Generic Construction of Probability Spaces for Paths of Stochastic Processes in Isabelle/HOL

Fabian Immler

October 13, 2012

Abstract

Stochastic processes are used in probability theory to describe the evolution of random systems over time. The principal mathematical problem is the construction of a probability space for the paths of stochastic processes. The Daniell-Kolmogorov theorem solves this problem: it shows how a family of finite-dimensional distributions defines the distribution of the stochastic process. The construction is generic, i.e., it works for discrete time as well as for continuous time.

Starting from the existing formalizations of measure theory and product probability spaces in Isabelle/HOL, we provide a formal proof of the Daniell-Kolmogorov theorem in Isabelle/HOL. This requires us to formalize concepts from topology, namely polish spaces and regularity of measures on polish spaces.

These results can serve as a foundation to formalize for example discrete-time or continuous-time Markov chains, Markov decision processes, or physical phenomena like Brownian motion.

This work is described in the Master's thesis of Immler [1]

Contents

theory *Auxiliarities*
imports *Probability*
begin

1 Auxiliarities

1.1 Functions: Injective and Inverse

lemma *inj-on-vimage-image-eq*:
  assumes *inj-on* f X $A \subseteq X$ **shows** f −' f ' $A \cap X = A$
using *assms* **by** (*auto simp*: *vimage-image-eq inj-on-def*)

lemma *inv-into-inv-into-superset-eq*:
  assumes *inj-on* f B
  assumes *bij-betw* f A A' $a \in A$ $A \subseteq B$
  shows *inv-into* A' (*inv-into* B f) $a = f\ a$
proof −
  let *?f'* = *inv-into* A f **let** *?e'* = *inv-into* B f
  let *?f''* = *inv-into* A' *?f'* **let** *?e''* = *inv-into* A' *?e'*
  have *1*: *bij-betw* *?f'* A' A **using** *assms* **by** (*auto simp add*: *bij-betw-inv-into*)
  obtain a' **where** *2*: $a' \in A'$ **and** *3*: *?f'* $a' = a$

using *1* ‹$a \in A$› **unfolding** *bij-betw-def* **by** *force*
have $f\ a = a'$ **using** *assms 2 3*
by (*auto simp add*: *bij-betw-def*)
have *inj-on* $?e'\ A'$
proof (*intro inj-onI*)
{ **fix** x **assume** $x \in A'$
hence $x \in f\ `\ A$ **using** *assms*(*2*) **by** (*auto simp*: *bij-betw-def*)
hence *inv-into* $A\ f\ x \in A$ **by** (*rule inv-into-into*)
also note ‹$A \subseteq B$›
finally have *inv-into* $B\ f\ x = ?f'\ x$
using *f-inv-into-f*[*OF* ‹$x \in$ *image* $f\ A$›]
by (*rule inv-into-f-eq*[*OF* ‹*inj-on* $f\ B$›])
}
moreover
fix $x\ y$ **assume** $x \in A'\ y \in A'$ *inv-into* $B\ f\ x =$ *inv-into* $B\ f\ y$
ultimately
have *inv-into* $A\ f\ x =$ *inv-into* $A\ f\ y$ **by** *simp*
thus $x = y$ **by** (*metis 1* ‹$x \in A'$› ‹$y \in A'$› *bij-betw-imp-inj-on inj-onD*)
qed
hence $?e''\ a = a'$ **using** *assms 2* ‹$f\ a = a'$› **by** (*intro inv-into-f-eq*) *auto*
thus $?e''\ a = f\ a$ **using** ‹$f\ a = a'$› **by** *simp*
qed

lemma *f-inv-into-onto*:
fixes $f{::}'a \Rightarrow\ 'b$ **and** $A{::}'a$ *set* **and** $B{::}'b$ *set*
assumes *inj-on* $f\ A\ B \subseteq f\ `\ A$
shows $f\ `$ *inv-into* $A\ f\ `\ B = B$
unfolding *image-image* **using** *assms*
proof *safe*
fix x **assume** $x \in B$
thus $x \in (\lambda x.\ f\ ($*inv-into* $A\ f\ x))\ `\ B$
unfolding *image-def*
using *assms* ‹$x \in B$›
by (*auto simp*: *Bex-def f-inv-into-f intro*!: *exI*[**where** $x{=}x$])
qed (*auto simp*: *f-inv-into-f*)

lemma *inj-on-image-subset-iff*: *inj-on* $f\ (A \cup B)$ ==> ($f`A$ <= $f`B$) = (A<=B)
by (*simp add*: *inj-on-def*, *blast*)

lemma *inv-into-eq*:
assumes *inj-on* $f\ A$ *inj-on* $g\ A$
assumes $x \in g\ `\ A$
assumes $\bigwedge i.\ i \in A \Longrightarrow f\ i = g\ i$
shows *inv-into* $A\ f\ x =$ *inv-into* $A\ g\ x$
proof −
from *assms* **obtain** y **where** $g\ y = x\ y \in A$ **by** *auto*
show *?thesis*
apply (*rule inv-into-f-eq*[*OF* ‹*inj-on* $f\ A$›])
apply (*rule inv-into-into*[*OF* ‹$x \in$ *image* $g\ A$›])

apply (*subst inv-into-f-eq*[*OF* ‹*inj-on g A*›])
using *assms* ‹$g\ y = x$› ‹$y \in A$› **by** *auto*
qed

lemma *inv-into-eq'*:
assumes *inj-on f A inj-on f B*
assumes $x \in f\ `\ (A \cap B)$
shows *inv-into A f x* = *inv-into B f x*
using *assms*
by (*metis* (*full-types*) *Int-iff f-inv-into-f inv-into-f-f inv-into-into*)

1.2 Topology

lemma *borel-def-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 ... $\in$ *sigma-sets UNIV* (*Collect closed*)
by (*rule sigma-sets.Compl*)
(*auto intro*!: *sigma-sets.Basic simp*: ‹*open x*›)
finally show $x \in$ *sigma-sets UNIV* (*Collect closed*) **by** *simp*
next
fix *x* :: *'a set* **assume** *closed x*
hence $x = UNIV - (UNIV - x)$ **by** *auto*
also have ... $\in$ *sigma-sets UNIV* (*Collect open*)
by (*rule sigma-sets.Compl*)
(*auto intro*!: *sigma-sets.Basic simp*: ‹*closed x*›)
finally show $x \in$ *sigma-sets UNIV* (*Collect open*) **by** *simp*
qed *simp-all*

lemma *compactE'*:
assumes *compact S* $\forall n \geq m.\ f\ n \in S$
obtains *l r* **where** $l \in S$ *subseq r* ((*f* ∘ *r*) −−−> *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 *compactE*[*OF* ‹*compact S*› *this*] **guess** *l r* **.**
hence $l \in S$ *subseq* ((λ*i*. *m* + *i*) *o r*) ∧ (*f* ∘ ((λ*i*. *m* + *i*) *o r*)) −−−−> *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* ∧ (*f* ∘ *r*) −−−−> *l* **by** *blast*
qed

lemma *compact-Union* [*intro*]: *finite S* ⟹ ∀ *T*∈*S*. *compact T* ⟹ *compact* ($\bigcup S$)
by (*induct set*: *finite*) *auto*

lemma *closed-UN* [*intro*]: *finite A* ⟹ ∀ *x*∈*A*. *compact* (*B x*) ⟹ *compact* ($\bigcup x \in A.$ *B x*)
unfolding *SUP-def* **by** (*rule compact-Union*) *auto*

1.3 Measures

lemma
UN-finite-countable-eq-Un:
fixes f :: $'a$::*countable set* $\Rightarrow$ -
assumes $\bigwedge s.\ P\ s \Longrightarrow$ *finite* s
shows $\bigcup\{f\ s|s.\ P\ s\} = (\bigcup n$::*nat*. *let* s = *set* (*from-nat* n) *in if* $P\ s$ *then* $f\ s$ *else* $\{\})$
proof *safe*
fix $x\ X\ s$ **assume** $x \in f\ s\ P\ s$
moreover with *assms* **obtain** l **where** s = *set* l **using** *finite-list* **by** *auto*
ultimately show $x \in (\bigcup n.$ *let* s = *set* (*from-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-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*: *split-if-asm*)
qed

lemma
countable-finite-comprehension:
fixes f :: $'a$::*countable set* $\Rightarrow$ -
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 −
from *UN-finite-countable-eq-Un*[*of* $P\ f$] *assms*
have $\bigcup\{f\ s|s.\ P\ s\} = (\bigcup n.$ *let* s = *set* (*from-nat* n) *in if* $P\ s$ *then* $f\ s$ *else* $\{\})$ **by** *simp*
also have ... $\in$ *sets* M **using** *assms* **by** (*auto simp*: *Let-def*)
finally show *?thesis* **.**
qed

lemma (**in** *ring-of-sets*) *union*:
assumes f: *positive* $M\ f$ *additive* $M\ f$ **and** $A \in M\ B \in M$
shows $f\ (A \cup B) = f\ A + f\ (B - A)$
using *assms* **by** (*subst additiveD*[*OF* ‹*additive* $M\ f$›, *symmetric*]) *auto*

lemma (**in** *ring-of-sets*) *plus*:
assumes f: *positive* $M\ f$ *additive* $M\ f$ **and** $A \in M\ B \in M$
shows $f\ B = f\ (A \cap B) + f\ (B - A)$
proof −
have $A \cap B \cup (B - A) = B$ **by** *auto*
thus *?thesis* **using** *assms*
by (*subst additiveD*[*OF* ‹*additive* $M\ f$›, *symmetric*]) *auto*
qed

lemma (**in** *ring-of-sets*) *union-inter-minus-equality*:
assumes f: *positive* $M\ f$ *additive* $M\ f$ **and** $A \in M\ B \in M$
shows $f\ (A \cup B) + f\ (A \cap B) + f\ (B - A) = f\ A + f\ B + f\ (B - A)$

using *union*[*OF assms*] *plus*[*OF assms*] **by** (*simp add*: *ac-simps*)

lemma (**in** *ring-of-sets*) *union-plus-inter-equality*:
assumes *f*: *positive M f additive M f* **and** $A \in M$ $B \in M$
shows $f\ (A \cup B) + f\ (A \cap B) = f\ A + f\ B$
proof *cases*
assume $f\ (B - A) = \infty$ **hence** $f\ B = \infty$ $f\ (A \cup B) = \infty$
using *plus*[*OF assms*] *union*[*OF assms*] **by** *simp-all*
thus *?thesis* **by** *simp*
next
assume $f\ (B - A) \neq \infty$ **thus** *?thesis* **using** *union-inter-minus-equality*[*OF assms*] *f assms*
by (*subst* (*asm*) *ereal-add-cancel-right*) (*auto dest*: *positiveD2*[**where** $A{=}B{-}A$])
qed

lemma *emeasure-union-plus-inter-equality*:
assumes $A \in$ *sets* M $B \in$ *sets* M
shows $M\ (A \cup B) + M\ (A \cap B) = M\ A + M\ B$
by (*rule union-plus-inter-equality*[*OF emeasure-positive emeasure-additive assms*])

lemma (**in** *finite-measure*) *measure-union*:
assumes $A \in$ *sets* M $B \in$ *sets* M
shows *measure* $M\ (A \cup B) =$ *measure* $M\ A +$ *measure* $M\ B -$ *measure* $M\ (A \cap B)$
using *union-plus-inter-equality*[*OF emeasure-positive emeasure-additive assms*]
by (*simp add*: *emeasure-eq-measure*)

lemma (**in** *ring-of-sets*) *subtractive*:
assumes *f*: *positive M f additive M f* **and** $A \in M$ $B \in M$ **and** $A \subseteq B$
and $f\ A < \infty$
shows $f\ (B - A) = f\ B - f\ A$
proof −
note *union-inter-minus-equality*[*OF assms*(*1*−*4*)]
moreover have $A \cup B = B$ **using** *assms* **by** *auto*
ultimately have $f\ B = f\ A + f\ (B - A)$ **using** *assms*
by (*subst additiveD*[*OF* ‹*additive M f*›, *symmetric*]) *auto*
hence $f\ B - f\ A = f\ A + f\ (B - A) - f\ A$ **using** *assms* **by** *simp*
also have $\ldots = f\ (B - A) + f\ A - f\ A$ **using** *assms* **by** (*auto simp*: *ac-simps*)
also have $\ldots = f\ (B - A) + (f\ A - f\ A)$
by (*metis ab-semigroup-add-class.add-ac*(*1*) *ereal-minus*(*6*) *ereal-uminus-uminus*)
also have $f\ A - f\ A = 0$ **using** *assms* **by** (*auto simp*: *positive-def*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *ring-of-sets*) *subadditive*:
assumes *f*: *positive M f additive M f* **and** *A*: *range* $A \subseteq M$ **and** *S*: *finite S*
shows $f\ (\bigcup i{\in}S.\ A\ i) \leq (\sum i{\in}S.\ f\ (A\ i))$
using *S*
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 *finite-Union*:
fixes *A*::*'a*::*countable set*
assumes $\bigwedge i.\ i \in A \Longrightarrow B\ i \in sigma\text{-}sets\ sp\ C$
shows $\bigcup B\ `\ A \in sigma\text{-}sets\ sp\ C$
proof *cases*
assume $A = \{\}$ **thus** *?thesis* **by** (*simp add*: *Empty*)
next
assume $A \neq \{\}$ **then obtain** *a* **where** $a \in A$ **by** *auto*
have *UN*: *UNION A B* =
UNION UNIV ($\lambda i.$ *if from-nat i* $\in$ *A then B* (*from-nat i*) *else B a*) **using** ‹$a \in A$›
apply *auto*
proof −
case *goal1* **thus** *?case*
by (*auto intro*: *exI*[**where** *x=to-nat xa*])
next
case *goal2* **thus** *?case* **by** (*auto split*: *split-if-asm simp add*: *Bex-def*)
qed
show *?thesis* **using** *assms* ‹$a{\in}A$› **by** (*auto intro*: *Union simp*: *UN*)
qed

1.4 Enumeration of Finite Set

definition *enum-finite-max* $J = (SOME\ n.\ \exists\ f.\ J = f\ `\ \{i.\ i < n\} \wedge inj\text{-}on\ f\ \{i.\ i < n\})$

definition *enum-finite* **where**
enum-finite $J =$
$(SOME\ f.\ J = f\ `\ \{i{::}nat.\ i < enum\text{-}finite\text{-}max\ J\} \wedge inj\text{-}on\ f\ \{i.\ i <$

enum-finite-max J})

lemma *enum-finite-max*:
 assumes *finite J*
 shows $\exists f$::*nat*$\Rightarrow$'*a*. $J = f$ ` {*i*. $i <$ *enum-finite-max J*} $\wedge$ *inj-on f* {*i*. $i <$ *enum-finite-max J*}
 unfolding *enum-finite-max-def*
 by (*rule someI-ex*) (*rule finite-imp-nat-seg-image-inj-on*[*OF* ‹*finite J*›])

lemma *enum-finite*:
 assumes *finite J*
 shows $J =$ *enum-finite J* ` {*i*::*nat*. $i <$ *enum-finite-max J*} $\wedge$
 inj-on (*enum-finite J*) {*i*::*nat*. $i <$ *enum-finite-max J*}
 unfolding *enum-finite-def*
 by (*rule someI-ex*[*of* λf. $J = f$ ` {*i*::*nat*. $i <$ *enum-finite-max J*} $\wedge$
 inj-on f {*i*. $i <$ *enum-finite-max J*}])
 (*rule enum-finite-max*[*OF* ‹*finite J*›])

lemma *in-set-enum-exist*:
 assumes *finite A*
 assumes $y \in A$
 shows $\exists\, i$. $y =$ *enum-finite A i*
 using *assms enum-finite* **by** *auto*

1.5 Enumeration of Countable Union of Finite Sets

locale *finite-set-sequence* =
 fixes *Js*::*nat* $\Rightarrow$ '*a set*
 assumes *finite-seq*[*simp*]: *finite* (*Js n*)
begin

definition *set-of-Un* **where** *set-of-Un j* = (*LEAST n*. $j \in$ *Js n*)

definition *index-in-set* **where** *index-in-set J j* = (*SOME n*. $j =$ *enum-finite J n*)

definition *Un-to-nat* **where**
 Un-to-nat j = *to-nat* (*set-of-Un j*, *index-in-set* (*Js* (*set-of-Un j*)) *j*)

lemma *inj-on-Un-to-nat*:
 shows *inj-on Un-to-nat* ($\bigcup n$::*nat*. *Js n*)
proof (*rule inj-onI*)
 fix *x y*
 assume $x \in (\bigcup n.\ Js\ n)$ $y \in (\bigcup n.\ Js\ n)$
 then obtain *ix iy* **where** *ix*: $x \in$ *Js ix* **and** *iy*: $y \in$ *Js iy* **by** *blast*
 assume *Un-to-nat x* = *Un-to-nat y*
 hence *set-of-Un x* = *set-of-Un y*
 index-in-set (*Js* (*set-of-Un y*)) *y* = *index-in-set* (*Js* (*set-of-Un x*)) *x*
 by (*auto simp*: *Un-to-nat-def*)
 moreover

have $y \in Js$ $(set\text{-}of\text{-}Un\ y)$ **unfolding** *set-of-Un-def* **using** *iy* **by** (*rule LeastI*)
have $x \in Js$ $(set\text{-}of\text{-}Un\ x)$ **unfolding** *set-of-Un-def* **using** *ix* **by** (*rule LeastI*)
have $y = enum\text{-}finite\ (Js\ (set\text{-}of\text{-}Un\ y))\ (index\text{-}in\text{-}set\ (Js\ (set\text{-}of\text{-}Un\ y))\ y)$
unfolding *index-in-set-def*
apply (*rule someI-ex*)
using ‹$y \in Js$ $(set\text{-}of\text{-}Un\ y)$› *finite-seq*
apply (*auto intro*!: *in-set-enum-exist*)
done
moreover have $x = enum\text{-}finite\ (Js\ (set\text{-}of\text{-}Un\ x))\ (index\text{-}in\text{-}set\ (Js\ (set\text{-}of\text{-}Un\ x))\ x)$
unfolding *index-in-set-def*
apply (*rule someI-ex*)
using ‹$x \in Js$ $(set\text{-}of\text{-}Un\ x)$› *finite-seq*
apply (*auto intro*!: *in-set-enum-exist*)
done
ultimately show $x = y$ **by** *simp*
qed

lemma *inj-Un*[*simp*]:
shows *inj-on* (*Un-to-nat*) (*Js n*)
by (*intro subset-inj-on*[*OF inj-on-Un-to-nat*]) (*auto simp*: *assms*)

lemma *Un-to-nat-injectiveD*:
assumes $Un\text{-}to\text{-}nat\ x = Un\text{-}to\text{-}nat\ y$
assumes $x \in Js\ i$ $y \in Js\ j$
shows $x = y$
using *assms*
by (*intro inj-onD*[*OF inj-on-Un-to-nat*]) *auto*

end

1.6 Sequence of Properties on Subsequences

lemma *subseq-mono*: **assumes** $subseq\ r\ m < n$ **shows** $r\ m < r\ n$
using *assms* **by** (*auto simp*: *subseq-def*)

locale *subseqs* =
fixes $P::nat \Rightarrow (nat \Rightarrow nat) \Rightarrow (nat \Rightarrow nat) \Rightarrow bool$
assumes *ex-subseq*: $\bigwedge n\ s.\ subseq\ s \Longrightarrow \exists r'.\ subseq\ r' \wedge P\ n\ s\ r'$
begin

primrec *seqseq* **where**
$seqseq\ 0 = id$
$|\ seqseq\ (Suc\ n) = seqseq\ n\ o\ (SOME\ r'.\ subseq\ r' \wedge P\ n\ (seqseq\ n)\ r')$

lemma *seqseq-ex*:
shows $subseq\ (seqseq\ n) \wedge$
$(\exists r'.\ seqseq\ (Suc\ n) = seqseq\ n\ o\ r' \wedge subseq\ r' \wedge P\ n\ (seqseq\ n)\ r')$
proof (*induct n*)

case *0*
let *?P* = $\lambda r'$. *subseq* r' ∧ *P 0 id* r'
let *?r* = *Eps ?P*
have *?P ?r* **using** *ex-subseq*[*of id 0*] **by** (*intro someI-ex*[*of ?P*]) (*auto simp*: *subseq-def*)
thus *?case* **by** (*auto simp*: *subseq-def*) (*simp add*: *id-def*)
next
case (*Suc n*)
then obtain r' **where**
Suc': *seqseq* (*Suc n*) = *seqseq n* ∘ r' *subseq* (*seqseq n*) *subseq* r'
P n (*seqseq n*) r'
by *blast*
let *?P* = $\lambda r'a$. *subseq* ($r'a$) ∧ *P* (*Suc n*) (*seqseq n o* r') $r'a$
let *?r* = *Eps ?P*
have *?P ?r* **using** *ex-subseq*[*of seqseq n o* r' *Suc n*] *Suc'*
by (*intro someI-ex*[*of ?P*]) (*auto intro*: *subseq-o simp*: *o-assoc*)
moreover have *seqseq* (*Suc* (*Suc n*)) = *seqseq n* ∘ r' ∘ *?r*
by (*subst seqseq.simps*) (*simp only*: *Suc' o-assoc*)
moreover note *subseq-o*[*OF* ‹*subseq* (*seqseq n*)› ‹*subseq* r'›]
ultimately show *?case* **unfolding** *Suc'* **by** (*auto simp*: *o-def*)
qed

lemma *subseq-seqseq*:
shows *subseq* (*seqseq n*) **using** *seqseq-ex*[*OF assms*] **by** *auto*

definition *reducer* **where** *reducer n* = (*SOME* r'. *subseq* r' ∧ *P n* (*seqseq n*) r')

lemma *subseq-reducer*: *subseq* (*reducer n*) **and** *reducer-reduces*: *P n* (*seqseq n*) (*reducer n*)
unfolding *atomize-conj* **unfolding** *reducer-def* **using** *subseq-seqseq*
by (*rule someI-ex*[*OF ex-subseq*])

lemma *seqseq-reducer*[*simp*]:
seqseq (*Suc n*) = *seqseq n o reducer n*
by (*simp add*: *reducer-def*)

declare *seqseq.simps*(*2*)[*simp del*]

definition *diagseq* **where** *diagseq i* = *seqseq i i*

lemma *diagseq-mono*: *diagseq n* < *diagseq* (*Suc n*)
unfolding *diagseq-def seqseq-reducer o-def*
by (*metis subseq-mono*[*OF subseq-seqseq*] *less-le-trans lessI seq-suble subseq-reducer*)

lemma *subseq-diagseq*: *subseq diagseq*
using *diagseq-mono* **by** (*simp add*: *subseq-Suc-iff diagseq-def*)

primrec *fold-reduce* **where**
fold-reduce n 0 = *id*

| *fold-reduce n* (*Suc k*) = *fold-reduce n k o reducer* (*n* + *k*)

lemma *subseq-fold-reduce*: *subseq* (*fold-reduce n k*)
proof (*induct k*)
  case (*Suc k*) **from** *subseq-o*[*OF this subseq-reducer*] **show** *?case* **by** (*simp add*: *o-def*)
qed (*simp add*: *subseq-def*)

lemma *ex-subseq-reduce-index*: *seqseq* (*n* + *k*) = *seqseq n o fold-reduce n k*
  by (*induct k*) *simp-all*

lemma *seqseq-fold-reduce*: *seqseq n* = *fold-reduce 0 n*
  by (*induct n*) (*simp-all*)

lemma *diagseq-fold-reduce*: *diagseq n* = *fold-reduce 0 n n*
  using *seqseq-fold-reduce* **by** (*simp add*: *diagseq-def*)

lemma *fold-reduce-add*: *fold-reduce 0* (*m* + *n*) = *fold-reduce 0 m o fold-reduce m n*
  by (*induct n*) *simp-all*

lemma *diagseq-add*: *diagseq* (*k* + *n*) = (*seqseq k o* (*fold-reduce k n*)) (*k* + *n*)
proof −
  have *diagseq* (*k* + *n*) = *fold-reduce 0* (*k* + *n*) (*k* + *n*)
    by (*simp add*: *diagseq-fold-reduce*)
  also have ... = (*seqseq k o fold-reduce k n*) (*k* + *n*)
    unfolding *fold-reduce-add seqseq-fold-reduce* **..**
  finally show *?thesis* **.**
qed

lemma *diagseq-sub*:
  assumes $m \leq n$ **shows** *diagseq n* = (*seqseq m o* (*fold-reduce m* (*n* − *m*))) *n*
  using *diagseq-add*[*of m n* − *m*] *assms* **by** *simp*

lemma *subseq-diagonal-rest*: *subseq* (λ*x*. *fold-reduce k x* (*k* + *x*))
  unfolding *subseq-Suc-iff fold-reduce.simps o-def*
  by (*metis subseq-mono*[*OF subseq-fold-reduce*] *less-le-trans lessI add-Suc-right seq-suble*
    subseq-reducer)

lemma *diagseq-seqseq*: *diagseq o* (*op* + *k*) = (*seqseq k o* (λ*x*. *fold-reduce k x* (*k* + *x*)))
  by (*auto simp*: *o-def diagseq-add*)

lemma *eventually-sequentially-diagseq*:
  assumes $\bigwedge n\ s\ r.\ P\ n\ s\ r = (\forall\, i.\ Q\ n\ ((s\ o\ r)\ i))$
  shows *eventually* (λ*i*. *Q n* (*diagseq i*)) *sequentially*
  unfolding *eventually-sequentially*
  apply (*intro exI*[**where** *x*=*Suc n*])

apply *safe*
apply (*subst diagseq-sub*) **apply** *simp*
using *reducer-reduces*[*of* n, *simplified assms*, *simplified seqseq-reducer*[*symmetric*]]
apply *simp*
done

lemma *diagseq-holds*:
assumes *seq-property*: $\bigwedge n\ s\ r.\ P\ n\ s\ r = Q\ n\ (s\ o\ r)$
assumes *subseq-closed*: $\bigwedge n\ s\ r.\ subseq\ r \Longrightarrow Q\ n\ s \Longrightarrow Q\ n\ (s\ o\ r)$
shows $P\ n\ diagseq\ (op + (Suc\ n))$
unfolding *seq-property diagseq-seqseq*
by (*intro subseq-closed subseq-diagonal-rest*)
(*auto simp*: *reducer-reduces seq-property*[*symmetric*])

end

1.7 Product Sets

lemma *PiE-def'*: $Pi_E\ I\ A = \{f.\ (\forall\, i \in I.\ f\ i \in A\ i) \wedge f = restrict\ f\ I\}$
apply *auto*
apply (*metis extensional-restrict*)
apply (*metis restrict-extensional*)
done

lemma *prod-emb-def'*: $prod\text{-}emb\ I\ M\ J\ X = \{a \in Pi_E\ I\ (\lambda i.\ space\ (M\ i)).\ restrict\ a\ J \in X\}$
by (*auto simp*: *prod-emb-def*)

lemma *prod-emb-subsetI*:
assumes $F \subseteq G$
shows $prod\text{-}emb\ A\ M\ B\ F \subseteq prod\text{-}emb\ A\ M\ B\ G$
using *assms* **by** (*auto simp*: *prod-emb-def*)

end

theory *Polish-Space*
imports *Auxiliarities*
begin

2 Topological Formalizations Leading to Polish Spaces

2.1 Characterization of Compact Sets

lemma *pos-approach-nat*:
fixes e::*real*
assumes $0 < e$
obtains n::*nat* **where** $1\ /\ (Suc\ n) < e$

proof *atomize-elim*
  have $1 \ / \ real \ (Suc \ (nat \ (ceiling \ (1/e)))) < 1 \ / \ (ceiling \ (1/e))$
    by (*rule divide-strict-left-mono*) (*auto intro*!: *mult-pos-pos simp*: ‹$0 < e$›)
  also have $1 \ / \ (ceiling \ (1/e)) \leq 1 \ / \ (1/e)$
    by (*rule divide-left-mono*) (*auto intro*!: *divide-pos-pos simp*: ‹$0 < e$›)
  also have $\ldots = e$ **by** *simp*
  finally show $\exists n. \ 1 \ / \ real \ (Suc \ n) < e$ **..**
qed

