The Stone-Čech Compactification

Mike Stannett

June 5, 2024

Contents

Building on parts of HOL-Analysis, we provide mathematical components for work on the Stone-Čech compactification. The main concepts covered are: C^*-embedding, weak topologies and compactification, focusing in particular on the Stone-Čech compactification of an arbitrary Tychonov space X. Many of the proofs given here derive from those of Willard (*General*

Topology, 1970, Addison-Wesley) and Walker (*The Stone-Čech Compactification*, 1974, Springer-Verlag).

Using traditional topological proof strategies we define the evaluation and projection functions for product spaces, and show that product spaces carry the weak topology induced by their projections whenever those projections separate points both from each other and from closed sets.

In particular, we show that the evaluation map from an arbitrary Tychonov space X into βX is a dense C^*-embedding, and then verify the Stone-Čech Extension Property: any continuous map from X to a compact Hausdorff space K extends uniquely to a continuous map from βX to K.

theory *Stone-Cech*
  imports *HOL.Topological-Spaces*
    HOL.Set
    HOL−Analysis.Urysohn

begin

Concrete definitions of finite intersections and arbitrary unions, and their relationship to the Analysis.Abstract_Topology versions.

definition *finite-intersections-of* :: $'a\ set\ set \Rightarrow 'a\ set\ set$
  where *finite-intersections-of* $S = \{ (\bigcap F) \mid F .\ F \subseteq S \wedge finite'\ F \}$

definition *arbitrary-unions-of* :: $'a\ set\ set \Rightarrow 'a\ set\ set$
  where *arbitrary-unions-of* $S = \{ (\bigcup F) \mid F .\ F \subseteq S \}$

lemma *generator-imp-arbitrary-union*:
  shows $S \subseteq$ *arbitrary-unions-of* S
  unfolding *arbitrary-unions-of-def* **by** *blast*

lemma *finite-intersections-container*:
  shows $\forall\ s \in$ *finite-intersections-of* S . $\bigcup S \cap s = s$
  unfolding *finite-intersections-of-def* **by** *blast*

lemma *generator-imp-finite-intersection*:
  shows $S \subseteq$ *finite-intersections-of* S
  unfolding *finite-intersections-of-def* **by** *blast*

lemma *finite-intersections-equiv*:
  shows (*finite′ intersection-of* $(\lambda x.\ x \in S)$) $U \longleftrightarrow U \in$ *finite-intersections-of* S
  unfolding *finite-intersections-of-def intersection-of-def*
  by *auto*

lemma *arbitrary-unions-equiv*:
  shows (*arbitrary union-of* $(\lambda\ x\ .\ x \in S)$) $U \longleftrightarrow U \in$ *arbitrary-unions-of* S
  unfolding *arbitrary-unions-of-def union-of-def arbitrary-def*

by *auto*

Supplementary information about topological bases and the topologies they generate

definition *base-generated-on-by* :: $'a\ set \Rightarrow 'a\ set\ set \Rightarrow 'a\ set\ set$
where *base-generated-on-by* $X\ S = \{\ X \cap s \mid s\ .\ s \in$ *finite-intersections-of* $S\}$

definition *opens-generated-on-by* :: $'a\ set \Rightarrow 'a\ set\ set \Rightarrow 'a\ set\ set$
where *opens-generated-on-by* $X\ S =$ *arbitrary-unions-of* (*base-generated-on-by* X S)

definition *base-generated-by* :: $'a\ set\ set \Rightarrow 'a\ set\ set$
where *base-generated-by* $S =$ *finite-intersections-of* S

definition *opens-generated-by* :: $'a\ set\ set \Rightarrow 'a\ set\ set$
where *opens-generated-by* $S =$ *arbitrary-unions-of* (*base-generated-by* S)

lemma *generators-are-basic*:
shows $S \subseteq$ *base-generated-by* S
unfolding *base-generated-by-def finite-intersections-of-def*
by *blast*

lemma *basics-are-open*:
shows *base-generated-by* $S \subseteq$ *opens-generated-by* S
unfolding *opens-generated-by-def arbitrary-unions-of-def*
by *blast*

lemma *generators-are-open*:
shows $S \subseteq$ *opens-generated-by* S
using *generators-are-basic basics-are-open*
by *blast*

lemma *generated-topspace*:
assumes $T =$ *topology-generated-by* S
shows *topspace* $T = \bigcup S$
using *assms* **by** *simp*

lemma *base-generated-by-alt*:
shows *base-generated-by* $S =$ *base-generated-on-by* $(\bigcup S)$ S
unfolding *base-generated-by-def base-generated-on-by-def*
using *finite-intersections-container*[*of* S]
by *auto*

lemma *opens-generated-by-alt*:
shows *opens-generated-by* $S =$ *arbitrary-unions-of* (*finite-intersections-of* S)
unfolding *opens-generated-by-def base-generated-by-def*
by *simp*

lemma *opens-generated-unfolded*:

shows *opens-generated-by* $S = \{\bigcup\ A \mid A\ .\ A \subseteq \{\ \bigcap\ B \mid B\ .\ \mathit{finite}'\ B \wedge B \subseteq S\}\}$
apply (*simp add*: *opens-generated-by-alt*)
unfolding *finite-intersections-of-def arbitrary-unions-of-def*
by *blast*

lemma *opens-eq-generated-topology*:
shows *openin* (*topology-generated-by* S) $U \longleftrightarrow U \in$ *opens-generated-by* S
proof $-$
have *openin* (*topology-generated-by* S) = *arbitrary union-of finite′ intersection-of* ($\lambda x.\ x \in S$)
by (*metis generate-topology-on-eq istopology-generate-topology-on topology-inverse′*)
also have ... = *arbitrary union-of* ($\lambda\ U\ .\ U \in$ *finite-intersections-of* S)
using *finite-intersections-equiv*[*of* S] **by** *presburger*
also have ... = ($\lambda\ U\ .\ U \in$ *arbitrary-unions-of* (*finite-intersections-of* S))
using *arbitrary-unions-equiv*[*of finite-intersections-of* S] **by** *presburger*
finally show *?thesis*
using *opens-generated-by-alt* **by** *auto*
qed

1 C^*-embedding

abbreviation *continuous-from-to*
:: *′a topology* ⇒ *′b topology* ⇒ (*′a* ⇒ *′b*) *set* (*cts*[-, -])
where *continuous-from-to* $X\ Y \equiv \{\ f\ .$ *continuous-map* $X\ Y\ f\ \}$

abbreviation *continuous-from-to-extensional*
:: *′a topology* ⇒ *′b topology* ⇒ (*′a* ⇒ *′b*) *set* (cts_E[-, -])
where *continuous-from-to-extensional* $X\ Y \equiv$ (*topspace* $X \rightarrow_E$ *topspace* Y) $\cap$ *cts*$[X,Y]$

abbreviation *continuous-maps-from-to-shared-where* ::
′a topology ⇒ (*′b topology* ⇒ *bool*) ⇒ (*′a* ⇒ *′b*) *set* ⇒ *bool* (*cts′-on - to′-shared* -)
where *continuous-maps-from-to-shared-where* $X\ P$
$\equiv$ (λ *fs* . ($\exists\ Y\ .\ P\ Y \wedge$ *fs* $\subseteq$ *cts*$[X,Y]$))

definition *dense-in* :: *′a topology* ⇒ *′a set* ⇒ *′a set* ⇒ *bool*
where *dense-in* $T\ A\ B \equiv T$ *closure-of* $A = B$

lemma *dense-in-closure*:
assumes *dense-in* $T\ A\ B$
shows *dense-in* (*subtopology* $T\ B$) $A\ B$
by (*metis Int-UNIV-right Int-absorb Int-commute assms closure-of-UNIV closure-of-restrict*
closure-of-subtopology dense-in-def topspace-subtopology)
abbreviation *dense-embedding* :: *′a topology* ⇒ *′b topology* ⇒ (*′a* ⇒ *′b*) ⇒ *bool*
where *dense-embedding small big f* $\equiv$ (*embedding-map small big f*)

$\wedge$ *dense-in big* (*f*'*topspace small*) (*topspace big*)

lemma *continuous-maps-on-dense-subset*:
assumes (*cts-on X to-shared Hausdorff-space*) {*f*,*g*}
and *dense-in X D* (*topspace X*)
and $\forall$ *x* $\in$ *D* . *f x* = *g x*
shows $\forall$ *x* $\in$ *topspace X* . *f x* = *g x*
proof −
obtain *Y* **where** *continuous-map X Y f* $\wedge$ *continuous-map X Y g* $\wedge$ *Hausdorff-space Y*
using *assms*(*1*) **by** *auto*
thus *?thesis* **using** *assms dense-in-def forall-in-closure-of-eq* **by** *fastforce*
qed

lemma *continuous-map-on-dense-embedding*:
assumes (*cts-on X to-shared Hausdorff-space*) {*f*,*g*}
and *dense-embedding D X e*
and $\forall$ *d* $\in$ *topspace D* . (*f o e*) *d* = (*g o e*) *d*
shows $\forall$ *x* $\in$ *topspace X* . *f x* = *g x*
using *assms continuous-maps-on-dense-subset*[*of f g X e* ' *topspace D*]
unfolding *dense-in-def* **by** *fastforce*

definition *range*′ :: ′*a topology* $\Rightarrow$ (′*a* $\Rightarrow$ *real*) $\Rightarrow$ *real set*
where *range*′ *X f* = *euclideanreal closure-of* (*f* ' *topspace X*)

abbreviation *fbounded-below* :: (′*a* $\Rightarrow$ *real*) $\Rightarrow$ ′*a topology* $\Rightarrow$ *bool*
where *fbounded-below f X* $\equiv$ ($\exists$ *m* . $\forall$ *y* $\in$ *topspace X* . *f y* $\geq$ *m*)

abbreviation *fbounded-above* :: (′*a* $\Rightarrow$ *real*) $\Rightarrow$ ′*a topology* $\Rightarrow$ *bool*
where *fbounded-above f X* $\equiv$ ($\exists$ *M* . $\forall$ *y* $\in$ *topspace X* . *f y* $\leq$ *M*)

abbreviation *fbounded* :: (′*a* $\Rightarrow$ *real*) $\Rightarrow$ ′*a topology* $\Rightarrow$ *bool*
where *fbounded f X* $\equiv$ ($\exists$ *m M* . $\forall$ *y* $\in$ *topspace X* . *m* $\leq$ *f y* $\wedge$ *f y* $\leq$ *M*)

lemma *fbounded-iff*:
shows *fbounded f X* $\longleftrightarrow$ *fbounded-below f X* $\wedge$ *fbounded-above f X*
by *auto*

abbreviation *c-of* :: ′*a topology* $\Rightarrow$ (′*a* $\Rightarrow$ *real*) *set* (*C*(-))
where *C*(*X*) $\equiv$ { *f* . *continuous-map X euclideanreal f* }

abbreviation *cstar-of* :: ′*a topology* $\Rightarrow$ (′*a* $\Rightarrow$ *real*) *set* (*C*$*$(-))
where *C*$*$ *X* $\equiv$ { *f* | *f* . *f* $\in$ *c-of X* $\wedge$ *fbounded f X* }

definition *cstar-id* :: ′*a topology* $\Rightarrow$ (′*a* $\Rightarrow$ *real*) $\Rightarrow$ ′*a* $\Rightarrow$ *real*
where *cstar-id X* = (λ *f* $\in$ *C*$*$ *X* . *f*)

abbreviation *c-embedding* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
  where *c-embedding S X e* ≡ *embedding-map S X e* ∧
  (∀ *fS* ∈ $C(S)$. ∃ *fX* ∈ $C(X)$. ∀ *x* ∈ *topspace S* . *fS x* = *fX* (*e x*))

abbreviation *cstar-embedding* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
  where *cstar-embedding S X e* ≡ *embedding-map S X e* ∧
  (∀ *fS* ∈ $C{*}(S)$. ∃ *fX* ∈ $C{*}(X)$. ∀ *x* ∈ *topspace S* . *fS x* = *fX* (*e x*))

definition *c-embedded* :: *'a topology* ⇒ *'b topology* ⇒ *bool*
  where *c-embedded S X* ≡ (∃ *e* . *c-embedding S X e*)

definition *cstar-embedded* :: *'a topology* ⇒ *'b topology* ⇒ *bool*
  where *cstar-embedded S X* ≡ (∃ *e* . *cstar-embedding S X e*)

lemma *bounded-range-iff-fbounded*:
  assumes *f* ∈ *C X*
  shows *bounded* (*f* ‘ *topspace X*) ⟷ *fbounded f X*
(**is** *?lhs* ⟷ *?rhs*)
proof
  assume *?lhs*
  then obtain *x e* **where** ∀ *y* ∈ *f* ‘ *topspace X* . *dist x y* ≤ *e*
    using *bounded-def*[*of f* ‘ *topspace X*] **by** *auto*
  hence ∀ *y* ∈ *f* ‘ *topspace X* . *y* ∈ { (*x*−*e*) .. (*x*+*e*) }
    using *dist-real-def* **by** *auto*
  thus *?rhs* **by** *auto*
next
  assume *?rhs*
  then obtain *m M* **where** ∀ *y* ∈ *f* ‘ *topspace X* . *y* ∈ {*m*..*M*} **by** *auto*
  thus *?lhs* **using** *bounded-closed-interval*[*of m M*] *subsetI bounded-subset*
    by *meson*
qed

Combinations of functions in $C(X)$ and $C^*(X)$

abbreviation *fconst* :: *real* ⇒ *'a* ⇒ *real*
  where *fconst v* ≡ (λ *x* . *v*)

definition *fmin* :: (*'a* ⇒ *real*) ⇒ (*'a* ⇒ *real*) ⇒ (*'a* ⇒ *real*)
  where *fmin f g* = (λ *x* . *min* (*f x*) (*g x*))

definition *fmax* :: (*'a* ⇒ *real*) ⇒ (*'a* ⇒ *real*) ⇒ (*'a* ⇒ *real*)
  where *fmax f g* = (λ *x* . *max* (*f x*) (*g x*))

definition *fmid* :: (*'a* ⇒ *real*) ⇒ (*'a* ⇒ *real*) ⇒ (*'a* ⇒ *real*) ⇒ *'a* ⇒ *real*
  where *fmid f m M* = *fmax m* (*fmin f M*)

definition *fbound* :: $('a \Rightarrow real) \Rightarrow real \Rightarrow real \Rightarrow 'a \Rightarrow real$
where *fbound f m M* = *fmid f* (*fconst m*) (*fconst M*)

lemma *fmin-cts*:
assumes $(f \in C\ X) \wedge (g \in C\ X)$
shows *fmin f g* $\in C\ X$
using *assms continuous-map-real-min*[*of X f g*] *fmin-def*[*of f g*] **by** *auto*

lemma *fmax-cts*:
assumes $(f \in C\ X) \wedge (g \in C\ X)$
shows *fmax f g* $\in C\ X$
using *assms continuous-map-real-max*[*of X f g*] *fmax-def*[*of f g*] **by** *auto*

lemma *fmid-cts*:
assumes $(f \in C\ X) \wedge (m \in C\ X) \wedge (M \in C\ X)$
shows *fmid f m M* $\in C\ X$
unfolding *fmid-def* **using** *assms fmin-cts*[*of f X M*] *fmax-cts*[*of m X* (*fmin f M*)]
by *auto*

lemma *fconst-cts*:
shows *fconst v* $\in C\ X$
by *simp*

lemma *fbound-cts*:
assumes $f \in C\ X$
shows *fbound f m M* $\in C\ X$
unfolding *fbound-def*
using *assms fmid-cts*[*of f X fconst m fconst M*] *fconst-cts*[*of m X*] *fconst-cts*[*of M X*]
by *auto*

