

Tensor Products in Hilbert Spaces

Dominique Unruh

March 17, 2025

Abstract

We formalize the tensor product of Hilbert spaces, and related material. Specifically, we define the product of vectors in Hilbert spaces, of operators on Hilbert spaces, and of subspaces of Hilbert spaces, and of von Neumann algebras, and study their properties.

The theory is based on the AFP entry `Complex_Bounded_Operators` that introduces Hilbert spaces and operators and related concepts, but in addition to their work, we defined and study a number of additional concepts needed for the tensor product.

Specifically: Hilbert-Schmidt and trace-class operators; compact operators; positive operators; the weak operator, strong operator, and weak* topology; the spectral theorem for compact operators; and the double commutant theorem.

Contents

1	<i>Misc-Tensor-Product</i> – Miscellaneous results missing from other theories	3
2	<i>Misc-Tensor-Product-BO</i> – Miscellaneous results missing from <code>Complex_Bounded_Operators</code>	55
3	<i>Strong-Operator-Topology</i> – Strong operator topology on complex bounded operators	69
4	<i>Positive-Operators</i> – Positive bounded operators	80
5	<i>HS2Ell2</i> – Representing any Hilbert space as $\ell_2(X)$	111
6	<i>Weak-Operator-Topology</i> – Weak operator topology on complex bounded operators	116
7	<i>Misc-Tensor-Product-TTS</i> – Miscellaneous results missing from <code>Complex_Bounded_Operators</code>	138
7.1	Retrieving axioms	138
7.2	Auxiliary lemmas	138
7.3	<i>plus</i>	141
7.3.1	<i>minus</i>	142
7.3.2	<i>uminus</i>	142

7.4	<i>semigroup</i>	142
7.5	<i>abel-semigroup</i>	143
7.6	<i>comm-monoid</i>	143
7.7	<i>topological-space</i>	144
7.8	<i>sum</i>	144
7.9	<i>t2-space</i>	145
7.9.1	<i>continuous-on</i>	145
7.10	<i>scaleR</i>	146
7.11	<i>scaleC</i>	146
7.12	<i>ab-group-add</i>	147
7.13	<i>vector-space</i>	147
7.14	<i>complex-vector</i>	148
7.15	<i>open-uniformity</i>	148
7.16	<i>uniformity-dist</i>	150
7.17	<i>sgn</i>	150
7.18	<i>sgn-div-norm</i>	151
7.19	<i>dist-norm</i>	151
7.20	<i>complex-inner</i>	152
7.21	<i>is-ortho-set</i>	153
7.22	<i>metric-space</i>	153
7.23	<i>nhds</i>	154
7.24	<i>at-within</i>	154
7.25	<i>(has-sum)</i>	155
7.26	<i>filterlim</i>	155
7.27	<i>convergent</i>	155
7.28	<i>uniform-space.cauchy-filter</i>	156
7.29	<i>uniform-space.Cauchy</i>	156
7.30	<i>complete-space</i>	156
7.31	<i>chilbert-space</i>	157
7.32	<i>(hull)</i>	158
7.33	<i>csubspace</i>	159
7.34	<i>cspan</i>	159
7.34.1	<i>(islimpt)</i>	160
7.34.2	<i>closure</i>	160
7.35	<i>continuous</i>	161
7.36	<i>is-onb</i>	161
7.37	Transferring theorems	162
8	Stuff relying on the above lifting	167
9	<i>Eigenvalues</i> – Material related to eigenvalues and eigenspaces	171

10	<i>Compact-Operators</i> – Finite rank and compact operators	184
10.1	Finite rank operators	184
10.2	Compact operators	189
11	<i>Spectral-Theorem</i> – The spectral theorem for compact operators	216
11.1	Spectral decomp, compact op	216
12	<i>Trace-Class</i> – Trace-class operators	231
12.1	Auxiliary lemmas	231
12.2	Trace-norm and trace-class	233
12.3	Hilbert-Schmidt operators	238
12.4	Trace-norm and trace-class, continued	252
12.5	More Hilbert-Schmidt	305
12.6	Spectral Theorem	306
12.7	More Trace-Class	313
13	<i>Weak-Star-Topology</i> – Weak* topology on complex bounded operators	319
14	<i>Hilbert-Space-Tensor-Product</i> – Tensor product of Hilbert Spaces	336
14.1	Tensor product on ℓ_2	336
14.2	Tensor product of operators on ℓ_2	348
14.3	Tensor product of subspaces	376
15	<i>Partial-Trace</i> – The partial trace	390
16	<i>Von-Neumann-Algebras</i> – Von Neumann algebras and the double commutant theorem	398
16.1	Commutants	398
16.2	Double commutant theorem	408
16.3	Von Neumann Algebras	421
17	<i>Tensor-Product-Code</i> – Support for code generation	427
1	<i>Misc-Tensor-Product</i> – Miscellaneous results missing from other theories	

theory *Misc-Tensor-Product*

imports *HOL-Analysis.Elementary-Topology* *HOL-Analysis.Abstract-Topology*
HOL-Analysis.Abstract-Limits *HOL-Analysis.Function-Topology* *HOL-Cardinals.Cardinals*
HOL-Analysis.Infinite-Sum *HOL-Analysis.Harmonic-Numbers* *Containers.Containers-Auxiliary*
Complex-Bounded-Operators.Extra-General
Complex-Bounded-Operators.Extra-Vector-Spaces
Complex-Bounded-Operators.Extra-Ordered-Fields

begin

unbundle *lattice-syntax*

lemma *local-defE*: $(\bigwedge x. x=y \implies P) \implies P$ **by** *metis*

— A helper lemma to introduce a local “definition“ in the current goal when backwards reasoning. *apply* (*rule local-defE*[*where* $x=\langle stuff \rangle$]) will insert $x = stuff$ as a premise. This can be useful before using *apply transfer* because it will introduce some additional knowledge about the properties of x into the transferred goal.

lemma *inv-prod-swap*[*simp*]: $\langle inv\ prod.swap = prod.swap \rangle$
by (*simp add: inv-unique-comp*)

lemma *filterlim-parametric*[*transfer-rule*]:

includes *lifting-syntax*

assumes [*transfer-rule*]: $\langle bi-unique\ S \rangle$

shows $\langle (R \implies S) \implies rel-filter\ S \implies rel-filter\ R \implies (=) filterlim\ filterlim \rangle$

using *filtermap-parametric*[*transfer-rule*] *le-filter-parametric*[*transfer-rule*] **apply** *fail?*

unfolding *filterlim-def*

by *transfer-prover*

definition *rel-topology* :: $\langle ('a \implies 'b \implies bool) \implies ('a\ topology \implies 'b\ topology \implies bool) \rangle$ **where**
 $\langle rel-topology\ R\ S\ T \iff (rel-fun\ (rel-set\ R)\ (=))\ (openin\ S)\ (openin\ T) \wedge (\forall U. openin\ S\ U \implies Domainp\ (rel-set\ R)\ U) \wedge (\forall U. openin\ T\ U \implies Rangep\ (rel-set\ R)\ U) \rangle$

lemma *rel-topology-eq*[*relator-eq*]: $\langle rel-topology\ (=) = (=) \rangle$
unfolding *rel-topology-def* **using** *openin-inject*
by (*auto simp: rel-fun-eq rel-set-eq fun-eq-iff*)

lemma *Rangep-conversep*[*simp*]: $\langle Rangep\ (R^{-1-1}) = Domainp\ R \rangle$
by *blast*

lemma *Domainp-conversep*[*simp*]: $\langle Domainp\ (R^{-1-1}) = Rangep\ R \rangle$
by *blast*

lemma *conversep-rel-fun*:

includes *lifting-syntax*

shows $\langle (T \implies U)^{-1-1} = (T^{-1-1}) \implies (U^{-1-1}) \rangle$

by (*auto simp: rel-fun-def*)

lemma *rel-topology-conversep*[*simp*]: $\langle rel-topology\ (R^{-1-1}) = ((rel-topology\ R)^{-1-1}) \rangle$
by (*auto simp add: rel-topology-def*[*abs-def*] *simp flip: conversep-rel-fun*[**where** $U=\langle (=) \rangle$, *simplified*])

lemma *openin-parametric*[*transfer-rule*]:

includes *lifting-syntax*

shows $\langle (rel-topology\ R \implies rel-set\ R \implies (=))\ openin\ openin \rangle$

by (*auto simp add: rel-fun-def rel-topology-def*)

```

lemma topspace-parametric [transfer-rule]:
  includes lifting-syntax
  shows  $\langle \text{rel-topology } R \implies \text{rel-set } R \rangle \text{ topspace } \text{topspace}$ 
proof -
  have *:  $\langle \exists y \in \text{topspace } T'. R \ x \ y \rangle$  if  $\langle \text{rel-topology } R \ T \ T' \rangle \langle x \in \text{topspace } T \rangle$  for  $x \ T \ T'$  and
   $R :: \langle 'q \Rightarrow 'r \Rightarrow \text{bool} \rangle$ 
proof -
  from that obtain  $U$  where  $\langle \text{openin } T \ U \rangle$  and  $\langle x \in U \rangle$ 
    unfolding topspace-def
    by auto
  from  $\langle \text{openin } T \ U \rangle$ 
  have  $\langle \text{Domainp } (\text{rel-set } R) \ U \rangle$ 
    using  $\langle \text{rel-topology } R \ T \ T' \rangle$  rel-topology-def by blast
  then obtain  $V$  where [transfer-rule]:  $\langle \text{rel-set } R \ U \ V \rangle$ 
    by blast
  with  $\langle x \in U \rangle$  obtain  $y$  where  $\langle R \ x \ y \rangle$  and  $\langle y \in V \rangle$ 
    by (meson rel-set-def)
  from  $\langle \text{rel-set } R \ U \ V \rangle \langle \text{rel-topology } R \ T \ T' \rangle \langle \text{openin } T \ U \rangle$ 
  have  $\langle \text{openin } T' \ V \rangle$ 
    by (simp add: rel-topology-def rel-fun-def)
  with  $\langle y \in V \rangle$  have  $\langle y \in \text{topspace } T' \rangle$ 
    using openin-subset by auto
  with  $\langle R \ x \ y \rangle$  show  $\langle \exists y \in \text{topspace } T'. R \ x \ y \rangle$ 
    by auto
qed