TODO: move to Topology-Euclidean-Space

lemma *compact-eq-totally-bounded*:
  shows $compact \ s \longleftrightarrow complete \ s \wedge (\forall e>0. \ \exists k. \ finite \ k \wedge s \subseteq (\bigcup((\lambda x. \ ball \ x \ e) \ ` \ k)))$
proof (*safe intro*!: *compact-imp-complete*)
  fix *e*::*real*
  def $f \equiv (\lambda x::'a. \ ball \ x \ e) \ ` \ UNIV$
  assume $0 < e$ *compact s*
  hence $(\forall t \in f. \ open \ t) \wedge s \subseteq \bigcup f \longrightarrow (\exists f' \subseteq f. \ finite \ f' \wedge s \subseteq \bigcup f')$
    by (*simp add*: *compact-eq-heine-borel*)
  moreover
  have *d0*: $\bigwedge x::'a. \ dist \ x \ x < e$ **using** ‹$0 < e$› **by** *simp*
  hence $(\forall t \in f. \ open \ t) \wedge s \subseteq \bigcup f$ **by** (*auto simp*: *f-def intro*!: *d0*)
  ultimately have $(\exists f' \subseteq f. \ finite \ f' \wedge s \subseteq \bigcup f')$ **..**
  then guess *K* **.. note** $K = this$
  have $\forall K' \in K. \ \exists k. \ K' = ball \ k \ e$ **using** *K* **by** (*auto simp*: *f-def*)
  then obtain *k* **where** $\bigwedge K'. \ K' \in K \Longrightarrow K' = ball \ (k \ K') \ e$ **unfolding** *bchoice-iff* **by** *blast*
  thus $\exists k. \ finite \ k \wedge s \subseteq \bigcup(\lambda x. \ ball \ x \ e) \ ` \ k$ **using** *K*
    by (*intro exI*[**where** *x*=*k* ‘ *K*]) (*auto simp*: *f-def*)
next
  assume *assms*: *complete s* $\forall e>0. \ \exists k. \ finite \ k \wedge s \subseteq \bigcup(\lambda x. \ ball \ x \ e) \ ` \ k$
  show *compact s*
  proof *cases*
    assume $s = \{\}$ **thus** *compact s* **by** *simp*
  next
    assume $s \neq \{\}$
    show *?thesis*
      unfolding *compact-def*
    proof *safe*
      fix $f::nat \Rightarrow$ - **assume** $\forall n. \ f \ n \in s$ **hence** *f*: $\bigwedge n. \ f \ n \in s$ **by** *simp*
      from *assms* **have** $\forall e. \ \exists k. \ e>0 \longrightarrow finite \ k \wedge s \subseteq (\bigcup((\lambda x. \ ball \ x \ e) \ ` \ k))$ **by** *simp*
      then obtain *K* **where**
        K: $\bigwedge e. \ e > 0 \Longrightarrow finite \ (K \ e) \wedge s \subseteq (\bigcup((\lambda x. \ ball \ x \ e) \ ` \ (K \ e)))$
        unfolding *choice-iff* **by** *blast*
      {
        fix *e*::*real* **and** *f'* **have** *f'*: $\bigwedge n::nat. \ (f \ o \ f') \ n \in s$ **using** *f* **by** *auto*
        assume $e > 0$
        from *K*[*OF this*] **have** *K*: $finite \ (K \ e) \ s \subseteq (\bigcup((\lambda x. \ ball \ x \ e) \ ` \ (K \ e)))$

by *simp-all*
have $\exists\, k{\in}(K\ e).\ \exists\, r.\ \mathit{subseq}\ r \wedge (\forall\, i.\ (f\ o\ f'\ o\ r)\ i \in \mathit{ball}\ k\ e)$
proof (*rule ccontr*)
from K **have** *finite* $(K\ e)$ $K\ e \neq \{\}$ $s \subseteq (\bigcup((\lambda x.\ \mathit{ball}\ x\ e)\ `\ (K\ e)))$
using ‹$s \neq \{\}$›
by *auto*
moreover
assume $\neg\ (\exists\, k{\in}K\ e.\ \exists\, r.\ \mathit{subseq}\ r \wedge (\forall\, i.\ (f \circ f'\ o\ r)\ i \in \mathit{ball}\ k\ e))$
hence $\bigwedge r\ k.\ k \in K\ e \Longrightarrow \mathit{subseq}\ r \Longrightarrow (\exists\, i.\ (f\ o\ f'\ o\ r)\ i \notin \mathit{ball}\ k\ e)$ **by** *simp*
ultimately
show *False* **using** f'
proof (*induct arbitrary*: $s\ f\ f'$ *rule*: *finite-ne-induct*)
case (*singleton* x)
have $\exists\, i.\ (f \circ f'\ o\ \mathit{id})\ i \notin \mathit{ball}\ x\ e$ **by** (*rule singleton*) (*auto simp*: *subseq-def*)
thus *?case* **using** *singleton* **by** (*auto simp*: *ball-def*)
next
case (*insert* x A)
show *?case*
proof *cases*
have *inf-ms*: *infinite* $((f\ o\ f') -`\ s)$ **using** *insert* **by** (*simp add*: *vimage-def*)
have *infinite* $((f\ o\ f') -`\ \bigcup((\lambda x.\ \mathit{ball}\ x\ e)\ `\ (\mathit{insert}\ x\ A)))$
using *insert* **by** (*intro infinite-super*[*OF - inf-ms*]) *auto*
also have $((f\ o\ f') -`\ \bigcup((\lambda x.\ \mathit{ball}\ x\ e)\ `\ (\mathit{insert}\ x\ A))) =$
$\{m.\ (f\ o\ f')\ m \in \mathit{ball}\ x\ e\} \cup \{m.\ (f\ o\ f')\ m \in \bigcup((\lambda x.\ \mathit{ball}\ x\ e)\ `\ A)\}$ **by** *auto*
finally have *infinite* . . . **.**
moreover assume *finite* $\{m.\ (f\ o\ f')\ m \in \mathit{ball}\ x\ e\}$
ultimately have *inf*: *infinite* $\{m.\ (f\ o\ f')\ m \in \bigcup((\lambda x.\ \mathit{ball}\ x\ e)\ `\ A)\}$ **by** *blast*
hence $A \neq \{\}$ **by** *auto* **then obtain** k **where** $k \in A$ **by** *auto*
def $r \equiv \mathit{enumerate}\ \{m.\ (f\ o\ f')\ m \in \bigcup((\lambda x.\ \mathit{ball}\ x\ e)\ `\ A)\}$
have *r-mono*: $\bigwedge n\ m.\ n < m \Longrightarrow r\ n < r\ m$
using *enumerate-mono*[*OF - inf*] **by** (*simp add*: *r-def*)
hence *subseq* r **by** (*simp add*: *subseq-def*)
have *r-in-set*: $\bigwedge n.\ r\ n \in \{m.\ (f\ o\ f')\ m \in \bigcup((\lambda x.\ \mathit{ball}\ x\ e)\ `\ A)\}$
using *enumerate-in-set*[*OF inf*] **by** (*simp add*: *r-def*)
show *False*
proof (*rule insert*)
show $\bigcup(\lambda x.\ \mathit{ball}\ x\ e)\ `\ A \subseteq \bigcup(\lambda x.\ \mathit{ball}\ x\ e)\ `\ A$ **by** *simp*
fix k s **assume** $k \in A$ *subseq* s
thus $\exists\, i.\ (f\ o\ f'\ o\ r\ o\ s)\ i \notin \mathit{ball}\ k\ e$ **using** ‹*subseq* r›
by (*subst* (*2*) *o-assoc*[*symmetric*]) (*intro insert*(*6*) *subseq-o*, *simp-all*)
next
fix n **show** $(f \circ f'\ o\ r)\ n \in \bigcup(\lambda x.\ \mathit{ball}\ x\ e)\ `\ A$ **using** *r-in-set* **by** *auto*
qed

next
assume *inf*: *infinite* {*m*. (*f o f*′) *m* ∈ *ball x e*}
def *r* ≡ *enumerate* {*m*. (*f o f*′) *m* ∈ *ball x e*}
have *r-mono*: ⋀*n m*. *n* < *m* ⟹ *r n* < *r m*
using *enumerate-mono*[*OF - inf*] **by** (*simp add*: *r-def*)
hence *subseq r* **by** (*simp add*: *subseq-def*)
from *insert*(*6*)[*OF insertI1 this*] **obtain** *i* **where** (*f o f*′) (*r i*) ∉ *ball x e* **by** *auto*
moreover
have *r-in-set*: ⋀*n*. *r n* ∈ {*m*. (*f o f*′) *m* ∈ *ball x e*}
using *enumerate-in-set*[*OF inf*] **by** (*simp add*: *r-def*)
hence (*f o f*′) (*r i*) ∈ *ball x e* **by** *simp*
ultimately show *False* **by** *simp*
qed
qed
qed
}
hence ∀*f*′. ∀*e* > *0*. (∃*k*∈*K e*. ∃*r*. *subseq r* ∧ (∀*i*. (*f o f*′ ∘ *r*) *i* ∈ *ball k e*)) **by** *simp*
hence ∀*f*′. ∀*e*. (∃*k*. *e* > *0* ⟶ (*k* ∈ *K e* ∧ (∃*r*. *subseq r* ∧ (∀*i*. (*f o f*′ ∘ *r*) *i* ∈ *ball k e*))))
by (*simp add*: *Bex-def*)
then obtain *k* **where** *k*: ∀*f*′. ∀*e* > *0*. (*k f*′ *e* ∈ *K e* ∧
(∃*r*. *subseq r* ∧ (∀*i*. (*f o f*′ ∘ *r*) *i* ∈ *ball* (*k f*′ *e*) *e*)))
unfolding *choice-iff* **by** *atomize-elim*
let *?P* = λ*n s x*. (∀*i*. (*f o s o x*) *i* ∈ *ball* (*k s* (*1*/*real* (*Suc n*))) (*1*/*real* (*Suc n*)))
interpret *subseqs ?P* **using** *k*
by *unfold-locales simp*
from ‹*complete s*› **have** *limI*: ⋀*f*. (⋀*n*. *f n* ∈ *s*) ⟹ *Cauchy f* ⟹ (∃*l*∈*s*. *f* −−−−> *l*)
by (*simp add*: *complete-def*)
have ∃*l*∈*s*. (*f o diagseq*) −−−−> *l*
proof (*intro limI metric-CauchyI*)
fix *e*::*real* **assume** *0* < *e* **hence** *0* < *e* / *2* **by** *auto*
from *pos-approach-nat*[*OF this*] **guess** *n* **. note** *n* = *this*
show ∃*M*. ∀*m*≥*M*. ∀*n*≥*M*. *dist* ((*f* ∘ *diagseq*) *m*) ((*f* ∘ *diagseq*) *n*) < *e*
proof (*rule exI*[**where** *x*=*Suc n*], *safe*)
fix *m mm* **assume** *Suc n* ≤ *m Suc n* ≤ *mm*
let *?e* = *1* / *real* (*Suc n*)
let *?k* = (*k* (*seqseq n*) *?e*)
from *reducer-reduces*[*of n*]
have ⋀*i*. (*f o seqseq* (*Suc n*)) *i* ∈ *ball ?k ?e* **unfolding** *seqseq-reducer* **by** *simp*
moreover
note *diagseq-sub*[*OF* ‹*Suc n* ≤ *m*›] *diagseq-sub*[*OF* ‹*Suc n* ≤ *mm*›]
ultimately have {(*f o diagseq*) *m*, (*f o diagseq*) *mm*} ⊆ *ball ?k ?e* **by** *auto*
also have ... ⊆ *ball ?k* (*e* / *2*) **using** *n* **by** (*intro subset-ball*) *simp*

finally
have *dist ?k* ((*f* ∘ *diagseq*) *m*) + *dist ?k* ((*f* ∘ *diagseq*) *mm*) < *e* / *2* + *e* /*2*
by (*intro add-strict-mono*) *auto*
hence *dist* ((*f* ∘ *diagseq*) *m*) *?k* + *dist* ((*f* ∘ *diagseq*) *mm*) *?k* < *e*
by (*simp add*: *dist-commute*)
moreover have *dist* ((*f* ∘ *diagseq*) *m*) ((*f* ∘ *diagseq*) *mm*) ≤
dist ((*f* ∘ *diagseq*) *m*) *?k* + *dist* ((*f* ∘ *diagseq*) *mm*) *?k*
by (*rule dist-triangle2*)
ultimately show *dist* ((*f* ∘ *diagseq*) *m*) ((*f* ∘ *diagseq*) *mm*) < *e*
by *simp*
qed
next
fix *n* **show** (*f o diagseq*) *n* ∈ *s* **using** *f* **by** *simp*
qed
thus ∃ *l*∈*s*. ∃ *r*. *subseq r* ∧ (*f* ∘ *r*) −−−−> *l* **using** *subseq-diagseq* **by** *auto*
qed
qed
qed

2.2 Infimum Distance

definition *infdist x A* = *Inf* {*dist x a*|*a*. *a* ∈ *A*}

lemma *infdist-nonneg*:
assumes *A* ≠ {}
shows *0* ≤ *infdist x A*
using *assms* **by** (*auto simp add*: *infdist-def*)

lemma *infdist-le*:
assumes *a* ∈ *A*
assumes *d* = *dist x a*
shows *infdist x A* ≤ *d*
using *assms* **by** (*auto intro*!: *SupInf*.*Inf-lower*[**where** *z*=*0*] *simp add*: *infdist-def*)

lemma *infdist-zero*[*simp*]:
assumes *a* ∈ *A* **shows** *infdist a A* = *0*
proof −
from *infdist-le*[*OF assms*, *of dist a a*] **have** *infdist a A* ≤ *0* **by** *auto*
with *infdist-nonneg*[*of A a*] *assms* **show** *infdist a A* = *0* **by** *auto*
qed

lemma *infdist-triangle*:
assumes *A* ≠ {}
shows *infdist x A* ≤ *infdist y A* + *dist x y*
proof −
from *assms* **obtain** *a* **where** *a* ∈ *A* **by** *auto*
have *infdist x A* ≤ *Inf* {*dist x y* + *dist y a* |*a*. *a* ∈ *A*}
proof

```
    from assms show {dist x y + dist y a |a. a ∈ A} ≠ {} by simp
    fix d assume d ∈ {dist x y + dist y a |a. a ∈ A}
    then obtain a where d: d = dist x y + dist y a a ∈ A by auto
    show infdist x A ≤ d
      unfolding infdist-def
    proof (rule Inf-lower2)
      show dist x a ∈ {dist x a |a. a ∈ A} using ‹a ∈ A› by auto
      show dist x a ≤ d unfolding d by (rule dist-triangle)
      fix d assume d ∈ {dist x a |a. a ∈ A}
      then obtain a where a ∈ A d = dist x a by auto
      thus infdist x A ≤ d by (rule infdist-le)
    qed
  qed
  also have ... = dist x y + infdist y A
  proof (rule Inf-eq, safe)
    fix a assume a ∈ A
    thus dist x y + infdist y A ≤ dist x y + dist y a by (auto intro: infdist-le)
  next
    fix i assume inf: ⋀d. d ∈ {dist x y + dist y a |a. a ∈ A} ⟹ i ≤ d
    hence i − dist x y ≤ infdist y A unfolding infdist-def using ‹a ∈ A›
      by (intro Inf-greatest) (auto simp: field-simps)
    thus i ≤ dist x y + infdist y A by simp
  qed
  finally show ?thesis by simp
qed

lemma
  in-closure-iff-infdist-zero:
  assumes A ≠ {}
  shows x ∈ closure A ⟷ infdist x A = 0
proof
  assume x ∈ closure A
  show infdist x A = 0
  proof (rule ccontr)
    assume infdist x A ≠ 0
    with infdist-nonneg[OF ‹A ≠ {}›, of x] have infdist x A > 0 by auto
    hence ball x (infdist x A) ∩ closure A = {} apply auto
     by (metis ‹0 < infdist x A› ‹x ∈ closure A› closure-approachable dist-commute
        eucl-less-not-refl euclidean-trans(2) infdist-le)
   hence x ∉ closure A by (metis ‹0 < infdist x A› centre-in-ball disjoint-iff-not-equal)
    thus False using ‹x ∈ closure A› by simp
  qed
next
  assume x: infdist x A = 0
  then obtain a where a ∈ A by atomize-elim (metis all-not-in-conv assms)
  show x ∈ closure A unfolding closure-approachable
  proof (safe, rule ccontr)
    fix e::real assume 0 < e
    assume ¬ (∃ y∈A. dist y x < e)
```

hence *infdist x A* $\geq$ *e* **using** ‹*a* $\in$ *A*›
unfolding *infdist-def*
by (*force intro*: *Inf-greatest simp*: *dist-commute*)
with *x* ‹*0* $<$ *e*› **show** *False* **by** *auto*
qed
qed

lemma
in-closed-iff-infdist-zero:
assumes *closed A A* $\neq$ {}
shows *x* $\in$ *A* $\longleftrightarrow$ *infdist x A* = *0*
proof −
have *x* $\in$ *closure A* $\longleftrightarrow$ *infdist x A* = *0*
by (*rule in-closure-iff-infdist-zero*) *fact*
with *assms* **show** *?thesis* **by** *simp*
qed

lemma *continuous-infdist*:
assumes *A* $\neq$ {}
shows *continuous* (*at x*) (λ*x*. *infdist x A*)
unfolding *continuous-at-eps-delta*
proof *safe*
fix *e* ::*real* **assume** *0* $<$ *e*
moreover **{**
fix *y*
from *infdist-triangle*[*OF* ‹*A* $\neq$ {}›, *of x y*] *infdist-triangle*[*OF* ‹*A* $\neq$ {}›, *of y x*]
have *dist* (*infdist y A*) (*infdist x A*) $\leq$ *dist y x* **by** (*simp add*: *dist-commute* *dist-real-def*)
also assume *dist y x* $<$ *e*
finally have *dist* (*infdist y A*) (*infdist x A*) $<$ *e* **.**
} **ultimately show** $\exists$ *d*>*0*. $\forall$ *x'*. *dist x' x* $<$ *d* $\longrightarrow$ *dist* (*infdist x' A*) (*infdist x A*) $<$ *e* **by** *blast*
qed

2.3 Topological Basis

context *topological-space*
begin

definition *topological-basis B* =
(($\forall$ *b*$\in$*B*. *open b*) $\wedge$ ($\forall$ *x*. *open x* $\longrightarrow$ ($\exists$ *B'*. *B'* $\subseteq$ *B* $\wedge$ *Union B'* = *x*)))

lemma *topological-basis-iff*:
assumes $\bigwedge$*B'*. *B'* $\in$ *B* $\Longrightarrow$ *open B'*
shows *topological-basis B* $\longleftrightarrow$ ($\forall$ *O'*. *open O'* $\longrightarrow$ ($\forall$ *x*$\in$*O'*. $\exists$ *B'*$\in$*B*. *x* $\in$ *B'* $\wedge$ *B'* $\subseteq$ *O'*))
(**is** - $\longleftrightarrow$ *?rhs*)
proof *safe*
fix *O'* **and** *x*::*'a*

```
  assume H: topological-basis B open O′ x ∈ O′
  hence (∃ B′⊆B. ⋃ B′ = O′) by (simp add: topological-basis-def)
  then obtain B′ where B′ ⊆ B O′ = ⋃ B′ by auto
  thus ∃ B′∈B. x ∈ B′ ∧ B′ ⊆ O′ using H by auto
next
  assume H: ?rhs
  show topological-basis B using assms unfolding topological-basis-def
  proof safe
    fix O′::′a set assume open O′
    with H obtain f where ∀ x∈O′. f x ∈ B ∧ x ∈ f x ∧ f x ⊆ O′
      by (force intro: bchoice simp: Bex-def)
    thus ∃ B′⊆B. ⋃ B′ = O′
      by (auto intro: exI[where x={f x |x. x ∈ O′}])
  qed
qed

lemma topological-basisI:
  assumes ⋀B′. B′ ∈ B ⟹ open B′
  assumes ⋀O′ x. open O′ ⟹ x ∈ O′ ⟹ ∃ B′∈B. x ∈ B′ ∧ B′ ⊆ O′
  shows topological-basis B
  using assms by (subst topological-basis-iff) auto

lemma topological-basisE:
  fixes O′
  assumes topological-basis B
  assumes open O′
  assumes x ∈ O′
  obtains B′ where B′ ∈ B x ∈ B′ B′ ⊆ O′
proof atomize-elim
  from assms have ⋀B′. B′∈B ⟹ open B′ by (simp add: topological-basis-def)
  with topological-basis-iff assms
  show ∃ B′. B′ ∈ B ∧ x ∈ B′ ∧ B′ ⊆ O′ using assms by (simp add: Bex-def)
qed

end
```

2.4 Enumerable Basis

```
class enumerable-basis = topological-space +
  assumes ex-enum-basis: ∃ f::nat ⇒ ′a set. topological-basis (range f)
begin

definition enum-basis′::nat ⇒ ′a set
  where enum-basis′ = Eps (topological-basis o range)

lemma enumerable-basis′: topological-basis (range enum-basis′)
  using ex-enum-basis
  unfolding enum-basis′-def o-def
  by (rule someI-ex)
```

lemmas *enumerable-basisE′* = *topological-basisE*[*OF enumerable-basis′*]

Extend enumeration of basis, such that it is closed under (finite) Union

definition *enum-basis*::*nat* ⇒ ′*a set*
  where *enum-basis n* = $\bigcup$(*set* (*map enum-basis′* (*from-nat n*)))

lemma
  open-enum-basis:
  assumes *B* ∈ *range enum-basis*
  shows *open B*
  using *assms enumerable-basis′*
  by (*force simp add*: *topological-basis-def enum-basis-def*)

lemma *enumerable-basis*: *topological-basis* (*range enum-basis*)
proof (*rule topological-basisI*[*OF open-enum-basis*])
  fix *O′ x* **assume** *open O′ x* ∈ *O′*
  from *topological-basisE*[*OF enumerable-basis′ this*] **guess** *B′* **. note** *B′* = *this*
  moreover then obtain *n* **where** *B′* = *enum-basis′ n* **by** *auto*
  moreover hence *B′* = *enum-basis* (*to-nat* [*n*]) **by** (*auto simp*: *enum-basis-def*)
  ultimately show ∃ *B′*∈*range enum-basis*. *x* ∈ *B′* ∧ *B′* ⊆ *O′* **by** *blast*
qed

lemmas *enumerable-basisE* = *topological-basisE*[*OF enumerable-basis*]

lemma *open-enumerable-basis-ex*:
  assumes *open X*
  shows ∃ *N*. *X* = ($\bigcup$*n*∈*N*. *enum-basis n*)
proof −
  from *enumerable-basis assms* **obtain** *B′* **where** *B′* ⊆ *range enum-basis X* = *Union B′*
    unfolding *topological-basis-def* **by** *blast*
  hence *Union B′* = ($\bigcup$*n*∈{*n*. *enum-basis n* ∈ *B′*}. *enum-basis n*) **by** *auto*
  with ‹*X* = *Union B′*› **show** *?thesis* **by** *blast*
qed

lemma *open-enumerable-basisE*:
  assumes *open X*
  obtains *N* **where** *X* = ($\bigcup$*n*∈*N*. *enum-basis n*)
  using *assms open-enumerable-basis-ex* **by** (*atomize-elim*) *simp*

Construction of an Increasing Sequence Approximating Open Sets

lemma *empty-basisI*[*intro*]: {} ∈ *range enum-basis*
proof
  show {} = *enum-basis* (*to-nat* ([]::*nat list*)) **by** (*simp add*: *enum-basis-def*)
qed *rule*

lemma *union-basisI*[*intro*]:
  assumes *A* ∈ *range enum-basis B* ∈ *range enum-basis*

shows $A \cup B \in$ *range enum-basis*
proof $-$
from *assms* **obtain** *a b* **where** $A \cup B =$ *enum-basis a* $\cup$ *enum-basis b* **by** *auto*
also have $\ldots =$ *enum-basis* (*to-nat* (*from-nat a* @ *from-nat b*::*nat list*))
by (*simp add*: *enum-basis-def*)
finally show *?thesis* **by** *simp*
qed

lemma *open-imp-Union-of-incseq*:
assumes *open X*
shows $\exists S.$ *incseq S* $\wedge$ $(\bigcup j.\ S\ j) = X \wedge$ *range S* $\subseteq$ *range enum-basis*
proof $-$
from *open-enumerable-basis-ex*[*OF* ‹*open X*›] **obtain** *N* **where** *N*: $X = (\bigcup n{\in}N.$ *enum-basis n*) **by** *auto*
hence *X*: $X = (\bigcup n.$ *if* $n \in N$ *then enum-basis n else* $\{\})$ **by** (*auto split*: *split-if-asm*)
def $S \equiv$ *nat-rec* (*if* $0 \in N$ *then enum-basis 0 else* $\{\}$)
($\lambda n\ S.$ *if* (*Suc n*) $\in N$ *then* $S \cup$ *enum-basis* (*Suc n*) *else S*)
have *S-simps*[*simp*]:
$S\ 0 =$ (*if* $0 \in N$ *then enum-basis 0 else* $\{\}$)
$\bigwedge n.\ S$ (*Suc n*) $=$ (*if* (*Suc n*) $\in N$ *then S n* $\cup$ *enum-basis* (*Suc n*) *else S n*)
by (*simp-all add*: *S-def*)
have *incseq S* **by** (*rule incseq-SucI*) *auto*
moreover
have $(\bigcup j.\ S\ j) = X$ **unfolding** *N*
proof *safe*
fix *x n* **assume** $n \in N$ $x \in$ *enum-basis n*
hence $x \in S\ n$ **by** (*cases n*) *auto*
thus $x \in (\bigcup j.\ S\ j)$ **by** *auto*
next
fix *x j*
assume $x \in S\ j$
thus $x \in$ *UNION N enum-basis* **by** (*induct j*) (*auto split*: *split-if-asm*)
qed
moreover have *range S* $\subseteq$ *range enum-basis*
proof *safe*
fix *j* **show** $S\ j \in$ *range enum-basis* **by** (*induct j*) *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *open-incseqE*:
assumes *open X*
obtains *S* **where** *incseq S* $(\bigcup j.\ S\ j) = X$ *range S* $\subseteq$ *range enum-basis*
using *open-imp-Union-of-incseq assms* **by** *atomize-elim*

end

lemma *borel-eq-sigma-enum-basis*:

sets borel = *sigma-sets* (*space borel*) (*range enum-basis*)
apply (*simp add*: *borel-def*)
proof (*intro sigma-sets-eqI*, *safe*)
fix x::$'a$ *set* **assume** *open* x
from *open-enumerable-basisE*[*OF this*] **guess** N **.**
hence x: $x = (\bigcup n.$ *if* $n \in N$ *then enum-basis n else* $\{\})$ **by** (*auto split*: *split-if-asm*)
also have ... $\in$ *sigma-sets UNIV* (*range enum-basis*) **by** (*rule Union*) *auto*
finally show $x \in$ *sigma-sets UNIV* (*range enum-basis*) **.**
next
fix n
have *open* (*enum-basis n*) **by** (*rule open-enum-basis*) *simp*
thus *enum-basis n* $\in$ *sigma-sets UNIV* (*Collect open*) **by** *auto*
qed

lemma *countable-dense-set*:
shows $\exists\, x$::*nat* $\Rightarrow$ -. $\forall\,(y$::$'a$::*enumerable-basis set*). *open* $y \longrightarrow y \neq \{\} \longrightarrow (\exists\, n.$ $x\ n \in y)$
proof −
def $x \equiv \lambda n.$ (*SOME* x::$'a$. $x \in$ *enum-basis n*)
have x: $\bigwedge n.$ *enum-basis n* $\neq (\{\}$::$'a$ *set*) $\Longrightarrow x\ n \in$ *enum-basis n* **unfolding** *x-def*
by (*rule someI-ex*) *auto*
have $\forall\, y.$ *open* $y \longrightarrow y \neq \{\} \longrightarrow (\exists\, n.\ x\ n \in y)$
proof (*intro allI impI*)
fix y::$'a$ *set* **assume** *open* y $y \neq \{\}$
from *open-enumerable-basisE*[*OF* ‹*open y*›] **guess** N **. note** N = *this*
obtain n **where** n: $n \in N$ *enum-basis n* $\neq (\{\}$::$'a$ *set*)
proof (*atomize-elim*, *rule ccontr*, *clarsimp*)
assume $\forall\, n.\ n \in N \longrightarrow$ *enum-basis n* $= (\{\}$::$'a$ *set*)
hence $(\bigcup n{\in}N.$ *enum-basis n*$) = (\bigcup n{\in}N.\ \{\}$::$'a$ *set*)
by (*intro UN-cong*) *auto*
hence $y = \{\}$ **unfolding** N **by** *simp*
with ‹$y \neq \{\}$› **show** *False* **by** *auto*
qed
with $x\ N\ n$ **have** $x\ n \in y$ **by** *auto*
thus $\exists\, n.\ x\ n \in y$ **..**
qed
thus *?thesis* **by** *blast*
qed

lemma *countable-dense-setE*:
obtains x :: *nat* $\Rightarrow$ -
where $\bigwedge(y$::$'a$::*enumerable-basis set*). *open* $y \Longrightarrow y \neq \{\} \Longrightarrow \exists\, n.\ x\ n \in y$
using *countable-dense-set* **by** *blast*

