The Gelfand–Naimark–Segal Construction

Richard Schmoetten and Jacques D. Fleuriot

June 25, 2026

Abstract

This entry formalises complete normed algebras equipped with an involution, so-called C^*-algebras. We provide both a class definition, and a locale for C^*-algebras on carrier sets in the spirit of existing developments of linear algebra and smooth manifolds. Bounded operators on a complex Hilbert space, with the operator norm and adjoints, form such an algebra. The main theorem of this entry is a result in the converse direction: the Gelfand–Naimark–Segal (GNS) construction, which starts with a single suitable functional on a C^*-algebra in order to obtain both a Hilbert space and a representation of the algebra in terms of bounded operators on that space. This is implemented as a type construction in Isabelle/HOL, taking advantage of existing mechanisms for quotient types, and integrating with existing type classes for Hilbert spaces and Cauchy completions.

Contents

1 C^*-algebras

theory *Cstar-Algebra-On*
imports
Lie-Groups.Algebra-On
Types-To-Sets-Extension.SML-Topological-Space
Complex-Bounded-Operators.Complex-Vector-Spaces
HOL−Analysis.Abstract-Metric-Spaces
begin

bundle *scaleC-syntax* **begin**
notation *Complex-Vector-Spaces0.scaleC-class.scaleC* (**infixr** ‹$*_C$› *75*)
end

unbundle *no scaleC-syntax*

1.1 Additional lemmas for *Lie-Groups.Algebra-On*

lemma (**in** *algebra-on*)
shows *distR'*: $[\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow (x-y) \bullet z = x \bullet z - y \bullet z$
and *distL'*: $[\![x{\in}S;\ y{\in}S;\ z{\in}S]\!] \Longrightarrow z \bullet (x-y) = z \bullet x - z \bullet y$
using *distR m1.mem-uminus* **apply** *fastforce*
using *distL m1.mem-uminus* **by** *fastforce*

1.2 Involutive rings and algebras

locale *involutive-semigroup* = *semigroup-add-on-with* +

fixes *involution* :: *'a⇒'a* (‹-†› [*99*] *100*)
assumes *involution*[*simp*]: $x{\in}S \Longrightarrow (x^\dagger)^\dagger = x$
and *antiautomorphism*: $[\![x{\in}S;\ y{\in}S]\!] \Longrightarrow (pls\ x\ y)^\dagger = pls\ (y^\dagger)\ (x^\dagger)$
and *involution-mem*[*simp*]: $x{\in}S \Longrightarrow x^\dagger \in S$

— Bundle for class constant notation (e.g. for (+)): disable inside Ballarin's set-based locale for rings, re-enable once we involve type classes for e.g. base fields of algebras.
bundle *class-ring-notation* **begin**
notation *plus* (**infixl** + *65*)
notation *minus* (**infixl** ‹−› *65*)
notation *uminus* (‹(‹*open-block notation*=‹*prefix* −›› -)› [*81*] *80*)
end

unbundle *no class-ring-notation*
locale *involutive-ring* =
Ring-Theory.ring R addition multiplication zero unit +
involutive-semigroup R multiplication involution
for *R* :: *'a set* **and** *addition* :: *'a⇒'a⇒'a* (**infixl** + *65*)
and *multiplication* (**infixl** · *70*) **and** *zero* (**0**) **and** *unit* (**1**)
and *involution* :: *'a⇒'a* (‹-†› [*99*] *100*) +
assumes *ivl-ring*: $[\![x{\in}R;\ y{\in}R]\!] \Longrightarrow (x{+}y)^\dagger = x^\dagger + y^\dagger$
begin

lemma *unit-self-adjoint*: $\mathbf{1}^\dagger = \mathbf{1}$
using *antiautomorphism*[*of* - **1**] *involution involution-mem*
by (*metis multiplicative.right-unit multiplicative.unit-closed*)

lemma *ivl-uminus*: $x{\in}R \Longrightarrow$ *involution* $(-x) = -$ *involution x*
by (*metis* (*no-types*, *lifting*) *additive.inverse-equality additive.invertible additive.invertibleE*
additive.invertible-left-inverse2 additive.unit-closed involution-mem ivl-ring)

lemma *ivl-minus*: $[\![x{\in}R;\ y{\in}R]\!] \Longrightarrow$ *involution* $(x{-}y) =$ *involution x* − *involution y*
by (*simp add*: *ivl-ring ivl-uminus*)

end
unbundle *class-ring-notation*

A general definition of *-algebra might look as follows.

Definition 1 (*-algebra)**.** Let A be a ring with involution *, and R any commutative ring with involution $'$. A is a *-algebra if it is an associative algebra over R, such that:

$$\forall r \in R. \forall a \in A. (ra)^* = r'a^*$$

A *-*homomorphism* $f\colon A \to B$ is an algebra homomorphism that respects

involution:

$$\forall a \in A. f(a)^{*_B} = f(a^{*_A})$$

Since nearly all formalisation seems to use classes for the vector addition and base field, and some texts (e.g. [2]) define *-algebras using the complex numbers as a base ring directly, we do so as well, without worrying about the involution of the base ring being specified only implicitly, on a type-level. Even category theory is, most of the time, only interested in categories of spaces over a shared base field/ring. Note also our involutive complex algebras are unital (and associative).

locale *involutive-complex-algebra* =
assoc-algebra-1-on S scale multiplication unit +
involutive-ring S (+) *multiplication 0 unit involution*
for *S* :: *'a*::*ab-group-add set*
and *scale* :: *complex* ⇒ *'a* ⇒ *'a* (**infixr** ‹$*_C$› *75*)
and *multiplication* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹•› *74*)
and *unit* :: *'a* (‹**1**›)
and *involution* :: *'a*⇒*'a* (‹-†› [*99*] *100*) +
assumes *ivl-scale*: *s*∈*S* ⟹ (*c* $*_C$ *s*)† = *cnj c* $*_C$ *s*†

lemma *involutive-complex-algebraI*:
fixes *S* :: *'a*::*ab-group-add set*
and *scl* :: *complex* ⇒ *'a* ⇒ *'a*
and *mlt* :: *'a* ⇒ *'a* ⇒ *'a*
and *e* :: *'a*
and *involution* :: *'a*⇒*'a*
assumes *assoc-algebra-1-on S scl mlt e*
and *involution*: ∀ *x*. *x*∈*S* ⟶ *involution* (*involution x*) = *x*
and *antiautomorphism*: ∀ *x y*. *x*∈*S* ⟶ *y*∈*S* ⟶ *involution* (*mlt x y*) = *mlt* (*involution y*) (*involution x*)
and *ivl-mem*: ∀ *x*. *x*∈*S* ⟶ *involution x* ∈ *S*
and *ivl-ring*: ∀ *x y*. *x*∈*S*⟶ *y*∈*S* ⟶ *involution* (*x*+*y*) = (*involution x*) + (*involution y*)
and *ivl-scale*: ∀ *x c*. *x*∈*S* ⟶ *involution* (*scl c x*) = *scl* (*cnj c*) (*involution x*)
shows *involutive-complex-algebra S scl mlt e involution*
proof −
interpret *alg*: *assoc-algebra-1-on S scl mlt e* **using** *assms*(*1*)**.**
interpret *mon*: *Group-Theory.monoid S* (+) *0*
by (*unfold-locales*, *simp-all add*: *group-cancel.add1 alg.m1.mem-add alg.m1.mem-zero*)
have *mon.invertible u* **if** *u* ∈ *S* **for** *u*
by (*meson ab-left-minus alg.m1.mem-uminus mon.invertibleI neg-eq-iff-add-eq-0 that*)
then show *involutive-complex-algebra S scl mlt e involution*
by (*unfold-locales*, *simp-all add*: *alg.m1.subspace-UNIV alg.m1.subspace-add add.commute*
add.left-commute alg.amult-assoc alg.distL alg.distR assms(*2*−))
qed

lemma (**in** *module-on*) *sub-module*:
assumes *subspace A A⊆S*
shows *module-on A scale*
using *assms module-on.intro module-on.subspace-def module-on-axioms scale-left-distrib-on scale-one-on scale-right-distrib-on scale-scale-on subset-eq* **by** (*smt* (*verit*, *ccfv-threshold*))

locale *involutive-subalgebra* =
subalg: *involutive-complex-algebra A* + *involutive-complex-algebra B* **for** *A B* +
assumes $A \subseteq B$

lemma (**in** *involutive-complex-algebra*) *subalgebraI*:
fixes *A*
assumes *unital-subspace*: *m1.subspace A* $A \subseteq S$ $\mathbf{1} \in A$
and *mult-closed*: $\forall\, x\ y.\ x \in A \longrightarrow y \in A \longrightarrow x \bullet y \in A$
and *involution-closed*: $\forall\, x.\ x \in A \longrightarrow$ *involution x* $\in A$
shows *involutive-subalgebra* $(*_C)$ $(\bullet)$ $\mathbf{1}$ *involution A S*
proof −
interpret *vector-space-on A*
using *m1.sub-module unital-subspace*(*1*,*2*) *vector-space-on.intro* **by** *blast*

interpret *subalg*: *assoc-algebra-1-on A* $(*_C)$ $(\bullet)$ $\mathbf{1}$
apply *unfold-locales*
apply (*meson in-mono linearL'*(*2*) *unital-subspace*(*2*))
apply (*meson unital-subspace*(*2*) *in-mono linearL'*(*1*))
apply (*meson distributive*(*1*) *in-mono unital-subspace*(*2*))
apply (*meson unital-subspace*(*2*) *linearR'*(*1*) *subsetD*)
using *assms*(*4*) **apply** *blast*
using *unital-subspace*(*2*) **apply** *blast*
apply (*simp add*: *unital-subspace*(*3*))
using *unital-subspace*(*2*) **apply** *blast*
using *unital-subspace*(*2*) **apply** *blast*
by *simp*

interpret *ivl-subalg*: *involutive-complex-algebra A*
apply (*intro involutive-complex-algebraI*)
subgoal by *unfold-locales*
using *involution unital-subspace*(*2*) **apply** *blast*
apply (*meson antiautomorphism in-mono unital-subspace*(*2*))
using *involution-closed* **apply** *blast*
apply (*meson ivl-ring subset-eq unital-subspace*(*2*))
using *assms*(*2*) *ivl-scale* **by** *blast*

show *?thesis*
by (*unfold-locales*, *simp add*: *assms*(*2*))
qed

1.3 Preliminaries and experiments

context *Metric-space* **begin**

sublocale *topological-space-ow M mopen*
  apply *unfold-locales*
  subgoal by (*simp add*: *gt-ex mball-subset-mspace mopen-def*)
  subgoal by (*metis mopen-def openin-Int openin-mtopology*)
  subgoal using *openin-Union*[*of - mtopology*] **by** (*simp add*: *mtopology-def*)
  done

end

1.4 Subfields of a *field* on a type universe

Compare to *subspace* in *HOL−Types-To-Sets.Linear-Algebra-On*.

abbreviation *closed-under-un K op* ≡ ∀ *x*∈*K*. *op x* ∈ *K*
abbreviation *closed-under-bin K op* ≡ ∀ *x*∈*K*. ∀ *y*∈*K*. *op x y* ∈ *K*

locale *subfield* =
  fixes *K* :: *'a*::*field set*
  assumes *subfield-1*[*simp*]: *1* ∈ *K*
  and *subfield-closed*[*simp*]:
    x∈*K* ⟹ − *x* ∈ *K*
    x∈*K* ⟹ *inverse x* ∈ *K*
    ⟦*x*∈*K*; *y*∈*K*⟧ ⟹ *x* + *y* ∈ *K*
    ⟦*x*∈*K*; *y*∈*K*⟧ ⟹ *x* ∗ *y* ∈ *K*
begin

— The zero element of a field coincides with the zero element of any of its subfields. Since *0* is a class constant, it is always the zero element of the field on the type universe.
lemma *subfield-0* [*simp*]: *0* ∈ *K*
  by (*metis add.right-inverse subfield-1 subfield-closed*(*1*,*3*))

end

— Any field is a subfield of itself.
lemma *subfield-UNIV*: *subfield UNIV*
  unfolding *subfield-def* **by** *simp*

— The real numbers are a subfield of the complex numbers.
lemma *real-subfield-complex*: *subfield* {*c*. *Im c* = *0*}
  unfolding *subfield-def* **by** *auto*

1.4.1 ... as a transfer relation?

The price of being able to talk about a subfield of a different type is representational definiteness. The subfield of type 'a is only a subfield "up to isomorphism". Quite categorical in flavour, this should make no practical difference. See also skeletal categories: https://ncatlab.org/nlab/show/skeleton

context includes *lifting-syntax* **begin**

definition *subfield-rel* (R :: *'a::field* ⇒ *'b::field* ⇒ *bool*) ≡
  left-total R ∧ *bi-unique R* ∧ ~~*inj-on R (UNIV::'a set)*~~
  R 1 1 ∧
  (*R* ===> *R* ===> *R*) (+) (+) ∧
  (*R* ===> *R* ===> *R*) (∗) (∗) ∧
  (*R* ===> *R*) *uminus uminus* ∧
  (*R* ===> *R*) *inverse inverse*

lemma *subfield-rel-0*: *R 0 0* **if** *subfield-rel R*
  using *add.right-inverse* **by** (*metis rel-funD that*[*unfolded subfield-rel-def*])

lemma *subfield-rel-UNIV*: *subfield-rel* (=)
  by (*simp add*: *bi-unique-eq left-total-eq rel-fun-eq subfield-rel-def*)

lemma *real-subfield-rel-complex*: *subfield-rel* (λ*r c*. *complex-of-real r* = *c*)
  using *of-real-add of-real-mult* **by** (*simp add*: *left-total-def bi-unique-def subfield-rel-def rel-fun-def*)

end

1.5 Topological Fields and Vector Spaces

The topology will be defined, as usual for locales, on a carrier set only. Here, that carrier is implicit as the union of all open sets, like for manifolds. This means e.g. addition is defined on the type universe, but is continuous only on the subset of that universe covered by the topology.

abbreviation *continuous-binary-op t f* ≡ *continuous-map* (*prod-topology t t*) *t* (λ(*a*,*b*). *f a b*)
locale *topological-field-ow* = *topological-space-ow F* τ
  for *F* :: *'at::field set*
  and τ :: *'at set* ⇒ *bool* +
  assumes *continuous-plus*: *continuous-binary-op* (*topology* τ) (+)
  and *continuous-times*: *continuous-binary-op* (*topology* τ) (∗)
  and *continuous-uminus*: *continuous-map* (*topology* τ) (*topology* τ) *uminus*
  and *continuous-inverse*: *continuous-map* (*topology* τ) (*topology* τ) *inverse*
begin

A somewhat Frankensteinian construction. The locale *topological-space-ow* is taken from the ETTS-powered *Types-To-Sets-Extension.SML-Topological-Space*,

and transferred from *topological-space*. By contrast, *continuous-map* stems from *HOL−Analysis.Abstract-Topology*. The field operations are still class constants, similarly to *vector-space-on*, with topological properties on the carrier F only.

end

locale *topological-vector-space-on* =
  vector-space-on V scale +
  topological-space-ow V τ_V +
  top-F: *topological-field-ow F* τ_F
 for F :: *'a*::*field set* **and** τ_F :: *'a set* ⇒ *bool*
  and V :: *'b*::*ab-group-add set*
  and *scale* :: *'a* ⇒ *'b* ⇒ *'b* (**infixr** ∗*s* *75*)
  and τ_V :: *'b set* ⇒ *bool* +
 assumes *continuous-add*: *continuous-binary-op* (*topology* τ_V) (+)
  and *continuous-scale*: *continuous-map* (*prod-topology* (*topology* τ_F) (*topology* τ_V)) (*topology* τ_V) (λ(*a*,*v*). *scale a v*)

1.6 Normed Fields and Vector Spaces

locale *normed-field* = *subfield F* **for** F :: *'a*::*field set* +
 fixes *norm* :: *'a*⇒*real* (||-|| *100*)
 assumes *normed-field*: $x{\in}F \Longrightarrow ||x|| \geq 0$
  $x{\in}F \Longrightarrow ||x|| = 0 \longleftrightarrow x = 0$
  $x{\in}F \Longrightarrow ||{-}x|| = ||x||$
  $[\![x{\in}F;\ y{\in}F]\!] \Longrightarrow ||x{+}y|| \leq ||x||{+}||y||$
  $[\![x{\in}F;\ y{\in}F]\!] \Longrightarrow ||x{*}y|| \leq ||x||{*}||y||$

locale *valued-field* = *normed-field* +
 assumes *multiplicative-norm*: $[\![x{\in}F;\ y{\in}F]\!] \Longrightarrow ||x{*}y|| = ||x||{*}||y||$
begin

lemma *norm-1* [*simp*]: ||*1*|| = *1*
 by (*metis mult-cancel-right1 multiplicative-norm normed-field*(*2*) *subfield-1 zero-neq-one*)

end

locale *seminormed-vector-space-on* =
  vector-space-on V scale +
  F: *valued-field F field-abs*
 for F :: *'f*::*field set* **and** *field-abs* :: *'f*⇒*real* (||-|| [*80*])
  and V :: *'v*::*ab-group-add set* **and** *scale* :: *'f*⇒*'v*⇒*'v* +
 fixes *vector-norm* :: *'v*::*ab-group-add* ⇒ *real* (||-|| [*65*]) — At priority 65, you can write sums inside the norm.
 assumes *triangle-add*: $[\![x{\in}V;\ y{\in}V]\!] \Longrightarrow$
    vector-norm (*x*+*y*) ≤ *vector-norm x* + *vector-norm y*
  and *absolute-homogeneous*: $[\![a{\in}F;\ x{\in}V]\!] \Longrightarrow$
    vector-norm (*scale a x*) = (*field-abs a*) ∗ (*vector-norm x*)

begin

lemma *norm-0*: *vector-norm 0 = 0*
  using *absolute-homogeneous*[*of 0 0*] *F.normed-field*(*2*) *scale-zero-left mem-zero F.subfield-0*
  by *fastforce*

lemma *norm-uminus*: $||-x|| = ||x||$ **if** *x*: $x \in V$
proof −
  have $||-x|| = \|-1\| * ||x||$
    using *absolute-homogeneous*[*OF - x, of −1*] **by** (*simp add*: *scale-minus-left-on x*)
  then show $||-x|| = ||x||$
    using *F.normed-field*(*3*)[*OF F.subfield-1*] *absolute-homogeneous x* **by** *simp*
qed

lemma *norm-nonneg* [*simp*]: $x \in V \Longrightarrow ||x|| \geq 0$
  using *mem-uminus norm-0 norm-uminus triangle-add*
  by *fastforce*

end

locale *normed-vector-space-on* = *seminormed-vector-space-on* +
  assumes *positive-definite*: $[\![x \in V;\ \textit{vector-norm}\ x = 0]\!] \Longrightarrow x = 0$
begin

lemma *norm-0-iff*: *vector-norm* $x = 0 \longleftrightarrow x = 0$ **if** $x \in V$
  using *positive-definite norm-0 that* **by** *blast*

abbreviation (*input*) *induced-metric*
  where *induced-metric x y* $\equiv ||y-x||$

The locale *Metric-space*, which we would like to instantiate, requires things like $0 \leq ||y - x||$. This is not true in general: $?x \in V \Longrightarrow 0 \leq ||?x||$ only guarantees this behaviour inside the carrier set. Thus we have to "totalise" the induced metric for use in the existing *Metric-space*, by defining behaviour outside the carrier set to satisfy the axioms of the locale.

definition *totalized-induced-metric* (d)

  where d *x y* $\equiv$ *if* $x \in V \wedge y \in V$ *then induced-metric x y else 0*

lemma *induced-metric-simps* [*simp*]:
  $[\![x \in V;\ y \in V]\!] \Longrightarrow$ d *x y* = *induced-metric x y*
  $x \notin V \Longrightarrow$ d *x y* = *0*
  $y \notin V \Longrightarrow$ d *x y* = *0*
  by (*simp-all add*: *totalized-induced-metric-def*)

lemma *metric-translation-invariant*: $[\![x \in V;\ y \in V;\ z \in V]\!] \Longrightarrow$ d *x y* = d (*x*+*z*)

