Standard Borel Spaces

Michikazu Hirata

April 21, 2024

Abstract

This entry includes a formalization of standard Borel spaces and (a variant of) the Borel isomorphism theorem. A separable complete metrizable topological space is called a polish space and a measurable space generated from a polish space is called a standard Borel space. We formalize the notion of standard Borel spaces by establishing set-based metric spaces, and then prove (a variant of) the Borel isomorphism theorem. The theorem states that a standard Borel spaces is either a countable discrete space or isomorphic to $\mathbb{R}$.

Contents

1 Lemmas **2**
1.1 Lemmas for Abstract Topology 2
1.1.1 Generated By 2
1.1.2 Isolated Point 9
1.1.3 Perfect Set 9
1.1.4 Bases and Sub-Bases in Abstract Topology 10
1.1.5 Separable Spaces 25
1.1.6 G_δ Set 30
1.1.7 Continuous Maps on First Countable Topology 32
1.1.8 Upper-Semicontinuous Functions 35
1.1.9 Lower-Semicontinuous Functions 36
1.2 Lemmas for Measure Theory 37
1.2.1 Lemmas for Measurable Sets 37
1.2.2 Measurable Isomorphisms 43
1.2.3 Borel Spaces Genereted from Abstract Topologies 55
1.3 Lemmas for Abstract Metric Spaces 64
1.3.1 Basic Lemmas 64
1.3.2 Dense in Metric Spaces 70
1.3.3 Separability in Metric Spaces 86
1.3.4 Compact Metric Spaces 90
1.3.5 Discrete Distance 96
1.3.6 Binary Product Metric Spaces 96

1.3.7 Sum Metric Spaces 101
1.3.8 Product Metric Spaces 110

2 Abstract Polish Spaces **130**
2.1 Polish Spaces 130
2.2 Extended Reals and Non-Negative Extended Reals 132
2.3 Continuous Embddings 141
2.3.1 Embedding into Hilbert Cube 142
2.3.2 Embedding from Cantor Space 146
2.4 Borel Spaces generated from Polish Spaces 157

3 Standard Borel Spaces **163**
3.1 Standard Borel Spaces 163
3.2 Isomorphism between $\mathscr{C}$ and $\mathscr{H}$ 169
3.3 Final Results 191

We refer to the HOL-Analysis library, the textbooks by Matsuzaka [2] and Srivastava [3], and the lecture note by Biskup [1].

1 Lemmas

theory *Lemmas-StandardBorel*
  imports *HOL−Probability.Probability*
begin

1.1 Lemmas for Abstract Topology

1.1.1 Generated By

lemma *topology-generated-by-sub*:
  assumes $\bigwedge U.\ U \in \mathcal{U} \Longrightarrow (openin\ X\ U)$
    and *openin* (*topology-generated-by* $\mathcal{U}$) *U*
   shows *openin X U*
proof −
  have *generate-topology-on* $\mathcal{U}$ *U*
   by (*simp add*: *assms*(*2*) *openin-topology-generated-by*)
  then show *?thesis*
   by *induction* (*use assms*(*1*) **in** *auto*)
qed

lemma *topology-generated-by-open*:
 $S = topology\text{-}generated\text{-}by\ \{U \mid U .\ openin\ S\ U\}$
  unfolding *topology-eq*
proof *standard+*
  fix *U*
  assume *openin* (*topology-generated-by* {*U* |*U*. *openin S U*}) *U*
  note *this*[*simplified openin-topology-generated-by-iff*]
  then show *openin S U*

```
    by induction auto
qed(simp add: openin-topology-generated-by-iff generate-topology-on.Basis)

lemma topology-generated-by-eq:
  assumes ⋀U. U ∈ 𝒰 ⟹ (openin (topology-generated-by 𝒪) U)
      and ⋀U. U ∈ 𝒪 ⟹ (openin (topology-generated-by 𝒰) U)
    shows topology-generated-by 𝒪 = topology-generated-by 𝒰
  using topology-generated-by-sub[of 𝒰, OF assms(1)] topology-generated-by-sub[of
𝒪,OF assms(2)]
  by(auto simp: topology-eq)

lemma topology-generated-by-homeomorphic-spaces:
  assumes homeomorphic-map X Y f X = topology-generated-by 𝒪
  shows Y = topology-generated-by ((`) f ` 𝒪)
  unfolding topology-eq
proof
  have f:open-map X Y f inj-on f (topspace X)
   using assms(1) by (simp-all add: homeomorphic-imp-open-map perfect-injective-eq-homeomorphic-map[sym
  obtain g where g: ⋀x. x ∈ topspace X ⟹ g (f x) = x ⋀y. y ∈ topspace Y ⟹
f (g y) = y open-map Y X g inj-on g (topspace Y)
   using homeomorphic-map-maps[of X Y f,simplified assms(1)] homeomorphic-imp-open-map
homeomorphic-maps-map[of X Y f] homeomorphic-imp-injective-map[of Y X] by
blast
  show ⋀S. openin Y S = openin (topology-generated-by ((`) f ` 𝒪)) S
  proof safe
    fix S
    assume openin Y S
    then have openin X (g ` S)
      using g(3) by (simp add: open-map-def)
    hence h:generate-topology-on 𝒪 (g ` S)
      by(simp add: assms(2) openin-topology-generated-by-iff)
    have S = f ` (g ` S)
      using openin-subset[OF ‹openin Y S›] g(2) by(fastforce simp: image-def)
    also have openin (topology-generated-by ((`) f ` 𝒪)) ...
      using h
    proof induction
      case Empty
      then show ?case by simp
    next
      case (Int a b)
      with inj-on-image-Int[OF f(2),of a b] show ?case
      by (metis assms(2) openin-Int openin-subset openin-topology-generated-by-iff)
    next
      case (UN K)
      then show ?case
        by(auto simp: image-Union)
    next
      case (Basis s)
      then show ?case
```

```
      by(auto intro!: generate-topology-on.Basis simp: openin-topology-generated-by-iff)
    qed
    finally show openin (topology-generated-by ((‘) f ‘ 𝒪)) S .
  next
    fix S
    assume openin (topology-generated-by ((‘) f ‘ 𝒪)) S
    then have generate-topology-on ((‘) f ‘ 𝒪) S
      by(simp add: openin-topology-generated-by-iff)
    thus openin Y S
    proof induction
      case (Basis s)
      then obtain U where u:U ∈ 𝒪 s = f ‘ U by auto
      then show ?case
      using assms(1) assms(2) homeomorphic-map-openness-eq topology-generated-by-Basis
by blast
    qed auto
  qed
qed

lemma open-map-generated-topo:
 assumes ⋀u. u ∈ U ⟹ openin S (f ‘ u) inj-on f (topspace (topology-generated-by
U))
  shows open-map (topology-generated-by U) S f
  unfolding open-map-def
proof safe
  fix u
  assume openin (topology-generated-by U) u
  then have generate-topology-on U u
    by(simp add: openin-topology-generated-by-iff)
  thus openin S (f ‘ u)
  proof induction
    case (Int a b)
    then have [simp]:f ‘ (a ∩ b) = f ‘ a ∩ f ‘ b
     by (meson assms(2) inj-on-image-Int openin-subset openin-topology-generated-by-iff)
    from Int show ?case by auto
  qed (simp-all add: image-Union openin-clauses(3) assms)
qed

lemma subtopology-generated-by:
 subtopology (topology-generated-by 𝒪) T = topology-generated-by {T ∩ U | U. U
∈ 𝒪}
  unfolding topology-eq openin-subtopology openin-topology-generated-by-iff
proof safe
  fix A
  assume generate-topology-on 𝒪 A
  then show generate-topology-on {T ∩ U |U. U ∈ 𝒪} (A ∩ T)
  proof induction
    case Empty
    then show ?case
```

```
      by (simp add: generate-topology-on.Empty)
    next
      case (Int a b)
      moreover have a ∩ b ∩ T = (a ∩ T) ∩ (b ∩ T) by auto
      ultimately show ?case
        by(auto intro!: generate-topology-on.Int)
    next
      case (UN K)
      moreover have (⋃ K ∩ T) = (⋃ { k ∩ T | k. k ∈ K}) by auto
      ultimately show ?case
        by(auto intro!: generate-topology-on.UN)
    next
      case (Basis s)
      then show ?case
        by(auto intro!: generate-topology-on.Basis)
    qed
  next
    fix A
    assume generate-topology-on {T ∩ U |U. U ∈ 𝒪} A
    then show ∃L. generate-topology-on 𝒪 L ∧ A = L ∩ T
    proof induction
      case Empty
      show ?case
        by(auto intro!: exI[where x={}] generate-topology-on.Empty)
    next
      case ih:(Int a b)
      then obtain La Lb where
       generate-topology-on 𝒪 La a = La ∩ T generate-topology-on 𝒪 Lb b = Lb ∩ T
        by auto
      thus ?case
        using ih by(auto intro!: exI[where x=La ∩ Lb] generate-topology-on.Int)
    next
      case ih:(UN K)
      then obtain L where
      ⋀k. k ∈ K ⟹ generate-topology-on 𝒪 (L k)  ⋀k. k ∈ K ⟹ k = (L k) ∩ T
        by metis
      thus ?case
       using ih by(auto intro!: exI[where x=⋃k∈K. L k] generate-topology-on.UN)
    next
      case (Basis s)
      then show ?case
        using generate-topology-on.Basis by fastforce
    qed
  qed

lemma prod-topology-generated-by:
   topology-generated-by { U × V | U V. U ∈ 𝒪 ∧ V ∈ 𝒰} = prod-topology
(topology-generated-by 𝒪) (topology-generated-by 𝒰)
  unfolding topology-eq
```

proof *safe*
fix *U*
assume *h*:*openin* (*topology-generated-by* {*U* × *V* |*U V*. *U* ∈ $\mathcal{O}$ ∧ *V* ∈ $\mathcal{U}$}) *U*
show *openin* (*prod-topology* (*topology-generated-by* $\mathcal{O}$) (*topology-generated-by* $\mathcal{U}$)) *U*
by(*auto simp*: *openin-prod-Times-iff*[*of topology-generated-by* $\mathcal{O}$ *topology-generated-by* $\mathcal{U}$]
intro!: *topology-generated-by-Basis topology-generated-by-sub*[*OF* - *h*])
next
fix *U*
assume *openin* (*prod-topology* (*topology-generated-by* $\mathcal{O}$) (*topology-generated-by* $\mathcal{U}$)) *U*
then have ∀ *z*∈*U*. ∃ *V1 V2*. *openin* (*topology-generated-by* $\mathcal{O}$) *V1* ∧ *openin* (*topology-generated-by* $\mathcal{U}$) *V2* ∧ *fst z* ∈ *V1* ∧ *snd z* ∈ *V2* ∧ *V1* × *V2* ⊆ *U*
by(*auto simp*: *openin-prod-topology-alt*)
hence ∃ *V1*. ∀ *z*∈*U*. ∃ *V2*. *openin* (*topology-generated-by* $\mathcal{O}$) (*V1 z*) ∧ *openin* (*topology-generated-by* $\mathcal{U}$) *V2* ∧ *fst z* ∈ (*V1 z*) ∧ *snd z* ∈ *V2* ∧ (*V1 z*) × *V2* ⊆ *U*
by(*rule bchoice*)
then obtain *V1* **where** ∀ *z*∈*U*. ∃ *V2*. *openin* (*topology-generated-by* $\mathcal{O}$) (*V1 z*) ∧ *openin* (*topology-generated-by* $\mathcal{U}$) *V2* ∧ *fst z* ∈ (*V1 z*) ∧ *snd z* ∈ *V2* ∧ (*V1 z*) × *V2* ⊆ *U*
by *auto*
hence ∃ *V2*. ∀ *z*∈*U*. *openin* (*topology-generated-by* $\mathcal{O}$) (*V1 z*) ∧ *openin* (*topology-generated-by* $\mathcal{U}$) (*V2 z*) ∧ *fst z* ∈ (*V1 z*) ∧ *snd z* ∈ (*V2 z*) ∧ (*V1 z*) × (*V2 z*) ⊆ *U*
by(*rule bchoice*)
then obtain *V2* **where** *hv12*:⋀*z*. *z*∈*U* ⟹ *openin* (*topology-generated-by* $\mathcal{O}$) (*V1 z*) ∧ *openin* (*topology-generated-by* $\mathcal{U}$) (*V2 z*) ∧ *fst z* ∈ (*V1 z*) ∧ *snd z* ∈ (*V2 z*) ∧ (*V1 z*) × (*V2 z*) ⊆ *U*
by *auto*
hence *1*:*U* = (⋃ *z*∈*U*. (*V1 z*) × (*V2 z*))
by *auto*
have *openin* (*topology-generated-by* {*U* × *V* |*U V*. *U* ∈ $\mathcal{O}$ ∧ *V* ∈ $\mathcal{U}$}) (⋃ *z*∈*U*. (*V1 z*) × (*V2 z*))
proof(*rule openin-Union*)
show ⋀*S*. *S* ∈ (λ*z*. *V1 z* × *V2 z*) ‘ *U* ⟹ *openin* (*topology-generated-by* {*U* × *V* |*U V*. *U* ∈ $\mathcal{O}$ ∧ *V* ∈ $\mathcal{U}$}) *S*
proof *safe*
fix *x y*
assume *h*:(*x*,*y*) ∈ *U*
then have *generate-topology-on* $\mathcal{O}$ (*V1* (*x*,*y*))
using *hv12* **by**(*auto simp*: *openin-topology-generated-by-iff*)
thus *openin* (*topology-generated-by* {*U* × *V* |*U V*. *U* ∈ $\mathcal{O}$ ∧ *V* ∈ $\mathcal{U}$}) (*V1* (*x*, *y*) × *V2* (*x*, *y*))
proof *induction*
case *Empty*
then show *?case* **by** *auto*
next
case (*Int a b*)
thus *?case*

by (*auto simp*: *Sigma-Int-distrib1*)
next
case (*UN K*)
then have *openin* (*topology-generated-by* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$) ($\bigcup\{ k \times V2\ (x,\ y) \mid k.\ k \in K\}$)
by *auto*
moreover have ($\bigcup\ \{k \times V2\ (x,\ y) \mid k.\ k \in K\}$) = ($\bigcup\ K \times V2\ (x,\ y)$)
by *blast*
ultimately show *?case* **by** *simp*
next
case *ho*:(*Basis s*)
have *generate-topology-on* $\mathcal{U}$ (*V2* (*x*,*y*))
using *h hv12* **by**(*auto simp*: *openin-topology-generated-by-iff*)
thus *?case*
proof *induction*
case *Empty*
then show *?case* **by** *auto*
next
case (*Int a b*)
then show *?case*
by (*auto simp*: *Sigma-Int-distrib2*)
next
case (*UN K*)
then have *openin* (*topology-generated-by* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$) ($\bigcup\ \{ s \times k \mid k.\ k \in K\}$)
by *auto*
moreover have ($\bigcup\ \{ s \times k \mid k.\ k \in K\}$) = $s \times \bigcup K$
by *blast*
ultimately show *?case* **by** *simp*
next
case (*Basis s'*)
then show *?case*
using *ho* **by**(*auto intro*!: *topology-generated-by-Basis*)
qed
qed
qed
qed
thus *openin* (*topology-generated-by* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$) *U*
using *1* **by** *auto*
qed

lemma *prod-topology-generated-by-open*:
prod-topology S S' = *topology-generated-by* $\{U \times V \mid U\ V.$ *openin S U* $\wedge$ *openin S' V*$\}$
using *prod-topology-generated-by*[*of* $\{U \mid U.$ *openin S U*$\}$ $\{U \mid U.$ *openin S' U*$\}$] *topology-generated-by-open*[*of S*,*symmetric*] *topology-generated-by-open*[*of S'*]
by *auto*

lemma *product-topology-cong*:

assumes $\bigwedge i.\ i \in I \Longrightarrow S\ i = K\ i$
shows *product-topology S I = product-topology K I*
proof −
have *1*:$\{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (S\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (S\ i)\}\} \subseteq \{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (K\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (K\ i)\}\}$ **if** $\bigwedge i.\ i \in I \Longrightarrow S\ i = K\ i$ **for** *S K* :: - ⇒ *'b topology*
proof
fix *x*
assume *hx*:$x \in \{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (S\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (S\ i)\}\}$
then obtain *X* **where** *hX*:
$x = (\Pi_E\ i{\in}I.\ X\ i)$ $\bigwedge i.\ openin\ (S\ i)\ (X\ i)$ $finite\ \{i.\ X\ i \neq topspace\ (S\ i)\}$
by *auto*
define *X'* **where** $X' \equiv (\lambda i.\ if\ i \in I\ then\ X\ i\ else\ topspace\ (K\ i))$
have $x = (\Pi_E\ i{\in}I.\ X'\ i)$
by(*auto simp*: *hX*(*1*) *X'-def PiE-def Pi-def*)
moreover have $finite\ \{i.\ X'\ i \neq topspace\ (K\ i)\}$
using *that* **by**(*auto intro*!: *finite-subset*[*OF* - *hX*(*3*)] *simp*: *X'-def*)
moreover have *openin* (*K i*) (*X' i*) **for** *i*
using *hX*(*2*)[*of i*] *that*[*of i*] **by**(*auto simp*: *X'-def*)
ultimately show $x \in \{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (K\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (K\ i)\}\}$
by(*auto intro*!: *exI*[**where** *x=X'*])
qed
have $\{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (S\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (S\ i)\}\} = \{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (K\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (K\ i)\}\}$
using *1*[*of S K*] *1*[*of K S*] *assms* **by** *auto*
thus *?thesis*
by(*simp add*: *product-topology-def*)
qed

lemma *topology-generated-by-without-empty*:
$topology\text{-}generated\text{-}by\ \mathcal{O} = topology\text{-}generated\text{-}by\ \{\ U \in \mathcal{O}.\ U \neq \{\}\}$
proof(*rule topology-generated-by-eq*)
fix *U*
show $U \in \mathcal{O} \Longrightarrow openin\ (topology\text{-}generated\text{-}by\ \{\ U \in \mathcal{O}.\ U \neq \{\}\})\ U$
by(*cases* $U = \{\}$) (*simp-all add*: *topology-generated-by-Basis*)
qed (*simp add*: *topology-generated-by-Basis*)

lemma *topology-from-bij*:
assumes *bij-betw f A* (*topspace S*)
shows *homeomorphic-map* (*pullback-topology A f S*) *S f topspace* (*pullback-topology A f S*) = *A*
proof −
note *h* = *bij-betw-imp-surj-on*[*OF assms*] *bij-betw-inv-into-left*[*OF assms*] *bij-betw-inv-into-right*[*OF assms*]
then show [*simp*]:*topspace* (*pullback-topology A f S*) = *A*
by(*auto simp*: *topspace-pullback-topology*)

show *homeomorphic-map* (*pullback-topology A f S*) *S f*
by(*auto simp*: *homeomorphic-map-maps homeomorphic-maps-def h continuous-map-pullback*[*OF continuous-map-id*,*simplified*] *inv-into-into intro*!: *exI*[**where** *x*=*inv-into A f*] *continuous-map-pullback′*[**where** *f*=*f*]) (*metis* (*mono-tags*, *opaque-lifting*) *comp-apply continuous-map-eq continuous-map-id h*(*3*) *id-apply*)
qed

lemma *openin-pullback-topology′*:
assumes *bij-betw f A* (*topspace S*)
shows *openin* (*pullback-topology A f S*) *u* $\longleftrightarrow$ (*openin S* (*f ‘ u*)) $\wedge$ *u* $\subseteq$ *A*
unfolding *openin-pullback-topology*
proof *safe*
fix *U*
assume *h*:*openin S U u* = *f* −‘ *U* ∩ *A*
from *openin-subset*[*OF this*(*1*)] *assms*
have [*simp*]:*f ‘* (*f* −‘ *U* ∩ *A*) = *U*
by(*auto simp*: *image-def vimage-def bij-betw-def*)
show *openin S* (*f ‘* (*f* −‘ *U* ∩ *A*))
by(*simp add*: *h*)
next
assume *openin S* (*f ‘ u*) *u* ⊆ *A*
with *assms* **show** ∃ *U*. *openin S U* ∧ *u* = *f* −‘ *U* ∩ *A*
by(*auto intro*!: *exI*[**where** *x*=*f ‘ u*] *simp*: *bij-betw-def inj-on-def*)
qed

1.1.2 Isolated Point

definition *isolated-points-of* :: ′*a topology* ⇒ ′*a set* ⇒ ′*a set* (**infixr** *isolated′-points′-of 80*) **where**
X isolated-points-of A ≡ {*x*∈*topspace X* ∩ *A*. *x* ∉ *X derived-set-of A*}

lemma *isolated-points-of-eq*:
X isolated-points-of A = {*x*∈*topspace X* ∩ *A*. ∃ *U*. *x* ∈ *U* ∧ *openin X U* ∧ *U* ∩ (*A* − {*x*}) = {}}
unfolding *isolated-points-of-def* **by**(*auto simp*: *in-derived-set-of*)

lemma *in-isolated-points-of*:
x ∈ *X isolated-points-of A* $\longleftrightarrow$ *x* ∈ *topspace X* ∧ *x* ∈ *A* ∧ (∃ *U*. *x* ∈ *U* ∧ *openin X U* ∧ *U* ∩ (*A* − {*x*}) = {})
by(*simp add*: *isolated-points-of-eq*)

lemma *derived-set-of-eq*:
x ∈ *X derived-set-of A* $\longleftrightarrow$ *x* ∈ *X closure-of* (*A* − {*x*})
by(*auto simp*: *in-derived-set-of in-closure-of*)

1.1.3 Perfect Set

definition *perfect-set* :: ′*a topology* ⇒ ′*a set* ⇒ *bool* **where**
perfect-set X A $\longleftrightarrow$ *closedin X A* ∧ *X isolated-points-of A* = {}

abbreviation *perfect-space X* $\equiv$ *perfect-set X* (*topspace X*)

lemma *perfect-space-euclidean*: *perfect-space* (*euclidean* :: *'a* :: *perfect-space topology*)
by(*auto simp*: *isolated-points-of-def perfect-set-def derived-set-of-eq closure-interior*)

lemma *perfect-setI*:
assumes *closedin X A*
and $\bigwedge x\ T.\ [\![x \in A;\ x \in T;\ openin\ X\ T]\!] \Longrightarrow \exists\, y{\neq}x.\ y \in T \wedge y \in A$
shows *perfect-set X A*
using *assms* **by**(*simp add*: *perfect-set-def isolated-points-of-def in-derived-set-of*) *blast*

lemma *perfect-spaceI*:
assumes $\bigwedge x\ T.\ [\![x \in T;\ openin\ X\ T]\!] \Longrightarrow \exists\, y{\neq}x.\ y \in T$
shows *perfect-space X*
using *assms* **by**(*auto intro*!: *perfect-setI*) (*meson in-mono openin-subset*)

lemma *perfect-setD*:
assumes *perfect-set X A*
shows *closedin X A* $A \subseteq$ *topspace X* $\bigwedge x\ T.\ [\![x \in A;\ x \in T;\ openin\ X\ T]\!] \Longrightarrow \exists\, y{\neq}x.\ y \in T \wedge y \in A$
using *assms closedin-subset*[*of X A*] **by**(*simp-all add*: *perfect-set-def isolated-points-of-def in-derived-set-of*) *blast*

lemma *perfect-space-perfect*:
perfect-set euclidean (*UNIV* :: *'a* :: *perfect-space set*)
by(*auto simp*: *perfect-set-def in-isolated-points-of*) (*metis Int-Diff inf-top.right-neutral insert-Diff not-open-singleton*)

lemma *perfect-set-subtopology*:
assumes *perfect-set X A*
shows *perfect-space* (*subtopology X A*)
using *perfect-setD*[*OF assms*] **by**(*auto intro*!: *perfect-setI simp*: *inf.absorb-iff2 openin-subtopology*)

1.1.4 Bases and Sub-Bases in Abstract Topology

definition *subbase-in* :: [*'a topology*, *'a set set*] $\Rightarrow$ *bool* **where**
subbase-in S $\mathcal{O} \longleftrightarrow S =$ *topology-generated-by* $\mathcal{O}$

definition *base-in* :: [*'a topology*, *'a set set*] $\Rightarrow$ *bool* **where**
base-in S $\mathcal{O} \longleftrightarrow (\forall\, U.\ openin\ S\ U \longleftrightarrow (\exists \mathcal{U}.\ U = \bigcup \mathcal{U} \wedge \mathcal{U} \subseteq \mathcal{O}))$

lemma *second-countable-base-in*: *second-countable S* $\longleftrightarrow (\exists\, \mathcal{O}.\ countable\ \mathcal{O} \wedge base\text{-}in\ S\ \mathcal{O})$
proof −
have [*simp*]: $\bigwedge \mathcal{B}.\ (openin\ S = arbitrary\ union\text{-}of\ (\lambda x.\ x \in \mathcal{B})) \longleftrightarrow (\forall\, U.\ openin\ S\ U \longleftrightarrow (\exists \mathcal{U}.\ U = \bigcup \mathcal{U} \wedge \mathcal{U} \subseteq \mathcal{B}))$

```
    by(simp add: arbitrary-def union-of-def fun-eq-iff) metis
  show ?thesis
    by(auto simp: second-countable base-in-def)
qed

definition zero-dimensional :: 'a topology ⇒ bool where
zero-dimensional S ⟷ (∃𝒪. base-in S 𝒪 ∧ (∀u∈𝒪. openin S u ∧ closedin S u))

lemma openin-base:
  assumes base-in S 𝒪  U = ⋃𝒰 and 𝒰 ⊆ 𝒪
  shows openin S U
  using assms by(auto simp: base-in-def)

lemma base-is-subbase:
  assumes base-in S 𝒪
  shows subbase-in S 𝒪
  unfolding subbase-in-def topology-eq openin-topology-generated-by-iff
proof safe
  fix U
  assume openin S U
  then obtain 𝒰 where hu:U = ⋃𝒰 𝒰 ⊆ 𝒪
    using assms by(auto simp: base-in-def)
  thus generate-topology-on 𝒪 U
   by(auto intro!: generate-topology-on.UN) (auto intro!: generate-topology-on.Basis)
next
  fix U
  assume generate-topology-on 𝒪 U
  then show openin S U
  proof induction
    case (Basis s)
    then show ?case
      using openin-base[OF assms,of s {s}]
      by auto
  qed auto
qed

lemma subbase-in-subset:
  assumes subbase-in S 𝒪 and U ∈ 𝒪
  shows U ⊆ topspace S
  using assms(1)[simplified subbase-in-def] topology-generated-by-topspace assms
  by auto

lemma subbase-in-openin:
  assumes subbase-in S 𝒪 and U ∈ 𝒪
  shows openin S U
  using assms by(simp add: subbase-in-def openin-topology-generated-by-iff gener-
ate-topology-on.Basis)

lemma base-in-subset:
```

assumes *base-in* S $\mathcal{O}$ **and** $U \in \mathcal{O}$
shows $U \subseteq$ *topspace* S
using *subbase-in-subset*[*OF base-is-subbase*[*OF assms*(*1*)] *assms*(*2*)] **.**

lemma *base-in-openin*:
assumes *base-in* S $\mathcal{O}$ **and** $U \in \mathcal{O}$
shows *openin* S U
using *subbase-in-openin*[*OF base-is-subbase*[*OF assms*(*1*)] *assms*(*2*)] **.**

lemma *base-in-def2*:
assumes $\bigwedge U.\ U \in \mathcal{O} \Longrightarrow$ *openin* S U
shows *base-in* S $\mathcal{O} \longleftrightarrow (\forall\, U.$ *openin* S $U \longrightarrow (\forall\, x{\in}U.\ \exists\, W{\in}\mathcal{O}.\ x \in W \wedge W \subseteq U))$
proof
assume *h*:*base-in* S $\mathcal{O}$
show $\forall\, U.$ *openin* S $U \longrightarrow (\forall\, x{\in}U.\ \exists\, W{\in}\mathcal{O}.\ x \in W \wedge W \subseteq U)$
proof *safe*
fix U x
assume h':*openin* S U $x \in U$
then obtain $\mathcal{U}$ **where** *hu*: $U = \bigcup\mathcal{U}$ $\mathcal{U} \subseteq \mathcal{O}$
using *h* **by**(*auto simp*: *base-in-def*)
then obtain W **where** $x \in W$ $W \in \mathcal{U}$
using $h'(2)$ **by** *blast*
thus $\exists\, W{\in}\mathcal{O}.\ x \in W \wedge W \subseteq U$
using *hu* **by**(*auto intro*!: *bexI*[**where** $x{=}W$])
qed
next
assume *h*:$\forall\, U.$ *openin* S $U \longrightarrow (\forall\, x{\in}U.\ \exists\, W{\in}\mathcal{O}.\ x \in W \wedge W \subseteq U)$
show *base-in* S $\mathcal{O}$
unfolding *base-in-def*
proof *safe*
fix U
assume *openin* S U
then have $\forall\, x{\in}U.\ \exists\, W.\ W{\in}\mathcal{O} \wedge x \in W \wedge W \subseteq U$
using *h* **by** *blast*
hence $\exists\, W.\ \forall\, x{\in}U.\ W\ x \in \mathcal{O} \wedge x \in W\ x \wedge W\ x \subseteq U$
by(*rule bchoice*)
then obtain W **where** *hw*:
$\forall\, x{\in}U.\ W\ x \in \mathcal{O} \wedge x \in W\ x \wedge W\ x \subseteq U$ **by** *auto*
thus $\exists\mathcal{U}.\ U = \bigcup\ \mathcal{U} \wedge \mathcal{U} \subseteq \mathcal{O}$
by(*auto intro*!: *exI*[**where** $x{=}W$ ` U])
next
fix U $\mathcal{U}$
show $\mathcal{U} \subseteq \mathcal{O} \Longrightarrow$ *openin* S $(\bigcup\ \mathcal{U})$
using *assms* **by** *auto*
qed
qed

lemma *base-in-def2'*:

```
 base-in S 𝒪 ⟷ (∀ b∈𝒪. openin S b) ∧ (∀ x. openin S x ⟶ (∃ B′⊆𝒪. ⋃ B′ =
x))
proof
  assume h:base-in S 𝒪
  show (∀ b∈𝒪. openin S b) ∧ (∀ x. openin S x ⟶ (∃ B′⊆𝒪. ⋃ B′ = x))
  proof(rule conjI)
    show ∀ b∈𝒪. openin S b
      using openin-base[OF h,of - {-}] by auto
  next
    show ∀ x. openin S x ⟶ (∃ B′⊆𝒪. ⋃ B′ = x)
      using h by(auto simp: base-in-def)
  qed
next
  assume h:(∀ b∈𝒪. openin S b) ∧ (∀ x. openin S x ⟶ (∃ B′⊆𝒪. ⋃ B′ = x))
  show base-in S 𝒪
    unfolding base-in-def
  proof safe
    fix U
    assume openin S U
    then obtain B′ where B′⊆𝒪 ⋃ B′ = U
      using h by blast
    thus ∃𝒰. U = ⋃ 𝒰 ∧ 𝒰 ⊆ 𝒪
      by(auto intro!: exI[where x=B′])
  next
    fix U 𝒰
    show 𝒰 ⊆ 𝒪 ⟹ openin S (⋃ 𝒰)
      using h by auto
  qed
qed

corollary base-in-in-subset:
  assumes base-in S 𝒪 openin S u x ∈ u
  shows ∃ v∈𝒪. x ∈ v ∧ v ⊆ u
  using assms base-in-def2 base-in-def2′ by fastforce

lemma base-in-without-empty:
  assumes base-in S 𝒪
  shows base-in S {U ∈ 𝒪. U ≠ {}}
  unfolding base-in-def2′
proof safe
  fix x
  assume x ∈ 𝒪  ¬ openin S x
  thus ⋀y. y ∈ {}
    using base-in-openin[OF assms ‹x ∈ 𝒪›] by simp
next
  fix x
  assume openin S x
  then obtain B′ where B′ ⊆𝒪 ⋃ B′ = x
    using assms by(simp add: base-in-def2′) metis
```

thus $\exists B' \subseteq \{U \in \mathcal{O}.\ U \neq \{\}\}.\ \bigcup\ B' = x$
by(*auto intro*!: *exI*[**where** $x=\{y \in B'.\ y \neq \{\}\}$])
qed

lemma *second-countable-ex-without-empty*:
assumes *second-countable S*
shows $\exists \mathcal{O}.$ *countable* $\mathcal{O} \wedge$ *base-in* $S\ \mathcal{O} \wedge (\forall U \in \mathcal{O}.\ U \neq \{\})$
proof −
obtain $\mathcal{O}$ **where** *countable* $\mathcal{O}$ *base-in* $S\ \mathcal{O}$
using *assms second-countable-base-in* **by** *blast*
thus *?thesis*
by(*auto intro*!: *exI*[**where** $x=\{U \in \mathcal{O}.\ U \neq \{\}\}$] *base-in-without-empty*)
qed

lemma *subtopology-subbase-in*:
assumes *subbase-in* $S\ \mathcal{O}$
shows *subbase-in* (*subtopology* $S\ T$) $\{T \cap U \mid U.\ U \in \mathcal{O}\}$
using *assms subtopology-generated-by*
by(*auto simp*: *subbase-in-def*)

lemma *subtopology-base-in*:
assumes *base-in* $S\ \mathcal{O}$
shows *base-in* (*subtopology* $S\ T$) $\{T \cap U \mid U.\ U \in \mathcal{O}\}$
unfolding *base-in-def*
proof
fix L
show *openin* (*subtopology* $S\ T$) $L = (\exists \mathcal{U}.\ L = \bigcup\ \mathcal{U} \wedge \mathcal{U} \subseteq \{T \cap U \mid U.\ U \in \mathcal{O}\})$
proof
assume *openin* (*subtopology* $S\ T$) L
then obtain T' **where** *ht*:
openin $S\ T'$ $L = T' \cap T$
by(*auto simp*: *openin-subtopology*)
then obtain $\mathcal{U}$ **where** *hu*:
$T' = (\bigcup\ \mathcal{U})\ \mathcal{U} \subseteq \mathcal{O}$
using *assms* **by**(*auto simp*: *base-in-def*)
show $\exists \mathcal{U}.\ L = \bigcup\ \mathcal{U} \wedge \mathcal{U} \subseteq \{T \cap U \mid U.\ U \in \mathcal{O}\}$
using *hu ht* **by**(*auto intro*!: *exI*[**where** $x=\{T \cap U \mid U.\ U \in \mathcal{U}\}$])
next
assume $\exists \mathcal{U}.\ L = \bigcup\ \mathcal{U} \wedge \mathcal{U} \subseteq \{T \cap U \mid U.\ U \in \mathcal{O}\}$
then obtain $\mathcal{U}$ **where** *hu*: $L = \bigcup\ \mathcal{U}\ \mathcal{U} \subseteq \{T \cap U \mid U.\ U \in \mathcal{O}\}$
by *auto*
hence $\forall U \in \mathcal{U}.\ \exists U' \in \mathcal{O}.\ U = T \cap U'$ **by** *blast*
then obtain k **where** *hk*:$\bigwedge U.\ U \in \mathcal{U} \Longrightarrow k\ U \in \mathcal{O}\ \bigwedge U.\ U \in \mathcal{U} \Longrightarrow U = T \cap k\ U$
by *metis*
hence $L = \bigcup\ \{T \cap k\ U \mid U.\ U \in \mathcal{U}\}$
using *hu* **by** *auto*
also have ... $= \bigcup\ \{k\ U \mid U.\ U \in \mathcal{U}\} \cap T$ **by** *auto*

finally have *1*:$L = \bigcup \{k\ U\ |U.\ U \in \mathcal{U}\} \cap T$ **.**
moreover have *openin S* $(\bigcup \{k\ U\ |U.\ U \in \mathcal{U}\})$
using *hu hk assms* **by**(*auto simp*: *base-in-def*)
ultimately show *openin* (*subtopology S T*) *L*
by(*auto intro*!: *exI*[**where** $x = \bigcup \{k\ U\ |U.\ U \in \mathcal{U}\}$] *simp*: *openin-subtopology*)
qed
qed

lemma *second-countable-subtopology*:
assumes *second-countable S*
shows *second-countable* (*subtopology S T*)
proof −
obtain $\mathcal{O}$ **where** *countable* $\mathcal{O}$ *base-in S* $\mathcal{O}$
using *assms second-countable-base-in* **by** *blast*
thus *?thesis*
by(*auto intro*!: *exI*[**where** $x = \{T \cap U \mid U.\ U \in \mathcal{O}\}$] *simp*: *second-countable-base-in Setcompr-eq-image dest*: *subtopology-base-in*)
qed

lemma *open-map-with-base*:
assumes *base-in S* $\mathcal{O}$ $\bigwedge A.\ A \in \mathcal{O} \Longrightarrow$ *openin* S' (f ‘ A)
shows *open-map S* S' *f*
unfolding *open-map-def*
proof *safe*
fix *U*
assume *openin S U*
then obtain $\mathcal{U}$ **where** $U = \bigcup \mathcal{U}$ $\mathcal{U} \subseteq \mathcal{O}$
using *assms*(*1*) **by**(*auto simp*: *base-in-def*)
hence f ‘ $U = \bigcup \{ f$ ‘ $A \mid A.\ A \in \mathcal{U}\}$ **by** *blast*
also have *openin* S' ...
using *assms*(*2*) ‹$\mathcal{U} \subseteq \mathcal{O}$› **by** *auto*
finally show *openin* S' (f ‘ U) **.**
qed

Construct a base from a subbase.

lemma *finite′-intersection-of-idempot* [*simp*]:
finite′ intersection-of finite′ intersection-of P = *finite′ intersection-of P*
proof
fix *A*
show (*finite′ intersection-of finite′ intersection-of P*) *A* = (*finite′ intersection-of P*) *A*
proof
assume (*finite′ intersection-of finite′ intersection-of P*) *A*
then obtain $\mathcal{U}$ **where** $\mathcal{U}$:*finite′* $\mathcal{U} \land \mathcal{U} \subseteq$ *Collect* (*finite′ intersection-of P*) $\land \bigcap \mathcal{U} = A$
by(*auto simp*: *intersection-of-def*)
hence $\forall\ U \in \mathcal{U}.\ \exists \mathcal{U}'.$ *finite′* $\mathcal{U}' \land \mathcal{U}' \subseteq$ *Collect P* $\land \bigcap \mathcal{U}' = U$
by(*auto simp*: *intersection-of-def*)
then obtain $\mathcal{U}'$ **where** $\mathcal{U}'$:

$\bigwedge U.\ U \in \mathcal{U} \Longrightarrow$ *finite'* $(\mathcal{U}'\ U)$ $\bigwedge U.\ U \in \mathcal{U} \Longrightarrow \mathcal{U}'\ U \subseteq$ *Collect P* $\bigwedge U.\ U \in \mathcal{U} \Longrightarrow \bigcap(\mathcal{U}'\ U) = U$
by *metis*
have *1*: $\bigcap\ (\bigcup\ (\mathcal{U}'\ `\ \mathcal{U})) = A$
using $\mathcal{U}\ \mathcal{U}'(3)$ **by** *blast*
show (*finite' intersection-of P*) *A*
unfolding *intersection-of-def*
using $\mathcal{U}\ \mathcal{U}'(1,2)$ *1* **by**(*auto intro*!: *exI*[**where** $x{=}\bigcup U{\in}\mathcal{U}.\ \mathcal{U}'\ U$])
qed(*rule finite'-intersection-of-inc*)
qed

lemma *finite'-intersection-of-countable*:
assumes *countable* $\mathcal{O}$
shows *countable* (*Collect* (*finite' intersection-of* $(\lambda x.\ x \in \mathcal{O})$))
proof −
have *Collect* (*finite' intersection-of* $(\lambda x.\ x \in \mathcal{O})) = (\bigcup i{\in}\{\mathcal{O}'.\ \mathcal{O}' \neq \{\} \wedge$ *finite* $\mathcal{O}' \wedge \mathcal{O}' \subseteq \mathcal{O}\}.\ \{\bigcap\ i\})$
by(*auto simp*: *intersection-of-def*)
also have *countable* ...
using *countable-Collect-finite-subset*[*OF assms*]
by(*auto intro*!: *countable-UN*[*of* $\{\ \mathcal{O}'.\ \mathcal{O}' \neq \{\} \wedge$ *finite* $\mathcal{O}' \wedge \mathcal{O}' \subseteq \mathcal{O}\}\ \lambda\mathcal{O}'.\ \{\bigcap \mathcal{O}'\}$])
(*auto intro*!: *countable-subset*[*of* $\{\mathcal{O}'.\ \mathcal{O}' \neq \{\} \wedge$ *finite* $\mathcal{O}' \wedge \mathcal{O}' \subseteq \mathcal{O}\}\ \{A.$ *finite* $A \wedge A \subseteq \mathcal{O}\}$])
finally show *?thesis* **.**
qed

lemma *finite'-intersection-of-openin*:
assumes (*finite' intersection-of* $(\lambda x.\ x \in \mathcal{O})$) *U*
shows *openin* (*topology-generated-by* $\mathcal{O}$) *U*
unfolding *openin-topology-generated-by-iff*
using *assms* **by**(*auto simp*: *generate-topology-on-eq arbitrary-union-of-inc*)

lemma *topology-generated-by-finite-intersections*:
topology-generated-by $\mathcal{O}$ = *topology-generated-by* (*Collect* (*finite' intersection-of* $(\lambda x.\ x \in \mathcal{O})$))
unfolding *topology-eq openin-topology-generated-by-iff* **by**(*simp add*: *generate-topology-on-eq*)

lemma *base-from-subbase*:
assumes *subbase-in S* $\mathcal{O}$
shows *base-in S* (*Collect* (*finite' intersection-of* $(\lambda x.\ x \in \mathcal{O})$))
unfolding *subbase-in-def base-in-def assms*[*simplified subbase-in-def*] *openin-topology-generated-by-iff*
by(*auto simp*: *arbitrary-def union-of-def generate-topology-on-eq*)

lemma *countable-base-from-countable-subbase*:
assumes *countable* $\mathcal{O}$ **and** *subbase-in S* $\mathcal{O}$
shows *second-countable S*
using *finite'-intersection-of-countable*[*OF assms*(*1*)] *base-from-subbase*[*OF assms*(*2*)]
by(*auto simp*: *second-countable-base-in*)

lemma *prod-topology-second-countable*:
  assumes *second-countable S* **and** *second-countable S'*
  shows *second-countable* (*prod-topology S S'*)
proof −
  obtain $\mathcal{O}$ $\mathcal{O}'$ **where** *ho*:
   countable $\mathcal{O}$ *base-in S* $\mathcal{O}$ *countable* $\mathcal{O}'$ *base-in S'* $\mathcal{O}'$
    using *assms* **by**(*auto simp*: *second-countable-base-in*)
  show *?thesis*
  proof(*rule countable-base-from-countable-subbase*[**where** $\mathcal{O}$={ $U \times V$ | U V. $U \in \mathcal{O} \wedge V \in \mathcal{O}'$}])
    have $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{O}'\} = (\lambda(U,V).\ U \times V)$ ‘ $(\mathcal{O} \times \mathcal{O}')$
      by *auto*
    also have *countable* ...
      using *ho*(*1*,*3*) **by** *auto*
    finally show *countable* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{O}'\}$ **.**
  next
    show *subbase-in* (*prod-topology S S'*) $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{O}'\}$
      using *base-is-subbase*[*OF ho*(*2*)] *base-is-subbase*[*OF ho*(*4*)]
      by(*simp add*: *subbase-in-def prod-topology-generated-by*)
  qed
qed

Abstract version of the theorem $\exists K.$ *topological-basis* $K \wedge$ *countable* $K \wedge (\forall k{\in}K.\ \exists X.\ k = Pi_E\ UNIV\ X \wedge (\forall i.\ open\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq UNIV\})$.

lemma *product-topology-countable-base-in*:
  assumes *countable I* **and** $\bigwedge i.\ i \in I \Longrightarrow$ *second-countable* (*S i*)
  shows $\exists \mathcal{O}'.$ *countable* $\mathcal{O}' \wedge$ *base-in* (*product-topology S I*) $\mathcal{O}' \wedge$
        $(\forall k \in \mathcal{O}'.\ \exists X.\ k = (\Pi_E\ i{\in}I.\ X\ i) \wedge (\forall i.\ openin\ (S\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (S\ i)\} \wedge \{i.\ X\ i \neq topspace\ (S\ i)\} \subseteq I)$
proof −
  obtain $\mathcal{O}$ **where** *ho*:
  $\bigwedge i.\ i \in I \Longrightarrow$ *countable* $(\mathcal{O}\ i)$ $\bigwedge i.\ i \in I \Longrightarrow$ *base-in* $(S\ i)\ (\mathcal{O}\ i)$
    using *assms*(*2*)[*simplified second-countable-base-in*] **by** *metis*
  show *?thesis*
    unfolding *second-countable-base-in*
  proof(*intro exI*[**where** $x=\{\Pi_E\ i{\in}I.\ U\ i \mid U.\ finite\ \{i{\in}I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall i{\in}\{i{\in}I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i)\}$] *conjI*)
   show *countable* $\{\Pi_E\ i{\in}I.\ U\ i \mid U.\ finite\ \{i{\in}I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall i{\in}\{i{\in}I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i)\}$
       (**is** *countable ?X*)
    proof −
     have $?X = \{\Pi_E\ i{\in}I.\ U\ i \mid U.\ finite\ \{i{\in}I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall i{\in}\{i{\in}I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall i \in (UNIV-\ I).\ U\ i = \{undefined\})\}$
        (**is** - = *?Y*)
      proof (*rule set-eqI*)
        show $\bigwedge x.\ x \in ?X \longleftrightarrow x \in ?Y$
        proof

fix x
assume $x \in$ *?X*
then obtain U **where** *hu*:
$x = (\Pi_E\ i{\in}I.\ U\ i)$ *finite* $\{i{\in}I.\ U\ i \neq$ *topspace* $(S\ i)\}$ $(\forall\, i{\in}\{i{\in}I.\ U\ i \neq$ *topspace* $(S\ i)\}.\ U\ i \in \mathcal{O}\ i)$
by *auto*
define U' **where** $U'\ i \equiv$ (*if* $i \in I$ *then* $U\ i$ *else* $\{$*undefined*$\}$) **for** i
have $x = (\Pi_E\ i{\in}I.\ U'\ i)$
using *hu*(*1*) **by**(*auto simp*: *U′-def PiE-def extensional-def Pi-def*)
moreover have *finite* $\{i{\in}I.\ U'\ i \neq$ *topspace* $(S\ i)\}$ $(\forall\, i{\in}\{i{\in}I.\ U'\ i \neq$ *topspace* $(S\ i)\}.\ U'\ i \in \mathcal{O}\ i)$ $\forall\, i \in (UNIV - I).\ U'\ i = \{$*undefined*$\}$
using *hu*(*2*,*3*) **by**(*auto simp*: *U′-def*) (*metis* (*mono-tags*, *lifting*) *Collect-cong*)
ultimately show $x \in$ *?Y* **by** *auto*
qed *auto*
qed
also have ... $= (\lambda U.\ \Pi_E\ i{\in}I.\ U\ i)$ ‘ $\{U.$ *finite* $\{i{\in}I.\ U\ i \neq$ *topspace* $(S\ i)\} \wedge (\forall\, i{\in}\{i{\in}I.\ U\ i \neq$ *topspace* $(S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i \in (UNIV - I).\ U\ i = \{$*undefined*$\})\}$ **by** *auto*
also have *countable* ...
proof(*rule countable-image*)
have $\{U.$ *finite* $\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\} \wedge (\forall\, i{\in}\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}UNIV - I.\ U\ i = \{$*undefined*$\})\} = \{U.\ \exists I'.$ *finite* $I' \wedge I' \subseteq I \wedge (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}(I - I').\ U\ i =$ *topspace* $(S\ i)) \wedge (\forall\, i{\in}UNIV - I.\ U\ i = \{$*undefined*$\})\}$
(**is** *?A* = *?B*)
proof (*rule set-eqI*)
show $\bigwedge x.\ x \in$ *?A* $\longleftrightarrow x \in$ *?B*
proof
fix U
assume $U \in \{U.$ *finite* $\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\} \wedge (\forall\, i{\in}\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}UNIV - I.\ U\ i = \{$*undefined*$\})\}$
then show $U \in \{U.\ \exists I'.$ *finite* $I' \wedge I' \subseteq I \wedge (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i =$ *topspace* $(S\ i)) \wedge (\forall\, i{\in}UNIV - I.\ U\ i = \{$*undefined*$\})\}$
by *auto*
next
fix U
assume *assm*: $U \in \{U.\ \exists I'.$ *finite* $I' \wedge I' \subseteq I \wedge (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i =$ *topspace* $(S\ i)) \wedge (\forall\, i{\in}UNIV - I.\ U\ i = \{$*undefined*$\})\}$
then obtain I' **where** *hi′*:
finite I' $I' \subseteq I$ $\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i$ $\forall\, i{\in}I - I'.\ U\ i =$ *topspace* $(S\ i)$ $\forall\, i{\in}UNIV - I.\ U\ i = \{$*undefined*$\}$
by *auto*
then have $\bigwedge i.\ i \in I \Longrightarrow U\ i \neq$ *topspace* $(S\ i) \Longrightarrow i \in I'$ **by** *auto*
hence $\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\} \subseteq I'$ **by** *auto*
hence *finite* $\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\}$
using *hi′*(*1*) **by** (*simp add*: *rev-finite-subset*)
thus $U \in \{U.$ *finite* $\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\} \wedge (\forall\, i{\in}\{i \in I.\ U\ i \neq$ *topspace* $(S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}UNIV - I.\ U\ i = \{$*undefined*$\})\}$

using *hi'* **by** *auto*
qed
qed
also have ... = $(\bigcup I'\in\{I'.\ finite\ I' \wedge I' \subseteq I\}.\ \{U.\ (\forall i\in I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall i\in I - I'.\ U\ i = topspace\ (S\ i)) \wedge (\forall i\in UNIV - I.\ U\ i = \{undefined\})\})$
by *auto*
also have *countable* ...
proof(*rule countable-UN*[*OF countable-Collect-finite-subset*[*OF assms*(*1*)]])
fix I'
assume $I' \in \{I'.\ finite\ I' \wedge I' \subseteq I\}$
hence *hi'*:*finite* I' $I' \subseteq I$ **by** *auto*
have $(\lambda U\ i.\ if\ i \in I'\ then\ U\ i\ else\ undefined)$ ' $\{U.\ (\forall i\in I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall i\in I - I'.\ U\ i = topspace\ (S\ i)) \wedge (\forall i\in UNIV - I.\ U\ i = \{undefined\})\} \subseteq (\Pi_E\ i\in I'.\ \mathcal{O}\ i)$
by *auto*
moreover have *countable* ...
using *hi'* **by**(*auto intro*!: *countable-PiE ho*)
ultimately have *countable* $((\lambda U\ i.\ if\ i \in I'\ then\ U\ i\ else\ undefined)$ ' $\{U.\ (\forall i\in I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall i\in I - I'.\ U\ i = topspace\ (S\ i)) \wedge (\forall i\in UNIV - I.\ U\ i = \{undefined\})\})$
by(*simp add*: *countable-subset*)
moreover have *inj-on* $(\lambda U\ i.\ if\ i \in I'\ then\ U\ i\ else\ undefined)$ $\{U.\ (\forall i\in I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall i\in I - I'.\ U\ i = topspace\ (S\ i)) \wedge (\forall i\in UNIV - I.\ U\ i = \{undefined\})\}$
(**is** *inj-on ?f ?X*)
proof
fix $x\ y$
assume *hxy*: $x \in ?X$ $y \in ?X$ $?f\ x = ?f\ y$
show $x = y$
proof
fix i
consider $i \in I'$ | $i \in I - I'$ | $i \in UNIV - I$
using *hi'*(*2*) **by** *blast*
then show $x\ i = y\ i$
proof *cases*
case *i*:*1*
then show *?thesis*
using *fun-cong*[*OF hxy*(*3*),*of i*] **by** *auto*
next
case *i*:*2*
then show *?thesis*
using *hxy*(*1*,*2*) **by** *auto*
next
case *i*:*3*
then show *?thesis*
using *hxy*(*1*,*2*) **by** *auto*
qed
qed
qed

ultimately show *countable* $\{U.\ (\forall i{\in}I'.\ U\ i \in \mathcal{O}\ i) \land (\forall i{\in}I - I'.\ U\ i = topspace\ (S\ i)) \land (\forall i{\in}UNIV - I.\ U\ i = \{undefined\})\}$
using *countable-image-inj-on* **by** *auto*
qed
finally show *countable* $\{U.\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\} \land (\forall i{\in}\{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i) \land (\forall i{\in}UNIV - I.\ U\ i = \{undefined\})\}$ **.**
qed
finally show *?thesis* **.**
qed
next
show *base-in* (*product-topology S I*) $\{\Pi_E\ i{\in}I.\ U\ i \mid U.\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\} \land (\forall i{\in}\{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i)\}$
(**is** *base-in* (*product-topology S I*) *?X*)
unfolding *base-in-def*
proof *safe*
fix *U*
assume *openin* (*product-topology S I*) *U*
then have $\forall x{\in}U.\ \exists\ Ux.\ finite\ \{i \in I.\ Ux\ i \neq topspace\ (S\ i)\} \land (\forall i{\in}I.\ openin\ (S\ i)\ (Ux\ i)) \land x \in Pi_E\ I\ Ux \land Pi_E\ I\ Ux \subseteq U$
by(*simp add*: *openin-product-topology-alt*)
hence $\exists\ Ux.\ \forall x{\in}U.\ finite\ \{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\} \land (\forall i{\in}I.\ openin\ (S\ i)\ (Ux\ x\ i)) \land x \in Pi_E\ I\ (Ux\ x) \land Pi_E\ I\ (Ux\ x) \subseteq U$
by(*rule bchoice*)
then obtain *Ux* **where** *hui*:
$\bigwedge x.\ x \in U \Longrightarrow finite\ \{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}\ \bigwedge x\ i.\ x \in U \Longrightarrow i \in I \Longrightarrow openin\ (S\ i)\ (Ux\ x\ i)\ \bigwedge x.\ x \in U \Longrightarrow x \in Pi_E\ I\ (Ux\ x)\ \bigwedge x.\ x \in U \Longrightarrow Pi_E\ I\ (Ux\ x) \subseteq U$
by *fastforce*
then have *1*:$\forall x{\in}U.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}.\ \exists \mathcal{U}xj.\ \mathcal{U}xj \subseteq \mathcal{O}\ i \land Ux\ x\ i = \bigcup \mathcal{U}xj$
using *ho*[*simplified base-in-def*] **by** (*metis* (*no-types*, *lifting*) *mem-Collect-eq*)

have $\forall x{\in}U.\ \exists \mathcal{U}xj.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}.\ \mathcal{U}xj\ i \subseteq \mathcal{O}\ i \land Ux\ x\ i = \bigcup\ (\mathcal{U}xj\ i)$
by(*standard*, *rule bchoice*) (*use 1* **in** *simp*)
hence $\exists \mathcal{U}xj.\ \forall x{\in}U.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}.\ \mathcal{U}xj\ x\ i \subseteq \mathcal{O}\ i \land Ux\ x\ i = \bigcup\ (\mathcal{U}xj\ x\ i)$
by(*rule bchoice*)
then obtain $\mathcal{U}xj$ **where**
$\forall x{\in}U.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}.\ \mathcal{U}xj\ x\ i \subseteq \mathcal{O}\ i \land Ux\ x\ i = \bigcup\ (\mathcal{U}xj\ x\ i)$
by *auto*
hence *huxj*: $\bigwedge x\ i.\ x \in U \Longrightarrow i \in \{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\} \Longrightarrow \mathcal{U}xj\ x\ i \subseteq \mathcal{O}\ i$
$\bigwedge x\ i.\ x \in U \Longrightarrow i \in \{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\} \Longrightarrow Ux\ x\ i = \bigcup\ (\mathcal{U}xj\ x\ i)$
by *blast+*
show $\exists \mathcal{U}.\ U = \bigcup \mathcal{U} \land \mathcal{U} \subseteq\ ?X$
proof(*intro exI*[**where** $x{=}\{\Pi_E\ i{\in}I.\ K\ i \mid K.\ \exists x{\in}U.\ finite\ \{i \in I.\ Ux\ x$

$i \neq$ *topspace* $(S\ i)\} \wedge (\forall i{\in}\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i \in \mathcal{U}xj\ x\ i) \wedge (\forall i{\in}UNIV\ -\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i =$ *topspace* $(S\ i))\}]$ *conjI*)

show $U = \bigcup \{\Pi_E\ i{\in}I.\ K\ i \mid K.\ \exists x{\in}U.$ *finite* $\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\} \wedge (\forall i{\in}\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i \in \mathcal{U}xj\ x\ i) \wedge (\forall i{\in}UNIV\ -\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i =$ *topspace* $(S\ i))\}$

proof *safe*

fix x

assume *hxu*: $x \in U$

have $\forall i{\in}\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ Ux\ x\ i = \bigcup (\mathcal{U}xj\ x\ i)$

using *huxj*[*OF hxu*] **by** *blast*

hence $\forall i{\in}\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ \exists Uxj.\ Uxj \in \mathcal{U}xj\ x\ i \wedge x\ i \in Uxj$

using *hui*(*3*)[*OF hxu*] **by** *auto*

hence $\exists Uxj.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ Uxj\ i \in \mathcal{U}xj\ x\ i \wedge x\ i \in Uxj\ i$

by(*rule bchoice*)

then obtain *Uxj* **where** *huxj'*:

$\bigwedge i.\ i \in \{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\} \Longrightarrow Uxj\ i \in \mathcal{U}xj\ x\ i$

$\bigwedge i.\ i \in \{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\} \Longrightarrow x\ i \in Uxj\ i$

by *auto*

define K **where** $K \equiv (\lambda i.$ *if* $i \in \{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}$ *then* $Uxj\ i$ *else topspace* $(S\ i))$

have $x \in (\Pi_E\ i{\in}I.\ K\ i)$

using *huxj'*(*2*) *hui*(*3*,*4*)[*OF hxu*] *openin-subset*[*OF hui*(*2*)[*OF hxu*]]

by(*auto simp*: *K-def PiE-def Pi-def*)

moreover have $\exists x{\in}U.$ *finite* $\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\} \wedge (\forall i{\in}\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i \in \mathcal{U}xj\ x\ i) \wedge (\forall i{\in}UNIV\ -\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i =$ *topspace* $(S\ i))$

by(*rule bexI*[*OF - hxu*], *rule conjI*,*simp add*: *hui*(*1*)[*OF hxu*]) (*use hui*(*2*) *hxu openin-subset huxj'*(*1*) *K-def* **in** *auto*)

ultimately show $x \in \bigcup \{\Pi_E\ i{\in}I.\ K\ i \mid K.\ \exists x{\in}U.$ *finite* $\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\} \wedge (\forall i{\in}\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i \in \mathcal{U}xj\ x\ i) \wedge (\forall i{\in}UNIV\ -\{i \in I.\ Ux\ x\ i \neq$ *topspace* $(S\ i)\}.\ K\ i =$ *topspace* $(S\ i))\}$

by *auto*

next

fix $x\ X\ K\ u$

assume *hu*: $x \in (\Pi_E\ i{\in}I.\ K\ i)$ $u \in U$ *finite* $\{i \in I.\ Ux\ u\ i \neq$ *topspace* $(S\ i)\}$ $\forall i{\in}\{i \in I.\ Ux\ u\ i \neq$ *topspace* $(S\ i)\}.\ K\ i \in \mathcal{U}xj\ u\ i$ $\forall i{\in}UNIV\ -\{i \in I.\ Ux\ u\ i \neq$ *topspace* $(S\ i)\}.\ K\ i =$ *topspace* $(S\ i)$

have $\bigwedge i.\ i \in \{i \in I.\ Ux\ u\ i \neq$ *topspace* $(S\ i)\} \Longrightarrow K\ i \subseteq Ux\ u\ i$

using *huxj*[*OF hu*(*2*)] *hu*(*4*) **by** *blast*

moreover have $\bigwedge i.\ i \in I - \{i \in I.\ Ux\ u\ i \neq$ *topspace* $(S\ i)\} \Longrightarrow K\ i = Ux\ u\ i$

using *hu*(*5*) **by** *auto*

ultimately have $\bigwedge i.\ i \in I \Longrightarrow K\ i \subseteq Ux\ u\ i$

by *blast*

thus $x \in U$

using *hui*(*4*)[*OF hu*(*2*)] *hu*(*1*) **by** *blast*

qed

next

show $\{\Pi_E\ i{\in}I.\ K\ i \mid K.\ \exists x{\in}U.\ \textit{finite}\ \{i \in I.\ Ux\ x\ i \neq \textit{topspace}\ (S\ i)\} \wedge (\forall i{\in}\{i \in I.\ Ux\ x\ i \neq \textit{topspace}\ (S\ i)\}.\ K\ i \in \mathcal{U}xj\ x\ i) \wedge (\forall i{\in}\textit{UNIV} - \{i \in I.\ Ux\ x\ i \neq \textit{topspace}\ (S\ i)\}.\ K\ i = \textit{topspace}\ (S\ i))\} \subseteq\ ?X$

proof

fix x

assume $x \in \{\Pi_E\ i{\in}I.\ K\ i \mid K.\ \exists x{\in}U.\ \textit{finite}\ \{i \in I.\ Ux\ x\ i \neq \textit{topspace}\ (S\ i)\} \wedge (\forall i{\in}\{i \in I.\ Ux\ x\ i \neq \textit{topspace}\ (S\ i)\}.\ K\ i \in \mathcal{U}xj\ x\ i) \wedge (\forall i{\in}\textit{UNIV} - \{i \in I.\ Ux\ x\ i \neq \textit{topspace}\ (S\ i)\}.\ K\ i = \textit{topspace}\ (S\ i))\}$

then obtain $u\ K$ **where** *hu*:

$x = (\Pi_E\ i{\in}I.\ K\ i)\ \ u \in U\ \textit{finite}\ \{i \in I.\ Ux\ u\ i \neq \textit{topspace}\ (S\ i)\}\ \forall i{\in}\{i \in I.\ Ux\ u\ i \neq \textit{topspace}\ (S\ i)\}.\ K\ i \in \mathcal{U}xj\ u\ i\ \forall i{\in}\textit{UNIV} - \{i \in I.\ Ux\ u\ i \neq \textit{topspace}\ (S\ i)\}.\ K\ i = \textit{topspace}\ (S\ i)$

by *auto*

have *hksubst*:$\{i \in I.\ K\ i \neq \textit{topspace}\ (S\ i)\} \subseteq \{i \in I.\ Ux\ u\ i \neq \textit{topspace}\ (S\ i)\}$

using *hu*(*5*) **by** *fastforce*

hence *finite* $\{i \in I.\ K\ i \neq \textit{topspace}\ (S\ i)\}$

using *hu*(*3*) **by** (*simp add*: *finite-subset*)

moreover have $\forall i{\in}\{i \in I.\ K\ i \neq \textit{topspace}\ (S\ i)\}.\ K\ i \in \mathcal{O}\ i$

using *huxj*(*1*)[*OF hu*(*2*)] *hu*(*4*) *hksubst*

by (*meson subsetD*)

ultimately show $x \in\ ?X$

using *hu*(*1*) **by** *auto*

qed

qed

next

fix $\mathcal{U}$

assume $\mathcal{U} \subseteq\ ?X$

have *openin* (*product-topology S I*) u **if** *hu*:$u \in \mathcal{U}$ **for** u

proof −

have *hu'*: $u \in\ ?X$

using ‹$\mathcal{U} \subseteq\ ?X$› *hu* **by** *auto*

then obtain U **where** *hU*:

$u = (\Pi_E\ i{\in}I.\ U\ i)\ \textit{finite}\ \{i \in I.\ U\ i \neq \textit{topspace}\ (S\ i)\}\ \forall i{\in}\{i \in I.\ U\ i \neq \textit{topspace}\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i$

by *auto*

define U' **where** $U' \equiv (\lambda i.\ \textit{if}\ i \in \{i \in I.\ U\ i \neq \textit{topspace}\ (S\ i)\}\ \textit{then}\ U\ i\ \textit{else}\ \textit{topspace}\ (S\ i))$

have *hU'*: $u = (\Pi_E\ i{\in}I.\ U'\ i)$

by(*auto simp*: *hU*(*1*) *U'-def PiE-def Pi-def*)

have *hUfinite* : *finite* $\{i.\ U'\ i \neq \textit{topspace}\ (S\ i)\}$

using *hU*(*2*) **by**(*auto simp*: *U'-def*)

have *hUoi*: $\forall i{\in}\{i.\ U'\ i \neq \textit{topspace}\ (S\ i)\}.\ U'\ i \in \mathcal{O}\ i$

using *hU*(*3*) **by**(*auto simp*: *U'-def*)

have *hUi*: $\forall i{\in}\{i.\ U'\ i \neq \textit{topspace}\ (S\ i)\}.\ i \in I$

using *hU*(*2*) **by**(*auto simp*: *U'-def*)

have *hallopen*:*openin* (*S i*) (*U' i*) **for** i

proof −

consider $i \in \{i.\ U'\ i \neq topspace\ (S\ i)\} \mid i \notin \{i.\ U'\ i \neq topspace\ (S\ i)\}$
by *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
using *hUoi ho(2)[of i] base-in-openin[of S i* $\mathcal{O}$ *i U' i] hUi*
by *auto*
next
case *2*
then have $U'\ i = topspace\ (S\ i)$ **by** *auto*
thus *?thesis* **by** *auto*
qed
qed
show *openin (product-topology S I) u*
using *hallopen hUfinite* **by**(*auto intro*!: *product-topology-basis simp*: *hU'*)
qed
thus *openin (product-topology S I)* $(\bigcup \mathcal{U})$
by *auto*
qed
next
show $\forall k \in \{Pi_E\ I\ U \mid U.\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall i \in \{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i)\}.\ \exists X.\ k = Pi_E\ I\ X \wedge (\forall i.\ openin\ (S\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (S\ i)\} \wedge \{i.\ X\ i \neq topspace\ (S\ i)\} \subseteq I$
proof
fix *k*
assume $k \in \{Pi_E\ I\ U \mid U.\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall i \in \{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i)\}$
then obtain *U* **where** *hu*:
$k = (\Pi_E\ i \in I.\ U\ i)\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\}\ \forall i \in \{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i$
by *auto*
define *X* **where** $X \equiv (\lambda i.\ if\ i \in \{i \in I.\ U\ i \neq topspace\ (S\ i)\}\ then\ U\ i\ else\ topspace\ (S\ i))$
have *hX1*: $k = (\Pi_E\ i \in I.\ X\ i)$
using *hu(1)* **by**(*auto simp*: *X-def PiE-def Pi-def*)
have *hX2*: *openin (S i) (X i)* **for** *i*
using *hu(3) base-in-openin[of S i - U i,OF ho(2)]*
by(*auto simp*: *X-def*)
have *hX3*: $finite\ \{i.\ X\ i \neq topspace\ (S\ i)\}$
using *hu(2)* **by**(*auto simp*: *X-def*)
have *hX4*: $\{i.\ X\ i \neq topspace\ (S\ i)\} \subseteq I$
by(*auto simp*: *X-def*)
show $\exists X.\ k = (\Pi_E\ i \in I.\ X\ i) \wedge (\forall i.\ openin\ (S\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (S\ i)\} \wedge \{i.\ X\ i \neq topspace\ (S\ i)\} \subseteq I$
using *hX1 hX2 hX3 hX4* **by**(*auto intro*!: *exI*[**where** *x=X*])
qed
qed
qed

lemma *product-topology-second-countable*:
assumes *countable I* **and** ⋀*i*. *i* ∈ *I* ⟹ *second-countable* (*S i*)
shows *second-countable* (*product-topology S I*)
using *product-topology-countable-base-in*[*OF assms*(*1*)] *assms*(*2*)
by(*fastforce simp*: *second-countable-base-in*)

lemma *second-countable-euclidean*[*simp*]:
second-countable (*euclidean* :: ′*a* :: *second-countable-topology topology*)
using *ex-countable-basis second-countable-def topological-basis-def* **by** *fastforce*

lemma *Cantor-Bendixon*:
assumes *second-countable X*
shows ∃ *U P*. *countable U* ∧ *openin X U* ∧ *perfect-set X P* ∧ *U* ∪ *P* = *topspace X* ∧ *U* ∩ *P* = {} ∧ (∀ *a*≠{}. *openin* (*subtopology X P*) *a* ⟶ *uncountable a*)
proof −
obtain 𝒪 **where** *o*: *countable* 𝒪 *base-in X* 𝒪
using *assms* **by**(*auto simp*: *second-countable-base-in*)
define *U* **where** *U* ≡ ⋃ {*u*∈𝒪. *countable u*}
define *P* **where** *P* ≡ *topspace X* − *U*
have *1*: *countable U*
using *o*(*1*) **by**(*auto simp*: *U-def intro*!: *countable-UN*[*of - id*,*simplified*])
have *2*: *openin X U*
using *base-in-openin*[*OF o*(*2*)] **by**(*auto simp*: *U-def*)
have *openin-c*:*countable v* ⟷ *v* ⊆ *U* **if** *openin X v* **for** *v*
proof
assume *countable v*
obtain 𝒰 **where** *v* = ⋃𝒰 𝒰 ⊆ 𝒪
using ‹*openin X v*› *o*(*2*) **by**(*auto simp*: *base-in-def*)
with ‹*countable v*› **have** ⋀*u*. *u* ∈ 𝒰 ⟹ *countable u*
by (*meson Sup-upper countable-subset*)
thus *v* ⊆ *U*
using ‹𝒰 ⊆ 𝒪› **by**(*auto simp*: ‹*v* = ⋃𝒰› *U-def*)
qed(*rule countable-subset*[*OF - 1*])
have *3*: *perfect-set X P*
proof(*rule perfect-setI*)
fix *x T*
assume *h*:*x* ∈ *P x* ∈ *T openin X T*
have *T-unc*:*uncountable T*
using *openin-c*[*OF h*(*3*)] *h*(*1*,*2*) **by**(*auto simp*: *P-def*)
obtain 𝒰 **where** *U*:*T* = ⋃𝒰 𝒰 ⊆ 𝒪
using *h*(*3*) *o*(*2*) **by**(*auto simp*: *base-in-def*)
then obtain *u* **where** *u*:*u* ∈ 𝒰 *uncountable u*
using *T-unc U-def h*(*3*) *openin-c* **by** *auto*
hence *uncountable* (*u* − {*x*}) **by** *simp*
hence ¬ (*u* − {*x*} ⊆ *U*)
using *1* **by** (*metis countable-subset*)
then obtain *y* **where** *y* ∈ *u* − {*x*} *y* ∉ *U*
by *blast*

thus ∃ *y*. *y* ≠ *x* ∧ *y* ∈ *T* ∧ *y* ∈ *P*
using *U u base-in-subset*[*OF o*(*2*),*of u*] **by**(*auto intro*!: *exI*[**where** *x*=*y*] *simp*:*P-def*)
qed(*use 2 P-def* **in** *auto*)
have *4* : *uncountable a* **if** *openin* (*subtopology X P*) *a a* ≠ {} **for** *a*
proof
assume *contable*:*countable a*
obtain *b* **where** *b*: *openin X b a* = *P* ∩ *b*
using ‹*openin* (*subtopology X P*) *a*› **by**(*auto simp*: *openin-subtopology*)
hence *uncountable b*
using *P-def openin-c that*(*2*) **by** *auto*
thus *False*
by (*metis 1 Diff-Int-distrib2 Int-absorb1 P-def b*(*1*) *b*(*2*) *contable countable-Int1 openin-subset uncountable-minus-countable*)
qed
show *?thesis*
using *1 2 3 4* **by**(*auto simp*: *P-def*)
qed