Bounded and bounding functions

lemma *fconst-bounded*:
shows *fbounded* (*fconst v*) *X*
by *auto*

lemma *fmin-bounded-below*:
assumes *fbounded-below f X* $\wedge$ *fbounded-below g X*
shows *fbounded-below* (*fmin f g*) *X*
proof −
obtain *mf mg* **where** $\forall\ y \in$ *topspace X* . $f\ y \geq mf \wedge g\ y \geq mg$ **using** *assms* **by** *auto*
hence $\forall\ y \in$ *topspace X* . *fmin f g y* $\geq$ *min mf mg* **unfolding** *fmin-def min-def* **by** *auto*
thus *?thesis* **by** *auto*

qed

lemma *fmax-bounded-above*:
assumes *fbounded-above f X* $\wedge$ *fbounded-above g X*
shows *fbounded-above* (*fmax f g*) *X*
proof −
obtain *mf mg* **where** $\forall$ *y* $\in$ *topspace X* . *f y* $\leq$ *mf* $\wedge$ *g y* $\leq$ *mg* **using** *assms* **by** *auto*
hence $\forall$ *y* $\in$ *topspace X* . *fmax f g y* $\leq$ *max mf mg* **unfolding** *fmax-def max-def* **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *fmid-bounded*:
assumes *fbounded m X* $\wedge$ *fbounded M X*
shows *fbounded* (*fmid f m M*) *X*
proof −
obtain *mmin mmax Mmin Mmax*
where $\forall$ *y* $\in$ *topspace X* . *mmin* $\leq$ *m y* $\wedge$ *m y* $\leq$ *mmax* $\wedge$ *Mmin* $\leq$ *M y* $\wedge$ *M y* $\leq$ *Mmax*
using *assms* **by** *blast*
hence $\forall$ *y* $\in$ *topspace X* . *min mmin Mmin* $\leq$ (*fmid f m M y*) $\wedge$ (*fmid f m M y*) $\leq$ *max mmax Mmax*
unfolding *fmid-def fmax-def fmin-def max-def min-def* **by** *auto*
thus *?thesis* **by** *auto*
qed

lemma *fbound-bounded*:
shows *fbounded* (*fbound f m M*) *X*
using *fmid-bounded*[*of X fconst m fconst M*] *fconst-bounded*[*of X m*] *fconst-bounded*[*of X M*]
unfolding *fbound-def* **by** *simp*

Members of $C^*(X)$

lemma *fconst-cstar*:
shows *fconst v* $\in$ *C*∗ *X*
using *fconst-cts*[*of v X*] *fconst-bounded*[*of X v*]
by *auto*

lemma *fbound-cstar*:
assumes *f* $\in$ *C X*
shows *fbound f m M* $\in$ *C*∗ *X*
using *assms fbound-cts*[*of f X m M*] *fbound-bounded*[*of X f m M*]
by *auto*

lemma *cstar-nonempty*:
shows {} $\neq$ *C*∗ *X*
using *fconst-cstar* **by** *blast*

2 Weak topologies

definition *funcset-types* :: $'a\ set \Rightarrow ('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c\ topology) \Rightarrow 'b\ set \Rightarrow bool$
  where *funcset-types* $S\ F\ T\ I = (\forall\ i \in I\ .\ F\ i \in S \rightarrow topspace\ (T\ i))$

lemma *cstar-types*:
  shows *funcset-types* (*topspace* X) (*cstar-id* X) ($\lambda f \in C{*}\ X$. *euclideanreal*) ($C{*}$ X)
  unfolding *funcset-types-def*
  by *simp*

lemma *cstar-types-restricted*:
  shows *funcset-types* (*topspace* X) (*cstar-id* X)
      ($\lambda f \in C{*}\ X$. (*subtopology euclideanreal* (*range′* $X\ f$))) ($C{*}\ X$)
proof −
  have $\forall\ f \in C{*}\ X\ .\ f$ ‘ *topspace* $X \subseteq$ *range′* $X\ f$ **using** *range′-def*[*of* X]
    by (*metis closedin-subtopology-refl closedin-topspace closure-of-subset topspace-euclidean-subtopology*)
  thus *?thesis* **unfolding** *funcset-types-def*
    by (*simp add*: *image-subset-iff cstar-id-def*)
qed

definition *inverse′* :: $('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b\ set \Rightarrow 'a\ set$
where *inverse′ f source target* = $\{\ x \in source\ .\ f\ x \in target\ \}$

lemma *inverse′-alt*:
  shows *inverse′* $f\ s\ t = (f\ -‘\ t) \cap s$
  using *inverse′-def*[*of* $f\ s\ t$] **by** *auto*

definition *open-sets-induced-by-func* :: $('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b\ topology \Rightarrow 'a\ set\ set$
  where *open-sets-induced-by-func f source T*
      = $\{\ (inverse'\ f\ source\ V)\ |\ V\ .\ openin\ T\ V\ \wedge f \in source \rightarrow topspace\ T\}$

definition *weak-generators* :: $'a\ set \Rightarrow ('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c\ topology) \Rightarrow 'b\ set \Rightarrow 'a\ set\ set$
  where *weak-generators source funcs tops index*
    = $\bigcup$ { *open-sets-induced-by-func* (*funcs* i) *source* (*tops* i) | $i.\ i \in index$ }

definition *weak-base* :: $'a\ set \Rightarrow ('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c\ topology) \Rightarrow 'b\ set \Rightarrow 'a\ set\ set$
  where *weak-base source funcs tops index* = *base-generated-by* (*weak-generators source funcs tops index*)

definition *weak-opens* :: $'a\ set \Rightarrow ('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c\ topology) \Rightarrow 'b\ set \Rightarrow 'a\ set\ set$
where *weak-opens source funcs tops index = opens-generated-by (weak-generators source funcs tops index)*

definition *weak-topology* :: $'a\ set \Rightarrow ('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c\ topology) \Rightarrow 'b\ set \Rightarrow 'a\ topology$
where *weak-topology source funcs tops index*
= *topology-generated-by (weak-generators source funcs tops index)*

lemma *weak-topology-alt*:
shows *openin (weak-topology S F T I) U* $\longleftrightarrow$ $U \in$ *weak-opens S F T I*
using *weak-topology-def[of S F T I] weak-opens-def[of S F T I]*
opens-eq-generated-topology[of weak-generators S F T I U]
by *auto*

lemma *weak-generators-exist-for-each-point-and-axis*:
assumes $x \in S$
and *funcset-types S F T I*
and $i \in I$
and $b =$ *inverse'* $(F\ i)\ S$ *(topspace (T i))*
and $F\ i \in S \to$ *topspace (T i)*
shows $x \in b \land b \in$ *weak-generators S F T I*
proof −
have *xprops*: $x \in \{r \in S\ .\ F\ i\ r \in topspace\ (T\ i)\}$
using *assms(2) funcset-types-def[of S F T I] assms(3) assms(1)*
by *blast*
hence *part1*: $x \in b$ **using** *assms(4) inverse'-def[of F i S topspace (T i)]*
by *auto*

have *openin (T i) (topspace (T i))* **by** *simp*
hence $b \in$ *open-sets-induced-by-func (F i) S (T i)*
using *open-sets-induced-by-func-def[of F i S T i] assms(4) assms(5)*
inverse'-def[of F i S topspace (T i)] xprops
by *auto*
thus *?thesis* **using** *part1 weak-generators-def[of S F T I] assms(3)* **by** *auto*
qed

lemma *weak-generators-topspace*:
assumes $W =$ *weak-topology S F T I*
shows *topspace* $W = \bigcup$ *(weak-generators S F T I)*
using *weak-topology-def[of S F T I] assms* **by** *simp*

lemma *weak-topology-topspace*:
assumes $W =$ *weak-topology S F T I*
and *funcset-types S F T I*
shows $(I = \{\} \longrightarrow topspace\ W = \{\}) \land (I \neq \{\} \longrightarrow topspace\ W = S)$
proof *(cases* $I = \{\}$*)*
case *True*

hence *weak-generators* $S\ F\ T\ I = \{\}$ **using** *assms*(*1*) *weak-generators-def*[*of* S $F\ T\ I$] **by** *auto*
hence *topspace* $W = \{\}$ **using** *assms*(*1*) *weak-generators-topspace*[*of* $W\ S\ F\ T$ I] **by** *simp*
then show *?thesis* **using** *True* **by** *simp*
next
case *False*
then obtain i **where** *iprops*: $i \in I$ **by** *auto*
hence $(F\ i)$ ' $S \subseteq$ *topspace* $(T\ i)$
using *assms*(*2*) **unfolding** *funcset-types-def* **by** *auto*
hence *inverse'* $(F\ i)\ S$ (*topspace* $(T\ i)) = S$
using *inverse'-def*[*of* $F\ i\ S$ *topspace* $(T\ i)$] **by** *auto*
moreover have *openin* $(T\ i)$ (*topspace* $(T\ i)$) **using** *weak-generators-def* **by** *simp*
ultimately have $S \in$ *open-sets-induced-by-func* $(F\ i)\ S\ (T\ i)$
using *open-sets-induced-by-func-def*[*of* $F\ i\ S\ T\ i$] *assms*(*2*) *iprops* **unfolding** *funcset-types-def*
by *auto*
hence $S \in$ *weak-generators* $S\ F\ T\ I$
using *weak-generators-def*[*of* $S\ F\ T\ I$] *iprops* **by** *auto*
hence $S \subseteq$ *topspace* W
using *weak-generators-topspace*[*of* $W\ S\ F\ T\ I$] *assms* **by** *auto*

moreover have *topspace* $W \subseteq S$
proof −
have *openin* W (*topspace* W) **by** *auto*
hence *topspace* $W \in$ *opens-generated-by* (*weak-generators* $S\ F\ T\ I$)
using *assms*(*1*) **unfolding** *weak-topology-def*
using *opens-eq-generated-topology*[*of weak-generators* $S\ F\ T\ I$ *topspace* W]
by *simp*
then obtain A **where** *topspace* $W = \bigcup A \wedge A \subseteq \{\bigcap B\ |B.$ *finite'* $B \wedge B \subseteq$ *weak-generators* $S\ F\ T\ I\}$
using *opens-generated-unfolded*[*of weak-generators* $S\ F\ T\ I$]
by *auto*
thus *?thesis* **using** *assms*(*2*)
unfolding *weak-generators-def open-sets-induced-by-func-def inverse'-def funcset-types-def*
by *blast*
qed
ultimately show *?thesis* **using** *False* **by** *auto*
qed

lemma *weak-opens-nhood-base*:
assumes $W =$ *weak-topology* $S\ F\ T\ I$
and *openin* $W\ U$
and $x \in U$
shows $\exists\ b \in$ *weak-base* $S\ F\ T\ I$. $x \in b \wedge b \subseteq U$
proof −
define G **where** $G =$ *weak-generators* $S\ F\ T\ I$

hence *Wprops*: $U \in$ *opens-generated-by* G
using *weak-topology-def*[*of* S F T I] *opens-eq-generated-topology*[*of* G] *assms*(*1*) *assms*(*2*)
by *presburger*
then obtain B **where** *Bprops*: $B \subseteq$ *base-generated-by* $G \wedge U = \bigcup B$
unfolding *opens-generated-by-def arbitrary-unions-of-def* **by** *auto*
then obtain b **where** $b \in$ *base-generated-by* $G \wedge x \in b$
using *assms*(*3*) **by** *blast*
thus *?thesis* **using** *G-def weak-base-def*[*of* S F T I]
by (*metis Union-iff Bprops assms*(*3*) *subset-eq*)
qed

lemma *opens-generate-opens*:
assumes $\forall$ $b \in S$. *openin* T b
shows $\forall$ $U \in$ *opens-generated-by* S . *openin* T U
by (*metis assms generate-topology-on-coarsest istopology-openin openin-topology-generated-by*

opens-eq-generated-topology)

lemma *weak-topology-is-weakest*:
assumes $W =$ *weak-topology* S F T I
and *funcset-types* S F T I
and *topspace* $X =$ *topspace* W
and $\forall$ $i \in I$. *continuous-map* X $(T\ i)$ $(F\ i)$
and *openin* W U
shows *openin* X U
proof −
{ **fix** b **assume** *bprops*: $b \in$ *weak-generators* S F T I
then obtain i **where** *iprops*: $i \in I \wedge b \in$ *open-sets-induced-by-func* $(F\ i)$ S $(T\ i)$
using *weak-generators-def*[*of* S F T I] **by** *auto*
hence *Sprops*: $S =$ *topspace* X
using *assms*(*1*) *assms*(*2*) *weak-topology-topspace*[*of* W S F T I]
unfolding *funcset-types-def assms*(*3*)
by *auto*
obtain V **where** *Vprops*: *openin* $(T\ i)$ $V \wedge b =$ *inverse′* $(F\ i)$ S V
using *iprops open-sets-induced-by-func-def*[*of* F i S T i] **by** *auto*
have *cts*: *continuous-map* X $(T\ i)$ $(F\ i)$ **using** *iprops assms*(*4*) **by** *auto*
hence $\forall U$. *openin* $(T\ i)$ $U \longrightarrow$ *openin* X $\{x \in$ *topspace* $X.\ F\ i\ x \in U\}$
unfolding *continuous-map-def* **by** *simp*
hence *openin* X $\{x \in$ *topspace* $X.\ F\ i\ x \in V\}$ **using** *Vprops* **by** *auto*
hence *openin* X b **using** *Vprops Sprops* **unfolding** *inverse′-def* **by** *auto*
}
hence $\forall$ $b \in$ *weak-generators* S F T I . *openin* X b **by** *auto*
hence $\forall$ $c \in$ *weak-opens* S F T I . *openin* X c
using *assms*(*5*) *weak-opens-def*[*of* S F T I] *opens-generate-opens*[*of weak-generators* S F T I X]
by *auto*
moreover have $U \in$ *weak-opens* S F T I

