Standard Borel Spaces

Michikazu Hirata

October 12, 2023

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

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-generated-by* $\{U \mid U\ .\ openin\ S\ U\}$
unfolding *topology-eq*
proof *standard+*
fix U
assume *openin* (*topology-generated-by* $\{U \mid U.\ openin\ S\ U\}$) 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 $\bigwedge U.\ U \in \mathcal{U} \Longrightarrow$ (*openin* (*topology-generated-by* $\mathcal{O}$) *U*)
and $\bigwedge U.\ U \in \mathcal{O} \Longrightarrow$ (*openin* (*topology-generated-by* $\mathcal{U}$) *U*)
shows *topology-generated-by* $\mathcal{O}$ = *topology-generated-by* $\mathcal{U}$
using *topology-generated-by-sub*[*of* $\mathcal{U}$, *OF assms*(*1*)] *topology-generated-by-sub*[*of* $\mathcal{O}$,*OF assms*(*2*)]
by(*auto simp*: *topology-eq*)

lemma *topology-generated-by-homeomorphic-spaces*:
assumes *homeomorphic-map X Y f X* = *topology-generated-by* $\mathcal{O}$
shows *Y* = *topology-generated-by* ((‘) *f* ‘ $\mathcal{O}$)
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*: $\bigwedge x.\ x \in$ *topspace X* $\Longrightarrow$ *g* (*f x*) = *x* $\bigwedge y.\ y \in$ *topspace Y* $\Longrightarrow$ *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 $\bigwedge S.$ *openin Y S* = *openin* (*topology-generated-by* ((‘) *f* ‘ $\mathcal{O}$)) *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* $\mathcal{O}$ (*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* ‘ $\mathcal{O}$)) ...
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 $a \cap b \cap T = (a \cap T) \cap (b \cap T)$ **by** *auto*
ultimately show *?case*
by(*auto intro*!: *generate-topology-on.Int*)
next
case (*UN K*)
moreover have $(\bigcup K \cap T) = (\bigcup \{ k \cap T \mid k.\ k \in K\})$ **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* $\{T \cap U \mid U.\ U \in \mathcal{O}\}$ *A*
then show $\exists L.$ *generate-topology-on* $\mathcal{O}$ *L* $\wedge$ $A = L \cap T$
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 $\mathcal{O}$ *La a* = *La* $\cap$ *T generate-topology-on* $\mathcal{O}$ *Lb b* = *Lb* $\cap$ *T*
by *auto*
thus *?case*
using *ih* **by**(*auto intro*!: *exI*[**where** *x*=*La* $\cap$ *Lb*] *generate-topology-on.Int*)
next
case *ih*:(*UN K*)
then obtain *L* **where**
$\bigwedge k.\ k \in K \Longrightarrow$ *generate-topology-on* $\mathcal{O}$ (*L k*) $\bigwedge k.\ k \in K \Longrightarrow k = (L\ k) \cap T$
by *metis*
thus *?case*
using *ih* **by**(*auto intro*!: *exI*[**where** *x*=$\bigcup k{\in}K.$ *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 $\{ U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$ = *prod-topology*

(*topology-generated-by* $\mathcal{O}$) (*topology-generated-by* $\mathcal{U}$)
unfolding *topology-eq*
proof *safe*
fix U
assume *h*:*openin* (*topology-generated-by* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$) 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 $\forall z{\in}U.\ \exists\ V1\ V2.$ *openin* (*topology-generated-by* $\mathcal{O}$) $V1 \wedge$ *openin* (*topology-generated-by* $\mathcal{U}$) $V2 \wedge$ *fst* $z \in V1 \wedge$ *snd* $z \in V2 \wedge V1 \times V2 \subseteq U$
by(*auto simp*: *openin-prod-topology-alt*)
hence $\exists\ V1.\ \forall z{\in}U.\ \exists\ V2.$ *openin* (*topology-generated-by* $\mathcal{O}$) $(V1\ z) \wedge$ *openin* (*topology-generated-by* $\mathcal{U}$) $V2 \wedge$ *fst* $z \in (V1\ z) \wedge$ *snd* $z \in V2 \wedge (V1\ z) \times V2 \subseteq U$
by(*rule bchoice*)
then obtain $V1$ **where** $\forall z{\in}U.\ \exists\ V2.$ *openin* (*topology-generated-by* $\mathcal{O}$) $(V1\ z) \wedge$ *openin* (*topology-generated-by* $\mathcal{U}$) $V2 \wedge$ *fst* $z \in (V1\ z) \wedge$ *snd* $z \in V2 \wedge (V1\ z) \times V2 \subseteq U$
by *auto*
hence $\exists\ V2.\ \forall z{\in}U.$ *openin* (*topology-generated-by* $\mathcal{O}$) $(V1\ z) \wedge$ *openin* (*topology-generated-by* $\mathcal{U}$) $(V2\ z) \wedge$ *fst* $z \in (V1\ z) \wedge$ *snd* $z \in (V2\ z) \wedge (V1\ z) \times (V2\ z) \subseteq U$
by(*rule bchoice*)
then obtain $V2$ **where** *hv12*:$\bigwedge z.\ z{\in}U \Longrightarrow$ *openin* (*topology-generated-by* $\mathcal{O}$) $(V1\ z) \wedge$ *openin* (*topology-generated-by* $\mathcal{U}$) $(V2\ z) \wedge$ *fst* $z \in (V1\ z) \wedge$ *snd* $z \in (V2\ z) \wedge (V1\ z) \times (V2\ z) \subseteq U$
by *auto*
hence *1*:$U = (\bigcup z{\in}U.\ (V1\ z) \times (V2\ z))$
by *auto*
have *openin* (*topology-generated-by* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$) $(\bigcup z{\in}U.\ (V1\ z) \times (V2\ z))$
proof(*rule openin-Union*)
show $\bigwedge S.\ S \in (\lambda z.\ V1\ z \times V2\ z)\ `\ U \Longrightarrow$ *openin* (*topology-generated-by* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$) S
proof *safe*
fix $x\ y$
assume *h*:$(x,y) \in U$
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* $\{U \times V \mid U\ V.\ U \in \mathcal{O} \wedge V \in \mathcal{U}\}$) $(V1\ (x,\ y) \times V2\ (x,\ y))$
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-generated-by $\mathcal{O}$ = *topology-generated-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* ⟷ (*openin S* (*f* ‘ *u*)) ∧ *u* ⊆ *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* ⟷ *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* ⟷ *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* $\wedge$ *X isolated-points-of A* $= \{\}$

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

lemma *perfect-setI*:
assumes *closedin X A*
and $\bigwedge x\ T.\ [\![x \in A;\ x \in T;\ \textit{openin}\ X\ T]\!] \Longrightarrow \exists\, y{\neq}x.\ y \in T \wedge y \in A$
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 $\bigwedge x\ T.\ [\![x \in T;\ \textit{openin}\ X\ T]\!] \Longrightarrow \exists\, y{\neq}x.\ y \in T$
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* $\bigwedge x\ T.\ [\![x \in A;\ x \in T;\ \textit{openin}\ X\ T]\!] \Longrightarrow$
$\exists\, y{\neq}x.\ y \in T \wedge y \in A$
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-of* :: [*'a topology*, *'a set set*] $\Rightarrow$ *bool* **where**
subbase-of S $\mathcal{O}$ $\longleftrightarrow$ *S* $=$ *topology-generated-by* $\mathcal{O}$

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

definition *second-countable* :: *'a topology* $\Rightarrow$ *bool* **where**
second-countable S $\longleftrightarrow$ $(\exists\, \mathcal{O}.$ *countable* $\mathcal{O} \wedge$ *base-of S* $\mathcal{O})$

definition *zero-dimensional* :: *'a topology* $\Rightarrow$ *bool* **where**
zero-dimensional S $\longleftrightarrow$ $(\exists\, \mathcal{O}.$ *base-of S* $\mathcal{O} \wedge (\forall\, u{\in}\mathcal{O}.$ *openin S u* $\wedge$ *closedin S u*$))$

lemma *openin-base*:

assumes *base-of S* $\mathcal{O}$ $U = \bigcup \mathcal{U}$ **and** $\mathcal{U} \subseteq \mathcal{O}$
shows *openin S U*
using *assms* **by**(*auto simp*: *base-of-def*)

lemma *base-is-subbase*:
assumes *base-of S* $\mathcal{O}$
shows *subbase-of S* $\mathcal{O}$
unfolding *subbase-of-def topology-eq openin-topology-generated-by-iff*
proof *safe*
fix *U*
assume *openin S U*
then obtain $\mathcal{U}$ **where** *hu*:$U = \bigcup \mathcal{U}$ $\mathcal{U} \subseteq \mathcal{O}$
using *assms* **by**(*auto simp*: *base-of-def*)
thus *generate-topology-on* $\mathcal{O}$ *U*
by(*auto intro*!: *generate-topology-on.UN*) (*auto intro*!: *generate-topology-on.Basis*)
next
fix *U*
assume *generate-topology-on* $\mathcal{O}$ *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-of-subset*:
assumes *subbase-of S* $\mathcal{O}$ **and** $U \in \mathcal{O}$
shows $U \subseteq$ *topspace S*
using *assms*(*1*)[*simplified subbase-of-def*] *topology-generated-by-topspace assms*
by *auto*

lemma *subbase-of-openin*:
assumes *subbase-of S* $\mathcal{O}$ **and** $U \in \mathcal{O}$
shows *openin S U*
using *assms* **by**(*simp add*: *subbase-of-def openin-topology-generated-by-iff generate-topology-on.Basis*)

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

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

lemma *base-of-def2*:
assumes $\bigwedge U.\ U \in \mathcal{O} \Longrightarrow$ *openin S U*
shows *base-of 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-of 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-of-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-of S* $\mathcal{O}$
unfolding *base-of-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-of-def2'*:
base-of S $\mathcal{O} \longleftrightarrow (\forall\, b{\in}\mathcal{O}.$ *openin S b*$) \wedge (\forall\, x.$ *openin S x* $\longrightarrow (\exists\, B'{\subseteq}\mathcal{O}.\ \bigcup\ B' = x))$
proof
assume *h*:*base-of S* $\mathcal{O}$
show $(\forall\, b{\in}\mathcal{O}.$ *openin S b*$) \wedge (\forall\, x.$ *openin S x* $\longrightarrow (\exists\, B'{\subseteq}\mathcal{O}.\ \bigcup\ B' = x))$
proof(*rule conjI*)
show $\forall\, b{\in}\mathcal{O}.$ *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-of-def)
  qed
next
  assume h:(∀b∈𝒪. openin S b) ∧ (∀x. openin S x ⟶ (∃B'⊆𝒪. ⋃ B' = x))
  show base-of S 𝒪
    unfolding base-of-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-of-in-subset:
  assumes base-of S 𝒪 openin S u x ∈ u
  shows ∃v∈𝒪. x ∈ v ∧ v ⊆ u
  using assms base-of-def2 base-of-def2' by fastforce

lemma base-of-without-empty:
  assumes base-of S 𝒪
  shows base-of S {U ∈ 𝒪. U ≠ {}}
  unfolding base-of-def2'
proof safe
  fix x
  assume x ∈ 𝒪  ¬ openin S x
  thus ⋀y. y ∈ {}
    using base-of-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-of-def2') metis
  thus ∃B'⊆{U ∈ 𝒪. U ≠ {}}. ⋃ B' = x
    by(auto intro!: exI[where x={y ∈ B'. y ≠ {}}])
qed

lemma second-countable-ex-without-empty:
  assumes second-countable S
  shows ∃𝒪. countable 𝒪 ∧ base-of S 𝒪 ∧ (∀U∈𝒪. U ≠ {})
proof −
  obtain 𝒪 where countable 𝒪 base-of S 𝒪
```

using *assms second-countable-def* **by** *blast*
thus *?thesis*
by(*auto intro*!: *exI*[**where** x=$\{U \in \mathcal{O}.\ U \neq \{\}\}$] *base-of-without-empty*)
qed

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

lemma *subtopology-base-of*:
assumes *base-of* S $\mathcal{O}$
shows *base-of* (*subtopology* S T) $\{T \cap U \mid U.\ U \in \mathcal{O}\}$
unfolding *base-of-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-of-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*:$L = \bigcup \{k\ U \mid U.\ U \in \mathcal{U}\} \cap T$ **.**
moreover have *openin* S $(\bigcup \{k\ U \mid U.\ U \in \mathcal{U}\})$
using *hu hk assms* **by**(*auto simp*: *base-of-def*)
ultimately show *openin* (*subtopology* S T) L
by(*auto intro*!: *exI*[**where** x=$\bigcup \{k\ U \mid U.\ U \in \mathcal{U}\}$] *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-of S* $\mathcal{O}$
using *assms second-countable-def* **by** *blast*
thus *?thesis*
by(*auto intro*!: *exI*[**where** *x*={$T \cap U \mid U.\ U \in \mathcal{O}$}] *simp*: *second-countable-def Setcompr-eq-image dest*: *subtopology-base-of*)
qed

lemma *Lindelof-of*:
assumes *second-countable S* $\bigwedge u.\ u \in U \Longrightarrow$ *openin S u* $\bigcup U$ = *topspace S*
shows $\exists U'.$ *countable* $U' \wedge U' \subseteq U \wedge \bigcup U'$ = *topspace S*
proof −
from *assms*(*1*) **obtain** $\mathcal{O}$ **where** *h*: *countable* $\mathcal{O}$ *base-of S* $\mathcal{O}$
by(*auto simp*: *second-countable-def*)
define B' **where** $B' \equiv \{v \in \mathcal{O}.\ \exists u \in U.\ v \subseteq u\}$
have B': *countable* B'
using *h*(*1*) **by**(*auto simp*: B'*-def*)
have $\forall v.\ v \in B' \longrightarrow (\exists u \in U.\ v \subseteq u)$ **by**(*auto simp*: B'*-def*)
then obtain U' **where** U':$\bigwedge v.\ v \in B' \Longrightarrow U'\ v \in U\ \bigwedge v.\ v \in B' \Longrightarrow v \subseteq U'\ v$
by *metis*
show *?thesis*
proof(*rule exI*[**where** *x*=U' ‘ B'])
show *countable* (U' ‘ B') $\wedge$ U' ‘ $B' \subseteq U \wedge \bigcup$ (U' ‘ B') = *topspace S*
proof *safe*
fix *x*
assume $x \in$ *topspace S*
then obtain *u* **where** *u*:$x \in u\ u \in U$
using *assms*(*3*) **by** *auto*
obtain *v* **where** *v*:$x \in v\ v \in \mathcal{O}\ v \subseteq u$
using *base-of-in-subset*[*OF h*(*2*) *assms*(*2*)[*OF u*(*2*)] *u*(*1*)] **by** *auto*
show $x \in \bigcup$ (U' ‘ B')
using *u v* U' **by**(*auto intro*!: *bexI*[**where** *x*=*v*]) (*auto simp*: B'*-def intro*!: *exI*[**where** *x*=*u*])
qed(*use* B' U' *assms*(*2*) *openin-subset* **in** *blast*)+
qed
qed

lemma *open-map-with-base*:
assumes *base-of 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-of-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.

definition *finite-intersections* :: *'a set set* ⇒ *'a set set* **where**
finite-intersections $\mathcal{O} \equiv \{ \bigcap \mathcal{O}' \mid \mathcal{O}'.\ \mathcal{O}' \neq \{\} \wedge$ *finite* $\mathcal{O}' \wedge \mathcal{O}' \subseteq \mathcal{O}\}$

lemma *finite-intersections-inI*:
assumes $U = \bigcap \mathcal{O}'$ $\mathcal{O}' \neq \{\}$ *finite* $\mathcal{O}'$ **and** $\mathcal{O}' \subseteq \mathcal{O}$
shows $U \in$ *finite-intersections* $\mathcal{O}$
using *assms* **by**(*auto simp*: *finite-intersections-def*)

lemma *finite-intersections-Uin*:
assumes $U \in \mathcal{O}$
shows $U \in$ *finite-intersections* $\mathcal{O}$
using *assms* **by**(*auto intro*!: *finite-intersections-inI*[*of* U $\{U\}$])

lemma *finite-intersections-int*:
assumes $U \in$ *finite-intersections* $\mathcal{O}$ **and** $V \in$ *finite-intersections* $\mathcal{O}$
shows $U \cap V \in$ *finite-intersections* $\mathcal{O}$
proof −
obtain $\mathcal{O}U$ $\mathcal{O}V$ **where**
$U = \bigcap \mathcal{O}U$ $\mathcal{O}U \neq \{\}$ *finite* $\mathcal{O}U$ $\mathcal{O}U \subseteq \mathcal{O}$ $V = \bigcap \mathcal{O}V$ *finite* $\mathcal{O}V$ $\mathcal{O}V \subseteq \mathcal{O}$
using *assms* **by**(*auto simp*: *finite-intersections-def*)
thus *?thesis*
by(*auto intro*!: *finite-intersections-inI*[*of* - $\mathcal{O}U \cup \mathcal{O}V$])
qed

lemma *finite-intersections-countable*:
assumes *countable* $\mathcal{O}$
shows *countable* (*finite-intersections* $\mathcal{O}$)
proof −
have *finite-intersections* $\mathcal{O} = (\bigcup i \in \{\mathcal{O}'.\ \mathcal{O}' \neq \{\} \wedge$ *finite* $\mathcal{O}' \wedge \mathcal{O}' \subseteq \mathcal{O}\}.\ \{\bigcap\ i\})$
by(*auto simp*: *finite-intersections-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-intersections-openin*:
assumes $U \in$ *finite-intersections* $\mathcal{O}$
shows *openin* (*topology-generated-by* $\mathcal{O}$) U
proof −

obtain $\mathcal{O}U$ **where** *hu*:
$U = \bigcap \mathcal{O}U$ $\mathcal{O}U \neq \{\}$ *finite* $\mathcal{O}U$ $\mathcal{O}U \subseteq \mathcal{O}$
using *assms* **by**(*auto simp*: *finite-intersections-def*)
show *?thesis*
using *hu* **by**(*auto intro*: *topology-generated-by-Basis*)
qed

lemma *topology-generated-by-finite-intersections*:
topology-generated-by $\mathcal{O}$ = *topology-generated-by* (*finite-intersections* $\mathcal{O}$)
proof(*rule topology-generated-by-eq*)
fix U
assume $U \in \mathcal{O}$
then show *openin* (*topology-generated-by* (*finite-intersections* $\mathcal{O}$)) U
by(*auto intro*!: *topology-generated-by-Basis simp*: *finite-intersections-Uin*)
qed (*rule finite-intersections-openin*)

lemma *topology-generated-by-is-union-of-finite-intersections*:
openin (*topology-generated-by* $\mathcal{O}$) $U \longleftrightarrow$ ($\exists \mathcal{U}.\ U = \bigcup \mathcal{U} \wedge \mathcal{U} \subseteq$ *finite-intersections* $\mathcal{O}$)
proof
assume *openin* (*topology-generated-by* $\mathcal{O}$) U
then have *generate-topology-on* $\mathcal{O}$ U
by (*simp add*: *openin-topology-generated-by-iff*)
thus $\exists \mathcal{U}.\ U = \bigcup\ \mathcal{U} \wedge \mathcal{U} \subseteq$ *finite-intersections* $\mathcal{O}$
proof *induction*
case *Empty*
then show *?case*
by *auto*
next
case (*Int a b*)
then obtain $\mathcal{U}a$ $\mathcal{U}b$ **where** *hab*:
$a = \bigcup\ \mathcal{U}a$ $\mathcal{U}a \subseteq$ *finite-intersections* $\mathcal{O}$ $b = \bigcup\ \mathcal{U}b$ $\mathcal{U}b \subseteq$ *finite-intersections* $\mathcal{O}$
by *auto*
then have $a \cap b = \bigcup \{\ Ua \cap Ub \mid Ua\ Ub.\ Ua \in \mathcal{U}a \wedge Ub \in \mathcal{U}b\}$
by *blast*
moreover have $\{\ Ua \cap Ub \mid Ua\ Ub.\ Ua \in \mathcal{U}a \wedge Ub \in \mathcal{U}b\} \subseteq$ *finite-intersections* $\mathcal{O}$
using *hab*(*2*,*4*) *finite-intersections-int* **by** *blast*
ultimately show *?case* **by** *auto*
next
case (*UN K*)
then have $\exists \mathcal{U}.\ \forall k{\in}K.\ k = \bigcup\ (\mathcal{U}\ k) \wedge \mathcal{U}\ k \subseteq$ *finite-intersections* $\mathcal{O}$
by(*auto intro*!: *bchoice*)
then obtain $\mathcal{U}$ **where**
$\forall k{\in}K.\ k = \bigcup\ (\mathcal{U}\ k) \wedge \mathcal{U}\ k \subseteq$ *finite-intersections* $\mathcal{O}$ **by** *auto*
thus *?case*
by(*auto intro*!: *exI*[**where** $x{=}\bigcup k{\in}K.\ (\mathcal{U}\ k)$]) (*metis UnionE*)
next
case (*Basis s*)

```
    then show ?case
      by(auto intro!: exI[where x={s}] finite-intersections-Uin)
  qed
next
  assume ∃𝒰. U = ⋃ 𝒰 ∧ 𝒰 ⊆ finite-intersections 𝒪
  then obtain 𝒰 where
   U = ⋃ 𝒰 𝒰 ⊆ finite-intersections 𝒪 by auto
  thus openin (topology-generated-by 𝒪) U
    using finite-intersections-openin
   by(auto simp: openin-topology-generated-by-iff intro!: generate-topology-on.UN)
qed

lemma base-from-subbase:
  assumes subbase-of S 𝒪
  shows base-of S (finite-intersections 𝒪)
 using topology-generated-by-is-union-of-finite-intersections[of 𝒪,simplified assms[simplified
subbase-of-def,symmetric]]
  by(simp add: base-of-def)

lemma countable-base-from-countable-subbase:
  assumes countable 𝒪 and subbase-of S 𝒪
  shows second-countable S
 using finite-intersections-countable[OF assms(1)] base-from-subbase[OF assms(2)]
  by(auto simp: second-countable-def)

lemma prod-topology-second-countable:
  assumes second-countable S and second-countable S'
  shows second-countable (prod-topology S S')
proof −
  obtain 𝒪 𝒪' where ho:
   countable 𝒪 base-of S 𝒪 countable 𝒪' base-of S' 𝒪'
    using assms by(auto simp: second-countable-def)
  show ?thesis
  proof(rule countable-base-from-countable-subbase[where 𝒪={ U × V | U V. U
∈ 𝒪 ∧ V ∈ 𝒪'}])
    have {U × V |U V. U ∈ 𝒪 ∧ V ∈ 𝒪'} = (λ(U,V). U × V) ‘ (𝒪 × 𝒪')
      by auto
    also have countable ...
      using ho(1,3) by auto
    finally show countable {U × V |U V. U ∈ 𝒪 ∧ V ∈ 𝒪'} .
  next
    show subbase-of (prod-topology S S') {U × V |U V. U ∈ 𝒪 ∧ V ∈ 𝒪'}
      using base-is-subbase[OF ho(2)] base-is-subbase[OF ho(4)]
      by(simp add: subbase-of-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-of*:
assumes *countable* I **and** $\bigwedge i.\ i \in I \Longrightarrow$ *second-countable* $(S\ i)$
shows $\exists \mathcal{O}'.$ *countable* $\mathcal{O}' \wedge$ *base-of* (*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-of* $(S\ i)\ (\mathcal{O}\ i)$
using *assms*(2)[*simplified second-countable-def*] **by** *metis*
show *?thesis*
unfolding *second-countable-def*
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 $U \in \{U.\ \mathit{finite}\ \{i \in I.\ U\ i \neq \mathit{topspace}\ (S\ i)\} \wedge (\forall\, i{\in}\{i \in I.\ U\ i \neq \mathit{topspace}\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\}$
then show $U \in \{U.\ \exists\, I'.\ \mathit{finite}\ I' \wedge I' \subseteq I \wedge (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i = \mathit{topspace}\ (S\ i)) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\}$
by *auto*
next
fix *U*
assume *assm:*$U \in \{U.\ \exists\, I'.\ \mathit{finite}\ I' \wedge I' \subseteq I \wedge (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i = \mathit{topspace}\ (S\ i)) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\}$
then obtain I' **where** *hi'*:
$\mathit{finite}\ I'\ I' \subseteq I\ \forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i\ \forall\, i{\in}I - I'.\ U\ i = \mathit{topspace}\ (S\ i)\ \forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\}$
by *auto*
then have $\bigwedge i.\ i \in I \Longrightarrow U\ i \neq \mathit{topspace}\ (S\ i) \Longrightarrow i \in I'$ **by** *auto*
hence $\{i \in I.\ U\ i \neq \mathit{topspace}\ (S\ i)\} \subseteq I'$ **by** *auto*
hence $\mathit{finite}\ \{i \in I.\ U\ i \neq \mathit{topspace}\ (S\ i)\}$
using *hi'*(*1*) **by** (*simp add*: *rev-finite-subset*)
thus $U \in \{U.\ \mathit{finite}\ \{i \in I.\ U\ i \neq \mathit{topspace}\ (S\ i)\} \wedge (\forall\, i{\in}\{i \in I.\ U\ i \neq \mathit{topspace}\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\}$
using *hi'* **by** *auto*
qed
qed
also have *...* $= (\bigcup I'{\in}\{I'.\ \mathit{finite}\ I' \wedge I' \subseteq I\}.\ \{U.\ (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i = \mathit{topspace}\ (S\ i)) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\})$
by *auto*
also have *countable ...*
proof(*rule countable-UN*[*OF countable-Collect-finite-subset*[*OF assms*(*1*)]])
fix I'
assume $I' \in \{I'.\ \mathit{finite}\ I' \wedge I' \subseteq I\}$
hence *hi'*:*finite* $I'\ I' \subseteq I$ **by** *auto*
have $(\lambda U\ i.\ \mathit{if}\ i \in I'\ \mathit{then}\ U\ i\ \mathit{else}\ \mathit{undefined})\ `\ \{U.\ (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i = \mathit{topspace}\ (S\ i)) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\} \subseteq (\Pi_E\ i{\in}I'.\ \mathcal{O}\ i)$
by *auto*
moreover have *countable ...*
using *hi'* **by**(*auto intro*!: *countable-PiE ho*)
ultimately have *countable* $((\lambda U\ i.\ \mathit{if}\ i \in I'\ \mathit{then}\ U\ i\ \mathit{else}\ \mathit{undefined})\ `\ \{U.\ (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i = \mathit{topspace}\ (S\ i)) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\})$
by(*simp add*: *countable-subset*)
moreover have *inj-on* $(\lambda U\ i.\ \mathit{if}\ i \in I'\ \mathit{then}\ U\ i\ \mathit{else}\ \mathit{undefined})\ \{U.\ (\forall\, i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall\, i{\in}I - I'.\ U\ i = \mathit{topspace}\ (S\ i)) \wedge (\forall\, i{\in}\mathit{UNIV} - I.\ U\ i = \{\mathit{undefined}\})\}$
(**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 $i \in I' \mid i \in I - I' \mid i \in UNIV - I$
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* $\{U.\ (\forall i{\in}I'.\ U\ i \in \mathcal{O}\ i) \wedge (\forall i{\in}I - I'.\ U\ i = topspace\ (S\ i)) \wedge (\forall i{\in}UNIV - I.\ U\ i = \{undefined\})\}$
using *countable-image-inj-on* **by** *auto*
qed
finally show *countable* $\{U.\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall i{\in}\{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i) \wedge (\forall i{\in}UNIV - I.\ U\ i = \{undefined\})\}$ **.**
qed
finally show *?thesis* **.**
qed
next
show *base-of* (*product-topology S I*) $\{\Pi_E\ i{\in}I.\ U\ i \mid U.\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall i{\in}\{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i)\}$
(**is** *base-of* (*product-topology S I*) *?X*)
unfolding *base-of-def*
proof *safe*
fix U
assume *openin* (*product-topology S I*) U
then have $\forall x{\in}U.\ \exists Ux.\ finite\ \{i \in I.\ Ux\ i \neq topspace\ (S\ i)\} \wedge (\forall i{\in}I.\ openin\ (S\ i)\ (Ux\ i)) \wedge x \in Pi_E\ I\ Ux \wedge Pi_E\ I\ Ux \subseteq U$
by(*simp add*: *openin-product-topology-alt*)
hence $\exists Ux.\ \forall x{\in}U.\ finite\ \{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\} \wedge (\forall i{\in}I.\ openin\ (S\ i)\ (Ux\ x\ i)) \wedge x \in Pi_E\ I\ (Ux\ x) \wedge Pi_E\ I\ (Ux\ x) \subseteq U$
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*:$\forall x{\in}U.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}.\ \exists \mathcal{U}xj.\ \mathcal{U}xj \subseteq \mathcal{O}\ i \wedge Ux\ x\ i = \bigcup\ \mathcal{U}xj$

using *ho*[*simplified base-of-def*] **by** (*metis* (*no-types*, *lifting*) *mem-Collect-eq*)

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

by(*standard*, *rule bchoice*) (*use 1* **in** *simp*)

hence $\exists \mathcal{U}xj.\ \forall x{\in}U.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}.\ \mathcal{U}xj\ x\ i \subseteq \mathcal{O}\ i \wedge Ux\ x\ i = \bigcup\ (\mathcal{U}xj\ x\ i)$

by(*rule bchoice*)

then obtain $\mathcal{U}xj$ **where**
$\forall x{\in}U.\ \forall i{\in}\{i \in I.\ Ux\ x\ i \neq topspace\ (S\ i)\}.\ \mathcal{U}xj\ x\ i \subseteq \mathcal{O}\ i \wedge Ux\ x\ i = \bigcup\ (\mathcal{U}xj\ x\ i)$

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 $\exists \mathcal{U}.\ U = \bigcup\ \mathcal{U} \wedge \mathcal{U} \subseteq ?X$

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

proof

fix x

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

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

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

by *auto*

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

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

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

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

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

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 𝒰
    assume 𝒰 ⊆ ?X
    have openin (product-topology S I) u if hu:u ∈ 𝒰 for u
    proof −
      have hu': u ∈ ?X
        using ‹𝒰 ⊆ ?X› hu by auto
      then obtain U where hU:
      u = (Π_E i∈I. U i) finite {i ∈ I. U i ≠ topspace (S i)} ∀ i∈{i ∈ I. U i ≠
topspace (S i)}. U i ∈ 𝒪 i
        by auto
      define U' where U' ≡ (λi. if i ∈ {i ∈ I. U i ≠ topspace (S i)} then U i
else topspace (S i))
      have hU': u = (Π_E i∈I. U' i)
        by(auto simp: hU(1) U'-def PiE-def Pi-def)
      have hUfinite : finite {i. U' i ≠ topspace (S i)}
        using hU(2) by(auto simp: U'-def)
      have hUoi: ∀ i∈{i. U' i ≠ topspace (S i)}. U' i ∈ 𝒪 i
        using hU(3) by(auto simp: U'-def)
      have hUi: ∀ i∈{i. U' i ≠ topspace (S i)}. i ∈ I
        using hU(2) by(auto simp: U'-def)
      have hallopen:openin (S i) (U' i) for i
      proof −
        consider i ∈ {i. U' i ≠ topspace (S i)} | i ∉ {i. U' i ≠ topspace (S i)}
by auto
        then show ?thesis
        proof cases
          case 1
          then show ?thesis
            using hUoi ho(2)[of i] base-of-openin[of S i 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) (⋃ 𝒰)
      by auto
  qed
 next
  show ∀ k∈{Pi_E I U |U. finite {i ∈ I. U i ≠ topspace (S i)} ∧ (∀ i∈{i ∈ I. U
i ≠ topspace (S i)}. U i ∈ 𝒪 i)}. ∃ X. k = Pi_E I X ∧ (∀ i. openin (S i) (X i)) ∧
finite {i. X i ≠ topspace (S i)} ∧ {i. X i ≠ topspace (S i)} ⊆ I
  proof
```

fix k
assume $k \in \{Pi_E\ I\ U\ |U.\ finite\ \{i \in I.\ U\ i \neq topspace\ (S\ i)\} \wedge (\forall\, i{\in}\{i \in I.\ U\ i \neq topspace\ (S\ i)\}.\ U\ i \in \mathcal{O}\ i)\}$
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-of-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** $\bigwedge i.\ i \in I \Longrightarrow second\text{-}countable\ (S\ i)$
shows *second-countable* (*product-topology S I*)
using *product-topology-countable-base-of*[*OF assms*(1)] *assms*(2)
by(*fastforce simp*: *second-countable-def*)

lemma *Cantor-Bendixon*:
assumes *second-countable X*
shows $\exists\, U\ P.\ countable\ U \wedge openin\ X\ U \wedge perfect\text{-}set\ X\ P \wedge U \cup P = topspace\ X \wedge U \cap P = \{\} \wedge (\forall\, a{\neq}\{\}.\ openin\ (subtopology\ X\ P)\ a \longrightarrow uncountable\ a)$
proof −
obtain $\mathcal{O}$ **where** o: *countable* $\mathcal{O}$ *base-of X* $\mathcal{O}$
using *assms* **by**(*auto simp*: *second-countable-def*)
define U **where** $U \equiv \bigcup\ \{u{\in}\mathcal{O}.\ countable\ u\}$
define P **where** $P \equiv topspace\ X - U$
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-of-openin*[*OF o*(2)] **by**(*auto simp*: *U-def*)
have *openin-c*:$countable\ v \longleftrightarrow v \subseteq U$ **if** *openin X v* **for** v
proof
assume *countable v*
obtain $\mathcal{U}$ **where** $v = \bigcup \mathcal{U}\ \mathcal{U} \subseteq \mathcal{O}$
using ‹*openin X v*› $o(2)$ **by**(*auto simp*: *base-of-def*)

with ‹*countable v*› **have** $\bigwedge u.\ u \in \mathcal{U} \Longrightarrow$ *countable u*
by (*meson Sup-upper countable-subset*)
thus $v \subseteq U$
using ‹$\mathcal{U} \subseteq \mathcal{O}$› **by**(*auto simp*: ‹$v = \bigcup \mathcal{U}$› *U-def*)
qed(*rule countable-subset*[*OF - 1*])
have *3*: *perfect-set X P*
proof(*rule perfect-setI*)
fix *x T*
assume *h*:$x \in P\ x \in T$ *openin X T*
have *T-unc*:*uncountable T*
using *openin-c*[*OF h*(*3*)] *h*(*1*,*2*) **by**(*auto simp*: *P-def*)
obtain $\mathcal{U}$ **where** *U*:$T = \bigcup \mathcal{U}\ \mathcal{U} \subseteq \mathcal{O}$
using *h*(*3*) *o*(*2*) **by**(*auto simp*: *base-of-def*)
then obtain *u* **where** *u*:$u \in \mathcal{U}$ *uncountable u*
using *T-unc U-def h*(*3*) *openin-c* **by** *auto*
hence *uncountable* $(u - \{x\})$ **by** *simp*
hence $\neg\ (u - \{x\} \subseteq U)$
using *1* **by** (*metis countable-subset*)
then obtain *y* **where** $y \in u - \{x\}\ y \notin U$
by *blast*
thus $\exists\, y.\ y \neq x \wedge y \in T \wedge y \in P$
using *U u base-of-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* $\neq \{\}$ **for** *a*
proof
assume *contable*:*countable a*
obtain *b* **where** *b*: *openin X b a* $= P \cap b$
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 Dense and Separable in Abstract Topology

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

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

lemma *dense-of-subset*:
assumes *dense-of S U*
shows $U \subseteq$ *topspace S*
using *assms* **by**(*simp add*: *dense-of-def*)

lemma *dense-of-nonempty*:
assumes *topspace S* $\neq \{\}$ *dense-of S U*
shows $U \neq \{\}$
using *assms* **by**(*auto simp*: *dense-of-def*)

definition *separable* :: *'a topology* $\Rightarrow$ *bool* **where**
separable S $\longleftrightarrow$ $(\exists\, U.$ *countable U* $\wedge$ *dense-of S U*$)$

lemma *dense-ofI*:
assumes $U \subseteq$ *topspace S*
and $\bigwedge V.$ *openin S V* $\Longrightarrow$ $V \neq \{\} \Longrightarrow U \cap V \neq \{\}$
shows *dense-of S U*
using *assms* **by**(*auto simp*: *dense-of-def*)

lemma *separable-if-second-countable*:
assumes *second-countable S*
shows *separable S*
proof −
obtain $\mathcal{O}$ **where** *ho*:
countable $\mathcal{O}$ *base-of S* $\mathcal{O}$ $\bigwedge u.\ u \in \mathcal{O} \Longrightarrow u \neq \{\}$
using *second-countable-ex-without-empty*[*OF assms*] **by** *auto*
then obtain *x* **where** *hx*: $\bigwedge u.\ u \in \mathcal{O} \Longrightarrow x\ u \in u$
by (*metis all-not-in-conv*)
show *?thesis*
unfolding *separable-def*
proof(*intro exI*[**where** x=$\{x\ u | u.\ u \in \mathcal{O}\}$] *conjI*)
show *countable* $\{x\ u\ |u.\ u \in \mathcal{O}\}$
using *ho*(*1*) **by** (*simp add*: *Setcompr-eq-image*)
next
show *dense-of S* $\{x\ u\ |u.\ u \in \mathcal{O}\}$
proof(*rule dense-ofI*)
show $\{x\ u\ |u.\ u \in \mathcal{O}\} \subseteq$ *topspace S*
using *hx base-of-subset*[*OF ho*(*2*)] **by** *auto*
next
fix *V*
assume *openin S V V* $\neq \{\}$
then obtain *B* **where** *hb*:$B \subseteq \mathcal{O}$ $V = \bigcup B$
using *base-of-def2'* *ho*(*2*) **by** *metis*
with ‹$V \neq \{\}$› **obtain** *b* **where** $b \in B$
by *auto*
hence $\{x\ u\ |u.\ u \in \mathcal{O}\} \cap b \subseteq \{x\ u\ |u.\ u \in \mathcal{O}\} \cap V$
using *hb*(*2*) **by** *auto*
moreover have $x\ b \in \{x\ u\ |u.\ u \in \mathcal{O}\} \cap b$

```
      using hb(1) ‹b ∈ B› hx[of b] by auto
    ultimately show {x u |u. u ∈ 𝒪} ∩ V ≠ {}
      by auto
  qed
 qed
qed

lemma dense-of-prod:
 assumes dense-of S U and dense-of S' U'
 shows dense-of (prod-topology S S') (U × U')
proof(rule dense-ofI)
 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-of-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-of-def)
qed

lemma separable-prod:
 assumes separable S and separable S'
 shows separable (prod-topology S S')
proof −
 obtain U U' where
  countable U dense-of S U countable U' dense-of S' U'
  using assms by(auto simp: separable-def)
 thus ?thesis
  by(auto intro!: exI[where x=U×U'] dense-of-prod simp: separable-def)
qed

lemma dense-of-product:
 assumes ⋀i. i ∈ I ⟹ dense-of (T i) (U i)
 shows dense-of (product-topology T I) (Π_E i∈I. U i)
proof(rule dense-ofI)
 fix V
 assume h:openin (product-topology T I) V V ≠ {}
 then obtain x where hx:x ∈ V by auto
 then obtain K where hk:
  finite {i ∈ I. K i ≠ topspace (T i)} ∀ i∈I. openin (T i) (K i) x ∈ (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 ⋀i. i ∈ I ⟹ K i ≠ {} by auto
  hence ⋀i. i ∈ I ⟹ U i ∩ K i ≠ {}
    using assms hk by(auto simp: dense-of-def)
  hence (Π_E i∈I. U i) ∩ (Π_E i∈I. K i) ≠ {}
    by (simp add: PiE-Int PiE-eq-empty-iff)
  thus (Π_E i∈I. U i) ∩ V ≠ {}
    using hk by auto
next
  show (Π_E i∈I. U i) ⊆ topspace (product-topology T I)
    using assms by(auto simp: dense-of-def)
qed

lemma separable-countable-product:
  assumes countable I and ⋀i. i ∈ I ⟹ separable (T i)
  shows separable (product-topology T I)
proof −
  consider ∃ i∈I. topspace (T i) = {} | ⋀i. i ∈ I ⟹ topspace (T i) ≠ {}
    by auto
  thus ?thesis
  proof cases
    case 1
    then obtain i where i:i ∈ I topspace (T i) = {}
      by auto
    show ?thesis
      unfolding separable-def dense-of-def
    proof(intro exI[where x={}] conjI)
      show  ∀ V. openin (product-topology T I) V ⟶ V ≠ {} ⟶ {} ∩ V ≠ {}
      proof safe
        fix V x
        assume h: openin (product-topology T I) V x ∈ V
        from i have topspace (product-topology T I) = {}
          by auto
        with h(1) have V = {}
          using openin-subset by blast
        thus x ∈ {}
          using h(2) by auto
      qed
    qed auto
  next
    case 2
    then have ∃ x. ∀ i∈I. x i ∈ topspace (T i)
      by (meson all-not-in-conv)
    moreover from 2
    have ∃ U. ∀ i∈I. countable (U i) ∧ dense-of (T i) (U i)
      using assms(2) by(auto intro!: bchoice simp: separable-def)
    ultimately
    obtain x U where hxu:
     ⋀i. i ∈ I ⟹ x i ∈ topspace (T i) ⋀i. i ∈ I ⟹ countable (U i) ⋀i. i ∈ I
⟹ dense-of (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-def
  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-of (product-topology T I) (⋃ {Π_E i∈I. U' J i |J. finite J ∧ J ⊆
I})
    proof(rule dense-ofI)
      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-of-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 (λ*i*. *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-of-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** ⋀*i*. *i* ∈ *I* ⟹ *separable* (*T i*)
shows *separable* (*product-topology T I*)
using *separable-countable-product*[*OF countable-finite*[*OF assms*(*1*)]] *assms* **by** *auto*

lemma *homeomorphic-separable*:
assumes *separable X X homeomorphic-space Y*
shows *separable Y*
proof −
obtain *f g* **where** *homeomorphic-maps X Y f g*
using *assms*(*2*) **by**(*auto simp*: *homeomorphic-space-def*)
hence *fg*:*continuous-map X Y f continuous-map Y X g* ⋀*x*. *x* ∈ *topspace X* ⟹ *g*(*f x*) = *x* ⋀*y*. *y* ∈ *topspace Y* ⟹ *f*(*g y*) = *y*

by(*auto simp*: *homeomorphic-maps-def*)
obtain *U* **where** *U*: *countable U dense-of X U*
using *assms*(*1*) **by**(*auto simp*: *separable-def*)
show *?thesis*
unfolding *separable-def dense-of-def countable-image*[*OF U*(*1*)]
proof(*intro exI*[**where** *x*=*f* ‘ *U*] *conjI*)
show *f* ‘ *U* ⊆ *topspace Y*
using *U*(*2*) *fg*(*1*) **by**(*auto simp*: *dense-of-def continuous-map-def*)
next
show ∀ *V*. *openin Y V* ⟶ *V* ≠ {} ⟶ *f* ‘ *U* ∩ *V* ≠ {}
proof *safe*
fix *V x*
assume *h*:*openin Y V f* ‘ *U* ∩ *V* = {} *x* ∈ *V*
then have *U* ∩ (*f* −‘ *V* ∩ *topspace X*) = {}
by *blast*
moreover have *f* −‘ *V* ∩ *topspace X* ≠ {}
using *continuous-map-preimage-topspace fg*(*2*) *fg*(*4*) *h*(*1*) *h*(*3*) *openin-subset*
by *fastforce*
moreover have *openin X* (*f* −‘ *V* ∩ *topspace X*)
using *h*(*1*) *fg*(*1*) **by** *auto*
ultimately show *x* ∈ {}
using *U*(*2*) **by**(*auto simp*: *dense-of-def*)
qed
qed(*rule countable-image*[*OF U*(*1*)])
qed

1.1.6 G_δ Set in Abstract Topology

definition *g-delta-of* :: [*'a topology*, *'a set*] ⇒ *bool* **where**
g-delta-of S A ⟷ (∃*U*. *U* ≠ {} ∧ *countable U* ∧ (∀ *b*∈*U*. *openin S b*) ∧ *A* = ⋂ *U*)

lemma *g-delta-ofI*:
assumes *U* ≠ {} *countable U* ⋀*b*. *b* ∈ *U* ⟹ *openin S b A* = ⋂ *U*
shows *g-delta-of S A*
using *assms* **by**(*auto simp*: *g-delta-of-def*)

lemma *g-delta-ofD*:
assumes *g-delta-of S A*
shows ∃*U*. *U* ≠ {} ∧ *countable U* ∧ (∀ *b*∈*U*. *openin S b*) ∧ *A* = ⋂ *U*
using *assms* **by**(*simp add*: *g-delta-of-def*)

lemma *g-delta-ofD'*:
assumes *g-delta-of S A*
shows ∃ *U*. (∀ *n*::*nat*. *openin S* (*U n*)) ∧ *A* = ⋂ (*range U*)
proof−
obtain *U* **where** *h*:*U* ≠ {} *countable U* ⋀*b*. *b*∈*U* ⟹ *openin S b A* = ⋂ *U*
using *g-delta-ofD*[*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 *g-delta-of-subset*:
assumes *g-delta-of S A*
shows $A \subseteq$ *topspace S*
using *assms openin-subset* **by**(*auto simp*: *g-delta-of-def*)

lemma *g-delta-of-open-set*[*simp*]:
assumes *openin S A*
shows *g-delta-of S A*
using *assms* **by**(*auto simp*: *g-delta-of-def intro*!: *exI*[**where** *x*={*A*}])

lemma *g-delta-of-empty*[*simp*]: *g-delta-of S* {}
by *simp*

lemma *g-delta-of-topspace*[*simp*]: *g-delta-of S* (*topspace S*)
by *simp*

lemma *g-delta-of-inter*:
assumes *g-delta-of S A* **and** *g-delta-of S B*
shows *g-delta-of S* ($A \cap B$)
proof −
obtain *Ua Ub* **where** *hu*:
Ua ≠ {} *countable Ua* $\bigwedge$*b*. *b* ∈ *Ua* ⟹ *openin S b A* = $\bigcap$ *Ua*
countable Ub $\bigwedge$*b*. *b* ∈ *Ub* ⟹ *openin S b B* = $\bigcap$ *Ub*
using *assms* **by**(*auto simp*: *g-delta-of-def*)
thus *?thesis*
by(*auto intro*!: *g-delta-ofI*[**where** *U*=*Ua* ∪ *Ub*])
qed

lemma *g-delta-of-Int*:
assumes $\bigwedge$*a*. *a* ∈ $\mathcal{U}$ ⟹ *g-delta-of X a countable* $\mathcal{U}$ $\mathcal{U}$ ≠ {}
shows *g-delta-of X* ($\bigcap$ $\mathcal{U}$)
proof −
obtain *Ua* **where** *u*:
$\bigwedge$*a*. *a* ∈ $\mathcal{U}$ ⟹ *Ua a* ≠ {} $\bigwedge$*a*. *a* ∈ $\mathcal{U}$ ⟹ *countable* (*Ua a*) $\bigwedge$*a b*. *a* ∈ $\mathcal{U}$ ⟹ *b* ∈ *Ua a* ⟹ *openin X b* $\bigwedge$*a*. *a* ∈ $\mathcal{U}$ ⟹ *a* = $\bigcap$ (*Ua a*)
using *g-delta-ofD*[*OF assms*(*1*)] **by** *metis*
have *1*: $\bigcup$ {*Ua a* |*a*. *a* ∈ $\mathcal{U}$} ≠ {}
using *assms*(*3*) *u*(*1*) **by** *auto*
have *2*: *countable* ($\bigcup$ {*Ua a* |*a*. *a* ∈ $\mathcal{U}$})
by (*simp add*: *Setcompr-eq-image assms*(*2*) *u*(*2*))
have *3*: $\bigwedge$*b*. *b* ∈ $\bigcup$ {*Ua a* |*a*. *a* ∈ $\mathcal{U}$} ⟹ *openin X b*
using *u*(*3*) **by** *auto*
show *?thesis*
using *u*(*4*) **by**(*fastforce intro*!: *g-delta-ofI*[*OF 1 2 3*])
qed

lemma *g-delta-of-continuous-map*:
assumes *continuous-map X Y f g-delta-of Y a*
shows *g-delta-of X* (*f* −‘ *a* ∩ *topspace X*)
proof −
obtain *Ua* **where** *u*:
Ua ≠ {} *countable Ua* ⋀*b*. *b* ∈ *Ua* ⟹ *openin Y b a* = ⋂ *Ua*
using *g-delta-ofD*[*OF assms*(*2*)] **by** *metis*
then have *0*:*f* −‘ *a* ∩ *topspace X* = ⋂ {*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*:⋀*c*. *c* ∈ {*f* −‘ *b* ∩ *topspace X*|*b*. *b* ∈ *Ua*} ⟹ *openin X c*
using *assms u*(*3*) **by** *blast*
show *?thesis*
using *g-delta-ofI*[*OF 1 2 3*] **by**(*simp add*: *0*)
qed

lemma *g-delta-of-inj-open-map*:
assumes *open-map X Y f inj-on f* (*topspace X*) *g-delta-of X a*
shows *g-delta-of Y* (*f* ‘ *a*)
proof −
obtain *Ua* **where** *u*:
Ua ≠ {} *countable Ua* ⋀*b*. *b* ∈ *Ua* ⟹ *openin X b a* = ⋂ *Ua*
using *g-delta-ofD*[*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*
using *g-delta-ofI*[*OF 1 2 3*] **by**(*simp add*: *0*)
qed

lemma *g-delta-of-homeo-morphic*:
assumes *g-delta-of X a homeomorphic-map X Y f*
shows *g-delta-of Y* (*f* ‘ *a*)
by(*auto intro*!: *g-delta-of-inj-open-map*[*of X Y f*] *simp*: *assms*(*1*) *homeomorphic-imp-injective-map*[*OF assms*(*2*)] *homeomorphic-imp-open-map*[*OF assms*(*2*)])

lemma *g-delta-of-prod*:
assumes *g-delta-of X A g-delta-of Y B*
shows *g-delta-of* (*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*
using *assms* **by**(*auto simp*: *g-delta-of-def*)
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*
using *g-delta-ofI*[*OF 1 2 3*] **by**(*simp add*: *0*)
qed

lemma *g-delta-of-prod1*:
assumes *g-delta-of X A*
shows *g-delta-of* (*prod-topology X Y*) (*A* × *topspace Y*)
by(*auto intro*!: *g-delta-of-prod assms*)

lemma *g-delta-of-prod2*:
assumes *g-delta-of Y B*
shows *g-delta-of* (*prod-topology X Y*) (*topspace X* × *B*)
by(*auto intro*!: *g-delta-of-prod assms*)

lemma *g-delta-of-subtopology*:
assumes *g-delta-of X A A* ⊆ *S*
shows *g-delta-of* (*subtopology X S*) *A*
proof −
obtain *Ua* **where** *u*:
Ua ≠ {} *countable Ua* ⋀*b*. *b* ∈ *Ua* ⟹ *openin X b A* = ⋂ *Ua*
using *g-delta-ofD*[*OF assms*(*1*)] **by** *metis*
have *0*: ⋂ *Ua* = ⋂ {*ua* ∩ *S* | *ua*. *ua* ∈ *Ua* }
using *assms*(*2*) *u*(*4*) **by** *auto*
have *1*: {*ua* ∩ *S* | *ua*. *ua* ∈ *Ua* } ≠ {}
using *u*(*1*) **by** *auto*
have *2*: *countable* {*ua* ∩ *S* | *ua*. *ua* ∈ *Ua* }
using *u*(*2*) **by** (*simp add*: *Setcompr-eq-image*)

have *3*: $\bigwedge b.\ b \in \{ua \cap S \mid ua.\ ua \in Ua\} \Longrightarrow$ *openin* (*subtopology X S*) *b*
using *u*(*3*) **by**(*auto simp*: *openin-subtopology*)
show *?thesis*
using *g-delta-ofI*[*OF 1 2 3 0*] **by**(*simp add*: *u*(*4*))
qed

lemma *g-delta-of-subtopology-inverse*:
assumes *g-delta-of* (*subtopology X S*) *A g-delta-of X S*
shows *g-delta-of X A*
proof −
obtain *Ua* **where** *ua*:
Ua ≠ {} *countable Ua* $\bigwedge b.\ b \in Ua \Longrightarrow$ *openin* (*subtopology X S*) *b A* = $\bigcap$ *Ua*
using *g-delta-ofD*[*OF assms*(*1*)] **by** *metis*
then obtain *T* **where** *t*: $\bigwedge b.\ b \in Ua \Longrightarrow$ *openin X* (*T b*) $\bigwedge b.\ b \in Ua \Longrightarrow b =$ *T b* ∩ *S*
by(*auto simp*: *openin-subtopology*) *metis*
have *0*: *A* = $\bigcap$ {*T b*|*b*. *b* ∈ *Ua*} ∩ *S*
using *ua*(*1*,*4*) *t*(*2*) **by** *blast*
have {*T b* |*b*. *b* ∈ *Ua*} ≠ {} *countable* {*T b* |*b*. *b* ∈ *Ua*}
using *ua*(*1*,*2*) **by**(*simp-all add*: *Setcompr-eq-image*)
from *g-delta-ofI*[*OF this*] *t*(*1*) **show** *?thesis*
by(*auto intro*!: *g-delta-of-inter*[*OF - assms*(*2*)] *simp*: *0*)
qed

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 Upper-Semicontinuous

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 $\{x \in \mathit{topspace}\ X.\ f\ x < a\} = \mathit{topspace}\ X - \{x \in \mathit{topspace}\ X.\ f\ x \geq a\}$ **for** a
by *auto*
thus *?thesis*
by (*simp add*: *closedin-def upper-semicontinuous-map-def*)
qed

lemma *upper-semicontinuous-map-real-iff*:
fixes $f :: 'a \Rightarrow \mathit{real}$
shows *upper-semicontinuous-map* $X\ f \longleftrightarrow$ *upper-semicontinuous-map* $X\ (\lambda x.\ \mathit{ereal}\ (f\ x))$
unfolding *upper-semicontinuous-map-def*
proof *safe*
fix $a :: \mathit{ereal}$
assume h:$\forall\ a$::*real*. *openin* $X\ \{x \in \mathit{topspace}\ X.\ f\ x < a\}$
consider $a = -\infty \mid a = \infty \mid a \neq -\infty \wedge a \neq \infty$ **by** *auto*
then show *openin* $X\ \{x \in \mathit{topspace}\ X.\ \mathit{ereal}\ (f\ x) < a\}$
proof *cases*
case *3*
then have $\mathit{ereal}\ (f\ x) < a \longleftrightarrow f\ x < \mathit{real\text{-}of\text{-}ereal}\ a$ **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 $a :: \mathit{real}$
assume h:$\forall\ a$::*ereal*. *openin* $X\ \{x \in \mathit{topspace}\ X.\ \mathit{ereal}\ (f\ x) < a\}$
then have *openin* $X\ \{x \in \mathit{topspace}\ X.\ \mathit{ereal}\ (f\ x) < \mathit{ereal}\ a\}$
by *blast*
moreover have*ereal* $(f\ x) < \mathit{real\text{-}of\text{-}ereal}\ a \longleftrightarrow f\ x < a$ **for** x
by *auto*
ultimately show *openin* $X\ \{x \in \mathit{topspace}\ X.\ f\ x < a\}$ **by** *auto*
qed

1.2 Lemmas for Limits

lemma *qlim-eq-lim-mono-at-bot*:
fixes $g :: \mathit{rat} \Rightarrow 'a :: \mathit{linorder\text{-}topology}$
assumes *mono* f $(g \longrightarrow a)$ *at-bot* $\bigwedge r$::*rat*. $f\ (\mathit{real\text{-}of\text{-}rat}\ r) = g\ r$
shows $(f \longrightarrow a)$ *at-bot*
proof −
have *mono* g
by(*metis assms*(*1*,*3*) *mono-def of-rat-less-eq*)
have *ga*:$\bigwedge r.\ g\ r \geq a$
proof(*rule ccontr*)
fix r
assume $\neg\ a \leq g\ r$
then have $g\ r < a$ **by** *simp*

```
    from order-topology-class.order-tendstoD(1)[OF assms(2) this]
    obtain Q :: rat where q: ⋀q. q ≤ Q ⟹ g r < g q
      by(auto simp: eventually-at-bot-linorder)
    define q where q ≡ min r Q
    show False
      using q[of q] ‹mono g›
      by(auto simp: q-def mono-def) (meson linorder-not-less min.cobounded1)
  qed
  show ?thesis
  proof(rule decreasing-tendsto)
    show ∀F n in at-bot. a ≤ f n
      unfolding eventually-at-bot-linorder
     by(rule exI[where x=undefined],auto) (metis Ratreal-def assms(1,3) dual-order.trans
ga less-eq-real-def lt-ex monoD of-rat-dense)
  next
    fix x
    assume a < x
    with topological-space-class.topological-tendstoD[OF assms(2),of {..<x}]
    obtain Q :: rat where q: ⋀q. q ≤ Q ⟹ g q < x
      by(auto simp: eventually-at-bot-linorder)
    show ∀F n in at-bot. f n < x
      using q assms(1,3) by(auto intro!: exI[where x=real-of-rat Q] simp: even-
tually-at-bot-linorder) (metis dual-order.refl monoD order-le-less-trans)
  qed
qed

lemma qlim-eq-lim-mono-at-top:
  fixes g :: rat ⇒ 'a :: linorder-topology
  assumes mono f (g ⟶ a) at-top ⋀r::rat. f (real-of-rat r) = g r
  shows (f ⟶ a) at-top
proof −
  have mono g
    by(metis assms(1,3) mono-def of-rat-less-eq)
  have ga:⋀r. g r ≤ a
  proof(rule ccontr)
    fix r
    assume ¬ g r ≤ a
    then have a < g r by simp
    from order-topology-class.order-tendstoD(2)[OF assms(2) this]
    obtain Q :: rat where q: ⋀q. Q ≤ q ⟹ g q < g r
      by(auto simp: eventually-at-top-linorder)
    define q where q ≡ max r Q
    show False
      using q[of q] ‹mono g› by(auto simp: q-def mono-def leD)
  qed
  show ?thesis
  proof(rule increasing-tendsto)
    show ∀F n in at-top. f n ≤ a
      unfolding eventually-at-top-linorder
```

```
    by(rule exI[where x=undefined],auto) (metis (no-types, opaque-lifting) assms(1)
assms(3) dual-order.trans ga gt-ex monoD of-rat-dense order-le-less)
  next
    fix x
    assume x < a
    with topological-space-class.topological-tendstoD[OF assms(2),of {x<..}]
    obtain Q :: rat where q: ⋀q. Q ≤ q ⟹ x < g q
      by(auto simp: eventually-at-top-linorder)
    show ∀F n in at-top. x < f n
     using q assms(1,3) by(auto simp: eventually-at-top-linorder intro!: exI[where
x=real-of-rat Q]) (metis dual-order.refl monoD order-less-le-trans)
  qed
qed

lemma tendsto-enn2real:
  assumes k < top and (f ⟶ k) F
  shows ((λn. enn2real (f n)) ⟶ enn2real k) F
proof −
  have 1:ennreal (enn2real k) = k enn2real k ≥ 0
    using assms(1) by auto
  show ?thesis
    using assms tendsto-enn2real[OF - 1(2),of f]
    by(simp add: 1(1))
qed

lemma LIMSEQ-inverse-not0:
  fixes xn :: nat ⇒ real
  assumes ⋀n. xn n ≠ 0 xn ⟶ x (λn. 1 / (xn n)) ⟶ b
  shows x ≠ 0
proof
  assume x:x = 0
  then have xn:⋀e. e > 0 ⟹ ∃N. ∀n≥N. |xn n| < e
    using LIMSEQ-D[OF assms(2)] by simp
  have ∃N. ∀n≥N. |1 / (xn n) − b| ≥ r if r:r > 0 for r
  proof −
    have 0 < 1 / (r + |b|)
      using that by auto
    with xn[OF this] obtain N where N':⋀n. n ≥ N ⟹ |xn n| < 1 / (r + |b|)
      by auto
    show ?thesis
    proof(rule exI[where x=N])
      show ∀n≥N. r ≤ |1 / xn n − b|
      proof safe
        fix n
        assume n ≥ N
        note N'[OF this]
        hence (r + |b|) * |xn n| < 1
       by (metis ‹0 < 1 / (r + |b|)› mult.commute pos-less-divide-eq zero-less-divide-1-iff)
```

```
      hence 1 / |xn n| > r + |b|
        using assms(1)[of n] by (simp add: less-divide-eq)
      hence r + |b| − |b| < 1 / |xn n| − |b|
        by simp
      also have ... = |1 / xn n| − |b| by simp
      also have ... ≤ |1 / xn n − b| by simp
      finally show  r ≤ |1 / xn n − b|
        by simp
    qed
  qed
 qed
 with LIMSEQ-D[OF assms(3)] show False
  by (metis less-le-not-le linorder-le-cases real-norm-def zero-less-one)
qed

lemma obtain-subsequence:
 fixes xn :: nat ⇒ -
 assumes infinite {n. P n (xn n)}
 obtains a :: nat ⇒ nat where strict-mono a ⋀n. P (a n) (xn (a n))
proof −
 have inf: infinite {n. n > m ∧ P n (xn n)} for m
 proof
  assume finite {n. n > m ∧ P n (xn n)}
  then have finite ({..m} ∪ {n. n > m ∧ P n (xn n)}) by auto
  hence finite {n. P n (xn n)}
   by(auto intro!: finite-subset[where B={..m} ∪ {n. n > m ∧ P n (xn n)}])
  with assms show False by simp
 qed
 define an where an ≡ rec-nat (SOME n. P n (xn n)) (λn an. SOME m. m >
an ∧ P m (xn m))
 have anSome: an (Suc n) = (SOME m. m > an n ∧ P m (xn m)) for n
  by(auto simp: an-def)
 have an1: P (an n) (xn (an n)) for n
 proof(cases n)
  case 0
  obtain m where m:P m (xn m)
   using assms not-finite-existsD by blast
  show ?thesis
   by(simp add: an-def 0,rule someI,rule m)
 next
  case (Suc n')
  obtain m where m:m > an n' P m (xn m)
   using inf not-finite-existsD by blast
  show ?thesis
   by(simp add: Suc anSome, rule someI2[where a=m],auto simp: m)
 qed
 have an2: strict-mono an
  unfolding strict-mono-Suc-iff anSome
 proof safe
```

fix n
obtain m **where** $m{:}m > an\ n\ P\ m\ (xn\ m)$
using *inf not-finite-existsD* **by** *blast*
show $an\ n < (SOME\ m.\ an\ n < m \wedge P\ m\ (xn\ m))$
by (*rule someI2*[**where** $a{=}m$],*auto simp*: m)
qed
show *?thesis*
using *an1 that*[*OF an2*] **by** *auto*
qed

1.3 Lemmas for Measure Theory

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 $\Omega = space\ M$
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

lemma *integral-measurable-subprob-algebra2*:
fixes $f :: - \Rightarrow - \Rightarrow -::\{banach,second\text{-}countable\text{-}topology\}$
assumes [*measurable*]: $(\lambda(x,\ y).\ f\ x\ y) \in borel\text{-}measurable\ (M \bigotimes_M N)\ L \in measurable\ M\ (subprob\text{-}algebra\ N)$
shows $(\lambda x.\ integral^L\ (L\ x)\ (f\ x)) \in borel\text{-}measurable\ M$
proof −
note *integral-measurable-subprob-algebra*[*measurable*]

note *measurable-distr2*[*measurable*]
have $(\lambda x.\ integral^L\ (distr\ (L\ x)\ (M \bigotimes_M N)\ (\lambda y.\ (x,\ y)))\ (\lambda(x,\ y).\ f\ x\ y)) \in$ *borel-measurable M*
by *measurable*
then show $(\lambda x.\ integral^L\ (L\ x)\ (f\ x)) \in$ *borel-measurable M*
by (*rule measurable-cong*[*THEN iffD1*, *rotated*])
(*simp add*: *integral-distr*)
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*]: $a \in A \Longrightarrow a \in$ *sigma-sets-cinter sp A*
| *Top-c*[*simp*]: *sp* ∈ *sigma-sets-cinter sp A*
| *Inter-c*: $(\bigwedge i{::}nat.\ a\ i \in$ *sigma-sets-cinter sp A*$) \Longrightarrow (\bigcap i.\ a\ i) \in$ *sigma-sets-cinter sp A*
| *Union-c*: $(\bigwedge i{::}nat.\ a\ i \in$ *sigma-sets-cinter sp A*$) \Longrightarrow (\bigcup i.\ a\ i) \in$ *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*]: $a \in A \Longrightarrow a \in$ *sigma-sets-cinter-dunion sp A*
| *Top-cd*[*simp*]: *sp* ∈ *sigma-sets-cinter-dunion sp A*
| *Inter-cd*: $(\bigwedge i{::}nat.\ a\ i \in$ *sigma-sets-cinter-dunion sp A*$) \Longrightarrow (\bigcap i.\ a\ i) \in$ *sigma-sets-cinter-dunion sp A*
| *Union-cd*: $(\bigwedge i{::}nat.\ a\ i \in$ *sigma-sets-cinter-dunion sp A*$) \Longrightarrow$ *disjoint-family a* $\Longrightarrow (\bigcup i.\ a\ i) \in$ *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 $a \in$ *sigma-sets-cinter sp A* $b \in$ *sigma-sets-cinter sp A*
shows $a \cap b \in$ *sigma-sets-cinter sp A*
proof −
have *1*:$a \cap b = (\bigcap i{::}nat.$ *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 $a \in$ *sigma-sets-cinter-dunion sp A* $b \in$ *sigma-sets-cinter-dunion sp A*
shows $a \cap b \in$ *sigma-sets-cinter-dunion sp A*
proof −
have *1*:$a \cap b = (\bigcap i{::}nat.$ *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 $a \in$ *sigma-sets-cinter sp A* $b \in$ *sigma-sets-cinter sp A*
shows $a \cup b \in$ *sigma-sets-cinter sp A*
proof −
have *1*:$a \cup b = (\bigcup i{::}nat.$ *if* $i = 0$ *then* a *else* $b)$ **by** *auto*
show *?thesis*
unfolding *1* **by**(*rule Union-c*,*use assms* **in** *auto*)
qed

Measurable isomorphisms.

definition *measurable-isomorphic-map*::['*a measure*, '*b measure*, '*a* ⇒ '*b*] ⇒ *bool*
where
measurable-isomorphic-map M N f ⟷ *bij-betw f* (*space M*) (*space N*) ∧ *f* ∈ *M* →$_M$ *N* ∧ *the-inv-into* (*space M*) *f* ∈ *N* →$_M$ *M*

lemma *measurable-isomorphic-map-sets-cong*:
assumes *sets M* = *sets M'* *sets N* = *sets N'*
shows *measurable-isomorphic-map M N f* ⟷ *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* ∈ *M* →$_M$ *N the-inv-into* (*space M*) *f* ∈ *N* →$_M$ *M*
shows *measurable-isomorphic-map M N f*

using *assms* **by**(*simp add*: *measurable-isomorphic-map-def*)

lemma *measurable-isomorphic-map-byWitness*:
assumes $f \in M \to_M N$ $g \in N \to_M M$ $\bigwedge x.\ x \in space\ M \Longrightarrow g\ (f\ x) = x$ $\bigwedge x.\ x \in space\ N \Longrightarrow f\ (g\ x) = x$
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* ∈ *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* ∈ *N* →$_M$ *M*
using *measurable-cong assms*(*2*) **by** *blast*
qed (*simp-all add*: ∗ *assms*(*1*))
qed

lemma *measurable-isomorphic-map-restrict-space*:
assumes $f \in M \to_M N$ $\bigwedge A.\ A \in sets\ M \Longrightarrow f\ `\ A \in sets\ N$ *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 $\bigwedge A.\ A \in sets\ M \Longrightarrow f\ `\ A \in sets\ N\ f \in M \to_M N$
$\exists g.\ bij\text{-}betw\ g\ (space\ N)\ (space\ M) \wedge g \in N \to_M M \wedge (\forall x \in space\ M.\ g\ (f\ x) = x) \wedge (\forall x \in space\ N.\ f\ (g\ x) = x) \wedge (\forall A \in sets\ N.\ g\ `\ A \in sets\ M)$
proof −
have *h*:*bij-betw f* (*space M*) (*space N*) $f \in M \to_M N$ *the-inv-into* (*space M*) $f \in N \to_M M$
using *assms* **by**(*simp-all add*: *measurable-isomorphic-map-def*)
show $f\ `\ A \in sets\ N$ **if** $A \in sets\ M$ **for** A
proof −
have $f\ `\ A =$ *the-inv-into* (*space M*) $f - `\ A \cap space\ N$
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 ... $\in$ *sets N*
using *that h*(*3*) **by** *auto*
finally show *?thesis* **.**
qed
show $f \in M \to_M N$
using *assms* **by**(*simp add*: *measurable-isomorphic-map-def*)

show $\exists g.\ bij\text{-}betw\ g\ (space\ N)\ (space\ M) \wedge g \in N \to_M M \wedge (\forall x \in space\ M.\ g\ (f\ x) = x) \wedge (\forall x \in space\ N.\ f\ (g\ x) = x) \wedge (\forall A \in sets\ N.\ g\ `\ A \in sets\ M)$
proof(*rule exI*[**where** *x*=*the-inv-into* (*space M*) *f*])
have $*$:*the-inv-into* (*space M*) $f\ `\ A \in sets\ M$ **if** $A \in sets\ N$ **for** A
proof −
have $\bigwedge x.\ x \in space\ M \Longrightarrow$ *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*) $f \in N \to_M M \wedge (\forall x \in space\ M.$ *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'* ∈ *N* →$_M$ *M* **and** *hf*:⋀*x*. *x*∈*space M* ⟹ *f'* (*f x*) = *x* ⋀*x*. *x*∈*space N* ⟹ *f* (*f' x*) = *x*
and [*measurable*]: *g'* ∈ *L* →$_M$ *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* ∈ *M* →$_M$ *N g* ∈ *N* →$_M$ *L*
using *assms* **by**(*auto simp*: *measurable-isomorphic-map-def*)
from *hf hg measurable-space*[*OF* ‹*f* ∈ *M* →$_M$ *N*›] *measurable-space*[*OF* ‹*g'* ∈ *L* →$_M$ *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* ⟷ *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 ∃*f g*. *f* ∈ *M* →$_M$ *N* ∧ *g* ∈ *N* →$_M$ *M* ∧ (∀ *x*∈*space M*. *g* (*f x*) = *x*) ∧ (∀ *y*∈*space N*. *f* (*g y*) = *y*) ∧ (∀ *A*∈*sets M*. *f* ‘ *A* ∈ *sets N*) ∧ (∀ *A*∈*sets N*. *g* ‘ *A* ∈ *sets M*)
using *assms measurable-isomorphic-mapD'*[*of M N*]
by (*metis* (*mono-tags*, *lifting*) *measurable-isomorphic-def*)

lemma *measurable-isomorphic-byWitness*:
assumes *f* ∈ *M* →$_M$ *N* ⋀*x*. *x*∈*space M* ⟹ *g* (*f x*) = *x*
and *g* ∈ *N* →$_M$ *M* ⋀*y*. *y*∈*space N* ⟹ *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 $(\lambda x\ i.$ *if* $i \in I$ *then* $f\ i\ (x\ i)$ *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* $(\Pi_M\ i{\in}I.\ M\ i)\ (\Pi_M\ i{\in}I.\ N\ i)\ (\lambda x\ i.$ *if* $i \in I$ *then* $h\ i\ (x\ i)$ *else undefined*$)$
proof −
obtain h' **where**
$\bigwedge i.\ i \in I \Longrightarrow h'\ i \in (N\ i) \to_M (M\ i)\ \bigwedge i\ x.\ i \in I \Longrightarrow x{\in}space\ (M\ i) \Longrightarrow h'\ i\ (h\ i\ x) = x\ \bigwedge i\ x.\ i \in I \Longrightarrow x{\in}space\ (N\ i) \Longrightarrow h\ i\ (h'\ i\ x) = x$

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* ‹⋀*i*. *i* ∈ *I* ⟹ *h' i* ∈ (*N i*) →$_M$ (*M i*)›]]
simp: *space-PiM PiE-iff extensional-def*)
qed

lemma *measurable-isomorphic-lift-product*:
assumes ⋀*i*. *i* ∈ *I* ⟹ (*M i*) *measurable-isomorphic* (*N i*)
shows (Π$_M$ *i*∈*I*. *M i*) *measurable-isomorphic* (Π$_M$ *i*∈*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* →$_M$ *restrict-space N* (*f* ‘ *space M*)
g ∈ *N* →$_M$ *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* →$_M$ *N g* ∈ *N* →$_M$ *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*)) →$_M$ *M g'* ∈ *restrict-space M* (*g* ‘ (*space N*)) →$_M$ *N*
and *hfg'*:⋀*x*. *x*∈*space M* ⟹ *f'* (*f x*) = *x* ⋀*x*. *x*∈*f* ‘ *space M* ⟹ *f* (*f' x*) = *x*

$\bigwedge x.\ x{\in}space\ N \Longrightarrow g'\ (g\ x) = x\ \bigwedge x.\ x{\in}g\ `\ space\ N \Longrightarrow g\ (g'\ x) = x$
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*:$(g \circ f)\ `\ A \in sets\ M$ **if** $A \in sets\ M$ **for** *A*
using *hset2(2)[OF hset2(1)[OF that]]* **by**(*simp add*: *image-comp*)
define *An* **where** $An \equiv (\lambda n.\ ((g \circ f) \widehat{\ } n)\ `\ (space\ M - g\ `\ (space\ N)))$
define *A* **where** $A \equiv (\bigcup n{\in}UNIV.\ An\ n)$
have $An\ n \in sets\ M$ **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 $An\ (Suc\ n) = (g \circ f)\ `\ (An\ n)$
by(*auto simp add*: *An-def*)
thus *?case*
using *hgfA*[*OF ih*] **by** *simp*
qed
hence *Asets*:$A \in sets\ M$
by(*simp add*: *A-def*)
have *Acompl*:$space\ M - A \subseteq g\ `\ space\ N$
proof −
have $space\ M - A \subseteq space\ M - An\ 0$
by(*auto simp*: *A-def*)
also have $... \subseteq g\ `\ space\ N$
by(*auto simp*: *An-def*)
finally show *?thesis* **.**
qed
define *h* **where** $h \equiv (\lambda x.\ if\ x \in A \cup (-\ space\ M)\ then\ f\ x\ else\ g'\ x)$
define *h'* **where** $h' \equiv (\lambda x.\ if\ x \in f\ `\ A\ then\ f'\ x\ else\ g\ x)$
have *xinA-iff*:$x \in A \longleftrightarrow h\ x \in f\ `\ A$ **if** $x \in space\ M$ **for** *x*
proof
assume $h\ x \in f\ `\ A$
show $x \in A$
proof(*rule ccontr*)
assume $x \notin A$
then have $\bigwedge n.\ x \notin An\ n$
by(*auto simp*: *A-def*)
from *this*[*of 0*] **have** $x \in g\ `\ (space\ N)$
using *that* **by**(*auto simp*: *An-def*)
have $g'\ x \in f\ `\ A$
using ‹$h\ x \in f\ `\ A$› ‹$x \notin A$›
by (*simp add*: *h-def that*)
hence $g\ (g'\ x) \in (g \circ f)\ `\ A$
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 ∈ space M. x ∈ A ∪ (− space M)} ∈ sets M
    using sets.Int-space-eq2[OF Asets] Asets by simp
  moreover have f ∈ restrict-space M {x. x ∈ A ∪ − space M} →M N
    by (simp add: measurable-restrict-space1)
  moreover have g' ∈ restrict-space M {x. x ∉ A ∪ (− space M)} →M N
  proof −
   have sets (restrict-space (restrict-space M (g ‘ space N)) {x. x ∉ A ∪ − space
M}) = sets (restrict-space M (g ‘ space N ∩ {x. x ∉ A ∪ − space M}))
     by(simp add: sets-restrict-restrict-space)
   also have ... = sets (restrict-space M (g ‘ space N ∩ {x. x ∈ space M − A}))
     by (metis Compl-iff DiffE DiffI Un-iff)
   also have ... = sets (restrict-space M {x. x ∈ space M − A})
     by (metis Acompl le-inf-iff mem-Collect-eq subsetI subset-antisym)
   also have ... = sets (restrict-space M {x. x ∉ A ∪ (− space M)})
     by (metis Compl-iff DiffE DiffI Un-iff)
   finally have sets (restrict-space (restrict-space M (g ‘ space N)) {x. x ∉ A ∪
− space M}) = sets (restrict-space M {x. x ∉ A ∪ − space M}) .
  from measurable-cong-sets[OF this refl] measurable-restrict-space1[OF hfg1'(2),of
{x. x ∉ A ∪ − space M}]
   show ?thesis by auto
  qed
  ultimately show h ∈ M →M N
   by(simp add: h-def measurable-If-restrict-space-iff)
 next
  have {x ∈ space N. x ∈ f ‘ A} ∈ sets N
   using sets.Int-space-eq2[OF hset2(1)[OF Asets]] hset2(1)[OF Asets] by simp
  moreover have f' ∈ restrict-space N {x. x ∈ f ‘ A} →M M
  proof −
   have sets (restrict-space (restrict-space N (f ‘ space M)) {x. x ∈ f ‘ A}) =
sets (restrict-space N (f ‘ space M ∩ {x. x ∈ f ‘ A}))
     by(simp add: sets-restrict-restrict-space)
   also have ... = sets (restrict-space N {x. x ∈ f ‘ A})
   proof −
    have f ‘ space M ∩ {x. x ∈ f ‘ A} = {x. x ∈ 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 ∈ f ‘
A}) = sets (restrict-space N {x. x ∈ f ‘ A}) .
  from measurable-cong-sets[OF this refl] measurable-restrict-space1[OF hfg1'(1),of
{x. x ∈ f ‘ A}]
    show ?thesis by auto
  qed
  moreover have g ∈ restrict-space N {x. x ∉ f ‘ A} →M M
    by (simp add: measurable-restrict-space1)
  ultimately show h' ∈ N →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 mea-
surable-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 (Π$_M$ *x*∈*I*. *M*) *measurable-isomorphic* (Π$_M$ (*x*::*nat*)∈*UNIV*. *M*)
proof(*rule measurable-isomorphic-byWitness*[**where** *f*=λ*x n*. *x* (*from-nat-into I n*) **and** *g*=λ*x*. λ*i*∈*I*. *x* (*to-nat-on I i*)])
show (λ*x n*. *x* (*from-nat-into I n*)) ∈ (Π$_M$ *x*∈*I*. *M*) →$_M$ (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 (λ*x*. λ*i*∈*I*. *x* (*to-nat-on I i*)) ∈ (Π$_M$ (*x*::*nat*)∈*UNIV*. *M*) →$_M$ (Π$_M$ *x*∈*I*. *M*)
by(*auto intro*!: *measurable-PiM-single*′)
next
show *x* ∈ *space* (Π$_M$ *x*∈*I*. *M*) ⟹ (λ*i*∈*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* (λ*x*. *M*)) ⟹ (λ*n*. (λ*i*∈*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*=λ*x* (*i*,*j*). *if* (*i*,*j*) ∈ *I* × *J then x i j else undefined* **and** *g*=λ*x 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*) →$_M$ *M a b* **if** *a* ∈ *I b* ∈ *J* **for** *a b*
using *measurable-component-singleton*[*OF that*(*1*),*of* λ*i*. Π$_M$ *j*∈*J*. *M i j*] *measurable-component-singleton*[*OF that*(*2*),*of M a*]
by *auto*
show (λ*x* (*i*, *j*). *if* (*i*, *j*) ∈ *I* × *J then x i j else undefined*) ∈ (Π$_M$ *i*∈*I*. Π$_M$ *j*∈*J*. *M i j*) →$_M$ (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*]: $(\lambda\omega.\ \omega\ (i,\ j))\ \in Pi_M\ (I \times J)\ (\lambda(i,\ j).\ M\ i\ j) \to_M\ M\ i\ j$ **if** $i \in I\ j \in J$ **for** $i\ j$
using *measurable-component-singleton*[*of* $(i,j)\ I \times J\ \lambda(i,\ j).\ M\ i\ j$] *that* **by** *auto*
show $(\lambda x\ i\ j.$ *if* $i \notin I$ *then undefined* j *else if* $j \notin J$ *then undefined else* $x\ (i,\ j)) \in (\Pi_M\ (i,j)\in I\times J.\ M\ i\ j) \to_M\ (\Pi_M\ i\in I.\ \Pi_M\ j\in J.\ M\ i\ j)$
by(*auto intro*!: *measurable-PiM-single'*) (*simp-all add*: *PiE-iff space-PiM extensional-def*)
next
show $x \in$ *space* $(\Pi_M\ i\in I.\ \Pi_M\ j\in J.\ M\ i\ j) \Longrightarrow (\lambda i\ j.$ *if* $i \notin I$ *then undefined* j *else if* $j \notin J$ *then undefined else case* $(i,\ j)$ *of* $(i,\ j) \Rightarrow$ *if* $(i,\ j) \in I \times J$ *then* $x\ i\ j$ *else undefined*$) = x$ **for** x
by *standard*+ (*auto simp*: *space-PiM PiE-def Pi-def extensional-def*)
next
show $y \in$ *space* $(\Pi_M\ (i,j)\in I\times J.\ M\ i\ j) \Longrightarrow (\lambda(i,\ j).$ *if* $(i,\ j) \in I \times J$ *then if* $i \notin I$ *then undefined* j *else if* $j \notin J$ *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: $\bigwedge A.\ A \in sets\ M \Longrightarrow f\ `\ A \in sets\ N\ f \in M \to_M\ N$ *bij-betw* $g$ (*space* $N$) (*space* $M$) $g \in N \to_M\ M\ \bigwedge x.\ x\in space\ M \Longrightarrow g\ (f\ x) = x\ \bigwedge x.\ x\in space\ N \Longrightarrow f\ (g\ x) = x\ \bigwedge A.\ A\in sets\ N \Longrightarrow g\ `\ A \in sets\ M$
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 $x \in sets\ N$
from *h*(*7*)[*OF this*] *assms*(*1*)
have $g\ `\ x \in$ *sigma-sets* (*space* M) U **by** *simp*
hence $f\ `\ (g\ `\ x) \in$ *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* ‹$x \in$ *sets* N›] *h*(*6*) **by**(*fastforce simp*: *image-def*)
ultimately show $x \in$ *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*:$\bigwedge x.\ x \in U \Longrightarrow x \subseteq$ *space* M
by (*simp add*: s'.*sets-into-space*)
fix x
assume *assm*:$x \in$ *sigma-sets* (*space* N) ((‘) f ‘ U)
then show $x \in$ *sets* N
by *induction* (*auto simp*: *assms*(*1*) *h*(*1*))
qed
qed

end

2 Set-Based Metric Spaces

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

2.1 Set-Based Metric Spaces

locale *metric-set* =
fixes S :: *'a set*
and *dist* :: *'a* ⇒ *'a* ⇒ *real*
assumes *dist-geq0*: $\bigwedge x\ y.$ *dist* $x\ y \geq 0$
and *dist-notin*: $\bigwedge x\ y.\ x \notin S \Longrightarrow$ *dist* $x\ y = 0$
and *dist-0*: $\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow (x = y) = (\mathit{dist}\ x\ y = 0)$
and *dist-sym*: $\bigwedge x\ y.$ *dist* $x\ y =$ *dist* $y\ x$
and *dist-tr*: $\bigwedge x\ y\ z.\ x \in S \Longrightarrow y \in S \Longrightarrow z \in S \Longrightarrow$ *dist* $x\ z \leq$ *dist* $x\ y +$ *dist* $y\ z$

lemma *metric-class-metric-set*[*simp*]: *metric-set UNIV dist*
by *standard* (*auto simp*: *dist-commute dist-triangle*)

context *metric-set*
begin

abbreviation *dist-typeclass* ≡ *Real-Vector-Spaces.dist*

lemma *dist-notin'*:
assumes $y \notin S$
shows *dist* $x\ y = 0$
by(*auto simp*: *dist-sym*[*of* $x\ y$] *intro*!: *dist-notin assms*)

lemma *dist-ge0*:
assumes $x \in S\ y \in S$

shows $x \neq y \longleftrightarrow dist\ x\ y > 0$
using *dist-0*[*OF assms*] *dist-geq0*[*of x y*] **by** *auto*

lemma *dist-0'*[*simp*]: $dist\ x\ x = 0$
by(*cases* $x \in S$) (*use dist-notin dist-0* **in** *auto*)

lemma *dist-tr-abs*:
assumes $x \in S\ y \in S\ z \in S$
shows $|dist\ x\ y - dist\ y\ z| \leq dist\ x\ z$
using *dist-tr*[*OF assms*(*1*,*3*,*2*),*simplified dist-sym*[*of z*]] *dist-tr*[*OF assms*(*2*,*1*,*3*),*simplified dist-sym*[*of - x*]]
by *auto*

