*. *g x* = *z*} *else 0*) $*_R$ *y*)
unfolding *simple-bochner-integral-def*
proof (*safe intro*!: *setsum.cong scaleR-cong-right*)
fix *y* **assume** *y*: *y* ∈ *space M f y* ≠ *0*
have [*simp*]: *g* ' *space M* ∩ {*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*} =
{*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*}
by *auto*
have *eq*:{*x* ∈ *space M*. *f x* = *f y*} =
($\bigcup i \in$ {*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*}. {*x* ∈ *space M*. *g x* = *i*})
by (*auto simp*: *eq-commute cong*: *sub rev-conj-cong*)
have *finite* (*g*'*space M*) **by** *simp*
then have *finite* {*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*}
by (*rule rev-finite-subset*) *auto*
moreover
{ fix *x* **assume** *x* ∈ *space M f x* = *f y*
then have *x* ∈ *space M f x* ≠ *0*
using *y* **by** *auto*
then have *emeasure M* {*y* ∈ *space M*. *g y* = *g x*} ≤ *emeasure M* {*y* ∈ *space M*. *f y* ≠ *0*}
by (*auto intro*!: *emeasure-mono cong*: *sub*)
then have *emeasure M* {*xa* ∈ *space M*. *g xa* = *g x*} < ∞
using *f* **by** (*auto simp*: *simple-bochner-integrable.simps less-top*) **}**

ultimately
show *measure M* {*x* ∈ *space M*. *f x* = *f y*} =
(∑ *z*∈*g* ' *space M*. *if* ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x then measure M* {*x* ∈ *space M*. *g x* = *z*} *else 0*)
apply (*simp add*: *setsum.If-cases eq*)
apply (*subst measure-finite-Union*[*symmetric*])
apply (*auto simp*: *disjoint-family-on-def less-top*)
done
qed
also have ... = (∑ *y*∈*f*'*space M*. (∑ *z*∈*g*'*space M*.
if ∃ *x*∈*space M*. *y* = *f x* ∧ *z* = *g x then measure M* {*x*∈*space M*. *g x* = *z*} ∗$_R$ *y else 0*))
by (*auto intro*!: *setsum.cong simp*: *scaleR-setsum-left*)
also have ... = *?r*
by (*subst setsum.commute*)
(*auto intro*!: *setsum.cong simp*: *setsum.If-cases scaleR-setsum-right*[*symmetric*] *eq*)
finally show *simple-bochner-integral M f* = *?r* **.**
qed

lemma *simple-bochner-integral-add*:
assumes *f*: *simple-bochner-integrable M f* **and** *g*: *simple-bochner-integrable M g*
shows *simple-bochner-integral M* (λ*x*. *f x* + *g x*) =
simple-bochner-integral M f + *simple-bochner-integral M g*
proof −
from *f g* **have** *simple-bochner-integral M* (λ*x*. *f x* + *g x*) =
(∑ *y*∈(λ*x*. (*f x*, *g x*)) ' *space M*. *measure M* {*x* ∈ *space M*. (*f x*, *g x*) = *y*} ∗$_R$ (*fst y* + *snd y*))
by (*intro simple-bochner-integral-partition*)
(*auto simp*: *simple-bochner-integrable-compose2 elim*: *simple-bochner-integrable.cases*)
moreover from *f g* **have** *simple-bochner-integral M f* =
(∑ *y*∈(λ*x*. (*f x*, *g x*)) ' *space M*. *measure M* {*x* ∈ *space M*. (*f x*, *g x*) = *y*} ∗$_R$ *fst y*)
by (*intro simple-bochner-integral-partition*)
(*auto simp*: *simple-bochner-integrable-compose2 elim*: *simple-bochner-integrable.cases*)
moreover from *f g* **have** *simple-bochner-integral M g* =
(∑ *y*∈(λ*x*. (*f x*, *g x*)) ' *space M*. *measure M* {*x* ∈ *space M*. (*f x*, *g x*) = *y*} ∗$_R$ *snd y*)
by (*intro simple-bochner-integral-partition*)
(*auto simp*: *simple-bochner-integrable-compose2 elim*: *simple-bochner-integrable.cases*)
ultimately show *?thesis*
by (*simp add*: *setsum.distrib*[*symmetric*] *scaleR-add-right*)
qed

lemma (**in** *linear*) *simple-bochner-integral-linear*:
assumes *g*: *simple-bochner-integrable M g*
shows *simple-bochner-integral M* (λ*x*. *f* (*g x*)) = *f* (*simple-bochner-integral M g*)
proof −

from *g* **have** *simple-bochner-integral M* (λ*x*. *f* (*g x*)) =
(∑ *y*∈*g* ' *space M*. *measure M* {*x* ∈ *space M*. *g x* = *y*} ∗$_R$ *f y*)
by (*intro simple-bochner-integral-partition*)
(*auto simp*: *simple-bochner-integrable-compose2*[**where** *p*=λ*x y*. *f x*] *zero*
elim: *simple-bochner-integrable.cases*)
also have ... = *f* (*simple-bochner-integral M g*)
by (*simp add*: *simple-bochner-integral-def setsum scaleR*)
finally show *?thesis* **.**
qed

lemma *simple-bochner-integral-minus*:
assumes *f*: *simple-bochner-integrable M f*
shows *simple-bochner-integral M* (λ*x*. − *f x*) = − *simple-bochner-integral M f*
proof −
interpret *linear uminus* **by** *unfold-locales auto*
from *f* **show** *?thesis*
by (*rule simple-bochner-integral-linear*)
qed

lemma *simple-bochner-integral-diff*:
assumes *f*: *simple-bochner-integrable M f* **and** *g*: *simple-bochner-integrable M g*
shows *simple-bochner-integral M* (λ*x*. *f x* − *g x*) =
simple-bochner-integral M f − *simple-bochner-integral M g*
unfolding *diff-conv-add-uminus* **using** *f g*
by (*subst simple-bochner-integral-add*)
(*auto simp*: *simple-bochner-integral-minus simple-bochner-integrable-compose2*[**where** *p*=λ*x y*. − *y*])

lemma *simple-bochner-integral-norm-bound*:
assumes *f*: *simple-bochner-integrable M f*
shows *norm* (*simple-bochner-integral M f*) ≤ *simple-bochner-integral M* (λ*x*. *norm* (*f x*))
proof −
have *norm* (*simple-bochner-integral M f*) ≤
(∑ *y*∈*f* ' *space M*. *norm* (*measure M* {*x* ∈ *space M*. *f x* = *y*} ∗$_R$ *y*))
unfolding *simple-bochner-integral-def* **by** (*rule norm-setsum*)
also have ... = (∑ *y*∈*f* ' *space M*. *measure M* {*x* ∈ *space M*. *f x* = *y*} ∗$_R$ *norm* *y*)
by *simp*
also have ... = *simple-bochner-integral M* (λ*x*. *norm* (*f x*))
using *f*
by (*intro simple-bochner-integral-partition*[*symmetric*])
(*auto intro*: *f simple-bochner-integrable-compose2 elim*: *simple-bochner-integrable.cases*)
finally show *?thesis* **.**
qed

lemma *simple-bochner-integral-nonneg*[*simp*]:
fixes *f* :: *'a* ⇒ *real*
shows (⋀*x*. *0* ≤ *f x*) ⟹ *0* ≤ *simple-bochner-integral M f*

by (*simp add*: *setsum-nonneg simple-bochner-integral-def*)

lemma *simple-bochner-integral-eq-nn-integral*:
assumes *f*: *simple-bochner-integrable M f* $\bigwedge x.\ 0 \le f\ x$
shows *simple-bochner-integral M f* $= (\int^+ x.\ f\ x\ \partial M)$
proof −
{ fix *x y z* **have** $(x \neq 0 \Longrightarrow y = z) \Longrightarrow$ *ennreal x* $*$ *y* $=$ *ennreal x* $*$ *z*
by (*cases x* $= 0$) (*auto simp*: *zero-ennreal-def*[*symmetric*]) **}**
note *ennreal-cong-mult* $=$ *this*

have [*measurable*]: *f* $\in$ *borel-measurable M*
using *f*(*1*) **by** (*auto intro*: *borel-measurable-simple-function elim*: *simple-bochner-integrable.cases*)

{ fix *y* **assume** *y*: *y* $\in$ *space M f y* $\neq 0$
have *ennreal* (*measure M* $\{x \in$ *space M*. *f x* $=$ *f y*$\}$) $=$ *emeasure M* $\{x \in$ *space M*. *f x* $=$ *f y*$\}$
proof (*rule emeasure-eq-ennreal-measure*[*symmetric*])
have *emeasure M* $\{x \in$ *space M*. *f x* $=$ *f y*$\} \le$ *emeasure M* $\{x \in$ *space M*. *f x* $\neq 0\}$
using *y* **by** (*intro emeasure-mono*) *auto*
with *f* **show** *emeasure M* $\{x \in$ *space M*. *f x* $=$ *f y*$\} \neq$ *top*
by (*auto simp*: *simple-bochner-integrable.simps top-unique*)
qed
moreover have $\{x \in$ *space M*. *f x* $=$ *f y*$\} = (\lambda x.$ *ennreal* (*f x*)) $-`$ $\{$*ennreal* (*f y*)$\} \cap$ *space M*
using *f* **by** *auto*
ultimately have *ennreal* (*measure M* $\{x \in$ *space M*. *f x* $=$ *f y*$\}$) $=$
emeasure M $((\lambda x.$ *ennreal* (*f x*)) $-`$ $\{$*ennreal* (*f y*)$\} \cap$ *space M*) **by** *simp* **}**
with *f* **have** *simple-bochner-integral M f* $= (\int^S x.\ f\ x\ \partial M)$
unfolding *simple-integral-def*
by (*subst simple-bochner-integral-partition*[*OF f*(*1*), **where** $g = \lambda x.$ *ennreal* (*f x*) **and** *v*=*enn2real*])
(*auto intro*: *f simple-function-compose1 elim*: *simple-bochner-integrable.cases*
intro!: *setsum.cong ennreal-cong-mult*
simp: *setsum-ennreal*[*symmetric*] *ac-simps ennreal-mult*
simp del: *setsum-ennreal*)
also have $\ldots = (\int^+ x.\ f\ x\ \partial M)$
using *f*
by (*intro nn-integral-eq-simple-integral*[*symmetric*])
(*auto simp*: *simple-function-compose1 simple-bochner-integrable.simps*)
finally show *?thesis* **.**
qed

lemma *simple-bochner-integral-bounded*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*real-normed-vector*, *second-countable-topology*}
assumes *f*[*measurable*]: *f* $\in$ *borel-measurable M*
assumes *s*: *simple-bochner-integrable M s* **and** *t*: *simple-bochner-integrable M t*
shows *ennreal* (*norm* (*simple-bochner-integral M s* $-$ *simple-bochner-integral M t*)) $\le$

$(\int^+ x.\ norm\ (f\ x - s\ x)\ \partial M) + (\int^+ x.\ norm\ (f\ x - t\ x)\ \partial M)$
$(\textbf{is}\ ennreal\ (norm\ (?s - ?t)) \leq ?S + ?T)$
proof −
have [*measurable*]: $s \in$ *borel-measurable* M $t \in$ *borel-measurable* M
using *s t* **by** (*auto intro*: *borel-measurable-simple-function elim*: *simple-bochner-integrable.cases*)

have *ennreal* (*norm* (*?s* − *?t*)) = *norm* (*simple-bochner-integral M* (λ*x*. *s x* − *t x*))
using *s t* **by** (*subst simple-bochner-integral-diff*) *auto*
also have ... ≤ *simple-bochner-integral M* (λ*x*. *norm* (*s x* − *t x*))
using *simple-bochner-integrable-compose2*[*of op* − *M s t*] *s t*
by (*auto intro*!: *simple-bochner-integral-norm-bound*)
also have ... = $(\int^+ x.\ norm\ (s\ x - t\ x)\ \partial M)$
using *simple-bochner-integrable-compose2*[*of* λ*x y*. *norm* (*x* − *y*) *M s t*] *s t*
by (*auto intro*!: *simple-bochner-integral-eq-nn-integral*)
also have ... ≤ $(\int^+ x.\ ennreal\ (norm\ (f\ x - s\ x)) + ennreal\ (norm\ (f\ x - t\ x))\ \partial M)$
by (*auto intro*!: *nn-integral-mono simp*: *ennreal-plus*[*symmetric*] *simp del*: *ennreal-plus*)
(*metis* (*erased*, *hide-lams*) *add-diff-cancel-left add-diff-eq diff-add-eq order-trans norm-minus-commute norm-triangle-ineq4 order-refl*)
also have ... = *?S* + *?T*
by (*rule nn-integral-add*) *auto*
finally show *?thesis* **.**
qed

inductive *has-bochner-integral* :: *'a measure* ⇒ (*'a* ⇒ *'b*) ⇒ *'b*::{*real-normed-vector*, *second-countable-topology*} ⇒ *bool*
for *M f x* **where**
f ∈ *borel-measurable M* ⟹
(⋀*i*. *simple-bochner-integrable M* (*s i*)) ⟹
(λ*i*. $\int^+ x.\ norm\ (f\ x - s\ i\ x)\ \partial M$) ⟶ *0* ⟹
(λ*i*. *simple-bochner-integral M* (*s i*)) ⟶ *x* ⟹
has-bochner-integral M f x

lemma *has-bochner-integral-cong*:
assumes *M* = *N* ⋀*x*. *x* ∈ *space N* ⟹ *f x* = *g x* ∧ *x* = *y*
shows *has-bochner-integral M f x* ⟷ *has-bochner-integral N g y*
unfolding *has-bochner-integral.simps assms*(*1*,*3*)
using *assms*(*2*) **by** (*simp cong*: *measurable-cong-strong nn-integral-cong-strong*)

lemma *has-bochner-integral-cong-AE*:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹ (*AE x in M*. *f x* = *g x*) ⟹
has-bochner-integral M f x ⟷ *has-bochner-integral M g x*
unfolding *has-bochner-integral.simps*
by (*intro arg-cong*[**where** *f*=*Ex*] *ext conj-cong rev-conj-cong refl arg-cong*[**where** *f*=λ*x*. *x* ⟶ *0*] *nn-integral-cong-AE*)

auto

lemma *borel-measurable-has-bochner-integral*:
has-bochner-integral M f x $\Longrightarrow$ *f* $\in$ *borel-measurable M*
by (*rule has-bochner-integral.cases*)

lemma *borel-measurable-has-bochner-integral'*[*measurable-dest*]:
has-bochner-integral M f x $\Longrightarrow$ *g* $\in$ *measurable N M* $\Longrightarrow$ $(\lambda x.\ f\ (g\ x)) \in$ *borel-measurable N*
using *borel-measurable-has-bochner-integral*[*measurable*] **by** *measurable*

lemma *has-bochner-integral-simple-bochner-integrable*:
simple-bochner-integrable M f $\Longrightarrow$ *has-bochner-integral M f* (*simple-bochner-integral M f*)
by (*rule has-bochner-integral.intros*[**where** *s*=λ-. *f*])
(*auto intro*: *borel-measurable-simple-function*
elim: *simple-bochner-integrable.cases*
simp: *zero-ennreal-def*[*symmetric*])

lemma *has-bochner-integral-real-indicator*:
assumes [*measurable*]: *A* $\in$ *sets M* **and** *A*: *emeasure M A* $< \infty$
shows *has-bochner-integral M* (*indicator A*) (*measure M A*)
proof −
have *sbi*: *simple-bochner-integrable M* (*indicator A*::'*a* $\Rightarrow$ *real*)
proof
have $\{y \in$ *space M*. (*indicator A y*::*real*) $\neq 0\} =$ *A*
using *sets.sets-into-space*[*OF* ‹*A*$\in$*sets M*›] **by** (*auto split*: *split-indicator*)
then show *emeasure M* $\{y \in$ *space M*. (*indicator A y*::*real*) $\neq 0\} \neq \infty$
using *A* **by** *auto*
qed (*rule simple-function-indicator assms*)+
moreover have *simple-bochner-integral M* (*indicator A*) = *measure M A*
using *simple-bochner-integral-eq-nn-integral*[*OF sbi*] *A*
by (*simp add*: *ennreal-indicator emeasure-eq-ennreal-measure*)
ultimately show *?thesis*
by (*metis has-bochner-integral-simple-bochner-integrable*)
qed

lemma *has-bochner-integral-add*[*intro*]:
has-bochner-integral M f x $\Longrightarrow$ *has-bochner-integral M g y* $\Longrightarrow$
has-bochner-integral M $(\lambda x.\ f\ x + g\ x)\ (x + y)$
proof (*safe intro*!: *has-bochner-integral.intros elim*!: *has-bochner-integral.cases*)
fix *sf sg*
assume *f-sf*: $(\lambda i.\ \int^+ x.\ norm\ (f\ x - sf\ i\ x)\ \partial M) \longrightarrow 0$
assume *g-sg*: $(\lambda i.\ \int^+ x.\ norm\ (g\ x - sg\ i\ x)\ \partial M) \longrightarrow 0$

assume *sf*: $\forall i.$ *simple-bochner-integrable M* (*sf i*)
and *sg*: $\forall i.$ *simple-bochner-integrable M* (*sg i*)
then have [*measurable*]: $\bigwedge i.$ *sf i* $\in$ *borel-measurable M* $\bigwedge i.$ *sg i* $\in$ *borel-measurable M*

by (*auto intro*: *borel-measurable-simple-function elim*: *simple-bochner-integrable.cases*)
assume [*measurable*]: $f \in$ *borel-measurable M* $g \in$ *borel-measurable M*

show $\bigwedge i$. *simple-bochner-integrable M* (λx. *sf i x* + *sg i x*)
using *sf sg* **by** (*simp add*: *simple-bochner-integrable-compose2*)

show (λi. $\int^+ x$. (*norm* (*f x* + *g x* − (*sf i x* + *sg i x*))) ∂M) $\longrightarrow$ *0*
(**is** *?f* $\longrightarrow$ *0*)
proof (*rule tendsto-sandwich*)
show *eventually* (λn. *0* ≤ *?f n*) *sequentially* (λ-. *0*) $\longrightarrow$ *0*
by *auto*
show *eventually* (λi. *?f i* ≤ ($\int^+ x$. (*norm* (*f x* − *sf i x*)) ∂M) + $\int^+ x$. (*norm* (*g x* − *sg i x*)) ∂M) *sequentially*
(**is** *eventually* (λi. *?f i* ≤ *?g i*) *sequentially*)
proof (*intro always-eventually allI*)
fix *i* **have** *?f i* ≤ ($\int^+ x$. (*norm* (*f x* − *sf i x*)) + *ennreal* (*norm* (*g x* − *sg i x*)) ∂M)
by (*auto intro*!: *nn-integral-mono norm-diff-triangle-ineq*
simp del: *ennreal-plus simp add*: *ennreal-plus*[*symmetric*])
also have ... = *?g i*
by (*intro nn-integral-add*) *auto*
finally show *?f i* ≤ *?g i* **.**
qed
show *?g* $\longrightarrow$ *0*
using *tendsto-add*[*OF f-sf g-sg*] **by** *simp*
qed
qed (*auto simp*: *simple-bochner-integral-add tendsto-add*)

lemma *has-bochner-integral-bounded-linear*:
assumes *bounded-linear T*
shows *has-bochner-integral M f x* ⟹ *has-bochner-integral M* (λx. *T* (*f x*)) (*T x*)
proof (*safe intro*!: *has-bochner-integral.intros elim*!: *has-bochner-integral.cases*)
interpret *T*: *bounded-linear T* **by** *fact*
have [*measurable*]: *T* ∈ *borel-measurable borel*
by (*intro borel-measurable-continuous-on1 T.continuous-on continuous-on-id*)
assume [*measurable*]: *f* ∈ *borel-measurable M*
then show (λx. *T* (*f x*)) ∈ *borel-measurable M*
by *auto*

fix *s* **assume** *f-s*: (λi. $\int^+ x$. *norm* (*f x* − *s i x*) ∂M) $\longrightarrow$ *0*
assume *s*: $\forall i$. *simple-bochner-integrable M* (*s i*)
then show $\bigwedge i$. *simple-bochner-integrable M* (λx. *T* (*s i x*))
by (*auto intro*: *simple-bochner-integrable-compose2 T.zero*)

have [*measurable*]: $\bigwedge i$. *s i* ∈ *borel-measurable M*
using *s* **by** (*auto intro*: *borel-measurable-simple-function elim*: *simple-bochner-integrable.cases*)

obtain *K* **where** *K*: *K* > *0* $\bigwedge x\ i$. *norm* (*T* (*f x*) − *T* (*s i x*)) ≤ *norm* (*f x* −

s i x) ∗ *K*
using *T.pos-bounded* **by** (*auto simp*: *T.diff*[*symmetric*])

show (λ*i*. ∫⁺ *x*. *norm* (*T* (*f x*) − *T* (*s i x*)) ∂*M*) ⟶ *0*
(**is** *?f* ⟶ *0*)
proof (*rule tendsto-sandwich*)
show *eventually* (λ*n*. *0* ≤ *?f n*) *sequentially* (λ-. *0*) ⟶ *0*
by *auto*

show *eventually* (λ*i*. *?f i* ≤ *K* ∗ (∫⁺ *x*. *norm* (*f x* − *s i x*) ∂*M*)) *sequentially*
(**is** *eventually* (λ*i*. *?f i* ≤ *?g i*) *sequentially*)
proof (*intro always-eventually allI*)
fix *i* **have** *?f i* ≤ (∫⁺ *x*. *ennreal K* ∗ *norm* (*f x* − *s i x*) ∂*M*)
using *K* **by** (*intro nn-integral-mono*) (*auto simp*: *ac-simps ennreal-mult*[*symmetric*])
also have ... = *?g i*
using *K* **by** (*intro nn-integral-cmult*) *auto*
finally show *?f i* ≤ *?g i* **.**
qed
show *?g* ⟶ *0*
using *ennreal-tendsto-cmult*[*OF - f-s*] **by** *simp*
qed

assume (λ*i*. *simple-bochner-integral M* (*s i*)) ⟶ *x*
with *s* **show** (λ*i*. *simple-bochner-integral M* (λ*x*. *T* (*s i x*))) ⟶ *T x*
by (*auto intro*!: *T.tendsto simp*: *T.simple-bochner-integral-linear*)
qed

lemma *has-bochner-integral-zero*[*intro*]: *has-bochner-integral M* (λ*x*. *0*) *0*
by (*auto intro*!: *has-bochner-integral.intros*[**where** *s*=λ- -. *0*]
simp: *zero-ennreal-def*[*symmetric*] *simple-bochner-integrable.simps*
simple-bochner-integral-def image-constant-conv)

lemma *has-bochner-integral-scaleR-left*[*intro*]:
(*c* ≠ *0* ⟹ *has-bochner-integral M f x*) ⟹ *has-bochner-integral M* (λ*x*. *f x* ∗*R*
c) (*x* ∗*R* *c*)
by (*cases c* = *0*) (*auto simp add*: *has-bochner-integral-bounded-linear*[*OF bounded-linear-scaleR-left*])

lemma *has-bochner-integral-scaleR-right*[*intro*]:
(*c* ≠ *0* ⟹ *has-bochner-integral M f x*) ⟹ *has-bochner-integral M* (λ*x*. *c* ∗*R* *f*
x) (*c* ∗*R* *x*)
by (*cases c* = *0*) (*auto simp add*: *has-bochner-integral-bounded-linear*[*OF bounded-linear-scaleR-right*])

lemma *has-bochner-integral-mult-left*[*intro*]:
fixes *c* :: -::{*real-normed-algebra*,*second-countable-topology*}
shows (*c* ≠ *0* ⟹ *has-bochner-integral M f x*) ⟹ *has-bochner-integral M* (λ*x*.
f x ∗ *c*) (*x* ∗ *c*)
by (*cases c* = *0*) (*auto simp add*: *has-bochner-integral-bounded-linear*[*OF bounded-linear-mult-left*])

lemma *has-bochner-integral-mult-right*[*intro*]:

fixes c :: -::{*real-normed-algebra*,*second-countable-topology*}
shows $(c \neq 0 \Longrightarrow$ *has-bochner-integral* $M\ f\ x) \Longrightarrow$ *has-bochner-integral* $M\ (\lambda x.$
$c * f\ x)\ (c * x)$
by (*cases* $c = 0$) (*auto simp add*: *has-bochner-integral-bounded-linear*[*OF bounded-linear-mult-right*])

lemmas *has-bochner-integral-divide* =
has-bochner-integral-bounded-linear[*OF bounded-linear-divide*]

lemma *has-bochner-integral-divide-zero*[*intro*]:
fixes c :: -::{*real-normed-field*, *field*, *second-countable-topology*}
shows $(c \neq 0 \Longrightarrow$ *has-bochner-integral* $M\ f\ x) \Longrightarrow$ *has-bochner-integral* $M\ (\lambda x.$
$f\ x\ /\ c)\ (x\ /\ c)$
using *has-bochner-integral-divide* **by** (*cases* $c = 0$) *auto*

lemma *has-bochner-integral-inner-left*[*intro*]:
$(c \neq 0 \Longrightarrow$ *has-bochner-integral* $M\ f\ x) \Longrightarrow$ *has-bochner-integral* $M\ (\lambda x.\ f\ x \cdot c)$
$(x \cdot c)$
by (*cases* $c = 0$) (*auto simp add*: *has-bochner-integral-bounded-linear*[*OF bounded-linear-inner-left*])

lemma *has-bochner-integral-inner-right*[*intro*]:
$(c \neq 0 \Longrightarrow$ *has-bochner-integral* $M\ f\ x) \Longrightarrow$ *has-bochner-integral* $M\ (\lambda x.\ c \cdot f\ x)$
$(c \cdot x)$
by (*cases* $c = 0$) (*auto simp add*: *has-bochner-integral-bounded-linear*[*OF bounded-linear-inner-right*])

lemmas *has-bochner-integral-minus* =
has-bochner-integral-bounded-linear[*OF bounded-linear-minus*[*OF bounded-linear-ident*]]
lemmas *has-bochner-integral-Re* =
has-bochner-integral-bounded-linear[*OF bounded-linear-Re*]
lemmas *has-bochner-integral-Im* =
has-bochner-integral-bounded-linear[*OF bounded-linear-Im*]
lemmas *has-bochner-integral-cnj* =
has-bochner-integral-bounded-linear[*OF bounded-linear-cnj*]
lemmas *has-bochner-integral-of-real* =
has-bochner-integral-bounded-linear[*OF bounded-linear-of-real*]
lemmas *has-bochner-integral-fst* =
has-bochner-integral-bounded-linear[*OF bounded-linear-fst*]
lemmas *has-bochner-integral-snd* =
has-bochner-integral-bounded-linear[*OF bounded-linear-snd*]

lemma *has-bochner-integral-indicator*:
$A \in$ *sets* $M \Longrightarrow$ *emeasure* $M\ A < \infty \Longrightarrow$
has-bochner-integral $M\ (\lambda x.$ *indicator* $A\ x *_R c)$ (*measure* $M\ A *_R c$)
by (*intro has-bochner-integral-scaleR-left has-bochner-integral-real-indicator*)

lemma *has-bochner-integral-diff*:
has-bochner-integral $M\ f\ x \Longrightarrow$ *has-bochner-integral* $M\ g\ y \Longrightarrow$
has-bochner-integral $M\ (\lambda x.\ f\ x - g\ x)\ (x - y)$
unfolding *diff-conv-add-uminus*
by (*intro has-bochner-integral-add has-bochner-integral-minus*)

lemma *has-bochner-integral-setsum*:
(⋀*i*. *i* ∈ *I* ⟹ *has-bochner-integral M* (*f i*) (*x i*)) ⟹
has-bochner-integral M (λ*x*. ∑ *i*∈*I*. *f i x*) (∑ *i*∈*I*. *x i*)
by (*induct I rule*: *infinite-finite-induct*) *auto*

lemma *has-bochner-integral-implies-finite-norm*:
has-bochner-integral M f x ⟹ (∫⁺*x*. *norm* (*f x*) ∂*M*) < ∞
proof (*elim has-bochner-integral.cases*)
fix *s v*
assume [*measurable*]: *f* ∈ *borel-measurable M* **and** *s*: ⋀*i*. *simple-bochner-integrable M* (*s i*) **and**
lim-0: (λ*i*. ∫⁺ *x*. *ennreal* (*norm* (*f x* − *s i x*)) ∂*M*) ⟶ *0*
from *order-tendstoD*[*OF lim-0*, *of* ∞]
obtain *i* **where** *f-s-fin*: (∫⁺ *x*. *ennreal* (*norm* (*f x* − *s i x*)) ∂*M*) < ∞
by (*auto simp*: *eventually-sequentially*)

have [*measurable*]: ⋀*i*. *s i* ∈ *borel-measurable M*
using *s* **by** (*auto intro*: *borel-measurable-simple-function elim*: *simple-bochner-integrable.cases*)

def *m* ≡ *if space M* = {} *then 0 else Max* ((λ*x*. *norm* (*s i x*))‘*space M*)
have *finite* (*s i* ‘ *space M*)
using *s* **by** (*auto simp*: *simple-function-def simple-bochner-integrable.simps*)
then have *finite* (*norm* ‘ *s i* ‘ *space M*)
by (*rule finite-imageI*)
then have ⋀*x*. *x* ∈ *space M* ⟹ *norm* (*s i x*) ≤ *m* *0* ≤ *m*
by (*auto simp*: *m-def image-comp comp-def Max-ge-iff*)
then have (∫⁺*x*. *norm* (*s i x*) ∂*M*) ≤ (∫⁺*x*. *ennreal m* ∗ *indicator* {*x*∈*space M*. *s i x* ≠ *0*} *x* ∂*M*)
by (*auto split*: *split-indicator intro*!: *Max-ge nn-integral-mono simp*:)
also have ... < ∞
using *s* **by** (*subst nn-integral-cmult-indicator*) (*auto simp*: ‹*0* ≤ *m*› *simple-bochner-integrable.simps ennreal-mult-less-top less-top*)
finally have *s-fin*: (∫⁺*x*. *norm* (*s i x*) ∂*M*) < ∞ **.**

have (∫⁺ *x*. *norm* (*f x*) ∂*M*) ≤ (∫⁺ *x*. *ennreal* (*norm* (*f x* − *s i x*)) + *ennreal* (*norm* (*s i x*)) ∂*M*)
by (*auto intro*!: *nn-integral-mono simp del*: *ennreal-plus simp add*: *ennreal-plus*[*symmetric*])
(*metis add.commute norm-triangle-sub*)
also have ... = (∫⁺*x*. *norm* (*f x* − *s i x*) ∂*M*) + (∫⁺*x*. *norm* (*s i x*) ∂*M*)
by (*rule nn-integral-add*) *auto*
also have ... < ∞
using *s-fin f-s-fin* **by** *auto*
finally show (∫⁺ *x*. *ennreal* (*norm* (*f x*)) ∂*M*) < ∞ **.**
qed

lemma *has-bochner-integral-norm-bound*:
assumes *i*: *has-bochner-integral M f x*
shows *norm x* ≤ (∫⁺*x*. *norm* (*f x*) ∂*M*)

using *assms* **proof**
fix *s* **assume**
x: (λ*i*. *simple-bochner-integral M* (*s i*)) ⟶ *x* (**is** *?s* ⟶ *x*) **and**
s[*simp*]: ⋀*i*. *simple-bochner-integrable M* (*s i*) **and**
lim: (λ*i*. ∫⁺ *x*. *ennreal* (*norm* (*f x* − *s i x*)) ∂*M*) ⟶ *0* **and**
f[*measurable*]: *f* ∈ *borel-measurable M*

have [*measurable*]: ⋀*i*. *s i* ∈ *borel-measurable M*
using *s* **by** (*auto simp*: *simple-bochner-integrable.simps intro*: *borel-measurable-simple-function*)

show *norm x* ≤ (∫⁺*x*. *norm* (*f x*) ∂*M*)
proof (*rule LIMSEQ-le*)
show (λ*i*. *ennreal* (*norm* (*?s i*))) ⟶ *norm x*
using *x* **by** (*auto simp*: *tendsto-ennreal-iff intro*: *tendsto-intros*)
show ∃*N*. ∀*n*≥*N*. *norm* (*?s n*) ≤ (∫⁺*x*. *norm* (*f x* − *s n x*) ∂*M*) + (∫⁺*x*. *norm* (*f x*) ∂*M*)
(**is** ∃*N*. ∀*n*≥*N*. - ≤ *?t n*)
proof (*intro exI allI impI*)
fix *n*
have *ennreal* (*norm* (*?s n*)) ≤ *simple-bochner-integral M* (λ*x*. *norm* (*s n x*))
by (*auto intro*!: *simple-bochner-integral-norm-bound*)
also have ... = (∫⁺*x*. *norm* (*s n x*) ∂*M*)
by (*intro simple-bochner-integral-eq-nn-integral*)
(*auto intro*: *s simple-bochner-integrable-compose2*)
also have ... ≤ (∫⁺*x*. *ennreal* (*norm* (*f x* − *s n x*)) + *norm* (*f x*) ∂*M*)
by (*auto intro*!: *nn-integral-mono simp del*: *ennreal-plus simp add*: *ennreal-plus*[*symmetric*])
(*metis add.commute norm-minus-commute norm-triangle-sub*)
also have ... = *?t n*
by (*rule nn-integral-add*) *auto*
finally show *norm* (*?s n*) ≤ *?t n* **.**
qed
have *?t* ⟶ *0* + (∫⁺ *x*. *ennreal* (*norm* (*f x*)) ∂*M*)
using *has-bochner-integral-implies-finite-norm*[*OF i*]
by (*intro tendsto-add tendsto-const lim*)
then show *?t* ⟶ ∫⁺ *x*. *ennreal* (*norm* (*f x*)) ∂*M*
by *simp*
qed
qed

lemma *has-bochner-integral-eq*:
has-bochner-integral M f x ⟹ *has-bochner-integral M f y* ⟹ *x* = *y*
proof (*elim has-bochner-integral.cases*)
assume *f*[*measurable*]: *f* ∈ *borel-measurable M*

fix *s t*
assume (λ*i*. ∫⁺ *x*. *norm* (*f x* − *s i x*) ∂*M*) ⟶ *0* (**is** *?S* ⟶ *0*)
assume (λ*i*. ∫⁺ *x*. *norm* (*f x* − *t i x*) ∂*M*) ⟶ *0* (**is** *?T* ⟶ *0*)
assume *s*: ⋀*i*. *simple-bochner-integrable M* (*s i*)
assume *t*: ⋀*i*. *simple-bochner-integrable M* (*t i*)

```
  have [measurable]: ⋀i. s i ∈ borel-measurable M ⋀i. t i ∈ borel-measurable M
   using s t by (auto intro: borel-measurable-simple-function elim: simple-bochner-integrable.cases)

  let ?s = λi. simple-bochner-integral M (s i)
  let ?t = λi. simple-bochner-integral M (t i)
  assume ?s ⟶ x ?t ⟶ y
  then have (λi. norm (?s i − ?t i)) ⟶ norm (x − y)
    by (intro tendsto-intros)
  moreover
  have (λi. ennreal (norm (?s i − ?t i))) ⟶ ennreal 0
  proof (rule tendsto-sandwich)
    show eventually (λi. 0 ≤ ennreal (norm (?s i − ?t i))) sequentially (λ-. 0)
⟶ ennreal 0
      by auto

    show eventually (λi. norm (?s i − ?t i) ≤ ?S i + ?T i) sequentially
      by (intro always-eventually allI simple-bochner-integral-bounded s t f)
    show (λi. ?S i + ?T i) ⟶ ennreal 0
      using tendsto-add[OF ‹?S ⟶ 0› ‹?T ⟶ 0›] by simp
  qed
  then have (λi. norm (?s i − ?t i)) ⟶ 0
    by (simp add: ennreal-0[symmetric] del: ennreal-0)
  ultimately have norm (x − y) = 0
    by (rule LIMSEQ-unique)
  then show x = y by simp
qed

lemma has-bochner-integralI-AE:
  assumes f: has-bochner-integral M f x
    and g: g ∈ borel-measurable M
    and ae: AE x in M. f x = g x
  shows has-bochner-integral M g x
  using f
proof (safe intro!: has-bochner-integral.intros elim!: has-bochner-integral.cases)
  fix s assume (λi. ∫⁺ x. ennreal (norm (f x − s i x)) ∂M) ⟶ 0
  also have (λi. ∫⁺ x. ennreal (norm (f x − s i x)) ∂M) = (λi. ∫⁺ x. ennreal
(norm (g x − s i x)) ∂M)
    using ae
    by (intro ext nn-integral-cong-AE, eventually-elim) simp
  finally show (λi. ∫⁺ x. ennreal (norm (g x − s i x)) ∂M) ⟶ 0 .
qed (auto intro: g)

lemma has-bochner-integral-eq-AE:
  assumes f: has-bochner-integral M f x
    and g: has-bochner-integral M g y
    and ae: AE x in M. f x = g x
  shows x = y
proof −
```

from *assms* **have** *has-bochner-integral M g x*
by (*auto intro*: *has-bochner-integralI-AE*)
from *this g* **show** $x = y$
by (*rule has-bochner-integral-eq*)
qed

lemma *simple-bochner-integrable-restrict-space*:
fixes f :: - ⇒ $'b$::*real-normed-vector*
assumes Ω: $\Omega \cap$ *space M* ∈ *sets M*
shows *simple-bochner-integrable* (*restrict-space M* Ω) *f* ⟷
simple-bochner-integrable M (λ*x*. *indicator* Ω *x* ∗$_R$ *f x*)
by (*simp add*: *simple-bochner-integrable.simps space-restrict-space*
simple-function-restrict-space[*OF* Ω] *emeasure-restrict-space*[*OF* Ω] *Collect-restrict*
indicator-eq-0-iff conj-ac)

lemma *simple-bochner-integral-restrict-space*:
fixes f :: - ⇒ $'b$::*real-normed-vector*
assumes Ω: $\Omega \cap$ *space M* ∈ *sets M*
assumes *f*: *simple-bochner-integrable* (*restrict-space M* Ω) *f*
shows *simple-bochner-integral* (*restrict-space M* Ω) *f* =
simple-bochner-integral M (λ*x*. *indicator* Ω *x* ∗$_R$ *f x*)
proof −
have *finite* ((λ*x*. *indicator* Ω *x* ∗$_R$ *f x*)‘*space M*)
using *f simple-bochner-integrable-restrict-space*[*OF* Ω, *of f*]
by (*simp add*: *simple-bochner-integrable.simps simple-function-def*)
then show *?thesis*
by (*auto simp*: *space-restrict-space measure-restrict-space*[*OF* Ω(*1*)] *le-infI2*
simple-bochner-integral-def Collect-restrict
split: *split-indicator split-indicator-asm*
intro!: *setsum.mono-neutral-cong-left arg-cong2*[**where** *f*=*measure*])
qed

context
notes [[*inductive-internals*]]
begin

inductive *integrable* **for** *M f* **where**
has-bochner-integral M f x ⟹ *integrable M f*

end

definition *lebesgue-integral* (*integral*^{L}) **where**
integral^L *M f* = (*if* ∃ *x*. *has-bochner-integral M f x then THE x*. *has-bochner-integral M f x else 0*)

syntax
-lebesgue-integral :: *pttrn* ⇒ *real* ⇒ $'a$ *measure* ⇒ *real* (∫ ((*2* -./ -)/ ∂-) [*60*,*61*] *110*)

translations
$\int x.\ f\ \partial M == CONST\ lebesgue\text{-}integral\ M\ (\lambda x.\ f)$

syntax
-ascii-lebesgue-integral :: *pttrn* ⇒ *'a measure* ⇒ *real* ⇒ *real* ((*3LINT (1-)/|(-)./ -) [0,110,60] 60*)

translations
$LINT\ x|M.\ f == CONST\ lebesgue\text{-}integral\ M\ (\lambda x.\ f)$

lemma *has-bochner-integral-integral-eq*: *has-bochner-integral M f x* ⟹ $integral^L$ *M f = x*
by (*metis the-equality has-bochner-integral-eq lebesgue-integral-def*)

lemma *has-bochner-integral-integrable*:
integrable M f ⟹ *has-bochner-integral M f* ($integral^L$ *M f*)
by (*auto simp*: *has-bochner-integral-integral-eq integrable.simps*)

lemma *has-bochner-integral-iff*:
has-bochner-integral M f x ⟷ *integrable M f* ∧ $integral^L$ *M f = x*
by (*metis has-bochner-integral-integrable has-bochner-integral-integral-eq integrable.intros*)

lemma *simple-bochner-integrable-eq-integral*:
simple-bochner-integrable M f ⟹ *simple-bochner-integral M f* = $integral^L$ *M f*
using *has-bochner-integral-simple-bochner-integrable*[*of M f*]
by (*simp add*: *has-bochner-integral-integral-eq*)

lemma *not-integrable-integral-eq*: ¬ *integrable M f* ⟹ $integral^L$ *M f = 0*
unfolding *integrable.simps lebesgue-integral-def* **by** (*auto intro*!: *arg-cong*[**where** *f=The*])

lemma *integral-eq-cases*:
integrable M f ⟷ *integrable N g* ⟹
(*integrable M f* ⟹ *integrable N g* ⟹ $integral^L$ *M f* = $integral^L$ *N g*) ⟹
$integral^L$ *M f* = $integral^L$ *N g*
by (*metis not-integrable-integral-eq*)

lemma *borel-measurable-integrable*[*measurable-dest*]: *integrable M f* ⟹ *f* ∈ *borel-measurable M*
by (*auto elim*: *integrable.cases has-bochner-integral.cases*)

lemma *borel-measurable-integrable'*[*measurable-dest*]:
integrable M f ⟹ *g* ∈ *measurable N M* ⟹ (λ*x*. *f* (*g x*)) ∈ *borel-measurable N*
using *borel-measurable-integrable*[*measurable*] **by** *measurable*

lemma *integrable-cong*:
M = N ⟹ (⋀*x*. *x* ∈ *space N* ⟹ *f x = g x*) ⟹ *integrable M f* ⟷ *integrable N g*
using *assms* **by** (*simp cong*: *has-bochner-integral-cong add*: *integrable.simps*)

lemma *integrable-cong-AE*:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹ *AE x in M*. *f x* = *g x* ⟹
integrable M f ⟷ *integrable M g*
unfolding *integrable.simps*
by (*intro has-bochner-integral-cong-AE arg-cong*[**where** *f*=*Ex*] *ext*)

lemma *integral-cong*:
M = *N* ⟹ (⋀*x*. *x* ∈ *space N* ⟹ *f x* = *g x*) ⟹ *integral*^{L} *M f* = *integral*^{L} *N g*
using *assms* **by** (*simp cong*: *has-bochner-integral-cong cong del*: *if-cong add*: *lebesgue-integral-def*)

lemma *integral-cong-AE*:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹ *AE x in M*. *f x* = *g x* ⟹
integral^L *M f* = *integral*^{L} *M g*
unfolding *lebesgue-integral-def*
by (*rule arg-cong*[**where** *x*=*has-bochner-integral M f*]) (*intro has-bochner-integral-cong-AE ext*)

lemma *integrable-add*[*simp*, *intro*]: *integrable M f* ⟹ *integrable M g* ⟹ *integrable M* (λ*x*. *f x* + *g x*)
by (*auto simp*: *integrable.simps*)

lemma *integrable-zero*[*simp*, *intro*]: *integrable M* (λ*x*. *0*)
by (*metis has-bochner-integral-zero integrable.simps*)

lemma *integrable-setsum*[*simp*, *intro*]: (⋀*i*. *i* ∈ *I* ⟹ *integrable M* (*f i*)) ⟹ *integrable M* (λ*x*. ∑ *i*∈*I*. *f i x*)
by (*metis has-bochner-integral-setsum integrable.simps*)

lemma *integrable-indicator*[*simp*, *intro*]: *A* ∈ *sets M* ⟹ *emeasure M A* < ∞ ⟹
integrable M (λ*x*. *indicator A x* ∗$_R$ *c*)
by (*metis has-bochner-integral-indicator integrable.simps*)

lemma *integrable-real-indicator*[*simp*, *intro*]: *A* ∈ *sets M* ⟹ *emeasure M A* < ∞ ⟹
integrable M (*indicator A* :: *'a* ⇒ *real*)
by (*metis has-bochner-integral-real-indicator integrable.simps*)

lemma *integrable-diff*[*simp*, *intro*]: *integrable M f* ⟹ *integrable M g* ⟹ *integrable M* (λ*x*. *f x* − *g x*)
by (*auto simp*: *integrable.simps intro*: *has-bochner-integral-diff*)

lemma *integrable-bounded-linear*: *bounded-linear T* ⟹ *integrable M f* ⟹ *integrable M* (λ*x*. *T* (*f x*))
by (*auto simp*: *integrable.simps intro*: *has-bochner-integral-bounded-linear*)

lemma *integrable-scaleR-left*[*simp*, *intro*]: $(c \neq 0 \Longrightarrow \textit{integrable}\ M\ f) \Longrightarrow \textit{integrable}\ M\ (\lambda x.\ f\ x *_R c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable-scaleR-right*[*simp*, *intro*]: $(c \neq 0 \Longrightarrow \textit{integrable}\ M\ f) \Longrightarrow \textit{integrable}\ M\ (\lambda x.\ c *_R f\ x)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable-mult-left*[*simp*, *intro*]:
fixes c :: -::{*real-normed-algebra*,*second-countable-topology*}
shows $(c \neq 0 \Longrightarrow \textit{integrable}\ M\ f) \Longrightarrow \textit{integrable}\ M\ (\lambda x.\ f\ x * c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable-mult-right*[*simp*, *intro*]:
fixes c :: -::{*real-normed-algebra*,*second-countable-topology*}
shows $(c \neq 0 \Longrightarrow \textit{integrable}\ M\ f) \Longrightarrow \textit{integrable}\ M\ (\lambda x.\ c * f\ x)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable-divide-zero*[*simp*, *intro*]:
fixes c :: -::{*real-normed-field*, *field*, *second-countable-topology*}
shows $(c \neq 0 \Longrightarrow \textit{integrable}\ M\ f) \Longrightarrow \textit{integrable}\ M\ (\lambda x.\ f\ x\ /\ c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable-inner-left*[*simp*, *intro*]:
$(c \neq 0 \Longrightarrow \textit{integrable}\ M\ f) \Longrightarrow \textit{integrable}\ M\ (\lambda x.\ f\ x \cdot c)$
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable-inner-right*[*simp*, *intro*]:
$(c \neq 0 \Longrightarrow \textit{integrable}\ M\ f) \Longrightarrow \textit{integrable}\ M\ (\lambda x.\ c \cdot f\ x)$
unfolding *integrable.simps* **by** *fastforce*

lemmas *integrable-minus*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-minus*[*OF bounded-linear-ident*]]
lemmas *integrable-divide*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-divide*]
lemmas *integrable-Re*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-Re*]
lemmas *integrable-Im*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-Im*]
lemmas *integrable-cnj*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-cnj*]
lemmas *integrable-of-real*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-of-real*]
lemmas *integrable-fst*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-fst*]
lemmas *integrable-snd*[*simp*, *intro*] =
integrable-bounded-linear[*OF bounded-linear-snd*]

lemma *integral-zero*[*simp*]: $integral^L\ M\ (\lambda x.\ 0) = 0$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-zero*)

lemma *integral-add*[*simp*]: *integrable* $M\ f \Longrightarrow$ *integrable* $M\ g \Longrightarrow$
$integral^L\ M\ (\lambda x.\ f\ x + g\ x) = integral^L\ M\ f + integral^L\ M\ g$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-add has-bochner-integral-integrable*)

lemma *integral-diff*[*simp*]: *integrable* $M\ f \Longrightarrow$ *integrable* $M\ g \Longrightarrow$
$integral^L\ M\ (\lambda x.\ f\ x - g\ x) = integral^L\ M\ f - integral^L\ M\ g$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-diff has-bochner-integral-integrable*)

lemma *integral-setsum*: $(\bigwedge i.\ i \in I \Longrightarrow$ *integrable* $M\ (f\ i)) \Longrightarrow$
$integral^L\ M\ (\lambda x.\ \sum i{\in}I.\ f\ i\ x) = (\sum i{\in}I.\ integral^L\ M\ (f\ i))$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-setsum has-bochner-integral-integrable*)

lemma *integral-setsum'*[*simp*]: $(\bigwedge i.\ i \in I$ =*simp*=> *integrable* $M\ (f\ i)) \Longrightarrow$
$integral^L\ M\ (\lambda x.\ \sum i{\in}I.\ f\ i\ x) = (\sum i{\in}I.\ integral^L\ M\ (f\ i))$
unfolding *simp-implies-def* **by** (*rule integral-setsum*)

lemma *integral-bounded-linear*: *bounded-linear* $T \Longrightarrow$ *integrable* $M\ f \Longrightarrow$
$integral^L\ M\ (\lambda x.\ T\ (f\ x)) = T\ (integral^L\ M\ f)$
by (*metis has-bochner-integral-bounded-linear has-bochner-integral-integrable has-bochner-integral-integral-eq*)

lemma *integral-bounded-linear'*:
assumes T: *bounded-linear* T **and** T': *bounded-linear* T'
assumes $*$: $\neg\ (\forall x.\ T\ x = 0) \Longrightarrow (\forall x.\ T'\ (T\ x) = x)$
shows $integral^L\ M\ (\lambda x.\ T\ (f\ x)) = T\ (integral^L\ M\ f)$
proof *cases*
assume $(\forall x.\ T\ x = 0)$ **then show** *?thesis*
by *simp*
next
assume $**$: $\neg\ (\forall x.\ T\ x = 0)$
show *?thesis*
proof *cases*
assume *integrable* $M\ f$ **with** T **show** *?thesis*
by (*rule integral-bounded-linear*)
next
assume *not*: $\neg$ *integrable* $M\ f$
moreover have $\neg$ *integrable* $M\ (\lambda x.\ T\ (f\ x))$
proof
assume *integrable* $M\ (\lambda x.\ T\ (f\ x))$
from *integrable-bounded-linear*[*OF* T' *this*] *not* $*$[*OF* $**$]
show *False*
by *auto*
qed
ultimately show *?thesis*
using T **by** (*simp add*: *not-integrable-integral-eq linear-simps*)
qed
qed

lemma *integral-scaleR-left*[*simp*]: $(c \neq 0 \Longrightarrow \mathit{integrable}\ M\ f) \Longrightarrow (\int\ x.\ f\ x *_R c\ \partial M) = \mathit{integral}^L\ M\ f *_R c$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-integrable has-bochner-integral-scaleR-left*)

lemma *integral-scaleR-right*[*simp*]: $(\int\ x.\ c *_R f\ x\ \partial M) = c *_R \mathit{integral}^L\ M\ f$
by (*rule integral-bounded-linear'*[*OF bounded-linear-scaleR-right bounded-linear-scaleR-right*[*of 1 / c*]]) *simp*

lemma *integral-mult-left*[*simp*]:
fixes c :: -::{*real-normed-algebra*,*second-countable-topology*}
shows $(c \neq 0 \Longrightarrow \mathit{integrable}\ M\ f) \Longrightarrow (\int\ x.\ f\ x * c\ \partial M) = \mathit{integral}^L\ M\ f * c$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-integrable has-bochner-integral-mult-left*)

lemma *integral-mult-right*[*simp*]:
fixes c :: -::{*real-normed-algebra*,*second-countable-topology*}
shows $(c \neq 0 \Longrightarrow \mathit{integrable}\ M\ f) \Longrightarrow (\int\ x.\ c * f\ x\ \partial M) = c * \mathit{integral}^L\ M\ f$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-integrable has-bochner-integral-mult-right*)

lemma *integral-mult-left-zero*[*simp*]:
fixes c :: -::{*real-normed-field*,*second-countable-topology*}
shows $(\int\ x.\ f\ x * c\ \partial M) = \mathit{integral}^L\ M\ f * c$
by (*rule integral-bounded-linear'*[*OF bounded-linear-mult-left bounded-linear-mult-left*[*of 1 / c*]]) *simp*

lemma *integral-mult-right-zero*[*simp*]:
fixes c :: -::{*real-normed-field*,*second-countable-topology*}
shows $(\int\ x.\ c * f\ x\ \partial M) = c * \mathit{integral}^L\ M\ f$
by (*rule integral-bounded-linear'*[*OF bounded-linear-mult-right bounded-linear-mult-right*[*of 1 / c*]]) *simp*

lemma *integral-inner-left*[*simp*]: $(c \neq 0 \Longrightarrow \mathit{integrable}\ M\ f) \Longrightarrow (\int\ x.\ f\ x \cdot c\ \partial M) = \mathit{integral}^L\ M\ f \cdot c$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-integrable has-bochner-integral-inner-left*)

lemma *integral-inner-right*[*simp*]: $(c \neq 0 \Longrightarrow \mathit{integrable}\ M\ f) \Longrightarrow (\int\ x.\ c \cdot f\ x\ \partial M) = c \cdot \mathit{integral}^L\ M\ f$
by (*intro has-bochner-integral-integral-eq has-bochner-integral-integrable has-bochner-integral-inner-right*)

lemma *integral-divide-zero*[*simp*]:
fixes c :: -::{*real-normed-field*, *field*, *second-countable-topology*}
shows $\mathit{integral}^L\ M\ (\lambda x.\ f\ x\ /\ c) = \mathit{integral}^L\ M\ f\ /\ c$
by (*rule integral-bounded-linear'*[*OF bounded-linear-divide bounded-linear-mult-left*[*of c*]]) *simp*

lemma *integral-minus*[*simp*]: $\mathit{integral}^L\ M\ (\lambda x.\ -\ f\ x) = -\ \mathit{integral}^L\ M\ f$
by (*rule integral-bounded-linear'*[*OF bounded-linear-minus*[*OF bounded-linear-ident*] *bounded-linear-minus*[*OF bounded-linear-ident*]]) *simp*

lemma *integral-complex-of-real*[*simp*]: $integral^L\ M\ (\lambda x.\ complex\text{-}of\text{-}real\ (f\ x)) = of\text{-}real\ (integral^L\ M\ f)$
by (*rule integral-bounded-linear'*[*OF bounded-linear-of-real bounded-linear-Re*]) *simp*

lemma *integral-cnj*[*simp*]: $integral^L\ M\ (\lambda x.\ cnj\ (f\ x)) = cnj\ (integral^L\ M\ f)$
by (*rule integral-bounded-linear'*[*OF bounded-linear-cnj bounded-linear-cnj*]) *simp*

lemmas *integral-divide*[*simp*] =
integral-bounded-linear[*OF bounded-linear-divide*]
lemmas *integral-Re*[*simp*] =
integral-bounded-linear[*OF bounded-linear-Re*]
lemmas *integral-Im*[*simp*] =
integral-bounded-linear[*OF bounded-linear-Im*]
lemmas *integral-of-real*[*simp*] =
integral-bounded-linear[*OF bounded-linear-of-real*]
lemmas *integral-fst*[*simp*] =
integral-bounded-linear[*OF bounded-linear-fst*]
lemmas *integral-snd*[*simp*] =
integral-bounded-linear[*OF bounded-linear-snd*]

lemma *integral-norm-bound-ennreal*:
$integrable\ M\ f \Longrightarrow norm\ (integral^L\ M\ f) \leq (\int^+ x.\ norm\ (f\ x)\ \partial M)$
by (*metis has-bochner-integral-integrable has-bochner-integral-norm-bound*)

lemma *integrableI-sequence*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: $f \in borel\text{-}measurable\ M$
assumes *s*: $\bigwedge i.\ simple\text{-}bochner\text{-}integrable\ M\ (s\ i)$
assumes *lim*: $(\lambda i.\ \int^+ x.\ norm\ (f\ x - s\ i\ x)\ \partial M) \longrightarrow 0$ (**is** $?S \longrightarrow 0$)
shows $integrable\ M\ f$
proof −
let $?s = \lambda n.\ simple\text{-}bochner\text{-}integral\ M\ (s\ n)$

have $\exists x.\ ?s \longrightarrow x$
unfolding *convergent-eq-cauchy*
proof (*rule metric-CauchyI*)
fix *e* :: *real* **assume** $0 < e$
then have $0 < ennreal\ (e\ /\ 2)$ **by** *auto*
from *order-tendstoD*(2)[*OF lim this*]
obtain *M* **where** *M*: $\bigwedge n.\ M \leq n \Longrightarrow ?S\ n < e\ /\ 2$
by (*auto simp*: *eventually-sequentially*)
show $\exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ dist\ (?s\ m)\ (?s\ n) < e$
proof (*intro exI allI impI*)
fix *m n* **assume** *m*: $M \leq m$ **and** *n*: $M \leq n$
have $?S\ n \neq \infty$
using *M*[*OF n*] **by** *auto*
have $norm\ (?s\ n - ?s\ m) \leq ?S\ n + ?S\ m$
by (*intro simple-bochner-integral-bounded s f*)

```
    also have ... < ennreal (e / 2) + e / 2
      by (intro add-strict-mono M n m)
    also have ... = e using ‹0<e› by (simp del: ennreal-plus add: ennreal-plus[symmetric])
    finally show dist (?s n) (?s m) < e
      using ‹0<e› by (simp add: dist-norm ennreal-less-iff)
  qed
 qed
 then obtain x where ?s ⟶ x ..
 show ?thesis
  by (rule, rule) fact+
qed

lemma nn-integral-dominated-convergence-norm:
 fixes u' :: - ⇒ -::{real-normed-vector, second-countable-topology}
 assumes [measurable]:
     ⋀i. u i ∈ borel-measurable M u' ∈ borel-measurable M w ∈ borel-measurable
M
  and bound: ⋀j. AE x in M. norm (u j x) ≤ w x
  and w: (∫+x. w x ∂M) < ∞
  and u': AE x in M. (λi. u i x) ⟶ u' x
 shows (λi. (∫+x. norm (u' x − u i x) ∂M)) ⟶ 0
proof −
 have AE x in M. ∀j. norm (u j x) ≤ w x
  unfolding AE-all-countable by rule fact
 with u' have bnd: AE x in M. ∀j. norm (u' x − u j x) ≤ 2 ∗ w x
 proof (eventually-elim, intro allI)
  fix i x assume (λi. u i x) ⟶ u' x ∀j. norm (u j x) ≤ w x ∀j. norm (u j
x) ≤ w x
  then have norm (u' x) ≤ w x norm (u i x) ≤ w x
    by (auto intro: LIMSEQ-le-const2 tendsto-norm)
  then have norm (u' x) + norm (u i x) ≤ 2 ∗ w x
    by simp
  also have norm (u' x − u i x) ≤ norm (u' x) + norm (u i x)
    by (rule norm-triangle-ineq4)
  finally (xtrans) show norm (u' x − u i x) ≤ 2 ∗ w x .
 qed
 have w-nonneg: AE x in M. 0 ≤ w x
  using bound[of 0] by (auto intro: order-trans[OF norm-ge-zero])

 have (λi. (∫+x. norm (u' x − u i x) ∂M)) ⟶ (∫+x. 0 ∂M)
 proof (rule nn-integral-dominated-convergence)
  show (∫+x. 2 ∗ w x ∂M) < ∞
    by (rule nn-integral-mult-bounded-inf[OF - w, of 2]) (insert w-nonneg, auto
simp: ennreal-mult )
  show AE x in M. (λi. ennreal (norm (u' x − u i x))) ⟶ 0
    using u'
  proof eventually-elim
    fix x assume (λi. u i x) ⟶ u' x
    from tendsto-diff[OF tendsto-const[of u' x] this]
```

```
    show (λi. ennreal (norm (u' x − u i x))) ⟶ 0
      by (simp add: tendsto-norm-zero-iff ennreal-0[symmetric] del: ennreal-0)
  qed
 qed (insert bnd w-nonneg, auto)
 then show ?thesis by simp
qed

lemma integrableI-bounded:
 fixes f :: 'a ⇒ 'b::{banach, second-countable-topology}
 assumes f[measurable]: f ∈ borel-measurable M and fin: (∫⁺x. norm (f x) ∂M)
< ∞
 shows integrable M f
proof −
 from borel-measurable-implies-sequence-metric[OF f, of 0] obtain s where
   s: ⋀i. simple-function M (s i) and
   pointwise: ⋀x. x ∈ space M ⟹ (λi. s i x) ⟶ f x and
   bound: ⋀i x. x ∈ space M ⟹ norm (s i x) ≤ 2 * norm (f x)
   by simp metis

 show ?thesis
 proof (rule integrableI-sequence)
   { fix i
     have (∫⁺x. norm (s i x) ∂M) ≤ (∫⁺x. ennreal (2 * norm (f x)) ∂M)
       by (intro nn-integral-mono) (simp add: bound)
     also have ... = 2 * (∫⁺x. ennreal (norm (f x)) ∂M)
       by (simp add: ennreal-mult nn-integral-cmult)
     also have ... < top
       using fin by (simp add: ennreal-mult-less-top)
     finally have (∫⁺x. norm (s i x) ∂M) < ∞
       by simp }
   note fin-s = this

   show ⋀i. simple-bochner-integrable M (s i)
     by (rule simple-bochner-integrableI-bounded) fact+

   show (λi. ∫⁺ x. ennreal (norm (f x − s i x)) ∂M) ⟶ 0
   proof (rule nn-integral-dominated-convergence-norm)
     show ⋀j. AE x in M. norm (s j x) ≤ 2 * norm (f x)
       using bound by auto
     show ⋀i. s i ∈ borel-measurable M (λx. 2 * norm (f x)) ∈ borel-measurable
M
       using s by (auto intro: borel-measurable-simple-function)
     show (∫⁺ x. ennreal (2 * norm (f x)) ∂M) < ∞
      using fin by (simp add: nn-integral-cmult ennreal-mult ennreal-mult-less-top)
     show AE x in M. (λi. s i x) ⟶ f x
       using pointwise by auto
   qed fact
 qed fact
qed
```

lemma *integrableI-bounded-set*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
assumes [*measurable*]: $A \in sets\ M\ f \in borel\text{-}measurable\ M$
assumes *finite*: $emeasure\ M\ A < \infty$
and *bnd*: $AE\ x\ in\ M.\ x \in A \longrightarrow norm\ (f\ x) \leq B$
and *null*: $AE\ x\ in\ M.\ x \notin A \longrightarrow f\ x = 0$
shows *integrable M f*
proof (*rule integrableI-bounded*)
{ fix $x :: 'b$ **have** $norm\ x \leq B \Longrightarrow 0 \leq B$
using *norm-ge-zero*[*of x*] **by** *arith* **}**
with *bnd null* **have** $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M) \leq (\int^+ x.\ ennreal\ (max\ 0\ B) * indicator\ A\ x\ \partial M)$
by (*intro nn-integral-mono-AE*) (*auto split*: *split-indicator split-max*)
also have $\ldots < \infty$
using *finite* **by** (*subst nn-integral-cmult-indicator*) (*auto simp*: *ennreal-mult-less-top*)
finally show $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M) < \infty$ **.**
qed *simp*

lemma *integrableI-bounded-set-indicator*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
shows $A \in sets\ M \Longrightarrow f \in borel\text{-}measurable\ M \Longrightarrow$
$emeasure\ M\ A < \infty \Longrightarrow (AE\ x\ in\ M.\ x \in A \longrightarrow norm\ (f\ x) \leq B) \Longrightarrow$
$integrable\ M\ (\lambda x.\ indicator\ A\ x *_R f\ x)$
by (*rule integrableI-bounded-set*[**where** *A*=*A*]) *auto*

lemma *integrableI-nonneg*:
fixes $f :: 'a \Rightarrow real$
assumes $f \in borel\text{-}measurable\ M\ AE\ x\ in\ M.\ 0 \leq f\ x\ (\int^+ x.\ f\ x\ \partial M) < \infty$
shows *integrable M f*
proof −
have $(\int^+ x.\ norm\ (f\ x)\ \partial M) = (\int^+ x.\ f\ x\ \partial M)$
using *assms* **by** (*intro nn-integral-cong-AE*) *auto*
then show *?thesis*
using *assms* **by** (*intro integrableI-bounded*) *auto*
qed

lemma *integrable-iff-bounded*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
shows $integrable\ M\ f \longleftrightarrow f \in borel\text{-}measurable\ M \wedge (\int^+ x.\ norm\ (f\ x)\ \partial M) < \infty$
using *integrableI-bounded*[*of f M*] *has-bochner-integral-implies-finite-norm*[*of M f*]
unfolding *integrable.simps has-bochner-integral.simps*[*abs-def*] **by** *auto*

lemma *integrable-bound*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
and $g :: 'a \Rightarrow 'c::\{banach, second\text{-}countable\text{-}topology\}$
shows $integrable\ M\ f \Longrightarrow g \in borel\text{-}measurable\ M \Longrightarrow (AE\ x\ in\ M.\ norm\ (g\ x)$

$\leq$ *norm* (*f x*)) $\Longrightarrow$
integrable M g
unfolding *integrable-iff-bounded*
proof *safe*
assume *f* $\in$ *borel-measurable M g* $\in$ *borel-measurable M*
assume *AE x in M*. *norm* (*g x*) $\leq$ *norm* (*f x*)
then have ($\int^+$ *x*. *ennreal* (*norm* (*g x*)) ∂M) $\leq$ ($\int^+$ *x*. *ennreal* (*norm* (*f x*)) ∂M)
by (*intro nn-integral-mono-AE*) *auto*
also assume ($\int^+$ *x*. *ennreal* (*norm* (*f x*)) ∂M) $< \infty$
finally show ($\int^+$ *x*. *ennreal* (*norm* (*g x*)) ∂M) $< \infty$ **.**
qed

lemma *integrable-mult-indicator*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
shows *A* $\in$ *sets M* $\Longrightarrow$ *integrable M f* $\Longrightarrow$ *integrable M* (λx. *indicator A x* $*_R$ *f x*)
by (*rule integrable-bound*[*of M f*]) (*auto split*: *split-indicator*)

lemma *integrable-real-mult-indicator*:
fixes *f* :: *'a* $\Rightarrow$ *real*
shows *A* $\in$ *sets M* $\Longrightarrow$ *integrable M f* $\Longrightarrow$ *integrable M* (λx. *f x* $*$ *indicator A x*)
using *integrable-mult-indicator*[*of A M f*] **by** (*simp add*: *mult-ac*)

lemma *integrable-abs*[*simp*, *intro*]:
fixes *f* :: *'a* $\Rightarrow$ *real*
assumes [*measurable*]: *integrable M f* **shows** *integrable M* (λx. |*f x*|)
using *assms* **by** (*rule integrable-bound*) *auto*

lemma *integrable-norm*[*simp*, *intro*]:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
assumes [*measurable*]: *integrable M f* **shows** *integrable M* (λx. *norm* (*f x*))
using *assms* **by** (*rule integrable-bound*) *auto*

lemma *integrable-norm-cancel*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
assumes [*measurable*]: *integrable M* (λx. *norm* (*f x*)) *f* $\in$ *borel-measurable M*
shows *integrable M f*
using *assms* **by** (*rule integrable-bound*) *auto*

lemma *integrable-norm-iff*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
shows *f* $\in$ *borel-measurable M* $\Longrightarrow$ *integrable M* (λx. *norm* (*f x*)) $\longleftrightarrow$ *integrable M f*
by (*auto intro*: *integrable-norm-cancel*)

lemma *integrable-abs-cancel*:
fixes *f* :: *'a* $\Rightarrow$ *real*

assumes [*measurable*]: *integrable M* (λ*x*. |*f x*|) *f* ∈ *borel-measurable M* **shows** *integrable M f*
using *assms* **by** (*rule integrable-bound*) *auto*

lemma *integrable-abs-iff*:
fixes *f* :: *'a* ⇒ *real*
shows *f* ∈ *borel-measurable M* ⟹ *integrable M* (λ*x*. |*f x*|) ⟷ *integrable M f*
by (*auto intro*: *integrable-abs-cancel*)

lemma *integrable-max*[*simp*, *intro*]:
fixes *f* :: *'a* ⇒ *real*
assumes *fg*[*measurable*]: *integrable M f integrable M g*
shows *integrable M* (λ*x*. *max* (*f x*) (*g x*))
using *integrable-add*[*OF integrable-norm*[*OF fg*(*1*)] *integrable-norm*[*OF fg*(*2*)]]
by (*rule integrable-bound*) *auto*

lemma *integrable-min*[*simp*, *intro*]:
fixes *f* :: *'a* ⇒ *real*
assumes *fg*[*measurable*]: *integrable M f integrable M g*
shows *integrable M* (λ*x*. *min* (*f x*) (*g x*))
using *integrable-add*[*OF integrable-norm*[*OF fg*(*1*)] *integrable-norm*[*OF fg*(*2*)]]
by (*rule integrable-bound*) *auto*

lemma *integral-minus-iff*[*simp*]:
integrable M (λ*x*. − *f x* ::*'a*::{*banach*, *second-countable-topology*}) ⟷ *integrable M f*
unfolding *integrable-iff-bounded*
by (*auto intro*: *borel-measurable-uminus*[*of* λ*x*. − *f x M*, *simplified*])

lemma *integrable-indicator-iff*:
integrable M (*indicator A*::- ⇒ *real*) ⟷ *A* ∩ *space M* ∈ *sets M* ∧ *emeasure M* (*A* ∩ *space M*) < ∞
by (*simp add*: *integrable-iff-bounded borel-measurable-indicator-iff ennreal-indicator nn-integral-indicator'*
cong: *conj-cong*)

lemma *integral-indicator*[*simp*]: *integral*^{L} *M* (*indicator A*) = *measure M* (*A* ∩ *space M*)
proof *cases*
assume ∗: *A* ∩ *space M* ∈ *sets M* ∧ *emeasure M* (*A* ∩ *space M*) < ∞
have *integral*^{L} *M* (*indicator A*) = *integral*^{L} *M* (*indicator* (*A* ∩ *space M*))
by (*intro integral-cong*) (*auto split*: *split-indicator*)
also have ... = *measure M* (*A* ∩ *space M*)
using ∗ **by** (*intro has-bochner-integral-integral-eq has-bochner-integral-real-indicator*) *auto*
finally show *?thesis* **.**
next
assume ∗: ¬ (*A* ∩ *space M* ∈ *sets M* ∧ *emeasure M* (*A* ∩ *space M*) < ∞)
have *integral*^{L} *M* (*indicator A*) = *integral*^{L} *M* (*indicator* (*A* ∩ *space M*) :: - ⇒

real)
  by (*intro integral-cong*) (*auto split*: *split-indicator*)
 also have ... = *0*
  using ∗ **by** (*subst not-integrable-integral-eq*) (*auto simp*: *integrable-indicator-iff*)
 also have ... = *measure M* (*A* ∩ *space M*)
  using ∗ **by** (*auto simp*: *measure-def emeasure-notin-sets not-less top-unique*)
 finally show *?thesis* **.**
qed

lemma *integrable-discrete-difference*:
 fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
 assumes *X*: *countable X*
 assumes *null*: ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0*
 assumes *sets*: ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
 assumes *eq*: ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *f x* = *g x*
 shows *integrable M f* ⟷ *integrable M g*
 unfolding *integrable-iff-bounded*
proof (*rule conj-cong*)
 { assume *f* ∈ *borel-measurable M* **then have** *g* ∈ *borel-measurable M*
    by (*rule measurable-discrete-difference*[**where** *X*=*X*]) (*auto simp*: *assms*) **}**
 moreover
 { assume *g* ∈ *borel-measurable M* **then have** *f* ∈ *borel-measurable M*
    by (*rule measurable-discrete-difference*[**where** *X*=*X*]) (*auto simp*: *assms*) **}**
 ultimately show *f* ∈ *borel-measurable M* ⟷ *g* ∈ *borel-measurable M* **..**
next
 have *AE x in M*. *x* ∉ *X*
  by (*rule AE-discrete-difference*) *fact+*
 then have (∫⁺ *x*. *norm* (*f x*) ∂*M*) = (∫⁺ *x*. *norm* (*g x*) ∂*M*)
  by (*intro nn-integral-cong-AE*) (*auto simp*: *eq*)
 then show (∫⁺ *x*. *norm* (*f x*) ∂*M*) < ∞ ⟷ (∫⁺ *x*. *norm* (*g x*) ∂*M*) < ∞
  by *simp*
qed

lemma *integral-discrete-difference*:
 fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
 assumes *X*: *countable X*
 assumes *null*: ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0*
 assumes *sets*: ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
 assumes *eq*: ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *f x* = *g x*
 shows *integral*^{L} *M f* = *integral*^{L} *M g*
proof (*rule integral-eq-cases*)
 show *eq*: *integrable M f* ⟷ *integrable M g*
  by (*rule integrable-discrete-difference*[**where** *X*=*X*]) *fact+*

 assume *f*: *integrable M f*
 show *integral*^{L} *M f* = *integral*^{L} *M g*
 proof (*rule integral-cong-AE*)
  show *f* ∈ *borel-measurable M g* ∈ *borel-measurable M*
   using *f eq* **by** (*auto intro*: *borel-measurable-integrable*)

```
    have AE x in M. x ∉ X
      by (rule AE-discrete-difference) fact+
    with AE-space show AE x in M. f x = g x
      by eventually-elim fact
  qed
qed

lemma has-bochner-integral-discrete-difference:
  fixes f :: 'a ⇒ 'b::{banach, second-countable-topology}
  assumes X: countable X
  assumes null: ⋀x. x ∈ X ⟹ emeasure M {x} = 0
  assumes sets: ⋀x. x ∈ X ⟹ {x} ∈ sets M
  assumes eq: ⋀x. x ∈ space M ⟹ x ∉ X ⟹ f x = g x
  shows has-bochner-integral M f x ⟷ has-bochner-integral M g x
  using integrable-discrete-difference[of X M f g, OF assms]
  using integral-discrete-difference[of X M f g, OF assms]
  by (metis has-bochner-integral-iff)

lemma
  fixes f :: 'a ⇒ 'b::{banach, second-countable-topology} and w :: 'a ⇒ real
  assumes f ∈ borel-measurable M ⋀i. s i ∈ borel-measurable M integrable M w
  assumes lim: AE x in M. (λi. s i x) ⟶ f x
  assumes bound: ⋀i. AE x in M. norm (s i x) ≤ w x
  shows integrable-dominated-convergence: integrable M f
    and integrable-dominated-convergence2: ⋀i. integrable M (s i)
    and integral-dominated-convergence: (λi. integral^L M (s i)) ⟶ integral^L
M f
proof −
  have w-nonneg: AE x in M. 0 ≤ w x
    using bound[of 0] by eventually-elim (auto intro: norm-ge-zero order-trans)
  then have (∫⁺x. w x ∂M) = (∫⁺x. norm (w x) ∂M)
    by (intro nn-integral-cong-AE) auto
  with ‹integrable M w› have w: w ∈ borel-measurable M (∫⁺x. w x ∂M) < ∞
    unfolding integrable-iff-bounded by auto

  show int-s: ⋀i. integrable M (s i)
    unfolding integrable-iff-bounded
  proof
    fix i
    have (∫⁺ x. ennreal (norm (s i x)) ∂M) ≤ (∫⁺x. w x ∂M)
      using bound[of i] w-nonneg by (intro nn-integral-mono-AE) auto
    with w show (∫⁺ x. ennreal (norm (s i x)) ∂M) < ∞ by auto
  qed fact

  have all-bound: AE x in M. ∀i. norm (s i x) ≤ w x
    using bound unfolding AE-all-countable by auto

  show int-f: integrable M f
```

unfolding *integrable-iff-bounded*
proof
have $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M) \leq (\int^+ x.\ w\ x\ \partial M)$
using *all-bound lim w-nonneg*
proof (*intro nn-integral-mono-AE*, *eventually-elim*)
fix x **assume** $\forall i.\ norm\ (s\ i\ x) \leq w\ x\ (\lambda i.\ s\ i\ x) \longrightarrow f\ x\ 0 \leq w\ x$
then show $ennreal\ (norm\ (f\ x)) \leq ennreal\ (w\ x)$
by (*intro LIMSEQ-le-const2*[**where** $X{=}\lambda i.\ ennreal\ (norm\ (s\ i\ x))$]) (*auto intro*: *tendsto-intros*)
qed
with w **show** $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M) < \infty$ **by** *auto*
qed *fact*

have $(\lambda n.\ ennreal\ (norm\ (integral^L\ M\ (s\ n) - integral^L\ M\ f))) \longrightarrow ennreal\ 0$ (**is** $?d \longrightarrow ennreal\ 0$)
proof (*rule tendsto-sandwich*)
show *eventually* $(\lambda n.\ ennreal\ 0 \leq ?d\ n)\ sequentially\ (\lambda\text{-}.\ ennreal\ 0) \longrightarrow ennreal\ 0$ **by** *auto*
show *eventually* $(\lambda n.\ ?d\ n \leq (\int^+ x.\ norm\ (s\ n\ x - f\ x)\ \partial M))\ sequentially$
proof (*intro always-eventually allI*)
fix n
have $?d\ n = norm\ (integral^L\ M\ (\lambda x.\ s\ n\ x - f\ x))$
using *int-f int-s* **by** *simp*
also have $\ldots \leq (\int^+ x.\ norm\ (s\ n\ x - f\ x)\ \partial M)$
by (*intro int-f int-s integrable-diff integral-norm-bound-ennreal*)
finally show $?d\ n \leq (\int^+ x.\ norm\ (s\ n\ x - f\ x)\ \partial M)$ **.**
qed
show $(\lambda n.\ \int^+ x.\ norm\ (s\ n\ x - f\ x)\ \partial M) \longrightarrow ennreal\ 0$
unfolding *ennreal-0*
apply (*subst norm-minus-commute*)
proof (*rule nn-integral-dominated-convergence-norm*[**where** $w{=}w$])
show $\bigwedge n.\ s\ n \in borel\text{-}measurable\ M$
using *int-s* **unfolding** *integrable-iff-bounded* **by** *auto*
qed *fact+*
qed
then have $(\lambda n.\ integral^L\ M\ (s\ n) - integral^L\ M\ f) \longrightarrow 0$
by (*simp add*: *tendsto-norm-zero-iff del*: *ennreal-0*)
from *tendsto-add*[*OF this tendsto-const*[*of integral*^{L} *M f*]]
show $(\lambda i.\ integral^L\ M\ (s\ i)) \longrightarrow integral^L\ M\ f$ **by** *simp*
qed

context
fixes $s :: real \Rightarrow 'a \Rightarrow 'b{::}\{banach, second\text{-}countable\text{-}topology\}$ **and** $w :: 'a \Rightarrow real$
and $f :: 'a \Rightarrow 'b$ **and** M
assumes $f \in borel\text{-}measurable\ M\ \bigwedge t.\ s\ t \in borel\text{-}measurable\ M\ integrable\ M\ w$
assumes *lim*: $AE\ x\ in\ M.\ (\lambda i.\ s\ i\ x) \longrightarrow f\ x)\ at\text{-}top$
assumes *bound*: $\forall_F\ i\ in\ at\text{-}top.\ AE\ x\ in\ M.\ norm\ (s\ i\ x) \leq w\ x$
begin

lemma *integral-dominated-convergence-at-top*: $((\lambda t.\ integral^L\ M\ (s\ t)) \longrightarrow integral^L\ M\ f)\ at\text{-}top$
proof (*rule tendsto-at-topI-sequentially*)
  fix $X :: nat \Rightarrow real$ **assume** X: *filterlim X at-top sequentially*
  from *filterlim-iff*[*THEN iffD1*, *OF this*, *rule-format*, *OF bound*]
  obtain N **where** w: $\bigwedge n.\ N \leq n \Longrightarrow AE\ x\ in\ M.\ norm\ (s\ (X\ n)\ x) \leq w\ x$
    by (*auto simp*: *eventually-sequentially*)

  show $(\lambda n.\ integral^L\ M\ (s\ (X\ n))) \longrightarrow integral^L\ M\ f$
  proof (*rule LIMSEQ-offset*, *rule integral-dominated-convergence*)
    show $AE\ x\ in\ M.\ norm\ (s\ (X\ (n + N))\ x) \leq w\ x$ **for** n
      by (*rule w*) *auto*
    show $AE\ x\ in\ M.\ (\lambda n.\ s\ (X\ (n + N))\ x) \longrightarrow f\ x$
      using *lim*
    proof *eventually-elim*
      fix x **assume** $((\lambda i.\ s\ i\ x) \longrightarrow f\ x)\ at\text{-}top$
      then show $(\lambda n.\ s\ (X\ (n + N))\ x) \longrightarrow f\ x$
        by (*intro LIMSEQ-ignore-initial-segment filterlim-compose*[*OF - X*])
    qed
  qed *fact+*
qed

lemma *integrable-dominated-convergence-at-top*: *integrable M f*
proof −
  from *bound* **obtain** N **where** w: $\bigwedge n.\ N \leq n \Longrightarrow AE\ x\ in\ M.\ norm\ (s\ n\ x) \leq w\ x$
    by (*auto simp*: *eventually-at-top-linorder*)
  show *?thesis*
  proof (*rule integrable-dominated-convergence*)
    show $AE\ x\ in\ M.\ norm\ (s\ (N + i)\ x) \leq w\ x$ **for** $i :: nat$
      by (*intro w*) *auto*
    show $AE\ x\ in\ M.\ (\lambda i.\ s\ (N + real\ i)\ x) \longrightarrow f\ x$
      using *lim*
    proof *eventually-elim*
      fix x **assume** $((\lambda i.\ s\ i\ x) \longrightarrow f\ x)\ at\text{-}top$
      then show $(\lambda n.\ s\ (N + n)\ x) \longrightarrow f\ x$
        by (*rule filterlim-compose*)
          (*auto intro*!: *filterlim-tendsto-add-at-top filterlim-real-sequentially*)
    qed
  qed *fact+*
qed

end

lemma *integrable-mult-left-iff*:
  fixes $f :: 'a \Rightarrow real$
  shows $integrable\ M\ (\lambda x.\ c * f\ x) \longleftrightarrow c = 0 \vee integrable\ M\ f$
  using *integrable-mult-left*[*of c M f*] *integrable-mult-left*[*of 1 / c M* $\lambda x.\ c * f\ x$]

by (*cases* $c = 0$) *auto*

lemma *integrableI-nn-integral-finite*:
assumes [*measurable*]: $f \in$ *borel-measurable* M
and *nonneg*: *AE* x *in* M. $0 \leq f\, x$
and *finite*: $(\int^+ x.\ f\, x\ \partial M) =$ *ennreal* x
shows *integrable* $M\ f$
proof (*rule integrableI-bounded*)
have $(\int^+ x.$ *ennreal* $(norm\ (f\, x))\ \partial M) = (\int^+ x.$ *ennreal* $(f\, x)\ \partial M)$
using *nonneg* **by** (*intro nn-integral-cong-AE*) *auto*
with *finite* **show** $(\int^+ x.$ *ennreal* $(norm\ (f\, x))\ \partial M) < \infty$
by *auto*
qed *simp*

lemma *integral-nonneg-AE*:
fixes $f :: 'a \Rightarrow$ *real*
assumes *nonneg*: *AE* x *in* M. $0 \leq f\, x$
shows $0 \leq$ *integral*^{L} $M\ f$
proof *cases*
assume f: *integrable* $M\ f$
then have [*measurable*]: $f \in M \rightarrow_M$ *borel*
by *auto*
have $(\lambda x.$ *max* $0\ (f\, x)) \in M \rightarrow_M$ *borel* $\bigwedge x.\ 0 \leq$ *max* $0\ (f\, x)$ *integrable* $M\ (\lambda x.$ *max* $0\ (f\, x))$
using f **by** *auto*
from *this* **have** $0 \leq$ *integral*^{L} $M\ (\lambda x.$ *max* $0\ (f\, x))$
proof (*induction rule*: *borel-measurable-induct-real*)
case (*add* $f\ g$)
then have *integrable* $M\ f$ *integrable* $M\ g$
by (*auto intro*!: *integrable-bound*[*OF add.prems*])
with *add* **show** *?case*
by (*simp add*: *nn-integral-add*)
next
case (*seq* U)
show *?case*
proof (*rule LIMSEQ-le-const*)
have *U-le*: $x \in$ *space* $M \Longrightarrow U\ i\ x \leq$ *max* $0\ (f\, x)$ **for** $x\ i$
using *seq* **by** (*intro incseq-le*) (*auto simp*: *incseq-def le-fun-def*)
with *seq nonneg* **show** $(\lambda i.$ *integral*^{L} $M\ (U\ i)) \longrightarrow$ *LINT* $x|M.$ *max* $0\ (f\, x)$
by (*intro integral-dominated-convergence*) *auto*
have *integrable* $M\ (U\ i)$ **for** i
using *seq.prems* **by** (*rule integrable-bound*) (*insert U-le seq*, *auto*)
with *seq* **show** $\exists N.\ \forall n \geq N.\ 0 \leq$ *integral*^{L} $M\ (U\ n)$
by *auto*
qed
qed (*auto simp*: *measure-nonneg integrable-mult-left-iff*)
also have $\ldots =$ *integral*^{L} $M\ f$
using *nonneg* **by** (*auto intro*!: *integral-cong-AE*)

```
  finally show ?thesis .
qed (simp add: not-integrable-integral-eq)

lemma integral-nonneg[simp]:
  fixes f :: 'a ⇒ real
  shows (⋀x. x ∈ space M ⟹ 0 ≤ f x) ⟹ 0 ≤ integral^L M f
  by (intro integral-nonneg-AE) auto

lemma nn-integral-eq-integral:
  assumes f: integrable M f
  assumes nonneg: AE x in M. 0 ≤ f x
  shows (∫⁺ x. f x ∂M) = integral^L M f
proof −
  { fix f :: 'a ⇒ real assume f: f ∈ borel-measurable M ⋀x. 0 ≤ f x integrable
M f
    then have (∫⁺ x. f x ∂M) = integral^L M f
    proof (induct rule: borel-measurable-induct-real)
      case (set A) then show ?case
      by (simp add: integrable-indicator-iff ennreal-indicator emeasure-eq-ennreal-measure)
    next
      case (mult f c) then show ?case
        by (auto simp add: integrable-mult-left-iff nn-integral-cmult ennreal-mult
integral-nonneg-AE)
    next
      case (add g f)
      then have integrable M f integrable M g
        by (auto intro!: integrable-bound[OF add.prems])
      with add show ?case
        by (simp add: nn-integral-add integral-nonneg-AE)
    next
      case (seq U)
      show ?case
      proof (rule LIMSEQ-unique)
        have U-le-f: x ∈ space M ⟹ U i x ≤ f x for x i
          using seq by (intro incseq-le) (auto simp: incseq-def le-fun-def)
        have int-U: ⋀i. integrable M (U i)
          using seq f U-le-f by (intro integrable-bound[OF f(3)]) auto
        from U-le-f seq have (λi. integral^L M (U i)) ⟶ integral^L M f
          by (intro integral-dominated-convergence) auto
        then show (λi. ennreal (integral^L M (U i))) ⟶ ennreal (integral^L M
f)
          using seq f int-U by (simp add: f integral-nonneg-AE)
        have (λi. ∫⁺ x. U i x ∂M) ⟶ ∫⁺ x. f x ∂M
          using seq U-le-f f
            by (intro nn-integral-dominated-convergence[where w=f]) (auto simp:
integrable-iff-bounded)
        then show (λi. ∫ x. U i x ∂M) ⟶ ∫⁺x. f x ∂M
          using seq int-U by simp
      qed
```

qed }
from *this*[*of* $\lambda x.\ max\ 0\ (f\ x)$] *assms* **have** $(\int^+ x.\ max\ 0\ (f\ x)\ \partial M) = integral^L\ M\ (\lambda x.\ max\ 0\ (f\ x))$
by *simp*
also have $\ldots = integral^L\ M\ f$
using *assms* **by** (*auto intro*!: *integral-cong-AE simp*: *integral-nonneg-AE*)
also have $(\int^+ x.\ max\ 0\ (f\ x)\ \partial M) = (\int^+ x.\ f\ x\ \partial M)$
using *assms* **by** (*auto intro*!: *nn-integral-cong-AE simp*: *max-def*)
finally show *?thesis* **.**
qed

lemma
fixes $f :: - \Rightarrow - \Rightarrow 'a :: \{banach, second\text{-}countable\text{-}topology\}$
assumes *integrable*[*measurable*]: $\bigwedge i.\ integrable\ M\ (f\ i)$
and *summable*: $AE\ x\ in\ M.\ summable\ (\lambda i.\ norm\ (f\ i\ x))$
and *sums*: $summable\ (\lambda i.\ (\int x.\ norm\ (f\ i\ x)\ \partial M))$
shows *integrable-suminf*: $integrable\ M\ (\lambda x.\ (\sum i.\ f\ i\ x))$ (**is** $integrable\ M\ ?S$)
and *sums-integral*: $(\lambda i.\ integral^L\ M\ (f\ i))\ sums\ (\int x.\ (\sum i.\ f\ i\ x)\ \partial M)$ (**is** $?f\ sums\ ?x$)
and *integral-suminf*: $(\int x.\ (\sum i.\ f\ i\ x)\ \partial M) = (\sum i.\ integral^L\ M\ (f\ i))$
and *summable-integral*: $summable\ (\lambda i.\ integral^L\ M\ (f\ i))$
proof −
have *1*: $integrable\ M\ (\lambda x.\ \sum i.\ norm\ (f\ i\ x))$
proof (*rule integrableI-bounded*)
have $(\int^+ x.\ ennreal\ (norm\ (\sum i.\ norm\ (f\ i\ x)))\ \partial M) = (\int^+ x.\ (\sum i.\ ennreal\ (norm\ (f\ i\ x)))\ \partial M)$
apply (*intro nn-integral-cong-AE*)
using *summable*
apply *eventually-elim*
apply (*simp add*: *suminf-nonneg ennreal-suminf-neq-top*)
done
also have $\ldots = (\sum i.\ \int^+ x.\ norm\ (f\ i\ x)\ \partial M)$
by (*intro nn-integral-suminf*) *auto*
also have $\ldots = (\sum i.\ ennreal\ (\int x.\ norm\ (f\ i\ x)\ \partial M))$
by (*intro arg-cong*[**where** $f{=}suminf$] *ext nn-integral-eq-integral integrable-norm integrable*) *auto*
finally show $(\int^+ x.\ ennreal\ (norm\ (\sum i.\ norm\ (f\ i\ x)))\ \partial M) < \infty$
by (*simp add*: *sums ennreal-suminf-neq-top less-top*[*symmetric*] *integral-nonneg-AE*)
qed *simp*

have *2*: $AE\ x\ in\ M.\ (\lambda n.\ \sum i{<}n.\ f\ i\ x) \longrightarrow (\sum i.\ f\ i\ x)$
using *summable* **by** *eventually-elim* (*auto intro*: *summable-LIMSEQ summable-norm-cancel*)

have *3*: $\bigwedge j.\ AE\ x\ in\ M.\ norm\ (\sum i{<}j.\ f\ i\ x) \le (\sum i.\ norm\ (f\ i\ x))$
using *summable*
proof *eventually-elim*
fix *j x* **assume** [*simp*]: $summable\ (\lambda i.\ norm\ (f\ i\ x))$
have $norm\ (\sum i{<}j.\ f\ i\ x) \le (\sum i{<}j.\ norm\ (f\ i\ x))$ **by** (*rule norm-setsum*)
also have $\ldots \le (\sum i.\ norm\ (f\ i\ x))$

```
    using setsum-le-suminf[of λi. norm (f i x)] unfolding sums-iff by auto
   finally show norm (∑ i<j. f i x) ≤ (∑ i. norm (f i x)) by simp
  qed

  note ibl = integrable-dominated-convergence[OF - - 1 2 3]
  note int = integral-dominated-convergence[OF - - 1 2 3]

  show integrable M ?S
   by (rule ibl) measurable

  show ?f sums ?x unfolding sums-def
   using int by (simp add: integrable)
  then show ?x = suminf ?f summable ?f
   unfolding sums-iff by auto
qed

lemma integral-norm-bound:
  fixes f :: - ⇒ 'a :: {banach, second-countable-topology}
  shows integrable M f ⟹ norm (integral^L M f) ≤ (∫ x. norm (f x) ∂M)
  using nn-integral-eq-integral[of M λx. norm (f x)]
  using integral-norm-bound-ennreal[of M f] by (simp add: integral-nonneg-AE)

lemma integral-eq-nn-integral:
  assumes [measurable]: f ∈ borel-measurable M
  assumes nonneg: AE x in M. 0 ≤ f x
  shows integral^L M f = enn2real (∫⁺ x. ennreal (f x) ∂M)
proof cases
  assume *: (∫⁺ x. ennreal (f x) ∂M) = ∞
  also have (∫⁺ x. ennreal (f x) ∂M) = (∫⁺ x. ennreal (norm (f x)) ∂M)
   using nonneg by (intro nn-integral-cong-AE) auto
  finally have ¬ integrable M f
   by (auto simp: integrable-iff-bounded)
  then show ?thesis
   by (simp add: * not-integrable-integral-eq)
next
  assume (∫⁺ x. ennreal (f x) ∂M) ≠ ∞
  then have integrable M f
   by (cases ∫⁺ x. ennreal (f x) ∂M rule: ennreal-cases)
      (auto intro!: integrableI-nn-integral-finite assms)
  from nn-integral-eq-integral[OF this] nonneg show ?thesis
   by (simp add: integral-nonneg-AE)
qed

lemma enn2real-nn-integral-eq-integral:
  assumes eq: AE x in M. f x = ennreal (g x) and nn: AE x in M. 0 ≤ g x
    and fin: (∫⁺x. f x ∂M) < top
    and [measurable]: g ∈ M →_M borel
  shows enn2real (∫⁺x. f x ∂M) = (∫ x. g x ∂M)
proof −
```

have *ennreal* (*enn2real* ($\int^+ x.$ *f x* ∂M)) = ($\int^+ x.$ *f x* ∂M)
using *fin* **by** (*intro ennreal-enn2real*) *auto*
also have ... = ($\int^+ x.$ *g x* ∂M)
using *eq* **by** (*rule nn-integral-cong-AE*)
also have ... = ($\int x.$ *g x* ∂M)
proof (*rule nn-integral-eq-integral*)
show *integrable M g*
proof (*rule integrableI-bounded*)
have ($\int^+ x.$ *ennreal* (*norm* (*g x*)) ∂M) = ($\int^+ x.$ *f x* ∂M)
using *eq nn* **by** (*auto intro*!: *nn-integral-cong-AE elim*!: *eventually-elim2*)
also note *fin*
finally show ($\int^+ x.$ *ennreal* (*norm* (*g x*)) ∂M) < ∞
by *simp*
qed *simp*
qed *fact*
finally show *?thesis*
using *nn* **by** (*simp add*: *integral-nonneg-AE*)
qed

lemma *has-bochner-integral-nn-integral*:
assumes *f* ∈ *borel-measurable M AE x in M*. 0 ≤ *f x* 0 ≤ *x*
assumes ($\int^+ x.$ *f x* ∂M) = *ennreal x*
shows *has-bochner-integral M f x*
unfolding *has-bochner-integral-iff*
using *assms* **by** (*auto simp*: *assms integral-eq-nn-integral intro*: *integrableI-nn-integral-finite*)

lemma *integrableI-simple-bochner-integrable*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *simple-bochner-integrable M f* ⟹ *integrable M f*
by (*intro integrableI-sequence*[**where** *s*=λ-. *f*] *borel-measurable-simple-function*)
(*auto simp*: *zero-ennreal-def*[*symmetric*] *simple-bochner-integrable.simps*)

lemma *integrable-induct*[*consumes 1*, *case-names base add lim*, *induct pred*: *integrable*]:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
assumes *integrable M f*
assumes *base*: ⋀*A c*. *A* ∈ *sets M* ⟹ *emeasure M A* < ∞ ⟹ *P* (λ*x*. *indicator A x* ∗$_R$ *c*)
assumes *add*: ⋀*f g*. *integrable M f* ⟹ *P f* ⟹ *integrable M g* ⟹ *P g* ⟹ *P* (λ*x*. *f x* + *g x*)
assumes *lim*: ⋀*f s*. (⋀*i*. *integrable M* (*s i*)) ⟹ (⋀*i*. *P* (*s i*)) ⟹
(⋀*x*. *x* ∈ *space M* ⟹ (λ*i*. *s i x*) ⟶ *f x*) ⟹
(⋀*i x*. *x* ∈ *space M* ⟹ *norm* (*s i x*) ≤ 2 ∗ *norm* (*f x*)) ⟹ *integrable M f* ⟹ *P f*
shows *P f*
proof −
from ‹*integrable M f*› **have** *f*: *f* ∈ *borel-measurable M* ($\int^+ x.$ *norm* (*f x*) ∂M) < ∞
unfolding *integrable-iff-bounded* **by** *auto*

from *borel-measurable-implies-sequence-metric*[*OF f*(*1*)]
obtain *s* **where** *s*: $\bigwedge i.$ *simple-function M* (*s i*) $\bigwedge x.$ $x \in$ *space M* $\Longrightarrow$ ($\lambda i.$ *s i x*) $\longrightarrow$ *f x*
$\bigwedge i\ x.$ $x \in$ *space M* $\Longrightarrow$ *norm* (*s i x*) $\leq$ *2* $*$ *norm* (*f x*)
unfolding *norm-conv-dist* **by** *metis*

{ fix *f A*
have [*simp*]: *P* ($\lambda x.$ *0*)
using *base*[*of* {} *undefined*] **by** *simp*
have ($\bigwedge i$::'*b*. $i \in A \Longrightarrow$ *integrable M* (*f i*::'*a* $\Rightarrow$ '*b*)) $\Longrightarrow$
($\bigwedge i.$ $i \in A \Longrightarrow$ *P* (*f i*)) $\Longrightarrow$ *P* ($\lambda x.$ $\sum i \in A.$ *f i x*)
by (*induct A rule*: *infinite-finite-induct*) (*auto intro*!: *add*) **}**
note *setsum* = *this*

def *s'* $\equiv$ $\lambda i\ z.$ *indicator* (*space M*) *z* $*_R$ *s i z*
then have *s'-eq-s*: $\bigwedge i\ x.$ $x \in$ *space M* $\Longrightarrow$ *s' i x* = *s i x*
by *simp*

have *sf*[*measurable*]: $\bigwedge i.$ *simple-function M* (*s' i*)
unfolding *s'-def* **using** *s*(*1*)
by (*intro simple-function-compose2*[**where** *h*=*op* $*_R$] *simple-function-indicator*) *auto*

{ fix *i*
have $\bigwedge z.$ {*y*. *s' i z* = *y* $\wedge$ *y* $\in$ *s' i* ‘ *space M* $\wedge$ *y* $\neq$ *0* $\wedge$ *z* $\in$ *space M*} =
(*if z* $\in$ *space M* $\wedge$ *s' i z* $\neq$ *0 then* {*s' i z*} *else* {})
by (*auto simp add*: *s'-def split*: *split-indicator*)
then have $\bigwedge z.$ *s' i* = ($\lambda z.$ $\sum y \in s'$ *i*‘*space M* $-$ {*0*}. *indicator* {*x*$\in$*space M*. *s' i x* = *y*} *z* $*_R$ *y*)
using *sf* **by** (*auto simp*: *fun-eq-iff simple-function-def s'-def*) **}**
note *s'-eq* = *this*

show *P f*
proof (*rule lim*)
fix *i*

have ($\int^+ x.$ *norm* (*s' i x*) ∂M) $\leq$ ($\int^+ x.$ *ennreal* (*2* $*$ *norm* (*f x*)) ∂M)
using *s* **by** (*intro nn-integral-mono*) (*auto simp*: *s'-eq-s*)
also have ... $< \infty$
using *f* **by** (*simp add*: *nn-integral-cmult ennreal-mult-less-top ennreal-mult*)
finally have *sbi*: *simple-bochner-integrable M* (*s' i*)
using *sf* **by** (*intro simple-bochner-integrableI-bounded*) *auto*
then show *integrable M* (*s' i*)
by (*rule integrableI-simple-bochner-integrable*)

{ fix *x* **assume***x* $\in$ *space M s' i x* $\neq$ *0*
then have *emeasure M* {*y* $\in$ *space M*. *s' i y* = *s' i x*} $\leq$ *emeasure M* {*y* $\in$ *space M*. *s' i y* $\neq$ *0*}
by (*intro emeasure-mono*) *auto*

```
    also have ... < ∞
      using sbi by (auto elim: simple-bochner-integrable.cases simp: less-top)
    finally have emeasure M {y ∈ space M. s' i y = s' i x} ≠ ∞ by simp }
  then show P (s' i)
    by (subst s'-eq) (auto intro!: setsum base simp: less-top)

  fix x assume x ∈ space M with s show (λi. s' i x) ⟶ f x
    by (simp add: s'-eq-s)
  show norm (s' i x) ≤ 2 * norm (f x)
    using ‹x ∈ space M› s by (simp add: s'-eq-s)
 qed fact
qed

lemma integral-eq-zero-AE:
 (AE x in M. f x = 0) ⟹ integral^L M f = 0
 using integral-cong-AE[of f M λ-. 0]
 by (cases integrable M f) (simp-all add: not-integrable-integral-eq)

lemma integral-nonneg-eq-0-iff-AE:
 fixes f :: - ⇒ real
 assumes f[measurable]: integrable M f and nonneg: AE x in M. 0 ≤ f x
 shows integral^L M f = 0 ⟷ (AE x in M. f x = 0)
proof
 assume integral^L M f = 0
 then have integral^N M f = 0
   using nn-integral-eq-integral[OF f nonneg] by simp
 then have AE x in M. ennreal (f x) ≤ 0
   by (simp add: nn-integral-0-iff-AE)
 with nonneg show AE x in M. f x = 0
   by auto
qed (auto simp add: integral-eq-zero-AE)

lemma integral-mono-AE:
 fixes f :: 'a ⇒ real
 assumes integrable M f integrable M g AE x in M. f x ≤ g x
 shows integral^L M f ≤ integral^L M g
proof −
 have 0 ≤ integral^L M (λx. g x − f x)
   using assms by (intro integral-nonneg-AE integrable-diff assms) auto
 also have ... = integral^L M g − integral^L M f
   by (intro integral-diff assms)
 finally show ?thesis by simp
qed

lemma integral-mono:
 fixes f :: 'a ⇒ real
 shows integrable M f ⟹ integrable M g ⟹ (⋀x. x ∈ space M ⟹ f x ≤ g x)
⟹
   integral^L M f ≤ integral^L M g
```

by (*intro integral-mono-AE*) *auto*

lemma (**in** *finite-measure*) *integrable-measure*:
assumes *I*: *disjoint-family-on X I countable I*
shows *integrable* (*count-space I*) (λ*i*. *measure M* (*X i*))
proof −
have (∫ +*i*. *measure M* (*X i*) ∂*count-space I*) = (∫ +*i*. *measure M* (*if X i* ∈ *sets M then X i else* {}) ∂*count-space I*)
by (*auto intro*!: *nn-integral-cong measure-notin-sets*)
also have ... = *measure M* (⋃*i*∈*I*. *if X i* ∈ *sets M then X i else* {})
using *I* **unfolding** *emeasure-eq-measure*[*symmetric*]
by (*subst emeasure-UN-countable*) (*auto simp*: *disjoint-family-on-def*)
finally show *?thesis*
by (*auto intro*!: *integrableI-bounded*)
qed

lemma *integrableI-real-bounded*:
assumes *f*: *f* ∈ *borel-measurable M* **and** *ae*: *AE x in M*. *0* ≤ *f x* **and** *fin*: *integral*^{N} *M f* < ∞
shows *integrable M f*
proof (*rule integrableI-bounded*)
have (∫ + *x*. *ennreal* (*norm* (*f x*)) ∂*M*) = ∫ + *x*. *ennreal* (*f x*) ∂*M*
using *ae* **by** (*auto intro*: *nn-integral-cong-AE*)
also note *fin*
finally show (∫ + *x*. *ennreal* (*norm* (*f x*)) ∂*M*) < ∞ **.**
qed *fact*

lemma *integral-real-bounded*:
assumes *0* ≤ *r integral*^{N} *M f* ≤ *ennreal r*
shows *integral*^{L} *M f* ≤ *r*
proof *cases*
assume [*simp*]: *integrable M f*

have *integral*^{L} *M* (λ*x*. *max 0* (*f x*)) = *integral*^{N} *M* (λ*x*. *max 0* (*f x*))
by (*intro nn-integral-eq-integral*[*symmetric*]) *auto*
also have ... = *integral*^{N} *M f*
by (*intro nn-integral-cong*) (*simp add*: *max-def ennreal-neg*)
also have ... ≤ *r*
by *fact*
finally have *integral*^{L} *M* (λ*x*. *max 0* (*f x*)) ≤ *r*
using ‹*0* ≤ *r*› **by** *simp*

moreover have *integral*^{L} *M f* ≤ *integral*^{L} *M* (λ*x*. *max 0* (*f x*))
by (*rule integral-mono-AE*) *auto*
ultimately show *?thesis*
by *simp*
next
assume ¬ *integrable M f* **then show** *?thesis*
using ‹*0* ≤ *r*› **by** (*simp add*: *not-integrable-integral-eq*)

qed

8.1 Restricted measure spaces

lemma *integrable-restrict-space*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes Ω[*simp*]: $\Omega \cap$ *space* $M \in$ *sets* M
shows *integrable* (*restrict-space* M Ω) $f \longleftrightarrow$ *integrable* M (λx. *indicator* Ω x $*_R$ f x)
unfolding *integrable-iff-bounded*
borel-measurable-restrict-space-iff[*OF* Ω]
nn-integral-restrict-space[*OF* Ω]
by (*simp add*: *ac-simps ennreal-indicator ennreal-mult*)

lemma *integral-restrict-space*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes Ω[*simp*]: $\Omega \cap$ *space* $M \in$ *sets* M
shows $integral^L$ (*restrict-space* M Ω) f = $integral^L$ M (λx. *indicator* Ω x $*_R$ f x)
proof (*rule integral-eq-cases*)
assume *integrable* (*restrict-space* M Ω) f
then show *?thesis*
proof *induct*
case (*base* A c) **then show** *?case*
by (*simp add*: *indicator-inter-arith*[*symmetric*] *sets-restrict-space-iff*
emeasure-restrict-space Int-absorb1 measure-restrict-space)
next
case (*add* g f) **then show** *?case*
by (*simp add*: *scaleR-add-right integrable-restrict-space*)
next
case (*lim* f s)
show *?case*
proof (*rule LIMSEQ-unique*)
show (λi. $integral^L$ (*restrict-space* M Ω) (s i)) $\longrightarrow$ $integral^L$ (*restrict-space* M Ω) f
using *lim* **by** (*intro integral-dominated-convergence*[**where** w=λx. 2 $*$ *norm* (f x)]) *simp-all*

show (λi. $integral^L$ (*restrict-space* M Ω) (s i)) $\longrightarrow$ ($\int$ x. *indicator* Ω x $*_R$ f x ∂M)
unfolding *lim*
using *lim*
by (*intro integral-dominated-convergence*[**where** w=λx. 2 $*$ *norm* (*indicator* Ω x $*_R$ f x)])
(*auto simp add*: *space-restrict-space integrable-restrict-space simp del*: *norm-scaleR*
split: *split-indicator*)
qed
qed

qed (*simp add*: *integrable-restrict-space*)

lemma *integral-empty*:
assumes *space M* = {}
shows *integral*^{L} *M f* = *0*
proof −
have (∫ *x*. *f x* ∂*M*) = (∫ *x*. *0* ∂*M*)
by(*rule integral-cong*)(*simp-all add*: *assms*)
thus *?thesis* **by** *simp*
qed

8.2 Measure spaces with an associated density

lemma *integrable-density*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*} **and** *g* :: *'a* ⇒ *real*
assumes [*measurable*]: *f* ∈ *borel-measurable M g* ∈ *borel-measurable M*
and *nn*: *AE x in M*. *0* ≤ *g x*
shows *integrable* (*density M g*) *f* ⟷ *integrable M* (λ*x*. *g x* ∗$_R$ *f x*)
unfolding *integrable-iff-bounded* **using** *nn*
apply (*simp add*: *nn-integral-density less-top*[*symmetric*])
apply (*intro arg-cong2*[**where** *f*=*op* =] *refl nn-integral-cong-AE*)
apply (*auto simp*: *ennreal-mult*)
done

lemma *integral-density*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*} **and** *g* :: *'a* ⇒ *real*
assumes *f*: *f* ∈ *borel-measurable M*
and *g*[*measurable*]: *g* ∈ *borel-measurable M AE x in M*. *0* ≤ *g x*
shows *integral*^{L} (*density M g*) *f* = *integral*^{L} *M* (λ*x*. *g x* ∗$_R$ *f x*)
proof (*rule integral-eq-cases*)
assume *integrable* (*density M g*) *f*
then show *?thesis*
proof *induct*
case (*base A c*)
then have [*measurable*]: *A* ∈ *sets M* **by** *auto*

have *int*: *integrable M* (λ*x*. *g x* ∗ *indicator A x*)
using *g base integrable-density*[*of indicator A* :: *'a* ⇒ *real M g*] **by** *simp*
then have *integral*^{L} *M* (λ*x*. *g x* ∗ *indicator A x*) = (∫$^+$ *x*. *ennreal* (*g x* ∗ *indicator A x*) ∂*M*)
using *g* **by** (*subst nn-integral-eq-integral*) *auto*
also have ... = (∫$^+$ *x*. *ennreal* (*g x*) ∗ *indicator A x* ∂*M*)
by (*intro nn-integral-cong*) (*auto split*: *split-indicator*)
also have ... = *emeasure* (*density M g*) *A*
by (*rule emeasure-density*[*symmetric*]) *auto*
also have ... = *ennreal* (*measure* (*density M g*) *A*)
using *base* **by** (*auto intro*: *emeasure-eq-ennreal-measure*)
also have ... = *integral*^{L} (*density M g*) (*indicator A*)
using *base* **by** *simp*

```
    finally show ?case
      using base g
      apply (simp add: int integral-nonneg-AE)
      apply (subst (asm) ennreal-inj)
      apply (auto intro!: integral-nonneg-AE)
      done
  next
    case (add f h)
    then have [measurable]: f ∈ borel-measurable M h ∈ borel-measurable M
      by (auto dest!: borel-measurable-integrable)
    from add g show ?case
      by (simp add: scaleR-add-right integrable-density)
  next
    case (lim f s)
    have [measurable]: f ∈ borel-measurable M ⋀i. s i ∈ borel-measurable M
      using lim(1,5)[THEN borel-measurable-integrable] by auto

    show ?case
    proof (rule LIMSEQ-unique)
      show (λi. integral^L M (λx. g x *_R s i x)) ⟶ integral^L M (λx. g x *_R f
x)
      proof (rule integral-dominated-convergence)
        show integrable M (λx. 2 * norm (g x *_R f x))
          by (intro integrable-mult-right integrable-norm integrable-density[THEN
iffD1] lim g) auto
        show AE x in M. (λi. g x *_R s i x) ⟶ g x *_R f x
          using lim(3) by (auto intro!: tendsto-scaleR AE-I2[of M])
        show ⋀i. AE x in M. norm (g x *_R s i x) ≤ 2 * norm (g x *_R f x)
            using lim(4) g by (auto intro!: AE-I2[of M] mult-left-mono simp:
field-simps)
      qed auto
      show (λi. integral^L M (λx. g x *_R s i x)) ⟶ integral^L (density M g) f
        unfolding lim(2)[symmetric]
        by (rule integral-dominated-convergence[where w=λx. 2 * norm (f x)])
          (insert lim(3−5), auto)
    qed
  qed
qed (simp add: f g integrable-density)

lemma
  fixes g :: 'a ⇒ real
  assumes f ∈ borel-measurable M AE x in M. 0 ≤ f x g ∈ borel-measurable M
  shows integral-real-density: integral^L (density M f) g = (∫ x. f x * g x ∂M)
    and integrable-real-density: integrable (density M f) g ⟷ integrable M (λx. f
x * g x)
  using assms integral-density[of g M f] integrable-density[of g M f] by auto

lemma has-bochner-integral-density:
  fixes f :: 'a ⇒ 'b::{banach, second-countable-topology} and g :: 'a ⇒ real
```

shows $f \in$ *borel-measurable* $M \Longrightarrow g \in$ *borel-measurable* $M \Longrightarrow$ (*AE* x *in* M. $0 \leq g\ x$) $\Longrightarrow$
has-bochner-integral M $(\lambda x.\ g\ x *_R f\ x)\ x \Longrightarrow$ *has-bochner-integral* (*density* M g) $f\ x$
by (*simp add*: *has-bochner-integral-iff integrable-density integral-density*)

8.3 Distributions

lemma *integrable-distr-eq*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes [*measurable*]: $g \in$ *measurable* $M\ N\ f \in$ *borel-measurable* N
shows *integrable* (*distr* $M\ N\ g$) $f \longleftrightarrow$ *integrable* M $(\lambda x.\ f\ (g\ x))$
unfolding *integrable-iff-bounded* **by** (*simp-all add*: *nn-integral-distr*)

lemma *integrable-distr*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
shows $T \in$ *measurable* $M\ M' \Longrightarrow$ *integrable* (*distr* $M\ M'\ T$) $f \Longrightarrow$ *integrable* M $(\lambda x.\ f\ (T\ x))$
by (*subst integrable-distr-eq*[*symmetric*, **where** $g{=}T$])
(*auto dest*: *borel-measurable-integrable*)

lemma *integral-distr*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes g[*measurable*]: $g \in$ *measurable* $M\ N$ **and** f: $f \in$ *borel-measurable* N
shows $integral^L$ (*distr* $M\ N\ g$) $f = integral^L\ M\ (\lambda x.\ f\ (g\ x))$
proof (*rule integral-eq-cases*)
assume *integrable* (*distr* $M\ N\ g$) f
then show *?thesis*
proof *induct*
case (*base* $A\ c$)
then have [*measurable*]: $A \in$ *sets* N **by** *auto*
from *base* **have** *int*: *integrable* (*distr* $M\ N\ g$) $(\lambda a.$ *indicator* $A\ a *_R c)$
by (*intro integrable-indicator*)

have $integral^L$ (*distr* $M\ N\ g$) $(\lambda a.$ *indicator* $A\ a *_R c) =$ *measure* (*distr* $M\ N\ g$) $A *_R c$
using *base* **by** *auto*
also have $\ldots =$ *measure* M $(g -` A \cap$ *space* $M) *_R c$
by (*subst measure-distr*) *auto*
also have $\ldots = integral^L\ M$ $(\lambda a.$ *indicator* $(g -` A \cap$ *space* $M)\ a *_R c)$
using *base* **by** (*auto simp*: *emeasure-distr*)
also have $\ldots = integral^L\ M$ $(\lambda a.$ *indicator* $A\ (g\ a) *_R c)$
using *int base* **by** (*intro integral-cong-AE*) (*auto simp*: *emeasure-distr split*: *split-indicator*)
finally show *?case* **.**
next
case (*add* $f\ h$)
then have [*measurable*]: $f \in$ *borel-measurable* $N\ h \in$ *borel-measurable* N
by (*auto dest*!: *borel-measurable-integrable*)

```
    from add g show ?case
      by (simp add: scaleR-add-right integrable-distr-eq)
  next
    case (lim f s)
    have [measurable]: f ∈ borel-measurable N ⋀i. s i ∈ borel-measurable N
      using lim(1,5)[THEN borel-measurable-integrable] by auto

    show ?case
    proof (rule LIMSEQ-unique)
      show (λi. integral^L M (λx. s i (g x))) ⟶ integral^L M (λx. f (g x))
      proof (rule integral-dominated-convergence)
        show integrable M (λx. 2 * norm (f (g x)))
          using lim by (auto simp: integrable-distr-eq)
        show AE x in M. (λi. s i (g x)) ⟶ f (g x)
          using lim(3) g[THEN measurable-space] by auto
        show ⋀i. AE x in M. norm (s i (g x)) ≤ 2 * norm (f (g x))
          using lim(4) g[THEN measurable-space] by auto
      qed auto
      show (λi. integral^L M (λx. s i (g x))) ⟶ integral^L (distr M N g) f
        unfolding lim(2)[symmetric]
        by (rule integral-dominated-convergence[where w=λx. 2 * norm (f x)])
           (insert lim(3-5), auto)
    qed
  qed
qed (simp add: f g integrable-distr-eq)

lemma has-bochner-integral-distr:
  fixes f :: 'a ⇒ 'b::{banach, second-countable-topology}
  shows f ∈ borel-measurable N ⟹ g ∈ measurable M N ⟹
    has-bochner-integral M (λx. f (g x)) x ⟹ has-bochner-integral (distr M N g)
f x
  by (simp add: has-bochner-integral-iff integrable-distr-eq integral-distr)
```

8.4 Lebesgue integration on *count-space*

```
lemma integrable-count-space:
  fixes f :: 'a ⇒ 'b::{banach,second-countable-topology}
  shows finite X ⟹ integrable (count-space X) f
  by (auto simp: nn-integral-count-space integrable-iff-bounded)

lemma measure-count-space[simp]:
  B ⊆ A ⟹ finite B ⟹ measure (count-space A) B = card B
  unfolding measure-def by (subst emeasure-count-space ) auto

lemma lebesgue-integral-count-space-finite-support:
  assumes f: finite {a∈A. f a ≠ 0}
  shows (∫ x. f x ∂count-space A) = (∑ a | a ∈ A ∧ f a ≠ 0. f a)
proof −
  have eq: ⋀x. x ∈ A ⟹ (∑ a | x = a ∧ a ∈ A ∧ f a ≠ 0. f a) = (∑ x∈{x}. f
```

x)
by (*intro setsum.mono-neutral-cong-left*) *auto*

have ($\int x.\ f\ x\ \partial$*count-space* A) = ($\int x.\ (\sum a \mid a \in A \wedge f\ a \neq 0.$ *indicator* $\{a\}$ $x *_R f\ a$) ∂*count-space* A)
by (*intro integral-cong refl*) (*simp add*: *f eq*)
also have ... = ($\sum a \mid a \in A \wedge f\ a \neq 0.$ *measure* (*count-space* A) $\{a\} *_R f\ a$)
by (*subst integral-setsum*) (*auto intro*!: *setsum.cong*)
finally show *?thesis*
by *auto*
qed

lemma *lebesgue-integral-count-space-finite*: *finite* $A \Longrightarrow$ ($\int x.\ f\ x\ \partial$*count-space* A) = ($\sum a{\in}A.\ f\ a$)
by (*subst lebesgue-integral-count-space-finite-support*)
(*auto intro*!: *setsum.mono-neutral-cong-left*)

lemma *integrable-count-space-nat-iff*:
fixes f :: *nat* $\Rightarrow$ -::{*banach*,*second-countable-topology*}
shows *integrable* (*count-space UNIV*) $f \longleftrightarrow$ *summable* ($\lambda x.$ *norm* ($f\ x$))
by (*auto simp add*: *integrable-iff-bounded nn-integral-count-space-nat ennreal-suminf-neq-top*
intro: *summable-suminf-not-top*)

lemma *sums-integral-count-space-nat*:
fixes f :: *nat* $\Rightarrow$ -::{*banach*,*second-countable-topology*}
assumes $*$: *integrable* (*count-space UNIV*) f
shows f *sums* (*integral*^{L} (*count-space UNIV*) f)
proof −
let *?f* = $\lambda n\ i.$ *indicator* $\{n\}$ $i *_R f\ i$
have f': $\bigwedge n\ i.$ *?f n i* = *indicator* $\{n\}$ $i *_R f\ n$
by (*auto simp*: *fun-eq-iff split*: *split-indicator*)

have ($\lambda i.\ \int n.$ *?f i n* ∂*count-space UNIV*) *sums* $\int n.$ ($\sum i.$ *?f i n*) ∂*count-space UNIV*
proof (*rule sums-integral*)
show $\bigwedge i.$ *integrable* (*count-space UNIV*) (*?f i*)
using $*$ **by** (*intro integrable-mult-indicator*) *auto*
show *AE n in count-space UNIV*. *summable* ($\lambda i.$ *norm* (*?f i n*))
using *summable-finite*[*of* $\{n\}$ $\lambda i.$ *norm* (*?f i n*) **for** n] **by** *simp*
show *summable* ($\lambda i.\ \int n.$ *norm* (*?f i n*) ∂*count-space UNIV*)
using $*$ **by** (*subst f'*) (*simp add*: *integrable-count-space-nat-iff*)
qed
also have ($\int n.$ ($\sum i.$ *?f i n*) ∂*count-space UNIV*) = ($\int n.$ *f n* ∂*count-space UNIV*)
using *suminf-finite*[*of* $\{n\}$ $\lambda i.$ *?f i n* **for** n] **by** (*auto intro*!: *integral-cong*)
also have ($\lambda i.\ \int n.$ *?f i n* ∂*count-space UNIV*) = f
by (*subst f'*) *simp*
finally show *?thesis* .
qed

lemma *integral-count-space-nat*:
fixes $f :: nat \Rightarrow$ -::{*banach*,*second-countable-topology*}
shows *integrable* (*count-space UNIV*) $f \Longrightarrow$ $integral^L$ (*count-space UNIV*) $f = (\sum x.\ f\ x)$
using *sums-integral-count-space-nat* **by** (*rule sums-unique*)

8.5 Point measure

lemma *lebesgue-integral-point-measure-finite*:
fixes $g :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
shows *finite* $A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow 0 \leq f\ a) \Longrightarrow$
$integral^L$ (*point-measure A f*) $g = (\sum a{\in}A.\ f\ a *_R g\ a)$
by (*simp add*: *lebesgue-integral-count-space-finite AE-count-space integral-density point-measure-def*)

lemma *integrable-point-measure-finite*:
fixes $g :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*} **and** $f :: 'a \Rightarrow real$
shows *finite* $A \Longrightarrow$ *integrable* (*point-measure A f*) g
unfolding *point-measure-def*
apply (*subst density-cong*[**where** $f'{=}\lambda x.$ *ennreal* (*max 0* (*f x*))])
apply (*auto split*: *split-max simp*: *ennreal-neg*)
apply (*subst integrable-density*)
apply (*auto simp*: *AE-count-space integrable-count-space*)
done

8.6 Lebesgue integration on *null-measure*

lemma *has-bochner-integral-null-measure-iff*[*iff*]:
has-bochner-integral (*null-measure M*) *f 0* $\longleftrightarrow$ $f \in$ *borel-measurable M*
by (*auto simp add*: *has-bochner-integral.simps simple-bochner-integral-def*[*abs-def*]
intro!: *exI*[*of* - $\lambda n\ x.\ 0$] *simple-bochner-integrable.intros*)

lemma *integrable-null-measure-iff*[*iff*]: *integrable* (*null-measure M*) *f* $\longleftrightarrow$ $f \in$ *borel-measurable M*
by (*auto simp add*: *integrable.simps*)

lemma *integral-null-measure*[*simp*]: $integral^L$ (*null-measure M*) *f* = *0*
by (*cases integrable* (*null-measure M*) *f*)
(*auto simp add*: *not-integrable-integral-eq has-bochner-integral-integral-eq*)

8.7 Legacy lemmas for the real-valued Lebesgue integral

lemma *real-lebesgue-integral-def*:
assumes *f*[*measurable*]: *integrable M f*
shows $integral^L$ *M f* = *enn2real* ($\int^+ x.\ f\ x\ \partial M$) − *enn2real* ($\int^+ x.$ *ennreal* (− *f x*) ∂M)
proof −
have $integral^L$ *M f* = $integral^L$ *M* ($\lambda x.$ *max 0* (*f x*) − *max 0* (− *f x*))
by (*auto intro*!: *arg-cong*[**where** *f*=$integral^L$ *M*])

also have ... = $integral^L\ M\ (\lambda x.\ max\ 0\ (f\ x)) - integral^L\ M\ (\lambda x.\ max\ 0\ (-\ f\ x))$
by (*intro integral-diff integrable-max integrable-minus integrable-zero f*)
also have $integral^L\ M\ (\lambda x.\ max\ 0\ (f\ x)) = enn2real\ (\int^+ x.\ ennreal\ (f\ x)\ \partial M)$
by (*subst integral-eq-nn-integral*) (*auto intro*!: *arg-cong*[**where** *f*=*enn2real*] *nn-integral-cong simp*: *max-def ennreal-neg*)
also have $integral^L\ M\ (\lambda x.\ max\ 0\ (-\ f\ x)) = enn2real\ (\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M)$
by (*subst integral-eq-nn-integral*) (*auto intro*!: *arg-cong*[**where** *f*=*enn2real*] *nn-integral-cong simp*: *max-def ennreal-neg*)
finally show *?thesis* **.**
qed

lemma *real-integrable-def*:
integrable M f $\longleftrightarrow$ *f* $\in$ *borel-measurable M* $\wedge$
$(\int^+ x.\ ennreal\ (f\ x)\ \partial M) \neq \infty \wedge (\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \neq \infty$
unfolding *integrable-iff-bounded*
proof (*safe del*: *notI*)
assume $*$: $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M) < \infty$
have $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) \leq (\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M)$
by (*intro nn-integral-mono*) *auto*
also note $*$
finally show $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) \neq \infty$
by *simp*
have $(\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \leq (\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M)$
by (*intro nn-integral-mono*) *auto*
also note $*$
finally show $(\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \neq \infty$
by *simp*
next
assume [*measurable*]: *f* $\in$ *borel-measurable M*
assume *fin*: $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) \neq \infty\ (\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \neq \infty$
have $(\int^+ x.\ norm\ (f\ x)\ \partial M) = (\int^+ x.\ ennreal\ (f\ x) + ennreal\ (-\ f\ x)\ \partial M)$
by (*intro nn-integral-cong*) (*auto simp*: *abs-real-def ennreal-neg*)
also have... = $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) + (\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M)$
by (*intro nn-integral-add*) *auto*
also have ... $< \infty$
using *fin* **by** (*auto simp*: *less-top*)
finally show $(\int^+ x.\ norm\ (f\ x)\ \partial M) < \infty$ **.**
qed

lemma *integrableD*[*dest*]:
assumes *integrable M f*
shows *f* $\in$ *borel-measurable M* $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) \neq \infty\ (\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \neq \infty$
using *assms* **unfolding** *real-integrable-def* **by** *auto*

lemma *integrableE*:
assumes *integrable M f*

obtains r q **where**
$(\int^+ x.\ ennreal\ (f\ x) \partial M) = ennreal\ r$
$(\int^+ x.\ ennreal\ (-f\ x) \partial M) = ennreal\ q$
$f \in borel\text{-}measurable\ M$ $integral^L\ M\ f = r - q$
using *assms* **unfolding** *real-integrable-def real-lebesgue-integral-def*[*OF assms*]
by (*cases rule*: *ennreal2-cases*[*of* $(\int^+ x.\ ennreal\ (-f\ x) \partial M)$ $(\int^+ x.\ ennreal\ (f\ x) \partial M)$]) *auto*

lemma *integral-monotone-convergence-nonneg*:
fixes $f :: nat \Rightarrow 'a \Rightarrow real$
assumes i: $\bigwedge i.\ integrable\ M\ (f\ i)$ **and** *mono*: $AE\ x\ in\ M.\ mono\ (\lambda n.\ f\ n\ x)$
and *pos*: $\bigwedge i.\ AE\ x\ in\ M.\ 0 \le f\ i\ x$
and *lim*: $AE\ x\ in\ M.\ (\lambda i.\ f\ i\ x) \longrightarrow u\ x$
and *ilim*: $(\lambda i.\ integral^L\ M\ (f\ i)) \longrightarrow x$
and u: $u \in borel\text{-}measurable\ M$
shows $integrable\ M\ u$
and $integral^L\ M\ u = x$
proof −
have *nn*: $AE\ x\ in\ M.\ \forall i.\ 0 \le f\ i\ x$
using *pos* **unfolding** *AE-all-countable* **by** *auto*
with *lim* **have** *u-nn*: $AE\ x\ in\ M.\ 0 \le u\ x$
by *eventually-elim* (*auto intro*: *LIMSEQ-le-const*)
have [*simp*]: $0 \le x$
by (*intro LIMSEQ-le-const*[*OF ilim*] *allI exI impI integral-nonneg-AE pos*)
have $(\int^+ x.\ ennreal\ (u\ x)\ \partial M) = (SUP\ n.\ (\int^+ x.\ ennreal\ (f\ n\ x)\ \partial M))$
proof (*subst nn-integral-monotone-convergence-SUP-AE*[*symmetric*])
fix i
from *mono nn* **show** $AE\ x\ in\ M.\ ennreal\ (f\ i\ x) \le ennreal\ (f\ (Suc\ i)\ x)$
by *eventually-elim* (*auto simp*: *mono-def*)
show $(\lambda x.\ ennreal\ (f\ i\ x)) \in borel\text{-}measurable\ M$
using i **by** *auto*
next
show $(\int^+ x.\ ennreal\ (u\ x)\ \partial M) = \int^+ x.\ (SUP\ i.\ ennreal\ (f\ i\ x))\ \partial M$
apply (*rule nn-integral-cong-AE*)
using *lim mono nn u-nn*
apply *eventually-elim*
apply (*simp add*: *LIMSEQ-unique*[*OF - LIMSEQ-SUP*] *incseq-def*)
done
qed
also have $\ldots = ennreal\ x$
using *mono i nn* **unfolding** *nn-integral-eq-integral*[*OF i pos*]
by (*subst LIMSEQ-unique*[*OF LIMSEQ-SUP*]) (*auto simp*: *mono-def integral-nonneg-AE pos intro*!: *integral-mono-AE ilim*)
finally have $(\int^+ x.\ ennreal\ (u\ x)\ \partial M) = ennreal\ x$ **.**
moreover have $(\int^+ x.\ ennreal\ (-\ u\ x)\ \partial M) = 0$
using u *u-nn* **by** (*subst nn-integral-0-iff-AE*) (*auto simp add*: *ennreal-neg*)
ultimately show $integrable\ M\ u$ $integral^L\ M\ u = x$
by (*auto simp*: *real-integrable-def real-lebesgue-integral-def u*)
qed

lemma
fixes $f :: nat \Rightarrow 'a \Rightarrow real$
assumes f: $\bigwedge i.\ integrable\ M\ (f\ i)$ **and** *mono*: $AE\ x\ in\ M.\ mono\ (\lambda n.\ f\ n\ x)$
and *lim*: $AE\ x\ in\ M.\ (\lambda i.\ f\ i\ x) \longrightarrow u\ x$
and *ilim*: $(\lambda i.\ integral^L\ M\ (f\ i)) \longrightarrow x$
and u: $u \in borel\text{-}measurable\ M$
shows *integrable-monotone-convergence*: $integrable\ M\ u$
and *integral-monotone-convergence*: $integral^L\ M\ u = x$
and *has-bochner-integral-monotone-convergence*: $has\text{-}bochner\text{-}integral\ M\ u\ x$
proof −
have *1*: $\bigwedge i.\ integrable\ M\ (\lambda x.\ f\ i\ x - f\ 0\ x)$
using f **by** *auto*
have *2*: $AE\ x\ in\ M.\ mono\ (\lambda n.\ f\ n\ x - f\ 0\ x)$
using *mono* **by** (*auto simp*: *mono-def le-fun-def*)
have *3*: $\bigwedge n.\ AE\ x\ in\ M.\ 0 \le f\ n\ x - f\ 0\ x$
using *mono* **by** (*auto simp*: *field-simps mono-def le-fun-def*)
have *4*: $AE\ x\ in\ M.\ (\lambda i.\ f\ i\ x - f\ 0\ x) \longrightarrow u\ x - f\ 0\ x$
using *lim* **by** (*auto intro*!: *tendsto-diff*)
have *5*: $(\lambda i.\ (\int x.\ f\ i\ x - f\ 0\ x\ \partial M)) \longrightarrow x - integral^L\ M\ (f\ 0)$
using *f ilim* **by** (*auto intro*!: *tendsto-diff*)
have *6*: $(\lambda x.\ u\ x - f\ 0\ x) \in borel\text{-}measurable\ M$
using *f*[*of 0*] u **by** *auto*
note *diff* = *integral-monotone-convergence-nonneg*[*OF 1 2 3 4 5 6*]
have $integrable\ M\ (\lambda x.\ (u\ x - f\ 0\ x) + f\ 0\ x)$
using *diff*(*1*) *f* **by** (*rule integrable-add*)
with *diff*(*2*) *f* **show** $integrable\ M\ u\ integral^L\ M\ u = x$
by *auto*
then show $has\text{-}bochner\text{-}integral\ M\ u\ x$
by (*metis has-bochner-integral-integrable*)
qed

lemma *integral-norm-eq-0-iff*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: $integrable\ M\ f$
shows $(\int x.\ norm\ (f\ x)\ \partial M) = 0 \longleftrightarrow emeasure\ M\ \{x \in space\ M.\ f\ x \neq 0\} = 0$
proof −
have $(\int^+ x.\ norm\ (f\ x)\ \partial M) = (\int x.\ norm\ (f\ x)\ \partial M)$
using *f* **by** (*intro nn-integral-eq-integral integrable-norm*) *auto*
then have $(\int x.\ norm\ (f\ x)\ \partial M) = 0 \longleftrightarrow (\int^+ x.\ norm\ (f\ x)\ \partial M) = 0$
by *simp*
also have ... $\longleftrightarrow emeasure\ M\ \{x \in space\ M.\ ennreal\ (norm\ (f\ x)) \neq 0\} = 0$
by (*intro nn-integral-0-iff*) *auto*
finally show *?thesis*
by *simp*
qed

lemma *integral-0-iff*:
fixes $f :: 'a \Rightarrow real$

shows *integrable M f* $\Longrightarrow$ $(\int x.\ |f\ x|\ \partial M) = 0 \longleftrightarrow$ *emeasure M* $\{x \in$ *space M*. *f x* $\neq 0\} = 0$
using *integral-norm-eq-0-iff*[*of M f*] **by** *simp*

lemma (**in** *finite-measure*) *integrable-const*[*intro*!, *simp*]: *integrable M* $(\lambda x.\ a)$
using *integrable-indicator*[*of space M M a*] **by** (*simp cong*: *integrable-cong add*: *less-top*[*symmetric*])

lemma *lebesgue-integral-const*[*simp*]:
fixes $a :: 'a ::$ {*banach*, *second-countable-topology*}
shows $(\int x.\ a\ \partial M) =$ *measure M* (*space M*) $*_R$ *a*
proof −
{ assume *emeasure M* (*space M*) $= \infty$ $a \neq 0$
then have *?thesis*
by (*auto simp add*: *not-integrable-integral-eq ennreal-mult-less-top measure-def integrable-iff-bounded*) **}**
moreover
{ assume $a = 0$ **then have** *?thesis* **by** *simp* **}**
moreover
{ assume *emeasure M* (*space M*) $\neq \infty$
interpret *finite-measure M*
proof qed *fact*
have $(\int x.\ a\ \partial M) = (\int x.$ *indicator* (*space M*) *x* $*_R$ *a* $\partial M)$
by (*intro integral-cong*) *auto*
also have $\ldots =$ *measure M* (*space M*) $*_R$ *a*
by (*simp add*: *less-top*[*symmetric*])
finally have *?thesis* **. }**
ultimately show *?thesis* **by** *blast*
qed

lemma (**in** *finite-measure*) *integrable-const-bound*:
fixes $f :: 'a \Rightarrow 'b ::$ {*banach*,*second-countable-topology*}
shows *AE x in M*. *norm* $(f\ x) \leq B \Longrightarrow f \in$ *borel-measurable M* $\Longrightarrow$ *integrable M f*
apply (*rule integrable-bound*[*OF integrable-const*[*of B*], *of f*])
apply *assumption*
apply (*cases* $0 \leq B$)
apply *auto*
done

lemma *integral-indicator-finite-real*:
fixes $f :: 'a \Rightarrow$ *real*
assumes [*simp*]: *finite A*
assumes [*measurable*]: $\bigwedge a.\ a \in A \Longrightarrow \{a\} \in$ *sets M*
assumes *finite*: $\bigwedge a.\ a \in A \Longrightarrow$ *emeasure M* $\{a\} < \infty$
shows $(\int x.\ f\ x * indicator\ A\ x\ \partial M) = (\sum a \in A.\ f\ a * measure\ M\ \{a\})$
proof −
have $(\int x.\ f\ x * indicator\ A\ x\ \partial M) = (\int x.\ (\sum a \in A.\ f\ a * indicator\ \{a\}\ x)\ \partial M)$
proof (*intro integral-cong refl*)

```
    fix x show f x * indicator A x = (∑ a∈A. f a * indicator {a} x)
      by (auto split: split-indicator simp: eq-commute[of x] cong: conj-cong)
  qed
  also have ... = (∑ a∈A. f a * measure M {a})
    using finite by (subst integral-setsum) (auto simp add: integrable-mult-left-iff)
  finally show ?thesis .
qed

lemma (in finite-measure) ennreal-integral-real:
  assumes [measurable]: f ∈ borel-measurable M
  assumes ae: AE x in M. f x ≤ ennreal B 0 ≤ B
  shows ennreal (∫ x. enn2real (f x) ∂M) = (∫⁺x. f x ∂M)
proof (subst nn-integral-eq-integral[symmetric])
  show integrable M (λx. enn2real (f x))
   using ae by (intro integrable-const-bound[where B=B]) (auto simp: enn2real-leI
enn2real-nonneg)
  show (∫⁺ x. ennreal (enn2real (f x)) ∂M) = integral^N M f
   using ae by (intro nn-integral-cong-AE) (auto simp: le-less-trans[OF - ennreal-less-top])
qed (auto simp: enn2real-nonneg)

lemma (in finite-measure) integral-less-AE:
  fixes X Y :: 'a ⇒ real
  assumes int: integrable M X integrable M Y
  assumes A: (emeasure M) A ≠ 0 A ∈ sets M AE x in M. x ∈ A ⟶ X x ≠ Y x
  assumes gt: AE x in M. X x ≤ Y x
  shows integral^L M X < integral^L M Y
proof −
  have integral^L M X ≤ integral^L M Y
    using gt int by (intro integral-mono-AE) auto
  moreover
  have integral^L M X ≠ integral^L M Y
  proof
    assume eq: integral^L M X = integral^L M Y
    have integral^L M (λx. |Y x − X x|) = integral^L M (λx. Y x − X x)
      using gt int by (intro integral-cong-AE) auto
    also have ... = 0
      using eq int by simp
    finally have (emeasure M) {x ∈ space M. Y x − X x ≠ 0} = 0
      using int by (simp add: integral-0-iff)
    moreover
    have (∫⁺x. indicator A x ∂M) ≤ (∫⁺x. indicator {x ∈ space M. Y x − X x
≠ 0} x ∂M)
      using A by (intro nn-integral-mono-AE) auto
    then have (emeasure M) A ≤ (emeasure M) {x ∈ space M. Y x − X x ≠ 0}
      using int A by (simp add: integrable-def)
    ultimately have emeasure M A = 0
      by simp
    with ‹(emeasure M) A ≠ 0› show False by auto
  qed
```

ultimately show *?thesis* **by** *auto*
qed

lemma (**in** *finite-measure*) *integral-less-AE-space*:
fixes X Y :: $'a \Rightarrow$ *real*
assumes *int*: *integrable M X integrable M Y*
assumes *gt*: *AE x in M. X x < Y x emeasure M* (*space M*) $\neq$ *0*
shows $integral^L$ *M X* < $integral^L$ *M Y*
using *gt* **by** (*intro integral-less-AE*[*OF int*, **where** *A*=*space M*]) *auto*

lemma *tendsto-integral-at-top*:
fixes *f* :: *real* $\Rightarrow$ $'a$::{*banach*, *second-countable-topology*}
assumes [*measurable-cong*]: *sets M* = *sets borel* **and** *f*[*measurable*]: *integrable M f*
shows (($\lambda y.$ $\int$ *x. indicator* {.. *y*} *x* $*_R$ *f x* ∂M) $\longrightarrow$ $\int$ *x. f x* ∂M) *at-top*
proof (*rule tendsto-at-topI-sequentially*)
fix *X* :: *nat* $\Rightarrow$ *real* **assume** *filterlim X at-top sequentially*
show ($\lambda n.$ $\int$ *x. indicator* {..*X n*} *x* $*_R$ *f x* ∂M) $\longrightarrow$ $integral^L$ *M f*
proof (*rule integral-dominated-convergence*)
show *integrable M* ($\lambda x.$ *norm* (*f x*))
by (*rule integrable-norm*) *fact*
show *AE x in M.* ($\lambda n.$ *indicator* {..*X n*} *x* $*_R$ *f x*) $\longrightarrow$ *f x*
proof
fix *x*
from ‹*filterlim X at-top sequentially*›
have *eventually* ($\lambda n.$ *x* $\leq$ *X n*) *sequentially*
unfolding *filterlim-at-top-ge*[**where** *c*=*x*] **by** *auto*
then show ($\lambda n.$ *indicator* {..*X n*} *x* $*_R$ *f x*) $\longrightarrow$ *f x*
by (*intro Lim-eventually*) (*auto split*: *split-indicator elim*!: *eventually-mono*)
qed
fix *n* **show** *AE x in M. norm* (*indicator* {..*X n*} *x* $*_R$ *f x*) $\leq$ *norm* (*f x*)
by (*auto split*: *split-indicator*)
qed *auto*
qed

lemma
fixes *f* :: *real* $\Rightarrow$ *real*
assumes *M*: *sets M* = *sets borel*
assumes *nonneg*: *AE x in M. 0* $\leq$ *f x*
assumes *borel*: *f* $\in$ *borel-measurable borel*
assumes *int*: $\bigwedge y.$ *integrable M* ($\lambda x.$ *f x* $*$ *indicator* {.. *y*} *x*)
assumes *conv*: (($\lambda y.$ $\int$ *x. f x* $*$ *indicator* {.. *y*} *x* ∂M) $\longrightarrow$ *x*) *at-top*
shows *has-bochner-integral-monotone-convergence-at-top*: *has-bochner-integral M f x*
and *integrable-monotone-convergence-at-top*: *integrable M f*
and *integral-monotone-convergence-at-top*:$integral^L$ *M f* = *x*
proof −
from *nonneg* **have** *AE x in M. mono* (λn::*nat. f x* $*$ *indicator* {..*real n*} *x*)
by (*auto split*: *split-indicator intro*!: *monoI*)

{ fix x **have** *eventually* $(\lambda n.\ f\ x * \mathit{indicator}\ \{..\mathit{real}\ n\}\ x = f\ x)$ *sequentially*
by (*rule eventually-sequentiallyI*[*of nat* $\lceil x \rceil$])
(*auto split*: *split-indicator simp*: *nat-le-iff ceiling-le-iff*) **}**
from *filterlim-cong*[*OF refl refl this*]
have *AE x in M*. $(\lambda i.\ f\ x * \mathit{indicator}\ \{..\mathit{real}\ i\}\ x) \longrightarrow f\ x$
by *simp*
have $(\lambda i.\ \int x.\ f\ x * \mathit{indicator}\ \{..\mathit{real}\ i\}\ x\ \partial M) \longrightarrow x$
using *conv filterlim-real-sequentially* **by** (*rule filterlim-compose*)
have *M-measure*[*simp*]: *borel-measurable M* = *borel-measurable borel*
using *M* **by** (*simp add*: *sets-eq-imp-space-eq measurable-def*)
have $f \in$ *borel-measurable M*
using *borel* **by** *simp*
show *has-bochner-integral M f x*
by (*rule has-bochner-integral-monotone-convergence*) *fact+*
then show *integrable M f* $\mathit{integral}^L\ M\ f = x$
by (*auto simp*: *-has-bochner-integral-iff*)
qed

8.8 Product measure

lemma (**in** *sigma-finite-measure*) *borel-measurable-lebesgue-integrable*[*measurable* (*raw*)]:
fixes f :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
assumes [*measurable*]: *case-prod f* ∈ *borel-measurable* $(N \bigotimes_M M)$
shows *Measurable.pred N* $(\lambda x.\ \mathit{integrable}\ M\ (f\ x))$
proof −
have [*simp*]: $\bigwedge x.\ x \in \mathit{space}\ N \Longrightarrow \mathit{integrable}\ M\ (f\ x) \longleftrightarrow (\int^+ y.\ \mathit{norm}\ (f\ x\ y)\ \partial M) < \infty$
unfolding *integrable-iff-bounded* **by** *simp*
show *?thesis*
by (*simp cong*: *measurable-cong*)
qed

lemma *Collect-subset* [*simp*]: $\{x{\in}A.\ P\ x\} \subseteq A$ **by** *auto*

lemma (**in** *sigma-finite-measure*) *measurable-measure*[*measurable* (*raw*)]:
$(\bigwedge x.\ x \in \mathit{space}\ N \Longrightarrow A\ x \subseteq \mathit{space}\ M) \Longrightarrow$
$\{x \in \mathit{space}\ (N \bigotimes_M M).\ \mathit{snd}\ x \in A\ (\mathit{fst}\ x)\} \in \mathit{sets}\ (N \bigotimes_M M) \Longrightarrow$
$(\lambda x.\ \mathit{measure}\ M\ (A\ x)) \in$ *borel-measurable N*
unfolding *measure-def* **by** (*intro measurable-emeasure borel-measurable-enn2real*) *auto*

lemma (**in** *sigma-finite-measure*) *borel-measurable-lebesgue-integral*[*measurable* (*raw*)]:
fixes f :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *case-prod f* ∈ *borel-measurable* $(N \bigotimes_M M)$
shows $(\lambda x.\ \int y.\ f\ x\ y\ \partial M) \in$ *borel-measurable N*
proof −
from *borel-measurable-implies-sequence-metric*[*OF f*, *of 0*] **guess** s **..**
then have s: $\bigwedge i.$ *simple-function* $(N \bigotimes_M M)\ (s\ i)$

$\bigwedge x\ y.\ x \in space\ N \Longrightarrow y \in space\ M \Longrightarrow (\lambda i.\ s\ i\ (x,\ y)) \longrightarrow f\ x\ y$
$\bigwedge i\ x\ y.\ x \in space\ N \Longrightarrow y \in space\ M \Longrightarrow norm\ (s\ i\ (x,\ y)) \leq 2 * norm\ (f\ x\ y)$
by (*auto simp*: *space-pair-measure*)

have [*measurable*]: $\bigwedge i.\ s\ i \in borel\text{-}measurable\ (N \bigotimes_M M)$
by (*rule borel-measurable-simple-function*) *fact*

have $\bigwedge i.\ s\ i \in measurable\ (N \bigotimes_M M)\ (count\text{-}space\ UNIV)$
by (*rule measurable-simple-function*) *fact*

def $f' \equiv \lambda i\ x.\ if\ integrable\ M\ (f\ x)\ then\ simple\text{-}bochner\text{-}integral\ M\ (\lambda y.\ s\ i\ (x,\ y))\ else\ 0$

{ fix *i x* **assume** $x \in space\ N$
then have $simple\text{-}bochner\text{-}integral\ M\ (\lambda y.\ s\ i\ (x,\ y)) = (\sum z \in s\ i\ `\ (space\ N \times space\ M).\ measure\ M\ \{y \in space\ M.\ s\ i\ (x,\ y) = z\} *_R z)$
using *s*(*1*)[*THEN simple-functionD*(*1*)]
unfolding *simple-bochner-integral-def*
by (*intro setsum.mono-neutral-cong-left*)
(*auto simp*: *eq-commute space-pair-measure image-iff cong*: *conj-cong*) **}**
note *eq* = *this*

show *?thesis*
proof (*rule borel-measurable-LIMSEQ-metric*)
fix *i* **show** $f'\ i \in borel\text{-}measurable\ N$
unfolding *f'-def* **by** (*simp-all add*: *eq cong*: *measurable-cong if-cong*)
next
fix *x* **assume** *x*: $x \in space\ N$
{ assume *int-f*: $integrable\ M\ (f\ x)$
have *int-2f*: $integrable\ M\ (\lambda y.\ 2 * norm\ (f\ x\ y))$
by (*intro integrable-norm integrable-mult-right int-f*)
have $(\lambda i.\ integral^L\ M\ (\lambda y.\ s\ i\ (x,\ y))) \longrightarrow integral^L\ M\ (f\ x)$
proof (*rule integral-dominated-convergence*)
from *int-f* **show** $f\ x \in borel\text{-}measurable\ M$ **by** *auto*
show $\bigwedge i.\ (\lambda y.\ s\ i\ (x,\ y)) \in borel\text{-}measurable\ M$
using *x* **by** *simp*
show $AE\ xa\ in\ M.\ (\lambda i.\ s\ i\ (x,\ xa)) \longrightarrow f\ x\ xa$
using *x s*(*2*) **by** *auto*
show $\bigwedge i.\ AE\ xa\ in\ M.\ norm\ (s\ i\ (x,\ xa)) \leq 2 * norm\ (f\ x\ xa)$
using *x s*(*3*) **by** *auto*
qed *fact*
moreover
{ fix *i*
have $simple\text{-}bochner\text{-}integrable\ M\ (\lambda y.\ s\ i\ (x,\ y))$
proof (*rule simple-bochner-integrableI-bounded*)
have $(\lambda y.\ s\ i\ (x,\ y))\ `\ space\ M \subseteq s\ i\ `\ (space\ N \times space\ M)$
using *x* **by** *auto*

```
        then show simple-function M (λy. s i (x, y))
          using simple-functionD(1)[OF s(1), of i] x
          by (intro simple-function-borel-measurable)
             (auto simp: space-pair-measure dest: finite-subset)
        have (∫⁺ y. ennreal (norm (s i (x, y))) ∂M) ≤ (∫⁺ y. 2 * norm (f x y)
∂M)
          using x s by (intro nn-integral-mono) auto
        also have (∫⁺ y. 2 * norm (f x y) ∂M) < ∞
          using int-2f by (simp add: integrable-iff-bounded)
        finally show (∫⁺ xa. ennreal (norm (s i (x, xa))) ∂M) < ∞ .
      qed
      then have integral^L M (λy. s i (x, y)) = simple-bochner-integral M (λy. s
i (x, y))
        by (rule simple-bochner-integrable-eq-integral[symmetric]) }
      ultimately have (λi. simple-bochner-integral M (λy. s i (x, y))) ⟶
integral^L M (f x)
       by simp }
   then
   show (λi. f' i x) ⟶ integral^L M (f x)
     unfolding f'-def
     by (cases integrable M (f x)) (simp-all add: not-integrable-integral-eq)
 qed
qed

lemma (in pair-sigma-finite) integrable-product-swap:
 fixes f :: - ⇒ -::{banach, second-countable-topology}
 assumes integrable (M1 ⨂M M2) f
 shows integrable (M2 ⨂M M1) (λ(x,y). f (y,x))
proof −
 interpret Q: pair-sigma-finite M2 M1 ..
  have ∗: (λ(x,y). f (y,x)) = (λx. f (case x of (x,y)⇒(y,x))) by (auto simp:
fun-eq-iff)
 show ?thesis unfolding ∗
   by (rule integrable-distr[OF measurable-pair-swap'])
      (simp add: distr-pair-swap[symmetric] assms)
qed

lemma (in pair-sigma-finite) integrable-product-swap-iff:
 fixes f :: - ⇒ -::{banach, second-countable-topology}
 shows integrable (M2 ⨂M M1) (λ(x,y). f (y,x)) ⟷ integrable (M1 ⨂M M2)
f
proof −
 interpret Q: pair-sigma-finite M2 M1 ..
 from Q.integrable-product-swap[of λ(x,y). f (y,x)] integrable-product-swap[of f]
 show ?thesis by auto
qed

lemma (in pair-sigma-finite) integral-product-swap:
 fixes f :: - ⇒ -::{banach, second-countable-topology}
```

assumes *f*: $f \in$ *borel-measurable* (*M1* $\bigotimes_M$ *M2*)
shows $(\int (x,y).\ f\ (y,x)\ \partial(M2 \bigotimes_M M1)) =$ *integral*^{L} (*M1* $\bigotimes_M$ *M2*) *f*
proof −
have ∗: $(\lambda(x,y).\ f\ (y,x)) = (\lambda x.\ f\ (case\ x\ of\ (x,y) \Rightarrow (y,x)))$ **by** (*auto simp*: *fun-eq-iff*)
show *?thesis* **unfolding** ∗
by (*simp add*: *integral-distr*[*symmetric*, *OF measurable-pair-swap′ f*] *distr-pair-swap*[*symmetric*])
qed

lemma (**in** *pair-sigma-finite*) *Fubini-integrable*:
fixes *f* :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *f* ∈ *borel-measurable* (*M1* $\bigotimes_M$ *M2*)
and *integ1*: *integrable M1* $(\lambda x.\ \int\ y.\ norm\ (f\ (x,\ y))\ \partial M2)$
and *integ2*: *AE x in M1*. *integrable M2* $(\lambda y.\ f\ (x,\ y))$
shows *integrable* (*M1* $\bigotimes_M$ *M2*) *f*
proof (*rule integrableI-bounded*)
have $(\int^+ p.\ norm\ (f\ p)\ \partial(M1 \bigotimes_M M2)) = (\int^+ x.\ (\int^+ y.\ norm\ (f\ (x,\ y))\ \partial M2)\ \partial M1)$
by (*simp add*: *M2.nn-integral-fst* [*symmetric*])
also have ... $= (\int^+ x.\ |\int y.\ norm\ (f\ (x,\ y))\ \partial M2|\ \partial M1)$
apply (*intro nn-integral-cong-AE*)
using *integ2*
proof *eventually-elim*
fix *x* **assume** *integrable M2* $(\lambda y.\ f\ (x,\ y))$
then have *f*: *integrable M2* $(\lambda y.\ norm\ (f\ (x,\ y)))$
by *simp*
then have $(\int^+ y.\ ennreal\ (norm\ (f\ (x,\ y)))\ \partial M2) = ennreal\ (LINT\ y|M2.\ norm\ (f\ (x,\ y)))$
by (*rule nn-integral-eq-integral*) *simp*
also have ... $=$ *ennreal* $|LINT\ y|M2.\ norm\ (f\ (x,\ y))|$
using *f* **by** *simp*
finally show $(\int^+ y.\ ennreal\ (norm\ (f\ (x,\ y)))\ \partial M2) = ennreal\ |LINT\ y|M2.\ norm\ (f\ (x,\ y))|$ **.**
qed
also have ... $< \infty$
using *integ1* **by** (*simp add*: *integrable-iff-bounded integral-nonneg-AE*)
finally show $(\int^+ p.\ norm\ (f\ p)\ \partial(M1 \bigotimes_M M2)) < \infty$ **.**
qed *fact*

lemma (**in** *pair-sigma-finite*) *emeasure-pair-measure-finite*:
assumes *A*: *A* ∈ *sets* (*M1* $\bigotimes_M$ *M2*) **and** *finite*: *emeasure* (*M1* $\bigotimes_M$ *M2*) *A* $< \infty$
shows *AE x in M1*. *emeasure M2* $\{y \in space\ M2.\ (x,\ y) \in A\} < \infty$
proof −
from *M2.emeasure-pair-measure-alt*[*OF A*] *finite*
have $(\int^+ x.\ emeasure\ M2\ (Pair\ x\ -`\ A)\ \partial M1) \neq \infty$
by *simp*
then have *AE x in M1*. *emeasure M2* $(Pair\ x\ -`\ A) \neq \infty$
by (*rule nn-integral-PInf-AE*[*rotated*]) (*intro M2.measurable-emeasure-Pair A*)

moreover have $\bigwedge x.\ x \in$ *space M1* $\Longrightarrow$ *Pair x* $-`$ *A* $=$ $\{y \in$ *space M2*. $(x,\ y) \in A\}$
using *sets.sets-into-space*[*OF A*] **by** (*auto simp*: *space-pair-measure*)
ultimately show *?thesis* **by** (*auto simp*: *less-top*)
qed

lemma (**in** *pair-sigma-finite*) *AE-integrable-fst'*:
fixes *f* :: - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *integrable* (*M1* $\bigotimes_M$ *M2*) *f*
shows *AE x in M1*. *integrable M2* ($\lambda y.$ *f* $(x,\ y)$)
proof −
have $(\int^+ x.\ (\int^+ y.$ *norm* (*f* $(x,\ y)$) ∂*M2*) ∂*M1*) $=$ $(\int^+ x.$ *norm* (*f x*) ∂(*M1* $\bigotimes_M$ *M2*))
by (*rule M2.nn-integral-fst*) *simp*
also have $(\int^+ x.$ *norm* (*f x*) ∂(*M1* $\bigotimes_M$ *M2*)) $\neq \infty$
using *f* **unfolding** *integrable-iff-bounded* **by** *simp*
finally have *AE x in M1*. $(\int^+ y.$ *norm* (*f* $(x,\ y)$) ∂*M2*) $\neq \infty$
by (*intro nn-integral-PInf-AE M2.borel-measurable-nn-integral*)
(*auto simp*: *measurable-split-conv*)
with *AE-space* **show** *?thesis*
by *eventually-elim*
(*auto simp*: *integrable-iff-bounded measurable-compose*[*OF* - *borel-measurable-integrable*[*OF f*]] *less-top*)
qed

lemma (**in** *pair-sigma-finite*) *integrable-fst'*:
fixes *f* :: - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *integrable* (*M1* $\bigotimes_M$ *M2*) *f*
shows *integrable M1* ($\lambda x.$ $\int y.$ *f* $(x,\ y)$ ∂*M2*)
unfolding *integrable-iff-bounded*
proof
show ($\lambda x.$ $\int$ $y.$ *f* $(x,\ y)$ ∂*M2*) $\in$ *borel-measurable M1*
by (*rule M2.borel-measurable-lebesgue-integral*) *simp*
have $(\int^+ x.$ *ennreal* (*norm* ($\int$ $y.$ *f* $(x,\ y)$ ∂*M2*)) ∂*M1*) $\leq$ $(\int^+ x.\ (\int^+ y.$ *norm* (*f* $(x,\ y)$) ∂*M2*) ∂*M1*)
using *AE-integrable-fst'*[*OF f*] **by** (*auto intro*!: *nn-integral-mono-AE integral-norm-bound-ennreal*)
also have $(\int^+ x.\ (\int^+ y.$ *norm* (*f* $(x,\ y)$) ∂*M2*) ∂*M1*) $=$ $(\int^+ x.$ *norm* (*f x*) ∂(*M1* $\bigotimes_M$ *M2*))
by (*rule M2.nn-integral-fst*) *simp*
also have $(\int^+ x.$ *norm* (*f x*) ∂(*M1* $\bigotimes_M$ *M2*)) $< \infty$
using *f* **unfolding** *integrable-iff-bounded* **by** *simp*
finally show $(\int^+$ $x.$ *ennreal* (*norm* ($\int$ $y.$ *f* $(x,\ y)$ ∂*M2*)) ∂*M1*) $< \infty$.
qed

lemma (**in** *pair-sigma-finite*) *integral-fst'*:
fixes *f* :: - $\Rightarrow$ -::{*banach*, *second-countable-topology*}
assumes *f*: *integrable* (*M1* $\bigotimes_M$ *M2*) *f*
shows $(\int x.\ (\int y.$ *f* $(x,\ y)$ ∂*M2*) ∂*M1*) $=$ *integral*^{L} (*M1* $\bigotimes_M$ *M2*) *f*
using *f* **proof** *induct*

case (*base A c*)
have *A*[*measurable*]: $A \in sets\ (M1 \bigotimes_M M2)$ **by** *fact*

have *eq*: $\bigwedge x\ y.\ x \in space\ M1 \Longrightarrow indicator\ A\ (x, y) = indicator\ \{y \in space\ M2.\ (x, y) \in A\}\ y$
using *sets.sets-into-space*[*OF A*] **by** (*auto split*: *split-indicator simp*: *space-pair-measure*)

have *int-A*: *integrable* $(M1 \bigotimes_M M2)$ (*indicator A* :: - ⇒ *real*)
using *base* **by** (*rule integrable-real-indicator*)

have $(\int x.\ \int y.\ indicator\ A\ (x, y) *_R c\ \partial M2\ \partial M1) = (\int x.\ measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\} *_R c\ \partial M1)$
proof (*intro integral-cong-AE*, *simp*, *simp*)
from *AE-integrable-fst'*[*OF int-A*] *AE-space*
show *AE x in M1*. $(\int y.\ indicator\ A\ (x, y) *_R c\ \partial M2) = measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\} *_R c$
by *eventually-elim* (*simp add*: *eq integrable-indicator-iff*)
qed
also have ... = $measure\ (M1 \bigotimes_M M2)\ A *_R c$
proof (*subst integral-scaleR-left*)
have $(\int^+ x.\ ennreal\ (measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\})\ \partial M1) =$
$(\int^+ x.\ emeasure\ M2\ \{y \in space\ M2.\ (x, y) \in A\}\ \partial M1)$
using *emeasure-pair-measure-finite*[*OF base*]
by (*intro nn-integral-cong-AE*, *eventually-elim*) (*simp add*: *emeasure-eq-ennreal-measure*)
also have ... = $emeasure\ (M1 \bigotimes_M M2)\ A$
using *sets.sets-into-space*[*OF A*]
by (*subst M2.emeasure-pair-measure-alt*)
(*auto intro*!: *nn-integral-cong arg-cong*[**where** *f*=*emeasure M2*] *simp*: *space-pair-measure*)
finally have ∗: $(\int^+ x.\ ennreal\ (measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\})\ \partial M1) = emeasure\ (M1 \bigotimes_M M2)\ A$ **.**

from *base* ∗ **show** *integrable M1* $(\lambda x.\ measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\})$
by (*simp add*: *integrable-iff-bounded*)
then have $(\int x.\ measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\}\ \partial M1) =$
$(\int^+ x.\ ennreal\ (measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\})\ \partial M1)$
by (*rule nn-integral-eq-integral*[*symmetric*]) *simp*
also note ∗
finally show $(\int x.\ measure\ M2\ \{y \in space\ M2.\ (x, y) \in A\}\ \partial M1) *_R c = measure\ (M1 \bigotimes_M M2)\ A *_R c$
using *base* **by** (*simp add*: *emeasure-eq-ennreal-measure*)
qed
also have ... = $(\int a.\ indicator\ A\ a *_R c\ \partial(M1 \bigotimes_M M2))$
using *base* **by** *simp*
finally show *?case* **.**
next
case (*add f g*)
then have [*measurable*]: $f \in borel\text{-}measurable\ (M1 \bigotimes_M M2)$ $g \in borel\text{-}measurable$

```
(M1 ⨂M M2)
    by auto
  have (∫ x. ∫ y. f (x, y) + g (x, y) ∂M2 ∂M1) =
    (∫ x. (∫ y. f (x, y) ∂M2) + (∫ y. g (x, y) ∂M2) ∂M1)
    apply (rule integral-cong-AE)
    apply simp-all
    using AE-integrable-fst'[OF add(1)] AE-integrable-fst'[OF add(3)]
    apply eventually-elim
    apply simp
    done
  also have ... = (∫ x. f x ∂(M1 ⨂M M2)) + (∫ x. g x ∂(M1 ⨂M M2))
    using integrable-fst'[OF add(1)] integrable-fst'[OF add(3)] add(2,4) by simp
  finally show ?case
    using add by simp
next
  case (lim f s)
 then have [measurable]: f ∈ borel-measurable (M1 ⨂M M2) ⋀i. s i ∈ borel-measurable
(M1 ⨂M M2)
    by auto

  show ?case
  proof (rule LIMSEQ-unique)
    show (λi. integral^L (M1 ⨂M M2) (s i)) ⟶ integral^L (M1 ⨂M M2) f
    proof (rule integral-dominated-convergence)
      show integrable (M1 ⨂M M2) (λx. 2 * norm (f x))
        using lim(5) by auto
    qed (insert lim, auto)
     have (λi. ∫ x. ∫ y. s i (x, y) ∂M2 ∂M1) ⟶ ∫ x. ∫ y. f (x, y) ∂M2
∂M1
    proof (rule integral-dominated-convergence)
      have AE x in M1. ∀ i. integrable M2 (λy. s i (x, y))
        unfolding AE-all-countable using AE-integrable-fst'[OF lim(1)] ..
      with AE-space AE-integrable-fst'[OF lim(5)]
      show AE x in M1. (λi. ∫ y. s i (x, y) ∂M2) ⟶ ∫ y. f (x, y) ∂M2
      proof eventually-elim
        fix x assume x: x ∈ space M1 and
          s: ∀ i. integrable M2 (λy. s i (x, y)) and f: integrable M2 (λy. f (x, y))
        show (λi. ∫ y. s i (x, y) ∂M2) ⟶ ∫ y. f (x, y) ∂M2
        proof (rule integral-dominated-convergence)
          show integrable M2 (λy. 2 * norm (f (x, y)))
             using f by auto
          show AE xa in M2. (λi. s i (x, xa)) ⟶ f (x, xa)
            using x lim(3) by (auto simp: space-pair-measure)
          show ⋀i. AE xa in M2. norm (s i (x, xa)) ≤ 2 * norm (f (x, xa))
            using x lim(4) by (auto simp: space-pair-measure)
        qed (insert x, measurable)
      qed
      show integrable M1 (λx. (∫ y. 2 * norm (f (x, y)) ∂M2))
        by (intro integrable-mult-right integrable-norm integrable-fst' lim)
```

fix *i* **show** *AE x in M1. norm* $(\int\ y.\ s\ i\ (x,\ y)\ \partial M2) \leq (\int\ y.\ 2 * norm\ (f\ (x,\ y))\ \partial M2)$
using *AE-space AE-integrable-fst'[OF lim(1), of i] AE-integrable-fst'[OF lim(5)]*
proof *eventually-elim*
fix *x* **assume** *x*: $x \in space\ M1$
and *s*: *integrable M2* $(\lambda y.\ s\ i\ (x,\ y))$ **and** *f*: *integrable M2* $(\lambda y.\ f\ (x,\ y))$
from *s* **have** $norm\ (\int\ y.\ s\ i\ (x,\ y)\ \partial M2) \leq (\int^+ y.\ norm\ (s\ i\ (x,\ y))\ \partial M2)$
by (*rule integral-norm-bound-ennreal*)
also have $\ldots \leq (\int^+ y.\ 2 * norm\ (f\ (x,\ y))\ \partial M2)$
using *x lim* **by** (*auto intro*!: *nn-integral-mono simp*: *space-pair-measure*)
also have $\ldots = (\int y.\ 2 * norm\ (f\ (x,\ y))\ \partial M2)$
using *f* **by** (*intro nn-integral-eq-integral*) *auto*
finally show $norm\ (\int\ y.\ s\ i\ (x,\ y)\ \partial M2) \leq (\int\ y.\ 2 * norm\ (f\ (x,\ y))\ \partial M2)$
by *simp*
qed
qed *simp-all*
then show $(\lambda i.\ integral^L\ (M1 \bigotimes_M M2)\ (s\ i)) \longrightarrow \int\ x.\ \int\ y.\ f\ (x,\ y)\ \partial M2\ \partial M1$
using *lim* **by** *simp*
qed
qed

lemma (**in** *pair-sigma-finite*)
fixes *f* :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*: *integrable* $(M1 \bigotimes_M M2)$ (*case-prod f*)
shows *AE-integrable-fst*: *AE x in M1. integrable M2* $(\lambda y.\ f\ x\ y)$ (**is** *?AE*)
and *integrable-fst*: *integrable M1* $(\lambda x.\ \int y.\ f\ x\ y\ \partial M2)$ (**is** *?INT*)
and *integral-fst*: $(\int x.\ (\int y.\ f\ x\ y\ \partial M2)\ \partial M1) = integral^L\ (M1 \bigotimes_M M2)\ (\lambda(x,\ y).\ f\ x\ y)$ (**is** *?EQ*)
using *AE-integrable-fst'[OF f] integrable-fst'[OF f] integral-fst'[OF f]* **by** *auto*

lemma (**in** *pair-sigma-finite*)
fixes *f* :: - ⇒ - ⇒ -::{*banach*, *second-countable-topology*}
assumes *f*[*measurable*]: *integrable* $(M1 \bigotimes_M M2)$ (*case-prod f*)
shows *AE-integrable-snd*: *AE y in M2. integrable M1* $(\lambda x.\ f\ x\ y)$ (**is** *?AE*)
and *integrable-snd*: *integrable M2* $(\lambda y.\ \int x.\ f\ x\ y\ \partial M1)$ (**is** *?INT*)
and *integral-snd*: $(\int y.\ (\int x.\ f\ x\ y\ \partial M1)\ \partial M2) = integral^L\ (M1 \bigotimes_M M2)$ (*case-prod f*) (**is** *?EQ*)
proof −
interpret *Q*: *pair-sigma-finite M2 M1* **..**
have *Q-int*: *integrable* $(M2 \bigotimes_M M1)$ $(\lambda(x,\ y).\ f\ y\ x)$
using *f* **unfolding** *integrable-product-swap-iff*[*symmetric*] **by** *simp*
show *?AE* **using** *Q.AE-integrable-fst'*[*OF Q-int*] **by** *simp*
show *?INT* **using** *Q.integrable-fst'*[*OF Q-int*] **by** *simp*
show *?EQ* **using** *Q.integral-fst'*[*OF Q-int*]
using *integral-product-swap*[*of case-prod f*] **by** *simp*
qed

lemma (**in** *pair-sigma-finite*) *Fubini-integral*:
fixes f :: - ⇒ - ⇒ - :: {*banach*, *second-countable-topology*}
assumes f: *integrable* (*M1* $\bigotimes_M$ *M2*) (*case-prod f*)
shows ($\int y.$ ($\int x.$ *f x y* ∂*M1*) ∂*M2*) = ($\int x.$ ($\int y.$ *f x y* ∂*M2*) ∂*M1*)
unfolding *integral-snd*[*OF assms*] *integral-fst*[*OF assms*] **..**

lemma (**in** *product-sigma-finite*) *product-integral-singleton*:
fixes f :: - ⇒ -::{*banach*, *second-countable-topology*}
shows $f \in$ *borel-measurable* (*M i*) ⟹ ($\int x.$ *f* (*x i*) ∂Pi_M {*i*} *M*) = *integral*^{L} (*M i*) *f*
apply (*subst distr-singleton*[*symmetric*])
apply (*subst integral-distr*)
apply *simp-all*
done

lemma (**in** *product-sigma-finite*) *product-integral-fold*:
fixes f :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *IJ*[*simp*]: $I \cap J = \{\}$ **and** *fin*: *finite I finite J*
and f: *integrable* (Pi_M ($I \cup J$) *M*) *f*
shows *integral*^{L} (Pi_M ($I \cup J$) *M*) *f* = ($\int x.$ ($\int y.$ *f* (*merge I J* (*x*, *y*)) ∂Pi_M *J M*) ∂Pi_M *I M*)
proof −
interpret *I*: *finite-product-sigma-finite M I* **by** *standard fact*
interpret *J*: *finite-product-sigma-finite M J* **by** *standard fact*
have *finite* ($I \cup J$) **using** *fin* **by** *auto*
interpret *IJ*: *finite-product-sigma-finite M I* ∪ *J* **by** *standard fact*
interpret *P*: *pair-sigma-finite* Pi_M *I M* Pi_M *J M* **..**
let *?M* = *merge I J*
let *?f* = λ*x*. *f* (*?M x*)
from *f* **have** *f-borel*: $f \in$ *borel-measurable* (Pi_M ($I \cup J$) *M*)
by *auto*
have *P-borel*: (λ*x*. *f* (*merge I J x*)) ∈ *borel-measurable* (Pi_M *I M* $\bigotimes_M$ Pi_M *J M*)
using *measurable-comp*[*OF measurable-merge f-borel*] **by** (*simp add*: *comp-def*)
have *f-int*: *integrable* (Pi_M *I M* $\bigotimes_M$ Pi_M *J M*) *?f*
by (*rule integrable-distr*[*OF measurable-merge*]) (*simp add*: *distr-merge*[*OF IJ fin*] *f*)
show *?thesis*
apply (*subst distr-merge*[*symmetric*, *OF IJ fin*])
apply (*subst integral-distr*[*OF measurable-merge f-borel*])
apply (*subst P.integral-fst'*[*symmetric*, *OF f-int*])
apply *simp*
done
qed

lemma (**in** *product-sigma-finite*) *product-integral-insert*:
fixes f :: - ⇒ -::{*banach*, *second-countable-topology*}
assumes *I*: *finite I i* ∉ *I*

```
    and f: integrable (Pi_M (insert i I) M) f
  shows integral^L (Pi_M (insert i I) M) f = (∫ x. (∫ y. f (x(i:=y)) ∂M i) ∂Pi_M
I M)
proof −
  have integral^L (Pi_M (insert i I) M) f = integral^L (Pi_M (I ∪ {i}) M) f
    by simp
  also have ... = (∫ x. (∫ y. f (merge I {i} (x,y)) ∂Pi_M {i} M) ∂Pi_M I M)
    using f I by (intro product-integral-fold) auto
  also have ... = (∫ x. (∫ y. f (x(i := y)) ∂M i) ∂Pi_M I M)
  proof (rule integral-cong[OF refl], subst product-integral-singleton[symmetric])
    fix x assume x: x ∈ space (Pi_M I M)
    have f-borel: f ∈ borel-measurable (Pi_M (insert i I) M)
      using f by auto
    show (λy. f (x(i := y))) ∈ borel-measurable (M i)
      using measurable-comp[OF measurable-component-update f-borel, OF x ‹i ∉
I›]
      unfolding comp-def .
    from x I show (∫ y. f (merge I {i} (x,y)) ∂Pi_M {i} M) = (∫ xa. f (x(i :=
xa i)) ∂Pi_M {i} M)
      by (auto intro!: integral-cong arg-cong[where f=f] simp: merge-def space-PiM
extensional-def PiE-def)
  qed
  finally show ?thesis .
qed

lemma (in product-sigma-finite) product-integrable-setprod:
  fixes f :: 'i ⇒ 'a ⇒ -::{real-normed-field,banach,second-countable-topology}
  assumes [simp]: finite I and integrable: ⋀i. i ∈ I ⟹ integrable (M i) (f i)
  shows integrable (Pi_M I M) (λx. (∏ i∈I. f i (x i))) (is integrable - ?f)
proof (unfold integrable-iff-bounded, intro conjI)
  interpret finite-product-sigma-finite M I by standard fact

  show ?f ∈ borel-measurable (Pi_M I M)
    using assms by simp
  have (∫⁺ x. ennreal (norm (∏ i∈I. f i (x i))) ∂Pi_M I M) =
      (∫⁺ x. (∏ i∈I. ennreal (norm (f i (x i)))) ∂Pi_M I M)
    by (simp add: setprod-norm setprod-ennreal)
  also have ... = (∏ i∈I. ∫⁺ x. ennreal (norm (f i x)) ∂M i)
    using assms by (intro product-nn-integral-setprod) auto
  also have ... < ∞
   using integrable by (simp add: less-top[symmetric] ennreal-setprod-eq-top integrable-iff-bounded)
  finally show (∫⁺ x. ennreal (norm (∏ i∈I. f i (x i))) ∂Pi_M I M) < ∞ .
qed

lemma (in product-sigma-finite) product-integral-setprod:
  fixes f :: 'i ⇒ 'a ⇒ -::{real-normed-field,banach,second-countable-topology}
  assumes finite I and integrable: ⋀i. i ∈ I ⟹ integrable (M i) (f i)
  shows (∫ x. (∏ i∈I. f i (x i)) ∂Pi_M I M) = (∏ i∈I. integral^L (M i) (f i))
using assms proof induct
```

case *empty*
interpret *finite-measure* $Pi_M\ \{\}\ M$
by *rule* (*simp add*: *space-PiM*)
show *?case* **by** (*simp add*: *space-PiM measure-def*)
next
case (*insert i I*)
then have *iI*: *finite* (*insert i I*) **by** *auto*
then have *prod*: $\bigwedge J.\ J \subseteq$ *insert i I* $\Longrightarrow$
integrable ($Pi_M\ J\ M$) ($\lambda x.\ (\prod i{\in}J.\ f\ i\ (x\ i))$)
by (*intro product-integrable-setprod insert*(*4*)) (*auto intro*: *finite-subset*)
interpret *I*: *finite-product-sigma-finite M I* **by** *standard fact*
have $*$: $\bigwedge x\ y.\ (\prod j{\in}I.\ f\ j\ (\textit{if}\ j = i\ \textit{then}\ y\ \textit{else}\ x\ j)) = (\prod j{\in}I.\ f\ j\ (x\ j))$
using ‹$i \notin I$›
by (*auto intro*!: *setprod.cong*)
show *?case*
unfolding *product-integral-insert*[*OF insert*(*1*,*2*) *prod*[*OF subset-refl*]]
by (*simp add*: $*$ *insert prod subset-insertI*)
qed

lemma *integrable-subalgebra*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
assumes *borel*: *f* $\in$ *borel-measurable N*
and *N*: *sets N* $\subseteq$ *sets M space N* = *space M* $\bigwedge A.\ A \in$ *sets N* $\Longrightarrow$ *emeasure N A* = *emeasure M A*
shows *integrable N f* $\longleftrightarrow$ *integrable M f* (**is** *?P*)
proof −
have *f* $\in$ *borel-measurable M*
using *assms* **by** (*auto simp*: *measurable-def*)
with *assms* **show** *?thesis*
using *assms* **by** (*auto simp*: *integrable-iff-bounded nn-integral-subalgebra*)
qed

lemma *integral-subalgebra*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::{*banach*, *second-countable-topology*}
assumes *borel*: *f* $\in$ *borel-measurable N*
and *N*: *sets N* $\subseteq$ *sets M space N* = *space M* $\bigwedge A.\ A \in$ *sets N* $\Longrightarrow$ *emeasure N A* = *emeasure M A*
shows $\textit{integral}^L\ N\ f = \textit{integral}^L\ M\ f$
proof *cases*
assume *integrable N f*
then show *?thesis*
proof *induct*
case *base* **with** *assms* **show** *?case* **by** (*auto simp*: *subset-eq measure-def*)
next
case (*add f g*)
then have $(\int\ a.\ f\ a + g\ a\ \partial N) = \textit{integral}^L\ M\ f + \textit{integral}^L\ M\ g$
by *simp*
also have ... = $(\int\ a.\ f\ a + g\ a\ \partial M)$
using *add integrable-subalgebra*[*OF - N*, *of f*] *integrable-subalgebra*[*OF - N*, *of g*] **by** *simp*

```
    finally show ?case .
  next
    case (lim f s)
    then have M: ⋀i. integrable M (s i) integrable M f
      using integrable-subalgebra[OF - N, of f] integrable-subalgebra[OF - N, of s i
for i] by simp-all
    show ?case
    proof (intro LIMSEQ-unique)
      show (λi. integral^L N (s i)) ⟶ integral^L N f
        apply (rule integral-dominated-convergence[where w=λx. 2 * norm (f x)])
        using lim
        apply auto
        done
      show (λi. integral^L N (s i)) ⟶ integral^L M f
        unfolding lim
        apply (rule integral-dominated-convergence[where w=λx. 2 * norm (f x)])
        using lim M N(2)
        apply auto
        done
    qed
  qed
qed (simp add: not-integrable-integral-eq integrable-subalgebra[OF assms])

hide-const (open) simple-bochner-integral
hide-const (open) simple-bochner-integrable

end
```