using *assms*(*1*) *weak-topology-def*[*of S F T I*] *weak-opens-def*[*of S F T I*]
opens-eq-generated-topology[*of weak-generators S F T I U*] *assms*(*5*)
by *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *weak-generators-continuous*:
assumes $W = $ *weak-topology S F T I*
and *funcset-types S F T I*
and $i \in I$
shows *continuous-map* W $(T\ i)$ $(F\ i)$
proof −
have $S =$ *topspace W* **using** *assms*(*1*) *assms*(*2*) *assms*(*3*) *weak-topology-topspace*[*of W S F T I*]
unfolding *funcset-types-def* **by** *auto*
hence $F\ i \in$ *topspace W* $\rightarrow$ *topspace* $(T\ i)$
using *assms funcset-types-def*[*of S F T I*] **by** *auto*
moreover have $\forall\ V$. *openin* $(T\ i)$ $V \longrightarrow$ *openin W* $\{x \in$ *topspace W*. $(F\ i)\ x \in V\}$
proof −
{ fix V **assume** *Vprops*: *openin* $(T\ i)$ V
{ assume *hyp*: *inverse′* $(F\ i)$ (*topspace W*) $V \neq \{\}$
have $\{x \in$ *topspace W*. $(F\ i)\ x \in V\} =$ *inverse′* $(F\ i)$ (*topspace W*) V
using *inverse′-def*[*of F i topspace W V*] **by** *simp*
moreover have (*inverse′* $(F\ i)$ (*topspace W*) V) $\in$ *open-sets-induced-by-func* $(F\ i)$ S $(T\ i)$
using *Vprops assms weak-topology-topspace*[*of W S F T I*] *hyp*
unfolding *open-sets-induced-by-func-def funcset-types-def*
by *fastforce*
ultimately have $\{x \in$ *topspace W*. $(F\ i)\ x \in V\} \in$ *weak-generators S F T I*
using *weak-generators-def*[*of S F T I*] *assms*(*3*) **by** *auto*
hence *openin W* $\{x \in$ *topspace W*. $(F\ i)\ x \in V\}$
using *assms*(*1*) *weak-topology-def*[*of S F T I*]
generators-are-open[*of weak-generators S F T I*]
opens-eq-generated-topology[*of weak-generators S F T I* $\{x \in$ *topspace W*. $(F\ i)\ x \in V\}$]
by *auto*
}
hence *inverse′* $(F\ i)$ (*topspace W*) $V \neq \{\} \longrightarrow$ *openin W* $\{x \in$ *topspace W*. $(F\ i)\ x \in V\}$
by *auto*
moreover have *inverse′* $(F\ i)$ (*topspace W*) $V = \{\} \longrightarrow$ *openin W* $\{x \in$ *topspace W*. $(F\ i)\ x \in V\}$
by (*metis openin-empty inverse′-def*)
ultimately have *openin W* $\{x \in$ *topspace W*. $(F\ i)\ x \in V\}$ **by** *auto*
}
thus *?thesis* **by** *auto*
qed

ultimately show *?thesis* **using** *continuous-map-def* **by** *blast*
qed

lemma *funcset-types-on-empty*:
shows *funcset-types* {} *F T I*
unfolding *funcset-types-def* **by** *simp*

lemma *weak-topology-on-empty*:
assumes W = *weak-topology* {} *F T I*
shows $\forall$ *U* . *openin W U* $\longleftrightarrow$ *U* = {}
proof −
have *topspace W* = {}
using *assms*(*1*) *weak-topology-topspace*[*of W* {} *F T I*] *funcset-types-on-empty*[*of F T I*]
by *blast*
thus *?thesis* **by** *simp*
qed

2.1 Tychonov spaces carry the weak topology induced by $C^*(X)$

abbreviation *tych-space* :: *'a topology* $\Rightarrow$ *bool*
where *tych-space X* $\equiv$ *t1-space X* $\wedge$ *completely-regular-space X*

abbreviation *compact-Hausdorff* :: *'a topology* $\Rightarrow$ *bool*
where *compact-Hausdorff X* $\equiv$ *compact-space X* $\wedge$ *Hausdorff-space X*

lemma *compact-Hausdorff-imp-tych*:
assumes *compact-Hausdorff K*
shows *tych-space K*
by (*simp add*: *Hausdorff-imp-t1-space assms compact-Hausdorff-or-regular-imp-normal-space*

normal-imp-completely-regular-space-A)

lemma *tych-space-imp-Hausdorff*:
assumes *tych-space X*
shows *Hausdorff-space X*
proof −
have *Hausdorff-space euclideanreal* **by** *auto*
moreover have (*0*::*real*) $\neq$ (*1*::*real*) **by** *simp*
moreover have (*0*::*real*) $\in$ *topspace euclideanreal* $\wedge$ (*1*::*real*) $\in$ *topspace euclideanreal* **by** *simp*
ultimately have $\exists$ *U V* . *openin euclideanreal U* $\wedge$ *openin euclideanreal V* $\wedge$ (*0*::*real*) $\in$ *U* $\wedge$ (*1*::*real*) $\in$ *V* $\wedge$ *disjnt U V*
using *Hausdorff-space-def*[*of euclideanreal*] **by** *blast*
then obtain *U V*
where *UVprops*: *openin euclideanreal U* $\wedge$ *openin euclideanreal V* $\wedge$ (*0*::*real*) $\in$ *U* $\wedge$ (*1*::*real*) $\in$ *V* $\wedge$ *disjnt U V*
by *auto*

```
{ fix x y assume xyprops: x ∈ topspace X ∧ y ∈ topspace X ∧ x ≠ y
  hence closedin X {y} ∧ x ∈ topspace X − {y}
    using assms(1) by (simp add: t1-space-closedin-finite)
  then obtain f
     where fprops: continuous-map X (top-of-set {0..1}) f ∧ f x = (0::real) ∧ f
y ∈ {1::real}
    using assms(1) completely-regular-space-def[of X] by blast
  hence freal: continuous-map X euclideanreal f ∧ f x = 0 ∧ f y = 1
    using continuous-map-into-fulltopology by auto

  define U′ where U′ = { v ∈ topspace X . f v ∈ U }
  define V′ where V′ = { v ∈ topspace X . f v ∈ V }
  have openin X U′ ∧ openin X V′
    using U′-def V′-def UVprops freal continuous-map-def[of X euclideanreal f]
    by auto
   moreover have U′ ∩ V′ = {} using UVprops U′-def V′-def disjnt-def[of U
V] by auto
  moreover have x ∈ U′ ∧ y ∈ V′ using UVprops U′-def V′-def fprops xyprops
by auto
   ultimately have ∃ U′ V′ . openin X U′ ∧ openin X V′ ∧ x ∈ U′ ∧ y ∈ V′
∧ disjnt U′ V′
    using disjnt-def[of U′ V′] by auto
}
hence ∀ x y . x ∈ topspace X ∧ y ∈ topspace X ∧ x ≠ y
          ⟶ (∃ U′ V′ . openin X U′ ∧ openin X V′ ∧ x ∈ U′ ∧ y ∈ V′ ∧
disjnt U′ V′ )
  by auto
thus ?thesis using Hausdorff-space-def[of X] by blast
qed


lemma cstar-range-restricted:
 assumes f ∈ C∗ X
 and     U ⊆ topspace euclideanreal
shows    inverse′ f (topspace X) U = inverse′ f (topspace X) (U ∩ range′ X f)
proof −
 define U′ where U′ = U ∩ range′ X f
 hence inverse′ f (topspace X) U′ ⊆ inverse′ f (topspace X) U
   unfolding inverse′-def U′-def by auto
 moreover have inverse′ f (topspace X) U ⊆ inverse′ f (topspace X) U′
 proof −
   { fix x assume hyp: x ∈ inverse′ f (topspace X) U
     hence f x ∈ U ∩ (f ‘ topspace X) unfolding inverse′-def by auto
     hence f x ∈ U ∩ range′ X f
       unfolding range′-def
     by (metis Int-iff closure-of-subset-Int inf.orderE inf-top-left topspace-euclidean)
     hence x ∈ inverse′ f (topspace X) U′
       unfolding inverse′-def
```

```
        using U′-def hyp inverse′-alt by fastforce
    }
    thus ?thesis
      by (simp add: subsetI)
  qed
  ultimately show ?thesis using U′-def by simp
qed

lemma weak-restricted-topology-eq-weak:
shows  weak-topology (topspace X) (cstar-id X) (λ f ∈ C∗ X . euclideanreal) (C∗ X)
         = weak-topology (topspace X) (cstar-id X) (λ f ∈ C∗ X . subtopology euclideanreal (range′ X f)) (C∗ X)
proof −
  define T  where T  = (λ f ∈ C∗ X . euclideanreal)
  define T′ where T′ = (λ f ∈ C∗ X . subtopology euclideanreal (range′ X f))
  define W  where W  = weak-topology (topspace X) (cstar-id X) T (C∗ X)
  define W′ where W′ = weak-topology (topspace X) (cstar-id X) T′ (C∗ X)

  have ∀ f ∈ C∗ X . f ∈ topspace X → topspace (T f)
    using T-def unfolding continuous-map-def T-def by auto

  have generators: weak-generators (topspace X) (cstar-id X) T (C∗ X)
             = weak-generators (topspace X) (cstar-id X) T′ (C∗ X)
  proof −

    have weak-generators (topspace X) (cstar-id X) T (C∗ X)
             ⊆ weak-generators (topspace X) (cstar-id X) T′ (C∗ X)
    proof −
      have weak-generators (topspace X) (cstar-id X) T (C∗ X)
             ⊆ weak-generators (topspace X) (cstar-id X) T′ (C∗ X)
      proof −
        { fix U assume Uprops: U ∈ weak-generators (topspace X) (cstar-id X) T (C∗ X)
          then obtain f where fprops: f ∈ (C∗ X) ∧ U ∈ open-sets-induced-by-func f (topspace X) (T f)
            unfolding weak-generators-def using cstar-id-def[of X]
            by (smt (verit) Union-iff mem-Collect-eq restrict-apply′)
          then obtain V where Vprops: U = inverse′ f (topspace X) V ∧ openin (T f) V
            unfolding open-sets-induced-by-func-def by blast
          hence U = inverse′ f (topspace X) V by auto
          hence rtp1: U ⊆ topspace X unfolding inverse′-def by auto

          have rtp2: openin (T′ f) (V ∩ range′ X f)
          proof −
            have openin euclideanreal V using fprops Vprops T-def by auto
            hence openin (subtopology euclideanreal (range′ X f)) (V ∩ range′ X f)
              by (simp add: openin-subtopology-Int)
```

thus *?thesis* **using** *fprops* T'*-def* **by** *auto*
qed

have *rtp3*: $f \in$ *topspace* $X \to$ *topspace* $(T'\,f)$
proof −
have f ‘ *topspace* $X \subseteq$ *topspace euclideanreal* **using** *fprops* **by** *auto*
hence f ‘ *topspace* $X \subseteq$ *range*$'$ $X\,f$ **unfolding** *range*$'$*-def*
by (*meson closure-of-subset*)
thus *?thesis* **using** T'*-def fprops* **by** *auto*
qed

hence *rtp4*: $U =$ *inverse*$'$ f (*topspace* X) ($V \cap$ *range*$'$ $X\,f$)
proof −
have *inverse*$'$ f (*topspace* X) ($V \cap$ *range*$'$ $X\,f$) $\subseteq U$
using *Vprops fprops* **unfolding** *inverse*$'$*-def* **by** *auto*
moreover have $U \subseteq$ *inverse*$'$ f (*topspace* X) ($V \cap$ *range*$'$ $X\,f$)
proof −
{ fix u **assume** *uprops*: $u \in U$
hence $f\,u \in V$ **using** *Vprops* **unfolding** *inverse*$'$*-def* **by** *auto*
moreover have $f\,u \in$ *range*$'$ $X\,f$ **using** *uprops rtp1* **unfolding** *range*$'$*-def*
by (*metis closure-of-subset-Int imageI inf-top-left subset-iff topspace-euclidean*)
ultimately have $u \in$ *inverse*$'$ f (*topspace* X) ($V \cap$ *range*$'$ $X\,f$)
unfolding *inverse*$'$*-def range*$'$*-def* **using** *rtp1 uprops* **by** *force*
}
thus *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

have $U \in$ *open-sets-induced-by-func* f (*topspace* X) ($T'\,f$)
using *rtp1 rtp2 rtp3 rtp4* **unfolding** *open-sets-induced-by-func-def*
by *blast*
hence $U \in$ *weak-generators* (*topspace* X) (*cstar-id* X) T' ($C*$ X)
using *fprops weak-generators-def*[*of* (*topspace* X) (*cstar-id* X) T' ($C*$ X)] *cstar-id-def*[*of* X]
by (*smt* (*verit*, *best*) *Sup-upper in-mono mem-Collect-eq restrict-apply*$'$)
}
thus *?thesis* **by** *auto*
qed
thus *?thesis* **by** *auto*
qed

moreover have *weak-generators* (*topspace* X) (*cstar-id* X) T' ($C*$ X)
$\subseteq$ *weak-generators* (*topspace* X) (*cstar-id* X) T ($C*$ X)
proof −
{ fix U **assume** *Uprops*: $U \in$ *weak-generators* (*topspace* X) (*cstar-id* X) T' ($C*$ X)

then obtain f **where** *fprops*: $f \in (C* \ X) \wedge U \in$ *open-sets-induced-by-func* f (*topspace* X) (T' f)
unfolding *weak-generators-def* **using** *cstar-id-def*[*of* X]
by (*smt* (*verit*) *Union-iff mem-Collect-eq restrict-apply'*)

then obtain V **where** *Vprops*: U = *inverse'* f (*topspace* X) $V \wedge$ *openin* (T' f) V
unfolding *open-sets-induced-by-func-def* **by** *blast*

have T' f = *subtopology* (T f) (*topspace* (T' f))
using *T-def T'-def fprops* **unfolding** *range'-def* **by** *auto*
moreover have *openin* (T' f) V **using** *Vprops* **by** *simp*
ultimately obtain *Vbig* **where** *Vbigprops*: *openin* (T f) *Vbig* $\wedge$ V = *Vbig* $\cap$ (*topspace* (T' f))
using *openin-subtopology*[*of* T f *topspace* (T' f)]
by *auto*

have *Vrestrict*: *Vbig* $\cap$ *topspace* (T' f) = *Vbig* $\cap$ *range'* X f
using *T'-def fprops* **by** *auto*

have *Vrange*: *inverse'* f (*topspace* X) (*Vbig* $\cap$ *range'* X f) = *inverse'* f (*topspace* X) *Vbig*
proof −
{ fix x **assume** $x \in$ *inverse'* f (*topspace* X) *Vbig*
hence $x \in$ *topspace* $X \wedge f\ x \in$ *Vbig* $\cap$ *range'* X f
using *range'-def*[*of* X f]
by (*metis Int-iff closure-of-subset image-subset-iff inverse'-alt subset-UNIV*

topspace-euclidean vimage-eq)
hence $x \in$ *inverse'* f (*topspace* X) (*Vbig* $\cap$ *range'* X f) **unfolding** *inverse'-def* **by** *auto*
}
hence *inverse'* f (*topspace* X) *Vbig* $\subseteq$ *inverse'* f (*topspace* X) (*Vbig* $\cap$ *range'* X f) **by** *auto*
thus *?thesis* **unfolding** *inverse'-def* **by** *auto*
qed
hence U = *inverse'* f (*topspace* X) *Vbig* $\wedge$ *openin* (T f) *Vbig*
by (*simp add*: *Vbigprops Vprops Vrestrict*)
moreover have *fcstar*: $f \in C* \ X$ **using** *fprops* **by** *simp*
ultimately have $U \in$ *open-sets-induced-by-func* f (*topspace* X) (T f)
using *open-sets-induced-by-func-def*[*of* f *topspace* X *euclideanreal*] *T-def*
by *auto*
hence $U \in$ *open-sets-induced-by-func* (*cstar-id* X f) (*topspace* X) (T f) $\wedge$ $f \in C* \ X$
using *fcstar cstar-id-def*[*of* X] **by** *auto*
hence $U \in$ *weak-generators* (*topspace* X) (*cstar-id* X) T ($C* \ X$)
using *fcstar* **unfolding** *weak-generators-def* **by** *auto*
}
thus *?thesis* **by** *auto*

qed
ultimately show *?thesis* **by** *auto*
qed
thus *?thesis* **by** (*simp add*: *T-def* *T′-def* *weak-topology-def* *cstar-id-def*)
qed