Ball

definition *open-ball* :: $'a \Rightarrow real \Rightarrow 'a\ set$ **where**
open-ball a r $\equiv$ *if* $a \in S$ *then* $\{x \in S.\ dist\ a\ x < r\}$ *else* $\{\}$

lemma *open-ball-subset-ofS*: *open-ball a* $\varepsilon \subseteq S$
by(*auto simp*: *open-ball-def*)

lemma *open-ballD*:
assumes $x \in$ *open-ball a* ε
shows $dist\ a\ x < \varepsilon$
proof −
have [*simp*]:$a \in S$
apply(*rule ccontr*) **using** *assms* **by**(*simp add*: *open-ball-def*)
show *?thesis*
using *assms* **by**(*simp add*: *open-ball-def*)
qed

lemma *open-ballD'*:
assumes $x \in$ *open-ball a* ε
shows $x \in S\ a \in S\ \varepsilon > 0$
proof −
have *1*:$a \in S$
apply(*rule ccontr*)
using *assms* **by**(*auto simp*: *open-ball-def*)
have *2*:$x \in S$
apply(*rule ccontr*)
using *assms 1* **by**(*auto simp*: *open-ball-def*)
have *3*: $dist\ a\ x < \varepsilon$
using *assms* **by**(*simp add*: *1 2 open-ball-def*)
show $\varepsilon > 0$
apply(*rule ccontr*)
using *3 dist-geq0*[*of a x*] **by** *auto*
show $x \in S\ a \in S$
by *fact+*
qed

lemma *open-ball-inverse*:
$x \in$ *open-ball* $y\ \varepsilon \longleftrightarrow y \in$ *open-ball* $x\ \varepsilon$
proof −
have *0*:$\bigwedge x\ y.\ x \in$ *open-ball* $y\ \varepsilon \Longrightarrow y \in$ *open-ball* $x\ \varepsilon$
proof −
fix $x\ y$
assume *1*:$x \in$ *open-ball* $y\ \varepsilon$
show $y \in$ *open-ball* $x\ \varepsilon$
using *open-ballD'*[*OF 1*] *dist-sym*[*of y x*] *1* **by**(*simp add*: *open-ball-def*)
qed
show *?thesis*
using *0*[*of x y*] *0*[*of y x*] **by** *auto*
qed

lemma *open-ball-ina*[*simp*]:
assumes $a \in S$ **and** $\varepsilon > 0$
shows $a \in$ *open-ball* $a\ \varepsilon$
using *assms dist-0*[*of a a*] **by**(*simp add*: *open-ball-def*)

lemma *open-ball-nin-le*:
assumes $a \in S\ \varepsilon > 0\ b \in S\ b \notin$ *open-ball* $a\ \varepsilon$
shows $\varepsilon \leq$ *dist* $a\ b$
using *assms* **by**(*simp add*: *open-ball-def*)

lemma *open-ball-le*:
assumes $r \leq l$
shows *open-ball* $a\ r \subseteq$ *open-ball* $a\ l$
using *assms* **by**(*auto simp*: *open-ball-def*)

lemma *open-ball-le-0*:
assumes $\varepsilon \leq 0$
shows *open-ball* $a\ \varepsilon = \{\}$
using *assms dist-geq0*[*of a*]
by(*auto simp*: *open-ball-def*) (*meson linorder-not-less order-trans*)

lemma *open-ball-nin*:
assumes $a \notin S$
shows *open-ball* $a\ \varepsilon = \{\}$
by(*simp add*: *open-ball-def assms*)

definition *closed-ball* :: *'a* ⇒ *real* ⇒ *'a set* **where**
closed-ball a r ≡ *if a* ∈ *S then* {*x* ∈ *S*. *dist a x* ≤ *r*} *else* {}

lemma *closed-ball-subset-ofS*:
closed-ball a $\varepsilon \subseteq S$
by(*auto simp*: *closed-ball-def*)

lemma *closed-ballD*:
assumes $x \in$ *closed-ball* $a\ \varepsilon$

shows *dist a x* $\leq \varepsilon$
proof −
have [*simp*]:*a* $\in S$
apply(*rule ccontr*) **using** *assms* **by**(*simp add*: *closed-ball-def*)
show *?thesis*
using *assms* **by**(*simp add*: *closed-ball-def*)
qed

lemma *closed-ballD'*:
assumes *x* $\in$ *closed-ball a* ε
shows *x* $\in S$ *a* $\in S$ $\varepsilon \geq 0$
proof −
have *1*:*a* $\in S$
apply(*rule ccontr*)
using *assms* **by**(*auto simp*: *closed-ball-def*)
have *2*:*x* $\in S$
apply(*rule ccontr*)
using *assms 1* **by**(*auto simp*: *closed-ball-def*)
have *3*: *dist a x* $\leq \varepsilon$
using *assms* **by**(*simp add*: *1 2 closed-ball-def*)
show $\varepsilon \geq 0$
apply(*rule ccontr*)
using *3 dist-geq0*[*of a x*] **by** *auto*
show *x* $\in$ *S a* $\in S$
by *fact*+
qed

lemma *closed-ball-ina*[*simp*]:
assumes *a* $\in S$ **and** $\varepsilon \geq 0$
shows *a* $\in$ *closed-ball a* ε
using *assms dist-0*[*of a a*] **by**(*simp add*: *closed-ball-def*)

lemma *closed-ball-le*:
assumes $r \leq l$
shows *closed-ball a r* $\subseteq$ *closed-ball a l*
using *closed-ballD'*[*of - a r*] *closed-ballD*[*of - a r*] *assms*
by(*fastforce simp*: *closed-ball-def*[*of - l*])

lemma *closed-ball-le-0*:
assumes $\varepsilon < 0$
shows *closed-ball a* $\varepsilon = \{\}$
using *assms dist-geq0*[*of a*]
by(*auto simp*: *closed-ball-def*) (*meson linorder-not-less order-trans*)

lemma *closed-ball-0*:
assumes *a* $\in S$
shows *closed-ball a 0* $= \{a\}$
using *assms dist-0*[*OF assms assms*] *dist-0*[*OF assms*] *dist-geq0*[*of a*] *order-antisym-conv*
by(*auto simp*: *closed-ball-def*)

lemma *closed-ball-nin*:
assumes $a \notin S$
shows *closed-ball a* $\varepsilon = \{\}$
by(*simp add*: *closed-ball-def assms*)

lemma *open-ball-closed-ball*:
open-ball a $\varepsilon \subseteq$ *closed-ball a* ε
using *open-ballD'*[*of - a* ε] *open-ballD*[*of - a* ε]
by(*fastforce simp*: *closed-ball-def*)

lemma *closed-ball-open-ball*:
assumes $e < f$
shows *closed-ball a e* $\subseteq$ *open-ball a f*
using *closed-ballD'*[*of - a e*] *closed-ballD*[*of - a e*] *assms*
by(*fastforce simp*: *open-ball-def*)

lemma *closed-ball-open-ball-un1*:
assumes $e > 0$
shows *open-ball a e* $\cup$ $\{x{\in}S.$ *dist a x* $= e\}$ = *closed-ball a e*
using *assms dist-notin* **by**(*auto simp*: *open-ball-def closed-ball-def*)

lemma *closed-ball-open-ball-un2*:
assumes $a \in S$
shows *open-ball a e* $\cup$ $\{x{\in}S.$ *dist a x* $= e\}$ = *closed-ball a e*
using *assms* **by**(*auto simp*: *open-ball-def closed-ball-def*)

definition *mtopology* :: *'a topology* **where**
mtopology = *topology* $(\lambda U.\ U \subseteq S \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.$ *open-ball x* $\varepsilon \subseteq U))$

lemma *mtopology-istopology*:
istopology $(\lambda U.\ U \subseteq S \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.$ *open-ball x* $\varepsilon \subseteq U))$
unfolding *istopology-def*
proof *safe*
fix *U1 U2 x*
assume *h1*: *U1* $\subseteq S \ \forall y{\in}U1.\ \exists \varepsilon{>}0.$ *open-ball y* $\varepsilon \subseteq$ *U1*
and *h2*: *U2* $\subseteq S \ \forall y{\in}U2.\ \exists \varepsilon{>}0.$ *open-ball y* $\varepsilon \subseteq$ *U2*
and *hx*: $x \in$ *U1* $x \in$ *U2*
obtain *ε1 ε2* **where**
ε1 > 0 *ε2* > 0*open-ball x ε1* $\subseteq$ *U1 open-ball x ε2* $\subseteq$ *U2*
using *h1 h2 hx* **by** *blast*
thus $\exists \varepsilon{>}0.$ *open-ball x* $\varepsilon \subseteq$ *U1* $\cap$ *U2*
using *open-ball-le*[*of min ε1 ε2 ε1 x*] *open-ball-le*[*of min ε1 ε2 ε2 x*]
by(*auto intro*!: *exI*[**where** *x*=*min ε1 ε2*])
next
fix $\mathcal{K}$ *U x*
assume *h*:$\forall K{\in}\mathcal{K}.\ K \subseteq S \wedge (\forall x{\in}K.\ \exists \varepsilon{>}0.$ *open-ball x* $\varepsilon \subseteq K)$
$U \in \mathcal{K}$ $x \in U$
then obtain ε **where**

$\varepsilon > 0$ *open-ball* $x\ \varepsilon \subseteq U$
by *blast*
thus $\exists \varepsilon{>}0.$ *open-ball* $x\ \varepsilon \subseteq \bigcup \mathcal{K}$
using $h(2)$ **by**(*auto intro*!: *exI*[**where** $x{=}\varepsilon$])
qed *auto*

lemma *mtopology-openin-iff*:
openin mtopology $U \longleftrightarrow U \subseteq S \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.$ *open-ball* $x\ \varepsilon \subseteq U)$
by (*simp add*: *mtopology-def mtopology-istopology*)

lemma *mtopology-topspace*: *topspace mtopology* $= S$
unfolding *topspace-def mtopology-def topology-inverse'*[*OF mtopology-istopology*]
proof −
have $\forall x{\in}S.\ \exists \varepsilon{>}0.$ *open-ball* $x\ \varepsilon \subseteq S$
by(*auto intro*!: *exI*[**where** $x{=}1$] *simp*: *open-ball-def*)
thus $\bigcup \{U.\ U \subseteq S \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.$ *open-ball* $x\ \varepsilon \subseteq U)\} = S$
by *auto*
qed

lemma *openin-S*[*simp*]: *openin mtopology S*
by (*metis openin-topspace mtopology-topspace*)

lemma *mtopology-open-ball-in'*:
assumes $x \in$ *open-ball* $a\ \varepsilon$
shows $\exists \varepsilon'{>}0.$ *open-ball* $x\ \varepsilon' \subseteq$ *open-ball* $a\ \varepsilon$
proof −
show $\exists \varepsilon'{>}0.$ *open-ball* $x\ \varepsilon' \subseteq$ *open-ball* $a\ \varepsilon$
proof(*intro exI*[**where** $x{=}\varepsilon - dist\ a\ x$] *conjI*)
show $0 < \varepsilon - dist\ a\ x$
using *open-ballD'*[*OF assms*] *open-ballD*[*OF assms*] **by** *auto*
next
show *open-ball* $x\ (\varepsilon - dist\ a\ x) \subseteq$ *open-ball* $a\ \varepsilon$
proof
fix y
assume *hy*:$y \in$ *open-ball* $x\ (\varepsilon - dist\ a\ x)$
show $y \in$ *open-ball* $a\ \varepsilon$
using *open-ballD*[*OF hy*] *open-ballD*[*OF assms*] *open-ballD'*(*2*)[*OF assms*] *dist-tr*[*OF open-ballD'*(*2*)[*OF assms*] *open-ballD'*(*1*)[*OF assms*] *open-ballD'*(*1*)[*OF hy*]]
by(*auto simp*: *open-ball-def assms*(*1*) *open-ballD'*[*OF hy*])
qed
qed
qed

lemma *mtopology-open-ball-in*:
assumes $a \in S$ **and** $\varepsilon > 0$
shows *openin mtopology* (*open-ball* $a\ \varepsilon$)
using *mtopology-open-ball-in' topology-inverse'*[*OF mtopology-istopology*] *open-ball-subset-ofS mtopology-def*

```
  by auto

lemma openin-open-ball: openin mtopology (open-ball a ε)
proof −
  consider a ∈ S ∧ ε > 0 | a ∉ S | ε ≤ 0 by fastforce
  thus ?thesis
    by cases (simp-all add: mtopology-open-ball-in open-ball-le-0 open-ball-nin)
qed

lemma closedin-closed-ball: closedin mtopology (closed-ball a ε)
  unfolding closedin-def mtopology-topspace mtopology-openin-iff
proof safe
  fix x
  assume h:x ∈ S x ∉ closed-ball a ε
  consider a ∉ S | ε < 0 | a ∈ S ε ≥ 0  by fastforce
  thus ∃ε'>0. open-ball x ε' ⊆ S − closed-ball a ε
  proof cases
    case 3
    then have dist a x > ε
      using h by(auto simp: closed-ball-def)
    show ?thesis
    proof(intro exI[where x=dist a x − ε] conjI)
      show open-ball x (dist a x − ε) ⊆ S − closed-ball a ε
      proof safe
        fix z
        assume h':z ∈ open-ball x (dist a x − ε) z ∈ closed-ball a ε
        have dist a x ≤ dist a z + dist z x
          by(auto intro!: dist-tr 3 open-ballD'[OF h'(1)])
        also have ... ≤ ε + dist z x
          using closed-ballD[OF h'(2)] by simp
        also have ... < dist a x
          using open-ballD[OF h'(1),simplified dist-sym[of x]] by auto
        finally show False ..
      qed(use open-ball-subset-ofS ‹dist a x > ε› in auto)
    qed(use open-ball-subset-ofS ‹dist a x > ε› in auto)
  qed(auto simp: closed-ball-nin closed-ball-le-0 open-ball-subset-ofS intro!: exI[where
x=1])
qed(use closed-ball-subset-ofS in auto)

lemma mtopology-def2:
 mtopology = topology-generated-by {open-ball a ε | a ε. a ∈ S ∧ ε > 0}
  (is ?lhs = ?rhs)
proof −
  have ⋀U. openin ?lhs U = openin ?rhs U
  proof
    fix U
    assume h:openin mtopology U
    then have ∀x∈ U. ∃ε > 0. open-ball x ε ⊆ U
      using topology-inverse'[OF mtopology-istopology]
```

```
    by(simp add: mtopology-def)
  then obtain ε where he:
    ⋀x. x ∈ U ⟹ ε x > 0 ∧ open-ball x (ε x) ⊆ U
    using bchoice[of U λx ε. ε > 0 ∧ open-ball x ε ⊆ U]
    by blast
  have U = ⋃{open-ball x (ε x)|x. x∈ U}
  proof
    show ⋃ {open-ball x (ε x) |x. x ∈ U} ⊆ U
      using he by auto
  next
    show U ⊆ ⋃ {open-ball x (ε x) |x. x ∈ U}
    proof
      fix a
      assume ha:a ∈ U
      then have a ∈ open-ball a (ε a)
        using he[of a] open-ball-ina[of a ε a] openin-subset[OF h,simplified]
        by(auto simp: mtopology-topspace)
      thus a ∈ ⋃ {open-ball x (ε x) |x. x ∈ U}
        using ha by auto
    qed
  qed
  also have generate-topology-on {open-ball a ε |a ε. a ∈ S ∧ 0 < ε} ...
    apply(rule generate-topology-on.UN)
    apply(rule generate-topology-on.Basis)
    using he openin-subset[OF h,simplified]
    by(fastforce simp: mtopology-topspace)
  finally show openin (topology-generated-by {open-ball a ε |a ε. a ∈ S ∧ 0 <
ε}) U
    by (simp add: openin-topology-generated-by-iff)
 next
  fix U
  assume openin (topology-generated-by {open-ball a ε |a ε. a ∈ S ∧ 0 < ε}) U
  then have generate-topology-on {open-ball a ε |a ε. a ∈ S ∧ 0 < ε} U
    by (simp add: openin-topology-generated-by-iff)
  thus openin mtopology U
    apply induction
    using mtopology-open-ball-in
    by auto
 qed
 thus ?thesis
  by(simp add: topology-eq)
qed

abbreviation mtopology-subbasis :: 'a set set ⇒ bool where
mtopology-subbasis 𝒪 ≡ subbase-of mtopology 𝒪

lemma mtopology-subbasis1:
 mtopology-subbasis {open-ball a ε | a ε. a ∈ S ∧ ε > 0}
  by(simp add: mtopology-def2 subbase-of-def)
```

abbreviation *mtopology-basis* :: *'a set set* ⇒ *bool* **where**
mtopology-basis $\mathcal{O}$ ≡ *base-of mtopology* $\mathcal{O}$

lemma *mtopology-basis-ball*:
mtopology-basis {*open-ball a* ε | *a* ε. *a* ∈ *S* ∧ ε > *0*}
unfolding *base-of-def*
proof −
show ∀ *U*. *openin mtopology U* = (∃$\mathcal{U}$. *U* = ⋃ $\mathcal{U}$ ∧ $\mathcal{U}$ ⊆ {*open-ball a* ε |*a* ε. *a* ∈ *S* ∧ *0* < ε})
proof *safe*
fix *U*
assume *openin mtopology U*
then have *U* ⊆ *S* ⋀*x*. *x*∈*U* ⟹ ∃ε>*0*. *open-ball x* ε ⊆ *U*
by(*auto simp*: *mtopology-openin-iff*)
then obtain ε **where** *he*:
⋀*x*. *x* ∈ *U* ⟹ ε *x* > *0* ⋀*x*. *x* ∈ *U* ⟹ *open-ball x* (ε *x*) ⊆ *U*
by *metis*
hence (⋃ { *open-ball x* (ε *x*) | *x*. *x* ∈ *U*}) = *U*
using ‹*U* ⊆ *S*› *open-ball-ina*[*of* - ε -] **by** *fastforce*
thus ∃$\mathcal{U}$. *U* = ⋃ $\mathcal{U}$ ∧ $\mathcal{U}$ ⊆ {*open-ball a* ε |*a* ε. *a* ∈ *S* ∧ *0* < ε}
using *he*(*1*) ‹*U* ⊆ *S*› **by**(*fastforce intro*!: *exI*[**where** *x*={ *open-ball x* (ε *x*) | *x*. *x* ∈ *U*}])
qed(*use mtopology-open-ball-in* **in** *blast*)
qed

abbreviation *sequence* :: (*nat* ⇒ *'a*) *set* **where**
sequence ≡ *UNIV* → *S*

lemma *sequence-comp*:
xn ∈ *sequence* ⟹ (λ*n*. (*xn* (*a n*))) ∈ *sequence*
xn ∈ *sequence* ⟹ *xn* ∘ *an* ∈ *sequence*
by *auto*

definition *converge-to-inS* :: [*nat* ⇒ *'a*, *'a*] ⇒ *bool* **where**
converge-to-inS f s ≡ *f* ∈ *sequence* ∧ *s* ∈ *S* ∧ (λ*n*. *dist* (*f n*) *s*) ⟶ *0*

lemma *converge-to-inS-const*:
assumes *x* ∈ *S*
shows *converge-to-inS* (λ*n*. *x*) *x*
using *assms dist-0*[*of x x*] **by**(*simp add*: *converge-to-inS-def*)

lemma *converge-to-inS-subseq*:
assumes *strict-mono a converge-to-inS f s*
shows *converge-to-inS* (*f* ∘ *a*) *s*
proof −
have ((λ*n*. *dist* (*f n*) *s*) ∘ *a*) ⟶ *0*
using *assms* **by**(*auto intro*!: *LIMSEQ-subseq-LIMSEQ simp*: *converge-to-inS-def*)
thus *?thesis*

using *assms* **by**(*auto simp*: *converge-to-inS-def comp-def*)
qed

lemma *converge-to-inS-ignore-initial*:
assumes *converge-to-inS xn x*
shows *converge-to-inS* ($\lambda n.$ *xn* (*n* + *k*)) *x*
using *LIMSEQ-ignore-initial-segment*[*of* $\lambda n.$ *dist* (*xn n*) *x 0 k*] *assms*
by(*auto simp*: *converge-to-inS-def*)

lemma *converge-to-inS-offset*:
assumes *converge-to-inS* ($\lambda n.$ *xn* (*n* + *k*)) *x xn* ∈ *sequence*
shows *converge-to-inS xn x*
using *LIMSEQ-offset*[*of* $\lambda n.$ *dist* (*xn n*) *x k*] *assms*
by(*auto simp*: *converge-to-inS-def*)

lemma *converge-to-inS-def2*:
converge-to-inS f s ⟷ (*f* ∈ *sequence* ∧ *s* ∈ *S* ∧ (∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *s* < *ε*))
proof
assume *h*:*converge-to-inS f s*
show *f* ∈ *sequence* ∧ *s* ∈ *S* ∧ (∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *s* < *ε*)
proof *safe*
fix *ε* :: *real*
assume *he*:*0* < *ε*
have *hs*:⋀*S*. *open S* ⟹ *0* ∈ *S* ⟹ (∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *s* ∈ *S*)
using *h lim-explicit*[*of* $\lambda n.$ *dist* (*f n*) *s 0*]
by(*simp add*: *converge-to-inS-def*)
then obtain *N* **where**
∀ *n*≥*N*. *dist* (*f n*) *s* ∈ {−*1*<..<*ε*}
using *hs*[*of* {−*1*<..<*ε*}] *he* **by** *fastforce*
thus ∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *s* < *ε*
by(*auto intro*!: *exI*[**where** *x*=*N*])
qed(*use h*[*simplified converge-to-inS-def*] **in** *auto*)
next
assume *h*:*f* ∈ *sequence* ∧ *s* ∈ *S* ∧ (∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *s* < *ε*)
have ∀ *S*. *open S* ⟶ *0* ∈ *S* ⟶ (∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *s* ∈ *S*)
proof *safe*
fix *S* :: *real set*
assume *hs*:*open S 0* ∈ *S*
then obtain *ε* **where** *he*:
ε > *0 ball 0 ε* ⊆ *S*
using *open-contains-ball*[*of S*] **by** *fastforce*
then obtain *N* **where**
∀ *n*≥*N*. *dist* (*f n*) *s* < *ε*
using *h* **by** *auto*
thus ∃ *N*. ∀ *n*≥*N*. *dist* (*f n*) *s* ∈ *S*
using *he dist-geq0* **by**(*auto intro*!: *exI*[**where** *x*=*N*])
qed
thus *converge-to-inS f s*

using *lim-explicit*[*of* $\lambda n.\ dist\ (f\ n)\ s\ 0$] *h*
by(*simp add*: *converge-to-inS-def*)
qed

lemma *converge-to-inS-def2'*:
$converge\text{-}to\text{-}inS\ f\ s \longleftrightarrow (f \in sequence \wedge s \in S \wedge (\forall \varepsilon{>}0.\ \exists N.\ \forall n{\geq}N.\ (f\ n) \in open\text{-}ball\ s\ \varepsilon))$
unfolding *converge-to-inS-def2 open-ball-def dist-sym*[*of s*]
by *fastforce*

lemma *converge-to-inS-unique*:
assumes *converge-to-inS f x converge-to-inS f y*
shows $x = y$
proof −
have *inS*:$\bigwedge n.\ f\ n \in S\ x \in S\ y \in S$
using *assms* **by**(*auto simp*: *converge-to-inS-def*)
have $|dist\ x\ y| < \varepsilon$ **if** $\varepsilon > 0$ **for** ε
proof −
have $0 < \varepsilon\ /\ 2$ **using** *that* **by** *simp*
then obtain *N1 N2* **where** *hn*:
$\bigwedge n.\ n \geq N1 \Longrightarrow dist\ (f\ n)\ x < \varepsilon\ /\ 2\ \bigwedge n.\ n \geq N2 \Longrightarrow dist\ (f\ n)\ y < \varepsilon\ /\ 2$
using *assms converge-to-inS-def2* **by** *blast*
have $dist\ x\ y \leq dist\ (f\ (max\ N1\ N2))\ x + dist\ (f\ (max\ N1\ N2))\ y$
unfolding *dist-sym*[*of f* (*max N1 N2*) *x*] **by**(*rule dist-tr*[*OF inS*(2) *inS*(1)[*of max N1 N2*] *inS*(3)])
also have $... < \varepsilon\ /\ 2 + \varepsilon\ /\ 2$
by(*rule add-strict-mono*) (*use hn*[*of max N1 N2*] **in** *auto*)
finally show *?thesis*
using *dist-geq0*[*of x y*] **by** *simp*
qed
hence $dist\ x\ y = 0$
using *zero-less-abs-iff* **by** *blast*
thus *?thesis*
using *dist-0*[*OF inS*(2,3)] **by** *simp*
qed

lemma *mtopology-closedin-iff*: $closedin\ mtopology\ M \longleftrightarrow M \subseteq S \wedge (\forall f{\in}(UNIV \to M).\ \forall s.\ converge\text{-}to\text{-}inS\ f\ s \longrightarrow s \in M)$
proof
assume *closedin mtopology M*
then have *h*:$\forall x{\in}S - M.\ \exists \varepsilon{>}0.\ open\text{-}ball\ x\ \varepsilon \subseteq S - M$
by (*simp add*: *closedin-def mtopology-openin-iff mtopology-topspace*)
show $M \subseteq S \wedge (\forall f{\in}UNIV \to M.\ \forall s.\ converge\text{-}to\text{-}inS\ f\ s \longrightarrow s \in M)$
proof *safe*
fix $f :: nat \Rightarrow 'a$ **and** *s*
assume *hf*:$f \in UNIV \to M\ converge\text{-}to\text{-}inS\ f\ s$
show $s \in M$
proof(*rule ccontr*)
assume $s \notin M$

```
      then have s ∈ S − M
        using hf(2) by(auto simp: converge-to-inS-def)
      then obtain ε where ε > 0 open-ball s ε ⊆ S − M
        using h by auto
      then obtain N where ⋀n. n ≥ N ⟹ f n ∈ open-ball s ε
        using hf(2) by(auto simp: converge-to-inS-def2') metis
      from ‹open-ball s ε ⊆ S − M› this[of N] hf(1)
      show False by auto
    qed
  qed(rule subsetD[OF closedin-subset[OF ‹closedin mtopology M›,simplified mtopol-
ogy-topspace]])
next
  assume h:M ⊆ S ∧ (∀f∈UNIV → M. ∀s. converge-to-inS f s ⟶ s ∈ M)
  show closedin mtopology M
    unfolding closedin-def mtopology-openin-iff mtopology-topspace
  proof safe
    fix x
    assume x ∈ S x ∉ M
    show ∃ε>0. open-ball x ε ⊆ S − M
    proof(rule ccontr)
      assume ¬ (∃ε>0. open-ball x ε ⊆ S − M)
      then have ∀ε>0. open-ball x ε ∩ M ≠ {}
        by (metis Diff-mono Diff-triv open-ball-subset-ofS subset-refl)
      hence ∀n. ∃a. a ∈ open-ball x (1 / real (Suc n)) ∩ M
      by (meson of-nat-0-less-iff subsetI subset-empty zero-less-Suc zero-less-divide-1-iff)
      then obtain f where hf:⋀n. f n ∈ open-ball x (1 / (Suc n)) ∩ M by metis
      hence f ∈ UNIV → M by auto
      moreover have converge-to-inS f x
        unfolding converge-to-inS-def2'
      proof safe
        show f x ∈ S for x
          using h hf by auto
      next
        fix ε
        assume (0::real) < ε
        then obtain N where 1 / real (Suc N) < ε
          using nat-approx-posE by blast
        show ∃N. ∀n≥N. f n ∈ open-ball x ε
        proof(rule exI[where x=N])
          show ∀n≥N. f n ∈ open-ball x ε
          proof safe
            fix n
            assume N ≤ n
            then have 1 / real (Suc n) ≤ 1 / real (Suc N)
              by (simp add: frac-le)
            also have ... ≤ ε
              using ‹1 / real (Suc N) < ε› by simp
            finally show f n ∈ open-ball x ε
              using open-ball-le[of 1 / real (Suc n) ε x] hf by auto
```

```
        qed
      qed
    qed fact
    ultimately show False
      using h ‹x ∉ M› by blast
  qed
 qed(use h in auto)
qed

lemma mtopology-closedin-iff2: closedin mtopology M ⟷ M ⊆ S ∧ (∀x. x ∈ M
⟷ (∀ε>0. open-ball x ε ∩ M ≠ {}))
proof
 assume h:closedin mtopology M
 have 1: M ⊆ S
  using h by(auto simp add: mtopology-closedin-iff)
 show M ⊆ S ∧ (∀x. (x ∈ M) = (∀ε>0. open-ball x ε ∩ M ≠ {}))
 proof safe
  fix ε x
  assume x ∈ M (0 :: real) < ε open-ball x ε ∩ M = {}
  thus False
   using open-ball-ina[of x ε] 1 by blast
 next
  fix x
  assume ∀ε>0. open-ball x ε ∩ M ≠ {}
  hence ∃f. f ∈ open-ball x (1 / real (Suc n)) ∩ M for n
  by (meson all-not-in-conv divide-pos-pos of-nat-0-less-iff zero-less-Suc zero-less-one)
  then obtain f where hf:⋀n. f n ∈ open-ball x (1 / real (Suc n)) ∩ M
   by metis
  hence x ∈ S f ∈ UNIV → M
   using open-ballD'(2)[of f 0 x] by auto
  have converge-to-inS f x
   unfolding converge-to-inS-def2'
  proof safe
   show ⋀x. f x ∈ S
    using 1 ‹f ∈ UNIV → M› by auto
  next
   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. f n ∈ open-ball x ε
   proof(rule exI[where x=N])
    show ∀n≥N. f n ∈ open-ball x ε
    proof safe
     fix n
     assume N ≤ n
     then have 1 / real (Suc n) ≤ 1 / real (Suc N)
      using inverse-of-nat-le by blast
     thus f n ∈ open-ball x ε
```

```
          using hf[of n] open-ball-le[of 1 / real (Suc n) ε x] hN
          by auto
        qed
      qed
    qed fact
    with ‹f ∈ UNIV → M› show x ∈ M
      using h[simplified mtopology-closedin-iff] by simp
  qed(use 1 in auto)
next
  assumeM ⊆ S ∧ (∀x. (x ∈ M) ⟷ (∀ε>0. open-ball x ε ∩ M ≠ {}))
  hence h:M ⊆ S ⋀x. (x ∈ M) ⟷ (∀ε>0. open-ball x ε ∩ M ≠ {})
    by simp-all
  show closedin mtopology M
    unfolding mtopology-closedin-iff
  proof safe
    fix f s
    assume h':f ∈ UNIV → M converge-to-inS f s
    hence s ∈ S by(simp add: converge-to-inS-def)
    have open-ball s ε ∩ M ≠ {} if ε > 0 for ε
    proof −
      obtain N where hN:⋀n. n ≥ N ⟹ dist (f n) s < ε
        using h'(2) ‹ε > 0› by(auto simp: converge-to-inS-def2) metis
      have f N ∈ open-ball s ε ∩ M
        using ‹f ∈ UNIV → M› ‹s ∈ S› hN[of N] that open-ball-def[of s ε] h(1)
dist-sym[of s]
        by auto
      thus open-ball s ε ∩ M ≠ {} by auto
    qed
    with h(2)[of s] show s ∈ M by simp
  qed(use h(1) in auto)
qed

lemma mtopology-openin-iff2:
 openin mtopology A ⟷ A ⊆ S ∧ (∀f x. converge-to-inS f x ∧ x ∈ A ⟶ (∃N.
∀n≥N. f n ∈ A))
proof
  show openin mtopology A ⟹ A ⊆ S ∧ (∀f x. converge-to-inS f x ∧ x ∈ A ⟶
(∃N. ∀n≥N. f n ∈ A))
    unfolding mtopology-openin-iff
  proof safe
    fix f x
    assume ∀x∈A. ∃ε>0. open-ball x ε ⊆ A converge-to-inS f x x ∈ A
    then obtain ε where ε > 0 open-ball x ε ⊆ A
      by auto
    then obtain N where ⋀n. n ≥ N ⟹ dist (f n) x < ε
      using ‹converge-to-inS f x› by(fastforce simp: converge-to-inS-def2)
    hence ⋀n. n ≥ N ⟹ f n ∈ open-ball x ε
      using ‹converge-to-inS f x› by(auto simp: dist-sym[of - x] open-ball-def con-
verge-to-inS-def)
```

with ‹*open-ball x ε* ⊆ *A*› **show** ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *A*
by(*auto intro*!: *exI*[**where** *x*=*N*])
qed
next
assume *A* ⊆ *S* ∧ (∀ *f x*. *converge-to-inS f x* ∧ *x* ∈ *A* ⟶ (∃ *N*. ∀ *n*≥*N*. *f n* ∈ *A*))
hence *h*:*A* ⊆ *S* ⋀*f x*. *converge-to-inS f x* ⟹ *x* ∈ *A* ⟹ ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *A*
by *auto*
have *closedin mtopology* (*S* − *A*)
unfolding *mtopology-closedin-iff*
proof *safe*
fix *f s*
assume *hf*:*f* ∈ *UNIV* → *S* − *A*
converge-to-inS f s
have *False* **if** *s* ∈ *A*
proof −
from *h*(*2*)[*OF hf*(*2*) *that*]
obtain *N* **where** ⋀*n*. *n* ≥ *N* ⟹ *f n* ∈ *A* **by** *auto*
from *hf* (*1*) *this*[*of N*] **show** *False* **by** *auto*
qed
thus *s* ∈ *S s* ∈ *A* ⟹ *False*
using *hf*(*2*) **by** (*auto simp*: *converge-to-inS-def*)
qed
thus *openin mtopology A*
using *h*(*1*) *mtopology-topspace* **by**(*simp add*: *openin-closedin-eq*)
qed

lemma *closure-of-mtopology*: *mtopology closure-of A* = {*a*. ∀ *ε*>*0*. *open-ball a ε* ∩ *A* ≠ {}}
proof *safe*
fix *x ε*
assume *x* ∈ *mtopology closure-of A* (*0* :: *real*) < *ε open-ball x ε* ∩ *A* = {}
then show *False*
using *mtopology-closedin-iff2*[*of mtopology closure-of A*,*simplified*]
by (*simp add*: *mtopology-open-ball-in*′ *mtopology-openin-iff open-ball-subset-ofS openin-Int-closure-of-eq-empty*)
next
fix *x*
assume ∀ *ε*>*0*. *open-ball x ε* ∩ *A* ≠ {}
then have ∀ *ε*>*0*. *open-ball x ε* ∩ *mtopology closure-of A* ≠ {}
by (*simp add*: *mtopology-open-ball-in*′ *mtopology-openin-iff open-ball-subset-ofS openin-Int-closure-of-eq-empty*)
thus *x* ∈ *mtopology closure-of A*
using *mtopology-closedin-iff2*[*of mtopology closure-of A*,*simplified*]
by *auto*
qed

lemma *closure-of-mtopology*′:
mtopology closure-of A = {*a*. ∃ *an*∈*UNIV* → *A*. *converge-to-inS an a*}

proof *safe*
fix *a*
assume *a* ∈ *mtopology closure-of A*
then have ∀ *ε*>*0*. *open-ball a ε* ∩ *A* ≠ {}
by(*simp add*: *closure-of-mtopology*)
hence ⋀*n*. ∃ *an*. *an* ∈ *open-ball a* (*1*/*real* (*Suc n*)) ∩ *A*
by (*meson all-not-in-conv divide-pos-pos of-nat-0-less-iff zero-less-Suc zero-less-one*)
then obtain *an* **where** *han*:⋀*n*. *an n* ∈ *open-ball a* (*1*/*real* (*Suc n*)) ∩ *A* **by** *metis*
hence *an* ∈ *UNIV* → *A* **by** *auto*
show ∃ *an*∈*UNIV* → *A*. *converge-to-inS an a*
proof(*safe intro*!: *bexI*[**where** *x*=*an*] ‹*an* ∈ *UNIV* → *A*›)
show *converge-to-inS an a*
unfolding *converge-to-inS-def2'*
proof *safe*
show *an n* ∈ *S a* ∈ *S* **for** *n*
using *open-ballD'*(*2*)[*of an 0 a*] *open-ballD'*(*1*)[*of an n*] *han* **by** *auto*
next
fix *ε*
assume (*0* :: *real*) < *ε*
then obtain *N* **where** *1* / *real* (*Suc N*) ≤ *ε*
by (*meson less-eq-real-def nat-approx-posE*)
show ∃ *N*. ∀ *n*≥*N*. *an n* ∈ *open-ball a ε*
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *N* ≤ *n*
then have *1* / *real* (*Suc n*) ≤ *1* / *real* (*Suc N*)
by (*simp add*: *frac-le*)
from *open-ball-le*[*OF order-trans*[*OF this* ‹*1* / *real* (*Suc N*) ≤ *ε*›]]
show *an n* ∈ *open-ball a ε*
using *han* **by** *auto*
qed
qed
qed
next
fix *a an*
assume *h*:*an* ∈ *UNIV* → *A converge-to-inS an a*
have ∀ *ε*>*0*. *open-ball a ε* ∩ *A* ≠ {}
proof *safe*
fix *ε*
assume (*0* :: *real*) < *ε open-ball a ε* ∩ *A* = {}
then obtain *N* **where** *an N* ∈ *open-ball a ε*
using *h*(*2*) *converge-to-inS-def2'* **by** *blast*
with ‹*open-ball a ε* ∩ *A* = {}› *h*(*1*) **show** *False* **by** *auto*
qed
thus *a* ∈ *mtopology closure-of A*
by(*simp add*: *closure-of-mtopology*)
qed

lemma *closure-of-mtopology-an*:
assumes $a \in$ *mtopology closure-of A*
obtains *an* **where** *an*$\in$*UNIV* $\to$ *A converge-to-inS an a*
using *assms* **by**(*auto simp*: *closure-of-mtopology'*)

lemma *closure-of-open-ball*: *mtopology closure-of open-ball a* $\varepsilon \subseteq$ *closed-ball a* ε
by(*rule closure-of-minimal-eq*[*THEN iffD2*]) (*auto simp*: *open-ball-subset-ofS mtopology-topspace closedin-closed-ball open-ball-closed-ball*)

lemma *interior-of-closed-ball*: *open-ball a e* $\subseteq$ *mtopology interior-of closed-ball a e*
by(*auto simp*: *interior-of-maximal-eq openin-open-ball open-ball-closed-ball*)

lemma *derived-set-of-mtopology*:
mtopology derived-set-of A = {*a*. $\exists$ *an*$\in$*UNIV* $\to$ *A*. ($\forall$ *n*. *an n* $\neq$ *a*) $\wedge$ *converge-to-inS an a*}
proof *safe*
fix *a*
assume *a* $\in$ *mtopology derived-set-of A*
then have *h*:*a* $\in$ *S* $\bigwedge$*v*. *a* $\in$ *v* $\Longrightarrow$ *openin mtopology v* $\Longrightarrow$ $\exists$ *y*. *y* $\neq$ *a* $\wedge$ *y* $\in$ *v* $\wedge$ *y* $\in$ *A*
by(*auto simp*: *in-derived-set-of mtopology-topspace*)
hence *a* $\in$ *open-ball a* (*1 / real* (*Suc n*)) **for** *n*
by(*auto intro*!: *open-ball-ina*)
from *h*(*2*)[*OF this openin-open-ball*[*of a*]]
obtain *an* **where** *an*:$\bigwedge$*n*. *an n* $\neq$ *a* $\bigwedge$*n*. *an n* $\in$ *open-ball a* (*1 / real* (*Suc n*)) $\bigwedge$*n*. *an n* $\in$ *A*
by *metis*
show $\exists$ *an*$\in$*UNIV* $\to$ *A*. ($\forall$ *n*. *an n* $\neq$ *a*) $\wedge$ *converge-to-inS an a*
proof(*safe intro*!: *bexI*[**where** *x*=*an*] *an*(*1*))
show *converge-to-inS an a*
unfolding *converge-to-inS-def2'*
proof *safe*
show $\bigwedge$*x*. *an x* $\in$ *S*
using *an*(*2*) *open-ball-subset-ofS* **by** *auto*
next
fix ε
assume (*0* :: *real*) < ε
then obtain *N* **where** *hN*:*1 / real* (*Suc N*) < ε
using *nat-approx-posE* **by** *blast*
show $\exists$ *N*. $\forall$ *n*$\geq$*N*. *an n* $\in$ *open-ball a* ε
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *N* $\leq$ *n*
then have *1 / real* (*Suc n*) $\leq$ *1 / real* (*Suc N*)
by (*simp add*: *frac-le*)
from *order.strict-trans1*[*OF this hN*] *open-ball-le*[*of - ε a*] *an*(*2*)[*of n*]
show *an n* $\in$ *open-ball a* ε **by**(*auto simp*: *less-le*)
qed
qed(*use h* **in** *auto*)

qed(*use an* **in** *auto*)
next
fix *a an*
assume *h*:*an* ∈ *UNIV* → *A* ∀ *n*. *an n* ≠ *a converge-to-inS an a*
have ∃ *y*. *y* ≠ *a* ∧ *y* ∈ *v* ∧ *y* ∈ *A* **if** *a* ∈ *v openin mtopology v* **for** *v*
proof −
obtain *ε* **where** *he*:*ε* > *0 a* ∈ *open-ball a ε open-ball a ε* ⊆ *v*
by (*meson* ‹*a* ∈ *v*› ‹*openin mtopology v*› *converge-to-inS-def2 h*(*3*) *mtopology-openin-iff open-ball-ina*)
then obtain *N* **where** *hn*:⋀*n*. *n* ≥ *N* ⟹ *an n* ∈ *open-ball a ε*
using *h*(*3*) **by**(*fastforce simp*: *converge-to-inS-def2* ′)
show ∃ *y*. *y* ≠ *a* ∧ *y* ∈ *v* ∧ *y* ∈ *A*
using *h*(*1*,*2*) *he hn* **by**(*auto intro*!: *exI*[**where** *x*=*an N*])
qed
thus *a* ∈ *mtopology derived-set-of A*
using *h*(*3*) **by**(*auto simp*: *in-derived-set-of converge-to-inS-def mtopology-topspace*)
qed

lemma *isolated-points-of-mtopology*:
mtopology isolated-points-of A = {*a*∈*S*∩*A*. ∀ *an*∈*UNIV* → *A*. *converge-to-inS an a* ⟶ (∃ *no*. ∀ *n*≥*no*. *an n* = *a*)}
proof *safe*
fix *a an*
assume *h*:*a* ∈ *mtopology isolated-points-of A converge-to-inS an a an* ∈ *UNIV* → *A*
then have *ha*:*a* ∈ *topspace mtopology a* ∈ *A* ∃ *U*. *a* ∈ *U* ∧ *openin mtopology U* ∧ *U* ∩ (*A* − {*a*}) = {}
by(*simp-all add*: *in-isolated-points-of*)
then obtain *U* **where** *u*:*a* ∈ *U openin mtopology U U* ∩ (*A* − {*a*}) = {}
by *auto*
then obtain *ε* **where** *e*: *ε* > *0 open-ball a ε* ⊆ *U*
by(*auto simp*: *mtopology-openin-iff*)
then obtain *N* **where** ⋀*n*. *n* ≥ *N* ⟹ *an n* ∈ *open-ball a ε*
using *h*(*2*) **by**(*fastforce simp*: *converge-to-inS-def2* ′)
thus ∃ *no*. ∀ *n*≥*no*. *an n* = *a*
using *h*(*3*) *e*(*2*) *u*(*3*) **by**(*auto intro*!: *exI*[**where** *x*=*N*])
qed (*auto simp*: *derived-set-of-mtopology isolated-points-of-def mtopology-topspace*)

lemma *perfect-set-open-ball-infinite*:
assumes *perfect-set mtopology A*
shows *closedin mtopology A* ∧ (∀ *a*∈*A*. ∀ *ε*>*0*. *infinite* (*open-ball a ε*))
proof *safe*
fix *a ε*
assume *h*: *a* ∈ *A* *0* < *ε finite* (*open-ball a ε*)
then have *a* ∈ *S*
using *open-ball-ina*[*OF* - ‹*0* < *ε*›,*of a*] *perfect-setD*(*2*)[*OF assms*]
by(*auto simp*: *mtopology-topspace*)
have ∃ *e* > *0*. *open-ball a e* = {*a*}
proof −

```
    consider open-ball a ε = {a} | {a} ⊂ open-ball a ε
      using open-ball-ina[OF ‹a ∈ S› h(2)] by blast
    thus ?thesis
    proof cases
      case 1
      with h(2) show ?thesis by auto
    next
      case 2
      then have nen:{dist a b |b. b ∈ open-ball a ε ∧ a ≠ b} ≠ {}
        by auto
      have fin: finite {dist a b |b. b ∈ open-ball a ε ∧ a ≠ b}
        using h(3) by auto
      define e where e ≡ Min {dist a b |b. b ∈ open-ball a ε ∧ a ≠ b}
      have e > 0
        using dist-0[OF ‹a ∈ S› open-ballD'(1)[of - a ε]] dist-geq0[of a]
        by(auto simp: e-def Min-gr-iff[OF fin nen] order-neq-le-trans)
      have bd:⋀b. b ∈ open-ball a ε ⟹ a ≠ b ⟹ e ≤ dist a b
        by(auto simp: e-def Min-le-iff[OF fin nen])
      have e ≤ ε
        using nen open-ballD[of - a ε]
        by(fastforce simp add: e-def Min-le-iff[OF fin nen])
      show ?thesis
      proof(safe intro!: exI[where x=e])
        fix x
        assume x:x ∈ open-ball a e
        then show x = a
          using open-ball-le[OF ‹e ≤ ε›,of a] open-ballD[OF x] bd[of x]
          by auto
      qed (simp-all add: open-ball-ina[OF ‹a ∈ S› ‹e > 0›] ‹0 < e›)
    qed
  qed
  then obtain e where e:e > 0 open-ball a e = {a} by auto
  show False
    using perfect-setD(3)[OF assms h(1) open-ball-ina[OF ‹a ∈ S› ‹e > 0›]]
    by(auto simp: openin-open-ball) (use e(2) in auto)
qed(use perfect-setD[OF assms] in simp)

lemma nbh-subset:
  assumes A: A ⊆ S and e: e > 0
  shows A ⊆ (⋃a∈A. open-ball a e)
  using A open-ball-ina[OF - e] by auto

lemma nbh-decseq:
  assumes decseq an
  shows decseq (λn. ⋃a∈A. open-ball a (an n))
proof(safe intro!: decseq-SucI)
  fix n a b
  assume a ∈ A b ∈ open-ball a (an (Suc n))
  with open-ball-le[OF decseq-SucD[OF assms]] show b ∈ (⋃c∈A. open-ball c (an
```

```
n))
    by(auto intro!: bexI[where x=a] simp: frac-le)
qed

lemma nbh-Int:
  assumes A: A ≠ {} A ⊆ S
      and an:⋀n. an n > 0 decseq an an ⟶ 0
  shows (⋂ n. ⋃ a∈A. open-ball a (an n)) = mtopology closure-of A
proof safe
  fix x
  assume x ∈ (⋂ n. ⋃ a∈A. open-ball a (an n))
  then have h:∀ n. ∃ a∈A. x ∈ open-ball a (an n)
    by auto
  hence x:x ∈ S
    using open-ball-subset-ofS by auto
  show x ∈ mtopology closure-of A
    unfolding closure-of-mtopology
  proof safe
    fix e :: real
    assume h':e > 0 open-ball x e ∩ A = {}
    then obtain n where n:an n < e
      using an(1,3) by(auto simp: LIMSEQ-def abs-of-pos) (metis dual-order.refl)
    from h obtain a where a ∈ A x ∈ open-ball a (an n)
      by auto
    with h'(2) open-ball-le[of an n e x] n
    show False
      by(auto simp: open-ball-inverse[of x])
  qed
next
  fix x n
  assume x ∈ mtopology closure-of A
  with an(1) have open-ball x (an n) ∩ A ≠ {}
    by(auto simp: closure-of-mtopology)
  thus x ∈ (⋃ a∈A. open-ball a (an n))
    by(auto simp: open-ball-inverse[of x])
qed

lemma nbh-add: (⋃ b∈(⋃ a∈A. open-ball a e). open-ball b f) ⊆ (⋃ a∈A. open-ball
a (e + f))
proof safe
  fix a x b
  assume h:a ∈ A b ∈ open-ball a e x ∈ open-ball b f
  show x ∈ (⋃ a∈A. open-ball a (e + f))
  proof(rule UN-I[OF h(1)])
    have dist a x ≤ dist a b + dist b x
      by(auto intro!: dist-tr open-ballD'(2)[OF h(2)] open-ballD'[OF h(3)])
    also have ... < e + f
      using  open-ballD[OF h(2)] open-ballD[OF h(3)] by auto
    finally show x ∈ open-ball a (e + f)
```

using *open-ballD'*[*OF h*(*2*)] *open-ballD'*[*OF h*(*3*)]
by(*auto simp*: *open-ball-def*)
qed
qed

definition *convergent-inS* :: (*nat* ⇒ *'a*) ⇒ *bool* **where**
convergent-inS f ≡ ∃ *s*. *converge-to-inS f s*

lemma *convergent-inS-const*:
assumes *x* ∈ *S*
shows *convergent-inS* (λ*n*. *x*)
using *converge-to-inS-const*[*OF assms*] **by**(*auto simp*: *convergent-inS-def*)

lemma *convergent-inS-ignore-initial*:
assumes *convergent-inS xn*
shows *convergent-inS* (λ*n*. *xn* (*n* + *k*))
using *converge-to-inS-ignore-initial*[*of xn*] *assms*
by(*auto simp*: *convergent-inS-def*)

lemma *convergent-inS-offset*:
assumes *convergent-inS* (λ*n*. *xn* (*n* + *k*)) *xn* ∈ *sequence*
shows *convergent-inS xn*
using *converge-to-inS-offset*[*of xn k*] *assms*
by(*auto simp*: *convergent-inS-def*)

definition *the-limit-of* :: (*nat* ⇒ *'a*) ⇒ *'a* **where**
the-limit-of xn ≡ *THE x*. *converge-to-inS xn x*

lemma *the-limit-if-converge*:
assumes *convergent-inS xn*
shows *converge-to-inS xn* (*the-limit-of xn*)
unfolding *the-limit-of-def*
by(*rule theI'*) (*auto simp*: *assms*[*simplified convergent-inS-def*] *converge-to-inS-unique*)

lemma *the-limit-of-eq*:
assumes *converge-to-inS xn x*
shows *the-limit-of xn* = *x*
using *assms converge-to-inS-unique the-limit-of-def* **by** *auto*

lemma *the-limit-of-inS*:
assumes *convergent-inS xn*
shows *the-limit-of xn* ∈ *S*
using *the-limit-if-converge*[*OF assms*] **by**(*simp add*:*converge-to-inS-def*)

lemma *the-limit-of-const*:
assumes *x* ∈ *S*
shows *the-limit-of* (λ*n*. *x*) = *x*
by(*rule the-limit-of-eq*[*OF converge-to-inS-const*[*OF assms*]])

lemma *convergent-inS-dest1*:
assumes *convergent-inS f*
shows *f n* ∈ *S*
using *assms* **by**(*auto simp*: *convergent-inS-def converge-to-inS-def2*)

definition *Cauchy-inS*:: (*nat* ⇒ *'a*) ⇒ *bool* **where**
Cauchy-inS f ≡ *f* ∈ *sequence* ∧ (∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. ∀ *m*≥*N*. *dist* (*f n*) (*f m*) < *ε*)

lemma *Cauchy-inS-def2*:
Cauchy-inS f ⟷ *f* ∈ *sequence* ∧ (∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball* (*f N*) *ε*)
unfolding *Cauchy-inS-def*
proof *safe*
fix *ε* :: *real*
assume *h*:*f* ∈ *sequence* ∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. ∀ *m*≥*N*. *dist* (*f n*) (*f m*) < *ε* *0* < *ε*
then obtain *N* **where** *hn*:
⋀*n m*. *n* ≥ *N* ⟹ *m*≥*N* ⟹ *dist* (*f n*) (*f m*) < *ε*
by *fastforce*
show ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball* (*f N*) *ε*
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *N* ≤ *n*
then show *f n* ∈ *open-ball* (*f N*) *ε*
using *h*(*1*) *hn*[*of N n*] **by**(*auto simp*: *open-ball-def*)
qed
next
fix *ε* :: *real*
assume *h*:*f* ∈ *sequence* ∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball* (*f N*) *ε* *0* < *ε*
then obtain *N* **where** *hn*:
⋀*n*. *n* ≥ *N* ⟹ *f n* ∈ *open-ball* (*f N*) (*ε*/*2*)
using *linordered-field-class.half-gt-zero*[*OF h*(*3*)] **by** *blast*
show ∃ *N*. ∀ *n*≥*N*. ∀ *m*≥*N*. *dist* (*f n*) (*f m*) < *ε*
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n m*
assume *N* ≤ *n* *N* ≤ *m*
from *order.strict-trans1*[*OF dist-tr* [*of f n f N f m*] *strict-ordered-ab-semigroup-add-class.add-strict-mono*[*OF open-ballD*[*OF hn*[*OF this*(*1*)],*simplified dist-sym*[*of - f n*]] *open-ballD*[*OF hn*[*OF this*(*2*)]],*simplified*]]
show *dist* (*f n*) (*f m*) < *ε*
using *h*(*1*) **by** *auto*
qed
qed

lemma *Cauchy-inS-def2'*:
Cauchy-inS f ⟷ *f* ∈ *sequence* ∧ (∀ *ε*>*0*. ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball x* *ε*)
unfolding *Cauchy-inS-def2*
proof *safe*
fix *ε* :: *real*

assume *h*:*f* ∈ *sequence* ∀ε>*0*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball* (*f N*) ε *0* < ε
then obtain *N* **where** ∀ *n*≥*N*. *f n* ∈ *open-ball* (*f N*) ε **by** *auto*
thus ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball x* ε
using *h*(*1*) **by**(*auto intro*!: *exI*[**where** *x*=*N*] *bexI*[**where** *x*=*f N*])
next
fix ε :: *real*
assume *h*:*f* ∈ *sequence* ∀ε>*0*. ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball x* ε *0* < ε
then obtain *x N* **where** *hxn*:
x ∈ *S* ⋀*n*. *n* ≥ *N* ⟹ *f n* ∈ *open-ball x* (ε/*2*)
using *linordered-field-class.half-gt-zero*[*OF h*(*3*)] **by** *blast*
show ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball* (*f N*) ε
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *N* ≤ *n*
from *order.strict-trans1*[*OF dist-tr strict-ordered-ab-semigroup-add-class.add-strict-mono*[*OF open-ballD*[*OF hxn*(*2*)[*OF order.refl*],*simplified dist-sym*[*of x*]] *open-ballD*[*OF hxn*(*2*)[*OF this*]],*simplified*]]
show *f n* ∈ *open-ball* (*f N*) ε
using *hxn*(*1*) *h*(*1*) **by**(*auto simp*: *open-ball-def*)
qed
qed

lemma *Cauchy-inS-def2''*:
Cauchy-inS f ⟷ *f* ∈ *sequence* ∧ (∀ε>*0*. ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *dist x* (*f n*) < ε)
unfolding *Cauchy-inS-def2'*
proof *safe*
fix ε :: *real*
assume *h*:*f* ∈ *sequence* ∀ε>*0*. ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball x* ε *0* < ε
then obtain *x N* **where**
x ∈ *S* ⋀*n*. *n* ≥ *N* ⟹ *f n* ∈ *open-ball x* ε
by *blast*
then show ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *dist x* (*f n*) < ε
by(*auto intro*!: *bexI*[**where** *x*=*x*] *exI*[**where** *x*=*N*] *simp*: *open-ballD*[*of* - *x* ε])
next
fix ε :: *real*
assume *h*:*f* ∈ *sequence* ∀ε>*0*. ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *dist x* (*f n*) < ε *0* < ε
then obtain *x N* **where**
x ∈ *S* ⋀*n*. *n* ≥ *N* ⟹ *dist x* (*f n*) < ε **by** *blast*
then show ∃ *x*∈*S*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *open-ball x* ε
using *h*(*1*) **by**(*auto intro*!: *bexI*[**where** *x*=*x*] *exI*[**where** *x*=*N*] *simp*: *open-ball-def*)
qed

lemma *Cauchy-inS-dest1*:
assumes *Cauchy-inS f*
shows *f n* ∈ *S*
using *assms* **by**(*auto simp*: *Cauchy-inS-def*)

lemma *Cauchy-if-convergent-inS*:
assumes *convergent-inS f*

```
  shows Cauchy-inS f
  unfolding Cauchy-inS-def
proof safe
  fix ε :: real
  assume h:0 < ε
  obtain s where hs:
   s ∈ S ∀ε>0. ∃ N. ∀ n≥N. dist (f n) s < ε
    using assms by(auto simp: convergent-inS-def converge-to-inS-def2)
  then obtain N where hn:
    ⋀n. n≥N ⟹ dist (f n) s < ε/2
    using half-gt-zero[OF h] by blast
  show ∃ N. ∀ n≥N. ∀ m≥N. dist (f n) (f m) < ε
  proof(safe intro!: exI[where x=N])
    fix n m
    assume hnm:N ≤ n N ≤ m
    have dist (f n) (f m) ≤ dist (f n) s + dist s (f m)
      using convergent-inS-dest1[OF assms] hs
      by(auto intro!: dist-tr)
    also have ... = dist (f n) s + dist (f m) s
      by(simp add: dist-sym[of s])
    also have ... < ε
      using hn[OF hnm(1)] hn[OF hnm(2)] by auto
    finally show dist (f n) (f m) < ε .
  qed
next
  show ⋀x. f x ∈ S
    using assms[simplified convergent-inS-def converge-to-inS-def]
    by auto
qed

corollary Cauchy-inS-const: a ∈ S ⟹ Cauchy-inS (λn. a)
  by(auto intro!: Cauchy-if-convergent-inS convergent-inS-const)

lemma converge-if-Cauchy-and-subconverge:
  assumes strict-mono a converge-to-inS (f ∘ a) s Cauchy-inS f
  shows converge-to-inS f s
  unfolding  converge-to-inS-def2
proof safe
  fix ε
  assume (0 :: real) < ε
  then have 1:0 < ε/2 by auto
  then obtain N where hn:⋀n. n ≥ N ⟹ dist (f (a n)) s < ε/2
    using assms(2) by(simp only: comp-def converge-to-inS-def2) metis
  obtain N' where hn':⋀n m. n ≥ N' ⟹ m ≥ N' ⟹ dist (f n) (f m) < ε/2
    using assms(3) 1 by(simp only: Cauchy-inS-def) metis
  show ∃ N. ∀ n≥N. dist (f n) s < ε
  proof(safe intro!: exI[where x=max N N'])
    fix n
    assume max N N' ≤ n
```

then have $N \leq n$ $N' \leq n$ **by** *auto*
show *dist* $(f\ n)$ $s < \varepsilon$
using *add-strict-mono*[*OF hn'*[*OF* ‹$N' \leq n$› *order-trans*[*OF* ‹$N' \leq n$› *seq-suble*[*OF assms*(*1*),*of n*]]] *hn*[*OF* ‹$N \leq n$›]] *assms*(*2*)
by(*auto simp*: *converge-to-inS-def intro*!: *order.strict-trans1*[*OF dist-tr*[*OF Cauchy-inS-dest1*[*OF assms*(*3*),*of n*] *Cauchy-inS-dest1*[*OF assms*(*3*),*of a n*],*of s*],*of* ε])
qed
qed(*auto simp*: *Cauchy-inS-dest1*[*OF assms*(*3*)] *assms*(*2*)[*simplified converge-to-inS-def*])

lemma *subCauchy-Cahcuy*:
assumes *Cauchy-inS xn strict-mono a*
shows *Cauchy-inS* $(xn \circ a)$
unfolding *Cauchy-inS-def*
proof *safe*
show $\bigwedge x.\ (xn \circ a)\ x \in S$
using *assms*(*1*) **by**(*simp add*: *Cauchy-inS-dest1*)
next
fix ε
assume $(0 :: real) < \varepsilon$
then obtain N **where** $\bigwedge n\ m.\ n \geq N \Longrightarrow m \geq N \Longrightarrow dist\ (xn\ n)\ (xn\ m) < \varepsilon$
using *assms*(*1*) **by**(*auto simp*: *Cauchy-inS-def*) *metis*
thus $\exists N.\ \forall n{\geq}N.\ \forall m{\geq}N.\ dist\ ((xn \circ a)\ n)\ ((xn \circ a)\ m) < \varepsilon$
by(*auto intro*!: *exI*[**where** *x*=*N*] *dest*: *order-trans*[*OF seq-suble*[*OF assms*(*2*)] *strict-mono-leD*[*OF assms*(*2*)]])
qed

corollary *Cauchy-inS-ignore-initial*:
assumes *Cauchy-inS xn*
shows *Cauchy-inS* $(\lambda n.\ xn\ (n + k))$
using *subCauchy-Cahcuy*[*OF assms*,*of* $\lambda n.\ n + k$]
by(*auto simp*: *comp-def strict-monoI*)

lemma *Cauchy-inS-dist-Cauchy*:
assumes *Cauchy-inS xn Cauchy-inS yn*
shows *Cauchy* $(\lambda n.\ dist\ (xn\ n)\ (yn\ n))$
unfolding *metric-space-class.Cauchy-altdef2 dist-real-def*
proof *safe*
have *h*:$\bigwedge n.\ xn\ n \in S\ \bigwedge n.\ yn\ n \in S$
using *assms* **by**(*auto simp*: *Cauchy-inS-dest1*)
fix *e* :: *real*
assume *e*:$0 < e$
with *assms* **obtain** *N1 N2* **where** *N*: $\bigwedge n\ m.\ n \geq N1 \Longrightarrow m \geq N1 \Longrightarrow dist\ (xn\ n)\ (xn\ m) < e\ /\ 2\ \bigwedge n\ m.\ n \geq N2 \Longrightarrow m \geq N2 \Longrightarrow dist\ (yn\ n)\ (yn\ m) < e\ /\ 2$
by (*metis Cauchy-inS-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 $\exists N.\ \forall n{\geq}N.\ |dist\ (xn\ n)\ (yn\ n) - dist\ (xn\ N)\ (yn\ N)| < e$
proof(*safe intro*!: *exI*[**where** $x{=}N$])
fix n
assume $n{:}N \leq n$
have $dist\ (xn\ n)\ (yn\ n) \leq dist\ (xn\ n)\ (xn\ N) + dist\ (xn\ N)\ (yn\ N) + dist\ (yn\ N)\ (yn\ n)$
$dist\ (xn\ N)\ (yn\ N) \leq dist\ (xn\ N)\ (xn\ n) + dist\ (xn\ n)\ (yn\ n) + dist\ (yn\ n)\ (yn\ N)$
using *dist-tr*[*OF h*(1)[*of n*] *h*(1)[*of N*] *h*(2)[*of n*]] *dist-tr*[*OF h*(1)[*of N*] *h*(2)[*of N*] *h*(2)[*of n*]]
dist-tr[*OF h*(1)[*of N*] *h*(2)[*of n*] *h*(2)[*of N*]] *dist-tr*[*OF h*(1)[*of N*] *h*(1)[*of n*] *h*(2)[*of n*]] **by** *auto*
thus $|dist\ (xn\ n)\ (yn\ n) - dist\ (xn\ N)\ (yn\ N)| < e$
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

corollary *Cauchy-inS-dist-convergent*:
assumes *Cauchy-inS xn Cauchy-inS yn*
shows *convergent* $(\lambda n.\ dist\ (xn\ n)\ (yn\ n))$
using *Cauchy-inS-dist-Cauchy*[*OF assms*] *Cauchy-convergent-iff* **by** *blast*

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

abbreviation *dense-set* ≡ *dense-of mtopology*

lemma *dense-set-def*:
dense-set $U \longleftrightarrow U \subseteq S \wedge (\forall x{\in}S.\ \forall \varepsilon{>}0.\ open\text{-}ball\ x\ \varepsilon \cap U \neq \{\})$
proof
assume h: $U \subseteq S \wedge (\forall x{\in}S.\ \forall \varepsilon{>}0.\ open\text{-}ball\ x\ \varepsilon \cap U \neq \{\})$
show *dense-of mtopology U*
proof(*rule dense-ofI*)
fix V
assume h':*openin mtopology V* $V \neq \{\}$
then obtain x **where** *1*:$x \in V$ **by** *auto*
then obtain ε **where** *2*:$\varepsilon{>}0$ *open-ball x* $\varepsilon \subseteq V$
using h' *mtopology-openin-iff*[*of V*] **by** *blast*
have *open-ball x* $\varepsilon \cap U \neq \{\}$
using *h 1 2 openin-subset*[*OF* $h'(1)$, *simplified mtopology-topspace*]
by *auto*
thus $U \cap V \neq \{\}$ **using** *2* **by** *auto*
next
show $U \subseteq$ *topspace mtopology*
using *h mtopology-topspace* **by** *auto*
qed
next
assume *h*:*dense-of mtopology U*

```
  have ∀ x∈S. ∀ ε>0. open-ball x ε ∩ U ≠ {}
  proof safe
    fix x ε
    assume x ∈ S (0 :: real) < ε open-ball x ε ∩ U = {}
    then have open-ball x ε ≠ {} openin mtopology (open-ball x ε)
      using open-ball-ina[of x ε] mtopology-open-ball-in[of x ε]
      by blast+
    thus False
      using h ‹open-ball x ε ∩ U = {}› by(auto simp: dense-of-def)
  qed
  thus U ⊆ S ∧ (∀ x∈S. ∀ ε>0. open-ball x ε ∩ U ≠ {})
    using h mtopology-topspace by(auto simp: dense-of-def)
qed

corollary dense-set-balls-cover:
  assumes dense-set U and e > 0
  shows (⋃ u∈U. open-ball u e) = S
  using assms open-ball-subset-ofS by(auto simp: dense-set-def) (meson Int-emptyI
open-ball-inverse)

lemma dense-set-empty-iff: dense-set {} ⟷ S = {}
  by(auto simp: dense-set-def ) (use zero-less-one in blast)

lemma dense-set-S: dense-set S
  using open-ball-ina dense-set-def by blast

lemma dense-set-def2:
 dense-set U ⟷ U ⊆ S ∧ (∀ x∈S. ∀ ε>0.∃ y∈U. dist x y < ε)
proof
  assume h: dense-set U
  show U ⊆ S ∧ (∀ x∈S. ∀ ε>0. ∃ y∈U. dist x y < ε)
  proof safe
    fix x ε
    assume hxe: x ∈ S (0 :: real) < ε
    then obtain z where
      z ∈ open-ball x ε ∩ U
      using h by(fastforce simp: dense-set-def)
    thus ∃ y∈U. dist x y < ε
      by(auto intro!: bexI[where x=z] simp: open-ball-def hxe)
  qed(use h[simplified dense-set-def] in auto)
next
  assume h:U ⊆ S ∧ (∀ x∈S. ∀ ε>0. ∃ y∈U. dist x y < ε)
  show dense-set U
    unfolding dense-set-def
  proof safe
    fix x ε
    assume hxe: x ∈ S (0 :: real) < ε open-ball x ε ∩ U = {}
    then obtain y where
      y ∈ U y ∈ S dist x y < ε
```

```
    using h by blast
  hence y ∈ open-ball x ε ∩ U
    by(auto simp: open-ball-def hxe)
  thus False
    using hxe(3) by auto
 qed(use h in auto)
qed

lemma dense-set-def2':
 dense-set U ⟷ U ⊆ S ∧ (∀x∈S. ∃f∈UNIV → U. converge-to-inS f x)
 unfolding dense-set-def
proof
 show U ⊆ S ∧ (∀x∈S. ∀ε>0. open-ball x ε ∩ U ≠ {}) ⟹ U ⊆ S ∧ (∀x∈S.
∃f∈UNIV → U. converge-to-inS f x)
 proof safe
  fix x
  assume h: U ⊆ S ∀x∈S. ∀ε>0. open-ball x ε ∩ U ≠ {} x ∈ S
  then have ⋀n::nat. open-ball x (1 / (real n + 1)) ∩ U ≠ {}
    by auto
  hence ∀n. ∃k. k ∈ open-ball x (1 / (real n + 1)) ∩ U by auto
  hence ∃a. ∀n. a n ∈ open-ball x (1 / (real n + 1)) ∩ U by(rule choice)
  then obtain a where hf: ⋀n :: nat. a n ∈ open-ball x (1 / (real n + 1)) ∩ U
    by auto
  show ∃f∈UNIV → U. converge-to-inS f x
    unfolding converge-to-inS-def2'
  proof(safe intro!: bexI[where x=a])
    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. a n ∈ open-ball 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 a n ∈ open-ball x ε
       using open-ball-le[OF 1,of x] open-ball-le[OF order.strict-implies-order[OF
hn''],of x] hf[of n]
        by auto
    qed
  next
```

```
      show ⋀x. a x ∈ S x ∈ S ⋀x. a x ∈ U
        using h(1,3) hf by auto
    qed
  qed
next
  assume h:U ⊆ S ∧ (∀x∈S. ∃f∈UNIV → U. converge-to-inS f x)
  have ∀x∈S. ∀ε>0. open-ball x ε ∩ U ≠ {}
  proof safe
    fix x ε
    assume hxe:x ∈ S (0 :: real) < ε open-ball x ε ∩ U = {}
    then obtain f N where
     f∈UNIV → U ∀n≥N :: nat. f n ∈ open-ball x ε
      using h[simplified converge-to-inS-def2 ] by blast
    hence f N ∈ open-ball x ε ∩ U
      by auto
    thus False using hxe by auto
  qed
  thus U ⊆ S ∧ (∀x∈S. ∀ε>0. open-ball x ε ∩ U ≠ {})
    using h by auto
qed

lemma dense-set-infinite:
  assumes infinite S dense-set U
  shows infinite U
proof
  assume finu:finite U
  with assms(1) obtain x where x:x ∈ S x ∉ U
    by (meson finite-subset subset-iff)
  define e where e ≡ Min {dist x y |y. y ∈ U}
  have nen: {dist x y |y. y ∈ U} ≠ {}
    using dense-set-empty-iff assms by auto
  have fin: finite {dist x y |y. y ∈ U}
    using finu by auto
  have epos: 0 < e
    unfolding Min-gr-iff[OF fin nen] e-def
  proof safe
    fix y
    assume y ∈ U
    then have y ∈ S x ≠ y
      using assms(2) x(2) by(auto simp: dense-set-def)
    thus 0 < dist x y
      using dist-0[OF x(1),of y] dist-geq0[of x y] by auto
  qed
  then obtain y where y:y∈U dist x y < e
    using assms(2) x(1) by(fastforce simp: dense-set-def2)
  thus False
    using Min-le[OF fin,of dist x y] by(auto simp: e-def)
qed
```

lemma *mtopology-Hausdorff*: *Hausdorff-space mtopology*
unfolding *Hausdorff-space-def*
proof *safe*
fix *x y*
assume *x* ∈ *topspace mtopology y* ∈ *topspace mtopology x* ≠ *y*
then have [*simp*]:*x* ∈ *S y* ∈ *S*
using *mtopology-topspace* **by** *auto*
with ‹*x* ≠ *y*› **have** *1*:*dist x y* > *0*
using *dist-0*[*of x y*] *dist-geq0*[*of x y*] **by** *auto*
show ∃ *U V*. *openin mtopology U* ∧ *openin mtopology V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *disjnt U V*
proof(*rule exI*[**where** *x*=*open-ball x* (*dist x y*/*2*)])
show ∃ *V*. *openin mtopology* (*open-ball x* (*dist x y* / *2*)) ∧ *openin mtopology V* ∧ *x* ∈ *open-ball x* (*dist x y* / *2*) ∧ *y* ∈ *V* ∧ *disjnt* (*open-ball x* (*dist x y* / *2*)) *V*
proof(*safe intro*!: *exI*[**where** *x*=*open-ball y* (*dist x y*/*2*)])
show *disjnt* (*open-ball x* (*dist x y* / *2*)) (*open-ball y* (*dist x y* / *2*))
unfolding *disjnt-iff*
proof *safe*
fix *z*
assume *h*:*z* ∈ *open-ball x* (*dist x y* / *2*) *z* ∈ *open-ball y* (*dist x y* / *2*)
show *False*
using *dist-tr*[*OF* ‹*x* ∈ *S*› *open-ballD'*(*1*)[*OF h*(*1*)] ‹*y* ∈ *S*›] *open-ballD*[*OF h*(*1*)] *open-ballD*[*OF h*(*2*)]
by (*simp add*: *dist-sym*)
qed
qed(*auto intro*!: *mtopology-open-ball-in 1 open-ball-ina*)
qed
qed

Diameter

definition *diam* :: *'a set* ⇒ *ennreal* **where**
diam A ≡ ⨆ {*ennreal* (*dist x y*) | *x y*. *x* ∈ *A* ∩ *S* ∧ *y* ∈ *A* ∩ *S*}

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

lemma *diam-def2*:
assumes *A* ⊆ *S*
shows *diam A* = ⨆ {*ennreal* (*dist x y*) | *x y*. *x* ∈ *A* ∧ *y* ∈ *A*}
using *assms* **by**(*auto simp*: *diam-def*) (*meson subset-eq*)

lemma *diam-subset*:
assumes *A* ⊆ *B*
shows *diam A* ≤ *diam B*
unfolding *diam-def* **using** *assms* **by**(*auto intro*!: *Sup-subset-mono*)

lemma *diam-cball-leq*: *diam* (*closed-ball a ε*) ≤ *ennreal* (*2* ∗ *ε*)
unfolding *Sup-le-iff diam-def*

proof *safe*
fix *x y*
assume *h*:*x* ∈ *closed-ball a ε y* ∈ *closed-ball a ε x* ∈ *S y* ∈ *S*
have *dist x y* ≤ *2* ∗ *ε*
using *dist-tr*[*OF h*(*3*) *closed-ballD*′(*2*)[*OF h*(*1*)] *h*(*4*)] *closed-ballD*[*OF h*(*1*),*simplified dist-sym*[*of a x*]] *closed-ballD*[*OF h*(*2*)]
by *auto*
thus *ennreal* (*dist x y*) ≤ *ennreal* (*2* ∗ *ε*)
using *dist-geq0*[*of x y*] *ennreal-leI*[*of - 2*∗*ε*] **by** *simp*
qed

lemma *diam-ball-leq*:
diam (*open-ball a ε*) ≤ *ennreal* (*2* ∗ *ε*)
using *diam-subset*[*OF open-ball-closed-ball*[*of a ε*]] *diam-cball-leq*[*of a ε*]
by *auto*

lemma *diam-is-sup*:
assumes *x* ∈ *A* ∩ *S y* ∈ *A* ∩ *S*
shows *dist x y* ≤ *diam A*
using *assms* **by**(*auto simp*: *diam-def intro*!:*Sup-upper*)

lemma *diam-is-sup*′:
assumes *x* ∈ *A* ∩ *S y* ∈ *A* ∩ *S diam A* ≤ *ennreal r r* ≥ *0*
shows *dist x y* ≤ *r*
using *order.trans*[*OF diam-is-sup*[*OF assms*(*1*,*2*)] *assms*(*3*)] *assms*(*4*) **by** *simp*

lemma *diam-le*:
assumes ⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *dist x y* ≤ *r*
shows *diam A* ≤ *r*
using *assms* **by**(*auto simp*: *diam-def Sup-le-iff ennreal-leI*)

lemma *diam-eq-closure*: *diam* (*mtopology closure-of A*) = *diam A*
proof(*rule antisym*)
show *diam A* ≤ *diam* (*mtopology closure-of A*)
by(*auto intro*!: *Sup-subset-mono simp*: *diam-def*) (*metis in-closure-of mtopology-topspace*)
next
have {*ennreal* (*dist x y*) |*x y*. *x* ∈ *A* ∩ *S* ∧ *y* ∈ *A* ∩ *S*} = *ennreal* ‘ {*dist x y* |*x y*. *x* ∈ *A* ∩ *S* ∧ *y* ∈ *A* ∩ *S*}
by *auto*
also have *diam* (*mtopology closure-of A*) ≤ ⨆ ...
unfolding *le-Sup-iff-less*
proof *safe*
fix *r*
assume *r* < *diam* (*mtopology closure-of A*)
then obtain *x y* **where** *xy*:*x* ∈ *mtopology closure-of A x* ∈ *S y* ∈ *mtopology closure-of A y* ∈ *S r* < *ennreal* (*dist x y*)
by(*auto simp*: *diam-def less-Sup-iff*)
hence *r* < ⊤

using *dual-order.strict-trans ennreal-less-top* **by** *blast*
define e **where** $e \equiv (dist\ x\ y - enn2real\ r)/2$
have $e > 0$
using $xy(5)$ ‹$r < \top$› **by**(*simp add*: *e-def*)
then obtain $x'\ y'$ **where** xy':$x' \in$ *open-ball x e* $x' \in A$ $y' \in$ *open-ball y e* $y' \in A$
using *xy* **by**(*fastforce simp*: *closure-of-mtopology*)
show $\exists\, i \in \{dist\ x\ y\ |x\ y.\ x \in A \cap S \wedge y \in A \cap S\}.\ r \leq ennreal\ i$
proof(*safe intro*!: *bexI*[**where** x=*dist* $x'\ y'$])
have *dist x y* $\leq$ *dist x* x' + *dist* $x'\ y'$ + *dist y* y'
using *dist-tr*[*OF xy*(*2*) *open-ballD'*(*1*)[*OF xy'*(*1*)] *xy*(*4*)] *dist-tr*[*OF open-ballD'*(*1*)[*OF xy'*(*1*)] *open-ballD'*(*1*)[*OF xy'*(*3*)] *xy*(*4*)]
by(*simp add*: *dist-sym*)
also have ... < *dist x y* − *enn2real r* + *dist* $x'\ y'$
using *open-ballD*[*OF xy'*(*1*)] *open-ballD*[*OF xy'*(*3*)]
by(*simp add*: *e-def*)
finally have *enn2real r* < *dist* $x'\ y'$ **by** *simp*
thus $r \leq ennreal\ (dist\ x'\ y')$
by (*simp add*: ‹$r < \top$›)
qed(*use open-ballD'*(*1*)[*OF xy'*(*1*)] *open-ballD'*(*1*)[*OF xy'*(*3*)] *xy'*(*2*,*4*) **in** *auto*)
qed
finally show *diam* (*mtopology closure-of A*) ≤ *diam A*
by(*simp add*: *diam-def*)
qed

definition *bounded-set* :: *'a set* ⇒ *bool* **where**
bounded-set A ⟷ *diam A* < ∞

lemma *bounded-set-def2'*: *bounded-set A* ⟷ (∃ *e*. ∀ *x*∈*A*∩*S*. ∀ *y*∈*A*∩*S*. *dist x y* < *e*)
proof
assume *bounded-set A*
consider $A \cap S = \{\} \mid A \cap S \neq \{\}$ **by** *auto*
then show ∃ *e*. ∀ *x*∈*A* ∩ *S*. ∀ *y*∈*A* ∩ *S*. *dist x y* < *e*
proof *cases*
case *h*:*2*
then have *1*:{*dist x y* |*x y*. $x \in A \cap S \wedge y \in A \cap S$} ≠ {} **by** *auto*
have *eq*:{*ennreal* (*dist x y*) | *x y*. $x \in A \cap S \wedge y \in A \cap S$} = *ennreal* ‘ {*dist x y* | *x y*. $x \in A \cap S \wedge y \in A \cap S$}
by *auto*
hence *2*:*diam A* = ⨆ (*ennreal* ‘ {*dist x y* | *x y*. $x \in A \cap S \wedge y \in A \cap S$})
by(*simp add*: *diam-def*)
obtain *x y* **where** *hxy*:
$x \in A \cap S$ $y \in A \cap S$ *diam A* < *ennreal* (*dist x y*) + *ennreal 1*
using *SUP-approx-ennreal*[*OF - 1 2*,*of 1*] ‹*bounded-set A*›
by(*fastforce simp*: *bounded-set-def*)
hence *diam A* < *ennreal* (*dist x y* + *1*)
using *dist-geq0* **by** *simp*
from *SUP-lessD*[*OF this*[*simplified 2*]]
have ⋀*w z*. $w \in A \cap S \Longrightarrow z \in A \cap S \Longrightarrow$ *ennreal* (*dist w z*) < *ennreal* (*dist*

x y + 1)
    by *blast*
  thus ∃ *e*. ∀ *x*∈*A* ∩ *S*. ∀ *y*∈*A* ∩ *S*. *dist x y* < *e*
   by(*auto intro*!: *exI*[**where** *x*=*dist x y + 1*] *simp*: *ennreal-less-iff*[*OF dist-geq0*])
 qed *simp*
next
 assume ∃ *e*. ∀ *x*∈*A*∩*S*. ∀ *y*∈*A*∩*S*. *dist x y* < *e*
 then obtain *e* **where** *he*: ⋀*x y*. *x* ∈ *A* ∩ *S* ⟹ *y* ∈ *A* ∩ *S* ⟹ *dist x y* < *e*
  by *auto*
 hence ⋀*z*. *z* ∈ {*ennreal* (*dist x y*) | *x y*. *x* ∈ *A* ∩ *S* ∧ *y* ∈ *A* ∩ *S*} ⟹ *z* < *ennreal e*
  using *ennreal-less-iff*[*OF dist-geq0*] **by** *auto*
 hence ⨆ {*ennreal* (*dist x y*) | *x y*. *x* ∈ *A* ∩ *S* ∧ *y* ∈ *A* ∩ *S*} ≤ *ennreal e*
  by (*meson Sup-least order-less-le*)
 thus *bounded-set A*
  by(*simp add*: *bounded-set-def diam-def order-le-less-trans*[*OF* - *ennreal-less-top*])
qed

lemma *bounded-set-def2*:
 assumes *A* ⊆ *S*
 shows *bounded-set A* ⟷ (∃ *e*. ∀ *x*∈*A*. ∀ *y*∈*A*. *dist x y* < *e*)
 using *assms* **by**(*fastforce simp*: *bounded-set-def2'*)

lemma *bounded-set-def3'*:
 assumes *S* ≠ {}
 shows *bounded-set A* ⟷ (∃ *e*. ∃ *x*∈*S*. ∀ *y*∈*A*∩*S*. *dist x y* < *e*)
 unfolding *bounded-set-def2'*
proof
 assume *h*:∃ *e*. ∀ *x*∈*A* ∩ *S*. ∀ *y*∈*A* ∩ *S*. *dist x y* < *e*
 obtain *s* **where** [*simp*]:*s* ∈ *S* **using** *assms* **by** *auto*
 consider *A* ∩ *S* = {} | *A* ∩ *S* ≠ {} **by** *auto*
 then show ∃ *e*. ∃ *x*∈*S*. ∀ *y*∈*A* ∩ *S*. *dist x y* < *e*
 proof *cases*
  case *1*
  then show *?thesis*
   by(*auto intro*!: *exI*[**where** *x*=*0*] *exI*[**where** *x*=*s*])
 next
  case *2*
  then obtain *sa* **where** [*simp*]:*sa* ∈ *A sa* ∈ *S* **by** *auto*
  obtain *e* **where** ∀ *x*∈*A* ∩ *S*. ∀ *y*∈*A* ∩ *S*. *dist x y* < *e*
   using *h* **by** *auto*
  then show *?thesis*
   by(*auto intro*!: *exI*[**where** *x*=*e*] *bexI*[**where** *x*=*sa*])
 qed
next
 assume ∃ *e*. ∃ *x*∈*S*. ∀ *y*∈*A* ∩ *S*. *dist x y* < *e*
 then obtain *e a* **where**
  [*simp*]:*a* ∈ *S* **and** *hea*:⋀*y*. *y* ∈ *A* ⟹ *y* ∈ *S* ⟹ *dist a y* < *e* **by** *auto*
 show ∃ *e*. ∀ *x*∈*A* ∩ *S*. ∀ *y*∈*A* ∩ *S*. *dist x y* < *e*

```
  proof(safe intro!: exI[where x=2*e])
    fix x y
    assume [simp]:x ∈ A x ∈ S y ∈ A y ∈ S
    show dist x y < 2 * e
      using dist-tr[of x a y] hea[of x] hea[of y]
      by(simp add: dist-sym[of x a])
  qed
qed

lemma bounded-set-def4':
 bounded-set A ⟷ (∃ x e. A ∩ S ⊆ open-ball x e)
proof
  assume h:bounded-set A
  consider A ∩ S = {} | A ∩ S ≠ {} by auto
  then show ∃ x e. A ∩ S ⊆ open-ball x e
  proof cases
    case 1
    then show ?thesis by auto
  next
    case 2
    then have ∃ e. ∃ x∈S. ∀ y∈A∩S. dist x y < e
      using bounded-set-def3' h by blast
    then obtain e x where
     [simp]: x ∈ S and hex: ⋀y. y ∈ A ⟹ y ∈ S ⟹ dist x y < e
      by auto
    thus ?thesis
      by(auto intro!: exI[where x=x] exI[where x=e] simp:open-ball-def)
  qed
next
  assume ∃ x e. A ∩ S ⊆ open-ball x e
  then obtain a e where hxe:A ∩ S ⊆ open-ball a e by auto
  show bounded-set A
    unfolding bounded-set-def2'
  proof(safe intro!: exI[where x=2*e])
    fix x y
    assume [simp]:x ∈ A x ∈ S y ∈ A y ∈ S
    then have x ∈ open-ball a e y ∈ open-ball a e
      using hxe by auto
    hence dist a x < e dist a y < e a ∈ S
      by(auto dest: open-ballD open-ballD')
    thus dist x y < 2 * e
      using dist-tr[of x a y] by(simp add: dist-sym[of x a])
  qed
qed

lemma bounded-set-def4:
  assumes A ⊆ S
  shows bounded-set A ⟷ (∃ x e. A ⊆ open-ball x e)
  using bounded-set-def4'[of A] assms by blast
```