1.1.5 Separable Spaces

definition *dense-in* :: [*'a topology*, *'a set*] ⇒ *bool* **where**
dense-in S U ⟷ (*U* ⊆ *topspace S* ∧ (∀ *V*. *openin S V* ⟶ *V* ≠ {} ⟶ *U* ∩ *V* ≠ {}))

lemma *dense-in-def2*:
dense-in S U ⟷ (*U* ⊆ *topspace S* ∧ (*S closure-of U*) = *topspace S*)
using *dense-intersects-open* **by**(*auto simp*: *dense-in-def closure-of-subset-topspace in-closure-of*) *auto*

lemma *dense-in-topspace*[*simp*]: *dense-in S* (*topspace S*)
by(*auto simp*: *dense-in-def2*)

lemma *dense-in-subset*:
assumes *dense-in S U*
shows *U* ⊆ *topspace S*
using *assms* **by**(*simp add*: *dense-in-def*)

lemma *dense-in-nonempty*:
assumes *topspace S* ≠ {} *dense-in S U*
shows *U* ≠ {}
using *assms* **by**(*auto simp*: *dense-in-def*)

lemma *dense-inI*:
assumes *U* ⊆ *topspace S*
and ⋀*V*. *openin S V* ⟹ *V* ≠ {} ⟹ *U* ∩ *V* ≠ {}
shows *dense-in S U*
using *assms* **by**(*auto simp*: *dense-in-def*)

lemma *dense-in-infinite*:
assumes *t1-space X infinite* (*topspace X*) *dense-in X U*
shows *infinite U*
proof
assume *fin*: *finite U*
then have *closedin X U*
by (*metis assms*(*1*,*3*) *dense-in-def t1-space-closedin-finite*)
hence *X closure-of U* = *U*
by (*simp add*: *closure-of-eq*)
thus *False*
by (*metis assms*(*2*) *assms*(*3*) *dense-in-def2 fin*)
qed

lemma *dense-in-prod*:
assumes *dense-in S U* **and** *dense-in S′ U′*
shows *dense-in* (*prod-topology S S′*) (*U* × *U′*)
proof(*rule dense-inI*)
fix *V*
assume *h*:*openin* (*prod-topology S S′*) *V V* ≠ {}
then obtain *x y* **where** *hxy*:(*x*,*y*) ∈ *V* **by** *auto*
then obtain *V1 V2* **where** *hv12*:
openin S V1 openin S′ V2 x ∈ *V1 y* ∈ *V2 V1* × *V2* ⊆ *V*
using *h*(*1*) *openin-prod-topology-alt*[*of S S′ V*] **by** *blast*
hence *V1* ≠ {} *V2* ≠ {} **by** *auto*
hence *U* ∩ *V1* ≠ {} *U′* ∩ *V2* ≠ {}
using *assms hv12* **by**(*auto simp*: *dense-in-def*)
thus *U* × *U′* ∩ *V* ≠ {}
using *hv12* **by** *auto*
next
show *U* × *U′* ⊆ *topspace* (*prod-topology S S′*)
using *assms* **by**(*auto simp add*: *dense-in-def*)
qed

lemma *separable-space-def2*:*separable-space S* ⟷ (∃ *U*. *countable U* ∧ *dense-in S U*)
by(*auto simp*: *separable-space-def dense-in-def2*)

lemma *countable-space-separable-space*:
assumes *countable* (*topspace S*)
shows *separable-space S*
using *assms* **by**(*auto simp*: *separable-space-def2 intro*!: *exI*[**where** *x*=*topspace S*])

lemma *separable-space-prod*:
assumes *separable-space S* **and** *separable-space S′*
shows *separable-space* (*prod-topology S S′*)
proof −
obtain *U U′* **where**
countable U dense-in S U countable U′ dense-in S′ U′

using *assms* **by**(*auto simp*: *separable-space-def2*)
thus *?thesis*
by(*auto intro*!: *exI*[**where** $x=U \times U'$] *dense-in-prod simp*: *separable-space-def2*)
qed

lemma *dense-in-product*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *dense-in* (*T i*) (*U i*)
shows *dense-in* (*product-topology T I*) (Π_E $i \in I.$ *U i*)
proof(*rule dense-inI*)
fix *V*
assume *h*:*openin* (*product-topology T I*) *V* $V \neq \{\}$
then obtain *x* **where** *hx*:$x \in V$ **by** *auto*
then obtain *K* **where** *hk*:
finite $\{i \in I.\ K\ i \neq$ *topspace* $(T\ i)\}$ $\forall i \in I.$ *openin* (*T i*) (*K i*) $x \in (\Pi_E\ i \in I.\ K\ i)$ $(\Pi_E\ i \in I.\ K\ i) \subseteq V$
using *h*(*1*) *openin-product-topology-alt*[*of T I V*] **by** *auto*
hence $\bigwedge i.\ i \in I \Longrightarrow K\ i \neq \{\}$ **by** *auto*
hence $\bigwedge i.\ i \in I \Longrightarrow U\ i \cap K\ i \neq \{\}$
using *assms hk* **by**(*auto simp*: *dense-in-def*)
hence $(\Pi_E\ i \in I.\ U\ i) \cap (\Pi_E\ i \in I.\ K\ i) \neq \{\}$
by (*simp add*: *PiE-Int PiE-eq-empty-iff*)
thus $(\Pi_E\ i \in I.\ U\ i) \cap V \neq \{\}$
using *hk* **by** *auto*
next
show $(\Pi_E\ i \in I.\ U\ i) \subseteq$ *topspace* (*product-topology T I*)
using *assms* **by**(*auto simp*: *dense-in-def*)
qed

lemma *separable-countable-product*:
assumes *countable I* **and** $\bigwedge i.\ i \in I \Longrightarrow$ *separable-space* (*T i*)
shows *separable-space* (*product-topology T I*)
proof −
consider $\exists i \in I.\ T\ i =$ *trivial-topology* | $\bigwedge i.\ i \in I \Longrightarrow T\ i \neq$ *trivial-topology*
by *auto*
thus *?thesis*
proof *cases*
case *1*
then obtain *i* **where** *i*:$i \in I$ *topspace* $(T\ i) = \{\}$
by *auto*
show *?thesis*
unfolding *separable-space-def2 dense-in-def*
proof(*intro exI*[**where** $x=\{\}$] *conjI*)
show $\forall V.$ *openin* (*product-topology T I*) $V \longrightarrow V \neq \{\} \longrightarrow \{\} \cap V \neq \{\}$
proof *safe*
fix *V x*
assume *h*: *openin* (*product-topology T I*) *V* $x \in V$
from *i* **have** (*product-topology T I*) = *trivial-topology*
using *product-topology-trivial-iff* **by** *auto*
with *h*(*1*) **have** $V = \{\}$

```
        by simp
      thus x ∈ {}
        using h(2) by auto
    qed
  qed auto
next
  case 2
 then have ∃ x. ∀ i∈I. x i ∈ topspace (T i) ∃ U. ∀ i∈I. countable (U i) ∧ dense-in
(T i) (U i)
   using assms(2) by(auto intro!: bchoice simp: separable-space-def2 ex-in-conv)
  then obtain x U where hxu:
   ⋀i. i ∈ I ⟹ x i ∈ topspace (T i) ⋀i. i ∈ I ⟹ countable (U i) ⋀i. i ∈ I
⟹ dense-in (T i) (U i)
    by auto
  define U' where U' ≡ (λJ i. if i ∈ J then U i else {x i})
  show ?thesis
    unfolding separable-space-def2
  proof(intro exI[where x=⋃{Π_E i∈I. U' J i | J. finite J ∧ J ⊆ I}] conjI)
    have (⋃{ Π_E i∈I. U' J i | J. finite J ∧ J ⊆ I}) = (⋃ ((λJ. Π_E i∈I. U' J
i) ` {J. finite J ∧ J ⊆ I}))
      by auto
    also have countable ...
    proof(rule countable-UN)
      fix J
      assume hj:J ∈ {J. finite J ∧ J ⊆ I}
      have inj-on (λf. (λi∈J. f i, λi∈(I−J). f i)) (Π_E i∈I. U' J i)
      proof(rule inj-onI)
        fix f g
        assume h:f ∈ Pi_E I (U' J) g ∈ Pi_E I (U' J)
               (restrict f J, restrict f (I − J)) = (restrict g J, restrict g (I − J))
        then have ⋀i. i ∈ J ⟹ f i = g i ⋀i. i ∈(I−J) ⟹ f i = g i
          by(auto simp: restrict-def) meson+
        thus f = g
          using h(1,2) by(auto simp: U'-def) (meson PiE-ext)
      qed
      moreover have countable ((λf. (λi∈J. f i, λi∈(I−J). f i)) ` (Π_E i∈I. U'
J i)) (is countable ?K)
      proof −
        have 1:?K ⊆ (Π_E i∈J. U i) × (Π_E i∈(I−J). {x i})
          using hj by(auto simp: U'-def PiE-def Pi-def)
        have 2:countable ...
        proof(rule countable-SIGMA)
          show countable (Pi_E J U)
            using hj hxu(2) by(auto intro!: countable-PiE)
        next
          have (Π_E i∈I − J. {x i}) = { λi∈I−J. x i }
            by(auto simp: PiE-def extensional-def restrict-def Pi-def)
          thus countable (Π_E i∈I − J. {x i})
            by simp
```

```
          qed
          show ?thesis
            by(rule countable-subset[OF 1 2])
        qed
        ultimately show countable (Π_E i∈I. U′ J i)
          by(simp add: countable-image-inj-eq)
      qed(rule countable-Collect-finite-subset[OF assms(1)])
      finally show countable (⋃{ Π_E i∈I. U′ J i | J. finite J ∧ J ⊆ I}) .
    next
     show dense-in (product-topology T I) (⋃ {Π_E i∈I. U′ J i |J. finite J ∧ J ⊆
I})
      proof(rule dense-inI)
        fix V
        assume h:openin (product-topology T I) V V ≠ {}
        then obtain y where hx:y ∈ V by auto
        then obtain K where hk:
         finite {i ∈ I. K i ≠ topspace (T i)} ⋀i. i∈I ⟹ openin (T i) (K i) y ∈
(Π_E i∈I. K i) (Π_E i∈I. K i) ⊆ V
          using h(1) openin-product-topology-alt[of T I V] by auto
        hence 3:⋀i. i ∈ I ⟹ K i ≠ {} by auto
         hence 4:i ∈ {i ∈ I. K i ≠ topspace (T i)} ⟹ K i ∩ U′ {i ∈ I. K i ≠
topspace (T i)} i ≠ {} for i
          using hxu(3)[of i] hk(2)[of i] by(auto simp: U′-def dense-in-def)
        have ∃z. ∀i∈{i ∈ I. K i ≠ topspace (T i)}. z i ∈ K i ∩ U′ {i ∈ I. K i ≠
topspace (T i)} i
          by(rule bchoice) (use 4 in auto)
        then obtain z where hz: ∀i∈{i ∈ I. K i ≠ topspace (T i)}. z i ∈ K i ∩
U′ {i ∈ I. K i ≠ topspace (T i)} i
          by auto
        have 5: i ∉ {i ∈ I. K i ≠ topspace (T i)} ⟹ i ∈ I ⟹ x i ∈ K i for i
          using hxu(1)[of i] by auto
        have (λi. if i ∈ {i ∈ I. K i ≠ topspace (T i)} then z i else if i ∈ I then x i
else undefined) ∈ (Π_E i∈I. U′ {i ∈ I. K i ≠ topspace (T i)} i) ∩ (Π_E i∈I. K i)
          using 4 5 hz by(auto simp: U′-def)
        thus ⋃ {Pi_E I (U′ J) |J. finite J ∧ J ⊆ I} ∩ V ≠ {}
          using hk(1,4) by blast
      next
        have ⋀J. J ⊆ I ⟹ (Π_E i∈I. U′ J i) ⊆ topspace (product-topology T I)
             using hxu by(auto simp: dense-in-def U′-def PiE-def Pi-def) (metis
subsetD)
       thus (⋃ {Π_E i∈I. U′ J i |J. finite J ∧ J ⊆ I}) ⊆ topspace (product-topology
T I)
          by auto
      qed
    qed
  qed
qed

lemma separable-finite-product:
```

assumes *finite I* **and** $\bigwedge i.\ i \in I \Longrightarrow$ *separable-space* (*T i*)
shows *separable-space* (*product-topology T I*)
using *separable-countable-product*[*OF countable-finite*[*OF assms*(*1*)]] *assms* **by** *auto*

1.1.6 G_δ Set

lemma *gdelta-inD*:
assumes *gdelta-in S A*
shows $\exists \mathcal{U}.\ \mathcal{U} \neq \{\} \wedge$ *countable* $\mathcal{U} \wedge (\forall b{\in}\mathcal{U}.$ *openin S b*$) \wedge A = \bigcap \mathcal{U}$
using *assms* **unfolding** *gdelta-in-def relative-to-def intersection-of-def*
by (*metis IntD1 Int-insert-right-if1 complete-lattice-class.Inf-insert countable-insert empty-not-insert inf.absorb-iff2 mem-Collect-eq openin-topspace*)

lemma *gdelta-inD′*:
assumes *gdelta-in S A*
shows $\exists$ *U*. ($\forall$ *n::nat*. *openin S* (*U n*)) $\wedge A = \bigcap$ (*range U*)
proof−
obtain $\mathcal{U}$ **where** *h*:$\mathcal{U} \neq \{\}$ *countable* $\mathcal{U}$ $\bigwedge b.\ b{\in}\mathcal{U} \Longrightarrow$ *openin S b* $A = \bigcap \mathcal{U}$
using *gdelta-inD*[*OF assms*] **by** *metis*
show *?thesis*
using *range-from-nat-into*[*OF h*(*1*,*2*)] *h*(*3*,*4*)
by(*auto intro*!: *exI*[**where** *x=from-nat-into* $\mathcal{U}$])
qed

lemma *gdelta-in-continuous-map*:
assumes *continuous-map X Y f gdelta-in Y a*
shows *gdelta-in X* (*f* −‘ *a* ∩ *topspace X*)
proof −
obtain *Ua* **where** *u*:
Ua ≠ {} *countable Ua* $\bigwedge b.\ b \in Ua \Longrightarrow$ *openin Y b a* = $\bigcap$ *Ua*
using *gdelta-inD*[*OF assms*(*2*)] **by** *metis*
then have *0*:*f* −‘ *a* ∩ *topspace X* = $\bigcap$ {*f* −‘ *b* ∩ *topspace X*|*b*. *b* ∈ *Ua*}
by *auto*
have *1*: {*f* −‘ *b* ∩ *topspace X* |*b*. *b* ∈ *Ua*} ≠ {}
using *u*(*1*) **by** *simp*
have *2*:*countable* {*f* −‘ *b* ∩ *topspace X*|*b*. *b* ∈ *Ua*}
using *u* **by** (*simp add*: *Setcompr-eq-image*)
have *3*:$\bigwedge c.\ c \in$ {*f* −‘ *b* ∩ *topspace X*|*b*. *b* ∈ *Ua*} $\Longrightarrow$ *openin X c*
using *assms u*(*3*) **by** *blast*
show *?thesis*
by (*metis* (*mono-tags*, *lifting*) *0 1 2 3 gdelta-in-Inter open-imp-gdelta-in*)
qed

lemma *g-delta-of-inj-open-map*:
assumes *open-map X Y f inj-on f* (*topspace X*) *gdelta-in X a*
shows *gdelta-in Y* (*f* ‘ *a*)
proof −
obtain *Ua* **where** *u*:

Ua ≠ {} *countable Ua* ⋀*b. b* ∈ *Ua* ⟹ *openin X b a* = ⋂ *Ua*
using *gdelta-inD*[*OF assms*(*3*)] **by** *metis*
then obtain *j* **where** *j* ∈ *Ua* **by** *auto*
have *f ' a* = *f '* ⋂ *Ua* **by**(*simp add*: *u*(*4*))
also have ... = ⋂ ((') *f ' Ua*)
using *u openin-subset* **by**(*auto intro*!: *image-INT*[*OF assms*(*2*) - ‹*j* ∈ *Ua*›,*of id*,*simplified*])
also have ... = ⋂ {*f ' u*|*u. u* ∈ *Ua*} **by** *auto*
finally have *0*: *f ' a* = ⋂ {*f ' u* |*u. u* ∈ *Ua*} **.**
have *1*:{*f ' u* |*u. u* ∈ *Ua*} ≠ {}
using *u*(*1*) **by** *auto*
have *2*:*countable* {*f ' u* |*u. u* ∈ *Ua*}
using *u*(*2*) **by** (*simp add*: *Setcompr-eq-image*)
have *3*: ⋀*c. c* ∈ {*f ' u* |*u. u* ∈ *Ua*} ⟹ *openin Y c*
using *assms*(*1*) *u*(*3*) **by**(*auto simp*: *open-map-def*)
show *?thesis*
by (*metis* (*no-types*, *lifting*) *0 1 2 3 gdelta-in-Inter open-imp-gdelta-in*)
qed

lemma *gdelta-in-prod*:
assumes *gdelta-in X A gdelta-in Y B*
shows *gdelta-in* (*prod-topology X Y*) (*A* × *B*)
proof −
obtain *Ua Ub* **where** *hu*:
Ua ≠ {} *countable Ua* ⋀*b. b* ∈ *Ua* ⟹ *openin X b A* = ⋂ *Ua*
Ub ≠ {} *countable Ub* ⋀*b. b* ∈ *Ub* ⟹ *openin Y b B* = ⋂ *Ub*
by (*meson gdelta-inD assms*)
then have *0*:*A* × *B* = ⋂ {*a* × *b* | *a b. a* ∈ *Ua* ∧ *b* ∈ *Ub*} **by** *blast*
have *1*: {*a* × *b* | *a b. a* ∈ *Ua* ∧ *b* ∈ *Ub*} ≠ {}
using *hu*(*1*,*5*) **by** *auto*
have *2*: *countable* {*a* × *b* | *a b. a* ∈ *Ua* ∧ *b* ∈ *Ub*}
proof −
have *countable* ((λ(*x*, *y*). *x* × *y*) ' (*Ua* × *Ub*))
using *hu*(*2*,*6*) **by**(*auto intro*!: *countable-image*[*of Ua* × *Ub* λ(*x*,*y*). *x* × *y*])
moreover have ... = {*a* × *b* | *a b. a* ∈ *Ua* ∧ *b* ∈ *Ub*} **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
have *3*: ⋀*c. c* ∈ {*a* × *b* | *a b. a* ∈ *Ua* ∧ *b* ∈ *Ub*} ⟹ *openin* (*prod-topology X Y*) *c*
using *hu*(*3*,*7*) **by**(*auto simp*: *openin-prod-Times-iff*)
show *?thesis*
by (*metis* (*no-types*, *lifting*) *gdelta-in-Inter open-imp-gdelta-in 0 1 2 3*)
qed

corollary *gdelta-in-prod1*:
assumes *gdelta-in X A*
shows *gdelta-in* (*prod-topology X Y*) (*A* × *topspace Y*)
by(*auto intro*!: *gdelta-in-prod assms*)

corollary *gdelta-in-prod2*:
assumes *gdelta-in Y B*
shows *gdelta-in* (*prod-topology X Y*) (*topspace X* × *B*)
by(*auto intro*!: *gdelta-in-prod assms*)

lemma *continuous-map-imp-closed-graph′*:
assumes *continuous-map X Y f Hausdorff-space Y*
shows *closedin* (*prod-topology Y X*) ((λ*x*. (*f x*,*x*)) ‘ *topspace X*)
using *assms closed-map-def closed-map-paired-continuous-map-left* **by** *blast*

1.1.7 Continuous Maps on First Countable Topology

Generalized version of *Metric-space ?M ?d* ⟹ *eventually ?P* (*atin* (*Metric-space.mtopology ?M ?d*) *?a*) = (∀ σ. *range* σ ⊆ *?M* − {*?a*} ∧ *limitin* (*Metric-space.mtopology ?M ?d*) σ *?a sequentially* ⟶ (∀$_F$ *n in sequentially*. *?P* (σ *n*)))

lemma *eventually-atin-sequentially*:
assumes *first-countable X*
shows *eventually P* (*atin X a*) ⟷ (∀ σ. *range* σ ⊆ *topspace X* − {*a*} ∧ *limitin X* σ *a sequentially* ⟶ *eventually* (λ*n*. *P* (σ *n*)) *sequentially*)
proof *safe*
fix *an*
assume *h*:*eventually P* (*atin X a*) *range an* ⊆ *topspace X* − {*a*} *limitin X an a sequentially*
then obtain *U* **where** *U*: *openin X U a* ∈ *U* ∀ *x*∈*U* − {*a*}. *P x*
by (*auto simp*: *eventually-atin limitin-topspace*)
with *h*(*3*) **obtain** *N* **where** ∀ *n*≥*N*. *an n* ∈ *U*
by (*meson limitin-sequentially*)
with *U*(*3*) *h*(*2*) **show** ∀$_F$ *n in sequentially*. *P* (*an n*)
unfolding *eventually-sequentially* **by** *blast*
next
assume *h*:∀ *an*. *range an* ⊆ *topspace X* − {*a*} ∧ *limitin X an a sequentially* ⟶ (∀$_F$ *n in sequentially*. *P* (*an n*))
consider *a* ∉ *topspace X* | *a* ∈ *topspace X*
by *blast*
then show *eventually P* (*atin X a*)
proof *cases*
assume *a*:*a* ∈ *topspace X*
from *a assms* **obtain** *B′* **where** *B′*: *countable B′* ⋀*V*. *V* ∈ *B′* ⟹ *openin X V* ⋀*U*. *openin X U* ⟹ *a* ∈ *U* ⟹ (∃ *V* ∈ *B′*. *a* ∈ *V* ∧ *V* ⊆ *U*)
by(*fastforce simp*: *first-countable-def*)
define *B* **where** *B* ≡ {*V*∈*B′*. *a* ∈ *V*}
have *B*:⋀*V*. *V* ∈ *B* ⟹ *openin X V countable B B* ≠ {} ⋀*U*. *openin X U* ⟹ *a* ∈ *U* ⟹ (∃ *V* ∈ *B*. *a* ∈ *V* ∧ *V* ⊆ *U*)
using *B′ B′*(*3*)[*OF* - *a*] **by**(*fastforce simp*: *B-def*)+
define *An* **where** *An* ≡ (λ*n*. ⋂*i*≤*n*. *from-nat-into B i*)
have *a-in-An*:*a* ∈ *An n* **for** *n*
by (*metis* (*no-types*, *lifting*) *An-def B-def B*(*3*) *INT-I from-nat-into mem-Collect-eq*)
have *openAn*:⋀*n*. *openin X* (*An n*)
using *B* **by**(*auto simp*: *An-def from-nat-into*[*OF B*(*3*)] *openin-Inter*)

```
    have deqseq-An:decseq An
      by(fastforce simp: decseq-def An-def)
    have ∃ U. openin X U ∧ a ∈ U ∧ Ball (U − {a}) P
    proof(rule ccontr)
      assume ∄ U. openin X U ∧ a ∈ U ∧ Ball (U − {a}) P
      then have ⋀U. openin X U ⟹ a ∈ U ⟹ ∃ x ∈ U − {a}. ¬ P x
        by blast
      hence ∃ b∈An n − {a}. ¬ P b for n
        using openAn a-in-An by auto
      then obtain an where an: ⋀n. an n ∈ An n ⋀n. an n ≠ a ⋀n. ¬ P (an n)
        by (metis Diff-iff singletonI)
      have limitin X an a sequentially
        unfolding limitin-sequentially
      proof safe
        fix U
        assume openin X U a ∈ U
        then obtain V where V:a ∈ V V ⊆ U V ∈ B
          using B by meson
        then obtain N where V = from-nat-into B N
          by (metis B(2) from-nat-into-surj)
        hence ⋀n. n ≥ N ⟹ an n ∈ V
          using an(1) An-def by blast
        thus ∃ N. ∀ n≥N. an n ∈ U
          using V by blast
      qed fact
      hence 1:∀F n in sequentially. P (an n)
        using an(2) h an(1) openin-subset[OF openAn] by blast
      thus False
        using an(3) by simp
    qed
    thus ?thesis
      by(simp add: eventually-atin)
  qed(auto simp: eventually-atin)
qed

lemma continuous-map-iff-limit-seq:
  assumes first-countable X
  shows continuous-map X Y f ⟷ (∀ xn x. limitin X xn x sequentially ⟶ limitin
Y (λn. f (xn n)) (f x) sequentially)
  unfolding continuous-map-atin
proof safe
  fix xn x
  assume h:∀ x∈topspace X. limitin Y f (f x) (atin X x) limitin X xn x sequentially
  then have limfx: limitin Y f (f x) (atin X x)
    by(simp add: limitin-topspace)
  show limitin Y (λn. f (xn n)) (f x) sequentially
    unfolding limitin-sequentially
  proof safe
    fix U
```

assume *U*:*openin Y U f x* ∈ *U*
then have *h'*:⋀σ. *range* σ ⊆ *topspace X* − {*x*} ⟹ *x* ∈ *topspace X* ⟹ *limitin X* σ *x sequentially* ⟹ (∃ *N*. ∀ *n*≥*N*. *f* (σ *n*) ∈ *U*)
using *limfx* **by**(*auto simp*: *limitin-def eventually-atin-sequentially*[*OF assms*(*1*)] *eventually-sequentially*)
show ∃ *N*. ∀ *n*≥*N*. *f* (*xn n*) ∈ *U*
proof(*cases finite* {*n*. *xn n* ≠ *x*})
assume *finite* {*n*. *xn n* ≠ *x*}
then obtain *N* **where** ⋀*n*. *n* ≥ *N* ⟹ *xn n* = *x*
using *infinite-nat-iff-unbounded-le* **by** *blast*
then show *?thesis*
using *U* **by** *auto*
next
assume *inf*:*infinite* {*n*. *xn n* ≠ *x*}
obtain *n0* **where** *n0*:⋀*n*. *n* ≥ *n0* ⟹ *xn n* ∈ *topspace X*
by (*meson h*(*2*) *limitin-sequentially openin-topspace*)
have *inf'*:*infinite* ({*n*. *xn n* ≠ *x*} ∩ {*n0*..})
proof
have *1*:({*n*. *xn n* ≠ *x*} ∩ {*n0*..}) ∪ ({*n*. *xn n* ≠ *x*} ∩ {..<*n0*}) = {*n*. *xn n* ≠ *x*}
by *auto*
assume *finite* ({*n*. *xn n* ≠ *x*} ∩ {*n0*..})
then have *finite* (({*n*. *xn n* ≠ *x*} ∩ {*n0*..}) ∪ ({*n*. *xn n* ≠ *x*} ∩ {..<*n0*}))
by *auto*
with *inf* **show** *False*
unfolding *1* **by** *blast*
qed
define *a* **where** *a* ≡ *enumerate* ({*n*. *xn n* ≠ *x*} ∩ {*n0*..})
have *a*: *strict-mono a range a* = ({*n*. *xn n* ≠ *x*} ∩ {*n0*..})
using *range-enumerate*[*OF inf'*] *strict-mono-enumerate*[*OF inf'*]
by(*auto simp*: *a-def*)
have ∃ *N*. ∀ *n*≥*N*. *f* (*xn* (*a n*)) ∈ *U*
using *limitin-subsequence*[*OF a*(*1*) *h*(*2*)] *a*(*2*) *n0*
by(*auto intro*!: *h'* *limitin-topspace*[*OF h*(*2*)] *simp*: *comp-def*)
then obtain *N* **where** *N*:⋀*n*. *n* ≥ *N* ⟹ *f* (*xn* (*a n*)) ∈ *U*
by *blast*
show ∃ *N*. ∀ *n*≥*N*. *f* (*xn n*) ∈ *U*
proof(*auto intro*!: *exI*[**where** *x*=*a N*])
fix *n*
assume *n*:*n* ≥ *a N*
show *f* (*xn n*) ∈ *U*
proof (*cases xn n* = *x*)
assume *xn n* ≠ *x*
moreover have *n0* ≤ *n*
using *seq-suble*[*OF a*(*1*),*of N*] *n a*(*2*)
by (*metis Int-Collect atLeast-def dual-order.trans rangeI*)
ultimately obtain *n1* **where** *n1*:*n* = *a n1*
by (*metis* (*mono-tags*, *lifting*) *Int-Collect atLeast-def imageE mem-Collect-eq a*(*2*))

have *n1* ≥ *N*
using *strict-mono-less-eq*[*OF a*(*1*),*of N n1*] *n* **by**(*simp add*: *n1*)
thus *?thesis*
by(*auto intro*!: *N simp*: *n1*)
qed(*auto simp*: *U*)
qed
qed
qed(*auto intro*!: *limitin-topspace limfx*)
next
fix *x*
assume *h*:∀ *xn x*. *limitin X xn x sequentially* ⟶ *limitin Y* (λ*n*. *f* (*xn n*)) (*f x*) *sequentially x* ∈ *topspace X*
then have *f x* ∈ *topspace Y*
by (*meson Abstract-Limits.limitin-const-iff limitin-topspace*)
thus *limitin Y f* (*f x*) (*atin X x*)
using *h* **by**(*auto simp*: *eventually-atin-sequentially*[*OF assms*(*1*)] *limitin-def*)
qed

1.1.8 Upper-Semicontinuous Functions

definition *upper-semicontinuous-map* :: ['*a topology*, '*a* ⇒ '*b* :: *linorder-topology*] ⇒ *bool* **where**
upper-semicontinuous-map X f ⟷ (∀ *a*. *openin X* {*x*∈*topspace X*. *f x* < *a*})

lemma *continuous-upper-semicontinuous*:
assumes *continuous-map X* (*euclidean* :: ('*b* :: *linorder-topology*) *topology*) *f*
shows *upper-semicontinuous-map X f*
unfolding *upper-semicontinuous-map-def*
proof *safe*
fix *a* :: '*b*
have ∗:*openin euclidean U* ⟹ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*} **for** *U*
using *assms* **by**(*simp add*: *continuous-map*)
have *openin euclidean* {..<*a*} **by** *auto*
with ∗[*of* {..<*a*}] **show** *openin X* {*x* ∈ *topspace X*. *f x* < *a*} **by** *auto*
qed

lemma *upper-semicontinuous-map-iff-closed*:
upper-semicontinuous-map X f ⟷ (∀ *a*. *closedin X* {*x*∈*topspace X*. *f x* ≥ *a*})
proof −
have {*x* ∈ *topspace X*. *f x* < *a*} = *topspace X* − {*x* ∈ *topspace X*. *f x* ≥ *a*} **for** *a*
by *auto*
thus *?thesis*
by (*simp add*: *closedin-def upper-semicontinuous-map-def*)
qed

lemma *upper-semicontinuous-map-real-iff*:
fixes *f* :: '*a* ⇒ *real*
shows *upper-semicontinuous-map X f* ⟷ *upper-semicontinuous-map X* (λ*x*.

ereal (*f x*))
  unfolding *upper-semicontinuous-map-def*
proof *safe*
  fix *a* :: *ereal*
  assume *h*:∀ *a*::*real*. *openin X* {*x* ∈ *topspace X*. *f x* < *a*}
  consider *a* = − ∞ | *a* = ∞ | *a* ≠ − ∞ ∧ *a* ≠ ∞ **by** *auto*
  then show *openin X* {*x* ∈ *topspace X*. *ereal* (*f x*) < *a*}
  proof *cases*
    case *3*
    then have *ereal* (*f x*) < *a* ⟷ *f x* < *real-of-ereal a* **for** *x*
      by (*metis ereal-less-eq*(*3*) *linorder-not-less real-of-ereal.elims*)
    thus *?thesis*
      using *h* **by** *simp*
  qed *simp-all*
next
  fix *a* :: *real*
  assume *h*:∀ *a*::*ereal*. *openin X* {*x* ∈ *topspace X*. *ereal* (*f x*) < *a*}
  then have *openin X* {*x* ∈ *topspace X*. *ereal* (*f x*) < *ereal a*}
    by *blast*
  moreover have*ereal* (*f x*) < *real-of-ereal a* ⟷ *f x* < *a* **for** *x*
    by *auto*
  ultimately show *openin X* {*x* ∈ *topspace X*. *f x* < *a*} **by** *auto*
qed

1.1.9 Lower-Semicontinuous Functions

definition *lower-semicontinuous-map* :: [*'a topology*, *'a* ⇒ *'b* :: *linorder-topology*] ⇒ *bool* **where**
lower-semicontinuous-map X f ⟷ (∀ *a*. *openin X* {*x*∈*topspace X*. *a* < *f x*})

lemma *continuous-lower-semicontinuous*:
  assumes *continuous-map X* (*euclidean* :: (*'b* :: *linorder-topology*) *topology*) *f*
  shows *lower-semicontinuous-map X f*
  unfolding *lower-semicontinuous-map-def*
proof *safe*
  fix *a* :: *'b*
  have ∗:*openin euclidean U* ⟹ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*} **for** *U*
    using *assms* **by**(*simp add*: *continuous-map*)
  have *openin euclidean* {*a*<..} **by** *auto*
  with ∗[*of* {*a*<..}] **show** *openin X* {*x* ∈ *topspace X*. *a* < *f x*} **by** *auto*
qed

lemma *lower-semicontinuous-map-iff-closed*:
 lower-semicontinuous-map X f ⟷ (∀ *a*. *closedin X* {*x*∈*topspace X*. *f x* ≤ *a*})
proof −
  have {*x* ∈ *topspace X*. *a* < *f x*} = *topspace X* − {*x* ∈ *topspace X*. *f x* ≤ *a*} **for** *a*
    by *auto*
  thus *?thesis*

by (*simp add*: *closedin-def lower-semicontinuous-map-def*)
qed

lemma *lower-semicontinuous-map-real-iff*:
fixes $f :: 'a \Rightarrow real$
shows *lower-semicontinuous-map* $X\ f \longleftrightarrow$ *lower-semicontinuous-map* $X\ (\lambda x.\ ereal\ (f\ x))$
unfolding *lower-semicontinuous-map-def*
proof *safe*
fix $a :: ereal$
assume $h{:}\forall a{::}real.\ openin\ X\ \{x \in topspace\ X.\ a < f\ x\}$
consider $a = -\infty \mid a = \infty \mid a \neq -\infty \wedge a \neq \infty$ **by** *auto*
then show $openin\ X\ \{x \in topspace\ X.\ a < ereal\ (f\ x)\}$
proof *cases*
case *3*
then have $a < ereal\ (f\ x) \longleftrightarrow real\text{-}of\text{-}ereal\ a < f\ x$ **for** x
by (*metis ereal-less-eq*(*3*) *linorder-not-less real-of-ereal.elims*)
thus *?thesis*
using *h* **by** *simp*
qed *simp-all*
next
fix $a :: real$
assume $h{:}\forall a{::}ereal.\ openin\ X\ \{x \in topspace\ X.\ a < ereal\ (f\ x)\}$
then have $openin\ X\ \{x \in topspace\ X.\ ereal\ (f\ x) > ereal\ a\}$
by *blast*
moreover have $ereal\ (f\ x) > real\text{-}of\text{-}ereal\ a \longleftrightarrow a < f\ x$ **for** x
by *auto*
ultimately show $openin\ X\ \{x \in topspace\ X.\ f\ x > a\}$ **by** *auto*
qed

1.2 Lemmas for Measure Theory

1.2.1 Lemmas for Measurable Sets

lemma *measurable-preserve-sigma-sets*:
assumes $sets\ M = sigma\text{-}sets\ \Omega\ S$ $S \subseteq Pow\ \Omega$
$\bigwedge a.\ a \in S \Longrightarrow f\ `\ a \in sets\ N$ $inj\text{-}on\ f\ (space\ M)$ $f\ `\ space\ M \in sets\ N$
and $b \in sets\ M$
shows $f\ `\ b \in sets\ N$
proof −
have $b \in sigma\text{-}sets\ \Omega\ S$
using *assms*(*1*,*6*) **by** *simp*
thus *?thesis*
proof *induction*
case (*Basic a*)
then show *?case* **by**(*rule assms*(*3*))
next
case *Empty*
then show *?case* **by** *simp*
next

case (*Compl a*)
moreover have Ω = *space M*
by (*metis assms*(*1*) *assms*(*2*) *sets.sets-into-space sets.top sigma-sets-into-sp sigma-sets-top subset-antisym*)
ultimately show *?case*
by (*metis Diff-subset assms*(*2*) *assms*(*4*) *assms*(*5*) *inj-on-image-set-diff sets.Diff sigma-sets-into-sp*)
next
case (*Union a*)
then show *?case*
by (*simp add*: *image-UN*)
qed
qed

inductive-set *sigma-sets-cinter* :: *'a set* ⇒ *'a set set* ⇒ *'a set set*
for *sp* :: *'a set* **and** *A* :: *'a set set*
where
Basic-c[*intro*, *simp*]: *a* ∈ *A* ⟹ *a* ∈ *sigma-sets-cinter sp A*
| *Top-c*[*simp*]: *sp* ∈ *sigma-sets-cinter sp A*
| *Inter-c*: (⋀*i*::*nat*. *a i* ∈ *sigma-sets-cinter sp A*) ⟹ (⋂ *i*. *a i*) ∈ *sigma-sets-cinter sp A*
| *Union-c*: (⋀*i*::*nat*. *a i* ∈ *sigma-sets-cinter sp A*) ⟹ (⋃ *i*. *a i*) ∈ *sigma-sets-cinter sp A*

inductive-set *sigma-sets-cinter-dunion* :: *'a set* ⇒ *'a set set* ⇒ *'a set set*
for *sp* :: *'a set* **and** *A* :: *'a set set*
where
Basic-cd[*intro*, *simp*]: *a* ∈ *A* ⟹ *a* ∈ *sigma-sets-cinter-dunion sp A*
| *Top-cd*[*simp*]: *sp* ∈ *sigma-sets-cinter-dunion sp A*
| *Inter-cd*: (⋀*i*::*nat*. *a i* ∈ *sigma-sets-cinter-dunion sp A*) ⟹ (⋂ *i*. *a i*) ∈ *sigma-sets-cinter-dunion sp A*
| *Union-cd*: (⋀*i*::*nat*. *a i* ∈ *sigma-sets-cinter-dunion sp A*) ⟹ *disjoint-family a* ⟹ (⋃ *i*. *a i*) ∈ *sigma-sets-cinter-dunion sp A*

lemma *sigma-sets-cinter-dunion-subset*: *sigma-sets-cinter-dunion sp A* ⊆ *sigma-sets-cinter sp A*
proof *safe*
fix *x*
assume *x* ∈ *sigma-sets-cinter-dunion sp A*
then show *x* ∈ *sigma-sets-cinter sp A*
by *induction* (*auto intro*!: *Union-c Inter-c*)
qed

lemma *sigma-sets-cinter-into-sp*:
assumes *A* ⊆ *Pow sp x* ∈ *sigma-sets-cinter sp A*
shows *x* ⊆ *sp*
using *assms*(*2*) **by** *induction* (*use assms*(*1*) *subsetD* **in** *blast*)+

lemma *sigma-sets-cinter-dunion-into-sp*:

assumes *A* ⊆ *Pow sp x* ∈ *sigma-sets-cinter-dunion sp A*
shows *x* ⊆ *sp*
using *assms*(*2*) **by** *induction* (*use assms*(*1*) *subsetD* **in** *blast*)+

lemma *sigma-sets-cinter-int*:
assumes *a* ∈ *sigma-sets-cinter sp A b* ∈ *sigma-sets-cinter sp A*
shows *a* ∩ *b* ∈ *sigma-sets-cinter sp A*
proof −
have *1*:*a* ∩ *b* = (⋂ *i*::*nat*. *if i* = *0 then a else b*) **by** *auto*
show *?thesis*
unfolding *1* **by**(*rule Inter-c*,*use assms* **in** *auto*)
qed

lemma *sigma-sets-cinter-dunion-int*:
assumes *a* ∈ *sigma-sets-cinter-dunion sp A b* ∈ *sigma-sets-cinter-dunion sp A*
shows *a* ∩ *b* ∈ *sigma-sets-cinter-dunion sp A*
proof −
have *1*:*a* ∩ *b* = (⋂ *i*::*nat*. *if i* = *0 then a else b*) **by** *auto*
show *?thesis*
unfolding *1* **by**(*rule Inter-cd*,*use assms* **in** *auto*)
qed

lemma *sigma-sets-cinter-un*:
assumes *a* ∈ *sigma-sets-cinter sp A b* ∈ *sigma-sets-cinter sp A*
shows *a* ∪ *b* ∈ *sigma-sets-cinter sp A*
proof −
have *1*:*a* ∪ *b* = (⋃ *i*::*nat*. *if i* = *0 then a else b*) **by** *auto*
show *?thesis*
unfolding *1* **by**(*rule Union-c*,*use assms* **in** *auto*)
qed

lemma *sigma-sets-eq-cinter-dunion*:
assumes *metrizable-space X*
shows *sigma-sets* (*topspace X*) {*U*. *openin X U*} = *sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*}
proof *safe*
fix *a*
interpret *sa*: *sigma-algebra topspace X sigma-sets* (*topspace X*) {*U*. *openin X U*}
by(*auto intro*!: *sigma-algebra-sigma-sets openin-subset*)
assume *a* ∈ *sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*}
then show *a* ∈ *sigma-sets* (*topspace X*) {*U*. *openin X U*}
by *induction auto*
next
have *c*:*sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*} ⊆ {*U*∈*sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*}. *topspace X* − *U* ∈ *sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*}}
proof
fix *a*

assume *a*: $a \in$ *sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*}
then show $a \in$ {*U* ∈ *sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*}. *topspace X* − *U* ∈ *sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*}}
proof *induction*
case *a*:(*Basic-cd a*)
then have *gdelta-in X* (*topspace X* − *a*)
by(*auto intro*!: *closed-imp-gdelta-in assms*)
from *gdelta-inD'*[*OF this*] **obtain** *U* **where** *U*:
⋀*n* :: *nat*. *openin X* (*U n*) *topspace X* − *a* = ⋂ (*range U*) **by** *auto*
show *?case*
using *a U*(*1*) **by**(*auto simp*: *U*(*2*) *intro*!: *Inter-cd*)
next
case *Top-cd*
then show *?case* **by** *auto*
next
case *ca*:(*Inter-cd a*)
define *b* **where** *b* ≡ (λ*n*. (*topspace X* − *a n*) ∩ (⋂*i*. *if i* < *n then a i else topspace X*))
have *bd*:*disjoint-family b*
using *nat-neq-iff* **by**(*fastforce simp*: *disjoint-family-on-def b-def*)
have *bin*:*b i* ∈ *sigma-sets-cinter-dunion* (*topspace X*) {*U*. *openin X U*} **for** *i*
unfolding *b-def*
apply(*rule sigma-sets-cinter-dunion-int*)
using *ca*(*2*)[*of i*]
apply *auto*[*1*]
apply(*rule Inter-cd*) **using** *ca* **by** *auto*
have *bun*:*topspace X* − (⋂ (*range a*)) = (⋃*i*. *b i*) (**is** *?lhs* = *?rhs*)
proof −
{ fix *x*
have *x* ∈ *?lhs* ⟷ *x* ∈ *topspace X* ∧ *x* ∈ (⋃*i*. *topspace X* − *a i*)
by *auto*
also have ... ⟷ *x* ∈ *topspace X* ∧ (∃*n*. *x* ∈ *topspace X* − *a n*)
by *auto*
also have ... ⟷ *x* ∈ *topspace X* ∧ (∃*n*. *x* ∈ *topspace X* − *a n* ∧ (∀*i*<*n*. *x* ∈ *a i*))
proof *safe*
fix *n*
assume *1*:*x* ∉ *a n x* ∈ *topspace X*
define *N* **where** *N* ≡ *Min* {*m*. *m* ≤ *n* ∧ *x* ∉ *a m*}
have *N*:*x* ∉ *a N N* ≤ *n*
using *linorder-class.Min-in*[*of* {*m*. *m* ≤ *n* ∧ *x* ∉ *a m*}] *1*
by(*auto simp*: *N-def*)
have *N'*:*x* ∈ *a i* **if** *i* < *N* **for** *i*
proof(*rule ccontr*)
assume *x* ∉ *a i*
then have *N* ≤ *i*
using *linorder-class.Min-le*[*of* {*m*. *m* ≤ *n* ∧ *x* ∉ *a m*} *i*] *that N*(*2*)
by(*auto simp*: *N-def*)
with *that* **show** *False* **by** *auto*

```
          qed
          show ∃n. x ∈ topspace X − a n ∧ (∀i<n. x ∈ a i)
            using N N′ by(auto intro!: exI[where x=N] 1)
        qed auto
        also have ... ⟷ x ∈ ?rhs
          by(auto simp: b-def)
        finally have x ∈ ?lhs ⟷ x ∈ ?rhs . }
      thus ?thesis by auto
    qed
    have ... ∈ sigma-sets-cinter-dunion (topspace X) {U. openin X U}
      by(rule Union-cd) (use bin bd in auto)
    thus ?case
      using Inter-cd[of a,OF ca(1)] by(auto simp: bun)
  next
    case ca:(Union-cd a)
    have topspace X − (⋃ (range a)) = (⋂i. (topspace X − a i))
      by simp
    have ... ∈ sigma-sets-cinter-dunion (topspace X) {U. openin X U}
      by(rule Inter-cd) (use ca in auto)
    then show ?case
      using Union-cd[of a,OF ca(1,2)] by auto
  qed
qed
fix a
assume a ∈ sigma-sets (topspace X) {U. openin X U}
then show a ∈ sigma-sets-cinter-dunion (topspace X) {U. openin X U}
proof induction
  case a:(Union a)
   define b where b ≡ (λn. a n ∩ (⋂i. if i < n then topspace X − a i else topspace X))
  have bd:disjoint-family b
     by(auto simp: disjoint-family-on-def b-def) (metis Diff-iff UnCI image-eqI linorder-neqE-nat mem-Collect-eq)
  have bin:b i ∈ sigma-sets-cinter-dunion (topspace X) {U. openin X U} for i
    unfolding b-def
    apply(rule sigma-sets-cinter-dunion-int)
    using a(2)[of i]
     apply auto[1]
    apply(rule Inter-cd) using c a by auto
  have bun:(⋃i. a i) = (⋃i. b i) (is ?lhs = ?rhs)
  proof −
    {
      fix x
      have x ∈ ?lhs ⟷ x ∈ topspace X ∧ x ∈ ?lhs
        using sigma-sets-cinter-dunion-into-sp[OF - a(2)]
        by (metis UN-iff subsetD subset-Pow-Union topspace-def)
      also have ... ⟷ x ∈ topspace X ∧ (∃n. x ∈ a n) by auto
       also have ... ⟷ x ∈ topspace X ∧ (∃n. x ∈ a n ∧ (∀i<n. x ∈ topspace X − a i))
```

```
        proof safe
          fix n
          assume 1:x ∈ topspace X x ∈ a n
          define N where N ≡ Min {m. m ≤ n ∧ x ∈ a m}
          have N:x ∈ a N N ≤ n
            using linorder-class.Min-in[of {m. m ≤ n ∧ x ∈ a m}] 1
            by(auto simp: N-def)
          have N':x ∉ a i if i < N for i
          proof(rule ccontr)
            assume ¬ x ∉ a i
            then have N ≤ i
              using linorder-class.Min-le[of {m. m ≤ n ∧ x ∈ a m} i] that N(2)
              by(auto simp: N-def)
            with that show False by auto
          qed
          show ∃n. x ∈ a n ∧ (∀i<n. x ∈ topspace X − a i)
            using N N' 1 by(auto intro!: exI[where x=N])
        qed auto
        also have ... ⟷ x ∈ ?rhs
        proof safe
          fix m
          assume x ∈ b m
          then show x ∈ topspace X ∃n. x ∈ a n ∧ (∀i<n. x ∈ topspace X − a i)
            by(auto intro!: exI[where x=m] simp: b-def)
        qed(auto simp: b-def)
        finally have x ∈ ?lhs ⟷ x ∈ ?rhs . }
      thus ?thesis by auto
    qed
    have ... ∈ sigma-sets-cinter-dunion (topspace X) {U. openin X U}
      by(rule Union-cd) (use bin bd in auto)
    thus ?case
      by(auto simp: bun)
  qed(use c in auto)
qed

lemma sigma-sets-eq-cinter:
  assumes metrizable-space X
  shows sigma-sets (topspace X) {U. openin X U} = sigma-sets-cinter (topspace X) {U. openin X U}
proof safe
  fix a
  interpret sa: sigma-algebra topspace X sigma-sets (topspace X) {U. openin X U}
    by(auto intro!: sigma-algebra-sigma-sets openin-subset)
  assume a ∈ sigma-sets-cinter (topspace X) {U. openin X U}
  then show a ∈ sigma-sets (topspace X) {U. openin X U}
    by induction auto
qed (use sigma-sets-cinter-dunion-subset sigma-sets-eq-cinter-dunion[OF assms] in auto)
```

1.2.2 Measurable Isomorphisms

definition *measurable-isomorphic-map*::[$'a$ *measure*, $'b$ *measure*, $'a \Rightarrow 'b$] $\Rightarrow$ *bool*
where
measurable-isomorphic-map M N f $\longleftrightarrow$ *bij-betw* f (*space* M) (*space* N) $\wedge$ $f \in M \rightarrow_M N$ $\wedge$ *the-inv-into* (*space* M) $f \in N \rightarrow_M M$

lemma *measurable-isomorphic-map-sets-cong*:
  assumes *sets* M = *sets* M' *sets* N = *sets* N'
  shows *measurable-isomorphic-map* M N f $\longleftrightarrow$ *measurable-isomorphic-map* M' N' f
  by(*simp add*: *measurable-isomorphic-map-def sets-eq-imp-space-eq*[*OF assms*(*1*)] *sets-eq-imp-space-eq*[*OF assms*(*2*)] *measurable-cong-sets*[*OF assms*] *measurable-cong-sets*[*OF assms*(*2*,*1*)])

lemma *measurable-isomorphic-map-surj*:
  assumes *measurable-isomorphic-map* M N f
  shows f ‘ *space* M = *space* N
  using *assms* **by**(*auto simp*: *measurable-isomorphic-map-def bij-betw-def*)

lemma *measurable-isomorphic-mapI*:
  assumes *bij-betw* f (*space* M) (*space* N) $f \in M \rightarrow_M N$ *the-inv-into* (*space* M) $f \in N \rightarrow_M M$
  shows *measurable-isomorphic-map* M N f
  using *assms* **by**(*simp add*: *measurable-isomorphic-map-def*)

lemma *measurable-isomorphic-map-byWitness*:
  assumes $f \in M \rightarrow_M N$ $g \in N \rightarrow_M M$ $\bigwedge x.\ x \in$ *space* $M \Longrightarrow g\ (f\ x) = x$ $\bigwedge x.\ x \in$ *space* $N \Longrightarrow f\ (g\ x) = x$
  shows *measurable-isomorphic-map* M N f
proof −
  have ∗:*bij-betw* f (*space* M) (*space* N)
   using *assms* **by**(*auto intro*!: *bij-betw-byWitness*[**where** f'=g] *dest*:*measurable-space*)
  show *?thesis*
  proof(*rule measurable-isomorphic-mapI*)
    have *the-inv-into* (*space* M) f x = g x **if** $x \in$ *space* N **for** x
      by (*metis* ∗ *assms*(*2*) *assms*(*4*) *bij-betw-imp-inj-on measurable-space that the-inv-into-f-f*)
    thus *the-inv-into* (*space* M) $f \in N \rightarrow_M M$
      using *measurable-cong assms*(*2*) **by** *blast*
  qed (*simp-all add*: ∗ *assms*(*1*))
qed

lemma *measurable-isomorphic-map-restrict-space*:
  assumes $f \in M \rightarrow_M N$ $\bigwedge A.\ A \in$ *sets* $M \Longrightarrow f$ ‘ $A \in$ *sets* N *inj-on* f (*space* M)
  shows *measurable-isomorphic-map* M (*restrict-space* N (f ‘ *space* M)) f
proof(*rule measurable-isomorphic-mapI*)
  show *bij-betw* f (*space* M) (*space* (*restrict-space* N (f ‘ *space* M)))
    by (*simp add*: *assms*(*2*,*3*) *inj-on-imp-bij-betw*)
next

show *f* ∈ *M* →$_M$ *restrict-space N* (*f* ' *space M*)
by (*simp add*: *assms*(*1*) *measurable-restrict-space2*)
next
show *the-inv-into* (*space M*) *f* ∈ *restrict-space N* (*f* ' *space M*) →$_M$ *M*
proof(*rule measurableI*)
show *x* ∈ *space* (*restrict-space N* (*f* ' *space M*)) ⟹ *the-inv-into* (*space M*) *f x* ∈ *space M* **for** *x*
by (*simp add*: *assms*(*2*,*3*) *the-inv-into-into*)
next
fix *A*
assume *A* ∈ *sets M*
have *the-inv-into* (*space M*) *f* −' *A* ∩ *space* (*restrict-space N* (*f* ' *space M*)) = *f* ' *A*
by (*simp add*: ‹*A* ∈ *sets M*› *assms*(*2*,*3*) *sets.sets-into-space the-inv-into-vimage*)
also note *assms*(*2*)[*OF* ‹*A* ∈ *sets M*›]
finally show *the-inv-into* (*space M*) *f* −' *A* ∩ *space* (*restrict-space N* (*f* ' *space M*)) ∈ *sets* (*restrict-space N* (*f* ' *space M*))
by (*simp add*: *assms*(*2*) *sets-restrict-space-iff*)
qed
qed

lemma *measurable-isomorphic-mapD'*:
assumes *measurable-isomorphic-map M N f*
shows ⋀*A*. *A* ∈ *sets M* ⟹ *f* ' *A* ∈ *sets N f* ∈ *M* →$_M$ *N*
∃ *g*. *bij-betw g* (*space N*) (*space M*) ∧ *g* ∈ *N* →$_M$ *M* ∧ (∀ *x* ∈ *space M*. *g* (*f x*) = *x*) ∧ (∀ *x*∈ *space N*. *f* (*g x*) = *x*) ∧ (∀ *A*∈*sets N*. *g* ' *A* ∈ *sets M*)
proof −
have *h*:*bij-betw f* (*space M*) (*space N*) *f* ∈ *M* →$_M$ *N the-inv-into* (*space M*) *f* ∈ *N* →$_M$ *M*
using *assms* **by**(*simp-all add*: *measurable-isomorphic-map-def*)
show *f* ' *A* ∈ *sets N* **if** *A* ∈ *sets M* **for** *A*
proof −
have *f* ' *A* = *the-inv-into* (*space M*) *f* −' *A* ∩ *space N*
using *the-inv-into-vimage*[*OF bij-betw-imp-inj-on*[*OF h*(*1*)] *sets.sets-into-space*[*OF that*]]
by(*simp add*: *bij-betw-imp-surj-on*[*OF h*(*1*)])
also have ... ∈ *sets N*
using *that h*(*3*) **by** *auto*
finally show *?thesis* **.**
qed
show *f* ∈ *M* →$_M$ *N*
using *assms* **by**(*simp add*: *measurable-isomorphic-map-def*)

show ∃ *g*. *bij-betw g* (*space N*) (*space M*) ∧ *g* ∈ *N* →$_M$ *M* ∧ (∀ *x* ∈ *space M*. *g* (*f x*) = *x*) ∧ (∀ *x*∈ *space N*. *f* (*g x*) = *x*) ∧ (∀ *A*∈*sets N*. *g* ' *A* ∈ *sets M*)
proof(*rule exI*[**where** *x*=*the-inv-into* (*space M*) *f*])
have ∗:*the-inv-into* (*space M*) *f* ' *A* ∈ *sets M* **if** *A* ∈ *sets N* **for** *A*
proof −
have ⋀*x*. *x* ∈ *space M* ⟹ *the-inv-into* (*space N*) (*the-inv-into* (*space M*) *f*)

x = *f x*
by (*metis bij-betw-imp-inj-on bij-betw-the-inv-into h*(*1*) *h*(*2*) *measurable-space the-inv-into-f-f*)
from *vimage-inter-cong*[*of space M - f A*,*OF this*] *the-inv-into-vimage*[*OF bij-betw-imp-inj-on*[*OF bij-betw-the-inv-into*[*OF h*(*1*)]] *sets.sets-into-space*[*OF that*]]
bij-betw-imp-surj-on[*OF bij-betw-the-inv-into*[*OF h*(*1*)]] *measurable-sets*[*OF h*(*2*) *that*]
show *?thesis*
by *fastforce*
qed
show *bij-betw* (*the-inv-into* (*space M*) *f*) (*space N*) (*space M*) ∧ *the-inv-into* (*space M*) *f* ∈ $N \rightarrow_M M$ ∧ (∀ *x*∈*space M*. *the-inv-into* (*space M*) *f* (*f x*) = *x*) ∧ (∀ *x*∈*space N*. *f* (*the-inv-into* (*space M*) *f x*) = *x*) ∧ (∀ *A*∈*sets N*. *the-inv-into* (*space M*) *f* ' *A* ∈ *sets M*)
using *bij-betw-the-inv-into*[*OF h*(*1*)]
by (*meson* ∗ *bij-betw-imp-inj-on f-the-inv-into-f-bij-betw h*(*1*) *h*(*3*) *the-inv-into-f-f*)
qed
qed

lemma *measurable-isomorphic-map-inv*:
assumes *measurable-isomorphic-map M N f*
shows *measurable-isomorphic-map N M* (*the-inv-into* (*space M*) *f*)
using *assms*[*simplified measurable-isomorphic-map-def*]
by(*auto intro*!: *measurable-isomorphic-map-byWitness*[**where** *g*=*f*] *bij-betw-the-inv-into f-the-inv-into-f-bij-betw*[*of f*] *bij-betw-imp-inj-on the-inv-into-f-f*)

lemma *measurable-isomorphic-map-comp*:
assumes *measurable-isomorphic-map M N f* **and** *measurable-isomorphic-map N L g*
shows *measurable-isomorphic-map M L* (*g* ∘ *f*)
proof −
obtain *f′ g′* **where**
[*measurable*]: $f' \in N \rightarrow_M M$ **and** *hf*:⋀*x*. *x*∈*space M* ⟹ *f′* (*f x*) = *x* ⋀*x*. *x*∈*space N* ⟹ *f* (*f′ x*) = *x*
and [*measurable*]: $g' \in L \rightarrow_M N$ **and** *hg*:⋀*x*. *x*∈*space N* ⟹ *g′* (*g x*) = *x* ⋀*x*. *x*∈*space L* ⟹ *g* (*g′ x*) = *x*
using *measurable-isomorphic-mapD′*[*OF assms*(*1*)] *measurable-isomorphic-mapD′*[*OF assms*(*2*)] **by** *metis*
have [*measurable*]: $f \in M \rightarrow_M N$ $g \in N \rightarrow_M L$
using *assms* **by**(*auto simp*: *measurable-isomorphic-map-def*)
from *hf hg measurable-space*[*OF* ‹$f \in M \rightarrow_M N$›] *measurable-space*[*OF* ‹$g' \in L \rightarrow_M N$›] **show** *?thesis*
by(*auto intro*!: *measurable-isomorphic-map-byWitness*[**where** *g*=*f′*∘*g′*])
qed

definition *measurable-isomorphic*::[*′a measure*, *′b measure*] ⇒ *bool* (**infixr** *measurable′-isomorphic 50*) **where**
M measurable-isomorphic N ⟷ (∃ *f*. *measurable-isomorphic-map M N f*)

lemma *measurable-isomorphic-sets-cong*:
assumes *sets M = sets M′ sets N = sets N′*
shows *M measurable-isomorphic N* $\longleftrightarrow$ *M′ measurable-isomorphic N′*
using *measurable-isomorphic-map-sets-cong*[*OF assms*]
by(*auto simp*: *measurable-isomorphic-def*)

lemma *measurable-isomorphicD*:
assumes *M measurable-isomorphic N*
shows $\exists f\, g.\ f \in M \to_M N \wedge g \in N \to_M M \wedge (\forall x{\in}space\ M.\ g\ (f\ x) = x) \wedge (\forall y{\in}space\ N.\ f\ (g\ y) = y) \wedge (\forall A{\in}sets\ M.\ f\ `\ A \in sets\ N) \wedge (\forall A{\in}sets\ N.\ g\ `\ A \in sets\ M)$
using *assms measurable-isomorphic-mapD′*[*of M N*]
by (*metis* (*mono-tags*, *lifting*) *measurable-isomorphic-def*)

lemma *measurable-isomorphic-cardinality-eq*:
assumes *M measurable-isomorphic N*
shows *space M* $\approx$ *space N*
by (*meson assms eqpoll-def measurable-isomorphic-def measurable-isomorphic-map-def*)

lemma *measurable-isomorphic-count-spaces*: *count-space A measurable-isomorphic count-space B* $\longleftrightarrow$ $A \approx B$
proof
assume $A \approx B$
then obtain *f* **where** *f*:*bij-betw f A B*
by(*auto simp*: *eqpoll-def*)
then show *count-space A measurable-isomorphic count-space B*
by(*auto simp*: *measurable-isomorphic-def measurable-isomorphic-map-def bij-betw-def the-inv-into-into intro*!: *exI*[**where** *x=f*])
qed(*use measurable-isomorphic-cardinality-eq* **in** *fastforce*)

lemma *measurable-isomorphic-byWitness*:
assumes $f \in M \to_M N$ $\bigwedge x.\ x{\in}space\ M \Longrightarrow g\ (f\ x) = x$
and $g \in N \to_M M$ $\bigwedge y.\ y{\in}space\ N \Longrightarrow f\ (g\ y) = y$
shows *M measurable-isomorphic N*
by(*auto simp*: *measurable-isomorphic-def assms intro*!: *exI*[**where** *x = f*] *measurable-isomorphic-map-byWitness*[**where** *g=g*])

lemma *measurable-isomorphic-refl*:
M measurable-isomorphic M
by(*auto intro*!: *measurable-isomorphic-byWitness*[**where** *f=id* **and** *g=id*])

lemma *measurable-isomorphic-sym*:
assumes *M measurable-isomorphic N*
shows *N measurable-isomorphic M*
using *assms measurable-isomorphic-map-inv*[*of M N*]
by(*auto simp*: *measurable-isomorphic-def*)

lemma *measurable-isomorphic-trans*:
assumes *M measurable-isomorphic N* **and** *N measurable-isomorphic L*

shows *M measurable-isomorphic L*
using *assms measurable-isomorphic-map-comp*[*of M N - L*]
by(*auto simp*: *measurable-isomorphic-def*)

lemma *measurable-isomorphic-empty*:
assumes *space M* = {} *space N* = {}
shows *M measurable-isomorphic N*
using *assms* **by**(*auto intro*!: *measurable-isomorphic-byWitness*[**where** *f*=*undefined* **and** *g*=*undefined*] *simp*: *measurable-empty-iff*)

lemma *measurable-isomorphic-empty1*:
assumes *space M* = {} *M measurable-isomorphic N*
shows *space N* = {}
using *measurable-isomorphicD*[*OF assms*(*2*)] **by**(*auto simp*: *measurable-empty-iff*[*OF assms*(*1*)])

lemma *measurable-ismorphic-empty2*:
assumes *space N* = {} *M measurable-isomorphic N*
shows *space M* = {}
using *measurable-isomorphic-sym*[*OF assms*(*2*)] *assms*(*1*)
by(*simp add*: *measurable-isomorphic-empty1*)

lemma *measurable-lift-product*:
assumes $\bigwedge i.\ i \in I \Longrightarrow f\ i \in (M\ i) \to_M (N\ i)$
shows $(\lambda x\ i.\ \mathit{if}\ i \in I\ \mathit{then}\ f\ i\ (x\ i)\ \mathit{else\ undefined}) \in (\Pi_M\ i{\in}I.\ M\ i) \to_M (\Pi_M\ i{\in}I.\ N\ i)$
using *measurable-space*[*OF assms*]
by(*auto intro*!: *measurable-PiM-single′ simp*: *assms measurable-PiM-component-rev space-PiM PiE-iff*)

lemma *measurable-isomorphic-map-lift-product*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *measurable-isomorphic-map* (*M i*) (*N i*) (*h i*)
shows *measurable-isomorphic-map* $(\Pi_M\ i{\in}I.\ M\ i)\ (\Pi_M\ i{\in}I.\ N\ i)\ (\lambda x\ i.\ \mathit{if}\ i \in I\ \mathit{then}\ h\ i\ (x\ i)\ \mathit{else\ undefined})$
proof −
obtain *h′* **where**
$\bigwedge i.\ i \in I \Longrightarrow h'\ i \in (N\ i) \to_M (M\ i)$ $\bigwedge i\ x.\ i \in I \Longrightarrow x{\in}\mathit{space}\ (M\ i) \Longrightarrow h'\ i\ (h\ i\ x) = x$ $\bigwedge i\ x.\ i \in I \Longrightarrow x{\in}\mathit{space}\ (N\ i) \Longrightarrow h\ i\ (h'\ i\ x) = x$
using *measurable-isomorphic-mapD′*(*3*)[*OF assms*] **by** *metis*
thus *?thesis*
by(*auto intro*!: *measurable-isomorphic-map-byWitness*[*OF measurable-lift-product*[*of I h M N*,*OF measurable-isomorphic-mapD′*(*2*)[*OF assms*]] *measurable-lift-product*[*of I h′ N M*,*OF* ‹$\bigwedge i.\ i \in I \Longrightarrow h'\ i \in (N\ i) \to_M (M\ i)$›]]
simp: *space-PiM PiE-iff extensional-def*)
qed

lemma *measurable-isomorphic-lift-product*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ (*M i*) *measurable-isomorphic* (*N i*)
shows $(\Pi_M\ i{\in}I.\ M\ i)$ *measurable-isomorphic* $(\Pi_M\ i{\in}I.\ N\ i)$

proof −
obtain *h* **where** ⋀*i*. *i* ∈ *I* ⟹ *measurable-isomorphic-map* (*M i*) (*N i*) (*h i*)
using *assms* **by**(*auto simp*: *measurable-isomorphic-def*) *metis*
thus *?thesis*
by(*auto intro*!: *measurable-isomorphic-map-lift-product exI*[**where** *x*=λ*x i*. *if i* ∈ *I then h i* (*x i*) *else undefined*] *simp*: *measurable-isomorphic-def*)
qed

https://math24.net/cantor-schroder-bernstein-theorem.html

lemma *Schroeder-Bernstein-measurable′*:
assumes *f* ‘ (*space M*) ∈ *sets N g* ‘ (*space N*) ∈ *sets M*
and *measurable-isomorphic-map M* (*restrict-space N* (*f* ‘ (*space M*))) *f* **and** *measurable-isomorphic-map N* (*restrict-space M* (*g* ‘ (*space N*))) *g*
shows ∃ *h*. *measurable-isomorphic-map M N h*
proof −
have *hset*:⋀*A*. *A* ∈ *sets M* ⟹ *f* ‘ *A* ∈ *sets* (*restrict-space N* (*f* ‘ *space M*))
⋀*A*. *A* ∈ *sets N* ⟹ *g* ‘ *A* ∈ *sets* (*restrict-space M* (*g* ‘ *space N*))
and *hfg*[*measurable*]:*f* ∈ *M* $\to_M$ *restrict-space N* (*f* ‘ *space M*)
g ∈ *N* $\to_M$ *restrict-space M* (*g* ‘ *space N*)
using *measurable-isomorphic-mapD′*(*1*,*2*)[*OF assms*(*3*)] *measurable-isomorphic-mapD′*(*1*,*2*)[*OF assms*(*4*)] *assms*(*1*,*2*)
by *auto*
have *hset2*:⋀*A*. *A* ∈ *sets M* ⟹ *f* ‘ *A* ∈ *sets N* ⋀*A*. *A* ∈ *sets N* ⟹ *g* ‘ *A* ∈ *sets M*
and *hfg2*[*measurable*]: *f* ∈ *M* $\to_M$ *N g* ∈ *N* $\to_M$ *M*
using *sets.Int-space-eq2*[*OF assms*(*1*)] *sets.Int-space-eq2*[*OF assms*(*2*)] *sets-restrict-space-iff*[*of f* ‘ *space M N*] *sets-restrict-space-iff*[*of g* ‘ *space N M*] *hset*
measurable-restrict-space2-iff[*of f M N*] *measurable-restrict-space2-iff*[*of g N M*] *hfg assms*(*1*,*2*)
by *auto*
have *bij1*:*bij-betw f* (*space M*) (*f* ‘ (*space M*)) *bij-betw g* (*space N*) (*g* ‘ (*space N*))
using *assms*(*3*,*4*) **by**(*auto simp*: *measurable-isomorphic-map-def space-restrict-space sets.Int-space-eq2*[*OF assms*(*1*)] *sets.Int-space-eq2*[*OF assms*(*2*)])
obtain *f′ g′* **where**
hfg1′[*measurable*]: *f′* ∈ *restrict-space N* (*f* ‘ (*space M*)) $\to_M$ *M g′* ∈ *restrict-space M* (*g* ‘ (*space N*)) $\to_M$ *N*
and *hfg′*:⋀*x*. *x*∈*space M* ⟹ *f′* (*f x*) = *x* ⋀*x*. *x*∈*f* ‘ *space M* ⟹ *f* (*f′ x*) = *x*
⋀*x*. *x*∈*space N* ⟹ *g′* (*g x*) = *x* ⋀*x*. *x*∈*g* ‘ *space N* ⟹ *g* (*g′ x*) = *x*
bij-betw f′ (*f* ‘ *space M*) (*space M*) *bij-betw g′* (*g* ‘ *space N*) (*space N*)
using *measurable-isomorphic-mapD′*(*3*)[*OF assms*(*3*)] *measurable-isomorphic-mapD′*(*3*)[*OF assms*(*4*)] *sets.Int-space-eq2*[*OF assms*(*1*)] *sets.Int-space-eq2*[*OF assms*(*2*)]
by (*metis space-restrict-space*)

have *hgfA*:(*g* ∘ *f*) ‘ *A* ∈ *sets M* **if** *A* ∈ *sets M* **for** *A*
using *hset2*(*2*)[*OF hset2*(*1*)[*OF that*]] **by**(*simp add*: *image-comp*)
define *An* **where** *An* ≡ (λ*n*. ((*g* ∘ *f*) ^^ *n*) ‘ (*space M* − *g* ‘ (*space N*)))
define *A* **where** *A* ≡ (⋃ *n*∈*UNIV*. *An n*)
have *An n* ∈ *sets M* **for** *n*

proof(*induction n*)
  case *0*
  thus *?case*
    using *hset2*[*OF sets.top*] **by**(*simp add*: *An-def*)
next
  case *ih*:(*Suc n*)
  have *An* (*Suc n*) = (*g* ∘ *f*) ' (*An n*)
    by(*auto simp add*: *An-def*)
  thus *?case*
    using *hgfA*[*OF ih*] **by** *simp*
qed
hence *Asets*:*A* ∈ *sets M*
  by(*simp add*: *A-def*)
have *Acompl*:*space M* − *A* ⊆ *g* ' *space N*
proof −
  have *space M* − *A* ⊆ *space M* − *An 0*
    by(*auto simp*: *A-def*)
  also have ... ⊆ *g* ' *space N*
    by(*auto simp*: *An-def*)
  finally show *?thesis* **.**
qed
define *h* **where** *h* ≡ (λ*x*. *if x* ∈ *A* ∪ (− *space M*) *then f x else g′ x*)
define *h′* **where** *h′* ≡ (λ*x*. *if x* ∈ *f* ' *A then f′ x else g x*)
have *xinA-iff*:*x* ∈ *A* ⟷ *h x* ∈ *f* ' *A* **if** *x* ∈ *space M* **for** *x*
proof
  assume *h x* ∈ *f* ' *A*
  show *x* ∈ *A*
  proof(*rule ccontr*)
    assume *x* ∉ *A*
    then have ⋀*n*. *x* ∉ *An n*
      by(*auto simp*: *A-def*)
    from *this*[*of 0*] **have** *x* ∈ *g* ' (*space N*)
      using *that* **by**(*auto simp*: *An-def*)
    have *g′ x* ∈ *f* ' *A*
      using ‹*h x* ∈ *f* ' *A*› ‹*x* ∉ *A*›
      by (*simp add*: *h-def that*)
    hence *g* (*g′ x*) ∈ (*g* ∘ *f*) ' *A*
      by *auto*
    hence *x* ∈ (*g* ∘ *f*) ' *A*
      using ‹*x* ∈ *g* ' (*space N*)› **by** (*simp add*: *hfg′*(*4*))
    then obtain *n* **where** *x* ∈ (*g* ∘ *f*) ' (*An n*)
      by(*auto simp*: *A-def*)
    hence *x* ∈ *An* (*Suc n*)
      by(*auto simp*: *An-def*)
    thus *False*
      using ‹⋀*n*. *x* ∉ *An n*› **by** *simp*
  qed
qed(*simp add*: *h-def*)