```

```

show ?thesis
  using * [where ?R.1 = R]
  using * [where ?R.1 =  $\langle R^{-1-1} \rangle$ ]
  by (auto intro!: rel-setI)
qed

```

```

lemma [transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]:  $\langle \text{bi-total } S \rangle$ 
  assumes [transfer-rule]:  $\langle \text{bi-unique } S \rangle$ 
  assumes [transfer-rule]:  $\langle \text{bi-total } R \rangle$ 
  assumes [transfer-rule]:  $\langle \text{bi-unique } R \rangle$ 
  shows  $\langle \text{rel-topology } R \implies \text{rel-topology } S \implies (R \implies S) \implies (=) \rangle$  continuous-map continuous-map
  unfolding continuous-map-def Pi-def
  by transfer-prover

```

```

lemma limitin-closedin:
  assumes  $\langle \text{limitin } T \ f \ c \ F \rangle$ 
  assumes  $\langle \text{range } f \subseteq S \rangle$ 
  assumes  $\langle \text{closedin } T \ S \rangle$ 
  assumes  $\langle \neg \text{trivial-limit } F \rangle$ 

```

```

  shows  $\langle c \in S \rangle$ 
proof -
  from assms have  $\langle T \text{ closure-of } S = S \rangle$ 
  by (simp add: closure-of-eq)
  moreover have  $\langle c \in T \text{ closure-of } S \rangle$ 
  using assms(1) - assms(4) apply (rule limitin-closure-of)
  using range-subsetD[OF assms(2)] by auto
  ultimately show ?thesis
  by simp
qed

```

lemma *closure-nhds-principal*: $\langle a \in \text{closure } A \iff \text{inf } (\text{nhds } a) (\text{principal } A) \neq \text{bot} \rangle$

```

proof (rule iffI)
  show  $\langle \text{inf } (\text{nhds } a) (\text{principal } A) \neq \text{bot} \rangle$  if  $\langle a \in \text{closure } A \rangle$ 
  proof (cases  $\langle a \in A \rangle$ )
    case True
    thus ?thesis
    unfolding filter-eq-iff eventually-inf eventually-principal eventually-nhds by force
  next
  case False
  have at a within A  $\neq \text{bot}$ 
  using False that by (subst at-within-eq-bot-iff) auto
  also have at a within A  $= \text{inf } (\text{nhds } a) (\text{principal } A)$ 
  using False by (simp add: at-within-def)
  finally show ?thesis .
qed
  show  $\langle a \in \text{closure } A \rangle$  if  $\langle \text{inf } (\text{nhds } a) (\text{principal } A) \neq \text{bot} \rangle$ 
  by (metis Diff-empty Diff-insert0 at-within-def closure-subset not-in-closure-trivial-limitI
subsetD that)
qed

```

lemma *limit-in-closure*:

```

  assumes lim:  $\langle f \longrightarrow x \rangle F$ 
  assumes nt:  $\langle F \neq \text{bot} \rangle$ 
  assumes inA:  $\langle \forall_F x \text{ in } F. f x \in A \rangle$ 
  shows  $\langle x \in \text{closure } A \rangle$ 
proof (rule Lim-in-closed-set[of - - F])
  show  $\forall_F x \text{ in } F. f x \in \text{closure } A$ 
  using inA by eventually-elim (use closure-subset in blast)
qed (use assms in auto)

```

lemma *filterlim-nhdsin-iff-limitin*:

```

 $\langle l \in \text{topspace } T \wedge \text{filterlim } f (\text{nhdsin } T l) F \iff \text{limitin } T f l F \rangle$ 
  unfolding limitin-def
proof safe
  fix U assume *:  $l \in \text{topspace } T \text{ filterlim } f (\text{nhdsin } T l) F \text{ openin } T \ U \ l \in U$ 

```

hence *eventually* $(\lambda y. y \in U)$ $(nhdsin\ T\ l)$
unfolding *eventually-nhdsin* **by** *blast*
thus *eventually* $(\lambda x. f\ x \in U)$ F
using $*(2)$ *eventually-compose-filterlim* **by** *blast*
next
assume $*$: $l \in\ topspace\ T\ \forall\ U. openin\ T\ U \wedge l \in U \longrightarrow (\forall_F\ x\ in\ F. f\ x \in U)$
show *filterlim* f $(nhdsin\ T\ l)$ F
unfolding *filterlim-def* *le-filter-def* *eventually-filtermap*
proof *safe*
fix $P :: 'a \Rightarrow bool$
assume *eventually* P $(nhdsin\ T\ l)$
then obtain U **where** $U: openin\ T\ U\ l \in U\ \forall x \in U. P\ x$
using $*(1)$ **unfolding** *eventually-nhdsin* **by** *blast*
with $*$ **have** *eventually* $(\lambda x. f\ x \in U)$ F
by *blast*
thus *eventually* $(\lambda x. P\ (f\ x))$ F
by *eventually-elim* $(use\ U\ in\ blast)$
qed
qed

lemma *pullback-topology-bi-cont*:
fixes $g :: \langle 'a \Rightarrow ('b \Rightarrow 'c::topological-space) \rangle$
and $f :: \langle 'a \Rightarrow 'a \Rightarrow 'a \rangle$ **and** $f' :: \langle 'c \Rightarrow 'c \Rightarrow 'c \rangle$
assumes $gf-f'g: \langle \bigwedge a\ b\ i. g\ (f\ a\ b)\ i = f'\ (g\ a\ i)\ (g\ b\ i) \rangle$
assumes $f'-cont: \langle \bigwedge a' b'. (case-prod\ f' \longrightarrow f'\ a' b')\ (nhds\ a' \times_F\ nhds\ b') \rangle$
defines $\langle T \equiv pullback-topology\ UNIV\ g\ euclidean \rangle$
shows $\langle LIM\ (x,y)\ nhdsin\ T\ a \times_F\ nhdsin\ T\ b. f\ x\ y :> nhdsin\ T\ (f\ a\ b) \rangle$
proof –
have *topspace[simp]*: $\langle topspace\ T = UNIV \rangle$
unfolding *T-def* *topspace-pullback-topology* **by** *simp*
have *openin*: $\langle openin\ T\ U \longleftrightarrow (\exists V. open\ V \wedge U = g\ -' V) \rangle$ **for** U
by $(simp\ add: assms\ openin-pullback-topology)$

have $1: \langle nhdsin\ T\ a = filtercomap\ g\ (nhds\ (g\ a)) \rangle$
for $a :: 'a$
by $(auto\ simp\ add: filter-eq-iff\ eventually-filtercomap\ eventually-nhds\ eventually-nhdsin\ openin)$

have $\langle ((g \circ case-prod\ f) \longrightarrow g\ (f\ a\ b))\ (nhdsin\ T\ a \times_F\ nhdsin\ T\ b) \rangle$
proof $(unfold\ tendsto-def, intro\ allI\ impI)$
fix S **assume** $\langle open\ S \rangle$ **and** $gfS: \langle g\ (f\ a\ b) \in S \rangle$
obtain U **where** $gf-PiE: \langle g\ (f\ a\ b) \in Pi_E\ UNIV\ U \rangle$ **and** $openU: \langle \forall i. openin\ euclidean\ (U\ i) \rangle$
and $finiteD: \langle finite\ \{i. U\ i \neq topspace\ euclidean\} \rangle$ **and** $US: \langle Pi_E\ UNIV\ U \subseteq S \rangle$
using *product-topology-open-contains-basis* $[OF\ \langle open\ S \rangle[unfolding\ open-fun-def]\ gfS]$
by *auto*

define D **where** $\langle D = \{i. U\ i \neq UNIV\} \rangle$
with $finiteD$ **have** $\langle finite\ D \rangle$

by *auto*

from *openU* **have** *openU*: $\langle \text{open } (U \ i) \rangle$ **for** *i*
 by *auto*

have *: $\langle f' (g \ a \ i) (g \ b \ i) \in U \ i \rangle$ **for** *i*
 by (*metis PiE-mem gf-PiE iso-tuple-UNIV-I gf-f'g*)