$(y+z)$
by (*simp add*: *mem-add*)
lemma *metric-translation-invariant'*: $[\![x \in V;\ y \in V]\!] \Longrightarrow \mathrm{d}\ x\ y = \mathrm{d}\ (x-y)\ 0$
using *mem-add mem-uminus mem-zero* **by** *fastforce*

sublocale *Metric-space* V d
proof
fix $x\ y\ z$
show $0 \leq \mathrm{d}\ x\ y$
using *mem-add mem-uminus norm-nonneg*
by (*metis ab-group-add-class.ab-diff-conv-add-uminus induced-metric-simps le-numeral-extra*(*3*))
show $\mathrm{d}\ x\ y = \mathrm{d}\ y\ x$
by (*metis mem-add mem-uminus minus-diff-eq norm-uminus totalized-induced-metric-def uminus-add-conv-diff*)
assume *x*: $x \in V$ **and** *y*: $y \in V$
thus $\mathrm{d}\ x\ y = 0 \longleftrightarrow x = y$
by *simp* (*metis diff-conv-add-uminus eq-iff-diff-eq-0 mem-add mem-uminus norm-0-iff*)
assume *z*: $z \in V$
have $||z - x|| = ||(z-y) + (y-x)||$ **by** *simp*
also have $\ldots \leq ||z - y|| + ||y - x||$
using *triangle-add*[*of* $z-y$ $y-x$] **by** (*metis diff-conv-add-uminus mem-add mem-uminus x y z*)
finally show $\mathrm{d}\ x\ z \leq \mathrm{d}\ x\ y + \mathrm{d}\ y\ z$
using $x\ y\ z$ **by** *simp*
qed

end

lemma *totalized-induced-metric-eq-dist*:
normed-vector-space-on.totalized-induced-metric UNIV norm = dist
proof −
interpret *normed-vector-space-on UNIV abs UNIV* :: *'a set scaleR norm*
apply *unfold-locales* **apply** *simp-all*
using *scaleR-right.add* **apply** *blast*
using *module.scale-left-distrib module-axioms* **apply** *blast*
apply (*simp add*: *abs-mult*)
using *abs-mult* **apply** *blast*
using *norm-triangle-ineq* **by** *blast*
show *totalized-induced-metric = dist*
unfolding *totalized-induced-metric-def*
by (*auto simp add*: *Met-TC.commute simp flip*: *dist-norm*)
qed

term ‹*x*::*'a*::*real-normed-vector*›

locale *banach-space-on = normed-vector-space-on +*
assumes *mcomplete*

thm *class.cbanach-def*
lemma *cbanach-is-banach-space-on*:
shows *banach-space-on UNIV cmod (UNIV::'cb::cbanach set) scaleC norm*
apply *unfold-locales* **apply** *simp-all*
subgoal using *complex-vector.vector-space-assms(1)* **by** *blast*
subgoal using *complex-vector.vector-space-assms(2)* **by** *blast*
subgoal using *norm-triangle-ineq* **by** *blast*
subgoal using *norm-mult-ineq* **by** *blast*
subgoal using *norm-mult* **by** *blast*
subgoal by (*simp add*: *norm-triangle-ineq*)
subgoal
unfolding *totalized-induced-metric-eq-dist mcomplete-iff-complete*
by (*simp add*: *complete-UNIV*)
done

1.7 C* algebras

locale *Cstar-algebra =*
banach-space-on F cmod V scale vector-norm +

involutive-complex-algebra V scale multiplication unit involution
for *F* :: *complex set*
and *V* :: *'v::ab-group-add set*
and *scale* :: *complex ⇒ 'v ⇒ 'v* (**infixr** ‹$*_C$› *75*)
and *multiplication* :: *'v ⇒ 'v ⇒ 'v* (**infixr** ‹•› *74*)
and *vector-norm* :: *'v ⇒ real* (‹||-||› [*65*])
and *unit* :: *'v* (‹**1**›)
and *involution* :: *'v⇒'v* (‹-†› [*99*] *100*) +
assumes *normed-algebra* : $\bigwedge A\ B.\ [\![A{\in}V;\ B{\in}V]\!] \Longrightarrow ||A•B|| \leq ||A|| * ||B||$
assumes *normed-unital*: $||\mathbf{1}|| = 1$
assumes *involution-isometric*: $\bigwedge A.\ A{\in}V \Longrightarrow ||A^\dagger|| = ||A||$
assumes *Cstar*: $\bigwedge A.\ A{\in}V \Longrightarrow ||(A^\dagger) • A|| = ||A^\dagger|| * ||A||$
begin

lemma *Bstar*: $||(A^\dagger) • A|| = ||A||^2$ **if** $A{\in}V$ **for** *A*
using *Cstar*[*OF that*]
unfolding *involution-isometric*[*OF that*] *power2-eq-square* **.**

end

lemma *Bstar-implies-isometric*:
assumes *seminormed-vector-space-on F cmod V scale vector-norm*
involutive-complex-algebra V scale multiplication unit involution
and $\bigwedge A\ B.\ [\![A{\in}V;\ B{\in}V]\!] \Longrightarrow$ *vector-norm* (*multiplication A B*) $\leq$ *vector-norm A * vector-norm B vector-norm unit = 1*

and $\bigwedge A.\ A \in V \Longrightarrow$ *vector-norm* (*multiplication* (*involution A*) *A*) = (*vector-norm A*)^2
and $A \in V$
shows *vector-norm* (*involution A*) = *vector-norm A*
proof −
interpret *seminormed-vector-space-on F cmod V scale vector-norm*
using *assms*(*1*) **.**
interpret *involutive-complex-algebra V scale multiplication unit involution*
using *assms*(*2*) **.**
from *assms*(*3*)[*of involution A A*]
have (*vector-norm A*)^2 ≤ *vector-norm* (*involution A*) ∗ *vector-norm A*
by (*simp add*: *assms*(*5*,*6*))
hence *ineq-1*: (*vector-norm A*) ≤ *vector-norm* (*involution A*)
unfolding *power2-eq-square*
using *norm-nonneg*[*OF involution-mem*[*OF assms*(*6*)]]
using *nle-le*
by (*smt* (*verit*, *ccfv-SIG*) *mult-mono vector-space-over-itself.scale-cancel-right*)
from *assms*(*3*)[*of A involution A*]
have (*vector-norm* (*involution A*))^2 ≤ *vector-norm A* ∗ *vector-norm* (*involution A*)
by (*metis assms*(*5*,*6*) *involution involution-mem*)
hence *ineq-2*: (*vector-norm* (*involution A*)) ≤ *vector-norm A*
unfolding *power2-eq-square*
using *norm-nonneg*[*OF assms*(*6*)]
using *linorder-not-le* **by** *fastforce*
from *ineq-1 ineq-2* **show** *?thesis* **by** *force*
qed

locale *Cstar-subalgebra* =
subalg: *Cstar-algebra* - *A* + *Cstar-algebra* - *B* **for** *A B* +
assumes *A* ⊆ *B*

lemma (**in** *banach-space-on*) *closed-iff-complete*:
assumes *A* ⊆ *V*
shows *closedin mtopology A* ⟷ *Metric-space.mcomplete A* d
proof −
interpret *sub-A*: *Submetric V* d *A*
using *assms* **by** *unfold-locales*
show *?thesis*
using *sub-A.closedin-eq-mcomplete assms banach-space-on-axioms*
by (*simp add*: *banach-space-on-def banach-space-on-axioms-def*)
qed

lemma (**in** *normed-vector-space-on*) *from-subspace*:
assumes *subspace A A*⊆*V*

shows *normed-vector-space-on F field-abs A scale vector-norm*
proof −
interpret *VS*: *vector-space-on A scale*
by (*simp add*: *assms sub-module vector-space-on-alt-def*)
show *?thesis*
apply *unfold-locales*
using *absolute-homogeneous assms*(*2*) *positive-definite triangle-add* **by** (*auto simp*: *in-mono*)
qed

Not particularly elegantly or generally written: it doesn't matter which (induced) metric we use, because all cases outside of the subspace carrier A don't matter. Used for C*-subalgebra intro lemma.

lemma (**in** *normed-vector-space-on*) *mcomplete-submetric-equal*:
assumes $A \subseteq V$ *subspace A*
shows *Metric-space.mcomplete A* (*normed-vector-space-on.totalized-induced-metric A vector-norm*) = *Metric-space.mcomplete A* d
proof −
interpret *VS-A*: *normed-vector-space-on - - A*
using *from-subspace*[*OF assms*(*2*,*1*)] .
interpret *met-A*: *Metric-space A* d
apply *unfold-locales*
using *nonneg commute assms*(*1*) *zero triangle subset-iff* **by** (*auto simp*: *in-mono*)
have *tot-met-simp*: *VS-A.totalized-induced-metric x y* = *totalized-induced-metric x y* **if** *x*∈*A y*∈*A* **for** *x y*
using *assms*(*1*) *that totalized-induced-metric-def* **by** *auto*
have *mball-local-simp*: *VS-A.mball x r* = *met-A.mball x r* **for** *x r*
using *tot-met-simp* **by** *auto*
show *?thesis*
unfolding *VS-A.mcomplete-def met-A.mcomplete-def*
unfolding *VS-A.MCauchy-def met-A.MCauchy-def*
unfolding *VS-A.mtopology-def met-A.mtopology-def*
unfolding *VS-A.mopen-def met-A.mopen-def*
apply (*subst mball-local-simp*)
using *tot-met-simp* **by** (*auto simp*: *range-subsetD*)
qed

lemma (**in** *banach-space-on*) *closed-iff-complete′*:
defines *complete-in-induced-submetric* ≡
$\lambda A\ n$. *Metric-space.mcomplete A* (*normed-vector-space-on.totalized-induced-metric A n*)
assumes $A \subseteq V$ *subspace A*
shows *closedin mtopology A* ⟷ *complete-in-induced-submetric A vector-norm*
using *closed-iff-complete mcomplete-submetric-equal assms* **by** *simp*

lemma (**in** *Cstar-algebra*) *subalgebraI*:
fixes *A*
assumes *unital-subalgebra*: *subspace A* $A \subseteq V$ $\mathbf{1} \in A$
and *mult-closed*: $\forall x\ y.\ x \in A \longrightarrow y \in A \longrightarrow x \bullet y \in A$

and *involution-closed*: $\forall\, x.\ x \in A \longrightarrow \mathit{involution}\ x \in A$
and *closed-subset*: *closedin mtopology A*
shows *Cstar-subalgebra F scale* $(\bullet)$ *vector-norm* $\mathbf{1}$ *involution A V*
proof −
interpret *involutive-subalgebra - - - - A V*
using *subalgebraI*[*OF assms*(*1*−*5*)] **.**
show *?thesis*
apply *unfold-locales*
using *triangle-add unital-subalgebra*(*2*) **apply** *blast*
using *absolute-homogeneous unital-subalgebra*(*2*) **apply** *blast*
using *assms*(*2*) *positive-definite* **apply** *blast*
using *closed-iff-complete'*[*OF assms*(*2*,*1*)] *closed-subset* **apply** *blast*
apply (*meson assms*(*2*) *normed-algebra subset-iff*)
using *normed-unital assms*(*2*) *involution-isometric Cstar* **by** *blast*+
qed

lemma (**in** *Cstar-algebra*) *subalgebraI'*:
fixes *A*
assumes *unital-subalgebra*: *subspace A* $A \subseteq V$ $\mathbf{1} \in A$
and *mult-closed*: $\forall\, x\ y.\ x \in A \longrightarrow y \in A \longrightarrow x \bullet y \in A$
and *involution-closed*: $\forall\, x.\ x \in A \longrightarrow \mathit{involution}\ x \in A$
and *complete-subset*: *Metric-space.mcomplete A* d
shows *Cstar-subalgebra F scale* $(\bullet)$ *vector-norm* $\mathbf{1}$ *involution A V*
using *subalgebraI*[*OF assms*(*1*−*5*)] *assms*(*2*,*6*) *closed-iff-complete* **by** *blast*

1.8 Cauchy-Schwarz Inequality for functionals on involutive algebras

context *involutive-complex-algebra* **begin**

Disable the notation from `Jacobson_Basic_Algebra` that conflicts with underlying class constants used for our algebras.

no-notation *additive.inverse* (− - [*66*] *65*) — conflicts with *uminus*
no-notation *subtraction* (**infixl** − *65*) — conflicts with (−)

lemma *additive-inverse-uminus*[*simp*]:
additive.inverse A $= -$ *A* **if** $A \in S$
by (*simp add*: *additive.inverse-equality m1.mem-uminus that*)

lemma *subtraction-minus*[*simp*]:
subtraction A B $= A - B$ **if** $A \in S$ $B \in S$
by (*simp add*: *that*(*2*))

declare *ivl-uminus* [[*simp del*]]
lemma *involution-uminus'*[*simp*]: *involution* $(-B) = -$ (*involution B*) **if** $B \in S$ **for** *B*
using *ivl-uminus* **by** (*simp add*: *that*)

declare *ivl-minus* [[*simp del*]]
lemma *ivl-minus'*[*simp*]: *involution* $(A - B)$ = *involution* A − (*involution* B)
if $A{\in}S$ $B{\in}S$ **for** B
using *ivl-minus*[*OF that*] **by** (*simp add*: *that*)

end

The nomenclature of "functional" is taken from operator algebras, where operators are often functions. Here "positive" is not strict: the corresponding axiom is named after non-negativity.

locale *positive-normalised-linear-functional* =
involutive-complex-algebra S scale multiplication unit involution +
linear-on S UNIV scale (∗) ω
for S :: $'a$::*ab-group-add set*
and *scale* :: *complex* ⇒ $'a$ ⇒ $'a$ (**infixr** ‹∗$_C$› *75*)
and *multiplication* :: $'a$ ⇒ $'a$ ⇒ $'a$ (**infixr** ‹•› *74*)
and *unit* :: $'a$ (‹**1**›)
and *involution* :: $'a{\Rightarrow}'a$ (‹-†› *73*)
and ω :: $'a{\Rightarrow}$*complex* +
assumes *nonneg*: $A{\in}S \Longrightarrow \omega\ ((A^\dagger) \bullet A) \geq 0$
and *one* [*simp*]: $\omega\ \mathbf{1} = 1$
begin

notation *cmod* (‖-‖) — *abs* has omitted priority

lemma *nonneg'*:
assumes $A \in S$
shows *Re* $(\omega\ (\mathit{involution}\ A \bullet A)) \geq 0$ *Re* $(\omega\ (A \bullet \mathit{involution}\ A)) \geq 0$ *Im* $(\omega\ (A \bullet \mathit{involution}\ A)) = 0$ *Im* $(\omega\ (\mathit{involution}\ A \bullet A)) = 0$
using *nonneg assms involution involution-mem*
by (*metis zero-complex.simps*(*1*) *Re-mono*,
metis zero-complex.simps(*1*) *Re-mono*,
(*metis comp-Im-same zero-complex.simps*(*2*))+)

lemma *involution-conjugate*: $\omega\ (\mathit{involution}\ A) = \mathit{cnj}\ (\omega\ A)$ **if** $A \in S$
proof −
have *is-Real*: *Im* $(\alpha * \omega\ A + \mathit{cnj}\ \alpha * \omega\ (\mathit{involution}\ A)) = 0$ **if** $A \in S$ **for** α::*complex* **and** A
proof −
let *?s* = $\mathbf{1} + \alpha *_C A$ — This is the crucial starting "guess".
note [*simp*] = *m1.mem-scale*
have *inS* [*simp*]: $A \in S$ *involution* $A \in S$ *?s* $\in S$
by (*simp-all add*: *that*)
have *?s* • *involution ?s* = $\mathbf{1}$ • *involution* $(\mathbf{1} + \alpha *_C A) + \alpha *_C A$ • *involution* $(\mathbf{1} + \alpha *_C A)$
by (*simp add*: *distR*[*of* **1** $\alpha *_C A$ *involution ?s*])
also have ... = $\mathbf{1}$ + *involution* $(\alpha *_C A) + \alpha *_C A + \alpha *_C A$ • *involution* $(\alpha *_C A)$

by (*simp add*: *ivl-ring unit-self-adjoint algebra-simps distL*)
finally have *?s* • *involution ?s* = **1** + $|\alpha|^2$ $*_C$ *A*•(*involution A*) + α $*_C$ *A* + *involution* (α $*_C$ *A*)
using *ivl-scale* **using** *complex-norm-square*[*of* α] **by** (*simp add*: *abs-complex-def mult.commute*)
then have *Im* (α ∗ ω *A* + *cnj* α ∗ ω (*involution A*)) = *Im* (ω (*?s* • *involution ?s*))
by (*simp add*: *abs-complex-def add scale ivl-scale nonneg'*(*3*))
then show *?thesis*
by (*simp add*: *nonneg'*(*3*))
qed
have *re-im-decompose*: *0* = *Re a* ∗ (*Im* (ω *A*) + *Im* (ω (*involution A*))) + *Im a* ∗ (*Re* (ω *A*) − *Re* (ω (*involution A*)))
for *a*::*complex*
using *is-Real*[*OF that*, *of a*]
by (*simp add*: *vector-space-over-itself.scale-right-diff-distrib vector-space-over-itself.scale-right-distrib*)
have *Im* (ω *A*) + *Im* (ω (*involution A*)) = *0 Re* (ω *A*) − *Re* (ω (*involution A*)) = *0*
using *re-im-decompose*[*of 1*] *re-im-decompose*[*of* i] **by** *simp-all*
thus *?thesis* **by** (*simp add*: *complex-eq-iff*)
qed

lemma *cnj-ivl*: *cnj* (ω (*involution A*)) = ω (*A*) **if** *A* ∈ *S*
using *involution-conjugate that* **by** *simp*

Any such functional can be used to define a sesquilinear form that acts as a generalised inner product. This interpretation is justified below by providing a Cauchy-Schwarz inequality.

definition *functional-inner* :: *'a* ⇒ *'a* ⇒ *complex* (⟨-|-⟩ *76*)
where *functional-inner A B* ≡ ω (($A^\dagger$) • *B*)

named-theorems *finner-simps*

lemma *finner-nonneg* [*finner-simps*]:
assumes *A* ∈ *S*
shows ⟨*A*|*A*⟩ ≥ *0*
using *nonneg* **by** (*simp add*: *assms functional-inner-def*)

lemma *finner-scale* [*finner-simps*]:
⟨*c* $*_C$ *A*|*B*⟩ = (*cnj c*) ∗ ⟨*A*|*B*⟩
⟨*A*|*c* $*_C$ *B*⟩ = *c* ∗ ⟨*A*|*B*⟩
if *A*∈*S B*∈*S* **for** *c* **and** *A B*
unfolding *functional-inner-def*
subgoal using *involution-mem that scale ivl-scale linearL'*(*1*) *multiplicative.composition-closed* **by** *presburger*
subgoal using *involution-mem that multiplicative.composition-closed scalar-compat'*(*2*) *scale* **by** *presburger*
done

lemma *finner-plus* [*finner-simps*]: $\langle A+B|C\rangle = \langle A|C\rangle + \langle B|C\rangle$ $\langle A|B+C\rangle = \langle A|B\rangle + \langle A|C\rangle$
if $A\in S$ $B\in S$ $C\in S$ **for** A B C
unfolding *functional-inner-def*
subgoal using *involution-mem that add amult-closed distR ivl-ring* **by** *presburger*
subgoal using *involution-mem that add amult-closed distL* **by** *presburger*
done

lemma *finner-uminus* [*finner-simps*]: $\langle uminus\ A|B\rangle = uminus\ (\langle A|B\rangle)$ $\langle A|uminus\ B\rangle = uminus\ (\langle A|B\rangle)$
if $A\in S$ $B\in S$ **for** c **and** A B
using *ivl-uminus mapsto-uminus* **by** (*simp-all add*: *that functional-inner-def*)

lemma *finner-minus* [*finner-simps*]: $\langle A-B|C\rangle = \langle A|C\rangle - \langle B|C\rangle$ $\langle A|B-C\rangle = \langle A|B\rangle - \langle A|C\rangle$
if $A\in S$ $B\in S$ $C\in S$ **for** c **and** A B C
subgoal using *finner-plus*[*OF - m1.mem-uminus*] *finner-uminus*(*1*) *that* **by** *force*
subgoal using *finner-plus*[*OF - - m1.mem-uminus*] *finner-uminus*(*2*) *that* **by** *force*
done

lemma *finner-cnj*:$\langle A|B\rangle = cnj\ (\langle B|A\rangle)$
if $A\in S$ $B\in S$ **for** A B
unfolding *functional-inner-def*
using *involution-conjugate*[*of involution A • B*] *antiautomorphism*[*of involution A B*] *involution*[*OF that*(*1*)]
by (*simp add*: *that*)

lemma *finner-real-commute*:
assumes $A\in S$ $B\in S$ $\langle A|B\rangle \in \mathbb{R}$
shows $\langle A|B\rangle = \langle B|A\rangle$
using *cnj-ivl*[*of involution B • A*] *antiautomorphism*[*of involution B A*] *involution*[*of B*] *assms*(*3*)[*unfolded Reals-cnj-iff*]
by (*simp add*: *assms*(*1,2*) *functional-inner-def*)

lemma *finner-zero-commute*:
assumes $A\in S$ $B\in S$ $\langle A|B\rangle = 0$
shows $\langle B|A\rangle = 0$
using *finner-real-commute assms Reals-0* **by** *metis*

lemma *finner-self-0-if*: $\langle A|A\rangle = 0$ **if** $A = 0$
by (*simp add*: *functional-inner-def mapsto-zero that*)

lemma *pythagorean-theorem*:
assumes $A\in S$ $B\in S$ $\langle A|B\rangle = 0$
shows $\langle A+B|A+B\rangle = \langle A|A\rangle + \langle B|B\rangle$
proof (*unfold functional-inner-def*)
have *inS*: *involution A ∈ S involution B ∈ S involution A • B ∈ S involution B*