Distance between a point and a set.

definition *dist-set* :: *'a set* ⇒ *'a* ⇒ *real* **where**
dist-set A ≡ (λ*x*. *if A* = {} *then 0 else Inf* {*dist x y* |*y*. *y* ∈ *A*})

lemma *dist-set-geq0*:
dist-set A x ≥ *0*
proof −
have {*dist x y* |*y*. *y* ∈ *A*} = *dist x* ‘ *A* **by** *auto*
thus *?thesis*
using *dist-geq0*[*of x*] **by**(*auto simp*: *dist-set-def intro*!: *cINF-greatest*[*of - - dist x*])
qed

lemma *dist-set-bdd-below*[*simp*]:
bdd-below {*dist x y* |*y*. *y* ∈ *A*}
by(*auto simp*: *bdd-below-def dist-geq0 intro*!: *exI*[**where** *x=0*])

lemma *dist-set-singleton*[*simp*]:
dist-set {*y*} *x* = *dist x y*
by(*auto simp*: *dist-set-def*)

lemma *dist-set-singleton'*[*simp*]:
dist-set {*y*} = (λ*x*. *dist x y*)
by *auto*

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

lemma *dist-set-nsubset0*[*simp*]:
assumes ¬ (*A* ⊆ *S*)
shows *dist-set A x* = *0*
proof −
obtain *a* **where** *a* ∈ *A a* ∉ *S*
using *assms* **by** *auto*
hence *A* ≠ {} *0* ∈ {*dist x y* |*y*. *y* ∈ *A*}
using *dist-notin'*[*of a x*] **by** *auto*
thus *?thesis*
using ‹*A* ≠ {}› *dist-set-geq0*[*of A x*] *cInf-lower*[*OF* ‹*0* ∈ {*dist x y* |*y*. *y* ∈ *A*}›]
by(*auto simp*: *dist-set-def*)
qed

lemma *dist-set-notin*[*simp*]:
assumes *x* ∉ *S*
shows *dist-set A x* = *0*
proof −
have *A* ≠ {} ⟹ {*dist x y* |*y*. *y* ∈ *A*} = {*0*}
using *dist-notin*[*OF* ‹*x* ∉ *S*›] **by** *auto*
thus *?thesis*

by(*simp add*: *dist-set-def*)
qed

lemma *dist-set-inA*:
assumes $x \in A$
shows *dist-set* $A\ x = 0$
proof(*cases* $A \subseteq S$)
case *h*:*True*
hence $A \neq \{\}$ $0 \in \{dist\ x\ y\ |y.\ y \in A\}$
using *dist-0*[*of x x*] *assms* **by** *force*+
thus *?thesis*
using *cInf-lower*[*OF* ‹$0 \in \{dist\ x\ y\ |y.\ y \in A\}$›] *dist-set-geq0*[*of A x*]
by(*auto simp*: *dist-set-def*)
qed (*simp add*: *dist-geq0*)

lemma *dist-set-nzeroD*:
assumes *dist-set* $A\ x \neq 0$
shows $A \subseteq S$ $x \notin A$
by(*rule ccontr*, *use assms dist-set-inA* **in** *auto*)

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

lemma *dist-set-bounded*:
assumes $\bigwedge y.\ y \in A \Longrightarrow dist\ x\ y < K$ $K > 0$
shows *dist-set* $A\ x < K$
proof(*cases* $A = \{\}$)
case *True*
then show *?thesis*
by(*simp add*: *assms*)
next
case *1*:*False*
then have *2*:$\{dist\ x\ y\ |y.\ y \in A\} \neq \{\}$ **by** *auto*
show *?thesis*
using *assms* **by** (*auto simp add*: *dist-set-def cInf-lessD*[*OF 2*] *cInf-less-iff*[*OF 2*])
qed

lemma *dist-set-tr*:
assumes $x \in S$ $y \in S$

```
  shows dist-set A x ≤ dist x y + dist-set A y
proof(cases A ⊆ S)
  case h:True
  consider A = {} | A ≠ {} by auto
  then show ?thesis
  proof cases
    case 1
    then show ?thesis
      by(simp add: dist-set-def dist-geq0)
  next
    case 2
    have dist-set A x ≤ Inf {dist x y + dist y a |a. a∈A}
    proof −
      have ⨅ {dist x y |y. y ∈ A} ≤ ⨅ {dist x y + dist y a |a. a ∈ A}
      proof(rule cInf-mono)
        fix b
        assume b ∈ {dist x y + dist y a |a. a ∈ A}
        then obtain a where a ∈ A b = dist x y + dist y a
          by auto
        thus ∃ a∈{dist x y |y. y ∈ A}. a ≤ b
          using h assms by(auto intro!: exI[where x=dist x a] dist-tr)
      qed(simp-all add: 2)
      thus ?thesis
        by(simp add: dist-set-def 2)
    qed
    also have ... = dist x y + Inf {dist y a |a. a∈A}
    proof −
      have ereal (Inf {dist x y + dist y a |a. a∈A}) = ereal (dist x y + Inf {dist y
a |a. a∈A})
          (is ?lhs = ?rhs)
      proof −
        have ?lhs = Inf (ereal ‘ {(dist x y + dist y a) |a. a∈A})
          using dist-geq0 by(auto intro!: ereal-Inf′ bdd-belowI[where m=0] simp:
2)
        also have ... = Inf {ereal (dist x y + dist y a) |a. a∈A}
        proof −
          have ereal ‘ {(dist x y + dist y a) |a. a∈A} = {ereal (dist x y + dist y a)
|a. a∈A}
            by auto
          thus ?thesis by simp
        qed
        also have ... = (⨅ a∈A. ereal (dist x y) + ereal (dist y a))
          by (simp add: Setcompr-eq-image)
        also have ... = ereal (dist x y) + (⨅ a∈A. ereal (dist y a))
          by(rule INF-ereal-add-right) (use 2 dist-geq0 in auto)
        also have ... = ereal (dist x y) + (⨅ (ereal ‘ {dist y a | a. a ∈ A}))
          by (simp add: Setcompr-eq-image image-image)
        also have ... = ereal (dist x y) + ereal (Inf {dist y a |a. a∈A})
        proof −
```

```
        have ereal (Inf {dist y a |a. a∈A}) = (⨅ (ereal ` {dist y a | a. a ∈ A}))
          using dist-geq0 by(auto intro!: ereal-Inf' simp: 2)
        thus ?thesis by simp
      qed
      also have ... = ?rhs by simp
      finally show ?thesis .
    qed
    thus ?thesis by simp
  qed
  also have ... = dist x y + dist-set A y
    by(simp add: 2 dist-set-def)
  finally show ?thesis .
 qed
qed (simp add: dist-geq0)

lemma dist-set-abs-le:
 assumes x ∈ S y ∈ S
 shows |dist-set A x − dist-set A y| ≤ dist x y
 using dist-set-tr[OF assms,of A] dist-set-tr[OF assms(2,1),of A,simplified dist-sym[of
y x]]
 by auto

lemma dist-set-inA-le:
 assumes y ∈ A
 shows dist-set A x ≤ dist x y
proof −
 consider x ∉ S ∨ y ∉ S | x ∈ S ∧ y ∈ S by auto
 thus ?thesis
 proof cases
  case 1
  have y ∉ S ⟹ ¬ (A ⊆ S)
    using assms by auto
  with 1 dist-geq0 show ?thesis
    by auto
 next
  case 2
  with dist-set-tr[of x y A] dist-set-inA[OF assms]
  show ?thesis by simp
 qed
qed

lemma dist-set-ball-open:
 openin mtopology {x∈S. dist-set A x < ε}
 unfolding mtopology-openin-iff
proof safe
 fix x
 assume h:x ∈ S dist-set A x < ε
 show ∃ε'>0. open-ball x ε' ⊆ {x ∈ S. dist-set A x < ε}
 proof(safe intro!: exI[where x=ε − dist-set A x])
```

```
    fix y
    assume h':y ∈ open-ball x (ε − dist-set A x)
    have dist-set A y ≤ dist x y + dist-set A x
      by(rule dist-set-tr[OF open-ballD'(1)[OF h'] h(1),simplified dist-sym[of y x]])
    also have ... < ε
      using open-ballD[OF h'] by auto
    finally show dist-set A y < ε .
  qed(use h open-ballD'(1) in auto)
qed

lemma dist-set-ball-empty:
  assumes A ≠ {} A ⊆ S e > 0 x ∈ S open-ball x e ∩ A = {}
  shows dist-set A x ≥ e
 using assms by(auto simp: dist-set-def assms(1) le-cInf-iff intro!: open-ball-nin-le[OF
assms(4,3)])

lemma dist-set-closed-ge0:
  assumes closedin mtopology A A ≠ {} x ∈ S x ∉ A
  shows dist-set A x > 0
proof −
  have a:A ⊆ S openin mtopology (S − A)
    using closedin-subset[OF assms(1)] assms(1)
    by(auto simp: closedin-def mtopology-topspace)
  with assms(3,4) obtain e where e: e > 0 open-ball x e ⊆ S − A
    by(auto simp: mtopology-openin-iff) (meson Diff-iff)
  thus ?thesis
     by(auto intro!: order.strict-trans2[OF e(1) dist-set-ball-empty[OF assms(2)
a(1) e(1) assms(3)]])
qed

lemma g-delta-of-closed:
  assumes closedin mtopology M
  shows g-delta-of mtopology M
proof(cases M = {})
  case True
  then show ?thesis by simp
next
  case M-ne:False
  have M ⊆ S
    using assms mtopology-topspace by (simp add: closedin-def)
  define U where U ≡ (λn. {x∈S. dist-set M x < 1 / real n})
  define 𝒰 where 𝒰 ≡ {U n| n. n > 0}
  have mun:M ⊆ U n if n > 0 for n
    using dist-set-inA[of - M] that ‹M ⊆ S› by(auto simp: U-def)
  show ?thesis
  proof(rule g-delta-ofI[of 𝒰])
    show 𝒰 ≠ {}
      by(auto simp: 𝒰-def)
  next
```

```
    have 𝒰 = U ` {0<..} by(auto simp: 𝒰-def)
    thus countable 𝒰 by simp
  next
    fix b
    assume b ∈ 𝒰
    then show openin mtopology b
      using dist-set-ball-open by(auto simp: 𝒰-def U-def)
  next
    show M = ⋂ 𝒰
    proof(standard;standard)
      fix x
      assume x ∈ M
      with mun
      show x ∈ ⋂ 𝒰
        by(auto simp: 𝒰-def)
    next
      fix x
      assume x ∈ ⋂ 𝒰
      then have Inf {dist x m|m. m∈M} < 1 / real n if n > 0 for n
        using that by(auto simp: 𝒰-def U-def M-ne dist-set-def)
      hence 1:Inf {dist x m|m. m∈M} < 1 / real (Suc n) for n
        by blast
      have ∃ m∈M. dist x m < 1 / real (Suc n) for n
       using 1[of n] cInf-less-iff[of {dist x m |m. m ∈ M} 1 / real (Suc n)] M-ne
        by auto
      then obtain m where hm: ⋀n. m n ∈ M ⋀n. dist x (m n) < 1 / real (Suc
n)
        by metis
      hence m ∈ UNIV → M by auto
      have converge-to-inS m x
        unfolding converge-to-inS-def2
      proof safe
        show ⋀x. m x ∈ S x ∈ S
          using ‹x ∈ ⋂ 𝒰› ‹m ∈ UNIV → M› ‹M ⊆ S›
          by(auto simp: 𝒰-def U-def)
      next
        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. dist (m n) x < ε
        proof(safe intro!: exI[where x=N])
          fix n
          assume N ≤ n
          then have 1 / real (Suc n) ≤ 1 / real (Suc N)
            by (simp add: frac-le)
          from  order.strict-trans1[OF this hN] hm(2)[of n]
          show dist (m n) x < ε
            by(simp add: dist-sym[of x])
```

qed
qed
thus $x \in M$
using *assms*[*simplified mtopology-closedin-iff*] ‹$m \in UNIV \to M$›
by *simp*
qed
qed
qed

Oscillation

definition *osc-on* :: [$'b$ *set*, $'b$ *topology*, $'b \Rightarrow 'a$, $'b$] $\Rightarrow$ *ennreal* **where**
osc-on A X f $\equiv (\lambda y.\ \prod\ \{diam\ (f\ `\ (A \cap U))\ |U.\ y \in U \wedge openin\ X\ U\})$

abbreviation *osc X* $\equiv$ *osc-on* (*topspace X*) *X*

lemma *osc-def*: *osc X f* $= (\lambda y.\ \prod\ \{diam\ (f\ `\ U)\ |U.\ y \in U \wedge openin\ X\ U\})$
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 $< t \longleftrightarrow (\exists\, v.\ x \in v \wedge openin\ X\ v \wedge diam\ (f\ `\ (A \cap v)) < t)$
by(*auto simp add*: *osc-on-def Inf-less-iff*)

lemma *osc-less-iff*:
osc X f x $< t \longleftrightarrow (\exists\, v.\ x \in v \wedge openin\ X\ v \wedge diam\ (f\ `\ v) < t)$
by(*auto simp add*: *osc-def Inf-less-iff*)

definition *sequentially-compact* :: *bool* **where**
sequentially-compact $\longleftrightarrow (\forall\, xn{\in}sequence.\ \exists\, a.\ strict\text{-}mono\ a \wedge convergent\text{-}inS\ (xn \circ a))$

definition *eps-net-on* :: $'a$ *set* $\Rightarrow$ *real* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool* **where**
eps-net-on B ε *A* $\longleftrightarrow \varepsilon > 0 \wedge finite\ A \wedge A \subseteq S \wedge B \subseteq (\bigcup a{\in}A.\ open\text{-}ball\ a\ \varepsilon)$

abbreviation *eps-net* $\equiv$ *eps-net-on S*

lemma *eps-net-def*: *eps-net* ε *A* $\longleftrightarrow \varepsilon > 0 \wedge finite\ A \wedge A \subseteq S \wedge S = \bigcup\ ((\lambda a.\ open\text{-}ball\ a\ \varepsilon)\ `\ A)$
using *open-ball-subset-ofS* **by**(*auto simp*: *eps-net-on-def*)

lemma *eps-net-onD*:
assumes *eps-net-on B e A*
shows *e* > 0 *finite A A* $\subseteq S$ *B* $\subseteq (\bigcup a{\in}A.\ open\text{-}ball\ a\ e)$ *B* $\subseteq S$
using *assms open-ball-subset-ofS* **by**(*auto simp*: *eps-net-on-def*) *blast*

lemma *eps-netD*:
assumes *eps-net* ε *A*
shows $\varepsilon > 0$ *finite A A* $\subseteq S$ *S* $= \bigcup\ ((\lambda a.\ open\text{-}ball\ a\ \varepsilon)\ `\ A)$
using *assms* **by**(*auto simp*: *eps-net-def*)

lemma *eps-net-le*:
assumes *eps-net e A* $e \leq e'$
shows *eps-net* e' *A*
using *assms open-ball-le*[*OF assms*(*2*)] *open-ballD'*(*1*)
by(*auto simp*: *eps-net-def*) *blast*

definition *totally-bounded-on* :: *'a set* ⇒ *bool* **where**
totally-bounded-on B ⟷ (∀ *e*>*0*. ∃ *A*. *eps-net-on B e A*)

abbreviation *totally-boundedS* ≡ *totally-bounded-on S*

lemma *totally-boundedS-def*: *totally-boundedS* ⟷ (∀ *e*>*0*. ∃ *A*. *eps-net e A*)
by(*auto simp*: *totally-bounded-on-def*)

lemma *totally-bounded-onD-sub*:
assumes *totally-bounded-on B*
shows *B* ⊆ *S*
by (*meson assms eps-net-onD*(*5*) *gt-ex totally-bounded-on-def*)

lemma *totally-bounded-onD*:
assumes *totally-bounded-on B e* > *0*
obtains *A* **where** *finite A A* ⊆ *S B* ⊆ (⋃ *a*∈*A*. *open-ball a e*)
by (*metis assms that eps-net-on-def totally-bounded-on-def*)

lemma *totally-boundedSD*:
assumes *totally-boundedS e* > *0*
obtains *A* **where** *finite A A* ⊆ *S S* = (⋃ *a*∈*A*. *open-ball a e*)
by (*metis assms that eps-net-def totally-boundedS-def*)

lemma *totally-bounded-on-iff*:
totally-bounded-on B ⟷ *B* ⊆ *S* ∧ (∀ *xn*∈(*UNIV* :: *nat set*) → *B*. ∃ *a*. *strict-mono a* ∧ *Cauchy-inS* (*xn* ∘ *a*))
proof *safe*
fix *xn* :: *nat* ⇒ *'a*
assume *h*:*totally-bounded-on B xn* ∈ *UNIV* → *B*
then have *h'*: *B* ⊆ *S*
by (*auto dest*: *totally-bounded-onD-sub*)
have *1*: ∃ *b*::*nat* ⇒ *nat*. *strict-mono b* ∧ (∀ *n m*. *dist* (*yn* (*b n*)) (*yn* (*b m*)) < *e*)
if *yn* ∈ *UNIV* → *B e* > *0* **for** *e yn*
proof −
obtain *A* **where** *A*: *finite A A* ⊆ *S B* ⊆ (⋃ *a*∈*A*. *open-ball a* (*e*/*2*))
using *totally-bounded-onD*[*OF h*(*1*) *half-gt-zero*[*OF* ‹*e* > *0*›]] **by** *metis*
have ¬ (∀ *a*∈*A*. *finite* {*n*. *yn n* ∈ *open-ball a* (*e*/*2*)})
proof
assume ∀ *a*∈*A*. *finite* {*n*. *yn n* ∈ *open-ball a* (*e*/*2*)}
then have *finite* (⋃ *a*∈*A*. {*n*. *yn n* ∈ *open-ball a* (*e*/*2*)})
using *A* **by** *auto*
moreover have *UNIV* = (⋃ *a*∈*A*. {*n*. *yn n* ∈ *open-ball a* (*e*/*2*)})

```
    using that(1) A(3) by auto
   ultimately show False by simp
  qed
  then obtain a where a:a ∈ A infinite {n. yn n ∈ open-ball a (e/2)}
   by auto
  then obtain b where b:strict-mono b ⋀n::nat. yn (b n) ∈ open-ball a (e/2)
   using obtain-subsequence[of λ- ynn. ynn ∈ open-ball a (e/2) yn] by auto
  show ?thesis
   using a A by(auto intro!: exI[where x=b] b order.strict-trans1[OF dist-tr[OF
open-ballD'(1)[OF b(2)] - open-ballD'(1)[OF b(2)],of a] add-strict-mono[OF open-ballD[OF
b(2),simplified dist-sym[of a]] open-ballD[OF b(2)]],simplified])
 qed

 define anm where anm ≡ rec-nat (xn ∘ (SOME b::nat ⇒ nat. strict-mono b ∧
(∀ n m. dist (xn (b n)) (xn (b m)) < 1))) (λn an. an ∘ (SOME b. strict-mono b ∧
(∀ l k. dist (an (b l)) (an (b k)) < 1 / Suc (Suc n))))
 have anm-Suc:anm (Suc n) = anm n ∘ (SOME b. strict-mono b ∧ (∀ l k. dist
(anm n (b l)) (anm n (b k)) < 1 / Suc (Suc n))) for n
  by(simp add: anm-def)
 have anm1:anm n ∈ UNIV → B ∧ (∀ l k. dist (anm n l) (anm n k) < 1 / Suc
n) for n
 proof(induction n)
  case 0
  obtain b ::nat ⇒ nat where b:strict-mono b ∀ l k. dist (xn (b l)) (xn (b k)) <
1
   using 1[OF h(2),of 1] by auto
  show ?case
   by(simp add: anm-def,rule someI2[where a=b]) (use b h(2) in auto)
 next
  case ih:(Suc n')
  obtain b ::nat ⇒ nat where b:strict-mono b ∀ l k. dist (anm n' (b l)) (anm n'
(b k)) < 1 / real (Suc (Suc n'))
   using 1[of anm n' 1 / Suc (Suc n')] ih by auto
  show ?case
   by(simp only: anm-Suc,rule someI2[where a=b]) (use ih b in auto)
 qed

 define bnm :: nat ⇒ nat ⇒ nat where bnm ≡ rec-nat (SOME b. strict-mono b
∧ anm 0 = xn ∘ b) (λn bn. SOME b. strict-mono b ∧ anm (Suc n) = anm n ∘ b)
 have bnm-Suc:bnm (Suc n) = (SOME b. strict-mono b ∧ anm (Suc n) = anm n
∘ b) for n
  by(simp add: bnm-def)
 have bnm0:strict-mono (bnm 0) ∧ anm 0 = xn ∘ (bnm 0)
 proof −
  have b0:∃ b::nat ⇒ nat. strict-mono b ∧ anm 0 = xn ∘ b
  proof −
   obtain b ::nat ⇒ nat where b:strict-mono b ∀ l k. dist (xn (b l)) (xn (b k))
< 1
    using 1[OF h(2),of 1] by auto
```

```
    show ?thesis
      by(simp add: anm-def,rule someI2[where a=b],auto simp: b)
   qed
   thus ?thesis
    unfolding bnm-def by(simp,rule someI-ex)
 qed
 have bnm-S: strict-mono (bnm (Suc n)) ∧ anm (Suc n) = anm n ∘ (bnm (Suc
n)) for n
 proof −
   have bn:∃ b::nat ⇒ nat. strict-mono b ∧ anm (Suc m) = anm m ∘ b for m
   proof −
     obtain b ::nat ⇒ nat where b:strict-mono b ∀ l k. dist (anm m (b l)) (anm
m (b k)) < 1 / real (Suc (Suc m))
      using 1[of anm m 1 / Suc (Suc m)] anm1 by auto
    show ?thesis
      by(simp only: anm-Suc,rule someI2[where a=b]) (auto simp: b[simplified])
   qed
   thus ?thesis
    by(simp add: bnm-Suc, rule someI-ex)
 qed
 define bnm-r where bnm-r ≡ rec-nat (bnm 0) (λn bn. bn ∘ (bnm (Suc n)))
 have bnm-r-Suc: bnm-r (Suc n) = bnm-r n ∘ (bnm (Suc n)) for n
   by(simp add: bnm-r-def)
 have anm-bnm-r:anm n = xn ∘ (bnm-r n) for n
   by(induction n,simp add: bnm0 bnm-r-def) (auto simp: bnm-S bnm-r-Suc)
 have bnm-r-sm:strict-mono (bnm-r n) for n
  by(induction n, simp add: bnm0 bnm-r-def) (insert bnm-S, auto simp: bnm-r-Suc
strict-mono-def)
 have bnm-r-Suc-le:bnm-r n l ≤ bnm-r (Suc n) l for l n
   using bnm-S bnm-r-sm by(auto simp: bnm-r-Suc strict-mono-imp-increasing
strict-mono-leD)
 have sm:strict-mono (λn. bnm-r n n)
  by(auto simp add: strict-mono-Suc-iff) (meson lessI order-le-less-trans strict-monoD
bnm-r-sm bnm-r-Suc-le)
 have bnm-r-de:∃ l. bnm-r (n + k) = bnm-r n ∘ l for n k
   by(induction k) (auto simp: bnm-r-Suc)
 show ∃ a::nat ⇒ nat. strict-mono a ∧ Cauchy-inS (xn ∘ a)
   unfolding Cauchy-inS-def
 proof(safe intro!: exI[where x=λn. bnm-r n n] sm)
   fix e :: real
   assume e > 0
   then obtain N where N:1 / Suc N < e
     using nat-approx-posE by blast
   show ∃ N. ∀ n≥N. ∀ m≥N. dist ((xn ∘ (λn. bnm-r n n)) n) ((xn ∘ (λn. bnm-r
n n)) m) < e
   proof(safe intro!: exI[where x=N])
     fix n m
     assume N ≤ n N ≤ m
     then have n = N + (n − N) m = N + (m − N) by auto
```

```
    then obtain l1 l2 where l:bnm-r n = bnm-r N ∘ l1 bnm-r m = bnm-r N ∘
l2
      by (metis bnm-r-de)
    have dist (xn (bnm-r n n)) (xn (bnm-r m m)) = dist (anm N (l1 n)) (anm
N (l2 m))
      by(simp add: l anm-bnm-r)
    also have ... < 1 / Suc N
      using anm1 by auto
    finally show dist ((xn ∘ (λn. bnm-r n n)) n) ((xn ∘ (λn. bnm-r n n)) m) < e
      using N by simp
  qed
 qed(use h h' in auto)
next
 assume h:∀ xn∈(UNIV :: nat set) → B. ∃ a. strict-mono a ∧ Cauchy-inS (xn ∘
a) B ⊆ S
 show totally-bounded-on B
 proof(rule ccontr)
  assume ¬ totally-bounded-on B
  then obtain e where e:e > 0 ⋀A. ¬ eps-net-on B e A
   by(auto simp: totally-bounded-on-def)
  have A:¬ B ⊆ (⋃ a∈A. open-ball a e) if finite A for A
  proof −
   have [simp]:(⋃ a∈A. open-ball a e) = (⋃ a∈A∩ S. open-ball a e)
    using Collect-cong IntD1 IntI Sup-set-def UN-iff open-ballD'(2) by auto
   have finite (A ∩ S) using that by auto
   thus ?thesis
    using e by(auto simp: eps-net-on-def)
  qed
  obtain a0 where a0:a0 ∈ B
   using A by fastforce
  define xnl where xnl ≡ rec-nat [a0] (λn ln. (SOME x. x ∈ B ∧ x ∉ (⋃ a∈set
ln. open-ball a e)) # ln)
  have xnl-Suc:xnl (Suc n) = (SOME x. x ∈ B ∧ x ∉ (⋃ a∈set (xnl n). open-ball
a e)) # xnl n for n
   by(simp add: xnl-def)
  define xn where xn = (λn. (xnl n) ! 0)
  have xn:xn (Suc n) ∈ B ∧ xn (Suc n) ∉ (⋃ a∈set (xnl n). open-ball a e) for n
  proof −
   have ∃ y. y ∈ B ∧ (∀ x∈set (xnl n). y ∉ open-ball x e)
    using A[OF finite-set] by fastforce
   thus ?thesis
    by(simp add: xn-def xnl-Suc,rule someI-ex)
  qed
  have xn0:xn 0 ∈ B
   by(auto simp: xnl-def xn-def a0)
  with xn have xns:xn ∈ UNIV → B
   by auto (metis old.nat.exhaust)
  have xnll:length (xnl n) = Suc n for n
   by(induction n) (simp add: xnl-def, auto simp: xnl-Suc)
```

have *xnin*:*xn m* ∈ *set* (*xnl* (*m* + *k*)) **for** *m k*
by(*induction k*) (*auto simp*: *xn-def xnl-Suc xnll intro*!: *nth-mem*)
obtain *a* **where** *a*:*strict-mono a Cauchy-inS* (*xn* ∘ *a*)
using *h xns* **by** *auto*
then obtain *N* **where** ⋀*n m*. *n* ≥ *N* ⟹ *m* ≥ *N* ⟹ *dist* (*xn* (*a n*)) (*xn* (*a m*)) < *e*
using *e Cauchy-inS-def* **by** *fastforce*
hence *e1*:*dist* (*xn* (*a N*)) (*xn* (*a* (*Suc N*))) < *e*
by *auto*
have *xn* (*a* (*Suc N*)) ∉ (⋃ *a*∈*set* (*xnl* (*a* (*Suc N*) − *1*)). *open-ball a e*)
by (*metis a*(*1*) *diff-Suc-1 le-0-eq not0-implies-Suc strict-mono-less-eq xn zero-le*)
moreover have *xn* (*a N*) ∈ *set* (*xnl* (*a* (*Suc N*) − *1*))
using *a*(*1*)[*simplified strict-mono-Suc-iff*] *xnin*[*of a N a* (*Suc N*) − *a N* − *1*]
by (*simp add*: *Suc-leI*)
ultimately have *xn* (*a* (*Suc N*)) ∉ *open-ball* (*xn* (*a N*)) *e*
by *auto*
from *open-ball-nin-le*[*OF - e*(*1*) *- this*] *xns e1 h*(*2*)
show *False* **by** *auto*
qed
qed(*auto dest*: *totally-bounded-onD-sub*)

corollary *totally-boundedS-iff*: *totally-boundedS* ⟷ (∀ *xn*∈*sequence*. ∃ *a*. *strict-mono a* ∧ *Cauchy-inS* (*xn* ∘ *a*))
by(*auto simp*: *totally-bounded-on-iff*)

Metric embedding

definition *embed-dist-on* :: ['*b set*, '*b* ⇒ '*a*, '*b*, '*b*] ⇒ *real* **where**
embed-dist-on B f a b ≡ (*if a* ∈ *B* ∧ *b* ∈ *B then dist* (*f a*) (*f b*) *else 0*)

context
fixes *f B*
assumes *f*: *f* ∈ *B* → *S inj-on f B*
begin

abbreviation *ed* ≡ *embed-dist-on B f*

lemma *embed-dist-dist*: *metric-set B* (*embed-dist-on B f*)
proof
fix *x y*
assume *x* ∈ *B y* ∈ *B*
then show *x* = *y* ⟷ *embed-dist-on B f x y* = *0*
using *inj-onD*[*OF f*(*2*)] *dist-0*[*of f x f y*] *f*(*1*)
by(*auto simp*: *embed-dist-on-def*)
next
fix *x y*
show *embed-dist-on B f x y* = *embed-dist-on B f y x*
by(*simp add*: *embed-dist-on-def dist-sym*[*of f x f y*])
next

```
  fix x y z
  assume x ∈ B y ∈ B z ∈ B
  then show embed-dist-on B f x z ≤ embed-dist-on B f x y + embed-dist-on B f
y z
    using dist-tr[of f x f y f z] f(1) by(auto simp: embed-dist-on-def)
qed(simp-all add: embed-dist-on-def  dist-geq0)

interpretation ed : metric-set B ed
  by(rule embed-dist-dist)

lemma embed-dist-open-ball:
  assumes a ∈ B
  shows f ‘ (ed.open-ball a e) = open-ball (f a) e ∩ f ‘ B
  using assms f by(auto simp: ed.open-ball-def open-ball-def embed-dist-on-def)

lemma embed-dist-closed-ball:
  assumes a ∈ B
  shows f ‘ (ed.closed-ball a e) = closed-ball (f a) e ∩ f ‘ B
  using assms f by(auto simp: ed.closed-ball-def closed-ball-def embed-dist-on-def)

lemma embed-dist-topology-homeomorphic-maps:
  assumes g1:⋀x. x ∈ B ⟹ g (f x) = x
  shows homeomorphic-maps ed.mtopology (subtopology mtopology (f ‘ B)) f g
proof −
  have g2: ⋀x. x ∈ f ‘ B ⟹ f (g x) = x g ∈ (f ‘ B) → B
    by(auto simp: g1)
  show ?thesis
    unfolding homeomorphic-maps-def mtopology-topspace ed.mtopology-topspace
  proof safe
    show continuous-map ed.mtopology (subtopology mtopology (f ‘ B)) f
      unfolding mtopology-def2 subtopology-generated-by
    proof(rule continuous-on-generated-topo)
      show ⋀U. U ∈ {f ‘ B ∩ U |U. U ∈ {open-ball a ε |a ε. a ∈ S ∧ 0 < ε}}
⟹ openin ed.mtopology (f −‘ U ∩ topspace ed.mtopology)
        unfolding ed.mtopology-topspace
      proof safe
        fix a and e :: real
        assume h:a ∈ S 0 < e
        have 1:(f −‘ (f ‘ B ∩ open-ball a e) ∩ B) = f −‘ open-ball a e ∩ B by blast
        show openin ed.mtopology (f −‘ (f ‘ B ∩ open-ball a e) ∩ B)
          unfolding 1 ed.mtopology-openin-iff
        proof safe
          fix x
          assume h':x ∈ B f x ∈ open-ball a e
          then obtain e' where e':e' > 0 open-ball (f x) e' ⊆ open-ball a e
            using mtopology-open-ball-in' by blast
          show ∃ε>0. ed.open-ball x ε ⊆ f −‘ open-ball a e ∩ B
          proof(safe intro!: exI[where x=e'])
            fix y
```

assume $y \in$ *ed.open-ball x e'*
with *e'(2)* **show** $y \in f\ -`$ *open-ball a e*
using *embed-dist-open-ball*[*OF h'(1),of e'*] **by** *blast*
qed(*use e' ed.open-ball-subset-ofS* **in** *auto*)
qed
qed
next
show *f ` topspace ed.mtopology* $\subseteq \bigcup$ {*f ` B* $\cap$ *U* |*U*. *U* $\in$ {*open-ball a* ε |*a* ε. *a* $\in$ *S* $\wedge$ *0* $<$ ε}}
by(*auto simp*: *ed.mtopology-topspace*) (*metis* (*mono-tags*, *opaque-lifting*) *IntE IntI closed-ball-def ed.closed-ball-ina ed.dist-set-geq0 ed.dist-set-inA embed-dist-closed-ball ennreal-le-epsilon ennreal-zero-less-top image-eqI le-zero-eq not-gr-zero open-ballD'(2) open-ball-ina open-ball-le-0*)
qed
next
show *continuous-map* (*subtopology mtopology* (*f ` B*)) *ed.mtopology g*
unfolding *ed.mtopology-def2*
proof(*rule continuous-on-generated-topo*)
show $\bigwedge$*U*. *U* $\in$ {*ed.open-ball a* ε |*a* ε. *a* $\in$ *B* $\wedge$ *0* $<$ ε} $\Longrightarrow$ *openin* (*subtopology mtopology* (*f ` B*)) (*g −` U* $\cap$ *topspace* (*subtopology mtopology* (*f ` B*)))
proof *safe*
fix *a* **and** *e* :: *real*
assume *h*: *a* $\in$ *B 0* $<$ *e*
then have *1*: *g −` ed.open-ball a e* $\cap$ (*S* $\cap$ *f ` B*) = *open-ball* (*f a*) *e* $\cap$ *f ` B*
using *f(1) g1 g2* **by**(*auto simp*: *ed.open-ball-def open-ball-def embed-dist-on-def*)
show *openin* (*subtopology mtopology* (*f ` B*)) (*g −` ed.open-ball a e* $\cap$ (*topspace* (*subtopology mtopology* (*f ` B*))))
by(*auto simp*: *1 openin-subtopology openin-open-ball mtopology-topspace intro*!: *exI*[**where** *x*=*open-ball* (*f a*) *e*])
qed
show *g ` topspace* (*subtopology mtopology* (*f ` B*)) $\subseteq \bigcup$ {*ed.open-ball a* ε |*a* ε. *a* $\in$ *B* $\wedge$ *0* $<$ ε}
by(*auto simp*: *mtopology-topspace*) (*metis ed.mtopology-openin-iff ed.open-ball-ina ed.openin-S g1*)
qed
qed(*use g1 g2* **in** *auto*)
qed

lemma *embed-dist-topology-homeomorphic-map*:
homeomorphic-map ed.mtopology (*subtopology mtopology* (*f ` B*)) *f*
proof −
define *g* **where** *g* $\equiv$ (λ*y*. *THE x*. *x* $\in$ *B* $\wedge$ *f x* = *y*)
have *g1*: *g* (*f b*) = *b* **if** *b* $\in$ *B* **for** *b*
unfolding *g-def* **by**(*rule theI2*[*of* - *b*]) (*insert that f(2)*, *auto simp*: *inj-on-def*)
thus *?thesis*
using *embed-dist-topology-homeomorphic-maps homeomorphic-map-maps* **by** *blast*
qed

corollary *embed-dist-topology-homeomorphic*:
ed.mtopology homeomorphic-space (*subtopology mtopology* (*f* ‘ *B*))
using *embed-dist-topology-homeomorphic-map*
by(*rule homeomorphic-map-imp-homeomorphic-space*)

corollary *embed-dist-topology-homeomorphic-map′*:
assumes *f* ‘ *B* = *S*
shows *homeomorphic-map ed.mtopology mtopology f*
using *embed-dist-topology-homeomorphic-map*[*simplified assms*]
by(*simp add*:*subtopology-topspace*[*of mtopology*, *simplified mtopology-topspace*])

corollary *embed-dist-topology-homeomorphic′*:
assumes *f* ‘ *B* = *S*
shows *ed.mtopology homeomorphic-space mtopology*
using *embed-dist-topology-homeomorphic-map′*[*OF assms*]
by(*rule homeomorphic-map-imp-homeomorphic-space*)

lemma *embed-dist-converge-to-inS-iff*:
ed.converge-to-inS xn x ⟷ *xn* ∈ *ed.sequence* ∧ *x* ∈ *B* ∧ *converge-to-inS* (λ*n*. *f* (*xn n*)) (*f x*)
proof *safe*
assume *h*:*ed.converge-to-inS xn x*
then show *h′*:*x* ∈ *B* ⋀*n*. *xn n* ∈ *B*
by(*auto simp*: *ed.converge-to-inS-def*)
thus *converge-to-inS* (λ*n*. *f* (*xn n*)) (*f x*)
using *h f* **by**(*auto simp*: *converge-to-inS-def2 ed.converge-to-inS-def2 embed-dist-on-def*)
next
assume *h*:*xn* ∈ *ed.sequence x* ∈ *B converge-to-inS* (λ*n*. *f* (*xn n*)) (*f x*)
show *ed.converge-to-inS xn x*
using *h f* **by**(*fastforce simp*: *ed.converge-to-inS-def2 h embed-dist-on-def converge-to-inS-def2*)
qed

lemma *embed-dist-convergent-inS-iff*:
assumes *closedin mtopology* (*f* ‘ *B*)
shows *ed.convergent-inS xn* ⟷ *xn* ∈ *ed.sequence* ∧ *convergent-inS* (λ*n*. *f* (*xn n*))
proof −
{
fix *s*
assume *h*:*xn* ∈ *ed.sequence converge-to-inS* (λ*n*. *f* (*xn n*)) *s*
with *f* **have** (λ*n*. *f* (*xn n*)) ∈ *UNIV* → *f* ‘ *B* **by** *auto*
hence *s* ∈ *f* ‘ *B*
using *assms h*(*2*) **by**(*auto simp*: *mtopology-closedin-iff*)
hence ∃ *b* ∈ *B*. *s* = *f b* **by** *auto*
}
thus *?thesis*

using *embed-dist-converge-to-inS-iff*[*of xn*] *f*(*1*)
by(*fastforce simp*: *ed.convergent-inS-def convergent-inS-def*)
qed

lemma *embed-dist-Cauchy-inS-iff*:
ed.Cauchy-inS xn ⟷ *xn* ∈ *ed.sequence* ∧ *Cauchy-inS* (λ*n*. *f* (*xn n*))
using *f*(*1*) **by**(*auto simp*: *ed.Cauchy-inS-def Cauchy-inS-def embed-dist-on-def*;
meson PiE UNIV-I)

end

end

Relations to elementary topology.

lemma *ball-def-set*: *ball a ε* = *metric-set.open-ball UNIV dist a ε*
using *metric-set.open-ball-def metric-class-metric-set*
by *fastforce*

lemma *converge-to-def-set*:
fixes *xn* :: *nat* ⇒ (*'a*::*metric-space*)
shows *xn* ⟶ *x* ⟷ *metric-set.converge-to-inS UNIV dist xn x*
proof −
interpret *m*: *metric-set UNIV dist*
by *simp*
show *?thesis*
by(*simp add*: *lim-sequentially m.converge-to-inS-def*)
qed

lemma *the-limit-of-limit*:
fixes *xn* :: *nat* ⇒ (*'a*::*metric-space*)
shows *metric-set.the-limit-of UNIV dist xn* = *lim xn*
by(*simp add*: *metric-set.the-limit-of-def lim-def converge-to-def-set*)

lemma *convergent-def-set*:
fixes *f* :: *nat* ⇒ (*'a*::*metric-space*)
shows *convergent f* ⟷ *metric-set.convergent-inS UNIV dist f*
proof −
interpret *m*: *metric-set UNIV dist*
by(*rule metric-class-metric-set*)
show *convergent f* ⟷ *m.convergent-inS f*
using *converge-to-def-set*[*of f*]
by(*auto simp*: *convergent-def m.convergent-inS-def*)
qed

lemma *Cahuchy-def-set*: *Cauchy f* ⟷ *metric-set.Cauchy-inS UNIV dist f*
proof −
interpret *m*: *metric-set UNIV dist*
by(*rule metric-class-metric-set*)
show *Cauchy f* = *m.Cauchy-inS f*

by(*simp add*: *Cauchy-def m.Cauchy-inS-def dist-real-def*)
qed

lemma *open-openin-set*: *open U* $\longleftrightarrow$ *openin* (*metric-set.mtopology UNIV dist*) *U*
(**is** *?LHS* $\longleftrightarrow$ *?RHS*)
proof −
interpret *m*: *metric-set UNIV dist*
by(*rule metric-class-metric-set*)
have *?LHS* $\longleftrightarrow$ ($\forall x{\in}U.\ \exists e{>}0.$ *ball x e* $\subseteq$ *U*)
by(*simp add*: *open-contains-ball*)
also have ... $\longleftrightarrow$ ($\forall x{\in}U.\ \exists e{>}0.$ *m.open-ball x e* $\subseteq$ *U*)
by(*simp add*: *ball-def-set*)
also have ... $\longleftrightarrow$ *?RHS*
by(*simp add*: *m.mtopology-openin-iff*[*of U*])
finally show *?thesis* **.**
qed

lemma *topological-basis-set*: *topological-basis* $\mathcal{O}$ $\longleftrightarrow$ *metric-set.mtopology-basis UNIV dist* $\mathcal{O}$
(**is** *?LHS* = *?RHS*)
proof −
interpret *m*: *metric-set UNIV dist*
by(*rule metric-class-metric-set*)
have *?LHS* $\longleftrightarrow$ ($\forall b{\in}\mathcal{O}.$ *open b*) $\wedge$ ($\forall x.$ *open x* $\longrightarrow$ ($\exists B'{\subseteq}\mathcal{O}.\ \bigcup\ B' = x$))
by(*simp add*: *topological-basis-def*)
also have ... $\longleftrightarrow$ ($\forall b{\in}\mathcal{O}.$ *openin m.mtopology b*) $\wedge$ ($\forall x.$ *openin m.mtopology x* $\longrightarrow$ ($\exists B'{\subseteq}\mathcal{O}.\ \bigcup\ B' = x$))
by(*simp add*: *open-openin-set*)
also have ... $\longleftrightarrow$ *?RHS*
by(*simp add*: *base-of-def2'*)
finally show *?thesis* **.**
qed

lemma *euclidean-mtopology*: *metric-set.mtopology UNIV dist* = *euclidean*
using *open-openin open-openin-set topology-eq* **by** *blast*

Distances generate the same topological space.

lemma *metric-generates-same-topology*:
assumes *metric-set S d metric-set S d'*
$\bigwedge x\ U.\ U \subseteq S \Longrightarrow (\forall y{\in}U.\ \exists\varepsilon{>}0.$ *metric-set.open-ball S d y* $\varepsilon \subseteq U) \Longrightarrow$
$x \in U \Longrightarrow \exists\varepsilon{>}0.$ *metric-set.open-ball S d' x* $\varepsilon \subseteq U$
and $\bigwedge x\ U.\ U \subseteq S \Longrightarrow (\forall y{\in}U.\ \exists\varepsilon{>}0.$ *metric-set.open-ball S d' y* $\varepsilon \subseteq U)$
$\Longrightarrow x \in U \Longrightarrow \exists\varepsilon{>}0.$ *metric-set.open-ball S d x* $\varepsilon \subseteq U$
shows *metric-set.mtopology S d* = *metric-set.mtopology S d'*
proof −
interpret *m1*: *metric-set S d* **by** *fact*
interpret *m2*: *metric-set S d'* **by** *fact*
have ($\lambda U.\ U \subseteq S \wedge (\forall x{\in}U.\ \exists\varepsilon{>}0.$ *m1.open-ball x* $\varepsilon \subseteq U$)) = ($\lambda U.\ U \subseteq S \wedge$
($\forall x{\in}U.\ \exists\varepsilon{>}0.$ *m2.open-ball x* $\varepsilon \subseteq U$))

by *standard* (*use assms*(*3*,*4*) **in** *auto*)
thus *?thesis*
using *topology.topology-inject m1.mtopology-istopology m2.mtopology-istopology*
by(*simp add*: *m2.mtopology-def m1.mtopology-def*)
qed

lemma *metric-generates-same-topology-inverse*:
assumes *metric-set S d metric-set S d′*
and *metric-set.mtopology S d* = *metric-set.mtopology S d′*
shows $U \subseteq S \Longrightarrow (\forall\, y{\in}U.\ \exists \varepsilon{>}0.\ \textit{metric-set.open-ball}\ S\ d\ y\ \varepsilon \subseteq U) \Longrightarrow x \in U \Longrightarrow \exists \varepsilon{>}0.\ \textit{metric-set.open-ball}\ S\ d'\ x\ \varepsilon \subseteq U$
and $U \subseteq S \Longrightarrow (\forall\, y{\in}U.\ \exists \varepsilon{>}0.\ \textit{metric-set.open-ball}\ S\ d'\ y\ \varepsilon \subseteq U) \Longrightarrow x \in U \Longrightarrow \exists \varepsilon{>}0.\ \textit{metric-set.open-ball}\ S\ d\ x\ \varepsilon \subseteq U$
proof −
interpret *m1*: *metric-set S d* **by** *fact*
interpret *m2*: *metric-set S d′* **by** *fact*
have $(\lambda U.\ U \subseteq S \wedge (\forall\, x{\in}U.\ \exists \varepsilon{>}0.\ \textit{m1.open-ball}\ x\ \varepsilon \subseteq U)) = (\lambda U.\ U \subseteq S \wedge (\forall\, x{\in}U.\ \exists \varepsilon{>}0.\ \textit{m2.open-ball}\ x\ \varepsilon \subseteq U))$
using *topology.topology-inject*[*of* $\lambda U.\ U \subseteq S \wedge (\forall\, x{\in}U.\ \exists \varepsilon{>}0.\ \textit{m1.open-ball}\ x\ \varepsilon \subseteq U)$ $\lambda U.\ U \subseteq S \wedge (\forall\, x{\in}U.\ \exists \varepsilon{>}0.\ \textit{m2.open-ball}\ x\ \varepsilon \subseteq U)$] *m1.mtopology-istopology m2.mtopology-istopology assms*(*3*)
by(*auto simp*: *m2.mtopology-def m1.mtopology-def*)
thus $U \subseteq S \Longrightarrow \forall\, y{\in}U.\ \exists \varepsilon{>}0.\ \textit{m1.open-ball}\ y\ \varepsilon \subseteq U \Longrightarrow x \in U \Longrightarrow \exists \varepsilon{>}0.\ \textit{m2.open-ball}\ x\ \varepsilon \subseteq U$
$U \subseteq S \Longrightarrow \forall\, y{\in}U.\ \exists \varepsilon{>}0.\ \textit{m2.open-ball}\ y\ \varepsilon \subseteq U \Longrightarrow x \in U \Longrightarrow \exists \varepsilon{>}0.\ \textit{m1.open-ball}\ x\ \varepsilon \subseteq U$
by(*auto dest*: *fun-cong*[**where** *x*=*U*])
qed

lemma *metric-generates-same-topology-converges′*:
assumes *metric-set S d metric-set S d′*
metric-set.mtopology S d = *metric-set.mtopology S d′*
and *metric-set.converge-to-inS S d f x*
shows *metric-set.converge-to-inS S d′ f x*
proof −
interpret *m1*: *metric-set S d* **by** *fact*
interpret *m2*: *metric-set S d′* **by** *fact*
show *?thesis*
unfolding *m2.converge-to-inS-def2′*
proof *safe*
fix ε :: *real*
assume *h*:$0 < \varepsilon$
obtain ε' **where** *he*:
$\varepsilon'{>}0$ *m1.open-ball x* $\varepsilon' \subseteq$ *m2.open-ball x* ε
using *m2.mtopology-open-ball-in′*[*of - x*] *assms*(*4*)[*simplified m1.converge-to-inS-def2′*] *metric-generates-same-topology-inverse*(*2*)[*OF assms*(*1*−*3*) *m2.open-ball-subset-ofS*, *of x* ε,*OF - m2.open-ball-ina*[*OF - h*,*of x*]]
by *auto*
then obtain *N* **where** *hn*:

$\forall n{\geq}N.\ f\ n \in m1.open\text{-}ball\ x\ \varepsilon'$
using *assms*(4)[*simplified m1.converge-to-inS-def2'*] **by** *auto*
show $\exists N.\ \forall n{\geq}N.\ f\ n \in m2.open\text{-}ball\ x\ \varepsilon$
using *hn he*(2) **by**(*auto intro*!: *exI*[**where** $x{=}N$])
next
show $\bigwedge x.\ f\ x \in S\ x \in S$
using *assms*(4)[*simplified m1.converge-to-inS-def2'*] **by** *auto*
qed
qed

lemma *metric-generates-same-topology-converges*:
assumes *metric-set S d metric-set S d'*
and *metric-set.mtopology S d* = *metric-set.mtopology S d'*
shows *metric-set.converge-to-inS S d f x* $\longleftrightarrow$ *metric-set.converge-to-inS S d' f x*
using *metric-generates-same-topology-converges'*[*OF assms*(2,1) *assms*(3)[*symmetric*]] *metric-generates-same-topology-converges'*[*OF assms*(1−3)]
by *auto*

lemma *metric-generates-same-topology-convergent*:
assumes *metric-set S d metric-set S d'*
and *metric-set.mtopology S d* = *metric-set.mtopology S d'*
shows *metric-set.convergent-inS S d f* $\longleftrightarrow$ *metric-set.convergent-inS S d' f*
using *metric-generates-same-topology-converges*[*OF assms*,*of f*]
by (*simp add*: *assms*(1) *assms*(2) *metric-set.convergent-inS-def*)

2.1.1 Sub-Metric Spaces

definition *submetric* :: ['*a set*, '*a* ⇒ '*a* ⇒ *real*] ⇒ '*a* ⇒ '*a* ⇒ *real* **where**
submetric S' d ≡ (λ*x y*. *if x* ∈ *S'* ∧ *y* ∈ *S' then d x y else* 0)

lemma(**in** *metric-set*) *submetric-metric-set*:
assumes $S' \subseteq S$
shows *metric-set S'* (*submetric S' dist*)
proof
show $\bigwedge x\ y.\ 0 \leq submetric\ S'\ dist\ x\ y$
$\bigwedge x\ y.\ x \notin S' \Longrightarrow submetric\ S'\ dist\ x\ y = 0$
$\bigwedge x\ y.\ x \in S' \Longrightarrow y \in S' \Longrightarrow (x = y) = (submetric\ S'\ dist\ x\ y = 0)$
$\bigwedge x\ y.\ submetric\ S'\ dist\ x\ y = submetric\ S'\ dist\ y\ x$
using *assms dist-geq0 dist-tr dist-0 dist-sym*
by(*fastforce simp*: *submetric-def*)+
next
show $\bigwedge x\ y\ z.\ x \in S' \Longrightarrow y \in S' \Longrightarrow z \in S' \Longrightarrow submetric\ S'\ dist\ x\ z \leq submetric\ S'\ dist\ x\ y + submetric\ S'\ dist\ y\ z$
by (*metis assms dist-tr submetric-def subset-iff*)
qed

lemma(**in** *metric-set*) *submetric-open-ball*:
assumes $S' \subseteq S$ **and** $a \in S'$

shows *open-ball a ε ∩ S′ = metric-set.open-ball S′ (submetric S′ dist) a ε*
proof −
interpret *m*: *metric-set S′ submetric S′ dist*
by(*rule submetric-metric-set*[*OF assms*(*1*)])
show *?thesis*
using *assms* **by**(*auto simp*: *open-ball-def m.open-ball-def*,*simp-all add*: *submetric-def*)
qed

lemma(**in** *metric-set*) *submetric-open-ball-subset*:
assumes *S′ ⊆ S*
shows *metric-set.open-ball S′ (submetric S′ dist) a ε ⊆ open-ball a ε*
proof −
interpret *m*: *metric-set S′ submetric S′ dist*
by(*rule submetric-metric-set*[*OF assms*(*1*)])
show *?thesis*
by (*metis assms empty-subsetI inf-commute inf-sup-ord*(*2*) *m.open-ball-nin submetric-open-ball*)
qed

lemma(**in** *metric-set*) *submetric-subtopology*:
assumes *S′ ⊆ S*
shows *subtopology mtopology S′ = metric-set.mtopology S′ (submetric S′ dist)*
proof −
interpret *m*: *metric-set S′ submetric S′ dist*
by(*rule submetric-metric-set*[*OF assms*(*1*)])
show *?thesis*
unfolding *topology-eq*
proof *safe*
fix *U*
assume *openin* (*subtopology mtopology S′*) *U*
then obtain *T* **where** *ht*: *openin mtopology T U = T ∩ S′*
by(*auto simp*: *openin-subtopology*)
have *U ⊆ S′*
by (*simp add*: *ht*(*2*))
show *openin m.mtopology U*
unfolding *m.mtopology-openin-iff*
proof *safe*
fix *x*
assume *x ∈ U*
then obtain *ε* **where** *he*: *ε > 0 open-ball x ε ⊆ T*
using *ht* **by**(*auto simp*: *mtopology-openin-iff*)
thus *∃ε>0. m.open-ball x ε ⊆ U*
using *ht*(*2*) ‹*x ∈ U*› *submetric-open-ball*[*OF assms*(*1*),*of x ε*]
by(*auto intro*!: *exI*[**where** *x*=*ε*])
qed(*use* ‹*U ⊆ S′*› **in** *auto*)
next
fix *U*
assume *openin m.mtopology U*

```
    then have ∀ x∈U. ∃ε>0. m.open-ball x ε ⊆ U
      by(simp add: m.mtopology-openin-iff)
    then obtain ε where he:
     ⋀x. x ∈ U ⟹ ε x > 0 ⋀x. x ∈ U ⟹ m.open-ball x (ε x) ⊆ U
      by metis
    have U ⊆ S'
      using ‹openin m.mtopology U› m.mtopology-openin-iff by auto

    show openin (subtopology mtopology S') U
      unfolding openin-subtopology
    proof(intro exI[where x=⋃ { open-ball x (ε x) | x. x∈U}] conjI)
      show openin mtopology (⋃ { open-ball x (ε x) | x. x∈U})
        by(rule openin-Union) (use he(1) open-ball-def mtopology-open-ball-in in
fastforce)
    next
      have *:U = (⋃ { m.open-ball x (ε x) | x. x∈U})
        using he m.open-ball-ina ‹U ⊆ S'› by fastforce
      also have ... = (⋃ { open-ball x (ε x) ∩ S' | x. x∈U})
        using submetric-open-ball[OF assms(1)] ‹U ⊆ S'› by auto
      also have ... = (⋃ { open-ball x (ε x) | x. x∈U}) ∩ S'
        by auto
      finally show U = ⋃ {open-ball x (ε x) |x. x ∈ U} ∩ S' .
    qed
  qed
qed

lemma(in metric-set) converge-to-insub-converge-to-inS:
  assumes S' ⊆ S and metric-set.converge-to-inS S' (submetric S' dist) f x
  shows converge-to-inS f x
proof −
  interpret m: metric-set S' submetric S' dist
    by(rule submetric-metric-set[OF assms(1)])
  have *:f ∈ m.sequence x ∈ S'
    using assms(2) by(auto simp: m.converge-to-inS-def)
  show ?thesis
   unfolding converge-to-inS-def2 using * assms[simplified m.converge-to-inS-def2]
    by(auto simp: submetric-def funcset-mem)
qed

lemma(in metric-set) convergent-insub-convergent-inS:
  assumes S' ⊆ S and metric-set.convergent-inS S' (submetric S' dist) f
  shows convergent-inS f
 by (meson assms(1) assms(2) converge-to-insub-converge-to-inS convergent-inS-def
in-mono metric-set.convergent-inS-def submetric-metric-set)

lemma(in metric-set) Cauchy-insub-Cauchy:
  assumes S' ⊆ S and metric-set.Cauchy-inS S' (submetric S' dist) f
  shows Cauchy-inS f
proof −
```

interpret *m*: *metric-set S′ submetric S′ dist*
by(*rule submetric-metric-set*[*OF assms*(*1*)])
have ∗:*f* ∈ *m.sequence*
using *assms*(*2*) **by**(*auto simp*: *m.Cauchy-inS-def*)
show *?thesis*
unfolding *Cauchy-inS-def* **using** ∗ *assms*[*simplified m.Cauchy-inS-def*]
by(*auto simp*: *submetric-def funcset-mem*[*OF* ∗])
qed

lemma(**in** *metric-set*) *Cauchy-insub-Cauchy-inverse*:
assumes *S′* ⊆ *S f* ∈ *UNIV* → *S′ Cauchy-inS f*
shows *metric-set.Cauchy-inS S′* (*submetric S′ dist*) *f*
proof −
interpret *m*: *metric-set S′ submetric S′ dist*
by(*rule submetric-metric-set*[*OF assms*(*1*)])
show *?thesis*
using *assms* **by**(*auto simp*: *m.Cauchy-inS-def Cauchy-inS-def*,*simp add*: *submetric-def*) *metis*
qed

lemma(**in** *metric-set*) *convergent-insubmetric*:
assumes *S′* ⊆ *S f* ∈ *UNIV* → *S′ x* ∈ *S′ converge-to-inS f x*
shows *metric-set.converge-to-inS S′* (*submetric S′ dist*) *f x*
proof −
interpret *m*: *metric-set S′ submetric S′ dist*
by(*rule submetric-metric-set*[*OF assms*(*1*)])
show *?thesis*
unfolding *m.converge-to-inS-def* **using** *assms*
by(*auto simp*: *converge-to-inS-def funcset-mem*[*OF assms*(*2*)] *submetric-def*)
qed

lemma(**in** *metric-set*) *the-limit-of-submetric-eq*:
assumes *S′* ⊆ *S metric-set.convergent-inS S′* (*submetric S′ dist*) *f*
shows *metric-set.the-limit-of S′* (*submetric S′ dist*) *f* = *the-limit-of f*
by (*meson assms*(*1*) *assms*(*2*) *converge-to-insub-converge-to-inS convergent-insub-convergent-inS metric-set.converge-to-inS-unique metric-set.the-limit-if-converge metric-set-axioms submetric-metric-set*)

lemma *submetric-of-euclidean*:
metric-set A (*submetric A dist*) *metric-set.mtopology A* (*submetric A dist*) = *top-of-set A*
using *metric-set.submetric-metric-set*[*OF metric-class-metric-set*,*of A*] *metric-set.submetric-subtopology*[*OF metric-class-metric-set*,*of A*]
by(*auto simp*: *euclidean-mtopology*)

lemma(**in** *metric-set*)
assumes *B* ⊆ *S*
shows *totally-bounded-on-submetric*: *totally-bounded-on B* ⟷ *metric-set.totally-boundedS B* (*submetric B dist*)

proof −
interpret *m*: *metric-set B submetric B dist*
by(*rule submetric-metric-set*[*OF assms*(*1*)])
show *?thesis*
unfolding *totally-bounded-on-def m.totally-boundedS-def*
proof *safe*
fix *e* :: *real*
assume *h*:∀ *e*>*0*. ∃ *A*. *eps-net-on B e A e* > *0*
then obtain *A* **where** *A*:*eps-net-on B* (*e* / *2*) *A*
by *fastforce*
define *A′* **where** *A′* ≡ *A* ∩ {*z*. *open-ball z* (*e* / *2*) ∩ *B* ≠ {}}
have *A′*: *eps-net-on B* (*e* / *2*) *A′*
unfolding *eps-net-on-def*
proof *safe*
fix *x*
assume *x*:*x* ∈ *B*
then obtain *a* **where** *a*:*a* ∈ *A x* ∈ *open-ball a* (*e* / *2*)
using *A* **by**(*auto dest*: *eps-net-onD*)
with *x* **have** *a* ∈ *A′*
by(*auto simp*: *A′-def*)
with *a* **show** *x* ∈ (⋃ *a*∈*A′*. *open-ball a* (*e* / *2*)) **by** *auto*
qed(*use h eps-net-on-def A′-def A* **in** *auto*)
define *b* **where** *b* ≡ (λ*a*. *SOME b*. *b* ∈ *B* ∧ *b* ∈ *open-ball a* (*e* / *2*))
have *b*:*b a* ∈ *B b a* ∈ *open-ball a* (*e* / *2*) **if** *a*: *a* ∈ *A′* **for** *a*
proof −
have *b a* ∈ *B* ∧ *b a* ∈ *open-ball a* (*e* / *2*)
unfolding *b-def* **by**(*rule someI-ex*) (*insert that*, *auto simp*: *A′-def*)
thus *b a* ∈ *B b a* ∈ *open-ball a* (*e* / *2*) **by** *auto*
qed
show ∃ *A*. *m.eps-net e A*
unfolding *m.eps-net-on-def*
proof(*safe intro*!: *exI*[**where** *x*=*b* ‘ *A′*])
fix *x*
assume *x* ∈ *B*
then obtain *a* **where** *a*: *a* ∈ *A′ x* ∈ *open-ball a* (*e* / *2*)
using *A′* **by**(*auto simp*: *eps-net-on-def*)
show *x* ∈ (⋃ *a*∈*b* ‘ *A′*. *m.open-ball a e*)
proof
show *b a* ∈ *b* ‘ *A′*
using *a* **by** *auto*
next
have [*simp*]: *b a* ∈ *S x* ∈ *S b a* ∈ *B x* ∈ *B a* ∈ *S*
using *b*(*1*)[*OF a*(*1*)] *assms* ‹*x* ∈ *B*› *a A′* **by** (*auto simp*: *eps-net-on-def*)
note *order.strict-trans1*[*OF dist-tr add-strict-mono*[*OF open-ballD*[*OF a*(*2*),*simplified dist-sym*[*of a*]] *open-ballD*[*OF b*(*2*)[*OF a*(*1*)]]],*simplified*]
hence *submetric B dist x* (*b a*) < *e*
by(*auto simp*: *submetric-def*)
thus *x* ∈ *m.open-ball* (*b a*) *e*
by(*auto simp*: *m.open-ball-def m.dist-sym*)

qed
qed(*insert h*(*2*) *A' b*, *auto simp*: *eps-net-on-def*)
next
fix *e* :: *real*
assume ∀ *e*>*0*. ∃ *A*. *m.eps-net e A e* > *0*
then obtain *A* **where** *A*: *m.eps-net e A* **by** *auto*
thus ∃ *A*. *eps-net-on B e A*
using *assms submetric-open-ball-subset*[*OF assms*] **by**(*auto intro*!: *exI*[**where** *x*=*A*] *simp*: *eps-net-on-def m.eps-net-def*) *blast*
qed
qed

Continuous functions

context
fixes *S* :: *'a set* **and** *d*
and *S'*:: *'b set* **and** *d'*
assumes *metric-set S d metric-set S' d'*
begin

interpretation *m1*: *metric-set S d* **by** *fact*
interpretation *m2*: *metric-set S' d'* **by** *fact*

lemma *metric-set-continuous-map-eq*:
shows *continuous-map m1.mtopology m2.mtopology f*
⟷ *f* ∈ *S* → *S'* ∧ (∀ *x*∈*S*. ∀ *ε*>*0*. ∃ *δ*>*0*. ∀ *y*∈*S*. *d x y* < *δ* ⟶ *d'* (*f x*) (*f y*) < *ε*)
proof *safe*
show ⋀*x*. *continuous-map m1.mtopology m2.mtopology f* ⟹ *x* ∈ *S* ⟹ *f x* ∈ *S'*
using *m1.mtopology-topspace m2.mtopology-topspace* **by**(*auto dest*: *continuous-map-image-subset-topspace*)
next
fix *x ε*
assume *continuous-map m1.mtopology m2.mtopology f*
x ∈ *S* (*0* :: *real*) < *ε*
then have *openin m1.mtopology* {*z* ∈ *S*. *f z* ∈ *m2.open-ball* (*f x*) *ε*} *f x* ∈ *S'*
using *openin-continuous-map-preimage*[*OF* ‹*continuous-map m1.mtopology m2.mtopology f*›] *m2.mtopology-open-ball-in*[*of f x*,*OF* - ‹*0* < *ε*›] *continuous-map-image-subset-topspace*[*OF* ‹*continuous-map m1.mtopology m2.mtopology f*›] *m1.mtopology-topspace m2.mtopology-topspace*
by *auto*
moreover have *x* ∈ {*z* ∈ *S*. *f z* ∈ *m2.open-ball* (*f x*) *ε*}
using ‹*x* ∈ *S*› ‹*0* < *ε*› *continuous-map-image-subset-topspace*[*OF* ‹*continuous-map m1.mtopology m2.mtopology f*›] *m1.mtopology-topspace m2.mtopology-topspace m2.dist-0*[*of f x f x*]
by(*auto simp*: *m2.open-ball-def*)
ultimately obtain *δ* **where**
δ>*0 m1.open-ball x δ* ⊆ {*z* ∈ *S*. *f z* ∈ *m2.open-ball* (*f x*) *ε*}
by (*auto simp*: *m1.mtopology-openin-iff*)
thus ∃ *δ*>*0*. ∀ *y*∈*S*. *d x y* < *δ* ⟶ *d'* (*f x*) (*f y*) < *ε*

using ‹$x \in S$› ‹$f\,x \in S'$›› **by**(*auto intro*!: *exI*[**where** $x{=}\delta$] *simp*: *m1.open-ball-def m2.open-ball-def*)
next
assume $f \in S \to S'$
and $h{:}\forall\, x{\in}S.\ \forall\, \varepsilon{>}0.\ \exists\, \delta{>}0.\ \forall\, y{\in}S.\ d\ x\ y < \delta \longrightarrow d'\ (f\ x)\ (f\ y) < \varepsilon$
show *continuous-map m1.mtopology m2.mtopology f*
unfolding *continuous-map*
proof *safe*
show $\bigwedge x.\ x \in$ *topspace m1.mtopology* $\Longrightarrow f\,x \in$ *topspace m2.mtopology*
using ‹$f \in S \to S'$› *m1.mtopology-topspace m2.mtopology-topspace* **by** *auto*
next
fix *U*
assume *openin m2.mtopology U*
show *openin m1.mtopology* $\{x \in$ *topspace m1.mtopology*. $f\,x \in U\}$
unfolding *m1.mtopology-openin-iff*
proof *safe*
show $\bigwedge x.\ x \in$ *topspace m1.mtopology* $\Longrightarrow f\,x \in U \Longrightarrow x \in S$
using ‹$f \in S \to S'$› *m1.mtopology-topspace m2.mtopology-topspace* **by** *auto*
next
fix x
assume $x \in$ *topspace m1.mtopology* $f\,x \in U$
then obtain ε **where** *he*:
$\varepsilon > 0$ *m2.open-ball* $(f\,x)\ \varepsilon \subseteq U$
using ‹*openin m2.mtopology U*› **by**(*auto simp*: *m2.mtopology-openin-iff*)
then obtain δ **where** *hd*:
$\delta > 0\ \bigwedge y.\ y \in S \Longrightarrow d\ x\ y < \delta \Longrightarrow d'\ (f\,x)\ (f\,y) < \varepsilon$
using ‹$x \in$ *topspace m1.mtopology*› *m1.mtopology-topspace h* **by** *metis*
thus $\exists\, \varepsilon{>}0.$ *m1.open-ball* $x\ \varepsilon \subseteq \{x \in$ *topspace m1.mtopology*. $f\,x \in U\}$
using *m1.open-ballD m1.open-ballD' m1.mtopology-topspace he*(2) ‹$f \in S \to S'$›
by(*auto intro*!: *exI*[**where** $x{=}\delta$] *simp*: *m2.open-ball-def*) *fastforce*
qed
qed
qed

lemma *metric-set-continuous-map-eq'*:
shows *continuous-map m1.mtopology m2.mtopology f*
$\longleftrightarrow f \in S \to S' \wedge (\forall\, x\ z.$ *m1.converge-to-inS* $x\ z \longrightarrow$ *m2.converge-to-inS* $(\lambda n.\ f\ (x\ n))\ (f\ z))$
proof
show *continuous-map m1.mtopology m2.mtopology* $f \Longrightarrow f \in S \to S' \wedge (\forall\, x\ z.$ *m1.converge-to-inS* $x\ z \longrightarrow$ *m2.converge-to-inS* $(\lambda n.\ f\ (x\ n))\ (f\ z))$
unfolding *metric-set-continuous-map-eq*
proof *safe*
fix $x\ z$
assume $h{:}f \in S \to S' \,\forall\, x{\in}S.\ \forall\, \varepsilon{>}0.\ \exists\, \delta{>}0.\ \forall\, y{\in}S.\ d\ x\ y < \delta \longrightarrow d'\ (f\,x)\ (f\ y) < \varepsilon$ *m1.converge-to-inS* $x\ z$
hence $h'{:}x \in$ *m1.sequence* $z \in S\ \bigwedge \varepsilon.\ \varepsilon > 0 \Longrightarrow \exists\, N.\ \forall\, n{\geq}N.\ d\ (x\ n)\ z < \varepsilon$
by(*auto simp*: *m1.converge-to-inS-def2*)

```
    show m2.converge-to-inS (λn. f (x n)) (f z)
      unfolding m2.converge-to-inS-def2
    proof safe
      show f (x n) ∈ S' f z ∈ S' for n
        using h'(1,2) h(1) by auto
    next
      fix ε
      assume he:(0 :: real) < ε
      then obtain δ where hd:δ > 0 ⋀y. y ∈ S ⟹ d z y < δ ⟹ d' (f z) (f y)
< ε
        using h(2) h'(2) by metis
      obtain N where hn: ⋀n. n ≥ N ⟹ d z (x n) < δ
        using h'(3)[OF hd(1),simplified m1.dist-sym[of - z]] by auto
      show ∃ N. ∀ n≥N. d' (f (x n)) (f z) < ε
      proof(safe intro!: exI[where x=N])
        fix n
        assume n ≥ N
        then have x n ∈ S d z (x n) < δ
          using hn[OF ‹n ≥ N›] h'(1) by auto
        thus d' (f (x n)) (f z) < ε
          by(auto intro!: hd(2) simp: m2.dist-sym[of - f z])
      qed
    qed
  qed
next
  assume f ∈ S → S' ∧ (∀ x z. m1.converge-to-inS x z ⟶ m2.converge-to-inS
(λn. f (x n)) (f z))
  hence h:f ∈ S → S' ⋀x z. m1.converge-to-inS x z ⟹ m2.converge-to-inS (λn.
f (x n)) (f z) by auto
  show continuous-map m1.mtopology m2.mtopology f
    unfolding continuous-map-closedin
  proof safe
    show x ∈ topspace m1.mtopology ⟹ f x ∈ topspace m2.mtopology for x
      using m1.mtopology-topspace m2.mtopology-topspace h(1) by auto
  next
    fix C
    assume hcl:closedin m2.mtopology C
    show closedin m1.mtopology {x ∈ topspace m1.mtopology. f x ∈ C}
      unfolding m1.mtopology-closedin-iff
    proof safe
      fix y s
    assume hg:y ∈ UNIV → {x ∈ topspace m1.mtopology. f x ∈ C}  m1.converge-to-inS
y s
      hence (λn. f (y n)) ∈ UNIV → C
        by auto
      thus f s ∈ C s ∈ topspace m1.mtopology
     using h(2)[OF hg(2)] hcl[simplified m2.mtopology-closedin-iff] hg(2)[simplified
m1.converge-to-inS-def] m1.mtopology-topspace
        by auto
```

qed(*simp add*: *m1.mtopology-topspace*)
qed
qed

lemma *continuous-map-limit-of*:
assumes *continuous-map m1.mtopology m2.mtopology f m1.convergent-inS xn*
shows *m2.the-limit-of* (λ*n*. *f* (*xn n*)) = *f* (*m1.the-limit-of xn*)
using *assms m1.the-limit-if-converge m2.the-limit-of-eq*
by(*simp add*: *metric-set-continuous-map-eq'*)

Uniform continuous functions.

definition *uniform-continuous-map* :: ('*a* ⇒ '*b*) ⇒ *bool* **where**
uniform-continuous-map f ⟷ *f* ∈ *S* → *S'* ∧ (∀ ε>0. ∃ δ>0. ∀ *x*∈*S*. ∀ *y*∈*S*. *d x y* < δ ⟶ *d'* (*f x*) (*f y*) < ε)

lemma *uniform-continuous-map-const*:
assumes *y* ∈ *S'*
shows *uniform-continuous-map* (λ*x*. *y*)
using *assms* **by**(*auto simp*: *uniform-continuous-map-def*)

lemma *continuous-if-uniform-continuous*:
assumes *uniform-continuous-map f*
shows *continuous-map m1.mtopology m2.mtopology f*
unfolding *metric-set-continuous-map-eq*
proof *safe*
show *x* ∈ *S* ⟹ *f x* ∈ *S'* **for** *x*
using *assms* **by**(*auto simp*: *uniform-continuous-map-def*)
next
fix *x* ε
assume [*simp*]:*x* ∈ *S* **and** (*0* :: *real*) < ε
then obtain δ **where** δ > *0* ⋀*x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *d x y* < δ ⟹ *d'* (*f x*) (*f y*) < ε
using *assms* **by**(*auto simp*: *uniform-continuous-map-def*)
thus ∃ δ>*0*. ∀ *y*∈*S*. *d x y* < δ ⟶ *d'* (*f x*) (*f y*) < ε
by(*auto intro*!: *exI*[**where** *x*=δ])
qed

definition *converges-uniformly* :: [*nat* ⇒ '*a* ⇒ '*b*, '*a* ⇒ '*b*] ⇒ *bool* **where**
converges-uniformly fn f ⟷ (∀ *n*. *fn n* ∈ *S* → *S'*) ∧ (*f* ∈ *S* → *S'*) ∧ (∀ *e*>*0*. ∃ *N*. ∀ *n*≥*N*. ∀ *x*∈*S*. *d'* (*fn n x*) (*f x*) < *e*)

lemma *converges-uniformly-continuous*:
assumes ⋀*n*. *continuous-map m1.mtopology m2.mtopology* (*fn n*)
and *converges-uniformly fn f*
shows *continuous-map m1.mtopology m2.mtopology f*
unfolding *metric-set-continuous-map-eq*
proof *safe*
fix *x e*
assume *h*:*x* ∈ *S e* > (*0* :: *real*)

then obtain N **where** N: $\bigwedge z\ n.\ n \geq N \Longrightarrow z \in S \Longrightarrow d'\ (fn\ n\ z)\ (f\ z) < e\ /\ 3$
using *assms*(2) **by**(*simp only*: *converges-uniformly-def*) (*meson zero-less-divide-iff zero-less-numeral*)
have f: $\bigwedge n\ x.\ x \in S \Longrightarrow fn\ n\ x \in S' \bigwedge x.\ x \in S \Longrightarrow f\ x \in S'$
using *assms*(2) **by**(*auto simp*: *converges-uniformly-def*)
from *assms*(1)[*of N*] *h* **obtain** δ **where** h': $\delta > 0\ \bigwedge y.\ y \in S \Longrightarrow d\ x\ y < \delta \Longrightarrow d'\ (fn\ N\ x)\ (fn\ N\ y) < e\ /\ 3$
by (*metis metric-set-continuous-map-eq zero-less-divide-iff zero-less-numeral*)
show $\exists \delta{>}0.\ \forall y{\in}S.\ d\ x\ y < \delta \longrightarrow d'\ (f\ x)\ (f\ y) < e$
proof(*safe intro*!: *exI*[**where** $x{=}\delta$])
fix y
assume $y{:}y \in S\ d\ x\ y < \delta$
have $d'\ (f\ x)\ (f\ y) \leq d'\ (f\ x)\ (fn\ N\ x) + d'\ (fn\ N\ x)\ (fn\ N\ y) + d'\ (fn\ N\ y)\ (f\ y)$
using *m2.dist-tr*[*of f x fn N x f y*] *m2.dist-tr*[*of fn N x fn N y f y*] *f*[*OF y*(1)] *f*[*OF h*(1)]
by *auto*
also have ... $< e$
using *N*[*OF order-refl h*(1),*simplified m2.dist-sym*] *N*[*OF order-refl y*(1)] *h'*(2)[*OF y*]
by *auto*
finally show $d'\ (f\ x)\ (f\ y) < e$ **.**
qed(*use h' in auto*)
qed(*use assms*(2) *converges-uniformly-def* **in** *auto*)

Lemma related *osc-on*.

lemma *osc-on-inA-0*:
assumes $x \in A \cap S$ *continuous-map* (*subtopology m1.mtopology* $(A \cap S)$) *m2.mtopology f*
shows *m2.osc-on A m1.mtopology f x* = *0*
proof −
interpret *subm1*: *metric-set* $A \cap S$ *submetric* $(A \cap S)$ d
by(*auto intro*!: *m1.submetric-metric-set*)
have *cont*: *continuous-map subm1.mtopology m2.mtopology f*
using *assms*(2) **by** (*simp add*: *m1.submetric-subtopology*)
have *m2.osc-on A m1.mtopology f x* < *ennreal* ε **if** $e{:}\varepsilon > 0$ **for** ε
unfolding *m2.osc-on-less-iff*
proof −
obtain ε' **where** $\varepsilon' > 0\ 2{*}\varepsilon' < \varepsilon$
using *e field-lbound-gt-zero*[*of* $\varepsilon/2$ $\varepsilon/2$] **by** *auto*
then obtain δ **where** $hd{:}\delta{>}0\ \bigwedge y.\ y \in A \Longrightarrow y{\in}S \Longrightarrow d\ x\ y < \delta \Longrightarrow d'\ (f\ x)\ (f\ y) < \varepsilon'$
using *assms*(1) *cont*[*simplified Set-Based-Metric-Space.metric-set-continuous-map-eq*[*OF subm1.metric-set-axioms m2.metric-set-axioms*]]
by(*fastforce simp*: *submetric-def*)
show $\exists v.\ x \in v \wedge$ *openin m1.mtopology v* $\wedge$ *m2.diam* $(f\ `\ (A \cap v)) <$ *ennreal* ε
proof(*safe intro*!: *exI*[**where** $x{=}$*m1.open-ball x* δ] *m1.open-ball-ina m1.mtopology-open-ball-in*)
have *m2.diam* $(f\ `\ (A \cap$ *m1.open-ball x* $\delta)) \leq$ *ennreal* $(2{*}\varepsilon')$

unfolding *m2.diam-def Sup-le-iff*
proof *safe*
fix *a1 a2*
assume *h*:*a1* ∈ *A a1* ∈ *m1.open-ball x δ f a1* ∈ *S′*
a2 ∈ *A a2* ∈ *m1.open-ball x δ f a2* ∈ *S′*
have *f x* ∈ *S′*
using *cont assms*(*1*) **by**(*auto simp*: *Set-Based-Metric-Space.metric-set-continuous-map-eq*[*OF subm1.metric-set-axioms m2.metric-set-axioms*])
have *d′* (*f a1*) (*f a2*) < *2*∗*ε′*
using *hd*(*2*)[*OF* ‹*a1* ∈ *A*› *m1.open-ballD′*(*1*)[*OF h*(*2*)] *m1.open-ballD*[*OF h*(*2*)]] *hd*(*2*)[*OF* ‹*a2* ∈ *A*› *m1.open-ballD′*(*1*)[*OF h*(*5*)] *m1.open-ballD*[*OF h*(*5*)]] *m2.dist-tr*[*OF* ‹*f a1* ∈ *S′*› ‹*f x* ∈ *S′*› ‹*f a2* ∈ *S′*›,*simplified m2.dist-sym*[*of f a1 f x*]]
by *auto*
thus *ennreal* (*d′* (*f a1*) (*f a2*)) ≤ *ennreal* (*2*∗*ε′*)
by (*simp add*: *ennreal-leI*)
qed
also have *...* < *ennreal ε*
using ‹*2*∗*ε′* < *ε*› *ennreal-lessI e* **by** *presburger*
finally show *m2.diam* (*f* ‘ (*A* ∩ *m1.open-ball x δ*)) < *ennreal ε* **.**
qed(*use hd*(*1*) *IntD2*[*OF assms*(*1*)] **in** *auto*)
qed
hence *m2.osc-on A m1.mtopology f x* < *ε* **if** *ε* > *0* **for** *ε*
by (*metis ennreal-enn2real ennreal-le-epsilon ennreal-less-zero-iff linorder-not-le order-le-less-trans that*)
thus *?thesis*
by *fastforce*
qed

end

context *metric-set*
begin

interpretation *rnv*: *metric-set UNIV* :: (*′b* :: *real-normed-vector*) *set dist-typeclass*
by *simp*

lemma *dist-set-uniform-continuous*:
uniform-continuous-map S dist UNIV dist-typeclass (*dist-set A*)
unfolding *uniform-continuous-map-def*[*OF metric-set-axioms rnv.metric-set-axioms*] *dist-real-def*
proof *safe*
fix *ε* :: *real*
assume *0* < *ε*
then show ∃ *δ*>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < *δ* ⟶ |*dist-set A x* − *dist-set A y*| < *ε*
using *order.strict-trans1*[*OF dist-set-abs-le*] **by**(*auto intro*!: *exI*[**where** *x*=*ε*])
qed *simp*

lemma *dist-set-continuous*: *continuous-map mtopology euclideanreal* (*dist-set A*)
unfolding *euclidean-mtopology*[*symmetric*]
by(*auto intro*!: *continuous-if-uniform-continuous simp*: *dist-set-uniform-continuous metric-set-axioms*)

lemma *uniform-continuous-map-add*:
fixes *f* :: *'a* ⇒ *'b*::*real-normed-vector*
assumes *uniform-continuous-map S dist UNIV dist-typeclass f uniform-continuous-map S dist UNIV dist-typeclass g*
shows *uniform-continuous-map S dist UNIV dist-typeclass* (λ*x*. *f x* + *g x*)
unfolding *uniform-continuous-map-def*[*OF metric-set-axioms rnv.metric-set-axioms*]
proof *safe*
fix *e* :: *real*
assume *e* > *0*
from *half-gt-zero*[*OF this*] *assms* **obtain** *d1 d2* **where** *d*: *d1* > *0 d2* > *0*
⋀*x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *dist x y* < *d1* ⟹ *dist-typeclass* (*f x*) (*f y*) < *e* / *2*
⋀*x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *dist x y* < *d2* ⟹ *dist-typeclass* (*g x*) (*g y*) < *e* / *2*
by(*simp only*: *uniform-continuous-map-def*[*OF metric-set-axioms rnv.metric-set-axioms*]) *metis*
show ∃ *δ*>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < *δ* ⟶ *dist-typeclass* (*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 *uniform-continuous-map-real-devide*:
fixes *f* :: *'a* ⇒ *real*
assumes *uniform-continuous-map S dist UNIV dist-typeclass f uniform-continuous-map S dist UNIV dist-typeclass g*
and ⋀*x*. *x* ∈ *S* ⟹ *g x* ≠ *0* ⋀*x*. *x* ∈ *S* ⟹ |*g x*| ≥ *a a* > *0* ⋀*x*. *x* ∈ *S* ⟹ |*g x*| < *Kg*
and ⋀*x*. *x* ∈ *S* ⟹ |*f x*| < *Kf*
shows *uniform-continuous-map S dist UNIV dist-typeclass* (λ*x*. *f x* / *g x*)
unfolding *uniform-continuous-map-def*[*OF metric-set-axioms rnv.metric-set-axioms*]
proof *safe*
fix *e* :: *real*
assume *e*[*arith*]:*e* > *0*
consider *S* = {} | *S* ≠ {} **by** *auto*
then show ∃ *δ*>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < *δ* ⟶ *dist-typeclass* (*f x* / *g x*) (*f y* / *g y*) < *e*
proof *cases*
case *1*
then show *?thesis* **by** (*auto intro*!: *exI*[**where** *x*=*1*])
next
case *S*:*2*
then have *Kfg-pos*[*arith*]: *Kg* > *0 Kf* ≥ *0*
using *assms*(*4*−*7*) **by** *auto fastforce*+
define *e'* **where** *e'* ≡ *a*^*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
    ⋀x y. x ∈ S ⟹ y ∈ S ⟹ dist x y < d1 ⟹ |f x − f y| < e'    ⋀x y. x ∈ S
⟹ y ∈ S ⟹ dist x y < d2 ⟹ |g x − g y| < e'
    by(auto simp: uniform-continuous-map-def[OF metric-set-axioms rnv.metric-set-axioms]
dist-real-def) metis
    show ?thesis
      unfolding dist-real-def
    proof(safe intro!: exI[where x=min d1 d2])
      fix x y
      assume x:x ∈ S and y:y ∈ S and dist x y < min d1 d2
      then have dist[arith]: dist x y < d1 dist 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
qed simp

lemma the-limit-of-dist-converge:
  assumes converge-to-inS xn x
  shows (λn. dist (xn n) y) ⟶ dist (the-limit-of xn) y
proof −
  have continuous-map mtopology euclideanreal (λz. dist z y)
    using dist-set-continuous[of {y}] by simp
  hence (λn. dist (xn n) y) ⟶ dist x y
    using assms
   by(auto simp: metric-set-continuous-map-eq'[OF metric-set-axioms rnv.metric-set-axioms,simplified
euclidean-mtopology] converge-to-def-set)
  thus ?thesis
    by(simp add: the-limit-of-eq[OF assms])
```

qed

lemma *the-limit-of-dist-converge'*:
assumes *converge-to-inS xn x* $\varepsilon > 0$
shows $\exists N.\ \forall n{\geq}N.\ |\ dist\ (xn\ n)\ y - dist\ (the\text{-}limit\text{-}of\ xn)\ y\ | < \varepsilon$
using *the-limit-of-dist-converge*[*OF assms*(*1*)] *assms*(*2*) **by**(*simp add*: *LIMSEQ-iff*)

lemma *the-limit-of-dist*:
assumes *converge-to-inS xn x*
shows *lim* $(\lambda n.\ dist\ (xn\ n)\ y) = dist\ (the\text{-}limit\text{-}of\ xn)\ y$
using *the-limit-of-dist-converge*[*OF assms*] *limI* **by** *blast*

Upper-semicontinuous functions.

lemma *upper-semicontinuous-map-def2*:
fixes $f :: 'a \Rightarrow ('b :: \{complete\text{-}linorder, linorder\text{-}topology\})$
shows *upper-semicontinuous-map mtopology f* $\longleftrightarrow$ $(\forall x\ y.$ *converge-to-inS x y* $\longrightarrow$ *limsup* $(\lambda n.\ f\ (x\ n)) \leq f\ y)$
proof
show *upper-semicontinuous-map mtopology f* $\Longrightarrow \forall x\ y.$ *converge-to-inS x y* $\longrightarrow$ *limsup* $(\lambda n.\ f\ (x\ n)) \leq f\ y$
unfolding *upper-semicontinuous-map-def*
proof *safe*
fix *x y*
assume *h*:$\forall a.$ *openin mtopology* $\{x \in$ *topspace mtopology*. $f\ x < a\}$ *converge-to-inS x y*
show *limsup* $(\lambda n.\ f\ (x\ n)) \leq f\ y$
unfolding *Limsup-le-iff eventually-sequentially*
proof *safe*
fix *c*
assume $f\ y < c$
show $\exists N.\ \forall n{\geq}N.\ f\ (x\ n) < c$
proof(*rule ccontr*)
assume $\nexists N.\ \forall n{\geq}N.\ f\ (x\ n) < c$
then have *hc*:$\bigwedge N.\ \exists n{\geq}N.\ f\ (x\ n) \geq c$
using *linorder-not-less* **by** *blast*
define *a* :: *nat* $\Rightarrow$ *nat* **where** *a* $\equiv$ *rec-nat* (*SOME n*. $f\ (x\ n) \geq c$) ($\lambda n\ an.$ *SOME m*. $m > an \wedge f\ (x\ m) \geq c$)
have *strict-mono a*
proof(*rule strict-monoI-Suc*)
fix *n*
have [*simp*]:*a* (*Suc n*) = (*SOME m*. $m > a\ n \wedge f\ (x\ m) \geq c$)
by(*auto simp*: *a-def*)
show $a\ n < a\ (Suc\ n)$
by *simp* (*metis* (*mono-tags*, *lifting*) *Suc-le-lessD hc someI*)
qed
have $*$:$f\ (x\ (a\ n)) \geq c$ **for** *n*
proof(*cases n*)
case *0*
then show *?thesis*