9 Caratheodory Extension Theorem

```
theory Caratheodory
  imports Measure-Space
begin
```

Originally from the Hurd/Coble measure theory development, translated by Lawrence Paulson.

```
lemma suminf-ennreal-2dimen:
  fixes f:: nat × nat ⇒ ennreal
  assumes ⋀m. g m = (∑ n. f (m,n))
  shows (∑ i. f (prod-decode i)) = suminf g
proof −
  have g-def: g = (λm. (∑ n. f (m,n)))
    using assms by (simp add: fun-eq-iff)
  have reindex: ⋀B. (∑ x∈B. f (prod-decode x)) = setsum f (prod-decode ' B)
    by (simp add: setsum.reindex[OF inj-prod-decode] comp-def)
  have (SUP n. ∑ i<n. f (prod-decode i)) = (SUP p : UNIV × UNIV. ∑ i<fst
p. ∑ n<snd p. f (i, n))
  proof (intro SUP-eq; clarsimp simp: setsum.cartesian-product reindex)
    fix n
```

let *?M* = λf. *Suc* (*Max* (*f* ‘ *prod-decode* ‘ {..<*n*}))
{ fix *a b x* **assume** $x < n$ **and** [*symmetric*]: (a, b) = *prod-decode x*
then have $a <$ *?M fst b* $<$ *?M snd*
by (*auto intro*!: *Max-ge le-imp-less-Suc image-eqI*) **}**
then have *setsum f* (*prod-decode* ‘ {..<*n*}) ≤ *setsum f* ({..<*?M fst*} × {..<*?M snd*})
by (*auto intro*!: *setsum-mono3*)
then show ∃ *a b*. *setsum f* (*prod-decode* ‘ {..<*n*}) ≤ *setsum f* ({..<*a*} × {..<*b*})
by *auto*
next
fix *a b*
let *?M* = *prod-decode* ‘ {..<*Suc* (*Max* (*prod-encode* ‘ ({..<*a*} × {..<*b*})))}
{ fix *a′ b′* **assume** $a' < a$ $b' < b$ **then have** $(a', b') \in$ *?M*
by (*auto intro*!: *Max-ge le-imp-less-Suc image-eqI*[**where** *x*=*prod-encode* (a', b')]) **}**
then have *setsum f* ({..<*a*} × {..<*b*}) ≤ *setsum f ?M*
by (*auto intro*!: *setsum-mono3*)
then show ∃ *n*. *setsum f* ({..<*a*} × {..<*b*}) ≤ *setsum f* (*prod-decode* ‘ {..<*n*})
by *auto*
qed
also have ... = (*SUP p*. $\sum i$<*p*. $\sum n$. *f* (i, n))
unfolding *suminf-setsum*[*OF summableI*, *symmetric*]
by (*simp add*: *suminf-eq-SUP SUP-pair setsum.commute*[*of* - {..< *fst* -}])
finally show *?thesis* **unfolding** *g-def*
by (*simp add*: *suminf-eq-SUP*)
qed

9.1 Characterizations of Measures

definition *outer-measure-space* **where**
outer-measure-space M f ⟷ *positive M f* ∧ *increasing M f* ∧ *countably-subadditive M f*

9.1.1 Lambda Systems

definition *lambda-system* :: *'a set* ⇒ *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *'a set set* **where**
lambda-system Ω *M f* = {*l* ∈ *M*. ∀ *x* ∈ *M*. *f* $(l \cap x)$ + *f* $((\Omega - l) \cap x)$ = *f x*}

lemma (**in** *algebra*) *lambda-system-eq*:
lambda-system Ω *M f* = {*l* ∈ *M*. ∀ *x* ∈ *M*. *f* $(x \cap l)$ + *f* $(x - l)$ = *f x*}
proof −
have [*simp*]: ⋀*l x*. *l* ∈ *M* ⟹ *x* ∈ *M* ⟹ $(\Omega - l) \cap x = x - l$
by (*metis Int-Diff Int-absorb1 Int-commute sets-into-space*)
show *?thesis*
by (*auto simp add*: *lambda-system-def*) (*metis Int-commute*)+
qed

lemma (**in** *algebra*) *lambda-system-empty*: *positive M f* ⟹ {} ∈ *lambda-system* Ω *M f*

by (*auto simp add*: *positive-def lambda-system-eq*)

lemma *lambda-system-sets*: $x \in$ *lambda-system* Ω M $f \Longrightarrow x \in M$
by (*simp add*: *lambda-system-def*)

lemma (**in** *algebra*) *lambda-system-Compl*:
fixes f:: $'a$ *set* $\Rightarrow$ *ennreal*
assumes x: $x \in$ *lambda-system* Ω M f
shows $\Omega - x \in$ *lambda-system* Ω M f
proof −
have $x \subseteq \Omega$
by (*metis sets-into-space lambda-system-sets x*)
hence $\Omega - (\Omega - x) = x$
by (*metis double-diff equalityE*)
with x **show** *?thesis*
by (*force simp add*: *lambda-system-def ac-simps*)
qed

lemma (**in** *algebra*) *lambda-system-Int*:
fixes f:: $'a$ *set* $\Rightarrow$ *ennreal*
assumes xl: $x \in$ *lambda-system* Ω M f **and** yl: $y \in$ *lambda-system* Ω M f
shows $x \cap y \in$ *lambda-system* Ω M f
proof −
from *xl yl* **show** *?thesis*
proof (*auto simp add*: *positive-def lambda-system-eq Int*)
fix u
assume x: $x \in M$ **and** y: $y \in M$ **and** u: $u \in M$
and fx: $\forall z{\in}M.\ f\ (z \cap x) + f\ (z - x) = f\ z$
and fy: $\forall z{\in}M.\ f\ (z \cap y) + f\ (z - y) = f\ z$
have $u - x \cap y \in M$
by (*metis Diff Diff-Int Un u x y*)
moreover
have $(u - (x \cap y)) \cap y = u \cap y - x$ **by** *blast*
moreover
have $u - x \cap y - y = u - y$ **by** *blast*
ultimately
have *ey*: $f\ (u - x \cap y) = f\ (u \cap y - x) + f\ (u - y)$ **using** *fy*
by *force*
have $f\ (u \cap (x \cap y)) + f\ (u - x \cap y)$
$= (f\ (u \cap (x \cap y)) + f\ (u \cap y - x)) + f\ (u - y)$
by (*simp add*: *ey ac-simps*)
also have ... = $(f\ ((u \cap y) \cap x) + f\ (u \cap y - x)) + f\ (u - y)$
by (*simp add*: *Int-ac*)
also have ... = $f\ (u \cap y) + f\ (u - y)$
using *fx* [*THEN bspec*, *of* $u \cap y$] *Int y u*
by *force*
also have ... = $f\ u$
by (*metis fy u*)
finally show $f\ (u \cap (x \cap y)) + f\ (u - x \cap y) = f\ u$ **.**

qed
qed

lemma (**in** *algebra*) *lambda-system-Un*:
fixes f:: $'a$ *set* $\Rightarrow$ *ennreal*
assumes *xl*: $x \in$ *lambda-system* Ω M f **and** *yl*: $y \in$ *lambda-system* Ω M f
shows $x \cup y \in$ *lambda-system* Ω M f
proof −
have $(\Omega - x) \cap (\Omega - y) \in M$
by (*metis Diff-Un Un compl-sets lambda-system-sets xl yl*)
moreover
have $x \cup y = \Omega - ((\Omega - x) \cap (\Omega - y))$
by *auto* (*metis subsetD lambda-system-sets sets-into-space xl yl*)+
ultimately show *?thesis*
by (*metis lambda-system-Compl lambda-system-Int xl yl*)
qed

lemma (**in** *algebra*) *lambda-system-algebra*:
positive M $f \Longrightarrow$ *algebra* Ω (*lambda-system* Ω M f)
apply (*auto simp add*: *algebra-iff-Un*)
apply (*metis lambda-system-sets set-mp sets-into-space*)
apply (*metis lambda-system-empty*)
apply (*metis lambda-system-Compl*)
apply (*metis lambda-system-Un*)
done

lemma (**in** *algebra*) *lambda-system-strong-additive*:
assumes *z*: $z \in M$ **and** *disj*: $x \cap y = \{\}$
and *xl*: $x \in$ *lambda-system* Ω M f **and** *yl*: $y \in$ *lambda-system* Ω M f
shows $f\ (z \cap (x \cup y)) = f\ (z \cap x) + f\ (z \cap y)$
proof −
have $z \cap x = (z \cap (x \cup y)) \cap x$ **using** *disj* **by** *blast*
moreover
have $z \cap y = (z \cap (x \cup y)) - x$ **using** *disj* **by** *blast*
moreover
have $(z \cap (x \cup y)) \in M$
by (*metis Int Un lambda-system-sets xl yl z*)
ultimately show *?thesis* **using** *xl yl*
by (*simp add*: *lambda-system-eq*)
qed

lemma (**in** *algebra*) *lambda-system-additive*: *additive* (*lambda-system* Ω M f) f
proof (*auto simp add*: *additive-def*)
fix x **and** y
assume *disj*: $x \cap y = \{\}$
and *xl*: $x \in$ *lambda-system* Ω M f **and** *yl*: $y \in$ *lambda-system* Ω M f
hence $x \in M$ $y \in M$ **by** (*blast intro*: *lambda-system-sets*)+
thus $f\ (x \cup y) = f\ x + f\ y$
using *lambda-system-strong-additive* [*OF top disj xl yl*]

by (*simp add*: *Un*)
qed

lemma *lambda-system-increasing*: *increasing M f* $\Longrightarrow$ *increasing* (*lambda-system* Ω *M f*) *f*
by (*simp add*: *increasing-def lambda-system-def*)

lemma *lambda-system-positive*: *positive M f* $\Longrightarrow$ *positive* (*lambda-system* Ω *M f*) *f*
by (*simp add*: *positive-def lambda-system-def*)

lemma (**in** *algebra*) *lambda-system-strong-sum*:
fixes *A*:: *nat* $\Rightarrow$ *'a set* **and** *f* :: *'a set* $\Rightarrow$ *ennreal*
assumes *f*: *positive M f* **and** *a*: $a \in M$
and *A*: *range A* $\subseteq$ *lambda-system* Ω *M f*
and *disj*: *disjoint-family A*
shows $(\sum i = 0..<n.\ f\ (a \cap A\ i)) = f\ (a \cap (\bigcup i \in \{0..<n\}.\ A\ i))$
proof (*induct n*)
case *0* **show** *?case* **using** *f* **by** (*simp add*: *positive-def*)
next
case (*Suc n*)
have *2*: $A\ n \cap UNION\ \{0..<n\}\ A = \{\}$ **using** *disj*
by (*force simp add*: *disjoint-family-on-def neq-iff*)
have *3*: $A\ n \in$ *lambda-system* Ω *M f* **using** *A*
by *blast*
interpret *l*: *algebra* Ω *lambda-system* Ω *M f*
using *f* **by** (*rule lambda-system-algebra*)
have *4*: $UNION\ \{0..<n\}\ A \in$ *lambda-system* Ω *M f*
using *A l.UNION-in-sets* **by** *simp*
from *Suc.hyps* **show** *?case*
by (*simp add*: *atLeastLessThanSuc lambda-system-strong-additive* [*OF a 2 3 4*])
qed

lemma (**in** *sigma-algebra*) *lambda-system-caratheodory*:
assumes *oms*: *outer-measure-space M f*
and *A*: *range A* $\subseteq$ *lambda-system* Ω *M f*
and *disj*: *disjoint-family A*
shows $(\bigcup i.\ A\ i) \in$ *lambda-system* Ω *M f* $\wedge$ $(\sum i.\ f\ (A\ i)) = f\ (\bigcup i.\ A\ i)$
proof −
have *pos*: *positive M f* **and** *inc*: *increasing M f*
and *csa*: *countably-subadditive M f*
by (*metis oms outer-measure-space-def*)+
have *sa*: *subadditive M f*
by (*metis countably-subadditive-subadditive csa pos*)
have *A'*: $\bigwedge S.\ A`S \subseteq$ (*lambda-system* Ω *M f*) **using** *A*
by *auto*
interpret *ls*: *algebra* Ω *lambda-system* Ω *M f*
using *pos* **by** (*rule lambda-system-algebra*)
have *A''*: *range A* $\subseteq$ *M*

by (*metis A image-subset-iff lambda-system-sets*)

have *U-in*: $(\bigcup i.\ A\ i) \in M$
by (*metis A'' countable-UN*)
have *U-eq*: $f\ (\bigcup i.\ A\ i) = (\sum i.\ f\ (A\ i))$
proof (*rule antisym*)
show $f\ (\bigcup i.\ A\ i) \leq (\sum i.\ f\ (A\ i))$
using *csa*[*unfolded countably-subadditive-def*] *A'' disj U-in* **by** *auto*
have *dis*: $\bigwedge N.$ *disjoint-family-on A* $\{..<N\}$ **by** (*intro disjoint-family-on-mono*[*OF - disj*]) *auto*
show $(\sum i.\ f\ (A\ i)) \leq f\ (\bigcup i.\ A\ i)$
using *ls.additive-sum* [*OF lambda-system-positive*[*OF pos*] *lambda-system-additive - A' dis*] *A''*
by (*intro suminf-le-const*[*OF summableI*]) (*auto intro*!: *increasingD*[*OF inc*] *countable-UN*)
qed
have $f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i)) = f\ a$
if *a* [*iff*]: $a \in M$ **for** *a*
proof (*rule antisym*)
have *range* $(\lambda i.\ a \cap A\ i) \subseteq M$ **using** *A''*
by *blast*
moreover
have *disjoint-family* $(\lambda i.\ a \cap A\ i)$ **using** *disj*
by (*auto simp add*: *disjoint-family-on-def*)
moreover
have $a \cap (\bigcup i.\ A\ i) \in M$
by (*metis Int U-in a*)
ultimately
have $f\ (a \cap (\bigcup i.\ A\ i)) \leq (\sum i.\ f\ (a \cap A\ i))$
using *csa*[*unfolded countably-subadditive-def*, *rule-format*, *of* $(\lambda i.\ a \cap A\ i)$]
by (*simp add*: *o-def*)
hence $f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i)) \leq (\sum i.\ f\ (a \cap A\ i)) + f\ (a - (\bigcup i.\ A\ i))$
by (*rule add-right-mono*)
also have $\ldots \leq f\ a$
proof (*intro ennreal-suminf-bound-add*)
fix *n*
have *UNION-in*: $(\bigcup i{\in}\{0..<n\}.\ A\ i) \in M$
by (*metis A'' UNION-in-sets*)
have *le-fa*: $f\ (UNION\ \{0..<n\}\ A \cap a) \leq f\ a$ **using** *A''*
by (*blast intro*: *increasingD* [*OF inc*] *A'' UNION-in-sets*)
have *ls*: $(\bigcup i{\in}\{0..<n\}.\ A\ i) \in$ *lambda-system* $\Omega\ M\ f$
using *ls.UNION-in-sets* **by** (*simp add*: *A*)
hence *eq-fa*: $f\ a = f\ (a \cap (\bigcup i{\in}\{0..<n\}.\ A\ i)) + f\ (a - (\bigcup i{\in}\{0..<n\}.\ A\ i))$
by (*simp add*: *lambda-system-eq UNION-in*)
have $f\ (a - (\bigcup i.\ A\ i)) \leq f\ (a - (\bigcup i{\in}\{0..<n\}.\ A\ i))$
by (*blast intro*: *increasingD* [*OF inc*] *UNION-in U-in*)
thus $(\sum i{<}n.\ f\ (a \cap A\ i)) + f\ (a - (\bigcup i.\ A\ i)) \leq f\ a$

by (*simp add*: *lambda-system-strong-sum pos A disj eq-fa add-left-mono atLeast0LessThan*[*symmetric*])
qed
finally show $f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i)) \leq f\ a$
by *simp*
next
have $f\ a \leq f\ (a \cap (\bigcup i.\ A\ i) \cup (a - (\bigcup i.\ A\ i)))$
by (*blast intro*: *increasingD* [*OF inc*] *U-in*)
also have $\ldots \leq\ f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i))$
by (*blast intro*: *subadditiveD* [*OF sa*] *U-in*)
finally show $f\ a \leq f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i))$ **.**
qed
thus *?thesis*
by (*simp add*: *lambda-system-eq sums-iff U-eq U-in*)
qed

lemma (**in** *sigma-algebra*) *caratheodory-lemma*:
assumes *oms*: *outer-measure-space M f*
defines $L \equiv$ *lambda-system* Ω *M f*
shows *measure-space* Ω *L f*
proof −
have *pos*: *positive M f*
by (*metis oms outer-measure-space-def*)
have *alg*: *algebra* Ω *L*
using *lambda-system-algebra* [*of f*, *OF pos*]
by (*simp add*: *algebra-iff-Un L-def*)
then
have *sigma-algebra* Ω *L*
using *lambda-system-caratheodory* [*OF oms*]
by (*simp add*: *sigma-algebra-disjoint-iff L-def*)
moreover
have *countably-additive L f positive L f*
using *pos lambda-system-caratheodory* [*OF oms*]
by (*auto simp add*: *lambda-system-sets L-def countably-additive-def positive-def*)
ultimately
show *?thesis*
using *pos* **by** (*simp add*: *measure-space-def*)
qed

definition *outer-measure* :: *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *'a set* ⇒ *ennreal*
where
outer-measure M f X =
(*INF A*:{*A*. *range A* ⊆ *M* ∧ *disjoint-family A* ∧ *X* ⊆ $(\bigcup i.\ A\ i)$}. $\sum i.\ f\ (A\ i)$)

lemma (**in** *ring-of-sets*) *outer-measure-agrees*:
assumes *posf*: *positive M f* **and** *ca*: *countably-additive M f* **and** *s*: $s \in M$
shows *outer-measure M f s* = *f s*
unfolding *outer-measure-def*

proof (*safe intro*!: *antisym INF-greatest*)
fix $A :: nat \Rightarrow 'a\ set$ **assume** A: *range* $A \subseteq M$ **and** dA: *disjoint-family* A **and** sA: $s \subseteq (\bigcup x.\ A\ x)$
have *inc*: *increasing* $M\ f$
by (*metis additive-increasing ca countably-additive-additive posf*)
have $f\ s = f\ (\bigcup i.\ A\ i \cap s)$
using sA **by** (*auto simp*: *Int-absorb1*)
also have $\ldots = (\sum i.\ f\ (A\ i \cap s))$
using sA dA A s
by (*intro ca*[*unfolded countably-additive-def*, *rule-format*, *symmetric*])
(*auto simp*: *Int-absorb1 disjoint-family-on-def*)
also have ... $\leq (\sum i.\ f\ (A\ i))$
using A s **by** (*auto intro*!: *suminf-le increasingD*[*OF inc*])
finally show $f\ s \leq (\sum i.\ f\ (A\ i))$ **.**
next
have $(\sum i.\ f\ (\textit{if } i = 0 \textit{ then } s \textit{ else } \{\})) \leq f\ s$
using *positiveD1*[*OF posf*] **by** (*subst suminf-finite*[*of* $\{0\}$]) *auto*
with s **show** (*INF* A:{A. *range* $A \subseteq M \wedge$ *disjoint-family* $A \wedge s \subseteq$ *UNION UNIV* A}. $\sum i.\ f\ (A\ i)) \leq f\ s$
by (*intro INF-lower2*[*of* $\lambda i.$ *if* $i = 0$ *then* s *else* $\{\}$])
(*auto simp*: *disjoint-family-on-def*)
qed

lemma *outer-measure-empty*:
positive $M\ f \Longrightarrow \{\} \in M \Longrightarrow$ *outer-measure* $M\ f\ \{\} = 0$
unfolding *outer-measure-def*
by (*intro antisym INF-lower2*[*of* λ-. $\{\}$]) (*auto simp*: *disjoint-family-on-def positive-def*)

lemma (**in** *ring-of-sets*) *positive-outer-measure*:
assumes *positive* $M\ f$ **shows** *positive* $(Pow\ \Omega)$ (*outer-measure* $M\ f$)
unfolding *positive-def* **by** (*auto simp*: *assms outer-measure-empty*)

lemma (**in** *ring-of-sets*) *increasing-outer-measure*: *increasing* $(Pow\ \Omega)$ (*outer-measure* $M\ f$)
by (*force simp*: *increasing-def outer-measure-def intro*!: *INF-greatest intro*: *INF-lower*)

lemma (**in** *ring-of-sets*) *outer-measure-le*:
assumes *pos*: *positive* $M\ f$ **and** *inc*: *increasing* $M\ f$ **and** A: *range* $A \subseteq M$ **and** X: $X \subseteq (\bigcup i.\ A\ i)$
shows *outer-measure* $M\ f\ X \leq (\sum i.\ f\ (A\ i))$
unfolding *outer-measure-def*
proof (*safe intro*!: *INF-lower2*[*of disjointed A*] *del*: *subsetI*)
show dA: *range* (*disjointed* A) $\subseteq M$
by (*auto intro*!: A *range-disjointed-sets*)
have $\forall n.\ f\ (\textit{disjointed } A\ n) \leq f\ (A\ n)$
by (*metis increasingD* [*OF inc*] *UNIV-I dA image-subset-iff disjointed-subset* A)
then show $(\sum i.\ f\ (\textit{disjointed } A\ i)) \leq (\sum i.\ f\ (A\ i))$

by (*blast intro*!: *suminf-le*)
qed (*auto simp*: *X UN-disjointed-eq disjoint-family-disjointed*)

lemma (**in** *ring-of-sets*) *outer-measure-close*:
outer-measure M f X < *e* ⟹ ∃ *A*. *range A* ⊆ *M* ∧ *disjoint-family A* ∧ *X* ⊆ (⋃ *i*. *A i*) ∧ (∑ *i*. *f* (*A i*)) < *e*
unfolding *outer-measure-def INF-less-iff* **by** *auto*

lemma (**in** *ring-of-sets*) *countably-subadditive-outer-measure*:
assumes *posf*: *positive M f* **and** *inc*: *increasing M f*
shows *countably-subadditive* (*Pow* Ω) (*outer-measure M f*)
proof (*simp add*: *countably-subadditive-def*, *safe*)
fix *A* :: *nat* ⇒ - **assume** *A*: *range A* ⊆ *Pow* (Ω) **and** *sb*: (⋃ *i*. *A i*) ⊆ Ω
let *?O* = *outer-measure M f*
show *?O* (⋃ *i*. *A i*) ≤ (∑ *n*. *?O* (*A n*))
proof (*rule ennreal-le-epsilon*)
fix *b* **and** *e* :: *real* **assume** *0* < *e* (∑ *n*. *outer-measure M f* (*A n*)) < *top*
then have ∗: ⋀*n*. *outer-measure M f* (*A n*) < *outer-measure M f* (*A n*) + *e* ∗ (*1*/*2*) ^*Suc n*
by (*auto simp add*: *less-top dest*!: *ennreal-suminf-lessD*)
obtain *B*
where *B*: ⋀*n*. *range* (*B n*) ⊆ *M*
and *sbB*: ⋀*n*. *A n* ⊆ (⋃ *i*. *B n i*)
and *Ble*: ⋀*n*. (∑ *i*. *f* (*B n i*)) ≤ *?O* (*A n*) + *e* ∗ (*1*/*2*) ^(*Suc n*)
by (*metis less-imp-le outer-measure-close*[*OF* ∗])

def *C* ≡ *case-prod B o prod-decode*
from *B* **have** *B-in-M*: ⋀*i j*. *B i j* ∈ *M*
by (*rule range-subsetD*)
then have *C*: *range C* ⊆ *M*
by (*auto simp add*: *C-def split-def*)
have *A-C*: (⋃ *i*. *A i*) ⊆ (⋃ *i*. *C i*)
using *sbB* **by** (*auto simp add*: *C-def subset-eq*) (*metis prod.case prod-encode-inverse*)

have *?O* (⋃ *i*. *A i*) ≤ *?O* (⋃ *i*. *C i*)
using *A-C A C* **by** (*intro increasing-outer-measure*[*THEN increasingD*]) (*auto dest*!: *sets-into-space*)
also have ... ≤ (∑ *i*. *f* (*C i*))
using *C* **by** (*intro outer-measure-le*[*OF posf inc*]) *auto*
also have ... = (∑ *n*. ∑ *i*. *f* (*B n i*))
using *B-in-M* **unfolding** *C-def comp-def* **by** (*intro suminf-ennreal-2dimen*) *auto*
also have ... ≤ (∑ *n*. *?O* (*A n*) + *e* ∗ (*1*/*2*) ^ *Suc n*)
using *B-in-M* **by** (*intro suminf-le suminf-nonneg allI Ble*) *auto*
also have ... = (∑ *n*. *?O* (*A n*)) + (∑ *n*. *ennreal e* ∗ *ennreal* ((*1*/*2*) ^ *Suc n*))
using ‹*0* < *e*› **by** (*subst suminf-add*[*symmetric*])
(*auto simp del*: *ennreal-suminf-cmult simp add*: *ennreal-mult*[*symmetric*])
also have ... = (∑ *n*. *?O* (*A n*)) + *e*

unfolding *ennreal-suminf-cmult*
by (*subst suminf-ennreal-eq*[*OF zero-le-power power-half-series*]) *auto*
finally show *?O* $(\bigcup i.\ A\ i) \leq (\sum n.\ ?O\ (A\ n)) + e$ **.**
qed
qed

lemma (**in** *ring-of-sets*) *outer-measure-space-outer-measure*:
positive M f $\Longrightarrow$ *increasing M f* $\Longrightarrow$ *outer-measure-space* (*Pow* Ω) (*outer-measure M f*)
by (*simp add*: *outer-measure-space-def*
positive-outer-measure increasing-outer-measure countably-subadditive-outer-measure)

lemma (**in** *ring-of-sets*) *algebra-subset-lambda-system*:
assumes *posf*: *positive M f* **and** *inc*: *increasing M f*
and *add*: *additive M f*
shows $M \subseteq$ *lambda-system* Ω (*Pow* Ω) (*outer-measure M f*)
proof (*auto dest*: *sets-into-space*
simp add: *algebra.lambda-system-eq* [*OF algebra-Pow*])
fix *x s* **assume** *x*: $x \in M$ **and** *s*: $s \subseteq \Omega$
have [*simp*]: $\bigwedge x.\ x \in M \Longrightarrow s \cap (\Omega - x) = s - x$ **using** *s*
by *blast*
have *outer-measure M f* $(s \cap x)$ + *outer-measure M f* $(s - x) \leq$ *outer-measure M f s*
unfolding *outer-measure-def*[*of M f s*]
proof (*safe intro*!: *INF-greatest*)
fix *A* :: *nat* $\Rightarrow$ *'a set* **assume** *A*: *disjoint-family A range A* $\subseteq M\ s \subseteq (\bigcup i.\ A\ i)$
have *outer-measure M f* $(s \cap x) \leq (\sum i.\ f\ (A\ i \cap x))$
unfolding *outer-measure-def*
proof (*safe intro*!: *INF-lower2*[*of* $\lambda i.\ A\ i \cap x$])
from *A*(*1*) **show** *disjoint-family* ($\lambda i.\ A\ i \cap x$)
by (*rule disjoint-family-on-bisimulation*) *auto*
qed (*insert x A*, *auto*)
moreover
have *outer-measure M f* $(s - x) \leq (\sum i.\ f\ (A\ i - x))$
unfolding *outer-measure-def*
proof (*safe intro*!: *INF-lower2*[*of* $\lambda i.\ A\ i - x$])
from *A*(*1*) **show** *disjoint-family* ($\lambda i.\ A\ i - x$)
by (*rule disjoint-family-on-bisimulation*) *auto*
qed (*insert x A*, *auto*)
ultimately have *outer-measure M f* $(s \cap x)$ + *outer-measure M f* $(s - x) \leq$
$(\sum i.\ f\ (A\ i \cap x)) + (\sum i.\ f\ (A\ i - x))$ **by** (*rule add-mono*)
also have $\ldots = (\sum i.\ f\ (A\ i \cap x) + f\ (A\ i - x))$
using *A*(*2*) *x posf* **by** (*subst suminf-add*) (*auto simp*: *positive-def*)
also have $\ldots = (\sum i.\ f\ (A\ i))$
using *A x*
by (*subst add*[*THEN additiveD*, *symmetric*])
(*auto intro*!: *arg-cong*[**where** *f*=*suminf*] *arg-cong*[**where** *f*=*f*])
finally show *outer-measure M f* $(s \cap x)$ + *outer-measure M f* $(s - x) \leq (\sum i.$ *f* $(A\ i))$ **.**

qed
moreover
have *outer-measure M f s* ≤ *outer-measure M f* (*s* ∩ *x*) + *outer-measure M f* (*s* − *x*)
proof −
have *outer-measure M f s* = *outer-measure M f* ((*s* ∩ *x*) ∪ (*s* − *x*))
by (*metis Un-Diff-Int Un-commute*)
also have ... ≤ *outer-measure M f* (*s* ∩ *x*) + *outer-measure M f* (*s* − *x*)
apply (*rule subadditiveD*)
apply (*rule ring-of-sets.countably-subadditive-subadditive* [*OF ring-of-sets-Pow*])
apply (*simp add*: *positive-def outer-measure-empty*[*OF posf*])
apply (*rule countably-subadditive-outer-measure*)
using *s* **by** (*auto intro*!: *posf inc*)
finally show *?thesis* **.**
qed
ultimately
show *outer-measure M f* (*s* ∩ *x*) + *outer-measure M f* (*s* − *x*) = *outer-measure M f s*
by (*rule order-antisym*)
qed

lemma *measure-down*: *measure-space* Ω *N* μ ⟹ *sigma-algebra* Ω *M* ⟹ *M* ⊆ *N* ⟹ *measure-space* Ω *M* μ
by (*auto simp add*: *measure-space-def positive-def countably-additive-def subset-eq*)

9.2 Caratheodory's theorem

theorem (**in** *ring-of-sets*) *caratheodory'*:
assumes *posf*: *positive M f* **and** *ca*: *countably-additive M f*
shows ∃ μ :: *'a set* ⇒ *ennreal*. (∀ *s* ∈ *M*. μ *s* = *f s*) ∧ *measure-space* Ω (*sigma-sets* Ω *M*) μ
proof −
have *inc*: *increasing M f*
by (*metis additive-increasing ca countably-additive-additive posf*)
let *?O* = *outer-measure M f*
def *ls* ≡ *lambda-system* Ω (*Pow* Ω) *?O*
have *mls*: *measure-space* Ω *ls ?O*
using *sigma-algebra.caratheodory-lemma*
[*OF sigma-algebra-Pow outer-measure-space-outer-measure* [*OF posf inc*]]
by (*simp add*: *ls-def*)
hence *sls*: *sigma-algebra* Ω *ls*
by (*simp add*: *measure-space-def*)
have *M* ⊆ *ls*
by (*simp add*: *ls-def*)
(*metis ca posf inc countably-additive-additive algebra-subset-lambda-system*)
hence *sgs-sb*: *sigma-sets* (Ω) (*M*) ⊆ *ls*
using *sigma-algebra.sigma-sets-subset* [*OF sls*, *of M*]
by *simp*
have *measure-space* Ω (*sigma-sets* Ω *M*) *?O*

by (*rule measure-down* [*OF mls*], *rule sigma-algebra-sigma-sets*)
(*simp-all add*: *sgs-sb space-closed*)
thus *?thesis* **using** *outer-measure-agrees* [*OF posf ca*]
by (*intro exI*[*of* - *?O*]) *auto*
qed

lemma (**in** *ring-of-sets*) *caratheodory-empty-continuous*:
assumes *f*: *positive M f additive M f* **and** *fin*: $\bigwedge A.\ A \in M \Longrightarrow f\ A \neq \infty$
assumes *cont*: $\bigwedge A.$ *range* $A \subseteq M \Longrightarrow$ *decseq* $A \Longrightarrow (\bigcap i.\ A\ i) = \{\} \Longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow 0$
shows $\exists \mu :: 'a\ set \Rightarrow ennreal.\ (\forall s \in M.\ \mu\ s = f\ s) \wedge$ *measure-space* Ω (*sigma-sets* Ω M) μ
proof (*intro caratheodory' empty-continuous-imp-countably-additive f*)
show $\forall A \in M.\ f\ A \neq \infty$ **using** *fin* **by** *auto*
qed (*rule cont*)

9.3 Volumes

definition *volume* :: $'a\ set\ set \Rightarrow ('a\ set \Rightarrow ennreal) \Rightarrow bool$ **where**
volume M f $\longleftrightarrow$
$(f\ \{\} = 0) \wedge (\forall a \in M.\ 0 \leq f\ a) \wedge$
$(\forall C \subseteq M.$ *disjoint* $C \longrightarrow$ *finite* $C \longrightarrow \bigcup C \in M \longrightarrow f\ (\bigcup C) = (\sum c \in C.\ f\ c))$

lemma *volumeI*:
assumes $f\ \{\} = 0$
assumes $\bigwedge a.\ a \in M \Longrightarrow 0 \leq f\ a$
assumes $\bigwedge C.\ C \subseteq M \Longrightarrow$ *disjoint* $C \Longrightarrow$ *finite* $C \Longrightarrow \bigcup C \in M \Longrightarrow f\ (\bigcup C) = (\sum c \in C.\ f\ c)$
shows *volume M f*
using *assms* **by** (*auto simp*: *volume-def*)

lemma *volume-positive*:
volume M f $\Longrightarrow a \in M \Longrightarrow 0 \leq f\ a$
by (*auto simp*: *volume-def*)

lemma *volume-empty*:
volume M f $\Longrightarrow f\ \{\} = 0$
by (*auto simp*: *volume-def*)

lemma *volume-finite-additive*:
assumes *volume M f*
assumes *A*: $\bigwedge i.\ i \in I \Longrightarrow A\ i \in M$ *disjoint-family-on A I finite I UNION I A* $\in M$
shows *f* (*UNION I A*) $= (\sum i \in I.\ f\ (A\ i))$
proof −
have *A*`*I* $\subseteq M$ *disjoint* (*A*`*I*) *finite* (*A*`*I*) $\bigcup$(*A*`*I*) $\in M$
using *A* **by** (*auto simp*: *disjoint-family-on-disjoint-image*)
with ‹*volume M f*› **have** *f* ($\bigcup$(*A*`*I*)) $= (\sum a \in A`I.\ f\ a)$
unfolding *volume-def* **by** *blast*

```
  also have ... = (∑ i∈I. f (A i))
  proof (subst setsum.reindex-nontrivial)
    fix i j assume i ∈ I j ∈ I i ≠ j A i = A j
    with ‹disjoint-family-on A I› have A i = {}
      by (auto simp: disjoint-family-on-def)
    then show f (A i) = 0
      using volume-empty[OF ‹volume M f›] by simp
  qed (auto intro: ‹finite I›)
  finally show f (UNION I A) = (∑ i∈I. f (A i))
    by simp
qed

lemma (in ring-of-sets) volume-additiveI:
  assumes pos: ⋀a. a ∈ M ⟹ 0 ≤ μ a
  assumes [simp]: μ {} = 0
  assumes add: ⋀a b. a ∈ M ⟹ b ∈ M ⟹ a ∩ b = {} ⟹ μ (a ∪ b) = μ a
+ μ b
  shows volume M μ
proof (unfold volume-def, safe)
  fix C assume finite C C ⊆ M disjoint C
  then show μ (⋃ C) = setsum μ C
  proof (induct C)
    case (insert c C)
    from insert(1,2,4,5) have μ (⋃ insert c C) = μ c + μ (⋃ C)
      by (auto intro!: add simp: disjoint-def)
    with insert show ?case
      by (simp add: disjoint-def)
  qed simp
qed fact+

lemma (in semiring-of-sets) extend-volume:
  assumes volume M μ
  shows ∃μ'. volume generated-ring μ' ∧ (∀ a∈M. μ' a = μ a)
proof −
  let ?R = generated-ring
  have ∀ a∈?R. ∃ m. ∃ C⊆M. a = ⋃ C ∧ finite C ∧ disjoint C ∧ m = (∑ c∈C.
μ c)
    by (auto simp: generated-ring-def)
  from bchoice[OF this] guess μ' .. note μ'-spec = this

  { fix C assume C: C ⊆ M finite C disjoint C
    fix D assume D: D ⊆ M finite D disjoint D
    assume ⋃ C = ⋃ D
    have (∑ d∈D. μ d) = (∑ d∈D. ∑ c∈C. μ (c ∩ d))
    proof (intro setsum.cong refl)
      fix d assume d ∈ D
      have Un-eq-d: (⋃ c∈C. c ∩ d) = d
        using ‹d ∈ D› ‹⋃ C = ⋃ D› by auto
      moreover have μ (⋃ c∈C. c ∩ d) = (∑ c∈C. μ (c ∩ d))
```

```
    proof (rule volume-finite-additive)
      { fix c assume c ∈ C then show c ∩ d ∈ M
          using C D ‹d ∈ D› by auto }
      show (⋃ a∈C. a ∩ d) ∈ M
        unfolding Un-eq-d using ‹d ∈ D› D by auto
      show disjoint-family-on (λa. a ∩ d) C
        using ‹disjoint C› by (auto simp: disjoint-family-on-def disjoint-def)
    qed fact+
    ultimately show μ d = (∑ c∈C. μ (c ∩ d)) by simp
  qed }
note split-sum = this

{ fix C assume C: C ⊆ M finite C disjoint C
  fix D assume D: D ⊆ M finite D disjoint D
  assume ⋃ C = ⋃ D
  with split-sum[OF C D] split-sum[OF D C]
  have (∑ d∈D. μ d) = (∑ c∈C. μ c)
    by (simp, subst setsum.commute, simp add: ac-simps) }
note sum-eq = this

{ fix C assume C: C ⊆ M finite C disjoint C
  then have ⋃ C ∈ ?R by (auto simp: generated-ring-def)
  with μ'-spec[THEN bspec, of ⋃ C]
  obtain D where
    D: D ⊆ M finite D disjoint D ⋃ C = ⋃ D and μ' (⋃ C) = (∑ d∈D. μ d)
    by auto
  with sum-eq[OF C D] have μ' (⋃ C) = (∑ c∈C. μ c) by simp }
note μ' = this

show ?thesis
proof (intro exI conjI ring-of-sets.volume-additiveI[OF generating-ring] ballI)
  fix a assume a ∈ M with μ'[of {a}] show μ' a = μ a
    by (simp add: disjoint-def)
next
  fix a assume a ∈ ?R then guess Ca .. note Ca = this
  with μ'[of Ca] ‹volume M μ›[THEN volume-positive]
  show 0 ≤ μ' a
    by (auto intro!: setsum-nonneg)
next
  show μ' {} = 0 using μ'[of {}] by auto
next
  fix a assume a ∈ ?R then guess Ca .. note Ca = this
  fix b assume b ∈ ?R then guess Cb .. note Cb = this
  assume a ∩ b = {}
  with Ca Cb have Ca ∩ Cb ⊆ {{}} by auto
  then have C-Int-cases: Ca ∩ Cb = {{}} ∨ Ca ∩ Cb = {} by auto

  from ‹a ∩ b = {}› have μ' (⋃ (Ca ∪ Cb)) = (∑ c∈Ca ∪ Cb. μ c)
    using Ca Cb by (intro μ') (auto intro!: disjoint-union)
```

```
    also have ... = (∑ c∈Ca ∪ Cb. μ c) + (∑ c∈Ca ∩ Cb. μ c)
      using C-Int-cases volume-empty[OF ‹volume M μ›] by (elim disjE) simp-all
    also have ... = (∑ c∈Ca. μ c) + (∑ c∈Cb. μ c)
      using Ca Cb by (simp add: setsum.union-inter)
    also have ... = μ' a + μ' b
      using Ca Cb by (simp add: μ')
    finally show μ' (a ∪ b) = μ' a + μ' b
      using Ca Cb by simp
  qed
qed
```

9.3.1 Caratheodory on semirings

```
theorem (in semiring-of-sets) caratheodory:
  assumes pos: positive M μ and ca: countably-additive M μ
   shows ∃μ' :: 'a set ⇒ ennreal. (∀s ∈ M. μ' s = μ s) ∧ measure-space Ω
(sigma-sets Ω M) μ'
proof −
  have volume M μ
  proof (rule volumeI)
    { fix a assume a ∈ M then show 0 ≤ μ a
        using pos unfolding positive-def by auto }
    note p = this

    fix C assume sets-C: C ⊆ M ⋃ C ∈ M and disjoint C finite C
    have ∃F'. bij-betw F' {..<card C} C
      by (rule finite-same-card-bij[OF - ‹finite C›]) auto
    then guess F' .. note F' = this
    then have F': C = F' ‘ {..< card C} inj-on F' {..< card C}
      by (auto simp: bij-betw-def)
    { fix i j assume *: i < card C j < card C i ≠ j
      with F' have F' i ∈ C F' j ∈ C F' i ≠ F' j
        unfolding inj-on-def by auto
      with ‹disjoint C›[THEN disjointD]
      have F' i ∩ F' j = {}
        by auto }
    note F'-disj = this
    def F ≡ λi. if i < card C then F' i else {}
    then have disjoint-family F
      using F'-disj by (auto simp: disjoint-family-on-def)
    moreover from F' have (⋃i. F i) = ⋃ C
      by (auto simp add: F-def split: if-split-asm) blast
    moreover have sets-F: ⋀i. F i ∈ M
      using F' sets-C by (auto simp: F-def)
    moreover note sets-C
    ultimately have μ (⋃ C) = (∑ i. μ (F i))
      using ca[unfolded countably-additive-def, THEN spec, of F] by auto
    also have ... = (∑ i<card C. μ (F' i))
    proof −
```

```
    have (λi. if i ∈ {..< card C} then μ (F' i) else 0) sums (∑ i<card C. μ (F' i))
      by (rule sums-If-finite-set) auto
    also have (λi. if i ∈ {..< card C} then μ (F' i) else 0) = (λi. μ (F i))
      using pos by (auto simp: positive-def F-def)
    finally show (∑ i. μ (F i)) = (∑ i<card C. μ (F' i))
      by (simp add: sums-iff)
  qed
  also have ... = (∑ c∈C. μ c)
    using F'(2) by (subst (2) F') (simp add: setsum.reindex)
  finally show μ (⋃ C) = (∑ c∈C. μ c) .
 next
  show μ {} = 0
    using ‹positive M μ› by (rule positiveD1)
 qed
 from extend-volume[OF this] obtain μ-r where
   V: volume generated-ring μ-r ⋀a. a ∈ M ⟹ μ a = μ-r a
   by auto

 interpret G: ring-of-sets Ω generated-ring
   by (rule generating-ring)

 have pos: positive generated-ring μ-r
  using V unfolding positive-def by (auto simp: positive-def intro!: volume-positive volume-empty)

 have countably-additive generated-ring μ-r
 proof (rule countably-additiveI)
  fix A' :: nat ⇒ 'a set assume A': range A' ⊆ generated-ring disjoint-family A'
    and Un-A: (⋃ i. A' i) ∈ generated-ring

   from generated-ringE[OF Un-A] guess C' . note C' = this

   { fix c assume c ∈ C'
     moreover def A ≡ λi. A' i ∩ c
     ultimately have A: range A ⊆ generated-ring disjoint-family A
       and Un-A: (⋃ i. A i) ∈ generated-ring
       using A' C'
         by (auto intro!: G.Int G.finite-Union intro: generated-ringI-Basic simp: disjoint-family-on-def)
     from A C' ‹c ∈ C'› have UN-eq: (⋃ i. A i) = c
       by (auto simp: A-def)

    have ∀ i::nat. ∃ f::nat ⇒ 'a set. μ-r (A i) = (∑ j. μ-r (f j)) ∧ disjoint-family f ∧ ⋃ range f = A i ∧ (∀ j. f j ∈ M)
       (is ∀ i. ?P i)
     proof
       fix i
       from A have Ai: A i ∈ generated-ring by auto
```

from *generated-ringE*[*OF this*] **guess** C **. note** $C = this$

have $\exists F'.$ *bij-betw* F' $\{..<card\ C\}$ C
by (*rule finite-same-card-bij*[*OF* - ‹*finite* C›]) *auto*
then guess F **.. note** $F = this$
def $f \equiv \lambda i.$ *if* $i <$ *card* C *then* $F\ i$ *else* $\{\}$
then have f: *bij-betw* f $\{..<$ *card* $C\}$ C
by (*intro bij-betw-cong*[*THEN iffD1*, *OF* - F]) *auto*
with C **have** $\forall j.\ f\ j \in M$
by (*auto simp*: *Pi-iff f-def dest*!: *bij-betw-imp-funcset*)
moreover
from f C **have** *d-f*: *disjoint-family-on* f $\{..<card\ C\}$
by (*intro disjoint-image-disjoint-family-on*) (*auto simp*: *bij-betw-def*)
then have *disjoint-family* f
by (*auto simp*: *disjoint-family-on-def f-def*)
moreover
have *Ai-eq*: $A\ i = (\bigcup x{<}card\ C.\ f\ x)$
using f C *Ai* **unfolding** *bij-betw-def* **by** *auto*
then have $\bigcup range\ f = A\ i$
using f C *Ai* **unfolding** *bij-betw-def*
by (*auto simp add*: *f-def cong del*: *strong-SUP-cong*)
moreover
{ have $(\sum j.\ \mu\text{-}r\ (f\ j)) = (\sum j.\ if\ j \in \{..<\ card\ C\}\ then\ \mu\text{-}r\ (f\ j)\ else\ 0)$
using *volume-empty*[*OF* $V(1)$] **by** (*auto intro*!: *arg-cong*[**where** $f{=}suminf$] *simp*: *f-def*)
also have $\ldots = (\sum j{<}card\ C.\ \mu\text{-}r\ (f\ j))$
by (*rule sums-If-finite-set*[*THEN sums-unique*, *symmetric*]) *simp*
also have $\ldots = \mu\text{-}r\ (A\ i)$
using C f[*THEN bij-betw-imp-funcset*] **unfolding** *Ai-eq*
by (*intro volume-finite-additive*[*OF* $V(1)$ - *d-f*, *symmetric*])
(*auto simp*: *Pi-iff Ai-eq intro*: *generated-ringI-Basic*)
finally have $\mu\text{-}r\ (A\ i) = (\sum j.\ \mu\text{-}r\ (f\ j))$ **.. }**
ultimately show *?P i*
by *blast*
qed
from *choice*[*OF this*] **guess** f **.. note** $f = this$
then have *UN-f-eq*: $(\bigcup i.\ case\text{-}prod\ f\ (prod\text{-}decode\ i)) = (\bigcup i.\ A\ i)$
unfolding *UN-extend-simps surj-prod-decode* **by** (*auto simp*: *set-eq-iff*)

have *d*: *disjoint-family* $(\lambda i.\ case\text{-}prod\ f\ (prod\text{-}decode\ i))$
unfolding *disjoint-family-on-def*
proof (*intro ballI impI*)
fix $m\ n$ :: *nat* **assume** $m \neq n$
then have *neq*: *prod-decode* $m \neq$ *prod-decode* n
using *inj-prod-decode*[*of UNIV*] **by** (*auto simp*: *inj-on-def*)
show *case-prod* f (*prod-decode* m) $\cap$ *case-prod* f (*prod-decode* n) $= \{\}$
proof *cases*
assume *fst* (*prod-decode* m) = *fst* (*prod-decode* n)
then show *?thesis*

```
        using neq f by (fastforce simp: disjoint-family-on-def)
      next
        assume neq: fst (prod-decode m) ≠ fst (prod-decode n)
        have case-prod f (prod-decode m) ⊆ A (fst (prod-decode m))
          case-prod f (prod-decode n) ⊆ A (fst (prod-decode n))
          using f[THEN spec, of fst (prod-decode m)]
          using f[THEN spec, of fst (prod-decode n)]
          by (auto simp: set-eq-iff)
        with f A neq show ?thesis
          by (fastforce simp: disjoint-family-on-def subset-eq set-eq-iff)
      qed
    qed
    from f have (∑ n. μ-r (A n)) = (∑ n. μ-r (case-prod f (prod-decode n)))
      by (intro suminf-ennreal-2dimen[symmetric] generated-ringI-Basic)
       (auto split: prod.split)
    also have ... = (∑ n. μ (case-prod f (prod-decode n)))
      using f V(2) by (auto intro!: arg-cong[where f=suminf] split: prod.split)
    also have ... = μ (⋃ i. case-prod f (prod-decode i))
      using f ‹c ∈ C'› C'
      by (intro ca[unfolded countably-additive-def, rule-format])
         (auto split: prod.split simp: UN-f-eq d UN-eq)
    finally have (∑ n. μ-r (A' n ∩ c)) = μ c
      using UN-f-eq UN-eq by (simp add: A-def) }
  note eq = this

  have (∑ n. μ-r (A' n)) = (∑ n. ∑ c∈C'. μ-r (A' n ∩ c))
    using C' A'
    by (subst volume-finite-additive[symmetric, OF V(1)])
       (auto simp: disjoint-def disjoint-family-on-def
             intro!: G.Int G.finite-Union arg-cong[where f=λX. suminf (λi. μ-r
(X i))] ext
             intro: generated-ringI-Basic)
  also have ... = (∑ c∈C'. ∑ n. μ-r (A' n ∩ c))
    using C' A'
   by (intro suminf-setsum G.Int G.finite-Union) (auto intro: generated-ringI-Basic)
  also have ... = (∑ c∈C'. μ-r c)
    using eq V C' by (auto intro!: setsum.cong)
  also have ... = μ-r (⋃ C')
    using C' Un-A
    by (subst volume-finite-additive[symmetric, OF V(1)])
       (auto simp: disjoint-family-on-def disjoint-def
             intro: generated-ringI-Basic)
  finally show (∑ n. μ-r (A' n)) = μ-r (⋃ i. A' i)
    using C' by simp
 qed
 from G.caratheodory'[OF ‹positive generated-ring μ-r› ‹countably-additive generated-ring
μ-r›]
 guess μ' ..
 with V show ?thesis
```

unfolding *sigma-sets-generated-ring-eq*
by (*intro exI*[*of* - μ']) (*auto intro*: *generated-ringI-Basic*)
qed

lemma *extend-measure-caratheodory*:
fixes $G :: 'i \Rightarrow 'a\ set$
assumes M: $M = extend\text{-}measure\ \Omega\ I\ G\ \mu$
assumes $i \in I$
assumes *semiring-of-sets* Ω (G ' I)
assumes *empty*: $\bigwedge i.\ i \in I \Longrightarrow G\ i = \{\} \Longrightarrow \mu\ i = 0$
assumes *inj*: $\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow G\ i = G\ j \Longrightarrow \mu\ i = \mu\ j$
assumes *nonneg*: $\bigwedge i.\ i \in I \Longrightarrow 0 \leq \mu\ i$
assumes *add*: $\bigwedge A{::}nat \Rightarrow 'i.\ \bigwedge j.\ A \in UNIV \rightarrow I \Longrightarrow j \in I \Longrightarrow$ *disjoint-family* ($G \circ A$) $\Longrightarrow$
($\bigcup i.\ G\ (A\ i)) = G\ j \Longrightarrow (\sum n.\ \mu\ (A\ n)) = \mu\ j$
shows *emeasure* M ($G\ i$) $= \mu\ i$
proof −
interpret *semiring-of-sets* Ω G ' I
by *fact*
have $\forall g{\in}G`I.\ \exists i{\in}I.\ g = G\ i$
by *auto*
then obtain *sel* **where** *sel*: $\bigwedge g.\ g \in G$ ' $I \Longrightarrow sel\ g \in I\ \bigwedge g.\ g \in G$ ' $I \Longrightarrow G$ ($sel\ g$) $= g$
by *metis*

have $\exists \mu'.\ (\forall s{\in}G$ ' $I.\ \mu'\ s = \mu\ (sel\ s)) \wedge measure\text{-}space\ \Omega$ (*sigma-sets* Ω (G ' I)) μ'
proof (*rule caratheodory*)
show *positive* (G ' I) ($\lambda s.\ \mu\ (sel\ s)$)
by (*auto simp*: *positive-def intro*!: *empty sel nonneg*)
show *countably-additive* (G ' I) ($\lambda s.\ \mu\ (sel\ s)$)
proof (*rule countably-additiveI*)
fix $A :: nat \Rightarrow 'a\ set$ **assume** *range* $A \subseteq G$ ' I *disjoint-family* A ($\bigcup i.\ A\ i) \in G$ ' I
then show ($\sum i.\ \mu\ (sel\ (A\ i))) = \mu\ (sel\ (\bigcup i.\ A\ i))$
by (*intro add*) (*auto simp*: *sel image-subset-iff-funcset comp-def Pi-iff intro*!: *sel*)
qed
qed
then obtain μ' **where** μ': $\forall s{\in}G$ ' $I.\ \mu'\ s = \mu\ (sel\ s)$ *measure-space* Ω (*sigma-sets* Ω (G ' I)) μ'
by *metis*

show *?thesis*
proof (*rule emeasure-extend-measure*[*OF M*])
{ **fix** i **assume** $i \in I$ **then show** $\mu'\ (G\ i) = \mu\ i$
using μ' **by** (*auto intro*!: *inj sel*) }
show G ' $I \subseteq Pow\ \Omega$
by *fact*

then show *positive* (*sets M*) μ' *countably-additive* (*sets M*) μ'
using μ' **by** (*simp-all add*: *M sets-extend-measure measure-space-def*)
qed *fact*
qed

lemma *extend-measure-caratheodory-pair*:
fixes $G :: 'i \Rightarrow 'j \Rightarrow 'a$ *set*
assumes *M*: M = *extend-measure* Ω $\{(a, b).\ P\ a\ b\}$ $(\lambda(a, b).\ G\ a\ b)$ $(\lambda(a, b).\ \mu\ a\ b)$
assumes $P\ i\ j$
assumes *semiring*: *semiring-of-sets* Ω $\{G\ a\ b \mid a\ b.\ P\ a\ b\}$
assumes *empty*: $\bigwedge i\ j.\ P\ i\ j \Longrightarrow G\ i\ j = \{\} \Longrightarrow \mu\ i\ j = 0$
assumes *inj*: $\bigwedge i\ j\ k\ l.\ P\ i\ j \Longrightarrow P\ k\ l \Longrightarrow G\ i\ j = G\ k\ l \Longrightarrow \mu\ i\ j = \mu\ k\ l$
assumes *nonneg*: $\bigwedge i\ j.\ P\ i\ j \Longrightarrow 0 \leq \mu\ i\ j$
assumes *add*: $\bigwedge A{::}nat \Rightarrow 'i.\ \bigwedge B{::}nat \Rightarrow 'j.\ \bigwedge j\ k.$
$(\bigwedge n.\ P\ (A\ n)\ (B\ n)) \Longrightarrow P\ j\ k \Longrightarrow$ *disjoint-family* $(\lambda n.\ G\ (A\ n)\ (B\ n)) \Longrightarrow$
$(\bigcup i.\ G\ (A\ i)\ (B\ i)) = G\ j\ k \Longrightarrow (\sum n.\ \mu\ (A\ n)\ (B\ n)) = \mu\ j\ k$
shows *emeasure M* $(G\ i\ j) = \mu\ i\ j$
proof −
have *emeasure M* $((\lambda(a, b).\ G\ a\ b)\ (i, j)) = (\lambda(a, b).\ \mu\ a\ b)\ (i, j)$
proof (*rule extend-measure-caratheodory*[*OF M*])
show *semiring-of-sets* Ω $((\lambda(a, b).\ G\ a\ b)$ ' $\{(a, b).\ P\ a\ b\})$
using *semiring* **by** (*simp add*: *image-def conj-commute*)
next
fix $A :: nat \Rightarrow ('i \times 'j)$ **and** j **assume** $A \in UNIV \rightarrow \{(a, b).\ P\ a\ b\}$ $j \in \{(a, b).\ P\ a\ b\}$
disjoint-family $((\lambda(a, b).\ G\ a\ b) \circ A)$
$(\bigcup i.$ *case A i of* $(a, b) \Rightarrow G\ a\ b) = ($*case j of* $(a, b) \Rightarrow G\ a\ b)$
then show $(\sum n.$ *case A n of* $(a, b) \Rightarrow \mu\ a\ b) = ($*case j of* $(a, b) \Rightarrow \mu\ a\ b)$
using *add*[*of* $\lambda i.$ *fst* $(A\ i)$ $\lambda i.$ *snd* $(A\ i)$ *fst j snd j*]
by (*simp add*: *split-beta' comp-def Pi-iff*)
qed (*auto split*: *prod.splits intro*: *assms*)
then show *?thesis* **by** *simp*
qed

end

10 Lebesgue measure

theory *Lebesgue-Measure*
imports *Finite-Product-Measure Bochner-Integration Caratheodory*
begin

10.1 Every right continuous and nondecreasing function gives rise to a measure

definition *interval-measure* :: $(real \Rightarrow real) \Rightarrow real\ measure$ **where**
interval-measure F = *extend-measure UNIV* $\{(a, b).\ a \leq b\}$ $(\lambda(a, b).\ \{a <..\ b\})$ $(\lambda(a, b).\ ennreal\ (F\ b - F\ a))$

lemma *emeasure-interval-measure-Ioc*:
assumes $a \leq b$
assumes *mono-F*: $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$
assumes *right-cont-F* : $\bigwedge a.$ *continuous* (*at-right* a) F
shows *emeasure* (*interval-measure* F) $\{a <..\ b\} = F\ b - F\ a$
proof (*rule extend-measure-caratheodory-pair*[*OF interval-measure-def* ‹$a \leq b$›])
show *semiring-of-sets UNIV* $\{\{a<..b\}\ |a\ b ::$ *real*. $a \leq b\}$
proof (*unfold-locales*, *safe*)
fix $a\ b\ c\ d ::$ *real* **assume** $*$: $a \leq b\ c \leq d$
then show $\exists\, C \subseteq \{\{a<..b\}\ |a\ b.\ a \leq b\}.$ *finite* $C \wedge$ *disjoint* $C \wedge \{a<..b\} - \{c<..d\} = \bigcup C$
proof *cases*
let *?C* $= \{\{a<..b\}\}$
assume $b < c \vee d \leq a \vee d \leq c$
with $*$ **have** *?C* $\subseteq \{\{a<..b\}\ |a\ b.\ a \leq b\} \wedge$ *finite ?C* $\wedge$ *disjoint ?C* $\wedge \{a<..b\} - \{c<..d\} = \bigcup$ *?C*
by (*auto simp add*: *disjoint-def*)
thus *?thesis* **..**
next
let *?C* $= \{\{a<..c\}, \{d<..b\}\}$
assume $\neg\ (b < c \vee d \leq a \vee d \leq c)$
with $*$ **have** *?C* $\subseteq \{\{a<..b\}\ |a\ b.\ a \leq b\} \wedge$ *finite ?C* $\wedge$ *disjoint ?C* $\wedge \{a<..b\} - \{c<..d\} = \bigcup$ *?C*
by (*auto simp add*: *disjoint-def Ioc-inj*) (*metis linear*)+
thus *?thesis* **..**
qed
qed (*auto simp*: *Ioc-inj*, *metis linear*)
next
fix $l\ r ::$ *nat* $\Rightarrow$ *real* **and** $a\ b ::$ *real*
assume *l-r*[*simp*]: $\bigwedge n.\ l\ n \leq r\ n$ **and** $a \leq b$ **and** *disj*: *disjoint-family* $(\lambda n.\ \{l\ n<..r\ n\})$
assume *lr-eq-ab*: $(\bigcup i.\ \{l\ i<..r\ i\}) = \{a<..b\}$

have [*intro*, *simp*]: $\bigwedge a\ b.\ a \leq b \Longrightarrow F\ a \leq F\ b$
by (*auto intro*!: *l-r mono-F*)

{ fix $S ::$ *nat set* **assume** *finite S*
moreover note ‹$a \leq b$›
moreover have $\bigwedge i.\ i \in S \Longrightarrow \{l\ i <..\ r\ i\} \subseteq \{a <..\ b\}$
unfolding *lr-eq-ab*[*symmetric*] **by** *auto*
ultimately have $(\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) \leq F\ b - F\ a$
proof (*induction S arbitrary*: *a rule*: *finite-psubset-induct*)
case (*psubset S*)
show *?case*
proof *cases*
assume $\exists\, i{\in}S.\ l\ i < r\ i$
with ‹*finite S*› **have** *Min* $(l\ `\ \{i{\in}S.\ l\ i < r\ i\}) \in l\ `\ \{i{\in}S.\ l\ i < r\ i\}$
by (*intro Min-in*) *auto*

```
      then obtain m where m: m ∈ S l m < r m l m = Min (l ‘ {i∈S. l i < r
i})
        by fastforce

      have (∑ i∈S. F (r i) − F (l i)) = (F (r m) − F (l m)) + (∑ i∈S − {m}.
F (r i) − F (l i))
        using m psubset by (intro setsum.remove) auto
       also have (∑ i∈S − {m}. F (r i) − F (l i)) ≤ F b − F (r m)
       proof (intro psubset.IH)
        show S − {m} ⊂ S
          using ‹m∈S› by auto
        show r m ≤ b
          using psubset.prems(2)[OF ‹m∈S›] ‹l m < r m› by auto
       next
        fix i assume i ∈ S − {m}
        then have i: i ∈ S i ≠ m by auto
        { assume i': l i < r i l i < r m
          moreover with ‹finite S› i m have l m ≤ l i
            by auto
          ultimately have {l i <.. r i} ∩ {l m <.. r m} ≠ {}
            by auto
          then have False
            using disjoint-family-onD[OF disj, of i m] i by auto }
        then have l i ≠ r i ⟹ r m ≤ l i
          unfolding not-less[symmetric] using l-r[of i] by auto
        then show {l i <.. r i} ⊆ {r m <.. b}
          using psubset.prems(2)[OF ‹i∈S›] by auto
       qed
       also have F (r m) − F (l m) ≤ F (r m) − F a
        using psubset.prems(2)[OF ‹m ∈ S›] ‹l m < r m›
        by (auto simp add: Ioc-subset-iff intro!: mono-F)
       finally show ?case
        by (auto intro: add-mono)
     qed (auto simp add: ‹a ≤ b› less-le)
   qed }
  note claim1 = this


  { fix S u v and l r :: nat ⇒ real
   assume finite S ⋀i. i∈S ⟹ l i < r i {u..v} ⊆ (⋃ i∈S. {l i<..< r i})
   then have F v − F u ≤ (∑ i∈S. F (r i) − F (l i))
   proof (induction arbitrary: v u rule: finite-psubset-induct)
     case (psubset S)
     show ?case
     proof cases
       assume S = {} then show ?case
         using psubset by (simp add: mono-F)
     next
```

assume $S \neq \{\}$
then obtain j **where** $j \in S$
by *auto*

let *?R* $= r\ j < u \vee l\ j > v \vee (\exists\, i{\in}S{-}\{j\}.\ l\ i \leq l\ j \wedge r\ j \leq r\ i)$
show *?case*
proof *cases*
assume *?R*
with ‹$j \in S$› *psubset.prems* **have** $\{u..v\} \subseteq (\bigcup i{\in}S{-}\{j\}.\ \{l\ i{<}..{<}\ r\ i\})$
apply (*auto simp*: *subset-eq Ball-def*)
apply (*metis Diff-iff less-le-trans leD linear singletonD*)
apply (*metis Diff-iff less-le-trans leD linear singletonD*)
apply (*metis order-trans less-le-not-le linear*)
done
with ‹$j \in S$› **have** $F\ v - F\ u \leq (\sum i{\in}S - \{j\}.\ F\ (r\ i) - F\ (l\ i))$
by (*intro psubset*) *auto*
also have $\ldots \leq (\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i))$
using *psubset.prems*
by (*intro setsum-mono2 psubset*) (*auto intro*: *less-imp-le*)
finally show *?thesis* **.**
next
assume $\neg$ *?R*
then have *j*: $u \leq r\ j\ l\ j \leq v\ \bigwedge i.\ i \in S - \{j\} \Longrightarrow r\ i < r\ j \vee l\ i > l\ j$
by (*auto simp*: *not-less*)
let *?S1* $= \{i \in S.\ l\ i < l\ j\}$
let *?S2* $= \{i \in S.\ r\ i > r\ j\}$

have $(\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) \geq (\sum i{\in}$*?S1* $\cup$ *?S2* $\cup \{j\}.\ F\ (r\ i) - F\ (l\ i))$
using ‹$j \in S$› ‹*finite S*› *psubset.prems j*
by (*intro setsum-mono2*) (*auto intro*: *less-imp-le*)
also have $(\sum i{\in}$*?S1* $\cup$ *?S2* $\cup \{j\}.\ F\ (r\ i) - F\ (l\ i)) =$
$(\sum i{\in}$*?S1*$.\ F\ (r\ i) - F\ (l\ i)) + (\sum i{\in}$*?S2* $.\ F\ (r\ i) - F\ (l\ i)) + (F\ (r\ j) - F\ (l\ j))$
using *psubset*(*1*) *psubset.prems*(*1*) *j*
apply (*subst setsum.union-disjoint*)
apply *simp-all*
apply (*subst setsum.union-disjoint*)
apply *auto*
apply (*metis less-le-not-le*)
done
also (*xtrans*) **have** $(\sum i{\in}$*?S1*$.\ F\ (r\ i) - F\ (l\ i)) \geq F\ (l\ j) - F\ u$
using ‹$j \in S$› ‹*finite S*› *psubset.prems j*
apply (*intro psubset.IH psubset*)
apply (*auto simp*: *subset-eq Ball-def*)
apply (*metis less-le-trans not-le*)
done
also (*xtrans*) **have** $(\sum i{\in}$*?S2*$.\ F\ (r\ i) - F\ (l\ i)) \geq F\ v - F\ (r\ j)$
using ‹$j \in S$› ‹*finite S*› *psubset.prems j*

```
        apply (intro psubset.IH psubset)
        apply (auto simp: subset-eq Ball-def)
        apply (metis le-less-trans not-le)
        done
      finally (xtrans) show ?case
        by (auto simp: add-mono)
    qed
  qed
 qed }
note claim2 = this


have ennreal (F b - F a) ≤ (∑ i. ennreal (F (r i) - F (l i)))
proof (rule ennreal-le-epsilon)
 fix epsilon :: real assume egt0: epsilon > 0
 have ∀ i. ∃ d>0. F (r i + d) < F (r i) + epsilon / 2^(i+2)
 proof
  fix i
  note right-cont-F [of r i]
  thus ∃ d>0. F (r i + d) < F (r i) + epsilon / 2^(i+2)
   apply -
   apply (subst (asm) continuous-at-right-real-increasing)
   apply (rule mono-F, assumption)
   apply (drule-tac x = epsilon / 2 ^ (i + 2) in spec)
   apply (erule impE)
   using egt0 by (auto simp add: field-simps)
 qed
 then obtain delta where
    deltai-gt0: ⋀i. delta i > 0 and
    deltai-prop: ⋀i. F (r i + delta i) < F (r i) + epsilon / 2^(i+2)
  by metis
 have ∃ a' > a. F a' - F a < epsilon / 2
  apply (insert right-cont-F [of a])
  apply (subst (asm) continuous-at-right-real-increasing)
  using mono-F apply force
  apply (drule-tac x = epsilon / 2 in spec)
  using egt0 unfolding mult.commute [of 2] by force
 then obtain a' where a'lea [arith]: a' > a and
  a-prop: F a' - F a < epsilon / 2
  by auto
 def S' ≡ {i. l i < r i}
 obtain S :: nat set where
  S ⊆ S' and finS: finite S and
  Sprop: {a'..b} ⊆ (⋃ i ∈ S. {l i<..<r i + delta i})
 proof (rule compactE-image)
  show compact {a'..b}
   by (rule compact-Icc)
  show ∀ i ∈ S'. open ({l i<..<r i + delta i}) by auto
  have {a'..b} ⊆ {a <.. b}
```

by *auto*
also have $\{a <.. b\} = (\bigcup i{\in}S'.\ \{l\ i{<}..r\ i\})$
unfolding *lr-eq-ab*[*symmetric*] **by** (*fastforce simp add*: *S'-def intro*: *less-le-trans*)
also have $\ldots \subseteq (\bigcup i \in S'.\ \{l\ i{<}..{<}r\ i + delta\ i\})$
apply (*intro UN-mono*)
apply (*auto simp*: *S'-def*)
apply (*cut-tac i=i* **in** *deltai-gt0*)
apply *simp*
done
finally show $\{a'..b\} \subseteq (\bigcup i \in S'.\ \{l\ i{<}..{<}r\ i + delta\ i\})$ **.**
qed
with *S'-def* **have** *Sprop2*: $\bigwedge i.\ i \in S \Longrightarrow l\ i < r\ i$ **by** *auto*
from *finS* **have** $\exists n.\ \forall i \in S.\ i \leq n$
by (*subst finite-nat-set-iff-bounded-le* [*symmetric*])
then obtain *n* **where** *Sbound* [*rule-format*]: $\forall i \in S.\ i \leq n$ **..**
have $F\ b - F\ a' \leq (\sum i{\in}S.\ F\ (r\ i + delta\ i) - F\ (l\ i))$
apply (*rule claim2* [*rule-format*])
using *finS Sprop* **apply** *auto*
apply (*frule Sprop2*)
apply (*subgoal-tac delta i > 0*)
apply *arith*
by (*rule deltai-gt0*)
also have ... $\leq (\sum i \in S.\ F(r\ i) - F(l\ i) + epsilon\ /\ 2\hat{}(i+2))$
apply (*rule setsum-mono*)
apply *simp*
apply (*rule order-trans*)
apply (*rule less-imp-le*)
apply (*rule deltai-prop*)
by *auto*
also have ... $= (\sum i \in S.\ F(r\ i) - F(l\ i)) +$
$(epsilon\ /\ 4) * (\sum i \in S.\ (1\ /\ 2)\hat{}i)$ (**is** - = *?t* + -)
by (*subst setsum.distrib*) (*simp add*: *field-simps setsum-right-distrib*)
also have ... $\leq$ *?t* + $(epsilon\ /\ 4) * (\sum\ i < Suc\ n.\ (1\ /\ 2)\hat{}i)$
apply (*rule add-left-mono*)
apply (*rule mult-left-mono*)
apply (*rule setsum-mono2*)
using *egt0* **apply** *auto*
by (*frule Sbound*, *auto*)
also have ... $\leq$ *?t* + $(epsilon\ /\ 2)$
apply (*rule add-left-mono*)
apply (*subst geometric-sum*)
apply *auto*
apply (*rule mult-left-mono*)
using *egt0* **apply** *auto*
done
finally have *aux2*: $F\ b - F\ a' \leq (\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) + epsilon\ /\ 2$
by *simp*

have $F\ b - F\ a = (F\ b - F\ a') + (F\ a' - F\ a)$

by *auto*
also have ... $\leq (F\ b - F\ a') + \mathit{epsilon} / 2$
using *a-prop* **by** (*intro add-left-mono*) *simp*
also have ... $\leq (\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) + \mathit{epsilon} / 2 + \mathit{epsilon} / 2$
apply (*intro add-right-mono*)
apply (*rule aux2*)
done
also have ... $= (\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) + \mathit{epsilon}$
by *auto*
also have ... $\leq (\sum i{\leq}n.\ F\ (r\ i) - F\ (l\ i)) + \mathit{epsilon}$
using *finS Sbound Sprop* **by** (*auto intro*!: *add-right-mono setsum-mono3*)
finally have *ennreal* $(F\ b - F\ a) \leq (\sum i{\leq}n.\ \mathit{ennreal}\ (F\ (r\ i) - F\ (l\ i))) +$ *epsilon*
using *egt0* **by** (*simp add*: *ennreal-plus*[*symmetric*] *setsum-nonneg del*: *ennreal-plus*)
then show *ennreal* $(F\ b - F\ a) \leq (\sum i.\ \mathit{ennreal}\ (F\ (r\ i) - F\ (l\ i))) + (\mathit{epsilon}$ :: *real*)
by (*rule order-trans*) (*auto intro*!: *add-mono setsum-le-suminf simp del*: *setsum-ennreal*)
qed
moreover have $(\sum i.\ \mathit{ennreal}\ (F\ (r\ i) - F\ (l\ i))) \leq \mathit{ennreal}\ (F\ b - F\ a)$
using ‹$a \leq b$› **by** (*auto intro*!: *suminf-le-const ennreal-le-iff*[*THEN iffD2*] *claim1*)
ultimately show $(\sum n.\ \mathit{ennreal}\ (F\ (r\ n) - F\ (l\ n))) = \mathit{ennreal}\ (F\ b - F\ a)$
by (*rule antisym*[*rotated*])
qed (*auto simp*: *Ioc-inj mono-F*)

lemma *measure-interval-measure-Ioc*:
assumes $a \leq b$
assumes *mono-F*: $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$
assumes *right-cont-F* : $\bigwedge a.\ \mathit{continuous}\ (\mathit{at\text{-}right}\ a)\ F$
shows *measure* (*interval-measure F*) $\{a <..\ b\} = F\ b - F\ a$
unfolding *measure-def*
apply (*subst emeasure-interval-measure-Ioc*)
apply *fact*+
apply (*simp add*: *assms*)
done

lemma *emeasure-interval-measure-Ioc-eq*:
$(\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y) \Longrightarrow (\bigwedge a.\ \mathit{continuous}\ (\mathit{at\text{-}right}\ a)\ F) \Longrightarrow$
emeasure (*interval-measure F*) $\{a <..\ b\} = (\mathit{if}\ a \leq b\ \mathit{then}\ F\ b - F\ a\ \mathit{else}\ 0)$
using *emeasure-interval-measure-Ioc*[*of a b F*] **by** *auto*

lemma *sets-interval-measure* [*simp*, *measurable-cong*]: *sets* (*interval-measure F*) = *sets borel*
apply (*simp add*: *sets-extend-measure interval-measure-def borel-sigma-sets-Ioc*)
apply (*rule sigma-sets-eqI*)
apply *auto*
apply (*case-tac* $a \leq ba$)
apply (*auto intro*: *sigma-sets.Empty*)

done

lemma *space-interval-measure* [*simp*]: *space* (*interval-measure F*) = *UNIV*
by (*simp add*: *interval-measure-def space-extend-measure*)

lemma *emeasure-interval-measure-Icc*:
assumes $a \leq b$
assumes *mono-F*: $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$
assumes *cont-F* : *continuous-on UNIV F*
shows *emeasure* (*interval-measure F*) $\{a\ ..\ b\}$ = $F\ b - F\ a$
proof (*rule tendsto-unique*)
{ fix *a b* :: *real* **assume** $a \leq b$ **then have** *emeasure* (*interval-measure F*) $\{a <..$
$b\}$ = $F\ b - F\ a$
using *cont-F*
by (*subst emeasure-interval-measure-Ioc*)
(*auto intro*: *mono-F continuous-within-subset simp*: *continuous-on-eq-continuous-within*)
}
note $* = this$

let *?F* = *interval-measure F*
show $((\lambda a.\ F\ b - F\ a) \longrightarrow$ *emeasure ?F* $\{a..b\})$ (*at-left a*)
proof (*rule tendsto-at-left-sequentially*)
show $a - 1 < a$ **by** *simp*
fix *X* **assume** $\bigwedge n.\ X\ n < a$ *incseq X* $X \longrightarrow a$
with ‹$a \leq b$› **have** $(\lambda n.$ *emeasure ?F* $\{X\ n<..b\}) \longrightarrow$ *emeasure ?F* $(\bigcap n.$
$\{X\ n <..b\})$
apply (*intro Lim-emeasure-decseq*)
apply (*auto simp*: *decseq-def incseq-def emeasure-interval-measure-Ioc* $*$)
apply *force*
apply (*subst* (*asm*) $*$)
apply (*auto intro*: *less-le-trans less-imp-le*)
done
also have $(\bigcap n.\ \{X\ n <..b\}) = \{a..b\}$
using ‹$\bigwedge n.\ X\ n < a$›
apply *auto*
apply (*rule LIMSEQ-le-const2*[*OF* ‹$X \longrightarrow a$›])
apply (*auto intro*: *less-imp-le*)
apply (*auto intro*: *less-le-trans*)
done
also have $(\lambda n.$ *emeasure ?F* $\{X\ n<..b\}) = (\lambda n.\ F\ b - F\ (X\ n))$
using ‹$\bigwedge n.\ X\ n < a$› ‹$a \leq b$› **by** (*subst* $*$) (*auto intro*: *less-imp-le less-le-trans*)
finally show $(\lambda n.\ F\ b - F\ (X\ n)) \longrightarrow$ *emeasure ?F* $\{a..b\}$ **.**
qed
show $((\lambda a.$ *ennreal* $(F\ b - F\ a))$ $\longrightarrow F\ b - F\ a)$ (*at-left a*)
by (*rule continuous-on-tendsto-compose*[**where** $g{=}\lambda x.\ x$ **and** *s*=*UNIV*])
(*auto simp*: *continuous-on-ennreal continuous-on-diff cont-F continuous-on-const*)
qed (*rule trivial-limit-at-left-real*)

lemma *sigma-finite-interval-measure*:

assumes *mono-F*: $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$
assumes *right-cont-F* : $\bigwedge a.$ *continuous* (*at-right a*) *F*
shows *sigma-finite-measure* (*interval-measure F*)
apply *unfold-locales*
apply (*intro exI*[*of* - ($\lambda(a,\ b).$ {*a* <.. *b*}) ‘ ($\mathbb{Q} \times \mathbb{Q}$)])
apply (*auto intro*!: *Rats-no-top-le Rats-no-bot-less countable-rat simp*: *emeasure-interval-measure-Ioc-eq*[*OF assms*])
done

10.2 Lebesgue-Borel measure

definition *lborel* :: ('*a* :: *euclidean-space*) *measure* **where**
lborel = *distr* (Π_M *b*∈*Basis*. *interval-measure* ($\lambda x.\ x$)) *borel* ($\lambda f.\ \sum b{\in}Basis.\ f\ b *_R b$)

lemma
shows *sets-lborel*[*simp*, *measurable-cong*]: *sets lborel* = *sets borel*
and *space-lborel*[*simp*]: *space lborel* = *space borel*
and *measurable-lborel1*[*simp*]: *measurable M lborel* = *measurable M borel*
and *measurable-lborel2*[*simp*]: *measurable lborel M* = *measurable borel M*
by (*simp-all add*: *lborel-def*)

context
begin

interpretation *sigma-finite-measure interval-measure* ($\lambda x.\ x$)
by (*rule sigma-finite-interval-measure*) *auto*
interpretation *finite-product-sigma-finite* λ-. *interval-measure* ($\lambda x.\ x$) *Basis*
proof qed *simp*

lemma *lborel-eq-real*: *lborel* = *interval-measure* ($\lambda x.\ x$)
unfolding *lborel-def Basis-real-def*
using *distr-id*[*of interval-measure* ($\lambda x.\ x$)]
by (*subst distr-component*[*symmetric*])
(*simp-all add*: *distr-distr comp-def del*: *distr-id cong*: *distr-cong*)

lemma *lborel-eq*: *lborel* = *distr* (Π_M *b*∈*Basis*. *lborel*) *borel* ($\lambda f.\ \sum b{\in}Basis.\ f\ b *_R b$)
by (*subst lborel-def*) (*simp add*: *lborel-eq-real*)

lemma *nn-integral-lborel-setprod*:
assumes [*measurable*]: $\bigwedge b.\ b \in Basis \Longrightarrow f\ b \in$ *borel-measurable borel*
assumes *nn*[*simp*]: $\bigwedge b\ x.\ b \in Basis \Longrightarrow 0 \leq f\ b\ x$
shows ($\int^+ x.\ (\prod b{\in}Basis.\ f\ b\ (x \cdot b))\ \partial lborel$) = ($\prod b{\in}Basis.\ (\int^+ x.\ f\ b\ x\ \partial lborel)$)
by (*simp add*: *lborel-def nn-integral-distr product-nn-integral-setprod*
product-nn-integral-singleton)

lemma *emeasure-lborel-Icc*[*simp*]:

fixes $l\ u :: real$
assumes [*simp*]: $l \leq u$
shows *emeasure lborel* $\{l\ ..\ u\} = u - l$
proof −
have $((\lambda f.\ f\ 1) -` \{l..u\} \cap space\ (Pi_M\ \{1\}\ (\lambda b.\ interval\text{-}measure\ (\lambda x.\ x)))) = \{1::real\} \rightarrow_E \{l..u\}$
by (*auto simp*: *space-PiM*)
then show *?thesis*
by (*simp add*: *lborel-def emeasure-distr emeasure-PiM emeasure-interval-measure-Icc continuous-on-id*)
qed

lemma *emeasure-lborel-Icc-eq*: *emeasure lborel* $\{l\ ..\ u\} = ennreal\ (if\ l \leq u\ then\ u - l\ else\ 0)$
by *simp*

lemma *emeasure-lborel-cbox*[*simp*]:
assumes [*simp*]: $\bigwedge b.\ b \in Basis \Longrightarrow l \cdot b \leq u \cdot b$
shows *emeasure lborel* $(cbox\ l\ u) = (\prod b{\in}Basis.\ (u - l) \cdot b)$
proof −
have $(\lambda x.\ \prod b{\in}Basis.\ indicator\ \{l{\cdot}b\ ..\ u{\cdot}b\}\ (x \cdot b) :: ennreal) = indicator\ (cbox\ l\ u)$
by (*auto simp*: *fun-eq-iff cbox-def split*: *split-indicator*)
then have *emeasure lborel* $(cbox\ l\ u) = (\int^+ x.\ (\prod b{\in}Basis.\ indicator\ \{l{\cdot}b\ ..\ u{\cdot}b\}\ (x \cdot b))\ \partial lborel)$
by *simp*
also have $\ldots = (\prod b{\in}Basis.\ (u - l) \cdot b)$
by (*subst nn-integral-lborel-setprod*) (*simp-all add*: *setprod-ennreal inner-diff-left*)
finally show *?thesis* **.**
qed

lemma *AE-lborel-singleton*: *AE x in lborel*::*'a*::*euclidean-space measure*. $x \neq c$
using *SOME-Basis AE-discrete-difference* [*of* $\{c\}$ *lborel*] *emeasure-lborel-cbox* [*of c c*]
by (*auto simp add*: *cbox-sing setprod-constant power-0-left*)

lemma *emeasure-lborel-Ioo*[*simp*]:
assumes [*simp*]: $l \leq u$
shows *emeasure lborel* $\{l <..< u\} = ennreal\ (u - l)$
proof −
have *emeasure lborel* $\{l <..< u\} =$ *emeasure lborel* $\{l\ ..\ u\}$
using *AE-lborel-singleton*[*of u*] *AE-lborel-singleton*[*of l*] **by** (*intro emeasure-eq-AE*) *auto*
then show *?thesis*
by *simp*
qed

lemma *emeasure-lborel-Ioc*[*simp*]:
assumes [*simp*]: $l \leq u$

shows *emeasure lborel* $\{l <.. u\} = ennreal\ (u - l)$
proof −
have *emeasure lborel* $\{l <.. u\} =$ *emeasure lborel* $\{l .. u\}$
using *AE-lborel-singleton*[*of u*] *AE-lborel-singleton*[*of l*] **by** (*intro emeasure-eq-AE*) *auto*
then show *?thesis*
by *simp*
qed

lemma *emeasure-lborel-Ico*[*simp*]:
assumes [*simp*]: $l \leq u$
shows *emeasure lborel* $\{l ..< u\} = ennreal\ (u - l)$
proof −
have *emeasure lborel* $\{l ..< u\} =$ *emeasure lborel* $\{l .. u\}$
using *AE-lborel-singleton*[*of u*] *AE-lborel-singleton*[*of l*] **by** (*intro emeasure-eq-AE*) *auto*
then show *?thesis*
by *simp*
qed

lemma *emeasure-lborel-box*[*simp*]:
assumes [*simp*]: $\bigwedge b.\ b \in Basis \Longrightarrow l \cdot b \leq u \cdot b$
shows *emeasure lborel* $(box\ l\ u) = (\prod b{\in}Basis.\ (u - l) \cdot b)$
proof −
have $(\lambda x.\ \prod b{\in}Basis.\ indicator\ \{l{\cdot}b <..< u{\cdot}b\}\ (x \cdot b) :: ennreal) = indicator\ (box\ l\ u)$
by (*auto simp*: *fun-eq-iff box-def split*: *split-indicator*)
then have *emeasure lborel* $(box\ l\ u) = (\int^+ x.\ (\prod b{\in}Basis.\ indicator\ \{l{\cdot}b <..< u{\cdot}b\}\ (x \cdot b))\ \partial lborel)$
by *simp*
also have $\ldots = (\prod b{\in}Basis.\ (u - l) \cdot b)$
by (*subst nn-integral-lborel-setprod*) (*simp-all add*: *setprod-ennreal inner-diff-left*)
finally show *?thesis* **.**
qed

lemma *emeasure-lborel-cbox-eq*:
emeasure lborel $(cbox\ l\ u) = (if\ \forall b{\in}Basis.\ l \cdot b \leq u \cdot b\ then\ \prod b{\in}Basis.\ (u - l) \cdot b\ else\ 0)$
using *box-eq-empty*(*2*)[*THEN iffD2*, *of u l*] **by** (*auto simp*: *not-le*)

lemma *emeasure-lborel-box-eq*:
emeasure lborel $(box\ l\ u) = (if\ \forall b{\in}Basis.\ l \cdot b \leq u \cdot b\ then\ \prod b{\in}Basis.\ (u - l) \cdot b\ else\ 0)$
using *box-eq-empty*(*1*)[*THEN iffD2*, *of u l*] **by** (*auto simp*: *not-le dest*!: *less-imp-le*) *force*

lemma
fixes *l u* :: *real*
assumes [*simp*]: $l \leq u$

shows *measure-lborel-Icc*[*simp*]: *measure lborel* {*l* .. *u*} = *u* − *l*
and *measure-lborel-Ico*[*simp*]: *measure lborel* {*l* ..< *u*} = *u* − *l*
and *measure-lborel-Ioc*[*simp*]: *measure lborel* {*l* <.. *u*} = *u* − *l*
and *measure-lborel-Ioo*[*simp*]: *measure lborel* {*l* <..< *u*} = *u* − *l*
by (*simp-all add*: *measure-def*)

lemma
assumes [*simp*]: ⋀*b*. *b* ∈ *Basis* ⟹ *l* · *b* ≤ *u* · *b*
shows *measure-lborel-box*[*simp*]: *measure lborel* (*box l u*) = (∏ *b*∈*Basis*. (*u* − *l*) · *b*)
and *measure-lborel-cbox*[*simp*]: *measure lborel* (*cbox l u*) = (∏ *b*∈*Basis*. (*u* − *l*) · *b*)
by (*simp-all add*: *measure-def inner-diff-left setprod-nonneg*)

lemma *sigma-finite-lborel*: *sigma-finite-measure lborel*
proof
show ∃ *A*::′*a set set*. *countable A* ∧ *A* ⊆ *sets lborel* ∧ ⋃ *A* = *space lborel* ∧ (∀ *a*∈*A*. *emeasure lborel a* ≠ ∞)
by (*intro exI*[*of - range* (λ*n*::*nat*. *box* (− *real n* ∗$_R$ *One*) (*real n* ∗$_R$ *One*))])
(*auto simp*: *emeasure-lborel-cbox-eq UN-box-eq-UNIV*)
qed

end

lemma *emeasure-lborel-UNIV*: *emeasure lborel* (*UNIV*::′*a*::*euclidean-space set*) = ∞
proof −
{ fix *n*::*nat*
let *?Ba* = *Basis* :: ′*a set*
have *real n* ≤ (*2*::*real*) ^ *card ?Ba* ∗ *real n*
by (*simp add*: *mult-le-cancel-right1*)
also
have ... ≤ (*2*::*real*) ^ *card ?Ba* ∗ *real* (*Suc n*) ^ *card ?Ba*
apply (*rule mult-left-mono*)
apply (*metis DIM-positive One-nat-def less-eq-Suc-le less-imp-le of-nat-le-iff of-nat-power self-le-power zero-less-Suc*)
apply (*simp add*: *DIM-positive*)
done
finally have *real n* ≤ (*2*::*real*) ^ *card ?Ba* ∗ *real* (*Suc n*) ^ *card ?Ba* .
} note [*intro*!] = *this*
show *?thesis*
unfolding *UN-box-eq-UNIV*[*symmetric*]
apply (*subst SUP-emeasure-incseq*[*symmetric*])
apply (*auto simp*: *incseq-def subset-box inner-add-left setprod-constant*
simp del: *Sup-eq-top-iff SUP-eq-top-iff*
intro!: *ennreal-SUP-eq-top*)
done
qed

lemma *emeasure-lborel-singleton*[*simp*]: *emeasure lborel* {*x*} = *0*
using *emeasure-lborel-cbox*[*of x x*] *nonempty-Basis*
by (*auto simp del*: *emeasure-lborel-cbox nonempty-Basis simp add*: *cbox-sing setprod-constant*)

lemma *emeasure-lborel-countable*:
fixes *A* :: *'a*::*euclidean-space set*
assumes *countable A*
shows *emeasure lborel A* = *0*
proof −
have *A* ⊆ (⋃ *i*. {*from-nat-into A i*}) **using** *from-nat-into-surj assms* **by** *force*
moreover have *emeasure lborel* (⋃ *i*. {*from-nat-into A i*}) = *0*
by (*rule emeasure-UN-eq-0*) *auto*
ultimately have *emeasure lborel A* ≤ *0* **using** *emeasure-mono*
by (*smt UN-E emeasure-empty equalityI from-nat-into order-refl singletonD subsetI*)
thus *?thesis* **by** (*auto simp add*:)
qed

lemma *countable-imp-null-set-lborel*: *countable A* ⟹ *A* ∈ *null-sets lborel*
by (*simp add*: *null-sets-def emeasure-lborel-countable sets.countable*)

lemma *finite-imp-null-set-lborel*: *finite A* ⟹ *A* ∈ *null-sets lborel*
by (*intro countable-imp-null-set-lborel countable-finite*)

lemma *lborel-neq-count-space*[*simp*]: *lborel* ≠ *count-space* (*A*::(*'a*::*ordered-euclidean-space*) *set*)
proof
assume *asm*: *lborel* = *count-space A*
have *space lborel* = *UNIV* **by** *simp*
hence [*simp*]: *A* = *UNIV* **by** (*subst* (*asm*) *asm*) (*simp only*: *space-count-space*)
have *emeasure lborel* {*undefined*::*'a*} = *1*
by (*subst asm*, *subst emeasure-count-space-finite*) *auto*
moreover have *emeasure lborel* {*undefined*} ≠ *1* **by** *simp*
ultimately show *False* **by** *contradiction*
qed

10.3 Affine transformation on the Lebesgue-Borel

lemma *lborel-eqI*:
fixes *M* :: *'a*::*euclidean-space measure*
assumes *emeasure-eq*: ⋀*l u*. (⋀*b*. *b* ∈ *Basis* ⟹ *l* ∙ *b* ≤ *u* ∙ *b*) ⟹ *emeasure M* (*box l u*) = (∏ *b*∈*Basis*. (*u* − *l*) ∙ *b*)
assumes *sets-eq*: *sets M* = *sets borel*
shows *lborel* = *M*
proof (*rule measure-eqI-generator-eq*)
let *?E* = *range* (λ(*a*, *b*). *box a b*::*'a set*)
show *Int-stable ?E*
by (*auto simp*: *Int-stable-def box-Int-box*)

show *?E ⊆ Pow UNIV sets lborel = sigma-sets UNIV ?E sets M = sigma-sets UNIV ?E*
by (*simp-all add*: *borel-eq-box sets-eq*)

let *?A = λn::nat. box (− (real n ∗R One)) (real n ∗R One) :: 'a set*
show *range ?A ⊆ ?E (⋃ i. ?A i) = UNIV*
unfolding *UN-box-eq-UNIV* **by** *auto*

{ fix *i* **show** *emeasure lborel (?A i) ≠ ∞* **by** *auto* **}**
{ fix *X* **assume** *X ∈ ?E* **then show** *emeasure lborel X = emeasure M X*
apply (*auto simp*: *emeasure-eq emeasure-lborel-box-eq*)
apply (*subst box-eq-empty(1)*[*THEN iffD2*])
apply (*auto intro*: *less-imp-le simp*: *not-le*)
done **}**
qed

lemma *lborel-affine*:
fixes *t :: 'a::euclidean-space* **assumes** *c ≠ 0*
shows *lborel = density (distr lborel borel (λx. t + c ∗R x)) (λ-. |c| ^DIM('a))*
(**is** *- = ?D*)
proof (*rule lborel-eqI*)
let *?B = Basis :: 'a set*
fix *l u* **assume** *le*: *⋀b. b ∈ ?B ⟹ l ∙ b ≤ u ∙ b*
show *emeasure ?D (box l u) = (∏ b∈?B. (u − l) ∙ b)*
proof *cases*
assume *0 < c*
then have *(λx. t + c ∗R x) −‘ box l u = box ((l − t) /R c) ((u − t) /R c)*
by (*auto simp*: *field-simps box-def inner-simps*)
with ‹*0 < c*› **show** *?thesis*
using *le*
by (*auto simp*: *field-simps inner-simps setprod-dividef setprod-constant setprod-nonneg*
ennreal-mult[*symmetric*] *emeasure-density nn-integral-distr*
emeasure-distr
nn-integral-cmult emeasure-lborel-box-eq borel-measurable-indicator')
next
assume *¬ 0 < c* **with** ‹*c ≠ 0*› **have** *c < 0* **by** *auto*
then have *box ((u − t) /R c) ((l − t) /R c) = (λx. t + c ∗R x) −‘ box l u*
by (*auto simp*: *field-simps box-def inner-simps*)
then have *∗: ⋀x. indicator (box l u) (t + c ∗R x) = (indicator (box ((u − t)*
/R c) ((l − t) /R c)) x :: ennreal)
by (*auto split*: *split-indicator*)
have *∗∗: (∏ x∈Basis. (l ∙ x − u ∙ x) / c) = (∏ x∈Basis. u ∙ x − l ∙ x) / (−c)*
^ card (Basis::'a set)
using ‹*c < 0*›
by (*auto simp add*: *field-simps setprod-dividef*[*symmetric*] *setprod-constant*[*symmetric*]
intro!: *setprod.cong*)
show *?thesis*
using ‹*c < 0*› *le*

by (*auto simp*: ∗ ∗∗ *field-simps emeasure-density nn-integral-distr nn-integral-cmult emeasure-lborel-box-eq inner-simps setprod-nonneg ennreal-mult*[*symmetric*] *borel-measurable-indicator'*)
qed
qed *simp*

lemma *lborel-real-affine*:
$c \neq 0 \Longrightarrow lborel = density\ (distr\ lborel\ borel\ (\lambda x.\ t + c * x))\ (\lambda\text{-}.\ ennreal\ (abs\ c))$
using *lborel-affine*[*of c t*] **by** *simp*

lemma *AE-borel-affine*:
fixes $P :: real \Rightarrow bool$
shows $c \neq 0 \Longrightarrow Measurable.pred\ borel\ P \Longrightarrow AE\ x\ in\ lborel.\ P\ x \Longrightarrow AE\ x\ in\ lborel.\ P\ (t + c * x)$
by (*subst lborel-real-affine*[**where** $t{=}-\ t\ /\ c$ **and** $c{=}1\ /\ c$])
(*simp-all add*: *AE-density AE-distr-iff field-simps*)

lemma *nn-integral-real-affine*:
fixes $c :: real$ **assumes** [*measurable*]: $f \in borel\text{-}measurable\ borel$ **and** c: $c \neq 0$
shows $(\int^+ x.\ f\ x\ \partial lborel) = |c| * (\int^+ x.\ f\ (t + c * x)\ \partial lborel)$
by (*subst lborel-real-affine*[*OF c, of t*])
(*simp add*: *nn-integral-density nn-integral-distr nn-integral-cmult*)

lemma *lborel-integrable-real-affine*:
fixes $f :: real \Rightarrow 'a :: \{banach, second\text{-}countable\text{-}topology\}$
assumes *f*: *integrable lborel f*
shows $c \neq 0 \Longrightarrow integrable\ lborel\ (\lambda x.\ f\ (t + c * x))$
using *f f*[*THEN borel-measurable-integrable*] **unfolding** *integrable-iff-bounded*
by (*subst* (*asm*) *nn-integral-real-affine*[**where** $c{=}c$ **and** $t{=}t$]) (*auto simp*: *ennreal-mult-less-top*)

lemma *lborel-integrable-real-affine-iff*:
fixes $f :: real \Rightarrow 'a :: \{banach, second\text{-}countable\text{-}topology\}$
shows $c \neq 0 \Longrightarrow integrable\ lborel\ (\lambda x.\ f\ (t + c * x)) \longleftrightarrow integrable\ lborel\ f$
using
lborel-integrable-real-affine[*of f c t*]
lborel-integrable-real-affine[*of* $\lambda x.\ f\ (t + c * x)\ 1/c\ -t/c$]
by (*auto simp add*: *field-simps*)

lemma *lborel-integral-real-affine*:
fixes $f :: real \Rightarrow 'a :: \{banach, second\text{-}countable\text{-}topology\}$ **and** $c :: real$
assumes c: $c \neq 0$ **shows** $(\int x.\ f\ x\ \partial\ lborel) = |c| *_R (\int x.\ f\ (t + c * x)\ \partial lborel)$
proof *cases*
assume *f*[*measurable*]: *integrable lborel f* **then show** *?thesis*
using *c f f*[*THEN borel-measurable-integrable*] *f*[*THEN lborel-integrable-real-affine, of c t*]
by (*subst lborel-real-affine*[*OF c, of t*])
(*simp add*: *integral-density integral-distr*)
next

assume ¬ *integrable lborel f* **with** *c* **show** *?thesis*
by (*simp add*: *lborel-integrable-real-affine-iff not-integrable-integral-eq*)
qed

lemma *divideR-right*:
fixes *x y* :: *'a*::*real-normed-vector*
shows $r \neq 0 \Longrightarrow y = x /_R r \longleftrightarrow r *_R y = x$
using *scaleR-cancel-left*[*of r y x* $/_R$ *r*] **by** *simp*

lemma *lborel-has-bochner-integral-real-affine-iff*:
fixes *x* :: *'a* :: {*banach*, *second-countable-topology*}
shows $c \neq 0 \Longrightarrow$
has-bochner-integral lborel f x $\longleftrightarrow$
has-bochner-integral lborel ($\lambda x.\ f\ (t + c * x)$) ($x /_R |c|$)
unfolding *has-bochner-integral-iff lborel-integrable-real-affine-iff*
by (*simp-all add*: *lborel-integral-real-affine*[*symmetric*] *divideR-right cong*: *conj-cong*)

lemma *lborel-distr-uminus*: *distr lborel borel uminus* = (*lborel* :: *real measure*)
by (*subst lborel-real-affine*[*of* −*1 0*])
(*auto simp*: *density-1 one-ennreal-def*[*symmetric*])

lemma *lborel-distr-mult*:
assumes (*c*::*real*) ≠ *0*
shows *distr lborel borel* (*op* ∗ *c*) = *density lborel* (λ-. *inverse* |*c*|)
proof−
have *distr lborel borel* (*op* ∗ *c*) = *distr lborel lborel* (*op* ∗ *c*) **by** (*simp cong*: *distr-cong*)
also from *assms* **have** ... = *density lborel* (λ-. *inverse* |*c*|)
by (*subst lborel-real-affine*[*of inverse c 0*]) (*auto simp*: *o-def distr-density-distr*)
finally show *?thesis* **.**
qed

lemma *lborel-distr-mult'*:
assumes (*c*::*real*) ≠ *0*
shows *lborel* = *density* (*distr lborel borel* (*op* ∗ *c*)) (λ-. |*c*|)
proof−
have *lborel* = *density lborel* (λ-. *1*) **by** (*rule density-1*[*symmetric*])
also from *assms* **have** (λ-. *1* :: *ennreal*) = (λ-. *inverse* |*c*| ∗ |*c*|) **by** (*intro ext*) *simp*
also have *density lborel* ... = *density* (*density lborel* (λ-. *inverse* |*c*|)) (λ-. |*c*|)
by (*subst density-density-eq*) (*auto simp*: *ennreal-mult*)
also from *assms* **have** *density lborel* (λ-. *inverse* |*c*|) = *distr lborel borel* (*op* ∗ *c*)
by (*rule lborel-distr-mult*[*symmetric*])
finally show *?thesis* **.**
qed

lemma *lborel-distr-plus*: *distr lborel borel* (*op* + *c*) = (*lborel* :: *real measure*)
by (*subst lborel-real-affine*[*of 1 c*]) (*auto simp*: *density-1 one-ennreal-def*[*symmetric*])

interpretation *lborel*: *sigma-finite-measure lborel*
by (*rule sigma-finite-lborel*)

interpretation *lborel-pair*: *pair-sigma-finite lborel lborel* **..**

lemma *lborel-prod*:
lborel $\bigotimes_M$ *lborel* = (*lborel* :: ('*a*::*euclidean-space* × '*b*::*euclidean-space*) *measure*)
proof (*rule lborel-eqI*[*symmetric*], *clarify*)
fix *la ua* :: '*a* **and** *lb ub* :: '*b*
assume *lu*: $\bigwedge a\ b.\ (a, b) \in Basis \Longrightarrow (la, lb) \cdot (a, b) \leq (ua, ub) \cdot (a, b)$
have [*simp*]:
$\bigwedge b.\ b \in Basis \Longrightarrow la \cdot b \leq ua \cdot b$
$\bigwedge b.\ b \in Basis \Longrightarrow lb \cdot b \leq ub \cdot b$
inj-on (λ*u*. (*u*, *0*)) *Basis inj-on* (λ*u*. (*0*, *u*)) *Basis*
(λ*u*. (*u*, *0*)) ' *Basis* ∩ (λ*u*. (*0*, *u*)) ' *Basis* = {}
box (*la*, *lb*) (*ua*, *ub*) = *box la ua* × *box lb ub*
using *lu*[*of - 0*] *lu*[*of 0*] **by** (*auto intro*!: *inj-onI simp add*: *Basis-prod-def ball-Un box-def*)
show *emeasure* (*lborel* $\bigotimes_M$ *lborel*) (*box* (*la*, *lb*) (*ua*, *ub*)) =
ennreal (*setprod* (*op* · ((*ua*, *ub*) − (*la*, *lb*))) *Basis*)
by (*simp add*: *lborel.emeasure-pair-measure-Times Basis-prod-def setprod.union-disjoint setprod.reindex ennreal-mult inner-diff-left setprod-nonneg*)
qed (*simp add*: *borel-prod*[*symmetric*])

lemma *lborelD-Collect*[*measurable* (*raw*)]: {*x*∈*space borel*. *P x*} ∈ *sets borel* ⟹ {*x*∈*space lborel*. *P x*} ∈ *sets lborel* **by** *simp*
lemma *lborelD*[*measurable* (*raw*)]: *A* ∈ *sets borel* ⟹ *A* ∈ *sets lborel* **by** *simp*

10.4 Equivalence Lebesgue integral on *lborel* and HK-integral

lemma *has-integral-measure-lborel*:
fixes *A* :: '*a*::*euclidean-space set*
assumes *A*[*measurable*]: *A* ∈ *sets borel* **and** *finite*: *emeasure lborel A* < ∞
shows ((λ*x*. *1*) *has-integral measure lborel A*) *A*
proof −
{ fix *l u* :: '*a*
have ((λ*x*. *1*) *has-integral measure lborel* (*box l u*)) (*box l u*)
proof *cases*
assume ∀ *b*∈*Basis*. *l* · *b* ≤ *u* · *b*
then show *?thesis*
apply *simp*
apply (*subst has-integral-restrict*[*symmetric*, *OF box-subset-cbox*])
apply (*subst has-integral-spike-interior-eq*[**where** *g*=λ-. *1*])
using *has-integral-const*[*of 1*::*real l u*]
apply (*simp-all add*: *inner-diff-left*[*symmetric*] *content-cbox-cases*)
done
next

```
    assume ¬ (∀ b∈Basis. l • b ≤ u • b)
    then have box l u = {}
      unfolding box-eq-empty by (auto simp: not-le intro: less-imp-le)
    then show ?thesis
      by simp
  qed }
 note has-integral-box = this

 { fix a b :: 'a let ?M = λA. measure lborel (A ∩ box a b)
  have Int-stable  (range (λ(a, b). box a b))
    by (auto simp: Int-stable-def box-Int-box)
  moreover have (range (λ(a, b). box a b)) ⊆ Pow UNIV
    by auto
  moreover have A ∈ sigma-sets UNIV (range (λ(a, b). box a b))
    using A unfolding borel-eq-box by simp
  ultimately have ((λx. 1) has-integral ?M A) (A ∩ box a b)
  proof (induction rule: sigma-sets-induct-disjoint)
    case (basic A) then show ?case
      by (auto simp: box-Int-box has-integral-box)
  next
    case empty then show ?case
      by simp
  next
    case (compl A)
    then have [measurable]: A ∈ sets borel
      by (simp add: borel-eq-box)

    have ((λx. 1) has-integral ?M (box a b)) (box a b)
      by (simp add: has-integral-box)
    moreover have ((λx. if x ∈ A ∩ box a b then 1 else 0) has-integral ?M A)
(box a b)
      by (subst has-integral-restrict) (auto intro: compl)
    ultimately have ((λx. 1 − (if x ∈ A ∩ box a b then 1 else 0)) has-integral
?M (box a b) − ?M A) (box a b)
      by (rule has-integral-sub)
    then have ((λx. (if x ∈ (UNIV − A) ∩ box a b then 1 else 0)) has-integral
?M (box a b) − ?M A) (box a b)
      by (rule has-integral-cong[THEN iffD1, rotated 1]) auto
    then have ((λx. 1) has-integral ?M (box a b) − ?M A) ((UNIV − A) ∩ box
a b)
      by (subst (asm) has-integral-restrict) auto
    also have ?M (box a b) − ?M A = ?M (UNIV − A)
        by (subst measure-Diff[symmetric]) (auto simp: emeasure-lborel-box-eq
Diff-Int-distrib2)
    finally show ?case .
  next
    case (union F)
    then have [measurable]: ⋀i. F i ∈ sets borel
      by (simp add: borel-eq-box subset-eq)
```

have ((λx. *if x ∈ UNION UNIV F ∩ box a b then 1 else 0*) *has-integral ?M* (⋃ *i*. *F i*)) (*box a b*)
proof (*rule has-integral-monotone-convergence-increasing*)
let *?f* = λ*k x*. ∑ *i*<*k*. *if x ∈ F i ∩ box a b then 1 else 0* :: *real*
show ⋀*k*. (*?f k has-integral* (∑ *i*<*k*. *?M* (*F i*))) (*box a b*)
using *union.IH* **by** (*auto intro*!: *has-integral-setsum simp del*: *Int-iff*)
show ⋀*k x*. *?f k x* ≤ *?f* (*Suc k*) *x*
by (*intro setsum-mono2*) *auto*
from *union*(*1*) **have** ∗: ⋀*x i j*. *x ∈ F i* ⟹ *x ∈ F j* ⟷ *j* = *i*
by (*auto simp add*: *disjoint-family-on-def*)
show ⋀*x*. (λ*k*. *?f k x*) ⟶ (*if x ∈ UNION UNIV F ∩ box a b then 1 else 0*)
apply (*auto simp*: ∗ *setsum.If-cases Iio-Int-singleton*)
apply (*rule-tac k*=*Suc xa* **in** *LIMSEQ-offset*)
apply *simp*
done
have ∗: *emeasure lborel* ((⋃ *x*. *F x*) ∩ *box a b*) ≤ *emeasure lborel* (*box a b*)
by (*intro emeasure-mono*) *auto*

with *union*(*1*) **show** (λ*k*. ∑ *i*<*k*. *?M* (*F i*)) ⟶ *?M* (⋃ *i*. *F i*)
unfolding *sums-def*[*symmetric*] *UN-extend-simps*
by (*intro measure-UNION*) (*auto simp*: *disjoint-family-on-def emeasure-lborel-box-eq top-unique*)
qed
then show *?case*
by (*subst* (*asm*) *has-integral-restrict*) *auto*
qed }
note ∗ = *this*

show *?thesis*
proof (*rule has-integral-monotone-convergence-increasing*)
let *?B* = λ*n*::*nat*. *box* (− *real n* ∗$_R$ *One*) (*real n* ∗$_R$ *One*) :: *'a set*
let *?f* = λ*n*::*nat*. λ*x*. *if x ∈ A ∩ ?B n then 1 else 0* :: *real*
let *?M* = λ*n*. *measure lborel* (*A ∩ ?B n*)

show ⋀*n*::*nat*. (*?f n has-integral ?M n*) *A*
using ∗ **by** (*subst has-integral-restrict*) *simp-all*
show ⋀*k x*. *?f k x* ≤ *?f* (*Suc k*) *x*
by (*auto simp*: *box-def*)
{ **fix** *x* **assume** *x ∈ A*
moreover have (λ*k*. *indicator* (*A ∩ ?B k*) *x* :: *real*) ⟶ *indicator* (⋃ *k*::*nat*. *A ∩ ?B k*) *x*
by (*intro LIMSEQ-indicator-incseq*) (*auto simp*: *incseq-def box-def*)
ultimately show (λ*k*. *if x ∈ A ∩ ?B k then 1 else 0*::*real*) ⟶ *1*
by (*simp add*: *indicator-def UN-box-eq-UNIV*) }

have (λ*n*. *emeasure lborel* (*A ∩ ?B n*)) ⟶ *emeasure lborel* (⋃ *n*::*nat*. *A ∩ ?B n*)
by (*intro Lim-emeasure-incseq*) (*auto simp*: *incseq-def box-def*)

also have (λn. *emeasure lborel* ($A \cap$ *?B n*)) = (λn. *measure lborel* ($A \cap$ *?B n*))
proof (*intro ext emeasure-eq-ennreal-measure*)
fix *n* **have** *emeasure lborel* ($A \cap$ *?B n*) $\leq$ *emeasure lborel* (*?B n*)
by (*intro emeasure-mono*) *auto*
then show *emeasure lborel* ($A \cap$ *?B n*) $\neq$ *top*
by (*auto simp*: *top-unique*)
qed
finally show (λn. *measure lborel* ($A \cap$ *?B n*)) $\longrightarrow$ *measure lborel A*
using *emeasure-eq-ennreal-measure*[*of lborel A*] *finite*
by (*simp add*: *UN-box-eq-UNIV less-top*)
qed
qed

lemma *nn-integral-has-integral*:
fixes *f*::'*a*::*euclidean-space* $\Rightarrow$ *real*
assumes *f*: *f* $\in$ *borel-measurable borel* $\bigwedge x$. *0* $\leq$ *f x* ($\int^+ x$. *f x* ∂*lborel*) = *ennreal r* *0* $\leq$ *r*
shows (*f has-integral r*) *UNIV*
using *f* **proof** (*induct f arbitrary*: *r rule*: *borel-measurable-induct-real*)
case (*set A*)
moreover then have ((λx. *1*) *has-integral measure lborel A*) *A*
by (*intro has-integral-measure-lborel*) (*auto simp*: *ennreal-indicator*)
ultimately show *?case*
by (*simp add*: *ennreal-indicator measure-def*) (*simp add*: *indicator-def*)
next
case (*mult g c*)
then have *ennreal c* $*$ ($\int^+ x$. *g x* ∂*lborel*) = *ennreal r*
by (*subst nn-integral-cmult*[*symmetric*]) (*auto simp*: *ennreal-mult*)
with ‹*0* $\leq$ *r*› ‹*0* $\leq$ *c*›
obtain *r'* **where** (*c* = *0* $\wedge$ *r* = *0*) $\vee$ (*0* $\leq$ *r'* $\wedge$ ($\int^+ x$. *ennreal* (*g x*) ∂*lborel*) = *ennreal r'* $\wedge$ *r* = *c* $*$ *r'*)
by (*cases* $\int^+ x$. *ennreal* (*g x*) ∂*lborel rule*: *ennreal-cases*)
(*auto split*: *if-split-asm simp*: *ennreal-mult-top ennreal-mult*[*symmetric*])
with *mult* **show** *?case*
by (*auto intro*!: *has-integral-cmult-real*)
next
case (*add g h*)
moreover
then have ($\int^+ x$. *h x* + *g x* ∂*lborel*) = ($\int^+ x$. *h x* ∂*lborel*) + ($\int^+ x$. *g x* ∂*lborel*)
by (*simp add*: *nn-integral-add*)
with *add* **obtain** *a b* **where** *0* $\leq$ *a* *0* $\leq$ *b* ($\int^+ x$. *h x* ∂*lborel*) = *ennreal a* ($\int^+ x$. *g x* ∂*lborel*) = *ennreal b r* = *a* + *b*
by (*cases* $\int^+ x$. *h x* ∂*lborel* $\int^+ x$. *g x* ∂*lborel rule*: *ennreal2-cases*)
(*auto simp*: *add-top nn-integral-add top-add ennreal-plus*[*symmetric*] *simp del*: *ennreal-plus*)
ultimately show *?case*
by (*auto intro*!: *has-integral-add*)

next
case (*seq U*)
note *seq*(*1*)[*measurable*] **and** *f*[*measurable*]

{ fix *i x*
have *U i x* ≤ *f x*
using *seq*(*5*)
apply (*rule LIMSEQ-le-const*)
using *seq*(*4*)
apply (*auto intro*!: *exI*[*of - i*] *simp*: *incseq-def le-fun-def*)
done }
note *U-le-f* = *this*

{ fix *i*
have (∫⁺*x*. *U i x* ∂*lborel*) ≤ (∫⁺*x*. *f x* ∂*lborel*)
using *seq*(*2*) *f*(*2*) *U-le-f* **by** (*intro nn-integral-mono*) *simp*
then obtain *p* **where** (∫⁺*x*. *U i x* ∂*lborel*) = *ennreal p p* ≤ *r* 0 ≤ *p*
using *seq*(*6*) ‹0≤*r*› **by** (*cases* ∫⁺*x*. *U i x* ∂*lborel rule*: *ennreal-cases*) (*auto simp*: *top-unique*)
moreover note *seq*
ultimately have ∃*p*. (∫⁺*x*. *U i x* ∂*lborel*) = *ennreal p* ∧ 0 ≤ *p* ∧ *p* ≤ *r* ∧ (*U i has-integral p*) *UNIV*
by *auto* **}**
then obtain *p* **where** *p*: ⋀*i*. (∫⁺*x*. *ennreal* (*U i x*) ∂*lborel*) = *ennreal* (*p i*)
and *bnd*: ⋀*i*. *p i* ≤ *r* ⋀*i*. 0 ≤ *p i*
and *U-int*: ⋀*i*.(*U i has-integral* (*p i*)) *UNIV* **by** *metis*

have *int-eq*: ⋀*i*. *integral UNIV* (*U i*) = *p i* **using** *U-int* **by** (*rule integral-unique*)

have ∗: *f integrable-on UNIV* ∧ (λ*k*. *integral UNIV* (*U k*)) ⟶ *integral UNIV f*
proof (*rule monotone-convergence-increasing*)
show ∀*k*. *U k integrable-on UNIV* **using** *U-int* **by** *auto*
show ∀*k*. ∀*x*∈*UNIV*. *U k x* ≤ *U* (*Suc k*) *x* **using** ‹*incseq U*› **by** (*auto simp*: *incseq-def le-fun-def*)
then show *bounded* {*integral UNIV* (*U k*) |*k*. *True*}
using *bnd int-eq* **by** (*auto simp*: *bounded-real intro*!: *exI*[*of - r*])
show ∀*x*∈*UNIV*. (λ*k*. *U k x*) ⟶ *f x*
using *seq* **by** *auto*
qed
moreover have (λ*i*. (∫⁺*x*. *U i x* ∂*lborel*)) ⟶ (∫⁺*x*. *f x* ∂*lborel*)
using *seq f*(*2*) *U-le-f* **by** (*intro nn-integral-dominated-convergence*[**where** *w*=*f*]) *auto*
ultimately have *integral UNIV f* = *r*
by (*auto simp add*: *bnd int-eq p seq intro*: *LIMSEQ-unique*)
with ∗ **show** *?case*
by (*simp add*: *has-integral-integral*)
qed

lemma *nn-integral-lborel-eq-integral*:
fixes f::$'a$::*euclidean-space* $\Rightarrow$ *real*
assumes f: $f \in$ *borel-measurable borel* $\bigwedge x.\ 0 \le f\ x$ $(\int^+ x.\ f\ x\ \partial lborel) < \infty$
shows $(\int^+ x.\ f\ x\ \partial lborel) =$ *integral UNIV f*
proof −
from *f*(*3*) **obtain** *r* **where** *r*: $(\int^+ x.\ f\ x\ \partial lborel) =$ *ennreal r* $0 \le r$
by (*cases* $\int^+ x.\ f\ x\ \partial lborel$ *rule*: *ennreal-cases*) *auto*
then show *?thesis*
using *nn-integral-has-integral*[*OF f*(*1*,*2*) *r*] **by** (*simp add*: *integral-unique*)
qed

lemma *nn-integral-integrable-on*:
fixes f::$'a$::*euclidean-space* $\Rightarrow$ *real*
assumes f: $f \in$ *borel-measurable borel* $\bigwedge x.\ 0 \le f\ x$ $(\int^+ x.\ f\ x\ \partial lborel) < \infty$
shows *f integrable-on UNIV*
proof −
from *f*(*3*) **obtain** *r* **where** *r*: $(\int^+ x.\ f\ x\ \partial lborel) =$ *ennreal r* $0 \le r$
by (*cases* $\int^+ x.\ f\ x\ \partial lborel$ *rule*: *ennreal-cases*) *auto*
then show *?thesis*
by (*intro has-integral-integrable*[**where** *i*=*r*] *nn-integral-has-integral*[**where** *r*=*r*] *f*)
qed

lemma *nn-integral-has-integral-lborel*:
fixes *f* :: $'a$::*euclidean-space* $\Rightarrow$ *real*
assumes *f-borel*: $f \in$ *borel-measurable borel* **and** *nonneg*: $\bigwedge x.\ 0 \le f\ x$
assumes *I*: (*f has-integral I*) *UNIV*
shows $integral^N$ *lborel f* = *I*
proof −
from *f-borel* **have** $(\lambda x.\ ennreal\ (f\ x)) \in$ *borel-measurable lborel* **by** *auto*
from *borel-measurable-implies-simple-function-sequence'*[*OF this*] **guess** *F* **. note** *F* = *this*
let *?B* = λi::*nat*. *box* $(-\ (real\ i *_R One))\ (real\ i *_R One)$ :: $'a$ *set*

note *F*(*1*)[*THEN borel-measurable-simple-function*, *measurable*]

have $0 \le I$
using *I* **by** (*rule has-integral-nonneg*) (*simp add*: *nonneg*)

have *F-le-f*: *enn2real* $(F\ i\ x) \le f\ x$ **for** *i x*
using *F*(*3*,*4*)[**where** *x*=*x*] *nonneg SUP-upper*[*of i UNIV* $\lambda i.\ F\ i\ x$]
by (*cases F i x rule*: *ennreal-cases*) *auto*
let *?F* = $\lambda i\ x.\ F\ i\ x * indicator\ (?B\ i)\ x$
have $(\int^+ x.\ ennreal\ (f\ x)\ \partial lborel) = (SUP\ i.\ integral^N\ lborel\ (\lambda x.\ ?F\ i\ x))$
proof (*subst nn-integral-monotone-convergence-SUP*[*symmetric*])
{ fix *x*
obtain *j* **where** *j*: $x \in ?B\ j$
using *UN-box-eq-UNIV* **by** *auto*

```
    have ennreal (f x) = (SUP i. F i x)
      using F(4)[of x] nonneg[of x] by (simp add: max-def)
    also have ... = (SUP i. ?F i x)
    proof (rule SUP-eq)
      fix i show ∃j∈UNIV. F i x ≤ ?F j x
        using j F(2)
        by (intro bexI[of - max i j])
           (auto split: split-max split-indicator simp: incseq-def le-fun-def box-def)
    qed (auto intro!: F split: split-indicator)
    finally have ennreal (f x) =  (SUP i. ?F i x) . }
  then show (∫⁺ x. ennreal (f x) ∂lborel) = (∫⁺ x. (SUP i. ?F i x) ∂lborel)
    by simp
qed (insert F, auto simp: incseq-def le-fun-def box-def split: split-indicator)
also have ... ≤ ennreal I
proof (rule SUP-least)
  fix i :: nat
  have finite-F: (∫⁺ x. ennreal (enn2real (F i x) * indicator (?B i) x) ∂lborel)
< ∞
  proof (rule nn-integral-bound-simple-function)
    have emeasure lborel {x ∈ space lborel. ennreal (enn2real (F i x) * indicator
(?B i) x) ≠ 0} ≤
      emeasure lborel (?B i)
      by (intro emeasure-mono) (auto split: split-indicator)
     then show emeasure lborel {x ∈ space lborel. ennreal (enn2real (F i x) *
indicator (?B i) x) ≠ 0} < ∞
      by (auto simp: less-top[symmetric] top-unique)
  qed (auto split: split-indicator
       intro!: F simple-function-compose1[where g=enn2real] simple-function-ennreal)

  have int-F: (λx. enn2real (F i x) * indicator (?B i) x) integrable-on UNIV
    using F(4) finite-F
  by (intro nn-integral-integrable-on) (auto split: split-indicator simp: enn2real-nonneg)

  have (∫⁺ x. F i x * indicator (?B i) x ∂lborel) =
    (∫⁺ x. ennreal (enn2real (F i x) * indicator (?B i) x) ∂lborel)
    using F(3,4)
  by (intro nn-integral-cong) (auto simp: image-iff eq-commute split: split-indicator)
  also have ... = ennreal (integral UNIV (λx. enn2real (F i x) * indicator (?B
i) x))
    using F
    by (intro nn-integral-lborel-eq-integral[OF - - finite-F])
       (auto split: split-indicator intro: enn2real-nonneg)
  also have ... ≤ ennreal I
      by (auto intro!: has-integral-le[OF integrable-integral[OF int-F] I] nonneg
F-le-f
            simp: ‹0 ≤ I› split: split-indicator )
  finally show (∫⁺ x. F i x * indicator (?B i) x ∂lborel) ≤ ennreal I .
qed
finally have (∫⁺ x. ennreal (f x) ∂lborel) < ∞
```

by (*auto simp*: *less-top*[*symmetric*] *top-unique*)
from *nn-integral-lborel-eq-integral*[*OF assms*(*1*,*2*) *this*] *I* **show** *?thesis*
by (*simp add*: *integral-unique*)
qed

lemma *has-integral-iff-emeasure-lborel*:
fixes *A* :: *'a*::*euclidean-space set*
assumes *A*[*measurable*]: *A* ∈ *sets borel* **and** [*simp*]: *0* ≤ *r*
shows ((λ*x*. *1*) *has-integral r*) *A* ⟷ *emeasure lborel A* = *ennreal r*
proof *cases*
assume *emeasure-A*: *emeasure lborel A* = ∞
have ¬ (λ*x*. *1*::*real*) *integrable-on A*
proof
assume *int*: (λ*x*. *1*::*real*) *integrable-on A*
then have (*indicator A*::*'a* ⇒ *real*) *integrable-on UNIV*
unfolding *indicator-def*[*abs-def*] *integrable-restrict-univ* **.**
then obtain *r* **where** ((*indicator A*::*'a*⇒*real*) *has-integral r*) *UNIV*
by *auto*
from *nn-integral-has-integral-lborel*[*OF - - this*] *emeasure-A* **show** *False*
by (*simp add*: *ennreal-indicator*)
qed
with *emeasure-A* **show** *?thesis*
by *auto*
next
assume *emeasure lborel A* ≠ ∞
moreover then have ((λ*x*. *1*) *has-integral measure lborel A*) *A*
by (*simp add*: *has-integral-measure-lborel less-top*)
ultimately show *?thesis*
by (*auto simp*: *emeasure-eq-ennreal-measure has-integral-unique*)
qed

lemma *has-integral-integral-real*:
fixes *f*::*'a*::*euclidean-space* ⇒ *real*
assumes *f*: *integrable lborel f*
shows (*f has-integral* (*integral*^{L} *lborel f*)) *UNIV*
using *f* **proof** *induct*
case (*base A c*) **then show** *?case*
by (*auto intro*!: *has-integral-mult-left simp*:)
(*simp add*: *emeasure-eq-ennreal-measure indicator-def has-integral-measure-lborel*)
next
case (*add f g*) **then show** *?case*
by (*auto intro*!: *has-integral-add*)
next
case (*lim f s*)
show *?case*
proof (*rule has-integral-dominated-convergence*)
show ⋀*i*. (*s i has-integral integral*^{L} *lborel* (*s i*)) *UNIV* **by** *fact*
show (λ*x*. *norm* (*2* ∗ *f x*)) *integrable-on UNIV*
using ‹*integrable lborel f*›

by (*intro nn-integral-integrable-on*)
(*auto simp*: *integrable-iff-bounded abs-mult nn-integral-cmult ennreal-mult ennreal-mult-less-top*)
show $\bigwedge k.\ \forall x{\in}UNIV.\ norm\ (s\ k\ x) \leq norm\ (2 * f\ x)$
using *lim* **by** (*auto simp add*: *abs-mult*)
show $\forall x{\in}UNIV.\ (\lambda k.\ s\ k\ x) \longrightarrow f\ x$
using *lim* **by** *auto*
show $(\lambda k.\ integral^L\ lborel\ (s\ k)) \longrightarrow integral^L\ lborel\ f$
using *lim lim*(*1*)[*THEN borel-measurable-integrable*]
by (*intro integral-dominated-convergence*[**where** $w{=}\lambda x.\ 2 * norm\ (f\ x)$]) *auto*
qed
qed

context
fixes $f{::}'a{::}euclidean\text{-}space \Rightarrow 'b{::}euclidean\text{-}space$
begin

lemma *has-integral-integral-lborel*:
assumes *f*: *integrable lborel f*
shows $(f\ has\text{-}integral\ (integral^L\ lborel\ f))\ UNIV$
proof −
have $((\lambda x.\ \sum b{\in}Basis.\ (f\ x \cdot b) *_R b)\ has\text{-}integral\ (\sum b{\in}Basis.\ integral^L\ lborel\ (\lambda x.\ f\ x \cdot b) *_R b))\ UNIV$
using *f* **by** (*intro has-integral-setsum finite-Basis ballI has-integral-scaleR-left has-integral-integral-real*) *auto*
also have *eq-f*: $(\lambda x.\ \sum b{\in}Basis.\ (f\ x \cdot b) *_R b) = f$
by (*simp add*: *fun-eq-iff euclidean-representation*)
also have $(\sum b{\in}Basis.\ integral^L\ lborel\ (\lambda x.\ f\ x \cdot b) *_R b) = integral^L\ lborel\ f$
using *f* **by** (*subst* (*2*) *eq-f*[*symmetric*]) *simp*
finally show *?thesis* **.**
qed

lemma *integrable-on-lborel*: *integrable lborel f* $\Longrightarrow$ *f integrable-on UNIV*
using *has-integral-integral-lborel* **by** *auto*

lemma *integral-lborel*: *integrable lborel f* $\Longrightarrow$ $integral\ UNIV\ f = (\int x.\ f\ x\ \partial lborel)$
using *has-integral-integral-lborel* **by** *auto*

end

10.5 Fundamental Theorem of Calculus for the Lebesgue integral

lemma *emeasure-bounded-finite*:
assumes *bounded A* **shows** *emeasure lborel A* $< \infty$
proof −
from *bounded-subset-cbox*[*OF* ‹*bounded A*›] **obtain** *a b* **where** $A \subseteq cbox\ a\ b$
by *auto*
then have $emeasure\ lborel\ A \leq emeasure\ lborel\ (cbox\ a\ b)$

by (*intro emeasure-mono*) *auto*
then show *?thesis*
by (*auto simp*: *emeasure-lborel-cbox-eq setprod-nonneg less-top*[*symmetric*] *top-unique split*: *if-split-asm*)
qed

lemma *emeasure-compact-finite*: *compact* $A \Longrightarrow$ *emeasure lborel* $A < \infty$
using *emeasure-bounded-finite*[*of* A] **by** (*auto intro*: *compact-imp-bounded*)

lemma *borel-integrable-compact*:
fixes $f :: 'a$::*euclidean-space* $\Rightarrow$ $'b$::{*banach*, *second-countable-topology*}
assumes *compact* S *continuous-on* S f
shows *integrable lborel* $(\lambda x.\ \mathit{indicator}\ S\ x *_R f\ x)$
proof *cases*
assume $S \neq \{\}$
have *continuous-on* S $(\lambda x.\ \mathit{norm}\ (f\ x))$
using *assms* **by** (*intro continuous-intros*)
from *continuous-attains-sup*[*OF* ‹*compact* S› ‹$S \neq \{\}$› *this*]
obtain M **where** M: $\bigwedge x.\ x \in S \Longrightarrow \mathit{norm}\ (f\ x) \leq M$
by *auto*

show *?thesis*
proof (*rule integrable-bound*)
show *integrable lborel* $(\lambda x.\ \mathit{indicator}\ S\ x * M)$
using *assms* **by** (*auto intro*!: *emeasure-compact-finite borel-compact integrable-mult-left*)
show $(\lambda x.\ \mathit{indicator}\ S\ x *_R f\ x) \in$ *borel-measurable lborel*
using *assms* **by** (*auto intro*!: *borel-measurable-continuous-on-indicator borel-compact*)
show *AE* x *in lborel*. *norm* $(\mathit{indicator}\ S\ x *_R f\ x) \leq \mathit{norm}\ (\mathit{indicator}\ S\ x * M)$
by (*auto split*: *split-indicator simp*: *abs-real-def dest*!: M)
qed
qed *simp*

lemma *borel-integrable-atLeastAtMost*:
fixes f :: *real* $\Rightarrow$ *real*
assumes f: $\bigwedge x.\ a \leq x \Longrightarrow x \leq b \Longrightarrow \mathit{isCont}\ f\ x$
shows *integrable lborel* $(\lambda x.\ f\ x * \mathit{indicator}\ \{a\ ..\ b\}\ x)$ (**is** *integrable - ?f*)
proof −
have *integrable lborel* $(\lambda x.\ \mathit{indicator}\ \{a\ ..\ b\}\ x *_R f\ x)$
proof (*rule borel-integrable-compact*)
from f **show** *continuous-on* $\{a..b\}$ f
by (*auto intro*: *continuous-at-imp-continuous-on*)
qed *simp*
then show *?thesis*
by (*auto simp*: *mult.commute*)
qed

For the positive integral we replace continuity with Borel-measurability.

lemma
fixes f :: *real* $\Rightarrow$ *real*

assumes [*measurable*]: $f \in$ *borel-measurable borel*
assumes *f*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow$ *DERIV F x :> f x* $\bigwedge x.\ x \in \{a..b\} \Longrightarrow 0 \leq f\ x$ **and** $a \leq b$
shows *nn-integral-FTC-Icc*: $(\int^+ x.$ *ennreal* $(f\ x) *$ *indicator* $\{a\ ..\ b\}\ x\ \partial lborel) = F\ b - F\ a$ (**is** *?nn*)
and *has-bochner-integral-FTC-Icc-nonneg*:
has-bochner-integral lborel $(\lambda x.\ f\ x *$ *indicator* $\{a\ ..\ b\}\ x)\ (F\ b - F\ a)$ (**is** *?has*)
and *integral-FTC-Icc-nonneg*: $(\int x.\ f\ x *$ *indicator* $\{a\ ..\ b\}\ x\ \partial lborel) = F\ b - F\ a$ (**is** *?eq*)
and *integrable-FTC-Icc-nonneg*: *integrable lborel* $(\lambda x.\ f\ x *$ *indicator* $\{a\ ..\ b\}\ x)$ (**is** *?int*)
proof −
have $*$: $(\lambda x.\ f\ x *$ *indicator* $\{a..b\}\ x) \in$ *borel-measurable borel* $\bigwedge x.\ 0 \leq f\ x *$ *indicator* $\{a..b\}\ x$
using *f*(*2*) **by** (*auto split*: *split-indicator*)

have *F-mono*: $a \leq x \Longrightarrow x \leq y \Longrightarrow y \leq b \Longrightarrow F\ x \leq F\ y$ **for** *x y*
using *f* **by** (*intro DERIV-nonneg-imp-nondecreasing*[*of x y F*]) (*auto intro*: *order-trans*)

have (*f has-integral F b − F a*) $\{a..b\}$
by (*intro fundamental-theorem-of-calculus*)
(*auto simp*: *has-field-derivative-iff-has-vector-derivative*[*symmetric*]
intro: *has-field-derivative-subset*[*OF f*(*1*)] ‹$a \leq b$›)
then have *i*: $((\lambda x.\ f\ x *$ *indicator* $\{a\ ..\ b\}\ x)$ *has-integral F b − F a*) *UNIV*
unfolding *indicator-def if-distrib*[**where** *f*=$\lambda x.\ a * x$ **for** *a*]
by (*simp cong del*: *if-cong del*: *atLeastAtMost-iff*)
then have *nn*: $(\int^+ x.\ f\ x *$ *indicator* $\{a\ ..\ b\}\ x\ \partial lborel) = F\ b - F\ a$
by (*rule nn-integral-has-integral-lborel*[*OF* $*$])
then show *?has*
by (*rule has-bochner-integral-nn-integral*[*rotated 3*]) (*simp-all add*: $*$ *F-mono* ‹$a \leq b$›)
then show *?eq ?int*
unfolding *has-bochner-integral-iff* **by** *auto*
show *?nn*
by (*subst nn*[*symmetric*])
(*auto intro*!: *nn-integral-cong simp add*: *ennreal-mult f split*: *split-indicator*)
qed

lemma
fixes *f* :: *real* ⇒ *'a* :: *euclidean-space*
assumes $a \leq b$
assumes $\bigwedge x.\ a \leq x \Longrightarrow x \leq b \Longrightarrow$ (*F has-vector-derivative f x*) (*at x within* $\{a\ ..\ b\}$)
assumes *cont*: *continuous-on* $\{a\ ..\ b\}\ f$
shows *has-bochner-integral-FTC-Icc*:
has-bochner-integral lborel $(\lambda x.$ *indicator* $\{a\ ..\ b\}\ x *_R f\ x)\ (F\ b - F\ a)$ (**is** *?has*)

and *integral-FTC-Icc*: $(\int x.\ indicator\ \{a\ ..\ b\}\ x *_R f\ x\ \partial lborel) = F\ b - F\ a$ (**is** *?eq*)
proof −
let *?f* = $\lambda x.\ indicator\ \{a\ ..\ b\}\ x *_R f\ x$
have *int*: *integrable lborel ?f*
using *borel-integrable-compact*[*OF - cont*] **by** *auto*
have (*f has-integral F b − F a*) {*a..b*}
using *assms*(*1*,*2*) **by** (*intro fundamental-theorem-of-calculus*) *auto*
moreover
have (*f has-integral integral*^{L} *lborel ?f*) {*a..b*}
using *has-integral-integral-lborel*[*OF int*]
unfolding *indicator-def if-distrib*[**where** *f*=$\lambda x.\ x *_R a$ **for** *a*]
by (*simp cong del*: *if-cong del*: *atLeastAtMost-iff*)
ultimately show *?eq*
by (*auto dest*: *has-integral-unique*)
then show *?has*
using *int* **by** (*auto simp*: *has-bochner-integral-iff*)
qed

lemma
fixes *f* :: *real* ⇒ *real*
assumes $a \le b$
assumes *deriv*: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow DERIV\ F\ x :> f\ x$
assumes *cont*: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow isCont\ f\ x$
shows *has-bochner-integral-FTC-Icc-real*:
has-bochner-integral lborel $(\lambda x.\ f\ x * indicator\ \{a\ ..\ b\}\ x)\ (F\ b - F\ a)$ (**is** *?has*)
and *integral-FTC-Icc-real*: $(\int x.\ f\ x * indicator\ \{a\ ..\ b\}\ x\ \partial lborel) = F\ b - F\ a$ (**is** *?eq*)
proof −
have *1*: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow$ (*F has-vector-derivative f x*) (*at x within* {*a .. b*})
unfolding *has-field-derivative-iff-has-vector-derivative*[*symmetric*]
using *deriv* **by** (*auto intro*: *DERIV-subset*)
have *2*: *continuous-on* {*a .. b*} *f*
using *cont* **by** (*intro continuous-at-imp-continuous-on*) *auto*
show *?has ?eq*
using *has-bochner-integral-FTC-Icc*[*OF* ‹$a \le b$› *1 2*] *integral-FTC-Icc*[*OF* ‹$a \le b$› *1 2*]
by (*auto simp*: *mult.commute*)
qed

lemma *nn-integral-FTC-atLeast*:
fixes *f* :: *real* ⇒ *real*
assumes *f-borel*: $f \in borel\text{-}measurable\ borel$
assumes *f*: $\bigwedge x.\ a \le x \Longrightarrow DERIV\ F\ x :> f\ x$
assumes *nonneg*: $\bigwedge x.\ a \le x \Longrightarrow 0 \le f\ x$
assumes *lim*: $(F \longrightarrow T)\ at\text{-}top$
shows $(\int^+ x.\ ennreal\ (f\ x) * indicator\ \{a\ ..\}\ x\ \partial lborel) = T - F\ a$

proof −
let *?f* = λ(*i*::*nat*) (*x*::*real*). *ennreal* (*f x*) ∗ *indicator* {*a*..*a* + *real i*} *x*
let *?fR* = λ*x*. *ennreal* (*f x*) ∗ *indicator* {*a* ..} *x*

have *F-mono*: *a* ≤ *x* ⟹ *x* ≤ *y* ⟹ *F x* ≤ *F y* **for** *x y*
using *f nonneg* **by** (*intro DERIV-nonneg-imp-nondecreasing*[*of x y F*]) (*auto intro*: *order-trans*)
then have *F-le-T*: *a* ≤ *x* ⟹ *F x* ≤ *T* **for** *x*
by (*intro tendsto-le-const*[*OF - lim*])
(*auto simp*: *trivial-limit-at-top-linorder eventually-at-top-linorder*)

have (*SUP i*::*nat*. *?f i x*) = *?fR x* **for** *x*
proof (*rule LIMSEQ-unique*[*OF LIMSEQ-SUP*])
from *reals-Archimedean2*[*of x* − *a*] **guess** *n* **..**
then have *eventually* (λ*n*. *?f n x* = *?fR x*) *sequentially*
by (*auto intro*!: *eventually-sequentiallyI*[**where** *c*=*n*] *split*: *split-indicator*)
then show (λ*n*. *?f n x*) ⟶ *?fR x*
by (*rule Lim-eventually*)
qed (*auto simp*: *nonneg incseq-def le-fun-def split*: *split-indicator*)
then have *integral*^{N} *lborel ?fR* = (∫$^+$ *x*. (*SUP i*::*nat*. *?f i x*) ∂*lborel*)
by *simp*
also have ... = (*SUP i*::*nat*. (∫$^+$ *x*. *?f i x* ∂*lborel*))
proof (*rule nn-integral-monotone-convergence-SUP*)
show *incseq ?f*
using *nonneg* **by** (*auto simp*: *incseq-def le-fun-def split*: *split-indicator*)
show ⋀*i*. (*?f i*) ∈ *borel-measurable lborel*
using *f-borel* **by** *auto*
qed
also have ... = (*SUP i*::*nat*. *ennreal* (*F* (*a* + *real i*) − *F a*))
by (*subst nn-integral-FTC-Icc*[*OF f-borel f nonneg*]) *auto*
also have ... = *T* − *F a*
proof (*rule LIMSEQ-unique*[*OF LIMSEQ-SUP*])
have (λ*x*. *F* (*a* + *real x*)) ⟶ *T*
apply (*rule filterlim-compose*[*OF lim filterlim-tendsto-add-at-top*])
apply (*rule LIMSEQ-const-iff*[*THEN iffD2*, *OF refl*])
apply (*rule filterlim-real-sequentially*)
done
then show (λ*n*. *ennreal* (*F* (*a* + *real n*) − *F a*)) ⟶ *ennreal* (*T* − *F a*)
by (*simp add*: *F-mono F-le-T tendsto-diff*)
qed (*auto simp*: *incseq-def intro*!: *ennreal-le-iff*[*THEN iffD2*] *F-mono*)
finally show *?thesis* **.**
qed

lemma *integral-power*:
a ≤ *b* ⟹ (∫ *x*. *x*^*k* ∗ *indicator* {*a*..*b*} *x* ∂*lborel*) = (*b*^*Suc k* − *a*^*Suc k*) / *Suc k*
proof (*subst integral-FTC-Icc-real*)
fix *x* **show** *DERIV* (λ*x*. *x*^*Suc k* / *Suc k*) *x* :> *x*^*k*
by (*intro derivative-eq-intros*) *auto*

qed (*auto simp*: *field-simps simp del*: *of-nat-Suc*)

10.6 Integration by parts

lemma *integral-by-parts-integrable*:
fixes *f g F G*::*real* ⇒ *real*
assumes *a* ≤ *b*
assumes *cont-f*[*intro*]: !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont f x*
assumes *cont-g*[*intro*]: !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont g x*
assumes [*intro*]: !!*x*. *DERIV F x* :> *f x*
assumes [*intro*]: !!*x*. *DERIV G x* :> *g x*
shows *integrable lborel* (λ*x*.((*F x*) ∗ (*g x*) + (*f x*) ∗ (*G x*)) ∗ *indicator* {*a* .. *b*} *x*)
by (*auto intro*!: *borel-integrable-atLeastAtMost continuous-intros*) (*auto intro*!: *DERIV-isCont*)

lemma *integral-by-parts*:
fixes *f g F G*::*real* ⇒ *real*
assumes [*arith*]: *a* ≤ *b*
assumes *cont-f*[*intro*]: !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont f x*
assumes *cont-g*[*intro*]: !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont g x*
assumes [*intro*]: !!*x*. *DERIV F x* :> *f x*
assumes [*intro*]: !!*x*. *DERIV G x* :> *g x*
shows (∫ *x*. (*F x* ∗ *g x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*)
= *F b* ∗ *G b* − *F a* ∗ *G a* − ∫ *x*. (*f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*
proof−
have *0*: (∫ *x*. (*F x* ∗ *g x* + *f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) = *F b* ∗ *G b* − *F a* ∗ *G a*
by (*rule integral-FTC-Icc-real*, *auto intro*!: *derivative-eq-intros continuous-intros*) (*auto intro*!: *DERIV-isCont*)

have (∫ *x*. (*F x* ∗ *g x* + *f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) =
(∫ *x*. (*F x* ∗ *g x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) + ∫ *x*. (*f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*
apply (*subst integral-add*[*symmetric*])
apply (*auto intro*!: *borel-integrable-atLeastAtMost continuous-intros*)
by (*auto intro*!: *DERIV-isCont integral-cong split*:*split-indicator*)

thus *?thesis* **using** *0* **by** *auto*
qed

lemma *integral-by-parts'*:
fixes *f g F G*::*real* ⇒ *real*
assumes *a* ≤ *b*
assumes !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont f x*
assumes !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont g x*
assumes !!*x*. *DERIV F x* :> *f x*
assumes !!*x*. *DERIV G x* :> *g x*

shows $(\int x.\ \mathit{indicator}\ \{a\ ..\ b\}\ x *_R (F\ x * g\ x)\ \partial \mathit{lborel})$
$= F\ b * G\ b - F\ a * G\ a - \int x.\ \mathit{indicator}\ \{a\ ..\ b\}\ x *_R (f\ x * G\ x)\ \partial \mathit{lborel}$
using *integral-by-parts*[*OF assms*] **by** (*simp add*: *ac-simps*)

lemma *has-bochner-integral-even-function*:
fixes $f :: \mathit{real} \Rightarrow 'a :: \{\mathit{banach}, \mathit{second\text{-}countable\text{-}topology}\}$
assumes *f*: *has-bochner-integral lborel* $(\lambda x.\ \mathit{indicator}\ \{0..\}\ x *_R f\ x)\ x$
assumes *even*: $\bigwedge x.\ f\ (-\ x) = f\ x$
shows *has-bochner-integral lborel f* $(2 *_R x)$
proof −
have *indicator*: $\bigwedge x$::*real*. *indicator* $\{..0\}\ (-\ x) = \mathit{indicator}\ \{0..\}\ x$
by (*auto split*: *split-indicator*)
have *has-bochner-integral lborel* $(\lambda x.\ \mathit{indicator}\ \{..\ 0\}\ x *_R f\ x)\ x$
by (*subst lborel-has-bochner-integral-real-affine-iff*[**where** *c*=−*1* **and** *t*=*0*])
(*auto simp*: *indicator even f*)
with *f* **have** *has-bochner-integral lborel* $(\lambda x.\ \mathit{indicator}\ \{0..\}\ x *_R f\ x + \mathit{indicator}\ \{..\ 0\}\ x *_R f\ x)\ (x + x)$
by (*rule has-bochner-integral-add*)
then have *has-bochner-integral lborel f* $(x + x)$
by (*rule has-bochner-integral-discrete-difference*[**where** *X*={*0*}, *THEN iffD1*, *rotated 4*])
(*auto split*: *split-indicator*)
then show *?thesis*
by (*simp add*: *scaleR-2*)
qed

lemma *has-bochner-integral-odd-function*:
fixes $f :: \mathit{real} \Rightarrow 'a :: \{\mathit{banach}, \mathit{second\text{-}countable\text{-}topology}\}$
assumes *f*: *has-bochner-integral lborel* $(\lambda x.\ \mathit{indicator}\ \{0..\}\ x *_R f\ x)\ x$
assumes *odd*: $\bigwedge x.\ f\ (-\ x) = -\ f\ x$
shows *has-bochner-integral lborel f 0*
proof −
have *indicator*: $\bigwedge x$::*real*. *indicator* $\{..0\}\ (-\ x) = \mathit{indicator}\ \{0..\}\ x$
by (*auto split*: *split-indicator*)
have *has-bochner-integral lborel* $(\lambda x.\ -\ \mathit{indicator}\ \{..\ 0\}\ x *_R f\ x)\ x$
by (*subst lborel-has-bochner-integral-real-affine-iff*[**where** *c*=−*1* **and** *t*=*0*])
(*auto simp*: *indicator odd f*)
from *has-bochner-integral-minus*[*OF this*]
have *has-bochner-integral lborel* $(\lambda x.\ \mathit{indicator}\ \{..\ 0\}\ x *_R f\ x)\ (-\ x)$
by *simp*
with *f* **have** *has-bochner-integral lborel* $(\lambda x.\ \mathit{indicator}\ \{0..\}\ x *_R f\ x + \mathit{indicator}\ \{..\ 0\}\ x *_R f\ x)\ (x + -\ x)$
by (*rule has-bochner-integral-add*)
then have *has-bochner-integral lborel f* $(x + -\ x)$
by (*rule has-bochner-integral-discrete-difference*[**where** *X*={*0*}, *THEN iffD1*, *rotated 4*])
(*auto split*: *split-indicator*)
then show *?thesis*

by *simp*
qed

end

11 Radon-Nikodým derivative

theory *Radon-Nikodym*
imports *Bochner-Integration*
begin

definition *diff-measure M N* =
measure-of (*space M*) (*sets M*) (λ*A*. *emeasure M A* − *emeasure N A*)

lemma
shows *space-diff-measure*[*simp*]: *space* (*diff-measure M N*) = *space M*
and *sets-diff-measure*[*simp*]: *sets* (*diff-measure M N*) = *sets M*
by (*auto simp*: *diff-measure-def*)

lemma *emeasure-diff-measure*:
assumes *fin*: *finite-measure M finite-measure N* **and** *sets-eq*: *sets M* = *sets N*
assumes *pos*: ⋀*A*. *A* ∈ *sets M* ⟹ *emeasure N A* ≤ *emeasure M A* **and** *A*: *A* ∈ *sets M*
shows *emeasure* (*diff-measure M N*) *A* = *emeasure M A* − *emeasure N A* (**is** - = *?μ A*)
unfolding *diff-measure-def*
proof (*rule emeasure-measure-of-sigma*)
show *sigma-algebra* (*space M*) (*sets M*) **..**
show *positive* (*sets M*) *?μ*
using *pos* **by** (*simp add*: *positive-def*)
show *countably-additive* (*sets M*) *?μ*
proof (*rule countably-additiveI*)
fix *A* :: *nat* ⇒ - **assume** *A*: *range A* ⊆ *sets M* **and** *disjoint-family A*
then have *suminf*:
(∑ *i*. *emeasure M* (*A i*)) = *emeasure M* (⋃ *i*. *A i*)
(∑ *i*. *emeasure N* (*A i*)) = *emeasure N* (⋃ *i*. *A i*)
by (*simp-all add*: *suminf-emeasure sets-eq*)
with *A* **have** (∑ *i*. *emeasure M* (*A i*) − *emeasure N* (*A i*)) =
(∑ *i*. *emeasure M* (*A i*)) − (∑ *i*. *emeasure N* (*A i*))
using *fin pos*[*of A* -]
by (*intro ennreal-suminf-minus*)
(*auto simp*: *sets-eq finite-measure.emeasure-eq-measure suminf-emeasure measure-nonneg*)
then show (∑ *i*. *emeasure M* (*A i*) − *emeasure N* (*A i*)) =
emeasure M (⋃ *i*. *A i*) − *emeasure N* (⋃ *i*. *A i*)
by (*simp add*: *suminf*)
qed
qed *fact*

lemma (**in** *sigma-finite-measure*) *Ex-finite-integrable-function*:
$\exists h{\in}$*borel-measurable M*. *integral*^{N} *M h* $\neq \infty \wedge (\forall x{\in}$*space M*. *0* $< h\ x \wedge h\ x < \infty)$
proof −
obtain *A* :: *nat* $\Rightarrow$ *'a set* **where**
range[*measurable*]: *range A* $\subseteq$ *sets M* **and**
space: $(\bigcup i.\ A\ i) =$ *space M* **and**
measure: $\bigwedge i.$ *emeasure M* $(A\ i) \neq \infty$ **and**
disjoint: *disjoint-family A*
using *sigma-finite-disjoint* **by** *blast*
let *?B* $= \lambda i.$ *2^Suc i* $*$ *emeasure M* $(A\ i)$
have $\forall i.\ \exists x.\ 0 < x \wedge x <$ *inverse* $(?B\ i)$
proof
fix *i* **show** $\exists x.\ 0 < x \wedge x <$ *inverse* $(?B\ i)$
using *measure*[*of i*]
by (*auto intro*!: *dense simp*: *ennreal-inverse-positive ennreal-mult-eq-top-iff power-eq-top-ennreal*)
qed
from *choice*[*OF this*] **obtain** *n* **where** *n*: $\bigwedge i.$ *0* $<$ *n i*
$\bigwedge i.$ *n i* $<$ *inverse* $(2\hat{}Suc\ i * emeasure\ M\ (A\ i))$ **by** *auto*
{ fix *i* **have** *0* $\leq$ *n i* **using** *n*(*1*)[*of i*] **by** *auto* **} note** *pos* $=$ *this*
let *?h* $= \lambda x.\ \sum i.$ *n i* $*$ *indicator* $(A\ i)\ x$
show *?thesis*
proof (*safe intro*!: *bexI*[*of* - *?h*] *del*: *notI*)
have $\bigwedge i.$ *A i* $\in$ *sets M*
using *range* **by** *fastforce*+
then have *integral*^{N} *M ?h* $= (\sum i.$ *n i* $*$ *emeasure M* $(A\ i))$ **using** *pos*
by (*simp add*: *nn-integral-suminf nn-integral-cmult-indicator*)
also have ... $\leq (\sum i.$ *ennreal* $((1/2)\hat{}Suc\ i))$
proof (*intro suminf-le allI*)
fix *N*
have *n N* $*$ *emeasure M* $(A\ N) \leq$ *inverse* $(2\hat{}Suc\ N *$ *emeasure M* $(A\ N)) *$ *emeasure M* $(A\ N)$
using *n*[*of N*] **by** (*intro mult-right-mono*) *auto*
also have ... $=$ $(1/2)\hat{}Suc\ N *$ (*inverse* (*emeasure M* $(A\ N)) *$ *emeasure M* $(A\ N))$
using *measure*[*of N*]
by (*simp add*: *ennreal-inverse-power divide-ennreal-def ennreal-inverse-mult power-eq-top-ennreal less-top*[*symmetric*] *mult-ac*
del: *power-Suc*)
also have ... $\leq$ *inverse* (*ennreal 2*) $\hat{}$ *Suc N*
using *measure*[*of N*]
apply (*cases emeasure M* $(A\ N)$ *rule*: *ennreal-cases*)
apply (*cases emeasure M* $(A\ N) = 0$)
apply (*auto simp*: *inverse-ennreal ennreal-mult*[*symmetric*] *divide-ennreal-def simp del*: *power-Suc*)
done
also have ... $=$ *ennreal* (*inverse 2* $\hat{}$ *Suc N*)
by (*subst ennreal-power*[*symmetric*], *simp*) (*simp add*: *inverse-ennreal*)

finally show $n\ N * emeasure\ M\ (A\ N) \leq ennreal\ ((1/2)\hat{}Suc\ N)$
by *simp*
qed *auto*
also have $\ldots < top$
unfolding *less-top*[*symmetric*]
apply (*rule ennreal-suminf-neq-top*)
apply (*subst summable-Suc-iff*)
apply (*subst summable-geometric*)
apply *auto*
done
finally show $integral^N\ M\ ?h \neq \infty$
by (*auto simp*: *top-unique*)
next
{ fix x **assume** $x \in space\ M$
then obtain i **where** $x \in A\ i$ **using** *space*[*symmetric*] **by** *auto*
with *disjoint n* **have** $?h\ x = n\ i$
by (*auto intro*!: *suminf-cmult-indicator intro*: *less-imp-le*)
then show $0 < ?h\ x$ **and** $?h\ x < \infty$ **using** n[*of i*] **by** (*auto simp*: *less-top*[*symmetric*]) **}**
note *pos* = *this*
qed *measurable*
qed

11.1 Absolutely continuous

definition *absolutely-continuous* :: *'a measure* ⇒ *'a measure* ⇒ *bool* **where**
absolutely-continuous M N ⟷ *null-sets M* ⊆ *null-sets N*

lemma *absolutely-continuousI-count-space*: *absolutely-continuous* (*count-space A*) *M*
unfolding *absolutely-continuous-def* **by** (*auto simp*: *null-sets-count-space*)

lemma *absolutely-continuousI-density*:
$f \in borel\text{-}measurable\ M \Longrightarrow absolutely\text{-}continuous\ M\ (density\ M\ f)$
by (*force simp add*: *absolutely-continuous-def null-sets-density-iff dest*: *AE-not-in*)

lemma *absolutely-continuousI-point-measure-finite*:
$(\bigwedge x.\ [\![\ x \in A\ ;\ f\ x \leq 0\ ]\!] \Longrightarrow g\ x \leq 0) \Longrightarrow absolutely\text{-}continuous\ (point\text{-}measure\ A\ f)\ (point\text{-}measure\ A\ g)$
unfolding *absolutely-continuous-def* **by** (*force simp*: *null-sets-point-measure-iff*)

lemma *absolutely-continuous-AE*:
assumes *sets-eq*: *sets M'* = *sets M*
and *absolutely-continuous M M' AE x in M*. *P x*
shows *AE x in M'*. *P x*
proof −
from ‹*AE x in M*. *P x*› **obtain** N **where** N: $N \in null\text{-}sets\ M$ $\{x \in space\ M.\ \neg P\ x\} \subseteq N$
unfolding *eventually-ae-filter* **by** *auto*

show *AE x in M'. P x*
proof (*rule AE-I'*)
show {*x*∈*space M'*. ¬ *P x*} ⊆ *N* **using** *sets-eq-imp-space-eq*[*OF sets-eq*] *N*(*2*)
by *simp*
from ‹*absolutely-continuous M M'*› **show** *N* ∈ *null-sets M'*
using *N* **unfolding** *absolutely-continuous-def sets-eq null-sets-def* **by** *auto*
qed
qed