have $\langle \forall_F x \text{ in } \text{nhds } (g \ a \ i) \times_F \text{nhds } (g \ b \ i). \text{ case-prod } f' \ x \in U \ i \rangle$ **for** *i*
 using *tendsto-def[THEN iffD1, rule-format, OF f'-cont openU *, of i]* **by** –

then obtain *Pa Pb* **where** $\langle \text{eventually } (Pa \ i) (\text{nhds } (g \ a \ i)) \rangle$ **and** $\langle \text{eventually } (Pb \ i) (\text{nhds } (g \ b \ i)) \rangle$
and *PaPb-plus*: $\langle (\forall x \ y. Pa \ i \ x \longrightarrow Pb \ i \ y \longrightarrow f' \ x \ y \in U \ i) \rangle$ **for** *i*
unfolding *eventually-prod-filter* **by** (*metis prod.simps(2)*)

from $\langle \bigwedge i. \text{eventually } (Pa \ i) (\text{nhds } (g \ a \ i)) \rangle$
obtain *Ua* **where** $\langle \text{open } (Ua \ i) \rangle$ **and** *a-Ua*: $\langle g \ a \ i \in Ua \ i \rangle$ **and** *Ua-Pa*: $\langle Ua \ i \subseteq \text{Collect } (Pa \ i) \rangle$ **for** *i*
unfolding *eventually-nhds*
apply *atomize-elim*
by (*metis mem-Collect-eq subsetI*)

from $\langle \bigwedge i. \text{eventually } (Pb \ i) (\text{nhds } (g \ b \ i)) \rangle$
obtain *Ub* **where** $\langle \text{open } (Ub \ i) \rangle$ **and** *b-Ub*: $\langle g \ b \ i \in Ub \ i \rangle$ **and** *Ub-Pb*: $\langle Ub \ i \subseteq \text{Collect } (Pb \ i) \rangle$ **for** *i*
unfolding *eventually-nhds*
apply *atomize-elim*
by (*metis mem-Collect-eq subsetI*)

have *UaUb-plus*: $\langle x \in Ua \ i \implies y \in Ub \ i \implies f' \ x \ y \in U \ i \rangle$ **for** *i x y*
by (*metis PaPb-plus Ua-Pa Ub-Pb mem-Collect-eq subsetD*)

define *Ua'* **where** $\langle Ua' \ i = (\text{if } i \in D \text{ then } Ua \ i \text{ else } UNIV) \rangle$ **for** *i*
define *Ub'* **where** $\langle Ub' \ i = (\text{if } i \in D \text{ then } Ub \ i \text{ else } UNIV) \rangle$ **for** *i*

have *Ua'-UNIV*: $\langle U \ i = UNIV \implies Ua' \ i = UNIV \rangle$ **for** *i*
by (*simp add: D-def Ua'-def*)

have *Ub'-UNIV*: $\langle U \ i = UNIV \implies Ub' \ i = UNIV \rangle$ **for** *i*
by (*simp add: D-def Ub'-def*)

have [*simp*]: $\langle \text{open } (Ua' \ i) \rangle$ **for** *i*
by (*simp add: Ua'-def <open (Ua -)>*)

have [*simp*]: $\langle \text{open } (Ub' \ i) \rangle$ **for** *i*
by (*simp add: Ub'-def <open (Ub -)>*)

have *a-Ua'*: $\langle g \ a \ i \in Ua' \ i \rangle$ **for** *i*
by (*simp add: Ua'-def a-Ua*)

have *b-Ub'*: $\langle g \ b \ i \in Ub' \ i \rangle$ **for** *i*
by (*simp add: Ub'-def b-Ub*)

have *UaUb'-plus*: $\langle a' \in Ua' \ i \implies b' \in Ub' \ i \implies f' \ a' \ b' \in U \ i \rangle$ **for** *i a' b'*
apply (*cases <i ∈ D>*)
using *UaUb-plus* **by** (*auto simp add: Ua'-def Ub'-def D-def*)


```

define  $Ua''$  where  $\langle Ua'' = Pi\ UNIV\ Ua' \rangle$ 
define  $Ub''$  where  $\langle Ub'' = Pi\ UNIV\ Ub' \rangle$ 

have  $\langle open\ Ua'' \rangle$ 
  using  $finiteD\ Ua'\text{-UNIV}$ 
  by (auto simp add: open-fun-def Ua''-def PiE-UNIV-domain
    openin-product-topology-alt D-def intro!: exI[where x= $\langle Ua' \rangle$ ] intro: rev-finite-subset)
have  $\langle open\ Ub'' \rangle$ 
  using  $finiteD\ Ub'\text{-UNIV}$ 
  by (auto simp add: open-fun-def Ub''-def PiE-UNIV-domain
    openin-product-topology-alt D-def intro!: exI[where x= $\langle Ub' \rangle$ ] intro: rev-finite-subset)
have  $a\text{-}Ua''$ :  $\langle g\ a \in Ua'' \rangle$ 
  by (simp add: Ua''-def a-Ua')
have  $b\text{-}Ub''$ :  $\langle g\ b \in Ub'' \rangle$ 
  by (simp add: Ub''-def b-Ub')
have  $UaUb''\text{-plus}$ :  $\langle a' \in Ua'' \implies b' \in Ub'' \implies f'(a' i) (b' i) \in U\ i \rangle$  for  $i\ a'\ b'$ 
  using  $UaUb''\text{-plus}$  by (force simp add: Ua''-def Ub''-def)

define  $Ua'''$  where  $\langle Ua''' = g\ \text{-}'\ Ua'' \rangle$ 
define  $Ub'''$  where  $\langle Ub''' = g\ \text{-}'\ Ub'' \rangle$ 
have  $\langle openin\ T\ Ua''' \rangle$ 
  using  $\langle open\ Ua'' \rangle$  by (auto simp: openin Ua'''-def)
have  $\langle openin\ T\ Ub''' \rangle$ 
  using  $\langle open\ Ub'' \rangle$  by (auto simp: openin Ub'''-def)
have  $a\text{-}Ua'''$ :  $\langle a \in Ua''' \rangle$ 
  by (simp add: Ua'''-def a-Ua'')
have  $b\text{-}Ub'''$ :  $\langle b \in Ub''' \rangle$ 
  by (simp add: Ub'''-def b-Ub'')
have  $UaUb'''\text{-plus}$ :  $\langle a \in Ua''' \implies b \in Ub''' \implies f'(g\ a\ i) (g\ b\ i) \in U\ i \rangle$  for  $i\ a\ b$ 
  by (simp add: Ua'''-def UaUb''-plus Ub'''-def)

define  $Pa'$  where  $\langle Pa'\ a \longleftrightarrow a \in Ua''' \rangle$  for  $a$ 
define  $Pb'$  where  $\langle Pb'\ b \longleftrightarrow b \in Ub''' \rangle$  for  $b$ 

have  $Pa'\text{-nhd}$ :  $\langle eventually\ Pa'\ (nhdsin\ T\ a) \rangle$ 
  using  $\langle openin\ T\ Ua''' \rangle$ 
  by (auto simp add: Pa'-def eventually-nhdsin intro!: exI[of -  $\langle Ua''' \rangle$ ] a-Ua'')
have  $Pb'\text{-nhd}$ :  $\langle eventually\ Pb'\ (nhdsin\ T\ b) \rangle$ 
  using  $\langle openin\ T\ Ub''' \rangle$ 
  by (auto simp add: Pb'-def eventually-nhdsin intro!: exI[of -  $\langle Ub''' \rangle$ ] b-Ub'')
have  $Pa'Pb'\text{-plus}$ :  $\langle (g \circ case\text{-prod}\ f) (a, b) \in S \rangle$  if  $\langle Pa'\ a \rangle \langle Pb'\ b \rangle$  for  $a\ b$ 
  using that UaUb''-plus US
  by (auto simp add: Pa'-def Pb'-def PiE-UNIV-domain Pi-iff gf-f'g)

show  $\langle \forall_F\ x\ in\ nhdsin\ T\ a \times_F\ nhdsin\ T\ b.\ (g \circ case\text{-prod}\ f)\ x \in S \rangle$ 
  using  $Pa'\text{-nhd}\ Pb'\text{-nhd}\ Pa'Pb'\text{-plus}$ 
  unfolding eventually-prod-filter
  apply  $-$ 

```

```

    apply (rule exI[of - Pa])
    apply (rule exI[of - Pb])
    by simp
qed
then show ?thesis
  unfolding 1 filterlim-filtercomap-iff by -
qed