2.5 Polish Spaces

Textbooks define Polish spaces as completely metrizable. We assume the topology to be complete for a given metric.

```
class polish-space = complete-space + enumerable-basis
```

TODO: Rules in *Topology-Euclidean-Space* should be proved in the *ordered-euclidean-space* locale! Then we can use subclass instead of instance.

```
instance ordered-euclidean-space ⊆ polish-space
proof
  def to-cube ≡ λ(a, b). {Chi (real-of-rat ∘ op ! a)<..<Chi (real-of-rat ∘ op !
b)}::'a set
  def enum ≡ λn. (to-cube (from-nat n)::'a set)
  have Ball (range enum) open unfolding enum-def
  proof safe
    fix n show open (to-cube (from-nat n))
      by (cases from-nat n::rat list × rat list)
        (simp add: open-interval to-cube-def)
  qed
  moreover have (∀ x. open x ⟶ (∃ B'⊆range enum. ⋃ B' = x))
  proof safe
    fix x::'a set assume open x
    def lists ≡ {(a, b) |a b. to-cube (a, b) ⊆ x}
    from open-UNION[OF ‹open x›]
    have ⋃(to-cube ` lists) = x unfolding lists-def to-cube-def
     by simp
    moreover have to-cube ` lists ⊆ range enum
    proof
      fix x assume x ∈ to-cube ` lists
      then obtain l where l ∈ lists x = to-cube l by auto
      hence x = enum (to-nat l) by (simp add: to-cube-def enum-def)
      thus x ∈ range enum by simp
    qed
    ultimately
    show ∃ B'⊆range enum. ⋃ B' = x by blast
  qed
  ultimately
  show ∃ f::nat⇒'a set. topological-basis (range f) unfolding topological-basis-def
by blast
qed

instantiation nat::topological-space
begin

definition open-nat::nat set ⇒ bool
  where open-nat s = True

instance proof qed (auto simp: open-nat-def)
end

instantiation nat::metric-space
begin
```

definition *dist-nat*::*nat* ⇒ *nat* ⇒ *real*
  where *dist-nat n m* = (*if n* = *m then 0 else 1*)

instance proof qed (*auto simp*: *open-nat-def dist-nat-def intro*: *exI*[**where** *x*=*1*])
end

instance *nat*::*complete-space*
proof
  fix *X*::*nat*⇒*nat* **assume** *Cauchy X*
  hence ∃ *n*. ∀ *m*≥*n*. *X m* = *X n*
   by (*force simp*: *dist-nat-def Cauchy-def split*: *split-if-asm dest*:*spec*[**where** *x*=*1*])
  then guess *n* **..**
  thus *convergent X*
    apply (*intro convergentI*[**where** *L*=*X n*] *tendstoI*)
    unfolding *eventually-sequentially dist-nat-def*
    apply (*intro exI*[**where** *x*=*n*])
    apply (*intro allI*)
    apply (*drule-tac x*=*na* **in** *spec*)
    apply *simp*
    done
qed

instance *nat*::*polish-space*
proof
  have *topological-basis* (*range* (λ*n*::*nat*. {*n*}))
    by (*intro topological-basisI*) (*auto simp*: *open-nat-def*)
  thus ∃ *f*::*nat*⇒*nat set*. *topological-basis* (*range f*) **by** *blast*
qed

2.6 Regularity of Measures

lemma *ereal-approx-SUP*:
  fixes *x*::*ereal*
  assumes *A-notempty*: *A* ≠ {}
  assumes *f-bound*: ⋀*i*. *i* ∈ *A* ⟹ *f i* ≤ *x*
  assumes *f-fin*: ⋀*i*. *i* ∈ *A* ⟹ *f i* ≠ ∞
  assumes *f-nonneg*: ⋀*i*. *0* ≤ *f i*
  assumes *approx*: ⋀*e*. (*e*::*real*) > *0* ⟹ ∃ *i* ∈ *A*. *x* ≤ *f i* + *e*
  shows *x* = (*SUP i* : *A*. *f i*)
proof (*subst eq-commute*, *rule ereal-SUPI*)
  show ⋀*i*. *i* ∈ *A* ⟹ *f i* ≤ *x* **using** *f-bound* **by** *simp*
next
  fix *y* :: *ereal* **assume** *f-le-y*: (⋀*i*::'*a*. *i* ∈ *A* ⟹ *f i* ≤ *y*)
  with *A-notempty f-nonneg* **have** *y* ≥ *0* **by** *auto* (*metis order-trans*)
  show *x* ≤ *y*
  proof (*rule ccontr*)
    assume ¬ *x* ≤ *y* **hence** *x* > *y* **by** *simp*
    hence *y-fin*: |*y*| ≠ ∞ **using** ‹*y* ≥ *0*› **by** *auto*
    have *x-fin*: |*x*| ≠ ∞ **using** ‹*x* > *y*› *f-fin approx*[**where** *e* = *1*] **by** *auto*

def $e \equiv$ *real* $((x - y) / 2)$
have e: $x > y + e$ $e > 0$ **using** ‹$x > y$› *y-fin x-fin* **by** (*auto simp*: *e-def field-simps*)
note $e(1)$
also from *approx*[*OF* ‹$e > 0$›] **obtain** i **where** i: $i \in A$ $x \leq f\ i + e$ **by** *blast*
note $i(2)$
finally have $y < f\ i$ **using** *y-fin f-fin* **by** (*metis add-right-mono linorder-not-le*)
moreover have $f\ i \leq y$ **by** (*rule f-le-y*) *fact*
ultimately show *False* **by** *simp*
qed
qed

lemma *ereal-approx-INF*:
fixes x::*ereal*
assumes *A-notempty*: $A \neq \{\}$
assumes *f-bound*: $\bigwedge i.\ i \in A \Longrightarrow x \leq f\ i$
assumes *f-fin*: $\bigwedge i.\ i \in A \Longrightarrow f\ i \neq \infty$
assumes *f-nonneg*: $\bigwedge i.\ 0 \leq f\ i$
assumes *approx*: $\bigwedge e.\ (e{::}real) > 0 \Longrightarrow \exists\, i \in A.\ f\ i \leq x + e$
shows $x = (INF\ i : A.\ f\ i)$
proof (*subst eq-commute*, *rule ereal-INFI*)
show $\bigwedge i.\ i \in A \Longrightarrow x \leq f\ i$ **using** *f-bound* **by** *simp*
next
fix y :: *ereal* **assume** *f-le-y*: $(\bigwedge i{::}'a.\ i \in A \Longrightarrow y \leq f\ i)$
with *A-notempty f-fin* **have** $y \neq \infty$ **by** *force*
show $y \leq x$
proof (*rule ccontr*)
assume $\neg\ y \leq x$ **hence** $y > x$ **by** *simp* **hence** $y \neq -\infty$ **by** *auto*
hence *y-fin*: $|y| \neq \infty$ **using** ‹$y \neq \infty$› **by** *auto*
have *x-fin*: $|x| \neq \infty$ **using** ‹$y > x$› *f-fin f-nonneg approx*[**where** $e = 1$] *A-notempty*
apply *auto* **by** (*metis ereal-infty-less-eq*(2) *f-le-y*)
def $e \equiv$ *real* $((y - x) / 2)$
have e: $y > x + e$ $e > 0$ **using** ‹$y > x$› *y-fin x-fin* **by** (*auto simp*: *e-def field-simps*)
from *approx*[*OF* ‹$e > 0$›] **obtain** i **where** i: $i \in A$ $x + e \geq f\ i$ **by** *blast*
note $i(2)$
also note $e(1)$
finally have $y > f\ i$ **.**
moreover have $y \leq f\ i$ **by** (*rule f-le-y*) *fact*
ultimately show *False* **by** *simp*
qed
qed

lemma *INF-approx-ereal*:
fixes x::*ereal* **and** e::*real*
assumes $e > 0$
assumes *INF*: $x = (INF\ i : A.\ f\ i)$
assumes $|x| \neq \infty$

shows $\exists\, i \in A.\ f\ i < x + e$
proof (*rule ccontr*, *clarsimp*)
assume $\forall\, i{\in}A.\ \neg\ f\ i < x + e$
moreover
from *INF* **have** $\bigwedge y.\ (\bigwedge i.\ i \in A \Longrightarrow y \le f\ i) \Longrightarrow y \le x$ **by** (*auto intro*: *INF-greatest*)
ultimately
have (*INF i* : *A*. *f i*) = $x + e$ **using** ‹$e > 0$›
by (*intro ereal-INFI*)
(*force*, *metis add.comm-neutral add-left-mono ereal-less*(*1*)
linorder-not-le not-less-iff-gr-or-eq)
thus *False* **using** *assms* **by** *auto*
qed

lemma *SUP-approx-ereal*:
fixes *x*::*ereal* **and** *e*::*real*
assumes $e > 0$
assumes *SUP*: $x = (SUP\ i : A.\ f\ i)$
assumes $|x| \neq \infty$
shows $\exists\, i \in A.\ x \le f\ i + e$
proof (*rule ccontr*, *clarsimp*)
assume $\forall\, i{\in}A.\ \neg\ x \le f\ i + e$
moreover
from *SUP* **have** $\bigwedge y.\ (\bigwedge i.\ i \in A \Longrightarrow f\ i \le y) \Longrightarrow y \ge x$ **by** (*auto intro*: *SUP-least*)
ultimately
have (*SUP i* : *A*. *f i*) = $x - e$ **using** ‹$e > 0$› ‹$|x| \neq \infty$›
by (*intro ereal-SUPI*)
(*metis PInfty-neq-ereal*(*2*) *abs-ereal.simps*(*1*) *ereal-minus-le linorder-linear*,
metis ereal-between(*1*) *ereal-less*(*2*) *less-eq-ereal-def order-trans*)
thus *False* **using** *assms* **by** *auto*
qed

lemma
fixes *M*::*'a*::*polish-space measure*
assumes *sb*: *sets M* = *sets borel*
assumes *emeasure M* (*space M*) $\neq \infty$
assumes $B \in$ *sets borel*
shows *inner-regular*: *emeasure M B* =
(*SUP K* : $\{K.\ K \subseteq B \wedge$ *compact K*$\}$. *emeasure M K*) (**is** *?inner B*)
and *outer-regular*: *emeasure M B* =
(*INF U* : $\{U.\ B \subseteq U \wedge$ *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*: $\bigwedge A.\ A \in$ *sets M* $\Longrightarrow$
($\bigwedge e.\ e > 0 \Longrightarrow \exists\, K.\ K \subseteq A \wedge$ *compact K* $\wedge$ *emeasure M A* $\le$ *emeasure M K* + *ereal e*) $\Longrightarrow$ *?inner A*

by (*rule ereal-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* +
ereal e) ⟹ *?outer A*
by (*rule ereal-approx-INF*)
(*force intro*!: *emeasure-mono simp*: *emeasure-eq-measure sb*)+
from *countable-dense-setE* **guess** *x*::*nat* ⇒ *'a* **. note** *x* = *this*
{
fix *r*::*real* **assume** *r* > *0* **hence** ⋀*y*. *open* (*ball y r*) ⋀*y*. *ball y r* ≠ {} **by** *auto*
with *x*[*OF this*]
have *x*: *space M* = (⋃*n*. *cball* (*x n*) *r*)
by (*auto simp add*: *sU*) (*metis dist-commute order-less-imp-le*)
have (λ*k*. *emeasure M* (⋃*n*∈{*0*..*k*}. *cball* (*x n*) *r*)) −−−−> *M* (⋃*k*. (⋃*n*∈{*0*..*k*}.
cball (*x n*) *r*))
by (*rule Lim-emeasure-incseq*)
(*auto intro*!: *borel-closed bexI simp*: *closed-cball incseq-def Us sb*)
also have (⋃*k*. (⋃*n*∈{*0*..*k*}. *cball* (*x n*) *r*)) = *space M*
unfolding *x* **by** *force*
finally have (λ*k*. *M* (⋃*n*∈{*0*..*k*}. *cball* (*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 intro*: *mult-pos-pos*)
from *M-space*[*OF* ‹*1*/*n*>*0*›]
have (λ*k*. *measure M* (⋃*i*∈{*0*..*k*}. *cball* (*x i*) (*1*/*real n*))) −−−−> *measure*
M (*space M*)
unfolding *emeasure-eq-measure* **by** *simp*
from *metric-LIMSEQ-D*[*OF this* ‹*0* < *e* ∗ *2 powr* −*n*›]
obtain *k* **where** *dist* (*measure M* (⋃*i*∈{*0*..*k*}. *cball* (*x i*) (*1*/*real n*))) (*measure*
M (*space M*)) <
e ∗ *2 powr* −*n*
by *auto*
hence *measure M* (⋃*i*∈{*0*..*k*}. *cball* (*x i*) (*1*/*real n*)) ≥
measure M (*space M*) − *e* ∗ *2 powr* −*real n*
by (*auto simp*: *dist-real-def*)
hence ∃ *k*. *measure M* (⋃*i*∈{*0*..*k*}. *cball* (*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 (⋃*i*∈{*0*..*k*}. *cball* (*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* (⋃*i*∈{*0*..*k e n*}. *cball* (*x i*) (*1* / *n*))
apply *atomize-elim* **unfolding** *bchoice-iff* **.**
hence *k*: ⋀*e n*. *e* > *0* ⟹ *n* > *0* ⟹ *measure M* (*space M*) − *e* ∗ *2 powr* − *n*
≤ *measure M* (⋃*i*∈{*0*..*k e n*}. *cball* (*x i*) (*1* / *n*))

unfolding *Ball-def* **by** *blast*
have *approx-space*:
$\bigwedge e.\ e > 0 \Longrightarrow$
$\exists K \in \{K.\ K \subseteq space\ M \wedge compact\ K\}.\ emeasure\ M\ (space\ M) \leq emeasure\ M\ K + ereal\ e$
(**is** $\bigwedge e.\ \text{-} \Longrightarrow\ ?thesis\ e$)
proof −
fix $e :: real$ **assume** $e > 0$
def $B \equiv \lambda n.\ \bigcup i\in\{0..k\ e\ (Suc\ n)\}.\ cball\ (x\ i)\ (1\ /\ Suc\ n)$
have $\bigwedge n.\ closed\ (B\ n)$ **by** (*auto simp*: *B-def closed-cball*)
hence [*simp*]: $\bigwedge n.\ B\ n \in sets\ M$ **by** (*simp add*: *sb*)
from *k*[*OF* ‹$e > 0$› *zero-less-Suc*]
have $\bigwedge n.\ measure\ M\ (space\ M) - measure\ M\ (B\ n) \leq e * 2\ powr - real\ (Suc\ n)$
by (*simp add*: *algebra-simps B-def finite-measure-compl*)
hence *B-compl-le*: $\bigwedge n::nat.\ measure\ M\ (space\ M - B\ n) \leq e * 2\ powr - real\ (Suc\ n)$
by (*simp add*: *finite-measure-compl*)
def $K \equiv \bigcap n.\ B\ n$
from ‹*closed* (*B* -)› **have** *closed K* **by** (*auto simp*: *K-def*)
hence [*simp*]: $K \in sets\ M$ **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 $\ldots = emeasure\ M\ (\bigcup n.\ space\ M - B\ n)$ **by** (*auto simp*: *K-def emeasure-eq-measure*)
also have $\ldots \leq (\sum n.\ emeasure\ M\ (space\ M - B\ n))$
by (*rule emeasure-subadditive-countably*) (*auto simp*: *summable-def*)
also have $\ldots \leq (\sum n.\ ereal\ (e*2\ powr - real\ (Suc\ n)))$
using *B-compl-le* **by** (*intro suminf-le-pos*) (*simp-all add*: *measure-nonneg emeasure-eq-measure*)
also have $\ldots \leq (\sum n.\ ereal\ (e * (1\ /\ 2)\ \hat{}\ Suc\ n))$
by (*simp add*: *powr-minus inverse-eq-divide powr-realpow field-simps power-divide*)
also have $\ldots = (\sum n.\ ereal\ e * ((1\ /\ 2)\ \hat{}\ Suc\ n))$
unfolding *times-ereal.simps*[*symmetric*] *ereal-power*[*symmetric*] *one-ereal-def numeral-eq-ereal*
by *simp*
also have $\ldots = ereal\ e * (\sum n.\ ((1\ /\ 2)\ \hat{}\ Suc\ n))$
by (*rule suminf-cmult-ereal*) (*auto simp*: ‹$0 < e$› *less-imp-le*)
also have $\ldots = e$ **unfolding** *suminf-half-series-ereal* **by** *simp*
finally have $measure\ M\ (space\ M) \leq measure\ M\ K + e$ **by** *simp*
hence $emeasure\ M\ (space\ M) \leq emeasure\ M\ K + e$ **by** (*simp add*: *emeasure-eq-measure*)
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 *pos-approach-nat*[*OF this*] **guess** *n* **. note** $n = this$
let $?k = x\ `\ \{0..k\ e\ (Suc\ n)\}$
have *finite ?k* **by** *simp*

```
      moreover have K ⊆ ⋃(λx. ball x e') ` ?k unfolding K-def B-def using n
by force
      ultimately show ∃k. finite k ∧ K ⊆ ⋃(λx. ball x e') ` k by blast
    qed
    ultimately
    show ?thesis e by (auto simp: sU)
  qed
  have closed-in-D: ⋀A. closed A ⟹ ?inner A ∧ ?outer A
  proof
   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)
    show ?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 M A − M (A ∩ K) = M (A ∪ K) − M K by (simp add: emeasure-eq-measure
measure-union)
      also have ... ≤ M (space M) − M K
        by (simp add: emeasure-eq-measure sU sb finite-measure-mono)
      also have ... ≤ e using K by (simp add: emeasure-eq-measure)
      finally have emeasure M A ≤ emeasure M (A ∩ K) + ereal e by (simp add:
emeasure-eq-measure)
      moreover have A ∩ K ⊆ A compact (A ∩ K) using ‹closed A› ‹compact K›
by auto
      ultimately show ∃K ⊆ A. compact K ∧ emeasure M A ≤ emeasure M K
+ ereal e
        by blast
    qed simp
    show ?outer A
    proof cases
      assume A ≠ {}
      let ?G = λd. {x. infdist x A < d}
      {
        fix d
        have ?G d = (λx. infdist x A) −` {..<d} by auto
        also have open ... using continuous-infdist[OF ‹A ≠ {}›]
          by (intro continuous-open-vimage) auto
        finally have open (?G d) .
      } note open-G = this
      from in-closed-iff-infdist-zero[OF ‹closed A› ‹A ≠ {}›]
      have A = {x. infdist x A = 0} by auto
      also have ... = (⋂i. ?G (1/real (Suc i)))
      proof (auto, rule ccontr)
        fix x
        assume infdist x A ≠ 0
         hence pos: infdist x A > 0 using infdist-nonneg[OF ‹A ≠ {}›, of x] by
```

simp
from *pos-approach-nat*[*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 $M\ \ldots = (INF\ n.\ emeasure\ M\ (?G\ (1\ /\ real\ (Suc\ n))))$
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*)
next
show $decseq\ (\lambda i.\ \{x.\ infdist\ x\ A < 1\ /\ real\ (Suc\ i)\})$
by (*auto intro*: *less-trans intro*!: *divide-strict-left-mono mult-pos-pos*
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 $\ldots \geq (INF\ U{:}\{U.\ A \subseteq U \wedge open\ U\}.\ emeasure\ M\ U)$
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 $M\ (?G\ (1\ /\ real\ (Suc\ m))) \leq M\ (?G\ (1\ /\ real\ (Suc\ m)))$ **by** *simp*
ultimately show $\exists\, U{\in}\{U.\ A \subseteq U \wedge open\ U\}.$
$emeasure\ M\ U \leq emeasure\ M\ \{x.\ infdist\ x\ A < 1\ /\ real\ (Suc\ m)\}$
by *blast*
qed
moreover
have $emeasure\ M\ A \leq (INF\ U{:}\{U.\ A \subseteq U \wedge open\ U\}.\ emeasure\ M\ U)$
by (*rule INF-greatest*) (*auto intro*!: *emeasure-mono simp*: *sb*)
ultimately show *?thesis* **by** *simp*
qed (*auto intro*!: *ereal-INFI*)
qed
let $?D = \{B \in sets\ M.\ ?inner\ B \wedge ?outer\ B\}$
interpret *dynkin*: *dynkin-system space M ?D*
proof (*rule dynkin-systemI*)
have $\{U{::}'a\ set.\ space\ M \subseteq U \wedge open\ U\} = \{space\ M\}$ **by** (*auto simp add*: *sU*)
hence *?outer* (*space M*) **by** (*simp add*: *min-def INF-def*)
moreover
have *?inner* (*space M*)
proof (*rule ereal-approx-SUP*)
fix *e*::*real* **assume** $0 < e$
thus $\exists\, K{\in}\{K.\ K \subseteq space\ M \wedge compact\ K\}.\ emeasure\ M\ (space\ M) \leq$

emeasure M K + *ereal e*
by (*rule approx-space*)
qed (*auto intro*: *emeasure-mono simp*: *sU sb intro*!: *exI*[**where** $x=\{\}$])
ultimately show *space M* ∈ *?D* **by** (*simp add*: *sU sb*)
next
fix *B* **assume** *B* ∈ *?D* **thus** *B* ⊆ *space M* **by** (*simp add*: *sU*)
from ‹*B* ∈ *?D*› **have** [*simp*]: *B* ∈ *sets M* **and** *?inner B ?outer B* **by** *auto*
hence *inner*: *emeasure M B* = (*SUP K*:{*K*. *K* ⊆ *B* ∧ *compact K*}. *emeasure M K*)
and *outer*: *emeasure M B* = (*INF U*:{*U*. *B* ⊆ *U* ∧ *open U*}. *emeasure M U*) **by** *auto*
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*)
unfolding *inner* **by** (*subst INFI-ereal-cminus*) *force*+
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*)
by (*subst INF-image*[*of* λ*u*. *space M* − *u*, *symmetric*])
(*rule INF-cong*, *auto simp add*: *sU intro*!: *INF-cong*)
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* ⊆ *U* ∧ *open U*}. *emeasure M U*) ≥ *emeasure M* (*space M* − *B*)
by (*auto simp*: *sb sU intro*!: *INF-greatest emeasure-mono*)
ultimately have *?outer* (*space M* − *B*) **by** *simp*
moreover
{
have *M* (*space M* − *B*) = *M* (*space M*) − *emeasure M B* **by** (*auto simp*: *emeasure-compl*)
also have ... = (*SUP U*: {*U*. *B* ⊆ *U* ∧ *open U*}. *M* (*space M*) − *M U*)
unfolding *outer* **by** (*subst SUPR-ereal-cminus*) *auto*
also have ... = (*SUP U*:{*U*. *B* ⊆ *U* ∧ *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* ⊆ *space M* − *B* ∧ *closed K*}. *emeasure M K*)
by (*subst SUP-image*[*of* λ*u*. *space M* − *u*, *symmetric*])
(*rule SUP-cong*, *auto simp*: *sU*)
also have ... = (*SUP K*:{*K*. *K* ⊆ *space M* − *B* ∧ *compact K*}. *emeasure M K*)
proof (*safe intro*!: *antisym SUP-least*)
fix *K* **assume** *closed K K* ⊆ *space M* − *B*
from *closed-in-D*[*OF* ‹*closed K*›]
have *K-inner*: *emeasure M K* = (*SUP K*:{*Ka*. *Ka* ⊆ *K* ∧ *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 have *?inner* (*space M* $-$ *B*) **.**
} **hence** *?inner* (*space M* $-$ *B*) **.**
ultimately show *space M* $-$ *B* $\in$ *?D* **by** *auto*
next
fix *D* :: *nat* $\Rightarrow$ -
assume *range D* $\subseteq$ *?D* **hence** *range D* $\subseteq$ *sets M* **by** *auto*
moreover assume *disjoint-family D*
ultimately have *M*[*symmetric*]: ($\sum$ *i*. *M* (*D i*)) = *M* ($\bigcup$ *i*. *D i*) **by** (*rule suminf-emeasure*)
also have (λ*n*. $\sum$ *i*$\in${*0*..<*n*}. *M* (*D i*)) $----$> ($\sum$ *i*. *M* (*D i*))
by (*intro summable-sumr-LIMSEQ-suminf summable-ereal-pos emeasure-nonneg*)
finally have *measure-LIMSEQ*: (λ*n*. $\sum$ *i* = *0*..<*n*. *measure M* (*D i*)) $----$> *measure M* ($\bigcup$ *i*. *D i*)
by (*simp add*: *emeasure-eq-measure*)
have ($\bigcup$ *i*. *D i*) $\in$ *sets M* **using** ‹*range D* $\subseteq$ *sets M*› **by** *auto*
moreover
hence *?inner* ($\bigcup$ *i*. *D i*)
proof (*rule approx-inner*)
fix *e*::*real* **assume** *e* > *0*
with *measure-LIMSEQ*
have $\exists$ *no*. $\forall$ *n*$\geq$*no*. |($\sum$ *i* = *0*..<*n*. *measure M* (*D i*)) $-$*measure M* ($\bigcup$ *x*. *D x*)| < *e*/*2*
by (*auto simp*: *LIMSEQ-def dist-real-def simp del*: *less-divide-eq-numeral1*)
hence $\exists$ *n0*. |($\sum$ *i* = *0*..<*n0*. *measure M* (*D i*)) $-$ *measure M* ($\bigcup$ *x*. *D x*)| < *e*/*2* **by** *auto*
then obtain *n0* **where** *n0*: |($\sum$ *i* = *0*..<*n0*. *measure M* (*D i*)) $-$ *measure M* ($\bigcup$ *i*. *D i*)| < *e*/*2*
unfolding *choice-iff* **by** *blast*
have *ereal* ($\sum$ *i* = *0*..<*n0*. *measure M* (*D i*)) = ($\sum$ *i* = *0*..<*n0*. *M* (*D i*))
by (*auto simp add*: *emeasure-eq-measure*)
also have ... = ($\sum$ *i*<*n0*. *M* (*D i*)) **by** (*rule setsum-cong*) *auto*
also have ... $\leq$ ($\sum$ *i*. *M* (*D i*)) **by** (*rule suminf-upper*) (*auto simp*: *emeasure-nonneg*)
also have ... = *M* ($\bigcup$ *i*. *D i*) **by** (*simp add*: *M*)
also have ... = *measure M* ($\bigcup$ *i*. *D i*) **by** (*simp add*: *emeasure-eq-measure*)
finally have *n0*: *measure M* ($\bigcup$ *i*. *D i*) $-$ ($\sum$ *i* = *0*..<*n0*. *measure M* (*D i*)) < *e*/*2*
using *n0* **by** *auto*
have $\forall$ *i*. $\exists$ *K*. *K* $\subseteq$ *D i* $\wedge$ *compact K* $\wedge$ *emeasure M* (*D i*) $\leq$ *emeasure M K* + *e*/(*2*$*$*Suc n0*)
proof
fix *i*
from ‹*0* < *e*› **have** *0* < *e*/(*2*$*$*Suc n0*) **by** (*auto intro*: *divide-pos-pos*)
have *emeasure M* (*D i*) = (*SUP K*:{*K*. *K* $\subseteq$ (*D i*) $\wedge$ *compact K*}. *emeasure*