2.2 A topology is a weak topology if it admits a continuous function set that separates points from closed sets

definition *funcset-separates-points* :: $'a$ *topology* $\Rightarrow$ $('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow 'b$ *set* $\Rightarrow$ *bool*
where *funcset-separates-points* $X\ F\ I$
$= (\forall\ x \in$ *topspace* $X\ .\ \forall\ y \in$ *topspace* $X\ .\ x \neq y \longrightarrow (\exists\ i \in I\ .\ F\ i\ x \neq F\ i\ y))$

definition *funcset-separates-points-from-closed-sets* ::
$'a$ *topology* $\Rightarrow ('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow ('b \Rightarrow 'c$ *topology*$) \Rightarrow 'b$ *set* $\Rightarrow$ *bool*
where *funcset-separates-points-from-closed-sets* $X\ F\ T\ I$
$= (\forall\ x\ .\ \forall\ A\ .$ *closedin* $X\ A \wedge x \in ($*topspace* $X - A)$
$\longrightarrow (\exists\ i \in I\ .\ F\ i\ x\ \notin (T\ i)$ *closure-of* $(F\ i\ `\ A)))$

lemma *funcset-separates-points-from-closed-sets-imp-weak*:
assumes *funcset-separates-points-from-closed-sets* $X\ F\ T\ I$
and $\forall\ i \in I\ .$ *continuous-map* $X\ (T\ i)\ (F\ i)$
and $W =$ *weak-topology* (*topspace* X) $F\ T\ I$
and *funcset-types* (*topspace* X) $F\ T\ I$
shows $X = W$
proof −
{ **fix** U **assume** *Uhyp*: *openin* $X\ U$
{ **fix** x **assume** *xhyp*: $x \in U$
define A **where** $A = ($*topspace* $X) - U$
have *xinX*: $x \in$ *topspace* X **using** *Uhyp* *xhyp* *openin-subset* **by** *auto*
moreover have *Aprops*: *closedin* $X\ A \wedge x \notin A$ **using** *Uhyp* *xhyp* *A-def* **by** *auto*
ultimately obtain i **where** *iprops*: $i \in I \wedge F\ i\ x\ \notin (T\ i)$ *closure-of* $(F\ i\ `\ A)$
using *assms*(*1*) *funcset-separates-points-from-closed-sets-def*[*of* $X\ F\ T\ I$] **by** *auto*

define V **where** $V =$ *topspace* $(T\ i) - (T\ i)$ *closure-of* $(F\ i\ `\ A)$
define R **where** $R = \{\ p \in ($*topspace* $X)\ .\ F\ i\ p \in V\ \}$
have *Vopen*: *openin* $(T\ i)\ V \wedge F\ i\ x \in V$ **using** *iprops* *xinX* *V-def*
by (*metis DiffI Int-iff assms*(*2*) *closedin-closure-of continuous-map-preimage-topspace*
openin-diff openin-topspace vimage-eq)
hence $x \in R$ **using** *R-def* *assms*(*2*) *xinX* **by** *simp*
moreover have $R \subseteq U$
proof −
have $F\ i\ `\ R \subseteq V$ **using** *R-def* **by** *auto*

hence $F\ i\ `\ R \cap (T\ i)$ *closure-of* $(F\ i\ `\ A) = \{\}$ **using** *V-def* **by** *auto*
moreover have $F\ i\ `\ A \subseteq (T\ i)$ *closure-of* $(F\ i\ `\ A)$
by (*metis Aprops assms*(*2*) *closure-of-eq continuous-map-subset-aux1 iprops*)
ultimately have $F\ i\ `\ R \cap (F\ i\ `A) = \{\}$ **by** *auto*
hence $R \cap A = \{\}$ **by** *auto*
thus *?thesis* **using** *A-def R-def* **by** *auto*
qed
moreover have *openin W R*
proof −
have $R =$ *inverse′* $(F\ i)$ (*topspace X*) V
by (*simp add*: *R-def inverse′-def*)
hence $R \in$ *open-sets-induced-by-func* $(F\ i)$ (*topspace X*) $(T\ i)$
using *open-sets-induced-by-func-def*[*of F i topspace X T i*] *Vopen*
assms(*2*) *continuous-map-funspace iprops* **by** *fastforce*
hence $R \in$ *weak-generators* (*topspace X*) *F T I*
using *weak-generators-def*[*of topspace X F T I*] *iprops* **by** *auto*
thus *?thesis* **using** *generators-are-open*[*of weak-generators* (*topspace X*) *F*
T I]
opens-eq-generated-topology[*of weak-generators* (*topspace X*) *F T I R*]
assms(*3*)
by (*simp add*: *topology-generated-by-Basis weak-topology-def*)
qed
ultimately have $x \in R \wedge R \subseteq U \wedge$ *openin W R* **by** *auto*
hence $\exists\ R\ .\ x \in R \wedge R \subseteq U \wedge$ *openin W R* **by** *auto*
}
hence $\forall\ x\ .\ x \in U \longrightarrow (\exists\ R\ .\ x \in R \wedge R \subseteq U \wedge$ *openin W R*)
by *auto*
hence *openin W U* **by** (*meson openin-subopen*)
}
hence *XimpW*: $\forall\ U\ .$ *openin X U* $\longrightarrow$ *openin W U* **by** *auto*

moreover have $\forall\ U\ .$ *openin W U* $\longrightarrow$ *openin X U*
proof −
have *topspace X* = *topspace W*
using *assms*(*3*) *assms*(*4*) *weak-topology-topspace*[*of W topspace X F T I*]
by (*metis XimpW openin-topspace openin-topspace-empty subtopology-eq-discrete-topology-empty*)
thus *?thesis*
using *assms*(*3*) *assms*(*4*) *assms*(*2*) *weak-topology-is-weakest*[*of W topspace X*
F T I X]
by *blast*
qed
ultimately show *?thesis* **by** (*meson topology-eq*)
qed

The canonical functions on a product space: evaluation and projection

definition *evaluation-map* :: $'a$ *topology* $\Rightarrow ('b \Rightarrow 'a \Rightarrow 'c) \Rightarrow 'b$ *set* $\Rightarrow 'a \Rightarrow 'b \Rightarrow$
$'c$
where *evaluation-map X F I* $= (\lambda\ x \in$ *topspace X* $.\ (\lambda\ i \in I\ .\ F\ i\ x))$

definition *product-projection* :: $('a \Rightarrow 'b\ topology) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b$
where *product-projection* $T\ I = (\lambda\ i \in I\ .\ (\lambda\ p \in topspace\ (product\text{-}topology\ T\ I)\ .\ p\ i))$

lemma *product-projection*:
shows $\forall\ i \in I\ .\ \forall\ p \in topspace\ (product\text{-}topology\ T\ I)\ .\ product\text{-}projection\ T\ I\ i\ p = p\ i$
using *product-projection-def*[*of T I*] **by** *simp*

lemma *evaluation-then-projection*:
assumes $\forall\ i \in I\ .\ F\ i \in topspace\ X \rightarrow topspace\ (T\ i)$
shows $\forall\ i \in I\ .\ \forall\ x \in topspace\ X\ .\ ((product\text{-}projection\ T\ I\ i)\ o\ (evaluation\text{-}map\ X\ F\ I))\ x = F\ i\ x$
proof −
{ fix *i* **assume** *iprops*: $i \in I$
{ fix *x* **assume** *xprops*: $x \in topspace\ X$
have *Fix*: $(\lambda\ i \in I\ .\ F\ i\ x) \in topspace\ (product\text{-}topology\ T\ I)$ **using** *xprops assms*(*1*) **by** *auto*
have $((product\text{-}projection\ T\ I\ i)\ o\ (evaluation\text{-}map\ X\ F\ I))\ x$
$= (product\text{-}projection\ T\ I\ i)\ ((\lambda\ x \in topspace\ X\ .\ (\lambda\ i \in I\ .\ F\ i\ x))\ x)$
unfolding *evaluation-map-def* **by** *auto*
moreover have $\ldots = (product\text{-}projection\ T\ I\ i)\ (\lambda\ i \in I\ .\ F\ i\ x)$ **using** *xprops* **by** *simp*
moreover have $\ldots = (\lambda\ p \in topspace\ (product\text{-}topology\ T\ I)\ .\ p\ i)\ (\lambda\ i \in I\ .\ F\ i\ x)$
unfolding *product-projection-def* **using** *iprops* **by** *auto*
moreover have $\ldots = F\ i\ x$ **using** *Fix iprops* **by** *simp*
ultimately have $((product\text{-}projection\ T\ I\ i)\ o\ (evaluation\text{-}map\ X\ F\ I))\ x = F\ i\ x$ **by** *auto*
}
hence $\forall\ x \in topspace\ X\ .\ ((product\text{-}projection\ T\ I\ i)\ o\ (evaluation\text{-}map\ X\ F\ I))\ x = F\ i\ x$
by *auto*
}
thus *?thesis* **by** *auto*
qed

2.3 A product topology is the weak topology induced by its projections if the projections separate points from closed sets.

lemma *projections-continuous*:
assumes $P = product\text{-}topology\ T\ I$
and $F = (\lambda\ i \in I\ .\ product\text{-}projection\ T\ I\ i)$
shows $\forall i{\in}I.\ continuous\text{-}map\ P\ (T\ i)\ (F\ i)$
using *assms*(*1*) *assms*(*2*) *product-projection-def*[*of T I*]
by *fastforce*

```
lemma product-topology-eq-weak-topology:
  assumes P = product-topology T I
and       F = (λ i ∈ I . product-projection T I i)
and       W = weak-topology (topspace P) F T I
and       funcset-types (topspace P) F T I
and       funcset-separates-points-from-closed-sets P F T I
shows      P = W
using assms product-projection-def[of T I] projections-continuous
      funcset-separates-points-from-closed-sets-imp-weak[of P F T I W]
    by simp
```

Reducing the domain and minimising the range of continuous functions, and related results concerning weak topologies.

```
lemma continuous-map-reduced:
  assumes continuous-map X Y f
  shows    continuous-map (subtopology X S) (subtopology Y (f‘S)) (restrict f S)
 using assms continuous-map-from-subtopology continuous-map-in-subtopology by
fastforce

lemma inj-on-imp:
  assumes inj-on f S
  shows ∀ y . (y ∈ f ‘ S) ⟷ (∃ x ∈ S . y = f x)
by (simp add: image-iff)

lemma injection-on-intersection:
  assumes inj-on f S
and       B ≠ {}
and       ∀ b ∈ B . b ⊆ S
  shows   f ‘ (⋂B) = ⋂{ f ‘ b | b . b ∈ B }
(is ?lhs = ?rhs)
proof −
  have ?lhs ⊆ ?rhs by auto
  moreover have ?rhs ⊆ ?lhs
  proof −
    { fix y assume rhs: y ∈ ?rhs
      then obtain b where bprops: y ∈ f ‘ b ∧ b ∈ B
        by (smt (verit, del-insts) Inter-iff assms(2) ex-in-conv mem-Collect-eq)
      then obtain x where xprops: x ∈ b ∧ b ∈ B ∧ y = f x by auto


      have ∀ b ∈ B . y ∈ f ‘ b using rhs by auto
      hence ∀ b ∈ B . f x ∈ f ‘ b using xprops by auto
      hence ∀ b ∈ B . x ∈ b using assms(1)
        by (meson assms(3) in-mono inj-on-image-mem-iff xprops)
      hence x ∈ ⋂B by auto
      hence y ∈ ?lhs using xprops by auto
    }
    thus ?thesis by auto
```

```
  qed
  ultimately show ?thesis by auto
qed
```

2.4 Evaluation is an embedding for weak topologies

```
lemma evaluation-is-embedding:
  assumes X = weak-topology (topspace X) F T I
and      P = product-topology T I
and      funcset-types (topspace X) F T I
and      funcset-separates-points X F I
shows     embedding-map X P (evaluation-map X F I)
proof −
  define ev   where ev   = evaluation-map X F I
  define proj where proj = product-projection T I
  define R    where R    = ev ‘ topspace X
  define Rtop where Rtop = subtopology P R


  have injective: inj-on ev (topspace X)
  proof −
    have sigs: ∀ i ∈ I . F i ∈ (topspace X) → (topspace (T i))
      using assms(3) funcset-types-def[of topspace X F T I]
      by blast

    { fix x y assume xyprops: x ∈ topspace X ∧ y ∈ topspace X
      { assume hyp: x ≠ y
        then obtain i where iprops: i ∈ I ∧ F i x ≠ F i y
          using assms(4) funcset-separates-points-def[of X F I] hyp xyprops
          by blast
        hence (proj i) (ev x) ≠ (proj i) (ev y)
          using evaluation-then-projection[of I F X T] proj-def ev-def
          by (simp add: sigs xyprops)
        hence ev x ≠ ev y by auto
      }
      hence ev x = ev y ⟶ x = y by auto
    }
    thus ?thesis using inj-on-def by blast
  qed

  moreover have ev-cts: continuous-map X Rtop ev
  proof −
    have main: ∀ i ∈ I . ∀ x ∈ topspace X . (proj i o ev) x = F i x
     using proj-def ev-def product-projection-def[of T I] evaluation-then-projection[of
I F X T]
          evaluation-map-def[of X F I]
    by (metis assms(1) assms(3) continuous-map-funspace weak-generators-continuous)
    moreover have ∀ i ∈ I . continuous-map X (T i) (F i)
      using weak-generators-continuous[of X topspace X F T I] assms by auto
```

```
    moreover have ∀ i ∈ I . ∀ x ∈ topspace X . F i x = ev x i
      using product-projection-def[of T I] main ev-def
      by (simp add: evaluation-map-def[of X F I])
    moreover have ev ‘ topspace X ⊆ extensional I
      using ev-def extensional-def assms evaluation-map-def[of X F I]
      by fastforce
    ultimately have continuous-map X P ev
       using assms proj-def ev-def Rtop-def continuous-map-componentwise[of X T
I ev]
            continuous-map-eq by fastforce
    thus ?thesis
      using Rtop-def R-def continuous-map-in-subtopology by blast
  qed

  moreover have open-map X Rtop ev
  proof −
    have open-map-on-gens: ∀ U ∈ weak-generators (topspace X) F T I . openin
Rtop (ev ‘ U)
    proof −
      { define Rs where Rs = (λ i ∈ I . (F i ‘ topspace X))
        define Rtops where Rtops = (λ i ∈ I . subtopology (T i) (Rs i))

        fix U assume U ∈ weak-generators (topspace X) F T I
        then obtain i where iprops: i ∈ I ∧ U ∈ open-sets-induced-by-func (F i)
(topspace X) (T i)
          using assms weak-generators-def[of topspace X F T I] by auto
        then obtain V
          where Vprops: openin (T i) V ∧ U = inverse′ (F i) (topspace X) V
          using open-sets-induced-by-func-def[of F i topspace X T i]
          by blast
        hence Uprops: openin (T i) V ∧ U = { x ∈ topspace X . F i x ∈ V }
          using inverse′-def[of F i topspace X V] by auto
        moreover have ∀ x ∈ topspace X . F i x = ((proj i) o ev) x
          using evaluation-then-projection[of I F X T] assms(3)
                funcset-types-def[of topspace X F T I] iprops
                proj-def ev-def
          by auto
         hence U = { x ∈ topspace X . ((proj i) o ev) x ∈ V } using Uprops by
auto
        hence ev ‘ U = { y ∈ R . (proj i) y ∈ V } using R-def by auto
        moreover have { y ∈ R . (proj i) y ∈ V } = R ∩ ((proj i) −‘ V )
          by auto
        moreover have continuous-map P (T i) (proj i)
          using continuous-map-product-projection[of i I T] iprops proj-def
                product-projection-def[of T I] assms(2)  by auto
       ultimately have summary: openin (T i) V ∧ continuous-map P (T i) (proj
i)
                          ∧ (ev ‘ U) = R ∩ ((proj i) −‘ V )  by auto
        hence ∀ U. openin (T i) U ⟶ openin P {x ∈ topspace P. proj i x ∈ U}
```