• $A \in S$
by (*simp-all add*: *assms*)
have *BA-0*: ω (*involution B* • *A*) = *0*
using *involution antiautomorphism complex-cnj-zero cnj-ivl*[*symmetric*]
by (*simp only*: *assms*(*1,2,3*)[*unfolded functional-inner-def*] *inS*)
have ω ((*involution* (*A* + *B*)) • (*A* + *B*)) = ω ((*involution A* • *A*) + *involution A* • *B* + *involution B* • *A* + *involution B* • *B*)
by (*simp add*: *add.commute add.left-commute assms*(*1,2*) *distributive inS*(*1,2*) *ivl-ring*)
thus ω (*involution* (*A* + *B*) • (*A* + *B*)) = ω (*involution A* • *A*) + ω (*involution B* • *B*)
using *add BA-0 finner-zero-commute*[*OF assms*]
using *assms*(*1,2,3*) *functional-inner-def* **by** *fastforce*
qed

lemma *finner-csqrt-nonneg*: *csqrt* (⟨*X*|*X*⟩) ≥ *0* **if** *X*∈*S*
unfolding *less-eq-complex-def*
using *finner-nonneg Re-csqrt* **apply** *simp*
using *cmod-Re that* **by** *blast+*

lemma *finner-cmod-eq*: ‖⟨*B*|*B*⟩‖ = ⟨*B*|*B*⟩ **if** *B*∈*S*
using *cmod-Re finner-nonneg nonnegative-complex-is-real of-real-Re that* **by** *presburger*

definition *functional-norm* :: *'a* ⇒ *real* (||-||$_\omega$)
where *functional-norm X* ≡ *Re* (*csqrt* (⟨*X*|*X*⟩))

lemma *fnorm-finner-squared*: ⟨*B*|*B*⟩ = ||*B*||$_\omega{}^2$ ‖⟨*B*|*B*⟩‖ = ||*B*||$_\omega{}^2$ **if** *B*∈*S*
unfolding *functional-norm-def*
apply (*metis Re-power-real complex-is-Real-iff complex-is-real-iff-compare0 finner-csqrt-nonneg functional-inner-def nonneg'*(*4*) *of-real-Re power2-csqrt that*)
by (*metis cmod-Re finner-csqrt-nonneg norm-power power2-csqrt that*)

lemma *fnorm-nonneg*: ||*X*||$_\omega$ ≥ *0*
using *Re-csqrt functional-norm-def* **by** *presburger*

lemma *finner-lindep*:
fixes *c*::*complex*
assumes *B*∈*S*
shows ‖⟨*c* *$_C$ *B*|*B*⟩‖ = ‖*c*‖ * ||*B*||$_\omega{}^2$
using *finner-scale*(*1*)[*OF assms assms*] *fnorm-finner-squared*(*2*)[*OF assms*]
by (*simp add*: *norm-mult*)

lemma *fnorm-scale*: ||*c* *$_C$ *X*||$_\omega$ = ‖*c*‖ * ||*X*||$_\omega$ **if** *X*∈*S* **for** *c*::*complex*
proof (*unfold functional-norm-def*)
have *2*: ⟨*c* *$_C$ *X*|*c* *$_C$ *X*⟩ = *c* * *cnj c* * ⟨*X*|*X*⟩
using *finner-scale*(*1*) *finner-scale*(*2*)[*OF m1.mem-scale*] **by** (*simp add*: *that*)
have ‖⟨*c* *$_C$ *X*|*c* *$_C$ *X*⟩‖ = *Re* (⟨*c* *$_C$ *X*|*c* *$_C$ *X*⟩) ‖⟨*X*|*X*⟩‖ = *Re* (⟨*X*|*X*⟩)
using *cmod-Re finner-nonneg*(*1*)

by (*simp add*: *functional-inner-def less-eq-complexI m1.mem-scale nonneg'*(*3*) *that*
del: *involution-mem*)+
thus *Re* (*csqrt* (⟨*c* *$_C$ *X*|*c* *$_C$ *X*⟩)) = ‖*c*‖ * *Re* (*csqrt* (⟨*X*|*X*⟩))
apply *simp*
using *2 cmod-Re complex-mod-sqrt-Re-mult-cnj real-sqrt-mult*
by (*metis* (*no-types*, *lifting*) *cnj-x-x-geq0 mult.commute norm-mult*)
qed

lemma *finner-lindep-cong*:
assumes *B*∈*S A* = *c* *$_C$ *B*
shows ‖⟨*A*|*B*⟩‖ = ||*A*||$_\omega$ * ||*B*||$_\omega$
using *finner-lindep fnorm-scale*
by (*simp add*: *assms power2-eq-square*)

context — Controlling notation for partitions/division. **begin**
no-notation *equivalence.Partition* (**infixl** ‹'/› *75*)

— Stolen from *Polygonal-Number-Theorem.Polygonal-Number-Theorem-Lemmas.qua-disc* on the AFP. The import brought in too many dependencies to be worth it.
lemma *qua-disc*:
fixes *a b c* :: *real*
assumes *a*>*0*
assumes ∀ *x*::*real*. *a*∗*x*^*2*+*b*∗*x* + *c* ≥ *0*
shows *b*^*2* − *4* ∗ *a* ∗ *c* ≤ *0*
proof −
from *assms* **have** *0*:∀ *x*::*real*. (*a*∗*x*^*2* + *b*∗*x* + *c*)/*a* ≥*0* **by** *simp*
from *assms* **have** *1*:∀ *x*::*real*.(*b*∗*x* + *c*)/*a* = *b*/*a*∗*x* + *c*/*a* **by** (*simp add*: *add-divide-distrib*)
from *assms* **have** ∀ *x*::*real*.(*a*∗*x*^*2* + *b*∗*x* + *c*)/*a* = *x*^*2* + (*b*∗*x* + *c*)/*a*
by (*simp add*: *add-divide-eq-if-simps*(*1*))
from *1 this* **have** ∀ *x*::*real*.(*a*∗*x*^*2* + *b*∗*x* + *c*)/*a* = *x*^*2* + *b*/*a*∗*x* + *c*/*a* **by** *simp*
hence *atleastzero*:∀ *x*::*real*. *x*^*2* + *b*/*a*∗*x* + *c*/*a* ≥*0* **using** *0* **by** *simp*

from *assms* **have** *2*:∀ *x*::*real*. *x*^*2* + *b*/*a*∗*x* + *c*/*a* = *x*^*2* + *2*∗*b*/(*2*∗*a*)∗*x* + *c*/*a* + *b*^*2*/(*4*∗*a*^*2*)−*b*^*2*/(*4*∗*a*^*2*) **by** *simp*
have *simp1*:∀ *x*::*real*.(*x* + *b*/(*2*∗*a*))^*2* = *x*^*2* + *2*∗*b*/(*2*∗*a*)∗*x* + (*b*/(*2*∗*a*))^*2* **by** (*simp add*: *power2-sum*)
have (*b*/(*2*∗*a*))^*2* = *b*^*2*/(*4*∗*a*^*2*) **by** (*metis four-x-squared power-divide*)
hence ∀ *x*::*real*. *x*^*2* + *b*/*a*∗*x* + *c*/*a* = (*x* + *b*/(*2*∗*a*))^*2* + *c*/*a*−*b*^*2*/(*4*∗*a*^*2*) **using** *2 simp1* **by** *auto*
hence ∀ *x*::*real*. (*x* + *b*/(*2*∗*a*))^*2* + *c*/*a*−*b*^*2*/(*4*∗*a*^*2*) ≥*0* **using** *atleastzero* **by** *presburger*
hence *3*:∀ *x*::*real*. *b*^*2*/(*4*∗*a*^*2*)−*c*/*a*≤(*x* + *b*/(*2*∗*a*))^*2* **by** (*smt* (*verit*, *del-insts*))
have ∃ *x*::*real*. (*x* + *b*/(*2*∗*a*))^*2*=*0* **by** (*metis diff-add-cancel power-zero-numeral*)
hence *b*^*2*/(*4*∗*a*^*2*)−*c*/*a*≤*0* **using** *3* **by** *metis*
hence *4*:*4*∗*a*^*2*∗(*b*^*2*/(*4*∗*a*^*2*)−*c*/*a*)≤*0* **using** *assms* **by** (*simp add*: *mult-nonneg-nonpos*)
have *5*:*4*∗*a*^*2*∗*b*^*2*/(*4*∗*a*^*2*) = *b*^*2* **using** *assms* **by** *simp*
have *6*:*4*∗*a*^*2* ∗ *c*/*a* = *4*∗*a* ∗ *c* **using** *assms* **by** (*simp add*: *power2-eq-square*)

show *?thesis* **using** *4 5 6 assms* **by** (*simp add*: *Rings.ring-distribs*(*4*))
qed

lemma *sign-of-discriminant*:
fixes *a b c* :: *real*
assumes ∀ *x*::*real*. *a*∗*x*^*2* + *b*∗*x* + *c* ≥*0*
shows *a* ≥ *0* ⟹ *b*^*2* − *4* ∗ *a* ∗ *c* ≤ *0* *a* < *0* ⟹ *b*^*2* − *4* ∗ *a* ∗ *c* ≥ *0*
subgoal apply (*cases a=0*)
subgoal by (*metis Groups.add-ac*(*2*) *assms diff-add-cancel diff-zero mult-zero-left mult-zero-right*
nonzero-mult-div-cancel-left real-sqrt-pow2 sum-power2-le-zero-iff times-divide-eq-right zero-eq-power2)
subgoal using *qua-disc assms* **by** *simp*
done
subgoal using *assms* **by** (*smt* (*verit*, *ccfv-threshold*) *zero-compare-simps*(*8*,*12*))
done

end

lemma *cauchy-schwarz-square*:
assumes *A*∈*S* *B*∈*S*
shows |⟨*A*|*B*⟩|^2 ≤ ⟨*A*|*A*⟩ ∗ ⟨*B*|*B*⟩
proof −
consider (*some-zero*) *A=0* ∨ *B=0* | (*nonzero*) *A≠0* ∧ *B≠0* **by** *fastforce*
then show *?thesis*
proof (*cases*)
case *some-zero*
then show *?thesis*
using *finner-plus*(*1*) *finner-zero-commute*
by (*metis assms cnj-x-x cnj-x-x-geq0 eq-add-iff mult-eq-0-iff*)
next
case *nonzero* — Mostly revolves around the discriminant of a quadratic, and the inequality ⟦*0* < *?a*; ∀ *x*. *0* ≤ *?a* ∗ *x*^2 + *?b* ∗ *x* + *?c*⟧ ⟹ *?b*^2 − *4* ∗ *?a* ∗ *?c* ≤ *0* (cribbed from *Polygonal-Number-Theorem*).

let *?c* = |⟨*B*|*A*⟩| / ⟨*B*|*A*⟩
have *c-witness*: ‖*?c*‖ = *1* *?c* ∗ ⟨*B*|*A*⟩ = |⟨*B*|*A*⟩| **if** ⟨*B*|*A*⟩ ≠ *0*
using *Real-Vector-Spaces.norm-divide*[*of* ‖⟨*B*|*A*⟩‖ ⟨*B*|*A*⟩] *that*
by (*simp-all add*: *abs-complex-def*)
have ∃ *c*::*complex*. *cmod c* = *1* ∧ *c* ∗ ⟨*B*|*A*⟩ = |⟨*B*|*A*⟩|
apply (*cases* ⟨*B*|*A*⟩*=0*)
subgoal using *norm-one* **by** *fastforce*
using *c-witness* **by** *blast*
then
obtain *c* :: *complex*
where *c*: *cmod c* = *1* *c* ∗ ⟨*B*|*A*⟩ = |⟨*B*|*A*⟩|
by *blast*
{ fix *x* :: *complex* **assume** *x*: *x*∈ℝ

have $\langle (x * c) *_C A + B | (x * c) *_C A + B \rangle =$
$\langle (x * c) *_C A | (x * c) *_C A \rangle +$
$\langle (x * c) *_C A | B \rangle + \langle B | (x * c) *_C A \rangle + \langle B | B \rangle$
using *m1.mem-scale finner-plus* **by** (*simp add*: *assms finner-plus(1)*)+
also have ... $= (cmod\ (x * c))^2 * \langle A|A\rangle + \langle (x * c) *_C A|B\rangle + \langle B|(x * c) *_C A\rangle + \langle B|B\rangle$
apply (*simp add*: *complex-mod-mult-cnj del*: *Real-Vector-Spaces.of-real-power*)
using *finner-scale*[*OF assms(1)*, *of - x * c*] *m1.mem-scale*[*OF assms(2)*]
using *assms(1) complex-norm-square m1.mem-scale* **by** *auto*
also have ... $= (cmod\ (x * c))^2 * \langle A|A\rangle + cnj\ (x * c) * \langle A|B\rangle + (x * c) * \langle B|A\rangle + \langle B|B\rangle$
using *finner-scale m1.mem-scale assms* **by** *simp*
also have ... $= \|x * c\|^2 * \langle A|A\rangle +$
$cnj\ (x * c * \langle B|A\rangle) + ((x * c) * \langle B|A\rangle) + \langle B|B\rangle$
by (*metis assms complex-cnj-mult finner-cnj*)
also have ... $= |x * c|^2 * \langle A|A\rangle + cnj\ (x*|\langle B|A\rangle|) + (x*|\langle B|A\rangle|) + \langle B|B\rangle$
apply (*simp add*: *complex-of-real-cmod*)
using *c(2)* **by** (*metis Extra-Ordered-Fields.sign-simps(4) complex-cnj-mult*)
also have ... $= \|\langle A|A\rangle\| * x^2 + (2*\|\langle B|A\rangle\|)*x + \|\langle B|B\rangle\|$
unfolding *fnorm-finner-squared(2)*[*OF assms(1)*] *fnorm-finner-squared(2)*[*OF assms(2)*]
apply (*simp add*: *complex-of-real-cmod*)
using *c(1) Reals-cnj-iff x assms abs-complex-real complex-norm-square complex-of-real-cmod*
fnorm-finner-squared(1) norm-mult norm-one
by (*smt* (*verit*) *sign-simps(3,5) more-arith-simps(6) of-real-power power2-eq-square*)
finally
have $||A||_\omega{}^2 * x^2 + (2*\|\langle B|A\rangle\|)*x + ||B||_\omega{}^2 \geq 0$
unfolding *fnorm-finner-squared(2)*[*OF assms(1)*] *fnorm-finner-squared(2)*[*OF assms(2)*]
by (*metis assms finner-nonneg m1.mem-add m1.mem-scale*) **}**
note *quadratic-nonnegative = this*
have $0 \leq \|\langle A|A\rangle\| * x^2 + 2 * \|\langle B|A\rangle\| * x + \|\langle B|B\rangle\|$ **for** *x*::*real*
using *quadratic-nonnegative*[*of x*, *simplified*]
unfolding *functional-norm-def*
using *assms(1,2) fnorm-finner-squared(1) quadratic-nonnegative*
by (*smt* (*verit*, *ccfv-threshold*) *Re-complex-of-real Reals-of-real cmod-Re finner-nonneg less-eq-complex-def*
of-real-0 of-real-mult of-real-power plus-complex.sel(1))
hence $(2*\|\langle B|A\rangle\|)^2 - 4*\|\langle A|A\rangle\|*\|\langle B|B\rangle\| \leq 0$
using *sign-of-discriminant(1) zero-le-power norm-ge-zero*
by *blast*
then show *?thesis*
apply (*simp add*: *abs-complex-def*)
apply (*subst of-real-power*[*of* $\|\langle A|B\rangle\|$ *2*, *symmetric*])
using *norm-of-real of-real-0 of-real-mult*
by (*smt* (*verit*, *del-insts*) *Im-complex-of-real Re-complex-of-real assms(1,2) cnj-x-x complex-norm-square*
finner-cnj finner-nonneg fnorm-finner-squared(1) less-eq-complex-def

x-cnj-x)
qed
qed

lemma *cauchy-schwarz*:
assumes $A \in S$ $B \in S$
shows $\|\langle A|B\rangle\| \leq ||A||_\omega * ||B||_\omega$
proof −
from *cauchy-schwarz-square*[*OF assms*]
have $\|\langle A|B\rangle\|^2 \leq ||A||_\omega{}^2 * ||B||_\omega{}^2$

unfolding *abs-complex-def* **by** (*simp add*: *assms fnorm-finner-squared*(*1*) *less-eq-complex-def*)
hence $\|\langle A|B\rangle\| \leq sqrt\ (||A||_\omega{}^2 * ||B||_\omega{}^2)$

using *real-le-rsqrt* **by** *blast*
also have $sqrt\ (||A||_\omega{}^2 * ||B||_\omega{}^2) = ||A||_\omega * ||B||_\omega$

by (*simp add*: *fnorm-nonneg real-sqrt-mult*)
finally show *?thesis* **.**
qed

1.9 Identities for states on unital C^*-algebras

For now, see [1, page 49, Prop. 2.3.11]. We already have $?A \in S \Longrightarrow \omega\ (?A^\dagger) = cnj\ (\omega\ ?A)$.

lemma *state-norm-leq-inner*: $\|\omega\ A\|^2 \leq \langle A|A\rangle$ **if** $A \in S$
using *cauchy-schwarz-square*[*OF that*, *of* **1**]
unfolding *functional-inner-def*
by (*metis amult-id*(*3*) *antiautomorphism cnj-ivl cnj-x-x complex-norm-square involution-conjugate*
local.one more-arith-simps(*6*) *multiplicative.unit-closed that unit-self-adjoint*)

lemma *state-zero-if-inner-zero*: $\omega\ A = 0$ **if** $\omega\ (involution\ A \bullet A) = 0$ $A \in S$
proof−
have $(norm\ (\omega\ A))^2 = 0$
using *state-norm-leq-inner* **apply** (*simp add*: *functional-inner-def*)
by (*metis complex-of-real-mono-iff norm-le-zero-iff*
norm-power of-real-eq-0-iff of-real-power that zero-eq-power2)
thus *?thesis* **by** *simp*
qed

end

locale *Cstar-state* =
Cstar-algebra F S scale multiplication vector-norm unit involution +
positive-normalised-linear-functional S scale multiplication unit involution ω
for F :: *complex set*

and $S :: 'a$::*ab-group-add set*
and *scale* :: *complex* $\Rightarrow$ $'a \Rightarrow 'a$ (**infixr** ‹$*_C$› *75*)
and *multiplication* :: $'a \Rightarrow 'a \Rightarrow 'a$ (**infixr** ‹•› *74*)
and *vector-norm* :: $'a \Rightarrow$ *real* (‹||-||› [*65*])
and *unit* :: $'a$ (‹**1**›)
and *involution* :: $'a \Rightarrow 'a$ (-† [*99*] *100*)
and ω :: $'a \Rightarrow$*complex*
begin

lemma *mult-inner-norm-bound*:
assumes $\bigwedge A\ B.\ [\![A \in S;\ B \in S]\!] \Longrightarrow \langle B \bullet A | B \bullet A\rangle \leq ||B||^2 * \omega\ (A^\dagger \bullet A)$
— This assumption is, in fact, a theorem. It can be proven using spectral theory, by defining positive operators to enable the following ordering relation:

$$A^\dagger B^\dagger B A \leq \|B\|^2 A^\dagger A$$

See Bratteli and Robinson's textbook, p. 51 for proof of the lemma, and p. 36 for ordering on positive operators [1]. The definition of an operator as positive depends on the operator's spectrum (i.e. it is in the positive half-line, p. 32). Spectral theory is, for now, out of scope for this development.
and $A \in S$ $B \in S$
shows $|\langle B \bullet A | A\rangle| \leq \langle A|A\rangle * ||B||$
proof–
have $|\langle B \bullet A | A\rangle|^2 \leq \langle B \bullet A | B \bullet A\rangle * \langle A|A\rangle$
using *cauchy-schwarz-square*[*of* ‹$B \bullet A$› ‹A›]
by (*auto simp*: *assms*(*2*,*3*))
also have $\ldots \leq (||B||^2) * \omega\ (\mathit{involution}\ A \bullet A) * \langle A|A\rangle$
using *assms*(*1*)[*OF assms*(*2*,*3*)] *finner-nonneg*[*OF assms*(*2*)] *mult-right-mono*
by *blast*
finally have $|\langle B \bullet A | A\rangle|^2 \leq \langle A|A\rangle * (||B||^2) * \langle A|A\rangle$
by (*metis Extra-Ordered-Fields.sign-simps*(*5*) *functional-inner-def*)
then have $|\langle B \bullet A | A\rangle|^2 \leq (\langle A|A\rangle * ||B||)^2$
by (*smt* (*verit*) *of-real-power ordered-field-class.sign-simps*(*23*,*4*,*5*) *power2-eq-square*)
moreover have $|0| \leq |\langle B \bullet A | A\rangle|$ **by** (*simp add*: *abs-nn*)
ultimately have $\|\langle B \bullet A | A\rangle\| \leq Re(\langle A|A\rangle) * ||B||$
apply (*intro power2-le-imp-le*[*of* $\|\langle B \bullet A | A\rangle\|$ *Re* $(\langle A|A\rangle) * ||B||$])
subgoal using *finner-cmod-eq x-cnj-x*
by (*smt* (*verit*, *best*) *Re-complex-of-real complex-norm-square*
less-eq-complex-def of-real-mult of-real-power assms(*2*))
subgoal by (*simp add*: *assms*(*2*,*3*) *functional-inner-def nonneg'*(*1*))
done
thus *?thesis*
by (*metis Re-complex-of-real Reals-cnj-iff assms*(*2*) *of-real-mult*
complex-cnj-complex-of-real complex-is-Real-iff
complex-of-real-cmod finner-cmod-eq less-eq-complexI)
qed

end

1.10 Morphisms

locale *involutive-complex-algebras* =
  A1: *involutive-complex-algebra S1 scale1 multiplication1 unit1 involution1* +
  A2: *involutive-complex-algebra S2 scale2 multiplication2 unit2 involution2*
  for *S1* :: *'a*::*ab-group-add set*
    and *scale1* :: *complex* ⇒ *'a* ⇒ *'a* (**infixr** ‹$*_{C1}$› *75*)
    and *multiplication1* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹$\bullet_1$› *74*)
    and *unit1* :: *'a* (‹$\mathbf{1}_1$›)
    and *involution1* :: *'a*⇒*'a*
    and *S2* :: *'b*::*ab-group-add set*
    and *scale2* :: *complex* ⇒ *'b* ⇒ *'b* (**infixr** ‹$*_{C2}$› *75*)
    and *multiplication2* :: *'b* ⇒ *'b* ⇒ *'b* (**infixr** ‹$\bullet_2$› *74*)
    and *unit2* :: *'b* (‹$\mathbf{1}_2$›)
    and *involution2* :: *'b*⇒*'b*