M K)
using ‹*range D* ⊆ *?D*› by *blast*
from *SUP-approx-ereal*[*OF* ‹*0* < *e*/(*2*∗*Suc n0*)› *this*]
show $\exists K.\ K \subseteq D\ i \wedge compact\ K \wedge emeasure\ M\ (D\ i) \leq emeasure\ M\ K + e/(2{*}Suc\ n0)$
by (*auto simp*: *emeasure-eq-measure*)
qed
then obtain *K* where *K*: $\bigwedge i.\ K\ i \subseteq D\ i$ $\bigwedge i.\ compact\ (K\ i)$
$\bigwedge i.\ emeasure\ M\ (D\ i) \leq emeasure\ M\ (K\ i) + e/(2{*}Suc\ n0)$
unfolding *choice-iff* by *blast*
let $?K = \bigcup i{\in}\{0..{<}n0\}.\ K\ i$
have *disjoint-family-on K* {*0*..<*n0*} using *K* ‹*disjoint-family D*›
unfolding *disjoint-family-on-def* by *blast*
hence *mK*: $measure\ M\ ?K = (\sum i = 0..{<}n0.\ measure\ M\ (K\ i))$ using *K*
by (*intro finite-measure-finite-Union*) (*auto simp*: *sb compact-imp-closed*)
have $measure\ M\ (\bigcup i.\ D\ i) < (\sum i = 0..{<}n0.\ measure\ M\ (D\ i)) + e/2$ using *n0* by *simp*
also have $(\sum i = 0..{<}n0.\ measure\ M\ (D\ i)) \leq (\sum i = 0..{<}n0.\ measure\ M\ (K\ i) + e/(2{*}Suc\ n0))$
using *K* by (*auto intro*: *setsum-mono simp*: *emeasure-eq-measure*)
also have $\ldots = (\sum i = 0..{<}n0.\ measure\ M\ (K\ i)) + (\sum i = 0..{<}n0.\ e/(2{*}Suc\ n0))$
by (*simp add*: *setsum.distrib*)
also have $\ldots \leq (\sum i = 0..{<}n0.\ measure\ M\ (K\ i)) + e\ /\ 2$ using ‹*0* < *e*›
by (*auto simp*: *real-of-nat-def*[*symmetric*] *field-simps intro*!: *mult-left-mono*)
finally
have $measure\ M\ (\bigcup i.\ D\ i) < (\sum i = 0..{<}n0.\ measure\ M\ (K\ i)) + e\ /\ 2 + e\ /\ 2$
by *auto*
hence $M\ (\bigcup i.\ D\ i) < M\ ?K + e$ by (*auto simp*: *mK emeasure-eq-measure*)
moreover
have $?K \subseteq (\bigcup i.\ D\ i)$ using *K* by *auto*
moreover
have *compact ?K* using *K* by *auto*
ultimately
have $?K{\subseteq}(\bigcup i.\ D\ i) \wedge compact\ ?K \wedge emeasure\ M\ (\bigcup i.\ D\ i) \leq emeasure\ M\ ?K + ereal\ e$ by *simp*
thus $\exists K{\subseteq}\bigcup i.\ D\ i.\ compact\ K \wedge emeasure\ M\ (\bigcup i.\ D\ i) \leq emeasure\ M\ K + ereal\ e$ ..
qed
moreover have $?outer\ (\bigcup i.\ D\ i)$
proof (*rule approx-outer*[*OF* ‹$(\bigcup i.\ D\ i) \in sets\ M$›])
fix *e*::*real* assume $e > 0$
have $\forall i{::}nat.\ \exists U.\ D\ i \subseteq U \wedge open\ U \wedge e/(2\ powr\ Suc\ i) > emeasure\ M\ U - emeasure\ M\ (D\ i)$
proof
fix *i*::*nat*
from ‹*0* < *e*› have $0 < e/(2\ powr\ Suc\ i)$ by (*auto intro*: *divide-pos-pos*)
have $emeasure\ M\ (D\ i) = (INF\ U{:}\{U.\ (D\ i) \subseteq U \wedge open\ U\}.\ emeasure\ M$

U)
using ‹*range D* $\subseteq$ *?D*› **by** *blast*
from *INF-approx-ereal*[*OF* ‹$0 < e/(2$ *powr Suc i*)› *this*]
show $\exists U$. *D i* $\subseteq$ *U* $\wedge$ *open U* $\wedge$ $e/(2$ *powr Suc i*) $>$ *emeasure M U* $-$ *emeasure M* (*D i*)
by (*auto simp*: *emeasure-eq-measure*)
qed
then obtain *U* **where** *U*: $\bigwedge i$. *D i* $\subseteq$ *U i* $\bigwedge i$. *open* (*U i*)
$\bigwedge i$. $e/(2$ *powr Suc i*) $>$ *emeasure M* (*U i*) $-$ *emeasure M* (*D i*)
unfolding *choice-iff* **by** *blast*
let *?U* $= \bigcup i$. *U i*
have *M ?U* $-$ *M* ($\bigcup i$. *D i*) $=$ *M* (*?U* $-$ ($\bigcup i$. *D i*)) **using** *U* ‹($\bigcup i$. *D i*) $\in$ *sets M*›
by (*subst emeasure-Diff*) (*auto simp*: *sb*)
also have ... $\leq$ *M* ($\bigcup i$. *U i* $-$ *D i*) **using** *U* ‹*range D* $\subseteq$ *sets M*›
by (*intro emeasure-mono*) (*auto simp*: *sb intro*!: *countable-nat-UN Diff*)
also have ... $\leq$ ($\sum i$. *M* (*U i* $-$ *D i*)) **using** *U* ‹*range D* $\subseteq$ *sets M*›
by (*intro emeasure-subadditive-countably*) (*auto intro*!: *Diff simp*: *sb*)
also have ... $\leq$ ($\sum i$. *ereal* $e/(2$ *powr Suc i*)) **using** *U* ‹*range D* $\subseteq$ *sets M*›
by (*intro suminf-le-pos*, *subst emeasure-Diff*)
(*auto simp*: *emeasure-Diff emeasure-eq-measure sb measure-nonneg intro*: *less-imp-le*)
also have ... $\leq$ ($\sum n$. *ereal* ($e *$ (1 / 2) ^ *Suc n*))
by (*simp add*: *powr-minus inverse-eq-divide powr-realpow field-simps power-divide*)
also have ... $=$ ($\sum n$. *ereal* $e *$ ((1 / 2) ^ *Suc n*))
unfolding *times-ereal.simps*[*symmetric*] *ereal-power*[*symmetric*] *one-ereal-def numeral-eq-ereal*
by *simp*
also have ... $=$ *ereal* $e *$ ($\sum n$. ((1 / 2) ^ *Suc n*))
by (*rule suminf-cmult-ereal*) (*auto simp*: ‹$0 < e$› *less-imp-le*)
also have ... $= e$ **unfolding** *suminf-half-series-ereal* **by** *simp*
finally
have *emeasure M ?U* $\leq$ *emeasure M* ($\bigcup i$. *D i*) $+$ *ereal e* **by** (*simp add*: *emeasure-eq-measure*)
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*) $+$ *ereal 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*) $+$ *ereal e* **..**
qed
ultimately show ($\bigcup i$. *D i*) $\in$ *?D* **by** *safe*
qed
have *sets borel* $=$ *sigma-sets* (*space M*) (*Collect closed*) **by** (*simp add*: *borel-def-closed sU*)
also have ... $=$ *dynkin* (*space M*) (*Collect closed*)

proof (*rule sigma-eq-dynkin*)
show *Collect closed* $\subseteq$ *Pow* (*space M*) **using** *Sigma-Algebra.sets-into-space* **by** (*auto simp*: *sU*)
show *Int-stable* (*Collect closed*) **by** (*auto simp*: *Int-stable-def*)
qed
also have ... $\subseteq$ *?D* **using** *closed-in-D*
by (*intro dynkin.dynkin-subset*) (*auto simp add*: *compact-imp-closed sb*)
finally have *sets borel* $\subseteq$ *?D* **.**
moreover have *?D* $\subseteq$ *sets borel* **by** (*auto simp*: *sb*)
ultimately have *sets borel* = *?D* **by** *simp*
with *assms* **show** *?inner B* **and** *?outer B* **by** *auto*
qed

end

theory *Fin-Map*
imports *Auxiliarities Polish-Space*
begin

3 Finite Maps

typedef (**open**) ('*i*, '*a*) *finmap* ((- $\Rightarrow_F$ /-) [*22*, *21*] *21*) =
{(*I*::'*i set*, *f*::'*i* $\Rightarrow$ '*a*). *finite I* $\wedge$ *f* $\in$ *extensional I*} **by** *auto*
print-theorems

3.1 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 [*1000*] *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*)

3.2 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 $\exists\ to\text{-}nat :: 'a \Rightarrow_F 'b \Rightarrow nat.\ inj\ to\text{-}nat$
    by (*intro exI*[*of - to-nat* ∘ *?F*] *inj-comp*) *auto*
qed

3.3 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 $finmap\text{-}of\ i\ m = finmap\text{-}of\ j\ n \longleftrightarrow i = j \land restrict\ m\ i = restrict\ n\ i$
  using *assms*
  apply (*auto simp*: *finmap-eq-iff restrict-def*) **by** *metis*

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 (*finmap-of* K x)$_F$ = (*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

lemma *finmap-choice*:
assumes $*$: $\bigwedge i.\ i \in I \Longrightarrow \exists x.\ P\ i\ x$ **and** I: *finite* I
shows $\exists fm.\ domain\ fm = I \wedge (\forall i{\in}I.\ P\ i\ (fm\ i))$
proof −
have $\exists f.\ \forall i{\in}I.\ P\ i\ (f\ i)$
unfolding *bchoice-iff* [*symmetric*] **using** $*$ **by** *auto*
then guess f **..**
with I **show** *?thesis*
by (*intro exI*[*of - finmap-of I f*]) *auto*
qed

3.4 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* ((*3PI′* -:-./ -) *10*)

syntax (*xsymbols*)
-Pi′ :: [*pttrn*, *′a set*, *′b set*] => (*′a* => *′b*) *set* ((*3*Π*′* -∈-./ -) *10*)

syntax (*HTML* **output**)
-Pi′ :: [*pttrn*, *′a set*, *′b set*] => (*′a* => *′b*) *set* ((*3*Π*′* -∈-./ -) *10*)

translations
PI′ x:*A*. *B* == *CONST Pi′ A* (%*x*. *B*)

abbreviation
finmapset :: [*′a set*, *′b set*] => (*′a* $\Rightarrow_F$ *′b*) *set*
(**infixr** ~> *60*) **where**
A ~> *B* ≡ *Pi′ A* (%-. *B*)

notation (*xsymbols*)
finmapset (**infixr** ⇝ *60*)

3.4.1 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 *finmapsetI*: *domain f* = *A* $\Longrightarrow$ ($\bigwedge x.\ x \in A \Longrightarrow f\ x \in B$) $\Longrightarrow f \in A \rightsquigarrow B$
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 *funcset-mem*: $[|f \in A \rightsquigarrow B;\ x \in A|]$ ==> $f\ x \in B$
by (*simp add*: *Pi'-def*)

lemma *funcset-image*: $f \in A \rightsquigarrow B$ ==> f ‘ $A \subseteq B$
by *auto*

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

3.5 Metric Space of Finite Maps

instantiation *finmap* :: (*type*, *metric-space*) *metric-space*
begin

definition *dist-finmap* **where**
  $dist\ P\ Q = (\sum i{\in}domain\ P \cup domain\ Q.\ dist\ ((P)_F\ i)\ ((Q)_F\ i)) +$
    $card\ ((domain\ P - domain\ Q) \cup (domain\ Q - domain\ P))$

lemma *dist-finmap-extend*:
  assumes *finite X*
  shows $dist\ P\ Q = (\sum i{\in}domain\ P \cup domain\ Q \cup X.\ dist\ ((P)_F\ i)\ ((Q)_F\ i)) +$
    $card\ ((domain\ P - domain\ Q) \cup (domain\ Q - domain\ P))$
    unfolding *dist-finmap-def add-right-cancel*
    using *assms extensional-arb*[*of* $(P)_F$] *extensional-arb*[*of* $(Q)_F$ *domain Q*]
    by (*intro setsum-mono-zero-cong-left*) *auto*

definition *open-finmap* :: $('a \Rightarrow_F 'b)\ set \Rightarrow bool$ **where**
  $open\text{-}finmap\ S = (\forall x{\in}S.\ \exists e{>}0.\ \forall y.\ dist\ y\ x < e \longrightarrow y \in S)$

lemma *add-eq-zero-iff* [*simp*]:
  fixes *a b*::*real*
  assumes $a \geq 0\ b \geq 0$
  shows $a + b = 0 \longleftrightarrow a = 0 \wedge b = 0$
using *assms* **by** *auto*

lemma *dist-le-1-imp-domain-eq*:
  assumes $dist\ P\ Q < 1$
  shows $domain\ P = domain\ Q$
proof −
  have $0 \leq (\sum i{\in}domain\ P \cup domain\ Q.\ dist\ (P\ i)\ (Q\ i))$
    by (*simp add*: *setsum-nonneg*)
  with *assms* **have** $card\ (domain\ P - domain\ Q \cup (domain\ Q - domain\ P)) =$
0
    unfolding *dist-finmap-def* **by** *arith*
  thus $domain\ P = domain\ Q$ **by** *auto*
qed

lemma *dist-proj*:
  shows $dist\ ((x)_F\ i)\ ((y)_F\ i) \leq dist\ x\ y$
proof −
  have $dist\ (x\ i)\ (y\ i) = (\sum i{\in}\{i\}.\ dist\ (x\ i)\ (y\ i))$ **by** *simp*
  also have $\ldots \leq (\sum i{\in}domain\ x \cup domain\ y \cup \{i\}.\ dist\ (x\ i)\ (y\ i))$
    by (*intro setsum-mono2*) *auto*
  also have $\ldots \leq dist\ x\ y$ **by** (*simp add*: *dist-finmap-extend*[*of* $\{i\}$])
  finally show *?thesis* **by** *simp*
qed

lemma *open-Pi'I*:
  assumes *open-component*: $\bigwedge i.\ i \in I \Longrightarrow open\ (A\ i)$

```
  shows open (Pi' I A)
proof (subst open-finmap-def, safe)
  fix x assume x: x ∈ Pi' I A
  hence dim-x: domain x = I by (simp add: Pi'-def)
  hence [simp]: finite I unfolding dim-x[symmetric] by simp
  have ∃ ei. ∀ i∈I. 0 < ei i ∧ (∀ y. dist y (x i) < ei i ⟶ y ∈ A i)
  proof (safe intro!: bchoice)
    fix i assume i: i ∈ I
    moreover with open-component have open (A i) by simp
    moreover have x i ∈ A i using x i
      by (auto simp: proj-def)
    ultimately show ∃ e>0. ∀ y. dist y (x i) < e ⟶ y ∈ A i
      using x by (auto simp: open-dist Ball-def)
  qed
  then guess ei .. note ei = this
  def es ≡ ei ‘ I
  def e ≡ if es = {} then 0.5 else min 0.5 (Min es)
  from ei have e > 0 using x
    by (auto simp add: e-def es-def Pi'-def Ball-def)
  moreover have ∀ y. dist y x < e ⟶ y ∈ Pi' I A
  proof (intro allI impI)
    fix y
    assume dist y x < e
    also have ... < 1 by (auto simp: e-def)
    finally have domain y = domain x by (rule dist-le-1-imp-domain-eq)
    with dim-x have dims: domain y = domain x domain x = I by auto
    show y ∈ Pi' I A
    proof
      show domain y = I using dims by simp
    next
      fix i
      assume i ∈ I
      have dist (y i) (x i) ≤ dist y x using dims ‹i ∈ I›
        by (auto intro: dist-proj)
      also have ... < e using ‹dist y x < e› dims
        by (simp add: dist-finmap-def)
      also have e ≤ Min (ei ‘ I) using dims ‹i ∈ I›
        by (auto simp: e-def es-def)
      also have ... ≤ ei i using ‹i ∈ I› by (simp add: e-def)
      finally have dist (y i) (x i) < ei i .
      with ei ‹i ∈ I› show y i ∈ A  i by simp
    qed
  qed
  ultimately
  show ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ Pi' I A by blast
qed

instance
proof
```

fix S::('$a \Rightarrow_F$ 'b) *set*
show *open* S = ($\forall x \in S$. $\exists e>0$. $\forall y$. *dist* y x < e $\longrightarrow$ $y \in S$)
unfolding *open-finmap-def* **..**
next
fix P Q::'$a \Rightarrow_F$ 'b
show *dist* P Q = *0* $\longleftrightarrow$ $P = Q$
by (*auto simp*: *finmap-eq-iff dist-finmap-def setsum-nonneg setsum-nonneg-eq-0-iff*)
next
fix P Q R::'$a \Rightarrow_F$ 'b
let *?symdiff* = λa b. *domain* a − *domain* b ∪ (*domain* b − *domain* a)
def $E \equiv$ *domain* P ∪ *domain* Q ∪ *domain* R
hence *finite* E **by** (*simp add*: *E-def*)
have *card* (*?symdiff* P Q) ≤ *card* (*?symdiff* P R ∪ *?symdiff* Q R)
by (*auto intro*: *card-mono*)
also have ... ≤ *card* (*?symdiff* P R) + *card* (*?symdiff* Q R)
by (*subst card-Un-Int*) *auto*
finally have *dist* P Q ≤ ($\sum i \in E$. *dist* (P i) (R i) + *dist* (Q i) (R i)) +
real (*card* (*?symdiff* P R) + *card* (*?symdiff* Q R))
unfolding *dist-finmap-extend*[*OF* ‹*finite* E›]
by (*intro add-mono*) (*auto simp*: *E-def intro*: *setsum-mono dist-triangle-le*)
also have ... ≤ *dist* P R + *dist* Q R
unfolding *dist-finmap-extend*[*OF* ‹*finite* E›] **by** (*simp add*: *ac-simps E-def setsum-addf*[*symmetric*])
finally show *dist* P Q ≤ *dist* P R + *dist* Q R **by** *simp*
qed

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* $\{m.\ P\ (domain\ m)\}$
proof $-$
have $\{m.\ P\ (domain\ m)\} = -\ (\bigcup i \in -\ Collect\ P.\ \{m.\ domain\ m = i\})$ **by** *auto*
also have *closed* ...
proof (*rule*, *rule*, *rule*, *cases*)
fix $i{::}'a\ set$
assume *finite i*
hence $\{m.\ domain\ m = i\} = Pi'\ i\ (\lambda\text{-}.\ UNIV)$ **by** (*auto simp*: *Pi'-def*)
also have *open* ... **by** (*auto intro*: *open-Pi'I simp*: ‹*finite i*›)
finally show *open* $\{m.\ domain\ m = i\}$ **.**
next
fix $i{::}'a\ set$
assume ¬ *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 *continuous-proj*:
shows *continuous-on s* $(\lambda x.\ (x)_F\ i)$
unfolding *continuous-on-topological*
proof *safe*
fix *x B* **assume** $x \in s$ *open B* $x\ i \in B$
let $?A = Pi'\ (domain\ x)\ (\lambda j.\ if\ i = j\ then\ B\ else\ UNIV)$
have *open ?A* **using** ‹*open B*› **by** (*auto intro*: *open-Pi'I*)
moreover have $x \in\ ?A$ **using** ‹$x\ i \in B$› **by** *auto*
moreover have $(\forall y{\in}s.\ y \in\ ?A \longrightarrow y\ i \in B)$
proof (*cases*, *safe*)
fix *y* **assume** $y \in s$
assume $i \notin domain\ x$ **hence** $undefined \in B$ **using** ‹$x\ i \in B$›
by *simp*
moreover
assume $y \in\ ?A$ **hence** $domain\ y = domain\ x$ **by** (*simp add*: *Pi'-def*)
hence $y\ i = undefined$ **using** ‹$i \notin domain\ x$› **by** *simp*
ultimately
show $y\ i \in B$ **by** *simp*
qed *force*
ultimately
show $\exists A.\ open\ A \wedge x \in A \wedge (\forall y{\in}s.\ y \in A \longrightarrow y\ i \in B)$ **by** *blast*
qed

3.6 Complete Space of Finite Maps

lemma *tendsto-dist-zero*:
assumes $(\lambda i.\ dist\ (f\ i)\ g) \ {-}{-}{-}{-}{>}\ 0$
shows $f \ {-}{-}{-}{-}{>}\ g$
using *assms* **by** (*auto simp*: *tendsto-iff dist-real-def*)

lemma *tendsto-dist-zero′*:
  assumes $(\lambda i.\ dist\ (f\ i)\ g)$ ———> x
  assumes $0 = x$
  shows f ———> g
  using *assms tendsto-dist-zero* **by** *simp*

lemma *tendsto-finmap*:
  fixes $f::nat \Rightarrow ('i \Rightarrow_F ('a::metric\text{-}space))$
  assumes *ind-f*: $\bigwedge n.\ domain\ (f\ n) = domain\ g$
  assumes *proj-g*: $\bigwedge i.\ i \in domain\ g \Longrightarrow (\lambda n.\ (f\ n)\ i)$ ———> $g\ i$
  shows f ———> g
  apply (*rule tendsto-dist-zero′*)
  unfolding *dist-finmap-def assms*
  apply (*rule tendsto-intros proj-g* | *simp*)+
  done

instance *finmap* :: (*type*, *complete-space*) *complete-space*
proof
  fix $P::nat \Rightarrow 'a \Rightarrow_F 'b$
  assume *Cauchy P*
  then obtain *Nd* **where** *Nd*: $\bigwedge n.\ n \geq Nd \Longrightarrow dist\ (P\ n)\ (P\ Nd) < 1$
    by (*force simp*: *cauchy*)
  def $d \equiv domain\ (P\ Nd)$
  with *Nd* **have** *dim*: $\bigwedge n.\ n \geq Nd \Longrightarrow domain\ (P\ n) = d$ **using** *dist-le-1-imp-domain-eq* **by** *auto*
  have [*simp*]: *finite d* **unfolding** *d-def* **by** *simp*
  def $p \equiv \lambda i\ n.\ (P\ n)\ i$
  def $q \equiv \lambda i.\ lim\ (p\ i)$
  def $Q \equiv finmap\text{-}of\ d\ q$
  have *q*: $\bigwedge i.\ i \in d \Longrightarrow q\ i = Q\ i$ **by** (*auto simp add*: *Q-def Abs-finmap-inverse*)
  {
    fix *i* **assume** $i \in d$
    have *Cauchy* (*p i*) **unfolding** *cauchy p-def*
    proof *safe*
      fix *e::real* **assume** $0 < e$
      with ‹*Cauchy P*› **obtain** *N* **where** *N*: $\bigwedge n.\ n \geq N \Longrightarrow dist\ (P\ n)\ (P\ N) < min\ e\ 1$
        by (*force simp*: *cauchy min-def*)
      hence $\bigwedge n.\ n \geq N \Longrightarrow domain\ (P\ n) = domain\ (P\ N)$ **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 ... < *e* **using** *N*[*OF* ‹*N* ≤ *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* −−−−> *q i* **unfolding** *q-def convergent-def* **by** (*metis limI*)
} **note** *p* = *this*
have *P* −−−−> *Q*
proof (*rule metric-LIMSEQ-I*)
fix *e*::*real* **assume** *0* < *e*
def *e'* ≡ *min 1* (*e* / (*card d* + *1*))
hence *0* < *e'* **using** ‹*0* < *e*› **by** (*auto simp*: *e'-def intro*: *divide-pos-pos*)
have ∃ *ni*. ∀ *i*∈*d*. ∀ *n*≥*ni i*. *dist* (*p i n*) (*q i*) < *e'*
proof (*safe intro*!: *bchoice*)
fix *i* **assume** *i* ∈ *d*
from *p*[*OF* ‹*i* ∈ *d*›, *THEN metric-LIMSEQ-D*, *OF* ‹*0* < *e'*›]
show ∃ *no*. ∀ *n*≥*no*. *dist* (*p i n*) (*q i*) < *e'* **.**
qed then guess *ni* **..** **note** *ni* = *this*
def *N* ≡ *max Nd* (*Max* (*ni* ‘ *d*))
show ∃ *N*. ∀ *n*≥*N*. *dist* (*P n*) *Q* < *e*
proof (*safe intro*!: *exI*[**where** *x*=*N*])
fix *n* **assume** *N* ≤ *n*
hence *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*)
hence *dist* (*P n*) *Q* = (∑ *i*∈*d*. *dist* ((*P n*) *i*) (*Q i*)) **by** (*simp add*: *dist-finmap-def*)
also have ... ≤ (∑ *i*∈*d*. *e'*)
proof (*intro setsum-mono less-imp-le*)
fix *i* **assume** *i* ∈ *d*
hence *ni i* ≤ *Max* (*ni* ‘ *d*) **by** *simp*
also have ... ≤ *N* **by** (*simp add*: *N-def*)
also have ... ≤ *n* **using** ‹*N* ≤ *n*› **.**
finally
show *dist* ((*P n*) *i*) (*Q i*) < *e'*
using *ni* ‹*i* ∈ *d*› **by** (*auto simp*: *p-def q N-def*)
qed
also have ... = *card d* ∗ *e'* **by** (*simp add*: *real-eq-of-nat*)
also have ... < *e* **using** ‹*0* < *e*› **by** (*simp add*: *e'-def field-simps min-def*)
finally show *dist* (*P n*) *Q* < *e* **.**
qed
qed
thus *convergent P* **by** (*auto simp*: *convergent-def*)
qed