```

definition $\langle \text{has-sum-in } T f A x \longleftrightarrow \text{limitin } T (\text{sum } f) x (\text{finite-subsets-at-top } A) \rangle$

lemma *has-sum-in-finite*:
assumes *finite F*
assumes $\langle \text{sum } f F \in \text{topspace } T \rangle$
shows $\text{has-sum-in } T f F (\text{sum } f F)$
using *assms*
by (*simp add: finite-subsets-at-top-finite has-sum-in-def limitin-def eventually-principal*)

definition $\langle \text{summable-on-in } T f A \longleftrightarrow (\exists x. \text{has-sum-in } T f A x) \rangle$

definition $\langle \text{infsum-in } T f A = (\text{let } L = \text{Collect } (\text{has-sum-in } T f A) \text{ in if card } L = 1 \text{ then the-elem } L \text{ else } 0) \rangle$

lemma *hausdorff-OFCLASS-t2-space*: $\langle \text{OFCLASS}('a::\text{topological-space}, \text{t2-space-class}) \text{ if } \langle \text{Hausdorff-space } (\text{euclidean } :: 'a \text{ topology}) \rangle$

proof *intro-classes*
fix *a b :: 'a*
assume $\langle a \neq b \rangle$
from *that*
show $\langle \exists U V. \text{open } U \wedge \text{open } V \wedge a \in U \wedge b \in V \wedge U \cap V = \{\} \rangle$
unfolding *Hausdorff-space-def disjnt-def*
using $\langle a \neq b \rangle$ **by** *auto*
qed

lemma *hausdorffI*:
assumes $\langle \bigwedge x y. x \in \text{topspace } T \implies y \in \text{topspace } T \implies x \neq y \implies \exists U V. \text{openin } T U \wedge \text{openin } T V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\} \rangle$
shows $\langle \text{Hausdorff-space } T \rangle$
using *assms* **by** (*auto simp: Hausdorff-space-def disjnt-def*)

lemma *hausdorff-euclidean[simp]*: $\langle \text{Hausdorff-space } (\text{euclidean } :: -::\text{t2-space topology}) \rangle$
apply (*rule hausdorffI*)
by (*metis (mono-tags, lifting) hausdorff open-openin*)

lemma *has-sum-in-unique*:
assumes $\langle \text{Hausdorff-space } T \rangle$
assumes $\langle \text{has-sum-in } T f A l \rangle$

```

assumes ⟨has-sum-in T f A l'⟩
shows ⟨l = l'⟩
using assms(2,3)[unfolded has-sum-in-def] - assms(1)
apply (rule limitin-Hausdorff-unique)
by simp

lemma infsun-in-def':
  assumes ⟨Hausdorff-space T⟩
  shows ⟨infsun-in T f A = (if summable-on-in T f A then (THE s. has-sum-in T f A s) else 0)⟩
proof (cases ⟨Collect (has-sum-in T f A) = {}⟩)
  case True
    then show ?thesis using True
    by (auto simp: infsun-in-def summable-on-in-def Let-def card-1-singleton-iff)
  next
    case False
    then have ⟨summable-on-in T f A⟩
    by (metis (no-types, lifting) empty-Collect-eq summable-on-in-def)
    from False ⟨Hausdorff-space T⟩
    have ⟨card (Collect (has-sum-in T f A)) = 1⟩
    by (metis (mono-tags, opaque-lifting) has-sum-in-unique is-singletonI' is-singleton-altdef mem-Collect-eq)
    then show ?thesis
    using ⟨summable-on-in T f A⟩
    by (smt (verit, best) assms card-1-singletonE has-sum-in-unique infsun-in-def mem-Collect-eq singletonI the-elem-eq the-equality)
qed

```

```

lemma has-sum-in-infsun-in:
  assumes ⟨Hausdorff-space T⟩ and summable: ⟨summable-on-in T f A⟩
  shows ⟨has-sum-in T f A (infsun-in T f A)⟩
  apply (simp add: infsun-in-def'[OF ⟨Hausdorff-space T⟩] summable)
  apply (rule theI'[of ⟨has-sum-in T f A⟩])
  using has-sum-in-unique[OF ⟨Hausdorff-space T⟩, of f A] summable
  by (meson summable-on-in-def)

```

```

lemma nhdsin-mono:
  assumes [simp]: ⟨ $\bigwedge x. \text{openin } T' x \implies \text{openin } T x$ ⟩
  assumes [simp]: ⟨topspace T = topspace T'⟩
  shows ⟨nhdsin T a ≤ nhdsin T' a⟩
  unfolding nhdsin-def
  by (auto intro!: INF-superset-mono)

```

```

lemma has-sum-in-cong:
  assumes  $\bigwedge x. x \in A \implies f x = g x$ 
  shows has-sum-in T f A x  $\longleftrightarrow$  has-sum-in T g A x
proof –

```

have $\langle (\forall_F x \text{ in } \text{finite-subsets-at-top } A. \text{ sum } f x \in U) \longleftrightarrow (\forall_F x \text{ in } \text{finite-subsets-at-top } A. \text{ sum } g x \in U) \rangle$ **for** U
apply (rule eventually-subst)
apply (subst eventually-finite-subsets-at-top)
by (metis (mono-tags, lifting) assms empty-subsetI finite.emptyI subset-eq sum.cong)
then show ?thesis
by (simp add: has-sum-in-def limitin-def)
qed

lemma *infsum-in-eqI'*:
fixes $f g :: \langle 'a \Rightarrow 'b :: \text{comm-monoid-add} \rangle$
assumes $\langle \bigwedge x. \text{ has-sum-in } T f A x \longleftrightarrow \text{ has-sum-in } T g B x \rangle$
shows $\langle \text{infsum-in } T f A = \text{infsum-in } T g B \rangle$
by (simp add: infsum-in-def assms[abs-def] summable-on-in-def)

lemma *infsum-in-cong*:
assumes $\bigwedge x. x \in A \implies f x = g x$
shows $\text{infsum-in } T f A = \text{infsum-in } T g A$
using assms *infsum-in-eqI'* *has-sum-in-cong* **by** blast

lemma *limitin-cong*: $\text{limitin } T f c F \longleftrightarrow \text{limitin } T g c F$ **if** eventually $(\lambda x. f x = g x) F$
by (smt (verit, best) eventually-elim2 limitin-transform-eventually that)

lemma *has-sum-in-reindex*:
assumes $\langle \text{inj-on } h A \rangle$
shows $\langle \text{has-sum-in } T g (h \text{ ` } A) x \longleftrightarrow \text{has-sum-in } T (g \circ h) A x \rangle$
proof –
have $\langle \text{has-sum-in } T g (h \text{ ` } A) x \longleftrightarrow \text{limitin } T (\text{sum } g) x (\text{finite-subsets-at-top } (h \text{ ` } A)) \rangle$
by (simp add: has-sum-in-def)
also have $\langle \dots \longleftrightarrow \text{limitin } T (\lambda F. \text{sum } g (h \text{ ` } F)) x (\text{finite-subsets-at-top } A) \rangle$
apply (subst filtermap-image-finite-subsets-at-top[symmetric])
by (simp-all add: assms eventually-filtermap limitin-def)
also have $\langle \dots \longleftrightarrow \text{limitin } T (\text{sum } (g \circ h)) x (\text{finite-subsets-at-top } A) \rangle$
apply (rule limitin-cong)
apply (rule eventually-finite-subsets-at-top-weakI)
apply (rule sum.reindex)
using assms subset-inj-on **by** blast
also have $\langle \dots \longleftrightarrow \text{has-sum-in } T (g \circ h) A x \rangle$
by (simp add: has-sum-in-def)
finally show ?thesis .
qed

lemma *summable-on-in-reindex*:
assumes $\langle \text{inj-on } h A \rangle$
shows $\langle \text{summable-on-in } T g (h \text{ ` } A) \longleftrightarrow \text{summable-on-in } T (g \circ h) A \rangle$
by (simp add: assms summable-on-in-def has-sum-in-reindex)

lemma *infsum-in-reindex*:
assumes $\langle \text{inj-on } h A \rangle$

shows $\langle \text{infsum-in } T g (h \text{ ' } A) = \text{infsum-in } T (g \circ h) A \rangle$
by (*metis Collect-cong assms has-sum-in-reindex infsum-in-def*)

lemma *has-sum-in-reindex-bij-betw*:

assumes *bij-betw g A B*

shows $\text{has-sum-in } T (\lambda x. f (g x)) A s \longleftrightarrow \text{has-sum-in } T f B s$

proof –

have $\langle \text{has-sum-in } T (\lambda x. f (g x)) A s \longleftrightarrow \text{has-sum-in } T f (g \text{ ' } A) s \rangle$

by (*metis (mono-tags, lifting) assms bij-betw-imp-inj-on has-sum-in-cong has-sum-in-reindex o-def*)

also have $\langle \dots = \text{has-sum-in } T f B s \rangle$

using *assms bij-betw-imp-surj-on* **by** *blast*

finally show *?thesis* .

qed

lemma *has-sum-euclidean-iff*: $\langle \text{has-sum-in euclidean } f A s \longleftrightarrow (f \text{ has-sum } s) A \rangle$

by (*simp add: has-sum-def has-sum-in-def*)

lemma *summable-on-euclidean-eq*: $\langle \text{summable-on-in euclidean } f A \longleftrightarrow f \text{ summable-on } A \rangle$

by (*auto simp add: infsum-def infsum-in-def has-sum-euclidean-iff[abs-def] has-sum-def t2-space-class.Lim-def summable-on-def summable-on-in-def*)

lemma *infsum-euclidean-eq*: $\langle \text{infsum-in euclidean } f A = \text{infsum } f A \rangle$

by (*auto simp add: infsum-def infsum-in-def' summable-on-euclidean-eq has-sum-euclidean-iff[abs-def] has-sum-def t2-space-class.Lim-def*)

lemma *infsum-in-reindex-bij-betw*:

assumes *bij-betw g A B*

shows $\text{infsum-in } T (\lambda x. f (g x)) A = \text{infsum-in } T f B$

proof –

have $\langle \text{infsum-in } T (\lambda x. f (g x)) A = \text{infsum-in } T f (g \text{ ' } A) \rangle$

by (*metis (mono-tags, lifting) assms bij-betw-imp-inj-on infsum-in-cong infsum-in-reindex o-def*)

also have $\langle \dots = \text{infsum-in } T f B \rangle$

using *assms bij-betw-imp-surj-on* **by** *blast*

finally show *?thesis* .