show *?thesis*
proof(*intro exI*[**where** $x=h$] *measurable-isomorphic-map-byWitness*[**where** $g=h'$])
have $\{x \in$ *space* $M.\ x \in A \cup (-$ *space* $M)\} \in$ *sets* M
using *sets.Int-space-eq2*[*OF Asets*] *Asets* **by** *simp*
moreover have $f \in$ *restrict-space* M $\{x.\ x \in A \cup -$ *space* $M\} \to_M N$
by (*simp add*: *measurable-restrict-space1*)
moreover have $g' \in$ *restrict-space* M $\{x.\ x \notin A \cup (-$ *space* $M)\} \to_M N$
proof −
have *sets* (*restrict-space* (*restrict-space* M (g ‘ *space* N)) $\{x.\ x \notin A \cup -$ *space* $M\}$) = *sets* (*restrict-space* M (g ‘ *space* $N \cap$ $\{x.\ x \notin A \cup -$ *space* $M\}$))
by(*simp add*: *sets-restrict-restrict-space*)
also have ... = *sets* (*restrict-space* M (g ‘ *space* $N \cap \{x.\ x \in$ *space* $M - A\}$))
by (*metis Compl-iff DiffE DiffI Un-iff*)
also have ... = *sets* (*restrict-space* M $\{x.\ x \in$ *space* $M - A\}$)
by (*metis Acompl le-inf-iff mem-Collect-eq subsetI subset-antisym*)
also have ... = *sets* (*restrict-space* M $\{x.\ x \notin A \cup (-$ *space* $M)\}$)
by (*metis Compl-iff DiffE DiffI Un-iff*)
finally have *sets* (*restrict-space* (*restrict-space* M (g ‘ *space* N)) $\{x.\ x \notin A \cup -$ *space* $M\}$) = *sets* (*restrict-space* M $\{x.\ x \notin A \cup -$ *space* $M\}$) **.**
from *measurable-cong-sets*[*OF this refl*] *measurable-restrict-space1* [*OF hfg1'*(*2*),*of* $\{x.\ x \notin A \cup -$ *space* $M\}$]
show *?thesis* **by** *auto*
qed
ultimately show $h \in M \to_M N$
by(*simp add*: *h-def measurable-If-restrict-space-iff*)
next
have $\{x \in$ *space* $N.\ x \in f$ ‘ $A\} \in$ *sets* N
using *sets.Int-space-eq2*[*OF hset2*(*1*)[*OF Asets*]] *hset2*(*1*)[*OF Asets*] **by** *simp*
moreover have $f' \in$ *restrict-space* N $\{x.\ x \in f$ ‘ $A\} \to_M M$
proof −
have *sets* (*restrict-space* (*restrict-space* N (f ‘ *space* M)) $\{x.\ x \in f$ ‘ $A\}$) = *sets* (*restrict-space* N (f ‘ *space* $M \cap \{x.\ x \in f$ ‘ $A\}$))
by(*simp add*: *sets-restrict-restrict-space*)
also have ... = *sets* (*restrict-space* N $\{x.\ x \in f$ ‘ $A\}$)
proof −
have f ‘ *space* $M \cap \{x.\ x \in f$ ‘ $A\} = \{x.\ x \in f$ ‘ $A\}$
using *sets.sets-into-space*[*OF Asets*] **by** *auto*
thus *?thesis* **by** *simp*
qed
finally have *sets* (*restrict-space* (*restrict-space* N (f ‘ *space* M)) $\{x.\ x \in f$ ‘ $A\}$) = *sets* (*restrict-space* N $\{x.\ x \in f$ ‘ $A\}$) **.**
from *measurable-cong-sets*[*OF this refl*] *measurable-restrict-space1* [*OF hfg1'*(*1*),*of* $\{x.\ x \in f$ ‘ $A\}$]
show *?thesis* **by** *auto*
qed
moreover have $g \in$ *restrict-space* N $\{x.\ x \notin f$ ‘ $A\} \to_M M$
by (*simp add*: *measurable-restrict-space1*)
ultimately show $h' \in N \to_M M$
by(*simp add*: *h'-def measurable-If-restrict-space-iff*)

```
  next
    fix x
    assume x ∈ space M
    then consider x ∈ A | x ∈ space M − A by auto
    thus h′ (h x) = x
    proof cases
      case xa:2
      hence h x ∉ f ‘ A
        using ‹x ∈ space M› xinA-iff by blast
      thus ?thesis
        using Acompl hfg′(4) xa by(auto simp add: h-def h′-def)
    qed(simp add: h-def h′-def ‹x ∈ space M› hfg′(1))
  next
    fix x
    assume x ∈ space N
    then consider x ∈ f ‘ A | x ∈ space N −  f ‘ A by auto
    thus h (h′ x) = x
    proof cases
      case hx:1
      hence x ∈ f ‘ (space M)
        using image-mono[OF sets.sets-into-space[OF Asets],of f] by auto
      have h′ x = f′ x
        using hx by(simp add: h′-def)
      also have ... ∈ A
        using hx sets.sets-into-space[OF Asets] hfg′(1) by auto
      finally show ?thesis
        using hfg′(2)[OF ‹x ∈ f ‘ (space M)›] hx by(auto simp: h-def h′-def)
    next
      case hx:2
      then have h′ x = g x
        by(simp add: h′-def)
      also have ... ∉ A
      proof(rule ccontr)
        assume ¬ g x ∉ A
        then have g x ∈ A by simp
        then obtain n where hg:g x ∈ An n by(auto simp: A-def)
        hence 0 < n using hx by(auto simp: An-def)
        then obtain n′ where [simp]:n = Suc n′
          using not0-implies-Suc by blast
        then have g x ∈ g ‘ f ‘  An n′
          using hg by(auto simp: An-def)
        hence x ∈ f ‘ An n′
         using inj-on-image-mem-iff[OF bij-betw-imp-inj-on[OF bij1(2)] ‹x ∈ space
N›,of f ‘ An n′]
            sets.sets-into-space[OF ‹An n′ ∈ sets M›] measurable-space[OF hfg2(1)]
by auto
        also have ... ⊆ f ‘ A
          by(auto simp: A-def)
        finally show False
```

using *hx* **by** *simp*
qed
finally show *?thesis*
using *hx hfg′(3)[OF ‹x ∈ space N›] measurable-space[OF hfg2(2) ‹x ∈ space N›]*
by(*auto simp*: *h-def h′-def*)
qed
qed
qed

lemma *Schroeder-Bernstein-measurable*:
assumes *f ∈ M →$_M$ N ⋀A. A ∈ sets M ⟹ f ‘ A ∈ sets N inj-on f (space M)*
and *g ∈ N →$_M$ M ⋀A. A ∈ sets N ⟹ g ‘ A ∈ sets M inj-on g (space N)*
shows *∃ h. measurable-isomorphic-map M N h*
using *Schroeder-Bernstein-measurable′[OF assms(2)[OF sets.top] assms(5)[OF sets.top] measurable-isomorphic-map-restrict-space[OF assms(1−3)] measurable-isomorphic-map-restrict-space assms(4−6)]]*
by *simp*

lemma *measurable-isomorphic-from-embeddings*:
assumes *M measurable-isomorphic (restrict-space N B) N measurable-isomorphic (restrict-space M A)*
and *A ∈ sets M B ∈ sets N*
shows *M measurable-isomorphic N*
proof −
obtain *f g* **where** *fg*:*measurable-isomorphic-map M (restrict-space N B) f measurable-isomorphic-map N (restrict-space M A) g*
using *assms(1,2)* **by**(*auto simp*: *measurable-isomorphic-def*)
have [*simp*]:*f ‘ space M = B g ‘ space N = A*
using *measurable-isomorphic-map-surj[OF fg(1)] measurable-isomorphic-map-surj[OF fg(2)] sets.sets-into-space[OF assms(3)] sets.sets-into-space[OF assms(4)]*
by(*auto simp*: *space-restrict-space*)
obtain *h* **where** *measurable-isomorphic-map M N h*
using *Schroeder-Bernstein-measurable′[of f M N g] assms(3,4) fg* **by** *auto*
thus *?thesis*
by(*auto simp*: *measurable-isomorphic-def*)
qed

lemma *measurable-isomorphic-antisym*:
assumes *B measurable-isomorphic (restrict-space C c) A measurable-isomorphic (restrict-space B b)*
and *c ∈ sets C b ∈ sets B C measurable-isomorphic A*
shows *C measurable-isomorphic B*
by(*rule measurable-isomorphic-from-embeddings[OF measurable-isomorphic-trans[OF assms(5,2)] assms(1) assms(3,4)]*)

lemma *countable-infinite-isomorphisc-to-nat-index*:
assumes *countable I* **and** *infinite I*
shows $(\Pi_M\ x{\in}I.\ M)$ *measurable-isomorphic* $(\Pi_M\ (x{::}nat){\in}UNIV.\ M)$

proof(*rule measurable-isomorphic-byWitness*[**where** *f*=*λx n. x* (*from-nat-into I n*) **and** *g*=*λx. λi*∈*I. x* (*to-nat-on I i*)])
show (*λx n. x* (*from-nat-into I n*)) ∈ (Π_M *x*∈*I. M*) $\rightarrow_M$ (Π_M (*x*::*nat*)∈*UNIV. M*)
by(*auto intro*!: *measurable-PiM-single′ measurable-component-singleton*[*OF from-nat-into*[*OF infinite-imp-nonempty*[*OF assms*(*2*)]]]])
(*simp add*: *PiE-iff infinite-imp-nonempty space-PiM from-nat-into*[*OF infinite-imp-nonempty*[*OF assms*(*2*)]]])
next
show (*λx. λi*∈*I. x* (*to-nat-on I i*)) ∈ (Π_M (*x*::*nat*)∈*UNIV. M*) $\rightarrow_M$ (Π_M *x*∈*I. M*)
by(*auto intro*!: *measurable-PiM-single′*)
next
show *x* ∈ *space* (Π_M *x*∈*I. M*) ⟹ (*λi*∈*I. x* (*from-nat-into I* (*to-nat-on I i*))) = *x* **for** *x*
by (*simp add*: *assms*(*1*) *restrict-ext space-PiM*)
next
show *y* ∈ *space* (Pi_M *UNIV* (*λx. M*)) ⟹ (*λn.* (*λi*∈*I. y* (*to-nat-on I i*)) (*from-nat-into I n*)) = *y* **for** *y*
by (*simp add*: *assms*(*1*) *assms*(*2*) *from-nat-into infinite-imp-nonempty*)
qed

lemma *PiM-PiM-isomorphic-to-PiM*:
(Π_M *i*∈*I.* Π_M *j*∈*J. M i j*) *measurable-isomorphic* (Π_M (*i,j*)∈*I*×*J. M i j*)
proof(*rule measurable-isomorphic-byWitness*[**where** *f*=*λx* (*i,j*). *if* (*i,j*) ∈ *I* × *J then x i j else undefined* **and** *g*=*λx i j. if i* ∉ *I then undefined j else if j* ∉ *J then undefined else x* (*i,j*)])
have [*simp*]: (*λω. ω a b*) ∈ (Π_M *i*∈*I.* Π_M *j*∈*J. M i j*) $\rightarrow_M$ *M a b* **if** *a* ∈ *I b* ∈ *J* **for** *a b*
using *measurable-component-singleton*[*OF that*(*1*),*of λi.* Π_M *j*∈*J. M i j*] *measurable-component-singleton*[*OF that*(*2*),*of M a*]
by *auto*
show (*λx* (*i, j*). *if* (*i, j*) ∈ *I* × *J then x i j else undefined*) ∈ (Π_M *i*∈*I.* Π_M *j*∈*J. M i j*) $\rightarrow_M$ (Π_M (*i,j*)∈*I*×*J. M i j*)
apply(*rule measurable-PiM-single′*)
apply *auto*[*1*]
apply(*auto simp*: *PiE-def Pi-def space-PiM extensional-def*;*meson*)
done
next
have [*simp*]: (*λω. ω* (*i, j*)) ∈ Pi_M (*I* × *J*) (*λ*(*i, j*). *M i j*) $\rightarrow_M$ *M i j* **if** *i* ∈ *I j* ∈ *J* **for** *i j*
using *measurable-component-singleton*[*of* (*i,j*) *I* × *J λ*(*i, j*). *M i j*] *that* **by** *auto*
show (*λx i j. if i* ∉ *I then undefined j else if j* ∉ *J then undefined else x* (*i, j*)) ∈ (Π_M (*i,j*)∈*I*×*J. M i j*) $\rightarrow_M$ (Π_M *i*∈*I.* Π_M *j*∈*J. M i j*)
by(*auto intro*!: *measurable-PiM-single′*) (*simp-all add*: *PiE-iff space-PiM extensional-def*)
next
show *x* ∈ *space* (Π_M *i*∈*I.* Π_M *j*∈*J. M i j*) ⟹ (*λi j. if i* ∉ *I then undefined j*

else if j ∉ J then undefined else case (i, j) of (i, j) ⇒ if (i, j) ∈ I × J then x i j else undefined) = x **for** *x*
  by *standard+ (auto simp: space-PiM PiE-def Pi-def extensional-def)*
next
 show *y ∈ space* (Π_M *(i,j)∈I×J. M i j) ⟹ (λ(i, j). if (i, j) ∈ I × J then if i ∉ I then undefined j else if j ∉ J then undefined else y (i, j) else undefined) = y* **for** *y*
  by *standard+ (auto simp: space-PiM PiE-def Pi-def extensional-def)*
qed

lemma *measurable-isomorphic-map-sigma-sets*:
 assumes *sets M = sigma-sets (space M) U measurable-isomorphic-map M N f*
 shows *sets N = sigma-sets (space N) ((‘) f ‘ U)*
proof −
 from *measurable-isomorphic-mapD′[OF assms(2)]*
 obtain *g* **where** *h*: *⋀A. A ∈ sets M ⟹ f ‘ A ∈ sets N f ∈ M* $\to_M$ *N bij-betw g (space N) (space M) g ∈ N* $\to_M$ *M ⋀x. x∈space M ⟹ g (f x) = x ⋀x. x∈space N ⟹ f (g x) = x ⋀A. A∈sets N ⟹ g ‘ A ∈ sets M*
  by *metis*
 interpret *s*: *sigma-algebra space N sigma-sets (space N) ((‘) f ‘ U)*
  by(*auto intro!*: *sigma-algebra-sigma-sets*) (*metis assms(1) h(2) measurable-space sets.sets-into-space sigma-sets-superset-generator subsetD*)
 show *?thesis*
 proof *safe*
  fix *x*
  assume *x ∈ sets N*
  from *h(7)[OF this] assms(1)*
  have *g ‘ x ∈ sigma-sets (space M) U* **by** *simp*
  hence *f ‘ (g ‘ x) ∈ sigma-sets (space N) ((‘) f ‘ U)*
  proof *induction*
   case *h*:(*Compl a*)
   have *f ‘ (space M − a) = f ‘ (space M) − f ‘ a*
     by(*rule inj-on-image-set-diff*[**where** *C=space M*], *insert assms h*) (*auto simp*: *measurable-isomorphic-map-def bij-betw-def sets.sets-into-space*)
   with *h* **show** *?case*
    by (*metis assms(2) measurable-isomorphic-map-surj s.Diff s.top*)
  qed (*auto simp*: *image-UN*)
  moreover have *f ‘ (g ‘ x) = x*
  using *sets.sets-into-space[OF ‹x ∈ sets N›] h(6)* **by**(*fastforce simp*: *image-def*)
  ultimately show *x ∈ sigma-sets (space N) ((‘) f ‘ U)* **by** *simp*
 next
  interpret *s′*: *sigma-algebra space M sigma-sets (space M) U*
   by(*simp add*: *assms(1)[symmetric] sets.sigma-algebra-axioms*)
  have *1*:*⋀x. x ∈ U ⟹ x ⊆ space M*
   by (*simp add*: *s′.sets-into-space*)
  fix *x*
  assume *assm*:*x ∈ sigma-sets (space N) ((‘) f ‘ U)*
  then show *x ∈ sets N*
   by *induction* (*auto simp*: *assms(1) h(1)*)

```
  qed
qed
```

1.2.3 Borel Spaces Genereted from Abstract Topologies

```
definition borel-of :: 'a topology ⇒ 'a measure where
borel-of X ≡ sigma (topspace X) {U. openin X U}

lemma emeasure-borel-of: emeasure (borel-of X) A = 0
  by (simp add: borel-of-def emeasure-sigma)

lemma borel-of-euclidean: borel-of euclidean = borel
  by(simp add: borel-of-def borel-def)

lemma space-borel-of: space (borel-of X) = topspace X
  by(simp add: space-measure-of-conv borel-of-def)

lemma sets-borel-of: sets (borel-of X) = sigma-sets (topspace X) {U. openin X
U}
  by (simp add: subset-Pow-Union topspace-def borel-of-def)

lemma sets-borel-of-closed: sets (borel-of X) = sigma-sets (topspace X) {U. closedin
X U}
  unfolding sets-borel-of
proof(safe intro!: sigma-sets-eqI)
  fix a
  assume a:openin X a
  have topspace X − (topspace X − a) ∈ sigma-sets (topspace X) {U. closedin X
U}
    by(rule sigma-sets.Compl) (use a in auto)
  thus a ∈ sigma-sets (topspace X) {U. closedin X U}
    using openin-subset[OF a] by (simp add: Diff-Diff-Int inf.absorb-iff2)
next
  fix b
  assume b:closedin X b
  have topspace X − (topspace X − b) ∈ sigma-sets (topspace X) {U. openin X
U}
    by(rule sigma-sets.Compl) (use b in auto)
  thus b ∈ sigma-sets (topspace X) {U. openin X U}
    using closedin-subset[OF b] by (simp add: Diff-Diff-Int inf.absorb-iff2)
qed

lemma borel-of-open:
  assumes openin X U
  shows U ∈ sets (borel-of X)
  using assms by (simp add: subset-Pow-Union topspace-def borel-of-def)

lemma borel-of-closed:
  assumes closedin X U
```

shows *U* ∈ *sets* (*borel-of X*)
using *assms sigma-sets.Compl*[*of topspace X* − *U topspace X*]
by (*simp add*: *closedin-def double-diff sets-borel-of*)

lemma(**in** *Metric-space*) *nbh-sets*[*measurable*]: (⋃ *a*∈*A*. *mball a e*) ∈ *sets* (*borel-of mtopology*)
by(*auto intro*!: *borel-of-open openin-clauses*(*3*))

lemma *borel-of-gdelta-in*:
assumes *gdelta-in X U*
shows *U* ∈ *sets* (*borel-of X*)
using *gdelta-inD*[*OF assms*] *borel-of-open*
by(*auto intro*!: *sets.countable-INT′*[*of - id*,*simplified*])

lemma *borel-of-subtopology*:
borel-of (*subtopology X U*) = *restrict-space* (*borel-of X*) *U*
proof(*rule measure-eqI*)
show *sets* (*borel-of* (*subtopology X U*)) = *sets* (*restrict-space* (*borel-of X*) *U*)
unfolding *restrict-space-eq-vimage-algebra′ sets-vimage-algebra sets-borel-of topspace-subtopology space-borel-of Int-commute*[*of U*]
proof(*rule sigma-sets-eqI*)
fix *a*
assume *a* ∈ *Collect* (*openin* (*subtopology X U*))
then obtain *T* **where** *openin X T a* = *T* ∩ *U*
by(*auto simp*: *openin-subtopology*)
show *a* ∈ *sigma-sets* (*topspace X* ∩ *U*) {(λ*x*. *x*) −‘ *A* ∩ (*topspace X* ∩ *U*) |*A*. *A* ∈ *sigma-sets* (*topspace X*) (*Collect* (*openin X*))}
using *openin-subset*[*OF* ‹*openin X T*›] ‹*a* = *T* ∩ *U*› **by**(*auto intro*!: *exI*[**where** *x*=*T*] ‹*openin X T*›)
next
fix *b*
assume *b* ∈ {(λ*x*. *x*) −‘ *A* ∩ (*topspace X* ∩ *U*) |*A*. *A* ∈ *sigma-sets* (*topspace X*) (*Collect* (*openin X*))}
then obtain *T* **where** *ht*:*b* = *T* ∩ (*topspace X* ∩ *U*) *T* ∈ *sigma-sets* (*topspace X*) (*Collect* (*openin X*))
by *auto*
hence *b* = *T* ∩ *U*
proof −
have *T* ⊆ *topspace X*
by(*rule sigma-sets-into-sp*[*OF - ht*(*2*)]) (*simp add*: *subset-Pow-Union topspace-def*)
thus *?thesis*
by(*auto simp*: *ht*(*1*))
qed
with *ht*(*2*) **show** *b* ∈ *sigma-sets* (*topspace X* ∩ *U*) (*Collect* (*openin* (*subtopology X U*)))
proof(*induction arbitrary*: *b U*)
case (*Basic a*)
then show *?case*

```
      by(auto simp: openin-subtopology)
    next
      case Empty
      then show ?case by simp
    next
      case ih:(Compl a)
      then show ?case
        by (simp add: Diff-Int-distrib2 sigma-sets.Compl)
    next
      case (Union a)
      then show ?case
        by (metis UN-extend-simps(4) sigma-sets.Union)
    qed
  qed
qed(simp add: emeasure-borel-of restrict-space-def emeasure-measure-of-conv)

lemma sets-borel-of-discrete-topology: sets (borel-of (discrete-topology I)) = sets
(count-space I)
  by (metis Pow-UNIV UNIV-eq-I borel-of-open borel-of-subtopology inf.absorb-iff2
openin-discrete-topology sets-count-space sets-restrict-space sets-restrict-space-count-space
subtopology-discrete-topology top-greatest)

lemma continuous-map-measurable:
  assumes continuous-map X Y f
  shows f ∈ borel-of X →M borel-of Y
proof(rule measurable-sigma-sets[OF sets-borel-of[of Y]])
  show {U. openin Y U} ⊆ Pow (topspace Y)
    by (simp add: subset-Pow-Union topspace-def)
next
  show f ∈ space (borel-of X) → topspace Y
    using continuous-map-image-subset-topspace[OF assms]
    by(auto simp: space-borel-of)
next
  fix U
  assume U ∈ {U. openin Y U}
  then have openin X (f −‘ U ∩ topspace X)
    using continuous-map[of X Y f] assms by auto
  thus f −‘ U ∩ space (borel-of X) ∈ sets (borel-of X)
    by(simp add: space-borel-of sets-borel-of)
qed

lemma upper-semicontinuous-map-measurable:
  fixes f :: 'a ⇒ 'b :: {linorder-topology, second-countable-topology}
  assumes upper-semicontinuous-map X f
  shows f ∈ borel-measurable (borel-of X)
 using assms by(auto intro!: borel-measurableI-less borel-of-open simp: space-borel-of
upper-semicontinuous-map-def)

lemma lower-semicontinuous-map-measurable:
```

fixes *f* :: *'a* ⇒ *'b* :: {*linorder-topology*, *second-countable-topology*}
assumes *lower-semicontinuous-map X f*
shows *f* ∈ *borel-measurable* (*borel-of X*)
using *assms* **by**(*auto intro*!: *borel-measurableI-greater borel-of-open simp*: *space-borel-of lower-semicontinuous-map-def*)

lemma *open-map-preserves-sets*:
assumes *open-map S T f inj-on f* (*topspace S*) *A* ∈ *sets* (*borel-of S*)
shows *f* ' *A* ∈ *sets* (*borel-of T*)
using *assms*(*3*)[*simplified sets-borel-of*]
proof(*induction*)
case (*Basic a*)
with *assms*(*1*) **show** *?case*
by(*auto simp*: *sets-borel-of open-map-def*)
next
case *Empty*
show *?case* **by** *simp*
next
case (*Compl a*)
moreover have *f* ' (*topspace S* − *a*) = *f* ' (*topspace S*) − *f* ' *a*
by (*metis Diff-subset assms*(*2*) *calculation*(*1*) *inj-on-image-set-diff sigma-sets-into-sp subset-Pow-Union topspace-def*)
moreover have *f* ' (*topspace S*) ∈ *sets* (*borel-of T*)
by (*meson assms*(*1*) *borel-of-open open-map-def openin-topspace*)
ultimately show *?case*
by *auto*
next
case (*Union a*)
then show *?case*
by (*simp add*: *image-UN*)
qed

lemma *open-map-preserves-sets'*:
assumes *open-map S* (*subtopology T* (*f* ' (*topspace S*))) *f inj-on f* (*topspace S*) *f* ' (*topspace S*) ∈ *sets* (*borel-of T*) *A* ∈ *sets* (*borel-of S*)
shows *f* ' *A* ∈ *sets* (*borel-of T*)
using *assms*(*4*)[*simplified sets-borel-of*]
proof(*induction*)
case (*Basic a*)
then have *openin* (*subtopology T* (*f* ' (*topspace S*))) (*f* ' *a*)
using *assms*(*1*) **by**(*auto simp*: *open-map-def*)
hence *f* ' *a* ∈ *sets* (*borel-of* (*subtopology T* (*f* ' (*topspace S*))))
by(*simp add*: *sets-borel-of*)
hence *f* ' *a* ∈ *sets* (*restrict-space* (*borel-of T*) (*f* ' (*topspace S*)))
by(*simp add*: *borel-of-subtopology*)
thus *?case*
by (*metis sets-restrict-space-iff assms*(*3*) *sets.Int-space-eq2*)
next
case *Empty*

show *?case* **by** *simp*
next
case (*Compl a*)
moreover have f ‘ (*topspace* $S - a$) = f ‘ (*topspace* S) − f ‘ a
by (*metis Diff-subset assms*(*2*) *calculation*(*1*) *inj-on-image-set-diff sigma-sets-into-sp subset-Pow-Union topspace-def*)
ultimately show *?case*
using *assms*(*3*) **by** *auto*
next
case (*Union a*)
then show *?case*
by (*simp add*: *image-UN*)
qed

Abstract topology version of *open* = *generate-topology ?X* ⟹ *borel* = *sigma UNIV ?X*.

lemma *borel-of-second-countable′*:
assumes *second-countable S* **and** *subbase-in S* $\mathcal{U}$
shows *borel-of S* = *sigma* (*topspace S*) $\mathcal{U}$
unfolding *borel-of-def*
proof(*rule sigma-eqI*)
show {*U*. *openin S U*} ⊆ *Pow* (*topspace S*)
by (*simp add*: *subset-Pow-Union topspace-def*)
next
show $\mathcal{U}$ ⊆ *Pow* (*topspace S*)
using *subbase-in-subset*[*OF assms*(*2*)] **by** *auto*
next
interpret *s*: *sigma-algebra topspace S sigma-sets* (*topspace S*) $\mathcal{U}$
using *subbase-in-subset*[*OF assms*(*2*)] **by**(*auto intro*!: *sigma-algebra-sigma-sets*)
obtain $\mathcal{O}$ **where** *ho*: *countable* $\mathcal{O}$ *base-in S* $\mathcal{O}$
using *assms*(*1*) **by**(*auto simp*: *second-countable-base-in*)
show *sigma-sets* (*topspace S*) {*U*. *openin S U*} = *sigma-sets* (*topspace S*) $\mathcal{U}$
proof(*rule sigma-sets-eqI*)
fix *U*
assume *U* ∈ {*U*. *openin S U*}
then have *generate-topology-on* $\mathcal{U}$ *U*
using *assms*(*2*) **by**(*simp add*: *subbase-in-def openin-topology-generated-by-iff*)
thus *U* ∈ *sigma-sets* (*topspace S*) $\mathcal{U}$
proof *induction*
case (*UN K*)
with *ho*(*2*) **obtain** *V* **where** *hv*:
$\bigwedge k.\ k \in K \Longrightarrow V\ k \subseteq \mathcal{O}$ $\bigwedge k.\ k \in K \Longrightarrow \bigcup (V\ k) = k$
by(*simp add*: *base-in-def openin-topology-generated-by-iff*[*symmetric*] *assms*(*2*)[*simplified subbase-in-def*,*symmetric*]) *metis*
define $\mathcal{U}k$ **where** $\mathcal{U}k = (\bigcup k{\in}K.\ V\ k)$
have *0*:*countable* $\mathcal{U}k$
using *hv* **by**(*auto intro*!: *countable-subset*[*OF - ho*(*1*)] *simp*: $\mathcal{U}k$*-def*)
have $\bigcup \mathcal{U}k = (\bigcup A{\in}\mathcal{U}k.\ A)$ **by** *auto*
also have ... = $\bigcup$ *K*

unfolding $\mathcal{U}k$*-def UN-simps* **by**(*simp add*: *hv*(*2*))
finally have *1*:$\bigcup$ $\mathcal{U}k$ = $\bigcup$ *K* **.**
have $\forall\, b{\in}\mathcal{U}k.\ \exists\, k{\in}K.\ b \subseteq k$
using *hv* **by** (*auto simp*: $\mathcal{U}k$*-def*)
then obtain V' **where** *hv'*: $\bigwedge b.\ b \in \mathcal{U}k \Longrightarrow V'\ b \in K$ **and** $\bigwedge b.\ b \in \mathcal{U}k \Longrightarrow b \subseteq V'\ b$
by *metis*
then have $(\bigcup b{\in}\mathcal{U}k.\ V'\ b) \subseteq \bigcup K$ $\bigcup\mathcal{U}k \subseteq (\bigcup b{\in}\mathcal{U}k.\ V'\ b)$
by *auto*
then have $\bigcup K = (\bigcup b{\in}\mathcal{U}k.\ V'\ b)$
unfolding *1* **by** *auto*
also have $\ldots \in$ *sigma-sets* (*topspace S*) $\mathcal{U}$
using *hv' UN* **by**(*auto intro*!: *s.countable-UN' simp*: *0*)
finally show $\bigcup K \in$ *sigma-sets* (*topspace S*) $\mathcal{U}$ **.**
qed *auto*
next
fix *U*
assume $U \in \mathcal{U}$
from *assms*(*2*)[*simplified subbase-in-def*] *openin-topology-generated-by-iff generate-topology-on.Basis*[*OF this*]
show $U \in$ *sigma-sets* (*topspace S*) {*U*. *openin S U*}
by *auto*
qed
qed

Abstract topology version *borel* $\bigotimes_M$ *borel* = *borel*.

lemma *borel-of-prod*:
assumes *second-countable S* **and** *second-countable S'*
shows *borel-of S* $\bigotimes_M$ *borel-of S'* = *borel-of* (*prod-topology S S'*)
proof −
have *borel-of S* $\bigotimes_M$ *borel-of S'* = *sigma* (*topspace S* × *topspace S'*) {*a* × *b* |*a b*. *a* ∈ {*a*. *openin S a*} ∧ *b* ∈ {*b*. *openin S' b*}}
proof −
obtain $\mathcal{O}$ $\mathcal{O}'$ **where** *ho*:
countable $\mathcal{O}$ *base-in S* $\mathcal{O}$ *countable* $\mathcal{O}'$ *base-in S'* $\mathcal{O}'$
using *assms* **by**(*auto simp*: *second-countable-base-in*)
show *?thesis*
unfolding *borel-of-def*
apply(*rule sigma-prod*)
using *topology-generated-by-topspace*[*of* $\mathcal{O}$,*simplified base-is-subbase*[*OF ho*(*2*),*simplified subbase-in-def*,*symmetric*]] *topology-generated-by-topspace*[*of* $\mathcal{O}'$,*simplified base-is-subbase*[*OF ho*(*4*),*simplified subbase-in-def*,*symmetric*]]
base-in-openin[*OF ho*(*2*)] *base-in-openin*[*OF ho*(*4*)]
by(*auto intro*!: *exI*[**where** *x*=$\mathcal{O}$] *exI*[**where** *x*=$\mathcal{O}'$] *simp*: *ho subset-Pow-Union topspace-def*)
qed
also have ... = *borel-of* (*prod-topology S S'*)
using *borel-of-second-countable'*[*OF prod-topology-second-countable*[*OF assms*],*simplified subbase-in-def*,*OF prod-topology-generated-by-open*]

by *simp*
finally show *?thesis* **.**
qed

lemma *product-borel-of-measurable*:
assumes $i \in I$
shows $(\lambda x.\ x\ i) \in (\textit{borel-of}\ (\textit{product-topology}\ S\ I)) \rightarrow_M \textit{borel-of}\ (S\ i)$
by(*auto intro*!: *continuous-map-measurable simp*: *assms*)

Abstract topology version of *sets* (Pi_M *UNIV* (λ-. *borel*)) ⊆ *sets borel*

lemma *sets-PiM-subset-borel-of*:
sets (Π_M *i*∈*I*. *borel-of* (*S i*)) ⊆ *sets* (*borel-of* (*product-topology S I*))
proof −
have ∗: (Π_E *i*∈*I*. *X i*) ∈ *sets* (*borel-of* (*product-topology S I*)) **if** [*measurable*]:⋀*i*. *X i* ∈ *sets* (*borel-of* (*S i*)) *finite* {*i*. *X i* ≠ *topspace* (*S i*)} **for** *X*
proof −
note [*measurable*] = *product-borel-of-measurable*
define *I′* **where** *I′* = {*i*. *X i* ≠ *topspace* (*S i*)} ∩ *I*
have *finite I′* **unfolding** *I′-def* **using** *that* **by** *simp*
have (Π_E *i*∈*I*. *X i*) = (⋂*i*∈*I′*. (λ*x*. *x i*)−‘(*X i*) ∩ *space* (*borel-of* (*product-topology S I*))) ∩ *space* (*borel-of* (*product-topology S I*))
proof(*standard*;*standard*)
fix *x*
assume *x* ∈ Pi_E *I X*
then show *x* ∈ (⋂*i*∈*I′*. (λ*x*. *x i*) −‘ *X i* ∩ *space* (*borel-of* (*product-topology S I*))) ∩ *space* (*borel-of* (*product-topology S I*))
using *sets.sets-into-space*[*OF that*(*1*)] **by**(*auto simp*: *PiE-def I′-def Pi-def space-borel-of*)
next
fix *x*
assume *1*:*x* ∈ (⋂*i*∈*I′*. (λ*x*. *x i*) −‘ *X i* ∩ *space* (*borel-of* (*product-topology S I*))) ∩ *space* (*borel-of* (*product-topology S I*))
have *x i* ∈ *X i* **if** *hi*:*i* ∈ *I* **for** *i*
proof −
consider *i* ∈ *I′* ∧ *I′* ≠ {} | *i* ∉ *I′* ∧ *I′* = {} | *i* ∉ *I′* ∧ *I′* ≠ {} **by** *auto*
then show *?thesis*
apply *cases*
using *sets.sets-into-space*[*OF* ‹⋀*i*. *X i* ∈ *sets* (*borel-of* (*S i*))›] *1 that*
by(*auto simp*: *space-borel-of I′-def*)
qed
then show *x* ∈ Pi_E *I X*
using *1* **by**(*auto simp*: *space-borel-of*)
qed
also have ... ∈ *sets* (*borel-of* (*product-topology S I*))
using *that* ‹*finite I′*› **by**(*auto simp*: *I′-def*)
finally show *?thesis* **.**
qed
then have {Pi_E *I X* |*X*. (∀ *i*. *X i* ∈ *sets* (*borel-of* (*S i*))) ∧ *finite* {*i*. *X i* ≠ *space* (*borel-of* (*S i*))}} ⊆ *sets* (*borel-of* (*product-topology S I*))

by(*auto simp*: *space-borel-of*)
show *?thesis* **unfolding** *sets-PiM-finite*
by(*rule sets.sigma-sets-subset′*,*fact*) (*simp add*: *borel-of-open*[*OF openin-topspace*, *of product-topology S I*,*simplified*] *space-borel-of*)
qed

Abstract topology version of *sets* (*Pi*$_M$ *UNIV* (λ*i*. *borel*)) = *sets borel*.

lemma *sets-PiM-equal-borel-of*:
assumes *countable I* **and** ⋀*i*. *i* ∈ *I* ⟹ *second-countable* (*S i*)
shows *sets* (Π$_M$ *i*∈*I*. *borel-of* (*S i*)) = *sets* (*borel-of* (*product-topology S I*))
proof
obtain *K* **where** *hk*:
countable K base-in (*product-topology S I*) *K*
⋀*k*. *k* ∈ *K* ⟹ ∃ *X*. (*k* = (Π$_E$ *i*∈*I*. *X i*)) ∧ (∀ *i*. *openin* (*S i*) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*S i*)} ∧ {*i*. *X i* ≠ *topspace* (*S i*)} ⊆ *I*
using *product-topology-countable-base-in*[*OF assms*(*1*)] *assms*(*2*)
by *force*
have ∗:*k* ∈ *sets* (Π$_M$ *i*∈*I*. *borel-of* (*S i*)) **if** *k* ∈ *K* **for** *k*
proof −
obtain *X* **where** *H*: *k* = (Π$_E$ *i*∈*I*. *X i*) ⋀*i*. *openin* (*S i*) (*X i*) *finite* {*i*. *X i* ≠ *topspace* (*S i*)} {*i*. *X i* ≠ *topspace* (*S i*)} ⊆ *I*
using *hk*(*3*)[*OF* ‹*k* ∈ *K*›] **by** *blast*
show *?thesis* **unfolding** *H*(*1*) *sets-PiM-finite*
using *borel-of-open*[*OF H*(*2*)] *H*(*3*) **by**(*auto simp*: *space-borel-of*)
qed
have ∗∗: *U* ∈ *sets* (Π$_M$ *i*∈*I*. *borel-of* (*S i*)) **if** *openin* (*product-topology S I*) *U* **for** *U*
proof −
obtain *B* **where** *B* ⊆ *K U* = (⋃*B*)
using ‹*openin* (*product-topology S I*) *U*› ‹*base-in* (*product-topology S I*) *K*› **by** (*metis base-in-def*)
have *countable B* **using** ‹*B* ⊆ *K*› ‹*countable K*› *countable-subset* **by** *blast*
moreover have *k* ∈ *sets* (Π$_M$ *i*∈*I*. *borel-of* (*S i*)) **if** *k* ∈ *B* **for** *k*
using ‹*B* ⊆ *K*› ∗ *that* **by** *auto*
ultimately show *?thesis* **unfolding** ‹*U* = (⋃*B*)› **by** *auto*
qed
have *sigma-sets* (*topspace* (*product-topology S I*)) {*U*. *openin* (*product-topology S I*) *U*} ⊆ *sets* (Π$_M$ *i*∈*I*. *borel-of* (*S i*))
apply (*rule sets.sigma-sets-subset′*) **using** ∗∗ **by**(*auto intro*!: *sets-PiM-I-countable*[*OF assms*(*1*)] *simp*: *borel-of-open*[*OF openin-topspace*])
thus *sets* (*borel-of* (*product-topology S I*)) ⊆ *sets* (Π$_M$ *i*∈*I*. *borel-of* (*S i*))
by (*simp add*: *subset-Pow-Union topspace-def borel-of-def*)
qed(*rule sets-PiM-subset-borel-of*)

lemma *homeomorphic-map-borel-isomorphic*:
assumes *homeomorphic-map X Y f*
shows *measurable-isomorphic-map* (*borel-of X*) (*borel-of Y*) *f*
proof −
obtain *g* **where** *homeomorphic-maps X Y f g*

using *assms* **by**(*auto simp*: *homeomorphic-map-maps*)
hence *continuous-map X Y f continuous-map Y X g*
$\bigwedge x.\ x \in topspace\ X \Longrightarrow g\ (f\ x) = x$
$\bigwedge y.\ y \in topspace\ Y \Longrightarrow f\ (g\ y) = y$
by(*auto simp*: *homeomorphic-maps-def*)
thus *?thesis*
by(*auto intro*!: *measurable-isomorphic-map-byWitness dest*: *continuous-map-measurable simp*: *space-borel-of*)
qed

lemma *homeomorphic-space-measurable-isomorphic*:
assumes *S homeomorphic-space T*
shows *borel-of S measurable-isomorphic borel-of T*
using *homeomorphic-map-borel-isomorphic*[*of S T*] *assms* **by**(*auto simp*: *measurable-isomorphic-def homeomorphic-space*)

lemma *measurable-isomorphic-borel-map*:
assumes *sets M* = *sets* (*borel-of S*) **and** *f*: *measurable-isomorphic-map M N f*
shows $\exists S'.$ *homeomorphic-map S S′ f* $\wedge$ *sets N* = *sets* (*borel-of S′*)
proof −
obtain *g* **where** *fg*:$f \in M \rightarrow_M N$ $g \in N \rightarrow_M M$ $\bigwedge x.\ x{\in}space\ M \Longrightarrow g\ (f\ x) = x$ $\bigwedge y.\ y{\in}space\ N \Longrightarrow f\ (g\ y) = y$ $\bigwedge A.\ A{\in}sets\ M \Longrightarrow f$ ‘ $A \in sets\ N$ $\bigwedge A.\ A{\in}sets\ N \Longrightarrow g$ ‘ $A \in sets\ M$ *bij-betw g* (*space N*) (*space M*)
using *measurable-isomorphic-mapD′*[*OF f*] **by** *metis*
have *g*:*measurable-isomorphic-map N M g*
by(*auto intro*!: *measurable-isomorphic-map-byWitness fg*)
have *g′*:*bij-betw g* (*space N*) (*topspace S*)
using *fg*(*7*) *sets-eq-imp-space-eq*[*OF assms*(*1*)] **by**(*auto simp*: *space-borel-of*)
show *?thesis*
proof(*intro exI*[**where** *x*=*pullback-topology* (*space N*) *g S*] *conjI*)
have [*simp*]: {*U*. *openin* (*pullback-topology* (*space N*) *g S*) *U*} = (‘) *f* ‘ {*U*. *openin S U*}
unfolding *openin-pullback-topology′*[*OF g′*]
proof *safe*
fix *u*
assume *u*:*openin S u*
then have *1*:$u \subseteq space\ M$
by(*simp add*: *sets-eq-imp-space-eq*[*OF assms*(*1*)] *space-borel-of openin-subset*)
with *fg*(*3*) **have** *g* ‘ *f* ‘ *u* = *u*
by(*fastforce simp*: *image-def*)
with *u* **show** *openin S* (*g* ‘ *f* ‘ *u*) **by** *simp*
fix *x*
assume $x \in u$
with *1 fg*(*1*) **show** $f\ x \in space\ N$ **by**(*auto simp*: *measurable-space*)
next
fix *u*
assume *openin S* (*g* ‘ *u*) $u \subseteq space\ N$
with *fg*(*4*) **show** $u \in$ (‘) *f* ‘ {*U*. *openin S U*}
by(*auto simp*: *image-def intro*!: *exI*[**where** *x*=*g* ‘ *u*]) (*metis in-mono*)

```
    qed
    have [simp]:g −‘ topspace S ∩ space N = space N
      using bij-betw-imp-surj-on g′ by blast
    show sets N = sets (borel-of (pullback-topology (space N) g S))
     by(auto simp: sets-borel-of topspace-pullback-topology intro!: measurable-isomorphic-map-sigma-sets[OF
assms(1)[simplified sets-borel-of space-borel-of[symmetric] sets-eq-imp-space-eq[OF
assms(1),symmetric]] f])
  next
    show homeomorphic-map S (pullback-topology (space N) g S) f
    proof(rule homeomorphic-maps-imp-map[where g=g])
      obtain f′ where f′:homeomorphic-maps (pullback-topology (space N) g S) S
g f′
        using topology-from-bij(1)[OF g′] homeomorphic-map-maps by blast
      have f′2:f′ y = f y if y:y ∈ topspace S for y
      proof −
        have [simp]:g −‘ topspace S ∩ space N = space N
          using bij-betw-imp-surj-on g′ by blast
        obtain x where x ∈ space N y = g x
          using g′ y by(auto simp: bij-betw-def image-def)
        thus ?thesis
         using fg(4) f′ by(auto simp: homeomorphic-maps-def topspace-pullback-topology)
      qed
      thus homeomorphic-maps S (pullback-topology (space N) g S) f g
      by(auto intro!: homeomorphic-maps-eq[OF f′] simp: homeomorphic-maps-sym[of
S])
    qed
  qed
qed

lemma measurable-isomorphic-borels:
  assumes sets M = sets (borel-of S) M measurable-isomorphic N
  shows ∃ S′. S homeomorphic-space S′ ∧ sets N = sets (borel-of S′)
 using measurable-isomorphic-borel-map[OF assms(1)] assms(2) homeomorphic-map-maps
  by(fastforce simp: measurable-isomorphic-def homeomorphic-space-def )

end
```

1.3 Lemmas for Abstract Metric Spaces

```
theory Set-Based-Metric-Space
  imports Lemmas-StandardBorel
begin
```

We prove additional lemmas related to set-based metric spaces.

1.3.1 Basic Lemmas

```
lemma
  assumes Metric-space M d ⋀x y. x ∈ M ⟹ y ∈ M ⟹ d x y = d′ x y
```

```
    and ⋀x y. d′ x y = d′ y x ⋀x y. d′ x y ≥ 0
  shows Metric-space-eq: Metric-space M d′
   and Metric-space-eq-mtopology: Metric-space.mtopology M d = Metric-space.mtopology
M d′
      and Metric-space-eq-mcomplete: Metric-space.mcomplete M d ⟷ Met-
ric-space.mcomplete M d′
proof −
 interpret m1: Metric-space M d by fact
 show Metric-space M d′
   using assms by(auto simp: Metric-space-def)
 then interpret m2:Metric-space M d′ .
 have [simp]: m1.mball x e = m2.mball x e for x e
   using assms by(auto simp: m1.mball-def m2.mball-def)
 show 1:m1.mtopology = m2.mtopology
   by(auto simp: topology-eq m1.openin-mtopology m2.openin-mtopology)
 show m1.mcomplete = m2.mcomplete
  by(auto simp: 1 m1.mcomplete-def m2.mcomplete-def m1.MCauchy-def m2.MCauchy-def
assms(2) in-mono)
qed

context Metric-space
begin

lemma mtopology-base-in-balls: base-in mtopology {mball a ε | a ε. a ∈ M ∧ ε >
0}
proof −
 have 1:⋀x. x ∈ {mball a ε | a ε. a ∈ M ∧ ε > 0} ⟹ openin mtopology x
   by auto
 show ?thesis
   unfolding base-in-def2[of {mball a ε | a ε. a ∈ M ∧ ε > 0},OF 1,simplified]
   by (metis centre-in-mball-iff in-mono openin-mtopology)
qed

lemma closedin-metric2: closedin mtopology C ⟷ C ⊆ M ∧ (∀x. x ∈ C ⟷
(∀ε>0. mball x ε ∩ C ≠ {}))
proof
 assume h:closedin mtopology C
 have 1: C ⊆ M
   using Metric-space.closedin-metric Metric-space-axioms h by blast
 show C ⊆ M ∧ (∀x. x ∈ C ⟷ (∀ε>0. mball x ε ∩ C ≠ {}))
 proof safe
   fix ε x
   assume x ∈ C (0 :: real) < ε mball x ε ∩ C = {}
   with 1 show False
     by blast
 next
   fix x
   assume ∀ε>0. mball x ε ∩ C ≠ {}
   hence ∃xn. xn ∈ mball x (1 / real (Suc n)) ∩ C for n
```

```
    by (meson all-not-in-conv divide-pos-pos of-nat-0-less-iff zero-less-Suc zero-less-one)
    then obtain xn where xn:⋀n. xn n ∈ mball x (1 / real (Suc n)) ∩ C
      by metis
    hence xxn:x ∈ M range xn ⊆ C
      using xn by auto
    have limitin mtopology xn x sequentially
      unfolding limitin-metric eventually-sequentially
    proof safe
      fix ε
      assume (0 :: real) < ε
      then obtain N where hN: 1 / real (Suc N) < ε
        using nat-approx-posE by blast
      show ∃N. ∀n≥N. xn n ∈ M ∧ d (xn n) x < ε
      proof(safe intro!: exI[where x=N])
        fix n
        assume n[arith]: N ≤ n
        then have 1 / real (Suc n) < ε
       by (metis Suc-le-mono hN inverse-of-nat-le nat.distinct(1) order-le-less-trans)
        with xn[of n] show d (xn n) x < ε
          by (simp add: commute)
      qed(use xxn 1 in auto)
    qed fact
    with h 1 xxn show x ∈ C
      by(auto simp: metric-closedin-iff-sequentially-closed)
  qed(use 1 in auto)
next
  assumeC ⊆ M ∧ (∀x. (x ∈ C) ⟷ (∀ε>0. mball x ε ∩ C ≠ {}))
  hence h:C ⊆ M ⋀x. (x ∈ C) ⟷ (∀ε>0. mball x ε ∩ C ≠ {})
    by simp-all
  show closedin mtopology C
    unfolding metric-closedin-iff-sequentially-closed
  proof safe
    fix xn x
    assume h':range xn ⊆ C limitin mtopology xn x sequentially
    hence x ∈ M by (simp add: limitin-mspace)
    have mball x ε ∩ C ≠ {} if ε > 0 for ε
    proof −
      obtain N where hN:⋀n. n ≥ N ⟹ d (xn n) x < ε
        using h'(2) ‹ε > 0› limit-metric-sequentially by blast
      have xn N ∈ mball x ε ∩ C
        using h'(1) hN[of N] ‹x ∈ M› commute h(1) by fastforce
      thus mball x ε ∩ C ≠ {} by auto
    qed
    with h(2)[of x] show x ∈ C by simp
  qed(use h(1) in auto)
qed

lemma openin-mtopology2:
 openin mtopology U ⟷ U ⊆ M ∧ (∀xn x. limitin mtopology xn x sequentially ∧
```

x ∈ *U* ⟶ (∃ *N*. ∀ *n*≥*N*. *xn n* ∈ *U*))
unfolding *openin-mtopology*
proof *safe*
fix *xn x*
assume *h*: ∀ *x*. *x* ∈ *U* ⟶ (∃ *r*>*0*. *mball x r* ⊆ *U*) *limitin mtopology xn x sequentially x* ∈ *U U* ⊆ *M*
then obtain *r* **where** *r*: *r* > *0 mball x r* ⊆ *U*
by *auto*
with *h*(*2*) **obtain** *N* **where** *N*: ⋀*n*. *n* ≥ *N* ⟹ *xn n* ∈ *M* ⋀*n*. *n* ≥ *N* ⟹ *d* (*xn n*) *x* < *r*
by (*metis limit-metric-sequentially*)
with *h* **have** ∃ *N*. ∀ *n*≥*N*. *xn n* ∈ *mball x r*
by(*auto intro*!: *exI*[**where** *x*=*N*] *simp*:*commute*)
with *r* **show** ∃ *N*. ∀ *n*≥*N*. *xn n* ∈ *U*
by *blast*
next
fix *x*
assume *h*:*U* ⊆ *M* ∀ *xn x*. *limitin mtopology xn x sequentially* ∧ *x* ∈ *U* ⟶ (∃ *N*. ∀ *n*≥*N*. *xn n* ∈ *U*) *x* ∈ *U*
show ∃ *r*>*0*. *mball x r* ⊆ *U*
proof(*rule ccontr*)
assume ¬ (∃ *r*>*0*. *mball x r* ⊆ *U*)
then have ∀ *n*. ∃ *xn*∈*mball x* (*1* / *Suc n*). *xn* ∉ *U*
by (*meson of-nat-0-less-iff subsetI zero-less-Suc zero-less-divide-1-iff*)
then obtain *xn* **where** *xn*: ⋀*n*. *xn n* ∈ *mball x* (*1* / *Suc n*) ⋀*n*. *xn n* ∉ *U*
by *metis*
have *limitin mtopology xn x sequentially*
unfolding *limit-metric-sequentially*
proof *safe*
fix *e* :: *real*
assume *e*:*0* < *e*
then obtain *N* **where** *N*: *1* / *real* (*Suc N*) < *e*
using *nat-approx-posE* **by** *blast*
show ∃ *N*. ∀ *n*≥*N*. *xn n* ∈ *M* ∧ *d* (*xn n*) *x* < *e*
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *n*: *n* ≥ *N*
then have *1* / *Suc n* < *e*
by (*metis N Suc-le-mono inverse-of-nat-le nat.distinct*(*1*) *order-le-less-trans*)
thus *d* (*xn n*) *x* < *e*
using *xn*(*1*)[*of n*] **by**(*auto simp*: *commute*)
qed(*use xn* **in** *auto*)
qed(*use h* **in** *auto*)
with *h*(*2*,*3*) *xn*(*2*) **show** *False*
by *auto*
qed
qed

lemma *closure-of-mball*: *mtopology closure-of mball a e* ⊆ *mcball a e*

by (*simp add*: *closure-of-minimal mball-subset-mcball*)

lemma *interior-of-mcball*: *mball a e* ⊆ *mtopology interior-of mcball a e*
by (*simp add*: *interior-of-maximal-eq mball-subset-mcball*)

lemma *isolated-points-of-mtopology*:
mtopology isolated-points-of A = {*x*∈*M*∩*A*. ∀ *xn*. *range xn* ⊆ *A* ∧ *limitin mtopology xn x sequentially* ⟶ (∃ *no*. ∀ *n*≥*no*. *xn n* = *x*)}
proof *safe*
fix *x xn*
assume *h*:*x* ∈ *mtopology isolated-points-of A limitin mtopology xn x sequentially range xn* ⊆ *A*
then have *ha*:*x* ∈ *topspace mtopology x* ∈ *A* ∃ *U*. *x* ∈ *U* ∧ *openin mtopology U* ∧ *U* ∩ (*A* − {*x*}) = {}
by(*simp-all add*: *in-isolated-points-of*)
then obtain *U* **where** *u*:*x* ∈ *U openin mtopology U U* ∩ (*A* − {*x*}) = {}
by *auto*
then obtain *e* **where** *e*: *e* > *0 mball x e* ⊆ *U*
by (*meson openin-mtopology*)
then obtain *N* **where** ⋀*n*. *n* ≥ *N* ⟹ *xn n* ∈ *mball x e*
using *h*(*2*) *commute limit-metric-sequentially* **by** *fastforce*
thus ∃ *no*. ∀ *n*≥*no*. *xn n* = *x*
using *h*(*3*) *e*(*2*) *u*(*3*) **by**(*fastforce intro*!: *exI*[**where** *x*=*N*])
qed (*auto simp*: *derived-set-of-sequentially isolated-points-of-def*, *blast*)

lemma *perfect-set-mball-infinite*:
assumes *perfect-set mtopology A a* ∈ *A e* > *0*
shows *infinite* (*mball a e*)
proof *safe*
assume *h*: *finite* (*mball a e*)
have *a* ∈ *M*
using *assms perfect-setD*(*2*)[*OF assms*(*1*)] **by** *auto*
have ∃ *e* > *0*. *mball a e* = {*a*}
proof −
consider *mball a e* = {*a*} | {*a*} ⊂ *mball a e*
using ‹*a* ∈ *M*› *assms*(*3*) **by** *blast*
thus *?thesis*
proof *cases*
case *1*
with *assms* **show** *?thesis* **by** *auto*
next
case *2*
then have *nen*:{*d a b* |*b*. *b* ∈ *mball a e* ∧ *a* ≠ *b*} ≠ {}
by *auto*
have *fin*: *finite* {*d a b* |*b*. *b* ∈ *mball a e* ∧ *a* ≠ *b*}
using *h* **by** (*auto simp del*: *in-mball*)
define *e*′ **where** *e*′ ≡ *Min* {*d a b* |*b*. *b* ∈ *mball a e* ∧ *a* ≠ *b*}
have *e*′ > *0*
using *mdist-pos-eq*[*OF* ‹*a* ∈ *M*›] **by**(*simp add*: *e*′-*def Min-gr-iff*[*OF fin nen*]

del: *in-mball*) *auto*
have *bd*:$\bigwedge b.\ b \in mball\ a\ e \Longrightarrow a \neq b \Longrightarrow e' \leq d\ a\ b$
by(*auto simp*: *e′-def Min-le-iff*[*OF fin nen*] *simp del*: *in-mball*)
have $e' \leq e$
unfolding *e′-def Min-le-iff*[*OF fin nen*]
using *nen* **by** *auto*
show *?thesis*
proof(*safe intro*!: *exI*[**where** *x*=*e′*])
fix *x*
assume *x*:$x \in mball\ a\ e'$
then show $x = a$
using ‹$e' \leq e$› *bd* **by** *fastforce*
qed (*use* ‹$a \in M$› ‹$e' > 0$› **in** *auto*)
qed
qed
then obtain *e′* **where** *e′*:$e' > 0$ $mball\ a\ e' = \{a\}$ **by** *auto*
show *False*
using *perfect-setD*(*3*)[*OF assms*(*1*,*2*) *centre-in-mball-iff*[*of a e′*,*THEN iffD2*]] ‹$a \in M$› ‹$e' > 0$› *e′*(*2*) **by** *blast*
qed

lemma *MCauchy-dist-Cauchy*:
assumes *MCauchy xn MCauchy yn*
shows *Cauchy* ($\lambda n.\ d\ (xn\ n)\ (yn\ n)$)
unfolding *metric-space-class.Cauchy-altdef2 dist-real-def*
proof *safe*
have *h*:$\bigwedge n.\ xn\ n \in M$ $\bigwedge n.\ yn\ n \in M$
using *assms* **by**(*auto simp*: *MCauchy-def*)
fix *e* :: *real*
assume *e*:$0 < e$
with *assms* **obtain** *N1 N2* **where** *N*: $\bigwedge n\ m.\ n \geq N1 \Longrightarrow m \geq N1 \Longrightarrow d\ (xn\ n)\ (xn\ m) < e\ /\ 2$ $\bigwedge n\ m.\ n \geq N2 \Longrightarrow m \geq N2 \Longrightarrow d\ (yn\ n)\ (yn\ m) < e\ /\ 2$
by (*metis MCauchy-def zero-less-divide-iff zero-less-numeral*)
define *N* **where** $N \equiv max\ N1\ N2$
then have *N′*: $N \geq N1$ $N \geq N2$ **by** *auto*
show $\exists N.\ \forall n \geq N.\ |d\ (xn\ n)\ (yn\ n) - d\ (xn\ N)\ (yn\ N)| < e$
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *n*:$N \leq n$
have $d\ (xn\ n)\ (yn\ n) \leq d\ (xn\ n)\ (xn\ N) + d\ (xn\ N)\ (yn\ N) + d\ (yn\ N)\ (yn\ n)$
$d\ (xn\ N)\ (yn\ N) \leq d\ (xn\ N)\ (xn\ n) + d\ (xn\ n)\ (yn\ n) + d\ (yn\ n)\ (yn\ N)$
using *triangle*[*OF h*(*1*)[*of n*] *h*(*1*)[*of N*] *h*(*2*)[*of n*]] *triangle*[*OF h*(*1*)[*of N*] *h*(*2*)[*of N*] *h*(*2*)[*of n*]]
triangle[*OF h*(*1*)[*of N*] *h*(*2*)[*of n*] *h*(*2*)[*of N*]] *triangle*[*OF h*(*1*)[*of N*] *h*(*1*)[*of n*] *h*(*2*)[*of n*]] **by** *auto*
thus $|d\ (xn\ n)\ (yn\ n) - d\ (xn\ N)\ (yn\ N)| < e$
using *N*(*1*)[*OF N′*(*1*) *order.trans*[*OF N′*(*1*) *n*]] *N*(*2*)[*OF N′*(*2*) *order.trans*[*OF N′*(*2*) *n*]] *N*(*1*)[*OF order.trans*[*OF N′*(*1*) *n*] *N′*(*1*)] *N*(*2*)[*OF order.trans*[*OF N′*(*2*)

n] *N*′(*2*)]
    by *auto*
  qed
qed

1.3.2 Dense in Metric Spaces

abbreviation *mdense* ≡ *dense-in mtopology*

https://people.bath.ac.uk/mw2319/ma30252/sec-dense.html

lemma *mdense-def*:
mdense U ⟷ *U* ⊆ *M* ∧ (∀ *x*∈*M*. ∀ *ε*>*0*. *mball x ε* ∩ *U* ≠ {})
proof *safe*
  assume *h*: *U* ⊆ *M* (∀ *x*∈*M*. ∀ *ε*>*0*. *mball x ε* ∩ *U* ≠ {})
  show *dense-in mtopology U*
  proof(*rule dense-inI*)
    fix *V*
    assume *h*′:*openin mtopology V V* ≠ {}
    then obtain *x* **where** *1*:*x* ∈ *V* **by** *auto*
    then obtain *ε* **where** *2*:*ε*>*0 mball x ε* ⊆ *V*
      by (*meson h*′(*1*) *openin-mtopology*)
    have *mball x ε* ∩ *U* ≠ {}
      using *h 1 2 openin-subset*[*OF h*′(*1*)]
      by (*auto simp del*: *in-mball*)
    thus *U* ∩ *V* ≠ {} **using** *2* **by** *auto*
  qed(*use h* **in** *auto*)
next
  fix *x ε*
  assume *h*:*x* ∈ *M* (*0* :: *real*) < *ε mball x ε* ∩ *U* = {} *mdense U*
  then have *mball x ε* ≠ {} *openin mtopology* (*mball x ε*)
    by *auto*
  with *h*(*4*) **have** *mball x ε* ∩ *U* ≠ {}
    by(*auto simp*: *dense-in-def*)
  with *h*(*3*) **show** *False*
    by *simp*
qed(*auto simp*: *dense-in-def*)

corollary *mdense-balls-cover*:
  assumes *mdense U* **and** *e* > *0*
  shows (⋃ *u*∈*U*. *mball u e*) = *M*
  using *assms*[*simplified mdense-def*] *commute* **by** *fastforce*

lemma *mdense-empty-iff*: *mdense* {} ⟷ *M* = {}
  by(*auto simp*: *mdense-def*) (*use zero-less-one* **in** *blast*)

lemma *mdense-M*: *mdense M*
  by(*auto simp*: *mdense-def*)

lemma *mdense-def2*:

```
 mdense U ⟷ U ⊆ M ∧ (∀x∈M. ∀ε>0.∃y∈U. d x y < ε)
proof safe
  fix x e
  assume h: mdense U and hxe: x ∈ M (0 :: real) < e
  then have x ∈ (⋃u∈U. mball u e)
    by(simp add: mdense-balls-cover)
  thus ∃y∈U. d x y < e
    by (fastforce simp: commute)
qed(fastforce simp: mdense-def)+

lemma mdense-def3:
 mdense U ⟷ U ⊆ M ∧ (∀x∈M. ∃xn. range xn ⊆ U ∧ limitin mtopology xn x
sequentially)
  unfolding mdense-def
proof safe
  fix x
  assume h: U ⊆ M ∀x∈M. ∀ε>0. mball x ε ∩ U ≠ {} x ∈ M
  then have ⋀n. mball x (1 / (real n + 1)) ∩ U ≠ {}
    by auto
  hence ∀n. ∃k. k ∈ mball x (1 / (real n + 1)) ∩ U by (auto simp del: in-mball)
  hence ∃a. ∀n. a n ∈ mball x (1 / (real n + 1)) ∩ U by(rule choice)
  then obtain xn where xn: ⋀n. xn n ∈ mball x (1 / (real n + 1)) ∩ U
    by auto
  show ∃xn. range xn ⊆ U ∧ limitin mtopology xn x sequentially
    unfolding limitin-metric eventually-sequentially
  proof(safe intro!: exI[where x=xn])
    fix ε :: real
    assume he:0 < ε
    then obtain N where hn: 1 / ε < real N
      using reals-Archimedean2 by blast
    have hn′: 0 < real N
      by(rule ccontr) (use hn he in fastforce)
    hence 1 / real N < ε
      using he hn by (metis divide-less-eq mult.commute)
    hence hn′′:1 / (real N + 1) < ε
     using hn′ by(auto intro!: order.strict-trans[OF linordered-field-class.divide-strict-left-mono[of
real N real N + 1 1]])
    show ∃N. ∀n≥N. xn n ∈ M ∧ d (xn n) x < ε
    proof(safe intro!: exI[where x=N])
      fix n
      assume N ≤ n
      then have 1:1 / (real n + 1) ≤ 1 / (real N + 1)
        using hn′ by(auto intro!: linordered-field-class.divide-left-mono)
      show d (xn n) x < ε
        using xn[of n] order.strict-trans1[OF 1 hn′′] by (auto simp: commute)
    qed(use xn in auto)
  qed(use xn h in auto)
next
  fix x and e :: real
```

assume *h*: $U \subseteq M$ $\forall x \in M.\ \exists xn.\ range\ xn \subseteq U \wedge limitin\ mtopology\ xn\ x\ sequentially$ $x \in M$ $0 < e$ $mball\ x\ e \cap U = \{\}$
then obtain *xn* **where** *xn*:$range\ xn \subseteq U$ *limitin mtopology xn x sequentially*
by *auto*
with *h*(*4*) **obtain** *N* **where** *N*: $\bigwedge n.\ n \geq N \Longrightarrow d\ (xn\ n)\ x < e$
by (*meson limit-metric-sequentially*)
have $xn\ N \in mball\ x\ e \cap U$
using *N*[*of N*] *xn*(*1*) *h*(*1*,*3*) **by** (*auto simp*: *commute*)
with *h*(*5*) **show** *False* **by** *simp*
qed

Diameter

definition *mdiameter* :: $'a\ set \Rightarrow ennreal$ **where**
$mdiameter\ A \equiv \bigsqcup\ \{ennreal\ (d\ x\ y) \mid x\ y.\ x \in A \cap M \wedge y \in A \cap M\}$

lemma *mdiameter-empty*[*simp*]:
$mdiameter\ \{\} = 0$
by(*simp add*: *mdiameter-def bot-ennreal*)

lemma *mdiameter-def2*:
assumes $A \subseteq M$
shows $mdiameter\ A = \bigsqcup\ \{ennreal\ (d\ x\ y) \mid x\ y.\ x \in A \wedge y \in A\}$
using *assms* **by**(*auto simp*: *mdiameter-def*) (*meson subset-eq*)

lemma *mdiameter-subset*:
assumes $A \subseteq B$
shows $mdiameter\ A \leq mdiameter\ B$
unfolding *mdiameter-def* **using** *assms* **by**(*auto intro*!: *Sup-subset-mono*)

lemma *mdiameter-cball-leq*: $mdiameter\ (mcball\ a\ \varepsilon) \leq ennreal\ (2 * \varepsilon)$
unfolding *Sup-le-iff mdiameter-def*
proof *safe*
fix *x y*
assume *h*:$x \in mcball\ a\ \varepsilon$ $y \in mcball\ a\ \varepsilon$ $x \in M$ $y \in M$
have $d\ x\ y \leq 2 * \varepsilon$
using *h*(*1*) *h*(*2*) *triangle''* **by** *fastforce*
thus $ennreal\ (d\ x\ y) \leq ennreal\ (2 * \varepsilon)$
using *ennreal-leI* **by** *blast*
qed

lemma *mdiameter-ball-leq*:
$mdiameter\ (mball\ a\ \varepsilon) \leq ennreal\ (2 * \varepsilon)$
using *mdiameter-subset*[*OF mball-subset-mcball*[*of a* ε]] *mdiameter-cball-leq*[*of a* ε]
by *auto*

lemma *mdiameter-is-sup*:
assumes $x \in A \cap M$ $y \in A \cap M$
shows $d\ x\ y \leq mdiameter\ A$

using *assms* **by**(*auto simp*: *mdiameter-def intro*!: *Sup-upper*)

lemma *mdiameter-is-sup'*:
assumes $x \in A \cap M$ $y \in A \cap M$ *mdiameter* $A \leq$ *ennreal* r $r \geq 0$
shows $d\ x\ y \leq r$
using *order.trans*[*OF mdiameter-is-sup*[*OF assms*(*1*,*2*)] *assms*(*3*)] *assms*(*4*) **by** *simp*

lemma *mdiameter-le*:
assumes $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow d\ x\ y \leq r$
shows *mdiameter* $A \leq r$
using *assms* **by**(*auto simp*: *mdiameter-def Sup-le-iff ennreal-leI*)

lemma *mdiameter-eq-closure*: *mdiameter* (*mtopology closure-of A*) = *mdiameter* *A*
proof(*rule antisym*)
show *mdiameter* $A \leq$ *mdiameter* (*mtopology closure-of A*)
by(*fastforce intro*!: *Sup-subset-mono simp*: *mdiameter-def metric-closure-of*)
next
have $\{$*ennreal* $(d\ x\ y)\ |x\ y.\ x \in A \cap M \wedge y \in A \cap M\}$ = *ennreal* ` $\{d\ x\ y\ |x\ y.\ x \in A \cap M \wedge y \in A \cap M\}$
by *auto*
also have *mdiameter* (*mtopology closure-of A*) $\leq \bigsqcup$ ...
unfolding *le-Sup-iff-less*
proof *safe*
fix *r*
assume $r <$ *mdiameter* (*mtopology closure-of A*)
then obtain $x\ y$ **where** *xy*:$x \in$ *mtopology closure-of A* $x \in M$ $y \in$ *mtopology closure-of A* $y \in M$ $r <$ *ennreal* $(d\ x\ y)$
by(*auto simp*: *mdiameter-def less-Sup-iff*)
hence $r < \top$
using *dual-order.strict-trans ennreal-less-top* **by** *blast*
define *e* **where** $e \equiv (d\ x\ y -$ *enn2real* $r)/2$
have $e > 0$
using *xy*(*5*) ‹$r < \top$› **by**(*simp add*: *e-def*)
then obtain $x'\ y'$ **where** *xy'*:$x' \in$ *mball* $x\ e$ $x' \in A$ $y' \in$ *mball* $y\ e$ $y' \in A$
using *xy* **by**(*fastforce simp*: *metric-closure-of*)
show $\exists\ i \in \{d\ x\ y\ |x\ y.\ x \in A \cap M \wedge y \in A \cap M\}.\ r \leq$ *ennreal* i
proof(*safe intro*!: *bexI*[**where** $x = d\ x'\ y'$])
have $d\ x\ y \leq d\ x\ x' + d\ x'\ y' + d\ y\ y'$
using *triangle*[*OF xy*(*2*) - *xy*(*4*),*of x'*] *xy' triangle*[*of x' y' y*]
by(*fastforce simp add*: *commute*)
also have ... $< d\ x\ y -$ *enn2real* $r + d\ x'\ y'$
using *xy'*(*1*) *xy'*(*3*) **by**(*simp add*: *e-def*)
finally have *enn2real* $r < d\ x'\ y'$ **by** *simp*
thus $r \leq$ *ennreal* $(d\ x'\ y')$
by (*simp add*: ‹$r < \top$›)
qed(*use xy'*(*1*) *xy'*(*3*) *xy'*(*2*,*4*) **in** *auto*)
qed

finally show *mdiameter* (*mtopology closure-of A*) ≤ *mdiameter A*
by(*simp add*: *mdiameter-def*)
qed

lemma *mbounded-finite-mdiameter*: *mbounded A* ⟷ *A* ⊆ *M* ∧ *mdiameter A* < ∞
proof *safe*
assume *mbounded A*
then obtain *x B* **where** *A* ⊆ *mcball x B*
by(*auto simp*: *mbounded-def*)
then have *mdiameter A* ≤ *mdiameter* (*mcball x B*)
by(*rule mdiameter-subset*)
also have ... ≤ *ennreal* (*2* ∗ *B*)
by(*rule mdiameter-cball-leq*)
also have ... < ∞
by *auto*
finally show *mdiameter A* < ∞ **.**
next
assume *h*:*mdiameter A* < ∞ *A* ⊆ *M*
consider *A* = {} | *A* ≠ {} **by** *auto*
then show *mbounded A*
proof *cases*
case *h2*:*2*
then have *1*:{*d x y* |*x y*. *x* ∈ *A* ∧ *y* ∈ *A*} ≠ {} **by** *auto*
have *eq*:{*ennreal* (*d x y*) | *x y*. *x* ∈ *A* ∧ *y* ∈ *A*} = *ennreal* ‘ {*d x y* | *x y*. *x* ∈ *A* ∧ *y* ∈ *A*}
by *auto*
hence *2*:*mdiameter A* = ⨆ (*ennreal* ‘ {*d x y* | *x y*. *x* ∈ *A* ∧ *y* ∈ *A*})
using *h* **by**(*auto simp add*: *mdiameter-def2*)
obtain *x y* **where** *hxy*:
x ∈ *A* *y* ∈ *A* *mdiameter A* < *ennreal* (*d x y* + *1*)
using *SUP-approx-ennreal*[*OF - 1 2*,*of 1*] *h* **by**(*fastforce simp*: *diameter-def*)
show *?thesis*
unfolding *mbounded-alt*
proof(*safe intro*!: *exI*[**where** *x*=*d x y* + *1*])
fix *w z*
assume *w* ∈ *A* *z* ∈ *A*
with *SUP-lessD*[*OF hxy*(*3*)[*simplified 2*]]
have *ennreal* (*d w z*) < *ennreal* (*d x y* + *1*)
by *blast*
thus *d w z* ≤ *d x y* + *1*
by (*metis canonically-ordered-monoid-add-class.lessE ennreal-le-iff2 ennreal-neg le-iff-add not-less-zero*)
qed (*use h* **in** *auto*)
qed(*auto simp*: *mbounded-def*)
qed(*auto simp*: *mbounded-def*)

Distance between a point and a set.