3.7 Polish Space of Finite Maps

instantiation *finmap* :: (*countable*, *polish-space*) *polish-space*
begin

definition *enum-basis-finmap* :: *nat* ⇒ ($'a \Rightarrow_F 'b$) *set* **where**
enum-basis-finmap n =
(*let m* = *from-nat n*::($'a \Rightarrow_F$ *nat*) *in Pi'* (*domain m*) (*enum-basis o* $(m)_F$))

lemma *range-enum-basis-eq*:
range enum-basis-finmap = {*Pi' I S*|*I S*. *finite I* ∧ (∀ *i* ∈ *I*. *S i* ∈ *range enum-basis*)}
proof (*auto simp*: *enum-basis-finmap-def*[*abs-def*])
fix *S*::($'a \Rightarrow 'b$ *set*) **and** *I*
assume ∀ *i*∈*I*. *S i* ∈ *range enum-basis*
hence ∀ *i*∈*I*. ∃ *n*. *S i* = *enum-basis n* **by** *auto*
then obtain *n* **where** *n*: ∀ *i*∈*I*. *S i* = *enum-basis* (*n i*)
unfolding *bchoice-iff* **by** *blast*
assume [*simp*]: *finite I*
have ∃ *fm*. *domain fm* = *I* ∧ (∀ *i*∈*I*. *n i* = (*fm i*))
by (*rule finmap-choice*) *auto*
then obtain *m* **where** *Pi' I S* = *Pi'* (*domain m*) (*enum-basis o m*)
using *n* **by** (*auto simp*: *Pi'-def*)
hence *Pi' I S* = (*let m* = *from-nat* (*to-nat m*) *in Pi'* (*domain m*) (*enum-basis* ∘ *m*))
by *simp*
thus *Pi' I S* ∈ *range* (λ*n*. *let m* = *from-nat n in Pi'* (*domain m*) (*enum-basis* ∘ *m*))
by *blast*
qed (*metis finite-domain o-apply rangeI*)

lemma *in-enum-basis-finmapI*:
assumes *finite I* **assumes** ⋀*i*. *i* ∈ *I* ⟹ *S i* ∈ *range enum-basis*
shows *Pi' I S* ∈ *range enum-basis-finmap*
using *assms* **unfolding** *range-enum-basis-eq* **by** *auto*

lemma *finmap-topological-basis*:
topological-basis (*range* (*enum-basis-finmap*))
proof (*subst topological-basis-iff*, *safe*)
fix *n*::*nat*
show *open* (*enum-basis-finmap n*::($'a \Rightarrow_F 'b$) *set*) **using** *enumerable-basis*
by (*auto intro*!: *open-Pi'I simp*: *topological-basis-def enum-basis-finmap-def Let-def*)
next
fix *O'*::($'a \Rightarrow_F 'b$) *set* **and** *x*
assume *open O' x* ∈ *O'*
then obtain *e* **where** *e*: *e* > *0* ⋀*y*. *dist y x* < *e* ⟹ *y* ∈ *O'* **unfolding** *open-dist* **by** *blast*
def *e'* ≡ *e* / (*card* (*domain x*) + *1*)

have ∃ *B*.
(∀ *i*∈*domain x*. *x i* ∈ *enum-basis* (*B i*) ∧ *enum-basis* (*B i*) ⊆ *ball* (*x i*) *e'*)
proof (*rule bchoice*, *safe*)
fix *i* **assume** *i* ∈ *domain x*

have *open* (*ball* (*x i*) *e'*) *x i* ∈ *ball* (*x i*) *e'* **using** *e*
by (*auto simp add*: *e'-def intro*!: *divide-pos-pos*)
from *enumerable-basisE*[*OF this*] **guess** *b'* **.**
thus ∃ *y*. *x i* ∈ *enum-basis y* ∧
enum-basis y ⊆ *ball* (*x i*) *e'* **by** *auto*
qed
then guess *B* **.. note** *B* = *this*
def *B'* ≡ *Pi'* (*domain x*) (λ*i*. *enum-basis* (*B i*)::'*b set*)
hence *B'* ∈ *range enum-basis-finmap* **unfolding** *B'-def*
by (*intro in-enum-basis-finmapI*) *auto*
moreover have *x* ∈ *B'* **unfolding** *B'-def* **using** *B* **by** *auto*
moreover have *B'* ⊆ *O'*
proof
fix *y* **assume** *y* ∈ *B'* **with** *B* **have** *domain y* = *domain x* **unfolding** *B'-def*
by (*simp add*: *Pi'-def*)
show *y* ∈ *O'*
proof (*rule e*)
have *dist y x* = ($\sum$ *i* ∈ *domain x*. *dist* (*y i*) (*x i*))
using ‹*domain y* = *domain x*› **by** (*simp add*: *dist-finmap-def*)
also have ... ≤ ($\sum$ *i* ∈ *domain x*. *e'*)
proof (*rule setsum-mono*)
fix *i* **assume** *i* ∈ *domain x*
with ‹*y* ∈ *B'*› *B* **have** *y i* ∈ *enum-basis* (*B i*)
by (*simp add*: *Pi'-def B'-def*)
hence *y i* ∈ *ball* (*x i*) *e'* **using** *B* ‹*domain y* = *domain x*› ‹*i* ∈ *domain x*›
by *force*
thus *dist* (*y i*) (*x i*) ≤ *e'* **by** (*simp add*: *dist-commute*)
qed
also have ... = *card* (*domain x*) ∗ *e'* **by** (*simp add*: *real-eq-of-nat*)
also have ... < *e* **using** *e* **by** (*simp add*: *e'-def field-simps*)
finally show *dist y x* < *e* **.**
qed
qed
ultimately
show ∃ *B'*∈*range enum-basis-finmap*. *x* ∈ *B'* ∧ *B'* ⊆ *O'* **by** *blast*
qed

lemma *range-enum-basis-finmap-imp-open*:
assumes *x* ∈ *range enum-basis-finmap*
shows *open x*
using *finmap-topological-basis assms* **by** (*auto simp*: *topological-basis-def*)

lemma
open-imp-ex-UNION-of-enum:
fixes *X*::('*a* $\Rightarrow_F$ '*b*) *set*
assumes *open X* **assumes** *X* ≠ {}
shows ∃ *A*::*nat*⇒'*a set*. ∃ *B*::*nat*⇒('*a* ⇒ '*b set*) . *X* = *UNION UNIV* (λ*i*. *Pi'* (*A i*) (*B i*)) ∧
(∀ *n*. ∀ *i*∈*A n*. (*B n*) *i* ∈ *range enum-basis*) ∧ (∀ *n*. *finite* (*A n*))

proof −
  from ‹*open X*› **obtain** B' **where** B': $B' \subseteq$ *range enum-basis-finmap* $\bigcup B' = X$
    using *finmap-topological-basis* **by** (*force simp add*: *topological-basis-def*)
  then obtain B **where** B: $B' = $ *enum-basis-finmap* ‘ B **by** (*auto simp*: *subset-image-iff*)
  show *?thesis*
  proof *cases*
    assume $B = \{\}$ **with** B **have** $B' = \{\}$ **by** *simp* **hence** *False* **using** B' *assms*
by *simp*
    thus *?thesis* **by** *simp*
  next
    assume $B \neq \{\}$ **then obtain** b **where** b: $b \in B$ **by** *auto*
    def $NA \equiv \lambda n{::}nat.$ *if* $n \in B$
      then domain ((*from-nat*::-⇒′*a* $\Rightarrow_F$ *nat*) *n*)
      else domain ((*from-nat*::-⇒′*a*$\Rightarrow_F$ *nat*) *b*)
    def $NB \equiv \lambda n{::}nat.$ *if* $n \in B$
      then ($\lambda i.$ (*enum-basis*::*nat*⇒′*b set*) (((*from-nat*::-⇒′*a* $\Rightarrow_F$ *nat*) *n*) *i*))
      else ($\lambda i.$ (*enum-basis*::*nat*⇒′*b set*) (((*from-nat*::-⇒′*a* $\Rightarrow_F$ *nat*) *b*) *i*))
    have X = *UNION UNIV* ($\lambda i.$ *Pi′* (*NA i*) (*NB i*)) **unfolding** B'(*2*)[*symmetric*]
using b
      unfolding B
      by *safe*
        (*auto simp add*: *NA-def NB-def enum-basis-finmap-def Let-def o-def split*:
split-if-asm)
    moreover
    have (∀ *n*. ∀ *i*∈*NA n*. (*NB n*) *i* ∈ *range enum-basis*)
      using *enumerable-basis* **by** (*auto simp*: *topological-basis-def NA-def NB-def*)
    moreover have (∀ *n*. *finite* (*NA n*)) **by** (*simp add*: *NA-def*)
    ultimately show *?thesis* **by** *auto*
  qed
qed

lemma
  open-imp-ex-UNION:
  fixes X::(′*a* $\Rightarrow_F$ ′*b*) *set*
  assumes *open X* **assumes** $X \neq \{\}$
  shows ∃ *A*::*nat*⇒′*a set*. ∃ *B*::*nat*⇒(′*a* ⇒ ′*b set*) . *X* = *UNION UNIV* ($\lambda i.$ *Pi′*
(*A i*) (*B i*)) ∧
    (∀ *n*. ∀ *i*∈*A n*. *open* ((*B n*) *i*)) ∧ (∀ *n*. *finite* (*A n*))
  using *open-imp-ex-UNION-of-enum*[*OF assms*]
  apply *auto*
  apply (*rule-tac x* = *A* **in** *exI*)
  apply (*rule-tac x* = *B* **in** *exI*)
  apply (*auto simp*: *open-enum-basis*)
  done

lemma
  open-basisE:
  assumes *open X* **assumes** $X \neq \{\}$
  obtains *A*::*nat*⇒′*a set* **and** *B*::*nat*⇒(′*a* ⇒ ′*b set*) **where**

$X = UNION\ UNIV\ (\lambda i.\ Pi'\ (A\ i)\ (B\ i))$ $\bigwedge n\ i.\ i{\in}A\ n \Longrightarrow open\ ((B\ n)\ i)$ $\bigwedge n.\ finite\ (A\ n)$
using *open-imp-ex-UNION*[*OF assms*] **by** *auto*

lemma
open-basis-of-enumE:
assumes *open X* **assumes** $X \neq \{\}$
obtains $A{::}nat{\Rightarrow}'a\ set$ **and** $B{::}nat{\Rightarrow}('a \Rightarrow 'b\ set)$ **where**
$X = UNION\ UNIV\ (\lambda i.\ Pi'\ (A\ i)\ (B\ i))$ $\bigwedge n\ i.\ i{\in}A\ n \Longrightarrow (B\ n)\ i \in range\ enum\text{-}basis$
$\bigwedge n.\ finite\ (A\ n)$
using *open-imp-ex-UNION-of-enum*[*OF assms*] **by** *auto*

instance proof qed (*blast intro*: *finmap-topological-basis*)

end

3.8 Product Measurable Space of Finite Maps

definition *PiF I M* $\equiv$
sigma
$(\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))$
$\{(\Pi'\ j{\in}J.\ X\ j)\ |X\ J.\ J \in I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$

abbreviation
$Pi_F\ I\ M \equiv PiF\ I\ M$

syntax
-PiF :: *pttrn* $\Rightarrow$ *'i set* $\Rightarrow$ *'a measure* $\Rightarrow$ (*'i* => *'a*) *measure* ((*3PIF* -:-./ -) *10*)

syntax (*xsymbols*)
-PiF :: *pttrn* $\Rightarrow$ *'i set* $\Rightarrow$ *'a measure* $\Rightarrow$ (*'i* => *'a*) *measure* (($3\Pi_F$ -$\in$-./ -) *10*)

syntax (*HTML* **output**)
-PiF :: *pttrn* $\Rightarrow$ *'i set* $\Rightarrow$ *'a measure* $\Rightarrow$ (*'i* => *'a*) *measure* (($3\Pi_F$ -$\in$-./ -) *10*)

translations
PIF x:*I*. *M* == *CONST PiF I* (%*x*. *M*)

lemma *PiF-gen-subset*: $\{(\Pi'\ j{\in}J.\ X\ j)\ |X\ J.\ J \in I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$ $\subseteq$
$Pow\ (\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))$
by (*auto simp*: *Pi'-def*) (*blast dest*: *sets-into-space*)

lemma *space-PiF*: $space\ (PiF\ I\ M) = (\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))$
unfolding *PiF-def* **using** *PiF-gen-subset* **by** (*rule space-measure-of*)

lemma *sets-PiF*:
$sets\ (PiF\ I\ M) = sigma\text{-}sets\ (\bigcup J \in I.\ (\Pi'\ j{\in}J.\ space\ (M\ j)))$

$\{(\Pi' j{\in}J.\ X\ j)\ |X\ J.\ J \in I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$
unfolding *PiF-def* **using** *PiF-gen-subset* **by** (*rule sets-measure-of*)

lemma *sets-PiF-singleton*:
$sets\ (PiF\ \{I\}\ M) = sigma\text{-}sets\ (\Pi' j{\in}I.\ space\ (M\ j))$
$\{(\Pi' j{\in}I.\ X\ j)\ |X.\ X \in (\Pi\ j{\in}I.\ sets\ (M\ j))\}$
unfolding *sets-PiF* **by** *simp*

lemma *in-sets-PiFI*:
assumes $X = (Pi'\ J\ S)\ J \in I\ \bigwedge i.\ i{\in}J \Longrightarrow S\ i \in sets\ (M\ i)$
shows $X \in sets\ (PiF\ I\ M)$
unfolding *sets-PiF*
using *assms* **by** *blast*

lemma *product-in-sets-PiFI*:
assumes $J \in I\ \bigwedge i.\ i{\in}J \Longrightarrow S\ i \in sets\ (M\ i)$
shows $(Pi'\ J\ S) \in sets\ (PiF\ I\ M)$
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
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 *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 *finmap-eq-Un*:
fixes $X::('a::countable \Rightarrow_F 'b)\ set$
shows $X = (\bigcup n.\ X \cap \{x.\ domain\ x = set\ (from\text{-}nat\ n)\})$
proof –
let $?P = \lambda i.\ finite\ i$
let $?f = \lambda s.\ \{x \in X.\ domain\ x = s\}$

have $X = \bigcup\{?f\ s\ |s.\ ?P\ s\}$ **by** *auto*
also have $\ldots = (\bigcup n.\ let\ s = set\ (from\text{-}nat\ n)\ in\ if\ ?P\ s\ then\ ?f\ s\ else\ \{\})$
by (*rule UN-finite-countable-eq-Un*) *simp*
also have $\ldots = (\bigcup n.\ \{x \in X.\ domain\ x = set\ (from\text{-}nat\ n)\})$
by (*intro UN-cong*) (*auto simp*: *Let-def space-PiF*)
finally show *?thesis* **by** *auto*
qed

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** $y \in sets\ N$
hence $A -`\ y \cap space\ (PiF\ I\ M) = (\bigcup n.\ A -`\ y \cap space\ (PiF\ (\{set\ (from\text{-}nat\ n)\}\cap I)\ M))$
by (*subst finmap-eq-Un*) (*auto simp*: *space-PiF Pi′-def*)
also have $\ldots \in sets\ (PiF\ I\ M)$
apply (*intro Int 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: $\bigwedge x.\ x \in space\ N \Longrightarrow domain\ (f\ x) \in I \land (\forall i{\in}domain\ (f\ x).\ (f\ x)\ i \in space\ (M\ i))$
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 $A[unfolded\ sets\text{-}PiF]$
apply $(induct\ A)$
unfolding $sets\text{-}PiF[symmetric]$ **unfolding** $space\text{-}PiF[symmetric]$
proof $-$
fix a **assume** $a \in \{Pi'\ J\ X\ |X\ J.\ J \in I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$
then obtain $K\ S$ **where** S: $a = Pi'\ K\ S\ K \in I\ (\forall\, i{\in}K.\ S\ i \in sets\ (M\ i))$
by *auto*
show $a \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M)$
proof *cases*
assume $K \in J$
hence $a \cap \{m.\ domain\ m \in J\} \in \{Pi'\ K\ X\ |X\ K.\ K \in J \wedge X \in (\Pi\ j{\in}K.\ sets\ (M\ j))\}$ **using** S
by $(auto\ intro!{:}\ exI[$**where** $x{=}K]\ exI[$**where** $x{=}S]\ simp{:}\ Pi'\text{-}def)$
also have $\ldots \subseteq sets\ (PiF\ J\ M)$ **unfolding** $sets\text{-}PiF$ **by** *auto*
finally show *?thesis* **.**
next
assume $K \notin J$
hence $a \cap \{m.\ domain\ m \in J\} = \{\}$ **using** S **by** $(auto\ simp{:}\ Pi'\text{-}def)$
also have $\ldots \in sets\ (PiF\ J\ M)$ **by** *simp*
finally show *?thesis* **.**
qed
next
show $\{\} \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M)$ **by** *simp*
next
fix $a :: nat \Rightarrow \text{-}$
assume a: $(\bigwedge i.\ a\ i \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M))$
have $UNION\ UNIV\ a \cap \{m.\ domain\ m \in J\} = (\bigcup i.\ (a\ i \cap \{m.\ domain\ m \in J\}))$
by *simp*
also have $\ldots \in sets\ (PiF\ J\ M)$ **using** a **by** $(intro\ countable\text{-}nat\text{-}UN)$ *auto*
finally show $UNION\ UNIV\ a \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M)$ **.**
next
fix a **assume** a: $a \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M)$
have $(space\ (PiF\ I\ M) - a) \cap \{m.\ domain\ m \in J\} = (space\ (PiF\ J\ M) - (a \cap \{m.\ domain\ m \in J\}))$
using $\langle J \subseteq I\rangle$ **by** $(auto\ simp{:}\ space\text{-}PiF\ Pi'\text{-}def)$
also have $\ldots \in sets\ (PiF\ J\ M)$ **using** a **by** *auto*
finally show $(space\ (PiF\ I\ M) - a) \cap \{m.\ domain\ m \in J\} \in sets\ (PiF\ J\ M)$ **.**
qed

lemma *measurable-finmap-of*:
assumes f: $\bigwedge i.\ (\exists\, x \in space\ N.\ i \in J\ x) \Longrightarrow (\lambda x.\ f\ x\ i) \in measurable\ N\ (M\ i)$
assumes J: $\bigwedge x.\ x \in space\ N \Longrightarrow J\ x \in I$ $\bigwedge x.\ x \in space\ N \Longrightarrow finite\ (J\ x)$
assumes JN: $\bigwedge S.\ \{x.\ J\ x = S\} \cap space\ N \in sets\ N$
shows $(\lambda x.\ finmap\text{-}of\ (J\ x)\ (f\ x)) \in measurable\ N\ (PiF\ I\ M)$
proof $(rule\ measurable\text{-}PiF)$
fix x **assume** $x \in space\ N$
with $J[of\ x]\ measurable\text{-}space[OF\ f]$

show *domain* (*finmap-of* (*J x*) (*f x*)) $\in$ *I* $\wedge$
($\forall$ *i*$\in$*domain* (*finmap-of* (*J x*) (*f x*)). (*finmap-of* (*J x*) (*f x*)) *i* $\in$ *space* (*M* *i*))
by *auto*
next
fix *K S* **assume** *K* $\in$ *I* **and** $*$: $\bigwedge$*i*. *i* $\in$ *K* $\Longrightarrow$ *S i* $\in$ *sets* (*M i*)
with *J* **have** *eq*: (λ*x*. *finmap-of* (*J x*) (*f x*)) $-$' *Pi'* *K S* $\cap$ *space N* =
(*if* $\exists$ *x* $\in$ *space N*. *K* = *J x* $\wedge$ *finite K then if K* = {} *then* {*x* $\in$ *space N*. *J x* = *K*}
else ($\bigcap$ *i*$\in$*K*. (λ*x*. *f x i*) $-$' *S i* $\cap$ {*x* $\in$ *space N*. *J x* = *K*}) *else* {})
by (*auto simp*: *Pi'-def*)
have *r*: {*x* $\in$ *space N*. *J x* = *K*} = *space N* $\cap$ ({*x*. *J x* = *K*} $\cap$ *space N*) **by** *auto*
show (λ*x*. *finmap-of* (*J x*) (*f x*)) $-$' *Pi'* *K S* $\cap$ *space N* $\in$ *sets N*
unfolding *eq r*
apply (*simp del*: *INT-simps add*:)
apply (*intro conjI impI finite-INT JN Int*[*OF top*])
apply *simp* **apply** *assumption*
apply (*subst Int-assoc*[*symmetric*])
apply (*rule 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* $\in$ *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* $\in$ *sets* (*M i*) *finite I*
shows (λ*x*. $(x)_F$ *i*) $-$' *A* $\cap$ *space* (*PiF* {*I*} *M*) $\in$ *sets* (*PiF* {*I*} *M*)
proof *cases*
assume *i* $\in$ *I*
hence (λ*x*. $(x)_F$ *i*) $-$' *A* $\cap$ *space* (*PiF* {*I*} *M*) =
Pi' *I* (λ*x*. *if x* = *i then A else space* (*M x*))
using *sets-into-space*[*OF*] ‹*A* $\in$ *sets* (*M i*)› *assms*
by (*auto simp*: *space-PiF Pi'-def*)
thus *?thesis* **using** *assms* ‹*A* $\in$ *sets* (*M i*)›
by (*intro in-sets-PiFI*) *auto*
next
assume *i* $\notin$ *I*

hence $(\lambda x.\ (x)_F\ i)$ −' $A \cap$ *space* (*PiF* $\{I\}$ M) =
(*if undefined* $\in A$ *then space* (*PiF* $\{I\}$ M) *else* $\{\}$) **by** (*auto simp*: *space-PiF* *Pi'-def*)
thus *?thesis* **by** *simp*
qed

lemma *measurable-proj-singleton*:
fixes I
assumes *finite* I $i \in I$
shows $(\lambda x.\ (x)_F\ i) \in$ *measurable* (*PiF* $\{I\}$ M) (M i)
proof (*unfold measurable-def*, *intro CollectI conjI ballI proj-measurable-singleton assms*)
qed (*insert* ‹$i \in I$›, *auto simp*: *space-PiF*)

lemma *measurable-proj-countable*:
fixes I::'*a*::*countable set set*
assumes $y \in$ *space* (M i)
shows (λx. *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 (λx. *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 (λx. *if* $i \in$ *domain* x *then* x i *else* y) −' $z \cap$ *space* (*PiF* $\{J\}$ M) =
(λx. *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 ... $\in$ *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 (λx. *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*) $\to$ (Π_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 {ω $\in$ *space* (*PiF* {*J*} *M*). $(\omega)_F$ *i* $\in$ *A*} $\in$ *sets* (*PiF* {*J*} *M*)
proof
have {ω $\in$ *space* (*PiF* {*J*} *M*). $(\omega)_F$ *i* $\in$ *A*} =
($\lambda\omega$. $(\omega)_F$ *i*) −‘ *A* $\cap$ *space* (*PiF* {*J*} *M*) **by** *auto*
also have ... $\in$ *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 *sets-subspaceI*:
assumes *A* $\cap$ *space M* $\in$ *sets M*
assumes *B* $\in$ *sets M*
shows *A* $\cap$ *B* $\in$ *sets M* **using** *assms*
proof −
have *A* $\cap$ *B* = (*A* $\cap$ *space M*) $\cap$ *B*
using *assms sets-into-space* **by** *auto*
thus *?thesis* **using** *assms* **by** *auto*
qed

lemma *space-PiF-singleton-eq-product*:
assumes *finite I*
shows *space* (*PiF* {*I*} *M*) = (Π' *i*$\in$*I*. *space* (*M i*))
by (*auto simp*: *product-def space-PiF assms*)

adapted from *sets* (Pi_M *?I ?M*) = *sigma-sets* (Π_E *i*$\in$*?I*. *space* (*?M i*)) {{*f* $\in$ Π_E *i*$\in$*?I*. *space* (*?M i*). *f i* $\in$ *A*} |*i A*. *i* $\in$ *?I* $\wedge$ *A* $\in$ *sets* (*?M i*)}

lemma *sets-PiF-single*:
assumes *finite I I* $\neq$ {}
shows *sets* (*PiF* {*I*} *M*) =
sigma-sets (Π' *i*$\in$*I*. *space* (*M i*))
{{*f*$\in$$\Pi'$ *i*$\in$*I*. *space* (*M i*). *f i* $\in$ *A*} | *i A*. *i* $\in$ *I* $\wedge$ *A* $\in$ *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* $\in$ {*Pi′ I X* |*X*. *X* $\in$ (Π *j*$\in$*I*. *sets* (*M j*))}
then obtain *X* **where** *X*: *A* = *Pi′ I X X* $\in$ (Π *j*$\in$*I*. *sets* (*M j*)) **by** *auto*
show *A* $\in$ *sigma-sets ?Ω ?R*
proof −
from ‹*I* $\neq$ {}› *X* **have** *A* = ($\bigcap$*j*$\in$*I*. {*f*$\in$*space* (*PiF* {*I*} *M*). *f j* $\in$ *X j*})
using *sets-into-space*

by (*auto simp*: *space-PiF product-def*) *blast*
also have ... $\in$ *sigma-sets* $?\Omega$ $?R$
using X ‹$I \neq \{\}$› *assms* **by** (*intro R.finite-INT*) (*auto simp*: *space-PiF*)
finally show $A \in$ *sigma-sets* $?\Omega$ $?R$ **.**
qed
next
fix A **assume** $A \in ?R$
then obtain i B **where** A: $A = \{f{\in}\Pi'\ i{\in}I.\ space\ (M\ i).\ f\ i \in B\}$ $i \in I$ $B \in$ *sets* (M i)
by *auto*
then have $A = (\Pi'\ j \in I.\ if\ j = i\ then\ B\ else\ space\ (M\ j))$
using *sets-into-space*[*OF A*(*3*)]
apply (*auto simp*: *Pi'-iff split*: *split-if-asm*)
apply *blast*
done
also have ... $\in$ *sigma-sets* $?\Omega$ $\{Pi'\ I\ X\ |X.\ X \in (\Pi\ j{\in}I.\ sets\ (M\ j))\}$
using A
by (*intro sigma-sets.Basic*)
(*auto intro*: *exI*[**where** $x{=}\lambda j.\ if\ j = i\ then\ B\ else\ space\ (M\ j)$])
finally show $A \in$ *sigma-sets* $?\Omega$ $\{Pi'\ I\ X\ |X.\ X \in (\Pi\ j{\in}I.\ sets\ (M\ j))\}$ **.**
qed

adapted from $(\bigwedge i.\ i \in ?I \Longrightarrow ?A\ i = ?B\ i) \Longrightarrow Pi_E\ ?I\ ?A = Pi_E\ ?I\ ?B$

lemma *Pi'-cong*:
assumes *finite I*
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i = g\ i$
shows $Pi'\ I\ f = Pi'\ I\ g$
using *assms* **by** (*auto simp*: *Pi'-def*)

adapted from $[\![finite\ ?I;\ \bigwedge i\ n\ m.\ [\![i \in ?I;\ n \le m]\!] \Longrightarrow ?A\ n\ i \subseteq ?A\ m\ i]\!] \Longrightarrow (\bigcup_n\ Pi\ ?I\ (?A\ n)) = (\Pi\ i{\in}?I.\ \bigcup_n\ ?A\ n\ i)$

lemma *Pi'-UN*:
fixes $A :: nat \Rightarrow 'i \Rightarrow 'a\ set$
assumes *finite I*
assumes *mono*: $\bigwedge i\ n\ m.\ i \in I \Longrightarrow n \le m \Longrightarrow A\ n\ i \subseteq A\ m\ i$
shows $(\bigcup n.\ Pi'\ I\ (A\ n)) = Pi'\ I\ (\lambda i.\ \bigcup n.\ A\ n\ i)$
proof (*intro set-eqI iffI*)
fix f **assume** $f \in Pi'\ I\ (\lambda i.\ \bigcup n.\ A\ n\ i)$
then have $\forall i{\in}I.\ \exists n.\ f\ i \in A\ n\ i$ *domain* $f = I$ **by** (*auto simp*: ‹*finite I*› *Pi'-def*)
from *bchoice*[*OF this*(*1*)] **obtain** n **where** n: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in (A\ (n\ i)\ i)$ **by** *auto*
obtain k **where** k: $\bigwedge i.\ i \in I \Longrightarrow n\ i \le k$
using ‹*finite I*› *finite-nat-set-iff-bounded-le*[*of n‘I*] **by** *auto*
have $f \in Pi'\ I\ (\lambda i.\ A\ k\ i)$
proof
fix i **assume** $i \in I$
from *mono*[*OF this*, *of n i k*] *k*[*OF this*] *n*[*OF this*] ‹*domain* $f = I$› ‹$i \in I$›
show $f\ i \in A\ k\ i$ **by** (*auto simp*: ‹*finite I*›)
qed (*simp add*: ‹*domain* $f = I$› ‹*finite I*›)

then show $f \in (\bigcup n.\ Pi'\ I\ (A\ n))$ **by** *auto*
qed (*auto simp*: *Pi'-def* ‹*finite I*›)