```
        using hc[of 0] by(auto simp: a-def intro!: someI-ex)
      next
        case (Suc n')
        then show ?thesis
         by(simp add: a-def) (metis (mono-tags, lifting) Suc-le-lessD hc someI-ex)
      qed
      obtain N where ⋀n. n ≥ N ⟹ x (a n) ∈ {x ∈ S. f x < c}
     using converge-to-inS-subseq[OF ‹strict-mono a› h(2)] mtopology-openin-iff2[of
{x ∈ S. f x < c}] h(2)[simplified converge-to-inS-def] mtopology-topspace ‹f y < c›
h
        by fastforce
      from ∗[of N] this[of N] show False by auto
    qed
  qed
 qed
next
  assume h:∀ x y. converge-to-inS x y ⟶ limsup (λn. f (x n)) ≤ f y
  show upper-semicontinuous-map mtopology f
   unfolding upper-semicontinuous-map-def mtopology-openin-iff2 mtopology-topspace
  proof safe
    fix a y s
    assume converge-to-inS y s s ∈ S f s < a
    then have limsup (λn. f (y n)) ≤ f s
      using h by auto
    with ‹f s < a› obtain N where ⋀n. n≥N ⟹ f (y n) < a
      by(auto simp: Limsup-le-iff eventually-sequentially)
    thus ∃ N. ∀ n≥N. y n ∈ {x ∈ S. f x < a}
       using ‹converge-to-inS y s› by(auto intro!: exI[where x=N] simp: con-
verge-to-inS-def)
  qed
qed

lemma upper-semicontinuous-map-def2real:
  fixes f :: 'a ⇒ real
  shows upper-semicontinuous-map mtopology f ⟷ (∀ x y. converge-to-inS x y
⟶ limsup (λn. f (x n)) ≤ f y)
  unfolding upper-semicontinuous-map-real-iff upper-semicontinuous-map-def2
  by auto

lemma osc-upper-semicontinuous-map:
 upper-semicontinuous-map X (osc X f)
proof −
  have {x ∈ topspace X. osc X f x < a} = ⋃ {V. openin X V ∧ diam (f ‘ V) <
a} for a
   using openin-subset by(auto simp add: osc-less-iff)
  thus ?thesis
    by(auto simp: upper-semicontinuous-map-def)
qed
```

end

Open maps.

lemma *metric-set-opem-map-from-dist*:
assumes *metric-set S d metric-set S′ d′ f* ∈ *S* → *S′*
and ⋀*x ε*. *x* ∈ *S* ⟹ *ε* > *0* ⟹ ∃*δ*>*0*. ∀*y*∈*S*. *d′* (*f x*) (*f y*) < *δ* ⟶ *d x y* < *ε*
shows *open-map* (*metric-set.mtopology S d*) (*subtopology* (*metric-set.mtopology S′ d′*) (*f* ‘ *S*)) *f*
proof −
interpret *m1*: *metric-set S d* **by** *fact*
interpret *m2*: *metric-set S′ d′* **by** *fact*
interpret *m2′*: *metric-set f* ‘ *S submetric* (*f* ‘ *S*) *d′*
using *assms*(*3*) **by**(*auto intro*!: *m2.submetric-metric-set*)
show *?thesis*
proof(*rule open-map-with-base*[*OF m1.mtopology-basis-ball*])
fix *A*
assume *A* ∈ {*m1.open-ball a ε* |*a ε*. *a* ∈ *S* ∧ *0* < *ε*}
then obtain *a ε* **where** *hae*:
a ∈ *S 0* < *ε A* = *m1.open-ball a ε* **by** *auto*
show *openin* (*subtopology m2.mtopology* (*f* ‘ *S*)) (*f* ‘ *A*)
unfolding *m2.submetric-subtopology*[*OF funcset-image*[*OF assms*(*3*)]] *m2′.mtopology-openin-iff*
proof
show *f* ‘ *A* ⊆ *f* ‘ *S*
using *m1.open-ball-subset-ofS*[*of a ε*] **by**(*auto simp*: *hae*(*3*))
next
show ∀*x*∈*f* ‘ *A*. ∃*ε*>*0*. *m2′.open-ball x ε* ⊆ *f* ‘ *A*
proof *safe*
fix *x*
assume *x* ∈ *A*
hence *x* ∈ *S*
using *m1.open-ball-subset-ofS*[*of a ε*] **by**(*auto simp*: *hae*(*3*))
moreover have *0* < *ε* − *d a x*
using ‹*x* ∈ *A*› *m1.open-ballD*[*of x a ε*] **by**(*auto simp*: *hae*(*3*))
ultimately obtain *δ* **where** *hd*:*δ* > *0* ⋀*y*. *y*∈*S* ⟹ *d′* (*f x*) (*f y*) < *δ* ⟹ *d x y* < *ε* − *d a x*
using *assms*(*4*) **by** *metis*
show ∃*ε*>*0*. *m2′.open-ball* (*f x*) *ε* ⊆ *f* ‘ *A*
proof(*safe intro*!: *exI*[**where** *x*=*δ*])
fix *z*
assume *z* ∈ *m2′.open-ball* (*f x*) *δ*
note *hz* = *m2′.open-ballD′*[*OF this*]
then obtain *y* **where** *y* ∈ *S z* = *f y* **by** *auto*
hence *d′* (*f x*) (*f y*) < *δ*
using *m2′.open-ballD*[*OF* ‹*z* ∈ *m2′.open-ball* (*f x*) *δ*›] ‹*x* ∈ *A*› *m1.open-ball-subset-ofS*[*of a ε*]
by(*auto simp*: *submetric-def hae*(*3*))
hence *d x y* < *ε* − *d a x*
by(*auto intro*!: *hd*(*2*)[*OF* ‹*y* ∈ *S*›])

hence $d\ a\ y < \varepsilon$
using *m1.dist-tr*[*OF* ‹$a \in S$› ‹$x \in S$› ‹$y \in S$›] **by** *auto*
thus $z \in f\ `\ A$
by (*simp add*: ‹$y \in S$› ‹$z = f\ y$› *hae*(*1*) *hae*(*3*) *m1.open-ball-def*)
qed(*use hd* **in** *auto*)
qed
qed
qed
qed

2.1.2 Complete Metric Spaces

locale *complete-metric-set* = *metric-set* +
assumes *convergence*: $\bigwedge f$. *Cauchy-inS f* $\Longrightarrow$ *convergent-inS f*

lemma *complete-space-complete-metric-set*:
complete-metric-set (*UNIV* :: $'a$ :: *complete-space set*) *dist*
proof −
interpret *m*: *metric-set UNIV dist*
by(*rule metric-class-metric-set*)
show *?thesis*
by *standard* (*simp add*: *Cahuchy-def-set*[*symmetric*] *convergent-def-set*[*symmetric*] *Cauchy-convergent-iff*)
qed

lemma(**in** *complete-metric-set*) *submetric-complete-iff*:
assumes $M \subseteq S$
shows *complete-metric-set M* (*submetric M dist*) $\longleftrightarrow$ *closedin mtopology M*
proof
assume *complete-metric-set M* (*submetric M dist*)
then interpret *m*: *complete-metric-set M submetric M dist* **.**
show *closedin mtopology M*
proof(*rule ccontr*)
assume ¬ *closedin mtopology M*
then have $\exists f{\in}$*m.sequence*. $\exists s$. *converge-to-inS f s* $\wedge$ $s \notin M$
using *assms mtopology-closedin-iff* **by** *auto*
then obtain *f s* **where** *hfs*:$f \in$ *m.sequence converge-to-inS f s* $s \notin M$
by *auto*
hence *convergent-inS f*
by(*auto simp*: *convergent-inS-def converge-to-inS-def*)
have *m.Cauchy-inS f*
using *Cauchy-if-convergent-inS*[*OF* ‹*convergent-inS f*›] *hfs*(*1*)
by(*auto simp*: *m.Cauchy-inS-def Cauchy-inS-def*) (*fastforce simp*: *submetric-def*)
then obtain s' **where** $s' \in M$ *m.converge-to-inS f* s'
using *m.convergence* **by**(*auto simp*: *m.convergent-inS-def m.converge-to-inS-def*)
from *converge-to-insub-converge-to-inS*[*OF assms this*(*2*)] *hfs*(*2*)
have $s' = s$
by(*rule converge-to-inS-unique*)

```
    thus False
      using ‹s' ∈ M› ‹s ∉ M› by simp
  qed
next
  interpret m: metric-set M submetric M dist
    by(rule submetric-metric-set[OF assms])
  assume cls:closedin mtopology M
  show complete-metric-set M (submetric M dist)
  proof
    fix f
    assume m.Cauchy-inS f
    then have f ∈ m.sequence by(simp add: m.Cauchy-inS-def)
    have Cauchy-inS f
      by(rule Cauchy-insub-Cauchy[OF assms ‹m.Cauchy-inS f›])
    then obtain x where hx:x ∈ S converge-to-inS f x
      using convergence by(auto simp: convergent-inS-def converge-to-inS-def)
    hence x ∈ M
      using cls[simplified mtopology-closedin-iff] ‹f ∈ m.sequence› assms
      by auto
    hence m.converge-to-inS f x
      using convergent-insubmetric[OF assms ‹f ∈ m.sequence›] hx by auto
    thus m.convergent-inS f
      using ‹x ∈ M› by(auto simp: m.convergent-inS-def)
  qed
qed

lemma(in complete-metric-set) embed-dist-complete:
  assumes f ∈ B → S inj-on f B closedin mtopology (f ‘ B)
  shows complete-metric-set B (embed-dist-on B f)
proof −
  interpret m: metric-set B embed-dist-on B f
    by(rule embed-dist-dist[OF assms(1,2)])
  show ?thesis
  proof
    fix xn
    assume m.Cauchy-inS xn
    hence h:xn ∈ m.sequence Cauchy-inS (λn. f (xn n))
      by(auto simp add: embed-dist-Cauchy-inS-iff[OF assms(1,2)])
    with convergence obtain x where x: converge-to-inS (λn. f (xn n)) x
      by(auto simp: convergent-inS-def)
    have x': x ∈ f ‘ B
    proof −
      have (λn. f (xn n)) ∈ UNIV → f ‘ B
        using assms(1) h(1) by auto
      thus ?thesis
        using assms(3) x by(auto simp: mtopology-closedin-iff)
    qed
    then obtain b where b: b ∈ B f b = x by auto
    show m.convergent-inS xn
```

by(*auto simp*: *m.convergent-inS-def embed-dist-converge-to-inS-iff*[*OF assms*(*1*,*2*)]
b x h intro!: *exI*[**where** *x*=*b*])
qed
qed

lemma(**in** *metric-set*) *Cantor-intersection-theorem*:
complete-metric-set S dist ⟷ (∀ *Fn*. (∀ *n*. *Fn n* ≠ {}) ∧ (∀ *n*. *closedin mtopology* (*Fn n*)) ∧ *decseq Fn* ∧ (∀ ε > *0*. ∃ *N*. ∀ *n*≥*N*. *diam* (*Fn n*) < ε) ⟶ (∃ *x*∈*S*. ⋂ (*range Fn*) = {*x*}))
proof *safe*
fix *Fn*
assume *complete-metric-set S dist*
interpret *complete-metric-set S dist* **by** *fact*
assume *h*: ∀ *n*. *Fn n* ≠ {} ∀ *n*. *closedin mtopology* (*Fn n*) *decseq Fn* ∀ ε > *0*. ∃ *N*. ∀ *n*≥*N*. *diam* (*Fn n*) < ε
then obtain *xn* **where** *xn1*: ⋀*n*. *xn n* ∈ *Fn n*
by (*meson all-not-in-conv*)
hence *xn2*: *xn* ∈ *sequence*
using *closedin-subset*[*of mtopology*] *h*(*2*) **by**(*auto simp*: *mtopology-topspace*)
have *Cauchy-inS xn*
unfolding *Cauchy-inS-def*
proof *safe*
fix ε :: *real*
assume *0* < ε
then obtain *N* **where** *N*:⋀*n*. *n*≥*N* ⟹ *diam* (*Fn n*) < *ennreal* ε
using *h*(*4*) *ennreal-less-zero-iff* **by** *blast*
show ∃ *N*. ∀ *n*≥*N*. ∀ *m*≥*N*. *dist* (*xn n*) (*xn m*) < ε
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n m*
assume *n* ≥ *N m* ≥ *N*
define *nm* **where** *nm* = *min m n*
have *nm* ≥ *N nm* ≤ *n nm* ≤ *m*
using ‹*n* ≥ *N*› ‹*m* ≥ *N*› **by**(*auto simp*: *nm-def*)
hence *xn n* ∈ *Fn nm xn m* ∈ *Fn nm*
using *decseqD*[*OF h*(*3*)] *xn1*[*of n*] *xn1*[*of m*] **by** *auto*
hence *ennreal* (*dist* (*xn n*) (*xn m*)) ≤ *diam* (*Fn nm*)
using *xn2* **by**(*auto intro*!: *diam-is-sup mtopology-topspace*)
also have ... < *ennreal* ε
by(*rule N*[*OF* ‹*nm* ≥ *N*›])
finally show *dist* (*xn n*) (*xn m*) < ε
by (*simp add*: *dist-geq0 ennreal-less-iff*)
qed
qed(*use xn2* **in** *auto*)
then obtain *x* **where** *x*:*x* ∈ *S converge-to-inS xn x*
using *convergence*[*of xn*] **by**(*auto simp*: *convergent-inS-def converge-to-inS-def*)
show ∃ *x*∈*S*. ⋂ (*range Fn*) = {*x*}
proof(*safe intro*!: *bexI*[**where** *x*=*x*])
fix *n*
show *x* ∈ *Fn n*

```
    proof(rule ccontr)
      assume x ∉ Fn n
      moreover have openin mtopology (S − Fn n)
        using h(2) by (simp add: openin-diff)
      ultimately obtain ε where e: ε > 0 open-ball x ε ⊆ S − Fn n
        using x(1) by(auto simp: mtopology-openin-iff)
      then have ∃ N. ∀ n≥N. xn n ∈ open-ball x ε
         using mtopology-openin-iff2[of open-ball x ε] open-ball-ina[OF x(1) e(1)]
x(2)
        by(auto simp: openin-open-ball)
      then obtain N where N:⋀m. m ≥ N ⟹ xn m ∈ open-ball x ε
        by auto
      hence xn m ∈ S − Fn m if m ≥ N m ≥ n for m
        using e(2) decseqD[OF h(3) that(2)] using that(1) by blast
      from xn1[of max N n] this[of max N n]
      show False by auto
    qed
  next
    fix y
    assume y ∈ ⋂ (range Fn)
    then have hy:∀ n. y ∈ Fn n by auto
    have y ∈ S
       using closedin-subset[of mtopology] h(2) hy mtopology-topspace by auto
    have converge-to-inS xn y
      unfolding converge-to-inS-def2
    proof safe
      fix ε :: real
      assume 0 < ε
      then obtain N where N:⋀n. n ≥ N ⟹ diam (Fn n) < ennreal ε
        using ennreal-less-zero-iff h(4) by presburger
      show ∃ N. ∀ n≥N. dist (xn n) y < ε
      proof(safe intro!: exI[where x=N])
        fix n
        assume n ≥ N
        then have ennreal (dist (xn n) y) < ennreal ε
          using ‹y ∈ S› hy xn1[of n] xn2
           by(auto intro!: order.strict-trans1[OF diam-is-sup[of xn n Fn n y] N[of
n]])
        thus dist (xn n) y < ε
          by (simp add: dist-geq0 ennreal-less-iff)
      qed
    qed(use xn2 ‹y ∈ S› in auto)
    with converge-to-inS-unique[OF x(2)]
    show y = x by simp
  qed(use x in auto)
next
  assume h:∀ Fn. (∀ n. Fn n ≠ {}) ∧ (∀ n. closedin mtopology (Fn n)) ∧ decseq Fn
∧ (∀ ε>0. ∃ N. ∀ n≥N. diam (Fn n) < ε) ⟶ (∃ x∈S. ⋂ (range Fn) = {x})
  show complete-metric-set S dist
```

proof
fix *xn*
assume *cauchy*:*Cauchy-inS xn*
hence *xn*: *xn* ∈ *sequence*
by (*simp add*: *Cauchy-inS-dest1*)
define *Fn* **where** *Fn* ≡ (λ*n*. *mtopology closure-of* {*xn m*|*m*. *m* ≥ *n*})
have *Fn0'*: {*xn m*|*m*. *m* ≥ *n*} ⊆ *Fn n* **for** *n*
using *xn* **by**(*auto intro*!: *closure-of-subset simp*: *Fn-def mtopology-topspace*)
have *Fn0*: ⋀*n*. *Fn n* ⊆ *S*
using *xn* **by**(*auto simp*: *Fn-def in-closure-of metric-set.mtopology-topspace metric-set-axioms*)
have *Fn1*: *Fn n* ≠ {} **for** *n*
using *xn closure-of-eq-empty*[*of* {*xn m*|*m*. *m* ≥ *n*} *mtopology*,*simplified mtopology-topspace*]
by(*auto simp*: *Fn-def*)
have *Fn2*:⋀*n*. *closedin mtopology* (*Fn n*)
by(*simp add*: *Fn-def*)
have *Fn3*: *decseq Fn*
by *standard* (*auto simp*: *Fn-def intro*!: *closure-of-mono*)
have *Fn4*:∀ *ε*>*0*. ∃ *N*. ∀ *n*≥*N*. *diam* (*Fn n*) < *ε*
proof *safe*
fix *ε* :: *ennreal*
assume *0* < *ε*
define *e* **where** *e* ≡ *min 1 ε*
have *he*: *e* ≤ *ε enn2real e* > *0 ennreal* (*enn2real e*) = *e*
using ‹*0* < *ε*› **by**(*auto simp*: *e-def enn2real-positive-iff min-less-iff-disj*)
then obtain *e'* **where** *e'*:*e'* > *0 e'* < *enn2real e*
using *field-lbound-gt-zero* **by** *auto*
then obtain *N* **where** *N*:⋀*n m*. *n* ≥ *N* ⟹ *m* ≥ *N* ⟹ *dist* (*xn n*) (*xn m*) ≤ *e'*
using *cauchy* **by**(*fastforce simp*: *Cauchy-inS-def*)
show ∃ *N*. ∀ *n*≥*N*. *diam* (*Fn n*) < *ε*
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *N* ≤ *n*
then have *diam* (*Fn n*) ≤ *ennreal e'*
by(*auto intro*!: *diam-le N simp*: *Fn-def diam-eq-closure*)
also have ... < *e*
using *e'*(*2*) *ennreal-lessI he*(*2*) *he*(*3*) **by** *fastforce*
finally show *diam* (*Fn n*) < *ε*
using *he*(*1*) **by** *auto*
qed
qed
obtain *x* **where** *x*:*x*∈*S* ⋂ (*range Fn*) = {*x*}
using *h Fn1 Fn2 Fn3 Fn4* **by** *metis*
show *convergent-inS xn*
unfolding *convergent-inS-def converge-to-inS-def2*
proof(*safe intro*!: *exI*[**where** *x*=*x*])
fix *ε* :: *real*

```
    assume he:0 < ε
    then have 0 < ennreal ε by simp
    then obtain N where N: ⋀n. n ≥ N ⟹ diam (Fn n) < ennreal ε
      using Fn4 by metis
    show ∃N. ∀n≥N. dist (xn n) x < ε
    proof(safe intro!: exI[where x=N])
      fix n
      assume N ≤ n
      then have xn n ∈ Fn N x ∈ Fn N
        using x(2) Fn0'[of N] by auto
      hence ennreal (dist (xn n) x) ≤ diam (Fn N)
        using Fn0 by(auto intro!: diam-is-sup)
      also have ... <  ennreal ε
        by(auto intro!: N)
      finally show dist (xn n) x < ε
        by (simp add: dist-geq0 ennreal-less-iff)
    qed
  qed(use xn x in auto)
 qed
qed

lemma(in complete-metric-set) closed-decseq-Inter':
 assumes ⋀n. Fn n ≠ {} ⋀n. closedin mtopology (Fn n) decseq Fn
     and ⋀ε. ε > 0 ⟹ ∃N. ∀n≥N. diam (Fn n) < ε
   shows ∃x∈S. ⋂ (range Fn) = {x}
 using assms Cantor-intersection-theorem by(simp add: complete-metric-set-axioms)

lemma(in complete-metric-set) closed-decseq-Inter:
 assumes ⋀n. Fn n ≠ {} ⋀n. closedin mtopology (Fn n) decseq Fn
     and ⋀ε. ε > 0 ⟹ ∃N. ∀n≥N. diam (Fn n) < ennreal ε
   shows ∃x∈S. ⋂ (range Fn) = {x}
proof −
 have ∃N. ∀n≥N. diam (Fn n) < ε if ε > 0 for ε
 proof −
   consider ε < ∞ | ε = ∞
     using top.not-eq-extremum by fastforce
   then show ?thesis
   proof cases
     case 1
     with that have 2:ennreal (enn2real ε) = ε
       by simp
     have 0 < enn2real ε
       using 1 that by(simp add: enn2real-positive-iff)
     from assms(4)[OF this] show ?thesis
       by(simp add: 2)
   next
     case 2
     then show ?thesis
     by (metis assms(4) ennreal-less-top gt-ex infinity-ennreal-def order-less-imp-not-less
```

top.not-eq-extremum)
  qed
 qed
 thus *?thesis*
  using *closed-decseq-Inter'*[*OF assms*(*1*−*3*)] **by** *simp*
qed

2.1.3 Separable Metric Spaces

locale *separable-metric-set* = *metric-set* +
 assumes *separable*: ∃ *U*. *countable U* ∧ *dense-set U*

lemma(**in** *metric-set*) *separable-if-countable*:
 assumes *countable S*
 shows *separable-metric-set S dist*
 apply *standard*
 using *assms* **by**(*auto intro*!: *exI*[**where** *x*=*S*] *simp*: *dense-set-S*)

lemma(**in** *metric-set*) *separable-iff-topological-separable*:
 separable-metric-set S dist ⟷ *separable mtopology*
 by(*simp add*: *separable-metric-set-def separable-metric-set-axioms-def separable-def metric-set-axioms*)

lemma(**in** *separable-metric-set*) *topological-separable-if-separable*:
 separable mtopology
 using *separable-iff-topological-separable*
 by (*simp add*: *separable-metric-set-axioms*)

lemma *separable-metric-setI*:
 assumes *metric-set S d separable* (*metric-set.mtopology S d*)
 shows *separable-metric-set S d*
 by (*simp add*: *assms*(*1*) *assms*(*2*) *metric-set.separable-iff-topological-separable*)

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

lemma(**in** *metric-set*) *generated-by-countable-balls*:
 assumes *countable U* **and** *dense-set U*
 shows *mtopology* = *topology-generated-by* {*open-ball y* (*1* / *real n*) | *y n*. *y* ∈ *U*}
proof −
 have *hu*: *U* ⊆ *S* ⋀*x ε*. *x* ∈ *S* ⟹ *ε* > *0* ⟹ *open-ball x ε* ∩ *U* ≠ {}
  using *assms* **by**(*auto simp*: *dense-set-def*)
 show *?thesis*
  unfolding *mtopology-def2*
 proof(*rule topology-generated-by-eq*)
  fix *K*
  assume *K* ∈ {*open-ball y* (*1* / *real n*) |*y n*. *y* ∈ *U*}
  then obtain *y n* **where** *hyn*:
   y ∈ *U y* ∈ *S K* = *open-ball y* (*1* / *real n*)
    using *hu*(*1*) **by** *auto*
  consider *n* = *0* | *n* > *0* **by** *auto*

then show *openin* (*topology-generated-by* {*open-ball a ε* |*a ε*. *a* ∈ *S* ∧ *0* < *ε*})
K
proof *cases*
case *1*
then have *K* = {}
using *hyn dist-geq0*[*of y*] *not-less* **by**(*auto simp*: *open-ball-def*)
thus *?thesis*
by *auto*
next
case *2*
then have *1* / *real n* > *0* **by** *auto*
thus *?thesis*
using *hyn mtopology-open-ball-in*[*simplified mtopology-def2*] **by** *auto*
qed
next
have *h0*:⋀*x ε*. *x* ∈ *S* ⟹ *ε* > *0* ⟹ ∃ *y*∈*U*. ∃ *n*. *x* ∈ *open-ball y* (*1* / *real n*)
∧ *open-ball y* (*1* / *real n*) ⊆ *open-ball x ε*
proof −
fix *x ε*
assume *h*: *x* ∈ *S* (*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 *open-ball x* (*1* / (*2* ∗ *real N*)) ∩ *U* ≠ {}
using *dense-set-def*[*of U*] *assms*(*2*) *h*(*1*) *hn0* **by** *fastforce*
then obtain *y* **where** *hy*:
y∈*U y* ∈ *S y* ∈ *open-ball x* (*1* / (*real* (*2* ∗ *N*)))
using *hu* **by** *auto*
show ∃ *y*∈*U*. ∃ *n*. *x* ∈ *open-ball y* (*1* / *real n*) ∧ *open-ball y* (*1* / *real n*) ⊆
open-ball x ε
proof(*intro bexI*[**where** *x*=*y*] *exI*[**where** *x*=*2* ∗ *N*] *conjI*)
show *x* ∈ *open-ball y* (*1* / *real* (*2* ∗ *N*))
using *hy*(*3*) **by**(*simp add*: *open-ball-inverse*[*of x y*])
next
show *open-ball y* (*1* / *real* (*2* ∗ *N*)) ⊆ *open-ball x ε*
proof
fix *y'*
assume *hy'*:*y'* ∈ *open-ball y* (*1* / *real* (*2* ∗ *N*))
have *dist x y'* < *ε* (**is** *?lhs* < *?rhs*)
proof −
have *?lhs* ≤ *dist x y* + *dist y y'*
using *hy*(*2*) *open-ballD'*(*1*)[*OF hy'*] *h*(*1*) **by**(*auto intro*!: *dist-tr*)
also have ... < *1* / *real* (*2* ∗ *N*) + *1* / *real* (*2* ∗ *N*)
apply(*rule strict-ordered-ab-semigroup-add-class.add-strict-mono*)
using *hy*(*3*) *hy*(*2*) *open-ballD'*(*1*)[*OF hy'*] *h*(*1*) *hy'* **by**(*simp-all add*:
open-ball-def dist-sym[*of x y*])

```
        finally show ?thesis
          using hn' by auto
      qed
      thus y' ∈ open-ball x ε
        using open-ballD'(1)[OF hy'] h(1) by(simp add: open-ball-def)
    qed
  qed fact
 qed
 fix K
 assume hk: K ∈ {open-ball a ε |a ε. a ∈ S ∧ 0 < ε}
 then obtain x εx where hxe:
   x ∈ S 0 < εx K = open-ball x εx by auto
 have gh:K = (⋃{open-ball y (1 / real n) | y n. y ∈ U ∧ open-ball y (1 / real
n) ⊆ K})
 proof
  show K ⊆ (⋃ {open-ball y (1 / real n) |y n. y ∈ U ∧ open-ball y (1 / real
n) ⊆ K})
   proof
    fix k
    assume hkink:k ∈ K
    then have hkinS:k ∈ S
      using open-ballD'(1)[of k] by(simp add: hxe)
    obtain εk where hek:
      εk > 0 open-ball k εk ⊆ K
      using mtopology-open-ball-in'[of k x] hkink
      by(auto simp: hxe)
    obtain y n where hyey:
     y ∈ U k ∈ open-ball y (1 / real n) open-ball y (1 / real n) ⊆ open-ball k εk
      using h0[OF hkinS hek(1)] by auto
    show k ∈ ⋃ {open-ball y (1 / real n) |y n. y ∈ U ∧ open-ball y (1 / real
n) ⊆ K}
      using hek(2) hyey by blast
   qed
 qed auto
 show openin (topology-generated-by {open-ball y (1 / real n) |y n. y ∈ U}) K
  unfolding openin-topology-generated-by-iff
  apply(rule generate-topology-on.UN[of {open-ball y (1 / real n) |y n. y ∈ U
∧ open-ball y (1 / real n) ⊆ K}, simplified gh[symmetric]])
  apply(rule generate-topology-on.Basis) by auto
qed
qed

lemma(in separable-metric-set) second-countable':
∃𝒪. countable 𝒪 ∧ mtopology-basis 𝒪
proof −
 obtain U where hu:
  countable U dense-set U
   using separable by auto
 show ?thesis
```

proof(*rule countable-base-from-countable-subbase*[**where** $\mathcal{O}$={*open-ball y (1 / real n)* | *y n. y* ∈ *U*},*simplified second-countable-def*])
have {*open-ball y (1 / real n)* |*y n. y* ∈ *U*} = (λ(*y*,*n*). *open-ball y (1 / real n*)) ‘ (*U* × *UNIV*)
by *auto*
also have *countable* ...
using *hu* **by** *auto*
finally show *countable* {*open-ball y (1 / real n)* |*y n. y* ∈ *U*} .
qed (*simp add*: *generated-by-countable-balls*[*OF hu*] *subbase-of-def*)
qed

lemma(**in** *separable-metric-set*) *second-countable*: *second-countable mtopology*
by(*simp add*: *second-countable-def second-countable'*)

lemma(**in** *metric-set*) *separable-if-second-countable*:
assumes *countable* $\mathcal{O}$ **and** *mtopology-basis* $\mathcal{O}$
shows *separable-metric-set S dist*
proof
have *1*:*mtopology* = *topology-generated-by* {*U* ∈ $\mathcal{O}$. *U* ≠ {}}
by(*simp add*: *topology-generated-by-without-empty*[*symmetric*] *base-is-subbase*[*OF assms*(*2*),*simplified subbase-of-def*])
have ∀ *U* ∈ {*U* ∈$\mathcal{O}$. *U* ≠ {} }. ∃ *x*. *x* ∈ *U*
by *auto*
then have ∃ *x*. ∀ *U* ∈ { *U* ∈ $\mathcal{O}$. *U* ≠ {} }. *x U* ∈ *U*
by(*rule bchoice*)
then obtain *x* **where** *hx*:
∀ *U* ∈ { *U* ∈ $\mathcal{O}$. *U* ≠ {} }. *x U* ∈ *U*
by *auto*
show ∃ *U*. *countable U* ∧ *dense-set U*
proof(*intro exI*[**where** *x*={ *x U* | *U*. *U* ∈ $\mathcal{O}$ ∧ *U* ≠ {}}] *conjI*)
have {*x U* |*U*. *U* ∈ $\mathcal{O}$ ∧ *U* ≠ {}} = (λ*U*. *x U*) ‘ { *U* ∈ $\mathcal{O}$. *U* ≠ {} }
by *auto*
also have *countable* ...
using *assms*(*1*) **by** *auto*
finally show *countable* {*x U* |*U*. *U* ∈ $\mathcal{O}$ ∧ *U* ≠ {}} .
next
show *dense-set* {*x U* |*U*. *U* ∈ $\mathcal{O}$ ∧ *U* ≠ {}}
unfolding *dense-set-def*
proof
have ⋀*U*. *U* ∈ $\mathcal{O}$ ⟹ *U* ⊆ *topspace mtopology*
using *assms*(*2*)[*simplified base-of-def2'*]
by(*auto intro*!: *openin-subset*)
then show {*x U* |*U*. *U* ∈ $\mathcal{O}$ ∧ *U* ≠ {}} ⊆ *S*
using *hx* **by**(*auto simp add*: *mtopology-topspace*)
next
show ∀ *xa*∈*S*. ∀ *ε*>*0*. *open-ball xa ε* ∩ {*x U* |*U*. *U* ∈ $\mathcal{O}$ ∧ *U* ≠ {}} ≠ {}
proof *safe*
fix *s ε*
assume *h*:*s* ∈ *S* (*0*::*real*) < *ε open-ball s ε* ∩ {*x U* |*U*. *U* ∈ $\mathcal{O}$ ∧ *U* ≠ {}}

$= \{\}$
then have *openin mtopology* (*open-ball s* ε)
by(*auto intro*!: *mtopology-open-ball-in*)
moreover have *open-ball s* $\varepsilon \neq \{\}$
using *h open-ball-ina* **by** *blast*
ultimately obtain U' **where**
$U' \in \mathcal{O}$ $U' \neq \{\}$ $U' \subseteq$ *open-ball s* ε
using *assms*(*2*)[*simplified base-of-def*] **by** *fastforce*
then have *x* $U' \in$ *open-ball s* $\varepsilon \cap \{x\ U\ |U.\ U \in \mathcal{O} \wedge U \neq \{\}\}$
using *hx* **by** *blast*
with *h* **show** *False*
by *auto*
qed
qed
qed
qed

lemma *metric-generates-same-topology-separable-if*:
assumes *metric-set S d metric-set S d'*
and *metric-set.mtopology S d* = *metric-set.mtopology S d'*
and *separable-metric-set S d*
shows *separable-metric-set S d'*
proof −
interpret *m1*: *separable-metric-set S d* **by** *fact*
interpret *m2*: *metric-set S d'* **by** *fact*
obtain $\mathcal{O}$ **where** *countable* $\mathcal{O}$ *m1.mtopology-basis* $\mathcal{O}$
using *m1.second-countable'* **by** *auto*
thus *?thesis*
by(*auto intro*!: *m2.separable-if-second-countable simp*: *assms*(*3*)[*symmetric*])
qed

lemma *metric-generates-same-topology-separable*:
assumes *metric-set S d metric-set S d'*
and *metric-set.mtopology S d* = *metric-set.mtopology S d'*
shows *separable-metric-set S d* $\longleftrightarrow$ *separable-metric-set S d'*
using *metric-generates-same-topology-separable-if*[*OF assms*] *metric-generates-same-topology-separable-if*[*OF assms*(*2*,*1*) *assms*(*3*)[*symmetric*]]
by *auto*

lemma(**in** *metric-set*) *separable-if-totally-bounded*:
assumes *totally-boundedS*
shows *separable-metric-set S dist*
unfolding *separable-iff-topological-separable*
proof −
have $\exists A.$ *finite* $A \wedge A \subseteq S \wedge S = \bigcup$ $((\lambda a.$ *open-ball a* (*1 / real* (*Suc n*))) ‘ $A)$
for *n*
using *totally-boundedSD*[*OF assms*,*of 1 / Suc n*] **by** *fastforce*
then obtain *A* **where** *A*:$\bigwedge n.$ *finite* $(A\ n)$ $\bigwedge n.$ $A\ n \subseteq S$ $\bigwedge n.$ $S = \bigcup$ $((\lambda a.$ *open-ball 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 mtopology*
unfolding *dense-set-def2 separable-def*
proof(*safe intro*!: *exI*[**where** *x*=*K*] *1*)
fix *x* **and** ε :: *real*
assume *h*: $x \in S$ $0 < \varepsilon$
then obtain *n* **where** *n*:*1* / *real* (*Suc n*) $\leq \varepsilon$
by (*meson nat-approx-posE order.strict-iff-not*)
then obtain *y* **where** *y*: $y \in A$ n $x \in$ *open-ball y* (*1* / *real* (*Suc n*))
using *h(1) A(3)*[*of n*] **by** *auto*
then show $\exists\, y{\in}K.$ *dist x y* $< \varepsilon$
using *open-ballD*[*OF y(2)*] *n* **by**(*auto intro*!: *bexI*[**where** *x*=*y*] *simp*: *dist-sym*[*of y x*] *K-def*)
qed(*use K-def A(2)* **in** *auto*)
qed

lemma *second-countable-metric-class-separable-set*:
separable-metric-set (*UNIV* :: *'a* ::{*metric-space*,*second-countable-topology*} *set*) *dist*
proof −
interpret *m*: *metric-set UNIV dist*
by(*rule metric-class-metric-set*)
obtain *B* :: *'a set set* **where** *countable B* ∧ *topological-basis B*
using *second-countable-topology-class.ex-countable-basis* **by** *auto*
then show *?thesis*
by(*auto intro*!: *m.separable-if-second-countable*[**where** $\mathcal{O}$=*B*] *simp*: *topological-basis-set*)
qed

lemma *second-countable-euclidean*[*simp*]:
second-countable (*euclidean* :: *'a* :: {*metric-space*,*second-countable-topology*} *topology*)
by (*metis euclidean-mtopology second-countable-metric-class-separable-set separable-metric-set.second-countable*)

lemma *separable-euclidean*[*simp*]:
separable (*euclidean* :: *'a* :: {*metric-space*,*second-countable-topology*} *topology*)
by(*auto intro*!: *separable-if-second-countable*)

lemma(**in** *separable-metric-set*) *submetric-separable*:
assumes $S' \subseteq S$
shows *separable-metric-set* S' (*submetric* S' *dist*)
proof −
interpret *m*: *metric-set* S' *submetric* S' *dist*
by(*rule submetric-metric-set*[*OF assms*])

```
  obtain 𝒪 where ho:countable 𝒪 mtopology-basis 𝒪
    using second-countable' by auto
  show ?thesis
  proof(rule m.separable-if-second-countable[where 𝒪={S' ∩ U | U. U∈𝒪}])
    show countable {S' ∩ U |U. U ∈ 𝒪}
      using countable-image[where f=(∩) S',OF ho(1)]
      by (simp add: Setcompr-eq-image)
  next
    show m.mtopology-basis {S' ∩ U |U. U ∈ 𝒪}
      by(auto simp: submetric-subtopology[OF assms,symmetric] intro!: subtopol-
ogy-base-of ho(2))
  qed
qed

lemma(in separable-metric-set) Lindelof-diam:
  assumes 0 < e
  shows ∃ U. countable U ∧ ⋃ U = S ∧ (∀ u∈U. diam u < ennreal e)
proof −
  have (⋀u. u ∈ {open-ball x (e / 3) |x. x ∈ S} ⟹ openin mtopology u)
    by(auto simp: openin-open-ball)
  moreover have ⋃ {open-ball x (e / 3) |x. x ∈ S} = S
    using open-ball-ina open-ball-subset-ofS assms by auto
  ultimately have ∃ U'. countable U' ∧ U' ⊆ {open-ball x (e / 3) |x. x ∈ S} ∧
⋃ U' = S
    by(rule Lindelof-of[OF second-countable,simplified mtopology-topspace]) auto
  then obtain U' where U': countable U' U' ⊆ {open-ball x (e / 3) |x. x ∈ S}
⋃ U' = S
    by auto
  show ?thesis
  proof(safe intro!: exI[where x=U'])
    fix u
    assume u ∈ U'
    then obtain x where u:u = open-ball x (e / 3)
      using U' by auto
    have diam u ≤ ennreal (2 * (e / 3))
      by(simp only: u diam-ball-leq)
    also have ... < ennreal e
      by(auto intro!: ennreal-lessI assms)
    finally show diam u < ennreal e .
  qed(use U' in auto)
qed
```

2.1.4 Polish Metric Spaces

```
locale polish-metric-set = complete-metric-set + separable-metric-set

lemma polish-class-polish-set[simp]:
 polish-metric-set (UNIV :: 'a :: polish-space set) dist
  using second-countable-metric-class-separable-set complete-space-complete-metric-set
```

by(*simp add*: *polish-metric-set-def*)

lemma(**in** *polish-metric-set*) *submetric-polish*:
assumes $M \subseteq S$ **and** *closedin mtopology M*
shows *polish-metric-set M* (*submetric M dist*)
using *submetric-separable*[*OF assms*(*1*)] *submetric-complete-iff*[*OF assms*(*1*)]
by(*simp add*: *polish-metric-set-def assms*(*2*))

lemma *polish-metric-setI*:
assumes *complete-metric-set S d separable* (*metric-set.mtopology S d*)
shows *polish-metric-set S d*
using *assms* **by**(*auto intro*!: *separable-metric-setI simp*: *polish-metric-set-def complete-metric-set-def*)

2.1.5 Compact Metric Spaces

locale *compact-metric-set* = *metric-set* +
assumes *mtopology-compact*:*compact-space mtopology*
begin

context
fixes S' :: *'b set* **and** *dist'*
assumes *S'-dist*: *metric-set S' dist'*
begin

interpretation *m'*: *metric-set S' dist'* **by** *fact*

lemma *continuous-map-is-uniform*:
assumes *continuous-map mtopology m'.mtopology f*
shows *uniform-continuous-map S dist S' dist' f*
unfolding *uniform-continuous-map-def*[*OF metric-set-axioms m'.metric-set-axioms*]
proof *safe*
show *goal1*:$\bigwedge x.\ x \in S \Longrightarrow f\ x \in S'$
using *assms* **by**(*auto simp*: *continuous-map-def mtopology-topspace m'.mtopology-topspace*)
fix *e* :: *real*
assume *e*:$0 < e$
{ fix *x*
assume *x*:$x \in S$
then have $\exists \delta{>}0.\ \forall y{\in}S.\ dist\ x\ y < \delta \longrightarrow dist'\ (f\ x)\ (f\ y) < e\ /\ 2$
using *assms*(*1*)[*simplified metric-set-continuous-map-eq*[*OF metric-set-axioms m'.metric-set-axioms*]] *half-gt-zero*[*OF e*]
by *metis*
}
then obtain δ **where** *delta*:$\bigwedge x.\ x \in S \Longrightarrow \delta\ x > 0\ \bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow dist\ x\ y < \delta\ x \Longrightarrow dist'\ (f\ x)\ (f\ y) < e\ /\ 2$
by *metis*
show $\exists \delta{>}0.\ \forall x{\in}S.\ \forall y{\in}S.\ dist\ x\ y < \delta \longrightarrow dist'\ (f\ x)\ (f\ y) < e$
proof(*cases* $S = \{\}$)
case *True*

```
    then show ?thesis
      by (auto intro!: exI[where x=1])
  next
    case nem:False
    have ∃F. finite F ∧ F ⊆ {open-ball x (δ x / 2)|x. x ∈ S} ∧ S ⊆ ⋃ F
      using open-ball-ina[OF - half-gt-zero[OF delta(1)]] mtopology-compact
     by(auto intro!: compactinD simp: compact-space-def mtopology-topspace openin-open-ball)
    then obtain F where F: finite F F ⊆ {open-ball x (δ x / 2)|x. x ∈ S} S ⊆
⋃ F
      by auto
    have F-nem:F ≠ {}
      using nem F by auto
    have a ∈ F ⟹ (∃x∈S. a = open-ball x (δ x / 2)) for a
      using F(2) by auto
     then obtain xa where xa:⋀a. a ∈ F ⟹ xa a ∈ S ⋀a. a ∈ F ⟹ a =
open-ball (xa a) (δ (xa a) / 2)
      by metis
    define δ' where δ' ≡ (MIN a∈F. δ (xa a) / 2)
    have fin:finite ((λa. δ (xa a)/ 2) ‘ F)
      using F by auto
    have nemd: ((λa. δ (xa a)/ 2) ‘ F) ≠ {}
      using F-nem by auto
    have d-pos: δ' > 0
       by(auto simp: δ'-def linorder-class.Min-gr-iff[OF fin nemd] intro!: delta(1)
xa)
    show ?thesis
    proof(safe intro!: exI[where x=δ'])
      fix x y
      assume h:x ∈ S y ∈ S dist x y < δ'
      then obtain a where a:x ∈ a a ∈ F
        using F(3) by auto
      have dist (xa a) y ≤ dist (xa a) x + dist x y
        by(auto intro!: dist-tr xa a simp: h)
      also have ... < δ' + δ (xa a) / 2
        using h xa(2)[OF a(2)] a(1) open-ballD[of x xa a] by fastforce
      also have ... ≤ δ (xa a) / 2 + δ (xa a) / 2
      proof −
        have δ' ≤ δ (xa a) / 2
          by(simp only: δ'-def,rule Min.coboundedI[OF fin]) (use a in auto)
        thus ?thesis by simp
      qed
      finally have 2:dist (xa a) y < δ (xa a) by simp
      have dist' (f x) (f y) ≤ dist' (f x) (f (xa a)) + dist' (f (xa a)) (f y)
        by(auto intro!: m'.dist-tr goal1 h xa a)
      also have ... < e
      proof −
        have [simp]:dist (xa a) x < δ (xa a)
          using a(1) xa[OF a(2)] delta(1) open-ballD by fastforce
        have dist' (f x) (f (xa a)) < e / 2
```

```
        by(simp only: m'.dist-sym[where x=f x],rule delta(2)) (auto intro!: xa a
h)
       moreover have dist' (f (xa a)) (f y) < e / 2
        by(rule delta(2)[OF - - 2]) (auto intro!: h xa a)
       ultimately show ?thesis by simp
     qed
     finally show dist' (f x) (f y) < e .
   qed(rule d-pos)
  qed
qed

end


lemma totally-bounded: totally-boundedS
  unfolding totally-boundedS-def
proof safe
  fix ε :: real
  assume 0 < ε
  define 𝒰 where 𝒰 ≡ (λa. open-ball a ε) ‘ S
  have 1: ⋀U. U ∈ 𝒰 ⟹ openin mtopology U
    by(auto simp: 𝒰-def openin-open-ball)
  have 2:⋃ 𝒰 = S
    using open-ball-ina[OF - ‹0 < ε›] open-ball-subset-ofS
    by(auto simp: 𝒰-def)
  obtain ℱ where ℱ ⊆ 𝒰 finite ℱ ⋃ ℱ = S
     using 1 2 compact-space[of mtopology,simplified mtopology-compact mtopol-
ogy-topspace] by metis
  then obtain A where A ⊆ S finite A ⋃ ((λa. open-ball a ε) ‘ A) = S
    by(simp add: 𝒰-def) (metis finite-subset-image)
  thus ∃ A. eps-net ε A
    by(auto intro!: exI[where x=A] simp: eps-net-def ‹0 < ε›)
qed

lemma sequentially-compact: sequentially-compact
  unfolding sequentially-compact-def
proof safe
  fix xn
  assume xn ∈ sequence
  then have xn:⋀n. xn n ∈ S by auto
  have ¬ (∀ x∈S. ∃ e>0. finite {n. xn n ∈ open-ball x e})
  proof
    assume contr:∀ x∈S. ∃ e>0. finite {n. xn n ∈ open-ball x e}
    then obtain e where e: ⋀x. x ∈ S ⟹ e x > 0 ⋀x. x ∈ S ⟹ finite {n. xn
n ∈ open-ball x (e x)}
     by metis
    define U where U ≡ {open-ball x (e x)|x. x ∈ S}
    have ⋀u. u ∈ U ⟹ openin mtopology u topspace mtopology ⊆ ⋃ U
      by(auto simp: U-def openin-open-ball mtopology-topspace open-ball-ina[OF -
```

```
e(1)])
    then obtain F where F: finite F F ⊆ U S ⊆ ⋃ F
      using mtopology-compact compactinD by (metis compact-space-def mtopol-
ogy-topspace)
    then have finite (⋃f∈F. {n. xn n ∈ f})
      using e(2) by(auto simp: U-def)
    moreover have UNIV = (⋃f∈F. {n. xn n ∈ f})
      using F(3) xn by auto
    ultimately show False by simp
  qed
  then obtain x where x:x ∈ S ⋀e. e > 0 ⟹ infinite {n. xn n ∈ open-ball x e}
    by metis
  have inf:infinite {n. n > m ∧ xn n ∈ open-ball x e} if e > 0 for e m
  proof
    assume finite {n. m < n ∧ xn n ∈ open-ball x e}
    then have finite ({..m} ∪ {n. m < n ∧ xn n ∈ open-ball x e})
      by auto
    moreover have {n. xn n ∈ open-ball x e} ⊆ {..m} ∪ {n. m < n ∧ xn n ∈
open-ball x e}
      by auto
    ultimately show False
      using x(2)[OF that] finite-subset by blast
  qed
  define a where a ≡ rec-nat (SOME n. xn n ∈ open-ball x 1) (λn an. SOME m.
m > an ∧ xn m ∈ open-ball x (1 / Suc n))
  have an: xn (a n) ∈ open-ball x (1 / n) if n > 0 for n
  proof(cases n)
    case 0
    with that show ?thesis by simp
  next
    case (Suc n)
    have [simp]:a (Suc n) = (SOME m. m > a n ∧ xn m ∈ open-ball x (1 / Suc
n))
      by(auto simp: a-def)
    obtain m where m:a n < m xn m ∈ open-ball x (1 / (Suc n))
      using inf[of 1 / (real (Suc n)) a n] not-finite-existsD by auto
    have a (Suc n) > a n ∧ xn (a (Suc n)) ∈ open-ball x (1 / (Suc n))
      by(simp,rule someI2[of - m]) (use m in auto)
    then show ?thesis
      by(simp only: Suc)
  qed
  have as:strict-mono a
    unfolding strict-mono-Suc-iff
  proof safe
    fix n
    have [simp]:a (Suc n) = (SOME m. m > a n ∧ xn m ∈ open-ball x (1 / Suc
n))
      by(auto simp: a-def)
    obtain m where m:a n < m xn m ∈ open-ball x (1 / (Suc n))
```

```
    using inf[of 1 / (real (Suc n)) a n] not-finite-existsD by auto
   have a (Suc n) > a n ∧ xn (a (Suc n)) ∈ open-ball x (1 / (Suc n))
    by(simp,rule someI2[of - m]) (use m in auto)
   thus a n < a (Suc n) by simp
  qed
  show ∃ a. strict-mono a ∧ convergent-inS (xn ∘ a)
   unfolding convergent-inS-def converge-to-inS-def2'
  proof(safe intro!: exI[where x=a] exI[where x=x])
   fix e :: real
   assume 0 < e
   then obtain N ::nat where N: N > 0 1 / N < e
    by (meson nat-approx-posE zero-less-Suc)
   show ∃ N. ∀ n≥N. (xn ∘ a) n ∈ open-ball x e
   proof(safe intro!: exI[where x=N])
    fix n
    assume n:n ≥ N
    show (xn ∘ a) n ∈ open-ball x e
     using order.trans[OF open-ball-le[of 1 / n] open-ball-le[of 1 / N e]] N n
an[of n] inverse-of-nat-le
     by auto
   qed
  qed(auto simp: simp: as  x xn)
qed

lemma polish: polish-metric-set S dist
  using separable-if-totally-bounded[OF totally-bounded]
 by(simp add: polish-metric-set-def complete-metric-set-def complete-metric-set-axioms-def
separable-metric-set-def)
    (meson Cauchy-inS-def converge-if-Cauchy-and-subconverge convergent-inS-def
sequentially-compact sequentially-compact-def)

sublocale polish-metric-set
  by(rule polish)

end

lemma(in metric-set) ex-lebesgue-number:
  assumes S ≠ {} sequentially-compact ⋀u. u ∈ U ⟹ openin mtopology u S ⊆
⋃ U
  shows ∃ d>0. ∀ a⊆S. diam a < ennreal d ⟶ (∃ u∈U. a ⊆ u)
proof(rule ccontr)
  assume ¬ (∃ d>0. ∀ a⊆S. diam a < ennreal d ⟶ (∃ u∈U. a ⊆ u))
  then have ⋀n. ∃ a⊆S. diam a < ennreal (1 / Suc n) ∧ (∀ x∈U. ¬ a ⊆ x) by
auto
  then obtain An where an: ⋀n. An n ⊆ S ⋀n. diam (An n) < ennreal (1 /
Suc n) ⋀n u. u ∈ U ⟹ ¬ (An n) ⊆ u
   by metis
  have An n ≠ {} for n
  proof
```

assume *An n = {}*
then have *U = {} ∨ (∀ u∈U. u = {})*
using *an(3)[of - n]* **by** *auto*
thus *False*
using *assms(1,4)* **by** *blast*
qed
then obtain *xn* **where** *xn:⋀n. xn n ∈ An n*
by (*meson ex-in-conv*)
then have *xn':⋀n. xn n ∈ S* **using** *an* **by** *auto*
then obtain *a x* **where** *ax:strict-mono a converge-to-inS (xn ∘ a) x*
using *assms(2)* **by**(*fastforce simp*: *sequentially-compact-def convergent-inS-def*)
then have *x*: *x ∈ S* **by**(*auto simp*: *converge-to-inS-def*)
then obtain *u* **where** *u:x ∈ u u ∈ U*
using *assms(4)* **by** *auto*
obtain *e* **where** *e:e > 0 open-ball x e ⊆ u*
using *assms(3)[OF u(2)] u(1) mtopology-openin-iff* **by** *fastforce*
obtain *n ::nat* **where** *n*: *1 / Suc n < e / 2*
using *e(1) half-gt-zero nat-approx-posE* **by** *blast*
obtain *n'* **where** *n':⋀n. n ≥ n' ⟹ xn (a n) ∈ open-ball x (e / 2)*
using *e(1) ax(2)* **by**(*auto simp*: *converge-to-inS-def2'*) (*meson half-gt-zero*)
define *n0* **where** *n0 ≡ max (a (Suc n)) (a n')*
have *n0:1 / Suc n0 < e / 2 xn n0 ∈ open-ball x (e / 2)*
proof −
have *Suc n0 ≥ Suc n*
using *seq-suble[OF ax(1),of Suc n]* **by** (*simp add*: *n0-def*)
hence *1 / Suc n0 ≤ 1 / Suc n*
using *inverse-of-nat-le* **by** *blast*
thus *1 / Suc n0 < e / 2*
using *n* **by** *auto*
show *xn n0 ∈ open-ball x (e / 2)*
by (*cases a (Suc n) ≤ a n'*) (*auto intro*!: *n' simp*: *n0-def ax(1) strict-mono-less-eq*)
qed
have *An n0 ⊆ open-ball x e*
unfolding *open-ball-def*
proof *safe*
fix *y*
assume *y:y ∈ An n0*
have *dist x y ≤ dist x (xn n0) + dist (xn n0) y*
using *y xn' an* **by**(*auto intro*!: *dist-tr simp*: *x*)
also have *... < e / 2 + dist (xn n0) y*
using *open-ballD[OF n0(2)]* **by** *auto*
also have *... ≤ e / 2 + 1 / Suc n0*
using *xn[of n0] xn' y an* **by**(*auto intro*!: *diam-is-sup'[OF - - order.strict-implies-order[OF an(2)[of n0]],simplified*])
also have *... < e*
using *n0(1)* **by** *auto*
finally show *y ∈ (if x ∈ S then {xa ∈ S. dist x xa < e} else {})*
using *an(1) x y* **by** *auto*
qed

hence *An n0* $\subseteq$ *u*
using *e* **by** *auto*
with *an*(*3*)[*OF u*(*2*)] **show** *False* **by** *auto*
qed

lemma(**in** *metric-set*) *sequentially-compact-iff1*:
sequentially-compact $\longleftrightarrow$ *totally-boundedS* $\wedge$ *complete-metric-set S dist*
proof *safe*
assume *h*:*sequentially-compact*
then show *totally-boundedS*
using *Cauchy-if-convergent-inS* **by**(*fastforce simp*: *totally-boundedS-iff sequentially-compact-def*)
show *complete-metric-set S dist*
proof
fix *xn*
assume *1*:*Cauchy-inS xn*
with *h* **obtain** *a x* **where** *2*:*strict-mono a converge-to-inS* (*xn* $\circ$ *a*) *x*
by(*fastforce dest*: *Cauchy-inS-dest1 simp*: *sequentially-compact-def convergent-inS-def*)
thus *convergent-inS xn*
by(*auto simp*: *convergent-inS-def converge-if-Cauchy-and-subconverge*[*OF 2 1*] *intro*!: *exI*[**where** *x*=*x*])
qed
next
assume *h*:*totally-boundedS complete-metric-set S dist*
show *sequentially-compact*
unfolding *sequentially-compact-def*
proof *safe*
fix *xn*
assume *xn* $\in$ *sequence*
then obtain *a* **where** *a*:*strict-mono a Cauchy-inS* (*xn* $\circ$ *a*)
using *h* **by**(*auto simp*: *totally-boundedS-iff*)
thus $\exists$ *a*. *strict-mono a* $\wedge$ *convergent-inS* (*xn* $\circ$ *a*)
using *h* **by**(*auto intro*!: *exI*[**where** *x*=*a*] *simp*: *complete-metric-set-def complete-metric-set-axioms-def*)
qed
qed

lemma(**in** *metric-set*) *sequentially-compact-compact*:
assumes *sequentially-compact*
shows *compact-metric-set S dist*
proof
show *compact-space mtopology*
proof(*cases S* = {})
case *True*
have [*simp*]:*topspace mtopology* = {}
by(*simp add*: *mtopology-topspace*,*fact*)
show *?thesis*
by(*auto simp*: *compact-space intro*!: *exI*[**where** *x*={}])

```
  next
    case 1:False
    {
      fix U
      assume h:⋀u. u ∈ U ⟹ openin mtopology u S ⊆ ⋃ U
      obtain d where d:d > 0 ⋀a. a ⊆ S ⟹ diam a < ennreal d ⟹ ∃ u∈U. a
⊆ u
        using ex-lebesgue-number[OF 1 assms h] by metis
      obtain B where B:finite B B ⊆ S S = (⋃ a∈B. open-ball a (d / 3))
        using totally-boundedSD[of d / 3] d(1) assms
        by(auto simp: sequentially-compact-iff1)
      have ∃ u∈U. open-ball b (d / 3) ⊆ u if b ∈ B for b
      using open-ballD' d(1) by(auto intro!: d(2) order.strict-trans1[OF diam-ball-leq[of
b d / 3]] simp: ennreal-less-iff)
      then obtain u where u:⋀b. b ∈ B ⟹ u b ∈ U ⋀b. b ∈ B ⟹ open-ball b
(d / 3) ⊆ u b
        by metis
      have ∃ F. finite F ∧ F ⊆ U ∧ S ⊆ (⋃ F)
        using B u by(fastforce intro!: exI[where x=u ‘ B])
    }
    thus ?thesis
      by (simp add: compact-space-alt mtopology-topspace)
  qed
qed

corollary(in metric-set) compact-iff-sequentially-compact:
compact-space mtopology ⟷ sequentially-compact
  using compact-metric-set.sequentially-compact sequentially-compact-compact com-
pact-metric-set-axioms-def compact-metric-set-def metric-set-axioms
  by blast

corollary(in metric-set) compact-iff2:
compact-space mtopology ⟷ totally-boundedS ∧ complete-metric-set S dist
  by(simp add: compact-iff-sequentially-compact sequentially-compact-iff1)

corollary(in complete-metric-set) compactin-closed-iff:
  assumes closedin mtopology C
  shows compactin mtopology C ⟷ totally-bounded-on C
proof −
  from assms have C:C ⊆ S
    using mtopology-closedin-iff by blast
  then interpret C: complete-metric-set C submetric C dist
    by(auto simp: submetric-complete-iff assms)
  show ?thesis
   by(simp add: compactin-subspace submetric-subtopology[OF C] totally-bounded-on-submetric[OF
C] mtopology-topspace C  C.compact-iff2 C.complete-metric-set-axioms)
qed
```

2.1.6 Completion

context *metric-set*
begin

abbreviation *Cauchys* ≡ *Collect Cauchy-inS*

definition *Cauchy-r* :: ((*nat* ⇒ *'a*) × (*nat* ⇒ *'a*)) *set* **where**
Cauchy-r ≡ {(*xn*,*yn*)|*xn yn*. *Cauchy-inS xn* ∧ *Cauchy-inS yn* ∧ (λ*n*. *dist* (*xn n*) (*yn n*)) ⟶ *0*}

lemma *Cauchy-r-equiv*[*simp*]: *equiv Cauchys Cauchy-r*
proof(*rule equivI*)
show *refl-on Cauchys Cauchy-r*
by(*auto simp*: *refl-on-def Cauchy-r-def*)
next
show *sym Cauchy-r*
using *dist-sym* **by**(*auto simp*: *sym-def Cauchy-r-def*)
next
show *trans Cauchy-r*
proof(*rule transI*)
show ⋀*x y z*. (*x*, *y*) ∈ *Cauchy-r* ⟹ (*y*, *z*) ∈ *Cauchy-r* ⟹ (*x*, *z*) ∈ *Cauchy-r*
unfolding *Cauchy-r-def*
proof *safe*
fix *xn yn zn*
assume *h*:*Cauchy-inS xn Cauchy-inS yn Cauchy-inS zn*
(λ*n*. *dist* (*xn n*) (*yn n*)) ⟶ *0* (λ*n*. *dist* (*yn n*) (*zn n*)) ⟶ *0*
then show ∃ *xn' yn'*. (*xn*, *zn*) = (*xn'*, *yn'*) ∧ *Cauchy-inS xn'* ∧ *Cauchy-inS yn'* ∧ (λ*n*. *dist* (*xn' n*) (*yn' n*)) ⟶ *0*
by(*auto intro*!: *tendsto-0-le*[*OF tendsto-add-zero*[*OF h*(*4*,*5*)],*of - 1*] *dist-tr eventuallyI simp*: *dist-geq0 Cauchy-inS-dest1*)
qed
qed
qed

abbreviation *S-completion* :: (*nat* ⇒ *'a*) *set set* (*S**) **where**
S-completion ≡ *Cauchys* // *Cauchy-r*

lemma *S-c-represent*:
assumes *X* ∈ *S**
obtains *xn* **where** *xn* ∈ *X Cauchy-inS xn*
using *equiv-Eps-in*[*OF - assms*] *equiv-Eps-preserves*[*OF - assms*] **by** *auto*

lemma *Cauchy-inS-ignore-initial-eq*:
assumes *Cauchy-inS xn*
shows (*xn*, (λ*n*. *xn* (*n* + *k*))) ∈ *Cauchy-r*
by(*auto simp*: *Cauchy-r-def Cauchy-inS-ignore-initial*[*OF assms*] *assms*,*insert assms*)
(*auto simp*: *LIMSEQ-def dist-real-def dist-geq0 Cauchy-inS-def*,*metis add.commute trans-le-add2*)

corollary *Cauchy-inS-r*: $a \in S \Longrightarrow (\lambda n.\ a,\ \lambda n.\ a) \in$ *Cauchy-r*
by(*auto intro*!: *Cauchy-inS-ignore-initial-eq Cauchy-inS-const*)

abbreviation *dist-completion'* :: [*nat* ⇒ *'a*, *nat* ⇒ *'a*] ⇒ *real* **where**
dist-completion' xn yn ≡ *lim* (λ*n*. *dist* (*xn n*) (*yn n*))

lemma *dist-of-completion-congruent2*: *dist-completion' respects2 Cauchy-r*
proof(*safe intro*!: *congruent2-commuteI*[*OF Cauchy-r-equiv*])
fix *xn yn zn*
assume *h*:(*xn*,*yn*) ∈ *Cauchy-r Cauchy-inS zn*
then have *h'*:*Cauchy-inS xn Cauchy-inS yn* (λ*n*. *dist* (*xn n*) (*yn n*)) ⟶ *0*
by(*auto simp*: *Cauchy-r-def*)
have *1*:(λ*n*. *dist* (*zn n*) (*xn n*)) ⟶ *lim* (λ*n*. *dist* (*zn n*) (*xn n*))
using *Cauchy-inS-dist-convergent*[*OF h*(*2*) *h'*(*1*)] **by**(*simp add*: *convergent-LIMSEQ-iff*)
have *2*:(λ*n*. *dist* (*zn n*) (*yn n*)) ⟶ *lim* (λ*n*. *dist* (*zn n*) (*xn n*))
using *h*(*2*) *h'*(*1*,*2*) *dist-tr-abs*[*of zn - xn - yn -*,*simplified abs-diff-le-iff*]
by(*auto intro*!: *real-tendsto-sandwich*[*OF - - tendsto-diff*[*OF 1 h'*(*3*),*simplified*]
tendsto-add[*OF 1 h'*(*3*),*simplified*]] *eventuallyI dist-tr dest*: *Cauchy-inS-dest1*) (*simp add*: *Cauchy-inS-dest1 add.commute diff-le-eq*)
show *dist-completion' zn xn* = *dist-completion' zn yn*
using *1 2* **by**(*auto dest*: *limI*)
qed(*auto simp*: *dist-sym*)

definition *dist-completion* :: [(*nat* ⇒ *'a*) *set*, (*nat* ⇒ *'a*) *set*] ⇒ *real* (*dist**) **where**
dist X Y* ≡ (*if X* ∈ *S** ∧ *Y* ∈ *S* then dist-completion'* (*SOME xn*. *xn* ∈ *X*)
(*SOME yn*. *yn* ∈ *Y*) *else 0*)

lemma *dist-c-def*:
assumes *xn* ∈ *X yn* ∈ *Y X* ∈ *S* Y* ∈ *S**
shows *dist* X Y* = *dist-completion' xn yn*
by(*auto simp*: *assms dist-completion-def*,*rule someI2*[*of* λ*x*. *x* ∈ *X*,*OF assms*(*1*)],*rule someI2*[*of* λ*x*. *x* ∈ *Y*,*OF assms*(*2*)])
(*auto simp*: *congruent2D*[*OF dist-of-completion-congruent2 quotient-eq-iff*[*OF - assms*(*3*,*3*,*1*),*simplified*] *quotient-eq-iff*[*OF - assms*(*4*,*4*,*2*),*simplified*]])

lemma *completion-metric-set*: *metric-set S* dist**
proof
fix *X Y*
consider *X* ∈ *S* Y* ∈ *S** | *X* ∉ *S** | *Y* ∉ *S** **by** *blast*
then show *0* ≤ *dist* X Y*
proof *cases*
case *1*
then obtain *xn yn* **where** *h*: *xn* ∈ *X yn* ∈ *Y Cauchy-inS xn Cauchy-inS yn*
by (*meson S-c-represent*)
with *1* **show** *?thesis*
by(*auto simp*: *dist-c-def intro*!: *Lim-bounded2*[*OF Cauchy-inS-dist-convergent*[*OF h*(*3*,*4*),*simplified convergent-LIMSEQ-iff*]] *dist-geq0*)

```
  qed(auto simp: dist-completion-def)
next
  fix X Y
  show dist* X Y = dist* Y X
    by(auto simp: dist-completion-def dist-sym)
next
  fix X Y
  assume h:X ∈ S* Y ∈ S*
  then obtain xn yn where h': xn ∈ X yn ∈ Y Cauchy-inS xn Cauchy-inS yn
    by (meson S-c-represent)
  show X = Y ⟷ dist* X Y = 0
  proof
    assume X = Y
    then show dist* X Y = 0
      using h' h by(auto simp: dist-c-def)
  next
    assume dist* X Y = 0
    then have (xn, yn) ∈ Cauchy-r
     using h h' convergent-LIMSEQ-iff[THEN iffD1,OF Cauchy-inS-dist-convergent[OF
h'(3,4)]]
      by(auto simp: dist-c-def Cauchy-r-def)
    thus X = Y
      by(simp add: quotient-eq-iff[OF - h h'(1,2)])
  qed
next
  fix X Y Z
  assume h:X ∈ S* Y ∈ S* Z ∈ S*
   then obtain xn yn zn where h': xn ∈ X yn ∈ Y zn ∈ Z Cauchy-inS xn
Cauchy-inS yn Cauchy-inS zn
    by (meson S-c-represent)
  have dist* X Z = dist-completion' xn zn
    using h h' by(simp add: dist-c-def)
  also have ... ≤ lim (λn. dist (xn n) (yn n) + dist (yn n) (zn n))
   using h' by(auto intro!: lim-mono[OF - convergent-LIMSEQ-iff[THEN iffD1,OF
Cauchy-inS-dist-convergent[OF h'(4,6)]] convergent-LIMSEQ-iff[THEN iffD1,OF
convergent-add[OF Cauchy-inS-dist-convergent[OF h'(4,5)] Cauchy-inS-dist-convergent[OF
h'(5,6)]]]] dist-tr dest: Cauchy-inS-dest1)
  also have ... = dist-completion' xn yn + dist-completion' yn zn
   using tendsto-add[OF convergent-LIMSEQ-iff[THEN iffD1,OF Cauchy-inS-dist-convergent[OF
h'(4,5)]] convergent-LIMSEQ-iff[THEN iffD1,OF Cauchy-inS-dist-convergent[OF
h'(5,6)]]]
    by(simp add: limI)
  also have ... = dist* X Y + dist* Y Z
    using h h' by(simp add: dist-c-def)
  finally show dist* X Z ≤ dist* X Y + dist* Y Z .
qed(simp add: dist-completion-def)

interpretation c:metric-set S* dist*
  by(rule completion-metric-set)
```

definition *into-S-c* :: *'a* ⇒ (*nat* ⇒ *'a*) *set* **where**
into-S-c a ≡ *Cauchy-r* `` {(λ*n*. *a*)}

lemma *into-S-c-in*:
assumes *a* ∈ *S*
shows (λ*n*. *a*) ∈ *into-S-c a*
using *Cauchy-inS-const*[*OF assms*] *Cauchy-inS-r*[*OF assms*]
by(*auto simp*: *into-S-c-def*)

lemma *into-S-c-into*:
assumes *a* ∈ *S*
shows *into-S-c a* ∈ *S**
by(*auto simp*: *into-S-c-def intro*!: *quotientI Cauchy-if-convergent-inS convergent-inS-const assms*)

lemma *into-S-inj*: *inj-on into-S-c S*
proof
fix *x y*
assume *x* ∈ *S y* ∈ *S into-S-c x* = *into-S-c y*
with *eq-equiv-class-iff*[*THEN iffD1*,*OF Cauchy-r-equiv - - this*(*3*)[*simplified into-S-c-def*]]
have (λ*n*. *x*, λ*n*. *y*) ∈ *Cauchy-r*
by(*auto simp*: *Cauchy-if-convergent-inS convergent-inS-const*)
thus *x* = *y*
using *dist-0*[*OF* ‹*x* ∈ *S*› ‹*y* ∈ *S*›]
by(*auto simp*: *Cauchy-r-def LIMSEQ-const-iff*)
qed

lemma *dist-into-S-c*:
assumes *x* ∈ *S y* ∈ *S*
shows *dist** (*into-S-c x*) (*into-S-c y*) = *dist x y*
using *into-S-c-in*[*OF assms*(*1*)] *into-S-c-in*[*OF assms*(*2*)] *into-S-c-into*[*OF assms*(*1*)] *into-S-c-into*[*OF assms*(*2*)]
by(*simp add*: *dist-c-def*)

lemma *S-c-isometry*:
c.ed into-S-c S = *dist*
by *standard*+ (*auto simp*: *c.embed-dist-on-def dist-into-S-c dist-notin dist-notin'*)

corollary *mtopology-embedding-S-c-map*:
homeomorphic-map mtopology (*subtopology c.mtopology* (*into-S-c* `` *S*)) *into-S-c*
using *into-S-c-into* **by**(*auto intro*!: *c.embed-dist-topology-homeomorphic-map*[*OF - into-S-inj*,*simplified S-c-isometry*])

corollary *mtopology-embedding-S-c*:
mtopology homeomorphic-space subtopology c.mtopology (*into-S-c* `` *S*)
using *mtopology-embedding-S-c-map homeomorphic-space* **by** *blast*

lemma *into-S-c-image-dense*: *c.dense-set* (*into-S-c* `` *S*)

```
  unfolding c.dense-set-def2'
proof safe
  fix X
  assume X:X ∈ S*
  from S-c-represent[OF this] obtain xn where xn:xn ∈ X Cauchy-inS xn
    by auto
  show ∃f∈UNIV → into-S-c ' S. c.converge-to-inS f X
  proof(safe intro!: bexI[where x=λn. into-S-c (xn n)])
    show c.converge-to-inS (λn. into-S-c (xn n)) X
      unfolding c.converge-to-inS-def2
    proof safe
      fix e :: real
      assume e:e > 0
      then obtain N where N:⋀n m. n ≥ N ⟹ m ≥ N ⟹ dist (xn n) (xn m)
< e / 2
        using xn(2) by (meson Cauchy-inS-def half-gt-zero)
      show ∃N. ∀n≥N. dist* (into-S-c (xn n)) X < e
      proof(safe intro!: exI[where x=N])
        fix n
        assume n:N ≤ n
        have dist* (into-S-c (xn n)) X = dist-completion' (λna. xn n) xn
          by(rule dist-c-def[OF into-S-c-in[OF Cauchy-inS-dest1[OF xn(2),of n]]
xn(1) into-S-c-into[OF Cauchy-inS-dest1[OF xn(2),of n]] X])
        also have ... ≤ e / 2
       by(rule Lim-bounded[OF Cauchy-inS-dist-convergent[OF Cauchy-inS-const[OF
Cauchy-inS-dest1[OF xn(2),of n]] xn(2),simplified convergent-LIMSEQ-iff],of N
e/2],auto dest: N[OF n])
        also have ... < e
          using e by auto
        finally show dist* (into-S-c (xn n)) X < e .
      qed
    qed(auto simp: Cauchy-inS-dest1[OF xn(2)] into-S-c-into X)
  qed(auto simp: Cauchy-inS-dest1[OF xn(2)] into-S-c-into)
qed (use into-S-c-into in auto)

lemma completion-complete:complete-metric-set S* dist*
proof
  fix Xn
  assume h:c.Cauchy-inS Xn
  have ⋀n. ∃xn∈S. dist* (Xn n) (into-S-c xn) < 1 / (Suc n)
    using into-S-c-image-dense c.Cauchy-inS-dest1[OF h]
    by(auto simp: c.dense-set-def2)
  then obtain xn where xn: ⋀n. xn n ∈ S ⋀n.  dist* (Xn n) (into-S-c (xn n))
< 1 / (Suc n)
    by metis
  have xnC:Cauchy-inS xn
    unfolding Cauchy-inS-def
  proof safe
    fix e :: real
```

```
    assume e:0 < e
    then obtain N1 where N1: 1 / Suc N1 < e / 3
      by (meson nat-approx-posE zero-less-divide-iff zero-less-numeral)
    obtain N2 where N2: ⋀n m. n ≥ N2 ⟹ m ≥ N2 ⟹ dist* (Xn n) (Xn m)
< e / 3
     using e h by(simp only: c.Cauchy-inS-def) (meson zero-less-divide-iff zero-less-numeral)
    show ∃ N. ∀ n≥N. ∀ m≥N. dist (xn n) (xn m) < e
    proof(safe intro!: exI[where x=max N1 N2])
      fix n m
      assume max N1 N2 ≤ n max N1 N2 ≤ m
      hence n: N1 ≤ n N2 ≤ n
        and m: N1 ≤ m N2 ≤ m by auto
      have dist (xn n) (xn m) = c.ed into-S-c S (xn n) (xn m)
        by(simp add: S-c-isometry)
      also have ... = dist* (into-S-c (xn n)) (into-S-c (xn m))
        using xn by(simp add: c.embed-dist-on-def)
     also have ... ≤ dist* (into-S-c (xn n)) (Xn n) + dist* (Xn n) (Xn m) + dist*
(Xn m) (into-S-c (xn m))
         using c.dist-tr[OF into-S-c-into[OF xn(1)[of n]] c.Cauchy-inS-dest1[OF
h,of m] into-S-c-into[OF xn(1)[of m]]] c.dist-tr[OF into-S-c-into[OF xn(1)[of n]]
c.Cauchy-inS-dest1[OF h,of n] c.Cauchy-inS-dest1[OF h,of m]]
        by simp
      also have ... < 1 / Suc n + e / 3 + 1 / Suc m
       using N2[OF n(2) m(2)] xn(2)[of m] xn(2)[of n,simplified c.dist-sym[of Xn
n]] by auto
      also have ... < e
      proof −
        have 1 / Suc n ≤ 1 / Suc N1 1 / Suc m ≤ 1 / Suc N1
          using n m inverse-of-nat-le by blast+
        thus ?thesis
          using N1 by linarith
      qed
      finally show dist (xn n) (xn m) < e .
    qed
  qed(simp add: xn)
  show c.convergent-inS Xn
    unfolding c.convergent-inS-def c.converge-to-inS-def2
  proof(safe intro!: exI[where x=Cauchy-r `` {xn}] quotientI xnC)
    fix e :: real
    assume e:0 < e
    then obtain N1 where N1: 1 / Suc N1 < e / 2
      by (meson nat-approx-posE zero-less-divide-iff zero-less-numeral)
    hence 1:dist* (Xn n) (into-S-c (xn n)) < e / 2 if n ≥ N1 for n
    proof −
      have 1 / Suc n ≤ 1 / Suc N1
        using that inverse-of-nat-le by blast
      thus ?thesis
        using xn(2)[of n] N1 by linarith
    qed
```