qed

lemma *limitin-parametric[transfer-rule]*:

includes *lifting-syntax*

assumes [*transfer-rule*]: $\langle \text{bi-unique } S \rangle$

shows $\langle (\text{rel-topology } S \text{ ==>} (R \text{ ==>} S) \text{ ==>} S \text{ ==>} \text{rel-filter } R \text{ ==>} (\longleftrightarrow)) \text{ limitin limitin} \rangle$

proof (*intro rel-funI, rename-tac T T' f f' l l' F F'*)

fix *T T' f f' l l' F F'*

assume [*transfer-rule*]: $\langle \text{rel-topology } S T T' \rangle$

assume [*transfer-rule*]: $\langle (R \text{ ==>} S) f f' \rangle$

assume [*transfer-rule*]: $\langle S l l' \rangle$

```

assume [transfer-rule]: ⟨rel-filter R F F'⟩

have topspace: ⟨l ∈ topspace T ↔ l' ∈ topspace T'⟩
  by transfer-prover

have open1: ⟨∀F x in F. f x ∈ U⟩
  if ⟨openin T U⟩ and ⟨l ∈ U⟩ and lhs: ⟨(∀ V. openin T' V ∧ l' ∈ V → (∀F x in F'. f' x
  ∈ V))⟩
  for U
proof –
  from ⟨rel-topology S T T'⟩ ⟨openin T U⟩
  obtain V where ⟨openin T' V⟩ and [transfer-rule]: ⟨rel-set S U V⟩
  by (smt (verit, best) Domainp.cases rel-fun-def rel-topology-def)
  with ⟨S l l'⟩ have ⟨l' ∈ V⟩
  by (metis (no-types, lifting) assms bi-uniqueDr rel-setD1 that(2))
  with lhs ⟨openin T' V⟩
  have ⟨∀F x in F'. f' x ∈ V⟩
  by auto
  then show ⟨∀F x in F. f x ∈ U⟩
  by transfer simp
qed

have open2: ⟨∀F x in F'. f' x ∈ V⟩
  if ⟨openin T' V⟩ and ⟨l' ∈ V⟩ and lhs: ⟨(∀ U. openin T U ∧ l ∈ U → (∀F x in F. f x ∈
  U))⟩
  for V
proof –
  from ⟨rel-topology S T T'⟩ ⟨openin T' V⟩
  obtain U where ⟨openin T U⟩ and [transfer-rule]: ⟨rel-set S U V⟩
  by (auto simp: rel-topology-def rel-fun-def)
  with ⟨S l l'⟩ have ⟨l ∈ U⟩
  by (metis (full-types) assms bi-unique-def rel-setD2 that(2))
  with lhs ⟨openin T U⟩
  have ⟨∀F x in F. f x ∈ U⟩
  by auto
  then show ⟨∀F x in F'. f' x ∈ V⟩
  by transfer simp
qed

from topspace open1 open2
show ⟨limitin T f l F = limitin T' f' l' F'⟩
  unfolding limitin-def by auto
qed

lemma finite-subsets-at-top-parametric[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: ⟨bi-unique R⟩
  shows ⟨(rel-set R ==> rel-filter (rel-set R)) finite-subsets-at-top finite-subsets-at-top⟩
proof (intro rel-funI)

```

```

fix A B assume ⟨rel-set R A B⟩
from ⟨bi-unique R⟩ obtain f where Rf: ⟨R x (f x)⟩ if ⟨x ∈ A⟩ for x
  by (metis (no-types, opaque-lifting) ⟨rel-set R A B⟩ rel-setD1)
have ⟨inj-on f A⟩
  by (metis (no-types, lifting) Rf assms bi-unique-def inj-onI)
have ⟨B = f ′ A⟩
  by (metis (mono-tags, lifting) Rf ⟨rel-set R A B⟩ assms bi-uniqueDr bi-unique-rel-set-lemma
image-cong)

have RfX: ⟨rel-set R X (f ′ X)⟩ if ⟨X ⊆ A⟩ for X
  apply (rule rel-setI)
  subgoal
    by (metis (no-types, lifting) Rf ⟨inj-on f A⟩ in-mono inj-on-image-mem-iff that)
  subgoal
    by (metis (no-types, lifting) Rf imageE subsetD that)
  done

have Piff: ⟨(∃ X. finite X ∧ X ⊆ A ∧ (∀ Y. finite Y ∧ X ⊆ Y ∧ Y ⊆ A ⟶ P (f ′ Y))) ⟷
(∃ X. finite X ∧ X ⊆ B ∧ (∀ Y. finite Y ∧ X ⊆ Y ∧ Y ⊆ B ⟶ P Y))⟩ for P
proof (rule iffI)
  assume ⟨∃ X. finite X ∧ X ⊆ A ∧ (∀ Y. finite Y ∧ X ⊆ Y ∧ Y ⊆ A ⟶ P (f ′ Y))⟩
  then obtain X where ⟨finite X⟩ and ⟨X ⊆ A⟩ and XP: ⟨finite Y ⟶ X ⊆ Y ⟶ Y ⊆
A ⟶ P (f ′ Y)⟩ for Y
    by auto
  define X' where ⟨X' = f ′ X⟩
  have ⟨finite X'⟩
    by (metis X'-def ⟨finite X⟩ finite-imageI)
  have ⟨X' ⊆ B⟩
    by (smt (verit, best) Rf X'-def ⟨X ⊆ A⟩ ⟨rel-set R A B⟩ assms bi-uniqueDr image-subset-iff
rel-setD1 subsetD)
  have ⟨P Y'⟩ if ⟨finite Y'⟩ and ⟨X' ⊆ Y'⟩ and ⟨Y' ⊆ B⟩ for Y'
  proof -
    define Y where ⟨Y = (f - ′ Y') ∩ A⟩
    have ⟨finite Y⟩
      by (metis Y-def ⟨inj-on f A⟩ finite-vimage-IntI that(1))
    moreover have ⟨X ⊆ Y⟩
      by (metis (no-types, lifting) X'-def Y-def ⟨X ⊆ A⟩ image-subset-iff-subset-vimage le-inf-iff
that(2))
    moreover have ⟨Y ⊆ A⟩
      by (metis (no-types, lifting) Y-def inf-le2)
    ultimately have ⟨P (f ′ Y)⟩
      by (rule XP)
    then show ⟨P Y'⟩
      by (metis (no-types, lifting) Int-greatest Y-def ⟨B = f ′ A⟩ dual-order.refl image-subset-iff-subset-vimage
inf-le1 subset-antisym subset-image-iff that(3))
  qed
  then show ⟨∃ X. finite X ∧ X ⊆ B ∧ (∀ Y. finite Y ∧ X ⊆ Y ∧ Y ⊆ B ⟶ P Y)⟩
    by (metis (no-types, opaque-lifting) ⟨X' ⊆ B⟩ ⟨finite X'⟩)
next