adapted from $[\![finite\ ?I;\ \bigwedge i.\ i \in ?I \Longrightarrow incseq\ (?S\ i);\ \bigwedge i.\ i \in ?I \Longrightarrow (\bigcup_j\ ?S\ i\ j) = space\ (?M\ i);\ \bigwedge i.\ i \in ?I \Longrightarrow range\ (?S\ i) \subseteq ?E\ i;\ \bigwedge i.\ i \in ?I \Longrightarrow ?E\ i \subseteq Pow\ (space\ (?M\ i));\ \bigwedge i.\ i \in ?I \Longrightarrow sets\ (?M\ i) = sigma\text{-}sets\ (space\ (?M\ i))\ (?E\ i)]\!] \Longrightarrow sets\ (Pi_M\ ?I\ ?M) = sigma\text{-}sets\ (space\ (Pi_M\ ?I\ ?M))\ \{Pi_E\ ?I\ F\ |F.\ \forall i{\in}?I.\ F\ i \in ?E\ i\}$

lemma *sigma-fprod-algebra-sigma-eq*:
  fixes $E :: 'i \Rightarrow 'a\ set\ set$
  assumes [*simp*]: $finite\ I\ I \neq \{\}$
  assumes *S-mono*: $\bigwedge i.\ i \in I \Longrightarrow incseq\ (S\ i)$
    and *S-union*: $\bigwedge i.\ i \in I \Longrightarrow (\bigcup j.\ S\ i\ j) = space\ (M\ i)$
    and *S-in-E*: $\bigwedge i.\ i \in I \Longrightarrow range\ (S\ i) \subseteq E\ i$
  assumes *E-closed*: $\bigwedge i.\ i \in I \Longrightarrow E\ i \subseteq Pow\ (space\ (M\ i))$
    and *E-generates*: $\bigwedge i.\ i \in I \Longrightarrow sets\ (M\ i) = sigma\text{-}sets\ (space\ (M\ i))\ (E\ i)$
  defines $P == \{\ Pi'\ I\ F\ |\ F.\ \forall i{\in}I.\ F\ i \in E\ i\ \}$
  shows $sets\ (PiF\ \{I\}\ M) = sigma\text{-}sets\ (space\ (PiF\ \{I\}\ M))\ P$
proof
  let $?P = sigma\ (space\ (Pi_F\ \{I\}\ M))\ P$
  have *P-closed*: $P \subseteq Pow\ (space\ (Pi_F\ \{I\}\ M))$
    using *E-closed* **by** (*auto simp*: *space-PiF P-def Pi'-iff subset-eq*)
  then have *space-P*: $space\ ?P = (\Pi'\ i{\in}I.\ space\ (M\ i))$
    by (*simp add*: *space-PiF*)
  have $sets\ (PiF\ \{I\}\ M) =$
    $sigma\text{-}sets\ (space\ ?P)\ \{\{f \in \Pi'\ i{\in}I.\ space\ (M\ i).\ f\ i \in A\}\ |i\ A.\ i \in I \wedge A \in sets\ (M\ i)\}$
    using *sets-PiF-single*[*of I M*] **by** (*simp add*: *space-P*)
  also have $\ldots \subseteq sets\ (sigma\ (space\ (PiF\ \{I\}\ M))\ P)$
  proof (*safe intro*!: *sigma-sets-subset*)
    fix *i A* **assume** $i \in I$ **and** *A*: $A \in sets\ (M\ i)$
    have $(\lambda x.\ (x)_F\ i) \in measurable\ ?P\ (sigma\ (space\ (M\ i))\ (E\ i))$
    proof (*subst measurable-iff-measure-of*)
      show $E\ i \subseteq Pow\ (space\ (M\ i))$ **using** ‹$i \in I$› **by** *fact*
      from *space-P* ‹$i \in I$› **show** $(\lambda x.\ (x)_F\ i) \in space\ ?P \rightarrow space\ (M\ i)$
        by *auto*
      show $\forall A{\in}E\ i.\ (\lambda x.\ (x)_F\ i)\ -`\ A \cap space\ ?P \in sets\ ?P$
      proof
        fix *A* **assume** *A*: $A \in E\ i$
        then have $(\lambda x.\ (x)_F\ i)\ -`\ A \cap space\ ?P = (\Pi'\ j{\in}I.\ if\ i = j\ then\ A\ else\ space\ (M\ j))$
          using *E-closed* ‹$i \in I$› **by** (*auto simp*: *space-P Pi-iff subset-eq split*: *split-if-asm*)
        also have $\ldots = (\Pi'\ j{\in}I.\ \bigcup n.\ if\ i = j\ then\ A\ else\ S\ j\ n)$
          by (*intro Pi'-cong*) (*simp-all add*: *S-union*)
        also have $\ldots = (\bigcup n.\ \Pi'\ j{\in}I.\ if\ i = j\ then\ A\ else\ S\ j\ n)$
          using *S-mono*
          by (*subst Pi'-UN*[*symmetric*, *OF* ‹*finite I*›]) (*auto simp*: *incseq-def*)
        also have $\ldots \in sets\ ?P$

proof (*safe intro*!: *countable-UN*)
fix *n* **show** (Π' *j*∈*I*. *if i* = *j then A else S j n*) ∈ *sets ?P*
using *A S-in-E*
by (*simp add*: *P-closed*)
(*auto simp*: *P-def subset-eq intro*!: *exI*[*of* - λ*j*. *if i* = *j then A else S j n*])
qed
finally show (λ*x*. $(x)_F$ *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 (λ*x*. $(x)_F$ *i*) −' *A* ∩ *space ?P* ∈ *sets ?P*
by (*simp add*: *E-generates*)
also have (λ*x*. $(x)_F$ *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*!: *sigma-sets-subset product-in-sets-PiFI simp*: *E-generates P-def*)
qed

lemma *enumerable-sigma-fprod-algebra-sigma-eq*:
assumes *I* ≠ {}
assumes [*simp*]: *finite I*
shows *sets* (*PiF* {*I*} (λ-. *borel*)) = *sigma-sets* (*space* (*PiF* {*I*} (λ-. *borel*)))
{*Pi' I F* |*F*. (∀ *i*∈*I*. *F i* ∈ *range enum-basis*)}
proof −
from *open-incseqE*[*OF open-UNIV*] **guess** *S*::*nat* ⇒ '*b set* **. note** *S* = *this*
show *?thesis*
proof (*rule sigma-fprod-algebra-sigma-eq*)
show *finite I* **by** *simp*
show *I* ≠ {} **by** *fact*
show *incseq S* (⋃*j*. *S j*) = *space borel range S* ⊆ *range enum-basis*
using *S* **by** *simp-all*
show *range enum-basis* ⊆ *Pow* (*space borel*) **by** *simp*
show *sets borel* = *sigma-sets* (*space borel*) (*range enum-basis*)
using *borel-eq-sigma-enum-basis* **.**
qed
qed

adapted from ⟦*?I* ≠ {}; *finite ?I*⟧ ⟹ *sets* (Pi_F {*?I*} (λ-. *borel*)) = *sigma-sets* (*space* (Pi_F {*?I*} (λ-. *borel*))) {*Pi' ?I F* |*F*. ∀ *i*∈*?I*. *F i* ∈ *range enum-basis*}

lemma *enumerable-sigma-prod-algebra-sigma-eq*:
assumes *I* ≠ {}
assumes [*simp*]: *finite I*

shows *sets* (*PiM I* (λ-. *borel*)) = *sigma-sets* (*space* (*PiM I* (λ-. *borel*)))
{Pi_E *I F* |*F*. ∀ *i*∈*I*. *F i* ∈ *range enum-basis*}
proof −
from *open-incseqE*[*OF open-UNIV*] **guess** *S*::*nat* ⇒ ′*b set* **. note** *S* = *this*
show *?thesis*
proof (*rule sigma-prod-algebra-sigma-eq*)
show *finite I* **by** *simp* **note**[[*show-types*]]
fix *i* **show** *incseq S* (⋃*j*. *S j*) = *space borel range S* ⊆ *range enum-basis*
using *S* **by** *simp-all*
show *range enum-basis* ⊆ *Pow* (*space borel*) **by** *simp*
show *sets borel* = *sigma-sets* (*space borel*) (*range enum-basis*)
using *borel-eq-sigma-enum-basis* **.**
qed
qed

lemma *product-open-generates-sets-PiF-single*:
assumes *I* ≠ {}
assumes [*simp*]: *finite I*
shows *sets* (*PiF* {*I*} (λ-. *borel*::′*b*::*enumerable-basis measure*)) =
sigma-sets (*space* (*PiF* {*I*} (λ-. *borel*))) {*Pi′ I F* |*F*. (∀ *i*∈*I*. *F i* ∈ *Collect open*)}
proof −
from *open-incseqE*[*OF open-UNIV*] **guess** *S*::*nat* ⇒ ′*b set* **. note** *S* = *this*
show *?thesis*
proof (*rule sigma-fprod-algebra-sigma-eq*)
show *finite I* **by** *simp*
show *I* ≠ {} **by** *fact*
show *incseq S* (⋃*j*. *S j*) = *space borel range S* ⊆ *Collect open*
using *S* **by** (*auto simp*: *open-enum-basis*)
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 *product-open-generates-sets-PiM*:
assumes *I* ≠ {}
assumes [*simp*]: *finite I*
shows *sets* (*PiM I* (λ-. *borel*::′*b*::*enumerable-basis measure*)) =
sigma-sets (*space* (*PiM I* (λ-. *borel*))) {Pi_E *I F* |*F*. ∀ *i*∈*I*. *F i* ∈ *Collect open*}
proof −
from *open-incseqE*[*OF open-UNIV*] **guess** *S*::*nat* ⇒ ′*b set* **. note** *S* = *this*
show *?thesis*
proof (*rule sigma-prod-algebra-sigma-eq*)
show *finite I* **by** *simp* **note**[[*show-types*]]
fix *i* **show** *incseq S* (⋃*j*. *S j*) = *space borel range S* ⊆ *Collect open*
using *S* **by** (*auto simp*: *open-enum-basis*)
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*]: $(\bigcup J{\in}Collect\ finite.\ J \rightsquigarrow UNIV) = UNIV$ **by** *auto*

lemma *borel-eq-PiF-borel*:
shows $(borel :: ('i::countable \Rightarrow_F 'a::polish\text{-}space)\ measure) =$
$PiF\ (Collect\ finite)\ (\lambda\text{-}.\ borel :: 'a\ measure)$
proof (*rule measure-eqI*)
have *C*: $Collect\ finite \neq \{\}$ **by** *auto*
show $sets\ (borel::('i \Rightarrow_F 'a)\ measure) = sets\ (PiF\ (Collect\ finite)\ (\lambda\text{-}.\ borel))$
proof
show $sets\ (borel::('i \Rightarrow_F 'a)\ measure) \subseteq sets\ (PiF\ (Collect\ finite)\ (\lambda\text{-}.\ borel))$
apply (*simp add*: *borel-def sets-PiF*)
proof (*rule sigma-sets-mono*, *safe*, *cases*)
fix $X::('i \Rightarrow_F 'a)\ set$ **assume** $open\ X\ X \neq \{\}$
from *open-basisE*[*OF this*] **guess** *NA NB* **. note** $N = this$
hence $X = (\bigcup i.\ Pi'\ (NA\ i)\ (NB\ i))$ **by** *simp*
also have $\ldots \in$
$sigma\text{-}sets\ UNIV\ \{Pi'\ J\ S\ |S\ J.\ finite\ J \wedge S \in J \rightarrow sigma\text{-}sets\ UNIV$
$(Collect\ open)\}$
using *N* **by** (*intro Union sigma-sets.Basic*) *blast*
finally show $X \in sigma\text{-}sets\ UNIV$
$\{Pi'\ J\ X\ |X\ J.\ finite\ J \wedge X \in J \rightarrow sigma\text{-}sets\ UNIV\ (Collect\ open)\}$ **.**
qed (*auto simp*: *Empty*)
next
show $sets\ (PiF\ (Collect\ finite)\ (\lambda\text{-}.\ borel)) \subseteq sets\ (borel::('i \Rightarrow_F 'a)\ measure)$
proof
fix *x* **assume** *x*: $x \in sets\ (PiF\ (Collect\ finite::'i\ set\ set)\ (\lambda\text{-}.\ borel::'a\ measure))$
hence *x-sp*: $x \subseteq space\ (PiF\ (Collect\ finite)\ (\lambda\text{-}.\ borel))$ **by** (*rule sets-into-space*)
from *finmap-eq-Un* **have** $x = (\bigcup n.\ x \cap \{xa.\ domain\ xa = set\ (from\text{-}nat\ n)\})$
(**is** $- = (\bigcup n.\ ?rx\ n)$).
also have $\ldots \in sets\ borel$
proof (*rule countable-nat-UN*, *safe*)
fix *i*
{ assume *ef*: $set\ (from\text{-}nat\ i) = (\{\}::'i\ set)$
{ assume *e*: $(?rx\ i) = \{\}$
hence $(?rx\ i) \in sets\ borel$ **unfolding** *e* **by** *simp*
} moreover {
assume $(?rx\ i) \neq \{\}$
then obtain *f* **where** $f \in x\ domain\ f = \{\}$ **using** *ef* **by** *auto*
hence $(?rx\ i) = \{f\}$ **using** ‹$set\ (from\text{-}nat\ i) = \{\}$›
by (*auto simp*: *finmap-eq-iff*)
also have $\{f\} \in sets\ borel$ **by** *simp*
finally have $(?rx\ i) \in sets\ borel$ **.**
} ultimately have $(?rx\ i) \in sets\ borel$ **by** *blast*
} moreover {
assume $set\ (from\text{-}nat\ i) \neq (\{\}::'i\ set)$

from *open-incseqE*[*OF open-UNIV*] **guess** S::*nat* ⇒ *'a set* **. note** S = *this*
have (*?rx i*) = x ∩ {*m. domain m* ∈ {*set* (*from-nat i*)}} **by** *auto*
also have ... ∈ *sets* (*PiF* {*set* (*from-nat i*)} (λ-. *borel*))
using x **apply** (*rule restrict-sets-measurable*) **by** (*simp add*: *enum-finite-def*)
also have ... = *sigma-sets* (*space* (*PiF* {*set* (*from-nat i*)} (λ-. *borel*)))
{*Pi'* (*set* (*from-nat i*)) *F* |*F*. (∀ *j*∈*set* (*from-nat i*). *F j* ∈ *range enum-basis*)}
(**is** - = *sigma-sets* - *?P*)
by (*rule enumerable-sigma-fprod-algebra-sigma-eq*[*OF* ‹*set* (*from-nat i*) ≠ {}›])
(*simp add*: *enum-finite-def*)
also have ... ⊆ *sets borel*
proof
fix x
assume x ∈ *sigma-sets* (*space* (*PiF* {*set* (*from-nat i*)} (λ-. *borel*))) *?P*
thus x ∈ *sets borel*
proof (*rule sigma-sets.induct*, *safe*)
fix *F*::*'i* ⇒ *'a set*
assume ∀ *j*∈*set* (*from-nat i*). *F j* ∈ *range enum-basis*
hence *Pi'* (*set* (*from-nat i*)) *F* ∈ *range enum-basis-finmap*
unfolding *range-enum-basis-eq* **by** *auto*
hence *open* (*Pi'* (*set* (*from-nat i*)) *F*) **by** (*rule range-enum-basis-finmap-imp-open*)
thus *Pi'* (*set* (*from-nat i*)) *F* ∈ *sets borel* **by** *simp*
next
fix *a*::(*'i* $\Rightarrow_F$ *'a*) *set*
have *space* (*PiF* {*set* (*from-nat i*)::*'i set*} (λ-. *borel*::*'a measure*)) =
Pi' (*set* (*from-nat i*)) (λ-. *UNIV*)
by (*auto simp*: *space-PiF product-def enum-finite-def*)
moreover have *open* (*Pi'* (*set* (*from-nat i*)::*'i set*) (λ-. *UNIV*::*'a set*))
by (*intro open-Pi'I*) (*auto simp*: *enum-finite-def*)
ultimately
have *space* (*PiF* {*set* (*from-nat i*)::*'i set*} (λ-. *borel*::*'a measure*)) ∈ *sets borel*
by *simp*
moreover
assume *a* ∈ *sets borel*
ultimately show *space* (*PiF* {*set* (*from-nat i*)} (λ-. *borel*)) − *a* ∈ *sets borel* **..**
qed *auto*
qed
finally have (*?rx i*) ∈ *sets borel* **.**
} **ultimately show** (*?rx i*) ∈ *sets borel* **by** *blast*
qed
finally show x ∈ *sets* (*borel*) **.**
qed
qed
qed (*simp add*: *emeasure-sigma borel-def PiF-def*)

3.9 Measure preservation

Measure preservation is not used at the moment.

definition *measure-preserving f A B* $\longleftrightarrow$ *f* $\in$ *measurable A B* $\wedge$ ($\forall x \in$ *sets B*. *distr A B f x* = *B x*)

lemma
  assumes *measure-preserving f A B*
  shows *measure-preserving-distr*: $\bigwedge x.\ x \in$ *sets B* $\Longrightarrow$ *distr A B f x* = *B x*
  and *measure-preserving-measurable*: *f* $\in$ *measurable A B*
  using *assms* **by** (*auto simp*: *measure-preserving-def*)

lemma *measure-preservingI*:
  assumes *f* $\in$ *measurable A B* $\bigwedge x.\ x \in$ *sets B* $\Longrightarrow$ *distr A B f x* = *B x*
  shows *measure-preserving f A B*
  using *assms* **by** (*auto simp*: *measure-preserving-def*)

lemma *measure-preservingI'*[*intro*]:
  assumes *AB*: *f* $\in$ *measurable A B*
  assumes *m*: $\bigwedge x.\ x \in$ *sets B* $\Longrightarrow$ *emeasure A* (*f* −‘ *x* $\cap$ *space A*) = *emeasure B*
x
  shows *measure-preserving f A B*
  apply (*rule measure-preservingI*[*OF AB*])
  apply (*subst emeasure-distr*[*OF AB*])
  apply *assumption*
  apply (*rule m*)
  apply *assumption*
  done

lemma
  measure-preserving-comp:
  assumes *AB*: *measure-preserving f A B*
  assumes *BC*: *measure-preserving g B C*
  shows *measure-preserving* (*g o f*) *A C*
proof
  note *mAB* = *measure-preserving-measurable*[*OF AB*]
  note *mBC* = *measure-preserving-measurable*[*OF BC*]
  show *g o f* $\in$ *measurable A C*
    using *mAB mBC* **..**
  fix *x* **assume** *x* $\in$ *sets C*
  hence *C x* = *distr B C g x*
    by (*rule measure-preserving-distr*[*OF BC*, *symmetric*])
  also have ... = *B* (*g* −‘ *x* $\cap$ *space B*)
    using *mBC* ‹*x* $\in$ *sets C*› **by** (*rule emeasure-distr*)
  also have ... = *distr A B f* (*g* −‘ *x* $\cap$ *space B*)
    using *measurable-sets*[*OF mBC* ‹*x* $\in$ *sets C*›]
    by (*rule measure-preserving-distr*[*OF AB*, *symmetric*])
  also have ... = *emeasure A* (*f* −‘ (*g* −‘ *x* $\cap$ *space B*) $\cap$ *space A*)
    using *mAB measurable-sets*[*OF mBC* ‹*x* $\in$ *sets C*›]

by (*rule emeasure-distr*)
also have ... = *emeasure* A (f −‘ g −‘ x ∩ (f −‘ *space* B ∩ *space* A))
by (*simp add*: *Int-assoc*)
also have f −‘ *space* B ∩ *space* A = *space* A
using *sets-into-space*[*OF measurable-sets*[*OF mAB top*]] *measurable-space*[*OF mAB*]
by *auto*
finally show *emeasure* A ((g ∘ f) −‘ x ∩ *space* A) = *emeasure* C x
by (*simp add*: *vimage-compose*)
qed

3.10 Isomorphism between Functions and Finite Maps

lemma
measurable-compose:
fixes f::'a ⇒ 'b
assumes *inj*: ⋀j. j ∈ J ⟹ f' (f j) = j
assumes *finite* J
shows (λm. *compose* J m f) ∈ *measurable* (*PiM* (f ‘ J) (λ-. M)) (*PiM* J (λ-. M))
proof (*rule measurable-PiM*)
show (λm. *compose* J m f)
∈ *space* (Pi_M (f ‘ J) (λ-. M)) →
(J → *space* M) ∩ *extensional* J
proof *safe*
fix x **and** i
assume x: x ∈ *space* (*PiM* (f ‘ J) (λ-. M)) i ∈ J
with *inj* **show** *compose* J x f i ∈ *space* M
by (*auto simp*: *space-PiM compose-def*)
next
fix x **assume** x ∈ *space* (*PiM* (f ‘ J) (λ-. M))
show (*compose* J x f) ∈ *extensional* J **by** (*rule compose-extensional*)
qed
next
fix S X
have *inv*: ⋀j. j ∈ f ‘ J ⟹ f (f' j) = j **using** *assms* **by** *auto*
assume S: S ≠ {} ∨ J = {} *finite* S S ⊆ J **and** P: ⋀i. i ∈ S ⟹ X i ∈ *sets* M
have (λm. *compose* J m f) −‘ *prod-emb* J (λ-. M) S (Pi_E S X) ∩
space (Pi_M (f ‘ J) (λ-. M)) = *prod-emb* (f ‘ J) (λ-. M) (f ‘ S) (Pi_E (f ‘ S) (λb. X (f' b)))
using *assms inv* S *sets-into-space*[*OF* P]
by (*force simp*: *prod-emb-iff compose-def space-PiM extensional-def Pi-def intro*: *imageI*)
also have ... ∈ *sets* (Pi_M (f ‘ J) (λ-. M))
proof
from S **show** f ‘ S ⊆ f ‘ J **by** *auto*
show (Π_E b∈f ‘ S. X (f' b)) ∈ *sets* (Pi_M (f ‘ S) (λ-. M))
proof
show *finite* (f ‘ S) **using** S **by** *simp*

fix i **assume** $i \in f$ ' S **hence** $f'\ i \in S$ **using** S *assms* **by** *auto*
thus $X\ (f'\ i) \in$ *sets* M **by** (*rule* P)
qed
qed
finally show ($\lambda m.$ *compose* $J\ m\ f$) $-$' *prod-emb* J (λ-. M) S ($Pi_E\ S\ X$) $\cap$
space (Pi_M (f ' J) (λ-. M)) $\in$ *sets* (Pi_M (f ' J) (λ-. M)) **.**
qed

lemma
measurable-compose-inv:
fixes f::'$a \Rightarrow$ 'b
assumes *inj*: $\bigwedge j.\ j \in J \Longrightarrow f'\ (f\ j) = j$
assumes *finite* J
shows ($\lambda m.$ *compose* (f ' J) $m\ f'$) $\in$ *measurable* (*PiM* J (λ-. M)) (*PiM* (f ' J) (λ-. M))
proof $-$
have ($\lambda m.$ *compose* (f ' J) $m\ f'$) $\in$ *measurable* (Pi_M (f' ' f ' J) (λ-. M)) (Pi_M (f ' J) (λ-. M))
using *assms* **by** (*auto intro*: *measurable-compose*)
moreover
from *inj* **have** f' ' f ' $J = J$ **by** (*metis* (*hide-lams*, *mono-tags*) *image-iff set-eqI*)
ultimately show *?thesis* **by** *simp*
qed

locale *function-to-finmap* =
fixes J::'a *set* **and** f :: '$a \Rightarrow$ '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 ($\lambda g.$ *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) $S \cap$ *space* ($Pi_M\ J\ M$) = ($\Pi_E\ j \in J.\ S\ (f\ j)$)
using *assms*
by (*auto simp*: *inv fm-def compose-def space-PiM Pi'-def*)

lemma *fm-measurable*:
assumes f ' $J \in N$
shows *fm* $\in$ *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*)
  apply (*rule comp-inj-on*)
  apply (*rule inj-on-compose-f'*)
  apply (*rule finmap-of-inj-on-extensional-finite*)
  apply *simp*
  apply (*auto*)
  done

lemma *fm-vimage-image-eq*:
  assumes ⋀*i*. *space* (*M i*) = *UNIV*
  assumes *X* ∈ *sets* (Pi_M *J M*)
  shows *fm* −‘ *fm* ‘ *X* ∩ *space* (Pi_M *J M*) = *X*
  using *assms*
  by (*intro inj-on-vimage-image-eq inj-on-fm*)
    (*auto simp*: *sets-into-space*)

to measure functions

definition *mf* = (λ*g*. *compose J g f*) *o proj*

lemma
  assumes *x* ∈ *space* (Pi_M *J* (λ-. *M*)) *finite J*
  shows *proj* (*finmap-of J x*) = *x*
  using *assms* **by** (*auto simp*: *space-PiM extensional-def*)

lemma
  assumes *x* ∈ *space* (Pi_F {*J*} (λ-. *M*))
  shows *finmap-of J* (*proj x*) = *x*

using *assms* **by** (*auto simp*: *space-PiF Pi′-def finmap-eq-iff*)

lemma *mf-fm*:
assumes $x \in space\ (Pi_M\ J\ (\lambda\text{-}.\ M))$
shows $mf\ (fm\ x) = x$
proof −
have $mf\ (fm\ x) \in extensional\ J$
by (*auto simp*: *mf-def extensional-def compose-def*)
moreover
have $x \in extensional\ J$ **using** *assms sets-into-space*
by (*force simp*: *space-PiM*)
moreover
{ fix *i* **assume** $i \in J$
hence $mf\ (fm\ x)\ i = x\ i$
by (*auto simp*: *inv mf-def compose-def fm-def*)
}
ultimately
show *?thesis* **by** (*rule extensionalityI*)
qed

lemma *mf-measurable*:
assumes $space\ M = UNIV$
shows $mf \in measurable\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))\ (PiM\ J\ (\lambda\text{-}.\ M))$
unfolding *mf-def*
proof (*rule measurable-comp*, *rule measurable-proj-PiM*)
show $(\lambda g.\ compose\ J\ g\ f) \in$
$measurable\ (Pi_M\ (f\ `\ J)\ (\lambda x.\ M))\ (Pi_M\ J\ (\lambda\text{-}.\ M))$
by (*rule measurable-compose*, *rule inv*) *auto*
qed (*auto simp add*: *space-PiM extensional-def assms*)

lemma *fm-image-measurable*:
assumes $space\ M = UNIV$
assumes $X \in sets\ (Pi_M\ J\ (\lambda\text{-}.\ M))$
shows $fm\ `\ X \in sets\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))$
proof −
have $fm\ `\ X = (mf) -`\ X \cap space\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))$
proof *safe*
fix *x* **assume** $x \in X$
with *mf-fm*[*of x*] *sets-into-space*[*OF assms*(*2*)] **show** $fm\ x \in mf -`\ X$ **by** *auto*
show $fm\ x \in space\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))$ **by** (*simp add*: *space-PiF assms*)
next
fix *y x*
assume *x*: $mf\ y \in X$
assume *y*: $y \in space\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))$
thus $y \in fm\ `\ X$
by (*intro image-eqI*[*OF - x*], *unfold finmap-eq-iff*)
(*auto simp*: *space-PiF fm-def mf-def compose-def inv Pi′-def*)
qed
also have $\ldots \in sets\ (PiF\ \{f\ `\ J\}\ (\lambda\text{-}.\ M))$

using *assms*
by (*intro measurable-sets*[*OF mf-measurable*]) *auto*
finally show *?thesis* **.**
qed

lemma *fm-image-measurable-finite*:
assumes *space M* = *UNIV*
assumes $X \in$ *sets* (Pi_M *J* (λ-. *M*::'*c measure*))
shows *fm* ' $X \in$ *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* = (Pi_M *J* (λ-. *N*))
assumes *space N* = *UNIV*
assumes $X \in$ *sets* (*PiF* (*Collect finite*) (λ-. *N*))
shows *emeasure* (*mapmeasure M* (λ-. *N*)) *X* = *emeasure M* ((*fm* −' $X \cap$ *extensional J*))
using *assms*
by (*auto simp*: *measurable-eqI*[*OF s1 refl s2 refl*] *mapmeasure-def emeasure-distr fm-measurable space-PiM*)

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*: $X \in$ *sets M*
shows *emeasure M X* = *emeasure* (*mapmeasure M* (λ-. *N*)) (*fm* ' *X*)
unfolding *mapmeasure-def*
proof (*subst emeasure-distr*, *subst measurable-eqI*[*OF s1 refl s2 refl*], *rule fm-measurable*)
from *fm-vimage-image-eq*[*OF* ‹*space N* = *UNIV*› *X*[*simplified s2*], *simplified s1*[*symmetric*]]
show *emeasure M X* = *emeasure M* (*fm* −' *fm* ' $X \cap$ *space M*)
by *simp*
show *fm* ' $X \in$ *sets* (*PiF* (*Collect finite*) (λ-. *N*))
by (*rule fm-image-measurable-finite*[*OF N X*[*simplified s2*]])

qed *simp*

end

end

theory *Projective-Limit*
  imports *Probability Polish-Space Fin-Map*
begin

4 Projective Limit

Formalization of the Daniell-Kolmogorov theorem.