definition *d-set* :: *'a set* ⇒ *'a* ⇒ *real* **where**

d-set A ≡ (λ*x*. *if A* ≠ {} ∧ *A* ⊆ *M* ∧ *x* ∈ *M then Inf* {*d x y* |*y*. *y* ∈ *A*} *else 0*)

lemma *d-set-nonneg*[*simp*]:
 d-set A x ≥ *0*
proof −
  have {*d x y* |*y*. *y* ∈ *A*} = *d x* ‘ *A* **by** *auto*
  thus *?thesis*
    by(*auto simp*: *d-set-def intro*!: *cINF-greatest*[*of* - - *d x*])
qed

lemma *d-set-bdd-below*[*simp*]:
 bdd-below {*d x y* |*y*. *y* ∈ *A*}
  by(*auto simp*: *bdd-below-def intro*!: *exI*[**where** *x*=*0*])

lemma *d-set-singleton*[*simp*]:
  x ∈ *M* ⟹ *y* ∈ *M* ⟹ *d-set* {*y*} *x* = *d x y*
  by(*auto simp*: *d-set-def*)

lemma *d-set-empty*[*simp*]:
 d-set {} *x* = *0*
  by(*simp add*: *d-set-def*)

lemma *d-set-notin*:
  x ∉ *M* ⟹ *d-set A x* = *0*
  by(*auto simp*: *d-set-def*)

lemma *d-set-inA*:
  assumes *x* ∈ *A*
  shows *d-set A x* = *0*
proof −
  {
    assume *x* ∈ *M A* ⊆ *M*
    then have *0* ∈ {*d x y* |*y*. *y* ∈ *A*}
      using *assms* **by**(*auto intro*: *exI*[**where** *x*=*x*])
    moreover have *A* ≠ {}
      using *assms* **by** *auto*
    ultimately have *Inf* {*d x y* |*y*. *y* ∈ *A*} = *0*
      using *cInf-lower*[*OF* ‹*0* ∈ {*d x y* |*y*. *y* ∈ *A*}›] *d-set-nonneg*[*of A x*] ‹*A* ⊆ *M*›
‹*x* ∈ *M*›
      by(*auto simp*: *d-set-def*)
  }
  thus *?thesis*
    using *assms* **by**(*auto simp*: *d-set-def*)
qed

lemma *d-set-nzeroD*:
  assumes *d-set A x* ≠ *0*
  shows *A* ⊆ *M x* ∉ *A A* ≠ {}
   by(*rule ccontr*, *use assms d-set-inA d-set-def* **in** *auto*)

lemma *d-set-antimono*:
  assumes $A \subseteq B$ $A \neq \{\}$ $B \subseteq M$
  shows *d-set* B $x \leq$ *d-set* A x
proof(*cases* $B = \{\}$)
  case *h*:*False*
  with *assms* **have** $\{d\ x\ y\ |y.\ y \in B\} \neq \{\}$ $\{d\ x\ y\ |y.\ y \in A\} \subseteq \{d\ x\ y\ |y.\ y \in B\}$ $A \subseteq M$
    by *auto*
  with *assms*(*3*) **show** *?thesis*
    by(*simp add*: *d-set-def cInf-superset-mono assms*(*2*))
qed(*use assms* **in** *simp*)

lemma *d-set-bounded*:
  assumes $\bigwedge y.\ y \in A \Longrightarrow d\ x\ y < K$ $K > 0$
  shows *d-set* A $x < K$
proof −
  consider $A = \{\}$ | $\neg A \subseteq M$ | $x \notin M$ | $A \neq \{\}$ $A \subseteq M$ $x \in M$
    by *blast*
  then show *?thesis*
  proof *cases*
    case *4*
    then have *2*:$\{d\ x\ y\ |y.\ y \in A\} \neq \{\}$ **by** *auto*
    show *?thesis*
      using *assms* **by** (*auto simp add*: *d-set-def cInf-lessD*[*OF 2*] *cInf-less-iff*[*OF 2*])
  qed(*auto simp*: *d-set-def assms*)
qed

lemma *d-set-tr*:
  assumes $x \in M$ $y \in M$
  shows *d-set* A $x \leq d\ x\ y$ + *d-set* A y
proof −
  consider $A = \{\}$ | $\neg A \subseteq M$ | $A \neq \{\}$ $A \subseteq M$
    by *blast*
  then show *?thesis*
  proof *cases*
    case *3*
    have *d-set* A $x \leq$ *Inf* $\{d\ x\ y + d\ y\ a\ |a.\ a{\in}A\}$
    proof −
      have $\bigsqcap$ $\{d\ x\ y\ |y.\ y \in A\} \leq \bigsqcap$ $\{d\ x\ y + d\ y\ a\ |a.\ a \in A\}$
        by(*rule cInf-mono*) (*use 3 assms triangle* **in** *fastforce*)+
      thus *?thesis*
        by(*simp add*: *d-set-def assms 3*)
    qed
    also have ... = ($\bigsqcap a{\in}A.\ d\ x\ y + d\ y\ a$)
      by (*simp add*: *Setcompr-eq-image*)
    also have ... = $d\ x\ y + \bigsqcap$ ($d\ y$ ‘ A)
      using *3* **by**(*auto intro*!: *Inf-add-eq bdd-belowI*[**where** *m*=*0*])

also have ... = *d x y* + *d-set A y*
using *assms 3* **by**(*auto simp*: *d-set-def Setcompr-eq-image*)
finally show *?thesis* **.**
qed(*auto simp*: *d-set-def*)
qed

lemma *d-set-abs-le*:
assumes $x \in M$ $y \in M$
shows |*d-set A x* − *d-set A y*| ≤ *d x y*
using *d-set-tr*[*OF assms*,*of A*] *d-set-tr*[*OF assms*(*2*,*1*),*of A*,*simplified commute*[*of y x*]]
by *auto*

lemma *d-set-inA-le*:
assumes $y \in A$
shows *d-set A x* ≤ *d x y*
proof −
consider $A \neq \{\}$ $A \subseteq M$ $x \in M$ | $\neg$ $A \subseteq M$ | $x \notin M$
using *assms* **by** *blast*
then show *?thesis*
proof *cases*
case *1*
with *assms* **have** $y \in M$ **by** *auto*
from *d-set-tr*[*OF 1*(*3*) *this*,*of A*] **show** *?thesis*
by(*simp add*: *d-set-inA*[*OF assms*])
qed(*auto simp*: *d-set-def*)
qed

lemma *d-set-ball-empty*:
assumes $A \neq \{\}$ $A \subseteq M$ $e > 0$ $x \in M$ *mball x e* ∩ *A* = {}
shows *d-set A x* ≥ *e*
using *assms* **by**(*fastforce simp*: *d-set-def assms*(*1*) *le-cInf-iff*)

lemma *d-set-closed-pos*:
assumes *closedin mtopology A* $A \neq \{\}$ $x \in M$ $x \notin A$
shows *d-set A x* > *0*
proof −
have *a*:$A \subseteq M$ *openin mtopology* (*M* − *A*)
using *closedin-subset*[*OF assms*(*1*)] *assms*(*1*) **by** *auto*
with *assms*(*3*,*4*) **obtain** *e* **where** *e*: *e* > *0* *mball x e* ⊆ *M* − *A*
using *openin-mtopology* **by** *auto*
thus *?thesis*
by(*auto intro*!: *order.strict-trans2*[*OF e*(*1*) *d-set-ball-empty*[*OF assms*(*2*) *a*(*1*) *e*(*1*) *assms*(*3*)]]])
qed

lemma *gdelta-in-closed*:
assumes *closedin mtopology M*
shows *gdelta-in mtopology M*

by(*auto intro*!: *closed-imp-gdelta-in metrizable-space-mtopology*)

Oscillation

definition *osc-on* :: [*'b set*, *'b topology*, *'b* ⇒ *'a*, *'b*] ⇒ *ennreal* **where**
osc-on A X f ≡ (λy. ⨅ {*mdiameter* (*f* ‘ (*A* ∩ *U*)) | *U*. *y* ∈ *U* ∧ *openin X U*})

abbreviation *osc X* ≡ *osc-on* (*topspace X*) *X*

lemma *osc-def*: *osc X f* = (λy. ⨅ {*mdiameter* (*f* ‘ *U*) | *U*. *y* ∈ *U* ∧ *openin X U*})
by(*standard*,*auto simp*: *osc-on-def*) (*metis* (*no-types*, *opaque-lifting*) *inf*.*absorb2* *openin-subset*)

lemma *osc-on-less-iff*:
osc-on A X f x < *t* ⟷ (∃ *v*. *x* ∈ *v* ∧ *openin X v* ∧ *mdiameter* (*f* ‘ (*A* ∩ *v*)) < *t*)
by(*auto simp add*: *osc-on-def Inf-less-iff*)

lemma *osc-less-iff*:
osc X f x < *t* ⟷ (∃ *v*. *x* ∈ *v* ∧ *openin X v* ∧ *mdiameter* (*f* ‘ *v*) < *t*)
by(*auto simp add*: *osc-def Inf-less-iff*)

end

definition *mdist-set* :: *'a metric* ⇒ *'a set* ⇒ *'a* ⇒ *real* **where**
mdist-set m ≡ *Metric-space*.*d-set* (*mspace m*) (*mdist m*)

lemma(**in** *Metric-space*) *mdist-set-Self*: *mdist-set Self* = *d-set*
by(*simp add*: *mdist-set-def*)

lemma *mdist-set-nonneg*[*simp*]: *mdist-set m A x* ≥ *0*
by(*auto simp*: *mdist-set-def Metric-space*.*d-set-nonneg*)

lemma *mdist-set-singleton*[*simp*]:
x ∈ *mspace m* ⟹ *y* ∈ *mspace m* ⟹ *mdist-set m* {*y*} *x* = *mdist m x y*
by(*auto simp*: *mdist-set-def Metric-space*.*d-set-singleton*)

lemma *mdist-set-empty*[*simp*]: *mdist-set m* {} *x* = *0*
by(*auto simp*: *mdist-set-def Metric-space*.*d-set-empty*)

lemma *mdist-set-inA*:
assumes *x* ∈ *A*
shows *mdist-set m A x* = *0*
by(*auto simp*: *assms mdist-set-def Metric-space*.*d-set-inA*)

lemma *mdist-set-nzeroD*:
assumes *mdist-set m A x* ≠ *0*
shows *A* ⊆ *mspace m x* ∉ *A A* ≠ {}
using *assms Metric-space*.*d-set-nzeroD*[*of mspace m mdist m*]
by(*auto simp*: *mdist-set-def*)

lemma *mdist-set-antimono*:
  assumes $A \subseteq B$ $A \neq \{\}$ $B \subseteq$ *mspace m*
  shows *mdist-set m B x* $\leq$ *mdist-set m A x*
  by(*auto simp*: *assms mdist-set-def Metric-space.d-set-antimono*)

lemma *mdist-set-bounded*:
  assumes $\bigwedge y.\ y \in A \Longrightarrow$ *mdist m x y* $< K$ $K > 0$
  shows *mdist-set m A x* $< K$
  by(*auto simp*: *assms mdist-set-def Metric-space.d-set-bounded*)

lemma *mdist-set-tr*:
  assumes $x \in$ *mspace m* $y \in$ *mspace m*
  shows *mdist-set m A x* $\leq$ *mdist m x y* + *mdist-set m A y*
  by(*auto simp*: *assms mdist-set-def Metric-space.d-set-tr*)

lemma *mdist-set-abs-le*:
  assumes $x \in$ *mspace m* $y \in$ *mspace m*
  shows |*mdist-set m A x* − *mdist-set m A y*| $\leq$ *mdist m x y*
  by(*auto simp*: *assms mdist-set-def Metric-space.d-set-abs-le*)

lemma *mdist-set-inA-le*:
  assumes $y \in A$
  shows *mdist-set m A x* $\leq$ *mdist m x y*
  by(*auto simp*: *assms mdist-set-def Metric-space.d-set-inA-le*)

lemma *mdist-set-ball-empty*:
  assumes $A \neq \{\}$ $A \subseteq$ *mspace m* $e > 0$ $x \in$ *mspace m mball-of m x e* $\cap A = \{\}$
  shows *mdist-set m A x* $\geq e$
  by (*metis Metric-space.d-set-ball-empty Metric-space-mspace-mdist assms mball-of-def mdist-set-def*)

lemma *mdist-set-closed-pos*:
  assumes *closedin* (*mtopology-of m*) A $A \neq \{\}$ $x \in$ *mspace m* $x \notin A$
  shows *mdist-set m A x* > 0
  by (*metis Metric-space.d-set-closed-pos Metric-space-mspace-mdist assms mdist-set-def mtopology-of-def*)

lemma *mdist-set-uniformly-continuous*: *uniformly-continuous-map m euclidean-metric* (*mdist-set m A*)
  unfolding *uniformly-continuous-map-def*
proof *safe*
  fix *e* :: *real*
  assume $0 < e$
  then show $\exists\delta>0.\ \forall x\in$*mspace m*. $\forall y\in$*mspace m*. *mdist m y x* $< \delta \longrightarrow$ *mdist euclidean-metric* (*mdist-set m A y*) (*mdist-set m A x*) $< e$
    using *order.strict-trans1*[*OF mdist-set-abs-le*] **by**(*auto intro*!: *exI*[**where** *x*=*e*] *simp*: *dist-real-def*)
qed *simp*

lemma *uniformly-continuous-map-add*:
fixes f :: $'a \Rightarrow 'b$::*real-normed-vector*
assumes *uniformly-continuous-map m euclidean-metric f uniformly-continuous-map m euclidean-metric g*
shows *uniformly-continuous-map m euclidean-metric* $(\lambda x.\ f\ x + g\ x)$
unfolding *uniformly-continuous-map-def*
proof *safe*
fix e :: *real*
assume $e > 0$
from *half-gt-zero*[*OF this*] *assms* **obtain** *d1 d2* **where** *d*: $d1 > 0$ $d2 > 0$
$\bigwedge x\ y.\ x \in mspace\ m \Longrightarrow y \in mspace\ m \Longrightarrow mdist\ m\ x\ y < d1 \Longrightarrow dist\ (f\ x)\ (f\ y) < e\ /\ 2$ $\bigwedge x\ y.\ x \in mspace\ m \Longrightarrow y \in mspace\ m \Longrightarrow mdist\ m\ x\ y < d2 \Longrightarrow dist\ (g\ x)\ (g\ y) < e\ /\ 2$
by(*simp only*: *uniformly-continuous-map-def mdist-euclidean-metric*) *metis*
show $\exists\delta>0.\ \forall y\in mspace\ m.\ \forall x\in mspace\ m.\ mdist\ m\ x\ y < \delta \longrightarrow mdist$ *euclidean-metric* $(f\ x + g\ x)\ (f\ y + g\ y) < e$
using *d* **by**(*fastforce intro*!: *exI*[**where** *x=min d1 d2*] *order.strict-trans1*[*OF dist-triangle-add*])
qed *simp*

lemma *uniformly-continuous-map-real-divide*:
fixes f :: $'a \Rightarrow real$
assumes *uniformly-continuous-map m euclidean-metric f uniformly-continuous-map m euclidean-metric g*
and $\bigwedge x.\ x \in mspace\ m \Longrightarrow g\ x \neq 0$ $\bigwedge x.\ x \in mspace\ m \Longrightarrow |g\ x| \geq a$ $a > 0$ $\bigwedge x.\ x \in mspace\ m \Longrightarrow |g\ x| < Kg$
and $\bigwedge x.\ x \in mspace\ m \Longrightarrow |f\ x| < Kf$
shows *uniformly-continuous-map m euclidean-metric* $(\lambda x.\ f\ x\ /\ g\ x)$
unfolding *uniformly-continuous-map-def*
proof *safe*
fix e :: *real*
assume *e*[*arith*]:$e > 0$
consider $mspace\ m \neq \{\}$ | $mspace\ m = \{\}$ **by** *auto*
then show $\exists\delta>0.\ \forall x\in mspace\ m.\ \forall y\in mspace\ m.\ mdist\ m\ y\ x < \delta \longrightarrow mdist$ *euclidean-metric* $(f\ y\ /\ g\ y)\ (f\ x\ /\ g\ x) < e$
proof *cases*
case *m*:*1*
then have *Kfg-pos*[*arith*]: $Kg > 0$ $Kf \geq 0$
using *assms*(*4*−*7*) **by** *auto fastforce*+
define e' **where** $e' \equiv a\hat{\ }2\ /\ (Kf + Kg)\ *\ e$
have e':$e' > 0$
using *assms*(*5*) **by**(*auto simp*: e'*-def*)
with *assms* **obtain** *d1 d2* **where** *d*: $d1 > 0$ $d2 > 0$
$\bigwedge x\ y.\ x \in mspace\ m \Longrightarrow y \in mspace\ m \Longrightarrow mdist\ m\ x\ y < d1 \Longrightarrow |f\ x - f\ y| < e'$ $\bigwedge x\ y.\ x \in mspace\ m \Longrightarrow y \in mspace\ m \Longrightarrow mdist\ m\ x\ y < d2 \Longrightarrow |g\ x - g\ y| < e'$
by(*simp only*: *uniformly-continuous-map-def mdist-euclidean-metric dist-real-def*) *metis*
show *?thesis*

unfolding *mdist-euclidean-metric dist-real-def*
proof(*safe intro*!: *exI*[**where** *x*=*min d1 d2*])
fix *x y*
assume *x*:*x* ∈ *mspace m* **and** *y*:*y* ∈ *mspace m* **and** *mdist m x y* < *min d1 d2*
then have *dist*[*arith*]: *mdist m x y* < *d1 mdist m x y* < *d2* **by** *auto*
note [*arith*] = *assms*(*3*,*4*,*6*,*7*)[*OF x*] *assms*(*3*,*4*,*6*,*7*)[*OF y*]
have |*f x* / *g x* − *f y* / *g y*| = |(*f x* ∗ *g y* − *f y* ∗ *g x*) / (*g x* ∗ *g y*)|
by(*simp add*: *diff-frac-eq*)
also have ... = |(*f x* ∗ *g y* − *f x* ∗ *g x* + (*f x* ∗ *g x* − *f y* ∗ *g x*)) / (*g x* ∗ *g y*)|
by *simp*
also have ... = |(*f x* − *f y*) ∗ *g x* − *f x* ∗ (*g x* − *g y*)| / (|*g x*| ∗ |*g y*|)
by(*simp add*: *left-diff-distrib right-diff-distrib abs-mult*)
also have ... ≤ (|*f x* − *f y*| ∗ |*g x*| + |*f x*| ∗ |*g x* − *g y*|) / (|*g x*| ∗ |*g y*|)
by(*rule divide-right-mono*) (*use abs-triangle-ineq4 abs-mult* **in** *metis*,*auto*)
also have ... < (*e*′ ∗ *Kg* + *Kf* ∗ *e*′) / (|*g x*| ∗ |*g y*|)
by(*rule divide-strict-right-mono*[*OF add-less-le-mono*]) (*auto intro*!: *mult-mono*′ *mult-strict-mono*, *use* *d*(*3*,*4*)[*OF x y*] **in** *auto*)
also have ... ≤ (*e*′ ∗ *Kg* + *Kf* ∗ *e*′) / *a*^*2*
by(*auto intro*!: *divide-left-mono simp*: *power2-eq-square*) (*insert assms*(*5*) *e*′, *auto simp*: ‹*a* ≤ |*g x*|› *mult-mono*′)
also have ... = (*Kf* + *Kg*) / *a*^*2* ∗ *e*′
by (*simp add*: *distrib-left mult*.*commute*)
also have ... = *e*
using *assms*(*5*) **by**(*auto simp*: *e*′-*def*)
finally show |*f x* / *g x* − *f y* / *g y*| < *e* **.**
qed(*use d* **in** *auto*)
qed (*auto intro*!: *exI*[**where** *x*=*1*])
qed *simp*

lemma
assumes *e* > *0*
shows *uniformly-continuous-map-from-capped-metric*:*uniformly-continuous-map* (*capped-metric e m1*) *m2 f* ⟷ *uniformly-continuous-map m1 m2 f* (**is** *?g1*)
and *uniformly-continuous-map-to-capped-metric*:*uniformly-continuous-map m1* (*capped-metric e m2*) *f* ⟷ *uniformly-continuous-map m1 m2 f* (**is** *?g2*)
proof −
have [*simp*]:(λ*n*. *min e* (*X n*)) ⟶ *0* ⟷ *X* ⟶ *0* **for** *X*
proof
assume *h*:(λ*n*. *min e* (*X n*)) ⟶ *0*
show *X* ⟶ *0*
unfolding *LIMSEQ-def dist-real-def*
proof *safe*
fix *r* :: *real*
assume *0* < *r*
then have *min* (*e* / *2*) *r* > *0*
using *assms* **by** *auto*
from *LIMSEQ-D*[*OF h this*] **obtain** *N* **where** *N*:⋀*n*. *n* ≥ *N* ⟹ |*min e* (*X n*)| < *min* (*e* / *2*) *r*
by *auto*

```
      have min e (X n) = X n if h:n ≥ N  for n
      proof(rule ccontr)
        assume min e (X n) ≠ X n
        then have min e (X n) = e
          by auto
        with N[OF h] show False
          by auto
      qed
      with N show ∃no. ∀n≥no. |X n − 0| < r
        by(auto intro!: exI[where x=N])
    qed
  qed(auto intro!: tendsto-eq-intros(4)[of λx. e e sequentially X - ] simp: assms)
  show ?g1 ?g2
   using assms by(auto simp: uniformly-continuous-map-sequentially capped-metric-mdist)
qed

lemma Urysohn-lemma-uniform:
  assumes closedin (mtopology-of m) T closedin (mtopology-of m) U T ∩ U = {}
⋀x y. x ∈ T ⟹ y ∈ U ⟹ mdist m x y ≥ e e > 0
  obtains f :: 'a ⇒ real
  where uniformly-continuous-map m euclidean-metric f
        ⋀x. f x ≥ 0 ⋀x. f x ≤ 1 ⋀x. x ∈ T ⟹ f x = 1 ⋀x. x ∈ U ⟹ f x = 0
proof −
  consider T = {} | U = {} | T ≠ {} U ≠ {} by auto
  then show ?thesis
  proof cases
    case 1
    define f where f ≡ (λx::'a. 0::real)
   with 1 have uniformly-continuous-map m euclidean-metric f ⋀x. f x ≥ 0 ⋀x.
f x ≤ 1 ⋀x. x ∈ T ⟹ f x = 1 ⋀x. x ∈ U ⟹ f x = 0
      by auto
    then show ?thesis
      using that by auto
  next
    case 2
    define f where f ≡ (λx::'a. 1::real)
   with 2 have uniformly-continuous-map m euclidean-metric f ⋀x. f x ≥ 0 ⋀x.
f x ≤ 1 ⋀x. x ∈ T ⟹ f x = 1 ⋀x. x ∈ U ⟹ f x = 0
      by auto
    then show ?thesis
      using that by auto
  next
    case TU:3
    then have STU:mspace m ≠ {} T ⊆ mspace m U ⊆ mspace m
      using assms(1,2) closedin-topspace-empty closedin-subset by fastforce+
    interpret m: Metric-space mspace m mdist (capped-metric e m)
      by (metis Metric-space-mspace-mdist capped-metric-mspace)
    have e:x ∈ T ⟹ y ∈ U ⟹ mdist (capped-metric e m) x y ≥ e for x y
      by (simp add: assms(4) capped-metric-mdist)
```

define *f* **where** $f \equiv (\lambda x.$ *mdist-set* (*capped-metric e m*) *U x* / (*mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*))
have *uniformly-continuous-map m euclidean-metric f*
unfolding *uniformly-continuous-map-from-capped-metric*[*OF assms*(*5*),*of m*,*symmetric*] *f-def*
proof(*rule uniformly-continuous-map-real-divide*[**where** *Kf*=*e* + *1* **and** *Kg*=*2* ∗ *e* + *1* **and** *a*=*e* / *2*])
show *uniformly-continuous-map* (*capped-metric e m*) *euclidean-metric* (*mdist-set* (*capped-metric e m*) *U*)
uniformly-continuous-map (*capped-metric e m*) *euclidean-metric* ($\lambda x.$ *mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*)
by(*auto intro*!: *mdist-set-uniformly-continuous uniformly-continuous-map-add*)
next
fix *x*
assume *x*:$x \in$ *mspace* (*capped-metric e m*)
then consider $x \notin T \mid x \notin U$
using *TU assms* **by** *auto*
thus *mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x* $\neq$ *0*
proof *cases*
case *1*
with *TU x assms* **have** *mdist-set* (*capped-metric e m*) *T x* > *0*
by(*auto intro*!: *mdist-set-closed-pos simp*: *mtopology-capped-metric*)
thus*?thesis*
by (*metis add-less-same-cancel2 linorder-not-less mdist-set-nonneg*)
next
case *2*
with *TU x assms* **have** *mdist-set* (*capped-metric e m*) *U x* > *0*
by(*auto intro*!: *mdist-set-closed-pos simp*: *mtopology-capped-metric*)
thus*?thesis*
by (*metis add-less-same-cancel1 linorder-not-less mdist-set-nonneg*)
qed
next
fix *x*
assume *x*:$x \in$ *mspace* (*capped-metric e m*)
consider $x \in (\bigcup a \in U.$ *m.mball a* (*e* / *2*)) $\mid x \notin (\bigcup a \in U.$ *m.mball a* (*e* / *2*)) **by** *auto*
then show *e* / *2* $\leq$ |*mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*|
proof *cases*
case *1*
have *m.mball x* (*e* / *2*) $\cap$ *T* = {}
proof(*rule ccontr*)
assume *m.mball x* (*e* / *2*) $\cap$ *T* $\neq$ {}
then obtain *y* **where** *y*: $y \in$ *m.mball x* (*e* / *2*) $y \in T$
by *blast*
then obtain *u* **where** *u*:$u \in U$ $x \in$ *m.mball u* (*e* / *2*)
using *1* **by** *auto*
have *mdist* (*capped-metric e m*) *y u* $\leq$ *mdist* (*capped-metric e m*) *y x* +

mdist (*capped-metric e m*) *x u*
using *STU u y x* **by**(*auto intro*!: *m.triangle*)
also have ... < *e* / *2* + *e* / *2*
using *y u* **by**(*auto simp*: *m.commute*)
also have ... = *e* **using** *assms*(*5*) **by** *linarith*
finally show *False*
using *e*[*OF y*(*2*) *u*(*1*)] **by** *simp*
qed
from *m.d-set-ball-empty*[*OF - - - - this*] *TU STU x assms*(*1*,*5*)
have *e* / *2* ≤ *m.d-set T x*
using *STU*(*2*) *x* **by** *auto*
also have ... ≤ |*mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*|
by (*simp add*: *mdist-set-def*)
finally show *?thesis* **.**
next
case *2*
then have *m.mball x* (*e* / *2*) ∩ *U* = {}
using *x* **by** (*auto simp*: *m.commute*)
hence *e* / *2* ≤ *m.d-set U x*
by (*metis STU*(*3*) *TU*(*2*) *assms*(*5*) *capped-metric-mspace half-gt-zero m.d-set-ball-empty x*)
also have ... ≤ |*mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*|
by (*simp add*: *mdist-set-def*)
finally show *?thesis* **.**
qed
next
fix *x*
assume *x* ∈ *mspace* (*capped-metric e m*)
have |*mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*| = *mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*
using *mdist-set-nonneg* **by** *auto*
also have ... < (*e* + *1* / *2*) + (*e* + *1* / *2*)
apply(*intro add-strict-mono mdist-set-bounded*)
using *assms*(*5*) *add-strict-increasing*[*of 1* / *2*,*OF - mdist-capped*[*OF assms*(*5*),*of m x*]] **by** (*auto simp add*: *add.commute*)
also have ... = *2* ∗ *e* + *1*
by *auto*
finally show |*mdist-set* (*capped-metric e m*) *U x* + *mdist-set* (*capped-metric e m*) *T x*| < *2* ∗ *e* + *1* **.**
show |*mdist-set* (*capped-metric e m*) *U x*| < *e* + *1*
using *assms*(*5*) *add-strict-increasing*[*of 1*,*OF - mdist-capped*[*OF assms*(*5*),*of m x*]] **by** (*auto simp add*: *add.commute intro*!: *mdist-set-bounded*)
qed(*use assms* **in** *auto*)
moreover have ⋀*x*. *f x* ∈{*0*..*1*}
proof −
fix *x*
have *f x* ≤ *mdist-set* (*capped-metric e m*) *U x* / *mdist-set* (*capped-metric e*

m) *U x*
    proof −
     consider *mdist-set* (*capped-metric e m*) *U x* = *0* | *mdist-set* (*capped-metric e m*) *U x* > *0*
        using *antisym-conv1* **by** *fastforce*
      thus *?thesis*
      proof *cases*
        case *2*
        show *?thesis*
          unfolding *f-def* **by**(*rule divide-left-mono*[*OF - - mult-pos-pos*]) (*auto simp*: *2 add-strict-increasing*)
      qed (*simp add*: *f-def*)
    qed
    also have ... ≤ *1* **by** *simp*
    finally show *f x* ∈ {*0*..*1*}
      by (*auto simp*: *f-def*)
  qed
  moreover have *f x* = *1* **if** *x*:*x* ∈ *T* **for** *x*
  proof −
    { assume *h*:*mdist-set* (*capped-metric e m*) *U x* = *0*
      then have *x* ∉ *U* **using** *assms STU x* **by** *blast*
      hence *False*
     by (*metis STU*(*2*) *TU*(*2*) *assms*(*2*) *capped-metric-mspace h mdist-set-closed-pos mtopology-capped-metric order-less-irrefl subset-eq x*)
    }
    thus *?thesis*
      by (*metis add.right-neutral divide-self-if f-def mdist-set-nzeroD*(*2*) *x*)
  qed
  moreover have ⋀*x*. *x* ∈ *U* ⟹ *f x* = *0*
    by (*simp add*: *f-def m.d-set-inA mdist-set-def*)
  ultimately show *?thesis*
    using *that* **by** *auto*
 qed
qed

Open maps

lemma *Metric-space-open-map-from-dist*:
 assumes *f* ∈ *mspace m1* → *mspace m2*
    and ⋀*x ε*. *x* ∈ *mspace m1* ⟹ *ε* > *0* ⟹ ∃ *δ*>*0*. ∀ *y*∈*mspace m1*. *mdist m2* (*f x*) (*f y*) < *δ* ⟶ *mdist m1 x y* < *ε*
  shows *open-map* (*mtopology-of m1*) (*subtopology* (*mtopology-of m2*) (*f* ‘ *mspace m1*)) *f*
proof −
 interpret *m1*: *Metric-space mspace m1 mdist m1* **by** *simp*
 interpret *m2*: *Metric-space mspace m2 mdist m2* **by** *simp*
 interpret *m2'*: *Submetric mspace m2 mdist m2 f* ‘ *mspace m1*
   using *assms*(*1*) **by**(*auto simp*: *Submetric-def Submetric-axioms-def*)
 have *open-map m1.mtopology m2'.sub.mtopology f*
 proof(*safe intro*!: *open-map-with-base*[*OF m1.mtopology-base-in-balls*])

```
    fix a and e :: real
    assume h:a ∈ mspace m1 0 < e
    show openin m2'.sub.mtopology (f ` m1.mball a e)
      unfolding m2'.sub.openin-mtopology
    proof safe
      fix x
      assume x:x ∈ m1.mball a e
      then have xs:x ∈ mspace m1
        by auto
      have 0 < e − mdist m1 a x
        using x by auto
       from assms(2)[OF xs this] obtain d where d:d > 0 ⋀y. y ∈ mspace m1
⟹ mdist m2 (f x) (f y) < d ⟹ mdist m1 x y < e − mdist m1 a x
        by auto
      show ∃r>0. m2'.sub.mball (f x) r ⊆ f ` m1.mball a e
      proof(safe intro!: exI[where x=d])
        fix z
        assume z:z ∈ m2'.sub.mball (f x) d
        then obtain y where y:z = f y y ∈ mspace m1
          by auto
        hence mdist m2 (f x) (f y) < d
          using z by simp
        hence mdist m1 x y < e − mdist m1 a x
          using d y by auto
        hence mdist m1 a y < e
          using h(1) x y m1.triangle[of a x y] by auto
        with h(1) show z ∈ f ` m1.mball a e
          by(auto simp: y)
      qed fact
    qed auto
  qed
   thus open-map (mtopology-of m1) (subtopology (mtopology-of m2) (f ` mspace
m1)) f
    by (simp add: mtopology-of-def m2'.mtopology-submetric)
qed
```

1.3.3 Separability in Metric Spaces

```
context Metric-space
begin
```

For a metric space M, M is separable iff M is second countable.

```
lemma generated-by-countable-balls:
  assumes countable U and mdense U
  shows mtopology = topology-generated-by {mball y (1 / real n) | y n. y ∈ U}
proof −
  have hu: U ⊆ M ⋀x ε. x ∈ M ⟹ ε > 0 ⟹ mball x ε ∩ U ≠ {}
    using assms by(auto simp: mdense-def)
  show ?thesis
```

```
    unfolding base-is-subbase[OF mtopology-base-in-balls,simplified subbase-in-def]
  proof(rule topology-generated-by-eq)
    fix K
    assume K ∈ {mball y (1 / real n) |y n. y ∈ U}
    then show openin (topology-generated-by {mball a ε |a ε. a ∈ M ∧ 0 < ε}) K
     by(auto simp: base-is-subbase[OF mtopology-base-in-balls,simplified subbase-in-def,symmetric])
  next
    have h0:⋀x ε. x ∈ M ⟹ ε > 0 ⟹ ∃y∈U. ∃n. x ∈ mball y (1 / real n) ∧
mball y (1 / real n) ⊆ mball x ε
    proof −
      fix x ε
      assume h: x ∈ M (0 :: real) < ε
      then obtain N where hn: 1 / ε < real N
        using reals-Archimedean2 by blast
      have hn0: 0 < real N
        by(rule ccontr) (use hn h in fastforce)
      hence hn':1 / real N < ε
        using h hn by (metis divide-less-eq mult.commute)
      have mball x (1 / (2 ∗ real N)) ∩ U ≠ {}
        using mdense-def[of U] assms(2) h(1) hn0 by fastforce
      then obtain y where hy:
        y∈U y ∈ M y ∈ mball x (1 / (real (2 ∗ N)))
        using hu by auto
      show ∃y∈U. ∃n. x ∈ mball y (1 / real n) ∧ mball y (1 / real n) ⊆ mball x ε
      proof(intro bexI[where x=y] exI[where x=2 ∗ N] conjI)
        show x ∈ mball y (1 / real (2 ∗ N))
          using hy(3) by (auto simp: commute)
      next
        show mball y (1 / real (2 ∗ N)) ⊆ mball x ε
        proof
          fix y'
          assume hy':y' ∈ mball y (1 / real (2 ∗ N))
          have d x y' < ε (is ?lhs < ?rhs)
          proof −
            have ?lhs ≤ d x y + d y y'
              using hy(2) hy' h(1) triangle by auto
            also have ... < 1 / real (2 ∗ N) + 1 / real (2 ∗ N)
                 by(rule strict-ordered-ab-semigroup-add-class.add-strict-mono) (use
hy(3) hy(2) hy' h(1) hy' in auto)
            finally show ?thesis
              using hn' by auto
          qed
          thus y' ∈ mball x ε
            using hy' h(1) by auto
        qed
      qed fact
    qed
    fix K
    assume hk: K ∈ {mball a ε |a ε. a ∈ M ∧ 0 < ε}
```

```
    then obtain x εx where hxe:
      x ∈ M 0 < εx K = mball x εx by auto
    have gh:K = (⋃{mball y (1 / real n) | y n. y ∈ U ∧ mball y (1 / real n) ⊆
K})
    proof
      show K ⊆ (⋃ {mball y (1 / real n) |y n. y ∈ U ∧ mball y (1 / real n) ⊆
K})
      proof
        fix k
        assume hkink:k ∈ K
        then have hkinS:k ∈ M
          by(simp add: hxe)
        obtain εk where hek: εk > 0 mball k εk ⊆ K
         by (metis Metric-space.openin-mtopology Metric-space-axioms hxe(3) hkink
openin-mball)
        obtain y n where hyey:
          y ∈ U k ∈ mball y (1 / real n) mball y (1 / real n) ⊆ mball k εk
          using h0[OF hkinS hek(1)] by auto
       show k ∈ ⋃ {mball y (1 / real n) |y n. y ∈ U ∧ mball y (1 / real n) ⊆ K}
          using hek(2) hyey by blast
      qed
    qed auto
    show openin (topology-generated-by {mball y (1 / real n) |y n. y ∈ U}) K
      unfolding openin-topology-generated-by-iff
       apply(rule generate-topology-on.UN[of {mball y (1 / real n) |y n. y ∈ U ∧
mball y (1 / real n) ⊆ K}, simplified gh[symmetric]])
      apply(rule generate-topology-on.Basis) by auto
  qed
qed

lemma separable-space-imp-second-countable:
  assumes separable-space mtopology
  shows second-countable mtopology
proof −
  obtain U where hu:
   countable U mdense U
    using assms separable-space-def2 by blast
  show ?thesis
   proof(rule countable-base-from-countable-subbase[where 𝒪={mball y (1 / real
n) | y n. y ∈ U}])
    have {mball y (1 / real n) |y n. y ∈ U} = (λ(y,n). mball y (1 / real n)) ‘ (U
× UNIV)
      by auto
    also have countable ...
      using hu by auto
    finally show countable {mball y (1 / real n) |y n. y ∈ U} .
  qed(use subbase-in-def generated-by-countable-balls[of U] hu in auto)
qed
```

```
corollary separable-space-iff-second-countable:
  separable-space mtopology ⟷ second-countable mtopology
  using second-countable-imp-separable-space separable-space-imp-second-countable
by auto

lemma Lindelof-mdiameter:
  assumes separable-space mtopology 0 < e
  shows ∃ U. countable U ∧ ⋃ U = M ∧ (∀ u∈U. mdiameter u < ennreal e)
proof −
  have (⋀u. u ∈ {mball x (e / 3) |x. x ∈ M} ⟹ openin mtopology u)
    by auto
  moreover have ⋃ {mball x (e / 3) |x. x ∈ M} = M
    using assms by auto
  ultimately have ∃ U'. countable U' ∧ U' ⊆ {mball x (e / 3) |x. x ∈ M} ∧ ⋃
U' = M
   using second-countable-imp-Lindelof-space[OF assms(1)[simplified separable-space-iff-second-countable]]
    by(auto simp: Lindelof-space-def)
  then obtain U' where U': countable U' U' ⊆ {mball x (e / 3) |x. x ∈ M} ⋃
U' = M
    by auto
  show ?thesis
  proof(safe intro!: exI[where x=U'])
    fix u
    assume u ∈ U'
    then obtain x where u:u = mball x (e / 3)
      using U' by auto
    have mdiameter u ≤ ennreal (2 ∗ (e / 3))
      by(simp only: u mdiameter-ball-leq)
    also have ... < ennreal e
      by(auto intro!: ennreal-lessI assms)
    finally show mdiameter u < ennreal e .
  qed(use U' in auto)
qed

end

lemma metrizable-space-separable-iff-second-countable:
  assumes metrizable-space X
  shows separable-space X ⟷ second-countable X
proof −
  obtain d where Metric-space (topspace X) d Metric-space.mtopology (topspace
X) d = X
    by (metis assms(1) Metric-space.topspace-mtopology metrizable-space-def)
  thus ?thesis
    using Metric-space.separable-space-iff-second-countable by fastforce
qed

abbreviation mdense-of m U ≡ dense-in (mtopology-of m) U
```

lemma *mdense-of-def*: *mdense-of m U* $\longleftrightarrow$ (*U* $\subseteq$ *mspace m* $\wedge$ ($\forall$*x*$\in$*mspace m*. $\forall\varepsilon$>*0*. *mball-of m x* ε $\cap$ *U* $\neq$ {}))
 using *Metric-space.mdense-def*[*of mspace m mdist m*] **by** (*simp add*: *mball-of-def mtopology-of-def*)

lemma *mdense-of-def2*: *mdense-of m U* $\longleftrightarrow$ (*U* $\subseteq$ *mspace m* $\wedge$ ($\forall$*x*$\in$*mspace m*. $\forall\varepsilon$>*0*. $\exists$*y*$\in$*U*. *mdist m x y* < ε))
 using *Metric-space.mdense-def2*[*of mspace m mdist m*] **by** (*simp add*: *mtopology-of-def*)

lemma *mdense-of-def3*: *mdense-of m U* $\longleftrightarrow$ (*U* $\subseteq$ *mspace m* $\wedge$ ($\forall$*x*$\in$*mspace m*. $\exists$*xn*. *range xn* $\subseteq$ *U* $\wedge$ *limitin* (*mtopology-of m*) *xn x sequentially*))
 using *Metric-space.mdense-def3*[*of mspace m mdist m*] **by** (*simp add*: *mtopology-of-def*)

1.3.4 Compact Metric Spaces

context *Metric-space*
begin

lemma *mtotally-bounded-eq-compact-closedin*:
 assumes *mcomplete closedin mtopology S*
 shows *mtotally-bounded S* $\longleftrightarrow$ *S* $\subseteq$ *M* $\wedge$ *compactin mtopology S*
 by (*metis assms closure-of-eq mtotally-bounded-eq-compact-closure-of*)

lemma *mtotally-bounded-def2*: *mtotally-bounded S* $\longleftrightarrow$ ($\forall\varepsilon$>*0*. $\exists$*K*. *finite K* $\wedge$ *K* $\subseteq$ *M* $\wedge$ *S* $\subseteq$ ($\bigcup$*x*$\in$*K*. *mball x* ε))
 unfolding *mtotally-bounded-def*
proof *safe*
 fix *e* :: *real*
 assume *h*:*e* > *0* $\forall$ *e*>*0*. $\exists$*K*. *finite K* $\wedge$ *K* $\subseteq$ *S* $\wedge$ *S* $\subseteq$ ($\bigcup$*x*$\in$*K*. *mball x e*)
 then show $\exists$*K*. *finite K* $\wedge$ *K* $\subseteq$ *M* $\wedge$ *S* $\subseteq$ ($\bigcup$*x*$\in$*K*. *mball x e*)
 by (*metis Metric-space.mbounded-subset Metric-space.mtotally-bounded-imp-mbounded Metric-space-axioms mbounded-subset-mspace mtotally-bounded-def*)
next
 fix *e* :: *real*
 assume *e* > *0* $\forall\varepsilon$>*0*. $\exists$*K*. *finite K* $\wedge$ *K* $\subseteq$ *M* $\wedge$ *S* $\subseteq$ ($\bigcup$*x*$\in$*K*. *mball x* ε)
 then obtain *K* **where** *K*: *finite K K* $\subseteq$ *M S* $\subseteq$ ($\bigcup$*x*$\in$*K*. *mball x* (*e* / *2*))
  by (*meson half-gt-zero*)
 define *K'* **where** *K'* $\equiv$ {*x*$\in$*K*. *mball x* (*e* / *2*) $\cap$ *S* $\neq$ {}}
 have *K'1*:*finite K' K'* $\subseteq$ *M*
  using *K* **by**(*auto simp*: *K'-def*)
 have *K'2*: *S* $\subseteq$ ($\bigcup$*x*$\in$*K'*. *mball x* (*e* / *2*))
 proof
  fix *x*
  assume *x*:*x* $\in$ *S*
  then obtain *k* **where** *k*:*k* $\in$ *K x* $\in$ *mball k* (*e* / *2*)
   using *K* **by** *auto*
  with *x* **have** *k* $\in$ *K'*

by(*auto simp*: *K′-def*)
with *k* **show** $x \in (\bigcup x{\in}K'.\ mball\ x\ (e\ /\ 2))$
by *auto*
qed
have $\forall k{\in}K'.\ \exists y.\ y \in mball\ k\ (e\ /\ 2) \cap S$
by(*auto simp*: *K′-def*)
then obtain *xk* **where** *xk*: $\bigwedge k.\ k \in K' \Longrightarrow xk\ k \in mball\ k\ (e\ /\ 2)$ $\bigwedge k.\ k \in K' \Longrightarrow xk\ k \in S$
by (*metis IntD2 inf-commute*)
hence $\bigwedge k.\ k \in K' \Longrightarrow mball\ k\ (e\ /\ 2) \subseteq mball\ (xk\ k)\ e$
using *triangle commute* **by** *fastforce*
hence $(\bigcup x{\in}K'.\ mball\ x\ (e\ /\ 2)) \subseteq (\bigcup x{\in}xk\ `\ K'.\ mball\ x\ e)$
by *blast*
with *K′2* **have** $S \subseteq (\bigcup x{\in}xk\ `\ K'.\ mball\ x\ e)$
by *blast*
thus $\exists K.\ finite\ K \wedge K \subseteq S \wedge S \subseteq (\bigcup x{\in}K.\ mball\ x\ e)$
by(*intro exI*[**where** *x=xk ‘ K′*]) (*use xk*(*2*) *K′1*(*1*) **in** *blast*)
qed

lemma *compact-space-imp-separable*:
assumes *compact-space mtopology*
shows *separable-space mtopology*
proof −
have $\exists A.\ finite\ A \wedge A \subseteq M \wedge M \subseteq \bigcup\ ((\lambda a.\ mball\ a\ (1\ /\ real\ (Suc\ n)))\ `\ A)$ **for** *n*
using *assms* **by**(*auto simp*: *compact-space-eq-mcomplete-mtotally-bounded mtotally-bounded-def*)
then obtain *A* **where** *A*: $\bigwedge n.\ finite\ (A\ n)$ $\bigwedge n.\ A\ n \subseteq M$ $\bigwedge n.\ M \subseteq \bigcup\ ((\lambda a.\ mball\ a\ (1\ /\ real\ (Suc\ n)))\ `\ (A\ n))$
by *metis*
define *K* **where** $K \equiv \bigcup\ (range\ A)$
have *1*: *countable K*
using *A*(*1*) **by**(*auto intro*!: *countable-UN*[*of - id*,*simplified*] *simp*: *K-def countable-finite*)
show *separable-space mtopology*
unfolding *mdense-def2 separable-space-def2*
proof(*safe intro*!: *exI*[**where** *x=K*] *1*)
fix *x* **and** ε :: *real*
assume *h*: $x \in M$ $0 < \varepsilon$
then obtain *n* **where** *n*:$1\ /\ real\ (Suc\ n) \le \varepsilon$
by (*meson nat-approx-posE order.strict-iff-not*)
then obtain *y* **where** *y*: $y \in A\ n$ $x \in mball\ y\ (1\ /\ real\ (Suc\ n))$
using *h*(*1*) *A*(*3*)[*of n*] **by** *auto*
thus $\exists y{\in}K.\ d\ x\ y < \varepsilon$
using *n* **by**(*auto intro*!: *bexI*[**where** *x=y*] *simp*: *commute K-def*)
qed(*use K-def A*(*2*) **in** *auto*)
qed

lemma *separable-space-cfunspace*:

assumes *separable-space mtopology mcomplete*
and *metrizable-space X compact-space X*
shows *separable-space* (*mtopology-of* (*cfunspace X Self*))
proof −
consider *topspace X* = {} | *topspace X* ≠ {} *M* = {} | *topspace X* ≠ {} *M* ≠ {} **by** *auto*
thus *?thesis*
proof *cases*
case *1*
then show *?thesis*
by(*auto intro*!: *countable-space-separable-space*)
next
case *2*
then have [*simp*]:*mtopology* = *trivial-topology*
using *topspace-mtopology* **by** *auto*
have *1*:*topspace* (*mtopology-of* (*cfunspace X Self*)) = {}
apply *simp*
using *2*(*1*) **by**(*auto simp*: *mtopology-of-def*)
show *?thesis*
by(*rule countable-space-separable-space*, *simp only*: *1*) *simp*
next
case *XS-nem*:*3*
have *cm*: *mcomplete-of* (*cfunspace X Self*)
by (*simp add*: *assms*(*2*) *mcomplete-cfunspace*)
show *?thesis*
proof −
obtain *dx* **where** *dx*: *Metric-space* (*topspace X*) *dx Metric-space.mtopology* (*topspace X*) *dx* = *X*
by (*metis Metric-space.topspace-mtopology assms*(*3*) *metrizable-space-def*)
interpret *dx*: *Metric-space topspace X dx*
by *fact*
have *dx-c*: *dx.mcomplete*
using *assms* **by**(*auto intro*!: *dx.compact-space-imp-mcomplete simp*: *dx*(*2*))
have ∃ *B*. *finite B* ∧ *B* ⊆ *topspace X* ∧ *topspace X* ⊆ (⋃ *a*∈*B*. *dx.mball a* (*1* / *Suc m*)) **for** *m*
using *dx.compactin-imp-mtotally-bounded*[*of topspace X*] *assms*(*4*)
by(*fastforce simp*: *dx*(*2*) *compact-space-def dx.mtotally-bounded-def2*)
then obtain *Xm* **where** *Xm*: ⋀*m*. *finite* (*Xm m*) ⋀*m*. (*Xm m*) ⊆ *topspace X* ⋀*m*. *topspace X* ⊆ (⋃ *a*∈*Xm m*. *dx.mball a* (*1* / *Suc m*))
by *metis*
hence *Xm-eq*: ⋀*m*. *topspace X* = (⋃ *a*∈*Xm m*. *dx.mball a* (*1* / *Suc m*))
by *fastforce*
have *Xm-nem*:⋀*m*. (*Xm m*) ≠ {}
using *XS-nem Xm-eq* **by** *blast*
define *xmk* **where** *xmk* ≡ (λ*m*. *from-nat-into* (*Xm m*))
define *km* **where** *km* ≡ (λ*m*. *card* (*Xm m*))
have *km-pos*:*km m* > *0* **for** *m*
by(*simp add*: *km-def card-gt-0-iff Xm Xm-nem*)
have *xmk-bij*: *bij-betw* (*xmk m*) {..<*km m*} (*Xm m*) **for** *m*

using *bij-betw-from-nat-into-finite*[*OF Xm*(*1*)] **by**(*simp add*: *km-def xmk-def*)
have *xmk-into*: *xmk m i* ∈ *Xm m* **for** *m i*
by (*simp add*: *Xm-nem from-nat-into xmk-def*)
have ∃ *U*. *countable U* ∧ ⋃ *U* = *M* ∧ (∀ *u*∈*U*. *mdiameter u* < *1* / (*Suc l*)) **for** *l*
by(*rule Lindelof-mdiameter*[*OF assms*(*1*)]) *auto*
then obtain *U* **where** *U*: ⋀*l*. *countable* (*U l*) ⋀*l*. (⋃ (*U l*)) = *M* ⋀*l u*. *u* ∈ *U l* ⟹ *mdiameter u* < *1* / *Suc l*
by *metis*
have *Ul-nem*: *U l* ≠ {} **for** *l*
using *XS-nem U*(*2*) **by** *auto*
define *uli* **where** *uli* ≡ (λ*l*. *from-nat-into* (*U l*))
have *uli-into*:*uli l i* ∈ *U l* **for** *l i*
by (*simp add*: *Ul-nem from-nat-into uli-def*)
hence *uli-diam*: *mdiameter* (*uli l i*) < *1* / *Suc l* **for** *l i*
using *U*(*3*) **by** *auto*
have *uli-un*:*M* = (⋃*i*. *uli l i*) **for** *l*
by(*auto simp*: *range-from-nat-into*[*OF Ul-nem*[*of l*] *U*(*1*)] *uli-def U*)
define *Cmn* **where** *Cmn* ≡ (λ*m n*. {*f* ∈ *mspace* (*cfunspace X Self*). ∀ *x*∈*topspace X*. ∀ *y*∈*topspace X*. *dx x y* < *1* / (*Suc m*) ⟶ *d* (*f x*) (*f y*) < *1* / *Suc n*})
define *fmnls* **where** *fmnls* ≡ (λ*m n l s*. *SOME f*. *f* ∈ *Cmn m n* ∧ (∀ *j*<*km m*. *f* (*xmk m j*) ∈ *uli l* (*s j*)))
define *Dmnl* **where** *Dmnl* ≡ (λ*m n l*. {*fmnls m n l s* |*s*. *s* ∈ {..<*km m*} $\rightarrow_E$ *UNIV* ∧ (∃ *f* ∈ *Cmn m n*. (∀ *j*<*km m*. *f* (*xmk m j*) ∈ *uli l* (*s j*))) })
have *in-Dmnl*: *fmnls m n l s* ∈ *Dmnl m n l* **if** *s*∈{..<*km m*} $\rightarrow_E$ *UNIV f*∈ *Cmn m n* ∀ *j*<*km m*. *f* (*xmk m j*) ∈ *uli l* (*s j*)**for** *m n l s f*
using *Dmnl-def that* **by** *blast*
define *Dmn* **where** *Dmn* ≡ (λ*m n*. ⋃*l*. *Dmnl m n l*)
have *Dmn-subset*: *Dmn m n* ⊆ *Cmn m n* **for** *m n*
proof −
have *Dmnl m n l* ⊆ *Cmn m n* **for** *l*
by(*auto simp*: *Dmnl-def fmnls-def someI*[*of* λ*f*. *f* ∈ *Cmn m n* ∧ (∀ *j*<*km m*. *f* (*xmk m j*) ∈ *uli l* (- *j*))])
thus *?thesis* **by**(*auto simp*: *Dmn-def*)
qed
have *c-Dmn*: *countable* (*Dmn m n*) **for** *m n*
proof −
have *countable* (*Dmnl m n l*) **for** *l*
proof −
have *1*:*Dmnl m n l* ⊆ (λ*s*. *fmnls m n l s*) ‘ ({..<*km m*} $\rightarrow_E$ *UNIV*)
by(*auto simp*: *Dmnl-def*)
have *countable* ...
by(*auto intro*!: *countable-PiE*)
with *1* **show** *?thesis*
using *countable-subset* **by** *blast*
qed
thus *?thesis*
by(*auto simp*: *Dmn-def*)

qed
have *claim*: $\exists g{\in}Dmn\ m\ n.\ \forall y{\in}Xm\ m.\ d\ (f\ y)\ (g\ y) < e$ **if** *f*:$f \in Cmn\ m\ n$ **and** *e*:$e > 0$ **for** *f m n e*
proof −
obtain *l* **where** *l*:$1\ /\ Suc\ l < e$
using *e nat-approx-posE* **by** *blast*
define *s* **where** $s \equiv (\lambda i{\in}\{..<km\ m\}.\ SOME\ j.\ f\ (xmk\ m\ i) \in uli\ l\ j)$
have *s1*:$s{\in}\{..<km\ m\} \rightarrow_E UNIV$ **by**(*simp add*: *s-def*)
have *s2*:$\forall i{<}km\ m.\ f\ (xmk\ m\ i) \in uli\ l\ (s\ i)$
proof −
fix *i*
have $f\ (xmk\ m\ i) \in uli\ l\ (SOME\ j.\ f\ (xmk\ m\ i) \in uli\ l\ j)$ **for** *i*
proof(*rule someI-ex*)
have $xmk\ m\ i \in topspace\ X$
using *Xm*(*2*) *xmk-into* **by** *auto*
hence $f\ (xmk\ m\ i) \in M$
using *f* **by**(*auto simp*: *Cmn-def continuous-map-def*)
thus $\exists x.\ f\ (xmk\ m\ i) \in uli\ l\ x$
using *uli-un* **by** *auto*
qed
thus *?thesis*
by (*auto simp*: *s-def*)
qed
have *fmnls*:$fmnls\ m\ n\ l\ s \in Cmn\ m\ n \wedge (\forall j{<}km\ m.\ fmnls\ m\ n\ l\ s\ (xmk\ m\ j) \in uli\ l\ (s\ j))$
by(*simp add*: *fmnls-def*,*rule someI*[**where** *x=f*],*auto simp*: *s2 f*)
show $\exists g{\in}Dmn\ m\ n.\ \forall y{\in}Xm\ m.\ d\ (f\ y)\ (g\ y) < e$
proof(*safe intro*!: *bexI*[**where** *x=fmnls m n l s*])
fix *y*
assume *y*:$y \in Xm\ m$
then obtain *i* **where** *i*:$i < km\ m$ $xmk\ m\ i = y$
by (*meson xmk-bij*[*of m*] *bij-betw-iff-bijections lessThan-iff*)
have $f\ y \in uli\ l\ (s\ i)$ $fmnls\ m\ n\ l\ s\ y \in uli\ l\ (s\ i)$
using *i*(*1*) *s2 fmnls* **by**(*auto simp*: *i*(*2*)[*symmetric*])
moreover have $f\ y \in M$ $fmnls\ m\ n\ l\ s\ y \in M$
using *f fmnls y Xm*(*2*)[*of m*] **by**(*auto simp*: *Cmn-def continuous-map-def*)
ultimately have $ennreal\ (d\ (f\ y)\ (fmnls\ m\ n\ l\ s\ y)) \le mdiameter\ (uli\ l\ (s\ i))$
by(*auto intro*!: *mdiameter-is-sup*)
also have $... < ennreal\ (1\ /\ Suc\ l)$
by(*rule uli-diam*)
also have $... < ennreal\ e$
using *l e* **by**(*auto intro*!: *ennreal-lessI*)
finally show $d\ (f\ y)\ (fmnls\ m\ n\ l\ s\ y) < e$
by(*simp add*: *ennreal-less-iff*[*OF nonneg*])
qed(*use in-Dmnl*[*OF s1 f s2*] *Dmn-def* **in** *auto*)
qed
show *separable-space* (*mtopology-of* (*cfunspace X Self*))
unfolding *separable-space-def2 mdense-of-def2*

proof(*safe intro*!: *exI*[**where** *x*=⋃ *n*. (⋃ *m*. *Dmn m n*)])
fix *f* **and** *e* :: *real*
assume *h*:*f* ∈ *mspace* (*cfunspace X Self*) *0* < *e*
then obtain *n* **where** *n*:*1* / *Suc n* < *e* / *4*
by (*metis zero-less-divide-iff zero-less-numeral nat-approx-posE*)
have ∃ *m*. ∀ *l*≥ *m*. *f* ∈ *Cmn l n*
proof −
have *uniformly-continuous-map dx*.*Self Self f*
using *continuous-eq-uniformly-continuous-map*[*of dx*.*Self Self f*] *h assms*(*4*) **by**(*auto simp*: *mtopology-of-def dx*)
then obtain δ **where** δ:δ > *0* ⋀*x y*. *x*∈*topspace X* ⟹ *y*∈*topspace X* ⟹ *dx x y* < δ ⟹ *d* (*f x*) (*f y*) < *1* / (*Suc n*)
by(*simp only*: *uniformly-continuous-map-def*) (*metis dx*.*mdist-Self dx*.*mspace-Self mdist-Self of-nat-0-less-iff zero-less-Suc zero-less-divide-1-iff*)
then obtain *m* **where** *m*:*1* / *Suc m* < δ
using *nat-approx-posE* **by** *blast*
have *l*: *l* ≥ *m* ⟹ *1* / *Suc l* ≤ *1* / *Suc m* **for** *l*
by (*simp add*: *frac-le*)
show *?thesis*
using δ(*2*)[*OF* - - *order*.*strict-trans*[*OF* - *order*.*strict-trans1*[*OF l m*]]] *h* **by**(*auto simp*: *Cmn-def*)
qed
then obtain *m* **where** *m*:*f* ∈ *Cmn m n* **by** *auto*
obtain *g* **where** *g*:*g*∈*Dmn m n* ⋀*y*. *y*∈*Xm m* ⟹ *d* (*f y*) (*g y*) < *e* / *4*
by (*metis claim*[*OF m*] *h*(*2*) *zero-less-divide-iff zero-less-numeral*)
have ∃ *n m*. ∃ *g*∈*Dmn m n*. *mdist* (*cfunspace X Self*) *f g* < *e*
proof(*rule exI*[**where** *x*=*n*])
show ∃ *m*. ∃ *g*∈*Dmn m n*. *mdist* (*cfunspace X Self*) *f g* < *e*
proof(*intro exI*[**where** *x*=*m*] *bexI*[*OF* - *g*(*1*)])
have *g-cm*:*g* ∈ *mspace* (*cfunspace X Self*)
using *g*(*1*) *Dmn-subset*[*of m n*] **by**(*auto simp*: *Cmn-def*)
have *mdist* (*cfunspace X Self*) *f g* ≤ *3* ∗ *e* / *4*
proof(*rule mdist-cfunspace-le*)
fix *x*
assume *x*:*x* ∈ *topspace X*
obtain *y* **where** *y*:*y* ∈ *Xm m x* ∈ *dx*.*mball y* (*1* / *real* (*Suc m*))
using *Xm*(*3*) *x* **by** *fastforce*
hence *ytop*:*y* ∈ *topspace X*
by *auto*
have *mdist Self* (*f x*) (*g x*) ≤ *d* (*f x*) (*f y*) + *d* (*f y*) (*g x*)
using *x g-cm h*(*1*) *ytop* **by**(*auto intro*!: *triangle simp*: *continuous-map-def*)
also have ... ≤ *d* (*f x*) (*f y*) + *d* (*f y*) (*g y*) + *d* (*g y*) (*g x*)
using *x g-cm h*(*1*) *ytop* **by**(*auto intro*!: *triangle simp*: *continuous-map-def*)
also have ... ≤ *e* / *4* + *e* / *4* + *e* / *4*
proof −
have *dxy*: *dx x y* < *1* / *Suc m dx y x* < *1* / *Suc m*
using *y*(*2*) **by**(*auto simp*: *dx*.*commute*)

hence *d* (*f x*) (*f y*) < *1* / (*Suc n*) *d* (*g y*) (*g x*) < *1* / (*Suc n*)
  using *m x ytop g Dmn-subset*[*of m n*] **by**(*auto simp*: *Cmn-def*)
hence *d* (*f x*) (*f y*) < *e* / *4* *d* (*g y*) (*g x*) < *e* / *4*
  using *n* **by** *auto*
thus *?thesis*
  using *g*(*2*)[*OF y*(*1*)] **by** *auto*
qed
finally show *mdist Self* (*f x*) (*g x*) ≤ *3* ∗ *e* / *4*
  by *simp*
qed(*use h* **in** *auto*)
also have ... < *e*
  using *h* **by** *auto*
finally show *mdist* (*cfunspace X Self*) *f g* < *e* **.**
qed
qed
thus ∃ *y*∈⋃ *n m*. *Dmn m n*. *mdist* (*cfunspace X Self*) *f y* < *e*
  by *auto*
qed(*insert Dmn-subset c-Dmn*,*unfold Cmn-def*, *blast*)+
qed
qed
qed

end

context *Submetric*
begin

lemma *separable-sub*:
  assumes *separable-space mtopology*
  shows *separable-space sub.mtopology*
  using *assms* **unfolding** *separable-space-iff-second-countable sub.separable-space-iff-second-countable*
  by(*auto simp*: *second-countable-subtopology mtopology-submetric*)

end

1.3.5 Discrete Distance

lemma(**in** *discrete-metric*) *separable-space-iff*: *separable-space disc.mtopology* ⟷ *countable M*
  by(*simp add*: *mtopology-discrete-metric separable-space-discrete-topology*)

1.3.6 Binary Product Metric Spaces

We define the L^1-distance. L^1-distance and L^2 distance (Euclid distance) generate the same topological space.

definition *prod-dist-L1* ≡ λ*d1 d2* (*x*,*y*) (*x*′,*y*′). *d1 x x*′ + *d2 y y*′

context *Metric-space12*
begin

lemma *prod-L1-metric*: *Metric-space* (*M1* × *M2*) (*prod-dist-L1 d1 d2*)
proof
  fix *x y z*
  assume *x* ∈ *M1* × *M2 y* ∈ *M1* × *M2 z* ∈ *M1* × *M2*
  then show *prod-dist-L1 d1 d2 x z* ≤ *prod-dist-L1 d1 d2 x y* + *prod-dist-L1 d1 d2 y z*
    by(*auto simp*: *prod-dist-L1-def*) (*metis M1.commute M1.triangle′′ M2.commute M2.triangle′′ ab-semigroup-add-class.add-ac*(*1*) *add.left-commute add-mono*)
qed(*auto simp*: *prod-dist-L1-def add-nonneg-eq-0-iff M1.commute M2.commute*)

sublocale *Prod-metric-L1*: *Metric-space M1* × *M2 prod-dist-L1 d1 d2*
  by(*simp add*: *prod-L1-metric*)

lemma *prod-dist-L1-geq*:
  shows *d1 x y* ≤ *prod-dist-L1 d1 d2* (*x*,*x′*) (*y*,*y′*)
        d2 x′ y′ ≤ *prod-dist-L1 d1 d2* (*x*,*x′*) (*y*,*y′*)
  by(*auto simp*: *prod-dist-L1-def*)

lemma *prod-dist-L1-ball*:
  assumes (*x*,*x′*) ∈ *Prod-metric-L1.mball* (*a*,*a′*) *ε*
    shows *x* ∈ *M1.mball a ε*
      and *x′* ∈ *M2.mball a′ ε*
  using *assms prod-dist-L1-geq order.strict-trans1* **by** *simp-all blast+*

lemma *prod-dist-L1-ball′*:
  assumes *z* ∈ *Prod-metric-L1.mball a ε*
    shows *fst z* ∈ *M1.mball* (*fst a*) *ε*
      and *snd z* ∈ *M2.mball* (*snd a*) *ε*
  using *prod-dist-L1-ball*[*of fst z snd z fst a snd a ε*] *assms* **by** *auto*

lemma *prod-dist-L1-ball1′*: *Prod-metric-L1.mball* (*a1*,*a2*) (*min e1 e2*) ⊆ *M1.mball a1 e1* × *M2.mball a2 e2*
  using *prod-dist-L1-geq order.strict-trans1* **by** *auto blast+*

lemma *prod-dist-L1-ball1*:
  assumes *b1* ∈ *M1.mball a1 e1 b2* ∈ *M2.mball a2 e2*
  shows ∃ *e12*>*0*. *Prod-metric-L1.mball* (*b1*,*b2*) *e12* ⊆ *M1.mball a1 e1* × *M2.mball a2 e2*
proof −
  obtain *ea1 ea2* **where** *ea*: *ea1* > *0 ea2* > *0 M1.mball b1 ea1* ⊆ *M1.mball a1 e1 M2.mball b2 ea2* ⊆ *M2.mball a2 e2*
    using *assms* **by** (*meson M1.openin-mball M1.openin-mtopology M2.openin-mball M2.openin-mtopology*)
  thus *?thesis*
    using *prod-dist-L1-ball1′*[*of b1 b2 ea1 ea2*] **by**(*auto intro*!: *exI*[**where** *x*=*min ea1 ea2*])
qed

lemma *prod-dist-L1-ball2'*:
M1.mball a1 e1 × *M2.mball a2 e2* ⊆ *Prod-metric-L1.mball* (*a1*,*a2*) (*e1* + *e2*)
by *auto* (*auto simp*: *prod-dist-L1-def*)

lemma *prod-dist-L1-ball2*:
assumes (*b1*,*b2*) ∈ *Prod-metric-L1.mball* (*a1*,*a2*) *e12*
shows ∃ *e1*>*0*. ∃ *e2*>*0*. *M1.mball b1 e1* × *M2.mball b2 e2* ⊆ *Prod-metric-L1.mball* (*a1*,*a2*) *e12*
proof −
obtain *e12'* **where** *e12'* > *0* *Prod-metric-L1.mball* (*b1*,*b2*) *e12'* ⊆ *Prod-metric-L1.mball* (*a1*,*a2*) *e12*
by (*metis assms Prod-metric-L1.openin-mball Prod-metric-L1.openin-mtopology*)
thus *?thesis*
using *prod-dist-L1-ball2'*[*of b1 e12'* / *2 b2 e12'* / *2*] **by**(*auto intro*!: *exI*[**where** *x*=*e12'* / *2*])
qed

lemma *prod-dist-L1-mtopology*:
Prod-metric-L1.mtopology = *prod-topology M1.mtopology M2.mtopology*
proof −
have *Prod-metric-L1.mtopology* = *topology-generated-by* { *M1.mball a1 e1* × *M2.mball a2 e2* | *a1 a2 e1 e2*. *a1* ∈ *M1* ∧ *a2* ∈ *M2* ∧ *e1* > *0* ∧ *e2* > *0*}
unfolding *base-is-subbase*[*OF Prod-metric-L1.mtopology-base-in-balls*,*simplified subbase-in-def*]
proof(*rule topology-generated-by-eq*)
fix *U*
assume *U* ∈ {*M1.mball a1 e1* × *M2.mball a2 e2* | *a1 a2 e1 e2*. *a1* ∈ *M1* ∧ *a2* ∈ *M2* ∧ *0* < *e1* ∧ *0* < *e2*}
then obtain *a1 e1 a2 e2* **where** *hae*:
U = *M1.mball a1 e1* × *M2.mball a2 e2 a1* ∈ *M1 a2* ∈ *M2 0* < *e1 0* < *e2*
by *auto*
show *openin* (*topology-generated-by* {*Prod-metric-L1.mball a ε* |*a ε*. *a* ∈ *M1* × *M2* ∧ *0* < *ε*}) *U*
unfolding *base-is-subbase*[*OF Prod-metric-L1.mtopology-base-in-balls*,*simplified subbase-in-def*,*symmetric*] *Prod-metric-L1.openin-mtopology hae*(*1*)
using *prod-dist-L1-ball1*[*of - a1 e1 - a2 e2*] **by** *fastforce*
next
fix *U*
assume *U* ∈ {*Prod-metric-L1.mball a ε* |*a ε*. *a* ∈ *M1* × *M2* ∧ *0* < *ε*}
then obtain *a1 a2 ε* **where** *hae*:
U = *Prod-metric-L1.mball* (*a1*,*a2*) *ε a1* ∈ *M1 a2* ∈ *M2 0* < *ε*
by *auto*
show *openin* (*topology-generated-by* {*M1.mball a1 e1* × *M2.mball a2 e2* | *a1 a2 e1 e2*. *a1* ∈ *M1* ∧ *a2* ∈ *M2* ∧ *0* < *e1* ∧ *0* < *e2*}) *U*
unfolding *openin-subopen*[*of - Prod-metric-L1.mball* (*a1*,*a2*) *ε*] *hae*(*1*)
proof *safe*
fix *b1 b2*
assume *h*:(*b1*, *b2*) ∈ *Prod-metric-L1.mball* (*a1*, *a2*) *ε*
from *prod-dist-L1-ball2*[*OF this*] **obtain** *e1 e2* **where** *e1* > *0 e2* > *0 M1.mball*

b1 e1 × *M2.mball b2 e2* ⊆ *Prod-metric-L1.mball* (*a1*, *a2*) *ε*
        by *metis*
     with *h* **show** ∃ *T*. *openin* (*topology-generated-by* {*M1.mball a1 e1* × *M2.mball a2 e2* | *a1 a2 e1 e2*. *a1* ∈ *M1* ∧ *a2* ∈ *M2* ∧ *0* < *e1* ∧ *0* < *e2*}) *T* ∧ (*b1*, *b2*) ∈ *T* ∧ *T* ⊆ *Prod-metric-L1.mball* (*a1*, *a2*) *ε*
        unfolding *openin-topology-generated-by-iff*
        by(*auto intro*!: *generate-topology-on.Basis exI*[**where** *x*=*M1.mball b1 e1* × *M2.mball b2 e2*])
    qed
  qed
  also have ... = *prod-topology M1.mtopology M2.mtopology*
  proof −
    have {*M1.mball a ε* × *M2.mball a′ ε′* |*a a′ ε ε′*. *a* ∈ *M1* ∧ *a′* ∈ *M2* ∧ *0* < *ε* ∧ *0* < *ε′*} = {*U* × *V* |*U V*. *U* ∈ {*M1.mball a ε* |*a ε*. *a* ∈ *M1* ∧ *0* < *ε*} ∧ *V* ∈ {*M2.mball a ε* |*a ε*. *a* ∈ *M2* ∧ *0* < *ε*}}
      by *blast*
    thus *?thesis*
     unfolding *base-is-subbase*[*OF M1.mtopology-base-in-balls*,*simplified subbase-in-def*] *base-is-subbase*[*OF M2.mtopology-base-in-balls*,*simplified subbase-in-def*]
      by(*simp only*: *prod-topology-generated-by*[*symmetric*])
  qed
  finally show *?thesis* **.**
qed

lemma *prod-dist-L1-limitin-iff*: *limitin Prod-metric-L1.mtopology zn z sequentially* ⟷ *limitin M1.mtopology* (*λn*. *fst* (*zn n*)) (*fst z*) *sequentially* ∧ *limitin M2.mtopology* (*λn*. *snd* (*zn n*)) (*snd z*) *sequentially*
proof *safe*
  assume *h*:*limitin Prod-metric-L1.mtopology zn z sequentially*
  show *limitin M1.mtopology* (*λn*. *fst* (*zn n*)) (*fst z*) *sequentially*
       limitin M2.mtopology (*λn*. *snd* (*zn n*)) (*snd z*) *sequentially*
    unfolding *M1.limit-metric-sequentially M2.limit-metric-sequentially*
  proof *safe*
    fix *e* :: *real*
    assume *e*: *0* < *e*
     with *h* **obtain** *N* **where** *N*:⋀*n*. *n* ≥ *N* ⟹ *zn n* ∈ *M1* × *M2* ⋀*n*. *n* ≥ *N* ⟹ *prod-dist-L1 d1 d2* (*zn n*) *z* < *e*
      by(*simp only*: *Prod-metric-L1.limit-metric-sequentially*) *metis*
    show ∃ *N*. ∀ *n*≥*N*. *fst* (*zn n*) ∈ *M1* ∧ *d1* (*fst* (*zn n*)) (*fst z*) < *e*
        ∃ *N*. ∀ *n*≥*N*. *snd* (*zn n*) ∈ *M2* ∧ *d2* (*snd* (*zn n*)) (*snd z*) < *e*
    proof(*safe intro*!: *exI*[**where** *x*=*N*])
      fix *n*
      assume *N* ≤ *n*
      from *N*[*OF this*]
      show *d1* (*fst* (*zn n*)) (*fst z*) < *e* *d2* (*snd* (*zn n*)) (*snd z*) < *e*
        using *order.strict-trans1*[*OF prod-dist-L1-geq*(*1*)[*of fst* (*zn n*) *fst z snd* (*zn n*) *snd z*]] *order.strict-trans1*[*OF prod-dist-L1-geq*(*2*)[*of snd* (*zn n*) *snd z fst* (*zn n*) *fst z*]]
        by *auto*

qed(*use N*(*1*)[*simplified mem-Times-iff*] **in** *auto*)
qed(*use h Prod-metric-L1.limit-metric-sequentially* **in** *auto*)
next
assume *h*:*limitin M1.mtopology* (λ*n. fst* (*zn n*)) (*fst z*) *sequentially*
limitin M2.mtopology (λ*n. snd* (*zn n*)) (*snd z*) *sequentially*
show *limitin Prod-metric-L1.mtopology zn z sequentially*
unfolding *Prod-metric-L1.limit-metric-sequentially*
proof *safe*
fix *e* :: *real*
assume *e*: $0 < e$
with *h* **obtain** *N1 N2* **where** *N*: $\bigwedge n.\ n \geq N1 \Longrightarrow fst\ (zn\ n) \in M1$ $\bigwedge n.\ n \geq N1 \Longrightarrow d1\ (fst\ (zn\ n))\ (fst\ z) < e\ /\ 2$
$\bigwedge n.\ n \geq N2 \Longrightarrow snd\ (zn\ n) \in M2$ $\bigwedge n.\ n \geq N2 \Longrightarrow d2\ (snd\ (zn\ n))\ (snd\ z) < e\ /\ 2$
unfolding *M1.limit-metric-sequentially M2.limit-metric-sequentially*
using *half-gt-zero* **by** *metis*
thus $\exists N.\ \forall n \geq N.\ zn\ n \in M1 \times M2 \wedge prod\text{-}dist\text{-}L1\ d1\ d2\ (zn\ n)\ z < e$
by(*fastforce intro*!: *exI*[**where** *x*=*max N1 N2*] *simp*: *prod-dist-L1-def split-beta′ mem-Times-iff*)
qed(*auto simp*: *mem-Times-iff h*[*simplified M1.limit-metric-sequentially M2.limit-metric-sequentially*])
qed

lemma *prod-dist-L1-MCauchy-iff*: *Prod-metric-L1.MCauchy zn* $\longleftrightarrow$ *M1.MCauchy* (λ*n. fst* (*zn n*)) $\wedge$ *M2.MCauchy* (λ*n. snd* (*zn n*))
proof *safe*
assume *h*:*Prod-metric-L1.MCauchy zn*
show *M1.MCauchy* (λ*n. fst* (*zn n*)) *M2.MCauchy* (λ*n. snd* (*zn n*))
unfolding *M1.MCauchy-def M2.MCauchy-def*
proof *safe*
fix *e* :: *real*
assume $0 < e$
with *h* **obtain** *N* **where** *N*:$\bigwedge n\ m.\ N \leq n \Longrightarrow N \leq m \Longrightarrow prod\text{-}dist\text{-}L1\ d1\ d2\ (zn\ n)\ (zn\ m) < e$
by(*fastforce simp*: *Prod-metric-L1.MCauchy-def*)
show $\exists N.\ \forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d1\ (fst\ (zn\ n))\ (fst\ (zn\ n')) < e$
$\exists N.\ \forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d2\ (snd\ (zn\ n))\ (snd\ (zn\ n')) < e$
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n m*
assume $n \geq N$ $m \geq N$
from *N*[*OF this*]
show $d1\ (fst\ (zn\ n))\ (fst\ (zn\ m)) < e$ $d2\ (snd\ (zn\ n))\ (snd\ (zn\ m)) < e$
using *order.strict-trans1*[*OF prod-dist-L1-geq*(*1*)[*of fst* (*zn n*) *fst* (*zn m*) *snd* (*zn n*) *snd* (*zn m*)]] *order.strict-trans1*[*OF prod-dist-L1-geq*(*2*)[*of snd* (*zn n*) *snd* (*zn m*) *fst* (*zn n*) *fst* (*zn m*)]]
by *auto*
qed
next
have $\bigwedge n.\ zn\ n \in M1 \times M2$
using *h* **by**(*auto simp*: *Prod-metric-L1.MCauchy-def*)

thus *fst* (*zn n*) ∈ *M1 snd* (*zn n*) ∈ *M2* **for** *n*
by (*auto simp*: *mem-Times-iff*)
qed
next
assume *h*:*M1.MCauchy* (λ*n*. *fst* (*zn n*)) *M2.MCauchy* (λ*n*. *snd* (*zn n*))
show *Prod-metric-L1.MCauchy zn*
unfolding *Prod-metric-L1.MCauchy-def*
proof *safe*
fix *e* :: *real*
assume *0* < *e*
with *h* **obtain** *N1 N2* **where** ⋀*n m*. *n* ≥ *N1* ⟹ *m* ≥ *N1* ⟹ *d1* (*fst* (*zn n*)) (*fst* (*zn m*)) < *e* / *2*
⋀*n m*. *n* ≥ *N2* ⟹ *m* ≥ *N2* ⟹ *d2* (*snd* (*zn n*)) (*snd* (*zn m*)) < *e* / *2*
unfolding *M1.MCauchy-def M2.MCauchy-def* **using** *half-gt-zero* **by** *metis*
thus ∃ *N*. ∀ *n n'*. *N* ≤ *n* ⟶ *N* ≤ *n'* ⟶ *prod-dist-L1 d1 d2* (*zn n*) (*zn n'*) < *e*
by(*fastforce intro*!: *exI*[**where** *x*=*max N1 N2*] *simp*: *prod-dist-L1-def split-beta'*)
next
fix *x y n*
assume *1*:(*x*,*y*) = *zn n*
have *fst* (*zn n*) ∈ *M1 snd* (*zn n*) ∈ *M2*
using *h*[*simplified M1.MCauchy-def M2.MCauchy-def*] **by** *auto*
thus *x* ∈ *M1 y* ∈ *M2*
by(*simp-all add*: *1*[*symmetric*])
qed
qed

end

1.3.7 Sum Metric Spaces

locale *Sum-metric* =
fixes *I* :: *'i set*
and *Mi* :: *'i* ⇒ *'a set*
and *di* :: *'i* ⇒ *'a* ⇒ *'a* ⇒ *real*
assumes *Mi-disj*: *disjoint-family-on Mi I*
and *d-nonneg*: ⋀*i x y*. *0* ≤ *di i x y*
and *d-bounded*: ⋀*i x y*. *di i x y* < *1*
and *Md-metric*: ⋀*i*. *i* ∈ *I* ⟹ *Metric-space* (*Mi i*) (*di i*)
begin

abbreviation *M* ≡ ⋃*i*∈*I*. *Mi i*

lemma *Mi-inj-on*:
assumes *i* ∈ *I j* ∈ *I a* ∈ *Mi i a* ∈ *Mi j*
shows *i* = *j*
using *Mi-disj assms* **by**(*auto simp*: *disjoint-family-on-def*)

definition *sum-dist* :: [*'a*, *'a*] ⇒ *real* **where**
sum-dist x y ≡ (*if x* ∈ *M* ∧ *y* ∈ *M then* (*if* ∃ *i*∈*I*. *x* ∈ *Mi i* ∧ *y* ∈ *Mi i then di*