using *continuous-map-def*[*of P T i proj i*] **by** *auto*
hence *openin P* ((*proj i* −‘ *V*) ∩ *topspace P*)
using *summary* **by** *blast*
moreover have *R* ⊆ *topspace P*
using *R-def ev-def evaluation-map-def*[*of X F I*] *assms*(*3*)
funcset-types-def[*of topspace X F T I*]
by (*metis Rtop-def ev-cts continuous-map-image-subset-topspace*
continuous-map-into-fulltopology)
ultimately have *openin Rtop* ((*proj i* −‘ *V*) ∩ *R*)
using *Rtop-def*
by (*metis inf.absorb-iff2 inf-assoc openin-subtopology*)
hence *openin Rtop* (*ev* ‘ *U*) **using** *summary*
by (*simp add*: *inf-commute*)
}
thus *?thesis* **by** *auto*
qed

have *open-map-on-basics*: ∀ *U* ∈ *weak-base* (*topspace X*) *F T I* . *openin Rtop*
(*ev* ‘ *U*)
proof −
have *Ugens*: ⋃ (*weak-generators* (*topspace X*) *F T I*) = *topspace X*
using *assms*(*1*) *weak-generators-topspace* **by** *blast*

{ fix *U* **assume** *bprops*: *U* ∈ *weak-base* (*topspace X*) *F T I*
hence *U* ∈ *finite-intersections-of* (*weak-generators* (*topspace X*) *F T I*)
by (*simp add*: *base-generated-by-def weak-base-def*)
then obtain *b* **where** *bprops*: *b* ⊆ *weak-generators* (*topspace X*) *F T I* ∧
finite′ b ∧ *U* = ⋂ *b*
unfolding *finite-intersections-of-def*
by *auto*
hence *finite′ b* ∧ (∀ *g* ∈ *b* . *openin Rtop* (*ev* ‘ *g*)) **using** *open-map-on-gens*
by *auto*
hence *openin Rtop* (⋂ { (*ev* ‘ *g*) | *g* . *g* ∈ *b* }) **by** *auto*
hence *openin Rtop* (*ev* ‘ ⋂ *b*)
using *injection-on-intersection*[*of ev topspace X b*] *bprops*
by (*metis* (*no-types*, *lifting*) *Ugens Union-upper in-mono injective*)
hence *openin Rtop* (*ev* ‘ *U*) **using** *bprops* **by** *metis*
}
thus *?thesis* **by** *auto*
qed
hence *open-map-on-opens*: ∀ *U* ∈ *weak-opens* (*topspace X*) *F T I* . *openin*
Rtop (*ev* ‘ *U*)
by (*smt* (*verit*, *ccfv-SIG*) *image-iff image-mono openin-subopen weak-opens-nhood-base*

weak-topology-alt)
thus *?thesis*
using *opens-eq-generated-topology*[*of weak-generators* (*topspace X*) *F T I*]
assms(*1*)
unfolding *weak-topology-def* **using** *open-map-def*[*of X Rtop*]

by (*simp add*: *weak-opens-def*)
qed

ultimately have *homeomorphic-map X Rtop ev*
by (*metis R-def Rtop-def bijective-open-imp-homeomorphic-map continuous-map-image-subset-topspace*

continuous-map-into-fulltopology topspace-subtopology-subset)
thus *?thesis* **using** *embedding-map-def*[*of X P ev*] *ev-def R-def Rtop-def*
by *auto*
qed

3 Compactification

3.1 Definition

lemma *embedding-map-id*:
assumes $S \subseteq$ *topspace X*
shows *embedding-map* (*subtopology X S*) *X id*
using *assms embedding-map-def topspace-subtopology-subset*
by *fastforce*

definition *compactification-via* :: $('a \Rightarrow 'b) \Rightarrow 'a$ *topology* $\Rightarrow 'b$ *topology* $\Rightarrow$ *bool*
where *compactification-via f X K* $\equiv$ *compact-space K* $\wedge$ *dense-embedding X K f*

definition *compactification* :: $'a$ *topology* $\Rightarrow 'b$ *topology* $\Rightarrow$ *bool*
where *compactification X K* = ($\exists$ *f* . *compactification-via f X K*)

lemma *compactification-compactification-via*:
assumes *compactification-via f X K*
shows *compactification X K*
using *assms* **unfolding** *compactification-def* **by** *fastforce*

3.2 Example: The Alexandroff compactification of a non-compact locally-compact Hausdorff space

lemma *Alexandroff-is-compactification-via-Some*:
assumes $\neg$ *compact-space X* $\wedge$ *Hausdorff-space X* $\wedge$ *locally-compact-space X*
shows *compactification-via Some X* (*Alexandroff-compactification X*)
using *assms compact-space-Alexandroff-compactification*
embedding-map-Some
Alexandroff-compactification-dense
compactification-via-def
by (*metis dense-in-def*)

3.3 Example: The closure of a subset of a compact space

lemma *compact-closure-is-compactification*:

assumes *compact-space K*
and $S \subseteq$ *topspace K*
shows *compactification-via id* (*subtopology K S*) (*subtopology K* (*K closure-of S*))
proof −
define *big* **where** *big* = *subtopology K* (*K closure-of S*)
define *small* **where** *small* = *subtopology K S*
have *dense-in big* (*id ' topspace small*) (*topspace big*)
by (*metis dense-in-def big-def small-def assms*(*2*) *closedin-topspace closure-of-minimal closure-of-subset closure-of-subtopology-open id-def image-id inf.orderE openin-imp-subset openin-subtopology-refl topspace-subtopology-subset*)
moreover have *embedding-map small big id*
by (*metis assms*(*2*) *big-def closure-of-subset-Int embedding-map-in-subtopology id-apply embedding-map-id image-id small-def topspace-subtopology*)
ultimately have *dense-embedding small big id* **by** *blast*
moreover have *compact-space big*
by (*simp add*: *big-def assms*(*1*) *closedin-compact-space compact-space-subtopology*)
ultimately show *?thesis*
unfolding *compactification-via-def* **using** *small-def big-def* **by** *blast*
qed

3.4 Example: A compact space is a compactification of itself

lemma *compactification-of-compact*:
assumes *compact-space K*
shows *compactification-via id K K*
using *compact-closure-is-compactification*[*of K topspace K*]
by (*simp add*: *assms*)

3.5 Example: A closed non-trivial real interval is a compactification of its interior

lemma *closed-interval-interior*:
shows {*a*::*real* <..< *b*} = *interior* {*a*..*b*}
by *auto*

lemma *open-interval-closure*:
shows (*a* < (*b*::*real*)) ⟶ {*a* .. *b*} = *closure* {*a* <..< *b*}
using *closure-greaterThanLessThan*[*of a b*] **by** *simp*

lemma *closed-interval-compactification*:
assumes (*a*::*real*) < *b*
and *open-interval* = *subtopology euclideanreal* {*a*<..<*b*}
and *closed-interval* = *subtopology euclideanreal* {*a*..*b*}
shows *compactification open-interval closed-interval*
proof −
have *compact-space closed-interval* **using** *assms*(*3*)

using *compact-space-subtopology compactin-euclidean-iff* **by** *blast*
moreover have *Hausdorff-space closed-interval*
by (*simp add*: *Hausdorff-space-subtopology assms*(*3*))
moreover have $\{a<..<b\} \subseteq$ *topspace closed-interval*
by (*simp add*: *assms*(*3*) *greaterThanLessThan-subseteq-atLeastAtMost-iff*)
ultimately have *compactification-via id open-interval closed-interval*
using *compact-closure-is-compactification*[*of closed-interval* $\{a<..<b\}$]
open-interval-closure[*of a b*]
by (*metis assms closedin-self closedin-subtopology-refl closure-of-subtopology euclidean-closure-of subtopology-subtopology subtopology-topspace topspace-subtopology-subset*)
thus *?thesis* **using**
compactification-compactification-via[*of id open-interval closed-interval*]
by *auto*
qed

4 The Stone-Čech compactification of a Tychonov space

lemma *compact-range'*:
assumes $f \in C*\ X$
shows *compact* (*range' X f*)
proof −
obtain *m M* **where** *mM*: $\forall\ y \in$ *topspace X* . $f\ y \in \{m..M\}$ **using** *assms* **by** *auto*
hence *f* ' *topspace X* $\subseteq \{m..M\}$ **by** *auto*
hence *range' X f* $\subseteq$ *euclideanreal closure-of* $\{m..M\}$
unfolding *range'-def* **by** (*meson closure-of-mono*)
moreover have *compact* $\{m..M\}$ **by** *auto*
ultimately show *?thesis*
by (*metis closed-Int-compact closed-atLeastAtMost closed-closedin closedin-closure-of*

closure-of-closedin inf.order-iff range'-def)
qed

lemma *c-range-nonempty*:
assumes $f \in C(X)$
and *topspace X* $\neq \{\}$
shows *range' X f* $\neq \{\}$
proof −
have *f* ' *topspace X* $\neq \{\}$ **using** *assms* **by** *blast*
thus *?thesis* **unfolding** *range'-def* **by** *simp*
qed

lemma *cstar-range-nonempty*:
assumes $f \in C*\ X$
and *topspace X* $\neq \{\}$
shows *range' X f* $\neq \{\}$

using *assms c-range-nonempty*[*of f X*]
by *auto*

lemma *cstar-separates-tych-space*:
assumes *tych-space X*
shows *funcset-separates-points-from-closed-sets X* (*cstar-id X*) ($\lambda f \in C* \ X$. *euclideanreal*) (*C∗ X*)
∧ *funcset-separates-points X* (*cstar-id X*) (*C∗ X*)
proof −
{ fix *x S* **assume** *closedin X S* ∧ *x* ∈ *topspace X* − *S*
then obtain *f*
where *fprops*: *continuous-map X* (*top-of-set* {*0*..(*1*::*real*)}) *f* ∧ *f x* = *0* ∧ *f* ‘ *S* ⊆ {*1*}
using *assms completely-regular-space-def*[*of X*]
by *presburger*
hence *f* ∈ *C X*
using *continuous-map-into-fulltopology*[*of X euclideanreal* {*0*..(*1*::*real*)} *f*]
by *auto*
moreover have *fbounded f X*
proof −
have ∀ *x* ∈ *topspace X* . *0* ≤ *f x* ∧ *f x* ≤ *1* **using** *fprops*
by (*simp add*: *continuous-map-in-subtopology image-subset-iff*)
thus *?thesis* **by** *auto*
qed
ultimately have *f-in-cstar*: *f* ∈ (*C∗ X*) **by** *auto*

moreover have *f-separates*: *f x* ∉ (*euclideanreal closure-of* (*f* ‘ *S*))
proof −
have *closedin euclideanreal* (*f* ‘ *S*)
by (*metis closed-closedin closed-empty closed-singleton fprops subset-singletonD*)
moreover have *f x* ∉ *f* ‘ *S* **using** *fprops* **by** *auto*
thus *?thesis* **using** *calculation* **by** *auto*
qed
ultimately have ∃ *f* ∈ *C∗ X* . *f x* ∉ *euclideanreal closure-of* (*f* ‘ *S*) **by** *auto*
}
hence *rtp1*: *funcset-separates-points-from-closed-sets X* (*cstar-id X*) ($\lambda f \in C*$ *X*. *euclideanreal*) (*C∗ X*)
using *cstar-id-def*[*of X*] **unfolding** *funcset-separates-points-from-closed-sets-def* **by** *auto*

moreover have *funcset-separates-points X* (*cstar-id X*) (*C∗ X*)
proof −
{ fix *x y* **assume** {*x*,*y*} ⊆ *topspace X* ∧ *x* ≠ *y*
hence *closedin X* {*y*} ∧ *x* ∈ *topspace X* − {*y*}
using *assms* **by** (*simp add*: *t1-space-closedin-finite*)
hence ∃*f* ∈ *C∗ X* . *cstar-id X f x* ∉ (λ*f*∈ *C∗ X* . *euclideanreal*) *f closure-of cstar-id X f* ‘ {*y*}
using *funcset-separates-points-from-closed-sets-def*[*of X cstar-id X* λ *f* ∈

$C*\ X$. *euclideanreal* $C*\ X$]
rtp1 **by** *presburger*
hence $\exists\ f \in C*\ X\ .\ f\ x \neq f\ y$
using *cstar-id-def*[*of X*] *t1-space-closedin-finite*[*of euclideanreal*] **by** *auto*
}
thus *?thesis* **using** *cstar-id-def*[*of X*] **unfolding** *funcset-separates-points-def*
by *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

The product topology induced by $C^*(X)$ on a Tychonov space.

definition *scT* :: *'a topology* ⇒ (*'a* ⇒ *real*) ⇒ *real topology*
where *scT X* = (λ *f* ∈ *C∗ X* . *subtopology euclideanreal* (*range' X f*))

definition *scT-full* :: *'a topology* ⇒ (*'a* ⇒ *real*) ⇒ *real topology*
where *scT-full X* = (λ *f* ∈ *C∗ X* . *euclideanreal*)

definition *scProduct* :: *'a topology* ⇒ ((*'a* ⇒ *real*) ⇒ *real*) *topology*
where *scProduct X* = *product-topology* (*scT X*) (*C∗ X*)

definition *scProject* :: *'a topology* ⇒ (*'a* ⇒ *real*) ⇒ ((*'a* ⇒ *real*) ⇒ *real*) ⇒ *real*
where *scProject X* = *product-projection* (*scT X*) (*C∗ X*)

definition *scEmbed* :: *'a topology* ⇒ *'a* ⇒ (*'a* ⇒ *real*) ⇒ *real*
where *scEmbed X* = *evaluation-map X* (*cstar-id X*) (*C∗ X*)

lemma *scT-images-compact-Hausdorff*:
shows ∀ *f* ∈ *C∗ X* . *compact-Hausdorff* (*scT X f*)
proof −
have *T*: ∀ *f* ∈ *C∗ X* . *scT X f* = *subtopology euclideanreal* (*range' X f*)
unfolding *scT-def* **by** *simp*
thus *?thesis* **using** *range'-def*[*of X f*]
by (*simp add*: *Hausdorff-space-subtopology compact-range' compact-space-subtopology*)
qed

lemma *scT-images-bounded*:
shows ∀ *f* ∈ *C∗ X* . *bounded* (*topspace* (*scT X f*))
using *scT-images-compact-Hausdorff*[*of X*] *scT-def*[*of X*]
by (*simp add*: *compact-imp-bounded compact-range'*)

lemma *scProduct-compact-Hausdorff*:
shows *compact-Hausdorff* (*scProduct X*)
unfolding *scProduct-def* **using** *scT-images-compact-Hausdorff*[*of X*]
using *compact-space-product-topology*

by (*metis* (*no-types*, *lifting*) *compact-Hausdorff-imp-regular-space regular-space-product-topology*

regular-t0-eq-Hausdorff-space t0-space-product-topology)