```
    then obtain N2 where N2:⋀n m. n ≥ N2 ⟹ m ≥ N2 ⟹ dist (xn n) (xn
m) < e / 3
     using xnC e by (meson Cauchy-inS-def zero-less-divide-iff zero-less-numeral)

    have 2:dist* (into-S-c (xn n)) (Cauchy-r `` {xn}) < e / 2 if n ≥ N2 for n
    proof −
      have dist* (into-S-c (xn n)) (Cauchy-r `` {xn}) = dist-completion' (λm. xn
n) xn
     using dist-c-def[OF into-S-c-in[OF Cauchy-inS-dest1[OF xnC,of n]] equiv-class-self[OF
Cauchy-r-equiv,of xn] into-S-c-into[OF Cauchy-inS-dest1[OF xnC,of n]]] xnC
       by (simp add: quotientI)
     also have ... ≤ e / 3
     by(rule Lim-bounded[OF Cauchy-inS-dist-convergent[OF Cauchy-inS-const[OF
Cauchy-inS-dest1[OF xnC,of n]] xnC,simplified convergent-LIMSEQ-iff],of N2 e/3],
auto dest: N2[OF that])
     also have ... < e / 2 using e by simp
     finally show dist* (into-S-c (xn n)) (Cauchy-r `` {xn}) < e / 2 .
    qed
    show ∃N. ∀n≥N. dist* (Xn n) (Cauchy-r `` {xn}) < e
    proof(safe intro!: exI[where x=max N1 N2])
     fix n
     assume max N1 N2 ≤ n
     then have n:n ≥ N1 n ≥ N2 by auto
     show dist* (Xn n) (Cauchy-r `` {xn}) < e
     using c.dist-tr[OF c.Cauchy-inS-dest1[OF h,of n] into-S-c-into[OF Cauchy-inS-dest1[OF
xnC],of n] quotientI[of xn]] xnC 1[OF n(1)] 2[OF n(2)]
       by auto
    qed
  qed(use c.Cauchy-inS-dest1[OF h] in auto)
qed

lemma dense-set-c-dense:
  assumes dense-set U
  shows c.dense-set (into-S-c ` U)
  unfolding c.dense-set-def2
proof safe
  fix X and e :: real
  assume h:X ∈ S* 0 < e
  then obtain xn where xn:xn ∈ X Cauchy-inS xn
    by(auto dest: S-c-represent)
  obtain y where y:y ∈ S dist* X (into-S-c y) < e / 2
   using h into-S-c-image-dense half-gt-zero[OF h(2)] by(simp only: c.dense-set-def2)
blast
  obtain z where z:z ∈ U dist y z < e / 2
    using half-gt-zero[OF h(2)] y(1) assms by(simp only: dense-set-def2) blast
  show ∃y∈into-S-c ` U. dist* X y < e
  proof(rule bexI[OF - imageI[OF z(1)]])
    have dist* X (into-S-c z) ≤ dist* X (into-S-c y) + dist* (into-S-c y) (into-S-c
z)
```

using *z(1) assms* **by**(*auto intro*!: *c.dist-tr h into-S-c-into y simp*: *dense-set-def*)
also have ... = *dist** *X* (*into-S-c y*) + *dist y z*
using *z(1) assms y(1) dist-into-S-c*[*of y z*] **by**(*auto simp*: *dense-set-def*)
also have ... < *e*
using *y(2) z(2)* **by** *simp*
finally show *dist** *X* (*into-S-c z*) < *e* **.**
qed
qed(*insert assms*, *auto simp*: *dense-set-def intro*!: *into-S-c-into*)

end

lemma(**in** *separable-metric-set*) *completion-polish*: *polish-metric-set S** *dist**
proof −
interpret *c*:*complete-metric-set S** *dist**
by(*rule completion-complete*)
show *?thesis*
proof
obtain *U* **where** *U*: *countable U dense-set U*
using *separable* **by** *blast*
show ∃ *U*. *countable U* ∧ *c.dense-set U*
using *U* **by**(*auto intro*!: *exI*[**where** *x*=*into-S-c* ‘ *U*] *dense-set-c-dense*)
qed
qed

2.2 Discrete Distance

definition *discrete-dist* :: ′*a set* ⇒ ′*a* ⇒ ′*a* ⇒ *real* **where**
discrete-dist S ≡ (λ*a b*. *if a* ∈ *S* ∧ *b* ∈ *S* ∧ *a* ≠ *b then 1 else 0*)

lemma
assumes *a* ∈ *S* **and** *b* ∈ *S*
shows *discrete-dist-iff-1*: *discrete-dist S a b* = *1* ⟷ *a* ≠ *b*
and *discrete-dist-iff-0*: *discrete-dist S a b* = *0* ⟷ *a* = *b*
using *assms* **by**(*auto simp*: *discrete-dist-def*)

lemma *discrete-dist-metric*:
metric-set S (*discrete-dist S*)
by(*auto simp*: *discrete-dist-def metric-set-def*)

lemma
shows *discrete-dist-ball-ge1*: *x* ∈ *S* ⟹ *1* < ε ⟹ *metric-set.open-ball S* (*discrete-dist S*) *x* ε = *S*
and *discrete-dist-ball-leq1*: *x* ∈ *S* ⟹ *0* < ε ⟹ ε ≤ *1* ⟹ *metric-set.open-ball S* (*discrete-dist S*) *x* ε = {*x*}
apply(*auto simp*: *metric-set.open-ball-def*[*OF discrete-dist-metric*],*simp-all add*: *discrete-dist-def*)
using *less-le-not-le* **by** *fastforce*

lemma *discrete-dist-complete-metric*:
complete-metric-set S (*discrete-dist S*)
proof −
interpret *m*: *metric-set S discrete-dist S*
by(*rule discrete-dist-metric*)
show *?thesis*
proof
fix *f*
assume *h*:*m.Cauchy-inS f*
then have $\bigwedge\varepsilon.\ \varepsilon{>}0 \Longrightarrow \exists\, x{\in}S.\ \exists\, N.\ \forall\, n{\geq}N.\ f\ n \in m.open\text{-}ball\ x\ \varepsilon$
by(*auto simp*: *m.Cauchy-inS-def2'*)
from *this*[*of 1*] **obtain** *x N* **where** *hxn*:
$x \in S\ \forall\, n{\geq}N.\ f\ n \in m.open\text{-}ball\ x\ 1$
by *auto*
hence $\bigwedge n.\ n \geq N \Longrightarrow f\ n = x$
using *discrete-dist-ball-leq1*[*of x S 1*] **by** *auto*
thus *m.convergent-inS f*
unfolding *m.convergent-inS-def* **using** *h hxn*(*1*)
by(*auto intro*!: *bexI*[**where** *x*=*x*] *exI*[**where** *x*=*N*] *simp*:*m.converge-to-inS-def2'*
m.Cauchy-inS-def)
qed
qed

lemma *discrete-dist-dense-set*:
metric-set.dense-set S (*discrete-dist S*) *U* $\longleftrightarrow$ $S = U$
proof −
interpret *m*: *metric-set S discrete-dist S*
by(*rule discrete-dist-metric*)
show *?thesis*
proof
assume *h*:*m.dense-set U*
show $S = U$
proof *safe*
fix *x*
assume *hx*:$x \in S$
then have $\bigwedge\varepsilon.\ \varepsilon{>}0 \Longrightarrow m.open\text{-}ball\ x\ \varepsilon \cap U \neq \{\}$
using *h* **by**(*simp add*: *m.dense-set-def*)
hence *m.open-ball x 1* $\cap$ $U \neq \{\}$ **by** *auto*
thus $x \in U$
using *discrete-dist-ball-leq1*[*OF hx*,*of 1*]
by *auto*
next
show $\bigwedge x.\ x \in U \Longrightarrow x \in S$
using *h* **by**(*auto simp*: *m.dense-set-def*)
qed
next
show $S = U \Longrightarrow m.dense\text{-}set\ U$
using *m.dense-set-S* **by** *auto*
qed

qed

lemma *discrete-dist-separable-iff*:
separable-metric-set S (*discrete-dist S*) ⟷ *countable S*
proof −
interpret *m*: *metric-set S discrete-dist S*
by(*rule discrete-dist-metric*)
show *?thesis*
proof
assume *separable-metric-set S* (*discrete-dist S*)
then obtain *U* **where** *countable U m.dense-set U*
by(*auto simp*: *separable-metric-set-def separable-metric-set-axioms-def*)
thus *countable S*
using *discrete-dist-dense-set*[*of S*] **by** *auto*
next
assume *countable S*
then show *separable-metric-set S* (*discrete-dist S*)
by(*auto simp*: *separable-metric-set-def separable-metric-set-axioms-def intro*!:*exI*[**where** *x*=*S*] *m.dense-set-S discrete-dist-metric*)
qed
qed

lemma *discrete-dist-polish-iff*: *polish-metric-set S* (*discrete-dist S*) ⟷ *countable S*
using *discrete-dist-separable-iff*[*of S*] *discrete-dist-complete-metric*[*of S*]
by(*auto simp*: *polish-metric-set-def*)

lemma *discrete-dist-topology-x*:
assumes *x* ∈ *S*
shows *openin* (*metric-set.mtopology S* (*discrete-dist S*)) {*x*}
proof −
interpret *m*: *metric-set S discrete-dist S*
by(*rule discrete-dist-metric*)
show *?thesis*
by(*auto simp*: *m.mtopology-open-ball-in*[*OF assms*,*of 1*, *simplified discrete-dist-ball-leq1*[*OF assms*]])
qed

lemma *discrete-dist-topology*:
openin (*metric-set.mtopology S* (*discrete-dist S*)) *U* ⟷ *U* ⊆ *S*
proof −
interpret *m*: *metric-set S discrete-dist S*
by(*rule discrete-dist-metric*)
show *?thesis*
proof
show *openin m.mtopology U* ⟹ *U* ⊆ *S*
using *m.mtopology-topspace*
by(*auto simp*: *topspace-def*)

next
assume $U \subseteq S$
then have $\bigwedge x.\ x \in U \Longrightarrow$ *openin m.mtopology* $\{x\}$
by(*auto simp*: *discrete-dist-topology-x*)
hence *openin m.mtopology* $(\bigcup \{\{x\} \mid x.\ x \in U\})$
by *auto*
moreover have $\bigcup \{\{x\} \mid x.\ x \in U\} = U$ **by** *blast*
ultimately show *openin m.mtopology U*
by *simp*
qed
qed

lemma *discrete-dist-topology'*:
metric-set.mtopology S (*discrete-dist S*) = *discrete-topology S*
by (*simp add*: *discrete-dist-topology topology-eq*)

Empty space.

lemma *empty-metric-compact*: *compact-metric-set* {} ($\lambda x\ y.\ 0$)
proof −
interpret *metric-set* {} $\lambda x\ y.\ 0$
by(*auto simp*: *metric-set-def*)
show *?thesis*
by *standard* (*use Hausdorff-space-finite-topspace*[*OF mtopology-Hausdorff*,*simplified mtopology-topspace*] **in** *blast*)
qed

corollary
shows *empty-metric-polish*: *polish-metric-set* {} ($\lambda x\ y.\ 0$)
and *empty-metric-complete*: *complete-metric-set* {} ($\lambda x\ y.\ 0$)
and *empty-metric-separable*: *separable-metric-set* {} ($\lambda x\ y.\ 0$)
and *empty-metric*: *metric-set* {} ($\lambda x\ y.\ 0$)
proof −
interpret *compact-metric-set* {} $\lambda x\ y.\ 0$
by(*rule empty-metric-compact*)
show *polish-metric-set* {} ($\lambda x\ y.\ 0$) *complete-metric-set* {} ($\lambda x\ y.\ 0$)
separable-metric-set {} ($\lambda x\ y.\ 0$) *metric-set* {} ($\lambda x\ y.\ 0$)
using *polish-metric-set-axioms complete-metric-set-axioms separable-metric-set-axioms metric-set-axioms*
by *blast*+
qed

lemma *empty-metric-unique*:
assumes *metric-set* {} *d*
shows *d* = ($\lambda x\ y.\ 0$)
apply *standard*+
using *assms* **by**(*auto simp*: *metric-set-def*)

lemma *empty-metric-mtopology*:
metric-set.mtopology {} ($\lambda x\ y.\ 0$) = *discrete-topology* {}

proof −
have *1*:$(\lambda U.\ U = \{\} \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.\ \textit{metric-set.open-ball}\ \{\}\ (\lambda x\ y.\ 0)\ x\ \varepsilon \subseteq U)) = (\lambda U.\ U = \{\})$
by *standard auto*
thus *?thesis*
using *metric-set.mtopology-def*[*of* {} $\lambda x\ y.\ 0$]
by(*simp add*: *metric-set-def discrete-topology-def 1*)
qed

Singleton space

lemma *singleton-metric-compact*:
compact-metric-set $\{a\}$ $(\lambda x\ y.\ 0)$
proof −
interpret *metric-set* $\{a\}$ $\lambda x\ y.\ 0$
by(*auto simp*: *metric-set-def*)
show *?thesis*
by *standard* (*use Hausdorff-space-finite-topspace*[*OF mtopology-Hausdorff*,*simplified mtopology-topspace*] **in** *blast*)
qed

corollary
shows *singleton-metric-polish*: *polish-metric-set* $\{a\}$ $(\lambda x\ y.\ 0)$
and *singleton-metric-complete*: *complete-metric-set* $\{a\}$ $(\lambda x\ y.\ 0)$
and *singleton-metric-separable*: *separable-metric-set* $\{a\}$ $(\lambda x\ y.\ 0)$
and *singleton-metric*: *metric-set* $\{a\}$ $(\lambda x\ y.\ 0)$
proof −
interpret *compact-metric-set* $\{a\}$ $\lambda x\ y.\ 0$
by(*rule singleton-metric-compact*)
show *polish-metric-set* $\{a\}$ $(\lambda x\ y.\ 0)$ *complete-metric-set* $\{a\}$ $(\lambda x\ y.\ 0)$
separable-metric-set $\{a\}$ $(\lambda x\ y.\ 0)$ *metric-set* $\{a\}$ $(\lambda x\ y.\ 0)$
using *polish-metric-set-axioms complete-metric-set-axioms separable-metric-set-axioms metric-set-axioms*
by *blast*+
qed

lemma *singleton-metric-unique*:
assumes *metric-set* $\{a\}$ *d*
shows $d = (\lambda x\ y.\ 0)$
by *standard*+ (*insert assms*,*auto simp*: *metric-set-def*, *metis*)

lemma *singleton-metric-mtopology*:
metric-set.mtopology $\{a\}$ $(\lambda x\ y.\ 0)$ = *discrete-topology* $\{a\}$
proof −
have $(\lambda U.\ U \subseteq \{a\} \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.\ \textit{metric-set.open-ball}\ \{a\}\ (\lambda x\ y.\ 0)\ x\ \varepsilon \subseteq U)) = (\lambda U.\ U \subseteq \{a\})$
proof
fix *U*
have ($U \subseteq \{a\} \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.\ \textit{metric-set.open-ball}\ \{a\}\ (\lambda x\ y.\ 0)\ x\ \varepsilon \subseteq U)$)
if $U \subseteq \{a\}$

proof *safe*
fix x
assume $x \in U$
then have $x = a$ **using** *that* **by** *auto*
thus $\exists \varepsilon{>}0.$ *metric-set.open-ball* $\{a\}$ $(\lambda x\ y.\ 0)$ $x\ \varepsilon \subseteq U$
by(*auto intro*!: *exI*[**where** *x=1*]) (*metis* ‹$x \in U$› *complete-metric-set-def empty-iff metric-set.open-ballD'*(*1*) *polish-metric-set-def singleton-metric-polish subset-singletonD that*)
qed(*use that* **in** *auto*)
thus $(U \subseteq \{a\} \wedge (\forall x{\in}U.\ \exists \varepsilon{>}0.$ *metric-set.open-ball* $\{a\}$ $(\lambda x\ y.\ 0)$ $x\ \varepsilon \subseteq U)) = (U \subseteq \{a\})$
by *auto*
qed
thus *?thesis*
using *metric-set.mtopology-def*[*of* $\{a\}$ $\lambda x\ y.\ 0$]
by(*simp add*: *metric-set-def discrete-topology-def*)
qed

2.3 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 *binary-distance* :: [$'a$ *set*, $'a \Rightarrow 'a \Rightarrow$ *real*, $'b$ *set*, $'b \Rightarrow 'b \Rightarrow$ *real*] $\Rightarrow 'a \times 'b \Rightarrow 'a \times 'b \Rightarrow$ *real* **where**
binary-distance S d S' d' $\equiv (\lambda(x,x')\ (y,y').$ *if* $(x,x') \in S \times S' \wedge (y,y') \in S \times S'$ *then d x y + d' x' y' else 0*)

context
fixes *S S' d d'*
assumes *metric-set S d metric-set S' d'*
begin

interpretation *m1*: *metric-set S d* **by** *fact*
interpretation *m2*: *metric-set S' d'* **by** *fact*

lemma *binary-metric-set*:
metric-set $(S \times S')$ (*binary-distance S d S' d'*)
proof
fix *x y z*
assume $x \in S \times S'$ $y \in S \times S'$ $z \in S \times S'$
then show *binary-distance S d S' d' x z* $\leq$ *binary-distance S d S' d' x y + binary-distance S d S' d' y z*
using *m1.dist-tr*[*of fst x fst y fst z*] *m2.dist-tr*[*of snd x snd y snd z*]
by(*fastforce simp*: *binary-distance-def split-beta'*)
next
show $\bigwedge x\ y.\ 0 \leq$ *binary-distance S d S' d' x y*
$\bigwedge x\ y.\ x \notin S \times S' \Longrightarrow$ *binary-distance S d S' d' x y = 0*
using *m1.dist-geq0 m2.dist-geq0 m1.dist-notin m2.dist-notin* **by**(*auto simp*:

binary-distance-def split-beta′)
next
fix *x y*
assume *x* ∈ *S* × *S′ y* ∈ *S* × *S′*
then show (*x* = *y*) = (*binary-distance S d S′ d′ x y* = *0*)
using *m1.dist-0*[*of fst x fst y*] *m2.dist-0*[*of snd x snd y*] *m1.dist-geq0*[*of fst x fst y*] *m2.dist-geq0*[*of snd x snd y*]
by(*auto simp*: *binary-distance-def split-beta*)
next
show ⋀*x y*. *binary-distance S d S′ d′ x y* = *binary-distance S d S′ d′ y x*
using *m1.dist-sym m2.dist-sym* **by**(*auto simp*: *binary-distance-def split-beta′*)
qed

interpretation *m*: *metric-set S* × *S′ binary-distance S d S′ d′*
by (*rule binary-metric-set*)

lemma *binary-distance-geq*:
assumes *x* ∈ *S y* ∈ *S x′* ∈ *S′ y′* ∈ *S′*
shows *d x y* ≤ *binary-distance S d S′ d′* (*x*,*x′*) (*y*,*y′*)
d′ x′ y′ ≤ *binary-distance S d S′ d′* (*x*,*x′*) (*y*,*y′*)
using *m1.dist-geq0 m2.dist-geq0 assms* **by**(*auto simp*: *binary-distance-def*)

lemma *binary-distance-ball*:
assumes (*x*,*x′*) ∈ *m.open-ball* (*a*,*a′*) *ε*
shows *x* ∈ *m1.open-ball a ε*
and *x′* ∈ *m2.open-ball a′ ε*
proof −
have *1*:*x* ∈ *S x′* ∈ *S′ ε* > *0 a* ∈ *S a′* ∈ *S′*
using *m.open-ballD′*[*OF assms*(*1*)] **by** *auto*
thus *x* ∈ *metric-set.open-ball S d a ε*
and *x′* ∈ *metric-set.open-ball S′ d′ a′ ε*
using *m.open-ballD*[*OF assms*(*1*)] *binary-distance-geq*[*OF 1*(*4*,*1*,*5*,*2*)] *1*
by(*auto simp*: *m1.open-ball-def m2.open-ball-def*)
qed

lemma *binary-distance-ball′*:
assumes *z* ∈ *m.open-ball a ε*
shows *fst z* ∈ *m1.open-ball* (*fst a*) *ε*
and *snd z* ∈ *m2.open-ball* (*snd a*) *ε*
using *binary-distance-ball*[*of fst z snd z fst a snd a ε*] *assms* **by** *auto*

lemma *binary-distance-ball1′*:
assumes *a* ∈ *S ε* > *0 a′*∈ *S′ ε′* > *0*
shows ∃*ε″*>*0*. *m.open-ball* (*a*,*a′*) *ε″* ⊆ *m1.open-ball a ε* × *m2.open-ball a′ ε′*
proof(*rule exI*[**where** *x*=*min ε ε′*])
show *0* < *min ε ε′* ∧ *m.open-ball* (*a*, *a′*) (*min ε ε′*) ⊆ *m1.open-ball a ε* × *m2.open-ball a′ ε′*
proof

```
    show 0 < min ε ε'
      using assms by auto
  next
    show m.open-ball (a, a') (min ε ε') ⊆ m1.open-ball a ε × m2.open-ball a' ε'
    proof safe
      fix x x'
      assume h:(x,x') ∈ m.open-ball (a, a') (min ε ε')
      then have hx:x ∈ S x' ∈ S'
        using m.open-ballD'(1)[of (x,x') (a, a') min ε ε'] by auto
      hence d a x + d' a' x' < min ε ε'
       using h assms by(auto simp: m.open-ball-def,auto simp: binary-distance-def)
      thus x ∈ m1.open-ball a ε x' ∈ m2.open-ball a' ε'
        using m1.dist-geq0[of a x] m2.dist-geq0[of a' x'] assms hx
        by(auto simp: m1.open-ball-def m2.open-ball-def)
    qed
  qed
qed

lemma binary-distance-ball1:
  assumes b ∈ m1.open-ball a ε b' ∈ m2.open-ball a' ε'
    shows ∃ε''>0. m.open-ball (b,b') ε'' ⊆ m1.open-ball a ε × m2.open-ball a' ε'
proof −
  obtain εa εa' where he:
    εa > 0 εa' > 0 m1.open-ball b εa ⊆ m1.open-ball a ε m2.open-ball b' εa' ⊆
m2.open-ball a' ε'
   using m1.mtopology-open-ball-in'[OF assms(1)] m2.mtopology-open-ball-in'[OF
assms(2)] by auto
  thus ?thesis
   using binary-distance-ball1'[OF m1.open-ballD'(1)[OF assms(1)] he(1) m2.open-ballD'(1)[OF
assms(2)] he(2)]
    by blast
qed


lemma binary-distance-ball2':
  assumes a ∈ S ε'' > 0 a'∈ S'
  shows ∃ε>0. ∃ε'>0. m1.open-ball a ε × m2.open-ball a' ε' ⊆ m.open-ball (a,a')
ε''
proof(safe intro!: exI[where x=ε''/2])
  fix x x'
  assume x ∈ m1.open-ball a (ε'' / 2) x' ∈ m2.open-ball a' (ε'' / 2)
  then have x ∈ S x' ∈ S' d a x < ε'' / 2 d' a' x' < ε'' / 2
    using assms by(auto simp: m1.open-ball-def m2.open-ball-def)
  thus (x,x') ∈ m.open-ball (a, a') ε''
    using assms by(auto simp: m.open-ball-def,auto simp: binary-distance-def)
qed (use assms in auto)

lemma binary-distance-ball2:
  assumes (b,b') ∈ m.open-ball (a,a') ε''
```

shows $\exists\varepsilon{>}0.\ \exists\varepsilon'{>}0.\ m1.open\text{-}ball\ b\ \varepsilon\ \times\ m2.open\text{-}ball\ b'\ \varepsilon' \subseteq m.open\text{-}ball\ (a,a')\ \varepsilon''$
proof −
obtain ε''' **where** $\varepsilon''' > 0\ m.open\text{-}ball\ (b,b')\ \varepsilon''' \subseteq m.open\text{-}ball\ (a,a')\ \varepsilon''$
using *m.mtopology-open-ball-in'*[*OF assms*(*1*)] **by** *blast*
thus *?thesis*
using *binary-distance-ball2'*[*of b* ε''' *b'*] *m.open-ballD'*[*OF assms*(*1*),*simplified*]
by *blast*
qed

lemma *binary-distance-mtopology*:
m.mtopology = *prod-topology m1.mtopology m2.mtopology*
proof −
have *m.mtopology* = *topology-generated-by* { *m1.open-ball a* ε × *m2.open-ball a'* ε' | *a a'* ε ε'. $a \in S \wedge a' \in S' \wedge \varepsilon > 0 \wedge \varepsilon' > 0$}
unfolding *m.mtopology-def2*
proof(*rule topology-generated-by-eq*)
fix *U*
assume $U \in \{m1.open\text{-}ball\ a\ \varepsilon \times m2.open\text{-}ball\ a'\ \varepsilon' \mid a\ a'\ \varepsilon\ \varepsilon'.\ a \in S \wedge a' \in S' \wedge 0 < \varepsilon \wedge 0 < \varepsilon'\}$
then obtain *a* ε *a'* ε' **where** *hae*:
$U = m1.open\text{-}ball\ a\ \varepsilon \times m2.open\text{-}ball\ a'\ \varepsilon'\ a \in S\ a' \in S'\ 0 < \varepsilon\ 0 < \varepsilon'$
by *auto*
show *openin* (*topology-generated-by* $\{m.open\text{-}ball\ a\ \varepsilon \mid a\ \varepsilon.\ a \in S \times S' \wedge 0 < \varepsilon\}$) *U*
unfolding *m.mtopology-def2*[*symmetric*] *m.mtopology-openin-iff hae*(*1*)
using *binary-distance-ball1*[*of - a* ε *- a'* ε'] *m1.open-ball-subset-ofS m2.open-ball-subset-ofS*
by *fastforce*
next
fix *U*
assume $U \in \{m.open\text{-}ball\ a\ \varepsilon \mid a\ \varepsilon.\ a \in S \times S' \wedge 0 < \varepsilon\}$
then obtain *a a'* ε **where** *hae*:
$U = m.open\text{-}ball\ (a,a')\ \varepsilon\ a \in S\ a' \in S'\ 0 < \varepsilon$
by *auto*
show *openin* (*topology-generated-by* $\{m1.open\text{-}ball\ a\ \varepsilon \times m2.open\text{-}ball\ a'\ \varepsilon' \mid a\ a'\ \varepsilon\ \varepsilon'.\ a \in S \wedge a' \in S' \wedge 0 < \varepsilon \wedge 0 < \varepsilon'\}$) *U*
unfolding *openin-subopen*[*of - m.open-ball* (a,a') ε] *hae*(*1*)
proof
fix *x*
assume *h*:$x \in m.open\text{-}ball\ (a,\ a')\ \varepsilon$
with *binary-distance-ball2*[*of fst x snd x a a'* ε]
obtain $\varepsilon'\ \varepsilon''$ **where** *he*:
$\varepsilon' > 0\ \varepsilon'' > 0\ m1.open\text{-}ball\ (fst\ x)\ \varepsilon' \times m2.open\text{-}ball\ (snd\ x)\ \varepsilon'' \subseteq m.open\text{-}ball\ (a,\ a')\ \varepsilon$
by *auto*
show $\exists T.\ openin\ (topology\text{-}generated\text{-}by\ \{m1.open\text{-}ball\ a\ \varepsilon \times m2.open\text{-}ball\ a'\ \varepsilon' \mid a\ a'\ \varepsilon\ \varepsilon'.\ a \in S \wedge a' \in S' \wedge 0 < \varepsilon \wedge 0 < \varepsilon'\})\ T \wedge x \in T \wedge T \subseteq m.open\text{-}ball\ (a,\ a')\ \varepsilon$
unfolding *openin-topology-generated-by-iff*

using *he m1.open-ball-ina*[*of fst x*,*OF - he*(*1*)] *m.open-ballD'*(*1*,*2*)[*OF h*] *m2.open-ball-ina*[*of snd x*,*OF - he*(*2*)]
by(*fastforce intro*!: *generate-topology-on.Basis exI*[**where** *x*=*m1.open-ball* (*fst x*) *ε'* × *m2.open-ball* (*snd x*) *ε''*] *exI*[**where** *x*=*fst x*] *exI*[**where** *x*=*snd x*])
qed
qed
also have ... = *prod-topology m1.mtopology m2.mtopology*
proof −
have {*m1.open-ball a ε* × *m2.open-ball a' ε'* |*a a' ε ε'*. *a* ∈ *S* ∧ *a'* ∈ *S'* ∧ *0* < *ε* ∧ *0* < *ε'*} = {*U* × *V* |*U V*. *U* ∈ {*m1.open-ball a ε* |*a ε*. *a* ∈ *S* ∧ *0* < *ε*} ∧ *V* ∈ {*m2.open-ball a ε* |*a ε*. *a* ∈ *S'* ∧ *0* < *ε*}}
by *blast*
thus *?thesis*
unfolding *m1.mtopology-def2 m2.mtopology-def2*
by(*simp only*: *prod-topology-generated-by*[*symmetric*])
qed
finally show *?thesis* **.**
qed

lemma *binary-distance-converge-to-inS-iff*:
m.converge-to-inS zn (*x*,*y*) ⟷ *m1.converge-to-inS* (λ*n*. *fst* (*zn n*)) *x* ∧ *m2.converge-to-inS* (λ*n*. *snd* (*zn n*)) *y*
proof *safe*
assume *m.converge-to-inS zn* (*x*, *y*)
then have *h*:*zn* ∈ *UNIV* → *S* × *S'* *x* ∈ *S* *y* ∈ *S'* ⋀*e*. *e*>*0* ⟹ ∃ *N*. ∀ *n*≥*N*. *zn n* ∈ *m.open-ball* (*x*, *y*) *e*
by(*auto simp*: *m.converge-to-inS-def2'*)
show *m1.converge-to-inS* (λ*n*. *fst* (*zn n*)) *x*
m2.converge-to-inS (λ*n*. *snd* (*zn n*)) *y*
unfolding *m1.converge-to-inS-def2' m2.converge-to-inS-def2'*
proof *safe*
fix *e* :: *real*
assume *e* > *0*
then obtain *N* **where** ⋀*n*. *n* ≥ *N* ⟹ *zn n* ∈ *m.open-ball* (*x*, *y*) *e*
using *h*(*4*) **by** *auto*
thus ∃ *N*. ∀ *n*≥*N*. *fst* (*zn n*) ∈ *m1.open-ball x e*
∃ *N*. ∀ *n*≥*N*. *snd* (*zn n*) ∈ *m2.open-ball y e*
using *binary-distance-ball'*[*of zn - (x,y)*]
by(*auto intro*!: *exI*[**where** *x*=*N*])
qed(*insert h*(*1*−*3*),*simp-all add*: *Pi-iff mem-Times-iff*)
next
assume *h*:*m1.converge-to-inS* (λ*n*. *fst* (*zn n*)) *x m2.converge-to-inS* (λ*n*. *snd* (*zn n*)) *y*
show *m.converge-to-inS zn* (*x*, *y*)
unfolding *m.converge-to-inS-def2'*
proof *safe*
show *goal1*:*x* ∈ *S* *y* ∈ *S'* *zn n* ∈ *S* × *S'* **for** *n*
using *h* **by**(*auto simp*: *m1.converge-to-inS-def m2.converge-to-inS-def Pi-iff mem-Times-iff*)

fix e :: *real*
assume $e > 0$
from *binary-distance-ball2'*[*OF goal1*(*1*) *this goal1*(*2*)]
obtain *e1 e2* **where** *e12*:$e1 > 0$ $e2 > 0$ $m1.open\text{-}ball\ x\ e1 \times m2.open\text{-}ball\ y\ e2 \subseteq m.open\text{-}ball\ (x, y)\ e$ **by** *auto*
then obtain *N1 N2* **where** *N12*: $\bigwedge n.\ n \geq N1 \Longrightarrow fst\ (zn\ n) \in m1.open\text{-}ball\ x\ e1$ $\bigwedge n.\ n \geq N2 \Longrightarrow snd\ (zn\ n) \in m2.open\text{-}ball\ y\ e2$
using *h* **by**(*auto simp*: *m1.converge-to-inS-def2' m2.converge-to-inS-def2'*) *metis*
with *e12* **have** $\bigwedge n.\ n \geq max\ N1\ N2 \Longrightarrow zn\ n \in m1.open\text{-}ball\ x\ e1 \times m2.open\text{-}ball\ y\ e2$
by (*simp add*: *mem-Times-iff*)
with *e12*(*3*) **show** $\exists N.\ \forall n{\geq}N.\ zn\ n \in m.open\text{-}ball\ (x, y)\ e$
by(*auto intro*!: *exI*[**where** $x{=}max\ N1\ N2$])
qed
qed

lemma *binary-distance-converge-to-inS-iff'*:
$m.converge\text{-}to\text{-}inS\ zn\ z \longleftrightarrow m1.converge\text{-}to\text{-}inS\ (\lambda n.\ fst\ (zn\ n))\ (fst\ z) \wedge m2.converge\text{-}to\text{-}inS\ (\lambda n.\ snd\ (zn\ n))\ (snd\ z)$
using *binary-distance-converge-to-inS-iff*[*of - fst z snd z*] **by** *simp*

corollary *binary-distance-convergent-inS-iff*:
$m.convergent\text{-}inS\ zn \longleftrightarrow m1.convergent\text{-}inS\ (\lambda n.\ fst\ (zn\ n)) \wedge m2.convergent\text{-}inS\ (\lambda n.\ snd\ (zn\ n))$
by(*auto simp*: *m.convergent-inS-def m1.convergent-inS-def m2.convergent-inS-def binary-distance-converge-to-inS-iff*)

lemma *binary-distance-Cauchy-inS-iff*:
$m.Cauchy\text{-}inS\ zn \longleftrightarrow m1.Cauchy\text{-}inS\ (\lambda n.\ fst\ (zn\ n)) \wedge m2.Cauchy\text{-}inS\ (\lambda n.\ snd\ (zn\ n))$
proof *safe*
assume *h*:$m.Cauchy\text{-}inS\ zn$
show $m1.Cauchy\text{-}inS\ (\lambda n.\ fst\ (zn\ n))$ $m2.Cauchy\text{-}inS\ (\lambda n.\ snd\ (zn\ n))$
unfolding *m1.Cauchy-inS-def2' m2.Cauchy-inS-def2'*
proof *safe*
fix e :: *real*
assume $e > 0$
then obtain $x\ y\ N$ **where** $x \in S$ $y \in S'$ $\bigwedge n.\ n \geq N \Longrightarrow zn\ n \in m.open\text{-}ball\ (x,y)\ e$
using *h* **by**(*auto simp*: *m.Cauchy-inS-def2'*) *metis*
thus $\exists x{\in}S.\ \exists N.\ \forall n{\geq}N.\ fst\ (zn\ n) \in m1.open\text{-}ball\ x\ e$
$\exists y{\in}S'.\ \exists N.\ \forall n{\geq}N.\ snd\ (zn\ n) \in m2.open\text{-}ball\ y\ e$
using *binary-distance-ball'*[*of zn - (x,y)*]
by(*auto intro*!: *exI*[**where** $x{=}x$] *exI*[**where** $x{=}y$] *exI*[**where** $x{=}N$]) *blast*
qed(*insert h, simp-all add*: *m.Cauchy-inS-def Pi-iff mem-Times-iff*)
next
assume *h*: *m1.Cauchy-inS* $(\lambda n.\ fst\ (zn\ n))$ *m2.Cauchy-inS* $(\lambda n.\ snd\ (zn\ n))$
show *m.Cauchy-inS zn*

```
    unfolding m.Cauchy-inS-def
  proof safe
    show 1:zn n ∈ S × S' for n
      using h(1,2) m1.Cauchy-inS-dest1 m2.Cauchy-inS-dest1 mem-Times-iff by
blast
    fix e :: real
    assume e > 0
   then obtain N1 N2 where N:⋀n m. n ≥ N1 ⟹ m ≥ N1 ⟹ d (fst (zn n))
(fst (zn m)) < e / 2 ⋀n m. n ≥ N2 ⟹ m ≥ N2 ⟹ d' (snd (zn n)) (snd (zn
m)) < e / 2
    by (metis h(1) h(2) less-divide-eq-numeral1(1) m1.Cauchy-inS-def m2.Cauchy-inS-def
mult-zero-left)
    show ∃ N. ∀ n≥N. ∀ m≥N. binary-distance S d S' d' (zn n) (zn m) < e
    proof(safe intro!: exI[where x=max N1 N2])
      fix n m
      assume max N1 N2 ≤ n max N1 N2 ≤ m
      then have le:N1 ≤ n N1 ≤ m N2 ≤ n N2 ≤ m by auto
      show binary-distance S d S' d' (zn n) (zn m) < e
        using N(1)[OF le(1,2)] N(2)[OF le(3,4)] ‹e > 0›
        by(auto simp: binary-distance-def split-beta')
    qed
  qed
qed

end

lemma binary-distance-separable:
  assumes separable-metric-set S d separable-metric-set S' d'
  shows separable-metric-set (S × S') (binary-distance S d S' d')
proof −
  interpret m1:separable-metric-set S d by fact
  interpret m2:separable-metric-set S' d' by fact
  interpret m : metric-set S × S' binary-distance S d S' d'
    by(auto intro!: binary-metric-set m1.metric-set-axioms m2.metric-set-axioms)
  show ?thesis
   using m.separable-iff-topological-separable separable-prod[OF m1.topological-separable-if-separable
m2.topological-separable-if-separable] binary-distance-mtopology[OF m1.metric-set-axioms
m2.metric-set-axioms]
    by auto
qed

lemma binary-distance-complete:
  assumes complete-metric-set S d complete-metric-set S' d'
  shows complete-metric-set (S × S') (binary-distance S d S' d')
proof −
  interpret m1:complete-metric-set S d by fact
  interpret m2:complete-metric-set S' d' by fact
  interpret m : metric-set S × S' binary-distance S d S' d'
    by(auto intro!: binary-metric-set m1.metric-set-axioms m2.metric-set-axioms)
```

show *?thesis*
by *standard* (*simp add*: *binary-distance-Cauchy-inS-iff*[*OF m1.metric-set-axioms m2.metric-set-axioms*] *binary-distance-convergent-inS-iff*[*OF m1.metric-set-axioms m2.metric-set-axioms*] *m1.convergence m2.convergence*)
qed

lemma *binary-distance-polish*:
assumes *polish-metric-set S d* **and** *polish-metric-set S′ d′*
shows *polish-metric-set* (*S*×*S′*) (*binary-distance S d S′ d′*)
using *assms* **by**(*simp add*: *polish-metric-set-def binary-distance-separable binary-distance-complete*)

2.4 Sum Metric Spaces

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

abbreviation *S* ≡ ⋃*i*∈*I*. *Si i*

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

definition *sum-dist* :: [*'a*, *'a*] ⇒ *real* **where**
sum-dist x y ≡ (*if x* ∈ *S* ∧ *y* ∈ *S then* (*if* ∃ *i*∈*I*. *x* ∈ *Si i* ∧ *y* ∈ *Si i then di* (*THE i*. *i* ∈ *I* ∧ *x* ∈ *Si i* ∧ *y* ∈ *Si i*) *x y else 1*) *else 0*)

lemma *sum-dist-simps*:
shows ⋀*i*. ⟦*i* ∈ *I*; *x* ∈ *Si i*; *y* ∈ *Si i*⟧ ⟹ *sum-dist x y* = *di i x y*
and ⋀*i j*. ⟦*i* ∈ *I*; *j* ∈ *I*; *i* ≠ *j*; *x* ∈ *Si i*; *y* ∈ *Si j*⟧ ⟹ *sum-dist x y* = *1*
and ⋀*i*. ⟦*i* ∈ *I*; *y* ∈ *S*; *x* ∈ *Si i*; *y* ∉ *Si i*⟧ ⟹ *sum-dist x y* = *1*
and ⋀*i*. ⟦*i* ∈ *I*; *x* ∈ *S*; *y* ∈ *Si i*; *x* ∉ *Si i*⟧ ⟹ *sum-dist x y* = *1*
and *x* ∉ *S* ⟹ *sum-dist x y* = *0*
proof −
{ fix *i*
assume *h*:*i* ∈ *I x* ∈ *Si i y* ∈ *Si i*
then have *sum-dist x y* = *di* (*THE i*. *i* ∈ *I* ∧ *x* ∈ *Si i* ∧ *y* ∈ *Si i*) *x y*
by(*auto simp*: *sum-dist-def*)
also have ... = *di i x y*
proof −
have (*THE i*. *i* ∈ *I* ∧ *x* ∈ *Si i* ∧ *y* ∈ *Si i*) = *i*
using *disj-fam 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 ∈ Si i; y ∈ Si j⟧ ⟹ sum-dist x y = 1
      ⋀i. ⟦i ∈ I; y ∈ S; x ∈ Si i; y ∉ Si i⟧ ⟹ sum-dist x y = 1 ⋀i. ⟦i ∈ I; x ∈
S; y ∈ Si i; x ∉ Si i⟧ ⟹ sum-dist x y = 1
      x ∉ S ⟹ sum-dist x y = 0
  using disj-fam by(auto simp: sum-dist-def disjoint-family-on-def dest:Si-inj-on)
qed

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

lemma inS-cases:
 assumes x ∈ S y ∈ S
    and ⋀i. ⟦i ∈ I; x ∈ Si i; y ∈ Si i⟧ ⟹ P x y
    and ⋀i j. ⟦i ∈ I; j ∈ I; i ≠ j; x ∈ Si i; y ∈ Si j; x ≠ y⟧ ⟹ P x y
  shows P x y using assms by auto

sublocale metric-set S sum-dist
proof
 fix x y
 assume x ∈ S y ∈ S
 then show x = y ⟷ sum-dist x y = 0
  by(rule inS-cases, insert sd-metric) (auto simp: sum-dist-simps metric-set-def)
next
 { fix i x y
  assume h: i ∈ I x ∈ Si i y ∈ Si i
  then have sum-dist x y = di i x y
            sum-dist y x = di i x y
    using sd-metric by(auto simp: sum-dist-simps metric-set-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 ∈ S y ∈ S z ∈ S
 show sum-dist x z ≤ sum-dist x y + sum-dist y z (is ?lhs ≤ ?rhs)
 proof(rule inS-cases[OF h(1,3)])
  fix i
  assume h':i ∈ I x ∈ Si i z ∈ Si i
  consider y ∈ Si i | y ∉ Si i by auto
  thus ?lhs ≤ ?rhs
  proof cases
   case 1
   with h' sd-metric [OF h'(1)]show ?thesis
```

```
      by(simp add: sum-dist-simps metric-set-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 ∈ Si i z ∈ Si j
    consider y ∉ Si i | y ∉ Si j
      using h' h(2) disj-fam 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(simp add: sum-dist-simps)

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 ∈ Si i
  shows metric-set.open-ball (Si i) (di i) x e = open-ball x e
proof −
  interpret m: metric-set Si i di i
    by(simp add: assms sd-metric)
  show ?thesis
     using assms sum-dist-simps(1)[OF assms(1) assms(3)] sum-dist-if-less1[OF
assms(1,3)]
    by(auto simp: m.open-ball-def open-ball-def) fastforce+
qed

lemma ball-le-sum-dist-ball:
  assumes i ∈ I
  shows metric-set.open-ball (Si i) (di i) x e ⊆ open-ball x e
proof −
  interpret m: metric-set Si i di i
    by(simp add: assms sd-metric)
  show ?thesis
  proof safe
    fix y
    assume y: y ∈ m.open-ball x e
    show y ∈ open-ball x e
      using m.open-ballD[OF y] m.open-ballD'[OF y] assms
      by(auto simp: open-ball-def sum-dist-simps)
  qed
qed
```

lemma *openin-sum-mtopology-iff*:
openin mtopology U $\longleftrightarrow$ *U* $\subseteq$ *S* $\wedge$ ($\forall$ *i*$\in$*I*. *openin* (*metric-set.mtopology* (*Si i*) (*di i*)) (*U* $\cap$ *Si i*))
proof *safe*
fix *i*
assume *h*:*openin mtopology U i* $\in$ *I*
then interpret *m*: *metric-set Si i di i*
using *sd-metric* **by** *simp*
show *openin m.mtopology* (*U* $\cap$ *Si i*)
unfolding *m.mtopology-openin-iff*
proof *safe*
fix *x*
assume *x*:*x* $\in$ *U x* $\in$ *Si i*
with *h* **obtain** *e* **where** *e*: *e* > *0 open-ball x e* $\subseteq$ *U*
by(*auto simp*: *mtopology-openin-iff*)
show $\exists$*ε*>*0*. *m.open-ball x ε* $\subseteq$ *U* $\cap$ *Si i*
proof(*safe intro*!: *exI*[**where** *x*=*e*])
fix *y*
assume *y* $\in$ *m.open-ball x e*
from *m.open-ballD*[*OF this*] *x*(*2*) *m.open-ballD'*(*1*)[*OF this*] *h*(*2*)
have *y* $\in$ *open-ball x e*
by(*auto simp*: *open-ball-def sum-dist-simps*)
with *e* **show** *y* $\in$ *U* **by** *auto*
qed(*use e m.open-ball-subset-ofS* **in** *auto*)
qed
next
show $\bigwedge$*x*. *openin mtopology U* $\Longrightarrow$ *x* $\in$ *U* $\Longrightarrow$ *x* $\in$ *S*
by(*auto simp*: *mtopology-openin-iff*)
next
assume *h*: *U* $\subseteq$ *S* $\forall$ *i*$\in$*I*. *openin* (*metric-set.mtopology* (*Si i*) (*di i*)) (*U* $\cap$ *Si i*)
show *openin mtopology U*
unfolding *mtopology-openin-iff*
proof *safe*
fix *x*
assume *x*: *x* $\in$ *U*
then obtain *i* **where** *i*: *i* $\in$ *I x* $\in$ *Si i*
using *h*(*1*) **by** *auto*
then interpret *m*: *metric-set Si i di i*
using *sd-metric* **by** *simp*
obtain *e* **where** *e*: *e* > *0 m.open-ball x e* $\subseteq$ *U* $\cap$ *Si i*
using *i h*(*2*) **by** (*meson IntI m.mtopology-openin-iff x*)
show $\exists$*ε*>*0*. *open-ball x ε* $\subseteq$ *U*
proof(*safe intro*!: *exI*[**where** *x*=*min e 1*])
fix *y*
assume *y*:*y* $\in$ *open-ball x* (*min e 1*)
then show *y* $\in$ *U*
using *sum-dist-ball-eq-ball*[*OF i*(*1*) - *i*(*2*),*of min e 1*] *e m.open-ball-le*[*of min e 1 e x*]
by *auto*

qed(*simp add*: *e*)
qed(*use h*(*1*) **in** *auto*)
qed

corollary *openin-sum-mtopology-Si*:
assumes $i \in I$
shows *openin mtopology* (*Si i*)
unfolding *openin-sum-mtopology-iff*
proof *safe*
fix *j*
assume *j*:$j \in I$
then interpret *m*: *metric-set Si j di j*
by(*simp add*: *sd-metric*)
show *openin m.mtopology* (*Si i* ∩ *Si j*)
by (*cases i* = *j*, *insert assms disj-fam j*) (*auto simp*: *disjoint-family-on-def*)
qed(*use assms* **in** *auto*)

lemma *converge-to-inSi-converge-to-inS*:
assumes $i \in I$ *metric-set.converge-to-inS* (*Si i*) (*di i*) *xn x*
shows *converge-to-inS xn x*
proof −
interpret *m*: *metric-set Si i di i*
by(*simp add*: *assms sd-metric*)
{
fix *e* :: *real*
assume *e* > *0*
then obtain *N* **where** $\bigwedge n.\ n \geq N \Longrightarrow xn\ n \in m.open\text{-}ball\ x\ e$
using *assms*(*2*) **by**(*auto simp*: *m.converge-to-inS-def2′*) *metis*
hence $\exists N.\ \forall n{\geq}N.\ xn\ n \in open\text{-}ball\ x\ e$
using *ball-le-sum-dist-ball*[*OF assms*(*1*),*of x e*]
by(*auto intro*!: *exI*[**where** *x*=*N*]) **}**
thus *?thesis*
using *assms* **by**(*auto simp*: *m.converge-to-inS-def2′ converge-to-inS-def2′*)
qed

corollary *convergent-inSi-convergent-inS*:
assumes $i \in I$ *metric-set.convergent-inS* (*Si i*) (*di i*) *xn*
shows *convergent-inS xn*
using *converge-to-inSi-converge-to-inS*[*OF assms*(*1*)] *assms*(*1*) *assms*(*2*) *convergent-inS-def metric-set.the-limit-if-converge sd-metric*
by *blast*

lemma *converge-to-inS-converge-to-inSi-off-set*:
assumes *converge-to-inS xn x*
shows $\exists n.\ \exists j{\in}I.$ *metric-set.converge-to-inS* (*Si j*) (*di j*) (λ*i*. *xn* (*i* + *n*)) *x*
proof −
obtain *i* **where** *i*: $i \in I\ x \in Si\ i$
using *assms* **by**(*auto simp*: *converge-to-inS-def*)
then interpret *m*: *metric-set Si i di i*

```
    by(simp add: sd-metric)
  obtain N where N: ⋀n. n ≥ N ⟹ sum-dist (xn n) x < 1
    using assms by(fastforce simp: converge-to-inS-def2)
  hence N': n ≥ N ⟹ xn n ∈ Si i for n
    using assms by(auto intro!: sum-dist-if-less1[OF i,of xn n] simp: dist-sym[of -
x] converge-to-inS-def)
  show ?thesis
  proof(safe intro!: exI[where x=N] bexI[OF - i(1)])
    show m.converge-to-inS (λi. xn (i + N)) x
      unfolding m.converge-to-inS-def2
    proof(safe intro!: N' i(2))
      fix e :: real
      assume 0 < e
      then obtain M where M: ⋀n. n ≥ M ⟹ sum-dist (xn n) x < e
        using assms by(fastforce simp: converge-to-inS-def2)
      hence n ≥ max N M ⟹ di i (xn n) x < e for n
        using sum-dist-simps(1)[OF i(1) N'[of n] i(2),symmetric] by auto
      thus ∃ M. ∀ n≥M. di i (xn (n + N)) x < e
        by(auto intro!: exI[where x=M])
    qed simp
  qed
qed

corollary convergent-inS-convergent-inSi-off-set:
  assumes convergent-inS xn
  shows ∃ n. ∃ j∈I. metric-set.convergent-inS (Si j) (di j) (λi. xn (i + n))
  using converge-to-inS-converge-to-inSi-off-set
  by (meson assms metric-set.convergent-inS-def metric-set-axioms sd-metric)


lemma Cauchy-inSi-Cauchy-inS:
  assumes i ∈ I metric-set.Cauchy-inS (Si i) (di i)xn
  shows Cauchy-inS xn
proof −
  interpret m: metric-set Si i di i
    by(simp add: assms sd-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)]
    by(auto simp: m.Cauchy-inS-def Cauchy-inS-def)
  show ?thesis
    using assms by(auto simp: m.Cauchy-inS-def Cauchy-inS-def)
qed

lemma Cauchy-inS-Cauchy-inSi:
  assumes Cauchy-inS xn
  shows ∃ n. ∃ j∈I. metric-set.Cauchy-inS (Si j) (di j) (λi. xn (i + n))
proof −
  obtain x i N where xiN: i ∈ I x ∈ Si i ⋀n. n ≥ N ⟹ xn n ∈ open-ball x 1
    using assms by(auto simp: Cauchy-inS-def2') (metis UNION-empty-conv(2)
```

d-bounded d-nonneg dist-0 empty-subsetI less-eq-real-def open-ball-le-0 subsetI subset-antisym sum-dist-le1)
then interpret *m*: *metric-set Si i di i*
by(*simp add*: *sd-metric*)
have *xn*: $n \geq N \Longrightarrow xn\ n \in Si\ i$ **for** *n*
using *xiN*(*3*)[*of n*] **by**(*auto simp*: *sum-dist-ball-eq-ball*[*OF xiN*(*1*) *order-refl xiN*(*2*),*symmetric*] *dest*: *m.open-ballD'*)
show *?thesis*
proof(*safe intro*!: *exI*[**where** *x*=*N*] *bexI*[*OF - xiN*(*1*)])
show *m.Cauchy-inS* ($\lambda i.\ xn\ (i + N)$)
unfolding *m.Cauchy-inS-def*
proof *safe*
fix *e* :: *real*
assume $0 < e$
then obtain *M* **where** *M*: $\bigwedge n\ m.\ n \geq M \Longrightarrow m \geq M \Longrightarrow$ *sum-dist* (*xn n*) (*xn m*) $< e$
using *assms* **by**(*auto simp*: *Cauchy-inS-def*) *metis*
have [*simp*]: $n \geq N \Longrightarrow m \geq N \Longrightarrow$ *di i* (*xn n*) (*xn m*) = *sum-dist* (*xn n*) (*xn m*) **for** *n m*
using *xn sum-dist-simps*(*1*)[*OF xiN*(*1*) *xn*[*of n*] *xn*[*of m*]] **by** *simp*
show $\exists N'.\ \forall n{\geq}N'.\ \forall m{\geq}N'.$ *di i* (*xn* ($n + N$)) (*xn* ($m + N$)) $< e$
using *M* **by**(*auto intro*!: *exI*[**where** *x*=*max N M*])
qed(*use xn* **in** *auto*)
qed
qed

end

lemma *sum-metricI*:
fixes *Si*
assumes *disjoint-family-on Si I*
and $\bigwedge i\ x\ y.\ i \notin I \Longrightarrow 0 \leq di\ i\ x\ y$
and $\bigwedge i\ x\ y.\ di\ i\ x\ y < 1$
and $\bigwedge i.\ i \in I \Longrightarrow$ *metric-set* (*Si i*) (*di i*)
shows *sum-metric I Si di*
using *assms* **by**(*auto simp*: *sum-metric-def*) (*meson metric-set.dist-geq0*)

locale *sum-separable-metric* = *sum-metric* +
assumes *I*: *countable I*
and *sd-separable-metric*: $\bigwedge i.\ i \in I \Longrightarrow$ *separable-metric-set* (*Si i*) (*di i*)
begin

sublocale *separable-metric-set S sum-dist*
proof
obtain *Ui* **where** *Ui*: $\bigwedge i.\ i \in I \Longrightarrow$ *countable* (*Ui i*) $\bigwedge i.\ i \in I \Longrightarrow$ *metric-set.dense-set* (*Si i*) (*di i*) (*Ui i*)
using *sd-separable-metric* **by**(*auto simp*: *separable-metric-set-def separable-metric-set-axioms-def*) *metis*
define *U* **where** $U \equiv \bigcup i{\in}I.\ Ui\ i$

```
  show ∃ U. countable U ∧ dense-set U
  proof(safe intro!: exI[where x=U])
    show countable U
      using Ui(1) I by(auto simp: U-def)
  next
    show dense-set U
      unfolding dense-set-def U-def
    proof safe
      fix i x
      assume i ∈ I x ∈ Ui i
      then show x ∈ S
        using sd-separable-metric Ui by(auto intro!: bexI[where x=i] simp: sepa-
rable-metric-set-def metric-set.dense-set-def)
    next
      fix i x e
      assume h:i ∈ I x ∈ Si i (0 :: real) < e open-ball x e ∩ ⋃ (Ui ‘ I) = {}
      then interpret sd: separable-metric-set Si i di i
        by(simp add: sd-separable-metric)
      have sd.open-ball x e ∩ Ui i ≠ {}
        using Ui(2)[OF h(1)] h(1−3) by(auto simp: U-def sd.dense-set-def)
      hence sd.open-ball 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

end

locale sum-complete-metric = sum-metric +
  assumes sd-complete-metric: ⋀i. i ∈ I ⟹ complete-metric-set (Si i) (di i)
begin

sublocale complete-metric-set S sum-dist
proof
  fix xn
  assume 1:Cauchy-inS xn
  from Cauchy-inS-Cauchy-inSi[OF this] obtain n j where h: j ∈ I metric-set.Cauchy-inS
(Si j) (di j) (λi. xn (i + n))
    by auto
  then have metric-set.convergent-inS (Si j) (di j) (λi. xn (i + n))
    by (simp add: complete-metric-set.convergence sd-complete-metric)
  from convergent-inS-offset[OF convergent-inSi-convergent-inS[OF h(1) this]] 1
  show convergent-inS xn
    by(simp add: Cauchy-inS-def)
qed

end
```

locale *sum-polish-metric* = *sum-complete-metric* + *sum-separable-metric*
begin

sublocale *polish-metric-set S sum-dist*
by (*simp add*: *complete-metric-set-axioms polish-metric-set-def separable-metric-set-axioms*)

end

lemma *sum-polish-metricI*:
fixes *Si*
assumes *countable I*
and *disjoint-family-on Si I*
and $\bigwedge i\ x\ y.\ i \notin I \Longrightarrow 0 \leq di\ i\ x\ y$
and $\bigwedge i\ x\ y.\ di\ i\ x\ y < 1$
and $\bigwedge i.\ i \in I \Longrightarrow$ *polish-metric-set* (*Si i*) (*di i*)
shows *sum-polish-metric I Si di*
using *assms* **by**(*auto simp*: *sum-polish-metric-def sum-complete-metric-def sum-separable-metric-def sum-complete-metric-axioms-def sum-separable-metric-axioms-def polish-metric-set-def complete-metric-set-def sum-metricI*)

end

2.5 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 $(\sum n.\ r\hat{}(n + k) * K) = r\hat{}k\ /\ (1 - r) * K$
(**is** *?lhs* = -)
proof −
have *?lhs* = $(\sum n.\ r\hat{}n * K) - (\sum n{\in}\{..<k\}.\ r\hat{}n * K)$
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 $(\sum n{\in}\{..<k\}.\ r\hat{}n * K) = (1 - r\hat{}k)\ /\ (1 - r) * K$
using *one-diff-power-eq*[*of r k*] *r scale-sum-left*[*of* $\lambda n.\ r\hat{}n$ $\{..<k\}$ *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 :: $nat \Rightarrow real$
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 :: $nat \Rightarrow real$
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 ... $= (\sum n.\ r\hat{}(n + Suc\ n' + k) * a'\ (n + Suc\ n' + l)) + (\sum i<Suc\ n'.\ r\hat{}(i + k) * a'\ (i + l))$
by(*auto simp*: *a'-def*)

also have ... = $(\sum n.\ r\hat{}(n + k) * a'\ (n + l))$
by(*rule suminf-split-initial-segment*[*symmetric*]) *simp*
also have ... $\leq r\hat{}k\ /\ (1 - r) * K$
by(*rule nsum-of-r-leq*[*OF r a'*])
finally show *?thesis* **.**
qed

definition *product-dist'* :: [*real*, *'i set*, *nat* $\Rightarrow$ *'i*, *'i* $\Rightarrow$ *'a set*, *'i* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *real*] $\Rightarrow$ (*'i* $\Rightarrow$ *'a*) $\Rightarrow$ (*'i* $\Rightarrow$ *'a*) $\Rightarrow$ *real* **where**
product-dist-def: *product-dist' r I g S d* $\equiv$ ($\lambda x\ y.$ *if* $x \in (\Pi_E\ i \in I.\ S\ i) \wedge y \in (\Pi_E\ i \in I.\ S\ i)$ *then* $(\sum n.$ *if g n* $\in I$ *then* $r\hat{}n * d\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *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* $\Rightarrow$ *nat*
and *g* :: *nat* $\Rightarrow$ *'i*
and *S* :: *'i* $\Rightarrow$ *'a set*
and *d* :: *'i* $\Rightarrow$ *'a* $\Rightarrow$ *'a* $\Rightarrow$ *real*
and *K* :: *real*
assumes *r*: *0 < r r < 1*
and *I*: *countable I*
and *gf-comp-id* : $\bigwedge i.\ i \in I \Longrightarrow g\ (f\ i) = i$
and *gf-if-finite*: *finite I* $\Longrightarrow$ *bij-betw f I* {*..<* *card I*}
finite I $\Longrightarrow$ *bij-betw g* {*..<* *card I*} *I*
and *gf-if-infinite*: *infinite I* $\Longrightarrow$ *bij-betw f I UNIV*
infinite I $\Longrightarrow$ *bij-betw g UNIV I*
$\bigwedge n.$ *infinite I* $\Longrightarrow f\ (g\ n) = n$
and *sd-metric*: $\bigwedge i.\ i \in I \Longrightarrow$ *metric-set* (*S i*) (*d i*)
and *d-nonneg*: $\bigwedge i\ x\ y.\ 0 \leq d\ i\ x\ y$
and *d-bound*: $\bigwedge i\ x\ y.\ d\ i\ x\ y \leq K$
and *K-pos* : *0 < K*

lemma *from-nat-into-to-nat-on-product-metric-pair*:
assumes *countable I*
shows $\bigwedge i.\ i \in I \Longrightarrow$ *from-nat-into I* (*to-nat-on I i*) = *i*
and *finite I* $\Longrightarrow$ *bij-betw* (*to-nat-on I*) *I* {*..<* *card I*}
and *finite I* $\Longrightarrow$ *bij-betw* (*from-nat-into I*) {*..<* *card I*} *I*
and *infinite I* $\Longrightarrow$ *bij-betw* (*to-nat-on I*) *I UNIV*
and *infinite I* $\Longrightarrow$ *bij-betw* (*from-nat-into I*) *UNIV I*
and $\bigwedge n.$ *infinite I* $\Longrightarrow$ *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 S d*

lemma *nsum-of-rK*: (∑ *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* ∗ *d* (*g n*) (*x* (*g n*)) (*y* (*g n*)))
apply(*rule summable-comparison-test'*[*of* λ*n*. *r*^*n* ∗ *K*])
using *r d-nonneg d-bound K-pos* **by**(*auto intro*!: *summable-mult2*)

lemma *product-dist-summable*[*simp*]:
summable (λ*n*. *if g n* ∈ *I then r*^*n* ∗ *d* (*g n*) (*x* (*g n*)) (*y* (*g n*)) *else 0*)

by(*rule summable-comparison-test'*[*of* $\lambda n.\ r\hat{}n * d\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$]) (*use r d-nonneg d-bound K-pos* **in** *auto*)

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

lemma *product-dist-distance*: *metric-set* $(\Pi_E\ i{\in}I.\ S\ i)$ *product-dist*
proof −
have *h'*: $\bigwedge i\ xi\ yi.\ i \in I \Longrightarrow xi \in S\ i \Longrightarrow yi \in S\ i \Longrightarrow xi = yi \longleftrightarrow d\ i\ xi\ yi = 0$
$\bigwedge i\ xi\ yi.\ i \in I \Longrightarrow d\ i\ xi\ yi = d\ i\ yi\ xi$
$\bigwedge i\ xi\ yi\ zi.\ i \in I \Longrightarrow xi \in S\ i \Longrightarrow yi \in S\ i \Longrightarrow zi \in S\ i \Longrightarrow d\ i\ xi\ zi \leq d\ i\ xi\ yi + d\ i\ yi\ zi$
using *sd-metric* **by**(*auto simp*: *metric-set-def*)
show *?thesis*
proof
show $\bigwedge x\ y.\ 0 \leq$ *product-dist x y*
using *d-nonneg r* **by**(*auto simp*: *product-dist-def intro*!: *suminf-nonneg product-dist-summable*)
next
show $\bigwedge x\ y.\ x \notin (\Pi_E\ i{\in}I.\ S\ i) \Longrightarrow$ *product-dist x y* $= 0$
by(*auto simp*: *product-dist-def*)
next
fix *x y*
assume *hxy*:$x \in (\Pi_E\ i{\in}I.\ S\ i)\ y \in (\Pi_E\ i{\in}I.\ S\ i)$
show $(x = y) \longleftrightarrow$ (*product-dist x y* $= 0$)
proof
assume *heq*:$x = y$
then have (*if g n* $\in I$ *then r* $\hat{}$ *n* $*$ *d* $(g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *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* $\hat{}$ *n* $*$ *d* $(g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *else* $0) = 0$
$\longleftrightarrow (\forall n.$ (*if g n* $\in I$ *then r*$\hat{}$*n* $*$ *d* $(g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *else* $0) = 0$)
apply(*rule suminf-eq-zero-iff*)
using *d-nonneg r* **by**(*auto simp*: *product-dist-def intro*!: *product-dist-summable*)
hence *hn0*:$\bigwedge n.$ (*if g n* $\in I$ *then r*$\hat{}$*n* $*$ *d* $(g\ n)\ (x\ (g\ n))\ (y\ (g\ n))$ *else* $0) = 0$
using *h0 hxy* **by**(*auto simp*: *product-dist-def*)
show $x = y$
proof
fix *i*
consider $i \in I \mid i \notin I$ **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 d (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 ∈ (Π_E i∈I. S i) y ∈ (Π_E i∈I. S i) z ∈ (Π_E i∈I. S i)
    have (if g n ∈ I then r ^ n * d (g n) (x (g n)) (z (g n)) else 0)
          ≤ (if g n ∈ I then r ^ n * d (g n) (x (g n)) (y (g n)) else 0) + (if g n ∈
I then r ^ n * d (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 ^ n,symmetric])
    thus product-dist x z ≤ product-dist x y + product-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 metric-set Π_E i∈I. S i product-dist
  by(rule product-dist-distance)

lemma product-dist-leqr: product-dist x y ≤ 1 / (1 − r) * K
proof −
  have product-dist x y ≤ (∑ n. if g n ∈ I then r^n * d (g n) (x (g n)) (y (g n))
else 0)
  proof −
    consider x ∈ (Π_E i∈I. S i) ∧ y ∈ (Π_E i∈I. S i) | ¬ (x ∈ (Π_E i∈I. S i) ∧ y
∈ (Π_E i∈I. S 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 ... ≤ (∑ n. if g n ∈ I then r^n * d (g n) (x (g n)) (y (g n)) else 0)
      using d-nonneg r by(auto intro!: suminf-nonneg product-dist-summable)
```

```
    finally show ?thesis .
  qed
 qed
 also have ... ≤ (∑ n. r^n * d (g n) (x (g n)) (y (g n)))
  using r d-nonneg d-bound by(auto intro!: suminf-le)
 also have ... ≤ (∑ n. r^n * K)
  using r d-bound d-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. S i) y ∈ (Π_E i∈I. S i)
  shows d i (x i) (y i) ≤ (1/r)^n * product-dist x y
      (is ?lhs ≤ ?rhs)
proof −
 interpret mi: metric-set S i d i
  by(rule sd-metric[OF assms(1)])
 have (λm. if m = f i then d (g m) (x (g m)) (y (g m)) else 0) sums d (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 d (g m) (x (g m)) (y (g m)) else
0)
            ?lhs = (∑ m. (if m = f i then d (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 * d (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 * d (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 d (g k) (x (g k)) (y (g k)) else 0) ≤ (1/r) ^ n * (if g k
```

$\in I$ *then* $r \hat{} k * d\ (g\ k)\ (x\ (g\ k))\ (y\ (g\ k))$ *else* 0)
using $*$ *d-nonneg r* $**$ *mult-right-mono*[*OF* $**$] **by**(*auto simp*: *vector-space-over-itself.scale-scale*[*of* $(1\ /\ r) \hat{} n$])
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 *converge-to-iff*:
assumes *xn* $\in$ *sequence* $x \in (\Pi_E\ i{\in}I.\ S\ i)$
shows *converge-to-inS xn x* $\longleftrightarrow$ $(\forall\, i{\in}I.$ *metric-set.converge-to-inS* $(S\ i)\ (d\ i)$ $(\lambda n.\ xn\ n\ i)\ (x\ i))$
proof *safe*
fix *i*
assume *h*:*converge-to-inS xn x i* $\in$ *I*
then interpret *m*: *metric-set S i d i*
using *sd-metric* **by** *blast*
show *m.converge-to-inS* $(\lambda n.\ xn\ n\ i)\ (x\ i)$
unfolding *m.converge-to-inS-def2*
proof *safe*
show *1*:$\bigwedge x.$ *xn x i* $\in$ *S i x i* $\in$ *S i*
using *h* **by**(*auto simp*: *converge-to-inS-def*)
next
fix ε :: *real*
assume $0 < \varepsilon$
then obtain $r \hat{} f\ i * \varepsilon > 0$ **using** *r* **by** *auto*
then obtain *N* **where** *N*:$\bigwedge n.\ n \geq N \Longrightarrow$ *product-dist* $(xn\ n)\ x < r \hat{} f\ i * \varepsilon$
using *h*(*1*) **by**(*auto simp*: *converge-to-inS-def2*) *metis*
show $\exists\, N.\ \forall\, n{\geq}N.\ d\ i\ (xn\ n\ i)\ (x\ i) < \varepsilon$
proof(*safe intro*!: *exI*[**where** $x{=}N$])
fix *n*
assume $N \leq n$
have $d\ i\ (xn\ n\ i)\ (x\ i) \leq (1\ /\ r) \hat{} f\ i *$ *product-dist* $(xn\ n)\ x$
using *h* **by**(*auto intro*!: *product-dist-geq*[*OF h*(*2*) *gf-comp-id*[*OF h*(*2*)]] *simp*: *converge-to-inS-def*)
also have ... $< (1\ /\ r) \hat{} f\ i * r \hat{} f\ i * \varepsilon$
using *N*[*OF* $\langle N \leq n \rangle$] *r* **by** *auto*
also have ... $\leq \varepsilon$
by (*simp add*: $\langle 0 < \varepsilon \rangle$ *power-one-over*)
finally show $d\ i\ (xn\ n\ i)\ (x\ i) < \varepsilon$ **.**
qed
qed
next
assume *h*:$\forall\, i{\in}I.$ *metric-set.converge-to-inS* $(S\ i)\ (d\ i)\ (\lambda n.\ xn\ n\ i)\ (x\ i)$
show *converge-to-inS xn x*
unfolding *converge-to-inS-def2*
proof *safe*

fix ε
assume *he*:$(0::real) < \varepsilon$
then have $0 < \varepsilon*((1-r)/K)$ **using** *r K-pos* **by** *auto*
hence $\exists k.\ r\hat{}k < \varepsilon*((1-r)/K)$
using *r*(2) *real-arch-pow-inv* **by** *blast*
then obtain *l* **where** $r\hat{}l < \varepsilon*((1-r)/K)$ **by** *auto*
hence *hk*:$r\hat{}l/(1-r)*K < \varepsilon$
using *mult-imp-div-pos-less*[*OF divide-pos-pos*[*OF - K-pos*,*of* $1-r$]] *r*(2) **by** *simp*
hence *hke*: $0 < \varepsilon - r\hat{}l/(1-r)*K$ **by** *auto*
consider $l = 0 \mid 0 < l$ **by** *auto*
then show $\exists N.\ \forall n{\geq}N.$ *product-dist* $(xn\ n)\ x < \varepsilon$
proof *cases*
case *1*
then have *he2*:$1\ /\ (1\ -\ r)*K < \varepsilon$ **using** *hk* **by** *auto*
show *?thesis*
using *order.strict-trans1*[*OF product-dist-leqr he2*]
by(*auto simp*: *complete-metric-set-def intro*!: *exI*[**where** $x=0$])
next
case *2*
with *hke* **have** $0 < 1\ /\ real\ l * (\varepsilon - r\hat{}l/(1-r)*K)$ **by** *auto*
hence $\forall i{\in}I.\ \exists N.\ \forall n{\geq}N.\ d\ i\ (xn\ n\ i)\ (x\ i) < 1\ /\ real\ l * (\varepsilon - r\hat{}l/(1-r)*K)$
using *h metric-set.converge-to-inS-def2*[*OF sd-metric*] **by** *auto*
then obtain *N* **where** *hn*:
$\bigwedge i\ n.\ i \in I \Longrightarrow n \geq N\ i \Longrightarrow d\ i\ (xn\ n\ i)\ (x\ i) < 1\ /\ real\ l * (\varepsilon - r\hat{}l/(1-r)*K)$
by *metis*
show *?thesis*
proof(*safe intro*!: *exI*[**where** x=*Sup* $\{N\ (g\ n) \mid n.\ n < l\}$])
fix *n*
assume *hsup*:$\bigsqcup\ \{N\ (g\ n)\ |n.\ n < l\} \leq n$
have *product-dist* $(xn\ n)\ x = (\sum m.$ *if* $g\ m \in I$ *then* $r\ \hat{}\ m * d\ (g\ m)\ (xn\ n\ (g\ m))\ (x\ (g\ m))$ *else* $0)$
using *assms* **by**(*auto simp*: *product-dist-def*)
also have $\ldots = (\sum m.$ *if* $g\ (m + l) \in I$ *then* $r\ \hat{}\ (m + l)*\ d\ (g\ (m + l))\ (xn\ n\ (g\ (m + l)))\ (x\ (g\ (m + l)))$ *else* $0) + (\sum m{<}l.$ *if* $g\ m \in I$ *then* $r\ \hat{}\ m * d\ (g\ m)\ (xn\ n\ (g\ m))\ (x\ (g\ m))$ *else* $0)$
by(*auto intro*!: *suminf-split-initial-segment*)
also have $\ldots \leq r\hat{}l/(1-r)*K + (\sum m{<}l.$ *if* $g\ m \in I$ *then* $r\ \hat{}\ m * d\ (g\ m)\ (xn\ n\ (g\ m))\ (x(g\ m))$ *else* $0)$
proof −
have $(\sum m.$ *if* $g\ (m + l) \in I$ *then* $r\ \hat{}\ (m + l)*\ d\ (g\ (m + l))\ (xn\ n\ (g\ (m + l)))\ (x\ (g\ (m + l)))$ *else* $0) \leq (\sum m.\ r\hat{}(m + l)*K)$
using *d-bound assms r K-pos* **by**(*auto intro*!: *suminf-le summable-ignore-initial-segment*)
also have $\ldots = r\hat{}l/(1-r)*K$
by(*rule nsum-of-rK*)
finally show *?thesis* **by** *auto*
qed
also have $\ldots \leq r\hat{}l\ /\ (1\ -\ r)*K + (\sum m{<}l.$ *if* $g\ m \in I$ *then* $d\ (g\ m)\ (xn\ n\ (g\ m))\ (x\ (g\ m))$ *else* $0)$

```
      proof −
        have (∑ m<l. if g m ∈ I then r ^ m * d (g m) (xn n (g m)) (x (g m))
else 0) ≤ (∑ m<l. if g m ∈ I then d (g m) (xn n (g m)) (x (g m)) else 0)
        using d-bound d-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 + (∑ m<l. 1 / real l * (ε − r^l/(1−r)*K))
      proof −
        have (∑ m<l. if g m ∈ I then d (g m) (xn n (g m)) (x (g m)) else 0) <
(∑ 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 d (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 d (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 d (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 d (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
 qed
 qed (use assms in auto)
qed

lemma product-dist-mtopology: product-topology (λi. metric-set.mtopology (S i) (d
i)) I = mtopology
proof −
 have htopspace:⋀i. i ∈ I ⟹ topspace (metric-set.mtopology (S i) (d i)) = S i
  by (simp add: sd-metric metric-set.mtopology-topspace)
 hence htopspace':(Π_E i∈I. topspace (metric-set.mtopology (S i) (d i))) = (Π_E
i∈I. S i) by auto
 consider I = {} | I ≠ {} by auto
 then show ?thesis
 proof cases
  case 1
  then have product-dist  = (λx y. 0)
   using metric-set-axioms by(simp add: singleton-metric-unique)
  thus ?thesis
   by(simp add: product-topology-empty-discrete 1 singleton-metric-mtopology)
 next
  case I':2
  show ?thesis
   unfolding mtopology-def2 product-topology-def
  proof(rule topology-generated-by-eq)
   fix U
   assume U ∈ {open-ball a ε |a ε. a ∈ (Π_E i∈I. S i) ∧ 0 < ε}
   then obtain a ε where hu:
    U = open-ball a ε a ∈ (Π_E i∈I. S i) 0 < ε by auto
     have ∃X. x ∈ (Π_E i∈I. X i) ∧ (Π_E i∈I. X i) ⊆ U ∧ (∀i. openin
(metric-set.mtopology (S i) (d i)) (X i)) ∧ finite {i. X i ≠ topspace (metric-set.mtopology
(S i) (d i))} if x ∈ U for x
   proof −
    consider ε ≤ 1 / (1 − r) * K | 1 / (1 − r) * K < ε by fastforce
      then show ∃X. x ∈ (Π_E i∈I. X i) ∧ (Π_E i∈I. X i) ⊆ U ∧  (∀i.
openin (metric-set.mtopology (S i) (d i)) (X i)) ∧  finite {i. X i ≠ topspace
(metric-set.mtopology (S i) (d i))}
    proof cases
     case he2:1
       note hx = open-ballD[OF that[simplified hu(1)]] open-ballD'(1)[OF
that[simplified hu(1)]]
```

then have $0 < (\varepsilon - \textit{product-dist a x}) * ((1-r)/\ K)$ **using** r *hu K-pos* **by** *auto*

hence $\exists k.\ r\hat{}k < (\varepsilon - \textit{product-dist a x}) * ((1-r)/\ K)$
using $r(2)$ *real-arch-pow-inv* **by** *blast*

then obtain k **where** $r\hat{}k < (\varepsilon - \textit{product-dist a x}) * ((1-r)/\ K)$ **by** *auto*

hence $hk{:}r\hat{}k\ /\ (1-r) * K < (\varepsilon - \textit{product-dist a x})$
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 dist-geq0*[*of a x*] **by** *auto*

define ε' **where** $\varepsilon' \equiv (1/(\textit{real } k)) * (\varepsilon - \textit{product-dist a x} - r\hat{}k\ /\ (1-r) * K)$

have $h\varepsilon'$: $0 < \varepsilon'$ **using** *hk* **by**(*auto simp*: ε'*-def hk'*)

define X **where** $X \equiv (\textit{if finite } I \textit{ then } (\lambda i.\ \textit{if } i \in I \textit{ then metric-set.open-ball } (S\ i)\ (d\ i)\ (x\ i)\ \varepsilon' \textit{ else topspace } (\textit{metric-set.mtopology } (S\ i)\ (d\ i))) \textit{ else } (\lambda i.\ \textit{if } i \in I \wedge f\ i < k \textit{ then metric-set.open-ball } (S\ i)\ (d\ i)\ (x\ i)\ \varepsilon' \textit{ else topspace } (\textit{metric-set.mtopology } (S\ i)\ (d\ i))))$

show *?thesis*

proof(*intro exI*[**where** $x{=}X$] *conjI*)

have $x\ i \in \textit{metric-set.open-ball } (S\ i)\ (d\ i)\ (x\ i)\ \varepsilon'$ **if** $i \in I$ **for** i
using $hx(2)$ **by** (*simp add*: *PiE-mem* $h\varepsilon'$ *metric-set.open-ball-ina sd-metric that*)

thus $x \in (\Pi_E\ i{\in}I.\ X\ i)$
using $hx(2)$ *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 $\bigwedge i.\ X\ i \subseteq \textit{topspace } (\textit{metric-set.mtopology } (S\ i)\ (d\ i))$
by (*simp add*: *X-def sd-metric htopspace metric-set.open-ball-subset-ofS*)

hence $y \in (\Pi_E\ i{\in}I.\ S\ i)$
using $\textit{htopspace}'\ \langle y \in (\Pi_E\ i{\in}I.\ X\ i)\rangle$ **by** *blast*

have $\textit{product-dist a y} < \varepsilon$

proof −

have $\textit{product-dist a y} \leq \textit{product-dist a x} + \textit{product-dist x y}$
by(*rule dist-tr*[*OF hu*(2) $hx(2)$ $\langle y \in (\Pi_E\ i{\in}I.\ S\ i)\rangle$])

also have $\ldots < \textit{product-dist a x} + (\varepsilon - \textit{product-dist a x})$

proof −

have $\textit{product-dist x y} < (\varepsilon - \textit{product-dist a x})$

proof −

have $\textit{product-dist x y} = (\sum n.\ \textit{if } g\ n \in I \textit{ then } r\ \hat{}\ n * d\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n)) \textit{ else } 0)$
by(*simp add*: *product-dist-def hx* $\langle y \in (\Pi_E\ i{\in}I.\ S\ i)\rangle$)

also have $\ldots = (\sum n.\ \textit{if } g\ (n+k) \in I \textit{ then } r\ \hat{}\ (n+k) * d\ (g\ (n+k))\ (x\ (g\ (n+k)))\ (y\ (g\ (n+k))) \textit{ else } 0) + (\sum n{<}k.\ \textit{if } g\ n \in I \textit{ then } r\ \hat{}\ n * d\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n)) \textit{ else } 0)$
by(*rule suminf-split-initial-segment*) *simp*

also have $\ldots \leq r\hat{}k\ /\ (1-r) * K + (\sum n{<}k.\ \textit{if } g\ n \in I \textit{ then } r\ \hat{}\ n * d\ (g\ n)\ (x\ (g\ n))\ (y\ (g\ n)) \textit{ else } 0)$

```
                proof −
                  have (∑ n. if g (n + k) ∈ I then r ^ (n + k)∗ d (g (n + k)) (x
(g (n + k))) (y (g (n + k))) else 0) ≤ (∑ n. r ^ (n + k) ∗ K)
                    using d-bound d-nonneg r K-pos by(auto intro!: suminf-le
summable-ignore-initial-segment)
                  also have ... = r^k / (1 − r) ∗ K
                    by(rule nsum-of-rK)
                  finally show ?thesis by simp
                qed
               also have ... < r^k / (1 − r) ∗ K + (ε − product-dist a x − r^k
/ (1 − r) ∗ K)
                proof −
                  have (∑ n<k. if g n ∈ I then r ^ n ∗ d (g n) (x (g n)) (y (g n))
else 0) < (∑ n<k. ε′)
                    proof(rule sum-strict-mono-ex1)
                      show ∀ l∈{..<k}. (if g l ∈ I then r ^ l ∗ d (g l) (x (g l)) (y (g
l)) else 0) ≤ ε′
                        proof −
                          {
                            fix l
                            assume g l ∈ I l < k
                            then interpret mbd: metric-set S (g l) d (g l)
                              by(auto intro!: sd-metric)
                            have r ^ l ∗ d (g l) (x (g l)) (y (g l)) ≤ d (g l) (x (g l)) (y
(g l))
                              using r by(auto intro!: mult-right-mono[of r ^ l 1,OF -
mbd.dist-geq0[of x (g l) y (g l)],simplified] simp: power-le-one)
                            also have ... < ε′
                            proof −
                              have y (g l) ∈ mbd.open-ball (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 dest: mbd.open-ballD)
                            qed
                            finally have r ^ l ∗ d (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 * d (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-set S (g 0) d (g 0)
                     by(auto intro!: sd-metric)
                   have y (g 0) ∈ mbd.open-ball (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 * d (g 0) (x (g 0)) (y (g 0))  < ε'
                     by(auto dest: mbd.open-ballD)
                 }
                 thus (if g 0 ∈ I then r ^ 0 * d (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) open-ball-def hu(2) ‹y ∈ (Π_E i∈I. S i)›)
    qed
  next
    have openin (metric-set.mtopology (S i) (d i)) (metric-set.open-ball (S
i) (d i) (x i) ε') if i ∈ I for i
    by (meson PiE-E hε' hx(2) metric-set.mtopology-open-ball-in sd-metric
that)
     moreover have openin (metric-set.mtopology (S i) (d i)) (topspace
```

```
(metric-set.mtopology (S i) (d i))) for i
          by auto
        ultimately show ∀ i. openin (metric-set.mtopology (S i) (d i)) (X i)
          by(auto simp: X-def)
      next
        show finite {i. X i ≠ topspace (metric-set.mtopology (S i) (d 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 = (Π_E i∈I. S i)
          unfolding hu(1) using order.strict-trans1[OF product-dist-leqr,of ε]
hu(2)
        by(simp add: open-ball-def)
      also have ... = (Π_E i∈I. topspace (metric-set.mtopology (S i) (d i)))
        using htopspace by auto
```

finally have $U = (\Pi_E\ i{\in}I.\ topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i)))$ **.**
thus *?thesis*
using *open-ballD'(1)*[*OF that*[*simplified hu(1)*]] *htopspace* **by**(*auto intro*!: *exI*[**where** $x{=}\lambda i.\ topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))$])
qed
qed
hence $\exists X.\ \forall x{\in}U.\ x \in (\Pi_E\ i{\in}I.\ X\ x\ i) \wedge (\Pi_E\ i{\in}I.\ X\ x\ i) \subseteq U \wedge (\forall i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ x\ i)) \wedge finite\ \{i.\ X\ x\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}$
by(*auto intro*!: *bchoice*)
then obtain X **where** $\forall x{\in}U.\ x \in (\Pi_E\ i{\in}I.\ X\ x\ i) \wedge (\Pi_E\ i{\in}I.\ X\ x\ i) \subseteq U \wedge (\forall i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ x\ i)) \wedge finite\ \{i.\ X\ x\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}$
by *auto*
hence *hX*: $\bigwedge x.\ x \in U \Longrightarrow x \in (\Pi_E\ i{\in}I.\ X\ x\ i)\ \bigwedge x.\ x \in U \Longrightarrow (\Pi_E\ i{\in}I.\ X\ x\ i) \subseteq U\ \bigwedge x.\ x \in U \Longrightarrow (\forall i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ x\ i))\ \bigwedge x.\ x \in U \Longrightarrow finite\ \{i.\ X\ x\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}$
by *auto*
hence *hXopen*: $\bigwedge x.\ x \in U \Longrightarrow (\Pi_E\ i{\in}I.\ X\ x\ i) \in \{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}\}$
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* $\{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}\}$) $(\bigcup\ \{(\Pi_E\ i{\in}I.\ X\ x\ i) \mid x.\ x \in U\})$
apply(*rule openin-Union*)
using *hXopen* **by**(*auto simp*: *openin-topology-generated-by-iff intro*!: *generate-topology-on.Basis*)
thus *openin* (*topology-generated-by* $\{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}\}$) *U*
using $\langle U = (\bigcup\ \{(\Pi_E\ i{\in}I.\ X\ x\ i) \mid x.\ x \in U\})\rangle$ **by** *simp*
next
fix *U*
assume $U \in \{\Pi_E\ i{\in}I.\ X\ i\ |X.\ (\forall i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}\}$
then obtain X **where** *hX*:
$U = (\Pi_E\ i{\in}I.\ X\ i)\ \bigwedge i.\ openin\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\ (X\ i)\ finite\ \{i.\ X\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\}$
by *auto*
have $\exists a\ \varepsilon.\ x \in open\text{-}ball\ a\ \varepsilon \wedge open\text{-}ball\ a\ \varepsilon \subseteq U$ **if** $x \in U$ **for** x
proof −
have *x-intop*:$x \in (\Pi_E\ i{\in}I.\ S\ i)$
unfolding *htopspace'*[*symmetric*] **using** *that hX*(*1*) *openin-subset*[*OF hX*(*2*)] **by** *auto*
define I' **where** $I' \equiv \{i.\ X\ i \neq topspace\ (metric\text{-}set.mtopology\ (S\ i)\ (d\ i))\} \cap I$