11.2 Existence of the Radon-Nikodym derivative

lemma (**in** *finite-measure*) *Radon-Nikodym-aux-epsilon*:
fixes *e* :: *real* **assumes** *0* < *e*
assumes *finite-measure N* **and** *sets-eq*: *sets N* = *sets M*
shows ∃ *A*∈*sets M*. *measure M* (*space M*) − *measure N* (*space M*) ≤ *measure M A* − *measure N A* ∧
(∀ *B*∈*sets M*. *B* ⊆ *A* ⟶ − *e* < *measure M B* − *measure N B*)
proof −
interpret *M'*: *finite-measure N* **by** *fact*
let *?d* = λ*A*. *measure M A* − *measure N A*
let *?A* = λ*A*. *if* (∀ *B*∈*sets M*. *B* ⊆ *space M* − *A* ⟶ −*e* < *?d B*)
then {}
else (*SOME B*. *B* ∈ *sets M* ∧ *B* ⊆ *space M* − *A* ∧ *?d B* ≤ −*e*)
def *A* ≡ λ*n*. ((λ*B*. *B* ∪ *?A B*) ^^ *n*) {}
have *A-simps*[*simp*]:
A 0 = {}
⋀*n*. *A* (*Suc n*) = (*A n* ∪ *?A* (*A n*)) **unfolding** *A-def* **by** *simp-all*
{ fix *A* **assume** *A* ∈ *sets M*
have *?A A* ∈ *sets M*
by (*auto intro*!: *someI2*[*of* - - λ*A*. *A* ∈ *sets M*] *simp*: *not-less*) **}**
note *A'-in-sets* = *this*
{ fix *n* **have** *A n* ∈ *sets M*
proof (*induct n*)
case (*Suc n*) **thus** *A* (*Suc n*) ∈ *sets M*
using *A'-in-sets*[*of A n*] **by** (*auto split*: *if-split-asm*)
qed (*simp add*: *A-def*) **}**
note *A-in-sets* = *this*
hence *range A* ⊆ *sets M* **by** *auto*
{ fix *n B*
assume *Ex*: ∃ *B*. *B* ∈ *sets M* ∧ *B* ⊆ *space M* − *A n* ∧ *?d B* ≤ −*e*
hence *False*: ¬ (∀ *B*∈*sets M*. *B* ⊆ *space M* − *A n* ⟶ −*e* < *?d B*) **by** (*auto simp*: *not-less*)
have *?d* (*A* (*Suc n*)) ≤ *?d* (*A n*) − *e* **unfolding** *A-simps if-not-P*[*OF False*]
proof (*rule someI2-ex*[*OF Ex*])
fix *B* **assume** *B* ∈ *sets M* ∧ *B* ⊆ *space M* − *A n* ∧ *?d B* ≤ − *e*
hence *A n* ∩ *B* = {} *B* ∈ *sets M* **and** *dB*: *?d B* ≤ −*e* **by** *auto*
hence *?d* (*A n* ∪ *B*) = *?d* (*A n*) + *?d B*
using ‹*A n* ∈ *sets M*› *finite-measure-Union M'.finite-measure-Union* **by** (*simp add*: *sets-eq*)

```
    also have ... ≤ ?d (A n) − e using dB by simp
    finally show ?d (A n ∪ B) ≤ ?d (A n) − e .
  qed }
 note dA-epsilon = this
 { fix n have ?d (A (Suc n)) ≤ ?d (A n)
  proof (cases ∃ B. B∈sets M ∧ B ⊆ space M − A n ∧ ?d B ≤ − e)
    case True from dA-epsilon[OF this] show ?thesis using ‹0 < e› by simp
  next
    case False
     hence ∀ B∈sets M. B ⊆ space M − A n ⟶ −e < ?d B by (auto simp:
not-le)
    thus ?thesis by simp
  qed }
 note dA-mono = this
 show ?thesis
 proof (cases ∃ n. ∀ B∈sets M. B ⊆ space M − A n ⟶ −e < ?d B)
   case True then obtain n where B: ⋀B. ⟦ B ∈ sets M; B ⊆ space M − A
n⟧ ⟹ −e < ?d B by blast
   show ?thesis
   proof (safe intro!: bexI[of - space M − A n])
    fix B assume B ∈ sets M B ⊆ space M − A n
    from B[OF this] show −e < ?d B .
   next
    show space M − A n ∈ sets M by (rule sets.compl-sets) fact
   next
    show ?d (space M) ≤ ?d (space M − A n)
    proof (induct n)
     fix n assume ?d (space M) ≤ ?d (space M − A n)
     also have ... ≤ ?d (space M − A (Suc n))
       using A-in-sets sets.sets-into-space dA-mono[of n] finite-measure-compl
M'.finite-measure-compl
      by (simp del: A-simps add: sets-eq sets-eq-imp-space-eq[OF sets-eq])
     finally show ?d (space M) ≤ ?d (space M − A (Suc n)) .
    qed simp
   qed
 next
   case False hence B: ⋀n. ∃ B. B∈sets M ∧ B ⊆ space M − A n ∧ ?d B ≤ −
e
    by (auto simp add: not-less)
   { fix n have ?d (A n) ≤ − real n * e
    proof (induct n)
      case (Suc n) with dA-epsilon[of n, OF B] show ?case by (simp del:
A-simps add: of-nat-Suc field-simps)
    next
     case 0 with measure-empty show ?case by (simp add: zero-ennreal-def)
    qed } note dA-less = this
   have decseq: decseq (λn. ?d (A n)) unfolding decseq-eq-incseq
   proof (rule incseq-SucI)
    fix n show − ?d (A n) ≤ − ?d (A (Suc n)) using dA-mono[of n] by auto
```

```
    qed
    have A: incseq A by (auto intro!: incseq-SucI)
    from finite-Lim-measure-incseq[OF - A] ‹range A ⊆ sets M›
      M'.finite-Lim-measure-incseq[OF - A]
    have convergent: (λi. ?d (A i)) ⟶ ?d (⋃ i. A i)
      by (auto intro!: tendsto-diff simp: sets-eq)
    obtain n :: nat where - ?d (⋃ i. A i) / e < real n using reals-Archimedean2
by auto
    moreover from order-trans[OF decseq-le[OF decseq convergent] dA-less]
    have real n ≤ - ?d (⋃ i. A i) / e using ‹0<e› by (simp add: field-simps)
    ultimately show ?thesis by auto
  qed
qed

lemma (in finite-measure) Radon-Nikodym-aux:
  assumes finite-measure N and sets-eq: sets N = sets M
  shows ∃ A∈sets M. measure M (space M) - measure N (space M) ≤
                    measure M A - measure N A ∧
                    (∀ B∈sets M. B ⊆ A ⟶ 0 ≤ measure M B - measure N B)
proof -
  interpret N: finite-measure N by fact
  let ?d = λA. measure M A - measure N A
  let ?P = λA n. if n = 0 then A = space M else (∀ C∈sets M. C ⊆ A ⟶ - 1
/ real (Suc n) < ?d C)
  let ?Q = λA B. A ⊆ B ∧ ?d B ≤ ?d A

  have ∃ A. ∀ n. (A n ∈ sets M ∧ ?P (A n) n) ∧ ?Q (A (Suc n)) (A n)
  proof (rule dependent-nat-choice)
    show ∃ A. A ∈ sets M ∧ ?P A 0
      by auto
  next
    fix A n assume A ∈ sets M ∧ ?P A n
    then have A: A ∈ sets M by auto
     then have finite-measure (density M (indicator A)) 0 < 1 / real (Suc (Suc
n))
          finite-measure (density N (indicator A)) sets (density N (indicator A)) =
sets (density M (indicator A))
      by (auto simp: finite-measure-restricted N.finite-measure-restricted sets-eq)
    from finite-measure.Radon-Nikodym-aux-epsilon[OF this] guess X .. note X
= this
    with A have A ∩ X ∈ sets M ∧ ?P (A ∩ X) (Suc n) ∧ ?Q (A ∩ X) A
      by (simp add: measure-restricted sets-eq sets.Int) (metis inf-absorb2)
    then show ∃ B. (B ∈ sets M ∧ ?P B (Suc n)) ∧ ?Q B A
      by blast
  qed
  then obtain A where A: ⋀n. A n ∈ sets M ⋀n. ?P (A n) n ⋀n. ?Q (A (Suc
n)) (A n)
    by metis
  then have mono-dA: mono (λi. ?d (A i)) and A-0[simp]: A 0 = space M
```

```
    by (auto simp add: mono-iff-le-Suc)
  show ?thesis
  proof (safe intro!: bexI[of - ⋂i. A i])
    show (⋂i. A i) ∈ sets M using ‹⋀n. A n ∈ sets M› by auto
    have decseq A using A by (auto intro!: decseq-SucI)
   from A(1) finite-Lim-measure-decseq[OF - this] N.finite-Lim-measure-decseq[OF
- this]
    have (λi. ?d (A i)) ⟶ ?d (⋂i. A i) by (auto intro!: tendsto-diff simp:
sets-eq)
    thus ?d (space M) ≤ ?d (⋂i. A i) using mono-dA[THEN monoD, of 0 -]
      by (rule-tac LIMSEQ-le-const) auto
  next
    fix B assume B: B ∈ sets M B ⊆ (⋂i. A i)
    show 0 ≤ ?d B
    proof (rule ccontr)
      assume ¬ 0 ≤ ?d B
      hence 0 < - ?d B by auto
      from reals-Archimedean[OF this]
      obtain n where ∗: ?d B < - 1 / real (Suc n)
        by (auto simp: field-simps)
      also have ... ≤ - 1 / real (Suc (Suc n))
        by (auto simp: field-simps)
      finally show False
        using ∗ A(2)[of Suc n] B by (auto elim!: ballE[of - - B])
    qed
  qed
qed

lemma (in finite-measure) Radon-Nikodym-finite-measure:
  assumes finite-measure N and sets-eq: sets N = sets M
  assumes absolutely-continuous M N
  shows ∃f ∈ borel-measurable M. (∀x. 0 ≤ f x) ∧ density M f = N
proof −
  interpret N: finite-measure N by fact
  def G ≡ {g ∈ borel-measurable M. (∀x. 0 ≤ g x) ∧ (∀A∈sets M. (∫⁺x. g x ∗
indicator A x ∂M) ≤ N A)}
  { fix f have f ∈ G ⟹ f ∈ borel-measurable M by (auto simp: G-def) }
  note this[measurable-dest]
  have (λx. 0) ∈ G unfolding G-def by auto
  hence G ≠ {} by auto
  { fix f g assume f: f ∈ G and g: g ∈ G
    have (λx. max (g x) (f x)) ∈ G (is ?max ∈ G) unfolding G-def
    proof safe
      show ?max ∈ borel-measurable M using f g unfolding G-def by auto
      let ?A = {x ∈ space M. f x ≤ g x}
      have ?A ∈ sets M using f g unfolding G-def by auto
      fix A assume A ∈ sets M
      hence sets: ?A ∩ A ∈ sets M (space M − ?A) ∩ A ∈ sets M using ‹?A ∈
sets M› by auto
```

hence *sets'*: *?A* ∩ *A* ∈ *sets N* (*space M* − *?A*) ∩ *A* ∈ *sets N* **by** (*auto simp*: *sets-eq*)
have *union*: ((*?A* ∩ *A*) ∪ ((*space M* − *?A*) ∩ *A*)) = *A*
using *sets.sets-into-space*[*OF* ‹*A* ∈ *sets M*›] **by** *auto*
have ⋀*x*. *x* ∈ *space M* ⟹ *max* (*g x*) (*f x*) ∗ *indicator A x* =
g x ∗ *indicator* (*?A* ∩ *A*) *x* + *f x* ∗ *indicator* ((*space M* − *?A*) ∩ *A*) *x*
by (*auto simp*: *indicator-def max-def*)
hence (∫⁺*x*. *max* (*g x*) (*f x*) ∗ *indicator A x* ∂*M*) =
(∫⁺*x*. *g x* ∗ *indicator* (*?A* ∩ *A*) *x* ∂*M*) +
(∫⁺*x*. *f x* ∗ *indicator* ((*space M* − *?A*) ∩ *A*) *x* ∂*M*)
using *f g sets* **unfolding** *G-def*
by (*auto cong*: *nn-integral-cong intro*!: *nn-integral-add*)
also have ... ≤ *N* (*?A* ∩ *A*) + *N* ((*space M* − *?A*) ∩ *A*)
using *f g sets* **unfolding** *G-def* **by** (*auto intro*!: *add-mono*)
also have ... = *N A*
using *plus-emeasure*[*OF sets'*] *union* **by** *auto*
finally show (∫⁺*x*. *max* (*g x*) (*f x*) ∗ *indicator A x* ∂*M*) ≤ *N A* **.**
next
fix *x* **show** *0* ≤ *max* (*g x*) (*f x*) **using** *f g* **by** (*auto simp*: *G-def split*: *split-max*)
qed }
note *max-in-G* = *this*
{ **fix** *f* **assume** *incseq f* **and** *f*: ⋀*i*. *f i* ∈ *G*
then have [*measurable*]: ⋀*i*. *f i* ∈ *borel-measurable M* **by** (*auto simp*: *G-def*)
have (λ*x*. *SUP i*. *f i x*) ∈ *G* **unfolding** *G-def*
proof *safe*
show (λ*x*. *SUP i*. *f i x*) ∈ *borel-measurable M* **by** *measurable*
{ **fix** *x* **show** *0* ≤ (*SUP i*. *f i x*)
using *f* **by** (*auto simp*: *G-def intro*: *SUP-upper2*) }
next
fix *A* **assume** *A* ∈ *sets M*
have (∫⁺*x*. (*SUP i*. *f i x*) ∗ *indicator A x* ∂*M*) =
(∫⁺*x*. (*SUP i*. *f i x* ∗ *indicator A x*) ∂*M*)
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also have ... = (*SUP i*. (∫⁺*x*. *f i x* ∗ *indicator A x* ∂*M*))
using ‹*incseq f*› *f* ‹*A* ∈ *sets M*›
by (*intro nn-integral-monotone-convergence-SUP*)
(*auto simp*: *G-def incseq-Suc-iff le-fun-def split*: *split-indicator*)
finally show (∫⁺*x*. (*SUP i*. *f i x*) ∗ *indicator A x* ∂*M*) ≤ *N A*
using *f* ‹*A* ∈ *sets M*› **by** (*auto intro*!: *SUP-least simp*: *G-def*)
qed }
note *SUP-in-G* = *this*
let *?y* = *SUP g* : *G*. *integral*^{N} *M g*
have *y-le*: *?y* ≤ *N* (*space M*) **unfolding** *G-def*
proof (*safe intro*!: *SUP-least*)
fix *g* **assume** ∀ *A*∈*sets M*. (∫⁺*x*. *g x* ∗ *indicator A x* ∂*M*) ≤ *N A*
from *this*[*THEN bspec*, *OF sets.top*] **show** *integral*^{N} *M g* ≤ *N* (*space M*)
by (*simp cong*: *nn-integral-cong*)
qed

from *ennreal-SUP-countable-SUP* [*OF* ‹$G \neq \{\}$›, *of* $integral^N\ M$] **guess** *ys* **..**
note $ys = this$
then have $\forall n.\ \exists g.\ g \in G \wedge integral^N\ M\ g = ys\ n$
proof *safe*
fix *n* **assume** $range\ ys \subseteq integral^N\ M$ ‘ G
hence $ys\ n \in integral^N\ M$ ‘ G **by** *auto*
thus $\exists g.\ g \in G \wedge integral^N\ M\ g = ys\ n$ **by** *auto*
qed
from *choice*[*OF this*] **obtain** *gs* **where** $\bigwedge i.\ gs\ i \in G\ \bigwedge n.\ integral^N\ M\ (gs\ n) = ys\ n$ **by** *auto*
hence *y-eq*: $?y = (SUP\ i.\ integral^N\ M\ (gs\ i))$ **using** *ys* **by** *auto*
let $?g = \lambda i\ x.\ Max\ ((\lambda n.\ gs\ n\ x)$ ‘ $\{..i\})$
def $f \equiv \lambda x.\ SUP\ i.\ ?g\ i\ x$
let $?F = \lambda A\ x.\ f\ x * indicator\ A\ x$
have *gs-not-empty*: $\bigwedge i\ x.\ (\lambda n.\ gs\ n\ x)$ ‘ $\{..i\} \neq \{\}$ **by** *auto*
{ fix *i* **have** $?g\ i \in G$
proof (*induct i*)
case *0* **thus** *?case* **by** *simp fact*
next
case (*Suc i*)
with *Suc gs-not-empty* ‹$gs\ (Suc\ i) \in G$› **show** *?case*
by (*auto simp add*: *atMost-Suc intro*!: *max-in-G*)
qed }
note *g-in-G* = *this*
have *incseq ?g* **using** *gs-not-empty*
by (*auto intro*!: *incseq-SucI le-funI simp add*: *atMost-Suc*)
from *SUP-in-G*[*OF this g-in-G*] **have** [*measurable*]: $f \in G$ **unfolding** *f-def* **.**
then have [*simp*, *intro*]: $f \in borel\text{-}measurable\ M$ **unfolding** *G-def* **by** *auto*
have $integral^N\ M\ f = (SUP\ i.\ integral^N\ M\ (?g\ i))$ **unfolding** *f-def*
using *g-in-G* ‹*incseq ?g*›
by (*auto intro*!: *nn-integral-monotone-convergence-SUP simp*: *G-def*)
also have $\ldots = ?y$
proof (*rule antisym*)
show $(SUP\ i.\ integral^N\ M\ (?g\ i)) \leq ?y$
using *g-in-G* **by** (*auto intro*: *SUP-mono*)
show $?y \leq (SUP\ i.\ integral^N\ M\ (?g\ i))$ **unfolding** *y-eq*
by (*auto intro*!: *SUP-mono nn-integral-mono Max-ge*)
qed
finally have *int-f-eq-y*: $integral^N\ M\ f = ?y$ **.**
have $\bigwedge x.\ 0 \leq f\ x$
unfolding *f-def* **using** ‹$\bigwedge i.\ gs\ i \in G$›
by (*auto intro*!: *SUP-upper2 Max-ge-iff*[*THEN iffD2*] *simp*: *G-def*)
let $?t = \lambda A.\ N\ A - (\int^+ x.\ ?F\ A\ x\ \partial M)$
let $?M = diff\text{-}measure\ N\ (density\ M\ f)$
have *f-le-N*: $\bigwedge A.\ A \in sets\ M \Longrightarrow (\int^+ x.\ ?F\ A\ x\ \partial M) \leq N\ A$
using ‹$f \in G$› **unfolding** *G-def* **by** *auto*
have *emeasure-M*: $\bigwedge A.\ A \in sets\ M \Longrightarrow emeasure\ ?M\ A = ?t\ A$
proof (*subst emeasure-diff-measure*)
from *f-le-N*[*of space M*] **show** *finite-measure N finite-measure* (*density M f*)

by (*auto intro*!: *finite-measureI simp*: *emeasure-density top-unique cong*: *nn-integral-cong*)
next
fix *B* **assume** *B* ∈ *sets N* **with** *f-le-N*[*of B*] **show** *emeasure* (*density M f*) *B* ≤ *emeasure N B*
by (*auto simp*: *sets-eq emeasure-density cong*: *nn-integral-cong*)
qed (*auto simp*: *sets-eq emeasure-density*)
from *emeasure-M*[*of space M*] *N*.*finite-emeasure-space*
interpret *M'*: *finite-measure ?M*
by (*auto intro*!: *finite-measureI simp*: *sets-eq-imp-space-eq*[*OF sets-eq*] *N*.*emeasure-eq-measure*)

have *ac*: *absolutely-continuous M ?M* **unfolding** *absolutely-continuous-def*
proof
fix *A* **assume** *A-M*: *A* ∈ *null-sets M*
with ‹*absolutely-continuous M N*› **have** *A-N*: *A* ∈ *null-sets N*
unfolding *absolutely-continuous-def* **by** *auto*
moreover from *A-M A-N* **have** (∫⁺ *x*. *?F A x* ∂*M*) ≤ *N A* **using** ‹*f* ∈ *G*› **by** (*auto simp*: *G-def*)
ultimately have *N A* − (∫⁺ *x*. *?F A x* ∂*M*) = *0*
by (*auto intro*!: *antisym*)
then show *A* ∈ *null-sets ?M*
using *A-M* **by** (*simp add*: *emeasure-M null-sets-def sets-eq*)
qed
have *upper-bound*: ∀ *A*∈*sets M*. *?M A* ≤ *0*
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *A* **where** *A*: *A* ∈ *sets M* **and** *pos*: *0* < *?M A*
by (*auto simp*: *zero-less-iff-neq-zero*)
note *pos*
also have *?M A* ≤ *?M* (*space M*)
using *emeasure-space*[*of ?M A*] **by** (*simp add*: *sets-eq*[*THEN sets-eq-imp-space-eq*])
finally have *pos-t*: *0* < *?M* (*space M*) **by** *simp*
moreover
from *pos-t* **have** *emeasure M* (*space M*) ≠ *0*
using *ac* **unfolding** *absolutely-continuous-def* **by** (*auto simp*: *null-sets-def*)
then have *pos-M*: *0* < *emeasure M* (*space M*)
by (*simp add*: *zero-less-iff-neq-zero*)
moreover
have (∫⁺*x*. *f x* ∗ *indicator* (*space M*) *x* ∂*M*) ≤ *N* (*space M*)
using ‹*f* ∈ *G*› **unfolding** *G-def* **by** *auto*
hence (∫⁺*x*. *f x* ∗ *indicator* (*space M*) *x* ∂*M*) ≠ ∞
using *M'*.*finite-emeasure-space* **by** (*auto simp*: *top-unique*)
moreover
def *b* ≡ *?M* (*space M*) / *emeasure M* (*space M*) / *2*
ultimately have *b*: *b* ≠ *0* ∧ *0* ≤ *b* ∧ *b* ≠ ∞
by (*auto simp*: *ennreal-divide-eq-top-iff*)
then have *b*: *b* ≠ *0 0* ≤ *b 0* < *b b* ≠ ∞
by (*auto simp*: *less-le*)

let *?Mb* = *density M* (λ-. *b*)
have *Mb*: *finite-measure ?Mb sets ?Mb* = *sets ?M*
using *b* **by** (*auto simp*: *emeasure-density-const sets-eq ennreal-mult-eq-top-iff intro*!: *finite-measureI*)
from *M'.Radon-Nikodym-aux*[*OF this*] **guess** *A0* **..**
then have *A0* ∈ *sets M*
and *space-le-A0*: *measure ?M* (*space M*) − *enn2real b* ∗ *measure M* (*space M*) ≤ *measure ?M A0* − *enn2real b* ∗ *measure M A0*
and ∗: ⋀*B*. *B* ∈ *sets M* ⟹ *B* ⊆ *A0* ⟹ 0 ≤ *measure ?M B* − *enn2real b* ∗ *measure M B*
using *b* **by** (*simp-all add*: *measure-density-const sets-eq-imp-space-eq*[*OF sets-eq*] *sets-eq*)
{ fix *B* **assume** *B*: *B* ∈ *sets M B* ⊆ *A0*
with ∗[*OF this*] **have** *b* ∗ *emeasure M B* ≤ *?M B*
using *b* **unfolding** *M'.emeasure-eq-measure emeasure-eq-measure*
by (*cases b rule*: *ennreal-cases*) (*auto simp*: *ennreal-mult*[*symmetric*] *measure-nonneg*) **}**
note *bM-le-t* = *this*
let *?f0* = λ*x*. *f x* + *b* ∗ *indicator A0 x*
{ fix *A* **assume** *A*: *A* ∈ *sets M*
hence *A* ∩ *A0* ∈ *sets M* **using** ‹*A0* ∈ *sets M*› **by** *auto*
have (∫⁺*x*. *?f0 x* ∗ *indicator A x* ∂*M*) =
(∫⁺*x*. *f x* ∗ *indicator A x* + *b* ∗ *indicator* (*A* ∩ *A0*) *x* ∂*M*)
by (*auto intro*!: *nn-integral-cong split*: *split-indicator*)
hence (∫⁺*x*. *?f0 x* ∗ *indicator A x* ∂*M*) =
(∫⁺*x*. *f x* ∗ *indicator A x* ∂*M*) + *b* ∗ *emeasure M* (*A* ∩ *A0*)
using ‹*A0* ∈ *sets M*› ‹*A* ∩ *A0* ∈ *sets M*› *A b* ‹*f* ∈ *G*›
by (*simp add*: *nn-integral-add nn-integral-cmult-indicator G-def*) **}**
note *f0-eq* = *this*
{ fix *A* **assume** *A*: *A* ∈ *sets M*
hence *A* ∩ *A0* ∈ *sets M* **using** ‹*A0* ∈ *sets M*› **by** *auto*
have *f-le-v*: (∫⁺*x*. *?F A x* ∂*M*) ≤ *N A* **using** ‹*f* ∈ *G*› *A* **unfolding** *G-def* **by** *auto*
note *f0-eq*[*OF A*]
also have (∫⁺*x*. *?F A x* ∂*M*) + *b* ∗ *emeasure M* (*A* ∩ *A0*) ≤ (∫⁺*x*. *?F A x* ∂*M*) + *?M* (*A* ∩ *A0*)
using *bM-le-t*[*OF* ‹*A* ∩ *A0* ∈ *sets M*›] ‹*A* ∈ *sets M*› ‹*A0* ∈ *sets M*›
by (*auto intro*!: *add-left-mono*)
also have ... ≤ (∫⁺*x*. *f x* ∗ *indicator A x* ∂*M*) + *?M A*
using *emeasure-mono*[*of A* ∩ *A0 A ?M*] ‹*A* ∈ *sets M*› ‹*A0* ∈ *sets M*›
by (*auto intro*!: *add-left-mono simp*: *sets-eq*)
also have ... ≤ *N A*
unfolding *emeasure-M*[*OF* ‹*A* ∈ *sets M*›]
using *f-le-v N.emeasure-eq-measure*[*of A*]
by (*cases* ∫⁺*x*. *?F A x* ∂*M N A rule*: *ennreal2-cases*)
(*auto simp*: *top-unique measure-nonneg ennreal-minus ennreal-plus*[*symmetric*] *simp del*: *ennreal-plus*)
finally have (∫⁺*x*. *?f0 x* ∗ *indicator A x* ∂*M*) ≤ *N A* **. }**
hence *?f0* ∈ *G* **using** ‹*A0* ∈ *sets M*› *b* ‹*f* ∈ *G*› **by** (*auto simp*: *G-def*)

have *int-f-finite*: $integral^N\ M\ f \neq \infty$
by (*metis top-unique infinity-ennreal-def int-f-eq-y y-le N.emeasure-finite*)
have *pos*: $0 < measure\ ?M\ (space\ M) - enn2real\ b * measure\ M\ (space\ M)$
using *pos-t pos-M*
by (*simp add*: *M'.emeasure-eq-measure emeasure-eq-measure b-def divide-ennreal ennreal-divide-numeral*)
also have $\ldots \leq measure\ ?M\ A0 - enn2real\ b * measure\ M\ A0$
by (*rule space-le-A0*)
finally have $enn2real\ b * measure\ M\ A0 < measure\ ?M\ A0$
by *simp*
with *b* **have** $?M\ A0 \neq 0$
by (*cases b rule*: *ennreal-cases*)
(*auto simp*: *M'.emeasure-eq-measure measure-nonneg mult-less-0-iff not-le*[*symmetric*])
then have $emeasure\ M\ A0 \neq 0$
using *ac* ‹$A0 \in sets\ M$› **by** (*auto simp*: *absolutely-continuous-def null-sets-def*)
then have $0 < emeasure\ M\ A0$
by (*auto simp*: *zero-less-iff-neq-zero*)
hence $0 < b * emeasure\ M\ A0$
using *b* **by** (*auto simp*: *ennreal-zero-less-mult-iff*)
with *int-f-finite* **have** $?y < integral^N\ M\ f + b * emeasure\ M\ A0$
unfolding *int-f-eq-y* **by** *auto*
also have $\ldots = integral^N\ M\ ?f0$
using *f0-eq*[*OF sets.top*] ‹$A0 \in sets\ M$› *sets.sets-into-space* **by** (*simp cong*: *nn-integral-cong*)
finally have $?y < integral^N\ M\ ?f0$
by *simp*
moreover have $integral^N\ M\ ?f0 \leq ?y$
using ‹$?f0 \in G$› **by** (*auto intro*!: *SUP-upper*)
ultimately show *False* **by** *auto*
qed
show *?thesis*
proof (*intro bexI*[*of - f*] *measure-eqI conjI*)
show $sets\ (density\ M\ f) = sets\ N$
by (*simp add*: *sets-eq*)
fix *A* **assume** *A*: $A \in sets\ (density\ M\ f)$
then show $emeasure\ (density\ M\ f)\ A = emeasure\ N\ A$
using ‹$f \in G$› *A upper-bound*[*THEN bspec, of A*] *N.emeasure-eq-measure*[*of A*]
by (*cases* $integral^N\ M\ (?F\ A)$)
(*auto intro*!: *antisym simp*: *emeasure-density G-def emeasure-M ennreal-minus-eq-0 top-unique*
simp del: *measure-nonneg*)
qed *auto*
qed

lemma (**in** *finite-measure*) *split-space-into-finite-sets-and-rest*:
assumes *ac*: *absolutely-continuous M N* **and** *sets-eq*: $sets\ N = sets\ M$
shows $\exists A0 \in sets\ M.\ \exists B::nat \Rightarrow 'a\ set.\ disjoint\text{-}family\ B \wedge range\ B \subseteq sets\ M \wedge A0 = space\ M - (\bigcup i.\ B\ i) \wedge$

```
   (∀ A∈sets M. A ⊆ A0 ⟶ (emeasure M A = 0 ∧ N A = 0) ∨ (emeasure M A
> 0 ∧ N A = ∞)) ∧
   (∀ i. N (B i) ≠ ∞)
proof −
  let ?Q = {Q∈sets M. N Q ≠ ∞}
  let ?a = SUP Q:?Q. emeasure M Q
  have {} ∈ ?Q by auto
  then have Q-not-empty: ?Q ≠ {} by blast
  have ?a ≤ emeasure M (space M) using sets.sets-into-space
    by (auto intro!: SUP-least emeasure-mono)
  then have ?a ≠ ∞
    using finite-emeasure-space
   by (auto simp: less-top[symmetric] top-unique simp del: SUP-eq-top-iff Sup-eq-top-iff)
  from ennreal-SUP-countable-SUP [OF Q-not-empty, of emeasure M]
  obtain Q'' where range Q'' ⊆ emeasure M ‘ ?Q and a: ?a = (SUP i::nat. Q''
i)
    by auto
  then have ∀ i. ∃ Q'. Q'' i = emeasure M Q' ∧ Q' ∈ ?Q by auto
  from choice[OF this] obtain Q' where Q': ⋀i. Q'' i = emeasure M (Q' i) ⋀i.
Q' i ∈ ?Q
    by auto
  then have a-Lim: ?a = (SUP i::nat. emeasure M (Q' i)) using a by simp
  let ?O = λn. ⋃i≤n. Q' i
  have Union: (SUP i. emeasure M (?O i)) = emeasure M (⋃i. ?O i)
  proof (rule SUP-emeasure-incseq[of ?O])
    show range ?O ⊆ sets M using Q' by auto
    show incseq ?O by (fastforce intro!: incseq-SucI)
  qed
  have Q'-sets: ⋀i. Q' i ∈ sets M using Q' by auto
  have O-sets: ⋀i. ?O i ∈ sets M using Q' by auto
  then have O-in-G: ⋀i. ?O i ∈ ?Q
  proof (safe del: notI)
    fix i have Q' ‘ {..i} ⊆ sets M using Q' by auto
    then have N (?O i) ≤ (∑ i≤i. N (Q' i))
      by (simp add: sets-eq emeasure-subadditive-finite)
    also have ... < ∞ using Q' by (simp add: less-top)
    finally show N (?O i) ≠ ∞ by simp
  qed auto
  have O-mono: ⋀n. ?O n ⊆ ?O (Suc n) by fastforce
  have a-eq: ?a = emeasure M (⋃i. ?O i) unfolding Union[symmetric]
  proof (rule antisym)
    show ?a ≤ (SUP i. emeasure M (?O i)) unfolding a-Lim
      using Q' by (auto intro!: SUP-mono emeasure-mono)
    show (SUP i. emeasure M (?O i)) ≤ ?a
    proof (safe intro!: Sup-mono, unfold bex-simps)
      fix i
      have ∗: (⋃(Q' ‘ {..i})) = ?O i by auto
      then show ∃ x. (x ∈ sets M ∧ N x ≠ ∞) ∧
        emeasure M (⋃(Q' ‘ {..i})) ≤ emeasure M x
```

```
      using O-in-G[of i] by (auto intro!: exI[of - ?O i])
    qed
  qed
  let ?O-0 = (⋃ i. ?O i)
  have ?O-0 ∈ sets M using Q' by auto
  def Q ≡ λi. case i of 0 ⇒ Q' 0 | Suc n ⇒ ?O (Suc n) − ?O n
  { fix i have Q i ∈ sets M unfolding Q-def using Q'[of 0] by (cases i) (auto
intro: O-sets) }
  note Q-sets = this
  show ?thesis
  proof (intro bexI exI conjI ballI impI allI)
    show disjoint-family Q
      by (fastforce simp: disjoint-family-on-def Q-def
        split: nat.split-asm)
    show range Q ⊆ sets M
      using Q-sets by auto
    { fix A assume A: A ∈ sets M A ⊆ space M − ?O-0
      show emeasure M A = 0 ∧ N A = 0 ∨ 0 < emeasure M A ∧ N A = ∞
      proof (rule disjCI, simp)
        assume ∗: emeasure M A = 0 ∨ N A ≠ top
        show emeasure M A = 0 ∧ N A = 0
        proof (cases emeasure M A = 0)
          case True
          with ac A have N A = 0
            unfolding absolutely-continuous-def by auto
          with True show ?thesis by simp
        next
          case False
          with ∗ have N A ≠ ∞ by auto
          with A have emeasure M ?O-0 + emeasure M A = emeasure M (?O-0 ∪
A)
            using Q' by (auto intro!: plus-emeasure sets.countable-UN)
          also have ... = (SUP i. emeasure M (?O i ∪ A))
          proof (rule SUP-emeasure-incseq[of λi. ?O i ∪ A, symmetric, simplified])
            show range (λi. ?O i ∪ A) ⊆ sets M
              using ‹N A ≠ ∞› O-sets A by auto
          qed (fastforce intro!: incseq-SucI)
          also have ... ≤ ?a
          proof (safe intro!: SUP-least)
            fix i have ?O i ∪ A ∈ ?Q
            proof (safe del: notI)
              show ?O i ∪ A ∈ sets M using O-sets A by auto
              from O-in-G[of i] have N (?O i ∪ A) ≤ N (?O i) + N A
                using emeasure-subadditive[of ?O i N A] A O-sets by (auto simp:
sets-eq)
              with O-in-G[of i] show N (?O i ∪ A) ≠ ∞
                using ‹N A ≠ ∞› by (auto simp: top-unique)
            qed
            then show emeasure M (?O i ∪ A) ≤ ?a by (rule SUP-upper)
```

```
        qed
        finally have emeasure M A = 0
       unfolding a-eq using measure-nonneg[of M A] by (simp add: emeasure-eq-measure)
        with ‹emeasure M A ≠ 0› show ?thesis by auto
      qed
    qed }
   { fix i show N (Q i) ≠ ∞
    proof (cases i)
      case 0 then show ?thesis
        unfolding Q-def using Q'[of 0] by simp
    next
      case (Suc n)
      with ‹?O n ∈ ?Q› ‹?O (Suc n) ∈ ?Q› emeasure-Diff[OF - - - O-mono, of
N n]
      show ?thesis
        by (auto simp: sets-eq Q-def)
    qed }
   show space M − ?O-0 ∈ sets M using Q'-sets by auto
   { fix j have (⋃i≤j. ?O i) = (⋃i≤j. Q i)
    proof (induct j)
      case 0 then show ?case by (simp add: Q-def)
    next
      case (Suc j)
      have eq: ⋀j. (⋃i≤j. ?O i) = (⋃i≤j. Q' i) by fastforce
      have {..j} ∪ {..Suc j} = {..Suc j} by auto
      then have (⋃i≤Suc j. Q' i) = (⋃i≤j. Q' i) ∪ Q (Suc j)
        by (simp add: UN-Un[symmetric] Q-def del: UN-Un)
      then show ?case using Suc by (auto simp add: eq atMost-Suc)
    qed }
   then have (⋃j. (⋃i≤j. ?O i)) = (⋃j. (⋃i≤j. Q i)) by simp
   then show space M − ?O-0 = space M − (⋃i. Q i) by fastforce
  qed
qed

lemma (in finite-measure) Radon-Nikodym-finite-measure-infinite:
  assumes absolutely-continuous M N and sets-eq: sets N = sets M
  shows ∃f∈borel-measurable M. (∀x. 0 ≤ f x) ∧ density M f = N
proof −
  from split-space-into-finite-sets-and-rest[OF assms]
  obtain Q0 and Q :: nat ⇒ 'a set
    where Q: disjoint-family Q range Q ⊆ sets M
    and Q0: Q0 ∈ sets M Q0 = space M − (⋃i. Q i)
    and in-Q0: ⋀A. A ∈ sets M ⟹ A ⊆ Q0 ⟹ emeasure M A = 0 ∧ N A = 0
∨ 0 < emeasure M A ∧ N A = ∞
    and Q-fin: ⋀i. N (Q i) ≠ ∞ by force
  from Q have Q-sets: ⋀i. Q i ∈ sets M by auto
  let ?N = λi. density N (indicator (Q i)) and ?M = λi. density M (indicator
(Q i))
  have ∀i. ∃f∈borel-measurable (?M i). (∀x. 0 ≤ f x) ∧ density (?M i) f = ?N i
```

proof (*intro allI finite-measure.Radon-Nikodym-finite-measure*)
fix *i*
from *Q* **show** *finite-measure* (*?M i*)
by (*auto intro*!: *finite-measureI cong*: *nn-integral-cong*
simp add: *emeasure-density subset-eq sets-eq*)
from *Q* **have** *emeasure* (*?N i*) (*space N*) = *emeasure N* (*Q i*)
by (*simp add*: *sets-eq*[*symmetric*] *emeasure-density subset-eq cong*: *nn-integral-cong*)
with *Q-fin* **show** *finite-measure* (*?N i*)
by (*auto intro*!: *finite-measureI*)
show *sets* (*?N i*) = *sets* (*?M i*) **by** (*simp add*: *sets-eq*)
have [*measurable*]: ⋀*A*. *A* ∈ *sets M* ⟹ *A* ∈ *sets N* **by** (*simp add*: *sets-eq*)
show *absolutely-continuous* (*?M i*) (*?N i*)
using ‹*absolutely-continuous M N*› ‹*Q i* ∈ *sets M*›
by (*auto simp*: *absolutely-continuous-def null-sets-density-iff sets-eq*
intro!: *absolutely-continuous-AE*[*OF sets-eq*])
qed
from *choice*[*OF this*[*unfolded Bex-def*]]
obtain *f* **where** *borel*: ⋀*i*. *f i* ∈ *borel-measurable M* ⋀*i x*. 0 ≤ *f i x*
and *f-density*: ⋀*i*. *density* (*?M i*) (*f i*) = *?N i*
by *force*
{ fix *A i* **assume** *A*: *A* ∈ *sets M*
with *Q borel* **have** (∫⁺*x*. *f i x* ∗ *indicator* (*Q i* ∩ *A*) *x* ∂*M*) = *emeasure* (*density* (*?M i*) (*f i*)) *A*
by (*auto simp add*: *emeasure-density nn-integral-density subset-eq*
intro!: *nn-integral-cong split*: *split-indicator*)
also have ... = *emeasure N* (*Q i* ∩ *A*)
using *A Q* **by** (*simp add*: *f-density emeasure-restricted subset-eq sets-eq*)
finally have *emeasure N* (*Q i* ∩ *A*) = (∫⁺*x*. *f i x* ∗ *indicator* (*Q i* ∩ *A*) *x* ∂*M*) **.. }**
note *integral-eq* = *this*
let *?f* = λ*x*. (∑ *i*. *f i x* ∗ *indicator* (*Q i*) *x*) + ∞ ∗ *indicator Q0 x*
show *?thesis*
proof (*safe intro*!: *bexI*[*of* - *?f*])
show *?f* ∈ *borel-measurable M* **using** *Q0 borel Q-sets*
by (*auto intro*!: *measurable-If*)
show ⋀*x*. 0 ≤ *?f x* **using** *borel* **by** (*auto intro*!: *suminf-0-le simp*: *indicator-def*)
show *density M ?f* = *N*
proof (*rule measure-eqI*)
fix *A* **assume** *A* ∈ *sets* (*density M ?f*)
then have *A* ∈ *sets M* **by** *simp*
have *Qi*: ⋀*i*. *Q i* ∈ *sets M* **using** *Q* **by** *auto*
have [*intro*,*simp*]: ⋀*i*. (λ*x*. *f i x* ∗ *indicator* (*Q i* ∩ *A*) *x*) ∈ *borel-measurable M*
⋀*i*. *AE x in M*. 0 ≤ *f i x* ∗ *indicator* (*Q i* ∩ *A*) *x*
using *borel Qi Q0*(*1*) ‹*A* ∈ *sets M*› **by** *auto*
have (∫⁺*x*. *?f x* ∗ *indicator A x* ∂*M*) = (∫⁺*x*. (∑ *i*. *f i x* ∗ *indicator* (*Q i* ∩ *A*) *x*) + ∞ ∗ *indicator* (*Q0* ∩ *A*) *x* ∂*M*)
using *borel* **by** (*intro nn-integral-cong*) (*auto simp*: *indicator-def*)
also have ... = (∫⁺*x*. (∑ *i*. *f i x* ∗ *indicator* (*Q i* ∩ *A*) *x*) ∂*M*) + ∞ ∗

emeasure M (Q0 ∩ A)
using *borel Qi Q0(1)* ‹*A ∈ sets M*›
by (*subst nn-integral-add*)
(*auto simp add*: *nn-integral-cmult-indicator sets.Int intro*!: *suminf-0-le*)
also have ... = (∑ *i*. *N* (*Q i* ∩ *A*)) + ∞ ∗ *emeasure M* (*Q0* ∩ *A*)
by (*subst integral-eq*[*OF* ‹*A ∈ sets M*›], *subst nn-integral-suminf*) *auto*
finally have (∫⁺*x*. *?f x* ∗ *indicator A x* ∂*M*) = (∑ *i*. *N* (*Q i* ∩ *A*)) + ∞ ∗ *emeasure M* (*Q0* ∩ *A*) **.**
moreover have (∑ *i*. *N* (*Q i* ∩ *A*)) = *N* ((⋃ *i*. *Q i*) ∩ *A*)
using *Q Q-sets* ‹*A ∈ sets M*›
by (*subst suminf-emeasure*) (*auto simp*: *disjoint-family-on-def sets-eq*)
moreover have ∞ ∗ *emeasure M* (*Q0* ∩ *A*) = *N* (*Q0* ∩ *A*)
proof −
have *Q0* ∩ *A ∈ sets M* **using** *Q0(1)* ‹*A ∈ sets M*› **by** *blast*
from *in-Q0*[*OF this*] **show** *?thesis* **by** (*auto simp*: *ennreal-top-mult*)
qed
moreover have *Q0* ∩ *A ∈ sets M* ((⋃ *i*. *Q i*) ∩ *A*) ∈ *sets M*
using *Q-sets* ‹*A ∈ sets M*› *Q0(1)* **by** *auto*
moreover have ((⋃ *i*. *Q i*) ∩ *A*) ∪ (*Q0* ∩ *A*) = *A* ((⋃ *i*. *Q i*) ∩ *A*) ∩ (*Q0* ∩ *A*) = {}
using ‹*A ∈ sets M*› *sets.sets-into-space Q0* **by** *auto*
ultimately have *N A* = (∫⁺*x*. *?f x* ∗ *indicator A x* ∂*M*)
using *plus-emeasure*[*of* (⋃ *i*. *Q i*) ∩ *A N Q0* ∩ *A*] **by** (*simp add*: *sets-eq*)
with ‹*A ∈ sets M*› *borel Q Q0(1)* **show** *emeasure* (*density M ?f*) *A* = *N A*
by (*auto simp*: *subset-eq emeasure-density*)
qed (*simp add*: *sets-eq*)
qed
qed

lemma (**in** *sigma-finite-measure*) *Radon-Nikodym*:
assumes *ac*: *absolutely-continuous M N* **assumes** *sets-eq*: *sets N* = *sets M*
shows ∃*f* ∈ *borel-measurable M*. (∀*x*. 0 ≤ *f x*) ∧ *density M f* = *N*
proof −
from *Ex-finite-integrable-function*
obtain *h* **where** *finite*: *integral*^{N} *M h* ≠ ∞ **and**
borel: *h ∈ borel-measurable M* **and**
nn: ⋀*x*. 0 ≤ *h x* **and**
pos: ⋀*x*. *x ∈ space M* ⟹ 0 < *h x* **and**
⋀*x*. *x ∈ space M* ⟹ *h x* < ∞ **by** *auto*
let *?T* = λ*A*. (∫⁺*x*. *h x* ∗ *indicator A x* ∂*M*)
let *?MT* = *density M h*
from *borel finite nn* **interpret** *T*: *finite-measure ?MT*
by (*auto intro*!: *finite-measureI cong*: *nn-integral-cong simp*: *emeasure-density*)
have *absolutely-continuous ?MT N sets N* = *sets ?MT*
proof (*unfold absolutely-continuous-def*, *safe*)
fix *A* **assume** *A ∈ null-sets ?MT*
with *borel* **have** *A ∈ sets M AE x in M*. *x ∈ A* ⟶ *h x* ≤ 0
by (*auto simp add*: *null-sets-density-iff*)
with *pos sets.sets-into-space* **have** *AE x in M*. *x* ∉ *A*

by (*elim eventually-mono*) (*auto simp*: *not-le*[*symmetric*])
then have $A \in$ *null-sets M*
using ‹$A \in$ *sets M*› **by** (*simp add*: *AE-iff-null-sets*)
with *ac* **show** $A \in$ *null-sets N*
by (*auto simp*: *absolutely-continuous-def*)
qed (*auto simp add*: *sets-eq*)
from *T.Radon-Nikodym-finite-measure-infinite*[*OF this*]
obtain *f* **where** *f-borel*: $f \in$ *borel-measurable M* $\bigwedge x.\ 0 \le f\ x$ *density ?MT f* = *N* **by** *auto*
with *nn borel* **show** *?thesis*
by (*auto intro*!: *bexI*[*of* - $\lambda x.\ h\ x * f\ x$] *simp*: *density-density-eq*)
qed

11.3 Uniqueness of densities

lemma *finite-density-unique*:
assumes *borel*: $f \in$ *borel-measurable M* $g \in$ *borel-measurable M*
assumes *pos*: *AE x in M*. $0 \le f\ x$ *AE x in M*. $0 \le g\ x$
and *fin*: $\mathit{integral}^N\ M\ f \neq \infty$
shows *density M f* = *density M g* $\longleftrightarrow$ (*AE x in M*. $f\ x = g\ x$)
proof (*intro iffI ballI*)
fix *A* **assume** *eq*: *AE x in M*. $f\ x = g\ x$
with *borel* **show** *density M f* = *density M g*
by (*auto intro*: *density-cong*)
next
let *?P* = $\lambda f\ A.\ \int^+ x.\ f\ x * \mathit{indicator}\ A\ x\ \partial M$
assume *density M f* = *density M g*
with *borel* **have** *eq*: $\forall A \in \mathit{sets}\ M.$ *?P f A* = *?P g A*
by (*simp add*: *emeasure-density*[*symmetric*])
from *this*[*THEN bspec*, *OF sets.top*] *fin*
have *g-fin*: $\mathit{integral}^N\ M\ g \neq \infty$ **by** (*simp cong*: *nn-integral-cong*)
{ fix *f g* **assume** *borel*: $f \in$ *borel-measurable M* $g \in$ *borel-measurable M*
and *pos*: *AE x in M*. $0 \le f\ x$ *AE x in M*. $0 \le g\ x$
and *g-fin*: $\mathit{integral}^N\ M\ g \neq \infty$ **and** *eq*: $\forall A \in \mathit{sets}\ M.$ *?P f A* = *?P g A*
let *?N* = {$x \in$ *space M*. $g\ x < f\ x$}
have *N*: *?N* $\in$ *sets M* **using** *borel* **by** *simp*
have *?P g ?N* $\le$ $\mathit{integral}^N\ M\ g$ **using** *pos*
by (*intro nn-integral-mono-AE*) (*auto split*: *split-indicator*)
then have *Pg-fin*: *?P g ?N* $\neq \infty$ **using** *g-fin* **by** (*auto simp*: *top-unique*)
have *?P* ($\lambda x.\ (f\ x - g\ x)$) *?N* = ($\int^+ x.\ f\ x * \mathit{indicator}$ *?N x* $-$ *g x* $*$ *indicator ?N x* ∂M)
by (*auto intro*!: *nn-integral-cong simp*: *indicator-def*)
also have ... = *?P f ?N* $-$ *?P g ?N*
proof (*rule nn-integral-diff*)
show ($\lambda x.\ f\ x * \mathit{indicator}$ *?N x*) $\in$ *borel-measurable M* ($\lambda x.\ g\ x * \mathit{indicator}$ *?N x*) $\in$ *borel-measurable M*
using *borel N* **by** *auto*
show *AE x in M*. $g\ x * \mathit{indicator}$ *?N x* $\le f\ x * \mathit{indicator}$ *?N x*
using *pos* **by** (*auto split*: *split-indicator*)

```
    qed fact
    also have ... = 0
      unfolding eq[THEN bspec, OF N] using Pg-fin by auto
    finally have AE x in M. f x ≤ g x
      using pos borel nn-integral-PInf-AE[OF borel(2) g-fin]
      by (subst (asm) nn-integral-0-iff-AE)
        (auto split: split-indicator simp: not-less ennreal-minus-eq-0) }
  from this[OF borel pos g-fin eq] this[OF borel(2,1) pos(2,1) fin] eq
  show AE x in M. f x = g x by auto
qed

lemma (in finite-measure) density-unique-finite-measure:
  assumes borel: f ∈ borel-measurable M f' ∈ borel-measurable M
  assumes pos: AE x in M. 0 ≤ f x AE x in M. 0 ≤ f' x
  assumes f: ⋀A. A ∈ sets M ⟹ (∫⁺x. f x * indicator A x ∂M) = (∫⁺x. f' x
* indicator A x ∂M)
    (is ⋀A. A ∈ sets M ⟹ ?P f A = ?P f' A)
  shows AE x in M. f x = f' x
proof −
  let ?D = λf. density M f
  let ?N = λA. ?P f A and ?N' = λA. ?P f' A
  let ?f = λA x. f x * indicator A x and ?f' = λA x. f' x * indicator A x

  have ac: absolutely-continuous M (density M f) sets (density M f) = sets M
    using borel by (auto intro!: absolutely-continuousI-density)
  from split-space-into-finite-sets-and-rest[OF this]
  obtain Q0 and Q :: nat ⇒ 'a set
    where Q: disjoint-family Q range Q ⊆ sets M
    and Q0: Q0 ∈ sets M Q0 = space M − (⋃ i. Q i)
    and in-Q0: ⋀A. A ∈ sets M ⟹ A ⊆ Q0 ⟹ emeasure M A = 0 ∧ ?D f A
= 0 ∨ 0 < emeasure M A ∧ ?D f A = ∞
    and Q-fin: ⋀i. ?D f (Q i) ≠ ∞ by force
  with borel pos have in-Q0: ⋀A. A ∈ sets M ⟹ A ⊆ Q0 ⟹ emeasure M A
= 0 ∧ ?N A = 0 ∨ 0 < emeasure M A ∧ ?N A = ∞
    and Q-fin: ⋀i. ?N (Q i) ≠ ∞ by (auto simp: emeasure-density subset-eq)

  from Q have Q-sets: ⋀i. Q i ∈ sets M by auto
  let ?D = {x∈space M. f x ≠ f' x}
  have ?D ∈ sets M using borel by auto
  have *: ⋀i x A. ⋀y::ennreal. y * indicator (Q i) x * indicator A x = y * indicator
(Q i ∩ A) x
    unfolding indicator-def by auto
  have ∀ i. AE x in M. ?f (Q i) x = ?f' (Q i) x using borel Q-fin Q pos
    by (intro finite-density-unique[THEN iffD1] allI)
      (auto intro!: f measure-eqI simp: emeasure-density * subset-eq)
  moreover have AE x in M. ?f Q0 x = ?f' Q0 x
  proof (rule AE-I')
    { fix f :: 'a ⇒ ennreal assume borel: f ∈ borel-measurable M
      and eq: ⋀A. A ∈ sets M ⟹ ?N A = (∫⁺x. f x * indicator A x ∂M)
```

let *?A* = $\lambda i.\ Q0 \cap \{x \in space\ M.\ f\ x < (i::nat)\}$
have $(\bigcup i.\ ?A\ i) \in null\text{-}sets\ M$
proof (*rule null-sets-UN*)
fix *i* ::*nat* **have** *?A i* ∈ *sets M*
using *borel Q0*(*1*) **by** *auto*
have *?N* (*?A i*) ≤ $(\int^+ x.\ (i::ennreal) * indicator\ (?A\ i)\ x\ \partial M)$
unfolding *eq*[*OF* ‹*?A i* ∈ *sets M*›]
by (*auto intro*!: *nn-integral-mono simp*: *indicator-def*)
also have ... = *i* ∗ *emeasure M* (*?A i*)
using ‹*?A i* ∈ *sets M*› **by** (*auto intro*!: *nn-integral-cmult-indicator*)
also have ... < ∞ **using** *emeasure-real*[*of ?A i*] **by** (*auto simp*: *ennreal-mult-less-top of-nat-less-top*)
finally have *?N* (*?A i*) ≠ ∞ **by** *simp*
then show *?A i* ∈ *null-sets M* **using** *in-Q0*[*OF* ‹*?A i* ∈ *sets M*›] ‹*?A i* ∈ *sets M*› **by** *auto*
qed
also have $(\bigcup i.\ ?A\ i) = Q0 \cap \{x \in space\ M.\ f\ x \neq \infty\}$
by (*auto simp*: *ennreal-Ex-less-of-nat less-top*[*symmetric*])
finally have *Q0* ∩ {*x*∈*space M*. *f x* ≠ ∞} ∈ *null-sets M* **by** *simp* **}**
from *this*[*OF borel*(*1*) *refl*] *this*[*OF borel*(*2*) *f*]
have *Q0* ∩ {*x*∈*space M*. *f x* ≠ ∞} ∈ *null-sets M Q0* ∩ {*x*∈*space M*. *f*′ *x* ≠ ∞} ∈ *null-sets M* **by** *simp-all*
then show (*Q0* ∩ {*x*∈*space M*. *f x* ≠ ∞}) ∪ (*Q0* ∩ {*x*∈*space M*. *f*′ *x* ≠ ∞}) ∈ *null-sets M* **by** (*rule null-sets.Un*)
show {*x* ∈ *space M*. *?f Q0 x* ≠ *?f*′ *Q0 x*} ⊆
(*Q0* ∩ {*x*∈*space M*. *f x* ≠ ∞}) ∪ (*Q0* ∩ {*x*∈*space M*. *f*′ *x* ≠ ∞}) **by** (*auto simp*: *indicator-def*)
qed
moreover have *AE x in M*. (*?f Q0 x* = *?f*′ *Q0 x*) ⟶ (∀ *i*. *?f* (*Q i*) *x* = *?f*′ (*Q i*) *x*) ⟶
?f (*space M*) *x* = *?f*′ (*space M*) *x*
by (*auto simp*: *indicator-def Q0*)
ultimately have *AE x in M*. *?f* (*space M*) *x* = *?f*′ (*space M*) *x*
unfolding *AE-all-countable*[*symmetric*]
by *eventually-elim* (*auto intro*!: *AE-I2 split*: *if-split-asm simp*: *indicator-def*)
then show *AE x in M*. *f x* = *f*′ *x* **by** *auto*
qed

lemma (**in** *sigma-finite-measure*) *density-unique*:
assumes *f*: *f* ∈ *borel-measurable M*
assumes *f*′: *f*′ ∈ *borel-measurable M*
assumes *density-eq*: *density M f* = *density M f*′
shows *AE x in M*. *f x* = *f*′ *x*
proof −
obtain *h* **where** *h-borel*: *h* ∈ *borel-measurable M*
and *fin*: *integral*^{N} *M h* ≠ ∞ **and** *pos*: ⋀*x*. *x* ∈ *space M* ⟹ 0 < *h x* ∧ *h x* < ∞ ⋀*x*. 0 ≤ *h x*
using *Ex-finite-integrable-function* **by** *auto*
then have *h-nn*: *AE x in M*. 0 ≤ *h x* **by** *auto*

let *?H* = *density M h*
interpret *h*: *finite-measure ?H*
using *fin h-borel pos*
by (*intro finite-measureI*) (*simp cong*: *nn-integral-cong emeasure-density add*: *fin*)
let *?fM* = *density M f*
let *?f'M* = *density M f'*
{ fix *A* **assume** *A* ∈ *sets M*
then have {*x* ∈ *space M*. *h x* ∗ *indicator A x* ≠ *0*} = *A*
using *pos*(*1*) *sets.sets-into-space* **by** (*force simp*: *indicator-def*)
then have (∫⁺*x*. *h x* ∗ *indicator A x* ∂*M*) = *0* ⟷ *A* ∈ *null-sets M*
using *h-borel* ‹*A* ∈ *sets M*› *h-nn* **by** (*subst nn-integral-0-iff*) *auto* **}**
note *h-null-sets* = *this*
{ fix *A* **assume** *A* ∈ *sets M*
have (∫⁺*x*. *f x* ∗ (*h x* ∗ *indicator A x*) ∂*M*) = (∫⁺*x*. *h x* ∗ *indicator A x* ∂*?fM*)
using ‹*A* ∈ *sets M*› *h-borel h-nn f f'*
by (*intro nn-integral-density*[*symmetric*]) *auto*
also have ... = (∫⁺*x*. *h x* ∗ *indicator A x* ∂*?f'M*)
by (*simp-all add*: *density-eq*)
also have ... = (∫⁺*x*. *f' x* ∗ (*h x* ∗ *indicator A x*) ∂*M*)
using ‹*A* ∈ *sets M*› *h-borel h-nn f f'*
by (*intro nn-integral-density*) *auto*
finally have (∫⁺*x*. *h x* ∗ (*f x* ∗ *indicator A x*) ∂*M*) = (∫⁺*x*. *h x* ∗ (*f' x* ∗ *indicator A x*) ∂*M*)
by (*simp add*: *ac-simps*)
then have (∫⁺*x*. (*f x* ∗ *indicator A x*) ∂*?H*) = (∫⁺*x*. (*f' x* ∗ *indicator A x*) ∂*?H*)
using ‹*A* ∈ *sets M*› *h-borel h-nn f f'*
by (*subst* (*asm*) (*1 2*) *nn-integral-density*[*symmetric*]) *auto* **}**
then have *AE x in ?H*. *f x* = *f' x* **using** *h-borel h-nn f f'*
by (*intro h.density-unique-finite-measure absolutely-continuous-AE*[*of M*]) *auto*
with *AE-space*[*of M*] *pos* **show** *AE x in M*. *f x* = *f' x*
unfolding *AE-density*[*OF h-borel*] **by** *auto*
qed

lemma (**in** *sigma-finite-measure*) *density-unique-iff*:
assumes *f*: *f* ∈ *borel-measurable M* **and** *f'*: *f'* ∈ *borel-measurable M*
shows *density M f* = *density M f'* ⟷ (*AE x in M*. *f x* = *f' x*)
using *density-unique*[*OF assms*] *density-cong*[*OF f f'*] **by** *auto*

lemma *sigma-finite-density-unique*:
assumes *borel*: *f* ∈ *borel-measurable M g* ∈ *borel-measurable M*
and *fin*: *sigma-finite-measure* (*density M f*)
shows *density M f* = *density M g* ⟷ (*AE x in M*. *f x* = *g x*)
proof
assume *AE x in M*. *f x* = *g x* **with** *borel* **show** *density M f* = *density M g*
by (*auto intro*: *density-cong*)
next

```
  assume eq: density M f = density M g
  interpret f: sigma-finite-measure density M f by fact
  from f.sigma-finite-incseq guess A . note cover = this

  have AE x in M. ∀ i. x ∈ A i ⟶ f x = g x
    unfolding AE-all-countable
  proof
    fix i
   have density (density M f) (indicator (A i)) = density (density M g) (indicator
(A i))
     unfolding eq ..
    moreover have (∫⁺x. f x * indicator (A i) x ∂M) ≠ ∞
     using cover(1) cover(3)[of i] borel by (auto simp: emeasure-density subset-eq)
    ultimately have AE x in M. f x * indicator (A i) x = g x * indicator (A i) x
      using borel cover(1)
      by (intro finite-density-unique[THEN iffD1]) (auto simp: density-density-eq
subset-eq)
    then show AE x in M. x ∈ A i ⟶ f x = g x
      by auto
  qed
  with AE-space show AE x in M. f x = g x
    apply eventually-elim
    using cover(2)[symmetric]
    apply auto
    done
qed

lemma (in sigma-finite-measure) sigma-finite-iff-density-finite':
  assumes f: f ∈ borel-measurable M
  shows sigma-finite-measure (density M f) ⟷ (AE x in M. f x ≠ ∞)
    (is sigma-finite-measure ?N ⟷ -)
proof
  assume sigma-finite-measure ?N
  then interpret N: sigma-finite-measure ?N .
  from N.Ex-finite-integrable-function obtain h where
    h: h ∈ borel-measurable M integral^N ?N h ≠ ∞ and
    fin: ∀ x∈space M. 0 < h x ∧ h x < ∞
    by auto
  have AE x in M. f x * h x ≠ ∞
  proof (rule AE-I')
    have integral^N ?N h = (∫⁺x. f x * h x ∂M)
      using f h by (auto intro!: nn-integral-density)
    then have (∫⁺x. f x * h x ∂M) ≠ ∞
      using h(2) by simp
    then show (λx. f x * h x) -` {∞} ∩ space M ∈ null-sets M
      using f h(1) by (auto intro!: nn-integral-PInf[unfolded infinity-ennreal-def]
borel-measurable-vimage)
  qed auto
  then show AE x in M. f x ≠ ∞
```

```
    using fin by (auto elim!: AE-Ball-mp simp: less-top ennreal-mult-less-top)
next
  assume AE: AE x in M. f x ≠ ∞
  from sigma-finite guess Q . note Q = this
  def A ≡ λi. f -` (case i of 0 ⇒ {∞} | Suc n ⇒ {.. ennreal(of-nat (Suc n))})
∩ space M
  { fix i j have A i ∩ Q j ∈ sets M
    unfolding A-def using f Q
    apply (rule-tac sets.Int)
    by (cases i) (auto intro: measurable-sets[OF f(1)]) }
  note A-in-sets = this

  show sigma-finite-measure ?N
  proof (standard, intro exI conjI ballI)
    show countable (range (λ(i, j). A i ∩ Q j))
      by auto
    show range (λ(i, j). A i ∩ Q j) ⊆ sets (density M f)
      using A-in-sets by auto
  next
    have ⋃ range (λ(i, j). A i ∩ Q j) = (⋃ i j. A i ∩ Q j)
      by auto
    also have ... = (⋃ i. A i) ∩ space M using Q by auto
    also have (⋃ i. A i) = space M
    proof safe
      fix x assume x: x ∈ space M
      show x ∈ (⋃ i. A i)
      proof (cases f x rule: ennreal-cases)
       case top with x show ?thesis unfolding A-def by (auto intro: exI[of - 0])
      next
        case (real r)
        with ennreal-Ex-less-of-nat[of f x] obtain n :: nat where f x < n
          by auto
        also have n < (Suc n :: ennreal)
          by simp
        finally show ?thesis
            using x real by (auto simp: A-def ennreal-of-nat-eq-real-of-nat intro!:
exI[of - Suc n])
      qed
    qed (auto simp: A-def)
    finally show ⋃ range (λ(i, j). A i ∩ Q j) = space ?N by simp
  next
    fix X assume X ∈ range (λ(i, j). A i ∩ Q j)
    then obtain i j where [simp]:X = A i ∩ Q j by auto
    have (∫+x. f x * indicator (A i ∩ Q j) x ∂M) ≠ ∞
    proof (cases i)
      case 0
      have AE x in M. f x * indicator (A i ∩ Q j) x = 0
        using AE by (auto simp: A-def ‹i = 0›)
      from nn-integral-cong-AE[OF this] show ?thesis by simp
```

```
    next
      case (Suc n)
      then have (∫⁺x. f x * indicator (A i ∩ Q j) x ∂M) ≤
        (∫⁺x. (Suc n :: ennreal) * indicator (Q j) x ∂M)
      by (auto intro!: nn-integral-mono simp: indicator-def A-def ennreal-of-nat-eq-real-of-nat)
      also have ... = Suc n * emeasure M (Q j)
        using Q by (auto intro!: nn-integral-cmult-indicator)
      also have ... < ∞
        using Q by (auto simp: ennreal-mult-less-top less-top of-nat-less-top)
      finally show ?thesis by simp
    qed
    then show emeasure ?N X ≠ ∞
      using A-in-sets Q f by (auto simp: emeasure-density)
  qed
qed

lemma (in sigma-finite-measure) sigma-finite-iff-density-finite:
  f ∈ borel-measurable M ⟹ sigma-finite-measure (density M f) ⟷ (AE x in
M. f x ≠ ∞)
  by (subst sigma-finite-iff-density-finite')
     (auto simp: max-def intro!: measurable-If)
```

11.4 Radon-Nikodym derivative

```
definition RN-deriv :: 'a measure ⇒ 'a measure ⇒ 'a ⇒ ennreal where
  RN-deriv M N =
    (if ∃f. f ∈ borel-measurable M ∧ density M f = N
       then SOME f. f ∈ borel-measurable M ∧ density M f = N
       else (λ-. 0))

lemma RN-derivI:
  assumes f ∈ borel-measurable M density M f = N
  shows density M (RN-deriv M N) = N
proof −
  have ∃f. f ∈ borel-measurable M ∧ density M f = N
    using assms by auto
  moreover then have density M (SOME f. f ∈ borel-measurable M ∧ density
M f = N) = N
    by (rule someI2-ex) auto
  ultimately show ?thesis
    by (auto simp: RN-deriv-def)
qed

lemma borel-measurable-RN-deriv[measurable]: RN-deriv M N ∈ borel-measurable
M
proof −
  { assume ex: ∃f. f ∈ borel-measurable M ∧ density M f = N
   have 1: (SOME f. f ∈ borel-measurable M ∧ density M f = N) ∈ borel-measurable
M
```

```
    using ex by (rule someI2-ex) auto }
  from this show ?thesis
    by (auto simp: RN-deriv-def)
qed

lemma density-RN-deriv-density:
  assumes f: f ∈ borel-measurable M
  shows density M (RN-deriv M (density M f)) = density M f
  by (rule RN-derivI[OF f]) simp

lemma (in sigma-finite-measure) density-RN-deriv:
  absolutely-continuous M N ⟹ sets N = sets M ⟹ density M (RN-deriv M N)
= N
  by (metis RN-derivI Radon-Nikodym)

lemma (in sigma-finite-measure) RN-deriv-nn-integral:
  assumes N: absolutely-continuous M N sets N = sets M
    and f: f ∈ borel-measurable M
  shows integral^N N f = (∫⁺x. RN-deriv M N x * f x ∂M)
proof −
  have integral^N N f = integral^N (density M (RN-deriv M N)) f
    using N by (simp add: density-RN-deriv)
  also have ... = (∫⁺x. RN-deriv M N x * f x ∂M)
    using f by (simp add: nn-integral-density)
  finally show ?thesis by simp
qed

lemma null-setsD-AE: N ∈ null-sets M ⟹ AE x in M. x ∉ N
  using AE-iff-null-sets[of N M] by auto

lemma (in sigma-finite-measure) RN-deriv-unique:
  assumes f: f ∈ borel-measurable M
  and eq: density M f = N
  shows AE x in M. f x = RN-deriv M N x
  unfolding eq[symmetric]
  by (intro density-unique-iff[THEN iffD1] f borel-measurable-RN-deriv
        density-RN-deriv-density[symmetric])

lemma RN-deriv-unique-sigma-finite:
  assumes f: f ∈ borel-measurable M
  and eq: density M f = N and fin: sigma-finite-measure N
  shows AE x in M. f x = RN-deriv M N x
  using fin unfolding eq[symmetric]
  by (intro sigma-finite-density-unique[THEN iffD1] f borel-measurable-RN-deriv
        density-RN-deriv-density[symmetric])

lemma (in sigma-finite-measure) RN-deriv-distr:
  fixes T :: 'a ⇒ 'b
  assumes T: T ∈ measurable M M' and T': T' ∈ measurable M' M
```

and *inv*: $\forall x{\in}space\ M.\ T'\ (T\ x) = x$
and *ac*[*simp*]: *absolutely-continuous* (*distr M M' T*) (*distr N M' T*)
and *N*: *sets N* = *sets M*
shows *AE x in M. RN-deriv* (*distr M M' T*) (*distr N M' T*) (*T x*) = *RN-deriv M N x*
proof (*rule RN-deriv-unique*)
have [*simp*]: *sets N* = *sets M* **by** *fact*
note *sets-eq-imp-space-eq*[*OF N*, *simp*]
have *measurable-N*[*simp*]: $\bigwedge M'$. *measurable N M'* = *measurable M M'* **by** (*auto simp*: *measurable-def*)
{ fix *A* **assume** *A* ∈ *sets M*
with *inv T T' sets.sets-into-space*[*OF this*]
have *T* −‘ *T'* −‘ *A* ∩ *T* −‘ *space M'* ∩ *space M* = *A*
by (*auto simp*: *measurable-def*) **}**
note *eq* = *this*[*simp*]
{ fix *A* **assume** *A* ∈ *sets M*
with *inv T T' sets.sets-into-space*[*OF this*]
have (*T'* ∘ *T*) −‘ *A* ∩ *space M* = *A*
by (*auto simp*: *measurable-def*) **}**
note *eq2* = *this*[*simp*]
let *?M'* = *distr M M' T* **and** *?N'* = *distr N M' T*
interpret *M'*: *sigma-finite-measure ?M'*
proof
from *sigma-finite-countable* **guess** *F* **.. note** *F* = *this*
show ∃ *A*. *countable A* ∧ *A* ⊆ *sets* (*distr M M' T*) ∧ ⋃ *A* = *space* (*distr M M' T*) ∧ (∀ *a*∈*A*. *emeasure* (*distr M M' T*) *a* ≠ ∞)
proof (*intro exI conjI ballI*)
show ∗: (λ*A*. *T'* −‘ *A* ∩ *space ?M'*) ‘ *F* ⊆ *sets ?M'*
using *F T'* **by** (*auto simp*: *measurable-def*)
show ⋃((λ*A*. *T'* −‘ *A* ∩ *space ?M'*)‘*F*) = *space ?M'*
using *F T'*[*THEN measurable-space*] **by** (*auto simp*: *set-eq-iff*)
next
fix *X* **assume** *X* ∈ (λ*A*. *T'* −‘ *A* ∩ *space ?M'*)‘*F*
then obtain *A* **where** [*simp*]: *X* = *T'* −‘ *A* ∩ *space ?M'* **and** *A* ∈ *F* **by** *auto*
have *X* ∈ *sets M'* **using** *F T'* ‹*A*∈*F*› **by** *auto*
moreover
have *Fi*: *A* ∈ *sets M* **using** *F* ‹*A*∈*F*› **by** *auto*
ultimately show *emeasure ?M' X* ≠ ∞
using *F T T'* ‹*A*∈*F*› **by** (*simp add*: *emeasure-distr*)
qed (*insert F*, *auto*)
qed
have (*RN-deriv ?M' ?N'*) ∘ *T* ∈ *borel-measurable M*
using *T ac* **by** *measurable*
then show (λ*x*. *RN-deriv ?M' ?N'* (*T x*)) ∈ *borel-measurable M*
by (*simp add*: *comp-def*)

have *N* = *distr N M* (*T'* ∘ *T*)
by (*subst measure-of-of-measure*[*of N*, *symmetric*])

(*auto simp add*: *distr-def sets.sigma-sets-eq intro*!: *measure-of-eq sets.space-closed*)
also have ... = *distr* (*distr N M' T*) *M T'*
using *T T'* **by** (*simp add*: *distr-distr*)
also have ... = *distr* (*density* (*distr M M' T*) (*RN-deriv* (*distr M M' T*) (*distr N M' T*))) *M T'*
using *ac* **by** (*simp add*: *M'.density-RN-deriv*)
also have ... = *density M* (*RN-deriv* (*distr M M' T*) (*distr N M' T*) ∘ *T*)
by (*simp add*: *distr-density-distr*[*OF T T'*, *OF inv*])
finally show *density M* (λ*x*. *RN-deriv* (*distr M M' T*) (*distr N M' T*) (*T x*)) = *N*
by (*simp add*: *comp-def*)
qed

lemma (**in** *sigma-finite-measure*) *RN-deriv-finite*:
assumes *N*: *sigma-finite-measure N* **and** *ac*: *absolutely-continuous M N sets N* = *sets M*
shows *AE x in M*. *RN-deriv M N x* ≠ ∞
proof −
interpret *N*: *sigma-finite-measure N* **by** *fact*
from *N* **show** *?thesis*
using *sigma-finite-iff-density-finite*[*OF borel-measurable-RN-deriv*, *of N*] *density-RN-deriv*[*OF ac*]
by *simp*
qed

lemma (**in** *sigma-finite-measure*)
assumes *N*: *sigma-finite-measure N* **and** *ac*: *absolutely-continuous M N sets N* = *sets M*
and *f*: *f* ∈ *borel-measurable M*
shows *RN-deriv-integrable*: *integrable N f* ⟷
integrable M (λ*x*. *enn2real* (*RN-deriv M N x*) ∗ *f x*) (**is** *?integrable*)
and *RN-deriv-integral*: *integral*^{L} *N f* = (∫ *x*. *enn2real* (*RN-deriv M N x*) ∗ *f x* ∂*M*) (**is** *?integral*)
proof −
note *ac*(*2*)[*simp*] **and** *sets-eq-imp-space-eq*[*OF ac*(*2*), *simp*]
interpret *N*: *sigma-finite-measure N* **by** *fact*

have *eq*: *density M* (*RN-deriv M N*) = *density M* (λ*x*. *enn2real* (*RN-deriv M N x*))
proof (*rule density-cong*)
from *RN-deriv-finite*[*OF assms*(*1*,*2*,*3*)]
show *AE x in M*. *RN-deriv M N x* = *ennreal* (*enn2real* (*RN-deriv M N x*))
by *eventually-elim* (*auto simp*: *less-top*)
qed (*insert ac*, *auto*)

show *?integrable*
apply (*subst density-RN-deriv*[*OF ac*, *symmetric*])
unfolding *eq*
apply (*intro integrable-real-density f AE-I2 enn2real-nonneg*)

```
    apply (insert ac, auto)
    done

  show ?integral
    apply (subst density-RN-deriv[OF ac, symmetric])
    unfolding eq
    apply (intro integral-real-density f AE-I2 enn2real-nonneg)
    apply (insert ac, auto)
    done
qed

lemma (in sigma-finite-measure) real-RN-deriv:
  assumes finite-measure N
  assumes ac: absolutely-continuous M N sets N = sets M
  obtains D where D ∈ borel-measurable M
    and AE x in M. RN-deriv M N x = ennreal (D x)
    and AE x in N. 0 < D x
    and ⋀x. 0 ≤ D x
proof
  interpret N: finite-measure N by fact

  note RN = borel-measurable-RN-deriv density-RN-deriv[OF ac]

  let ?RN = λt. {x ∈ space M. RN-deriv M N x = t}

  show (λx. enn2real (RN-deriv M N x)) ∈ borel-measurable M
    using RN by auto

  have N (?RN ∞) = (∫⁺ x. RN-deriv M N x * indicator (?RN ∞) x ∂M)
    using RN(1) by (subst RN(2)[symmetric]) (auto simp: emeasure-density)
  also have ... = (∫⁺ x. ∞ * indicator (?RN ∞) x ∂M)
    by (intro nn-integral-cong) (auto simp: indicator-def)
  also have ... = ∞ * emeasure M (?RN ∞)
    using RN by (intro nn-integral-cmult-indicator) auto
  finally have eq: N (?RN ∞) = ∞ * emeasure M (?RN ∞) .
  moreover
  have emeasure M (?RN ∞) = 0
  proof (rule ccontr)
    assume emeasure M {x ∈ space M. RN-deriv M N x = ∞} ≠ 0
    then have 0 < emeasure M {x ∈ space M. RN-deriv M N x = ∞}
      by (auto simp: zero-less-iff-neq-zero)
    with eq have N (?RN ∞) = ∞ by (simp add: ennreal-mult-eq-top-iff)
    with N.emeasure-finite[of ?RN ∞] RN show False by auto
  qed
  ultimately have AE x in M. RN-deriv M N x < ∞
   using RN by (intro AE-iff-measurable[THEN iffD2]) (auto simp: less-top[symmetric])
  then show AE x in M. RN-deriv M N x = ennreal (enn2real (RN-deriv M N
x))
    by auto
```

then have *eq*: *AE x in N. RN-deriv M N x = ennreal (enn2real (RN-deriv M N x))*
using *ac absolutely-continuous-AE* **by** *auto*

have *N (?RN 0) = (∫⁺ x. RN-deriv M N x * indicator (?RN 0) x ∂M)*
by (*subst RN(2)[symmetric]*) (*auto simp*: *emeasure-density*)
also have *... = (∫⁺ x. 0 ∂M)*
by (*intro nn-integral-cong*) (*auto simp*: *indicator-def*)
finally have *AE x in N. RN-deriv M N x ≠ 0*
using *RN* **by** (*subst AE-iff-measurable[OF - refl]*) (*auto simp*: *ac cong*: *sets-eq-imp-space-eq*)
with *eq* **show** *AE x in N. 0 < enn2real (RN-deriv M N x)*
by (*auto simp*: *enn2real-positive-iff less-top[symmetric] zero-less-iff-neq-zero*)
qed (*rule enn2real-nonneg*)

lemma (**in** *sigma-finite-measure*) *RN-deriv-singleton*:
assumes *ac*: *absolutely-continuous M N sets N = sets M*
and *x*: *{x} ∈ sets M*
shows *N {x} = RN-deriv M N x * emeasure M {x}*
proof −
from ‹*{x} ∈ sets M*›
have *density M (RN-deriv M N) {x} = (∫⁺w. RN-deriv M N x * indicator {x} w ∂M)*
by (*auto simp*: *indicator-def emeasure-density intro*!: *nn-integral-cong*)
with *x density-RN-deriv[OF ac]* **show** *?thesis*
by (*auto simp*: *max-def*)
qed

end

12 Probability measure

theory *Probability-Measure*
imports *Lebesgue-Measure Radon-Nikodym*
begin

lemma (**in** *finite-measure*) *countable-support*:
countable {x. measure M {x} ≠ 0}
proof *cases*
assume *measure M (space M) = 0*
with *bounded-measure measure-le-0-iff* **have** *{x. measure M {x} ≠ 0} = {}*
by *auto*
then show *?thesis*
by *simp*
next
let *?M = measure M (space M)* **and** *?m = λx. measure M {x}*
assume *?M ≠ 0*
then have *∗*: *{x. ?m x ≠ 0} = (⋃n. {x. ?M / Suc n < ?m x})*
using *reals-Archimedean[of ?m x / ?M* **for** *x]*

```
    by (auto simp: field-simps not-le[symmetric] measure-nonneg divide-le-0-iff
measure-le-0-iff)
  have **: ⋀n. finite {x. ?M / Suc n < ?m x}
  proof (rule ccontr)
    fix n assume infinite {x. ?M / Suc n < ?m x} (is infinite ?X)
    then obtain X where finite X card X = Suc (Suc n) X ⊆ ?X
      by (metis infinite-arbitrarily-large)
    from this(3) have *: ⋀x. x ∈ X ⟹ ?M / Suc n ≤ ?m x
      by auto
    { fix x assume x ∈ X
       from ‹?M ≠ 0› *[OF this] have ?m x ≠ 0 by (auto simp: field-simps
measure-le-0-iff)
      then have {x} ∈ sets M by (auto dest: measure-notin-sets) }
    note singleton-sets = this
    have ?M < (∑ x∈X. ?M / Suc n)
      using ‹?M ≠ 0›
    by (simp add: ‹card X = Suc (Suc n)› of-nat-Suc field-simps less-le measure-nonneg)
    also have ... ≤ (∑ x∈X. ?m x)
      by (rule setsum-mono) fact
    also have ... = measure M (⋃ x∈X. {x})
      using singleton-sets ‹finite X›
    by (intro finite-measure-finite-Union[symmetric]) (auto simp: disjoint-family-on-def)
    finally have ?M < measure M (⋃ x∈X. {x}) .
    moreover have measure M (⋃ x∈X. {x}) ≤ ?M
    using singleton-sets[THEN sets.sets-into-space] by (intro finite-measure-mono)
auto
    ultimately show False by simp
  qed
  show ?thesis
    unfolding * by (intro countable-UN countableI-type countable-finite[OF **])
qed

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
proof −
  interpret finite-measure M
  proof
    show emeasure M (space M) ≠ ∞ using * by simp
  qed
  show prob-space M by standard fact
qed

lemma prob-space-imp-sigma-finite: prob-space M ⟹ sigma-finite-measure M
  unfolding prob-space-def finite-measure-def by simp
```

abbreviation (**in** *prob-space*) *events* ≡ *sets M*
abbreviation (**in** *prob-space*) *prob* ≡ *measure M*
abbreviation (**in** *prob-space*) *random-variable M' X* ≡ *X* ∈ *measurable M M'*
abbreviation (**in** *prob-space*) *expectation* ≡ *integral*^{L} *M*
abbreviation (**in** *prob-space*) *variance X* ≡ *integral*^{L} *M* (λ*x*. (*X x* − *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* ∈ *measurable M M'* **shows** *prob-space* (*distr M M' f*)
proof (*rule prob-spaceI*)
have *f* −‘ *space M'* ∩ *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* ∈ *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* ∩ *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*
using *emeasure-space-1* **unfolding** *measure-def* **by** (*simp add*: *one-ennreal.rep-eq*)

lemma (**in** *prob-space*) *prob-le-1*[*simp*, *intro*]: *prob A* ≤ *1*
using *bounded-measure*[*of A*] **by** (*simp add*: *prob-space*)

lemma (**in** *prob-space*) *not-empty*: *space M* ≠ {}
using *prob-space* **by** *auto*

lemma (**in** *prob-space*) *emeasure-eq-1-AE*:
S ∈ *sets M* ⟹ *AE x in M*. *x* ∈ *S* ⟹ *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*: *emeasure M S* ≤ *1*
unfolding *ennreal-1*[*symmetric*] *emeasure-eq-measure* **by** (*subst ennreal-le-iff*) *auto*

lemma (**in** *prob-space*) *AE-iff-emeasure-eq-1*:
assumes [*measurable*]: *Measurable.pred M P*
shows (*AE x in M*. *P x*) ⟷ *emeasure M* {*x*∈*space M*. *P x*} = *1*

proof −
have ∗: $\{x \in space\ M.\ \neg\ P\ x\} = space\ M - \{x \in space\ M.\ P\ x\}$
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*: $emeasure\ M\ X \leq 1$
using *emeasure-space*[*of M X*] **by** (*simp add*: *emeasure-space-1*)

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

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

lemma (**in** *prob-space*) *prob-compl*:
assumes *A*: $A \in events$
shows $prob\ (space\ M - A) = 1 - prob\ A$
using *finite-measure-compl*[*OF A*] **by** (*simp add*: *prob-space*)

lemma (**in** *prob-space*) *AE-in-set-eq-1*:
assumes *A*[*measurable*]: $A \in events$ **shows** $(AE\ x\ in\ M.\ x \in A) \longleftrightarrow prob\ A = 1$
proof −
have ∗: $\{x \in space\ M.\ x \in A\} = A$
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*: $(AE\ x\ in\ M.\ False) \longleftrightarrow False$
proof
assume $AE\ x\ in\ M.\ False$
then have $AE\ x\ in\ M.\ x \in \{\}$ **by** *simp*
then show *False*
by (*subst* (*asm*) *AE-in-set-eq-1*) *auto*
qed *simp*

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

assumes *prob A = 1* **shows** *AE x in M. x ∈ A*
proof −
from ‹*prob A = 1*› **have** *A ∈ 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 in M. P) ⟷ P*
by (*cases P*) (*auto simp*: *AE-False*)

lemma (**in** *prob-space*) *ae-filter-bot*: *ae-filter M ≠ bot*
by (*simp add*: *trivial-limit-def*)

lemma (**in** *prob-space*) *AE-contr*:
assumes *ae*: *AE ω in M. P ω AE ω in M. ¬ P ω*
shows *False*
proof −
from *ae* **have** *AE ω in M. False* **by** *eventually-elim auto*
then show *False* **by** *auto*
qed

lemma (**in** *prob-space*) *integral-ge-const*:
fixes *c* :: *real*
shows *integrable M f ⟹ (AE x in M. c ≤ f x) ⟹ c ≤ (∫ x. f x ∂M)*
using *integral-mono-AE*[*of M λx. c f*] *prob-space* **by** *simp*

lemma (**in** *prob-space*) *integral-le-const*:
fixes *c* :: *real*
shows *integrable M f ⟹ (AE x in M. f x ≤ c) ⟹ (∫ x. f x ∂M) ≤ c*
using *integral-mono-AE*[*of M f λx. c*] *prob-space* **by** *simp*

lemma (**in** *prob-space*) *nn-integral-ge-const*:
(AE x in M. c ≤ f x) ⟹ c ≤ (∫⁺x. f x ∂M)
using *nn-integral-mono-AE*[*of λx. c f M*] *emeasure-space-1*
by (*simp split*: *if-split-asm*)

lemma (**in** *prob-space*) *expectation-less*:
fixes *X* :: *- ⇒ real*
assumes [*simp*]: *integrable M X*
assumes *gt*: *AE x in M. X x < b*
shows *expectation X < b*
proof −
have *expectation X < expectation (λx. 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 :: - ⇒ real
  assumes [simp]: integrable M X
  assumes gt: AE x in M. a < X x
  shows a < expectation X
proof −
  have expectation (λx. a) < 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 :: real ⇒ real
  assumes X: integrable M X AE x in M. X x ∈ I
  assumes I: I = {a <..< b} ∨ I = {a <..} ∨ I = {..< b} ∨ I = UNIV
  assumes q: integrable M (λx. q (X x)) convex-on I q
  shows q (expectation X) ≤ expectation (λx. q (X x))
proof −
  let ?F = λx. Inf ((λt. (q x − q t) / (x − t)) ‘ ({x<..} ∩ I))
  from X(2) AE-False have I ≠ {} by auto

  from I have open I by auto

  note I
  moreover
  { assume I ⊆ {a <..}
    with X have a < expectation X
      by (intro expectation-greater) auto }
  moreover
  { assume I ⊆ {..< b}
    with X have expectation X < b
      by (intro expectation-less) auto }
  ultimately have expectation X ∈ I
    by (elim disjE) (auto simp: subset-eq)
  moreover
  { fix y assume y: y ∈ I
    with q(2) ‹open I› have Sup ((λx. q x + ?F x ∗ (y − x)) ‘ I) = q y
      by (auto intro!: cSup-eq-maximum convex-le-Inf-differential image-eqI [OF -
y] simp: interior-open) }
  ultimately have q (expectation X) = Sup ((λx. q x + ?F x ∗ (expectation X −
x)) ‘ I)
    by simp
  also have ... ≤ expectation (λw. q (X w))
  proof (rule cSup-least)
    show (λx. q x + ?F x ∗ (expectation X − x)) ‘ I ≠ {}
      using ‹I ≠ {}› by auto
  next
    fix k assume k ∈ (λx. q x + ?F x ∗ (expectation X − x)) ‘ I
    then guess x .. note x = this
```

have $q\ x + ?F\ x * (expectation\ X\ -\ x) = expectation\ (\lambda w.\ q\ x + ?F\ x * (X\ w - x))$
using *prob-space* **by** (*simp add*: *X*)
also have $\ldots \leq expectation\ (\lambda w.\ q\ (X\ w))$
using ‹*x* ∈ *I*› ‹*open I*› *X*(*2*)
apply (*intro integral-mono-AE integrable-add integrable-mult-right 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 $k \leq expectation\ (\lambda w.\ q\ (X\ w))$ **using** *x* **by** *auto*
qed
finally show $q\ (expectation\ X) \leq expectation\ (\lambda x.\ q\ (X\ x))$ **.**
qed

12.1 Introduce binder for probability

syntax
-prob :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* (('*P*'((/- *in* -./ -)'')))

translations
$\mathcal{P}(x\ in\ M.\ P) => CONST\ measure\ M\ \{x \in CONST\ space\ M.\ P\}$

print-translation ‹
let
fun to-pattern (*Const* (@{*const-syntax Pair*}, -) $ *l* $ *r*) =
Syntax.const @{*const-syntax Pair*} :: *to-pattern l* @ *to-pattern r*
| *to-pattern* (*t as* (*Const* (@{*syntax-const -bound*}, -)) $ -) = [*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 -pattern*}, -) $
t1 $
(*t as* (*Const* (@{*syntax-const -pattern*}, -) $ - $ -)))
= *let*
val (- $ *t2* $ *t3*) = *unnest-tuples t*
in
Syntax.const @{*syntax-const -pattern*} $
unnest-tuples t1 $

```
        (Syntax.const @{syntax-const -patterns} $ t2 $ t3)
      end
    | unnest-tuples pat = pat

    fun tr' [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' 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}, K tr')]
  end
›
```

definition

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

-conditional-prob :: *pttrn* ⇒ *logic* ⇒ *logic* ⇒ *logic* ⇒ *logic* (($'\mathcal{P}'$(- in -. - |/ -')))

translations

$\mathcal{P}(x \text{ in } M.\ P \mid Q)$ => *CONST cond-prob M* ($\lambda x.\ P$) ($\lambda x.\ Q$)

lemma (**in** *prob-space*) *AE-E-prob*:
assumes *ae*: *AE x in M. P x*
obtains *S* **where** $S \subseteq \{x \in space\ M.\ P\ x\}$ $S \in events$ *prob S = 1*
proof −
from *ae*[*THEN AE-E*] **guess** *N* **.**
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*: $\{x \in space\ M.\ P\ x\} \in events \Longrightarrow \mathcal{P}(x\ in\ M.\ \neg\ P\ x) = 1 - \mathcal{P}(x\ in\ M.\ P\ x)$
by (*auto intro*!: *arg-cong*[**where** *f*=*prob*] *simp add*: *prob-compl*[*symmetric*])

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

lemma (**in** *prob-space*) *prob-eq-0-AE*:
assumes *not*: *AE x in M.* ¬ *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*:
$\{x \in space\ M.\ P\ x\} \in sets\ M \Longrightarrow \mathcal{P}(x\ in\ M.\ P\ x) = 0 \longleftrightarrow (AE\ x\ in\ M.\ \neg\ P\ x)$
using *AE-iff-measurable*[*OF - refl*, *of M* $\lambda x.$ ¬ *P x*] **by** (*simp add*: *emeasure-eq-measure measure-nonneg*)

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

lemma (**in** *prob-space*) *prob-eq-0*:
$A \in sets\ M \Longrightarrow prob\ A = 0 \longleftrightarrow (AE\ x\ in\ M.\ x \notin A)$
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*:
$A \in sets\ M \Longrightarrow prob\ A = 1 \longleftrightarrow (AE\ x\ in\ M.\ x \in A)$
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*] **guess** S **. note** $S = this$
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-setsum*:
assumes [*simp*, *intro*]: *finite* I
assumes P: $\bigwedge n.\ n \in I \Longrightarrow \{x{\in}space\ M.\ P\ n\ x\} \in events$
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*] **guess** S **. note** $S = this$
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*:
$AE\ x\ in\ M.\ x \in \{x{\in}space\ M.\ Q\ x\} \longleftrightarrow x \in (\bigcup n{\in}I.\ \{x{\in}space\ M.\ P\ n\ x\} \cap S)$
$\bigwedge n.\ n \in I \Longrightarrow AE\ x\ in\ M.\ x \in \{x{\in}space\ M.\ P\ n\ x\} \longleftrightarrow x \in \{x{\in}space\ M.\ P\ n\ x\} \cap S$
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** $**$:
$\bigwedge n.\ n \in I \Longrightarrow \mathcal{P}(x\ in\ M.\ P\ n\ x) = prob\ (\{x{\in}space\ M.\ P\ n\ x\} \cap S)$
using $P\ Q\ S$ **by** (*intro finite-measure-eq-AE*) *auto*
show *?thesis*
using $S\ P$ *disj*
by (*auto simp add*: $*\ **$ *simp del*: *UN-simps intro*!: *finite-measure-finite-Union*)
qed

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

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

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

lemma (**in** *prob-space*) *joint-distribution-Times-le-fst*:
random-variable MX X $\Longrightarrow$ *random-variable MY Y* $\Longrightarrow$ $A \in sets\ MX \Longrightarrow B \in sets\ MY$
$\Longrightarrow$ *emeasure* (*distr M* (*MX* $\bigotimes_M$ *MY*) ($\lambda x.$ (*X x*, *Y x*))) ($A \times B$) $\leq$ *emeasure* (*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 $\Longrightarrow$ *random-variable MY Y* $\Longrightarrow$ $A \in$ *sets MX* $\Longrightarrow$ $B \in$ *sets MY*
$\Longrightarrow$ *emeasure* (*distr M* (*MX* $\bigotimes_M$ *MY*) ($\lambda x.$ (*X x*, *Y x*))) ($A \times B$) $\leq$ *emeasure* (*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) − (*expectation X*)^2
by (*simp add*: *field-simps prob-space power2-diff power2-eq-square*[*symmetric*])

lemma (**in** *prob-space*) *variance-positive*: *0* $\leq$ *variance* (*X*::$'a \Rightarrow$ *real*)
by (*intro integral-nonneg-AE*) (*auto intro*!: *integral-nonneg-AE*)

lemma (**in** *prob-space*) *variance-mean-zero*:
expectation X = *0* $\Longrightarrow$ *variance X* = *expectation* ($\lambda x.$ (*X x*) ^ *2*)
by *simp*

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* $\bigotimes_M$ *M2*
proof
show *emeasure* (*M1* $\bigotimes_M$ *M2*) (*space* (*M1* $\bigotimes_M$ *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*: $\bigwedge i.$ *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* $\subseteq$ *prob-space* Π_M $i \in I.$ *M i*
proof
show *emeasure* (Π_M $i \in I.$ *M i*) (*space* (Π_M $i \in I.$ *M i*)) = *1*
by (*simp add*: *measure-times M.emeasure-space-1 setprod.neutral-const space-PiM*)
qed

lemma (**in** *finite-product-prob-space*) *prob-times*:
assumes *X*: $\bigwedge i.$ $i \in I \Longrightarrow$ *X i* $\in$ *sets* (*M i*)

shows *prob* $(\Pi_E\ i{\in}I.\ X\ i) = (\prod i{\in}I.\ M.prob\ i\ (X\ i))$
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 $\ldots = (\prod i{\in}I.$ *emeasure* $(M\ i)\ (X\ i))$
using *measure-times* X **by** *simp*
also have $\ldots =$ *ennreal* $(\prod i{\in}I.$ *measure* $(M\ i)\ (X\ i))$
using X **by** (*simp add*: *M.emeasure-eq-measure setprod-ennreal measure-nonneg*)
finally show *?thesis* **by** (*simp add*: *measure-nonneg setprod-nonneg*)
qed

12.2 Distributions

definition *distributed* :: *'a measure* ⇒ *'b measure* ⇒ (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *ennreal*) ⇒ *bool*
where
distributed M N X f ⟷
distr M N X = *density N f* ∧ *f* ∈ *borel-measurable N* ∧ *X* ∈ *measurable M N*

term *distributed*

lemma
assumes *distributed M N X f*
shows *distributed-distr-eq-density*: *distr M N X* = *density N f*
and *distributed-measurable*: *X* ∈ *measurable M N*
and *distributed-borel-measurable*: *f* ∈ *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 ∈ *measurable L M* ⟹ $(\lambda x.\ X\ (g\ x))$ ∈ *measurable L N*
and *distributed-borel-measurable'*[*measurable-dest*]:
h ∈ *measurable L N* ⟹ $(\lambda x.\ f\ (h\ x))$ ∈ *borel-measurable L*
using *distributed-measurable*[*OF D*] *distributed-borel-measurable*[*OF D*]
by *simp-all*

lemma *distributed-real-measurable*:
$(\bigwedge x.\ x \in$ *space N* ⟹ *0* ≤ *f x*) ⟹ *distributed M N X* $(\lambda x.$ *ennreal* $(f\ x))$ ⟹ *f* ∈ *borel-measurable N*
by (*simp-all add*: *distributed-def*)

lemma *distributed-real-measurable'*:
$(\bigwedge x.\ x \in$ *space N* ⟹ *0* ≤ *f x*) ⟹ *distributed M N X* $(\lambda x.$ *ennreal* $(f\ x))$ ⟹
h ∈ *measurable L N* ⟹ $(\lambda x.\ f\ (h\ x))$ ∈ *borel-measurable L*
using *distributed-real-measurable*[*measurable*] **by** *simp*

lemma *joint-distributed-measurable1*:

distributed M $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *f* $\Longrightarrow$ *h1* $\in$ *measurable N M* $\Longrightarrow$ $(\lambda x.\ X\ (h1\ x)) \in$ *measurable N S*
by *simp*

lemma *joint-distributed-measurable2*:
distributed M $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *f* $\Longrightarrow$ *h2* $\in$ *measurable N M* $\Longrightarrow$ $(\lambda x.\ Y\ (h2\ x)) \in$ *measurable N T*
by *simp*

lemma *distributed-count-space*:
assumes *X*: *distributed M* (*count-space A*) *X P* **and** *a*: $a \in A$ **and** *A*: *finite A*
shows *P a* = *emeasure M* $(X -` \{a\} \cap$ *space M*$)$
proof −
have *emeasure M* $(X -` \{a\} \cap$ *space M*$)$ = *emeasure* (*distr M* (*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* $*$ *indicator* $\{a\}$ *x* ∂*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*) $\Longrightarrow$ *g* $\in$ *borel-measurable N* $\Longrightarrow$ *f* $\in$ *borel-measurable N* $\Longrightarrow$
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* ∂*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 $T -` \{x \in$*space Q*. $g\ x = 0\} \cap$ *space P* $\in$ *null-sets* (*distr M P X*)
using *T* **by** (*subst* (*asm*) *null-sets-distr-iff*) *auto*
also have $T -` \{x \in$*space Q*. $g\ x = 0\} \cap$ *space P* $= \{x \in$*space P*. $g\ (T\ x) = 0\}$
using *T* **by** (*auto dest*: *measurable-space*)
finally show *?thesis*
using *f g* **by** (*auto simp add*: *null-sets-density-iff distributed-def*)
qed

lemma *subdensity-real*:
fixes $g :: 'a \Rightarrow$ *real* **and** $f :: 'b \Rightarrow$ *real*
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*. $0 \le g\ (T\ x)) \Longrightarrow$ (*AE x in P*. $0 \le f\ x) \Longrightarrow$ *AE x in P*. $g\ (T\ x) = 0 \longrightarrow f\ x = 0$
using *subdensity*[*OF T*, *of M X* λx. *ennreal* $(f\ x)$ *Y* λx. *ennreal* $(g\ x)$] *assms*
by *auto*

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

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

lemma *distributed-integral*:
distributed M N X f $\Longrightarrow$ *g* $\in$ *borel-measurable N* $\Longrightarrow$ $(\bigwedge x.\ x \in$ *space N* $\Longrightarrow 0 \le f\ x) \Longrightarrow$

$(\int x.\ f\ x * g\ x\ \partial N) = (\int x.\ g\ (X\ x)\ \partial M)$
supply *distributed-real-measurable*[*measurable*]
by (*auto simp*: *distributed-distr-eq-density*[*symmetric*] *integral-real-density*[*symmetric*] *integral-distr*)

lemma *distributed-transform-integral*:
assumes *Px*: *distributed M N X Px* $\bigwedge x.\ x \in$ *space N* $\Longrightarrow 0 \leq Px\ x$
assumes *distributed M P Y Py* $\bigwedge x.\ x \in$ *space P* $\Longrightarrow 0 \leq Py\ x$
assumes *Y*: $Y = T \circ X$ **and** *T*: $T \in$ *measurable N P* **and** *f*: $f \in$ *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 (*rule distributed-integral*) *fact+*
also have $\ldots = (\int x.\ f\ (T\ (X\ x))\ \partial M)$
using *Y* **by** *simp*
also have $\ldots = (\int x.\ Px\ x * f\ (T\ x)\ \partial N)$
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*]: $X \in$ *measurable M S* **and** *Y*[*measurable*]: $Y \in$ *measurable M T*
assumes [*measurable*]: $f \in$ *borel-measurable* $(S \bigotimes_M T)$ **and** *f*: *AE x in* $S \bigotimes_M T$. $0 \leq f\ x$
assumes *eq*: $\bigwedge A\ B.\ A \in$ *sets S* $\Longrightarrow B \in$ *sets T* $\Longrightarrow$
emeasure M $\{x \in$ *space M*. $X\ x \in A \wedge Y\ x \in B\} = (\int^+ x.\ (\int^+ y.\ f\ (x,\ y) *$ *indicator B y* $\partial T) *$ *indicator A x* $\partial S)$
shows *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *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 guess F :: nat ⇒ ('b × 'c)
set .. note F = this
  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)
    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 ⊆ ?E (⋃i. F i) = space ?P ⋀i. emeasure ?L (F i) ≠ ∞
      using F by (auto simp: space-pair-measure)
  next
    fix E assume E ∈ ?E
    then obtain A B where E[simp]: E = A × B
      and A[measurable]: A ∈ sets S and B[measurable]: B ∈ sets T by auto
    have emeasure ?L E = emeasure M {x ∈ space M. X x ∈ A ∧ Y x ∈ B}
      by (auto intro!: arg-cong[where f=emeasure M] simp add: emeasure-distr
measurable-Pair)
    also have ... = (∫+x. (∫+y. (f (x, y) * indicator B y) * indicator A x ∂T)
∂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 ⨂M T) (λx. (X x, Y x)) Pxy
  shows distributed M (T ⨂M S) (λx. (Y x, X x)) (λ(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 ⨂M T) (T ⨂M S) (λ(x, y). (y, x))
    show 1: (λ(x, y). Pxy (y, x)) ∈ borel-measurable ?D
      by auto
    show 2: random-variable (distr (S ⨂M T) (T ⨂M S) (λ(x, y). (y, x))) (λx.
(Y x, X x))
      using Pxy by auto
    { fix A assume A: A ∈ sets (T ⨂M S)
      let ?B = (λ(x, y). (y, x)) -` A ∩ space (S ⨂M T)
      from sets.sets-into-space[OF A]
      have emeasure M ((λx. (Y x, X x)) -` A ∩ 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*) (λ*x*. (*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 ... = ∫$^+$ *x*. *Px x* ∗ *indicator A x* ∂*S*
proof (*rule nn-integral-cong*)
fix *x* **assume** *x* ∈ *space S*
moreover have *eq*: ⋀*y*. *y* ∈ *space T* ⟹ *indicator* (*A* × *space T*) (*x*, *y*) = *indicator A x*
by (*auto simp*: *indicator-def*)
ultimately have (∫$^+$ *y*. *Pxy* (*x*, *y*) ∗ *indicator* (*A* × *space T*) (*x*, *y*) ∂*T*) = (∫$^+$ *y*. *Pxy* (*x*, *y*) ∂*T*) ∗ *indicator A x*
by (*simp add*: *eq nn-integral-multc cong*: *nn-integral-cong*)
also have (∫$^+$ *y*. *Pxy* (*x*, *y*) ∂*T*) = *Px x*
by (*simp add*: *Px-def*)
finally show (∫$^+$ *y*. *Pxy* (*x*, *y*) ∗ *indicator* (*A* × *space T*) (*x*, *y*) ∂*T*) = *Px x* ∗ *indicator A x* **.**
qed
finally show *emeasure* (*distr M S X*) *A* = *emeasure* (*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* ⨂$_M$ *T*) (λ*x*. (*X x*, *Y x*)) *Pxy*
shows *distributed M T Y* (λ*y*. (∫$^+$*x*. *Pxy* (*x*, *y*) ∂*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* ⨂$_M$ *T*) (λ*x*. (*X x*, *Y x*)) *Pxy*
shows *AE x in S*. *Px x* = (∫$^+$*y*. *Pxy* (*x*, *y*) ∂*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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *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*: *distr M S X* $\bigotimes_M$ *distr M T Y* $=$ *distr M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$
shows *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ $(\lambda(x,\ y).\ Px\ x * Py\ y)$
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-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-Y*: *sigma-finite-measure* (*density T Py*) **..**

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

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

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

lemma *distributed-integrable*:
distributed M N X f $\Longrightarrow$ *g* $\in$ *borel-measurable N* $\Longrightarrow$ $(\bigwedge x.\ x \in$ *space N* $\Longrightarrow$ *0* $\leq f\ x) \Longrightarrow$
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* $\Longrightarrow$ *0* $\leq$ *Px x*
assumes *distributed M P Y Py* $\bigwedge x.\ x \in$ *space P* $\Longrightarrow$ *0* $\leq$ *Py x*
assumes *Y*: $Y = (\lambda x.\ T\ (X\ x))$ **and** *T*: $T \in$ *measurable N P* **and** *f*: $f \in$ *borel-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 ... $\longleftrightarrow$ *integrable M* $(\lambda x.\ f\ (T\ (X\ x)))$
using *Y* **by** *simp*
also have ... $\longleftrightarrow$ *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* $(\lambda x.\ ennreal\ (f\ x)) \Longrightarrow (\bigwedge x.\ 0 \leq f\ x) \Longrightarrow$
integrable lborel $(\lambda x.\ f\ x * x) \Longrightarrow$ *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* $= (\int x.\ x^2 * f\ (x + expectation\ X)\ \partial lborel)$
proof (*subst distributed-integral*[*OF D*, *symmetric*])
show $(\int x.\ f\ x * (x - expectation\ X)^2\ \partial lborel) = (\int x.\ x^2 * f\ (x + expectation\ X)\ \partial lborel)$
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* $(\lambda x.\ a + b * X\ x) = b^2 *$ *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 (λx. ennreal (f x)) ∧
finite (X'space M)

lemma *simple-distributed-nonneg*[*dest*]: *simple-distributed M X f ⟹ 0 ≤ f x*
by (*auto simp*: *simple-distributed-def*)

lemma *simple-distributed*:
simple-distributed M X Px ⟹ 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 ⟹ finite (X'space M)*
by (*simp add*: *simple-distributed-def*)

lemma (**in** *prob-space*) *distributed-simple-function-superset*:
assumes *X*: *simple-function M X* ⋀*x. x ∈ X ' space M ⟹ P x = measure M (X −' {x} ∩ space M)*
assumes *A*: *X'space M ⊆ A finite A*
defines *S ≡ count-space A* **and** *P' ≡ (λx. if x ∈ X'space M then P x else 0)*
shows *distributed M S X P'*
unfolding *distributed-def*
proof *safe*
show *(λx. ennreal (P' x)) ∈ borel-measurable S* **unfolding** *S-def* **by** *simp*
show *distr M S X = density S P'*
proof (*rule measure-eqI-finite*)
show *sets (distr M S X) = Pow A sets (density S P') = Pow A*
using *A* **unfolding** *S-def* **by** *auto*
show *finite A* **by** *fact*
fix *a* **assume** *a*: *a ∈ A*
then have *a ∉ X'space M ⟹ X −' {a} ∩ space M = {}* **by** *auto*
with *A a X* **have** *emeasure (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=prob*])
also have *... = (∫⁺x. ennreal (P' a) ∗ indicator {a} x ∂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 *... = (∫⁺x. ennreal (P' x) ∗ indicator {a} x ∂S)*
by (*auto simp*: *indicator-def intro*!: *nn-integral-cong*)
also have *... = emeasure (density S P') {a}*
using *a A* **by** (*intro emeasure-density*[*symmetric*]) (*auto simp*: *S-def*)
finally show *emeasure (distr M S X) {a} = emeasure (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*
$\bigwedge x.\ 0 \leq P\ x$
$\bigwedge x.\ x \in X\ `\ space\ M \Longrightarrow P\ x = measure\ M\ (X - `\ \{x\} \cap space\ M)$
shows *simple-distributed M X P*
unfolding *simple-distributed-def*
proof (*safe intro*!: X)
have *distributed M* (*count-space* ($X\ `\ space\ M$)) $X$ ($\lambda x.$ *ennreal* (*if* $x \in X$`space *M then P x else 0*))
(**is** *?A*)
using *simple-functionD*[*OF X*(*1*)] **by** (*intro distributed-simple-function-superset*[*OF X*(*1*,*3*)]) *auto*
also have *?A* $\longleftrightarrow$ *distributed M* (*count-space* ($X\ `\ space\ M$)) X ($\lambda x.$ *ennreal* (P 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* ($\lambda x.$ ($X\ x,\ Y\ x$)) *Px*
shows *finite* ($X\ `\ space\ M$) *finite* ($Y\ `\ space\ M$)
proof −
have *finite* (($\lambda x.$ ($X\ x,\ Y\ x$)) `\ space *M*)
using X **by** (*auto simp*: *simple-distributed-def simple-functionD*)
then have *finite* (*fst* `\ ($\lambda x.$ ($X\ x,\ Y\ x$)) `\ space *M*) *finite* (*snd* `\ ($\lambda x.$ ($X\ x,\ Y\ x$)) `\ space *M*)
by *auto*
then show *fin*: *finite* ($X\ `\ space\ M$) *finite* ($Y\ `\ space\ M$)
by (*auto simp*: *image-image*)
qed

lemma *simple-distributed-joint2-finite*:
assumes X: *simple-distributed M* ($\lambda x.$ ($X\ x,\ Y\ x,\ Z\ x$)) *Px*
shows *finite* ($X\ `\ space\ M$) *finite* ($Y\ `\ space\ M$) *finite* ($Z\ `\ space\ M$)
proof −
have *finite* (($\lambda x.$ ($X\ x,\ Y\ x,\ Z\ x$)) `\ space *M*)
using X **by** (*auto simp*: *simple-distributed-def simple-functionD*)
then have *finite* (*fst* `\ ($\lambda x.$ ($X\ x,\ Y\ x,\ Z\ x$)) `\ space *M*)
finite ((*fst* $\circ$ *snd*) `\ ($\lambda x.$ ($X\ x,\ Y\ x,\ Z\ x$)) `\ space *M*)
finite ((*snd* $\circ$ *snd*) `\ ($\lambda x.$ ($X\ x,\ Y\ x,\ Z\ x$)) `\ space *M*)
by *auto*
then show *fin*: *finite* ($X\ `\ space\ M$) *finite* ($Y\ `\ space\ M$) *finite* ($Z\ `\ space\ M$)
by (*auto simp*: *image-image*)
qed

lemma *simple-distributed-simple-function*:
simple-distributed M X Px $\Longrightarrow$ *simple-function M X*
unfolding *simple-distributed-def distributed-def*
by (*auto simp*: *simple-function-def measurable-count-space-eq2*)

lemma *simple-distributed-measure*:
simple-distributed M X P $\Longrightarrow$ *a* $\in$ *X'space M* $\Longrightarrow$ *P a* = *measure M* (*X* $-$' {*a*} $\cap$ *space M*)
using *distributed-count-space*[*of M X'space M X P a*, *symmetric*]
by (*auto simp*: *simple-distributed-def measure-def*)

lemma (**in** *prob-space*) *simple-distributed-joint*:
assumes *X*: *simple-distributed M* (*λx*. (*X x*, *Y x*)) *Px*
defines *S* $\equiv$ *count-space* (*X'space M*) $\bigotimes_M$ *count-space* (*Y'space M*)
defines *P* $\equiv$ (*λx*. *if x* $\in$ (*λx*. (*X x*, *Y x*))'*space M then Px x else 0*)
shows *distributed M S* (*λx*. (*X x*, *Y x*)) *P*
proof $-$
from *simple-distributed-joint-finite*[*OF X*, *simp*]
have *S-eq*: *S* = *count-space* (*X'space M* $\times$ *Y'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* (*λx*. (*X x*, *Y x*))
using *X* **by** (*rule simple-distributed-simple-function*)
fix *x* **assume** *x* $\in$ (*λx*. (*X x*, *Y x*)) ' *space M*
from *simple-distributed-measure*[*OF X this*]
show *Px x* = *prob* ((*λx*. (*X x*, *Y x*)) $-$' {*x*} $\cap$ *space M*) .
qed *auto*
qed

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

lemma (**in** *prob-space*) *simple-distributed-setsum-space*:
assumes *X*: *simple-distributed M X f*

shows *setsum f* (*X'space M*) = *1*
proof −
from *X* **have** *setsum f* (*X'space M*) = *prob* (⋃ *i*∈*X'space M*. *X* −' {*i*} ∩ *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!: *setsum.cong arg-cong*[**where** *f*=*prob*])
also have ... = *prob* (*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* (λ*x*. (*X x*, *Y x*)) *Pxy*
assumes *y*: *y* ∈ *Y'space M*
shows *Py y* = (∑ *x*∈*X'space M*. *if* (*x*, *y*) ∈ (λ*x*. (*X x*, *Y x*)) ' *space M then Pxy* (*x*, *y*) *else 0*)
proof −
note *Px* = *simple-distributedI*[*OF Px measure-nonneg refl*]
have *AE y in count-space* (*Y* ' *space M*). *ennreal* (*Py y*) =
∫⁺ *x*. *ennreal* (*if* (*x*, *y*) ∈ (λ*x*. (*X x*, *Y x*)) ' *space M then Pxy* (*x*, *y*) *else 0*) ∂*count-space* (*X* ' *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 *ennreal* (*Py y*) =
(∑ *x*∈*X'space M*. *ennreal* (*if* (*x*, *y*) ∈ (λ*x*. (*X x*, *Y x*)) ' *space M then Pxy* (*x*, *y*) *else 0*))
using *y Px*[*THEN simple-distributed-finite*]
by (*auto simp*: *AE-count-space nn-integral-count-space-finite*)
also have ... = (∑ *x*∈*X'space M*. *if* (*x*, *y*) ∈ (λ*x*. (*X x*, *Y x*)) ' *space M then Pxy* (*x*, *y*) *else 0*)
using *Pxy* **by** (*intro setsum-ennreal*) *auto*
finally show *?thesis*
using *simple-distributed-nonneg*[*OF Py*] *simple-distributed-nonneg*[*OF Pxy*]
by (*subst* (*asm*) *ennreal-inj*) (*auto intro*!: *setsum-nonneg*)
qed

lemma *distributedI-real*:
fixes *f* :: *'a* ⇒ *real*
assumes *gen*: *sets M1* = *sigma-sets* (*space M1*) *E* **and** *Int-stable E*
and *A*: *range A* ⊆ *E* (⋃ *i*::*nat*. *A i*) = *space M1* ⋀*i*. *emeasure* (*distr M M1 X*) (*A i*) ≠ ∞
and *X*: *X* ∈ *measurable M M1*

```
    and f: f ∈ borel-measurable M1 AE x in M1. 0 ≤ f x
    and eq: ⋀A. A ∈ E ⟹ emeasure M (X -` A ∩ space M) = (∫⁺ x. f x *
indicator A x ∂M1)
  shows distributed M M1 X f
  unfolding distributed-def
proof (intro conjI)
  show distr M M1 X = density M1 f
  proof (rule measure-eqI-generator-eq[where A=A])
    { fix A assume A: A ∈ E
      then have A ∈ sigma-sets (space M1) E by auto
      then have A ∈ 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 ∈ E
      then have e ∈ sigma-sets (space M1) E by auto
      then have e ∈ sets M1 unfolding gen .
      then have e ⊆ space M1 by (rule sets.sets-into-space) }
    then show E ⊆ Pow (space M1) by auto
    show sets (distr M M1 X) = sigma-sets (space M1) E
      sets (density M1 (λx. ennreal (f x))) = sigma-sets (space M1) E
      unfolding gen[symmetric] by auto
  qed fact+
qed (insert X f, auto)

lemma distributedI-borel-atMost:
  fixes f :: real ⇒ real
  assumes [measurable]: X ∈ borel-measurable M
    and [measurable]: f ∈ borel-measurable borel and f[simp]: AE x in lborel. 0 ≤
f x
    and g-eq: ⋀a. (∫⁺x. f x * indicator {..a} x ∂lborel)  = ennreal (g a)
    and M-eq: ⋀a. emeasure M {x∈space M. X x ≤ a} = ennreal (g a)
  shows distributed M lborel X f
proof (rule distributedI-real)
  show sets (lborel::real measure) = sigma-sets (space lborel) (range atMost)
    by (simp add: borel-eq-atMost)
  show Int-stable (range atMost :: real set set)
    by (auto simp: Int-stable-def)
  have vimage-eq: ⋀a. (X -` {..a} ∩ space M) = {x∈space M. X x ≤ a} by auto
  def A ≡ λi::nat. {.. real i}
  then show range A ⊆ range atMost (⋃i. A i) = space lborel
    ⋀i. emeasure (distr M lborel X) (A i) ≠ ∞
    by (auto simp: real-arch-simple emeasure-distr vimage-eq M-eq)

  fix A :: real set assume A ∈ range atMost
```

then obtain *a* **where** *A*: $A = \{..a\}$ **by** *auto*
show *emeasure M* $(X - ` A \cap$ *space M*$) = (\int^+ x.$ *f x* $*$ *indicator A x* ∂*lborel*)
unfolding *vimage-eq A M-eq g-eq* **..**
qed *auto*

lemma (**in** *prob-space*) *uniform-distributed-params*:
assumes *X*: *distributed M MX X* ($\lambda x.$ *indicator A x / measure MX A*)
shows $A \in$ *sets MX 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-ennreal-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* $\Longrightarrow$ *A* $\neq$ {} $\Longrightarrow$ *prob-space* (*uniform-count-measure A*)
by *standard* (*auto simp*: *emeasure-uniform-count-measure space-uniform-count-measure one-ennreal-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 in uniform-measure M* {*x*∈*space M*. *Q x*}. *P x*) = $\mathcal{P}$(*x in M*. *P x* | *Q x*)
proof *cases*
assume *Q*: *measure M* {*x*∈*space M*. *Q x*} = *0*
then have $*$: *AE x in M*. ¬ *Q x*
by (*simp add*: *prob-eq-0*)
then have *density M* ($\lambda x.$ *indicator* {*x* ∈ *space M*. *Q x*} *x / emeasure M* {*x* ∈ *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*∈*space M*. *Q x*} ≠ *0*
then show *P*(*x in uniform-measure M* {*x* ∈ *space M*. *Q x*}. *P x*) = *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 ⟹ (⋀*s*. *s* ∈ *S* ⟹ *0* ≤ *p s*) ⟹ (∑ *s*∈*S*. *p s*) = *1* ⟹ *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*: *distr* (*N* ⨂$_M$ *M*) *N fst* = *N*
proof (*intro measure-eqI*)
fix *A* **assume** *A*: *A* ∈ *sets* (*distr* (*N* ⨂$_M$ *M*) *N fst*)
from *A* **have** *emeasure* (*distr* (*N* ⨂$_M$ *M*) *N fst*) *A* = *emeasure* (*N* ⨂$_M$ *M*) (*A* × *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** *emeasure* (*distr* (*N* ⨂$_M$ *M*) *N fst*) *A* = *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* ∈ *J* → *K finite K*
shows *distr* (*PiM K M*) (*Pi*$_M$ *J* (λ*x*. *M* (*t x*))) (λω. λ*n*∈*J*. ω (*t n*)) = *PiM J* (λ*x*. *M* (*t x*))
proof (*rule product-sigma-finite.PiM-eqI*)
show *product-sigma-finite* (λ*x*. *M* (*t x*)) **..**
have *t*‘*J* ⊆ *K* **using** *assms* **by** *auto*
then show [*simp*]: *finite J*
by (*rule finite-imageD*[*OF finite-subset*]) *fact*+
fix *A* **assume** *A*: ⋀*i*. *i* ∈ *J* ⟹ *A i* ∈ *sets* (*M* (*t i*))
moreover have ((λω. λ*n*∈*J*. ω (*t n*)) −‘ *Pi*$_E$ *J A* ∩ *space* (*Pi*$_M$ *K M*)) =
(Π$_E$ *i*∈*K*. *if i* ∈ *t*‘*J then A* (*the-inv-into J t i*) *else space* (*M i*))
using *A A*[*THEN sets.sets-into-space*] ‹*t* ∈ *J* → *K*› ‹*inj-on t J*›
by (*subst prod-emb-Pi*[*symmetric*]) (*auto simp*: *space-PiM PiE-iff the-inv-into-f-f prod-emb-def*)
ultimately show *distr* (*Pi*$_M$ *K M*) (*Pi*$_M$ *J* (λ*x*. *M* (*t x*))) (λω. λ*n*∈*J*. ω (*t n*)) (*Pi*$_E$ *J A*) = (∏ *i*∈*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*› *setprod.reindex*[*OF* ‹*inj-on t J*›]
if-distrib[**where** *f*=*emeasure* (*M* -)] *setprod.If-cases emeasure-space-1 Int-absorb1* ‹*t*‘*J* ⊆ *K*›)
done
qed *simp*

lemma (**in** *product-prob-space*) *distr-restrict*:
J ⊆ *K* ⟹ *finite K* ⟹ (Π$_M$ *i*∈*J*. *M i*) = *distr* (Π$_M$ *i*∈*K*. *M i*) (Π$_M$ *i*∈*J*. *M i*) (λ*f*. *restrict f J*)
using *distr-reorder*[*of* λ*x*. *x J K*] **by** (*simp add*: *Pi-iff subset-eq*)

lemma (**in** *product-prob-space*) *emeasure-prod-emb*[*simp*]:
assumes *L*: *J* ⊆ *L finite L* **and** *X*: *X* ∈ *sets* (*Pi$_M$ J M*)
shows *emeasure* (*Pi$_M$ L M*) (*prod-emb L M J X*) = *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* ⊆ *K* **and** *Q*[*measurable-cong*]: *sets Q* = *sets* (*PiM K M*) **and** *A*[*measurable*]: *A* ∈ *sets* (*PiM I M*)
shows *emeasure* (*distr Q* (*PiM I M*) (λω. *restrict* ω *I*)) *A* = *emeasure Q* (*prod-emb K M I A*)
using ‹*I*⊆*K*› *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*)

end

theory *Complete-Measure*
imports *Bochner-Integration Probability-Measure*
begin

definition
split-completion M A p = (*if A* ∈ *sets M then p* = (*A*, {}) *else*
∃*N*′. *A* = *fst p* ∪ *snd p* ∧ *fst p* ∩ *snd p* = {} ∧ *fst p* ∈ *sets M* ∧ *snd p* ⊆ *N*′ ∧ *N*′ ∈ *null-sets M*)

definition
main-part M A = *fst* (*Eps* (*split-completion M A*))

definition
null-part M A = *snd* (*Eps* (*split-completion M A*))

definition *completion* :: ′*a measure* ⇒ ′*a measure* **where**
completion M = *measure-of* (*space M*) { *S* ∪ *N* |*S N N*′. *S* ∈ *sets M* ∧ *N*′ ∈

null-sets M ∧ N ⊆ N′ }
(*emeasure M ∘ main-part M*)

lemma *completion-into-space*:
{ *S ∪ N |S N N′. S ∈ sets M ∧ N′ ∈ null-sets M ∧ N ⊆ N′* } ⊆ *Pow* (*space M*)
using *sets.sets-into-space* **by** *auto*

lemma *space-completion*[*simp*]: *space* (*completion M*) = *space M*
unfolding *completion-def* **using** *space-measure-of*[*OF completion-into-space*] **by** *simp*

lemma *completionI*:
assumes *A* = *S ∪ N N ⊆ N′ N′ ∈ null-sets M S ∈ sets M*
shows *A* ∈ { *S ∪ N |S N N′. S ∈ sets M ∧ N′ ∈ null-sets M ∧ N ⊆ N′* }
using *assms* **by** *auto*

lemma *completionE*:
assumes *A* ∈ { *S ∪ N |S N N′. S ∈ sets M ∧ N′ ∈ null-sets M ∧ N ⊆ N′* }
obtains *S N N′* **where** *A* = *S ∪ N N ⊆ N′ N′ ∈ null-sets M S ∈ sets M*
using *assms* **by** *auto*

lemma *sigma-algebra-completion*:
sigma-algebra (*space M*) { *S ∪ N |S N N′. S ∈ sets M ∧ N′ ∈ null-sets M ∧ N ⊆ N′* }
(**is** *sigma-algebra - ?A*)
unfolding *sigma-algebra-iff2*
proof (*intro conjI ballI allI impI*)
show *?A ⊆ Pow* (*space M*)
using *sets.sets-into-space* **by** *auto*
next
show {} ∈ *?A* **by** *auto*
next
let *?C* = *space M*
fix *A* **assume** *A* ∈ *?A* **from** *completionE*[*OF this*] **guess** *S N N′* **.**
then show *space M − A* ∈ *?A*
by (*intro completionI*[*of - (?C − S) ∩ (?C − N′) (?C − S) ∩ N′ ∩ (?C − N)*]) *auto*
next
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A ⊆ ?A*
then have ∀ *n*. ∃ *S N N′. A n* = *S ∪ N ∧ S ∈ sets M ∧ N′ ∈ null-sets M ∧ N ⊆ N′*
by (*auto simp*: *image-subset-iff*)
from *choice*[*OF this*] **guess** *S* **..**
from *choice*[*OF this*] **guess** *N* **..**
from *choice*[*OF this*] **guess** *N′* **..**
then show *UNION UNIV A ∈ ?A*
using *null-sets-UN*[*of N′*]
by (*intro completionI*[*of - UNION UNIV S UNION UNIV N UNION UNIV*

N']) *auto*
qed

lemma *sets-completion*:
sets (*completion M*) = { *S* ∪ *N* |*S N N'*. *S* ∈ *sets M* ∧ *N'* ∈ *null-sets M* ∧ *N* ⊆ *N'* }
using *sigma-algebra.sets-measure-of-eq*[*OF sigma-algebra-completion*] **by** (*simp add*: *completion-def*)

lemma *sets-completionE*:
assumes *A* ∈ *sets* (*completion M*)
obtains *S N N'* **where** *A* = *S* ∪ *N N* ⊆ *N'* *N'* ∈ *null-sets M S* ∈ *sets M*
using *assms* **unfolding** *sets-completion* **by** *auto*

lemma *sets-completionI*:
assumes *A* = *S* ∪ *N N* ⊆ *N'* *N'* ∈ *null-sets M S* ∈ *sets M*
shows *A* ∈ *sets* (*completion M*)
using *assms* **unfolding** *sets-completion* **by** *auto*

lemma *sets-completionI-sets*[*intro*, *simp*]:
A ∈ *sets M* ⟹ *A* ∈ *sets* (*completion M*)
unfolding *sets-completion* **by** *force*

lemma *null-sets-completion*:
assumes *N'* ∈ *null-sets M N* ⊆ *N'* **shows** *N* ∈ *sets* (*completion M*)
using *assms* **by** (*intro sets-completionI*[*of N* {} *N N'*]) *auto*

lemma *split-completion*:
assumes *A* ∈ *sets* (*completion M*)
shows *split-completion M A* (*main-part M A*, *null-part M A*)
proof *cases*
assume *A* ∈ *sets M* **then show** *?thesis*
by (*simp add*: *split-completion-def*[*abs-def*] *main-part-def null-part-def*)
next
assume *nA*: *A* ∉ *sets M*
show *?thesis*
unfolding *main-part-def null-part-def if-not-P*[*OF nA*]
proof (*rule someI2-ex*)
from *assms*[*THEN sets-completionE*] **guess** *S N N'* **. note** *A* = *this*
let *?P* = (*S*, *N* − *S*)
show ∃ *p*. *split-completion M A p*
unfolding *split-completion-def if-not-P*[*OF nA*] **using** *A*
proof (*intro exI conjI*)
show *A* = *fst ?P* ∪ *snd ?P* **using** *A* **by** *auto*
show *snd ?P* ⊆ *N'* **using** *A* **by** *auto*
qed *auto*
qed *auto*
qed

lemma
assumes $S \in sets\ (completion\ M)$
shows *main-part-sets*[*intro*, *simp*]: $main\text{-}part\ M\ S \in sets\ M$
and *main-part-null-part-Un*[*simp*]: $main\text{-}part\ M\ S \cup null\text{-}part\ M\ S = S$
and *main-part-null-part-Int*[*simp*]: $main\text{-}part\ M\ S \cap null\text{-}part\ M\ S = \{\}$
using *split-completion*[*OF assms*]
by (*auto simp*: *split-completion-def split*: *if-split-asm*)

lemma *main-part*[*simp*]: $S \in sets\ M \Longrightarrow main\text{-}part\ M\ S = S$
using *split-completion*[*of S M*]
by (*auto simp*: *split-completion-def split*: *if-split-asm*)

lemma *null-part*:
assumes $S \in sets\ (completion\ M)$ **shows** $\exists N.\ N \in null\text{-}sets\ M \wedge null\text{-}part\ M\ S \subseteq N$
using *split-completion*[*OF assms*] **by** (*auto simp*: *split-completion-def split*: *if-split-asm*)

lemma *null-part-sets*[*intro*, *simp*]:
assumes $S \in sets\ M$ **shows** $null\text{-}part\ M\ S \in sets\ M\ emeasure\ M\ (null\text{-}part\ M\ S) = 0$
proof −
have *S*: $S \in sets\ (completion\ M)$ **using** *assms* **by** *auto*
have $S - main\text{-}part\ M\ S \in sets\ M$ **using** *assms* **by** *auto*
moreover
from *main-part-null-part-Un*[*OF S*] *main-part-null-part-Int*[*OF S*]
have $S - main\text{-}part\ M\ S = null\text{-}part\ M\ S$ **by** *auto*
ultimately show *sets*: $null\text{-}part\ M\ S \in sets\ M$ **by** *auto*
from *null-part*[*OF S*] **guess** *N* **..**
with *emeasure-eq-0*[*of N - null-part M S*] *sets*
show $emeasure\ M\ (null\text{-}part\ M\ S) = 0$ **by** *auto*
qed

lemma *emeasure-main-part-UN*:
fixes $S :: nat \Rightarrow 'a\ set$
assumes $range\ S \subseteq sets\ (completion\ M)$
shows $emeasure\ M\ (main\text{-}part\ M\ (\bigcup i.\ (S\ i))) = emeasure\ M\ (\bigcup i.\ main\text{-}part\ M\ (S\ i))$
proof −
have *S*: $\bigwedge i.\ S\ i \in sets\ (completion\ M)$ **using** *assms* **by** *auto*
then have *UN*: $(\bigcup i.\ S\ i) \in sets\ (completion\ M)$ **by** *auto*
have $\forall i.\ \exists N.\ N \in null\text{-}sets\ M \wedge null\text{-}part\ M\ (S\ i) \subseteq N$
using *null-part*[*OF S*] **by** *auto*
from *choice*[*OF this*] **guess** *N* **..** **note** $N = this$
then have *UN-N*: $(\bigcup i.\ N\ i) \in null\text{-}sets\ M$ **by** (*intro null-sets-UN*) *auto*
have $(\bigcup i.\ S\ i) \in sets\ (completion\ M)$ **using** *S* **by** *auto*
from *null-part*[*OF this*] **guess** N' **..** **note** $N' = this$
let *?N* = $(\bigcup i.\ N\ i) \cup N'$
have *null-set*: *?N* $\in null\text{-}sets\ M$ **using** N' *UN-N* **by** (*intro null-sets.Un*) *auto*
have $main\text{-}part\ M\ (\bigcup i.\ S\ i) \cup$ *?N* = $(main\text{-}part\ M\ (\bigcup i.\ S\ i) \cup null\text{-}part\ M$

```
(⋃i. S i)) ∪ ?N
    using N' by auto
  also have ... = (⋃i. main-part M (S i) ∪ null-part M (S i)) ∪ ?N
    unfolding main-part-null-part-Un[OF S] main-part-null-part-Un[OF UN] by
auto
  also have ... = (⋃i. main-part M (S i)) ∪ ?N
    using N by auto
  finally have *: main-part M (⋃i. S i) ∪ ?N = (⋃i. main-part M (S i)) ∪ ?N
.
  have emeasure M (main-part M (⋃i. S i)) = emeasure M (main-part M (⋃i.
S i) ∪ ?N)
    using null-set UN by (intro emeasure-Un-null-set[symmetric]) auto
  also have ... = emeasure M ((⋃i. main-part M (S i)) ∪ ?N)
    unfolding * ..
  also have ... = emeasure M (⋃i. main-part M (S i))
    using null-set S by (intro emeasure-Un-null-set) auto
  finally show ?thesis .
qed

lemma emeasure-completion[simp]:
  assumes S: S ∈ sets (completion M) shows emeasure (completion M) S =
emeasure M (main-part M S)
proof (subst emeasure-measure-of[OF completion-def completion-into-space])
  let ?μ = emeasure M ∘ main-part M
  show S ∈ sets (completion M) ?μ S = emeasure M (main-part M S) using S
by simp-all
  show positive (sets (completion M)) ?μ
    by (simp add: positive-def)
  show countably-additive (sets (completion M)) ?μ
  proof (intro countably-additiveI)
    fix A :: nat ⇒ 'a set assume A: range A ⊆ sets (completion M) disjoint-family
A
    have disjoint-family (λi. main-part M (A i))
    proof (intro disjoint-family-on-bisimulation[OF A(2)])
      fix n m assume A n ∩ A m = {}
      then have (main-part M (A n) ∪ null-part M (A n)) ∩ (main-part M (A
m) ∪ null-part M (A m)) = {}
        using A by (subst (1 2) main-part-null-part-Un) auto
      then show main-part M (A n) ∩ main-part M (A m) = {} by auto
    qed
    then have (∑n. emeasure M (main-part M (A n))) = emeasure M (⋃i.
main-part M (A i))
      using A by (auto intro!: suminf-emeasure)
    then show (∑n. ?μ (A n)) = ?μ (UNION UNIV A)
      by (simp add: completion-def emeasure-main-part-UN[OF A(1)])
  qed
qed

lemma emeasure-completion-UN:
```

range S ⊆ sets (completion M) ⟹
emeasure (completion M) (⋃i::nat. (S i)) = emeasure M (⋃i. main-part M (S i))
by (*subst emeasure-completion*) (*auto simp add*: *emeasure-main-part-UN*)

lemma *emeasure-completion-Un*:
assumes *S*: *S ∈ sets (completion M)* **and** *T*: *T ∈ sets (completion M)*
shows *emeasure (completion M) (S ∪ T) = emeasure M (main-part M S ∪ main-part M T)*
proof (*subst emeasure-completion*)
have *UN*: *(⋃i. binary (main-part M S) (main-part M T) i) = (⋃i. main-part M (binary S T i))*
unfolding *binary-def* **by** (*auto split*: *if-split-asm*)
show *emeasure M (main-part M (S ∪ T)) = emeasure M (main-part M S ∪ main-part M T)*
using *emeasure-main-part-UN*[*of binary S T M*] *assms*
by (*simp add*: *range-binary-eq*, *simp add*: *Un-range-binary UN*)
qed (*auto intro*: *S T*)

lemma *sets-completionI-sub*:
assumes *N*: *N' ∈ null-sets M N ⊆ N'*
shows *N ∈ sets (completion M)*
using *assms* **by** (*intro sets-completionI*[*of - {} N N'*]) *auto*

lemma *completion-ex-simple-function*:
assumes *f*: *simple-function (completion M) f*
shows *∃f'. simple-function M f' ∧ (AE x in M. f x = f' x)*
proof −
let *?F = λx. f −` {x} ∩ space M*
have *F*: *⋀x. ?F x ∈ sets (completion M)* **and** *fin*: *finite (f`space M)*
using *simple-functionD*[*OF f*] *simple-functionD*[*OF f*] **by** *simp-all*
have *∀x. ∃N. N ∈ null-sets M ∧ null-part M (?F x) ⊆ N*
using *F null-part* **by** *auto*
from *choice*[*OF this*] **obtain** *N* **where**
N: *⋀x. null-part M (?F x) ⊆ N x ⋀x. N x ∈ null-sets M* **by** *auto*
let *?N = ⋃x∈f`space M. N x*
let *?f' = λx. if x ∈ ?N then undefined else f x*
have *sets*: *?N ∈ null-sets M* **using** *N fin* **by** (*intro null-sets.finite-UN*) *auto*
show *?thesis* **unfolding** *simple-function-def*
proof (*safe intro*!: *exI*[*of - ?f'*])
have *?f' ` space M ⊆ f`space M ∪ {undefined}* **by** *auto*
from *finite-subset*[*OF this*] *simple-functionD*(*1*)[*OF f*]
show *finite (?f' ` space M)* **by** *auto*
next
fix *x* **assume** *x ∈ space M*
have *?f' −` {?f' x} ∩ space M =*
(if x ∈ ?N then ?F undefined ∪ ?N
else if f x = undefined then ?F (f x) ∪ ?N
else ?F (f x) − ?N)

```
    using N(2) sets.sets-into-space by (auto split: if-split-asm simp: null-sets-def)
   moreover { fix y have ?F y ∪ ?N ∈ sets M
    proof cases
     assume y: y ∈ f`space M
     have ?F y ∪ ?N = (main-part M (?F y) ∪ null-part M (?F y)) ∪ ?N
      using main-part-null-part-Un[OF F] by auto
     also have ... = main-part M (?F y) ∪ ?N
      using y N by auto
     finally show ?thesis
      using F sets by auto
    next
     assume y ∉ f`space M then have ?F y = {} by auto
     then show ?thesis using sets by auto
    qed }
   moreover {
    have ?F (f x) - ?N = main-part M (?F (f x)) ∪ null-part M (?F (f x)) -
?N
     using main-part-null-part-Un[OF F] by auto
    also have ... = main-part M (?F (f x)) - ?N
     using N ‹x ∈ space M› by auto
    finally have ?F (f x) - ?N ∈ sets M
     using F sets by auto }
   ultimately show ?f' -` {?f' x} ∩ space M ∈ sets M by auto
  next
   show AE x in M. f x = ?f' x
    by (rule AE-I', rule sets) auto
  qed
qed

lemma completion-ex-borel-measurable:
  fixes g :: 'a ⇒ ennreal
  assumes g: g ∈ borel-measurable (completion M)
  shows ∃g'∈borel-measurable M. (AE x in M. g x = g' x)
proof -
  from g[THEN borel-measurable-implies-simple-function-sequence'] guess f . note
f = this
  from this(1)[THEN completion-ex-simple-function]
  have ∀i. ∃f'. simple-function M f' ∧ (AE x in M. f i x = f' x) ..
  from this[THEN choice] obtain f' where
    sf: ⋀i. simple-function M (f' i) and
    AE: ∀i. AE x in M. f i x = f' i x by auto
  show ?thesis
  proof (intro bexI)
    from AE[unfolded AE-all-countable[symmetric]]
    show AE x in M. g x = (SUP i. f' i x) (is AE x in M. g x = ?f x)
    proof (elim AE-mp, safe intro!: AE-I2)
      fix x assume eq: ∀i. f i x = f' i x
      moreover have g x = (SUP i. f i x)
        unfolding f by (auto split: split-max)
```

ultimately show *g x = ?f x* **by** *auto*
qed
show *?f ∈ borel-measurable M*
using *sf*[*THEN borel-measurable-simple-function*] **by** *auto*
qed
qed

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

lemma *null-sets-completionI*: *N ∈ null-sets M ⟹ N ∈ null-sets* (*completion M*)
by (*auto simp*: *null-sets-def*)

lemma *AE-completion*: (*AE x in M. P x*) ⟹ (*AE x in completion M. P x*)
unfolding *eventually-ae-filter* **by** (*auto intro*: *null-sets-completionI*)

lemma *null-sets-completion-iff*: *N ∈ sets M ⟹ N ∈ null-sets* (*completion M*)
⟷ *N ∈ null-sets M*
by (*auto simp*: *null-sets-def*)

lemma *AE-completion-iff*: {*x*∈*space M. P x*} ∈ *sets M* ⟹ (*AE x in M. P x*)
⟷ (*AE x in completion M. P x*)
by (*simp add*: *AE-iff-null null-sets-completion-iff*)

end

13 Finite Maps

theory *Fin-Map*
imports *Finite-Product-Measure*
begin

Auxiliary type that is 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.

typedef ('*i*, '*a*) *finmap* ((- ⇒$_F$ /-) [*22*, *21*] *21*) =
{(*I*::'*i set*, *f*::'*i* ⇒ '*a*). *finite I* ∧ *f* ∈ *extensional I*} **by** *auto*

13.1 Domain and Application

definition *domain* **where** *domain P* = *fst* (*Rep-finmap P*)

lemma *finite-domain*[*simp*, *intro*]: *finite* (*domain P*)
by (*cases P*) (*auto simp*: *domain-def Abs-finmap-inverse*)

definition *proj* ('((-)')$_F$ [*0*] *1000*) **where** *proj P i* = *snd* (*Rep-finmap P*) *i*

declare [[*coercion proj*]]

lemma *extensional-proj*[*simp*, *intro*]: $(P)_F \in$ *extensional* (*domain* P)
by (*cases* P) (*auto simp*: *domain-def Abs-finmap-inverse proj-def*[*abs-def*])

lemma *proj-undefined*[*simp*, *intro*]: $i \notin$ *domain* $P \Longrightarrow P\ i =$ *undefined*
using *extensional-proj*[*of* P] **unfolding** *extensional-def* **by** *auto*

lemma *finmap-eq-iff*: $P = Q \longleftrightarrow$ (*domain* P = *domain* $Q \wedge$ ($\forall i \in$ *domain* P. $P\ i$ = $Q\ i$))
by (*cases* P, *cases* Q)
(*auto simp add*: *Abs-finmap-inject extensional-def domain-def proj-def Abs-finmap-inverse intro*: *extensionalityI*)

13.2 Countable Finite Maps

instance *finmap* :: (*countable*, *countable*) *countable*
proof
obtain *mapper* **where** *mapper*: $\bigwedge fm :: 'a \Rightarrow_F 'b.$ *set* (*mapper fm*) = *domain fm*
by (*metis finite-list*[*OF finite-domain*])
have *inj* ($\lambda fm.$ *map* ($\lambda i.$ (i, $(fm)_F\ i$)) (*mapper fm*)) (**is** *inj ?F*)
proof (*rule inj-onI*)
fix *f1 f2* **assume** *?F f1* = *?F f2*
then have *map fst* (*?F f1*) = *map fst* (*?F f2*) **by** *simp*
then have *mapper f1* = *mapper f2* **by** (*simp add*: *comp-def*)
then have *domain f1* = *domain f2* **by** (*simp add*: *mapper*[*symmetric*])
with ‹*?F f1* = *?F f2*› **show** *f1* = *f2*
unfolding ‹*mapper f1* = *mapper f2*› *map-eq-conv mapper*
by (*simp add*: *finmap-eq-iff*)
qed
then show $\exists$ *to-nat* :: $'a \Rightarrow_F 'b \Rightarrow$ *nat*. *inj to-nat*
by (*intro exI*[*of - to-nat* ∘ *?F*] *inj-comp*) *auto*
qed

13.3 Constructor of Finite Maps

definition *finmap-of inds f* = *Abs-finmap* (*inds*, *restrict f inds*)

lemma *proj-finmap-of*[*simp*]:
assumes *finite inds*
shows $(finmap\text{-}of\ inds\ f)_F$ = *restrict f inds*
using *assms*
by (*auto simp*: *Abs-finmap-inverse finmap-of-def proj-def*)

lemma *domain-finmap-of*[*simp*]:
assumes *finite inds*
shows *domain* (*finmap-of inds f*) = *inds*
using *assms*
by (*auto simp*: *Abs-finmap-inverse finmap-of-def domain-def*)

lemma *finmap-of-eq-iff*[*simp*]:
assumes *finite i finite j*
shows *finmap-of i m = finmap-of j n* $\longleftrightarrow$ $i = j \wedge (\forall k{\in}i.\ m\ k{=}\ n\ k)$
using *assms* **by** (*auto simp*: *finmap-eq-iff*)

lemma *finmap-of-inj-on-extensional-finite*:
assumes *finite K*
assumes $S \subseteq$ *extensional K*
shows *inj-on* (*finmap-of K*) *S*
proof (*rule inj-onI*)
fix *x y*::$'a \Rightarrow 'b$
assume *finmap-of K x = finmap-of K y*
hence $(\textit{finmap-of K x})_F = (\textit{finmap-of K y})_F$ **by** *simp*
moreover
assume $x \in S\ y \in S$ **hence** $x \in$ *extensional K* $y \in$ *extensional K* **using** *assms*
by *auto*
ultimately
show $x = y$ **using** *assms* **by** (*simp add*: *extensional-restrict*)
qed

13.4 Product set of Finite Maps

This is *Pi* for Finite Maps, most of this is copied

definition *Pi'* :: $'i\ set \Rightarrow ('i \Rightarrow 'a\ set) \Rightarrow ('i \Rightarrow_F 'a)\ set$ **where**
Pi' I A = { *P*. *domain P = I* $\wedge$ $(\forall i.\ i \in I \longrightarrow (P)_F\ i \in A\ i)$ }

syntax
-Pi' :: [*pttrn*, $'a\ set$, $'b\ set$] => $('a => 'b)\ set$ ((3Π' -$\in$-./ -) 10)
translations
$\Pi'\ x{\in}A.\ B$ == *CONST Pi' A* $(\lambda x.\ B)$

13.4.1 Basic Properties of *Pi'*

lemma *Pi'-I*[*intro*!]: *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'*[*simp*]: *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'-mem*: $f{\in}$ *Pi' A B* $\Longrightarrow x \in A \Longrightarrow f\ x \in B\ x$
by (*simp add*: *Pi'-def*)

lemma *Pi'-iff*: $f \in$ *Pi' I X* $\longleftrightarrow$ *domain f = I* $\wedge$ $(\forall i{\in}I.\ f\ i \in X\ i)$
unfolding *Pi'-def* **by** *auto*

lemma *Pi'E* [*elim*]:

$f \in Pi'\ A\ B \Longrightarrow (f\ x \in B\ x \Longrightarrow domain\ f = A \Longrightarrow Q) \Longrightarrow (x \notin A \Longrightarrow Q) \Longrightarrow Q$
by(*auto simp*: *Pi'-def*)

lemma *in-Pi'-cong*:
$domain\ f = domain\ g \Longrightarrow (\bigwedge w.\ w \in A \Longrightarrow f\ w = g\ w) \Longrightarrow f \in Pi'\ A\ B \longleftrightarrow g \in Pi'\ A\ B$
by (*auto simp*: *Pi'-def*)

lemma *Pi'-eq-empty*[*simp*]:
assumes *finite A* **shows** $(Pi'\ A\ B) = \{\} \longleftrightarrow (\exists x{\in}A.\ B\ x = \{\})$
using *assms*
apply (*simp add*: *Pi'-def*, *auto*)
apply (*drule-tac x* = *finmap-of A* ($\lambda u.$ *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*: $(\bigwedge x.\ x \in A \Longrightarrow B\ x \subseteq C\ x) \Longrightarrow Pi'\ A\ B \subseteq Pi'\ A\ C$
by (*auto simp*: *Pi'-def*)

lemma *Pi-Pi'*: *finite A* $\Longrightarrow (Pi_E\ A\ B) = proj\ `\ Pi'\ A\ B$
apply (*auto simp*: *Pi'-def Pi-def extensional-def*)
apply (*rule-tac x* = *finmap-of A* (*restrict x A*) **in** *image-eqI*)
apply *auto*
done

13.5 Topological Space of Finite Maps

instantiation *finmap* :: (*type*, *topological-space*) *topological-space*
begin

definition *open-finmap* :: $('a \Rightarrow_F 'b)\ set \Rightarrow bool$ **where**
[*code del*]: *open-finmap* = *generate-topology* $\{Pi'\ a\ b | a\ b.\ \forall i{\in}a.\ open\ (b\ i)\}$

lemma *open-Pi'I*: $(\bigwedge i.\ i \in I \Longrightarrow open\ (A\ i)) \Longrightarrow open\ (Pi'\ I\ A)$
by (*auto intro*: *generate-topology.Basis simp*: *open-finmap-def*)

instance using *topological-space-generate-topology*
by *intro-classes* (*auto simp*: *open-finmap-def class.topological-space-def*)

end

lemma *open-restricted-space*:
shows *open* $\{m.\ P\ (domain\ m)\}$
proof −
have $\{m.\ P\ (domain\ m)\} = (\bigcup i \in Collect\ P.\ \{m.\ domain\ m = i\})$ **by** *auto*
also have *open* ...
proof (*rule*, *safe*, *cases*)
fix *i*::'*a set*

```
    assume finite i
    hence {m. domain m = i} = Pi' i (λ-. 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 ¬ 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 λx. ¬ P x]
  unfolding closed-def by (rule back-subst) auto

lemma tendsto-proj: ((λx. x) ⟶ a) F ⟹ ((λx. (x)_F i) ⟶ (a)_F i) F
  unfolding tendsto-def
proof safe
  fix S::'b set
  let ?S = Pi' (domain a) (λx. if x = i then S else UNIV)
  assume open S hence open ?S by (auto intro!: open-Pi'I)
  moreover assume ∀ S. open S ⟶ a ∈ S ⟶ eventually (λx. x ∈ S) F a i ∈ S
  ultimately have eventually (λx. x ∈ ?S) F by auto
  thus eventually (λx. (x)_F i ∈ S) F
    by (eventually-elim (insert ‹a i ∈ S›, force simp: Pi'-iff split: if-split-asm)
qed

lemma continuous-proj:
  shows continuous-on s (λx. (x)_F i)
  unfolding continuous-on-def by (safe intro!: tendsto-proj tendsto-ident-at)

instance finmap :: (type, first-countable-topology) first-countable-topology
proof
  fix x::'a⇒_F'b
  have ∀ i. ∃ A. countable A ∧ (∀ a∈A. x i ∈ a) ∧ (∀ a∈A. open a) ∧
    (∀ S. open S ∧ x i ∈ S ⟶ (∃ a∈A. a ⊆ S)) ∧ (∀ a b. a ∈ A ⟶ b ∈ A ⟶ a
∩ b ∈ A) (is ∀ i. ?th i)
  proof
    fix i from first-countable-basis-Int-stableE[of x i] guess A .
    thus ?th i by (intro exI[where x=A]) simp
  qed
  then guess A unfolding choice-iff .. note A = this
  hence open-sub: ⋀i S. i∈domain x ⟹ open (S i) ⟹ x i∈(S i) ⟹ (∃ a∈A i.
a⊆(S i)) by auto
  have A-notempty: ⋀i. i ∈ domain x ⟹ A i ≠ {} using open-sub[of - λ-. UNIV]
by auto
```

let *?A* = (λ*f*. *Pi′* (*domain x*) *f*) ‘ (*Pi*$_E$ (*domain x*) *A*)
show ∃ *A*::*nat* ⇒ (′*a*⇒$_F$ ′*b*) *set*. (∀ *i*. *x* ∈ (*A i*) ∧ *open* (*A i*)) ∧ (∀ *S*. *open S* ∧ *x* ∈ *S* ⟶ (∃ *i*. *A i* ⊆ *S*))
proof (*rule first-countableI*[**where** *A*=*?A*], *safe*)
show *countable ?A* **using** *A* **by** (*simp add*: *countable-PiE*)
next
fix *S*::(′*a* ⇒$_F$ ′*b*) *set* **assume** *open S x* ∈ *S*
thus ∃ *a*∈*?A*. *a* ⊆ *S* **unfolding** *open-finmap-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 ∈ *Pi*$_E$ (*domain x*) *A Pi′* (*domain x*) *f* ⊆ *a g* ∈ *Pi*$_E$ (*domain x*) *A Pi′* (*domain x*) *g* ⊆ *b*
by *auto*
thus *?case* **using** *A*
by (*auto simp*: *Pi′-iff PiE-iff extensional-def Int-stable-def*
intro!: *bexI*[**where** *x*=λ*i*. *f i* ∩ *g i*])
next
case (*UN B*)
then obtain *b* **where** *x* ∈ *b b* ∈ *B* **by** *auto*
hence ∃ *a*∈*?A*. *a* ⊆ *b* **using** *UN* **by** *simp*
thus *?case* **using** ‹*b* ∈ *B*› **by** *blast*
next
case (*Basis s*)
then obtain *a b* **where** *xs*: *x*∈ *Pi′ a b s* = *Pi′ a b* ⋀*i*. *i*∈*a* ⟹ *open* (*b i*) **by** *auto*
have ∀ *i*. ∃ *a*. (*i* ∈ *domain x* ∧ *open* (*b i*) ∧ (*x*)$_F$ *i* ∈ *b i*) ⟶ (*a*∈*A i* ∧ *a* ⊆ *b i*)
using *open-sub*[*of* - *b*] **by** *auto*
then obtain *b′*
where ⋀*i*. *i* ∈ *domain x* ⟹ *open* (*b i*) ⟹ (*x*)$_F$ *i* ∈ *b i* ⟹ (*b′ i* ∈*A i* ∧ *b′ i* ⊆ *b i*)
unfolding *choice-iff* **by** *auto*
with *xs* **have** ⋀*i*. *i* ∈ *a* ⟹ (*b′ i* ∈*A i* ∧ *b′ i* ⊆ *b i*) *Pi′ a b′* ⊆ *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*=*restrict b′* (*domain x*)])
qed
qed (*insert A*,*auto simp*: *PiE-iff intro*!: *open-Pi′I*)
qed

13.6 Metric Space of Finite Maps

instantiation *finmap* :: (*type*, *metric-space*) *dist*
begin

definition *dist-finmap* **where**
dist P Q = Max (range (λi. dist ((P)$_F$ i) ((Q)$_F$ i))) + (if domain P = domain Q then 0 else 1)

instance ..
end

instantiation *finmap* :: (*type*, *metric-space*) *uniformity-dist*
begin

definition [*code del*]:
(uniformity :: (('*a*, '*b*) *finmap* × ('*a*, '*b*) *finmap*) *filter*) =
(INF e:{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

instance
by *standard* (*rule uniformity-finmap-def*)
end

declare *uniformity-Abort*[**where** '*a*=('*a*, '*b*::*metric-space*) *finmap*, *code*]

instantiation *finmap* :: (*type*, *metric-space*) *metric-space*
begin

lemma *finite-proj-image'*: *x* ∉ *domain P* ⟹ *finite* ((*P*)$_F$ ‘ *S*)
by (*rule finite-subset*[*of - proj P* ‘ (*domain P* ∩ *S* ∪ {*x*})]) *auto*

lemma *finite-proj-image*: *finite* ((*P*)$_F$ ‘ *S*)
by (*cases* ∃*x*. *x* ∉ *domain P*) (*auto intro*: *finite-proj-image' finite-subset*[**where** *B*=*domain P*])

lemma *finite-proj-diag*: *finite* ((λ*i*. *d* ((*P*)$_F$ *i*) ((*Q*)$_F$ *i*)) ‘ *S*)
proof −
have (λ*i*. *d* ((*P*)$_F$ *i*) ((*Q*)$_F$ *i*)) ‘ *S* = (λ(*i*, *j*). *d i j*) ‘ ((λ*i*. ((*P*)$_F$ *i*, (*Q*)$_F$ *i*)) ‘ *S*) **by** *auto*
moreover have ((λ*i*. ((*P*)$_F$ *i*, (*Q*)$_F$ *i*)) ‘ *S*) ⊆ (λ*i*. (*P*)$_F$ *i*) ‘ *S* × (λ*i*. (*Q*)$_F$ *i*) ‘ *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 *dist P Q* < *1* ⟹ *domain P* = *domain Q*
by (*simp add*: *dist-finmap-def finite-proj-diag split*: *if-split-asm*)

lemma *dist-proj*:
shows *dist* ((*x*)$_F$ *i*) ((*y*)$_F$ *i*) ≤ *dist x y*

```
proof −
  have dist (x i) (y i) ≤ Max (range (λi. dist (x i) (y i)))
    by (simp add: Max-ge-iff finite-proj-diag)
  also have ... ≤ dist x y by (simp add: dist-finmap-def)
  finally show ?thesis .
qed

lemma dist-finmap-lessI:
  assumes domain P = domain Q
  assumes 0 < e
  assumes ⋀i. i ∈ domain P ⟹ dist (P i) (Q i) < e
  shows dist P Q < e
proof −
  have dist P Q = Max (range (λi. dist (P i) (Q i)))
    using assms by (simp add: dist-finmap-def finite-proj-diag)
  also have ... < e
  proof (subst Max-less-iff, safe)
    fix i
    show 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 ⇒_F 'b) set
  have ∗: open S = (∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S) (is - = ?od)
  proof
    assume open S
    thus ?od
      unfolding open-finmap-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 ∈ a x ∈ b
        with Int x obtain e1 e2 where
          e1>0 ∀ y. dist y x < e1 ⟶ y ∈ a e2>0 ∀ y. dist y x < e2 ⟶ y ∈ b by
force
        thus ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ a ∩ b
          by (auto intro!: exI[where x=min e1 e2])
      qed
    next
      case (UN K)
      show ?case
      proof safe
```

```
      fix x X assume x ∈ X and X: X ∈ K
      with UN obtain e where e>0 ⋀y. dist y x < e ⟶ y ∈ X by force
      with X show ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ ⋃ K by auto
    qed
  next
    case (Basis s) then obtain a b where s: s = Pi' a b and b: ⋀i. i∈a ⟹
open (b i) by auto
    show ?case
    proof safe
      fix x assume x ∈ s
      hence [simp]: finite a and a-dom: a = domain x using s by (auto simp:
Pi'-iff)
      obtain es where es: ∀ i ∈ a. es i > 0 ∧ (∀ y. dist y (proj x i) < es i ⟶
y ∈ b i)
        using b ‹x ∈ s› by atomize-elim (intro bchoice, auto simp: open-dist s)
      hence in-b: ⋀i y. i ∈ a ⟹ dist y (proj x i) < es i ⟹ y ∈ b i by auto
      show ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ s
      proof (cases, rule, safe)
        assume a ≠ {}
        show 0 < min 1 (Min (es ‘ a)) using es by (auto simp: ‹a ≠ {}›)
        fix y assume d: dist y x < min 1 (Min (es ‘ a))
        show y ∈ s unfolding s
        proof
      show domain y = a using d s ‹a ≠ {}› by (auto simp: dist-le-1-imp-domain-eq
a-dom)
          fix i assume i: i ∈ a
          hence dist ((y)_F i) ((x)_F i) < es i using d
        by (auto simp: dist-finmap-def ‹a ≠ {}› intro!: le-less-trans[OF dist-proj])
          with i show y i ∈ b i by (rule in-b)
        qed
      next
        assume ¬a ≠ {}
        thus ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ s
          using s ‹x ∈ s› by (auto simp: Pi'-def dist-le-1-imp-domain-eq intro!:
exI[where x=1])
      qed
    qed
  qed
  next
    assume ∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S
    then obtain e where e-pos: ⋀x. x ∈ S ⟹ e x > 0 and
      e-in: ⋀x y . x ∈ S ⟹ dist y x < e x ⟹ y ∈ S
      unfolding bchoice-iff
      by auto
    have S-eq: S = ⋃{Pi' a b| a b. ∃ x∈S. domain x = a ∧ b = (λi. ball (x i) (e
x))}
    proof safe
      fix x assume x ∈ S
      thus x ∈ ⋃{Pi' a b| a b. ∃ x∈S. domain x = a ∧ b = (λi. ball (x i) (e x))}
```

using *e-pos* **by** (*auto intro*!: *exI*[**where** *x*=*Pi'* (*domain x*) (λ*i*. *ball* (*x i*) (*e x*))])
next
fix *x y*
assume *y* ∈ *S*
moreover
assume *x* ∈ (Π' *i*∈*domain y*. *ball* (*y i*) (*e y*))
hence *dist x y* < *e y* **using** *e-pos* ‹*y* ∈ *S*›
by (*auto simp*: *dist-finmap-def Pi'-iff finite-proj-diag dist-commute*)
ultimately show *x* ∈ *S* **by** (*rule e-in*)
qed
also have *open* ...
unfolding *open-finmap-def*
by (*intro generate-topology.UN*) (*auto intro*: *generate-topology.Basis*)
finally show *open S* **.**
qed
show *open S* = (∀ *x*∈*S*. ∀ *F* (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *S*)
unfolding ∗ *eventually-uniformity-metric*
by (*simp del*: *split-paired-All add*: *dist-finmap-def dist-commute eq-commute*)
next
fix *P Q*::'*a* ⇒*F* '*b*
have *Max-eq-iff*: ⋀*A m*. *finite A* ⟹ *A* ≠ {} ⟹ (*Max A* = *m*) = (*m* ∈ *A* ∧ (∀ *a*∈*A*. *a* ≤ *m*))
by (*auto intro*: *Max-in Max-eqI*)
show *dist P Q* = *0* ⟷ *P* = *Q*
by (*auto simp*: *finmap-eq-iff dist-finmap-def Max-ge-iff finite-proj-diag Max-eq-iff add-nonneg-eq-0-iff intro*!: *Max-eqI image-eqI*[**where** *x*=*undefined*])
next
fix *P Q R*::'*a* ⇒*F* '*b*
let *?dists* = λ*P Q i*. *dist* ((*P*)*F i*) ((*Q*)*F i*)
let *?dpq* = *?dists P Q* **and** *?dpr* = *?dists P R* **and** *?dqr* = *?dists Q R*
let *?dom* = λ*P Q*. (*if domain P* = *domain Q then 0 else 1*::*real*)
have *dist P Q* = *Max* (*range ?dpq*) + *?dom P Q*
by (*simp add*: *dist-finmap-def*)
also obtain *t* **where** *t* ∈ *range ?dpq t* = *Max* (*range ?dpq*) **by** (*simp add*: *finite-proj-diag*)
then obtain *i* **where** *Max* (*range ?dpq*) = *?dpq i* **by** *auto*
also have *?dpq i* ≤ *?dpr i* + *?dqr i* **by** (*rule dist-triangle2*)
also have *?dpr i* ≤ *Max* (*range ?dpr*) **by** (*simp add*: *finite-proj-diag*)
also have *?dqr i* ≤ *Max* (*range ?dqr*) **by** (*simp add*: *finite-proj-diag*)
also have *?dom P Q* ≤ *?dom P R* + *?dom Q R* **by** *simp*
finally show *dist P Q* ≤ *dist P R* + *dist Q R* **by** (*simp add*: *dist-finmap-def ac-simps*)
qed

end

13.7 Complete Space of Finite Maps

lemma *tendsto-finmap*:
fixes f::*nat* ⇒ ('*i* ⇒$_F$ ('*a*::*metric-space*))
assumes *ind-f*: ⋀*n*. *domain* (*f n*) = *domain g*
assumes *proj-g*: ⋀*i*. *i* ∈ *domain g* ⟹ (λ*n*. (*f n*) *i*) ⟶ *g i*
shows *f* ⟶ *g*
unfolding *tendsto-iff*
proof *safe*
fix *e*::*real* **assume** *0* < *e*
let *?dists* = λ*x i*. *dist* ((*f x*)$_F$ *i*) ((*g*)$_F$ *i*)
have *eventually* (λ*x*. ∀ *i*∈*domain g*. *?dists x i* < *e*) *sequentially*
using *finite-domain*[*of g*] *proj-g*
proof *induct*
case (*insert i G*)
with ‹*0* < *e*› **have** *eventually* (λ*x*. *?dists x i* < *e*) *sequentially* **by** (*auto simp add*: *tendsto-iff*)
moreover
from *insert* **have** *eventually* (λ*x*. ∀ *i*∈*G*. *dist* ((*f x*)$_F$ *i*) ((*g*)$_F$ *i*) < *e*) *sequentially* **by** *simp*
ultimately show *?case* **by** *eventually-elim auto*
qed *simp*
thus *eventually* (λ*x*. *dist* (*f x*) *g* < *e*) *sequentially*
by *eventually-elim* (*auto simp add*: *dist-finmap-def finite-proj-diag ind-f* ‹*0* < *e*›)
qed

instance *finmap* :: (*type*, *complete-space*) *complete-space*
proof
fix *P*::*nat* ⇒ '*a* ⇒$_F$ '*b*
assume *Cauchy P*
then obtain *Nd* **where** *Nd*: ⋀*n*. *n* ≥ *Nd* ⟹ *dist* (*P n*) (*P Nd*) < *1*
by (*force simp*: *cauchy*)
def *d* ≡ *domain* (*P Nd*)
with *Nd* **have** *dim*: ⋀*n*. *n* ≥ *Nd* ⟹ *domain* (*P n*) = *d* **using** *dist-le-1-imp-domain-eq* **by** *auto*
have [*simp*]: *finite d* **unfolding** *d-def* **by** *simp*
def *p* ≡ λ*i n*. (*P n*) *i*
def *q* ≡ λ*i*. *lim* (*p i*)
def *Q* ≡ *finmap-of d q*
have *q*: ⋀*i*. *i* ∈ *d* ⟹ *q i* = *Q i* **by** (*auto simp add*: *Q-def Abs-finmap-inverse*)
{
fix *i* **assume** *i* ∈ *d*
have *Cauchy* (*p i*) **unfolding** *cauchy p-def*
proof *safe*
fix *e*::*real* **assume** *0* < *e*
with ‹*Cauchy P*› **obtain** *N* **where** *N*: ⋀*n*. *n*≥*N* ⟹ *dist* (*P n*) (*P N*) < *min e 1*
by (*force simp*: *cauchy min-def*)
hence ⋀*n*. *n* ≥ *N* ⟹ *domain* (*P n*) = *domain* (*P N*) **using** *dist-le-1-imp-domain-eq*

by *auto*
with *dim* **have** *dim*: $\bigwedge n.\ n \geq N \Longrightarrow$ *domain* $(P\ n) = d$ **by** (*metis nat-le-linear*)
show $\exists N.\ \forall n{\geq}N.$ *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 *dist* $((P\ n)\ i)\ ((P\ N)\ i) \leq$ *dist* $(P\ n)\ (P\ N)$
using *dim*[*OF* ‹$N \leq n$›] *dim*[*OF* ‹$N \leq N$›] ‹$i \in d$›
by (*auto intro*!: *dist-proj*)
also have $\ldots < e$ **using** *N*[*OF* ‹$N \leq n$›] **by** *simp*
finally show *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 $\exists ni.\ \forall i{\in}d.\ \forall n{\geq}ni\ i.$ *dist* $(p\ i\ n)\ (q\ i) < e$
proof (*safe intro*!: *bchoice*)
fix i **assume** $i \in d$
from *p*[*OF* ‹$i \in d$›, *THEN metric-LIMSEQ-D*, *OF* ‹$0 < e$›]
show $\exists no.\ \forall n{\geq}no.$ *dist* $(p\ i\ n)\ (q\ i) < e$ **.**
qed **then guess** *ni* **..** **note** $ni = this$
def $N \equiv max\ Nd\ (Max\ (ni\ `\ d))$
show $\exists N.\ \forall n{\geq}N.$ *dist* $(P\ n)\ Q < e$
proof (*safe intro*!: *exI*[**where** $x{=}N$])
fix n **assume** $N \leq n$
hence *dom*: *domain* $(P\ n) = d$ *domain* $Q = d$ *domain* $(P\ n) =$ *domain* Q
using *dim* **by** (*simp-all add*: *N-def Q-def dim-def Abs-finmap-inverse*)
show *dist* $(P\ n)\ Q < e$
proof (*rule dist-finmap-lessI*[*OF dom*(*3*) ‹$0 < e$›])
fix i
assume $i \in$ *domain* $(P\ n)$
hence $ni\ i \leq Max\ (ni\ `\ d)$ **using** *dom* **by** *simp*
also have $\ldots \leq N$ **by** (*simp add*: *N-def*)
finally show *dist* $((P\ n)_F\ i)\ ((Q)_F\ i) < e$ **using** *ni* ‹$i \in$ *domain* $(P\ n)$› ‹$N \leq n$› *dom*
by (*auto simp*: *p-def q N-def less-imp-le*)
qed
qed
qed
thus *convergent P* **by** (*auto simp*: *convergent-def*)
qed

13.8 Second Countable Space of Finite Maps

instantiation *finmap* :: (*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** ⋀*i*. *i* ∈ *I* ⟹ *S i* ∈ *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* = (λ*f*. *Pi'* (*domain f*) (λ*i*. *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* (λ*x*. *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*: *basis-proj* = {} ⟹ *basis-finmap* = {*Pi'* {} *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'* ∈ *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': *open* O' $x \in O'$
then obtain a **where** a:
$x \in Pi'$ (*domain* x) a Pi' (*domain* x) $a \subseteq O' \bigwedge i.\ i \in domain\ x \Longrightarrow open\ (a\ i)$
unfolding *open-finmap-def*
proof (*atomize-elim*, *induct rule*: *generate-topology.induct*)
case (*Int* a b)
let *?p*=$\lambda a\ f.\ x \in Pi'$ (*domain* x) $f \wedge Pi'$ (*domain* x) $f \subseteq a \wedge (\forall i.\ i \in domain\ x \longrightarrow open\ (f\ i))$
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** $x \in kk$ $kk \in k$ $x \in Pi'$ (*domain* x) a Pi' (*domain* x) $a \subseteq kk$
$\bigwedge i.\ i \in domain\ x \Longrightarrow open\ (a\ i)$
by *force*
thus *?case* **by** *blast*
qed (*auto simp*: *Pi'-def*)
have $\exists B.$
$(\forall i \in domain\ x.\ x\ i \in B\ i \wedge B\ i \subseteq a\ i \wedge B\ i \in basis\text{-}proj)$
proof (*rule bchoice*, *safe*)
fix i **assume** $i \in domain\ x$
hence *open* (a i) $x\ i \in a\ i$ **using** a **by** *auto*
from *topological-basisE*[*OF basis-proj this*] **guess** b' **.**
thus $\exists y.\ x\ i \in y \wedge y \subseteq a\ i \wedge y \in basis\text{-}proj$ **by** *auto*
qed
then guess B **.. note** $B = this$
def $B' \equiv Pi'$ (*domain* x) ($\lambda i.\ (B\ i)$::$'b$ *set*)
have $B' \subseteq Pi'$ (*domain* x) a **using** B **by** (*auto intro*!: *Pi'-mono simp*: *B'-def*)
also note ‹$\ldots \subseteq O'$›
finally show $\exists B' \in basis\text{-}finmap.\ x \in B' \wedge B' \subseteq O'$ **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 $x \in basis\text{-}finmap$
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

13.9 Polish Space of Finite Maps

instance *finmap* :: (*countable*, *polish-space*) *polish-space* **proof qed**

13.10 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* ≡ *PiF I M*

syntax
-*PiF* :: *pttrn* ⇒ *'i set* ⇒ *'a measure* ⇒ (*'i* => *'a*) *measure* ((3Π_F -∈-./ -) 10)
translations
Π_F *x*∈*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 (⋃ *J* ∈ *I*. (Π′ *j*∈*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* (⋃ *J* ∈ *I*. (Π′ *j*∈*J*. *space* (*M j*)))
{(Π′ *j*∈*J*. *X j*) |*X J*. *J* ∈ *I* ∧ *X* ∈ (Π *j*∈*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* (Π′ *j*∈*I*. *space* (*M j*))
{(Π′ *j*∈*I*. *X j*) |*X*. *X* ∈ (Π *j*∈*I*. *sets* (*M j*))}
unfolding *sets-PiF* **by** *simp*

lemma *in-sets-PiFI*:
assumes *X* = (*Pi′ J S*) *J* ∈ *I* ⋀*i*. *i*∈*J* ⟹ *S i* ∈ *sets* (*M i*)
shows *X* ∈ *sets* (*PiF I M*)
unfolding *sets-PiF*
using *assms* **by** *blast*

lemma *product-in-sets-PiFI*:
assumes *J* ∈ *I* ⋀*i*. *i*∈*J* ⟹ *S i* ∈ *sets* (*M i*)
shows (*Pi′ J S*) ∈ *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 *space* $(PiF\ \{J\}\ M) \in sets\ (PiF\ I\ M)$
using *assms*
by (*intro in-sets-PiFI*[**where** $J{=}J$ **and** $S{=}\lambda i.\ space\ (M\ i)$])
(*auto simp*: *product-def space-PiF*)

lemma *singleton-subspace-set-in-sets*:
assumes *A*: $A \in sets\ (PiF\ \{J\}\ M)$
assumes *finite* J
assumes $J \in I$
shows $A \in sets\ (PiF\ I\ M)$
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 $\bigwedge J.\ J \in I \Longrightarrow finite\ J$
assumes *MN*: $\bigwedge J.\ J \in I \Longrightarrow A \in measurable\ (PiF\ \{J\}\ M)\ N$
shows $A \in measurable\ (PiF\ I\ M)\ N$
unfolding *measurable-def*
proof *safe*
fix *y* **assume** $y \in sets\ N$
have $A -` y \cap space\ (PiF\ I\ M) = (\bigcup J{\in}I.\ A -` y \cap space\ (PiF\ \{J\}\ M))$
by (*auto simp*: *space-PiF*)
also have $\ldots \in sets\ (PiF\ I\ M)$
proof (*rule sets.finite-UN*)
show *finite I* **by** *fact*
fix *J* **assume** $J \in I$
with *assms* **have** *finite J* **by** *simp*
show $A -` y \cap space\ (PiF\ \{J\}\ M) \in sets\ (PiF\ I\ M)$
by (*rule singleton-subspace-set-in-sets*[*OF measurable-sets*[*OF assms*(*3*)]])
fact+
qed
finally show $A -` y \cap space\ (PiF\ I\ M) \in sets\ (PiF\ I\ M)$ **.**
next
fix *x* **assume** $x \in space\ (PiF\ I\ M)$ **thus** $A\ x \in space\ N$
using *MN*[*of domain x*]
by (*auto simp*: *space-PiF measurable-space Pi'-def*)
qed

lemma *countable-finite-comprehension*:
fixes *f* :: *'a*::*countable set* ⇒ -
assumes $\bigwedge s.\ P\ s \Longrightarrow finite\ s$
assumes $\bigwedge s.\ P\ s \Longrightarrow f\ s \in sets\ M$
shows $\bigcup \{f\ s | s.\ P\ s\} \in sets\ M$

proof −
have $\bigcup\{f\ s|s.\ P\ s\} = (\bigcup n{::}nat.\ let\ s = set\ (from\text{-}nat\ n)\ in\ if\ P\ s\ then\ f\ s\ else\ \{\})$
proof *safe*
fix *x X s* **assume** ∗: $x \in f\ s\ P\ s$
with *assms* **obtain** *l* **where** $s = set\ l$ **using** *finite-list* **by** *blast*
with ∗ **show** $x \in (\bigcup n.\ let\ s = set\ (from\text{-}nat\ n)\ in\ if\ P\ s\ then\ f\ s\ else\ \{\})$
using ‹*P s*›
by (*auto intro*!: *exI*[**where** *x*=*to-nat l*])
next
fix *x n* **assume** $x \in (let\ s = set\ (from\text{-}nat\ n)\ in\ if\ P\ s\ then\ f\ s\ else\ \{\})$
thus $x \in \bigcup\{f\ s|s.\ P\ s\}$ **using** *assms* **by** (*auto simp*: *Let-def split*: *if-split-asm*)
qed
hence $\bigcup\{f\ s|s.\ P\ s\} = (\bigcup n.\ let\ s = set\ (from\text{-}nat\ n)\ in\ if\ P\ s\ then\ f\ s\ else\ \{\})$
by *simp*
also have $\ldots \in sets\ M$ **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 $\bigwedge j.\ j \in J \Longrightarrow finite\ j$
shows $space\ (PiF\ J\ M) \in sets\ (PiF\ I\ M)$
proof −
have $space\ (PiF\ J\ M) = \bigcup\{space\ (PiF\ \{j\}\ M)|j.\ j \in J\}$
unfolding *space-PiF* **by** *blast*
also have $\ldots \in sets\ (PiF\ I\ M)$ **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*::*countable set set*
assumes *A*: $A \in sets\ (PiF\ J\ M)$
assumes $J \subseteq I$
assumes $\bigwedge j.\ j \in J \Longrightarrow finite\ j$
shows $A \in sets\ (PiF\ I\ M)$
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*!: *space-subset-in-sets*)

lemma *countable-measurable-PiFI*:
fixes *I*::*'a*::*countable set set*
assumes *MN*: $\bigwedge J.\ J \in I \Longrightarrow finite\ J \Longrightarrow A \in measurable\ (PiF\ \{J\}\ M)\ N$
shows $A \in measurable\ (PiF\ I\ M)\ N$
unfolding *measurable-def*

proof *safe*
fix y **assume** $y \in sets\ N$
have $A -` y = (\bigcup\{A -` y \cap \{x.\ domain\ x = J\} | J.\ finite\ J\})$ **by** *auto*
{ fix $x::'a \Rightarrow_F 'b$
from *finite-list*[*of domain x*] **obtain** *xs* **where** *set xs = domain x* **by** *auto*
hence $\exists n.\ domain\ x = set\ (from\text{-}nat\ n)$
by (*intro exI*[**where** *x=to-nat xs*]) *auto* **}**
hence $A -` y \cap space\ (PiF\ I\ M) = (\bigcup n.\ A -` y \cap space\ (PiF\ (\{set\ (from\text{-}nat\ n)\} \cap I)\ M))$
by (*auto simp*: *space-PiF Pi'-def*)
also have $\ldots \in sets\ (PiF\ I\ M)$
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* ‹$y \in sets\ N$›
apply (*auto simp*: *space-PiF*)
done
finally show $A -` y \cap space\ (PiF\ I\ M) \in sets\ (PiF\ I\ M)$ **.**
next
fix x **assume** $x \in space\ (PiF\ I\ M)$ **thus** $A\ x \in space\ N$
using *MN*[*of domain x*] **by** (*auto simp*: *space-PiF measurable-space Pi'-def*)
qed

lemma *measurable-PiF*:
assumes *f*: $\bigwedge x.\ x \in space\ N \Longrightarrow domain\ (f\ x) \in I \wedge (\forall i \in domain\ (f\ x).\ (f\ x)\ i \in space\ (M\ i))$
assumes *S*: $\bigwedge J\ S.\ J \in I \Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow S\ i \in sets\ (M\ i)) \Longrightarrow$
$f -` (Pi'\ J\ S) \cap space\ N \in sets\ N$
shows $f \in measurable\ N\ (PiF\ I\ M)$
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*: $A \in sets\ (PiF\ I\ M)$ **and** $J \subseteq I$
shows $A \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M)$
using *A*[*unfolded sets-PiF*]
proof (*induct A*)
case (*Basic a*)
then obtain *K S* **where** *S*: $a = Pi'\ K\ S \wedge K \in I \wedge (\forall i \in K.\ S\ i \in sets\ (M\ i))$
by *auto*
show *?case*
proof *cases*
assume $K \in J$
hence $a \cap \{m.\ domain\ m \in J\} \in \{Pi'\ K\ X \mid X\ K.\ K \in J \wedge X \in (\Pi\ j \in K.$

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 *UNION UNIV a* ∩ {*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*: ⋀*i*. (∃ *x* ∈ *space N*. *i* ∈ *J x*) ⟹ (λ*x*. *f x i*) ∈ *measurable N* (*M i*)
assumes *J*: ⋀*x*. *x* ∈ *space N* ⟹ *J x* ∈ *I* ⋀*x*. *x* ∈ *space N* ⟹ *finite* (*J x*)
assumes *JN*: ⋀*S*. {*x*. *J x* = *S*} ∩ *space N* ∈ *sets N*
shows (λ*x*. *finmap-of* (*J x*) (*f x*)) ∈ *measurable N* (*PiF I M*)
proof (*rule measurable-PiF*)
fix *x* **assume** *x* ∈ *space N*
with *J*[*of x*] *measurable-space*[*OF f*]
show *domain* (*finmap-of* (*J x*) (*f x*)) ∈ *I* ∧
(∀ *i*∈*domain* (*finmap-of* (*J x*) (*f x*)). (*finmap-of* (*J x*) (*f x*)) *i* ∈ *space* (*M i*))
by *auto*
next
fix *K S* **assume** *K* ∈ *I* **and** ∗: ⋀*i*. *i* ∈ *K* ⟹ *S i* ∈ *sets* (*M i*)
with *J* **have** *eq*: (λ*x*. *finmap-of* (*J x*) (*f x*)) −‘ *Pi' K S* ∩ *space N* =
(*if* ∃ *x* ∈ *space N*. *K* = *J x* ∧ *finite K then if K* = {} *then* {*x* ∈ *space N*. *J x* = *K*}
else (⋂ *i*∈*K*. (λ*x*. *f x i*) −‘ *S i* ∩ {*x* ∈ *space N*. *J x* = *K*}) *else* {})
by (*auto simp*: *Pi'-def*)
have *r*: {*x* ∈ *space N*. *J x* = *K*} = *space N* ∩ ({*x*. *J x* = *K*} ∩ *space N*) **by** *auto*
show (λ*x*. *finmap-of* (*J x*) (*f x*)) −‘ *Pi' K S* ∩ *space N* ∈ *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 *finmap-of J* ∈ *measurable* (*Pi*$_M$ *J M*) (*PiF* {*J*} *M*)
apply (*rule measurable-finmap-of*)
apply (*rule measurable-component-singleton*)
apply *simp*
apply *rule*
apply (*rule* ‹*finite J*›)
apply *simp*
done

lemma *proj-measurable-singleton*:
assumes *A* ∈ *sets* (*M i*)
shows (λ*x*. (*x*)$_F$ *i*) −‘ *A* ∩ *space* (*PiF* {*I*} *M*) ∈ *sets* (*PiF* {*I*} *M*)
proof *cases*
assume *i* ∈ *I*
hence (λ*x*. (*x*)$_F$ *i*) −‘ *A* ∩ *space* (*PiF* {*I*} *M*) =
Pi′ *I* (λ*x*. *if x* = *i then A else space* (*M x*))
using *sets.sets-into-space*[*OF*] ‹*A* ∈ *sets* (*M i*)› *assms*
by (*auto simp*: *space-PiF Pi*′*-def*)
thus *?thesis* **using** *assms* ‹*A* ∈ *sets* (*M i*)›
by (*intro in-sets-PiFI*) *auto*
next
assume *i* ∉ *I*
hence (λ*x*. (*x*)$_F$ *i*) −‘ *A* ∩ *space* (*PiF* {*I*} *M*) =
(*if undefined* ∈ *A then space* (*PiF* {*I*} *M*) *else* {}) **by** (*auto simp*: *space-PiF Pi*′*-def*)
thus *?thesis* **by** *simp*
qed

lemma *measurable-proj-singleton*:
assumes *i* ∈ *I*
shows (λ*x*. (*x*)$_F$ *i*) ∈ *measurable* (*PiF* {*I*} *M*) (*M i*)
by (*unfold measurable-def*, *intro CollectI conjI ballI proj-measurable-singleton assms*)
(*insert* ‹*i* ∈ *I*›, *auto simp*: *space-PiF*)

lemma *measurable-proj-countable*:

fixes $I::'a::countable\ set\ set$
assumes $y \in space\ (M\ i)$
shows $(\lambda x.\ if\ i \in domain\ x\ then\ (x)_F\ i\ else\ y) \in measurable\ (PiF\ I\ M)\ (M\ i)$
proof (*rule countable-measurable-PiFI*)
fix J **assume** $J \in I\ finite\ J$
show $(\lambda x.\ if\ i \in domain\ x\ then\ x\ i\ else\ y) \in measurable\ (PiF\ \{J\}\ M)\ (M\ i)$
unfolding *measurable-def*
proof *safe*
fix z **assume** $z \in sets\ (M\ i)$
have $(\lambda x.\ if\ i \in domain\ x\ then\ x\ i\ else\ y) -` z \cap space\ (PiF\ \{J\}\ M) =$
$(\lambda x.\ if\ i \in J\ then\ (x)_F\ i\ else\ y) -` z \cap space\ (PiF\ \{J\}\ M)$
by (*auto simp*: *space-PiF Pi'-def*)
also have $\ldots \in sets\ (PiF\ \{J\}\ M)$ **using** ‹$z \in sets\ (M\ i)$› ‹*finite J*›
by (*cases* $i \in J$) (*auto intro*!: *measurable-sets*[*OF measurable-proj-singleton*])
finally show $(\lambda x.\ if\ i \in domain\ x\ then\ x\ i\ else\ y) -` z \cap space\ (PiF\ \{J\}\ M) \in$
$sets\ (PiF\ \{J\}\ M)$ **.**
qed (*insert* ‹$y \in space\ (M\ i)$›, *auto simp*: *space-PiF Pi'-def*)
qed

lemma *measurable-restrict-proj*:
assumes $J \in II\ finite\ J$
shows *finmap-of* $J \in measurable\ (PiM\ J\ M)\ (PiF\ II\ M)$
using *assms*
by (*intro measurable-finmap-of measurable-component-singleton*) *auto*

lemma *measurable-proj-PiM*:
fixes $J\ K :: 'a::countable\ set$ **and** $I::'a\ set\ set$
assumes *finite* $J\ J \in I$
assumes $x \in space\ (PiM\ J\ M)$
shows $proj \in measurable\ (PiF\ \{J\}\ M)\ (PiM\ J\ M)$
proof (*rule measurable-PiM-single*)
show $proj \in space\ (PiF\ \{J\}\ M) \to (\Pi_E\ i \in J.\ space\ (M\ i))$
using *assms* **by** (*auto simp add*: *space-PiM space-PiF extensional-def sets-PiF Pi'-def*)
next
fix $A\ i$ **assume** A: $i \in J\ A \in sets\ (M\ i)$
show $\{\omega \in space\ (PiF\ \{J\}\ M).\ (\omega)_F\ i \in A\} \in sets\ (PiF\ \{J\}\ M)$
proof
have $\{\omega \in space\ (PiF\ \{J\}\ M).\ (\omega)_F\ i \in A\} =$
$(\lambda\omega.\ (\omega)_F\ i) -` A \cap space\ (PiF\ \{J\}\ M)$ **by** *auto*
also have $\ldots \in sets\ (PiF\ \{J\}\ M)$
using *assms A* **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 *space* (*PiF* {*I*} *M*) = (Π′ *i*∈*I*. *space* (*M i*))
by (*auto simp*: *product-def space-PiF assms*)

adapted from *sets* (*Pi*$_M$ *?I ?M*) = *sigma-sets* (Π$_E$ *i*∈*?I*. *space* (*?M i*)) {{*f* ∈ Π$_E$ *i*∈*?I*. *space* (*?M i*). *f i* ∈ *A*} |*i A*. *i* ∈ *?I* ∧ *A* ∈ *sets* (*?M i*)}

lemma *sets-PiF-single*:
assumes *finite I I* ≠ {}
shows *sets* (*PiF* {*I*} *M*) =
sigma-sets (Π′ *i*∈*I*. *space* (*M i*))
{{*f*∈Π′ *i*∈*I*. *space* (*M i*). *f i* ∈ *A*} | *i A*. *i* ∈ *I* ∧ *A* ∈ *sets* (*M i*)}
(**is** - = *sigma-sets ?*Ω *?R*)
unfolding *sets-PiF-singleton*
proof (*rule sigma-sets-eqI*)
interpret *R*: *sigma-algebra ?*Ω *sigma-sets ?*Ω *?R* **by** (*rule sigma-algebra-sigma-sets*) *auto*
fix *A* **assume** *A* ∈ {*Pi′ I X* |*X*. *X* ∈ (Π *j*∈*I*. *sets* (*M j*))}
then obtain *X* **where** *X*: *A* = *Pi′ I X X* ∈ (Π *j*∈*I*. *sets* (*M j*)) **by** *auto*
show *A* ∈ *sigma-sets ?*Ω *?R*
proof −
from ‹*I* ≠ {}› *X* **have** *A* = (⋂*j*∈*I*. {*f*∈*space* (*PiF* {*I*} *M*). *f j* ∈ *X j*})
using *sets.sets-into-space*
by (*auto simp*: *space-PiF product-def*) *blast*
also have ... ∈ *sigma-sets ?*Ω *?R*
using *X* ‹*I* ≠ {}› *assms* **by** (*intro R.finite-INT*) (*auto simp*: *space-PiF*)
finally show *A* ∈ *sigma-sets ?*Ω *?R* **.**
qed
next
fix *A* **assume** *A* ∈ *?R*
then obtain *i B* **where** *A*: *A* = {*f*∈Π′ *i*∈*I*. *space* (*M i*). *f i* ∈ *B*} *i* ∈ *I B* ∈ *sets* (*M i*)
by *auto*
then have *A* = (Π′ *j* ∈ *I*. *if j* = *i then B 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 ... ∈ *sigma-sets ?*Ω {*Pi′ I X* |*X*. *X* ∈ (Π *j*∈*I*. *sets* (*M j*))}
using *A*
by (*intro sigma-sets.Basic*)
(*auto intro*: *exI*[**where** *x*=λ*j*. *if j* = *i then B else space* (*M j*)])
finally show *A* ∈ *sigma-sets ?*Ω {*Pi′ I X* |*X*. *X* ∈ (Π *j*∈*I*. *sets* (*M j*))} **.**
qed

adapted from (⋀*i*. *i* ∈ *?I* ⟹ *?A i* = *?B i*) ⟹ *Pi*$_E$ *?I ?A* = *Pi*$_E$ *?I ?B*

lemma *Pi′-cong*:
assumes *finite I*
assumes ⋀*i*. *i* ∈ *I* ⟹ *f i* = *g i*
shows *Pi′ I f* = *Pi′ I g*

using *assms* **by** (*auto simp*: *Pi'-def*)

adapted from ⟦*finite ?I*; ⋀*i n m*. ⟦*i* ∈ *?I*; *n* ≤ *m*⟧ ⟹ *?A n i* ⊆ *?A m i*⟧ ⟹ (⋃*n Pi ?I* (*?A n*)) = (Π *i*∈*?I*. ⋃*n ?A n i*)

lemma *Pi'-UN*:
fixes *A* :: *nat* ⇒ *'i* ⇒ *'a set*
assumes *finite I*
assumes *mono*: ⋀*i n m*. *i* ∈ *I* ⟹ *n* ≤ *m* ⟹ *A n i* ⊆ *A m i*
shows (⋃*n*. *Pi' I* (*A n*)) = *Pi' I* (λ*i*. ⋃*n*. *A n i*)
proof (*intro set-eqI iffI*)
fix *f* **assume** *f* ∈ *Pi' I* (λ*i*. ⋃*n*. *A n i*)
then have ∀ *i*∈*I*. ∃ *n*. *f i* ∈ *A n i domain f* = *I* **by** (*auto simp*: ‹*finite I*› *Pi'-def*)
from *bchoice*[*OF this*(*1*)] **obtain** *n* **where** *n*: ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ (*A* (*n i*) *i*)
by *auto*
obtain *k* **where** *k*: ⋀*i*. *i* ∈ *I* ⟹ *n i* ≤ *k*
using ‹*finite I*› *finite-nat-set-iff-bounded-le*[*of n'I*] **by** *auto*
have *f* ∈ *Pi' I* (λ*i*. *A k i*)
proof
fix *i* **assume** *i* ∈ *I*
from *mono*[*OF this*, *of n i k*] *k*[*OF this*] *n*[*OF this*] ‹*domain f* = *I*› ‹*i* ∈ *I*›
show *f i* ∈ *A k i* **by** (*auto simp*: ‹*finite I*›)
qed (*simp add*: ‹*domain f* = *I*› ‹*finite I*›)
then show *f* ∈ (⋃*n*. *Pi' I* (*A n*)) **by** *auto*
qed (*auto simp*: *Pi'-def* ‹*finite I*›)

adapted from ⟦⋀*i*. *i* ∈ *?I* ⟹ ∃ *S*⊆*?E i*. *countable S* ∧ *?Ω i* = ⋃*S*; ⋀*i*. *i* ∈ *?I* ⟹ *?E i* ⊆ *Pow* (*?Ω i*); ⋀*j*. *j* ∈ *?J* ⟹ *finite j*; ⋃ *?J* = *?I*⟧ ⟹ *sets* (*Pi*$_M$ *?I* (λ*i*. *sigma* (*?Ω i*) (*?E i*))) = *sets* (*sigma* (*Pi*$_E$ *?I ?Ω*) {{*f* ∈ *Pi*$_E$ *?I ?Ω*. ∀ *i*∈*j*. *f i* ∈ *A i*} |*A j*. *j* ∈ *?J* ∧ *A* ∈ *Pi j ?E*})

lemma *sigma-fprod-algebra-sigma-eq*:
fixes *E* :: *'i* ⇒ *'a set set* **and** *S* :: *'i* ⇒ *nat* ⇒ *'a set*
assumes [*simp*]: *finite I I* ≠ {}
and *S-union*: ⋀*i*. *i* ∈ *I* ⟹ (⋃*j*. *S i j*) = *space* (*M i*)
and *S-in-E*: ⋀*i*. *i* ∈ *I* ⟹ *range* (*S i*) ⊆ *E i*
assumes *E-closed*: ⋀*i*. *i* ∈ *I* ⟹ *E i* ⊆ *Pow* (*space* (*M i*))
and *E-generates*: ⋀*i*. *i* ∈ *I* ⟹ *sets* (*M i*) = *sigma-sets* (*space* (*M i*)) (*E i*)
defines *P* == { *Pi' I F* | *F*. ∀ *i*∈*I*. *F i* ∈ *E i* }
shows *sets* (*PiF* {*I*} *M*) = *sigma-sets* (*space* (*PiF* {*I*} *M*)) *P*
proof
let *?P* = *sigma* (*space* (*Pi*$_F$ {*I*} *M*)) *P*
from ‹*finite I*›[*THEN ex-bij-betw-finite-nat*] **guess** *T* **..**
then have *T*: ⋀*i*. *i* ∈ *I* ⟹ *T i* < *card I* ⋀*i*. *i*∈*I* ⟹ *the-inv-into I T* (*T i*) = *i*
by (*auto simp add*: *bij-betw-def set-eq-iff image-iff the-inv-into-f-f simp del*: ‹*finite I*›)
have *P-closed*: *P* ⊆ *Pow* (*space* (*Pi*$_F$ {*I*} *M*))
using *E-closed* **by** (*auto simp*: *space-PiF P-def Pi'-iff subset-eq*)
then have *space-P*: *space ?P* = (Π' *i*∈*I*. *space* (*M i*))
by (*simp add*: *space-PiF*)

```
  have sets (PiF {I} M) =
      sigma-sets (space ?P) {{f ∈ Π' i∈I. space (M i). f i ∈ A} |i A. i ∈ I ∧ A ∈
sets (M i)}
    using sets-PiF-single[of I M] by (simp add: space-P)
  also have ... ⊆ sets (sigma (space (PiF {I} M)) P)
  proof (safe intro!: sets.sigma-sets-subset)
    fix i A assume i ∈ I and A: A ∈ sets (M i)
    have (λx. (x)_F i) ∈ measurable ?P (sigma (space (M i)) (E i))
    proof (subst measurable-iff-measure-of)
      show E i ⊆ Pow (space (M i)) using ‹i ∈ I› by fact
      from space-P ‹i ∈ I› show (λx. (x)_F i) ∈ space ?P → space (M i)
        by auto
      show ∀ A∈E i. (λx. (x)_F i) -` A ∩ space ?P ∈ sets ?P
      proof
        fix A assume A: A ∈ E i
        then have (λx. (x)_F i) -` A ∩ space ?P = (Π' j∈I. if i = j then A else
space (M j))
            using E-closed ‹i ∈ I› by (auto simp: space-P Pi-iff subset-eq split:
if-split-asm)
        also have ... = (Π' j∈I. ⋃ n. if i = j then A else S j n)
          by (intro Pi'-cong) (simp-all add: S-union)
        also have ... = (⋃ xs∈{xs. length xs = card I}. Π' j∈I. if i = j then A
else S j (xs ! T j))
          using T
          apply (auto simp del: Union-iff)
          apply (simp-all add: Pi'-iff bchoice-iff del: Union-iff)
          apply (erule conjE exE)+
          apply (rule-tac x=map (λn. f (the-inv-into I T n)) [0..<card I] in exI)
          apply (auto simp: bij-betw-def)
          done
        also have ... ∈ sets ?P
        proof (safe intro!: sets.countable-UN)
          fix xs show (Π' j∈I. if i = j then A else S j (xs ! T j)) ∈ sets ?P
            using A S-in-E
            by (simp add: P-closed)
              (auto simp: P-def subset-eq intro!: exI[of - λj. if i = j then A else S j
(xs ! T j)])
        qed
        finally show (λx. (x)_F i) -` A ∩ space ?P ∈ sets ?P
          using P-closed by simp
      qed
    qed
    from measurable-sets[OF this, of A] A ‹i ∈ I› E-closed
    have (λx. (x)_F i) -` A ∩ space ?P ∈ sets ?P
      by (simp add: E-generates)
    also have (λx. (x)_F i) -` A ∩ space ?P = {f ∈ Π' i∈I. space (M i). f i ∈ A}
      using P-closed by (auto simp: space-PiF)
    finally show ... ∈ sets ?P .
  qed
```

finally show *sets* (*PiF* {*I*} *M*) ⊆ *sigma-sets* (*space* (*PiF* {*I*} *M*)) *P*
by (*simp add*: *P-closed*)
show *sigma-sets* (*space* (*PiF* {*I*} *M*)) *P* ⊆ *sets* (*PiF* {*I*} *M*)
using ‹*finite I*› ‹*I* ≠ {}›
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* ≠ {}
assumes [*simp*]: *finite I*
shows *sets* (*PiF* {*I*} (λ-. *borel*::'*b*::*second-countable-topology measure*)) =
sigma-sets (*space* (*PiF* {*I*} (λ-. *borel*))) {*Pi'* *I* *F* |*F*. (∀ *i*∈*I*. *F i* ∈ *Collect open*)}
proof −
from *open-countable-basisE*[*OF open-UNIV*] **guess** *S*::'*b set set* **. note** *S* = *this*
show *?thesis*
proof (*rule sigma-fprod-algebra-sigma-eq*)
show *finite I* **by** *simp*
show *I* ≠ {} **by** *fact*
def *S'*≡*from-nat-into S*
show (⋃*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 countable-basis-proj countable-subset from-nat-into-surj*)
done
show *range S'* ⊆ *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 set-mp topological-basis-open*[*OF basis-proj*] *basis-proj-def*)
done
show *Collect open* ⊆ *Pow* (*space borel*) **by** *simp*
show *sets borel* = *sigma-sets* (*space borel*) (*Collect open*)
by (*simp add*: *borel-def*)
qed
qed

lemma *finmap-UNIV*[*simp*]: (⋃*J*∈*Collect finite*. Π' *j*∈*J*. *UNIV*) = *UNIV* **by** *auto*

lemma *borel-eq-PiF-borel*:
shows (*borel* :: ('*i*::*countable* ⇒$_F$ '*a*::*polish-space*) *measure*) =
PiF (*Collect finite*) (λ-. *borel* :: '*a measure*)
unfolding *borel-def PiF-def*
proof (*rule measure-eqI*, *clarsimp*, *rule sigma-sets-eqI*)
fix *a*::('*i* ⇒$_F$ '*a*) *set* **assume** *a* ∈ *Collect open* **hence** *open a* **by** *simp*
then obtain *B'* **where** *B'*: *B'*⊆*basis-finmap a* = ⋃*B'*