The Stone-Čech compactification of a Tychonov space and its extension properties

lemma *tych-space-weak*:
assumes *tych-space X*
shows X = *weak-topology* (*topspace X*) (*cstar-id X*) (*scT X*) ($C*$ X)
proof (*cases topspace X* = {})
case *True*
then show *?thesis*
using *weak-topology-on-empty*[*of weak-topology* (*topspace X*) (*cstar-id X*) (*scT X*) ($C*$ X)]
topology-eq **by** *fastforce*
next
case *False*
define *W* **where** *W* = *weak-topology* (*topspace X*) (*cstar-id X*) (*scT X*) ($C*$ X)

hence *topspace W* = *topspace X*
using *cstar-types-restricted*[*of X*] *scT-def*[*of X*] *W-def cstar-nonempty*[*of X*]
weak-topology-topspace[*of W topspace X cstar-id X scT X* $C*$ *X*]
by *auto*
moreover have $\forall f \in C*\ X$. *continuous-map X* (*scT X f*) *f*
unfolding *scT-def range′-def*
by (*metis* (*mono-tags*, *lifting*) *closure-of-subset continuous-map-image-subset-topspace*

continuous-map-in-subtopology mem-Collect-eq restrict-apply′)
ultimately have $\forall$ *U* . *openin W U* $\longrightarrow$ *openin X U*
using *W-def cstar-types-restricted*[*of X*] *scT-def*[*of X*] *cstar-id-def*[*of X*]
weak-topology-is-weakest[*of W* (*topspace X*) (*cstar-id X*) (*scT X*) $C*$ *X X*]
by (*smt* (*verit*, *ccfv-threshold*) *restrict-apply′*)

moreover have $\forall$ *U* . *openin X U* $\longrightarrow$ *openin W U*
proof −
{ fix *U* **assume** *props*: *openin X U*
{ fix *x* **assume** *xprops*: $x \in U$
hence *x-in-X*: $x \in$ *topspace X*
using *openin-subset props* **by** *fastforce*

define *S* **where** *S* = *topspace X* − *U*
hence *props′*: $x \in$ *topspace X* − *S* $\wedge$ *closedin X S*
using *props openin-closedin-eq xprops* **by** *fastforce*
then obtain *f* **where** *fprops*: *continuous-map X* (*top-of-set* {*0..1*::*real*}) *f* $\wedge$ *f x* = *0* $\wedge$ *f* ‘ *S* $\subseteq$ {*1*}
using *assms*(*1*) *completely-regular-space-def*[*of X*]
by *meson*

then obtain *ffull*
where *ffullprops*: $(ffull \in C\ X) \wedge ffull\ x = (0::real) \wedge ffull\ ‘\ S \subseteq \{1\}$
using *continuous-map-into-fulltopology*
by (*metis mem-Collect-eq*)

define F **where** $F = fbound\ ffull\ 0\ 1$
hence *Fcstar*: $F \in C{*}\ X$ **using** *ffullprops fbound-cstar*[*of ffull X 0 1*] **by** *auto*
hence *Ftype*: $F \in topspace\ X \to topspace\ euclideanreal$
unfolding *continuous-map-def* **by** *auto*

define I **where** $I = \{(-1) <..< 1::real\}$
hence *Iprops*: *openin euclideanreal I*
by (*simp add*: *openin-delete*)

define V **where** $V = inverse'\ F\ (topspace\ X)\ I$

have *crprops*: $F\ x = 0 \wedge F\ ‘\ S \subseteq \{1\}$
using *ffullprops F-def*
unfolding *fbound-def fmid-def fmin-def fmax-def min-def max-def*
by *auto*

hence $V \subseteq U$
proof −
{ fix v **assume** $v \in V$
hence $v \in topspace\ X \wedge F\ v \in I$ **unfolding** *inverse′-def V-def* **by** *auto*
hence $v \in U$ **using** *S-def crprops I-def* **by** *auto*
}
thus *?thesis* **by** *auto*
qed
moreover have $x \in V$
using *crprops I-def x-in-X* **unfolding** *inverse′-def V-def* **by** *auto*
moreover have *openin W V*
proof −
have $V \in$ *open-sets-induced-by-func F* (*topspace X*) *euclideanreal*
unfolding *open-sets-induced-by-func-def* **using** *Ftype V-def Iprops*
by *blast*
moreover have *open-sets-induced-by-func F* (*topspace X*) *euclideanreal* $\subseteq$
weak-generators (*topspace X*) (*cstar-id X*) (*scT-full X*) ($C{*}$ *X*)
using *weak-generators-def*[*of topspace X* (*cstar-id X*) *scT-full X C∗ X*]
scT-full-def[*of X*] *cstar-id-def*[*of X*] *Fcstar*
by (*smt* (*verit*, *ccfv-SIG*) *Sup-upper mem-Collect-eq restrict-apply′*)
ultimately have $V \in$ *weak-generators* (*topspace X*) (*cstar-id X*) (*scT-full X*) ($C{*}$ *X*)
by *auto*
hence *openin* (*topology-generated-by* (*weak-generators* (*topspace X*) (*cstar-id X*) (*scT-full X*) ($C{*}$ *X*))) *V*
using *generators-are-open*[*of weak-generators* (*topspace X*) (*cstar-id X*) (*scT-full X*) (*C∗ X*)]

topology-generated-by-Basis **by** *blast*
thus *?thesis*
using *W-def weak-restricted-topology-eq-weak*[*of X*]
unfolding *scT-def scT-full-def weak-topology-def*
by *simp*
qed
ultimately have $x \in V \wedge V \subseteq U \wedge$ *openin W V* **by** *auto*
hence $\exists\ V .\ x \in V \wedge V \subseteq U \wedge$ *openin W V* **by** *auto*
}
hence $\forall\ x \in U .\ \exists\ V .\ x \in V \wedge V \subseteq U \wedge$ *openin W V* **by** *blast*
hence *openin W U* **by** (*meson openin-subopen*)
}
thus *?thesis* **by** *auto*
qed
ultimately have $\forall\ U$. *openin X U* $\longleftrightarrow$ *openin W U* **by** *auto*
hence $X = W$ **by** (*simp add*: *topology-eq*)
thus *?thesis* **using** *W-def* **by** *simp*
qed

4.1 Definition of βX

definition *scEmbeddedCopy* :: *'a topology* $\Rightarrow$ ((*'a* $\Rightarrow$ *real*) $\Rightarrow$ *real*) *set*
where *scEmbeddedCopy X* = *scEmbed X* ‘ *topspace X*

definition *scCompactification* :: *'a topology* $\Rightarrow$ ((*'a* $\Rightarrow$ *real*) $\Rightarrow$ *real*) *topology* (β -)
where *scCompactification X*
= *subtopology* (*scProduct X*) ((*scProduct X*) *closure-of* (*scEmbeddedCopy X*))

lemma *sc-topspace*:
shows *topspace* (β *X*) = (*scProduct X*) *closure-of* (*scEmbeddedCopy X*)
using *scCompactification-def*[*of X*] *closure-of-subset-topspace* **by** *force*

lemma *scProject'*:
shows $\forall\ f \in C*\ X .\ \forall\ p \in$ *topspace* (β *X*) . *scProject X f p* = *p f*
proof −
have *topspace* (β *X*) $\subseteq$ *topspace* (*scProduct X*) **unfolding** *scCompactification-def* **by** *auto*
thus *?thesis*
unfolding *scProject-def product-projection-def scProduct-def*
by *auto*
qed

Evaluation densely embeds Tychonov X in βX

lemma *dense-embedding-scEmbed*:
assumes *tych-space X*
shows *dense-embedding X* (β *X*) (*scEmbed X*)
proof −
define *W* **where** *W* = *weak-topology* (*topspace X*) (*cstar-id X*) ($\lambda f \in C*\ X.$

euclideanreal) (*C*∗ *X*)
hence *X* = *W* **using** *assms tych-space-weak*[*of X*]
by (*metis* (*mono-tags*, *lifting*) *scT-def weak-restricted-topology-eq-weak*)

hence *Xweak*: *X* = *weak-topology* (*topspace X*) (*cstar-id X*) (*scT X*) (*C*∗ *X*)
using *scT-def*[*of X*] *W-def cstar-id-def*[*of X*]
weak-restricted-topology-eq-weak[**where** *X* =*X*] **by** *auto*
moreover have *scProduct X* = *product-topology* (*scT X*) (*C*∗ *X*) **using** *scProduct-def*[*of X*] **by** *auto*
moreover have *funcset-types* (*topspace X*) (*cstar-id X*) (*scT X*) (*C*∗ *X*)
unfolding *scT-def* **using** *cstar-types-restricted*[*of X*] **by** *auto*
moreover have *funcset-separates-points X* (*cstar-id X*) (*C*∗ *X*)
using *cstar-separates-tych-space*[*of X*] *assms*(*1*) **by** *auto*
moreover have (*C*∗ *X*) ≠ {} **using** *cstar-nonempty* **by** *auto*
ultimately have *embedding-map X* (*scProduct X*) (*scEmbed X*)
using *evaluation-is-embedding*[*of X cstar-id X scT X C*∗ *X scProduct X*]
unfolding *scProduct-def scEmbed-def*
by *auto*
hence *embeds*: *embedding-map X* (*β X*) (*scEmbed X*)
unfolding *scCompactification-def*
by (*metis closure-of-subset embedding-map-in-subtopology scEmbeddedCopy-def subtopology-topspace*)
moreover have *dense-in* (*β X*) (*scEmbed X ‘ topspace X*) (*topspace* (*β X*))
unfolding *dense-in-def* **using** *scCompactification-def*[*of X*] *scEmbeddedCopy-def*[*of X*]
by (*metis Int-absorb1 closure-of-subset closure-of-subset-topspace closure-of-subtopology*

embedding-map-in-subtopology embeds set-eq-subset subtopology-topspace

topspace-subtopology-subset)
ultimately show *?thesis* **by** *auto*
qed

4.2 βX is a compactification of X

lemma *scCompactification-compact-Hausdorff*:
assumes *tych-space X*
shows *compact-Hausdorff* (*β X*)
using *scCompactification-def*[*of X*] *scProduct-compact-Hausdorff*[*of X*]
by (*simp add*: *Hausdorff-space-subtopology closedin-compact-space compact-space-subtopology*)

lemma *scCompactification-is-compactification-via-scEmbed*:
assumes *tych-space X*
shows *compactification-via* (*scEmbed X*) *X* (*β X*)
using *compactification-via-def*[*of scEmbed X X β X*]
scCompactification-compact-Hausdorff[*of X*]
dense-embedding-scEmbed[*of X*] *assms*
by *auto*

lemma *scCompactification-is-compactification*:
assumes *tych-space X*
shows *compactification X* (β *X*)
using *assms compactification-compactification-via*
scCompactification-is-compactification-via-scEmbed
by *blast*

lemma *scEvaluation-range*:
assumes $x \in$ *topspace X*
and *tych-space X*
shows (λ $f \in C{*}$ *X* . *f x*) $\in$ *topspace* (*product-topology* (*scT X*) *C*∗ *X*)
proof −
have *funcset-types* (*topspace X*) (*cstar-id X*) ($\lambda f\in C{*}$ *X* . *top-of-set* (*range′ X f*)) *C*∗ *X*
using *cstar-types-restricted*[*of X*] **by** *auto*
hence $\forall f\in C{*}$ *X* . *f* $\in$ *topspace X* $\rightarrow$ *topspace* (*scT X f*)
unfolding *funcset-types-def scT-def cstar-id-def*[*of X*] **by** *auto*
thus *?thesis* **using** *topspace-product-topology*[*of scT X C*∗ *X*] *assms*(*1*) **by** *auto*
qed

lemma *scEmbed-then-project*:
assumes $f \in C{*}$ *X*
and $x \in$ *topspace X*
and *tych-space X*
shows *scProject X f* (*scEmbed X x*) = *f x*
proof −
have *fequiv*: $\forall$ $y \in$ *topspace X* . (λ $g \in C{*}$ *X* . (*cstar-id X*) *g y*) = (λ $g \in C{*}$ *X* . *g y*)
proof −
{ fix *y* **assume** *yprops*: $y \in$ *topspace X*
hence $\forall$ $g \in C{*}$ *X* . (*cstar-id X*) *g y* = *g y* **unfolding** *cstar-id-def* **by** *auto*
hence (λ $g \in C{*}$ *X* . (*cstar-id X*) *g y*) = (λ $g \in C{*}$ *X* . *g y*)
by (*meson restrict-ext*)
}
thus *?thesis* **by** *auto*
qed

have *scProject X f* (*scEmbed X x*) = *scProject X f* (*evaluation-map X* (*cstar-id X*) (*C*∗ *X*) *x*)
unfolding *scEmbed-def* **by** *auto*
also have ... = *scProject X f* ($\lambda g\in C{*}$ *X* . *g x*)
unfolding *evaluation-map-def* **using** *assms*(*2*) *fequiv* **by** *auto*
also have ... = ($\lambda g\in C{*}$ *X*. $\lambda p\in$*topspace* (*product-topology* (*scT X*) (*C*∗ *X*)). *p g*) *f* ($\lambda g\in C{*}$ *X* . *g x*)
unfolding *product-projection-def scProject-def* **by** *auto*
also have ... = ($\lambda p\in$*topspace* (*product-topology* (*scT X*) (*C*∗ *X*)). *p f*) ($\lambda g\in C{*}$ *X* . *g x*)
using *assms*(*1*) **by** *auto*
also have ... = *f x* **using** *scEvaluation-range*[*of x X*] *assms* **by** *auto*

ultimately show *?thesis* **by** *auto*
qed

4.3 Evaluation is a C^*-embedding of X into βX

definition *scExtend* :: *'a topology* ⇒ (*'a* ⇒ *real*) ⇒ ((*'a* ⇒ *real*) ⇒ *real*) ⇒ *real*
where *scExtend X* = (λ *f* ∈ *C∗ X* . *restrict* (*scProject X f*) (*topspace* (β *X*)))

proposition *scExtend-extends*:
assumes *tych-space X*
shows ∀ *f* ∈ *C∗ X* . ∀ *x* ∈ *topspace X* . *f x* = (*scExtend X f*) (*scEmbed X x*)
proof −
{ fix *f* **assume** *fprops*: *f* ∈ *C∗ X*
have ∀ *x* ∈ *topspace X* . (*scProject X f*) (*scEmbed X x*) = (*scExtend X f*) (*scEmbed X x*)
proof −
{ fix *x* **assume** *xprops*: *x* ∈ *topspace X*
define *p* **where** *pprops*: *p* = *scEmbed X x*