We define *-homomorphisms to be unital always. I believe a minority of authors consider non-unital homomorphisms of rings, but then you need to consider degenerate cases (e.g. $\lambda x.\ 0$). For some discussion, see e.g. https://mathoverflow.net/questions/34332/consequences-of-not-requiring-ring-homomorphisms-to-be-unital .

locale *involutive-complex-homomorphism* = *involutive-complex-algebras* +
  fixes *f* :: *'a*⇒*'b*
  assumes *f-codomain*: *f*‘*S1* ⊆ *S2*
    and *f-unit*: *f* $\mathbf{1}_1$ = $\mathbf{1}_2$
    and *f-scale*: ⋀*r x*. ⟦*x*∈*S1*⟧ ⟹ *f* (*r* $*_{C1}$ *x*) = *r* $*_{C2}$ (*f x*)
    and *f-plus*: ⋀*x y*. ⟦*x*∈*S1*; *y*∈*S1*⟧ ⟹ *f* (*x* + *y*) = (*f x*) + (*f y*)
    and *f-multiplication*: ⋀*x y*. ⟦*x*∈*S1*; *y*∈*S1*⟧ ⟹
      f (*x* $\bullet_1$ *y*) = (*f x*) $\bullet_2$ (*f y*)
    and *f-involution*: ⋀*x*. ⟦*x*∈*S1*⟧ ⟹
      f (*involution1 x*) = *involution2* (*f x*)
begin

lemma *linear-f*: *linear-on S1 S2 scale1 scale2 f*
  apply *unfold-locales*
  using *f-plus f-scale* **by** *auto*

interpretation *linear-f*: *linear-on S1 S2 scale1 scale2 f*
  using *linear-f* **.**

end

abbreviation (**in** *involutive-complex-algebras*) *homomorphism*
  where *homomorphism f* ≡ *involutive-complex-homomorphism*
    S1 scale1 multiplication1 unit1 involution1
    S2 scale2 multiplication2 unit2 involution2
    f

locale *involutive-complex-isomorphism* = *involutive-complex-homomorphism* +

assumes *bij-f*: *bij-betw f S1 S2*

abbreviation (**in** *involutive-complex-algebras*) *isomorphism*
where *isomorphism f* ≡ *involutive-complex-isomorphism*
S1 scale1 multiplication1 unit1 involution1
S2 scale2 multiplication2 unit2 involution2
f

locale *involutive-complex-automorphism* =
involutive-complex-isomorphism S scale multiplication unit involution
S scale multiplication unit involution f
for *S scale multiplication unit involution f*

abbreviation (**in** *involutive-complex-algebra*) *automorphism*
where *automorphism f* ≡ *involutive-complex-automorphism S scale multiplication unit involution f*

locale *Cstar-algebras* =
A1: *Cstar-algebra F1 S1 scale1 multiplication1 vnorm1 unit1 involution1* +
A2: *Cstar-algebra F2 S2 scale2 multiplication2 vnorm2 unit2 involution2*
for *F1* :: *complex set*
and *S1* :: *'a*::*ab-group-add set*
and *scale1* :: *complex* ⇒ *'a* ⇒ *'a* (**infixr** ‹$*_{C1}$› *75*)
and *multiplication1* :: *'a* ⇒ *'a* ⇒ *'a* (**infixr** ‹$•_1$› *74*)
and *vnorm1* :: *'a* ⇒ *real* (‹$||\text{-}||_1$› [*65*])
and *unit1* :: *'a* (‹$\mathbf{1}_1$›)
and *involution1* :: *'a*⇒*'a*
and *F2* :: *complex set*
and *S2* :: *'b*::*ab-group-add set*
and *scale2* :: *complex* ⇒ *'b* ⇒ *'b* (**infixr** ‹$*_{C2}$› *75*)
and *multiplication2* :: *'b* ⇒ *'b* ⇒ *'b* (**infixr** ‹$•_2$› *74*)
and *vnorm2* :: *'b* ⇒ *real* (‹$||\text{-}||_2$› [*65*])
and *unit2* :: *'b* (‹$\mathbf{1}_2$›)
and *involution2* :: *'b*⇒*'b*
begin

sublocale *involutive-complex-algebras*
by *unfold-locales*

end

A C^*-homomorphism is a homomorphism of *-algebras between C^*-algebras.

locale *Cstar-homomorphism* = *Cstar-algebras* + *involutive-complex-homomorphism*

lemma (**in** *Cstar-algebras*) *homomorphismI*:
assumes *homomorphism f*
shows *Cstar-homomorphism F1 S1 scale1 multiplication1 vnorm1 unit1 involution1*
F2 S2 scale2 multiplication2 vnorm2 unit2 involution2 f
by (*simp add*: *assms Cstar-algebras-axioms Cstar-homomorphism.intro*)

locale *Cstar-isomorphism* = *Cstar-algebras* + *involutive-complex-isomorphism*

locale *Cstar-automorphism* =
Cstar-isomorphism F V scale multiplication vector-norm unit involution
F V scale multiplication vector-norm unit involution f
for *F* :: *complex set*
and *V* :: *'v*::*ab-group-add set*
and *scale* :: *complex* ⇒ *'v* ⇒ *'v* (**infixr** ‹$*_C$› *75*)
and *multiplication* :: *'v* ⇒ *'v* ⇒ *'v* (**infixr** ‹•› *74*)
and *vector-norm* :: *'v* ⇒ *real* (‹||-||› [*65*])
and *unit* :: *'v* (‹**1**›)
and *involution* :: *'v*⇒*'v* (‹-†› *73*)
and *f*
begin

lemma *is-star-aut*: *involutive-complex-automorphism V* ($*_C$) (•) **1** *involution f*
using *involutive-complex-isomorphism-axioms* **by** (*simp only*: *involutive-complex-automorphism-def*)

sublocale *as-star-aut*: *involutive-complex-automorphism V*
using *is-star-aut* **.**

end

lemma (**in** *Cstar-algebra*) *automorphismI*:
assumes *automorphism f*
shows *Cstar-automorphism F V scale multiplication vector-norm unit involution f*
using *involutive-complex-isomorphism.axioms*[*OF assms*[*unfolded involutive-complex-automorphism-def*]]
unfolding *involutive-complex-isomorphism-axioms-def*
unfolding *involutive-complex-homomorphism-def involutive-complex-homomorphism-axioms-def*
by *unfold-locales presburger*+

end

2 Completions of Normed Vector Spaces

theory *Inner-Completion*
imports
Complex-Bounded-Operators.Complex-Inner-Product

Gromov-Hyperbolicity.Metric-Completion
begin

2.1 Helper lemmas

lemma *Cauchy-norm-bounded*:
assumes *Cauchy X*
shows $\exists B.\ \forall n.\ norm\ (X\ n) < B$
by (*meson cauchy-imp-bounded assms bounded-normE-less rangeI*)

lemma *convergent-Cauchy-norm*:
fixes *u v*::*nat* ⇒ (*'a*::*real-normed-vector*)
assumes *Cauchy u*
shows *convergent* (λ*n*. *norm* (*u n*))
unfolding *norm-conv-dist* **using** *convergent-Cauchy-dist assms convergent-Cauchy convergent-const*
by *blast*

lemma *Cauchy-norm-if-Cauchy*:
assumes *Cauchy u*
shows *Cauchy* (λ*n*. *norm* (*u n*))
unfolding *norm-conv-dist*
using *assms convergent-Cauchy convergent-Cauchy-norm* **by** *auto*

lemma *lim-ge*: *convergent f* ⟹ (⋀*n*. *f n* ≥ *x*) ⟹ *lim f* ≥ *x*
for *x* :: *'a*::*linorder-topology*
using *LIMSEQ-le-const*[*of f lim f x*] **by** (*simp add*: *convergent-LIMSEQ-iff*)

lemma
assumes *convergent u*
shows *lim-Re*: *lim* (λ*n*. *Re* (*u n*)) = *Re* (*lim* (λ*n*. *u n*))
and *lim-Im*: *lim* (λ*n*. *Im* (*u n*)) = *Im* (*lim* (λ*n*. *u n*))
by (*intro limI tendsto-Re tendsto-Im*, *simp add*: *assms*[*unfolded convergent-LIMSEQ-iff*])+

lemma *Cauchy-sum*:
fixes *x1 x2* :: *nat*⇒*'a*::*real-normed-vector*
assumes *Cauchy x1 Cauchy x2*
shows *Cauchy* (*x1*+*x2*)
proof (*unfold Cauchy-def*[*of x*+*y* **for** *x y*], *intro allI impI*)
fix *e*::*real*
assume *0* < *e*
then
obtain *M1 M3*
where *M*: ∀ *m*≥*M1*. ∀ *n*≥*M1*. *dist* ((*x1*) *m*) ((*x1*) *n*) < *e*/*2*
∀ *m*≥*M3*. ∀ *n*≥*M3*. *dist* ((*x2*) *m*) ((*x2*) *n*) < *e*/*2*
using *zero-less-divide-iff zero-less-numeral*
by (*metis Cauchy-def assms*)
let *?M* = *if M1*≥*M3 then M1 else M3*
{ fix *m n* **assume** *m*≥*?M n*≥*?M*

hence *mn*: $m \geq M1$ $n \geq M1$ $m \geq M3$ $n \geq M3$ **by** *presburger+*
have $norm\ (x1\ n - x1\ m) + norm\ (x2\ m - x2\ n) < e$
using *M mn* **unfolding** *dist-norm* **by** *fastforce*
hence $norm\ ((x1 + x2)\ m - (x1 + x2)\ n) < e$
unfolding *plus-fun-apply*
by (*metis add.commute add-diff-eq diff-diff-eq norm-minus-commute norm-triangle-lt*)
hence $dist\ ((x1 + x2)\ m)\ ((x1 + x2)\ n) < e$
unfolding *dist-norm* **by** *blast* **}**
thus $\exists M.\ \forall m \geq M.\ \forall n \geq M.\ dist\ ((x1 + x2)\ m)\ ((x1 + x2)\ n) < e$
by *blast*
qed

2.2 The Cauchy completion of a normed vector space is a Banach space.

instantiation *metric-completion* :: (*real-normed-vector*) *scaleR*
begin
lift-definition *scaleR-metric-completion* :: *real* ⇒ *'a metric-completion* ⇒ *'a metric-completion*
is $\lambda r\ x.\ \lambda n.\ (scaleR\ r)\ (x\ n)$
using *bounded-linear.Cauchy bounded-linear-scaleR-right tendsto-mult-right-zero*
by (*auto simp add*: *scaleR-diff-right*[*symmetric*] *dist-norm*)
instance by *standard*
end

instantiation *metric-completion* :: (*real-normed-vector*) *real-vector*
begin

lift-definition *uminus-metric-completion* :: *'a metric-completion* ⇒ *'a metric-completion*
is *uminus*
by (*simp add*: *Cauchy-def dist-minus*)

lift-definition *zero-metric-completion* :: *'a metric-completion*
is $\lambda n.\ 0$
by (*simp add*: *Cauchy-def*)

lift-definition *plus-metric-completion* ::
'a metric-completion ⇒ *'a metric-completion* ⇒ *'a metric-completion*
is *plus*
proof (*intro conjI*; *elim conjE*)
fix *x1 x2 x3 x4* :: *nat* ⇒ *'a*
assume *cauchy*: *Cauchy x1 Cauchy x2 Cauchy x3 Cauchy x4*
assume *dist-conv*: $(\lambda n.\ dist\ (x1\ n)\ (x2\ n)) \longrightarrow 0\ (\lambda n.\ dist\ (x3\ n)\ (x4\ n)) \longrightarrow 0$
show *cauchy-sum*: *Cauchy* $(x1 + x3)$ *Cauchy* $(x2 + x4)$
using *Cauchy-sum cauchy* **by** *auto*
show $(\lambda n.\ dist\ ((x1 + x3)\ n)\ ((x2 + x4)\ n)) \longrightarrow 0$
proof −
have *dist-conv-add*: $(\lambda n.\ dist\ (x1\ n)\ (x2\ n) + dist\ (x3\ n)\ (x4\ n)) \longrightarrow 0$

```
    using dist-conv tendsto-add by fastforce
  show ?thesis
   apply (rule tendsto-sandwich[of λ-. 0 - - λn. dist (x1 n) (x2 n) + dist (x3 n)
(x4 n)])
    using dist-conv-add by (auto simp: dist-triangle-add)
 qed
qed

definition minus-metric-completion ::
  'a metric-completion ⇒ 'a metric-completion ⇒ 'a metric-completion
  where minus-metric-completion a b ≡ a + (−b)

instance
 apply intro-classes
 subgoal by (transfer, simp add: Cauchy-sum Groups.add-ac)
 subgoal by (transfer, simp add: Groups.add-ac Cauchy-sum)
 subgoal apply transfer by (transfer, simp add: plus-fun-def)
 subgoal by (transfer, simp add: Cauchy-def)
 subgoal by (simp add: minus-metric-completion-def)
 subgoal apply (transfer, auto)
  subgoal using Cauchy-sum bounded-linear.Cauchy
      bounded-linear-scaleR-right by fastforce
 subgoal by (simp add: Cauchy-sum bounded-linear.Cauchy bounded-linear-scaleR-right)
  subgoal by (simp add: scaleR-add-right)
  done
 subgoal apply (transfer, auto)
  using bounded-linear.Cauchy bounded-linear-scaleR-right apply blast
 using bounded-linear.Cauchy bounded-linear-scaleR-right apply (metis Cauchy-sum)
  using dist-self scaleR-add-left by (metis (mono-tags, lifting) LIMSEQ-def)
subgoal by (transfer, simp add: bounded-linear.Cauchy bounded-linear-scaleR-right)
 subgoal by (transfer, simp)
 done
end


instantiation metric-completion :: (complex-normed-vector) scaleC
begin
lift-definition scaleC-metric-completion :: complex ⇒ 'a metric-completion ⇒ 'a
metric-completion
 is λc x. λn. (scaleC c) (x n)
using bounded-clinear.Cauchy bounded-clinear-scaleC-right tendsto-mult-right-zero
 by (auto simp add: scaleC-diff-right[symmetric] dist-norm)
instance
 apply intro-classes
 unfolding scaleR-metric-completion-def scaleC-metric-completion-def
 by (simp add: scaleR-scaleC)
end

instance metric-completion :: (complex-normed-vector) complex-vector
```

```
  apply intro-classes
  subgoal apply (transfer, auto)
    subgoal using Inner-Completion.Cauchy-sum bounded-clinear.Cauchy
        bounded-clinear-scaleC-right by fastforce
   subgoal by (simp add: Cauchy-sum bounded-clinear.Cauchy bounded-clinear-scaleC-right)
    subgoal by (simp add: scaleC-add-right)
    done
  subgoal apply (transfer, auto)
    using bounded-clinear.Cauchy bounded-clinear-scaleC-right apply blast
   using bounded-clinear.Cauchy bounded-clinear-scaleC-right apply (metis Cauchy-sum)
    using dist-self scaleC-add-left by (metis (mono-tags, lifting) LIMSEQ-def)
 subgoal by (transfer, simp add: bounded-clinear.Cauchy bounded-clinear-scaleC-right)
  subgoal by (transfer, simp)
  done

instantiation metric-completion :: (real-normed-vector) banach
begin

lift-definition norm-metric-completion :: 'a metric-completion ⇒ real
  is λX. lim (λn. norm (X n))
  unfolding dist-norm
  apply (rule convergent-add-null(2))
  subgoal using convergent-Cauchy-dist zero-metric-completion.rsp by force
  using Lim-null-comparison eventually-sequentially norm-triangle-ineq3
  by (metis (mono-tags, lifting) real-norm-def)

definition sgn-metric-completion :: 'a metric-completion ⇒ 'a metric-completion
  where sgn-metric-completion x = x /R norm x

lemma metric-completion-triangle-ineq:
  fixes x y :: 'a metric-completion
  shows norm (x + y) ≤ norm x + norm y
proof −
  have 1: lim (λn. norm (x n + y n)) ≤ lim (λn. norm (x n)) + lim (λn. norm (y
n))
    if cauchy: Cauchy x Cauchy y
    for x y :: nat ⇒ 'a
  proof −
    have lim-norm-sum-distrib: lim (λn. norm (x n) + norm (y n)) =
        lim (λn. norm (x n)) + lim (λn. norm (y n))
      apply (intro limI)
      apply (rule tendsto-add)
    using cauchy convergent-Cauchy-dist convergent-LIMSEQ-iff zero-metric-completion.rsp
      by fastforce+
    have conv-norm-plus: convergent (λn. norm (x n + y n))
      apply (simp only: norm-conv-dist)
      apply (intro convergent-Cauchy-dist)
      apply (metis (no-types, lifting) ext Cauchy-sum cauchy(1,2) plus-fun-apply)
      by (simp add: zero-metric-completion.rsp)
```

have *conv-plus-norm*: *convergent* (*λn. norm* (*x n*) + *norm* (*y n*))
apply (*rule convergent-add*)
apply (*simp-all only*: *norm-conv-dist*)
apply (*rule convergent-Cauchy-dist*[*OF cauchy*(*1*) *zero-metric-completion.rsp*[*THEN conjunct1*]])
by (*rule convergent-Cauchy-dist*[*OF cauchy*(*2*) *zero-metric-completion.rsp*[*THEN conjunct1*]])
thus *?thesis*
apply (*simp flip*: *lim-norm-sum-distrib*)
apply (*rule lim-mono*[*of* - (*λn. norm* (*x n* + *y n*)) (*λn. norm* (*x n*) + *norm* (*y n*))])
using *conv-norm-plus conv-plus-norm*
by (*simp-all add*: *norm-triangle-ineq convergent-LIMSEQ-iff*[*symmetric*])
qed
show *?thesis*
apply *transfer*
using *1* **by** *auto*
qed

instance
apply *intro-classes* — Remember to unfold non-lifted definitions before transferring!
unfolding *minus-metric-completion-def sgn-metric-completion-def*
subgoal by (*transfer*, *simp add*: *dist-norm*)
subgoal apply (*transfer*, *auto*)
using *bounded-linear.Cauchy bounded-linear-scaleR-right* **by** *blast*
subgoal apply (*transfer*, *auto*)
subgoal using *zero-metric-completion.rsp* **by** *blast*
subgoal using *convergent-Cauchy-dist convergent-LIMSEQ-iff zero-metric-completion.rsp* **by** *force*
subgoal by (*simp add*: *tendsto-Lim*)
done
subgoal using *metric-completion-triangle-ineq* **.**
subgoal
apply (*transfer*, *auto*)
apply (*intro limI*)
apply (*rule tendsto-mult*)
apply *blast*
apply (*simp only*: *convergent-LIMSEQ-iff*[*symmetric*] *norm-conv-dist*)
apply (*rule convergent-Cauchy-dist*)
using *zero-metric-completion.rsp* **by** *simp-all*
done
end

instance *metric-completion* :: (*complex-normed-vector*) *cbanach*
apply *intro-classes*
apply (*transfer*, *auto*)
apply (*intro limI*)

apply (*rule tendsto-mult*)
apply *blast*
apply (*simp only*: *convergent-LIMSEQ-iff*[*symmetric*] *norm-conv-dist*)
apply (*rule convergent-Cauchy-dist*)
using *zero-metric-completion.rsp* **by** *simp-all*