```
    using finmap-topological-basis by (force simp add: topological-basis-def)
  have a ∈ sigma UNIV {Pi' J X |X J. finite J ∧ X ∈ J → sigma-sets UNIV
(Collect open)}
    unfolding ‹a = ⋃ B'›
  proof (rule sets.countable-Union)
   from B' countable-basis-finmap show countable B' by (metis countable-subset)
  next
    show B' ⊆ sets (sigma UNIV
      {Pi' J X |X J. finite J ∧ X ∈ J → sigma-sets UNIV (Collect open)}) (is -
⊆ sets ?s)
    proof
      fix x assume x ∈ B' with B' have x ∈ basis-finmap by auto
       then obtain J X where x = Pi' J X finite J X ∈ J → sigma-sets UNIV
(Collect open)
        by (auto simp: basis-finmap-def topological-basis-open[OF basis-proj])
      thus x ∈ sets ?s by auto
    qed
  qed
  thus a ∈ sigma-sets UNIV {Pi' J X |X J. finite J ∧ X ∈ J → sigma-sets UNIV
(Collect open)}
    by simp
next
  fix b::('i ⇒_F 'a) set
  assume b ∈ {Pi' J X |X J. finite J ∧ X ∈ J → sigma-sets UNIV (Collect
open)}
  hence b': b ∈ sets (Pi_F (Collect finite) (λ-. borel)) by (auto simp: sets-PiF
borel-def)
  let ?b = λJ. b ∩ {x. domain x = J}
  have b = ⋃((λJ. ?b J) ` Collect finite) by auto
  also have ... ∈ sets borel
  proof (rule sets.countable-Union, safe)
    fix J::'i set assume finite J
    { assume ef: J = {}
      have ?b J ∈ sets borel
      proof cases
        assume ?b J ≠ {}
        then obtain f where f ∈ b domain f = {} using ef by auto
        hence ?b J = {f} using ‹J = {}›
          by (auto simp: finmap-eq-iff)
        also have {f} ∈ sets borel by simp
        finally show ?thesis .
      qed simp
    } moreover {
      assume J ≠ ({}::'i set)
      have (?b J) = b ∩ {m. domain m ∈ {J}} by auto
      also have ... ∈ sets (PiF {J} (λ-. borel))
        using b' by (rule restrict-sets-measurable) (auto simp: ‹finite J›)
      also have ... = sigma-sets (space (PiF {J} (λ-. borel)))
        {Pi' (J) F |F. (∀j∈J. F j ∈ Collect open)}
```

(**is** - = *sigma-sets* - *?P*)
by (*rule product-open-generates-sets-PiF-single*[*OF* ‹$J \neq \{\}$› ‹*finite J*›])
also have ... $\subseteq$ *sigma-sets UNIV* (*Collect open*)
by (*intro sigma-sets-mono''*) (*auto intro*!: *open-Pi'I simp*: *space-PiF*)
finally have (*?b J*) $\in$ *sets borel* **by** (*simp add*: *borel-def*)
} **ultimately show** (*?b J*) $\in$ *sets borel* **by** *blast*
qed (*simp add*: *countable-Collect-finite*)
finally show *b* $\in$ *sigma-sets UNIV* (*Collect open*) **by** (*simp add*: *borel-def*)
qed (*simp add*: *emeasure-sigma borel-def PiF-def*)

13.11 Isomorphism between Functions and Finite Maps

lemma *measurable-finmap-compose*:
shows (λm. *compose J m f*) $\in$ *measurable* (*PiM* (*f* ‘ *J*) (λ-. *M*)) (*PiM J* (λ-. *M*))
unfolding *compose-def* **by** *measurable*

lemma *measurable-compose-inv*:
assumes *inj*: $\bigwedge j.\ j \in J \Longrightarrow f'\ (f\ j) = j$
shows (λm. *compose* (*f* ‘ *J*) *m f'*) $\in$ *measurable* (*PiM J* (λ-. *M*)) (*PiM* (*f* ‘ *J*) (λ-. *M*))
unfolding *compose-def* **by** (*rule measurable-restrict*) (*auto simp*: *inj*)

locale *function-to-finmap* =
fixes *J*::'*a set* **and** *f* :: '*a* ⇒ '*b*::*countable* **and** *f'*
assumes [*simp*]: *finite J*
assumes *inv*: $i \in J \Longrightarrow f'\ (f\ i) = i$
begin

to measure finmaps

definition *fm* = (*finmap-of* (*f* ‘ *J*)) *o* (λg. *compose* (*f* ‘ *J*) *g f'*)

lemma *domain-fm*[*simp*]: *domain* (*fm x*) = *f* ‘ *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* −‘ *Pi'* (*f* ‘ *J*) *S* ∩ *space* (Pi_M *J M*) = (Π_E *j* ∈ *J*. *S* (*f j*))
using *assms*
by (*auto simp*: *inv fm-def compose-def space-PiM Pi'-def*)

lemma *fm-measurable*:
assumes *f* ‘ *J* ∈ *N*
shows *fm* ∈ *measurable* (Pi_M *J* (λ-. *M*)) (Pi_F *N* (λ-. *M*))
unfolding *fm-def*
proof (*rule measurable-comp*, *rule measurable-compose-inv*)

show *finmap-of* (*f ' J*) ∈ *measurable* (*Pi*$_M$ (*f ' J*) (λ-. *M*)) (*PiF N* (λ-. *M*))
using *assms* **by** (*intro measurable-finmap-of measurable-component-singleton*)
auto
qed (*simp-all add*: *inv*)

lemma *proj-fm*:
assumes *x* ∈ *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* (λ*g*. *compose* (*f ' J*) *g f'*) (*extensional J*)
proof (*rule inj-on-inverseI*)
fix *x*::'*a* ⇒ '*c* **assume** *x* ∈ *extensional J*
thus (λ*x*. *compose J x f*) (*compose* (*f ' J*) *x f'*) = *x*
by (*auto simp*: *compose-def inv extensional-def*)
qed

lemma *inj-on-fm*:
assumes ⋀*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* = (λ*g*. *compose J g f*) *o proj*

lemma *mf-fm*:
assumes *x* ∈ *space* (*Pi*$_M$ *J* (λ-. *M*))
shows *mf* (*fm x*) = *x*
proof −
have *mf* (*fm x*) ∈ *extensional J*
by (*auto simp*: *mf-def extensional-def compose-def*)
moreover
have *x* ∈ *extensional J* **using** *assms sets.sets-into-space*
by (*force simp*: *space-PiM PiE-def*)
moreover
{ fix *i* **assume** *i* ∈ *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 *space M = UNIV*
shows *mf ∈ measurable (PiF {f ‘ J} (λ-. M)) (PiM J (λ-. M))*
unfolding *mf-def*
proof (*rule measurable-comp*, *rule measurable-proj-PiM*)
show (*λg. compose J g f*) *∈ measurable* (*Pi*$_M$ (*f ‘ J*) (*λx. M*)) (*Pi*$_M$ *J* (*λ-. M*))
by (*rule measurable-finmap-compose*)
qed (*auto simp add*: *space-PiM extensional-def assms*)

lemma *fm-image-measurable*:
assumes *space M = UNIV*
assumes *X ∈ sets* (*Pi*$_M$ *J* (*λ-. M*))
shows *fm ‘ X ∈ sets* (*PiF* {*f ‘ J*} (*λ-. M*))
proof −
have *fm ‘ X = (mf) −‘ X ∩ space* (*PiF* {*f ‘ J*} (*λ-. M*))
proof *safe*
fix *x* **assume** *x ∈ X*
with *mf-fm*[*of x*] *sets.sets-into-space*[*OF assms*(*2*)] **show** *fm x ∈ mf −‘ X* **by** *auto*
show *fm x ∈ space* (*PiF* {*f ‘ J*} (*λ-. M*)) **by** (*simp add*: *space-PiF assms*)
next
fix *y x*
assume *x*: *mf y ∈ X*
assume *y*: *y ∈ space* (*PiF* {*f ‘ J*} (*λ-. M*))
thus *y ∈ 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* (*Pi*$_M$ *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 = distr M* (*PiF* (*Collect finite*) *N*) (*fm*)

lemma *sets-mapmeasure*[*simp*]: *sets* (*mapmeasure M N*) = *sets* (*PiF* (*Collect finite*) *N*)
unfolding *mapmeasure-def* **by** *simp*

lemma *space-mapmeasure*[*simp*]: *space* (*mapmeasure M N*) = *space* (*PiF* (*Collect finite*) *N*)
unfolding *mapmeasure-def* **by** *simp*

lemma *mapmeasure-PiF*:
assumes *s1*: *space M* = *space* (*Pi*$_M$ *J* (λ-. *N*))
assumes *s2*: *sets M* = *sets* (*Pi*$_M$ *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* (*Pi*$_M$ *J* (λ-. *N*))
assumes *s2*: *sets M* = (*Pi*$_M$ *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* (*Pi*$_M$ *J* (λ-. *N*)) **using** *assms* **by** (*simp add*: *sets.sets-into-space*)
from *assms inj-on-fm*[*of* λ-. *N*] *set-mp*[*OF this*] **have** *fm* −‘ *fm* ‘ *X* ∩ *space* (*Pi*$_M$ *J* (λ-. *N*)) = *X*
by (*auto simp*: *vimage-image-eq inj-on-def*)
thus *emeasure M X* = *emeasure M* (*fm* −‘ *fm* ‘ *X* ∩ *space M*) **using** *s1*
by *simp*
show *fm* ‘ *X* ∈ *sets* (*PiF* (*Collect finite*) (λ-. *N*))
by (*rule fm-image-measurable-finite*[*OF N X*[*simplified s2*]])
qed *simp*

end

end

14 Regularity of Measures

theory *Regularity*
imports *Measure-Space Borel-Space*
begin

lemma
fixes *M*::′*a*::{*second-countable-topology*, *complete-space*} *measure*
assumes *sb*: *sets M* = *sets borel*
assumes *emeasure M* (*space M*) ≠ ∞
assumes *B* ∈ *sets borel*

```
  shows inner-regular: emeasure M B =
    (SUP K : {K. K ⊆ B ∧ compact K}. emeasure M K) (is ?inner B)
  and outer-regular: emeasure M B =
    (INF U : {U. B ⊆ U ∧ open U}. emeasure M U) (is ?outer B)
proof −
  have Us: UNIV = space M by (metis assms(1) sets-eq-imp-space-eq space-borel)
  hence sU: space M = UNIV by simp
  interpret finite-measure M by rule fact
  have approx-inner: ⋀A. A ∈ sets M ⟹
    (⋀e. e > 0 ⟹ ∃K. K ⊆ A ∧ compact K ∧ emeasure M A ≤ emeasure M K
+ ennreal e) ⟹ ?inner A
    by (rule ennreal-approx-SUP)
     (force intro!: emeasure-mono simp: compact-imp-closed emeasure-eq-measure)+
  have approx-outer: ⋀A. A ∈ sets M ⟹
    (⋀e. e > 0 ⟹ ∃B. A ⊆ B ∧ open B ∧ emeasure M B ≤ emeasure M A +
ennreal e) ⟹ ?outer A
    by (rule ennreal-approx-INF)
       (force intro!: emeasure-mono simp: emeasure-eq-measure sb)+
  from countable-dense-setE guess X::'a set . note X = this
  {
   fix r::real assume r > 0 hence ⋀y. open (ball y r) ⋀y. ball y r ≠ {} by auto
   with X(2)[OF this]
   have x: space M = (⋃x∈X. cball x r)
     by (auto simp add: sU) (metis dist-commute order-less-imp-le)
   let ?U = ⋃k. (⋃n∈{0..k}. cball (from-nat-into X n) r)
    have (λk. emeasure M (⋃n∈{0..k}. cball (from-nat-into X n) r)) ⟶ M
?U
     by (rule Lim-emeasure-incseq) (auto intro!: borel-closed bexI simp: incseq-def
Us sb)
   also have ?U = space M
   proof safe
     fix x from X(2)[OF open-ball[of x r]] ‹r > 0› obtain d where d: d∈X d ∈
ball x r by auto
     show x ∈ ?U
       using X(1) d
       by simp (auto intro!: exI [where x = to-nat-on X d] simp: dist-commute
Bex-def)
   qed (simp add: sU)
    finally have (λk. M (⋃n∈{0..k}. cball (from-nat-into X n) r)) ⟶ M
(space M) .
  } note M-space = this
  {
   fix e ::real and n :: nat assume e > 0 n > 0
   hence 1/n > 0 e * 2 powr − n > 0 by (auto)
   from M-space[OF ‹1/n>0›]
  have (λk. measure M (⋃i∈{0..k}. cball (from-nat-into X i) (1/real n))) ⟶
measure M (space M)
     unfolding emeasure-eq-measure by (auto simp: measure-nonneg)
   from metric-LIMSEQ-D[OF this ‹0 < e * 2 powr −n›]
```

obtain k **where** *dist* (*measure M* ($\bigcup i{\in}\{0..k\}$. *cball* (*from-nat-into X i*) (*1/real n*))) (*measure M* (*space M*)) <
*e * 2 powr −n*
by *auto*
hence *measure M* ($\bigcup i{\in}\{0..k\}$. *cball* (*from-nat-into X i*) (*1/real n*)) ≥
measure M (*space M*) − *e * 2 powr −real n*
by (*auto simp*: *dist-real-def*)
hence ∃ *k*. *measure M* ($\bigcup i{\in}\{0..k\}$. *cball* (*from-nat-into X i*) (*1/real n*)) ≥
measure M (*space M*) − *e * 2 powr − real n* **..**
} note *k=this*
hence ∀ *e*∈{*0<..*}. ∀ (*n*::*nat*)∈{*0<..*}. ∃ *k*.
measure M ($\bigcup i{\in}\{0..k\}$. *cball* (*from-nat-into X i*) (*1/real n*)) ≥ *measure M* (*space M*) − *e * 2 powr − real n*
by *blast*
then obtain k **where** *k*: ∀ *e*∈{*0<..*}. ∀ *n*∈{*0<..*}. *measure M* (*space M*) − *e * 2 powr − real* (*n*::*nat*)
≤ *measure M* ($\bigcup i{\in}\{0..k\ e\ n\}$. *cball* (*from-nat-into X i*) (*1 / n*))
by *metis*
hence *k*: ⋀*e n*. *e > 0* ⟹ *n > 0* ⟹ *measure M* (*space M*) − *e * 2 powr − n*
≤ *measure M* ($\bigcup i{\in}\{0..k\ e\ n\}$. *cball* (*from-nat-into X i*) (*1 / n*))
unfolding *Ball-def* **by** *blast*
have *approx-space*:
∃ *K* ∈ {*K*. *K* ⊆ *space M* ∧ *compact K*}. *emeasure M* (*space M*) ≤ *emeasure M K* + *ennreal e*
(**is** *?thesis e*) **if** *0 < e* **for** *e* :: *real*
proof −
def *B* ≡ λ*n*. $\bigcup i{\in}\{0..k\ e\ (Suc\ n)\}$. *cball* (*from-nat-into X i*) (*1 / Suc n*)
have ⋀*n*. *closed* (*B n*) **by** (*auto simp*: *B-def*)
hence [*simp*]: ⋀*n*. *B n* ∈ *sets M* **by** (*simp add*: *sb*)
from *k*[*OF* ‹*e > 0*› *zero-less-Suc*]
have ⋀*n*. *measure M* (*space M*) − *measure M* (*B n*) ≤ *e * 2 powr − real* (*Suc n*)
by (*simp add*: *algebra-simps B-def finite-measure-compl*)
hence *B-compl-le*: ⋀*n*::*nat*. *measure M* (*space M* − *B n*) ≤ *e * 2 powr − real* (*Suc n*)
by (*simp add*: *finite-measure-compl*)
def *K* ≡ ⋂ *n*. *B n*
from ‹*closed* (*B -*)› **have** *closed K* **by** (*auto simp*: *K-def*)
hence [*simp*]: *K* ∈ *sets M* **by** (*simp add*: *sb*)
have *measure M* (*space M*) − *measure M K* = *measure M* (*space M* − *K*)
by (*simp add*: *finite-measure-compl*)
also have ... = *emeasure M* ($\bigcup n$. *space M* − *B n*) **by** (*auto simp*: *K-def emeasure-eq-measure*)
also have ... ≤ ($\sum n$. *emeasure M* (*space M* − *B n*))
by (*rule emeasure-subadditive-countably*) (*auto simp*: *summable-def*)
also have ... ≤ ($\sum n$. *ennreal* (*e*∗*2 powr − real* (*Suc n*)))
using *B-compl-le* **by** (*intro suminf-le*) (*simp-all add*: *measure-nonneg emeasure-eq-measure ennreal-leI*)
also have ... ≤ ($\sum n$. *ennreal* (*e * (1 / 2)* ^ *Suc n*))

by (*simp add*: *powr-minus powr-realpow field-simps del*: *of-nat-Suc*)
also have ... = *ennreal e* * (∑ *n*. *ennreal* ((*1* / *2*) ^ *Suc n*))
unfolding *ennreal-power*[*symmetric*]
using ‹*0* < *e*›
by (*simp add*: *ac-simps ennreal-mult' divide-ennreal*[*symmetric*] *divide-ennreal-def*
ennreal-power[*symmetric*])
also have ... = *e*
by (*subst suminf-ennreal-eq*[*OF zero-le-power power-half-series*]) *auto*
finally have *measure M* (*space M*) ≤ *measure M K* + *e*
using ‹*0* < *e*› **by** *simp*
hence *emeasure M* (*space M*) ≤ *emeasure M K* + *e*
using ‹*0* < *e*› **by** (*simp add*: *emeasure-eq-measure measure-nonneg ennreal-plus*[*symmetric*]
del: *ennreal-plus*)
moreover have *compact K*
unfolding *compact-eq-totally-bounded*
proof *safe*
show *complete K* **using** ‹*closed K*› **by** (*simp add*: *complete-eq-closed*)
fix *e'*::*real* **assume** *0* < *e'*
from *nat-approx-posE*[*OF this*] **guess** *n* **. note** *n* = *this*
let *?k* = *from-nat-into X* ‘ {*0*..*k e* (*Suc n*)}
have *finite ?k* **by** *simp*
moreover have *K* ⊆ (⋃ *x*∈*?k*. *ball x e'*) **unfolding** *K-def B-def* **using** *n* **by**
force
ultimately show ∃ *k*. *finite k* ∧ *K* ⊆ (⋃ *x*∈*k*. *ball x e'*) **by** *blast*
qed
ultimately
show *?thesis* **by** (*auto simp*: *sU*)
qed
{ fix *A*::*'a set* **assume** *closed A* **hence** *A* ∈ *sets borel* **by** (*simp add*: *compact-imp-closed*)
hence [*simp*]: *A* ∈ *sets M* **by** (*simp add*: *sb*)
have *?inner A*
proof (*rule approx-inner*)
fix *e*::*real* **assume** *e* > *0*
from *approx-space*[*OF this*] **obtain** *K* **where**
K: *K* ⊆ *space M compact K emeasure M* (*space M*) ≤ *emeasure M K* + *e*
by (*auto simp*: *emeasure-eq-measure*)
hence [*simp*]: *K* ∈ *sets M* **by** (*simp add*: *sb compact-imp-closed*)
have *measure M A* − *measure M* (*A* ∩ *K*) = *measure M* (*A* − *A* ∩ *K*)
by (*subst finite-measure-Diff*) *auto*
also have *A* − *A* ∩ *K* = *A* ∪ *K* − *K* **by** *auto*
also have *measure M* ... = *measure M* (*A* ∪ *K*) − *measure M K*
by (*subst finite-measure-Diff*) *auto*
also have ... ≤ *measure M* (*space M*) − *measure M K*
by (*simp add*: *emeasure-eq-measure sU sb finite-measure-mono*)
also have ... ≤ *e*
using *K* ‹*0* < *e*› **by** (*simp add*: *emeasure-eq-measure ennreal-plus*[*symmetric*]
measure-nonneg del: *ennreal-plus*)
finally have *emeasure M A* ≤ *emeasure M* (*A* ∩ *K*) + *ennreal e*
using ‹*0*<*e*› **by** (*simp add*: *emeasure-eq-measure algebra-simps ennreal-plus*[*symmetric*]

measure-nonneg del: *ennreal-plus*)
moreover have $A \cap K \subseteq A$ *compact* $(A \cap K)$ **using** ‹*closed A*› ‹*compact K*›
by *auto*
ultimately show $\exists K \subseteq A$. *compact* $K \wedge$ *emeasure* $M\ A \leq$ *emeasure* $M\ K$ $+$ *ennreal e*
by *blast*
qed *simp*
have *?outer A*
proof *cases*
assume $A \neq \{\}$
let *?G* $= \lambda d.\ \{x.\ infdist\ x\ A < d\}$
{
fix *d*
have *?G d* $= (\lambda x.\ infdist\ x\ A) -` \{..<d\}$ **by** *auto*
also have *open* ...
by (*intro continuous-open-vimage*) (*auto intro*!: *continuous-infdist continuous-ident*)
finally have *open* (*?G d*) **.**
} note *open-G* $=$ *this*
from *in-closed-iff-infdist-zero*[*OF* ‹*closed A*› ‹$A \neq \{\}$›]
have $A = \{x.\ infdist\ x\ A = 0\}$ **by** *auto*
also have ... $= (\bigcap i.\ ?G\ (1/real\ (Suc\ i)))$
proof (*auto simp del*: *of-nat-Suc*, *rule ccontr*)
fix *x*
assume *infdist x A* $\neq 0$
hence *pos*: *infdist x A* > 0 **using** *infdist-nonneg*[*of x A*] **by** *simp*
from *nat-approx-posE*[*OF this*] **guess** *n* **.**
moreover
assume $\forall i.\ infdist\ x\ A < 1\ /\ real\ (Suc\ i)$
hence *infdist x A* $< 1\ /\ real\ (Suc\ n)$ **by** *auto*
ultimately show *False* **by** *simp*
qed
also have *M* ... $=$ (*INF n. emeasure M* (*?G* (*1 / real* (*Suc n*))))
proof (*rule INF-emeasure-decseq*[*symmetric*], *safe*)
fix *i*::*nat*
from *open-G*[*of 1 / real* (*Suc i*)]
show *?G* (*1 / real* (*Suc i*)) $\in$ *sets M* **by** (*simp add*: *sb borel-open*)
next
show *decseq* $(\lambda i.\ \{x.\ infdist\ x\ A < 1\ /\ real\ (Suc\ i)\})$
by (*auto intro*: *less-trans intro*!: *divide-strict-left-mono*
simp: *decseq-def le-eq-less-or-eq*)
qed *simp*
finally
have *emeasure M A* $= (INF\ n.\ emeasure\ M\ \{x.\ infdist\ x\ A < 1\ /\ real\ (Suc\ n)\})$ **.**
moreover
have ... $\geq (INF\ U{:}\{U.\ A \subseteq U \wedge open\ U\}.\ emeasure\ M\ U)$
proof (*intro INF-mono*)
fix *m*
have *?G* (*1 / real* (*Suc m*)) $\in \{U.\ A \subseteq U \wedge open\ U\}$ **using** *open-G* **by**

```
auto
      moreover have M (?G (1 / real (Suc m))) ≤ M (?G (1 / real (Suc m)))
by simp
       ultimately show ∃ U∈{U. A ⊆ U ∧ open U}.
         emeasure M U ≤ emeasure M {x. infdist x A < 1 / real (Suc m)}
         by blast
     qed
     moreover
     have emeasure M A ≤ (INF U:{U. A ⊆ U ∧ open U}. emeasure M U)
       by (rule INF-greatest) (auto intro!: emeasure-mono simp: sb)
     ultimately show ?thesis by simp
   qed (auto intro!: INF-eqI)
   note ‹?inner A› ‹?outer A› }
  note closed-in-D = this
  from ‹B ∈ sets borel›
   have Int-stable (Collect closed) Collect closed ⊆ Pow UNIV B ∈ sigma-sets
UNIV (Collect closed)
    by (auto simp: Int-stable-def borel-eq-closed)
  then show ?inner B ?outer B
  proof (induct B rule: sigma-sets-induct-disjoint)
    case empty
    { case 1 show ?case by (intro SUP-eqI[symmetric]) auto }
    { case 2 show ?case by (intro INF-eqI[symmetric]) (auto elim!: meta-allE[of
- {}]) }
  next
    case (basic B)
    { case 1 from basic closed-in-D show ?case by auto }
    { case 2 from basic closed-in-D show ?case by auto }
  next
    case (compl B)
    note inner = compl(2) and outer = compl(3)
    from compl have [simp]: B ∈ sets M by (auto simp: sb borel-eq-closed)
    case 2
     have M (space M − B) = M (space M) − emeasure M B by (auto simp:
emeasure-compl)
    also have ... = (INF K:{K. K ⊆ B ∧ compact K}. M (space M) −  M K)
      by (subst ennreal-SUP-const-minus) (auto simp: less-top[symmetric] inner)
    also have ... = (INF U:{U. U ⊆ B ∧ compact U}. M (space M − U))
      by (rule INF-cong) (auto simp add: emeasure-compl sb compact-imp-closed)
    also have ... ≥ (INF U:{U. U ⊆ B ∧ closed U}. M (space M − U))
      by (rule INF-superset-mono) (auto simp add: compact-imp-closed)
    also have (INF U:{U. U ⊆ B ∧ closed U}. M (space M − U)) =
       (INF U:{U. space M − B ⊆ U ∧ open U}. emeasure M U)
     unfolding INF-image [of - λu. space M − u -, symmetric, unfolded comp-def]
       by (rule INF-cong) (auto simp add: sU Compl-eq-Diff-UNIV [symmetric,
simp])
    finally have
     (INF U:{U. space M − B ⊆ U ∧ open U}. emeasure M U) ≤ emeasure M
(space M − B) .
```

moreover have
(*INF* U:{U. *space* $M - B \subseteq U \wedge$ *open* U}. *emeasure* M U) $\geq$ *emeasure* M (*space* $M - B$)
by (*auto simp*: *sb sU intro*!: *INF-greatest emeasure-mono*)
ultimately show *?case* **by** (*auto intro*!: *antisym simp*: *sets-eq-imp-space-eq*[*OF sb*])

case *1*
have M (*space* $M - B$) = M (*space* M) − *emeasure* M B **by** (*auto simp*: *emeasure-compl*)
also have ... = (*SUP* U: {U. $B \subseteq U \wedge$ *open* U}. M (*space* M) − M U)
unfolding *outer* **by** (*subst ennreal-INF-const-minus*) *auto*
also have ... = (*SUP* U:{U. $B \subseteq U \wedge$ *open* U}. M (*space* $M - U$))
by (*rule SUP-cong*) (*auto simp add*: *emeasure-compl sb compact-imp-closed*)
also have ... = (*SUP* K:{K. $K \subseteq$ *space* $M - B \wedge$ *closed* K}. *emeasure* M K)
unfolding *SUP-image* [*of* - λu. *space* $M - u$ -, *symmetric*, *unfolded comp-def*]
by (*rule SUP-cong*) (*auto simp add*: *sU*)
also have ... = (*SUP* K:{K. $K \subseteq$ *space* $M - B \wedge$ *compact* K}. *emeasure* M K)
proof (*safe intro*!: *antisym SUP-least*)
fix K **assume** *closed* K $K \subseteq$ *space* $M - B$
from *closed-in-D*[*OF* ‹*closed* K›]
have *K-inner*: *emeasure* M K = (*SUP* K:{Ka. $Ka \subseteq K \wedge$ *compact* Ka}. *emeasure* M K) **by** *simp*
show *emeasure* M $K \leq$ (*SUP* K:{K. $K \subseteq$ *space* $M - B \wedge$ *compact* K}. *emeasure* M K)
unfolding *K-inner* **using** ‹$K \subseteq$ *space* $M - B$›
by (*auto intro*!: *SUP-upper SUP-least*)
qed (*fastforce intro*!: *SUP-least SUP-upper simp*: *compact-imp-closed*)
finally show *?case* **by** (*auto intro*!: *antisym simp*: *sets-eq-imp-space-eq*[*OF sb*])
next
case (*union D*)
then have *range* $D \subseteq$ *sets* M **by** (*auto simp*: *sb borel-eq-closed*)
with *union* **have** M[*symmetric*]: ($\sum i.\ M\ (D\ i)$) = M ($\bigcup i.\ D\ i$) **by** (*intro suminf-emeasure*)
also have ($\lambda n.\ \sum i{<}n.\ M\ (D\ i)$) ⟶ ($\sum i.\ M\ (D\ i)$)
by (*intro summable-LIMSEQ*) *auto*
finally have *measure-LIMSEQ*: ($\lambda n.\ \sum i{<}n.$ *measure* M $(D\ i)$) ⟶ *measure* M ($\bigcup i.\ D\ i$)
by (*simp add*: *emeasure-eq-measure measure-nonneg setsum-nonneg*)
have ($\bigcup i.\ D\ i$) ∈ *sets* M **using** ‹*range* $D \subseteq$ *sets* M› **by** *auto*

case *1*
show *?case*
proof (*rule approx-inner*)
fix e::*real* **assume** $e > 0$
with *measure-LIMSEQ*
have ∃ *no*. ∀ $n{\geq}no$. |($\sum i{<}n.$ *measure* M $(D\ i)$) − *measure* M ($\bigcup x.\ D\ x$)| < $e/2$

by (*auto simp*: *lim-sequentially dist-real-def simp del*: *less-divide-eq-numeral1*)
hence $\exists n0.\ |(\sum i{<}n0.\ measure\ M\ (D\ i)) - measure\ M\ (\bigcup x.\ D\ x)| < e/2$ **by** *auto*
then obtain *n0* **where** *n0*: $|(\sum i{<}n0.\ measure\ M\ (D\ i)) - measure\ M\ (\bigcup i.\ D\ i)| < e/2$
unfolding *choice-iff* **by** *blast*
have $ennreal\ (\sum i{<}n0.\ measure\ M\ (D\ i)) = (\sum i{<}n0.\ M\ (D\ i))$
by (*auto simp add*: *emeasure-eq-measure setsum-nonneg measure-nonneg*)
also have $\ldots \le (\sum i.\ M\ (D\ i))$ **by** (*rule setsum-le-suminf*) *auto*
also have $\ldots = M\ (\bigcup i.\ D\ i)$ **by** (*simp add*: *M*)
also have $\ldots = measure\ M\ (\bigcup i.\ D\ i)$ **by** (*simp add*: *emeasure-eq-measure*)
finally have *n0*: $measure\ M\ (\bigcup i.\ D\ i) - (\sum i{<}n0.\ measure\ M\ (D\ i)) < e/2$
using *n0* **by** (*auto simp*: *measure-nonneg setsum-nonneg*)
have $\forall i.\ \exists K.\ K \subseteq D\ i \wedge compact\ K \wedge emeasure\ M\ (D\ i) \le emeasure\ M\ K + e/(2{*}Suc\ n0)$
proof
fix *i*
from ‹$0 < e$› **have** $0 < e/(2{*}Suc\ n0)$ **by** *simp*
have $emeasure\ M\ (D\ i) = (SUP\ K{:}\{K.\ K \subseteq (D\ i) \wedge compact\ K\}.\ emeasure\ M\ K)$
using *union* **by** *blast*
from *SUP-approx-ennreal*[*OF* ‹$0 < e/(2{*}Suc\ n0)$› *- this*]
show $\exists K.\ K \subseteq D\ i \wedge compact\ K \wedge emeasure\ M\ (D\ i) \le emeasure\ M\ K + e/(2{*}Suc\ n0)$
by (*auto simp*: *emeasure-eq-measure intro*: *less-imp-le compact-empty*)
qed
then obtain *K* **where** *K*: $\bigwedge i.\ K\ i \subseteq D\ i\ \bigwedge i.\ compact\ (K\ i)$
$\bigwedge i.\ emeasure\ M\ (D\ i) \le emeasure\ M\ (K\ i) + e/(2{*}Suc\ n0)$
unfolding *choice-iff* **by** *blast*
let *?K* $= \bigcup i{\in}\{..{<}n0\}.\ K\ i$
have *disjoint-family-on K* $\{..{<}n0\}$ **using** *K* ‹*disjoint-family D*›
unfolding *disjoint-family-on-def* **by** *blast*
hence *mK*: $measure\ M\ ?K = (\sum i{<}n0.\ measure\ M\ (K\ i))$ **using** *K*
by (*intro finite-measure-finite-Union*) (*auto simp*: *sb compact-imp-closed*)
have $measure\ M\ (\bigcup i.\ D\ i) < (\sum i{<}n0.\ measure\ M\ (D\ i)) + e/2$ **using** *n0* **by** *simp*
also have $(\sum i{<}n0.\ measure\ M\ (D\ i)) \le (\sum i{<}n0.\ measure\ M\ (K\ i) + e/(2{*}Suc\ n0))$
using *K* ‹$0 < e$›
by (*auto intro*: *setsum-mono simp*: *emeasure-eq-measure measure-nonneg ennreal-plus*[*symmetric*] *simp del*: *ennreal-plus*)
also have $\ldots = (\sum i{<}n0.\ measure\ M\ (K\ i)) + (\sum i{<}n0.\ e/(2{*}Suc\ n0))$
by (*simp add*: *setsum.distrib*)
also have $\ldots \le (\sum i{<}n0.\ measure\ M\ (K\ i)) + e\ /\ 2$ **using** ‹$0 < e$›
by (*auto simp*: *field-simps intro*!: *mult-left-mono*)
finally
have $measure\ M\ (\bigcup i.\ D\ i) < (\sum i{<}n0.\ measure\ M\ (K\ i)) + e\ /\ 2 + e\ /\ 2$
by *auto*
hence $M\ (\bigcup i.\ D\ i) < M\ ?K + e$

```
        using ‹0<e› by (auto simp: mK emeasure-eq-measure measure-nonneg
setsum-nonneg ennreal-less-iff ennreal-plus[symmetric] simp del: ennreal-plus)
      moreover
      have ?K ⊆ (⋃ i. D i) using K by auto
      moreover
      have compact ?K using K by auto
      ultimately
      have ?K⊆(⋃ i. D i) ∧ compact ?K ∧ emeasure M (⋃ i. D i) ≤ emeasure M
?K + ennreal e by simp
      thus ∃ K⊆⋃ i. D i. compact K ∧ emeasure M (⋃ i. D i) ≤ emeasure M K
+ ennreal e ..
    qed fact
    case 2
    show ?case
    proof (rule approx-outer[OF ‹(⋃ i. D i) ∈ sets M›])
      fix e::real assume e > 0
      have ∀ i::nat. ∃ U. D i ⊆ U ∧ open U ∧ e/(2 powr Suc i) > emeasure M U
− emeasure M (D i)
      proof
        fix i::nat
        from ‹0 < e› have 0 < e/(2 powr Suc i) by simp
        have emeasure M (D i) = (INF U:{U. (D i) ⊆ U ∧ open U}. emeasure M
U)
          using union by blast
        from INF-approx-ennreal[OF ‹0 < e/(2 powr Suc i)› this]
         show ∃ U. D i ⊆ U ∧ open U ∧ e/(2 powr Suc i) > emeasure M U −
emeasure M (D i)
          using ‹0<e›
            by (auto simp: emeasure-eq-measure measure-nonneg setsum-nonneg
ennreal-less-iff ennreal-plus[symmetric] ennreal-minus
                      finite-measure-mono sb
                simp del: ennreal-plus)
      qed
      then obtain U where U: ⋀i. D i ⊆ U i ⋀i. open (U i)
        ⋀i. e/(2 powr Suc i) > emeasure M (U i) − emeasure M (D i)
        unfolding choice-iff by blast
      let ?U = ⋃ i. U i
      have ennreal (measure M ?U − measure M (⋃ i. D i)) = M ?U − M (⋃ i.
D i)
        using U(1,2)
        by (subst ennreal-minus[symmetric])
        (auto intro!: finite-measure-mono simp: sb measure-nonneg emeasure-eq-measure)
      also have ... = M (?U − (⋃ i. D i)) using U ‹(⋃ i. D i) ∈ sets M›
        by (subst emeasure-Diff) (auto simp: sb)
      also have ... ≤ M (⋃ i. U i − D i) using U ‹range D ⊆ sets M›
          by (intro emeasure-mono) (auto simp: sb intro!: sets.countable-nat-UN
sets.Diff)
      also have ... ≤ (∑ i. M (U i − D i)) using U ‹range D ⊆ sets M›
        by (intro emeasure-subadditive-countably) (auto intro!: sets.Diff simp: sb)
```

also have ... $\leq (\sum i.\ ennreal\ e/(2\ powr\ Suc\ i))$ **using** U ‹*range D ⊆ sets M*›
using ‹*0<e*›
by (*intro suminf-le*, *subst emeasure-Diff*)
(*auto simp*: *emeasure-Diff emeasure-eq-measure sb measure-nonneg ennreal-minus*
finite-measure-mono divide-ennreal ennreal-less-iff
intro: *less-imp-le*)
also have ... $\leq (\sum n.\ ennreal\ (e * (1\ /\ 2)\ \hat{}\ Suc\ n))$
using ‹*0<e*›
by (*simp add*: *powr-minus powr-realpow field-simps divide-ennreal del*: *of-nat-Suc*)
also have ... $=$ *ennreal e* $* (\sum n.\ ennreal\ ((1\ /\ 2)\ \hat{}\ Suc\ n))$
unfolding *ennreal-power*[*symmetric*]
using ‹*0 < e*›
by (*simp add*: *ac-simps ennreal-mult' divide-ennreal*[*symmetric*] *divide-ennreal-def*
ennreal-power[*symmetric*])
also have ... $=$ *ennreal e*
by (*subst suminf-ennreal-eq*[*OF zero-le-power power-half-series*]) *auto*
finally have *emeasure M ?U* $\leq$ *emeasure M* $(\bigcup i.\ D\ i)$ $+$ *ennreal e*
using ‹*0<e*› **by** (*simp add*: *emeasure-eq-measure ennreal-plus*[*symmetric*] *measure-nonneg del*: *ennreal-plus*)
moreover
have $(\bigcup i.\ D\ i) \subseteq$ *?U* **using** U **by** *auto*
moreover
have *open ?U* **using** U **by** *auto*
ultimately
have $(\bigcup i.\ D\ i) \subseteq$ *?U* $\wedge$ *open ?U* $\wedge$ *emeasure M ?U* $\leq$ *emeasure M* $(\bigcup i.\ D\ i)$ $+$ *ennreal e* **by** *simp*
thus $\exists B.\ (\bigcup i.\ D\ i) \subseteq B \wedge$ *open B* $\wedge$ *emeasure M B* $\leq$ *emeasure M* $(\bigcup i.\ D\ i)$ $+$ *ennreal e* **..**
qed
qed
qed

end

theory *Set-Integral*
imports *Bochner-Integration Lebesgue-Measure*
begin

abbreviation *set-borel-measurable M A f* $\equiv (\lambda x.\ indicator\ A\ x *_R f\ x) \in$ *borel-measurable M*

abbreviation *set-integrable M A f* $\equiv$ *integrable M* $(\lambda x.\ indicator\ A\ x *_R f\ x)$

abbreviation *set-lebesgue-integral M A f ≡ lebesgue-integral M (λx. indicator A x *$_R$ *f x)*

syntax
-ascii-set-lebesgue-integral :: pttrn ⇒ 'a set ⇒ 'a measure ⇒ real ⇒ real
((4LINT (-):(-)/|(-)./ -) [0,60,110,61] 60)

translations
LINT x:A|M. f == CONST set-lebesgue-integral M A (λx. f)

abbreviation
set-almost-everywhere A M P ≡ AE x in M. x ∈ A ⟶ P x

syntax
-set-almost-everywhere :: pttrn ⇒ 'a set ⇒ 'a ⇒ bool ⇒ bool
(AE -∈- in -./ - [0,0,0,10] 10)

translations
AE x∈A in M. P == CONST set-almost-everywhere A M (λx. P)

syntax
-lebesgue-borel-integral :: pttrn ⇒ real ⇒ real
((2LBINT -./ -) [0,60] 60)

translations
LBINT x. f == CONST lebesgue-integral CONST lborel (λx. f)

syntax
-set-lebesgue-borel-integral :: pttrn ⇒ real set ⇒ real ⇒ real
((3LBINT -:-./ -) [0,60,61] 60)

translations
LBINT x:A. f == CONST set-lebesgue-integral CONST lborel A (λx. f)

lemma *set-borel-measurable-sets*:
fixes *f :: - ⇒ -::real-normed-vector*
assumes *set-borel-measurable M X f B ∈ sets borel X ∈ sets M*
shows *f −‘ B ∩ X ∈ sets M*
proof −
have *f ∈ borel-measurable (restrict-space M X)*
using *assms* **by** *(subst borel-measurable-restrict-space-iff) auto*
then have *f −‘ B ∩ space (restrict-space M X) ∈ sets (restrict-space M X)*
by *(rule measurable-sets) fact*

with ‹$X \in$ *sets* M› **show** *?thesis*
by (*subst* (*asm*) *sets-restrict-space-iff*) (*auto simp*: *space-restrict-space*)
qed

lemma *set-lebesgue-integral-cong*:
assumes $A \in$ *sets* M **and** $\forall x.\ x \in A \longrightarrow f\ x = g\ x$
shows ($LINT\ x{:}A|M.\ f\ x$) = ($LINT\ x{:}A|M.\ g\ x$)
using *assms* **by** (*auto intro*!: *integral-cong split*: *split-indicator simp add*: *sets.sets-into-space*)

lemma *set-lebesgue-integral-cong-AE*:
assumes [*measurable*]: $A \in$ *sets* M $f \in$ *borel-measurable* M $g \in$ *borel-measurable* M
assumes $AE\ x \in A$ *in* $M.\ f\ x = g\ x$
shows $LINT\ x{:}A|M.\ f\ x = LINT\ x{:}A|M.\ g\ x$
proof−
have $AE\ x$ *in* $M.$ *indicator* $A\ x *_R f\ x$ = *indicator* $A\ x *_R g\ x$
using *assms* **by** *auto*
thus *?thesis* **by** (*intro integral-cong-AE*) *auto*
qed

lemma *set-integrable-cong-AE*:
$f \in$ *borel-measurable* $M \Longrightarrow g \in$ *borel-measurable* $M \Longrightarrow$
$AE\ x \in A$ *in* $M.\ f\ x = g\ x \Longrightarrow A \in$ *sets* $M \Longrightarrow$
set-integrable $M\ A\ f$ = *set-integrable* $M\ A\ g$
by (*rule integrable-cong-AE*) *auto*

lemma *set-integrable-subset*:
fixes $M\ A\ B$ **and** f :: - ⇒ - :: {*banach*, *second-countable-topology*}
assumes *set-integrable* $M\ A\ f\ B \in$ *sets* $M\ B \subseteq A$
shows *set-integrable* $M\ B\ f$
proof −
have *set-integrable* $M\ B$ ($\lambda x.$ *indicator* $A\ x *_R f\ x$)
by (*rule integrable-mult-indicator*) *fact*+
with ‹$B \subseteq A$› **show** *?thesis*
by (*simp add*: *indicator-inter-arith*[*symmetric*] *Int-absorb2*)
qed

lemma *set-integral-scaleR-right* [*simp*]: $LINT\ t{:}A|M.\ a *_R f\ t = a *_R$ ($LINT\ t{:}A|M.\ f\ t$)
by (*subst integral-scaleR-right*[*symmetric*]) (*auto intro*!: *integral-cong*)

lemma *set-integral-mult-right* [*simp*]:
fixes a :: $'a$::{*real-normed-field*, *second-countable-topology*}
shows $LINT\ t{:}A|M.\ a * f\ t = a *$ ($LINT\ t{:}A|M.\ f\ t$)
by (*subst integral-mult-right-zero*[*symmetric*]) (*auto intro*!: *integral-cong*)

lemma *set-integral-mult-left* [*simp*]:
fixes a :: $'a$::{*real-normed-field*, *second-countable-topology*}
shows *LINT* t:$A|M$. $f\ t * a$ = (*LINT* t:$A|M$. $f\ t$) $*\ a$
by (*subst integral-mult-left-zero*[*symmetric*]) (*auto intro*!: *integral-cong*)

lemma *set-integral-divide-zero* [*simp*]:
fixes a :: $'a$::{*real-normed-field*, *field*, *second-countable-topology*}
shows *LINT* t:$A|M$. $f\ t\ /\ a$ = (*LINT* t:$A|M$. $f\ t$) $/\ a$
by (*subst integral-divide-zero*[*symmetric*], *intro integral-cong*)
(*auto split*: *split-indicator*)

lemma *set-integrable-scaleR-right* [*simp*, *intro*]:
shows ($a \neq 0 \Longrightarrow$ *set-integrable* $M\ A\ f$) $\Longrightarrow$ *set-integrable* $M\ A$ ($\lambda t.\ a *_R f\ t$)
unfolding *scaleR-left-commute* **by** (*rule integrable-scaleR-right*)

lemma *set-integrable-scaleR-left* [*simp*, *intro*]:
fixes a :: - :: {*banach*, *second-countable-topology*}
shows ($a \neq 0 \Longrightarrow$ *set-integrable* $M\ A\ f$) $\Longrightarrow$ *set-integrable* $M\ A$ ($\lambda t.\ f\ t *_R a$)
using *integrable-scaleR-left*[*of a M* $\lambda x.$ *indicator* $A\ x *_R f\ x$] **by** *simp*

lemma *set-integrable-mult-right* [*simp*, *intro*]:
fixes a :: $'a$::{*real-normed-field*, *second-countable-topology*}
shows ($a \neq 0 \Longrightarrow$ *set-integrable* $M\ A\ f$) $\Longrightarrow$ *set-integrable* $M\ A$ ($\lambda t.\ a * f\ t$)
using *integrable-mult-right*[*of a M* $\lambda x.$ *indicator* $A\ x *_R f\ x$] **by** *simp*

lemma *set-integrable-mult-left* [*simp*, *intro*]:
fixes a :: $'a$::{*real-normed-field*, *second-countable-topology*}
shows ($a \neq 0 \Longrightarrow$ *set-integrable* $M\ A\ f$) $\Longrightarrow$ *set-integrable* $M\ A$ ($\lambda t.\ f\ t * a$)
using *integrable-mult-left*[*of a M* $\lambda x.$ *indicator* $A\ x *_R f\ x$] **by** *simp*

lemma *set-integrable-divide* [*simp*, *intro*]:
fixes a :: $'a$::{*real-normed-field*, *field*, *second-countable-topology*}
assumes $a \neq 0 \Longrightarrow$ *set-integrable* $M\ A\ f$
shows *set-integrable* $M\ A$ ($\lambda t.\ f\ t\ /\ a$)
proof −
have *integrable* M ($\lambda x.$ *indicator* $A\ x *_R f\ x\ /\ a$)
using *assms* **by** (*rule integrable-divide-zero*)
also have ($\lambda x.$ *indicator* $A\ x *_R f\ x\ /\ a$) = ($\lambda x.$ *indicator* $A\ x *_R (f\ x\ /\ a)$)
by (*auto split*: *split-indicator*)
finally show *?thesis* **.**
qed

lemma *set-integral-add* [*simp*, *intro*]:
fixes $f\ g$ :: - $\Rightarrow$ - :: {*banach*, *second-countable-topology*}
assumes *set-integrable* $M\ A\ f$ *set-integrable* $M\ A\ g$
shows *set-integrable* $M\ A$ ($\lambda x.\ f\ x + g\ x$)
and *LINT* x:$A|M$. $f\ x + g\ x$ = (*LINT* x:$A|M$. $f\ x$) + (*LINT* x:$A|M$. $g\ x$)
using *assms* **by** (*simp-all add*: *scaleR-add-right*)

lemma *set-integral-diff* [*simp*, *intro*]:
assumes *set-integrable M A f set-integrable M A g*
shows *set-integrable M A* ($\lambda x.\ f\ x - g\ x$) **and** *LINT x:A|M. f x − g x* =
(*LINT x:A|M. f x*) − (*LINT x:A|M. g x*)
using *assms* **by** (*simp-all add*: *scaleR-diff-right*)

lemma *set-integral-reflect*:
fixes *S* **and** *f* :: *real* ⇒ *'a* :: {*banach*, *second-countable-topology*}
shows (*LBINT x* : *S. f x*) = (*LBINT x* : {*x. − x* ∈ *S*}. *f* (− *x*))
using *assms*
by (*subst lborel-integral-real-affine*[**where** *c=−1* **and** *t=0*])
(*auto intro*!: *integral-cong split*: *split-indicator*)

lemma *set-integral-uminus*: *set-integrable M A f* ⟹ *LINT x:A|M. − f x* = −
(*LINT x:A|M. f x*)
by (*subst integral-minus*[*symmetric*]) *simp-all*

lemma *set-integral-complex-of-real*:
LINT x:A|M. complex-of-real (*f x*) = *of-real* (*LINT x:A|M. f x*)
by (*subst integral-complex-of-real*[*symmetric*])
(*auto intro*!: *integral-cong split*: *split-indicator*)

lemma *set-integral-mono*:
fixes *f g* :: - ⇒ *real*
assumes *set-integrable M A f set-integrable M A g*
⋀*x. x* ∈ *A* ⟹ *f x* ≤ *g x*
shows (*LINT x:A|M. f x*) ≤ (*LINT x:A|M. g x*)
using *assms* **by** (*auto intro*: *integral-mono split*: *split-indicator*)

lemma *set-integral-mono-AE*:
fixes *f g* :: - ⇒ *real*
assumes *set-integrable M A f set-integrable M A g*
AE x ∈ *A in M. f x* ≤ *g x*
shows (*LINT x:A|M. f x*) ≤ (*LINT x:A|M. g x*)
using *assms* **by** (*auto intro*: *integral-mono-AE split*: *split-indicator*)

lemma *set-integrable-abs*: *set-integrable M A f* ⟹ *set-integrable M A* ($\lambda x.\ |f\ x|$ :: *real*)
using *integrable-abs*[*of M* $\lambda x.\ f\ x * indicator\ A\ x$] **by** (*simp add*: *abs-mult ac-simps*)

lemma *set-integrable-abs-iff*:
fixes *f* :: - ⇒ *real*
shows *set-borel-measurable M A f* ⟹ *set-integrable M A* ($\lambda x.\ |f\ x|$) = *set-integrable M A f*
by (*subst* (*2*) *integrable-abs-iff*[*symmetric*]) (*simp-all add*: *abs-mult ac-simps*)

lemma *set-integrable-abs-iff'*:

fixes $f :: - \Rightarrow real$
shows $f \in borel\text{-}measurable\ M \Longrightarrow A \in sets\ M \Longrightarrow$
$set\text{-}integrable\ M\ A\ (\lambda x.\ |f\ x|) = set\text{-}integrable\ M\ A\ f$
by (*intro set-integrable-abs-iff*) *auto*

lemma *set-integrable-discrete-difference*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
assumes *countable X*
assumes *diff*: $(A - B) \cup (B - A) \subseteq X$
assumes $\bigwedge x.\ x \in X \Longrightarrow emeasure\ M\ \{x\} = 0\ \bigwedge x.\ x \in X \Longrightarrow \{x\} \in sets\ M$
shows $set\text{-}integrable\ M\ A\ f \longleftrightarrow set\text{-}integrable\ M\ B\ f$
proof (*rule integrable-discrete-difference*[**where** $X{=}X$])
show $\bigwedge x.\ x \in space\ M \Longrightarrow x \notin X \Longrightarrow indicator\ A\ x *_R f\ x = indicator\ B\ x *_R f\ x$
using *diff* **by** (*auto split*: *split-indicator*)
qed *fact+*

lemma *set-integral-discrete-difference*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}$
assumes *countable X*
assumes *diff*: $(A - B) \cup (B - A) \subseteq X$
assumes $\bigwedge x.\ x \in X \Longrightarrow emeasure\ M\ \{x\} = 0\ \bigwedge x.\ x \in X \Longrightarrow \{x\} \in sets\ M$
shows $set\text{-}lebesgue\text{-}integral\ M\ A\ f = set\text{-}lebesgue\text{-}integral\ M\ B\ f$
proof (*rule integral-discrete-difference*[**where** $X{=}X$])
show $\bigwedge x.\ x \in space\ M \Longrightarrow x \notin X \Longrightarrow indicator\ A\ x *_R f\ x = indicator\ B\ x *_R f\ x$
using *diff* **by** (*auto split*: *split-indicator*)
qed *fact+*

lemma *set-integrable-Un*:
fixes $f\ g :: - \Rightarrow - :: \{banach, second\text{-}countable\text{-}topology\}$
assumes *f-A*: *set-integrable M A f* **and** *f-B*: *set-integrable M B f*
and [*measurable*]: $A \in sets\ M\ B \in sets\ M$
shows *set-integrable M* $(A \cup B)$ *f*
proof −
have *set-integrable M* $(A - B)$ *f*
using *f-A* **by** (*rule set-integrable-subset*) *auto*
from *integrable-add*[*OF this f-B*] **show** *?thesis*
by (*rule integrable-cong*[*THEN iffD1*, *rotated 2*]) (*auto split*: *split-indicator*)
qed

lemma *set-integrable-UN*:
fixes $f :: - \Rightarrow - :: \{banach, second\text{-}countable\text{-}topology\}$
assumes *finite I* $\bigwedge i.\ i{\in}I \Longrightarrow set\text{-}integrable\ M\ (A\ i)\ f$
$\bigwedge i.\ i{\in}I \Longrightarrow A\ i \in sets\ M$
shows *set-integrable M* $(\bigcup i{\in}I.\ A\ i)$ *f*
using *assms* **by** (*induct I*) (*auto intro*!: *set-integrable-Un*)

lemma *set-integral-Un*:

fixes $f :: - \Rightarrow - :: \{banach, second\text{-}countable\text{-}topology\}$
assumes $A \cap B = \{\}$
and *set-integrable M A f*
and *set-integrable M B f*
shows $LINT\ x{:}A \cup B|M.\ f\ x = (LINT\ x{:}A|M.\ f\ x) + (LINT\ x{:}B|M.\ f\ x)$
by (*auto simp add*: *indicator-union-arith indicator-inter-arith*[*symmetric*] *scaleR-add-left assms*)

lemma *set-integral-cong-set*:
fixes $f :: - \Rightarrow - :: \{banach, second\text{-}countable\text{-}topology\}$
assumes [*measurable*]: *set-borel-measurable M A f set-borel-measurable M B f*
and *ae*: *AE x in M*. $x \in A \longleftrightarrow x \in B$
shows $LINT\ x{:}B|M.\ f\ x = LINT\ x{:}A|M.\ f\ x$
proof (*rule integral-cong-AE*)
show *AE x in M*. *indicator B x* $*_R$ *f x* = *indicator A x* $*_R$ *f x*
using *ae* **by** (*auto simp*: *subset-eq split*: *split-indicator*)
qed *fact+*

lemma *set-borel-measurable-subset*:
fixes $f :: - \Rightarrow - :: \{banach, second\text{-}countable\text{-}topology\}$
assumes [*measurable*]: *set-borel-measurable M A f B* $\in$ *sets M* **and** $B \subseteq A$
shows *set-borel-measurable M B f*
proof −
have *set-borel-measurable M B* (λx. *indicator A x* $*_R$ *f x*)
by *measurable*
also have (λx. *indicator B x* $*_R$ *indicator A x* $*_R$ *f x*) = (λx. *indicator B x* $*_R$ *f x*)
using ‹$B \subseteq A$› **by** (*auto simp*: *fun-eq-iff split*: *split-indicator*)
finally show *?thesis* **.**
qed

lemma *set-integral-Un-AE*:
fixes $f :: - \Rightarrow - :: \{banach, second\text{-}countable\text{-}topology\}$
assumes *ae*: *AE x in M*. $\neg\ (x \in A \wedge x \in B)$ **and** [*measurable*]: *A* $\in$ *sets M B* $\in$ *sets M*
and *set-integrable M A f*
and *set-integrable M B f*
shows $LINT\ x{:}A \cup B|M.\ f\ x = (LINT\ x{:}A|M.\ f\ x) + (LINT\ x{:}B|M.\ f\ x)$
proof −
have *f*: *set-integrable M* ($A \cup B$) *f*
by (*intro set-integrable-Un assms*)
then have *f'*: *set-borel-measurable M* ($A \cup B$) *f*
by (*rule borel-measurable-integrable*)
have $LINT\ x{:}A \cup B|M.\ f\ x = LINT\ x{:}(A - A \cap B) \cup (B - A \cap B)|M.\ f\ x$
proof (*rule set-integral-cong-set*)
show *AE x in M*. $(x \in A - A \cap B \cup (B - A \cap B)) = (x \in A \cup B)$
using *ae* **by** *auto*
show *set-borel-measurable M* ($A - A \cap B \cup (B - A \cap B)$) *f*
using *f'* **by** (*rule set-borel-measurable-subset*) *auto*

qed *fact*
also have ... = (*LINT* x:(A − $A \cap B$)|M. $f\ x$) + (*LINT* x:(B − $A \cap B$)|M. f x)
by (*auto intro*!: *set-integral-Un set-integrable-subset*[*OF f*])
also have ... = (*LINT* x:A|M. $f\ x$) + (*LINT* x:B|M. $f\ x$)
using *ae*
by (*intro arg-cong2*[**where** f=$op+$] *set-integral-cong-set*)
(*auto intro*!: *set-borel-measurable-subset*[*OF f'*])
finally show *?thesis* **.**
qed

lemma *set-integral-finite-Union*:
fixes f :: - ⇒ - :: {*banach*, *second-countable-topology*}
assumes *finite I disjoint-family-on A I*
and ⋀i. $i \in I \Longrightarrow$ *set-integrable M* ($A\ i$) f ⋀i. $i \in I \Longrightarrow A\ i \in$ *sets M*
shows (*LINT* x:(⋃i∈I. $A\ i$)|M. $f\ x$) = (∑i∈I. *LINT* x:$A\ i$|M. $f\ x$)
using *assms*
apply *induct*
apply (*auto intro*!: *set-integral-Un set-integrable-Un set-integrable-UN simp*: *disjoint-family-on-def*)
by (*subst set-integral-Un*, *auto intro*: *set-integrable-UN*)

lemma *pos-integrable-to-top*:
fixes l::*real*
assumes ⋀i. $A\ i \in$ *sets M mono A*
assumes *nneg*: ⋀$x\ i$. $x \in A\ i \Longrightarrow 0 \leq f\ x$
and *intgbl*: ⋀i::*nat*. *set-integrable M* ($A\ i$) f
and *lim*: (λi::*nat*. *LINT* x:$A\ i$|M. $f\ x$) ⟶ l
shows *set-integrable M* (⋃i. $A\ i$) f
apply (*rule integrable-monotone-convergence*[**where** f = λi::*nat*. λx. *indicator* ($A\ i$) x ∗$_R$ $f\ x$ **and** x = l])
apply (*rule intgbl*)
prefer *3* **apply** (*rule lim*)
apply (*rule AE-I2*)
using ‹*mono A*› **apply** (*auto simp*: *mono-def nneg split*: *split-indicator*) []
proof (*rule AE-I2*)
{ fix x **assume** $x \in$ *space M*
show (λi. *indicator* ($A\ i$) x ∗$_R$ $f\ x$) ⟶ *indicator* (⋃i. $A\ i$) x ∗$_R$ $f\ x$
proof *cases*
assume ∃i. $x \in A\ i$
then guess i **..**
then have ∗: *eventually* (λi. $x \in A\ i$) *sequentially*
using ‹$x \in A\ i$› ‹*mono A*› **by** (*auto simp*: *eventually-sequentially mono-def*)
show *?thesis*
apply (*intro Lim-eventually*)
using ∗
apply *eventually-elim*
apply (*auto split*: *split-indicator*)
done

```
    qed auto }
  then show (λx. indicator (⋃i. A i) x *R f x) ∈ borel-measurable M
    apply (rule borel-measurable-LIMSEQ-real)
    apply assumption
    apply (intro borel-measurable-integrable intgbl)
    done
qed


lemma lebesgue-integral-countable-add:
  fixes f :: - ⇒ 'a :: {banach, second-countable-topology}
  assumes meas[intro]: ⋀i::nat. A i ∈ sets M
    and disj: ⋀i j. i ≠ j ⟹ A i ∩ A j = {}
    and intgbl: set-integrable M (⋃i. A i) f
  shows LINT x:(⋃i. A i)|M. f x = (∑i. (LINT x:(A i)|M. f x))
proof (subst integral-suminf[symmetric])
  show int-A: ⋀i. set-integrable M (A i) f
    using intgbl by (rule set-integrable-subset) auto
  { fix x assume x ∈ space M
    have (λi. indicator (A i) x *R f x) sums (indicator (⋃i. A i) x *R f x)
      by (intro sums-scaleR-left indicator-sums) fact }
  note sums = this

  have norm-f: ⋀i. set-integrable M (A i) (λx. norm (f x))
    using int-A[THEN integrable-norm] by auto

  show AE x in M. summable (λi. norm (indicator (A i) x *R f x))
    using disj by (intro AE-I2) (auto intro!: summable-mult2 sums-summable[OF
indicator-sums])

  show summable (λi. LINT x|M. norm (indicator (A i) x *R f x))
  proof (rule summableI-nonneg-bounded)
    fix n
    show 0 ≤ LINT x|M. norm (indicator (A n) x *R f x)
      using norm-f by (auto intro!: integral-nonneg-AE)

    have (∑i<n. LINT x|M. norm (indicator (A i) x *R f x)) =
      (∑i<n. set-lebesgue-integral M (A i) (λx. norm (f x)))
      by (simp add: abs-mult)
    also have ... = set-lebesgue-integral M (⋃i<n. A i) (λx. norm (f x))
      using norm-f
      by (subst set-integral-finite-Union) (auto simp: disjoint-family-on-def disj)
    also have ... ≤ set-lebesgue-integral M (⋃i. A i) (λx. norm (f x))
      using intgbl[THEN integrable-norm]
      by (intro integral-mono set-integrable-UN[of {..<n}] norm-f)
        (auto split: split-indicator)
    finally show (∑i<n. LINT x|M. norm (indicator (A i) x *R f x)) ≤
      set-lebesgue-integral M (⋃i. A i) (λx. norm (f x))
      by simp
```

qed
show *set-lebesgue-integral M (UNION UNIV A) f = LINT x|M. (∑ i. indicator (A i) x *R f x)*
apply (*rule integral-cong*[*OF refl*])
apply (*subst suminf-scaleR-left*[*OF sums-summable*[*OF indicator-sums*, *OF disj*], *symmetric*])
using *sums-unique*[*OF indicator-sums*[*OF disj*]]
apply *auto*
done
qed

lemma *set-integral-cont-up*:
fixes *f* :: - ⇒ 'a :: {*banach*, *second-countable-topology*}
assumes [*measurable*]: ⋀*i*. *A i* ∈ *sets M* **and** *A*: *incseq A*
and *intgbl*: *set-integrable M* (⋃ *i*. *A i*) *f*
shows (λ*i*. *LINT x*:(*A i*)|*M*. *f x*) ⟶ *LINT x*:(⋃ *i*. *A i*)|*M*. *f x*
proof (*intro integral-dominated-convergence*[**where** *w*=λ*x*. *indicator* (⋃ *i*. *A i*) *x* *R norm* (*f x*)])
have *int-A*: ⋀*i*. *set-integrable M* (*A i*) *f*
using *intgbl* **by** (*rule set-integrable-subset*) *auto*
then show ⋀*i*. *set-borel-measurable M* (*A i*) *f set-borel-measurable M* (⋃ *i*. *A i*) *f*
set-integrable M (⋃ *i*. *A i*) (λ*x*. *norm* (*f x*))
using *intgbl integrable-norm*[*OF intgbl*] **by** *auto*

{ fix *x i* **assume** *x* ∈ *A i*
with *A* **have** (λ*xa*. *indicator* (*A xa*) *x*::*real*) ⟶ *1* ⟷ (λ*xa*. *1*::*real*) ⟶ *1*
by (*intro filterlim-cong refl*)
(*fastforce simp*: *eventually-sequentially incseq-def subset-eq intro*!: *exI*[*of* - *i*]) **}**
then show *AE x in M*. (λ*i*. *indicator* (*A i*) *x* *R f x*) ⟶ *indicator* (⋃ *i*. *A i*) *x* *R f x*
by (*intro AE-I2 tendsto-intros*) (*auto split*: *split-indicator*)
qed (*auto split*: *split-indicator*)

lemma *set-integral-cont-down*:
fixes *f* :: - ⇒ 'a :: {*banach*, *second-countable-topology*}
assumes [*measurable*]: ⋀*i*. *A i* ∈ *sets M* **and** *A*: *decseq A*
and *int0*: *set-integrable M* (*A 0*) *f*
shows (λ*i*::*nat*. *LINT x*:(*A i*)|*M*. *f x*) ⟶ *LINT x*:(⋂ *i*. *A i*)|*M*. *f x*
proof (*rule integral-dominated-convergence*)
have *int-A*: ⋀*i*. *set-integrable M* (*A i*) *f*
using *int0* **by** (*rule set-integrable-subset*) (*insert A*, *auto simp*: *decseq-def*)
show *set-integrable M* (*A 0*) (λ*x*. *norm* (*f x*))
using *int0*[*THEN integrable-norm*] **by** *simp*
have *set-integrable M* (⋂ *i*. *A i*) *f*
using *int0* **by** (*rule set-integrable-subset*) (*insert A*, *auto simp*: *decseq-def*)

with *int-A* **show** *set-borel-measurable M* $(\bigcap i.\ A\ i)$ *f* $\bigwedge i.$ *set-borel-measurable M* $(A\ i)$ *f*
by *auto*
show $\bigwedge i.$ *AE x in M. norm* $(indicator\ (A\ i)\ x *_R f\ x) \leq indicator\ (A\ 0)\ x *_R norm\ (f\ x)$
using *A* **by** (*auto split*: *split-indicator simp*: *decseq-def*)
{ fix *x i* **assume** $x \in space\ M\ x \notin A\ i$
with *A* **have** $(\lambda i.\ indicator\ (A\ i)\ x{::}real) \longrightarrow 0 \longleftrightarrow (\lambda i.\ 0{::}real) \longrightarrow 0$
by (*intro filterlim-cong refl*)
(*auto split*: *split-indicator simp*: *eventually-sequentially decseq-def intro*!: *exI*[*of - i*]) **}**
then show *AE x in M.* $(\lambda i.\ indicator\ (A\ i)\ x *_R f\ x) \longrightarrow indicator\ (\bigcap i.\ A\ i)\ x *_R f\ x$
by (*intro AE-I2 tendsto-intros*) (*auto split*: *split-indicator*)
qed

lemma *set-integral-at-point*:
fixes *a* :: *real*
assumes *set-integrable M* $\{a\}$ *f*
and [*simp*]: $\{a\} \in sets\ M$ **and** $(emeasure\ M)\ \{a\} \neq \infty$
shows $(LINT\ x{:}\{a\} \mid M.\ f\ x) = f\ a * measure\ M\ \{a\}$
proof−
have *set-lebesgue-integral M* $\{a\}$ *f* = *set-lebesgue-integral M* $\{a\}$ $(\%x.\ f\ a)$
by (*intro set-lebesgue-integral-cong*) *simp-all*
then show *?thesis* **using** *assms* **by** *simp*
qed

abbreviation *complex-integrable* :: *'a measure* ⇒ (*'a* ⇒ *complex*) ⇒ *bool* **where**
complex-integrable M f ≡ *integrable M f*

abbreviation *complex-lebesgue-integral* :: *'a measure* ⇒ (*'a* ⇒ *complex*) ⇒ *complex* $(integral^C)$ **where**
$integral^C\ M\ f == integral^L\ M\ f$

syntax
-complex-lebesgue-integral :: *pttrn* ⇒ *complex* ⇒ *'a measure* ⇒ *complex*
$(\int^C$ -. - ∂- [*60*,*61*] *110*)

translations
$\int^C x.\ f\ \partial M == CONST\ complex\text{-}lebesgue\text{-}integral\ M\ (\lambda x.\ f)$

syntax
-ascii-complex-lebesgue-integral :: *pttrn* ⇒ *'a measure* ⇒ *real* ⇒ *real*
((*3CLINT* -|-. -) [*0*,*110*,*60*] *60*)

translations
CLINT x|*M. f* == *CONST complex-lebesgue-integral M* $(\lambda x.\ f)$

lemma *complex-integrable-cnj* [*simp*]:
complex-integrable M (λ*x*. *cnj* (*f x*)) ⟷ *complex-integrable M f*
proof
assume *complex-integrable M* (λ*x*. *cnj* (*f x*))
then have *complex-integrable M* (λ*x*. *cnj* (*cnj* (*f x*)))
by (*rule integrable-cnj*)
then show *complex-integrable M f*
by *simp*
qed *simp*

lemma *complex-of-real-integrable-eq*:
complex-integrable M (λ*x*. *complex-of-real* (*f x*)) ⟷ *integrable M f*
proof
assume *complex-integrable M* (λ*x*. *complex-of-real* (*f x*))
then have *integrable M* (λ*x*. *Re* (*complex-of-real* (*f x*)))
by (*rule integrable-Re*)
then show *integrable M f*
by *simp*
qed *simp*

abbreviation *complex-set-integrable* :: *'a measure* ⇒ *'a set* ⇒ (*'a* ⇒ *complex*) ⇒ *bool* **where**
complex-set-integrable M A f ≡ *set-integrable M A f*

abbreviation *complex-set-lebesgue-integral* :: *'a measure* ⇒ *'a set* ⇒ (*'a* ⇒ *complex*) ⇒ *complex* **where**
complex-set-lebesgue-integral M A f ≡ *set-lebesgue-integral M A f*

syntax
-ascii-complex-set-lebesgue-integral :: *pttrn* ⇒ *'a set* ⇒ *'a measure* ⇒ *real* ⇒ *real* ((*4CLINT -:-|-. -*) [*0*,*60*,*110*,*61*] *60*)

translations
CLINT x:*A*|*M*. *f* == *CONST complex-set-lebesgue-integral M A* (λ*x*. *f*)

lemma *borel-integrable-atLeastAtMost'*:
fixes *f* :: *real* ⇒ *'a*::{*banach*, *second-countable-topology*}
assumes *f*: *continuous-on* {*a*..*b*} *f*
shows *set-integrable lborel* {*a*..*b*} *f* (**is** *integrable - ?f*)
by (*intro borel-integrable-compact compact-Icc f*)

lemma *integral-FTC-atLeastAtMost*:
fixes *f* :: *real* ⇒ *'a* :: *euclidean-space*
assumes *a* ≤ *b*
and *F*: ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ (*F has-vector-derivative f x*) (*at x within* {*a* .. *b*})

```
    and f: continuous-on {a .. b} f
  shows integral^L lborel (λx. indicator {a .. b} x *_R f x) = F b - F a
proof -
  let ?f = λx. indicator {a .. b} x *_R f x
  have (?f has-integral (∫ x. ?f x ∂lborel)) UNIV
   using borel-integrable-atLeastAtMost'[OF f] by (rule has-integral-integral-lborel)
  moreover
  have (f has-integral F b - F a) {a .. b}
    by (intro fundamental-theorem-of-calculus ballI assms) auto
  then have (?f has-integral F b - F a) {a .. b}
    by (subst has-integral-cong[where g=f]) auto
  then have (?f has-integral F b - F a) UNIV
    by (intro has-integral-on-superset[where t=UNIV and s={a..b}]) auto
  ultimately show integral^L lborel ?f = F b - F a
    by (rule has-integral-unique)
qed

lemma set-borel-integral-eq-integral:
  fixes f :: real ⇒ 'a::euclidean-space
  assumes set-integrable lborel S f
  shows f integrable-on S LINT x : S | lborel. f x = integral S f
proof -
  let ?f = λx. indicator S x *_R f x
  have (?f has-integral LINT x : S | lborel. f x) UNIV
    by (rule has-integral-integral-lborel) fact
  hence 1: (f has-integral (set-lebesgue-integral lborel S f)) S
    apply (subst has-integral-restrict-univ [symmetric])
    apply (rule has-integral-eq)
    by auto
  thus f integrable-on S
    by (auto simp add: integrable-on-def)
  with 1 have (f has-integral (integral S f)) S
    by (intro integrable-integral, auto simp add: integrable-on-def)
  thus LINT x : S | lborel. f x = integral S f
    by (intro has-integral-unique [OF 1])
qed

lemma set-borel-measurable-continuous:
  fixes f :: - ⇒ -::real-normed-vector
  assumes S ∈ sets borel continuous-on S f
  shows set-borel-measurable borel S f
proof -
  have (λx. if x ∈ S then f x else 0) ∈ borel-measurable borel
    by (intro assms borel-measurable-continuous-on-if continuous-on-const)
  also have (λx. if x ∈ S then f x else 0) = (λx. indicator S x *_R f x)
    by auto
  finally show ?thesis .
qed
```

lemma *set-measurable-continuous-on-ivl*:
assumes *continuous-on* $\{a..b\}$ $(f :: real \Rightarrow real)$
shows *set-borel-measurable borel* $\{a..b\}$ *f*
by (*rule set-borel-measurable-continuous*[*OF - assms*]) *simp*

end

theory *Interval-Integral*
imports *Set-Integral*
begin

lemma *continuous-on-vector-derivative*:
$(\bigwedge x.\ x \in S \Longrightarrow (f \text{ has-vector-derivative } f'\ x)\ (at\ x\ within\ S)) \Longrightarrow$ *continuous-on S f*
by (*auto simp*: *continuous-on-eq-continuous-within intro*!: *has-vector-derivative-continuous*)

lemma *has-vector-derivative-weaken*:
fixes *x D* **and** *f g s t*
assumes *f*: (*f has-vector-derivative D*) (*at x within t*)
and $x \in s\ s \subseteq t$
and $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
shows (*g has-vector-derivative D*) (*at x within s*)
proof −
have (*f has-vector-derivative D*) (*at x within s*) $\longleftrightarrow$ (*g has-vector-derivative D*) (*at x within s*)
unfolding *has-vector-derivative-def has-derivative-iff-norm*
using *assms* **by** (*intro conj-cong Lim-cong-within refl*) *auto*
then show *?thesis*
using *has-vector-derivative-within-subset*[*OF f* ‹$s \subseteq t$›] **by** *simp*
qed

definition *einterval a b* $= \{x.\ a < ereal\ x \wedge ereal\ x < b\}$

lemma *einterval-eq*[*simp*]:
shows *einterval-eq-Icc*: *einterval* (*ereal a*) (*ereal b*) $= \{a <..< b\}$
and *einterval-eq-Ici*: *einterval* (*ereal a*) $\infty = \{a <..\}$
and *einterval-eq-Iic*: *einterval* $(-\infty)$ (*ereal b*) $= \{..< b\}$
and *einterval-eq-UNIV*: *einterval* $(-\infty)\ \infty = UNIV$
by (*auto simp*: *einterval-def*)

lemma *einterval-same*: *einterval a a* $= \{\}$
by (*auto simp add*: *einterval-def*)

lemma *einterval-iff*: $x \in$ *einterval a b* $\longleftrightarrow a <$ *ereal x* $\wedge$ *ereal x* $< b$
by (*simp add*: *einterval-def*)

lemma *einterval-nonempty*: $a < b \Longrightarrow \exists\, c.\ c \in$ *einterval a b*
by (*cases a b rule*: *ereal2-cases*, *auto simp*: *einterval-def intro*!: *dense gt-ex lt-ex*)

lemma *open-einterval*[*simp*]: *open* (*einterval a b*)
by (*cases a b rule*: *ereal2-cases*)
(*auto simp*: *einterval-def intro*!: *open-Collect-conj open-Collect-less continuous-intros*)

lemma *borel-einterval*[*measurable*]: *einterval a b* ∈ *sets borel*
unfolding *einterval-def* **by** *measurable*

lemma *filterlim-sup1*: (*LIM x F*. *f x* :> *G1*) ⟹ (*LIM x F*. *f x* :> (*sup G1 G2*))
unfolding *filterlim-def* **by** (*auto intro*: *le-supI1*)

lemma *ereal-incseq-approx*:
fixes *a b* :: *ereal*
assumes *a* < *b*
obtains *X* :: *nat* ⇒ *real* **where**
incseq X ⋀*i*. *a* < *X i* ⋀*i*. *X i* < *b X* ⟶ *b*
proof (*cases b*)
case *PInf*
with ‹*a* < *b*› **have** *a* = −∞ ∨ (∃ *r*. *a* = *ereal r*)
by (*cases a*) *auto*
moreover have (λ*x*. *ereal* (*real* (*Suc x*))) ⟶ ∞
apply (*subst LIMSEQ-Suc-iff*)
apply (*simp add*: *Lim-PInfty*)
using *nat-ceiling-le-eq* **by** *blast*
moreover have ⋀*r*. (λ*x*. *ereal* (*r* + *real* (*Suc x*))) ⟶ ∞
apply (*subst LIMSEQ-Suc-iff*)
apply (*subst Lim-PInfty*)
apply (*metis add.commute diff-le-eq nat-ceiling-le-eq ereal-less-eq*(*3*))
done
ultimately show *thesis*
by (*intro that*[*of* λ*i*. *real-of-ereal a* + *Suc i*])
(*auto simp*: *incseq-def PInf*)
next
case (*real b*′)
def *d* ≡ *b*′ − (*if a* = −∞ *then b*′ − *1 else real-of-ereal a*)
with ‹*a* < *b*› **have** *a*′: *0* < *d*
by (*cases a*) (*auto simp*: *real*)
moreover
have ⋀*i r*. *r* < *b*′ ⟹ (*b*′ − *r*) ∗ *1* < (*b*′ − *r*) ∗ *real* (*Suc* (*Suc i*))
by (*intro mult-strict-left-mono*) *auto*
with ‹*a* < *b*› *a*′ **have** ⋀*i*. *a* < *ereal* (*b*′ − *d* / *real* (*Suc* (*Suc i*)))
by (*cases a*) (*auto simp*: *real d-def field-simps*)
moreover have (λ*i*. *b*′ − *d* / *Suc* (*Suc i*)) ⟶ *b*′
apply (*subst filterlim-sequentially-Suc*)
apply (*subst filterlim-sequentially-Suc*)
apply (*rule tendsto-eq-intros*)
apply (*auto intro*!: *tendsto-divide-0*[*OF tendsto-const*] *filterlim-sup1*

```
        simp: at-infinity-eq-at-top-bot filterlim-real-sequentially)
    done
  ultimately show thesis
    by (intro that[of λi. b' − d / Suc (Suc i)])
       (auto simp add: real incseq-def intro!: divide-left-mono)
qed (insert ‹a < b›, auto)

lemma ereal-decseq-approx:
  fixes a b :: ereal
  assumes a < b
  obtains X :: nat ⇒ real where
    decseq X ⋀i. a < X i ⋀i. X i < b X ⟶ a
proof −
  have −b < −a using ‹a < b› by simp
  from ereal-incseq-approx[OF this] guess X .
  then show thesis
    apply (intro that[of λi. − X i])
    apply (auto simp add: uminus-ereal.simps[symmetric] decseq-def incseq-def
          simp del: uminus-ereal.simps)
   apply (metis ereal-minus-less-minus ereal-uminus-uminus ereal-Lim-uminus)+
    done
qed

lemma einterval-Icc-approximation:
  fixes a b :: ereal
  assumes a < b
  obtains u l :: nat ⇒ real where
    einterval a b = (⋃i. {l i .. u i})
    incseq u decseq l ⋀i. l i < u i ⋀i. a < l i ⋀i. u i < b
    l ⟶ a u ⟶ b
proof −
  from dense[OF ‹a < b›] obtain c where a < c c < b by safe
  from ereal-incseq-approx[OF ‹c < b›] guess u . note u = this
  from ereal-decseq-approx[OF ‹a < c›] guess l . note l = this
  { fix i from less-trans[OF ‹l i < c› ‹c < u i›] have l i < u i by simp }
  have einterval a b = (⋃i. {l i .. u i})
  proof (auto simp: einterval-iff)
    fix x assume a < ereal x ereal x < b
    have eventually (λi. ereal (l i) < ereal x) sequentially
      using l(4) ‹a < ereal x› by (rule order-tendstoD)
    moreover
    have eventually (λi. ereal x < ereal (u i)) sequentially
      using u(4) ‹ereal x< b› by (rule order-tendstoD)
    ultimately have eventually (λi. l i < x ∧ x < u i) sequentially
      by eventually-elim auto
    then show ∃i. l i ≤ x ∧ x ≤ u i
      by (auto intro: less-imp-le simp: eventually-sequentially)
  next
    fix x i assume l i ≤ x x ≤ u i
```

with ‹$a < ereal\ (l\ i)$› ‹$ereal\ (u\ i) < b$›
show $a < ereal\ x\ ereal\ x < b$
by (*auto simp del*: *ereal-less-eq*(3) *simp add*: *ereal-less-eq*(3)[*symmetric*])
qed
show *thesis*
by (*intro that*) *fact*+
qed

definition *interval-lebesgue-integral* :: *real measure* ⇒ *ereal* ⇒ *ereal* ⇒ (*real* ⇒ *'a*) ⇒ *'a*::{*banach*, *second-countable-topology*} **where**
interval-lebesgue-integral M a b f =
(*if a* ≤ *b then* (*LINT x*:*einterval a b*|*M*. *f x*) *else* − (*LINT x*:*einterval b a*|*M*. *f x*))

syntax
-ascii-interval-lebesgue-integral :: *pttrn* ⇒ *real* ⇒ *real* ⇒ *real measure* ⇒ *real* ⇒ *real*
((*5LINT -=-..-|-. -*) [*0*,*60*,*60*,*61*,*100*] *60*)

translations
LINT x=a..b|M. f == *CONST interval-lebesgue-integral M a b* (λ*x. f*)

definition *interval-lebesgue-integrable* :: *real measure* ⇒ *ereal* ⇒ *ereal* ⇒ (*real* ⇒ *'a*::{*banach*, *second-countable-topology*}) ⇒ *bool* **where**
interval-lebesgue-integrable M a b f =
(*if a* ≤ *b then set-integrable M* (*einterval a b*) *f else set-integrable M* (*einterval b a*) *f*)

syntax
-ascii-interval-lebesgue-borel-integral :: *pttrn* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real*
((*4LBINT -=-..-. -*) [*0*,*60*,*60*,*61*] *60*)

translations
LBINT x=a..b. f == *CONST interval-lebesgue-integral CONST lborel a b* (λ*x. f*)

lemma *interval-lebesgue-integral-cong*:
a ≤ *b* ⟹ (⋀*x. x* ∈ *einterval a b* ⟹ *f x* = *g x*) ⟹ *einterval a b* ∈ *sets M* ⟹
interval-lebesgue-integral M a b f = *interval-lebesgue-integral M a b g*
by (*auto intro*: *set-lebesgue-integral-cong simp*: *interval-lebesgue-integral-def*)

lemma *interval-lebesgue-integral-cong-AE*:
f ∈ *borel-measurable M* ⟹ *g* ∈ *borel-measurable M* ⟹
a ≤ *b* ⟹ *AE x* ∈ *einterval a b in M*. *f x* = *g x* ⟹ *einterval a b* ∈ *sets M* ⟹
interval-lebesgue-integral M a b f = *interval-lebesgue-integral M a b g*

by (*auto intro*: *set-lebesgue-integral-cong-AE simp*: *interval-lebesgue-integral-def*)

lemma *interval-integrable-mirror*:
shows *interval-lebesgue-integrable lborel a b* (λ*x*. *f* (−*x*)) ⟷
interval-lebesgue-integrable lborel (−*b*) (−*a*) *f*
proof −
have ∗: *indicator* (*einterval a b*) (− *x*) = (*indicator* (*einterval* (−*b*) (−*a*)) *x* :: *real*)
for *a b* :: *ereal* **and** *x* :: *real*
by (*cases a b rule*: *ereal2-cases*) (*auto simp*: *einterval-def split*: *split-indicator*)
show *?thesis*
unfolding *interval-lebesgue-integrable-def*
using *lborel-integrable-real-affine-iff*[*symmetric*, *of* −*1* λ*x*. *indicator* (*einterval* - -) *x* ∗$_R$ *f x 0*]
by (*simp add*: ∗)
qed

lemma *interval-lebesgue-integral-add* [*intro*, *simp*]:
fixes *M a b f*
assumes *interval-lebesgue-integrable M a b f interval-lebesgue-integrable M a b g*
shows *interval-lebesgue-integrable M a b* (λ*x*. *f x* + *g x*) **and**
interval-lebesgue-integral M a b (λ*x*. *f x* + *g x*) =
interval-lebesgue-integral M a b f + *interval-lebesgue-integral M a b g*
using *assms* **by** (*auto simp add*: *interval-lebesgue-integral-def interval-lebesgue-integrable-def*

field-simps)

lemma *interval-lebesgue-integral-diff* [*intro*, *simp*]:
fixes *M a b f*
assumes *interval-lebesgue-integrable M a b f*
interval-lebesgue-integrable M a b g
shows *interval-lebesgue-integrable M a b* (λ*x*. *f x* − *g x*) **and**
interval-lebesgue-integral M a b (λ*x*. *f x* − *g x*) =
interval-lebesgue-integral M a b f − *interval-lebesgue-integral M a b g*
using *assms* **by** (*auto simp add*: *interval-lebesgue-integral-def interval-lebesgue-integrable-def*

field-simps)

lemma *interval-lebesgue-integrable-mult-right* [*intro*, *simp*]:
fixes *M a b c* **and** *f* :: *real* ⇒ *'a*::{*banach*, *real-normed-field*, *second-countable-topology*}
shows (*c* ≠ *0* ⟹ *interval-lebesgue-integrable M a b f*) ⟹
interval-lebesgue-integrable M a b (λ*x*. *c* ∗ *f x*)
by (*simp add*: *interval-lebesgue-integrable-def*)

lemma *interval-lebesgue-integrable-mult-left* [*intro*, *simp*]:
fixes *M a b c* **and** *f* :: *real* ⇒ *'a*::{*banach*, *real-normed-field*, *second-countable-topology*}
shows (*c* ≠ *0* ⟹ *interval-lebesgue-integrable M a b f*) ⟹
interval-lebesgue-integrable M a b (λ*x*. *f x* ∗ *c*)
by (*simp add*: *interval-lebesgue-integrable-def*)

lemma *interval-lebesgue-integrable-divide* [*intro*, *simp*]:
fixes *M a b c* **and** *f* :: *real* ⇒ *'a*::{*banach*, *real-normed-field*, *field*, *second-countable-topology*}
shows (*c* ≠ *0* ⟹ *interval-lebesgue-integrable M a b f*) ⟹
interval-lebesgue-integrable M a b (λ*x*. *f x* / *c*)
by (*simp add*: *interval-lebesgue-integrable-def*)

lemma *interval-lebesgue-integral-mult-right* [*simp*]:
fixes *M a b c* **and** *f* :: *real* ⇒ *'a*::{*banach*, *real-normed-field*, *second-countable-topology*}
shows *interval-lebesgue-integral M a b* (λ*x*. *c* ∗ *f x*) =
c ∗ *interval-lebesgue-integral M a b f*
by (*simp add*: *interval-lebesgue-integral-def*)

lemma *interval-lebesgue-integral-mult-left* [*simp*]:
fixes *M a b c* **and** *f* :: *real* ⇒ *'a*::{*banach*, *real-normed-field*, *second-countable-topology*}
shows *interval-lebesgue-integral M a b* (λ*x*. *f x* ∗ *c*) =
interval-lebesgue-integral M a b f ∗ *c*
by (*simp add*: *interval-lebesgue-integral-def*)

lemma *interval-lebesgue-integral-divide* [*simp*]:
fixes *M a b c* **and** *f* :: *real* ⇒ *'a*::{*banach*, *real-normed-field*, *field*, *second-countable-topology*}
shows *interval-lebesgue-integral M a b* (λ*x*. *f x* / *c*) =
interval-lebesgue-integral M a b f / *c*
by (*simp add*: *interval-lebesgue-integral-def*)

lemma *interval-lebesgue-integral-uminus*:
interval-lebesgue-integral M a b (λ*x*. − *f x*) = − *interval-lebesgue-integral M a b*
f
by (*auto simp add*: *interval-lebesgue-integral-def interval-lebesgue-integrable-def*)

lemma *interval-lebesgue-integral-of-real*:
interval-lebesgue-integral M a b (λ*x*. *complex-of-real* (*f x*)) =
of-real (*interval-lebesgue-integral M a b f*)
unfolding *interval-lebesgue-integral-def*
by (*auto simp add*: *interval-lebesgue-integral-def set-integral-complex-of-real*)

lemma *interval-lebesgue-integral-le-eq*:
fixes *a b f*
assumes *a* ≤ *b*
shows *interval-lebesgue-integral M a b f* = (*LINT x* : *einterval a b* | *M*. *f x*)
using *assms* **by** (*auto simp add*: *interval-lebesgue-integral-def*)

lemma *interval-lebesgue-integral-gt-eq*:
fixes *a b f*
assumes *a* > *b*
shows *interval-lebesgue-integral M a b f* = −(*LINT x* : *einterval b a* | *M*. *f x*)
using *assms* **by** (*auto simp add*: *interval-lebesgue-integral-def less-imp-le einterval-def*)

lemma *interval-lebesgue-integral-gt-eq'*:

fixes *a b f*
assumes *a > b*
shows *interval-lebesgue-integral M a b f = − interval-lebesgue-integral M b a f*
using *assms* **by** (*auto simp add*: *interval-lebesgue-integral-def less-imp-le einterval-def*)

lemma *interval-integral-endpoints-same* [*simp*]: (*LBINT x=a..a. f x*) = *0*
by (*simp add*: *interval-lebesgue-integral-def einterval-same*)

lemma *interval-integral-endpoints-reverse*: (*LBINT x=a..b. f x*) = −(*LBINT x=b..a. f x*)
by (*cases a b rule*: *linorder-cases*) (*auto simp*: *interval-lebesgue-integral-def einterval-same*)

lemma *interval-integrable-endpoints-reverse*:
interval-lebesgue-integrable lborel a b f ⟷
interval-lebesgue-integrable lborel b a f
by (*cases a b rule*: *linorder-cases*) (*auto simp*: *interval-lebesgue-integrable-def einterval-same*)

lemma *interval-integral-reflect*:
(*LBINT x=a..b. f x*) = (*LBINT x=−b..−a. f* (−*x*))
proof (*induct a b rule*: *linorder-wlog*)
case (*sym a b*) **then show** *?case*
by (*auto simp add*: *interval-lebesgue-integral-def interval-integrable-endpoints-reverse*
split: *if-split-asm*)
next
case (*le a b*) **then show** *?case*
unfolding *interval-lebesgue-integral-def*
by (*subst set-integral-reflect*)
(*auto simp*: *interval-lebesgue-integrable-def einterval-iff*
ereal-uminus-le-reorder ereal-uminus-less-reorder not-less
uminus-ereal.simps[*symmetric*]
simp del: *uminus-ereal.simps*
intro!: *integral-cong*
split: *split-indicator*)
qed

lemma *interval-lebesgue-integral-0-infty*:
interval-lebesgue-integrable M 0 ∞ *f* ⟷ *set-integrable M* {*0<..*} *f*
interval-lebesgue-integral M 0 ∞ *f* = (*LINT x*:{*0<..*}|*M. f x*)
unfolding *zero-ereal-def*
by (*auto simp*: *interval-lebesgue-integral-le-eq interval-lebesgue-integrable-def*)

lemma *interval-integral-to-infinity-eq*: (*LINT x=ereal a..*∞ | *M. f x*) = (*LINT x* : {*a<..*} | *M. f x*)
unfolding *interval-lebesgue-integral-def* **by** *auto*

lemma *interval-integrable-to-infinity-eq*: (*interval-lebesgue-integrable M a* ∞ *f*) =

(*set-integrable M* {*a<..*} *f*)

unfolding *interval-lebesgue-integrable-def* **by** *auto*

lemma *interval-integral-zero* [*simp*]:
fixes *a b* :: *ereal*
shows*LBINT x=a..b. 0 = 0*
using *assms* **unfolding** *interval-lebesgue-integral-def einterval-eq*
by *simp*

lemma *interval-integral-const* [*intro*, *simp*]:
fixes *a b c* :: *real*
shows *interval-lebesgue-integrable lborel a b* (λ*x*. *c*) **and** *LBINT x=a..b. c = c* ∗ (*b* − *a*)
using *assms* **unfolding** *interval-lebesgue-integral-def interval-lebesgue-integrable-def einterval-eq*
by (*auto simp add*: *less-imp-le field-simps measure-def*)

lemma *interval-integral-cong-AE*:
assumes [*measurable*]: *f* ∈ *borel-measurable borel g* ∈ *borel-measurable borel*
assumes *AE x* ∈ *einterval* (*min a b*) (*max a b*) *in lborel. f x = g x*
shows *interval-lebesgue-integral lborel a b f = interval-lebesgue-integral lborel a b g*
using *assms*
proof (*induct a b rule*: *linorder-wlog*)
case (*sym a b*) **then show** *?case*
by (*simp add*: *min.commute max.commute interval-integral-endpoints-reverse*[*of a b*])
next
case (*le a b*) **then show** *?case*
by (*auto simp*: *interval-lebesgue-integral-def max-def min-def*
intro!: *set-lebesgue-integral-cong-AE*)
qed

lemma *interval-integral-cong*:
assumes ⋀*x*. *x* ∈ *einterval* (*min a b*) (*max a b*) ⟹ *f x = g x*
shows *interval-lebesgue-integral lborel a b f = interval-lebesgue-integral lborel a b g*
using *assms*
proof (*induct a b rule*: *linorder-wlog*)
case (*sym a b*) **then show** *?case*
by (*simp add*: *min.commute max.commute interval-integral-endpoints-reverse*[*of a b*])
next
case (*le a b*) **then show** *?case*
by (*auto simp*: *interval-lebesgue-integral-def max-def min-def*
intro!: *set-lebesgue-integral-cong*)
qed

lemma *interval-lebesgue-integrable-cong-AE*:
  $f \in$ *borel-measurable lborel* $\Longrightarrow$ $g \in$ *borel-measurable lborel* $\Longrightarrow$
  AE $x \in$ *einterval* (*min a b*) (*max a b*) *in lborel*. $f\ x = g\ x \Longrightarrow$
  interval-lebesgue-integrable lborel a b f = *interval-lebesgue-integrable lborel a b g*
 apply (*simp add*: *interval-lebesgue-integrable-def*)
 apply (*intro conjI impI set-integrable-cong-AE*)
 apply (*auto simp*: *min-def max-def*)
 done

lemma *interval-integrable-abs-iff*:
 fixes f :: *real* $\Rightarrow$ *real*
 shows $f \in$ *borel-measurable lborel* $\Longrightarrow$
  interval-lebesgue-integrable lborel a b ($\lambda x.\ |f\ x|$) = *interval-lebesgue-integrable lborel a b f*
 unfolding *interval-lebesgue-integrable-def*
 by (*subst* (*1 2*) *set-integrable-abs-iff'*) *simp-all*

lemma *interval-integral-Icc*:
 fixes *a b* :: *real*
 shows $a \le b \Longrightarrow$ (*LBINT* x=*a*..*b*. $f\ x$) = (*LBINT* $x : \{a..b\}$. $f\ x$)
 by (*auto intro*!: *set-integral-discrete-difference*[**where** X=$\{a, b\}$]
  simp add: *interval-lebesgue-integral-def*)

lemma *interval-integral-Icc'*:
 $a \le b \Longrightarrow$ (*LBINT* x=*a*..*b*. $f\ x$) = (*LBINT* $x : \{x.\ a \le$ *ereal* $x \wedge$ *ereal* $x \le b\}$. $f\ x$)
 by (*auto intro*!: *set-integral-discrete-difference*[**where** X=$\{$*real-of-ereal a*, *real-of-ereal b*$\}$]
  simp add: *interval-lebesgue-integral-def einterval-iff*)

lemma *interval-integral-Ioc*:
 $a \le b \Longrightarrow$ (*LBINT* x=*a*..*b*. $f\ x$) = (*LBINT* $x : \{a<..b\}$. $f\ x$)
 by (*auto intro*!: *set-integral-discrete-difference*[**where** X=$\{a, b\}$]
  simp add: *interval-lebesgue-integral-def einterval-iff*)

lemma *interval-integral-Ioc'*:
 $a \le b \Longrightarrow$ (*LBINT* x=*a*..*b*. $f\ x$) = (*LBINT* $x : \{x.\ a <$ *ereal* $x \wedge$ *ereal* $x \le b\}$. $f\ x$)
 by (*auto intro*!: *set-integral-discrete-difference*[**where** X=$\{$*real-of-ereal a*, *real-of-ereal b*$\}$]
  simp add: *interval-lebesgue-integral-def einterval-iff*)

lemma *interval-integral-Ico*:
 $a \le b \Longrightarrow$ (*LBINT* x=*a*..*b*. $f\ x$) = (*LBINT* $x : \{a..<b\}$. $f\ x$)
 by (*auto intro*!: *set-integral-discrete-difference*[**where** X=$\{a, b\}$]
  simp add: *interval-lebesgue-integral-def einterval-iff*)

lemma *interval-integral-Ioi*:

$|a| < \infty \Longrightarrow (LBINT\ x{=}a..\infty.\ f\ x) = (LBINT\ x : \{real\text{-}of\text{-}ereal\ a <..\}.\ f\ x)$
by (*auto simp add*: *interval-lebesgue-integral-def einterval-iff*)

lemma *interval-integral-Ioo*:
$a \leq b \Longrightarrow |a| < \infty ==> |b| < \infty \Longrightarrow (LBINT\ x{=}a..b.\ f\ x) = (LBINT\ x : \{real\text{-}of\text{-}ereal\ a <..< real\text{-}of\text{-}ereal\ b\}.\ f\ x)$
by (*auto simp add*: *interval-lebesgue-integral-def einterval-iff*)

lemma *interval-integral-discrete-difference*:
fixes $f :: real \Rightarrow 'b{::}\{banach, second\text{-}countable\text{-}topology\}$ **and** $a\ b :: ereal$
assumes *countable* X
and *eq*: $\bigwedge x.\ a \leq b \Longrightarrow a < x \Longrightarrow x < b \Longrightarrow x \notin X \Longrightarrow f\ x = g\ x$
and *anti-eq*: $\bigwedge x.\ b \leq a \Longrightarrow b < x \Longrightarrow x < a \Longrightarrow x \notin X \Longrightarrow f\ x = g\ x$
assumes $\bigwedge x.\ x \in X \Longrightarrow emeasure\ M\ \{x\} = 0$ $\bigwedge x.\ x \in X \Longrightarrow \{x\} \in sets\ M$
shows *interval-lebesgue-integral M a b f* = *interval-lebesgue-integral M a b g*
unfolding *interval-lebesgue-integral-def*
apply (*intro if-cong refl arg-cong*[**where** $f{=}\lambda x.\ -\ x$] *integral-discrete-difference*[*of* X] *assms*)
apply (*auto simp*: *eq anti-eq einterval-iff split*: *split-indicator*)
done

lemma *interval-integral-sum*:
fixes $a\ b\ c :: ereal$
assumes *integrable*: *interval-lebesgue-integrable lborel* (*min a* (*min b c*)) (*max a* (*max b c*)) *f*
shows $(LBINT\ x{=}a..b.\ f\ x) + (LBINT\ x{=}b..c.\ f\ x) = (LBINT\ x{=}a..c.\ f\ x)$
proof −
let *?I* = $\lambda a\ b.\ LBINT\ x{=}a..b.\ f\ x$
{ fix $a\ b\ c :: ereal$ **assume** *interval-lebesgue-integrable lborel a c f* $a \leq b\ b \leq c$
then have *ord*: $a \leq b\ b \leq c\ a \leq c$ **and** f': *set-integrable lborel* (*einterval a c*) *f*
by (*auto simp*: *interval-lebesgue-integrable-def*)
then have *f*: *set-borel-measurable borel* (*einterval a c*) *f*
by (*drule-tac borel-measurable-integrable*) *simp*
have $(LBINT\ x{:}einterval\ a\ c.\ f\ x) = (LBINT\ x{:}einterval\ a\ b \cup einterval\ b\ c.\ f\ x)$
proof (*rule set-integral-cong-set*)
show *AE x in lborel*. $(x \in einterval\ a\ b \cup einterval\ b\ c) = (x \in einterval\ a\ c)$
using *AE-lborel-singleton*[*of real-of-ereal b*] *ord*
by (*cases a b c rule*: *ereal3-cases*) (*auto simp*: *einterval-iff*)
qed (*insert ord*, *auto intro*!: *set-borel-measurable-subset*[*OF f*] *simp*: *einterval-iff*)
also have ... = $(LBINT\ x{:}einterval\ a\ b.\ f\ x) + (LBINT\ x{:}einterval\ b\ c.\ f\ x)$
using *ord*
by (*intro set-integral-Un-AE*) (*auto intro*!: *set-integrable-subset*[*OF f'*] *simp*: *einterval-iff not-less*)
finally have *?I a b + ?I b c = ?I a c*
using *ord* **by** (*simp add*: *interval-lebesgue-integral-def*)
} note *1 = this*
{ fix $a\ b\ c :: ereal$ **assume** *interval-lebesgue-integrable lborel a c f* $a \leq b\ b \leq c$
from *1*[*OF this*] **have** *?I b c + ?I a b = ?I a c*

```
    by (metis add.commute)
  } note 2 = this
  have 3: ⋀a b. b ≤ a ⟹ (LBINT x=a..b. f x) = - (LBINT x=b..a. f x)
    by (rule interval-integral-endpoints-reverse)
  show ?thesis
    using integrable
    by (cases a b b c a c rule: linorder-le-cases[case-product linorder-le-cases
linorder-cases])
      (simp-all add: min-absorb1 min-absorb2 max-absorb1 max-absorb2 field-simps
1 2 3)
qed

lemma interval-integrable-isCont:
  fixes a b and f :: real ⇒ 'a::{banach, second-countable-topology}
  shows (⋀x. min a b ≤ x ⟹ x ≤ max a b ⟹ isCont f x) ⟹
    interval-lebesgue-integrable lborel a b f
proof (induct a b rule: linorder-wlog)
  case (le a b) then show ?case
    by (auto simp: interval-lebesgue-integrable-def
          intro!: set-integrable-subset[OF borel-integrable-compact[of {a .. b}]]
                  continuous-at-imp-continuous-on)
qed (auto intro: interval-integrable-endpoints-reverse[THEN iffD1])

lemma interval-integrable-continuous-on:
  fixes a b :: real and f
  assumes a ≤ b and continuous-on {a..b} f
  shows interval-lebesgue-integrable lborel a b f
using assms unfolding interval-lebesgue-integrable-def apply simp
  by (rule set-integrable-subset, rule borel-integrable-atLeastAtMost' [of a b], auto)

lemma interval-integral-eq-integral:
  fixes f :: real ⇒ 'a::euclidean-space
  shows a ≤ b ⟹ set-integrable lborel {a..b} f ⟹ LBINT x=a..b. f x = integral
{a..b} f
  by (subst interval-integral-Icc, simp) (rule set-borel-integral-eq-integral)

lemma interval-integral-eq-integral':
  fixes f :: real ⇒ 'a::euclidean-space
  shows a ≤ b ⟹ set-integrable lborel (einterval a b) f ⟹ LBINT x=a..b. f x
= integral (einterval a b) f
 by (subst interval-lebesgue-integral-le-eq, simp) (rule set-borel-integral-eq-integral)



lemma interval-integral-Icc-approx-nonneg:
  fixes a b :: ereal
  assumes a < b
  fixes u l :: nat ⇒ real
  assumes  approx: einterval a b = (⋃i. {l i .. u i})
```

```
    incseq u decseq l ⋀i. l i < u i ⋀i. a < l i ⋀i. u i < b
    l ⟶ a u ⟶ b
  fixes f :: real ⇒ real
  assumes f-integrable: ⋀i. set-integrable lborel {l i..u i} f
  assumes f-nonneg: AE x in lborel. a < ereal x ⟶ ereal x < b ⟶ 0 ≤ f x
  assumes f-measurable: set-borel-measurable lborel (einterval a b) f
  assumes lbint-lim: (λi. LBINT x=l i.. u i. f x) ⟶ C
  shows
    set-integrable lborel (einterval a b) f
    (LBINT x=a..b. f x) = C
proof −
  have 1: ⋀i. set-integrable lborel {l i..u i} f by (rule f-integrable)
  have 2: AE x in lborel. mono (λn. indicator {l n..u n} x *R f x)
  proof −
     from f-nonneg have AE x in lborel. ∀ i. l i ≤ x ⟶ x ≤ u i ⟶ 0 ≤ f x
      by eventually-elim
      (metis approx(5) approx(6) dual-order.strict-trans1 ereal-less-eq(3) le-less-trans)
    then show ?thesis
      apply eventually-elim
      apply (auto simp: mono-def split: split-indicator)
      apply (metis approx(3) decseqD order-trans)
      apply (metis approx(2) incseqD order-trans)
      done
  qed
   have 3: AE x in lborel. (λi. indicator {l i..u i} x *R f x) ⟶ indicator
(einterval a b) x *R f x
  proof −
    { fix x i assume l i ≤ x x ≤ u i
      then have eventually (λi. l i ≤ x ∧ x ≤ u i) sequentially
        apply (auto simp: eventually-sequentially intro!: exI[of - i])
        apply (metis approx(3) decseqD order-trans)
        apply (metis approx(2) incseqD order-trans)
        done
      then have eventually (λi. f x * indicator {l i..u i} x = f x) sequentially
        by eventually-elim auto }
    then show ?thesis
    unfolding approx(1) by (auto intro!: AE-I2 Lim-eventually split: split-indicator)
  qed
  have 4: (λi. ∫ x. indicator {l i..u i} x *R f x ∂lborel) ⟶ C
    using lbint-lim by (simp add: interval-integral-Icc approx less-imp-le)
  have 5: set-borel-measurable lborel (einterval a b) f by (rule assms)
  have (LBINT x=a..b. f x) = lebesgue-integral lborel (λx. indicator (einterval a
b) x *R f x)
    using assms by (simp add: interval-lebesgue-integral-def less-imp-le)
  also have ... = C by (rule integral-monotone-convergence [OF 1 2 3 4 5])
  finally show (LBINT x=a..b. f x) = C .

  show set-integrable lborel (einterval a b) f
    by (rule integrable-monotone-convergence[OF 1 2 3 4 5])
```

qed

lemma *interval-integral-Icc-approx-integrable*:
fixes *u l* :: *nat* ⇒ *real* **and** *a b* :: *ereal*
fixes *f* :: *real* ⇒ *'a*::{*banach*, *second-countable-topology*}
assumes *a* < *b*
assumes *approx*: *einterval a b* = (⋃ *i*. {*l i* .. *u i*})
incseq u decseq l ⋀*i*. *l i* < *u i* ⋀*i*. *a* < *l i* ⋀*i*. *u i* < *b*
l ⟶ *a u* ⟶ *b*
assumes *f-integrable*: *set-integrable lborel* (*einterval a b*) *f*
shows (λ*i*. *LBINT x*=*l i*.. *u i*. *f x*) ⟶ (*LBINT x*=*a*..*b*. *f x*)
proof −
have (λ*i*. *LBINT x*:{*l i*.. *u i*}. *f x*) ⟶ (*LBINT x*:*einterval a b*. *f x*)
proof (*rule integral-dominated-convergence*)
show *integrable lborel* (λ*x*. *norm* (*indicator* (*einterval a b*) *x* ∗*R f x*))
by (*rule integrable-norm*) *fact*
show *set-borel-measurable lborel* (*einterval a b*) *f*
using *f-integrable* **by** (*rule borel-measurable-integrable*)
then show ⋀*i*. *set-borel-measurable lborel* {*l i*..*u i*} *f*
by (*rule set-borel-measurable-subset*) (*auto simp*: *approx*)
show ⋀*i*. *AE x in lborel*. *norm* (*indicator* {*l i*..*u i*} *x* ∗*R f x*) ≤ *norm* (*indicator* (*einterval a b*) *x* ∗*R f x*)
by (*intro AE-I2*) (*auto simp*: *approx split*: *split-indicator*)
show *AE x in lborel*. (λ*i*. *indicator* {*l i*..*u i*} *x* ∗*R f x*) ⟶ *indicator* (*einterval a b*) *x* ∗*R f x*
proof (*intro AE-I2 tendsto-intros Lim-eventually*)
fix *x*
{ fix *i* **assume** *l i* ≤ *x x* ≤ *u i*
with ‹*incseq u*›[*THEN incseqD*, *of i*] ‹*decseq l*›[*THEN decseqD*, *of i*]
have *eventually* (λ*i*. *l i* ≤ *x* ∧ *x* ≤ *u i*) *sequentially*
by (*auto simp*: *eventually-sequentially decseq-def incseq-def intro*: *order-trans*)
}
then show *eventually* (λ*xa*. *indicator* {*l xa*..*u xa*} *x* = (*indicator* (*einterval a b*) *x*::*real*)) *sequentially*
using *approx order-tendstoD*(*2*)[*OF* ‹*l* ⟶ *a*›, *of x*] *order-tendstoD*(*1*)[*OF* ‹*u* ⟶ *b*›, *of x*]
by (*auto split*: *split-indicator*)
qed
qed
with ‹*a* < *b*› ‹⋀*i*. *l i* < *u i*› **show** *?thesis*
by (*simp add*: *interval-lebesgue-integral-le-eq*[*symmetric*] *interval-integral-Icc less-imp-le*)
qed

lemma *interval-integral-FTC-finite*:
fixes *f F* :: *real* ⇒ *'a*::*euclidean-space* **and** *a b* :: *real*
assumes *f*: *continuous-on* {*min a b*..*max a b*} *f*
assumes *F*: ⋀*x*. *min a b* ≤ *x* ⟹ *x* ≤ *max a b* ⟹ (*F has-vector-derivative* (*f x*)) (*at x within* {*min a b*..*max a b*})
shows (*LBINT x*=*a*..*b*. *f x*) = *F b* − *F a*
apply (*case-tac a* ≤ *b*)
apply (*subst interval-integral-Icc*, *simp*)
apply (*rule integral-FTC-atLeastAtMost*, *assumption*)
apply (*metis F max-def min-def*)
using *f* **apply** (*simp add*: *min-absorb1 max-absorb2*)
apply (*subst interval-integral-endpoints-reverse*)
apply (*subst interval-integral-Icc*, *simp*)
apply (*subst integral-FTC-atLeastAtMost*, *auto*)
apply (*metis F max-def min-def*)
using *f* **by** (*simp add*: *min-absorb2 max-absorb1*)

lemma *interval-integral-FTC-nonneg*:
fixes *f F* :: *real* ⇒ *real* **and** *a b* :: *ereal*
assumes *a* < *b*
assumes *F*: ⋀*x*. *a* < *ereal x* ⟹ *ereal x* < *b* ⟹ *DERIV F x* :> *f x*
assumes *f*: ⋀*x*. *a* < *ereal x* ⟹ *ereal x* < *b* ⟹ *isCont f x*
assumes *f-nonneg*: *AE x in lborel*. *a* < *ereal x* ⟶ *ereal x* < *b* ⟶ 0 ≤ *f x*
assumes *A*: ((*F* ∘ *real-of-ereal*) ⟶⟶ *A*) (*at-right a*)
assumes *B*: ((*F* ∘ *real-of-ereal*) ⟶⟶ *B*) (*at-left b*)
shows
set-integrable lborel (*einterval a b*) *f*
(*LBINT x*=*a*..*b*. *f x*) = *B* − *A*
proof −
from *einterval-Icc-approximation*[*OF* ‹*a* < *b*›] **guess** *u l* **. note** *approx* = *this*
have [*simp*]: ⋀*x i*. *l i* ≤ *x* ⟹ *a* < *ereal x*
by (*rule order-less-le-trans*, *rule approx*, *force*)
have [*simp*]: ⋀*x i*. *x* ≤ *u i* ⟹ *ereal x* < *b*
by (*rule order-le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *rule approx*)
have *FTCi*: ⋀*i*. (*LBINT x*=*l i*..*u i*. *f x*) = *F* (*u i*) − *F* (*l i*)
using *assms approx* **apply** (*intro interval-integral-FTC-finite*)
apply (*auto simp add*: *less-imp-le min-def max-def*
has-field-derivative-iff-has-vector-derivative[*symmetric*])
apply (*rule continuous-at-imp-continuous-on*, *auto intro*!: *f*)
by (*rule DERIV-subset* [*OF F*], *auto*)
have *1*: ⋀*i*. *set-integrable lborel* {*l i*..*u i*} *f*
proof −
fix *i* **show** *set-integrable lborel* {*l i* .. *u i*} *f*
using ‹*a* < *l i*› ‹*u i* < *b*›
by (*intro borel-integrable-compact f continuous-at-imp-continuous-on compact-Icc ballI*)
(*auto simp del*: *ereal-less-eq simp add*: *ereal-less-eq*(*3*)[*symmetric*])

qed
have *2*: *set-borel-measurable lborel* (*einterval a b*) *f*
by (*auto simp del*: *real-scaleR-def intro*!: *set-borel-measurable-continuous*
simp: *continuous-on-eq-continuous-at einterval-iff f*)
have *3*: (λ*i*. *LBINT x=l i..u i*. *f x*) ⟶ *B* − *A*
apply (*subst FTCi*)
apply (*intro tendsto-intros*)
using *B approx* **unfolding** *tendsto-at-iff-sequentially comp-def*
using *tendsto-at-iff-sequentially*[**where** *'a=real*]
apply (*elim allE*[*of* - λ*i*. *ereal* (*u i*)], *auto*)
using *A approx* **unfolding** *tendsto-at-iff-sequentially comp-def*
by (*elim allE*[*of* - λ*i*. *ereal* (*l i*)], *auto*)
show (*LBINT x=a..b*. *f x*) = *B* − *A*
by (*rule interval-integral-Icc-approx-nonneg* [*OF* ‹*a* < *b*› *approx 1 f-nonneg 2 3*])
show *set-integrable lborel* (*einterval a b*) *f*
by (*rule interval-integral-Icc-approx-nonneg* [*OF* ‹*a* < *b*› *approx 1 f-nonneg 2 3*])
qed

lemma *interval-integral-FTC-integrable*:
fixes *f F* :: *real* ⇒ *'a*::*euclidean-space* **and** *a b* :: *ereal*
assumes *a* < *b*
assumes *F*: ⋀*x*. *a* < *ereal x* ⟹ *ereal x* < *b* ⟹ (*F has-vector-derivative f x*) (*at x*)
assumes *f*: ⋀*x*. *a* < *ereal x* ⟹ *ereal x* < *b* ⟹ *isCont f x*
assumes *f-integrable*: *set-integrable lborel* (*einterval a b*) *f*
assumes *A*: ((*F* ∘ *real-of-ereal*) ⟶ *A*) (*at-right a*)
assumes *B*: ((*F* ∘ *real-of-ereal*) ⟶ *B*) (*at-left b*)
shows (*LBINT x=a..b*. *f x*) = *B* − *A*
proof −
from *einterval-Icc-approximation*[*OF* ‹*a* < *b*›] **guess** *u l* **. note** *approx* = *this*
have [*simp*]: ⋀*x i*. *l i* ≤ *x* ⟹ *a* < *ereal x*
by (*rule order-less-le-trans*, *rule approx*, *force*)
have [*simp*]: ⋀*x i*. *x* ≤ *u i* ⟹ *ereal x* < *b*
by (*rule order-le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *rule approx*)
have *FTCi*: ⋀*i*. (*LBINT x=l i..u i*. *f x*) = *F* (*u i*) − *F* (*l i*)
using *assms approx*
by (*auto simp add*: *less-imp-le min-def max-def*
intro!: *f continuous-at-imp-continuous-on interval-integral-FTC-finite*
intro: *has-vector-derivative-at-within*)
have (λ*i*. *LBINT x=l i..u i*. *f x*) ⟶ *B* − *A*
apply (*subst FTCi*)
apply (*intro tendsto-intros*)
using *B approx* **unfolding** *tendsto-at-iff-sequentially comp-def*
apply (*elim allE*[*of* - λ*i*. *ereal* (*u i*)], *auto*)
using *A approx* **unfolding** *tendsto-at-iff-sequentially comp-def*
by (*elim allE*[*of* - λ*i*. *ereal* (*l i*)], *auto*)
moreover have (λ*i*. *LBINT x=l i..u i*. *f x*) ⟶ (*LBINT x=a..b*. *f x*)

by (*rule interval-integral-Icc-approx-integrable* [*OF* ‹$a < b$› *approx f-integrable*])
ultimately show *?thesis*
by (*elim LIMSEQ-unique*)
qed

lemma *interval-integral-FTC2*:
fixes *a b c* :: *real* **and** *f* :: *real* ⇒ *'a*::*euclidean-space*
assumes $a \leq c$ $c \leq b$
and *contf*: *continuous-on* {*a*..*b*} *f*
fixes *x* :: *real*
assumes $a \leq x$ **and** $x \leq b$
shows ((λ*u*. *LBINT y*=*c*..*u*. *f y*) *has-vector-derivative* (*f x*)) (*at x within* {*a*..*b*})
proof −
let *?F* = (λ*u*. *LBINT y*=*a*..*u*. *f y*)
have *intf*: *set-integrable lborel* {*a*..*b*} *f*
by (*rule borel-integrable-atLeastAtMost'*, *rule contf*)
have ((λ*u*. *integral* {*a*..*u*} *f*) *has-vector-derivative f x*) (*at x within* {*a*..*b*})
apply (*intro integral-has-vector-derivative*)
using ‹$a \leq x$› ‹$x \leq b$› **by** (*intro continuous-on-subset* [*OF contf*], *auto*)
then have ((λ*u*. *integral* {*a*..*u*} *f*) *has-vector-derivative* (*f x*)) (*at x within* {*a*..*b*})
by *simp*
then have (*?F has-vector-derivative* (*f x*)) (*at x within* {*a*..*b*})
by (*rule has-vector-derivative-weaken*)
(*auto intro*!: *assms interval-integral-eq-integral*[*symmetric*] *set-integrable-subset* [*OF intf*])
then have ((λ*x*. (*LBINT y*=*c*..*a*. *f y*) + *?F x*) *has-vector-derivative* (*f x*)) (*at x within* {*a*..*b*})
by (*auto intro*!: *derivative-eq-intros*)
then show *?thesis*
proof (*rule has-vector-derivative-weaken*)
fix *u* **assume** $u \in \{a \mathinner{..} b\}$
then show (*LBINT y*=*c*..*a*. *f y*) + (*LBINT y*=*a*..*u*. *f y*) = (*LBINT y*=*c*..*u*. *f y*)
using *assms*
apply (*intro interval-integral-sum*)
apply (*auto simp add*: *interval-lebesgue-integrable-def simp del*: *real-scaleR-def*)
by (*rule set-integrable-subset* [*OF intf*], *auto simp add*: *min-def max-def*)
qed (*insert assms*, *auto*)
qed

lemma *einterval-antiderivative*:
fixes *a b* :: *ereal* **and** *f* :: *real* ⇒ *'a*::*euclidean-space*
assumes $a < b$ **and** *contf*: $\bigwedge x$ :: *real*. $a < x \Longrightarrow x < b \Longrightarrow$ *isCont f x*
shows ∃ *F*. ∀ *x* :: *real*. $a < x \longrightarrow x < b \longrightarrow$ (*F has-vector-derivative f x*) (*at x*)
proof −
from *einterval-nonempty* [*OF* ‹$a < b$›] **obtain** *c* :: *real* **where** [*simp*]: $a < c$ $c < b$

```
    by (auto simp add: einterval-def)
  let ?F = (λu. LBINT y=c..u. f y)
  show ?thesis
  proof (rule exI, clarsimp)
    fix x :: real
    assume [simp]: a < x x < b
    have 1: a < min c x by simp
    from einterval-nonempty [OF 1] obtain d :: real where [simp]: a < d d < c
d < x
      by (auto simp add: einterval-def)
    have 2: max c x < b by simp
    from einterval-nonempty [OF 2] obtain e :: real where [simp]: c < e x < e
e < b
      by (auto simp add: einterval-def)
    show (?F has-vector-derivative f x) (at x)

      unfolding has-vector-derivative-def
      apply (subst has-derivative-within-open [of - {d<..<e}, symmetric], auto)
      apply (subst has-vector-derivative-def [symmetric])
      apply (rule has-vector-derivative-within-subset [of - - - {d..e}])
      apply (rule interval-integral-FTC2, auto simp add: less-imp-le)
      apply (rule continuous-at-imp-continuous-on)
      apply (auto intro!: contf)
      apply (rule order-less-le-trans, rule ‹a < d›, auto)
      apply (rule order-le-less-trans) prefer 2
      by (rule ‹e < b›, auto)
  qed
qed



lemma interval-integral-substitution-finite:
  fixes a b :: real and f :: real ⇒ 'a::euclidean-space
  assumes a ≤ b
  and derivg: ⋀x. a ≤ x ⟹ x ≤ b ⟹ (g has-real-derivative (g' x)) (at x within
{a..b})
  and contf : continuous-on (g ` {a..b}) f
  and contg': continuous-on {a..b} g'
  shows LBINT x=a..b. g' x *R f (g x) = LBINT y=g a..g b. f y
proof−
  have v-derivg: ⋀x. a ≤ x ⟹ x ≤ b ⟹ (g has-vector-derivative (g' x)) (at x
within {a..b})
    using derivg unfolding has-field-derivative-iff-has-vector-derivative .
  then have contg [simp]: continuous-on {a..b} g
    by (rule continuous-on-vector-derivative) auto
  have 1: ⋀u. min (g a) (g b) ≤ u ⟹ u ≤ max (g a) (g b) ⟹
     ∃x∈{a..b}. u = g x
    apply (case-tac g a ≤ g b)
    apply (auto simp add: min-def max-def less-imp-le)
```

apply (*frule* (*1*) *IVT'* [*of g*], *auto simp add*: *assms*)
by (*frule* (*1*) *IVT2'* [*of g*], *auto simp add*: *assms*)
from *contg* ‹$a \le b$› **have** $\exists c\ d.\ g\ `\ \{a..b\} = \{c..d\} \wedge c \le d$
by (*elim continuous-image-closed-interval*)
then obtain *c d* **where** *g-im*: $g\ `\ \{a..b\} = \{c..d\}$ **and** $c \le d$ **by** *auto*
have $\exists F.\ \forall x{\in}\{a..b\}.\ (F$ *has-vector-derivative* $(f\ (g\ x)))\ (at\ (g\ x)$ *within* $(g\ `\ \{a..b\}))$
apply (*rule exI*, *auto*, *subst g-im*)
apply (*rule interval-integral-FTC2* [*of c c d*])
using ‹$c \le d$› **apply** *auto*
apply (*rule continuous-on-subset* [*OF contf*])
using *g-im* **by** *auto*
then guess *F* **..**
then have *derivF*: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow$
$(F$ *has-vector-derivative* $(f\ (g\ x)))\ (at\ (g\ x)$ *within* $(g\ `\ \{a..b\}))$ **by** *auto*
have *contf2*: *continuous-on* $\{min\ (g\ a)\ (g\ b)..max\ (g\ a)\ (g\ b)\}\ f$
apply (*rule continuous-on-subset* [*OF contf*])
apply (*auto simp add*: *image-def*)
by (*erule 1*)
have *contfg*: *continuous-on* $\{a..b\}\ (\lambda x.\ f\ (g\ x))$
by (*blast intro*: *continuous-on-compose2 contf contg*)
have *LBINT* $x{=}a..b.\ g'\ x *_R f\ (g\ x) = F\ (g\ b) - F\ (g\ a)$
apply (*subst interval-integral-Icc*, *simp add*: *assms*)
apply (*rule integral-FTC-atLeastAtMost*[*of a b* $\lambda x.\ F\ (g\ x)$, *OF* ‹$a \le b$›])
apply (*rule vector-diff-chain-within*[*OF v-derivg derivF*, *unfolded comp-def*])
apply (*auto intro*!: *continuous-on-scaleR contg' contfg*)
done
moreover have *LBINT* $y{=}(g\ a)..(g\ b).\ f\ y = F\ (g\ b) - F\ (g\ a)$
apply (*rule interval-integral-FTC-finite*)
apply (*rule contf2*)
apply (*frule* (*1*) *1*, *auto*)
apply (*rule has-vector-derivative-within-subset* [*OF derivF*])
apply (*auto simp add*: *image-def*)
by (*rule 1*, *auto*)
ultimately show *?thesis* **by** *simp*
qed

lemma *interval-integral-substitution-integrable*:
fixes $f :: real \Rightarrow 'a{::}euclidean\text{-}space$ **and** *a b u v* :: *ereal*
assumes $a < b$
and *deriv-g*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow DERIV\ g\ x :> g'\ x$
and *contf*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow isCont\ f\ (g\ x)$
and *contg'*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow isCont\ g'\ x$
and *g'-nonneg*: $\bigwedge x.\ a \le ereal\ x \Longrightarrow ereal\ x \le b \Longrightarrow 0 \le g'\ x$
and *A*: $((ereal \circ g \circ real\text{-}of\text{-}ereal) \longrightarrow A)\ (at\text{-}right\ a)$
and *B*: $((ereal \circ g \circ real\text{-}of\text{-}ereal) \longrightarrow B)\ (at\text{-}left\ b)$
and *integrable*: *set-integrable lborel* $(einterval\ a\ b)\ (\lambda x.\ g'\ x *_R f\ (g\ x))$

and *integrable2*: *set-integrable lborel* (*einterval A B*) (λ*x*. *f x*)
shows (*LBINT x*=*A*..*B*. *f x*) = (*LBINT x*=*a*..*b*. *g*′ *x* ∗$_R$ *f* (*g x*))
proof −
from *einterval-Icc-approximation*[*OF* ‹*a* < *b*›] **guess** *u l* **. note** *approx* [*simp*] = *this*
note *less-imp-le* [*simp*]
have [*simp*]: ⋀*x i*. *l i* ≤ *x* ⟹ *a* < *ereal x*
by (*rule order-less-le-trans*, *rule approx*, *force*)
have [*simp*]: ⋀*x i*. *x* ≤ *u i* ⟹ *ereal x* < *b*
by (*rule order-le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *rule approx*)
have [*simp*]: ⋀*i*. *l i* < *b*
apply (*rule order-less-trans*) **prefer** *2*
by (*rule approx*, *auto*, *rule approx*)
have [*simp*]: ⋀*i*. *a* < *u i*
by (*rule order-less-trans*, *rule approx*, *auto*, *rule approx*)
have [*simp*]: ⋀*i j*. *i* ≤ *j* ⟹ *l j* ≤ *l i* **by** (*rule decseqD*, *rule approx*)
have [*simp*]: ⋀*i j*. *i* ≤ *j* ⟹ *u i* ≤ *u j* **by** (*rule incseqD*, *rule approx*)
have *g-nondec* [*simp*]: ⋀*x y*. *a* < *x* ⟹ *x* ≤ *y* ⟹ *y* < *b* ⟹ *g x* ≤ *g y*
apply (*erule DERIV-nonneg-imp-nondecreasing*, *auto*)
apply (*rule exI*, *rule conjI*, *rule deriv-g*)
apply (*erule order-less-le-trans*, *auto*)
apply (*rule order-le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *auto*)
apply (*rule g*′*-nonneg*)
apply (*rule less-imp-le*, *erule order-less-le-trans*, *auto*)
by (*rule less-imp-le*, *rule le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *auto*)
have *A* ≤ *B* **and** *un*: *einterval A B* = (⋃*i*. {*g*(*l i*)<..<*g*(*u i*)})
proof −
have *A2*: (λ*i*. *g* (*l i*)) ⟶ *A*
using *A* **apply** (*auto simp add*: *einterval-def tendsto-at-iff-sequentially comp-def*)
by (*drule-tac x* = λ*i*. *ereal* (*l i*) **in** *spec*, *auto*)
hence *A3*: ⋀*i*. *g* (*l i*) ≥ *A*
by (*intro decseq-le*, *auto simp add*: *decseq-def*)
have *B2*: (λ*i*. *g* (*u i*)) ⟶ *B*
using *B* **apply** (*auto simp add*: *einterval-def tendsto-at-iff-sequentially comp-def*)
by (*drule-tac x* = λ*i*. *ereal* (*u i*) **in** *spec*, *auto*)
hence *B3*: ⋀*i*. *g* (*u i*) ≤ *B*
by (*intro incseq-le*, *auto simp add*: *incseq-def*)
show *A* ≤ *B*
apply (*rule order-trans* [*OF A3* [*of 0*]])
apply (*rule order-trans* [*OF* - *B3* [*of 0*]])
by *auto*
{ fix *x* :: *real*
assume *A* < *x* **and** *x* < *B*
then have *eventually* (λ*i*. *ereal* (*g* (*l i*)) < *x* ∧ *x* < *ereal* (*g* (*u i*))) *sequentially*
apply (*intro eventually-conj order-tendstoD*)
by (*rule A2*, *assumption*, *rule B2*, *assumption*)
hence ∃*i*. *g* (*l i*) < *x* ∧ *x* < *g* (*u i*)
by (*simp add*: *eventually-sequentially*, *auto*)

```
    } note AB = this
    show einterval A B = (⋃ i. {g(l i)<..<g(u i)})
      apply (auto simp add: einterval-def)
      apply (erule (1) AB)
      apply (rule order-le-less-trans, rule A3, simp)
      apply (rule order-less-le-trans) prefer 2
      by (rule B3, simp)
  qed

  {
    fix i
    have (LBINT x=l i.. u i. g' x *R f (g x)) = (LBINT y=g (l i)..g (u i). f y)
      apply (rule interval-integral-substitution-finite, auto)
      apply (rule DERIV-subset)
      unfolding has-field-derivative-iff-has-vector-derivative[symmetric]
      apply (rule deriv-g)
      apply (auto intro!: continuous-at-imp-continuous-on contf contg')
      done
  } note eq1 = this
  have (λi. LBINT x=l i..u i. g' x *R f (g x)) ⟶ (LBINT x=a..b. g' x *R f
(g x))
    apply (rule interval-integral-Icc-approx-integrable [OF ‹a < b› approx])
    by (rule assms)
  hence 2: (λi. (LBINT y=g (l i)..g (u i). f y)) ⟶ (LBINT x=a..b. g' x *R
f (g x))
    by (simp add: eq1)
  have incseq: incseq (λi. {g (l i)<..<g (u i)})
    apply (auto simp add: incseq-def)
    apply (rule order-le-less-trans)
    prefer 2 apply (assumption, auto)
    by (erule order-less-le-trans, rule g-nondec, auto)
  have (λi. (LBINT y=g (l i)..g (u i). f y)) ⟶ (LBINT x = A..B. f x)
    apply (subst interval-lebesgue-integral-le-eq, auto simp del: real-scaleR-def)
    apply (subst interval-lebesgue-integral-le-eq, rule ‹A ≤ B›)
    apply (subst un, rule set-integral-cont-up, auto simp del: real-scaleR-def)
    apply (rule incseq)
    apply (subst un [symmetric])
    by (rule integrable2)
  thus ?thesis by (intro LIMSEQ-unique [OF - 2])
qed



lemma interval-integral-substitution-nonneg:
  fixes f g g'::: real ⇒ real and a b u v :: ereal
  assumes a < b
  and deriv-g: ⋀x. a < ereal x ⟹ ereal x < b ⟹ DERIV g x :> g' x
  and contf: ⋀x. a < ereal x ⟹ ereal x < b ⟹ isCont f (g x)
  and contg': ⋀x. a < ereal x ⟹ ereal x < b ⟹ isCont g' x
```

and *f-nonneg*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow 0 \leq f\ (g\ x)$
and *g'-nonneg*: $\bigwedge x.\ a \leq ereal\ x \Longrightarrow ereal\ x \leq b \Longrightarrow 0 \leq g'\ x$
and *A*: $((ereal \circ g \circ real\text{-}of\text{-}ereal) \longrightarrow A)\ (at\text{-}right\ a)$
and *B*: $((ereal \circ g \circ real\text{-}of\text{-}ereal) \longrightarrow B)\ (at\text{-}left\ b)$
and *integrable-fg*: *set-integrable lborel* $(einterval\ a\ b)\ (\lambda x.\ f\ (g\ x) * g'\ x)$
shows
set-integrable lborel $(einterval\ A\ B)\ f$
$(LBINT\ x{=}A..B.\ f\ x) = (LBINT\ x{=}a..b.\ (f\ (g\ x) * g'\ x))$
proof −
from *einterval-Icc-approximation*[*OF* ‹$a < b$›] **guess** $u\ l$ **. note** *approx* [*simp*] = *this*
note *less-imp-le* [*simp*]
have [*simp*]: $\bigwedge x\ i.\ l\ i \leq x \Longrightarrow a < ereal\ x$
by (*rule order-less-le-trans*, *rule approx*, *force*)
have [*simp*]: $\bigwedge x\ i.\ x \leq u\ i \Longrightarrow ereal\ x < b$
by (*rule order-le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *rule approx*)
have [*simp*]: $\bigwedge i.\ l\ i < b$
apply (*rule order-less-trans*) **prefer** *2*
by (*rule approx*, *auto*, *rule approx*)
have [*simp*]: $\bigwedge i.\ a < u\ i$
by (*rule order-less-trans*, *rule approx*, *auto*, *rule approx*)
have [*simp*]: $\bigwedge i\ j.\ i \leq j \Longrightarrow l\ j \leq l\ i$ **by** (*rule decseqD*, *rule approx*)
have [*simp*]: $\bigwedge i\ j.\ i \leq j \Longrightarrow u\ i \leq u\ j$ **by** (*rule incseqD*, *rule approx*)
have *g-nondec* [*simp*]: $\bigwedge x\ y.\ a < x \Longrightarrow x \leq y \Longrightarrow y < b \Longrightarrow g\ x \leq g\ y$
apply (*erule DERIV-nonneg-imp-nondecreasing*, *auto*)
apply (*rule exI*, *rule conjI*, *rule deriv-g*)
apply (*erule order-less-le-trans*, *auto*)
apply (*rule order-le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *auto*)
apply (*rule g'-nonneg*)
apply (*rule less-imp-le*, *erule order-less-le-trans*, *auto*)
by (*rule less-imp-le*, *rule le-less-trans*, *subst ereal-less-eq*(*3*), *assumption*, *auto*)
have $A \leq B$ **and** *un*: $einterval\ A\ B = (\bigcup i.\ \{g(l\ i){<}..{<}g(u\ i)\})$
proof −
have *A2*: $(\lambda i.\ g\ (l\ i)) \longrightarrow A$
using *A* **apply** (*auto simp add*: *einterval-def tendsto-at-iff-sequentially comp-def*)
by (*drule-tac* $x = \lambda i.\ ereal\ (l\ i)$ **in** *spec*, *auto*)
hence *A3*: $\bigwedge i.\ g\ (l\ i) \geq A$
by (*intro decseq-le*, *auto simp add*: *decseq-def*)
have *B2*: $(\lambda i.\ g\ (u\ i)) \longrightarrow B$
using *B* **apply** (*auto simp add*: *einterval-def tendsto-at-iff-sequentially comp-def*)
by (*drule-tac* $x = \lambda i.\ ereal\ (u\ i)$ **in** *spec*, *auto*)
hence *B3*: $\bigwedge i.\ g\ (u\ i) \leq B$
by (*intro incseq-le*, *auto simp add*: *incseq-def*)
show $A \leq B$
apply (*rule order-trans* [*OF A3* [*of 0*]])
apply (*rule order-trans* [*OF* - *B3* [*of 0*]])
by *auto*
{ fix $x :: real$

```
    assume A < x and x < B
   then have eventually (λi. ereal (g (l i)) < x ∧ x < ereal (g (u i))) sequentially
     apply (intro eventually-conj order-tendstoD)
     by (rule A2, assumption, rule B2, assumption)
    hence ∃ i. g (l i) < x ∧ x < g (u i)
     by (simp add: eventually-sequentially, auto)
  } note AB = this
  show einterval A B = (⋃ i. {g(l i)<..<g(u i)})
   apply (auto simp add: einterval-def)
   apply (erule (1) AB)
   apply (rule order-le-less-trans, rule A3, simp)
   apply (rule order-less-le-trans) prefer 2
   by (rule B3, simp)
 qed

 {
   fix i
   have (LBINT x=l i.. u i. g' x *R f (g x)) = (LBINT y=g (l i)..g (u i). f y)
     apply (rule interval-integral-substitution-finite, auto)
     apply (rule DERIV-subset, rule deriv-g, auto)
     apply (rule continuous-at-imp-continuous-on, auto, rule contf, auto)
     by (rule continuous-at-imp-continuous-on, auto, rule contg', auto)
    then have (LBINT x=l i.. u i. (f (g x) * g' x)) = (LBINT y=g (l i)..g (u
i). f y)
     by (simp add: ac-simps)
 } note eq1 = this
 have (λi. LBINT x=l i..u i. f (g x) * g' x)
    ⟶ (LBINT x=a..b. f (g x) * g' x)
  apply (rule interval-integral-Icc-approx-integrable [OF ‹a < b› approx])
  by (rule assms)
 hence 2: (λi. (LBINT y=g (l i)..g (u i). f y)) ⟶ (LBINT x=a..b. f (g x)
* g' x)
  by (simp add: eq1)
 have incseq: incseq (λi. {g (l i)<..<g (u i)})
  apply (auto simp add: incseq-def)
  apply (rule order-le-less-trans)
  prefer 2 apply assumption
  apply (rule g-nondec, auto)
  by (erule order-less-le-trans, rule g-nondec, auto)
 have img: ⋀x i. g (l i) ≤ x ⟹ x ≤ g (u i) ⟹ ∃ c ≥ l i. c ≤ u i ∧ x = g c
  apply (frule (1) IVT' [of g], auto)
  apply (rule continuous-at-imp-continuous-on, auto)
  by (rule DERIV-isCont, rule deriv-g, auto)
 have nonneg-f2: ⋀x i. g (l i) ≤ x ⟹ x ≤ g (u i) ⟹ 0 ≤ f x
  by (frule (1) img, auto, rule f-nonneg, auto)
 have contf-2: ⋀x i. g (l i) ≤ x ⟹ x ≤ g (u i) ⟹ isCont f x
  by (frule (1) img, auto, rule contf, auto)
 have integrable: set-integrable lborel (⋃ i. {g (l i)<..<g (u i)}) f
  apply (rule pos-integrable-to-top, auto simp del: real-scaleR-def)
```

apply (*rule incseq*)
apply (*rule nonneg-f2*, *erule less-imp-le*, *erule less-imp-le*)
apply (*rule set-integrable-subset*)
apply (*rule borel-integrable-atLeastAtMost'*)
apply (*rule continuous-at-imp-continuous-on*)
apply (*clarsimp*, *erule* (*1*) *contf-2*, *auto*)
apply (*erule less-imp-le*)+
using *2* **unfolding** *interval-lebesgue-integral-def*
by *auto*
thus *set-integrable lborel* (*einterval A B*) *f*
by (*simp add*: *un*)

have (*LBINT x=A..B. f x*) = (*LBINT x=a..b. g' x* $*_R$ *f* (*g x*))
proof (*rule interval-integral-substitution-integrable*)
show *set-integrable lborel* (*einterval a b*) (λ*x*. *g' x* $*_R$ *f* (*g x*))
using *integrable-fg* **by** (*simp add*: *ac-simps*)
qed *fact*+
then show (*LBINT x=A..B. f x*) = (*LBINT x=a..b.* (*f* (*g x*) ∗ *g' x*))
by (*simp add*: *ac-simps*)
qed

syntax
-complex-lebesgue-borel-integral :: *pttrn* ⇒ *real* ⇒ *complex*
((*2CLBINT* -. -) [*0*,*60*] *60*)

translations
CLBINT x. f == *CONST complex-lebesgue-integral CONST lborel* (λ*x*. *f*)

syntax
-complex-set-lebesgue-borel-integral :: *pttrn* ⇒ *real set* ⇒ *real* ⇒ *complex*
((*3CLBINT* -:-. -) [*0*,*60*,*61*] *60*)

translations
CLBINT x:*A. f* == *CONST complex-set-lebesgue-integral CONST lborel A* (λ*x*. *f*)

abbreviation *complex-interval-lebesgue-integral* ::
real measure ⇒ *ereal* ⇒ *ereal* ⇒ (*real* ⇒ *complex*) ⇒ *complex* **where**
complex-interval-lebesgue-integral M a b f ≡ *interval-lebesgue-integral M a b f*

abbreviation *complex-interval-lebesgue-integrable* ::
real measure ⇒ *ereal* ⇒ *ereal* ⇒ (*real* ⇒ *complex*) ⇒ *bool* **where**
complex-interval-lebesgue-integrable M a b f ≡ *interval-lebesgue-integrable M a b f*

syntax
-ascii-complex-interval-lebesgue-borel-integral :: *pttrn* ⇒ *ereal* ⇒ *ereal* ⇒ *real* ⇒ *complex*
((*4CLBINT* -=-..-. -) [*0*,*60*,*60*,*61*] *60*)

translations
CLBINT x=a..b. f == CONST complex-interval-lebesgue-integral CONST lborel a b (λx. f)

lemma *interval-integral-norm*:
fixes *f* :: *real* ⇒ *'a* :: {*banach*, *second-countable-topology*}
shows *interval-lebesgue-integrable lborel a b f* ⟹ *a* ≤ *b* ⟹
norm (*LBINT t=a..b. f t*) ≤ *LBINT t=a..b. norm* (*f t*)
using *integral-norm-bound*[*of lborel λx. indicator* (*einterval a b*) *x* ∗$_R$ *f x*]
by (*auto simp add*: *interval-lebesgue-integral-def interval-lebesgue-integrable-def*)

lemma *interval-integral-norm2*:
interval-lebesgue-integrable lborel a b f ⟹
norm (*LBINT t=a..b. f t*) ≤ |*LBINT t=a..b. norm* (*f t*)|
proof (*induct a b rule*: *linorder-wlog*)
case (*sym a b*) **then show** *?case*
by (*simp add*: *interval-integral-endpoints-reverse*[*of a b*] *interval-integrable-endpoints-reverse*[*of a b*])
next
case (*le a b*)
then have |*LBINT t=a..b. norm* (*f t*)| = *LBINT t=a..b. norm* (*f t*)
using *integrable-norm*[*of lborel λx. indicator* (*einterval a b*) *x* ∗$_R$ *f x*]
by (*auto simp add*: *interval-lebesgue-integral-def interval-lebesgue-integrable-def intro*!: *integral-nonneg-AE abs-of-nonneg*)
then show *?case*
using *le* **by** (*simp add*: *interval-integral-norm*)
qed

lemma *integral-cos*: *t* ≠ *0* ⟹ *LBINT x=a..b. cos* (*t* ∗ *x*) = *sin* (*t* ∗ *b*) / *t* − *sin* (*t* ∗ *a*) / *t*
apply (*intro interval-integral-FTC-finite continuous-intros*)
by (*auto intro*!: *derivative-eq-intros simp*: *has-field-derivative-iff-has-vector-derivative*[*symmetric*])

end

15 Integration by Substition

theory *Lebesgue-Integral-Substitution*
imports *Interval-Integral*
begin

lemma *nn-integral-substitution-aux*:
fixes *f* :: *real* ⇒ *ennreal*
assumes *Mf*: *f* ∈ *borel-measurable borel*
assumes *nonnegf*: ⋀*x*. *f x* ≥ *0*
assumes *derivg*: ⋀*x*. *x* ∈ {*a..b*} ⟹ (*g has-real-derivative g' x*) (*at x*)

```
  assumes contg': continuous-on {a..b} g'
  assumes derivg-nonneg: ⋀x. x ∈ {a..b} ⟹ g' x ≥ 0
  assumes a < b
  shows (∫⁺x. f x * indicator {g a..g b} x ∂lborel) =
          (∫⁺x. f (g x) * g' x * indicator {a..b} x ∂lborel)
proof−
  from ‹a < b› have [simp]: a ≤ b by simp
 from derivg have contg: continuous-on {a..b} g by (rule has-real-derivative-imp-continuous-on)
  from this and contg' have Mg: set-borel-measurable borel {a..b} g and
                            Mg': set-borel-measurable borel {a..b} g'
    by (simp-all only: set-measurable-continuous-on-ivl)
  from derivg have derivg': ⋀x. x ∈ {a..b} ⟹ (g has-vector-derivative g' x) (at
x)
    by (simp only: has-field-derivative-iff-has-vector-derivative)

  have real-ind[simp]: ⋀A x. enn2real (indicator A x) = indicator A x
    by (auto split: split-indicator)
  have ennreal-ind[simp]: ⋀A x. ennreal (indicator A x) = indicator A x
    by (auto split: split-indicator)
  have [simp]: ⋀x A. indicator A (g x) = indicator (g −‘ A) x
    by (auto split: split-indicator)

  from derivg derivg-nonneg have monog: ⋀x y. a ≤ x ⟹ x ≤ y ⟹ y ≤ b ⟹
g x ≤ g y
    by (rule deriv-nonneg-imp-mono) simp-all
  with monog have [simp]: g a ≤ g b by (auto intro: mono-onD)

  show ?thesis
  proof (induction rule: borel-measurable-induct[OF Mf, case-names cong set mult
add sup])
    case (cong f1 f2)
    from cong.hyps(3) have f1 = f2 by auto
    with cong show ?case by simp
  next
    case (set A)
    from set.hyps show ?case
    proof (induction rule: borel-set-induct)
      case empty
      thus ?case by simp
    next
      case (interval c d)
      {
        fix u v :: real assume asm: {u..v} ⊆ {g a..g b} u ≤ v

        obtain u' v' where u'v': {a..b} ∩ g−‘{u..v} = {u'..v'} u' ≤ v' g u' = u g
v' = v
            using asm by (rule-tac continuous-interval-vimage-Int[OF contg monog,
of u v]) simp-all
        hence {u'..v'} ⊆ {a..b} {u'..v'} ⊆ g −‘ {u..v} by blast+
```

with $u'v'(2)$ **have** $u' \in g -` \{u..v\}$ $v' \in g -` \{u..v\}$ **by** *auto*
from $u'v'(1)$ **have** [*simp*]: $\{a..b\} \cap g -` \{u..v\} \in$ *sets borel* **by** *simp*

have *A*: *continuous-on* $\{min\ u'\ v'..max\ u'\ v'\}$ g'
by (*simp only*: $u'v'$ *max-absorb2 min-absorb1*)
(*intro continuous-on-subset*[*OF contg'*], *insert* $u'v'$, *auto*)
have $\bigwedge x.\ x \in \{u'..v'\} \Longrightarrow$ (g *has-real-derivative* $g'\ x$) (*at* x *within* $\{u'..v'\}$)
using *asm* **by** (*intro has-field-derivative-subset*[*OF derivg*] *set-mp*[*OF* ‹$\{u'..v'\} \subseteq \{a..b\}$›]) *auto*
hence *B*: $\bigwedge x.\ min\ u'\ v' \le x \Longrightarrow x \le max\ u'\ v' \Longrightarrow$
(g *has-vector-derivative* $g'\ x$) (*at* x *within* $\{min\ u'\ v'..max\ u'\ v'\}$)
by (*simp only*: $u'v'$ *max-absorb2 min-absorb1*)
(*auto simp*: *has-field-derivative-iff-has-vector-derivative*)
have *integrable lborel* ($\lambda x.$ *indicator* ($\{a..b\} \cap g -` \{u..v\}$) $x *_R g'\ x$)
by (*rule set-integrable-subset*[*OF borel-integrable-atLeastAtMost'*[*OF contg'*]]) *simp-all*
hence ($\int^+ x.$ *ennreal* ($g'\ x$) $*$ *indicator* ($\{a..b\} \cap g -` \{u..v\}$) x *∂lborel*) =
LBINT x:$\{a..b\} \cap g -` \{u..v\}$. $g'\ x$
by (*subst nn-integral-eq-integral*[*symmetric*])
(*auto intro*!: *derivg-nonneg nn-integral-cong split*: *split-indicator*)
also from *interval-integral-FTC-finite*[*OF A B*]
have *LBINT* x:$\{a..b\} \cap g -` \{u..v\}$. $g'\ x = v - u$
by (*simp add*: $u'v'$ *interval-integral-Icc* ‹$u \le v$›)
finally have ($\int^+ x.$ *ennreal* ($g'\ x$) $*$ *indicator* ($\{a..b\} \cap g -` \{u..v\}$) x *∂lborel*) =
ennreal $(v - u)$ **.**
} **note** A = *this*

have ($\int^+ x.$ *indicator* $\{c..d\}$ $(g\ x)$ $*$ *ennreal* ($g'\ x$) $*$ *indicator* $\{a..b\}$ x *∂lborel*) =
($\int^+ x.$ *ennreal* ($g'\ x$) $*$ *indicator* ($\{a..b\} \cap g -` \{c..d\}$) x *∂lborel*)
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also have $\{a..b\} \cap g -` \{c..d\} = \{a..b\} \cap g -` \{max\ (g\ a)\ c..min\ (g\ b)\ d\}$
using ‹$a \le b$› ‹$c \le d$›
by (*auto intro*!: *monog intro*: *order.trans*)
also have ($\int^+ x.$ *ennreal* ($g'\ x$) $*$ *indicator* ... x *∂lborel*) =
(*if* $max\ (g\ a)\ c \le min\ (g\ b)\ d$ *then* $min\ (g\ b)\ d - max\ (g\ a)\ c$ *else* 0)
using ‹$c \le d$› **by** (*simp add*: A)
also have ... = ($\int^+ x.$ *indicator* ($\{g\ a..g\ b\} \cap \{c..d\}$) x *∂lborel*)
by (*subst nn-integral-indicator*) (*auto intro*!: *measurable-sets Mg simp*:)
also have ... = ($\int^+ x.$ *indicator* $\{c..d\}$ x $*$ *indicator* $\{g\ a..g\ b\}$ x *∂lborel*)
by (*intro nn-integral-cong*) (*auto split*: *split-indicator*)
finally show *?case* **..**

next

case (*compl A*)
note ‹$A \in$ *sets borel*›[*measurable*]
from *emeasure-mono*[*of* $A \cap \{g\ a..g\ b\}$ $\{g\ a..g\ b\}$ *lborel*]

have [*simp*]: *emeasure lborel* $(A \cap \{g\ a..g\ b\}) \neq top$ **by** (*auto simp*: *top-unique*)
have [*simp*]: $g -` A \cap \{a..b\} \in$ *sets borel*
by (*rule set-borel-measurable-sets*[*OF Mg*]) *auto*
have [*simp*]: $g -` (-A) \cap \{a..b\} \in$ *sets borel*
by (*rule set-borel-measurable-sets*[*OF Mg*]) *auto*

have $(\int^+x.$ *indicator* $(-A)\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) =$
$(\int^+x.$ *indicator* $(-A \cap \{g\ a..g\ b\})\ x\ \partial lborel)$
by (*rule nn-integral-cong*) (*simp split*: *split-indicator*)
also from *compl* **have** ... = *emeasure lborel* $(\{g\ a..g\ b\} - A)$ **using** *derivg-nonneg*
by (*simp add*: *vimage-Compl diff-eq Int-commute*[*of* $-A$])
also have $\{g\ a..g\ b\} - A = \{g\ a..g\ b\} - A \cap \{g\ a..g\ b\}$ **by** *blast*
also have *emeasure lborel* ... $= g\ b - g\ a -$ *emeasure lborel* $(A \cap \{g\ a..g\ b\})$
using ‹$A \in$ *sets borel*› **by** (*subst emeasure-Diff*) (*auto simp*:)
also have *emeasure lborel* $(A \cap \{g\ a..g\ b\}) =$
$\int^+x.$ *indicator* $A\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel$
using ‹$A \in$ *sets borel*›
by (*subst nn-integral-indicator*[*symmetric*], *simp*, *intro nn-integral-cong*)
(*simp split*: *split-indicator*)
also have ... $= \int^+ x.$ *indicator* $(g-`A \cap \{a..b\})\ x *$ *ennreal* $(g'\ x *$ *indicator* $\{a..b\}\ x)$ *∂lborel* (**is** - = *?I*)
by (*subst compl.IH*, *intro nn-integral-cong*) (*simp split*: *split-indicator*)
also have $g\ b - g\ a =$ *LBINT* x:$\{a..b\}.\ g'\ x$ **using** *derivg′*
by (*intro integral-FTC-atLeastAtMost*[*symmetric*])
(*auto intro*: *continuous-on-subset*[*OF contg′*] *has-field-derivative-subset*[*OF derivg*]
has-vector-derivative-at-within)
also have *ennreal* ... $= \int^+ x.\ g'\ x *$ *indicator* $\{a..b\}\ x\ \partial lborel$
using *borel-integrable-atLeastAtMost′*[*OF contg′*]
by (*subst nn-integral-eq-integral*)
(*simp-all add*: *mult.commute derivg-nonneg split*: *split-indicator*)
also have *Mg″*: $(\lambda x.$ *indicator* $(g -` A \cap \{a..b\})\ x *$ *ennreal* $(g'\ x *$ *indicator* $\{a..b\}\ x))$
$\in$ *borel-measurable borel* **using** *Mg′*
by (*intro borel-measurable-times-ennreal borel-measurable-indicator*)
(*simp-all add*: *mult.commute*)
have *le*: $(\int^+x.$ *indicator* $(g-`A \cap \{a..b\})\ x *$ *ennreal* $(g'\ x *$ *indicator* $\{a..b\}\ x)\ \partial lborel) \leq$
$(\int^+x.$ *ennreal* $(g'\ x) *$ *indicator* $\{a..b\}\ x\ \partial lborel)$
by (*intro nn-integral-mono*) (*simp split*: *split-indicator add*: *derivg-nonneg*)
note *integrable* = *borel-integrable-atLeastAtMost′*[*OF contg′*]
with *le* **have** *notinf*: $(\int^+x.$ *indicator* $(g-`A \cap \{a..b\})\ x *$ *ennreal* $(g'\ x *$ *indicator* $\{a..b\}\ x)\ \partial lborel) \neq top$
by (*auto simp*: *real-integrable-def nn-integral-set-ennreal mult.commute top-unique*)
have $(\int^+ x.\ g'\ x *$ *indicator* $\{a..b\}\ x\ \partial lborel) -$ *?I* =
$\int^+ x.$ *ennreal* $(g'\ x *$ *indicator* $\{a..b\}\ x) -$
indicator $(g -` A \cap \{a..b\})\ x *$ *ennreal* $(g'\ x *$ *indicator* $\{a..b\}$

```
x) ∂lborel
      apply (intro nn-integral-diff[symmetric])
      apply (insert Mg', simp add: mult.commute) []
      apply (insert Mg'', simp) []
      apply (simp split: split-indicator add: derivg-nonneg)
      apply (rule notinf)
      apply (simp split: split-indicator add: derivg-nonneg)
      done
    also have ... = ∫⁺ x. indicator (−A) (g x) * ennreal (g' x) * indicator {a..b}
x ∂lborel
      by (intro nn-integral-cong) (simp split: split-indicator)
    finally show ?case .

  next
    case (union f)
    then have [simp]: ⋀i. {a..b} ∩ g −` f i ∈ sets borel
      by (subst Int-commute, intro set-borel-measurable-sets[OF Mg]) auto
    have g −` (⋃ i. f i) ∩ {a..b} = (⋃ i. {a..b} ∩ g −` f i) by auto
    hence g −` (⋃ i. f i) ∩ {a..b} ∈ sets borel by (auto simp del: UN-simps)

    have (∫⁺x. indicator (⋃ i. f i) x * indicator {g a..g b} x ∂lborel) =
            ∫⁺x. indicator (⋃ i. {g a..g b} ∩ f i) x ∂lborel
      by (intro nn-integral-cong) (simp split: split-indicator)
    also from union have ... = emeasure lborel (⋃ i. {g a..g b} ∩ f i) by simp
    also from union have ... = (∑ i. emeasure lborel ({g a..g b} ∩ f i))
      by (intro suminf-emeasure[symmetric]) (auto simp: disjoint-family-on-def)
    also from union have ... = (∑ i. ∫⁺x. indicator ({g a..g b} ∩ f i) x ∂lborel)
by simp
    also have (λi. ∫⁺x. indicator ({g a..g b} ∩ f i) x ∂lborel) =
                        (λi. ∫⁺x. indicator (f i) x * indicator {g a..g b} x ∂lborel)
      by (intro ext nn-integral-cong) (simp split: split-indicator)
    also from union.IH have (∑ i. ∫⁺x. indicator (f i) x * indicator {g a..g b}
x ∂lborel) =
          (∑ i. ∫⁺ x. indicator (f i) (g x) * ennreal (g' x) * indicator {a..b} x
∂lborel) by simp
    also have (λi. ∫⁺ x. indicator (f i) (g x) * ennreal (g' x) * indicator {a..b}
x ∂lborel) =
                          (λi. ∫⁺ x. ennreal (g' x * indicator {a..b} x) * indicator
({a..b} ∩ g −` f i) x ∂lborel)
      by (intro ext nn-integral-cong) (simp split: split-indicator)
    also have (∑ i. ... i) = ∫⁺ x. (∑ i. ennreal (g' x * indicator {a..b} x) *
indicator ({a..b} ∩ g −` f i) x) ∂lborel
      using Mg'
      apply (intro nn-integral-suminf[symmetric])
      apply (rule borel-measurable-times-ennreal, simp add: mult.commute)
      apply (rule borel-measurable-indicator, subst sets-lborel)
      apply (simp-all split: split-indicator add: derivg-nonneg)
      done
    also have (λx i. ennreal (g' x * indicator {a..b} x) * indicator ({a..b} ∩ g
```

```
−‘ f i) x) =
                (λx i. ennreal (g′ x * indicator {a..b} x) * indicator (g −‘ f i) x)
      by (intro ext) (simp split: split-indicator)
    also have (∫⁺ x. (∑ i. ennreal (g′ x * indicator {a..b} x) * indicator (g −‘
f i) x) ∂lborel) =
                ∫⁺ x. ennreal (g′ x * indicator {a..b} x) * (∑ i. indicator (g −‘
f i) x) ∂lborel
      by (intro nn-integral-cong) (auto split: split-indicator simp: derivg-nonneg)
    also from union have (λx. ∑ i. indicator (g −‘ f i) x :: ennreal) = (λx.
indicator (⋃ i. g −‘ f i) x)
      by (intro ext suminf-indicator) (auto simp: disjoint-family-on-def)
    also have (∫⁺x. ennreal (g′ x * indicator {a..b} x) * ... x ∂lborel) =
                (∫⁺x. indicator (⋃ i. f i) (g x) * ennreal (g′ x) * indicator {a..b}
x ∂lborel)
      by (intro nn-integral-cong) (simp split: split-indicator)
    finally show ?case .
  qed

next
  case (mult f c)
    note Mf[measurable] = ‹f ∈ borel-measurable borel›
    let ?I = indicator {a..b}
   have (λx. f (g x * ?I x) * ennreal (g′ x * ?I x)) ∈ borel-measurable borel using
Mg Mg′
      by (intro borel-measurable-times-ennreal measurable-compose[OF - Mf])
         (simp-all add: mult.commute)
    also have (λx. f (g x * ?I x) * ennreal (g′ x * ?I x)) = (λx. f (g x) * ennreal
(g′ x) * ?I x)
      by (intro ext) (simp split: split-indicator)
    finally have Mf′: (λx. f (g x) * ennreal (g′ x) * ?I x) ∈ borel-measurable borel
.
    with mult show ?case
      by (subst (1 2 3) mult-ac, subst (1 2) nn-integral-cmult) (simp-all add:
mult-ac)

next
  case (add f2 f1)
    let ?I = indicator {a..b}
    {
      fix f :: real ⇒ ennreal assume Mf: f ∈ borel-measurable borel
      have (λx. f (g x * ?I x) * ennreal (g′ x * ?I x)) ∈ borel-measurable borel
using Mg Mg′
        by (intro borel-measurable-times-ennreal measurable-compose[OF - Mf])
           (simp-all add: mult.commute)
      also have (λx. f (g x * ?I x) * ennreal (g′ x * ?I x)) = (λx. f (g x) * ennreal
(g′ x) * ?I x)
        by (intro ext) (simp split: split-indicator)
      finally have (λx. f (g x) * ennreal (g′ x) * ?I x) ∈ borel-measurable borel .
    } note Mf′ = this[OF ‹f1 ∈ borel-measurable borel›] this[OF ‹f2 ∈ borel-measurable
```

borel›]

have $(\int^+ x.\ (f1\ x + f2\ x) * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) =$
$(\int^+ x.\ f1\ x * indicator\ \{g\ a..g\ b\}\ x + f2\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel)$
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also from *add* **have** ... $= (\int^+ x.\ f1\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel) +$
$(\int^+ x.\ f2\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel)$
by (*simp-all add*: *nn-integral-add*)
also from *add* **have** ... $= (\int^+ x.\ f1\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x +$
$f2\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel)$
by (*intro nn-integral-add*[*symmetric*])
(*auto simp add*: *Mf' derivg-nonneg split*: *split-indicator*)
also have ... $= \int^+ x.\ (f1\ (g\ x) + f2\ (g\ x)) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel$
by (*intro nn-integral-cong*) (*simp split*: *split-indicator add*: *distrib-right*)
finally show *?case* **.**

next
case (*sup F*)
{
fix *i*
let *?I* = *indicator* $\{a..b\}$
have $(\lambda x.\ F\ i\ (g\ x * ?I\ x) * ennreal\ (g'\ x * ?I\ x)) \in$ *borel-measurable borel* **using** *Mg Mg'*
by (*rule-tac borel-measurable-times-ennreal*, *rule-tac measurable-compose*[*OF - sup.hyps*(*1*)])
(*simp-all add*: *mult.commute*)
also have $(\lambda x.\ F\ i\ (g\ x * ?I\ x) * ennreal\ (g'\ x * ?I\ x)) = (\lambda x.\ F\ i\ (g\ x) * ennreal\ (g'\ x) * ?I\ x)$
by (*intro ext*) (*simp split*: *split-indicator*)
finally have ... $\in$ *borel-measurable borel* **.**
} note *Mf' = this*

have $(\int^+ x.\ (SUP\ i.\ F\ i\ x) * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) =$
$\int^+ x.\ (SUP\ i.\ F\ i\ x * indicator\ \{g\ a..g\ b\}\ x)\ \partial lborel$
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also from *sup* **have** ... $= (SUP\ i.\ \int^+ x.\ F\ i\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel)$
by (*intro nn-integral-monotone-convergence-SUP*)
(*auto simp*: *incseq-def le-fun-def split*: *split-indicator*)
also from *sup* **have** ... $= (SUP\ i.\ \int^+ x.\ F\ i\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel)$
by *simp*
also from *sup* **have** ... $= \int^+ x.\ (SUP\ i.\ F\ i\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x)\ \partial lborel$
by (*intro nn-integral-monotone-convergence-SUP*[*symmetric*])

```
      (auto simp: incseq-def le-fun-def derivg-nonneg Mf' split: split-indicator
          intro!: mult-right-mono)
  also from sup have ... = ∫⁺x. (SUP i. F i (g x)) * ennreal (g' x) * indicator
{a..b} x ∂lborel
    by (subst mult.assoc, subst mult.commute, subst SUP-mult-left-ennreal)
      (auto split: split-indicator simp: derivg-nonneg mult-ac)
  finally show ?case by simp
 qed
qed

lemma nn-integral-substitution:
 fixes f :: real ⇒ real
 assumes Mf[measurable]: set-borel-measurable borel {g a..g b} f
 assumes derivg: ⋀x. x ∈ {a..b} ⟹ (g has-real-derivative g' x) (at x)
 assumes contg': continuous-on {a..b} g'
 assumes derivg-nonneg: ⋀x. x ∈ {a..b} ⟹ g' x ≥ 0
 assumes a ≤ b
 shows (∫⁺x. f x * indicator {g a..g b} x ∂lborel) =
         (∫⁺x. f (g x) * g' x * indicator {a..b} x ∂lborel)
proof (cases a = b)
 assume a ≠ b
 with ‹a ≤ b› have a < b by auto
 let ?f' = λx. f x * indicator {g a..g b} x

 from derivg derivg-nonneg have monog: ⋀x y. a ≤ x ⟹ x ≤ y ⟹ y ≤ b ⟹
g x ≤ g y
   by (rule deriv-nonneg-imp-mono) simp-all
 have bounds: ⋀x. x ≥ a ⟹ x ≤ b ⟹ g x ≥ g a ⋀x. x ≥ a ⟹ x ≤ b ⟹ g
x ≤ g b
   by (auto intro: monog)

 from derivg-nonneg have nonneg:
   ⋀f x. x ≥ a ⟹ x ≤ b ⟹ g' x ≠ 0 ⟹ f x * ennreal (g' x) ≥ 0 ⟹ f x ≥ 0
   by (force simp: field-simps)
 have nonneg': ⋀x. a ≤ x ⟹ x ≤ b ⟹ ¬ 0 ≤ f (g x) ⟹ 0 ≤ f (g x) * g' x
⟹ g' x = 0
   by (metis atLeastAtMost-iff derivg-nonneg eq-iff mult-eq-0-iff mult-le-0-iff)

 have (∫⁺x. f x * indicator {g a..g b} x ∂lborel) =
         (∫⁺x. ennreal (?f' x) * indicator {g a..g b} x ∂lborel)
   by (intro nn-integral-cong)
     (auto split: split-indicator split-max simp: zero-ennreal.rep-eq ennreal-neg)
 also have ... = ∫⁺ x. ?f' (g x) * ennreal (g' x) * indicator {a..b} x ∂lborel
using Mf
   by (subst nn-integral-substitution-aux[OF - - derivg contg' derivg-nonneg ‹a <
b›])
     (auto simp add: mult.commute)
 also have ... = ∫⁺ x. f (g x) * ennreal (g' x) * indicator {a..b} x ∂lborel
    by (intro nn-integral-cong) (auto split: split-indicator simp: max-def dest:
```

bounds)
also have ... = $\int^+ x.$ *ennreal* (*f* (*g x*) * *g'* *x* * *indicator* {*a..b*} *x*) *∂lborel*
by (*intro nn-integral-cong*) (*auto simp*: *mult.commute derivg-nonneg ennreal-mult'* *split*: *split-indicator*)
finally show *?thesis* **.**
qed *auto*

lemma *integral-substitution*:
assumes *integrable*: *set-integrable lborel* {*g a..g b*} *f*
assumes *derivg*: $\bigwedge x.$ *x* ∈ {*a..b*} ⟹ (*g has-real-derivative g' x*) (*at x*)
assumes *contg'*: *continuous-on* {*a..b*} *g'*
assumes *derivg-nonneg*: $\bigwedge x.$ *x* ∈ {*a..b*} ⟹ *g' x* ≥ *0*
assumes *a* ≤ *b*
shows *set-integrable lborel* {*a..b*} (λ*x*. *f* (*g x*) * *g' x*)
and (*LBINT x*. *f x* * *indicator* {*g a..g b*} *x*) = (*LBINT x*. *f* (*g x*) * *g' x* * *indicator* {*a..b*} *x*)
proof−
from *derivg* **have** *contg*: *continuous-on* {*a..b*} *g* **by** (*rule has-real-derivative-imp-continuous-on*)
from *this* **and** *contg'* **have** *Mg*: *set-borel-measurable borel* {*a..b*} *g* **and**
Mg': *set-borel-measurable borel* {*a..b*} *g'*
by (*simp-all only*: *set-measurable-continuous-on-ivl*)
from *derivg derivg-nonneg* **have** *monog*: $\bigwedge x\ y.$ *a* ≤ *x* ⟹ *x* ≤ *y* ⟹ *y* ≤ *b* ⟹ *g x* ≤ *g y*
by (*rule deriv-nonneg-imp-mono*) *simp-all*

have (λ*x*. *ennreal* (*f x*) * *indicator* {*g a..g b*} *x*) =
(λ*x*. *ennreal* (*f x* * *indicator* {*g a..g b*} *x*))
by (*intro ext*) (*simp split*: *split-indicator*)
with *integrable* **have** *M1*: (λ*x*. *f x* * *indicator* {*g a..g b*} *x*) ∈ *borel-measurable borel*
unfolding *real-integrable-def* **by** (*force simp*: *mult.commute*)
from *integrable* **have** *M2*: (λ*x*. −*f x* * *indicator* {*g a..g b*} *x*) ∈ *borel-measurable borel*
unfolding *real-integrable-def* **by** (*force simp*: *mult.commute*)

have *LBINT x*. (*f x* :: *real*) * *indicator* {*g a..g b*} *x* =
enn2real ($\int^+$ *x*. *ennreal* (*f x*) * *indicator* {*g a..g b*} *x ∂lborel*) −
enn2real ($\int^+$ *x*. *ennreal* (− (*f x*)) * *indicator* {*g a..g b*} *x ∂lborel*) **using** *integrable*
by (*subst real-lebesgue-integral-def*) (*simp-all add*: *nn-integral-set-ennreal mult.commute*)
also have ($\int^+ x.$ *ennreal* (*f x*) * *indicator* {*g a..g b*} *x ∂lborel*) =
($\int^+ x.$ *ennreal* (*f x* * *indicator* {*g a..g b*} *x*) *∂lborel*)
by (*intro nn-integral-cong*) (*simp split*: *split-indicator*)
also with *M1* **have** *A*: ($\int^+$ *x*. *ennreal* (*f x* * *indicator* {*g a..g b*} *x*) *∂lborel*) =
($\int^+$ *x*. *ennreal* (*f* (*g x*) * *g' x* * *indicator* {*a..b*} *x*) *∂lborel*)
by (*subst nn-integral-substitution*[*OF - derivg contg' derivg-nonneg* ‹*a* ≤ *b*›])
(*auto simp*: *nn-integral-set-ennreal mult.commute*)
also have ($\int^+$ *x*. *ennreal* (− (*f x*)) * *indicator* {*g a..g b*} *x ∂lborel*) =
($\int^+$ *x*. *ennreal* (− (*f x*) * *indicator* {*g a..g b*} *x*) *∂lborel*)

```
    by (intro nn-integral-cong) (simp split: split-indicator)
  also with M2 have B: (∫⁺ x. ennreal (− (f x) ∗ indicator {g a..g b} x) ∂lborel)
=
                                (∫⁺ x. ennreal (− (f (g x)) ∗ g' x ∗ indicator {a..b} x)
∂lborel)
    by (subst nn-integral-substitution[OF - derivg contg' derivg-nonneg ‹a ≤ b›])
       (auto simp: nn-integral-set-ennreal mult.commute)

  also {
    from integrable have Mf: set-borel-measurable borel {g a..g b} f
      unfolding real-integrable-def by simp
    from borel-measurable-times[OF measurable-compose[OF Mg Mf] Mg']
     have (λx. f (g x ∗ indicator {a..b} x) ∗ indicator {g a..g b} (g x ∗ indicator
{a..b} x) ∗
                    (g' x ∗ indicator {a..b} x)) ∈ borel-measurable borel  (is ?f ∈ -)
      by (simp add: mult.commute)
    also have ?f = (λx. f (g x) ∗ g' x ∗ indicator {a..b} x)
      using monog by (intro ext) (auto split: split-indicator)
    finally show set-integrable lborel {a..b} (λx. f (g x) ∗ g' x)
      using A B integrable unfolding real-integrable-def
      by (simp-all add: nn-integral-set-ennreal mult.commute)
  } note integrable' = this

  have enn2real (∫⁺ x. ennreal (f (g x) ∗ g' x ∗ indicator {a..b} x) ∂lborel) −
                    enn2real (∫⁺ x. ennreal (−f (g x) ∗ g' x ∗ indicator {a..b} x)
∂lborel) =
              (LBINT x. f (g x) ∗ g' x ∗ indicator {a..b} x) using integrable'
    by (subst real-lebesgue-integral-def) (simp-all add: field-simps)
  finally show (LBINT x. f x ∗ indicator {g a..g b} x) =
                    (LBINT x. f (g x) ∗ g' x ∗ indicator {a..b} x) .
qed

lemma interval-integral-substitution:
  assumes integrable: set-integrable lborel {g a..g b} f
  assumes derivg: ⋀x. x ∈ {a..b} ⟹ (g has-real-derivative g' x) (at x)
  assumes contg': continuous-on {a..b} g'
  assumes derivg-nonneg: ⋀x. x ∈ {a..b} ⟹ g' x ≥ 0
  assumes a ≤ b
  shows set-integrable lborel {a..b} (λx. f (g x) ∗ g' x)
    and (LBINT x=g a..g b. f x) = (LBINT x=a..b. f (g x) ∗ g' x)
  apply (rule integral-substitution[OF assms], simp, simp)
  apply (subst (1 2) interval-integral-Icc, fact)
  apply (rule deriv-nonneg-imp-mono[OF derivg derivg-nonneg], simp, simp, fact)
  using integral-substitution(2)[OF assms]
  apply (simp add: mult.commute)
  done

lemma set-borel-integrable-singleton[simp]:
  set-integrable lborel {x} (f :: real ⇒ real)
```

by (*subst integrable-discrete-difference*[**where** X={x} **and** g=λ-. *0*]) *auto*

end

16 Adhoc overloading of constants based on their types

theory *Adhoc-Overloading*
imports *Pure*
keywords *adhoc-overloading* :: *thy-decl* **and** *no-adhoc-overloading* :: *thy-decl*
begin

ML-file *adhoc-overloading.ML*

end

17 Monad notation for arbitrary types

theory *Monad-Syntax*
imports *Main* ~~/*src*/*Tools*/*Adhoc-Overloading*
begin

We provide a convenient do-notation for monadic expressions well-known from Haskell. *Let* is printed specially in do-expressions.

consts
bind :: ['*a*, '*b* ⇒ '*c*] ⇒ '*d* (**infixr** ⤜ *54*)

notation (*ASCII*)
bind (**infixr** >>= *54*)

abbreviation (*do-notation*)
bind-do :: ['*a*, '*b* ⇒ '*c*] ⇒ '*d*
where *bind-do* ≡ *bind*

notation (**output**)
bind-do (**infixr** ⤜ *54*)

notation (*ASCII* **output**)
bind-do (**infixr** >>= *54*)

nonterminal *do-binds* **and** *do-bind*
syntax
-do-block :: *do-binds* ⇒ '*a* (*do* {//(*2* -)//} [*12*] *62*)
-do-bind :: [*pttrn*, '*a*] ⇒ *do-bind* ((*2*- ←/ -) *13*)
-do-let :: [*pttrn*, '*a*] ⇒ *do-bind* ((*2let* - =/ -) [*1000*, *13*] *13*)

-do-then :: *'a* ⇒ *do-bind* (- [*14*] *13*)
-do-final :: *'a* ⇒ *do-binds* (-)
-do-cons :: [*do-bind*, *do-binds*] ⇒ *do-binds* (-;//- [*13*, *12*] *12*)
-thenM :: [*'a*, *'b*] ⇒ *'c* (**infixr** ≫ *54*)

syntax (*ASCII*)
-do-bind :: [*pttrn*, *'a*] ⇒ *do-bind* ((*2*- <−/ -) *13*)
-thenM :: [*'a*, *'b*] ⇒ *'c* (**infixr** >> *54*)

translations
-do-block (*-do-cons* (*-do-then t*) (*-do-final e*))
⇌ *CONST bind-do t* (λ-. *e*)
-do-block (*-do-cons* (*-do-bind p t*) (*-do-final e*))
⇌ *CONST bind-do t* (λ*p*. *e*)
-do-block (*-do-cons* (*-do-let p t*) *bs*)
⇌ *let p* = *t in -do-block bs*
-do-block (*-do-cons b* (*-do-cons c cs*))
⇌ *-do-block* (*-do-cons b* (*-do-final* (*-do-block* (*-do-cons c cs*))))
-do-cons (*-do-let p t*) (*-do-final s*)
⇌ *-do-final* (*let p* = *t in s*)
-do-block (*-do-final e*) ⇀ *e*
(*m* ≫ *n*) ⇀ (*m* ≫= (λ-. *n*))

adhoc-overloading
bind Set.bind Predicate.bind Option.bind List.bind

end

theory *Giry-Monad*
imports *Probability-Measure Lebesgue-Integral-Substitution* ~~/*src*/*HOL*/*Library*/*Monad-Syntax*
begin

18 Sub-probability spaces

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

lemma *subprob-spaceI*[*Pure.intro*!]:
assumes ∗: *emeasure M* (*space M*) ≤ *1*
assumes *space M* ≠ {}
shows *subprob-space M*
proof −
interpret *finite-measure M*
proof
show *emeasure M* (*space M*) ≠ ∞ **using** ∗ **by** (*auto simp*: *top-unique*)
qed
show *subprob-space M* **by** *standard fact*+

qed

lemma *prob-space-imp-subprob-space*:
prob-space M $\Longrightarrow$ *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* $\Longrightarrow$ *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*]: $\Omega \neq \{\}$ $\Longrightarrow$ *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* $\neq \{\}$ $\Longrightarrow$ *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*)) $\leq$ *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* $\leq$ *1*
by (*rule order.trans*[*OF emeasure-space emeasure-space-le-1*])

lemma (**in** *subprob-space*) *subprob-measure-le-1*: *measure M X* $\leq$ *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* ∂M) $\leq$ *c*
proof $-$
have ($\int^+$ *x*. *f x* ∂M) $\leq$ ($\int^+$ *x*. *c* ∂M)
by(*rule nn-integral-mono-AE*) *fact*
also have ... $\leq$ *c* $*$ *emeasure M* (*space M*)
using ‹*0* $\leq$ *c*› **by** *simp*
also have ... $\leq$ *c* $*$ *1* **using** *emeasure-space-le-1* ‹*0* $\leq$ *c*› **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*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g$ *has-real-derivative* $g'\ x)$ (*at* x)
assumes *contg'*: *continuous-on* $\{a..b\}\ g'$
assumes *mono*: *strict-mono-on* $g\ \{a..b\}$ **and** *inv*: $\bigwedge x.\ h\ x \in \{a..b\} \Longrightarrow g\ (h\ x) = x$
assumes *range*: $\{a..b\} \subseteq$ *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 g* $\{a..b\}$ **by** (*rule strict-mono-on-imp-mono-on*)
from *mono' derivg* **have** $\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow g'\ x \geq 0$
by (*rule mono-on-imp-deriv-nonneg*) *auto*
from *contg' this* **have** *derivg-nonneg*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g'\ x \geq 0$
by (*rule continuous-ge-on-Ioo*) (*simp-all add*: ‹$a < b$›)

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: $y \in \{g\ a..g\ b\}$
with *IVT'*[*OF - - - contg, of y*] **obtain** x **where** $x \in \{a..b\}\ y = g\ x$ **by** *auto*
with *range* **and** *inv* **show** $y \in h -` \{a..b\}$ **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*) $\{a..b\} =$
$\int^+ x.\ f\ x * indicator\ (h -` \{a..b\})\ x\ \partial lborel$
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*) $\{a..b\} \leq 1$
by (*rule subprob-space.subprob-emeasure-le-1*) (*rule prob'*)
hence *emeasure* (*distr* (*density lborel f*) *lborel h*) $\{a..b\} \neq \infty$ **by** (*auto simp*: *top-unique*)
with *assms* **have** $(\int^+ x.\ f\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) =$
$(\int^+ x.\ f\ (g\ x) * g'\ x * indicator\ \{a..b\}\ x\ \partial lborel)$

by (*intro nn-integral-substitution-aux*)
(*auto simp*: *derivg-nonneg A B emeasure-density mult.commute* ‹$a < b$›)
also have ... = *emeasure* (*density lborel* ($\lambda x.\ f\ (g\ x) * g'\ x$)) {$a..b$}
by (*simp add*: *emeasure-density*)
finally show *?thesis* **.**
next
assume $\neg a < b$
with ‹$a \leq b$› **have** [*simp*]: $b = a$ **by** (*simp add*: *not-less del*: ‹$a \leq b$›)
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* $\bigotimes_M$ *M2*
proof
from *mult-le-one*[*OF M1.emeasure-space-le-1 - M2.emeasure-space-le-1*]
show *emeasure* (*M1* $\bigotimes_M$ *M2*) (*space* (*M1* $\bigotimes_M$ *M2*)) $\leq$ *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* $\bigotimes_M$ *M2*) $\neq$ {}
by (*simp add*: *space-pair-measure*)
qed

lemma *subprob-space-null-measure-iff*:
subprob-space (*null-measure M*) $\longleftrightarrow$ *space M* $\neq$ {}
by (*auto intro*!: *subprob-spaceI dest*: *subprob-space.subprob-not-empty*)

lemma *subprob-space-restrict-space*:
assumes *M*: *subprob-space M*
and *A*: *A* ∩ *space M* ∈ *sets M A* ∩ *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* ∩ *space M*)
using *A* **by**(*simp add*: *emeasure-restrict-space space-restrict-space*)
also have ... $\leq$ *1* **by**(*rule subprob-space.subprob-emeasure-le-1*)(*rule M*)
finally show *emeasure* (*restrict-space M A*) (*space* (*restrict-space M A*)) $\leq$ *1* **.**
next
show *space* (*restrict-space M A*) $\neq$ {}
using *A* **by**(*simp add*: *space-restrict-space*)
qed

definition *subprob-algebra* :: *'a measure* ⇒ *'a measure measure* **where**
subprob-algebra K =
(⨆$_\sigma$ *A*∈*sets K*. *vimage-algebra* {*M*. *subprob-space M* ∧ *sets M* = *sets K*} (λ*M*.

emeasure M A) *borel*)

lemma *space-subprob-algebra*: *space* (*subprob-algebra A*) = {*M*. *subprob-space M* ∧ *sets M* = *sets A*}
by (*auto simp add*: *subprob-algebra-def space-Sup-sigma*)

lemma *subprob-algebra-cong*: *sets M* = *sets N* ⟹ *subprob-algebra M* = *subprob-algebra N*
by (*simp add*: *subprob-algebra-def*)

lemma *measurable-emeasure-subprob-algebra*[*measurable*]:
a ∈ *sets A* ⟹ (λ*M*. *emeasure M a*) ∈ *borel-measurable* (*subprob-algebra A*)
by (*auto intro*!: *measurable-Sup-sigma1 measurable-vimage-algebra1 simp*: *subprob-algebra-def*)

lemma *measurable-measure-subprob-algebra*[*measurable*]:
a ∈ *sets A* ⟹ (λ*M*. *measure M a*) ∈ *borel-measurable* (*subprob-algebra A*)
unfolding *measure-def* **by** *measurable*

lemma *subprob-measurableD*:
assumes *N*: *N* ∈ *measurable M* (*subprob-algebra S*) **and** *x*: *x* ∈ *space M*
shows *space* (*N x*) = *space S*
and *sets* (*N x*) = *sets S*
and *measurable* (*N x*) *K* = *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 ‹