hence *scExtend X f p* = (*restrict* (*scProject X f*) (*topspace* β *X*)) *p*
using *xprops fprops* **unfolding** *scExtend-def* **by** *auto*
moreover have *p* ∈ *topspace* β *X*
using *assms*(*1*) *pprops dense-embedding-scEmbed*[*of X*]
scCompactification-def[*of X*] *scEmbeddedCopy-def*[*of X*]
by (*metis* (*no-types*, *lifting*) *embedding-map-in-subtopology image-eqI*
in-mono subtopology-topspace xprops)
ultimately have *scExtend X f p* = *scProject X f p*
using *pprops scEmbeddedCopy-def*[*of X*] *scEmbed-def*[*of X*] *evaluation-map-def* **by** *auto*
}
thus *?thesis* **by** *auto*
qed
hence ∀ *x* ∈ *topspace X* . *f x* = (*scExtend X f*) (*scEmbed X x*)
using *scEmbed-then-project*[*of f X*] *assms*(*1*) *fprops* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *scExtend-extends-cstar*:
assumes *tych-space X*
shows ∀ *f* ∈ *C∗ X* . (∀ *x* ∈ *topspace X* . *f x* = (*scExtend X f*) (*scEmbed X x*)) ∧ *scExtend X f* ∈ *C∗* (β *X*)
proof −
define *e* **where** *e* = *scExtend X*
{ fix *f* **assume** *fprops*: *f* ∈ *C∗ X*
hence *continuous-map* (*scProduct X*) (*scT X f*) (*scProject X f*)
using *scProduct-def*[*of X*] *scProject-def*[*of X*]
projections-continuous[*of scProduct X scT X C∗ X scProject X*]

product-projection-def[*of scT X C∗ X*]
by (*metis* (*no-types*, *lifting*) *restrict-extensional extensional-restrict*)
hence *continuous-map* (β *X*) (*scT X f*) (*scProject X f*)
by (*simp add*: *continuous-map-from-subtopology scCompactification-def*)
hence *c-embedded-f*: *continuous-map* (β *X*) (*scT X f*) (*scExtend X f*)
using *scExtend-def*[*of X*] *fprops* **by** *force*
moreover have *fbounded-f*:*fbounded* (*scExtend X f*) (β *X*)
proof −
obtain *m M* **where** *f ‘ topspace X* ⊆ {*m..M*} **using** *fprops* **by** *force*
hence *extend-on-embedded*: *e f ‘* (*scEmbeddedCopy X*) ⊆ {*m..M*}
using *scExtend-extends*[*of X*] *e-def*
by (*smt* (*verit*, *ccfv-SIG*) *fprops assms*(*1*) *image-cong image-image scEmbeddedCopy-def*)
hence *e f ‘* (*topspace* (β *X*)) ⊆ {*m..M*}
proof −
{ fix *p* **assume** *pprops*: *p* ∈ *e f ‘* (*topspace* (β *X*))
then obtain *v* **where** *vprops*: *v* ∈ *topspace* (β *X*) ∧ *p* = *e f v* **by** *auto*
{ fix *U* **assume** *Uprops*: *openin* (*scT X f*) *U* ∧ *p* ∈ *U*
define *V* **where** *V* = *inverse′* (*e f*) (*topspace* (β *X*)) *U*
hence *openin* (β *X*) *V*
using *c-embedded-f Uprops e-def* **unfolding** *continuous-map-def inverse′-def*
by *auto*
moreover have *topspace* (β *X*) = (β *X*) *closure-of scEmbeddedCopy X*
using *scCompactification-def*[*of X*] *closure-of-subset-topspace*[*of* β *X scEmbeddedCopy X*]
dense-embedding-scEmbed[*of X*] *scEmbeddedCopy-def*[*of X*]
by (*metis assms closure-of-subtopology-open embedding-map-in-subtopology*

subtopology-topspace topspace-subtopology)
moreover have *v* ∈ *V* ∧ *v* ∈ *topspace* (β *X*)
using *vprops V-def Uprops* **unfolding** *inverse′-def* **by** *auto*
ultimately obtain *x* **where** *xprops*: *x* ∈ *scEmbeddedCopy X* ∧ *x* ∈ *V*
using *in-closure-of*[*of v* β *X scEmbeddedCopy X*]
by *presburger*
define *w* **where** *w* = *e f x*
hence *w* ∈ {*m..M*} **using** *extend-on-embedded xprops* **by** *blast*
moreover have *w* ∈ *U* **using** *w-def xprops vprops V-def*
by (*simp add*: *inverse′-alt*)
ultimately have ∃ *w*. *w* ∈ *U* ∩ {*m..M*} **by** *auto*
}
hence ∀ *U* . *openin* (*scT X f*) *U* ∧ *p* ∈ *U* ⟶ (∃ *w*. *w* ∈ *U* ∩ {*m..M*}) **by** *auto*
moreover have *p* ∈ *topspace* (*scT X f*)
by (*metis e-def Int-iff vprops c-embedded-f continuous-map-preimage-topspace vimageE*)
ultimately have *p* ∈ (*scT X f*) *closure-of* {*m..M*}
using *in-closure-of*[*of p scT X f* {*m..M*}]
by *auto*

hence *p* ∈ *euclideanreal closure-of* {*m..M*}
using *scT-def*[*of X*] *range'-def*[*of X f*]
by (*metis* (*no-types*, *lifting*) *closure-of-subtopology-subset fprops restrict-apply' subsetD*)
hence *p* ∈ {*m..M*} **by** *auto*
}
thus *?thesis* **by** *auto*
qed
thus *?thesis* **by** (*metis e-def atLeastAtMost-iff image-subset-iff*)
qed
ultimately have *scExtend X f* ∈ *C∗* (*β X*)
using *scT-def*[*of X*] *continuous-map-into-fulltopology fprops* **by** *auto*
}
hence ∀ *f* ∈ *C∗ X* . *scExtend X f* ∈ *C∗* (*β X*) **by** *auto*
thus *?thesis* **using** *assms scExtend-extends* **by** *blast*
qed

lemma *cstar-embedding-scEmbed*:
assumes *tych-space X*
shows *cstar-embedding X* (*β X*) (*scEmbed X*)
using *assms scExtend-extends-cstar*[*of X*] *dense-embedding-scEmbed*[*of X*]
by *meson*

A compact Hausdorff space is its own Stone-Cech compactification

lemma *scCompactification-of-compact-Hausdorff*:
assumes *compact-Hausdorff X*
shows *homeomorphic-map X* (*β X*) (*scEmbed X*)
proof −
have *dense*: *dense-embedding X* (*β X*) (*scEmbed X*)
by (*simp add*: *assms compact-Hausdorff-imp-tych dense-embedding-scEmbed*)
moreover have *closed*: *closed-map X* (*β X*) (*scEmbed X*)
by (*meson T1-Spaces.continuous-imp-closed-map assms compact-Hausdorff-imp-tych*

continuous-map-in-subtopology embedding-map-def dense
homeomorphic-eq-everything-map scCompactification-compact-Hausdorff)
moreover have *open-map X* (*β X*) (*scEmbed X*)
by (*metis closed closure-of-subset-eq dense-in-def embedding-imp-closed-map-eq*

embedding-map-def homeomorphic-imp-open-map local.dense subtopology-superset)
thus *?thesis*
by (*metis closed closure-of-subset-eq dense-in-def embedding-imp-closed-map-eq*

embedding-map-def local.dense subtopology-superset)
qed

4.4 The Stone-Čech Extension Property: Any continuous map from X to a compact Hausdorff space K extends uniquely to a continuous map from βX to K.

proposition *gof-cstar*:
  assumes *compact-Hausdorff K*
and *continuous-map X K f*
shows $\forall$ *g* $\in$ *C∗ K* . (*g o f*) $\in$ *C∗ X*
proof −
  have *tych-K*: *tych-space K*
    using *assms*(*1*) *compact-Hausdorff-imp-tych* **by** *auto*

  { fix *g* **assume** *gprops*: *g* $\in$ *C∗ K*
    have *continuous-map K* (*scT K g*) *g*
      using *scT-def*[*of K*] *range′-def*[*of K g*] *cstar-types-restricted*[*of K*] *assms*(*2*)
          gprops weak-generators-continuous[*of K topspace K cstar-id K* (*scT K*)
(*C∗ K*) *g*]
     by (*metis* (*mono-tags*, *lifting*) *closure-of-topspace continuous-map-image-closure-subset*

          continuous-map-in-subtopology mem-Collect-eq restrict-apply′)
    hence *cts-scT*: *continuous-map X* (*scT K g*) (*g o f*)
      using *assms* **by** (*simp add*: *continuous-map-compose*)
    hence *gofprops*: (*g o f*) $\in$ (*C X*)
      using *scT-def*[*of K*] *range′-def*[*of K*]
     by (*metis* (*mono-tags*, *lifting*) *continuous-map-in-subtopology gprops mem-Collect-eq*
restrict-apply′)
    moreover have *fbounded* (*g o f*) *X*
    proof −
      have *compact* (*g* ‘ *topspace K*) **using** *assms*(*1*) *gprops*
        using *compact-space-def compactin-euclidean-iff image-compactin* **by** *blast*
      hence *bounded* (*g* ‘ *topspace K*)
        by (*simp add*: *compact-imp-bounded*)
      moreover have (*g o f*) ‘ *topspace X* $\subseteq$ *g* ‘ *topspace K*
        by (*metis assms*(*2*) *continuous-map-image-subset-topspace image-comp im-
age-mono*)
      ultimately have *bounded* ((*g o f*) ‘ *topspace X*)
        by (*metis bounded-subset*)
      thus *?thesis* **using** *bounded-range-iff-fbounded*[*of g o f X*] *gofprops* **by** *auto*
    qed
    ultimately have (*g o f*) $\in$ *C∗ X* **by** *auto*
  }
  thus *?thesis* **by** *auto*
qed

proposition *scEmbed-range*:
  assumes *tych-space X*
and *x* $\in$ *topspace X*
shows *scEmbed X x* $\in$ *topspace* (β *X*)
 using *assms*(*1*) *assms*(*2*) *dense-embedding-scEmbed embedding-map-in-subtopology*

by *fastforce*

proposition *scEmbed-range'*:
  assumes *tych-space X*
and $x \in$ *topspace X*
shows *scEmbed X x* $\in$ *topspace* (*scProduct X*)
  using *assms*(*1*) *assms*(*2*) *scEmbed-range*[*of X*]
  by (*simp add*: *scCompactification-def*)

proposition *scProjection*:
  shows $\forall$ *f* $\in$ *C*∗ *X*. $\forall$ *p* $\in$ *topspace* (*scProduct X*) . *scProject X f p* = *p f*
  using *scProject-def*[*of X*] *scProduct-def*[*of X*] *product-projection*[*of C*∗ *X scT X*]
  by *simp*

proposition *scProjections-continuous*:
  shows $\forall$ *f* $\in$ *C*∗ *X* . *continuous-map* (*scProduct X*) (*scT X f*) (*scProject X f*)
proof −
  have $\forall$ *f* $\in$ *C*∗ *X* . *continuous-map* (*scProduct X*) (*scT X f*) (*scProject X f*)
    using *scProduct-def*[*of X*] *scProject-def*[*of X*]
    by (*metis* (*mono-tags*, *lifting*) *projections-continuous restrict-apply'*)
  thus *?thesis* **using** *scCompactification-def*[*of X*] **by** *simp*
qed

proposition *continuous-embedding-inverse*:
  assumes *embedding-map X Y e*
  shows $\exists$ *e'* . *continuous-map* (*subtopology Y* (*e* ‘ *topspace X*)) *X e'* $\wedge$ ($\forall$ *x* $\in$ *topspace X* . *e'* (*e x*) = *x*)
 by (*meson assms embedding-map-def homeomorphic-map-maps homeomorphic-maps-def*)

lemma *scExtension-exists*:
  assumes *tych-space X*
and *compact-Hausdorff K*
shows $\forall$ *f* $\in$ *cts*[*X*,*K*] . $\exists$ *F* $\in$ *cts*[β *X*, *K*] . ($\forall$ *x* $\in$ *topspace X* . *F* (*scEmbed X x*) = *f x*)
proof−
  { fix *f* **assume** *fprops*: *f* $\in$ *cts*[*X*,*K*]

    have *tych-K*: *tych-space K* **using** *assms*(*2*) *compact-Hausdorff-imp-tych*[*of K*] **by** *simp*

    define *Xspace* **where** *Xspace* = *topspace* (*scProduct X*)
    define *Kspace* **where** *Kspace* = *topspace* (*scProduct K*)

    define *H* **where** *H* = (λ *p* $\in$ *Xspace* . λ *g* $\in$ *C*∗ *K* . *scProject X* (*g o f*) *p*)

have *H-of-scEmbed*: ∀ $x \in$ *topspace* X . H (*scEmbed X x*) = *scEmbed K* (*f x*)
proof −
{ **fix** x **assume** *xprops*: $x \in$ *topspace* X
hence H (*scEmbed X x*) = (λ $p \in$ *Xspace* . λ $g \in C*$ K . *scProject X* (*g o f*) *p*) (*scEmbed X x*)
using *H-def* **by** *auto*
moreover have (*scEmbed X x*) $\in$ *Xspace*
using *Xspace-def assms*(*1*) *scEmbed-range′*[*of X x*] *xprops* **by** *auto*
ultimately have H (*scEmbed X x*) = (λ $g \in C*$ K . *scProject X* (*g o f*) (*scEmbed X x*))
by *auto*
also have ... = (λ $g \in C*$ K . (*g o f*) *x*)
using *assms*(*2*) *gof-cstar*[*of K X f*] *xprops fprops assms*(*1*)
scEmbed-then-project[**where** *x*=*x* **and** *X*=*X*]
by (*metis* (*no-types, lifting*) *mem-Collect-eq restrict-ext*)
also have ... = (λ $g \in C*$ K . *g* (*f x*)) **by** *auto*
finally have H (*scEmbed X x*) = *scEmbed K* (*f x*)
using *scEmbed-def*[*of K*] *cstar-id-def*[*of K*] *evaluation-map-def*[*of K cstar-id K C∗ K*]
by (*smt* (*verit*) *continuous-map-image-subset-topspace fprops xprops image-subset-iff*
mem-Collect-eq restrict-apply′ restrict-ext xprops)
}
thus *?thesis* **by** *auto*
qed
hence *H-on-embedded*: H ‘ *scEmbeddedCopy X* ⊆ *scEmbeddedCopy K*
proof −
{ **fix** p **assume** $p \in H$ ‘ *scEmbeddedCopy X*
then obtain q **where** *qprops*: $q \in$ *scEmbeddedCopy X* ∧ $p = H\ q$ **by** *auto*
then obtain x **where** *xprops*: $x \in$ *topspace* X ∧ q = *scEmbed X x*
using *scEmbeddedCopy-def*[*of X*] **by** *auto*
hence p = *scEmbed K* (*f x*) **using** *qprops H-of-scEmbed* **by** *auto*
hence $p \in$ *scEmbeddedCopy K*
using *scEmbeddedCopy-def*[*of K*] *xprops qprops fprops*
by (*metis continuous-map-image-subset-topspace image-eqI in-mono mem-Collect-eq*)
}
thus *?thesis* **by** *auto*
qed