(*THE i. i ∈ I ∧ x ∈ Mi i ∧ y ∈ Mi i*) *x y else 1*) *else 0*)

lemma *sum-dist-simps*:
  shows ⋀*i*. ⟦*i ∈ I*; *x ∈ Mi i*; *y ∈ Mi i*⟧ ⟹ *sum-dist x y = di i x y*
    and ⋀*i j*. ⟦*i ∈ I*; *j ∈ I*; *i ≠ j*; *x ∈ Mi i*; *y ∈ Mi j*⟧ ⟹ *sum-dist x y = 1*
    and ⋀*i*. ⟦*i ∈ I*; *y ∈ M*; *x ∈ Mi i*; *y ∉ Mi i*⟧ ⟹ *sum-dist x y = 1*
    and ⋀*i*. ⟦*i ∈ I*; *x ∈ M*; *y ∈ Mi i*; *x ∉ Mi i*⟧ ⟹ *sum-dist x y = 1*
    and *x ∉ M* ⟹ *sum-dist x y = 0 y ∉ M* ⟹ *sum-dist x y = 0*
proof −
  { fix *i*
    assume *h*:*i ∈ I x ∈ Mi i y ∈ Mi i*
    then have *sum-dist x y = di* (*THE i. i ∈ I ∧ x ∈ Mi i ∧ y ∈ Mi i*) *x y*
      by(*auto simp*: *sum-dist-def*)
    also have ... = *di i x y*
    proof −
      have (*THE i. i ∈ I ∧ x ∈ Mi i ∧ y ∈ Mi i*) = *i*
        using *Mi-disj h* **by**(*auto intro*!: *the1-equality simp*: *disjoint-family-on-def*)
      thus *?thesis* **by** *simp*
    qed
    finally show *sum-dist x y = di i x y* **. }**
  show ⋀*i j*. ⟦*i ∈ I*; *j ∈ I*; *i ≠ j*; *x ∈ Mi i*; *y ∈ Mi j*⟧ ⟹ *sum-dist x y = 1*
      ⋀*i*. ⟦*i ∈ I*; *y ∈ M*; *x ∈ Mi i*; *y ∉ Mi i*⟧ ⟹ *sum-dist x y = 1* ⋀*i*. ⟦*i ∈ I*; *x*
∈ *M*; *y ∈ Mi i*; *x ∉ Mi i*⟧ ⟹ *sum-dist x y = 1*
      x ∉ M ⟹ *sum-dist x y = 0 y ∉ M* ⟹ *sum-dist x y = 0*
   using *Mi-disj* **by**(*auto simp*: *sum-dist-def disjoint-family-on-def dest*: *Mi-inj-on*)
qed

lemma *sum-dist-if-less1*:
  assumes *i ∈ I x ∈ Mi i y ∈ M sum-dist x y < 1*
  shows *y ∈ Mi i*
  using *assms sum-dist-simps*(*3*) **by** *fastforce*

lemma *inM-cases*:
  assumes *x ∈ M y ∈ M*
    and ⋀*i*. ⟦*i ∈ I*; *x ∈ Mi i*; *y ∈ Mi i*⟧ ⟹ *P x y*
    and ⋀*i j*. ⟦*i ∈ I*; *j ∈ I*; *i ≠ j*; *x ∈ Mi i*; *y ∈ Mi j*; *x ≠ y*⟧ ⟹ *P x y*
  shows *P x y* **using** *assms* **by** *auto*

sublocale *Sum-metric*: *Metric-space M sum-dist*
proof
  fix *x y*
  assume *x ∈ M y ∈ M*
  then show *sum-dist x y = 0 ⟷ x = y*
   by(*rule inM-cases*, *insert Md-metric*) (*auto simp*: *sum-dist-simps Metric-space-def*)
next
  { fix *i x y*
    assume *h*: *i ∈ I x ∈ Mi i y ∈ Mi i*
    then have *sum-dist x y = di i x y*
          sum-dist y x = di i x y

```
      using Md-metric by(auto simp: sum-dist-simps Metric-space-def) }
  thus ⋀x y. sum-dist x y = sum-dist y x
    by (metis (no-types, lifting) sum-dist-def)
next
  show 1:⋀x y. 0 ≤ sum-dist x y
    using d-nonneg by(simp add: sum-dist-def)
  fix x y z
  assume h: x ∈ M y ∈ M z ∈ M
  show sum-dist x z ≤ sum-dist x y + sum-dist y z (is ?lhs ≤ ?rhs)
  proof(rule inM-cases[OF h(1,3)])
    fix i
    assume h':i ∈ I x ∈ Mi i z ∈ Mi i
    consider y ∈ Mi i | y ∉ Mi i by auto
    thus ?lhs ≤ ?rhs
    proof cases
      case 1
      with h' Md-metric [OF h'(1)] show ?thesis
        by(simp add: sum-dist-simps Metric-space-def)
    next
      case 2
      with h' h(2) d-bounded[of i x z] 1[of y z]
      show ?thesis
        by(auto simp: sum-dist-simps)
    qed
  next
    fix i j
    assume h': i ∈ I j ∈ I i ≠ j x ∈ Mi i z ∈ Mi j
    consider y ∉ Mi i | y ∉ Mi j
      using h' h(2) Mi-disj by(auto simp: disjoint-family-on-def)
    thus ?lhs ≤ ?rhs
      by (cases, insert 1[of x y] 1[of y z] h' h(2)) (auto simp: sum-dist-simps)
  qed
qed

lemma sum-dist-le1: sum-dist x y ≤ 1
  using d-bounded[of - x y] by(auto simp: sum-dist-def less-eq-real-def)


lemma sum-dist-ball-eq-ball:
  assumes i ∈ I e ≤ 1 x ∈ Mi i
  shows Metric-space.mball (Mi i) (di i) x e = Sum-metric.mball x e
proof -
  interpret m: Metric-space Mi i di i
    by(simp add: assms Md-metric)
  show ?thesis
     using assms sum-dist-simps(1)[OF assms(1) assms(3)] sum-dist-if-less1[OF
assms(1,3)]
    by(fastforce simp: Sum-metric.mball-def)
qed
```

lemma *ball-le-sum-dist-ball*:
  assumes $i \in I$
  shows *Metric-space.mball* (*Mi i*) (*di i*) x $e \subseteq$ *Sum-metric.mball* x e
proof −
  interpret *m*: *Metric-space Mi i di i*
    by(*simp add*: *assms Md-metric*)
  show *?thesis*
    using *assms* **by**(*auto simp*: *sum-dist-simps*)
qed

lemma *openin-mtopology-iff*:
openin Sum-metric.mtopology U $\longleftrightarrow U \subseteq M \wedge (\forall i \in I.$ *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) ($U \cap$ *Mi i*))
proof *safe*
  fix *i*
  assume *h*:*openin Sum-metric.mtopology U* $i \in I$
  then interpret *m*: *Metric-space Mi i di i*
    using *Md-metric* **by** *simp*
  show *openin m.mtopology* ($U \cap$ *Mi i*)
    unfolding *m.openin-mtopology*
  proof *safe*
    fix *x*
    assume *x*:$x \in U$ $x \in$ *Mi i*
    with *h* **obtain** *e* **where** *e*: $e > 0$ *Sum-metric.mball* x $e \subseteq U$
      by(*auto simp*: *Sum-metric.openin-mtopology*)
    show $\exists \varepsilon > 0.$ *m.mball* x $\varepsilon \subseteq U \cap$ *Mi i*
    proof(*safe intro*!: *exI*[**where** $x=e$])
      fix *y*
      assume $y \in$ *m.mball* x e
      with *h*(*2*) **have** $y \in$ *Sum-metric.mball* x e
        by(*auto simp*:*sum-dist-simps*)
      with *e* **show** $y \in U$ **by** *blast*
    qed(*use e* **in** *auto*)
  qed
next
  show $\bigwedge x.$ *openin Sum-metric.mtopology U* $\Longrightarrow x \in U \Longrightarrow x \in M$
    by(*auto simp*: *Sum-metric.openin-mtopology*)
next
  assume *h*: $U \subseteq M$ $\forall i \in I.$ *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) ($U \cap$ *Mi i*)
  show *openin Sum-metric.mtopology U*
    unfolding *Sum-metric.openin-mtopology*
  proof *safe*
    fix *x*
    assume *x*: $x \in U$
    then obtain *i* **where** *i*: $i \in I$ $x \in$ *Mi i*
      using *h*(*1*) **by** *auto*
    then interpret *m*: *Metric-space Mi i di i*

```
      using Md-metric by simp
    obtain e where e: e > 0 m.mball x e ⊆ U ∩ Mi i
      using i h(2) by (meson IntI m.openin-mtopology x)
    show ∃ε>0. Sum-metric.mball x ε ⊆ U
    proof(safe intro!: exI[where x=min e 1])
      fix y
      assume y:y ∈ Sum-metric.mball x (min e 1)
      then show y ∈ U
        using sum-dist-ball-eq-ball[OF i(1) - i(2),of min e 1] e by fastforce
    qed(simp add: e)
  qed(use h(1) in auto)
qed

corollary openin-mtopology-Mi:
  assumes i ∈ I
  shows openin Sum-metric.mtopology (Mi i)
  unfolding openin-mtopology-iff
proof safe
  fix j
  assume j:j ∈ I
  then interpret m: Metric-space Mi j di j
    by(simp add: Md-metric)
  show openin m.mtopology (Mi i ∩ Mi j)
    by (cases i = j, insert assms Mi-disj j) (auto simp: disjoint-family-on-def)
qed(use assms in auto)

corollary subtopology-mtopology-Mi:
  assumes i ∈ I
  shows subtopology Sum-metric.mtopology (Mi i) = Metric-space.mtopology (Mi
i) (di i)
proof −
  interpret Mi: Metric-space Mi i di i
    by (simp add: Md-metric assms)
  show ?thesis
    unfolding topology-eq openin-subtopology
  proof safe
    fix T
    assume openin Sum-metric.mtopology T
    thus openin Mi.mtopology (T ∩ Mi i)
      using assms by(auto simp: openin-mtopology-iff)
  next
    fix S
    assume h:openin Mi.mtopology S
    show ∃ T. openin Sum-metric.mtopology T ∧ S = T ∩ Mi i
    proof(safe intro!: exI[where x=S])
      show openin Sum-metric.mtopology S
        unfolding openin-mtopology-iff
      proof safe
        fix j
```

```
        assume j:j ∈ I
        then interpret Mj: Metric-space Mi j di j
          using Md-metric by auto
        have i ≠ j ⟹ S ∩ Mi j = {}
          using openin-subset[OF h] Mi-disj j assms
          by(auto simp: disjoint-family-on-def)
        thus openin Mj.mtopology (S ∩ Mi j)
          by(cases i = j) (use openin-subset[OF h] h in auto)
      qed(use openin-subset[OF h] assms in auto)
    qed(use openin-subset[OF h] assms in auto)
  qed
qed

lemma limitin-Mi-limitin-M:
  assumes i ∈ I limitin (Metric-space.mtopology (Mi i) (di i)) xn x sequentially
  shows limitin Sum-metric.mtopology xn x sequentially
proof −
  interpret m: Metric-space Mi i di i
    by(simp add: assms Md-metric)
  {
    fix e :: real
    assume e > 0
    then obtain N where ⋀n. n ≥ N ⟹ xn n ∈ m.mball x e
      using assms(2) m.commute m.limit-metric-sequentially by fastforce
    hence ∃N. ∀n≥N. xn n ∈ Sum-metric.mball x e
      using ball-le-sum-dist-ball[OF assms(1),of x e]
      by(fastforce intro!: exI[where x=N]) }
  thus ?thesis
   by (metis Sum-metric.commute Sum-metric.in-mball Sum-metric.limit-metric-sequentially
UN-I m.limitin-mspace assms)
qed

lemma limitin-M-limitin-Mi:
  assumes limitin Sum-metric.mtopology xn x sequentially
  shows ∃i∈I. limitin (Metric-space.mtopology (Mi i) (di i)) xn x sequentially
proof −
  obtain i where i: i ∈ I x ∈ Mi i
    using assms by (meson Sum-metric.limitin-mspace UN-E)
  then interpret m: Metric-space Mi i di i
    by(simp add: Md-metric)
  obtain N where N: ⋀n. n ≥ N ⟹ sum-dist (xn n) x < 1 ⋀n. n ≥ N ⟹
(xn n) ∈ M
   using assms by (metis d-bounded i(2) m.mdist-zero Sum-metric.limit-metric-sequentially)
  hence xni: n ≥ N ⟹ xn n ∈ Mi i for n
   using assms by(auto intro!: sum-dist-if-less1[OF i,of xn n] simp: Sum-metric.commute)
  show ?thesis
  proof(safe intro!: bexI[where x=i] i)
    show limitin m.mtopology xn x sequentially
      unfolding m.limit-metric-sequentially
```

```
    proof safe
      fix e :: real
      assume e: 0 < e
      then obtain N' where N': ⋀n. n ≥ N' ⟹ sum-dist (xn n) x < e
        using assms by (meson Sum-metric.limit-metric-sequentially)
      hence n ≥ max N N' ⟹ di i (xn n) x < e for n
        using sum-dist-simps(1)[OF i(1) xni[of n] i(2),symmetric] by auto
      thus ∃N. ∀n≥N. xn n ∈ Mi i ∧ di i (xn n) x < e
        using xni by(auto intro!: exI[where x=max N N'])
    qed fact
  qed
qed

lemma MCauchy-Mi-MCauchy-M:
  assumes i ∈ I Metric-space.MCauchy (Mi i) (di i) xn
  shows Sum-metric.MCauchy xn
proof −
  interpret m: Metric-space Mi i di i
    by(simp add: assms Md-metric)
  have [simp]:sum-dist (xn n) (xn m) = di i (xn n) (xn m) for n m
    using assms sum-dist-simps(1)[OF assms(1),of xn n xn m]
    by(auto simp: m.MCauchy-def)
  show ?thesis
    using assms by(auto simp: m.MCauchy-def Sum-metric.MCauchy-def)
qed

lemma MCauchy-M-MCauchy-Mi:
  assumes Sum-metric.MCauchy xn
  shows ∃m. ∃i∈I. Metric-space.MCauchy (Mi i) (di i) (λn. xn (n + m))
proof −
  obtain N where N: ⋀n m. n ≥ N ⟹ m ≥ N ⟹ sum-dist (xn n) (xn m) < 1
    using assms by(fastforce simp: Sum-metric.MCauchy-def)
  obtain i where i: i ∈ I xn N ∈ Mi i
    by (metis assms Sum-metric.MCauchy-def UNIV-I UN-E image-eqI subsetD)
  then have xn:⋀n. n ≥ N ⟹ xn n ∈ Mi i
      by (metis N Sum-metric.MCauchy-def assms dual-order.refl range-subsetD
sum-dist-if-less1)
  interpret m: Metric-space Mi i di i
    using i Md-metric by auto
  show ?thesis
  proof(safe intro!: exI[where x=N] bexI[where x=i])
    show m.MCauchy (λn. xn (n + N))
      unfolding m.MCauchy-def
    proof safe
      show 1: ⋀n. xn (n + N) ∈ Mi i
        by(auto intro!: xn)
      fix e :: real
      assume 0 < e
      then obtain N' where N': ⋀n m. n ≥ N' ⟹ m ≥ N' ⟹ sum-dist (xn n)
```

```
(xn m) < e
        using Sum-metric.MCauchy-def assms by blast
      thus ∃N'. ∀n n'. N' ≤ n ⟶ N' ≤ n' ⟶ di i (xn (n + N)) (xn (n' + N))
< e
          by(auto intro!: exI[where x=N'] simp: sum-dist-simps(1)[OF i(1) xn
xn,symmetric])
    qed
  qed fact
qed

lemma separable-Mi-separable-M:
  assumes countable I ⋀i. i ∈ I ⟹ separable-space (Metric-space.mtopology (Mi
i) (di i))
  shows separable-space Sum-metric.mtopology
proof -
  obtain Ui where Ui: ⋀i. i ∈ I ⟹ countable (Ui i) ⋀i. i ∈ I ⟹ dense-in
(Metric-space.mtopology (Mi i) (di i)) (Ui i)
    using assms by(simp only: separable-space-def2) metis
  define U where U ≡ ⋃i∈I. Ui i
  show separable-space Sum-metric.mtopology
    unfolding separable-space-def2
  proof(safe intro!: exI[where x=U])
    show countable U
      using Ui(1) assms by(auto simp: U-def)
  next
    show Sum-metric.mdense U
      unfolding Sum-metric.mdense-def U-def
    proof safe
      fix i x
      assume i ∈ I x ∈ Ui i
      then show x ∈ M
          using Ui(2) by(auto intro!: bexI[where x=i] simp: Md-metric Met-
ric-space.mdense-def2)
    next
      fix i x e
      assume h:i ∈ I x ∈ Mi i (0 :: real) < e Sum-metric.mball x e ∩ ⋃ (Ui ` I)
= {}
      then interpret m: Metric-space Mi i di i
        by(simp add: Md-metric)
      have m.mball x e ∩ Ui i ≠ {}
        using Ui(2)[OF h(1)] h by(auto simp: m.mdense-def)
      hence m.mball x e ∩ ⋃ (Ui ` I) ≠ {}
        using h(1) by blast
      thus False
        using ball-le-sum-dist-ball[OF ‹i ∈ I›,of x e] h(4) by blast
    qed
  qed
qed
```

lemma *separable-M-separable-Mi*:
assumes *separable-space Sum-metric.mtopology* $\bigwedge i.\ i \in I$
shows *separable-space* (*Metric-space.mtopology* (*Mi i*) (*di i*))
using *subtopology-mtopology-Mi*[*OF assms*(*2*)] *separable-space-open-subset*[*OF assms*(*1*) *openin-mtopology-Mi*[*OF assms*(*2*)]]
by *simp*

lemma *mcomplete-Mi-mcomplete-M*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *Metric-space.mcomplete* (*Mi i*) (*di i*)
shows *Sum-metric.mcomplete*
unfolding *Sum-metric.mcomplete-def*
proof *safe*
fix *xn*
assume *Sum-metric.MCauchy xn*
from *MCauchy-M-MCauchy-Mi*[*OF this*] **obtain** *m i* **where** *mi*:
$i \in I$ *Metric-space.MCauchy* (*Mi i*) (*di i*) ($\lambda n.\ xn\ (n + m)$)
by *metis*
then interpret *m*: *Metric-space Mi i di i*
by(*simp add*: *Md-metric*)
from *assms*[*OF mi*(*1*)] *mi*(*2*) **obtain** *x* **where** *x*: *limitin m.mtopology* ($\lambda n.\ xn\ (n + m)$) *x sequentially*
by(*auto simp*: *m.mcomplete-def*)
from *limitin-Mi-limitin-M*[*OF mi*(*1*) *limitin-sequentially-offset-rev*[*OF this*]]
show $\exists x.$ *limitin Sum-metric.mtopology xn x sequentially*
by *auto*
qed

lemma *mcomplete-M-mcomplete-Mi*:
assumes *Sum-metric.mcomplete* $i \in I$
shows *Metric-space.mcomplete* (*Mi i*) (*di i*)
proof −
interpret *Mi*: *Metric-space Mi i di i*
using *assms*(*2*) *Md-metric* **by** *fastforce*
show *?thesis*
unfolding *Mi.mcomplete-def*
proof *safe*
fix *xn*
assume *xn*:*Mi.MCauchy xn*
from *MCauchy-Mi-MCauchy-M*[*OF assms*(*2*) *this*] *assms*(*1*) **obtain** *x* **where** *limitin Sum-metric.mtopology xn x sequentially*
by(*auto simp*: *Sum-metric.mcomplete-def*)
from *limitin-M-limitin-Mi*[*OF this*] **obtain** *j* **where** *j*:$j \in I$ *limitin* (*Metric-space.mtopology* (*Mi j*) (*di j*)) *xn x sequentially*
by *auto*
have $j = i$
proof −
obtain *N* **where** $\bigwedge n.\ n \geq N \Longrightarrow xn\ n \in Mi\ j$
by (*metis Md-metric Metric-space.limitin-metric-dist-null eventually-sequentially j*)

```
      hence xn N ∈ Mi i ∩ Mi j
        using xn by(auto simp: Mi.MCauchy-def)
      with Mi-disj j(1) assms(2) show ?thesis
        by(fastforce simp: disjoint-family-on-def)
    qed
    thus ∃x. limitin Mi.mtopology xn x sequentially
      using j(2) by(auto intro!: exI[where x=x])
  qed
qed

end

lemma sum-metricI:
  fixes Si
  assumes disjoint-family-on Si I
      and ⋀i x y. i ∉ I ⟹ 0 ≤ di i x y
      and ⋀i x y. di i x y < 1
      and ⋀i. i ∈ I ⟹ Metric-space (Si i) (di i)
    shows Sum-metric I Si di
  using assms by (metis Metric-space.nonneg Sum-metric-def)



end
```

1.3.8 Product Metric Spaces

```
theory Set-Based-Metric-Product
  imports Set-Based-Metric-Space
begin

lemma nsum-of-r':
  fixes r :: real
  assumes r:0 < r r < 1
  shows (∑n. r^(n + k) ∗ K) = r^k / (1 − r) ∗ K
 (is ?lhs = -)
proof −
  have ?lhs = (∑n. r^n ∗ K) − (∑n∈{..<k}. r^n ∗ K)
    using r by(auto intro!: suminf-minus-initial-segment summable-mult2)
  also have ... = 1 / (1 − r) ∗ K − (1 − r^k) / (1 − r) ∗ K
  proof −
    have (∑n∈{..<k}. r^n ∗ K) = (1 − r^k) / (1 − r) ∗ K
     using one-diff-power-eq[of r k] r scale-sum-left[of λn. r^n {..<k} K,symmetric]
      by auto
    thus ?thesis
        using r by(auto simp add: suminf-geometric[of r] suminf-mult2[where
c=K,symmetric])
  qed
  finally show ?thesis
```

using *r* **by** (*simp add*: *diff-divide-distrib left-diff-distrib*)
qed

lemma *nsum-of-r-leq*:
fixes *r* :: *real* **and** *a* :: *nat* ⇒ *real*
assumes *r*:$0 < r$ $r < 1$
and *a*:$\bigwedge n.\ 0 \leq a\ n$ $\bigwedge n.\ a\ n \leq K$
shows $0 \leq (\sum n.\ r\hat{}(n + k) * a\ (n + l))$ $(\sum n.\ r\hat{}(n + k) * a\ (n + l)) \leq r\hat{}k / (1 - r) * K$
proof −
have [*simp*]: *summable* $(\lambda n.\ r\ \hat{}\ (n + k) * a\ (n + l))$
apply(*rule summable-comparison-test′*[*of* $\lambda n.\ r\hat{}(n{+}k) * K$])
using *r a* **by**(*auto intro*!: *summable-mult2*)
show $0 \leq (\sum n.\ r\hat{}(n + k) * a\ (n + l))$
using *r a* **by**(*auto intro*!: *suminf-nonneg*)
have $(\sum n.\ r\hat{}(n + k) * a\ (n + l)) \leq (\sum n.\ r\hat{}(n + k) * K)$
using *a r* **by**(*auto intro*!: *suminf-le summable-mult2*)
also have ... $= r\hat{}k / (1 - r) * K$
by(*rule nsum-of-r′*[*OF r*])
finally show $(\sum n.\ r\hat{}(n + k) * a\ (n + l)) \leq r\hat{}k / (1 - r) * K$ **.**
qed

lemma *nsum-of-r-le*:
fixes *r* :: *real* **and** *a* :: *nat* ⇒ *real*
assumes *r*:$0 < r$ $r < 1$
and *a*:$\bigwedge n.\ 0 \leq a\ n$ $\bigwedge n.\ a\ n \leq K$ $\exists n' \geq l.\ a\ n' < K$
shows $(\sum n.\ r\hat{}(n + k) * a\ (n + l)) < r\hat{}k / (1 - r) * K$
proof −
obtain n' **where** *hn′*: $a\ (n' + l) < K$
using *a*(*3*) **by** (*metis add.commute le-iff-add*)
define a' **where** $a' = (\lambda n.\ if\ n = n' + l\ then\ K\ else\ a\ n)$
have *a′*: $\bigwedge n.\ 0 \leq a'\ n$ $\bigwedge n.\ a'\ n \leq K$
using *a*(*1*,*2*) *le-trans order.trans*[*OF a*(*1*,*2*)[*of 0*]] **by**(*auto simp*: *a′-def*)
have [*simp*]: *summable* $(\lambda n.\ r\ \hat{}\ (n + k) * a\ (n + l))$
apply(*rule summable-comparison-test′*[*of* $\lambda n.\ r\hat{}(n{+}k) * K$])
using *r a* **by**(*auto intro*!: *summable-mult2*)
have [*simp*]: *summable* $(\lambda n.\ r\hat{}(n + k) * a'\ (n + l))$
apply(*rule summable-comparison-test′*[*of* $\lambda n.\ r\hat{}(n{+}k) * K$])
using *r a′* **by**(*auto intro*!: *summable-mult2*)
have $(\sum n.\ r\hat{}(n + k) * a\ (n + l)) = (\sum n.\ r\hat{}(n + Suc\ n' + k) * a\ (n + Suc\ n' + l)) + (\sum i<Suc\ n'.\ r\hat{}(i + k) * a\ (i + l))$
by(*rule suminf-split-initial-segment*) *simp*
also have ... $= (\sum n.\ r\hat{}(n + Suc\ n' + k) * a\ (n + Suc\ n' + l)) + (\sum i<n'.\ r\hat{}(i + k) * a\ (i + l)) + r\hat{}(n' + k) * a\ (n' + l)$
by *simp*
also have ... $< (\sum n.\ r\hat{}(n + Suc\ n' + k) * a\ (n + Suc\ n' + l)) + (\sum i<n'.\ r\hat{}(i + k) * a\ (i + l)) + r\hat{}(n' + k) * K$
using *r hn′* **by** *auto*
also have ... $= (\sum n.\ r\hat{}(n + Suc\ n' + k) * a'\ (n + Suc\ n' + l)) + (\sum i<Suc$

n′. *r*^(*i* + *k*) ∗ *a′* (*i* + *l*))
  by(*auto simp*: *a′-def*)
 also have ... = (∑ *n*. *r*^(*n* + *k*) ∗ *a′* (*n* + *l*))
  by(*rule suminf-split-initial-segment*[*symmetric*]) *simp*
 also have ... ≤ *r*^*k* / (*1* − *r*) ∗ *K*
  by(*rule nsum-of-r-leq*[*OF r a′*])
 finally show *?thesis* **.**
qed

definition *product-dist′* :: [*real*, *′i set*, *nat* ⇒ *′i*, *′i* ⇒ *′a set*, *′i* ⇒ *′a* ⇒ *′a* ⇒ *real*] ⇒ (*′i* ⇒ *′a*) ⇒ (*′i* ⇒ *′a*) ⇒ *real* **where**
product-dist-def: *product-dist′ r I g Mi di* ≡ (λ*x y*. *if x* ∈ (Π_E *i*∈*I*. *Mi i*) ∧ *y* ∈ (Π_E *i*∈*I*. *Mi i*) *then* (∑ *n*. *if g n* ∈ *I then r*^*n* ∗ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*)) *else* *0*) *else 0*)

$d(x,y) = \sum_{n\in\mathbb{N}} r^n * d_{g_I(i)}(x_{g_I(i)}, y_{g_I(i)})$.

locale *Product-metric* =
 fixes *r* :: *real*
  and *I* :: *′i set*
  and *f* :: *′i* ⇒ *nat*
  and *g* :: *nat* ⇒ *′i*
  and *Mi* :: *′i* ⇒ *′a set*
  and *di* :: *′i* ⇒ *′a* ⇒ *′a* ⇒ *real*
  and *K* :: *real*
 assumes *r*: *0* < *r r* < *1*
   and *I*: *countable I*
   and *gf-comp-id* : ⋀*i*. *i* ∈ *I* ⟹ *g* (*f i*) = *i*
   and *gf-if-finite*: *finite I* ⟹ *bij-betw f I* {..< *card I*}
        finite I ⟹ *bij-betw g* {..< *card I*} *I*
   and *gf-if-infinite*: *infinite I* ⟹ *bij-betw f I UNIV*
        infinite I ⟹ *bij-betw g UNIV I*
        ⋀*n*. *infinite I* ⟹ *f* (*g n*) = *n*
   and *Md-metric*: ⋀*i*. *i* ∈ *I* ⟹ *Metric-space* (*Mi i*) (*di i*)
   and *di-nonneg*: ⋀*i x y*. *0* ≤ *di i x y*
   and *di-bounded*: ⋀*i x y*. *di i x y* ≤ *K*
   and *K-pos* : *0* < *K*

lemma *from-nat-into-to-nat-on-product-metric-pair*:
 assumes *countable I*
 shows ⋀*i*. *i* ∈ *I* ⟹ *from-nat-into I* (*to-nat-on I i*) = *i*
  and *finite I* ⟹ *bij-betw* (*to-nat-on I*) *I* {..< *card I*}
  and *finite I* ⟹ *bij-betw* (*from-nat-into I*) {..< *card I*} *I*
  and *infinite I* ⟹ *bij-betw* (*to-nat-on I*) *I UNIV*
  and *infinite I* ⟹ *bij-betw* (*from-nat-into I*) *UNIV I*
  and ⋀*n*. *infinite I* ⟹ *to-nat-on I* (*from-nat-into I n*) = *n*
 by(*simp-all add*: *assms to-nat-on-finite bij-betw-from-nat-into-finite to-nat-on-infinite bij-betw-from-nat-into*)

lemma *product-metric-pair-finite-nat*:

bij-betw id {..*n*} {..< *card* {..*n*}} *bij-betw id* {..< *card* {..*n*}} {..*n*}
by(*auto simp*: *bij-betw-def*)

lemma *product-metric-pair-finite-nat′*:
bij-betw id {..<*n*} {..< *card* {..<*n*}} *bij-betw id* {..< *card* {..<*n*}} {..<*n*}
by(*auto simp*: *bij-betw-def*)

context *Product-metric*
begin

abbreviation *product-dist* ≡ *product-dist′ r I g Mi di*

lemma *nsum-of-rK*: ($\sum$ *n*. *r*^(*n* + *k*)∗*K*) = *r*^*k* / (*1* − *r*) ∗ *K*
by(*rule nsum-of-r′*[*OF r*])

lemma *i-min*:
assumes *i* ∈ *I g n* = *i*
shows *f i* ≤ *n*
proof(*cases finite I*)
case *h*:*True*
show *?thesis*
proof(*rule ccontr*)
assume ¬ *f i* ≤ *n*
then have *h0*:*n* < *f i* **by** *simp*
have *f i* ∈ {..<*card I*}
using *bij-betwE*[*OF gf-if-finite*(*1*)[*OF h*]] *assms*(*1*) **by** *simp*
moreover have *n* ∈ {..<*card I*} *n* ≠ *f i*
using *h0* ‹*f i* ∈ {..<*card I*}› **by** *auto*
ultimately have *g n* ≠ *g* (*f i*)
using *bij-betw-imp-inj-on*[*OF gf-if-finite*(*2*)[*OF h*]]
by (*simp add*: *inj-on-contraD*)
thus *False*
by(*simp add*: *gf-comp-id*[*OF assms*(*1*)] *assms*(*2*))
qed
next
show *infinite I* ⟹ *f i* ≤ *n*
using *assms*(*2*) *gf-if-infinite*(*3*)[*of n*] **by** *simp*
qed

lemma *g-surj*:
assumes *i* ∈ *I*
shows ∃ *n*. *g n* = *i*
using *gf-comp-id*[*of i*] *assms* **by** *auto*

lemma *product-dist-summable′*[*simp*]:
summable (λ*n*. *r*^*n* ∗ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*)))
apply(*rule summable-comparison-test′*[*of* λ*n*. *r*^*n* ∗ *K*])
using *r di-nonneg di-bounded K-pos* **by**(*auto intro*!: *summable-mult2*)

lemma *product-dist-summable*[*simp*]:
summable ($\lambda n.$ *if g n* $\in$ *I then r*^*n* $*$ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*)) *else 0*)
by(*rule summable-comparison-test'*[*of* $\lambda n.$ *r*^*n* $*$ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*))])
(*use r di-nonneg di-bounded K-pos* **in** *auto*)

lemma *summable-rK*[*simp*]: *summable* ($\lambda n.$ *r*^*n* $*$ *K*)
using *r* **by**(*auto intro*!: *summable-mult2*)

lemma *Product-metric*: *Metric-space* (Π_E *i*$\in$*I*. *Mi i*) *product-dist*
proof $-$
have *h'*: $\bigwedge$*i xi yi*. *i* $\in$ *I* $\Longrightarrow$ *xi* $\in$ *Mi i* $\Longrightarrow$ *yi* $\in$ *Mi i* $\Longrightarrow$ *xi* = *yi* $\longleftrightarrow$ *di i xi yi* = *0*
$\bigwedge$*i xi yi*. *i* $\in$ *I* $\Longrightarrow$ *di i xi yi* = *di i yi xi*
$\bigwedge$*i xi yi zi*. *i* $\in$ *I* $\Longrightarrow$ *xi* $\in$ *Mi i* $\Longrightarrow$ *yi* $\in$ *Mi i* $\Longrightarrow$ *zi* $\in$ *Mi i* $\Longrightarrow$ *di i xi zi* $\le$ *di i xi yi* + *di i yi zi*
using *Md-metric* **by**(*auto simp*: *Metric-space-def*)
show *?thesis*
proof
show $\bigwedge$*x y*. *0* $\le$ *product-dist x y*
using *di-nonneg r* **by**(*auto simp*: *product-dist-def intro*!: *suminf-nonneg product-dist-summable*)
next
fix *x y*
assume *hxy*:*x* $\in$ (Π_E *i*$\in$*I*. *Mi i*) *y* $\in$ (Π_E *i*$\in$*I*. *Mi i*)
show (*product-dist x y* = *0*) $\longleftrightarrow$ (*x* = *y*)
proof
assume *heq*:*x* = *y*
then have (*if g n* $\in$ *I then r* ^ *n* $*$ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*)) *else 0*) = *0* **for** *n*
using *hxy h'*(*1*)[*of g n x* (*g n*) *y* (*g n*)] **by**(*auto simp*: *product-dist-def*)
thus *product-dist x y* = *0*
by(*auto simp*: *product-dist-def*)
next
assume *h0*:*product-dist x y* = *0*
have ($\sum$*n*. *if g n* $\in$ *I then r* ^ *n* $*$ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*)) *else 0*) = *0*
$\longleftrightarrow$ ($\forall$*n*. (*if g n* $\in$ *I then r*^*n* $*$ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*)) *else 0*) = *0*)
apply(*rule suminf-eq-zero-iff*)
using *di-nonneg r* **by**(*auto simp*: *product-dist-def intro*!: *product-dist-summable*)
hence *hn0*:$\bigwedge$*n*. (*if g n* $\in$ *I then r*^*n* $*$ *di* (*g n*) (*x* (*g n*)) (*y* (*g n*)) *else 0*) = *0*
using *h0 hxy* **by**(*auto simp*: *product-dist-def*)
show *x* = *y*
proof
fix *i*
consider *i* $\in$ *I* | *i* $\notin$ *I* **by** *auto*
thus *x i* = *y i*
proof *cases*
case *1*
from *g-surj*[*OF this*] **obtain** *n* **where**

hn: $g\ n = i$ **by** *auto*
have $di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n)) = 0$
using *hn* $h'(1)[OF\ 1,of\ x\ i\ y\ i]$ *hxy hn0*[*of n*] *1 r* **by** *simp*
thus *?thesis*
using *hn* $h'(1)[OF\ 1,of\ x\ i\ y\ i]$ *hxy 1* **by** *auto*
next
case *2*
then show *?thesis*
by(*simp add*: *PiE-arb*[*OF hxy*(*1*) *2*] *hxy PiE-arb*[*OF hxy*(*2*) *2*])
qed
qed
qed
next
show *product-dist x y = product-dist y x* **for** *x y*
using $h'(2)$ **by**(*auto simp*: *product-dist-def*) (*metis* (*no-types*, *opaque-lifting*))
next
fix *x y z*
assume *hxyz*:$x \in (\Pi_E\ i{\in}I.\ Mi\ i)$ $y \in (\Pi_E\ i{\in}I.\ Mi\ i)$ $z \in (\Pi_E\ i{\in}I.\ Mi\ i)$
have (*if* $g\ n \in I$ *then* $r\ \widehat{}\ n * di\ (g\ n)\ (x\ (g\ n))\ (z\ (g\ n))$ *else 0*)
$\leq$ (*if* $g\ n \in I$ *then* $r\ \widehat{}\ n * di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *else 0*) + (*if* $g\ n \in I$ *then* $r\ \widehat{}\ n * di\ (g\ n)\ (y\ (g\ n))\ (z\ (g\ n))$ *else 0*) **for** *n*
using $h'(3)[of\ g\ n\ x\ (g\ n)\ y\ (g\ n)\ z\ (g\ n)]$ *hxyz r*
by(*auto simp*: *distrib-left*[*of* $r\ \widehat{}\ n$,*symmetric*])
thus $product\text{-}dist\ x\ z \leq product\text{-}dist\ x\ y + product\text{-}dist\ y\ z$
by(*auto simp add*: *product-dist-def suminf-add*[*OF product-dist-summable*[*of x y*] *product-dist-summable*[*of y z*]] *hxyz intro*!: *suminf-le summable-add*)
qed
qed

sublocale *Product-metric*: *Metric-space* $\Pi_E\ i{\in}I.\ Mi\ i$ *product-dist*
by(*rule Product-metric*)

lemma *product-dist-leqr*: $product\text{-}dist\ x\ y \leq 1\ /\ (1 - r) * K$
proof −
have $product\text{-}dist\ x\ y \leq (\sum n.$ *if* $g\ n \in I$ *then* $r\widehat{}n * di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *else 0*)
proof −
consider $x \in (\Pi_E\ i{\in}I.\ Mi\ i) \wedge y \in (\Pi_E\ i{\in}I.\ Mi\ i)\ |\ \neg\ (x \in (\Pi_E\ i{\in}I.\ Mi\ i) \wedge y \in (\Pi_E\ i{\in}I.\ Mi\ i))$ **by** *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis* **by**(*auto simp*: *product-dist-def*)
next
case *2*
then have *product-dist x y = 0*
by(*auto simp*: *product-dist-def*)
also have ... $\leq (\sum n.$ *if* $g\ n \in I$ *then* $r\widehat{}n * di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *else 0*)

```
        using di-nonneg r by(auto intro!: suminf-nonneg product-dist-summable)
      finally show ?thesis .
    qed
  qed
  also have ... ≤ (∑n. r^n * di (g n) (x (g n)) (y (g n)))
   using r di-nonneg di-bounded by(auto intro!: suminf-le)
  also have ... ≤ (∑n. r^n * K)
    using r di-bounded di-nonneg by(auto intro!: suminf-le)
  also have ... = 1 / (1 − r) * K
    using r nsum-of-rK[of 0] by simp
  finally show ?thesis .
qed

lemma product-dist-geq:
  assumes i ∈ I and g n = i x ∈ (Π_E i∈I. Mi i) y ∈ (Π_E i∈I. Mi i)
    shows di i (x i) (y i) ≤ (1/r)^n * product-dist x y
        (is ?lhs ≤ ?rhs)
proof −
  interpret mi: Metric-space Mi i di i
    by(rule Md-metric[OF assms(1)])
  have (λm. if m = f i then di (g m) (x (g m)) (y (g m)) else 0) sums di (g (f i))
(x (g (f i))) (y (g (f i)))
    by(rule sums-single)
  also have ... = ?lhs
    by(simp add: gf-comp-id[OF assms(1)])
  finally have 1:summable (λm. if m = f i then di (g m) (x (g m)) (y (g m)) else
0)
                ?lhs = (∑m. (if m = f i then di (g m) (x (g m)) (y (g m)) else 0))
    by(auto simp: sums-iff)
  note 1(2)
  also have ... ≤ (∑m. (1/r)^n * (if g m ∈ I then r^m * di (g m) (x (g m)) (y
(g m)) else 0))
  proof(rule suminf-le)
    show summable (λm. (1/r)^n * (if g m ∈ I then r^m * di (g m) (x (g m)) (y
(g m)) else 0))
      by(auto intro!: product-dist-summable)
  next
    fix k
    have **:1 ≤ (1/r) ^ n * r ^ f i
    proof −
      have (1/r) ^ n * (r ^ f i) = (1/r)^(n−f i) * (1/r)^(f i) * r ^ f i
      using r by(simp add: power-diff[OF - i-min[OF assms(1,2)],of 1/r,simplified])
      also have ... = (1/r) ^ (n−f i)
        using r by (simp add: power-one-over)
      finally show ?thesis
        using r by auto
    qed
    have *:g k ∈ I if k = f i
      using gf-comp-id[OF assms(1)] assms(1) that by auto
```

show (*if* $k = f\ i$ *then di* (*g k*) (*x* (*g k*)) (*y* (*g k*)) *else 0*) $\leq$ (*1*/*r*) ^ *n* ∗ (*if g k* ∈ *I then r* ^ *k* ∗ *di* (*g k*) (*x* (*g k*)) (*y* (*g k*)) *else 0*)
using ∗ *di-nonneg r* ∗∗ *mult-right-mono*[*OF* ∗∗] **by**(*auto simp*: *vector-space-over-itself.scale-scale*[*of* (*1* / *r*) ^ *n*])
qed(*simp add*: *1*)
also have ... = *?rhs*
unfolding *product-dist-def*
using *assms* **by**(*auto intro*!: *suminf-mult product-dist-summable*)
finally show *?thesis* **.**
qed

lemma *limitin-M-iff-limitin-Mi*:
shows *limitin Product-metric.mtopology xn x sequentially* ⟷ (∃ *N*. ∀ *n*≥*N*. (∀ *i*∈*I*. *xn n i* ∈ *Mi i*) ∧ (∀ *i*. *i*∉*I* ⟶ *xn n i* = *undefined*)) ∧ (∀ *i*∈*I*. *limitin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (λ*n*. *xn n i*) (*x i*) *sequentially*) ∧ *x* ∈ (Π_E *i*∈*I*. *Mi i*)
proof *safe*
fix *i*
assume *h*:*limitin Product-metric.mtopology xn x sequentially*
then show ∃ *N*. ∀ *n*≥*N*. (∀ *i*∈*I*. *xn n i* ∈ *Mi i*) ∧ (∀ *i*. *i*∉*I* ⟶ *xn n i* = *undefined*)
by(*simp only*: *Product-metric.limit-metric-sequentially*) (*metis PiE-E r*(*1*))
then obtain *N′* **where** *N′*: ⋀*i n*. *i* ∈ *I* ⟹ *n* ≥ *N′* ⟹ *xn n i* ∈ *Mi i* ⋀*i n*. *i* ∉ *I* ⟹ *n* ≥ *N′* ⟹ *xn n i* = *undefined*
by *blast*
assume *i*:*i* ∈ *I*
then interpret *m*: *Metric-space Mi i di i*
using *Md-metric* **by** *blast*
show *limitin m.mtopology* (λ*n*. *xn n i*) (*x i*) *sequentially*
unfolding *m.limitin-metric eventually-sequentially*
proof *safe*
show *x i* ∈ *Mi i*
using *h i* **by**(*auto simp*: *Product-metric.limitin-metric*)
next
fix ε :: *real*
assume *0* < ε
then obtain *r*^*f i* ∗ ε > *0* **using** *r* **by** *auto*
then obtain *N* **where** *N*:⋀*n*. *n* ≥ *N* ⟹ *product-dist* (*xn n*) *x* < *r*^*f i* ∗ ε
using *h*(*1*) **by**(*fastforce simp*: *Product-metric.limitin-metric eventually-sequentially*)
show ∃ *N*. ∀ *n*≥*N*. *xn n i* ∈ *Mi i* ∧ *di i* (*xn n i*) (*x i*) < ε
proof(*safe intro*!: *exI*[**where** *x*=*max N N′*])
fix *n*
assume *max N N′* ≤ *n*
then have *N* ≤ *n N′* ≤ *n*
by *auto*
have *di i* (*xn n i*) (*x i*) ≤ (*1* / *r*) ^*f i* ∗ *product-dist* (*xn n*) *x*
thm *product-dist-geq*[*OF i gf-comp-id*[*OF i*]]
using *h i N′*[*OF* - ‹*N′* ≤ *n*›] **by**(*auto intro*!: *product-dist-geq*[*OF i gf-comp-id*[*OF i*]] *dest*: *Product-metric.limitin-mspace*)

```
      also have ... < (1 / r) ^f i * r^f i * ε
        using N[OF ‹N ≤ n›] r by auto
      also have ... ≤ ε
        by (simp add: ‹0 < ε› power-one-over)
      finally show di i (xn n i) (x i) < ε .
    qed(use N' h i in auto)
  qed
next
  fix N'
  assume N': ∀n≥N'. (∀i∈I. xn n i ∈ Mi i) ∧ (∀i. i ∉ I ⟶ xn n i = undefined)
  assume h:∀i∈I. limitin (Metric-space.mtopology (Mi i) (di i)) (λn. xn n i) (x i)
sequentially x ∈ (Π_E i∈I. Mi i)
  show limitin Product-metric.mtopology xn x sequentially
    unfolding Product-metric.limitin-metric eventually-sequentially
  proof safe
    fix ε
    assume he:(0::real) < ε
    then have 0 < ε*((1−r)/K) using r K-pos by auto
    hence ∃k. r^k < ε*((1−r)/K)
      using r(2) real-arch-pow-inv by blast
    then obtain l where r^l < ε*((1−r)/K) by auto
    hence hk:r^l/(1−r)*K < ε
      using mult-imp-div-pos-less[OF divide-pos-pos[OF - K-pos,of 1−r]] r(2) by
simp
    hence hke: 0 < ε − r^l/(1−r)*K by auto
    consider l = 0 | 0 < l by auto
    then show ∃N. ∀n≥N. xn n ∈ (Π_E i∈I. Mi i) ∧ product-dist (xn n) x < ε
    proof cases
      case 1
      then have he2:1 / (1 − r)*K < ε using hk by auto
      show ?thesis
        using order.strict-trans1[OF product-dist-leqr he2] N'
        by(auto intro!: exI[where x=N'])
    next
      case 2
      with hke have 0 < 1 / real l * (ε − r^l/(1−r)*K) by auto
      hence ∀i∈I. ∃N. ∀n≥N. di i (xn n i) (x i) < 1 / real l * (ε − r^l/(1−r)*K)
        using h by (meson Md-metric Metric-space.limit-metric-sequentially)
      then obtain N where hn:
    ⋀i n. i ∈ I ⟹ n ≥ N i ⟹ di i (xn n i) (x i) < 1 / real l * (ε − r^l/(1−r)*K)
        by metis
      show ?thesis
      proof(safe intro!: exI[where x=max (Sup {N (g n) | n. n < l}) N'])
        fix n
        assume max (⨆ {N (g n) |n. n < l}) N' ≤ n
        then have hsup:⨆ {N (g n) |n. n < l} ≤ n and N'n: N' ≤ n
          by auto
        have product-dist (xn n) x = (∑m. if g m ∈ I then r ^ m * di (g m) (xn n
(g m)) (x (g m)) else 0)
```

using *N′ N′n h* **by**(*auto simp*: *product-dist-def*)
also have ... = ($\sum$*m. if g* (*m* + *l*) ∈ *I then r* ^ (*m* + *l*)∗ *di* (*g* (*m* + *l*)) (*xn n* (*g* (*m* + *l*))) (*x* (*g* (*m* + *l*))) *else 0*) + ($\sum$*m*<*l. if g m* ∈ *I then r* ^ *m* ∗ *di* (*g m*) (*xn n* (*g m*)) (*x* (*g m*)) *else 0*)
by(*auto intro*!: *suminf-split-initial-segment*)
also have ... ≤ *r*^*l*/(*1*−*r*)∗*K* + ($\sum$*m*<*l. if g m* ∈ *I then r* ^ *m* ∗ *di* (*g m*) (*xn n* (*g m*)) (*x*(*g m*)) *else 0*)
proof −
have ($\sum$*m. if g* (*m* + *l*) ∈ *I then r* ^ (*m* + *l*)∗ *di* (*g* (*m* + *l*)) (*xn n* (*g* (*m* + *l*))) (*x* (*g* (*m* + *l*))) *else 0*) ≤ ($\sum$*m. r*^(*m* + *l*)∗*K*)
using *di-bounded N′ r K-pos* **by**(*auto intro*!: *suminf-le summable-ignore-initial-segment*)
also have ... = *r*^*l*/(*1*−*r*)∗*K*
by(*rule nsum-of-rK*)
finally show *?thesis* **by** *auto*
qed
also have ... ≤ *r*^*l* / (*1* − *r*)∗*K* + ($\sum$*m*<*l. if g m* ∈ *I then di* (*g m*) (*xn n* (*g m*)) (*x* (*g m*)) *else 0*)
proof −
have ($\sum$*m*<*l. if g m* ∈ *I then r* ^ *m* ∗ *di* (*g m*) (*xn n* (*g m*)) (*x* (*g m*)) *else 0*) ≤ ($\sum$*m*<*l. if g m* ∈ *I then di* (*g m*) (*xn n* (*g m*)) (*x* (*g m*)) *else 0*)
using *di-bounded di-nonneg r* **by**(*auto intro*!: *sum-mono simp*: *mult-left-le-one-le power-le-one*)
thus *?thesis* **by** *simp*
qed
also have ... < *r*^*l* / (*1* − *r*)∗*K* + ($\sum$*m*<*l. 1* / *real l* ∗ (ε − *r*^*l*/(*1*−*r*)∗*K*))
proof −
have ($\sum$*m*<*l. if g m* ∈ *I then di* (*g m*) (*xn n* (*g m*)) (*x* (*g m*)) *else 0*) < ($\sum$*m*<*l. 1* / *real l* ∗ (ε − *r*^*l*/(*1*−*r*)∗*K*))
proof(*rule sum-strict-mono-ex1*)
show ∀ *p*∈{..<*l*}. (*if g p* ∈ *I then di* (*g p*) (*xn n* (*g p*)) (*x* (*g p*)) *else 0*) ≤ *1* / *real l* ∗ (ε − *r* ^ *l* / (*1* − *r*)∗*K*)
proof −
have *0* ≤ (ε − *r* ^ *l* ∗ *K* / (*1* − *r*)) / *real l*
using *hke* **by** *auto*
moreover {
fix *p*
assume *p* < *l g p* ∈ *I*
then have *N* (*g p*) ∈ {*N* (*g n*) |*n. n* < *l*}
by *auto*
from *le-cSup-finite*[*OF* - *this*] *hsup* **have** *N* (*g p*) ≤ *n*
by *auto*
hence *di* (*g p*) (*xn n* (*g p*)) (*x* (*g p*)) ≤ (ε − *r* ^ *l* ∗*K*/ (*1* − *r*)) / *real l*
using *hn*[*OF* ‹*g p* ∈ *I*›,*of n*] **by** *simp*
}
ultimately show *?thesis*
by *auto*
qed
next

show ∃ *a*∈{..<*l*}. (*if g a* ∈ *I then di* (*g a*) (*xn n* (*g a*)) (*x* (*g a*)) *else 0*) < *1* / *real l* ∗ (ε − *r* ^ *l* / (*1* − *r*)∗*K*)
proof −
have *0* < (ε − *r* ^ *l* ∗ *K* / (*1* − *r*)) / *real l*
using *hke 2* **by** *auto*
moreover {
assume *g 0* ∈ *I*
have *N* (*g 0*) ∈ {*N* (*g n*) |*n*. *n* < *l*}
using *2* **by** *auto*
from *le-cSup-finite*[*OF - this*] *hsup* **have** *N* (*g 0*) ≤ *n*
by *auto*
hence *di* (*g 0*) (*xn n* (*g 0*)) (*x* (*g 0*)) < (ε − *r* ^ *l* ∗ *K*/ (*1* − *r*)) / *real l*
using *hn*[*OF* ‹*g 0* ∈ *I*›,*of n*] **by** *simp*
}
ultimately show *?thesis*
by(*auto intro*!: *bexI*[**where** *x*=*0*] *simp*: *2*)
qed
qed *simp*
thus *?thesis* **by** *simp*
qed
also have ... = ε
using *2* **by** *auto*
finally show *product-dist* (*xn n*) *x* < ε **.**
qed(*use N*′ **in** *auto*)
qed
qed (*use N*′ *h* **in** *auto*)
qed(*auto simp*: *Product-metric.limitin-metric*)

lemma *Product-metric-mtopology-eq*: *product-topology* (λ*i*. *Metric-space.mtopology* (*Mi i*) (*di i*)) *I* = *Product-metric.mtopology*
proof −
have *htopspace*:⋀*i*. *i* ∈ *I* ⟹ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*)) = *Mi i*
by (*simp add*: *Md-metric Metric-space.topspace-mtopology*)
hence *htopspace*′:(Π_E *i*∈*I*. *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))) = (Π_E *i*∈*I*. *Mi i*) **by** *auto*
consider *I* = {} | *I* ≠ {} **by** *auto*
then show *?thesis*
proof *cases*
case *1*
interpret *d*: *discrete-metric* {λ*x*. *undefined*} **.**
have *product-dist* = (λ*x y*. *0*)
by *standard*+ (*auto simp*: *product-dist-def 1*)
hence *2*:*Product-metric.mtopology* = *d.disc.mtopology*
by (*metis 1 PiE-empty-domain Product-metric.open-in-mspace Product-metric.topspace-mtopology d.mtopology-discrete-metric discrete-topology-unique singleton-iff*)
show *?thesis*
unfolding *2* **by**(*simp add*: *product-topology-empty-discrete 1 d.mtopology-discrete-metric*)

next
case *I′:2*
show *?thesis*
unfolding *base-is-subbase*[*OF Product-metric.mtopology-base-in-balls,simplified subbase-in-def*] *product-topology-def*
proof(*rule topology-generated-by-eq*)
fix *U*
assume $U \in \{Product\text{-}metric.mball\ a\ \varepsilon\ |a\ \varepsilon.\ a \in (\Pi_E\ i{\in}I.\ Mi\ i) \wedge 0 < \varepsilon\}$
then obtain *a ε* **where** *hu*:
$U = Product\text{-}metric.mball\ a\ \varepsilon\ a \in (\Pi_E\ i{\in}I.\ Mi\ i)\ 0 < \varepsilon$ **by** *auto*
have $\exists X.\ x \in (\Pi_E\ i{\in}I.\ X\ i) \wedge (\Pi_E\ i{\in}I.\ X\ i) \subseteq U \wedge (\forall i.\ openin\ (Metric\text{-}space.mtopology\ (Mi\ i)\ (di\ i))\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (Metric\text{-}space.mtopology\ (Mi\ i)\ (di\ i))\}$ **if** $x \in U$ **for** *x*
proof −
consider $\varepsilon \le 1\ /\ (1 - r) * K \mid 1\ /\ (1 - r) * K < \varepsilon$ **by** *fastforce*
then show $\exists X.\ x \in (\Pi_E\ i{\in}I.\ X\ i) \wedge (\Pi_E\ i{\in}I.\ X\ i) \subseteq U \wedge (\forall i.\ openin\ (Metric\text{-}space.mtopology\ (Mi\ i)\ (di\ i))\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (Metric\text{-}space.mtopology\ (Mi\ i)\ (di\ i))\}$
proof *cases*
case *he2:1*
have $0 < (\varepsilon - product\text{-}dist\ a\ x)*((1-r)/\ K)$ **using** *r hu K-pos that hu* **by** *auto*
hence $\exists k.\ r\hat{}k < (\varepsilon - product\text{-}dist\ a\ x)*((1-r)/\ K)$
using *r(2) real-arch-pow-inv* **by** *blast*
then obtain *k* **where** $r\hat{}k < (\varepsilon - product\text{-}dist\ a\ x)*((1-r)/\ K)$ **by** *auto*
hence $hk{:}r\hat{}k\ /\ (1-r) * K < (\varepsilon - product\text{-}dist\ a\ x)$
using *mult-imp-div-pos-less*[*OF divide-pos-pos*[*OF - K-pos,of 1−r*]] *r(2)* **by** *auto*
have *hk′*: $0 < k$ **apply**(*rule ccontr*) **using** *hk he2 Product-metric.nonneg*[*of a x*] **by** *auto*
define *ε′* **where** $\varepsilon' \equiv (1/(real\ k))*(\varepsilon - product\text{-}dist\ a\ x - r\hat{}k\ /\ (1-r) * K)$
have *hε′* : $0 < \varepsilon'$ **using** *hk* **by**(*auto simp*: *ε′-def hk′*)
define *X* **where** $X \equiv (if\ finite\ I\ then\ (\lambda i.\ if\ i \in I\ then\ Metric\text{-}space.mball\ (Mi\ i)\ (di\ i)\ (x\ i)\ \varepsilon'\ else\ topspace\ (Metric\text{-}space.mtopology\ (Mi\ i)\ (di\ i)))\ else\ (\lambda i.\ if\ i \in I \wedge f\ i < k\ then\ Metric\text{-}space.mball\ (Mi\ i)\ (di\ i)\ (x\ i)\ \varepsilon'\ else\ topspace\ (Metric\text{-}space.mtopology\ (Mi\ i)\ (di\ i))))$
show *?thesis*
proof(*intro exI*[**where** *x=X*] *conjI*)
have $x\ i \in Metric\text{-}space.mball\ (Mi\ i)\ (di\ i)\ (x\ i)\ \varepsilon'$ **if** $i \in I$ **for** *i*
using *hu* ‹$x \in U$› **by** (*auto simp add*: *PiE-mem hε′ Md-metric Metric-space.centre-in-mball-iff that*)
thus $x \in (\Pi_E\ i{\in}I.\ X\ i)$
using *hu that htopspace* **by**(*auto simp*: *X-def*)
next
show $(\Pi_E\ i{\in}I.\ X\ i) \subseteq U$
proof
fix *y*
assume $y \in (\Pi_E\ i{\in}I.\ X\ i)$

have $\bigwedge i.\ X\ i \subseteq topspace\ (Metric\text{-}space.mtopology\ (Mi\ i)\ (di\ i))$
by (*simp add*: *Md-metric Metric-space.mball-subset-mspace X-def htopspace*)
hence $y \in (\Pi_E\ i{\in}I.\ Mi\ i)$
using *htopspace'* ‹$y \in (\Pi_E\ i{\in}I.\ X\ i)$› **by** *blast*
have $product\text{-}dist\ a\ y < \varepsilon$
proof −
have $product\text{-}dist\ a\ y \le product\text{-}dist\ a\ x + product\text{-}dist\ x\ y$
using *Product-metric.triangle* ‹$y \in Pi_E\ I\ Mi$› *hu*(*1*) *that* **by** *auto*
also have ... $< product\text{-}dist\ a\ x + (\varepsilon - product\text{-}dist\ a\ x)$
proof −
have $product\text{-}dist\ x\ y < (\varepsilon - product\text{-}dist\ a\ x)$
proof −
have $product\text{-}dist\ x\ y = (\sum n.\ if\ g\ n \in I\ then\ r\ \hat{}\ n * di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))\ else\ 0)$
by (*metis* (*no-types*, *lifting*) *hu*(*1*) *that* ‹$y \in (\Pi_E\ i{\in}I.\ Mi\ i)$› *Product-metric.in-mball product-dist-def suminf-cong*)
also have ... $= (\sum n.\ if\ g\ (n + k) \in I\ then\ r\ \hat{}\ (n + k)*\ di\ (g\ (n + k))\ (x\ (g\ (n + k)))\ (y\ (g\ (n + k)))\ else\ 0) + (\sum n{<}k.\ if\ g\ n \in I\ then\ r\ \hat{}\ n * di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))\ else\ 0)$
by(*rule suminf-split-initial-segment*) *simp*
also have ... $\le r\hat{}k\ /\ (1 - r) * K + (\sum n{<}k.\ if\ g\ n \in I\ then\ r\ \hat{}\ n * di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))\ else\ 0)$
proof −
have $(\sum n.\ if\ g\ (n + k) \in I\ then\ r\ \hat{}\ (n + k)*\ di\ (g\ (n + k))\ (x\ (g\ (n + k)))\ (y\ (g\ (n + k)))\ else\ 0) \le (\sum n.\ r\ \hat{}\ (n + k) * K)$
using *di-bounded di-nonneg r K-pos* **by**(*auto intro*!: *suminf-le summable-ignore-initial-segment*)
also have ... $= r\hat{}k\ /\ (1 - r) * K$
by(*rule nsum-of-rK*)
finally show *?thesis* **by** *simp*
qed
also have ... $< r\hat{}k\ /\ (1 - r) * K + (\varepsilon - product\text{-}dist\ a\ x - r\hat{}k\ /\ (1 - r) * K)$
proof −
have $(\sum n{<}k.\ if\ g\ n \in I\ then\ r\ \hat{}\ n * di\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))\ else\ 0) < (\sum n{<}k.\ \varepsilon')$
proof(*rule sum-strict-mono-ex1*)
show $\forall\ l{\in}\{..{<}k\}.\ (if\ g\ l \in I\ then\ r\ \hat{}\ l * di\ (g\ l)\ (x\ (g\ l))\ (y\ (g\ l))\ else\ 0) \le \varepsilon'$
proof −
{
fix *l*
assume $g\ l \in I\ l < k$
then interpret *mbd*: *Metric-space Mi* (*g l*) *di* (*g l*)
by(*auto intro*!: *Md-metric*)
have $r\ \hat{}\ l * di\ (g\ l)\ (x\ (g\ l))\ (y\ (g\ l)) \le di\ (g\ l)\ (x\ (g\ l))\ (y\ (g\ l))$
using *r* **by**(*auto intro*!: *mult-right-mono*[*of r ^ l 1*,*OF* -

```
mbd.nonneg[of x (g l) y (g l)],simplified] simp: power-le-one)
                also have ... < ε′
                proof −
                  have y (g l) ∈ mbd.mball (x (g l)) ε′
                  proof(cases finite I)
                    case True
                    then show ?thesis
                      using PiE-mem[OF ‹y ∈ (Π_E i∈I. X i)› ‹g l ∈ I›]
                      by(simp add: X-def ‹g l ∈ I›)
                  next
                    case False
                    then show ?thesis
                        using PiE-mem[OF ‹y ∈ (Π_E i∈I. X i)› ‹g l ∈ I›]
gf-if-infinite(3)
                      by(simp add: X-def ‹g l ∈ I› ‹l < k›)
                  qed
                  thus ?thesis
                    by auto
                qed
               finally have r ^ l ∗ di (g l) (x (g l)) (y (g l)) ≤ ε′ by simp
              }
              thus ?thesis
                by(auto simp: order.strict-implies-order[OF hε′])
            qed
          next
            show ∃ a∈{..<k}. (if g a ∈ I then r ^ a ∗ di (g a) (x (g a)) (y
(g a)) else 0) < ε′
            proof(rule bexI[where x=0])
              {
                assume g 0 ∈ I
                then interpret mbd: Metric-space Mi (g 0) di (g 0)
                  by(auto intro!: Md-metric)
                have y (g 0) ∈ mbd.mball (x (g 0)) ε′
                proof(cases finite I)
                  case True
                  then show ?thesis
                    using PiE-mem[OF ‹y ∈ (Π_E i∈I. X i)› ‹g 0 ∈ I›]
                    by(simp add: X-def ‹g 0 ∈ I›)
                next
                  case False
                  then show ?thesis
                      using PiE-mem[OF ‹y ∈ (Π_E i∈I. X i)› ‹g 0 ∈ I›]
gf-if-infinite(3)
                    by(simp add: X-def ‹g 0 ∈ I› ‹0 < k›)
                qed
                hence r ^ 0 ∗ di (g 0) (x (g 0)) (y (g 0))  < ε′
                  by auto
              }
             thus (if g 0 ∈ I then r ^ 0 ∗ di (g 0) (x (g 0)) (y (g 0)) else
```

```
0) < ε′
                  using  hε′ by auto
                qed(use hk′ in auto)
              qed simp
              also have ... = (ε − product-dist a x − r ^ k / (1 − r) ∗ K)
                by(simp add: ε′-def hk′)
              finally show ?thesis by simp
            qed
            finally show ?thesis by simp
          qed
          thus ?thesis by simp
        qed
        finally show ?thesis by auto
      qed
      thus y ∈ U
        by(simp add: hu(1) hu(2) ‹y ∈ (Π_E i∈I. Mi i)›)
    qed
  next
     have openin (Metric-space.mtopology (Mi i) (di i)) (Metric-space.mball
(Mi i) (di i) (x i) ε′) if i ∈ I for i
      by (simp add: Md-metric Metric-space.openin-mball that)
    moreover have openin (Metric-space.mtopology (Mi i) (di i)) (topspace
(Metric-space.mtopology (Mi i) (di i))) for i
      by auto
    ultimately show ∀ i. openin (Metric-space.mtopology (Mi i) (di i)) (X
i)
      by(auto simp: X-def)
  next
    show finite {i. X i ≠ topspace (Metric-space.mtopology (Mi i) (di i))}
    proof(cases finite I)
      case True
      then show ?thesis
        by(simp add: X-def)
    next
      case Iinf:False
      have finite {i ∈ I. f i < k}
      proof −
        have {i ∈ I. f i < k} = inv-into I f ‘ {..<k}
        proof −
          have ∗:⋀i. i ∈ I =⇒ inv-into I f (f i) = i
                ⋀n. f (inv-into I f n) = n
            using bij-betw-inv-into-left[OF gf-if-infinite(1)[OF Iinf]]
                  bij-betw-inv-into-right[OF gf-if-infinite(1)[OF Iinf]]
            by auto
          show ?thesis
          proof
            show {i ∈ I. f i < k} ⊆ inv-into I f ‘ {..<k}
            proof
            show p ∈ {i ∈ I. f i < k} =⇒ p ∈ inv-into I f ‘ {..<k} for p
```

using $*(1)[of\ p]$ **by** (*auto simp*: *rev-image-eqI*)
qed
next
show *inv-into I f ' {..<k} ⊆ {i ∈ I. f i < k}*
using $*(2)$ *bij-betw-inv-into*[*OF gf-if-infinite*(*1*)[*OF Iinf*]]
by (*auto simp*: *bij-betw-def*)
qed
qed
also have *finite ...* **by** *auto*
finally show *?thesis* **.**
qed
thus *?thesis*
by(*simp add*: *X-def Iinf*)
qed
qed
next
case *2*
then have $U = (\Pi_E\ i{\in}I.\ Mi\ i)$
unfolding *hu*(*1*) **using** *order.strict-trans1*[*OF product-dist-leqr*,*of ε*] *hu*(*2*)
by *auto*
also have ... $= (\Pi_E\ i{\in}I.$ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*)))
using *htopspace* **by** *auto*
finally have $U = (\Pi_E\ i{\in}I.$ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))) **.**
thus *?thesis*
using *that hu htopspace* **by**(*auto intro*!: *exI*[**where** *x=λi. topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))])
qed
qed
hence $\exists X.\ \forall x{\in}U.\ x \in (\Pi_E\ i{\in}I.\ X\ x\ i) \wedge (\Pi_E\ i{\in}I.\ X\ x\ i) \subseteq U \wedge (\forall i.$ *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X x i*)) ∧ *finite* {*i. X x i ≠ topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}
by(*auto intro*!: *bchoice*)
then obtain *X* **where** $\forall x{\in}U.\ x \in (\Pi_E\ i{\in}I.\ X\ x\ i) \wedge (\Pi_E\ i{\in}I.\ X\ x\ i) \subseteq U \wedge (\forall i.$ *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X x i*)) ∧ *finite* {*i. X x i ≠ topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}
by *auto*
hence *hX*: $\bigwedge x.\ x \in U \Longrightarrow x \in (\Pi_E\ i{\in}I.\ X\ x\ i)$ $\bigwedge x.\ x \in U \Longrightarrow (\Pi_E\ i{\in}I.\ X\ x\ i) \subseteq U$ $\bigwedge x.\ x \in U \Longrightarrow (\forall i.$ *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X x i*)) $\bigwedge x.\ x \in U \Longrightarrow$ *finite* {*i. X x i ≠ topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}
by *auto*
hence *hXopen*: $\bigwedge x.\ x \in U \Longrightarrow (\Pi_E\ i{\in}I.\ X\ x\ i) \in \{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.$ *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X i*)) ∧ *finite* {*i. X i ≠ topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}}
by *blast*
have $U = (\bigcup \{(\Pi_E\ i{\in}I.\ X\ x\ i) \mid x.\ x \in U\})$
using *hX*(*1*,*2*) **by** *blast*