```

```

assume  $\langle \exists X. \text{finite } X \wedge X \subseteq B \wedge (\forall Y. \text{finite } Y \wedge X \subseteq Y \wedge Y \subseteq B \longrightarrow P Y) \rangle$ 
then obtain  $X$  where  $\langle \text{finite } X \rangle$  and  $\langle X \subseteq B \rangle$  and  $XP: \langle \text{finite } Y \Longrightarrow X \subseteq Y \Longrightarrow Y \subseteq B \Longrightarrow P Y \rangle$ 
for  $Y$ 
  by auto
define  $X'$  where  $\langle X' = (f \text{ ` } X) \cap A \rangle$ 
have  $\langle \text{finite } X' \rangle$ 
  by (simp add: X'-def  $\langle \text{finite } X \rangle$  inj-on f A finite-vimage-IntI)
have  $\langle X' \subseteq A \rangle$ 
  by (simp add: X'-def)
have  $\langle P (f \text{ ` } Y') \rangle$  if  $\langle \text{finite } Y' \rangle$  and  $\langle X' \subseteq Y' \rangle$  and  $\langle Y' \subseteq A \rangle$  for  $Y'$ 
proof –
  define  $Y$  where  $\langle Y = f \text{ ` } Y' \rangle$ 
have  $\langle \text{finite } Y \rangle$ 
  by (metis Y-def finite-imageI that(1))
moreover have  $\langle X \subseteq Y \rangle$ 
  using X'-def Y-def  $\langle B = f \text{ ` } A \rangle$   $\langle X \subseteq B \rangle$  that(2) by blast
moreover have  $\langle Y \subseteq B \rangle$ 
  by (metis Y-def  $\langle B = f \text{ ` } A \rangle$  image-mono that(3))
ultimately have  $\langle P Y \rangle$ 
  by (rule XP)
then show  $\langle P (f \text{ ` } Y') \rangle$ 
  by (smt (z3) Y-def  $\langle B = f \text{ ` } A \rangle$  imageE imageI subset-antisym subset-iff that(3) vimage-eq)
qed
then show  $\langle \exists X. \text{finite } X \wedge X \subseteq A \wedge (\forall Y. \text{finite } Y \wedge X \subseteq Y \wedge Y \subseteq A \longrightarrow P (f \text{ ` } Y)) \rangle$ 
  by (metis  $\langle X' \subseteq A \rangle$   $\langle \text{finite } X' \rangle$ )
qed

define  $Z$  where  $\langle Z = \text{filtermap } (\lambda M. (M, f \text{ ` } M)) (\text{finite-subsets-at-top } A) \rangle$ 
have  $\langle \forall_F (x, y) \text{ in } Z. \text{rel-set } R x y \rangle$ 
  by (auto intro!: eventually-finite-subsets-at-top-weakI simp add: Z-def eventually-filtermap RfX)
moreover have  $\langle \text{map-filter-on } \{(x, y). \text{rel-set } R x y\} \text{fst } Z = \text{finite-subsets-at-top } A \rangle$ 
  apply (rule filter-eq-iff[THEN iffD2])
  apply (subst eventually-map-filter-on)
subgoal
  by (auto intro!: eventually-finite-subsets-at-top-weakI simp add: Z-def eventually-filtermap RfX)[1]
subgoal
  by (auto simp add: Z-def eventually-filtermap eventually-finite-subsets-at-top RfX)
done
moreover have  $\langle \text{map-filter-on } \{(x, y). \text{rel-set } R x y\} \text{snd } Z = \text{finite-subsets-at-top } B \rangle$ 
  apply (rule filter-eq-iff[THEN iffD2])
  apply (subst eventually-map-filter-on)
subgoal
  by (auto intro!: eventually-finite-subsets-at-top-weakI simp add: Z-def eventually-filtermap RfX)[1]
subgoal
  by (simp add: Z-def eventually-filtermap eventually-finite-subsets-at-top RfX Piff)
done

```


ultimately show $\langle \text{rel-filter } (\text{rel-set } R) (\text{finite-subsets-at-top } A) (\text{finite-subsets-at-top } B) \rangle$
by $(\text{rule } \text{rel-filter.intros}[\text{where } Z=Z])$
qed

lemma *sum-parametric'*[*transfer-rule*]:
includes *lifting-syntax*
fixes $R :: \langle 'a \Rightarrow 'b \Rightarrow \text{bool} \rangle$ **and** $S :: \langle 'c::\text{comm-monoid-add} \Rightarrow 'd::\text{comm-monoid-add} \Rightarrow \text{bool} \rangle$
assumes [*transfer-rule*]: $\langle \text{bi-unique } R \rangle$
assumes [*transfer-rule*]: $\langle (S \text{ ===} \Rightarrow S \text{ ===} \Rightarrow S) (+) (+) \rangle$
assumes [*transfer-rule*]: $\langle S \ 0 \ 0 \rangle$
shows $\langle ((R \text{ ===} \Rightarrow S) \text{ ===} \Rightarrow \text{rel-set } R \text{ ===} \Rightarrow S) \text{ sum sum} \rangle$
proof (*intro rel-funI*)
fix $A \ B \ f \ g$ **assume** $\langle \text{rel-set } R \ A \ B \rangle$ **and** $\langle (R \text{ ===} \Rightarrow S) \ f \ g \rangle$
from $\langle \text{bi-unique } R \rangle$ **obtain** p **where** $Rf: \langle R \ x \ (p \ x) \rangle$ **if** $\langle x \in A \rangle$ **for** x
by (*metis (no-types, opaque-lifting) rel-set R A B rel-setD1*)
have $\langle \text{inj-on } p \ A \rangle$
by (*metis (no-types, lifting) Rf rel-set R A B bi-unique-def inj-onI*)
have $\langle B = p \ 'A \rangle$
by (*metis (mono-tags, lifting) Rf rel-set R A B rel-set R A B bi-uniqueDr bi-unique-rel-set-lemma image-cong*)

define *A-copy* **where** $\langle A\text{-copy} = A \rangle$

have $*$: $\langle S \ (f \ x + \text{sum } f \ F) \ (g \ (p \ x) + \text{sum } g \ (p \ 'F)) \rangle$
if [*transfer-rule*]: $\langle S \ (\text{sum } f \ F) \ (\text{sum } g \ (p \ 'F)) \rangle$ **and** [*simp*]: $\langle x \in A \ \text{for } x \ F \rangle$
by (*metis (no-types, opaque-lifting) Rf rel-set R A B assms(2) rel-fun-def that(1) that(2)*)
have *ind-step*: $\langle S \ (\text{sum } f \ (\text{insert } x \ F)) \ (\text{sum } g \ (p \ ' \text{insert } x \ F)) \rangle$
if $\langle S \ (\text{sum } f \ F) \ (\text{sum } g \ (p \ 'F)) \rangle$ $\langle x \in A \rangle$ $\langle x \notin F \rangle$ $\langle \text{finite } F \rangle$ $\langle F \subseteq A \rangle$ **for** $x \ F$
proof –
have $\text{sum } g \ (p \ ' \text{insert } x \ F) = g \ (p \ x) + \text{sum } g \ (p \ 'F)$
unfolding *image-insert* **using** *that*
by (*subst sum.insert*) (*use inj-onD[OF inj-on p A, of x] in auto*)
thus *?thesis*
using *that* $*$ **by** *simp*
qed

have $\langle S \ (\sum x \in A. f \ x) \ (\sum x \in p \ 'A. g \ x) \rangle$ **if** $\langle A \subseteq A\text{-copy} \rangle$
using *that*
apply (*induction A rule:infinite-finite-induct*)
unfolding *A-copy-def*
subgoal
by (*metis (no-types, lifting) inj-on p A assms(3) finite-image-iff subset-inj-on sum.infinite*)
using $\langle S \ 0 \ 0 \rangle$ *ind-step* **by** *auto*
hence $\langle S \ (\sum x \in A. f \ x) \ (\sum x \in p \ 'A. g \ x) \rangle$
by (*simp add: A-copy-def*)
also have $\langle \dots = (\sum x \in B. g \ x) \rangle$
by (*metis (full-types) B = p 'A*)
finally show $\langle S \ (\sum x \in A. f \ x) \ (\sum x \in B. g \ x) \rangle$

by –
qed

lemma *has-sum-in-parametric*[transfer-rule]:

includes *lifting-syntax*

fixes $R :: \langle 'a \Rightarrow 'b \Rightarrow \text{bool} \rangle$ **and** $S :: \langle 'c :: \text{comm-monoid-add} \Rightarrow 'd :: \text{comm-monoid-add} \Rightarrow \text{bool} \rangle$

assumes [transfer-rule]: $\langle \text{bi-unique } R \rangle$

assumes [transfer-rule]: $\langle \text{bi-unique } S \rangle$

assumes [transfer-rule]: $\langle (S \text{ ===== } S \text{ ===== } S) (+) (+) \rangle$

assumes [transfer-rule]: $\langle S \ 0 \ 0 \rangle$

shows $\langle (\text{rel-topology } S \text{ ===== } (R \text{ ===== } S) \text{ ===== } (\text{rel-set } R) \text{ ===== } S \text{ ===== } (=))$

has-sum-in has-sum-in

proof –

note *sum-parametric'*[transfer-rule]

show *?thesis*

unfolding *has-sum-in-def*

by *transfer-prover*

qed

lemma *has-sum-in-topspace*: $\langle \text{has-sum-in } T \ f \ A \ s \Longrightarrow s \in \text{topspace } T \rangle$

by (*metis has-sum-in-def limitin-def*)

lemma *summable-on-in-parametric*[transfer-rule]:

includes *lifting-syntax*

fixes $R :: \langle 'a \Rightarrow 'b \Rightarrow \text{bool} \rangle$

assumes [transfer-rule]: $\langle \text{bi-unique } R \rangle$

assumes [transfer-rule]: $\langle \text{bi-unique } S \rangle$

assumes [transfer-rule]: $\langle (S \text{ ===== } S \text{ ===== } S) (+) (+) \rangle$

assumes [transfer-rule]: $\langle S \ 0 \ 0 \rangle$

shows $\langle (\text{rel-topology } S \text{ ===== } (R \text{ ===== } S) \text{ ===== } (\text{rel-set } R) \text{ ===== } (=))$

summable-on-in

proof (*intro rel-funI*)

fix $T \ T'$ **assume** [transfer-rule]: $\langle \text{rel-topology } S \ T \ T' \rangle$

fix $f \ f'$ **assume** [transfer-rule]: $\langle (R \text{ ===== } S) \ f \ f' \rangle$

fix $A \ A'$ **assume** [transfer-rule]: $\langle \text{rel-set } R \ A \ A' \rangle$

define $ExT \ ExT'$ **where** $\langle ExT \ P \longleftrightarrow (\exists x \in \text{Collect } (\text{Domainp } S). \ P \ x) \rangle$ **and** $\langle ExT' \ P' \longleftrightarrow (\exists x \in \text{Collect } (\text{Rangep } S). \ P' \ x) \rangle$ **for** $P \ P'$