```
fun subprob-cong thm ctxt = (
  let
    val thm' = Thm.transfer (Proof-Context.theory-of 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: $K \in \mathit{measurable}\ M\ (\mathit{subprob\text{-}algebra}\ N)$
begin

lemma *subprob-space-kernel*: $a \in \mathit{space}\ M \Longrightarrow \mathit{subprob\text{-}space}\ (K\ a)$
using *measurable-space*[*OF* K] **by** (*simp add*: *space-subprob-algebra*)

lemma *sets-kernel*: $a \in \mathit{space}\ M \Longrightarrow \mathit{sets}\ (K\ a) = \mathit{sets}\ N$
using *measurable-space*[*OF* K] **by** (*simp add*: *space-subprob-algebra*)

lemma *measurable-emeasure-kernel*[*measurable*]:
$A \in \mathit{sets}\ N \Longrightarrow (\lambda a.\ \mathit{emeasure}\ (K\ a)\ A) \in \mathit{borel\text{-}measurable}\ M$
using *measurable-compose*[*OF* K *measurable-emeasure-subprob-algebra*] .

end

lemma *measurable-subprob-algebra*:
$(\bigwedge a.\ a \in \mathit{space}\ M \Longrightarrow \mathit{subprob\text{-}space}\ (K\ a)) \Longrightarrow$
$(\bigwedge a.\ a \in \mathit{space}\ M \Longrightarrow \mathit{sets}\ (K\ a) = \mathit{sets}\ N) \Longrightarrow$
$(\bigwedge A.\ A \in \mathit{sets}\ N \Longrightarrow (\lambda a.\ \mathit{emeasure}\ (K\ a)\ A) \in \mathit{borel\text{-}measurable}\ M) \Longrightarrow$
$K \in \mathit{measurable}\ M\ (\mathit{subprob\text{-}algebra}\ N)$
by (*auto intro*!: *measurable-Sup-sigma2 measurable-vimage-algebra2 simp*: *subprob-algebra-def*)

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

lemma *space-subprob-algebra-empty-iff*:
$\mathit{space}\ (\mathit{subprob\text{-}algebra}\ N) = \{\} \longleftrightarrow \mathit{space}\ N = \{\}$
proof
have $\bigwedge x.\ x \in \mathit{space}\ N \Longrightarrow \mathit{density}\ N\ (\lambda\text{-}.\ 0) \in \mathit{space}\ (\mathit{subprob\text{-}algebra}\ N)$
by (*auto simp*: *space-subprob-algebra emeasure-density intro*!: *subprob-spaceI*)
then show $\mathit{space}\ (\mathit{subprob\text{-}algebra}\ N) = \{\} \Longrightarrow \mathit{space}\ N = \{\}$
by *auto*
next
assume $\mathit{space}\ N = \{\}$
hence $\mathit{sets}\ N = \{\{\}\}$ **by** (*simp add*: *space-empty-iff*)
moreover have $\bigwedge M.\ \mathit{subprob\text{-}space}\ M \Longrightarrow \mathit{sets}\ M \neq \{\{\}\}$

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* ∈ *borel-measurable N*
shows (λ*M*. *integral*^{N} *M f*) ∈ *borel-measurable* (*subprob-algebra N*) (**is** - ∈ *?B*)
using *f*
proof *induct*
case (*cong f g*)
moreover have (λ*M'*. ∫$^+$*M''*. *f M'' ∂M'*) ∈ *?B* ⟷ (λ*M'*. ∫$^+$*M''*. *g M'' ∂M'*) ∈ *?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*)
moreover then have (λ*M'*. ∫$^+$*M''*. *indicator B M'' ∂M'*) ∈ *?B* ⟷ (λ*M'*. *emeasure M' B*) ∈ *?B*
by (*intro measurable-cong nn-integral-indicator*) (*simp add*: *space-subprob-algebra*)
ultimately show *?case*
by (*simp add*: *measurable-emeasure-subprob-algebra*)
next
case (*mult f c*)
moreover then have (λ*M'*. ∫$^+$*M''*. *c* ∗ *f M'' ∂M'*) ∈ *?B* ⟷ (λ*M'*. *c* ∗ ∫$^+$*M''*. *f M'' ∂M'*) ∈ *?B*
by (*intro measurable-cong nn-integral-cmult*) (*auto simp add*: *space-subprob-algebra*)
ultimately show *?case*
by *simp*
next
case (*add f g*)
moreover then have (λ*M'*. ∫$^+$*M''*. *f M'' + g M'' ∂M'*) ∈ *?B* ⟷ (λ*M'*. (∫$^+$*M''*. *f M'' ∂M'*) + (∫$^+$*M''*. *g M'' ∂M'*)) ∈ *?B*
by (*intro measurable-cong nn-integral-add*) (*auto simp add*: *space-subprob-algebra*)
ultimately show *?case*
by (*simp add*: *ac-simps*)
next
case (*seq F*)
moreover then have (λ*M'*. ∫$^+$*M''*. (*SUP i*. *F i*) *M'' ∂M'*) ∈ *?B* ⟷ (λ*M'*. *SUP i*. (∫$^+$*M''*. *F i M'' ∂M'*)) ∈ *?B*
unfolding *SUP-apply*
by (*intro measurable-cong nn-integral-monotone-convergence-SUP*) (*auto simp add*: *space-subprob-algebra*)
ultimately show *?case*
by (*simp add*: *ac-simps*)
qed

lemma *measurable-distr*:
assumes [*measurable*]: *f* ∈ *measurable M N*

shows ($\lambda M'$. *distr* M' N f) $\in$ *measurable* (*subprob-algebra* M) (*subprob-algebra* N)
proof (*cases space* $N = \{\}$)
assume *not-empty*: *space* $N \neq \{\}$
show *?thesis*
proof (*rule measurable-subprob-algebra*)
fix A **assume** A: $A \in$ *sets* N
then have ($\lambda M'$. *emeasure* (*distr* M' N f) A) $\in$ *borel-measurable* (*subprob-algebra* M) $\longleftrightarrow$
($\lambda M'$. *emeasure* M' (f $-$‘ $A \cap$ *space* M)) $\in$ *borel-measurable* (*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*=*op* $\cap$])
also have ...
using A **by** (*intro measurable-emeasure-subprob-algebra*) *simp*
finally show ($\lambda M'$. *emeasure* (*distr* M' N f) A) $\in$ *borel-measurable* (*subprob-algebra* M) **.**
qed (*auto intro*!: *subprob-space.subprob-space-distr simp*: *space-subprob-algebra not-empty cong*: *measurable-cong-sets*)
qed (*insert assms*, *auto simp*: *measurable-empty-iff space-subprob-algebra-empty-iff*)

lemma *emeasure-space-subprob-algebra*[*measurable*]:
(λa. *emeasure* a (*space* a)) $\in$ *borel-measurable* (*subprob-algebra* N)
proof−
have (λa. *emeasure* a (*space* N)) $\in$ *borel-measurable* (*subprob-algebra* N) (**is** *?f* $\in$ *?M*)
by (*rule measurable-emeasure-subprob-algebra*) *simp*
also have *?f* $\in$ *?M* $\longleftrightarrow$ (λa. *emeasure* a (*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 :: $'a \Rightarrow 'b$::{*banach*, *second-countable-topology*}
assumes [*measurable*]: $f \in$ *borel-measurable* N
shows *Measurable.pred* (*subprob-algebra* N) (λM. *integrable* M f)
proof (*rule measurable-cong*[*THEN iffD2*])
show $M \in$ *space* (*subprob-algebra* N) $\Longrightarrow$ *integrable* M f $\longleftrightarrow$ ($\int^+ x$. *norm* (f x) ∂M) $< \infty$ **for** M
by (*auto simp*: *space-subprob-algebra integrable-iff-bounded*)
qed *measurable*

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

```
  from borel-measurable-implies-sequence-metric[OF f, of 0]
  obtain F where F: ⋀i. simple-function N (F i)
    ⋀x. x ∈ space N ⟹ (λi. F i x) ⟶ f x
    ⋀i x. x ∈ space N ⟹ norm (F i x) ≤ 2 * norm (f x)
    unfolding norm-conv-dist by blast

  have [measurable]: F i ∈ N →M count-space UNIV for i
    using F(1) by (rule measurable-simple-function)

  def F' ≡ λM i. if integrable M f then integral^L M (F i) else 0

  have (λM. F' M i) ∈ subprob-algebra N →M borel for i
  proof (rule measurable-cong[THEN iffD2])
    fix M assume M ∈ space (subprob-algebra N)
    then have [simp]: sets M = sets N space M = space N 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 = (if integrable M f then Bochner-Integration.simple-bochner-integral
M (F i) 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 = (if integrable M f then ∑y∈F i ‘ space N. measure M
{x∈space N. F i x = y} *R y else 0)
      unfolding simple-bochner-integral-def by simp
  qed measurable
  moreover
  have F' M ⟶ integral^L M f if M: M ∈ space (subprob-algebra N) for M
  proof cases
    from M have [simp]: sets M = sets N space M = 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=λx. 2 * 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 ∈ measurable M (subprob-algebra N)
  assumes g: g ∈ measurable M (subprob-algebra L)
  shows (λx. f x ⨂M g x) ∈ measurable M (subprob-algebra (N ⨂M L))
proof (rule measurable-subprob-algebra)
  { fix x assume x ∈ space M
    with measurable-space[OF f] measurable-space[OF g]
```

have *fx*: $f\ x \in$ *space* (*subprob-algebra* N) **and** *gx*: $g\ x \in$ *space* (*subprob-algebra* L)
by *auto*
interpret F: *subprob-space* $f\ x$
using *fx* **by** (*simp add*: *space-subprob-algebra*)
interpret G: *subprob-space* $g\ x$
using *gx* **by** (*simp add*: *space-subprob-algebra*)

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

have *1*: $\bigwedge A\ B.\ A \in$ *sets* $N \Longrightarrow B \in$ *sets* $L \Longrightarrow$ *emeasure* $(f\ x \bigotimes_M g\ x)\ (A \times B) =$ *emeasure* $(f\ x)\ A *$ *emeasure* $(g\ x)\ B$
using *fx gx* **by** (*intro G.emeasure-pair-measure-Times*) (*auto simp*: *space-subprob-algebra*)
have *emeasure* $(f\ x \bigotimes_M g\ x)$ (*space* $(f\ x \bigotimes_M g\ x)) =$
emeasure $(f\ x)$ (*space* $(f\ x)) *$ *emeasure* $(g\ x)$ (*space* $(g\ x)$)
by (*subst G.emeasure-pair-measure-Times*[*symmetric*]) (*simp-all add*: *space-pair-measure*)
hence *2*: $\bigwedge A.\ A \in$ *sets* $(N \bigotimes_M L) \Longrightarrow$ *emeasure* $(f\ x \bigotimes_M g\ x)$ (*space* $N \times$ *space* $L - A) =$
... $-$ *emeasure* $(f\ x \bigotimes_M g\ x)\ A$
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 sets-eq* **}**
note *Times* = *this*(*1*) **and** *Compl* = *this*(*2*) **and** *sets-eq* = *this*(*3*)

fix A **assume** A: $A \in$ *sets* $(N \bigotimes_M L)$
show $(\lambda a.$ *emeasure* $(f\ a \bigotimes_M g\ a)\ A) \in$ *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: $A \in$ *sets* $(N \bigotimes_M L)$
by (*auto simp*: *sets-pair-measure*)
have $(\lambda x.$ *emeasure* $(f\ x)$ (*space* $(f\ x)) *$ *emeasure* $(g\ x)$ (*space* $(g\ x)) -$
emeasure $(f\ x \bigotimes_M g\ x)\ A) \in$ *borel-measurable* M (**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 *range A* $\subseteq$ *sets* $(N \bigotimes_M L)$ *disjoint-family A*
by (*auto simp*: *sets-pair-measure*)

```
    then have (λa. emeasure (f a ⨂M g a) (⋃i. A i)) ∈ borel-measurable M ⟷
      (λa. ∑i. emeasure (f a ⨂M g a) (A i)) ∈ 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 ≠ {} X ∈ sets K
  assumes N: N ∈ measurable M (subprob-algebra K)
  shows (λx. restrict-space (N x) X) ∈ measurable M (subprob-algebra (restrict-space
K X))
proof (rule measurable-subprob-algebra)
  fix a assume a: a ∈ space M
  from N[THEN measurable-space, OF this]
  have subprob-space (N a) and [simp]: sets (N a) = sets K space (N a) = 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 (restrict-space (N a) X)
  proof
    show space (restrict-space (N a) X) ≠ {}
      using X by (auto simp add: space-restrict-space)
    show emeasure (restrict-space (N a) X) (space (restrict-space (N a) X)) ≤ 1
    using X by (simp add: emeasure-restrict-space space-restrict-space subprob-emeasure-le-1)
  qed
  show sets (restrict-space (N a) X) = sets (restrict-space K X)
    by (intro sets-restrict-space-cong) fact
next
  fix A assume A: A ∈ sets (restrict-space K X)
  show (λa. emeasure (restrict-space (N a) X) A) ∈ borel-measurable M
  proof (subst measurable-cong)
    fix a assume a ∈ space M
    from N[THEN measurable-space, OF this]
    have [simp]: sets (N a) = sets K space (N a) = space K
      by (auto simp add: space-subprob-algebra dest: sets-eq-imp-space-eq)
    show emeasure (restrict-space (N a) X) A = emeasure (N a) (A ∩ X)
    using X A by (subst emeasure-restrict-space) (auto simp add: sets-restrict-space
ac-simps)
  next
    show (λw. emeasure (N w) (A ∩ X)) ∈ borel-measurable M
      using A X
      by (intro measurable-compose[OF N measurable-emeasure-subprob-algebra])
         (auto simp: sets-restrict-space)
  qed
```

qed

19 Properties of return

definition *return* :: *'a measure* ⇒ *'a* ⇒ *'a measure* **where**
return R x = *measure-of* (*space R*) (*sets R*) (λ*A*. *indicator A x*)

lemma *space-return*[*simp*]: *space* (*return M x*) = *space M*
by (*simp add*: *return-def*)

lemma *sets-return*[*simp*]: *sets* (*return M x*) = *sets M*
by (*simp add*: *return-def*)

lemma *measurable-return1*[*simp*]: *measurable* (*return N x*) *L* = *measurable N L*
by (*simp cong*: *measurable-cong-sets*)

lemma *measurable-return2*[*simp*]: *measurable L* (*return N x*) = *measurable L N*
by (*simp cong*: *measurable-cong-sets*)

lemma *return-sets-cong*: *sets M* = *sets N* ⟹ *return M* = *return N*
by (*auto dest*: *sets-eq-imp-space-eq simp*: *fun-eq-iff return-def*)

lemma *return-cong*: *sets A* = *sets B* ⟹ *return A x* = *return B x*
by (*auto simp add*: *return-def dest*: *sets-eq-imp-space-eq*)

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

lemma *prob-space-return*: *x* ∈ *space M* ⟹ *prob-space* (*return M x*)
by *rule simp*

lemma *subprob-space-return*: *x* ∈ *space M* ⟹ *subprob-space* (*return M x*)
by (*intro prob-space-return prob-space-imp-subprob-space*)

lemma *subprob-space-return-ne*:
assumes *space M* ≠ {} **shows** *subprob-space* (*return M x*)
proof
show *emeasure* (*return M x*) (*space* (*return M x*)) ≤ *1*
by (*subst emeasure-return*) (*auto split*: *split-indicator*)
qed (*simp*, *fact*)

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

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

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

lemma *integral-return*:
fixes $g :: - \Rightarrow 'a :: \{banach, second\text{-}countable\text{-}topology\}$
assumes $x \in space\ M\ g \in borel\text{-}measurable\ M$
shows $(\int a.\ g\ a\ \partial return\ M\ x) = g\ x$
proof−
interpret *prob-space return M x* **by** (*rule prob-space-return*[*OF* ‹$x \in space\ M$›])
have $(\int a.\ g\ a\ \partial return\ M\ x) = (\int a.\ g\ x\ \partial return\ M\ x)$ **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* $\in$ *measurable N* (*subprob-algebra N*)
by (*rule measurable-subprob-algebra*) (*auto simp*: *subprob-space-return*)

lemma *distr-return*:
assumes $f \in measurable\ M\ N$ **and** $x \in space\ M$
shows *distr* (*return M x*) *N f* = *return N* (*f x*)
using *assms* **by** (*intro measure-eqI*) (*simp-all add*: *indicator-def emeasure-distr*)

lemma *return-restrict-space*:
  $\Omega \in$ *sets* $M \Longrightarrow$ *return* (*restrict-space* $M\ \Omega$) $x =$ *restrict-space* (*return* $M\ x$) Ω
  by (*auto intro*!: *measure-eqI simp*: *sets-restrict-space emeasure-restrict-space*)

lemma *measurable-distr2*:
  assumes f[*measurable*]: *case-prod* $f \in$ *measurable* ($L \bigotimes_M M$) N
  assumes g[*measurable*]: $g \in$ *measurable* L (*subprob-algebra* M)
  shows ($\lambda x.$ *distr* ($g\ x$) N ($f\ x$)) $\in$ *measurable* L (*subprob-algebra* N)
proof −
  have ($\lambda x.$ *distr* ($g\ x$) N ($f\ x$)) $\in$ *measurable* L (*subprob-algebra* N)
    $\longleftrightarrow$ ($\lambda x.$ *distr* (*return* $L\ x \bigotimes_M g\ x$) N (*case-prod* f)) $\in$ *measurable* L (*subprob-algebra* N)
  proof (*rule measurable-cong*)
    fix x **assume** x: $x \in$ *space* L
    have gx: $g\ x \in$ *space* (*subprob-algebra* M)
      using *measurable-space*[*OF* $g\ x$] **.**
    then have [*simp*]: *sets* ($g\ x$) = *sets* M
      by (*simp add*: *space-subprob-algebra*)
    then have [*simp*]: *space* ($g\ x$) = *space* M
      by (*rule sets-eq-imp-space-eq*)
    let *?R* = *return* $L\ x$
    from *measurable-compose-Pair1*[*OF* $x\ f$] **have** *f-M'*: $f\ x \in$ *measurable* $M\ N$
      by *simp*
    interpret *subprob-space* $g\ x$
      using gx **by** (*simp add*: *space-subprob-algebra*)
    have *space-pair-M'*[*simp*]: $\bigwedge X.$ *space* ($X \bigotimes_M g\ x$) = *space* ($X \bigotimes_M M$)
      by (*simp add*: *space-pair-measure*)
    show *distr* ($g\ x$) N ($f\ x$) = *distr* (*?R* $\bigotimes_M g\ x$) N (*case-prod* f) (**is** *?l* = *?r*)
    proof (*rule measure-eqI*)
      show *sets ?l* = *sets ?r*
        by *simp*
    next
      fix A **assume** $A \in$ *sets ?l*
      then have A[*measurable*]: $A \in$ *sets* N
        by *simp*
      then have *emeasure ?r* A = *emeasure* (*?R* $\bigotimes_M g\ x$) ($(\lambda(x,\ y).\ f\ x\ y) -` A \cap$ *space* (*?R* $\bigotimes_M g\ x$))
        by (*auto simp add*: *emeasure-distr f-M' cong*: *measurable-cong-sets*)
      also have ... = ($\int^+ M''$. *emeasure* ($g\ x$) ($f\ M'' -` A \cap$ *space* M) ∂*?R*)
        apply (*subst emeasure-pair-measure-alt*)
        apply (*rule measurable-sets*[*OF* - A])
        apply (*auto simp add*: *f-M' cong*: *measurable-cong-sets*)
        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* ($g\ x$) ($f\ x -` A \cap$ *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 ...
  apply (intro measurable-compose[OF measurable-pair-measure measurable-distr])
  apply (rule return-measurable)
  apply measurable
  done
 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. integral^N (L x) (f x)) ∈ borel-measurable M
proof −
 note nn-integral-measurable-subprob-algebra[measurable]
 note measurable-distr2[measurable]
 have (λx. integral^N (distr (L x) (M ⨂M N) (λy. (x, y))) (λ(x, y). f x y)) ∈
borel-measurable M
   by measurable
 then show (λx. integral^N (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 ∗: ⋀x. fst x ∈ space M ⟹ snd x ∈ A (fst x) ⟷ x ∈ (SIGMA x:space
M. A x)
   by (auto simp: fun-eq-iff)
 have (λ(x, y). indicator (A x) y::ennreal) ∈ borel-measurable (M ⨂M N)
   apply measurable
   apply (subst measurable-cong)
   apply (rule ∗)
   apply (auto simp: space-pair-measure)
   done
 then have (λx. integral^N (L x) (indicator (A x))) ∈ borel-measurable M
```

```
    by (intro nn-integral-measurable-subprob-algebra2[where N=N] L)
  then show (λx. emeasure (L x) (A x)) ∈ borel-measurable M
    apply (rule measurable-cong[THEN iffD1, rotated])
    apply (rule nn-integral-indicator)
    apply (simp add: subprob-measurableD[OF L] **)
    done
qed

lemma measure-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. measure (L x) (A x)) ∈ 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) ⟹ 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) ≠ {}
    with eq show ?thesis
      by (fastforce simp add: space-subprob-algebra)
  qed
qed

lemma space-select-sets[simp]:
  sets M = sets (subprob-algebra N) ⟹ space (select-sets M) = space N
```

by (*intro sets-eq-imp-space-eq sets-select-sets*)

20 Join

definition *join* :: *'a measure measure* ⇒ *'a measure* **where**
join M = *measure-of* (*space* (*select-sets M*)) (*sets* (*select-sets M*)) (λ*B*. ∫⁺ *M'*. *emeasure M' B* ∂*M*)

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

lemma *emeasure-join*:
assumes *M*[*simp*, *measurable-cong*]: *sets M* = *sets* (*subprob-algebra N*) **and** *A*: *A* ∈ *sets N*
shows *emeasure* (*join M*) *A* = (∫⁺ *M'*. *emeasure M' A* ∂*M*)
proof (*rule emeasure-measure-of*[*OF join-def*])
show *countably-additive* (*sets* (*join M*)) (λ*B*. ∫⁺ *M'*. *emeasure M' B* ∂*M*)
proof (*rule countably-additiveI*)
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A* ⊆ *sets* (*join M*) *disjoint-family A*
have (∑ *i*. ∫⁺ *M'*. *emeasure M'* (*A i*) ∂*M*) = (∫⁺*M'*. (∑ *i*. *emeasure M'* (*A i*)) ∂*M*)
using *A* **by** (*subst nn-integral-suminf*) (*auto simp*: *measurable-emeasure-subprob-algebra*)
also have ... = (∫⁺*M'*. *emeasure M'* (⋃ *i*. *A i*) ∂*M*)
proof (*rule nn-integral-cong*)
fix *M'* **assume** *M'* ∈ *space M*
then show (∑ *i*. *emeasure M'* (*A i*)) = *emeasure M'* (⋃ *i*. *A i*)
using *A sets-eq-imp-space-eq*[*OF M*] **by** (*simp add*: *suminf-emeasure space-subprob-algebra*)
qed
finally show (∑ *i*. ∫⁺*M'*. *emeasure M'* (*A i*) ∂*M*) = (∫⁺*M'*. *emeasure M'* (⋃ *i*. *A i*) ∂*M*) **.**
qed
qed (*auto simp*: *A sets.space-closed positive-def*)

lemma *measurable-join*:
join ∈ *measurable* (*subprob-algebra* (*subprob-algebra N*)) (*subprob-algebra N*)
proof (*cases space N* ≠ {}, *rule measurable-subprob-algebra*)
fix *A* **assume** *A* ∈ *sets N*
let *?B* = *borel-measurable* (*subprob-algebra* (*subprob-algebra N*))
have (λ*M'*. *emeasure* (*join M'*) *A*) ∈ *?B* ⟷ (λ*M'*. (∫⁺ *M''*. *emeasure M'' A* ∂*M'*)) ∈ *?B*
proof (*rule measurable-cong*)
fix *M'* **assume** *M'* ∈ *space* (*subprob-algebra* (*subprob-algebra N*))
then show *emeasure* (*join M'*) *A* = (∫⁺ *M''*. *emeasure M'' A* ∂*M'*)
by (*intro emeasure-join*) (*auto simp*: *space-subprob-algebra* ‹*A*∈*sets N*›)
qed
also have (λ*M'*. ∫⁺*M''*. *emeasure M'' A* ∂*M'*) ∈ *?B*

using *measurable-emeasure-subprob-algebra*[*OF* ‹*A*∈*sets N*›]
by (*rule nn-integral-measurable-subprob-algebra*)
finally show (λ*M'*. *emeasure* (*join M'*) *A*) ∈ *borel-measurable* (*subprob-algebra* (*subprob-algebra N*)) **.**
next
assume [*simp*]: *space N* ≠ {}
fix *M* **assume** *M*: *M* ∈ *space* (*subprob-algebra* (*subprob-algebra N*))
then have (∫⁺*M'*. *emeasure M'* (*space N*) ∂*M*) ≤ (∫⁺*M'*. *1* ∂*M*)
apply (*intro nn-integral-mono*)
apply (*auto simp*: *space-subprob-algebra*
dest!: *sets-eq-imp-space-eq subprob-space.emeasure-space-le-1*)
done
with *M* **show** *subprob-space* (*join M*)
by (*intro subprob-spaceI*)
(*auto simp*: *emeasure-join space-subprob-algebra M assms dest*: *subprob-space.emeasure-space-le-1*)
next
assume ¬(*space N* ≠ {})
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* ∈ *borel-measurable N*
and *M*[*measurable-cong*]: *sets M* = *sets* (*subprob-algebra N*)
shows (∫⁺*x*. *f x* ∂*join M*) = (∫⁺*M'*. ∫⁺*x*. *f x* ∂*M' ∂M*)
using *f*
proof *induct*
case (*cong f g*)
moreover have *integral*^{N} (*join M*) *f* = *integral*^{N} (*join M*) *g*
by (*intro nn-integral-cong cong*) (*simp add*: *M*)
moreover from *M* **have** (∫⁺ *M'*. *integral*^{N} *M' f* ∂*M*) = (∫⁺ *M'*. *integral*^{N} *M' g* ∂*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*)
moreover with *M* **have** (∫⁺ *M'*. *integral*^{N} *M'* (*indicator A*) ∂*M*) = (∫⁺ *M'*. *emeasure M' A* ∂*M*)
by (*intro nn-integral-cong nn-integral-indicator*)
(*auto simp*: *space-subprob-algebra dest*!: *sets-eq-imp-space-eq*)
ultimately show *?case*
using *M* **by** (*simp add*: *emeasure-join*)
next
case (*mult f c*)
have (∫⁺ *M'*. ∫⁺ *x*. *c* ∗ *f x* ∂*M' ∂M*) = (∫⁺ *M'*. *c* ∗ ∫⁺ *x*. *f x* ∂*M' ∂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 ... = *c* ∗ (∫⁺ *M'*. ∫⁺ *x*. *f x* ∂*M' ∂M*)

using *nn-integral-measurable-subprob-algebra*[*OF mult*(*2*)]
by (*intro nn-integral-cmult mult*) (*simp add*: *M*)
also have ... = $c * (integral^N\ (join\ M)\ f)$
by (*simp add*: *mult*)
also have ... = $(\int^+ x.\ c * f\ x\ \partial join\ M)$
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 ... = $(\int^+ M'.\ \int^+ x.\ f\ x\ \partial M'\ \partial M) + (\int^+ M'.\ \int^+ x.\ g\ x\ \partial M'\ \partial M)$
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 ... = $(integral^N\ (join\ M)\ f) + (integral^N\ (join\ M)\ g)$
by (*simp add*: *add*)
also have ... = $(\int^+ x.\ f\ x + g\ x\ \partial join\ M)$
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 $(\int^+ M'.\ \int^+ x.\ (SUP\ i.\ F\ i)\ x\ \partial M'\ \partial M) = (\int^+ M'.\ (SUP\ i.\ \int^+ x.\ F\ i\ x\ \partial M')\ \partial M)$
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 ... = $(SUP\ i.\ \int^+ M'.\ \int^+ x.\ F\ i\ x\ \partial M'\ \partial M)$
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 ... = $(SUP\ i.\ integral^N\ (join\ M)\ (F\ i))$
by (*simp add*: *seq*)
also have ... = $(\int^+ x.\ (SUP\ i.\ F\ i\ x)\ \partial join\ M)$
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*)
qed

lemma *measurable-join1*:
$[\![f \in measurable\ N\ K;\ sets\ M = sets\ (subprob\text{-}algebra\ N)\ ]\!]$
$\Longrightarrow f \in measurable\ (join\ M)\ K$
by(*simp add*: *measurable-def*)

lemma

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

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

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

{ fix $f\ M'$
assume [*measurable*]: $f \in borel\text{-}measurable\ N$
and *f-bounded*: $\bigwedge x.\ x \in space\ N \Longrightarrow f\ x \leq B$
and $M' \in space\ M\ emeasure\ M'\ (space\ M') \leq ennreal\ B'$
have $AE\ x\ in\ M'.\ ennreal\ (f\ x) \leq ennreal\ B$
proof(*rule AE-I2*)
fix x
assume $x \in space\ M'$
with ‹$M' \in space\ M$› *sets-eq-imp-space-eq*[*OF M*]
have $x \in space\ N$ **by**(*auto simp add*: *space-subprob-algebra dest*: *sets-eq-imp-space-eq*)
from *f-bounded*[*OF this*] **show** $ennreal\ (f\ x) \leq ennreal\ B$ **by** *simp*
qed
then have $(\int^+ x.\ ennreal\ (f\ x)\ \partial M') \leq (\int^+ x.\ ennreal\ B\ \partial M')$
by(*rule nn-integral-mono-AE*)
also have $\ldots = ennreal\ B * emeasure\ M'\ (space\ M')$ **by**(*simp*)
also have $\ldots \leq ennreal\ B * ennreal\ B'$ **by**(*rule mult-left-mono*)(*fact*, *simp*)
also have $\ldots \leq ennreal\ B * ennreal\ |B'|$ **by**(*rule mult-left-mono*)(*simp-all*)
finally have $(\int^+ x.\ ennreal\ (f\ x)\ \partial M') \leq ennreal\ (B * |B'|)$ **by** (*simp add*: *ennreal-mult*) **}**

note *bounded1 = this*

have *bounded*:
$\bigwedge f.\ [\![\ f \in$ *borel-measurable* $N;\ \bigwedge x.\ x \in$ *space* $N \Longrightarrow f\,x \le B\ ]\!]$
$\Longrightarrow (\int^+ x.$ *ennreal* $(f\,x)\ \partial$*join* $M) \neq$ *top*
proof −
fix *f*
assume [*measurable*]: $f \in$ *borel-measurable* N
and *f-bounded*: $\bigwedge x.\ x \in$ *space* $N \Longrightarrow f\,x \le B$
have $(\int^+ x.$ *ennreal* $(f\,x)\ \partial$*join* $M) = (\int^+ M'.\ \int^+ x.$ *ennreal* $(f\,x)\ \partial M'\ \partial M)$
by(*rule nn-integral-join*[*OF - M*]) *simp*
also have $\ldots \le \int^+ M'.\ B * |B'|\ \partial M$
using *bounded1*[*OF* ‹$f \in$ *borel-measurable* N› *f-bounded*]
by(*rule nn-integral-mono-AE*[*OF AE-mp*[*OF M-bounded AE-I2*], *rule-format*])
also have $\ldots =$ $B * |B'| *$ *emeasure* M (*space* M) **by** *simp*
also have $\ldots < \infty$
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*: $(\int^+ x.$ *ennreal* $(f\,x)\ \partial$*join* $M) \neq \infty$
and *f-neg*: $(\int^+ x.$ *ennreal* $(-f\,x)\ \partial$*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*: $(\int^+ x.\ f\,x\ \partial$*join* $M) = \int^+ M'.\ \int^+ x.\ f\,x\ \partial M'\ \partial M$
by(*simp add*: *nn-integral-join*[*OF - M*])
have *int-mf*: $(\int^+ x.\ -f\,x\ \partial$*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' *in* $M.\ (\int^+ x.\ f\,x\ \partial M') \neq \infty$
using *AE-space M-bounded*
proof *eventually-elim*
fix M' **assume** $M' \in$ *space* M *emeasure* M' (*space* M') $\le$ *ennreal* B'
then have $(\int^+ x.$ *ennreal* $(f\,x)\ \partial M') \le$ *ennreal* $(B * |B'|)$
using *f-measurable* **by**(*auto intro*!: *bounded1 dest*: *f-bounded*)
then show $(\int^+ x.$ *ennreal* $(f\,x)\ \partial M') \neq \infty$
by (*auto simp*: *top-unique*)
qed
hence [*simp*]: $(\int^+ M'.$ *ennreal* $($*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-pos* **have** [*simp*]: *integrable* M $(\lambda M'.$ *enn2real* $(\int^+ x.\ f\,x\ \partial M'))$
by(*simp add*: *int-f real-integrable-def nn-integral-0-iff-AE*[*THEN iffD2*] *ennreal-neg enn2real-nonneg*)

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

have $(\int x.\ f\ x\ \partial$*join* $M) =$ *enn2real* $(\int^+ N.\ \int^+ x.\ f\ x\ \partial N\ \partial M) -$ *enn2real* $(\int^+ N.\ \int^+ x. -f\ x\ \partial N\ \partial M)$
unfolding *real-lebesgue-integral-def*[*OF* ‹*?integrable*›] **by** (*simp add*: *nn-integral-join*[*OF - M*])
also have $\ldots = (\int N.$ *enn2real* $(\int^+ x.\ f\ x\ \partial N)\ \partial M) - (\int N.$ *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 $\ldots = (\int N.$ *enn2real* $(\int^+ x.\ f\ x\ \partial N) -$ *enn2real* $(\int^+ x. -f\ x\ \partial N)\ \partial M)$
by *simp*
also have $\ldots = \int M'.\ \int x.\ f\ x\ \partial M'\ \partial M$
proof (*rule integral-cong-AE*)
show $AE\ x$ *in* $M.$
enn2real $(\int^+ x.$ *ennreal* $(f\ x)\ \partial x) -$ *enn2real* $(\int^+ x.$ *ennreal* $(-f\ x)\ \partial x) = integral^L\ x\ f$
using *AE-space M-bounded*
proof *eventually-elim*
fix M' **assume** $M' \in$ *space* M *emeasure* M' (*space* M') $\leq B'$
then interpret *subprob-space* M'
by (*auto simp*: M[*THEN sets-eq-imp-space-eq*] *space-subprob-algebra*)

from ‹$M' \in$ *space* M› *sets-eq-imp-space-eq*[*OF* M]
have [*measurable-cong*]: *sets* $M' =$ *sets* N **by**(*simp add*: *space-subprob-algebra*)
hence [*simp*]: *space* $M' =$ *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*
apply (*rule measure-eqI*)
apply (*simp add*: *assms*)
apply (*subgoal-tac return N* ∈ *measurable M* (*subprob-algebra N*))
apply (*simp add*: *emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra assms*)
apply (*subst measurable-cong-sets*, *rule assms*[*symmetric*], *rule refl*, *rule return-measurable*)
done

lemma *join-distr-distr*:
fixes *f* :: *'a* ⇒ *'b* **and** *M* :: *'a measure measure* **and** *N* :: *'b measure*
assumes *sets M* = *sets* (*subprob-algebra R*) **and** *f* ∈ *measurable R N*
shows *join* (*distr M* (*subprob-algebra N*) (λ*M*. *distr M N f*)) = *distr* (*join M*) *N f* (**is** *?r* = *?l*)
proof (*rule measure-eqI*)
fix *A* **assume** *A* ∈ *sets ?r*
hence *A-in-N*: *A* ∈ *sets N* **by** *simp*

from *assms* **have** *f* ∈ *measurable* (*join M*) *N*
by (*simp cong*: *measurable-cong-sets*)
moreover from *assms* **and** *A-in-N* **have** *f*−‘*A* ∩ *space R* ∈ *sets R*
by (*intro measurable-sets*) *simp-all*

ultimately have *emeasure* (*distr* (*join M*) *N f*) *A* = $\int^+ M'$. *emeasure M'* (*f* −‘*A* ∩ *space R*) *∂M*
by (*simp-all add*: *A-in-N emeasure-distr emeasure-join assms*)

also have ... = $\int^+ x$. *emeasure* (*distr x N f*) *A ∂M* **using** *A-in-N*
proof (*intro nn-integral-cong*, *subst emeasure-distr*)
fix *M'* **assume** *M'* ∈ *space M*
from *assms* **have** *space M* = *space* (*subprob-algebra R*)
using *sets-eq-imp-space-eq* **by** *blast*
with ‹*M'* ∈ *space M*› **have** [*simp*]: *sets M'* = *sets R* **using** *space-subprob-algebra*
by *blast*
show *f* ∈ *measurable M' N* **by** (*simp cong*: *measurable-cong-sets add*: *assms*)
have *space M'* = *space R* **by** (*rule sets-eq-imp-space-eq*) *simp*
thus *emeasure M'* (*f* −‘ *A* ∩ *space R*) = *emeasure M'* (*f* −‘ *A* ∩ *space M'*)
by *simp*
qed

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

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

adhoc-overloading *Monad-Syntax.bind bind*

lemma *bind-empty*:
space M = {} ⟹ *bind M f* = *count-space* {}
by (*simp add*: *bind-def*)

lemma *bind-nonempty*:
space M ≠ {} ⟹ *bind M f* = *join* (*distr M* (*subprob-algebra* (*f* (*SOME x*. *x* ∈ *space M*))) *f*)
by (*simp add*: *bind-def*)

lemma *sets-bind-empty*: *sets M* = {} ⟹ *sets* (*bind M f*) = {{}}
by (*auto simp*: *bind-def*)

lemma *space-bind-empty*: *space M* = {} ⟹ *space* (*bind M f*) = {}
by (*simp add*: *bind-def*)

lemma *sets-bind*[*simp*, *measurable-cong*]:
assumes *f*: ⋀*x*. *x* ∈ *space M* ⟹ *sets* (*f x*) = *sets N* **and** *M*: *space M* ≠ {}
shows *sets* (*bind M f*) = *sets N*

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

lemma *space-bind*[*simp*]:
assumes ⋀*x*. *x* ∈ *space M* ⟹ *sets* (*f x*) = *sets N* **and** *space M* ≠ {}
shows *space* (*bind M f*) = *space N*
using *assms* **by** (*intro sets-eq-imp-space-eq sets-bind*)

lemma *bind-cong*:
assumes ∀ *x* ∈ *space M*. *f x* = *g x*
shows *bind M f* = *bind M g*
proof (*cases space M* = {})
assume *space M* ≠ {}
hence (*SOME x*. *x* ∈ *space M*) ∈ *space M* **by** (*rule-tac someI-ex*) *blast*
with *assms* **have** *f* (*SOME x*. *x* ∈ *space M*) = *g* (*SOME x*. *x* ∈ *space M*) **by** *blast*
with ‹*space M* ≠ {}› **and** *assms* **show** *?thesis* **by** (*simp add*: *bind-nonempty cong*: *distr-cong*)
qed (*simp add*: *bind-empty*)

lemma *bind-nonempty′*:
assumes *f* ∈ *measurable M* (*subprob-algebra N*) *x* ∈ *space M*
shows *bind M f* = *join* (*distr M* (*subprob-algebra N*) *f*)
using *assms*
apply (*subst bind-nonempty*, *blast*)
apply (*subst subprob-algebra-cong*[*OF sets-kernel*[*OF assms*(*1*) *someI-ex*]], *blast*)
apply (*simp add*: *subprob-algebra-cong*[*OF sets-kernel*[*OF assms*]])
done

lemma *bind-nonempty″*:
assumes *f* ∈ *measurable M* (*subprob-algebra N*) *space M* ≠ {}
shows *bind M f* = *join* (*distr M* (*subprob-algebra N*) *f*)
using *assms* **by** (*auto intro*: *bind-nonempty′*)

lemma *emeasure-bind*:
⟦*space M* ≠ {}; *f* ∈ *measurable M* (*subprob-algebra N*);*X* ∈ *sets N*⟧
⟹ *emeasure* (*M* ⤜ *f*) *X* = ∫⁺*x*. *emeasure* (*f x*) *X* ∂*M*
by (*simp add*: *bind-nonempty″ emeasure-join nn-integral-distr measurable-emeasure-subprob-algebra*)

lemma *nn-integral-bind*:
assumes *f*: *f* ∈ *borel-measurable B*
assumes *N*: *N* ∈ *measurable M* (*subprob-algebra B*)
shows (∫⁺*x*. *f x* ∂(*M* ⤜ *N*)) = (∫⁺*x*. ∫⁺*y*. *f y* ∂*N x* ∂*M*)
proof *cases*
assume *M*: *space M* ≠ {} **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 *P*[*measurable*]: *Measurable.pred B P*
assumes *N*[*measurable*]: *N* ∈ *measurable M* (*subprob-algebra B*)
shows (*AE x in M* ⤜ *N*. *P x*) ⟷ (*AE x in M*. *AE y in N x*. *P y*)
proof *cases*
assume *M*: *space M* = {} **show** *?thesis*
unfolding *bind-empty*[*OF M*] **unfolding** *space-empty*[*OF M*] **by** (*simp add*: *AE-count-space*)
next
assume *M*: *space M* ≠ {}
note *sets-kernel*[*OF N*, *simp*]
have ∗: (∫⁺*x*. *indicator* {*x*. ¬ *P x*} *x* ∂(*M* ⤜ *N*)) = (∫⁺*x*. *indicator* {*x*∈*space B*. ¬ *P x*} *x* ∂(*M* ⤜ *N*))
by (*intro nn-integral-cong*) (*simp add*: *space-bind*[*OF* - *M*] *split*: *split-indicator*)

have (*AE x in M* ⤜ *N*. *P x*) ⟷ (∫⁺ *x*. *integral*^{N} (*N x*) (*indicator* {*x* ∈ *space B*. ¬ *P x*}) ∂*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 ... = (*AE x in M*. *AE y in N x*. *P y*)
apply (*subst nn-integral-0-iff-AE*)
apply (*rule measurable-compose*[*OF N nn-integral-measurable-subprob-algebra*])
apply *measurable*
apply (*intro eventually-subst AE-I2*)
apply (*auto simp add*: *subprob-measurableD*(*1*)[*OF N*]
intro!: *AE-iff-measurable*[*symmetric*])
done
finally show *?thesis* **.**
qed

lemma *measurable-bind′*:
assumes *M1*: *f* ∈ *measurable M* (*subprob-algebra N*) **and**
M2: *case-prod g* ∈ *measurable* (*M* ⨂$_M$ *N*) (*subprob-algebra R*)
shows (λ*x*. *bind* (*f x*) (*g x*)) ∈ *measurable M* (*subprob-algebra R*)
proof (*subst measurable-cong*)
fix *x* **assume** *x-in-M*: *x* ∈ *space M*
with *assms* **have** *space* (*f x*) ≠ {}
by (*blast dest*: *subprob-space-kernel subprob-space.subprob-not-empty*)
moreover from *M2 x-in-M* **have** *g x* ∈ *measurable* (*f x*) (*subprob-algebra R*)
by (*subst measurable-cong-sets*[*OF sets-kernel*[*OF M1 x-in-M*] *refl*])
(*auto dest*: *measurable-Pair2*)
ultimately show *bind* (*f x*) (*g x*) = *join* (*distr* (*f x*) (*subprob-algebra R*) (*g x*))
by (*simp-all add*: *bind-nonempty′*)
next
show (λ*w*. *join* (*distr* (*f w*) (*subprob-algebra R*) (*g w*))) ∈ *measurable M* (*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*: $f \in$ *measurable* M (*subprob-algebra* N) **and**
M2: $(\lambda x.\ g\ (fst\ x)\ (snd\ x)) \in$ *measurable* $(M \bigotimes_M N)$ (*subprob-algebra* R)
shows $(\lambda x.\ bind\ (f\ x)\ (g\ x)) \in$ *measurable* M (*subprob-algebra* R)
using *assms* **by** (*auto intro*: *measurable-bind'* *simp*: *measurable-split-conv*)

lemma *measurable-bind2*:
assumes $f \in$ *measurable* M (*subprob-algebra* N) **and** $g \in$ *measurable* N (*subprob-algebra* R)
shows $(\lambda x.\ bind\ (f\ x)\ g) \in$ *measurable* M (*subprob-algebra* R)
using *assms* **by** (*intro measurable-bind' measurable-const*) *auto*

lemma *subprob-space-bind*:
assumes *subprob-space* $M\ f \in$ *measurable* M (*subprob-algebra* N)
shows *subprob-space* $(M \gg\!= f)$
proof (*rule subprob-space-kernel*[*of* $\lambda x.\ x \gg\!= f$])
show $(\lambda x.\ x \gg\!= f) \in$ *measurable* (*subprob-algebra* M) (*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** $M \in$ *space* (*subprob-algebra* M)
by (*simp add*: *space-subprob-algebra*)
qed

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

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

have $integral^L\ (join\ (distr\ M\ (subprob\text{-}algebra\ K)\ N))\ f = \int M'.\ integral^L\ M'\ f\ \partial distr\ M\ (subprob\text{-}algebra\ K)\ N$
using *f-measurable f-bounded*

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

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

lemma (**in** *subprob-space*) *bind-in-space*:
A ∈ *measurable M* (*subprob-algebra N*) ⟹ (*M* ⤜ *A*) ∈ *space* (*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* ∈ *measurable M* (*subprob-algebra N*) **and** *X*: *X* ∈ *sets N*
shows *measure* (*M* ⤜ *f*) *X* = ∫ *x*. *measure* (*f x*) *X* ∂*M*
proof −
interpret *Mf*: *subprob-space M* ⤜ *f*
by (*rule subprob-space-bind*[*OF* - *f*]) *unfold-locales*

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

have *emeasure* (*M* ⤜ *f*) *X* = ∫$^+$*x*. *emeasure* (*f x*) *X* ∂*M*
using *subprob-not-empty f X* **by** (*rule emeasure-bind*)
also have ... = ∫$^+$*x*. *ennreal* (*measure* (*f x*) *X*) ∂*M*
by (*intro nn-integral-cong*) *simp*

also have $\ldots = \int x.\ measure\ (f\ x)\ X\ \partial M$
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.emeasure-eq-measure measure-nonneg integral-nonneg*)
qed

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

lemma *emeasure-bind-const'*:
assumes *subprob-space M subprob-space N*
shows $emeasure\ (M \gg\!= (\lambda x.\ N))\ X = emeasure\ N\ X * emeasure\ M\ (space\ M)$
using *assms*
proof (*case-tac* $X \in sets\ N$)
fix *X* **assume** $X \in sets\ N$
thus $emeasure\ (M \gg\!= (\lambda x.\ N))\ X = emeasure\ N\ X * emeasure\ M\ (space\ M)$
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** $X \notin sets\ N$
with *assms* **show** $emeasure\ (M \gg\!= (\lambda x.\ N))\ X = emeasure\ N\ X * emeasure\ M\ (space\ M)$
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 $f \in measurable\ M\ (subprob\text{-}algebra\ N)$ **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* $\gg\!=$ (λx. *distr* (*N x*) *R f*))
unfolding *bind-nonempty''*[*OF N*]
apply (*subst bind-nonempty''*[*OF measurable-compose*[*OF N*(*1*) *measurable-distr*] *N*(*2*)])
apply (*rule f*)
apply (*simp add*: *join-distr-distr*[*OF - f*, *symmetric*])
apply (*subst distr-distr*[*OF measurable-distr*, *OF f N*(*1*)])
apply (*simp add*: *comp-def*)
done

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* $\gg\!=$ (λx. *N* (*f x*)))
proof −
have *space X* $\neq \{\}$ *space M* $\neq \{\}$
using ‹*space M* $\neq \{\}$› *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* $\{x\}$ $\gg\!=$ *f* = *f x*
proof−
have *A*: $\bigwedge A.\ A \subseteq \{x\} \Longrightarrow A = \{\} \vee A = \{x\}$ **by** *auto*
have *count-space* $\{x\}$ = *return* (*count-space* $\{x\}$) *x*
by (*intro measure-eqI*) (*auto dest*: *A*)
also have ... $\gg\!=$ *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*: $N \in$ *measurable M* (*subprob-algebra K*)
assumes *space M* $\neq \{\}$
assumes *X*[*simp*]: $X \in$ *sets K X* $\neq \{\}$
shows *restrict-space* (*bind M N*) *X* = *bind M* (λx. *restrict-space* (*N x*) *X*)
proof (*rule measure-eqI*)
note *N-sets* = *sets-bind*[*OF sets-kernel*[*OF N*] *assms*(*2*), *simp*]
note *N-space* = *sets-eq-imp-space-eq*[*OF N-sets*, *simp*]
show *sets* (*restrict-space* (*bind M N*) *X*) = *sets* (*bind M* (λx. *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* ∈ *sets* (*restrict-space* (*M* ⤜ *N*) *X*)
with *X* **have** *A* ∈ *sets K A* ⊆ *X*
by (*auto simp*: *sets-restrict-space*)
then show *emeasure* (*restrict-space* (*M* ⤜ *N*) *X*) *A* = *emeasure* (*M* ⤜ (λ*x*. *restrict-space* (*N x*) *X*)) *A*
using *assms*
apply (*subst emeasure-restrict-space*)
apply (*simp-all add*: *emeasure-bind*[*OF assms*(*2*,*1*)])
apply (*subst emeasure-bind*[*OF - restrict-space-measurable*[*OF - - N*]])
apply (*auto simp*: *sets-restrict-space emeasure-restrict-space space-subprob-algebra intro*!: *nn-integral-cong dest*!: *measurable-space*)
done
qed

lemma *bind-restrict-space*:
assumes *A*: *A* ∩ *space M* ≠ {} *A* ∩ *space M* ∈ *sets M*
and *f*: *f* ∈ *measurable* (*restrict-space M A*) (*subprob-algebra N*)
shows *restrict-space M A* ⤜ *f* = *M* ⤜ (λ*x*. *if x* ∈ *A then f x else null-measure* (*f* (*SOME x*. *x* ∈ *A* ∧ *x* ∈ *space M*)))
(**is** *?lhs* = *?rhs* **is** - = *M* ⤜ *?f*)
proof −
let *?P* = λ*x*. *x* ∈ *A* ∧ *x* ∈ *space M*
let *?x* = *Eps ?P*
let *?c* = *null-measure* (*f ?x*)
from *A* **have** *?P ?x* **by**−(*rule someI-ex*, *blast*)
hence *?x* ∈ *space* (*restrict-space M A*) **by**(*simp add*: *space-restrict-space*)
with *f* **have** *f ?x* ∈ *space* (*subprob-algebra N*) **by**(*rule measurable-space*)
hence *sps*: *subprob-space* (*f ?x*)
and *sets*: *sets* (*f ?x*) = *sets N*
by(*simp-all add*: *space-subprob-algebra*)
have *space* (*f ?x*) ≠ {} **using** *sps* **by**(*rule subprob-space.subprob-not-empty*)
moreover have *sets ?c* = *sets N* **by**(*simp add*: *null-measure-def*)(*simp add*: *sets*)
ultimately have *c*: *?c* ∈ *space* (*subprob-algebra N*)
by(*simp add*: *space-subprob-algebra subprob-space-null-measure*)
from *f A c* **have** *f'*: *?f* ∈ *measurable M* (*subprob-algebra N*)
by(*simp add*: *measurable-restrict-space-iff*)

from *A* **have** [*simp*]: *space M* ≠ {} **by** *blast*

have *?lhs* = *join* (*distr* (*restrict-space M A*) (*subprob-algebra N*) *f*)
using *assms* **by**(*simp add*: *space-restrict-space bind-nonempty''*)
also have ... = *join* (*distr M* (*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 ... = *?rhs* **using** *f'* **by**(*simp add*: *bind-nonempty''*)
finally show *?thesis* **.**

qed

lemma *bind-const'*: ⟦*prob-space M*; *subprob-space N*⟧ ⟹ *M* ⤜ (λ*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*:
space M ≠ {} ⟹ *f* ∈ *measurable M N* ⟹ *bind M* (*return N* ∘ *f*) = *distr M N f*
apply (*simp add*: *bind-nonempty*)
apply (*subst subprob-algebra-cong*)
apply (*rule sets-return*)
apply (*subst distr-distr*[*symmetric*])
apply (*auto intro*!: *return-measurable simp*: *distr-distr*[*symmetric*] *join-return'*)
done

lemma *bind-return-distr'*:
space M ≠ {} ⟹ *f* ∈ *measurable M N* ⟹ *bind M* (λ*x*. *return N* (*f x*)) = *distr M N f*
using *bind-return-distr*[*of M f N*] **by** (*simp add*: *comp-def*)

lemma *bind-assoc*:
fixes *f* :: *'a* ⇒ *'b measure* **and** *g* :: *'b* ⇒ *'c measure*
assumes *M1*: *f* ∈ *measurable M* (*subprob-algebra N*) **and** *M2*: *g* ∈ *measurable N* (*subprob-algebra R*)
shows *bind* (*bind M f*) *g* = *bind M* (λ*x*. *bind* (*f x*) *g*)
proof (*cases space M* = {})
assume [*simp*]: *space M* ≠ {}
from *assms* **have** [*simp*]: *space N* ≠ {} *space R* ≠ {}
by (*auto simp*: *measurable-empty-iff space-subprob-algebra-empty-iff*)
from *assms* **have** *sets-fx*[*simp*]: ⋀*x*. *x* ∈ *space M* ⟹ *sets* (*f x*) = *sets N*
by (*simp add*: *sets-kernel*)
have *ex-in*: ⋀*A*. *A* ≠ {} ⟹ ∃ *x*. *x* ∈ *A* **by** *blast*
note *sets-some*[*simp*] = *sets-kernel*[*OF M1 someI-ex*[*OF ex-in*[*OF* ‹*space M* ≠ {}›]]]
sets-kernel[*OF M2 someI-ex*[*OF ex-in*[*OF* ‹*space N* ≠ {}›]]]
note *space-some*[*simp*] = *sets-eq-imp-space-eq*[*OF this*(*1*)] *sets-eq-imp-space-eq*[*OF this*(*2*)]

have *bind M* (λ*x*. *bind* (*f x*) *g*) =
join (*distr M* (*subprob-algebra R*) (*join* ∘ (λ*x*. (*distr x* (*subprob-algebra R*) *g*)) ∘ *f*))
by (*simp add*: *sets-eq-imp-space-eq*[*OF sets-fx*] *bind-nonempty o-def*
cong: *subprob-algebra-cong distr-cong*)
also have *distr M* (*subprob-algebra R*) (*join* ∘ (λ*x*. (*distr x* (*subprob-algebra R*) *g*)) ∘ *f*) =
distr (*distr* (*distr M* (*subprob-algebra N*) *f*)
(*subprob-algebra* (*subprob-algebra R*))
(λ*x*. *distr x* (*subprob-algebra R*) *g*))

```
            (subprob-algebra R) join
    apply (subst distr-distr,
           (blast intro: measurable-comp measurable-distr measurable-join M1
M2)+)+
    apply (simp add: o-assoc)
    done
  also have join ... = bind (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 ∈ measurable N (subprob-algebra N')
  assumes Mf: f ∈ measurable M (subprob-algebra M')
  assumes Mh: case-prod h ∈ measurable (M ⨂M M') N
  shows do {x ← M; y ← f x; g (h x y)} = do {x ← M; y ← f x; return N (h x
y)} ⤜ g
proof−
  have do {x ← M; y ← f x; return N (h x y)} ⤜ g =
        do {x ← M; do {y ← f x; return N (h x y)} ⤜ g}
    using Mh by (auto intro!: bind-assoc measurable-bind'[OF Mf] Mf Mg
          measurable-compose[OF - return-measurable] simp: measurable-split-conv)
  also from Mh have ⋀x. x ∈ space M ⟹ h x ∈ measurable M' N by measurable
  hence do {x ← M; do {y ← f x; return N (h x y)} ⤜ g} =
        do {x ← M; y ← f x; return N (h x y) ⤜ g}
    apply (intro ballI bind-cong 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 ⋀x. x ∈ space M ⟹ space (f x) = space M'
    by (intro sets-eq-imp-space-eq sets-kernel[OF Mf])
  with measurable-space[OF Mh]
    have do {x ← M; y ← f x; return N (h x y) ⤜ g} = do {x ← M; y ← 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 ∈ space (subprob-algebra
M)
  by (simp add: space-subprob-algebra) unfold-locales

lemma (in pair-prob-space) pair-measure-eq-bind:
  (M1 ⨂M M2) = (M1 ⤜ (λx. M2 ⤜ (λy. return (M1 ⨂M M2) (x, y))))
proof (rule measure-eqI)
  have ps-M2: prob-space M2 by unfold-locales
  note return-measurable[measurable]
  show sets (M1 ⨂M M2) = sets (M1 ⤜ (λx. M2 ⤜ (λy. return (M1 ⨂M
M2) (x, y))))
```

by (*simp-all add*: *M1.not-empty M2.not-empty*)
fix *A* **assume** [*measurable*]: *A* ∈ *sets* (*M1* ⨂$_M$ *M2*)
show *emeasure* (*M1* ⨂$_M$ *M2*) *A* = *emeasure* (*M1* ⤜ (λ*x*. *M2* ⤜ (λ*y*. *return* (*M1* ⨂$_M$ *M2*) (*x*, *y*)))) *A*
by (*auto simp*: *M2.emeasure-pair-measure M1.not-empty M2.not-empty emeasure-bind*[**where** *N*=*M1* ⨂$_M$ *M2*]
intro!: *nn-integral-cong*)
qed

lemma (**in** *pair-prob-space*) *bind-rotate*:
assumes *C*[*measurable*]: (λ(*x*, *y*). *C x y*) ∈ *measurable* (*M1* ⨂$_M$ *M2*) (*subprob-algebra N*)
shows (*M1* ⤜ (λ*x*. *M2* ⤜ (λ*y*. *C x y*))) = (*M2* ⤜ (λ*y*. *M1* ⤜ (λ*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* ∈ *space* (*subprob-algebra M1*) *M2* ∈ *space* (*subprob-algebra M2*)
by (*auto simp*: *space-subprob-algebra*) *unfold-locales*
have (*M1* ⤜ (λ*x*. *M2* ⤜ (λ*y*. *C x y*))) =
(*M1* ⤜ (λ*x*. *M2* ⤜ (λ*y*. *return* (*M1* ⨂$_M$ *M2*) (*x*, *y*)))) ⤜ (λ(*x*, *y*). *C x y*)
by (*auto intro*!: *bind-cong simp*: *bind-return*[**where** *N*=*N*] *space-pair-measure bind-assoc*[**where** *N*=*M1* ⨂$_M$ *M2* **and** *R*=*N*])
also have ... = (*distr* (*M2* ⨂$_M$ *M1*) (*M1* ⨂$_M$ *M2*) (λ(*x*, *y*). (*y*, *x*))) ⤜ (λ(*x*, *y*). *C x y*)
unfolding *pair-measure-eq-bind*[*symmetric*] *distr-pair-swap*[*symmetric*] ..
also have ... = (*M2* ⤜ (λ*x*. *M1* ⤜ (λ*y*. *return* (*M2* ⨂$_M$ *M1*) (*x*, *y*)))) ⤜ (λ(*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 ... = (*M2* ⤜ (λ*y*. *M1* ⤜ (λ*x*. *C x y*)))
by (*auto intro*!: *bind-cong simp*: *bind-return*[**where** *N*=*N*] *space-pair-measure bind-assoc*[**where** *N*=*M2* ⨂$_M$ *M1* **and** *R*=*N*])
finally show *?thesis* .
qed

21 Measures form a ω-chain complete partial order

definition *SUP-measure* :: (*nat* ⇒ *'a measure*) ⇒ *'a measure* **where**
SUP-measure M = *measure-of* (⋃*i*. *space* (*M i*)) (⋃*i*. *sets* (*M i*)) (λ*A*. *SUP i*. *emeasure* (*M i*) *A*)

lemma
assumes *const*: ⋀*i j*. *sets* (*M i*) = *sets* (*M j*)

```
  shows space-SUP-measure: space (SUP-measure M) = space (M i) (is ?sp)
    and sets-SUP-measure: sets (SUP-measure M) = sets (M i) (is ?st)
proof −
  have (⋃ i. sets (M i)) = sets (M i)
    using const by auto
  moreover have (⋃ i. space (M i)) = space (M i)
    using const[THEN sets-eq-imp-space-eq] by auto
  moreover have ⋀ i. sets (M i) ⊆ Pow (space (M i))
    by (auto dest: sets.sets-into-space)
  ultimately show ?sp ?st
    by (simp-all add: SUP-measure-def)
qed

lemma emeasure-SUP-measure:
  assumes const: ⋀ i j. sets (M i) = sets (M j)
    and mono: mono (λi. emeasure (M i))
  shows emeasure (SUP-measure M) A = (SUP i. emeasure (M i) A)
proof cases
  assume A ∈ sets (SUP-measure M)
  show ?thesis
  proof (rule emeasure-measure-of[OF SUP-measure-def])
    show countably-additive (sets (SUP-measure M)) (λA. SUP i. emeasure (M i)
A)
    proof (rule countably-additiveI)
      fix A :: nat ⇒ 'a set assume range A ⊆ sets (SUP-measure M)
      then have ⋀ i j. A i ∈ sets (M j)
        using sets-SUP-measure[of M, OF const] by simp
      moreover assume disjoint-family A
      ultimately show (∑ i. SUP ia. emeasure (M ia) (A i)) = (SUP i. emeasure
(M i) (⋃ i. A i))
        using suminf-SUP-eq
          using mono by (subst ennreal-suminf-SUP-eq) (auto simp: mono-def
le-fun-def intro!: SUP-cong suminf-emeasure)
    qed
    show positive (sets (SUP-measure M)) (λA. SUP i. emeasure (M i) A)
      by (auto simp: positive-def intro: SUP-upper2)
    show (⋃ i. sets (M i)) ⊆ Pow (⋃ i. space (M i))
      using sets.sets-into-space by auto
  qed fact
next
  assume A ∉ sets (SUP-measure M)
  with sets-SUP-measure[of M, OF const] show ?thesis
    by (simp add: emeasure-notin-sets)
qed

lemma bind-return'': sets M = sets N ⟹ M ⤜ 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 ∈ *space N* ⟹ *distr M N* (λ*x*. *c*) = *return N c*
by (*rule measure-eqI*) (*auto simp*: *emeasure-distr emeasure-space-1*)

lemma *return-count-space-eq-density*:
return (*count-space M*) *x* = *density* (*count-space M*) (*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*]:
null-measure M ∈ *space* (*subprob-algebra M*) ⟷ *space M* ≠ {}
by(*simp add*: *space-subprob-algebra subprob-space-null-measure-iff*)

end

22 Projective Family

theory *Projective-Family*
imports *Finite-Product-Measure Giry-Monad*
begin

lemma *vimage-restrict-preseve-mono*:
assumes *J*: *J* ⊆ *I*
and *sets*: *A* ⊆ (Π$_E$ *i*∈*J*. *S i*) *B* ⊆ (Π$_E$ *i*∈*J*. *S i*) **and** *ne*: (Π$_E$ *i*∈*I*. *S i*) ≠ {}
and *eq*: (λ*x*. *restrict x J*) −‘ *A* ∩ (Π$_E$ *i*∈*I*. *S i*) ⊆ (λ*x*. *restrict x J*) −‘ *B* ∩ (Π$_E$ *i*∈*I*. *S i*)
shows *A* ⊆ *B*
proof (*intro subsetI*)
fix *x* **assume** *x* ∈ *A*
from *ne* **obtain** *y* **where** *y*: ⋀*i*. *i* ∈ *I* ⟹ *y i* ∈ *S i* **by** *auto*
have *J* ∩ (*I* − *J*) = {} **by** *auto*
show *x* ∈ *B*
proof *cases*
assume *x*: *x* ∈ (Π$_E$ *i*∈*J*. *S i*)
have *merge J* (*I* − *J*) (*x*,*y*) ∈ (λ*x*. *restrict x J*) −‘ *A* ∩ (Π$_E$ *i*∈*I*. *S i*)
using *y x* ‹*J* ⊆ *I*› *PiE-cancel-merge*[*of J I* − *J x y S*] ‹*x*∈*A*›
by (*auto simp del*: *PiE-cancel-merge simp add*: *Un-absorb1*)
also have ... ⊆ (λ*x*. *restrict x J*) −‘ *B* ∩ (Π$_E$ *i*∈*I*. *S i*) **by** *fact*
finally show *x* ∈ *B*
using *y x* ‹*J* ⊆ *I*› *PiE-cancel-merge*[*of J I* − *J x y S*] ‹*x*∈*A*› *eq*
by (*auto simp del*: *PiE-cancel-merge simp add*: *Un-absorb1*)
qed (*insert* ‹*x*∈*A*› *sets*, *auto*)
qed

locale *projective-family* =
fixes *I* :: *'i set* **and** *P* :: *'i set* ⇒ (*'i* ⇒ *'a*) *measure* **and** *M* :: *'i* ⇒ *'a measure*
assumes *P*: ⋀*J H*. *J* ⊆ *H* ⟹ *finite H* ⟹ *H* ⊆ *I* ⟹ *P J* = *distr* (*P H*) (*PiM J M*) (λ*f*. *restrict f J*)

assumes *prob-space-P*: $\bigwedge J.$ *finite* $J \Longrightarrow J \subseteq I \Longrightarrow$ *prob-space* $(P\ J)$
begin

lemma *sets-P*: *finite* $J \Longrightarrow J \subseteq I \Longrightarrow$ *sets* $(P\ J) =$ *sets* $(PiM\ J\ M)$
by (*subst P*[*of J J*]) *simp-all*

lemma *space-P*: *finite* $J \Longrightarrow J \subseteq I \Longrightarrow$ *space* $(P\ J) =$ *space* $(PiM\ J\ M)$
using *sets-P* **by** (*rule sets-eq-imp-space-eq*)

lemma *not-empty-M*: $i \in I \Longrightarrow$ *space* $(M\ i) \neq \{\}$
using *prob-space-P*[*THEN prob-space.not-empty*] **by** (*auto simp*: *space-PiM space-P*)

lemma *not-empty*: *space* $(PiM\ I\ M) \neq \{\}$
by (*simp add*: *not-empty-M*)

abbreviation
$emb\ L\ K \equiv$ *prod-emb* $L\ M\ K$

lemma *emb-preserve-mono*:
assumes $J \subseteq L\ L \subseteq I$ **and** *sets*: $X \in$ *sets* $(Pi_M\ J\ M)\ Y \in$ *sets* $(Pi_M\ J\ M)$
assumes *emb* $L\ J\ X \subseteq$ *emb* $L\ J\ Y$
shows $X \subseteq Y$
proof (*rule vimage-restrict-preseve-mono*)
show $X \subseteq (\Pi_E\ i{\in}J.$ *space* $(M\ i))\ Y \subseteq (\Pi_E\ i{\in}J.$ *space* $(M\ i))$
using *sets*[*THEN sets.sets-into-space*] **by** (*auto simp*: *space-PiM*)
show $(\Pi_E\ i{\in}L.$ *space* $(M\ i)) \neq \{\}$
using ‹$L \subseteq I$› **by** (*auto simp add*: *not-empty-M space-PiM*[*symmetric*])
show $(\lambda x.$ *restrict* $x\ J) -`` X \cap (\Pi_E\ i{\in}L.$ *space* $(M\ i)) \subseteq (\lambda x.$ *restrict* $x\ J) -``$
$Y \cap (\Pi_E\ i{\in}L.$ *space* $(M\ i))$
using ‹*prod-emb L M J X* ⊆ *prod-emb L M J Y*› **by** (*simp add*: *prod-emb-def*)
qed *fact*

lemma *emb-injective*:
assumes *L*: $J \subseteq L\ L \subseteq I$ **and** *X*: $X \in$ *sets* $(Pi_M\ J\ M)$ **and** *Y*: $Y \in$ *sets* $(Pi_M$
$J\ M)$
shows *emb* $L\ J\ X =$ *emb* $L\ J\ Y \Longrightarrow X = Y$
by (*intro antisym emb-preserve-mono*[*OF L X Y*] *emb-preserve-mono*[*OF L Y*
X]) *auto*

lemma *emeasure-P*: $J \subseteq K \Longrightarrow$ *finite* $K \Longrightarrow K \subseteq I \Longrightarrow X \in$ *sets* $(PiM\ J\ M)$
$\Longrightarrow P\ K$ $(emb\ K\ J\ X) = P\ J\ X$
by (*auto intro*!: *emeasure-distr-restrict*[*symmetric*] *simp*: *P sets-P*)

inductive-set *generator* :: $('i \Rightarrow 'a)$ *set set* **where**
finite $J \Longrightarrow J \subseteq I \Longrightarrow X \in$ *sets* $(Pi_M\ J\ M) \Longrightarrow$ *emb* $I\ J\ X \in$ *generator*

lemma *algebra-generator*: *algebra* $($*space* $(PiM\ I\ M))$ *generator*
unfolding *algebra-iff-Int*
proof (*safe elim*!: *generator.cases*)

fix J X **assume** J: *finite* J $J \subseteq I$ **and** X: $X \in$ *sets* (*PiM* J M)

from X[*THEN sets.sets-into-space*] J **show** $x \in$ *emb* I J $X \Longrightarrow x \in$ *space* (*PiM* 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 $\{i\}$ (Π_E $j \in \{i\}$. A)
by (*auto simp*: *prod-emb-def space-PiM restrict-def Pi-iff PiE-iff extensional-def*)
with ‹$i \in I$› 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* (μG) **where**
μG 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* (λx. x) μG

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* ⊆ *I X* ∈ *sets* (*Pi*$_M$ *J M*)
shows *μG* (*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* ⊆ *I Y* ∈ *sets* (*Pi*$_M$ *K M*)
and [*simp*]: *emb I J X* = *emb I K Y*
have *emeasure* (*P K*) *Y* = *emeasure* (*P* (*K* ∪ *J*)) (*emb* (*K* ∪ *J*) *K Y*)
using *K J* **by** (*simp add*: *emeasure-P*)
also have *emb* (*K* ∪ *J*) *K Y* = *emb* (*K* ∪ *J*) *J X*
using *K J* **by** (*simp add*: *emb-injective*[*of K* ∪ *J I*])
also have *emeasure* (*P* (*K* ∪ *J*)) (*emb* (*K* ∪ *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* ⊆ *I* **by** *auto*
then have *μG* (*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* ⊆ *I X* ∈ *sets* (*PiM J M*)
and *K*: *finite K K* ⊆ *I Y* ∈ *sets* (*PiM K M*)
and *emb I J X* ∩ *emb I K Y* = {}
then have *JK-disj*: *emb* (*J* ∪ *K*) *J X* ∩ *emb* (*J* ∪ *K*) *K Y* = {}
by (*intro emb-injective*[*of J* ∪ *K I* - {}]) (*auto simp*: *sets.Int prod-emb-Int*)
have *μG* (*emb I J X* ∪ *emb I K Y*) = *μG* (*emb I* (*J* ∪ *K*) (*emb* (*J* ∪ *K*) *J X*
∪ *emb* (*J* ∪ *K*) *K Y*))
using *J K* **by** *simp*
also have ... = *emeasure* (*P* (*J* ∪ *K*)) (*emb* (*J* ∪ *K*) *J X* ∪ *emb* (*J* ∪ *K*) *K*
Y)
using *J K* **by** (*simp add*: *mu-G-spec sets.Un del*: *prod-emb-Un*)
also have ... = *μG* (*emb I J X*) + *μG* (*emb I K Y*)
using *J K JK-disj* **by** (*simp add*: *plus-emeasure*[*symmetric*] *mu-G-spec emeasure-P*
sets-P)
finally show *μG* (*emb I J X* ∪ *emb I K Y*) = *μG* (*emb I J X*) + *μG* (*emb I*
K Y) **.**
qed

lemma *emeasure-lim*:
assumes *JX*: *finite J* $J \subseteq I$ $X \in$ *sets* (*PiM J M*)
assumes *cont*: $\bigwedge J\ X.\ (\bigwedge i.\ J\ i \subseteq I) \Longrightarrow$ *incseq J* $\Longrightarrow$ $(\bigwedge i.$ *finite* $(J\ i)) \Longrightarrow (\bigwedge i.$ $X\ i \in$ *sets* (*PiM* $(J\ i)$ *M*)) $\Longrightarrow$
decseq $(\lambda i.$ *emb I* $(J\ i)$ $(X\ i)) \Longrightarrow 0 <$ (*INF i. P* $(J\ i)$ $(X\ i)) \Longrightarrow (\bigcap i.$ *emb I* $(J\ i)$ $(X\ i)) \neq \{\}$
shows *emeasure lim* (*emb I J X*) = *P J X*
proof −
have $\exists \mu.\ (\forall\, s{\in}$*generator*. $\mu\ s = \mu G\ s) \wedge$
measure-space (*space* (*PiM I M*)) (*sigma-sets* (*space* (*PiM I M*)) *generator*) μ
proof (*rule generator.caratheodory-empty-continuous*[*OF positive-mu-G additive-mu-G*])
show $\bigwedge A.\ A \in$ *generator* $\Longrightarrow \mu G\ A \neq \infty$
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 $\bigwedge X.\ X \in$ *sets* (Pi_M *J M*) $\Longrightarrow$ *emeasure* (*P J*) *X* $\neq$ *top*
by *simp*
qed
next
fix *A* **assume** *range A* $\subseteq$ *generator* **and** *decseq A* $(\bigcap i.\ A\ i) = \{\}$
then have $\forall i.\ \exists J\ X.\ A\ i =$ *emb I J X* $\wedge$ *finite J* $\wedge$ $J \subseteq I \wedge X \in$ *sets* (*PiM J M*)
unfolding *image-subset-iff generator.simps* **by** *blast*
then obtain *J X* **where** *A*: $\bigwedge i.\ A\ i =$ *emb I* $(J\ i)$ $(X\ i)$
and ∗: $\bigwedge i.$ *finite* $(J\ i)$ $\bigwedge i.\ J\ i \subseteq I$ $\bigwedge i.\ X\ i \in$ *sets* (*PiM* $(J\ i)$ *M*)
by *metis*
have (*INF i. P* $(J\ i)$ $(X\ i)) = 0$
proof (*rule ccontr*)
assume *INF-P*: (*INF i. P* $(J\ i)$ $(X\ i)) \neq 0$
have $(\bigcap i.$ *emb I* $(\bigcup n{\leq}i.\ J\ n)$ (*emb* $(\bigcup n{\leq}i.\ J\ n)$ $(J\ i)$ $(X\ i))) \neq \{\}$
proof (*rule cont*)
show *decseq* $(\lambda i.$ *emb I* $(\bigcup n{\leq}i.\ J\ n)$ (*emb* $(\bigcup n{\leq}i.\ J\ n)$ $(J\ i)$ $(X\ i)))$
using ∗ ‹*decseq A*› **by** (*subst prod-emb-trans*) (*auto simp*: *A*[*abs-def*])
show *0* < (*INF i. P* $(\bigcup n{\leq}i.\ J\ n)$ (*emb* $(\bigcup n{\leq}i.\ J\ n)$ $(J\ i)$ $(X\ i)))$
using ∗ *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* ∗, *auto*)
with ‹$(\bigcap i.\ A\ i) = \{\}$› ∗ **show** *False*
by (*subst* (*asm*) *prod-emb-trans*) (*auto simp*: *A*[*abs-def*])
qed
moreover have $(\lambda i.\ P\ (J\ i)\ (X\ i)) \longrightarrow$ (*INF i. P* $(J\ i)$ $(X\ i))$
proof (*intro LIMSEQ-INF antimonoI*)
fix *x y* :: *nat* **assume** $x \leq y$
have *P* $(J\ y \cup J\ x)$ (*emb* $(J\ y \cup J\ x)$ $(J\ y)$ $(X\ y)) \leq P\ (J\ y \cup J\ x)$ (*emb* $(J\ y \cup J\ x)$ $(J\ x)$ $(X\ x))$
using ‹*decseq A*›[*THEN decseqD*, *OF* ‹$x{\leq}y$›] ∗

```
      by (auto simp: A sets-P del: subsetI intro!: emeasure-mono ‹x ≤ y›
            emb-preserve-mono[of J y ∪ J x I, where X=emb (J y ∪ J x) (J y)
(X y)])
     then show P (J y) (X y) ≤ P (J x) (X x)
       using ∗ by (simp add: emeasure-P)
   qed
   ultimately show (λi. μG (A i)) ⟶ 0
     by (auto simp: A[abs-def] mu-G-spec ∗)
 qed
 then obtain μ where eq: ∀ s∈generator. μ s = μG s
   and ms: measure-space (space (PiM I M)) (sets (PiM I M)) μ
   by (metis sets-PiM-generator)
 show ?thesis
 proof (subst emeasure-extend-measure[OF lim-def])
   show A ∈ generator ⟹ μ A = μG A for A
     using eq by simp
   show positive (sets lim) μ countably-additive (sets lim) μ
     using ms by (auto simp add: measure-space-def)
   show (λx. x) ‘ generator ⊆ Pow (space (Pi_M I M))
     using generator.space-closed by simp
   show emb I J X ∈ generator μG (emb I J X) = emeasure (P J) X
     using JX by (auto intro: generator.intros simp: mu-G-spec)
 qed
qed

end

sublocale product-prob-space ⊆ projective-family I λJ. PiM J M M
 unfolding projective-family-def
proof (intro conjI allI impI distr-restrict)
 show ⋀J. finite J ⟹ prob-space (Pi_M 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 :: nat ⇒ (nat ⇒ 'a) ⇒ 'a measure and M :: nat ⇒ 'a measure
 assumes P[measurable]: ⋀i. P i ∈ measurable (PiM {0..<i} M) (subprob-algebra
(M i))
 assumes prob-space-P: ⋀i x. x ∈ space (PiM {0..<i} M) ⟹ prob-space (P i
x)
begin

lemma non-empty[simp]: space (M i) ≠ {}
proof (induction i rule: less-induct)
 case (less i)
 then obtain x where ⋀j. j < i ⟹ x j ∈ space (M j)
   unfolding ex-in-conv[symmetric] by metis
 then have ∗: restrict x {0..<i} ∈ 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*) ≠ {}
unfolding *space-PiM-empty-iff* **by** *auto*

lemma *space-P*: *x* ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *space* (*P n x*) = *space* (*M n*)
by (*simp add*: *P*[*THEN subprob-measurableD*(*1*)])

lemma *sets-P*[*measurable-cong*]: *x* ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *sets* (*P n x*) = *sets* (*M n*)
by (*simp add*: *P*[*THEN subprob-measurableD*(*2*)])

definition *eP* :: *nat* ⇒ (*nat* ⇒ *'a*) ⇒ (*nat* ⇒ *'a*) *measure* **where**
eP n ω = *distr* (*P n ω*) (*PiM* {*0..<Suc n*} *M*) (*fun-upd ω n*)

lemma *measurable-eP*[*measurable*]:
eP n ∈ *measurable* (*PiM* {*0..< n*} *M*) (*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 ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *space* (*eP n x*) = *space* (*PiM* {*0..<Suc n*} *M*)
by (*simp add*: *measurable-eP*[*THEN subprob-measurableD*(*1*)])

lemma *sets-eP*[*measurable*]:
x ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *sets* (*eP n x*) = *sets* (*PiM* {*0..<Suc n*} *M*)
by (*simp add*: *measurable-eP*[*THEN subprob-measurableD*(*2*)])

lemma *prob-space-eP*: *x* ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *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*:
ω ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *f* ∈ *borel-measurable* (*PiM* {*0..<Suc n*} *M*)
⟹
(∫⁺*x*. *f x* ∂*eP n ω*) = (∫⁺*x*. *f* (*ω*(*n* := *x*)) ∂*P n ω*)
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*]: *ω* ∈ *space* (*PiM* {*0..<n*} *M*) **and** *A*[*measurable*]: *A* ∈ *sets*
(*PiM* {*0..<Suc n*} *M*)

shows *eP n ω A = P n ω ((λx. ω(n := x)) −‘ A ∩ space (M n))*
using *nn-integral-eP*[*of ω 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 ω*] *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* ⇒ *nat* ⇒ (*nat* ⇒ *'a*) ⇒ (*nat* ⇒ *'a*) *measure* **where**
C n 0 ω = return (*PiM* {*0..<n*} *M*) *ω*
| *C n* (*Suc m*) *ω = C n m ω* ⤜ *eP* (*n* + *m*)

lemma *measurable-C*[*measurable*]:
C n m ∈ *measurable* (*PiM* {*0..<n*} *M*) (*subprob-algebra* (*PiM* {*0..<n* + *m*} *M*))
by (*induction m*) *auto*

lemma *space-C*:
x ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *space* (*C n m x*) = *space* (*PiM* {*0..<n* + *m*} *M*)
by (*simp add*: *measurable-C*[*THEN subprob-measurableD*(*1*)])

lemma *sets-C*[*measurable-cong*]:
x ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *sets* (*C n m x*) = *sets* (*PiM* {*0..<n* + *m*} *M*)
by (*simp add*: *measurable-C*[*THEN subprob-measurableD*(*2*)])

lemma *prob-space-C*: *x* ∈ *space* (*PiM* {*0..<n*} *M*) ⟹ *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..<Suc* (*n* + *m*)} *M*]
simp: *space-C prob-space-eP*)
qed (*auto intro*!: *prob-space-return simp*: *space-PiM*)

lemma *split-C*:
assumes *ω*: *ω* ∈ *space* (*PiM* {*0..<n*} *M*) **shows** (*C n m ω* ⤜ *C* (*n* + *m*) *l*) = *C n* (*m* + *l*) *ω*
proof (*induction l*)
case *0*
with *ω* **show** *?case*
by (*simp add*: *bind-return-distr' prob-space-C*[*THEN prob-space.not-empty*] *distr-cong*[*OF refl sets-C*[*symmetric*, *OF ω*]])
next
case (*Suc l*) **with** *ω* **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*[*measurable*]: $f \in$ *borel-measurable* (*PiM* {*0*..<*n*+*m*} *M*)
and *nonneg*: $\bigwedge x.\ x \in$ *space* (*PiM* {*0*..<*n*+*m*} *M*) $\Longrightarrow$ $0 \leq f\ x$
and *x*: $x \in$ *space* (*PiM* {*0*..<*n*} *M*)
shows $(\int^+ x.\ f\ x\ \partial C\ n\ m\ x) = (\int^+ x.\ f\ (\textit{restrict}\ x\ \{0..<n+m\})\ \partial C\ n\ m'\ x)$
using ‹$m \leq m'$›
proof (*induction rule*: *dec-induct*)
case (*step i*)
let *?E* = $\lambda x.\ f\ (\textit{restrict}\ x\ \{0..<n+m\})$ **and** *?C* = $\lambda i\ f.\ \int^+ x.\ f\ x\ \partial C\ n\ i\ x$
from ‹$m \leq i$› *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*[*measurable*]: $A \in$ *sets* (*PiM* {*0*..<*n*+*m*} *M*) **and** [*simp*]: $x \in$ *space* (*PiM* {*0*..<*n*} *M*)
shows *emeasure* (*C n m' x*) (*prod-emb* {*0*..<*n* + *m'*} *M* {*0*..<*n*+*m*} *A*) = *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*]: $x \in$ *space* (*PiM* {*0*..<*n*} *M*)
shows *C n m x* = *distr* (*C n m' x*) (*PiM* {*0*..<*n*+*m*} *M*) ($\lambda x.$ *restrict x* {*0*..<*n*+*m*})
proof (*rule measure-eqI*)
fix *A* **assume** *A* ∈ *sets* (*C n m x*)
with ‹$m \leq m'$› **show** *emeasure* (*C n m x*) *A* = *emeasure* (*distr* (*C n m' x*) (Pi_M {*0*..<*n* + *m*} *M*) ($\lambda x.$ *restrict x* {*0*..<*n* + *m*})) *A*
by (*subst emeasure-C*[*symmetric*, *OF* ‹$m \leq m'$›]) (*auto intro*!: *emeasure-distr-restrict*[*symmetric*] *simp*: *sets-C*)
qed (*simp add*: *sets-C*)

definition *up-to* :: *nat set* ⇒ *nat* **where**
up-to J = (*LEAST n*. ∀ *i*≥*n*. *i* ∉ *J*)

lemma *up-to-less*: *finite J* $\Longrightarrow$ $i \in J \Longrightarrow i <$ *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*: *finite J* $\Longrightarrow$ *up-to J* $\leq n \longleftrightarrow (\forall\, i{\in}J.\ i < n)$
proof *safe*
show *finite J* $\Longrightarrow$ *up-to J* $\leq n \Longrightarrow i \in J \Longrightarrow i < n$ **for** *i*
using *up-to-less*[*of J i*] **by** *auto*
qed (*auto simp*: *up-to-def intro*!: *Least-le*)

lemma *up-to-iff-Ico*: *finite J* $\Longrightarrow$ *up-to J* $\leq n \longleftrightarrow J \subseteq \{0..{<}n\}$
by (*auto simp*: *up-to-iff*)

lemma *up-to*: *finite J* $\Longrightarrow J \subseteq \{0..{<}$ *up-to J*$\}$
by (*auto simp*: *up-to-less*)

lemma *up-to-mono*: $J \subseteq H \Longrightarrow$ *finite H* $\Longrightarrow$ *up-to J* $\leq$ *up-to H*
by (*auto simp add*: *up-to-iff finite-subset up-to-less*)

definition *CI* :: *nat set* $\Rightarrow$ (*nat* $\Rightarrow$ *'a*) *measure* **where**
CI J = *distr* (*C 0* (*up-to J*) ($\lambda x.$ *undefined*)) (*PiM J M*) ($\lambda f.$ *restrict f J*)

sublocale *PF*: *projective-family UNIV CI*
unfolding *projective-family-def*
proof *safe*
show *finite J* $\Longrightarrow$ *prob-space* (*CI J*) **for** *J*
using *up-to*[*of J*] **unfolding** *CI-def*
by (*intro prob-space.prob-space-distr prob-space-C measurable-restrict*) *auto*
note *measurable-cong-sets*[*OF sets-C*, *simp*]
have [*simp*]: $J \subseteq H \Longrightarrow (\lambda f.$ *restrict f J*$) \in$ *measurable* (Pi_M *H M*) (Pi_M *J M*)
for *H J*
by (*auto intro*!: *measurable-restrict*)

show $J \subseteq H \Longrightarrow$ *finite H* $\Longrightarrow$ *CI J* = *distr* (*CI H*) (Pi_M *J M*) ($\lambda f.$ *restrict f J*) **for** *J H*
by (*simp add*: *CI-def distr-C*[*OF up-to-mono*[*of J H*]] *up-to up-to-mono distr-distr comp-def*
inf.absorb2 finite-subset)
qed

lemma *emeasure-CI'*:
finite J $\Longrightarrow X \in$ *sets* (*PiM J M*) $\Longrightarrow$ *CI J X* = *C 0* (*up-to J*) (λ-. *undefined*) (*PF.emb* $\{0..{<}$*up-to J*$\}$ *J X*)
unfolding *CI-def* **using** *up-to*[*of J*] **by** (*rule emeasure-distr-restrict*) (*auto simp*: *sets-C*)

lemma *emeasure-CI*:
$J \subseteq \{0..{<}n\} \Longrightarrow X \in$ *sets* (*PiM J M*) $\Longrightarrow$ *CI J X* = *C 0 n* (λ-. *undefined*) (*PF.emb* $\{0..{<}n\}$ *J X*)
apply (*subst emeasure-CI'*, *simp-all add*: *finite-subset*)
apply (*subst emeasure-C*[*symmetric*, *of up-to J n*])
apply (*auto simp*: *finite-subset up-to-iff-Ico up-to-less*)

apply (*subst prod-emb-trans*)
apply (*auto simp*: *up-to-less finite-subset up-to-iff-Ico*)
done

lemma *lim*:
assumes *J*: *finite J* **and** *X*: $X \in sets\ (PiM\ J\ M)$
shows *emeasure PF.lim* (*PF.emb UNIV J X*) = *emeasure* (*CI J*) *X*
proof (*rule PF.emeasure-lim*[*OF J subset-UNIV X*])
fix *J X'* **assume** *J*[*simp*]: $\bigwedge i.$ *finite* (*J i*) **and** *X'*[*measurable*]: $\bigwedge i.\ X'\ i \in sets\ (Pi_M\ (J\ i)\ M)$
and *dec*: *decseq* ($\lambda i.$ *PF.emb UNIV* (*J i*) (*X' i*))
def $X \equiv \lambda n.\ (\bigcap i \in \{i.\ J\ i \subseteq \{0..<n\}\}.\ PF.emb\ \{0..<n\}\ (J\ i)\ (X'\ i)) \cap space\ (PiM\ \{0..<n\}\ M)$

have *sets-X*[*measurable*]: $X\ n \in sets\ (PiM\ \{0..<n\}\ M)$ **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*: $n \le m \Longrightarrow X\ m \subseteq PF.emb\ \{0..<m\}\ \{0..<n\}\ (X\ n)$ **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.emb* $\{0..<n\}$ (*J j*) (*X' j*) $\subseteq$ *PF.emb* $\{0..<n\}$ (*J i*) (*X' i*)
if [*simp*]: *J i* $\subseteq \{0..<n\}$ *J j* $\subseteq \{0..<n\}$ $i \le j$ **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 $\ldots \le$ (*INF i. C 0 i* ($\lambda x.$ *undefined*) (*X i*))
proof (*intro INF-greatest*)
fix *n*
interpret *C*: *prob-space C 0 n* ($\lambda x.$ *undefined*)
by (*rule prob-space-C*) *simp*
show (*INF i. CI* (*J i*) (*X' i*)) $\le$ *C 0 n* ($\lambda x.$ *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 $*$: $\{i.\ J\ i \subseteq \{0..<n\}\} \neq \{\}$
have (*INF i. CI* (*J i*) (*X' i*)) $\le$
(*INF i*:$\{i.\ J\ i \subseteq \{0..<n\}\}$. *C 0 n* ($\lambda$-. *undefined*) (*PF.emb* $\{0..<n\}$ (*J i*) (*X' i*)))
by (*intro INF-superset-mono*) (*auto simp*: *emeasure-CI*)
also have $\ldots$ = *C 0 n* (λ-. *undefined*) ($\bigcap i \in \{i.\ J\ i \subseteq \{0..<n\}\}.$ (*PF.emb* $\{0..<n\}$ (*J i*) (*X' i*)))
using $*$ **by** (*intro emeasure-INT-decseq-subset*[*symmetric*]) (*auto intro*!: *dec-X' del*: *subsetI simp*: *sets-C*)

```
    also have ... = C 0 n (λ-. undefined) (X n)
      using ∗ 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 (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': 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 (∫⁺x. (INF i. ?C' i x) ∂P n ω) ≠ 0
    by simp
  with M have ∃ F x in ae-filter (P n ω). 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 ∃ F x in ae-filter (P n ω). x ∈ space (M n) ∧ 0 < (INF i. ?C' i x)
    by (rule frequently-mp[rotated]) (auto simp: space-P ω)
  then obtain x where x ∈ 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 ∃ x. ω(n := x) ∈ X (Suc n) ∧ 0 < (INF i. ?C' i x)
    by (intro exI[of - x]) (simp split: split-indicator-asm) }
 note step = this

 let ?ω = λω n x. (restrict ω {0..<n})(n := x)
 let ?L = λω n r. INF i. emeasure (C (Suc n) i (?ω ω n r)) (X (Suc n + i))
 have ∗: (⋀i. i < n ⟹ ?ω ω i (ω i) ∈ X (Suc i)) ⟹
   restrict ω {0..<n} ∈ space (Pi_M {0..<n} M) for ω n
   using sets.sets-into-space[OF sets-X, of n]
   by (cases n) (auto simp: atLeastLessThanSuc restrict-def[of - {}])
 have ∃ω. ∀ n. ?ω ω n (ω n) ∈ X (Suc n) ∧ 0 < ?L ω n (ω n)
 proof (rule dependent-wellorder-choice)
   fix n ω assume IH: ⋀i. i < n ⟹ ?ω ω i (ω i) ∈ X (Suc i) ∧ 0 < ?L ω i
(ω i)
   show ∃ r. ?ω ω n r ∈ X (Suc n) ∧ 0 < ?L ω n r
   proof (rule step)
     show restrict ω {0..<n} ∈ space (Pi_M {0..<n} M)
       using IH[THEN conjunct1] by (rule ∗)
     show 0 < (INF i. emeasure (C n i (restrict ω {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 ω where ω: ⋀n. ?ω ω n (ω n) ∈ X (Suc n)
   by auto
 from this[THEN ∗] have ω-space: ω ∈ space (PiM UNIV M)
   by (auto simp: space-PiM PiE-iff Ball-def)
 have ∗: ω ∈ PF.emb UNIV {0..<n} (X n) for n
 proof (cases n)
   case 0 with ω-space ‹X 0 ≠ {}› 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 *ω*[*of i*] *ω-space* **by** (*auto simp*: *prod-emb-def space-PiM PiE-iff atLeast-LessThanSuc*)
qed
have ∗∗: {*i*. *J i* ⊆ {*0*..<*up-to* (*J n*)}} ≠ {} **for** *n*
by (*auto intro*!: *exI*[*of* - *n*] *up-to J*)
have *ω* ∈ *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 (⋂ *i*. *PF.emb UNIV* (*J i*) (*X*′ *i*)) ≠ {}
by *auto*
qed

lemma *distr-lim*: **assumes** *J*[*simp*]: *finite J* **shows** *distr PF.lim* (*PiM J M*) (λ*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* ⊆ *I X* ∈ *sets* (*PiM J M*)
shows *emeasure lim* (*emb I J X*) = *emeasure* (*Pi*$_M$ *J M*) *X*
proof (*rule emeasure-lim*[*OF* ∗], *goal-cases*)
case (*1 J X*)

have ∃ *Q*. (∀ *i*. *sets Q* = *PiM* (⋃ *i*. *J i*) *M* ∧ *distr Q* (*PiM* (*J i*) *M*) (λ*x*. *restrict x* (*J i*)) = *Pi*$_M$ (*J i*) *M*)
proof *cases*
assume *finite* (⋃ *i*. *J i*)
then have *distr* (*PiM* (⋃ *i*. *J i*) *M*) (*Pi*$_M$ (*J i*) *M*) (λ*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* (⋃ *i*. *J i*)
moreover have *count*: *countable* (⋃ *i*. *J i*)
using *1*(*3*) **by** (*auto intro*: *countable-finite*)
def *f* ≡ *from-nat-into* (⋃ *i*. *J i*) **and** *t* ≡ *to-nat-on* (⋃ *i*. *J i*)
have *ft*[*simp*]: *x* ∈ *J i* ⟹ *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 (⋃ 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 λi ω. M (f i) λi. M (f i)
    by standard auto
  interpret Mf: product-prob-space λx. M (f x) UNIV
    by standard
  have C-eq-PiM: IT.C 0 n (λ-. undefined) = PiM {0..<n} (λx. 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] emeasure-bind[OF
- IT.measurable-eP])
    apply (auto simp: Mf.product-nn-integral-insert nn-integral-indicator[symmetric]
atLeastLessThanSuc IT.emeasure-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 (λx. 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 (⋃ i. J i) M) (λω. λx∈⋃ i. J i. ω (t x))
  { fix i
    have distr ?Q (Pi_M (J i) M) (λx. restrict x (J i)) =
      distr IT.PF.lim (Pi_M (J i) M) ((λω. λn∈J i. ω (t n)) ∘ (λω. restrict ω (t‘J
i)))
    proof (subst distr-distr)
      have (λω. ω (t x)) ∈ measurable (Pi_M UNIV (λx. M (f x))) (M x) if x: x
∈ J i for x i
        using measurable-component-singleton[of t x UNIV λx. M (f x)] unfolding
ft[OF x] by simp
      then show (λω. λx∈⋃ i. J i. ω (t x)) ∈ measurable IT.PF.lim (Pi_M
(UNION UNIV J) 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) (λx. M (f x))) (λω.
restrict ω (t‘J i))) (Pi_M (J i) M) (λω. λn∈J i. ω (t n))
    proof (intro distr-distr[symmetric])
      have (λω. ω (t x)) ∈ measurable (Pi_M (t‘J i) (λx. M (f x))) (M x) if x: x
∈ J i for x
        using measurable-component-singleton[of t x t‘J i λx. M (f x)] x unfolding
ft[OF x] by auto
      then show (λω. λn∈J i. ω (t n)) ∈ measurable (Pi_M (t ‘ J i) (λx. M (f
x))) (Pi_M (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*) (λ*x*. *M* (*f x*))) (*Pi*$_M$ (*J i*) *M*) (λω. λ*n*∈*J i*. ω (*t n*))
using ‹*finite* (*J i*)› **by** (*subst IT.distr-lim*) (*auto simp*: *CI-eq-PiM*)
also have ... = *Pi*$_M$ (*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* (*Pi*$_M$ (*J i*) *M*) (λ*x*. *restrict x* (*J i*)) = *Pi*$_M$ (*J i*) *M* **.**
}
then show ∃ *Q*. ∀ *i*. *sets Q* = *PiM* (⋃ *i*. *J i*) *M* ∧ *distr Q* (*Pi*$_M$ (*J i*) *M*) (λ*x*. *restrict x* (*J i*)) = *Pi*$_M$ (*J i*) *M*
by (*intro exI*[*of* - *?Q*]) *auto*
qed
then obtain *Q* **where** *sets-Q*: *sets Q* = *PiM* (⋃ *i*. *J i*) *M*
and *Q*: ⋀*i*. *distr Q* (*PiM* (*J i*) *M*) (λ*x*. *restrict x* (*J i*)) = *Pi*$_M$ (*J i*) *M* **by** *blast*

from *1* **interpret** *Q*: *prob-space Q*
by (*intro prob-space-distrD*[*of* λ*x*. *restrict x* (*J 0*) *Q PiM* (*J 0*) *M*])
(*auto simp*: *Q measurable-cong-sets*[*OF sets-Q*]
intro!: *prob-space-P measurable-restrict measurable-component-singleton*)

have *0* < (*INF i*. *emeasure* (*Pi*$_M$ (*J i*) *M*) (*X i*)) **by** *fact*
also have ... = (*INF i*. *emeasure Q* (*emb* (⋃ *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* (⋂ *i*. *emb* (⋃ *i*. *J i*) (*J i*) (*X i*))
proof (*rule INF-emeasure-decseq*)
from *1* **show** *decseq* (λ*n*. *emb* (⋃ *i*. *J i*) (*J n*) (*X n*))
by (*intro antimonoI emb-preserve-mono*[**where** *X*=*emb* (⋃ *i*. *J i*) (*J n*) (*X n*) **and** *L*=*I* **and** *J*=⋃ *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 (⋂ *i*. *emb* (⋃ *i*. *J i*) (*J i*) (*X i*)) ≠ {}
by *auto*
moreover have (⋂ *i*. *emb I* (*J i*) (*X i*)) = {} ⟹ (⋂ *i*. *emb* (⋃ *i*. *J i*) (*J i*) (*X i*)) = {}
using *1* **by** (*intro emb-injective*[*of* ⋃ *i*. *J i I* - {}] *sets.countable-INT*) (*auto simp*: *SUP-least SUP-upper*)
ultimately show *?case* **by** *auto*
qed

end

23 Infinite Product Measure

theory *Infinite-Product-Measure*

imports *Probability-Measure Caratheodory Projective-Family*
begin

lemma (**in** *product-prob-space*) *distr-PiM-restrict-finite*:
assumes *finite J* $J \subseteq I$
shows *distr* (*PiM I M*) (*PiM J M*) (λx. *restrict x J*) = *PiM J M*
proof (*rule PiM-eqI*)
fix *X* **assume** *X*: $\bigwedge i.\ i \in J \Longrightarrow X\ i \in sets\ (M\ i)$
{ fix *J X* **assume** *J*: $J \neq \{\} \vee I = \{\}$ *finite J* $J \subseteq I$ **and** *X*: $\bigwedge i.\ i \in J \Longrightarrow X\ i \in sets\ (M\ i)$
have *emeasure* (*PiM I M*) (*emb I J* (*PiE J X*)) = ($\prod i \in J.\ M\ i\ (X\ i)$)
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*) $\in$ *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* (*PiE J X*)) = ($\prod i \in J.\ M\ i\ (X\ i)$)
proof *cases*
assume $\neg\ (J \neq \{\} \vee I = \{\})$
then obtain *i* **where** $J = \{\}$ $i \in I$ **by** *auto*
moreover then have *emb I* $\{\}$ $\{\lambda x.\ undefined\}$ = *emb I* $\{i\}$ ($\Pi_E\ i \in \{i\}.\ space\ (M\ i)$)
by (*auto simp*: *space-PiM prod-emb-def*)
ultimately show *?thesis*
by (*simp add*: $*$ *M.emeasure-space-1*)
qed (*simp add*: $*$[*OF - assms X*])
with *assms* **show** *emeasure* (*distr* (*Pi*$_M$ *I M*) (*Pi*$_M$ *J M*) ($\lambda x.$ *restrict x J*)) (*Pi*$_E$ *J X*) = ($\prod i \in J.$ *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'*:
$J \subseteq I \Longrightarrow$ *finite J* $\Longrightarrow X \in sets\ (PiM\ J\ M) \Longrightarrow$ *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*:

$J \subseteq I \Longrightarrow$ *finite* $J \Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow X\ i \in$ *sets* $(M\ i)) \Longrightarrow$
emeasure $(Pi_M\ I\ M)$ $(emb\ I\ J\ (Pi_E\ J\ X)) = (\prod\ i \in J.$ *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 $*$: *emb* $I\ \{\}\ \{\lambda x.$ *undefined*$\} =$ *space* $(PiM\ I\ M)$
by (*auto simp*: *prod-emb-def space-PiM*)
show *emeasure* $(Pi_M\ I\ M)$ (*space* $(Pi_M\ I\ M)) = 1$
using *emeasure-PiM-emb*[*of* $\{\}\ \lambda$-. $\{\}$] **by** (*simp add*: $*$)
qed

lemma (**in** *product-prob-space*) *emeasure-PiM-Collect*:
assumes *X*: $J \subseteq I$ *finite* $J\ \bigwedge i.\ i \in J \Longrightarrow X\ i \in$ *sets* $(M\ i)$
shows *emeasure* $(Pi_M\ I\ M)$ $\{x \in$*space* $(Pi_M\ I\ M).\ \forall i \in J.\ x\ i \in X\ i\} = (\prod\ i \in J.$ *emeasure* $(M\ i)\ (X\ i))$
proof −
have $\{x \in$*space* $(Pi_M\ I\ M).\ \forall i \in J.\ x\ i \in X\ i\} =$ *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*: $i \in I\ A \in$ *sets* $(M\ i)$
shows *emeasure* $(Pi_M\ I\ M)$ $\{x \in$*space* $(Pi_M\ I\ M).\ x\ i \in A\} =$ *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 $J \subseteq I$ *finite* $J\ \bigwedge i.\ i \in J \Longrightarrow X\ i \in$ *sets* $(M\ i)$
shows *measure* $(PiM\ I\ M)$ $(emb\ I\ J\ (Pi_E\ J\ X)) = (\prod\ i \in J.$ *measure* $(M\ i)\ (X\ i))$
using *emeasure-PiM-emb*[*OF assms*]
unfolding *emeasure-eq-measure M.emeasure-eq-measure*
by (*simp add*: *setprod-ennreal measure-nonneg setprod-nonneg*)

lemma *sets-Collect-single'*:
$i \in I \Longrightarrow \{x \in$*space* $(M\ i).\ P\ x\} \in$ *sets* $(M\ i) \Longrightarrow \{x \in$*space* $(PiM\ I\ M).\ P\ (x\ i)\} \in$ *sets* $(PiM\ I\ M)$
using *sets-Collect-single*[*of i I* $\{x \in$*space* $(M\ i).\ P\ x\}\ M$]
by (*simp add*: *space-PiM PiE-iff cong*: *conj-cong*)

lemma (**in** *finite-product-prob-space*) *finite-measure-PiM-emb*:
$(\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *sets* $(M\ i)) \Longrightarrow$ *measure* $(PiM\ I\ M)$ $(Pi_E\ I\ A) = (\prod i \in I.$ *measure* $(M\ i)\ (A\ i))$
using *measure-PiM-emb*[*of I A*] *finite-index prod-emb-PiE-same-index*[*OF sets.sets-into-space*, *of I A M*]
by *auto*

lemma (**in** *product-prob-space*) *PiM-component*:
assumes $i \in I$
shows *distr* (*PiM I M*) (*M i*) ($\lambda\omega.\ \omega\ i$) = *M i*
proof (*rule measure-eqI*[*symmetric*])
fix *A* **assume** *A* ∈ *sets* (*M i*)
moreover have (($\lambda\omega.\ \omega\ i$) −‘ *A* ∩ *space* (*PiM I M*)) = {*x*∈*space* (*PiM I M*). *x i* ∈ *A*}
by *auto*
ultimately show *emeasure* (*M i*) *A* = *emeasure* (*distr* (*PiM I M*) (*M i*) ($\lambda\omega.$ $\omega\ i$)) *A*
by (*auto simp*: ‹*i*∈*I*› *emeasure-distr measurable-component-singleton emeasure-PiM-Collect-single*)
qed *simp*

lemma (**in** *product-prob-space*) *PiM-eq*:
assumes *M*′: *sets M*′ = *sets* (*PiM I M*)
assumes *eq*: $\bigwedge J\ F.$ *finite J* ⟹ *J* ⊆ *I* ⟹ ($\bigwedge j.\ j \in J \Longrightarrow F\ j \in$ *sets* (*M j*)) ⟹
emeasure M′ (*prod-emb I M J* ($\Pi_E\ j{\in}J.\ F\ j$)) = ($\prod j{\in}J.$ *emeasure* (*M j*) (*F j*))
shows *M*′ = (*PiM I M*)
proof (*rule measure-eqI-PiM-infinite*[*symmetric*, *OF refl M*′])
show *finite-measure* (Pi_M *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* ∈ *I* ⟹ *AE x in M i*. *P x* ⟹ *AE x 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

23.1 Sequence space

definition *comb-seq* :: *nat* ⇒ (*nat* ⇒ ′*a*) ⇒ (*nat* ⇒ ′*a*) ⇒ (*nat* ⇒ ′*a*) **where**
comb-seq i ω ω′ *j* = (*if j* < *i then* ω *j else* ω′ (*j* − *i*))

lemma *split-comb-seq*: *P* (*comb-seq i* ω ω′ *j*) ⟷ (*j* < *i* ⟶ *P* (ω *j*)) ∧ (∀ *k*. *j* = *i* + *k* ⟶ *P* (ω′ *k*))
by (*auto simp*: *comb-seq-def not-less*)

lemma *split-comb-seq-asm*: *P* (*comb-seq i* ω ω′ *j*) ⟷ ¬ ((*j* < *i* ∧ ¬ *P* (ω *j*)) ∨ (∃ *k*. *j* = *i* + *k* ∧ ¬ *P* (ω′ *k*)))
by (*auto simp*: *comb-seq-def*)

lemma *measurable-comb-seq*:
($\lambda(\omega, \omega').$ *comb-seq i* ω ω′) ∈ *measurable* (($\Pi_M\ i{\in}UNIV.\ M$) $\bigotimes_M$ ($\Pi_M\ i{\in}UNIV.$

M)) (Π_M $i{\in}UNIV.$ M)
proof (*rule measurable-PiM-single*)
show ($\lambda(\omega,\ \omega').$ *comb-seq* i ω ω') $\in$ *space* ((Π_M $i{\in}UNIV.$ M) $\bigotimes_M$ (Π_M $i{\in}UNIV.$ M)) $\rightarrow$ ($UNIV$ $\rightarrow_E$ *space* M)
by (*auto simp*: *space-pair-measure space-PiM PiE-iff split*: *split-comb-seq*)
fix j :: *nat* **and** A **assume** A: $A \in sets\ M$
then have $*$: {$\omega \in space$ ((Π_M $i{\in}UNIV.$ M) $\bigotimes_M$ (Π_M $i{\in}UNIV.$ M)). *case-prod* (*comb-seq* i) ω $j \in A$} =
(*if* $j < i$ *then* {$\omega \in space$ (Π_M $i{\in}UNIV.$ M). ω $j \in A$} $\times$ *space* (Π_M $i{\in}UNIV.$ M)
else space (Π_M $i{\in}UNIV.$ M) $\times$ {$\omega \in space$ (Π_M $i{\in}UNIV.$ M). ω (j $-$ i) $\in A$})
by (*auto simp*: *space-PiM space-pair-measure comb-seq-def dest*: *sets.sets-into-space*)
show {$\omega \in space$ ((Π_M $i{\in}UNIV.$ M) $\bigotimes_M$ (Π_M $i{\in}UNIV.$ M)). *case-prod* (*comb-seq* i) ω $j \in A$} $\in$ *sets* ((Π_M $i{\in}UNIV.$ M) $\bigotimes_M$ (Π_M $i{\in}UNIV.$ M))
unfolding $*$ **by** (*auto simp*: A *intro*!: *sets-Collect-single*)
qed

lemma *measurable-comb-seq'*[*measurable* (*raw*)]:
assumes f: $f \in measurable\ N$ (Π_M $i{\in}UNIV.$ M) **and** g: $g \in measurable\ N$ (Π_M $i{\in}UNIV.$ M)
shows ($\lambda x.$ *comb-seq* i ($f\ x$) ($g\ x$)) $\in$ *measurable* N (Π_M $i{\in}UNIV.$ M)
using *measurable-compose*[*OF measurable-Pair*[*OF f g*] *measurable-comb-seq*] **by** *simp*

lemma *comb-seq-0*: *comb-seq* 0 ω $\omega' = \omega'$
by (*auto simp add*: *comb-seq-def*)

lemma *comb-seq-Suc*: *comb-seq* (*Suc* n) ω $\omega' =$ *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*: $i < n \Longrightarrow$ *comb-seq* n ω ω' $i = \omega$ i
by (*auto split*: *split-comb-seq*)

lemma *comb-seq-add*: *comb-seq* n ω ω' (i + n) = ω' 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) ω' i
by (*auto split*: *nat.split split-comb-seq*)

lemma *case-nat-comb-seq'*:
case-nat s (*comb-seq* i ω ω') = *comb-seq* (*Suc* i) (*case-nat* s ω) ω'
by (*auto split*: *split-comb-seq nat.split*)

locale *sequence-space* = *product-prob-space* $\lambda i.\ M\ UNIV$ :: *nat set* **for** M
begin

abbreviation $S \equiv \Pi_M\ i{\in}UNIV{::}nat\ set.\ M$

lemma *infprod-in-sets*[*intro*]:
fixes E :: *nat* $\Rightarrow$ $'a$ *set* **assumes** E: $\bigwedge i.\ E\ i \in sets\ M$
shows $Pi\ UNIV\ E \in sets\ S$
proof −
have $Pi\ UNIV\ E = (\bigcap i.\ emb\ UNIV\ \{..i\}\ (\Pi_E\ j{\in}\{..i\}.\ E\ j))$
using E E[*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 :: *nat* $\Rightarrow$ $'a$ *set* **assumes** E: $\bigwedge i.\ E\ i \in sets\ M$
shows $(\lambda n.\ \prod i{\le}n.\ measure\ M\ (E\ i)) \longrightarrow measure\ S\ (Pi\ UNIV\ E)$
proof −
let *?E* = $\lambda n.\ emb\ UNIV\ \{..n\}\ (Pi_E\ \{..\ n\}\ E)$
have $\bigwedge n.\ (\prod i{\le}n.\ measure\ M\ (E\ i)) = measure\ S\ (?E\ n)$
using E **by** (*simp add*: *measure-PiM-emb*)
moreover have $Pi\ UNIV\ E = (\bigcap n.\ ?E\ n)$
using E E[*THEN sets.sets-into-space*]
by (*auto simp*: *prod-emb-def extensional-def Pi-iff*)
moreover have *range ?E* $\subseteq$ *sets S*
using E **by** *auto*
moreover have *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 $a\ b\ c$ :: *nat*
shows $c \le b \Longrightarrow a = b - c \longleftrightarrow a + c = b$
by *auto*

lemma *PiM-comb-seq*:
$distr\ (S \bigotimes_M S)\ S\ (\lambda(\omega, \omega').\ comb\text{-}seq\ i\ \omega\ \omega') = S$ (**is** *?D* = -)
proof (*rule PiM-eq*)
let *?I* = $UNIV{::}nat\ set$ **and** *?M* = $\lambda n.\ M$
let *distr - - ?f* = *?D*

fix $J\ E$ **assume** J: *finite J* $J \subseteq$ *?I* $\bigwedge j.\ j \in J \Longrightarrow E\ j \in sets\ M$
let *?X* = *prod-emb ?I ?M J* $(\Pi_E\ j{\in}J.\ E\ j)$
have $\bigwedge j\ x.\ j \in J \Longrightarrow x \in E\ j \Longrightarrow x \in space\ M$
using $J(3)$[*THEN sets.sets-into-space*] **by** (*auto simp*: *space-PiM Pi-iff subset-eq*)
with J **have** *?f* −‘ *?X* $\cap$ *space* $(S \bigotimes_M S) =$

(*prod-emb ?I ?M* (*J* ∩ {..<*i*}) (*PIE j*:*J* ∩ {..<*i*}. *E j*)) ×
(*prod-emb ?I ?M* ((*op* + *i*) −‘ *J*) (*PIE j*:(*op* + *i*) −‘ *J*. *E* (*i* + *j*))) (**is** - = *?E* × *?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 *emeasure ?D ?X* = *emeasure* (*S* ⨂$_M$ *S*) (*?E* × *?F*)
by (*subst emeasure-distr*[*OF measurable-comb-seq*])
(*auto intro*!: *sets-PiM-I simp*: *split-beta*′ *J*)
also have ... = *emeasure S ?E* ∗ *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 *emeasure S ?F* = (∏*j*∈(*op* + *i*) −‘ *J*. *emeasure M* (*E* (*i* + *j*)))
using *J* **by** (*intro emeasure-PiM-emb*) (*simp-all add*: *finite-vimageI inj-on-def*)
also have ... = (∏*j*∈*J* − (*J* ∩ {..<*i*}). *emeasure M* (*E j*))
by (*rule setprod.reindex-cong* [*of* λ*x*. *x* − *i*])
(*auto simp*: *image-iff Bex-def not-less nat-eq-diff-eq ac-simps cong*: *conj-cong intro*!: *inj-onI*)
also have *emeasure S ?E* = (∏*j*∈*J* ∩ {..<*i*}. *emeasure M* (*E j*))
using *J* **by** (*intro emeasure-PiM-emb*) *simp-all*
also have (∏*j*∈*J* ∩ {..<*i*}. *emeasure M* (*E j*)) ∗ (∏*j*∈*J* − (*J* ∩ {..<*i*}). *emeasure M* (*E j*)) = (∏*j*∈*J*. *emeasure M* (*E j*))
by (*subst mult.commute*) (*auto simp*: *J setprod.subset-diff*[*symmetric*])
finally show *emeasure ?D ?X* = (∏*j*∈*J*. *emeasure M* (*E j*)) .
qed *simp-all*

lemma *PiM-iter*:
distr (*M* ⨂$_M$ *S*) *S* (λ(*s*, ω). *case-nat s* ω) = *S* (**is** *?D* = -)
proof (*rule PiM-eq*)
let *?I* = *UNIV*::*nat set* **and** *?M* = λ*n*. *M*
let *distr* - - *?f* = *?D*

fix *J E* **assume** *J*: *finite J J* ⊆ *?I* ⋀*j*. *j* ∈ *J* ⟹ *E j* ∈ *sets M*
let *?X* = *prod-emb ?I ?M J* (*PIE j*:*J*. *E j*)
have ⋀*j x*. *j* ∈ *J* ⟹ *x* ∈ *E j* ⟹ *x* ∈ *space M*
using *J*(*3*)[*THEN sets.sets-into-space*] **by** (*auto simp*: *space-PiM Pi-iff subset-eq*)
with *J* **have** *?f* −‘ *?X* ∩ *space* (*M* ⨂$_M$ *S*) = (*if 0* ∈ *J then E 0 else space M*) ×
(*prod-emb ?I ?M* (*Suc* −‘ *J*) (*PIE j*:*Suc* −‘ *J*. *E* (*Suc j*))) (**is** - = *?E* × *?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 *emeasure ?D ?X* = *emeasure* (*M* ⨂$_M$ *S*) (*?E* × *?F*)
by (*subst emeasure-distr*)
(*auto intro*!: *sets-PiM-I simp*: *split-beta*′ *J*)
also have ... = *emeasure M ?E* ∗ *emeasure S ?F*
using *J* **by** (*intro P.emeasure-pair-measure-Times*) (*auto intro*!: *sets-PiM-I finite-vimageI*)
also have *emeasure S ?F* = (∏*j*∈*Suc* −‘ *J*. *emeasure M* (*E* (*Suc j*)))
using *J* **by** (*intro emeasure-PiM-emb*) (*simp-all add*: *finite-vimageI*)
also have ... = (∏*j*∈*J* − {*0*}. *emeasure M* (*E j*))

by (*rule setprod.reindex-cong* [*of* $\lambda x.\ x - 1$])
(*auto simp*: *image-iff Bex-def not-less nat-eq-diff-eq ac-simps cong*: *conj-cong intro*!: *inj-onI*)
also have *emeasure M ?E* $*$ $(\prod j{\in}J - \{0\}.\ emeasure\ M\ (E\ j)) = (\prod j{\in}J.\ emeasure\ M\ (E\ j))$
by (*auto simp*: *M.emeasure-space-1 setprod.remove J*)
finally show *emeasure ?D ?X* $= (\prod j{\in}J.\ emeasure\ M\ (E\ j))$ **.**
qed *simp-all*

end

end

24 Projective Limit

theory *Projective-Limit*
imports
Caratheodory
Fin-Map
Regularity
Projective-Family
Infinite-Product-Measure
~~/*src/HOL/Library/Diagonal-Subsequence*
begin

24.1 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*: $\bigwedge n.\ k \in K\ n \Longrightarrow domain\ k = domain\ (f\ n)$
assumes *proj-in-K*:
$\bigwedge t\ n\ m.\ m \geq n \Longrightarrow t \in domain\ (f\ n) \Longrightarrow (f\ m)_F\ t \in (\lambda k.\ (k)_F\ t)\ `\ K\ n$
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* - ‹$k \in K\ (Suc\ 0)$›]
simp: *domain-K*[*OF* ‹$k \in K\ (Suc\ 0)$›])
qed *blast*

lemma *proj-in-KE*:
obtains n **where** $\bigwedge m.\ m \geq n \Longrightarrow (f\ m)_F\ t \in (\lambda k.\ (k)_F\ t)\ `\ K\ n$

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 $S :: 'a :: metric\text{-}space\ set$
assumes *compact* $S\ \forall\, n{\geq}m.\ f\ n \in S$
obtains $l\ r$ **where** $l \in S\ subseq\ r\ ((f \circ r) \longrightarrow l)\ sequentially$
proof *atomize-elim*
have *subseq* $(op + m)$ **by** (*simp add*: *subseq-def*)
have $\forall\, n.\ (f\ o\ (\lambda i.\ m + i))\ n \in S$ **using** *assms* **by** *auto*
from *seq-compactE*[*OF* ‹*compact S*›[*unfolded compact-eq-seq-compact-metric*] *this*]
guess $l\ r$ **.**
hence $l \in S\ subseq\ ((\lambda i.\ m + i)\ o\ r) \wedge (f \circ ((\lambda i.\ m + i)\ o\ r)) \longrightarrow l$
using *subseq-o*[*OF* ‹*subseq* $(op + m)$› ‹*subseq r*›] **by** (*auto simp*: *o-def*)
thus $\exists\, l\ r.\ l \in S \wedge subseq\ r \wedge (f \circ r) \longrightarrow l$ **by** *blast*
qed

sublocale *finmap-seqs-into-compact* $\subseteq$ *subseqs* $\lambda n\ s.\ (\exists\, l.\ (\lambda i.\ ((f\ o\ s)\ i)_F\ n) \longrightarrow l)$
proof
fix $n\ s$
assume *subseq s*
from *proj-in-KE*[*of n*] **guess** *n0* **. note** $n0 = this$
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 $\ldots \leq s\ i$ **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
from *compactE'*[*OF compact-projset this*] **guess** *ls rs* **.**
thus $\exists\, r'.\ subseq\ r' \wedge (\exists\, l.\ (\lambda i.\ ((f \circ (s \circ r'))\ i)_F\ n) \longrightarrow l)$ **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** $(\lambda i.\ ((f\ o\ (diagseq\ o\ op + (Suc\ n)))\ i)_F\ n) \longrightarrow l$
proof (*atomize-elim*, *rule diagseq-holds*)
fix $r\ s\ n$
assume *subseq r*
assume $\exists\, l.\ (\lambda i.\ ((f \circ s)\ i)_F\ n) \longrightarrow l$

then obtain l **where** $((\lambda i.\ (f\ i)_F\ n)\ o\ s) \longrightarrow l$
by (*auto simp*: *o-def*)
hence $((\lambda i.\ (f\ i)_F\ n)\ o\ s\ o\ r) \longrightarrow l$ **using** ‹*subseq r*›
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 $(\lambda i.\ ((f\ (diagseq\ (i + Suc\ n))))_F\ n) \longrightarrow l$ **by** (*simp add*: *ac-simps*)
hence $(\lambda i.\ (f\ (diagseq\ i))_F\ n) \longrightarrow l$ **by** (*rule LIMSEQ-offset*)
thus *?thesis* **..**
qed

24.2 Daniell-Kolmogorov Theorem

Existence of Projective Limit

locale *polish-projective* = *projective-family I P* λ*-. borel*::*'a*::*polish-space measure*
for *I*::*'i set* **and** *P*
begin

lemma *emeasure-lim-emb*:
assumes *X*: $J \subseteq I$ *finite J X* $\in$ *sets* $(\Pi_M\ i{\in}J.\ borel)$
shows *lim* $(emb\ I\ J\ X) = P\ J\ X$
proof (*rule emeasure-lim*)
write *mu-G* (μG)
interpret *generator*: *algebra space* $(PiM\ I\ (\lambda i.\ borel))$ *generator*
by (*rule algebra-generator*)

fix *J* **and** *B* :: *nat* $\Rightarrow$ $('i \Rightarrow 'a)$ *set*
assume *J*: $\bigwedge n.$ *finite* $(J\ n)$ $\bigwedge n.\ J\ n \subseteq I$ $\bigwedge n.\ B\ n \in sets\ (\Pi_M\ i{\in}J\ n.\ borel)$ *incseq J*
and *B*: *decseq* $(\lambda n.\ emb\ I\ (J\ n)\ (B\ n))$
and $0 < (INF\ i.\ P\ (J\ i)\ (B\ i))$ (**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*: $?a = ennreal\ r\ 0 < r\ r \leq 1$
by (*cases ?a*) (*auto simp*: *top-unique*)
def $Z \equiv \lambda n.\ emb\ I\ (J\ n)\ (B\ n)$
have *Z-mono*: $n \leq m \Longrightarrow Z\ m \subseteq Z\ n$ **for** *n m*
unfolding *Z-def* **using** *B*[*THEN antimonoD*, *of n m*] **.**
have *J-mono*: $\bigwedge n\ m.\ n \leq m \Longrightarrow J\ n \subseteq J\ m$
using ‹*incseq J*› **by** (*force simp*: *incseq-def*)
note [*simp*] = ‹$\bigwedge n.$ *finite* $(J\ n)$›
interpret *prob-space P* $(J\ i)$ **for** *i* **using** *J prob-space-P* **by** *simp*

have *P-eq*[*simp*]:
sets $(P\ (J\ i)) = sets\ (\Pi_M\ i{\in}J\ i.\ borel)$ *space* $(P\ (J\ i)) = space\ (\Pi_M\ i{\in}J\ i.\ borel)$ **for** *i*
using *J* **by** (*auto simp*: *sets-P space-P*)

have $Z\ i \in generator$ **for** *i*

```
    unfolding Z-def by (auto intro!: generator.intros J)

  have countable-UN-J: countable (⋃ n. J n) by (simp add: countable-finite)
  def Utn ≡ to-nat-on (⋃ n. J n)
  interpret function-to-finmap J n Utn from-nat-into (⋃ n. J n) for n
   by unfold-locales (auto simp: Utn-def intro: from-nat-into-to-nat-on[OF countable-UN-J])
  have inj-on-Utn: inj-on Utn (⋃ n. J n)
    unfolding Utn-def using countable-UN-J by (rule inj-on-to-nat-on)
  hence inj-on-Utn-J: ⋀n. inj-on Utn (J n) by (rule subset-inj-on) auto
  def P' ≡ λn. mapmeasure n (P (J n)) (λ-. borel)
  interpret P': prob-space P' n for n
    unfolding P'-def mapmeasure-def using J
    by (auto intro!: prob-space-distr fm-measurable simp: measurable-cong-sets[OF
sets-P])

  let ?SUP = λn. SUP K : {K. K ⊆ fm n ' (B n) ∧ compact K}. emeasure (P'
n) K
  { fix n
    have emeasure (P (J n)) (B n) = emeasure (P' n) (fm n ' (B n))
      using J by (auto simp: P'-def mapmeasure-PiM space-P sets-P)
    also
    have ... = ?SUP n
    proof (rule inner-regular)
      show sets (P' n) = sets borel by (simp add: borel-eq-PiF-borel P'-def)
    next
      show fm n ' B n ∈ sets borel
      unfolding borel-eq-PiF-borel by (auto simp: P'-def fm-image-measurable-finite
sets-P J(3))
    qed simp
    finally have *: emeasure (P (J n)) (B n) = ?SUP n .
    have ?SUP n ≠ ∞
      unfolding *[symmetric] by simp
    note * this
  } note R = this
  have ∀ n. ∃ K. emeasure (P (J n)) (B n) - emeasure (P' n) K ≤ 2 powr (-n)
* ?a ∧ compact K ∧ K ⊆ fm n ' B n
  proof
    fix n show ∃ K. emeasure (P (J n)) (B n) - emeasure (P' n) K ≤ ennreal (2
powr - real n) * ?a ∧
        compact K ∧ K ⊆ fm n ' B n
      unfolding R[of n]
    proof (rule ccontr)
      assume H: ∄ K'. ?SUP n - emeasure (P' n) K' ≤ ennreal (2 powr - real
n) * ?a ∧
        compact K' ∧ K' ⊆ fm n ' B n
      have ?SUP n + 0 < ?SUP n + 2 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
 def K ≡ λn. fm n −‘ K' n ∩ space (Pi_M (J n) (λ-. borel))
 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
 def Z' ≡ λn. emb I (J n) (K 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 ∈ B n
    hence y ∈ space (Pi_M (J n) (λ-. borel)) using J sets.sets-into-space[of B n
```

P (J n)]
by (*auto simp add*: *space-P sets-P*)
assume *fm n x = fm n y*
note *inj-onD*[*OF inj-on-fm*[*OF space-borel*],
OF ‹*fm n x = fm n y*› ‹*x ∈ space -*› ‹*y ∈ space -*›]
with *y* **show** *x ∈ B n* **by** *simp*
qed
qed
have ⋀*n*. *Z' n ∈ generator* **using** *J K'*(*2*) **unfolding** *Z'-def*
by (*auto intro*!: *generator.intros measurable-sets*[*OF fm-measurable*[*of - Collect finite*]]
simp: *K-def borel-eq-PiF-borel*[*symmetric*] *compact-imp-closed*)
def *Y* ≡ λ*n*. ⋂*i*∈{*1..n*}. *Z' i*
hence ⋀*n k*. *Y* (*n* + *k*) ⊆ *Y n* **by** (*induct-tac k*) (*auto simp*: *Y-def*)
hence *Y-mono*: ⋀*n m*. *n* ≤ *m* ⟹ *Y m* ⊆ *Y n* **by** (*auto simp*: *le-iff-add*)
have *Y-Z'*: ⋀*n*. *n* ≥ *1* ⟹ *Y n* ⊆ *Z' n* **by** (*auto simp*: *Y-def*)
hence *Y-Z*: ⋀*n*. *n* ≥ *1* ⟹ *Y n* ⊆ *Z n* **using** *Z'* **by** *auto*

have *Y-notempty*: ⋀*n*. *n* ≥ *1* ⟹ (*Y n*) ≠ {}
proof −
fix *n*::*nat* **assume** *n* ≥ *1* **hence** *Y n* ⊆ *Z n* **by** *fact*
have *Y n* = (⋂*i*∈{*1..n*}. *emb I* (*J n*) (*emb* (*J n*) (*J i*) (*K i*))) **using** *J J-mono*
by (*auto simp*: *Y-def Z'-def*)
also have ... = *prod-emb I* (λ-. *borel*) (*J n*) (⋂*i*∈{*1..n*}. *emb* (*J n*) (*J i*) (*K i*))
using ‹*n* ≥ *1*›
by (*subst prod-emb-INT*) *auto*
finally
have *Y-emb*:
Y n = prod-emb I (λ-. *borel*) (*J n*) (⋂*i*∈{*1..n*}. *prod-emb* (*J n*) (λ-. *borel*) (*J i*) (*K i*)) **.**
hence *Y n* ∈ *generator* **using** *J J-mono K-sets* ‹*n* ≥ *1*›
by (*auto simp del*: *prod-emb-INT intro*!: *generator.intros*)
have ∗: *μG* (*Z n*) = *P* (*J n*) (*B n*)
unfolding *Z-def* **using** *J* **by** (*intro mu-G-spec*) *auto*
then have *μG* (*Z n*) ≠ ∞ **by** *auto*
note ∗
moreover have ∗: *μG* (*Y n*) = *P* (*J n*) (⋂*i*∈{*Suc 0..n*}. *prod-emb* (*J n*) (λ-. *borel*) (*J i*) (*K i*))
unfolding *Y-emb* **using** *J J-mono K-sets* ‹*n* ≥ *1*› **by** (*subst mu-G-spec*) *auto*
then have *μG* (*Y n*) ≠ ∞ **by** *auto*
note ∗
moreover have *μG* (*Z n* − *Y n*) =
P (*J n*) (*B n* − (⋂*i*∈{*Suc 0..n*}. *prod-emb* (*J n*) (λ-. *borel*) (*J i*) (*K i*)))
unfolding *Z-def Y-emb prod-emb-Diff*[*symmetric*] **using** *J J-mono K-sets* ‹*n* ≥ *1*›
by (*subst mu-G-spec*) (*auto intro*!: *sets.Diff*)
ultimately
have *μG* (*Z n*) − *μG* (*Y n*) = *μG* (*Z n* − *Y n*)

using *J J-mono K-sets* ‹$n \geq 1$›
by (*simp only*: *emeasure-eq-measure Z-def*)
(*auto dest*!: *bspec*[**where** x=n] *intro*!: *measure-Diff*[*symmetric*] *set-mp*[*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 ‹$n \geq 1$› **unfolding** *Y-def UN-extend-simps*(7) **by** (*intro UN-mono Diff-mono Z-mono order-refl*) *auto*
have $Z\ n - Y\ n \in$ *generator* $(\bigcup i{\in}\{1..n\}.\ (Z\ i - Z'\ i)) \in$ *generator*
using ‹Z' - ∈ *generator*› ‹Z - ∈ *generator*› ‹Y - ∈ *generator*› **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 $\ldots \leq (\sum\ i{\in}\{1..n\}.\ \mu G\ (Z\ i - Z'\ i))$ **using** ‹Z - ∈ *generator*› ‹Z' - ∈ *generator*›
by (*intro generator.subadditive*[*OF positive-mu-G additive-mu-G*]) *auto*
also have $\ldots \leq (\sum\ i{\in}\{1..n\}.\ 2\ powr\ -real\ i * ?a)$
proof (*rule setsum-mono*)
fix i **assume** $i \in \{1..n\}$ **hence** $i \leq n$ **by** *simp*
have $\mu G\ (Z\ i - Z'\ i) = \mu G\ (prod\text{-}emb\ I\ (\lambda\text{-}.\ borel)\ (J\ i)\ (B\ i - K\ i))$
unfolding *Z'-def Z-def* **by** *simp*
also have $\ldots = P\ (J\ i)\ (B\ i - K\ i)$
using *J K-sets* **by** (*subst mu-G-spec*) *auto*
also have $\ldots = P\ (J\ i)\ (B\ i) - P\ (J\ i)\ (K\ i)$
using *K-sets J* ‹K - ⊆ B -› **by** (*simp add*: *emeasure-Diff*)
also have $\ldots = P\ (J\ i)\ (B\ i) - P'\ i\ (K'\ i)$
unfolding *K-def P'-def*
by (*auto simp*: *mapmeasure-PiF borel-eq-PiF-borel*[*symmetric*]
compact-imp-closed[*OF* ‹*compact* (K' -)›] *space-PiM PiE-def*)
also have $\ldots \leq ennreal\ (2\ powr\ -\ real\ i) * ?a$ **using** $K'(1)$[*of i*] **.**
finally show $\mu G\ (Z\ i - Z'\ i) \leq (2\ powr\ -\ real\ i) * ?a$ **.**
qed
also have $\ldots = ennreal\ ((\sum\ i{\in}\{1..n\}.\ (2\ powr\ -enn2real\ i)) * enn2real\ ?a)$
using r **by** (*simp add*: *setsum-left-distrib ennreal-mult*[*symmetric*])
also have $\ldots <$ *ennreal* $(1 * enn2real\ ?a)$
proof (*intro ennreal-lessI mult-strict-right-mono*)
have $(\sum i = 1..n.\ 2\ powr\ -\ real\ i) = (\sum i = 1..<Suc\ n.\ (1/2)\ \hat{}\ i)$
by (*rule setsum.cong*) (*auto simp*: *powr-realpow powr-divide power-divide powr-minus-divide*)
also have $\{1..<Suc\ n\} = \{..<Suc\ n\} - \{0\}$ **by** *auto*
also have *setsum* $(op\ \hat{}\ (1\ /\ 2::real))\ (\{..<Suc\ n\} - \{0\}) =$
setsum $(op\ \hat{}\ (1\ /\ 2))\ (\{..<Suc\ n\}) - 1$ **by** (*auto simp*: *setsum-diff1*)
also have $\ldots < 1$ **by** (*subst geometric-sum*) *auto*
finally show $(\sum i = 1..n.\ 2\ powr\ -\ enn2real\ i) < 1$ **by** *simp*
qed (*auto simp*: r *enn2real-positive-iff*)
also have $\ldots = ?a$ **by** (*auto simp*: r)

also have $\ldots \leq \mu G\ (Z\ n)$
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 ‹$\mu G\ (Y\ n) \neq \infty$› **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*: $\bigwedge n.\ n \geq 1 \Longrightarrow y\ n \in Y\ n$ **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* ‹$m \geq n$›] *y*[*OF* ‹$1 \leq m$›] **have** $y\ m \in Y\ n$ **by** *auto*
also have $\ldots \subseteq Z'\ n$ **using** *Y-Z'*[*OF* ‹$1 \leq n$›] **.**
finally
have *fm n* (*restrict* (*y m*) (*J n*)) ∈ *K' n*
unfolding *Z'-def K-def prod-emb-iff* **by** (*simp add*: *Z'-def K-def prod-emb-iff*)
moreover have *finmap-of* (*J n*) (*restrict* (*y m*) (*J n*)) = *finmap-of* (*J n*) (*y m*)
using *J* **by** (*simp add*: *fm-def*)
ultimately have *fm n* (*y m*) ∈ *K' n* **by** *simp*
} note *fm-in-K' = this*
interpret *finmap-seqs-into-compact* λ*n*. *K'* (*Suc n*) λ*k*. *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*) ∈ *Y* (*Suc n*) **by** *auto*
also have $\ldots \subseteq Z'\ (Suc\ n)$ **using** *Y-Z'* **by** *auto*
finally
have *fm* (*Suc n*) (*restrict* (*y* (*Suc n*)) (*J* (*Suc n*))) ∈ *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*) ≠ {} **by** *auto*
fix *k*
assume *k* ∈ *K'* (*Suc n*)
with *K'*[*of Suc n*] *sets.sets-into-space* **have** *k* ∈ *fm* (*Suc n*) ‘ *B* (*Suc n*) **by** *auto*
then obtain *b* **where** *k* = *fm* (*Suc n*) *b* **by** *auto*
thus *domain k* = *domain* (*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* ≤ *Suc m* **by** *simp*
assume *t* ∈ *domain* (*fm* (*Suc n*) (*y* (*Suc n*)))
then obtain *j* **where** *j*: *t* = *Utn j j* ∈ *J* (*Suc n*) **by** *auto*

```
    hence j ∈ J (Suc m) using J-mono[OF ‹Suc n ≤ Suc m›] by auto
    have img: fm (Suc n) (y (Suc m)) ∈ K' (Suc n) using ‹n ≤ m›
      by (intro fm-in-K') simp-all
    show (fm (Suc m) (y (Suc m)))_F t ∈ (λk. (k)_F t) ‘ K' (Suc n)
      apply (rule image-eqI[OF - img])
      using ‹j ∈ J (Suc n)› ‹j ∈ J (Suc m)›
      unfolding j by (subst proj-fm, auto)+
  qed
  have ∀t. ∃z. (λi. (fm (Suc (diagseq i)) (y (Suc (diagseq i))))_F t) ⟶ z
    using diagonal-tendsto ..
  then obtain z where z:
    ⋀t. (λi. (fm (Suc (diagseq i)) (y (Suc (diagseq i))))_F t) ⟶ z t
    unfolding choice-iff by blast
  {
    fix n :: nat assume n ≥ 1
    have ⋀i. domain (fm n (y (Suc (diagseq i)))) = domain (finmap-of (Utn ‘ J
n) z)
      by simp
    moreover
    {
      fix t
      assume t: t ∈ domain (finmap-of (Utn ‘ J n) z)
      hence t ∈ Utn ‘ J n by simp
      then obtain j where j: t = Utn j j ∈ J n by auto
      have (λi. (fm n (y (Suc (diagseq i))))_F t) ⟶ z t
        apply (subst (2) tendsto-iff, subst eventually-sequentially)
      proof safe
        fix e :: real assume 0 < e
        { fix i and x :: 'i ⇒ 'a assume i: i ≥ n
          assume t ∈ domain (fm n x)
          hence t ∈ domain (fm i x) using J-mono[OF ‹i ≥ n›] 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: ⋀i. i ≥ n ⟹ Suc (diagseq i) ≥ n
          apply (rule le-SucI)
          apply (rule order-trans) apply simp
          apply (rule seq-suble[OF subseq-diagseq])
          done
        from z
        have ∃N. ∀i≥N. dist ((fm (Suc (diagseq i)) (y (Suc (diagseq i))))_F t) (z
t) < e
          unfolding tendsto-iff eventually-sequentially using ‹0 < e› by auto
        then obtain N where N: ⋀i. i ≥ N ⟹
          dist ((fm (Suc (diagseq i)) (y (Suc (diagseq i))))_F t) (z t) < e by auto
        show ∃N. ∀na≥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 ≤ na
          hence  dist ((fm n (y (Suc (diagseq na))))_F t) (z t) =
```

```
            dist ((fm (Suc (diagseq na)) (y (Suc (diagseq na))))_F t) (z t) using
t
          by (subst index-shift[OF I]) auto
        also have ... < e using ‹max N n ≤ na› by (intro N) simp
        finally show dist ((fm n (y (Suc (diagseq na))))_F t) (z t) < e .
      qed
    qed
    hence (λi. (fm n (y (Suc (diagseq i))))_F t) ⟶ (finmap-of (Utn ‘ J n)
z)_F t
      by (simp add: tendsto-intros)
  } ultimately
  have (λi. fm n (y (Suc (diagseq i)))) ⟶ finmap-of (Utn ‘ J n) z
    by (rule tendsto-finmap)
  hence ((λi. fm n (y (Suc (diagseq i)))) o (λi. i + n)) ⟶ finmap-of (Utn
‘ J n) z
    by (rule LIMSEQ-subseq-LIMSEQ) (simp add: subseq-def)
  moreover
  have (∀ i. ((λi. fm n (y (Suc (diagseq i)))) o (λi. i + n)) i ∈ K' n)
    apply (auto simp add: o-def intro!: fm-in-K' ‹1 ≤ n› le-SucI)
    apply (rule le-trans)
    apply (rule le-add2)
    using seq-suble[OF subseq-diagseq]
    apply auto
    done
  moreover
  from ‹compact (K' n)› have closed (K' n) by (rule compact-imp-closed)
  ultimately
  have finmap-of (Utn ‘ J n) z ∈ K' n
    unfolding closed-sequential-limits by blast
  also have finmap-of (Utn ‘ J n) z = fm n (λi. z (Utn i))
    unfolding finmap-eq-iff
  proof clarsimp
    fix i assume i: i ∈ J n
    hence from-nat-into (⋃ 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) = (fm n (λi. z (Utn i)))_F (Utn i)
      by (simp add: finmap-eq-iff fm-def compose-def)
  qed
  finally have fm n (λi. z (Utn i)) ∈ K' n .
  moreover
  let ?J = ⋃ n. J n
  have (?J ∩ J n) = J n by auto
  ultimately have restrict (λi. z (Utn i)) (?J ∩ J n) ∈ K n
    unfolding K-def by (auto simp: space-P space-PiM)
  hence restrict (λi. z (Utn i)) ?J ∈ Z' n unfolding Z'-def
    using J by (auto simp: prod-emb-def PiE-def extensional-def)
  also have ... ⊆ Z n using Z' by simp
  finally have restrict (λi. z (Utn i)) ?J ∈ 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*:
$J \subseteq I \Longrightarrow$ *finite* $J \Longrightarrow X \in$ *sets* $(\Pi_M\ i{\in}J.$ *borel*$) \Longrightarrow$ *measure lim* (*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**) *Abs-finmap*
hide-const (**open**) *Rep-finmap*
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 $*$: *emb I* {} $\{\lambda x.\ undefined\}$ = *space* $(\Pi_M\ i{\in}I.$ *borel*$)$
by (*auto simp*: *prod-emb-def space-PiM*)
interpret *prob-space P* {}
using *prob-space-P* **by** *simp*
show *emeasure lim* (*space lim*) = *1*
using *emeasure-lim-emb*[*of* {} $\{\lambda x.\ undefined\}$] *emeasure-space-1*
by (*simp add*: $*$ *PiM-empty space-P*)
qed

locale *polish-product-prob-space* =
product-prob-space λ-. *borel*::('*a*::*polish-space*) *measure I* **for** *I*::'*i set*

sublocale *polish-product-prob-space* $\subseteq$ *P*: *polish-projective I* $\lambda J.$ *PiM J* (λ-. *borel*::('*a*) *measure*)
proof qed

lemma (**in** *polish-product-prob-space*) *limP-eq-PiM*: *lim* = *PiM I* (λ-. *borel*)
by (*rule PiM-eq*) (*auto simp*: *emeasure-PiM emeasure-lim-emb*)

end

25 Probability mass function

theory *Probability-Mass-Function*

imports
Giry-Monad
~~/src/HOL/Library/Multiset
begin

lemma *AE-emeasure-singleton*:
assumes *x*: *emeasure M* {*x*} ≠ *0* **and** *ae*: *AE x in M*. *P x* **shows** *P x*
proof −
from *x* **have** *x-M*: {*x*} ∈ *sets M*
by (*auto intro*: *emeasure-notin-sets*)
from *ae* **obtain** *N* **where** *N*: {*x*∈*space M*. ¬ *P x*} ⊆ *N emeasure M N* = *0 N* ∈ *sets M*
by (*auto elim*: *AE-E*)
{ assume ¬ *P x*
with *x-M*[*THEN sets.sets-into-space*] *N* **have** *emeasure M* {*x*} ≤ *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*} ≠ *0* ⟹ *AE x in M*. *P x* ⟹ *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*} ≠ *0*) ⟷ (∃ *S*. *countable S* ∧ (*AE x in M*. *x* ∈ *S*))
proof
assume ∃ *S*. *countable S* ∧ (*AE x in M*. *x* ∈ *S*)
then obtain *S* **where** *S*[*intro*]: *countable S* **and** *ae*: *AE x in M*. *x* ∈ *S*
by *auto*
then have *emeasure M* (⋃ *x*∈{*x*∈*S*. *emeasure M* {*x*} ≠ *0*}. {*x*}) =
(∫⁺ *x*. *emeasure M* {*x*} ∗ *indicator* {*x*∈*S*. *emeasure M* {*x*} ≠ *0*} *x* ∂*count-space UNIV*)
by (*subst emeasure-UN-countable*)
(*auto simp*: *disjoint-family-on-def nn-integral-restrict-space*[*symmetric*] *restrict-count-space*)
also have ... = (∫⁺ *x*. *emeasure M* {*x*} ∗ *indicator S x* ∂*count-space UNIV*)
by (*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *emeasure M* (⋃ *x*∈*S*. {*x*})
by (*subst emeasure-UN-countable*)
(*auto simp*: *disjoint-family-on-def nn-integral-restrict-space*[*symmetric*] *restrict-count-space*)
also have ... = *emeasure M* (*space M*)
using *ae* **by** (*intro emeasure-eq-AE*) *auto*
finally have *emeasure M* {*x* ∈ *space M*. *x*∈*S* ∧ *emeasure M* {*x*} ≠ *0*} = *emeasure M* (*space M*)
by (*simp add*: *emeasure-single-in-space cong*: *rev-conj-cong*)
with *finite-measure-compl*[*of* {*x* ∈ *space M*. *x*∈*S* ∧ *emeasure M* {*x*} ≠ *0*}]

have *AE x in M. x ∈ S ∧ emeasure M {x} ≠ 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. measure M {x} ≠ 0*
by (*auto simp*: *emeasure-eq-measure*)
qed (*auto intro*!: *exI*[*of* - {*x. measure M {x} ≠ 0*}] *countable-support*)

25.1 PMF as measure

typedef *'a pmf = {M :: 'a measure. prob-space M ∧ sets M = UNIV ∧ (AE x in M. measure M {x} ≠ 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*]: ⋀*y*. *y* ∈ *A* ⟹ (λ*x*. *f* (*x*, *y*)) ∈ *measurable M L*
shows *f* ∈ *measurable* (*M* ⨂$_M$ *restrict-space* (*measure-pmf N*) *A*) *L* (**is** *f* ∈ *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*=λ*a 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*]: ⋀*x*. *x* ∈ *A* ⟹ (λ*y*. *f* (*x*, *y*)) ∈ *measurable N L*
shows *f* ∈ *measurable* (*restrict-space* (*measure-pmf M*) *A* ⨂$_M$ *N*) *L*
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*=λ*a b*. *f* (*a*, *snd b*) **and** *g*=*fst* **and** *I*=*A*,
unfolded prod.collapse] *assms*)
measurable
qed

lift-definition *pmf* :: *'a pmf* ⇒ *'a* ⇒ *real* **is** λ*M x*. *measure M* {*x*} .

lift-definition *set-pmf* :: *'a pmf* ⇒ *'a set* **is** λ*M*. {*x*. *measure M* {*x*} ≠ *0*} .
declare [[*coercion set-pmf*]]

lemma *AE-measure-pmf*: *AE x in* (*M*::*'a pmf*). *x* ∈ *M*
by *transfer simp*

lemma *emeasure-pmf-single-eq-zero-iff*:
fixes *M* :: *'a pmf*
shows *emeasure M* {*y*} = *0* ⟷ *y* ∉ *M*
unfolding *set-pmf.rep-eq* **by** (*simp add*: *measure-pmf.emeasure-eq-measure*)

lemma *AE-measure-pmf-iff*: (*AE x in measure-pmf M*. *P x*) ⟷ (∀ *y*∈*M*. *P y*)
using *AE-measure-singleton*[*of M*] *AE-measure-pmf*[*of M*]
by (*auto simp*: *set-pmf.rep-eq*)

lemma *AE-pmfI*: (⋀*y*. *y* ∈ *set-pmf M* ⟹ *P y*) ⟹ *almost-everywhere* (*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* ∈ *set-pmf p* ⟹ *0* < *pmf p x*
by *transfer* (*simp add*: *less-le*)

lemma *pmf-nonneg*[*simp*]: *0* ≤ *pmf p x*
by *transfer simp*

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 *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-setsum-singleton*) (*auto simp*: *emeasure-pmf-single pmf-nonneg*)

lemma *measure-measure-pmf-finite*: *finite S* ⟹ *measure* (*measure-pmf M*) *S* = *setsum* (*pmf M*) *S*
using *emeasure-measure-pmf-finite*[*of S M*]
by (*simp add*: *measure-pmf.emeasure-eq-measure measure-nonneg setsum-nonneg pmf-nonneg*)

lemma *nn-integral-measure-pmf-support*:
fixes *f* :: *'a* ⇒ *ennreal*
assumes *f*: *finite A* **and** *nn*: ⋀*x*. *x* ∈ *A* ⟹ *0* ≤ *f x* ⋀*x*. *x* ∈ *set-pmf M* ⟹ *x* ∉ *A* ⟹ *f x* = *0*
shows (∫⁺*x*. *f x* ∂*measure-pmf M*) = (∑ *x*∈*A*. *f x* ∗ *pmf M x*)
proof −
have (∫⁺*x*. *f x* ∂*M*) = (∫⁺*x*. *f x* ∗ *indicator A x* ∂*M*)
using *nn* **by** (*intro nn-integral-cong-AE*) (*auto simp*: *AE-measure-pmf-iff split*: *split-indicator*)
also have ... = (∑ *x*∈*A*. *f x* ∗ *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* ⇒ *ennreal*
assumes *f*: *finite* (*set-pmf M*) **and** *nn*: ⋀*x*. *x* ∈ *set-pmf M* ⟹ *0* ≤ *f x*
shows (∫⁺*x*. *f x* ∂*measure-pmf M*) = (∑ *x*∈*set-pmf M*. *f x* ∗ *pmf M x*)
using *assms* **by** (*intro nn-integral-measure-pmf-support*) *auto*

lemma *integrable-measure-pmf-finite*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *finite* (*set-pmf M*) ⟹ *integrable M f*
by (*auto intro*!: *integrableI-bounded simp*: *nn-integral-measure-pmf-finite ennreal-mult-less-top*)

lemma *integral-measure-pmf*:
assumes [*simp*]: *finite A* **and** ⋀*a*. *a* ∈ *set-pmf M* ⟹ *f a* ≠ *0* ⟹ *a* ∈ *A*
shows (∫ *x*. *f x* ∂*measure-pmf M*) = (∑ *a*∈*A*. *f a* ∗ *pmf M a*)
proof −
have (∫ *x*. *f x* ∂*measure-pmf M*) = (∫ *x*. *f x* ∗ *indicator A x* ∂*measure-pmf M*)
using *assms*(*2*) **by** (*intro integral-cong-AE*) (*auto split*: *split-indicator simp*: *AE-measure-pmf-iff*)

also have ... = $(\sum a{\in}A.\ f\ a * pmf\ M\ a)$
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 $(\int^+ x.\ pmf\ M\ x\ \partial count\text{-}space\ X) = (\int^+ x.\ pmf\ M\ x\ \partial count\text{-}space\ (M \cap X))$
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* $(M \cap X)$) (*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*: $(\int x.\ pmf\ M\ x\ \partial count\text{-}space\ X) = measure\ M\ X$
proof −
have $(\int x.\ pmf\ M\ x\ \partial count\text{-}space\ X) = (\int^+ x.\ pmf\ M\ x\ \partial count\text{-}space\ X)$
by (*simp add*: *pmf-nonneg integrable-pmf nn-integral-eq-integral*)
also have ... = $(\int^+ x.\ emeasure\ M\ \{x\}\ \partial count\text{-}space\ (X \cap M))$
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* $(X \cap M)$
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*:
$(f::'a \Rightarrow 'b::\{banach, second\text{-}countable\text{-}topology\}) \in borel\text{-}measurable\ (count\text{-}space\ UNIV) \Longrightarrow$
$(\int x.\ f\ x\ \partial measure\text{-}pmf\ M) = (\int x.\ f\ x\ \partial restrict\text{-}space\ M\ M)$
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*=λ*x*. *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*)

25.2 Monad Interpretation

lemma *measurable-measure-pmf*[*measurable*]:
(λ*x*. *measure-pmf* (*M x*)) ∈ *measurable* (*count-space UNIV*) (*subprob-algebra* (*count-space UNIV*))
by (*auto simp*: *space-subprob-algebra intro*!: *prob-space-imp-subprob-space*) *unfold-locales*

lemma *bind-measure-pmf-cong*:
assumes ⋀*x*. *A x* ∈ *space* (*subprob-algebra N*) ⋀*x*. *B x* ∈ *space* (*subprob-algebra N*)
assumes ⋀*i*. *i* ∈ *set-pmf x* ⟹ *A i* = *B i*
shows *bind* (*measure-pmf x*) *A* = *bind* (*measure-pmf x*) *B*
proof (*rule measure-eqI*)
show *sets* (*measure-pmf x* ⤜ *A*) = *sets* (*measure-pmf x* ⤜ *B*)
using *assms* **by** (*subst* (*1 2*) *sets-bind*) (*auto simp*: *space-subprob-algebra*)
next
fix *X* **assume** *X* ∈ *sets* (*measure-pmf x* ⤜ *A*)
then have *X*: *X* ∈ *sets N*
using *assms* **by** (*subst* (*asm*) *sets-bind*) (*auto simp*: *space-subprob-algebra*)
show *emeasure* (*measure-pmf x* ⤜ *A*) *X* = *emeasure* (*measure-pmf x* ⤜ *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 pmf* ⇒ (*'a* ⇒ *'b pmf*) ⇒ *'b pmf* **is** *bind*
proof (*clarify*, *intro conjI*)
fix *f* :: *'a measure* **and** *g* :: *'a* ⇒ *'b measure*
assume *prob-space f*
then interpret *f*: *prob-space f* **.**
assume *sets f* = *UNIV* **and** *ae-f*: *AE x in f*. *measure f* {*x*} ≠ *0*
then have *s-f*[*simp*]: *sets f* = *sets* (*count-space UNIV*)
by *simp*
assume *g*: ⋀*x*. *prob-space* (*g x*) ∧ *sets* (*g x*) = *UNIV* ∧ (*AE y in g x*. *measure* (*g x*) {*y*} ≠ *0*)
then have *g*: ⋀*x*. *prob-space* (*g x*) **and** *s-g*[*simp*]: ⋀*x*. *sets* (*g x*) = *sets* (*count-space*

```
UNIV)
    and ae-g: ⋀x. AE y in g x. measure (g x) {y} ≠ 0
    by auto

  have [measurable]: g ∈ measurable f (subprob-algebra (count-space UNIV))
   by (auto simp: measurable-def space-subprob-algebra prob-space-imp-subprob-space
g)

  show prob-space (f ⤜ g)
    using g by (intro f.prob-space-bind[where S=count-space UNIV]) auto
  then interpret fg: prob-space f ⤜ g .
  show [simp]: sets (f ⤜ 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 ⤜ g. measure (f ⤜ g) {x} ≠ 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

lemma ennreal-pmf-bind: pmf (bind-pmf N f) i = (∫⁺x. pmf (f x) i ∂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 = (∫ x. pmf (f x) i ∂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 (λx. c) = c
  by transfer (simp add: bind-const' prob-space-imp-subprob-space)

lemma set-bind-pmf[simp]: set-pmf (bind-pmf M N) = (⋃M∈set-pmf M. set-pmf
(N M))
proof −
  have set-pmf (bind-pmf M N) = {x. ennreal (pmf (bind-pmf M N) x) ≠ 0}
    by (simp add: set-pmf-eq pmf-nonneg)
  also have ... = (⋃M∈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*:
assumes $p = q$
shows $(\bigwedge x.\ x \in$ *set-pmf* $q \Longrightarrow f\ x = g\ x) \Longrightarrow$ *bind-pmf* $p\ f =$ *bind-pmf* $q\ g$
unfolding $\langle p = q \rangle$[*symmetric*] *measure-pmf-inject*[*symmetric*] *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*:
$p = q \Longrightarrow (\bigwedge x.\ x \in$ *set-pmf* q =*simp*=> $f\ x = g\ x) \Longrightarrow$ *bind-pmf* $p\ f =$ *bind-pmf* $q\ g$
by (*simp add*: *simp-implies-def cong*: *bind-pmf-cong*)

lemma *measure-pmf-bind*: *measure-pmf* (*bind-pmf* $M\ f$) = (*measure-pmf* $M \gg\!=$ ($\lambda x.$ *measure-pmf* ($f\ x$)))
by *transfer simp*

lemma *nn-integral-bind-pmf*[*simp*]: $(\int^+ x.\ f\ x\ \partial$*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*]: *emeasure* (*bind-pmf M N*) $X = (\int^+ x.$ *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* :: $'a \Rightarrow$ $'a$ *pmf* **is** *return* (*count-space UNIV*)
by (*auto intro*!: *prob-space-return simp*: *AE-return measure-return*)

lemma *bind-return-pmf*: *bind-pmf* (*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*]: *set-pmf* (*return-pmf* x) = $\{x\}$
by *transfer* (*auto simp add*: *measure-return split*: *split-indicator*)

lemma *bind-return-pmf'*: *bind-pmf N return-pmf = N*
proof (*transfer*, *clarify*)
fix *N* :: *'a measure* **assume** *sets N = UNIV* **then show** *N* ⤜ *return* (*count-space UNIV*) = *N*
by (*subst return-sets-cong*[**where** *N=N*]) (*simp-all add*: *bind-return'*)
qed

lemma *bind-assoc-pmf*: *bind-pmf* (*bind-pmf A B*) *C* = *bind-pmf A* (λ*x*. *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* (λ*x*. *return-pmf* (*f x*))

lemma *map-bind-pmf*: *map-pmf f* (*bind-pmf M g*) = *bind-pmf M* (λ*x*. *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* (λ*x*. *g* (*f x*))
by (*simp add*: *map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *map-pmf-transfer*[*transfer-rule*]:
rel-fun op = (*rel-fun cr-pmf cr-pmf*) (λ*f M*. *distr M* (*count-space UNIV*) *f*) *map-pmf*
proof −
have *rel-fun op* = (*rel-fun pmf-as-measure.cr-pmf pmf-as-measure.cr-pmf*)
(λ*f M*. *M* ⤜ (*return* (*count-space UNIV*) *o f*)) *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* (λ*x*. *x*) = (λ*x*. *x*)
using *map-pmf-id* **unfolding** *id-def* **.**

lemma *map-pmf-compose*: *map-pmf* (*f* ∘ *g*) = *map-pmf f* ∘ *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* (λ*x*. *f* (*g x*)) *M*
using *map-pmf-compose*[*of f g*] **by** (*simp add*: *comp-def*)

lemma *map-pmf-cong*: *p* = *q* ⟹ (⋀*x*. *x* ∈ *set-pmf q* ⟹ *f x* = *g x*) ⟹ *map-pmf f p* = *map-pmf g q*
unfolding *map-pmf-def* **by** (*rule bind-pmf-cong*) *auto*

lemma *pmf-set-map*: *set-pmf* ∘ *map-pmf f* = *op* ‘ *f* ∘ *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*]: (∫⁺*x*. *f x* ∂*map-pmf g M*) = (∫⁺*x*. *f* (*g x*) ∂*M*)
unfolding *map-pmf-rep-eq* **by** (*intro nn-integral-distr*) *auto*

lemma *ennreal-pmf-map*: *pmf* (*map-pmf f p*) *x* = (∫⁺ *y*. *indicator* (*f* −‘ {*x*}) *y* ∂*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*: (∫⁺ *x*. *pmf p x* ∂*count-space A*) = *emeasure* (*measure-pmf*

p) *A*
proof −
have (∫⁺ *x*. *pmf p x* ∂*count-space A*) = (∫⁺ *x*. *pmf p x* ∂*count-space* (*A* ∩ *set-pmf p*))
by(*auto simp add*: *nn-integral-count-space-indicator indicator-def set-pmf-iff intro*: *nn-integral-cong*)
also have ... = *emeasure* (*measure-pmf p*) (⋃*x*∈*A* ∩ *set-pmf p*. {*x*})
by(*subst emeasure-UN-countable*)(*auto simp add*: *emeasure-pmf-single disjoint-family-on-def*)
also have ... = *emeasure* (*measure-pmf p*) ((⋃*x*∈*A* ∩ *set-pmf p*. {*x*}) ∪ {*x*. *x* ∈ *A* ∧ *x* ∉ *set-pmf p*})
by(*rule emeasure-Un-null-set*[*symmetric*])(*auto intro*: *in-null-sets-measure-pmfI*)
also have ... = *emeasure* (*measure-pmf p*) *A*
by(*auto intro*: *arg-cong2*[**where** *f*=*emeasure*])
finally show *?thesis* **.**
qed

lemma *integral-map-pmf*[*simp*]:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second-countable-topology*}
shows *integral*^{L} (*map-pmf g p*) *f* = *integral*^{L} *p* (λ*x*. *f* (*g x*))
by (*simp add*: *integral-distr map-pmf-rep-eq*)

lemma *map-return-pmf* [*simp*]: *map-pmf f* (*return-pmf x*) = *return-pmf* (*f x*)
by *transfer* (*simp add*: *distr-return*)

lemma *map-pmf-const*[*simp*]: *map-pmf* (λ-. *c*) *M* = *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* ≤ *f x* ⟹ (∫⁺*x*. *f x* ∂*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 *return-pmf-inj*[*simp*]: *return-pmf x* = *return-pmf y* ⟷ *x* = *y*
by (*metis insertI1 set-return-pmf singletonD*)

lemma *map-pmf-eq-return-pmf-iff*:
map-pmf f p = *return-pmf x* ⟷ (∀ *y* ∈ *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 ∀ *y* ∈ *set-pmf p*. *f y* = *x* **by** *auto*
next
assume ∀ *y* ∈ *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* (λ*x*. *bind-pmf B* (λ*y*. *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*[**where** *A*={*b*}])
apply (*auto simp*: *indicator-eq-0-iff*)
apply (*subst integral-measure-pmf*[**where** *A*={*a*}])
apply (*auto simp*: *indicator-eq-0-iff setsum-nonneg-eq-0-iff pmf-nonneg*)
done

lemma *set-pair-pmf*[*simp*]: *set-pmf* (*pair-pmf A B*) = *set-pmf A* × *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 ∈ *space* (*subprob-algebra* (*count-space UNIV*))
by (*simp add*: *space-subprob-algebra*) *intro-locales*

lemma *nn-integral-pair-pmf'*: (∫⁺*x*. *f x* ∂*pair-pmf A B*) = (∫⁺*a*. ∫⁺*b*. *f* (*a*, *b*) ∂*B* ∂*A*)
proof −
have (∫⁺*x*. *f x* ∂*pair-pmf A B*) = (∫⁺*x*. *f x* ∗ *indicator* (*A* × *B*) *x* ∂*pair-pmf A B*)
by (*auto simp*: *AE-measure-pmf-iff intro*!: *nn-integral-cong-AE*)
also have ... = (∫⁺*a*. ∫⁺*b*. *f* (*a*, *b*) ∗ *indicator* (*A* × *B*) (*a*, *b*) ∂*B* ∂*A*)
by (*simp add*: *pair-pmf-def*)
also have ... = (∫⁺*a*. ∫⁺*b*. *f* (*a*, *b*) ∂*B* ∂*A*)
by (*auto intro*!: *nn-integral-cong-AE simp*: *AE-measure-pmf-iff*)
finally show *?thesis* **.**
qed

lemma *bind-pair-pmf*:
assumes *M*[*measurable*]: *M* ∈ *measurable* (*count-space UNIV* ⨂$_M$ *count-space UNIV*) (*subprob-algebra N*)
shows *measure-pmf* (*pair-pmf A B*) ⤜ *M* = (*measure-pmf A* ⤜ (λ*x*. *measure-pmf B* ⤜ (λ*y*. *M* (*x*, *y*))))
(**is** *?L* = *?R*)
proof (*rule measure-eqI*)
have *M'*[*measurable*]: *M* ∈ *measurable* (*pair-pmf A B*) (*subprob-algebra N*)
using *M*[*THEN measurable-space*] **by** (*simp-all add*: *space-pair-measure*)

note *measurable-bind*[**where** *N*=*count-space UNIV*, *measurable*]
note *measure-pmf-in-subprob-space*[*simp*]

have *sets-eq-N*: *sets ?L* = *N*
by (*subst sets-bind*[*OF sets-kernel*[*OF M'*]]) *auto*
show *sets ?L* = *sets ?R*

```
    using measurable-space[OF M]
    by (simp add: sets-eq-N space-pair-measure space-subprob-algebra)
  fix X assume X ∈ sets ?L
  then have X[measurable]: X ∈ sets N
    unfolding sets-eq-N .
  then show emeasure ?L X = emeasure ?R X
    apply (simp add: emeasure-bind[OF - M' X])
    apply (simp add: nn-integral-bind[where B=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 ⟹ (∫⁺x. pmf p (f x) ∂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 ≤ 0 ⟷ 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 ⟷ p ∉ set-pmf M
  unfolding set-pmf-iff by simp

lemma pmf-map-inj: inj-on f (set-pmf M) ⟹ x ∈ set-pmf M ⟹ 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 pmf-map-inj': inj f ⟹ pmf (map-pmf f M) (f x) = pmf M x
apply(cases x ∈ 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 *pmf-map-outside*: $x \notin f$ ' *set-pmf* $M \Longrightarrow$ *pmf* (*map-pmf* f M) $x = 0$
unfolding *pmf-eq-0-set-pmf* **by** *simp*

25.3 PMFs as function

context
fixes $f :: 'a \Rightarrow$ *real*
assumes *nonneg*: $\bigwedge x.\ 0 \leq f\ x$
assumes *prob*: $(\int^+ x.\ f\ x\ \partial$*count-space UNIV*$) = 1$
begin

lift-definition *embed-pmf* :: $'a$ *pmf* **is** *density* (*count-space UNIV*) (*ennreal* $\circ$ f)
proof (*intro conjI*)
have $*$[*simp*]: $\bigwedge x\ y.$ *ennreal* $(f\ y) *$ *indicator* $\{x\}\ y =$ *ennreal* $(f\ x) *$ *indicator* $\{x\}\ y$
by (*simp split*: *split-indicator*)
show *AE x in density* (*count-space UNIV*) (*ennreal* $\circ$ f).
measure (*density* (*count-space UNIV*) (*ennreal* $\circ$ f)) $\{x\} \neq 0$
by (*simp add*: *AE-density nonneg measure-def emeasure-density max-def*)
show *prob-space* (*density* (*count-space UNIV*) (*ennreal* $\circ$ f))
by *standard* (*simp add*: *emeasure-density prob*)
qed *simp*

lemma *pmf-embed-pmf*: *pmf embed-pmf* $x = f\ x$
proof *transfer*
have $*$[*simp*]: $\bigwedge x\ y.$ *ennreal* $(f\ y) *$ *indicator* $\{x\}\ y =$ *ennreal* $(f\ x) *$ *indicator* $\{x\}\ y$
by (*simp split*: *split-indicator*)
fix x **show** *measure* (*density* (*count-space UNIV*) (*ennreal* $\circ$ f)) $\{x\} = f\ x$
by *transfer* (*simp add*: *measure-def emeasure-density nonneg max-def*)
qed

lemma *set-embed-pmf*: *set-pmf embed-pmf* $= \{x.\ f\ x \neq 0\}$
by(*auto simp add*: *set-pmf-eq assms pmf-embed-pmf*)

end

lemma *embed-pmf-transfer*:
rel-fun (*eq-onp* $(\lambda f.\ (\forall x.\ 0 \leq f\ x) \wedge (\int^+ x.$ *ennreal* $(f\ x)\ \partial$*count-space UNIV*$) = 1)$) *pmf-as-measure.cr-pmf* $(\lambda f.$ *density* (*count-space UNIV*) (*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* $M\ \{x\} \neq 0$

assume *prob-space M* **then interpret** *prob-space M* **.**
show *M = density (count-space UNIV) (λx. ennreal (measure M {x}))*
proof *(rule measure-eqI)*
fix *A :: 'a set*
have *(∫⁺ x. ennreal (measure M {x}) ∗ indicator A x ∂count-space UNIV) = (∫⁺ x. emeasure M {x} ∗ indicator (A ∩ {x. measure M {x} ≠ 0}) x ∂count-space UNIV)*
by *(auto intro!: nn-integral-cong simp: emeasure-eq-measure split: split-indicator)*
also have *... = (∫⁺ x. emeasure M {x} ∂count-space (A ∩ {x. measure M {x} ≠ 0}))*
by *(subst nn-integral-restrict-space[symmetric]) (auto simp: restrict-count-space)*
also have *... = emeasure M (⋃x∈(A ∩ {x. measure M {x} ≠ 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) (λx. ennreal (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 ⇒ real. (∀x. 0 ≤ f x) ∧ (∫⁺x. ennreal (f x) ∂count-space UNIV) = 1}
unfolding *type-definition-def*
proof *safe*
fix *p :: 'a pmf*
have *(∫⁺ x. 1 ∂measure-pmf p) = 1*
using *measure-pmf.emeasure-space-1[of p]* **by** *simp*
then show *∗: (∫⁺ x. ennreal (pmf p x) ∂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 ⇒ real* **assume** *∀x. 0 ≤ f x (∫⁺x. f x ∂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: (∫⁺ x. f x ∂measure-pmf p) = ∫⁺ x. ennreal (pmf p x) ∗ f x ∂count-space UNIV*
by(*simp add: measure-pmf-eq-density nn-integral-density pmf-nonneg*)

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*) (λ*f*. {*x*. *f x* ≠ *0*}) *set-pmf*
  using ‹*bi-total A*›
  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*: (⋀*i*. *pmf M i* = *pmf N i*) ⟹ *M* = *N*
  by *transfer auto*

lemma *pmf-eq-iff*: *M* = *N* ⟷ (∀ *i*. *pmf M i* = *pmf N i*)
  by (*auto intro*: *pmf-eqI*)

lemma *bind-commute-pmf*: *bind-pmf A* (λ*x*. *bind-pmf B* (*C x*)) = *bind-pmf B* (λ*y*. *bind-pmf A* (λ*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.emeasure-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.emeasure-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 (∫ *x*. ∫ *y*. *pmf* (*C x y*) *i* ∂*B* ∂*A*) = (∫ *x*. (∫ *y*. *pmf* (*C x y*) *i* ∂*restrict-space B B*) ∂*A*)
    by (*rule integral-cong*) (*auto intro*!: *integral-pmf-restrict*)
  also have ... = (∫ *x*. (∫ *y*. *pmf* (*C x y*) *i* ∂*restrict-space B B*) ∂*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 ... = ($\int$ *y*. $\int$ *x*. *pmf* (*C x y*) *i* ∂*restrict-space A A* ∂*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 ... = ($\int$ *y*. $\int$ *x*. *pmf* (*C x y*) *i* ∂*restrict-space A A* ∂*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 ... = ($\int$ *y*. $\int$ *x*. *pmf* (*C x y*) *i* ∂*A* ∂*B*)
by (*rule integral-cong*) (*auto intro*!: *integral-pmf-restrict*[*symmetric*])
finally show ($\int$ *x*. $\int$ *y*. *pmf* (*C x y*) *i* ∂*B* ∂*A*) = ($\int$ *y*. $\int$ *x*. *pmf* (*C x y*) *i* ∂*A* ∂*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*]: ⋀*c d*. *indicator* (*apfst f* −‘ {(*a*, *b*)}) (*c*, *d*) = *indicator* (*f* −‘ {*a*}) *c* ∗ (*indicator* {*b*} *d*::*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*]: ⋀*c d*. *indicator* (*apsnd f* −‘ {(*a*, *b*)}) (*c*, *d*) = *indicator* {*a*} *c* ∗ (*indicator* (*f* −‘ {*b*}) *d*::*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* (λ(*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*: *pair-pmf* (*return-pmf x*) *y* = *map-pmf* (*Pair x*) *y*
by(*simp add*: *pair-pmf-def bind-return-pmf map-pmf-def*)

lemma *pair-return-pmf2*: *pair-pmf x* (*return-pmf y*) = *map-pmf* (λ*x*. (*x*, *y*)) *x*
by(*simp add*: *pair-pmf-def bind-return-pmf map-pmf-def*)

lemma *pair-pair-pmf*: *pair-pmf* (*pair-pmf u v*) *w* = *map-pmf* (λ(*x*, (*y*, *z*)). ((*x*, *y*), *z*)) (*pair-pmf u* (*pair-pmf v w*))
by(*simp add*: *pair-pmf-def bind-return-pmf map-pmf-def bind-assoc-pmf*)

lemma *pair-commute-pmf*: *pair-pmf x y* = *map-pmf* (λ(*x*, *y*). (*y*, *x*)) (*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*: *set-pmf p* ⊆ {*x*} ⟷ *p* = *return-pmf x*
proof(*intro iffI pmf-eqI*)
fix *i*
assume *x*: *set-pmf p* ⊆ {*x*}
hence ∗: *set-pmf p* = {*x*} **using** *set-pmf-not-empty*[*of p*] **by** *auto*
have *ennreal* (*pmf p x*) = ∫⁺ *i*. *indicator* {*x*} *i* ∂*p* **by**(*simp add*: *emeasure-pmf-single*)
also have ... = ∫⁺ *i*. *1* ∂*p* **by**(*rule nn-integral-cong-AE*)(*simp add*: *AE-measure-pmf-iff* ∗)
also have ... = *1* **by** *simp*
finally show *pmf p i* = *pmf* (*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*:
bind-pmf p f = *return-pmf x* ⟷ (∀ *y*∈*set-pmf p*. *f y* = *return-pmf x*)
(**is** *?lhs* ⟷ *?rhs*)
proof(*intro iffI strip*)
fix *y*
assume *y*: *y* ∈ *set-pmf p*
assume *?lhs*
hence *set-pmf* (*bind-pmf p f*) = {*x*} **by** *simp*
hence (⋃ *y*∈*set-pmf p*. *set-pmf* (*f y*)) = {*x*} **by** *simp*
hence *set-pmf* (*f y*) ⊆ {*x*} **using** *y* **by** *auto*
thus *f y* = *return-pmf x* **by**(*simp add*: *set-pmf-subset-singleton*)

next
assume ∗: *?rhs*
show *?lhs*
proof(*rule pmf-eqI*)
fix *i*
have *ennreal* (*pmf* (*bind-pmf p f*) *i*) = ∫⁺ *y*. *ennreal* (*pmf* (*f y*) *i*) ∂*p*
by (*simp add*: *ennreal-pmf-bind*)
also have ... = ∫⁺ *y*. *ennreal* (*pmf* (*return-pmf x*) *i*) ∂*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*: *pmf p False* = *1* − *pmf p 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*: *pmf p True* = *1* − *pmf p False*
by(*simp add*: *pmf-False-conv-True*)

25.4 Conditional Probabilities

lemma *measure-pmf-zero-iff*: *measure* (*measure-pmf p*) *s* = *0* ⟷ *set-pmf p* ∩ *s* = {}
by (*subst measure-pmf.prob-eq-0*) (*auto simp*: *AE-measure-pmf-iff*)

context
fixes *p* :: *'a pmf* **and** *s* :: *'a set*
assumes *not-empty*: *set-pmf p* ∩ *s* ≠ {}
begin

interpretation *pmf-as-measure* **.**

lemma *emeasure-measure-pmf-not-zero*: *emeasure* (*measure-pmf p*) *s* ≠ *0*
proof
assume *emeasure* (*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*: *measure* (*measure-pmf p*) *s* ≠ *0*
using *emeasure-measure-pmf-not-zero* **by** (*simp add*: *measure-pmf.emeasure-eq-measure measure-nonneg*)

lift-definition *cond-pmf* :: *'a 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 *AE x in uniform-measure* (*measure-pmf p*) *s*. *measure* (*uniform-measure* (*measure-pmf p*) *s*) {*x*} ≠ *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*: *pmf cond-pmf x* = (*if x* ∈ *s then pmf p x* / *measure p s else 0*)
by *transfer* (*simp add*: *emeasure-measure-pmf-not-zero pmf.rep-eq*)

lemma *set-cond-pmf*[*simp*]: *set-pmf cond-pmf* = *set-pmf p* ∩ *s*
by (*auto simp add*: *set-pmf-iff pmf-cond measure-measure-pmf-not-zero split*: *if-split-asm*)

end

lemma *cond-map-pmf*:
assumes *set-pmf p* ∩ *f* −‘ *s* ≠ {}
shows *cond-pmf* (*map-pmf f p*) *s* = *map-pmf f* (*cond-pmf p* (*f* −‘ *s*))
proof −
have ∗: *set-pmf* (*map-pmf f p*) ∩ *s* ≠ {}
using *assms* **by** *auto*
{ fix *x*
have *ennreal* (*pmf* (*map-pmf f* (*cond-pmf p* (*f* −‘ *s*))) *x*) =
emeasure p (*f* −‘ *s* ∩ *f* −‘ {*x*}) / *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* ∩ *f* −‘ {*x*} = (*if x* ∈ *s then f* −‘ {*x*} *else* {})
by *auto*
also have *emeasure p* (*if x* ∈ *s then f* −‘ {*x*} *else* {}) / *emeasure p* (*f* −‘ *s*)
=
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 *ennreal* (*pmf* (*cond-pmf* (*map-pmf f p*) *s*) *x*) = *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*]: $\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow$ *set-pmf q* $\cap \{y.\ R\ x\ y\} \neq \{\}$
assumes [*simp*]: $\bigwedge y.\ y \in$ *set-pmf q* $\Longrightarrow$ *set-pmf p* $\cap \{x.\ R\ x\ y\} \neq \{\}$
assumes [*simp*]: $\bigwedge x\ y.\ x \in$ *set-pmf p* $\Longrightarrow\ y \in$ *set-pmf q* $\Longrightarrow R\ x\ y \Longrightarrow$ *measure q* $\{y.\ R\ x\ y\} =$ *measure p* $\{x.\ R\ x\ y\}$
shows *bind-pmf p* $(\lambda x.$ *cond-pmf q* $\{y.\ R\ x\ y\}) = q$
proof (*rule pmf-eqI*)
fix *i*
have *ennreal* (*pmf* (*bind-pmf p* $(\lambda x.$ *cond-pmf q* $\{y.\ R\ x\ y\})$) *i*) =
$(\int^+ x.$ *ennreal* (*pmf q i* / *measure p* $\{x.\ R\ x\ i\}) *$ *ennreal* (*indicator* $\{x.\ R\ x\ i\}\ x)\ \partial p)$
by (*auto simp add*: *ennreal-pmf-bind AE-measure-pmf-iff pmf-cond pmf-eq-0-set-pmf pmf-nonneg measure-nonneg intro*!: *nn-integral-cong-AE*)
also have ... = (*pmf q i* $*$ *measure p* $\{x.\ R\ x\ i\})$ / *measure p* $\{x.\ R\ x\ i\}$
by (*simp add*: *pmf-nonneg measure-nonneg zero-ennreal-def*[*symmetric*] *ennreal-indicator nn-integral-cmult measure-pmf.emeasure-eq-measure ennreal-mult*[*symmetric*])
also have ... = *pmf q i*
by (*cases pmf q i* = *0*) (*simp-all add*: *pmf-eq-0-set-pmf measure-measure-pmf-not-zero pmf-nonneg*)
finally show *pmf* (*bind-pmf p* $(\lambda x.$ *cond-pmf q* $\{y.\ R\ x\ y\}))$ *i* = *pmf q i*
by (*simp add*: *pmf-nonneg*)
qed

25.5 Relator

inductive *rel-pmf* :: $('a \Rightarrow 'b \Rightarrow bool) \Rightarrow$ *'a pmf* $\Rightarrow$ *'b pmf* $\Rightarrow$ *bool*
for *R p q*
where
$[\![\bigwedge x\ y.\ (x, y) \in$ *set-pmf pq* $\Longrightarrow R\ x\ y$;
map-pmf fst pq = *p*; *map-pmf snd pq* = *q* $]\!]$
$\Longrightarrow$ *rel-pmf R p q*

lemma *rel-pmfI*:
assumes *R*: *rel-set R* (*set-pmf p*) (*set-pmf q*)
assumes *eq*: $\bigwedge x\ y.\ x \in$ *set-pmf p* $\Longrightarrow\ y \in$ *set-pmf q* $\Longrightarrow R\ x\ y \Longrightarrow$
measure p $\{x.\ R\ x\ y\} =$ *measure q* $\{y.\ R\ x\ y\}$
shows *rel-pmf R p q*
proof
let *?pq* = *bind-pmf p* $(\lambda x.$ *bind-pmf* (*cond-pmf q* $\{y.\ R\ x\ y\})$ $(\lambda y.$ *return-pmf* $(x, y)))$
have $\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow$ *set-pmf q* $\cap \{y.\ R\ x\ y\} \neq \{\}$
using *R* **by** (*auto simp*: *rel-set-def*)
then show $\bigwedge x\ y.\ (x, y) \in$ *set-pmf ?pq* $\Longrightarrow R\ x\ y$
by *auto*