have *openin* (*topology-generated-by* {Π_E *i*∈*I*. *X i* |*X*. (∀ *i*. *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}}) (⋃ {(Π_E *i*∈*I*. *X x i*) | *x*. *x* ∈ *U*})
apply(*rule openin-Union*)
using *hXopen* **by**(*auto simp*: *openin-topology-generated-by-iff intro*!: *generate-topology-on.Basis*)
thus *openin* (*topology-generated-by* {Π_E *i*∈*I*. *X i* |*X*. (∀ *i*. *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}}) *U*
using ‹*U* = (⋃ {(Π_E *i*∈*I*. *X x i*) | *x*. *x* ∈ *U*})› **by** *simp*
next
fix *U*
assume *U* ∈ {Π_E *i*∈*I*. *X i* |*X*. (∀ *i*. *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}}
then obtain *X* **where** *hX*:
U = (Π_E *i*∈*I*. *X i*) ⋀*i*. *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X i*) *finite* {*i*. *X i* ≠ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))}
by *auto*
have ∃ *a ε*. *x* ∈ *Product-metric.mball a ε* ∧ *Product-metric.mball a ε* ⊆ *U* **if** *x* ∈ *U* **for** *x*
proof −
have *x-intop*:*x* ∈ (Π_E *i*∈*I*. *Mi i*)
unfolding *htopspace′*[*symmetric*] **using** *that hX*(*1*) *openin-subset*[*OF hX*(*2*)] **by** *auto*
define *I′* **where** *I′* ≡ {*i*. *X i* ≠ *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))} ∩ *I*
then have *I′*:*finite I′ I′* ⊆ *I* **using** *hX*(*3*) **by** *auto*
consider *I′* = {} | *I′* ≠ {} **by** *auto*
then show *?thesis*
proof *cases*
case *1*
then have ⋀*i*. *i* ∈ *I* ⟹ *X i* = *topspace* (*Metric-space.mtopology* (*Mi i*) (*di i*))
by(*auto simp*: *I′-def*)
then have *U* = (Π_E *i*∈*I*. *Mi i*)
by (*simp add*: *PiE-eq hX*(*1*) *htopspace*)
thus *?thesis*
using *1 that* **by**(*auto intro*!: *exI*[**where** *x*=*x*] *exI*[**where** *x*=*1*])
next
case *I′-nonempty*:*2*
hence ⋀*i*. *i* ∈ *I′* ⟹ *openin* (*Metric-space.mtopology* (*Mi i*) (*di i*)) (*X i*)
using *hX*(*2*) **by**(*simp add*: *I′-def*)
hence ∃*ε*>*0*. *Metric-space.mball* (*Mi i*) (*di i*) (*x i*) *ε* ⊆ (*X i*) **if** *i* ∈ *I′* **for** *i*
using *I′*(*2*) *Md-metric Metric-space.openin-mtopology PiE-mem* ‹*x* ∈ *U*› *hX*(*1*) *subset-eq that* **by** *blast*
then obtain *εi′* **where** *hei*:⋀*i*. *i* ∈ *I′* ⟹ *εi′ i* > *0* ⋀*i*. *i* ∈ *I′* ⟹ *Metric-space.mball* (*Mi i*) (*di i*) (*x i*) (*εi′ i*) ⊆ (*X i*)
by *metis*

```
      define ε where ε ≡ Min {εi′ i |i. i ∈ I′}
      have εmin: ⋀i. i ∈ I′ ⟹ ε ≤ εi′ i
        using I′ by(auto simp: ε-def intro!: Min.coboundedI)
      have hε: ε > 0
        using I′ I′-nonempty Min-gr-iff[of {εi′ i |i. i ∈ I′} 0] hei(1)
        by(auto simp: ε-def)
      define n where n ≡ Max {f i | i. i ∈ I′}
      have ⋀i. i ∈ I′ ⟹f i ≤ n
       using I′ by(auto intro!: Max.coboundedI[of {f i | i. i ∈ I′}] simp: n-def)
      hence hn2:⋀i. i ∈ I′ ⟹ (1 / r) ^f i ≤ (1 / r)^n
        using r by auto
      have hε′ : 0 < ε∗(r^n) using hε r by auto
      show ?thesis
      proof(safe intro!: exI[where x=x] exI[where x=ε∗(r^n)])
        fix y
        assume y ∈ Product-metric.mball x (ε ∗ r ^ n)
        have y i ∈ X i if i ∈ I′ for i
        proof −
          interpret mi: Metric-space Mi i di i
            using Md-metric that by(simp add: I′-def)
          have di i (x i) (y i) < εi′ i
          proof −
            have di i (x i) (y i) ≤ (1 / r) ^f i ∗ product-dist x y
              using that ‹y ∈ Product-metric.mball x (ε ∗ r ^ n)› by(auto intro!:
product-dist-geq[of i,OF - gf-comp-id x-intop] simp: I′-def)
            also have ... ≤ (1 / r)^n ∗ product-dist x y
              by(rule mult-right-mono[OF hn2[OF that] Product-metric.nonneg])
            also have ... < ε
              using ‹y ∈ Product-metric.mball x (ε ∗ r ^ n)› r
              by (simp add: pos-divide-less-eq power-one-over)
            also have ... ≤ εi′ i
              by(rule εmin[OF that])
            finally show ?thesis .
          qed
          hence (y i) ∈ mi.mball (x i) (εi′ i)
            using ‹y ∈ Product-metric.mball x (ε ∗ r ^ n)› x-intop that
            by(auto simp: I′-def)
          thus ?thesis
            using hei[OF that] by auto
        qed
        moreover have y i ∈ X i if i ∈ I − I′ for i
          using that htopspace ‹y ∈ Product-metric.mball x (ε ∗ r ^ n)›
          by(auto simp: I′-def)
        ultimately show y ∈ U
          using ‹y ∈ Product-metric.mball x (ε ∗ r ^ n)›
          by(auto simp: hX(1))
      qed(use x-intop hε′ in auto)
    qed
  qed
```

then obtain *a* **where** $\forall x{\in}U.\ \exists\varepsilon.\ x \in$ *Product-metric.mball* $(a\ x)\ \varepsilon \wedge$ *Product-metric.mball* $(a\ x)\ \varepsilon \subseteq U$
by *metis*
then obtain ε **where** *hae*: $\bigwedge x.\ x \in U \Longrightarrow x \in$ *Product-metric.mball* $(a\ x)$ $(\varepsilon\ x)$ $\bigwedge x.\ x \in U \Longrightarrow$ *Product-metric.mball* $(a\ x)\ (\varepsilon\ x) \subseteq U$
by *metis*
hence *hae'*: $\bigwedge x.\ x \in U \Longrightarrow a\ x \in (\Pi_E\ i{\in}I.\ Mi\ i)$ $\bigwedge x.\ x \in U \Longrightarrow 0 < \varepsilon\ x$
by *auto*[*1*] (*metis Product-metric.mball-eq-empty empty-iff hae*(*1*) *linorder-not-less*)
have *openin* (*topology-generated-by* {*Product-metric.mball* $a\ \varepsilon$ |$a\ \varepsilon.\ a \in (\Pi_E\ i{\in}I.\ Mi\ i) \wedge 0 < \varepsilon$}) ($\bigcup$ { *Product-metric.mball* $(a\ x)\ (\varepsilon\ x)$ |$x.\ x \in U$})
using *Product-metric.openin-mball* ‹*Product-metric.mtopology* = *topology-generated-by* {*Product-metric.mball* $a\ \varepsilon$ |$a\ \varepsilon.\ a \in Pi_E\ I\ Mi \wedge 0 < \varepsilon$}› **by** *auto*
moreover have $U = (\bigcup$ {*Product-metric.mball* $(a\ x)\ (\varepsilon\ x)$ |$x.\ x \in U$})
using *hae* **by** (*auto simp del*: *Product-metric.in-mball*)
ultimately show *openin* (*topology-generated-by* {*Product-metric.mball* $a\ \varepsilon$ |$a\ \varepsilon.\ a \in (\Pi_E\ i{\in}I.\ Mi\ i) \wedge 0 < \varepsilon$}) U
by *simp*
qed
qed
qed

corollary *separable-Mi-separable-M*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *separable-space* (*Metric-space.mtopology* (*Mi i*) (*di i*))
shows *separable-space Product-metric.mtopology*
by(*simp add*: *Product-metric-mtopology-eq*[*symmetric*] *separable-countable-product assms I*)

lemma *mcomplete-Mi-mcomplete-M*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *Metric-space.mcomplete* (*Mi i*) (*di i*)
shows *Product-metric.mcomplete*
proof(*cases* $I = \{\}$)
case *1*:*True*
interpret *d*: *discrete-metric* {$\lambda x.$ *undefined*} **.**
have *2*:*product-dist* $= (\lambda x\ y.\ 0)$
by *standard+* (*auto simp*: *product-dist-def 1*)
show *?thesis*
apply(*simp add*: *Product-metric.mcomplete-def Product-metric.limitin-metric eventually-sequentially Product-metric.MCauchy-def*)
apply(*simp add*: *2*)
by(*auto simp*: *1 intro*!: *exI*[**where** $x=(\lambda i.$ *undefined*)])
next
assume *2*: $I \neq \{\}$
show *?thesis*
unfolding *Product-metric.mcomplete-def*
proof *safe*
fix *xn*
assume *h*: *Product-metric.MCauchy xn*
have $*$:*Metric-space.MCauchy* (*Mi i*) (*di i*) ($\lambda n.$ *xn n i*) **if** *hi*:$i \in I$ **for** *i*

```
  proof -
    interpret mi: Metric-space Mi i di i
      by(simp add: Md-metric hi)
    show mi.MCauchy (λn. xn n i)
      unfolding mi.MCauchy-def
    proof safe
      show xn n i ∈ Mi i for n
        using h hi by(auto simp: Product-metric.MCauchy-def)
    next
      fix ε
      assume he:(0::real) < ε
      then have 0 < ε ∗ r^(f i) using r by auto
      then obtain N where N:
        ⋀n m. n≥N ⟹ m≥N ⟹ product-dist (xn n) (xn m) < ε ∗ r^(f i)
        using h Product-metric.MCauchy-def by fastforce
      show ∃ N. ∀ n n'. N ≤ n ⟶ N ≤ n' ⟶ di i (xn n i) (xn n' i) < ε
      proof(safe intro!: exI[where x=N])
        fix n m
        assume n:n ≥ N m ≥ N
        have di i (xn n i) (xn m i) ≤ (1 / r) ^ (f i) ∗ product-dist (xn n) (xn m)
          by(rule product-dist-geq) (use h[simplified Product-metric.MCauchy-def]
hi gf-comp-id[of i] in auto)
        also have ... < ε
         using N[OF n] by (simp add: mult-imp-div-pos-less power-one-over r(1))
        finally show di i (xn n i) (xn m i) < ε .
      qed
    qed
  qed
  hence ∀ i∈I. ∃ x. limitin (Metric-space.mtopology (Mi i) (di i)) (λn. xn n i) x
sequentially
    using Md-metric Metric-space.mcomplete-alt assms by blast
   then obtain x where hx:⋀i. i ∈ I ⟹ limitin (Metric-space.mtopology (Mi
i) (di i)) (λn. xn n i) (x i) sequentially
    by metis
  hence hx':(λi∈I. x i) ∈ (Π_E i∈I. Mi i)
    by (simp add: Md-metric Metric-space.limit-metric-sequentially)
  thus ∃ x. limitin Product-metric.mtopology xn x sequentially
   using h by(auto intro!: exI[where x=λi∈I. x i] limitin-M-iff-limitin-Mi[THEN
iffD2,of xn] simp: Product-metric.MCauchy-def hx) blast
 qed
qed

end

lemma product-metricI:
 assumes 0 < r r < 1 countable I ⋀i. i ∈ I ⟹ Metric-space (Mi i) (di i)
   and ⋀i x y. 0 ≤ di i x y ⋀i x y. di i x y ≤ K 0 < K
  shows Product-metric r I (to-nat-on I) (from-nat-into I) Mi di K
 using from-nat-into-to-nat-on-product-metric-pair[OF assms(3)] assms
```

by(*simp add*: *Product-metric-def Metric-space-def*)

lemma *product-metric-natI*:
assumes *0 < r r < 1* ⋀*n. Metric-space* (*Mi n*) (*di n*)
and ⋀*i x y. 0 ≤ di i x y* ⋀*i x y. di i x y ≤ K 0 < K*
shows *Product-metric r UNIV id id Mi di K*
using *assms* **by**(*auto simp*: *Product-metric-def*)

end

2 Abstract Polish Spaces

theory *Abstract-Metrizable-Topology*
imports *Set-Based-Metric-Product*
begin

2.1 Polish Spaces

definition *Polish-space X* ≡ *completely-metrizable-space X* ∧ *separable-space X*

lemma(**in** *Metric-space*) *Polish-space-mtopology*:
assumes *mcomplete separable-space mtopology*
shows *Polish-space mtopology*
by (*simp add*: *assms completely-metrizable-space-mtopology Polish-space-def*)

lemma
assumes *Polish-space X*
shows *Polish-space-imp-completely-metrizable-space*: *completely-metrizable-space X*
and *Polish-space-imp-metrizable-space*: *metrizable-space X*
and *Polish-space-imp-second-countable*: *second-countable X*
and *Polish-space-imp-separable-space*: *separable-space X*
using *assms* **by**(*auto simp*: *completely-metrizable-imp-metrizable-space Polish-space-def metrizable-space-separable-iff-second-countable*)

lemma *Polish-space-closedin*:
assumes *Polish-space X closedin X A*
shows *Polish-space* (*subtopology X A*)
using *assms* **by**(*auto simp*: *completely-metrizable-imp-metrizable-space Polish-space-def completely-metrizable-space-closedin second-countable-subtopology metrizable-space-separable-iff-second-countal*

lemma *Polish-space-gdelta-in*:
assumes *Polish-space X gdelta-in X A*
shows *Polish-space* (*subtopology X A*)
using *assms* **by**(*auto simp*: *completely-metrizable-imp-metrizable-space Polish-space-def completely-metrizable-space-gdelta-in second-countable-subtopology metrizable-space-separable-iff-second-counta*

corollary *Polish-space-openin*:
assumes *Polish-space X openin X A*

shows *Polish-space* (*subtopology X A*)
by (*simp add*: *open-imp-gdelta-in assms Polish-space-gdelta-in*)

lemma *homeomorphic-Polish-space-aux*:
assumes *Polish-space X X homeomorphic-space Y*
shows *Polish-space Y*
using *assms* **by**(*simp add*: *homeomorphic-completely-metrizable-space-aux homeomorphic-separable-space Polish-space-def*)

corollary *homeomorphic-Polish-space*:
assumes *X homeomorphic-space Y*
shows *Polish-space X* ⟷ *Polish-space Y*
by (*meson assms homeomorphic-Polish-space-aux homeomorphic-space-sym*)

lemma *Polish-space-euclidean*[*simp*]: *Polish-space* (*euclidean* :: (′*a* :: *polish-space*) *topology*)
by (*simp add*: *completely-metrizable-space-euclidean Polish-space-def second-countable-imp-separable-space*)

lemma *Polish-space-countable*[*simp*]:
Polish-space (*euclidean* :: ′*a* :: {*countable*,*discrete-topology*} *topology*)
proof −
interpret *discrete-metric UNIV* :: ′*a set* **.**
have [*simp*]:*euclidean* = *disc.mtopology*
by (*metis discrete-topology-class.open-discrete discrete-topology-unique istopology-open mtopology-discrete-metric topology-inverse′ topspace-euclidean*)
show *?thesis*
by(*auto simp*: *Polish-space-def intro*!: *disc.completely-metrizable-space-mtopology mcomplete-discrete-metric countable-space-separable-space*)
qed

lemma *Polish-space-discrete-topology*: *Polish-space* (*discrete-topology I*) ⟷ *countable I*
by (*simp add*: *completely-metrizable-space-discrete-topology Polish-space-def separable-space-discrete-topology*)

lemma *Polish-space-prod*:
assumes *Polish-space X* **and** *Polish-space Y*
shows *Polish-space* (*prod-topology X Y*)
using *assms* **by** (*simp add*: *completely-metrizable-space-prod-topology Polish-space-def separable-space-prod*)

lemma *Polish-space-product*:
assumes *countable I* **and** $\bigwedge i.\ i \in I \Longrightarrow$ *Polish-space* (*S i*)
shows *Polish-space* (*product-topology S I*)
using *assms* **by**(*auto simp*: *separable-countable-product Polish-space-def completely-metrizable-space-product-topology*)

lemma(**in** *Product-metric*) *Polish-spaceI*:
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *separable-space* (*Metric-space.mtopology* (*Mi i*) (*di i*))

and $\bigwedge i.\ i \in I \Longrightarrow$ *Metric-space.mcomplete* (*Mi i*) (*di i*)
shows *Polish-space Product-metric.mtopology*
using *assms I* **by**(*auto simp*: *Polish-space-def Product-metric-mtopology-eq*[*symmetric*] *completely-metrizable-space-product-topology intro*!: *separable-countable-product Metric-space.completely-metrizable-space-mtopology Md-metric*)

lemma(**in** *Sum-metric*) *Polish-spaceI*:
assumes *countable I*
and $\bigwedge i.\ i \in I \Longrightarrow$ *separable-space* (*Metric-space.mtopology* (*Mi i*) (*di i*))
and $\bigwedge i.\ i \in I \Longrightarrow$ *Metric-space.mcomplete* (*Mi i*) (*di i*)
shows *Polish-space Sum-metric.mtopology*
by(*auto simp*: *Polish-space-def intro*!: *separable-Mi-separable-M assms mcomplete-Mi-mcomplete-M Sum-metric.completely-metrizable-space-mtopology*)

lemma *compact-metrizable-imp-Polish-space*:
assumes *metrizable-space X compact-space X*
shows *Polish-space X*
proof −
obtain *d* **where** *Metric-space* (*topspace X*) *d Metric-space.mtopology* (*topspace X*) *d* = *X*
by (*metis assms*(*1*) *Metric-space.topspace-mtopology metrizable-space-def*)
thus *?thesis*
by (*metis Metric-space.compact-space-imp-separable assms*(*1*) *assms*(*2*) *compact-imp-locally-compact-space locally-compact-imp-completely-metrizable-space Polish-space-def*)
qed

2.2 Extended Reals and Non-Negative Extended Reals

lemma *Polish-space-ereal*:*Polish-space* (*euclidean* :: *ereal topology*)
proof(*rule homeomorphic-Polish-space-aux*)
show *Polish-space* (*subtopology euclideanreal* $\{-1..1\}$)
by (*auto intro*!: *Polish-space-closedin*)
next
define *f* :: *real* ⇒ *ereal*
where $f \equiv (\lambda r.\ \textit{if } r = -1 \textit{ then } -\infty \textit{ else if } r = 1 \textit{ then } \infty \textit{ else if } r \leq 0 \textit{ then ereal } (1 - (1 / (1 + r))) \textit{ else ereal } ((1 / (1 - r)) - 1))$
define *g* :: *ereal* ⇒ *real*
where $g \equiv (\lambda r.\ \textit{if } r \leq 0 \textit{ then real-of-ereal } (\textit{inverse } (1 - r)) - 1 \textit{ else } 1 - \textit{real-of-ereal } (\textit{inverse } (1 + r)))$
show *top-of-set* $\{-1..1::real\}$ *homeomorphic-space* (*euclidean* :: *ereal topology*)
unfolding *homeomorphic-space-def homeomorphic-maps-def continuous-map-iff-continuous*
proof(*safe intro*!: *exI*[**where** *x*=*f*] *exI*[**where** *x*=*g*])
show *continuous-on* $\{-1..1\}$ *f*
unfolding *continuous-on-eq-continuous-within*
proof *safe*
fix *x* :: *real*
assume $x \in \{-1..1\}$
then consider $x = -1 \mid -1 < x\ x < 0 \mid x = 0 \mid 0 < x\ x < 1 \mid x = 1$

by *fastforce*
then show *continuous* (*at x within* {− *1*..*1*}) *f*
proof *cases*
show − *1* < *x* ⟹ *x* < *0* ⟹ *?thesis*
by(*simp add*: *at-within-Icc-at*, *intro isCont-cong*[**where** *f*=*λr*. *ereal* (*1* − (*1* / (*1* + *r*))) **and** *g*=*f*,*THEN iffD1*,*OF* - *continuous-on-interior*[*of* {−*1*<..<*0*}]]) (*auto simp*: *at-within-Icc-at eventually-nhds f-def intro*!: *exI*[**where** *x*={−*1*<..<*0*}] *continuous-on-divide continuous-on-ereal continuous-on-diff continuous-on-add*)
next
have ∗:*isCont* (*λr*. *if r* ≤ *0 then ereal* (*1* − (*1* / (*1* + *r*))) *else ereal* ((*1* / (*1* − *r*)) − *1*)) *0*
by(*rule isCont-If-ge*) (*auto simp add*: *continuous-within intro*!: *continuous-on-Icc-at-leftD*[**where** *a*=− (*1* / *2*) **and** *b*=*0* **and** *f*=*λr*::*real*. *1* − (*1* / (*1* + *r*)),*simplified*] *continuous-on-Icc-at-rightD*[**where** *a*=*0* **and** *b*=*1* / *2* **and** *f*=*λr*::*real*. (*1* / (*1* − *r*)) − *1*,*simplified*] *continuous-on-diff continuous-on-divide continuous-on-add*)
show *?thesis* **if** *x*:*x* = *0*
unfolding *x at-within-Icc-at*[*of* − *1* :: *real 0 1*,*simplified*]
by(*rule isCont-cong*[*THEN iffD1*,*OF* - ∗]) (*auto simp*: *eventually-nhds f-def intro*!: *exI*[**where** *x*={−*1* / *2*<..<*1*/*2*}])
next
show *0* < *x* ⟹ *x* < *1* ⟹ *?thesis*
by(*simp add*: *at-within-Icc-at*, *intro isCont-cong*[**where** *f*=*λr*. *ereal* ((*1* / (*1* − *r*)) − *1*) **and** *g*=*f*,*THEN iffD1*,*OF* - *continuous-on-interior*[*of* {*0*<..<*1*}]]) (*auto simp*: *at-within-Icc-at eventually-nhds f-def intro*!: *exI*[**where** *x*={*0*<..<*1*}] *continuous-on-divide continuous-on-ereal continuous-on-diff continuous-on-add*)
next
show *?thesis* **if** *x*:*x* = −*1*
unfolding *x at-within-Icc-at-right*[**where** *a*=− *1* :: *real* **and** *b*=*1*,*simplified*] *continuous-within ereal-tendsto-simps*(*2*)[*symmetric*]
proof(*subst tendsto-cong*)
show ∀$_F$ *r in at-right* (*ereal* (− *1*)). (*f* ∘ *real-of-ereal*) *r* = *1* − (*1* / (*1* + *r*))
unfolding *eventually-at-right*[*of ereal* (− *1*) *0*,*simplified*]
proof(*safe intro*!: *exI*[**where** *x*=*ereal* (− *1* / *2*)])
fix *y*
assume *ereal* (− *1*) < *y y* < *ereal* (− *1* / *2*)
then obtain *y*′ **where** *y*′:*y* = *ereal y*′ − *1* < *y*′ *y*′ < − *1* / *2*
by (*metis ereal-real*′ *less-ereal.simps*(*1*) *not-inftyI*)
show (*f* ∘ *real-of-ereal*) *y* = *1* − *1* / (*1* + *y*)
using *y*′(*2*,*3*) **by**(*auto simp add*: *y*′(*1*) *f-def one-ereal-def*)
qed *simp*
next
have ((*λr*. *1* − *1* / (*1* + *r*)) ⟶ − ∞) (*at-right* (*ereal* (− *1*)))
unfolding *tendsto-MInfty*
proof *safe*
fix *r* :: *real*
consider *r* ≥ *1* | *r* < *1*
by *argo*

```
          then show ∀F x in at-right (ereal (− 1)). 1 − 1 / (1 + x) < ereal r
          proof cases
            case [arith]:1
            show ?thesis
              unfolding eventually-at-right[of ereal (− 1) 0,simplified]
            proof(safe intro!: exI[where x=0])
              fix y
              assume ereal (− 1) < y y < 0
              then obtain y′ where y′:y = ereal y′ − 1 < y′ y′ < 0
                using not-inftyI by force
              then have ∗:1 − 1 / (1 + y) < 1
                by (simp add: one-ereal-def)
              show 1 − 1 / (1 + y) < ereal r
                by(rule order.strict-trans2[OF ∗]) (use 1 in auto)
            qed simp
          next
            case 2
            show ?thesis
              unfolding eventually-at-right[of ereal (− 1) 0,simplified]
            proof(safe intro!: exI[where x=ereal (1 / (1 − r) − 1)])
              fix y
              assume  ereal (− 1) < y y < ereal (1 / (1 − r) − 1)
             then obtain y′ where y′:y = ereal y′ − 1 < y′ y′ < 1 / (1 − r) − 1
                by (metis ereal-less-eq(3) ereal-real′ linorder-not-le not-inftyI)
              have 1 − 1 / (1 + y) = ereal (1 − 1 / (1 + y′))
                by (metis ereal-divide ereal-minus(1) one-ereal-def order-less-irrefl
plus-ereal.simps(1) real-0-less-add-iff y′(1) y′(2))
              also have ... < ereal r
              proof −
                have 1 + y′ < 1 / (1 − r)
                  using y′ by simp
                hence 1 − r < 1 / (1 + y′)
                  using y′ 2 by (simp add: less-divide-eq mult.commute)
                thus ?thesis
                  by simp
              qed
              finally show 1 − 1 / (1 + y) < ereal r .
            qed(use 2 in auto)
          qed
        qed
        thus ((λr. 1 − 1 / (1 + r)) ⟶ f (− 1)) (at-right (ereal (− 1)))
          by(simp add: f-def)
      qed
    next
      show ?thesis if x:x = 1
       unfolding x at-within-Icc-at-left[where a=− 1 :: real and b=1,simplified]
continuous-within ereal-tendsto-simps(1)[symmetric]
      proof(subst tendsto-cong)
        show ∀F r in at-left (ereal 1). (f ∘ real-of-ereal) r = (1 / (1 − r)) − 1
```

unfolding *eventually-at-left*[*of 0 ereal 1*,*simplified*]
proof(*safe intro*!: *exI*[**where** *x*=*ereal* (*1* / *2*)])
fix *y*
assume *ereal* (*1* / *2*) < *y* *y* < *ereal* *1*
then obtain *y′* **where** *y′*:*y* = *ereal* *y′* *1* / *2* < *y′* *y′* < *1*
using *ereal-less-ereal-Ex* **by** *force*
show (*f* ∘ *real-of-ereal*) *y* = *1* / (*1* − *y*) − *1*
using *y′*(*2*,*3*) **by**(*auto simp add*: *y′*(*1*) *f-def one-ereal-def*)
qed *simp*
next
have ((λ*r*. (*1* / (*1* − *r*)) − *1*) ⟶ ∞) (*at-left* (*ereal* *1*))
unfolding *tendsto-PInfty*
proof *safe*
fix *r* :: *real*
consider *r* ≤ − *1* | − *1* < *r*
by *argo*
then show $\forall_F$ *x in at-left* (*ereal* *1*). (*1* / (*1* − *x*)) − *1* > *ereal* *r*
proof *cases*
case [*arith*]:*1*
show *?thesis*
unfolding *eventually-at-left*[*of 0 ereal 1*,*simplified*]
proof(*safe intro*!: *exI*[**where** *x*=*0*])
fix *y*
assume *0* < *y* *y* < *ereal* *1*
then obtain *y′* **where** *y′*:*y* = *ereal* *y′* *0* < *y′* *y′* < *1*
using *not-inftyI* **by** *force*
then have ∗:(*1* / (*1* − *y*)) − *1* > *ereal* (− *1*)
by (*simp add*: *one-ereal-def*)
show *ereal* *r* < *1* / (*1* − *y*) − *1*
by(*rule order.strict-trans1*[*OF* - ∗]) (*use* *1* **in** *auto*)
qed *simp*
next
case *2*
show *?thesis*
unfolding *eventually-at-left*[*of 0 ereal 1*,*simplified*]
proof(*safe intro*!: *exI*[**where** *x*=*ereal* (*1* − *1* / (*1* + *r*))])
fix *y*
assume *ereal* (*1* − *1* / (*1* + *r*)) < *y* *y* < *ereal* *1*
then obtain *y′* **where** *y′*:*y* = *ereal* *y′* *1* − *1* / (*1* + *r*) < *y′* *y′* < *1*
by (*metis ereal-less-eq*(*3*) *ereal-real′ linorder-not-le not-inftyI*)
have *ereal* *r* < *ereal* (*1* / (*1* − *y′*) − *1*)
proof −
have *1* − *y′* < *1* / (*r* + *1*)
using *y′*(*2*) **by** *argo*
hence *r* + *1* < *1* / (*1* − *y′*)
using *y′* *2* **by** (*simp add*: *less-divide-eq mult.commute*)
thus *?thesis*
by *simp*
qed

```
          also have ... = 1 / (1 − y) − 1
        by (metis diff-gt-0-iff-gt ereal-divide ereal-minus(1) less-numeral-extra(3)
one-ereal-def y′(1) y′(3))
          finally show ereal r < 1 / (1 − y) − 1 .
         qed(use 2 in auto)
        qed
       qed
       thus ((λr. 1 / (1 − r) − 1) ⟶ f 1) (at-left (ereal 1))
        by(simp add: f-def)
      qed
     qed
    qed
  next
   show continuous-map euclidean (top-of-set {− 1..1}) g
   proof(safe intro!: continuous-map-into-subtopology)
    show continuous-map euclidean euclideanreal g
    unfolding Abstract-Topology.continuous-map-iff-continuous2 continuous-on-eq-continuous-within
    proof safe
     fix x :: ereal
     consider x = − ∞ | − ∞ < x x < 0 | x = 0 | 0 < x x < ∞ | x = ∞
      by fastforce
     then show isCont g x
     proof cases
      assume x:− ∞ < x x < 0
      then obtain x′ where x′:x = ereal x′ x′ < 0
       by (metis ereal-infty-less(2) ereal-less-ereal-Ex zero-ereal-def)
      show ?thesis
      proof(subst isCont-cong)
       have [simp]:isCont ((−) 1) x
       proof −
        have ∗:isCont (λx. ereal (real-of-ereal 1 − real-of-ereal x)) x
        using x′ by(auto simp add: continuous-at-iff-ereal[symmetric,simplified
comp-def] intro!: continuous-diff continuous-at-of-ereal)
        have ∗∗: ereal (x′ − 1) < x ⟹ x < 0 ⟹ ereal (1 − real-of-ereal x)
= ereal 1 − x for x
            by (metis ereal-minus(1) less-ereal.simps(2) less-ereal.simps(3)
real-of-ereal.elims)
        show ?thesis
         apply(rule isCont-cong[THEN iffD1,OF - ∗])
          using x′(2) ∗∗ by(auto simp: eventually-nhds x′(1) one-ereal-def
intro!: exI[where x={x−1<..<0}])
       qed
       have ∗:abs (1 − x) ≠ ∞  1 − x ≠ 0
        using x′(2) by(auto simp add: x′(1) one-ereal-def)
       show isCont (λr. real-of-ereal (inverse (1 − r)) − 1) x
       using x ∗ by(auto intro!: continuous-diff continuous-divide isCont-o2[OF
- continuous-at-of-ereal])
      next
       show ∀F x in nhds x. g x = real-of-ereal (inverse (1 − x)) − 1
```

```
            using x(2) by(auto simp: eventually-nhds x'(1) g-def one-ereal-def
intro!: exI[where x={x−1<..<0}])
        qed
      next
        assume x:∞ > x x > 0
        then obtain x' where x':x = ereal x' x' > 0
          by (metis ereal-less(2) less-ereal.elims(2) less-ereal.simps(2))
        show ?thesis
        proof(subst isCont-cong)
          have [simp]: isCont ((+) 1) x
          proof −
            have ∗:isCont (λx. ereal (real-of-ereal 1 + real-of-ereal x)) x
            using x' by(auto simp add: continuous-at-iff-ereal[symmetric,simplified
comp-def] intro!: continuous-add continuous-at-of-ereal)
            have ∗∗: 0 < x =⇒ x < ereal (x' + 1) =⇒ ereal (1 + real-of-ereal x)
= ereal 1 + x for x
              using ereal-less-ereal-Ex by auto
            show ?thesis
              apply(rule isCont-cong[THEN iffD1,OF - ∗])
                using x'(2) ∗∗ by(auto simp: eventually-nhds x'(1) one-ereal-def
intro!: exI[where x={0<..<x + 1}])
          qed
          have real-of-ereal (1 + x) ≠ 0
            using x' by auto
          thus isCont (λr. 1 − real-of-ereal (inverse (1 + r))) x
           using x by(auto intro!: continuous-diff continuous-divide isCont-o2[OF
- continuous-at-of-ereal])
        next
          show ∀F x in nhds x. g x = 1 − real-of-ereal (inverse (1 + x))
              using x(2) by(auto simp: eventually-nhds x'(1) g-def one-ereal-def
intro!: exI[where x={0<..<x+1}])
        qed
      next
        show isCont g x if x:x = − ∞
          unfolding x
        proof(safe intro!: continuous-at-sequentiallyI)
          fix u :: nat ⇒ ereal
          assume u:u ⟶ − ∞
          show (λn. g (u n)) ⟶ g (− ∞)
            unfolding LIMSEQ-def
          proof safe
            fix r :: real
            assume r[arith]: r > 0
            obtain no where no: ⋀n. n ≥ no =⇒ u n < ereal (min (1 − 1 / r)
0)
              using u unfolding tendsto-MInfty eventually-sequentially by blast
            show ∃ no. ∀ n≥no. dist (g (u n)) (g (− ∞)) < r
            proof(safe intro!: exI[where x=no])
              fix n
```

```
            assume n:n ≥ no
            have r0:1 − min (ereal (1 − 1 / r)) (ereal 0) > 0
              by (simp add: ereal-diff-gr0 min.strict-coboundedI2)
            have u1:1 − u n > 0
                  by (metis ereal-0-less-1 ereal-diff-gr0 ereal-min linorder-not-le
min.strict-coboundedI2 n no order-le-less-trans order-less-not-sym zero-ereal-def)
            have real-of-ereal (inverse (1 − u n)) < r
            proof −
             have real-of-ereal (inverse (1 − u n)) < real-of-ereal (inverse (1 −
ereal (min (1 − 1 / r) 0)))
              proof(safe intro!: ereal-less-real-iff[THEN iffD2])
               have ereal (real-of-ereal (inverse (1 − u n))) = inverse (1 − u n)
                 by(rule ereal-real') (use no[OF n] u1 in auto)
                also have ... < inverse (1 − ereal (min (1 − 1 / r) 0))
                 apply(rule ereal-inverse-antimono-strict)
            using no[OF n] apply(simp add: ereal-diff-positive min.coboundedI2)
                by (metis (no-types, lifting) no[OF n] ereal-add-uminus-conv-diff
ereal-eq-minus-iff ereal-less-minus-iff ereal-minus-less-minus ereal-times(1) ereal-times(3))
                finally show ereal (real-of-ereal (inverse (1 − u n))) < inverse
(1 − ereal (min (1 − 1 / r) 0)) .
              qed(use r0 in auto)
              also have ... ≤ r
                by(cases r ≥ 1) (auto simp add: real-of-ereal-minus)
              finally show real-of-ereal (inverse (1 − u n)) < r .
            qed
            thus dist (g (u n)) (g (− ∞)) < r
              using u1 no[OF n] by(auto simp: g-def zero-ereal-def dist-real-def)
          qed
        qed
      qed
    next
      show isCont g x if x:x = ∞
        unfolding x
      proof(safe intro!: continuous-at-sequentiallyI)
        fix u :: nat ⇒ ereal
        assume u:u ⟶ ∞
        show (λn. g (u n)) ⟶ g ∞
          unfolding LIMSEQ-def
        proof safe
          fix r :: real
          assume r[arith]: r > 0
          obtain no where no: ⋀n. n ≥ no ⟹ u n > ereal (max (1 / r − 1)
0)
            using u unfolding tendsto-PInfty eventually-sequentially by blast
          show ∃ no. ∀ n≥no. dist (g (u n)) (g ∞) < r
          proof(safe intro!: exI[where x=no])
            fix n
            assume n:n ≥ no
            have u0: 1 + u n > 0
```

```
                  using no[OF n] by simp (metis add-nonneg-pos zero-ereal-def
zero-less-one-ereal)
              have |− real-of-ereal (inverse (1 + u n))| < r
              proof −
                have |− real-of-ereal (inverse (1 + u n))| < |− (inverse (1 + max
(1 / r − 1) 0))|
                  unfolding abs-real-of-ereal abs-minus
                proof(safe intro!: real-less-ereal-iff[THEN iffD2])
                  have |inverse (1 + u n)| < inverse (1 + ereal (max (1 / r − 1)
0))
                          using no[OF n] u0 by (simp add: ereal-add-strict-mono
ereal-inverse-antimono-strict inverse-ereal-ge0I le-max-iff-disj order-less-imp-le u0)
                  also have ... = ereal |inverse (1 + max (1 / r − 1) 0)|
                 by(auto simp: abs-ereal.simps(1)[symmetric] ereal-max[symmetric]
simp del: abs-ereal.simps(1) ereal-max)
                  finally show |inverse (1 + u n)| < ereal |inverse (1 + max (1 /
r − 1) 0)| .
                qed auto
                also have ... = inverse (1 + max (1 / r − 1) 0)
                  by auto
                also have ... ≤ r
                  by(cases r ≤ 1) auto
                finally show ?thesis .
              qed
              thus dist (g (u n)) (g ∞) < r
                using no[OF n] by(auto simp: g-def dist-real-def zero-ereal-def)
            qed
          qed
        qed
      next
        show isCont g x if x:x = 0
          unfolding x g-def
        proof(safe intro!: isCont-If-ge)
          have ((λx. real-of-ereal (1 − x)) ⟶ 1) (at-left 0)
          proof(subst tendsto-cong)
            show ((λx. 1 − real-of-ereal x) ⟶ 1) (at-left 0)
               by(auto intro!: tendsto-diff[where a=1 and b=0,simplified] simp:
zero-ereal-def)
          next
            show ∀F x in at-left 0. real-of-ereal (1 − x) = 1 − real-of-ereal x
          by(auto simp: eventually-at-left[where y=− 1 and x=0::ereal,simplified]
real-of-ereal-minus ereal-uminus-eq-reorder  intro!: exI[where x=−1])
          qed
          thus continuous (at-left 0) (λx. real-of-ereal (inverse (1 − x)) − 1)
            unfolding continuous-within
               by (auto intro!: tendsto-diff[where a = 1 and b=1,simplified]
tendsto-divide[where a=1 and b=1,simplified])
        next
          have ((λx. real-of-ereal (1 + x)) ⟶ 1) (at-right 0)
```

proof(*subst tendsto-cong*)
show $((\lambda x.\ 1 + \textit{real-of-ereal}\ x) \longrightarrow 1)\ (\textit{at-right}\ 0)$
by(*auto intro*!: *tendsto-add*[**where** *a=1* **and** *b=0*,*simplified*] *simp*: *zero-ereal-def*)
next
show $\forall_F\ x\ \textit{in at-right}\ 0.\ \textit{real-of-ereal}\ (1 + x) = 1 + \textit{real-of-ereal}\ x$
by(*auto simp*: *eventually-at-right*[**where** *y=1* **and** *x=0::ereal*,*simplified*] *real-of-ereal-add ereal-uminus-eq-reorder intro*!: *exI*[**where** *x=1*])
qed
thus $((\lambda x.\ 1 - \textit{real-of-ereal}\ (\textit{inverse}\ (1 + x))) \longrightarrow \textit{real-of-ereal}\ (\textit{inverse}\ (1 - 0)) - 1)\ (\textit{at-right}\ 0)$
by (*auto intro*!: *tendsto-diff*[**where** *a = 1* **and** *b=1*,*simplified*] *tendsto-divide*[**where** *a=1* **and** *b=1*,*simplified*])
qed
qed
qed
next
fix *x* :: *ereal*
consider $x = -\infty \mid -\infty < x\ x \le 0 \mid 0 < x\ x < \infty \mid x = \infty$
by *fastforce*
then show $g\ x \in \{-1..1\}$
proof *cases*
assume $-\infty < x\ x \le 0$
then obtain x' **where** $x = \textit{ereal}\ x'\ x' \le 0$
by (*metis dual-order.refl ereal-less-ereal-Ex order-less-le zero-ereal-def*)
then show *?thesis*
by(*auto simp*: *g-def real-of-ereal-minus intro*!: *pos-divide-le-eq*[*THEN iffD2*])
next
assume $0 < x\ x < \infty$
then obtain x' **where** $x = \textit{ereal}\ x'\ x' > 0$
by (*metis ereal-less*(*2*) *less-ereal.elims*(*2*) *order-less-le*)
then show *?thesis*
by(*auto simp*: *g-def real-of-ereal-add inverse-eq-divide intro*!: *pos-divide-le-eq*[*THEN iffD2*])
qed(*auto simp*: *g-def*)
qed
next
fix *x* :: *ereal*
consider $x = -\infty \mid -\infty < x\ x \le 0 \mid 0 < x\ x < \infty \mid x = \infty$
by *fastforce*
then show $f\ (g\ x) = x$
proof *cases*
assume $-\infty < x\ x \le 0$
then obtain x' **where** $x'{:}x = \textit{ereal}\ x'\ x' \le 0$
by (*metis dual-order.refl ereal-less-ereal-Ex order-less-le zero-ereal-def*)
then have [*arith*]:$1\ /\ (1 - x') - 1 \le 0$
by *simp*
show *?thesis*
using x' **by**(*auto simp*: *g-def real-of-ereal-minus f-def*)

next
assume *0 < x x < ∞*
then obtain *x′* **where** *x′:x = ereal x′ x′ > 0*
by (*metis ereal-less*(*2*) *less-ereal.elims*(*2*) *order-less-le*)
hence [*arith*]: *1 − 1 / (x′ + 1) ≥ 0*
by *simp*
show *?thesis*
using *x′* **by**(*simp add*: *g-def inverse-eq-divide f-def*)
qed(*auto simp*: *f-def g-def*)
next
fix *x* :: *real*
assume *x ∈ topspace* (*top-of-set* {*− 1..1*})
then consider *x = − 1* | *− 1 < x x ≤ 0* | *0 < x x < 1* | *x = 1*
by *fastforce*
then show *g* (*f x*) *= x*
by *cases* (*auto simp*: *f-def g-def real-of-ereal-minus real-of-ereal-add*)
qed
qed

corollary *Polish-space-ennreal*:*Polish-space* (*euclidean* :: *ennreal topology*)
proof(*rule homeomorphic-Polish-space-aux*)
show *Polish-space* (*top-of-set* {*0*::*ereal*..})
using *Polish-space-closedin Polish-space-ereal* **by** *fastforce*
next
show *top-of-set* {*0*::*ereal*..} *homeomorphic-space* (*euclidean* :: *ennreal topology*)
by(*auto intro*!: *exI*[**where** *x*=*e2ennreal*] *exI*[**where** *x*=*enn2ereal*] *simp*: *homeomorphic-space-def homeomorphic-maps-def enn2ereal-e2ennreal continuous-on-e2ennreal continuous-map-in-subtopology continuous-on-enn2ereal image-subset-iff*)
qed

2.3 Continuous Embddings

abbreviation *Hilbert-cube-topology* :: (*nat* ⇒ *real*) *topology* **where**
Hilbert-cube-topology ≡ (*product-topology* (*λn. top-of-set* {*0..1*}) *UNIV*)

lemma *topspace-Hilbert-cube*: *topspace Hilbert-cube-topology* = (Π_E *x*∈*UNIV*. {*0..1*})
by *simp*

lemma *Polish-space-Hilbert-cube*: *Polish-space Hilbert-cube-topology*
by(*auto intro*!: *Polish-space-closedin Polish-space-product*)

abbreviation *Cantor-space-topology* :: (*nat* ⇒ *real*) *topology* **where**
Cantor-space-topology ≡ (*product-topology* (*λn. top-of-set* {*0,1*}) *UNIV*)

lemma *topspace-Cantor-space*:
topspace Cantor-space-topology = (Π_E *x*∈*UNIV*. {*0,1*})
by *simp*

lemma *Polish-space-Cantor-space*: *Polish-space Cantor-space-topology*

by(*auto intro*!: *Polish-space-closedin Polish-space-product*)

corollary *completely-metrizable-space-homeo-image-gdelta-in*:
assumes *completely-metrizable-space X completely-metrizable-space Y B* $\subseteq$ *topspace Y X homeomorphic-space subtopology Y B*
shows *gdelta-in Y B*
using *assms completely-metrizable-space-eq-gdelta-in homeomorphic-completely-metrizable-space*
by *blast*

2.3.1 Embedding into Hilbert Cube

lemma *embedding-into-Hilbert-cube*:
assumes *metrizable-space X separable-space X*
shows $\exists$ *A* $\subseteq$ *topspace Hilbert-cube-topology. X homeomorphic-space* (*subtopology Hilbert-cube-topology A*)
proof −
consider *X* = *trivial-topology* | *topspace X* $\neq$ {} **by** *auto*
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by(*auto intro*!: *exI*[**where** *x*={}] *simp*: *homeomorphic-empty-space-eq*)
next
case *S-ne*:*2*
then obtain *U* **where** *U*:*countable U dense-in X U U* $\neq$ {}
using *assms*(*2*) **by**(*auto simp*: *separable-space-def2 dense-in-nonempty*)
obtain *xn* **where** *xn*:$\bigwedge$*n*::*nat. xn n* $\in$ *U U* = *range xn*
by (*metis U*(*1*) *U*(*3*) *from-nat-into range-from-nat-into*)
then have *xns*:*xn n* $\in$ *topspace X* **for** *n*
using *dense-in-subset*[*OF U*(*2*)] **by** *auto*
obtain *d′* **where** *d′*:*Metric-space* (*topspace X*) *d′ Metric-space.mtopology* (*topspace X*) *d′* = *X*
by (*metis Metric-space.topspace-mtopology assms*(*1*) *metrizable-space-def*)
interpret *ms′*: *Metric-space topspace X d′* **by** *fact*
define *d* **where** *d* = *ms′.capped-dist* (*1*/*2*)
have *d*: *Metric-space.mtopology* (*topspace X*) *d* = *X* $\bigwedge$*x y. d x y* < *1*
by(*simp add*: *d-def ms′.mtopology-capped-metric d′*) (*simp add*: *d-def ms′.capped-dist-def*)
interpret *ms*: *Metric-space topspace X d*
by (*simp add*: *d-def ms′.capped-dist*)
define *f* **where** *f* ≡ (λ*x n. d x* (*xn n*))
have *f-inj*:*inj-on f* (*topspace X*)
proof
fix *x y*
assume *xy*:*x* $\in$ *topspace X y* $\in$ *topspace X f x* = *f y*
then have $\bigwedge$*n. d x* (*xn n*) = *d y* (*xn n*) **by**(*auto simp*: *f-def dest*: *fun-cong*)
hence *d2*:*d x y* ≤ *2* ∗ *d x* (*xn n*) **for** *n*
using *ms.triangle*[*OF xy*(*1*) - *xy*(*2*),*of xn n*,*simplified ms.commute*[*of xn n y*]] *dense-in-subset*[*OF U*(*2*)] *xn*(*1*)[*of n*]
by *auto*

    have *d x y* < *ε* **if** *ε* > *0* **for** *ε*
    proof −
      have *0* < *ε* / *2* **using** *that* **by** *simp*
      then obtain *n* **where** *d x* (*xn n*) < *ε* / *2*
        using *ms.mdense-def2*[*of U*,*simplified d*(*1*)] *U*(*2*) *xy*(*1*) *xn*(*2*) **by** *blast*
      with *d2*[*of n*] **show** *?thesis* **by** *simp*
    qed
    hence *d x y* = *0*
      by (*metis ms.nonneg*[*of x y*] *dual-order.irrefl order-neq-le-trans*)
    thus *x* = *y*
      using *xy* **by** *simp*
  qed
  have *f-img*: *f* ‘ *topspace X* ⊆ *topspace Hilbert-cube-topology*
   using *d*(*2*) *ms.nonneg* **by**(*auto simp*: *topspace-Hilbert-cube f-def less-le-not-le*)
  have *f-cont*: *continuous-map X Hilbert-cube-topology f*
  unfolding *continuous-map-componentwise-UNIV f-def continuous-map-in-subtopology*
  proof *safe*
    show *continuous-map X euclideanreal* (*λx. d x* (*xn k*)) **for** *k*
    proof(*rule continuous-map-eq*[*of - - mdist-set ms.Self* {*xn k*}])
      show *continuous-map X euclideanreal* (*mdist-set ms.Self* {*xn k*})
      by (*metis d*(*1*) *mdist-set-uniformly-continuous ms.mdist-Self ms.mspace-Self mtopology-of-def mtopology-of-euclidean uniformly-continuous-imp-continuous-map*)
    next
      fix *x*
      assume *x* ∈ *topspace X*
      then show *mdist-set ms.Self* {*xn k*} *x* = *d x* (*xn k*)
        by(*auto simp*: *ms.mdist-set-Self xns*)
    qed **next**
    show *d x* (*xn k*) ∈ {*0..1*} **for** *x k*
      using *d*(*2*) *ms.nonneg* **by**(*auto simp*: *less-le-not-le*)
  qed
  hence *f-cont′*: *continuous-map X* (*subtopology Hilbert-cube-topology* (*f* ‘ *topspace X*)) *f*
    using *continuous-map-into-subtopology* **by** *blast*
  obtain *g* **where** *g*: *g* ‘ (*f* ‘ *topspace X*) = *topspace X* ⋀*x. x* ∈ *topspace X* ⟹ *g* (*f x*) = *x* ⋀*x. x* ∈ *f* ‘ *topspace X* ⟹ *f* (*g x*) = *x*
    by (*meson f-inj f-the-inv-into-f the-inv-into-f-eq the-inv-into-onto*)
  have *g-cont*: *continuous-map* (*subtopology Hilbert-cube-topology* (*f* ‘ *topspace X*)) *X g*
  proof −
    interpret *m01*: *Submetric UNIV dist* {*0..1*::*real*}
    by(*simp add*: *Submetric-def Submetric-axioms-def Met-TC.Metric-space-axioms*)
    have *m01-eq*: *m01.sub.mtopology* = *top-of-set* {*0..1*}
      using *m01.mtopology-submetric* **by** *auto*
    have *m01-Polish*: *Polish-space m01.sub.mtopology*
      by(*auto simp*: *m01-eq intro*!: *Polish-space-closedin*)
    interpret *m01′*: *Metric-space* {*0..1*::*real*} *λx y. if 0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1 then dist x y else 0*
      by(*auto intro*!: *Metric-space-eq*[*OF m01.sub.Metric-space-axioms*]) *metric*

have *m01′-eq*: *m01′.mtopology* = *top-of-set* {*0..1*}
by(*auto intro*!: *Metric-space-eq-mtopology*[*OF m01.sub.Metric-space-axioms*,*simplified m01-eq*,*symmetric*]) *metric*
have *dist x y* ≤ *1 dist x y* ≥ *0* **if** *x* ≥ *0 x* ≤ *1 y* ≥ *0 y* ≤ *1* **for** *x y* :: *real*
using *dist-real-def that* **by** *auto*
then interpret *ppm*: *Product-metric 1/2 UNIV* :: *nat set id id* λ-. {*0..1*::*real*} λ- *x y*. *if* *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1 then dist x y else 0 1*
by(*auto intro*!: *product-metric-natI Metric-space-eq*[*OF m01.sub.Metric-space-axioms*] *simp*: *m01.sub.commute*)

have *Hilbert-cube-eq*: *ppm.Product-metric.mtopology* = *Hilbert-cube-topology*
by(*simp add*: *ppm.Product-metric-mtopology-eq*[*symmetric*] *m01′-eq*)
interpret *f-S*: *Submetric* Π_E *x*∈*UNIV*. {*0..1*} *ppm.product-dist f* ‘ *topspace X*
by(*auto simp*: *Submetric-def ppm.Product-metric.Metric-space-axioms Submetric-axioms-def f-def order.strict-implies-order*[*OF d*(*2*)])
have *1*:*subtopology Hilbert-cube-topology* (*f* ‘ *topspace X*) = *f-S.sub.mtopology*
using *Hilbert-cube-eq f-S.mtopology-submetric* **by** *auto*
have *continuous-map f-S.sub.mtopology ms.mtopology g*
unfolding *continuous-map-iff-limit-seq*[*OF f-S.sub.first-countable-mtopology*]
proof *safe*
fix *yn y*
assume *h*: *limitin f-S.sub.mtopology yn y sequentially*
have *h′*:*limitin ppm.Product-metric.mtopology yn y sequentially*
using *f-S.limitin-submetric-iff h* **by** *blast*
hence *m01-conv*:⋀*n*. *limitin* *m01′.mtopology* (λ*i*. *yn i n*) (*y n*) *sequentially* *y* ∈ *UNIV* $\rightarrow_E$ {*0..1*}
by(*auto simp*: *ppm.limitin-M-iff-limitin-Mi*)
have ∃ *N*. ∀ *n*≥*N*. ∃ *zn*. *yn n* = *f zn* ∧ *zn* ∈ *topspace X*
using *h g* **by**(*simp only*: *f-S.sub.limit-metric-sequentially*) (*meson imageE ppm.K-pos*)
then obtain *N′ zn* **where** *zn*:⋀*n*. *n* ≥ *N′* ⟹ *f* (*zn n*) = *yn n* ⋀*n*. *n* ≥ *N′* ⟹ *zn n* ∈ *topspace X*
by *metis*
obtain *z* **where** *z*:*f z* = *y z* ∈ *topspace X*
using *h f-S.sub.limitin-mspace* **by** *blast*
show *limitin ms.mtopology* (λ*n*. *g* (*yn n*)) (*g y*) *sequentially*
unfolding *ms.limit-metric-sequentially*
proof *safe*
fix *ε* :: *real*
assume *he*: *0* < *ε*
then have *0* < *ε* / *3* **by** *simp*
then obtain *m* **where** *m*:*d z* (*xn m*) < *ε* / *3*
using *ms.mdense-def2*[*of U*,*simplified d*(*1*)] *U*(*2*) *z*(*2*) *xn*(*2*) **by** *blast*
have ⋀*e*. *e*>*0* ⟹ ∃ *N*. ∀ *n*≥*N*. *yn n m* ∈ {*0..1*} ∧ *dist* (*yn n m*) (*y m*) < *e*
using *m01-conv*(*1*)[*of m*,*simplified m01′.limit-metric-sequentially*]
by *fastforce*
from *this*[*OF* ‹*0* < *ε* / *3*›] **obtain** *N* **where** ⋀*n*. *n* ≥ *N* ⟹ |*yn n m* −

y m| < ε / 3 ⋀n. n ≥ N ⟹ yn n m ∈ {0..1}
by(auto simp: dist-real-def)
hence N:⋀n. n ≥ N ⟹ yn n m < ε / 3 + y m
by (metis abs-diff-less-iff add.commute)
have ∃ N. ∀ n≥N. zn n ∈ topspace X ∧ d (zn n) z < ε
proof(safe intro!: exI[where x=max N N'])
fix n
assume max N N' ≤ n
then have N ≤ n N' ≤ n
by auto
then have d (zn n) z ≤ f (zn n) m + d z (xn m)
using ms.triangle[OF zn(2)[of n] xns[of m] z(2),simplified ms.commute[of
xn m z]]
by(auto simp: f-def)
also have ... < ε / 3 + y m + d z (xn m)
using N[OF ‹N≤n›] zn(1)[of n] ‹N' ≤ n› by simp
also have ... = ε / 3 + d z (xn m) + d z (xn m)
by(simp add: z(1)[symmetric] f-def)
also have ... < ε
using m by auto
finally show d (zn n) z < ε .
qed(use zn in auto)
thus ∃ N. ∀ n≥N. g (yn n) ∈ topspace X ∧ d (g (yn n)) (g y) < ε
by (metis dual-order.trans nle-le zn(1) z(1) g(2)[OF z(2)] g(2)[OF
zn(2)])
qed(use g z in auto)
qed
hence continuous-map f-S.sub.mtopology ms.mtopology g
by(auto simp: mtopology-of-def)
thus ?thesis
by(simp add: d(1) 1)
qed
show ?thesis
using f-img g(2,3) f-cont' g-cont
by(auto intro!: exI[where x=f ‘ topspace X] homeomorphic-maps-imp-homeomorphic-space[where
f=f and g=g] simp: homeomorphic-maps-def)
qed
qed

corollary *embedding-into-Hilbert-cube-gdelta-in*:
assumes *Polish-space X*
shows ∃ A. gdelta-in Hilbert-cube-topology A ∧ X homeomorphic-space (subtopology
Hilbert-cube-topology A)
proof −
obtain A **where** h:A ⊆ topspace Hilbert-cube-topology X homeomorphic-space
subtopology Hilbert-cube-topology A
using embedding-into-Hilbert-cube Polish-space-imp-metrizable-space Polish-space-imp-separable-space
assms **by** blast
with completely-metrizable-space-homeo-image-gdelta-in[OF Polish-space-imp-completely-metrizable-space[OF

assms] *Polish-space-imp-completely-metrizable-space*[*OF Polish-space-Hilbert-cube*] *h*(*1*,*2*)]
show *?thesis*
by *blast*
qed

2.3.2 Embedding from Cantor Space

lemma *embedding-from-Cantor-space*:
assumes *Polish-space X uncountable* (*topspace X*)
shows ∃ *A*. *gdelta-in X A* ∧ *Cantor-space-topology homeomorphic-space* (*subtopology X A*)
proof −
obtain *U P* **where** *up*: *countable U openin X U perfect-set X PU* ∪ *P* = *topspace X U* ∩ *P* = {} ⋀*a*. *a* ≠ {} ⟹ *openin* (*subtopology X P*) *a* ⟹ *uncountable a*
using *Cantor-Bendixon*[*OF Polish-space-imp-second-countable*[*OF assms*(*1*)]] **by** *auto*
have *P*: *closedin X P P* ⊆ *topspace X uncountable P*
using *countable-Un-iff*[*of U P*] *up*(*1*) *assms up*(*4*)
by(*simp-all add*: *perfect-setD*[*OF up*(*3*)])
then have *pp*: *Polish-space* (*subtopology X P*)
by (*simp add*: *assms*(*1*) *Polish-space-closedin*)
have *Ptop*: *topspace* (*subtopology X P*) = *P*
using *P*(*2*) **by** *auto*
obtain *U* **where** *U*: *countable U dense-in* (*subtopology X P*) *U*
using *Polish-space-def pp separable-space-def2* **by** *auto*
with *uncountable-infinite*[*OF P*(*3*)] *P*(*2*)
have *infinite U*
by (*metis Metric-space.t1-space-mtopology Ptop assms*(*1*) *completely-metrizable-space-def dense-in-infinite Polish-space-def t1-space-subtopology*)
obtain *d* **where** *Metric-space P d* **and** *d*:*Metric-space.mtopology P d* = *subtopology X P* **and** *mdc*: *Metric-space.mcomplete P d*
by (*metis Metric-space.topspace-mtopology Ptop completely-metrizable-space-def Polish-space-def pp*)
interpret *md*: *Metric-space P d* **by** *fact*
define *xn* **where** *xn* ≡ *from-nat-into U*
have *xn*: *bij-betw xn UNIV U* ⋀*n m*. *n* ≠ *m* ⟹ *xn n* ≠ *xn m* ⋀*n*. *xn n* ∈ *U* ⋀*n*. *xn n* ∈ *P md.mdense* (*range xn*)
using *bij-betw-from-nat-into*[*OF U*(*1*) ‹*infinite U*›] *dense-in-subset*[*OF U*(*2*)] *d U*(*2*) *range-from-nat-into*[*OF infinite-imp-nonempty*[*OF* ‹*infinite U*›] *U*(*1*)]
by(*auto simp add*: *xn-def U*(*1*) ‹*infinite U*› *from-nat-into*[*OF infinite-imp-nonempty*[*OF* ‹*infinite U*›]])
have *perfect*:*perfect-space md.mtopology*
using *d perfect-set-subtopology up*(*3*) **by** *simp*
define *jn* **where** *jn* ≡ (*λn*. *LEAST i*. *i* > *n* ∧ *md.mcball* (*xn i*) ((*1*/*2*)^*i*) ⊆ *md.mball* (*xn n*) ((*1*/*2*)^*n*) − *md.mball* (*xn n*) ((*1*/*2*)^*i*))
define *kn* **where** *kn* ≡ (*λn*. *LEAST k*. *k* > *jn n* ∧ *md.mcball* (*xn k*) ((*1*/*2*)^*k*) ⊆ *md.mball* (*xn n*) ((*1*/*2*)^*jn n*))
have *dxmxn*: ∀ *n n'*. ∃ *m*. *m* > *n* ∧ *m* > *n'* ∧ (*1*/*2*)^(*m*−*1*) < *d* (*xn n*) (*xn m*)

```
∧ d (xn n) (xn m) < (1/2)^(Suc n')
  proof safe
    fix n n'
    have hinfin':infinite (md.mball x e ∩ (range xn)) if x ∈ P e > 0 for x e
    proof
      assume h-fin:finite (md.mball x e ∩ range xn)
      have h-nen:md.mball x e ∩ range xn ≠ {}
        using xn(5) that by(auto simp: md.mdense-def)
      have infin: infinite (md.mball x e)
        using md.perfect-set-mball-infinite[OF perfect] that by simp
      then obtain y where y:y ∈ md.mball x e y ∉ range xn
        using h-fin by(metis inf.absorb-iff2 inf-commute subsetI)
      define e' where e' = Min {d y xk |xk. xk ∈ md.mball x e ∩ range xn}
      have fin: finite  {d y xk |xk. xk ∈ md.mball x e ∩ range xn}
        using finite-imageI[OF h-fin,of d y] by (metis Setcompr-eq-image)
      have nen: {d y xk |xk. xk ∈ md.mball x e ∩ range xn} ≠ {}
        using h-nen by auto
      have e' > 0
        unfolding e'-def Min-gr-iff[OF fin nen]
      proof safe
        fix l
        assume xn l ∈ md.mball x e
        with y
        show 0 < d y (xn l)
          by auto
      qed
      obtain e'' where e'': e'' > 0 md.mball y e'' ⊆ md.mball x e y ∈ md.mball y
e''
      by (meson md.centre-in-mball-iff md.in-mball md.openin-mball md.openin-mtopology
y(1))
      define ε where ε ≡ min e' e''
      have ε > 0
        using e''(1) ‹e' > 0› by(simp add: ε-def)
      then obtain m where m: d y (xn m) < ε
        using md.mdense-def2[of range xn] xn(5) y(1) by fastforce
      consider xn m ∈ md.mball x e | xn m ∈ P − md.mball x e
        using xn(4) by auto
      then show False
      proof cases
        case 1
        then have e' ≤ d y (xn m)
          using Min-le-iff[OF fin nen] by(auto simp: e'-def)
        thus ?thesis
          using m by(simp add: ε-def)
      next
        case 2
        then have xn m ∉ md.mball y e''
          using e''(2) by auto
        hence e'' ≤ d y (xn m)
```

using *y e'' xn* **by** *auto*
thus *?thesis*
using *m* **by**(*simp add*: *ε-def*)
qed
qed
have *hinfin:infinite* (*md.mball x e* ∩ (*xn* ' {*l*<..})) **if** *x* ∈ *P e* > *0* **for** *x e l*
proof
assume *finite* (*md.mball x e* ∩ *xn* ' {*l*<..})
moreover have *finite* (*md.mball x e* ∩ *xn* ' {..*l*}) **by** *simp*
moreover have (*md.mball x e* ∩ (*range xn*)) = (*md.mball x e* ∩ *xn* ' {*l*<..}) ∪ (*md.mball x e* ∩ *xn* ' {..*l*})
by *fastforce*
ultimately have *finite* (*md.mball x e* ∩ (*range xn*))
by *auto*
with *hinfin'*[*OF that*] **show** *False* **..**
qed
have *infinite* (*md.mball* (*xn n*) ((*1/2*)^*Suc n'*))
using *md.perfect-set-mball-infinite*[*OF perfect*] *xn*(*4*)[*of n*] **by** *simp*
then obtain *x* **where** *x*: *x* ∈ *md.mball* (*xn n*) ((*1/2*)^*Suc n'*) *x* ≠ *xn n*
by (*metis finite-insert finite-subset infinite-imp-nonempty singletonI subsetI*)
then obtain *e* **where** *e*: *e* > *0 md.mball x e* ⊆ *md.mball* (*xn n*) ((*1/2*)^*Suc n'*) *x* ∈ *md.mball x e*
by (*meson md.centre-in-mball-iff md.in-mball md.openin-mball md.openin-mtopology*)
have *d* (*xn n*) *x* > *0*
using *xn x* **by** *simp*
then obtain *m'* **where** *m'*: *m'* − *1* > *0* (*1/2*)^(*m'* − *1*) < *d* (*xn n*) *x*
by (*metis One-nat-def diff-Suc-Suc diff-zero one-less-numeral-iff reals-power-lt-ex semiring-norm*(*76*))
define *m* **where** *m* ≡ *max m'* (*max n'* (*Suc n*))
then have *m* ≥ *m' m* ≥ *n' m* ≥ *Suc n* **by** *simp-all*
hence *m*: *m* − *1* > *0* (*1/2*)^(*m* − *1*) < *d* (*xn n*) *x m* > *n*
using *m' less-trans*[*OF - m'*(*2*),*of* (*1* / *2*) ^ (*m* − *1*)]
by *auto* (*metis diff-less-mono le-eq-less-or-eq*)
define *ε* **where** *ε* ≡ *min e* (*d* (*xn n*) *x* − (*1/2*)^(*m* − *1*))
have *ε* > *0*
using *e m* **by**(*simp add*: *ε-def*)
have *ball-le*:*md.mball x ε* ⊆ *md.mball* (*xn n*) ((*1* / *2*) ^ *Suc n'*)
using *e*(*2*) **by**(*auto simp add*: *ε-def*)
obtain *k* **where** *k*: *xn k* ∈ *md.mball x ε k* > *m*
using ‹*ε* > *0*› *infinite-imp-nonempty*[*OF hinfin*,*of - ε*] *x*(*1*) **by** *fastforce*
show ∃ *m*>*n*. *n'* < *m* ∧ (*1* / *2*) ^ (*m* − *1*) < *d* (*xn n*) (*xn m*) ∧ *d* (*xn n*) (*xn m*) < (*1* / *2*) ^ *Suc n'*
proof(*intro exI*[**where** *x*=*k*] *conjI*)
have (*1* / *2*) ^ (*k* − *1*) < (*1* / (*2* :: *real*)) ^ (*m* − *1*)
using *k*(*2*) *m*(*3*) **by** *simp*
also have ... = *d* (*xn n*) *x* + ((*1/2*)^ (*m* − *1*) − *d* (*xn n*) *x*) **by** *simp*
also have ... < *d* (*xn n*) *x* − *d* (*xn k*) *x*
using *k* **by**(*auto simp*: *ε-def md.commute*)
also have ... ≤ *d* (*xn n*) (*xn k*)

using *xn x md.mdist-reverse-triangle*[*of xn n x xn k*] **by**(*auto simp*: *md.commute*)
finally show (*1* / *2*) ^ (*k* − *1*) < *d* (*xn n*) (*xn k*) **.**
qed(*use* ‹*m* ≥ *n*'› *k ball-le m*(*3*) **in** *auto*)
qed
have *jn n* > *n* ∧ *md.mcball* (*xn* (*jn n*)) ((*1*/*2*)^(*jn n*)) ⊆ *md.mball* (*xn n*) ((*1*/*2*)^*n*) − *md.mball* (*xn n*) ((*1*/*2*)^(*jn n*)) **for** *n*
unfolding *jn-def*
proof(*rule LeastI-ex*)
obtain *m* **where** *m*:*m* > *n* (*1* / *2*) ^ (*m* − *1*) < *d* (*xn n*) (*xn m*) *d* (*xn n*) (*xn m*) < (*1* / *2*) ^ *Suc n*
using *dxmxn* **by** *auto*
show ∃*x*>*n*. *md.mcball* (*xn x*) ((*1* / *2*) ^ *x*) ⊆ *md.mball* (*xn n*) ((*1* / *2*) ^ *n*) − *md.mball* (*xn n*) ((*1* / *2*) ^ *x*)
proof(*safe intro*!: *exI*[**where** *x*=*m*] *m*(*1*))
fix *x*
assume *h*:*x* ∈ *md.mcball* (*xn m*) ((*1* / *2*) ^ *m*)
have *1*:*d* (*xn n*) *x* < (*1* / *2*) ^ *n*
proof −
have *d* (*xn n*) *x* < (*1* / *2*) ^ *Suc n* + (*1* / *2*) ^ *m*
using *m*(*3*) *md.triangle*[*OF xn*(*4*)[*of n*] *xn*(*4*)[*of m*],*of x*] *h* **by** *auto*
also have ... ≤ (*1* / *2*) ^ *Suc n* + (*1* / *2*) ^ *Suc n*
by (*metis Suc-lessI add-mono divide-less-eq-1-pos divide-pos-pos less-eq-real-def m*(*1*) *one-less-numeral-iff power-strict-decreasing-iff semiring-norm*(*76*) *zero-less-numeral zero-less-one*)
finally show *?thesis* **by** *simp*
qed
have *2*:(*1* / *2*) ^ *m* ≤ *d* (*xn n*) *x*
proof −
have (*1* / *2*) ^ (*m* − *1*) < *d* (*xn n*) *x* + (*1* / *2*) ^ *m*
using *order.strict-trans2*[*OF m*(*2*) *md.triangle*[*OF xn*(*4*)[*of n*] - *xn*(*4*)[*of m*]]] *h md.commute* **by** *fastforce*
hence (*1* / *2*) ^ (*m* − *1*) − (*1* / *2*) ^ *m* ≤ *d* (*xn n*) *x*
by *simp*
thus *?thesis*
using *not0-implies-Suc*[*OF gr-implies-not0*[*OF m*(*1*)]] **by** *auto*
qed
show *x* ∈ *md.mball* (*xn n*) ((*1* / *2*) ^ *n*)
x ∈ *md.mball* (*xn n*) ((*1* / *2*) ^ *m*) ⟹ *False*
using *xn h 1 2* **by** *auto*
qed
qed
hence *jn*: ⋀*n*. *jn n* > *n* ⋀*n*. *md.mcball* (*xn* (*jn n*)) ((*1*/*2*)^(*jn n*)) ⊆ *md.mball* (*xn n*) ((*1*/*2*)^*n*) − *md.mball* (*xn n*) ((*1*/*2*)^(*jn n*))
by *simp-all*
have *kn n* > *jn n* ∧ *md.mcball* (*xn* (*kn n*)) ((*1*/*2*)^(*kn n*)) ⊆ *md.mball* (*xn n*) ((*1*/*2*)^*jn n*) **for** *n*
unfolding *kn-def*
proof(*rule LeastI-ex*)

obtain *m* **where** *m*:*m* > *jn n d* (*xn n*) (*xn m*) < (*1* / *2*) ^ *Suc* (*jn n*)
using *dxmxn* **by** *blast*
show ∃ *x*>*jn n*. *md.mcball* (*xn x*) ((*1* / *2*) ^ *x*) ⊆ *md.mball* (*xn n*) ((*1* / *2*) ^ *jn n*)
proof(*intro exI*[**where** *x*=*m*] *conjI*)
show *md.mcball* (*xn m*) ((*1* / *2*) ^ *m*) ⊆ *md.mball* (*xn n*) ((*1* / *2*) ^ *jn n*)
proof
fix *x*
assume *h*:*x* ∈ *md.mcball* (*xn m*) ((*1* / *2*) ^ *m*)
have *d* (*xn n*) *x* < (*1* / *2*)^ *Suc* (*jn n*) + (*1* / *2*) ^ *m*
using *md.triangle*[*OF xn*(*4*)[*of n*] *xn*(*4*)[*of m*]] *h m*(*2*) **by** *fastforce*
also have ... ≤ (*1* / *2*)^ *Suc* (*jn n*) + (*1* / *2*)^ *Suc* (*jn n*)
by (*metis Suc-le-eq add-mono dual-order.refl less-divide-eq-1-pos linorder-not-less m*(*1*) *not-numeral-less-one power-decreasing zero-le-divide-1-iff zero-le-numeral zero-less-numeral*)
finally show *x* ∈ *md.mball* (*xn n*) ((*1* / *2*) ^ *jn n*)
using *xn*(*4*) *h* **by** *auto*
qed
qed(*use m*(*1*) **in** *auto*)
qed
hence *kn*: ⋀*n*. *kn n* > *jn n* ⋀*n*. *md.mcball* (*xn* (*kn n*)) ((*1*/*2*)^(*kn n*)) ⊆ *md.mball* (*xn n*) ((*1*/*2*)^(*jn n*))
by *simp-all*
have *jnkn-pos*: *jn n* > *0 kn n* > *0* **for** *n*
using *not0-implies-Suc*[*OF gr-implies-not0*[*OF jn*(*1*)[*of n*]]] *kn*(*1*)[*of n*] **by** *auto*