2.3 The Cauchy completion of an inner product space is a Hilbert space.

instantiation *metric-completion* :: (*complex-inner*) *chilbert-space*
begin

lift-definition *cinner-metric-completion* ::
'a metric-completion ⇒ *'a metric-completion* ⇒ *complex*
is λ*x y*. *lim* (λ*n*. *cinner* (*x n*) (*y n*))
proof (*elim conjE*)
fix *x1 x2 x3 x4* :: *nat* ⇒ *'a*
assume *cauchy*: *Cauchy x1 Cauchy x2 Cauchy x3 Cauchy x4*
and *dist-conv*: (λ*n*. *dist* (*x1 n*) (*x2 n*)) ⟶ *0*
(λ*n*. *dist* (*x3 n*) (*x4 n*)) ⟶ *0*

have *1*: (λ*x*. *cmod* ((*x1 x* − *x2 x*) ·$_C$ *x3 x*)) ⟶ *0*
if *cauchy*: *Cauchy x1 Cauchy x2 Cauchy x3*
and *dist-conv*: (λ*n*. *dist* (*x1 n*) (*x2 n*)) ⟶ *0*
for *x1 x2 x3* :: *nat* ⇒ *'a*
proof −
obtain *B* **where** *B*: ∀ *n*. *norm* (*x3 n*) < *B*
using *Cauchy-norm-bounded*[*OF cauchy*(*3*)] **by** *blast*
have ∗: (λ*n*. *norm* (*x1 n* − *x2 n*) ∗ *B*) ⟶ *0*
apply (*rule tendsto-mult*[**where** *a*=*0* **and** *b*=*B*, *simplified*])
by (*metis* (*no-types*, *lifting*) *ext dist-conv dist-norm*) *simp*
have *norm-mult-tendsto-0*: (λ*n*. *norm* (*x1 n* − *x2 n*) ∗ *norm* (*x3 n*)) ⟶ *0*
apply (*rule tendsto-sandwich*[**where** *f*=λ*n*. *0* **and** *h*=λ*n*. *norm* (*x1 n* − *x2 n*) ∗ *B*])
using ∗ *B* **by** (*simp-all add*: *less-imp-le mult-left-mono*)
show *?thesis*
apply (*rule tendsto-sandwich*[**where** *f*=λ*n*. *0* **and** *h*=λ*n*. *norm* (*x1 n* − *x2 n*) ∗ *norm* (*x3 n*)])
by (*simp-all add*: *Cauchy-Schwarz-ineq2 norm-mult-tendsto-0*)
qed

have *2*: (λ*n*. *x1 n* ·$_C$ *x3 n* − *x2 n* ·$_C$ *x4 n*) ⟶ *0*
proof −
have ∗: *x1 n* ·$_C$ *x3 n* − *x2 n* ·$_C$ *x4 n* =
(*x1 n* − *x2 n*) ·$_C$ *x3 n* + *x2 n* ·$_C$ (*x3 n* − *x4 n*) **for** *n*
by (*simp add*: *cinner-simps*(*3*,*4*))
show *?thesis*
apply (*simp only*: ∗)
apply (*rule tendsto-add*[*of* - *0* - - *0*, *simplified*]; *rule tendsto-norm-zero-cancel*)

```
    apply (rule 1[OF cauchy(1−3) dist-conv(1)])
   apply (subst complex-mod-cnj[symmetric])
   apply (subst cinner-commute[symmetric])
   by (rule 1[OF cauchy(3,4,2) dist-conv(2)])
 qed

 show lim (λn. x1 n ·C x3 n) = lim (λn. x2 n ·C x4 n)
  apply (intro convergent-add-null(2))
 using 2 by (simp-all add: Cauchy-cinner-Cauchy cauchy(2,4) complex-Cauchy-convergent)
qed


instance
proof (intro-classes; transfer; elim conjE)
 have conv-cinner-self: convergent (λn. x n ·C x n)
  if Cauchy x for x :: nat ⇒ 'a
  using Cauchy-cinner-Cauchy Cauchy-convergent-iff that by blast

 fix x :: nat ⇒ 'a
 assume x: Cauchy x
 note conv-xx = conv-cinner-self[OF x]

 have 0 ≤ Re (lim (λn. x n ·C x n))
  unfolding lim-Re[symmetric, OF conv-xx]
  apply (rule lim-ge)
  using cinner-ge-zero less-eq-complex-def
  by (auto simp add: Cauchy-Re Cauchy-cinner-Cauchy real-Cauchy-convergent
x)
 moreover have Im (lim (λn. x n ·C x n)) = 0
  by (simp add: lim-Im[symmetric, OF conv-xx])
 ultimately show 0 ≤ lim (λn. x n ·C x n)
  by (simp add: less-eq-complex-def)

 show lim (λn. norm (x n)) = sqrt (cmod (lim (λn. x n ·C x n)))
  apply (intro limI)
  apply (rule tendsto-real-sqrt[
       where f=λn. cmod (x n ·C x n) for x n,
       folded norm-eq-sqrt-cinner])
   using Cauchy-cinner-Cauchy convergent-eq-Cauchy limI tendsto-norm x by
fastforce

 show (lim (λn. x n ·C x n) = 0) =
      (Cauchy x ∧ Cauchy (λn. 0::'a) ∧ (λn. dist (x n) 0) ⟶ 0)
 proof −
  have Cauchy-0: Cauchy (λn. 0::'a)
   using zero-metric-completion.rsp by simp
  { assume lim-cinner-self: lim (λn. x n ·C x n) = 0
   — Notice convergence Cauchy ?x ⟹ convergent (λn. ?x n ·C ?x n) is proved
completely independently!
```

```
    have tendsto-cinner-self: (λn. cmod (x n ·C x n)) ⟶ 0
      using lim-cinner-self conv-xx unfolding convergent-LIMSEQ-iff
      by (simp add: tendsto-norm-zero)
    have (λn. norm (x n)) ⟶ 0
      unfolding norm-eq-sqrt-cinner[of x n for n]
      apply (rule tendsto-real-sqrt[of - 0, simplified])
      using tendsto-cinner-self . }
  note t0-if-inner-t0 = this
  { assume (λn. norm (x n)) ⟶ 0
    hence lim (λn. x n ·C x n) = 0
      apply (intro limI)
      using tendsto-cinner tendsto-norm-zero-cancel by fastforce }
  note inner-t0-if-t0 = this
  show ?thesis
    using t0-if-inner-t0 inner-t0-if-t0 x Cauchy-0 by auto
qed

fix y :: nat ⇒ 'a assume y: Cauchy y

show lim (λn. x n ·C y n) = cnj (lim (λn. y n ·C x n))
  apply (intro limI)
  unfolding lim-cnj[of λn. x n ·C y n for x y, unfolded cinner-commute', of - -
- sequentially]
  using Cauchy-cinner-Cauchy convergent-eq-Cauchy limI x y by blast

{ fix r :: complex
  have lim (λn. cnj r * (x n ·C y n)) = cnj r * lim (λn. x n ·C y n)
    apply (intro limI tendsto-mult)
    using Cauchy-cinner-Cauchy convergent-eq-Cauchy limI x y by blast+
  thus lim (λn. r *C x n ·C y n) = cnj r * lim (λn. x n ·C y n) by simp }

fix z :: nat ⇒ 'a assume z: Cauchy z
show lim (λn. ((x + y) n) ·C z n) = lim (λn. x n ·C z n) + lim (λn. y n ·C z n)
  apply (simp only: cinner-add-left plus-fun-def)
  apply (intro limI tendsto-add)
  using Cauchy-cinner-Cauchy convergent-eq-Cauchy limI x y z by blast+
qed
end
```

2.4 The embedding *to-metric-completion*

And now we get all sorts of metric lemmas for inner products, e.g. an isometry (see *isometry-on UNIV to-metric-completion*) into a dense subset (see *closure* (*range to-metric-completion*) = *UNIV*) of a *chilbert-space*. This isometry is (complex-)linear, and preserves norms and inner products.

```
lemma to-metric-completion-linear: linear to-metric-completion (is ‹linear ?g›)
  and to-metric-completion-clinear: clinear to-metric-completion (is ‹clinear ?f›)
proof −
  note plus-abs-mc = plus-metric-completion.abs-eq
```

[*symmetric*, *of* $\lambda n.\ b1\ \lambda n.\ b2$ **for** *b1 b2*, *simplified*, *unfolded plus-fun-def*]
have $\bigwedge r\ b.\ ?g\ (r *_R b) = r *_R\ ?g\ b\ \bigwedge r\ b.\ ?f\ (r *_C b) = r *_C\ ?f\ b$
unfolding *to-metric-completion-def*
by (*simp-all add*: *Cauchy-def scaleR-metric-completion.abs-eq scaleC-metric-completion.abs-eq*)
moreover have $\bigwedge b1\ b2.\ ?g\ (b1 + b2) = ?g\ b1 + ?g\ b2\ \bigwedge b1\ b2.\ ?f\ (b1 + b2) = ?f\ b1 + ?f\ b2$
unfolding *to-metric-completion-def*
by (*rule plus-abs-mc*; *simp add*: *Cauchy-def*)+
ultimately show *linear ?g clinear ?f*
by *unfold-locales* (*auto*)
qed

lemma *to-metric-completion-norm*:
shows *norm* (*to-metric-completion c*) = *norm c*
unfolding *to-metric-completion-def*
apply (*rule norm-metric-completion.abs-eq*[*of* $\lambda n.\ c$, *simplified*])
by (*simp add*: *Cauchy-def*)

lemma *to-metric-completion-bounded-clinear*:
shows *bounded-clinear to-metric-completion*
apply *unfold-locales*
apply (*simp add*: *linear-add to-metric-completion-linear*)
apply (*simp add*: *complex-vector.linear-scale to-metric-completion-clinear*)
by (*metis mult.right-neutral norm-imp-pos-and-ge to-metric-completion-norm*)

lemma *to-metric-completion-cinner*:
(*to-metric-completion c*) $\cdot_C$ (*to-metric-completion d*) = $c \cdot_C d$
unfolding *to-metric-completion-def*
apply (*rule cinner-metric-completion.abs-eq*[*of* $\lambda n.\ b1\ \lambda n.\ b2$ **for** *b1 b2*, *simplified*])
by (*simp-all add*: *Cauchy-def*)

end

3 The Gelfand–Naimark–Segal Construction

theory *Gelfand-Naimark-Segal*
imports
Inner-Completion
Cstar-Algebra-On
Complex-Bounded-Operators.Complex-Bounded-Linear-Function
Smooth-Manifolds.Analysis-More
begin

unbundle *lifting-syntax*

3.1 Strengthen lifting lemmas

3.1.1 Lifting to the metric completion

lemma *inj-to-metric-completion*: *inj to-metric-completion*
using *isometry-on-injective to-metric-completion-isometry* **by** *blast*

definition *from-metric-completion* ≡ *the-inv to-metric-completion*

lemma *lift-to-metric-completion2*:
fixes *f*::('*a*::*metric-space*) ⇒ ('*b*::*complete-space*)
assumes *uniformly-continuous-on UNIV f*
shows ∃ *g*. (*uniformly-continuous-on UNIV g*)
∧ (*f* = *g o to-metric-completion*)
∧ (∀ *x* ∈ *range to-metric-completion*. *g x* = *f* (*from-metric-completion x*))
unfolding *from-metric-completion-def*
using *lift-to-metric-completion*[*OF assms*]
by (*simp add*: *inj-to-metric-completion the-inv-f-f*)

lemma *lift-to-metric-completion-isometry2*:
fixes *f*::('*a*::*metric-space*) ⇒ ('*b*::*complete-space*)
assumes *isometry-on UNIV f*
shows ∃ *g*. *isometry-on UNIV g*
∧ *range g* = *closure*(*range f*)
∧ *f* = *g o to-metric-completion*
∧ (∀ *x* ∈ *range to-metric-completion*. *g x* = *f* (*from-metric-completion x*))
unfolding *from-metric-completion-def*
using *lift-to-metric-completion-isometry*[*OF assms*]
by (*simp add*: *inj-to-metric-completion the-inv-f-f*)

lemma *dense-imp-conv-seq*:
assumes *closure S* = *U x*∈*U*
shows ∃ *s*. (*s n*) ∈ *S* ∧ *s* ⟶ *x*
using *LIMSEQ-if-less*
by (*metis* (*full-types*, *lifting*) *ext all-not-in-conv*
assms closed-empty closure-closed order.strict-iff-order)

lemma *to-metric-completion-sequence-ex*:
shows ∃ *s*. (*to-metric-completion o s*) ⟶ *x*
using *dense-imp-conv-seq*[*of range to-metric-completion UNIV*]
by (*smt* (*verit*, *best*) *UNIV-I closure-sequential function-factors-right image-iff*
to-metric-completion-dense′)

lemma *cinner-extensionality-dense*:
assumes *dense-S*: *closure S* = *UNIV*
and *inner-on-S*: ∀ *z*∈*S*. *cinner z x* = *cinner z y*
shows *x* = *y*
proof (*rule cinner-extensionality*)
fix *z* :: '*a*::*complex-inner*
obtain *s* **where** *s*: *s* ⟶ *z* ⋀*n*::*nat*. (*s n*) ·$_C$ *x* = (*s n*) ·$_C$ *y*

by (*metis UNIV-I closure-sequential dense-S inner-on-S*)
— We could obtain $\forall\, n.\ s\ n \in S$ at the same time, but this is not needed here.
have $(\lambda n.\ s\ n \cdot_C x) \longrightarrow z \cdot_C x$
and$(\lambda n.\ s\ n \cdot_C x) \longrightarrow z \cdot_C y$
apply (*simp add*: *s(1) tendsto-cinner*)
unfolding *s(2)* **by** (*simp add*: *s(1) tendsto-cinner*)
thus $z \cdot_C x = z \cdot_C y$
using *LIMSEQ-unique* **by** *blast*
qed

lemma *cinner-extensionality-metric-completion*:
assumes $\forall\, z.$ *cinner* (*to-metric-completion z*) $x =$ *cinner* (*to-metric-completion z*) y
shows $x = y$
using *cinner-extensionality-dense*[*OF to-metric-completion-dense'*] *assms*
by *blast*

lemma *completion-ext-simp1*:
fixes f::('*m*::*metric-space*) $\Rightarrow$ ('*c*::*complete-space*) **and** g
defines $D \equiv$ *range to-metric-completion*
shows ($f = g$ *o to-metric-completion*) $\longleftrightarrow$
$(\forall\, x \in D.\ g\ x = f$ (*from-metric-completion x*))
unfolding *assms o-def from-metric-completion-def*
using *f-the-inv-into-f the-inv-f-f inj-to-metric-completion*
by (*metis rangeI*)

lemma *lift-to-metric-completion4*:
fixes f::('*m*::*metric-space*) $\Rightarrow$ ('*c*::*complete-space*)
assumes *uniformly-continuous-on UNIV f*
shows $\exists!g.$ (*uniformly-continuous-on UNIV g*)
$\wedge$ ($f = g \circ$ *to-metric-completion*)
proof −
{ fix *x y* **and** *v* :: '*m metric-completion*
assume *x*: *isUCont x f* = *x* ∘ *to-metric-completion*
and *y*: *isUCont y f* = *y* ∘ *to-metric-completion*
have *x3*: $\forall\, b \in$ *range to-metric-completion.* *x b* = *f* (*from-metric-completion b*)
and *y3*: $\forall\, b \in$ *range to-metric-completion.* *y b* = *f* (*from-metric-completion b*)
using *completion-ext-simp1 x(2) y(2)* **by** *blast+*
have $\exists\, s.$ (*to-metric-completion o s*) $\longrightarrow v$
using *to-metric-completion-sequence-ex* **.**
then obtain *s* :: *nat*⇒'*m* **where** *s*: (*to-metric-completion* ∘ *s*) $\longrightarrow v$
by *blast*
let *?s* = *to-metric-completion o s*
have *x-to-x*: (*x* ∘ *?s*) $\longrightarrow x\ v$
and *y-to-y*: (*y* ∘ *?s*) $\longrightarrow y\ v$
using *continuous-at-sequentially isUCont-isCont s x(1) y(1)* **by** *blast+*
moreover have $\forall\, b \in$ *range to-metric-completion.* *x b* = *y b*
using *x3 y3* **by** *presburger*
ultimately have *x-to-y*: (*x* ∘ *?s*) $\longrightarrow y\ v$

by (*metis fun.map-comp x(2) y(2)*)
have *x v = y v*
using *LIMSEQ-unique x-to-x x-to-y* **by** *blast*
} note *1 = this*
have ∃!*x. isUCont x* ∧
f = x ∘ *to-metric-completion* ∧
(∀ *xa*∈*range to-metric-completion.*
x xa = f (*from-metric-completion xa*))
by (*metis* (*no-types*, *lifting*) *ext 1 assms lift-to-metric-completion2*)
thus *?thesis*
using *completion-ext-simp1*
by (*metis* (*no-types*, *lifting*))
qed

lemma *lift-to-metric-completion3*:
fixes *f*::(*'m*::*metric-space*) ⇒ (*'c*::*complete-space*)
assumes *uniformly-continuous-on UNIV f*
shows ∃!*g.* (*uniformly-continuous-on UNIV g*)
∧ (*f = g o to-metric-completion*)
∧ (∀ *x* ∈ *range to-metric-completion. g x = f* (*from-metric-completion x*))
by (*metis assms lift-to-metric-completion2 lift-to-metric-completion4*)

3.1.2 Lifting a bounded operator from an inner product space to a Hilbert space

lemma *to-metric-completion-cblinfun-exists*:
fixes *f* :: ‹*'a*::*complex-normed-vector* ⇒ *'b*::*cbanach*›
assumes *f-add*: ‹⋀*x y. f* (*x* + *y*) = *f x* + *f y*›
assumes *f-scale*: ‹⋀*c x y. f* (*c* ∗*C x*) = *c* ∗*C f x*›
assumes *bounded*: ‹⋀*x. norm* (*f x*) ≤ *B* ∗ *norm x*›
shows *cblinfun-extension-exists* (*range to-metric-completion*)
(*map-fun* (*from-metric-completion*) *id f*)
(**is** ‹*cblinfun-extension-exists ?D ?f*›)
proof −
interpret *f*: *bounded-clinear f*
by *unfold-locales* (*use f-add f-scale bounded* **in** ‹*auto simp*: *mult.commute*›)
obtain *K* **where** *K*: ⋀*x. norm* (*f x*) ≤ *norm x* ∗ *K*
using *f.real.bounded* **by** *blast*
interpret *m*: *bounded-clinear to-metric-completion*
using *to-metric-completion-bounded-clinear* **.**

show *?thesis*
proof (*rule cblinfun-extension-exists-bounded-dense*)
fix *c x y*
assume *x*: *x* ∈ *?D*
show *?f* (*c* ∗*C x*) = *c* ∗*C ?f x*
unfolding *map-fun-def from-metric-completion-def* **apply** *simp*
using *the-inv-f-f*[*OF inj-to-metric-completion*]

```
    by (metis x assms(2) m.scale rangeE)
   show norm (?f x) ≤ K * norm x
    unfolding map-fun-def from-metric-completion-def apply simp
    using the-inv-f-f[OF inj-to-metric-completion]
    using to-metric-completion-norm
    by (metis Extra-Ordered-Fields.sign-simps(5) K rangeE x)
  assume y: y ∈ ?D
  show ?f (x + y) = ?f x + ?f y
    unfolding map-fun-def from-metric-completion-def apply simp
  using the-inv-f-f[OF inj-to-metric-completion] f-the-inv-into-f[OF inj-to-metric-completion]
    using m.add f.add
    by (metis (no-types, lifting) x y)
 qed (use range-is-csubspace to-metric-completion-clinear in blast,
  use to-metric-completion-dense' in blast)
qed

lemma to-metric-completion-cblinfun-exists':
 fixes f :: ‹('a::complex-normed-vector, 'b::cbanach) cblinfun›
 shows cblinfun-extension-exists (range to-metric-completion)
    (map-fun (from-metric-completion) id f)
 using to-metric-completion-cblinfun-exists
      cblinfun.add-right cblinfun.scaleC-right norm-cblinfun
 by meson
```

3.2 The C^*-algebra of bounded operators on a Hilbert space

The assumption on the next lemma is a nondegeneracy condition found as the defining assumption of *assoc-algebra-1-on*. It's similar also to the class *zero-neq-one*, which enters for example into *ring-1*, *field*, and *one-dim*.

```
lemma Cstar-cblinfun:
 includes cblinfun-syntax
 assumes (id-cblinfun :: ('b ⇒CL ('b::chilbert-space))) ≠ 0
  shows Cstar-algebra UNIV (UNIV :: ('b ⇒CL ('b::chilbert-space)) set) (*C)
(oCL) norm id-cblinfun adj
 (is Cstar-algebra ?C ?B - - - - -)
proof −
 interpret normed-vector-space-on ?C cmod ?B (*C) norm
   by (unfold-locales,
   auto simp: norm-triangle-ineq norm-mult-ineq norm-mult complex-vector.vector-space-assms(1,2))

 interpret Group-Theory.monoid ?B (+) 0
   by unfold-locales auto
 interpret involutive-complex-algebra ?B (*C) (oCL) id-cblinfun adj
   by (unfold-locales,
   auto simp: assms cblinfun-compose-add-left cblinfun-compose-add-right lift-cblinfun-comp(2)
       add-eq-0-iff adj-plus)

 have complete ?B
   by (simp add: complete-UNIV)
```

then have *mcomplete*
unfolding *mcomplete-iff-complete*[*symmetric*] *dist-cblinfun-def totalized-induced-metric-def*
by (*simp add*: *norm-minus-commute*)
show *?thesis*
apply *unfold-locales*
using ‹*mcomplete*› **by** (*auto simp*:
norm-is-Proj-nonzero norm-cblinfun-compose power2-eq-square)
qed

The above can be used e.g. as *Cstar-algebra UNIV UNIV* ($*_C$) *cblinfun-compose norm id-cblinfun adj*, or using *zero-neq-one*.