```
  show map-pmf fst ?pq = p
    by (simp add: map-bind-pmf bind-return-pmf')

  show map-pmf snd ?pq = q
    using R 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: rel-pmf R p q ⟹ rel-set R (set-pmf p) (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: ⋀a b. R a b ⟹ R a y ⟷ R x b
  assumes x ∈ set-pmf p y ∈ set-pmf q
  shows measure p {x. R x y} = measure q {y. R x y}
proof −
  from rel-R obtain pq where pq: ⋀x y. (x, y) ∈ set-pmf pq ⟹ R x y
    and eq: p = map-pmf fst pq q = map-pmf snd pq
    by (auto elim: rel-pmf.cases)
  have measure p {x. R x y} = measure pq {x. R (fst x) y}
    by (simp add: eq map-pmf-rep-eq measure-distr)
  also have ... = measure pq {y. R x (snd y)}
    by (intro measure-pmf.finite-measure-eq-AE
       (auto simp: AE-measure-pmf-iff R dest!: pq)
  also have ... = measure q {y. R x y}
    by (simp add: eq map-pmf-rep-eq measure-distr)
  finally show measure p {x. R x y} = measure q {y. R x y} .
qed

lemma rel-pmf-measureD:
  assumes rel-pmf R p q
  shows measure (measure-pmf p) A ≤ measure (measure-pmf q) {y. ∃x∈A. R x
y} (is ?lhs ≤ ?rhs)
using assms
proof cases
  fix pq
  assume R: ⋀x y. (x, y) ∈ set-pmf pq ⟹ R x y
    and p[symmetric]: map-pmf fst pq = p
    and q[symmetric]: map-pmf snd pq = q
  have ?lhs = measure (measure-pmf pq) (fst −‘ A) by(simp add: p)
  also have ... ≤ measure (measure-pmf pq) {y. ∃x∈A. R x (snd y)}
   by(rule measure-pmf.finite-measure-mono-AE)(auto 4 3 simp add: AE-measure-pmf-iff
dest: R)
  also have ... = ?rhs by(simp add: q)
  finally show ?thesis .
qed
```

lemma *rel-pmf-iff-measure*:
assumes *symp R transp R*
shows *rel-pmf R p q* ⟷
rel-set R (*set-pmf p*) (*set-pmf q*) ∧
(∀ *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*: *equiv X R* ⟹ *A* ∈ *X* // *R* ⟹ *x* ∈ *A* ⟹ *A* = *R* `` {*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* ∈ *UNIV* // *?R* **and** *R*: *rel-set R* (*set-pmf p*) (*set-pmf q*)
assume *eq*: ∀ *x*∈*set-pmf p*. ∀ *y*∈*set-pmf q*. *R x y* ⟶ *measure p* {*x*. *R x y*} = *measure q* {*y*. *R x y*}

show *measure p C* = *measure q C*
proof *cases*
assume *p* ∩ *C* = {}
moreover then have *q* ∩ *C* = {}
using *quotient-rel-set-disjoint*[*OF assms C R*] **by** *simp*
ultimately show *?thesis*
unfolding *measure-pmf-zero-iff*[*symmetric*] **by** *simp*
next
assume *p* ∩ *C* ≠ {}
moreover then have *q* ∩ *C* ≠ {}
using *quotient-rel-set-disjoint*[*OF assms C R*] **by** *simp*
ultimately obtain *x y* **where** *in-set*: *x* ∈ *set-pmf p y* ∈ *set-pmf q* **and** *in-C*: *x* ∈ *C y* ∈ *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 measure p {x. R x y} = 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 : natLeq rel: rel-pmf
proof −
```

show *map-pmf id = id* **by** (*rule map-pmf-id*)
show $\bigwedge f\ g.$ *map-pmf* $(f \circ g) =$ *map-pmf f* $\circ$ *map-pmf g* **by** (*rule map-pmf-compose*)
show $\bigwedge f\ g{::}'a \Rightarrow 'b.\ \bigwedge p.\ (\bigwedge x.\ x \in$ *set-pmf p* $\Longrightarrow f\ x = g\ x) \Longrightarrow$ *map-pmf f p = map-pmf g p*
by (*intro map-pmf-cong refl*)

show $\bigwedge f{::}'a \Rightarrow 'b.$ *set-pmf* $\circ$ *map-pmf f = op* ‘ *f* $\circ$ *set-pmf*
by (*rule pmf-set-map*)

show (*card-of* (*set-pmf p*), *natLeq*) $\in$ *ordLeq* **for** *p* :: *'s pmf*
proof −
have (*card-of* (*set-pmf p*), *card-of* (*UNIV* :: *nat set*)) $\in$ *ordLeq*
by (*rule card-of-ordLeqI*[**where** *f=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* **.**
qed

show $\bigwedge R.$ *rel-pmf R* = $(\lambda x\ y.\ \exists z.$ *set-pmf z* $\subseteq \{(x, y).\ R\ x\ y\} \wedge$
map-pmf fst z = x $\wedge$ *map-pmf snd z = y*)
by (*auto simp add*: *fun-eq-iff rel-pmf.simps*)

show *rel-pmf R OO rel-pmf S* $\leq$ *rel-pmf* (*R OO S*)
for *R* :: *'a* $\Rightarrow$ *'b* $\Rightarrow$ *bool* **and** *S* :: *'b* $\Rightarrow$ *'c* $\Rightarrow$ *bool*
proof −
{ fix *p q r*
assume *pq*: *rel-pmf R p q*
and *qr*:*rel-pmf S q r*
from *pq* **obtain** *pq* **where** *pq*: $\bigwedge x\ y.$ $(x, y) \in$ *set-pmf pq* $\Longrightarrow R\ x\ y$
and *p*: *p = map-pmf fst pq* **and** *q*: *q = map-pmf snd pq* **by** *cases auto*
from *qr* **obtain** *qr* **where** *qr*: $\bigwedge y\ z.$ $(y, z) \in$ *set-pmf qr* $\Longrightarrow S\ y\ z$
and *q′*: *q = map-pmf fst qr* **and** *r*: *r = map-pmf snd qr* **by** *cases auto*

def *pr* $\equiv$ *bind-pmf pq* $(\lambda xy.$ *bind-pmf* (*cond-pmf qr* $\{yz.$ *fst yz = snd xy*$\}$) $(\lambda yz.$ *return-pmf* (*fst xy*, *snd yz*)))
have *pr-welldefined*: $\bigwedge y.\ y \in q \Longrightarrow qr \cap \{yz.$ *fst yz = y*$\} \neq \{\}$
by (*force simp*: *q′*)

have *rel-pmf* (*R 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 pr-welldefined pr-def split-beta*)
with *pq qr* **show** (*R OO S*) *x z*
by *blast*
next
have *map-pmf snd pr = map-pmf snd* (*bind-pmf q* $(\lambda y.$ *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 (*fact natLeq-card-order natLeq-cinfinite*)+

lemma *map-pmf-idI*: (⋀*x*. *x* ∈ *set-pmf p* ⟹ *f x* = *x*) ⟹ *map-pmf f p* = *p*
by(*simp cong*: *pmf.map-cong*)

lemma *rel-pmf-conj*[*simp*]:
rel-pmf (λ*x y*. *P* ∧ *Q x y*) *x y* ⟷ *P* ∧ *rel-pmf Q x y*
rel-pmf (λ*x y*. *Q x y* ∧ *P*) *x y* ⟷ *P* ∧ *rel-pmf Q x y*
using *set-pmf-not-empty* **by** (*fastforce simp*: *pmf.in-rel subset-eq*)+

lemma *rel-pmf-top*[*simp*]: *rel-pmf top* = *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*: *rel-pmf R* (*return-pmf x*) *M* ⟷ (∀ *a*∈*M*. *R x a*)
proof *safe*
fix *a* **assume** *a* ∈ *M rel-pmf R* (*return-pmf x*) *M*
then obtain *pq* **where** ∗: ⋀*a b*. (*a*, *b*) ∈ *set-pmf pq* ⟹ *R a b*
and *eq*: *return-pmf x* = *map-pmf fst pq M* = *map-pmf snd pq*
by (*force elim*: *rel-pmf.cases*)
moreover have *set-pmf* (*return-pmf x*) = {*x*}
by *simp*
with ‹*a* ∈ *M*› **have** (*x*, *a*) ∈ *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*: *rel-pmf R M* (*return-pmf x*) ⟷ (∀ *a*∈*M*. *R a x*)
by (*subst pmf.rel-flip*[*symmetric*]) (*simp add*: *rel-pmf-return-pmf1*)

lemma *rel-return-pmf*[*simp*]: *rel-pmf R* (*return-pmf x1*) (*return-pmf x2*) = *R x1 x2*
unfolding *rel-pmf-return-pmf2 set-return-pmf* **by** *simp*

lemma *rel-pmf-False*[*simp*]: *rel-pmf* (λ*x y*. *False*) *x y* = *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'*) ⟷ *rel-pmf R A B* ∧ *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*: ⋀*a b c d*. ((*a*, *c*), (*b*, *d*)) ∈ *set-pmf pq* ⟹ *R a b* ∧ *S c d*
    and *eq*: *map-pmf fst pq* = *pair-pmf A A' map-pmf snd pq* = *pair-pmf B B'*
    by (*force elim*: *rel-pmf.cases*)
  show *rel-pmf R A B*
  proof (*rule rel-pmf.intros*)
    let *?f* = λ(*a*, *b*). (*fst a*, *fst b*)
    have [*simp*]: (λ*x*. *fst* (*?f x*)) = *fst o fst* (λ*x*. *snd* (*?f x*)) = *fst o 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*) ∈ *set-pmf* (*map-pmf ?f pq*)
    then obtain *c d* **where** ((*a*, *c*), (*b*, *d*)) ∈ *set-pmf pq*
      by *auto*
    from *pq*[*OF this*] **show** *R a b* **..**
  qed
  show *rel-pmf S A' B'*
  proof (*rule rel-pmf.intros*)
    let *?f* = λ(*a*, *b*). (*snd a*, *snd b*)
    have [*simp*]: (λ*x*. *fst* (*?f x*)) = *snd o fst* (λ*x*. *snd* (*?f x*)) = *snd o 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*) ∈ *set-pmf* (*map-pmf ?f pq*)
    then obtain *a b* **where** ((*a*, *c*), (*b*, *d*)) ∈ *set-pmf pq*
      by *auto*
    from *pq*[*OF this*] **show** *S c d* **..**
  qed
next
  assume *rel-pmf R A B rel-pmf S A' B'*
  then obtain *Rpq Spq*
    where *Rpq*: ⋀*a b*. (*a*, *b*) ∈ *set-pmf Rpq* ⟹ *R a b*
      map-pmf fst Rpq = *A map-pmf snd Rpq* = *B*
     and *Spq*: ⋀*a b*. (*a*, *b*) ∈ *set-pmf Spq* ⟹ *S a b*
      map-pmf fst Spq = *A' map-pmf snd Spq* = *B'*

by (*force elim*: *rel-pmf.cases*)

let *?f* = (λ((*a*, *c*), (*b*, *d*)). ((*a*, *b*), (*c*, *d*)))
let *?pq* = *map-pmf ?f* (*pair-pmf Rpq Spq*)
have [*simp*]: (λ*x*. *fst* (*?f x*)) = (λ(*a*, *b*). (*fst a*, *fst b*)) (λ*x*. *snd* (*?f x*)) = (λ(*a*, *b*). (*snd a*, *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-reflI*:
assumes ⋀*x*. *x* ∈ *set-pmf p* ⟹ *P x x*
shows *rel-pmf P p p*
by (*rule rel-pmf.intros*[**where** *pq*=*map-pmf* (λ*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*: ⋀*x*. *x* ∈ *set-pmf p* ⟹ *pmf p x* = *pmf q* (*f x*)
shows *rel-pmf* (λ*x y*. *f x* = *y*) *p q*
proof(*rule rel-pmf.intros*)
let *?pq* = *map-pmf* (λ*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* ∈ *set-pmf q*)
case *True*
with *f* **obtain** *j* **where** *i* = *f j j* ∈ *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
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* = *bind-pmf M* (λ*x*. *x*)

lemma *bind-eq-join-pmf*: *bind-pmf M f* = *join-pmf* (*map-pmf f M*)
unfolding *join-pmf-def bind-map-pmf* **..**

lemma *join-eq-bind-pmf*: *join-pmf M* = *bind-pmf M id*
by (*simp add*: *join-pmf-def id-def*)

lemma *pmf-join*: *pmf* (*join-pmf N*) *i* = (∫ *M*. *pmf M i* ∂*measure-pmf N*)
unfolding *join-pmf-def pmf-bind* **..**

lemma *ennreal-pmf-join*: *ennreal* (*pmf* (*join-pmf N*) *i*) = (∫⁺ *M*. *pmf M i* ∂*measure-pmf N*)
unfolding *join-pmf-def ennreal-pmf-bind* **..**

lemma *set-pmf-join-pmf*[*simp*]: *set-pmf* (*join-pmf f*) = (⋃ *p*∈*set-pmf f*. *set-pmf p*)
by (*simp add*: *join-pmf-def*)

lemma *join-return-pmf*: *join-pmf* (*return-pmf M*) = *M*
by (*simp add*: *integral-return pmf-eq-iff pmf-join return-pmf*.*rep-eq*)

lemma *map-join-pmf*: *map-pmf f* (*join-pmf AA*) = *join-pmf* (*map-pmf* (*map-pmf f*) *AA*)
by (*simp add*: *join-pmf-def map-pmf-def bind-assoc-pmf bind-return-pmf*)

lemma *join-map-return-pmf*: *join-pmf* (*map-pmf 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* = *map-pmf fst pq*
and *q*: *q* = *map-pmf snd pq*
and *P*: ⋀*x y*. (*x*, *y*) ∈ *set-pmf pq* ⟹ *rel-pmf P x y*
by *cases auto*
from *P* **obtain** *PQ*
where *PQ*: ⋀*x y a b*. ⟦ (*x*, *y*) ∈ *set-pmf pq*; (*a*, *b*) ∈ *set-pmf* (*PQ x y*) ⟧ ⟹ *P a b*
and *x*: ⋀*x y*. (*x*, *y*) ∈ *set-pmf pq* ⟹ *map-pmf fst* (*PQ x y*) = *x*
and *y*: ⋀*x y*. (*x*, *y*) ∈ *set-pmf pq* ⟹ *map-pmf snd* (*PQ x y*) = *y*
by(*metis rel-pmf*.*simps*)

let *?r* = *bind-pmf pq* (λ(*x*, *y*). *PQ x y*)
have ⋀*a b*. (*a*, *b*) ∈ *set-pmf ?r* ⟹ *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*: $\bigwedge x\ y.\ R\ x\ y \Longrightarrow$ *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, http://dx.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*: $\bigwedge x\ y.\ (x, y) \in$ *set-pmf pq* $\Longrightarrow$ *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*!: *measure-pmf.finite-measure-mono-A1 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* $\in$ *UNIV* // {(*x*, *y*). *inf R* R^{-1-1} *x y*}
then obtain *x* **where** *C*: *C* = {*y*. *R x y* $\wedge$ *R y x*}

by (*auto elim*: *quotientE*)

let *?R* = λ*x y*. *R x y* ∧ *R y x*
let *?μR* = λ*y*. *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} = *op* =
using *refl antisym* **by** (*auto intro*!: *ext simp add*: *reflpD dest*: *antisymD*)
finally show *?thesis* **unfolding** *pmf.rel-eq* **.**
qed

lemma *reflp-rel-pmf*: *reflp R* ⟹ *reflp* (*rel-pmf R*)
by(*blast intro*: *reflpI rel-pmf-reflI reflpD*)

lemma *antisymP-rel-pmf*:
⟦ *reflp R*; *transp R*; *antisymP R* ⟧
⟹ *antisymP* (*rel-pmf R*)
by(*rule antisymI*)(*blast intro*: *rel-pmf-antisym*)

lemma *transp-rel-pmf*:
assumes *transp R*
shows *transp* (*rel-pmf R*)
proof (*rule transpI*)
fix *x y z*
assume *rel-pmf R x y* **and** *rel-pmf R y z*
hence *rel-pmf* (*R OO R*) *x z* **by** (*simp add*: *pmf.rel-compp relcompp.relcompI*)

thus *rel-pmf R x z*
using *assms* **by** (*metis* (*no-types*) *pmf.rel-mono rev-predicate2D transp-relcompp-less-eq*)
qed

25.6 Distributions

context
begin

interpretation *pmf-as-function* **.**

25.6.1 Bernoulli Distribution

lift-definition *bernoulli-pmf* :: *real* ⇒ *bool pmf* **is**
$\lambda p\ b.\ ((\lambda p.\ \textit{if}\ b\ \textit{then}\ p\ \textit{else}\ 1 - p) \circ \textit{min}\ 1 \circ \textit{max}\ 0)\ p$
by (*auto simp*: *nn-integral-count-space-finite*[**where** *A*={*False*, *True*}] *UNIV-bool*
split: *split-max split-min*)

lemma *pmf-bernoulli-True*[*simp*]: $0 \leq p \Longrightarrow p \leq 1 \Longrightarrow \textit{pmf}\ (\textit{bernoulli-pmf}\ p)\ \textit{True} = p$
by *transfer simp*

lemma *pmf-bernoulli-False*[*simp*]: $0 \leq p \Longrightarrow p \leq 1 \Longrightarrow \textit{pmf}\ (\textit{bernoulli-pmf}\ p)\ \textit{False} = 1 - p$
by *transfer simp*

lemma *set-pmf-bernoulli*[*simp*]: $0 < p \Longrightarrow p < 1 \Longrightarrow \textit{set-pmf}\ (\textit{bernoulli-pmf}\ p) = \textit{UNIV}$
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 $(\int^+ x.\ f\ x\ \partial\textit{bernoulli-pmf}\ p) = f\ \textit{True} * p + f\ \textit{False} * (1 - p)$
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 $(\int x.\ f\ x\ \partial\textit{bernoulli-pmf}\ p) = f\ \textit{True} * p + f\ \textit{False} * (1 - p)$
by (*subst integral-measure-pmf*[*of UNIV*]) (*auto simp*: *UNIV-bool*)

lemma *pmf-bernoulli-half* [*simp*]: $\textit{pmf}\ (\textit{bernoulli-pmf}\ (1\ /\ 2))\ x = 1\ /\ 2$
by(*cases x*) *simp-all*

lemma *measure-pmf-bernoulli-half*: $\textit{measure-pmf}\ (\textit{bernoulli-pmf}\ (1\ /\ 2)) = \textit{uniform-count-measure}\ \textit{UNIV}$
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)

25.6.2 Geometric Distribution

context
fixes p :: *real* **assumes** p[*arith*]: $0 < p\ p \leq 1$
begin

lift-definition *geometric-pmf* :: *nat pmf* **is** $\lambda n.\ (1 - p)\hat{}n * p$
proof
have $(\sum i.\ ennreal\ (p * (1 - p)\ \hat{}\ i)) = ennreal\ (p * (1 / (1 - (1 - p))))$
by (*intro suminf-ennreal-eq sums-mult geometric-sums*) *auto*
then show $(\int^+ x.\ ennreal\ ((1 - p)\hat{}x * p)\ \partial count\text{-}space\ UNIV) = 1$
by (*simp add*: *nn-integral-count-space-nat field-simps*)
qed *simp*

lemma *pmf-geometric*[*simp*]: *pmf geometric-pmf n* $= (1 - p)\hat{}n * p$
by *transfer rule*

end

lemma *set-pmf-geometric*: $0 < p \Longrightarrow p < 1 \Longrightarrow$ *set-pmf* (*geometric-pmf p*) = *UNIV*
by (*auto simp*: *set-pmf-iff*)

25.6.3 Uniform Multiset Distribution

context
fixes M :: *'a multiset* **assumes** *M-not-empty*: $M \neq \{\#\}$
begin

lift-definition *pmf-of-multiset* :: *'a pmf* **is** $\lambda x.$ *count M x / size M*
proof
show $(\int^+ x.\ ennreal\ (real\ (count\ M\ x)\ /\ real\ (size\ M))\ \partial count\text{-}space\ UNIV) = 1$
using *M-not-empty*
by (*simp add*: *zero-less-divide-iff nn-integral-count-space nonempty-has-size setsum-divide-distrib*[*symmetric*])
(*auto simp*: *size-multiset-overloaded-eq intro*!: *setsum.cong*)
qed *simp*

lemma *pmf-of-multiset*[*simp*]: *pmf pmf-of-multiset x = count M x / size M*
by *transfer rule*

lemma *set-pmf-of-multiset*[*simp*]: *set-pmf pmf-of-multiset = set-mset M*
by (*auto simp*: *set-pmf-iff*)

end

25.6.4 Uniform Distribution

context
fixes $S :: 'a\ set$ **assumes** *S-not-empty*: $S \neq \{\}$ **and** *S-finite*: *finite S*
begin

lift-definition *pmf-of-set* :: $'a\ pmf$ **is** $\lambda x.\ indicator\ S\ x\ /\ card\ S$
proof
show $(\int^+ x.\ ennreal\ (indicator\ S\ x\ /\ real\ (card\ S))\ \partial count\text{-}space\ UNIV) = 1$
using *S-not-empty S-finite*
by (*subst nn-integral-count-space'*[*of S*])
(*auto simp*: *ennreal-of-nat-eq-real-of-nat ennreal-mult*[*symmetric*])
qed *simp*

lemma *pmf-of-set*[*simp*]: *pmf pmf-of-set x* = *indicator S x / card S*
by *transfer rule*

lemma *set-pmf-of-set*[*simp*]: *set-pmf pmf-of-set* = *S*
using *S-finite S-not-empty* **by** (*auto simp*: *set-pmf-iff*)

lemma *emeasure-pmf-of-set-space*[*simp*]: *emeasure pmf-of-set S* = *1*
by (*rule measure-pmf.emeasure-eq-1-AE*) (*auto simp*: *AE-measure-pmf-iff*)

lemma *nn-integral-pmf-of-set*: *nn-integral* (*measure-pmf pmf-of-set*) *f* = *setsum f S / card S*
by (*subst nn-integral-measure-pmf-finite*)
(*simp-all add*: *setsum-left-distrib*[*symmetric*] *card-gt-0-iff S-not-empty S-finite divide-ennreal-def*
divide-ennreal[*symmetric*] *ennreal-of-nat-eq-real-of-nat*[*symmetric*] *ennreal-times-divide*)

lemma *integral-pmf-of-set*: $integral^L$ (*measure-pmf pmf-of-set*) *f* = *setsum f S / card S*
by (*subst integral-measure-pmf*[*of S*]) (*auto simp*: *S-finite setsum-divide-distrib*)

lemma *emeasure-pmf-of-set*: *emeasure* (*measure-pmf pmf-of-set*) *A* = *card* (*S* ∩ *A*) / *card S*
by (*subst nn-integral-indicator*[*symmetric*], *simp*)
(*simp add*: *S-finite S-not-empty card-gt-0-iff indicator-def setsum.If-cases divide-ennreal*
ennreal-of-nat-eq-real-of-nat nn-integral-pmf-of-set)

lemma *measure-pmf-of-set*: *measure* (*measure-pmf pmf-of-set*) *A* = *card* (*S* ∩ *A*) / *card S*
using *emeasure-pmf-of-set*[*OF assms*, *of A*]
by (*simp add*: *measure-nonneg measure-pmf.emeasure-eq-measure*)

end

lemma *pmf-of-set-singleton*: *pmf-of-set* {*x*} = *return-pmf x*

by(*rule pmf-eqI*)(*simp add*: *indicator-def*)

lemma *map-pmf-of-set-inj*:
assumes *f*: *inj-on f A*
and [*simp*]: $A \neq \{\}$ *finite A*
shows *map-pmf f* (*pmf-of-set A*) = *pmf-of-set* (*f* ' *A*) (**is** *?lhs* = *?rhs*)
proof(*rule pmf-eqI*)
fix *i*
show *pmf ?lhs i* = *pmf ?rhs i*
proof(*cases i* ∈ *f* ' *A*)
case *True*
then obtain *i'* **where** *i* = *f i' i'* ∈ *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 *bernoulli-pmf-half-conv-pmf-of-set*: *bernoulli-pmf* (*1* / *2*) = *pmf-of-set UNIV*
by (*rule pmf-eqI*) *simp-all*

25.6.5 Poisson Distribution

context
fixes *rate* :: *real* **assumes** *rate-pos*: *0* < *rate*
begin

lift-definition *poisson-pmf* :: *nat pmf* **is** λ*k*. *rate* ^ *k* / *fact k* ∗ *exp* (−*rate*)
proof
have *summable*: *summable* (λ*x*::*nat*. *rate* ^ *x* / *fact x*) **using** *summable-exp*
by (*simp add*: *field-simps divide-inverse* [*symmetric*])
have (∫⁺(*x*::*nat*). *rate* ^ *x* / *fact x* ∗ *exp* (−*rate*) ∂*count-space UNIV*) =
exp (−*rate*) ∗ (∫⁺(*x*::*nat*). *rate* ^ *x* / *fact x* ∂*count-space UNIV*)
by (*simp add*: *field-simps nn-integral-cmult*[*symmetric*] *ennreal-mult'*[*symmetric*])
also from *rate-pos* **have** (∫⁺(*x*::*nat*). *rate* ^ *x* / *fact x* ∂*count-space UNIV*) =
(∑ *x*. *rate* ^ *x* / *fact x*)
by (*simp-all add*: *nn-integral-count-space-nat suminf-ennreal summable ennreal-suminf-neq-top*)
also have ... = *exp rate* **unfolding** *exp-def*
by (*simp add*: *field-simps divide-inverse* [*symmetric*])
also have *ennreal* (*exp* (−*rate*)) ∗ *ennreal* (*exp rate*) = *1*
by (*simp add*: *mult-exp-exp ennreal-mult*[*symmetric*])
finally show (∫⁺ *x*. *ennreal* (*rate* ^ *x* / (*fact x*) ∗ *exp* (− *rate*)) ∂*count-space UNIV*) = *1* .
qed (*simp add*: *rate-pos*[*THEN less-imp-le*])

lemma *pmf-poisson*[*simp*]: *pmf poisson-pmf k* = *rate* ^ *k* / *fact k* ∗ *exp* (−*rate*)

by *transfer rule*

lemma *set-pmf-poisson*[*simp*]: *set-pmf poisson-pmf* = *UNIV*
using *rate-pos* **by** (*auto simp*: *set-pmf-iff*)

end

25.6.6 Binomial Distribution

context
fixes *n* :: *nat* **and** *p* :: *real* **assumes** *p-nonneg*: 0 ≤ *p* **and** *p-le-1*: *p* ≤ 1
begin

lift-definition *binomial-pmf* :: *nat pmf* **is** λ*k*. (*n choose k*) ∗ *p*^*k* ∗ (1 − *p*)^(*n* − *k*)
proof
have (∫⁺*k*. *ennreal* (*real* (*n choose k*) ∗ *p* ^ *k* ∗ (1 − *p*) ^ (*n* − *k*)) ∂*count-space UNIV*) =
ennreal (∑ *k*≤*n*. *real* (*n choose k*) ∗ *p* ^ *k* ∗ (1 − *p*) ^ (*n* − *k*))
using *p-le-1 p-nonneg* **by** (*subst nn-integral-count-space'*) *auto*
also have (∑ *k*≤*n*. *real* (*n choose k*) ∗ *p* ^ *k* ∗ (1 − *p*) ^ (*n* − *k*)) = (*p* + (1 − *p*)) ^ *n*
by (*subst binomial-ring*) (*simp add*: *atLeast0AtMost*)
finally show (∫⁺ *x*. *ennreal* (*real* (*n choose x*) ∗ *p* ^ *x* ∗ (1 − *p*) ^ (*n* − *x*)) ∂*count-space UNIV*) = 1
by *simp*
qed (*insert p-nonneg p-le-1*, *simp*)

lemma *pmf-binomial*[*simp*]: *pmf binomial-pmf k* = (*n choose k*) ∗ *p*^*k* ∗ (1 − *p*)^(*n* − *k*)
by *transfer rule*

lemma *set-pmf-binomial-eq*: *set-pmf binomial-pmf* = (*if p* = 0 *then* {0} *else if p* = 1 *then* {*n*} *else* {.. *n*})
using *p-nonneg p-le-1* **unfolding** *set-eq-iff set-pmf-iff pmf-binomial* **by** (*auto simp*: *set-pmf-iff*)

end

end

lemma *set-pmf-binomial-0*[*simp*]: *set-pmf* (*binomial-pmf n* 0) = {0}
by (*simp add*: *set-pmf-binomial-eq*)

lemma *set-pmf-binomial-1*[*simp*]: *set-pmf* (*binomial-pmf n* 1) = {*n*}
by (*simp add*: *set-pmf-binomial-eq*)

lemma *set-pmf-binomial*[*simp*]: 0 < *p* ⟹ *p* < 1 ⟹ *set-pmf* (*binomial-pmf n p*) = {..*n*}

by (*simp add*: *set-pmf-binomial-eq*)

context begin interpretation *lifting-syntax* **.**

lemma *bind-pmf-parametric* [*transfer-rule*]:
(*rel-pmf A* ===> (*A* ===> *rel-pmf B*) ===> *rel-pmf B*) *bind-pmf bind-pmf*
by(*blast intro*: *rel-pmf-bindI dest*: *rel-funD*)

lemma *return-pmf-parametric* [*transfer-rule*]: (*A* ===> *rel-pmf A*) *return-pmf return-pmf*
by(*rule rel-funI*) *simp*

end

end

26 Infinite Streams

theory *Stream*
imports ~~/*src*/*HOL*/*Library*/*Nat-Bijection*
begin

codatatype (*sset*: 'a) *stream* =
SCons (*shd*: 'a) (*stl*: 'a *stream*) (**infixr** ## 65)
for
map: *smap*
rel: *stream-all2*

context
begin

qualified definition *smember* :: 'a ⇒ 'a *stream* ⇒ *bool* **where**
[*code-abbrev*]: *smember x s* ⟷ *x* ∈ *sset s*

lemma *smember-code*[*code*, *simp*]: *smember x* (*y* ## *s*) = (*if x* = *y then True else smember x s*)
unfolding *smember-def* **by** *auto*

end

lemmas *smap-simps*[*simp*] = *stream.map-sel*
lemmas *shd-sset* = *stream.set-sel*(*1*)
lemmas *stl-sset* = *stream.set-sel*(*2*)

theorem *sset-induct*[*consumes 1*, *case-names shd stl*, *induct set*: *sset*]:
assumes *y* ∈ *sset s* **and** ⋀*s*. *P* (*shd s*) *s* **and** ⋀*s y*. ⟦*y* ∈ *sset* (*stl s*); *P y* (*stl s*)⟧ ⟹ *P y s*
shows *P y s*

using *assms* **by** *induct* (*metis stream.sel(1)*, *auto*)

lemma *smap-ctr*: *smap f s = x ## s' ⟷ f (shd s) = x ∧ smap f (stl s) = s'*
by (*cases s*) *simp*

26.1 prepend list to stream

primrec *shift* :: *'a list ⇒ 'a stream ⇒ 'a stream* (**infixr** @− *65*) **where**
shift [] s = s
| *shift (x # xs) s = x ## shift xs s*

lemma *smap-shift*[*simp*]: *smap f (xs @− s) = map f xs @− smap f s*
by (*induct xs*) *auto*

lemma *shift-append*[*simp*]: *(xs @ ys) @− s = xs @− ys @− s*
by (*induct xs*) *auto*

lemma *shift-simps*[*simp*]:
shd (xs @− s) = (if xs = [] then shd s else hd xs)
stl (xs @− s) = (if xs = [] then stl s else tl xs @− s)
by (*induct xs*) *auto*

lemma *sset-shift*[*simp*]: *sset (xs @− s) = set xs ∪ sset s*
by (*induct xs*) *auto*

lemma *shift-left-inj*[*simp*]: *xs @− s1 = xs @− s2 ⟷ s1 = s2*
by (*induct xs*) *auto*

26.2 set of streams with elements in some fixed set

context
notes [[*inductive-internals*]]
begin

coinductive-set
streams :: *'a set ⇒ 'a stream set*
for *A* :: *'a set*
where
Stream[*intro*!, *simp*, *no-atp*]: *⟦a ∈ A; s ∈ streams A⟧ ⟹ a ## s ∈ streams A*

end

lemma *in-streams*: *stl s ∈ streams S ⟹ shd s ∈ S ⟹ s ∈ streams S*
by (*cases s*) *auto*

lemma *streamsE*: *s ∈ streams A ⟹ (shd s ∈ A ⟹ stl s ∈ streams A ⟹ P)*
⟹ P
by (*erule streams.cases*) *simp-all*

lemma *Stream-image*: *x ## y ∈ (op ## x') ‘ Y ⟷ x = x' ∧ y ∈ Y*

by *auto*

lemma *shift-streams*: $[\![w \in lists\ A;\ s \in streams\ A]\!] \Longrightarrow w\ @{-}\ s \in streams\ A$
by (*induct w*) *auto*

lemma *streams-Stream*: $x\ \#\#\ s \in streams\ A \longleftrightarrow x \in A \wedge s \in streams\ A$
by (*auto elim*: *streams.cases*)

lemma *streams-stl*: $s \in streams\ A \Longrightarrow stl\ s \in streams\ A$
by (*cases s*) (*auto simp*: *streams-Stream*)

lemma *streams-shd*: $s \in streams\ A \Longrightarrow shd\ s \in A$
by (*cases s*) (*auto simp*: *streams-Stream*)

lemma *sset-streams*:
assumes $sset\ s \subseteq A$
shows $s \in streams\ A$
using *assms* **proof** (*coinduction arbitrary*: *s*)
case *streams* **then show** *?case* **by** (*cases s*) *simp*
qed

lemma *streams-sset*:
assumes $s \in streams\ A$
shows $sset\ s \subseteq A$
proof
fix *x* **assume** $x \in sset\ s$ **from** *this* ‹$s \in streams\ A$› **show** $x \in A$
by (*induct s*) (*auto intro*: *streams-shd streams-stl*)
qed

lemma *streams-iff-sset*: $s \in streams\ A \longleftrightarrow sset\ s \subseteq A$
by (*metis sset-streams streams-sset*)

lemma *streams-mono*: $s \in streams\ A \Longrightarrow A \subseteq B \Longrightarrow s \in streams\ B$
unfolding *streams-iff-sset* **by** *auto*

lemma *streams-mono2*: $S \subseteq T \Longrightarrow streams\ S \subseteq streams\ T$
by (*auto intro*: *streams-mono*)

lemma *smap-streams*: $s \in streams\ A \Longrightarrow (\bigwedge x.\ x \in A \Longrightarrow f\ x \in B) \Longrightarrow smap\ f\ s \in streams\ B$
unfolding *streams-iff-sset stream.set-map* **by** *auto*

lemma *streams-empty*: $streams\ \{\} = \{\}$
by (*auto elim*: *streams.cases*)

lemma *streams-UNIV*[*simp*]: $streams\ UNIV = UNIV$
by (*auto simp*: *streams-iff-sset*)

26.3 nth, take, drop for streams

primrec *snth* :: *'a stream* ⇒ *nat* ⇒ *'a* (**infixl** !! *100*) **where**
  s !! *0* = *shd s*
| *s* !! *Suc n* = *stl s* !! *n*

lemma *snth-Stream*: (*x* ## *s*) !! *Suc i* = *s* !! *i*
  by *simp*

lemma *snth-smap*[*simp*]: *smap f s* !! *n* = *f* (*s* !! *n*)
  by (*induct n arbitrary*: *s*) *auto*

lemma *shift-snth-less*[*simp*]: *p* < *length xs* ⟹ (*xs* @− *s*) !! *p* = *xs* ! *p*
  by (*induct p arbitrary*: *xs*) (*auto simp*: *hd-conv-nth nth-tl*)

lemma *shift-snth-ge*[*simp*]: *p* ≥ *length xs* ⟹ (*xs* @− *s*) !! *p* = *s* !! (*p* − *length xs*)
  by (*induct p arbitrary*: *xs*) (*auto simp*: *Suc-diff-eq-diff-pred*)

lemma *shift-snth*: (*xs* @− *s*) !! *n* = (*if n* < *length xs then xs* ! *n else s* !! (*n* − *length xs*))
  by *auto*

lemma *snth-sset*[*simp*]: *s* !! *n* ∈ *sset s*
  by (*induct n arbitrary*: *s*) (*auto intro*: *shd-sset stl-sset*)

lemma *sset-range*: *sset s* = *range* (*snth s*)
proof (*intro equalityI subsetI*)
  fix *x* **assume** *x* ∈ *sset s*
  thus *x* ∈ *range* (*snth s*)
  proof (*induct s*)
    case (*stl s x*)
    then obtain *n* **where** *x* = *stl s* !! *n* **by** *auto*
    thus *?case* **by** (*auto intro*: *range-eqI*[*of - - Suc n*])
  qed (*auto intro*: *range-eqI*[*of - - 0*])
qed *auto*

lemma *streams-iff-snth*: *s* ∈ *streams X* ⟷ (∀ *n*. *s* !! *n* ∈ *X*)
  by (*force simp*: *streams-iff-sset sset-range*)

lemma *snth-in*: *s* ∈ *streams X* ⟹ *s* !! *n* ∈ *X*
  by (*simp add*: *streams-iff-snth*)

primrec *stake* :: *nat* ⇒ *'a stream* ⇒ *'a list* **where**
  stake 0 s = []
| *stake* (*Suc n*) *s* = *shd s* # *stake n* (*stl s*)

lemma *length-stake*[*simp*]: *length* (*stake n s*) = *n*
  by (*induct n arbitrary*: *s*) *auto*

lemma *stake-smap*[*simp*]: *stake n* (*smap f s*) = *map f* (*stake n s*)
by (*induct n arbitrary*: *s*) *auto*

lemma *take-stake*: *take n* (*stake m s*) = *stake* (*min n m*) *s*
proof (*induct m arbitrary*: *s n*)
case (*Suc m*) **thus** *?case* **by** (*cases n*) *auto*
qed *simp*

primrec *sdrop* :: *nat* ⇒ *'a stream* ⇒ *'a stream* **where**
sdrop 0 s = *s*
| *sdrop* (*Suc n*) *s* = *sdrop n* (*stl s*)

lemma *sdrop-simps*[*simp*]:
shd (*sdrop n s*) = *s* !! *n stl* (*sdrop n s*) = *sdrop* (*Suc n*) *s*
by (*induct n arbitrary*: *s*) *auto*

lemma *sdrop-smap*[*simp*]: *sdrop n* (*smap f s*) = *smap f* (*sdrop n s*)
by (*induct n arbitrary*: *s*) *auto*

lemma *sdrop-stl*: *sdrop n* (*stl s*) = *stl* (*sdrop n s*)
by (*induct n*) *auto*

lemma *drop-stake*: *drop n* (*stake m s*) = *stake* (*m* − *n*) (*sdrop n s*)
proof (*induct m arbitrary*: *s n*)
case (*Suc m*) **thus** *?case* **by** (*cases n*) *auto*
qed *simp*

lemma *stake-sdrop*: *stake n s* @− *sdrop n s* = *s*
by (*induct n arbitrary*: *s*) *auto*

lemma *id-stake-snth-sdrop*:
s = *stake i s* @− *s* !! *i* ## *sdrop* (*Suc i*) *s*
by (*subst stake-sdrop*[*symmetric*, *of - i*]) (*metis sdrop-simps stream.collapse*)

lemma *smap-alt*: *smap f s* = *s'* ⟷ (∀ *n*. *f* (*s* !! *n*) = *s'* !! *n*) (**is** *?L* = *?R*)
proof
assume *?R*
then have ⋀*n*. *smap f* (*sdrop n s*) = *sdrop n s'*
by *coinduction* (*auto intro*: *exI*[*of - 0*] *simp del*: *sdrop.simps*(*2*))
then show *?L* **using** *sdrop.simps*(*1*) **by** *metis*
qed *auto*

lemma *stake-invert-Nil*[*iff*]: *stake n s* = [] ⟷ *n* = *0*
by (*induct n*) *auto*

lemma *sdrop-shift*: *sdrop i* (*w* @− *s*) = *drop i w* @− *sdrop* (*i* − *length w*) *s*
by (*induct i arbitrary*: *w s*) (*auto simp*: *drop-tl drop-Suc neq-Nil-conv*)

lemma *stake-shift*: *stake i* (*w* @− *s*) = *take i w* @ *stake* (*i* − *length w*) *s*

by (*induct i arbitrary*: *w s*) (*auto simp*: *neq-Nil-conv*)

lemma *stake-add*[*simp*]: *stake m s @ stake n* (*sdrop m s*) = *stake* (*m* + *n*) *s*
by (*induct m arbitrary*: *s*) *auto*

lemma *sdrop-add*[*simp*]: *sdrop n* (*sdrop m s*) = *sdrop* (*m* + *n*) *s*
by (*induct m arbitrary*: *s*) *auto*

lemma *sdrop-snth*: *sdrop n s* !! *m* = *s* !! (*n* + *m*)
by (*induct n arbitrary*: *m s*) *auto*

partial-function (*tailrec*) *sdrop-while* :: ('*a* ⇒ *bool*) ⇒ '*a stream* ⇒ '*a stream*
where
sdrop-while P s = (*if P* (*shd s*) *then sdrop-while P* (*stl s*) *else s*)

lemma *sdrop-while-SCons*[*code*]:
sdrop-while P (*a ## s*) = (*if P a then sdrop-while P s else a ## s*)
by (*subst sdrop-while.simps*) *simp*

lemma *sdrop-while-sdrop-LEAST*:
assumes ∃ *n*. *P* (*s* !! *n*)
shows *sdrop-while* (*Not o P*) *s* = *sdrop* (*LEAST n*. *P* (*s* !! *n*)) *s*
proof −
from *assms* **obtain** *m* **where** *P* (*s* !! *m*) ⋀*n*. *P* (*s* !! *n*) ⟹ *m* ≤ *n*
and ∗: (*LEAST n*. *P* (*s* !! *n*)) = *m* **by** *atomize-elim* (*auto intro*: *LeastI Least-le*)
thus *?thesis* **unfolding** ∗
proof (*induct m arbitrary*: *s*)
case (*Suc m*)
hence *sdrop-while* (*Not* ∘ *P*) (*stl s*) = *sdrop m* (*stl s*)
by (*metis* (*full-types*) *not-less-eq-eq snth.simps*(*2*))
moreover from *Suc*(*3*) **have** ¬ (*P* (*s* !! *0*)) **by** *blast*
ultimately show *?case* **by** (*subst sdrop-while.simps*) *simp*
qed (*metis comp-apply sdrop.simps*(*1*) *sdrop-while.simps snth.simps*(*1*))
qed

primcorec *sfilter* **where**
shd (*sfilter P s*) = *shd* (*sdrop-while* (*Not o P*) *s*)
| *stl* (*sfilter P s*) = *sfilter P* (*stl* (*sdrop-while* (*Not o P*) *s*))

lemma *sfilter-Stream*: *sfilter P* (*x ## s*) = (*if P x then x ## sfilter P s else sfilter P s*)
proof (*cases P x*)
case *True* **thus** *?thesis* **by** (*subst sfilter.ctr*) (*simp add*: *sdrop-while-SCons*)
next
case *False* **thus** *?thesis* **by** (*subst* (*1 2*) *sfilter.ctr*) (*simp add*: *sdrop-while-SCons*)
qed

26.4 unary predicates lifted to streams

definition *stream-all P s* = (∀ *p*. *P* (*s* !! *p*))

lemma *stream-all-iff*[*iff*]: *stream-all P s* ⟷ *Ball* (*sset s*) *P*
unfolding *stream-all-def sset-range* **by** *auto*

lemma *stream-all-shift*[*simp*]: *stream-all P* (*xs* @− *s*) = (*list-all P xs* ∧ *stream-all P s*)
unfolding *stream-all-iff list-all-iff* **by** *auto*

lemma *stream-all-Stream*: *stream-all P* (*x* ## *X*) ⟷ *P x* ∧ *stream-all P X*
by *simp*

26.5 recurring stream out of a list

primcorec *cycle* :: *'a list* ⇒ *'a stream* **where**
shd (*cycle xs*) = *hd xs*
| *stl* (*cycle xs*) = *cycle* (*tl xs* @ [*hd xs*])

lemma *cycle-decomp*: *u* ≠ [] ⟹ *cycle u* = *u* @− *cycle u*
proof (*coinduction arbitrary*: *u*)
case *Eq-stream* **then show** *?case* **using** *stream.collapse*[*of cycle u*]
by (*auto intro*!: *exI*[*of* - *tl u* @ [*hd u*]])
qed

lemma *cycle-Cons*[*code*]: *cycle* (*x* # *xs*) = *x* ## *cycle* (*xs* @ [*x*])
by (*subst cycle.ctr*) *simp*

lemma *cycle-rotated*: ⟦*v* ≠ []; *cycle u* = *v* @− *s*⟧ ⟹ *cycle* (*tl u* @ [*hd u*]) = *tl v* @− *s*
by (*auto dest*: *arg-cong*[*of* - - *stl*])

lemma *stake-append*: *stake n* (*u* @− *s*) = *take* (*min* (*length u*) *n*) *u* @ *stake* (*n* − *length u*) *s*
proof (*induct n arbitrary*: *u*)
case (*Suc n*) **thus** *?case* **by** (*cases u*) *auto*
qed *auto*

lemma *stake-cycle-le*[*simp*]:
assumes *u* ≠ [] *n* < *length u*
shows *stake n* (*cycle u*) = *take n u*
using *min-absorb2*[*OF less-imp-le-nat*[*OF assms*(*2*)]]
by (*subst cycle-decomp*[*OF assms*(*1*)], *subst stake-append*) *auto*

lemma *stake-cycle-eq*[*simp*]: *u* ≠ [] ⟹ *stake* (*length u*) (*cycle u*) = *u*
by (*subst cycle-decomp*) (*auto simp*: *stake-shift*)

lemma *sdrop-cycle-eq*[*simp*]: *u* ≠ [] ⟹ *sdrop* (*length u*) (*cycle u*) = *cycle u*
by (*subst cycle-decomp*) (*auto simp*: *sdrop-shift*)

lemma *stake-cycle-eq-mod-0*[*simp*]: ⟦*u* ≠ []; *n mod length u* = *0*⟧ ⟹
stake n (*cycle u*) = *concat* (*replicate* (*n div length u*) *u*)
by (*induct n div length u arbitrary*: *n u*) (*auto simp*: *stake-add*[*symmetric*])

lemma *sdrop-cycle-eq-mod-0*[*simp*]: ⟦*u* ≠ []; *n mod length u* = *0*⟧ ⟹
sdrop n (*cycle u*) = *cycle u*
by (*induct n div length u arbitrary*: *n u*) (*auto simp*: *sdrop-add*[*symmetric*])

lemma *stake-cycle*: *u* ≠ [] ⟹
stake n (*cycle u*) = *concat* (*replicate* (*n div length u*) *u*) @ *take* (*n mod length u*) *u*
by (*subst mod-div-equality*[*of n length u*, *symmetric*], *unfold stake-add*[*symmetric*]) *auto*

lemma *sdrop-cycle*: *u* ≠ [] ⟹ *sdrop n* (*cycle u*) = *cycle* (*rotate* (*n mod length u*) *u*)
by (*induct n arbitrary*: *u*) (*auto simp*: *rotate1-rotate-swap rotate1-hd-tl rotate-conv-mod*[*symmetric*])

26.6 iterated application of a function

primcorec *siterate* **where**
shd (*siterate f x*) = *x*
| *stl* (*siterate f x*) = *siterate f* (*f x*)

lemma *stake-Suc*: *stake* (*Suc n*) *s* = *stake n s* @ [*s* !! *n*]
by (*induct n arbitrary*: *s*) *auto*

lemma *snth-siterate*[*simp*]: *siterate f x* !! *n* = (*f* ^ *n*) *x*
by (*induct n arbitrary*: *x*) (*auto simp*: *funpow-swap1*)

lemma *sdrop-siterate*[*simp*]: *sdrop n* (*siterate f x*) = *siterate f* ((*f* ^ *n*) *x*)
by (*induct n arbitrary*: *x*) (*auto simp*: *funpow-swap1*)

lemma *stake-siterate*[*simp*]: *stake n* (*siterate f x*) = *map* (λ*n*. (*f* ^ *n*) *x*) [*0* ..< *n*]
by (*induct n arbitrary*: *x*) (*auto simp del*: *stake.simps*(*2*) *simp*: *stake-Suc*)

lemma *sset-siterate*: *sset* (*siterate f x*) = {(*f* ^ *n*) *x* | *n*. *True*}
by (*auto simp*: *sset-range*)

lemma *smap-siterate*: *smap f* (*siterate f x*) = *siterate f* (*f x*)
by (*coinduction arbitrary*: *x*) *auto*

26.7 stream repeating a single element

abbreviation *sconst* ≡ *siterate id*

lemma *shift-replicate-sconst*[*simp*]: *replicate n x* @− *sconst x* = *sconst x*
by (*subst* (*3*) *stake-sdrop*[*symmetric*]) (*simp add*: *map-replicate-trivial*)

lemma *sset-sconst*[*simp*]: *sset* (*sconst x*) = {*x*}
by (*simp add*: *sset-siterate*)

lemma *sconst-alt*: *s* = *sconst x* ⟷ *sset s* = {*x*}
proof
assume *sset s* = {*x*}
then show *s* = *sconst x*
proof (*coinduction arbitrary*: *s*)
case *Eq-stream*
then have *shd s* = *x sset* (*stl s*) ⊆ {*x*} **by** (*case-tac* [!] *s*) *auto*
then have *sset* (*stl s*) = {*x*} **by** (*cases stl s*) *auto*
with ‹*shd s* = *x*› **show** *?case* **by** *auto*
qed
qed *simp*

lemma *sconst-cycle*: *sconst x* = *cycle* [*x*]
by *coinduction auto*

lemma *smap-sconst*: *smap f* (*sconst x*) = *sconst* (*f x*)
by *coinduction auto*

lemma *sconst-streams*: *x* ∈ *A* ⟹ *sconst x* ∈ *streams A*
by (*simp add*: *streams-iff-sset*)

26.8 stream of natural numbers

abbreviation *fromN* ≡ *siterate Suc*

abbreviation *nats* ≡ *fromN 0*

lemma *sset-fromN*[*simp*]: *sset* (*fromN n*) = {*n* ..}
by (*auto simp add*: *sset-siterate le-iff-add*)

lemma *stream-smap-fromN*: *s* = *smap* (λ*j*. *let i* = *j* − *n in s* !! *i*) (*fromN n*)
by (*coinduction arbitrary*: *s n*)
(*force simp*: *neq-Nil-conv Let-def snth.simps*(*2*)[*symmetric*] *Suc-diff-Suc*
intro: *stream.map-cong split*: *if-splits simp del*: *snth.simps*(*2*))

lemma *stream-smap-nats*: *s* = *smap* (*snth s*) *nats*
using *stream-smap-fromN*[**where** *n* = *0*] **by** *simp*

26.9 flatten a stream of lists

primcorec *flat* **where**
shd (*flat ws*) = *hd* (*shd ws*)
| *stl* (*flat ws*) = *flat* (*if tl* (*shd ws*) = [] *then stl ws else tl* (*shd ws*) ## *stl ws*)

lemma *flat-Cons*[*simp*, *code*]: *flat* ((*x* # *xs*) ## *ws*) = *x* ## *flat* (*if xs* = [] *then ws else xs* ## *ws*)
by (*subst flat.ctr*) *simp*

lemma *flat-Stream*[*simp*]: $xs \neq [] \Longrightarrow flat\ (xs\ \#\#\ ws) = xs\ @-\ flat\ ws$
by (*induct xs*) *auto*

lemma *flat-unfold*: $shd\ ws \neq [] \Longrightarrow flat\ ws = shd\ ws\ @-\ flat\ (stl\ ws)$
by (*cases ws*) *auto*

lemma *flat-snth*: $\forall xs \in sset\ s.\ xs \neq [] \Longrightarrow flat\ s\ !!\ n = (if\ n < length\ (shd\ s)$
then
$shd\ s\ !\ n\ else\ flat\ (stl\ s)\ !!\ (n - length\ (shd\ s)))$
by (*metis flat-unfold not-less shd-sset shift-snth-ge shift-snth-less*)

lemma *sset-flat*[*simp*]: $\forall xs \in sset\ s.\ xs \neq [] \Longrightarrow$
$sset\ (flat\ s) = (\bigcup xs \in sset\ s.\ set\ xs)$ (**is** $?P \Longrightarrow ?L = ?R$)
proof *safe*
fix x **assume** $?P\ x : ?L$
then obtain m **where** $x = flat\ s\ !!\ m$ **by** (*metis image-iff sset-range*)
with ‹$?P$› **obtain** $n\ m'$ **where** $x = s\ !!\ n\ !\ m'\ m' < length\ (s\ !!\ n)$
proof (*atomize-elim*, *induct m arbitrary*: *s rule*: *less-induct*)
case (*less y*)
thus *?case*
proof ($cases\ y < length\ (shd\ s)$)
case *True* **thus** *?thesis* **by** (*metis flat-snth less*(*2*,*3*) *snth.simps*(*1*))
next
case *False*
hence $x = flat\ (stl\ s)\ !!\ (y - length\ (shd\ s))$ **by** (*metis less*(*2*,*3*) *flat-snth*)
moreover
{ **from** *less*(*2*) **have** $*$: $length\ (shd\ s) > 0$ **by** (*cases s*) *simp-all*
with *False* **have** $y > 0$ **by** (*cases y*) *simp-all*
with $*$ **have** $y - length\ (shd\ s) < y$ **by** *simp*
}
moreover have $\forall xs \in sset\ (stl\ s).\ xs \neq []$ **using** *less*(*2*) **by** (*cases s*) *auto*
ultimately have $\exists n\ m'.\ x = stl\ s\ !!\ n\ !\ m' \wedge m' < length\ (stl\ s\ !!\ n)$ **by**
(*intro less*(*1*)) *auto*
thus *?thesis* **by** (*metis snth.simps*(*2*))
qed
qed
thus $x \in ?R$ **by** (*auto simp*: *sset-range dest*!: *nth-mem*)
next
fix *x xs* **assume** $xs \in sset\ s\ ?P\ x \in set\ xs$ **thus** $x \in ?L$
by (*induct rule*: *sset-induct*)
(*metis UnI1 flat-unfold shift.simps*(*1*) *sset-shift*,
metis UnI2 flat-unfold shd-sset stl-sset sset-shift)
qed

26.10 merge a stream of streams

definition *smerge* :: *'a stream stream* ⇒ *'a stream* **where**
$smerge\ ss = flat\ (smap\ (\lambda n.\ map\ (\lambda s.\ s\ !!\ n)\ (stake\ (Suc\ n)\ ss)\ @\ stake\ n\ (ss\ !!$

n)) *nats*)

lemma *stake-nth*[*simp*]: $m < n \Longrightarrow$ *stake n s* ! *m* = *s* !! *m*
by (*induct n arbitrary*: *s m*) (*auto simp*: *nth-Cons'*, *metis Suc-pred snth.simps*(*2*))

lemma *snth-sset-smerge*: *ss* !! *n* !! *m* ∈ *sset* (*smerge ss*)
proof (*cases n* ≤ *m*)
case *False* **thus** *?thesis* **unfolding** *smerge-def*
by (*subst sset-flat*)
(*auto simp*: *stream.set-map in-set-conv-nth simp del*: *stake.simps*
intro!: *exI*[*of* - *n*, *OF disjI2*] *exI*[*of* - *m*, *OF mp*])
next
case *True* **thus** *?thesis* **unfolding** *smerge-def*
by (*subst sset-flat*)
(*auto simp*: *stream.set-map in-set-conv-nth image-iff simp del*: *stake.simps*
snth.simps
intro!: *exI*[*of* - *m*, *OF disjI1*] *bexI*[*of* - *ss* !! *n*] *exI*[*of* - *n*, *OF mp*])
qed

lemma *sset-smerge*: *sset* (*smerge ss*) = *UNION* (*sset ss*) *sset*
proof *safe*
fix *x* **assume** *x* ∈ *sset* (*smerge ss*)
thus *x* ∈ *UNION* (*sset ss*) *sset*
unfolding *smerge-def* **by** (*subst* (*asm*) *sset-flat*)
(*auto simp*: *stream.set-map in-set-conv-nth sset-range simp del*: *stake.simps*,
fast+)
next
fix *s x* **assume** *s* ∈ *sset ss x* ∈ *sset s*
thus *x* ∈ *sset* (*smerge ss*) **using** *snth-sset-smerge* **by** (*auto simp*: *sset-range*)
qed

26.11 product of two streams

definition *sproduct* :: *'a stream* ⇒ *'b stream* ⇒ (*'a* × *'b*) *stream* **where**
sproduct s1 s2 = *smerge* (*smap* (λ*x*. *smap* (*Pair x*) *s2*) *s1*)

lemma *sset-sproduct*: *sset* (*sproduct s1 s2*) = *sset s1* × *sset s2*
unfolding *sproduct-def sset-smerge* **by** (*auto simp*: *stream.set-map*)

26.12 interleave two streams

primcorec *sinterleave* **where**
shd (*sinterleave s1 s2*) = *shd s1*
| *stl* (*sinterleave s1 s2*) = *sinterleave s2* (*stl s1*)

lemma *sinterleave-code*[*code*]:
sinterleave (*x* ## *s1*) *s2* = *x* ## *sinterleave s2 s1*
by (*subst sinterleave.ctr*) *simp*

lemma *sinterleave-snth*[*simp*]:

even n ⟹ *sinterleave s1 s2* !! *n* = *s1* !! (*n div 2*)
odd n ⟹ *sinterleave s1 s2* !! *n* = *s2* !! (*n div 2*)
by (*induct n arbitrary*: *s1 s2*) *simp-all*

lemma *sset-sinterleave*: *sset* (*sinterleave s1 s2*) = *sset s1* ∪ *sset s2*
proof (*intro equalityI subsetI*)
fix *x* **assume** *x* ∈ *sset* (*sinterleave s1 s2*)
then obtain *n* **where** *x* = *sinterleave s1 s2* !! *n* **unfolding** *sset-range* **by** *blast*
thus *x* ∈ *sset s1* ∪ *sset s2* **by** (*cases even n*) *auto*
next
fix *x* **assume** *x* ∈ *sset s1* ∪ *sset s2*
thus *x* ∈ *sset* (*sinterleave s1 s2*)
proof
assume *x* ∈ *sset s1*
then obtain *n* **where** *x* = *s1* !! *n* **unfolding** *sset-range* **by** *blast*
hence *sinterleave s1 s2* !! (*2* ∗ *n*) = *x* **by** *simp*
thus *?thesis* **unfolding** *sset-range* **by** *blast*
next
assume *x* ∈ *sset s2*
then obtain *n* **where** *x* = *s2* !! *n* **unfolding** *sset-range* **by** *blast*
hence *sinterleave s1 s2* !! (*2* ∗ *n* + *1*) = *x* **by** *simp*
thus *?thesis* **unfolding** *sset-range* **by** *blast*
qed
qed

26.13 zip

primcorec *szip* **where**
shd (*szip s1 s2*) = (*shd s1*, *shd s2*)
| *stl* (*szip s1 s2*) = *szip* (*stl s1*) (*stl s2*)

lemma *szip-unfold*[*code*]: *szip* (*a* ## *s1*) (*b* ## *s2*) = (*a*, *b*) ## (*szip s1 s2*)
by (*subst szip.ctr*) *simp*

lemma *snth-szip*[*simp*]: *szip s1 s2* !! *n* = (*s1* !! *n*, *s2* !! *n*)
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *stake-szip*[*simp*]:
stake n (*szip s1 s2*) = *zip* (*stake n s1*) (*stake n s2*)
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *sdrop-szip*[*simp*]: *sdrop n* (*szip s1 s2*) = *szip* (*sdrop n s1*) (*sdrop n s2*)
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *smap-szip-fst*:
smap (λ*x*. *f* (*fst x*)) (*szip s1 s2*) = *smap f s1*
by (*coinduction arbitrary*: *s1 s2*) *auto*

lemma *smap-szip-snd*:

smap (λ*x*. *g* (*snd x*)) (*szip s1 s2*) = *smap g s2*
by (*coinduction arbitrary*: *s1 s2*) *auto*

26.14 zip via function

primcorec *smap2* **where**
shd (*smap2 f s1 s2*) = *f* (*shd s1*) (*shd s2*)
| *stl* (*smap2 f s1 s2*) = *smap2 f* (*stl s1*) (*stl s2*)

lemma *smap2-unfold*[*code*]:
smap2 f (*a ## s1*) (*b ## s2*) = *f a b ##* (*smap2 f s1 s2*)
by (*subst smap2.ctr*) *simp*

lemma *smap2-szip*:
smap2 f s1 s2 = *smap* (*case-prod f*) (*szip s1 s2*)
by (*coinduction arbitrary*: *s1 s2*) *auto*

lemma *smap-smap2*[*simp*]:
smap f (*smap2 g s1 s2*) = *smap2* (λ*x y*. *f* (*g x y*)) *s1 s2*
unfolding *smap2-szip stream.map-comp o-def split-def* **..**

lemma *smap2-alt*:
(*smap2 f s1 s2* = *s*) = (∀ *n*. *f* (*s1* !! *n*) (*s2* !! *n*) = *s* !! *n*)
unfolding *smap2-szip smap-alt* **by** *auto*

lemma *snth-smap2*[*simp*]:
smap2 f s1 s2 !! *n* = *f* (*s1* !! *n*) (*s2* !! *n*)
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *stake-smap2*[*simp*]:
stake n (*smap2 f s1 s2*) = *map* (*case-prod f*) (*zip* (*stake n s1*) (*stake n s2*))
by (*induct n arbitrary*: *s1 s2*) *auto*

lemma *sdrop-smap2*[*simp*]:
sdrop n (*smap2 f s1 s2*) = *smap2 f* (*sdrop n s1*) (*sdrop n s2*)
by (*induct n arbitrary*: *s1 s2*) *auto*

end

27 List prefixes, suffixes, and homeomorphic embedding

theory *Sublist*
imports *Main*
begin

27.1 Prefix order on lists

definition *prefixeq* :: *'a list* ⇒ *'a list* ⇒ *bool*

where *prefixeq xs ys* $\longleftrightarrow$ $(\exists zs.\ ys = xs\ @\ zs)$

definition *prefix* :: *'a list* $\Rightarrow$ *'a list* $\Rightarrow$ *bool*
where *prefix xs ys* $\longleftrightarrow$ *prefixeq xs ys* $\wedge$ $xs \neq ys$

interpretation *prefix-order*: *order prefixeq prefix*
by *standard* (*auto simp*: *prefixeq-def prefix-def*)

interpretation *prefix-bot*: *order-bot Nil prefixeq prefix*
by *standard* (*simp add*: *prefixeq-def*)

lemma *prefixeqI* [*intro?*]: $ys = xs\ @\ zs \Longrightarrow$ *prefixeq xs ys*
unfolding *prefixeq-def* **by** *blast*

lemma *prefixeqE* [*elim?*]:
assumes *prefixeq xs ys*
obtains *zs* **where** $ys = xs\ @\ zs$
using *assms* **unfolding** *prefixeq-def* **by** *blast*

lemma *prefixI'* [*intro?*]: $ys = xs\ @\ z\ \#\ zs \Longrightarrow$ *prefix xs ys*
unfolding *prefix-def prefixeq-def* **by** *blast*

lemma *prefixE'* [*elim?*]:
assumes *prefix xs ys*
obtains *z zs* **where** $ys = xs\ @\ z\ \#\ zs$
proof −
from ‹*prefix xs ys*› **obtain** *us* **where** $ys = xs\ @\ us$ **and** $xs \neq ys$
unfolding *prefix-def prefixeq-def* **by** *blast*
with *that* **show** *?thesis* **by** (*auto simp add*: *neq-Nil-conv*)
qed

lemma *prefixI* [*intro?*]: *prefixeq xs ys* $\Longrightarrow$ $xs \neq ys \Longrightarrow$ *prefix xs ys*
unfolding *prefix-def* **by** *blast*

lemma *prefixE* [*elim?*]:
fixes *xs ys* :: *'a list*
assumes *prefix xs ys*
obtains *prefixeq xs ys* **and** $xs \neq ys$
using *assms* **unfolding** *prefix-def* **by** *blast*

27.2 Basic properties of prefixes

theorem *Nil-prefixeq* [*iff*]: *prefixeq* [] *xs*
by (*simp add*: *prefixeq-def*)

theorem *prefixeq-Nil* [*simp*]: (*prefixeq xs* []) = (xs = [])
by (*induct xs*) (*simp-all add*: *prefixeq-def*)

lemma *prefixeq-snoc* [*simp*]: *prefixeq xs* (*ys* @ [*y*]) $\longleftrightarrow$ $xs = ys\ @\ [y] \vee$ *prefixeq*

xs ys
proof
assume *prefixeq xs* (*ys* @ [*y*])
then obtain *zs* **where** *zs*: *ys* @ [*y*] = *xs* @ *zs* **..**
show *xs* = *ys* @ [*y*] ∨ *prefixeq xs ys*
by (*metis append-Nil2 butlast-append butlast-snoc prefixeqI zs*)
next
assume *xs* = *ys* @ [*y*] ∨ *prefixeq xs ys*
then show *prefixeq xs* (*ys* @ [*y*])
by (*metis prefix-order.eq-iff prefix-order.order-trans prefixeqI*)
qed

lemma *Cons-prefixeq-Cons* [*simp*]: *prefixeq* (*x* # *xs*) (*y* # *ys*) = (*x* = *y* ∧ *prefixeq xs ys*)
by (*auto simp add*: *prefixeq-def*)

lemma *prefixeq-code* [*code*]:
prefixeq [] *xs* ⟷ *True*
prefixeq (*x* # *xs*) [] ⟷ *False*
prefixeq (*x* # *xs*) (*y* # *ys*) ⟷ *x* = *y* ∧ *prefixeq xs ys*
by *simp-all*

lemma *same-prefixeq-prefixeq* [*simp*]: *prefixeq* (*xs* @ *ys*) (*xs* @ *zs*) = *prefixeq ys zs*
by (*induct xs*) *simp-all*

lemma *same-prefixeq-nil* [*iff*]: *prefixeq* (*xs* @ *ys*) *xs* = (*ys* = [])
by (*metis append-Nil2 append-self-conv prefix-order.eq-iff prefixeqI*)

lemma *prefixeq-prefixeq* [*simp*]: *prefixeq xs ys* ⟹ *prefixeq xs* (*ys* @ *zs*)
by (*metis prefix-order.le-less-trans prefixeqI prefixE prefixI*)

lemma *append-prefixeqD*: *prefixeq* (*xs* @ *ys*) *zs* ⟹ *prefixeq xs zs*
by (*auto simp add*: *prefixeq-def*)

theorem *prefixeq-Cons*: *prefixeq xs* (*y* # *ys*) = (*xs* = [] ∨ (∃ *zs*. *xs* = *y* # *zs* ∧ *prefixeq zs ys*))
by (*cases xs*) (*auto simp add*: *prefixeq-def*)

theorem *prefixeq-append*:
prefixeq xs (*ys* @ *zs*) = (*prefixeq xs ys* ∨ (∃ *us*. *xs* = *ys* @ *us* ∧ *prefixeq us zs*))
apply (*induct zs rule*: *rev-induct*)
apply *force*
apply (*simp del*: *append-assoc add*: *append-assoc* [*symmetric*])
apply (*metis append-eq-appendI*)
done

lemma *append-one-prefixeq*:
prefixeq xs ys ⟹ *length xs* < *length ys* ⟹ *prefixeq* (*xs* @ [*ys* ! *length xs*]) *ys*
proof (*unfold prefixeq-def*)

assume *a1*: $\exists zs.\ ys = xs\ @\ zs$
then obtain *sk* :: *'a list* **where** *sk*: $ys = xs\ @\ sk$ **by** *fastforce*
assume *a2*: *length xs* < *length ys*
have *f1*: $\bigwedge v.\ ([]::'a\ list)\ @\ v = v$ **using** *append-Nil2* **by** *simp*
have $[] \neq sk$ **using** *a1 a2 sk less-not-refl* **by** *force*
hence $\exists v.\ xs\ @\ hd\ sk\ \#\ v = ys$ **using** *sk* **by** (*metis hd-Cons-tl*)
thus $\exists zs.\ ys = (xs\ @\ [ys\ !\ length\ xs])\ @\ zs$ **using** *f1* **by** *fastforce*
qed

theorem *prefixeq-length-le*: *prefixeq xs ys* $\Longrightarrow$ *length xs* $\leq$ *length ys*
by (*auto simp add*: *prefixeq-def*)

lemma *prefixeq-same-cases*:
prefixeq (xs_1::*'a list*) *ys* $\Longrightarrow$ *prefixeq* xs_2 *ys* $\Longrightarrow$ *prefixeq* xs_1 xs_2 $\vee$ *prefixeq* xs_2 xs_1
unfolding *prefixeq-def* **by** (*force simp*: *append-eq-append-conv2*)

lemma *set-mono-prefixeq*: *prefixeq xs ys* $\Longrightarrow$ *set xs* $\subseteq$ *set ys*
by (*auto simp add*: *prefixeq-def*)

lemma *take-is-prefixeq*: *prefixeq* (*take n xs*) *xs*
unfolding *prefixeq-def* **by** (*metis append-take-drop-id*)

lemma *map-prefixeqI*: *prefixeq xs ys* $\Longrightarrow$ *prefixeq* (*map f xs*) (*map f ys*)
by (*auto simp*: *prefixeq-def*)

lemma *prefixeq-length-less*: *prefix xs ys* $\Longrightarrow$ *length xs* < *length ys*
by (*auto simp*: *prefix-def prefixeq-def*)

lemma *prefix-simps* [*simp*, *code*]:
prefix xs [] $\longleftrightarrow$ *False*
prefix [] (*x* # *xs*) $\longleftrightarrow$ *True*
prefix (*x* # *xs*) (*y* # *ys*) $\longleftrightarrow$ $x = y \wedge$ *prefix xs ys*
by (*simp-all add*: *prefix-def cong*: *conj-cong*)

lemma *take-prefix*: *prefix xs ys* $\Longrightarrow$ *prefix* (*take n xs*) *ys*
apply (*induct n arbitrary*: *xs ys*)
apply (*case-tac ys*; *simp*)
apply (*metis prefix-order.less-trans prefixI take-is-prefixeq*)
done

lemma *not-prefixeq-cases*:
assumes *pfx*: $\neg$ *prefixeq ps ls*
obtains
(*c1*) $ps \neq []$ **and** $ls = []$
| (*c2*) *a as x xs* **where** $ps = a\#as$ **and** $ls = x\#xs$ **and** $x = a$ **and** $\neg$ *prefixeq as xs*
| (*c3*) *a as x xs* **where** $ps = a\#as$ **and** $ls = x\#xs$ **and** $x \neq a$
proof (*cases ps*)

```
  case Nil
  then show ?thesis using pfx by simp
next
  case (Cons a as)
  note c = ‹ps = a#as›
  show ?thesis
  proof (cases ls)
    case Nil then show ?thesis by (metis append-Nil2 pfx c1 same-prefixeq-nil)
  next
    case (Cons x xs)
    show ?thesis
    proof (cases x = a)
      case True
      have ¬ prefixeq as xs using pfx c Cons True by simp
      with c Cons True show ?thesis by (rule c2)
    next
      case False
      with c Cons show ?thesis by (rule c3)
    qed
  qed
qed

lemma not-prefixeq-induct [consumes 1, case-names Nil Neq Eq]:
  assumes np: ¬ prefixeq ps ls
    and base: ⋀x xs. P (x#xs) []
    and r1: ⋀x xs y ys. x ≠ y ⟹ P (x#xs) (y#ys)
    and r2: ⋀x xs y ys. ⟦ x = y; ¬ prefixeq xs ys; P xs ys ⟧ ⟹ P (x#xs) (y#ys)
  shows P ps ls using np
proof (induct ls arbitrary: ps)
  case Nil then show ?case
    by (auto simp: neq-Nil-conv elim!: not-prefixeq-cases intro!: base)
next
  case (Cons y ys)
  then have npfx: ¬ prefixeq ps (y # ys) by simp
  then obtain x xs where pv: ps = x # xs
    by (rule not-prefixeq-cases) auto
  show ?case by (metis Cons.hyps Cons-prefixeq-Cons npfx pv r1 r2)
qed
```

27.3 Parallel lists

```
definition parallel :: 'a list ⇒ 'a list ⇒ bool  (infixl ∥ 50)
  where (xs ∥ ys) = (¬ prefixeq xs ys ∧ ¬ prefixeq ys xs)

lemma parallelI [intro]: ¬ prefixeq xs ys ⟹ ¬ prefixeq ys xs ⟹ xs ∥ ys
  unfolding parallel-def by blast

lemma parallelE [elim]:
  assumes xs ∥ ys
```

```
  obtains ¬ prefixeq xs ys ∧ ¬ prefixeq ys xs
  using assms unfolding parallel-def by blast

theorem prefixeq-cases:
  obtains prefixeq xs ys | prefix ys xs | xs ‖ ys
  unfolding parallel-def prefix-def by blast

theorem parallel-decomp:
  xs ‖ ys ⟹ ∃ as b bs c cs. b ≠ c ∧ xs = as @ b # bs ∧ ys = as @ c # cs
proof (induct xs rule: rev-induct)
  case Nil
  then have False by auto
  then show ?case ..
next
  case (snoc x xs)
  show ?case
  proof (rule prefixeq-cases)
    assume le: prefixeq xs ys
    then obtain ys' where ys: ys = xs @ ys' ..
    show ?thesis
    proof (cases ys')
      assume ys' = []
      then show ?thesis by (metis append-Nil2 parallelE prefixeqI snoc.prems ys)
    next
      fix c cs assume ys': ys' = c # cs
      have x ≠ c using snoc.prems ys ys' by fastforce
      thus ∃ as b bs c cs. b ≠ c ∧ xs @ [x] = as @ b # bs ∧ ys = as @ c # cs
        using ys ys' by blast
    qed
  next
    assume prefix ys xs
    then have prefixeq ys (xs @ [x]) by (simp add: prefix-def)
    with snoc have False by blast
    then show ?thesis ..
  next
    assume xs ‖ ys
    with snoc obtain as b bs c cs where neq: (b::'a) ≠ c
      and xs: xs = as @ b # bs and ys: ys = as @ c # cs
      by blast
    from xs have xs @ [x] = as @ b # (bs @ [x]) by simp
    with neq ys show ?thesis by blast
  qed
qed

lemma parallel-append: a ‖ b ⟹ a @ c ‖ b @ d
  apply (rule parallelI)
    apply (erule parallelE, erule conjE,
      induct rule: not-prefixeq-induct, simp+)+
  done
```

lemma *parallel-appendI*: $xs \parallel ys \Longrightarrow x = xs \mathbin{@} xs' \Longrightarrow y = ys \mathbin{@} ys' \Longrightarrow x \parallel y$
by (*simp add*: *parallel-append*)

lemma *parallel-commute*: $a \parallel b \longleftrightarrow b \parallel a$
unfolding *parallel-def* **by** *auto*

27.4 Suffix order on lists

definition *suffixeq* :: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where *suffixeq xs ys* = $(\exists zs.\ ys = zs \mathbin{@} xs)$

definition *suffix* :: $'a\ list \Rightarrow 'a\ list \Rightarrow bool$
where *suffix xs ys* $\longleftrightarrow$ $(\exists us.\ ys = us \mathbin{@} xs \wedge us \neq [])$

lemma *suffix-imp-suffixeq*:
suffix xs ys $\Longrightarrow$ *suffixeq xs ys*
by (*auto simp*: *suffixeq-def suffix-def*)

lemma *suffixeqI* [*intro?*]: $ys = zs \mathbin{@} xs \Longrightarrow$ *suffixeq xs ys*
unfolding *suffixeq-def* **by** *blast*

lemma *suffixeqE* [*elim?*]:
assumes *suffixeq xs ys*
obtains *zs* **where** $ys = zs \mathbin{@} xs$
using *assms* **unfolding** *suffixeq-def* **by** *blast*

lemma *suffixeq-refl* [*iff*]: *suffixeq xs xs*
by (*auto simp add*: *suffixeq-def*)
lemma *suffix-trans*:
suffix xs ys $\Longrightarrow$ *suffix ys zs* $\Longrightarrow$ *suffix xs zs*
by (*auto simp*: *suffix-def*)
lemma *suffixeq-trans*: $[\![$*suffixeq xs ys*; *suffixeq ys zs*$]\!] \Longrightarrow$ *suffixeq xs zs*
by (*auto simp add*: *suffixeq-def*)
lemma *suffixeq-antisym*: $[\![$*suffixeq xs ys*; *suffixeq ys xs*$]\!] \Longrightarrow xs = ys$
by (*auto simp add*: *suffixeq-def*)

lemma *suffixeq-tl* [*simp*]: *suffixeq* (*tl xs*) *xs*
by (*induct xs*) (*auto simp*: *suffixeq-def*)

lemma *suffix-tl* [*simp*]: $xs \neq [] \Longrightarrow$ *suffix* (*tl xs*) *xs*
by (*induct xs*) (*auto simp*: *suffix-def*)

lemma *Nil-suffixeq* [*iff*]: *suffixeq* [] *xs*
by (*simp add*: *suffixeq-def*)
lemma *suffixeq-Nil* [*simp*]: (*suffixeq xs* []) = $(xs = [])$
by (*auto simp add*: *suffixeq-def*)

lemma *suffixeq-ConsI*: *suffixeq xs ys* $\Longrightarrow$ *suffixeq xs* $(y \# ys)$

by (*auto simp add*: *suffixeq-def*)
lemma *suffixeq-ConsD*: *suffixeq* (*x* # *xs*) *ys* ⟹ *suffixeq xs ys*
by (*auto simp add*: *suffixeq-def*)

lemma *suffixeq-appendI*: *suffixeq xs ys* ⟹ *suffixeq xs* (*zs* @ *ys*)
by (*auto simp add*: *suffixeq-def*)
lemma *suffixeq-appendD*: *suffixeq* (*zs* @ *xs*) *ys* ⟹ *suffixeq xs ys*
by (*auto simp add*: *suffixeq-def*)

lemma *suffix-set-subset*:
suffix xs ys ⟹ *set xs* ⊆ *set ys* **by** (*auto simp*: *suffix-def*)

lemma *suffixeq-set-subset*:
suffixeq xs ys ⟹ *set xs* ⊆ *set ys* **by** (*auto simp*: *suffixeq-def*)

lemma *suffixeq-ConsD2*: *suffixeq* (*x* # *xs*) (*y* # *ys*) ⟹ *suffixeq xs ys*
proof −
assume *suffixeq* (*x* # *xs*) (*y* # *ys*)
then obtain *zs* **where** *y* # *ys* = *zs* @ *x* # *xs* **..**
then show *?thesis*
by (*induct zs*) (*auto intro*!: *suffixeq-appendI suffixeq-ConsI*)
qed

lemma *suffixeq-to-prefixeq* [*code*]: *suffixeq xs ys* ⟷ *prefixeq* (*rev xs*) (*rev ys*)
proof
assume *suffixeq xs ys*
then obtain *zs* **where** *ys* = *zs* @ *xs* **..**
then have *rev ys* = *rev xs* @ *rev zs* **by** *simp*
then show *prefixeq* (*rev xs*) (*rev ys*) **..**
next
assume *prefixeq* (*rev xs*) (*rev ys*)
then obtain *zs* **where** *rev ys* = *rev xs* @ *zs* **..**
then have *rev* (*rev ys*) = *rev zs* @ *rev* (*rev xs*) **by** *simp*
then have *ys* = *rev zs* @ *xs* **by** *simp*
then show *suffixeq xs ys* **..**
qed

lemma *distinct-suffixeq*: *distinct ys* ⟹ *suffixeq xs ys* ⟹ *distinct xs*
by (*clarsimp elim*!: *suffixeqE*)

lemma *suffixeq-map*: *suffixeq xs ys* ⟹ *suffixeq* (*map f xs*) (*map f ys*)
by (*auto elim*!: *suffixeqE intro*: *suffixeqI*)

lemma *suffixeq-drop*: *suffixeq* (*drop n as*) *as*
unfolding *suffixeq-def*
apply (*rule exI* [**where** *x* = *take n as*])
apply *simp*
done

lemma *suffixeq-take*: *suffixeq xs ys* $\Longrightarrow$ *ys* = *take* (*length ys* − *length xs*) *ys* @ *xs*
by (*auto elim*!: *suffixeqE*)

lemma *suffixeq-suffix-reflclp-conv*: *suffixeq* = *suffix*$^{==}$
proof (*intro ext iffI*)
fix *xs ys* :: *'a list*
assume *suffixeq xs ys*
show *suffix*$^{==}$ *xs ys*
proof
assume *xs* ≠ *ys*
with ‹*suffixeq xs ys*› **show** *suffix xs ys*
by (*auto simp*: *suffixeq-def suffix-def*)
qed
next
fix *xs ys* :: *'a list*
assume *suffix*$^{==}$ *xs ys*
then show *suffixeq xs ys*
proof
assume *suffix xs ys* **then show** *suffixeq xs ys*
by (*rule suffix-imp-suffixeq*)
next
assume *xs* = *ys* **then show** *suffixeq xs ys*
by (*auto simp*: *suffixeq-def*)
qed
qed

lemma *parallelD1*: *x* ∥ *y* $\Longrightarrow$ ¬ *prefixeq x y*
by *blast*

lemma *parallelD2*: *x* ∥ *y* $\Longrightarrow$ ¬ *prefixeq y x*
by *blast*

lemma *parallel-Nil1* [*simp*]: ¬ *x* ∥ []
unfolding *parallel-def* **by** *simp*

lemma *parallel-Nil2* [*simp*]: ¬ [] ∥ *x*
unfolding *parallel-def* **by** *simp*

lemma *Cons-parallelI1*: *a* ≠ *b* $\Longrightarrow$ *a* # *as* ∥ *b* # *bs*
by *auto*

lemma *Cons-parallelI2*: ⟦ *a* = *b*; *as* ∥ *bs* ⟧ $\Longrightarrow$ *a* # *as* ∥ *b* # *bs*
by (*metis Cons-prefixeq-Cons parallelE parallelI*)

lemma *not-equal-is-parallel*:
assumes *neq*: *xs* ≠ *ys*
and *len*: *length xs* = *length ys*
shows *xs* ∥ *ys*
using *len neq*

proof (*induct rule*: *list-induct2*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a as b bs*)
have *ih*: $as \neq bs \Longrightarrow as \parallel bs$ **by** *fact*
show *?case*
proof (*cases a = b*)
case *True*
then have $as \neq bs$ **using** *Cons* **by** *simp*
then show *?thesis* **by** (*rule Cons-parallelI2* [*OF True ih*])
next
case *False*
then show *?thesis* **by** (*rule Cons-parallelI1*)
qed
qed

lemma *suffix-reflclp-conv*: $suffix^{==} = suffixeq$
by (*intro ext*) (*auto simp*: *suffixeq-def suffix-def*)

lemma *suffix-lists*: *suffix xs ys* $\Longrightarrow$ *ys* $\in$ *lists A* $\Longrightarrow$ *xs* $\in$ *lists A*
unfolding *suffix-def* **by** *auto*

27.5 Homeomorphic embedding on lists

inductive *list-emb* :: $('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \Rightarrow bool$
for $P :: ('a \Rightarrow 'a \Rightarrow bool)$
where
list-emb-Nil [*intro*, *simp*]: *list-emb P* [] *ys*
| *list-emb-Cons* [*intro*] : *list-emb P xs ys* $\Longrightarrow$ *list-emb P xs* (*y*#*ys*)
| *list-emb-Cons2* [*intro*]: *P x y* $\Longrightarrow$ *list-emb P xs ys* $\Longrightarrow$ *list-emb P* (*x*#*xs*) (*y*#*ys*)

lemma *list-emb-mono*:
assumes $\bigwedge x\ y.\ P\ x\ y \longrightarrow Q\ x\ y$
shows *list-emb P xs ys* $\longrightarrow$ *list-emb Q xs ys*
proof
assume *list-emb P xs ys*
then show *list-emb Q xs ys* **by** (*induct*) (*auto simp*: *assms*)
qed

lemma *list-emb-Nil2* [*simp*]:
assumes *list-emb P xs* [] **shows** *xs* = []
using *assms* **by** (*cases rule*: *list-emb.cases*) *auto*

lemma *list-emb-refl*:
assumes $\bigwedge x.\ x \in set\ xs \Longrightarrow P\ x\ x$
shows *list-emb P xs xs*
using *assms* **by** (*induct xs*) *auto*

lemma *list-emb-Cons-Nil* [*simp*]: *list-emb P* (*x*#*xs*) [] = *False*
proof −
{ assume *list-emb P* (*x*#*xs*) []
from *list-emb-Nil2* [*OF this*] **have** *False* **by** *simp*
} moreover {
assume *False*
then have *list-emb P* (*x*#*xs*) [] **by** *simp*
} ultimately show *?thesis* **by** *blast*
qed

lemma *list-emb-append2* [*intro*]: *list-emb P xs ys* ⟹ *list-emb P xs* (*zs* @ *ys*)
by (*induct zs*) *auto*

lemma *list-emb-prefix* [*intro*]:
assumes *list-emb P xs ys* **shows** *list-emb P xs* (*ys* @ *zs*)
using *assms*
by (*induct arbitrary*: *zs*) *auto*

lemma *list-emb-ConsD*:
assumes *list-emb P* (*x*#*xs*) *ys*
shows ∃ *us v vs*. *ys* = *us* @ *v* # *vs* ∧ *P x v* ∧ *list-emb P xs vs*
using *assms*
proof (*induct x* ≡ *x* # *xs ys arbitrary*: *x xs*)
case *list-emb-Cons*
then show *?case* **by** (*metis append-Cons*)
next
case (*list-emb-Cons2 x y xs ys*)
then show *?case* **by** *blast*
qed

lemma *list-emb-appendD*:
assumes *list-emb P* (*xs* @ *ys*) *zs*
shows ∃ *us vs*. *zs* = *us* @ *vs* ∧ *list-emb P xs us* ∧ *list-emb P ys vs*
using *assms*
proof (*induction xs arbitrary*: *ys zs*)
case *Nil* **then show** *?case* **by** *auto*
next
case (*Cons x xs*)
then obtain *us v vs* **where**
zs: *zs* = *us* @ *v* # *vs* **and** *p*: *P x v* **and** *lh*: *list-emb P* (*xs* @ *ys*) *vs*
by (*auto dest*: *list-emb-ConsD*)
obtain *sk*$_0$ :: *'a list* ⇒ *'a list* ⇒ *'a list* **and** *sk*$_1$ :: *'a list* ⇒ *'a list* ⇒ *'a list*
where
sk: ∀ *x*$_0$ *x*$_1$. ¬ *list-emb P* (*xs* @ *x*$_0$) *x*$_1$ ∨ *sk*$_0$ *x*$_0$ *x*$_1$ @ *sk*$_1$ *x*$_0$ *x*$_1$ = *x*$_1$ ∧ *list-emb P xs* (*sk*$_0$ *x*$_0$ *x*$_1$) ∧ *list-emb P x*$_0$ (*sk*$_1$ *x*$_0$ *x*$_1$)
using *Cons*(*1*) **by** (*metis* (*no-types*))
hence ∀ *x*$_2$. *list-emb P* (*x* # *xs*) (*x*$_2$ @ *v* # *sk*$_0$ *ys vs*) **using** *p lh* **by** *auto*
thus *?case* **using** *lh zs sk* **by** (*metis* (*no-types*) *append-Cons append-assoc*)
qed

lemma *list-emb-suffix*:
assumes *list-emb P xs ys* **and** *suffix ys zs*
shows *list-emb P xs zs*
using *assms(2)* **and** *list-emb-append2* [*OF assms(1)*] **by** (*auto simp*: *suffix-def*)

lemma *list-emb-suffixeq*:
assumes *list-emb P xs ys* **and** *suffixeq ys zs*
shows *list-emb P xs zs*
using *assms* **and** *list-emb-suffix* **unfolding** *suffixeq-suffix-reflclp-conv* **by** *auto*

lemma *list-emb-length*: *list-emb P xs ys* $\Longrightarrow$ *length xs* $\leq$ *length ys*
by (*induct rule*: *list-emb.induct*) *auto*

lemma *list-emb-trans*:
assumes $\bigwedge x\ y\ z.\ [\![x \in set\ xs;\ y \in set\ ys;\ z \in set\ zs;\ P\ x\ y;\ P\ y\ z]\!] \Longrightarrow P\ x\ z$
shows $[\![list\text{-}emb\ P\ xs\ ys;\ list\text{-}emb\ P\ ys\ zs]\!] \Longrightarrow list\text{-}emb\ P\ xs\ zs$
proof −
assume *list-emb P xs ys* **and** *list-emb P ys zs*
then show *list-emb P xs zs* **using** *assms*
proof (*induction arbitrary*: *zs*)
case *list-emb-Nil* **show** *?case* **by** *blast*
next
case (*list-emb-Cons xs ys y*)
from *list-emb-ConsD* [*OF* ‹*list-emb P* (*y*#*ys*) *zs*›] **obtain** *us v vs*
where *zs*: *zs* = *us* @ *v* # *vs* **and** $P^{==}$ *y v* **and** *list-emb P ys vs* **by** *blast*
then have *list-emb P ys* (*v*#*vs*) **by** *blast*
then have *list-emb P ys zs* **unfolding** *zs* **by** (*rule list-emb-append2*)
from *list-emb-Cons.IH* [*OF this*] **and** *list-emb-Cons.prems* **show** *?case* **by** *auto*
next
case (*list-emb-Cons2 x y xs ys*)
from *list-emb-ConsD* [*OF* ‹*list-emb P* (*y*#*ys*) *zs*›] **obtain** *us v vs*
where *zs*: *zs* = *us* @ *v* # *vs* **and** *P y v* **and** *list-emb P ys vs* **by** *blast*
with *list-emb-Cons2* **have** *list-emb P xs vs* **by** *auto*
moreover have *P x v*
proof −
from *zs* **have** *v* ∈ *set zs* **by** *auto*
moreover have *x* ∈ *set* (*x*#*xs*) **and** *y* ∈ *set* (*y*#*ys*) **by** *simp-all*
ultimately show *?thesis*
using ‹*P x y*› **and** ‹*P y v*› **and** *list-emb-Cons2*
by *blast*
qed
ultimately have *list-emb P* (*x*#*xs*) (*v*#*vs*) **by** *blast*
then show *?case* **unfolding** *zs* **by** (*rule list-emb-append2*)
qed
qed

lemma *list-emb-set*:

assumes *list-emb P xs ys* **and** $x \in$ *set xs*
obtains y **where** $y \in$ *set ys* **and** *P x y*
using *assms* **by** (*induct*) *auto*

27.6 Sublists (special case of homeomorphic embedding)

abbreviation *sublisteq* :: *'a list* ⇒ *'a list* ⇒ *bool*
where *sublisteq xs ys* ≡ *list-emb* (*op* =) *xs ys*

lemma *sublisteq-Cons2*: *sublisteq xs ys* ⟹ *sublisteq* (*x*#*xs*) (*x*#*ys*) **by** *auto*

lemma *sublisteq-same-length*:
assumes *sublisteq xs ys* **and** *length xs* = *length ys* **shows** *xs* = *ys*
using *assms* **by** (*induct*) (*auto dest*: *list-emb-length*)

lemma *not-sublisteq-length* [*simp*]: *length ys* < *length xs* ⟹ ¬ *sublisteq xs ys*
by (*metis list-emb-length linorder-not-less*)

lemma [*code*]:
list-emb P [] *ys* ⟷ *True*
list-emb P (*x*#*xs*) [] ⟷ *False*
by (*simp-all*)

lemma *sublisteq-Cons'*: *sublisteq* (*x*#*xs*) *ys* ⟹ *sublisteq xs ys*
by (*induct xs*, *simp*, *blast dest*: *list-emb-ConsD*)

lemma *sublisteq-Cons2'*:
assumes *sublisteq* (*x*#*xs*) (*x*#*ys*) **shows** *sublisteq xs ys*
using *assms* **by** (*cases*) (*rule sublisteq-Cons'*)

lemma *sublisteq-Cons2-neq*:
assumes *sublisteq* (*x*#*xs*) (*y*#*ys*)
shows *x* ≠ *y* ⟹ *sublisteq* (*x*#*xs*) *ys*
using *assms* **by** (*cases*) *auto*

lemma *sublisteq-Cons2-iff* [*simp*, *code*]:
sublisteq (*x*#*xs*) (*y*#*ys*) = (*if x* = *y then sublisteq xs ys else sublisteq* (*x*#*xs*) *ys*)
by (*metis list-emb-Cons sublisteq-Cons2 sublisteq-Cons2' sublisteq-Cons2-neq*)

lemma *sublisteq-append'*: *sublisteq* (*zs* @ *xs*) (*zs* @ *ys*) ⟷ *sublisteq xs ys*
by (*induct zs*) *simp-all*

lemma *sublisteq-refl* [*simp*, *intro*!]: *sublisteq xs xs* **by** (*induct xs*) *simp-all*

lemma *sublisteq-antisym*:
assumes *sublisteq xs ys* **and** *sublisteq ys xs*
shows *xs* = *ys*
using *assms*
proof (*induct*)

```
  case list-emb-Nil
  from list-emb-Nil2 [OF this] show ?case by simp
next
  case list-emb-Cons2
  thus ?case by simp
next
  case list-emb-Cons
  hence False using sublisteq-Cons' by fastforce
  thus ?case ..
qed

lemma sublisteq-trans: sublisteq xs ys ⟹ sublisteq ys zs ⟹ sublisteq xs zs
  by (rule list-emb-trans [of - - - op =]) auto

lemma sublisteq-append-le-same-iff: sublisteq (xs @ ys) ys ⟷ xs = []
  by (auto dest: list-emb-length)

lemma list-emb-append-mono:
  ⟦ list-emb P xs xs'; list-emb P ys ys' ⟧ ⟹ list-emb P (xs@ys) (xs'@ys')
  apply (induct rule: list-emb.induct)
    apply (metis eq-Nil-appendI list-emb-append2)
   apply (metis append-Cons list-emb-Cons)
  apply (metis append-Cons list-emb-Cons2)
  done
```

27.7 Appending elements

```
lemma sublisteq-append [simp]:
  sublisteq (xs @ zs) (ys @ zs) ⟷ sublisteq xs ys (is ?l = ?r)
proof
  { fix xs' ys' xs ys zs :: 'a list assume sublisteq xs' ys'
    then have xs' = xs @ zs & ys' = ys @ zs ⟶ sublisteq xs ys
    proof (induct arbitrary: xs ys zs)
      case list-emb-Nil show ?case by simp
    next
      case (list-emb-Cons xs' ys' x)
      { assume ys=[] then have ?case using list-emb-Cons(1) by auto }
      moreover
      { fix us assume ys = x#us
     then have ?case using list-emb-Cons(2) by(simp add: list-emb.list-emb-Cons)
}
      ultimately show ?case by (auto simp:Cons-eq-append-conv)
    next
      case (list-emb-Cons2 x y xs' ys')
      { assume xs=[] then have ?case using list-emb-Cons2(1) by auto }
      moreover
    { fix us vs assume xs=x#us ys=x#vs then have ?case using list-emb-Cons2
by auto}
      moreover
```

{ fix *us* **assume** *xs=x#us ys=[]* **then have** *?case* **using** *list-emb-Cons2(2)* **by** *bestsimp* **}**
ultimately show *?case* **using** ‹*op = x y*› **by** (*auto simp*: *Cons-eq-append-conv*)
qed }
moreover assume *?l*
ultimately show *?r* **by** *blast*
next
assume *?r* **then show** *?l* **by** (*metis list-emb-append-mono sublisteq-refl*)
qed

lemma *sublisteq-drop-many*: *sublisteq xs ys* ⟹ *sublisteq xs* (*zs* @ *ys*)
by (*induct zs*) *auto*

lemma *sublisteq-rev-drop-many*: *sublisteq xs ys* ⟹ *sublisteq xs* (*ys* @ *zs*)
by (*metis append-Nil2 list-emb-Nil list-emb-append-mono*)

27.8 Relation to standard list operations

lemma *sublisteq-map*:
assumes *sublisteq xs ys* **shows** *sublisteq* (*map f xs*) (*map f ys*)
using *assms* **by** (*induct*) *auto*

lemma *sublisteq-filter-left* [*simp*]: *sublisteq* (*filter P xs*) *xs*
by (*induct xs*) *auto*

lemma *sublisteq-filter* [*simp*]:
assumes *sublisteq xs ys* **shows** *sublisteq* (*filter P xs*) (*filter P ys*)
using *assms* **by** *induct auto*

lemma *sublisteq xs ys* ⟷ (∃ *N*. *xs* = *sublist ys N*) (**is** *?L* = *?R*)
proof
assume *?L*
then show *?R*
proof (*induct*)
case *list-emb-Nil* **show** *?case* **by** (*metis sublist-empty*)
next
case (*list-emb-Cons xs ys x*)
then obtain *N* **where** *xs* = *sublist ys N* **by** *blast*
then have *xs* = *sublist* (*x#ys*) (*Suc* ‘ *N*)
by (*clarsimp simp add*:*sublist-Cons inj-image-mem-iff*)
then show *?case* **by** *blast*
next
case (*list-emb-Cons2 x y xs ys*)
then obtain *N* **where** *xs* = *sublist ys N* **by** *blast*
then have *x#xs* = *sublist* (*x#ys*) (*insert 0* (*Suc* ‘ *N*))
by (*clarsimp simp add*:*sublist-Cons inj-image-mem-iff*)
moreover from *list-emb-Cons2* **have** *x* = *y* **by** *simp*
ultimately show *?case* **by** *blast*
qed

next
assume *?R*
then obtain *N* **where** *xs* = *sublist ys N* **..**
moreover have *sublisteq* (*sublist ys N*) *ys*
proof (*induct ys arbitrary*: *N*)
case *Nil* **show** *?case* **by** *simp*
next
case *Cons* **then show** *?case* **by** (*auto simp*: *sublist-Cons*)
qed
ultimately show *?L* **by** *simp*
qed

end

28 Linear Temporal Logic on Streams

theory *Linear-Temporal-Logic-on-Streams*
imports *Stream Sublist Extended-Nat Infinite-Set*
begin

29 Preliminaries

lemma *shift-prefix*:
assumes *xl* @− *xs* = *yl* @− *ys* **and** *length xl* ≤ *length yl*
shows *prefixeq xl yl*
using *assms* **proof**(*induct xl arbitrary*: *yl xs ys*)
case (*Cons x xl yl xs ys*)
thus *?case* **by** (*cases yl*) *auto*
qed *auto*

lemma *shift-prefix-cases*:
assumes *xl* @− *xs* = *yl* @− *ys*
shows *prefixeq xl yl* ∨ *prefixeq yl xl*
using *shift-prefix*[*OF assms*]
by (*cases length xl* ≤ *length yl*) (*metis*, *metis assms nat-le-linear shift-prefix*)

30 Linear temporal logic

abbreviation (*input*) *IMPL* (**infix** *impl 60*)
where *φ impl ψ* ≡ λ *xs*. *φ xs* ⟶ *ψ xs*

abbreviation (*input*) *OR* (**infix** *or 60*)
where *φ or ψ* ≡ λ *xs*. *φ xs* ∨ *ψ xs*

abbreviation (*input*) *AND* (**infix** *aand 60*)
where *φ aand ψ* ≡ λ *xs*. *φ xs* ∧ *ψ xs*

abbreviation (*input*) *not φ* ≡ λ *xs*. ¬ *φ xs*

abbreviation (*input*) *true* ≡ λ *xs*. *True*

abbreviation (*input*) *false* ≡ λ *xs*. *False*

lemma *impl-not-or*: φ *impl* ψ = (*not* φ) *or* ψ
by *blast*

lemma *not-or*: *not* (φ *or* ψ) = (*not* φ) *aand* (*not* ψ)
by *blast*

lemma *not-aand*: *not* (φ *aand* ψ) = (*not* φ) *or* (*not* ψ)
by *blast*

lemma *non-not*[*simp*]: *not* (*not* φ) = φ **by** *simp*

fun *holds* **where** *holds P xs* ⟷ *P* (*shd xs*)
fun *nxt* **where** *nxt* φ *xs* = φ (*stl xs*)

definition *HLD s* = *holds* (λ*x*. *x* ∈ *s*)

abbreviation *HLD-nxt* (**infixr** · *65*) **where**
s · *P* ≡ *HLD s aand nxt P*

context
notes [[*inductive-internals*]]
begin

inductive *ev* **for** φ **where**
base: φ *xs* ⟹ *ev* φ *xs*
|
step: *ev* φ (*stl xs*) ⟹ *ev* φ *xs*

coinductive *alw* **for** φ **where**
alw: ⟦φ *xs*; *alw* φ (*stl xs*)⟧ ⟹ *alw* φ *xs*

coinductive *UNTIL* (**infix** *until 60*) **for** φ ψ **where**
base: ψ *xs* ⟹ (φ *until* ψ) *xs*
|
step: ⟦φ *xs*; (φ *until* ψ) (*stl xs*)⟧ ⟹ (φ *until* ψ) *xs*

end

lemma *holds-mono*:
assumes *holds*: *holds P xs* **and** *0*: ⋀ *x*. *P x* ⟹ *Q x*
shows *holds Q xs*
using *assms* **by** *auto*

lemma *holds-aand*:
(*holds P aand holds Q*) *steps* $\longleftrightarrow$ *holds* (λ *step*. *P step* $\wedge$ *Q step*) *steps* **by** *auto*

lemma *HLD-iff*: *HLD s* ω $\longleftrightarrow$ *shd* $\omega \in s$
by (*simp add*: *HLD-def*)

lemma *HLD-Stream*[*simp*]: *HLD X* (*x* ## ω) $\longleftrightarrow$ $x \in X$
by (*simp add*: *HLD-iff*)

lemma *nxt-mono*:
assumes *nxt*: *nxt* φ *xs* **and** *0*: $\bigwedge$ *xs*. φ *xs* $\Longrightarrow$ ψ *xs*
shows *nxt* ψ *xs*
using *assms* **by** *auto*

declare *ev.intros*[*intro*]
declare *alw.cases*[*elim*]

lemma *ev-induct-strong*[*consumes 1*, *case-names base step*]:
ev φ *x* $\Longrightarrow$ ($\bigwedge$*xs*. φ *xs* $\Longrightarrow$ *P xs*) $\Longrightarrow$ ($\bigwedge$*xs*. *ev* φ (*stl xs*) $\Longrightarrow$ $\neg$ φ *xs* $\Longrightarrow$ *P* (*stl xs*) $\Longrightarrow$ *P xs*) $\Longrightarrow$ *P x*
by (*induct rule*: *ev.induct*) *auto*

lemma *alw-coinduct*[*consumes 1*, *case-names alw stl*]:
X x $\Longrightarrow$ ($\bigwedge$*x*. *X x* $\Longrightarrow$ φ *x*) $\Longrightarrow$ ($\bigwedge$*x*. *X x* $\Longrightarrow$ $\neg$ *alw* φ (*stl x*) $\Longrightarrow$ *X* (*stl x*)) $\Longrightarrow$ *alw* φ *x*
using *alw.coinduct*[*of X x* φ] **by** *auto*

lemma *ev-mono*:
assumes *ev*: *ev* φ *xs* **and** *0*: $\bigwedge$ *xs*. φ *xs* $\Longrightarrow$ ψ *xs*
shows *ev* ψ *xs*
using *ev* **by** *induct* (*auto simp*: *0*)

lemma *alw-mono*:
assumes *alw*: *alw* φ *xs* **and** *0*: $\bigwedge$ *xs*. φ *xs* $\Longrightarrow$ ψ *xs*
shows *alw* ψ *xs*
using *alw* **by** *coinduct* (*auto simp*: *0*)

lemma *until-monoL*:
assumes *until*: (φ*1 until* ψ) *xs* **and** *0*: $\bigwedge$ *xs*. φ*1 xs* $\Longrightarrow$ φ*2 xs*
shows (φ*2 until* ψ) *xs*
using *until* **by** *coinduct* (*auto elim*: *UNTIL.cases simp*: *0*)

lemma *until-monoR*:
assumes *until*: (φ *until* ψ*1*) *xs* **and** *0*: $\bigwedge$ *xs*. ψ*1 xs* $\Longrightarrow$ ψ*2 xs*
shows (φ *until* ψ*2*) *xs*
using *until* **by** *coinduct* (*auto elim*: *UNTIL.cases simp*: *0*)

lemma *until-mono*:

assumes *until*: $(\varphi 1\ until\ \psi 1)\ xs$ **and**
0: $\bigwedge xs.\ \varphi 1\ xs \Longrightarrow \varphi 2\ xs \bigwedge xs.\ \psi 1\ xs \Longrightarrow \psi 2\ xs$
shows $(\varphi 2\ until\ \psi 2)\ xs$
using *until* **by** *coinduct* (*auto elim*: *UNTIL.cases simp*: *0*)

lemma *until-false*: $\varphi\ until\ false = alw\ \varphi$
proof−
  {fix *xs* **assume** $(\varphi\ until\ false)\ xs$ **hence** $alw\ \varphi\ xs$
   by *coinduct* (*auto elim*: *UNTIL.cases*)
  }
  moreover
  {fix *xs* **assume** $alw\ \varphi\ xs$ **hence** $(\varphi\ until\ false)\ xs$
   by *coinduct auto*
  }
  ultimately show *?thesis* **by** *blast*
qed

lemma *ev-nxt*: $ev\ \varphi = (\varphi\ or\ nxt\ (ev\ \varphi))$
by (*rule ext*) (*metis ev.simps nxt.simps*)

lemma *alw-nxt*: $alw\ \varphi = (\varphi\ aand\ nxt\ (alw\ \varphi))$
by (*rule ext*) (*metis alw.simps nxt.simps*)

lemma *ev-ev*[*simp*]: $ev\ (ev\ \varphi) = ev\ \varphi$
proof−
  {fix *xs*
   assume $ev\ (ev\ \varphi)\ xs$ **hence** $ev\ \varphi\ xs$
   by *induct auto*
  }
  thus *?thesis* **by** *auto*
qed

lemma *alw-alw*[*simp*]: $alw\ (alw\ \varphi) = alw\ \varphi$
proof−
  {fix *xs*
   assume $alw\ \varphi\ xs$ **hence** $alw\ (alw\ \varphi)\ xs$
   by *coinduct auto*
  }
  thus *?thesis* **by** *auto*
qed

lemma *ev-shift*:
assumes $ev\ \varphi\ xs$
shows $ev\ \varphi\ (xl\ @-\ xs)$
using *assms* **by** (*induct xl*) *auto*

lemma *ev-imp-shift*:
assumes $ev\ \varphi\ xs$ **shows** $\exists\ xl\ xs2.\ xs = xl\ @-\ xs2 \wedge \varphi\ xs2$
using *assms* **by** *induct* (*metis shift.simps*(*1*), *metis shift.simps*(*2*) *stream.collapse*)+

lemma *alw-ev-shift*: *alw* φ *xs1* $\Longrightarrow$ *ev* (*alw* φ) (*xl* @− *xs1*)
by (*auto intro*: *ev-shift*)

lemma *alw-shift*:
assumes *alw* φ (*xl* @− *xs*)
shows *alw* φ *xs*
using *assms* **by** (*induct xl*) *auto*

lemma *ev-ex-nxt*:
assumes *ev* φ *xs*
shows $\exists$ *n*. (*nxt* ^^ *n*) φ *xs*
using *assms* **proof** *induct*
case (*base xs*) **thus** *?case* **by** (*intro exI*[*of* - *0*]) *auto*
next
case (*step xs*)
then obtain *n* **where** (*nxt* ^^ *n*) φ (*stl xs*) **by** *blast*
thus *?case* **by** (*intro exI*[*of* - *Suc n*]) (*metis funpow.simps*(*2*) *nxt.simps o-def*)
qed

lemma *alw-sdrop*:
assumes *alw* φ *xs* **shows** *alw* φ (*sdrop n xs*)
by (*metis alw-shift assms stake-sdrop*)

lemma *nxt-sdrop*: (*nxt* ^^ *n*) φ *xs* $\longleftrightarrow$ φ (*sdrop n xs*)
by (*induct n arbitrary*: *xs*) *auto*

definition *wait* φ *xs* $\equiv$ *LEAST n*. (*nxt* ^^ *n*) φ *xs*

lemma *nxt-wait*:
assumes *ev* φ *xs* **shows** (*nxt* ^^ (*wait* φ *xs*)) φ *xs*
unfolding *wait-def* **using** *ev-ex-nxt*[*OF assms*] **by**(*rule LeastI-ex*)

lemma *nxt-wait-least*:
assumes *ev*: *ev* φ *xs* **and** *nxt*: (*nxt* ^^ *n*) φ *xs* **shows** *wait* φ *xs* $\leq$ *n*
unfolding *wait-def* **using** *ev-ex-nxt*[*OF ev*] **by** (*metis Least-le nxt*)

lemma *sdrop-wait*:
assumes *ev* φ *xs* **shows** φ (*sdrop* (*wait* φ *xs*) *xs*)
using *nxt-wait*[*OF assms*] **unfolding** *nxt-sdrop* **.**

lemma *sdrop-wait-least*:
assumes *ev*: *ev* φ *xs* **and** *nxt*: φ (*sdrop n xs*) **shows** *wait* φ *xs* $\leq$ *n*
using *assms nxt-wait-least* **unfolding** *nxt-sdrop* **by** *auto*

lemma *nxt-ev*: (*nxt* ^^ *n*) φ *xs* $\Longrightarrow$ *ev* φ *xs*
by (*induct n arbitrary*: *xs*) *auto*

lemma *not-ev*: *not* (*ev* φ) = *alw* (*not* φ)

proof(*rule ext*, *safe*)
fix *xs* **assume** *not* (*ev* φ) *xs* **thus** *alw* (*not* φ) *xs*
by (*coinduct*) *auto*
next
fix *xs* **assume** *ev* φ *xs* **and** *alw* (*not* φ) *xs* **thus** *False*
by (*induct*) *auto*
qed

lemma *not-alw*: *not* (*alw* φ) = *ev* (*not* φ)
proof−
have *not* (*alw* φ) = *not* (*alw* (*not* (*not* φ))) **by** *simp*
also have ... = *ev* (*not* φ) **unfolding** *not-ev*[*symmetric*] **by** *simp*
finally show *?thesis* **.**
qed

lemma *not-ev-not*[*simp*]: *not* (*ev* (*not* φ)) = *alw* φ
unfolding *not-ev* **by** *simp*

lemma *not-alw-not*[*simp*]: *not* (*alw* (*not* φ)) = *ev* φ
unfolding *not-alw* **by** *simp*

lemma *alw-ev-sdrop*:
assumes *alw* (*ev* φ) (*sdrop m xs*)
shows *alw* (*ev* φ) *xs*
using *assms*
by *coinduct* (*metis alw-nxt ev-shift funpow-swap1 nxt.simps nxt-sdrop stake-sdrop*)

lemma *ev-alw-imp-alw-ev*:
assumes *ev* (*alw* φ) *xs* **shows** *alw* (*ev* φ) *xs*
using *assms* **by** *induct* (*metis* (*full-types*) *alw-mono ev.base*, *metis alw alw-nxt ev.step*)

lemma *alw-aand*: *alw* (φ *aand* ψ) = *alw* φ *aand alw* ψ
proof−
{fix *xs* **assume** *alw* (φ *aand* ψ) *xs* **hence** (*alw* φ *aand alw* ψ) *xs*
by (*auto elim*: *alw-mono*)
}
moreover
{fix *xs* **assume** (*alw* φ *aand alw* ψ) *xs* **hence** *alw* (φ *aand* ψ) *xs*
by *coinduct auto*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *ev-or*: *ev* (φ *or* ψ) = *ev* φ *or ev* ψ
proof−
{fix *xs* **assume** (*ev* φ *or ev* ψ) *xs* **hence** *ev* (φ *or* ψ) *xs*
by (*auto elim*: *ev-mono*)
}

moreover
{fix *xs* **assume** *ev* (*φ or ψ*) *xs* **hence** (*ev φ or ev ψ*) *xs*
by *induct auto*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *ev-alw-aand*:
assumes *φ*: *ev* (*alw φ*) *xs* **and** *ψ*: *ev* (*alw ψ*) *xs*
shows *ev* (*alw* (*φ aand ψ*)) *xs*
proof−
obtain *xl xs1* **where** *xs1*: *xs* = *xl* @− *xs1* **and** *φφ*: *alw φ xs1*
using *φ* **by** (*metis ev-imp-shift*)
moreover obtain *yl ys1* **where** *xs2*: *xs* = *yl* @− *ys1* **and** *ψψ*: *alw ψ ys1*
using *ψ* **by** (*metis ev-imp-shift*)
ultimately have *0*: *xl* @− *xs1* = *yl* @− *ys1* **by** *auto*
hence *prefixeq xl yl* ∨ *prefixeq yl xl* **using** *shift-prefix-cases* **by** *auto*
thus *?thesis* **proof**
assume *prefixeq xl yl*
then obtain *yl1* **where** *yl*: *yl* = *xl* @ *yl1* **by** (*elim prefixeqE*)
have *xs1′*: *xs1* = *yl1* @− *ys1* **using** *0* **unfolding** *yl* **by** *simp*
have *alw φ ys1* **using** *φφ* **unfolding** *xs1′* **by** (*metis alw-shift*)
hence *alw* (*φ aand ψ*) *ys1* **using** *ψψ* **unfolding** *alw-aand* **by** *auto*
thus *?thesis* **unfolding** *xs2* **by** (*auto intro*: *alw-ev-shift*)
next
assume *prefixeq yl xl*
then obtain *xl1* **where** *xl*: *xl* = *yl* @ *xl1* **by** (*elim prefixeqE*)
have *ys1′*: *ys1* = *xl1* @− *xs1* **using** *0* **unfolding** *xl* **by** *simp*
have *alw ψ xs1* **using** *ψψ* **unfolding** *ys1′* **by** (*metis alw-shift*)
hence *alw* (*φ aand ψ*) *xs1* **using** *φφ* **unfolding** *alw-aand* **by** *auto*
thus *?thesis* **unfolding** *xs1* **by** (*auto intro*: *alw-ev-shift*)
qed
qed

lemma *ev-alw-alw-impl*:
assumes *ev* (*alw φ*) *xs* **and** *alw* (*alw φ impl ev ψ*) *xs*
shows *ev ψ xs*
using *assms* **by** *induct auto*

lemma *ev-alw-stl*[*simp*]: *ev* (*alw φ*) (*stl x*) ⟷ *ev* (*alw φ*) *x*
by (*metis* (*full-types*) *alw-nxt ev-nxt nxt.simps*)

lemma *alw-alw-impl-ev*:
alw (*alw φ impl ev ψ*) = (*ev* (*alw φ*) *impl alw* (*ev ψ*)) (**is** *?A* = *?B*)
proof−
{fix *xs* **assume** *?A xs* ∧ *ev* (*alw φ*) *xs* **hence** *alw* (*ev ψ*) *xs*
by *coinduct* (*auto elim*: *ev-alw-alw-impl*)
}
moreover

{fix *xs* **assume** *?B xs* **hence** *?A xs*
by *coinduct auto*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *ev-alw-impl*:
assumes *ev* φ *xs* **and** *alw* (φ *impl* ψ) *xs* **shows** *ev* ψ *xs*
using *assms* **by** *induct auto*

lemma *ev-alw-impl-ev*:
assumes *ev* φ *xs* **and** *alw* (φ *impl ev* ψ) *xs* **shows** *ev* ψ *xs*
using *ev-alw-impl*[*OF assms*] **by** *simp*

lemma *alw-mp*:
assumes *alw* φ *xs* **and** *alw* (φ *impl* ψ) *xs*
shows *alw* ψ *xs*
proof−
{assume *alw* φ *xs* $\wedge$ *alw* (φ *impl* ψ) *xs* **hence** *?thesis*
by *coinduct auto*
}
thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *all-imp-alw*:
assumes $\bigwedge$ *xs*. φ *xs* **shows** *alw* φ *xs*
proof−
{assume $\forall$ *xs*. φ *xs*
hence *?thesis* **by** *coinduct auto*
}
thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *alw-impl-ev-alw*:
assumes *alw* (φ *impl ev* ψ) *xs*
shows *alw* (*ev* φ *impl ev* ψ) *xs*
using *assms* **by** *coinduct* (*auto dest*: *ev-alw-impl*)

lemma *ev-holds-sset*:
ev (*holds P*) *xs* $\longleftrightarrow$ ($\exists$ *x* $\in$ *sset xs*. *P x*) (**is** *?L* $\longleftrightarrow$ *?R*)
proof *safe*
assume *?L* **thus** *?R* **by** *induct* (*metis holds.simps stream.set-sel*(*1*), *metis stl-sset*)
next
fix *x* **assume** *x* $\in$ *sset xs P x*
thus *?L* **by** (*induct rule*: *sset-induct*) (*simp-all add*: *ev.base ev.step*)
qed

lemma *alw-invar*:

assumes φ *xs* **and** *alw* (φ *impl nxt* φ) *xs*
shows *alw* φ *xs*
proof−
 {assume φ *xs* $\wedge$ *alw* (φ *impl nxt* φ) *xs* **hence** *?thesis*
  by *coinduct auto*
 }
 thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *variance*:
assumes *1*: φ *xs* **and** *2*: *alw* (φ *impl* (ψ *or nxt* φ)) *xs*
shows (*alw* φ *or ev* ψ) *xs*
proof−
 {assume ¬ *ev* ψ *xs* **hence** *alw* (*not* ψ) *xs* **unfolding** *not-ev*[*symmetric*] **.**
  moreover have *alw* (*not* ψ *impl* (φ *impl nxt* φ)) *xs*
  using *2* **by** *coinduct auto*
  ultimately have *alw* (φ *impl nxt* φ) *xs* **by**(*auto dest*: *alw-mp*)
  with *1* **have** *alw* φ *xs* **by**(*rule alw-invar*)
 }
 thus *?thesis* **by** *blast*
qed

lemma *ev-alw-imp-nxt*:
assumes *e*: *ev* φ *xs* **and** *a*: *alw* (φ *impl* (*nxt* φ)) *xs*
shows *ev* (*alw* φ) *xs*
proof−
 obtain *xl xs1* **where** *xs*: *xs* = *xl* @− *xs1* **and** φ: φ *xs1*
 using *e* **by** (*metis ev-imp-shift*)
  have φ *xs1* $\wedge$ *alw* (φ *impl* (*nxt* φ)) *xs1* **using** *a* φ **unfolding** *xs* **by** (*metis alw-shift*)
 hence *alw* φ *xs1* **by**(*coinduct xs1 rule*: *alw.coinduct*) *auto*
 thus *?thesis* **unfolding** *xs* **by** (*auto intro*: *alw-ev-shift*)
qed

inductive *ev-at* :: ('*a stream* ⇒ *bool*) ⇒ *nat* ⇒ '*a stream* ⇒ *bool* **for** *P* :: '*a stream* ⇒ *bool* **where**
 base: *P* ω ⟹ *ev-at P 0* ω
| *step*:¬ *P* ω ⟹ *ev-at P n* (*stl* ω) ⟹ *ev-at P* (*Suc n*) ω

inductive-simps *ev-at-0*[*simp*]: *ev-at P 0* ω
inductive-simps *ev-at-Suc*[*simp*]: *ev-at P* (*Suc n*) ω

lemma *ev-at-imp-snth*: *ev-at P n* ω ⟹ *P* (*sdrop n* ω)
 by (*induction n arbitrary*: ω) *auto*

lemma *ev-at-HLD-imp-snth*: *ev-at* (*HLD X*) *n* ω ⟹ ω !! *n* ∈ *X*
 by (*auto dest*!: *ev-at-imp-snth simp*: *HLD-iff*)

lemma *ev-at-HLD-single-imp-snth*: *ev-at* (*HLD* {*x*}) *n* ω ⟹ ω !! *n* = *x*
by (*drule ev-at-HLD-imp-snth*) *simp*

lemma *ev-at-unique*: *ev-at P n* ω ⟹ *ev-at P m* ω ⟹ *n* = *m*
proof (*induction arbitrary*: *m rule*: *ev-at.induct*)
case (*base* ω) **then show** *?case*
by (*simp add*: *ev-at.simps*[*of* - - ω])
next
case (*step* ω *n*) **from** *step.prems step.hyps step.IH*[*of m − 1*] **show** *?case*
by (*auto simp add*: *ev-at.simps*[*of* - - ω])
qed

lemma *ev-iff-ev-at*: *ev P* ω ⟷ (∃ *n*. *ev-at P n* ω)
proof
assume *ev P* ω **then show** ∃ *n*. *ev-at P n* ω
by (*induction rule*: *ev-induct-strong*) (*auto intro*: *ev-at.intros*)
next
assume ∃ *n*. *ev-at P n* ω
then obtain *n* **where** *ev-at P n* ω
by *auto*
then show *ev P* ω
by *induction auto*
qed

lemma *ev-at-shift*: *ev-at* (*HLD X*) *i* (*stake* (*Suc i*) ω @− ω′ :: *'s stream*) ⟷ *ev-at* (*HLD X*) *i* ω
by (*induction i arbitrary*: ω) (*auto simp*: *HLD-iff*)

lemma *ev-iff-ev-at-unqiue*: *ev P* ω ⟷ (∃!*n*. *ev-at P n* ω)
by (*auto intro*: *ev-at-unique simp*: *ev-iff-ev-at*)

lemma *alw-HLD-iff-streams*: *alw* (*HLD X*) ω ⟷ ω ∈ *streams X*
proof
assume *alw* (*HLD X*) ω **then show** ω ∈ *streams X*
proof (*coinduction arbitrary*: ω)
case (*streams* ω) **then show** *?case* **by** (*cases* ω) *auto*
qed
next
assume ω ∈ *streams X* **then show** *alw* (*HLD X*) ω
proof (*coinduction arbitrary*: ω)
case (*alw* ω) **then show** *?case* **by** (*cases* ω) *auto*
qed
qed

lemma *not-HLD*: *not* (*HLD X*) = *HLD* (− *X*)
by (*auto simp*: *HLD-iff*)

lemma *not-alw-iff*: ¬ (*alw P* ω) ⟷ *ev* (*not P*) ω
using *not-alw*[*of P*] **by** (*simp add*: *fun-eq-iff*)

lemma *not-ev-iff*: $\neg\ (ev\ P\ \omega) \longleftrightarrow alw\ (not\ P)\ \omega$
using *not-alw-iff*[*of not P ω*, *symmetric*] **by** *simp*

lemma *ev-Stream*: $ev\ P\ (x\ \#\#\ s) \longleftrightarrow P\ (x\ \#\#\ s) \vee ev\ P\ s$
by (*auto elim*: *ev.cases*)

lemma *alw-ev-imp-ev-alw*:
assumes $alw\ (ev\ P)\ \omega$ **shows** $ev\ (P\ aand\ alw\ (ev\ P))\ \omega$
proof −
have $ev\ P\ \omega$ **using** *assms* **by** *auto*
from *this assms* **show** *?thesis*
by *induct auto*
qed

lemma *ev-False*: $ev\ (\lambda x.\ False)\ \omega \longleftrightarrow False$
proof
assume $ev\ (\lambda x.\ False)\ \omega$ **then show** *False*
by *induct auto*
qed *auto*

lemma *alw-False*: $alw\ (\lambda x.\ False)\ \omega \longleftrightarrow False$
by *auto*

lemma *ev-iff-sdrop*: $ev\ P\ \omega \longleftrightarrow (\exists\, m.\ P\ (sdrop\ m\ \omega))$
proof *safe*
assume $ev\ P\ \omega$ **then show** $\exists\, m.\ P\ (sdrop\ m\ \omega)$
by (*induct rule*: *ev-induct-strong*) (*auto intro*: *exI*[*of - 0*] *exI*[*of - Suc n* **for** *n*])
next
fix *m* **assume** $P\ (sdrop\ m\ \omega)$ **then show** $ev\ P\ \omega$
by (*induct m arbitrary*: *ω*) *auto*
qed

lemma *alw-iff-sdrop*: $alw\ P\ \omega \longleftrightarrow (\forall\, m.\ P\ (sdrop\ m\ \omega))$
proof *safe*
fix *m* **assume** $alw\ P\ \omega$ **then show** $P\ (sdrop\ m\ \omega)$
by (*induct m arbitrary*: *ω*) *auto*
next
assume $\forall\, m.\ P\ (sdrop\ m\ \omega)$ **then show** $alw\ P\ \omega$
by (*coinduction arbitrary*: *ω*) (*auto elim*: *allE*[*of - 0*] *allE*[*of - Suc n* **for** *n*])
qed

lemma *infinite-iff-alw-ev*: $infinite\ \{m.\ P\ (sdrop\ m\ \omega)\} \longleftrightarrow alw\ (ev\ P)\ \omega$
unfolding *infinite-nat-iff-unbounded-le alw-iff-sdrop ev-iff-sdrop*
by *simp* (*metis le-Suc-ex le-add1*)

lemma *alw-inv*:
assumes *stl*: $\bigwedge s.\ f\ (stl\ s) = stl\ (f\ s)$
shows $alw\ P\ (f\ s) \longleftrightarrow alw\ (\lambda x.\ P\ (f\ x))\ s$

proof
assume *alw P (f s)* **then show** *alw (λx. P (f x)) s*
by (*coinduction arbitrary*: *s rule*: *alw-coinduct*)
(*auto simp*: *stl*)
next
assume *alw (λx. P (f x)) s* **then show** *alw P (f s)*
by (*coinduction arbitrary*: *s rule*: *alw-coinduct*) (*auto simp*: *stl*[*symmetric*])
qed

lemma *ev-inv*:
assumes *stl*: ⋀*s*. *f* (*stl s*) = *stl* (*f s*)
shows *ev P* (*f s*) ⟷ *ev* (*λx*. *P* (*f x*)) *s*
proof
assume *ev P* (*f s*) **then show** *ev* (*λx*. *P* (*f x*)) *s*
by (*induction f s arbitrary*: *s*) (*auto simp*: *stl*)
next
assume *ev* (*λx*. *P* (*f x*)) *s* **then show** *ev P* (*f s*)
by *induction* (*auto simp*: *stl*[*symmetric*])
qed

lemma *alw-smap*: *alw P* (*smap f s*) ⟷ *alw* (*λx*. *P* (*smap f x*)) *s*
by (*rule alw-inv*) *simp*

lemma *ev-smap*: *ev P* (*smap f s*) ⟷ *ev* (*λx*. *P* (*smap f x*)) *s*
by (*rule ev-inv*) *simp*

lemma *alw-cong*:
assumes *P*: *alw P ω* **and** *eq*: ⋀*ω*. *P ω* ⟹ *Q1 ω* ⟷ *Q2 ω*
shows *alw Q1 ω* ⟷ *alw Q2 ω*
proof −
from *eq* **have** (*alw P aand Q1*) = (*alw P aand Q2*) **by** *auto*
then have *alw* (*alw P aand Q1*) *ω* = *alw* (*alw P aand Q2*) *ω* **by** *auto*
with *P* **show** *alw Q1 ω* ⟷ *alw Q2 ω*
by (*simp add*: *alw-aand*)
qed

lemma *ev-cong*:
assumes *P*: *alw P ω* **and** *eq*: ⋀*ω*. *P ω* ⟹ *Q1 ω* ⟷ *Q2 ω*
shows *ev Q1 ω* ⟷ *ev Q2 ω*
proof −
from *P* **have** *alw* (*λxs*. *Q1 xs* ⟶ *Q2 xs*) *ω* **by** (*rule alw-mono*) (*simp add*: *eq*)
moreover from *P* **have** *alw* (*λxs*. *Q2 xs* ⟶ *Q1 xs*) *ω* **by** (*rule alw-mono*)
(*simp add*: *eq*)
moreover note *ev-alw-impl*[*of Q1 ω Q2*] *ev-alw-impl*[*of Q2 ω Q1*]
ultimately show *ev Q1 ω* ⟷ *ev Q2 ω*
by *auto*
qed

lemma *alwD*: *alw P x* ⟹ *P x*

by *auto*

lemma *alw-alwD*: *alw P ω ⟹ alw (alw P) ω*
by *simp*

lemma *alw-ev-stl*: *alw (ev P) (stl ω) ⟷ alw (ev P) ω*
by (*auto intro*: *alw.intros*)

lemma *holds-Stream*: *holds P (x ## s) ⟷ P x*
by *simp*

lemma *holds-eq1*[*simp*]: *holds (op = x) = HLD {x}*
by *rule* (*auto simp*: *HLD-iff*)

lemma *holds-eq2*[*simp*]: *holds (λy. y = x) = HLD {x}*
by *rule* (*auto simp*: *HLD-iff*)

lemma *not-holds-eq*[*simp*]: *holds (− op = x) = not (HLD {x})*
by *rule* (*auto simp*: *HLD-iff*)

Strong until

context
notes [[*inductive-internals*]]
begin

inductive *suntil* (**infix** *suntil 60*) **for** *φ ψ* **where**
base: *ψ ω ⟹ (φ suntil ψ) ω*
| *step*: *φ ω ⟹ (φ suntil ψ) (stl ω) ⟹ (φ suntil ψ) ω*

inductive-simps *suntil-Stream*: *(φ suntil ψ) (x ## s)*

end

lemma *suntil-induct-strong*[*consumes 1*, *case-names base step*]:
(φ suntil ψ) x ⟹
(⋀ω. ψ ω ⟹ P ω) ⟹
(⋀ω. φ ω ⟹ ¬ ψ ω ⟹ (φ suntil ψ) (stl ω) ⟹ P (stl ω) ⟹ P ω) ⟹ P x
using *suntil.induct*[*of φ ψ x P*] **by** *blast*

lemma *ev-suntil*: *(φ suntil ψ) ω ⟹ ev ψ ω*
by (*induct rule*: *suntil.induct*) *auto*

lemma *suntil-inv*:
assumes *stl*: *⋀s. f (stl s) = stl (f s)*
shows *(P suntil Q) (f s) ⟷ ((λx. P (f x)) suntil (λx. Q (f x))) s*
proof
assume *(P suntil Q) (f s)* **then show** *((λx. P (f x)) suntil (λx. Q (f x))) s*
by (*induction f s arbitrary*: *s*) (*auto simp*: *stl intro*: *suntil.intros*)
next

assume $((\lambda x.\ P\ (f\ x))\ suntil\ (\lambda x.\ Q\ (f\ x)))\ s$ **then show** $(P\ suntil\ Q)\ (f\ s)$
by *induction* (*auto simp*: *stl*[*symmetric*] *intro*: *suntil.intros*)
qed

lemma *suntil-smap*: $(P\ suntil\ Q)\ (smap\ f\ s) \longleftrightarrow ((\lambda x.\ P\ (smap\ f\ x))\ suntil\ (\lambda x.\ Q\ (smap\ f\ x)))\ s$
by (*rule suntil-inv*) *simp*

lemma *hld-smap*: $HLD\ x\ (smap\ f\ s) = holds\ (\lambda y.\ f\ y \in x)\ s$
by (*simp add*: *HLD-def*)

lemma *suntil-mono*:
assumes *eq*: $\bigwedge\omega.\ P\ \omega \Longrightarrow Q1\ \omega \Longrightarrow Q2\ \omega\ \bigwedge\omega.\ P\ \omega \Longrightarrow R1\ \omega \Longrightarrow R2\ \omega$
assumes $*$: $(Q1\ suntil\ R1)\ \omega\ alw\ P\ \omega$ **shows** $(Q2\ suntil\ R2)\ \omega$
using $*$ **by** *induct* (*auto intro*: *eq suntil.intros*)

lemma *suntil-cong*:
$alw\ P\ \omega \Longrightarrow (\bigwedge\omega.\ P\ \omega \Longrightarrow Q1\ \omega \longleftrightarrow Q2\ \omega) \Longrightarrow (\bigwedge\omega.\ P\ \omega \Longrightarrow R1\ \omega \longleftrightarrow R2\ \omega) \Longrightarrow$
$(Q1\ suntil\ R1)\ \omega \longleftrightarrow (Q2\ suntil\ R2)\ \omega$
using *suntil-mono*[*of P Q1 Q2 R1 R2 ω*] *suntil-mono*[*of P Q2 Q1 R2 R1 ω*] **by** *auto*

lemma *ev-suntil-iff*: $ev\ (P\ suntil\ Q)\ \omega \longleftrightarrow ev\ Q\ \omega$
proof
assume $ev\ (P\ suntil\ Q)\ \omega$ **then show** $ev\ Q\ \omega$
by *induct* (*auto dest*: *ev-suntil*)
next
assume $ev\ Q\ \omega$ **then show** $ev\ (P\ suntil\ Q)\ \omega$
by *induct* (*auto intro*: *suntil.intros*)
qed

lemma *true-suntil*: $((\lambda\text{-}.\ True)\ suntil\ P) = ev\ P$
by (*simp add*: *suntil-def ev-def*)

lemma *suntil-lfp*: $(\varphi\ suntil\ \psi) = lfp\ (\lambda P\ s.\ \psi\ s \vee (\varphi\ s \wedge P\ (stl\ s)))$
by (*simp add*: *suntil-def*)

lemma *sfilter-P*[*simp*]: $P\ (shd\ s) \Longrightarrow sfilter\ P\ s = shd\ s\ \#\#\ sfilter\ P\ (stl\ s)$
using *sfilter-Stream*[*of P shd s stl s*] **by** *simp*

lemma *sfilter-not-P*[*simp*]: $\neg\ P\ (shd\ s) \Longrightarrow sfilter\ P\ s = sfilter\ P\ (stl\ s)$
using *sfilter-Stream*[*of P shd s stl s*] **by** *simp*

lemma *sfilter-eq*:
assumes $ev\ (holds\ P)\ s$
shows $sfilter\ P\ s = x\ \#\#\ s' \longleftrightarrow$
$P\ x \wedge (not\ (holds\ P)\ suntil\ (HLD\ \{x\}\ aand\ nxt\ (\lambda s.\ sfilter\ P\ s = s')))\ s$
using *assms*

by (*induct rule*: *ev-induct-strong*)
(*auto simp add*: *HLD-iff intro*: *suntil.intros elim*: *suntil.cases*)

lemma *sfilter-streams*:
alw (*ev* (*holds P*)) $\omega \Longrightarrow \omega \in$ *streams A* $\Longrightarrow$ *sfilter P* $\omega \in$ *streams* $\{x{\in}A.\ P\ x\}$
proof (*coinduction arbitrary*: ω)
case (*streams* ω)
then have *ev* (*holds P*) ω **by** *blast*
from *this streams* **show** *?case*
by (*induct rule*: *ev-induct-strong*) (*auto elim*: *streamsE*)
qed

lemma *alw-sfilter*:
assumes $*$: *alw* (*ev* (*holds P*)) *s*
shows *alw Q* (*sfilter P s*) $\longleftrightarrow$ *alw* ($\lambda x.$ *Q* (*sfilter P x*)) *s*
proof
assume *alw Q* (*sfilter P s*) **with** $*$ **show** *alw* ($\lambda x.$ *Q* (*sfilter P x*)) *s*
proof (*coinduction arbitrary*: *s rule*: *alw-coinduct*)
case (*stl s*)
then have *ev* (*holds P*) *s*
by *blast*
from *this stl* **show** *?case*
by (*induct rule*: *ev-induct-strong*) *auto*
qed *auto*
next
assume *alw* ($\lambda x.$ *Q* (*sfilter P x*)) *s* **with** $*$ **show** *alw Q* (*sfilter P s*)
proof (*coinduction arbitrary*: *s rule*: *alw-coinduct*)
case (*stl s*)
then have *ev* (*holds P*) *s*
by *blast*
from *this stl* **show** *?case*
by (*induct rule*: *ev-induct-strong*) *auto*
qed *auto*
qed

lemma *ev-sfilter*:
assumes $*$: *alw* (*ev* (*holds P*)) *s*
shows *ev Q* (*sfilter P s*) $\longleftrightarrow$ *ev* ($\lambda x.$ *Q* (*sfilter P x*)) *s*
proof
assume *ev Q* (*sfilter P s*) **from** *this* $*$ **show** *ev* ($\lambda x.$ *Q* (*sfilter P x*)) *s*
proof (*induction sfilter P s arbitrary*: *s rule*: *ev-induct-strong*)
case (*step s*)
then have *ev* (*holds P*) *s*
by *blast*
from *this step* **show** *?case*
by (*induct rule*: *ev-induct-strong*) *auto*
qed *auto*
next
assume *ev* ($\lambda x.$ *Q* (*sfilter P x*)) *s* **then show** *ev Q* (*sfilter P s*)

```
  proof (induction rule: ev-induct-strong)
    case (step s) then show ?case
      by (cases P (shd s)) auto
  qed auto
qed

lemma holds-sfilter:
  assumes ev (holds Q) s shows holds P (sfilter Q s) ⟷ (not (holds Q) suntil
(holds (Q aand P))) s
proof
  assume holds P (sfilter Q s) with assms show (not (holds Q) suntil (holds (Q
aand P))) s
    by (induct rule: ev-induct-strong) (auto intro: suntil.intros)
next
  assume (not (holds Q) suntil (holds (Q aand P))) s then show holds P (sfilter
Q s)
    by induct auto
qed

lemma suntil-aand-nxt:
  (φ suntil (φ aand nxt ψ)) ω ⟷ (φ aand nxt (φ suntil ψ)) ω
proof
  assume (φ suntil (φ aand nxt ψ)) ω then show (φ aand nxt (φ suntil ψ)) ω
    by induction (auto intro: suntil.intros)
next
  assume (φ aand nxt (φ suntil ψ)) ω
  then have (φ suntil ψ) (stl ω) φ ω
    by auto
  then show (φ suntil (φ aand nxt ψ)) ω
    by (induction stl ω arbitrary: ω)
       (auto elim: suntil.cases intro: suntil.intros)
qed

lemma alw-sconst: alw P (sconst x) ⟷ P (sconst x)
proof
  assume P (sconst x) then show alw P (sconst x)
    by coinduction auto
qed auto

lemma ev-sconst: ev P (sconst x) ⟷ P (sconst x)
proof
  assume ev P (sconst x) then show P (sconst x)
    by (induction sconst x) auto
qed auto

lemma suntil-sconst: (φ suntil ψ) (sconst x) ⟷ ψ (sconst x)
proof
  assume (φ suntil ψ) (sconst x) then show ψ (sconst x)
    by (induction sconst x) auto
```

qed (*auto intro*: *suntil.intros*)

lemma *hld-smap'*: *HLD x* (*smap f s*) = *HLD* (*f* −‘ *x*) *s*
by (*simp add*: *HLD-def*)

end

theory *Stream-Space*
imports
Infinite-Product-Measure
~~/*src*/*HOL*/*Library*/*Stream*
~~/*src*/*HOL*/*Library*/*Linear-Temporal-Logic-on-Streams*
begin

lemma *stream-eq-Stream-iff*: *s* = *x* ## *t* ⟷ (*shd s* = *x* ∧ *stl s* = *t*)
by (*cases s*) *simp*

lemma *Stream-snth*: (*x* ## *s*) !! *n* = (*case n of 0* ⇒ *x* | *Suc n* ⇒ *s* !! *n*)
by (*cases n*) *simp-all*

definition *to-stream* :: (*nat* ⇒ *'a*) ⇒ *'a stream* **where**
to-stream X = *smap X nats*

lemma *to-stream-nat-case*: *to-stream* (*case-nat x X*) = *x* ## *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*: *to-stream X* ∈ *streams S* ⟷ (∀ *n*. *X n* ∈ *S*)
by (*simp add*: *to-stream-def streams-iff-snth*)

definition *stream-space* :: *'a measure* ⇒ *'a stream measure* **where**
stream-space M =
distr (Π$_M$ *i*∈*UNIV*. *M*) (*vimage-algebra* (*streams* (*space M*)) *snth* (Π$_M$ *i*∈*UNIV*. *M*)) *to-stream*

lemma *space-stream-space*: *space* (*stream-space M*) = *streams* (*space M*)
by (*simp add*: *stream-space-def*)

lemma *streams-stream-space*[*intro*]: *streams* (*space M*) ∈ *sets* (*stream-space M*)
using *sets.top*[*of stream-space M*] **by** (*simp add*: *space-stream-space*)

lemma *stream-space-Stream*:
x ## ω ∈ *space* (*stream-space M*) ⟷ *x* ∈ *space M* ∧ ω ∈ *space* (*stream-space M*)
by (*simp add*: *space-stream-space streams-Stream*)

lemma *stream-space-eq-distr*: *stream-space M* = *distr* (Π$_M$ *i*∈*UNIV*. *M*) (*stream-space*

M) *to-stream*
unfolding *stream-space-def* **by** (*rule distr-cong*) *auto*

lemma *sets-stream-space-cong*[*measurable-cong*]:
sets M = *sets N* ⟹ *sets* (*stream-space M*) = *sets* (*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*: (λω *n*. ω !! *n*) ∈ *measurable* (*stream-space M*) (Π$_M$ *i*∈*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*]: (λω. ω !! *n*) ∈ *measurable* (*stream-space M*) *M*
using *measurable-snth-PiM measurable-component-singleton* **by** (*rule measurable-compose*) *simp*

lemma *measurable-shd*[*measurable*]: *shd* ∈ *measurable* (*stream-space M*) *M*
using *measurable-snth*[*of 0*] **by** *simp*

lemma *measurable-stream-space2*:
assumes *f-snth*: ⋀*n*. (λ*x*. *f x* !! *n*) ∈ *measurable N M*
shows *f* ∈ *measurable N* (*stream-space M*)
unfolding *stream-space-def measurable-distr-eq2*
proof (*rule measurable-vimage-algebra2*)
show *f* ∈ *space N* → *streams* (*space M*)
using *f-snth*[*THEN measurable-space*] **by** (*auto simp add*: *streams-iff-sset sset-range*)
show (λ*x*. *op* !! (*f x*)) ∈ *measurable N* (*Pi*$_M$ *UNIV* (λ*i*. *M*))
proof (*rule measurable-PiM-single'*)
show (λ*x*. *op* !! (*f x*)) ∈ *space N* → *UNIV* →$_E$ *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*: ⋀*f*. *F f* ⟹ (λ*x*. *shd* (*f x*)) ∈ *measurable N M*
assumes *t*: ⋀*f*. *F f* ⟹ *F* (λ*x*. *stl* (*f x*))
shows *f* ∈ *measurable N* (*stream-space M*)
proof (*rule measurable-stream-space2*)
fix *n* **show** (λ*x*. *f x* !! *n*) ∈ *measurable N M*
using ‹*F f*› **by** (*induction n arbitrary*: *f*) (*auto intro*: *h t*)
qed

lemma *measurable-sdrop*[*measurable*]: *sdrop n* ∈ *measurable* (*stream-space M*) (*stream-space M*)

by (*rule measurable-stream-space2*) (*simp add*: *sdrop-snth*)

lemma *measurable-stl*[*measurable*]: (λω. *stl* ω) ∈ *measurable* (*stream-space M*) (*stream-space M*)
by (*rule measurable-stream-space2*) (*simp del*: *snth.simps add*: *snth.simps*[*symmetric*])

lemma *measurable-to-stream*[*measurable*]: *to-stream* ∈ *measurable* (Π$_M$ *i*∈*UNIV*. *M*) (*stream-space M*)
by (*rule measurable-stream-space2*) (*simp add*: *to-stream-def*)

lemma *measurable-Stream*[*measurable* (*raw*)]:
assumes *f*[*measurable*]: *f* ∈ *measurable N M*
assumes *g*[*measurable*]: *g* ∈ *measurable N* (*stream-space M*)
shows (λ*x*. *f x* ## *g x*) ∈ *measurable N* (*stream-space M*)
by (*rule measurable-stream-space2*) (*simp add*: *Stream-snth*)

lemma *measurable-smap*[*measurable*]:
assumes *X*[*measurable*]: *X* ∈ *measurable N M*
shows *smap X* ∈ *measurable* (*stream-space N*) (*stream-space M*)
by (*rule measurable-stream-space2*) *simp*

lemma *measurable-stake*[*measurable*]:
stake i ∈ *measurable* (*stream-space* (*count-space UNIV*)) (*count-space* (*UNIV* :: 'a::*countable list set*))
by (*induct i*) *auto*

lemma *measurable-shift*[*measurable*]:
assumes *f*: *f* ∈ *measurable N* (*stream-space M*)
assumes [*measurable*]: *g* ∈ *measurable N* (*stream-space M*)
shows (λ*x*. *stake n* (*f x*) @− *g x*) ∈ *measurable N* (*stream-space M*)
using *f* **by** (*induction n arbitrary*: *f*) *simp-all*

lemma *measurable-ev-at*[*measurable*]:
assumes [*measurable*]: *Measurable.pred* (*stream-space M*) *P*
shows *Measurable.pred* (*stream-space M*) (*ev-at P n*)
by (*induction n*) *auto*

lemma *measurable-alw*[*measurable*]:
Measurable.pred (*stream-space M*) *P* ⟹ *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* ⟹ *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* $\Longrightarrow$ *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* $\Longrightarrow$ *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*:
($\lambda(\omega 1, \omega 2)$. *szip* $\omega 1$ $\omega 2$) $\in$ *measurable* (*stream-space M* $\bigotimes_M$ *stream-space N*) (*stream-space* (*M* $\bigotimes_M$ *N*))
proof (*rule measurable-stream-space2*)
fix *n*
have (λx. (*case x of* ($\omega 1$, $\omega 2$) $\Rightarrow$ *szip* $\omega 1$ $\omega 2$) !! *n*) = ($\lambda(\omega 1, \omega 2)$. ($\omega 1$!! *n*, $\omega 2$!! *n*))
by *auto*
also have ... $\in$ *measurable* (*stream-space M* $\bigotimes_M$ *stream-space N*) (*M* $\bigotimes_M$ *N*)
by *measurable*
finally show (λx. (*case x of* ($\omega 1$, $\omega 2$) $\Rightarrow$ *szip* $\omega 1$ $\omega 2$) !! *n*) $\in$ *measurable* (*stream-space M* $\bigotimes_M$ *stream-space N*) (*M* $\bigotimes_M$ *N*)
.
qed

lemma (**in** *prob-space*) *prob-space-stream-space*: *prob-space* (*stream-space M*)
proof −
interpret *product-prob-space* λ-. *M UNIV* **..**
show *?thesis*

by (*subst stream-space-eq-distr*) (*auto intro*!: *P.prob-space-distr*)
qed

lemma (**in** *prob-space*) *nn-integral-stream-space*:
assumes [*measurable*]: $f \in$ *borel-measurable* (*stream-space M*)
shows $(\int^+ X.\ f\ X\ \partial$*stream-space M*$) = (\int^+ x.\ (\int^+ X.\ f\ (x\ \#\#\ X)\ \partial$*stream-space M*$)\ \partial M)$
proof −
interpret *S*: *sequence-space M* **..**
interpret *P*: *pair-sigma-finite M* Π_M *i*::*nat*∈*UNIV*. *M* **..**

have $(\int^+ X.\ f\ X\ \partial$*stream-space M*$) = (\int^+ X.\ f\ ($*to-stream X*$)\ \partial S.S)$
by (*subst stream-space-eq-distr*) (*simp add*: *nn-integral-distr*)
also have ... $= (\int^+ X.\ f\ ($*to-stream* $((\lambda(s,\ \omega).$ *case-nat s* $\omega)\ X))\ \partial(M \bigotimes_M S.S))$
by (*subst S.PiM-iter*[*symmetric*]) (*simp add*: *nn-integral-distr*)
also have ... $= (\int^+ x.\ \int^+ X.\ f\ ($*to-stream* $((\lambda(s,\ \omega).$ *case-nat s* $\omega)\ (x,\ X)))\ \partial S.S\ \partial M)$
by (*subst S.nn-integral-fst*) *simp-all*
also have ... $= (\int^+ x.\ \int^+ X.\ f\ (x\ \#\#$ *to-stream X*$)\ \partial S.S\ \partial M)$
by (*auto intro*!: *nn-integral-cong simp*: *to-stream-nat-case*)
also have ... $= (\int^+ x.\ \int^+ X.\ f\ (x\ \#\#\ X)\ \partial$*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*[*measurable*]: *X* ∈ *sets* (*stream-space M*)
shows *emeasure* (*stream-space M*) *X* = $(\int^+ t.$ *emeasure* (*stream-space M*) $\{x \in$*space* (*stream-space M*)*. t ## x* ∈ *X* $\}\ \partial M)$
proof −
have *eq*: $\bigwedge x\ xs.$ *xs* ∈ *space* (*stream-space M*) $\Longrightarrow$ *x* ∈ *space M* $\Longrightarrow$
indicator X (*x ## xs*) = *indicator* $\{xs \in$*space* (*stream-space M*)*. x ## xs* ∈ *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*[*measurable*]: $\{x \in$*space* (*stream-space M*)*. P x*$\} \in$ *sets* (*stream-space M*)
shows $\mathcal{P}($*x in stream-space M. P x*$) = (\int^+ t.\ \mathcal{P}($*x in stream-space M. P* (t ##

```
x)) ∂M)
proof −
  interpret S: prob-space stream-space M
    by (rule prob-space-stream-space)
  show ?thesis
    unfolding S.emeasure-eq-measure[symmetric]
   by (subst emeasure-stream-space) (auto simp: stream-space-Stream intro!: nn-integral-cong)
qed

lemma (in prob-space) AE-stream-space:
  assumes [measurable]: Measurable.pred (stream-space M) P
  shows (AE X in stream-space M. P X) = (AE x in M. AE X in stream-space
M. P (x ## X))
proof −
  interpret stream: prob-space stream-space M
    by (rule prob-space-stream-space)

  have eq: ⋀x X. indicator {x. ¬ P x} (x ## X) = indicator {X. ¬ P (x ##
X)} X
    by (auto split: split-indicator)
  show ?thesis
    apply (subst AE-iff-nn-integral, simp)
    apply (subst nn-integral-stream-space, simp)
    apply (subst eq)
    apply (subst nn-integral-0-iff-AE, simp)
    apply (simp add: AE-iff-nn-integral[symmetric])
    done
qed

lemma (in prob-space) AE-stream-all:
  assumes [measurable]: Measurable.pred M P and P: AE x in M. P x
  shows AE x in stream-space M. stream-all P x
proof −
  { fix n have AE x in stream-space M. P (x !! n)
    proof (induct n)
      case 0 with P show ?case
        by (subst AE-stream-space) (auto elim!: 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* (λ*p 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*. (λ*x*. *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-Sup-sigma2 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 (⨆$_\sigma$ *i*∈*UNIV*. *vimage-algebra* (*streams* (*space M*)) (λ*s*. *s* !! *i*) *M*)
by (*auto intro*!: *sets-stream-space-in-sets sets-Sup-in-sets sets-image-in-sets measurable-Sup-sigma1 snth-in measurable-vimage-algebra1 del*: *subsetI*
simp: *space-Sup-sigma 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*)
apply *simp*
apply (*auto intro*: *streams-mono*) []
apply (*simp add*: *image-image space-restrict-space*)
apply (*intro SUP-sigma-cong*)
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 set* ⇒ *'a list* ⇒ *'a stream set* **where**
sstart S [] = *streams S*
| [*simp del*]: *sstart S* (*x* # *xs*) = *op* ## *x* ‘ *sstart S xs*

lemma *in-sstart*[*simp*]: *s* ∈ *sstart S* (*x* # *xs*) ⟷ *shd s* = *x* ∧ *stl s* ∈ *sstart S xs*
by (*cases s*) (*auto simp*: *sstart.simps*(*2*))

lemma *sstart-in-streams*: *xs* ∈ *lists S* ⟹ *sstart S xs* ⊆ *streams S*
by (*induction xs*) (*auto simp*: *sstart.simps*(*2*))

lemma *sstart-eq*: *x* ∈ *streams S* ⟹ *x* ∈ *sstart S xs* = (∀ *i*<*length xs*. *x* !! *i* = *xs* ! *i*)
by (*induction xs arbitrary*: *x*) (*auto simp*: *nth-Cons streams-stl split*: *nat.splits*)

lemma *sstart-sets*: *sstart S xs* ∈ *sets* (*stream-space* (*count-space UNIV*))
proof (*induction xs*)
case (*Cons x xs*)
note *Cons*[*measurable*]
have *sstart S* (*x* # *xs*) =
{*s*∈*space* (*stream-space* (*count-space UNIV*)). *shd s* = *x* ∧ *stl s* ∈ *sstart S xs*}
by (*simp add*: *set-eq-iff space-stream-space*)
also have ... ∈ *sets* (*stream-space* (*count-space UNIV*))
by *measurable*
finally show *?case* **.**
qed (*simp add*: *streams-sets*)

lemma *sigma-sets-singletons*:
assumes *countable S*
shows *sigma-sets S* ((λ*s*. {*s*}) ‘*S*) = *Pow S*
proof *safe*
interpret *sigma-algebra S sigma-sets S* ((λ*s*. {*s*}) ‘*S*)
by (*rule sigma-algebra-sigma-sets*) *auto*
fix *A* **assume** *A* ⊆ *S*
with *assms* **have** (⋃ *a*∈*A*. {*a*}) ∈ *sigma-sets S* ((λ*s*. {*s*}) ‘*S*)
by (*intro countable-UN'*) (*auto dest*: *countable-subset*)
then show *A* ∈ *sigma-sets S* ((λ*s*. {*s*}) ‘*S*)
by *simp*
qed (*auto dest*: *sigma-sets-into-sp*[*rotated*])

lemma *sets-count-space-eq-sigma*:
countable S ⟹ *sets* (*count-space S*) = *sets* (*sigma S* ((λ*s*. {*s*}) ‘*S*))
by (*subst sets-measure-of*) (*auto simp*: *sigma-sets-singletons*)

lemma *sets-stream-space-sstart*:
assumes *S*[*simp*]: *countable S*
shows *sets* (*stream-space* (*count-space S*)) = *sets* (*sigma* (*streams S*) (*sstart*

S'lists S ∪ {{}}))
proof
have [*simp*]: *sstart S ' lists S ⊆ Pow* (*streams S*)
by (*simp add*: *image-subset-iff sstart-in-streams*)

let *?S* = *sigma* (*streams S*) (*sstart S ' lists S ∪ {{}}*)
{ fix *i a* **assume** *a ∈ S*
{ fix *x* **have** (*x !! i = a ∧ x ∈ streams S*) ⟷ (∃ *xs*∈*lists S*. *length xs = i ∧ x ∈ 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* ‹*a ∈ S*›, *auto intro*: *streams-stl in-streams*) **}**
then have (*λx. x !! i*) −' {*a*} ∩ *streams S* = (⋃ *xs*∈{*xs*∈*lists S*. *length xs = i*}. *sstart S* (*xs @* [*a*]))
by (*auto simp add*: *set-eq-iff*)
also have ... ∈ *sets ?S*
using ‹*a*∈*S*› **by** (*intro sets.countable-UN'*) (*auto intro*!: *sigma-sets.Basic image-eqI*)
finally have (*λx. x !! i*) −' {*a*} ∩ *streams S ∈ sets ?S* **. }**
then show *sets* (*stream-space* (*count-space S*)) ⊆ *sets* (*sigma* (*streams S*) (*sstart S'lists S ∪ {{}}*))
by (*intro sets-stream-space-in-sets*) (*auto simp*: *measurable-count-space-eq-countable snth-in*)

have *sigma-sets* (*space* (*stream-space* (*count-space S*))) (*sstart S'lists S ∪ {{}}*) ⊆ *sets* (*stream-space* (*count-space S*))
proof (*safe intro*!: *sets.sigma-sets-subset*)
fix *xs* **assume** *∀ x*∈*set xs*. *x ∈ S*
then have *sstart S xs* = {*x*∈*space* (*stream-space* (*count-space S*)). *∀ i*<*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 ... ∈ *sets* (*stream-space* (*count-space S*))
by *measurable*
finally show *sstart S xs ∈ sets* (*stream-space* (*count-space S*)) **.**
qed
then show *sets* (*sigma* (*streams S*) (*sstart S'lists S ∪ {{}}*)) ⊆ *sets* (*stream-space* (*count-space S*))
by (*simp add*: *space-stream-space*)
qed

lemma *Int-stable-sstart*: *Int-stable* (*sstart S'lists S ∪ {{}}*)
proof −
{ fix *xs ys* **assume** *xs ∈ lists S ys ∈ lists S*
then have *sstart S xs ∩ sstart S ys ∈ sstart S ' lists S ∪ {{}}*
proof (*induction xs ys rule*: *list-induct2'*)

```
    case (4 x xs y ys)
    show ?case
    proof cases
      assume x = y
      then have sstart S (x # xs) ∩ sstart S (y # ys) = op ## x ‘ (sstart S xs
∩ sstart S ys)
        by (auto simp: image-iff intro!: stream.collapse[symmetric])
      also have ... ∈ sstart S ‘ lists S ∪ {{}}
        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[simp]: countable S
 assumes P: prob-space M prob-space N
 assumes ae: AE x in M. x ∈ streams S AE x in N. x ∈ streams S
 assumes sets-M: sets M = sets (stream-space (count-space UNIV))
 assumes sets-N: sets N = sets (stream-space (count-space UNIV))
 assumes *: ⋀xs. xs ≠ [] ⟹ xs ∈ lists S ⟹ emeasure M (sstart S xs) =
emeasure N (sstart S xs)
 shows M = N
proof (rule measure-eqI-restrict-generator[OF Int-stable-sstart])
 have [simp]: sstart S ‘ lists S ⊆ Pow (streams S)
  by (simp add: image-subset-iff sstart-in-streams)

 interpret M: prob-space M by fact

 show sstart S ‘ lists S ∪ {{}} ⊆ Pow (streams S)
  by (auto dest: sstart-in-streams del: in-listsD)

 { fix M :: 'a stream measure assume M: sets M = sets (stream-space (count-space
UNIV))
   have sets (restrict-space M (streams S)) = sigma-sets (streams S) (sstart S ‘
lists S ∪ {{}})
    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 ∪ {{}}
```

$\bigcup\{streams\ S\} = streams\ S\ \bigwedge s.\ s \in \{streams\ S\} \Longrightarrow emeasure\ M\ s \neq \infty$
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*)

end

31 Embed Measure Spaces with a Function

theory *Embed-Measure*
imports *Binary-Product-Measure*
begin

definition *embed-measure* :: *'a measure* ⇒ (*'a* ⇒ *'b*) ⇒ *'b measure* **where**
embed-measure M f = measure-of (*f ‘ space M*) {*f ‘ A* |*A*. *A* ∈ *sets M*}
(λ*A*. *emeasure M* (*f* −‘ *A* ∩ *space M*))

lemma *space-embed-measure*: *space* (*embed-measure M f*) = *f ‘ space M*
unfolding *embed-measure-def*
by (*subst space-measure-of*) (*auto dest*: *sets.sets-into-space*)

lemma *sets-embed-measure'*:
assumes *inj*: *inj-on f* (*space M*)
shows *sets* (*embed-measure M f*) = {*f ‘ A* |*A*. *A* ∈ *sets M*}
unfolding *embed-measure-def*
proof (*intro sigma-algebra.sets-measure-of-eq sigma-algebra-iff2*[*THEN iffD2*] *conjI allI ballI impI*)
fix *s* **assume** *s* ∈ {*f ‘ A* |*A*. *A* ∈ *sets M*}
then obtain *s'* **where** *s'-props*: *s = f ‘ s' s'* ∈ *sets M* **by** *auto*
hence *f ‘ space M* − *s = f ‘* (*space M* − *s'*) **using** *inj*
by (*auto dest*: *inj-onD sets.sets-into-space*)
also have ... ∈ {*f ‘ A* |*A*. *A* ∈ *sets M*} **using** *s'-props* **by** *auto*
finally show *f ‘ space M* − *s* ∈ {*f ‘ A* |*A*. *A* ∈ *sets M*} **.**

next
fix $A :: nat \Rightarrow$ - **assume** *range* $A \subseteq \{f \text{ ` } A \mid A.\ A \in sets\ M\}$
then obtain A' **where** $\bigwedge i.\ A\ i = f \text{ ` } A'\ i\ \bigwedge i.\ A'\ i \in sets\ M$
by (*auto simp*: *subset-eq choice-iff*)
moreover from *this* **have** $(\bigcup x.\ f \text{ ` } A'\ x) = f \text{ ` } (\bigcup x.\ A'\ x)$ **by** *blast*
ultimately show $(\bigcup i.\ A\ i) \in \{f \text{ ` } A \mid A.\ A \in sets\ M\}$
by *simp blast*
qed (*auto dest*: *sets.sets-into-space*)

lemma *the-inv-into-vimage*:
inj-on f X $\Longrightarrow$ $A \subseteq X \Longrightarrow$ *the-inv-into X f* $-$ ` $A \cap (f`X) = f \text{ ` } A$
by (*auto simp*: *the-inv-into-f-f*)

lemma *sets-embed-eq-vimage-algebra*:
assumes *inj-on f* (*space M*)
shows *sets* (*embed-measure M f*) = *sets* (*vimage-algebra* (*f`space M*) (*the-inv-into* (*space M*) *f*) *M*)
by (*auto simp*: *sets-embed-measure'*[*OF assms*] *Pi-iff the-inv-into-f-f assms sets-vimage-algebra2 simple-image*
dest: *sets.sets-into-space*
intro!: *image-cong the-inv-into-vimage*[*symmetric*])

lemma *sets-embed-measure*:
assumes *inj*: *inj f*
shows *sets* (*embed-measure M f*) = $\{f \text{ ` } A \mid A.\ A \in sets\ M\}$
using *assms* **by** (*subst sets-embed-measure'*) (*auto intro*!: *inj-onI dest*: *injD*)

lemma *in-sets-embed-measure*: $A \in sets\ M \Longrightarrow f \text{ ` } A \in sets$ (*embed-measure M f*)
unfolding *embed-measure-def*
by (*intro in-measure-of*) (*auto dest*: *sets.sets-into-space*)

lemma *measurable-embed-measure1*:
assumes *g*: $(\lambda x.\ g\ (f\ x)) \in measurable\ M\ N$
shows $g \in measurable$ (*embed-measure M f*) *N*
unfolding *measurable-def*
proof *safe*
fix A **assume** $A \in sets\ N$
with *g* **have** $(\lambda x.\ g\ (f\ x)) -$ ` $A \cap space\ M \in sets\ M$
by (*rule measurable-sets*)
then have $f \text{ ` } ((\lambda x.\ g\ (f\ x)) -$ ` $A \cap space\ M) \in sets$ (*embed-measure M f*)
by (*rule in-sets-embed-measure*)
also have $f \text{ ` } ((\lambda x.\ g\ (f\ x)) -$ ` $A \cap space\ M) = g -$ ` $A \cap space$ (*embed-measure M f*)
by (*auto simp*: *space-embed-measure*)
finally show $g -$ ` $A \cap space$ (*embed-measure M f*) $\in sets$ (*embed-measure M f*)
.
qed (*insert measurable-space*[*OF assms*], *auto simp*: *space-embed-measure*)

lemma *measurable-embed-measure2'*:

```
  assumes inj-on f (space M)
  shows f ∈ measurable M (embed-measure M f)
proof−
  {
    fix A assume A: A ∈ sets M
    also from this have A = A ∩ space M by auto
    also have ... = f −‘ f ‘ A ∩ space M using A assms
      by (auto dest: inj-onD sets.sets-into-space)
    finally have f −‘ f ‘ A ∩ space M ∈ sets M .
  }
  thus ?thesis using assms unfolding embed-measure-def
    by (intro measurable-measure-of) (auto dest: sets.sets-into-space)
qed

lemma measurable-embed-measure2:
  assumes [simp]: inj f shows f ∈ measurable M (embed-measure M f)
  by (auto simp: inj-vimage-image-eq embed-measure-def
          intro!: measurable-measure-of dest: sets.sets-into-space)

lemma embed-measure-eq-distr′:
  assumes inj-on f (space M)
  shows embed-measure M f = distr M (embed-measure M f) f
proof−
  have distr M (embed-measure M f) f =
          measure-of (f ‘ space M) {f ‘ A |A. A ∈ sets M}
                     (λA. emeasure M (f −‘ A ∩ space M)) unfolding distr-def
    by (simp add: space-embed-measure sets-embed-measure′[OF assms])
  also have ... = embed-measure M f unfolding embed-measure-def ..
  finally show ?thesis ..
qed

lemma embed-measure-eq-distr:
    inj f ⟹ embed-measure M f = distr M (embed-measure M f) f
  by (rule embed-measure-eq-distr′) (auto intro!: inj-onI dest: injD)

lemma nn-integral-embed-measure′:
  inj-on f (space M) ⟹ g ∈ borel-measurable (embed-measure M f) ⟹
  nn-integral (embed-measure M f) g = nn-integral M (λx. g (f x))
  apply (subst embed-measure-eq-distr′, simp)
  apply (subst nn-integral-distr)
  apply (simp-all add: measurable-embed-measure2′)
  done

lemma nn-integral-embed-measure:
  inj f ⟹ g ∈ borel-measurable (embed-measure M f) ⟹
  nn-integral (embed-measure M f) g = nn-integral M (λx. g (f x))
  by(erule nn-integral-embed-measure′[OF subset-inj-on]) simp

lemma emeasure-embed-measure′:
```

assumes *inj-on f* (*space M*) *A* ∈ *sets* (*embed-measure M f*)
shows *emeasure* (*embed-measure M f*) *A* = *emeasure M* (*f* −‘ *A* ∩ *space M*)
by (*subst embed-measure-eq-distr'*[*OF assms*(*1*)])
(*simp add*: *emeasure-distr*[*OF measurable-embed-measure2'*[*OF assms*(*1*)] *assms*(*2*)])

lemma *emeasure-embed-measure*:
assumes *inj f A* ∈ *sets* (*embed-measure M f*)
shows *emeasure* (*embed-measure M f*) *A* = *emeasure M* (*f* −‘ *A* ∩ *space M*)
using *assms* **by** (*intro emeasure-embed-measure'*) (*auto intro*!: *inj-onI dest*: *injD*)

lemma *embed-measure-comp*:
assumes [*simp*]: *inj f inj g*
shows *embed-measure* (*embed-measure M f*) *g* = *embed-measure M* (*g* ∘ *f*)
proof−
have [*simp*]: *inj* (λ*x*. *g* (*f x*)) **by** (*subst o-def*[*symmetric*]) (*auto intro*: *inj-comp*)
note *measurable-embed-measure2*[*measurable*]
have *embed-measure* (*embed-measure M f*) *g* =
distr M (*embed-measure* (*embed-measure M f*) *g*) (*g* ∘ *f*)
by (*subst* (*1 2*) *embed-measure-eq-distr*)
(*simp-all add*: *distr-distr sets-embed-measure cong*: *distr-cong*)
also have ... = *embed-measure M* (*g* ∘ *f*)
by (*subst* (*3*) *embed-measure-eq-distr*, *simp add*: *o-def*, *rule distr-cong*)
(*auto simp*: *sets-embed-measure o-def image-image*[*symmetric*]
intro: *inj-comp cong*: *distr-cong*)
finally show *?thesis* **.**
qed

lemma *sigma-finite-embed-measure*:
assumes *sigma-finite-measure M* **and** *inj*: *inj f*
shows *sigma-finite-measure* (*embed-measure M f*)
proof −
from *assms*(*1*) **interpret** *sigma-finite-measure M* **.**
from *sigma-finite-countable* **obtain** *A* **where**
A-props: *countable A A* ⊆ *sets M* ⋃ *A* = *space M* ⋀*X*. *X*∈*A* ⟹ *emeasure M X* ≠ ∞ **by** *blast*
from *A-props* **have** *countable* (*op* ‘ *f*‘*A*) **by** *auto*
moreover
from *inj* **and** *A-props* **have** *op* ‘ *f*‘*A* ⊆ *sets* (*embed-measure M f*)
by (*auto simp*: *sets-embed-measure*)
moreover
from *A-props* **and** *inj* **have** ⋃(*op* ‘ *f*‘*A*) = *space* (*embed-measure M f*)
by (*auto simp*: *space-embed-measure intro*!: *imageI*)
moreover
from *A-props* **and** *inj* **have** ∀ *a*∈*op* ‘ *f* ‘ *A*. *emeasure* (*embed-measure M f*) *a* ≠ ∞
by (*intro ballI*, *subst emeasure-embed-measure*)
(*auto simp*: *inj-vimage-image-eq intro*: *in-sets-embed-measure*)
ultimately show *?thesis* **by** − (*standard*, *blast*)
qed

lemma *embed-measure-count-space'*:
inj-on f A ⟹ embed-measure (count-space A) f = count-space (f‘A)
apply (*subst distr-bij-count-space*[*of f A f‘A*, *symmetric*])
apply (*simp add*: *inj-on-def bij-betw-def*)
apply (*subst embed-measure-eq-distr'*)
apply *simp*
apply(*auto 4 3 intro*!: *measure-eqI imageI simp add*: *sets-embed-measure' subset-image-iff*)
apply (*subst* (*1 2*) *emeasure-distr*)
apply (*auto simp*: *space-embed-measure sets-embed-measure'*)
done

lemma *embed-measure-count-space*:
inj f ⟹ embed-measure (count-space A) f = count-space (f‘A)
by(*rule embed-measure-count-space'*)(*erule subset-inj-on*, *simp*)

lemma *sets-embed-measure-alt*:
inj f ⟹ sets (embed-measure M f) = (op‘f) ‘ sets M
by (*auto simp*: *sets-embed-measure*)

lemma *emeasure-embed-measure-image'*:
assumes *inj-on f* (*space M*) *X* ∈ *sets M*
shows *emeasure* (*embed-measure M f*) (*f‘X*) = *emeasure M X*
proof−
from *assms* **have** *emeasure* (*embed-measure M f*) (*f‘X*) = *emeasure M* (*f* −‘ *f*
‘ *X* ∩ *space M*)
by (*subst emeasure-embed-measure'*) (*auto simp*: *sets-embed-measure'*)
also from *assms* **have** *f* −‘ *f* ‘ *X* ∩ *space M* = *X* **by** (*auto dest*: *inj-onD*
sets.sets-into-space)
finally show *?thesis* **.**
qed

lemma *emeasure-embed-measure-image*:
inj f ⟹ X ∈ *sets M ⟹ emeasure* (*embed-measure M f*) (*f‘X*) = *emeasure*
M X
by (*simp-all add*: *emeasure-embed-measure in-sets-embed-measure inj-vimage-image-eq*)

lemma *embed-measure-eq-iff*:
assumes *inj f*
shows *embed-measure A f = embed-measure B f* ⟷ *A = B* (**is** *?M = ?N* ⟷
-)
proof
from *assms* **have** *I*: *inj* (*op‘ f*) **by** (*auto intro*: *injI dest*: *injD*)
assume *asm*: *?M = ?N*
hence *sets* (*embed-measure A f*) = *sets* (*embed-measure B f*) **by** *simp*
with *assms* **have** *sets A = sets B* **by** (*simp only*: *I inj-image-eq-iff sets-embed-measure-alt*)
moreover {
fix *X* **assume** *X* ∈ *sets A*
from *asm* **have** *emeasure ?M* (*f‘X*) = *emeasure ?N* (*f‘X*) **by** *simp*

```
    with ‹X ∈ sets A› and ‹sets A = sets B› and assms
     have emeasure A X = emeasure B X by (simp add: emeasure-embed-measure-image)
  }
  ultimately show A = B by (rule measure-eqI)
qed simp

lemma the-inv-into-in-Pi: inj-on f A ⟹ the-inv-into A f ∈ f ‘ A → A
  by (auto simp: the-inv-into-f-f)

lemma map-prod-image: map-prod f g ‘ (A × B) = (f‘A) × (g‘B)
  using map-prod-surj-on[OF refl refl] .

lemma map-prod-vimage: map-prod f g −‘ (A × B) = (f−‘A) × (g−‘B)
  by auto

lemma embed-measure-prod:
 assumes f: inj f and g: inj g and [simp]: sigma-finite-measure M sigma-finite-measure
N
  shows embed-measure M f ⨂M embed-measure N g = embed-measure (M ⨂M
N) (λ(x, y). (f x, g y))
    (is ?L = -)
  unfolding map-prod-def[symmetric]
proof (rule pair-measure-eqI)
  have fg[simp]: ⋀A. inj-on (map-prod f g) A ⋀A. inj-on f A ⋀A. inj-on g A
    using f g by (auto simp: inj-on-def)

  show sets: sets ?L = sets (embed-measure (M ⨂M N) (map-prod f g))
    unfolding map-prod-def[symmetric]
    apply (simp add: sets-pair-eq-sets-fst-snd sets-embed-eq-vimage-algebra
      cong: vimage-algebra-cong)
    apply (subst vimage-algebra-Sup-sigma)
    apply (simp-all add: space-pair-measure[symmetric])
    apply (auto simp add: the-inv-into-f-f
                simp del: map-prod-simp
                del: prod-fun-imageE) []
    apply (subst (1 2 3 4 ) vimage-algebra-vimage-algebra-eq)
   apply (simp-all add: the-inv-into-in-Pi Pi-iff[of snd] Pi-iff[of fst] space-pair-measure)
   apply (simp-all add: Pi-iff[of snd] Pi-iff[of fst] the-inv-into-in-Pi vimage-algebra-vimage-algebra-eq
       space-pair-measure[symmetric] map-prod-image[symmetric])
   apply (intro arg-cong[where f=sets] arg-cong[where f=Sup-sigma] arg-cong2[where
f=insert] vimage-algebra-cong)
    apply (auto simp: map-prod-image the-inv-into-f-f
                simp del: map-prod-simp del: prod-fun-imageE)
    apply (simp-all add: the-inv-into-f-f space-pair-measure)
    done

  note measurable-embed-measure2[measurable]
  fix A B assume AB: A ∈ sets (embed-measure M f) B ∈ sets (embed-measure
N g)
```

moreover have $f -` A \times g -` B \cap space\ (M \bigotimes_M N) = (f -` A \cap space\ M) \times (g -` B \cap space\ N)$
by (*auto simp*: *space-pair-measure*)
ultimately show *emeasure* (*embed-measure M f*) $A * emeasure$ (*embed-measure N g*) $B =$
emeasure (*embed-measure* $(M \bigotimes_M N)$ (*map-prod f g*)) $(A \times B)$
by (*simp add*: *map-prod-vimage sets*[*symmetric*] *emeasure-embed-measure sigma-finite-measure.emeasure-pair-measure-Times*)
qed (*insert assms*, *simp-all add*: *sigma-finite-embed-measure*)

lemma *density-embed-measure*:
assumes *inj*: *inj f* **and** *Mg*[*measurable*]: $g \in$ *borel-measurable* (*embed-measure M f*)
shows *density* (*embed-measure M f*) *g* = *embed-measure* (*density M* $(g \circ f)$) *f* (**is** *?M1 = ?M2*)
proof (*rule measure-eqI*)
fix *X* **assume** *X*: $X \in$ *sets ?M1*
from *inj* **have** *Mf*[*measurable*]: $f \in$ *measurable M* (*embed-measure M f*)
by (*rule measurable-embed-measure2*)
from *Mg* **and** *X* **have** *emeasure ?M1 X* $= \int^+ x.\ g\ x * indicator\ X\ x\ \partial$*embed-measure M f*
by (*subst emeasure-density*) *simp-all*
also from *X* **have** ... $= \int^+ x.\ g\ (f\ x) * indicator\ X\ (f\ x)\ \partial M$
by (*subst embed-measure-eq-distr*[*OF inj*], *subst nn-integral-distr*) *auto*
also have ... $= \int^+ x.\ g\ (f\ x) * indicator\ (f -` X \cap space\ M)\ x\ \partial M$
by (*intro nn-integral-cong*) (*auto split*: *split-indicator*)
also from *X* **have** ... = *emeasure* (*density M* $(g \circ f)$) $(f -` X \cap space\ M)$
by (*subst emeasure-density*) (*simp-all add*: *measurable-comp*[*OF Mf Mg*] *measurable-sets*[*OF Mf*])
also from *X* **and** *inj* **have** ... = *emeasure ?M2 X*
by (*subst emeasure-embed-measure*) (*simp-all add*: *sets-embed-measure*)
finally show *emeasure ?M1 X = emeasure ?M2 X* **.**
qed (*simp-all add*: *sets-embed-measure inj*)

lemma *density-embed-measure'*:
assumes *inj*: *inj f* **and** *inv*: $\bigwedge x.\ f'\ (f\ x) = x$ **and** *Mg*[*measurable*]: $g \in$ *borel-measurable M*
shows *density* (*embed-measure M f*) $(g \circ f')$ = *embed-measure* (*density M g*) *f*
proof−
have *density* (*embed-measure M f*) $(g \circ f')$ = *embed-measure* (*density M* $(g \circ f' \circ f)$) *f*
by (*rule density-embed-measure*[*OF inj*])
(*rule measurable-comp*, *rule measurable-embed-measure1*, *subst measurable-cong*, *rule inv*, *rule measurable-ident-sets*, *simp*, *rule Mg*)
also have *density M* $(g \circ f' \circ f)$ = *density M g*
by (*intro density-cong*) (*subst measurable-cong*, *simp add*: *o-def inv*, *simp-all add*: *Mg inv*)
finally show *?thesis* **.**
qed

lemma *inj-on-image-subset-iff*:
assumes *inj-on f C A* ⊆ *C B* ⊆ *C*
shows *f* ‘ *A* ⊆ *f* ‘ *B* ⟷ *A* ⊆ *B*
proof (*intro iffI subsetI*)
fix *x* **assume** *A*: *f* ‘ *A* ⊆ *f* ‘ *B* **and** *B*: *x* ∈ *A*
from *B* **have** *f x* ∈ *f* ‘ *A* **by** *blast*
with *A* **have** *f x* ∈ *f* ‘ *B* **by** *blast*
then obtain *y* **where** *f x* = *f y* **and** *y* ∈ *B* **by** *blast*
with *assms* **and** *B* **have** *x* = *y* **by** (*auto dest*: *inj-onD*)
with ‹*y* ∈ *B*› **show** *x* ∈ *B* **by** *simp*
qed *auto*

lemma *AE-embed-measure′*:
assumes *inj*: *inj-on f* (*space M*)
shows (*AE x in embed-measure M f*. *P x*) ⟷ (*AE x in M*. *P* (*f x*))
proof
let *?M* = *embed-measure M f*
assume *AE x in ?M*. *P x*
then obtain *A* **where** *A-props*: *A* ∈ *sets ?M emeasure ?M A* = *0* {*x*∈*space ?M*. ¬*P x*} ⊆ *A*
by (*force elim*: *AE-E*)
then obtain *A′* **where** *A′-props*: *A* = *f* ‘ *A′ A′* ∈ *sets M* **by** (*auto simp*: *sets-embed-measure′ inj*)
moreover have *B*: {*x*∈*space ?M*. ¬*P x*} = *f* ‘ {*x*∈*space M*. ¬*P* (*f x*)}
by (*auto simp*: *inj space-embed-measure*)
from *A-props*(*3*) **have** {*x*∈*space M*. ¬*P* (*f x*)} ⊆ *A′*
by (*subst* (*asm*) *B*, *subst* (*asm*) *A′-props*, *subst* (*asm*) *inj-on-image-subset-iff*[*OF inj*])
(*insert A′-props*, *auto dest*: *sets.sets-into-space*)
moreover from *A-props A′-props* **have** *emeasure M A′* = *0*
by (*simp add*: *emeasure-embed-measure-image′ inj*)
ultimately show *AE x in M*. *P* (*f x*) **by** (*intro AE-I*)
next
let *?M* = *embed-measure M f*
assume *AE x in M*. *P* (*f x*)
then obtain *A* **where** *A-props*: *A* ∈ *sets M emeasure M A* = *0* {*x*∈*space M*. ¬*P* (*f x*)} ⊆ *A*
by (*force elim*: *AE-E*)
hence *f*‘*A* ∈ *sets ?M emeasure ?M* (*f*‘*A*) = *0* {*x*∈*space ?M*. ¬*P x*} ⊆ *f*‘*A*
by (*auto simp*: *space-embed-measure emeasure-embed-measure-image′ sets-embed-measure′ inj*)
thus *AE x in ?M*. *P x* **by** (*intro AE-I*)
qed

lemma *AE-embed-measure*:
assumes *inj*: *inj f*
shows (*AE x in embed-measure M f*. *P x*) ⟷ (*AE x in M*. *P* (*f x*))

using *assms* **by** (*intro AE-embed-measure'*) (*auto intro*!: *inj-onI dest*: *injD*)

lemma *nn-integral-monotone-convergence-SUP-countable*:
fixes $f :: 'a \Rightarrow 'b \Rightarrow ennreal$
assumes *nonempty*: $Y \neq \{\}$
and *chain*: *Complete-Partial-Order.chain op* $\leq$ $(f \mathbin{`} Y)$
and *countable*: *countable B*
shows $(\int^+ x.\ (SUP\ i{:}Y.\ f\ i\ x)\ \partial count\text{-}space\ B) = (SUP\ i{:}Y.\ (\int^+ x.\ f\ i\ x\ \partial count\text{-}space\ B))$
(**is** *?lhs* = *?rhs*)
proof −
let *?f* = $(\lambda i\ x.\ f\ i\ (from\text{-}nat\text{-}into\ B\ x) * indicator\ (to\text{-}nat\text{-}on\ B \mathbin{`} B)\ x)$
have *?lhs* = $\int^+ x.\ (SUP\ i{:}Y.\ f\ i\ (from\text{-}nat\text{-}into\ B\ (to\text{-}nat\text{-}on\ B\ x)))\ \partial count\text{-}space\ B$
by(*rule nn-integral-cong*)(*simp add*: *countable*)
also have ... = $\int^+ x.\ (SUP\ i{:}Y.\ f\ i\ (from\text{-}nat\text{-}into\ B\ x))\ \partial count\text{-}space\ (to\text{-}nat\text{-}on\ B \mathbin{`} B)$
by(*simp add*: *embed-measure-count-space'*[*symmetric*] *inj-on-to-nat-on countable nn-integral-embed-measure' measurable-embed-measure1*)
also have ... = $\int^+ x.\ (SUP\ i{:}Y.\ ?f\ i\ x)\ \partial count\text{-}space\ UNIV$
by(*simp add*: *nn-integral-count-space-indicator ennreal-indicator*[*symmetric*] *SUP-mult-right-ennreal nonempty*)
also have ... = $(SUP\ i{:}Y.\ \int^+ x.\ ?f\ i\ x\ \partial count\text{-}space\ UNIV)$
proof(*rule nn-integral-monotone-convergence-SUP-nat*)
show *Complete-Partial-Order.chain op* $\leq$ $(?f \mathbin{`} Y)$
by(*rule chain-imageI*[*OF chain*, *unfolded image-image*])(*auto intro*!: *le-funI split*: *split-indicator dest*: *le-funD*)
qed *fact*
also have ... = $(SUP\ i{:}Y.\ \int^+ x.\ f\ i\ (from\text{-}nat\text{-}into\ B\ x)\ \partial count\text{-}space\ (to\text{-}nat\text{-}on\ B \mathbin{`} B))$
by(*simp add*: *nn-integral-count-space-indicator*)
also have ... = $(SUP\ i{:}Y.\ \int^+ x.\ f\ i\ (from\text{-}nat\text{-}into\ B\ (to\text{-}nat\text{-}on\ B\ x))\ \partial count\text{-}space\ B)$
by(*simp add*: *embed-measure-count-space'*[*symmetric*] *inj-on-to-nat-on countable nn-integral-embed-measure' measurable-embed-measure1*)
also have ... = *?rhs*
by(*intro arg-cong2*[**where** *f*=*SUPREMUM*] *ext nn-integral-cong-AE*)(*simp-all add*: *AE-count-space countable*)
finally show *?thesis* .
qed

end

32 Non-denumerability of the Continuum.

theory *ContNotDenum*
imports *Complex-Main Countable-Set*
begin

32.1 Abstract

The following document presents a proof that the Continuum is uncountable. It is formalised in the Isabelle/Isar theorem proving system.

Theorem: The Continuum $\mathbb{R}$ is not denumerable. In other words, there does not exist a function f: $\mathbb{N} \Rightarrow \mathbb{R}$ such that f is surjective.

Outline: An elegant informal proof of this result uses Cantor's Diagonalisation argument. The proof presented here is not this one. First we formalise some properties of closed intervals, then we prove the Nested Interval Property. This property relies on the completeness of the Real numbers and is the foundation for our argument. Informally it states that an intersection of countable closed intervals (where each successive interval is a subset of the last) is non-empty. We then assume a surjective function f: $\mathbb{N} \Rightarrow \mathbb{R}$ exists and find a real x such that x is not in the range of f by generating a sequence of closed intervals then using the NIP.

theorem *real-non-denum*: $\neg$ ($\exists f$ :: *nat* $\Rightarrow$ *real*. *surj f*)
proof
assume $\exists f$::*nat* $\Rightarrow$ *real*. *surj f*
then obtain *f* :: *nat* $\Rightarrow$ *real* **where** *surj f* **..**

First we construct a sequence of nested intervals, ignoring *range f*.

have $\forall$ *a b c*::*real*. *a* < *b* $\longrightarrow$ ($\exists$ *ka kb*. *ka* < *kb* $\wedge$ {*ka*..*kb*} $\subseteq$ {*a*..*b*} $\wedge$ *c* $\notin$ {*ka*..*kb*})
using *assms*
by (*auto simp add*: *not-le cong*: *conj-cong*)
(*metis dense le-less-linear less-linear less-trans order-refl*)
then obtain *i j* **where** *ij*:
$\bigwedge$*a b c*::*real*. *a* < *b* $\Longrightarrow$ *i a b c* < *j a b c*
$\bigwedge$*a b c*. *a* < *b* $\Longrightarrow$ {*i a b c* .. *j a b c*} $\subseteq$ {*a* .. *b*}
$\bigwedge$*a b c*. *a* < *b* $\Longrightarrow$ *c* $\notin$ {*i a b c* .. *j a b c*}
by *metis*

def *ivl* $\equiv$ *rec-nat* (*f 0 + 1*, *f 0 + 2*) (λ*n x*. (*i* (*fst x*) (*snd x*) (*f n*), *j* (*fst x*) (*snd x*) (*f n*)))
def *I* $\equiv$ λ*n*. {*fst* (*ivl n*) .. *snd* (*ivl n*)}

have *ivl*[*simp*]:
ivl 0 = (*f 0 + 1*, *f 0 + 2*)
$\bigwedge$*n*. *ivl* (*Suc n*) = (*i* (*fst* (*ivl n*)) (*snd* (*ivl n*)) (*f n*), *j* (*fst* (*ivl n*)) (*snd* (*ivl n*)) (*f n*))
unfolding *ivl-def* **by** *simp-all*

This is a decreasing sequence of non-empty intervals.

{ fix *n* **have** *fst* (*ivl n*) < *snd* (*ivl n*)
by (*induct n*) (*auto intro*!: *ij*) **}**
note *less* = *this*

```
  have decseq I
    unfolding I-def decseq-Suc-iff ivl fst-conv snd-conv by (intro ij allI less)
```

Now we apply the finite intersection property of compact sets.