4.1 (Finite) Product of Measures

TODO: unifiy with Pi_M

definition
  PiP I M P = extend-measure
    (Π_E *i*∈*I*. *space* (*M i*))
    {*x*. (*domain x* ≠ {} ∨ *I* = {}) ∧
     finite (*domain x*) ∧ *domain x* ⊆ *I* ∧ $(x)_F$ ∈ (Π_E *i*∈(*domain x*). *sets* (*M i*))}
    (λ*x*. *prod-emb I M* (*domain x*) (Pi_E (*domain x*) $(x)_F$))
    (λ*x*. *emeasure* (*P* (*domain x*)) (Pi_E (*domain x*) $(x)_F$))

definition *proj-algebra* **where**
  proj-algebra I M = (λ*x*. *prod-emb I M* (*domain x*) (Pi_E (*domain x*) $(x)_F$)) ‘
    {*x*. (*domain x* ≠ {} ∨ *I* = {}) ∧
      finite (*domain x*) ∧ *domain x* ⊆ *I* ∧ $(x)_F$ ∈ (Π_E *i*∈*domain x*. *sets* (*M i*))}

lemma *proj-algebra-eq-prod-algebra*:
  proj-algebra I M = *prod-algebra I M*
proof *safe*
  case *goal1* **then obtain** *X* **where** *x* = *prod-emb I M* (*domain X*) (Pi_E (*domain X*) $(X)_F$)
    domain X ≠ {} ∨ *I* = {} *finite* (*domain X*) *domain X* ⊆ *I*
    $(X)_F$ ∈ (Π_E *i*∈*domain X*. *sets* (*M i*))
    by (*auto simp*: *proj-algebra-def*)
  thus *?case* **by** (*auto simp*: *prod-algebra-def intro*!: *image-eqI*[**where** *x*=(*domain X*, $(X)_F$)])
next
  case *goal2* **then obtain** *J X* **where** *x* = *prod-emb I M J* (Pi_E *J X*)
    J ≠ {} ∨ *I* = {} *finite J J* ⊆ *I X* ∈ (Π *j*∈*J*. *sets* (*M j*))
    by (*auto simp*: *prod-algebra-def*)
  thus *?case* **by** (*auto simp*: *Pi-def proj-algebra-def intro*!: *image-eqI*[**where** *x*=*finmap-of J X*])

qed

lemma
shows *proj-algebra-eq*:
proj-algebra I M = {*prod-emb* I M J (Pi_E J F) |J F.
($J \neq \{\} \vee I = \{\}$) $\wedge$ *finite* J $\wedge$ $J \subseteq I$ $\wedge$ ($\forall i \in J$. F i $\in$ *sets* (M i))}
unfolding *proj-algebra-def*
proof (*rule*, *blast*, *rule*)
case *goal1*
then obtain J F **where** x = *prod-emb* I M J (Pi_E J F)
$J \neq \{\} \vee I = \{\}$ *finite* J $J \subseteq I$ $\bigwedge i.\ i{\in}J \Longrightarrow F\ i \in$ *sets* (M i) **by** *auto*
thus *?case* **by** (*auto intro*!: *image-eqI*[**where** x=*finmap-of* J F] *simp*: *Pi-def*)
qed

lemma *proj-algebra-eq'*:
assumes $I \neq \{\}$
shows *proj-algebra* I M =
{*prod-emb* I M J (Pi_E J F) |J F. $J \neq \{\}$ $\wedge$ *finite* J $\wedge$ $J \subseteq I$ $\wedge$ ($\forall i \in J$. F i
$\in$ *sets* (M i))}
unfolding *proj-algebra-eq*
proof (*intro antisym subsetI*)
case *goal1*
then obtain J F **where** *JF*: x = *prod-emb* I M J (Pi_E J F)
$J \neq \{\} \vee I = \{\}$ *finite* J $J \subseteq I$ $\bigwedge i.\ i{\in}J \Longrightarrow F\ i \in$ *sets* (M i) **by** *auto*
show *?case* **using** *assms JF* **by** (*auto intro*!: *exI*[**where** x=J] *exI*[**where** x=F])
qed *auto*

lemma *space-PiP*[*simp*]: *space* (*PiP* I M P) = *space* (*PiM* I M)
by (*auto simp*: *PiP-def space-PiM prod-emb-def intro*!: *space-extend-measure*)

lemma *sets-PiP'*: *sets* (*PiP* I M P) = *sigma-sets* (Π_E $i{\in}I$. *space* (M i)) (*proj-algebra*
I M)
using *prod-algebra-sets-into-space*[*of* I M, *simplified proj-algebra-eq-prod-algebra*[*symmetric*]]
unfolding *PiP-def proj-algebra-def*
by (*intro sets-extend-measure*) *simp*

lemma *sets-PiP*[*simp*]: *sets* (*PiP* I M P) = *sets* (*PiM* I M)
unfolding *sets-PiP' sets-PiM proj-algebra-eq-prod-algebra* **..**

lemma *measurable-PiP1*[*simp*]: *measurable* (*PiP* I M P) M' = *measurable* (Π_M
$i{\in}I$. M i) M'
unfolding *measurable-def* **by** *auto*

lemma *measurable-PiP2*[*simp*]: *measurable* M' (*PiP* I M P) = *measurable* M'
(Π_M $i{\in}I$. M i)
unfolding *measurable-def* **by** *auto*

4.2 Projective Family

locale *projective-family* =
  fixes I::$'i$ *set* **and** P::$'i$ *set* ⇒ $('i \Rightarrow 'a)$ *measure* **and** M::$('i \Rightarrow 'a$ *measure*$)$
  assumes *projective*: $\bigwedge J\ H.\ J \subseteq H \Longrightarrow H \subseteq I \Longrightarrow$ *finite* $H \Longrightarrow$
    $(P\ H)$ (*prod-emb* $H\ M\ J\ X) = (P\ J)\ X$
  assumes *prob-space*: $\bigwedge J.$ *prob-space* $(P\ J)$
  assumes *proj-sets*: $\bigwedge J.$ *sets* $(P\ J)$ = *sets* (*PiM* $J\ M$)
  assumes *proj-space*: $\bigwedge J.$ *space* $(P\ J)$ = *space* (*PiM* $J\ M$)
  assumes *measure-space*: $\bigwedge i.$ *prob-space* $(M\ i)$
  — TODO: generalize definitions from *product-prob-space* to *product-measure-space*

begin

lemma *measurable-P1*[*simp*]: *measurable* $(P\ J)\ M'$ = *measurable* $(\Pi_M\ i{\in}J.\ M\ i)$ M'
  unfolding *measurable-def proj-sets proj-space* **by** *auto*

lemma *measurable-P2*[*simp*]: *measurable* M' $(P\ J)$ = *measurable* M' $(\Pi_M\ i{\in}J.$ $M\ i)$
  unfolding *measurable-def proj-sets proj-space* **by** *auto*

end

sublocale *projective-family* ⊆ M: *prob-space* $M\ i$ **for** i **using** *measure-space* **.**

sublocale *projective-family* ⊆ *prob-space*: *prob-space* $P\ J$ **for** J **using** *prob-space* **.**

sublocale *projective-family* ⊆ *MP*: *product-prob-space* M **..**

context *projective-family* **begin**

lemma *emeasure-PiP*:
  assumes *finite* J
  assumes $J \subseteq I$
  assumes A: $\bigwedge i.\ i{\in}J \Longrightarrow A\ i \in$ *sets* $(M\ i)$
  shows *emeasure* (*PiP* $J\ M\ P$) $(Pi_E\ J\ A)$ = *emeasure* $(P\ J)$ $(Pi_E\ J\ A)$
proof −
  def $f \equiv$ *finmap-of* $J\ A$
  def $\mu' \equiv P\ J$
  have $Pi_E\ J$ (*restrict* $A\ J$) ⊆ $(\Pi_E\ i{\in}J.$ *space* $(M\ i))$
  proof *safe*
    fix $x\ j$ **assume** $x \in Pi\ J$ (*restrict* $A\ J$) $j \in J$
    hence $x\ j \in$ *restrict* $A\ J\ j$ **by** (*auto simp*: *Pi-def*)
    also have ... ⊆ *space* $(M\ j)$ **using** *sets-into-space* A ‹$j \in J$› **by** *auto*
    finally show $x\ j \in$ *space* $(M\ j)$ **.**
  qed
  hence *emeasure* (*PiP* $J\ M\ P$) $(Pi_E\ J\ A)$ =
    emeasure (*PiP* $J\ M\ P$) (*emb* J (*domain* f) $(Pi_E$ (*domain* f) $f))$

using *assms(1−3) sets-into-space* **by** (*auto simp add*: *f-def prod-emb-id Pi-def*)
also have ... = *emeasure* (*P J*) (Pi_E *J A*)
proof (*subst emeasure-extend-measure*[*OF PiP-def*, *of - -* μ'])
show *positive* (*sets* (*PiP J M P*)) μ' **unfolding** μ'*-def positive-def* **by** *auto*
show *countably-additive* (*sets* (*PiP J M P*)) μ' **unfolding** μ'*-def countably-additive-def*
by (*auto simp*: *suminf-emeasure proj-sets*)
show *emeasure* (*P* (*domain f*)) (Pi_E (*domain f*) *f*) = *emeasure* (*P J*) (Pi_E *J*
A)
using *assms* **by** (*simp add*: *f-def Pi-def*)
show $f \in \{x.\ (domain\ x \neq \{\} \vee J = \{\}) \wedge finite\ (domain\ x) \wedge domain\ x \subseteq J$
$\wedge$
$(x)_F \in (\Pi_E\ i{\in}domain\ x.\ sets\ (M\ i))\}$
using *assms* **by** (*auto simp*: *f-def*)
show $(\lambda x.\ emb\ J\ (domain\ x)\ (Pi_E\ (domain\ x)\ (x)_F))$ ‘ $\{x.\ (domain\ x \neq \{\} \vee$
$J = \{\}) \wedge$
$finite\ (domain\ x) \wedge domain\ x \subseteq J \wedge (x)_F \in (Pi_E\ (domain\ x)\ M)\} \subseteq$
$Pow\ (\Pi_E\ i{\in}J.\ space\ (M\ i))$ **by** (*auto simp*: *prod-emb-def*)
fix $i{::}'i \Rightarrow_F\ 'a\ set$
assume $i \in \{x.\ (domain\ x \neq \{\} \vee J = \{\}) \wedge finite\ (domain\ x) \wedge domain\ x \subseteq$
$J \wedge$
$(x)_F \in (\Pi_E\ i{\in}(domain\ x).\ sets\ (M\ i))\}$
with *assms* **have**
$finite\ (domain\ i)\ domain\ i \subseteq J\ (i)_F \in (\Pi\ i{\in}domain\ i.\ sets\ (M\ i))$
by *auto*
thus $\mu'\ (emb\ J\ (domain\ i)\ (Pi_E\ (domain\ i)\ (i)_F)) =$
$emeasure\ (P\ (domain\ i))\ (Pi_E\ (domain\ i)\ (i)_F)$
using *assms* **by** (*auto simp*: *projective* μ'*-def*)
qed
finally show *?thesis* **.**
qed

lemma *PiP-finite*:
assumes *finite J*
assumes $J \subseteq I$
shows *PiP J M P* = *P J* (**is** *?P* = -)
proof (*rule measure-eqI-generator-eq*)
interpret *J*: *finite-product-prob-space M J* **proof qed** *fact*
let $?J = \{Pi_E\ J\ E\ |\ E.\ \forall i{\in}J.\ E\ i \in sets\ (M\ i)\}$
let $?F = \lambda i.\ \Pi_E\ k{\in}J.\ space\ (M\ k)$
let $?\Omega = (\Pi_E\ k{\in}J.\ space\ (M\ k))$
show *Int-stable ?J*
by (*rule Int-stable-PiE*)
show *emeasure ?P* (*?F* -) $\neq \infty$ **using** *assms* ‹*finite J*› **by** (*auto simp*: *emeasure-PiP*)
show $?J \subseteq Pow\ ?\Omega$ **by** (*auto simp*: *Pi-iff dest*: *sets-into-space*)
show *sets* (*PiP J M P*) = *sigma-sets* $?\Omega$ *?J sets* (*P J*) = *sigma-sets* $?\Omega$ *?J*
using ‹*finite J*› *proj-sets* **by** (*simp-all add*: *sets-PiM prod-algebra-eq-finite Pi-iff*)
fix *X* **assume** $X \in ?J$
then obtain *E* **where** [*simp*]: $X = Pi_E\ J\ E$ **and** *E*: $\forall i{\in}J.\ E\ i \in sets\ (M\ i)$
by *auto*

with ‹*finite J*› **have** X: $X \in$ *sets* (*PiP J M P*) **by** *auto*
show *emeasure* (*PiP J M P*) X = *emeasure* (*P J*) X **using** *assms* ‹*finite J*› E
by (*auto simp*: *emeasure-PiP*)
qed (*insert* ‹*finite J*›, *auto intro*!: *prod-algebraI-finite*)

lemma *emeasure-fun-emb*[*simp*]:
assumes L: $J \subseteq L$ *finite* L $L \subseteq I$ **and** X: $X \in$ *sets* (*PiP J M P*)
shows *emeasure* (*PiP L M P*) (*emb L J X*) = *emeasure* (*PiP J M P*) X
using *assms*
by (*subst PiP-finite*) (*auto simp*: *PiP-finite finite-subset projective*)

lemma *distr-restrict*:
assumes $J \subseteq K$ *finite* K $K \subseteq I$
shows (*PiP J M P*) = *distr* (*PiP K M P*) (*PiP J M P*) (λf. *restrict f J*) (**is** *?P* = *?D*)
proof (*rule measure-eqI*)
show *sets* (*PiP J M P*) = *sets* (*distr* (*PiP K M P*) (*PiP J M P*) (λf. *restrict f J*)) **by** *simp*
fix A **assume** $A \in$ *sets* (*PiP J M P*)
with *assms* **show** *emeasure* (*PiP J M P*) A =
emeasure (*distr* (*PiP K M P*) (*PiP J M P*) (λf. *restrict f J*)) A
by (*auto simp*: *emeasure-distr measurable-restrict-subset space-PiM prod-emb-def* [*symmetric*])
qed

4.3 Content on Generator

definition
$\mu G'$ A =
(*THE* x. $\forall J$. $J \neq \{\} \longrightarrow$ *finite* $J \longrightarrow J \subseteq I \longrightarrow$
($\forall X \in$*sets* (*PiP J M P*). A = *emb I J X* $\longrightarrow x$ = *emeasure* (*PiP J M P*) X))

lemma $\mu G'$*-spec*:
assumes J: $J \neq \{\}$ *finite* J $J \subseteq I$ A = *emb I J X* $X \in$ *sets* (*PiP J M P*)
shows $\mu G'$ A = *emeasure* (*PiP J M P*) X
unfolding $\mu G'$*-def*
proof (*intro the-equality allI impI ballI*)
fix K Y **assume** K: $K \neq \{\}$ *finite* K $K \subseteq I$ A = *emb I K Y* $Y \in$ *sets* (*PiP K M P*)
have *emeasure* (*PiP K M P*) Y = *emeasure* (*PiP* ($K \cup J$) *M P*) (*emb* ($K \cup J$) K Y)
using K J **by** *simp*
also have *emb* ($K \cup J$) K Y = *emb* ($K \cup J$) J X
using K J **by** (*simp add*: *prod-emb-injective*[*of* $K \cup J$ I])
also have *emeasure* (*PiP* ($K \cup J$) *M P*) (*emb* ($K \cup J$) J X) = *emeasure* (*PiP J M P*) X
using K J **by** *simp*
finally show *emeasure* (*PiP J M P*) X = *emeasure* (*PiP K M P*) Y **..**
qed (*insert* J, *force*)

lemma $\mu G'$-*eq*:
$J \neq \{\} \Longrightarrow$ *finite* $J \Longrightarrow J \subseteq I \Longrightarrow X \in$ *sets* (*PiP J M P*) $\Longrightarrow$
$\mu G'$ (*emb I J X*) = *emeasure* (*PiP J M P*) X
by (*intro* $\mu G'$-*spec*) *auto*

lemma *generator-Ex'*:
assumes $*$: $A \in$ *generator*
shows $\exists J\ X.\ J \neq \{\} \wedge$ *finite* $J \wedge J \subseteq I \wedge X \in$ *sets* (Π_M $i{\in}J.$ M i) $\wedge A =$ *emb I J X* $\wedge$
$\mu G'$ A = *emeasure* (*PiP J M P*) X
proof −
from $*$ **obtain** J X **where** J: $J \neq \{\}$ *finite* J $J \subseteq I$ A = *emb I J X* $X \in$ *sets* (*PiP J M P*)
unfolding *generator-def* **by** *auto*
with $\mu G'$-*spec*[*OF this*] **show** *?thesis* **by** *auto*
qed

lemma *generatorE'*:
assumes A: $A \in$ *generator*
obtains J X **where** $J \neq \{\}$ *finite* J $J \subseteq I$ $X \in$ *sets* (*PiP J M P*) *emb I J X* = A
$\mu G'$ A = *emeasure* (*PiP J M P*) X
proof −
from *generator-Ex'*[*OF A*] **obtain** X J **where** $J \neq \{\}$ *finite* J $J \subseteq I$ $X \in$ *sets* (*PiP J M P*)
emb I J X = A $\mu G'$ A = *emeasure* (*PiP J M P*) X **by** *auto*
then show *thesis* **by** (*intro that*) *auto*
qed

lemma *positive*-$\mu G'$:
assumes $I \neq \{\}$
shows *positive generator* $\mu G'$
proof −
interpret $G!$: *algebra* Π_E $i{\in}I.$ *space* (M i) *generator* **by** (*rule algebra-generator*) *fact*
show *?thesis*
proof (*intro positive-def*[*THEN iffD2*] *conjI ballI*)
from *generatorE'*[*OF G.empty-sets*] **guess** J X . **note** *this*[*simplified*, *simp*]
interpret J: *finite-product-sigma-finite M J* **by** *default fact*
have $X = \{\}$
by (*rule prod-emb-injective*[*of J I*]) *simp-all*
then show $\mu G'$ $\{\}$ = 0 **by** *simp*
next
fix A **assume** $A \in$ *generator*
from *generatorE'*[*OF this*] **guess** J X . **note** *this*[*simp*]
interpret J: *finite-product-sigma-finite M J* **by** *default fact*
show $0 \leq \mu G'$ A **by** (*simp add*: *emeasure-nonneg*)
qed
qed

lemma *additive-*$\mu G'$:
  assumes $I \neq \{\}$
  shows *additive generator* $\mu G'$
proof −
  interpret $G!$: *algebra* Π_E $i{\in}I$. *space* $(M\ i)$ *generator* **by** (*rule algebra-generator*) *fact*
  show *?thesis*
  proof (*intro additive-def*[*THEN iffD2*] *ballI impI*)
    fix A **assume** $A \in$ *generator* **with** *generatorE'* **guess** $J\ X$ **. note** $J = this$
    fix B **assume** $B \in$ *generator* **with** *generatorE'* **guess** $K\ Y$ **. note** $K = this$
    assume $A \cap B = \{\}$
    have JK: $J \cup K \neq \{\}$ $J \cup K \subseteq I$ *finite* $(J \cup K)$
      using $J\ K$ **by** *auto*
    interpret JK: *finite-product-sigma-finite* $M\ J \cup K$ **by** *default fact*
    have *JK-disj*: *emb* $(J \cup K)$ $J\ X \cap$ *emb* $(J \cup K)$ $K\ Y = \{\}$
      apply (*rule prod-emb-injective*[*of* $J \cup K\ I$])
      apply (*insert* ‹$A \cap B = \{\}$› $JK\ J\ K$)
      apply (*simp-all add*: *Int prod-emb-Int*)
      done
    have AB: $A =$ *emb* I $(J \cup K)$ (*emb* $(J \cup K)$ $J\ X$) $B =$ *emb* I $(J \cup K)$ (*emb* $(J \cup K)$ $K\ Y$)
      using $J\ K$ **by** *simp-all*
    then have $\mu G'$ $(A \cup B) = \mu G'$ (*emb* I $(J \cup K)$ (*emb* $(J \cup K)$ $J\ X \cup$ *emb* $(J \cup K)$ $K\ Y$))
      by *simp*
    also have $\ldots =$ *emeasure* (*PiP* $(J \cup K)$ $M\ P$) (*emb* $(J \cup K)$ $J\ X \cup$ *emb* $(J \cup K)$ $K\ Y$)
      using $JK\ J(1, 4)\ K(1, 4)$ **by** (*simp add*: $\mu G'$*-eq Un del*: *prod-emb-Un*)
    also have $\ldots = \mu G'\ A + \mu G'\ B$
      using $J\ K$ *JK-disj* **by** (*simp add*: *plus-emeasure*[*symmetric*])
    finally show $\mu G'$ $(A \cup B) = \mu G'\ A + \mu G'\ B$ **.**
  qed
qed

end

4.4 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

sublocale *finmap-seqs-into-compact* $\subseteq$ *subseqs* $\lambda n\ s\ r.\ (\exists l. (\lambda i. ((f\ o\ s\ o\ r)\ i)_F$ $n)$ ––––> $l)$
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* $r' \wedge (\exists l. (\lambda i. ((f \circ s \circ r')\ i)_F\ n)$ ––––> $l)$ **by** (*auto simp*: *o-def*)
qed

lemma (**in** *finmap-seqs-into-compact*)
  diagonal-tendsto: $\exists l. (\lambda i. (f\ (diagseq\ i))_F\ n)$ ––––> l
proof –
  have $\bigwedge i\ n0.\ (f\ o\ seqseq\ i)\ i = f\ (diagseq\ i)$ **unfolding** *diagseq-def* **by** *simp*
   from *reducer-reduces* **obtain** l **where** l: $(\lambda i. ((f \circ seqseq\ (Suc\ n))\ i)_F\ n)$ ––––> l
    unfolding *seqseq-reducer*
  by *auto*
  have $(\lambda i. (f\ (diagseq\ (i + Suc\ n)))_F\ n) =$
    $(\lambda i. ((f\ o\ (diagseq\ o\ (op + (Suc\ n))))\ i)_F\ n)$ **by** (*simp add*: *add-commute*)

also have ... =
$(\lambda i.\ ((f\ o\ ((seqseq\ (Suc\ n)\ o\ (\lambda x.\ fold\text{-}reduce\ (Suc\ n)\ x\ (Suc\ n + x))))) \ i)_F\ n)$
unfolding *diagseq-seqseq* **by** *simp*
also have ... $= (\lambda i.\ ((f\ o\ ((seqseq\ (Suc\ n)))) \ i)_F\ n)\ o\ (\lambda x.\ fold\text{-}reduce\ (Suc\ n)$
$x\ (Suc\ n + x))$
by (*simp add*: *o-def*)
also have ... $----> l$
proof (*rule LIMSEQ-subseq-LIMSEQ*[*OF - subseq-diagonal-rest*], *rule tendstoI*)
fix *e*::*real* **assume** $0 < e$
from *tendstoD*[*OF l* ‹$0 < e$›]
show *eventually* $(\lambda x.\ dist\ (((f \circ seqseq\ (Suc\ n))\ x)_F\ n)\ l < e)$
sequentially **.**
qed
finally show *?thesis* **by** (*intro exI*) (*rule LIMSEQ-offset*)
qed