3.3 Type class of C^*-algebras

Make a bundle to (de)activate dagger notation for adjoints of functions between types in the class of complex inner product spaces.

bundle *cadjoint-syntax* **begin**
notation *Complex-Inner-Product.cadjoint* (-† [*99*] *100*)
end
unbundle *no cadjoint-syntax*

class *cstar* = *complex-normed-algebra-1* + *cbanach* +
fixes *involution*::*'a*⇒*'a* (-† [*99*] *100*)
assumes *ivl-scaleC*: $(c *_C s)^\dagger$ = *cnj c* $*_C$ $s^\dagger$ — ⟦*involutive-complex-algebra ?S ?scale ?multiplication ?unit ?involution*; *?s* ∈ *?S*⟧ ⟹ *?involution* (*?scale ?c ?s*) = *?scale* (*cnj ?c*) (*?involution ?s*)
and *ivl-plus* [*simp*]: $(x+y)^\dagger$ = $x^\dagger$ + $y^\dagger$ — ⟦*involutive-ring ?R ?addition ?multiplication ?zero ?unit ?involution*; *?x* ∈ *?R*; *?y* ∈ *?R*⟧ ⟹ *?involution* (*?addition ?x ?y*) = *?addition* (*?involution ?x*) (*?involution ?y*)
and *involution*[*simp*]: $(x^\dagger)^\dagger$ = *x* — ⟦*involutive-semigroup ?S ?pls ?involution*; *?x* ∈ *?S*⟧ ⟹ *?involution* (*?involution ?x*) = *?x*
and *ivl-times* [*simp*]: $(x * y)^\dagger$ = $y^\dagger$ * $x^\dagger$ — ⟦*involutive-semigroup ?S ?pls ?involution*; *?x* ∈ *?S*; *?y* ∈ *?S*⟧ ⟹ *?involution* (*?pls ?x ?y*) = *?pls* (*?involution ?y*) (*?involution ?x*)
and *ivl-isometric*: *norm* ($A^\dagger$) = *norm A* — ⟦*Cstar-algebra ?F ?V ?scale ?multiplication ?vector-norm ?unit ?involution*; *?A* ∈ *?V*⟧ ⟹ *?vector-norm* (*?involution ?A*) = *?vector-norm ?A*
and *Cstar*: *norm* ($A^\dagger$ * *A*) = (*norm* ($A^\dagger$)) * (*norm A*) — ⟦*Cstar-algebra ?F ?V ?scale ?multiplication ?vector-norm ?unit ?involution*; *?A* ∈ *?V*⟧ ⟹ *?vector-norm* (*?multiplication* (*?involution ?A*) *?A*) = *?vector-norm* (*?involution ?A*) * *?vector-norm ?A*
begin

A class analogue of *Cstar-algebra*, reproducing the axioms *Cstar-algebra-axioms ?V ?multiplication ?vector-norm ?unit ?involution* ≡ ((∀ *A B*. *A* ∈ *?V* ⟶ *B* ∈ *?V* ⟶ *?vector-norm* (*?multiplication A B*) ≤ *?vector-norm A* * *?vector-norm B*) ∧ *?vector-norm ?unit* = *1*) ∧ (∀ *A*. *A* ∈ *?V* ⟶ *?vector-norm* (*?involution A*) = *?vector-norm A*) ∧ (∀ *A*. *A* ∈ *?V* ⟶

?vector-norm (*?multiplication* (*?involution A*) *A*) = *?vector-norm* (*?involution A*) ∗ *?vector-norm A*) one-by-one. Some assumptions of *Cstar-algebra* are in *real-normed-algebra*, compare:

- ⟦*Cstar-algebra ?F ?V ?scale ?multiplication ?vector-norm ?unit ?involution*; *?A* ∈ *?V*; *?B* ∈ *?V*⟧ ⟹ *?vector-norm* (*?multiplication ?A ?B*) ≤ *?vector-norm ?A* ∗ *?vector-norm ?B* with *norm* (*?x* ∗ *?y*) ≤ *norm ?x* ∗ *norm ?y*,
- *Cstar-algebra ?F ?V ?scale ?multiplication ?vector-norm ?unit ?involution* ⟹ *?vector-norm ?unit* = *1* with *norm 1* = *1*.

end

bundle *cstar-syntax* **begin**
notation *involution* (-† [*99*] *100*)
end

unbundle *cstar-syntax*

lemma *cstar-locale*: *Cstar-algebra UNIV* (*UNIV*::*'a*::*cstar set*)
scaleC times norm 1 involution
proof −
have *0*: *Metric-space.mcomplete* (*UNIV*::*'a set*)
(*normed-vector-space-on.totalized-induced-metric UNIV norm*)
by (*simp add*: *complete-UNIV totalized-induced-metric-eq-dist*)
have *1*: *monoid.invertible UNIV* (+) *0 u*
for *u* :: *'a*
proof −
interpret *Group-Theory.monoid UNIV* (+) *0*::*'a*
by *unfold-locales simp-all*
show *?thesis*
unfolding *invertible-def*[*simplified*]
by (*intro exI*[*of* - −*u*]) *simp*
qed
have *2*: *involution u* ∗ *involution* (*of-complex c*) =
of-complex (*cnj c*) ∗ *involution u*
for *u* :: *'a* **and** *c*::*complex*
using *ivl-scaleC* **by** *auto*
have *3*: *norm* (*of-complex c* ∗ *u*) = *cmod c* ∗ *norm u*
for *u* :: *'a* **and** *c*::*complex*
by (*metis norm-scaleC scaleC-conv-of-complex*)
have *4*: *v* ∗ (*of-complex c* ∗ *u*) = *of-complex c* ∗ (*v* ∗ *u*)
for *u v* :: *'a* **and** *c*::*complex*
by (*metis mult-scaleC-right scaleC-conv-of-complex*)
show *?thesis*
apply *unfold-locales*
apply (*simp-all add*: *ivl-isometric Cstar*

norm-triangle-ineq norm-mult Groups.mult-ac(1)
nat-distrib(2) cross3-simps(23) norm-mult-ineq)
using *0 1 2 3 4* **by** *simp-all*
qed

We *have to* interpret *cstar* as a *Cstar-algebra* in stages, and name the interpretation of the *involutive-complex-algebra*, otherwise there is a naming conflict between $[\![$*algebra-on ?S ?scale ?amult*; $\forall x{\in}?S.$ *?amult x x = 0*; $\forall x{\in}?S.\ \forall y{\in}?S.\ \forall z{\in}?S.$ *0 = ?amult x (?amult y z) + ?amult y (?amult z x) + ?amult z (?amult x y)*$]\!] \Longrightarrow$ *lie-algebra ?S ?scale ?amult* and $[\![$*vector-space-on ?S ?scale*; $\bigwedge x\ y\ z.\ [\![x \in ?S;\ y \in ?S;\ z \in ?S]\!] \Longrightarrow$ *?lie-bracket (x + y) z = ?lie-bracket x z + ?lie-bracket y z* $\wedge$ *?lie-bracket z (x + y) = ?lie-bracket z x + ?lie-bracket z y*; $\bigwedge a\ x\ y.\ [\![x \in ?S;\ y \in ?S]\!] \Longrightarrow$ *?lie-bracket (?scale a x) y = ?scale a (?lie-bracket x y)* $\wedge$ *?lie-bracket x (?scale a y) = ?scale a (?lie-bracket x y)*; $\bigwedge x\ y.\ [\![x \in ?S;\ y \in ?S]\!] \Longrightarrow$ *?lie-bracket x y* $\in ?S$; $\forall x{\in}?S.$ *?lie-bracket x x = 0*; $\forall x{\in}?S.\ \forall y{\in}?S.\ \forall z{\in}?S.$ *0 = ?lie-bracket x (?lie-bracket y z) + ?lie-bracket y (?lie-bracket z x) + ?lie-bracket z (?lie-bracket x y)*$]\!] \Longrightarrow$ *lie-algebra ?S ?scale ?lie-bracket.*

interpretation *cstar1*: *involutive-complex-algebra*
UNIV::*'a*::*cstar set scaleC times 1 involution*
using *cstar-locale* **unfolding** *Cstar-algebra-def* **by** *blast*

interpretation *Cstar-algebra*
UNIV UNIV::*'a*::*cstar set scaleC times norm 1 involution*
using *cstar-locale* **.**

Some simp set adjustments.

declare *cstar1.involution-uminus'*[*simp del*]
declare *cstar1.ivl-minus'*[*simp del*]

context *cstar* **begin**
named-theorems *cstar-simps*
lemmas *cross3-simps*(*10*,*11*,*23*,*24*,*27*,*28*)[*cstar-simps*]
lemmas
cstar1.involution-uminus'[*OF UNIV-I*, *cstar-simps*]
cstar1.ivl-minus'[*OF UNIV-I UNIV-I*, *cstar-simps*]
end

3.4 GNS construction

context
fixes ω :: *'a*::*cstar* $\Rightarrow$ *complex*
assumes *omega-nonneg*: $\bigwedge A$::*'a*. $\omega\ (A^\dagger * A) \geq 0$
and *one* [*simp*]: $\omega\ 1 = 1$
and *clinear* ω
begin

3.4.1 Interpretation of existing locales

thm *linear-def clinear-def linear-on-def*

interpretation ω: *clinear* ω
using ‹*clinear* ω› **.**

lemma *cstar-state*: *Cstar-state UNIV UNIV scaleC times norm 1 involution* ω
apply *unfold-locales*
using ω.*add* ω.*scaleC omega-nonneg*
by (*simp-all add*: *nat-distrib*(*2*) *cross3-simps*(*52*))

interpretation *Cstar-state UNIV UNIV scaleC times norm 1 involution* ω
using *cstar-state* **.**

3.4.2 Some identities for *cstar* and its left ideal.

Show the kernel of our intended inner product is a (left) ideal.

definition *zero-inner-set*: $\mathfrak{I} = \{A::'a.\ \omega\ (A^\dagger * A) = 0\}$

lemma *is-zero-inner*: $\omega\ (A^\dagger * A) = 0$ **if** $A \in \mathfrak{I}$
using *that* **unfolding** *zero-inner-set* **by** *simp*

lemma *zero-inner-zero*: $\omega\ i = 0$ **if** $i \in \mathfrak{I}$
using *state-zero-if-inner-zero that* **unfolding** *zero-inner-set*
by *blast*

lemma *zero-inner-times*: $a*i \in \mathfrak{I}$ **if** $i \in \mathfrak{I}$ **for** $a\ i :: 'a$
proof −
let *?b* = $a^\dagger * a * i$
from *cauchy-schwarz-square*[*of ?b i*]
have $|\omega\ ((a*i)^\dagger * (a*i))|^2 \leq \omega\ (?b^\dagger * ?b) * \omega\ (i^\dagger * i)$
by (*simp add*: *cstar-simps functional-inner-def*)
then show *?thesis*
unfolding *is-zero-inner*[*OF that*] *zero-inner-set* **apply** *simp*
using *cnj-x-x-geq0 cnj-x-x*
by (*metis abs-eq-0-iff le-less less-le-not-le zero-eq-power2*)
qed

lemma *zero-inner-zero2*: $\omega\ (a*i) = 0$ **if** $i \in \mathfrak{I}$
apply (*rule zero-inner-zero*)
apply (*rule zero-inner-times*)
using *that* **.**

end

class *cstar-state* = *cstar* +
fixes *state* :: $'a \Rightarrow$ *complex* (‹ω›)

assumes *omega-nonneg*: $\bigwedge A::'a.\ \omega\ (A^\dagger * A) \geq 0$
and *omega-one* [*simp*]: $\omega\ 1 = 1$
and *omega-linear*[*simp*]: $\omega\ (u + v) = \omega\ u + \omega\ v$
$\omega\ (c *_C u) = c * \omega\ u$
and *extra-assm*[*no-atp*]:
$\langle\omega\ ((B*A)^\dagger * B*A) \leq (norm\ B)^2 * \omega\ (A^\dagger * A)\rangle$

lemma *omega-clinear*: *clinear* ω
apply *unfold-locales* **using** *omega-linear* **by** *auto*

interpretation *omega-linear*: *Vector-Spaces.linear scaleC times* ω
apply *unfold-locales* **using** *omega-linear*(*2*)
by *auto*

— Top-level interpretation of *Cstar-state* for any type in *cstar-state*.
interpretation *cstar-state*: *Cstar-state UNIV UNIV scaleC times norm 1 involution* ω
proof −
interpret *Cstar-algebra UNIV UNIV scaleC times norm 1 involution*
using *cstar-locale* **.**
show *Cstar-state UNIV UNIV* $(*_C)$ $(*)$ *norm 1 involution* ω
apply *unfold-locales*
using *omega-linear*(*2*) *omega-nonneg* **by** (*auto simp*: *cstar-simps*)
qed

lemma *zero-inner-zero'*: $\omega\ A = 0$ **if** $\omega\ (A^\dagger * A) = 0$
using *zero-inner-zero zero-inner-set that*
using *omega-nonneg omega-one omega-clinear*
by (*metis* (*full-types*,*lifting*) *ctr-simps-mem-Collect-eq*)

lemma *zero-inner-zero2'*: $\omega\ (B*A) = 0$ **if** $\omega\ (A^\dagger * A) = 0$
using *zero-inner-zero2 zero-inner-set that*
using *omega-nonneg omega-one omega-clinear*
by (*metis* (*full-types*,*lifting*) *ctr-simps-mem-Collect-eq*)

Some identities, restated for *cstar-state*.

lemma *state-ivl-cnj*: $\omega\ (A^\dagger) = cnj\ (\omega\ A)$
using *positive-normalised-linear-functional.involution-conjugate*
using *cstar-state*[*of* ω, *OF omega-nonneg omega-one omega-clinear*]
unfolding *Cstar-state-def* **by** *blast*

lemma *state-cnj-ivl* [*cstar-simps*]: $cnj\ (\omega\ (A^\dagger)) = \omega\ A$
using *positive-normalised-linear-functional.cnj-ivl*
using *cstar-state*[*of* ω, *OF omega-nonneg omega-one omega-clinear*]
unfolding *Cstar-state-def* **by** *blast*

lemma *state-norm-leq-inner*: $|\omega\ A|^2 \leq \omega\ (A^\dagger * A)$

using *positive-normalised-linear-functional.state-norm-leq-inner*
using *cstar-state*[*of* ω, *OF omega-nonneg omega-one omega-clinear*]
unfolding *Cstar-state-def*
by (*metis UNIV-I complex-of-real-cmod of-real-power*
positive-normalised-linear-functional.functional-inner-def)

lemma *zero-inner-equivp*: *equivp* $(\lambda u\ v.\ \exists A.\ \omega\ (A^\dagger * A) = 0 \wedge u = v + A)$
proof −
have *1*: $\exists B.\ \omega\ (B^\dagger * B) = 0 \wedge A + B = 0$
if $\omega\ (A^\dagger * A) = 0$ **for** $A :: 'a$
apply (*intro exI*[*of* - $-A$])
by (*simp add*: *cstar1.involution-uminus' that*)
have *2*: $\omega\ ((B^\dagger + A^\dagger) * (B + A)) = 0$
if $\omega\ (A^\dagger * A) = 0\ \omega\ (B^\dagger * B) = 0$
for $A\ B :: 'a$
by (*simp add*: *ordered-field-class.sign-simps*(*18*) *that zero-inner-zero2'*)
show *?thesis*
apply (*intro equivpI reflpI sympI transpI*)
by (*auto simp*: *1 2*)
qed

3.4.3 Quotient construction for the inner product

quotient-type (**overloaded**) $'a$ *gns-precomplete* =
$'a$::*cstar-state* / $\lambda u\ v.\ \exists A.\ (\omega\ (A^\dagger * A) = 0 \wedge u = v + A)$
using *zero-inner-equivp* **.**

instantiation *gns-precomplete* :: (*cstar-state*) *scaleC*
begin

lift-definition *scaleC-gns-precomplete* ::
complex ⇒ $'a$ *gns-precomplete* ⇒ $'a$ *gns-precomplete*
is *scaleC*
apply (*simp add*: *cstar-simps*)
by (*metis cross3-simps*(*10*) *nat-distrib*(*2*) *zero-inner-zero2'*)

lift-definition *scaleR-gns-precomplete* ::
real ⇒ $'a$ *gns-precomplete* ⇒ $'a$ *gns-precomplete*
is *scaleR*
proof −
{ fix $r :: real$ **and** $u\ i :: 'a$
assume *asm*: $\omega\ (i^\dagger * i) = 0$
have $\exists i'.\ \omega\ (i'^\dagger * i') = 0 \wedge r *_R (u + i) = r *_R u + i'$
apply (*rule exI*[*of* - *scaleR r i*], *intro conjI*)
apply (*metis scaleC-gns-precomplete.abs-eq asm arithmetic-simps*(*49*)
gns-precomplete.abs-eq-iff pth-4(*2*) *scaleR-scaleC*)
using *cross3-simps*(*30*) **by** *blast* **}**
thus $\exists A.\ \omega\ (A^\dagger * A) = 0 \wedge r *_R a1 = r *_R a2 + A$

if $\exists A.\ \omega\ (A^\dagger * A) = 0 \wedge a1 = a2 + A$
for r **and** $a1\ a2 :: 'a$
using *that* **by** *blast*
qed

instance
by (*intro-classes*) (*simp add*: *scaleC-gns-precomplete-def*
scaleR-gns-precomplete-def scaleR-scaleC)

end

The lifting of the vector space structure is almost entirely a transfer+sledgehammer affair: identities proven earlier are enough to let sledgehammer find respectfulness proofs.

instantiation *gns-precomplete* :: (*cstar-state*) *complex-vector*
begin

lift-definition *zero-gns-precomplete* :: *'a gns-precomplete*
is *0* **.**

lemma $(a+b)* c = a* c + b* (c::'a)$
using *cross3-simps*(*23*) **by** *blast*

lift-definition *minus-gns-precomplete* ::
'a gns-precomplete ⇒ *'a gns-precomplete* ⇒ *'a gns-precomplete*
is *minus*
proof −
{ fix $u\ v :: 'a$
assume *u*: $\omega\ (u^\dagger * u) = 0$ **and** *v*: $\omega\ (v^\dagger * v) = 0$
have $\omega\ ((u^\dagger - v^\dagger) * (u - v)) = \omega\ (u^\dagger * u) - \omega\ (v^\dagger * u) - \omega\ (u^\dagger * v) + \omega\ (v^\dagger * v)$
by (*simp add*: *omega-linear.diff cstar-simps*)
also have $\ldots = 0$
using *zero-inner-zero2'*[*OF u*] *zero-inner-zero2'*[*OF v*] **by** *simp*
finally have $\omega\ ((u^\dagger - v^\dagger) * (u - v)) = 0$ **. }**
thus $\bigwedge a1\ a2\ a3\ a4.$
$\exists A.\ \omega\ (A^\dagger * A) = 0 \wedge a1 = a2 + A \Longrightarrow$
$\exists A.\ \omega\ (A^\dagger * A) = 0 \wedge a3 = a4 + A \Longrightarrow$
$\exists A.\ \omega\ (A^\dagger * A) = 0 \wedge a1 - a3 = a2 - a4 + A$
apply (*auto simp add*: *cstar1.involution-uminus' cstar1.ivl-minus'*)
apply (*metis arith-simps*(*57*) *cross3-simps*(*27*) *omega-linear.diff zero-inner-zero2'*)
done
qed

lift-definition *uminus-gns-precomplete* ::
'a gns-precomplete ⇒ *'a gns-precomplete*
is *uminus* **by** (*auto simp*: *cstar-simps*)

lift-definition *plus-gns-precomplete* ::