```
  have I 0 ∩ (⋂ i. I i) ≠ {}
  proof (rule compact-imp-fip-image)
    fix S :: nat set assume fin: finite S
    have {} ⊂ I (Max (insert 0 S))
      unfolding I-def using less[of Max (insert 0 S)] by auto
    also have I (Max (insert 0 S)) ⊆ (⋂ i∈insert 0 S. I i)
        using fin decseqD[OF ‹decseq I›, of - Max (insert 0 S)] by (auto simp:
Max-ge-iff)
    also have (⋂ i∈insert 0 S. I i) = I 0 ∩ (⋂ i∈S. I i)
      by auto
    finally show I 0 ∩ (⋂ i∈S. I i) ≠ {}
      by auto
  qed (auto simp: I-def)
  then obtain x where ⋀n. x ∈ I n
    by blast
  moreover from ‹surj f› obtain j where x = f j
    by blast
  ultimately have f j ∈ I (Suc j)
    by blast
  with ij(3)[OF less] show False
    unfolding I-def ivl fst-conv snd-conv by auto
qed

lemma uncountable-UNIV-real: uncountable (UNIV::real set)
  using real-non-denum unfolding uncountable-def by auto

lemma bij-betw-open-intervals:
  fixes a b c d :: real
  assumes a < b c < d
  shows ∃f. bij-betw f {a<..<b} {c<..<d}
proof −
  def f ≡ λa b c d x::real. (d − c)/(b − a) * (x − a) + c
  { fix a b c d x :: real assume *: a < b c < d a < x x < b
    moreover from * have (d − c) * (x − a) < (d − c) * (b − a)
      by (intro mult-strict-left-mono) simp-all
    moreover from * have 0 < (d − c) * (x − a) / (b − a)
      by simp
    ultimately have f a b c d x < d c < f a b c d x
      by (simp-all add: f-def field-simps) }
  with assms have bij-betw (f a b c d) {a<..<b} {c<..<d}
    by (intro bij-betw-byWitness[where f'=f c d a b]) (auto simp: f-def)
  thus ?thesis by auto
qed

lemma bij-betw-tan: bij-betw tan {−pi/2<..<pi/2} UNIV
```

using *arctan-ubound* **by** (*intro bij-betw-byWitness*[**where** *f'*=*arctan*]) (*auto simp*: *arctan arctan-tan*)

lemma *uncountable-open-interval*:
fixes *a b* :: *real*
shows *uncountable* {*a*<..<*b*} ⟷ *a* < *b*
proof
assume *uncountable* {*a*<..<*b*}
then show *a* < *b*
using *uncountable-def* **by** *force*
next
assume *a* < *b*
show *uncountable* {*a*<..<*b*}
proof −
obtain *f* **where** *bij-betw f* {*a* <..< *b*} {−*pi*/2<..<*pi*/2}
using *bij-betw-open-intervals*[*OF* ‹*a* < *b*›, *of* −*pi*/2 *pi*/2] **by** *auto*
then show *?thesis*
by (*metis bij-betw-tan uncountable-bij-betw uncountable-UNIV-real*)
qed
qed

lemma *uncountable-half-open-interval-1*:
fixes *a* :: *real* **shows** *uncountable* {*a*..<*b*} ⟷ *a*<*b*
apply *auto*
using *atLeastLessThan-empty-iff* **apply** *fastforce*
using *uncountable-open-interval* [*of a b*]
by (*metis countable-Un-iff ivl-disj-un-singleton*(*3*))

lemma *uncountable-half-open-interval-2*:
fixes *a* :: *real* **shows** *uncountable* {*a*<..*b*} ⟷ *a*<*b*
apply *auto*
using *atLeastLessThan-empty-iff* **apply** *fastforce*
using *uncountable-open-interval* [*of a b*]
by (*metis countable-Un-iff ivl-disj-un-singleton*(*4*))

lemma *real-interval-avoid-countable-set*:
fixes *a b* :: *real* **and** *A* :: *real set*
assumes *a* < *b* **and** *countable A*
shows ∃ *x*∈{*a*<..<*b*}. *x* ∉ *A*
proof −
from ‹*countable A*› **have** *countable* (*A* ∩ {*a*<..<*b*}) **by** *auto*
moreover with ‹*a* < *b*› **have** ¬ *countable* {*a*<..<*b*}
by (*simp add*: *uncountable-open-interval*)
ultimately have *A* ∩ {*a*<..<*b*} ≠ {*a*<..<*b*} **by** *auto*
hence *A* ∩ {*a*<..<*b*} ⊂ {*a*<..<*b*}
by (*intro psubsetI*, *auto*)
hence ∃ *x*. *x* ∈ {*a*<..<*b*} − *A* ∩ {*a*<..<*b*}
by (*rule psubset-imp-ex-mem*)
thus *?thesis* **by** *auto*

qed

lemma *open-minus-countable*:
fixes S A :: *real set* **assumes** *countable* A $S \neq \{\}$ *open* S
shows $\exists x{\in}S.\ x \notin A$
proof −
obtain x **where** $x \in S$
using ‹$S \neq \{\}$› **by** *auto*
then obtain e **where** $0 < e$ $\{y.\ dist\ y\ x < e\} \subseteq S$
using ‹*open S*› **by** (*auto simp*: *open-dist subset-eq*)
moreover have $\{y.\ dist\ y\ x < e\} = \{x - e <..< x + e\}$
by (*auto simp*: *dist-real-def*)
ultimately have *uncountable* $(S - A)$
using *uncountable-open-interval*[*of* $x - e$ $x + e$] ‹*countable A*›
by (*intro uncountable-minus-countable*) (*auto dest*: *countable-subset*)
then show *?thesis*
unfolding *uncountable-def* **by** *auto*
qed

end

33 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* ~~/*src*/*HOL*/*Library*/*ContNotDenum*
begin

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

33.1 Properties of cdf's

definition
cdf :: *real measure* ⇒ *real* ⇒ *real*
where
cdf $M \equiv \lambda x.\ measure\ M\ \{..x\}$

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-super-borel*: *sets borel* ⊆ *sets M*
begin

lemma *sets-M*[*intro*]: *a* ∈ *sets borel* ⟹ *a* ∈ *sets M*
  using *M-super-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*} ∪ {*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* ≤ *y* ⟹ *cdf M x* ≤ *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-super-borel*)

lemma *cdf-nonneg*: *cdf M x* ≥ *0*
  unfolding *cdf-def* **by** (*rule measure-nonneg*)

lemma *cdf-bounded*: *cdf M x* ≤ *measure M* (*space M*)
  unfolding *cdf-def* **using** *assms* **by** (*intro bounded-measure*)

lemma *cdf-lim-infty*:
  ((λ*i*. *cdf M* (*real i*)) ⟶ *measure M* (*space M*))
proof −
  have (λ*i*. *cdf M* (*real i*)) ⟶ *measure M* (⋃ *i*::*nat*. {..*real i*})
    unfolding *cdf-def* **by** (*rule finite-Lim-measure-incseq*) (*auto simp*: *incseq-def*)
  also have (⋃ *i*::*nat*. {..*real i*}) = *space M*
    by (*auto simp*: *borel-UNIV intro*: *real-arch-simple*)
  finally show *?thesis* **.**
qed

lemma *cdf-lim-at-top*: (*cdf M* ⟶ *measure M* (*space M*)) *at-top*
  by (*rule tendsto-at-topI-sequentially-real*)
    (*simp-all add*: *mono-def cdf-nondecreasing cdf-lim-infty*)

lemma *cdf-lim-neg-infty*: ((λ*i*. *cdf M* (− *real i*)) ⟶ *0*)
proof −
  have (λ*i*. *cdf M* (− *real i*)) ⟶ *measure M* (⋂ *i*::*nat*. {.. − *real i* })
    unfolding *cdf-def* **by** (*rule finite-Lim-measure-decseq*) (*auto simp*: *decseq-def*)

```
  also have (⋂ i::nat. {..− real i}) = {}
    by auto (metis leD le-minus-iff reals-Archimedean2)
  finally show ?thesis
    by simp
qed

lemma cdf-lim-at-bot: (cdf M ⟶ 0) at-bot
proof −
  have ∗: ((λx :: real. − cdf M (− x)) ⟶ 0) at-top
    by (intro tendsto-at-topI-sequentially-real monoI)
     (auto simp: cdf-nondecreasing cdf-lim-neg-infty 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 ⇒ real and x assume f: decseq f f ⟶ a
  then have (λn. cdf M (f n)) ⟶ measure M (⋂ i. {.. f i})
    using ‹decseq f› unfolding cdf-def
    by (intro finite-Lim-measure-decseq) (auto simp: decseq-def)
  also have (⋂ i. {.. f i}) = {.. a}
    using decseq-le[OF f] by (auto intro: order-trans LIMSEQ-le-const[OF f(2)])
  finally show (λn. cdf M (f n)) ⟶ cdf M a
    by (simp add: cdf-def)
qed simp

lemma cdf-at-left: (cdf M ⟶ measure M {..<a}) (at-left a)
proof (rule tendsto-at-left-sequentially[of a − 1])
  fix f :: nat ⇒ real and x assume f: incseq f f ⟶ a ⋀x. f x < a ⋀x. a −
1 < f x
  then have (λn. cdf M (f n)) ⟶ measure M (⋃ i. {.. f i})
    using ‹incseq f› unfolding cdf-def
    by (intro finite-Lim-measure-incseq) (auto simp: incseq-def)
  also have (⋃ 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 (λn. cdf M (f n)) ⟶ measure M {..<a}
    by (simp add: cdf-def)
qed auto

lemma isCont-cdf: isCont (cdf M) x ⟷ measure M {x} = 0
proof −
  have isCont (cdf M) x ⟷ 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} ⟷ 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* **and** *space-eq-univ* [*simp*]: *space M = UNIV*
begin

sublocale *finite-borel-measure M*
by *standard auto*

lemma *cdf-bounded-prob*: ⋀*x. cdf M x ≤ 1*
by (*subst prob-space* [*symmetric*], *rule cdf-bounded*)

lemma *cdf-lim-infty-prob*: (*λi. cdf M* (*real i*)) ⟶ *1*
by (*subst prob-space* [*symmetric*], *rule cdf-lim-infty*)

lemma *cdf-lim-at-top-prob*: (*cdf M* ⟶ *1*) *at-top*
by (*subst prob-space* [*symmetric*], *rule cdf-lim-at-top*)

lemma *measurable-finite-borel* [*simp*]:
f ∈ borel-measurable borel ⟹ f ∈ 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 ⟹ real-distribution (*distr M borel X*)
unfolding *real-distribution-def real-distribution-axioms-def* **by** (*auto intro*!: *prob-space-distr*)

33.2 uniqueness

lemma (**in** *real-distribution*) *emeasure-Ioc*:
assumes *a ≤ b* **shows** *emeasure M {a <.. b} = cdf M b − cdf M a*
proof −
have *{a <.. b} = {..b} − {..a}*
by *auto*
with ‹*a ≤ b*› **show** *?thesis*
by (*simp add*: *emeasure-eq-measure finite-measure-Diff cdf-def*)
qed

lemma *cdf-unique*:
fixes *M1 M2*
assumes *real-distribution M1* **and** *real-distribution M2*
assumes *cdf M1 = cdf M2*
shows *M1 = M2*
proof (*rule measure-eqI-generator-eq*[**where** Ω=*UNIV*])
fix *X* **assume** *X* ∈ *range* (λ(*a*, *b*). {*a*<..*b*::*real*})
then obtain *a b* **where** *Xeq*: *X* = {*a*<..*b*} **by** *auto*
then show *emeasure M1 X = emeasure M2 X*
by (*cases a* ≤ *b*)
(*simp-all add*: *assms*(*1*,*2*)[*THEN real-distribution.emeasure-Ioc*] *assms*(*3*))
next
show (⋃ *i*. {− *real* (*i*::*nat*)<..*real i*}) = *UNIV*
by (*rule UN-Ioc-eq-UNIV*)
qed (*auto simp*: *real-distribution.emeasure-Ioc*[*OF assms*(*1*)]
assms(*1*,*2*)[*THEN real-distribution.events-eq-borel*] *borel-sigma-sets-Ioc*
Int-stable-def)

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
have *ennreal* (*1* − *0*) = (*SUP i*::*nat*. *ennreal* (*F* (*real i*) − *F* (− *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 filterlim-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*::*nat*. *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 show *emeasure ?F* (*space ?F*) = *1*
by (*simp add*: *one-ereal-def*)
qed
show *?thesis*
proof qed *simp-all*
qed

lemma *cdf-interval-measure*:

fixes $F :: real \Rightarrow real$
assumes $nondecF : \bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$ **and**
$right\text{-}cont\text{-}F : \bigwedge a.\ continuous\ (at\text{-}right\ a)\ F$ **and**
$lim\text{-}F\text{-}at\text{-}bot : (F \longrightarrow 0)\ at\text{-}bot$ **and**
$lim\text{-}F\text{-}at\text{-}top : (F \longrightarrow 1)\ at\text{-}top$
shows $cdf\ (interval\text{-}measure\ F) = F$
unfolding *cdf-def*
proof (*intro ext*)
interpret *real-distribution interval-measure F*
by (*rule real-distribution-interval-measure*) *fact+*
fix x
have $F\ x - 0 = measure\ (interval\text{-}measure\ F)\ (\bigcup i{::}nat.\ \{-real\ i <..\ x\})$
proof (*intro LIMSEQ-unique*[*OF - finite-Lim-measure-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*])
then show $(\lambda i.\ measure\ (interval\text{-}measure\ F)\ \{-\ real\ i{<}..x\}) \longrightarrow F\ x - 0$
apply (*rule filterlim-cong*[*OF refl refl*, *THEN iffD1*, *rotated*])
apply (*rule eventually-sequentiallyI*[**where** *c*=*nat* (*ceiling* (− *x*))])
apply (*simp add*: *measure-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 $measure\ (interval\text{-}measure\ F)\ \{..x\} = F\ x$
by *simp*
qed

end

34 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

35 Weak Convergence of Functions

definition
$weak\text{-}conv :: (nat \Rightarrow (real \Rightarrow real)) \Rightarrow (real \Rightarrow real) \Rightarrow bool$
where
$weak\text{-}conv\ F\text{-}seq\ F \equiv \forall x.\ isCont\ F\ x \longrightarrow (\lambda n.\ F\text{-}seq\ n\ x) \longrightarrow F\ x$

36 Weak Convergence of Distributions

definition
weak-conv-m :: (*nat* ⇒ *real measure*) ⇒ *real measure* ⇒ *bool*
where
weak-conv-m M-seq M ≡ *weak-conv* (λ*n*. *cdf* (*M-seq n*)) (*cdf M*)

37 Skorohod's theorem

locale *right-continuous-mono* =
fixes *f* :: *real* ⇒ *real* **and** *a b* :: *real*
assumes *cont*: ⋀*x*. *continuous* (*at-right x*) *f*
assumes *mono*: *mono f*
assumes *bot*: (*f* ⟶ *a*) *at-bot*
assumes *top*: (*f* ⟶ *b*) *at-top*
begin

abbreviation *I* :: *real* ⇒ *real* **where**
I ω ≡ *Inf* {*x*. ω ≤ *f x*}

lemma *pseudoinverse*: **assumes** *a* < ω ω < *b* **shows** ω ≤ *f x* ⟷ *I* ω ≤ *x*
proof
let *?F* = {*x*. ω ≤ *f x*}
obtain *y* **where** *f y* < ω
by (*metis eventually-happens′ trivial-limit-at-bot-linorder order-tendstoD*(2) *bot* ‹*a* < ω›)
with *mono* **have** *bdd*: *bdd-below ?F*
by (*auto intro*!: *bdd-belowI*[*of - y*] *elim*: *mono-invE*[*OF - less-le-trans*])

have *ne*: *?F* ≠ {}
using *order-tendstoD*(1)[*OF top* ‹ω < *b*›]
by (*auto dest*!: *eventually-happens′*[*OF trivial-limit-at-top-linorder*] *intro*: *less-imp-le*)

show ω ≤ *f x* ⟹ *I* ω ≤ *x*
by (*auto intro*!: *cInf-lower bdd*)

{ assume ∗: *I* ω ≤ *x*
have ω ≤ (*INF s*:{*x*. ω ≤ *f x*}. *f s*)
by (*rule cINF-greatest*[*OF ne*]) *auto*
also have ... = *f* (*I* ω)
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 I* {*a* <..< *b*}
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 measure-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* $\Rightarrow$ *real measure*
    and M :: *real measure*
  assumes μ: $\bigwedge n$. *real-distribution* (μ n)
  assumes M: *real-distribution* M
  assumes μ-*to-M*: *weak-conv-m* μ M
begin

theorem *Skorohod*:
$\exists$ (Ω :: *real measure*) (*Y-seq* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *real*) (*Y* :: *real* $\Rightarrow$ *real*).
    prob-space Ω $\wedge$
    ($\forall$ n. *Y-seq* n $\in$ *measurable* Ω *borel*) $\wedge$
    ($\forall$ n. *distr* Ω *borel* (*Y-seq* n) = μ n) $\wedge$
    Y $\in$ *measurable* Ω *lborel* $\wedge$
    distr Ω *borel* Y = M $\wedge$
    ($\forall$ x $\in$ *space* Ω. (λn. *Y-seq* n x) $\longrightarrow$ 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 $\Longrightarrow$ (λn. μ.C n x) $\longrightarrow$ 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-cnv*: (λn. μ.I n ω) $\longrightarrow$ M.I ω
    if ω: ω $\in$ {0<..<1} *isCont* M.I ω **for** ω :: *real*
  proof (*intro limsup-le-liminf-real*)
    show *liminf* (λn. μ.I n ω) $\geq$ M.I ω
      unfolding *le-Liminf-iff*
    proof *safe*
      fix B :: *ereal* **assume** B: B < M.I ω
      then show $\forall_F$ n *in sequentially*. B < μ.I n ω
      proof (*cases* B)
        case (*real* r)
        with B **have** r: r < M.I ω
          by *simp*
        then obtain x **where** x: r < x x < M.I ω *measure* M {x} = 0

```
        using open-minus-countable[OF M.countable-support, of {r<..<M.I ω}]
by auto
      then have Fx-less: M.C x < ω
        using M.pseudoinverse' ω not-less by blast

      have ∀F n in sequentially. μ.C n x < ω
        using order-tendstoD(2)[OF conv[OF x(3)] Fx-less] .
      then have ∀F n in sequentially. x < μ.I n ω
          by eventually-elim (insert ω μ.pseudoinverse[symmetric], simp add:
not-le[symmetric])
      then show ?thesis
        by eventually-elim (insert x(1), simp add: real)
    qed auto
  qed

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

    have ∀F n in sequentially. ω ≤ μ.C n y
      using order-tendstoD(1)[OF conv[OF y(3)] Fy-gt] by eventually-elim (rule
less-imp-le)
    then have 2: ∀F n in sequentially. μ.I n ω ≤ ereal y
      by simp (subst μ.pseudoinverse'[rule-format, OF ω(1), symmetric])
    then show limsup (λn. μ.I n ω) ≤ B'
      using ‹y < B›
    by (intro Limsup-bounded[rotated]) (auto intro: le-less-trans elim: eventually-mono)
  qed simp

  have ∗∗: (M.I ⟶ ereal (M.I ω)) (at-right ω)
    using ω(2) by (auto intro: tendsto-within-subset simp: continuous-at)
  show limsup (λn. μ.I n ω) ≤ M.I ω
    using ω
    by (intro tendsto-le-const[OF trivial-limit-at-right-real ∗∗])
      (auto intro!: exI[of - 1] ∗ simp: eventually-at-right[of - 1])
qed
```

let *?D* = {ω∈{*0*<..<*1*}. ¬ *isCont M.I* ω}
have *D-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*: *emeasure ?*Ω *?D* = *0*
using *emeasure-lborel-countable*[*OF D-countable*]
by (*subst emeasure-restrict-space*) *auto*

def *Y'* ≡ λω. *if* ω ∈ *?D then 0 else M.I* ω
have *Y'-AE*: *AE* ω *in ?*Ω. *Y'* ω = *M.I* ω
by (*rule AE-I* [*OF* - *D*]) (*auto simp*: *space-restrict-space sets-restrict-space-iff Y'-def*)

def *Y-seq'* ≡ λ*n* ω. *if* ω ∈ *?D then 0 else* μ.*I n* ω
have *Y-seq'-AE*: ⋀*n*. *AE* ω *in ?*Ω. *Y-seq' n* ω = μ.*I n* ω
by (*rule AE-I* [*OF* - *D*]) (*auto simp*: *space-restrict-space sets-restrict-space-iff Y-seq'-def*)

have *Y'-cnv*: ∀ω∈{*0*<..<*1*}. (λ*n*. *Y-seq' n* ω) ⟶ *Y'* ω
by (*auto simp*: *Y'-def Y-seq'-def Y-cts-cnv*)

have [*simp*]: *Y-seq' n* ∈ *borel-measurable ?*Ω **for** *n*
by (*rule measurable-discrete-difference*[*of* μ.*I n* - - *?D*])
(*insert* μ.*measurable-CI*[*of n*] *D-countable*, *auto simp*: *sets-restrict-space Y-seq'-def*)
moreover have *distr ?*Ω *borel* (*Y-seq' n*) = μ *n* **for** *n*
using μ.*distr-I-eq-M* [*of n*] *Y-seq'-AE* [*of n*]
by (*subst distr-cong-AE*[**where** *f* = *Y-seq' n* **and** *g* = *M.I n*], *auto*)
moreover have [*simp*]: *Y'* ∈ *borel-measurable ?*Ω
by (*rule measurable-discrete-difference*[*of M.I* - - *?D*])
(*insert M.measurable-CI D-countable*, *auto simp*: *sets-restrict-space Y'-def*)
moreover have *distr ?*Ω *borel Y'* = *M*
using *M.distr-I-eq-M Y'-AE*
by (*subst distr-cong-AE*[**where** *f* = *Y'* **and** *g* = *M.I*], *auto*)
ultimately have *prob-space ?*Ω ∧ (∀ *n*. *Y-seq' n* ∈ *borel-measurable ?*Ω) ∧
(∀ *n*. *distr ?*Ω *borel* (*Y-seq' n*) = μ *n*) ∧ *Y'* ∈ *measurable ?*Ω *lborel* ∧ *distr ?*Ω *borel Y'* = *M* ∧
(∀ *x*∈*space ?*Ω. (λ*n*. *Y-seq' n x*) ⟶ *Y' x*)
using *Y'-cnv* ‹*prob-space ?*Ω› **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 :: *real* ⇒ *'a*::{*banach*, *second-countable-topology*}
assumes
discont-null: *M* ({*x*. ¬ *isCont f x*}) = *0* **and**

f-bdd: $\bigwedge x.\ norm\ (f\ x) \leq B$ **and**
[*measurable*]: $f \in$ *borel-measurable borel*
shows
$(\lambda\ n.\ integral^L\ (\mu\ n)\ f) \longrightarrow integral^L\ M\ f$
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-seq n* $\in$ *borel-measurable Omega* **and**
distr-Y-seq: $\bigwedge n.$ *distr Omega borel* (*Y-seq n*) $= \mu\ n$ **and**
Y-measurable [*measurable*]: $Y \in$ *borel-measurable Omega* **and**
distr-Y: *distr Omega borel Y* $= M$ **and**
YnY: $\bigwedge x :: real.\ x \in$ *space Omega* $\Longrightarrow (\lambda n.$ *Y-seq n x*$) \longrightarrow Y\ x$ **by** *force*
interpret *prob-space Omega* **by** *fact*
have $*$: *emeasure Omega* (*Y* −‘ $\{x.\ \neg$ *isCont f x*$\}$ ∩ *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 $f :: real \Rightarrow 'a::\{banach,\ second\text{-}countable\text{-}topology\}$
assumes
$\bigwedge x.$ *isCont f x* **and**
$\bigwedge x.\ norm\ (f\ x) \leq B$
shows
$(\lambda\ n.\ integral^L\ (\mu\ n)\ f) \longrightarrow integral^L\ M\ f$
using *assms*
by (*intro weak-conv-imp-bdd-ae-continuous-conv*)
(*auto intro*!: *borel-measurable-continuous-on1 continuous-at-imp-continuous-on*)

theorem *weak-conv-imp-continuity-set-conv*:
fixes $f :: real \Rightarrow real$
assumes [*measurable*]: $A \in$ *sets borel* **and** M (*frontier A*) $= 0$
shows $(\lambda n.\ measure\ (\mu\ n)\ A) \longrightarrow measure\ M\ A$
proof −
interpret *M*: *real-distribution M* **by** *fact*
interpret μ: *real-distribution* $\mu\ n$ **for** *n* **by** *fact*

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

by *simp*
qed

end

definition
cts-step :: *real* ⇒ *real* ⇒ *real* ⇒ *real*
where
cts-step a b x ≡ *if x* ≤ *a then 1 else if x* ≥ *b then 0 else* (*b* − *x*) / (*b* − *a*)

lemma *cts-step-uniformly-continuous*:
assumes [*arith*]: *a* < *b*
shows *uniformly-continuous-on UNIV* (*cts-step a b*)
unfolding *uniformly-continuous-on-def*
proof *clarsimp*
fix *e* :: *real* **assume** [*arith*]: *0* < *e*
let *?d* = *min* (*e* ∗ (*b* − *a*)) (*b* − *a*)
have *?d* > *0*
by (*auto simp add*: *field-simps*)
moreover have *dist x′ x* < *?d* ⟹ *dist* (*cts-step a b x′*) (*cts-step a b x*) < *e*
for *x x′*
by (*auto simp*: *dist-real-def divide-simps cts-step-def*)
ultimately show ∃ *d* > *0*. ∀ *x x′*. *dist x′ x* < *d* ⟶ *dist* (*cts-step a b x′*) (*cts-step a b x*) < *e*
by *blast*
qed

lemma (**in** *real-distribution*) *integrable-cts-step*: *a* < *b* ⟹ *integrable M* (*cts-step a b*)
by (*rule integrable-const-bound* [*of - 1*]) (*auto simp*: *cts-step-def*[*abs-def*])

lemma (**in** *real-distribution*) *cdf-cts-step*:
assumes [*arith*]: *x* < *y*
shows *cdf M x* ≤ *integral*^{L} *M* (*cts-step x y*) **and** *integral*^{L} *M* (*cts-step x y*) ≤ *cdf M y*
proof −
have *cdf M x* = *integral*^{L} *M* (*indicator* {*..x*})
by (*simp add*: *cdf-def*)
also have ... ≤ *expectation* (*cts-step x y*)
by (*intro integral-mono integrable-cts-step*)
(*auto simp*: *cts-step-def less-top*[*symmetric*] *split*: *split-indicator*)
finally show *cdf M x* ≤ *expectation* (*cts-step x y*) **.**
next
have *expectation* (*cts-step x y*) ≤ *integral*^{L} *M* (*indicator* {*..y*})
by (*intro integral-mono integrable-cts-step*)
(*auto simp*: *cts-step-def less-top*[*symmetric*] *split*: *split-indicator*)
also have ... = *cdf M y*
by (*simp add*: *cdf-def*)
finally show *expectation* (*cts-step x y*) ≤ *cdf M y* **.**

qed

context
fixes *M-seq* :: *nat* ⇒ *real measure*
and *M* :: *real measure*
assumes *distr-M-seq* [*simp*]: ⋀*n*. *real-distribution* (*M-seq n*)
assumes *distr-M* [*simp*]: *real-distribution M*
begin

theorem *continuity-set-conv-imp-weak-conv*:
fixes *f* :: *real* ⇒ *real*
assumes ∗: ⋀*A*. *A* ∈ *sets borel* ⟹ *M* (*frontier A*) = *0* ⟹ (λ *n*. (*measure* (*M-seq n*) *A*)) ⟶ *measure M A*
shows *weak-conv-m M-seq M*
proof −
interpret *real-distribution M* **by** *simp*
show *?thesis*
by (*auto intro*!: ∗ *simp*: *frontier-real-Iic isCont-cdf emeasure-eq-measure weak-conv-m-def weak-conv-def cdf-def2*)
qed

theorem *integral-cts-step-conv-imp-weak-conv*:
assumes *integral-conv*: ⋀*x y*. *x* < *y* ⟹ (λ*n*. *integral*^{L} (*M-seq n*) (*cts-step x y*)) ⟶ *integral*^{L} *M* (*cts-step x y*)
shows *weak-conv-m M-seq M*
unfolding *weak-conv-m-def weak-conv-def*
proof (*clarsimp*)
interpret *real-distribution M* **by** (*rule distr-M*)
fix *x* **assume** *isCont* (*cdf M*) *x*
hence *left-cont*: *continuous* (*at-left x*) (*cdf M*)
unfolding *continuous-at-split* **..**
{ fix *y* :: *real* **assume** [*arith*]: *x* < *y*
have *limsup* (λ*n*. *cdf* (*M-seq n*) *x*) ≤ *limsup* (λ*n*. *integral*^{L} (*M-seq n*) (*cts-step x y*))
by (*auto intro*!: *Limsup-mono always-eventually real-distribution.cdf-cts-step*)
also have ... = *integral*^{L} *M* (*cts-step x y*)
by (*intro lim-imp-Limsup*) (*auto intro*: *integral-conv*)
also have ... ≤ *cdf M y*
by (*simp add*: *cdf-cts-step*)
finally have *limsup* (λ*n*. *cdf* (*M-seq n*) *x*) ≤ *cdf M y* **.**
} note ∗ = *this*
{ fix *y* :: *real* **assume** [*arith*]: *x* > *y*
have *cdf M y* ≤ *ereal* (*integral*^{L} *M* (*cts-step y x*))
by (*simp add*: *cdf-cts-step*)
also have ... = *liminf* (λ*n*. *integral*^{L} (*M-seq n*) (*cts-step y x*))
by (*intro lim-imp-Liminf*[*symmetric*]) (*auto intro*: *integral-conv*)
also have ... ≤ *liminf* (λ*n*. *cdf* (*M-seq n*) *x*)
by (*auto intro*!: *Liminf-mono always-eventually real-distribution.cdf-cts-step*)
finally have *liminf* (λ*n*. *cdf* (*M-seq n*) *x*) ≥ *cdf M y* **.**

} note $** = this$

have *limsup* $(\lambda n.\ cdf\ (M\text{-}seq\ n)\ x) \leq cdf\ M\ x$
proof (*rule tendsto-le-const*)
show $\forall_F\ i\ in\ at\text{-}right\ x.\ limsup\ (\lambda xa.\ ereal\ (cdf\ (M\text{-}seq\ xa)\ x)) \leq ereal\ (cdf\ M\ i)$
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 $cdf\ M\ x \leq liminf\ (\lambda n.\ cdf\ (M\text{-}seq\ n)\ x)$
proof (*rule tendsto-ge-const*)
show $\forall_F\ i\ in\ at\text{-}left\ x.\ ereal\ (cdf\ M\ i) \leq liminf\ (\lambda xa.\ ereal\ (cdf\ (M\text{-}seq\ xa)\ x))$
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 $(\lambda n.\ cdf\ (M\text{-}seq\ n)\ x) \longrightarrow cdf\ M\ x$
by (*elim limsup-le-liminf-real*)
qed

theorem *integral-bdd-continuous-conv-imp-weak-conv*:
assumes
$\bigwedge f.\ (\bigwedge x.\ isCont\ f\ x) \Longrightarrow (\bigwedge x.\ abs\ (f\ x) \leq 1) \Longrightarrow (\lambda n.\ integral^L\ (M\text{-}seq\ n)\ f :: real) \longrightarrow integral^L\ M\ f$
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

38 Independent families of events, event sets, and random variables

theory *Independent-Family*
imports *Probability-Measure Infinite-Product-Measure*
begin

definition (**in** *prob-space*)
indep-sets F I $\longleftrightarrow$ $(\forall i \in I.\ F\ i \subseteq events) \wedge$
$(\forall J \subseteq I.\ J \neq \{\} \longrightarrow finite\ J \longrightarrow (\forall A \in Pi\ J\ F.\ prob\ (\bigcap j \in J.\ A\ j) = (\prod j \in J.\ prob\ (A\ j))))$

definition (**in** *prob-space*)

indep-set A B ⟷ *indep-sets* (*case-bool A B*) *UNIV*

definition (**in** *prob-space*)
indep-events-def-alt: *indep-events A I* ⟷ *indep-sets* (λ*i*. {*A i*}) *I*

lemma (**in** *prob-space*) *indep-events-def*:
indep-events A I ⟷ (*A*`*I* ⊆ *events*) ∧
(∀ *J*⊆*I*. *J* ≠ {} ⟶ *finite J* ⟶ *prob* (⋂*j*∈*J*. *A j*) = (∏*j*∈*J*. *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*:
(⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *sets M*) ⟹ (⋀*J*. *J* ⊆ *I* ⟹ *finite J* ⟹ *J* ≠ {} ⟹ *prob*
(⋂*i*∈*J*. *F i*) = (∏*i*∈*J*. *prob* (*F i*))) ⟹ *indep-events F I*
by (*auto simp*: *indep-events-def*)

definition (**in** *prob-space*)
indep-event A B ⟷ *indep-events* (*case-bool A B*) *UNIV*

lemma (**in** *prob-space*) *indep-sets-cong*:
I = *J* ⟹ (⋀*i*. *i* ∈ *I* ⟹ *F i* = *G i*) ⟹ *indep-sets F I* ⟷ *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*:
indep-events F I ⟷ (∀ *J*⊆*I*. *J* ≠ {} ⟶ *finite J* ⟶ *indep-events F J*)
by (*auto simp*: *indep-events-def*)

lemma (**in** *prob-space*) *indep-sets-finite-index-sets*:
indep-sets F I ⟷ (∀ *J*⊆*I*. *J* ≠ {} ⟶ *finite J* ⟶ *indep-sets F J*)
proof (*intro iffI allI impI*)
assume ∗: ∀ *J*⊆*I*. *J* ≠ {} ⟶ *finite J* ⟶ *indep-sets F J*
show *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 $(\forall i{\in}I.\ F\ i \subseteq events) \Longrightarrow (\forall i{\in}I.\ G\ i \subseteq events)$
using *mono* **by** *auto*
moreover have $\bigwedge A\ J.\ J \subseteq I \Longrightarrow A \in (\Pi\ j{\in}J.\ G\ j) \Longrightarrow A \in (\Pi\ j{\in}J.\ F\ j)$
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*: $J \subseteq I\ \bigwedge i.\ i{\in}J \Longrightarrow G\ i \subseteq F\ i$
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 $\bigwedge i.\ i \in I \Longrightarrow F\ i \subseteq events$
and $\bigwedge A\ J.\ J \neq \{\} \Longrightarrow J \subseteq I \Longrightarrow finite\ J \Longrightarrow (\forall j{\in}J.\ A\ j \in F\ j) \Longrightarrow prob\ (\bigcap j{\in}J.\ A\ j) = (\prod j{\in}J.\ prob\ (A\ j))$
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** $J \subseteq I\ J \neq \{\}$ *finite J* $\forall j{\in}J.\ A\ j \in F\ j$
shows $prob\ (\bigcap j{\in}J.\ A\ j) = (\prod j{\in}J.\ prob\ (A\ j))$
using *assms* **unfolding** *indep-sets-def* **by** *auto*

lemma (**in** *prob-space*) *indep-setI*:
assumes *ev*: $A \subseteq events\ B \subseteq events$
and *indep*: $\bigwedge a\ b.\ a \in A \Longrightarrow b \in B \Longrightarrow prob\ (a \cap b) = prob\ a * prob\ b$
shows *indep-set A B*
unfolding *indep-set-def*
proof (*rule indep-setsI*)
fix *F J* **assume** $J \neq \{\}\ J \subseteq UNIV$
and *F*: $\forall j{\in}J.\ F\ j \in (case\ j\ of\ True \Rightarrow A \mid False \Rightarrow B)$
have *J* ∈ *Pow UNIV* **by** *auto*
with *F* ‹$J \neq \{\}$› *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* (λi. *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* = λS *i*. *if i* $\in$ *S then ?F i else F i*
assume *finite J J* $\subseteq$ *K*
then have *indep-sets* (*?G J*) *K*
proof *induct*
case (*insert j J*)
moreover def *G* $\equiv$ *?G J*
ultimately have *G*: *indep-sets G K* $\bigwedge i$. *i* $\in$ *K* $\Longrightarrow$ *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* $\neq$ {} $\Longrightarrow$ *J* $\subseteq$ *K* $\Longrightarrow$ *finite J* $\Longrightarrow$ *j* $\notin$ *J* $\Longrightarrow$ ($\forall$ *i*$\in$*J*. *A i* $\in$ *G i*)
$\Longrightarrow$ *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* $\neq$ {} *J* $\subseteq$ *K finite J* $\forall$ *i*$\in$*J*. *A i* $\in$ (*G*(*j* := {*X*})) *i*
show *prob* ($\bigcap$ *j*$\in$*J*. *A j*) = ($\prod$ *j*$\in$*J*. *prob* (*A j*))
proof *cases*
assume *j* $\in$ *J*
with *J* **have** *A j* = *X* **by** *auto*
show *?thesis*
proof *cases*
assume *J* = {*j*} **then show** *?thesis* **by** *simp*
next
assume *J* $\neq$ {*j*}
have *prob* ($\bigcap$ *i*$\in$*J*. *A i*) = *prob* (($\bigcap$ *i*$\in$*J*$-${*j*}. *A i*) $\cap$ *X*)
using ‹*j* $\in$ *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 setprod.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!: setprod.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!: setprod.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*: $\bigwedge i.\ i \in I \Longrightarrow$ *Int-stable* (*F i*)
shows *indep-sets* (λ*i*. *sigma-sets* (*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** $i \in I$
with *indep* **have** $F\ i \subseteq$ *events* **by** (*auto simp*: *indep-sets-def*)
with *sets.sets-into-space* **show** $F\ i \subseteq$ *Pow* (*space M*) **by** *auto*
qed
qed

lemma (**in** *prob-space*) *indep-sets-sigma-sets-iff*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *Int-stable* (*F i*)
shows *indep-sets* (λ*i*. *sigma-sets* (*space M*) (*F i*)) *I* ⟷ *indep-sets F I*
proof
assume *indep-sets F I* **then show** *indep-sets* (λ*i*. *sigma-sets* (*space M*) (*F i*)) *I*
by (*rule indep-sets-sigma*) *fact*
next
assume *indep-sets* (λ*i*. *sigma-sets* (*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* ⟷
(∀ *i*∈*I*. *random-variable* (*M′ i*) (*X i*)) ∧
indep-sets (λ*i*. { *X i* −‘ *A* ∩ *space M* | *A*. *A* ∈ *sets* (*M′ i*)}) *I*

definition (**in** *prob-space*)
indep-var Ma A Mb B ⟷ *indep-vars* (*case-bool Ma Mb*) (*case-bool A B*) *UNIV*

lemma (**in** *prob-space*) *indep-vars-def*:
indep-vars M′ X I ⟷
(∀ *i*∈*I*. *random-variable* (*M′ i*) (*X i*)) ∧
indep-sets (λ*i*. *sigma-sets* (*space M*) { *X i* −‘ *A* ∩ *space M* | *A*. *A* ∈ *sets* (*M′ 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 ⟷
(*random-variable S X* ∧ *random-variable T Y*) ∧

```
    indep-set
      (sigma-sets (space M) { X -` A ∩ space M | A. A ∈ sets S})
      (sigma-sets (space M) { Y -` A ∩ space M | A. A ∈ sets T})
  unfolding indep-var-def indep-vars-def indep-set-def UNIV-bool
  by (intro arg-cong2[where f=op ∧] arg-cong2[where f=indep-sets] ext)
     (auto split: bool.split)

lemma (in prob-space) indep-sets2-eq:
  indep-set A B ⟷ A ⊆ events ∧ B ⊆ events ∧ (∀ a∈A. ∀ b∈B. prob (a ∩ 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 ∈ A b ∈ B
    with indep-setsD[OF indep, of UNIV case-bool a b]
    show prob (a ∩ b) = prob a * prob b
      unfolding UNIV-bool by (simp add: ac-simps) }
  from indep show A ⊆ events B ⊆ events
    unfolding indep-sets-def UNIV-bool by auto
next
  assume *: A ⊆ events ∧ B ⊆ events ∧ (∀ a∈A. ∀ b∈B. prob (a ∩ b) = prob a
* prob b)
  show indep-sets (case-bool A B) UNIV
  proof (rule indep-setsI)
    fix i show (case i of True ⇒ A | False ⇒ B) ⊆ events
      using * by (auto split: bool.split)
  next
    fix J X assume J ≠ {} J ⊆ UNIV and X: ∀ j∈J. X j ∈ (case j of True ⇒
A | False ⇒ B)
    then have J = {True} ∨ J = {False} ∨ J = {True,False}
      by (auto simp: UNIV-bool)
    then show prob (⋂ j∈J. X j) = (∏ j∈J. 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 (λi. sigma-sets (space M) (case i of True ⇒ A | False ⇒ 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 ⇒ A | False ⇒ 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*: $\bigwedge i.\ i \in I \Longrightarrow \{x \in space\ (N\ i).\ P\ i\ x\} \in sets\ (N\ i)$
shows *indep-events* $(\lambda i.\ \{x \in space\ M.\ P\ i\ (X\ i\ x)\})\ I$
proof −
have *indep-sets* $(\lambda i.\ \{X\ i -` A \cap space\ M\ |A.\ A \in sets\ (N\ i)\})\ I$
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 $\{\{x \in space\ M.\ P\ i\ (X\ i\ x)\}\} = \{X\ i -` \{x \in space\ (N\ i).\ P\ i\ x\} \cap space\ M\}$
using *indep* **by** (*auto simp*: *indep-vars-def dest*: *measurable-space*)
also have $\ldots \subseteq \{X\ i -` A \cap space\ M\ |A.\ A \in sets\ (N\ i)\}$
using *P*[*OF* ‹*i* ∈ *I*›] **by** *blast*
finally show $\{\{x \in space\ M.\ P\ i\ (X\ i\ x)\}\} \subseteq \{X\ i -` A \cap space\ M\ |A.\ A \in sets\ (N\ i)\}$ **.**
qed
qed

lemma (**in** *prob-space*) *indep-sets-collect-sigma*:
fixes $I :: 'j \Rightarrow 'i\ set$ **and** $J :: 'j\ set$ **and** $E :: 'i \Rightarrow 'a\ set\ set$
assumes *indep*: *indep-sets E* $(\bigcup j \in J.\ I\ j)$
assumes *Int-stable*: $\bigwedge i\ j.\ j \in J \Longrightarrow i \in I\ j \Longrightarrow$ *Int-stable* $(E\ i)$
assumes *disjoint*: *disjoint-family-on I J*
shows *indep-sets* $(\lambda j.\ sigma\text{-}sets\ (space\ M)\ (\bigcup i \in I\ j.\ E\ i))\ J$
proof −
let *?E* $= \lambda j.\ \{\bigcap k \in K.\ E'\ k |\ E'\ K.\ finite\ K \wedge K \neq \{\} \wedge K \subseteq I\ j \wedge (\forall k \in K.\ E'\ k \in E\ k)\ \}$

from *indep* **have** *E*: $\bigwedge j\ i.\ j \in J \Longrightarrow i \in I\ j \Longrightarrow E\ i \subseteq events$
unfolding *indep-sets-def* **by** *auto*
{ fix *j*
let *?S* $=$ *sigma-sets* $(space\ M)\ (\bigcup i \in I\ j.\ E\ i)$
assume $j \in J$
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* $(space\ M)\ (\bigcup i \in I\ j.\ E\ i) = sigma\text{-}sets\ (space\ M)\ (?E\ j)$
proof (*rule sigma-sets-eqI*)
fix *A* **assume** $A \in (\bigcup i \in I\ j.\ E\ i)$
then guess *i* **..**
then show $A \in sigma\text{-}sets\ (space\ M)\ (?E\ j)$

```
      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 ∈ 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] guess E' ..
      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 ‹K ⊆ J› ‹k ∈ K› ‹j ∈ K› have I k ∩ I j = {}
            unfolding disjoint-family-on-def by auto
          with L(2,3)[OF ‹j ∈ K›] L(2,3)[OF ‹k ∈ K›]
          show False using ‹l ∈ L k› ‹l ∈ L j› by auto
        qed }
      note L-inj = this

      def k ≡ λl. (SOME k. k ∈ K ∧ l ∈ L k)
      { fix x j l assume ∗: j ∈ K l ∈ L j
        have k l = j unfolding k-def
        proof (rule some-equality)
          fix k assume k ∈ K ∧ l ∈ L k
          with ∗ L-inj show k = j by auto
```

qed (*insert* $*$, *simp*) **}**
note *k-simp*[*simp*] = *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 ... = ($\prod l{\in}(\bigcup k{\in}K.\ L\ k).\ prob\ (?E'\ l)$)
using *L K E'* **by** (*intro indep-setsD*[*OF indep*]) (*simp-all add*: *UN-mono*)
also have ... = ($\prod j{\in}K.\ \prod l{\in}L\ j.\ prob\ (E'\ j\ l)$)
using *K L L-inj* **by** (*subst setprod.UNION-disjoint*) *auto*
also have ... = ($\prod j{\in}K.\ prob\ (A\ j)$)
using *K L E'* **by** (*auto simp add*: *A intro*!: *setprod.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* $\neq \{\}$ $Ka \subseteq I\ j$ $\bigwedge k.\ k{\in}Ka \Longrightarrow Ea\ k \in E\ k$ **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* $\neq \{\}$ $Kb \subseteq I\ j$ $\bigwedge k.\ k{\in}Kb \Longrightarrow Eb\ k \in E\ k$ **by** *auto*
let *?f* = $\lambda k.$ (*if* $k \in Ka \cap Kb$ *then Ea k* $\cap$ *Eb k else if* $k \in Kb$ *then Eb k else if* $k \in Ka$ *then Ea k 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* ‹$j \in J$› *Int-stableD*[*OF Int-stable*] **show** $a \cap b \in ?E\ j$
by (*intro CollectI exI*[*of* - $Ka \cup Kb$] *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*: $\bigwedge j.\ j \in L \Longrightarrow K\ j \subseteq I$ **and** *J*: *disjoint-family-on K L*
shows *indep-vars* ($\lambda j.$ *PiM* (*K j*) *M'*) ($\lambda j\ \omega.$ *restrict* ($\lambda i.$ *X i* ω) (*K j*)) *L*

```
  unfolding indep-vars-def
proof safe
  fix j assume j ∈ L then show random-variable (Pi_M (K j) M') (λω. λi∈K j.
X i ω)
    using K ind by (auto simp: indep-vars-def intro!: measurable-restrict)
next
  have X: ⋀i. i ∈ I ⟹ X i ∈ measurable M (M' i)
    using ind by (auto simp: indep-vars-def)
  let ?proj = λj S. {(λω. λi∈K j. X i ω) -` A ∩ space M |A. A ∈ S}
  let ?UN = λj. sigma-sets (space M) (⋃i∈K j. { X i -` A ∩ space M| A. A ∈
sets (M' i) })
  show indep-sets (λi. 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: ∀i. i ∈ J
⟶ E i ∈ sets (M' i)
      show (λω. λi∈K j. X i ω) -` prod-emb (K j) M' J (Pi_E J E) ∩ space M ∈
?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 ⋀A. (⋀i. i ∈ J ⟹ A i ∈ ?UN j) ⟹ INTER J A ∈ ?UN j
            using ‹finite J› ‹J ≠ {}› by (rule finite-INT) blast }
        note INT = this

        from ‹J ≠ {}› J K E[rule-format, THEN sets.sets-into-space] j
        have (λω. λi∈K j. X i ω) -` prod-emb (K j) M' J (Pi_E J E) ∩ space M
          = (⋂i∈J. X i -` E i ∩ 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 ... ∈ ?UN j
```

```
        apply (rule INT)
        apply (rule sigma-sets.Basic)
        using ‹J ⊆ K j› E
        apply auto
        done
      finally show ?thesis .
    qed
  qed
  finally show sigma-sets (space M) (?proj j (sets (Pi_M (K j) M'))) ⊆ ?UN j .
 next
  show indep-sets ?UN L
  proof (rule indep-sets-collect-sigma)
   show indep-sets (λi. {X i -‘ A ∩ space M |A. A ∈ sets (M' i)}) (⋃j∈L. K
j)
    proof (rule indep-sets-mono-index)
     show indep-sets (λi. {X i -‘ A ∩ space M |A. A ∈ sets (M' i)}) I
      using ind unfolding indep-vars-def2 by auto
     show (⋃l∈L. K l) ⊆ I
      using K by auto
    qed
   next
    fix l i assume l ∈ L i ∈ K l
    show Int-stable {X i -‘ A ∩ space M |A. A ∈ sets (M' i)}
     apply (auto simp: Int-stable-def)
     apply (rule-tac x=A ∩ 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 ∩ B = {} A ⊆ I B ⊆ I
  shows indep-var (PiM A M') (λω. restrict (λi. X i ω) A) (PiM B M') (λω.
restrict (λi. X i ω) B)
proof -
  have ∗:
    case-bool (Pi_M A M') (Pi_M 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*:
$I = J \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow X\ i = Y\ i) \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow M'\ i = N'\ i) \Longrightarrow$ *indep-vars M′ X I* $\longleftrightarrow$ *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 = $(\bigcap n.$ *sigma-sets* (*space M*) (*UNION* {*n*..} *A*))

lemma (**in** *prob-space*) *tail-events-sets*:
assumes *A*: $\bigwedge i$::*nat*. *A i* $\subseteq$ *events*
shows *tail-events A* $\subseteq$ *events*
proof
fix *X* **assume** *X*: *X* $\in$ *tail-events A*
let *?A* = $(\bigcap n.$ *sigma-sets* (*space M*) (*UNION* {*n*..} *A*))
from *X* **have** $\bigwedge n$::*nat*. *X* $\in$ *sigma-sets* (*space M*) (*UNION* {*n*..} *A*) **by** (*auto simp*: *tail-events-def*)
from *this*[*of 0*] **have** *X* $\in$ *sigma-sets* (*space M*) (*UNION UNIV A*) **by** *simp*
then show *X* $\in$ *events*
by *induct* (*insert A*, *auto*)
qed

lemma (**in** *prob-space*) *sigma-algebra-tail-events*:
assumes $\bigwedge i$::*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* = $(\bigcap n.$ *sigma-sets* (*space M*) (*UNION* {*n*..} *A*))
interpret *A*: *sigma-algebra space M A i* **for** *i* **by** *fact*
{ fix *X x* **assume** *X* $\in$ *?A x* $\in$ *X*
then have $\bigwedge n$. *X* $\in$ *sigma-sets* (*space M*) (*UNION* {*n*..} *A*) **by** *auto*
from *this*[*of 0*] **have** *X* $\in$ *sigma-sets* (*space M*) (*UNION UNIV A*) **by** *simp*
then have *X* $\subseteq$ *space M*
by *induct* (*insert A.sets-into-space*, *auto*)
with ‹*x* $\in$ *X*› **show** *x* $\in$ *space M* **by** *auto* **}**
{ fix *F* :: *nat* $\Rightarrow$ *'a set* **and** *n* **assume** *range F* $\subseteq$ *?A*
then show (*UNION UNIV F*) $\in$ *sigma-sets* (*space M*) (*UNION* {*n*..} *A*)
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* :: *nat* $\Rightarrow$ *'a set set*
assumes $\bigwedge i$::*nat*. *sigma-algebra* (*space M*) (*A i*)
assumes *indep*: *indep-sets A UNIV*
and *X*: *X* $\in$ *tail-events A*
shows *prob X* = *0* $\vee$ *prob X* = *1*

proof −
have *A*: $\bigwedge i.\ A\ i \subseteq events$
using *indep* **unfolding** *indep-sets-def* **by** *simp*

let *?D* = $\{D \in events.\ prob\ (X \cap D) = prob\ X * prob\ D\}$
interpret *A*: *sigma-algebra space M A i* **for** *i* **by** *fact*
interpret *T*: *sigma-algebra space M tail-events A*
by (*rule sigma-algebra-tail-events*) *fact*
have $X \subseteq space\ M$ **using** *T.space-closed X* **by** *auto*

have *X-in*: $X \in events$
using *tail-events-sets A X* **by** *auto*

interpret *D*: *dynkin-system space M ?D*
proof (*rule dynkin-systemI*)
fix *D* **assume** $D \in ?D$ **then show** $D \subseteq space\ M$
using *sets.sets-into-space* **by** *auto*
next
show $space\ M \in ?D$
using *prob-space* ‹$X \subseteq space\ M$› **by** (*simp add*: *Int-absorb2*)
next
fix *A* **assume** *A*: $A \in ?D$
have $prob\ (X \cap (space\ M - A)) = prob\ (X - (X \cap A))$
using ‹$X \subseteq space\ M$› **by** (*auto intro*!: *arg-cong*[**where** *f*=*prob*])
also have $\ldots = prob\ X - prob\ (X \cap A)$
using *X-in A* **by** (*intro finite-measure-Diff*) *auto*
also have $\ldots = prob\ X * prob\ (space\ M) - prob\ X * prob\ A$
using *A prob-space* **by** *auto*
also have $\ldots = prob\ X * prob\ (space\ M - A)$
using *X-in A sets.sets-into-space*
by (*subst finite-measure-Diff*) (*auto simp*: *field-simps*)
finally show $space\ M - A \in ?D$
using *A* ‹$X \subseteq space\ M$› **by** *auto*
next
fix *F* :: $nat \Rightarrow 'a\ set$ **assume** *dis*: *disjoint-family F* **and** $range\ F \subseteq ?D$
then have *F*: $range\ F \subseteq events$ $\bigwedge i.\ prob\ (X \cap F\ i) = prob\ X * prob\ (F\ i)$
by *auto*
have $(\lambda i.\ prob\ (X \cap F\ i))\ sums\ prob\ (\bigcup i.\ X \cap F\ i)$
proof (*rule finite-measure-UNION*)
show $range\ (\lambda i.\ X \cap F\ i) \subseteq events$
using *F X-in* **by** *auto*
show *disjoint-family* $(\lambda i.\ X \cap F\ i)$
using *dis* **by** (*rule disjoint-family-on-bisimulation*) *auto*
qed
with *F* **have** $(\lambda i.\ prob\ X * prob\ (F\ i))\ sums\ prob\ (X \cap (\bigcup i.\ F\ i))$
by *simp*
moreover have $(\lambda i.\ prob\ X * prob\ (F\ i))\ sums\ (prob\ X * prob\ (\bigcup i.\ F\ i))$
by (*intro sums-mult finite-measure-UNION F dis*)
ultimately have $prob\ (X \cap (\bigcup i.\ F\ i)) = prob\ X * prob\ (\bigcup i.\ F\ i)$

```
    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 (⋃ n. sigma-sets (space M) (⋃ m∈{..n}. A m)) ⊆ ?D (is ?A ⊆ -)
  by auto

 note ‹X ∈ tail-events A›
 also {
  have ⋀n. sigma-sets (space M) (⋃ i∈{n..}. A i) ⊆ sigma-sets (space M) ?A
    by (intro sigma-sets-subseteq UN-mono) auto
 then have tail-events A ⊆ sigma-sets (space M) ?A
    unfolding tail-events-def by auto }
 also have sigma-sets (space M) ?A = dynkin (space M) ?A
 proof (rule sigma-eq-dynkin)
  { fix B n assume B ∈ sigma-sets (space M) (⋃ m∈{..n}. A m)
```

```
      then have B ⊆ space M
        by induct (insert A sets.sets-into-space[of - M], auto) }
    then show ?A ⊆ Pow (space M) by auto
    show Int-stable ?A
    proof (rule Int-stableI)
      fix a assume a ∈ ?A then guess n .. note a = this
      fix b assume b ∈ ?A then guess m .. note b = this
      interpret Amn: sigma-algebra space M sigma-sets (space M) (⋃i∈{..max m
n}. A i)
        using A sets.sets-into-space[of - M] by (intro sigma-algebra-sigma-sets)
auto
    have sigma-sets (space M) (⋃i∈{..n}. A i) ⊆ sigma-sets (space M) (⋃i∈{..max
m n}. A i)
        by (intro sigma-sets-subseteq UN-mono) auto
      with a have a ∈ sigma-sets (space M) (⋃i∈{..max m n}. A i) by auto
      moreover
    have sigma-sets (space M) (⋃i∈{..m}. A i) ⊆ sigma-sets (space M) (⋃i∈{..max
m n}. A i)
        by (intro sigma-sets-subseteq UN-mono) auto
      with b have b ∈ sigma-sets (space M) (⋃i∈{..max m n}. A i) by auto
      ultimately have a ∩ b ∈ sigma-sets (space M) (⋃i∈{..max m n}. A i)
        using Amn.Int[of a b] by simp
      then show a ∩ b ∈ (⋃n. sigma-sets (space M) (⋃i∈{..n}. A i)) by auto
    qed
  qed
  also have dynkin (space M) ?A ⊆ ?D
    using ‹?A ⊆ ?D› by (auto intro!: D.dynkin-subset)
  finally show ?thesis by auto
qed

lemma (in prob-space) borel-0-1-law:
  fixes F :: nat ⇒ 'a set
  assumes F2: indep-events F UNIV
  shows prob (⋂n. ⋃m∈{n..}. F m) = 0 ∨ prob (⋂n. ⋃m∈{n..}. F m) = 1
proof (rule kolmogorov-0-1-law[of λi. sigma-sets (space M) { F i }])
  have F1: range F ⊆ 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 ≠ {} finite I
    and F: ⋀i. i ∈ I ⟹ F i ⊆ events ⋀i. i ∈ I ⟹ space M ∈ F i
  shows indep-sets F I ⟷ (∀ A∈Pi I F. prob (⋂ j∈I. A j) = (∏ j∈I. prob (A
j)))
proof
```

```
  assume ∗: indep-sets F I
  from I show ∀ A∈Pi I F. prob (⋂j∈I. A j) = (∏j∈I. prob (A j))
    by (intro indep-setsD[OF ∗] ballI) auto
next
  assume indep: ∀ A∈Pi I F. prob (⋂j∈I. A j) = (∏j∈I. prob (A j))
  show indep-sets F I
  proof (rule indep-setsI[OF F(1)])
    fix A J assume J: J ≠ {} J ⊆ I finite J
    assume A: ∀ j∈J. A j ∈ F j
    let ?A = λj. if j ∈ J then A j else space M
    have prob (⋂j∈I. ?A j) = prob (⋂j∈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 (λj. if j ∈ J then A j else space M) ∈ Pi I F (is ?A ∈ -)
      by (auto split: if-split-asm)
    with indep have prob (⋂j∈I. ?A j) = (∏j∈I. prob (?A j))
      by auto
    also have ... = (∏j∈J. prob (A j))
      unfolding if-distrib setprod.If-cases[OF ‹finite I›]
      using prob-space ‹J ⊆ I› by (simp add: Int-absorb1 setprod.neutral-const)
    finally show prob (⋂j∈J. A j) = (∏j∈J. prob (A j)) ..
  qed
qed

lemma (in prob-space) indep-vars-finite:
  fixes I :: 'i set
  assumes I: I ≠ {} finite I
    and M': ⋀i. i ∈ I ⟹ sets (M' i) = sigma-sets (space (M' i)) (E i)
    and rv: ⋀i. i ∈ I ⟹ random-variable (M' i) (X i)
    and Int-stable: ⋀i. i ∈ I ⟹ Int-stable (E i)
    and space: ⋀i. i ∈ I ⟹ space (M' i) ∈ E i and closed: ⋀i. i ∈ I ⟹ E i ⊆
Pow (space (M' i))
  shows indep-vars M' X I ⟷
    (∀ A∈(Π i∈I. E i). prob (⋂j∈I. X j -` A j ∩ space M) = (∏j∈I. prob (X j
-` A j ∩ space M)))
proof -
  from rv have X: ⋀i. i ∈ I ⟹ X i ∈ space M → space (M' i)
    unfolding measurable-def by simp

  { fix i assume i∈I
    from closed[OF ‹i ∈ I›]
    have sigma-sets (space M) {X i -` A ∩ space M |A. A ∈ sets (M' i)}
      = sigma-sets (space M) {X i -` A ∩ space M |A. A ∈ E i}
     unfolding sigma-sets-vimage-commute[OF X, OF ‹i ∈ I›, symmetric] M'[OF
‹i ∈ I›]
      by (subst sigma-sets-sigma-sets-eq) auto }
  note sigma-sets-X = this
```

{ fix i **assume** $i{\in}I$
have *Int-stable* $\{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\}$
proof (*rule Int-stableI*)
fix a **assume** $a \in \{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\}$
then obtain A **where** $a = X\ i\ -`\ A \cap \mathit{space}\ M\ A \in E\ i$ **by** *auto*
moreover
fix b **assume** $b \in \{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\}$
then obtain B **where** $b = X\ i\ -`\ B \cap \mathit{space}\ M\ B \in E\ i$ **by** *auto*
moreover
have $(X\ i\ -`\ A \cap \mathit{space}\ M) \cap (X\ i\ -`\ B \cap \mathit{space}\ M) = X\ i\ -`\ (A \cap B) \cap \mathit{space}\ M$ **by** *auto*
moreover note *Int-stable*[*OF* ‹$i \in I$›]
ultimately
show $a \cap b \in \{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\}$
by (*auto simp del*: *vimage-Int intro*!: *exI*[*of* - $A \cap B$] *dest*: *Int-stableD*)
qed }
note *indep-sets-X* = *indep-sets-sigma-sets-iff*[*OF this*]

{ fix i **assume** $i \in I$
{ fix A **assume** $A \in E\ i$
with M'[*OF* ‹$i \in I$›] **have** $A \in \mathit{sets}\ (M'\ i)$ **by** *auto*
moreover
from *rv*[*OF* ‹$i{\in}I$›] **have** $X\ i \in \mathit{measurable}\ M\ (M'\ i)$ **by** *auto*
ultimately
have $X\ i\ -`\ A \cap \mathit{space}\ M \in \mathit{sets}\ M$ **by** (*auto intro*: *measurable-sets*) **}**
with X[*OF* ‹$i{\in}I$›] *space*[*OF* ‹$i{\in}I$›]
have $\{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\} \subseteq \mathit{events}$
$\mathit{space}\ M \in \{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\}$
by (*auto intro*!: *exI*[*of* - *space* $(M'\ i)$]) **}**
note *indep-sets-finite-X* = *indep-sets-finite*[*OF I this*]

have $(\forall A{\in}\Pi\ i{\in}I.\ \{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\}.\ \mathit{prob}\ (\mathit{INTER}\ I\ A) = (\prod j{\in}I.\ \mathit{prob}\ (A\ j))) =$
$(\forall A{\in}\Pi\ i{\in}I.\ E\ i.\ \mathit{prob}\ ((\bigcap j{\in}I.\ X\ j\ -`\ A\ j) \cap \mathit{space}\ M) = (\prod x{\in}I.\ \mathit{prob}\ (X\ x\ -`\ A\ x \cap \mathit{space}\ M)))$
(**is** *?L* = *?R*)
proof *safe*
fix A **assume** *?L* **and** A: $A \in (\Pi\ i{\in}I.\ E\ i)$
from ‹*?L*›[*THEN bspec*, *of* $\lambda i.\ X\ i\ -`\ A\ i \cap \mathit{space}\ M$] A ‹$I \neq \{\}$›
show *prob* $((\bigcap j{\in}I.\ X\ j\ -`\ A\ j) \cap \mathit{space}\ M) = (\prod x{\in}I.\ \mathit{prob}\ (X\ x\ -`\ A\ x \cap \mathit{space}\ M))$
by (*auto simp add*: *Pi-iff*)
next
fix A **assume** *?R* **and** A: $A \in (\Pi\ i{\in}I.\ \{X\ i\ -`\ A \cap \mathit{space}\ M\ |A.\ A \in E\ i\})$
from A **have** $\forall i{\in}I.\ \exists B.\ A\ i = X\ i\ -`\ B \cap \mathit{space}\ M \wedge B \in E\ i$ **by** *auto*
from *bchoice*[*OF this*] **obtain** B **where** B: $\forall i{\in}I.\ A\ i = X\ i\ -`\ B\ i \cap \mathit{space}\ M$
$B \in (\Pi\ i{\in}I.\ E\ i)$ **by** *auto*
from ‹*?R*›[*THEN bspec*, *OF* $B(2)$] $B(1)$ ‹$I \neq \{\}$›
show *prob* $(\mathit{INTER}\ I\ A) = (\prod j{\in}I.\ \mathit{prob}\ (A\ j))$

```
    by simp
  qed
  then show ?thesis using ‹I ≠ {}›
    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: ⋀i. i ∈ I ⟹ Y i ∈ measurable (M' i) (N i)
  shows indep-vars N (λi. Y i ∘ X i) I
  unfolding indep-vars-def
proof
  from rv ‹indep-vars M' X I›
  show ∀ i∈I. random-variable (N i) (Y i ∘ X i)
    by (auto simp: indep-vars-def)

  have indep-sets (λi. sigma-sets (space M) {X i -` A ∩ space M |A. A ∈ sets
(M' i)}) I
    using ‹indep-vars M' X I› by (simp add: indep-vars-def)
  then show indep-sets (λi. sigma-sets (space M) {(Y i ∘ X i) -` A ∩ space M
|A. A ∈ sets (N i)}) I
  proof (rule indep-sets-mono-sets)
    fix i assume i ∈ I
    with ‹indep-vars M' X I› have X: X i ∈ space M → space (M' i)
      unfolding indep-vars-def measurable-def by auto
    { fix A assume A ∈ sets (N i)
      then have ∃ B. (Y i ∘ X i) -` A ∩ space M = X i -` B ∩ space M ∧ B ∈
sets (M' i)
        by (intro exI[of - Y i -` A ∩ space (M' i)])
         (auto simp: vimage-comp intro!: measurable-sets rv ‹i ∈ I› funcset-mem[OF
X]) }
    then show sigma-sets (space M) {(Y i ∘ X i) -` A ∩ space M |A. A ∈ sets
(N i)} ⊆
      sigma-sets (space M) {X i -` A ∩ space M |A. A ∈ 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: ⋀i. i ∈ I ⟹ Y i ∈ measurable (M' i) (N i)
  shows indep-vars N (λi 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 ∈ measurable M1 N1 Y2 ∈ measurable
M2 N2
  shows indep-var N1 (Y1 ∘ X1) N2 (Y2 ∘ X2)
```

proof −
have *indep-vars* (*case-bool N1 N2*) (λ*b*. *case-bool Y1 Y2 b* ∘ *case-bool X1 X2 b*) *UNIV*
using *assms*
by (*intro indep-vars-compose*[**where** *M'*=*case-bool M1 M2*])
(*auto simp*: *indep-var-def split*: *bool.split*)
also have (λ*b*. *case-bool Y1 Y2 b* ∘ *case-bool X1 X2 b*) = *case-bool* (*Y1* ∘ *X1*) (*Y2* ∘ *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* ⇒ *'a* ⇒ *real*
assumes *I*: *finite I i* ∉ *I* **and** *indep*: *indep-vars* (λ-. *borel*) *X* (*insert i I*)
shows *indep-var borel* (*X i*) *borel* (λω. *Min* ((λ*i*. *X i* ω)‘*I*))
proof −
have *indep-var*
borel ((λ*f*. *f i*) ∘ (λω. *restrict* (λ*i*. *X i* ω) {*i*}))
borel ((λ*f*. *Min* (*f*‘*I*)) ∘ (λω. *restrict* (λ*i*. *X i* ω) *I*))
using *I* **by** (*intro indep-var-compose*[*OF indep-var-restrict*[*OF indep*]] *borel-measurable-Min*) *auto*
also have ((λ*f*. *f i*) ∘ (λω. *restrict* (λ*i*. *X i* ω) {*i*})) = *X i*
by *auto*
also have ((λ*f*. *Min* (*f*‘*I*)) ∘ (λω. *restrict* (λ*i*. *X i* ω) *I*)) = (λω. *Min* ((λ*i*. *X i* ω)‘*I*))
by (*auto cong*: *rev-conj-cong*)
finally show *?thesis*
unfolding *indep-var-def* **.**
qed

lemma (**in** *prob-space*) *indep-vars-setsum*:
fixes *X* :: *'i* ⇒ *'a* ⇒ *real*
assumes *I*: *finite I i* ∉ *I* **and** *indep*: *indep-vars* (λ-. *borel*) *X* (*insert i I*)
shows *indep-var borel* (*X i*) *borel* (λω. ∑*i*∈*I*. *X i* ω)
proof −
have *indep-var*
borel ((λ*f*. *f i*) ∘ (λω. *restrict* (λ*i*. *X i* ω) {*i*}))
borel ((λ*f*. ∑*i*∈*I*. *f i*) ∘ (λω. *restrict* (λ*i*. *X i* ω) *I*))
using *I* **by** (*intro indep-var-compose*[*OF indep-var-restrict*[*OF indep*]]) *auto*
also have ((λ*f*. *f i*) ∘ (λω. *restrict* (λ*i*. *X i* ω) {*i*})) = *X i*
by *auto*
also have ((λ*f*. ∑*i*∈*I*. *f i*) ∘ (λω. *restrict* (λ*i*. *X i* ω) *I*)) = (λω. ∑*i*∈*I*. *X i* ω)
by (*auto cong*: *rev-conj-cong*)
finally show *?thesis* **.**
qed

lemma (**in** *prob-space*) *indep-vars-setprod*:
fixes $X :: 'i \Rightarrow 'a \Rightarrow real$
assumes I: *finite* I $i \notin I$ **and** *indep*: *indep-vars* (λ-. *borel*) X (*insert* i I)
shows *indep-var borel* (X i) *borel* ($\lambda\omega$. $\prod i \in I$. X i ω)
proof −
have *indep-var*
borel ((λf. f i) $\circ$ ($\lambda\omega$. *restrict* (λi. X i ω) {i}))
borel ((λf. $\prod i \in I$. f i) $\circ$ ($\lambda\omega$. *restrict* (λi. X i ω) I))
using I **by** (*intro indep-var-compose*[*OF indep-var-restrict*[*OF indep*]]) *auto*
also have ((λf. f i) $\circ$ ($\lambda\omega$. *restrict* (λi. X i ω) {i})) = X i
by *auto*
also have ((λf. $\prod i \in I$. f i) $\circ$ ($\lambda\omega$. *restrict* (λi. X i ω) I)) = ($\lambda\omega$. $\prod i \in I$. X i ω)
by (*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 $\bigwedge i$. $i \in I \Longrightarrow A$ $i \in$ *sets* (M' i)
shows *prob* ($\bigcap i \in I$. X i −‘ A i ∩ *space* M) = ($\prod i \in I$. *prob* (X i −‘ A i ∩ *space* M))
proof (*rule indep-setsD*)
show *indep-sets* (λi. *sigma-sets* (*space* M) {X i −‘ A ∩ *space* M |A. $A \in$ *sets* (M' i)}) I
using X **by** (*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 ∩ *space* $M \in$ *sigma-sets* (*space* M) {X i −‘ A ∩ *space* M |A. $A \in$ *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 $J \subseteq I$ $\bigwedge i$. $i \in J \Longrightarrow A$ $i \in$ *sets* (M' i)
shows *prob* ($\bigcap i \in J$. X i −‘ A i ∩ *space* M) = ($\prod i \in J$. *prob* (X i −‘ A i ∩ *space* M))
proof (*rule indep-setsD*)
show *indep-sets* (λi. *sigma-sets* (*space* M) {X i −‘ A ∩ *space* M |A. $A \in$ *sets* (M' i)}) I
using X **by** (*auto simp*: *indep-vars-def*)
show $\forall i \in J$. X i −‘ A i ∩ *space* $M \in$ *sigma-sets* (*space* M) {X i −‘ A ∩ *space* M |A. $A \in$ *sets* (M' i)}
using I **by** *auto*
qed *fact+*

lemma (**in** *prob-space*) *indep-vars-iff-distr-eq-PiM*:
fixes $I :: 'i$ *set* **and** $X :: 'i \Rightarrow 'a \Rightarrow 'b$
assumes $I \neq \{\}$
assumes *rv*: $\bigwedge i$. *random-variable* (M' i) (X i)

shows *indep-vars M′ X I* ⟷
distr M (Π$_M$ *i*∈*I*. *M′ i*) (λ*x*. λ*i*∈*I*. *X i x*) = (Π$_M$ *i*∈*I*. *distr M* (*M′ i*) (*X i*))
proof −
let *?P* = Π$_M$ *i*∈*I*. *M′ i*
let *?X* = λ*x*. λ*i*∈*I*. *X i x*
let *?D* = *distr M ?P ?X*
have *X*: *random-variable ?P ?X* **by** (*intro measurable-restrict rv*)
interpret *D*: *prob-space ?D* **by** (*intro prob-space-distr X*)

let *?D′* = λ*i*. *distr M* (*M′ i*) (*X i*)
let *?P′* = Π$_M$ *i*∈*I*. *distr M* (*M′ i*) (*X i*)
interpret *D′*: *prob-space ?D′ i* **for** *i* **by** (*intro prob-space-distr rv*)
interpret *P*: *product-prob-space ?D′ I* **..**

show *?thesis*
proof
assume *indep-vars M′ X I*
show *?D* = *?P′*
proof (*rule measure-eqI-generator-eq*)
show *Int-stable* (*prod-algebra I M′*)
by (*rule Int-stable-prod-algebra*)
show *prod-algebra I M′* ⊆ *Pow* (*space ?P*)
using *prod-algebra-sets-into-space* **by** (*simp add*: *space-PiM*)
show *sets ?D* = *sigma-sets* (*space ?P*) (*prod-algebra I M′*)
by (*simp add*: *sets-PiM space-PiM*)
show *sets ?P′* = *sigma-sets* (*space ?P*) (*prod-algebra I M′*)
by (*simp add*: *sets-PiM space-PiM cong*: *prod-algebra-cong*)
let *?A* = λ*i*. Π$_E$ *i*∈*I*. *space* (*M′ i*)
show *range ?A* ⊆ *prod-algebra I M′* (⋃ *i*. *?A i*) = *space* (*Pi*$_M$ *I M′*)
by (*auto simp*: *space-PiM intro*!: *space-in-prod-algebra cong*: *prod-algebra-cong*)
{ fix *i* **show** *emeasure ?D* (Π$_E$ *i*∈*I*. *space* (*M′ i*)) ≠ ∞ **by** *auto* **}**
next
fix *E* **assume** *E*: *E* ∈ *prod-algebra I M′*
from *prod-algebraE*[*OF E*] **guess** *J Y* **. note** *J* = *this*

from *E* **have** *E* ∈ *sets ?P* **by** (*auto simp*: *sets-PiM*)
then have *emeasure ?D E* = *emeasure M* (*?X* −‘ *E* ∩ *space M*)
by (*simp add*: *emeasure-distr X*)
also have *?X* −‘ *E* ∩ *space M* = (⋂ *i*∈*J*. *X i* −‘ *Y i* ∩ *space M*)
using *J* ‹*I* ≠ {}› *measurable-space*[*OF rv*] **by** (*auto simp*: *prod-emb-def PiE-iff split*: *if-split-asm*)
also have *emeasure M* (⋂ *i*∈*J*. *X i* −‘ *Y i* ∩ *space M*) = (∏ *i*∈*J*. *emeasure M* (*X i* −‘ *Y i* ∩ *space M*))
using ‹*indep-vars M′ X I*› *J* ‹*I* ≠ {}› **using** *indep-varsD*[*of M′ X I J*]
by (*auto simp*: *emeasure-eq-measure setprod-ennreal measure-nonneg setprod-nonneg*)
also have ... = (∏ *i*∈*J*. *emeasure* (*?D′ i*) (*Y i*))
using *rv J* **by** (*simp add*: *emeasure-distr*)
also have ... = *emeasure ?P′ E*
using *P.emeasure-PiM-emb*[*of J Y*] *J* **by** (*simp add*: *prod-emb-def*)

```
    finally show emeasure ?D E = 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 ∩ space M |A. A ∈ sets (M' i)}
⊆ events
     by (auto intro!: sets.sigma-sets-subset measurable-sets rv)
   next
    fix J Y' assume J: J ≠ {} J ⊆ I finite J
    assume Y': ∀j∈J. Y' j ∈ sigma-sets (space M) {X j -` A ∩ space M |A.
A ∈ sets (M' j)}
    have ∀j∈J. ∃ Y. Y' j = X j -` Y ∩ space M ∧ Y ∈ sets (M' j)
    proof
     fix j assume j ∈ J
     from Y'[rule-format, OF this] rv[of j]
     show ∃ Y. Y' j = X j -` Y ∩ space M ∧ Y ∈ 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: ⋀j. j ∈ J ⟹ Y' j = X j -` Y j ∩ space M ⋀j. j ∈ J ⟹ Y j ∈ sets
(M' j) by auto
    let ?E = prod-emb I M' J (Pi_E J Y)
    from Y have (⋂j∈J. Y' j) = ?X -` ?E ∩ space M
      using J ‹I ≠ {}› measurable-space[OF rv] by (auto simp: prod-emb-def
PiE-iff split: if-split-asm)
     then have emeasure M (⋂j∈J. Y' j) = emeasure M (?X -` ?E ∩ 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 setprod-ennreal measure-nonneg setprod-nonneg)
   qed
  qed
qed

lemma (in prob-space) indep-varD:
  assumes indep: indep-var Ma A Mb B
```

assumes *sets*: $Xa \in sets\ Ma\ Xb \in sets\ Mb$
shows $prob\ ((\lambda x.\ (A\ x,\ B\ x)) -`\ (Xa \times Xb) \cap space\ M) =$
$prob\ (A -`\ Xa \cap space\ M) * prob\ (B -`\ Xb \cap space\ M)$
proof −
have $prob\ ((\lambda x.\ (A\ x,\ B\ x)) -`\ (Xa \times Xb) \cap space\ M) =$
$prob\ (\bigcap i \in UNIV.\ (case\text{-}bool\ A\ B\ i -`\ case\text{-}bool\ Xa\ Xb\ i \cap space\ M))$
by (*auto intro*!: *arg-cong*[**where** f=*prob*] *simp*: *UNIV-bool*)
also have $\ldots = (\prod i \in UNIV.\ prob\ (case\text{-}bool\ A\ B\ i -`\ case\text{-}bool\ Xa\ Xb\ i \cap space\ M))$
using *indep* **unfolding** *indep-var-def*
by (*rule indep-varsD*) (*auto split*: *bool.split intro*: *sets*)
also have $\ldots = prob\ (A -`\ Xa \cap space\ M) * prob\ (B -`\ Xb \cap space\ M)$
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*]: $A \in sets\ N\ B \in sets\ N$
shows $\mathcal{P}(x\ in\ M.\ X\ x \in A \wedge Y\ x \in B) = \mathcal{P}(x\ in\ M.\ X\ x \in A) * \mathcal{P}(x\ in\ M.\ Y\ x \in B)$
proof−
have $\mathcal{P}(x\ in\ M.\ (X\ x) \in A \wedge\ (Y\ x) \in B\ ) = prob\ ((\lambda x.\ (X\ x,\ Y\ x)) -`\ (A \times B) \cap space\ M)$
by (*auto intro*!: *arg-cong*[**where** f= *prob*])
also have ...= $prob\ (X -`\ A \cap space\ M) * prob\ (Y -`\ B \cap space\ M)$
by (*auto intro*!: *indep-varD*[**where** *Ma*=*N* **and** *Mb*=*N*])
also have ... $= \mathcal{P}(x\ in\ M.\ X\ x \in A) * \mathcal{P}(x\ in\ M.\ Y\ x \in B)$
by (*auto intro*!: *arg-cong2*[**where** f= *op* $*$] *arg-cong*[**where** f= *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 $\forall i \in UNIV.\ random\text{-}variable\ (case\text{-}bool\ S\ T\ i)\ (case\text{-}bool\ X\ Y\ i)$
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*:
$indep\text{-}var\ S\ X\ T\ Y \longleftrightarrow random\text{-}variable\ S\ X \wedge random\text{-}variable\ T\ Y \wedge$
$distr\ M\ S\ X \bigotimes_M distr\ M\ T\ Y = distr\ M\ (S \bigotimes_M T)\ (\lambda x.\ (X\ x,\ Y\ x))$ (**is** - $\longleftrightarrow$ - $\wedge$ - $\wedge$ *?S* $\bigotimes_M$ *?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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$
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 $?S \bigotimes_M ?T = ?J$
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 $\ldots =$ *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 $\ldots =$ *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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$
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 $?S \bigotimes_M ?T = ?J$

{ 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** $A \in$ *sets* S **then show** $X -` A \cap$ *space* $M \in$ *sets* M
using ‹$X \in$ *measurable* M S› **by** (*auto intro*: *measurable-sets*) **}**
{ fix A **assume** $A \in$ *sets* T **then show** $Y -` A \cap$ *space* $M \in$ *sets* M
using ‹$Y \in$ *measurable* M T› **by** (*auto intro*: *measurable-sets*) **}**
next
fix A B **assume** *ab*: $A \in$ *sets* S $B \in$ *sets* T
then have *prob* (($X -` A \cap$ *space* $M$) $\cap$ ($Y -` B \cap$ *space* M)) = *emeasure* *?J* ($A \times B$)
using XY **by** (*auto simp add*: *emeasure-distr emeasure-eq-measure measure-nonneg intro*!: *arg-cong*[**where** f=*prob*])
also have ... = *emeasure* (*?S* $\bigotimes_M$ *?T*) ($A \times B$)
unfolding ‹*?S* $\bigotimes_M$ *?T* = *?J*› **..**
also have ... = *emeasure* *?S* A $*$ *emeasure* *?T* B
using *ab* **by** (*simp add*: Y.*emeasure-pair-measure-Times*)
finally show *prob* (($X -` A \cap$ *space* $M$) $\cap$ ($Y -` B \cap$ *space* M)) =
prob ($X -` A \cap$ *space* $M$) $*$ *prob* ($Y -` B \cap$ *space* M)
using *rvs ab* **by** (*simp add*: *emeasure-eq-measure emeasure-distr measure-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 *Px* **and** Y: *distributed* M T Y *Py*
assumes *indep*: *indep-var* S X T Y
shows *distributed* M ($S \bigotimes_M T$) (λx. (X x, Y x)) ($\lambda(x, y)$. *Px* x $*$ *Py* 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* (λ-. *borel*) X I $\bigwedge i\ \omega$. $i \in I \Longrightarrow 0 \le X\ i\ \omega$
shows ($\int^+ \omega$. ($\prod i \in I$. $X\ i\ \omega$) ∂M) = ($\prod i \in I$. $\int^+ \omega$. $X\ i\ \omega\ \partial M$)
proof *cases*
assume $I \neq \{\}$
def $Y \equiv \lambda i\ \omega$. *if* $i \in I$ *then* $X\ i\ \omega$ *else* 0
{ fix i **have** $i \in I \Longrightarrow$ *random-variable borel* ($X\ i$)
using $I(2)$ **by** (*cases* $i \in I$) (*auto simp*: *indep-vars-def*) **}**
note *rv-X* = *this*

{ 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*]

interpret Y: *prob-space distr* M *borel* ($Y\ i$) **for** i

using *I*(*2*) **by** (*cases i* ∈ *I*) (*auto intro*!: *prob-space-distr simp*: *indep-vars-def prob-space-return*)
interpret *product-sigma-finite* λ*i*. *distr M borel* (*Y i*)
..

have *indep-Y*: *indep-vars* (λ*i*. *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*(*3*) **by** (*auto intro*!: *nn-integral-cong setprod.cong simp add*: *Y-def max-def*)
also have ... = (∫ $^+$ω. (∏ *i*∈*I*. ω *i*) ∂*distr M* (*Pi*$_M$ *I* (λ*i*. *borel*)) (λ*x*. λ*i*∈*I*. *Y i x*))
by (*subst nn-integral-distr*) *auto*
also have ... = (∫ $^+$ω. (∏ *i*∈*I*. ω *i*) ∂*Pi*$_M$ *I* (λ*i*. *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-nn-integral-setprod*) (*auto intro*: ‹*finite I*›)
also have ... = (∏ *i*∈*I*. ∫ $^+$ω. *X i* ω ∂*M*)
by (*intro setprod.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* ⇒ *'a* ⇒ *'b*::{*real-normed-field*, *banach*, *second-countable-topology*}
assumes *I*: *finite I indep-vars* (λ-. *borel*) *X I* ⋀*i*. *i* ∈ *I* ⟹ *integrable M* (*X i*)
shows *indep-vars-lebesgue-integral*: (∫ ω. (∏ *i*∈*I*. *X i* ω) ∂*M*) = (∏ *i*∈*I*. ∫ ω. *X i* ω ∂*M*) (**is** *?eq*)
and *indep-vars-integrable*: *integrable M* (λω. (∏ *i*∈*I*. *X i* ω)) (**is** *?int*)
proof (*induct rule*: *case-split*)
assume *I* ≠ {}
def *Y* ≡ λ*i* ω. *if i* ∈ *I then X i* ω *else 0*
{ fix *i* **have** *i* ∈ *I* ⟹ *random-variable borel* (*X i*)
using *I*(*2*) **by** (*cases i*∈*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*∈*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*∈*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* ∈ *I*) (*auto intro*!: *prob-space-distr simp*: *indep-vars-def prob-space-return*)

interpret *product-sigma-finite* $\lambda i.$ *distr M borel* (*Y i*)
..

have *indep-Y*: *indep-vars* ($\lambda i.$ *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** (*simp add*: *Y-def*)
also have $\ldots = (\int \omega.\ (\prod i{\in}I.\ \omega\ i)\ \partial distr\ M\ (Pi_M\ I\ (\lambda i.\ borel))\ (\lambda x.\ \lambda i{\in}I.\ Y\ i\ x))$
by (*subst integral-distr*) *auto*
also have $\ldots = (\int \omega.\ (\prod i{\in}I.\ \omega\ i)\ \partial Pi_M\ I\ (\lambda i.\ distr\ M\ borel\ (Y\ i)))$
unfolding *indep-vars-iff-distr-eq-PiM*[*THEN iffD1*, *OF* ‹$I \neq \{\}$› *rv-Y indep-Y*]
..
also have $\ldots = (\prod i{\in}I.\ (\int \omega.\ \omega\ \partial distr\ M\ borel\ (Y\ i)))$
by (*rule product-integral-setprod*) (*auto intro*: ‹*finite I*› *simp*: *integrable-distr-eq int-Y*)
also have $\ldots = (\prod i{\in}I.\ \int \omega.\ X\ i\ \omega\ \partial M)$
by (*intro setprod.cong integral-cong*)
(*auto simp*: *integral-distr Y-def rv-X*)
finally show *?eq* **.**

have *integrable* (*distr M* (*Pi*$_M$ *I* ($\lambda i.$ *borel*)) ($\lambda x.\ \lambda i{\in}I.\ Y\ i\ x$)) ($\lambda\omega.\ (\prod i{\in}I.\ \omega\ i)$)
unfolding *indep-vars-iff-distr-eq-PiM*[*THEN iffD1*, *OF* ‹$I \neq \{\}$› *rv-Y indep-Y*]
by (*intro product-integrable-setprod*[*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*: $(\int \omega.\ X1\ \omega * X2\ \omega\ \partial M) = (\int \omega.\ X1\ \omega\ \partial M) * (\int \omega.\ X2\ \omega\ \partial M)$ (**is** *?eq*)
and *indep-var-integrable*: *integrable M* ($\lambda\omega.\ X1\ \omega * X2\ \omega$) (**is** *?int*)
unfolding *indep-var-def*
proof −
have ∗: ($\lambda\omega.\ X1\ \omega * X2\ \omega$) = ($\lambda\omega.\ \prod i{\in}UNIV.\ (case\text{-}bool\ X1\ X2\ i\ \omega)$)
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

39 Convolution Measure

theory *Convolution*
imports *Independent-Family*
begin

lemma (**in** *finite-measure*) *sigma-finite-measure*: *sigma-finite-measure M*
..

definition *convolution* :: ('*a* :: *ordered-euclidean-space*) *measure* ⇒ '*a measure* ⇒ '*a measure* (**infix** ⋆ *50*) **where**
convolution M N = *distr* (*M* ⨂$_M$ *N*) *borel* (λ(*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* ∈ *borel-measurable borel*
shows (∫$^+$*x*. *f x* ∂*convolution M N*) = (∫$^+$*x*. ∫$^+$*y*. *f* (*x* + *y*) ∂*N* ∂*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* ∈ *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* ⋆ *N*) *A* = ∫⁺*x*. (*emeasure N* {*a*. *a* + *x* ∈ *A*}) ∂*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* ∈ *sets borel*
assumes [*simp*]: *finite-measure M finite-measure N*
assumes [*simp*]: *sets N = sets borel sets M = sets borel*
shows *emeasure* (*M* ⋆ *N*) *A* = ∫⁺*x*. ∫⁺*y*. (*indicator A* (*x* + *y*)) ∂*N* ∂*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 = sets borel sets M = sets borel*
shows *finite-measure* (*M* ⋆ *N*)
unfolding *convolution-def*
by (*intro finite-measure-pair-measure finite-measure.finite-measure-distr*) *auto*

lemma *convolution-emeasure-3*:
assumes [*simp*, *measurable*]: *A* ∈ *sets borel*
assumes [*simp*]: *finite-measure M finite-measure N finite-measure L*
assumes [*simp*]: *sets N = sets borel sets M = sets borel sets L = sets borel*
shows *emeasure* (*L* ⋆ (*M* ⋆ *N*)) *A* = ∫⁺*x*. ∫⁺*y*. ∫⁺*z*. *indicator A* (*x* + *y* + *z*) ∂*N* ∂*M* ∂*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* ∈ *sets borel*
assumes [*simp*]: *finite-measure M finite-measure N finite-measure L*
assumes [*measurable-cong*, *simp*]: *sets N = sets borel sets M = sets borel sets L = sets borel*
shows *emeasure* ((*L* ⋆ *M*) ⋆ *N*) *A* = ∫⁺*x*. ∫⁺*y*. ∫⁺*z*. *indicator A* (*x* + *y* + *z*) ∂*N* ∂*M* ∂*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 = sets borel sets M = sets borel*
shows *(M ⋆ N) = (N ⋆ 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 ⋆ N) = sets (N ⋆ M)* **by** *simp*

fix *A* **assume** *A ∈ sets (M ⋆ N)*
then have *1*[*measurable*]:*A ∈ sets borel* **by** *simp*
have *emeasure (M ⋆ N) A = ∫⁺x. ∫⁺y. indicator A (x + y) ∂N ∂M* **by** (*auto intro*!: *convolution-emeasure'*)
also have *... = ∫⁺x. ∫⁺y. (λ(x,y). indicator A (x + y)) (x, y) ∂N ∂M* **by** (*auto intro*!: *nn-integral-cong*)
also have *... = ∫⁺y. ∫⁺x. (λ(x,y). indicator A (x + y)) (x, y) ∂M ∂N* **by** (*rule Fubini*[*symmetric*]) *simp*
also have *... = emeasure (N ⋆ M) A* **by** (*auto intro*!: *nn-integral-cong simp*: *add.commute convolution-emeasure'*)
finally show *emeasure (M ⋆ N) A = emeasure (N ⋆ M) A* **by** *simp*
qed

lemma *convolution-associative*:
assumes [*simp*]: *finite-measure M finite-measure N finite-measure L*
assumes [*simp*]: *sets N = sets borel sets M = sets borel sets L = sets borel*
shows *(L ⋆ (M ⋆ N)) = ((L ⋆ M) ⋆ 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 (λx. X x + Y x) = convolution (distr M borel X) (distr M 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 (λx. X x + Y x) = convolution (distr M lborel X) (distr M 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 :: real ⇒ ennreal*

assumes [*measurable*]: $f \in$ *borel-measurable borel* $g \in$ *borel-measurable borel*
assumes [*simp*]:*finite-measure* (*density lborel f*) *finite-measure* (*density lborel g*)
shows *density lborel f* $\star$ *density lborel g* = *density lborel* ($\lambda x.\ \int^+ y.\ f\ (x - y) * g\ y\ \partial lborel$)
(**is** *?l* = *?r*)
proof (*intro measure-eqI*)
fix *A* **assume** $A \in$ *sets ?l*
then have [*measurable*]: $A \in$ *sets borel*
by *simp*

have ($\int^+ x.\ f\ x * (\int^+ y.\ g\ y * indicator\ A\ (x + y)\ \partial lborel)\ \partial lborel$) =
($\int^+ x.\ (\int^+ y.\ g\ y * (f\ x * indicator\ A\ (x + y))\ \partial lborel)\ \partial lborel$)
proof (*intro nn-integral-cong-AE*, *eventually-elim*)
fix *x*
have *f x* $*$ ($\int^+ y.\ g\ y * indicator\ A\ (x + y)\ \partial lborel$) =
($\int^+ y.\ f\ x * (g\ y * indicator\ A\ (x + y))\ \partial lborel$)
by (*intro nn-integral-cmult*[*symmetric*]) *auto*
then show *f x* $*$ ($\int^+ y.\ g\ y * indicator\ A\ (x + y)\ \partial lborel$) =
($\int^+ y.\ g\ y * (f\ x * indicator\ A\ (x + y))\ \partial lborel$)
by (*simp add*: *ac-simps*)
qed
also have ... = ($\int^+ y.\ (\int^+ x.\ g\ y * (f\ x * indicator\ A\ (x + y))\ \partial lborel)\ \partial lborel$)
by (*intro lborel-pair.Fubini'*) *simp*
also have ... = ($\int^+ y.\ (\int^+ x.\ f\ (x - y) * g\ y * indicator\ A\ x\ \partial lborel)\ \partial lborel$)
proof (*intro nn-integral-cong-AE*, *eventually-elim*)
fix *y*
have ($\int^+ x.\ g\ y * (f\ x * indicator\ A\ (x + y))\ \partial lborel$) =
g y $*$ ($\int^+ x.\ f\ x * indicator\ A\ (x + y)\ \partial lborel$)
by (*intro nn-integral-cmult*) *auto*
also have ... = *g y* $*$ ($\int^+ x.\ f\ (x - y) * indicator\ A\ x\ \partial lborel$)
by (*subst nn-integral-real-affine*[**where** *c=1* **and** *t=*$-y$])
(*auto simp add*: *one-ennreal-def*[*symmetric*])
also have ... = ($\int^+ x.\ g\ y * (f\ (x - y) * indicator\ A\ x)\ \partial lborel$)
by (*intro nn-integral-cmult*[*symmetric*]) *auto*
finally show ($\int^+ x.\ g\ y * (f\ x * indicator\ A\ (x + y))\ \partial lborel$) =
($\int^+ x.\ f\ (x - y) * g\ y * indicator\ A\ x\ \partial lborel$)
by (*simp add*: *ac-simps*)
qed
also have ... = ($\int^+ x.\ (\int^+ y.\ f\ (x - y) * g\ y * indicator\ A\ x\ \partial lborel)\ \partial lborel$)
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 $\Longrightarrow$ *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 :: *real* ⇒ -
assumes *indep*: *indep-var borel X borel Y*
assumes X: *distributed M lborel X f*
assumes Y: *distributed M lborel Y g*
shows *distributed M lborel* (λx. $X\ x + Y\ x$) (λx. $\int^+ y.\ f\ (x - y) * g\ y\ \partial lborel$)
unfolding *distributed-def*
proof *safe*
have *fg*[*measurable*]: $f \in$ *borel-measurable borel* $g \in$ *borel-measurable borel*
using *distributed-borel-measurable*[*OF X*] *distributed-borel-measurable*[*OF Y*]
by *simp-all*

show (λx. $\int^+ xa.\ f\ (x - xa) * g\ xa\ \partial lborel) \in$ *borel-measurable lborel*
by *measurable*

have *distr M borel* (λx. $X\ x + Y\ x$) = (*distr M borel X* ⋆ *distr M borel Y*)
using *distributed-measurable*[*OF X*] *distributed-measurable*[*OF Y*]
by (*intro sum-indep-random-variable*) (*auto simp*: *indep*)
also have ... = (*density lborel f* ⋆ *density lborel g*)
using *distributed-distr-eq-density*[*OF X*] *distributed-distr-eq-density*[*OF Y*]
by (*simp cong*: *distr-cong*)
also have ... = *density lborel* (λx. $\int^+ y.\ f\ (x - y) * g\ y\ \partial lborel$)
proof (*rule convolution-density*)
show *finite-measure* (*density lborel f*)
using X **by** (*rule distributed-finite-measure-density*)
show *finite-measure* (*density lborel g*)
using Y **by** (*rule distributed-finite-measure-density*)
qed *fact*+
finally show *distr M lborel* (λx. $X\ x + Y\ x$) = *density lborel* (λx. $\int^+ y.\ f\ (x - y) * g\ y\ \partial lborel$)
by (*simp cong*: *distr-cong*)
show *random-variable lborel* (λx. $X\ x + Y\ x$)
using *distributed-measurable*[*OF X*] *distributed-measurable*[*OF Y*] **by** *simp*
qed

lemma *prob-space-convolution-density*:
fixes f:: *real* ⇒ -
fixes g:: *real* ⇒ -
assumes [*measurable*]: $f \in$ *borel-measurable borel*
assumes [*measurable*]: $g \in$ *borel-measurable borel*
assumes *gt-0*[*simp*]: $\bigwedge x.\ 0 \le f\ x\ \bigwedge x.\ 0 \le g\ x$
assumes *prob-space* (*density lborel f*) (**is** *prob-space ?F*)
assumes *prob-space* (*density lborel g*) (**is** *prob-space ?G*)
shows *prob-space* (*density lborel* (λx.$\int^+ y.\ f\ (x - y) * g\ y\ \partial lborel$)) (**is** *prob-space ?D*)
proof (*subst convolution-density*[*symmetric*])

interpret *F*: *prob-space ?F* **by** *fact*
show *finite-measure ?F* **by** *unfold-locales*
interpret *G*: *prob-space ?G* **by** *fact*
show *finite-measure ?G* **by** *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

40 Information theory

theory *Information*
imports
Independent-Family
~~/*src*/*HOL*/*Library*/*Convex*
begin

lemma *log-le*: *1* < *a* ⟹ *0* < *x* ⟹ *x* ≤ *y* ⟹ *log a x* ≤ *log a y*
by (*subst log-le-cancel-iff*) *auto*

lemma *log-less*: *1* < *a* ⟹ *0* < *x* ⟹ *x* < *y* ⟹ *log a x* < *log a y*
by (*subst log-less-cancel-iff*) *auto*

lemma *setsum-cartesian-product'*:
(∑ *x*∈*A* × *B*. *f x*) = (∑ *x*∈*A*. *setsum* (λ*y*. *f* (*x*, *y*)) *B*)
unfolding *setsum.cartesian-product* **by** *simp*

lemma *split-pairs*:
((*A*, *B*) = *X*) ⟷ (*fst X* = *A* ∧ *snd X* = *B*) **and**
(*X* = (*A*, *B*)) ⟷ (*fst X* = *A* ∧ *snd X* = *B*) **by** *auto*

40.1 Information theory

locale *information-space* = *prob-space* +
fixes *b* :: *real* **assumes** *b-gt-1*: *1* < *b*

context *information-space*
begin

Introduce some simplification rules for logarithm of base *b*.

lemma *log-neg-const*:
assumes *x* ≤ *0*
shows *log b x* = *log b 0*
proof −
{ fix *u* :: *real*
have *x* ≤ *0* **by** *fact*

also have $0 < exp\ u$
using *exp-gt-zero* **.**
finally have $exp\ u \neq x$
by *auto* **}**
then show $log\ b\ x = log\ b\ 0$
by (*simp add*: *log-def ln-real-def*)
qed

lemma *log-mult-eq*:
$log\ b\ (A * B) = (if\ 0 < A * B\ then\ log\ b\ |A| + log\ b\ |B|\ else\ log\ b\ 0)$
using *log-mult*[*of b* |*A*| |*B*|] *b-gt-1 log-neg-const*[*of A* ∗ *B*]
by (*auto simp*: *zero-less-mult-iff mult-le-0-iff*)

lemma *log-inverse-eq*:
$log\ b\ (inverse\ B) = (if\ 0 < B\ then\ -\ log\ b\ B\ else\ log\ b\ 0)$
using *log-inverse*[*of b B*] *log-neg-const*[*of inverse B*] *b-gt-1* **by** *simp*

lemma *log-divide-eq*:
$log\ b\ (A\ /\ B) = (if\ 0 < A * B\ then\ log\ b\ |A| - log\ b\ |B|\ else\ log\ b\ 0)$
unfolding *divide-inverse log-mult-eq log-inverse-eq abs-inverse*
by (*auto simp*: *zero-less-mult-iff mult-le-0-iff*)

lemmas *log-simps* = *log-mult-eq log-inverse-eq log-divide-eq*

end

40.2 Kullback−Leibler divergence

The Kullback−Leibler divergence is also known as relative entropy or Kullback−Leibler distance.

definition
$entropy\text{-}density\ b\ M\ N = log\ b \circ enn2real \circ RN\text{-}deriv\ M\ N$

definition
$KL\text{-}divergence\ b\ M\ N = integral^L\ N\ (entropy\text{-}density\ b\ M\ N)$

lemma *measurable-entropy-density*[*measurable*]: *entropy-density b M N* ∈ *borel-measurable M*
unfolding *entropy-density-def* **by** *auto*

lemma (**in** *sigma-finite-measure*) *KL-density*:
fixes $f :: 'a \Rightarrow real$
assumes $1 < b$
assumes *f*[*measurable*]: $f \in borel\text{-}measurable\ M$ **and** *nn*: $AE\ x\ in\ M.\ 0 \leq f\ x$
shows $KL\text{-}divergence\ b\ M\ (density\ M\ f) = (\int x.\ f\ x * log\ b\ (f\ x)\ \partial M)$
unfolding *KL-divergence-def*
proof (*subst integral-real-density*)
show [*measurable*]: $entropy\text{-}density\ b\ M\ (density\ M\ (\lambda x.\ ennreal\ (f\ x))) \in borel\text{-}measurable\ M$

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*
show (∫ *x*. *f x* ∗ *entropy-density b M* (*density M* (λ*x*. *ennreal* (*f x*))) *x* ∂*M*) = (∫ *x*. *f x* ∗ *log b* (*f x*) ∂*M*)
apply (*intro integral-cong-AE*)
apply *measurable*
using *eq nn*
apply *eventually-elim*
apply (*auto simp*: *entropy-density-def*)
done
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*) (λ*x*. *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* (λ*x*. *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*: (∫⁺ *x*. *ennreal* (− *D x*) ∂*M*) = *0*
using *D* **by** (*subst nn-integral-0-iff-AE*) (*auto simp*: *ennreal-neg*)

have (∫⁺ *x*. *ennreal* (*D x*) ∂*M*) = *emeasure* (*density M D*) (*space M*)
using *D* **by** (*simp add*: *emeasure-density cong*: *nn-integral-cong*)
then have *D-pos*: (∫⁺ *x*. *ennreal* (*D x*) ∂*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 *integral*^{L} *M D* = *1*
using *D D-pos D-neg* **by** (*simp add*: *real-lebesgue-integral-def*)

have *0* ≤ *1* − *measure M ?D-set*
using *prob-le-1* **by** (*auto simp*: *field-simps*)
also have ... = (∫ *x*. *D x* − *indicator ?D-set x* ∂*M*)
using ‹*integrable M D*› ‹*integral*^{L} *M D* = *1*›
by (*simp add*: *emeasure-eq-measure*)
also have ... < (∫ *x*. *D x* ∗ (*ln b* ∗ *log b* (*D x*)) ∂*M*)
proof (*rule integral-less-AE*)
show *integrable M* (λ*x*. *D x* − *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* (λ*x*. *D x* ∗ (*ln b* ∗ *log b* (*D x*)))
by (*simp add*: *ac-simps*)
next
show *emeasure M* {*x*∈*space M*. *D x* ≠ *1* ∧ *D x* ≠ *0*} ≠ *0*
proof
assume *eq-0*: *emeasure M* {*x*∈*space M*. *D x* ≠ *1* ∧ *D x* ≠ *0*} = *0*
then have *disj*: *AE x in M*. *D x* = *1* ∨ *D x* = *0*
using *D*(*1*) **by** (*auto intro*!: *AE-I*[*OF subset-refl*] *sets.sets-Collect*)

have *emeasure M* {*x*∈*space M*. *D x* = *1*} = (∫⁺ *x*. *indicator* {*x*∈*space M*. *D x* = *1*} *x* ∂*M*)
using *D*(*1*) **by** *auto*
also have ... = (∫⁺ *x*. *ennreal* (*D x*) ∂*M*)
using *disj* **by** (*auto intro*!: *nn-integral-cong-AE simp*: *indicator-def one-ennreal-def*)
finally have *AE x in M*. *D x* = *1*
using *D D-pos* **by** (*intro AE-I-eq-1*) *auto*
then have (∫⁺*x*. *indicator A x*∂*M*) = (∫⁺*x*. *ennreal* (*D x*) ∗ *indicator A x*∂*M*)
by (*intro nn-integral-cong-AE*) (*auto simp*: *one-ennreal-def*[*symmetric*])

```
    also have ... = density M D A
      using ‹A ∈ sets M› D by (simp add: emeasure-density)
    finally show False using ‹A ∈ sets M› ‹emeasure (density M D) A ≠ emeasure
M A› by simp
  qed
  show {x∈space M. D x ≠ 1 ∧ D x ≠ 0} ∈ sets M
    using D(1) by (auto intro: sets.sets-Collect-conj)

  show AE t in M. t ∈ {x∈space M. D x ≠ 1 ∧ D x ≠ 0} ⟶
    D t − indicator ?D-set t ≠ D t * (ln b * log b (D t))
    using D(2)
  proof (eventually-elim, safe)
    fix t assume Dt: t ∈ space M D t ≠ 1 D t ≠ 0 0 ≤ D t
      and eq: D t − indicator ?D-set t = D t * (ln b * log b (D t))

    have D t − 1 = D t − indicator ?D-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-gt-1 ‹D t ≠ 0› ‹0 ≤ D t›
      by (simp add: log-def ln-div less-le)
    finally have ln (1 / D t) = 1 / D t − 1
      using ‹D t ≠ 0› by (auto simp: field-simps)
    from ln-eq-minus-one[OF - this] ‹D t ≠ 0› ‹0 ≤ D t› ‹D t ≠ 1›
    show False by auto
  qed

  show AE t in M. D t − indicator ?D-set t ≤ D t * (ln b * log b (D t))
    using D(2) AE-space
  proof eventually-elim
    fix t assume t ∈ space M 0 ≤ D t
    show D t − indicator ?D-set t ≤ D t * (ln b * log b (D t))
    proof cases
      assume asm: D t ≠ 0
      then have 0 < D t using ‹0 ≤ D t› by auto
      then have 0 < 1 / D t by auto
      have D t − indicator ?D-set t ≤ − D t * (1 / D t − 1)
        using asm ‹t ∈ space M› by (simp add: field-simps)
      also have − D t * (1 / D t − 1) ≤ − D t * ln (1 / D t)
        using ln-le-minus-one ‹0 < 1 / D t› by (intro mult-left-mono-neg) auto
      also have ... = D t * (ln b * log b (D t))
        using ‹0 < D t› b-gt-1
        by (simp-all add: log-def ln-div)
      finally show ?thesis by simp
    qed simp
  qed
qed
also have ... = (∫ x. ln b * (D x * log b (D x)) ∂M)
  by (simp add: ac-simps)
```

```
  also have ... = ln b * (∫ x. D x * log b (D x) ∂M)
    using int by simp
  finally show ?thesis
    using b-gt-1 D by (subst KL-density) (auto simp: zero-less-mult-iff)
qed

lemma (in sigma-finite-measure) KL-same-eq-0: KL-divergence b M M = 0
proof −
  have AE x in M. 1 = RN-deriv M M x
  proof (rule RN-deriv-unique)
    show density M (λx. 1) = M
      apply (auto intro!: measure-eqI emeasure-density)
      apply (subst emeasure-density)
      apply auto
      done
  qed auto
  then have AE x in M. log b (enn2real (RN-deriv M M x)) = 0
    by (elim AE-mp) simp
  from integral-cong-AE[OF - - this]
  have integral^L M (entropy-density b M M) = 0
    by (simp add: entropy-density-def comp-def)
  then show KL-divergence b M M = 0
    unfolding KL-divergence-def
    by auto
qed

lemma (in information-space) KL-eq-0-iff-eq:
  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))
  shows KL-divergence b M (density M D) = 0 ⟷ density M D = M
  using KL-same-eq-0[of b] KL-gt-0[OF assms]
  by (auto simp: less-le)

lemma (in information-space) KL-eq-0-iff-eq-ac:
  fixes D :: 'a ⇒ real
  assumes prob-space N
  assumes ac: absolutely-continuous M N sets N = sets M
  assumes int: integrable N (entropy-density b M N)
  shows KL-divergence b M N = 0 ⟷ N = M
proof −
  interpret N: prob-space N by fact
  have finite-measure N by unfold-locales
  from real-RN-deriv[OF this ac] guess D . note D = this

  have N = density M (RN-deriv M N)
    using ac by (rule density-RN-deriv[symmetric])
  also have ... = 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 (λx. 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 (λx. D x * log b (D x))
   by (subst integrable-real-density[symmetric]) (auto simp: N[symmetric] comp-def)
  with ‹prob-space N› 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 ∈ borel-measurable M AE x in M. 0 ≤ D x
  assumes int: integrable M (λx. D x * log b (D x))
  shows 0 ≤ KL-divergence b M (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 ⇒ real
  assumes 1 < b
  assumes f: f ∈ borel-measurable M AE x in M. 0 ≤ f x prob-space (density M
f)
  assumes g: g ∈ borel-measurable M AE x in M. 0 ≤ g x prob-space (density M
g)
  assumes ac: AE x in M. f x = 0 ⟶ g x = 0
  assumes int: integrable M (λx. g x * log b (g x / f x))
  shows 0 ≤ KL-divergence b (density M f) (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: density (density M f) (λx. g x / f x) = density M g (is ?DD = -)
    using f g ac by (subst density-density-divide) simp-all

  have 0 ≤ KL-divergence b (density M f) (density (density M f) (λx. g x / f x))
  proof (rule Mf.KL-nonneg)
    show prob-space ?DD unfolding eq by fact
    from f g show (λx. g x / f x) ∈ borel-measurable (density M f)
      by auto
    show AE x in density M f. 0 ≤ g x / f x
      using f g by (auto simp: AE-density)
    show integrable (density M f) (λx. g x / f x * log b (g x / f x))
      using ‹1 < b› f g ac
      by (subst integrable-density)
         (auto intro!: integrable-cong-AE[THEN iffD2, OF - - - int] measurable-If)
  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

40.3 Finite Entropy

definition (**in** *information-space*) *finite-entropy* :: *'b measure* ⇒ (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *real*) ⇒ *bool*
where
finite-entropy S X f ⟷
distributed M S X f ∧
integrable S (λ*x*. *f x* ∗ *log b* (*f x*)) ∧
(∀ *x*∈*space S*. *0* ≤ *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* (λ*a*. *measure M* {*x* ∈ *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*))
(λ*x*. *prob* {*xa* ∈ *space M*. *X xa* = *x*} ∗ *log b* (*prob* {*xa* ∈ *space M*. *X xa* = *x*}))
by (*rule integrable-count-space*)
have *d*: *distributed M* (*count-space* (*X* ‘ *space M*)) *X* (λ*x*. *ennreal* (*if x* ∈ *X*‘*space M then prob* {*xa* ∈ *space M*. *X xa* = *x*} *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* (λ*x*. *ennreal* (*prob* {*xa* ∈ *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* ⨂$_M$ *T*) *S fst*)
proof −
interpret *sigma-finite-measure T* **by** *fact*
{ fix *A* **assume** *A*: *A* ∈ *sets S emeasure S A* = *0*
then have *fst* −‘ *A* ∩ *space* (*S* ⨂$_M$ *T*) = *A* × *space T*
by (*auto simp*: *space-pair-measure dest*!: *sets.sets-into-space*)
with *A* **have** *emeasure* (*S* ⨂$_M$ *T*) (*fst* −‘ *A* ∩ *space* (*S* ⨂$_M$ *T*)) = *0*
by (*simp add*: *emeasure-pair-measure-Times*) **}**
then show *?thesis*
unfolding *absolutely-continuous-def*
apply (*auto simp*: *null-sets-distr-iff*)

```
    apply (auto simp: null-sets-def intro!: measurable-sets)
    done
qed

lemma ac-snd:
  assumes sigma-finite-measure T
  shows absolutely-continuous T (distr (S ⨂M T) T snd)
proof −
  interpret sigma-finite-measure T by fact
  { fix A assume A: A ∈ sets T emeasure T A = 0
    then have snd −‘ A ∩ space (S ⨂M T) = space S × A
      by (auto simp: space-pair-measure dest!: sets.sets-into-space)
    with A have emeasure (S ⨂M T) (snd −‘ A ∩ space (S ⨂M T)) = 0
      by (simp add: emeasure-pair-measure-Times) }
  then show ?thesis
    unfolding absolutely-continuous-def
    apply (auto simp: null-sets-distr-iff)
    apply (auto simp: null-sets-def intro!: measurable-sets)
    done
qed

lemma integrable-cong-AE-imp:
  integrable M g ⟹ f ∈ borel-measurable M ⟹ (AE x in M. g x = f x) ⟹
integrable M f
  using integrable-cong-AE[of f M g] by (auto simp: eq-commute)

lemma (in information-space) finite-entropy-integrable:
  finite-entropy S X Px ⟹ integrable S (λx. Px x ∗ log b (Px x))
  unfolding finite-entropy-def by auto

lemma (in information-space) finite-entropy-distributed:
  finite-entropy S X Px ⟹ distributed M S X Px
  unfolding finite-entropy-def by auto

lemma (in information-space) finite-entropy-nn:
  finite-entropy S X Px ⟹ x ∈ space S ⟹ 0 ≤ Px x
  by (auto simp: finite-entropy-def)

lemma (in information-space) finite-entropy-measurable:
  finite-entropy S X Px ⟹ Px ∈ S →M 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 ⇒ real and f :: 'c ⇒ real
  assumes T: T ∈ measurable P Q
  assumes f: finite-entropy P X f
  assumes g: finite-entropy Q Y g
  assumes Y: Y = T ∘ X
```

shows *AE x in P. g (T x) = 0 ⟶ f x = 0*
using *subdensity[OF T, of M X λx. ennreal (f x) Y λx. 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*:
finite-entropy S X Px ⟹ distributed M T Y Py ⟹ (⋀x. x ∈ space T ⟹ 0 ≤ Py x) ⟹
*X = (λx. f (Y x)) ⟹ f ∈ measurable T S ⟹ integrable T (λx. Py x * log b (Px (f x)))*
using *distributed-transform-integrable[of M T Y Py S X Px f λx. log b (Px x)]*
using *distributed-real-measurable[of S Px M X]*
by (*auto simp*: *finite-entropy-def*)

40.4 Mutual Information

definition (**in** *prob-space*)
mutual-information b S T X Y =
KL-divergence b (distr M S X ⨂M distr M T Y) (distr M (S ⨂M T) (λx. (X x, Y x)))

lemma (**in** *information-space*) *mutual-information-indep-vars*:
fixes *S T X Y*
defines *P ≡ distr M S X ⨂M distr M T Y*
defines *Q ≡ distr M (S ⨂M T) (λx. (X x, Y x))*
shows *indep-var S X T Y ⟷*
(*random-variable S X ∧ random-variable T Y ∧*
absolutely-continuous P Q ∧ integrable Q (entropy-density b P Q) ∧
mutual-information b S T X Y = 0)
unfolding *indep-var-distribution-eq*
proof *safe*
assume *rv*[*measurable*]: *random-variable S X random-variable T Y*

interpret *X*: *prob-space distr M S X*
by (*rule prob-space-distr*) *fact*
interpret *Y*: *prob-space distr M T Y*
by (*rule prob-space-distr*) *fact*
interpret *XY*: *pair-prob-space distr M S X distr M T Y* **by** *standard*
interpret *P*: *information-space P b* **unfolding** *P-def* **by** *standard* (*rule b-gt-1*)

interpret *Q*: *prob-space Q* **unfolding** *Q-def*
by (*rule prob-space-distr*) *simp*

{ assume *distr M S X ⨂M distr M T Y = distr M (S ⨂M T) (λx. (X x, Y x))*
then have [*simp*]: *Q = P* **unfolding** *Q-def P-def* **by** *simp*

show *ac*: *absolutely-continuous P Q* **by** (*simp add*: *absolutely-continuous-def*)
then have *ed*: *entropy-density b P Q* ∈ *borel-measurable P*
by *simp*

have *AE x in P. 1* = *RN-deriv P Q x*
proof (*rule P.RN-deriv-unique*)
show *density P* (λ*x. 1*) = *Q*
unfolding ‹*Q* = *P*› **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 *eventually-elim* (*auto simp*: *entropy-density-def*)
then have *integrable P* (*entropy-density b P Q*) ⟷ *integrable Q* (λ*x. 0*::*real*)
using *ed* **unfolding** ‹*Q* = *P*› **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* = (∫ *x. 0* ∂*P*)
unfolding *mutual-information-def KL-divergence-def P-def*[*symmetric*] *Q-def*[*symmetric*]
‹*Q* = *P*›
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* ⨂$_M$ *distr M T Y* = *distr M* (*S* ⨂$_M$ *T*) (λ*x.* (*X x*, *Y x*))
unfolding *P-def Q-def* **.. }**
qed

abbreviation (**in** *information-space*)
mutual-information-Pow (*ℐ′*(- ; -′)) **where**
ℐ(*X* ; *Y*) ≡ *mutual-information b* (*count-space* (*X*‘*space M*)) (*count-space* (*Y*‘*space M*)) *X Y*

lemma (**in** *information-space*)
fixes *Pxy* :: *′b* × *′c* ⇒ *real* **and** *Px* :: *′b* ⇒ *real* **and** *Py* :: *′c* ⇒ *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* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *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* = $integral^L$ $(S \bigotimes_M T)$ *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*: $\bigwedge x.\ x \in space\ S \Longrightarrow 0 \leq Px\ x$
using *Fx* **by** (*auto simp*: *finite-entropy-def*)
have *Py*: *distributed M T Y Py* **and** *Py-nn*: $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \leq Py\ x$
using *Fy* **by** (*auto simp*: *finite-entropy-def*)
have *Pxy*: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*: $\bigwedge x.\ x \in space\ (S \bigotimes_M T) \Longrightarrow 0 \leq Pxy\ x$
$\bigwedge x\ y.\ x \in space\ S \Longrightarrow y \in space\ T \Longrightarrow 0 \leq Pxy\ (x,\ y)$
using *Fxy* **by** (*auto simp*: *finite-entropy-def space-pair-measure*)

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*]: $Pxy \in (S \bigotimes_M T) \rightarrow_M borel$
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* = $S \bigotimes_M T$
let *?D* = *distr M ?P* $(\lambda x.\ (X\ x,\ Y\ x))$

{ fix *A* **assume** $A \in sets\ S$
with *X*[*THEN measurable-space*] *Y*[*THEN measurable-space*]
have *emeasure* (*distr M S X*) *A* = *emeasure ?D* $(A \times space\ T)$
by (*auto simp*: *emeasure-distr intro*!: *arg-cong*[**where** *f*=*emeasure M*]) **}**
note *marginal-eq1* = *this*
{ fix *A* **assume** $A \in sets\ T$
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* $\bigotimes_M$ *distr M T Y* = *density ?P* $(\lambda x.\ ennreal\ (Px\ (fst\ x) * Py\ (snd\ x)))$
unfolding *Px*(*1*)[*THEN distributed-distr-eq-density*] *Py*(*1*)[*THEN distributed-distr-eq-density*]

proof (*subst pair-measure-density*)
show (λ*x*. *ennreal* (*Px x*)) ∈ *borel-measurable S* (λ*y*. *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*) (λ*x*. *ennreal* (*Px* (*fst x*) ∗ *Py* (*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-divergence b* (*density ?P* (λ*x*. *ennreal* (*Px* (*fst x*) ∗ *Py* (*snd x*)))) (*density ?P* (λ*x*. *ennreal* (*Pxy x*)))
unfolding *mutual-information-def distr-eq Pxy*(*1*)[*THEN distributed-distr-eq-density*]
..

from *Px Py* **have** *f*: (λ*x*. *Px* (*fst x*) ∗ *Py* (*snd x*)) ∈ *borel-measurable ?P*
by (*intro borel-measurable-times*) (*auto intro*: *distributed-real-measurable measurable-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 *A*: (*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 *B*: (*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 *eventually-elim auto*

show *?M* = *?R*
unfolding *M f-def* **using** *Pxy-nn Px-nn Py-nn*
by (*intro ST.KL-density-density b-gt-1 f PxPy-nonneg ac*) (*auto simp*: *space-pair-measure*)

have *X*: *X* = *fst* ∘ (λ*x*. (*X x*, *Y x*)) **and** *Y*: *Y* = *snd* ∘ (λ*x*. (*X x*, *Y x*))
by *auto*

have *integrable* (*S* ⨂$_M$ *T*) (λ*x*. *Pxy x* ∗ *log b* (*Pxy x*) − *Pxy x* ∗ *log b* (*Px* (*fst x*)) − *Pxy x* ∗ *log b* (*Py* (*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* ∈ *borel-measurable* (*S* ⨂$_M$ *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 ⨂M T) f
    apply (rule integrable-cong-AE-imp)
    using A B AE-space
    by eventually-elim
     (auto simp: f-def log-divide-eq log-mult-eq field-simps space-pair-measure Px-nn
Py-nn Pxy-nn
                 less-le)

  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 = integral^L (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* = $S \bigotimes_M T$
let *?D* = *distr M ?P* ($\lambda x.\ (X\ x,\ Y\ x)$)

{ fix A **assume** $A \in$ *sets S*
with X[*THEN measurable-space*] Y[*THEN measurable-space*]
have *emeasure* (*distr M S X*) A = *emeasure ?D* ($A \times$ *space T*)
by (*auto simp*: *emeasure-distr intro*!: *arg-cong*[**where** *f*=*emeasure M*]) **}**
note *marginal-eq1* = *this*
{ fix A **assume** $A \in$ *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* $\bigotimes_M$ *distr M T Y* = *density ?P* ($\lambda x.$ *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 ($\lambda x.$ *ennreal* (*Px x*)) $\in$ *borel-measurable S* ($\lambda y.$ *ennreal* (*Py y*)) $\in$ *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* ($S \bigotimes_M T$) ($\lambda(x,\ y).$ *ennreal* (*Px x*) $*$ *ennreal* (*Py y*)) =
density ($S \bigotimes_M T$) ($\lambda x.$ *ennreal* (*Px* (*fst x*) $*$ *Py* (*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-divergence b* (*density ?P* ($\lambda x.$ *ennreal* (*Px* (*fst x*) $*$ *Py* (*snd x*)))) (*density ?P* ($\lambda x.$ *ennreal* (*Pxy x*)))
unfolding *mutual-information-def distr-eq Pxy*(*1*)[*THEN distributed-distr-eq-density*]
..

from *Px Py* **have** *f*: ($\lambda x.$ *Px* (*fst x*) $*$ *Py* (*snd x*)) $\in$ *borel-measurable ?P*
by (*intro borel-measurable-times*) (*auto intro*: *distributed-real-measurable measurable-fst''* *measurable-snd''*)
have *PxPy-nonneg*: *AE x in ?P*. *0* $\leq$ *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* $\longrightarrow$ *Pxy x* = *0*)
by (*rule subdensity-real*[*OF measurable-fst Pxy Px*]) (*insert Px-nn Pxy-nn*, *auto simp*: *space-pair-measure*)
moreover
have (*AE x in ?P*. *Py* (*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 in ?P. Px* (*fst x*) ∗ *Py* (*snd x*) = *0* ⟶ *Pxy x* = *0*
by *eventually-elim 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* ⨂$_M$ *T*) *f*
show *0* ≤ *?M* **unfolding** *M*
proof (*intro ST.KL-density-density-nonneg*)
show *prob-space* (*density* (*S* ⨂$_M$ *T*) (λ*x*. *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*) (λ*x*. *ennreal* (*Px* (*fst x*) ∗ *Py* (*snd x*))))
unfolding *distr-eq*[*symmetric*] **by** *unfold-locales*
show *integrable* (*S* ⨂$_M$ *T*) (λ*x*. *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*[*measurable*]: *distributed M S X Px* **and** *Px-nn*: ⋀*x*. *x* ∈ *space S* ⟹ *0* ≤ *Px x*
and *Py*[*measurable*]: *distributed M T Y Py* **and** *Py-nn*: ⋀*x*. *x* ∈ *space T* ⟹ *0* ≤ *Py x*
and *Pxy*[*measurable*]: *distributed M* (*S* ⨂$_M$ *T*) (λ*x*. (*X x*, *Y x*)) *Pxy*
and *Pxy-nn*: ⋀*x*. *x* ∈ *space* (*S* ⨂$_M$ *T*) ⟹ *0* ≤ *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* ⨂$_M$ *T*. *Px* (*fst x*) = *0* ⟶ *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* ⨂$_M$ *T*. *Py* (*snd x*) = *0* ⟶ *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* $\bigotimes_M$ *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* $\bigotimes_M$ *T. Pxy x * log b (Pxy x / (Px (fst x) * Py (snd x))) = 0*
by *eventually-elim simp*
then have $(\int x.\ Pxy\ x * log\ b\ (Pxy\ x\ /\ (Px\ (fst\ x) * Py\ (snd\ x)))\ \partial(S \bigotimes_M T)) = (\int x.\ 0\ \partial(S \bigotimes_M T))$
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* (λ*x*. (*X x*, *Y x*)) *Pxy*
shows $\mathcal{I}(X\ ;\ Y) = (\sum (x,\ y)\in(\lambda x.\ (X\ x,\ Y\ x))\ `space\ M.\ Pxy\ (x,\ y) * log\ b\ (Pxy\ (x,\ y)\ /\ (Px\ x * Py\ y)))$
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* = λ*x*. (*if x* ∈ (λ*x*. (*X x*, *Y x*)) ‘ *space M then Pxy x else 0*)
let *?f* = λ*x*. *?Pxy x * log b* (*?Pxy x* / (*Px* (*fst x*) * *Py* (*snd x*)))
have $\bigwedge x.$ *?f x = (if x* ∈ (λ*x*. (*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** $(\int\ x.\ ?f\ x\ \partial(count\text{-}space\ (X\ `\ space\ M) \bigotimes_M count\text{-}space\ (Y\ `\ space\ M))) =$
$(\sum (x,\ y)\in(\lambda x.\ (X\ x,\ Y\ x))\ `\ space\ M.\ Pxy\ (x,\ y) * log\ b\ (Pxy\ (x,\ y)\ /\ (Px\ x * Py\ y)))$
by (*auto simp add*: *pair-measure-count-space lebesgue-integral-count-space-finite setsum.If-cases split-beta'*
intro!: *setsum.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* (λ*x*. (*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 $(\sum (x,\ y)\in(\lambda x.\ (X\ x,\ Y\ x))\ `\ space\ M.\ Pxy\ (x,\ y) * log\ b\ (Pxy\ (x,\ y)\ /\ (Px\ x * Py\ y))) =$

```
    (∑ (x, y)∈(λx. (X x, Y x)) ‘ space M. 0)
    by (intro setsum.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
```

40.5 Entropy

```
definition (in prob-space) entropy :: real ⇒ 'b measure ⇒ ('a ⇒ 'b) ⇒ real where
  entropy b S X = − KL-divergence b S (distr M S X)

abbreviation (in information-space)
  entropy-Pow (ℋ'(-')) where
  ℋ(X) ≡ entropy b (count-space (X‘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 −
  note D = distributed-measurable[OF X] distributed-borel-measurable[OF X]
  interpret X: prob-space distr M S X
    using D(1) by (rule prob-space-distr)

  have sf: sigma-finite-measure (distr M S X) by standard
  show ?thesis
    using D
    apply (subst eq-commute)
    apply (intro RN-deriv-unique-sigma-finite)
    apply (auto simp: distributed-distr-eq-density[symmetric, OF X] sf)
    done
qed

lemma (in information-space)
  fixes X :: 'a ⇒ 'b
  assumes X[measurable]: distributed M MX X f and nn: ⋀x. x ∈ space MX ⟹
0 ≤ f x
  shows entropy-distr: entropy b MX X = − (∫ x. f x * log b (f x) ∂MX) (is ?eq)
proof −
  note D = distributed-measurable[OF X] distributed-borel-measurable[OF X]
  note ae = distributed-RN-deriv[OF X]
  note distributed-real-measurable[OF nn X, measurable]

  have ae-eq: AE x in distr M MX X. log b (enn2real (RN-deriv MX (distr M MX
X) x)) =
    log b (f x)
    unfolding distributed-distr-eq-density[OF X]
    apply (subst AE-density)
    using D apply simp
    using ae apply eventually-elim
```

```
    apply auto
    done

  have int-eq: (∫ x. f x * log b (f x) ∂MX) = (∫ x. log b (f x) ∂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 ⇒ real and MX :: 'b measure
   assumes X[measurable]: distributed M MX X Px and Px-nn[simp]: ⋀x. x ∈
space MX ⟹ 0 ≤ Px x
  and fin: emeasure MX {x ∈ space MX. Px x ≠ 0} ≠ top
  and int: integrable MX (λx. − Px x * log b (Px x))
  shows entropy b MX X ≤ log b (measure MX {x ∈ space MX. Px x ≠ 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)

  have − log b (measure MX {x ∈ space MX. Px x ≠ 0}) =
    − log b (∫ x. indicator {x ∈ space MX. Px x ≠ 0} x ∂MX)
    using Px Px-nn fin by (auto simp: measure-def)
  also have − log b (∫ x. indicator {x ∈ space MX. Px x ≠ 0} x ∂MX) = − log
b (∫ x. 1 / Px x ∂distr M MX X)
    unfolding distributed-distr-eq-density[OF X] using Px Px-nn
    apply (intro arg-cong[where f=log b] arg-cong[where f=uminus])
   by (subst integral-density) (auto simp del: integral-indicator intro!: integral-cong)
  also have ... ≤ (∫ x. − log b (1 / Px x) ∂distr M MX X)
  proof (rule X.jensens-inequality[of λx. 1 / Px x {0<..} 0 1 λx. − log b x])
    show AE x in distr M MX X. 1 / Px x ∈ {0<..}
      unfolding distributed-distr-eq-density[OF X]
      using Px by (auto simp: AE-density)
     have [simp]: ⋀x. x ∈ space MX ⟹ ennreal (if Px x = 0 then 0 else 1) =
indicator {x ∈ space MX. Px x ≠ 0} x
      by (auto simp: one-ennreal-def)
    have (∫⁺ x. ennreal (− (if Px x = 0 then 0 else 1)) ∂MX) = (∫⁺ x. 0 ∂MX)
      by (intro nn-integral-cong) (auto simp: ennreal-neg)
    then show integrable (distr M MX X) (λx. 1 / Px x)
      unfolding distributed-distr-eq-density[OF X] using Px
     by (auto simp: nn-integral-density real-integrable-def fin ennreal-neg ennreal-mult[symmetric]
           cong: nn-integral-cong)
    have integrable MX (λx. Px x * log b (1 / Px x)) =
```

```
    integrable MX (λx. − Px x ∗ log b (Px x))
    using Px
    by (intro integrable-cong-AE) (auto simp: log-divide-eq less-le)
  then show integrable (distr M MX X) (λx. − log b (1 / Px x))
    unfolding distributed-distr-eq-density[OF X]
    using Px int
    by (subst integrable-real-density) auto
 qed (auto simp: minus-log-convex[OF b-gt-1])
 also have ... = (∫ x. log b (Px x) ∂distr M MX X)
   unfolding distributed-distr-eq-density[OF X] using Px
   by (intro integral-cong-AE) (auto simp: AE-density log-divide-eq)
 also have ... = − 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 ⇒ real and MX :: 'b measure
 assumes X: distributed M MX X Px and Px-nn[simp]: ⋀x. x ∈ space MX ⟹
0 ≤ Px x
 and fin: finite-measure MX
 and int: integrable MX (λx. − Px x ∗ log b (Px x))
 shows entropy b MX X ≤ log b (measure MX (space MX))
proof −
 note Px = distributed-borel-measurable[OF X]
 interpret finite-measure MX by fact
 have entropy b MX X ≤ log b (measure MX {x ∈ space MX. Px x ≠ 0})
   using int X by (intro entropy-le) auto
 also have ... ≤ log b (measure MX (space MX))
   using Px distributed-imp-emeasure-nonzero[OF X]
   by (intro log-le)
     (auto intro!: finite-measure-mono b-gt-1 less-le[THEN iffD2]
           simp: emeasure-eq-measure cong: conj-cong)
 finally show ?thesis .
qed

lemma (in information-space) entropy-uniform:
  assumes X: distributed M MX X (λx. indicator A x / measure MX A) (is
distributed - - - ?f)
 shows entropy b MX X = log b (measure MX A)
proof (subst entropy-distr[OF X])
 have [simp]: emeasure MX A ≠ ∞
   using uniform-distributed-params[OF X] by (auto simp add: measure-def)
 have eq: (∫ x. indicator A x / measure MX A ∗ log b (indicator A x / measure
MX A) ∂MX) =
   (∫ x. (− log b (measure MX A) / measure MX A) ∗ indicator A x ∂MX)
   using uniform-distributed-params[OF X]
```

by (*intro integral-cong*) (*auto split*: *split-indicator simp*: *log-divide-eq zero-less-measure-iff*)
show $-$ ($\int$ *x*. *indicator A x* / *measure MX A* $*$ *log b* (*indicator A x* / *measure MX A*) *∂MX*) =
log b (*measure MX A*)
unfolding *eq* **using** *uniform-distributed-params*[*OF X*]
by (*subst 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 $\Longrightarrow$ $\mathcal{H}(X) = -$ ($\sum x \in X$'*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* = *count-space* (*X*'*space M*)
have $\mathcal{H}(X) \leq$ *log b* (*measure ?X* {*x* $\in$ *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* $\in$ *space ?X*. *f x* $\neq$ *0*} = *card* (*X* ' *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* = *count-space* (*X*'*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

40.6 Conditional Mutual Information

definition (**in** *prob-space*)
conditional-mutual-information b MX MY MZ X Y Z $\equiv$

mutual-information b MX $(MY \bigotimes_M MZ)$ *X* $(\lambda x.\ (Y\ x,\ Z\ x))$ −
mutual-information b MX MZ X Z

abbreviation (**in** *information-space*)
conditional-mutual-information-Pow $(\mathcal{I}'(\ -\ ;\ -\ |\ -\ '))$ **where**
$\mathcal{I}(X\ ;\ Y\ |\ Z) \equiv$ *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*]: $\bigwedge x.\ x \in$ *space* $S \Longrightarrow 0 \le Px\ x$
assumes *Pz*[*measurable*]: *distributed M P Z Pz*
and *Pz-nn*[*simp*]: $\bigwedge z.\ z \in$ *space* $P \Longrightarrow 0 \le Pz\ z$
assumes *Pyz*[*measurable*]: *distributed M* $(T \bigotimes_M P)$ $(\lambda x.\ (Y\ x,\ Z\ x))$ *Pyz*
and *Pyz-nn*[*simp*]: $\bigwedge y\ z.\ y \in$ *space* $T \Longrightarrow z \in$ *space* $P \Longrightarrow 0 \le Pyz\ (y,\ z)$
assumes *Pxz*[*measurable*]: *distributed M* $(S \bigotimes_M P)$ $(\lambda x.\ (X\ x,\ Z\ x))$ *Pxz*
and *Pxz-nn*[*simp*]: $\bigwedge x\ z.\ x \in$ *space* $S \Longrightarrow z \in$ *space* $P \Longrightarrow 0 \le Pxz\ (x,\ z)$
assumes *Pxyz*[*measurable*]: *distributed M* $(S \bigotimes_M T \bigotimes_M P)$ $(\lambda x.\ (X\ x,\ Y\ x,\ Z\ x))$ *Pxyz*
and *Pxyz-nn*[*simp*]: $\bigwedge x\ y\ z.\ x \in$ *space* $S \Longrightarrow y \in$ *space* $T \Longrightarrow z \in$ *space* $P \Longrightarrow 0 \le Pxyz\ (x,\ y,\ z)$
assumes *I1*: *integrable* $(S \bigotimes_M T \bigotimes_M P)$ $(\lambda(x,\ y,\ z).\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxyz\ (x,\ y,\ z)\ /\ (Px\ x * Pyz\ (y,\ z))))$
assumes *I2*: *integrable* $(S \bigotimes_M T \bigotimes_M P)$ $(\lambda(x,\ y,\ z).\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxz\ (x,\ z)\ /\ (Px\ x * Pz\ z)))$
shows *conditional-mutual-information-generic-eq*: *conditional-mutual-information b S T P X Y Z*
$= (\int (x,\ y,\ z).\ Pxyz\ (x,\ y,\ z) * log\ b\ (Pxyz\ (x,\ y,\ z)\ /\ (Pxz\ (x,\ z) * (Pyz\ (y,z)\ /\ Pz\ z)))\ \partial(S \bigotimes_M T \bigotimes_M P))$ (**is** *?eq*)
and *conditional-mutual-information-generic-nonneg*: $0 \le$ *conditional-mutual-information b S T P X Y Z* (**is** *?nonneg*)
proof −
have [*measurable*]: $Px \in S \to_M$ *borel*
using *Px Px-nn* **by** (*intro distributed-real-measurable*)
have [*measurable*]: $Pz \in P \to_M$ *borel*
using *Pz Pz-nn* **by** (*intro distributed-real-measurable*)
have *measurable-Pyz*[*measurable*]: $Pyz \in (T \bigotimes_M P) \to_M$ *borel*
using *Pyz Pyz-nn* **by** (*intro distributed-real-measurable*) (*auto simp*: *space-pair-measure*)
have *measurable-Pxz*[*measurable*]: $Pxz \in (S \bigotimes_M P) \to_M$ *borel*
using *Pxz Pxz-nn* **by** (*intro distributed-real-measurable*) (*auto simp*: *space-pair-measure*)
have *measurable-Pxyz*[*measurable*]: $Pxyz \in (S \bigotimes_M T \bigotimes_M P) \to_M$ *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* $\bigotimes_M$ *P T* **..**
interpret *STP*: *pair-sigma-finite S T* $\bigotimes_M$ *P* **..**
interpret *TPS*: *pair-sigma-finite T* $\bigotimes_M$ *P S* **..**
have *TP*: *sigma-finite-measure* (*T* $\bigotimes_M$ *P*) **..**
have *SP*: *sigma-finite-measure* (*S* $\bigotimes_M$ *P*) **..**
have *YZ*: *random-variable* (*T* $\bigotimes_M$ *P*) (λ*x*. (*Y x*, *Z x*))
using *Pyz* **by** (*simp add*: *distributed-measurable*)

from *Pxz Pxyz* **have** *distr-eq*: *distr M* (*S* $\bigotimes_M$ *P*) (λ*x*. (*X x*, *Z x*)) =
distr (*distr M* (*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*) (λ*x*. (*X x*, *Y x*, *Z x*))) (*S* $\bigotimes_M$ *P*) (λ(*x*, *y*, *z*). (*x*, *z*))
by (*simp add*: *comp-def distr-distr*)

have *mutual-information b S P X Z* =
(∫ *x*. *Pxz x* ∗ *log b* (*Pxz x* / (*Px* (*fst x*) ∗ *Pz* (*snd x*))) ∂(*S* $\bigotimes_M$ *P*))
by (*rule mutual-information-distr*[*OF S P Px Px-nn Pz Pz-nn Pxz Pxz-nn*])
also have ... = (∫ (*x*,*y*,*z*). *Pxyz* (*x*,*y*,*z*) ∗ *log b* (*Pxz* (*x*,*z*) / (*Px x* ∗ *Pz z*)) ∂(*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*))
using *b-gt-1 Pxz Px Pz*
by (*subst distributed-transform-integral*[*OF Pxyz - Pxz -*, **where** *T*=λ(*x*, *y*, *z*). (*x*, *z*)])
(*auto simp*: *split-beta′ space-pair-measure*)
finally have *mi-eq*:
mutual-information b S P X Z = (∫ (*x*,*y*,*z*). *Pxyz* (*x*,*y*,*z*) ∗ *log b* (*Pxz* (*x*,*z*) / (*Px x* ∗ *Pz z*)) ∂(*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*)) **.**

have *ae1*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Px* (*fst x*) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-real*[*of fst*, *OF - Pxyz Px*]) (*auto simp*: *space-pair-measure*)
moreover have *ae2*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Pz* (*snd* (*snd x*)) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-real*[*of* λ*x*. *snd* (*snd x*), *OF - Pxyz Pz*]) (*auto simp*: *space-pair-measure*)
moreover have *ae3*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Pxz* (*fst x*, *snd* (*snd x*)) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-real*[*of* λ*x*. (*fst x*, *snd* (*snd x*)), *OF - Pxyz Pxz*]) (*auto simp*: *space-pair-measure*)
moreover have *ae4*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Pyz* (*snd x*) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-real*[*of snd*, *OF - Pxyz Pyz*]) (*auto simp*: *space-pair-measure*)
ultimately have *ae*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *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* ≠ *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 integral-diff*[*symmetric*])
apply (*auto intro*!: *integral-cong-AE simp*: *split-beta′ simp del*: *integral-diff*)
done

let *?P* = *density* (*S* ⨂$_M$ *T* ⨂$_M$ *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* ⨂$_M$ *P*) *Pyz*
interpret *Q*: *prob-space ?Q*
unfolding *distributed-distr-eq-density*[*OF Pyz*, *symmetric*]
by (*rule prob-space-distr*) *simp*

let *?f* = λ(*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* ⨂$_M$ *P*. *Pz* (*snd x*) = *0* ⟶ *Pyz x* = *0*
by (*auto simp*: *comp-def space-pair-measure*)
have *aeX2*: *AE x in T* ⨂$_M$ *P*. *0* ≤ *Pz* (*snd x*)
using *Pz* **by** (*intro TP.AE-pair-measure*) (*auto simp*: *comp-def*)

have *aeX3*: *AE y in T* ⨂$_M$ *P*. (∫$^+$ *x*. *ennreal* (*Pxz* (*x*, *snd y*)) ∂*S*) = *ennreal* (*Pz* (*snd y*))
using *Pz distributed-marginal-eq-joint2*[*OF P S Pz Pxz*]
by (*intro TP.AE-pair-measure*) *auto*

have (∫$^+$ *x*. *?f x* ∂*?P*) ≤ (∫$^+$ (*x*, *y*, *z*). *Pxz* (*x*, *z*) ∗ (*Pyz* (*y*, *z*) / *Pz z*) ∂(*S* ⨂$_M$ *T* ⨂$_M$ *P*))
by (*subst nn-integral-density*)
(*auto intro*!: *nn-integral-mono simp*: *space-pair-measure ennreal-mult*[*symmetric*])
also have ... = (∫$^+$(*y*, *z*). (∫$^+$ *x*. *ennreal* (*Pxz* (*x*, *z*)) ∗ *ennreal* (*Pyz* (*y*, *z*)

/ Pz z) ∂S) ∂(T ⨂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 ... = (∫⁺x. *ennreal* (*Pyz x*) * *1* ∂*T* ⨂M *P*)
apply (*rule nn-integral-cong-AE*)
using *aeX1 aeX2 aeX3 AE-space*
apply *eventually-elim*
proof (*case-tac x*, *simp add*: *space-pair-measure*)
fix *a b* **assume** *Pz b = 0 ⟶ Pyz* (*a*, *b*) = *0 a* ∈ *space T* ∧ *b* ∈ *space P*
(∫⁺ *x*. *ennreal* (*Pxz* (*x*, *b*)) ∂*S*) = *ennreal* (*Pz b*)
then show (∫⁺ *x*. *ennreal* (*Pxz* (*x*, *b*)) * *ennreal* (*Pyz* (*a*, *b*) / *Pz b*) ∂*S*) = *ennreal* (*Pyz* (*a*, *b*))
by (*subst nn-integral-multc*) (*auto split*: *prod.split simp*: *ennreal-mult*[*symmetric*])
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*: (∫⁺ *x*. *?f x* ∂*?P*) ≤ *1* **.**
also have ... < ∞ **by** *simp*
finally have *fin*: (∫⁺ *x*. *?f x* ∂*?P*) ≠ ∞ **by** *simp*

have *pos*: (∫⁺*x*. *?f x* ∂*?P*) ≠ *0*
apply (*subst nn-integral-density*)
apply (*simp-all add*: *split-beta′*)
proof
let *?g* = λ*x*. *ennreal* (*Pxyz x*) * (*Pxz* (*fst x*, *snd* (*snd x*)) * *Pyz* (*snd x*) / (*Pz* (*snd* (*snd x*)) * *Pxyz x*))
assume (∫⁺*x*. *?g x* ∂(*S* ⨂M *T* ⨂M *P*)) = *0*
then have *AE x in S* ⨂M *T* ⨂M *P*. *?g x* = *0*
by (*intro nn-integral-0-iff-AE*[*THEN iffD1*]) *auto*
then have *AE x in S* ⨂M *T* ⨂M *P*. *Pxyz x* = *0*
using *ae1 ae2 ae3 ae4 AE-space*
by *eventually-elim* (*auto split*: *if-split-asm simp*: *mult-le-0-iff divide-le-0-iff space-pair-measure*)
then have (∫⁺ *x*. *ennreal* (*Pxyz x*) ∂*S* ⨂M *T* ⨂M *P*) = *0*
by (*subst nn-integral-cong-AE*[*of* - λ*x*. *0*]) *auto*
with *P.emeasure-space-1* **show** *False*
by (*subst* (*asm*) *emeasure-density*) (*auto cong*: *nn-integral-cong*)
qed

have *neg*: (∫⁺ *x*. − *?f x* ∂*?P*) = *0*
apply (*rule nn-integral-0-iff-AE*[*THEN iffD2*])
apply *simp*
apply (*subst AE-density*)
apply (*auto simp*: *space-pair-measure ennreal-neg*)
done

have *I3*: *integrable* (*S* ⨂M *T* ⨂M *P*) (λ(*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 - - - integrable-diff[OF I1
I2]])
   using ae
   apply (auto simp: split-beta')
   done

 have - log b 1 ≤ - log b (integral^L ?P ?f)
 proof (intro le-imp-neg-le log-le[OF b-gt-1])
   have If: integrable ?P ?f
     unfolding real-integrable-def
   proof (intro conjI)
     from neg show (∫⁺ x. - ?f x ∂?P) ≠ ∞
       by simp
     from fin show (∫⁺ x. ?f x ∂?P) ≠ ∞
       by simp
   qed simp
   then have (∫⁺ x. ?f x ∂?P) = (∫ x. ?f x ∂?P)
     apply (rule nn-integral-eq-integral)
     apply (subst AE-density)
     apply simp
     apply (auto simp: space-pair-measure ennreal-neg)
     done
   with pos le1
   show 0 < (∫ x. ?f x ∂?P) (∫ x. ?f x ∂?P) ≤ 1
     by (simp-all add: one-ennreal.rep-eq zero-less-iff-neq-zero[symmetric])
 qed
 also have - log b (integral^L ?P ?f) ≤ (∫ x. - log b (?f x) ∂?P)
 proof (rule P.jensens-inequality[where a=0 and b=1 and I={0<..}])
   show AE x in ?P. ?f x ∈ {0<..}
     unfolding AE-density[OF distributed-borel-measurable[OF Pxyz]]
     using ae1 ae2 ae3 ae4 AE-space
     by eventually-elim (auto simp: space-pair-measure less-le)
   show integrable ?P ?f
     unfolding real-integrable-def
     using fin neg by (auto simp: split-beta')
   show integrable ?P (λx. - log b (?f x))
     apply (subst integrable-real-density)
     apply simp
     apply (auto simp: space-pair-measure) []
     apply simp
     apply (rule integrable-cong-AE[THEN iffD1, OF - - - I3])
     apply simp
     apply simp
     using ae1 ae2 ae3 ae4 AE-space
     apply eventually-elim
      apply (auto simp: log-divide-eq log-mult-eq zero-le-mult-iff zero-less-mult-iff
zero-less-divide-iff field-simps
       less-le space-pair-measure)
```

```
    done
  qed (auto simp: b-gt-1 minus-log-convex)
  also have ... = conditional-mutual-information b S T P X Y Z
    unfolding ‹?eq›
    apply (subst integral-real-density)
    apply simp
    apply (auto simp: space-pair-measure) []
    apply simp
    apply (intro integral-cong-AE)
    using ae1 ae2 ae3 ae4
    apply (auto simp: log-divide-eq zero-less-mult-iff zero-less-divide-iff field-simps
      space-pair-measure less-le)
    done
  finally show ?nonneg
    by simp
qed

lemma (in information-space)
  fixes Px :: - ⇒ 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 ⨂M P) (λx. (Y x, Z x)) Pyz
  assumes Fxz: finite-entropy (S ⨂M P) (λx. (X x, Z x)) Pxz
  assumes Fxyz: finite-entropy (S ⨂M T ⨂M P) (λx. (X x, Y x, Z x)) Pxyz
 shows conditional-mutual-information-generic-eq': conditional-mutual-information
b S T P X Y Z
    = (∫ (x, y, z). Pxyz (x, y, z) * log b (Pxyz (x, y, z) / (Pxz (x, z) * (Pyz (y,z)
/ Pz z))) ∂(S ⨂M T ⨂M P)) (is ?eq)
   and conditional-mutual-information-generic-nonneg': 0 ≤ conditional-mutual-information
b S T P X Y Z (is ?nonneg)
proof −
  note Px = Fx[THEN finite-entropy-distributed, measurable]
  note Pz = Fz[THEN finite-entropy-distributed, measurable]
  note Pyz = Fyz[THEN finite-entropy-distributed, measurable]
  note Pxz = Fxz[THEN finite-entropy-distributed, measurable]
  note Pxyz = Fxyz[THEN finite-entropy-distributed, measurable]

  note Px-nn = Fx[THEN finite-entropy-nn]
  note Pz-nn = Fz[THEN finite-entropy-nn]
  note Pyz-nn = Fyz[THEN finite-entropy-nn]
  note Pxz-nn = Fxz[THEN finite-entropy-nn]
  note Pxyz-nn = Fxyz[THEN finite-entropy-nn]

  note Px' = Fx[THEN finite-entropy-measurable, measurable]
  note Pz' = Fz[THEN finite-entropy-measurable, measurable]
  note Pyz' = Fyz[THEN finite-entropy-measurable, measurable]
  note Pxz' = Fxz[THEN finite-entropy-measurable, measurable]
```

note *Pxyz'* = *Fxyz*[*THEN finite-entropy-measurable*, *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* $\bigotimes_M$ *P T* **..**
interpret *STP*: *pair-sigma-finite S T* $\bigotimes_M$ *P* **..**
interpret *TPS*: *pair-sigma-finite T* $\bigotimes_M$ *P S* **..**
have *TP*: *sigma-finite-measure* (*T* $\bigotimes_M$ *P*) **..**
have *SP*: *sigma-finite-measure* (*S* $\bigotimes_M$ *P*) **..**

from *Pxz Pxyz* **have** *distr-eq*: *distr M* (*S* $\bigotimes_M$ *P*) (λ*x*. (*X x*, *Z x*)) =
distr (*distr M* (*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*) (λ*x*. (*X x*, *Y x*, *Z x*))) (*S* $\bigotimes_M$ *P*) (λ(*x*, *y*, *z*). (*x*, *z*))
by (*simp add*: *distr-distr comp-def*)

have *mutual-information b S P X Z* =
(∫ *x*. *Pxz x* ∗ *log b* (*Pxz x* / (*Px* (*fst x*) ∗ *Pz* (*snd x*))) ∂(*S* $\bigotimes_M$ *P*))
using *Px Px-nn Pz Pz-nn Pxz Pxz-nn*
by (*rule mutual-information-distr*[*OF S P*]) (*auto simp*: *space-pair-measure*)
also have ... = (∫ (*x*,*y*,*z*). *Pxyz* (*x*,*y*,*z*) ∗ *log b* (*Pxz* (*x*,*z*) / (*Px x* ∗ *Pz z*)) ∂(*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*))
using *b-gt-1 Pxz Pxz-nn Pxyz Pxyz-nn*
by (*subst distributed-transform-integral*[*OF Pxyz - Pxz*, **where** *T*=λ(*x*, *y*, *z*). (*x*, *z*)])
(*auto simp*: *split-beta'*)
finally have *mi-eq*:
mutual-information b S P X Z = (∫ (*x*,*y*,*z*). *Pxyz* (*x*,*y*,*z*) ∗ *log b* (*Pxz* (*x*,*z*) / (*Px x* ∗ *Pz z*)) ∂(*S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*)) **.**

have *ae1*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Px* (*fst x*) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-finite-entropy*[*of fst*, *OF - Fxyz Fx*]) *auto*
moreover have *ae2*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Pz* (*snd* (*snd x*)) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-finite-entropy*[*of* λ*x*. *snd* (*snd x*), *OF - Fxyz Fz*]) *auto*
moreover have *ae3*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Pxz* (*fst x*, *snd* (*snd x*)) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-finite-entropy*[*of* λ*x*. (*fst x*, *snd* (*snd x*)), *OF - Fxyz Fxz*]) *auto*
moreover have *ae4*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *P*. *Pyz* (*snd x*) = *0* ⟶ *Pxyz x* = *0*
by (*intro subdensity-finite-entropy*[*of snd*, *OF - Fxyz Fyz*]) *auto*
ultimately have *ae*: *AE x in S* $\bigotimes_M$ *T* $\bigotimes_M$ *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* ≠ *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* ⨂$_M$ *T* ⨂$_M$ *P*)
(λ*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 (λ(*x*, *y*, *z*). *Pxyz* (*x*, *y*, *z*) ∗ *log b* (*Pxyz* (*x*, *y*, *z*) / (*Px x* ∗ *Pyz* (*y*, *z*)))) ∈ *borel-measurable* (*S* ⨂$_M$ *T* ⨂$_M$ *P*)
using *Pxyz Px Pyz* **by** *simp*
ultimately have *I1*: *integrable* (*S* ⨂$_M$ *T* ⨂$_M$ *P*) (λ(*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 AE-space*
by *eventually-elim*
(*insert Px-nn Pyz-nn Pxyz-nn*,
auto simp: *log-divide-eq log-mult-eq field-simps zero-less-mult-iff space-pair-measure less-le*)

have *integrable* (*S* ⨂$_M$ *T* ⨂$_M$ *P*)
(λ*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-nn*, *of* λ*x*. (*fst x*, *snd* (*snd x*))]
using *finite-entropy-integrable-transform*[*OF Fx Pxyz Pxyz-nn*, *of fst*]
using *finite-entropy-integrable-transform*[*OF Fz Pxyz Pxyz-nn*, *of snd* ∘ *snd*]
by *simp*
moreover have (λ(*x*, *y*, *z*). *Pxyz* (*x*, *y*, *z*) ∗ *log b* (*Pxz* (*x*, *z*) / (*Px x* ∗ *Pz z*))) ∈ *borel-measurable* (*S* ⨂$_M$ *T* ⨂$_M$ *P*)
using *Pxyz Px Pz*
by *auto*
ultimately have *I2*: *integrable* (*S* ⨂$_M$ *T* ⨂$_M$ *P*) (λ(*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 AE-space
    by eventually-elim
       (insert Px-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn,
            auto simp: log-divide-eq log-mult-eq field-simps zero-less-mult-iff less-le
space-pair-measure)

  from ae 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 Pyz-nn Pxyz
Pxyz-nn])
    apply simp
    apply simp
    apply (simp add: space-pair-measure)
    apply (subst integral-diff[symmetric])
    apply (auto intro!: integral-cong-AE simp: split-beta' simp del: integral-diff)
    done

  let ?P = density (S ⨂M T ⨂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 ⨂M P) Pyz
  interpret Q: prob-space ?Q
   unfolding distributed-distr-eq-density[OF Pyz, symmetric] by (rule prob-space-distr)
simp

  let ?f = λ(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 ⨂M P. Pz (snd x) = 0 ⟶ Pyz x = 0 by (auto simp:
comp-def)
  have aeX2: AE x in T ⨂M P. 0 ≤ Pz (snd x)
    using Pz by (intro TP.AE-pair-measure) (auto intro: Pz-nn)

  have aeX3: AE y in T ⨂M P. (∫+ x. ennreal (Pxz (x, snd y)) ∂S) = ennreal
(Pz (snd y))
    using Pz distributed-marginal-eq-joint2[OF P S Pz Pxz]
    by (intro TP.AE-pair-measure) (auto )
  have (∫+ x. ?f x ∂?P) ≤ (∫+ (x, y, z). Pxz (x, z) * (Pyz (y, z) / Pz z) ∂(S
⨂M T ⨂M P))
    using Px-nn Pz-nn Pxz-nn Pyz-nn Pxyz-nn
    by (subst nn-integral-density)
     (auto intro!: nn-integral-mono simp: space-pair-measure ennreal-mult[symmetric])
  also have ... = (∫+(y, z). ∫+ x. ennreal (Pxz (x, z)) * ennreal (Pyz (y, z) /
Pz z) ∂S ∂T ⨂M P)
```

using *Px-nn Pz-nn Pxz-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 ... = $(\int^+ x.\ ennreal\ (Pyz\ x) * 1\ \partial T \bigotimes_M P)$
apply (*rule nn-integral-cong-AE*)
using *aeX1 aeX2 aeX3 AE-space*
apply *eventually-elim*
proof (*case-tac x*, *simp add*: *space-pair-measure*)
fix *a b* **assume** $Pz\ b = 0 \longrightarrow Pyz\ (a, b) = 0\ 0 \le Pz\ b\ a \in space\ T \wedge b \in space\ P$
$(\int^+ x.\ ennreal\ (Pxz\ (x, b))\ \partial S) = ennreal\ (Pz\ b)$
then show $(\int^+ x.\ ennreal\ (Pxz\ (x, b)) * ennreal\ (Pyz\ (a, b) / Pz\ b)\ \partial S) = ennreal\ (Pyz\ (a, b))$
using *Pyz-nn*[*of* (*a*,*b*)]
by (*subst nn-integral-multc*) (*auto simp*: *space-pair-measure ennreal-mult*[*symmetric*])
qed
also have ... = *1*
using *Q.emeasure-space-1 Pyz-nn distributed-distr-eq-density*[*OF Pyz*]
by (*subst nn-integral-density*[*symmetric*]) *auto*
finally have *le1*: $(\int^+ x.\ ?f\ x\ \partial ?P) \le 1$ **.**
also have ... $< \infty$ **by** *simp*
finally have *fin*: $(\int^+ x.\ ?f\ x\ \partial ?P) \neq \infty$ **by** *simp*

have $(\int^+ x.\ ?f\ x\ \partial ?P) \neq 0$
using *Pxyz-nn*
apply (*subst nn-integral-density*)
apply (*simp-all add*: *split-beta′ ennreal-mult′*[*symmetric*] *cong*: *nn-integral-cong*)
proof
let $?g = \lambda x.\ ennreal\ (if\ Pxyz\ x = 0\ then\ 0\ else\ Pxz\ (fst\ x, snd\ (snd\ x)) * Pyz\ (snd\ x) / Pz\ (snd\ (snd\ x)))$
assume $(\int^+ x.\ ?g\ x\ \partial(S \bigotimes_M T \bigotimes_M P)) = 0$
then have *AE x in* $S \bigotimes_M T \bigotimes_M P.\ ?g\ x = 0$
by (*intro nn-integral-0-iff-AE*[*THEN iffD1*]) *auto*
then have *AE x in* $S \bigotimes_M T \bigotimes_M P.\ Pxyz\ x = 0$
using *ae1 ae2 ae3 ae4 AE-space*
by *eventually-elim*
(*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 $(\int^+ x.\ ennreal\ (Pxyz\ x)\ \partial S \bigotimes_M T \bigotimes_M P) = 0$
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 ennreal-neg*)

have *I3*: *integrable* ($S \bigotimes_M T \bigotimes_M P$) ($\lambda(x, y, z)$. *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 - - - integrable-diff*[*OF I1 I2*]])
using *ae*
apply (*auto simp*: *split-beta'*)
done

have $-$ *log b 1* ≤ $-$ *log b* (*integral*^{L} *?P ?f*)
proof (*intro le-imp-neg-le log-le*[*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 $(\int^+ x. ?f\ x\ \partial ?P) = (\int x. ?f\ x\ \partial ?P)$
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* (*integral*^{L} *?P ?f*) ≤ $(\int x. - log\ b\ (?f\ x)\ \partial ?P)$
proof (*rule P.jensens-inequality*[**where** *a=0* **and** *b=1* **and** *I*={*0*<..}])
show *AE x in ?P*. *?f x* ∈ {*0*<..}
unfolding *AE-density*[*OF distributed-borel-measurable*[*OF Pxyz*]]
using *ae1 ae2 ae3 ae4 AE-space*
by *eventually-elim* (*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'*)
show *integrable ?P* ($\lambda x. - log\ b\ (?f\ x)$)
using *Pz-nn Pxz-nn Pyz-nn Pxyz-nn*
apply (*subst integrable-real-density*)
apply *simp*
apply *simp*
apply *simp*
apply (*rule integrable-cong-AE*[*THEN iffD1*, *OF - - - I3*])
apply *simp*
apply *simp*
using *ae1 ae2 ae3 ae4 AE-space*

```
    apply eventually-elim
    apply (auto simp: log-divide-eq log-mult-eq zero-le-mult-iff zero-less-mult-iff
                zero-less-divide-iff field-simps space-pair-measure less-le)
    done
  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 AE-space
    apply (auto simp: log-divide-eq zero-less-mult-iff zero-less-divide-iff
                field-simps space-pair-measure less-le)
    done
  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 (λx. (Y x, Z x)) Pyz
  assumes Pxz: simple-distributed M (λx. (X x, Z x)) Pxz
  assumes Pxyz: simple-distributed M (λx. (X x, Y x, Z x)) Pxyz
  shows 𝒥(X ; Y | Z) =
   (∑(x, y, z)∈(λx. (X x, Y x, Z x))‘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) ⨂M count-space (Y ‘ space M) ⨂M count-space
(Z ‘ space M) =
      count-space (X‘space M × Y‘space M × Z‘space M)
    (is ?P = ?C)
    by (simp add: pair-measure-count-space)

  let ?Px = λx. measure M (X -‘ {x} ∩ space M)
   have (λx. (X x, Z x)) ∈ measurable M (count-space (X ‘ space M) ⨂M
count-space (Z ‘ 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* = (λ*x*. *if x* ∈ (λ*x*. (*X x*, *Y x*, *Z x*)) ' *space M then Pxyz x else 0*)
let *?g* = (λ*x*. *if x* ∈ (λ*x*. (*Y x*, *Z x*)) ' *space M then Pyz x else 0*)
let *?h* = (λ*x*. *if x* ∈ (λ*x*. (*X x*, *Z x*)) ' *space M then Pxz 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* = λ*x y z*. *?f* (*x*, *y*, *z*) ∗ *log b* (*?f* (*x*, *y*, *z*) / (*?h* (*x*, *z*) ∗ (*?g* (*y*, *z*) / *Pz z*)))
let *?j* = λ*x y z*. *Pxyz* (*x*, *y*, *z*) ∗ *log b* (*Pxyz* (*x*, *y*, *z*) / (*Pxz* (*x*, *z*) ∗ (*Pyz* (*y*,*z*) / *Pz z*)))
have (λ(*x*, *y*, *z*). *?i x y z*) = (λ*x*. *if x* ∈ (λ*x*. (*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*)∈(λ*x*. (*X x*, *Y x*, *Z x*)) ' *space M*. *?j x y z*)
by (*auto intro*!: *setsum.cong simp add*: ‹*?P* = *?C*› *lebesgue-integral-count-space-finite simple-distributed-finite setsum.If-cases split-beta'*)
qed (*insert Pz Pyz Pxz 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* ≤ *I*(*X* ; *Y* | *Z*)
proof −
have [*simp*]: *count-space* (*X ' space M*) ⨂$_M$ *count-space* (*Y ' space M*) ⨂$_M$ *count-space* (*Z ' space M*) =
count-space (*X'space M* × *Y'space M* × *Z'space M*)
by (*simp add*: *pair-measure-count-space X Y Z simple-functionD*)
note *sf* = *sigma-finite-measure-count-space-finite*[*OF simple-functionD*(*1*)]
note *sd* = *simple-distributedI*[*OF - - refl*]
note *sp* = *simple-function-Pair*
show *?thesis*
apply (*rule conditional-mutual-information-generic-nonneg*[*OF sf*[*OF X*] *sf*[*OF Y*] *sf*[*OF Z*]])
apply (*rule simple-distributed*[*OF sd*[*OF X*]])
apply *simp*

apply *simp*
apply (*rule simple-distributed*[*OF sd*[*OF Z*]])
apply *simp*
apply *simp*
apply (*rule simple-distributed-joint*[*OF sd*[*OF sp*[*OF Y Z*]]])
apply *simp*
apply *simp*
apply (*rule simple-distributed-joint*[*OF sd*[*OF sp*[*OF X Z*]]])
apply *simp*
apply *simp*
apply (*rule simple-distributed-joint2*[*OF sd*[*OF sp*[*OF X sp*[*OF Y Z*]]]])
apply *simp*
apply *simp*
apply (*auto intro*!: *integrable-count-space simp*: *X Y Z simple-functionD*)
done
qed