have *components-cts*: ∀ $g \in C*$ K . *continuous-map* (*scProduct X*) (*scT K g*) ($\lambda x \in$ *Xspace* . *H x g*)
proof −
{ **fix** g **assume** *gprops*: $g \in C*$ K

have *continuous-map* (*scProduct X*) (*scT X* (*g o f*)) (λ $x \in$ *Xspace* . *H x g*)

proof −
have $\forall f\in C*\ X$. *continuous-map* (*scProduct X*) (*scT X f*) (*scProject X f*)
using *scProjections-continuous*[*of X*] **by** *simp*
hence *continuous-map* (*scProduct X*) (*scT X* (*g o f*)) (*scProject X* (*g o f*))
using *assms*(*2*) *fprops gprops gof-cstar*[*of K X f*] **by** *auto*
moreover have $\forall x \in$ *Xspace*. *H x g* = (*scProject X* (*g o f*)) *x*
using *gprops H-def Xspace-def*
by *auto*
ultimately show *?thesis*
using *Xspace-def continuous-map-eq* **by** *fastforce*
qed

moreover have *scT X* (*g o f*) = *subtopology* (*scT K g*) (*range′ X* (*g o f*))
proof −
have (*g o f*) ‘ *topspace X* ⊆ *g* ‘ *topspace K*
using *gprops fprops* **unfolding** *continuous-map-def* **by** *auto*
hence *range′ X* (*g o f*) ⊆ *range′ K g*
unfolding *range′-def* **by** (*meson closure-of-mono*)
hence *top-of-set* (*range′ X* (*g o f*))
= *subtopology* (*top-of-set* (*range′ K g*)) (*range′ X* (*g o f*))
by (*simp add*: *inf.absorb-iff2 subtopology-subtopology*)
hence *scT X* (*g o f*) = *subtopology* (*scT K g*) (*range′ X* (*g o f*))
using *scT-def*[*of X*] *scT-def*[*of K*] *gprops assms*(*2*) *gof-cstar*[*of K X f*] *fprops* **by** *auto*
thus *?thesis* **by** *auto*
qed
ultimately have *continuous-map* (*scProduct X*) (*scT K g*) (λ *x* ∈ *Xspace* . *H x g*)
using *continuous-map-in-subtopology* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

hence *Hcts*: *continuous-map* (*scProduct X*) (*scProduct K*) *H*
using *continuous-map-coordinatewise-then-product*[*of C∗ K scProduct X scT K H*]
scProduct-def[*of X*] *scProduct-def*[*of K*] *H-def Xspace-def*
by (*smt* (*verit*, *del-insts*) *continuous-map-eq restrict-apply*)

have *H-on-beta*: *H* ‘ *topspace* (*β X*) ⊆ *scEmbeddedCopy K*
proof −
have *H* ‘ *scEmbeddedCopy X* ⊆ *scEmbeddedCopy K* **using** *H-on-embedded* **by** *auto*
hence *H* ‘ *topspace* (*β X*) ⊆ *scProduct K closure-of scEmbeddedCopy K*
using *scCompactification-def*[*of X*] *Hcts closure-of-mono*
continuous-map-eq-image-closure-subset **by** *fastforce*
thus *?thesis* **using** *scEmbeddedCopy-def*

by (*metis assms*(*2*) *closure-of-subset-topspace homeomorphic-imp-surjective-map*

scCompactification-def scCompactification-of-compact-Hausdorff topspace-subtopology-subset)
qed

have *embeds*: *dense-embedding K* (β *K*) (*scEmbed K*) **using** *dense-embedding-scEmbed*[*of K*] *tych-K* **by** *auto*
have *closed*: *closedin* (*scProduct K*) (*scEmbeddedCopy K*)
using *assms*(*2*) *scEmbeddedCopy-def*[*of X*] *scCompactification-def*[*of K*]
scCompactification-compact-Hausdorff[*of K*]
by (*metis closure-of-eq closure-of-subset-topspace closure-of-topspace dense-in-def embeds*
homeomorphic-map-closure-of scCompactification-of-compact-Hausdorff scEmbeddedCopy-def
topspace-subtopology-subset)
hence *onto*: *scEmbeddedCopy K* = *topspace* (β *K*)
using *scCompactification-def*[*of K*]
by (*metis closure-of-closedin closure-of-subset-topspace topspace-subtopology-subset*)
then obtain *e′*
where *e′props*: *continuous-map* (β *K*) *K e′*
$\wedge$ ($\forall$ *x* $\in$ *topspace K* . *e′* (*scEmbed K x*) = *x*)
by (*metis continuous-embedding-inverse embeds scEmbeddedCopy-def subtopology-topspace*)

define *F* **where** *F* = *e′ o* (λ *p* $\in$ *topspace* (β *X*) . *restrict H* (*topspace* β *X*) *p*)

have *Fcts*: *F* $\in$ *cts*[β *X*, *K*]
proof −
have (λ *p* $\in$ *topspace* (β *X*) . *restrict H* (*topspace* β *X*) *p*) $\in$ *cts*[β *X*, *scProduct K*]
using *Hcts Xspace-def continuous-map-from-subtopology scCompactification-def*
by (*metis closedin-subset closedin-topspace mem-Collect-eq restrict-continuous-map*)
moreover have *H* ‘ (*topspace* β *X*) $\subseteq$ *topspace* (β *K*)
using *Xspace-def H-on-beta Xspace-def scCompactification-def*[*of K*] *onto* **by** *blast*
ultimately have (λ *p* $\in$ *topspace* (β *X*) . *restrict H* (*topspace* β *X*) *p*) $\in$ *cts*[β *X*, β *K*]
using *scCompactification-def*[*of K*]
by (*metis closed closure-of-closedin continuous-map-in-subtopology image-restrict-eq mem-Collect-eq onto*)
moreover have *e′* $\in$ *cts*[β *K*, *K*] **using** *e′props* **by** *simp*
ultimately show *?thesis*
using *F-def continuous-map-compose*[*of* β *X* β *K* (λ *p* $\in$ *topspace* (β *X*) . *restrict H* (*topspace* β *X*) *p*)]
by *auto*
qed

moreover have *Fextends*: $\forall\ x \in topspace\ X\ .\ (F\ o\ scEmbed\ X)\ x = f\ x$
proof $-$
{ **fix** x **assume** *xprops*: $x \in topspace\ X$
have $(F\ o\ scEmbed\ X)\ x = F\ (scEmbed\ X\ x)$ **by** *auto*
moreover have $scEmbed\ X\ x \in topspace\ (\beta\ X)$
using *assms(1) scEmbed-range[of X x] xprops* **by** *auto*
ultimately have $(F\ o\ scEmbed\ X)\ x$
$= (e'\ o\ (\lambda\ p \in topspace\ (\beta\ X)\ .\ restrict\ H\ (topspace\ \beta\ X)\ p))\ (scEmbed\ X\ x)$
using *F-def* **by** *simp*
also have $\ldots = (e'\ o\ (\lambda\ p \in topspace\ (\beta\ X)\ .\ H\ p))\ (scEmbed\ X\ x)$ **by** *auto*
finally have *step1*: $(F\ o\ scEmbed\ X)\ x = e'\ ((\lambda\ p \in topspace\ (\beta\ X)\ .\ H\ p)\ (scEmbed\ X\ x))$ **by** *auto*
have $(\lambda\ p \in topspace\ (\beta\ X)\ .\ H\ p)\ (scEmbed\ X\ x) = H\ (scEmbed\ X\ x)$
using *scEmbed-range[of X x] assms(1) xprops* **by** *auto*
also have $\ldots = scEmbed\ K\ (f\ x)$ **using** *H-of-scEmbed xprops* **by** *auto*
finally have *step2*: $(\lambda\ p \in topspace\ (\beta\ X)\ .\ H\ p)\ (scEmbed\ X\ x) = scEmbed\ K\ (f\ x)$
by *auto*
have $(F\ o\ scEmbed\ X)\ x = e'\ ((\lambda\ p \in topspace\ (\beta\ X)\ .\ H\ p)\ (scEmbed\ X\ x))$
using *step1* **by** *simp*
also have $\ldots = e'\ (scEmbed\ K\ (f\ x))$ **using** *step2* **by** *auto*
finally have $(F\ o\ scEmbed\ X)\ x = f\ x$
using *e'props tych-K scEmbed-range[of K f x] xprops fprops*
by (*metis continuous-map-image-subset-topspace image-subset-iff mem-Collect-eq*)
}
thus *?thesis* **by** *auto*
qed

ultimately have $F \in cts[\beta\ X,\ K] \wedge (\forall\ x \in topspace\ X\ .\ F\ (scEmbed\ X\ x) = f\ x)$
by *auto*
hence $\exists\ F \in cts[\beta\ X,\ K]\ .\ (\forall\ x \in topspace\ X\ .\ F\ (scEmbed\ X\ x) = f\ x)$ **by** *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *scExtension-unique*:
assumes $F \in cts[\beta\ X,\ K] \wedge (\forall\ x \in topspace\ X\ .\ F\ (scEmbed\ X\ x) = f\ x)$
and *compact-Hausdorff K*
shows $(\forall\ G\ .\ G \in cts[\beta\ X,\ K] \wedge (\forall\ x \in topspace\ X\ .\ G\ (scEmbed\ X\ x) = f\ x)$
$\longrightarrow (\forall\ p \in topspace\ (\beta\ X)\ .\ F\ p = G\ p))$
proof $-$
{ **fix** G **assume** *Gprops*: $G \in cts[\beta\ X,\ K] \wedge (\forall\ x \in topspace\ X\ .\ G\ (scEmbed\ X\ x) = f\ x)$

have ∀ $p \in$ *scEmbeddedCopy X* . $F\ p = G\ p$
proof −
{ **fix** p **assume** *pprops*: $p \in$ *scEmbeddedCopy X*
then obtain x **where** *xprops*: $x \in$ *topspace X* ∧ $p =$ *scEmbed X x*
using *scEmbeddedCopy-def*[*of X*] **by** *auto*
hence $F\ p = G\ p$ **using** *assms Gprops* **by** *auto*
}
thus *?thesis* **by** *auto*
qed
moreover have *dense-in* (β *X*) (*scEmbeddedCopy X*) (*topspace* (β *X*))
by (*metis closure-of-subset-topspace dense-in-closure dense-in-def scCompactification-def*
topspace-subtopology-subset)
moreover have (*cts-on* β *X to-shared Hausdorff-space*) {F,G}
proof −
have *Hausdorff-space K* **using** *assms*(*2*) **by** *auto*
moreover have ∀ $g \in \{F,G\}$. $g \in$ *cts*[β *X*, *K*]
using *assms Gprops* **by** *auto*
ultimately have ∃ *K* . *Hausdorff-space K* ∧ {F,G} ⊆ *cts*[β *X*,*K*] **by** *auto*
thus *?thesis* **by** *auto*
qed
ultimately have (∀ $p \in$ *topspace* (β *X*) . $F\ p = G\ p$)
using *continuous-maps-on-dense-subset*[*of F G* β *X scEmbeddedCopy X*]
by *auto*
}
thus *?thesis* **by** *auto*
qed

lemma *scExtension-property*:
assumes *tych-space X*
and *compact-Hausdorff K*
shows ∀ $f \in$ *cts*[*X*,*K*] . ∃! $F \in cts_E$[β *X*, *K*] . (∀ $x \in$ *topspace X* . F (*scEmbed X x*) = $f\ x$)
proof −
{ **fix** f **assume** *fprops*: $f \in$ *cts*[*X*,*K*]
define P **where** P = (λg . $g \in cts_E$[β *X*, *K*] ∧ (∀ $x \in$ *topspace X* . g (*scEmbed X x*) = $f\ x$))
then obtain F **where** *Fprops*: $F \in$ *cts*[β *X*, *K*] ∧ (∀ $x \in$ *topspace X* . F (*scEmbed X x*) = $f\ x$)
using *scExtension-exists*[*of X K*] *assms fprops* **by** *auto*
define F' **where** F' = *restrict F* (*topspace* β *X*)

have $F \in$ (*topspace* β *X*) → *topspace K* **using** *Fprops continuous-map-def*[*of* β *X K F*] **by** *auto*
hence *F′ext*: $F' \in$ (*topspace* β *X*) $\rightarrow_E$ *topspace K*
using *F′-def restrict-def*[*of F topspace* β *X*] *extensional-def*[*of topspace* β *X*]
by *auto*
moreover have *F′cts*: $F' \in$ *cts*[β *X*, *K*]
proof −

have $F' \in (topspace\ \beta\ X) \rightarrow topspace\ K$ **using** $F'ext$ **by** *auto*
moreover have $\forall\ U\ .\ \{x \in topspace\ \beta\ X.\ F\ x \in U\} = \{x \in topspace\ \beta\ X.\ F'\ x \in U\}$
using F'-*def* **by** *auto*
ultimately show *?thesis* **using** *Fprops* **unfolding** *continuous-map-def* **by** *auto*
qed
ultimately have $F' \in cts_E[\ \beta\ X,\ K\ ]$ **by** *auto*
moreover have $F'embed$: $(\forall\ x \in topspace\ X\ .\ F'\ (scEmbed\ X\ x) = f\ x)$
proof −
have $\forall\ x \in topspace\ X\ .\ scEmbed\ X\ x \in topspace\ \beta\ X$
using *assms(1) scEmbed-range[of X]* **by** *blast*
thus *?thesis* **using** F'-*def Fprops* **by** *fastforce*
qed
ultimately have $P\ F'$ **using** *P-def* **by** *auto*

moreover have $\forall\ G\ .\ P\ G \longrightarrow G = F'$
proof −
{ fix G **assume** *Gprops*: $P\ G$
{ fix p
have $F'\ p = G\ p$
proof (*cases* $p \in topspace\ \beta\ X$)
case *True*
hence $F' \in cts[\ \beta\ X,\ K] \wedge (\forall x{\in}topspace\ X.\ F'\ (scEmbed\ X\ x) = f\ x)$
using $F'cts\ F'embed$ **by** *auto*
moreover have $G \in cts[\ \beta\ X,\ K] \wedge (\forall x{\in}topspace\ X.\ G\ (scEmbed\ X\ x) = f\ x)$
using *Gprops P-def* **by** *auto*
ultimately show *?thesis*
using *assms(2) scExtension-unique*[*of* $F'\ X\ K\ f$] *True* **by** *blast*
next
case *False*
hence $F'\ p = undefined$ **using** F'-*def* **by** *auto*
moreover have $G\ p = undefined$
using *Gprops P-def extensional-def*[*of topspace* β *X*] *False* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
}
hence $\forall\ p\ .\ F'\ p = G\ p$ **by** *auto*
}
thus *?thesis* **by** *auto*
qed
ultimately have $\exists!\ F'\ .\ P\ F'$ **by** *blast*
hence $\exists!\ F \in cts_E[\beta\ X,\ K]\ .\ (\forall\ x \in topspace\ X\ .\ F\ (scEmbed\ X\ x) = f\ x)$
using *P-def* **by** *auto*
}
thus *?thesis* **by** *auto*
qed

end

References

[Wal74] Russell C. Walker. *The Stone-Čech Compactification*. Springer-Verlag, 1974.

[Wil70] Stephen Willard. *General Topology*. Addison-Wesley, 1970.