'a gns-precomplete ⇒ 'a gns-precomplete ⇒ 'a gns-precomplete
is *plus*
by (*auto simp*: *cstar-simps*) (*metis add-0-iff zero-inner-zero2'*)

instance
apply (*intro-classes*; *transfer*)
by (*auto simp add*: *cstar-simps*)

end

instantiation *gns-precomplete* :: (*cstar-state*) *complex-inner*
begin

The interesting definition is for the inner product, which implies that of the norm.

lift-definition *cinner-gns-precomplete* ::
'a gns-precomplete ⇒ 'a gns-precomplete ⇒ complex
is $\lambda u\ v.\ \omega\ (u^\dagger * v)$
proof *clarify*
fix *u v i1 i2* :: *'a*
assume *i-in-ideal*: $\omega\ (i1^\dagger * i1) = 0\ \omega\ (i2^\dagger * i2) = 0$
have *1*: $\omega\ ((v+i2)^\dagger * (u+i1)) =$
$\omega\ (v^\dagger * u) + \omega\ (v^\dagger * i1) + cnj\ (\omega\ (u^\dagger * i2)) + \omega\ (i2^\dagger * i1)$
apply (*simp add*: *ring-distribs*)
using *involution ivl-times omega-clinear omega-nonneg omega-one*
by (*metis* (*no-types*, *lifting*) *Cstar-state-def UNIV-I cstar-state*
positive-normalised-linear-functional.involution-conjugate)
also have ... $= \omega\ (v^\dagger * u)$
using *i-in-ideal* **by** (*simp add*: *cstar-simps zero-inner-zero2'*)
finally show $\omega\ ((v + i2)^\dagger * (u + i1)) = \omega\ (v^\dagger * u)$
by (*simp add*: *cstar-simps*)
qed

— This could be defined in terms of $\lambda u.\ u \cdot_C u$, which will be a proof obligation later on anyway, but I think this is easier (since we can leverage identities to prove the necessary anyway) in the long term: we keep a concrete construction.
lift-definition *norm-gns-precomplete* :: *'a gns-precomplete ⇒ real*
is $\lambda u.\ Re\ (csqrt\ (\omega\ (u^\dagger * u)))$
by *clarsimp* (*metis* (*no-types*, *lifting*) *cinner-gns-precomplete.abs-eq*
gns-precomplete.abs-eq-iff ivl-plus)

The distance is given by the norm (and vector subtraction), as is standard.

definition *dist-gns-precomplete* ::
'a gns-precomplete ⇒ 'a gns-precomplete ⇒ real
where *dist-gns-precomplete x y = norm* $(y-x)$

The uniform and topological structures are prescribed, in turn, by the metric structure.

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

definition *open-gns-precomplete* :: '*a gns-precomplete set* ⇒ *bool*
where *open-gns-precomplete* U = (∀ x∈U. *eventually* (λ(*x'*, *y*). x' = x ⟶ y ∈ U) *uniformity*)

definition *sgn-gns-precomplete* ::
'*a gns-precomplete* ⇒ '*a gns-precomplete*
where *sgn-gns-precomplete* x ≡ x /$_R$ *norm* x

lemma *norm-sym-gns-precomplete*:
fixes x y :: '*a gns-precomplete*
shows *norm* (y − x) = *norm* (x − y)
proof *transfer*
fix x y :: '*a*
have ω ((y − x)† ∗ (y − x)) = ω ((x − y)† ∗ (x − y))
by (*simp add*: *cstar-simps omega-linear.diff*)
thus *Re* (*csqrt* (ω ((y−x)† ∗ (y−x)))) = *Re* (*csqrt* (ω ((x−y)† ∗ (x−y))))
by *simp*
qed

instance
apply *intro-classes*
subgoal unfolding *dist-gns-precomplete-def* **using** *norm-sym-gns-precomplete* **.**
subgoal unfolding *sgn-gns-precomplete-def* **by** *blast*
subgoal unfolding *uniformity-gns-precomplete-def* **by** *blast*
subgoal unfolding *open-gns-precomplete-def* **by** *blast*
subgoal apply *transfer*
using *state-ivl-cnj* **by** (*metis involution ivl-times*)
subgoal apply *transfer*
by (*simp add*: *cstar-simps*)
subgoal by *transfer* (*simp only*: *ivl-scaleC mult-scaleC-left omega-linear.scale*)
subgoal by *transfer* (*simp add*: *omega-nonneg*)
subgoal by *transfer simp*
subgoal by *transfer* (*simp add*: *cmod-Re omega-nonneg*)
done

end

type-synonym ('*a*) *gns* = '*a gns-precomplete metric-completion*

3.4.4 Representation of '*a* on the Hilbert space '*a gns*.

The action on the dense subspace *range to-gns* is given by lifted multiplication. Notice this isn't quite the dense subspace – this is the action on the

precomplete quotient, a type, not a set on *'a gns*.

lift-definition *action-gns-precomplete* ::
  'a::*cstar-state* ⇒ *'a gns-precomplete* ⇒ *'a gns-precomplete*
 is (∗)
 by (*metis Extra-Ordered-Fields.sign-simps*(*4*) *nat-distrib*(*2*) *zero-inner-zero2'*)

lemma *action-gns-precomplete-alt*: *action-gns-precomplete u* ≡
  λ*v*. *abs-gns-precomplete* (*u* ∗ (*rep-gns-precomplete v*))
 unfolding *action-gns-precomplete-def map-fun-def o-def id-def* **.**

— We can't use lift-definition for the action on the completion, because *from-gns* (below) isn't total (since *from-metric-completion* isn't).

definition *to-gns* :: *'a*::*cstar-state* ⇒ *'a gns*
 where *to-gns* ≡ *to-metric-completion* ∘ *abs-gns-precomplete*

definition *from-gns* :: *'a*::*cstar-state gns* ⇒ *'a*
 where *from-gns* ≡ *rep-gns-precomplete* ∘ *from-metric-completion*

definition *action-gns-precomplete'* ::
  'a::*cstar-state* ⇒ *'a gns-precomplete* ⇒ *'a gns*
 where *action-gns-precomplete' u* ≡
  to-metric-completion ∘ (*action-gns-precomplete u*)

lemma *action-gns-precomplete'-alt*: *action-gns-precomplete'* ≡
  (*id* −−−> *id* −−−> *to-metric-completion*) *action-gns-precomplete*
 unfolding *action-gns-precomplete'-def map-fun-def o-def* **by** *simp*

lemma *action-gns-precomplete'-alt2*: *action-gns-precomplete'* ≡
  (*id* −−−> *rep-gns-precomplete* −−−> (*to-gns*)) (∗)
 unfolding *action-gns-precomplete'-def action-gns-precomplete-def*
  to-gns-def map-fun-def o-def id-def **by** *argo*

3.4.5 The action is bounded.

This is where $\omega\ ((?B * ?A)^\dagger * ?B * ?A) \leq$ *complex-of-real* $((norm\ ?B)^2) * \omega\ (?A^\dagger * ?A)$ is first used.

lemma *state-norm-sandwich-leq*: ‹$|\omega\ (A^\dagger * B * A)| \leq \omega\ (A^\dagger * A) * norm\ B$›
proof−
 from *cstar-state.mult-inner-norm-bound extra-assm*
 show *?thesis*
  unfolding *cstar-state.functional-inner-def*
  by (*metis* (*no-types*, *lifting*) *UNIV-I involution ivl-isometric ivl-times mult.assoc*)
qed

lemma *action-gns-bounded-norm-square*:
 $(norm\ ((action\text{-}gns\text{-}precomplete\ u)\ v))^2 \leq (norm\ u)^2 * (norm\ v)^2$

proof −
  { fix u v :: $'a$
    have $|\omega\ (v^\dagger * u^\dagger * u * v)| \leq \omega\ (v^\dagger * v) * (norm\ (u^\dagger * u))$
      using *state-norm-sandwich-leq*[*of* v $u^\dagger * u$] **by** (*simp add*: *cstar-simps*)
    then have $\|\omega\ ((u * v)^\dagger * (u * v))\| \leq (norm\ u)^2 * \|\omega\ (v^\dagger * v)\|$
      apply (*simp add*: *cstar-simps*)
      using *Bstar*[*simplified*]
    by (*smt* (*verit*) *Extra-Ordered-Fields.sign-simps*(5) *UNIV-I complex-of-real-cmod*
       complex-of-real-mono-iff cstar-state.finner-cmod-eq cstar-state.functional-inner-def
         of-real-mult) **}**
  thus *?thesis*
    unfolding *norm-eq-sqrt-cinner*[*of* (*action-gns-precomplete u*) *v*]
    unfolding *norm-eq-sqrt-cinner*[*of v*]
    apply *transfer*
     by (*simp only*: *abs-norm-cancel real-sqrt-power*[*of - 2*, *symmetric*, *unfolded*
real-sqrt-abs])
qed

lemma *action-gns-bounded-norm-square'*:
  shows $(norm\ ((action\text{-}gns\text{-}precomplete'\ u)\ v))^2 \leq (norm\ u)^2 * (norm\ v)^2$
  using *to-metric-completion-norm*[*of action-gns-precomplete u v*]
  by (*simp add*: *action-gns-precomplete'-def action-gns-bounded-norm-square*)

lemma *action-gns-bounded-norm*:
  shows $(norm\ ((action\text{-}gns\text{-}precomplete\ u)\ v)) \leq (norm\ u) * (norm\ v)$
  using *action-gns-bounded-norm-square norm-ge-zero*
  by (*metis power2-le-imp-le power-mult-distrib zero-compare-simps*(4))

lemma *action-gns-bounded-norm'*:
  shows $(norm\ ((action\text{-}gns\text{-}precomplete'\ u)\ v)) \leq (norm\ u) * (norm\ v)$
  using *to-metric-completion-norm*[*of action-gns-precomplete u v*]
  by (*simp add*: *action-gns-precomplete'-def action-gns-bounded-norm*)

lemma *action-gns-blin*: *bounded-linear* (*action-gns-precomplete u*)
proof −
  interpret *linear action-gns-precomplete u*
    by *unfold-locales* (*transfer*, *simp add*: *cstar-simps*)+
  have $\forall x.\ norm\ (action\text{-}gns\text{-}precomplete\ u\ x) \leq norm\ x * norm\ u$
    by (*simp add*: *action-gns-bounded-norm*[*of u*] *cstar-simps*)
  then show *?thesis*
    by *unfold-locales blast*
qed

lemma *action-gns-blin'*: *bounded-linear* (*action-gns-precomplete'* u)
  unfolding *action-gns-precomplete'-def o-def*
  apply (*rule bounded-linear-compose*[*OF - action-gns-blin*])
  using *to-metric-completion-linear to-metric-completion-norm*
  by (*metis Groups.mult-ac*(2) *bounded-linear.intro*
      bounded-linear-axioms-def more-arith-simps(5) *order.refl*)

lemma *action-gns-unif-cts'*: *uniformly-continuous-on UNIV* (*action-gns-precomplete'* *u*)
using *bounded-linear.isUCont*[*OF action-gns-blin'*] **.**

— Never unfold this. Use the lemma below instead.
definition *action-gns-closure*:: *'a*::*cstar-state* ⇒ *'a gns*⇒*'a gns*
where *action-gns-closure u* ≡ *THE g*.
(*uniformly-continuous-on UNIV g*) ∧
(*action-gns-precomplete' u* = *g* ∘ *to-metric-completion*)

lemma *action-gns-closure*:
uniformly-continuous-on UNIV (*action-gns-closure u*)
action-gns-precomplete' u =
(*action-gns-closure u*) ∘ *to-metric-completion*
⋀*x*. *x*∈*range to-metric-completion* ⟹
(*action-gns-closure u*) *x* = *action-gns-precomplete' u* (*from-metric-completion* *x*)
using *theI'*[*OF lift-to-metric-completion3*[*OF action-gns-unif-cts'*], *of u*]
by (*auto simp add*: *completion-ext-simp1 from-metric-completion-def action-gns-closure-def*)

lemma *action-gns-cblin*: *bounded-clinear* (*action-gns-precomplete u*)
proof −
interpret *clinear action-gns-precomplete u*
apply *unfold-locales*
apply *transfer*
apply (*simp add*: *cstar-simps*; *fail*)
apply *transfer*
using *mult-scaleC-right mult-zero-right omega-linear.zero*
by (*metis nat-arith.rule0*)
have ∀ *x*. *norm* (*action-gns-precomplete u x*) ≤ *norm x* ∗ *norm u*
by (*simp add*: *action-gns-bounded-norm*[*of u*] *cstar-simps*)
then show *?thesis*
by *unfold-locales blast*
qed

lemma *action-gns-cblin'*: *bounded-clinear* (*action-gns-precomplete' u*)
unfolding *action-gns-precomplete'-def o-def*
apply (*rule bounded-clinear-compose*[*OF - action-gns-cblin*])
using *to-metric-completion-clinear to-metric-completion-norm*
by (*metis Groups.mult-ac*(*2*) *bounded-clinear.intro*
bounded-clinear-axioms-def more-arith-simps(*5*) *order.refl*)

— See https://proofwiki.org/wiki/Bounded_Linear_Transformation_to_Banach_Space_has_Unique_Extension_to_Closure_of_Domain and https://en.wikipedia.

org/wiki/Continuous_linear_extension
lemma *linear-continuous-imp-bounded*:
assumes *linear f continuous-on UNIV f*
shows *bounded-linear f*
proof −
interpret *linear f* **using** *assms(1)* **.**
find-theorems *name*: *continuous-on norm*
thm *continuous-on-closure-norm-le continuous-on-compact-bound*
thm *assms(2)*[*unfolded continuous-on-def tendsto-def*]
have *continuous* (*at 0*) *f*
using *assms(2)* **by** (*simp add*: *continuous-on-interior*)
then obtain *d* **where** *d>0* ∀ *x*. *norm* (*x*−*0*) < *d* ⟶ *norm* (*f x* − *f 0*) < *1*
unfolding *dist-norm*[*symmetric*]
using *continuous-at-eps-delta rel-simps(68)* **by** *blast*
hence *d*: *d>0* ⋀*x*. *norm* (*x*) < *d* ⟹ *norm* (*f x*) < *1*
by *auto*
{ fix *x* :: *'a*
let *?dx* = *d* /*R* (*2* ∗ *norm x*)
let *?y* = *?dx* ∗*R x*
have *1*: *norm ?y* < *d*
proof −
have *norm x* ∗ *inverse* (*norm x*) < *2* ∧ *0* ≤ *d*
by (*metis d(1) less-le linorder-not-le mult-zero-left*
numeral-One numeral-less-iff right-inverse semiring-norm(83)
verit-comp-simplify(21) zero-less-numeral)
thus *?thesis*
by (*simp add*: *Groups.mult-ac(2) d(1)*)
qed
have *2*: *norm* (*f ?y*) = *?dx* ∗ *norm* (*f x*)
using *d(1) scaleR* **by** *fastforce*
then have *3*: *?dx* ∗ *norm* (*f x*) < *1*
using *d(2)*[*OF 1*] **by** *presburger*
then have *norm* (*f x*) ≤ (*2* / *d*) ∗ *norm x*
using *norm-ge-zero zero-less-norm-iff d(1)*
proof −
have ∀ *r*::*real*. *1* ∗ *r* = *r*
by *simp*
moreover have ∀ *r ra*. (*r*::*real*) ∗ (*1* / *ra*) = *r* / *ra*
by *auto*
moreover have *norm* (*f x*) < *1* / (*d* / (*norm x* ∗ *2*)) ∨ ¬ *0* < *d* / (*norm x*
∗ *2*)
by (*metis Extra-Ordered-Fields.sign-simps(5) 3 divide-inverse linordered-field-class.pos-less-divide-eq*
real-scaleR-def)
ultimately have *norm* (*f x*) ≤ *norm x* ∗ (*2* / *d*)
by (*metis* (*no-types*) *arith-extra-simps(18) d(1) divide-divide-eq-right less-le*
linordered-field-class.pos-less-divide-eq local.zero norm-ge-zero zero-less-norm-iff
zero-less-numeral)
then show *?thesis*
by (*simp add*: *Extra-Ordered-Fields.sign-simps(5)*)

```
    qed
  } then
  show ?thesis
    by unfold-locales (metis Extra-Ordered-Fields.sign-simps(5))
qed

lemma linear-continuous-iff-bounded:
  assumes linear f
  shows continuous-on UNIV f ⟷ bounded-linear f
  using assms linear-continuous-imp-bounded linear-continuous-on
  by blast

lemma clinear-continuous-iff-bounded:
  assumes clinear f
  shows continuous-on UNIV f ⟷ bounded-clinear f
  using assms linear-continuous-iff-bounded
  by (metis bounded-clinear.bounded-linear
      bounded-linear-bounded-clinear clinear.linear clinear.scaleC)

lemma linear-UCont-iff-bounded:
  assumes linear f
  shows uniformly-continuous-on UNIV f ⟷ bounded-linear f
  using linear-continuous-iff-bounded[OF assms]
  using bounded-linear.isUCont uniformly-continuous-imp-continuous by blast

lemma clinear-UCont-iff-bounded:
  assumes clinear f
  shows uniformly-continuous-on UNIV f ⟷ bounded-clinear f
  using clinear-continuous-iff-bounded[OF assms]
    bounded-linear.isUCont[OF bounded-clinear.bounded-linear]
    uniformly-continuous-imp-continuous by blast

definition action-cblinfun ::
    'a::cstar-state ⇒ ('a gns, 'a gns) cblinfun (‹$\pi_\omega$›)
  where action-cblinfun u ≡ cblinfun-extension
    (range to-metric-completion)
    (map-fun from-metric-completion id (action-gns-precomplete' u))

lemma action-cblinfun: cblinfun-extension-exists
    (range to-metric-completion)
    (map-fun from-metric-completion id (action-gns-precomplete' u))
  for u :: 'a::cstar-state
proof −
  interpret bounded-clinear action-gns-precomplete' u
    using action-gns-cblin' .
  obtain K where K: ⋀x. norm (action-gns-precomplete' u x) ≤ norm x * K
    using real.bounded by blast
  show ?thesis
   by (meson action-gns-bounded-norm' real.add scaleC to-metric-completion-cblinfun-exists)
```

qed

lemma *action-cblinfun-apply*:
shows *action-cblinfun u* (*to-metric-completion x*) = *action-gns-precomplete′ u x*
using *cblinfun-extension-apply*[*OF action-cblinfun*]
using *the-inv-f-f*[*OF inj-to-metric-completion*]
unfolding *action-cblinfun-def from-metric-completion-def*
by (*smt* (*verit*) *eq-id-iff map-fun-apply rangeI*)

lemma *action-cblinfun-cong*:
assumes *cspan* (*range to-metric-completion*) = *cspan* (*to-metric-completion ‘ B′*)
⋀*x*. *x* ∈ *B′* ⟹ *action-cblinfun u* (*to-metric-completion x*) = *g* (*to-metric-completion x*)
shows *cblinfun-extension* (*to-metric-completion ‘ B′*) *g* = *action-cblinfun u*
using *assms*
using *cblinfun-extension-cong*[*OF - - - action-cblinfun*]
using *the-inv-f-f*[*OF inj-to-metric-completion*]
by (*smt* (*verit*, *ccfv-threshold*) *action-cblinfun-apply action-cblinfun-def eq-id-iff from-metric-completion-def image-iff image-mono map-fun-apply subset-UNIV*)

lemmas *action-cblinfun-apply′* = *cblinfun-extension-apply*[*OF action-cblinfun*, *folded action-cblinfun-def*]
and *action-cblinfun-cong′* = *cblinfun-extension-cong*[*OF - - - action-cblinfun*, *folded action-cblinfun-def*]
and *action-cblinfun-restrict′* = *cblinfun-extension-exists-restrict*[*OF - - action-cblinfun*, *folded action-cblinfun-def*]

lemma *action-UCont*: *isUCont* (*action-cblinfun u*)
using *clinear-UCont-iff-bounded bounded-cbilinear-apply-bounded-clinear bounded-cbilinear-cblinfun-apply cblinfun-apply-clinear* **by** *blast*

lemma *action-closure-cblinfun*:
action-gns-closure u = (*action-cblinfun u*)
proof−
have *action-cblinfun-to-completion*:
action-gns-precomplete′ u = *action-cblinfun u* ∘ *to-metric-completion*
x ∈ *range to-metric-completion* ⟹
action-cblinfun u x = *action-gns-precomplete′ u* (*from-metric-completion x*)
for *x*
by (*simp-all add*: *action-cblinfun-apply′ action-cblinfun-apply o-def*)
show *?thesis*
using *action-UCont action-cblinfun-to-completion*(*1*) *action-gns-closure*(*1*,*2*) *action-gns-unif-cts′*
lift-to-metric-completion4 **by** *blast*
qed