```
      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-set.mtopology (S i) (d
i))
          by(auto simp: I'-def)
        then have U = (Π_E i∈I. S i)
          by (simp add: PiE-eq hX(1) htopspace)
        thus ?thesis
          using open-ball-subset-ofS[of x 1] that
          by(auto intro!: exI[where x=x] exI[where x=1])
      next
        case I'-nonempty:2
        hence ⋀i. i ∈ I' ⟹ openin (metric-set.mtopology (S i) (d i)) (X i)
          using hX(2) by(simp add: I'-def)
       hence ∃ε>0. metric-set.open-ball (S i) (d i) (x i) ε ⊆ (X i) if i ∈ I' for i
            using metric-set.mtopology-openin-iff[of S i d i X i] sd-metric[of i]
hX(1,2) ‹x ∈ U› that
          using I'-def by blast
          then obtain εi' where hei:⋀i. i ∈ I' ⟹ εi' i > 0 ⋀i. i ∈ I' ⟹
metric-set.open-ball (S i) (d 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 ∈ open-ball x (ε * r ^ n)
          have y i ∈ X i if i ∈ I' for i
          proof −
            interpret mi: metric-set S i d i
              using sd-metric that by(simp add: I'-def)
            have d i (x i) (y i) < εi' i
            proof −
              have d i (x i) (y i) ≤ (1 / r) ^ f i * product-dist x y
                  using that by(auto intro!: product-dist-geq[of i,OF  - gf-comp-id
x-intop open-ballD'(1)[OF ‹y ∈ open-ball x (ε * r ^ n)›]] simp: I'-def)
```

```
            also have ... ≤ (1 / r) ^ n * product-dist x y
              by(rule mult-right-mono[OF hn2[OF that] dist-geq0])
            also have ... < ε
              using open-ballD[OF ‹y ∈ open-ball 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.open-ball (x i) (εi' i)
            using open-ballD'(1)[OF ‹y ∈ open-ball x (ε * r ^ n)›] x-intop that
            by(auto simp: mi.open-ball-def 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 open-ballD'(1)[OF ‹y ∈ open-ball x (ε * r ^ n)›]
          by(auto simp: I'-def)
        ultimately show y ∈ U
          using open-ballD'(1)[OF ‹y ∈ open-ball x (ε * r ^ n)›]
          by(auto simp: hX(1))
      qed(use open-ball-ina[OF x-intop hε'] in auto)
    qed
  qed
  then obtain a where ∀x∈U. ∃ε. x ∈ open-ball (a x) ε ∧ open-ball (a x) ε
⊆ U
    by metis
  then obtain ε where hae: ⋀x. x ∈ U ⟹ x ∈ open-ball (a x) (ε x) ⋀x. x
∈ U ⟹ open-ball (a x) (ε x) ⊆ U
    by metis
  hence hae': ⋀x. x ∈ U ⟹ a x ∈ (Π_E i∈I. S i) ⋀x. x ∈ U ⟹ 0 < ε x
    using open-ballD'(2) by meson (use open-ballD'(2,3) hae in meson)
  have openin (topology-generated-by {open-ball a ε |a ε. a ∈ (Π_E i∈I. S i) ∧
0 < ε}) (⋃ { open-ball (a x) (ε x) |x. x ∈ U})
  by(auto intro!: openin-Union[of - mtopology] simp: mtopology-def2[symmetric]
hae' metric-set-axioms metric-set.mtopology-open-ball-in)
  moreover have U = (⋃ {open-ball (a x) (ε x) |x. x ∈ U})
    using hae by auto
  ultimately show openin (topology-generated-by {open-ball a ε |a ε. a ∈ (Π_E
i∈I. S i) ∧ 0 < ε}) U
    by simp
 qed
qed

end

lemma product-metricI:
  assumes 0 < r r < 1 countable I ⋀i. i ∈ I ⟹ metric-set (S i) (d i)
```

and $\bigwedge i\ x\ y.\ 0 \leq d\ i\ x\ y\ \bigwedge i\ x\ y.\ d\ i\ x\ y \leq K\ 0 < K$
shows *product-metric r I* (*to-nat-on I*) (*from-nat-into I*) *S d K*
using *from-nat-into-to-nat-on-product-metric-pair*[*OF assms*(*3*)] *assms*
by(*simp add*: *product-metric-def metric-set-def*)

Case: all (S_i, d_i) are separable metric spaces.

locale *product-separable-metric* = *product-metric* +
assumes *sd-separable-metric*: $\bigwedge i.\ i \in I \Longrightarrow$ *separable-metric-set* (*S i*) (*d i*)
begin

sublocale *separable-metric-set* Π_E *i*∈*I*. *S i product-dist*
proof −
have $\bigwedge i.\ i \in I \Longrightarrow$ *second-countable* (*metric-set.mtopology* (*S i*) (*d i*))
by (*simp add*: *sd-separable-metric separable-metric-set.second-countable*)
hence *second-countable* (*product-topology* (λ*i*. *metric-set.mtopology* (*S i*) (*d i*)) *I*)
by(*rule product-topology-second-countable*[*OF I*])
hence *second-countable* (*metric-set.mtopology* (*Pi*$_E$ *I S*) *product-dist*)
using *product-dist-mtopology sd-metric*
by(*simp add*: *separable-metric-set-def*)
thus *separable-metric-set* (Π_E *i*∈*I*. *S i*) *product-dist*
by (*meson I d-bound d-nonneg metric-set.separable-if-second-countable product-dist-distance r*(*1*) *r*(*2*) *sd-metric second-countable-def separable-metric-set.axioms*(*1*))
qed

end

Case: all (S_i, d_i) are complete metric spaces.

locale *product-complete-metric* = *product-metric* +
assumes *sd-complete-metric*: $\bigwedge i.\ i \in I \Longrightarrow$ *complete-metric-set* (*S i*) (*d i*)
begin

lemma *product-dist-complete′*:
assumes $I \neq \{\}$
shows *complete-metric-set* (Π_E *i*∈*I*. *S i*) *product-dist*
proof −
show *?thesis*
proof
fix *k*
assume *h*:*Cauchy-inS k*
have ∗:*i* ∈ *I* ⟹ *metric-set.Cauchy-inS* (*S i*) (*d i*) (λ*n*. *k n i*) **for** *i*
proof −
assume *hi*:*i* ∈ *I*
then interpret *mi*: *complete-metric-set S i d i*
by(*simp add*: *sd-complete-metric*)
show *mi.Cauchy-inS* (λ*n*. *k n i*)
unfolding *mi.Cauchy-inS-def2″*
proof
show (λ*n*. *k n i*) ∈ *mi.sequence*

```
      using h hi by(auto simp: Cauchy-inS-def)
    next
     show ∀ε>0. ∃ x∈S i. ∃ N. ∀ n≥N. d i x (k n i) < ε
     proof safe
      fix ε
      assume he:(0::real) < ε
      then have 0 < ε * r^(f i) using r by auto
      then obtain x N where hxn:
       x∈(Π_E i∈I. S i) ⋀n. n≥N ⟹ product-dist x (k n) < ε * r^(f i)
        using h[simplified Cauchy-inS-def2''] by blast
      hence hxn':⋀n. n≥N ⟹ (1/r)^(f i) * product-dist x (k n) < ε
        by (simp add: pos-divide-less-eq power-divide r(1))
      show ∃ x∈S i. ∃ N. ∀ n≥N. d i x (k n i) < ε
      proof(safe intro!: bexI[where x=x i] exI[where x=N])
        show x i ∈ S i
          using hi hxn by auto
      next
        fix n
        assume hnn:N ≤ n
        have hf:k n ∈ (Π_E i∈I. S i)
          using h by(auto simp: Cauchy-inS-def)
        have d i (x i) (k n i) ≤ (1/r)^(f i) * product-dist x (k n)
          using product-dist-geq[OF hi gf-comp-id[OF hi] hxn(1) hf]
          by simp
        also have ... < ε
          using hxn'[OF hnn] .
        finally show d i (x i) (k n i) < ε .
      qed
     qed
    qed
    have ∃ x. metric-set.converge-to-inS (S i) (d i) (λn. k n i) x if i ∈ I for i
     using complete-metric-set.convergence[OF sd-complete-metric[OF that] *[OF
that]] metric-set.convergent-inS-def[OF sd-metric[OF that]]
     by auto
    then obtain x where hx:⋀i. i ∈ I ⟹ metric-set.converge-to-inS (S i) (d i)
(λn. k n i) (x i)
     by metis
    have hx':(λi∈I. x i) ∈ (Π_E i∈I. S i)
     using hx metric-set.converge-to-inS-def[OF sd-metric] by auto
    show convergent-inS k
     using converge-to-iff[OF - hx',of k]
     by(auto intro!: exI[where x=λi∈I. x i] simp: h[simplified Cauchy-inS-def] hx
convergent-inS-def)
  qed
qed

sublocale complete-metric-set Π_E i∈I. S i product-dist
proof −
```

consider $I = \{\} \mid I \neq \{\}$ **by** *auto*
then show *complete-metric-set* $(\Pi_E\ i{\in}I.\ S\ i)$ *product-dist*
proof *cases*
case *1*
then have *product-dist* $= (\lambda x\ y.\ 0)$
using *metric-set-axioms singleton-metric-unique*[*of* $\lambda x.$ *undefined*] **by** *auto*
with *1 singleton-metric-polish*[*of* $\lambda x.$ *undefined*]
show *?thesis* **by**(*auto simp*: *polish-metric-set-def*)
next
case *2*
with *product-dist-complete'* **show** *?thesis* **by** *simp*
qed
qed

end

lemma *product-complete-metricI*:
assumes $0 < r\ r < 1$ *countable* $I\ \bigwedge i.\ i \in I \Longrightarrow$ *complete-metric-set* $(S\ i)\ (d\ i)$
and $\bigwedge i\ x\ y.\ 0 \leq d\ i\ x\ y\ \bigwedge i\ x\ y.\ d\ i\ x\ y \leq K\ 0 < K$
shows *product-complete-metric* $r\ I$ (*to-nat-on* I) (*from-nat-into* I) $S\ d\ K$
using *from-nat-into-to-nat-on-product-metric-pair*[*OF assms*(*3*)] *assms*
by(*simp add*: *product-complete-metric-def product-metric-def product-complete-metric-axioms-def complete-metric-set-def*)

lemma *product-complete-metric-natI*:
assumes $0 < r\ r < 1\ \bigwedge n.$ *complete-metric-set* $(S\ n)\ (d\ n)$
and $\bigwedge i\ x\ y.\ 0 \leq d\ i\ x\ y\ \bigwedge i\ x\ y.\ d\ i\ x\ y \leq K\ 0 < K$
shows *product-complete-metric* r *UNIV id id* $S\ d\ K$
using *assms* **by**(*simp add*: *product-complete-metric-def product-metric-def product-complete-metric-axioms-def polish-metric-set-def complete-metric-set-def*)

locale *product-polish-metric* = *product-complete-metric* + *product-separable-metric*
begin

sublocale *polish-metric-set* $\Pi_E\ i{\in}I.\ S\ i$ *product-dist*
by (*simp add*: *complete-metric-set-axioms polish-metric-set-def separable-metric-set-axioms*)

end

lemma *product-polish-metricI*:
assumes $0 < r\ r < 1$ *countable* $I\ \bigwedge i.\ i \in I \Longrightarrow$ *polish-metric-set* $(S\ i)\ (d\ i)$
and $\bigwedge i\ x\ y.\ 0 \leq d\ i\ x\ y\ \bigwedge i\ x\ y.\ d\ i\ x\ y \leq K\ 0 < K$
shows *product-polish-metric* $r\ I$ (*to-nat-on* I) (*from-nat-into* I) $S\ d\ K$
using *from-nat-into-to-nat-on-product-metric-pair*[*OF assms*(*3*)] *assms*
by(*simp add*: *product-polish-metric-def product-complete-metric-def product-separable-metric-def product-metric-def product-complete-metric-axioms-def product-separable-metric-axioms-def polish-metric-set-def complete-metric-set-def*)

lemma *product-polish-metric-natI*:

assumes $0 < r\ r < 1\ \bigwedge n.$ *polish-metric-set* $(S\ n)\ (d\ n)$
and $\bigwedge i\ x\ y.\ 0 \leq d\ i\ x\ y\ \bigwedge i\ x\ y.\ d\ i\ x\ y \leq K\ 0 < K$
shows *product-polish-metric r UNIV id id S d K*
using *assms* **by**(*simp add*: *product-polish-metric-def product-complete-metric-def product-separable-metric-def product-metric-def product-complete-metric-axioms-def product-separable-metric-axioms-def polish-metric-set-def complete-metric-set-def*)

Define a bounded distance function from a distance function

definition *bounded-dist* :: $('a \Rightarrow 'a \Rightarrow real) \Rightarrow 'a \Rightarrow 'a \Rightarrow real$ **where**
bounded-dist $d \equiv (\lambda a\ b.\ d\ a\ b\ /\ (1 + d\ a\ b))$

lemma *bounded-dist-mono*:
fixes $r\ l$ :: *real*
assumes $0 \leq r\ 0 \leq l$ **and** $r \leq l$
shows $r\ /\ (1 + r) \leq l\ /\ (1 + l)$
proof −
have $(1 + l) * r \leq l * (1 + r)$
using *assms* **by** (*simp add*: *distrib-left distrib-right*)
hence $((1 + l) * r) * (1\ /\ (1 + r)) \leq (l * (1 + r)) * (1\ /\ (1 + r))$
using *linordered-ring-strict-class.mult-le-cancel-right*[*of* $(1 + l) * r\ 1\ /\ (1 + r)\ l * (1 + r)$] *assms*(*1*)
by *auto*
hence $(1\ /\ (1 + l)) * (((1 + l) * r) * (1\ /\ (1 + r))) \leq (1\ /\ (1 + l)) * ((l * (1 + r)) * (1\ /\ (1 + r)))$
using *linordered-ring-strict-class.mult-le-cancel-left*[*of* $1\ /\ (1 + l)\ ((1 + l) * r) * (1\ /\ (1 + r))\ (l * (1 + r)) * (1\ /\ (1 + r))$] *assms*(*2*)
by *auto*
thus *?thesis*
using *assms* **by** *auto*
qed

lemma *bounded-dist-mono-strict*:
fixes $r\ l$ :: *real*
assumes $0 \leq r\ 0 \leq l$ **and** $r < l$
shows $r\ /\ (1 + r) < l\ /\ (1 + l)$
proof −
have $(1 + l) * r < l * (1 + r)$
using *assms* **by** (*simp add*: *distrib-left distrib-right*)
hence $((1 + l) * r) * (1\ /\ (1 + r)) < (l * (1 + r)) * (1\ /\ (1 + r))$
using *linordered-ring-strict-class.mult-less-cancel-right*[*of* $(1 + l) * r\ 1\ /\ (1 + r)\ l * (1 + r)$] *assms*(*1*)
by *auto*
hence $(1\ /\ (1 + l)) * (((1 + l) * r) * (1\ /\ (1 + r))) < (1\ /\ (1 + l)) * ((l * (1 + r)) * (1\ /\ (1 + r)))$
using *linordered-ring-strict-class.mult-less-cancel-left*[*of* $1\ /\ (1 + l)\ ((1 + l) * r) * (1\ /\ (1 + r))\ (l * (1 + r)) * (1\ /\ (1 + r))$] *assms*(*2*)
by *auto*
thus *?thesis*
using *assms* **by** *auto*

qed

lemma *bounded-dist-mono-inverse*:
fixes $r\ l :: real$
assumes $0 \leq r\ 0 \leq l$ **and** $r\ /\ (1 + r) \leq l\ /\ (1 + l)$
shows $r \leq l$
proof −
have $(1\ /\ (1 + l)) * (((1 + l) * r) * (1\ /\ (1 + r))) \leq (1\ /\ (1 + l)) * ((l * (1 + r))\ * (1\ /\ (1 + r)))$
using *assms* **by** *auto*
hence $((1 + l) * r) * (1\ /\ (1 + r)) \leq (l * (1 + r))\ * (1\ /\ (1 + r))$
using *linordered-ring-strict-class.mult-le-cancel-left*[*of* $1\ /\ (1 + l)$ $((1 + l) * r) * (1\ /\ (1 + r))$ $(l * (1 + r))\ * (1\ /\ (1 + r))$] *assms*(*2*)
by *auto*
hence $(1 + l) * r \leq l * (1 + r)$
using *linordered-ring-strict-class.mult-le-cancel-right*[*of* $(1 + l) * r\ 1\ /\ (1 + r)\ l * (1 + r)$] *assms*(*1*)
by *auto*
thus *?thesis*
using *assms* **by** (*simp add*: *distrib-left distrib-right*)
qed

lemma *bounded-dist-mono-strict-inverse*:
fixes $r\ l :: real$
assumes $0 \leq r\ 0 \leq l$ **and** $r\ /\ (1 + r) < l\ /\ (1 + l)$
shows $r < l$
proof −
have $(1\ /\ (1 + l)) * (((1 + l) * r) * (1\ /\ (1 + r))) < (1\ /\ (1 + l)) * ((l * (1 + r))\ * (1\ /\ (1 + r)))$
using *assms* **by** *auto*
hence $((1 + l) * r) * (1\ /\ (1 + r)) < (l * (1 + r))\ * (1\ /\ (1 + r))$
using *linordered-ring-strict-class.mult-less-cancel-left*[*of* $1\ /\ (1 + l)$ $((1 + l) * r) * (1\ /\ (1 + r))$ $(l * (1 + r))\ * (1\ /\ (1 + r))$] *assms*(*2*)
by *auto*
hence $(1 + l) * r < l * (1 + r)$
using *linordered-ring-strict-class.mult-less-cancel-right*[*of* $(1 + l) * r\ 1\ /\ (1 + r)\ l * (1 + r)$] *assms*(*1*)
by *auto*
thus *?thesis*
using *assms* **by** (*simp add*: *distrib-left distrib-right*)
qed

lemma *bounded-dist-inverse-comp*:
fixes $\varepsilon :: real$
assumes $0 < \varepsilon$ **and** $\varepsilon < 1$
shows $\varepsilon = (\varepsilon\ /\ (1 - \varepsilon))\ /\ (1 + (\varepsilon\ /\ (1 - \varepsilon)))$
(**is** - = $?\varepsilon'\ /\ (1 + ?\varepsilon')$)
proof −
have $1 + \varepsilon\ /\ (1 - \varepsilon) = (1 - \varepsilon)\ /\ (1 - \varepsilon) + \varepsilon\ /\ (1 - \varepsilon)$

```
    using assms by auto
  also have ... = 1 / (1 − ε)
    by(simp only: division-ring-class.add-divide-distrib[symmetric], simp)
  finally show ε = ?ε' / (1 + ?ε')
    using assms by simp
qed

lemma(in metric-set) bounded-dist-dist:
  shows metric-set S (bounded-dist dist)
    and bounded-dist dist a b < 1
proof −
  show metric-set S (bounded-dist dist)
  proof
    show ⋀x y. 0 ≤ bounded-dist dist x y
         ⋀x y. x ∉ S ⟹ bounded-dist dist x y = 0
         ⋀x y. bounded-dist dist x y = bounded-dist dist y x
      using dist-geq0 dist-notin dist-sym
      by(auto simp: bounded-dist-def)
  next
    fix x y
    assume hxy:x ∈ S y ∈ S
    show x = y ⟷ (bounded-dist dist x y = 0)
    proof
      assume bounded-dist dist x y = 0
      then have dist x y / (1 + dist x y) = 0
        by(simp add: bounded-dist-def)
      hence dist x y = 0
        using field-class.divide-eq-0-iff[of d x y] dist-geq0
        by (simp add: add-nonneg-eq-0-iff)
      thus x = y
        using dist-0[OF hxy] by simp
    qed (simp add: bounded-dist-def dist-0[OF hxy])
  next
    fix x y z
    assume hxyz:x ∈ S y ∈ S z ∈ S
    have bounded-dist dist x z ≤ (dist x y + dist y z) / (1 + dist x y + dist y z)
     using bounded-dist-mono[OF - - dist-tr[OF hxyz],simplified semigroup-add-class.add.assoc[symmetric]]
dist-geq0
      by(simp add: bounded-dist-def)
    also have ... = dist x y / (1 + dist x y + dist y z) + dist y z / (1 + dist x y
+ dist y z)
      using add-divide-distrib by auto
    also have ... ≤ bounded-dist dist x y + bounded-dist dist y z
      apply(rule add-mono-thms-linordered-semiring(1))
      unfolding bounded-dist-def
      using dist-geq0
     by(auto intro!: linordered-field-class.divide-left-mono linordered-semiring-strict-class.mult-pos-pos
add-pos-nonneg )
    finally show bounded-dist dist x z ≤ bounded-dist dist x y + bounded-dist dist
```

```
y z .
  qed
  show bounded-dist dist a b < 1
    using dist-geq0[of a b] by(auto simp: bounded-dist-def)
qed

lemma(in metric-set) bounded-dist-ball-eq:
  assumes x ∈ S and ε > 0
  shows open-ball x ε = metric-set.open-ball S (bounded-dist dist) x (ε / (1 + ε))
proof(rule set-eqI)
  interpret m2: metric-set S bounded-dist dist
    by(rule bounded-dist-dist)
  fix y
  have y ∈ open-ball x ε ⟷ y ∈ S ∧ dist x y < ε
    using assms by(simp add: open-ball-def)
  also have ... ⟷ y ∈ S ∧ dist x y / (1 + dist x y) < ε / (1 + ε)
   using bounded-dist-mono-strict[of dist x y ε] bounded-dist-mono-strict-inverse[of
dist x y ε] dist-geq0 assms(2)
    by auto
  also have ... ⟷ y ∈ m2.open-ball x (ε / (1 + ε))
    using assms by(simp add: m2.open-ball-def,simp add: bounded-dist-def)
  finally show y ∈ open-ball x ε ⟷ y ∈ m2.open-ball x (ε / (1 + ε)) .
qed

lemma(in metric-set) bounded-dist-ball-ge1:
  assumes x ∈ S and 1 ≤ ε
  shows metric-set.open-ball S (bounded-dist dist) x ε = S
proof −
  interpret m2: metric-set S bounded-dist dist
    by(rule bounded-dist-dist)
  show ?thesis
    using order.strict-trans2[OF bounded-dist-dist(2)[of x] assms(2)] assms(1)
    by(auto simp: m2.open-ball-def)
qed

lemma(in metric-set) bounded-dist-generate-same-topology:
  mtopology = metric-set.mtopology S (bounded-dist dist)
proof −
  interpret m2: metric-set S bounded-dist dist
    by(rule bounded-dist-dist)
  show ?thesis
 proof(rule metric-generates-same-topology[OF metric-set-axioms bounded-dist-dist(1)])
    fix x U
    assume h: U ⊆ S ∀ x∈U. ∃ε>0. open-ball x ε ⊆ U x ∈ U
    then obtain ε where he:
      ε > 0 open-ball x ε ⊆ U by auto
    show ∃ε>0. m2.open-ball x ε ⊆ U
      using he bounded-dist-ball-eq[of x ε] h
      by(auto intro!: exI[where x=ε / (1 + ε)])
```

```
  next
    fix x U
    assume h: U ⊆ S ∀x∈U. ∃ε>0. m2.open-ball x ε ⊆ U x ∈ U
    then obtain ε where he:
      ε > 0 m2.open-ball x ε ⊆ U by auto
    consider ε < 1 | 1 ≤ ε by fastforce
    then show ∃ε>0. open-ball x ε ⊆ U
    proof cases
      case 1
      let ?ε' = ε / (1 − ε)
      note 2 = bounded-dist-inverse-comp[OF he(1) 1]
      have 3:0 < ?ε'
        using he 1 by auto
      show ?thesis
        using h(1,3) he(2) 3 bounded-dist-ball-eq[of x ?ε',simplified 2[symmetric]]
        by(auto intro!: exI[where x=?ε'])
    next
      case 2
      have U = S
        using bounded-dist-ball-ge1[of x,OF - 2] h(1,3) he(2)
        by auto
      thus ?thesis
        using open-ball-subset-ofS
        by(auto intro!: exI[where x=1])
    qed
  qed
qed

lemma(in metric-set) bounded-dist-converge-to-inS-iff:
 converge-to-inS xn x ⟷ metric-set.converge-to-inS S (bounded-dist dist) xn x
 by(simp add: metric-generates-same-topology-converges[OF metric-set-axioms bounded-dist-dist(1)
bounded-dist-generate-same-topology])

lemma(in metric-set) bounded-dist-Cauchy-eq:
 Cauchy-inS f ⟷ metric-set.Cauchy-inS S (bounded-dist dist) f
proof −
  interpret m2: metric-set S bounded-dist dist
    by(rule bounded-dist-dist)
  show ?thesis
  proof
    assume h:Cauchy-inS f
    show m2.Cauchy-inS f
      unfolding m2.Cauchy-inS-def2'
    proof safe
      fix ε :: real
      assume he: 0 < ε
      consider ε < 1 | 1 ≤ ε by fastforce
      then show ∃x∈S. ∃N. ∀n≥N. f n ∈ m2.open-ball x ε
      proof cases
```

```
      case 1
      let ?ε = ε / (1 − ε)
      note 2 = bounded-dist-inverse-comp[OF he(1) 1]
      have 3:0 < ?ε
        using he 1 by auto
      then obtain x N where hxn:
       x ∈ S ⋀n. n≥N ⟹ f n ∈ open-ball x ?ε
        using Cauchy-inS-def2'[of f] h by blast
      show ?thesis
        using hxn bounded-dist-ball-eq[OF hxn(1) 3,simplified 2[symmetric]]
        by(auto intro!: bexI[where x=x] exI[where x=N])
    next
      case 2
      then show ?thesis
        using bounded-dist-ball-ge1[of f 0 ε] Cauchy-inS-def2'[of f] h
        by(auto intro!: bexI[where x=f 0] exI[where x=0])
    qed
   qed(rule Cauchy-inS-dest1[OF h])
  next
   assume h:m2.Cauchy-inS f
   show Cauchy-inS f
    unfolding Cauchy-inS-def2'
   proof safe
    fix ε :: real
    assume he:0 < ε
    then have 0 < ε / (1 + ε) by simp
    then obtain x N where
     x ∈ S ⋀n. n ≥N ⟹ f n ∈ m2.open-ball x (ε / (1 + ε))
      using h[simplified m2.Cauchy-inS-def2'] by blast
    thus ∃x∈S. ∃N. ∀n≥N. f n ∈ open-ball x ε
      using he bounded-dist-ball-eq[of x ε]
      by(auto intro!: bexI[where x=x] exI[where x=N])
   qed(rule m2.Cauchy-inS-dest1[OF h])
  qed
qed

lemma(in complete-metric-set) bounded-dist-complete:
 complete-metric-set S (bounded-dist dist)
  unfolding complete-metric-set-def complete-metric-set-axioms-def
  by(auto intro!: bounded-dist-dist convergence simp: bounded-dist-Cauchy-eq[symmetric]
metric-generates-same-topology-convergent[OF metric-set-axioms bounded-dist-dist(1)
bounded-dist-generate-same-topology,symmetric])

lemma(in polish-metric-set) bounded-dist-polish:
 polish-metric-set S (bounded-dist dist)
  unfolding polish-metric-set-def
  using metric-generates-same-topology-separable[OF metric-set-axioms bounded-dist-dist(1)
bounded-dist-generate-same-topology]
  by(auto intro!: bounded-dist-complete separable-metric-set-axioms)
```

lemma(**in** *metric-set*) *uniform-continuous-map-bounded-dist-equiv*:
assumes *metric-set T f*
shows *uniform-continuous-map S dist T f* = *uniform-continuous-map S* (*bounded-dist dist*) *T f*
proof
fix *g*
interpret *bS*: *metric-set S bounded-dist dist*
by (*rule bounded-dist-dist*(*1*))
interpret *T*: *metric-set T f* **by** *fact*
show *uniform-continuous-map S dist T f g* = *uniform-continuous-map S* (*bounded-dist dist*) *T f g*
unfolding *uniform-continuous-map-def*[*OF metric-set-axioms T.metric-set-axioms*] *uniform-continuous-map-def*[*OF bS.metric-set-axioms T.metric-set-axioms*]
proof *safe*
fix *e* :: *real*
assume *h*: *e* > *0 g* ∈ *S* → *T* ∀ε>*0*. ∃ δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < δ ⟶ *f* (*g x*) (*g y*) < ε
with *h*(*3*) **obtain** *d* **where** *d*: *d* > *0* ⋀*x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *dist x y* < *d* ⟹ *f* (*g x*) (*g y*) < *e*
by *metis*
consider *d* ≥ *1* | *d* < *1* **by** *fastforce*
show ∃ δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *bounded-dist dist x y* < δ ⟶ *f* (*g x*) (*g y*) < *e*
proof(*safe intro*!: *exI*[**where** *x*=*d* / (*1* + *d*)])
fix *x y*
assume *xy*:*x* ∈ *S y* ∈ *S bounded-dist dist x y* < *d* / (*1* + *d*)
then have *dist x y* < *d*
using *d*(*1*) *dist-geq0 bounded-dist-mono-strict-inverse*[*of dist x y d*] **by**(*auto simp*: *bounded-dist-def*)
thus *f* (*g x*) (*g y*) < *e*
by(*auto intro*!: *d xy*)
qed(*use d* **in** *auto*)
next
fix *e* :: *real*
assume *h*: *e* > *0 g* ∈ *S* → *T* ∀ε>*0*. ∃ δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *bounded-dist dist x y* < δ ⟶ *f* (*g x*) (*g y*) < ε
with *h*(*3*) **obtain** *d* **where** *d*: *d* > *0* ⋀*x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *bounded-dist dist x y* < *d* ⟹ *f* (*g x*) (*g y*) < *e*
by *metis*
show ∃ δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < δ ⟶ *f* (*g x*) (*g y*) < *e*
proof(*safe intro*!: *exI*[**where** *x*=*d*])
fix *x y*
assume *xy*: *x* ∈ *S y* ∈ *S dist x y* < *d*
then have *bounded-dist dist x y* < *d*
using *dist-geq0*[*of x y*] **by**(*auto intro*!: *order.strict-trans1*[*OF divide-left-mono*[*OF le-add-same-cancel1*[*THEN iffD2*,*OF dist-geq0*,*of 1*] *dist-geq0*],*simplified*] *simp*: *bounded-dist-def*)
from *d*(*2*)[*OF xy*(*1*,*2*) *this*] **show** *f* (*g x*) (*g y*) < *e* **.**
qed(*use d* **in** *auto*)
qed

qed

lemma(**in** *metric-set*) *uniform-continuous-map-bounded-dist-equiv'*:
assumes *metric-set T f*
shows *uniform-continuous-map S dist T f* = *uniform-continuous-map S* (*bounded-dist dist*) *T* (*bounded-dist f*)
proof
fix *g*
interpret *bS*: *metric-set S bounded-dist dist*
by (*rule bounded-dist-dist*(*1*))
interpret *T*: *metric-set T f* **by** *fact*
interpret *bT*: *metric-set T bounded-dist f*
by(*rule T.bounded-dist-dist*(*1*))
show *uniform-continuous-map S dist T f g* = *uniform-continuous-map S* (*bounded-dist dist*) *T* (*bounded-dist f*) *g*
unfolding *uniform-continuous-map-def*[*OF metric-set-axioms T.metric-set-axioms*] *uniform-continuous-map-def*[*OF bS.metric-set-axioms bT.metric-set-axioms*]
proof *safe*
fix *e* :: *real*
assume *h*: *e* > *0 g* ∈ *S* → *T* ∀ε>*0*. ∃δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < δ ⟶ *f* (*g x*) (*g y*) < ε
with *h*(*3*) **obtain** *d* **where** *d*: *d* > *0* ⋀*x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *dist x y* < *d* ⟹ *f* (*g x*) (*g y*) < *e*
by *metis*
consider *d* ≥ *1* | *d* < *1* **by** *fastforce*
show ∃δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *bounded-dist dist x y* < δ ⟶ *bounded-dist f* (*g x*) (*g y*) < *e*
proof(*safe intro*!: *exI*[**where** *x*=*d* / (*1* + *d*)])
fix *x y*
assume *xy*:*x* ∈ *S y* ∈ *S* *bounded-dist dist x y* < *d* / (*1* + *d*)
then have *dist x y* < *d*
using *d*(*1*) *dist-geq0 bounded-dist-mono-strict-inverse*[*of dist x y d*] **by**(*auto simp*: *bounded-dist-def*)
then have *f* (*g x*) (*g y*) < *e*
by(*auto intro*!: *d xy*)
thus *bounded-dist f* (*g x*) (*g y*) < *e*
using *T.dist-geq0*[*of g x g y*] **by**(*auto intro*!: *order.strict-trans1*[*OF divide-left-mono*[*OF le-add-same-cancel1*[*THEN iffD2*,*OF T.dist-geq0*,*of 1*] *T.dist-geq0*],*simplified*] *simp*: *bounded-dist-def*)
qed(*use d* **in** *auto*)
next
fix *e* :: *real*
assume *h*: *e* > *0 g* ∈ *S* → *T* ∀ε>*0*. ∃δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *bounded-dist dist x y* < δ ⟶ *bounded-dist f* (*g x*) (*g y*) < ε
then have *e* / (*1* + *e*) > *0* **by** *auto*
with *h*(*3*) **obtain** *d* **where** *d*: *d* > *0* ⋀*x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *bounded-dist dist x y* < *d* ⟹ *bounded-dist f* (*g x*) (*g y*) < *e* / (*1* + *e*)
by *metis*
show ∃δ>*0*. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < δ ⟶ *f* (*g x*) (*g y*) < *e*

```
    proof(safe intro!: exI[where x=d])
      fix x y
      assume xy: x ∈ S y ∈ S dist x y < d
      then have bounded-dist dist x y < d
      using dist-geq0[of x y] by(auto intro!: order.strict-trans1[OF divide-left-mono[OF
le-add-same-cancel1[THEN iffD2,OF dist-geq0,of 1] dist-geq0],simplified] simp: bounded-dist-def)
      from d(2)[OF xy(1,2) this] show f (g x) (g y) < e
        using h(1) T.dist-geq0 by(auto intro!: bounded-dist-mono-strict-inverse[of
f (g x) (g y) e] simp: bounded-dist-def)
    qed(use d in auto)
  qed
qed

lemma(in metric-set) Urysohn-uniform:
  assumes closedin mtopology T closedin mtopology U T ∩ U = {} ⋀x y. x ∈ T
⟹ y ∈ U ⟹ dist x y ≥ e e > 0
  obtains f :: 'a ⇒ real
  where uniform-continuous-map S dist UNIV dist-typeclass 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 uniform-continuous-map S dist UNIV dist-typeclass f ⋀x. f x
∈{0..1} ⋀x. x ∈ T ⟹ f x = 1 ⋀x. x ∈ U ⟹ f x = 0
       by(auto intro!: uniform-continuous-map-const[OF metric-set-axioms met-
ric-class-metric-set] simp: f-def)
    then show ?thesis
      using that by auto
  next
    case 2
    define f where f ≡ (λx::'a. 1::real)
     with 2 have uniform-continuous-map S dist UNIV dist-typeclass f ⋀x. f x
∈{0..1} ⋀x. x ∈ T ⟹ f x = 1 ⋀x. x ∈ U ⟹ f x = 0
       by(auto intro!: uniform-continuous-map-const[OF metric-set-axioms met-
ric-class-metric-set] simp: f-def)
    then show ?thesis
      using that by auto
  next
    case TU:3
    then have STU:S ≠ {} T ⊆ S U ⊆ S
      using assms(1,2) closedin-topspace-empty mtopology-topspace closedin-subset
by fastforce+
    interpret bd: metric-set S bounded-dist dist
      by (rule bounded-dist-dist(1))
    have e:⋀x y. x ∈ T ⟹ y ∈ U ⟹ bounded-dist dist x y ≥ e / (1 + e)
     using assms by(auto intro!: bounded-dist-mono simp: bounded-dist-def dist-geq0)
```

define f **where** $f \equiv (\lambda x.\ bd.dist\text{-}set\ U\ x\ /\ (bd.dist\text{-}set\ U\ x\ +\ bd.dist\text{-}set\ T\ x))$
have *uniform-continuous-map S dist UNIV dist-typeclass f*
unfolding *f-def*
proof(*rule uniform-continuous-map-real-devide*[**where** *Kf=1* **and** *Kg=2*])
show *uniform-continuous-map S dist UNIV dist-typeclass* (*bd.dist-set U*)
uniform-continuous-map S dist UNIV dist-typeclass ($\lambda x.$ *bd.dist-set U x + bd.dist-set T x*)
by(*auto simp*: *uniform-continuous-map-bounded-dist-equiv*[*OF metric-class-metric-set*] *bd.dist-set-uniform-continuous intro*!: *bd.uniform-continuous-map-add*)
next
fix x
assume x:$x \in S$
consider $x \in (\bigcup a{\in}U.\ bd.open\text{-}ball\ a\ ((e\ /\ (1\ +\ e))\ /\ 2))\ |\ x \notin (\bigcup a{\in}U.\ bd.open\text{-}ball\ a\ ((e\ /\ (1\ +\ e))\ /\ 2))$ **by** *auto*
then show $(e\ /\ (1\ +\ e))\ /\ 2 \leq |bd.dist\text{-}set\ U\ x\ +\ bd.dist\text{-}set\ T\ x|$
proof *cases*
case *1*
have $bd.open\text{-}ball\ x\ ((e\ /\ (1\ +\ e))\ /\ 2) \cap T = \{\}$
proof(*rule ccontr*)
assume $bd.open\text{-}ball\ x\ ((e\ /\ (1\ +\ e))\ /\ 2) \cap T \neq \{\}$
then obtain y **where** y:$y \in bd.open\text{-}ball\ x\ ((e\ /\ (1\ +\ e))\ /\ 2)\ y \in T$
by *auto*
obtain u **where** u:$u \in U\ x \in bd.open\text{-}ball\ u\ ((e\ /\ (1\ +\ e))\ /\ 2)$
using *1* **by** *auto*
have *bounded-dist dist y u* $\leq$ *bounded-dist dist y x + bounded-dist dist x u*
using *STU u y x* **by**(*auto intro*!: *bd.dist-tr*)
also have ... $<$ $(e\ /\ (1\ +\ e))\ /\ 2\ +\ (e\ /\ (1\ +\ e))\ /\ 2$
using *bd.open-ballD*[*OF u(2)*] *bd.open-ballD*[*OF y(1)*] **by**(*simp add*: *bd.dist-sym*)
also have ... $=$ $e\ /\ (1\ +\ e)$ **using** *assms(5)* **by** *linarith*
finally show *False*
using *e*[*OF y(2) u(1)*] **by** *simp*
qed
from *bd.dist-set-ball-empty*[*OF TU(1) STU(2) - x this*] *assms*
have $e\ /\ (1\ +\ e)\ /\ 2 \leq bd.dist\text{-}set\ T\ x$ **by** *auto*
also have ... $\leq |bd.dist\text{-}set\ U\ x\ +\ bd.dist\text{-}set\ T\ x|$
using *bd.dist-set-geq0* **by** *auto*
finally show *?thesis* **.**
next
case *2*
then have $bd.open\text{-}ball\ x\ ((e\ /\ (1\ +\ e))\ /\ 2) \cap U = \{\}$
by(*auto simp*: *bd.open-ball-inverse*[*of x*])
from *bd.dist-set-ball-empty*[*OF TU(2) STU(3) - x this*] *assms*
have $e\ /\ (1\ +\ e)\ /\ 2 \leq bd.dist\text{-}set\ U\ x$ **by** *auto*
also have ... $\leq |bd.dist\text{-}set\ U\ x\ +\ bd.dist\text{-}set\ T\ x|$
using *bd.dist-set-geq0* **by** *auto*
finally show *?thesis* **.**
qed
thus $bd.dist\text{-}set\ U\ x\ +\ bd.dist\text{-}set\ T\ x \neq 0$

```
      using bd.dist-set-geq0 assms(5) order-antisym-conv by fastforce
    next
      show 0 < e / (1 + e) / 2
        using assms by auto
    next
      fix x
      have |bd.dist-set U x + bd.dist-set T x| = bd.dist-set U x + bd.dist-set T x
        using bd.dist-set-geq0 by auto
      also have ... < 2
      by (metis add-mono-thms-linordered-field(5) one-add-one bd.dist-set-bounded[OF
bounded-dist-dist(2),simplified])
      finally show |bd.dist-set U x + bd.dist-set T x| < 2 .
      show |bd.dist-set U x| < 1
        using bd.dist-set-geq0 bd.dist-set-bounded[OF bounded-dist-dist(2)] by auto
    qed
    moreover have ⋀x. f x ∈{0..1}
      unfolding f-def
    proof −
      fix x
      have bd.dist-set U x / (bd.dist-set U x + bd.dist-set T x) ≤ bd.dist-set U x /
bd.dist-set U x
      proof −
        consider bd.dist-set U x = 0 | bd.dist-set U x > 0
          using bd.dist-set-geq0 by (auto simp: less-eq-real-def)
        thus ?thesis
        proof cases
          case 2
          show ?thesis
         by(rule divide-left-mono[OF - - mult-pos-pos]) (insert 2 bd.dist-set-geq0,simp-all
add: add.commute add-nonneg-pos)
        qed simp
      qed
      also have ... ≤ 1 by simp
      finally show bd.dist-set U x / (bd.dist-set U x + bd.dist-set T x) ∈ {0..1}
        using bd.dist-set-geq0 by auto
    qed
    moreover have f x = 1 if x:x ∈ T for x
    proof −
      { assume h:bd.dist-set U x = 0
        then have x ∉ U using assms STU x by blast
        hence False
       using bd.dist-set-closed-ge0[simplified bounded-dist-generate-same-topology[symmetric],OF
assms(2) TU(2),of x] STU h x
          by auto
      }
      thus ?thesis
        by(auto simp: f-def bd.dist-set-inA x)
    qed
    moreover have ⋀x. x ∈ U ⟹ f x = 0
```

by (*auto simp*: *f-def bd.dist-set-inA*)
ultimately show *?thesis*
using *that* **by** *auto*
qed
qed

lemma *product-metricI'*:
assumes *0 < r r < 1 countable I* $\bigwedge i.\ i \in I \Longrightarrow$ *metric-set* (*S i*) (*d i*)
shows *product-metric r I* (*to-nat-on I*) (*from-nat-into I*) *S* ($\lambda i\ x\ y.$ *if i ∈ I then bounded-dist* (*d i*) *x y else 0*) *1*
proof −
have $\bigwedge i.\ i \in I \Longrightarrow$ *metric-set* (*S i*) (*bounded-dist* (*d i*))
$\bigwedge i\ x\ y.\ i \in I \Longrightarrow$ *bounded-dist* (*d i*) *x y* ≤ *1*
using *assms*(*4*) **by**(*auto intro*!: *metric-set.bounded-dist-dist*(*1*) *less-imp-le*[*OF metric-set.bounded-dist-dist*(*2*)])
thus *?thesis*
by(*auto intro*!: *product-metricI*[*OF assms*(*1−3*)] *simp*: *metric-set-def*)
qed

lemma *product-complete-metricI'*:
assumes *0 < r r < 1 countable I* $\bigwedge i.\ i \in I \Longrightarrow$ *complete-metric-set* (*S i*) (*d i*)
shows *product-complete-metric r I* (*to-nat-on I*) (*from-nat-into I*) *S* ($\lambda i\ x\ y.$ *if i ∈ I then bounded-dist* (*d i*) *x y else 0*) *1*
proof −
have $\bigwedge i.\ i \in I \Longrightarrow$ *complete-metric-set* (*S i*) (*bounded-dist* (*d i*))
$\bigwedge i\ x\ y.\ i \in I \Longrightarrow$ *bounded-dist* (*d i*) *x y* ≤ *1*
using *assms*(*4*) **by**(*auto intro*!: *metric-set.bounded-dist-dist*(*1*) *less-imp-le*[*OF metric-set.bounded-dist-dist*(*2*)] *simp*: *complete-metric-set-def*) (*simp add*: *assms*(*4*) *complete-metric-set.axioms*(*2*) *complete-metric-set.bounded-dist-complete*)
thus *?thesis*
by(*auto intro*!: *product-complete-metricI*[*OF assms*(*1−3*)] *simp*: *complete-metric-set-def*) (*metis metric-set.dist-geq0*)
qed

lemma *product-complete-metric-natI'*:
assumes *0 < r r < 1* $\bigwedge n.$ *complete-metric-set* (*S n*) (*d n*)
shows *product-complete-metric r UNIV id id S* ($\lambda n.$ *bounded-dist* (*d n*)) *1*
proof −
have $\bigwedge n.$ *complete-metric-set* (*S n*) (*bounded-dist* (*d n*))
$\bigwedge n\ x\ y.$ *bounded-dist* (*d n*) *x y* ≤ *1*
using *assms*(*3*) **by**(*auto intro*!: *metric-set.bounded-dist-dist*(*1*) *less-imp-le*[*OF metric-set.bounded-dist-dist*(*2*)] *simp*: *complete-metric-set-def*) (*simp add*: *assms*(*3*) *complete-metric-set.axioms*(*2*) *complete-metric-set.bounded-dist-complete*)
thus *?thesis*
by(*auto intro*!: *product-complete-metric-natI*[*OF assms*(*1,2*)]) (*meson complete-metric-set-def metric-set.dist-geq0*)
qed

lemma *product-polish-metricI'*:

assumes *0 < r r < 1 countable I* $\bigwedge i.\ i \in I \Longrightarrow$ *polish-metric-set* (*S i*) (*d i*)
shows *product-polish-metric r I* (*to-nat-on I*) (*from-nat-into I*) *S* ($\lambda i\ x\ y.$ *if* $i \in I$ *then bounded-dist* (*d i*) *x y else 0*) *1*
proof −
have $\bigwedge i.\ i \in I \Longrightarrow$ *metric-set* (*S i*) (*bounded-dist* (*d i*))
$\bigwedge i\ x\ y.\ i \in I \Longrightarrow$ *bounded-dist* (*d i*) *x y* $\leq$ *1*
using *assms*(*4*) **by**(*auto intro*!: *metric-set.bounded-dist-dist*(*1*) *less-imp-le*[*OF metric-set.bounded-dist-dist*(*2*)] *simp*: *polish-metric-set-def complete-metric-set-def*)
thus *?thesis*
using *assms*(*4*) **by**(*auto intro*!: *product-polish-metricI*[*OF assms*(*1*−*3*)] *polish-metric-set.bounded-dist-polish simp*: *metric-set-def*)
qed

end

3 Abstract Metrizable Topology

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

3.1 Metrizable Spaces

locale *metrizable* =
fixes *S* :: *'a topology*
assumes *ex-metric*:$\exists\,\varrho.$ *metric-set* (*topspace S*) $\varrho \wedge$ *S* = *metric-set.mtopology* (*topspace S*) ϱ
begin

lemma *metric*:
obtains ϱ **where** *metric-set* (*topspace S*) ϱ *metric-set.mtopology* (*topspace S*) ϱ = *S*
using *ex-metric* **by** *metis*

lemma *bounded-metric*:
obtains ϱ **where** *metric-set* (*topspace S*) ϱ *metric-set.mtopology* (*topspace S*) ϱ = *S*
$\bigwedge x\ y.\ \varrho\ x\ y < 1$
proof −
obtain ϱ **where** *metric-set* (*topspace S*) ϱ *metric-set.mtopology* (*topspace S*) ϱ = *S*
by(*rule metric*)
then have $\exists\,\varrho.$ *metric-set* (*topspace S*) $\varrho \wedge$ *metric-set.mtopology* (*topspace S*) ϱ = *S* $\wedge$ ($\forall\, x\ y.\ \varrho\ x\ y < 1$)
using *metric-set.bounded-dist-dist*(*1*) *metric-set.bounded-dist-dist*(*2*) *metric-set.bounded-dist-generate-same*
by(*fastforce intro*!: *exI*[**where** *x*=*bounded-dist* ϱ])
thus *?thesis*
using *that* **by** *auto*
qed

lemma *second-countable-if-separable*:
assumes *separable S*
shows *second-countable S*
proof −
obtain *d* **where** *hd*:*metric-set* (*topspace S*) *d S* = *metric-set.mtopology* (*topspace S*) *d*
using *ex-metric* **by**(*auto simp*: *metrizable-def*)
then interpret *m*: *separable-metric-set topspace S d*
using *metric-set.separable-iff-topological-separable*[*of topspace S d*] *assms*
by *auto*
show *second-countable S*
using *m.second-countable* ‹*S* = *m.mtopology*› **by** *simp*
qed

corollary *second-countable-iff-separable*: *second-countable S* ⟷ *separable S*
using *second-countable-if-separable separable-if-second-countable*
by *auto*

lemma *Hausdorff*: *Hausdorff-space S*
using *ex-metric metric-set.mtopology-Hausdorff* **by** *fastforce*

lemma *subtopology*: *metrizable* (*subtopology S X*)
proof −
obtain *ϱ* **where** *h*:*metric-set* (*topspace S*) *ϱ metric-set.mtopology* (*topspace S*) *ϱ* = *S*
by(*rule metric*)
then show *?thesis*
using *metric-set.submetric-subtopology*[*OF h*(*1*),*of topspace S* ∩ *X*]
by(*auto intro*!: *exI*[**where** *x*=*submetric* (*topspace S* ∩ *X*) *ϱ*] *simp*: *metrizable-def subtopology-restrict metric-set.mtopology-topspace metric-set.submetric-metric-set*)
qed

lemma *g-delta-of-closedin*:
assumes *closedin S X*
shows *g-delta-of S X*
using *assms ex-metric metric-set.g-delta-of-closed* **by** *fastforce*

lemma *closedin-singleton*:
assumes *s* ∈ *topspace S*
shows *closedin S* {*s*}
proof −
obtain *ϱ* **where** *h*:*metric-set* (*topspace S*) *ϱ metric-set.mtopology* (*topspace S*) *ϱ* = *S*
by(*rule metric*)
then show *?thesis*
using *metric-set.closedin-closed-ball*[*OF h*(*1*),*of s 0*]
by(*simp add*: *metric-set.closed-ball-0*[*OF h*(*1*) *assms*])
qed

lemma *dense-of-infinite*:
assumes *infinite* (*topspace S*) *dense-of S U*
shows *infinite U*
proof −
obtain ϱ **where** *h*:*metric-set* (*topspace S*) ϱ *metric-set.mtopology* (*topspace S*) ϱ = *S*
by(*rule metric*)
show *?thesis*
by(*rule metric-set.dense-set-infinite*[*OF h*(*1*),*simplified h*(*2*),*OF assms*])
qed

lemma *homeomorphic-metrizable*:
assumes *S homeomorphic-space S′*
shows *metrizable S′*
proof(*rule metric*)
fix *d*
assume *h*: *metric-set* (*topspace S*) *d metric-set.mtopology* (*topspace S*) *d* = *S*
then interpret *m*: *metric-set topspace S d* **by** *simp*
from *assms* **obtain** *f g* **where** *fg*: *homeomorphic-maps S S′ f g*
by(*auto simp*: *homeomorphic-space-def*)
hence *g*: *g* ∈ *topspace S′* → *topspace S inj-on g* (*topspace S′*) *g* ‘ (*topspace S′*) = *topspace S*
by (*auto simp*: *homeomorphic-eq-injective-perfect-map homeomorphic-maps-map perfect-map-def*)
have *f*: *f* ∈ *topspace S* → *topspace S′ inj-on f* (*topspace S*) *f* ‘ (*topspace S*) = *topspace S′*
using *fg* **by** (*auto simp*: *homeomorphic-eq-injective-perfect-map homeomorphic-maps-map perfect-map-def*)
interpret *m′*: *metric-set topspace S′ m.ed g* (*topspace S′*)
by(*simp add*: *m.embed-dist-dist*[*OF g*(*1*,*2*)])
show *metrizable S′*
unfolding *metrizable-def*
proof(*safe intro*!: *exI*[**where** *x*=*m.ed g* (*topspace S′*)])
have [*simp*]:*m′.ed f* (*topspace S*) = *d*
by *standard*+ (*insert f g fg m.dist-notin m.dist-notin′*,*auto simp*: *m′.embed-dist-on-def m.embed-dist-on-def homeomorphic-maps-def*)
have [*simp*]:((‘) *f* ‘ {*m.open-ball a* ε |*a* ε. *a* ∈ *topspace S* ∧ 0 < ε}) = {*m′.open-ball a* ε |*a* ε. *a* ∈ *topspace S′* ∧ 0 < ε}
proof *safe*
fix *a* **and** *e* :: *real*
assume *a* ∈ *topspace S* 0 < *e*
then show ∃ *b e′*. *f* ‘ *m.open-ball a e* = *m′.open-ball b e′* ∧ *b* ∈ *topspace S′* ∧ 0 < *e′*
using *f g fg* **by**(*auto simp*: *m.open-ball-def m′.open-ball-def m.embed-dist-on-def homeomorphic-maps-def intro*!: *exI*[**where** *x*=*f a*] *exI*[**where** *x*=*e*]) (*metis* (*no-types*, *lifting*) *image-eqI mem-Collect-eq*)
next
fix *a* **and** *e* :: *real*

assume $a \in$ *topspace* S' $0 < e$
then show *m'.open-ball a e* $\in$ (‘) *f* ‘ {*m.open-ball a* ε |*a* ε. $a \in$ *topspace S* $\wedge$ $0 < \varepsilon$}
using *m'.embed-dist-open-ball*[*OF f*(*1*,*2*),*simplified*,*of g a e*] *f g fg m'.open-ballD'*(*1*)
by(*auto simp*: *m.embed-dist-on-def homeomorphic-maps-def image-def intro*!: *exI*[**where** *x*=*g a*] *exI*[**where** *x*=*e*] *exI*[**where** *x*=*m.open-ball* (*g a*) *e*]) *blast*
qed
show $S' = m'.mtopology$
using *topology-generated-by-homeomorphic-spaces*[*OF homeomorphic-maps-imp-map*[*OF fg*] *h*(*2*)[*symmetric*,*simplified m.mtopology-def2*]]
by(*simp add*: *m'.mtopology-def2*)
qed(*rule m'.metric-set-axioms*)
qed

end

lemma *euclidean-metrizable*: *metrizable* (*euclidean* :: ('*a* ::*metric-space*) *topology*)
by (*metis euclidean-mtopology metric-class-metric-set metrizable.intro topspace-euclidean*)

sublocale *metric-set* $\subseteq$ *metrizable mtopology*
using *metric-set-axioms metrizable-def mtopology-topspace* **by** *fastforce*

lemma *metrizable-prod*:
assumes *metrizable X metrizable Y*
shows *metrizable* (*prod-topology X Y*)
proof
obtain *dx dy* **where** *metric-set* (*topspace X*) *dx metric-set.mtopology* (*topspace X*) *dx* = *X metric-set* (*topspace Y*) *dy metric-set.mtopology* (*topspace Y*) *dy* = *Y*
using *metrizable.metric*[*OF assms*(*2*)] *metrizable.metric*[*OF assms*(*1*)] **by** *metis*
then show $\exists \varrho$. *metric-set* (*topspace* (*prod-topology X Y*)) ϱ $\wedge$ *prod-topology X Y* = *metric-set.mtopology* (*topspace* (*prod-topology X Y*)) ϱ
by(*auto intro*!: *exI*[**where** *x*=*binary-distance* (*topspace X*) *dx* (*topspace Y*) *dy*] *simp*: *binary-metric-set binary-distance-mtopology*)
qed

lemma *metrizable-product*:
assumes *countable I* $\bigwedge i$. $i \in I \Longrightarrow$ *metrizable* (*X i*)
shows *metrizable* (*product-topology X I*)
proof −
obtain *d* **where** *hd*:$\bigwedge i$. $i \in I \Longrightarrow$ *metric-set* (*topspace* (*X i*)) (*d i*) $\bigwedge i$. $i \in I \Longrightarrow$ *X i* = *metric-set.mtopology* (*topspace* (*X i*)) (*d i*)
using *assms*(*2*) **by**(*auto simp*: *metrizable-def*) *metis*
from *product-metricI'*[*of 1*/*2* - - *d*,*OF* - - *assms*(*1*) *this*(*1*)]
interpret *pd*: *product-metric 1* / *2 I to-nat-on I from-nat-into I* λi. *topspace* (*X i*) λi *x y*. *if i* $\in$ *I then bounded-dist* (*d i*) *x y else 0 1*
by *simp*
show *?thesis*
using *hd*(*2*) **by**(*auto simp*: *metrizable-def pd.product-dist-distance pd.product-dist-mtopology*[*symmetric*] *hd*(*1*) *metric-set.bounded-dist-generate-same-topology intro*!: *exI*[**where** *x*=*pd.product-dist*]

product-topology-cong)
qed

3.2 Complete Metrizable Spaces

locale *complete-metrizable* =
fixes $S :: 'a$ *topology*
assumes *ex-cmetric*: $\exists \varrho$. *complete-metric-set* (*topspace S*) $\varrho \wedge S =$ *metric-set.mtopology* (*topspace S*) ϱ
begin

lemma *cmetric*:
obtains ϱ **where** *complete-metric-set* (*topspace S*) ϱ *metric-set.mtopology* (*topspace S*) $\varrho = S$
using *ex-cmetric* **by** *metis*

lemma *bounded-cmetric*:
obtains ϱ **where** *complete-metric-set* (*topspace S*) ϱ *metric-set.mtopology* (*topspace S*) $\varrho = S$
$\bigwedge x\ y.\ \varrho\ x\ y < 1$
proof −
obtain ϱ **where** *complete-metric-set* (*topspace S*) ϱ *metric-set.mtopology* (*topspace S*) $\varrho = S$
by(*rule cmetric*)
then have $\exists \varrho$. *complete-metric-set* (*topspace S*) $\varrho \wedge$ *metric-set.mtopology* (*topspace S*) $\varrho = S \wedge (\forall x\ y.\ \varrho\ x\ y < 1)$
using *metric-set.bounded-dist-dist*(*1*) *metric-set.bounded-dist-dist*(*2*) *metric-set.bounded-dist-generate-same complete-metric-set.bounded-dist-complete complete-metric-set-def*
by(*fastforce intro*!: *exI*[**where** *x*=*bounded-dist* ϱ])
thus *?thesis*
using *that* **by** *auto*
qed

lemma *metrizable*: *metrizable S*
using *complete-metric-set-def complete-metrizable-axioms complete-metrizable-def metrizable-def* **by** *blast*

sublocale *metrizable*
by(*rule metrizable*)

lemma *closedin-complete-metrizable*:
assumes *closedin S A*
shows *complete-metrizable* (*subtopology S A*)
by (*metis assms closedin-def complete-metric-set.submetric-complete-iff complete-metric-set-def complete-metrizable-axioms complete-metrizable-def metric-set.submetric-subtopology topspace-subtopology-subset*)

lemma *homeomorphic-complete-metrizable*:
assumes *S homeomorphic-space S′*

shows *complete-metrizable S′*
proof(*rule cmetric*)
fix *d*
assume *h*: *complete-metric-set* (*topspace S*) *d metric-set.mtopology* (*topspace S*) *d* = *S*
then interpret *m*: *complete-metric-set topspace S d* **by** *simp*
from *assms* **obtain** *f g* **where** *fg*: *homeomorphic-maps S S′ f g*
by(*auto simp*: *homeomorphic-space-def*)
hence *g*: *g* ∈ *topspace S′* → *topspace S inj-on g* (*topspace S′*) *g* ‘ (*topspace S′*) = *topspace S*
by (*auto simp*: *homeomorphic-eq-injective-perfect-map homeomorphic-maps-map perfect-map-def*)
have *f*: *f* ∈ *topspace S* → *topspace S′ inj-on f* (*topspace S*) *f* ‘ (*topspace S*) = *topspace S′*
using *fg* **by** (*auto simp*: *homeomorphic-eq-injective-perfect-map homeomorphic-maps-map perfect-map-def*)
interpret *m′*: *complete-metric-set topspace S′ m.ed g* (*topspace S′*)
by(*auto intro*!: *m.embed-dist-complete*[*OF g*(*1*,*2*)] *simp*: *h*(*2*) *g*(*3*))
show *complete-metrizable S′*
unfolding *complete-metrizable-def*
proof(*safe intro*!: *exI*[**where** *x*=*m.ed g* (*topspace S′*)])
have [*simp*]:*m′.ed f* (*topspace S*) = *d*
by *standard*+ (*insert f g fg m.dist-notin m.dist-notin′*,*auto simp*: *m′.embed-dist-on-def m.embed-dist-on-def homeomorphic-maps-def*)
have [*simp*]:((‘) *f* ‘ {*m.open-ball a* ε |*a* ε. *a* ∈ *topspace S* ∧ 0 < ε}) = {*m′.open-ball a* ε |*a* ε. *a* ∈ *topspace S′* ∧ 0 < ε}
proof *safe*
fix *a* **and** *e* :: *real*
assume *a* ∈ *topspace S* 0 < *e*
then show ∃ *b e′*. *f* ‘ *m.open-ball a e* = *m′.open-ball b e′* ∧ *b* ∈ *topspace S′* ∧ 0 < *e′*
using *f g fg* **by**(*auto simp*: *m.open-ball-def m′.open-ball-def m.embed-dist-on-def homeomorphic-maps-def intro*!: *exI*[**where** *x*=*f a*] *exI*[**where** *x*=*e*]) (*metis* (*no-types*, *lifting*) *image-eqI mem-Collect-eq*)
next
fix *a* **and** *e* :: *real*
assume *a* ∈ *topspace S′* 0 < *e*
then show *m′.open-ball a e* ∈ (‘) *f* ‘ {*m.open-ball a* ε |*a* ε. *a* ∈ *topspace S* ∧ 0 < ε}
using *m′.embed-dist-open-ball*[*OF f*(*1*,*2*),*simplified*,*of g a e*] *f g fg m′.open-ballD′*(*1*)
by(*auto simp*: *m.embed-dist-on-def homeomorphic-maps-def image-def intro*!: *exI*[**where** *x*=*g a*] *exI*[**where** *x*=*e*] *exI*[**where** *x*=*m.open-ball* (*g a*) *e*]) *blast*
qed
show *S′* = *m′.mtopology*
using *topology-generated-by-homeomorphic-spaces*[*OF homeomorphic-maps-imp-map*[*OF fg*] *h*(*2*)[*symmetric*,*simplified m.mtopology-def2*]]
by(*simp add*: *m′.mtopology-def2*)
qed(*rule m′.complete-metric-set-axioms*)
qed

end

lemma *euclidean-complete-metrizable*[*simp*]:
complete-metrizable (*euclidean* :: ('*a* ::*complete-space*) *topology*)
by (*metis complete-metrizable.intro complete-space-complete-metric-set euclidean-mtopology topspace-euclidean*)

sublocale *complete-metric-set* ⊆ *complete-metrizable mtopology*
using *complete-metric-set-axioms complete-metrizable-def mtopology-topspace* **by** *fastforce*

lemma *complete-metrizable-prod*:
assumes *complete-metrizable X complete-metrizable Y*
shows *complete-metrizable* (*prod-topology X Y*)
proof
obtain *dx dy* **where** *complete-metric-set* (*topspace X*) *dx metric-set.mtopology* (*topspace X*) *dx* = *X complete-metric-set* (*topspace Y*) *dy metric-set.mtopology* (*topspace Y*) *dy* = *Y*
using *complete-metrizable.cmetric*[*OF assms*(*2*)] *complete-metrizable.cmetric*[*OF assms*(*1*)] **by** *metis*
then show ∃ ϱ. *complete-metric-set* (*topspace* (*prod-topology X Y*)) ϱ ∧ *prod-topology X Y* = *metric-set.mtopology* (*topspace* (*prod-topology X Y*)) ϱ
using *binary-distance-complete* **by**(*auto intro*!: *exI*[**where** *x*=*binary-distance* (*topspace X*) *dx* (*topspace Y*) *dy*] *simp*: *binary-distance-mtopology complete-metric-set-def*)
qed

lemma *complete-metrizable-product*:
assumes *countable I* ⋀*i*. *i* ∈ *I* ⟹ *complete-metrizable* (*X i*)
shows *complete-metrizable* (*product-topology X I*)
proof −
obtain *d* **where** *hd*:⋀*i*. *i* ∈ *I* ⟹ *complete-metric-set* (*topspace* (*X i*)) (*d i*) ⋀*i*. *i* ∈ *I* ⟹ *X i* = *metric-set.mtopology* (*topspace* (*X i*)) (*d i*)
using *assms*(*2*) **by**(*auto simp*: *complete-metrizable-def*) *metis*
from *product-complete-metricI'*[*of 1/2 - - d*,*OF - - assms*(*1*) *this*(*1*)]
interpret *pd*: *product-complete-metric 1 / 2 I to-nat-on I from-nat-into I* λ*i*. *topspace* (*X i*) λ*i x y*. *if i* ∈ *I then bounded-dist* (*d i*) *x y else 0 1*
by *simp*
show *?thesis*
using *hd*(*2*) **by**(*auto simp*: *complete-metrizable-def pd.product-dist-distance pd.product-dist-mtopology*[*symmetric*] *hd*(*1*) *complete-metric-set.axioms*(*1*) *metric-set.bounded-dist-generate-so pd.complete-metric-set-axioms intro*!: *exI*[**where** *x*=*pd.product-dist*] *product-topology-cong*)
qed

lemma(**in** *complete-metrizable*) *g-delta-of-complete-metrizable*:
assumes *g-delta-of S B*
shows *complete-metrizable* (*subtopology S B*)
proof −
obtain *d* **where** *d*:*complete-metric-set* (*topspace S*) *d metric-set.mtopology* (*topspace*

S) *d* = *S*
by(*rule cmetric*)
interpret *m*: *complete-metric-set topspace S d* **by** *fact*
obtain *U* :: *nat* ⇒ - **where** *U*: ⋀*n*. *openin S* (*U n*) *B* = ⋂ (*range U*)
using *g-delta-ofD'*[*OF assms*] **by** *metis*
consider *topspace* (*subtopology S B*) = {} | *topspace* (*subtopology S B*) = *topspace S* | *topspace* (*subtopology S B*) ≠ {} *topspace* (*subtopology S B*) ⊂ *topspace S*
by (*metis assms g-delta-of-subset order-le-less topspace-subtopology-subset*)
then show *?thesis*
proof *cases*
case *1*
with *empty-metric-polish* **show** *?thesis*
by(*auto intro*!: *exI*[**where** *x*=λ*x y*. *0*] *simp*: *complete-metrizable-def polish-metric-set-def separable-metric-set-def Int-absorb1 assms empty-metric-mtopology g-delta-of-subset subtopology-eq-discrete-topology-eq*)
next
case *2*
then have *B* = *topspace S*
using *g-delta-of-subset*[*OF assms*] **by** *auto*
thus *?thesis*
by(*simp add*: *complete-metrizable-axioms*)
next
case *3*
then have *h*: *B* ≠ {} ⋀*n*. *U n* ≠ {} **by**(*auto simp*: *U*(*2*))
define *f* **where** *f* ≡ (λ*x*. (*x*, (λ*i*. *1* / *m.dist-set* (*topspace S* − (*U i*)) *x*)))
have *f-inj*:*inj f*
by(*auto simp*: *inj-def f-def*)
have *f-inv*: ⋀*x*. *x* ∈ *f* ' *B* ⟹ *f* (*fst x*) = *x* ⋀*x*. *fst* (*f x*) = *x*
by(*auto simp*: *f-def*)
have *continuous-map* (*subtopology S B*) (*prod-topology S* (*powertop-real UNIV*)) *f*
unfolding *continuous-map-pairwise continuous-map-componentwise-UNIV*
proof *safe*
have [*simp*]:*fst* ∘ *f* = *id*
by(*auto simp*: *f-def*)
show *continuous-map* (*subtopology S B*) *S* (*fst* ∘ *f*)
by *simp*
next
fix *k*
show *continuous-map* (*subtopology S B*) *euclideanreal* (λ*x*. (*snd* ∘ *f*) *x k*)
proof(*cases U k* = *topspace S*)
case *True*
then show *?thesis*
by(*simp add*: *f-def*)
next
case *False*
then have [*simp*]:(λ*x*. *snd* (*f x*) *k*) = (λ*x*. *1* / *m.dist-set* (*topspace S* − (*U k*)) *x*)
by(*simp add*: *f-def*)

have *continuous-map* (*subtopology S B*) *euclideanreal* ...
proof(*rule continuous-map-real-divide*)
show *continuous-map* (*subtopology S B*) *euclideanreal* (*m.dist-set* (*topspace S − U k*))
using *m.dist-set-continuous*[*simplified d*(*2*),*of topspace S − U k*]
by (*simp add*: *continuous-map-from-subtopology*)
next
fix *x*
assume *x* ∈ *topspace* (*subtopology S B*)
then have *h*′:*x* ∈ *topspace S x* ∈ *B* **by** *auto*
have *1*: *closedin S* (*topspace S − U k*) *topspace S − U k* ≠ {}
using *U*(*1*) *d*(*2*) *m.mtopology-openin-iff2 False* **by** *auto*
with *h*′(*2*) *m.dist-set-closed-ge0*[*simplified d*(*2*),*OF 1 h*′(*1*)]
show *m.dist-set* (*topspace S − U k*) *x* ≠ *0*
by(*auto simp*: *U*(*2*))
qed *simp*
thus *?thesis* **by** *simp*
qed
qed
hence *f-cont*: *continuous-map* (*subtopology S B*) (*subtopology* (*prod-topology S* (*powertop-real UNIV*)) (*f* ‘ *B*)) *f*
using *g-delta-of-subset*[*OF assms*] **by**(*auto simp*: *continuous-map-in-subtopology*)
have *f-invcont*: *continuous-map* (*subtopology* (*prod-topology S* (*powertop-real UNIV*)) (*f* ‘ *B*)) (*subtopology S B*) *fst*
by(*auto intro*!: *continuous-map-into-subtopology simp*: *continuous-map-subtopology-fst f-def*)

have *homeo*: *subtopology* (*prod-topology S* (*powertop-real UNIV*)) (*f* ‘ *B*) *homeomorphic-space subtopology S B*
using *f-inv*(*2*) **by**(*auto simp*: *homeomorphic-space-def homeomorphic-maps-def f-cont f-invcont intro*!: *exI*[**where** *x*=*f*] *exI*[**where** *x*=*fst*])

show *?thesis*
proof(*safe intro*!: *complete-metrizable.homeomorphic-complete-metrizable*[*OF - homeo*] *complete-metrizable.closedin-complete-metrizable*[*of - f* ‘ *B*] *complete-metrizable-prod complete-metrizable-product complete-metrizable-axioms*)
interpret *r*: *polish-metric-set UNIV dist* :: *real* ⇒ - **by** *simp*
interpret *pd*: *product-complete-metric 1*/*2 UNIV id id* λ*n*. *UNIV* λ*n*. *bounded-dist* (*dist* :: *real* ⇒ -) *1*
by(*auto intro*!: *product-complete-metric-natI*′ *simp*: *r.complete-metric-set-axioms*)
interpret *bpd*: *complete-metric-set topspace S* × (Π*E* *x*∈(*UNIV*::*nat set*). (*UNIV*::*real set*)) *binary-distance* (*topspace S*) *d* (Π*E* *x*∈(*UNIV*::*nat set*). (*UNIV*::*real set*)) *pd.product-dist*
using *pd.complete-metric-set-axioms* **by**(*auto intro*!: *binary-distance-complete d*(*1*))

have *closedin bpd.mtopology* (*f* ‘ *B*)
proof −
{ fix *a b* **and** *zn* :: *nat* ⇒ -

assume h':$zn \in UNIV \to f$ ' B $m.converge\text{-}to\text{-}inS$ $(\lambda n.\ fst\ (zn\ n))$ a $\forall i.$ $r.converge\text{-}to\text{-}inS$ $(\lambda n.\ snd\ (zn\ n)\ i)$ $(b\ i)$
then obtain xn **where** xn: $\bigwedge n.\ xn\ n \in B$ $\bigwedge n.\ zn\ n = f\ (xn\ n)$
by (*metis PiE UNIV-I f-inv*(2) *imageE*)

have h: $m.converge\text{-}to\text{-}inS$ xn a $\bigwedge i.$ $r.converge\text{-}to\text{-}inS$ $(\lambda n.\ 1\ /\ m.dist\text{-}set$ $(topspace\ S - U\ i)\ (xn\ n))\ (b\ i)$
proof −
{
fix i
have $(\lambda n.\ snd\ (zn\ n)\ i) = (\lambda n.\ 1\ /\ m.dist\text{-}set\ (topspace\ S - U\ i)\ (xn$ $n))$
by *standard* (*simp add*: *xn*(2) *f-def*)
}
thus $m.converge\text{-}to\text{-}inS$ xn a $\bigwedge i.$ $r.converge\text{-}to\text{-}inS$ $(\lambda n.\ 1\ /\ m.dist\text{-}set$ $(topspace\ S - U\ i)\ (xn\ n))\ (b\ i)$
using h' **by**(*auto simp*: *xn*(2) *f-def*)
qed
have *conv1*: $r.converge\text{-}to\text{-}inS$ $(\lambda n.\ m.dist\text{-}set\ (topspace\ S - U\ i)\ (xn\ n))$ $(m.dist\text{-}set\ (topspace\ S - U\ i)\ a)$ **for** i
using $m.dist\text{-}set\text{-}continuous\ h(1)$ **by**(*simp add*: *metric-set-continuous-map-eq'*[*OF m.metric-set-axioms r.metric-set-axioms*,*simplified euclidean-mtopology*])
have *dista-n0*:$m.dist\text{-}set\ (topspace\ S - U\ i)\ a \neq 0$ **if** $U\ i \neq topspace\ S$ **for** i
proof(*rule LIMSEQ-inverse-not0*[*OF - conv1*[*simplified converge-to-def-set*[*symmetric*]] *h*(2)[*simplified converge-to-def-set*[*symmetric*]]])
fix n
have $0 < m.dist\text{-}set\ (topspace\ S - U\ i)\ (xn\ n)$
using *xn*(1) *U*(1)[*of i*] **by**(*auto intro*!: *m.dist-set-closed-ge0 simp*: *U*(2) *d*(2) *in-mono openin-subset*) (*use openin-subset that* **in** *blast*)+
thus $m.dist\text{-}set\ (topspace\ S - U\ i)\ (xn\ n) \neq 0$ **by** *simp*
qed
from *tendsto-inverse-real*[*OF conv1*[*simplified converge-to-def-set*[*symmetric*]] *this*]
have *conv1'*:$r.converge\text{-}to\text{-}inS$ $(\lambda n.\ 1\ /\ m.dist\text{-}set\ (topspace\ S - U\ i)\ (xn$ $n))\ (1\ /\ m.dist\text{-}set\ (topspace\ S - U\ i)\ a)$ **if** $U\ i \neq topspace\ S$ **for** i
by(*simp add*: *that converge-to-def-set*)
have $a \in U\ n$ **for** n
proof(*cases* $U\ n = topspace\ S$)
case *True*
then show *?thesis*
using $h'(2)$ **by**(*auto simp*: *m.converge-to-inS-def*)
next
case *False*
with *m.dist-set-nzeroD*(2)[*OF dista-n0*[*OF this*]] *dista-n0*
show *?thesis*
by *fastforce*
qed
hence $a \in B$

```
      by(auto simp: U(2))
    moreover have b n = 1 / m.dist-set (topspace S − (U n)) a for n
    proof(cases U n = topspace S)
      case True
      then show ?thesis
        using h(2)[of n,simplified converge-to-def-set[symmetric]]
        by (simp add: LIMSEQ-const-iff)
    next
      case False
      from conv1'[OF this] h(2)[of n]
      show ?thesis
        by(simp add: r.converge-to-inS-unique)
    qed
    ultimately have (a, b) ∈ f ‘ B
      by(auto simp: f-def image-def)
  }
  thus ?thesis
 unfolding bpd.mtopology-closedin-iff binary-distance-converge-to-inS-iff'[OF
m.metric-set-axioms pd.metric-set-axioms]
  using pd.converge-to-iff[simplified r.bounded-dist-converge-to-inS-iff[symmetric]]
g-delta-of-subset[OF assms] f-def
    by auto
 qed
 thus closedin (prod-topology S (powertop-real UNIV)) (f ‘ B)
 by(simp only: binary-distance-mtopology[OF m.metric-set-axioms pd.metric-set-axioms]
pd.product-dist-mtopology[symmetric] r.bounded-dist-generate-same-topology[symmetric]
euclidean-mtopology d(2))
 qed simp-all
 qed
qed

corollary(in complete-metrizable) openin-complete-metrizable:
 assumes openin S u
 shows complete-metrizable (subtopology S u)
 using assms by(auto intro!: g-delta-of-complete-metrizable )
```

3.3 Polish Spaces