40.7 Conditional Entropy

definition (**in** *prob-space*)
conditional-entropy b S T X Y = − (∫ (*x*, *y*). *log b* (*enn2real* (*RN-deriv* (*S* ⨂$_M$ *T*) (*distr M* (*S* ⨂$_M$ *T*) (λ*x*. (*X x*, *Y x*))) (*x*, *y*)) /
enn2real (*RN-deriv T* (*distr M T Y*) *y*)) ∂*distr M* (*S* ⨂$_M$ *T*) (λ*x*. (*X x*, *Y x*)))

abbreviation (**in** *information-space*)
conditional-entropy-Pow (ℋ′(- | -′)) **where**
ℋ(*X* | *Y*) ≡ *conditional-entropy b* (*count-space* (*X*‘*space M*)) (*count-space* (*Y*‘*space M*)) *X Y*

lemma (**in** *information-space*) *conditional-entropy-generic-eq*:
fixes *Pxy* :: - ⇒ *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*) (λ*x*. (*X x*, *Y x*)) *Pxy*
and *Pxy-nn*[*simp*]: ⋀*x y*. *x* ∈ *space S* ⟹ *y* ∈ *space T* ⟹ 0 ≤ *Pxy* (*x*, *y*)
shows *conditional-entropy b S T X Y* = − (∫ (*x*, *y*). *Pxy* (*x*, *y*) ∗ *log b* (*Pxy* (*x*, *y*) / *Py y*) ∂(*S* ⨂$_M$ *T*))
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 *AE x in density* $(S \bigotimes_M T)$ $(\lambda x.\ ennreal\ (Pxy\ x))$. $Pxy\ x = enn2real$ $(RN\text{-}deriv\ (S \bigotimes_M T)\ (distr\ M\ (S \bigotimes_M T)\ (\lambda x.\ (X\ x,\ Y\ x)))\ x)$
unfolding *AE-density*[*OF distributed-borel-measurable*, *OF Pxy*]
unfolding *distributed-distr-eq-density*[*OF Pxy*]
using *distributed-RN-deriv*[*OF Pxy*]
by *auto*
moreover
have *AE x in density* $(S \bigotimes_M T)$ $(\lambda x.\ ennreal\ (Pxy\ x))$. $Py\ (snd\ x) = enn2real$ $(RN\text{-}deriv\ T\ (distr\ M\ T\ Y)\ (snd\ x))$
unfolding *AE-density*[*OF distributed-borel-measurable*, *OF Pxy*]
unfolding *distributed-distr-eq-density*[*OF Py*]
apply (*rule ST.AE-pair-measure*)
apply *auto*
using *distributed-RN-deriv*[*OF Py*]
apply *auto*
done
ultimately
have *conditional-entropy b S T X Y* $= -(\int x.\ Pxy\ x * log\ b\ (Pxy\ x\ /\ Py\ (snd\ x))\ \partial(S \bigotimes_M T))$
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 $Px :: 'b \Rightarrow real$ **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-nn*[*simp*]: $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \le Py\ x$
assumes *Pxy*[*measurable*]: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*[*simp*]: $\bigwedge x\ y.\ x \in space\ S \Longrightarrow y \in space\ T \Longrightarrow 0 \le Pxy\ (x,\ y)$
assumes *I1*: *integrable* $(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Pxy\ x))$
assumes *I2*: *integrable* $(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Py\ (snd\ x)))$
shows *conditional-entropy b S T X Y* $=$ *entropy b* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ $-$ *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*]: $Py \in T \to_M borel$
using *Py Py-nn* **by** (*intro distributed-real-measurable*)
have *measurable-Pxy*[*measurable*]: $Pxy \in (S \bigotimes_M T) \to_M borel$
using *Pxy Pxy-nn* **by** (*intro distributed-real-measurable*) (*auto simp*: *space-pair-measure*)

have *entropy b T Y* $= -(\int y.\ Py\ y * log\ b\ (Py\ y)\ \partial T)$

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*!: *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* ⨂$_M$ *T*. *Py* (*snd x*) = 0 ⟶ *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* ⨂$_M$ *T*. 0 ≤ *Py* (*snd x*)
using *Py* **by** (*intro ST.AE-pair-measure*) (*auto simp*: *comp-def intro*!: *measurable-snd′*)
ultimately have *AE x in S* ⨂$_M$ *T*. 0 ≤ *Pxy x* ∧ 0 ≤ *Py* (*snd x*) ∧
(*Pxy x* = 0 ∨ (*Pxy x* ≠ 0 ⟶ 0 < *Pxy x* ∧ 0 < *Py* (*snd x*)))
using *AE-space* **by** *eventually-elim* (*auto simp*: *space-pair-measure less-le*)
then have *ae*: *AE x in S* ⨂$_M$ *T*.
Pxy x ∗ *log b* (*Pxy x*) − *Pxy x* ∗ *log b* (*Py* (*snd x*)) = *Pxy x* ∗ *log b* (*Pxy x* / *Py* (*snd x*))
by *eventually-elim* (*auto simp*: *log-simps field-simps b-gt-1*)
have *conditional-entropy b S T X Y* =
− (∫ *x*. *Pxy x* ∗ *log b* (*Pxy x*) − *Pxy x* ∗ *log b* (*Py* (*snd x*)) ∂(*S* ⨂$_M$ *T*))
unfolding *conditional-entropy-generic-eq*[*OF S T Py Py-nn Pxy Pxy-nn*, *simplified*] *neg-equal-iff-equal*
apply (*intro integral-cong-AE*)
using *ae*
apply *auto*
done
also have ... = − (∫ *x*. *Pxy x* ∗ *log b* (*Pxy x*) ∂(*S* ⨂$_M$ *T*)) − − (∫ *x*. *Pxy x* ∗ *log b* (*Py* (*snd x*)) ∂(*S* ⨂$_M$ *T*))
by (*simp add*: *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*: *simple-function M X* **and** *Y*: *simple-function M Y*
shows $\mathcal{H}(X \mid Y)$ = *entropy b* (*count-space* (*X*‘*space M*) ⨂$_M$ *count-space* (*Y*‘*space M*)) (λ*x*. (*X x*, *Y x*)) − $\mathcal{H}(Y)$
proof −
have *count-space* (*X* ‘ *space M*) ⨂$_M$ *count-space* (*Y* ‘ *space M*) = *count-space* (*X*‘*space M* × *Y*‘*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*: *simple-distributed M Y Py*
assumes *XY*: *simple-distributed M* (λ*x*. (*X x*, *Y x*)) *Pxy*
shows $\mathcal{H}(X \mid Y) = -$ (∑ (*x*, *y*)∈(λ*x*. (*X x*, *Y x*)) ‘ *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* ((λ*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* = (λ*x*. *if x* ∈ (λ*x*. (*X x*, *Y x*)) ‘ *space M then Pxy x else 0*)
have *count-space* (*X* ‘ *space M*) ⨂$_M$ *count-space* (*Y* ‘ *space M*) = *count-space* (*X*‘*space M* × *Y*‘*space M*)
(**is** *?P = ?C*)
using *Y* **by** (*simp add*: *simple-distributed-finite pair-measure-count-space*)
have *eq*: (λ(*x*, *y*). *?f* (*x*, *y*) ∗ *log b* (*?f* (*x*, *y*) / *Py y*)) =
(λ*x*. *if x* ∈ (λ*x*. (*X x*, *Y x*)) ‘ *space M then Pxy x* ∗ *log b* (*Pxy x* / *Py* (*snd x*)) *else 0*)
by *auto*
from *Y* **show** − (∫ (*x*, *y*). *?f* (*x*, *y*) ∗ *log b* (*?f* (*x*, *y*) / *Py y*) ∂*?P*) =
− (∑ (*x*, *y*)∈(λ*x*. (*X x*, *Y x*)) ‘ *space M*. *Pxy* (*x*, *y*) ∗ *log b* (*Pxy* (*x*, *y*) / *Py y*))
by (*auto intro*!: *setsum.cong simp add*: ‹*?P = ?C*› *lebesgue-integral-count-space-finite simple-distributed-finite eq setsum.If-cases split-beta'*)
qed (*insert Y XY*, *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 −
def *Py* ≡ λ*x*. *if x* ∈ *Y*‘*space M then measure M* (*Y* −‘ {*x*} ∩ *space M*) *else 0*
def *Pxy* ≡ λ*x*. *if x* ∈ (λ*x*. (*X x*, *Y x*))‘*space M then measure M* ((λ*x*. (*X x*, *Y x*)) −‘ {*x*} ∩ *space M*) *else 0*
def *Pxxy* ≡ λ*x*. *if x* ∈ (λ*x*. (*X x*, *X x*, *Y x*))‘*space M then measure M* ((λ*x*. (*X*

$x,\ X\ x,\ Y\ x))\ -`\ \{x\} \cap$ *space* $M)$ *else* 0
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* $M\ (\lambda x.\ (X\ x,\ Y\ x))\ Pxy$
using XY **by** (*rule simple-distributedI*) (*auto simp*: *Pxy-def measure-nonneg*)
have $Pxxy$: *simple-distributed* $M\ (\lambda x.\ (X\ x,\ X\ x,\ Y\ x))\ Pxxy$
using XXY **by** (*rule simple-distributedI*) (*auto simp*: *Pxxy-def measure-nonneg*)
have *eq*: $(\lambda x.\ (X\ x,\ X\ x,\ Y\ x))$ ` *space* $M = (\lambda(x,\ y).\ (x,\ x,\ y))$ ` $(\lambda x.\ (X\ x,\ Y\ x))$ ` *space* M
by *auto*
have *inj*: $\bigwedge A.$ *inj-on* $(\lambda(x,\ y).\ (x,\ x,\ y))\ A$
by (*auto simp*: *inj-on-def*)
have *Pxxy-eq*: $\bigwedge x\ y.\ Pxxy\ (x,\ x,\ y) = Pxy\ (x,\ y)$
by (*auto simp*: *Pxxy-def Pxy-def intro*!: *arg-cong*[**where** f=*prob*])
have *AE x in count-space* $((\lambda x.\ (X\ x,\ Y\ x))$`*space* $M).\ Py\ (snd\ x) = 0 \longrightarrow Pxy\ x = 0$
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 Pxxy*])
apply (*subst conditional-entropy-eq*[*OF Py Pxy*])
apply (*auto intro*!: *setsum.cong simp*: *Pxxy-eq setsum-negf*[*symmetric*] *eq setsum.reindex*[*OF inj*]
log-simps zero-less-mult-iff zero-le-mult-iff field-simps mult-less-0-iff AE-count-space)
using Py[*THEN simple-distributed*] Pxy[*THEN simple-distributed*]
apply (*auto simp add*: *not-le AE-count-space less-le antisym*
simple-distributed-nonneg[*OF Py*] *simple-distributed-nonneg*[*OF Pxy*])
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-nonne*
X X Y]
by *simp*

40.8 Equalities

lemma (**in** *information-space*) *mutual-information-eq-entropy-conditional-entropy-distr*:
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*[*measurable*]: *distributed M S X Px*
and *Px-nn*[*simp*]: $\bigwedge x.\ x \in space\ S \Longrightarrow 0 \leq Px\ x$
and *Py*[*measurable*]: *distributed M T Y Py*
and *Py-nn*[*simp*]: $\bigwedge x.\ x \in space\ T \Longrightarrow 0 \leq Py\ x$
and *Pxy*[*measurable*]: *distributed M* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$ *Pxy*
and *Pxy-nn*[*simp*]: $\bigwedge x\ y.\ x \in space\ S \Longrightarrow y \in space\ T \Longrightarrow 0 \leq Pxy\ (x,\ y)$
assumes *Ix*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Px\ (fst\ x)))$
assumes *Iy*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Py\ (snd\ x)))$
assumes *Ixy*: *integrable*$(S \bigotimes_M T)$ $(\lambda x.\ Pxy\ x * log\ b\ (Pxy\ x))$
shows *mutual-information b S T X Y* $=$ *entropy b S X* $+$ *entropy b T Y* $-$ *entropy b* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x))$
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*]: $Pxy \in (S \bigotimes_M T) \rightarrow_M borel$
using *Pxy Pxy-nn* **by** (*intro distributed-real-measurable*) (*auto simp*: *space-pair-measure*)

have *X*: *entropy b S X* $= -$ $(\int x.\ Pxy\ x * log\ b\ (Px\ (fst\ x))\ \partial(S \bigotimes_M T))$
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* $= -$ $(\int x.\ Pxy\ x * log\ b\ (Py\ (snd\ x))\ \partial(S \bigotimes_M T))$
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* $(S \bigotimes_M T)$ **..**

have *XY*: *entropy b* $(S \bigotimes_M T)$ $(\lambda x.\ (X\ x,\ Y\ x)) = -$ $(\int x.\ Pxy\ x * log\ b\ (Pxy\ x)\ \partial(S \bigotimes_M T))$
by (*subst entropy-distr*[*OF Pxy*]) (*auto intro*!: *integral-cong simp*: *space-pair-measure*)

have *AE x in* $S \bigotimes_M T$. $Px\ (fst\ x) = 0 \longrightarrow Pxy\ x = 0$
by (*intro subdensity-real*[*of fst*, *OF - Pxy Px*]) (*auto intro*: *measurable-Pair simp*: *space-pair-measure*)
moreover have *AE x in* $S \bigotimes_M T$. $Py\ (snd\ x) = 0 \longrightarrow Pxy\ x = 0$
by (*intro subdensity-real*[*of snd*, *OF - Pxy Py*]) (*auto intro*: *measurable-Pair simp*: *space-pair-measure*)
moreover have *AE x in* $S \bigotimes_M T$. $0 \leq Px\ (fst\ x)$

```
  using Px by (intro ST.AE-pair-measure) (auto simp: comp-def intro!: measurable-fst'')
 moreover have AE x in S ⨂M T. 0 ≤ Py (snd x)
  using Py by (intro ST.AE-pair-measure) (auto simp: comp-def intro!: measurable-snd'')
 ultimately have AE x in S ⨂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))
= integral^L (S ⨂M T) ?f
   unfolding X Y XY
   apply (subst integral-diff)
   apply (intro integrable-diff Ixy Ix Iy)+
   apply (subst integral-diff)
   apply (intro Ixy Ix Iy)+
   apply (simp add: field-simps)
   done
 also have ... = integral^L (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* (λ*x*. *measure M* (*X* −‘ {*x*} ∩ *space M*))
using *sf-X* **by** (*rule simple-distributedI*) (*auto simp*: *measure-nonneg*)
have *Y*: *simple-distributed M Y* (λ*x*. *measure M* (*Y* −‘ {*x*} ∩ *space M*))
using *sf-Y* **by** (*rule simple-distributedI*) (*auto simp*: *measure-nonneg*)
have *sf-XY*: *simple-function M* (λ*x*. (*X x*, *Y x*))
using *sf-X sf-Y* **by** (*rule simple-function-Pair*)
then have *XY*: *simple-distributed M* (λ*x*. (*X x*, *Y x*)) (λ*x*. *measure M* ((λ*x*. (*X x*, *Y x*)) −‘ {*x*} ∩ *space M*))
by (*rule simple-distributedI*) (*auto simp*: *measure-nonneg*)
from *simple-distributed-joint-finite*[*OF this*, *simp*]
have *eq*: *count-space* (*X* ‘ *space M*) ⨂$_M$ *count-space* (*Y* ‘ *space M*) = *count-space* (*X* ‘ *space M* × *Y* ‘ *space M*)
by (*simp add*: *pair-measure-count-space*)

have $\mathcal{I}(X\ ;\ Y) = \mathcal{H}(X) + \mathcal{H}(Y) -$ *entropy b* (*count-space* (*X*‘*space M*) ⨂$_M$ *count-space* (*Y*‘*space M*)) (λ*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 *sf-X*: *simple-function M X* **and** *sf-Y*: *simple-function M Y*
shows $0 \le \mathcal{I}(X\ ;\ Y)$
proof −
have *X*: *simple-distributed M X* (λ*x*. *measure M* (*X* −‘ {*x*} ∩ *space M*))
using *sf-X* **by** (*rule simple-distributedI*) (*auto simp*: *measure-nonneg*)
have *Y*: *simple-distributed M Y* (λ*x*. *measure M* (*Y* −‘ {*x*} ∩ *space M*))
using *sf-Y* **by** (*rule simple-distributedI*) (*auto simp*: *measure-nonneg*)

have *sf-XY*: *simple-function M* (λ*x*. (*X x*, *Y x*))
using *sf-X sf-Y* **by** (*rule simple-function-Pair*)
then have *XY*: *simple-distributed M* (λ*x*. (*X x*, *Y x*)) (λ*x*. *measure M* ((λ*x*. (*X x*, *Y x*)) −‘ {*x*} ∩ *space M*))

by (*rule simple-distributedI*) (*auto simp*: *measure-nonneg*)

from *simple-distributed-joint-finite*[*OF this*, *simp*]
have *eq*: *count-space* (*X* ' *space M*) $\bigotimes_M$ *count-space* (*Y* ' *space M*) = *count-space* (*X* ' *space M* × *Y* ' *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* ≤ $\mathcal{I}(X ; Z)$ **using** *X Z* **by** (*rule mutual-information-nonneg-simple*)
also have $\mathcal{I}(X ; Z) = \mathcal{H}(X) - \mathcal{H}(X \mid Z)$ **using** *mutual-information-eq-entropy-conditional-entropy*[*OF assms*] **.**
finally show *?thesis* **by** *auto*
qed

lemma (**in** *information-space*)
fixes *Px* :: *'b* ⇒ *real* **and** *Py* :: *'c* ⇒ *real* **and** *Pxy* :: (*'b* × *'c*) ⇒ *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* $\bigotimes_M$ *T*) (λ*x*. (*X x*, *Y x*)) *Pxy*
shows *conditional-entropy b S T X Y* ≤ *entropy b S X*
proof −

have *0* ≤ *mutual-information b S T X Y*
by (*rule mutual-information-nonneg'*) *fact+*
also have ... = *entropy b S X* − *conditional-entropy b S T X Y*
apply (*rule mutual-information-eq-entropy-conditional-entropy'*)
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*)
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 $\mathcal{H}(\lambda x.\ (X\ x,\ Y\ x)) = \mathcal{H}(X) + \mathcal{H}(Y|X)$
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* = λ*x*. *prob* ((λ*x*. (*X x*, *Y x*)) −‘ {*x*} ∩ *space M*)
let *?g* = λ*x*. *prob* ((λ*x*. (*Y x*, *X x*)) −‘ {*x*} ∩ *space M*)
let *?h* = λ*x*. *if x* ∈ (λ*x*. (*Y x*, *X x*)) ‘ *space M then prob* ((λ*x*. (*Y x*, *X x*)) −‘ {*x*} ∩ *space M*) *else 0*
have ℋ(λ*x*. (*X x*, *Y x*)) = − (∑ *x*∈(λ*x*. (*X x*, *Y x*)) ‘ *space M*. *?f x* ∗ *log b* (*?f x*))
using *XY* **by** (*rule entropy-simple-distributed*)
also have ... = − (∑ *x*∈(λ(*x*, *y*). (*y*, *x*)) ‘ (λ*x*. (*X x*, *Y x*)) ‘ *space M*. *?g x* ∗ *log b* (*?g x*))
by (*subst* (*2*) *setsum.reindex*) (*auto simp*: *inj-on-def intro*!: *setsum.cong arg-cong*[**where** *f*=λ*A*. *prob A* ∗ *log b* (*prob A*)])
also have ... = − (∑ *x*∈(λ*x*. (*Y x*, *X x*)) ‘ *space M*. *?h x* ∗ *log b* (*?h x*))
by (*auto intro*!: *setsum.cong*)
also have ... = *entropy b* (*count-space* (*Y* ‘ *space M*) ⨂$_M$ *count-space* (*X* ‘ *space M*)) (λ*x*. (*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: *setsum.cong intro*!: *setsum.mono-neutral-left measure-nonneg*)
finally have ℋ(λ*x*. (*X x*, *Y x*)) = *entropy b* (*count-space* (*Y* ‘ *space M*) ⨂$_M$ *count-space* (*X* ‘ *space M*)) (λ*x*. (*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 ℋ(*X*) = ℋ(*f* ∘ *X*) + ℋ(*X*|*f* ∘ *X*)
proof −
note *fX* = *simple-function-compose*[*OF X*, *of f*]
have *eq*: (λ*x*. ((*f* ∘ *X*) *x*, *X x*)) ‘ *space M* = (λ*x*. (*f x*, *x*)) ‘ *X* ‘ *space M* **by** *auto*
have *inj*: ⋀*A*. *inj-on* (λ*x*. (*f x*, *x*)) *A*
by (*auto simp*: *inj-on-def*)
show *?thesis*
apply (*subst entropy-chain-rule*[*symmetric*, *OF fX X*])
apply (*subst entropy-simple-distributed*[*OF simple-distributedI*[*OF simple-function-Pair*[*OF fX X*] *measure-nonneg refl*]])
apply (*subst entropy-simple-distributed*[*OF simple-distributedI*[*OF X measure-nonneg refl*]])
unfolding *eq*
apply (*subst setsum.reindex*[*OF inj*])
apply (*auto intro*!: *setsum.cong arg-cong*[**where** *f*=λ*A*. *prob A* ∗ *log b* (*prob A*)])
done

qed

corollary (**in** *information-space*) *entropy-data-processing*:
assumes X: *simple-function M X* **shows** $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$
proof −
note fX = *simple-function-compose*[*OF X*, *of f*]
from X **have** $\mathcal{H}(X) = \mathcal{H}(f \circ X) + \mathcal{H}(X|f \circ X)$ **by** (*rule entropy-partition*)
then show $\mathcal{H}(f \circ X) \leq \mathcal{H}(X)$
by (*auto intro*: *conditional-entropy-nonneg*[*OF X fX*])
qed

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 $\mathcal{H}(X) = \mathcal{H}$(*the-inv-into* (*X*‘*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*=λx. *prob* (X −‘ $\{x\} \cap$ *space M*)])
apply (*auto intro*!: *setsum.cong arg-cong*[**where** *f*=*prob*] *image-eqI simp*: *the-inv-into-f-f*[*OF inj*] *comp-def measure-nonneg*)
done
also have ... $\leq \mathcal{H}(f \circ X)$
using *entropy-data-processing*[*OF sf*] **.**
finally show $\mathcal{H}(X) \leq \mathcal{H}(f \circ X)$ **.**
qed

end

41 Properties of Various Distributions

theory *Distributions*
imports *Convolution Information*
begin

lemma (**in** *prob-space*) *distributed-affine*:
fixes f :: *real* ⇒ *ennreal*
assumes f: *distributed M lborel X f*
assumes c: $c \neq 0$
shows *distributed M lborel* (λx. $t + c * X\ x$) (λx. $f\ ((x - t) / c) / |c|$)
unfolding *distributed-def*
proof *safe*
have [*measurable*]: $f \in$ *borel-measurable borel*

```
    using f by (simp add: distributed-def)
  have [measurable]: X ∈ borel-measurable M
    using f by (simp add: distributed-def)

  show (λx. f ((x − t) / c) / |c|) ∈ borel-measurable lborel
    by simp
  show random-variable lborel (λx. t + c * X x)
    by simp

  have eq: ennreal |c| * (f x / ennreal |c|) = f x for x
    using c
    by (cases f x)
     (auto simp: divide-ennreal ennreal-mult[symmetric] ennreal-top-divide ennreal-mult-top)

  have density lborel f = distr M lborel X
    using f by (simp add: distributed-def)
  with c show distr M lborel (λx. t + c * X x) = density lborel (λx. f ((x − t)
/ c) / ennreal |c|)
    by (subst (2) 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 :: real ⇒ ennreal and c :: real
  assumes f: distributed M lborel (λx. (X x − t) / c) (λx. |c| * f (x * c + t))
  assumes c: c ≠ 0
  shows distributed M lborel X f
proof −
  have eq: f x * ennreal |c| / 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 Measurable.pred N P
  shows (AE x in M. P (X x)) ⟷ (AE x in N. 0 < f x ⟶ P x)
proof −
  have (AE x in M. P (X x)) ⟷ (AE x in distr M N X. P x)
    by (simp add: AE-distr-iff)
  also have ... ⟷ (AE x in density N f. P x)
    unfolding distributed-distr-eq-density[OF assms(1)] ..
  also have ... ⟷ (AE x in N. 0 < f x ⟶ P x)
    by (rule AE-density) simp
  finally show ?thesis .
qed
```

41.1 Erlang

lemma *nn-intergal-power-times-exp-Icc*:
assumes [*arith*]: $0 \leq a$
shows $(\int^+ x.\ ennreal\ (x\hat{}k * exp\ (-x)) * indicator\ \{0\ ..\ a\}\ x\ \partial lborel) =$
$(1 - (\sum n{\leq}k.\ (a\hat{}n * exp\ (-a))\ /\ fact\ n)) * fact\ k$ (**is** *?I* = -)
proof −
let *?f* = $\lambda k\ x.\ x\hat{}k * exp\ (-x)\ /\ fact\ k$
let *?F* = $\lambda k\ x.\ -\ (\sum n{\leq}k.\ (x\hat{}n * exp\ (-x))\ /\ fact\ n)$
have *?I* $*$ (*inverse* (*real-of-nat* (*fact k*))) =
$(\int^+ x.\ ennreal\ (x\hat{}k * exp\ (-x)) * indicator\ \{0\ ..\ a\}\ x * (inverse\ (real\text{-}of\text{-}nat$
(*fact k*))) *∂lborel*)
by (*intro nn-integral-multc*[*symmetric*]) *auto*
also have ... = $(\int^+ x.\ ennreal\ (?f\ k\ x) * indicator\ \{0\ ..\ a\}\ x\ \partial lborel)$
by (*intro nn-integral-cong*)
(*simp add*: *field-simps ennreal-mult'*[*symmetric*] *indicator-mult-ennreal*)
also have ... = *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* − ($x\hat{}$*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 add: *field-simps power-Suc*[*symmetric*])
also have ($\lambda x.$ *?F k x* − ($x\hat{}$*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 add*: *of-nat-Suc field-simps*)
finally show *?case* **.**
qed
qed *auto*
also have ... = *ennreal* (1 − ($\sum n{\leq}k.$ ($a\hat{}n$ $*$ *exp* (−*a*)) / *fact n*))
by (*auto simp*: *power-0-left if-distrib*[**where** *f*=$\lambda x.$ *x* / *a* **for** *a*] *setsum.If-cases*)
also have ... = *ennreal* ((1 − ($\sum n{\leq}k.$ ($a\hat{}n$ $*$ *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-intergal-power-times-exp-Ici*:
shows $(\int^+ x.\ ennreal\ (x\hat{}k * exp\ (-x)) * indicator\ \{0\ ..\}\ x\ \partial lborel) = real\text{-}of\text{-}nat$
(*fact k*)

proof (*rule LIMSEQ-unique*)
let *?X* = $\lambda n.\ \int^+ x.\ ennreal\ (x\hat{}k * exp\ (-x)) * indicator\ \{0\ ..\ real\ n\}\ x\ \partial lborel$
show *?X* $\longrightarrow$ $(\int^+ x.\ ennreal\ (x\hat{}k * exp\ (-x)) * indicator\ \{0\ ..\}\ x\ \partial lborel)$
apply (*intro nn-integral-LIMSEQ*)
apply (*auto simp*: *incseq-def le-fun-def eventually-sequentially*
split: *split-indicator intro*!: *Lim-eventually*)
apply (*metis nat-ceiling-le-eq*)
done

have $((\lambda x{::}real.\ (1 - (\sum n{\le}k.\ (x\ \hat{}\ n\ /\ exp\ x)\ /\ (fact\ n))) * fact\ k) \longrightarrow$
$(1 - (\sum n{\le}k.\ 0\ /\ (fact\ n))) * fact\ k)$ *at-top*
by (*intro tendsto-intros tendsto-power-div-exp-0*) *simp*
then show *?X* $\longrightarrow$ *real-of-nat* (*fact k*)
by (*subst nn-intergal-power-times-exp-Icc*)
(*auto simp*: *exp-minus field-simps intro*!: *filterlim-compose*[*OF - filterlim-real-sequentially*])
qed

definition *erlang-density* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
erlang-density k l x = (*if x* < *0 then 0 else* ($l\hat{}(Suc\ k) * x\hat{}k * exp\ (-\ l * x)$) / *fact k*)

definition *erlang-CDF* :: *nat* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ *real* **where**
erlang-CDF k l x = (*if x* < *0 then 0 else 1* − $(\sum n{\le}k.\ ((l * x)\ \hat{}n * exp\ (-\ l * x)\ /\ fact\ n)))$

lemma *erlang-density-nonneg*[*simp*]: *0* ≤ *l* $\Longrightarrow$ *0* ≤ *erlang-density k l x*
by (*simp add*: *erlang-density-def*)

lemma *borel-measurable-erlang-density*[*measurable*]: *erlang-density k l* ∈ *borel-measurable borel*
by (*auto simp add*: *erlang-density-def*[*abs-def*])

lemma *erlang-CDF-transform*: *0* < *l* $\Longrightarrow$ *erlang-CDF k l a* = *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* ≤ *erlang-CDF k l x*
unfolding *erlang-CDF-def*
proof (*clarsimp simp*: *not-less*)
assume *0* ≤ *x*
have $(\sum n{\le}k.\ (l * x)\ \hat{}\ n * exp\ (-\ (l * x))\ /\ fact\ n) =$
exp $(-\ (l * x)) * (\sum n{\le}k.\ (l * x)\ \hat{}\ n\ /\ fact\ n)$
unfolding *setsum-right-distrib* **by** (*intro setsum.cong*) (*auto simp*: *field-simps*)
also have ... = $(\sum n{\le}k.\ (l * x)\ \hat{}\ n\ /\ fact\ n)\ /\ exp\ (l * x)$
by (*simp add*: *exp-minus field-simps*)
also have ... ≤ *1*
proof (*subst divide-le-eq-1-pos*)
show $(\sum n{\le}k.\ (l * x)\ \hat{}\ n\ /\ fact\ n) \le exp\ (l * x)$
using ‹*0* < *l*› ‹*0* ≤ *x*› *summable-exp-generic*[*of l * x*]

by (*auto simp*: *exp-def divide-inverse ac-simps intro*!: *setsum-le-suminf*)
qed *simp*
finally show $(\sum n{\le}k.\ (l * x)\ \hat{}\ n * exp\ (-\ (l * x))\ /\ fact\ n) \le 1$ **.**
qed

lemma *nn-integral-erlang-density*:
assumes [*arith*]: $0 < l$
shows $(\int^+ x.\ ennreal\ (erlang\text{-}density\ k\ l\ x) * indicator\ \{..\ a\}\ x\ \partial lborel) = erlang\text{-}CDF\ k\ l\ a$
proof *cases*
assume [*arith*]: $0 \le a$
have *eq*: $\bigwedge x.\ indicator\ \{0..a\}\ (x\ /\ l) = indicator\ \{0..a*l\}\ x$
by (*simp add*: *field-simps split*: *split-indicator*)
have $(\int^+ x.\ ennreal\ (erlang\text{-}density\ k\ l\ x) * indicator\ \{..\ a\}\ x\ \partial lborel) =$
$(\int^+ x.\ (l/fact\ k) * (ennreal\ ((l*x)\ \hat{}\ k * exp\ (-\ (l*x))) * indicator\ \{0\ ..\ a\}\ x)\ \partial lborel)$
by (*intro nn-integral-cong*)
(*auto simp*: *erlang-density-def power-mult-distrib ennreal-mult*[*symmetric*] *split*: *split-indicator*)
also have $\ldots = (l/fact\ k) * (\int^+ x.\ ennreal\ ((l*x)\ \hat{}\ k * exp\ (-\ (l*x))) * indicator\ \{0\ ..\ a\}\ x\ \partial lborel)$
by (*intro nn-integral-cmult*) *auto*
also have $\ldots = ennreal\ (l/fact\ k) * ((1/l) * (\int^+ x.\ ennreal\ (x\ \hat{}\ k * exp\ (-\ x)) * indicator\ \{0\ ..\ l * a\}\ x\ \partial lborel))$
by (*subst nn-integral-real-affine*[**where** *c=1 / l* **and** *t=0*]) (*auto simp*: *field-simps eq*)
also have $\ldots = (1\ -\ (\sum n{\le}k.\ ((l * a)\ \hat{}\ n * exp\ (-(l * a)))\ /\ fact\ n))$
by (*subst nn-intergal-power-times-exp-Icc*) (*auto simp*: *ennreal-mult'*[*symmetric*])
also have $\ldots = erlang\text{-}CDF\ k\ l\ a$
by (*auto simp*: *erlang-CDF-def*)
finally show *?thesis* **.**
next
assume $\neg\ 0 \le a$
moreover then have $(\int^+ x.\ ennreal\ (erlang\text{-}density\ k\ l\ x) * indicator\ \{..\ a\}\ x\ \partial lborel) = (\int^+ x.\ 0\ \partial(lborel::real\ measure))$
by (*intro nn-integral-cong*) (*auto simp*: *erlang-density-def*)
ultimately show *?thesis*
by (*simp add*: *erlang-CDF-def*)
qed

lemma *emeasure-erlang-density*:
$0 < l \Longrightarrow emeasure\ (density\ lborel\ (erlang\text{-}density\ k\ l))\ \{..\ a\} = erlang\text{-}CDF\ k\ l\ a$
by (*simp add*: *emeasure-density nn-integral-erlang-density*)

lemma *nn-integral-erlang-ith-moment*:
fixes *k i* :: *nat* **and** *l* :: *real*
assumes [*arith*]: $0 < l$
shows $(\int^+ x.\ ennreal\ (erlang\text{-}density\ k\ l\ x * x\ \hat{}\ i)\ \partial lborel) = fact\ (k + i)\ /$

(*fact k * l ^ i*)
proof −
have *eq*: ⋀*x*. *indicator* {*0*..} (*x* / *l*) = *indicator* {*0*..} *x*
by (*simp add*: *field-simps split*: *split-indicator*)
have (∫⁺ *x*. *ennreal* (*erlang-density k l x * x^i*) *∂lborel*) =
(∫⁺*x*. (*l*/(*fact k * l^i*)) * (*ennreal* ((*l**x*)^(*k*+*i*) * *exp* (− (*l**x*))) * *indicator* {*0* ..} *x*) *∂lborel*)
by (*intro nn-integral-cong*)
(*auto simp*: *erlang-density-def power-mult-distrib power-add ennreal-mult'*[*symmetric*] *split*: *split-indicator*)
also have ... = (*l*/(*fact k * l^i*)) * (∫⁺*x*. *ennreal* ((*l**x*)^(*k*+*i*) * *exp* (− (*l**x*))) * *indicator* {*0* ..} *x* *∂lborel*)
by (*intro nn-integral-cmult*) *auto*
also have ... = *ennreal* (*l*/(*fact k * l^i*)) * ((*1*/*l*) * (∫⁺*x*. *ennreal* (*x^(k+i) * exp* (− *x*)) * *indicator* {*0* ..} *x* *∂lborel*))
by (*subst nn-integral-real-affine*[**where** *c=1* / *l* **and** *t=0*]) (*auto simp*: *field-simps eq*)
also have ... = *fact* (*k* + *i*) / (*fact k * l ^ i*)
by (*subst nn-intergal-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* ≤ *a*
shows 𝒫(*x in M*. *X x* ≤ *a*) = *erlang-CDF k l a*
proof −
have *emeasure M* {*x* ∈ *space M*. *X x* ≤ *a* } = *emeasure* (*distr M lborel X*) {.. *a*}
using *distributed-measurable*[*OF D*]
by (*subst emeasure-distr*) (*auto intro*!: *arg-cong2*[**where** *f=emeasure*])
also have ... = *emeasure* (*density lborel* (*erlang-density k l*)) {.. *a*}
unfolding *distributed-distr-eq-density*[*OF D*] **..**
also have ... = *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 ≤ a*
shows $\mathcal{P}$(*x in M. a < X x*) = *1 −* (*erlang-CDF k l a*)
proof −
have *1 −* (*erlang-CDF k l a*) = *1 −* $\mathcal{P}$(*x in M. X x ≤ a*) **by** (*subst erlang-distributed-le*) *auto*
also have ... = *prob* (*space M −* {*x ∈ space M. X x ≤ a* })
using *distributed-measurable*[*OF D*] **by** (*auto simp*: *prob-compl*)
also have ... = $\mathcal{P}$(*x 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 ∈ borel-measurable M* **and** [*arith*]: *0 < l*
and *X-distr*: ⋀*a. 0 ≤ a ⟹ emeasure M* {*x∈space M. X x ≤ a*} = *erlang-CDF k l a*
shows *distributed M lborel X* (*erlang-density k l*)
proof (*rule distributedI-borel-atMost*)
fix *a* :: *real*
{ assume *a ≤ 0*
with *X* **have** *emeasure M* {*x∈space M. X x ≤ a*} ≤ *emeasure M* {*x∈space M. X x ≤ 0*}
by (*intro emeasure-mono*) *auto*
also have ... = *0* **by** (*auto intro*!: *erlang-CDF-at0 simp*: *X-distr*[*of 0*])
finally have *emeasure M* {*x∈space M. X x ≤ a*} ≤ *0* **by** *simp*
then have *emeasure M* {*x∈space M. X x ≤ a*} = *0* **by** *simp*
}
note *eq-0* = *this*

show (∫$^+$ *x. erlang-density k l x ∗ indicator* {..*a*} *x ∂lborel*) = *ennreal* (*erlang-CDF k l a*)
using *nn-integral-erlang-density*[*of l k a*]
by (*simp add*: *ennreal-indicator ennreal-mult*)

show *emeasure M* {*x∈space M. X x ≤ a*} = *ennreal* (*erlang-CDF k l a*)
using *X-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*) ⟷
(*X ∈ borel-measurable M ∧ 0 < l ∧* (∀ *a*≥*0*. $\mathcal{P}$(*x in M. X x ≤ a*) = *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-ennreal-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* (λ*x*. *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* (λ*x*. *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** *erlang-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* (λ*x*. *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* (λ*x*. *X x* ^ *i*)
by *rule* (*rule has-bochner-integral-erlang-ith-moment*)

lemma (**in** *prob-space*) *erlang-ith-moment*:

0 < l ⟹ distributed M lborel X (erlang-density k l) ⟹
*expectation (λx. X x ^ i) = fact (k + i) / (fact k * l ^ 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 lborel X (erlang-density k l)*
shows *variance X = (k + 1) / l²*
proof (*subst variance-eq*)
show *integrable M X integrable M (λx. (X x)²)*
using *erlang-ith-moment-integrable*[*OF assms*, *of 1*] *erlang-ith-moment-integrable*[*OF assms*, *of 2*]
by *auto*

show *expectation (λx. (X x)²) − (expectation X)² = real (k + 1) / l²*
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

41.2 Exponential distribution

abbreviation *exponential-density* :: *real ⇒ real ⇒ real* **where**
exponential-density ≡ erlang-density 0

lemma *exponential-density-def*:
*exponential-density l x = (if x < 0 then 0 else l * exp (− x * l))*
by (*simp add*: *fun-eq-iff erlang-density-def*)

lemma *erlang-CDF-0*: *erlang-CDF 0 l a = (if 0 ≤ a then 1 − exp (− l * a) else 0)*
by (*simp add*: *erlang-CDF-def*)

lemma *prob-space-exponential-density*: *0 < l ⟹ prob-space* (*density lborel* (*exponential-density l*))
by (*rule prob-space-erlang-density*)

lemma (**in** *prob-space*) *exponential-distributedD-le*:
assumes *D*: *distributed M lborel X* (*exponential-density l*) **and** *a*: *0 ≤ a* **and** *l*: *0 < l*
shows *𝒫*(*x in M. X x ≤ a*) = *1 − exp (− a * l)*
using *erlang-distributed-le*[*OF D l a*] *a* **by** (*simp add*: *erlang-CDF-def*)

lemma (**in** *prob-space*) *exponential-distributedD-gt*:
assumes *D*: *distributed M lborel X* (*exponential-density l*) **and** *a*: *0 ≤ a* **and** *l*: *0 < l*
shows *𝒫*(*x in M. a < X x*) = *exp (− a * l)*
using *erlang-distributed-gt*[*OF D l a*] *a* **by** (*simp add*: *erlang-CDF-def*)

lemma (**in** *prob-space*) *exponential-distributed-memoryless*:
assumes *D*: *distributed M lborel X* (*exponential-density l*) **and** *a*: *0 ≤ 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 ‹$0 \leq t$› **by** (*auto simp*: *cond-prob-def intro*!: *arg-cong*[**where** *f*=*prob*] *arg-cong2*[**where** *f*=*op* /])
also have ... = *exp* (− (*a* + *t*) ∗ *l*) / *exp* (− *a* ∗ *l*)
using *a t* **by** (*simp add*: *exponential-distributedD-gt*[*OF D* - *l*])
also have ... = *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*[*measurable*]: $X \in$ *borel-measurable M* **and** [*arith*]: $0 < l$
and *X-distr*: $\bigwedge a.\ 0 \leq a \Longrightarrow$ *emeasure M* {*x*∈*space M*. *X x* ≤ *a*} = 1 − *exp* (− *a* ∗ *l*)
shows *distributed M lborel X* (*exponential-density l*)
proof (*rule erlang-distributedI*)
fix *a* :: *real* **assume** $0 \leq a$ **then show** *emeasure M* {*x* ∈ *space M*. *X x* ≤ *a*} = *ennreal* (*erlang-CDF 0 l a*)
using *X-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 *distributed M lborel X* (*exponential-density l*) ⟷
(*X* ∈ *borel-measurable M* ∧ (∀ *a*≥*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*:
$0 < l \Longrightarrow$ *distributed M lborel X* (*exponential-density l*) $\Longrightarrow$ *expectation X* = *1* / *l*
using *erlang-ith-moment*[*of l X 0 1*] **by** *simp*

lemma *exponential-density-nonneg*: $0 < l \Longrightarrow 0 \leq$ *exponential-density l x*
by (*auto simp*: *exponential-density-def*)

lemma (**in** *prob-space*) *exponential-distributed-min*:
assumes $0 < l$ $0 < u$
assumes *expX*: *distributed M lborel X* (*exponential-density l*)
assumes *expY*: *distributed M lborel Y* (*exponential-density u*)
assumes *ind*: *indep-var borel X borel Y*
shows *distributed M lborel* (λ*x*. *min* (*X x*) (*Y x*)) (*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* (λ*x*. *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 in M*. *a < X x*) = *exp* (− *a* ∗ *l*)
by (*rule exponential-distributedD-gt*[*OF expX a*]) *fact*
have *gt2*[*simp*]: $\mathcal{P}$(*x in M*. *a < Y x*) = *exp* (− *a* ∗ *u*)
by (*rule exponential-distributedD-gt*[*OF expY a*]) *fact*

have $\mathcal{P}$(*x in M*. *a* < (*min* (*X x*) (*Y x*))) = $\mathcal{P}$(*x in M*. *a* < (*X x*) ∧ *a* < (*Y x*)) **by** (*auto intro*!:*arg-cong*[**where** *f*=*prob*])

also have ... = $\mathcal{P}$(*x in M*. *a* < (*X x*)) ∗ $\mathcal{P}$(*x 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 in M*. *a* < (*min* (*X x*) (*Y x*))) = *exp* (− *a* ∗ (*l* + *u*)) **.**

have {*x* ∈ *space M*. (*min* (*X x*) (*Y x*)) ≤*a* } = (*space M* − {*x* ∈ *space M*. *a*<(*min* (*X x*) (*Y x*)) })
by *auto*
then have *1* − *prob* {*x* ∈ *space M*. *a* < (*min* (*X x*) (*Y x*))} = *prob* {*x* ∈ *space M*. (*min* (*X x*) (*Y x*)) ≤ *a*}
using *randX randY* **by** (*auto simp*: *prob-compl*)
then show *prob* {*x* ∈ *space M*. (*min* (*X x*) (*Y x*)) ≤ *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* ≠ {}
assumes *l*: ⋀*i*. *i* ∈ *I* ⟹ *0 < l i*
assumes *expX*: ⋀*i*. *i* ∈ *I* ⟹ *distributed M lborel* (*X i*) (*exponential-density* (*l i*))
assumes *ind*: *indep-vars* (λ*i*. *borel*) *X I*
shows *distributed M lborel* (λ*x*. *Min* ((λ*i*. *X i x*)‘*I*)) (*exponential-density* (∑ *i*∈*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* (λ*x*. *min* (*X i x*) (*Min* ((λ*i*. *X i x*) ‘*I*))) (*exponential-density* (*l i* + (∑ *i*∈*I*. *l i*)))
by (*intro exponential-distributed-min indep-vars-Min insert*)
(*auto intro*: *indep-vars-subset setsum-pos*)
then show *?case*
using *insert* **by** *simp*
qed

lemma (**in** *prob-space*) *exponential-distributed-variance*:
0 < *l* ⟹ *distributed M lborel X* (*exponential-density l*) ⟹ *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* ⟹ (∫⁺*x*. *f x* ∂*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 (λ*x*. ∫⁺*y*. *ennreal* (*erlang-density* k_1 *l* (*x* − *y*)) ∗ *ennreal* (*erlang-density* k_2 *l y*) ∂*lborel*) =
(*erlang-density* (*Suc* k_1 + *Suc* k_2 − *1*) *l*)
(**is** *?LHS* = *?RHS*)
proof
fix *x* :: *real*
have *x* ≤ *0* ∨ *0* < *x*
by *arith*
then show *?LHS x* = *?RHS x*
proof
assume *x* ≤ *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*: (∫⁺ *x*. *ennreal* (*erlang-density* (*Suc* k_1 + *Suc* k_2 − *1*) *l x* ∗ *indicator* {*0*<..} *x*) ∂*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*: $integral^N$ *lborel ?LHS = 1*
by (*auto dest*!: *prob-space.emeasure-space-1 simp*: *emeasure-density*)

let *?I* = ($integral^N$ *lborel* (λ*y*. *ennreal* ((*1* − *y*) ^ k_1 ∗ *y*^k_2 ∗ *indicator* {*0..1*} *y*)))
let *?C* = (*fact* (*Suc* (k_1 + k_2))) / ((*fact* k_1) ∗ (*fact* k_2))
let *?s* = *Suc* k_1 + *Suc* k_2 − *1*
let *?L* = (λ*x*. ∫ +*y*. *ennreal* (*erlang-density* k_1 *l* (*x*− *y*) ∗ *erlang-density* k_2 *l y* ∗ *indicator* {*0..x*} *y*) ∂*lborel*)

{ fix *x* :: *real* **assume** [*arith*]: *0* < *x*
have ∗: ⋀*x y n*. (*x* − *y* ∗ *x*::*real*) ^*n* = *x* ^*n* ∗ (*1* − *y*) ^*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 ... = (λ*x*. *ennreal ?C* ∗ *?I* ∗ *erlang-density ?s 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 (∫ +*y*. *ennreal* (*erlang-density* k_1 *l* (*x* − *y*) ∗ *erlang-density* k_2 *l y*) ∂*lborel*) =
(λ*x*. *ennreal ?C* ∗ *?I* ∗ *erlang-density ?s l x*) *x*
by (*simp add*: *ennreal-mult*) **}**
note ∗ = *this*

assume [*arith*]: *0* < *x*
have *3*: *1* = $integral^N$ *lborel* (λ*xa*. *?LHS xa* ∗ *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 ... = $integral^N$ *lborel* (λ*x*. (*ennreal* (*?C*) ∗ *?I*) ∗ ((*erlang-density ?s l x*) ∗ *indicator* {*0*<*..*} *x*))
by (*auto intro*!: *nn-integral-cong simp*: *ennreal-mult*[*symmetric*] ∗ *split*: *split-indicator*)
also have ... = *ennreal* (*?C*) ∗ *?I*
using *1*
by (*auto simp*: *nn-integral-cmult*)
finally have *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* (*Suc* k_1 + *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-setsum*:
assumes *finI* : *finite I*
assumes *A*: $I \neq \{\}$
assumes [*simp*, *arith*]: $0 < l$
assumes *expX*: $\bigwedge i.\ i \in I \Longrightarrow$ *distributed M lborel* (*X i*) (*erlang-density* (*k i*) *l*)
assumes *ind*: *indep-vars* ($\lambda i.$ *borel*) *X I*
shows *distributed M lborel* ($\lambda x.\ \sum i{\in}I.\ X\ i\ x$) (*erlang-density* (($\sum i{\in}I.$ *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* (*Suc* (*k i*) + *Suc* (($\sum i{\in}I.$ *Suc* (*k i*)) − *1*) − *1*) *l*)
by(*intro sum-indep-erlang indep-vars-setsum*) (*auto intro*!: *indep-vars-subset*)
also have ($\lambda x.$ (*X i x*) + ($\sum i{\in}$ *I*. *X i x*)) = ($\lambda x.\ \sum i{\in}$*insert i I*. *X i x*)
using *insert* **by** *auto*
also have *Suc*(*k i*) + *Suc* (($\sum i{\in}I.$ *Suc* (*k i*)) − *1*) − *1* = ($\sum i{\in}$*insert i I*. *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-setsum*:
assumes *finI*: *finite I*
assumes *A*: $I \neq \{\}$
assumes *l*: $0 < l$
assumes *expX*: $\bigwedge i.\ i \in I \Longrightarrow$ *distributed M lborel* (*X i*) (*exponential-density l*)
assumes *ind*: *indep-vars* ($\lambda i.$ *borel*) *X I*

shows *distributed M lborel* ($\lambda x.\ \sum i{\in}I.\ X\ i\ x$) (*erlang-density* ((*card I*) − *1*) *l*)
using *erlang-distributed-setsum*[*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* λ-. *1*] **by** *simp*

have [*simp*]: *integral*^{L} *lborel* (*exponential-density l*) = *1*
using *distributed-integral*[*OF D*, *of* λ-. *1*] **by** (*simp add*: *prob-space*)

have [*simp*]: *integrable lborel* (λ*x*. *exponential-density l x* ∗ *x*)
using *erlang-ith-moment-integrable*[*OF l D*, *of 1*] *distributed-integrable*[*OF D*, *of* λ*x*. *x*] **by** *simp*

have [*simp*]: *integral*^{L} *lborel* (λ*x*. *exponential-density l x* ∗ *x*) = *1* / *l*
using *erlang-ith-moment*[*OF l D*, *of 1*] *distributed-integral*[*OF D*, *of* λ*x*. *x*] **by** *simp*

have *entropy b lborel X* = − (∫ *x*. *exponential-density l x* ∗ *log b* (*exponential-density l x*) ∂*lborel*)
using *D* **by** (*rule entropy-distr*) *simp*
also have (∫ *x*. *exponential-density l x* ∗ *log b* (*exponential-density l x*) ∂*lborel*) =
(∫ *x*. (*ln l* ∗ *exponential-density l x* − *l* ∗ (*exponential-density l x* ∗ *x*)) / *ln b* ∂*lborel*)
by (*intro integral-cong*) (*auto simp*: *log-def ln-mult exponential-density-def field-simps*)
also have ... = (*ln l* − *1*) / *ln b*
by *simp*
finally show *?thesis*
by (*simp add*: *log-def divide-simps ln-div*)
qed

41.3 Uniform distribution

lemma *uniform-distrI*:
assumes *X*: *X* ∈ *measurable M M'*
and *A*: *A* ∈ *sets M' emeasure M' A* ≠ ∞ *emeasure M' A* ≠ *0*
assumes *distr*: ⋀*B*. *B* ∈ *sets M'* ⟹ *emeasure M* (*X* −‘ *B* ∩ *space M*) = *emeasure M'* (*A* ∩ *B*) / *emeasure M' A*
shows *distr M M' X* = *uniform-measure M' A*
unfolding *uniform-measure-def*
proof (*intro measure-eqI*)
let *?f* = λ*x*. *indicator A x* / *emeasure M' A*
fix *B* **assume** *B*: *B* ∈ *sets* (*distr M M' X*)

with X **have** *emeasure* M $(X - `` B \cap$ *space* $M) =$ *emeasure* M' $(A \cap B)$ / *emeasure* M' A
by (*simp add*: *distr*[*of* B] *measurable-sets*)
also have $\ldots =$ (*1* / *emeasure* M' A) $*$ *emeasure* M' $(A \cap B)$
by (*simp add*: *divide-ennreal-def ac-simps*)
also have $\ldots = (\int^+ x.$ (*1* / *emeasure* M' A) $*$ *indicator* $(A \cap B)$ x $\partial M')$
using A B
by (*intro nn-integral-cmult-indicator*[*symmetric*]) (*auto intro*!:)
also have $\ldots = (\int^+ x.$ *?f* $x *$ *indicator* B x $\partial M')$
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*]: $X \in$ *borel-measurable* M **and** A: *emeasure lborel* $A =$ *ennreal r 0* $< r$
and [*measurable*]: $A \in$ *sets borel*
assumes *distr*: $\bigwedge a.$ *emeasure* M $\{x \in$ *space* $M.$ X $x \leq a\} =$ *emeasure lborel* $(A \cap \{..$ $a\})$ / r
shows *distributed M lborel X* $(\lambda x.$ *indicator* A x / *measure lborel* $A)$
proof (*rule distributedI-borel-atMost*)
let *?f* $= \lambda x.$ *1* / $r *$ *indicator* A x
fix a
have *emeasure lborel* $(A \cap \{..a\}) \leq$ *emeasure lborel* A
using A **by** (*intro emeasure-mono*) *auto*
also have $\ldots < \infty$
using A **by** *simp*
finally have *fin*: *emeasure lborel* $(A \cap \{..a\}) \neq$ *top*
by *simp*
from *emeasure-eq-ennreal-measure*[*OF this*]
have *fin-eq*: *emeasure lborel* $(A \cap \{..a\})$ / $r =$ *ennreal* (*measure lborel* $(A \cap \{..a\})$ / $r)$
using A **by** (*simp add*: *divide-ennreal measure-nonneg*)
then show *emeasure* M $\{x \in$ *space* $M.$ X $x \leq a\} =$ *ennreal* (*measure lborel* $(A \cap \{..a\})$ / $r)$
using *distr* **by** *simp*

have $(\int^+ x.$ *ennreal* (*indicator* A x / *measure lborel* $A *$ *indicator* $\{..a\}$ $x)$ ∂*lborel*$) =$
$(\int^+ x.$ *ennreal* (*1* / *measure lborel* $A) *$ *indicator* $(A \cap \{..a\})$ x ∂*lborel*$)$
by (*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots =$ *ennreal* (*1* / *measure lborel* $A) *$ *emeasure lborel* $(A \cap \{..a\})$
using ‹$A \in$ *sets borel*›
by (*intro nn-integral-cmult-indicator*) (*auto simp*: *measure-nonneg*)
also have $\ldots =$ *ennreal* (*measure lborel* $(A \cap \{..a\})$ / $r)$
unfolding *emeasure-eq-ennreal-measure*[*OF fin*] **using** A
by (*simp add*: *measure-def ennreal-mult'*[*symmetric*])

finally show ($\int^+ x.$ *ennreal* (*indicator A x / measure lborel A* $*$ *indicator* $\{..a\}$ *x*) *∂lborel*) =
ennreal (*measure lborel* ($A \cap \{..a\}$) / *r*) **.**
qed (*auto simp*: *measure-nonneg*)

lemma (**in** *prob-space*) *uniform-distrI-borel-atLeastAtMost*:
fixes *a b* :: *real*
assumes *X*: $X \in$ *borel-measurable M* **and** $a < b$
assumes *distr*: $\bigwedge t.\ a \leq t \Longrightarrow t \leq b \Longrightarrow \mathcal{P}(x \text{ in } M.\ X\ x \leq t) = (t - a) / (b - a)$
shows *distributed M lborel X* ($\lambda x.$ *indicator* $\{a..b\}$ *x / 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 *emeasure M* $\{x \in$ *space M*. *X x* $\leq t\}$ = *emeasure lborel* ($\{a$ .. $b\} \cap \{..t\}$) / $(b - a)$
proof (*elim disjE conjE*)
assume $t < a$
then have *emeasure M* $\{x \in$ *space M*. *X x* $\leq t\} \leq$ *emeasure M* $\{x \in$ *space M*. *X x* $\leq a\}$
using *X* **by** (*auto intro*!: *emeasure-mono measurable-sets*)
also have ... = *0*
using *distr*[*of a*] ‹$a < b$› **by** (*simp add*: *emeasure-eq-measure*)
finally have *emeasure M* $\{x \in$ *space M*. *X x* $\leq t\} = 0$
by (*simp add*: *antisym measure-nonneg*)
with ‹$t < a$› **show** *?thesis* **by** *simp*
next
assume *bnds*: $a \leq t$ $t \leq b$
have $\{a..b\} \cap \{..t\} = \{a..t\}$
using *bnds* **by** *auto*
then show *?thesis* **using** ‹$a \leq t$› ‹$a < b$›
using *distr*[*OF bnds*] **by** (*simp add*: *emeasure-eq-measure divide-ennreal*)
next
assume $b < t$
have *1* = *emeasure M* $\{x \in$ *space M*. *X x* $\leq b\}$
using *distr*[*of b*] ‹$a < b$› **by** (*simp add*: *one-ennreal-def emeasure-eq-measure*)
also have ... $\leq$ *emeasure M* $\{x \in$ *space M*. *X x* $\leq t\}$
using *X* ‹$b < t$› **by** (*auto intro*!: *emeasure-mono measurable-sets*)
finally have *emeasure M* $\{x \in$ *space M*. *X x* $\leq t\} = 1$
by (*simp add*: *antisym emeasure-eq-measure*)
with ‹$b < t$› ‹$a < b$› **show** *?thesis* **by** (*simp add*: *measure-def divide-ennreal*)
qed
qed (*insert X* ‹$a < b$›, *auto*)

lemma (**in** *prob-space*) *uniform-distributed-measure*:
fixes *a b* :: *real*
assumes *D*: *distributed M lborel X* ($\lambda x.$ *indicator* $\{a$ .. $b\}$ *x / measure lborel* $\{a$ .. $b\}$)

assumes t: $a \leq t$ $t \leq b$
shows $\mathcal{P}(x \text{ in } M.\ X\ x \leq t) = (t - a) \mathbin{/} (b - a)$
proof −
have *emeasure* M $\{x \in \text{space } M.\ X\ x \leq t\} = \text{emeasure } (\text{distr } M \text{ lborel } X)$ $\{..\ t\}$
using *distributed-measurable*[*OF D*]
by (*subst emeasure-distr*) (*auto intro*!: *arg-cong2*[**where** *f*=*emeasure*])
also have ... $= (\int^+ x.\ \text{ennreal } (1 \mathbin{/} (b - a)) * \text{indicator } \{a\ ..\ t\}\ x\ \partial\text{lborel})$
using *distributed-borel-measurable*[*OF D*] ‹$a \leq t$› ‹$t \leq b$›
unfolding *distributed-distr-eq-density*[*OF D*]
by (*subst emeasure-density*)
(*auto intro*!: *nn-integral-cong simp*: *measure-def split*: *split-indicator*)
also have ... $= \text{ennreal } (1 \mathbin{/} (b - a)) * (t - a)$
using ‹$a \leq t$› ‹$t \leq b$›
by (*subst nn-integral-cmult-indicator*) *auto*
finally show *?thesis*
using t **by** (*simp add*: *emeasure-eq-measure ennreal-mult''*[*symmetric*] *measure-nonneg*)
qed

lemma (**in** *prob-space*) *uniform-distributed-bounds*:
fixes a b :: *real*
assumes D: *distributed M lborel X* ($\lambda x.$ *indicator* $\{a\ ..\ b\}$ x / *measure lborel* $\{a\ ..\ b\}$)
shows $a < b$
proof (*rule ccontr*)
assume ¬ $a < b$
then have $\{a\ ..\ b\} = \{\} \vee \{a\ ..\ b\} = \{a\ ..\ a\}$ **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* ($\lambda x.$ *indicator* $\{a..b\}$ x / *measure lborel* $\{a..b\}$)
$\longleftrightarrow$
($X \in$ *borel-measurable M* $\wedge$ $a < b$ $\wedge$ $(\forall t \in \{a\ ..\ b\}.\ \mathcal{P}(x \text{ in } M.\ X\ x \leq t) = (t - a) \mathbin{/} (b - a))$)
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*)

lemma (**in** *prob-space*) *uniform-distributed-expectation*:
fixes a b :: *real*
assumes D: *distributed M lborel X* ($\lambda x.$ *indicator* $\{a\ ..\ b\}$ x / *measure lborel* $\{a\ ..\ b\}$)
shows *expectation* $X = (a + b) \mathbin{/} 2$
proof (*subst distributed-integral*[*OF D*, *of* $\lambda x.\ x$, *symmetric*])
have $a < b$

using *uniform-distributed-bounds*[*OF D*] **.**

have ($\int$ *x. indicator* {*a .. b*} *x / measure lborel* {*a .. b*} ∗ *x ∂lborel*) =
($\int$ *x.* (*x / measure lborel* {*a .. b*}) ∗ *indicator* {*a .. b*} *x ∂lborel*)
by (*intro integral-cong*) *auto*
also have ($\int$ *x.* (*x / measure lborel* {*a .. b*}) ∗ *indicator* {*a .. b*} *x ∂lborel*) =
(*a* + *b*) / *2*
proof (*subst integral-FTC-Icc-real*)
fix *x*
show *DERIV* (λ*x.* x^2 / (*2* ∗ *measure lborel* {*a..b*})) *x* :> *x / measure lborel*
{*a..b*}
using *uniform-distributed-params*[*OF D*]
by (*auto intro*!: *derivative-eq-intros*)
show *isCont* (λ*x. x / Sigma-Algebra.measure lborel* {*a..b*}) *x*
using *uniform-distributed-params*[*OF D*]
by (*auto intro*!: *isCont-divide*)
have ∗: b^2 / (*2* ∗ *measure lborel* {*a..b*}) − a^2 / (*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 b^2 / (*2* ∗ *measure lborel* {*a..b*}) − a^2 / (*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 ($\int$ *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* (λ*x. 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* = λ*x. indicator* {*a..b*} (*x* + *?μ*) / *measure lborel*
{*a..b*}
have ($\int$ *x.* x^2 ∗ (*?D x*) *∂lborel*) = ($\int$ *x.* x^2 ∗ (*indicator* {*a* − *?μ .. b* − *?μ*} *x*)
/ *measure lborel* {*a .. b*} *∂lborel*)
by (*intro 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 ($\int$ *x.* x^2 ∗ *?D x ∂lborel*) = (*b* − *a*)^2 / *12* **.**
qed (*auto intro*: *D simp*: *measure-nonneg*)

41.4 Normal distribution

definition *normal-density* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* **where**
normal-density μ σ x = *1* / *sqrt* (*2* ∗ *pi* ∗ σ^2) ∗ *exp* ($-(x - \mu)^2$/ (*2* ∗ σ^2))

abbreviation *std-normal-density* :: *real* ⇒ *real* **where**
std-normal-density ≡ *normal-density 0 1*

lemma *std-normal-density-def*: *std-normal-density* x = (*1* / *sqrt* (*2* ∗ *pi*)) ∗ *exp* ($-$ x^2 / *2*)
unfolding *normal-density-def* **by** *simp*

lemma *normal-density-nonneg*[*simp*]: *0* ≤ *normal-density* μ σ x
by (*auto simp*: *normal-density-def*)

lemma *normal-density-pos*: *0* < σ ⟹ *0* < *normal-density* μ σ x
by (*auto simp*: *normal-density-def*)

lemma *borel-measurable-normal-density*[*measurable*]: *normal-density* μ σ ∈ *borel-measurable borel*
by (*auto simp*: *normal-density-def*[*abs-def*])

lemma *gaussian-moment-0*:
has-bochner-integral lborel (λ*x*. *indicator* {*0*..} *x* ∗$_R$ *exp* ($-$ x^2)) (*sqrt pi* / *2*)
proof −
let *?pI* = λ*f*. (∫$^+$*s*. *f* (*s*::*real*) ∗ *indicator* {*0*..} *s* ∂*lborel*)
let *?gauss* = λ*x*. *exp* ($-$ x^2)

let *?I* = *indicator* {*0*<..} :: *real* ⇒ *real*
let *?ff*= λ*x s*. *x* ∗ *exp* ($-$ x^2 ∗ (*1* + s^2)) :: *real*

have ∗: *?pI ?gauss* = (∫$^+$*x*. *?gauss x* ∗ *?I x* ∂*lborel*)
by (*intro nn-integral-cong-AE AE-I*[**where** *N*={*0*}]) (*auto split*: *split-indicator*)

have *?pI ?gauss* ∗ *?pI ?gauss* = (∫$^+$*x*. ∫$^+$*s*. *?gauss x* ∗ *?gauss s* ∗ *?I s* ∗ *?I x* ∂*lborel* ∂*lborel*)
by (*auto simp*: *nn-integral-cmult*[*symmetric*] *nn-integral-multc*[*symmetric*] ∗ *ennreal-mult*[*symmetric*]
intro!: *nn-integral-cong split*: *split-indicator*)
also have ... = (∫$^+$*x*. ∫$^+$*s*. *?ff x s* ∗ *?I s* ∗ *?I x* ∂*lborel* ∂*lborel*)
proof (*rule nn-integral-cong*, *cases*)
fix *x* :: *real* **assume** *x* ≠ *0*
then show (∫$^+$*s*. *?gauss x* ∗ *?gauss s* ∗ *?I s* ∗ *?I x* ∂*lborel*) = (∫$^+$*s*. *?ff x s* ∗ *?I s* ∗ *?I x* ∂*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 ... = ∫$^+$*s*. ∫$^+$*x*. *?ff x s* ∗ *?I s* ∗ *?I x* ∂*lborel* ∂*lborel*

```
    by (rule lborel-pair.Fubini'[symmetric]) auto
  also have ... = ?pI (λs. ?pI (λx. ?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 ?gauss^2 = 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 ((λa::real. - exp (- a²) / 2) ⟶ 0) at-top
      by simp
  qed (auto intro!: derivative-eq-intros)
  also have ... = 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 (λx::real. indicator {0..} x *R (exp (-x²)*x^(2 *
k)))
       ((sqrt pi / 2) * (fact (2 * k) / (2 ^ (2 * k) * fact k)))
       (is ?even)
    and gaussian-moment-odd-pos:
      has-bochner-integral lborel (λx::real. indicator {0..} x *R (exp (-x²)*x^(2 *
k + 1))) (fact k / 2)
      (is ?odd)
proof -
  let ?M = λk x. exp (- x²) * x^k :: real

  { fix k I assume Mk: has-bochner-integral lborel (λx. indicator {0..} x *R ?M
k x) I
    have 2 ≠ (0::real)
      by linarith
    let ?f = λb. ∫ x. indicator {0..} x *R ?M (k + 2) x * indicator {..b} x ∂lborel
    have ((λb. (k + 1) / 2 * (∫ x. indicator {..b} x *R (indicator {0..} x *R ?M
k x) ∂lborel) - ?M (k + 1) b / 2) ⟶
        (k + 1) / 2 * (∫ x. indicator {0..} x *R ?M k x ∂lborel) - 0 / 2) at-top
(is ?tendsto)
   proof (intro tendsto-intros ‹2 ≠ 0› tendsto-integral-at-top sets-lborel Mk[THEN
integrable.intros])
      show (?M (k + 1) ⟶ 0) at-top
      proof cases
        assume even k
        have ((λx. ((x²) ^(k div 2 + 1) / exp (x²)) * (1 / x) :: real) ⟶ 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 $(\lambda x.\ ((x^2)\ \hat{}\ (k\ div\ 2\ +\ 1)\ /\ exp\ (x^2)) * (1\ /\ x) :: real) = ?M\ (k + 1)$
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 $((\lambda x.\ ((x^2)\ \hat{}\ ((k - 1)\ div\ 2 + 1)\ /\ exp\ (x^2)) :: real) \longrightarrow 0)\ at\text{-}top$
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 $(\lambda x.\ ((x^2)\ \hat{}\ ((k - 1)\ div\ 2 + 1)\ /\ exp\ (x^2)) :: real) = ?M\ (k + 1)$
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 $?tendsto \longleftrightarrow ((?f \longrightarrow (k + 1)\ /\ 2 * (\int x.\ indicator\ \{0..\}\ x *_R ?M\ k\ x\ \partial lborel) - 0\ /\ 2)\ at\text{-}top)$
proof (*intro filterlim-cong refl eventually-at-top-linorder*[*THEN iffD2*] *exI*[*of - 0*] *allI impI*)
fix $b :: real$ **assume** b: $0 \le b$
have $Suc\ k * (\int x.\ indicator\ \{0..b\}\ x *_R ?M\ k\ x\ \partial lborel) = (\int x.\ indicator\ \{0..b\}\ x *_R (exp\ (-\ x^2) * ((Suc\ k) * x\ \hat{}\ k))\ \partial lborel)$
unfolding *integral-mult-right-zero*[*symmetric*] **by** (*intro integral-cong*) *auto*
also have $\ldots = exp\ (-\ b^2) * b\ \hat{}\ (Suc\ k) - exp\ (-\ 0^2) * 0\ \hat{}\ (Suc\ k) - (\int x.\ indicator\ \{0..b\}\ x *_R (-\ 2 * x * exp\ (-\ x^2) * x\ \hat{}\ (Suc\ k))\ \partial lborel)$
by (*rule integral-by-parts'*)
(*auto intro*!: *derivative-eq-intros b*
simp: *diff-Suc of-nat-Suc field-simps split*: *nat.split*)
also have $(\int x.\ indicator\ \{0..b\}\ x *_R (-\ 2 * x * exp\ (-\ x^2) * x\ \hat{}\ (Suc\ k))\ \partial lborel) =$
$(\int x.\ indicator\ \{0..b\}\ x *_R (-\ 2 * (exp\ (-\ x^2) * x\ \hat{}\ (k + 2)))\ \partial lborel)$
by (*intro integral-cong*) *auto*
finally have $Suc\ k * (\int x.\ indicator\ \{0..b\}\ x *_R ?M\ k\ x\ \partial lborel) = exp\ (-\ b^2) * b\ \hat{}\ (Suc\ k) + 2 * (\int x.\ indicator\ \{0..b\}\ x *_R ?M\ (k + 2)\ x\ \partial lborel)$
by (*simp del*: *real-scaleR-def integral-mult-right add*: *integral-mult-right*[*symmetric*])
then show $(k + 1)\ /\ 2 * (\int x.\ indicator\ \{..b\}\ x *_R (indicator\ \{0..\}\ x *_R ?M\ k\ x)\partial lborel) - exp\ (-\ b^2) * b\ \hat{}\ (k + 1)\ /\ 2 = ?f\ b$
by (*simp add*: *field-simps atLeastAtMost-def indicator-inter-arith*)
qed
finally have *int-M-at-top*: $((?f \longrightarrow (k + 1)\ /\ 2 * (\int x.\ indicator\ \{0..\}\ x *_R ?M\ k\ x\ \partial lborel))\ at\text{-}top)$
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 divide-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
```

$(2 * k) / ((2 / \sigma^2) \hat{} k * fact\ k))$
proof −
have *eq*: $\bigwedge x$::*real*. $x^2 \hat{} k = (x \hat{} k)^2$
by (*simp add*: *power-mult*[*symmetric*] *ac-simps*)

have *has-bochner-integral lborel* (λx. *exp* $(-x^2) * x \hat{} (2 * k)$)
(*sqrt pi* $*$ (*fact* $(2 * k)$ / $(2 \hat{} (2 * k) * fact\ k)$))
using *has-bochner-integral-even-function*[*OF gaussian-moment-even-pos*[**where** *k*=*k*]] **by** *simp*
then have *has-bochner-integral lborel* (λx. (*exp* $(-x^2) * x \hat{} (2 * k)$) $*$ $((2 * \sigma^2) \hat{} k$ / *sqrt* $(2 * pi * \sigma^2)))$
((*sqrt pi* $*$ (*fact* $(2 * k)$ / $(2 \hat{} (2 * k) * fact\ k))) * ((2 * \sigma^2) \hat{} k$ / *sqrt* $(2 * pi * \sigma^2)))$
by (*rule has-bochner-integral-mult-left*)
also have (λx. (*exp* $(-x^2) * x \hat{} (2 * k)$) $*$ $((2 * \sigma^2) \hat{} k$ / *sqrt* $(2 * pi * \sigma^2))) =$
(λx. *exp* $(-((sqrt\ 2 * \sigma) * x)^2 / (2 * \sigma^2)) * ((sqrt\ 2 * \sigma) * x) \hat{} (2 * k)$ / *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 ((*sqrt pi* $*$ (*fact* $(2 * k)$ / $(2 \hat{} (2 * k) * fact\ k))) * ((2 * \sigma^2) \hat{} k$ / *sqrt* $(2 * pi * \sigma^2))) =$
(*inverse* (*sqrt* $2 * \sigma$) $*$ ((*fact* $(2 * k)$)) / $((2/\sigma^2) \hat{} k * (fact\ k)))$
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*=*sqrt* $2 * \sigma$ **and** *t*=μ]) *simp-all*
qed

lemma *normal-moment-abs-odd*:
has-bochner-integral lborel (λx. *normal-density* $\mu\ \sigma\ x * |x - \mu| \hat{} (2 * k + 1))$ $(2 \hat{} k * \sigma \hat{} (2 * k + 1) * fact\ k * sqrt\ (2 / pi))$
proof −
have *has-bochner-integral lborel* (λx::*real*. *indicator* $\{0..\}\ x *_R$ $(exp\ (-x^2) * |x| \hat{} (2 * k + 1)))$ $(fact\ k / 2)$
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* (λx::*real*. *exp* $(-x^2) * |x| \hat{} (2 * k + 1))$ $(fact\ k)$
by *simp*
then have *has-bochner-integral lborel* (λx. (*exp* $(-x^2) * |x| \hat{} (2 * k + 1)) * (2 \hat{} k * \sigma \hat{} (2 * k + 1)$ / *sqrt* $(pi * \sigma^2)))$
$(fact\ k * (2 \hat{} k * \sigma \hat{} (2 * k + 1)$ / *sqrt* $(pi * \sigma^2)))$
by (*rule has-bochner-integral-mult-left*)
also have (λx. (*exp* $(-x^2) * |x| \hat{} (2 * k + 1)) * (2 \hat{} k * \sigma \hat{} (2 * k + 1)$ / *sqrt* $(pi * \sigma^2))) =$
(λx. *exp* $(-(((sqrt\ 2 * \sigma) * x)^2 / (2 * \sigma^2))) * |sqrt\ 2 * \sigma * x| \hat{} (2 * k + 1)$

/ *sqrt* (2 * *pi* * σ^2))
by (*simp add*: *field-simps abs-mult real-sqrt-power*[*symmetric*] *power-mult real-sqrt-mult*)
also have (*fact k* * (2^*k* * σ ^(2 * *k* + 1) / *sqrt* (*pi* * σ^2))) =
(*inverse* (*sqrt* 2) * *inverse* σ * (2 ^ *k* * (σ * σ ^ (2 * *k*)) * (*fact k*) * *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 * σ **and** *t*=μ])
simp-all
qed

lemma *normal-moment-odd*:
has-bochner-integral lborel (λx. *normal-density* μ σ *x* * (*x* − μ) ^(2 * *k* + 1)) 0
proof −
have *has-bochner-integral lborel* (λx. *exp* (− x^2) * *x*^(2 * *k* + 1)::*real*) 0
using *gaussian-moment-odd-pos* **by** (*rule has-bochner-integral-odd-function*) *simp*
then have *has-bochner-integral lborel* (λx. (*exp* (−x^2)*x^(2 * *k* + 1)) * (2^*k*σ ^(2*k*)/*sqrt pi*))
(0 * (2^*k*σ ^(2*k*)/*sqrt pi*))
by (*rule has-bochner-integral-mult-left*)
also have (λx. (*exp* (−x^2)*x^(2 * *k* + 1)) * (2^*k*σ ^(2*k*)/*sqrt pi*)) =
(λx. *exp* (− ((*sqrt* 2 * σ * *x*)^2 / (2 * σ^2))) *
(*sqrt* 2 * σ * *x* * (*sqrt* 2 * σ * *x*) ^ (2 * *k*)) /
sqrt (2 * *pi* * σ^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 * σ **and** *t*=μ]) *simp-all*
qed

lemma *integral-normal-moment-even*:
$\mathit{integral}^L$ *lborel* (λx. *normal-density* μ σ *x* * (*x* − μ) ^(2 * *k*)) = *fact* (2 * *k*) / ((2 / σ^2) ^*k* * *fact k*)
using *normal-moment-even* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-normal-moment-abs-odd*:
$\mathit{integral}^L$ *lborel* (λx. *normal-density* μ σ *x* * |*x* − μ| ^(2 * *k* + 1)) = 2 ^ *k* * σ ^ (2 * *k* + 1) * *fact k* * *sqrt* (2 / *pi*)
using *normal-moment-abs-odd* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integral-normal-moment-odd*:
$\mathit{integral}^L$ *lborel* (λx. *normal-density* μ σ *x* * (*x* − μ) ^(2 * *k* + 1)) = 0

using *normal-moment-odd* **by** (*rule has-bochner-integral-integral-eq*)

end

context
fixes σ :: *real*
assumes σ-*pos*[*arith*]: $0 < \sigma$
begin

lemma *normal-moment-nz-1*: *has-bochner-integral lborel* (λx. *normal-density* μ σ $x * x$) μ
proof −
note *normal-moment-even*[*OF* σ-*pos*, *of* μ *0*]
note *normal-moment-odd*[*OF* σ-*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* (λx. *normal-density* μ σ $x * x$) = μ
using *normal-moment-nz-1* **by** (*rule has-bochner-integral-integral-eq*)

lemma *integrable-normal-moment-nz-1*: *integrable lborel* (λx. *normal-density* μ σ $x * x$)
using *normal-moment-nz-1* **by** *rule*

lemma *integrable-normal-moment*: *integrable lborel* (λx. *normal-density* μ σ $x *$ $(x - \mu)$ ^ 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* (λx. *normal-density* μ σ x $* |x - \mu|$ ^ 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*. *normal-density* μ σ *x* ∂*lborel*) = *1*
using *integral-normal-moment-even*[*of* σ μ *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* :: *nat*
begin

lemma *std-normal-moment-even*:
has-bochner-integral lborel (λ*x*. *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 (λ*x*. *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 (λ*x*. *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* (λ*x*. *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* (λ*x*. *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* (λ*x*. *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* (λ*x*. *std-normal-density*

$x * |x| \hat{} k$)
using *integrable-normal-moment-abs*[*of 1 0 k*] **by** *simp*

lemma *integrable-std-normal-moment*: *integrable lborel* (λx. *std-normal-density x* $*$ *x*$\hat{}$*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* μ σ)
assumes [*simp*, *arith*]: $0 < \sigma$ $\alpha \neq 0$
shows *distributed M lborel* (λx. $\beta + \alpha * X\ x$) (*normal-density* ($\beta + \alpha * \mu$) ($|\alpha| * \sigma$))
proof −
have *eq*: $\bigwedge x$. $|\alpha| *$ *normal-density* ($\beta + \alpha * \mu$) ($|\alpha| * \sigma$) ($x * \alpha + \beta$) =
normal-density μ σ *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* - ‹$\alpha \neq 0$›, **where** *t*=β])
(*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* μ σ) = *distributed M lborel* (λx. (*X x* − μ) / σ) *std-normal-density*
proof *auto*
assume *distributed M lborel X* (λx. *ennreal* (*normal-density* μ σ *x*))
then have *distributed M lborel* (λx. $-\mu$ / σ + (*1*/σ) $*$ *X x*) (λx. *ennreal* (*normal-density* ($-\mu$ / σ + (*1*/σ)$*$ μ) (|*1*/σ| $*$ σ) *x*))
by(*rule normal-density-affine*) *auto*

then show *distributed M lborel* (λx. (*X x* − μ) / σ) (λx. *ennreal* (*std-normal-density x*))
by (*simp add*: *diff-divide-distrib*[*symmetric*] *field-simps*)
next
assume $*$: *distributed M lborel* (λx. (*X x* − μ) / σ) (λx. *ennreal* (*std-normal-density x*))
have *distributed M lborel* (λx. $\mu + \sigma * ((X\ x - \mu) / \sigma)$) ($\lambda x$. *ennreal* (*normal-density* μ |σ| *x*))
using *normal-density-affine*[*OF* $*$, *of* σ μ] **by** *simp*
then show *distributed M lborel X* (λx. *ennreal* (*normal-density* μ σ *x*)) **by** *simp*
qed

lemma *conv-normal-density-zero-mean*:
assumes [*simp*, *arith*]: $0 < \sigma$ $0 < \tau$
shows (λx. $\int^+ y$. *ennreal* (*normal-density 0* σ (x − y) $*$ *normal-density 0* τ y)

∂lborel) =
  normal-density 0 (*sqrt* (σ^2 + τ^2)) (**is** *?LHS* = *?RHS*)
proof −
 def $\sigma' \equiv \sigma^2$ **and** $\tau' \equiv \tau^2$
 then have [*simp*, *arith*]: *0* < σ' *0* < τ'
  by *simp-all*
 let *?σ* = *sqrt* ((σ' * τ') / (σ' + τ'))
 have *sqrt*: (*sqrt* (*2* * *pi* * (σ' + τ')) * *sqrt* (*2* * *pi* * (σ' * τ') / (σ' + τ'))) =
  (*sqrt* (*2* * *pi* * σ') * *sqrt* (*2* * *pi* * τ'))
  by (*subst power-eq-iff-eq-base*[*symmetric*, **where** *n*=*2*])
   (*simp-all add*: *real-sqrt-mult*[*symmetric*] *power2-eq-square*)
 have *?LHS* =
  (λ*x*. ∫⁺*y*. *ennreal*((*normal-density 0* (*sqrt* (σ' + τ')) *x*) * *normal-density* (τ' * *x* / (σ' + τ')) *?σ y*) *∂lborel*)
  apply (*intro ext nn-integral-cong*)
  apply (*simp add*: *normal-density-def σ'-def*[*symmetric*] *τ'-def*[*symmetric*] *sqrt mult-exp-exp*)
  apply (*simp add*: *divide-simps power2-eq-square*)
  apply (*simp add*: *field-simps*)
  done

 also have ... =
  (λ*x*. (*normal-density 0* (*sqrt* (σ^2 + τ^2)) *x*) * ∫⁺*y*. *ennreal*(*normal-density* (τ^2* *x* / (σ^2 + τ^2)) *?σ y*) *∂lborel*)
  by (*subst nn-integral-cmult*[*symmetric*])
   (*auto simp*: *σ'-def τ'-def normal-density-def ennreal-mult'*[*symmetric*])

 also have ... = (λ*x*. (*normal-density 0* (*sqrt* (σ^2 + τ^2)) *x*))
  by (*subst nn-integral-eq-integral*) (*auto simp*: *normal-density-nonneg*)

 finally show *?thesis* **by** *fast*
qed

lemma *conv-std-normal-density*:
 (λ*x*. ∫⁺*y*. *ennreal* (*std-normal-density* (*x* − *y*) * *std-normal-density y*) *∂lborel*) =
 (*normal-density 0* (*sqrt 2*))
 by (*subst conv-normal-density-zero-mean*) *simp-all*

lemma (**in** *prob-space*) *sum-indep-normal*:
 assumes *indep*: *indep-var borel X borel Y*
 assumes *pos-var*[*arith*]: *0* < σ *0* < τ
 assumes *normalX*[*simp*]: *distributed M lborel X* (*normal-density* μ σ)
 assumes *normalY*[*simp*]: *distributed M lborel Y* (*normal-density* ν τ)
 shows *distributed M lborel* (λ*x*. *X x* + *Y x*) (*normal-density* (μ + ν) (*sqrt* (σ^2 + τ^2)))
proof −
 have *ind*[*simp*]: *indep-var borel* (λ*x*. − μ + *X x*) *borel* (λ*x*. − ν + *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* σ)
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* τ)
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* $(-\ \nu)$ τ x)) **by** *simp*

have *distributed M lborel* $(\lambda x.\ X\ x + (-\ Y\ x))$ (*normal-density* $(\mu + -\ \nu)$ (*sqrt* $(\sigma^2 + \tau^2)$))
apply (*rule sum-indep-normal*)
apply (*rule indep-var-compose*[*unfolded comp-def*, *of borel - borel -* $\lambda x.\ x$ *-* $\lambda x.$ $-\ x$])

apply *simp-all*
done
then show *?thesis* **by** *simp*
qed

lemma (**in** *prob-space*) *setsum-indep-normal*:
assumes *finite I I* ≠ {} *indep-vars* (λ*i*. *borel*) *X I*
assumes ⋀*i*. *i* ∈ *I* ⟹ 0 < σ *i*
assumes *normal*: ⋀*i*. *i* ∈ *I* ⟹ *distributed M lborel* (*X i*) (*normal-density* (μ *i*) (σ *i*))
shows *distributed M lborel* (λ*x*. ∑ *i*∈*I*. *X i x*) (*normal-density* (∑ *i*∈*I*. μ *i*) (*sqrt* (∑ *i*∈*I*. (σ *i*)²)))
using *assms*
proof (*induct I rule*: *finite-ne-induct*)
case (*singleton i*) **then show** *?case* **by** (*simp add* : *abs-of-pos*)
next
case (*insert i I*)
then have *1*: *distributed M lborel* (λ*x*. (*X i x*) + (∑ *i*∈*I*. *X i x*))
(*normal-density* (μ *i* + *setsum* μ *I*) (*sqrt* ((σ *i*)² + (*sqrt* (∑ *i*∈*I*. (σ *i*)²))²)))
apply (*intro sum-indep-normal indep-vars-setsum insert real-sqrt-gt-zero*)
apply (*auto intro*: *indep-vars-subset intro*!: *setsum-pos*)
apply *fastforce*
done
have *2*: (λ*x*. (*X i x*)+ (∑ *i*∈*I*. *X i x*)) = (λ*x*. (∑ *j*∈*insert i I*. *X j x*))
μ *i* + *setsum* μ *I* = *setsum* μ (*insert i I*)
using *insert* **by** *auto*

have *3*: (*sqrt* ((σ *i*)² + (*sqrt* (∑ *i*∈*I*. (σ *i*)²))²)) = (*sqrt* (∑ *i*∈*insert i I*. (σ *i*)²))
using *insert* **by** (*simp add*: *setsum-nonneg*)

show *?case* **using** *1 2 3* **by** *simp*
qed

lemma (**in** *prob-space*) *standard-normal-distributed-expectation*:
assumes *D*: *distributed M lborel X std-normal-density*
shows *expectation X* = *0*
using *integral-std-normal-moment-odd*[*of 0*]
distributed-integral[*OF D*, *of* λ*x*. *x*, *symmetric*]
by (*auto simp*:)

lemma (**in** *prob-space*) *normal-distributed-expectation*:
assumes σ[*arith*]: *0* < σ
assumes *D*: *distributed M lborel X* (*normal-density* μ σ)
shows *expectation X* = μ
using *integral-normal-moment-nz-1*[*OF* σ, *of* μ] *distributed-integral*[*OF D*, *of* λ*x*. *x*, *symmetric*]
by (*auto simp*: *field-simps*)

lemma (**in** *prob-space*) *normal-distributed-variance*:
fixes *a b* :: *real*
assumes [*simp*, *arith*]: $0 < \sigma$
assumes *D*: *distributed M lborel X* (*normal-density* μ σ)
shows *variance X* = σ^2
using *integral-normal-moment-even*[*of* σ μ *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 $\Longrightarrow$ *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* μ σ)
shows *entropy b lborel X* = *log b* (*2* * *pi* * *exp 1* * σ^2) / *2*
proof −
have *entropy b lborel X* = − (∫ *x*. *normal-density* μ σ *x* * *log b* (*normal-density* μ σ *x*) ∂*lborel*)
using *D* **by** (*rule entropy-distr*) *simp*
also have ... = − (∫ *x*. *normal-density* μ σ *x* * (− *ln* (*2* * *pi* * σ^2) − (*x* − μ)^2 / σ^2) / (*2* * *ln b*) ∂*lborel*)
by (*intro arg-cong*[**where** *f*=*uminus*] *integral-cong*)
(*auto simp*: *normal-density-def field-simps ln-mult log-def ln-div ln-sqrt*)
also have ... = − (∫ *x*. − (*normal-density* μ σ *x* * (*ln* (*2* * *pi* * σ^2)) + (*normal-density* μ σ *x* * (*x* − μ)^2) / σ^2) / (*2* * *ln b*) ∂*lborel*)
by (*intro arg-cong*[**where** *f*=*uminus*] *integral-cong*) (*auto simp*: *divide-simps field-simps*)
also have ... = (∫ *x*. *normal-density* μ σ *x* * (*ln* (*2* * *pi* * σ^2)) + (*normal-density* μ σ *x* * (*x* − μ)^2) / σ^2 ∂*lborel*) / (*2* * *ln b*)
by (*simp del*: *minus-add-distrib*)
also have ... = (*ln* (*2* * *pi* * σ^2) + *1*) / (*2* * *ln b*)
using *integral-normal-moment-even*[*of* σ μ *1*] **by** (*simp add*: *integrable-normal-moment fact-numeral*)
also have ... = *log b* (*2* * *pi* * *exp 1* * σ^2) / *2*
by (*simp add*: *log-def field-simps ln-mult*)
finally show *?thesis* **.**
qed

end

42 Characteristic Functions

theory *Characteristic-Functions*
imports *Weak-Convergence Interval-Integral Independent-Family Distributions*
begin

lemma *mult-min-right*: $a \geq 0 \Longrightarrow (a :: real) * min\ b\ c = min\ (a * b)\ (a * c)$
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*]: $\bigwedge n.\ n \geq n\text{-}e \Longrightarrow P\ (2 * n)$
by (*auto simp*: *eventually-sequentially*)
moreover
from *O* **obtain** *n-o* **where** [*intro*]: $\bigwedge n.\ n \geq n\text{-}o \Longrightarrow P\ (Suc\ (2 * n))$
by (*auto simp*: *eventually-sequentially*)
show *?thesis*
unfolding *eventually-sequentially*
proof (*intro exI allI impI*)
fix *n* **assume** $max\ (2 * n\text{-}e)\ (2 * n\text{-}o + 1) \leq n$ **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* :: *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*)

42.1 Application of the FTC: integrating $e^i x$

abbreviation *iexp* :: *real* ⇒ *complex* **where**
iexp ≡ ($\lambda x.\ exp\ (\mathrm{i} * complex\text{-}of\text{-}real\ x)$)

lemma *isCont-iexp* [*simp*]: *isCont iexp x*
by (*intro continuous-intros*)

lemma *has-vector-derivative-iexp*[*derivative-intros*]:
(*iexp has-vector-derivative* i ∗ *iexp x*) (*at x 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* :: *real*
shows (*CLBINT x*=*a*..*b*. *iexp x*) = *ii* ∗ *iexp a* − *ii* ∗ *iexp b*
by (*subst interval-integral-FTC-finite* [**where** $F = \lambda x.\ -ii * iexp\ x$])
(*auto intro*!: *derivative-eq-intros continuous-intros*)

42.2 The Characteristic Function of a Real Measure.

definition
char :: *real measure* ⇒ *real* ⇒ *complex*
where

*char M t = CLINT x|M. iexp (t * x)*

lemma (**in** *real-distribution*) *char-zero*: *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 ∈ borel-measurable M* ⋀*x. Im (f x) = 0*
shows *integrable M* (λ*x. exp* (*ii* * (*f x*)))
proof (*intro integrable-const-bound* [*of - 1*])
from *f* **have** ⋀*x. of-real* (*Re* (*f x*)) = *f x*
by (*simp add*: *complex-eq-iff*)
then show *AE x in M. cmod* (*exp* (i * *f x*)) ≤ *1*
using *norm-exp-ii-times*[*of Re* (*f x*) **for** *x*] **by** *simp*
qed (*insert f*, *simp*)

lemma (**in** *real-distribution*) *cmod-char-le-1*: *norm* (*char M t*) ≤ *1*
proof −
have *norm* (*char M t*) ≤ (∫ *x. norm* (*iexp* (*t* * *x*)) ∂*M*)
unfolding *char-def* **by** (*intro integral-norm-bound integrable-iexp*) *auto*
also have ... ≤ *1*
by (*simp del*: *of-real-mult*)
finally show *?thesis* **.**
qed

lemma (**in** *real-distribution*) *isCont-char*: *isCont* (*char M*) *t*
unfolding *continuous-at-sequentially*
proof *safe*
fix *X* **assume** *X*: *X* ⟶ *t*
show (*char M* ∘ *X*) ⟶ *char M t*
unfolding *comp-def char-def*
by (*rule integral-dominated-convergence*[**where** *w*=λ-. *1*]) (*auto intro*!: *tendsto-intros X*)
qed

lemma (**in** *real-distribution*) *char-measurable* [*measurable*]: *char M ∈ borel-measurable borel*
by (*auto intro*!: *borel-measurable-continuous-on1 continuous-at-imp-continuous-on isCont-char*)

42.3 Independence

lemma (**in** *prob-space*) *char-distr-sum*:
fixes *X1 X2* :: *'a* ⇒ *real* **and** *t* :: *real*
assumes *indep-var borel X1 borel X2*
shows *char* (*distr M borel* (λω. *X1* ω + *X2* ω)) *t* =
char (*distr M borel X1*) *t* * *char* (*distr M 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 *char* (*distr M borel* (λω. *X1* ω + *X2* ω)) *t* = (*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-setsum*:
indep-vars (λ*i*. *borel*) *X A* ⟹
char (*distr M borel* (λω. ∑*i*∈*A*. *X i* ω)) *t* = (∏*i*∈*A*. *char* (*distr M borel* (*X* *i*)) *t*)
proof (*induct A rule*: *infinite-finite-induct*)
case (*insert x F*) **with** *indep-vars-subset*[*of* λ-. *borel X insert x F F*] **show** *?case*
by (*auto simp add*: *char-distr-sum indep-vars-setsum*)
qed (*simp-all add*: *char-def integral-distr prob-space del*: *distr-const*)

42.4 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* ≡ *complex-of-real* ((*x* − *s*) ^ *m*)
shows (*CLBINT s=0..x*. *f s n* ∗ *iexp s*) =
x^*Suc n* / *Suc n* + (*ii* / *Suc n*) ∗ (*CLBINT s=0..x*. *f s* (*Suc n*) ∗ *iexp s*)
proof −
have *1*:
((λ*s*. *complex-of-real*(−((*x* − *s*) ^ (*Suc n*) / (*Suc n*))) ∗ *iexp s*)
has-vector-derivative complex-of-real((*x* − *s*)^*n*) ∗ *iexp s* + (*ii* ∗ *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* = λ*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* + (*ii* ∗ *iexp s*) ∗ −(*f s* (*Suc n*) / (*Suc n*)))) (**is** *?LHS* = *?RHS*)
proof −
have *?RHS* = (*CLBINT s=0..x*. (*f s n* ∗ *iexp s* + (*ii* ∗ *iexp s*) ∗
complex-of-real(−((*x* − *s*) ^ (*Suc n*) / (*Suc n*)))))
by (*cases* 0 ≤ *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. (ii * x)^k / (fact k)) + ((ii ^ (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 setsum-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. (ii*x)^k/fact k) + ii^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)

  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: ii ^ (Suc (Suc n)) / (fact (Suc n)) = (ii ^ (Suc n) / (fact n)) * (ii
/ (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-field-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 ... = *x* ^ (*Suc n*) / (*Suc n*) **by** *simp*
finally show *?thesis* **.**
qed
finally show *?thesis* **.**
qed

lemma *iexp-approx1*: *cmod* (*iexp x* − (∑ *k* ≤ *n*. (*ii* ∗ *x*) ^ *k* / *fact k*)) ≤ |*x*| ^ (*Suc n*) / *fact* (*Suc n*)
proof −
have *iexp x* − (∑ *k* ≤ *n*. (*ii* ∗ *x*) ^ *k* / *fact k*) =
((*ii* ^ (*Suc n*)) / (*fact n*)) ∗ (*CLBINT s=0..x*. (*x* − *s*) ^ *n* ∗ (*iexp s*)) (**is** *?t1* = *?t2*)
by (*subst iexp-eq1* [*of* - *n*], *simp add*: *field-simps*)
then have *cmod* (*?t1*) = *cmod* (*?t2*)
by *simp*
also have ... = (*1* / *of-nat* (*fact n*)) ∗ *cmod* (*CLBINT s=0..x*. (*x* − *s*) ^ *n* ∗ (*iexp s*))
by (*simp add*: *norm-mult norm-divide norm-power*)
also have ... ≤ (*1* / *of-nat* (*fact n*)) ∗ |*LBINT s=0..x*. *cmod* ((*x* − *s*) ^ *n* ∗ (*iexp s*))|
by (*intro mult-left-mono interval-integral-norm2*)
(*auto simp*: *zero-ereal-def intro*: *interval-integrable-isCont*)
also have ... ≤ (*1* / *of-nat* (*fact n*)) ∗ |*LBINT s=0..x*. |(*x* − *s*) ^ *n*||
by (*simp add*: *norm-mult del*: *of-real-diff of-real-power*)
also have ... ≤ (*1* / *of-nat* (*fact n*)) ∗ |*x* ^ (*Suc n*) / (*Suc n*)|
by (*simp add*: *abs-LBINT-I0c-abs-power-diff*)
also have *1* / *real-of-nat* (*fact n*::*nat*) ∗ |*x* ^ *Suc n* / *real* (*Suc n*)| =
|*x*| ^ *Suc n* / *fact* (*Suc n*)
by (*simp add*: *abs-mult power-abs*)
finally show *?thesis* **.**
qed

lemma *iexp-approx2*: *cmod* (*iexp x* − (∑ *k* ≤ *n*. (*ii* ∗ *x*) ^ *k* / *fact k*)) ≤ *2* ∗ |*x*| ^ *n* / *fact n*
proof (*induction n*) — really cases
case (*Suc n*)
have ∗: ⋀*a b*. *interval-lebesgue-integrable lborel a b f* ⟹ *interval-lebesgue-integrable lborel a b g* ⟹
|*LBINT s=a..b*. *f s*| ≤ |*LBINT s=a..b*. *g s*|
if *f*: ⋀*s*. *0* ≤ *f s* **and** *g*: ⋀*s*. *f s* ≤ *g s* **for** *f g* :: - ⇒ *real*
using *order-trans*[*OF f g*] *f g* **unfolding** *interval-lebesgue-integral-def interval-lebesgue-integrable-def*
by (*auto simp*: *integral-nonneg-AE*[*OF AE-I2*] *intro*!: *integral-mono mult-mono*)

have *iexp x* − (∑ *k* ≤ *Suc n*. (*ii* ∗ *x*) ^ *k* / *fact k*) =
((*ii* ^ (*Suc n*)) / (*fact n*)) ∗ (*CLBINT s=0..x*. (*x* − *s*) ^ *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 *... = (1 / (fact n)) * cmod (CLBINT s=0..x. (x − s) ^ n * (iexp s − 1))*
by (*simp add*: *norm-mult norm-divide norm-power*)
also have *... ≤ (1 / (fact n)) * |LBINT s=0..x. cmod ((x − s) ^ n * (iexp s − 1))|*
by (*intro mult-left-mono interval-integral-norm2*)
(*auto intro*!: *interval-integrable-isCont simp*: *zero-ereal-def*)
also have *... = (1 / (fact n)) * |LBINT s=0..x. abs ((x − s) ^ n) * cmod((iexp s − 1))|*
by (*simp add*: *norm-mult del*: *of-real-diff of-real-power*)
also have *... ≤ (1 / (fact n)) * |LBINT s=0..x. abs ((x − s) ^ 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 *... = (2 / (fact n)) * |x ^ (Suc n) / (Suc n)|*
by (*simp add*: *abs-LBINT-I0c-abs-power-diff abs-mult*)
also have *2 / fact n * |x ^ Suc n / real (Suc n)| = 2 * |x| ^ 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*: ⋀*k*. *k ≤ n ⟹ integrable M (λx. x ^ k)*
shows *cmod (char M t − (∑ k ≤ n. ((ii * t) ^ k / fact k) * expectation (λx. x ^ k))) ≤*
(2|t| ^ n / fact n) * expectation (λx. |x| ^ n)* (**is** *cmod (char M t − ?t1) ≤ -*)
proof −
have *integ-iexp*: *integrable M (λx. iexp (t * x))*
by (*intro integrable-const-bound*) *auto*

def *c ≡ λk x. (ii * t) ^ k / fact k * complex-of-real (x ^ k)*
have *integ-c*: ⋀*k*. *k ≤ n ⟹ integrable M (λx. c k x)*
unfolding *c-def* **by** (*intro integrable-mult-right integrable-of-real integrable-moments*)

have *k ≤ n ⟹ expectation (c k) = (𝗂*t) ^ k * (expectation (λx. x ^ 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. (∑ k ≤ n. c k x)))*
by (*simp add*: *integ-c*)
also have *... = norm ((CLINT x | M. iexp (t * x) − (∑ k ≤ n. c k x)))*
unfolding *char-def* **by** (*subst integral-diff* [*OF integ-iexp*]) (*auto intro*!: *integ-c*)
also have *... ≤ expectation (λx. cmod (iexp (t * x) − (∑ k ≤ n. c k x)))*
by (*intro integral-norm-bound integrable-diff integ-iexp integrable-setsum integ-c*) *simp*
also have *... ≤ expectation (λx. 2 * |t| ^ n / fact n * |x| ^ n)*
proof (*rule integral-mono*)

```
    show integrable M (λx. cmod (iexp (t * x) - (∑ k≤n. c k x)))
      by (intro integrable-norm integrable-diff integ-iexp integrable-setsum integ-c)
simp
    show integrable M (λx. 2 * |t| ^ n / fact n * |x| ^ n)
      unfolding power-abs[symmetric]
      by (intro integrable-mult-right integrable-abs integrable-moments) simp
    show cmod (iexp (t * x) - (∑ k≤n. c k x)) ≤ 2 * |t| ^ n / fact n * |x| ^ 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: ⋀k. k ≤ n ⟹ integrable M (λx. x ^ k)
  shows cmod (char M t - (∑ k ≤ n. ((ii * t) ^k / fact k) * expectation (λx.
x ^k))) ≤
    (|t| ^n / fact (Suc n)) * expectation (λx. min (2 * |x| ^n * Suc n) (|t| * |x| ^Suc
n))
    (is cmod (char M t - ?t1) ≤ -)
proof -
  have integ-iexp: integrable M (λx. iexp (t * x))
    by (intro integrable-const-bound) auto

  def c ≡ λk x. (ii * t) ^k / fact k * complex-of-real (x ^k)
  have integ-c: ⋀k. k ≤ n ⟹ integrable M (λx. c k x)
   unfolding c-def by (intro integrable-mult-right integrable-of-real integrable-moments)

  have *: min (2 * |t * x| ^n / fact n) (|t * x| ^Suc n / fact (Suc n)) =
      |t| ^n / fact (Suc n) * min (2 * |x| ^n * real (Suc n)) (|t| * |x| ^(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 ≤ n ⟹ expectation (c k) = (i*t) ^ k * (expectation (λx. x ^ 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. (∑ k
≤ n. c k x)))
    by (simp add: integ-c)
  also have ... = norm ((CLINT x | M. iexp (t * x) - (∑ k ≤ n. c k x)))
   unfolding char-def by (subst integral-diff[OF integ-iexp]) (auto intro!: integ-c)
  also have ... ≤ expectation (λx. cmod (iexp (t * x) - (∑ k ≤ n. c k x)))
   by (intro integral-norm-bound integrable-diff integ-iexp integrable-setsum integ-c)
simp
```

```
  also have ... ≤ expectation (λx. min (2 * |t * x| ^ n / fact n) (|t * x| ^ (Suc n)
/ fact (Suc n)))
    (is - ≤ expectation ?f)
  proof (rule integral-mono)
    show integrable M (λx. cmod (iexp (t * x) - (∑ k≤n. c k x)))
      by (intro integrable-norm integrable-diff integ-iexp integrable-setsum integ-c)
simp
    show integrable M ?f
      by (rule integrable-bound[where f=λx. 2 * |t * x| ^ n / fact n])
        (auto simp: integrable-moments power-abs[symmetric] power-mult-distrib)
    show cmod (iexp (t * x) - (∑ k≤n. c k x)) ≤ ?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 / fact (Suc n)) * expectation (λx. min (2 * |x| ^ n * Suc
n) (|t| * |x| ^ Suc n))
    unfolding *
  proof (rule integral-mult-right)
    show integrable M (λx. min (2 * |x| ^ n * real (Suc n)) (|t| * |x| ^ Suc n))
      by (rule integrable-bound[where f=λx. 2 * |x| ^ n * real (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 (λx. x) and
    integral-1: expectation (λx. x) = 0 and
    integrable-2: integrable M (λx. x^2) and
    integral-2: variance (λx. x) = σ2
  shows cmod (char M t - (1 - t^2 * σ2 / 2)) ≤
    (t^2 / 6) * expectation (λx. min (6 * x^2) (abs t * (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 (λx. x * x) = σ2
    by (simp add: integral-1 numeral-eq-Suc)
  have 1: k ≤ 2 ⟹ integrable M (λx. x^k) for k
    using assms by (auto simp: eval-nat-numeral le-Suc-eq)
  note char-approx1
  note 2 = char-approx1 [of 2 t, OF 1, simplified]
  have cmod (char M t - (∑ k≤2. (i * t) ^ k * (expectation (λx. x ^ k)) / (fact
k))) ≤
      t^2 * expectation (λx. min (6 * x^2) (|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 - t^2
```

$* \sigma 2 \ / \ 2$
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 $t^2 *$ *expectation* $(\lambda x.\ min\ (6 * x^2)\ (|t| * |x|\ \hat{}\ 3))\ /\ 6 =$
$t^2\ /\ 6 *$ *expectation* $(\lambda x.\ min\ (6 * x^2)\ (|t| * |x|\ \hat{}\ 3))$ **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* $(\lambda x.\ (X\ x)\ \hat{}\ 2)$ *expectation X = 0*
and *var-X*: *variance X* $= \sigma 2$
and μ*-def*: μ *= distr M borel X*
shows *cmod* (*char* μ *t* $-$ $(1 - t\ \hat{}\ 2 * \sigma 2\ /\ 2)) \leq$
$(t\ \hat{}\ 2\ /\ 6) *$ *expectation* $(\lambda x.\ min\ (6 * (X\ x)\ \hat{}\ 2)\ (|t| * |X\ x|\ \hat{}\ 3))$
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* : $\bigwedge k.\ k \leq n \Longrightarrow$ *integrable M* $(\lambda x.\ X\ x\ \hat{}\ k)$
and *rv-X*[*measurable*]: *random-variable borel X*
and μ*-distr* : *distr M borel X* $= \mu$
shows *cmod* (*char* μ *t* $-$ $(\sum k \leq n.\ ((ii * t)\ \hat{}\ k\ /\ fact\ k) *$ *expectation* $(\lambda x.\ (X\ x)\ \hat{}\ k))) \leq$
$(2 * |t|\ \hat{}\ n\ /\ fact\ n) *$ *expectation* $(\lambda x.\ |X\ x|\ \hat{}\ n)$
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

42.5 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 :: *nat*
shows (*LINT x|std-normal-distribution. x^(2 * k)*) = *fact (2 * k) / (2^k * fact k*)
and *integrable std-normal-distribution* (*λx. 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* (*λx. x^k*)
by (*auto simp*: *normal-density-nonneg integrable-std-normal-moment integrable-density*)

lemma *integral-std-normal-distribution-moment-odd*:
odd k ⟹ *integral*^{L} *std-normal-distribution* (*λx. 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 :: *nat*
shows (*LINT x|std-normal-distribution. |x| ^(2 * k)*) = *fact (2 * k) / (2^k * 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 :: *nat*
shows (*LINT x|std-normal-distribution. |x| ^(2 * k + 1)*) = *sqrt (2 / pi) * 2 ^ k * 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 = (*λt. complex-of-real* (*exp* (− (*t^2*) / *2*)))
proof (*intro ext LIMSEQ-unique*)
fix t :: *real*
let *?f' = λk.* (*ii * t*) *^k / fact k * (LINT x | std-normal-distribution. x^k*)
let *?f = λn.* (∑ *k* ≤ *n. ?f' k*)
show *?f* ⟶ *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-setsum
      by (intro setsum.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: inverse-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 * t^2]]) 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
```

43 Helly's selection theorem

The set of bounded, monotone, right continuous functions is sequentially compact

```
theory Helly-Selection
  imports ~~/src/HOL/Library/Diagonal-Subsequence Weak-Convergence
begin

lemma minus-one-less: x − 1 < (x::real)
  by simp

theorem Helly-selection:
  fixes f :: nat ⇒ real ⇒ real
  assumes rcont: ⋀n x. continuous (at-right x) (f n)
  assumes mono: ⋀n. mono (f n)
  assumes bdd: ⋀n x. |f n x| ≤ M
  shows ∃ s. subseq s ∧ (∃ F. (∀ x. continuous (at-right x) F) ∧ mono F ∧ (∀ x.
|F x| ≤ M) ∧
    (∀ x. continuous (at x) F ⟶ (λn. f (s n) x) ⟶⟶ F x))
proof −
  obtain m :: real ⇒ nat where bij-betw m ℚ UNIV
    using countable-rat Rats-infinite by (erule countableE-infinite)
  then obtain r :: nat ⇒ real where bij: bij-betw r UNIV ℚ
    using bij-betw-inv by blast

  have dense-r: ⋀x y. x < y ⟹ ∃ n. x < r n ∧ r n < y
    by (metis Rats-dense-in-real bij f-the-inv-into-f bij-betw-def)

  let ?P = λn. λs. convergent (λk. f (s k) (r n))
  interpret nat: subseqs ?P
  proof (unfold convergent-def, unfold subseqs-def, auto)
    fix n :: nat and s :: nat ⇒ nat assume s: subseq s
    have bounded {−M..M}
      using bounded-closed-interval by auto
    moreover have ⋀k. f (s k) (r n) ∈ {−M..M}
      using bdd by (simp add: abs-le-iff minus-le-iff)
    ultimately have ∃ l s'. subseq s' ∧ ((λk. f (s k) (r n)) ∘ s') ⟶⟶ l
      using compact-Icc compact-imp-seq-compact seq-compactE by metis
    thus ∃ s'. subseq s' ∧ (∃ l. (λk. f (s (s' k)) (r n)) ⟶⟶ l)
      by (auto simp: comp-def)
```

qed
def $d \equiv nat.diagseq$
have *subseq*: *subseq d*
unfolding *d-def* **using** *nat.subseq-diagseq* **by** *auto*
have *rat-cnv*: *?P n d* **for** *n*
proof −
have *Pn-seqseq*: *?P n* (*nat.seqseq* (*Suc n*))
by (*rule nat.seqseq-holds*)
have *1*: (λ*k*. *f* ((*nat.seqseq* (*Suc n*) ∘ (λ*k*. *nat.fold-reduce* (*Suc n*) *k*
(*Suc n* + *k*))) *k*) (*r n*)) = (λ*k*. *f* (*nat.seqseq* (*Suc n*) *k*) (*r n*)) ∘
(λ*k*. *nat.fold-reduce* (*Suc n*) *k* (*Suc n* + *k*))
by *auto*
have *2*: *?P n* (*d* ∘ (*op* + (*Suc n*)))
unfolding *d-def nat.diagseq-seqseq 1*
by (*intro convergent-subseq-convergent Pn-seqseq nat.subseq-diagonal-rest*)
then obtain *L* **where** *3*: (λ*na*. *f* (*d* (*na* + *Suc n*)) (*r n*)) ⟶ *L*
by (*auto simp*: *add.commute dest*: *convergentD*)
then have (λ*k*. *f* (*d k*) (*r n*)) ⟶ *L*
by (*rule LIMSEQ-offset*)
then show *?thesis*
by (*auto simp*: *convergent-def*)
qed
let *?f* = λ*n*. λ*k*. *f* (*d k*) (*r n*)
have *lim-f*: *?f n* ⟶ *lim* (*?f n*) **for** *n*
using *rat-cnv convergent-LIMSEQ-iff* **by** *auto*
have *lim-bdd*: *lim* (*?f n*) ∈ {−*M*..*M*} **for** *n*
proof −
have *closed* {−*M*..*M*} **using** *closed-real-atLeastAtMost* **by** *auto*
hence (∀ *i*. *?f n i* ∈ {−*M*..*M*}) ∧ *?f n* ⟶ *lim* (*?f n*) ⟶ *lim* (*?f n*) ∈
{−*M*..*M*}
unfolding *closed-sequential-limits* **by** (*drule-tac x* = λ*k*. *f* (*d k*) (*r n*) **in**
spec) *blast*
moreover have ∀ *i*. *?f n i* ∈ {−*M*..*M*}
using *bdd* **by** (*simp add*: *abs-le-iff minus-le-iff*)
ultimately show *lim* (*?f n*) ∈ {−*M*..*M*}
using *lim-f* **by** *auto*
qed
then have *limset-bdd*: ⋀*x*. {*lim* (*?f n*) |*n*. *x* < *r n*} ⊆ {−*M*..*M*}
by *auto*
then have *bdd-below*: *bdd-below* {*lim* (*?f n*) |*n*. *x* < *r n*} **for** *x*
by (*metis* (*mono-tags*) *bdd-below-Icc bdd-below-mono*)
have *r-unbdd*: ∃ *n*. *x* < *r n* **for** *x*
using *dense-r*[*OF less-add-one*, *of x*] **by** *auto*
then have *nonempty*: {*lim* (*?f n*) |*n*. *x* < *r n*} ≠ {} **for** *x*
by *auto*

def *F* ≡ λ*x*. *Inf* {*lim* (*?f n*) |*n*. *x* < *r n*}
have *F-eq*: *ereal* (*F x*) = (*INF n*:{*n*. *x* < *r n*}. *ereal* (*lim* (*?f n*))) **for** *x*
unfolding *F-def* **by** (*subst ereal-Inf'*[*OF bdd-below nonempty*]) (*simp add*:

setcompr-eq-image)
have *mono-F*: *mono F*
using *nonempty* **by** (*auto intro*!: *cInf-superset-mono simp*: *F-def bdd-below mono-def*)
moreover have $\bigwedge x.$ *continuous* (*at-right x*) *F*
unfolding *continuous-within order-tendsto-iff eventually-at-right*[*OF less-add-one*]
proof *safe*
show $F\ x < u \Longrightarrow \exists\, b{>}x.\ \forall\, y{>}x.\ y < b \longrightarrow F\ y < u$ **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*: $l < F\ x$ **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 $|F\ x| \le M$ **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 *limsup* $(\lambda n.\ f\ (d\ n)\ x) \le F\ x$
proof (*rule tendsto-le-const*)
show $(F \longrightarrow ereal\ (F\ x))$ (*at-right x*)
using *cts* **unfolding** *continuous-at-split* **by** (*auto simp*: *continuous-within*)
show $\forall_F\ i$ *in at-right x. limsup* $(\lambda n.\ f\ (d\ n)\ x) \le F\ i$
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 $*$: $y < r\ n' \Longrightarrow lim\ (?f\ N) \le lim\ (?f\ n')$ **for** n'
using ‹$r\ N < y$› **by** (*intro LIMSEQ-le*[*OF lim-f lim-f*]) (*auto intro*!: *mono*[*THEN monoD*])
have *limsup* $(\lambda n.\ f\ (d\ n)\ x) \le limsup\ (?f\ N)$
using ‹$x < r\ N$› **by** (*auto intro*!: *Limsup-mono always-eventually mono*[*THEN monoD*])
also have $\ldots = lim\ (\lambda n.\ ereal\ (?f\ N\ n))$
using *rat-cnv*[*of N*] **by** (*force intro*!: *convergent-limsup-cl simp*: *convergent-def*)
also have $\ldots \le F\ y$
by (*auto intro*!: *INF-greatest* $*$ *simp*: *convergent-real-imp-convergent-ereal rat-cnv F-eq*)
finally show *limsup* $(\lambda n.\ f\ (d\ n)\ x) \le F\ y$ **.**
qed
qed *simp*
show $F\ x \le liminf\ (\lambda n.\ f\ (d\ n)\ x)$
proof (*rule tendsto-ge-const*)
show $(F \longrightarrow ereal\ (F\ x))$ (*at-left x*)
using *cts* **unfolding** *continuous-at-split* **by** (*auto simp*: *continuous-within*)

show $\forall_F\ i$ *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 ‹$y < r\ N$› **by** (*auto simp*: *F-eq convergent-real-imp-convergent-ereal rat-cnv convergent-liminf-cl intro*!: *INF-lower2*)
also have $\ldots \leq$ *liminf* $(\lambda n.\ f\ (d\ n)\ x)$
using ‹$r\ N < x$› **by** (*auto intro*!: *Liminf-mono monoD*[*OF mono*] *always-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* ⇒ *real measure*) ⇒ *bool*
where
tight $\mu \equiv (\forall n.$ *real-distribution* $(\mu\ n)) \wedge (\forall (\varepsilon{::}real){>}0.\ \exists a\ b{::}real.\ a < b \wedge (\forall n.$ *measure* $(\mu\ n)\ \{a{<}..b\} > 1 - \varepsilon))$

theorem *tight-imp-convergent-subsubsequence*:
assumes μ: *tight* μ *subseq s*
shows $\exists r\ M.$ *subseq* $r \wedge$ *real-distribution* $M \wedge$ *weak-conv-m* $(\mu \circ s \circ r)\ M$
proof −
def $f \equiv \lambda k.$ *cdf* $(\mu\ (s\ k))$
interpret μ: *real-distribution* $\mu\ k$ **for** k
using μ **unfolding** *tight-def* **by** *auto*

have *rcont*: $\bigwedge x.$ *continuous* (*at-right x*) (*f k*)
and *mono*: *mono* (*f k*)
and *top*: (*f k* ⟶ *1*) *at-top*
and *bot*: (*f k* ⟶ *0*) *at-bot* **for** k
unfolding *f-def mono-def*
using μ*.cdf-nondecreasing* μ*.cdf-is-right-cont* μ*.cdf-lim-at-top-prob* μ*.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* μ*.cdf-nonneg* μ*.cdf-bounded-prob*)

from *Helly-selection*[*OF rcont mono bdd*, *of* $\lambda x.\ x$] **obtain** $r\ F$
where *F*: *subseq r* $\bigwedge x.$ *continuous* (*at-right x*) *F mono F* $\bigwedge x.\ |F\ x| \leq 1$
and *lim-F*: $\bigwedge x.$ *continuous* (*at x*) $F \Longrightarrow (\lambda n.\ f\ (r\ n)\ x) \longrightarrow F\ x$
by *blast*

```
  have 0 ≤ f n x for n x
    unfolding f-def by (rule μ.cdf-nonneg)
  have F-nonneg: 0 ≤ F x for x
  proof −
    obtain y where y < x isCont F y
      using open-minus-countable[OF mono-ctble-discont[OF ‹mono F›], of {..<
x}] by auto
    then have 0 ≤ F y
      by (intro LIMSEQ-le-const[OF lim-F]) (auto simp: f-def μ.cdf-nonneg)
    also have ... ≤ F x
      using ‹y < x› by (auto intro!: monoD[OF ‹mono F›])
    finally show 0 ≤ F x .
  qed

  have Fab: ∃ a b. (∀ x≥b. F x ≥ 1 − ε) ∧ (∀ x≤a. F x ≤ ε) if ε: 0 < ε for ε
  proof auto
    obtain a' b' where a'b': a' < b' ⋀k. measure (μ k) {a'<..b'} > 1 − ε
      using ε μ by (auto simp: tight-def)
    obtain a where a: a < a' isCont F a
      using open-minus-countable[OF mono-ctble-discont[OF ‹mono F›], of {..<
a'}] by auto
    obtain b where b: b' < b isCont F b
      using open-minus-countable[OF mono-ctble-discont[OF ‹mono F›], of {b'
<..}] by auto
    have a < b
      using a b a'b' by simp

    let ?μ = λk. measure (μ (s (r k)))
    have ab: ?μ k {a<..b} > 1 − ε for k
    proof −
      have ?μ k {a'<..b'} ≤ ?μ k {a<..b}
        using a b by (intro μ.finite-measure-mono) auto
      then show ?thesis
        using a'b'(2) by (metis less-eq-real-def less-trans)
    qed

    have (λk. ?μ k {..b}) ⟶ F b
      using b(2) lim-F unfolding f-def cdf-def o-def by auto
    then have 1 − ε ≤ F b
    proof (rule tendsto-le-const[OF sequentially-bot], intro always-eventually allI)
      fix k
      have 1 − ε < ?μ k {a<..b}
        using ab by auto
      also have ... ≤ ?μ k {..b}
        by (auto intro!: μ.finite-measure-mono)
      finally show 1 − ε ≤ ?μ k {..b}
        by (rule less-imp-le)
    qed
```

```
    then show ∃ b. ∀ x≥b. 1 − ε ≤ F x
      using F unfolding mono-def by (metis order.trans)

    have (λk. ?μ k {..a}) ⟶ F a
      using a(2) lim-F unfolding f-def cdf-def o-def by auto
    then have Fa: F a ≤ ε
    proof (rule tendsto-ge-const[OF sequentially-bot], intro always-eventually allI)
      fix k
      have ?μ k {..a} + ?μ k {a<..b} ≤ 1
        by (subst μ.finite-measure-Union[symmetric]) auto
      then show ?μ k {..a} ≤ ε
        using ab[of k] by simp
    qed
    then show ∃ a. ∀ x≤a. F x ≤ ε
      using F unfolding mono-def by (metis order.trans)
  qed

  have (F ⟶ 1) at-top
  proof (rule order-tendstoI)
    show 1 < y ⟹ ∀⇩F x in at-top. F x < y for y
    using ‹⋀x. |F x| ≤ 1› ‹⋀x. 0 ≤ F x› by (auto intro: le-less-trans always-eventually)
    fix y :: real assume y < 1
    then obtain z where y < z z < 1
      using dense[of y 1] by auto
    with Fab[of 1 − z] show ∀⇩F x in at-top. y < F x
      by (auto simp: eventually-at-top-linorder intro: less-le-trans)
  qed
  moreover
  have (F ⟶ 0) at-bot
  proof (rule order-tendstoI)
    show y < 0 ⟹ ∀⇩F x in at-bot. y < F x for y
      using ‹⋀x. 0 ≤ F x› by (auto intro: less-le-trans always-eventually)
    fix y :: real assume 0 < y
    then obtain z where 0 < z z < y
      using dense[of 0 y] by auto
    with Fab[of z] show ∀⇩F x in 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 (μ ∘ s ∘ r) (interval-measure
F)
    by (auto simp: weak-conv-def weak-conv-m-def f-def comp-def)
  then show ∃ r M. subseq r ∧ (real-distribution M ∧ weak-conv-m (μ ∘ s ∘ r)
M)
    using F M by auto
qed
```

corollary *tight-subseq-weak-converge*:
fixes μ :: *nat* $\Rightarrow$ *real measure* **and** M :: *real measure*
assumes $\bigwedge n$. *real-distribution* (μ *n*) *real-distribution M* **and** *tight*: *tight* μ **and**
subseq: $\bigwedge s\ \nu$. *subseq s* $\Longrightarrow$ *real-distribution* ν $\Longrightarrow$ *weak-conv-m* ($\mu \circ s$) ν $\Longrightarrow$ *weak-conv-m* ($\mu \circ s$) *M*
shows *weak-conv-m* μ *M*
proof (*rule ccontr*)
def $f \equiv \lambda n$. *cdf* (μ *n*) **and** $F \equiv$ *cdf M*

assume $\neg$ *weak-conv-m* μ *M*
then obtain *x* **where** *x*: *isCont F x* $\neg$ (λn. *f n x*) $\longrightarrow$ *F x*
by (*auto simp*: *weak-conv-m-def weak-conv-def f-def F-def*)
then obtain ε **where** $\varepsilon > 0$ **and** *infinite* $\{n. \neg$ *dist* (*f n x*) (*F x*) $< \varepsilon\}$
by (*auto simp*: *tendsto-iff not-eventually INFM-iff-infinite cofinite-eq-sequentially*[*symmetric*])
then obtain *s* **where** *s*: $\bigwedge n. \neg$ *dist* (*f* (*s n*) *x*) (*F x*) $< \varepsilon$ **and** *subseq s*
using *enumerate-in-set enumerate-mono* **by** (*fastforce simp*: *subseq-def*)
then obtain *r* ν **where** *r*: *subseq r real-distribution* ν *weak-conv-m* ($\mu \circ s \circ r$) ν
using *tight-imp-convergent-subsubsequence*[*OF tight*] **by** *blast*
then have *weak-conv-m* ($\mu \circ (s \circ r)$) *M*
using ‹*subseq s*› *r* **by** (*intro subseq subseq-o*) (*auto simp*: *comp-assoc*)
then have (λn. *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* ‹$\varepsilon > 0$› **by** (*auto dest*: *tendstoD*)
qed

end

44 Integral of sinc

theory *Sinc-Integral*
imports *Distributions*
begin

44.1 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 (λ*x*. *x*^*k* ∗ *exp* (−*x*) ∗ *indicator* {*0* ..} *x*::*real*) (*fact k*)
using *nn-intergal-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 (λ*x*. *x*^*k* ∗ *exp* (−*x*) ∗ *indicator* {*0* <..} *x*::*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*::*real*) ⟹ *set-integrable lborel* {*0* <..} (λ*x*. *exp* (−(*x* ∗ *u*)))
using *lborel-integrable-real-affine-iff*[*of u* λ*x*. *indicator* {*0* <..} *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*)

lemma *LBINT-I0i-exp-mscale*: *0* < (*u*::*real*) ⟹ *LBINT x*=*0*..∞. *exp* (−(*x* ∗ *u*)) = *1* / *u*
using *lborel-integral-real-affine*[*of u* λ*x*. *indicator* {*0*<..} *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-infty 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 has-vector-derivative exp* (− (*u* ∗ *x*)) ∗ *sin x*) (*at x within* {*min 0 t*..*max 0 t*}) **for** *x*
by (*auto intro*!: *derivative-eq-intros*
simp: *has-field-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*..∞. |*exp* (−*u* ∗ *x*) ∗ *sin x*| = |*sin x*| / *x*
proof (*subst interval-integral-FTC-nonneg*)
let *?F* = λ*u*. *1* / *x* ∗ (*1* − *exp* (− *u* ∗ *x*)) ∗ |*sin x*|
show ⋀*t*. (*?F has-real-derivative* |*exp* (− *t* ∗ *x*) ∗ *sin x*|) (*at t*)
using ‹*0* < *x*› **by** (*auto intro*!: *derivative-eq-intros simp*: *abs-mult*)
show ((*?F* ∘ *real-of-ereal*) ⟶ *0*) (*at-right 0*)
using ‹*0* < *x*› **by** (*auto simp*: *zero-ereal-def ereal-tendsto-simps intro*!: *tendsto-eq-intros*)

have $*$: $((\lambda t.\ exp\ (-\ t * x)) \longrightarrow 0)$ *at-top*
using ‹$0 < x$›
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 $((?F \circ real\text{-}of\text{-}ereal) \longrightarrow |sin\ x|\ /\ x)\ (at\text{-}left\ \infty)$
using ‹$0 < x$› **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* $(-\infty)\ \infty$) $(\lambda x.\ inverse\ (1\ +\ x\hat{}2))$
and *LBINT-inverse-1-plus-square*:
LBINT $x{=}-\infty..\infty.\ inverse\ (1\ +\ x\hat{}2) = pi$
proof −
have *1*: $-\ (pi\ /\ 2) < x \Longrightarrow x * 2 < pi \Longrightarrow (tan\ has\text{-}real\text{-}derivative\ 1\ +\ (tan\ x)^2)\ (at\ x)$ **for** *x*
using *cos-gt-zero-pi*[*of x*] **by** (*subst tan-sec*) (*auto intro*!: *DERIV-tan simp*: *power-inverse*)
have *2*: $-\ (pi\ /\ 2) < x \Longrightarrow x * 2 < pi \Longrightarrow isCont\ (\lambda x.\ 1\ +\ (tan\ x)^2)\ x$ **for** *x*
using *cos-gt-zero-pi*[*of x*] **by** *auto*
show *LBINT* $x{=}-\infty..\infty.\ inverse\ (1\ +\ x\hat{}2) = pi$
by (*subst interval-integral-substitution-nonneg*[*of* $-pi/2\ pi/2\ tan\ \lambda x.\ 1\ +\ (tan\ x)\ \hat{}\ 2$])
(*auto intro*: *derivative-eq-intros 1 2 filterlim-tan-at-right*
simp add: *ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff*)
show *set-integrable lborel* (*einterval* $(-\infty)\ \infty$) $(\lambda x.\ inverse\ (1\ +\ x\hat{}2))$
by (*subst interval-integral-substitution-nonneg*[*of* $-pi/2\ pi/2\ tan\ \lambda x.\ 1\ +\ (tan\ x)\ \hat{}\ 2$])
(*auto intro*: *derivative-eq-intros 1 2 filterlim-tan-at-right*
simp add: *ereal-tendsto-simps filterlim-tan-at-left add-nonneg-eq-0-iff*)
qed

lemma
shows *integrable-I0i-1-div-plus-square*:
interval-lebesgue-integrable lborel 0 ∞ $(\lambda x.\ 1\ /\ (1\ +\ x\hat{}2))$
and *LBINT-I0i-1-div-plus-square*:
LBINT $x{=}0..\infty.\ 1\ /\ (1\ +\ x\hat{}2) = pi\ /\ 2$
proof −
have *1*: $0 < x \Longrightarrow x * 2 < pi \Longrightarrow (tan\ has\text{-}real\text{-}derivative\ 1\ +\ (tan\ x)^2)\ (at\ x)$ **for** *x*
using *cos-gt-zero-pi*[*of x*] **by** (*subst tan-sec*) (*auto intro*!: *DERIV-tan simp*: *power-inverse*)
have *2*: $0 < x \Longrightarrow x * 2 < pi \Longrightarrow isCont\ (\lambda x.\ 1\ +\ (tan\ x)^2)\ x$ **for** *x*
using *cos-gt-zero-pi*[*of x*] **by** *auto*
show *LBINT* $x{=}0..\infty.\ 1\ /\ (1\ +\ x\hat{}2) = pi\ /\ 2$
by (*subst interval-integral-substitution-nonneg*[*of 0 pi/2 tan* $\lambda x.\ 1\ +\ (tan\ x)\ \hat{}\ 2$])
(*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*)
show *interval-lebesgue-integrable lborel 0* ∞ ($\lambda x.$ *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*)
qed

45 The sinc function, and the sine integral (Si)

abbreviation *sinc* :: *real* $\Rightarrow$ *real* **where**
sinc $\equiv$ ($\lambda x.$ *if x = 0 then 1 else sin x / x*)

lemma *sinc-at-0*: (($\lambda x.$ *sin x / x*::*real*) $\longrightarrow$ *1*) (*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=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* = *abs x* **and** *f* = $\lambda x.$ *sin x / x*])
(*auto simp add*: *dist-real-def* ‹*x* $\neq$ *0*›)
qed

lemma *continuous-on-sinc*[*continuous-intros*]:
continuous-on S f $\Longrightarrow$ *continuous-on S* ($\lambda x.$ *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-on1 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 ¬ *0* ≤ *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* ≥ *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*) (λ*t*. *sin* (*t* ∗ *ϑ*) / *t*)
proof −
have ∗: *interval-lebesgue-integrable lborel* (*ereal 0*) (*ereal T*) (λ*t*. *ϑ* ∗ *sinc* (*t* ∗ *ϑ*))
by (*intro interval-lebesgue-integrable-mult-right interval-integrable-isCont continuous-within-compose3* [*OF isCont-sinc*])
auto
show *?thesis*
by (*rule interval-lebesgue-integrable-cong-AE*[*THEN iffD1*, *OF - - -* ∗])
(*insert AE-lborel-singleton*[*of 0*], *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*) ≤ *x x* ≤ *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-field-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* ∈ *borel-measurable borel*
by (*auto intro*: *isCont-Si continuous-at-imp-continuous-on borel-measurable-continuous-on1*)

lemma *Si-at-top-LBINT*:
((λ*t*. (*LBINT x*=*0*..∞. *exp* (−(*x* ∗ *t*)) ∗ (*x* ∗ *sin t* + *cos t*) / (*1* + *x*^*2*))) ⟶ *0*) *at-top*
proof −
let *?F* = λ*x t*. *exp* (− (*x* ∗ *t*)) ∗ (*x* ∗ *sin t* + *cos t*) / (*1* + *x*^{2}) :: *real*
have *int*: *set-integrable lborel* {*0*<..} (λ*x*. *exp* (− *x*) ∗ (*x* + *1*) :: *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*)

have ((λ*t*::*real*. *LBINT x*:{*0*<..}. *?F x t*) ⟶ *LBINT x*::*real*:{*0*<..}. *0*) *at-top*
proof (*rule integral-dominated-convergence-at-top*[*OF - - int*], *simp-all del*: *abs-divide split*: *split-indicator*)
have ∗: *0* < *x* ⟹ |*x* ∗ *sin t* + *cos t*| / (*1* + *x*^{2}) ≤ (*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* ≤ *t* ⟹ *0* < *x* ⟹ |*?F x t*| ≤ *exp* (− *x*) ∗ (*x* + *1*) **for** *x t* :: *real*
by (*auto simp add*: *abs-mult times-divide-eq-right*[*symmetric*] *simp del*: *times-divide-eq-right*
intro!: *mult-mono*)

show ∀$_F$ *i in at-top*. *AE x in lborel*. *0* < *x* ⟶ |*?F x i*| ≤ *exp* (− *x*) ∗ (*x* + *1*)
using *eventually-ge-at-top*[*of 1*::*real*] ∗∗ **by** (*auto elim*: *eventually-mono*)
show *AE x in lborel*. *0* < *x* ⟶ (*?F x* ⟶ *0*) *at-top*
proof (*intro always-eventually impI allI*)
fix *x* :: *real* **assume** *0* < *x*
show ((λ*t*. *exp* (− (*x* ∗ *t*)) ∗ (*x* ∗ *sin t* + *cos t*) / (*1* + *x*^{2})) ⟶ *0*) *at-top*
proof (*rule Lim-null-comparison*)
show ∀$_F$ *t in at-top*. *norm* (*?F x t*) ≤ *exp* (− (*x* ∗ *t*)) ∗ (*x* + *1*)
using *eventually-ge-at-top*[*of 1*::*real*] ∗ ‹*0* < *x*›
by (*auto simp add*: *abs-mult times-divide-eq-right*[*symmetric*] *simp del*: *times-divide-eq-right*
intro!: *mult-mono elim*: *eventually-mono*)
show ((λ*t*. *exp* (− (*x* ∗ *t*)) ∗ (*x* + *1*)) ⟶ *0*) *at-top*

```
      by (auto simp: filterlim-uminus-at-top [symmetric]
                intro!: filterlim-tendsto-pos-mult-at-top[OF tendsto-const] ‹0<x›
filterlim-ident
                    tendsto-mult-left-zero filterlim-compose[OF exp-at-bot])
    qed
  qed
 qed
 then show ((λt. (LBINT x=0..∞. exp (−(x * t)) * (x * sin t + cos t) / (1 +
x^2))) ⟶ 0) at-top
  by (simp add: interval-lebesgue-integral-0-infty)
qed

lemma Si-at-top-integrable:
 assumes t ≥ 0
 shows interval-lebesgue-integrable lborel 0 ∞ (λx. exp (− (x * t)) * (x * sin t +
cos t) / (1 + x²))
 using ‹0 ≤ t› 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 ≤ a ⟹ 0 < x ⟹ a / (1 + x²) ≤ a for a x :: real
  using zero-le-power2[of x, arith] using divide-left-mono[of 1 1+x² a] by auto
 have set-integrable lborel {0<..} (λx. (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)
 from lborel-integrable-real-affine[OF this, of t 0]
 show ?thesis
  unfolding interval-lebesgue-integral-0-infty
  by (rule integrable-bound) (auto simp: field-simps * split: split-indicator)
qed

lemma Si-at-top: (Si ⟶ pi / 2) at-top
proof −
 have ∀F t in at-top. pi / 2 − (LBINT u=0..∞. 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..∞. 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..∞. indicator {0 <..< t} x *R
sin x * exp (−(u * x)))
```

using ‹$0 \leq t$› **by** (*simp add*: *zero-ereal-def interval-lebesgue-integral-le-eq mult-ac*)
also have ... = *LBINT x*. (*LBINT u*. *indicator* ({0<..} × {0 <..< t}) (u, x) $*_R$ (*sin x* ∗ *exp* (−(u ∗ x))))
by (*subst interval-integral-Ioi*) (*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: $x \neq 0$ $x \neq t$
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 integral-cong*) (*auto split*: *split-indicator simp*: *abs-mult*)
also have ... = |*sin x*| ∗ *indicator* {0<..<t} x ∗ (*LBINT y*=0..∞. *exp* (− (*y* ∗ x)))
by (*cases x* > 0)
(*auto simp*: *field-simps interval-lebesgue-integral-0-infty 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 *set-integrable lborel* {0 .. t} (λx. *abs* (*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 $0 < x \Longrightarrow$ *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 split*: *split-indicator*)
qed
also have ... = *LBINT u*=0..∞. (*LBINT x*=0..t. *exp* (−(u ∗ x)) ∗ *sin x*)
using ‹$0 \leq t$›
by (*auto simp*: *interval-lebesgue-integral-def zero-ereal-def ac-simps*

split: *split-indicator intro*!: *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 + u^2))*
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 + u^2)) = Si t* **..**
qed
then show *?thesis*
by (*rule Lim-transform-eventually*)
(*auto intro*!: *tendsto-eq-intros Si-at-top-LBINT*)
qed

45.1 The final theorems: boundedness and scalability

lemma *bounded-Si*: ∃ *B*. ∀ *T*. |*Si T*| ≤ *B*
proof −
have ∗: *0 ≤ y* ⟹ *dist x y < z* ⟹ *abs x ≤ y + z* **for** *x y z* :: *real*
by (*simp add*: *dist-real-def*)

have *eventually* (λ*T*. *dist* (*Si T*) (*pi / 2*) < *1*) *at-top*
using *Si-at-top* **by** (*elim tendstoD*) *simp*
then have *eventually* (λ*T*. *0 ≤ T* ∧ |*Si T*| ≤ *pi / 2 + 1*) *at-top*
using *eventually-ge-at-top*[*of 0*] **by** *eventually-elim* (*insert* ∗[*of pi/2 Si - 1*], *auto*)
then have ∃ *T*. *0 ≤ T* ∧ (∀ *t* ≥ *T*. |*Si t*| ≤ *pi / 2 + 1*)
by (*auto simp add*: *eventually-at-top-linorder*)
then obtain *T* **where** *T*: *0 ≤ T* ⋀*t*. *t ≥ T* ⟹ |*Si t*| ≤ *pi / 2 + 1*
by *auto*
moreover have *t* ≤ − *T* ⟹ |*Si t*| ≤ *pi / 2 + 1* **for** *t*
using *T*(*1*) *T*(*2*)[*of* −*t*] *Si-neg*[*of* − *t*] **by** *simp*
moreover have ∃ *M*. ∀ *t*. −*T ≤ t* ∧ *t ≤ T* ⟶ |*Si t*| ≤ *M*
by (*rule isCont-bounded*) (*auto intro*!: *isCont-Si continuous-intros* ‹*0 ≤ T*›)
then obtain *M* **where** *M*: ⋀*t*. −*T ≤ t* ∧ *t ≤ T* ⟹ |*Si t*| ≤ *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 ≥ 0*
shows *LBINT t=0..T. sin (t ∗ ϑ) / t = sgn ϑ ∗ Si (T ∗ |ϑ|)*
proof −
{ assume *0 < ϑ*
have (*LBINT t=ereal 0..T. sin (t ∗ ϑ) / t*) = (*LBINT t=ereal 0..T. ϑ ∗R sinc*

```
(t * ϑ))
      by (rule interval-integral-discrete-difference[of {0}]) auto
    also have ... = (LBINT t=ereal (0 * ϑ)..T * ϑ. 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 * ϑ)..T * ϑ. sin t / t)
      by (rule interval-integral-discrete-difference[of {0}]) auto
    finally have (LBINT t=ereal 0..T. sin (t * ϑ) / t) = (LBINT t=ereal 0..T *
ϑ. sin t / t)
      by simp
    hence LBINT x. indicator {0<..<T} x * sin (x * ϑ) / x =
        LBINT x. indicator {0<..<T * ϑ} x * sin x / x
        using assms ‹0 < ϑ› unfolding interval-lebesgue-integral-def einterval-eq
zero-ereal-def
        by (auto simp: ac-simps)
  } note aux1 = this
  { assume 0 > ϑ
    have [simp]: integrable lborel (λx. sin (x * ϑ) * indicator {0<..<T} x / x)
      using integrable-sinc' [of T ϑ] assms
      by (simp add: interval-lebesgue-integrable-def ac-simps)
    have (LBINT t=ereal 0..T. sin (t * −ϑ) / t) = (LBINT t=ereal 0..T. −ϑ *R
sinc (t * −ϑ))
      by (rule interval-integral-discrete-difference[of {0}]) auto
    also have ... = (LBINT t=ereal (0 * −ϑ)..T * −ϑ. 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 * −ϑ)..T * −ϑ. sin t / t)
      by (rule interval-integral-discrete-difference[of {0}]) auto
    finally have (LBINT t=ereal 0..T. sin (t * −ϑ) / t) = (LBINT t=ereal 0..T
* −ϑ. sin t / t)
      by simp
    hence LBINT x. indicator {0<..<T} x * sin (x * ϑ) / x =
      − (LBINT x. indicator {0<..<− (T * ϑ)} x * sin x / x)
        using assms ‹0 > ϑ› unfolding interval-lebesgue-integral-def einterval-eq
zero-ereal-def
        by (auto simp add: field-simps mult-le-0-iff split: if-split-asm)
  } note aux2 = this
  show ?thesis
   using assms unfolding Si-def interval-lebesgue-integral-def sgn-real-def einterval-eq
zero-ereal-def
    apply auto
    apply (erule aux1)
    apply (rule aux2)
    apply auto
    done
qed
```

end

46 The Levy inversion theorem, and the Levy continuity theorem.

theory *Levy*
imports *Characteristic-Functions Helly-Selection Sinc-Integral*
begin

lemma *LIM-zero-cancel*:
fixes $f :: - \Rightarrow 'b$::*real-normed-vector*
shows $((\lambda x.\ f\ x - l) \longrightarrow 0)\ F \Longrightarrow (f \longrightarrow l)\ F$
unfolding *tendsto-iff dist-norm* **by** *simp*

46.1 The Levy inversion theorem

lemma *Levy-Inversion-aux1*:
fixes $a\ b :: real$
assumes $a \leq b$
shows $((\lambda t.\ (iexp\ (-(t * a)) - iexp\ (-(t * b)))\ /\ (ii * t)) \longrightarrow b - a)\ (at\ 0)$
(**is** $(?F \longrightarrow -)\ (at\ -)$)
proof −
have *1*: $cmod\ (?F\ t - (b - a)) \leq a\hat{}2\ /\ 2 * abs\ t + b\hat{}2\ /\ 2 * abs\ t$ **if** $t \neq 0$
for t
proof −
have $cmod\ (?F\ t - (b - a)) = cmod\ (((iexp\ (-(t * a)) - (1 + ii * -(t * a)))\ /\ (ii * t) -$
$(iexp\ (-(t * b)) - (1 + ii * -(t * b)))\ /\ (ii * t))$
(**is** $- = cmod\ (?one\ /\ (ii * t) - ?two\ /\ (ii * t))$)
using ‹$t \neq 0$› **by** (*intro arg-cong*[**where** f=*norm*]) (*simp add*: *field-simps*)
also have $\ldots \leq cmod\ (?one\ /\ (ii * t)) + cmod\ (?two\ /\ (ii * t))$
by (*rule norm-triangle-ineq4*)
also have $cmod\ (?one\ /\ (ii * t)) = cmod\ ?one\ /\ abs\ t$
by (*simp add*: *norm-divide norm-mult*)
also have $cmod\ (?two\ /\ (ii * t)) = cmod\ ?two\ /\ abs\ t$
by (*simp add*: *norm-divide norm-mult*)
also have $cmod\ ?one\ /\ abs\ t + cmod\ ?two\ /\ abs\ t \leq$
$((-(a * t))\hat{}2\ /\ 2)\ /\ abs\ t + ((-(b * t))\hat{}2\ /\ 2)\ /\ abs\ t$
apply (*rule add-mono*)
apply (*rule divide-right-mono*)
using *iexp-approx1* [*of* $-(t * a)$ *1*] **apply** (*simp add*: *field-simps eval-nat-numeral*)
apply *force*
apply (*rule divide-right-mono*)
using *iexp-approx1* [*of* $-(t * b)$ *1*] **apply** (*simp add*: *field-simps eval-nat-numeral*)
by *force*
also have $\ldots = a\hat{}2\ /\ 2 * abs\ t + b\hat{}2\ /\ 2 * abs\ t$
using ‹$t \neq 0$› **apply** (*case-tac* $t \geq 0$, *simp add*: *field-simps power2-eq-square*)
using ‹$t \neq 0$› **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)) / (ii ∗ t)) ≤ b − a (is ?F ≤ -)
proof −
  have ?F = cmod (iexp (t ∗ a) ∗ (iexp (t ∗ (b − a)) − 1) / (ii ∗ 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-ii-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 μ ≡ measure M and φ ≡ char M
  assumes μ {a} = 0 and μ {b} = 0
  shows (λT. 1 / (2 ∗ pi) ∗ (CLBINT t=−T..T. (iexp (−(t ∗ a)) − iexp (−(t ∗
b))) / (ii ∗ t) ∗ φ t))
    ⟶ μ {a<..b}
    (is (λT. 1 / (2 ∗ pi) ∗ (CLBINT t=−T..T. ?F t ∗ φ t)) ⟶ of-real (μ
{a<..b}))
proof −
  interpret P: pair-sigma-finite lborel M ..
  from bounded-Si obtain B where Bprop: ⋀T. abs (Si T) ≤ B by auto
  from Bprop [of 0] have [simp]: B ≥ 0 by auto
  let ?f = λt x :: real. (iexp (t ∗ (x − a)) − iexp(t ∗ (x − b))) / (ii ∗ t)
  { fix T :: real
    assume T ≥ 0
    let ?f' = λ(t, x). indicator {−T<..<T} t ∗R ?f t x
    { fix x
```

have *1*: *complex-interval-lebesgue-integrable lborel u v* (λ*t*. *?f t x*) **for** *u v* :: *real*
using *Levy-Inversion-aux2*[*of x − b x − a*]
apply (*simp add*: *interval-lebesgue-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 (*CLBINT t*. *?f'* (*t*, *x*)) = (*CLBINT t*=*− T*..*T*. *?f t x*)
using ‹*T ≥ 0*› **by** (*simp add*: *interval-lebesgue-integral-def*)
also have ... = (*CLBINT t*=*− T*..(*0* :: *real*). *?f t x*) + (*CLBINT t*=(*0* :: *real*)..*T*. *?f t x*)
(**is** - = - + *?t*)
using *1* **by** (*intro interval-integral-sum*[*symmetric*]) (*simp add*: *min-absorb1 max-absorb2* ‹*T ≥ 0*›)
also have (*CLBINT t*=*− T*..(*0* :: *real*). *?f t x*) = (*CLBINT t*=(*0*::*real*)..*T*. *?f* (*−t*) *x*)
by (*subst interval-integral-reflect*) *auto*
also have ... + *?t* = (*CLBINT t*=(*0*::*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 ... = (*CLBINT t*=(*0*::*real*)..*T*. ((*iexp*(*t* ∗ (*x − a*)) − *iexp* (*−*(*t* ∗ (*x − a*)))) −
(*iexp*(*t* ∗ (*x − b*)) − *iexp* (*−*(*t* ∗ (*x − b*))))) / (*ii* ∗ *t*))
using ‹*T ≥ 0*› **by** (*intro interval-integral-cong*) (*auto simp add*: *divide-simps*)
also have ... = (*CLBINT t*=(*0*::*real*)..*T*. *complex-of-real*(
2 ∗ (*sin* (*t* ∗ (*x − a*)) / *t*) − *2* ∗ (*sin* (*t* ∗ (*x − b*)) / *t*)))
using ‹*T ≥ 0*›
apply (*intro interval-integral-cong*)
apply (*simp add*: *field-simps cis.ctr Im-divide Re-divide Im-exp Re-exp complex-eq-iff*)
unfolding *minus-diff-eq*[*symmetric*, *of y ∗ x y ∗ a* **for** *y a*] *sin-minus cos-minus*
apply (*simp add*: *field-simps power2-eq-square*)
done
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* (*2* ∗ (*sgn* (*x − a*) ∗ *Si* (*T* ∗ *abs* (*x − a*)) −
sgn (*x − b*) ∗ *Si* (*T* ∗ *abs* (*x − b*))))
apply (*subst interval-lebesgue-integral-diff*)
apply (*rule interval-lebesgue-integrable-mult-right*, *rule integrable-sinc'*)+
apply (*subst interval-lebesgue-integral-mult-right*)+
apply (*simp add*: *zero-ereal-def*[*symmetric*] *LBINT-I0c-sin-scale-divide*[*OF* ‹*T ≥ 0*›])
done
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<..<T} t))
    using ‹T ≥ 0› unfolding φ-def char-def interval-lebesgue-integral-def
    by (auto split: split-indicator intro!: integral-cong)
  also have ... = (CLBINT t. (CLINT x | M. ?f' (t, x)))
      by (auto intro!: integral-cong simp: field-simps exp-diff exp-minus split:
split-indicator)
  also have ... = (CLINT x | M. (CLBINT t. ?f' (t, x)))
 proof (intro P.Fubini-integral [symmetric] integrableI-bounded-set [where B=b
− a])
    show emeasure (lborel ⨂M M) ({− T<..<T} × space M) < ∞
      using ‹T ≥ 0›
      by (subst emeasure-pair-measure-Times)
        (auto simp: ennreal-mult-less-top not-less top-unique)
    show AE x∈{− T<..<T} × space M in lborel ⨂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 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 :: real measure
 assumes real-distribution M1 real-distribution M2 and
  char M1 = 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. measure M1 {x} ≠ 0} ∪ {x. measure M2 {x} ≠ 0})
  by (intro countable-Un M2.countable-support M1.countable-support)
 then have count: countable {x. measure M1 {x} ≠ 0 ∨ measure M2 {x} ≠ 0}
  by (simp add: Un-def)

 have cdf M1 = cdf M2
 proof (rule ext)
  fix x
  from M1.cdf-is-right-cont [of x] have (cdf M1 ⟶ cdf M1 x) (at-right x)
   by (simp add: continuous-within)
```

from *M2.cdf-is-right-cont* [*of x*] **have** (*cdf M2* ⟶ *cdf M2 x*) (*at-right x*)
by (*simp add*: *continuous-within*)

{ fix ε :: *real*
assume ε > *0*
from ‹ε > *0*› ‹(*cdf M1* ⟶ *0*) *at-bot*› ‹(*cdf M2* ⟶ *0*) *at-bot*›
have *eventually* (λ*y*. |*cdf M1 y*| < ε / *4* ∧ |*cdf M2 y*| < ε / *4* ∧ *y* ≤ *x*) *at-bot*
by (*simp only*: *tendsto-iff dist-real-def diff-0-right eventually-conj eventually-le-at-bot*)
then obtain *M* **where** ⋀*y*. *y* ≤ *M* ⟹ |*cdf M1 y*| < ε / *4* ⋀*y*. *y* ≤ *M* ⟹ |*cdf M2 y*| < ε / *4* *M* ≤ *x*
unfolding *eventually-at-bot-linorder* **by** *auto*
with *open-minus-countable*[*OF count*, *of* {..< *M*}] **obtain** *a* **where**
measure M1 {*a*} = *0 measure M2* {*a*} = *0 a* < *M a* ≤ *x* |*cdf M1 a*| < ε / *4* |*cdf M2 a*| < ε / *4*
by *auto*

from ‹ε > *0*› ‹(*cdf M1* ⟶ *cdf M1 x*) (*at-right x*)› ‹(*cdf M2* ⟶ *cdf M2 x*) (*at-right x*)›
have *eventually* (λ*y*. |*cdf M1 y* − *cdf M1 x*| < ε / *4* ∧ |*cdf M2 y* − *cdf M2 x*| < ε / *4* ∧ *x* < *y*) (*at-right x*)
by (*simp only*: *tendsto-iff dist-real-def eventually-conj eventually-at-right-less*)
then obtain *N* **where** *N* > *x* ⋀*y*. *x* < *y* ⟹ *y* < *N* ⟹ |*cdf M1 y* − *cdf M1 x*| < ε / *4*
⋀*y*. *x* < *y* ⟹ *y* < *N* ⟹ |*cdf M2 y* − *cdf M2 x*| < ε / *4* ⋀*y*. *x* < *y* ⟹ *y* < *N* ⟹ *x* < *y*
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 b* < *N*
measure M1 {*b*} = *0 measure M2* {*b*} = *0* |*cdf M2 x* − *cdf M2 b*| < ε / *4* |*cdf M1 x* − *cdf M1 b*| < ε / *4*
by (*auto simp*: *abs-minus-commute*)
from ‹*a* ≤ *x*› ‹*x* < *b*› **have** *a* < *b a* ≤ *b* **by** *auto*

from ‹*char M1* = *char M2*›
M1.Levy-Inversion [*OF* ‹*a* ≤ *b*› ‹*measure M1* {*a*} = *0*› ‹*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 *measure M1* {*a*<..*b*} = *measure M2* {*a*<..*b*} **by** *auto*
then have ∗: *cdf M1 b* − *cdf M1 a* = *cdf M2 b* − *cdf M2 a*
unfolding *M1.cdf-diff-eq* [*OF* ‹*a* < *b*›] *M2.cdf-diff-eq* [*OF* ‹*a* < *b*›] **.**

have *abs* (*cdf M1 x* − (*cdf M1 b* − *cdf M1 a*)) = *abs* (*cdf M1 x* − *cdf M1 b* + *cdf M1 a*)
by *simp*
also have ... ≤ *abs* (*cdf M1 x* − *cdf M1 b*) + *abs* (*cdf M1 a*)

by (*rule abs-triangle-ineq*)
also have ... $\leq \varepsilon$ / 4 + ε / 4
by (*intro add-mono less-imp-le* ‹|*cdf M1 a*| < ε / 4› ‹|*cdf M1 x* − *cdf M1 b*| < ε / 4›)
finally have *1*: *abs* (*cdf M1 x* − (*cdf M1 b* − *cdf M1 a*)) ≤ ε / 2 **by** *simp*

have *abs* (*cdf M2 x* − (*cdf M2 b* − *cdf M2 a*)) = *abs* (*cdf M2 x* − *cdf M2 b* + *cdf M2 a*)
by *simp*
also have ... ≤ *abs* (*cdf M2 x* − *cdf M2 b*) + *abs* (*cdf M2 a*)
by (*rule abs-triangle-ineq*)
also have ... ≤ ε / 4 + ε / 4
by (*intro add-mono less-imp-le* ‹|*cdf M2 x* − *cdf M2 b*| < ε / 4› ‹|*cdf M2 a*| < ε / 4›)
finally have *2*: *abs* (*cdf M2 x* − (*cdf M2 b* − *cdf M2 a*)) ≤ ε / 2 **by** *simp*

have *abs* (*cdf M1 x* − *cdf M2 x*) = *abs* ((*cdf M1 x* − (*cdf M1 b* − *cdf M1 a*)) −
(*cdf M2 x* − (*cdf M2 b* − *cdf M2 a*))) **by** (*subst* ∗, *simp*)
also have ... ≤ *abs* (*cdf M1 x* − (*cdf M1 b* − *cdf M1 a*)) +
abs (*cdf M2 x* − (*cdf M2 b* − *cdf M2 a*)) **by** (*rule abs-triangle-ineq4*)
also have ... ≤ ε / 2 + ε / 2 **by** (*rule add-mono* [*OF 1 2*])
finally have *abs* (*cdf M1 x* − *cdf M2 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

46.2 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 (λ*n*. *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*. (λ*n*. *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..0. 1 − iexp (t * x)) + (CLBINT t=0..u. 1
− iexp (t * x))
      using ‹u > 0›
      apply (subst interval-integral-sum)
      apply (simp add: min-absorb1 min-absorb2 max-absorb1 max-absorb2)
      apply (rule interval-integrable-isCont)
      apply auto
      done
     also have ... = (CLBINT t=ereal 0..u. 1 − iexp (t * −x)) + (CLBINT
t=ereal 0..u. 1 − iexp (t * x))
      apply (subgoal-tac 0 = ereal 0, erule ssubst)
      by (subst interval-integral-reflect, auto)
    also have ... = (LBINT t=ereal 0..u. 2 − 2 * cos (t * x))
      apply (subst interval-lebesgue-integral-add (2) [symmetric])
      apply ((rule interval-integrable-isCont, auto)+) [2]
      unfolding exp-Euler cos-of-real
      apply (simp add: of-real-mult interval-lebesgue-integral-of-real[symmetric])
      done
    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: ⋀u n. u > 0 ⟹ Re (CLBINT t:{−u..u}. 1 − char (M n)
t) ≥
    u * measure (M n) {x. abs x ≥ 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]: ⋀x. complex-integrable (M n) (λt. exp (ı * complex-of-real
(x * t)))
      by (rule Mn.integrable-const-bound [where B = 1], auto)
    have Mn3: set-integrable (M n ⨂M lborel) (UNIV × {− u..u}) (λa. 1 − exp
```

```
(i * complex-of-real (snd a * fst a)))
      using ‹0 < u›
      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))))
      by (rule 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')
    also have ... = (CLINT x | M n. (if x = 0 then 0 else 2 * (u − sin (u * x)
/ x)))
     using ‹u > 0› by (intro 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 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› by (subst 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›
        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}
    by (simp add: Mn.emeasure-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: ⋀ε. ε > 0 ⟹ ∃ a b. a < b ∧ (∀ n. 1 − ε < measure (M n)
{a<..b})
 proof −
   fix ε :: real
   assume ε > 0
   note M'.isCont-char [of 0]
   hence ∃ d>0. ∀ t. abs t < d ⟶ cmod (char M' t − 1) < ε / 4
     apply (subst (asm) continuous-at-eps-delta)
     apply (drule-tac x = ε / 4 in spec)
    using ‹ε > 0› by (auto simp add: dist-real-def dist-complex-def M'.char-zero)
   then obtain d where d > 0 ∧ (∀ t. (abs t < d ⟶ cmod (char M' t − 1) <
ε / 4)) ..
   hence d0: d > 0 and d1: ⋀t. abs t < d ⟹ cmod (char M' t − 1) < ε / 4
by auto
   have 1: ⋀x. cmod (1 − char M' x) ≤ 2
   by (rule order-trans [OF norm-triangle-ineq4], auto simp add: M'.cmod-char-le-1)
   then have 2: ⋀u v. complex-set-integrable lborel {u..v} (λx. 1 − char M' x)
   by (intro integrableI-bounded-set-indicator[where B=2]) (auto simp: emeasure-lborel-Icc-eq)
   have 3: ⋀u v. set-integrable lborel {u..v} (λx. cmod (1 − char M' x))
   by (intro borel-integrable-compact[OF compact-Icc] continuous-at-imp-continuous-on
            continuous-intros ballI M'.isCont-char continuous-intros)
  have cmod (CLBINT t:{−d/2..d/2}. 1 − char M' t) ≤ LBINT t:{−d/2..d/2}.
cmod (1 − char M' t)
     using integral-norm-bound[OF 2] by simp
   also have ... ≤ LBINT t:{−d/2..d/2}. ε / 4
     apply (rule integral-mono [OF 3])
     apply (simp add: emeasure-lborel-Icc-eq)
     apply (case-tac x ∈ {−d/2..d/2}, auto)
     apply (subst norm-minus-commute)
     apply (rule less-imp-le)
     apply (rule d1 [simplified])
     using d0 by auto
   also with d0 have ... = d * ε / 4
     by simp
   finally have bound: cmod (CLBINT t:{−d/2..d/2}. 1 − char M' t) ≤ d * ε
/ 4 .
   { fix n x
     have cmod (1 − char (M n) x) ≤ 2
     by (rule order-trans [OF norm-triangle-ineq4], auto simp add: Mn.cmod-char-le-1)
   } note bd1 = this
    have (λn. CLBINT t:{−d/2..d/2}. 1 − char (M n) t) ⟶⟶ (CLBINT
t:{−d/2..d/2}. 1 − char M' t)
     using bd1
```

apply (*intro integral-dominated-convergence*[**where** $w{=}\lambda x.\ indicator\ \{-d/2..d/2\}\ x *_R 2$])
apply (*auto intro*!: *char-conv tendsto-intros*
simp: *emeasure-lborel-Icc-eq*
split: *split-indicator*)
done
hence *eventually* $(\lambda n.\ cmod\ ((CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t) -$
$(CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ M'\ t)) < d * \varepsilon\ /\ 4)$ *sequentially*
using *d0* ‹$\varepsilon > 0$› **apply** (*subst* (*asm*) *tendsto-iff*)
by (*subst* (*asm*) *dist-complex-def*, *drule spec*, *erule mp*, *auto*)
hence $\exists N.\ \forall n \geq N.\ cmod\ ((CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t) -$
$(CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ M'\ t)) < d * \varepsilon\ /\ 4$ **by** (*simp add*: *eventually-sequentially*)
then guess N **..**
hence N: $\bigwedge n.\ n \geq N \Longrightarrow cmod\ ((CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t) -$
$(CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ M'\ t)) < d * \varepsilon\ /\ 4$ **by** *auto*
{ fix n
assume $n \geq N$
have $cmod\ (CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t) =$
$cmod\ ((CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t) - (CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ M'\ t)$
$+ (CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ M'\ t))$ **by** *simp*
also have $\ldots \leq cmod\ ((CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t) -$
$(CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ M'\ t)) + cmod(CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ M'\ t)$
by (*rule norm-triangle-ineq*)
also have $\ldots < d * \varepsilon\ /\ 4 + d * \varepsilon\ /\ 4$
by (*rule add-less-le-mono* [*OF N* [*OF* ‹$n \geq N$›] *bound*])
also have $\ldots = d * \varepsilon\ /\ 2$ **by** *auto*
finally have $cmod\ (CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t) < d * \varepsilon\ /\ 2$ **.**
hence $d * \varepsilon\ /\ 2 > Re\ (CLBINT\ t{:}\{-d/2..d/2\}.\ 1 - char\ (M\ n)\ t)$
by (*rule order-le-less-trans* [*OF complex-Re-le-cmod*])
hence $d * \varepsilon\ /\ 2 > Re\ (CLBINT\ t{:}\{-(d/2)..d/2\}.\ 1 - char\ (M\ n)\ t)$ (**is** - > *?lhs*) **by** *simp*
also have $?lhs \geq (d\ /\ 2) * measure\ (M\ n)\ \{x.\ abs\ x \geq 2\ /\ (d\ /\ 2)\}$
using *d0* **by** (*intro main-bound*, *simp*)
finally (*xtrans*) **have** $d * \varepsilon\ /\ 2 > (d\ /\ 2) * measure\ (M\ n)\ \{x.\ abs\ x \geq 2\ /\ (d\ /\ 2)\}$ **.**
with *d0* ‹$\varepsilon > 0$› **have** $\varepsilon > measure\ (M\ n)\ \{x.\ abs\ x \geq 2\ /\ (d\ /\ 2)\}$ **by** (*simp add*: *field-simps*)
hence $\varepsilon > 1 - measure\ (M\ n)\ (UNIV - \{x.\ abs\ x \geq 2\ /\ (d\ /\ 2)\})$
apply (*subst Mn.borel-UNIV* [*symmetric*])
by (*subst Mn.prob-compl*, *auto*)
also have $UNIV - \{x.\ abs\ x \geq 2\ /\ (d\ /\ 2)\} = \{x.\ -(4\ /\ d) < x \wedge x < (4\ /\ d)\}$
using *d0* **apply** (*auto simp add*: *field-simps*)

apply (*case-tac* $x \geq 0$, *auto simp add*: *field-simps*)
apply (*subgoal-tac* $0 \leq x * d$, *arith*, *rule mult-nonneg-nonneg*, *auto*)
apply (*case-tac* $x \geq 0$, *auto simp add*: *field-simps*)
apply (*subgoal-tac* $x * d \leq 0$, *arith*)
apply (*rule mult-nonpos-nonneg*, *auto*)
by (*case-tac* $x \geq 0$, *auto simp add*: *field-simps*)
finally have *measure* $(M\ n)$ $\{x.\ -(4\ /\ d) < x \wedge x < (4\ /\ d)\} > 1 - \varepsilon$
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 $(\lambda k.\ measure\ (M\ n)\ \{-\ real\ k<..real\ k\}) \longrightarrow$
measure $(M\ n)$ $(UN\ (k :: nat).\ \{-\ real\ k<..real\ k\})$
by (*rule Mn.finite-Lim-measure-incseq*, *auto simp add*: *incseq-def*)
hence $(\lambda k.\ measure\ (M\ n)\ \{-\ real\ k<..real\ k\}) \longrightarrow 1$
using *Mn.prob-space* **unfolding** $*$ *Mn.borel-UNIV* **by** *simp*
hence *eventually* $(\lambda k.\ measure\ (M\ n)\ \{-\ real\ k<..real\ k\} > 1 - \varepsilon)$ *sequentially*
apply (*elim order-tendstoD* (*1*))
using ‹$\varepsilon > 0$› **by** *auto*
} note $7 = this$
{ fix n :: *nat*
have *eventually* $(\lambda k.\ \forall m < n.\ measure\ (M\ m)\ \{-\ real\ k<..real\ k\} > 1 - \varepsilon)$ *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** $\exists K :: nat.\ \forall k \geq K.\ \forall m<N.\ 1 - \varepsilon <$
Sigma-Algebra.measure $(M\ m)$ $\{-\ real\ k<..real\ k\}$
by (*auto simp add*: *eventually-sequentially*)
hence $\exists K :: nat.\ \forall m<N.\ 1 - \varepsilon < Sigma\text{-}Algebra.measure\ (M\ m)\ \{-\ real\ K<..real\ K\}$ **by** *auto*
then obtain K :: *nat* **where**
$\forall m<N.\ 1 - \varepsilon < Sigma\text{-}Algebra.measure\ (M\ m)\ \{-\ real\ K<..real\ K\}$ **..**
hence K: $\bigwedge m.\ m < N \Longrightarrow 1 - \varepsilon < Sigma\text{-}Algebra.measure\ (M\ m)\ \{-\ real\ K<..real\ K\}$
by *auto*
let $?K' = max\ K\ (4\ /\ d)$
have $-?K' < ?K' \wedge (\forall n.\ 1 - \varepsilon < measure\ (M\ n)\ \{-?K'<..?K'\})$
using *d0* **apply** *auto*
apply (*rule max.strict-coboundedI2*, *auto*)
proof −
fix n
show $1 - \varepsilon < measure\ (M\ n)\ \{-\ max\ (real\ K)\ (4\ /\ d)<..max\ (real\ K)\ (4\ /\ d)\}$
apply (*case-tac* $n < N$)

apply (*rule order-less-le-trans*)
apply (*erule K*)
apply (*rule Mn.finite-measure-mono*, *auto*)
apply (*rule order-less-le-trans*)
apply (*rule 6*, *erule leI*)
by (*rule Mn.finite-measure-mono*, *auto*)
qed
thus $\exists\, a\ b.\ a < b \wedge (\forall\, n.\ 1 - \varepsilon < measure\ (M\ n)\ \{a<..b\})$ **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*: *subseq s*
assume *nu*: *weak-conv-m* $(M \circ s)\ \nu$
assume $*$: *real-distribution* ν
have *2*: $\bigwedge n.$ *real-distribution* $((M \circ s)\ n)$ **unfolding** *comp-def* **by** (*rule assms*)
have *3*: $\bigwedge t.\ (\lambda n.\ char\ ((M \circ s)\ n)\ t) \longrightarrow char\ \nu\ t$ **by** (*intro levy-continuity1* [*OF 2* $*$ *nu*])
have *4*: $\bigwedge t.\ (\lambda n.\ char\ ((M \circ s)\ n)\ t) = ((\lambda n.\ char\ (M\ n)\ t) \circ s)$ **by** (*rule ext*, *simp*)
have *5*: $\bigwedge t.\ (\lambda n.\ char\ ((M \circ s)\ n)\ t) \longrightarrow char\ M'\ t$
by (*subst 4*, *rule LIMSEQ-subseq-LIMSEQ* [*OF - s*], *rule assms*)
hence *char* ν = *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

47 The Central Limit Theorem

theory *Central-Limit-Theorem*
imports *Levy*
begin

theorem (**in** *prob-space*) *central-limit-theorem*:
fixes *X* :: *nat* $\Rightarrow$ *'a* $\Rightarrow$ *real*
and μ :: *real measure*
and σ :: *real*
and *S* :: *nat* $\Rightarrow$ *'a* $\Rightarrow$ *real*
assumes *X-indep*: *indep-vars* $(\lambda i.\ borel)\ X\ UNIV$
and *X-integrable*: $\bigwedge n.$ *integrable M* $(X\ n)$
and *X-mean-0*: $\bigwedge n.$ *expectation* $(X\ n) = 0$
and σ*-pos*: $\sigma > 0$
and *X-square-integrable*: $\bigwedge n.$ *integrable M* $(\lambda x.\ (X\ n\ x)^2)$

and *X-variance*: $\bigwedge n.\ variance\ (X\ n) = \sigma^2$
and *X-distrib*: $\bigwedge n.\ distr\ M\ borel\ (X\ n) = \mu$
defines $S\ n \equiv \lambda x.\ \sum i{<}n.\ X\ i\ x$
shows *weak-conv-m* $(\lambda n.\ distr\ M\ borel\ (\lambda x.\ S\ n\ x\ /\ sqrt\ (n * \sigma^2)))$ *std-normal-distribution*
proof −
let $?S' = \lambda n\ x.\ S\ n\ x\ /\ sqrt\ (real\ n * \sigma^2)$
def $\varphi \equiv \lambda n.\ char\ (distr\ M\ borel\ (?S'\ n))$
def $\psi \equiv \lambda n\ t.\ char\ \mu\ (t\ /\ sqrt\ (\sigma^2 * n))$

have *X-rv* [*simp*, *measurable*]: $\bigwedge n.\ random\text{-}variable\ borel\ (X\ n)$
using *X-indep* **unfolding** *indep-vars-def2* **by** *simp*
interpret μ: *real-distribution* μ
by (*subst X-distrib* [*symmetric*, *of 0*], *rule real-distribution-distr*, *simp*)

have μ-*integrable* [*simp*]: *integrable* $\mu\ (\lambda x.\ x)$
and μ-*mean-integrable* [*simp*]: $\mu.expectation\ (\lambda x.\ x) = 0$
and μ-*square-integrable* [*simp*]: *integrable* $\mu\ (\lambda x.\ x\hat{}2)$
and μ-*variance* [*simp*]: $\mu.expectation\ (\lambda x.\ x\hat{}2) = \sigma^2$
using *assms* **by** (*simp-all add*: *X-distrib* [*symmetric*, *of 0*] *integrable-distr-eq integral-distr*)

have *main*: $\forall_F\ n\ in\ sequentially.$
$cmod\ (\varphi\ n\ t - (1 + (-(t\hat{}2)\ /\ 2)\ /\ n)\hat{}n) \leq$
$t^2\ /\ (6 * \sigma^2) * (LINT\ x|\mu.\ min\ (6 * x^2)\ (|t\ /\ sqrt\ (\sigma^2 * n)| * |x|\ \hat{}\ 3))$ **for** t
proof (*rule eventually-sequentiallyI*)
fix $n :: nat$
assume $n \geq nat\ (ceiling\ (t\hat{}2\ /\ 4))$
hence n: $n \geq t\hat{}2\ /\ 4$ **by** (*subst nat-ceiling-le-eq* [*symmetric*])
let $?t = t\ /\ sqrt\ (\sigma^2 * n)$

def $\psi' \equiv \lambda n\ i.\ char\ (distr\ M\ borel\ (\lambda x.\ X\ i\ x\ /\ sqrt\ (\sigma^2 * n)))$
have $*$: $\bigwedge n\ i\ t.\ \psi'\ n\ i\ t = \psi\ n\ t$
unfolding ψ-*def* ψ'-*def char-def*
by (*subst X-distrib* [*symmetric*]) (*auto simp*: *integral-distr*)

have $\varphi\ n\ t = char\ (distr\ M\ borel\ (\lambda x.\ \sum i{<}n.\ X\ i\ x\ /\ sqrt\ (\sigma^2 * real\ n)))\ t$
by (*auto simp*: φ-*def S-def setsum-divide-distrib ac-simps*)
also have $\ldots = (\prod\ i < n.\ \psi'\ n\ i\ t)$
unfolding ψ'-*def*
apply (*rule char-distr-setsum*)
apply (*rule indep-vars-compose2*[**where** $X{=}X$])
apply (*rule indep-vars-subset*)
apply (*rule X-indep*)
apply *auto*
done
also have $\ldots = (\psi\ n\ t)\hat{}n$
by (*auto simp add*: $*$ *setprod-constant*)
finally have φ-*eq*: $\varphi\ n\ t = (\psi\ n\ t)\hat{}n$ **.**

```
    have norm (ψ n t - (1 - ?t^2 * σ² / 2)) ≤ ?t² / 6 * (LINT x|μ. min (6 *
x²) (|?t| * |x| ^ 3))
      unfolding ψ-def by (rule μ.char-approx3, auto)
    also have ?t^2 * σ² = 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 (ψ n t - (1 + (-(t^2) / 2) / n)) ≤
      ?t² / 6 * (LINT x|μ. min (6 * x²) (|?t| * |x| ^ 3)) by simp

    have norm (φ n t - (complex-of-real (1 + (-(t^2) / 2) / n)) ^n) ≤
        n * norm (ψ n t - (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 ... ≤ n * (?t² / 6 * (LINT x|μ. min (6 * x²) (|?t| * |x| ^ 3)))
      by (rule mult-left-mono [OF **], simp)
    also have ... = (t² / (6 * σ²) * (LINT x|μ. min (6 * x²) (|?t| * |x| ^ 3)))
      using σ-pos by (simp add: field-simps min-absorb2)
    finally show norm (φ n t - (1 + (-(t^2) / 2) / n) ^n) ≤
       (t² / (6 * σ²) * (LINT x|μ. min (6 * x²) (|?t| * |x| ^ 3)))
      by simp
  qed

  show ?thesis
  proof (rule levy-continuity)
    fix t
    let ?t = λn. t / sqrt (σ² * n)
    have ⋀x. (λn. min (6 * x²) (|t| * |x| ^ 3 / |sqrt (σ² * real n)|)) ⟶ 0
      using σ-pos
      by (auto simp: real-sqrt-mult min-absorb2
          intro!: tendsto-min[THEN tendsto-eq-rhs] sqrt-at-top[THEN filterlim-compose]
                  filterlim-tendsto-pos-mult-at-top filterlim-at-top-imp-at-infinity
                  tendsto-divide-0 filterlim-real-sequentially)
    then have (λn. LINT x|μ. min (6 * x²) (|?t n| * |x| ^ 3)) ⟶ (LINT x|μ.
0)
      by (intro integral-dominated-convergence [where w = λx. 6 * x^2]) auto
    then have *: (λn. t² / (6 * σ²) * (LINT x|μ. min (6 * x²) (|?t n| * |x| ^ 3)))
⟶ 0
      by (simp only: integral-zero tendsto-mult-right-zero)

    have (λn. complex-of-real ((1 + (-(t^2) / 2) / n) ^n)) ⟶ complex-of-real
(exp (-(t^2) / 2))
      by (rule isCont-tendsto-compose [OF - tendsto-exp-limit-sequentially]) auto
    then have (λn. φ n t) ⟶ complex-of-real (exp (-(t^2) / 2))
      by (rule Lim-transform) (rule Lim-null-comparison [OF main *])
  then show (λn. char (distr M borel (?S' n)) t) ⟶ char std-normal-distribution
```

t
  by (*simp add*: φ*-def char-std-normal-distribution*)
 qed (*auto intro*!: *real-dist-normal-dist simp*: *S-def*)
qed

end

theory *Probability*
imports
  Discrete-Topology
  Complete-Measure
  Projective-Limit
  Probability-Mass-Function
  Stream-Space
  Embed-Measure
  Central-Limit-Theorem
begin

end