The cyclic vector is the abstraction of the unit

definition *gns-1* (‹Ω›)
where *gns-1* ≡ *to-gns 1*

lemma *gns-1'*: $\Omega \cdot_C (\textit{action-gns-closure}\ u\ \Omega) = \omega\ u$
proof −
  have (*abs-gns-precomplete 1*) $\cdot_C$ *action-gns-precomplete u* (*abs-gns-precomplete 1*) $= \omega\ u$
    by *transfer* (*simp add*: *cstar1.unit-self-adjoint*)
  then have *to-metric-completion* (*abs-gns-precomplete 1*) $\cdot_C$
      action-gns-precomplete' u (*abs-gns-precomplete 1*) =
      $\omega\ u$
    unfolding *action-gns-precomplete'-def*
    by (*simp add*: *to-metric-completion-cinner*)
  thus *?thesis*
    using *action-gns-closure*(*2*) **by** (*metis comp-def gns-1-def to-gns-def*)
qed

— Basically an Isabelle phrasing of Bratteli and Robinson's last sentence on page 55 [1].
lemma *cyclic-set*:
  shows $\{\pi_\omega\ u\ \Omega \mid u.\ \textit{True}\} = \{\textit{to-gns}\ u \mid u.\ \textit{True}\}$
    and $\{\textit{action-gns-closure}\ u\ \Omega \mid u.\ \textit{True}\} = \{\textit{to-gns}\ u \mid u.\ \textit{True}\}$
proof −
  have $\exists u.\ \textit{action-gns-closure}\ v\ \Omega = \textit{to-gns}\ u$ **for** $v :: 'c{::}\textit{cstar-state}$
    unfolding *gns-1-def to-gns-def*
   by (*metis action-gns-closure*(*2*) *action-gns-precomplete'-def action-gns-precomplete-alt o-def*)
  moreover have $\exists v.\ \textit{to-gns}\ u = \textit{action-gns-closure}\ v\ \Omega$ **for** $u :: 'c{::}\textit{cstar-state}$
    unfolding *gns-1-def to-gns-def*
    by (*metis UNIV-I action-gns-closure*(*2*) *action-gns-precomplete'-def*
      action-gns-precomplete.abs-eq comp-eq-dest-lhs cstar1.multiplicative.right-unit)
  ultimately show $\{\textit{action-gns-closure}\ u\ \Omega \mid u.\ \textit{True}\} = \{\textit{to-gns}\ u \mid u.\ \textit{True}\}$
    by (*auto simp add*: *gns-1-def to-gns-def*)
  thus $\{\pi_\omega\ u\ \Omega \mid u.\ \textit{True}\} = \{\textit{to-gns}\ u \mid u.\ \textit{True}\}$
    by (*simp add*: *action-closure-cblinfun*)
qed

lemma *gns-cyclic'*: *Elementary-Topology.closure* $\{\textit{action-gns-closure}\ u\ \Omega \mid u :: 'a{::}\textit{cstar-state}.\ \textit{True}\} = \textit{UNIV}$
proof −
  have $\exists u.$ *to-metric-completion* $x =$ *to-metric-completion* (*abs-gns-precomplete u*)
    for $x :: 'b{::}\textit{cstar-state gns-precomplete}$
    by (*metis add.inverse-inverse map-fun-apply uminus-gns-precomplete-def*)
  hence $\{\textit{to-gns}\ u \mid u.\ \textit{True}\} = \textit{range to-metric-completion}$
    by (*auto simp add*: *to-gns-def*)
  thus *?thesis*
    using *cyclic-set*(*2*) *to-metric-completion-dense'*
    by (*metis* (*lifting*) *ext*)
qed

3.4.6 The action is a homomorphism

hide-const (**open**) *closure*

Following *Smooth-Manifolds.Analysis-More* to instantiate the vector space on functions.

instantiation *fun* :: (*type*,*scaleC*) *scaleC*
begin
definition *scaleC-fun* :: *complex* ⇒ (*'a* ⇒ *'b*) ⇒ *'a* ⇒ *'b*
where *scaleC-fun r f* = (λ*x*. *r* ∗$_C$ *f x*)
instance
apply *intro-classes* **unfolding** *scaleC-fun-def scaleR-fun-def*
by (*simp add*: *scaleR-scaleC*)
end

instance *fun* :: (*type*, *complex-vector*) *complex-vector*
by *standard* (*auto simp*: *scaleC-fun-def algebra-simps*)

— The action is (by design) not injective. The norm is defined through the inner product, not as a lifting of the norm – it may not be preserved exactly.
lemma *action-injective-on-quotient*: ∃ *I*. ω (*I*† ∗ *I*) = *0* ∧ *a* = *b* + *I*
if ∀ *x*. *action-gns-precomplete a x* = *action-gns-precomplete b x*
for *a b* :: *'a*::*cstar-state* **and** *x* :: *'a gns-precomplete*
using *that* **apply** *transfer*
by (*metis verit-prod-simplify*(*2*))

— We already know this action outputs *bounded-clinear* maps. We don't yet know it is linear itself.
lemma *action-precomplete-morphism*:
shows *action-gns-precomplete* (*a*+*b*) = *action-gns-precomplete a* + *action-gns-precomplete b*
and *action-gns-precomplete* (*a*∗*b*) = *action-gns-precomplete a* ∘ *action-gns-precomplete b*
and *action-gns-precomplete* (*c*∗$_C$ *a*) = *c* ∗$_C$ (*action-gns-precomplete a*)
and *action-gns-precomplete 1* = *id*
and *action-gns-precomplete* (*involution a*) = *cadjoint* (*action-gns-precomplete a*)
proof −
let *?pi* = *action-gns-precomplete*
show *?pi* (*a*+*b*) = *?pi a* + *?pi b*
unfolding *plus-fun-def* **apply** (*intro ext*) **apply** *transfer*
by (*simp add*: *cross3-simps*(*23*))
show *?pi* (*a*∗*b*) = *?pi a* ∘ *?pi b*
unfolding *o-def* **apply** (*intro ext*) **apply** *transfer*
by (*simp add*: *ordered-field-class.sign-simps*(*4*))
show *?pi* (*c*∗$_C$ *a*) = *c* ∗$_C$ (*?pi a*)
unfolding *o-def scaleC-fun-def* **apply** (*intro ext*) **apply** *transfer*
by (*metis arith-simps*(*50*) *mult-scaleC-left mult-zero-right omega-linear.zero*)
show *?pi 1* = *id*

```
    unfolding id-def apply (intro ext)
    by transfer fastforce
  show ?pi (a†) = cadjoint (?pi a)
  proof −
    have (?pi (a†) x) ·C y = x ·C (?pi a y) for x y
      apply transfer by (simp add: Groups.mult-ac(1))
    thus ?thesis
      by (metis cadjoint-eqI)
  qed
qed

lemma action-precomplete-morphism':
 shows action-gns-precomplete' (a+b) = action-gns-precomplete' a + action-gns-precomplete'
b
    and action-gns-precomplete' (c∗C a) = c ∗C (action-gns-precomplete' a)
    and action-gns-precomplete' 1 = to-metric-completion
    and action-gns-precomplete' (a∗b) = action-gns-precomplete' a ∘
        from-metric-completion ∘ action-gns-precomplete' b
    and action-gns-precomplete' (a†) = to-metric-completion ∘
      cadjoint (from-metric-completion ∘ action-gns-precomplete' a)
proof −
  let ?pi = action-gns-precomplete'
  show ?pi 1 = to-metric-completion
    unfolding action-gns-precomplete'-def o-def
    by (simp add: action-precomplete-morphism(4))

  interpret clinear to-metric-completion
    using to-metric-completion-clinear .
  show ?pi (a+b) = ?pi a + ?pi b
    unfolding action-gns-precomplete'-def o-def
    by (auto simp add: action-precomplete-morphism(1) add)
  show ?pi (c∗C a) = c ∗C (?pi a)
    unfolding action-gns-precomplete'-def o-def
    by (simp only: action-precomplete-morphism(3) scaleC-fun-def scaleC)
  show ?pi (a∗b) = (?pi a) ∘ from-metric-completion ∘ (?pi b)
    unfolding action-gns-precomplete'-def
     by (metis (mono-tags) action-precomplete-morphism(2) completion-ext-simp1
o-apply)
   show ?pi (a†) = to-metric-completion ∘ (cadjoint (from-metric-completion ∘
action-gns-precomplete' a))
    unfolding action-gns-precomplete'-def action-precomplete-morphism(5) o-def
    by (simp add: from-metric-completion-def inj-to-metric-completion the-inv-f-f)
qed

method action-uniq-eqI =
  (unfold action-gns-closure-def,
   intro the1-equality[OF lift-to-metric-completion4[OF action-gns-unif-cts']] conjI,
    fold action-gns-closure-def)
```

lemma
shows *action-closure-morphism*:
action-gns-closure $(a{+}b)$ = *action-gns-closure* a + *action-gns-closure* b
action-gns-closure $(a{*}b)$ = *comp* (*action-gns-closure* a) (*action-gns-closure* b)
action-gns-closure $(c{*_C}a)$ = c $*_C$ (*action-gns-closure* a)
action-gns-closure 1 = *id-cblinfun*
action-gns-closure (*involution* a) = *cadjoint* (*action-gns-closure* a)
and *action-cblinfun-morphism*:
π_ω $(a{+}b)$ = π_ω a + π_ω b
π_ω $(a{*}b)$ = *cblinfun-compose* (π_ω a) (π_ω b)
π_ω $(c{*_C}a)$ = c $*_C$ (π_ω a)
π_ω 1 = *id-cblinfun*
π_ω (*involution* a) = *adj* (π_ω a)
proof −
show *action-gns-closure* $(a{+}b)$ = *action-gns-closure* a + *action-gns-closure* b
proof *action-uniq-eqI*
show *isUCont* (*action-gns-closure* a + *action-gns-closure* b)
unfolding *plus-fun-def*
using *action-gns-closure*(1) *uniformly-continuous-on-add* **by** *blast*
show *action-gns-precomplete'* $(a + b)$ =
(*action-gns-closure* a + *action-gns-closure* b) ∘ *to-metric-completion*
unfolding *plus-compose*
using *action-gns-closure*(2) *action-precomplete-morphism'*(1)
by *metis*
qed
thus π_ω $(a{+}b)$ = π_ω a + π_ω b
unfolding *action-closure-cblinfun*
by (*intro cblinfun-eqI*) (*simp add*: *cblinfun.add-left*)

show *action-gns-closure* $(a{*}b)$ = *action-gns-closure* a ∘ *action-gns-closure* b
proof *action-uniq-eqI*
show *isUCont* (*action-gns-closure* a ∘ *action-gns-closure* b)
unfolding *action-closure-cblinfun o-def*
apply (*rule bounded-linear.isUCont*)
apply (*rule bounded-linear-compose*[*of cblinfun-apply* (π_ω a)])
by (*simp-all add*: *cblinfun.real.bounded-linear-right*)
show *action-gns-precomplete'* $(a * b)$ =
(*action-gns-closure* a ∘ *action-gns-closure* b) ∘ *to-metric-completion*
using *action-gns-closure*(2) *action-precomplete-morphism*(2)
by (*metis* (*no-types*, *lifting*) *action-gns-precomplete'-def fun.map-comp*)
qed
thus π_ω $(a{*}b)$ = *cblinfun-compose* (π_ω a) (π_ω b)
unfolding *action-closure-cblinfun*
by (*intro cblinfun-eqI*) (*simp*)

show *action-gns-closure* $(c{*_C}a)$ = c $*_C$ (*action-gns-closure* a)
proof *action-uniq-eqI*
show *isUCont* (c $*_C$ (*action-gns-closure* a))
unfolding *action-closure-cblinfun scaleC-fun-def*

by (*auto intro*: *bounded-linear.isUCont simp add*: *bounded-clinear.bounded-linear*

bounded-clinear-scaleC-right bounded-linear-compose cblinfun.real.bounded-linear-right)
show *action-gns-precomplete'* (*c*∗$_C$ *a*) =
(*c* ∗$_C$ (*action-gns-closure a*)) ∘ *to-metric-completion*
using *action-gns-closure*(*2*) *action-precomplete-morphism'*(*2*)
unfolding *o-def scaleC-fun-def* **by** *metis*
qed
thus π_ω (*c*∗$_C$ *a*) = *c* ∗$_C$ (π_ω *a*)
unfolding *action-closure-cblinfun*
by (*intro cblinfun-eqI*) (*simp add*: *scaleC-fun-def*)

show *action-gns-closure 1* = *id-cblinfun*
by (*metis action-gns-closure*(*1,2*) *action-gns-unif-cts' action-precomplete-morphism'*(*3*)
apply-id-cblinfun
bounded-linear.isUCont cblinfun.real.bounded-linear-right fun.map-id lift-to-metric-completion4)
thus π_ω *1* = *id-cblinfun*
by (*smt* (*verit*, *best*) *action-closure-cblinfun cblinfun-eqI*)

show *action-gns-closure* ($a^\dagger$) = *cadjoint* (*action-gns-closure a*)
proof *action-uniq-eqI*
have *cblin-action*: *bounded-clinear* (*action-gns-closure a*)
unfolding *action-closure-cblinfun*
by (*metis* (*no-types*) *bounded-clinear-intro cblinfun.add-right*
cblinfun.scaleC-right norm-cblinfun ordered-field-class.sign-simps(*5*))

show *isUCont* (*cadjoint* (*action-gns-closure a*))
by (*auto intro*!: *bounded-linear.isUCont bounded-clinear.bounded-linear*
simp add: *cblin-action cadjoint-bounded-clinear*)
have *to-metric-completion-cadjoint*:
cadjoint (*action-gns-closure a*) (*to-metric-completion x*) ·$_C$
(*to-metric-completion y*) =
cadjoint (*action-gns-precomplete a*) *x* ·$_C$ *y*
for *x y* :: *'a gns-precomplete*
proof −
let *?x* = *to-metric-completion x*
let *?y* = *to-metric-completion y*
have *cadjoint* (*action-gns-closure a*) *?x* ·$_C$ *?y* = *?x* ·$_C$ ((*action-gns-closure a*)
?y)
using *cadjoint-univ-prop*[*OF cblin-action*] **by** *presburger*
also have ... = *?x* ·$_C$ (*to-metric-completion* ((*action-gns-precomplete a*) *y*))
using *action-gns-closure*(*2*)[*unfolded action-gns-precomplete'-def o-def*]
by *metis*
also have ... = *x* ·$_C$ ((*action-gns-precomplete a*) *y*)
by (*rule to-metric-completion-cinner*)
also have ... = (*cadjoint* (*action-gns-precomplete a*) *x*) ·$_C$ *y*
using *cadjoint-univ-prop*[*OF cblin-action*]
by (*metis* (*no-types*, *lifting*) *Groups.mult-ac*(*1*) *action-gns-precomplete'-def*
action-gns-precomplete-alt

action-precomplete-morphism'(*3*) *action-precomplete-morphism*(*5*) *cinner-gns-precomplete.abs-eq involution*
ivl-times more-arith-simps(*5*) *o-def to-metric-completion-cinner*)
finally show *cadjoint* (*action-gns-closure a*) *?x* $\cdot_C$ *?y* =
cadjoint (*action-gns-precomplete a*) *x* $\cdot_C$ *y*
by −
qed

show *action-gns-precomplete'* ($a^\dagger$) =
cadjoint (*action-gns-closure a*) ∘ *to-metric-completion*
unfolding *action-gns-closure*(*2*)
unfolding *o-def* **apply** (*intro ext*)
apply (*intro cinner-extensionality-metric-completion allI*)
by (*metis* (*mono-tags*, *opaque-lifting*) *action-cblinfun-apply action-closure-cblinfun action-gns-precomplete'-def*
action-precomplete-morphism(*5*) *cinner-eq-flip o-def to-metric-completion-cadjoint to-metric-completion-cinner*)
qed
thus π_ω (*involution a*) = *adj* (π_ω *a*)
unfolding *action-closure-cblinfun*
by (*intro cblinfun-eqI*) (*simp add*: *adj.rep-eq*)
qed

3.5 GNS theorem for π_ω

It's a little sparser than in textbooks: the existence of the Hilbert space and a representation in terms of bounded operators is implicit in the construction of *'a gns* and π_ω.

theorem *GNS-construction*:
shows *gns-1*: ∀ *u*. Ω $\cdot_C$ (π_ω *u* Ω) = ω *u*
and *gns-cyclic*: *closure* {π_ω *u* Ω | *u*. *True*} = *UNIV*
and *gns-hom*: π_ω (*a*+*b*) = π_ω *a* + π_ω *b*
π_ω (*a*∗*b*) = *cblinfun-compose* (π_ω *a*) (π_ω *b*)
π_ω (*c*∗$_C$*a*) = *c* ∗$_C$ (π_ω *a*)
π_ω *1* = *id-cblinfun*
π_ω (*involution a*) = *adj* (π_ω *a*)
proof −
show ∀ *u*. Ω $\cdot_C$ *cblinfun-apply* (π_ω *u*) Ω = ω *u*
using *gns-1'*[*unfolded action-closure-cblinfun*] **by** *blast*
show *closure* {*cblinfun-apply* (π_ω *u*) Ω |*u*. *True*} = *UNIV*
using *gns-cyclic'*[*unfolded action-closure-cblinfun*] .
qed (*use action-cblinfun-morphism* **in** *simp-all*)

unbundle *cblinfun-syntax*
no-notation *gns-1* (‹Ω›)
no-notation *action-cblinfun* (‹π_ω›)

For presentation only, we give the existential form of the GNS construc-

tion. There's no disadvantage to using explicit constructions and $\forall\, u.\ gns\text{-}1\ \cdot_C\ (action\text{-}cblinfun\ u\ *_V\ gns\text{-}1) = \omega\ u$
closure $\{action\text{-}cblinfun\ u\ *_V\ gns\text{-}1\ |u.\ True\} = UNIV$
action-cblinfun $(?a + ?b) = action\text{-}cblinfun\ ?a + action\text{-}cblinfun\ ?b$
action-cblinfun $(?a * ?b) = action\text{-}cblinfun\ ?a\ o_{CL}\ action\text{-}cblinfun\ ?b$
action-cblinfun $(?c *_C ?a) = ?c *_C action\text{-}cblinfun\ ?a$
action-cblinfun 1 $= id\text{-}cblinfun$
action-cblinfun $(?a^\dagger) = action\text{-}cblinfun\ ?a*$.

theorem *GNS-construction'*:
obtains $\pi_\omega :: 'a::cstar\text{-}state \Rightarrow ('a\ gns \Rightarrow_{CL} 'a\ gns)$
and $\Omega :: 'a\ gns$
where — properties of Ω
$\forall\, u.\ \Omega\ \cdot_C\ (\pi_\omega\ u\ \Omega) = \omega\ u$
and *closure* $\{\pi_\omega\ u\ \Omega \mid u.\ True\} = UNIV$
— π is a *-homomorphism
and $\pi_\omega\ (a+b) = \pi_\omega\ a + \pi_\omega\ b$
and $\pi_\omega\ (a*b) = cblinfun\text{-}compose\ (\pi_\omega\ a)\ (\pi_\omega\ b)$
and $\pi_\omega\ (c*_C a) = c *_C (\pi_\omega\ a)$
and $\pi_\omega\ 1 = id\text{-}cblinfun$
and $\pi_\omega\ (involution\ a) = adj\ (\pi_\omega\ a)$
using *GNS-construction* **by** *blast*
unbundle *no cblinfun-syntax*
notation *gns-1* $(\langle\Omega\rangle)$
notation *action-cblinfun* $(\langle\pi_\omega\rangle)$

unbundle *no lifting-syntax*
unbundle *no cstar-syntax*
unbundle *scaleC-syntax*

end

References

[1] O. Bratteli and D. W. Robinson. *Operator algebras and quantum statistical mechanics.* Texts and monographs in physics. Springer-Verlag, New York, 1979.

[2] C. J. Fewster and K. Rejzner. Algebraic Quantum Field Theory – an introduction.