```
locale polish-topology = complete-metrizable +
 assumes S-separable:separable S
begin

lemma S-second-countable: second-countable S
 by(rule second-countable-if-separable[OF S-separable])

lemma closedin-polish:
 assumes closedin S A
 shows polish-topology (subtopology S A)
 by (simp add: S-second-countable assms closedin-complete-metrizable polish-topology-axioms-def
```

polish-topology-def second-countable-subtopology separable-if-second-countable)

lemma *g-delta-of-polish*:
assumes *g-delta-of S A*
shows *polish-topology* (*subtopology S A*)
by(*simp add*: *polish-topology-def g-delta-of-complete-metrizable*[*OF assms*] *polish-topology-axioms-def S-second-countable second-countable-subtopology separable-if-second-countable*)

corollary *openin-polish*:
assumes *openin S A*
shows *polish-topology* (*subtopology S A*)
by (*simp add*: *assms g-delta-of-polish*)

lemma *homeomorphic-polish-topology*:
assumes *S homeomorphic-space S'*
shows *polish-topology S'*
by(*simp add*: *polish-topology-def homeomorphic-complete-metrizable*[*OF assms*] *homeomorphic-separable*[*OF S-separable assms*] *polish-topology-axioms-def*)

end

lemma *polish-topology-def2*:
polish-topology S $\longleftrightarrow$ ($\exists$ *ϱ*. *polish-metric-set* (*topspace S*) *ϱ* $\wedge$ *S* = *metric-set.mtopology* (*topspace S*) *ϱ*)
by (*metis complete-metric-set.axioms*(*1*) *complete-metrizable-def metric-set.separable-iff-topological-separable polish-metric-set.axioms*(*1*) *polish-metric-set.axioms*(*2*) *polish-metric-set.intro polish-topology-axioms-def polish-topology-def*)

lemma(**in** *polish-topology*) *polish-metric*:
obtains *d* **where** *polish-metric-set* (*topspace S*) *d*
and *S* = *metric-set.mtopology* (*topspace S*) *d*
using *polish-topology-axioms* **by**(*auto simp*: *polish-topology-def2*)

lemma(**in** *polish-topology*) *bounded-polish-metric*:
obtains *d* **where** *polish-metric-set* (*topspace S*) *d*
and *S* = *metric-set.mtopology* (*topspace S*) *d*
and $\bigwedge x\ y.\ d\ x\ y < 1$
proof −
obtain *d* **where** *d*:*polish-metric-set* (*topspace S*) *d S* = *metric-set.mtopology* (*topspace S*) *d*
by(*rule polish-metric*)
interpret *d*: *polish-metric-set topspace S d* **by** *fact*
have $\exists d'.$ *polish-metric-set* (*topspace S*) *d'* $\wedge$ *S* = *metric-set.mtopology* (*topspace S*) *d'* $\wedge$ ($\forall x\ y.\ d'\ x\ y < 1$)
using *d* **by**(*auto intro*!: *exI*[**where** *x*=*bounded-dist d*] *polish-metric-set.bounded-dist-polish simp*:*d.bounded-dist-generate-same-topology d.bounded-dist-dist*)
with *that* **show** *?thesis*
by *auto*
qed

sublocale *polish-metric-set* $\subseteq$ *polish-topology mtopology*
using *mtopology-topspace* **by**(*auto simp*: *polish-topology-def2 polish-metric-set-axioms intro*!: *exI*[**where** *x*=*dist*])

lemma *polish-topology-euclidean*[*simp*]: *polish-topology* (*euclidean* :: ('*a* :: *polish-space*) *topology*)
using *polish-class-polish-set*
by(*auto simp*: *polish-topology-def2 intro*!: *exI*[**where** *x*=*dist*]) (*use open-openin open-openin-set topology-eq* **in** *blast*)

lemma *polish-topology-countable*[*simp*]:
polish-topology (*euclidean* :: '*a* :: {*countable*,*discrete-topology*} *topology*)
proof −
interpret *polish-metric-set UNIV* :: '*a set discrete-dist UNIV*
by(*simp add*: *discrete-dist-polish-iff*)
show *?thesis*
unfolding *polish-topology-def2*
by(*auto intro*!: *exI*[**where** *x*=*discrete-dist UNIV*] *simp*: *topology-eq polish-metric-set-axioms discrete-dist-topology*[*of UNIV* :: '*a set*] *discrete-topology-class.open-discrete*)
qed

lemma *polish-topology-prod*:
assumes *polish-topology S* **and** *polish-topology S'*
shows *polish-topology* (*prod-topology S S'*)
proof −
obtain *ϱ ϱ'* **where** *hr*:
polish-metric-set (*topspace S*) *ϱ S* = *metric-set.mtopology* (*topspace S*) *ϱ*
polish-metric-set (*topspace S'*) *ϱ' S'* = *metric-set.mtopology* (*topspace S'*) *ϱ'*
using *assms* **by**(*auto simp*: *polish-topology-def2*)
interpret *m1*:*polish-metric-set topspace S ϱ* **by** *fact*
interpret *m2*:*polish-metric-set topspace S' ϱ'* **by** *fact*
interpret *m*: *polish-metric-set topspace S* × *topspace S' binary-distance* (*topspace S*) *ϱ* (*topspace S'*) *ϱ'*
by(*auto intro*!: *binary-distance-polish simp*: *m1.polish-metric-set-axioms m2.polish-metric-set-axioms*)
show *?thesis*
unfolding *polish-topology-def2*
using *binary-distance-mtopology*[*OF m1.metric-set-axioms m2.metric-set-axioms*,*simplified space-pair-measure*[*symmetric*]] *hr*(*2*,*4*)
by(*auto intro*!: *exI*[**where** *x*=*binary-distance* (*topspace S*) *ϱ* (*topspace S'*) *ϱ'*] *m.polish-metric-set-axioms*)
qed

lemma *polish-topology-product*:
assumes *countable I* **and** ⋀*i*. *i* ∈ *I* ⟹ *polish-topology* (*S i*)
shows *polish-topology* (*product-topology S I*)
proof −
obtain *ϱ* **where** *hr*:
⋀*i*. *i* ∈ *I* ⟹ *polish-metric-set* (*topspace* (*S i*)) (*ϱ i*) ⋀*i*. *i* ∈ *I* ⟹ *S i* =

metric-set.mtopology (*topspace* (*S i*)) (*ϱ i*)
using *assms*(*2*) **by**(*auto simp*: *polish-topology-def2*) *metis*
define *ϱ′* **where** *ϱ′* ≡ (*λi x y*. *if i* ∈ *I then bounded-dist* (*ϱ i*) *x y else 0*)
interpret *pd*: *product-polish-metric 1/2 I to-nat-on I from-nat-into I λi*. *topspace* (*S i*) *ϱ′ 1*
using *assms hr* **by**(*auto intro*!: *product-polish-metricI′ simp*: *ϱ′-def*)
have *product-topology S I* = *product-topology* (*λi*. *metric-set.mtopology* (*topspace* (*S i*)) (*ϱ i*)) *I*
by(*auto intro*!: *product-topology-cong hr*(*2*))
also have ... = *product-topology* (*λi*. *metric-set.mtopology* (*topspace* (*S i*)) (*ϱ′ i*)) *I*
by(*auto intro*!: *product-topology-cong simp*: *ϱ′-def*)
(*use hr*(*1*) *metric-set.bounded-dist-generate-same-topology polish-metric-set.axioms*(*2*) *separable-metric-set-def* **in** *blast*)
also have ... = *pd.mtopology*
by(*rule pd.product-dist-mtopology*)
finally have *product-topology S I* = *pd.mtopology* **.**
show *?thesis*
unfolding *polish-topology-def2*
by(*auto intro*!: *exI*[**where** *x*=*pd.product-dist*] *simp*: *pd.polish-metric-set-axioms*) *fact*
qed

lemma *polish-topology-closedin-polish*:
assumes *polish-topology S* **and** *closedin S U*
shows *polish-topology* (*subtopology S U*)
proof −
obtain *ϱ* **where** ∗:
polish-metric-set (*topspace S*) *ϱ S* = *metric-set.mtopology* (*topspace S*) *ϱ*
using *assms* **by**(*auto simp*: *polish-topology-def2*)
interpret *m*:*polish-metric-set topspace S ϱ* **by** *fact*
interpret *m′*:*polish-metric-set U submetric U ϱ*
using *m.submetric-complete-iff*[*OF closedin-subset*[*OF assms*(*2*)]] *m.submetric-separable*[*OF closedin-subset*[*OF assms*(*2*)]] *assms*(*2*) ∗
by(*simp add*: *polish-metric-set-def*)
have *subtopology S U* = *m′.mtopology*
using *m.submetric-subtopology*[*OF closedin-subset*[*OF assms*(*2*)]] ∗ **by** *simp*
thus *?thesis*
using *m′.mtopology-topspace*
by(*auto simp*: *polish-topology-def2 m′.polish-metric-set-axioms intro*!: *exI*[**where** *x*=*submetric U ϱ*])
qed

3.4 Compact Metrizable Spaces

locale *compact-metrizable* = *metrizable* +
assumes *compact*: *compact-space S*
begin

sublocale *polish-topology*
proof −
obtain *d* **where** *compact-metric-set* (*topspace S*) *d metric-set.mtopology* (*topspace S*) *d* = *S*
using *metric compact* **by**(*auto simp*: *compact-metric-set-def compact-metric-set-axioms.intro*)
then interpret *m*: *polish-metric-set topspace S d*
by(*simp add*: *compact-metric-set.polish*)
show *polish-topology S*
using ‹*m.mtopology* = *S*› *m.polish-topology-axioms* **by** *simp*
qed

lemma *compact-metric*:
obtains *d* **where** *compact-metric-set* (*topspace S*) *d metric-set.mtopology* (*topspace S*) *d* = *S*
by (*metis metric compact compact-metric-set.intro compact-metric-set-axioms.intro*)

end

3.5 Continuous Embddings

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

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

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

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

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

lemma *Cantor-space-Polish-topology*:
polish-topology Cantor-space-as-topology
by(*auto intro*!: *polish-topology-closedin-polish polish-topology-product*)

Proposition 2.2.3 in [3]

lemma *continuous-map-metrizable-extension*:
assumes *A* ⊆ *topspace W metrizable W complete-metrizable Z continuous-map* (*subtopology W A*) *Z f*
shows ∃ *h gd*. *g-delta-of W gd* ∧ (∀ *a*∈*A*. *f a* = *h a*) ∧ *A* ⊆ *gd* ∧ *continuous-map* (*subtopology W gd*) *Z h*
proof −
obtain *dz* **where** *hdz*: *complete-metric-set* (*topspace Z*) *dz metric-set.mtopology*

(*topspace Z*) *dz* = *Z* ⋀*x y*. *dz x y* < *1*
using *complete-metrizable.bounded-cmetric*[*OF assms*(*3*)] **by** *auto*
interpret *dz*: *complete-metric-set topspace Z dz* **by** *fact*
obtain *dw* **where** *hdw*: *metric-set* (*topspace W*) *dw metric-set.mtopology* (*topspace W*) *dw* = *W*
using *metrizable.metric*[*OF assms*(*2*)] **by** *auto*
interpret *dw*: *metric-set topspace W dw* **by** *fact*
interpret *subd*: *metric-set A submetric A dw*
using *assms* **by**(*auto intro*!: *dw.submetric-metric-set*)
have *subd.mtopology* = *subtopology W A*
using *assms*(*1*) *dw.submetric-subtopology hdw*(*2*) **by** *auto*
let *?oscf* = *dz.osc-on A W f*
define *gd* **where** *gd* ≡ {*x*∈*W closure-of A*. *?oscf x* = *0*}
have *g-delta*: *g-delta-of W gd*
proof −
have ∗:{*x*∈*W closure-of A*. *?oscf x* < *t*} = ⋃ {*V* ∩ (*W closure-of A*)| *V*. *openin W V* ∧ *dz.diam* (*f* ‘ (*A* ∩ *V*)) < *t*} **for** *t*
by(*auto simp*: *dz.osc-on-less-iff*)
have *1*:*gd* = ⋂ {{*x*∈*W closure-of A*. *?oscf x* < *1* / *real n*}|*n*. *n* ∈ {*0*<..}}
proof −
have *x* ∈ *gd* **if** *h*:⋀*n*. *n* ∈ {*0*<..} ⟹ *x* ∈ {*x*∈*W closure-of A*. *?oscf x* < *1* / *real n*} **for** *x*
proof −
have *?oscf x* < *ε* **if** *he*:*ε*>*0* **for** *ε*
proof −
obtain *n* **where** *1* / *real* (*Suc n*) < *ε*
by (*meson enn2real-le-iff enn2real-positive-iff ennreal-less-top ennreal-less-zero-iff he linorder-not-le nat-approx-posE order-le-less-trans*)
thus *?thesis*
using *h*[*of Suc n*] **by** *auto*
qed
hence *?oscf x* = *0*
using *not-gr-zero* **by** *blast*
thus *?thesis*
using *that* **by**(*auto simp*: *gd-def*)
qed
thus *?thesis*
by (*auto simp*: *gd-def*)
qed
also have ... = ⋂ {⋃ {*V* ∩ (*W closure-of A*)| *V*. *openin W V* ∧ *dz.diam* (*f* ‘ (*A* ∩ *V*)) < *1* / *real n*}|*n*. *n* ∈ {*0*<..}}
using ∗ **by** *auto*
also have ... = *W closure-of A* ∩ ⋂ {⋃ {*V*. *openin W V* ∧ *dz.diam* (*f* ‘ (*A* ∩ *V*)) < *1* / *real n*}|*n*. *n* ∈ {*0*<..}}
by *blast*
also have *g-delta-of W* ...
proof −
have {⋃ {*V*. *openin W V* ∧ *dz.diam* (*f* ‘ (*A* ∩ *V*)) < *ennreal* (*1* / *real n*)} | *n*. *0* < *n*} = (λ*n*. ⋃ {*V*. *openin W V* ∧ *dz.diam* (*f* ‘ (*A* ∩ *V*)) < *ennreal* (*1* /

real n)}) ‘ {*0*<..} **by** *auto*
also have *countable* ... **by** *auto*
finally show *?thesis*
by(*auto intro*!: *dw.g-delta-of-closed*[*simplified hdw*(*2*),*of W closure-of A*]
g-delta-of-inter[*OF - g-delta-ofI*[*of* {⋃ {*V*. *openin W V* ∧ *dz.diam* (*f* ‘ (*A* ∩ *V*))
< *ennreal* (*1* / *real n*)} | *n*. *n* ∈ {*0*<..}} - ⋂ {⋃ {*V*. *openin W V* ∧ *dz.diam* (*f*
‘ (*A* ∩ *V*)) < *1* / *real n*}|*n*. *0* < *n*}]])
qed
finally show *?thesis* **.**
qed
have *oscf0*: *?oscf a* = *0* **if** *a* ∈ *A* **for** *a*
using *assms that* **by**(*auto intro*!: *osc-on-inA-0*[*OF dw.metric-set-axioms dz.metric-set-axioms*,*simplified*
‹*dz.mtopology* = *Z*› ‹*dw.mtopology* = *W*›] *simp*: *le-iff-inf*)
hence *A-subst-of-gd*: *A* ⊆ *gd*
using *closure-of-subset*[*OF assms*(*1*)] **by**(*auto simp add*: *gd-def*)
define *h* **where** *h x* ≡ *let xn* = (*SOME an*. *an* ∈ *UNIV* → *A* ∧ *dw.converge-to-inS*
an x) *in dz.the-limit-of* (λ*n*. *f* (*xn n*)) **for** *x*
have *h-extends*:*f a* = *h a* **if** *a* ∈ *A* **for** *a*
proof −
obtain *an* **where** *han*: *an* ∈ *UNIV* → *A dw.converge-to-inS an a*
using *dw.closure-of-mtopology-an*[*of a A*] *A-subst-of-gd* ‹*a* ∈ *A*› *gd-def hdw*(*2*)
by *auto*
show *?thesis*
unfolding *h-def Let-def*
proof(*rule someI2*[*of - an* λ*t*. *f a* = *dz.the-limit-of* (λ*n*. *f* (*t n*))])
fix *bn*
assume *h*:*bn* ∈ *UNIV* → *A* ∧ *dw.converge-to-inS bn a*
hence *subd.converge-to-inS bn a*
using *assms*(*1*) *dw.convergent-insubmetric that* **by** *fastforce*
hence *dz.converge-to-inS* (λ*n*. *f* (*bn n*)) (*f a*)
using *metric-set-continuous-map-eq'*[*OF subd.metric-set-axioms dz.metric-set-axioms*,*of*
f,*simplified* ‹*subd.mtopology* = *subtopology W A*› ‹*dz.mtopology* = *Z*› *assms*(*4*)]
by *auto*
thus *f a* = *dz.the-limit-of* (λ*n*. *f* (*bn n*))
by(*simp add*: *dz.the-limit-of-eq*)
qed(*use han* **in** *auto*)
qed
have *gd* ⊆ *topspace W*
by(*simp add*: *gd-def in-closure-of*)
then interpret *subd-on-gd*: *metric-set gd submetric gd dw*
by(*auto intro*!: *dw.submetric-metric-set*)
have *subtopology W gd* = *subd-on-gd.mtopology*
using ‹*gd* ⊆ *topspace W*› *dw.submetric-subtopology hdw*(*2*) **by** *auto*
have *Cauchyf*:*dz.Cauchy-inS* (λ*n*. *f* (*an n*)) **if** *subd.Cauchy-inS an dw.converge-to-inS*
an a ?oscf a = *0* **for** *an a*
proof −
have {*dz.diam* (*f* ‘ (*A* ∩ *U*)) |*U*. *a* ∈ *U* ∧ *openin W U*} = (λ*U*. *dz.diam* (*f* ‘
(*A* ∩ *U*))) ‘ {*U*. *a* ∈ *U* ∧ *openin W U*}
by *auto*

```
    hence (⨅ i∈{U. a ∈ U ∧ openin W U}. dz.diam (f ‘ (A ∩ i))) = ⊥
      using that(3) by(auto simp: dz.osc-on-def bot-ennreal)
    from this[simplified INF-eq-bot-iff]
    have ⋀ε. ε > 0 ⟹ ∃ u∈{U. a ∈ U ∧ openin W U}. dz.diam (f ‘ (A ∩ u)) <
ε
      by(simp add: bot-ennreal)
    hence he:⋀ε. ε > 0 ⟹ ∃ u∈{U. a ∈ U ∧ openin W U}. dz.diam (f ‘ (A ∩
u)) < ennreal ε
      by auto
    show ?thesis
      unfolding dz.Cauchy-inS-def
    proof safe
      show ⋀x. f (an x) ∈ topspace Z
        using assms(1,4) subd.Cauchy-inS-dest1[OF that(1)] by(auto simp: con-
tinuous-map-def)
    next
      fix ε
      assume (0 :: real) < ε
      from he[OF this] obtain U where hu:a ∈ U openin W U dz.diam (f ‘ (A ∩
U)) < ennreal ε
        by auto
      then obtain e where he:e > 0 a ∈ dw.open-ball a e dw.open-ball a e ⊆ U
      by (metis ‹dw.converge-to-inS an a› dw.metric-set-axioms dw.mtopology-openin-iff
dw.open-ball-ina hdw(2) metric-set.converge-to-inS-def2')
      then obtain N where ⋀n. n ≥ N ⟹ an n ∈ dw.open-ball a e
        using ‹dw.converge-to-inS an a› dw.converge-to-inS-def2' by blast
      hence hn: ⋀n. n ≥ N ⟹ an n ∈ A ∩ U
        using he(3) that(1) by(auto simp: subd.Cauchy-inS-def)
      show ∃ N. ∀ n≥N. ∀ m≥N. dz (f (an n)) (f (an m)) < ε
      proof(safe intro!: exI[where x=N])
        fix n m
        assume N ≤ n N ≤ m
        then have an n ∈ A ∩ U an m ∈ A ∩ U
          using hn by auto
        hence f (an n) ∈ f ‘ (A ∩ U) f (an m) ∈ f ‘ (A ∩ U)
          by auto
        then have ennreal (dz (f (an n)) (f (an m))) ≤ dz.diam (f ‘ (A ∩ U))
          using assms(4) subd.mtopology-topspace by(auto intro!: dz.diam-is-sup
simp:‹subd.mtopology = subtopology W A› continuous-map-def)
        also have ... < ennreal ε by fact
        finally show dz (f (an n)) (f (an m)) < ε
          using dz.dist-geq0 by (simp add: ennreal-less-iff)
      qed
    qed
  qed
  have continuous-map (subtopology W gd) Z h
  proof −
    have h-image:h ∈ gd → topspace Z
    proof
```

fix x
assume $x \in gd$
then obtain xn **where** hxn: $xn \in UNIV \to A$ $dw.converge\text{-}to\text{-}inS\ xn\ x$
using $dw.closure\text{-}of\text{-}mtopology\text{-}an[of\ x\ A]$ **by**($auto\ simp$: $gd\text{-}def\ hdw(2)$)
show $h\ x \in topspace\ Z$
unfolding $h\text{-}def\ Let\text{-}def$
proof($rule\ someI2[of\ -\ xn\ \lambda t.\ dz.the\text{-}limit\text{-}of\ (\lambda n.\ f\ (t\ n)) \in topspace\ Z]$)
fix an
assume $an \in subd.sequence \wedge dw.converge\text{-}to\text{-}inS\ an\ x$
then have h:$an \in subd.sequence\ dw.converge\text{-}to\text{-}inS\ an\ x$ **by** $auto$
then have $dz.Cauchy\text{-}inS\ (\lambda n.\ f\ (an\ n))$
using ‹$x \in gd$› **by**($auto\ intro!$: $Cauchyf[OF\ dw.Cauchy\text{-}insub\text{-}Cauchy\text{-}inverse[OF\ assms(1)\ h(1)\ dw.Cauchy\text{-}if\text{-}convergent\text{-}inS]\ h(2)]\ simp$: $gd\text{-}def\ dw.convergent\text{-}inS\text{-}def$)
thus $dz.the\text{-}limit\text{-}of\ (\lambda n.\ f\ (an\ n)) \in topspace\ Z$
by($simp\ add$: $dz.convergence\ dz.the\text{-}limit\text{-}of\text{-}inS$)
qed($use\ hxn$ **in** $auto$)
qed
show $?thesis$
unfolding $metric\text{-}set\text{-}continuous\text{-}map\text{-}eq[OF\ subd\text{-}on\text{-}gd.metric\text{-}set\text{-}axioms\ dz.metric\text{-}set\text{-}axioms,simplified\ ‹subtopology\ W\ gd = subd\text{-}on\text{-}gd.mtopology›[symmetric]\ ‹dz.mtopology = Z›]$
proof $safe$
fix x **and** ε :: $real$
assume $x \in gd\ 0 < \varepsilon$
then have $?oscf\ x < \varepsilon\ /\ 3$
by($auto\ simp$: $gd\text{-}def$)
then obtain u **where** hu: $openin\ W\ u\ x \in u\ dz.diam\ (f\ `\ (A \cap u)) < \varepsilon\ /\ 3$
by($auto\ simp$: $dz.osc\text{-}on\text{-}def\ Inf\text{-}less\text{-}iff$)
hence $openin\ subd\text{-}on\text{-}gd.mtopology\ (u \cap gd)$
by($auto\ simp$: ‹$subtopology\ W\ gd = subd\text{-}on\text{-}gd.mtopology$›$[symmetric]\ openin\text{-}subtopology$)
then obtain δ **where** hd: $\delta > 0\ subd\text{-}on\text{-}gd.open\text{-}ball\ x\ \delta \subseteq u \cap gd\ x \in subd\text{-}on\text{-}gd.open\text{-}ball\ x\ \delta$
by ($metis\ Int\text{-}iff$ ‹$x \in gd$› $hu(2)\ subd\text{-}on\text{-}gd.mtopology\text{-}openin\text{-}iff\ subd\text{-}on\text{-}gd.open\text{-}ball\text{-}ina$)
show $\exists \delta{>}0.\ \forall y{\in}gd.\ submetric\ gd\ dw\ x\ y < \delta \longrightarrow dz\ (h\ x)\ (h\ y) < \varepsilon$
proof($safe\ intro!$: $exI[\mathbf{where}\ x{=}\delta]$ ‹$\delta > 0$›)
fix y
assume h':$y \in gd\ submetric\ gd\ dw\ x\ y < \delta$
then have $y \in subd\text{-}on\text{-}gd.open\text{-}ball\ x\ \delta$
by($simp\ add$: ‹$x \in gd$› $subd\text{-}on\text{-}gd.open\text{-}ball\text{-}def$)
then obtain δy **where** hdy: $\delta y > 0\ subd\text{-}on\text{-}gd.open\text{-}ball\ y\ \delta y \subseteq subd\text{-}on\text{-}gd.open\text{-}ball\ x\ \delta\ y \in subd\text{-}on\text{-}gd.open\text{-}ball\ y\ \delta y$
using $h'(1)\ subd\text{-}on\text{-}gd.mtopology\text{-}open\text{-}ball\text{-}in'\ subd\text{-}on\text{-}gd.open\text{-}ball\text{-}ina$
by $blast$
obtain $xn'\ yn'$ **where** $hxyn'$:$xn' \in UNIV \to A\ dw.converge\text{-}to\text{-}inS\ xn'\ x\ yn' \in UNIV \to A\ dw.converge\text{-}to\text{-}inS\ yn'\ y$
using $dw.closure\text{-}of\text{-}mtopology\text{-}an[of\ -\ A]$ ‹$y \in gd$› ‹$x \in gd$› **by**($simp\ add$: $gd\text{-}def\ hdw(2)$) $metis$
show $dz\ (h\ x)\ (h\ y) < \varepsilon$

```
    proof −
      { fix xn yn
        assume hxyn:xn ∈ subd.sequence dw.converge-to-inS xn x
                  yn ∈ subd.sequence dw.converge-to-inS yn y
       then have Cauchyxyn: dz.Cauchy-inS (λn. f (xn n)) dz.Cauchy-inS (λn.
f (yn n))
              using Cauchyf[OF dw.Cauchy-insub-Cauchy-inverse[OF assms(1)
hxyn(1) dw.Cauchy-if-convergent-inS] hxyn(2)] Cauchyf[OF dw.Cauchy-insub-Cauchy-inverse[OF
assms(1) hxyn(3) dw.Cauchy-if-convergent-inS] hxyn(4)] ‹x ∈ gd› ‹y ∈ gd›
          by(auto simp: gd-def dw.convergent-inS-def)
        have convxyn:subd-on-gd.converge-to-inS xn x subd-on-gd.converge-to-inS
yn y
       using hxyn ‹x ∈ gd› ‹y ∈ gd› ‹A ⊆ gd› by(auto intro!: dw.convergent-insubmetric
‹gd ⊆ topspace W›)
      then obtain Nx Ny where hnxy: ⋀n. n ≥ Nx ⟹ xn n ∈ subd-on-gd.open-ball
x δ ⋀n. n ≥ Ny ⟹ yn n ∈ subd-on-gd.open-ball y δy
          using hd(1) hdy(1) subd-on-gd.converge-to-inS-def2′ by blast
        have 0 < ε / 3 using ‹0 < ε› by simp
           obtain Nfx Nfy where hnfxy: ⋀n. n ≥ Nfx ⟹ dz (f (xn n))
(dz.the-limit-of (λn. f (xn n))) < ε / 3 ⋀n. n ≥ Nfy ⟹ dz (f (yn n)) (dz.the-limit-of
(λn. f (yn n))) < ε / 3
            using dz.the-limit-if-converge[OF dz.convergence[OF Cauchyxyn(1)]]
dz.the-limit-if-converge[OF dz.convergence[OF Cauchyxyn(2)]]
       by(auto simp: dz.converge-to-inS-def2) (meson ‹0 < ε / 3› less-divide-eq-numeral1(1))

        define N where N ≡ Max {Nx,Ny,Nfx,Nfy}
        have N:N ≥ Nx N ≥ Ny N ≥ Nfx N ≥ Nfy
          by(simp-all add: N-def)
        have dz (dz.the-limit-of (λn. f (xn n))) (dz.the-limit-of (λn. f (yn n)))
< ε
            (is ?lhs < -)
        proof −
         have ?lhs ≤ dz (dz.the-limit-of (λn. f (xn n))) (f (xn N)) + dz (f (xn
N)) (dz.the-limit-of (λn. f (yn n)))
                using dz.dist-tr[OF dz.the-limit-of-inS[OF dz.convergence[OF
Cauchyxyn(1)]] - dz.the-limit-of-inS[OF dz.convergence[OF Cauchyxyn(2)]],of ‹f
(xn N)›] dz.Cauchy-inS-dest1[OF Cauchyxyn(1)]
           by simp
         also have ... ≤ dz (dz.the-limit-of (λn. f (xn n))) (f (xn N)) + dz (f
(xn N)) (f (yn N)) + dz (f (yn N)) (dz.the-limit-of (λn. f (yn n)))
              using dz.dist-tr[OF - - dz.the-limit-of-inS[OF dz.convergence[OF
Cauchyxyn(2)]],of f (xn N) f (yn N)] dz.Cauchy-inS-dest1[OF Cauchyxyn(1)]
dz.Cauchy-inS-dest1[OF Cauchyxyn(2)]
           by simp
         also have ... < ε / 3 + dz (f (xn N)) (f (yn N)) + ε / 3
          using hnfxy[of N] N by(simp add: dz.dist-sym[of dz.the-limit-of (λn.
f (xn n))])
         also have ... < ε
         proof −
```

```
            have xn N ∈ A ∩ u yn N ∈ A ∩ u
              using hdy(2) hd(2) hnxy[of N] N hxyn(1,3) by auto
            hence ennreal (dz (f (xn N)) (f (yn N))) ≤ dz.diam (f ‘ (A ∩ u))
           by(auto intro!: dz.diam-is-sup dz.Cauchy-inS-dest1[OF Cauchyxyn(1)]
dz.Cauchy-inS-dest1[OF Cauchyxyn(2)])
            also have ... < ennreal (ε / 3) by fact
            finally have dz (f (xn N)) (f (yn N)) < ε / 3
              using dz.dist-geq0 ennreal-less-iff by blast
            thus ?thesis by simp
          qed
          finally show ?thesis .
        qed
      }
      note h = this
      show ?thesis
        apply(simp only: h-def[of x] Let-def)
         apply(rule someI2[of λk. k ∈ subd.sequence ∧ dw.converge-to-inS k x
xn',OF conjI[OF hxyn'(1,2)]])
        apply(simp only: h-def[of y] Let-def)
         apply(rule someI2[of λk. k ∈ subd.sequence ∧ dw.converge-to-inS k y
yn',OF conjI[OF hxyn'(3,4)]])
        using h by auto
     qed
    qed
  qed(use h-image in auto)
 qed

 with h-extends g-delta A-subst-of-gd
 show ?thesis by auto
qed

lemma Lavrentiev-theorem:
  assumes complete-metrizable X complete-metrizable Y A ⊆ topspace X B ⊆
topspace Y homeomorphic-map (subtopology X A) (subtopology Y B) f
 shows ∃h gda gdb. g-delta-of X gda ∧ g-delta-of Y gdb ∧ A ⊆ gda ∧ B ⊆ gdb ∧
(∀x∈A. f x = h x) ∧ homeomorphic-map (subtopology X gda) (subtopology Y gdb)
h
proof −
 interpret cmx: complete-metrizable X by fact
 interpret cmy: complete-metrizable Y by fact
 interpret mxy: metrizable prod-topology X Y
  by(auto intro!: metrizable-prod cmx.metrizable cmy.metrizable)
 obtain g where homeomorphic-maps (subtopology X A) (subtopology Y B) f g
  using assms(5) homeomorphic-map-maps by blast
 then have hfg: continuous-map (subtopology X A) (subtopology Y B) f continu-
ous-map (subtopology Y B) (subtopology X A) g
            ⋀x. x ∈ A ⟹ g (f x) = x ⋀y. y ∈ B ⟹ f (g y) = y
  using assms(3,4) by(auto simp: homeomorphic-maps-def)
 obtain f' g' gda gdb where h:
```

g-delta-of X gda $\bigwedge a.\ a \in A \Longrightarrow f\ a = f'\ a\ A \subseteq gda$ *continuous-map* (*subtopology X gda*) *Y f'*
g-delta-of Y gdb $\bigwedge b.\ b \in B \Longrightarrow g\ b = g'\ b\ B \subseteq gdb$ *continuous-map* (*subtopology Y gdb*) *X g'*
using *continuous-map-metrizable-extension*[*OF assms*(*3*) *cmx.metrizable assms*(*2*) *continuous-map-into-fulltopology*[*OF hfg*(*1*)]]
continuous-map-metrizable-extension[*OF assms*(*4*) *cmy.metrizable assms*(*1*) *continuous-map-into-fulltopology*[*OF hfg*(*2*)]]
by *auto*
define *H* **where** $H \equiv SIGMA\ x{:}gda.\ \{f'\ x\}$
have *Heq*:$H = \{(x,y).\ x \in gda \wedge y \in topspace\ Y \wedge y = f'\ x\}$
using *g-delta-of-subset*[*OF h*(*1*)] *h*(*4*) **by**(*auto simp*: *continuous-map-def H-def*)
define *K* **where** *Keq*:$K = \{(x,y).\ x \in topspace\ X \wedge y \in gdb \wedge x = g'\ y\}$
define *A'* **where** $A' \equiv fst\ `\ (H \cap K)$
define *B'* **where** $B' \equiv snd\ `\ (H \cap K)$
have *A'eq*: $A' = \{x \in gda.\ (x, f'\ x) \in K\}$
using *h*(*4*)
by (*auto simp*: *A'-def Keq Heq image-def continuous-map-def Pi-def*)
(*metis* (*mono-tags*, *lifting*) *IntI case-prod-conv fst-conv mem-Collect-eq*)
have *B'eq*: $B' = \{y \in gdb.\ (g'\ y,\ y) \in H\}$
using *h*(*8*)
by (*auto simp*: *B'-def Keq Heq image-def continuous-map-def Pi-def*)
(*metis* (*mono-tags*, *lifting*) *IntI case-prod-conv snd-conv mem-Collect-eq*)
have *A'-gd*: *g-delta-of X A'*
proof −
have *K-gd*:*g-delta-of* (*prod-topology X Y*) *K*
proof −
have *closedin* (*subtopology* (*prod-topology X Y*) (*topspace X* × *gdb*)) *K*
proof −
have $K = ((\lambda y.\ (g'\ y,\ y))\ `\ topspace\ (subtopology\ Y\ gdb))$
using *h*(*8*) *g-delta-of-subset*[*OF h*(*5*)] **by**(*auto simp add*: *Keq continuous-map-def*)
thus *?thesis*
using *cmx.Hausdorff continuous-map-imp-closed-graph'*[*OF h*(*8*)]
by(*auto simp*: *prod-topology-subtopology*(*2*))
qed
then obtain *T* **where** *hT*:*closedin* (*prod-topology X Y*) *T K* = *T* ∩ (*topspace X* × *gdb*)
using *closedin-subtopology* **by** *metis*
thus *?thesis*
by(*auto intro*!: *g-delta-of-inter g-delta-of-prod simp*: *h*(*5*) *mxy.g-delta-of-closedin*)
qed
have $A' = ((\lambda x.\ (x,f'\ x)) -`\ K\ \cap topspace\ (subtopology\ X\ gda))$
by(*auto simp add*: *A'eq Keq*)
also have *g-delta-of X* ...
by(*rule g-delta-of-subtopology-inverse*[*OF g-delta-of-continuous-map*[*OF - K-gd*] *h*(*1*)]) (*auto intro*!: *continuous-map-pairedI h*(*4*))
finally show *?thesis* **.**
qed

have *A-subst-A'*: $A \subseteq A'$
proof
fix a
assume *0*:$a \in A$
then have $f'\ a = f\ a\ f'\ a \in B$
using *h(2)*[*OF 0*,*symmetric*] *hfg(1) assms(3)* **by**(*auto simp*: *continuous-map-def*)
thus $a \in A'$
using *h(6)*[*OF* ‹$f'\ a \in B$›,*symmetric*] *hfg(3)*[*OF 0*] *0 assms(3) h(3) h(7)*
by(*auto simp*: *A'eq Keq*)
qed
have *B'-gd*: *g-delta-of Y B'*
proof −
have *H-gd*:*g-delta-of* (*prod-topology X Y*) *H*
proof −
have *closedin* (*subtopology* (*prod-topology X Y*) (*gda* × *topspace Y*)) *H*
proof −
have $H = ((\lambda y.\ (y, f'\ y))\ `\ topspace\ (subtopology\ X\ gda))$
using *h(4) g-delta-of-subset*[*OF h(1)*] **by**(*auto simp add*: *Heq continuous-map-def*)
thus *?thesis*
using *cmy.Hausdorff continuous-map-imp-closed-graph*[*OF h(4)*]
by(*auto simp*: *prod-topology-subtopology(1)*)
qed
then obtain *T* **where** *hT*:*closedin* (*prod-topology X Y*) *T H* = *T* ∩ (*gda* × *topspace Y*)
using *closedin-subtopology* **by** *metis*
thus *?thesis*
by(*auto intro*!: *g-delta-of-inter g-delta-of-prod simp*: *h(1) mxy.g-delta-of-closedin*)
qed
have $B' = ((\lambda x.\ (g'\ x,x)) -`\ H\ \cap\ topspace\ (subtopology\ Y\ gdb))$
by(*auto simp add*: *B'eq Heq*)
also have *g-delta-of Y* ...
by(*rule g-delta-of-subtopology-inverse*[*OF g-delta-of-continuous-map*[*OF - H-gd*] *h(5)*]) (*auto intro*!: *continuous-map-pairedI h(8)*)
finally show *?thesis* **.**
qed
have *B-subst-B'*: $B \subseteq B'$
proof
fix b
assume *0*:$b \in B$
then have $g'\ b = g\ b\ g'\ b \in A$
using *h(6)*[*OF 0*,*symmetric*] *hfg(2) assms(4)* **by**(*auto simp*: *continuous-map-def*)
thus $b \in B'$
using *h(2)*[*OF* ‹$g'\ b \in A$›,*symmetric*] *hfg(4)*[*OF 0*] *0 assms(4) h(3) h(7)*
by(*auto simp*: *B'eq Heq*)
qed
have *homeomorphic-map* (*subtopology X A'*) (*subtopology Y B'*) *f'*
proof(*rule homeomorphic-maps-imp-map*[**where** *g*=*g'*])
show *homeomorphic-maps* (*subtopology X A'*) (*subtopology Y B'*)

f′ g′
unfolding *homeomorphic-maps-def*
proof *safe*
show *continuous-map* (*subtopology X A′*) (*subtopology Y B′*) *f′*
using *g-delta-of-subset*[*OF h*(*5*)]
by(*auto intro*!: *continuous-map-into-subtopology continuous-map-from-subtopology-mono*[*OF h*(*4*)] *simp*: *A′eq B′eq Heq Keq*)
next
show *continuous-map* (*subtopology Y B′*) (*subtopology X A′*) *g′*
using *g-delta-of-subset*[*OF h*(*1*)]
by(*auto intro*!: *continuous-map-into-subtopology continuous-map-from-subtopology-mono*[*OF h*(*8*)] *simp*: *A′eq B′eq Heq Keq*)
qed(*auto simp*: *A′eq B′eq Keq Heq*)
qed

with *A′-gd B′-gd A-subst-A′ B-subst-B′ h*(*2*)
show *?thesis* **by** *auto*
qed

corollary(**in** *complete-metrizable*) *complete-metrizable-subtopology-is-g-delta*:
assumes *A* ⊆ *topspace S complete-metrizable* (*subtopology S A*)
shows *g-delta-of S A*
proof −
obtain *h gda gdb* **where** *h*:
g-delta-of S gda g-delta-of (*subtopology S A*) *gdb A* ⊆ *gda A* ⊆ *gdb* ∀ *x*∈*A*. *x* = *h x homeomorphic-map* (*subtopology* (*subtopology S A*) *gdb*) (*subtopology S gda*) *h*
using *Lavrentiev-theorem*[*OF assms*(*2*) *complete-metrizable-axioms* - *assms*(*1*),*of A id*] *assms*(*1*)
by *simp* (*metis subtopology-topspace topspace-subtopology-subset*)
have *gdb* = *A*
using *g-delta-of-subset*[*OF h*(*2*)] *h*(*4*) *assms*(*1*) **by** *auto*
hence *homeomorphic-map* (*subtopology S A*) (*subtopology S gda*) *h*
using *h*(*6*) **by** (*simp add*: *subtopology-subtopology*)
hence *homeomorphic-map* (*subtopology S A*) (*subtopology S gda*) *id*
by(*rule homeomorphic-map-eq*) (*use assms*(*1*) *h*(*5*) **in** *auto*)
hence *subtopology S A* = *subtopology S gda* **by** *simp*
hence *A* = *gda*
by (*metis assms*(*1*) *g-delta-of-subset h*(*1*) *topspace-subtopology-subset*)
thus *?thesis*
by(*simp add*: *h*(*1*))
qed

corollary(**in** *complete-metrizable*) *subtopology-complete-metrizable-iff*:
assumes *A* ⊆ *topspace S*
shows *complete-metrizable* (*subtopology S A*) ⟷ *g-delta-of S A*
by(*auto simp* : *g-delta-of-complete-metrizable complete-metrizable-subtopology-is-g-delta*[*OF assms*])

corollary *complete-metrizable-homeo-image-g-delta*:

assumes *complete-metrizable X complete-metrizable Y B* ⊆ *topspace Y X homeomorphic-space subtopology Y B*
shows *g-delta-of Y B*
proof −
obtain *f* **where** *f*:*homeomorphic-map X* (*subtopology Y B*) *f*
using *assms*(*4*) *homeomorphic-space* **by** *blast*
obtain *h gda gdb* **where** *h*:
g-delta-of X gda g-delta-of Y gdb topspace X ⊆ *gda B* ⊆ *gdb* ∀ *x*∈*topspace X*. *f x* = *h x homeomorphic-map* (*subtopology X gda*) (*subtopology Y gdb*) *h*
using *Lavrentiev-theorem*[*OF assms*(*1*,*2*) *subset-refl assms*(*3*),*simplified*,*OF f*]
by *metis*
hence [*simp*]: *gda* = *topspace X*
using *g-delta-of-subset* **by** *blast*
have *homeomorphic-map X* (*subtopology Y gdb*) *f*
using *h*(*5*,*6*) **by**(*auto intro*!: *homeomorphic-map-eq*[**where** *f*=*h*])
hence *f* ‘ *topspace X* = *B f* ‘ *topspace X* = *gdb*
using *homeomorphic-imp-surjective-map*[*OF f*] *assms*(*3*) *g-delta-of-subset*[*OF h*(*2*)] *h*(*4*) *homeomorphic-imp-surjective-map*[*OF* ‹*homeomorphic-map X* (*subtopology Y gdb*) *f*›]
by *auto*
with *h*(*2*) **show** *?thesis* **by** *auto*
qed

lemma(**in** *metrizable*) *embedding-into-Hilbert-cube*:
assumes *separable S*
shows ∃ *A* ⊆ *topspace Hilbert-cube-as-topology*. *S homeomorphic-space* (*subtopology Hilbert-cube-as-topology A*)
proof −
consider *topspace S* = {} | *topspace S* ≠ {} **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-of S U U* ≠ {}
using *assms*(*1*) **by**(*auto simp*: *separable-def dense-of-nonempty*)
obtain *xn* **where** *xn*:⋀*n*::*nat*. *xn n* ∈ *U U* = *range xn*
by (*metis U*(*1*) *U*(*3*) *from-nat-into range-from-nat-into*)
then have *xns*:*xn n* ∈ *topspace S* **for** *n*
using *dense-of-subset*[*OF U*(*2*)] **by** *auto*
obtain *d* **where** *d*:*metric-set* (*topspace S*) *d metric-set.mtopology* (*topspace S*) *d* = *S* ⋀*x y*. *d x y* < *1*
using *bounded-metric* **by** *auto*
interpret *ms*: *metric-set topspace S d* **by** *fact*
define *f* **where** *f* ≡ (λ*x n*. *d x* (*xn n*))
have *f-inj*:*inj-on f* (*topspace S*)
proof

fix x y
assume xy:$x \in$ *topspace* S $y \in$ *topspace* S $f\ x = f\ y$
then have $\bigwedge n.\ d\ x\ (xn\ n) = d\ y\ (xn\ n)$ **by**(*auto simp*: *f-def dest*: *fun-cong*)
hence *d2*:$d\ x\ y \leq 2 * d\ x\ (xn\ n)$ **for** n
using *ms.dist-tr*[*OF xy*(*1*) - *xy*(*2*),*of xn n*,*simplified ms.dist-sym*[*of xn n y*]] *dense-of-subset*[*OF U*(*2*)] *xn*(*1*)[*of n*]
by *auto*
have $d\ x\ y < \varepsilon$ **if** $\varepsilon > 0$ **for** ε
proof −
have $0 < \varepsilon\ /\ 2$ **using** *that* **by** *simp*
then obtain n **where** $d\ x\ (xn\ n) < \varepsilon\ /\ 2$
using *ms.dense-set-def2*[*of U*,*simplified d*(*2*)] *U*(*2*) *xy*(*1*) *xn*(*2*) **by** *blast*
with *d2*[*of n*] **show** *?thesis* **by** *simp*
qed
hence $d\ x\ y = 0$
using *ms.dist-geq0*[*of x y*]
by (*metis dual-order.irrefl order-neq-le-trans*)
thus $x = y$
using *ms.dist-0*[*OF xy*(*1*,*2*)] **by** *simp*
qed
have *f-img*: f ‘ *topspace* $S \subseteq$ *topspace Hilbert-cube-as-topology*
using *d*(*3*) *ms.dist-geq0* **by**(*auto simp*: *topspace-Hilbert-cube f-def less-le-not-le*)
have *f-cont*: *continuous-map S Hilbert-cube-as-topology f*
unfolding *continuous-map-componentwise-UNIV f-def continuous-map-in-subtopology*
proof *safe*
show *continuous-map S euclideanreal* ($\lambda x.\ d\ x\ (xn\ k)$) **for** k
using *ms.dist-set-continuous*[*of* {*xn k*}] **by**(*simp add*: *d*(*2*))
next
show $d\ x\ (xn\ k) \in \{0..1\}$ **for** $x\ k$
using *d*(*3*) *ms.dist-geq0* **by**(*auto simp*: *less-le-not-le*)
qed
hence *f-cont′*: *continuous-map S* (*subtopology Hilbert-cube-as-topology* (*f* ‘ *topspace S*)) *f*
using *continuous-map-into-subtopology* **by** *blast*
obtain *g* **where** *g*: *g* ‘ (*f* ‘ *topspace S*) = *topspace S* $\bigwedge x.\ x \in$ *topspace* $S \Longrightarrow g\ (f\ x) = x$ $\bigwedge x.\ x \in f$ ‘ *topspace* $S \Longrightarrow f\ (g\ x) = x$
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-as-topology* (*f* ‘ *topspace S*)) *S g*
proof −
interpret *m01*: *polish-metric-set* {*0..1*::*real*} *submetric* {*0..1*} *dist*
by (*metis closed-atLeastAtMost closed-closedin euclidean-mtopology polish-class-polish-set polish-metric-set.submetric-polish subset-UNIV*)
have *m01-eq*: *m01.mtopology* = *top-of-set* {*0..1*}
by(*rule submetric-of-euclidean*(*2*)[*of* {*0..1*::*real*}])
have *submetric* {*0..1*::*real*} *dist x y* ≤ 1 *submetric* {*0..1*::*real*} *dist x y* ≥ 0 **for** $x\ y$
using *dist-real-def* **by**(*auto simp*: *submetric-def*)
then interpret *ppm*: *product-polish-metric 1/2 UNIV* :: *nat set id id* λ-.

$\{0..1\}$ λ-. *submetric* $\{0..1$::*real*$\}$ *dist 1*
by(*auto intro*!: *product-polish-metric-natI m01.polish-metric-set-axioms*)
have *Hilbert-cube-eq*: *ppm.mtopology* = *Hilbert-cube-as-topology*
by(*simp add*: *ppm.product-dist-mtopology*[*symmetric*] *m01-eq*)
interpret *f-S*: *metric-set f* ‘ *topspace S submetric* (*f* ‘ *topspace S*) *ppm.product-dist*
using *f-img* **by**(*auto intro*!: *ppm.submetric-metric-set*)
have *1*:*subtopology Hilbert-cube-as-topology* (*f* ‘ *topspace S*) = *f-S.mtopology*
using *ppm.submetric-subtopology*[*of f* ‘ *topspace S*] *f-img* **by**(*simp add*: *Hilbert-cube-eq*)
have *continuous-map f-S.mtopology ms.mtopology g*
unfolding *metric-set-continuous-map-eq'*[*OF f-S.metric-set-axioms ms.metric-set-axioms*]
proof *safe*
show *x* ∈ *topspace S* ⟹ *g* (*f x*) ∈ *topspace S* **for** *x*
by(*simp add*: *g*(*2*))
next
fix *yn y*
assume *h*:*f-S.converge-to-inS yn y*
have *ppm.converge-to-inS yn y*
using *ppm.converge-to-insub-converge-to-inS*[*OF - h*] *f-img* **by** *auto*
hence *m01-conv*:⋀*n*. *m01.converge-to-inS* (λ*i*. *yn i n*) (*y n*)
using *ppm.converge-to-iff*[*of yn y*] **by**(*auto simp*: *ppm.converge-to-inS-def*)
have ⋀*n*. ∃ *zn*. *yn n* = *f zn* ∧ *zn* ∈ *topspace S*
using *h* **by**(*auto simp*: *f-S.converge-to-inS-def*)
then obtain *zn* **where** *zn*:⋀*n*. *f* (*zn n*) = *yn n* ⋀*n*. *zn n* ∈ *topspace S*
by *metis*
obtain *z* **where** *z*:*f z* = *y z* ∈ *topspace S*
using *h* **by**(*auto simp*: *f-S.converge-to-inS-def*)
show *ms.converge-to-inS* (λ*n*. *g* (*yn n*)) (*g y*)
unfolding *ms.converge-to-inS-def2*
proof *safe*
show *g* (*yn n*) ∈ *topspace S g y* ∈ *topspace S* **for** *n*
using *g*(*2*)[*of z*] *g*(*2*)[*of zn n*] *zn*[*of n*] *z* **by** *simp-all*
next
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.dense-set-def2*[*of U*,*simplified d*(*2*)] *U*(*2*) *z*(*2*) *xn*(*2*) **by** *blast*
obtain *N* **where** ⋀*n*. *n* ≥ *N* ⟹ |*yn n m* − *y m* | < ε / *3*
using *m01-conv*[*of m*,*simplified m01.converge-to-inS-def2*] ‹*0* < ε / *3*›
by(*simp only*: *submetric-def dist-real-def*) (*metis* (*full-types*, *lifting*) *PiE UNIV-I*)
hence *N*:⋀*n*. *n* ≥ *N* ⟹ *yn n m* < ε / *3* + *y m*
by (*metis abs-diff-less-iff add.commute*)
have ∃ *N*. ∀ *n*≥*N*. *d* (*zn n*) *z* < ε
proof(*safe intro*!: *exI*[**where** *x*=*N*])
fix *n*
assume *N* ≤ *n*
have *d* (*zn n*) *z* ≤ *f* (*zn n*) *m* + *d z* (*xn m*)

```
          using ms.dist-tr[OF zn(2)[of n] xns[of m] z(2),simplified ms.dist-sym[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] 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
        thus ∃ N. ∀ n≥N. d (g (yn n)) (g y) < ε
          using zn(1) z(1) g(2)[OF z(2)] g(2)[OF zn(2)] by auto
      qed
    qed
    thus ?thesis
      by(simp add: d(2) 1)
  qed
  show ?thesis
    using f-img g(2,3) f-cont' g-cont
   by(auto intro!: exI[where x=f ‘ topspace S] homeomorphic-maps-imp-homeomorphic-space[where
f=f and g=g] simp: homeomorphic-maps-def)
 qed
qed

corollary(in complete-metrizable) embedding-into-Hilbert-cube-g-delta-of:
 assumes separable S
 shows ∃ A. g-delta-of Hilbert-cube-as-topology A ∧ S homeomorphic-space (subtopology
Hilbert-cube-as-topology A)
proof −
 obtain A where h:A ⊆ topspace Hilbert-cube-as-topology S homeomorphic-space
subtopology Hilbert-cube-as-topology A
   using embedding-into-Hilbert-cube[OF assms(1)] by blast
 with complete-metrizable-homeo-image-g-delta[OF complete-metrizable-axioms pol-
ish-topology.axioms(1)[OF Hilbert-cube-Polish-topology] h(1,2)]
 show ?thesis
   by(auto intro!: exI[where x=A])
qed

corollary(in polish-topology) embedding-into-Hilbert-cube-g-delta-of:
 ∃ A. g-delta-of Hilbert-cube-as-topology A ∧ S homeomorphic-space (subtopology
Hilbert-cube-as-topology A)
 by(rule embedding-into-Hilbert-cube-g-delta-of[OF S-separable])

lemma(in polish-topology) uncountable-contains-Cantor-space':
 assumes uncountable (topspace S)
 shows ∃ A⊆ topspace S. Cantor-space-as-topology homeomorphic-space (subtopology
S A)
proof −
```

obtain U P **where** *up*: *countable U openin S U perfect-set S P* $U \cup P = topspace\ S\ U \cap P = \{\} \bigwedge a.\ a \neq \{\} \Longrightarrow openin\ (subtopology\ S\ P)\ a \Longrightarrow uncountable\ a$
using *Cantor-Bendixon*[*OF S-second-countable*] **by** *auto*
have *P*: *closedin S P P* $\subseteq$ *topspace S uncountable P*
using *countable-Un-iff*[*of U P*] *up*(*1*) *assms up*(*4*)
by(*simp-all add*: *perfect-setD*[*OF up*(*3*)])
then interpret *pp*: *polish-topology subtopology S P*
by(*simp add*: *closedin-polish*)
have *Ptop*: *topspace* (*subtopology S P*) = *P*
using *P*(*2*) **by** *auto*
obtain *U* **where** *U*: *countable U dense-of* (*subtopology S P*) *U*
using *pp.S-separable separable-def* **by** *blast*
with *uncountable-infinite*[*OF P*(*3*)] *pp.dense-of-infinite P*(*2*)
have *infinite U* **by** (*metis topspace-subtopology-subset*)
obtain *d* **where** *complete-metric-set P d* **and** *d*:*metric-set.mtopology P d* = *subtopology S P*
using *pp.cmetric* **by**(*simp only*: *Ptop*,*auto*)
interpret *md*: *complete-metric-set P d* **by** *fact*
define *xn* **where** *xn* $\equiv$ *from-nat-into U*
have *xn*: *bij-betw xn UNIV U* $\bigwedge n\ m.\ n \neq m \Longrightarrow xn\ n \neq xn\ m\ \bigwedge n.\ xn\ n \in U\ \bigwedge n.\ xn\ n \in P\ md.dense\text{-}set\ (range\ xn)$
using *bij-betw-from-nat-into*[*OF U*(*1*) ‹*infinite U*›] *dense-of-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 [*simp*]:*topspace md.mtopology* = *P*
using *Ptop* **by**(*simp add*: *md.mtopology-topspace*)
have *perfect*:*perfect-space md.mtopology*
using *d perfect-set-subtopology up*(*3*) **by** *simp*
define *jn* **where** *jn* $\equiv$ ($\lambda n.$ *LEAST i. i* > *n* $\wedge$ *md.closed-ball* (*xn i*) ((*1*/*2*)^*i*) $\subseteq$ *md.open-ball* (*xn n*) ((*1*/*2*)^*n*) − *md.open-ball* (*xn n*) ((*1*/*2*)^*i*))
define *kn* **where** *kn* $\equiv$ ($\lambda n.$ *LEAST k. k* > *jn n* $\wedge$ *md.closed-ball* (*xn k*) ((*1*/*2*)^*k*) $\subseteq$ *md.open-ball* (*xn n*) ((*1*/*2*)^*jn n*))
have *dxmxn*: $\forall\ n\ n'.\ \exists\ m.\ m > n \wedge m > n' \wedge (1/2)\hat{}(m-1) < d\ (xn\ n)\ (xn\ m) \wedge d\ (xn\ n)\ (xn\ m) < (1/2)\hat{}(Suc\ n')$
proof *safe*
fix *n n*′
have *hinfin*′:*infinite* (*md.open-ball x e* $\cap$ (*range xn*)) **if** *x* $\in$ *P e* > *0* **for** *x e*
proof
assume *h-fin*:*finite* (*md.open-ball x e* $\cap$ *range xn*)
have *h-nen*:*md.open-ball x e* $\cap$ *range xn* $\neq$ {}
using *xn*(*5*) *that* **by**(*auto simp*: *md.dense-set-def*)
have *infin*: *infinite* (*md.open-ball x e*)
using *md.perfect-set-open-ball-infinite*[*OF perfect*] *that* **by** *simp*
then obtain *y* **where** *y*:*y* $\in$ *md.open-ball x e y* $\notin$ *range xn*
using *h-fin* **by**(*metis inf.absorb-iff2 inf-commute subsetI*)
define *e*′ **where** *e*′ = *Min* {*d y xk* |*xk*. *xk* $\in$ *md.open-ball x e* $\cap$ *range xn*}
have *fin*: *finite* {*d y xk* |*xk*. *xk* $\in$ *md.open-ball x e* $\cap$ *range xn*}
using *finite-imageI*[*OF h-fin*,*of d y*] **by** (*metis Setcompr-eq-image*)

have *nen*: $\{d\ y\ xk\ |xk.\ xk \in md.open\text{-}ball\ x\ e \cap range\ xn\} \neq \{\}$
using *h-nen* **by** *auto*
have $e' > 0$
unfolding *e'-def Min-gr-iff*[*OF fin nen*]
proof *safe*
fix *l*
assume $xn\ l \in md.open\text{-}ball\ x\ e$
from $y(2)$ *md.dist-0*[*OF md.open-ballD'*(1)[*OF* $y(1)$] *md.open-ballD'*(1)[*OF this*]] *md.dist-geq0*[*of y xn l*]
show $0 < d\ y\ (xn\ l)$
by *auto*
qed
obtain e'' **where** e'': $e'' > 0$ $md.open\text{-}ball\ y\ e'' \subseteq md.open\text{-}ball\ x\ e\ y \in md.open\text{-}ball\ y\ e''$
by (*meson md.mtopology-open-ball-in' md.open-ballD'*(1) *md.open-ball-ina* $y(1)$)
define ε **where** $\varepsilon \equiv min\ e'\ e''$
have $\varepsilon > 0$
using $e''(1)$ ‹$e' > 0$› **by**(*simp add*: ε-*def*)
then obtain *m* **where** *m*: $d\ y\ (xn\ m) < \varepsilon$
using *md.dense-set-def2*[*of range xn*] *xn*(5) *md.open-ballD'*(1)[*OF* $y(1)$] **by** *blast*
consider $xn\ m \in md.open\text{-}ball\ x\ e \mid xn\ m \in P - md.open\text{-}ball\ x\ e$
using *xn*(4) **by** *auto*
then show *False*
proof *cases*
case *1*
then have $e' \leq d\ y\ (xn\ m)$
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 $xn\ m \notin md.open\text{-}ball\ y\ e''$
using $e''(2)$ **by** *auto*
hence $e'' \leq d\ y\ (xn\ m)$
by(*rule md.open-ball-nin-le*[*OF md.open-ballD'*(1)[*OF* $y(1)$] $e''(1)$ *xn*(4)[*of m*]])
thus *?thesis*
using *m* **by**(*simp add*: ε-*def*)
qed
qed
have *hinfin*:*infinite* $(md.open\text{-}ball\ x\ e \cap (xn\ `\ \{l<..\}))$ **if** $x \in P\ e > 0$ **for** $x\ e\ l$
proof
assume *finite* $(md.open\text{-}ball\ x\ e \cap xn\ `\ \{l<..\})$
moreover have *finite* $(md.open\text{-}ball\ x\ e \cap xn\ `\ \{..l\})$ **by** *simp*
moreover have $(md.open\text{-}ball\ x\ e \cap (range\ xn)) = (md.open\text{-}ball\ x\ e \cap xn\ `\ \{l<..\}) \cup (md.open\text{-}ball\ x\ e \cap xn\ `\ \{..l\})$
by *fastforce*

ultimately have *finite* (*md.open-ball x e* ∩ (*range xn*))
by *auto*
with *hinfin'*[*OF that*] **show** *False* **..**
qed
have *infinite* (*md.open-ball* (*xn n*) ((*1/2*)^*Suc n'*))
using *md.perfect-set-open-ball-infinite*[*OF perfect*] *xn*(*4*)[*of n*] **by** *simp*
then obtain *x* **where** *x*: *x* ∈ *md.open-ball* (*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.open-ball x e* ⊆ *md.open-ball* (*xn n*) ((*1/2*)^*Suc n'*) *x* ∈ *md.open-ball x e*
by (*meson md.mtopology-open-ball-in' md.open-ballD'*(*1*) *md.open-ball-ina*)
have *d* (*xn n*) *x* > *0*
using *md.dist-geq0*[*of xn n x*] *md.dist-0*[*OF xn*(*4*)[*of n*] *md.open-ballD'*(*1*)[*OF x*(*1*)]] *x*(*2*) **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.open-ball x* ε ⊆ *md.open-ball* (*xn n*) ((*1 / 2*) ^ *Suc n'*)
using *md.open-ball-le*[*of* ε *e x*] *e*(*2*) **by**(*simp add*: ε-*def*)
obtain *k* **where** *k*: *xn k* ∈ *md.open-ball x* ε *k* > *m*
using *infinite-imp-nonempty*[*OF hinfin*[*OF md.open-ballD'*(*1*)[*OF x*(*1*)] ‹ε > *0*›,*of m*]] **by** *auto*
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 *md.open-ballD*[*OF k*(*1*)] **by**(*simp add*: ε-*def md.dist-sym*[*of x* -])
also have ... ≤ *d* (*xn n*) (*xn k*)
using *md.dist-tr*[*OF xn*(*4*)[*of n*] *xn*(*4*)[*of k*] *md.open-ballD'*(*1*)[*OF x*(*1*)]] **by** *simp*
finally show (*1 / 2*) ^ (*k* − *1*) < *d* (*xn n*) (*xn k*) **.**
qed(*use* ‹*m* ≥ *n'*› *k ball-le md.open-ballD*[*of xn k xn n* (*1 / 2*) ^ *Suc n'*] *m*(*3*) **in** *auto*)
qed
have *jn n* > *n* ∧ *md.closed-ball* (*xn* (*jn n*)) ((*1/2*)^(*jn n*)) ⊆ *md.open-ball* (*xn n*) ((*1/2*)^*n*) − *md.open-ball* (*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.closed-ball (xn x) ((1 / 2) ^ x) ⊆ md.open-ball (xn n) ((1 /
2) ^ n) − md.open-ball (xn n) ((1 / 2) ^ x)
    proof(safe intro!: exI[where x=m] m(1))
      fix x
      assume h:x ∈ md.closed-ball (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.dist-tr[OF xn(4)[of n] xn(4)[of m] md.closed-ballD'(1)[OF
h]] md.closed-ballD[OF h]
          by simp
        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.dist-tr[OF xn(4)[of n] md.closed-ballD'(1)[OF
h] xn(4)[of m]]] md.closed-ballD(1)[OF h]
          by(simp add: md.dist-sym)
        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.open-ball (xn n) ((1 / 2) ^ n)
           x ∈ md.open-ball (xn n) ((1 / 2) ^ m) ⟹ False
      using xn(4)[of n] md.closed-ballD'(1)[OF h] 1 2 by(auto simp: md.open-ball-def)
    qed
  qed
   hence jn: ⋀n. jn n > n ⋀n. md.closed-ball (xn (jn n)) ((1/2)^(jn n)) ⊆
md.open-ball (xn n) ((1/2)^n) − md.open-ball (xn n) ((1/2)^(jn n))
    by simp-all
  have kn n > jn n ∧ md.closed-ball (xn (kn n)) ((1/2)^(kn n)) ⊆ md.open-ball
(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.closed-ball (xn x) ((1 / 2) ^ x) ⊆ md.open-ball (xn n) ((1
/ 2) ^ jn n)
    proof(intro exI[where x=m] conjI)
      show md.closed-ball (xn m) ((1 / 2) ^ m) ⊆ md.open-ball (xn n) ((1 / 2) ^
```

```
jn n)
      proof
        fix x
        assume h:x ∈ md.closed-ball (xn m) ((1 / 2) ^ m)
        have d (xn n) x < (1 / 2)^ Suc (jn n) + (1 / 2) ^ m
          using md.dist-tr[OF xn(4)[of n] xn(4)[of m] md.closed-ballD'(1)[OF h]]
m(2) md.closed-ballD[OF h]
          by simp
        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.open-ball (xn n) ((1 / 2) ^ jn n)
          by(simp add: xn(4)[of n] md.closed-ballD'(1)[OF h] md.open-ball-def)
      qed
    qed(use m(1) in auto)
  qed
  hence kn: ⋀n. kn n > jn n ⋀n. md.closed-ball (xn (kn n)) ((1/2)^(kn n)) ⊆
md.open-ball (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 (λa 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 ≡ (λx . rec-nat [] (λn 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 ≡ (λn. Min (range (λx. (1 / 2 :: real) ^ bn (to-listn x n))))
  have rn-fin: finite (range (λx. (1 / 2 :: real) ^ bn (to-listn x n))) for n
  proof −
    have finite (range (λx. bn (to-listn x n)))
    proof(induction n)
      case ih:(Suc n)
       have (range (λx. bn (to-listn x (Suc n)))) ⊆ (range (λx. jn (bn (to-listn x
n)))) ∪ (range (λx. 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 (λx. (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.closed-ball (xn (bn (a#l))) ((1/2)^(bn (a#l))) ⊆ md.open-ball
(xn (bn l)) ((1/2) ^ (bn l)) for a l
  using kn(2)[of bn l] md.open-ball-le[of (1 / 2) ^ jn (bn l) (1 / 2) ^ bn l xn
(bn l)] less-imp-le [OF jn(1)] jn(2)
  by(simp add: bn-simp) blast
hence cball-le:md.closed-ball (xn (bn (a#l))) ((1/2)^(bn (a#l))) ⊆ md.closed-ball
(xn (bn l)) ((1/2) ^ (bn l)) for a l
  using md.open-ball-closed-ball by blast
have cball-disj: md.closed-ball (xn (bn (0#l))) ((1/2)^(bn (0#l))) ∩ md.closed-ball
(xn (bn (1#l))) ((1/2)^(bn (1#l))) = {} for l
  using jn(2) kn(2) by(auto simp: bn-simp)
 have ∀x. ∃xa∈P. (⋂n. md.closed-ball (xn (bn (to-listn x n))) ((1 / 2) ^ bn
(to-listn x n))) = {xa}
proof
  fix x
 show ∃xa∈P. (⋂n. md.closed-ball (xn (bn (to-listn x n))) ((1 / 2) ^ bn (to-listn
x n))) = {xa}
  proof(rule md.closed-decseq-Inter)
    show md.closed-ball (xn (bn (to-listn x n))) ((1 / 2) ^ bn (to-listn x n)) ≠
{} for n
      using md.closed-ball-ina[OF xn(4)[of bn (to-listn x n)],of (1 / 2) ^ bn
(to-listn x n)] by auto
  next
   show decseq (λn. md.closed-ball (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 ε :: real
    assume 0 < ε
    then obtain N where N: (1 / 2) ^ N < (1/2) * ε
    by (metis divide-pos-pos mult.commute mult.right-neutral one-less-numeral-iff
reals-power-lt-ex semiring-norm(76) times-divide-eq-right zero-less-numeral)
    show ∃N. ∀n≥N. md.diam (md.closed-ball (xn (bn (to-listn x n))) ((1 / 2)
^ bn (to-listn x n))) < ε
    proof(safe intro!: exI[where x=N])
      fix n
      assume N ≤ n
```

have *md.diam* (*md.closed-ball* (*xn* (*bn* (*to-listn x n*))) ((*1 / 2*) ^ *bn* (*to-listn x n*))) ≤ *md.diam* (*md.closed-ball* (*xn* (*bn* (*to-listn x n*))) ((*1 / 2*) ^ *n*))
using *bn-gtn*[*of n x*] **by**(*auto intro*!: *md.diam-subset md.closed-ball-le*)
also have ... ≤ *ennreal* (*2* ∗ (*1 / 2*) ^ *n*)
by(*simp add*: *md.diam-cball-leq*)
also have ... ≤ *ennreal* (*2* ∗ (*1 / 2*) ^ *N*)
using ‹*N* ≤ *n*› **by** (*simp add*: *numeral-mult-ennreal*)
also have ... < *ennreal* (*2* ∗(*1*/*2*) ∗ *ε*)
using *N* **by** (*simp add*: ‹*0* < *ε*› *ennreal-lessI le-less numeral-mult-ennreal*)
also have ... = *ennreal ε*
by (*simp add*: ‹*0* < *ε*› *le-less numeral-mult-ennreal*)
finally show *md.diam* (*md.closed-ball* (*xn* (*bn* (*to-listn x n*))) ((*1 / 2*) ^ *bn* (*to-listn x n*))) < *ennreal ε* **.**
qed
qed(*rule md.closedin-closed-ball*)
qed
then obtain *f* **where** *f*:⋀*x*. *f x* ∈ *P* ⋀*x*. (⋂ *n*. *md.closed-ball* (*xn* (*bn* (*to-listn x n*))) ((*1 / 2*) ^ *bn* (*to-listn x n*))) = {*f x*}
by *metis*
hence *f'*: ⋀*n x*. *f x* ∈ *md.closed-ball* (*xn* (*bn* (*to-listn x n*))) ((*1 / 2*) ^ *bn* (*to-listn x n*))
by *blast*
have *f''*: *f x* ∈ *md.open-ball* (*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**(*auto simp*: *to-listn-simp*)
from *f* **interpret** *bdmd*: *metric-set f* ‘ (Π$_E$ *i*∈*UNIV*. {*0*,*1*}) *submetric* (*f* ‘ (Π$_E$ *i*∈*UNIV*. {*0*,*1*})) *d*
by(*auto intro*!: *md.submetric-metric-set*)
have *bdmd-top*: *bdmd.mtopology* = *subtopology md.mtopology* (*f* ‘ (Π$_E$ *i*∈*UNIV*. {*0*,*1*}))
by (*simp add*: *f*(*1*) *image-subset-iff md.submetric-subtopology*)
have *bdmd-sub*: *bdmd.mtopology* = *subtopology S* (*f* ‘ (Π$_E$ *i*∈*UNIV*. {*0*,*1*}))
using *f*(*1*) *Int-absorb1*[*of f* ‘ (*UNIV* →$_E$ {*0*, *1*}) *P*] **by**(*fastforce simp*: *bdmd-top d subtopology-subtopology*)
interpret *d01*: *polish-metric-set* {*0*,*1*::*real*} *submetric* {*0*,*1*::*real*} *dist*
by(*auto intro*!: *polish-metric-set.submetric-polish*[*OF polish-class-polish-set*] *simp*: *euclidean-mtopology*)
interpret *pd*: *product-polish-metric 1*/*2 UNIV id id* λ-. {*0*,*1*::*real*} λ-. *submetric* {*0*,*1*::*real*} *dist 1*
by(*auto intro*!: *product-polish-metric-natI simp*: *d01.polish-metric-set-axioms*) (*auto simp*: *submetric-def*)
have *mpd-top*: *pd.mtopology* = *Cantor-space-as-topology*
by(*auto simp*: *pd.product-dist-mtopology*[*symmetric*] *submetric-of-euclidean*(*2*) *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 ⋀n. n < def-at x y ⟹ x n = y n x (def-at x y) ≠ y (def-at x y)
      using not-less-Least by(auto simp: def-at-def)
  qed
  have def-at-le-if: pd.product-dist x y ≤ (1/2)^n ⟹ n ≤ def-at x y if assm:x ≠
y x ∈ (Π_E i∈UNIV. {0,1}) y ∈ (Π_E i∈UNIV. {0,1}) for x y n
  proof −
    assume h:pd.product-dist x y ≤ (1 / 2) ^ n
    have x m = y m if m-less-n: m < n for m
    proof(rule ccontr)
      assume nen: x m ≠ y m
      then have submetric {0, 1} dist (x m) (y m) = 1
        using assm(2,3) by(auto simp: submetric-def)
      hence 1 ≤ 2 ^ m * pd.product-dist x y
        using pd.product-dist-geq[of m m,simplified,OF assm(2,3)] by simp
      hence (1/2)^m ≤ 2^m * (1/2)^m * pd.product-dist x y by simp
      hence (1/2)^m ≤ pd.product-dist x y by (simp add: power-one-over)
      also have ... ≤ (1 / 2) ^ n
        by(simp add: h)
      finally show False
        using that by auto
    qed
    thus n ≤ def-at x y
      by (meson def-atxy(2) linorder-not-le that(1))
  qed
  have def-at-le-then: pd.product-dist x y ≤ 2 * (1/2)^n if assm:x ≠ y x ∈ (Π_E
i∈UNIV. {0,1}) y ∈ (Π_E i∈UNIV. {0,1}) n ≤ def-at x y for x y n
  proof −
    have ⋀m. m < n ⟹ x m = y m
      by (metis def-atxy(1) order-less-le-trans that(4))
    hence 1:⋀m. m < n ⟹ submetric {0, 1} dist (x m) (y m) = 0
      by (simp add: submetric-def)
    have pd.product-dist x y = (∑ i. (1/2)^(i + n) * (submetric {0, 1} dist (x
(i + n)) (y (i + n)))) + (∑ i<n. (1/2)^i * (submetric {0, 1} dist (x i) (y i)))
    using assm pd.product-dist-summable'[simplified] by(auto intro!: suminf-split-initial-segment
simp: product-dist-def)
    also have ... = (∑ i. (1/2)^(i + n) * (submetric {0, 1} dist (x (i + n)) (y
(i + n))))
      by(simp add: 1)
    also have ... ≤ (∑ i. (1/2)^(i + n))
      using pd.product-dist-summable'[simplified] pd.d-bound by(auto intro!: sum-
inf-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) ≤ (1/2)^(def-at x y) if assm:x ≠ y x ∈ (Π_E i∈UNIV.
{0,1}) y ∈ (Π_E i∈UNIV. {0,1}) for x y
```

```
  proof −
    have 1:to-listn x n = to-listn y n if n ≤ def-at x y for n
    proof −
      have ⋀m. m < n ⟹ 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.closed-ball (xn (bn (to-listn x (def-at x y)))) ((1 / 2) ^ bn
(to-listn x (def-at x y)))
         f y ∈ md.closed-ball (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.diam-is-sup'[OF - - md.diam-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.closed-ball (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.closed-ball (xn (bn (to-listn x m))) ((1 / 2) ^ bn (to-listn x m))
      using ih(2) cball-le by(auto 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.closed-ball (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.closed-ball (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 md.open-ballD[OF 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.closed-ball (xn (bn (to-listn x n))) ((1/2)^bn (to-listn x
n))
     using md.dist-tr[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.closed-ball-def)
   with fy-in[OF assm h(1)] have ∀ m < n. x m = y m
    by simp
   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 S
   using f(1) P(2) by auto
  have 1: continuous-map pd.mtopology bdmd.mtopology f
   unfolding metric-set-continuous-map-eq[OF pd.metric-set-axioms bdmd.metric-set-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∈UNIV →_E {0, 1}. pd.product-dist x y < δ ⟶ submetric (f
‘ (UNIV →_E {0, 1})) 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 submetric (f ‘ (UNIV →_E {0, 1})) d (f x) (f y) < ε
    proof cases
     case 1
```

```
      with y(1) h md.dist-0[OF f(1)[of y] f(1)[of y]]
      show ?thesis by(auto simp add: submetric-def)
    next
      case 2
      then have n ≤ def-at x y
        using h(1) y by(auto intro!: def-at-le-if)
      have submetric (f ‘ (UNIV →E {0, 1})) 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)] submetric-def)
      also have ... ≤ (1/2)^n
        using ‹n ≤ def-at x y› by simp
      finally show ?thesis
        using n by simp
    qed
  qed simp
 qed simp
 have 2: open-map pd.mtopology bdmd.mtopology f
 proof(rule metric-set-opem-map-from-dist[OF pd.metric-set-axioms bdmd.metric-set-axioms,of
f,simplified subtopology-topspace[of bdmd.mtopology,simplified bdmd.mtopology-topspace]])
   fix x :: nat ⇒ real and ε :: real
   assume h:x ∈ (ΠE 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
   obtain e where e: e > 0 ⋀y. y ∈ (ΠE i∈UNIV. {0,1}) ⟹ x ≠ y ⟹ d (f
x) (f y) < e ⟹ Suc n ≤ def-at x y
     using d-le-rn-then[OF h(1),of Suc n] by auto
   show ∃δ>0. ∀y∈UNIV →E {0, 1}. submetric (f ‘ (UNIV →E {0, 1})) d (f
x) (f y) < δ ⟶ pd.product-dist x y < ε
   proof(safe intro!: exI[where x=e])
     fix y
    assume y:y ∈ (ΠE i∈UNIV. {0,1}) and submetric (f ‘ (UNIV →E {0, 1}))
d (f x) (f y) < e
     then have d':d (f x) (f y) < e
       using h(1) by(simp add: submetric-def)
     consider x = y | x ≠ y by auto
     thus pd.product-dist x y < ε
       by cases (use pd.dist-0[OF y y] h(2) def-at-le-then[OF - h(1) y e(2)[OF y
- d']] n in auto)
   qed(use e(1) in auto)
 qed simp
 have 3: f ‘ (topspace pd.mtopology) = topspace bdmd.mtopology
   by(simp add: bdmd.mtopology-topspace pd.mtopology-topspace)
 have 4: inj-on f (topspace pd.mtopology)
   unfolding pd.mtopology-topspace
 proof
   fix x y
   assume h:x ∈ (ΠE i∈UNIV. {0,1}) y ∈ (ΠE i∈UNIV. {0,1}) f x = f y
   show x = y
   proof
     fix n
```

have *f y* ∈ *md.closed-ball* (*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(*auto simp*: *bdmd-sub mpd-top*)
qed

lemma(**in** *polish-topology*) *uncountable-contains-Cantor-space*:
assumes *uncountable* (*topspace S*)
shows ∃ *A*. *g-delta-of S A* ∧ *Cantor-space-as-topology homeomorphic-space* (*subtopology S A*)
proof −
obtain *A* **where** *A*:*A* ⊆ *topspace S Cantor-space-as-topology homeomorphic-space* (*subtopology S A*)
using *uncountable-contains-Cantor-space'*[*OF assms*] **by** *auto*
then have *g-delta-of S A*
using *Cantor-space-Polish-topology*
by(*auto intro*!: *complete-metrizable-homeo-image-g-delta simp*: *polish-topology-def complete-metrizable-axioms*)
thus *?thesis*
by(*auto intro*!: *exI*[**where** *x*=*A*] *A*(*2*))
qed

3.6 Borel Spaces

Borel spaces generated from abstract topology

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

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

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

lemma *sets-borel-of-closed*: *sets* (*borel-of S*) = *sigma-sets* (*topspace S*) {*U*. *closedin S U*}

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

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

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

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

lemma *borel-of-g-delta-of*:
assumes *g-delta-of S U*
shows *U ∈ sets (borel-of S)*
using *g-delta-ofD*[*OF assms*] *borel-of-open*
by(*auto intro*!: *sets.countable-INT'*[*of - id*,*simplified*])

lemma *borel-of-subtopology*:
borel-of (subtopology S U) = restrict-space (borel-of S) U
proof(*rule measure-eqI*)
show *sets (borel-of (subtopology S U)) = sets (restrict-space (borel-of S) 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 S U))*
then obtain *T* **where** *openin S T a = T ∩ U*

by(*auto simp*: *openin-subtopology*)
show *a* ∈ *sigma-sets* (*topspace S* ∩ *U*) {(λ*x*. *x*) −‘ *A* ∩ (*topspace S* ∩ *U*) |*A*. *A* ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))}
using *openin-subset*[*OF* ‹*openin S T*›] ‹*a* = *T* ∩ *U*› **by**(*auto intro*!: *exI*[**where** *x*=*T*] ‹*openin S T*›)
next
fix *b*
assume *b* ∈ {(λ*x*. *x*) −‘ *A* ∩ (*topspace S* ∩ *U*) |*A*. *A* ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))}
then obtain *T* **where** *ht*:*b* = *T* ∩ (*topspace S* ∩ *U*) *T* ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))
by *auto*
hence *b* = *T* ∩ *U*
proof −
have *T* ⊆ *topspace S*
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 S* ∩ *U*) (*Collect* (*openin* (*subtopology S 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(**in** *metrizable*) *sigma-sets-eq-cinter-dunion*:
sigma-sets (*topspace S*) {*U*. *openin S U*} = *sigma-sets-cinter-dunion* (*topspace S*) {*U*. *openin S U*}
proof *safe*
fix *a*
interpret *sa*: *sigma-algebra topspace S sigma-sets* (*topspace S*) {*U*. *openin S U*}
by(*auto intro*!: *sigma-algebra-sigma-sets openin-subset*)

assume $a \in$ *sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S U*$\}$
then show $a \in$ *sigma-sets* (*topspace S*) $\{U.$ *openin S U*$\}$
by *induction auto*
next
have *c*:*sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S U*$\} \subseteq \{U \in$ *sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S U*$\}.$ *topspace S* $-$ *U* $\in$ *sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S U*$\}\}$
proof
fix *a*
assume *a*: $a \in$ *sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S U*$\}$
then show $a \in \{U \in$ *sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S U*$\}.$ *topspace S* $-$ *U* $\in$ *sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S U*$\}\}$
proof *induction*
case *a*:(*Basic-cd a*)
then have *g-delta-of S* (*topspace S* $-$ *a*)
by(*auto intro*!: *g-delta-of-closedin*)
from *g-delta-ofD'*[*OF this*] **obtain** *U* **where** *U*:
$\bigwedge n ::$ *nat. openin S* (*U n*) *topspace S* $-$ *a* $= \bigcap$ (*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* $\equiv (\lambda n.$ (*topspace S* $-$ *a n*) $\cap$ ($\bigcap i.$ *if i* $<$ *n then a i else topspace S*))
have *bd*:*disjoint-family b*
using *nat-neq-iff* **by**(*fastforce simp*: *disjoint-family-on-def b-def*)
have *bin*:*b i* $\in$ *sigma-sets-cinter-dunion* (*topspace S*) $\{U.$ *openin S 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 S* $-$ ($\bigcap$ (*range a*)) = ($\bigcup i.$ *b i*) (**is** *?lhs* = *?rhs*)
proof $-$
{ fix *x*
have *x* $\in$ *?lhs* $\longleftrightarrow$ *x* $\in$ *topspace S* $\wedge$ *x* $\in$ ($\bigcup i.$ *topspace S* $-$ *a i*)
by *auto*
also have ... $\longleftrightarrow$ *x* $\in$ *topspace S* $\wedge$ ($\exists n.$ *x* $\in$ *topspace S* $-$ *a n*)
by *auto*
also have ... $\longleftrightarrow$ *x* $\in$ *topspace S* $\wedge$ ($\exists n.$ *x* $\in$ *topspace S* $-$ *a n* $\wedge$ ($\forall i<n.$ *x* $\in$ *a i*))
proof *safe*
fix *n*
assume *1*:*x* $\notin$ *a n x* $\in$ *topspace S*
define *N* **where** *N* $\equiv$ *Min* $\{m.$ *m* $\leq$ *n* $\wedge$ *x* $\notin$ *a m*$\}$
have *N*:*x* $\notin$ *a N N* $\leq$ *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 S − 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 S) {U. openin S 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 S − (⋃ (range a)) = (⋂ i. (topspace S − a i))
    by simp
  have ... ∈ sigma-sets-cinter-dunion (topspace S) {U. openin S 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 S) {U. openin S U}
then show a ∈ sigma-sets-cinter-dunion (topspace S) {U. openin S U}
proof induction
 case a:(Union a)
  define b where b ≡ (λn. a n ∩  (⋂ i. if i < n then topspace S − a i else
topspace S))
 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 S) {U. openin S 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 S ∧ 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 S ∧ (∃ n. x ∈ a n) by auto
        also have ... ⟷ x ∈ topspace S ∧ (∃ n. x ∈ a n ∧ (∀ i<n. x ∈ topspace S
− a i))
        proof safe
          fix n
          assume 1:x ∈ topspace S 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 S − 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 S ∃ n. x ∈ a n ∧ (∀ i<n. x ∈ topspace S − 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 S) {U. openin S U}
      by(rule Union-cd) (use bin bd in auto)
    thus ?case
      by(auto simp: bun)
  qed(use c in auto)
qed

lemma(in metrizable) sigma-sets-eq-cinter:
 sigma-sets (topspace S) {U. openin S U} = sigma-sets-cinter (topspace S) {U.
openin S U}
proof safe
  fix a
```

interpret *sa*: *sigma-algebra topspace S sigma-sets* (*topspace S*) {*U*. *openin S U*}
by(*auto intro*!: *sigma-algebra-sigma-sets openin-subset*)
assume *a* ∈ *sigma-sets-cinter* (*topspace S*) {*U*. *openin S U*}
then show *a* ∈ *sigma-sets* (*topspace S*) {*U*. *openin S U*}
by *induction auto*
qed (*use sigma-sets-cinter-dunion-subset sigma-sets-eq-cinter-dunion* **in** *auto*)

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 *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-of 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-of-subset*[*OF assms*(*2*)] **by** *auto*
next
interpret *s*: *sigma-algebra topspace S sigma-sets* (*topspace S*) $\mathcal{U}$

using *subbase-of-subset*[*OF assms*(*2*)] **by**(*auto intro*!: *sigma-algebra-sigma-sets*)
obtain $\mathcal{O}$ **where** *ho*: *countable* $\mathcal{O}$ *base-of S* $\mathcal{O}$
using *assms*(*1*) **by**(*auto simp*: *second-countable-def*)
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-of-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*:
⋀*k*. *k* ∈ *K* ⟹ *V k* ⊆ $\mathcal{O}$ ⋀*k*. *k* ∈ *K* ⟹ ⋃ (*V k*) = *k*
by(*simp add*: *base-of-def openin-topology-generated-by-iff*[*symmetric*] *assms*(*2*)[*simplified subbase-of-def*,*symmetric*]) *metis*
define $\mathcal{U}k$ **where** $\mathcal{U}k$ = (⋃*k*∈*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 ⋃ $\mathcal{U}k$ = (⋃*A*∈$\mathcal{U}k$. *A*) **by** *auto*
also have ... = ⋃ *K*
unfolding $\mathcal{U}k$-*def UN-simps* **by**(*simp add*: *hv*(*2*))
finally have *1*:⋃ $\mathcal{U}k$ = ⋃ *K* **.**
have ∀ *b*∈$\mathcal{U}k$. ∃ *k*∈*K*. *b* ⊆ *k*
using *hv* **by** (*auto simp*: $\mathcal{U}k$-*def*)
then obtain *V'* **where** *hv'*: ⋀*b*. *b* ∈ $\mathcal{U}k$ ⟹ *V' b* ∈ *K* **and** ⋀*b*. *b* ∈ $\mathcal{U}k$ ⟹ *b* ⊆ *V' b*
by *metis*
then have (⋃ *b*∈$\mathcal{U}k$. *V' b*) ⊆ ⋃ *K* ⋃$\mathcal{U}k$ ⊆ (⋃ *b*∈$\mathcal{U}k$. *V' b*)
by *auto*
then have ⋃ *K* = (⋃ *b*∈$\mathcal{U}k$. *V' b*)
unfolding *1* **by** *auto*
also have ... ∈ *sigma-sets* (*topspace S*) $\mathcal{U}$
using *hv' UN* **by**(*auto intro*!: *s.countable-UN' simp*: *0*)
finally show ⋃ *K* ∈ *sigma-sets* (*topspace S*) $\mathcal{U}$ **.**
qed *auto*
next
fix *U*
assume *U* ∈ $\mathcal{U}$
from *assms*(*2*)[*simplified subbase-of-def*] *openin-topology-generated-by-iff generate-topology-on.Basis*[*OF this*]
show *U* ∈ *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-of S* $\mathcal{O}$ *countable* $\mathcal{O}'$ *base-of S′* $\mathcal{O}'$
using *assms* **by**(*auto simp*: *second-countable-def*)
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-of-def*,*symmetric*]] *topology-generated-by-topspace*[*of* $\mathcal{O}'$,*simplified base-is-subbase*[*OF ho*(*4*),*simplified subbase-of-def*,*symmetric*]]
base-of-openin[*OF ho*(*2*)] *base-of-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-of-def*,*OF prod-topology-generated-by-open*]
by *simp*
finally show *?thesis* **.**
qed

lemma *product-borel-of-measurable*:
assumes *i* ∈ *I*
shows (λ*x. x i*) ∈ (*borel-of* (*product-topology S I*)) →$_M$ *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*:$x \in (\bigcap i{\in}I'.\ (\lambda x.\ x\ i) -` X\ i \cap space\ (borel\text{-}of\ (product\text{-}topology\ S\ I))) \cap space\ (borel\text{-}of\ (product\text{-}topology\ S\ I))$
have $x\ i \in X\ i$ **if** *hi*:$i \in I$ **for** i
proof −
consider $i \in I' \wedge I' \neq \{\} \mid i \notin I' \wedge I' = \{\} \mid i \notin I' \wedge I' \neq \{\}$ **by** *auto*
then show *?thesis*
apply *cases*
using *sets.sets-into-space*[*OF* ‹$\bigwedge i.\ X\ i \in sets\ (borel\text{-}of\ (S\ i))$›] *1 that*
by(*auto simp*: *space-borel-of I'-def*)
qed
then show $x \in Pi_E\ I\ X$
using *1* **by**(*auto simp*: *space-borel-of*)
qed
also have $... \in sets\ (borel\text{-}of\ (product\text{-}topology\ S\ I))$
using *that* ‹*finite I'*› **by**(*auto simp*: *I'-def*)
finally show *?thesis* **.**
qed
then have $\{Pi_E\ I\ X \mid X.\ (\forall i.\ X\ i \in sets\ (borel\text{-}of\ (S\ i))) \wedge finite\ \{i.\ X\ i \neq space\ (borel\text{-}of\ (S\ i))\}\} \subseteq sets\ (borel\text{-}of\ (product\text{-}topology\ S\ I))$
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 $sets\ (Pi_M\ UNIV\ (\lambda i.\ borel)) = sets\ borel$.