define *bn* :: *real list* ⇒ *nat*
where *bn* ≡ *rec-list 1* (λ*a l t*. *if a* = *0 then jn t else kn t*)
have *bn-simp*: *bn* [] = *1 bn* (*a* # *l*) = (*if a* = *0 then jn* (*bn l*) *else kn* (*bn l*)) **for** *a l*
by(*simp-all add*: *bn-def*)
define *to-listn* :: (*nat* ⇒ *real*) ⇒ *nat* ⇒ *real list*
where *to-listn* ≡ (λ*x* . *rec-nat* [] (λ*n t*. *x n* # *t*))
have *to-listn-simp*: *to-listn x 0* = [] *to-listn x* (*Suc n*) = *x n* # *to-listn x n* **for** *x n*
by(*simp-all add*: *to-listn-def*)
have *to-listn-eq*: (⋀*m*. *m* < *n* ⟹ *x m* = *y m*) ⟹ *to-listn x n* = *to-listn y n* **for** *x y n*
by(*induction n*) (*auto simp*: *to-listn-simp*)
have *bn-gtn*: *bn* (*to-listn x n*) > *n* **for** *x n*
apply(*induction n arbitrary*: *x*)
using *jn*(*1*) *kn*(*1*) **by**(*auto simp*: *bn-simp to-listn-simp*) (*meson Suc-le-eq le-less less-trans-Suc*)+
define *rn* **where** *rn* ≡ (λ*n*. *Min* (*range* (λ*x*. (*1* / *2* :: *real*) ^ *bn* (*to-listn x n*))))
have *rn-fin*: *finite* (*range* (λ*x*. (*1* / *2* :: *real*) ^ *bn* (*to-listn x n*))) **for** *n*
proof −
have *finite* (*range* (λ*x*. *bn* (*to-listn x n*)))
proof(*induction n*)
case *ih*:(*Suc n*)
have (*range* (λ*x*. *bn* (*to-listn x* (*Suc n*)))) ⊆ (*range* (λ*x*. *jn* (*bn* (*to-listn x*

n)))) ∪ (*range* (λ*x*. *kn* (*bn* (*to-listn x n*))))
by(*auto simp*: *to-listn-simp bn-simp*)
moreover have *finite* ...
using *ih finite-range-imageI* **by** *auto*
ultimately show *?case* **by**(*rule finite-subset*)
qed(*simp add*: *to-listn-simp*)
thus *?thesis*
using *finite-range-imageI* **by** *blast*
qed
have *rn-nen*: (*range* (λ*x*. (*1* / *2* :: *real*) ^ *bn* (*to-listn x n*))) ≠ {} **for** *n*
by *simp*
have *rn-pos*: *0* < *rn n* **for** *n*
by(*simp add*: *Min-gr-iff*[*OF rn-fin rn-nen*] *rn-def*)
have *rn-less*: *rn n* < (*1*/*2*)^*n* **for** *n*
using *bn-gtn*[*of n*] **by**(*auto simp*: *rn-def Min-less-iff*[*OF rn-fin rn-nen*])
have *cball-le-ball*:*md.mcball* (*xn* (*bn* (*a*#*l*))) ((*1*/*2*)^(*bn* (*a*#*l*))) ⊆ *md.mball* (*xn* (*bn l*)) ((*1*/*2*) ^(*bn l*)) **for** *a l*
using *kn*(*2*)[*of bn l*] *less-imp-le*[*OF jn*(*1*)] *jn*(*2*) *md.mball-subset-concentric*[*of* (*1* / *2*) ^ *jn* (*bn l*) (*1* / *2*) ^ *bn l xn* (*bn l*)]
by(*auto simp*: *bn-simp*)
hence *cball-le*:*md.mcball* (*xn* (*bn* (*a*#*l*))) ((*1*/*2*)^(*bn* (*a*#*l*))) ⊆ *md.mcball* (*xn* (*bn l*)) ((*1*/*2*) ^(*bn l*)) **for** *a l*
using *md.mball-subset-mcball* **by** *blast*
have *cball-disj*: *md.mcball* (*xn* (*bn* (*0*#*l*))) ((*1*/*2*)^(*bn* (*0*#*l*))) ∩ *md.mcball* (*xn* (*bn* (*1*#*l*))) ((*1*/*2*)^(*bn* (*1*#*l*))) = {} **for** *l*
using *jn*(*2*) *kn*(*2*) **by**(*auto simp*: *bn-simp*)
have ∀ *x*. ∃ *l*. *l* ∈ *P* ∧ (⋂ *n*. *md.mcball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*))) = {*l*}
proof
fix *x*
show ∃ *l*. *l* ∈ *P* ∧ (⋂ *n*. *md.mcball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*))) = {*l*}
proof(*safe intro*!: *md.mcomplete-nest-sing*[*THEN iffD1*,*OF mdc*,*rule-format*])
show *md.mcball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*)) = {} ⟹ *False* **for** *n*
using *md.mcball-eq-empty xn*(*4*) **by** *auto*
next
show *decseq* (λ*n*. *md.mcball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*)))
by(*intro decseq-SucI*,*simp add*: *to-listn-simp cball-le*)
next
fix *e* :: *real*
assume *0* < *e*
then obtain *N* **where** *N*: (*1* / *2*) ^ *N* < *e*
by (*meson reals-power-lt-ex rel-simps*(*49*) *rel-simps*(*9*))
show ∃ *n a*. *md.mcball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*)) ⊆ *md.mcball a e*
proof(*safe intro*!: *exI*[**where** *x*=*N*] *exI*[**where** *x*=*xn* (*bn* (*to-listn x N*))])
fix *y*

assume *y* ∈ *md.mcball* (*xn* (*bn* (*to-listn x N*))) ((*1* / *2*) ^ *bn* (*to-listn x N*))
then have *y* ∈ *md.mcball* (*xn* (*bn* (*to-listn x N*))) ((*1* / *2*) ^ *N*)
using *md.mcball-subset-concentric*[*OF power-decreasing*[*OF less-imp-le*[*OF bn-gtn*[*of N x*]],*of 1/2*]]
by *fastforce*
thus *y* ∈ *md.mcball* (*xn* (*bn* (*to-listn x N*))) *e*
using *N* ‹*0* < *e*› **by** *auto*
qed
qed
qed
then obtain *f* **where** *f*:⋀*x*. *f x* ∈ *P* ⋀*x*. (⋂*n*. *md.mcball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*))) = {*f x*}
by *metis*
hence *f'*: ⋀*n x*. *f x* ∈ *md.mcball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*))
by *blast*
have *f''*: *f x* ∈ *md.mball* (*xn* (*bn* (*to-listn x n*))) ((*1* / *2*) ^ *bn* (*to-listn x n*)) **for** *n x*
using *f'*[*of x Suc n*] *cball-le-ball*[*of - to-listn x n*] **by**(*fastforce simp*: *to-listn-simp*)
interpret *bdmd*: *Submetric P d f* ‘ (Π_E *i*∈*UNIV*. {*0*,*1*})
by *standard* (*use f* **in** *auto*)
have *bdmd-sub*: *bdmd.sub.mtopology* = *subtopology X* (*f* ‘ (Π_E *i*∈*UNIV*. {*0*,*1*}))
using *f*(*1*) *Int-absorb1*[*of f* ‘ (*UNIV* $\rightarrow_E$ {*0*, *1*}) *P*] **by**(*fastforce simp*: *bdmd.mtopology-submetric d subtopology-subtopology*)
let *?d* = λ*x y*. *if* (*x* = *0* ∨ *x* = *1*) ∧ (*y* = *0* ∨ *y* = *1*) *then dist x y else 0*
interpret *d01*: *Metric-space* {*0*,*1*::*real*} *?d*
by(*auto simp*: *Metric-space-def*)
have *d01*: *d01.mtopology* = *top-of-set* {*0*,*1*}
proof −
have *d01.mtopology* = *Metric-space.mtopology* {*0*,*1*} *dist*
by(*auto intro*!: *Metric-space-eq-mtopology simp*: *Metric-space-def metric-space-class.dist-commute*)
also have ... = *top-of-set* {*0*,*1*}
by(*auto intro*!: *Submetric.mtopology-submetric*[*of UNIV dist* {*0*,*1*::*real*},*simplified*] *simp*: *Submetric-def Metric-space-def Submetric-axioms-def dist-real-def*)
finally show *?thesis* **.**
qed
interpret *pd*: *Product-metric 1/2 UNIV id id* λ-. {*0*,*1*::*real*} λ-. *?d 1*
by(*auto intro*!: *product-metric-natI d01.Metric-space-axioms*)
have *mpd-top*: *pd.Product-metric.mtopology* = *Cantor-space-topology*
by(*auto simp*: *pd.Product-metric-mtopology-eq*[*symmetric*] *d01 intro*!: *product-topology-cong*)
define *def-at* **where** *def-at x y* ≡ *LEAST n*. *x n* ≠ *y n* **for** *x y* :: *nat* ⇒ *real*
have *def-atxy*: ⋀*n*. *n* < *def-at x y* ⟹ *x n* = *y n x* (*def-at x y*) ≠ *y* (*def-at x y*) **if** *x* ≠ *y* **for** *x y*
proof −
have ∃ *n*. *x n* ≠ *y n*
using *that* **by** *auto*
from *LeastI-ex*[*OF this*]

show $\bigwedge n.\ n < \textit{def-at } x\ y \Longrightarrow x\ n = y\ n\ x\ (\textit{def-at } x\ y) \neq y\ (\textit{def-at } x\ y)$
using *not-less-Least* **by**(*auto simp*: *def-at-def*)
qed
have *def-at-le-if*: $\textit{pd.product-dist } x\ y \leq (1/2)\hat{}n \Longrightarrow n \leq \textit{def-at } x\ y$ **if** *assm*:$x \neq y\ x \in (\Pi_E\ i{\in}\textit{UNIV}.\ \{0,1\})\ y \in (\Pi_E\ i{\in}\textit{UNIV}.\ \{0,1\})$ **for** *x y n*
proof −
assume *h*:$\textit{pd.product-dist } x\ y \leq (1\ /\ 2)\ \hat{}\ n$
have $x\ m = y\ m$ **if** *m-less-n*: $m < n$ **for** *m*
proof(*rule ccontr*)
assume *nen*: $x\ m \neq y\ m$
then have $?d\ (x\ m)\ (y\ m) = 1$
using *assm*(*2*,*3*) **by**(*auto simp*: *submetric-def*)
hence $1 \leq 2\ \hat{}\ m * \textit{pd.product-dist } x\ y$
using *pd.product-dist-geq*[*of m m*,*simplified*,*OF assm*(*2*,*3*)] **by** *simp*
hence $(1/2)\hat{}m \leq 2\hat{}m * (1/2)\hat{}m * \textit{pd.product-dist } x\ y$ **by** *simp*
hence $(1/2)\hat{}m \leq \textit{pd.product-dist } x\ y$ **by** (*simp add*: *power-one-over*)
also have ... $\leq (1\ /\ 2)\ \hat{}\ n$
by(*simp add*: *h*)
finally show *False*
using *that* **by** *auto*
qed
thus $n \leq \textit{def-at } x\ y$
by (*meson def-atxy*(*2*) *linorder-not-le that*(*1*))
qed
have *def-at-le-then*: $\textit{pd.product-dist } x\ y \leq 2 * (1/2)\hat{}n$ **if** *assm*:$x \neq y\ x \in (\Pi_E\ i{\in}\textit{UNIV}.\ \{0,1\})\ y \in (\Pi_E\ i{\in}\textit{UNIV}.\ \{0,1\})\ n \leq \textit{def-at } x\ y$ **for** *x y n*
proof −
have $\bigwedge m.\ m < n \Longrightarrow x\ m = y\ m$
by (*metis def-atxy*(*1*) *order-less-le-trans that*(*4*))
hence *1*:$\bigwedge m.\ m < n \Longrightarrow ?d\ (x\ m)\ (y\ m) = 0$
by (*simp add*: *submetric-def*)
have $\textit{pd.product-dist } x\ y = (\sum i.\ (1/2)\hat{}(i + n) * (?d\ (x\ (i + n))\ (y\ (i + n)))) + (\sum i{<}n.\ (1/2)\hat{}i * (?d\ (x\ i)\ (y\ i)))$
using *assm pd.product-dist-summable*′[*simplified*] **unfolding** *product-dist-def id-apply* **by**(*auto intro*!: *suminf-split-initial-segment simp*: *product-dist-def*)
also have ... $= (\sum i.\ (1/2)\hat{}(i + n) * (?d\ (x\ (i + n))\ (y\ (i + n))))$
by(*simp add*: *1*)
also have ... $\leq (\sum i.\ (1/2)\hat{}(i + n))$
using *pd.product-dist-summable*′ **unfolding** *id-apply* **by**(*auto intro*!: *suminf-le summable-ignore-initial-segment*)
finally show *?thesis*
using *pd.nsum-of-rK*[*of n*] **by** *simp*
qed
have *d-le-def*: $d\ (f\ x)\ (f\ y) \leq (1/2)\hat{}(\textit{def-at } x\ y)$ **if** *assm*:$x \neq y\ x \in (\Pi_E\ i{\in}\textit{UNIV}.\ \{0,1\})\ y \in (\Pi_E\ i{\in}\textit{UNIV}.\ \{0,1\})$ **for** *x y*
proof −
have *1*:$\textit{to-listn } x\ n = \textit{to-listn } y\ n$ **if** $n \leq \textit{def-at } x\ y$ **for** *n*
proof −
have $\bigwedge m.\ m < n \Longrightarrow x\ m = y\ m$

by (*metis def-atxy*(*1*) *order-less-le-trans that*)
then show *?thesis*
by(*auto intro*!: *to-listn-eq*)
qed
have *f x* ∈ *md.mcball* (*xn* (*bn* (*to-listn x* (*def-at x y*)))) ((*1* / *2*) ^ *bn* (*to-listn x* (*def-at x y*)))
f y ∈ *md.mcball* (*xn* (*bn* (*to-listn x* (*def-at x y*)))) ((*1* / *2*) ^ *bn* (*to-listn x* (*def-at x y*)))
using *f'*[*of x def-at x y*] *f'*[*of y def-at x y*] **by**(*auto simp*: *1*[*OF order-refl*])
hence *d* (*f x*) (*f y*) ≤ *2* ∗ (*1* / *2*) ^ *bn* (*to-listn x* (*def-at x y*))
using *f*(*1*) **by**(*auto intro*!: *md.mdiameter-is-sup'*[*OF* - - *md.mdiameter-cball-leq*])
also have ... ≤ (*1*/*2*)^(*def-at x y*)
proof −
have *Suc* (*def-at x y*) ≤ *bn* (*to-listn x* (*def-at x y*))
using *bn-gtn*[*of def-at x y x*] **by** *simp*
hence (*1* / *2*) ^ *bn* (*to-listn x* (*def-at x y*)) ≤ (*1* / *2* :: *real*) ^ *Suc* (*def-at x y*)
using *power-decreasing-iff*[*OF pd.r*] **by** *blast*
thus *?thesis*
by *simp*
qed
finally show *d* (*f x*) (*f y*) ≤ (*1*/*2*)^(*def-at x y*) **.**
qed
have *fy-in*:*f y* ∈ *md.mcball* (*xn* (*bn* (*to-listn x m*))) ((*1*/*2*)^*bn* (*to-listn x m*)) ⟹ ∀ *l*<*m*. *x l* = *y l* **if** *assm*:*x* ∈ (Π_E *i*∈*UNIV*. {*0*,*1*}) *y* ∈ (Π_E *i*∈*UNIV*. {*0*,*1*}) **for** *x y m*
proof(*induction m*)
case *ih*:(*Suc m*)
have *f y* ∈ *md.mcball* (*xn* (*bn* (*to-listn x m*))) ((*1* / *2*) ^ *bn* (*to-listn x m*))
using *ih*(*2*) *cball-le* **by**(*fastforce simp*: *to-listn-simp*)
with *ih*(*1*) **have** *k*:*k* < *m* ⟹ *x k* = *y k* **for** *k* **by** *simp*
show *?case*
proof *safe*
fix *l*
assume *l* < *Suc m*
then consider *l* < *m* | *l* = *m*
using ‹*l* < *Suc m*› **by** *fastforce*
thus *x l* = *y l*
proof *cases*
case *2*
have *3*:*f y* ∈ *md.mcball* (*xn* (*bn* (*y l* # *to-listn y l*))) ((*1* / *2*) ^ *bn* (*y l* # *to-listn y l*))
using *f'*[*of y Suc l*] **by**(*simp add*: *to-listn-simp*)
have *4*:*f y* ∈ *md.mcball* (*xn* (*bn* (*x l* # *to-listn y l*))) ((*1* / *2*) ^ *bn* (*x l* # *to-listn y l*))
using *ih*(*2*) *to-listn-eq*[*of m x y*,*OF k*] **by**(*simp add*: *to-listn-simp 2*)
show *?thesis*
proof(*rule ccontr*)
assume *x l* ≠ *y l*

```
        then consider x l = 0 y l = 1 | x l = 1 y l = 0
          using assm(1,2) by(auto simp: PiE-def Pi-def) metis
        thus False
          by cases (use cball-disj[of to-listn y l] 3 4 in auto)
      qed
    qed(simp add: k)
  qed
qed simp
have d-le-rn-then: ∃ e>0. ∀ y ∈ (Π_E i∈UNIV. {0,1}). x ≠ y ⟶ d (f x) (f y)
< e ⟶ n ≤ def-at x y if assm: x ∈ (Π_E i∈UNIV. {0,1}) for x n
proof(safe intro!: exI[where x=(1/2)^bn (to-listn x n) − d (xn (bn (to-listn x
n))) (f x)])
  show 0 < (1 / 2) ^ bn (to-listn x n) − d (xn (bn (to-listn x n))) (f x)
    using f'' by auto
next
  fix y
  assume h:y ∈ (Π_E i∈UNIV. {0,1}) d (f x) (f y) < (1 / 2) ^ bn (to-listn x n)
− d (xn (bn (to-listn x n))) (f x) x ≠ y
  then have f y ∈ md.mcball (xn (bn (to-listn x n))) ((1/2)^bn (to-listn x n))
    using md.triangle[OF xn(4)[of bn (to-listn x n)] f(1)[of x] f(1)[of y]]
    by(simp add: xn(4)[of bn (to-listn x n)] f(1)[of y] md.mcball-def)
  with fy-in[OF assm h(1)] have ∀ m < n. x m = y m
    by auto
  thus n ≤ def-at x y
    by (meson def-atxy(2) linorder-not-le h(3))
qed
have 0: f ‘ (Π_E i∈UNIV. {0,1}) ⊆ topspace X
  using f(1) P(2) by auto
have 1: continuous-map pd.Product-metric.mtopology bdmd.sub.mtopology f
 unfolding pd.Product-metric.metric-continuous-map[OF bdmd.sub.Metric-space-axioms]
proof safe
  fix x :: nat ⇒ real and ε :: real
  assume h:x ∈ (Π_E i∈UNIV. {0,1}) 0 < ε
  then obtain n where n:(1/2)^n < ε
    using real-arch-pow-inv[OF - pd.r(2)] by auto
  show ∃δ>0. ∀ y. y∈UNIV →_E {0, 1} ∧ pd.product-dist x y < δ ⟶ d (f x)
(f y) < ε
  proof(safe intro!: exI[where x=(1/2)^n])
    fix y
    assume y:y ∈ (Π_E i∈UNIV. {0,1}) pd.product-dist x y < (1 / 2) ^ n
    consider x = y | x ≠ y by auto
    thus d (f x) (f y) < ε
    proof cases
      case 1
      with y(1) h md.zero[OF f(1)[of y] f(1)[of y]]
      show ?thesis by simp
    next
      case 2
      then have n ≤ def-at x y
```

```
        using h(1) y by(auto intro!: def-at-le-if)
      have d (f x) (f y) ≤ (1/2)^(def-at x y)
        using h(1) y(1) by(auto simp: d-le-def[OF 2 h(1) y(1)])
      also have ... ≤ (1/2)^n
        using ‹n ≤ def-at x y› by simp
      finally show ?thesis
        using n by simp
    qed
  qed simp
qed
have 2: open-map pd.Product-metric.mtopology bdmd.sub.mtopology f
proof –
 have open-map (mtopology-of pd.Product-metric.Self) (subtopology (mtopology-of
md.Self) (f ` mspace pd.Product-metric.Self)) f
  proof(safe intro!: Metric-space-open-map-from-dist)
    fix x :: nat ⇒ real and ε :: real
    assume h:x ∈ mspace pd.Product-metric.Self 0 < ε
    then have x:x ∈ (Π_E i∈UNIV. {0,1}) by simp
    from h obtain n where n: (1/2)^n < ε
      using real-arch-pow-inv[OF - pd.r(2)] by auto
    obtain e where e: e > 0 ⋀y. y ∈ (Π_E i∈UNIV. {0,1}) ⟹ x ≠ y ⟹ d (f
x) (f y) < e ⟹ Suc n ≤ def-at x y
      using d-le-rn-then[OF x,of Suc n] by auto
    show ∃δ>0. ∀y∈mspace pd.Product-metric.Self. mdist md.Self (f x) (f y) <
δ ⟶ mdist pd.Product-metric.Self x y < ε
    unfolding md.mdist-Self pd.Product-metric.mspace-Self pd.Product-metric.mdist-Self
    proof(safe intro!: exI[where x=e])
      fix y
      assume y:y ∈ (Π_E i∈UNIV. {0,1}) and d (f x) (f y) < e
      then have d':d (f x) (f y) < e
        using h(1) by simp
      consider x = y | x ≠ y by auto
      thus pd.product-dist x y < ε
        by cases (use pd.Product-metric.zero[OF y y] h(2) def-at-le-then[OF - x y
e(2)[OF y - d']] n in auto)
    qed(use e(1) in auto)
  qed(use f in auto)
  thus ?thesis
    by (simp add: bdmd.mtopology-submetric mtopology-of-def)
qed
have 3: f ` (topspace pd.Product-metric.mtopology) = topspace bdmd.sub.mtopology
  by simp
have 4: inj-on f (topspace pd.Product-metric.mtopology)
  unfolding pd.Product-metric.topspace-mtopology
proof
  fix x y
  assume h:x ∈ (Π_E i∈UNIV. {0,1}) y ∈ (Π_E i∈UNIV. {0,1}) f x = f y
  show x = y
  proof
```

fix *n*
have *f y* ∈ *md.mcball* (*xn* (*bn* (*to-listn x* (*Suc n*)))) ((*1/2*)^*bn* (*to-listn x* (*Suc n*)))
using *f′*[*of x Suc n*] **by**(*simp add*: *h*)
thus *x n* = *y n*
using *fy-in*[*OF h*(*1*,*2*),*of Suc n*] **by** *simp*
qed
qed
show *?thesis*
using *homeomorphic-map-imp-homeomorphic-space*[*OF bijective-open-imp-homeomorphic-map*[*OF 1 2 3 4*]] *0*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *bdmd-sub completely-metrizable-space-homeo-image-gdelta-in mpd-top Polish-space-Cantor-space Polish-space-def*)
qed

2.4 Borel Spaces generated from Polish Spaces

lemma *closedin-clopen-topology*:
assumes *Polish-space X closedin X a*
shows ∃ *X′*. *Polish-space X′* ∧ (∀ *u*. *openin X u* ⟶ *openin X′ u*) ∧ *topspace X* = *topspace X′* ∧ *sets* (*borel-of X*) = *sets* (*borel-of X′*) ∧ *openin X′ a* ∧ *closedin X′ a*
proof −
have *p1*:*Polish-space* (*subtopology X a*)
by (*simp add*: *assms Polish-space-closedin*)
then obtain *da′* **where** *da′*: *Metric-space a da′ subtopology X a* = *Metric-space.mtopology a da′ Metric-space.mcomplete a da′*
by (*metis Metric-space.topspace-mtopology assms*(*2*) *closedin-subset completely-metrizable-space-def Polish-space-imp-completely-metrizable-space topspace-subtopology-subset*)
define *da* **where** *da* = *Metric-space.capped-dist da′* (*1/2*)
have *da*: *subtopology X a* = *Metric-space.mtopology a da Metric-space.mcomplete a da*
using *da′* **by**(*auto simp*: *da-def Metric-space.mtopology-capped-metric Metric-space.mcomplete-capped-metric*)
interpret *pa*: *Metric-space a da*
using *da′* **by**(*simp add*: *Metric-space.capped-dist da-def*)
have *da-bounded*: ⋀*x y*. *da x y* < *1*
using *da′* **by**(*auto simp*: *da-def Metric-space.capped-dist-def*)
have *p2*:*Polish-space* (*subtopology X* (*topspace X* − *a*))
by (*meson assms*(*1*) *assms*(*2*) *closedin-def Polish-space-openin*)
then obtain *db′* **where** *db′*: *Metric-space* (*topspace X* − *a*) *db′ subtopology X* (*topspace X* − *a*) = *Metric-space.mtopology* (*topspace X* − *a*) *db′ Metric-space.mcomplete* (*topspace X* − *a*) *db′*
by (*metis Diff-subset Metric-space.topspace-mtopology completely-metrizable-space-def Polish-space-imp-completely-metrizable-space topspace-subtopology-subset*)
define *db* **where** *db* = *Metric-space.capped-dist db′* (*1/2*)
have *db*: *subtopology X* (*topspace X* − *a*) = *Metric-space.mtopology* (*topspace X* − *a*) *db Metric-space.mcomplete* (*topspace X* − *a*) *db*
using *db′* **by**(*auto simp*: *db-def Metric-space.mtopology-capped-metric Met-*

ric-space.mcomplete-capped-metric)
interpret *pb*: *Metric-space topspace X* − *a db*
using *db′* **by**(*simp add*: *Metric-space.capped-dist db-def*)
have *db-bounded*: $\bigwedge x\ y.\ db\ x\ y < 1$
using *db′* **by**(*auto simp*: *db-def Metric-space.capped-dist-def*)
interpret *p*: *Sum-metric UNIV* λ*b*. *if b then a else topspace X* − *a* λ*b*. *if b then da else db*
using *da db da-bounded db-bounded* **by**(*auto intro*!: *sum-metricI simp*: *disjoint-family-on-def pa.Metric-space-axioms pb.Metric-space-axioms*)
have *0*: ($\bigcup$*i*. *if i then a else topspace X* − *a*) = *topspace X*
using *closedin-subset assms* **by** *auto*

have *1*: *sets* (*borel-of X*) = *sets* (*borel-of p.Sum-metric.mtopology*)
proof −
have *sigma-sets* (*topspace X*) (*Collect* (*openin X*)) = *sigma-sets* (*topspace X*) (*Collect* (*openin p.Sum-metric.mtopology*))
proof(*rule sigma-sets-eqI*)
fix *a*
assume *a* ∈ *Collect* (*openin X*)
then have *openin p.Sum-metric.mtopology a*
by(*simp only*: *p.openin-mtopology-iff*) (*auto simp*: *0 da*(*1*)[*symmetric*] *db*(*1*)[*symmetric*] *openin-subtopology dest*: *openin-subset*)
thus *a* ∈ *sigma-sets* (*topspace X*) (*Collect* (*openin p.Sum-metric.mtopology*))
by *auto*
next
interpret *s*: *sigma-algebra topspace X sigma-sets* (*topspace X*) (*Collect* (*openin X*))
by(*auto intro*!: *sigma-algebra-sigma-sets openin-subset*)
fix *b*
assume *b* ∈ *Collect* (*openin p.Sum-metric.mtopology*)
then have *openin p.Sum-metric.mtopology b* **by** *auto*
then have *b*:*b* ⊆ *topspace X openin* (*subtopology X a*) (*b* ∩ *a*) *openin* (*subtopology X* (*topspace X* − *a*)) (*b* ∩ (*topspace X* − *a*))
by(*simp-all only*: *p.openin-mtopology-iff*,*insert 0 da*(*1*) *db*(*1*)) (*auto simp*: *all-bool-eq*)
have [*simp*]: (*b* ∩ *a*) ∪ (*b* ∩ (*topspace X* − *a*)) = *b*
using *Diff-partition b*(*1*) **by** *blast*
have (*b* ∩ *a*) ∪ (*b* ∩ (*topspace X* − *a*)) ∈ *sigma-sets* (*topspace X*) (*Collect* (*openin X*))
proof(*rule sigma-sets-Un*)
have [*simp*]:*a* ∈ *sigma-sets* (*topspace X*) (*Collect* (*openin X*))
proof −
have *topspace X* − (*topspace X* − *a*) ∈ *sigma-sets* (*topspace X*) (*Collect* (*openin X*))
by(*rule sigma-sets.Compl*) (*use assms* **in** *auto*)
thus *?thesis*
using *double-diff*[*OF closedin-subset*[*OF assms*(*2*)]] **by** *simp*
qed
from *b*(*2*,*3*) **obtain** *T T′* **where** *T*:*openin X T openin X T′* **and** [*simp*]:*b*

```
∩ a = T ∩ a b ∩ (topspace X − a) = T′ ∩ (topspace X − a)
          by(auto simp: openin-subtopology)
        show b ∩ a ∈ sigma-sets (topspace X) (Collect (openin X))
             b ∩ (topspace X − a) ∈ sigma-sets (topspace X) (Collect (openin X))
          using T assms by auto
      qed
      thus b ∈ sigma-sets (topspace X) (Collect (openin X))
        by simp
    qed
    thus ?thesis
      by(simp only: sets-borel-of p.Sum-metric.topspace-mtopology) (use 0 in auto)
  qed
  have 2:⋀u. openin X u ⟹ openin p.Sum-metric.mtopology u
    by(simp only: p.openin-mtopology-iff) (auto simp: all-bool-eq da(1)[symmetric]
db(1)[symmetric] openin-subtopology dest: openin-subset)
  have 3:openin p.Sum-metric.mtopology a
    by(simp only: p.openin-mtopology-iff) (auto simp: all-bool-eq)
  have 4:closedin p.Sum-metric.mtopology a
    by (metis 0 2 assms(2) closedin-def p.Sum-metric.topspace-mtopology)
  have 5: topspace X = topspace p.Sum-metric.mtopology
    by(simp only: p.Sum-metric.topspace-mtopology) (simp only: 0)
  have 6: Polish-space p.Sum-metric.mtopology
     by(rule p.Polish-spaceI,insert da(2) db(2)  p1 p2) (auto simp: da(1) db(1)
Polish-space-def)
  show ?thesis
    by(rule exI[where x=p.Sum-metric.mtopology]) (insert 5 2 6, simp only: 1 3
4 ,auto)
qed

lemma Polish-space-union-Polish:
  fixes X :: nat ⇒ 'a topology
  assumes ⋀n. Polish-space (X n) ⋀n. topspace (X n) = Xt ⋀x y. x ∈ Xt ⟹ y
∈ Xt ⟹ x ≠ y ⟹ ∃ Ox Oy. (∀ n. openin (X n) Ox) ∧ (∀ n. openin (X n) Oy) ∧
x ∈ Ox ∧ y ∈ Oy ∧ disjnt Ox Oy
  defines Xun ≡ topology-generated-by (⋃n. {u. openin (X n) u})
  shows Polish-space Xun
proof −
  have topsXun:topspace Xun = Xt
    using assms(2) by(auto simp: Xun-def dest:openin-subset)
  define f :: 'a ⇒ nat ⇒ 'a where f ≡ (λx n. x)
  have continuous-map Xun (product-topology X UNIV) f
     by(auto simp: assms(2) topsXun f-def continuous-map-componentwise, auto
simp: Xun-def openin-topology-generated-by-iff continuous-map-def assms(2) dest:openin-subset[of
X -,simplified assms(2)] )
       (insert openin-subopen, fastforce intro!: generate-topology-on.Basis)
   hence 1: continuous-map Xun (subtopology (product-topology X UNIV) (f `
(topspace Xun))) f
    by(auto simp: continuous-map-in-subtopology)
  have 2: inj-on f (topspace Xun)
```

by(*auto simp*: *inj-on-def f-def dest*:*fun-cong*)
have *3*: *f* ' (*topspace Xun*) = *topspace* (*subtopology* (*product-topology X UNIV*) (*f* ' (*topspace Xun*)))
by(*auto simp*: *topsXun assms*(*2*) *f-def*)
have *4*: *open-map Xun* (*subtopology* (*product-topology X UNIV*) (*f* ' (*topspace Xun*))) *f*
proof(*safe intro*!: *open-map-generated-topo*[*OF - 2*[*simplified Xun-def*],*simplified Xun-def*[*symmetric*]])
fix *u n*
assume *u*:*openin* (*X n*) *u*
show *openin* (*subtopology* (*product-topology X UNIV*) (*f* ' *topspace Xun*)) (*f* ' *u*)
unfolding *openin-subtopology*
proof(*safe intro*!: *exI*[**where** *x*={ λi. *if i* = *n then a else b i* |*a b*. *a* ∈*u* ∧ *b* ∈ *UNIV* → *Xt*}])
show *openin* (*product-topology X UNIV*) {λi. *if i* = *n then a else b i* |*a b*. *a* ∈*u* ∧ *b* ∈ *UNIV* → *Xt*}
by(*auto simp*: *openin-product-topology-alt u assms*(*2*) *openin-topspace*[*of X -*,*simplified assms*(*2*)] *intro*!: *exI*[**where** *x*=λi. *if i* = *n then u else Xt*])
(*auto simp*: *PiE-def Pi-def*, *metis openin-subset*[*OF u*,*simplified assms*(*2*)] *in-mono*)
next
show ⋀*y*. *y* ∈ *u* ⟹ ∃ *a b*. *f y* = (λi. *if i* = *n then a else b i*) ∧ *a* ∈ *u* ∧ *b* ∈ *UNIV* → *Xt*
using *assms*(*2*) *f-def openin-subset u* **by** *fastforce*
next
show ⋀*y*. *y* ∈ *u* ⟹ *f y* ∈ *f* ' *topspace Xun*
using *openin-subset*[*OF u*] **by**(*auto simp*: *assms*(*2*) *topsXun*)
next
show ⋀*x xa a b*. *xa* ∈ *topspace Xun* ⟹ *f xa* = (λi. *if i* = *n then a else b i*) ⟹ *a* ∈ *u* ⟹ *b* ∈ *UNIV* → *Xt* ⟹ *f xa* ∈ *f* ' *u*
using *openin-subset*[*OF u*] **by**(*auto simp*: *topsXun assms*(*2*)) (*metis f-def imageI*)
qed
qed
have *5*:(*subtopology* (*product-topology X UNIV*) (*f* ' *topspace Xun*)) *homeomorphic-space Xun*
using *homeomorphic-map-imp-homeomorphic-space*[*OF bijective-open-imp-homeomorphic-map*[*OF 1 4 3 2*]]
by(*simp add*: *homeomorphic-space-sym*[*of Xun*])
show *?thesis*
proof(*safe intro*!: *homeomorphic-Polish-space-aux*[*OF Polish-space-closedin*[*OF Polish-space-product*] *5*] *assms*)
show *closedin* (*product-topology X UNIV*) (*f* ' *topspace Xun*)
proof −
have *1*: *openin* (*product-topology X UNIV*) ((*UNIV* $\rightarrow_E$ *Xt*) − *f* ' *Xt*)
proof(*rule openin-subopen*[*THEN iffD2*])
show ∀ *x*∈(*UNIV* $\rightarrow_E$ *Xt*) − *f* ' *Xt*. ∃ *T*. *openin* (*product-topology X UNIV*) *T* ∧ *x* ∈ *T* ∧ *T* ⊆ (*UNIV* $\rightarrow_E$ *Xt*) − *f* ' *Xt*

proof *safe*
fix *x*
assume *x*:*x* ∈ *UNIV* →$_E$ *Xt x* ∉ *f* ‘ *Xt*
have ∃ *n*. *x n* ≠ *x 0*
proof(*rule ccontr*)
assume ∄ *n*. *x n* ≠ *x 0*
then have ∀ *n*. *x n* = *x 0* **by** *auto*
hence *x* = (λ-. *x 0*) **by** *auto*
thus *False*
using *x* **by**(*auto simp*: *f-def topsXun assms*(*2*))
qed
then obtain *n* **where** *n*: *n* ≠ *0 x n* ≠ *x 0*
by *metis*
from *assms*(*3*)[*OF* - - *this*(*2*)] *x*
obtain *On O0* **where** *h*:⋀*n*. *openin* (*X n*) *On* ⋀*n*. *openin* (*X n*) *O0 x n* ∈ *On x 0* ∈ *O0 disjnt On O0*
by *fastforce*
have *openin* (*product-topology X UNIV*) ((λ*x*. *x 0*) −‘ *O0* ∩ *topspace* (*product-topology X UNIV*))
using *continuous-map-product-coordinates*[*of 0 UNIV X*] *h*(*2*)[*of 0*] **by** *blast*
moreover have *openin* (*product-topology X UNIV*) ((λ*x*. *x n*) −‘ *On* ∩ *topspace* (*product-topology X UNIV*))
using *continuous-map-product-coordinates*[*of n UNIV X*] *h*(*1*)[*of n*] **by** *blast*
ultimately have *op*: *openin* (*product-topology X UNIV*) ((λ*T*. *T 0*) −‘ *O0* ∩ *topspace* (*product-topology X UNIV*) ∩ ((λ*T*. *T n*) −‘ *On* ∩ *topspace* (*product-topology X UNIV*)))
by *auto*
have *xin*:*x* ∈ (λ*T*. *T 0*) −‘ *O0* ∩ *topspace* (*product-topology X UNIV*) ∩ ((λ*T*. *T n*) −‘ *On* ∩ *topspace* (*product-topology X UNIV*))
using *x h*(*3*,*4*) **by**(*auto simp*: *assms*(*2*))
have *subset*:(λ*T*. *T 0*) −‘ *O0* ∩ *topspace* (*product-topology X UNIV*) ∩ ((λ*T*. *T n*) −‘ *On* ∩ *topspace* (*product-topology X UNIV*)) ⊆ (*UNIV* →$_E$ *Xt*) − *f* ‘ *Xt*
using *h*(*5*) **by**(*auto simp*: *assms*(*2*) *disjnt-def f-def*)

show ∃ *T*. *openin* (*product-topology X UNIV*) *T* ∧ *x* ∈ *T* ∧ *T* ⊆ (*UNIV* →$_E$ *Xt*) − *f* ‘ *Xt*
by(*rule exI*[**where** *x*=((λ*x*. *x 0*) −‘ *O0* ∩ *topspace* (*product-topology X UNIV*)) ∩ ((λ*x*. *x n*) −‘ *On* ∩ *topspace* (*product-topology X UNIV*))]) (*use op xin subset* **in** *auto*)
qed
qed
thus *?thesis*
by(*auto simp*: *closedin-def assms*(*2*) *topsXun f-def*)
qed
qed(*simp add*: *f-def*)
qed

lemma *sets-clopen-topology*:
assumes *Polish-space X a* ∈ *sets* (*borel-of X*)
shows ∃ *X′*. *Polish-space X′* ∧ (∀ *u*. *openin X u* ⟶ *openin X′ u*) ∧ *topspace X* = *topspace X′* ∧ *sets* (*borel-of X*) = *sets* (*borel-of X′*) ∧ *openin X′ a* ∧ *closedin X′ a*
proof −
have *a* ∈ *sigma-sets* (*topspace X*) {*U*. *closedin X U*}
using *assms* **by**(*simp add*: *sets-borel-of-closed*)
thus *?thesis*
proof *induction*
case (*Basic a*)
then show *?case*
by(*simp add*: *assms closedin-clopen-topology*)
next
case *Empty*
with *polish-space-axioms assms* **show** *?case*
by *auto*
next
case (*Compl a*)
then obtain *X′* **where** *S′*:*Polish-space X′* (∀ *u*. *openin X u* ⟶ *openin X′ u*) *topspace X* = *topspace X′ sets* (*borel-of X*) = *sets* (*borel-of X′*) *openin X′ a closedin X′ a*
by *auto*
from *closedin-clopen-topology*[*OF S′*(*1*) *S′*(*6*)] *S′*
show *?case* **by** *auto*
next
case *ih*:(*Union a*)
then obtain *Si* **where** *Si*:
⋀*i*. *Polish-space* (*Si i*) ⋀*u i*. *openin X u* ⟹ *openin* (*Si i*) *u* ⋀*i*::*nat*. *topspace X* = *topspace* (*Si i*) ⋀*i*. *sets* (*borel-of X*) = *sets* (*borel-of* (*Si i*)) ⋀*i*. *openin* (*Si i*) (*a i*) ⋀*i*. *closedin* (*Si i*) (*a i*)
by *metis*
define *Sun* **where** *Sun* ≡ *topology-generated-by* (⋃ *n*. {*u*. *openin* (*Si n*) *u*})
have *Sun1*: *Polish-space Sun*
unfolding *Sun-def*
proof(*safe intro*!: *Polish-space-union-Polish*[**where** *Xt*=*topspace X*])
fix *x y*
assume *xy*:*x* ∈ *topspace X y* ∈ *topspace X x* ≠ *y*
then obtain *Ox Oy* **where** *Oxy*: *x* ∈ *Ox y* ∈ *Oy openin X Ox openin X Oy disjnt Ox Oy*
using *metrizable-imp-Hausdorff-space*[*OF Polish-space-imp-metrizable-space*[*OF assms*(*1*)]]
by(*simp only*: *Hausdorff-space-def*) *metis*
show ∃ *Ox Oy*. (∀ *x*. *openin* (*Si x*) *Ox*) ∧ (∀ *x*. *openin* (*Si x*) *Oy*) ∧ *x* ∈ *Ox* ∧ *y* ∈ *Oy* ∧ *disjnt Ox Oy*
by(*rule exI*[**where** *x*=*Ox*],*insert Si*(*2*) *Oxy*, *auto intro*!: *exI*[**where** *x*=*Oy*])
qed (*use Si* **in** *auto*)
have *Suntop*:*topspace X* = *topspace Sun*

using *Si*(*3*) **by**(*auto simp*: *Sun-def dest*: *openin-subset*)
have *Sunsets*: *sets* (*borel-of X*) = *sets* (*borel-of Sun*) (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *sigma-sets* (*topspace X*) ($\bigcup$*n*. {*u*. *openin* (*Si n*) *u*})
proof
show *sets* (*borel-of X*) ⊆ *sigma-sets* (*topspace X*) ($\bigcup$*n*. {*u*. *openin* (*Si n*) *u*})
using *Si*(*2*) **by**(*auto simp*: *sets-borel-of intro*!: *sigma-sets-mono′*)
next
show *sigma-sets* (*topspace X*) ($\bigcup$*n*. {*u*. *openin* (*Si n*) *u*}) ⊆ *sets* (*borel-of X*)
by(*simp add*: *sigma-sets-le-sets-iff*[*of borel-of X* $\bigcup$*n*. {*u*. *openin* (*Si n*) *u*},*simplified space-borel-of*]) (*use Si*(*4*) *sets-borel-of* **in** *fastforce*)
qed
also have ... = *?rhs*
using *borel-of-second-countable′*[*OF Polish-space-imp-second-countable*[*OF Sun1*],*of* $\bigcup$*n*. {*u*. *openin* (*Si n*) *u*}]
by (*simp add*: *Sun-def Suntop subbase-in-def subset-Pow-Union*)
finally show *?thesis* **.**
qed
have *Sun-open*: $\bigwedge$*u i*. *openin* (*Si i*) *u* ⟹ *openin Sun u*
by(*auto simp*: *Sun-def openin-topology-generated-by-iff intro*!: *generate-topology-on.Basis*)
have *Sun-opena*: *openin Sun* ($\bigcup$*i*. *a i*)
using *Sun-open*[*OF Si*(*5*),*simplified Sun-def*] **by**(*auto simp*: *Sun-def openin-topology-generated-by-iff intro*!: *generate-topology-on.UN*)
hence *closedin Sun* (*topspace Sun* − ($\bigcup$*i*. *a i*))
by *auto*
from *closedin-clopen-topology*[*OF Sun1 this*]
show *?case*
using *Suntop Sunsets Sun-open*[*OF Si*(*2*)] *Sun-opena*
by (*metis closedin-def openin-closedin-eq*)
qed
qed

end

3 Standard Borel Spaces

3.1 Standard Borel Spaces

theory *StandardBorel*
imports *Abstract-Metrizable-Topology*
begin

locale *standard-borel* =
fixes *M* :: *′a measure*
assumes *Polish-space*: ∃*S*. *Polish-space S* ∧ *sets M* = *sets* (*borel-of S*)
begin

lemma *singleton-sets*:
assumes $x \in$ *space M*
shows $\{x\} \in$ *sets M*
proof –
obtain *S* **where** *s*:*Polish-space S sets M* = *sets* (*borel-of S*)
using *Polish-space* **by** *blast*
have *closedin S* $\{x\}$
using *assms* **by**(*simp add*: *sets-eq-imp-space-eq*[*OF s*(*2*)] *closedin-Hausdorff-sing-eq*[*OF metrizable-imp-Hausdorff-space*[*OF Polish-space-imp-metrizable-space*[*OF s*(*1*)]]] *space-borel-of*)
thus *?thesis*
using *borel-of-closed s* **by** *simp*
qed

corollary *countable-sets*:
assumes $A \subseteq$ *space M countable A*
shows $A \in$ *sets M*
using *sets.countable*[*OF singleton-sets assms*(*2*)] *assms*(*1*)
by *auto*

lemma *standard-borel-restrict-space*:
assumes $A \in$ *sets M*
shows *standard-borel* (*restrict-space M A*)
proof –
obtain *S* **where** *s*:*Polish-space S sets M* = *sets* (*borel-of S*)
using *Polish-space* **by** *blast*
obtain *S'* **where** *S'*:*Polish-space S' sets M* = *sets* (*borel-of S'*) *openin S' A*
using *sets-clopen-topology*[*OF s*(*1*),*simplified s*(*2*)[*symmetric*],*OF assms*] **by** *auto*
show *?thesis*
using *Polish-space-openin*[*OF S'*(*1*,*3*)] *S'*(*2*)
by(*auto simp*: *standard-borel-def borel-of-subtopology sets-restrict-space intro*!: *exI*[**where** *x*=*subtopology S' A*])
qed

end

locale *standard-borel-ne* = *standard-borel* +
assumes *space-ne*: *space M* $\neq$ {}
begin

lemma *standard-borel-ne-restrict-space*:
assumes $A \in$ *sets M A* $\neq$ {}
shows *standard-borel-ne* (*restrict-space M A*)
using *assms* **by**(*auto simp*: *standard-borel-ne-def standard-borel-ne-axioms-def standard-borel-restrict-space*)

lemma *standard-borel*: *standard-borel M*
by(*rule standard-borel-axioms*)

end

lemma *standard-borel-sets*:
  assumes *standard-borel M* **and** *sets M* = *sets N*
  shows *standard-borel N*
  using *assms* **by**(*simp add*: *standard-borel-def*)

lemma *standard-borel-ne-sets*:
  assumes *standard-borel-ne M* **and** *sets M* = *sets N*
  shows *standard-borel-ne N*
  using *assms* **by**(*simp add*: *standard-borel-def standard-borel-ne-def sets-eq-imp-space-eq*[*OF assms*(*2*)] *standard-borel-ne-axioms-def*)

lemma *pair-standard-borel*:
  assumes *standard-borel M standard-borel N*
  shows *standard-borel* ($M \bigotimes_M N$)
proof −
  obtain *S S′* **where** *hs*:
   Polish-space S sets M = *sets* (*borel-of S*) *Polish-space S′ sets N* = *sets* (*borel-of S′*)
    using *assms* **by**(*auto simp*: *standard-borel-def*)
  have *sets* ($M \bigotimes_M N$) = *sets* (*borel-of* (*prod-topology S S′*))
    unfolding *borel-of-prod*[*OF Polish-space-imp-second-countable*[*OF hs*(*1*)] *Polish-space-imp-second-countable*[*OF hs*(*3*)],*symmetric*]
    using *sets-pair-measure-cong*[*OF hs*(*2*,*4*)] **.**
  thus *?thesis*
    unfolding *standard-borel-def* **by**(*auto intro*!: *exI*[**where** *x=prod-topology S S′*] *simp*: *Polish-space-prod*[*OF hs*(*1*,*3*)])
qed

lemma *pair-standard-borel-ne*:
  assumes *standard-borel-ne M standard-borel-ne N*
  shows *standard-borel-ne* ($M \bigotimes_M N$)
  using *assms* **by**(*auto simp*: *pair-standard-borel standard-borel-ne-def standard-borel-ne-axioms-def space-pair-measure*)

lemma *product-standard-borel*:
  assumes *countable I*
      and $\bigwedge i.\ i \in I \Longrightarrow$ *standard-borel* (*M i*)
    shows *standard-borel* ($\Pi_M\ i{\in}I.\ M\ i$)
proof −
  obtain *S* **where** *hs*:
   $\bigwedge i.\ i \in I \Longrightarrow$ *Polish-space* (*S i*) $\bigwedge i.\ i \in I \Longrightarrow$ *sets* (*M i*) = *sets* (*borel-of* (*S i*))
    using *assms*(*2*) **by**(*auto simp*: *standard-borel-def*) *metis*
  have *sets* ($\Pi_M\ i{\in}I.\ M\ i$) = *sets* ($\Pi_M\ i{\in}I.$ *borel-of* (*S i*))
    using *hs*(*2*) **by**(*auto intro*!: *sets-PiM-cong*)
  also have ... = *sets* (*borel-of* (*product-topology S I*))
    using *assms*(*1*) *Polish-space-imp-second-countable*[*OF hs*(*1*)] **by**(*auto intro*!: *sets-PiM-equal-borel-of*)

finally have *1*:*sets* (Π_M *i*∈*I*. *M i*) = *sets* (*borel-of* (*product-topology S I*)).
show *?thesis*
unfolding *standard-borel-def*
using *assms*(*1*) *hs*(*1*) **by**(*auto intro*!: *exI*[**where** *x*=*product-topology S I*] *Polish-space-product simp*: *1*)
qed

lemma *product-standard-borel-ne*:
assumes *countable I*
and ⋀*i*. *i* ∈ *I* ⟹ *standard-borel-ne* (*M i*)
shows *standard-borel-ne* (Π_M *i*∈*I*. *M i*)
using *assms* **by**(*auto simp*: *standard-borel-ne-def standard-borel-ne-axioms-def product-standard-borel*)

lemma *closed-set-standard-borel*[*simp*]:
fixes *U* :: ′*a* :: *topological-space set*
assumes *Polish-space* (*euclidean* :: ′*a topology*) *closed U*
shows *standard-borel* (*restrict-space borel U*)
by(*auto simp*: *standard-borel-def borel-of-euclidean borel-of-subtopology assms intro*!: *exI*[**where** *x*=*subtopology euclidean U*] *Polish-space-closedin*)

lemma *closed-set-standard-borel-ne*[*simp*]:
fixes *U* :: ′*a* :: *topological-space set*
assumes *Polish-space* (*euclidean* :: ′*a topology*) *closed U U* ≠ {}
shows *standard-borel-ne* (*restrict-space borel U*)
using *assms* **by**(*simp add*: *standard-borel-ne-def standard-borel-ne-axioms-def*)

lemma *open-set-standard-borel*[*simp*]:
fixes *U* :: ′*a* :: *topological-space set*
assumes *Polish-space* (*euclidean* :: ′*a topology*) *open U*
shows *standard-borel* (*restrict-space borel U*)
by(*auto simp*: *standard-borel-def borel-of-euclidean borel-of-subtopology assms intro*!: *exI*[**where** *x*=*subtopology euclidean U*] *Polish-space-openin*)

lemma *open-set-standard-borel-ne*[*simp*]:
fixes *U* :: ′*a* :: *topological-space set*
assumes *Polish-space* (*euclidean* :: ′*a topology*) *open U U* ≠ {}
shows *standard-borel-ne* (*restrict-space borel U*)
using *assms* **by**(*simp add*: *standard-borel-ne-def standard-borel-ne-axioms-def*)

lemma *standard-borel-ne-borel*[*simp*]: *standard-borel-ne* (*borel* :: (′*a* :: *polish-space*) *measure*)
and *standard-borel-ne-lborel*[*simp*]: *standard-borel-ne lborel*
unfolding *standard-borel-def standard-borel-ne-def standard-borel-ne-axioms-def*
by(*auto intro*!: *exI*[**where** *x*=*euclidean*] *simp*: *borel-of-euclidean*)

lemma *count-space-standard*′[*simp*]:
assumes *countable I*
shows *standard-borel* (*count-space I*)

by(*rule standard-borel-sets*[*OF - sets-borel-of-discrete-topology*]) (*auto simp add*: *assms Polish-space-discrete-topology standard-borel-def intro*!: *exI*[**where** *x*=*discrete-topology I*])

lemma *count-space-standard-ne*[*simp*]: *standard-borel-ne* (*count-space* (*UNIV* :: (- :: *countable*) *set*))
by (*simp add*: *standard-borel-ne-def standard-borel-ne-axioms-def*)

corollary *measure-pmf-standard-borel-ne*[*simp*]: *standard-borel-ne* (*measure-pmf* (*p* :: (- :: *countable*) *pmf*))
using *count-space-standard-ne sets-measure-pmf-count-space standard-borel-ne-sets* **by** *blast*

corollary *measure-spmf-standard-borel-ne*[*simp*]: *standard-borel-ne* (*measure-spmf* (*p* :: (- :: *countable*) *spmf*))
using *count-space-standard-ne sets-measure-spmf standard-borel-ne-sets* **by** *blast*

corollary *countable-standard-ne*[*simp*]:
standard-borel-ne (*borel* :: *'a* :: {*countable*,*t2-space*} *measure*)
by(*simp add*: *standard-borel-sets*[*OF - sets-borel-eq-count-space*[*symmetric*]] *standard-borel-ne-def standard-borel-ne-axioms-def*)

lemma(**in** *standard-borel*) *countable-discrete-space*:
assumes *countable* (*space M*)
shows *sets M* = *Pow* (*space M*)
proof *safe*
fix *A*
assume *A* ⊆ *space M*
with *assms* **have** *countable A*
by(*simp add*: *countable-subset*)
thus *A* ∈ *sets M*
using ‹*A* ⊆ *space M*› *singleton-sets*
by(*auto intro*!: *sets.countable*[*of A*])
qed(*use sets.sets-into-space* **in** *auto*)

lemma(**in** *standard-borel*) *measurable-isomorphic-standard*:
assumes *M measurable-isomorphic N*
shows *standard-borel N*
proof −
obtain *S* **where** *S*:*Polish-space S sets M* = *sets* (*borel-of S*)
using *Polish-space* **by** *auto*
from *measurable-isomorphic-borels*[*OF S*(*2*) *assms*]
obtain *S'* **where** *S'*: *S homeomorphic-space S'* ∧ *sets N* = *sets* (*borel-of S'*)
by *auto*
thus *?thesis*
by(*auto simp*: *standard-borel-def homeomorphic-Polish-space-aux*[*OF S*(*1*)] *intro*!:*exI*[**where** *x*=*S'*])
qed

lemma(**in** *standard-borel-ne*) *measurable-isomorphic-standard-ne*:
assumes *M measurable-isomorphic N*
shows *standard-borel-ne N*
using *measurable-ismorphic-empty2*[*OF* - *assms*] **by**(*auto simp*: *measurable-isomorphic-standard*[*OF assms*] *standard-borel-ne-def standard-borel-ne-axioms-def space-ne*)

lemma(**in** *standard-borel*) *standard-borel-embed-measure*:
assumes*inj-on f* (*space M*)
shows *standard-borel* (*embed-measure M f*)
using *measurable-embed-measure2'*[*OF assms*]
by(*auto intro*!: *measurable-isomorphic-standard exI*[**where** *x*=*f*] *simp*: *measurable-isomorphic-def measurable-isomorphic-map-def assms in-sets-embed-measure measurable-def sets.sets-into-space space-embed-measure the-inv-into-into the-inv-into-vimage bij-betw-def*)

corollary(**in** *standard-borel-ne*) *standard-borel-ne-embed-measure*:
assumes*inj-on f* (*space M*)
shows *standard-borel-ne* (*embed-measure M f*)
by (*simp add*: *assms space-embed-measure space-ne standard-borel-embed-measure standard-borel-ne-axioms-def standard-borel-ne-def*)

lemma
shows *standard-ne-ereal*: *standard-borel-ne* (*borel* :: *ereal measure*)
and *standard-ne-ennreal*: *standard-borel-ne* (*borel* :: *ennreal measure*)
using *Polish-space-ereal Polish-space-ennreal* **by**(*auto simp*: *standard-borel-ne-def standard-borel-ne-axioms-def standard-borel-def borel-of-euclidean*)

Cantor space $\mathscr{C}$

definition *Cantor-space* :: (*nat* ⇒ *real*) *measure* **where**
Cantor-space ≡ (Π_M *i*∈ *UNIV*. *restrict-space borel* {*0*,*1*})

lemma *Cantor-space-standard-ne*: *standard-borel-ne Cantor-space*
by(*auto simp*: *Cantor-space-def intro*!: *product-standard-borel-ne*)

lemma *Cantor-space-borel*:
sets (*borel-of Cantor-space-topology*) = *sets Cantor-space*
(**is** *?lhs* = -)
proof −
have *?lhs* = *sets* (Π_M *i*∈ *UNIV*. *borel-of* (*top-of-set* {*0*,*1*}))
by(*auto intro*!: *sets-PiM-equal-borel-of*[*symmetric*] *second-countable-subtopology*)
thus *?thesis*
by(*simp add*: *borel-of-subtopology Cantor-space-def borel-of-euclidean*)
qed

Hilbert cube $\mathscr{H}$

definition *Hilbert-cube* :: (*nat* ⇒ *real*) *measure* **where**
Hilbert-cube ≡ (Π_M *i*∈ *UNIV*. *restrict-space borel* {*0*..*1*})

lemma *Hilbert-cube-standard-ne*: *standard-borel-ne Hilbert-cube*

by(*auto simp*: *Hilbert-cube-def intro*!: *product-standard-borel-ne*)

lemma *Hilbert-cube-borel*:
sets (*borel-of Hilbert-cube-topology*) = *sets Hilbert-cube* (**is** *?lhs* = -)
proof −
have *?lhs* = *sets* (Π_M *i*∈ *UNIV*. *borel-of* (*top-of-set* {*0..1*}))
by(*auto intro*!: *sets-PiM-equal-borel-of*[*symmetric*] *second-countable-subtopology*)
thus *?thesis*
by(*simp add*: *borel-of-subtopology Hilbert-cube-def borel-of-euclidean*)
qed

3.2 Isomorphism between $\mathscr{C}$ and $\mathscr{H}$

lemma *Cantor-space-isomorphic-to-Hilbert-cube*:
Cantor-space measurable-isomorphic Hilbert-cube
proof −

Isomorphism between $\mathscr{C}$ and $[0, 1]$

have *Cantor-space-isomorphic-to-01closed*: *Cantor-space measurable-isomorphic* (*restrict-space borel* {*0..1*::*real*})
proof −
have *space-Cantor-space*: *space Cantor-space* = (Π_E *i*∈ *UNIV*. {*0*,*1*})
by(*simp add*: *Cantor-space-def space-PiM*)
have *space-Cantor-space-01*[*simp*]: *0* ≤ *x n x n* ≤ *1 x n* ∈ {*0*,*1*} **if** *x* ∈ *space Cantor-space* **for** *x n*
using *PiE-mem*[*OF that*[*simplified space-Cantor-space*],*of n*]
by *auto*
have *Cantor-minus-abs-cantor*: (λ*n*. |*x n* − *y n*|) ∈ *space Cantor-space* **if** *assms*:*x* ∈ *space Cantor-space y* ∈ *space Cantor-space* **for** *x y*
unfolding *space-Cantor-space*
proof *safe*
fix *n*
assume |*x n* − *y n*| ≠ *0*
then consider *x n* = *0* ∧ *y n* = *1* | *x n* = *1* ∧ *y n* = *0*
using *space-Cantor-space-01*[*OF assms*(*1*),*of n*] *space-Cantor-space-01*[*OF assms*(*2*),*of n*]
by *auto*
thus |*x n* − *y n*| = *1*
by *cases auto*
qed *simp*

define *Cantor-to-01* :: (*nat* ⇒ *real*) ⇒ *real* **where**
Cantor-to-01 ≡ (λ*x*. ($\sum$ *n*. (*1*/*3*)^(*Suc n*)∗ *x n*))

Cantor-to-01 is a measurable injective embedding.

have *Cantor-to-01-summable'*[*simp*]: *summable* (λ*n*. (*1*/*3*)^(*Suc n*)∗ *x n*) **if** *x* ∈ *space Cantor-space* **for** *x*
proof(*rule summable-comparison-test'*[**where** *g*=λ*n*. (*1*/*3*)^ *n* **and** *N*=*0*])
show *norm* ((*1* / *3*) ^ *Suc n* ∗ *x n*) ≤ (*1* / *3*) ^ *n* **for** *n*

using *space-Cantor-space-01*[*OF that*,*of n*] **by** *auto*
qed *simp*

have *Cantor-to-01-summable*[*simp*]: $\bigwedge x.\ x \in$ *space Cantor-space* $\Longrightarrow$ *summable* ($\lambda n.$ (*1/3*)^ *n*∗ *x n*)
using *Cantor-to-01-summable′* **by** *simp*

have *Cantor-to-01-subst-summable*[*simp*]: *summable* ($\lambda n.$ (*1/3*)^ *n*∗ (*x n* − *y n*)) **if** *assms*:*x* ∈ *space Cantor-space y* ∈ *space Cantor-space* **for** *x y*
proof(*rule summable-comparison-test′*[**where** *g*=$\lambda n.$ (*1/3*)^ *n* **and** *N*=*0*])
show *norm* ((*1* / *3*) ^ *n* ∗ (*x n* − *y n*)) ≤ (*1* / *3*) ^ *n* **for** *n*
using *space-Cantor-space-01*[*OF Cantor-minus-abs-cantor*[*OF assms*],*of n*]
by(*auto simp*: *idom-abs-sgn-class.abs-mult*)
qed *simp*

have *Cantor-to-01-image*: *Cantor-to-01* ∈ *space Cantor-space* → {*0..1*}
proof
fix *x*
assume *h*:*x* ∈ *space Cantor-space*
have *Cantor-to-01 x* ≤ ($\sum$ *n.* (*1/3*)^(*Suc n*))
unfolding *Cantor-to-01-def*
by(*rule suminf-le*) (*use h Cantor-to-01-summable*[*OF h*] **in** *auto*)
also have ... = ($\sum$ *n.* (*1* / *3*) ^ *n*) − (*1*::*real*)
using *suminf-minus-initial-segment*[*OF complete-algebra-summable-geometric*[*of 1/3*::*real*],*of 1*]
by *auto*
finally have *Cantor-to-01 x* ≤ *1*
by(*simp add*: *suminf-geometric*[*of 1/3*])
moreover have *0* ≤ *Cantor-to-01 x*
unfolding *Cantor-to-01-def*
by(*rule suminf-nonneg*) (*use Cantor-to-01-summable*[*OF h*] *h* **in** *auto*)
ultimately show *Cantor-to-01 x* ∈ {*0..1*}
by *simp*
qed
have *Cantor-to-01-measurable*: *Cantor-to-01* ∈ *Cantor-space* $\rightarrow_M$ *restrict-space borel* {*0..1*}
proof(*rule measurable-restrict-space2*)
show *Cantor-to-01* ∈ *borel-measurable Cantor-space*
unfolding *Cantor-to-01-def*
proof(*rule borel-measurable-suminf*)
fix *n*
have ($\lambda x.$ *x n*) ∈ *Cantor-space* $\rightarrow_M$ *restrict-space borel* {*0*, *1*}
by(*simp add*: *Cantor-space-def*)
hence ($\lambda x.$ *x n*) ∈ *borel-measurable Cantor-space*
by(*simp add*: *measurable-restrict-space2-iff*)
thus ($\lambda x.$ (*1* / *3*) ^ *Suc n* ∗ *x n*) ∈ *borel-measurable Cantor-space*
by *simp*
qed
qed(*rule Cantor-to-01-image*)

have *Cantor-to-01-inj*: *inj-on Cantor-to-01* (*space Cantor-space*)
and *Cantor-to-01-preserves-sets*: *A* ∈ *sets Cantor-space* ⟹ *Cantor-to-01* ' *A* ∈ *sets* (*restrict-space borel* {*0..1*}) **for** *A*
proof −
have *sets-Cantor*: *sets Cantor-space* = *sets* (*borel-of* (*product-topology* (λ-. *subtopology euclidean* {*0*,*1*}) *UNIV*))
(**is** *?lhs* = -)
proof −
have *?lhs* = *sets* (Π_M *i*∈ *UNIV*. *borel-of* (*subtopology euclidean* {*0*,*1*}))
by (*simp add*: *Cantor-space-def borel-of-euclidean borel-of-subtopology*)
thus *?thesis*
by(*auto intro*!: *sets-PiM-equal-borel-of second-countable-subtopology Polish-space-imp-second-countable*[*of euclideanreal*])
qed
have *s*:*space Cantor-space* = *topspace* (*product-topology* (λ-. *subtopology euclidean* {*0*,*1*}) *UNIV*)
by(*simp add*: *space-Cantor-space*)

let *?d* = λ*x y*::*real*. *if* (*x* = *0* ∨ *x* = *1*) ∧ (*y* = *0* ∨ *y* = *1*) *then dist x y else 0*
interpret *d01*: *Metric-space* {*0*,*1*::*real*} *?d*
by(*auto simp*: *Metric-space-def*)
have *d01*: *d01.mtopology* = *top-of-set* {*0*,*1*} *d01.mcomplete*
proof −
interpret *Metric-space* {*0*,*1*} *dist*
by (*simp add*: *Met-TC.subspace*)
have *d01.mtopology* = *mtopology*
by(*auto intro*!: *Metric-space-eq-mtopology simp*: *Metric-space-def metric-space-class.dist-commute*)
also have ... = *top-of-set* {*0*,*1*}
by(*auto intro*!: *Submetric.mtopology-submetric*[*of UNIV dist* {*0*,*1*::*real*},*simplified*] *simp*: *Submetric-def Metric-space-def Submetric-axioms-def dist-real-def*)
finally show *d01.mtopology* = *top-of-set* {*0*,*1*} **.**
show *d01.mcomplete*
using *Metric-space-eq-mcomplete*[*OF d01.Metric-space-axioms*,*of dist*] *d01.compact-space-eq-Bolzano-Weierstrass d01.compact-space-imp-mcomplete finite.emptyI finite-subset* **by** *blast*
qed
interpret *pd*: *Product-metric 1/3 UNIV id id* λ-. {*0*,*1*::*real*} λ-. *?d 1*
by(*auto intro*!: *product-metric-natI d01.Metric-space-axioms*)
have *mpd-top*: *pd.Product-metric.mtopology* = *Cantor-space-topology*
by(*auto simp*: *pd.Product-metric-mtopology-eq*[*symmetric*] *d01 intro*!: *product-topology-cong*)
have *pd-mcomplete*: *pd.Product-metric.mcomplete*
by(*auto intro*!: *pd.mcomplete-Mi-mcomplete-M d01*)
interpret *m01*: *Submetric UNIV dist* {*0..1*::*real*}
by(*simp add*: *Submetric-def Submetric-axioms-def Met-TC.Metric-space-axioms*)
have *restrict-space borel* {*0..1*} = *borel-of m01.sub.mtopology*

by (*simp add*: *borel-of-euclidean borel-of-subtopology m01.mtopology-submetric*)
have *pd-def*: *pd.product-dist x y* = ($\sum$ *n*. (*1/3*)^*n* ∗ |*x n* − *y n*|) **if** *x* ∈ *space Cantor-space y* ∈ *space Cantor-space* **for** *x y*
using *space-Cantor-space-01*[*OF that*(*1*)] *space-Cantor-space-01*[*OF that*(*2*)] *that* **by**(*auto simp*: *product-dist-def dist-real-def*)
have *1*: |*Cantor-to-01 x* − *Cantor-to-01 y*| ≤ *pd.product-dist x y* (**is** *?lhs* ≤ *?rhs*) **if** *x* ∈ *space Cantor-space y* ∈ *space Cantor-space* **for** *x y*
proof −
have *?lhs* = |($\sum$ *n*. (*1/3*)^(*Suc n*)∗ *x n* − (*1/3*)^(*Suc n*)∗ *y n*)|
using *that* **by**(*simp add*: *suminf-diff Cantor-to-01-def*)
also have ... = |$\sum$ *n*. (*1/3*)^(*Suc n*) ∗ (*x n* − *y n*) |
by (*simp add*: *right-diff-distrib*)
also have ... ≤ ($\sum$ *n*. |(*1/3*)^(*Suc n*) ∗ (*x n* − *y n*)|)
proof(*rule summable-rabs*)
have (λ*n*. |(*1* / *3*) ^ *Suc n* ∗ (*x n* − *y n*)|) = (λ*n*. (*1* / *3*) ^ *Suc n* ∗ |(*x n* − *y n*)|)
by (*simp add*: *abs-mult-pos mult.commute*)
moreover have *summable* ...
using *Cantor-minus-abs-cantor*[*OF that*] **by** *simp*
ultimately show *summable* (λ*n*. |(*1* / *3*) ^ *Suc n* ∗ (*x n* − *y n*)|) **by** *simp*
qed
also have ... = ($\sum$ *n*. (*1/3*)^(*Suc n*) ∗ |*x n* − *y n*|)
by (*simp add*: *abs-mult-pos mult.commute*)
also have ... ≤ *pd.product-dist x y*
unfolding *pd-def*[*OF that*]
by(*rule suminf-le*) (*use Cantor-minus-abs-cantor*[*OF that*] **in** *auto*)
finally show *?thesis* **.**
qed

have *2*:|*Cantor-to-01 x* − *Cantor-to-01 y*| ≥ *1* / *9* ∗ *pd.product-dist x y* (**is** *?lhs* ≤ *?rhs*) **if** *x* ∈ *space Cantor-space y* ∈ *space Cantor-space* **for** *x y*
proof(*cases x* = *y*)
case *True*
then show *?thesis*
using *pd.Product-metric.zero*[*of x y*] *that* **by**(*simp add*: *space-Cantor-space*)
next
case *False*
then obtain *n′* **where** *x n′* ≠ *y n′* **by** *auto*
define *n* **where** *n* ≡ *Min* {*n*. *n* ≤ *n′* ∧ *x n* ≠ *y n*}
have *n* ≤ *n′*
using ‹*x n′* ≠ *y n′*› *n-def* **by** *fastforce*
have *x n* ≠ *y n*
using ‹*x n′* ≠ *y n′*› *linorder-class.Min-in*[*of* {*n*. *n* ≤ *n′* ∧ *x n* ≠ *y n*}]
by(*auto simp*: *n-def*)
have ∀ *i*<*n*. *x i* = *y i*
proof *safe*
fix *i*
assume *i* < *n*