4.5 The Daniell-Kolmogorov theorem

locale *polish-projective* = *projective-family I P* λ-. *borel*::'*a*::*polish-space measure*
for *I*::'*i set* **and** *P*
begin

abbreviation $PiB \equiv (\lambda J\ P.\ PiP\ J\ (\lambda\text{-}.\ borel)\ P)$

lemma
emeasure-PiB-emb-not-empty:
assumes $I \neq \{\}$
assumes X: $J \neq \{\}\ J \subseteq I$ *finite J* $\forall i{\in}J.\ B\ i \in sets\ borel$
shows *emeasure* $(PiB\ I\ P)\ (emb\ I\ J\ (Pi_E\ J\ B)) =$ *emeasure* $(PiB\ J\ P)\ (Pi_E\ J$
$B)$
proof −
let $?\Omega = \Pi_E\ i{\in}I.\ space\ borel$
let $?G = generator$
interpret *G*!: *algebra* $?\Omega$ *generator* **by** (*intro algebra-generator*) *fact*
note $\mu G'$-*mono* =
G.additive-increasing[*OF positive-*$\mu G'$[*OF* ‹$I \neq \{\}$›] *additive-*$\mu G'$[*OF* ‹$I \neq \{\}$›],
THEN increasingD]
have $\exists \mu.\ (\forall s{\in}?G.\ \mu\ s = \mu G'\ s) \wedge$ *measure-space* $?\Omega$ (*sigma-sets* $?\Omega\ ?G)\ \mu$
proof (*rule G.caratheodory-empty-continuous*[*OF positive-*$\mu G'$ *additive-*$\mu G'$,
OF ‹$I \neq \{\}$›, *OF* ‹$I \neq \{\}$›])
fix *A* **assume** $A \in ?G$
with *generatorE'* **guess** *J X* **.**
thus $\mu G'\ A \neq \infty$ **by** (*simp add*: *PiP-finite*)
next
fix *Z* **assume** *Z*: *range* $Z \subseteq ?G$ *decseq Z* $(\bigcap i.\ Z\ i) = \{\}$
then have *decseq* $(\lambda i.\ \mu G'\ (Z\ i))$
by (*auto intro*!: $\mu G'$-*mono simp*: *decseq-def*)
moreover
have $(INF\ i.\ \mu G'\ (Z\ i)) = 0$

```
  proof (rule ccontr)
    assume (INF i. μG' (Z i)) ≠ 0 (is ?a ≠ 0)
    moreover have 0 ≤ ?a
      using Z positive-μG'[OF ‹I ≠ {}›] by (auto intro!: INF-greatest simp:
positive-def)
    ultimately have 0 < ?a by auto
    hence ?a ≠ −∞ by auto
    have ∀n. ∃J B. J ≠ {} ∧ finite J ∧ J ⊆ I ∧ B ∈ sets (Pi_M J (λ-. borel)) ∧
      Z n = emb I J B ∧ μG' (Z n) = emeasure (PiB J P) B
      using Z by (intro allI generator-Ex') auto
    then obtain J' B' where J': ⋀n. J' n ≠ {} ⋀n. finite (J' n) ⋀n. J' n ⊆ I
        ⋀n. B' n ∈ sets (Π_M i∈J' n. borel)
      and Z-emb: ⋀n. Z n = emb I (J' n) (B' n)
      unfolding choice-iff by blast
    moreover def J ≡ λn. (⋃i≤n. J' i)
    moreover def B ≡ λn. emb (J n) (J' n) (B' n)
    ultimately have J: ⋀n. J n ≠ {} ⋀n. finite (J n) ⋀n. J n ⊆ I
      ⋀n. B n ∈ sets (Π_M i∈J n. borel)
      by auto
    have J-mono: ⋀n m. n ≤ m ⟹ J n ⊆ J m
      unfolding J-def by force
    have ∀n. ∃j. j ∈ J n using J by blast
    then obtain j where j: ⋀n. j n ∈ J n
      unfolding choice-iff by blast
    note [simp] = ‹⋀n. finite (J n)›
    from J Z-emb have Z-eq: ⋀n. Z n = emb I (J n) (B n) ⋀n. Z n ∈ ?G
      unfolding J-def B-def by (subst prod-emb-trans) (insert Z, auto)

    have ?a ≤ μG' (Z 0) by (auto intro: INF-lower)
    also have ... < ∞ using J by (auto simp: Z-eq μG'-eq PiP-finite proj-sets)
    finally have ?a ≠ ∞ by simp
    have ⋀n. |μG' (Z n)| ≠ ∞ unfolding Z-eq using J J-mono
      by (subst μG'-eq) (auto simp: PiP-finite proj-sets μG'-eq)

    interpret finite-set-sequence J by unfold-locales simp
    def Utn ≡ Un-to-nat
    interpret function-to-finmap J n Utn inv-into (J n) Utn for n
      by unfold-locales (auto simp: Utn-def)
    def P' ≡ λn. mapmeasure n (P (J n)) (λ-. borel)
    let ?SUP = λn. SUP K : {K. K ⊆ fm n ‘ (B n) ∧ compact K}. emeasure
(P' n) K
    {
      fix n
      interpret finite-measure P (J n) by unfold-locales
      have emeasure (P (J n)) (B n) = emeasure (P' n) (fm n ‘ (B n))
        using J
        by (auto simp: P'-def mapmeasure-PiM proj-space proj-sets)
      also
      have ... = ?SUP n
```

proof (*rule inner-regular*)
show *emeasure* (P' *n*) (*space* (P' *n*)) $\neq \infty$
unfolding P'*-def*
by (*auto simp*: P'*-def mapmeasure-PiF fm-measurable proj-space proj-sets*)
show *sets* (P' *n*) = *sets borel* **by** (*simp add*: *borel-eq-PiF-borel* P'*-def*)
next
show *fm n ‘ B n* $\in$ *sets borel*
unfolding *borel-eq-PiF-borel*
by (*auto simp del*: *J*(*2*) *simp*: P'*-def fm-image-measurable-finite proj-sets J*)
qed
finally
have *emeasure* (*P* (*J n*)) (*B n*) = *?SUP n ?SUP n* $\neq \infty$ *?SUP n* $\neq - \infty$ **by** *auto*
} **note** *R* = *this*
have $\forall$ *n*. $\exists$ *K*. *emeasure* (*P* (*J n*)) (*B n*) $-$ *emeasure* (P' *n*) *K* $\leq$ *2 powr* ($-n$) $*$ *?a*
$\wedge$ *compact K* $\wedge$ *K* $\subseteq$ *fm n ‘ B n*
proof
fix *n*
have *emeasure* (P' *n*) (*space* (P' *n*)) $\neq \infty$
by (*simp add*: *mapmeasure-PiF* P'*-def proj-space proj-sets*)
then interpret *finite-measure* P' *n* **..**
show $\exists$ *K*. *emeasure* (*P* (*J n*)) (*B n*) $-$ *emeasure* (P' *n*) *K* $\leq$ *ereal* (*2 powr* $-$ *real n*) $*$ *?a* $\wedge$
compact K $\wedge$ *K* $\subseteq$ *fm n ‘ B n*
unfolding *R*
proof (*rule ccontr*)
assume *H*: $\neg$ ($\exists$ K'. *?SUP n* $-$ *emeasure* (P' *n*) K' $\leq$ *ereal* (*2 powr* $-$ *real n*) $*$ *?a* $\wedge$
compact K' $\wedge$ K' $\subseteq$ *fm n ‘ B n*)
have *?SUP n* $\leq$ *?SUP n* $-$ *2 powr* ($-n$) $*$ *?a*
proof (*intro SUP-least*)
fix *K*
assume *K* $\in$ {*K*. *K* $\subseteq$ *fm n ‘ B n* $\wedge$ *compact K*}
with *H* **have** $\neg$ *?SUP n* $-$ *emeasure* (P' *n*) *K* $\leq$ *2 powr* ($-n$) $*$ *?a*
by *auto*
hence *?SUP n* $-$ *emeasure* (P' *n*) *K* $>$ *2 powr* ($-n$) $*$ *?a*
unfolding *not-less*[*symmetric*] **by** *simp*
hence *?SUP n* $-$ *2 powr* ($-n$) $*$ *?a* $>$ *emeasure* (P' *n*) *K*
using ‹*0* $<$ *?a*› **by** (*auto simp add*: *ereal-less-minus-iff ac-simps*)
thus *?SUP n* $-$ *2 powr* ($-n$) $*$ *?a* $\geq$ *emeasure* (P' *n*) *K* **by** *simp*
qed
hence *?SUP n* + *0* $\leq$ *?SUP n* $-$ (*2 powr* ($-n$) $*$ *?a*) **using** ‹*0* $<$ *?a*› **by** *simp*
hence *?SUP n* + *0* $\leq$ *?SUP n* + $-$ (*2 powr* ($-n$) $*$ *?a*) **unfolding** *minus-ereal-def* **.**
hence *0* $\leq$ $-$ (*2 powr* ($-n$) $*$ *?a*)
using ‹*?SUP* - $\neq \infty$› ‹*?SUP* - $\neq - \infty$›

by (*subst* (*asm*) *ereal-add-le-add-iff*) (*auto simp*:)
moreover have *ereal* (*2 powr* $-$ *real n*) $*$ *?a* $>$ *0* **using** ‹*0* $<$ *?a*›
by (*auto simp*: *ereal-zero-less-0-iff*)
ultimately show *False* **by** *simp*
qed
qed
then obtain K' **where** K':
$\bigwedge n$. *emeasure* (*P* (*J n*)) (*B n*) $-$ *emeasure* (P' *n*) (K' *n*) $\leq$ *ereal* (*2 powr* $-$ *real n*) $*$ *?a*
$\bigwedge n$. *compact* (K' *n*) $\bigwedge n$. K' *n* $\subseteq$ *fm n* ‘ *B n*
unfolding *choice-iff* **by** *blast*
def *K* $\equiv$ λn. *fm n* $-$‘ K' *n* $\cap$ *space* (*P* (*J n*))
have *K-sets*: $\bigwedge n$. *K n* $\in$ *sets* (Pi_M (*J n*) (λ-. *borel*))
unfolding *K-def proj-space*
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 $\bigwedge n$. *K n* $\subseteq$ *B n*
proof
fix *x n*
assume *x* $\in$ *K n* **hence** *fm-in*: *fm n x* $\in$ *fm n* ‘ *B n*
using K' **by** (*force simp*: *K-def*)
show *x* $\in$ *B n*
apply (*rule inj-on-image-mem-iff*[*OF inj-on-fm - fm-in*])
using ‹*x* $\in$ *K n*› *K-sets J*[*of n*] *sets-into-space*
apply (*auto simp*: *proj-space*)
using *J*[*of n*] *sets-into-space* **apply** *auto*
done
qed
def Z' $\equiv$ λn. *emb I* (*J n*) (*K n*)
have Z': $\bigwedge n$. Z' *n* $\subseteq$ *Z n*
unfolding *Z-eq* **unfolding** Z'*-def*
proof (*rule prod-emb-subsetI*, *safe*)
fix *n x* **assume** *x* $\in$ *K n*
hence *fm n x* $\in$ K' *n x* $\in$ *space* (Pi_M (*J n*) (λ-. *borel*))
by (*simp-all add*: *K-def proj-space*)
note *this*(*1*)
also have K' *n* $\subseteq$ *fm n* ‘ *B n* **by** (*simp add*: K')
finally have *fm n x* $\in$ *fm n* ‘ *B n* **.**
thus *x* $\in$ *B n*
proof *safe*
fix *y* **assume** *y* $\in$ *B n*
moreover
hence *y* $\in$ *space* (Pi_M (*J n*) (λ-. *borel*)) **using** *J sets-into-space*[*of B n P* (*J n*)]
by (*auto simp add*: *proj-space proj-sets*)
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* $\in$ *space* -› ‹*y* $\in$ *space* -›]

ultimately show $x \in B\ n$ **by** *simp*
qed
qed
{ fix n
have $Z'\ n \in\ ?G$ **using** K' **unfolding** Z'*-def*
apply (*intro generatorI'*[*OF* $J(1-3)$])
unfolding *K-def proj-space*
apply (*rule measurable-sets*[*OF fm-measurable*[*of - Collect finite*]])
apply (*auto simp add*: P'*-def borel-eq-PiF-borel*[*symmetric*] *compact-imp-closed*)
done
}
def $Y \equiv \lambda n.\ \bigcap i{\in}\{1..n\}.\ Z'\ i$
hence $\bigwedge n\ k.\ Y\ (n + k) \subseteq Y\ n$ **by** (*induct-tac k*) (*auto simp*: *Y-def*)
hence *Y-mono*: $\bigwedge n\ m.\ n \le m \Longrightarrow Y\ m \subseteq Y\ n$ **by** (*auto simp*: *le-iff-add*)
have *Y-Z'*: $\bigwedge n.\ n \ge 1 \Longrightarrow Y\ n \subseteq Z'\ n$ **by** (*auto simp*: *Y-def*)
hence *Y-Z*: $\bigwedge n.\ n \ge 1 \Longrightarrow Y\ n \subseteq Z\ n$ **using** Z' **by** *auto*
have *Y-notempty*: $\bigwedge n.\ n \ge 1 \Longrightarrow (Y\ n) \neq \{\}$
proof −
fix n::*nat* **assume** $n \ge 1$ **hence** $Y\ n \subseteq Z\ n$ **by** *fact*
have $Y\ n = (\bigcap\ i{\in}\{1..n\}.\ emb\ I\ (J\ n)\ (emb\ (J\ n)\ (J\ i)\ (K\ i)))$ **using** J *J-mono*
by (*auto simp*: *Y-def* Z'*-def*)
also have $\ldots = $ *prod-emb* I (λ-. *borel*) $(J\ n)$ $(\bigcap\ i{\in}\{1..n\}.\ emb\ (J\ n)\ (J\ i)\ (K\ i))$
using $\langle n \ge 1\rangle$
by (*subst prod-emb-INT*) *auto*
finally
have *Y-emb*:
$Y\ n = $ *prod-emb* I (λ-. *borel*) $(J\ n)$
$(\bigcap\ i{\in}\{1..n\}.$ *prod-emb* $(J\ n)$ (λ-. *borel*) $(J\ i)\ (K\ i))$ **.**
hence $Y\ n \in\ ?G$ **using** J *J-mono K-sets* $\langle n \ge 1\rangle$ **by** (*intro generatorI*[*OF* - - - - *Y-emb*]) *auto*
hence $|\mu G'\ (Y\ n)| \neq \infty$ **unfolding** *Y-emb* **using** J *J-mono K-sets* $\langle n \ge 1\rangle$
by (*subst* $\mu G'$*-eq*) (*auto simp*: *PiP-finite proj-sets* $\mu G'$*-eq*)
interpret *finite-measure* (*PiP* $(J\ n)$ (λ-. *borel*) P)
proof
have *emeasure* (*PiP* $(J\ n)$ (λ-. *borel*) P) $(J\ n \to_E$ *space borel*$) \neq \infty$
using J **by** (*subst emeasure-PiP*) *auto*
thus *emeasure* (*PiP* $(J\ n)$ (λ-. *borel*) P) (*space* (*PiP* $(J\ n)$ (λ-. *borel*) P)) $\neq \infty$
by (*simp add*: *space-PiM*)
qed
have $\mu G'\ (Z\ n) - \mu G'\ (Y\ n) = \mu G'\ (Z\ n - Y\ n)$ **using** J *J-mono K-sets* $\langle n \ge 1\rangle$
apply (*intro G.subtractive*[*OF positive-*$\mu G'$ *additive-*$\mu G'$,
OF $\langle I \neq \{\}\rangle$ $\langle I \neq \{\}\rangle$ $\langle Y\ n \in\ ?G\rangle$ $\langle Z\ n \in\ ?G\rangle$ $\langle Y\ n \subseteq Z\ n\rangle$, *symmetric*])
apply (*subst* $\mu G'$*-spec*[*OF* $\langle J\ n \neq \{\}\rangle$ $\langle$*finite* $(J\ n)\rangle$ $\langle J\ n \subseteq I\rangle$ *Y-emb*])
apply *auto* **done**
also have *subs*: $Z\ n - Y\ n \subseteq (\bigcup\ i{\in}\{1..n\}.\ (Z\ i - Z'\ i))$ **using** Z' Z $\langle n$

≥ 1›
    unfolding Y-def
    apply (auto simp: decseq-def Y-def)
   proof –
    case goal1 hence x ∈ Z xa by (metis set-mp)
    with goal1 show x ∈ Z′ xa by auto
   qed
   have Z n − Y n ∈ ?G (⋃ i∈{1..n}. (Z i − Z′ i)) ∈ ?G
    using ‹Z′ - ∈ ?G› ‹Z - ∈ ?G› ‹Y - ∈ ?G› by auto
   hence μG′ (Z n − Y n) ≤ μG′ (⋃ i∈{1..n}. (Z i − Z′ i))
   using subs G.additive-increasing[OF positive-μG′[OF ‹I ≠ {}›] additive-μG′[OF ‹I ≠ {}›]]
    unfolding increasing-def by auto
   also have ... ≤ (∑ i∈{1..n}. μG′ (Z i − Z′ i)) using ‹Z - ∈ ?G› ‹Z′ - ∈ ?G›
    by (intro G.subadditive[OF positive-μG′ additive-μG′, OF ‹I ≠ {}› ‹I ≠ {}›]) auto
   also have ... ≤ (∑ i∈{1..n}. 2 powr −real i ∗ ?a)
   proof (rule setsum-mono)
    fix i assume i ∈ {1..n} hence i ≤ n by simp
    have μG′ (Z i − Z′ i) = μG′ (prod-emb I (λ-. borel) (J i) (B i − K i))
     unfolding Z′-def Z-eq by simp
    also have ... = P (J i) (B i − K i)
     apply (subst μG′-eq) using J K-sets apply auto
     apply (subst PiP-finite) apply auto
     done
    also have ... = P (J i) (B i) − P (J i) (K i)
     apply (subst emeasure-Diff) using K-sets J ‹K - ⊆ B -› apply (auto simp: proj-sets)
     done
    also have ... = P (J i) (B i) − P′ i (K′ i)
     unfolding K-def P′-def
   by (auto simp: mapmeasure-PiF proj-space proj-sets borel-eq-PiF-borel[symmetric]
     compact-imp-closed[OF ‹compact (K′ -)›] space-PiM)
    also have ... ≤ ereal (2 powr − real i) ∗ ?a using K′(1)[of i] .
    finally show μG′ (Z i − Z′ i) ≤ (2 powr − real i) ∗ ?a .
   qed
   also have ... = (∑ i∈{1..n}. ereal (2 powr −real i) ∗ ereal(real ?a))
    using ‹?a ≠ ∞› ‹?a ≠ − ∞› by (subst ereal-real′) auto
   also have ... = ereal (∑ i∈{1..n}. (2 powr −real i) ∗ (real ?a)) by simp
   also have ... = ereal ((∑ i∈{1..n}. (2 powr −real i)) ∗ real ?a)
    by (simp add: setsum-left-distrib)
   also have ... < ereal (1 ∗ real ?a) unfolding less-ereal.simps
   proof (rule mult-strict-right-mono)
    have (∑ i∈{1..n}. 2 powr − real i) = (∑ i∈{1..<Suc n}. (1/2) ^ i)
     by (rule setsum-cong)
     (auto simp: powr-realpow[symmetric] powr-minus powr-divide inverse-eq-divide)
    also have {1..<Suc n} = {0..<Suc n} − {0} by auto
    also have setsum (op ^ (1 / 2::real)) ({0..<Suc n} − {0}) =

setsum (*op* ^ (*1* / *2*)) ({*0*..<*Suc n*}) − *1* **by** (*auto simp*: *setsum-diff1*)
also have ... < *1* **by** (*subst sumr-geometric*) *auto*
finally show ($\sum i = 1..n.\ 2\ powr - real\ i$) < *1* **.**
qed (*auto simp*:
‹$0 < ?a$› ‹$?a \neq \infty$› ‹$?a \neq - \infty$› *ereal-less-real-iff zero-ereal-def*[*symmetric*])
also have ... = *?a* **using** ‹$0 < ?a$› ‹$?a \neq \infty$› **by** (*auto simp*: *ereal-real'*)
also have ... $\leq \mu G'$ (*Z n*) **by** (*auto intro*: *INF-lower*)
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** (*simp add*: *ereal-minus-less*)
have $0 \leq$ (− $\mu G'$ (*Z n*)) + $\mu G'$ (*Z n*) **using** ‹$|\mu G'$ (*Z n*)$| \neq \infty$› **by** *auto*
also have ... < (− $\mu G'$ (*Z n*)) + ($\mu G'$ (*Z n*) + $\mu G'$ (*Y n*))
apply (*rule ereal-less-add*[*OF* - *R*]) **using** ‹$|\mu G'$ (*Z n*)$| \neq \infty$› **by** *auto*
finally have $\mu G'$ (*Y n*) > *0*
using ‹$|\mu G'$ (*Z n*)$| \neq \infty$› **by** (*auto simp*: *ac-simps zero-ereal-def*[*symmetric*])
thus *Y n* ≠ {} **using** *positive*-$\mu G'$ ‹*I* ≠ {}› **by** (*auto simp add*: *positive-def*)
qed
hence ∀ *n*∈{*1*..}. ∃ *y*. *y* ∈ *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* ∈ *Y n* **by** *auto*
also have ... ⊆ *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 ... ⊆ *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-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 ≤ 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 x assume i ≥ n t ∈ domain (fm n x)
            moreover
            hence t ∈ domain (fm i x) using J-mono[OF ‹i ≥ n›] by auto
            ultimately have (fm i x)_F t = (fm n x)_F t
              using j by (auto simp: proj-fm dest!:
                Un-to-nat-injectiveD[simplified Utn-def[symmetric]])
          } 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: $\bigwedge i.\ i \geq N \Longrightarrow$
dist $((fm\ (Suc\ (diagseq\ i))\ (y\ (Suc\ (diagseq\ i))))_F\ t)\ (z\ t) < e$ **by** *auto*
show $\exists N.\ \forall na \geq N.\ dist\ ((fm\ n\ (y\ (Suc\ (diagseq\ na))))_F\ t)\ (z\ t) < e$
proof (*rule exI*[**where** x=*max N n*], *safe*)
fix *na* **assume** *max N n* $\leq$ *na*
hence $dist\ ((fm\ n\ (y\ (Suc\ (diagseq\ na))))_F\ t)\ (z\ t) =$
$dist\ ((fm\ (Suc\ (diagseq\ na))\ (y\ (Suc\ (diagseq\ na))))_F\ t)\ (z\ t)$
using t
by (*subst index-shift*[*OF I*]) *auto*
also have $\ldots < e$ **using** ‹*max N n* $\leq$ *na*› **by** (*intro N*) *simp*
finally show $dist\ ((fm\ n\ (y\ (Suc\ (diagseq\ na))))_F\ t)\ (z\ t) < e$ **.**
qed
qed
hence $(\lambda i.\ (fm\ n\ (y\ (Suc\ (diagseq\ i))))_F\ t)$ ––––> $(finmap\text{-}of\ (Utn\ `\ J\ n)\ z)_F\ t$
by (*simp add*: *tendsto-intros*)
} **ultimately**
have $(\lambda i.\ fm\ n\ (y\ (Suc\ (diagseq\ i))))$ ––––> *finmap-of* (*Utn* ‘ *J n*) *z*
by (*rule tendsto-finmap*)
hence $((\lambda i.\ fm\ n\ (y\ (Suc\ (diagseq\ i))))\ o\ (\lambda i.\ i + n))$ ––––> *finmap-of* (*Utn* ‘ *J n*) *z*
by (*intro lim-subseq*) (*simp add*: *subseq-def*)
moreover
have $(\forall i.\ ((\lambda i.\ fm\ n\ (y\ (Suc\ (diagseq\ i))))\ o\ (\lambda i.\ i + n))\ i \in K'\ n)$
apply (*auto simp add*: *o-def intro*!: *fm-in-K'* ‹$1 \leq n$› *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*) $z \in K'\ n$
unfolding *closed-sequential-limits* **by** *blast*
also have *finmap-of* (*Utn* ‘ *J n*) $z = fm\ n\ (\lambda i.\ z\ (Utn\ i))$
by (*auto simp*: *finmap-eq-iff fm-def compose-def f-inv-into-f*)
finally have $fm\ n\ (\lambda i.\ z\ (Utn\ i)) \in K'\ n$ **.**
moreover
let $?J = \bigcup n.\ J\ n$
have $(?J \cap J\ n) = J\ n$ **by** *auto*
ultimately have $restrict\ (\lambda i.\ z\ (Utn\ i))\ (?J \cap J\ n) \in K\ n$
unfolding *K-def* **by** (*auto simp*: *proj-space space-PiM*)
hence $restrict\ (\lambda i.\ z\ (Utn\ i))\ ?J \in Z'\ n$ **unfolding** *Z'-def*
using J **by** (*auto simp*: *prod-emb-def extensional-def*)
also have $\ldots \subseteq Z\ n$ **using** Z' **by** *simp*
finally have $restrict\ (\lambda i.\ z\ (Utn\ i))\ ?J \in Z\ n$ **.**

} **note** *in-Z* = *this*
hence $(\bigcap i{\in}\{1..\}.\ Z\ i) \neq \{\}$ **by** *auto*
hence $(\bigcap i.\ Z\ i) \neq \{\}$ **using** *Z INT-decseq-offset*[*OF* ‹*decseq Z*›] **by** *simp*
thus *False* **using** *Z* **by** *simp*
qed
ultimately show $(\lambda i.\ \mu G'\ (Z\ i)) \text{ ----> } 0$
using *LIMSEQ-ereal-INFI*[*of* $\lambda i.\ \mu G'\ (Z\ i)$] **by** *simp*
qed
then guess μ .. **note** μ = *this*
def $f \equiv$ *finmap-of J B*
have *emeasure* (*PiB I P*) (*emb I J* (Pi_E *J B*)) =
emeasure (*PiB I P*) (*emb I* (*domain f*) (Pi_E (*domain f*) $(f)_F$))
using *assms sets-into-space*
by (*simp add*: *f-def Pi-def*)
also have ... = *emeasure* (*PiB J P*) (Pi_E *J B*)
proof (*subst emeasure-extend-measure*[*OF PiP-def*, *of I* λ-. *borel* μ])
show *positive* (*sets* (*PiB I P*)) μ *countably-additive* (*sets* (*PiB I P*)) μ
using μ **unfolding** *sets-PiP sets-PiM-generator*[*OF* ‹$I \neq \{\}$›] **by** (*auto simp*: *measure-space-def*)
next
show $f \in \{x.\ (domain\ x \neq \{\} \vee I = \{\}) \wedge finite\ (domain\ x) \wedge domain\ x \subseteq I \wedge$
$(x)_F \in (\Pi_E\ i{\in}domain\ x.\ sets\ borel)\}$
using *assms* **by** (*auto simp*: *f-def*)
next
show $(\lambda x.\ emb\ I\ (domain\ x)\ (Pi_E\ (domain\ x)\ (x)_F))$ `
$\{x.\ (domain\ x \neq \{\} \vee I = \{\}) \wedge finite\ (domain\ x) \wedge domain\ x \subseteq I \wedge$
$(x)_F \in (\Pi_E\ i{\in}domain\ x.\ sets\ borel)\}$
$\subseteq Pow\ (\Pi_E\ i{\in}I.\ space\ borel)$ **by** (*auto simp*: *prod-emb-def*)
next
fix $i{::}'i \Rightarrow_F\ 'a\ set$
assume *i*: $i \in \{x.\ (domain\ x \neq \{\} \vee I = \{\}) \wedge finite\ (domain\ x) \wedge domain\ x \subseteq I \wedge$
$(x)_F \in (\Pi_E\ i{\in}domain\ x.\ sets\ borel)\}$
hence *emb I* (*domain i*) (Pi_E (*domain i*) $(i)_F$) $\in$ *generator*
using *assms* **by** (*auto intro*!: *generatorI'*)
hence μ (*emb I* (*domain i*) (Pi_E (*domain i*) $(i)_F$)) =
$\mu G'$ (*emb I* (*domain i*) (Pi_E (*domain i*) $(i)_F$))
using μ **by** *simp*
also have ... = *emeasure* (*P* (*domain i*)) (Pi_E (*domain i*) $(i)_F$)
using *i assms proj-sets* **by** (*subst* $\mu G'$-*eq*) (*auto simp*: $\mu G'$-*eq PiP-finite*)
finally show μ (*emb I* (*domain i*) (Pi_E (*domain i*) $(i)_F$)) =
emeasure (*P* (*domain i*)) (Pi_E (*domain i*) $(i)_F$) **.**
next
show *emeasure* (*P* (*domain f*)) (Pi_E (*domain f*) $(f)_F$) = *emeasure* (*PiB J P*) (Pi_E *J B*)
using *assms* **by** (*simp add*: *f-def PiP-finite Pi-def*)
qed
finally show *?thesis* **.**

qed

end

sublocale *polish-projective* ⊆ *P*: *prob-space* (*PiB I P*)
proof
  show *emeasure* (*PiB I P*) (*space* (*PiB I P*)) = *1*
  proof *cases*
    assume *I* = {} **then show** *?thesis*
      by (*simp add*: *space-PiM-empty PiP-finite prob-space.emeasure-space-1*)
  next
    assume *I* ≠ {}
    then obtain *i* **where** *i* ∈ *I* **by** *auto*
    moreover then have *R*: (*space* (*PiB I P*)) = (*emb I* {*i*} (Pi_E {*i*} (λ-. *space borel*)))
      by (*auto simp*: *prod-emb-def space-PiM*)
    moreover have *extensional* {*i*} = *space* (*P* {*i*}) **by** (*simp add*: *proj-space space-PiM*)
    ultimately show *?thesis*
      apply (*subst R*)
      apply (*subst emeasure-PiB-emb-not-empty*)
      apply (*auto simp*: *PiP-finite prob-space.emeasure-space-1*)
      done
  qed
qed

context *polish-projective* **begin**

lemma *emeasure-PiB-emb*:
  assumes *X*: *J* ⊆ *I finite J* ∀ *i*∈*J*. *B i* ∈ *sets borel*
  shows *emeasure* (*PiB I P*) (*emb I J* (Pi_E *J B*)) = *emeasure* (*P J*) (Pi_E *J B*)
proof *cases*
  assume *J* = {}
  moreover have *emb I* {} {λ*x*. *undefined*} = *space* (*PiB I P*)
    by (*auto simp*: *space-PiM prod-emb-def*)
  moreover have {λ*x*. *undefined*} = *space* (*PiB* {} *P*)
    by (*auto simp*: *space-PiM prod-emb-def*)
  ultimately show *?thesis*
    by (*simp add*: *P.emeasure-space-1 PiP-finite prob-space.emeasure-space-1 del*: *space-PiP*)
next
  assume *J* ≠ {} **with** *X* **show** *?thesis*
    by (*subst emeasure-PiB-emb-not-empty*) (*auto simp*: *PiP-finite*)
qed

lemma *measure-PiB-emb*:
  assumes *J* ⊆ *I finite J* ∀ *i*∈*J*. *B i* ∈ *sets borel*
  shows *measure* (*PiB I P*) (*emb I J* (Pi_E *J B*)) = *measure* (*P J*) (Pi_E *J B*)
  using *emeasure-PiB-emb*[*OF assms*]

unfolding *emeasure-eq-measure PiP-finite*[*OF* ‹*finite J*› ‹$J \subseteq I$›] *prob-space.emeasure-eq-measure*
by *simp*

end

end

References

[1] F. Immler. Generic construction of probability spaces for paths of stochastic processes in Isabelle/HOL. Master's thesis, Technische Universität München, October 2012. Submitted.