lemma *sets-PiM-equal-borel-of*:
assumes *countable I* **and** $\bigwedge i.\ i \in I \Longrightarrow second\text{-}countable\ (S\ i)$
shows $sets\ (\Pi_M\ i{\in}I.\ borel\text{-}of\ (S\ i)) = sets\ (borel\text{-}of\ (product\text{-}topology\ S\ I))$
proof
obtain K **where** *hk*:
countable K base-of (*product-topology S I*) K
$\bigwedge k.\ k \in K \Longrightarrow \exists X.\ (k = (\Pi_E\ i{\in}I.\ X\ i)) \wedge (\forall i.\ openin\ (S\ i)\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq topspace\ (S\ i)\} \wedge \{i.\ X\ i \neq topspace\ (S\ i)\} \subseteq I$
using *product-topology-countable-base-of*[*OF assms*(*1*)] *assms*(*2*)
by *force*
have $*$:$k \in sets\ (\Pi_M\ i{\in}I.\ borel\text{-}of\ (S\ i))$ **if** $k \in K$ **for** k
proof −
obtain X **where** H: $k = (\Pi_E\ i{\in}I.\ X\ i)\ \bigwedge i.\ openin\ (S\ i)\ (X\ i)\ finite\ \{i.\ X\ i \neq topspace\ (S\ i)\}\ \{i.\ X\ i \neq topspace\ (S\ i)\} \subseteq I$
using *hk*(*3*)[*OF* ‹$k \in K$›] **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 $**$: $U \in sets\ (\Pi_M\ i{\in}I.\ borel\text{-}of\ (S\ i))$ **if** $openin\ (product\text{-}topology\ S\ I)\ U$ **for** U
proof −

obtain B **where** $B \subseteq K$ $U = (\bigcup B)$
using ‹*openin* (*product-topology S I*) *U*› ‹*base-of* (*product-topology S I*) *K*›
by (*metis base-of-def*)
have *countable B* **using** ‹$B \subseteq K$› ‹*countable K*› *countable-subset* **by** *blast*
moreover have $k \in$ *sets* (Π_M *i*∈*I*. *borel-of* (*S i*)) **if** $k \in B$ **for** k
using ‹$B \subseteq K$› ∗ *that* **by** *auto*
ultimately show *?thesis* **unfolding** ‹$U = (\bigcup B)$› **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*
⋀*x*. *x* ∈ *topspace X* ⟹ *g* (*f x*) = *x*
⋀*y*. *y* ∈ *topspace Y* ⟹ *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 ∃ *S'*. *homeomorphic-map S S' f* ∧ *sets N* = *sets* (*borel-of S'*)
proof −
obtain *g* **where** *fg*:*f* ∈ *M* →$_M$ *N g* ∈ *N* →$_M$ *M* ⋀*x*. *x*∈*space M* ⟹ *g* (*f x*) = *x* ⋀*y*. *y*∈*space N* ⟹ *f* (*g y*) = *y* ⋀*A*. *A*∈*sets M* ⟹ *f* ‘ *A* ∈ *sets N* ⋀*A*. *A*∈*sets N* ⟹ *g* ‘ *A* ∈ *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 ⊆ 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 ∈ u
      with 1 fg(1) show f x ∈ space N by(auto simp: measurable-space)
    next
      fix u
      assume openin S (g ` u) u ⊆ space N
      with fg(4) show u ∈ (`) 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*)

lemma(**in** *polish-topology*) *closedin-clopen-topology*:
assumes *closedin S a*
shows ∃ *S'. polish-topology S' ∧ (∀ u. openin S u ⟶ openin S' u) ∧ topspace S = topspace S' ∧ sets (borel-of S) = sets (borel-of S') ∧ openin S' a ∧ closedin S' a*
proof −
have *polish-topology (subtopology S a)*
by(*rule closedin-polish*[*OF assms*])
from *polish-topology.bounded-polish-metric*[*OF this*] **obtain** *da* **where** *da*:
polish-metric-set a da subtopology S a = metric-set.mtopology a da ⋀x y. da x y < 1
by (*metis topspace-subtopology-subset closedin-subset*[*OF assms*])
interpret *pa*: *polish-metric-set a da* **by** *fact*
have *polish-topology (subtopology S (topspace S − a))*
using *assms* **by**(*auto intro*!: *openin-polish*)
from *polish-topology.bounded-polish-metric*[*OF this*]
obtain *db* **where** *db*: *polish-metric-set (topspace S − a) db subtopology S (topspace S − a) = metric-set.mtopology (topspace S − a) db ⋀x y. db x y < 1*
by (*metis Diff-subset topspace-subtopology-subset*)
interpret *pb*: *polish-metric-set topspace S − a db* **by** *fact*
interpret *p*: *sum-polish-metric UNIV λb. if b then a else topspace S − a λb. if b then da else db*
using *da db* **by**(*auto intro*!: *sum-polish-metricI simp*: *disjoint-family-on-def*)
have *0*: (⋃ *i. if i then a else topspace S − a) = topspace S*
using *closedin-subset assms* **by** *auto*

have *1*: *sets (borel-of S) = sets (borel-of p.mtopology)*
proof −
have *sigma-sets (topspace S) (Collect (openin S)) = sigma-sets (topspace S) (Collect (openin p.mtopology))*
proof(*rule sigma-sets-eqI*)
fix *a*
assume *a ∈ Collect (openin S)*
then have *openin p.mtopology a*
by(*simp only*: *p.openin-sum-mtopology-iff*) (*auto simp*: *0 da*(*2*)[*symmetric*] *db*(*2*)[*symmetric*] *openin-subtopology dest*:*openin-subset*)
thus *a ∈ sigma-sets (topspace S) (Collect (openin p.mtopology))*
by *auto*
next
interpret *s*: *sigma-algebra topspace S sigma-sets (topspace S) (Collect (openin*

S))
by(*auto intro*!: *sigma-algebra-sigma-sets openin-subset*)
fix *b*
assume *b* ∈ *Collect* (*openin p.mtopology*)
then have *openin p.mtopology b* **by** *auto*
then have *b*:*b* ⊆ *topspace S openin* (*subtopology S a*) (*b* ∩ *a*) *openin* (*subtopology S* (*topspace S* − *a*)) (*b* ∩ (*topspace S* − *a*))
by(*simp-all only*: *p.openin-sum-mtopology-iff*,*insert 0 da*(*2*) *db*(*2*)) (*auto simp*: *all-bool-eq*)
have [*simp*]: (*b* ∩ *a*) ∪ (*b* ∩ (*topspace S* − *a*)) = *b*
using *Diff-partition b*(*1*) **by** *blast*
have (*b* ∩ *a*) ∪ (*b* ∩ (*topspace S* − *a*)) ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))
proof(*rule sigma-sets-Un*)
have [*simp*]:*a* ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))
proof −
have *topspace S* − (*topspace S* − *a*) ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))
by(*rule sigma-sets.Compl*) (*use assms* **in** *auto*)
thus *?thesis*
using *double-diff*[*OF closedin-subset*[*OF assms*]] **by** *simp*
qed
from *b*(*2*,*3*) **obtain** *T T*′ **where** *T*:*openin S T openin S T*′ **and** [*simp*]:*b* ∩ *a* = *T* ∩ *a b* ∩ (*topspace S* − *a*) = *T*′ ∩ (*topspace S* − *a*)
by(*auto simp*: *openin-subtopology*)
show *b* ∩ *a* ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))
b ∩ (*topspace S* − *a*) ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))
using *T assms* **by** *auto*
qed
thus *b* ∈ *sigma-sets* (*topspace S*) (*Collect* (*openin S*))
by *simp*
qed
thus *?thesis*
by(*simp only*: *sets-borel-of p.mtopology-topspace*) (*use 0* **in** *auto*)
qed
have *2*:⋀*u*. *openin S u* ⟹ *openin p.mtopology u*
by(*simp only*: *p.openin-sum-mtopology-iff*) (*auto simp*: *all-bool-eq da*(*2*)[*symmetric*] *db*(*2*)[*symmetric*] *openin-subtopology dest*:*openin-subset*)
have *3*:*openin p.mtopology a*
by(*simp only*: *p.openin-sum-mtopology-iff*) (*auto simp*: *all-bool-eq*)
have *4*:*closedin p.mtopology a*
by (*metis 0 2 assms closedin-def p.mtopology-topspace*)
have *5*: *topspace S* = *topspace p.mtopology*
by(*simp only*: *p.mtopology-topspace*) (*simp only*: *0*)
have *6*: *polish-topology p.mtopology*
using *p.polish-topology-axioms* **by** *blast*
show *?thesis*
by(*rule exI*[**where** *x*=*p.mtopology*]) (*insert 5 2 6*, *simp only*: *1 3 4* ,*auto*)
qed

lemma *polish-topology-union-polish*:
fixes $X :: nat \Rightarrow 'a\ topology$
assumes $\bigwedge n.\ polish\text{-}topology\ (X\ n)\ \bigwedge n.\ topspace\ (X\ n) = Xt\ \bigwedge x\ y.\ x \in Xt \Longrightarrow y \in Xt \Longrightarrow x \neq y \Longrightarrow \exists\, Ox\ Oy.\ (\forall\, n.\ openin\ (X\ n)\ Ox) \wedge (\forall\, n.\ openin\ (X\ n)\ Oy) \wedge x \in Ox \wedge y \in Oy \wedge disjnt\ Ox\ Oy$
defines $Xun \equiv topology\text{-}generated\text{-}by\ (\bigcup n.\ \{u.\ openin\ (X\ n)\ u\})$
shows *polish-topology Xun*
proof −
have *topsXun*:*topspace Xun* = *Xt*
using *assms*(*2*) **by**(*auto simp*: *Xun-def dest*:*openin-subset*)
define $f :: 'a \Rightarrow nat \Rightarrow 'a$ **where** $f \equiv (\lambda x\ n.\ x)$
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*=$\{\ \lambda i.\ if\ i = n\ then\ a\ else\ b\ i\ |a\ b.\ a \in u \wedge b \in UNIV \rightarrow Xt\}$])
show *openin* (*product-topology X UNIV*) $\{\lambda i.\ if\ i = n\ then\ a\ else\ b\ i\ |a\ b.\ a \in u \wedge b \in UNIV \rightarrow Xt\}$
by(*auto simp*: *openin-product-topology-alt u assms*(*2*) *openin-topspace*[*of X -*,*simplified assms*(*2*)] *intro*!: *exI*[**where** *x*=$\lambda i.\ if\ i = n\ then\ u\ else\ Xt$])
(*auto simp*: *PiE-def Pi-def*, *metis openin-subset*[*OF u*,*simplified assms*(*2*)] *in-mono*)
next
show $\bigwedge y.\ y \in u \Longrightarrow \exists\, a\ b.\ f\ y = (\lambda i.\ if\ i = n\ then\ a\ else\ b\ i) \wedge a \in u \wedge b \in UNIV \rightarrow Xt$
using *assms*(*2*) *f-def openin-subset u* **by** *fastforce*
next
show $\bigwedge y.\ y \in u \Longrightarrow f\ y \in f$ ‘ *topspace Xun*
using *openin-subset*[*OF u*] **by**(*auto simp*: *assms*(*2*) *topsXun*)

next
show $\bigwedge x\ xa\ a\ b.\ xa \in topspace\ Xun \Longrightarrow f\ xa = (\lambda i.\ if\ i = n\ then\ a\ else\ b\ i) \Longrightarrow a \in u \Longrightarrow b \in UNIV \to Xt \Longrightarrow f\ xa \in f\ `\ u$
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*!: *polish-topology.homeomorphic-polish-topology*[*OF polish-topology.closedin-polish*[*OF polish-topology-product*] *5*] *assms*)
show *closedin* (*product-topology X UNIV*) (*f* ` *topspace Xun*)
proof −
have *1*: *openin* (*product-topology X UNIV*) ((*UNIV* $\to_E$ *Xt*) − *f* ` *Xt*)
proof(*rule openin-subopen*[*THEN iffD2*])
show $\forall x{\in}(UNIV \to_E Xt) - f\ `\ Xt.\ \exists T.\ openin\ (product\text{-}topology\ X\ UNIV)\ T \wedge x \in T \wedge T \subseteq (UNIV \to_E Xt) - f\ `\ Xt$
proof *safe*
fix *x*
assume *x*:$x \in UNIV \to_E Xt\ x \notin f\ `\ Xt$
have $\exists n.\ x\ n \neq x\ 0$
proof(*rule ccontr*)
assume $\nexists n.\ x\ n \neq x\ 0$
then have $\forall n.\ x\ n = x\ 0$ **by** *auto*
hence $x = (\lambda\text{-}.\ x\ 0)$ **by** *auto*
thus *False*
using *x* **by**(*auto simp*: *f-def topsXun assms*(*2*))
qed
then obtain *n* **where** *n*: $n \neq 0\ x\ n \neq x\ 0$
by *metis*
from *assms*(*3*)[*OF - - this*(*2*)] *x*
obtain *On O0* **where** *h*:$\bigwedge n.\ openin\ (X\ n)\ On\ \bigwedge n.\ openin\ (X\ n)\ O0\ x\ n \in On\ x\ 0 \in O0\ disjnt\ On\ O0$
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*:$x \in (\lambda T.\ T\ 0) -` O0 \cap topspace\ (product\text{-}topology\ X\ UNIV) \cap ((\lambda T.\ T\ n) -` On \cap topspace\ (product\text{-}topology\ X\ UNIV))$
using x $h(3,4)$ **by**(*auto simp*: *assms*(2))
have *subset*:$(\lambda T.\ T\ 0) -` O0 \cap topspace\ (product\text{-}topology\ X\ UNIV) \cap ((\lambda T.\ T\ n) -` On \cap topspace\ (product\text{-}topology\ X\ UNIV)) \subseteq (UNIV \to_E Xt) - f ` Xt$
using $h(5)$ **by**(*auto simp*: *assms*(2) *disjnt-def f-def*)

show $\exists T.\ openin\ (product\text{-}topology\ X\ UNIV)\ T \wedge x \in T \wedge T \subseteq (UNIV \to_E Xt) - f ` Xt$
by(*rule exI*[**where** x=$((\lambda x.\ x\ 0) -` O0 \cap topspace\ (product\text{-}topology\ X\ UNIV)) \cap ((\lambda x.\ x\ n) -` On \cap topspace\ (product\text{-}topology\ X\ UNIV))$]) (*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(**in** *polish-topology*) *sets-clopen-topology*:
assumes $a \in sets\ (borel\text{-}of\ S)$
shows $\exists S'.\ polish\text{-}topology\ S' \wedge (\forall u.\ openin\ S\ u \longrightarrow openin\ S'\ u) \wedge topspace\ S = topspace\ S' \wedge sets\ (borel\text{-}of\ S) = sets\ (borel\text{-}of\ S') \wedge openin\ S'\ a \wedge closedin\ S'\ a$
proof −
have $a \in sigma\text{-}sets\ (topspace\ S)\ \{U.\ closedin\ S\ U\}$
using *assms* **by**(*simp add*: *sets-borel-of-closed*)
thus *?thesis*
proof *induction*
case (*Basic a*)
then show *?case*
by(*simp add*: *closedin-clopen-topology*)
next
case *Empty*
with *polish-topology-axioms* **show** *?case*
by *auto*
next
case (*Compl a*)
then obtain S' **where** S':*polish-topology* S' $(\forall u.\ openin\ S\ u \longrightarrow openin\ S'\ u)$ $topspace\ S = topspace\ S'$ $sets\ (borel\text{-}of\ S) = sets\ (borel\text{-}of\ S')$ $openin\ S'\ a$ $closedin\ S'\ a$
by *auto*
from *polish-topology.closedin-clopen-topology*[*OF* $S'(1)$ $S'(6)$] S'
show *?case* **by** *auto*
next
case *ih*:(*Union a*)

then obtain *Si* **where** *Si*:
$\bigwedge i.$ *polish-topology* (*Si i*) $\bigwedge u\ i.$ *openin S u* $\Longrightarrow$ *openin* (*Si i*) *u* $\bigwedge i$::*nat. topspace S* = *topspace* (*Si i*) $\bigwedge i.$ *sets* (*borel-of S*) = *sets* (*borel-of* (*Si i*)) $\bigwedge i.$ *openin* (*Si i*) (*a i*) $\bigwedge i.$ *closedin* (*Si i*) (*a i*)
by *metis*
define *Sun* **where** *Sun* ≡ *topology-generated-by* ($\bigcup n.$ {*u. openin* (*Si n*) *u*})
have *Sun1*: *polish-topology Sun*
unfolding *Sun-def*
proof(*safe intro*!: *polish-topology-union-polish*[**where** *Xt*=*topspace S*])
fix *x y*
assume *xy*:*x* ∈ *topspace S y* ∈ *topspace S x* ≠ *y*
then obtain *Ox Oy* **where** *Oxy*: *x* ∈ *Ox y* ∈ *Oy openin S Ox openin S Oy disjnt Ox Oy*
using *Hausdorff* **by**(*auto simp*: *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 S* = *topspace Sun*
using *Si*(3) **by**(*auto simp*: *Sun-def dest*: *openin-subset*)
have *Sunsets*: *sets* (*borel-of S*) = *sets* (*borel-of Sun*) (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *sigma-sets* (*topspace S*) ($\bigcup n.$ {*u. openin* (*Si n*) *u*})
proof
show *sets* (*borel-of S*) ⊆ *sigma-sets* (*topspace S*) ($\bigcup n.$ {*u. openin* (*Si n*) *u*})
using *Si*(2) **by**(*auto simp*: *sets-borel-of intro*!: *sigma-sets-mono'*)
next
show *sigma-sets* (*topspace S*) ($\bigcup n.$ {*u. openin* (*Si n*) *u*}) ⊆ *sets* (*borel-of S*)
by(*simp add*: *sigma-sets-le-sets-iff*[*of borel-of S* $\bigcup n.$ {*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-topology.S-second-countable*[*OF Sun1*],*of* $\bigcup n.$ {*u. openin* (*Si n*) *u*}]
by (*simp add*: *Sun-def Suntop subbase-of-def subset-Pow-Union*)
finally show *?thesis* **.**
qed
have *Sun-open*: $\bigwedge u\ i.$ *openin* (*Si i*) *u* $\Longrightarrow$ *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 *polish-topology.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

4 Standard Borel Spaces

4.1 Standard Borel Spaces

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

locale *standard-borel* =
fixes *M* :: *'a measure*
assumes *polish-topology*: ∃ *S*. *polish-topology S* ∧ *sets M* = *sets* (*borel-of S*)
begin

lemma *singleton-sets*:
assumes *x* ∈ *space M*
shows {*x*} ∈ *sets M*
proof −
obtain *S* **where** *s*:*polish-topology S sets M* = *sets* (*borel-of S*)
using *polish-topology* **by** *blast*
interpret *s*:*polish-topology S* **by** *fact*
have *closedin S* {*x*}
using *s.closedin-singleton*[*of x*] *assms sets-eq-imp-space-eq*[*OF s*(*2*)]
by(*simp add*: *space-borel-of*)
thus *?thesis*
using *borel-of-closed s* **by** *simp*
qed

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

lemma *standard-borel-restrict-space*:
assumes *A* ∈ *sets M*
shows *standard-borel* (*restrict-space M A*)
proof −
obtain *S* **where** *s*:*polish-topology S sets M* = *sets* (*borel-of S*)
using *polish-topology* **by** *blast*
obtain *S'* **where** *S'*:*polish-topology S' sets M* = *sets* (*borel-of S'*) *openin S' A*
using *polish-topology.sets-clopen-topology*[*OF s*(*1*),*simplified s*(*2*)[*symmetric*],*OF assms*] **by** *auto*
show *?thesis*

using *polish-topology.openin-polish*[*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* ≠ {}
begin

lemma *standard-borel-ne-restrict-space*:
assumes *A* ∈ *sets M A* ≠ {}
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* ⨂$_M$ *N*)
proof −
obtain *S S'* **where** *hs*:
polish-topology S sets M = *sets* (*borel-of S*) *polish-topology S' sets N* = *sets* (*borel-of S'*)
using *assms* **by**(*auto simp*: *standard-borel-def*)
have *sets* (*M* ⨂$_M$ *N*) = *sets* (*borel-of* (*prod-topology S S'*))
unfolding *borel-of-prod*[*OF polish-topology.S-second-countable*[*OF hs*(*1*)] *polish-topology.S-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-topology-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-topology* ($S\ i$) $\bigwedge i.\ i \in I \Longrightarrow$ *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-topology.S-second-countable*[*OF hs*(*1*)] **by**(*auto intro*!: *sets-PiM-equal-borel-of*)
finally have *1*:*sets* ($\Pi_M\ i{\in}I.\ M\ i$) = *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-topology-product simp*: *1*)
qed

lemma *product-standard-borel-ne*:
assumes *countable I*
and $\bigwedge i.\ i \in I \Longrightarrow$ *standard-borel-ne* ($M\ i$)
shows *standard-borel-ne* ($\Pi_M\ i{\in}I.\ M\ i$)
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-topology* (*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-topology-closedin-polish*)

lemma *closed-set-standard-borel-ne*[*simp*]:
fixes *U* :: *'a* :: *topological-space set*
assumes *polish-topology* (*euclidean* :: *'a topology*) *closed U* $U \neq \{\}$
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 $U :: 'a :: topological\text{-}space\ set$
assumes *polish-topology* (*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-topology.openin-polish*)

lemma *open-set-standard-borel-ne*[*simp*]:
fixes $U :: 'a :: topological\text{-}space\ set$
assumes *polish-topology* (*euclidean* :: *'a topology*) *open U* $U \neq \{\}$
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*)
proof −
interpret *polish-metric-set I discrete-dist I*
by(*simp add*: *discrete-dist-polish-iff assms*)
show *?thesis*
unfolding *standard-borel-def*
proof(*intro exI*[**where** *x*=*mtopology*] *conjI*)
have $\bigwedge x.\ x \in I \Longrightarrow \{x\} \in sets\ (borel\text{-}of\ mtopology)$
unfolding *sets-borel-of* **by**(*rule sigma-sets.Basic*) (*simp add*: *discrete-dist-topology*)
hence *sets* (*borel-of mtopology*) = *Pow I*
by(*auto intro*!: *sets-eq-countable*[*OF assms*] *simp*: *space-borel-of mtopology-topspace*)
thus *sets* (*count-space I*) = *sets* (*borel-of mtopology*)
by *simp*
qed (*rule polish-topology-axioms*)
qed

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-topology S sets M* = *sets* (*borel-of S*)
using *polish-topology* **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 polish-topology.homeomorphic-polish-topology*[*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 *ereal-standard-ne*: *standard-borel-ne* (*borel* :: *ereal measure*)
proof −
interpret *s*: *standard-borel-ne restrict-space borel* {*0..1*::*real*}
by *auto*
define *f* :: *real* ⇒ *ereal*
where *f* ≡ (λ*r*. *if r* = *0 then bot else if r* = *1 then top else tan* (*pi* ∗ *r* − (*pi* / *2*)))
define *g* :: *ereal* ⇒ *real*
where *g* ≡ (λ*r*. *if r* = *top then 1 else if r* = *bot then 0 else arctan* (*real-of-ereal r*) / *pi* + *1* / *2*)

```
  show ?thesis
  proof(rule s.measurable-isomorphic-standard-ne[OF measurable-isomorphic-byWitness[where
f=f and g = g]])
    show f ∈ borel-measurable (restrict-space borel {0..1})
    proof −
      have 1:{0..1} ∩ {r. r ≠ 0} ∩ {x. x ≠ 1} = {0<..<1::real} by auto
      have 2:(λx. ereal (tan (pi ∗ x − pi / 2))) ∈ borel-measurable (restrict-space
borel ({0..1} ∩ {r. r ≠ 0} ∩ {x. x ≠ 1}))
        unfolding 1
      proof(safe intro!: borel-measurable-continuous-on-restrict continuous-on-ereal
Transcendental.continuous-on-tan)
        show continuous-on {0<..<1} (λx::real. pi ∗ x − pi / 2)
          by(auto intro!: continuous-at-imp-continuous-on)
      next
        fix x :: real
        assume h:cos (pi ∗ x − pi / 2) = 0 x ∈ {0<..<1}
        hence − (pi / 2) < pi ∗ x − pi / 2 pi ∗ x − pi / 2 < pi / 2
          by simp-all
        from cos-gt-zero-pi[OF this] h(1)
        show False by simp
      qed
      have {r:: real. r = 0 ∧ 0 ≤ r ∧ r ≤ 1} ∈ sets (restrict-space borel {0..1})
{x::real. x = 1 ∧ 0 ≤ x ∧ x ≤ 1 ∧ x ≠ 0} ∈ sets (restrict-space borel ({0..1} ∩
{r. r ≠ 0}))
        by(auto simp: sets-restrict-space)
      with 2 show ?thesis
      by(auto intro!: measurable-If-restrict-space-iff[THEN iffD2] simp: restrict-restrict-space
f-def)
    qed
  next
    show g ∈ borel →M restrict-space borel {0..1}
      unfolding g-def measurable-restrict-space2-iff
    proof safe
      fix x :: ereal
      have −1 / 2 < arctan (real-of-ereal x) / pi arctan (real-of-ereal x) / pi < 1
/ 2
        using arctan-lbound[of real-of-ereal x] arctan-ubound[of real-of-ereal x]
        by (simp-all add: mult-imp-less-div-pos)
      hence 0 ≤ arctan (real-of-ereal x) / pi + 1 / 2 arctan (real-of-ereal x) / pi
+ 1 / 2 ≤ 1
        by linarith+
      thus (if x = ⊤ then 1 else if x = ⊥ then 0 else arctan (real-of-ereal x) / pi
+ 1 / 2) ∈ {0..1}
        by auto
    qed measurable
  next
    fix r ::real
    assume r ∈ space (restrict-space borel {0..1})
    then consider r = 0 | r = 1 | 0 < r r < 1 by auto linarith
```

```
    then show  g (f r) = r
    proof cases
      case 3
      then have 1:- (pi / 2) < pi * r - pi / 2 pi * r - pi / 2 < pi / 2
        by simp-all
      have arctan (tan (pi * r - pi / 2)) / pi + 1 / 2 = r
        by(simp add: arctan-tan[OF 1] diff-divide-distrib)
      thus ?thesis
        by(auto simp: f-def g-def top-ereal-def bot-ereal-def)
    qed(auto simp: g-def f-def top-ereal-def bot-ereal-def)
  next
    fix y :: ereal
    consider y = top | y = bot | y ≠ bot y ≠ top by auto
    then show f (g y) = y
    proof cases
      case 3
      hence [simp]: |y| ≠ ∞ by(auto simp: top-ereal-def bot-ereal-def)
      have -1 / 2 < arctan (real-of-ereal y) / pi arctan (real-of-ereal y) / pi < 1
/ 2
        using arctan-lbound[of real-of-ereal y] arctan-ubound[of real-of-ereal y]
        by (simp-all add: mult-imp-less-div-pos)
      hence arctan (real-of-ereal y) / pi + 1 / 2 < 1 arctan (real-of-ereal y) / pi
+ 1 / 2 > 0
        by linarith+
      thus ?thesis
        using arctan-lbound[of real-of-ereal y] arctan-ubound[of real-of-ereal y]
        by(auto simp: f-def g-def distrib-left tan-arctan ereal-real')
    qed(auto simp: f-def g-def)
  qed
qed

corollary ennreal-stanndard-ne: standard-borel-ne (borel :: ennreal measure)
  by(auto intro!: standard-borel-ne.measurable-isomorphic-standard-ne[OF standard-borel-ne.standard-borel-ne-
ereal-standard-ne,of {0..},simplified]] measurable-isomorphic-byWitness[where f=e2ennreal
and g=enn2ereal] measurable-restrict-space1 measurable-restrict-space2 enn2ereal-e2ennreal)
```

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-as-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

Baire space

definition *Baire-space* :: (*nat* ⇒ *nat*) *measure* **where**
Baire-space ≡ (Π$_M$ *i*∈ *UNIV*. *borel*)

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

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-as-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

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

lemma *space-Cantor-space*: *space Cantor-space* = (Π$_E$ *i*∈ *UNIV*. {*0*,*1*})
by(*simp add*: *Cantor-space-def space-PiM*)

lemma *space-Cantor-space-01*[*simp*]:
assumes *x* ∈ *space Cantor-space*
shows *0* ≤ *x n x n* ≤ *1 x n* ∈ {*0*,*1*}
using *PiE-mem*[*OF assms*[*simplified space-Cantor-space*],*of n*]
by *auto*

lemma *Cantor-minus-abs-cantor*:
assumes *x* ∈ *space Cantor-space y* ∈ *space Cantor-space*
shows (λ*n*. |*x n* − *y n*|) ∈ *space Cantor-space*
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*

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

definition *Cantor-to-01* :: $(nat \Rightarrow real) \Rightarrow real$ **where**
Cantor-to-01 $\equiv (\lambda x.\ (\sum n.\ (1/3)\hat{}(Suc\ n) * x\ n))$

Cantor-to-01 is a measurable injective embedding.

lemma *Cantor-to-01-summable'*[*simp*]:
assumes $x \in space\ Cantor\text{-}space$
shows *summable* $(\lambda n.\ (1/3)\hat{}(Suc\ n) * x\ n)$
proof(*rule summable-comparison-test'*[**where** $g{=}\lambda n.\ (1/3)\hat{}\ n$ **and** $N{=}0$])
show *norm* $((1\ /\ 3)\ \hat{}\ Suc\ n * x\ n) \leq (1\ /\ 3)\ \hat{}\ n$ **for** n
using *space-Cantor-space-01*[*OF assms*,*of n*] **by** *auto*
qed *simp*

lemma *Cantor-to-01-summable*[*simp*]:
assumes $x \in space\ Cantor\text{-}space$
shows *summable* $(\lambda n.\ (1/3)\hat{}\ n * x\ n)$
using *Cantor-to-01-summable'*[*OF assms*] **by** *simp*

lemma *Cantor-to-01-subst-summable*[*simp*]:
assumes $x \in space\ Cantor\text{-}space\ y \in space\ Cantor\text{-}space$
shows *summable* $(\lambda n.\ (1/3)\hat{}\ n * (x\ n - y\ n))$
proof(*rule summable-comparison-test'*[**where** $g{=}\lambda n.\ (1/3)\hat{}\ n$ **and** $N{=}0$])
show *norm* $((1\ /\ 3)\ \hat{}\ n * (x\ n - y\ n)) \leq (1\ /\ 3)\ \hat{}\ n$ **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*

lemma *Cantor-to-01-image*: *Cantor-to-01* $\in space\ Cantor\text{-}space \to \{0..1\}$
proof
fix x
assume *h*:$x \in space\ Cantor\text{-}space$
have *Cantor-to-01* $x \leq (\sum n.\ (1/3)\hat{}(Suc\ n))$
unfolding *Cantor-to-01-def*
by(*rule suminf-le*) (*use h Cantor-to-01-summable*[*OF h*] **in** *auto*)
also have ... $= (\sum n.\ (1\ /\ 3)\ \hat{}\ n) - (1{::}real)$
using *suminf-minus-initial-segment*[*OF complete-algebra-summable-geometric*[*of 1/3*::*real*],*of 1*]
by *auto*
finally have *Cantor-to-01* $x \leq 1$
by(*simp add*: *suminf-geometric*[*of 1/3*])
moreover have $0 \leq$ *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* $x \in \{0..1\}$
by *simp*

qed

lemma *Cantor-to-01-measurable*: *Cantor-to-01* ∈ *Cantor-space* →$_M$ *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 (λ*x*. *x n*) ∈ *Cantor-space* →$_M$ *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* / *3*) ^ *Suc n* ∗ *x n*) ∈ *borel-measurable Cantor-space*
   by *simp*
 qed
qed(*rule Cantor-to-01-image*)

lemma
 shows *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*})
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-topology.S-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*)

 interpret *m01*: *polish-metric-set* {*0*, *1*::*real*} λ*x y*. *if* (*x* = *0* ∨ *x* = *1*) ∧ (*y* = *0* ∨ *y* = *1*) *then* |*x* − *y*| *else 0*
 proof −
  have (λ*x y*. *if x* ∈ {*0*,*1*} ∧ *y* ∈ {*0*,*1*} *then* |*x* − *y*| *else 0*) = *discrete-dist* {*0*,*1*::*real*}
   by *standard*+ (*auto simp*: *discrete-dist-def*)
  moreover have *polish-metric-set* {*0*, *1*} ...
   by(*simp add*: *discrete-dist-polish-iff*)
  ultimately show *polish-metric-set* {*0*, *1*::*real*} (λ*x y*. *if* (*x* = *0* ∨ *x* = *1*) ∧ (*y* = *0* ∨ *y* = *1*) *then* |*x* − *y*| *else 0*) **by** *simp*
 qed

interpret *pm*: *product-polish-metric 1/3 UNIV* :: *nat set id id* $\lambda i.\ \{0,\ 1{::}real\}$ $\lambda i\ x\ y.$ *if* $(x = 0 \vee x = 1) \wedge (y = 0 \vee y = 1)$ *then* $|x - y|$ *else 0 1*
by(*auto intro*!: *product-polish-metric-natI simp*: *m01.polish-metric-set-axioms*)
have *product-topology* (λ-. *top-of-set* $\{0,\ 1\}$) *UNIV* = *pm.mtopology*
proof −
have *top-of-set* $\{0,\ 1\}$ = *m01.mtopology*
proof −
have *openin* (*top-of-set* $\{0,1\}$) $A \longleftrightarrow A \subseteq \{0,1\}$ **for** A :: *real set*
proof
assume $A \subseteq \{0,\ 1\}$
then consider $A = \{\} \mid A = \{0\} \mid A = \{1\} \mid A = \{0,1\}$
by *auto*
thus *openin* (*top-of-set* $\{0,\ 1\}$) A
by *cases* (*auto simp*: *openin-subtopology*)
qed (*rule openin-subset*[*of top-of-set* $\{0,\ 1\}$,*simplified*])
moreover have *openin m01.mtopology A* $\longleftrightarrow$ $A \subseteq \{0,1\}$ **for** A
proof
assume $A \subseteq \{0,\ 1\}$
then consider $A = \{\} \mid A = \{0\} \mid A = \{1\} \mid A = \{0,1\}$
by *auto*
thus *openin m01.mtopology A*
by *cases* (*auto simp*: *m01.mtopology-openin-iff m01.open-ball-def intro*!: *exI*[**where** *x=1*])
next
show *openin m01.mtopology A* $\Longrightarrow$ $A \subseteq \{0,\ 1\}$
using *m01.mtopology-topspace* **by**(*auto dest*: *openin-subset*)
qed
ultimately show *?thesis*
by(*simp add*: *topology-eq*)
qed
thus *?thesis*
using *pm.product-dist-mtopology* **by** *simp*
qed

interpret *real* : *polish-metric-set UNIV* :: *real set dist*
by *simp*
have [*simp*]: *real.mtopology* = *euclideanreal*
by (*simp add*: *euclidean-mtopology*)
interpret *m01′*: *polish-metric-set* $\{0..1{::}real\}$ *submetric* $\{0..1\}$ *dist*
by(*auto intro*!: *real.submetric-polish*)
have *restrict-space borel* $\{0..1\}$ = *borel-of m01′.mtopology*
by (*metis borel-of-euclidean borel-of-subtopology open-openin open-openin-set real.submetric-subtopology subset-UNIV topology-eq*)

have *pd-def*: *pm.product-dist x y* = $(\sum n.\ (1/3)\hat{}n * |x\ n - y\ n|)$ **if** $x \in$ *space Cantor-space* $y \in$ *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*)

have *sd-def*: *submetric* {*0..1*} (λ*x y*. |*x* − *y*|) (*Cantor-to-01 x*) (*Cantor-to-01 y*) = |*Cantor-to-01 x* − *Cantor-to-01 y*| **if** *x* ∈ *space Cantor-space y* ∈ *space Cantor-space* **for** *x y*
using *Cantor-to-01-image that* **by**(*auto simp*: *submetric-def*)
have *1*:|*Cantor-to-01 x* − *Cantor-to-01 y*| ≤ *pm.product-dist x y* (**is** *?lhs* ≤ *?rhs*) **if** *x* ∈ *space Cantor-space y* ∈ *space Cantor-space* **for** *x y*
proof −
have *?lhs* = |(∑ *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 ... = |∑ *n*. (*1*/*3*)^(*Suc n*) ∗ (*x n* − *y n*) |
by (*simp add*: *right-diff-distrib*)
also have ... ≤ (∑ *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 ... = (∑ *n*. (*1*/*3*)^(*Suc n*) ∗ |*x n* − *y n*|)
by (*simp add*: *abs-mult-pos mult.commute*)
also have ... ≤ *pm.product-dist x y*
unfolding *pd-def*[*OF that*]
apply(*rule suminf-le*)
using *Cantor-minus-abs-cantor*[*OF that*] **by** *auto*
finally show *?thesis* **.**
qed

have *2*:|*Cantor-to-01 x* − *Cantor-to-01 y*| ≥ *1* / *9* ∗*pm.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 *pm.dist-0*[*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 ≠ y i
      then have i ∈ {n. n ≤ n' ∧ x n ≠ y n}
        using ‹n ≤ n'› ‹i < n› by auto
      thus False
        using ‹i < n› linorder-class.Min-gr-iff[of {n. n ≤ n' ∧ x n ≠ y n} i] ‹x
n' ≠ y n'›
        by(auto simp: n-def)
    qed
  qed

  have u1: (1/3) ^ (Suc n) * (1/2) ≤ |Cantor-to-01 x - Cantor-to-01 y|
  proof -
    have (1/3) ^ (Suc n) * (1/2) ≤ |(∑ 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 ≤ (1 / 3) ^ Suc n - |(∑ m.
(1 / 3) ^ Suc (m + Suc n) * (y (m + Suc n) - x (m + Suc n)))|
      proof -
        have |(∑ m. (1 / 3) ^ Suc (m + Suc n) * (y (m + Suc n) - x (m + Suc
n)))| ≤ (1/3)^(n + 2) * 3/2
            (is ?lhs ≤ -)
        proof -
          have ?lhs ≤ (∑ 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 ... = (∑ m. (1 / 3) ^ Suc (m + Suc n) * |y (m + Suc n) -
x (m + Suc n)|)
            by(simp add: abs-mult)
          also have ... ≤ (∑ 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 ... = (∑ m. (1 / 3) ^ (m + Suc (Suc n)) * 1) by simp
          also have ... = (1/3)^(n + 2) * 3/(2::real)
            by(simp only: pm.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)| - |∑ m. (1 / 3) ^ Suc (m
+ Suc n) * (y (m + Suc n) - x (m + Suc n))|
        using ‹x n ≠ 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))
        apply(rule suminf-split-initial-segment)
        using that by 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) * pm.product-dist x y ≤ (1/3) ^ (Suc n) * (1/2)
  proof −
    have pm.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]] Can-
tor-minus-abs-cantor[OF that]
      by(auto intro!: suminf-le summable-ignore-initial-segment[of - n])
    also have ... = (1 / 3) ^ n * (3 / 2)
      using pm.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 pm.product-dist x y = 0
    using 2[OF h(1,2)] pm.dist-geq0[of x y]
    by simp
  thus x = y
    using pm.dist-0[of x y] h(1,2)
    by(simp add: space-Cantor-space)
 qed

 have closed: closedin m01'.mtopology (Cantor-to-01 ' (space Cantor-space))
  unfolding m01'.mtopology-closedin-iff
 proof safe
  show a ∈ space Cantor-space ⟹ Cantor-to-01 a ∈ {0..1} for a
    using Cantor-to-01-image by auto
 next
  fix f s
  assume h:f ∈ UNIV → Cantor-to-01 ' space Cantor-space m01'.converge-to-inS
f s
  then have m01'.Cauchy-inS f
   using m01'.Cauchy-if-convergent-inS by(auto simp: m01'.convergent-inS-def)
```

have $\forall n.\ \exists x{\in}space\ Cantor\text{-}space.\ f\ n = Cantor\text{-}to\text{-}01\ x$ **using** $h(1)$ **by** *auto*
then obtain x **where** $hx{:}\bigwedge n.\ x\ n \in space\ Cantor\text{-}space\ \bigwedge n.\ f\ n = Cantor\text{-}to\text{-}01\ (x\ n)$ **by** *metis*
have *pm.Cauchy-inS x*
unfolding *pm.Cauchy-inS-def2''*
proof
show $x \in UNIV \to (\Pi_E\ i{\in}\ UNIV.\ \{0{,}1\})$
using $hx(1)$ **by**(*auto simp*: *space-Cantor-space*)
next
show $\forall \varepsilon{>}0.\ \exists y{\in}UNIV \to_E \{0,\ 1\}.\ \exists N.\ \forall n{\geq}N.$ *pm.product-dist y* $(x\ n) < \varepsilon$
proof *safe*
fix ε
assume $(0 :: real) < \varepsilon$
hence $0 < \varepsilon\ /\ 9$ **by** *auto*
then obtain N' **where** $\forall n{\geq}N'.\ f\ n \in m01'.open\text{-}ball\ (f\ N')\ (\varepsilon\ /\ 9)$
using ‹*m01'.Cauchy-inS f*› *m01'.Cauchy-inS-def2*[*of f*] **by** *blast*
hence $\bigwedge n.\ n \geq N' \Longrightarrow |f\ N' - f\ n| < (\varepsilon\ /\ 9)$
using *m01'.Cauchy-inS-dest1*[*OF* ‹*m01'.Cauchy-inS f*›]
by(*auto simp*: *m01'.open-ball-def*) (*auto simp*: *submetric-def dist-real-def*)

thus $\exists y{\in}(\Pi_E\ i{\in}\ UNIV.\ \{0{,}1\}).\ \exists N.\ \forall n{\geq}N.$ *pm.product-dist y* $(x\ n) < \varepsilon$
using *order.strict-trans1*[*OF 2*[*OF hx(1)*[*of N'*] *hx(1)*],*of - ε/9*] *hx(1)*
by(*auto intro*!: *exI*[**where** $x{=}N'$] *bexI*[**where** $x{=}x\ N'$] *simp*: *hx(2) space-Cantor-space*)
qed
qed
then obtain y **where** *pm.converge-to-inS x y*
using *pm.convergence* **by**(*auto simp*: *pm.convergent-inS-def*)
hence $y \in space\ Cantor\text{-}space$
by(*auto simp*: *pm.converge-to-inS-def space-Cantor-space*)
have *m01'.converge-to-inS f* (*Cantor-to-01 y*)
unfolding *m01'.converge-to-inS-def2*
proof *safe*
show $f\ a \in \{0..1\}$ *Cantor-to-01 y* $\in \{0..1\}$ **for** a
using $h(1)$ *funcset-image*[*OF Cantor-to-01-image*]
by (*simp-all add*: $hx(1)$ $hx(2)$ *image-subset-iff pm.converge-to-inS-def* ‹$y \in$ *space Cantor-space*›)
next
fix ε
assume $(0 :: real) < \varepsilon$
then obtain N **where** $\bigwedge n.\ n \geq N \Longrightarrow$ *pm.product-dist* $(x\ n)\ y < \varepsilon$
using ‹*pm.converge-to-inS x y*› **by**(*auto simp*: *pm.converge-to-inS-def2*) *meson*
thus $\exists N.\ \forall n{\geq}N.$ *submetric* $\{0..1\}$ *dist* $(f\ n)$ (*Cantor-to-01 y*) $< \varepsilon$
by(*auto intro*!: *exI*[**where** $x{=}N$] *order.strict-trans1*[*OF 1*[*OF hx(1)* ‹$y \in$ *space Cantor-space*›]] *simp*: *submetric-def* ‹$0 < \varepsilon$› $hx(2)$ *dist-real-def*)
qed
hence *Cantor-to-01 y* $= s$

using *h*(2) *m01'.converge-to-inS-unique* **by** *blast*
with ‹*y* ∈ *space Cantor-space*› **show** *s* ∈ *Cantor-to-01* ‘ *space Cantor-space*
by *auto*
qed

have *open-map*:*open-map pm.mtopology* (*subtopology m01'.mtopology* (*Cantor-to-01* ‘ (*space Cantor-space*))) *Cantor-to-01*
unfolding *space-Cantor-space*
proof(*rule metric-set-opem-map-from-dist*[*OF pm.metric-set-axioms m01'.metric-set-axioms Cantor-to-01-image*[*simplified space-Cantor-space*]])
fix *x ε*
assume *x* ∈ (*UNIV* :: *nat set*) →$_E$ {0, 1::*real*} (0 :: *real*) < *ε*
show ∃ *δ*>0. ∀ *y*∈*UNIV* →$_E$ {0, 1}. *submetric* {0..1} *dist* (*Cantor-to-01 x*) (*Cantor-to-01 y*) < *δ* ⟶ *pm.product-dist x y* < *ε*
proof(*safe intro*!: *exI*[**where** *x*=*ε*/9])
fix *y*
assume *h*:*y* ∈ (*UNIV* :: *nat set*) →$_E$ {0, 1::*real*}
submetric {0..1} *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* →$_E$ {0, 1}› **by**(*simp-all add*: *space-Cantor-space*)
have |*Cantor-to-01 x* − *Cantor-to-01 y*| < *ε* / 9
using *sd-def*[*OF sc*] *h*(2) **by** (*metis dist-real-def submetric-def*)
with 2[*OF sc*] **show** *pm.product-dist x y* < *ε*
by *simp*
qed (*use* ‹*ε* > 0› **in** *auto*)
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 pm.mtopology m01'.mtopology Cantor-to-01 A*] *borel-of-closed*[*OF closed*] ‹*product-topology* (λ-. *top-of-set* {0, 1}) *UNIV* = *pm.mtopology*› ‹*restrict-space borel* {0..1} = *borel-of m01'.mtopology*› *inj pm.mtopology-topspace that space-Cantor-space open-map sets-Cantor*
by *auto*

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 $0,1^{\mathbb{N}}$.

definition *to-Cantor-from-01* :: *real* ⇒ *nat* ⇒ *real* **where**
to-Cantor-from-01 ≡ (λ*r 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.

lemma *to-Cantor-from-01-image'*: *to-Cantor-from-01 r n* ∈ {0,1}

unfolding *to-Cantor-from-01-def* **by** *auto*

lemma *to-Cantor-from-01-image''*: $0 \leq$ *to-Cantor-from-01 r n to-Cantor-from-01 r n* ≤ 1
using *to-Cantor-from-01-image'*[*of r n*] **by** *auto*

lemma *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*)

lemma *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*)

lemma *to-Cantor-from-01-summable*[*simp*]:
summable ($\lambda n.$ $(1/2)\hat{}n *$ *to-Cantor-from-01 r n*)
proof(*rule summable-comparison-test'*[**where** $g{=}\lambda n.$ $(1/2)\hat{}$ *n*])
show *norm* $((1 / 2) \hat{} n *$ *to-Cantor-from-01 r n*$) \leq (1 / 2) \hat{} n$ **for** *n*
using *to-Cantor-from-01-image'*[*of r n*] **by** *auto*
qed *simp*

lemma *to-Cantor-from-sumn'*:
assumes $r \in \{0..<1\}$
shows $(\sum i{<}n.$ $(1/2)\hat{}(Suc\ i)*$*to-Cantor-from-01 r i*$) \leq r$
and $r - (\sum i{<}n.$ $(1/2)\hat{}(Suc\ i)*$*to-Cantor-from-01 r i*$) < (1/2)\hat{}n$
and *to-Cantor-from-01 r n* $= 1 \longleftrightarrow (1/2)\hat{}(Suc\ n) \leq r - (\sum i{<}n.$ $(1/2)\hat{}(Suc\ i)*$*to-Cantor-from-01 r i*)
and *to-Cantor-from-01 r n* $= 0 \longleftrightarrow r - (\sum i{<}n.$ $(1/2)\hat{}(Suc\ i)*$*to-Cantor-from-01 r i*$) < (1/2)\hat{}(Suc\ n)$
proof −
let *?f* $=$ *to-Cantor-from-01 r*
have *f-simp*: *?f l* $=$ *real-of-int* $(\lfloor 2\hat{}(Suc\ l) * r \rfloor\ mod\ 2)$ **for** *l*
using *assms* **by**(*simp add*: *to-Cantor-from-01-def*)
define *S* **where** $S = (\lambda n.$ $\sum i{<}n.$ $(1/2)\hat{}(Suc\ i)*$*?f i*)
have *SSuc*:*S* $(Suc\ k) = S\ k + (1/2)\hat{}(Suc\ k) *$ *to-Cantor-from-01 r k* **for** *k*
by(*simp add*: *S-def*)
have *Sfloor*: $\lfloor 2\hat{}(Suc\ m) * (l - S\ m) \rfloor\ mod\ 2 = \lfloor 2\hat{}(Suc\ m) * l \rfloor\ mod\ 2$ **for** *l m*
proof −
have $\exists z.$ $2\hat{}(Suc\ m) * ((1/2)\hat{}(Suc\ k) *$ *?f k*$) =$ *2*$*$*real-of-int z* **if** $k < m$ **for** *k*
proof −
have *0*:(*2*::*real*) $\hat{}\ m * (1 / 2) \hat{}\ k = 2 * 2\hat{}(m{-}k{-}1)$
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** $\bigwedge k.$ $k < m \Longrightarrow 2\hat{}(Suc\ m) * ((1/2)\hat{}(Suc\ k) *$ *?f k*$) =$ *2*$*$*real-of-int* $(z\ k)$

```
    by metis
  hence Sm: 2^(Suc m) * S m = real-of-int (2 * (∑ k<m. (z k)))
    by(auto simp: S-def sum-distrib-left)
  have ⌊2^(Suc m) * (l - S m)⌋ mod 2 = ⌊2^(Suc m) * l - 2^(Suc m) *  S m⌋
mod 2
    by (simp add: right-diff-distrib)
  also have ... = ⌊2^(Suc m) * l⌋ mod 2
    unfolding Sm
    by(simp only: floor-diff-of-int) presburger
  finally show ?thesis .
 qed

 have S n ≤ r ∧ r - S n < (1/2)^n ∧ (?f n = 1 ⟷ (1/2)^(Suc n) ≤ r - S
n) ∧ (?f n = 0 ⟷ r - S n < (1/2)^(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 ≤ r r - S n < (1 / 2) ^ n
          ?f n = 1 ⟹ (1 / 2) ^ Suc n ≤ r - S n
          ?f n = 0 ⟹ r - S n < (1 / 2) ^ Suc n
    by simp-all
   have SSuc':?f n = 0 ∧ S (Suc n) = S n ∨ ?f n = 1 ∧ S (Suc n) = S n +
(1/2)^(Suc n)
    using to-Cantor-from-01-image'[of r n] by(simp add: SSuc)
  have goal1:S (Suc n) ≤ r
    using SSuc' ih(1) ih(3) by auto
  have goal2: r - S (Suc n) < (1 / 2) ^ Suc n
    using SSuc' ih(4) ih(2) by auto
  have goal3-1: (1 / 2) ^ Suc (Suc n) ≤ r - S (Suc n) if ?f (Suc n) = 1
  proof(rule ccontr)
    assume ¬ (1 / 2) ^ Suc (Suc n) ≤ r - S (Suc n)
    then have r - S (Suc n) < (1 / 2) ^ Suc (Suc n) by simp
    hence 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)]
      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 False
      using that[simplified f-simp] Sfloor[of Suc n r]
      by fastforce
  qed
  have goal3-2: ?f (Suc n) = 1 if (1 / 2) ^ Suc (Suc n) ≤ r - S (Suc n)
  proof -
```

```
    have 1 ≤ 2 ^ Suc (Suc n) * (r - S (Suc n))
      using that[simplified f-simp] 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 ?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

lemma *to-Cantor-from-sumn*:
assumes *r* ∈ {*0*..*1*}
shows (∑ *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*)
proof −
have *nsum*:(∑ *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 (∑ *i*<*n*. (*1*/*2*)^(*Suc i*)*to-Cantor-from-01 r *i*) ≤ *r* ∧ *r* − (∑ *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* − (∑ *i*<*n*. (*1*/*2*)^(*Suc i*)*to-Cantor-from-01 r *i*)) ∧ (*to-Cantor-from-01 r n* = *0* ⟷ *r* − (∑ *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*,*of n*]
show *?thesis*
by *auto*
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*
qed

lemma *to-Cantor-from-sum*:
assumes *r* ∈ {*0*..*1*}
shows (∑ *n*. (*1*/*2*)^(*Suc n*)*to-Cantor-from-01 r *n*) = *r*
proof −
have *1*:*r* ≤ (∑ *n*. (*1*/*2*)^(*Suc n*)*to-Cantor-from-01 r *n*)

```
  proof −
    have 0:r ≤ (1 / 2) ^ n + (∑ n. (1/2)^(Suc n)*to-Cantor-from-01 r n) for n
    proof −
      have r ≤ (1 / 2) ^ n + (∑ 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 + (∑ 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=λn. (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

lemma to-Cantor-from-sum':
  assumes r ∈ {0..1}
  shows (∑ 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))
  using suminf-minus-initial-segment[of λn. (1 / 2) ^ Suc n * to-Cantor-from-01
r n n] to-Cantor-from-sum[OF assms]
  by auto

lemma to-Cantor-from-01-exist0:
  assumes r ∈ {0..<1}
  shows ∀ n.∃ k≥n. to-Cantor-from-01 r k = 0
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' ≠ 0
      then have to-Cantor-from-01 r n' = 1
        using to-Cantor-from-01-image'[of r n'] by auto
      hence n' ∈ {i. i ≤ n0 ∧ (∀ k≥i. to-Cantor-from-01 r k = 1)}
        using hn eqn not-less-eq-eq order-antisym-conv by fastforce
      hence n ≤ n'
       using Min.coboundedI[of {i. i ≤ n0 ∧ (∀ k≥i. to-Cantor-from-01 r k = 1)}
n']
        by(simp add: n-def)
      thus False
        using eqn by simp
    qed
    hence le1:r - (∑ 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 - (∑ i<n'. (1 / 2) ^ Suc i * to-Cantor-from-01 r i) = (1 / 2) ^ n
        (is ?lhs = -)
    proof -
      have ?lhs = (∑ 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 ... = (∑ m. (1/2)^(m + Suc n)*to-Cantor-from-01 r (m + n))
      proof -
        have (∑ n. (1 / 2) ^ (Suc n + Suc n') * to-Cantor-from-01 r (Suc n +
n')) = (∑ 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 ... = (∑ 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

lemma to-Cantor-from-01-if-exist0:
  assumes ⋀n. a n ∈ {0,1} ∀n.∃k≥n. a k = 0
  shows to-Cantor-from-01 (∑ n. (1 / 2) ^ Suc n * a n) = a
proof
  fix n
  have [simp]: summable (λn. (1 / 2) ^ n * a n)
  proof(rule summable-comparison-test'[where g=λn. (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 λn. (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 λn. (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
        apply(rule suminf-nonneg,rule summable-ignore-initial-segment)
        using assms(1) space-Cantor-space-01[of a,simplified space-Cantor-space]
        by 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

lemma to-Cantor-from-01-sum-of-to-Cantor-from-01:
 assumes r ∈ {0..1}
 shows to-Cantor-from-01 (∑ n. (1 / 2) ^ Suc n * to-Cantor-from-01 r n) =
to-Cantor-from-01 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

lemma 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

lemma to-Cantor-from-01-preserves-sets:
 assumes A ∈ sets (restrict-space borel {0..1})
 shows to-Cantor-from-01 ‘ A ∈ sets Cantor-space
proof −
 define f :: (nat ⇒ real) ⇒ real where f ≡ (λx. ∑ n. (1/2)^(Suc n)* x n)
```

have *f-meas*:$f \in$ *Cantor-space* $\rightarrow_M$ *restrict-space borel* $\{0..1\}$
proof −
have $f \in$ *borel-measurable Cantor-space*
unfolding *Cantor-to-01-def f-def*
proof(*rule borel-measurable-suminf*)
fix *n*
have $(\lambda x.\ x\ n) \in$ *Cantor-space* $\rightarrow_M$ *restrict-space borel* $\{0,\ 1\}$
by(*simp add*: *Cantor-space-def*)
hence $(\lambda x.\ x\ n) \in$ *borel-measurable Cantor-space*
by(*simp add*: *measurable-restrict-space2-iff*)
thus $(\lambda x.\ (1\ /\ 2)\ \widehat{}\ Suc\ n * x\ n) \in$ *borel-measurable Cantor-space*
by *simp*
qed
moreover have $0 \leq f\ x\ f\ x \leq 1$ **if** $x \in$ *space Cantor-space* **for** *x*
proof −
have [*simp*]:*summable* $(\lambda n.\ (1/2)\widehat{}n * x\ n)$
proof(*rule summable-comparison-test'*[**where** g=$\lambda n.\ (1/2)\widehat{}\ n$])
show *norm* $((1\ /\ 2)\ \widehat{}\ n * x\ n) \leq (1\ /\ 2)\ \widehat{}\ n$ **for** *n*
using *that* **by** *simp*
qed *simp*
show $0 \leq f\ x$
using *that* **by**(*auto intro*!: *suminf-nonneg simp*: *f-def*)
show $f\ x \leq 1$
proof −
have $f\ x \leq (\sum n.\ (1/2)\widehat{}(Suc\ n))$
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* $\{0..1\}$)) $\in$ *sets Cantor-space*
(**is** *?A* $\in$ -)
proof −
have *?A* $\subseteq$ *space Cantor-space*
using *to-Cantor-from-01-image* **by** *auto*
have *comple-sets*:$(\Pi_E\ i \in UNIV.\ \{0,1\})$ − *?A* $\in$ *sets Cantor-space*
proof −
have *eq1*:*?A* = $\{\lambda n.\ 1\} \cup \{x.\ (\forall n.\ x\ n \in \{0,1\}) \wedge (\forall n.\ \exists k \geq n.\ x\ k = 0)\}$
proof
show *?A* $\subseteq \{\lambda n.\ 1\} \cup \{x.\ (\forall n.\ x\ n \in \{0,\ 1\}) \wedge (\forall n.\ \exists k \geq n.\ x\ k = 0)\}$
proof
fix *x*
assume $x \in$ *?A*
then obtain *r* **where** *hr*:$r \in \{0..1\}$ 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 {λn. 1} ∪ {x. (∀ n. x n ∈ {0, 1}) ∧ (∀ n. ∃ k≥n. x k = 0)} ⊆ ?A
  proof
    fix x :: nat ⇒ real
    assume x ∈ {λn. 1} ∪ {x. (∀ n. x n ∈ {0,1}) ∧ (∀ n. ∃ k≥n. x k = 0)}
    then consider x = (λn. 1) | (∀ n. x n ∈ {0,1}) ∧ (∀ n. ∃ k≥n. x k = 0)
      by auto
    thus x ∈ ?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 ⋀n. 0 ≤ x n ⋀n. x n ≤ 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=∑ n. (1 / 2) ^ Suc n * x n])
    qed
  qed
qed
have (Π_E i∈ UNIV. {0,1}) - ?A = {x. (∀ n. x n ∈ {0,1}) ∧ (∃ n. ∀ k≥n. x
k = 1)} - {λn. 1}
proof
  show (Π_E i∈ UNIV. {0,1}) - ?A ⊆ {x. (∀ n. x n ∈ {0,1}) ∧ (∃ n. ∀ k≥n.
x k = 1)} - {λn. 1}
  proof
    fix x :: nat ⇒ real
    assume x ∈ (Π_E i∈ UNIV. {0,1}) - ?A
    then have ∀ n. x n ∈ {0,1} ¬ (∀ n. ∃ k≥n. x k = 0) x ≠ (λn. 1)
      using eq1 by blast+
    thus x ∈ {x. (∀ n. x n ∈ {0,1}) ∧ (∃ n. ∀ k≥n. x k = 1)} - {λn. 1}
      by blast
  qed
next
```

show (Π_E *i*$\in$ *UNIV*. {*0*,*1*}) − *?A* ⊇ {*x*. (∀ *n*. *x n* ∈ {*0*,*1*}) ∧ (∃ *n*. ∀ *k*≥*n*. *x k* = *1*)} − {λ*n*. *1*}
proof
fix *x* :: *nat* ⇒ *real*
assume *h*:*x* ∈ {*x*. (∀ *n*. *x n* ∈ {*0*,*1*}) ∧ (∃ *n*. ∀ *k*≥*n*. *x k* = *1*)} − {λ*n*. *1*}
then have ∀ *n*. *x n* ∈ {*0*,*1*} ∃ *n*. ∀ *k*≥*n*. *x k* = *1* *x* ≠ (λ*n*. *1*)
by *blast*+
hence ¬ (∀ *n*. ∃ *k*≥*n*. *x k* = *0*)
by *fastforce*
with ‹∀ *n*. *x n* ∈ {*0*,*1*}› ‹*x* ≠ (λ*n*. *1*)›
show *x* ∈ (Π_E *i*$\in$ *UNIV*. {*0*,*1*}) − *?A*
using *eq1* **by** *blast*
qed
qed
also have ... = (⋃ ((λ*n*. {*x*. (∀ *n*. *x n* ∈ {*0*,*1*}) ∧ (∀ *k*≥*n*. *x k* = *1*)}) ‘ *UNIV*)) − {λ*n*. *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* (λ*n*. *if n* < *m then x n else 1*)
by(*auto simp add*: *g-def PiE-def extensional-def*)
moreover have (λ*n*. *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

lemma Cantor-space-isomorphic-to-01closed:
 Cantor-space measurable-isomorphic (restrict-space borel {0..1::real})
 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)

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

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-topology S sets* (*borel-of S*) = *sets M*
using *polish-topology* **by** *blast*
obtain *A* **where** *A*:*g-delta-of Hilbert-cube-as-topology A S homeomorphic-space subtopology Hilbert-cube-as-topology A*
using *polish-topology.embedding-into-Hilbert-cube-g-delta-of*[*OF S*(*1*)] **by** *blast*
show *?thesis*
using *borel-of-g-delta-of*[*OF A*(*1*)] *homeomorphic-space-measurable-isomorphic*[*OF A*(*2*)] *measurable-isomorphic-sets-cong*[*OF S*(*2*),*of borel-of* (*subtopology Hilbert-cube-as-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*) *uncountable-contains-Cantor-space*:
assumes *uncountable* (*space M*)
shows ∃ *A* ∈ *sets M*. *Cantor-space measurable-isomorphic* (*restrict-space M A*)
proof −
obtain *S* **where** *S*:*polish-topology S sets* (*borel-of S*) = *sets M*
using *polish-topology* **by** *blast*
then obtain *A* **where** *A*:*g-delta-of S A Cantor-space-as-topology homeomorphic-space subtopology S A*
using *polish-topology.uncountable-contains-Cantor-space*[*of S*] *assms sets-eq-imp-space-eq*[*OF S*(*2*)]
by(*auto simp*: *space-borel-of*)
show *?thesis*
using *borel-of-g-delta-of*[*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

lemma(**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 uncountable-contains-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

lemma(**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

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* →$_M$ *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* →$_M$ *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 →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 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(cases space M = {})
  case True
  then show ?thesis
   by (meson countable-empty measurable-isomorphic-empty sets.empty-sets space-restrict-space2
that)
next
  case nin:False
  define A where A ≡ to-real ' (space M)
  have countable A A ∈ sets borel M measurable-isomorphic (restrict-space borel
A)
  proof −
    show countable A A ∈ sets borel
     using assms(1) standard-borel.countable-sets[of borel A] standard-borel-ne-borel
by(auto simp: A-def standard-borel-ne-def)
    show M measurable-isomorphic restrict-space borel A
      using from-real-to-real A-def ‹A ∈ sets borel›
    by(auto intro!: measurable-isomorphic-byWitness[OF measurable-restrict-space2[OF
- to-real-measurable] - measurable-restrict-space1[OF from-real-measurable'[OF nin]]])
  qed
  with that show ?thesis
    by auto
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* →$_M$ *M*
by(*simp add*: *from-real-measurable′ space-ne*)

end

4.3 Example: The Metric Space of Continuous Functions

theory *Space-of-Continuous-Maps*
imports *StandardBorel*
begin

definition *cmaps* :: [*'a topology*, *'b topology*] ⇒ (*'a* ⇒ *'b*) *set* **where**
cmaps X Y ≡ {*f*. *f* ∈ *extensional* (*topspace X*) ∧ *continuous-map X Y f*}

definition *cmaps-dist* :: [*'a topology*, *'b topology*, *'b* ⇒ *'b* ⇒ *real*, *'a* ⇒ *'b*, *'a* ⇒ *'b*] ⇒ *real* **where**
cmaps-dist X Y d f g ≡ *if f* ∈ *cmaps X Y* ∧ *g* ∈ *cmaps X Y* ∧ *topspace X* ≠ {} *then* (⨆ {*d* (*f x*) (*g x*) |*x*. *x* ∈ *topspace X*}) *else 0*

lemma *cmaps-X-empty*:
assumes *topspace X* = {}
shows *cmaps X Y* = {λ*x*. *undefined*}
by(*auto simp*: *cmaps-def assms simp flip*: *null-topspace-iff-trivial*)

lemma *cmaps-Y-empty*:
assumes *topspace X* ≠ {} *topspace Y* = {}
shows *cmaps X Y* = {}
by(*auto simp*: *cmaps-def assms continuous-map-def Pi-def simp flip*: *null-topspace-iff-trivial*)

lemma *cmaps-dist-X-empty*:
assumes *topspace X* = {}
shows *cmaps-dist X* = (λ*Y d f g*. *0*)
by *standard*+ (*auto simp*: *cmaps-dist-def assms*)

lemma *cmaps-dist-Y-empty*:
assumes *topspace X* $\neq \{\}$ *topspace Y* $= \{\}$
shows *cmaps-dist X Y* $= (\lambda d\ f\ g.\ 0)$
by *standard*+ (*auto simp*: *cmaps-dist-def assms cmaps-Y-empty*)

4.3.1 Definition

context *metric-set*
begin

context
fixes *K* **and** *X* :: *'b topology*
assumes *m-bounded* :$\bigwedge x\ y.\ dist\ x\ y \leq K$
begin

lemma *cm-dest*:
shows $\bigwedge f\ x.\ f \in (cmaps\ X\ mtopology) \Longrightarrow x \in topspace\ X \Longrightarrow f\ x \in S$
and $\bigwedge f\ x.\ f \in (cmaps\ X\ mtopology) \Longrightarrow x \notin topspace\ X \Longrightarrow f\ x = undefined$
and $\bigwedge f.\ f \in (cmaps\ X\ mtopology) \Longrightarrow continuous\text{-}map\ X\ mtopology\ f$
using *continuous-map-image-subset-topspace*[*of X mtopology*,*simplified mtopology-topspace*]
by(*auto simp*: *cmaps-def extensional-def*)

lemma *cmaps-dist-bdd-above*[*simp*]: *bdd-above* $\{dist\ (f\ x)\ (g\ x)\ |x.\ x \in A\}$
using *m-bounded* **by**(*auto intro*!: *bdd-aboveI*[**where** *M*=*K*])

lemma *cmaps-metric-set*: *metric-set* (*cmaps X mtopology*) (*cmaps-dist X mtopology dist*)
proof(*cases topspace X* $= \{\}$)
case *True*
then show *?thesis*
by(*simp add*: *singleton-metric cmaps-X-empty cmaps-dist-X-empty*)
next
case *h*:*False*
then have *nen*[*simp*]:$\{dist\ (f\ x)\ (g\ x)|x.\ x \in topspace\ X\} \neq \{\}$ **for** *f g*
by *auto*
show *?thesis*
proof
show (*cmaps-dist X mtopology dist*) *f g* ≥ 0 **for** *f g*
by(*auto simp*: *cmaps-dist-def dist-geq0 intro*!: *cSup-upper2*[**where** *x*=*dist - -*]
simp flip: *null-topspace-iff-trivial*)
next
fix *f g*
assume *f* $\notin$ (*cmaps X mtopology*)
then show (*cmaps-dist X mtopology dist*) *f g* $= 0$
by(*simp add*: *cmaps-dist-def*)
next
show (*cmaps-dist X mtopology dist*) *f g* $=$ (*cmaps-dist X mtopology dist*) *g f*
for *f g*

```
    by(simp add: cmaps-dist-def dist-sym)
  next
    fix f g
    assume fg:f ∈ (cmaps X mtopology) g ∈ (cmaps X mtopology)
    show f = g ⟷ (cmaps-dist X mtopology dist) f g = 0
    proof safe
      have {dist (g x) (g x) |x. x ∈ topspace X} = {0}
        using h by fastforce
      thus (cmaps-dist X mtopology dist) g g = 0
        by(simp add: cmaps-dist-def)
    next
      assume (cmaps-dist X mtopology dist) f g = 0
      with fg h have ⨆ {dist (f x) (g x)|x. x ∈ topspace X} ≤ 0
        by(auto simp: cmaps-dist-def)
      hence ⋀x. x ∈ topspace X ⟹ dist (f x) (g x) ≤ 0
        by(auto simp: cSup-le-iff[OF nen])
      from antisym[OF this dist-geq0] have fgeq:⋀x. x ∈ topspace X ⟹ f x = g x
        using dist-0[OF cm-dest(1)[OF fg(1)] cm-dest(1)[OF fg(2)]] by auto
      show f = g
      proof
        fix x
        show f x = g x
          by(cases x ∈ topspace X,insert fg) (auto simp: cm-dest fgeq)
      qed
    qed
  next
    fix f g h
    assume fgh: f ∈ (cmaps X mtopology) g ∈ (cmaps X mtopology) h ∈ (cmaps
X mtopology)
    show (cmaps-dist X mtopology dist) f h ≤ (cmaps-dist X mtopology dist) f g +
(cmaps-dist X mtopology dist) g h (is ?lhs ≤ ?rhs)
    proof −
      have bdd1:bdd-above {dist (f x) (g x) + dist (g x) (h x) | x. x ∈ topspace X}
          using add-mono[OF m-bounded m-bounded] by(auto simp: bdd-above-def
intro!: exI[where x=K + K])
      have nen1:{dist (f x) (g x) + dist (g x) (h x) |x. x ∈ topspace X} ≠ {}
        using h by auto
      have ?lhs ≤ (⨆ {dist (f x) (g x) + dist (g x) (h x)|x. x ∈ topspace X})
      proof −
        {
          fix x
          assume x ∈ topspace X
          hence ∃ z. (∃ x. z = dist (f x) (g x) + dist (g x) (h x) ∧ x ∈ topspace X)
∧ dist (f x) (h x) ≤ z
          by(auto intro!: exI[where x=dist (f x) (g x) + dist (g x) (h x)] exI[where
x=x] dist-tr cm-dest fgh)
        }
        thus ?thesis
            by(auto simp: cmaps-dist-def fgh h intro!: cSup-mono bdd1 simp flip:
```

null-topspace-iff-trivial)
qed
also have ... ≤ *?rhs*
by(*auto simp*: *cSup-le-iff*[*OF nen1 bdd1*] *cmaps-dist-def fgh h intro*!: *add-mono cSup-upper*)
finally show *?thesis* .
qed
qed
qed

end

end

4.3.2 Topology of Uniform Convergence

locale *topology-of-uniform-convergence-c* = *complete-metric-set* + *compact-metrizable X* **for** *X*
+ **fixes** *K*
assumes *d-bounded*: ⋀*x y*. *dist x y* ≤ *K*
begin

lemmas *cm-dist-bdd-above*[*simp*] = *cmaps-dist-bdd-above*[*OF d-bounded*]

abbreviation *cm* ≡ *cmaps X mtopology*
abbreviation *cm-dist* ≡ *cmaps-dist X mtopology dist*

lemma *cm-complete-metric-set*: *complete-metric-set cm cm-dist*
proof −
interpret *m*: *metric-set cm cm-dist*
by(*auto intro*!: *cmaps-metric-set d-bounded*)
show *?thesis*
proof
obtain *dx* **where** *dx*: *compact-metric-set* (*topspace X*) *dx metric-set.mtopology* (*topspace X*) *dx* = *X*
by(*rule compact-metric*)
interpret *dx*: *compact-metric-set topspace X dx*
by *fact*
fix *fn*
assume *h*:*m.Cauchy-inS fn*
note *fn-cm* = *m.Cauchy-inS-dest1*[*OF this*]
have *c*:∃ *N*. ∀ *n*≥*N*. ∀ *m*≥*N*. ∀ *x*∈*topspace X*. *dist* (*fn n x*) (*fn m x*) < *e* **if** *e*:*e* > *0* **for** *e*
proof −
obtain *N* **where** *N*:⋀*n m*. *n* ≥ *N* ⟹ *m* ≥ *N* ⟹ *cm-dist* (*fn n*) (*fn m*) < *e*
by(*metis e h m.Cauchy-inS-def*)
show *?thesis*
proof(*safe intro*!: *exI*[**where** *x*=*N*])

```
      fix n m x
      assume nmx: n ≥ N m ≥ N x ∈ topspace X
      then have dist (fn n x) (fn m x) ≤ cm-dist (fn n) (fn m)
        using fn-cm by(auto simp: cmaps-dist-def intro!: cSup-upper)
      also have ... < e
        by(auto intro!: N nmx)
      finally show dist (fn n x) (fn m x) < e .
    qed
  qed
  have convergent-inS (λn. fn n x) if x:x ∈ topspace X for x
    by (rule convergence,auto simp: Cauchy-inS-def,insert c x fn-cm)
     (auto simp: cmaps-def continuous-map-def mtopology-topspace, blast, meson)
  then obtain f where f:⋀x. x ∈ topspace X ⟹ converge-to-inS (λn. fn n x)
(f x)
    by(auto simp: convergent-inS-def) metis
  define f' where f' ≡ (λx∈topspace X. f x)
  have f':⋀x. x ∈ topspace X ⟹ converge-to-inS (λn. fn n x) (f' x)
    using f by(auto simp: f'-def)
  have cu:converges-uniformly (topspace X) S dist fn f'
  unfolding converges-uniformly-def[OF dx.metric-set-axioms metric-set-axioms]
  proof safe
    fix e :: real
    assume e:0 < e
    obtain N where N: ⋀n m x. n≥N ⟹ m≥N ⟹ x∈topspace X ⟹ dist
(fn n x) (fn m x) < e / 2
      by(metis c[OF half-gt-zero[OF e]])
    show ∃ N. ∀ n≥N. ∀ x∈topspace X. dist (fn n x) (f' x) < e
    proof(rule ccontr)
      assume ∄ N. ∀ n≥N. ∀ x∈topspace X. dist (fn n x) (f' x) < e
      with N obtain n x where nx: n ≥ N x ∈ topspace X e ≤ dist (fn n x) (f'
x)
        by (meson linorder-le-less-linear)
      from f'[OF this(2)] half-gt-zero[OF e]
      obtain N' where N':⋀n. n ≥ N' ⟹ dist (fn n x) (f' x) < e / 2
        by(metis converge-to-inS-def2)
      define N0 where N0 ≡ max N N'
      have N0 : N0 ≥ N N0 ≥ N' by(auto simp: N0-def)
      have e ≤ dist (fn n x) (f' x) by fact
      also have ... ≤ dist (fn n x) (fn N0 x) + dist (fn N0 x) (f' x)
        using f'[OF nx(2)] by(auto intro!: dist-tr simp: converge-to-inS-def)
      also have ... < e
        using N[OF nx(1) N0(1) nx(2)] N'[OF N0(2)] by auto
      finally show False ..
    qed
  qed(use f' converge-to-inS-def in auto)
  show m.convergent-inS fn
    unfolding m.convergent-inS-def m.converge-to-inS-def2
  proof(safe intro!: exI[where x=f'])
    have continuous-map dx.mtopology mtopology f'
```

```
    using fn-cm by(auto intro!: converges-uniformly-continuous[OF dx.metric-set-axioms
metric-set-axioms - cu] simp: cmaps-def,auto simp:  dx)
    thus f'-cm:f' ∈ cm
      by(auto simp: cmaps-def dx f'-def)
    fix e :: real
    assume e:0 < e
    obtain N where N:⋀n x. n ≥ N ⟹ x ∈ topspace X ⟹ dist (fn n x) (f'
x) < e / 2
    by(metis half-gt-zero[OF e] cu[simplified converges-uniformly-def[OF dx.metric-set-axioms
metric-set-axioms]])
    show ∃ N. ∀ n≥N. cm-dist (fn n) f' < e
    proof(safe intro!: exI[where x=N])
      fix n
      assume n:N ≤ n
      have cm-dist (fn n) f' ≤ e / 2
      proof(cases topspace X = {})
        case True
        then show ?thesis
       by(auto simp: order.strict-implies-order[OF e] cmaps-X-empty cmaps-dist-X-empty)
      next
        case False
        then have 1:{dist (fn n x) (f' x) |x. x ∈ topspace X} ≠ {} by auto
        hence cm-dist (fn n) f' = (⨆ {dist (fn n x) (f' x) |x. x ∈ topspace X})
          by(auto simp: f'-cm fn-cm cmaps-dist-def)
        also have ... ≤ e / 2
          by(simp only: cSup-le-iff[OF 1,simplified]) (insert N[OF n], auto intro!:
order.strict-implies-order)
        finally show ?thesis .
      qed
      also have ... < e
        using e by simp
      finally show cm-dist (fn n) f' < e .
    qed
  qed(use fn-cm in auto)
 qed
qed

end

locale topology-of-uniform-convergence = polish-metric-set + compact-metrizable
X for X
 + fixes K
assumes d-bounded: ⋀x y. dist x y ≤ K
begin

sublocale topology-of-uniform-convergence-c
 by (simp add: compact-metrizable-axioms complete-metric-set-axioms d-bounded
topology-of-uniform-convergence-c-axioms-def topology-of-uniform-convergence-c-def)
```

lemma *cm-polish-metric-set*: *polish-metric-set cm cm-dist*
proof −
consider *topspace X* = {} | *topspace X* ≠ {} *S* = {} | *topspace X* ≠ {} *S* ≠ {}
by *auto*
thus *?thesis*
proof *cases*
case *1*
then show *?thesis*
by(*simp add*: *singleton-metric-polish cmaps-X-empty cmaps-dist-X-empty*)
next
case *2*
then show *?thesis*
by(*simp add*: *empty-metric-polish cmaps-Y-empty*[*of - mtopology*,*simplified mtopology-topspace*] *cmaps-dist-Y-empty*[*of - mtopology*,*simplified mtopology-topspace*])
next
case *XS-nem*:*3*
interpret *m*: *complete-metric-set cm cm-dist*
by(*rule cm-complete-metric-set*)
show *?thesis*
proof
obtain *dx* **where** *dx*: *compact-metric-set* (*topspace X*) *dx metric-set.mtopology* (*topspace X*) *dx* = *X*
by(*rule compact-metric*)
interpret *dx*: *compact-metric-set topspace X dx*
by *fact*
have ∃ *B*. *finite B* ∧ *B* ⊆ *topspace X* ∧ *topspace X* = (⋃ *a*∈*B*. *dx.open-ball a* (*1 / Suc m*)) **for** *m*
using *dx.totally-boundedSD*[*OF dx.totally-bounded*,*of 1 / Suc m*] **by** *fastforce*
then obtain *Xm* **where** *Xm*: ⋀*m*. *finite* (*Xm m*) ⋀*m*. (*Xm m*) ⊆ *topspace X* ⋀*m*. *topspace X* = (⋃ *a*∈*Xm m*. *dx.open-ball a* (*1 / Suc m*))
by *metis*
have *Xm-nem*:⋀*m*. (*Xm m*) ≠ {}
using *XS-nem Xm*(*3*) **by** *fastforce*
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* = *S* ∧ (∀ *u*∈*U*. *diam u* < *1 / Suc l*)) **for** *l*
by(*rule Lindelof-diam*) *auto*
then obtain *U* **where** *U*: ⋀*l*. *countable* (*U l*) ⋀*l*. (⋃ (*U l*)) = *S* ⋀*l u*. *u* ∈ *U l* ⟹ *diam 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*: *diam* (*uli l i*) < *1 / Suc l* **for** *l i*
using *U*(*3*) **by** *auto*
have *uli-un*:*S* = (⋃ *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* ∈ *cm*. ∀ *x*∈*topspace X*. ∀ *y*∈*topspace X*. *dx x y* < *1 / Suc m*) ⟶ *dist* (*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*} →$_E$ *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*} →$_E$ *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*} →$_E$ *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*: ∃ *g*∈*Dmn m n*. ∀ *y*∈*Xm m*. *dist* (*f y*) (*g y*) < *e* **if** *f*:*f* ∈ *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* ≡ (λ*i*∈{..<*km m*}. *SOME j*. *f* (*xmk m i*) ∈ *uli l j*)
have *s1*:*s*∈{..<*km m*} →$_E$ *UNIV* **by**(*simp add*: *s-def*)
have *s2*:∀ *i*<*km m*. *f* (*xmk m i*) ∈ *uli l* (*s i*)
proof −
fix *i*
have *f* (*xmk m i*) ∈ *uli l* (*SOME j*. *f* (*xmk m i*) ∈ *uli l j*) **for** *i*

```
      proof(rule someI-ex)
        have xmk m i ∈ topspace X
          using Xm(2) xmk-into by auto
        hence f (xmk m i) ∈ S
          using f by(auto simp: Cmn-def cmaps-def continuous-map-def mtopol-
ogy-topspace)
        thus ∃ x. f (xmk m i) ∈ uli l x
          using uli-un by auto
      qed
      thus ?thesis
        by (auto simp: s-def)
    qed
    have fmnls:fmnls m n l s ∈ Cmn m n ∧ (∀ j<km m. fmnls m n l s (xmk m
j) ∈ uli l (s j))
      by(simp add: fmnls-def,rule someI[where x=f],auto simp: s2 f)
    show ∃ g∈Dmn m n. ∀ y∈Xm m. dist (f y) (g y) < e
    proof(safe intro!: bexI[where x=fmnls m n l s])
      fix y
      assume y:y ∈ 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 ∈ uli l (s i) fmnls m n l s y ∈ uli l (s i)
        using i(1) s2 fmnls by(auto simp: i(2)[symmetric])
      moreover have f y ∈ S fmnls m n l s y ∈ S
        using f fmnls y Xm(2)[of m] by(auto simp: Cmn-def cmaps-def contin-
uous-map-def mtopology-topspace)
      ultimately have ennreal (dist (f y) (fmnls m n l s y)) ≤ diam (uli l (s i))
        by(auto intro!: diam-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 dist (f y) (fmnls m n l s y) < e
        by(simp add: ennreal-less-iff[OF dist-geq0])
    qed(use in-Dmnl[OF s1 f s2] Dmn-def in auto)
  qed

  show ∃ U. countable U ∧ m.dense-set U
    unfolding m.dense-set-def2
  proof(safe intro!: exI[where x=⋃ n. (⋃ m. Dmn m n)])
    fix f and e :: real
    assume h:f ∈ cm 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 uniform-continuous-map (topspace X) dx S dist f
      using h by(auto intro!: dx.continuous-map-is-uniform[OF metric-set-axioms]
simp: cmaps-def dx)
```

```
        then obtain d where d:d > 0 ⋀x y. x∈topspace X ⟹ y∈topspace X
⟹ dx x y < d ⟹ dist (f x) (f y) < 1 / (Suc n)
        by(auto simp: uniform-continuous-map-def[OF dx.metric-set-axioms met-
ric-set-axioms]) (metis less-add-same-cancel2 linorder-neqE-linordered-idom of-nat-Suc
of-nat-less-0-iff zero-less-divide-1-iff zero-less-one)
        then obtain m where m:1 / Suc m < d
          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 d(2)[OF - - order.strict-trans[OF - order.strict-trans1[OF l m]]]
by(auto simp: Cmn-def h)
      qed
      then obtain m where m:f ∈ Cmn m n by auto
      obtain g where g:g∈Dmn m n ⋀y. y∈Xm m ⟹ dist (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. cm-dist f g < e
      proof(rule exI[where x=n])
        show ∃ m. ∃ g∈Dmn m n. cm-dist f g < e
        proof(intro exI[where x=m] bexI[OF - g(1)])
          have g-cm:g ∈ cm
            using g(1) Dmn-subset[of m n] by(auto simp: Cmn-def)
          hence cm-dist f g = (⨆ {dist (f x) (g x) |x. x ∈ topspace X})
           by(auto simp: cmaps-dist-def h XS-nem simp flip: null-topspace-iff-trivial)
          also have ... ≤ 3 * e / 4
          proof −
            have 1:{dist (f x) (g x) |x. x ∈ topspace X} ≠ {}
              using XS-nem by auto
            have 2:dist (f x) (g x) ≤ 3 * e / 4 if x:x ∈ topspace X for x
            proof −
              obtain y where y:y ∈ Xm m x ∈ dx.open-ball y (1 / real (Suc m))
                using Xm(3) x by auto
              hence ytop:y ∈ topspace X
                using Xm(2) by auto
              have dist (f x) (g x) ≤ dist (f x) (f y) + dist (f y) (g x)
                  using x g-cm h(1) ytop by(auto intro!: dist-tr simp: cmaps-def
continuous-map-def mtopology-topspace)
              also have ... ≤ dist (f x) (f y) + dist (f y) (g y) + dist (g y) (g x)
                  using x g-cm h(1) ytop by(auto intro!: dist-tr simp: cmaps-def
continuous-map-def mtopology-topspace)
              also have ... ≤ e / 4 + e / 4 + e / 4
              proof −
                have dxy: dx x y < 1 / Suc m dx y x < 1 / Suc m
                  using dx.open-ballD[OF y(2)] by(auto simp: dx.dist-sym)
                hence dist (f x) (f y) < 1 / (Suc n) dist (g y) (g x) < 1 / (Suc n)
                  using m x ytop g Dmn-subset[of m n] by(auto simp: Cmn-def)
                hence dist (f x) (f y) < e / 4 dist (g y) (g x) < e / 4
                  using n by auto
                thus ?thesis
```

```
            using g(2)[OF y(1)] by auto
          qed
          finally show dist (f x) (g x) ≤ 3 * e / 4 by simp
        qed
        show ?thesis
          using 2 by(auto simp only: cSup-le-iff[OF 1,simplified])
      qed
      also have ... < e
        using h by auto
      finally show cm-dist f g < e .
    qed
  qed
  thus ∃ y∈⋃ n m. Dmn m n. cm-dist f y < e
    by auto
 qed(use Dmn-subset c-Dmn Cmn-def in auto)
 qed
 qed
qed

end


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.