show $x\ i = y\ i$
proof(*rule ccontr*)
assume $x\ i \neq y\ i$
then have $i \in \{n.\ n \leq n' \wedge x\ n \neq y\ n\}$
using ‹$n \leq n'$› ‹$i < n$› **by** *auto*
thus *False*
using ‹$i < n$› *linorder-class.Min-gr-iff*[*of* $\{n.\ n \leq n' \wedge x\ n \neq y\ n\}$ *i*] ‹$x\ n' \neq y\ n'$›
by(*auto simp*: *n-def*)
qed
qed
have *u1*: $(1/3)$ ^$(Suc\ n) * (1/2) \leq |Cantor\text{-}to\text{-}01\ x - Cantor\text{-}to\text{-}01\ y|$
proof −
have $(1/3)$ ^$(Suc\ n) * (1/2) \leq |(\sum m.\ (1/3)$^$(Suc\ (m + Suc\ n)) * (x\ (m + Suc\ n) - y\ (m + Suc\ n))) + (1\ /\ 3)$ ^ $Suc\ n * (x\ n - y\ n)|$
proof −
have $(1\ /\ 3)$ ^ $Suc\ n - (1/3)$^$(n + 2) * 3/2 \leq (1\ /\ 3)$ ^ $Suc\ n - |(\sum m.\ (1\ /\ 3)$ ^ $Suc\ (m + Suc\ n) * (y\ (m + Suc\ n) - x\ (m + Suc\ n)))|$
proof −
have $|(\sum m.\ (1\ /\ 3)$ ^ $Suc\ (m + Suc\ n) * (y\ (m + Suc\ n) - x\ (m + Suc\ n)))| \leq (1/3)$^$(n + 2) * 3/2$
(**is** *?lhs* ≤ -)
proof −
have *?lhs* $\leq (\sum m.\ |(1\ /\ 3)$ ^ $Suc\ (m + Suc\ n) * (y\ (m + Suc\ n) - x\ (m + Suc\ n))|)$
apply(*rule summable-rabs*,*rule summable-ignore-initial-segment*[*of* - *Suc n*])
using *Cantor-minus-abs-cantor*[*OF that*(*2*,*1*)] **by**(*simp add*: *abs-mult*)
also have ... $= (\sum m.\ (1\ /\ 3)$ ^ $Suc\ (m + Suc\ n) * |y\ (m + Suc\ n) - x\ (m + Suc\ n)|)$
by(*simp add*: *abs-mult*)
also have ... $\leq (\sum m.\ (1\ /\ 3)$ ^ $Suc\ (m + Suc\ n))$
apply(*rule suminf-le*)
using *space-Cantor-space-01*[*OF Cantor-minus-abs-cantor*[*OF that*(*2*,*1*)]]
apply *simp*
apply(*rule summable-ignore-initial-segment*[*of* - *Suc n*])
using *Cantor-minus-abs-cantor*[*OF that*(*2*,*1*)] **by** *auto*
also have ... $= (\sum m.\ (1\ /\ 3)$ ^$(m + Suc\ (Suc\ n)) * 1)$ **by** *simp*
also have ... $= (1/3)$^$(n + 2) * 3/(2::real)$
by(*simp only*: *pd.nsum-of-rK*[*of Suc* (*Suc n*)],*simp*)
finally show *?thesis* **.**
qed
thus *?thesis* **by** *simp*
qed
also have ... $= |(1\ /\ 3)$ ^ $Suc\ n * (x\ n - y\ n)| - |\sum m.\ (1\ /\ 3)$ ^ $Suc\ (m + Suc\ n) * (y\ (m + Suc\ n) - x\ (m + Suc\ n))|$
using ‹$x\ n \neq y\ n$› *space-Cantor-space-01*[*OF Cantor-minus-abs-cantor*[*OF that*],*of n*] **by**(*simp add*: *abs-mult*)

also have ... ≤ |(*1* / *3*) ^ *Suc n* ∗ (*x n* − *y n*) − (∑ *m*. (*1* / *3*) ^ *Suc* (*m* + *Suc n*) ∗ (*y* (*m* + *Suc n*) − *x* (*m* + *Suc n*)))|
by *simp*
also have ... = |(*1* / *3*) ^ *Suc n* ∗ (*x n* − *y n*) + (∑ *m*. (*1* / *3*) ^ *Suc* (*m* + *Suc n*) ∗ (*x* (*m* + *Suc n*) − *y* (*m* + *Suc n*)))|
proof −
have (∑ *m*. (*1* / *3*) ^ *Suc* (*m* + *Suc n*) ∗ (*x* (*m* + *Suc n*) − *y* (*m* + *Suc n*))) = (∑ *m*. − ((*1* / *3*) ^ *Suc* (*m* + *Suc n*) ∗ (*y* (*m* + *Suc n*) − *x* (*m* + *Suc n*))))
proof −
{ fix *nn* :: *nat*
have ⋀*r ra rb*. − ((− (*r*::*real*) + *ra*) / (*1* / *rb*)) = (− *ra* + *r*) / (*1* / *rb*)
by (*simp add*: *left-diff-distrib*)
then have − ((*y* (*Suc* (*n* + *nn*)) + − *x* (*Suc* (*n* + *nn*))) ∗ (*1* / *3*) ^ *Suc* (*Suc* (*n* + *nn*))) = (*x* (*Suc* (*n* + *nn*)) + − *y* (*Suc* (*n* + *nn*))) ∗ (*1* / *3*) ^ *Suc* (*Suc* (*n* + *nn*))
by *fastforce*
then have − ((*1* / *3*) ^ *Suc* (*nn* + *Suc n*) ∗ (*y* (*nn* + *Suc n*) − *x* (*nn* + *Suc n*))) = (*1* / *3*) ^ *Suc* (*nn* + *Suc n*) ∗ (*x* (*nn* + *Suc n*) − *y* (*nn* + *Suc n*))
by (*simp add*: *add.commute mult.commute*) **}**
then show *?thesis*
by *presburger*
qed
also have ... = − (∑ *m*. (*1* / *3*) ^ *Suc* (*m* + *Suc n*) ∗ (*y* (*m* + *Suc n*) − *x* (*m* + *Suc n*)))
apply(*rule suminf-minus*)
apply(*rule summable-ignore-initial-segment*[*of* - *Suc n*])
using *that* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... = |(∑ *m*. (*1* / *3*) ^ *Suc* (*m* + *Suc n*) ∗ (*x* (*m* + *Suc n*) − *y* (*m* + *Suc n*))) + (*1* / *3*) ^ *Suc n* ∗ (*x n* − *y n*)|
using *1* **by** *simp*
finally show *?thesis* **by** *simp*
qed
also have ... = |(∑ *m*. (*1*/*3*)^(*Suc* (*m* + *Suc n*)) ∗ (*x* (*m* + *Suc n*) − *y* (*m* + *Suc n*))) + (∑ *m*<*Suc n*. (*1*/*3*)^(*Suc m*) ∗ (*x m* − *y m*))|
using ‹∀ *i*<*n*. *x i* = *y i*› **by** *auto*
also have ... = |∑ *n*. (*1*/*3*)^(*Suc n*) ∗ (*x n* − *y n*)|
proof −
have (∑ *n*. (*1* / *3*) ^ *Suc n* ∗ (*x n* − *y n*)) = (∑ *m*. (*1* / *3*) ^ *Suc* (*m* + *Suc n*) ∗ (*x* (*m* + *Suc n*) − *y* (*m* + *Suc n*))) + (∑ *m*<*Suc n*. (*1* / *3*) ^ *Suc m* ∗ (*x m* − *y m*))
by(*rule suminf-split-initial-segment*) (*use that* **in** *simp*)
thus *?thesis* **by** *simp*
qed
also have ... = |(∑ *n*. (*1*/*3*)^(*Suc n*)∗ *x n* − (*1*/*3*)^(*Suc n*)∗ *y n*)|

```
      by (simp add: right-diff-distrib)
    also have ... = |Cantor-to-01 x − Cantor-to-01 y|
      using that by(simp add: suminf-diff Cantor-to-01-def)
    finally show ?thesis .
  qed
  have u2: (1/9) ∗ pd.product-dist x y ≤ (1/3) ^(Suc n) ∗ (1/2)
  proof −
    have pd.product-dist x y = (∑ m. (1/3)^m ∗ |x m − y m|)
      by(simp add: that pd-def)
     also have ... = (∑ m. (1/3)^(m + n) ∗ |x (m + n) − y (m + n)|) +
(∑ m<n. (1/3)^m ∗ |x m − y m|)
   using Cantor-minus-abs-cantor[OF that] by(auto intro!: suminf-split-initial-segment)
    also have ... = (∑ m. (1/3)^(m + n) ∗ |x (m + n) − y (m + n)|)
      using ‹∀ i<n. x i = y i›  by simp
    also have ... ≤ (∑ m. (1/3)^(m + n))
         using space-Cantor-space-01[OF Cantor-minus-abs-cantor[OF that]]
Cantor-minus-abs-cantor[OF that]
      by(auto intro!: suminf-le summable-ignore-initial-segment[of - n])
    also have ... =  (1 / 3) ^ n ∗ (3 / 2)
      using pd.nsum-of-rK[of n] by auto
    finally show ?thesis
      by auto
  qed
  from u1 u2 show ?thesis by simp
qed

have inj: inj-on Cantor-to-01 (space Cantor-space)
proof
  fix x y
  assume h:x ∈ space Cantor-space y ∈ space Cantor-space
         Cantor-to-01 x = Cantor-to-01 y
  then have pd.product-dist x y = 0
    using 2[OF h(1,2)] pd.Product-metric.nonneg[of x y]
    by simp
  thus x = y
    using pd.Product-metric.zero[of x y] h(1,2)
    by(simp add: space-Cantor-space)
qed

have closed: closedin m01.sub.mtopology (Cantor-to-01 ‘ (space Cantor-space))
  unfolding m01.sub.metric-closedin-iff-sequentially-closed
proof safe
  show a ∈ space Cantor-space =⇒ Cantor-to-01 a ∈ {0..1} for a
    using Cantor-to-01-image by auto
next
  fix xn x
 assume h:range xn ⊆ Cantor-to-01 ‘ space Cantor-space limitin m01.sub.mtopology
xn x sequentially
  have ⋀n. xn n ∈ {0..1}
```

using *h*(*1*) *measurable-space*[*OF Cantor-to-01-measurable*]
by (*metis* (*no-types*, *lifting*) *UNIV-I atLeastAtMost-borel image-subset-iff space-restrict-space2 subsetD*)
with *h*(*2*) **have** *xnC*:*m01.sub.MCauchy xn*
by(*auto intro*!: *m01.sub.convergent-imp-MCauchy*)
have ∀ *n*. ∃ *x*∈*space Cantor-space*. *xn n* = *Cantor-to-01 x* **using** *h*(*1*) **by** *auto*
then obtain *yn* **where** *hx*:⋀*n*. *yn n* ∈ *space Cantor-space* ⋀*n*. *xn n* = *Cantor-to-01* (*yn n*) **by** *metis*
have *pd.Product-metric.MCauchy yn*
unfolding *pd.Product-metric.MCauchy-def*
proof *safe*
fix *ε*
assume (*0* :: *real*) < *ε*
hence *0* < *ε* / *9* **by** *auto*
then obtain *N′* **where** ⋀*n m*. *n* ≥ *N′* ⟹ *m* ≥ *N′* ⟹ |*xn n* − *xn m*| < *ε* / *9*
using *xnC m01.sub.MCauchy-def xnC* **unfolding** *dist-real-def* **by** *blast*
thus ∃ *N*. ∀ *n n′*. *N* ≤ *n* ⟶ *N* ≤ *n′* ⟶ *pd.product-dist* (*yn n*) (*yn n′*) < *ε*
using *order.strict-trans1*[*OF 2*[*OF hx*(*1*) *hx*(*1*)],*of* - - *ε*/*9*] *hx*(*1*)
by(*auto intro*!: *exI*[**where** *x*=*N′*] *simp*: *hx*(*2*) *space-Cantor-space*)
qed(*use hx space-Cantor-space* **in** *auto*)
then obtain *y* **where** *y*:*limitin pd.Product-metric.mtopology yn y sequentially*
using *pd-mcomplete pd.Product-metric.mcomplete-def* **by** *blast*
hence *y* ∈ *space Cantor-space*
by (*simp add*: *pd.Product-metric.limitin-mspace space-Cantor-space*)
have *limitin m01.sub.mtopology xn* (*Cantor-to-01 y*) *sequentially*
unfolding *m01.sub.limit-metric-sequentially*
proof *safe*
show *Cantor-to-01 y* ∈ {*0*..*1*}
using *h*(*1*) *funcset-image*[*OF Cantor-to-01-image*] ‹*y* ∈ *space Cantor-space*› **by** *blast*
next
fix *ε*
assume (*0* :: *real*) < *ε*
then obtain *N* **where** ⋀*n*. *n* ≥ *N* ⟹ *pd.product-dist* (*yn n*) *y* < *ε* ⋀*n*. *n* ≥ *N* ⟹ *yn n* ∈ *UNIV* $\rightarrow_E$ {*0*, *1*}
using *y* **by**(*fastforce simp*: *pd.Product-metric.limit-metric-sequentially*)
with ‹⋀*n*. *xn n* ∈ {*0*..*1*}› **show** ∃ *N*. ∀ *n*≥*N*. *xn n* ∈ {*0*..*1*} ∧ *dist* (*xn n*) (*Cantor-to-01 y*) < *ε*
by(*auto intro*!: *exI*[**where** *x*=*N*] *order.strict-trans1*[*OF 1*[*OF hx*(*1*) ‹*y* ∈ *space Cantor-space*›]] *simp*: *submetric-def* ‹*0* < *ε*› *hx*(*2*) *dist-real-def*)
qed
hence *Cantor-to-01 y* = *x*
using *h*(*2*) **by**(*auto dest*: *m01.sub.limitin-metric-unique*)
with ‹*y* ∈ *space Cantor-space*› **show** *x* ∈ *Cantor-to-01* ‘ *space Cantor-space*
by *auto*
qed

have *open-map*:*open-map pd.Product-metric.mtopology* (*subtopology m01.sub.mtopology* (*Cantor-to-01* ‘ (*space Cantor-space*))) *Cantor-to-01*
proof −
have *open-map* (*mtopology-of pd.Product-metric.Self*) (*subtopology* (*mtopology-of m01.sub.Self*) (*Cantor-to-01* ‘ *mspace pd.Product-metric.Self*)) *Cantor-to-01*
proof(*rule Metric-space-open-map-from-dist*)
fix *x ε*
assume (*0* :: *real*) < *ε* *x* ∈ *mspace pd.Product-metric.Self*
then have *x* ∈ (*UNIV* :: *nat set*) $\rightarrow_E$ {*0*, *1*::*real*}
by *simp*
show ∃ *δ*>*0*. ∀ *y*∈*mspace pd.Product-metric.Self*. *mdist m01.sub.Self* (*Cantor-to-01 x*) (*Cantor-to-01 y*) < *δ* ⟶ *mdist pd.Product-metric.Self x y* < *ε*
unfolding *pd.Product-metric.mspace-Self pd.Product-metric.mdist-Self m01.sub.mdist-Self*
proof(*safe intro*!: *exI*[**where** *x*=*ε*/*9*])
fix *y*
assume *h*:*y* ∈ (*UNIV* :: *nat set*) $\rightarrow_E$ {*0*, *1*::*real*} *dist* (*Cantor-to-01 x*) (*Cantor-to-01 y*) < *ε* / *9*
then have *sc*:*x* ∈ *space Cantor-space y* ∈ *space Cantor-space*
using ‹*x* ∈ *UNIV* $\rightarrow_E$ {*0*, *1*}› **by**(*simp-all add*: *space-Cantor-space*)
have |*Cantor-to-01 x* − *Cantor-to-01 y*| < *ε* / *9*
using *h* **by**(*simp add*: *dist-real-def*)
with *2*[*OF sc*] **show** *pd.product-dist x y* < *ε*
by *simp*
qed (*use* ‹*ε* > *0*› **in** *auto*)
qed(*use Cantor-to-01-image space-Cantor-space* **in** *auto*)
thus *?thesis*
by (*simp add*: *mtopology-of-def space-Cantor-space*)
qed
have *Cantor-to-01* ‘ *A* ∈ *sets* (*restrict-space borel* {*0*..*1*}) **if** *A* ∈ *sets Cantor-space* **for** *A*
using *open-map-preserves-sets′*[*of pd.Product-metric.mtopology m01.sub.mtopology Cantor-to-01 A*] *borel-of-closed*[*OF closed*] *inj sets-Cantor open-map that mpd-top* ‹*restrict-space borel* {*0*..*1*} = *borel-of m01.sub.mtopology*›
by (*simp add*: *space-Cantor-space*)
with *inj* **show** *inj-on Cantor-to-01* (*space Cantor-space*)
and*A* ∈ *sets Cantor-space* ⟹ *Cantor-to-01* ‘ *A* ∈ *sets* (*restrict-space borel* {*0*..*1*})
by *simp-all*
qed

Next, we construct measurable embedding from $[0,1]$ to $0,1^{\mathbb{N}}$.

define *to-Cantor-from-01* :: *real* ⇒ *nat* ⇒ *real* **where**
to-Cantor-from-01 ≡ (*λr n*. *if r* = *1 then 1 else real-of-int* (⌊*2*^(*Suc n*) ∗ *r*⌋ *mod 2*))

to-Cantor-from-01 is a measurable injective embedding into Cantor space.

have *to-Cantor-from-01-image′*: *to-Cantor-from-01 r n* ∈ {*0*,*1*} **for** *r n*

unfolding *to-Cantor-from-01-def* **by** *auto*
have *to-Cantor-from-01-image''*: $\bigwedge r\ n.\ 0 \le$ *to-Cantor-from-01 r n* $\bigwedge r\ n.$ *to-Cantor-from-01 r n* ≤ 1
by (*auto simp add*: *to-Cantor-from-01-def*)
have *to-Cantor-from-01-image*: *to-Cantor-from-01* $\in \{0..1\} \to$ *space Cantor-space*
using *to-Cantor-from-01-image'* **by**(*auto simp*: *space-Cantor-space*)
have *to-Cantor-from-01-measurable*:
to-Cantor-from-01 $\in$ *restrict-space borel* $\{0..1\} \to_M$ *Cantor-space*
unfolding *to-Cantor-from-01-def Cantor-space-def*
by(*auto intro*!: *measurable-restrict-space3 measurable-abs-UNIV*)
have *to-Cantor-from-01-summable*[*simp*]:
summable ($\lambda n.\ (1/2)$^$n *$ *to-Cantor-from-01 r n*) **for** *r*
proof(*rule summable-comparison-test'*[**where** $g=\lambda n.\ (1/2)$^ n])
show *norm* ((*1* / *2*) ^ $n *$ *to-Cantor-from-01 r n*) $\le$ (*1* / *2*) ^ *n* **for** *n*
using *to-Cantor-from-01-image'*[*of r n*] **by** *auto*
qed *simp*

have *to-Cantor-from-sumn'*: ($\sum i<n.\ (1/2)$^($Suc\ i$)$*$*to-Cantor-from-01 r i*) $\le$ *r*
r $-$ ($\sum i<n.\ (1/2)$^($Suc\ i$)$*$*to-Cantor-from-01 r i*) $<$ $(1/2)$^n
to-Cantor-from-01 r n $= 1 \longleftrightarrow (1/2)$^($Suc\ n$) $\le r -$ ($\sum i<n.\ (1/2)$^(Suc i)$*$*to-Cantor-from-01 r i*)
to-Cantor-from-01 r n $= 0 \longleftrightarrow r -$ ($\sum i<n.\ (1/2)$^($Suc\ i$)$*$*to-Cantor-from-01 r i*) $<$ $(1/2)$^($Suc\ n$) **if** *assms*:$r \in \{0..<1\}$ **for** *r n*
proof $-$
let *?f* = *to-Cantor-from-01 r*
have *f-simp*: *?f l* = *real-of-int* ($\lfloor$ *2*^($Suc\ l$) $*$ $r\rfloor$ *mod 2*) **for** *l*
using *assms* **by**(*simp add*: *to-Cantor-from-01-def*)
define *S* **where** $S = (\lambda n.\ \sum i<n.\ (1/2)$^($Suc\ i$)$*$*?f i*)
have *SSuc*:*S* (*Suc k*) = *S k* + $(1/2)$^($Suc\ k$) $*$ *to-Cantor-from-01 r k* **for** *k*
by(*simp add*: *S-def*)
have *Sfloor*: $\lfloor$*2*^($Suc\ m$) $*$ ($l - S\ m$)$\rfloor$ *mod 2* = $\lfloor$*2*^($Suc\ m$) $*$ $l\rfloor$ *mod 2* **for** *l m*
proof $-$
have $\exists z.$ *2*^($Suc\ m$) $*$ (($1/2$)^($Suc\ k$) $*$ *?f k*) = *2*$*$*real-of-int z* **if** $k < m$ **for** *k*
proof $-$
have *0*:(*2*::*real*) ^ *m* $*$ (*1* / *2*) ^ *k* = *2* $*$ *2*^($m-k-1$)
using *that* **by** (*simp add*: *power-diff-conv-inverse*)
consider *?f k* = *0* | *?f k* = *1*
using *to-Cantor-from-01-image'*[*of r k*] **by** *auto*
thus *?thesis*
apply *cases* **using** *that 0* **by** *auto*
qed
then obtain *z* **where** $\bigwedge k.\ k < m \Longrightarrow$ *2*^($Suc\ m$) $*$ (($1/2$)^($Suc\ k$) $*$ *?f k*) = *2*$*$*real-of-int* (*z k*)
by *metis*
hence *Sm*: *2*^($Suc\ m$) $*$ *S m* = *real-of-int* (*2* $*$ ($\sum k<m.$ (*z k*)))

by(*auto simp*: *S-def sum-distrib-left*)
have $\lfloor 2\hat{}(Suc\ m) * (l - S\ m)\rfloor$ *mod 2* = $\lfloor 2\hat{}(Suc\ m) * l - 2\hat{}(Suc\ m) * S\ m\rfloor$ *mod 2*
by (*simp add*: *right-diff-distrib*)
also have ... = $\lfloor 2\hat{}(Suc\ m) * l\rfloor$ *mod 2*
unfolding *Sm*
by(*simp only*: *floor-diff-of-int*) *presburger*
finally show *?thesis* **.**
qed
have $S\ n \le r \wedge r - S\ n < (1/2)\hat{}n \wedge (?f\ n = 1 \longleftrightarrow (1/2)\hat{}(Suc\ n) \le r - S\ n) \wedge (?f\ n = 0 \longleftrightarrow r - S\ n < (1/2)\hat{}(Suc\ n))$
proof(*induction n*)
case *0*
then show *?case*
using *assms* **by**(*auto simp*: *S-def to-Cantor-from-01-def*) *linarith+*
next
case (*Suc n*)
hence *ih*: $S\ n \le r$ $r - S\ n < (1\ /\ 2)\ \hat{}\ n$
$?f\ n = 1 \Longrightarrow (1\ /\ 2)\ \hat{}\ Suc\ n \le r - S\ n$
$?f\ n = 0 \Longrightarrow r - S\ n < (1\ /\ 2)\ \hat{}\ Suc\ n$
by *simp-all*
have *SSuc'*: $?f\ n = 0 \wedge S\ (Suc\ n) = S\ n \vee ?f\ n = 1 \wedge S\ (Suc\ n) = S\ n + (1/2)\hat{}(Suc\ n)$
using *to-Cantor-from-01-image'*[*of r n*] **by**(*simp add*: *SSuc*)
have *goal1*: $S\ (Suc\ n) \le r$
using *SSuc' ih*(*1*) *ih*(*3*) **by** *auto*
have *goal2*: $r - S\ (Suc\ n) < (1\ /\ 2)\ \hat{}\ Suc\ n$
using *SSuc' ih*(*4*) *ih*(*2*) **by** *auto*
have *goal3-1*: $(1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n) \le r - S\ (Suc\ n)$ **if** $?f\ (Suc\ n) = 1$
proof(*rule ccontr*)
assume $\neg\ (1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n) \le r - S\ (Suc\ n)$
then have $r - S\ (Suc\ n) < (1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n)$ **by** *simp*
hence *h*: $2\ \hat{}\ Suc\ (Suc\ n) * (r - S\ (Suc\ n)) < 1$
using *mult-less-cancel-left-pos*[*of* $2\ \hat{}\ Suc\ (Suc\ n)$ $r - S\ (Suc\ n)$ $(1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n)$]
by (*simp add*: *power-one-over*)
moreover have $0 \le 2\ \hat{}\ Suc\ (Suc\ n) * (r - S\ (Suc\ n))$
using *goal1* **by** *simp*
ultimately have $\lfloor 2\ \hat{}\ Suc\ (Suc\ n) * (r - S\ (Suc\ n))\rfloor = 0$
by *linarith*
thus *False*
using *that*[*simplified f-simp*] *Sfloor*[*of Suc n r*]
by *fastforce*
qed
have *goal3-2*: $?f\ (Suc\ n) = 1$ **if** $(1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n) \le r - S\ (Suc\ n)$
proof −
have $1 \le 2\ \hat{}\ Suc\ (Suc\ n) * (r - S\ (Suc\ n))$
using *that*[*simplified f-simp*] *mult-le-cancel-left-pos*[*of* $2\ \hat{}\ Suc\ (Suc\ n)$ $(1\ /\ 2)\ \hat{}\ Suc\ (Suc\ n)$ $r - S\ (Suc\ n)$]

by (*simp add*: *power-one-over*)
moreover have *2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*)) < *2*
using *mult-less-cancel-left-pos*[*of 2* ^ *Suc* (*Suc n*) *r* − *S* (*Suc n*) (*1* / *2*) ^ *Suc n*] *goal2*
by (*simp add*: *power-one-over*)
ultimately have ⌊*2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*))⌋ = *1*
by *linarith*
thus *?thesis*
using *Sfloor*[*of Suc n r*] **by**(*auto simp*: *f-simp*)
qed
have *goal4-1*: *r* − *S* (*Suc n*) < (*1* / *2*) ^ *Suc* (*Suc n*) **if** *?f* (*Suc n*) = *0*
proof(*rule ccontr*)
assume ¬ *r* − *S* (*Suc n*) < (*1* / *2*) ^ *Suc* (*Suc n*)
then have (*1* / *2*) ^ *Suc* (*Suc n*) ≤ *r* − *S* (*Suc n*) **by** *simp*
hence *1* ≤ *2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*))
using *mult-le-cancel-left-pos*[*of 2* ^ *Suc* (*Suc n*) (*1* / *2*) ^ *Suc* (*Suc n*) *r* − *S* (*Suc n*)]
by (*simp add*: *power-one-over*)
moreover have *2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*)) < *2*
using *mult-less-cancel-left-pos*[*of 2* ^ *Suc* (*Suc n*) *r* − *S* (*Suc n*) (*1* / *2*) ^ *Suc n*] *goal2*
by (*simp add*: *power-one-over*)
ultimately have ⌊*2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*))⌋ = *1*
by *linarith*
thus *False*
using *that Sfloor*[*of Suc n r*] **by**(*auto simp*: *f-simp*)
qed
have *goal4-2*: *?f* (*Suc n*) = *0* **if** *r* − *S* (*Suc n*) < (*1* / *2*) ^ *Suc* (*Suc n*)
proof −
have *h*:*2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*)) < *1*
using *mult-less-cancel-left-pos*[*of 2* ^ *Suc* (*Suc n*) *r* − *S* (*Suc n*) (*1* / *2*) ^ *Suc* (*Suc n*)] *that*
by (*simp add*: *power-one-over*)
moreover have *0* ≤ *2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*))
using *goal1* **by** *simp*
ultimately have ⌊*2* ^ *Suc* (*Suc n*) ∗ (*r* − *S* (*Suc n*))⌋ = *0*
by *linarith*
thus *?thesis*
using *Sfloor*[*of Suc n r*] **by**(*auto simp*: *f-simp*)
qed
show *?case*
using *goal1 goal2 goal3-1 goal3-2 goal4-1 goal4-2* **by** *blast*
qed
thus (∑ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) ≤ *r*
and *r* − (∑ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) < (*1*/*2*)^*n*
and *to-Cantor-from-01 r n* = *1* ⟷ (*1*/*2*)^(*Suc n*) ≤ *r* − (∑ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*)
and *to-Cantor-from-01 r n* = *0* ⟷ *r* − (∑ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) < (*1*/*2*)^(*Suc n*)

by(*simp-all add*: *S-def*)
qed
have *to-Cantor-from-sumn*: ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) ≤ *r*
r − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) ≤ (*1*/*2*)^*n*
to-Cantor-from-01 r n = *1* ⟷ (*1*/*2*)^(*Suc n*) ≤ *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*)
to-Cantor-from-01 r n = *0* ⟷ *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) < (*1*/*2*)^(*Suc n*) **if** *assms*:*r* ∈ {*0*..*1*} **for** *r n*
proof −
have *nsum*:($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)) = *1* − (*1* / (*2*::*real*)) ^ *n*
using *one-diff-power-eq*[*of 1*/(*2*::*real*) *n*] **by**(*auto simp*: *sum-divide-distrib*[*symmetric*])

consider *r* = *1* | *r* ∈{*0*..<*1*} **using** *assms* **by** *fastforce*
hence ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) ≤ *r* ∧ *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) ≤ (*1*/*2*)^*n* ∧ (*to-Cantor-from-01 r n* = *1* ⟷ (*1*/*2*)^(*Suc n*) ≤ *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*)) ∧ (*to-Cantor-from-01 r n* = *0* ⟷ *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) < (*1*/*2*)^(*Suc n*))
proof *cases*
case *1*
then show *?thesis*
using *nsum* **by**(*auto simp*: *to-Cantor-from-01-def*)
next
case *2*
from *to-Cantor-from-sumn′*[*OF this*]
show *?thesis*
using *less-eq-real-def* **by** *blast*
qed
thus ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) ≤ *r*
and *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) ≤ (*1*/*2*)^*n*
and *to-Cantor-from-01 r n* = *1* ⟷ (*1*/*2*)^(*Suc n*) ≤ *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*)
and *to-Cantor-from-01 r n* = *0* ⟷ *r* − ($\sum$ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) < (*1*/*2*)^(*Suc n*)
by *simp-all*
qed

have *to-Cantor-from-sum*: ($\sum$ *n*. (*1*/*2*)^(*Suc n*)∗*to-Cantor-from-01 r n*) = *r* **if** *assms*:*r* ∈ {*0*..*1*} **for** *r*
proof −
have *1*:*r* ≤ ($\sum$ *n*. (*1*/*2*)^(*Suc n*)∗*to-Cantor-from-01 r n*)
proof −
have *0*:*r* ≤ (*1* / *2*) ^ *n* + ($\sum$ *n*. (*1*/*2*)^(*Suc n*)∗*to-Cantor-from-01 r n*) **for** *n*
proof −
have *r* ≤ (*1* / *2*) ^ *n* + ($\sum$ *i*<*n*. (*1* / *2*) ^ *Suc i* ∗ *to-Cantor-from-01 r i*)
using *to-Cantor-from-sumn*(*2*)[*OF assms*,*of n*] **by** *auto*
also have ... ≤ (*1* / *2*) ^ *n* + ($\sum$ *n*. (*1*/*2*)^(*Suc n*)∗*to-Cantor-from-01 r*

n)
using *to-Cantor-from-01-image''*[*of r*] **by**(*auto intro*!: *sum-le-suminf*)
finally show *?thesis* **.**
qed
have *00*:∃ *no*. ∀ *n*≥*no*. (*1* / *2*) ^ *n* < *r* **if** *r*>*0* **for** *r* :: *real*
proof −
obtain *n0* **where** (*1* / *2*) ^ *n0* < *r*
using *reals-power-lt-ex*[*of - 2* :: *real*,*OF* ‹*r*>*0*›] **by** *auto*
thus *?thesis*
using *order.strict-trans1*[*OF power-decreasing*[*of n0 - 1*/*2*::*real*]]
by(*auto intro*!: *exI*[**where** *x*=*n0*])
qed
show *?thesis*
apply(*rule Lim-bounded2*[**where** *f*=λ*n*. (*1* / *2*) ^ *n* + (∑ *n*. (*1*/*2*)^(*Suc n*)∗*to-Cantor-from-01 r n*) **and** *N*=*0*])
using *0 00* **by**(*auto simp*: *LIMSEQ-iff*)
qed
have *2*:(∑ *n*. (*1*/*2*)^(*Suc n*)∗*to-Cantor-from-01 r n*) ≤ *r*
using *to-Cantor-from-sumn*[*OF assms*] **by**(*auto intro*!: *suminf-le-const*)
show *?thesis*
using *1 2* **by** *simp*
qed
have *to-Cantor-from-sum'*: (∑ *i*<*n*. (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 r i*) = *r* − (∑ *m*. (*1*/*2*)^(*Suc* (*m* + *n*))∗*to-Cantor-from-01 r* (*m* + *n*)) **if** *assms*:*r* ∈ {*0*..*1*} **for** *r n*
using *suminf-minus-initial-segment*[*of* λ*n*. (*1* / *2*) ^*Suc n* ∗ *to-Cantor-from-01 r n n*] *to-Cantor-from-sum*[*OF assms*]
by *auto*

have *to-Cantor-from-01-exist0*: ∀ *n*.∃ *k*≥*n*. *to-Cantor-from-01 r k* = *0* **if** *assms*:*r* ∈ {*0*..<*1*} **for** *r*
proof(*rule ccontr*)
assume ¬ (∀ *n*.∃ *k*≥*n*. *to-Cantor-from-01 r k* = *0*)
then obtain *n0* **where** *hn0*:
⋀*k*. *k* ≥ *n0* ⟹ *to-Cantor-from-01 r k* = *1*
using *to-Cantor-from-01-image'*[*of r*] **by** *auto*
define *n* **where** *n* = *Min* {*i*. *i* ≤ *n0* ∧ (∀ *k*≥*i*. *to-Cantor-from-01 r k* = *1*)}
have *n0in*: *n0* ∈ {*i*. *i* ≤ *n0* ∧ (∀ *k*≥*i*. *to-Cantor-from-01 r k* = *1*)}
using *hn0* **by** *auto*
have *hn*:*n* ≤ *n0* ⋀*k*. *k* ≥ *n* ⟹ *to-Cantor-from-01 r k* = *1*
using *n0in Min-in*[*of* {*i*. *i* ≤ *n0* ∧ (∀ *k*≥*i*. *to-Cantor-from-01 r k* = *1*)}]
by(*auto simp*: *n-def*)
show *False*
proof(*cases n*)
case *0*
then have *r* = (∑ *n*. (*1* / *2*) ^ *Suc n*)
using *to-Cantor-from-sum*[*of r*] *assms hn*(*2*) **by** *simp*
also have ... = *1*
using *nsum-of-r'*[*of 1*/*2 1 1*] **by** *auto*

finally show *?thesis*
using *assms* **by** *auto*
next
case *eqn*:(*Suc n'*)
have *to-Cantor-from-01 r n' = 0*
proof(*rule ccontr*)
assume *to-Cantor-from-01 r n'* $\neq$ *0*
then have *to-Cantor-from-01 r n' = 1*
using *to-Cantor-from-01-image'*[*of r n'*] **by** *auto*
hence $n' \in \{i.\ i \leq n0 \wedge (\forall k \geq i.$ *to-Cantor-from-01 r k = 1*$)\}$
using *hn eqn not-less-eq-eq order-antisym-conv* **by** *fastforce*
hence $n \leq n'$
using *Min.coboundedI*[*of* $\{i.\ i \leq n0 \wedge (\forall k \geq i.$ *to-Cantor-from-01 r k =* *1*$)\}$ *n'*]
by(*simp add*: *n-def*)
thus *False*
using *eqn* **by** *simp*
qed
hence *le1*:*r* − ($\sum$ *i*<*n'*. (*1* / *2*) ^ *Suc i* ∗ *to-Cantor-from-01 r i*) < (*1* / *2*) ^ *n*
using *to-Cantor-from-sumn'*(*4*)[*OF assms*,*of n'*] **by** (*simp add*: *eqn*)
have *r* − ($\sum$ *i*<*n'*. (*1* / *2*) ^ *Suc i* ∗ *to-Cantor-from-01 r i*) = (*1* / *2*) ^ *n*
(**is** *?lhs* = -)
proof −
have *?lhs* = ($\sum$ *m*. (*1*/*2*)^(*m* + *Suc n'*)∗*to-Cantor-from-01 r* (*m* + *n'*))
using *to-Cantor-from-sum'*[*of r n'*] *assms* **by** *simp*
also have ... = ($\sum$ *m*. (*1*/*2*)^(*m* + *Suc n*)∗*to-Cantor-from-01 r* (*m* + *n*))
proof −
have ($\sum$ *n*. (*1* / *2*) ^(*Suc n* + *Suc n'*) ∗ *to-Cantor-from-01 r* (*Suc n* + *n'*)) = ($\sum$ *m*. (*1* / *2*) ^(*m* + *Suc n'*) ∗ *to-Cantor-from-01 r* (*m* + *n'*)) − (*1* / *2*) ^(*0* + *Suc n'*) ∗ *to-Cantor-from-01 r* (*0* + *n'*)
by(*rule suminf-split-head*) (*auto intro*!: *summable-ignore-initial-segment*)
thus *?thesis*
using ‹*to-Cantor-from-01 r n' = 0*› **by**(*simp add*: *eqn*)
qed
also have ... = ($\sum$ *m*. (*1*/*2*)^(*m* + *Suc n*))
using *hn* **by** *simp*
also have ... = (*1* / *2*) ^ *n*
using *nsum-of-r'*[*of 1*/*2 Suc n 1*,*simplified*] **by** *simp*
finally show *?thesis* **.**
qed
with *le1* **show** *False*
by *simp*
qed
qed
have *to-Cantor-from-01-if-exist0*: *to-Cantor-from-01* ($\sum$ *n*. (*1* / *2*) ^ *Suc n* ∗ *a n*) = *a* **if** *assms*:$\bigwedge$*n*. *a n* ∈ {*0*,*1*} ∀ *n*.∃ *k*≥*n*. *a k* = *0* **for** *a*
proof
fix *n*

have [*simp*]: *summable* ($\lambda n.$ (*1* / *2*) ^ *n* ∗ *a n*)
proof(*rule summable-comparison-test'*[**where** *g*=$\lambda n.$ (*1*/*2*)^ *n*])
show *norm* ((*1* / *2*) ^ *n* ∗ *a n*) ≤ (*1* / *2*) ^ *n* **for** *n*
using *assms*(*1*)[*of n*] **by** *auto*
qed *simp*
let *?r* = ∑ *n.* (*1* / *2*) ^ *Suc n* ∗ *a n*
have *?r* ∈ {*0..1*}
using *assms*(*1*) *space-Cantor-space-01*[*of a*,*simplified space-Cantor-space*] *nsum-of-r-leq*[*of 1/2 a 1 1 0*]
by *auto*
show *to-Cantor-from-01 ?r n* = *a n*
proof(*rule less-induct*)
fix *x*
assume *ih*:*y* < *x* ⟹ *to-Cantor-from-01 ?r y* = *a y* **for** *y*
have *eq1*:*?r* − (∑ *i*<*x.* (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 ?r i*) = (∑ *n.* (*1*/*2*)^(*Suc* (*n* + *x*))∗ *a* (*n* + *x*))
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (∑ *n.* (*1* / *2*) ^ *Suc* (*n* + *x*) ∗ *a* (*n* + *x*)) + (∑ *i*<*x.* (*1*/*2*)^(*Suc i*)∗ *a i*) − (∑ *i*<*x.* (*1*/*2*)^(*Suc i*)∗*to-Cantor-from-01 ?r i*)
using *suminf-split-initial-segment*[*of* $\lambda n.$ (*1* / *2*) ^(*Suc n*) ∗ *a n x*] **by** *simp*
also have ... = (∑ *n.* (*1* / *2*) ^ *Suc* (*n* + *x*) ∗ *a* (*n* + *x*)) + (∑ *i*<*x.* (*1*/*2*)^(*Suc i*)∗ *a i*) − (∑ *i*<*x.* (*1*/*2*)^(*Suc i*)∗*a i*)
using *ih* **by** *simp*
finally show *?thesis* **by** *simp*
qed
define *Sn* **where** *Sn* = (∑ *n.* (*1*/*2*)^(*Suc* (*n* + *x*))∗ *a* (*n* + *x*))
define *Sn'* **where** *Sn'* = (∑ *n.* (*1*/*2*)^(*Suc* (*n* + (*Suc x*)))∗ *a* (*n* + (*Suc x*)))
have *SnSn'*:*Sn* = (*1*/*2*)^(*Suc x*) ∗ *a x* + *Sn'*
using *suminf-split-head*[*of* $\lambda n.$ (*1*/*2*)^(*Suc* (*n* + *x*))∗ *a* (*n* + *x*),*OF summable-ignore-initial-segment*]
by(*auto simp*: *Sn-def Sn'-def*)
have *hsn*:*0* ≤ *Sn'* *Sn'* < (*1*/*2*)^(*Suc x*)
proof −
show *0* ≤ *Sn'*
unfolding *Sn'-def*
by(*rule suminf-nonneg*,*rule summable-ignore-initial-segment*) (*use assms*(*1*) *space-Cantor-space-01*[*of a*,*simplified space-Cantor-space*] **in** *fastforce*)+
next
have ∃ *n'*≥*Suc x. a n'* < *1*
using *assms* **by** *fastforce*
thus *Sn'* < (*1*/*2*)^(*Suc x*)
using *nsum-of-r-le*[*of 1/2 a 1 Suc x Suc* (*Suc x*)] *assms*(*1*) *space-Cantor-space-01*[*of a*,*simplified space-Cantor-space*]
by(*auto simp*: *Sn'-def*)
qed
have *goal1*: *to-Cantor-from-01 ?r x* = *1* ⟷ *a x* = *1*

```
    proof −
      have to-Cantor-from-01 ?r x = 1 ⟷ (1 / 2) ^ Suc x ≤ Sn
        using to-Cantor-from-sumn(3)[OF ‹?r ∈ {0..1}›] eq1
        by(fastforce simp: Sn-def)
      also have ... ⟷ (1 / 2) ^ Suc x ≤ (1/2)^(Suc x) ∗ a x + Sn′
        by(simp add: SnSn′)
      also have ... ⟷ a x = 1
      proof −
        have a x = 1 if (1 / 2) ^ Suc x ≤ (1/2)^(Suc x) ∗ a x + Sn′
        proof(rule ccontr)
          assume a x ≠ 1
          then have a x = 0
            using assms(1) by auto
          hence (1 / 2) ^ Suc x ≤ Sn′
            using that by simp
          thus False
            using hsn by auto
        qed
        thus ?thesis
          by(auto simp: hsn)
      qed
      finally show ?thesis .
    qed
    have goal2: to-Cantor-from-01 ?r x = 0 ⟷ a x = 0
    proof −
      have to-Cantor-from-01 ?r x = 0 ⟷ Sn < (1 / 2) ^ Suc x
        using to-Cantor-from-sumn(4)[OF ‹?r ∈ {0..1}›] eq1
        by(fastforce simp: Sn-def)
      also have ... ⟷ (1/2)^(Suc x) ∗ a x + Sn′ < (1 / 2) ^ Suc x
        by(simp add: SnSn′)
      also have ... ⟷ a x = 0
      proof −
        have a x = 0 if (1/2)^(Suc x) ∗ a x + Sn′ < (1 / 2) ^ Suc x
        proof(rule ccontr)
          assume a x ≠ 0
          then have a x = 1
            using assms(1) by auto
          thus False
            using that hsn by auto
        qed
        thus ?thesis
          using hsn by auto
      qed
      finally show ?thesis .
    qed
    show to-Cantor-from-01 ?r x = a x
      using goal1 goal2 to-Cantor-from-01-image′[of ?r x] by auto
  qed
qed
```

have *to-Cantor-from-01-sum-of-to-Cantor-from-01*: *to-Cantor-from-01* ($\sum$ *n*. (*1* / *2*) ^ *Suc n* ∗ *to-Cantor-from-01 r n*) = *to-Cantor-from-01 r* **if** *assms*:*r* ∈ {*0..1*} **for** *r*
  proof −
    consider *r* = *1* | *r* ∈ {*0..<1*}
      using *assms* **by** *fastforce*
    then show *?thesis*
    proof *cases*
      case *1*
      then show *?thesis*
        using *nsum-of-r′*[*of 1/2 1 1*]
        by(*auto simp*: *to-Cantor-from-01-def*)
    next
      case *2*
    from *to-Cantor-from-01-if-exist0*[*OF to-Cantor-from-01-image′ to-Cantor-from-01-exist0*[*OF this*]]
      show *?thesis* **.**
    qed
  qed
  have *to-Cantor-from-01-inj*: *inj-on to-Cantor-from-01* (*space* (*restrict-space borel* {*0..1*}))
  proof
    fix *x y* :: *real*
    assume *x* ∈ *space* (*restrict-space borel* {*0..1*}) *y* ∈ *space* (*restrict-space borel* {*0..1*})
      and *h*:*to-Cantor-from-01 x* = *to-Cantor-from-01 y*
    then have *xyin*:*x* ∈ {*0..1*} *y* ∈ {*0..1*}
      by *simp-all*
    show *x* = *y*
      using *to-Cantor-from-sum*[*OF xyin*(*1*)] *to-Cantor-from-sum*[*OF xyin*(*2*)] *h*
      by *simp*
  qed
  have *to-Cantor-from-01-preserves-sets*: *to-Cantor-from-01* ‘ *A* ∈ *sets Cantor-space* **if** *assms*: *A* ∈ *sets* (*restrict-space borel* {*0..1*}) **for** *A*
  proof −
    define *f* :: (*nat* ⇒ *real*) ⇒ *real* **where** *f* ≡ (λ*x*. $\sum$ *n*. (*1*/*2*)^(*Suc n*)∗ *x n*)
    have *f-meas*:*f* ∈ *Cantor-space* $\rightarrow_M$ *restrict-space borel* {*0..1*}
    proof −
      have *f* ∈ *borel-measurable Cantor-space*
        unfolding *Cantor-to-01-def f-def*
      proof(*rule borel-measurable-suminf*)
        fix *n*
        have (λ*x*. *x n*) ∈ *Cantor-space* $\rightarrow_M$ *restrict-space borel* {*0*, *1*}
          by(*simp add*: *Cantor-space-def*)
        hence (λ*x*. *x n*) ∈ *borel-measurable Cantor-space*
          by(*simp add*: *measurable-restrict-space2-iff*)
        thus (λ*x*. (*1* / *2*) ^ *Suc n* ∗ *x n*) ∈ *borel-measurable Cantor-space*
          by *simp*
      qed

```
    moreover have 0 ≤ f x f x ≤ 1 if x ∈ space Cantor-space for x
    proof −
      have [simp]:summable (λn. (1/2)^n∗ x n)
      proof(rule summable-comparison-test′[where g=λn. (1/2)^ n])
        show norm ((1 / 2) ^ n ∗ x n) ≤ (1 / 2) ^ n for n
          using that by simp
      qed simp
      show 0 ≤ f x
        using that by(auto intro!: suminf-nonneg simp: f-def)
      show f x ≤ 1
      proof −
        have f x ≤ (∑ n. (1/2)^(Suc n))
          using that by(auto intro!: suminf-le simp: f-def)
        also have ... = 1
          using nsum-of-r′[of 1/2 1 1] by simp
        finally show ?thesis .
      qed
    qed
    ultimately show ?thesis
      by(auto intro!: measurable-restrict-space2)
  qed
  have image-sets:to-Cantor-from-01 ‘ (space (restrict-space borel {0..1})) ∈
sets Cantor-space
    (is ?A ∈ -)
  proof −
    have ?A ⊆ space Cantor-space
      using to-Cantor-from-01-image by auto
    have comple-sets:(Π_E i∈ UNIV. {0,1}) − ?A ∈ sets Cantor-space
    proof −
     have eq1:?A = {λn. 1} ∪ {x. (∀ n. x n ∈ {0,1}) ∧ (∀ n. ∃ k≥n. x k = 0)}
      proof
       show ?A ⊆ {λn. 1} ∪ {x. (∀ n. x n ∈ {0, 1}) ∧ (∀ n. ∃ k≥n. x k = 0)}
        proof
          fix x
          assume x ∈ ?A
          then obtain r where hr:r ∈ {0..1} x = to-Cantor-from-01 r
            by auto
          then consider r = 1 | r ∈ {0..<1} by fastforce
          thus x ∈ {λn. 1} ∪ {x. (∀ n. x n ∈ {0,1}) ∧ (∀ n. ∃ k≥n. x k = 0)}
          proof cases
            case 1
            then show ?thesis
              by(simp add: hr(2) to-Cantor-from-01-def)
          next
            case 2
            from to-Cantor-from-01-exist0[OF this] to-Cantor-from-01-image′
            show ?thesis by(auto simp: hr(2))
          qed
        qed
```

next
show $\{\lambda n.\ 1\} \cup \{x.\ (\forall n.\ x\ n \in \{0,\ 1\}) \wedge (\forall n.\ \exists k \geq n.\ x\ k = 0)\} \subseteq ?A$
proof
fix $x :: nat \Rightarrow real$
assume $x \in \{\lambda n.\ 1\} \cup \{x.\ (\forall n.\ x\ n \in \{0,1\}) \wedge (\forall n.\ \exists k \geq n.\ x\ k = 0)\}$
then consider $x = (\lambda n.\ 1) \mid (\forall n.\ x\ n \in \{0,1\}) \wedge (\forall n.\ \exists k \geq n.\ x\ k = 0)$
by *auto*
thus $x \in ?A$
proof *cases*
case *1*
then show *?thesis*
by(*auto intro*!: *image-eqI*[**where** *x=1*] *simp*: *to-Cantor-from-01-def*)
next
case *2*
hence $\bigwedge n.\ 0 \leq x\ n$ $\bigwedge n.\ x\ n \leq 1$
by (*metis dual-order.refl empty-iff insert-iff zero-less-one-class.zero-le-one*)+
with *2 to-Cantor-from-01-if-exist0*[*of x*] *nsum-of-r-leq*[*of 1/2 x 1 1 0*]
show *?thesis*
by(*auto intro*!: *image-eqI*[**where** $x = \sum n.\ (1\ /\ 2)\ \hat{}\ Suc\ n * x\ n$])
qed
qed
qed
have $(\Pi_E\ i \in UNIV.\ \{0,1\}) - ?A = \{x.\ (\forall n.\ x\ n \in \{0,1\}) \wedge (\exists n.\ \forall k \geq n.\ x\ k = 1)\} - \{\lambda n.\ 1\}$
proof
show $(\Pi_E\ i \in UNIV.\ \{0,1\}) - ?A \subseteq \{x.\ (\forall n.\ x\ n \in \{0,1\}) \wedge (\exists n.\ \forall k \geq n.\ x\ k = 1)\} - \{\lambda n.\ 1\}$
proof
fix $x :: nat \Rightarrow real$
assume $x \in (\Pi_E\ i \in UNIV.\ \{0,1\}) - ?A$
then have $\forall n.\ x\ n \in \{0,1\}$ $\neg (\forall n.\ \exists k \geq n.\ x\ k = 0)$ $x \neq (\lambda n.\ 1)$
using *eq1* **by** *blast+*
thus $x \in \{x.\ (\forall n.\ x\ n \in \{0,1\}) \wedge (\exists n.\ \forall k \geq n.\ x\ k = 1)\} - \{\lambda n.\ 1\}$
by *blast*
qed
next
show $(\Pi_E\ i \in UNIV.\ \{0,1\}) - ?A \supseteq \{x.\ (\forall n.\ x\ n \in \{0,1\}) \wedge (\exists n.\ \forall k \geq n.\ x\ k = 1)\} - \{\lambda n.\ 1\}$
proof
fix $x :: nat \Rightarrow real$
assume $h{:}x \in \{x.\ (\forall n.\ x\ n \in \{0,1\}) \wedge (\exists n.\ \forall k \geq n.\ x\ k = 1)\} - \{\lambda n.\ 1\}$
then have $\forall n.\ x\ n \in \{0,1\}$ $\exists n.\ \forall k \geq n.\ x\ k = 1$ $x \neq (\lambda n.\ 1)$
by *blast+*
hence $\neg (\forall n.\ \exists k \geq n.\ x\ k = 0)$
by *fastforce*
with ‹$\forall n.\ x\ n \in \{0,1\}$› ‹$x \neq (\lambda n.\ 1)$›
show $x \in (\Pi_E\ i \in UNIV.\ \{0,1\}) - ?A$

```
            using eq1 by blast
          qed
        qed
        also have ... = (⋃ ((λn. {x. (∀ n. x n ∈ {0,1}) ∧ (∀ k≥n. x k = 1)}) ‘
UNIV)) − {λn. 1}
          by blast
        also have ... ∈ sets Cantor-space (is ?B ∈ -)
        proof −
          have countable ?B
          proof −
            have countable {x :: nat ⇒ real. (∀ n. x n = 0 ∨ x n = 1) ∧ (∀ k≥m.
x k = 1)} for m :: nat
            proof −
              let ?C = {x::nat ⇒ real. (∀ n. x n = 0 ∨ x n = 1) ∧ (∀ k≥m. x k =
1)}
                define g where g = (λ(x::nat ⇒ real) n. if n < m then x n else
undefined)
              have 1:g ‘ ?C = (Π_E i ∈{..<m}. {0,1})
              proof(standard; standard)
                fix x
                assume x ∈ g ‘ ?C
                then show x ∈ (Π_E i ∈{..<m}. {0,1})
                  by(auto simp: g-def PiE-def extensional-def)
              next
                fix x
                assume h:x ∈ (Π_E i ∈{..<m}. {0,1::real})
                then have x = g (λn. if n < m then x n else 1)
                  by(auto simp add: g-def PiE-def extensional-def)
                moreover have (λn. if n < m then x n else 1) ∈ ?C
                  using h by auto
                ultimately show x ∈ g ‘ ?C
                  by auto
              qed
              have 2:inj-on g ?C
              proof
                fix x y
                assume hxyg:x ∈ ?C y :  ?C g x = g y
                show x = y
                proof
                  fix n
                  consider n < m | m ≤ n by fastforce
                  thus x n = y n
                  proof cases
                    case 1
                    then show ?thesis
                      using fun-cong[OF hxyg(3),of n] by(simp add: g-def)
                  next
                    case 2
                    then show ?thesis
```

```
                using hxyg(1,2) by auto
              qed
            qed
          qed
          show countable {x::nat ⇒ real. (∀ n. x n = 0 ∨ x n = 1) ∧ (∀ k≥m.
x k = 1)}
            by(rule countable-image-inj-on[OF - 2]) (auto intro!: countable-PiE
simp: 1)
        qed
        thus ?thesis
          by auto
      qed
      moreover have ?B ⊆ space Cantor-space
        by(auto simp: space-Cantor-space)
      ultimately show ?thesis
    using Cantor-space-standard-ne by(simp add: standard-borel.countable-sets
standard-borel-ne-def)
    qed
    finally show ?thesis .
  qed
  moreover have space Cantor-space − ((Π_E i∈ UNIV. {0,1}) − ?A) = ?A
    using ‹?A ⊆ space Cantor-space› space-Cantor-space by blast
  ultimately show ?thesis
    using sets.compl-sets[OF comple-sets] by auto
qed
have to-Cantor-from-01 ‘ A = f −‘ A ∩ to-Cantor-from-01 ‘ (space (restrict-space
borel {0..1}))
proof
   show to-Cantor-from-01 ‘ A ⊆ f −‘ A ∩ to-Cantor-from-01 ‘ space
(restrict-space borel {0..1})
  proof
    fix x
    assume x ∈ to-Cantor-from-01 ‘ A
    then obtain a where ha:a ∈ A x = to-Cantor-from-01 a by auto
    hence a ∈ {0..1}
      using sets.sets-into-space[OF assms] by auto
    have f x = a
      using to-Cantor-from-sum[OF ‹a ∈ {0..1}›] by(simp add: f-def ha(2))
    thus x ∈ f −‘ A ∩ to-Cantor-from-01 ‘ space (restrict-space borel {0..1})
      using sets.sets-into-space[OF assms] ha by auto
  qed
next
   show to-Cantor-from-01 ‘ A ⊇ f −‘ A ∩ to-Cantor-from-01 ‘ space
(restrict-space borel {0..1})
  proof
    fix x
     assume h:x ∈ f −‘ A ∩ to-Cantor-from-01 ‘ space (restrict-space borel
{0..1})
    then obtain r where r ∈ {0..1} x = to-Cantor-from-01 r
```

by *auto*
from *h* **have** *f x* ∈ *A*
by *simp*
hence *to-Cantor-from-01* (*f x*) = *x*
using *to-Cantor-from-01-sum-of-to-Cantor-from-01*[*OF* ‹*r* ∈ {*0..1*}›]
by(*simp add*: *f-def* ‹*x* = *to-Cantor-from-01 r*›)
with ‹*f x* ∈ *A*›
show *x* ∈ *to-Cantor-from-01* ‘ *A*
by (*simp add*: *rev-image-eqI*)
qed
qed
also have ... ∈ *sets Cantor-space*
proof −
have *f* −‘ *A* ∩ *space Cantor-space* ∩ *to-Cantor-from-01* ‘ *space* (*restrict-space borel* {*0..1*}) = *f* −‘ *A* ∩ *to-Cantor-from-01* ‘ (*space* (*restrict-space borel* {*0..1*}))
using *to-Cantor-from-01-image sets.sets-into-space*[*OF assms*,*simplified*]
by *auto*
thus *?thesis*
using *sets.Int*[*OF measurable-sets*[*OF f-meas assms*] *image-sets*]
by *fastforce*
qed
finally show *?thesis* **.**
qed
show *?thesis*
using *Schroeder-Bernstein-measurable*[*OF Cantor-to-01-measurable Cantor-to-01-preserves-sets Cantor-to-01-inj to-Cantor-from-01-measurable to-Cantor-from-01-preserves-sets to-Cantor-from-01-inj*]
by(*simp add*: *measurable-isomorphic-def*)
qed
have *1*:*Cantor-space measurable-isomorphic* (Π_M (*i*::*nat*,*j*::*nat*)∈ *UNIV* × *UNIV*. *restrict-space borel* {*0*,*1*::*real*})
unfolding *Cantor-space-def*
by(*auto intro*!: *measurable-isomorphic-sym*[*OF countable-infinite-isomorphisc-to-nat-index*] *simp*: *split-beta′ finite-prod*)
have *2*:(Π_M (*i*::*nat*,*j*::*nat*)∈ *UNIV* × *UNIV*. *restrict-space borel* {*0*,*1*::*real*}) *measurable-isomorphic* (Π_M (*i*::*nat*)∈ *UNIV*. *Cantor-space*)
unfolding *Cantor-space-def* **by**(*rule measurable-isomorphic-sym*[*OF PiM-PiM-isomorphic-to-PiM*])
have *3*:(Π_M (*i*::*nat*)∈ *UNIV*. *Cantor-space*) *measurable-isomorphic Hilbert-cube*
unfolding *Hilbert-cube-def* **by**(*rule measurable-isomorphic-lift-product*[*OF Cantor-space-isomorphic-to-01closed*])
show *?thesis*
by(*rule measurable-isomorphic-trans*[*OF measurable-isomorphic-trans*[*OF 1 2*] *3*])
qed

3.3 Final Results

lemma(**in** *standard-borel*) *embedding-into-Hilbert-cube*:
∃ *A* ∈ *sets Hilbert-cube*. *M measurable-isomorphic* (*restrict-space Hilbert-cube A*)

proof −
obtain *S* **where** *S*:*Polish-space S sets* (*borel-of S*) = *sets M*
using *Polish-space* **by** *blast*
obtain *A* **where** *A*:*gdelta-in Hilbert-cube-topology A S homeomorphic-space subtopology Hilbert-cube-topology A*
using *embedding-into-Hilbert-cube-gdelta-in*[*OF S*(*1*)] **by** *blast*
show *?thesis*
using *borel-of-gdelta-in*[*OF A*(*1*)] *homeomorphic-space-measurable-isomorphic*[*OF A*(*2*)] *measurable-isomorphic-sets-cong*[*OF S*(*2*),*of borel-of* (*subtopology Hilbert-cube-topology A*) *restrict-space Hilbert-cube A*] *Hilbert-cube-borel sets-restrict-space-cong*[*OF Hilbert-cube-borel*]
by(*auto intro*!: *bexI*[**where** *x*=*A*] *simp*: *borel-of-subtopology*)
qed

lemma(**in** *standard-borel*) *embedding-from-Cantor-space*:
assumes *uncountable* (*space M*)
shows ∃ *A* ∈ *sets M*. *Cantor-space measurable-isomorphic* (*restrict-space M A*)
proof −
obtain *S* **where** *S*:*Polish-space S sets* (*borel-of S*) = *sets M*
using *Polish-space* **by** *blast*
then obtain *A* **where** *A*:*gdelta-in S A Cantor-space-topology homeomorphic-space subtopology S A*
using *embedding-from-Cantor-space*[*of S*] *assms sets-eq-imp-space-eq*[*OF S*(*2*)]
by(*auto simp*: *space-borel-of*)
show *?thesis*
using *borel-of-gdelta-in*[*OF A*(*1*)] *S*(*2*) *homeomorphic-space-measurable-isomorphic*[*OF A*(*2*)] *measurable-isomorphic-sets-cong*[*OF Cantor-space-borel restrict-space-sets-cong*[*OF refl S*(*2*)],*of A*]
by(*auto intro*!: *bexI*[**where** *x*=*A*] *simp*: *borel-of-subtopology*)
qed

corollary(**in** *standard-borel*) *uncountable-isomorphic-to-Hilbert-cube*:
assumes *uncountable* (*space M*)
shows *Hilbert-cube measurable-isomorphic M*
proof −
obtain *A B* **where** *AB*:
M measurable-isomorphic (*restrict-space Hilbert-cube A*) *Cantor-space measurable-isomorphic* (*restrict-space M B*)
A ∈ *sets Hilbert-cubeB* ∈ *sets M*
using *embedding-into-Hilbert-cube embedding-from-Cantor-space*[*OF assms*] **by** *auto*
show *?thesis*
by(*rule measurable-isomorphic-antisym*[*OF AB measurable-isomorphic-sym*[*OF Cantor-space-isomorphic-to-Hilbert-cube*]])
qed

corollary(**in** *standard-borel*) *uncountable-isomorphic-to-real*:
assumes *uncountable* (*space M*)
shows *M measurable-isomorphic* (*borel* :: *real measure*)
proof −

interpret *r*: *standard-borel-ne borel* :: *real measure*
by *simp*
show *?thesis*
by(*auto intro*!: *measurable-isomorphic-trans*[*OF measurable-isomorphic-sym*[*OF uncountable-isomorphic-to-Hilbert-cube*[*OF assms*]] *r.uncountable-isomorphic-to-Hilbert-cube*] *simp*: *uncountable-UNIV-real*)
qed

lemma(**in** *standard-borel*) *isomorphic-subset-real*:
assumes $A \in$ *sets* (*borel* :: *real measure*) *uncountable A*
obtains *B* **where** $B \in$ *sets borel* $B \subseteq A$ *M measurable-isomorphic restrict-space borel B*
proof(*cases countable* (*space M*))
assume *count*:*countable* (*space M*)
have $\exists B \subseteq A$. *space M* $\approx$ *B*
proof(*cases finite* (*space M*))
assume *fin*:*finite* (*space M*)
then obtain *B* **where** *B*:*card B* = *card* (*space M*) *finite B* $B \subseteq A$
by (*meson assms*(*2*) *countable-finite infinite-arbitrarily-large*)
thus *?thesis*
using *fin* **by**(*auto intro*!: *exI*[**where** *x*=*B*] *simp*:*eqpoll-iff-card*)
next
assume *inf*:*infinite* (*space M*)
obtain *B* **where** *B*: $B \subseteq A$ *countable B infinite B*
using *assms*(*2*) *countable-finite infinite-countable-subset′ that* **by** *auto*
thus *?thesis*
using *bij-betw-from-nat-into*[*OF count inf*] *bij-betw-from-nat-into*[*OF B*(*2*,*3*)]
by (*meson eqpoll-def eqpoll-sym eqpoll-trans*)
qed
then obtain *B* **where** *B*:$B \subseteq A$ *space M* $\approx$ *B countable B*
by (*metis countable-eqpoll eqpoll-sym count*)
then obtain *f* **where** *f*:*bij-betw f* (*space M*) *B*
using *eqpoll-def* **by** *blast*
have *1*:$C \in$ *sets borel* **if** *C*:$C \subseteq B$ **for** *C*
proof −
have $C = (\bigcup c \in C.\ \{c\})$
by *auto*
also have ... $\in$ *sets borel*
using *B C* **by**(*intro sets.countable-UN′*) (*auto simp*: *countable-subset*)
finally show *?thesis* **.**
qed
have *2*:*sets M* = *sets* (*count-space* (*space M*))
by (*simp add*: *countable-discrete-space count*)
have *3*:*sets* (*restrict-space borel B*) = *sets* (*count-space B*)
using *1* **by**(*auto simp*: *sets-restrict-space*)
have [*simp*]:*measurable M* (*restrict-space borel B*) = *measurable* (*count-space* (*space M*)) (*count-space B*)
measurable (*restrict-space borel B*) *M* = *measurable* (*count-space B*) (*count-space* (*space M*))

using *2 3* **by**(*auto intro*!: *measurable-cong-sets*)
have *M measurable-isomorphic restrict-space borel B*
using *bij-betw-the-inv-into*[*OF f*] *f* **by**(*auto simp*: *measurable-isomorphic-def measurable-isomorphic-map-def space-restrict-space intro*!: *exI*[**where** *x*=*f*] *dest*: *bij-betwE*)
with *1 B that* **show** *?thesis*
by *blast*
next
assume *uncountable* (*space M*)
then have *M measurable-isomorphic* (*borel* :: *real measure*)
using *uncountable-isomorphic-to-real* **by** *blast*
moreover have *restrict-space borel A measurable-isomorphic* (*borel* :: *real measure*)
by(*auto intro*!: *standard-borel.uncountable-isomorphic-to-real standard-borel.standard-borel-restrict-space*[*OF standard-borel-ne.standard-borel*] *simp*: *assms space-restrict-space*)
ultimately have *M measurable-isomorphic restrict-space borel A*
using *measurable-isomorphic-sym measurable-isomorphic-trans* **by** *blast*
with *assms*(*1*) *that* **show** *?thesis*
by *blast*
qed

lemma(**in** *standard-borel*) *countable-isomorphic-to-subset-real*:
assumes *countable* (*space M*)
obtains *A* :: *real set*
where *countable A A* ∈ *sets borel M measurable-isomorphic restrict-space borel A*
proof −
obtain *A* :: *real set* **where** *A*:*A* ∈ *sets borel M measurable-isomorphic restrict-space borel A*
using *isomorphic-subset-real*[*of UNIV*] *uncountable-UNIV-real* **by** *auto*
moreover have *countable A*
using *measurable-isomorphic-cardinality-eq*[*OF A*(*2*)] *assms*(*1*)
by(*simp add*: *space-restrict-space countable-eqpoll*[*OF - eqpoll-sym*])
ultimately show *?thesis*
using *that* **by** *blast*
qed

theorem *Borel-isomorphism-theorem*:
assumes *standard-borel M standard-borel N*
shows *space M* ≈ *space N* ⟷ *M measurable-isomorphic N*
proof
assume *h*:*space M* ≈ *space N*
interpret *M*: *standard-borel M* **by** *fact*
interpret *N*: *standard-borel N* **by** *fact*
consider *countable* (*space M*) *countable* (*space N*) | *uncountable* (*space M*) *uncountable* (*space N*)
by (*meson countable-eqpoll eqpoll-sym h*)
thus *M measurable-isomorphic N*
proof *cases*

case *1*
then have *2*:*sets M* = *sets* (*count-space* (*space M*)) *sets N* = *sets* (*count-space* (*space N*))
by (*simp-all add*: *M*.*countable-discrete-space N*.*countable-discrete-space*)
show *?thesis*
by(*simp add*: *measurable-isomorphic-sets-cong*[*OF 2*] *measurable-isomorphic-count-spaces h*)
next
case *2*
then have *M measurable-isomorphic* (*borel* :: *real measure*) *N measurable-isomorphic* (*borel* :: *real measure*)
by(*simp-all add*: *M*.*uncountable-isomorphic-to-real N*.*uncountable-isomorphic-to-real*)
thus *?thesis*
using *measurable-isomorphic-sym measurable-isomorphic-trans* **by** *blast*
qed
qed(*rule measurable-isomorphic-cardinality-eq*)

definition *to-real-on* :: *'a measure* ⇒ *'a* ⇒ *real* **where**
to-real-on M ≡ (*if uncountable* (*space M*) *then* (*SOME f*. *measurable-isomorphic-map M* (*borel* :: *real measure*) *f*) *else* (*real* ∘ *to-nat-on* (*space M*)))

definition *from-real-into* :: *'a measure* ⇒ *real* ⇒ *'a* **where**
from-real-into M ≡ (*if uncountable* (*space M*) *then the-inv-into* (*space M*) (*to-real-on M*) *else* (λ*r*. *from-nat-into* (*space M*) (*nat* ⌊*r*⌋)))

context *standard-borel*
begin

abbreviation *to-real* ≡ *to-real-on M*
abbreviation *from-real* ≡ *from-real-into M*

lemma *to-real-def-countable*:
assumes *countable* (*space M*)
shows *to-real* = (λ*r*. *real* (*to-nat-on* (*space M*) *r*))
using *assms* **by**(*auto simp*: *to-real-on-def*)

lemma *from-real-def-countable*:
assumes *countable* (*space M*)
shows *from-real* = (λ*r*. *from-nat-into* (*space M*) (*nat* ⌊*r*⌋))
using *assms* **by**(*simp add*: *from-real-into-def*)

lemma *from-real-to-real*[*simp*]:
assumes *x* ∈ *space M*
shows *from-real* (*to-real x*) = *x*
proof −
have [*simp*]: *space M* ≠ {}
using *assms* **by** *auto*
consider *countable* (*space M*) | *uncountable* (*space M*) **by** *auto*
then show *?thesis*

proof *cases*
case *1*
then show *?thesis*
by(*simp add*: *to-real-def-countable from-real-def-countable assms*)
next
case *2*
then obtain *f* **where** *f*: *measurable-isomorphic-map M* (*borel* :: *real measure*) *f*
using *uncountable-isomorphic-to-real* **by**(*auto simp*: *measurable-isomorphic-def*)
have *1*:*to-real* = *Eps* (*measurable-isomorphic-map M borel*) *from-real* = *the-inv-into* (*space M*) (*Eps* (*measurable-isomorphic-map M borel*))
by(*simp-all add*: *to-real-on-def 2 from-real-into-def*)
show *?thesis*
unfolding *1*
by(*rule someI2*[*of measurable-isomorphic-map M* (*borel* :: *real measure*) *f*,*OF f*])
(*meson assms bij-betw-imp-inj-on measurable-isomorphic-map-def the-inv-into-f-f*)
qed
qed

lemma *to-real-measurable*[*measurable*]:
to-real ∈ $M \to_M$ *borel*
proof(*cases countable* (*space M*))
case *1*:*True*
then have *sets M* = *Pow* (*space M*)
by(*rule countable-discrete-space*)
then show *?thesis*
by(*simp add*: *to-real-def-countable 1 borel-measurableI-le*)
next
case *1*:*False*
then obtain *f* **where** *f*: *measurable-isomorphic-map M* (*borel* :: *real measure*) *f*
using *uncountable-isomorphic-to-real* **by**(*auto simp*: *measurable-isomorphic-def*)
have *2*:*to-real* = *Eps* (*measurable-isomorphic-map M borel*)
by(*simp add*: *to-real-on-def 1 from-real-into-def*)
show *?thesis*
unfolding *2*
by(*rule someI2*[*of measurable-isomorphic-map M* (*borel* :: *real measure*) *f*,*OF f*],*simp add*: *measurable-isomorphic-map-def*)
qed

lemma *from-real-measurable'*:
assumes *space M* ≠ {}
shows *from-real* ∈ *borel* $\to_M$ *M*
proof(*cases countable* (*space M*))
case *1*:*True*
then have *2*:*sets M* = *Pow* (*space M*)
by(*rule countable-discrete-space*)
have [*measurable*]:*from-nat-into* (*space M*) ∈ *count-space UNIV* $\to_M$ *M*
using *from-nat-into*[*OF assms*] **by** *auto*

show *?thesis*
by(*simp add*: *from-real-def-countable 1 borel-measurableI-le*)
next
case *2*:*False*
then obtain *f* **where** *f*: *measurable-isomorphic-map M* (*borel* :: *real measure*) *f*
using *uncountable-isomorphic-to-real* **by**(*auto simp*: *measurable-isomorphic-def*)
have *1*: *from-real* = *the-inv-into* (*space M*) (*Eps* (*measurable-isomorphic-map M borel*))
by(*simp add*: *to-real-on-def 2 from-real-into-def*)
show *?thesis*
unfolding *1*
by(*rule someI2*[*of measurable-isomorphic-map M* (*borel* :: *real measure*) *f*,*OF f*],*simp add*: *measurable-isomorphic-map-def*)
qed

lemma *to-real-from-real*:
assumes *uncountable* (*space M*)
shows *to-real* (*from-real r*) = *r*
proof −
obtain *f* **where** *f*: *measurable-isomorphic-map M* (*borel* :: *real measure*) *f*
using *assms uncountable-isomorphic-to-real* **by**(*auto simp*: *measurable-isomorphic-def*)
have *1*:*to-real* = *Eps* (*measurable-isomorphic-map M borel*) *from-real* = *the-inv-into* (*space M*) (*Eps* (*measurable-isomorphic-map M borel*))
by(*simp-all add*: *to-real-on-def assms from-real-into-def*)
show *?thesis*
unfolding *1*
by(*rule someI2*[*of measurable-isomorphic-map M* (*borel* :: *real measure*) *f*,*OF f*])
(*metis UNIV-I f-the-inv-into-f-bij-betw measurable-isomorphic-map-def space-borel*)
qed

end

lemma(**in** *standard-borel-ne*) *from-real-measurable*[*measurable*]: *from-real* ∈ *borel* $\rightarrow_M$ *M*
by(*simp add*: *from-real-measurable′ space-ne*)

end

References

[1] Lecture note of math245b in UCLA. https://web.archive.org/web/20210506130459/https://www.math.ucla.edu/~biskup/245b.1.20w/, 2020. Accessed: June 27. 2023.

[2] K. Matsuzaka. 集合・位相入門. Iwanami Shoten, 1968. written in Japanese.

[3] S. M. Srivastava. *A Course on Borel Sets.* Springer, 1998.