have [transfer-rule]: $\langle ((S \text{ ===== } (\longleftrightarrow)) \text{ ===== } (\longleftrightarrow)) \ ExT \ ExT' \rangle$

by (*smt (z3) Domainp-iff ExT'-def ExT-def RangePI Rangep.cases mem-Collect-eq rel-fun-def*)

from $\langle \text{rel-topology } S \ T \ T' \rangle$ **have** *top1*: $\langle \text{topspace } T \subseteq \text{Collect } (\text{Domainp } S) \rangle$

unfolding *rel-topology-def*

by (*metis (no-types, lifting) Domainp-set mem-Collect-eq openin-topspace subsetI*)

from $\langle \text{rel-topology } S \ T \ T' \rangle$ **have** *top2*: $\langle \text{topspace } T' \subseteq \text{Collect } (\text{Rangep } S) \rangle$

unfolding *rel-topology-def*

by (*metis (no-types, lifting) RangePI Rangep.cases mem-Collect-eq openin-topspace rel-setD2 subsetI*)


```

have cardLL': ⟨card L = 1 ⟷ card L' = 1⟩
  by (metis (no-types, lifting) ⟨rel-set S L L'⟩ assms(2) bi-unique-rel-set-lemma card-image)

have ⟨S (infsun-in T f A) (infsun-in T' f' A')⟩ if ⟨card L ≠ 1⟩
  using that cardLL' by (simp add: infsun-in-def L'-def L-def Let-def that ⟨S 0 0⟩ flip: sum-iff)

moreover have ⟨S (infsun-in T f A) (infsun-in T' f' A')⟩ if [simp]: ⟨card L = 1⟩
proof -
  have [simp]: ⟨card L' = 1⟩
    using that cardLL' by simp
  have ⟨S (the-elem L) (the-elem L')⟩
    using ⟨rel-set S L L'⟩
    by (metis (no-types, opaque-lifting) ⟨card L' = 1⟩ is-singleton-altdef is-singleton-the-elem
rel-setD1 singleton-iff that)
  then show ?thesis
    by (simp add: infsun-in-def flip: L'-def L-def)
qed

ultimately show ⟨S (infsun-in T f A) (infsun-in T' f' A')⟩
  by auto
qed

lemma infsun-parametric[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: ⟨bi-unique R⟩
  shows ⟨((R == => (=)) == => (rel-set R) == => (=)) infsun infsun⟩
  unfolding infsun-euclidean-eq[symmetric]
  by transfer-prover

lemma summable-on-transfer[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: ⟨bi-unique R⟩
  shows ⟨((R == => (=)) == => (rel-set R) == => (=)) Infinite-Sum.summable-on Infi-
nite-Sum.summable-on⟩
  unfolding summable-on-euclidean-eq[symmetric]
  by transfer-prover

lemma abs-gbinomial: ⟨abs (a gchoose n) = (-1)^(n - nat (ceiling a)) * (a gchoose n)⟩
proof -
  have ⟨(∏ i=0..<n. abs (a - of-nat i)) = (- 1) ^ (n - nat (ceiling a)) * (∏ i=0..<n. a -
of-nat i)⟩
  proof (induction n)
    case 0
    then show ?case
      by simp
  next
    case (Suc n)
    consider (geq) ⟨of-int n ≥ a⟩ | (lt) ⟨of-int n < a⟩
    by fastforce

```

```

then show ?case
proof cases
  case geq
    from geq have  $\langle \text{abs } (a - \text{of-int } n) = - (a - \text{of-int } n) \rangle$ 
      by simp
    moreover from geq have  $\langle (\text{Suc } n - \text{nat } (\text{ceiling } a)) = (n - \text{nat } (\text{ceiling } a)) + 1 \rangle$ 
      by (metis Suc-diff-le Suc-eq-plus1 ceiling-le nat-le-iff)
    ultimately show ?thesis
      apply (simp add: Suc)
      by (metis (no-types, lifting)  $\langle |a - \text{of-int } (\text{int } n)| = - (a - \text{of-int } (\text{int } n)) \rangle$  mult.assoc
mult-minus-right of-int-of-nat-eq)
  next
    case lt
      from lt have  $\langle \text{abs } (a - \text{of-int } n) = (a - \text{of-int } n) \rangle$ 
        by simp
      moreover from lt have  $\langle (\text{Suc } n - \text{nat } (\text{ceiling } a)) = (n - \text{nat } (\text{ceiling } a)) \rangle$ 
        by (smt (verit, ccfv-threshold) Suc-leI cancel-comm-monoid-add-class.diff-cancel diff-commute
diff-diff-cancel diff-le-self less-ceiling-iff linorder-not-le order-less-le zless-nat-eq-int-zless)
      ultimately show ?thesis
        by (simp add: Suc)
    qed
  qed
then show ?thesis
  by (simp add: gbinomial-prod-rev abs-prod)
qed

```

lemma *gbinomial-sum-lower-abs:*

```

fixes a ::  $\langle 'a :: \{\text{floor-ceiling}\} \rangle$ 
defines  $\langle a' \equiv \text{nat } (\text{ceiling } a) \rangle$ 
assumes  $\langle \text{of-nat } m \geq a - 1 \rangle$ 
shows  $\langle \sum_{k \leq m}. \text{abs } (a \text{ gchoose } k) =$ 
   $(-1)^{a'} * ((-1)^m * (a - 1 \text{ gchoose } m))$ 
   $- (-1)^{a'} * \text{of-bool } (a' > 0) * ((-1)^{(a'-1)} * (a - 1 \text{ gchoose } (a' - 1)))$ 
   $+ (\sum_{k < a'}. \text{abs } (a \text{ gchoose } k)) \rangle$ 

```

proof –

```

from assms
have  $\langle a' \leq \text{Suc } m \rangle$ 
  using ceiling-mono by force

```

```

have  $\langle (\sum_{k \leq m}. \text{abs } (a \text{ gchoose } k)) = (\sum_{k = a'..m}. \text{abs } (a \text{ gchoose } k)) + (\sum_{k < a'}. \text{abs } (a \text{ gchoose } k)) \rangle$ 

```

```

  apply (subst asm-rl[of  $\langle \{..m\} = \{a'..m\} \cup \{..<a'\} \rangle$ ])
  using  $\langle a' \leq \text{Suc } m \rangle$  apply auto[1]
  apply (subst sum.union-disjoint)
  by auto

```

```

also have  $\langle \dots = (\sum_{k = a'..m}. (-1)^{(k-a')} * (a \text{ gchoose } k)) + (\sum_{k < a'}. \text{abs } (a \text{ gchoose } k)) \rangle$ 
  apply (rule arg-cong[where  $f = \langle \lambda x. x + - \rangle$ ])
  apply (rule sum.cong[OF refl])
  apply (subst abs-gbinomial)

```

using a' -def **by** blast
also have $\langle \dots = (\sum k=a'..m. (-1)^\wedge k * (-1)^\wedge a' * (a \text{ gchoose } k)) + (\sum k < a'. \text{abs } (a \text{ gchoose } k)) \rangle$
apply (rule arg-cong[where $f = \langle \lambda x. x + \cdot \rangle$])
apply (rule sum.cong[OF refl])
by (simp add: power-diff-conv-inverse)
also have $\langle \dots = (-1)^\wedge a' * (\sum k=a'..m. (a \text{ gchoose } k) * (-1)^\wedge k) + (\sum k < a'. \text{abs } (a \text{ gchoose } k)) \rangle$
by (auto intro: sum.cong simp: sum-distrib-left)
also have $\langle \dots = (-1)^\wedge a' * (\sum k \leq m. (a \text{ gchoose } k) * (-1)^\wedge k) - (-1)^\wedge a' * (\sum k < a'. (a \text{ gchoose } k) * (-1)^\wedge k) + (\sum k < a'. \text{abs } (a \text{ gchoose } k)) \rangle$
apply (subst asm-rl[of $\langle \{..m\} = \{..<a'\} \cup \{a'..m\} \rangle$])
using $\langle a' \leq \text{Suc } m \rangle$ **apply** auto[1]
apply (subst sum.union-disjoint)
by (auto simp: distrib-left)
also have $\langle \dots = (-1)^\wedge a' * ((-1)^\wedge m * (a - 1 \text{ gchoose } m)) - (-1)^\wedge a' * (\sum k < a'. (a \text{ gchoose } k) * (-1)^\wedge k) + (\sum k < a'. \text{abs } (a \text{ gchoose } k)) \rangle$
apply (subst gbinomial-sum-lower-neg)
by simp
also have $\langle \dots = (-1)^\wedge a' * ((-1)^\wedge m * (a - 1 \text{ gchoose } m)) - (-1)^\wedge a' * \text{of-bool } (a' > 0) * ((-1)^\wedge (a'-1) * (a-1 \text{ gchoose } (a'-1))) + (\sum k < a'. \text{abs } (a \text{ gchoose } k)) \rangle$
apply (cases $\langle a' = 0 \rangle$)
subgoal
by simp
subgoal
by (subst asm-rl[of $\langle \{..<a'\} = \{..a'-1\} \rangle$]) (auto simp: gbinomial-sum-lower-neg)
done
finally show ?thesis
by -
qed

lemma abs-gbinomial-leq1:

fixes $a :: \langle 'a :: \{\text{linordered-field}\} \rangle$

assumes $\langle \text{abs } a \leq 1 \rangle$

shows $\langle \text{abs } (a \text{ gchoose } b) \leq 1 \rangle$

proof -

have *: $\langle -1 \leq a \rangle \langle a \leq 1 \rangle$

using abs-le-D2 assms minus-le-iff abs-le-iff assms **by** auto

have $\langle \text{abs } (a \text{ gchoose } b) = \text{abs } ((\prod i = 0..<b. a - \text{of-nat } i) / \text{fact } b) \rangle$

by (simp add: gbinomial-prod-rev)

also have $\langle \dots = \text{abs } ((\prod i=0..<b. a - \text{of-nat } i)) / \text{fact } b \rangle$

apply (subst abs-divide)

by simp

also have $\langle \dots = (\prod i=0..<b. \text{abs } (a - \text{of-nat } i)) / \text{fact } b \rangle$

apply (subst abs-prod) **by** simp

also have $\langle \dots \leq (\prod i=0..<b. \text{of-nat } (\text{Suc } i)) / \text{fact } b \rangle$

proof (intro divide-right-mono prod-mono conjI)

fix i **assume** $i \in \{0..<b\}$

```

have |a - of-nat i| ≤ |a| + |of-nat i|
  by linarith
also have |a| ≤ 1
  by fact
finally show |a - of-nat i| ≤ of-nat (Suc i)
  by simp
qed auto
also have ⟨... = fact b / fact b⟩
  by (subst (2) fact-prod-Suc) auto
also have ⟨... = 1⟩
  by simp
finally show ?thesis
  by -
qed

```

lemma *gbinomial-summable-abs*:

```

fixes a :: real
assumes ⟨a ≥ 0⟩ and ⟨a ≤ 1⟩
shows ⟨summable (λn. abs (a gchoose n))⟩
proof -
define a' where ⟨a' = nat (ceiling a)⟩
have a': ⟨a' = 0 ∨ a' = 1⟩
  by (metis One-nat-def a'-def assms(2) ceiling-le-one less-one nat-1 nat-mono order-le-less)
have aux1: ⟨abs x ≤ x' ⟹ abs y ≤ y' ⟹ abs z ≤ z' ⟹ x - y + z ≤ x' + y' + z'⟩ for x
y z x' y' z' :: real
  by auto
have ⟨(∑ i ≤ n. |a gchoose i|) = (- 1) ^ a' * ((- 1) ^ n * (a - 1 gchoose n)) -
(- 1) ^ a' * of-bool (0 < a') * ((- 1) ^ (a' - 1) * (a - 1 gchoose (a' - 1))) +
(∑ k < a'. |a gchoose k|)⟩ for n
  unfolding a'-def by (rule gbinomial-sum-lower-abs) (use assms in auto)
also have ⟨... n ≤ 1 + 1 + 1⟩ for n
  by (rule aux1) (use a' in ⟨auto simp add: abs-mult abs-gbinomial-leq1 assms⟩)
also have ⟨... = 3⟩
  by simp
finally show ?thesis
  by (meson abs-ge-zero bounded-imp-summable)
qed

```

lemma *summable-tendsto-times-n*:

```

fixes f :: ⟨nat ⇒ real⟩
assumes pos: ⟨∧ n. f n ≥ 0⟩
assumes dec: ⟨decseq (λn. (n+M) * f (n + M))⟩
assumes sum: ⟨summable f⟩
shows ⟨(λn. n * f n) ⟶ 0⟩
proof (rule ccontr)
assume lim-not-0: ⟨¬ (λn. n * f n) ⟶ 0⟩
obtain B where ⟨(λn. (n+M) * f (n+M)) ⟶ B⟩ and nfB': ⟨(n+M) * f (n+M) ≥ B⟩
for n
  apply (rule decseq-convergent[where B=0, OF dec])

```

```

    using pos that by auto
  then have lim-B:  $\langle \lambda n. n * f n \longrightarrow B \rangle$ 
    by - (rule LIMSEQ-offset)
  have  $\langle B \geq 0 \rangle$ 
    apply (subgoal-tac  $\langle \bigwedge n. n * f n \geq 0 \rangle$ )
    using Lim-bounded2 lim-B apply blast
    by (simp add: pos)
  moreover have  $\langle B \neq 0 \rangle$ 
    using lim-B lim-not-0 by blast
  ultimately have  $\langle B > 0 \rangle$ 
    by linarith

  have ge:  $\langle f n \geq B / n \rangle$  if  $\langle n \geq M \rangle$  for n
    using nfb'[of  $\langle n - M \rangle$ ] that  $\langle B > 0 \rangle$  by (auto simp: divide-simps mult-ac)

  have  $\langle \text{summable } (\lambda n. B / n) \rangle$ 
    by (rule summable-comparison-test'[where  $N=M$ ]) (use sum  $\langle B > 0 \rangle$  ge in auto)

  moreover have  $\langle \neg \text{summable } (\lambda n. B / n) \rangle$ 
  proof (rule ccontr)
    define C where  $\langle C = (\sum n. 1 / \text{real } n) \rangle$ 
    assume  $\langle \neg \neg \text{summable } (\lambda n. B / \text{real } n) \rangle$ 
    then have  $\langle \text{summable } (\lambda n. \text{inverse } B * (B / \text{real } n)) \rangle$ 
      using summable-mult by blast
    then have  $\langle \text{summable } (\lambda n. 1 / \text{real } n) \rangle$ 
      using  $\langle B \neq 0 \rangle$  by auto
    then have  $\langle (\sum n=1..m. 1 / \text{real } n) \leq C \rangle$  for m
      unfolding C-def by (rule sum-le-suminf) auto
    then have  $\langle \text{harm } m \leq C \rangle$  for m
      by (simp add: harm-def inverse-eq-divide)
    then have  $\langle \text{harm } (\text{nat } (\text{ceiling } (\text{exp } C))) \leq C \rangle$ 
      by -
    then have  $\langle \ln (\text{real } (\text{nat } (\text{ceiling } (\text{exp } C)))) + 1 \leq C \rangle$ 
      by (smt (verit, best) ln-le-harm)
    then show False
      by (smt (z3) exp-ln ln-ge-iff of-nat-0-le-iff real-nat-ceiling-ge)
  qed

  ultimately show False
    by simp
  qed

```

```

lemma gbinomial-tendsto-0:
  fixes a :: real
  assumes  $\langle a > -1 \rangle$ 
  shows  $\langle \lambda n. (a \text{ gchoose } n) \longrightarrow 0 \rangle$ 
  proof -

```



```

have thesis1: ⟨(λn. (a gchoose n)) ⟶ 0⟩ if ⟨a ≥ 0⟩ for a :: real
proof -
  define m where ⟨m = nat (floor a)⟩
  have m: ⟨a ≥ real m⟩ ⟨a ≤ real m + 1⟩
    by (simp-all add: m-def that)
  show ?thesis
  proof (insert m, induction m arbitrary: a)
    case 0
    then have *: ⟨a ≥ 0⟩ ⟨a ≤ 1⟩
      using assms by auto
    show ?case
      using gbinomial-summable-abs[OF *]
      using summable-LIMSEQ-zero tendsto-rabs-zero-iff by blast
  next
  case (Suc m)
  have 1: ⟨(λn. (a-1 gchoose n)) ⟶ 0⟩
    by (rule Suc.IH) (use Suc.prem in auto)
  then have ⟨(λn. (a-1 gchoose Suc n)) ⟶ 0⟩
    using filterlim-sequentially-Suc by blast
  with 1 have ⟨(λn. (a-1 gchoose n) + (a-1 gchoose Suc n)) ⟶ 0⟩
    by (simp add: tendsto-add-zero)
  then have ⟨(λn. (a gchoose Suc n)) ⟶ 0⟩
    using gbinomial-Suc-Suc[of ⟨a-1⟩] by simp
  then show ?case
    using filterlim-sequentially-Suc by blast
  qed
qed

have thesis2: ⟨(λn. (a gchoose n)) ⟶ 0⟩ if ⟨a > -1⟩ ⟨a ≤ 0⟩
proof -
  have decseq: ⟨decseq (λn. abs (a gchoose n))⟩
  proof (rule decseq-SucI)
    fix n
    have ⟨|a gchoose Suc n| = |a gchoose n| * (|a - real n| / (1 + n))⟩
      unfolding gbinomial-prod-rev by (simp add: abs-mult)
    also have ⟨... ≤ |a gchoose n|⟩
      apply (rule mult-left-le)
      using assms that(2) by auto
    finally show ⟨|a gchoose Suc n| ≤ |a gchoose n|⟩
      by -
  qed
  have abs-a1: ⟨abs (a+1) = a+1⟩
    using assms by auto

  have ⟨0 ≤ |a + 1 gchoose n|⟩ for n
    by simp
  moreover have ⟨decseq (λn. (n+1) * abs (a+1 gchoose (n+1)))⟩
    using decseq apply (simp add: gbinomial-rec abs-mult)
    by (smt (verit, best) decseq-def mult.commute mult-left-mono)

```

moreover have $\langle \text{summable } (\lambda n. \text{abs } (a+1 \text{ gchoose } n)) \rangle$
apply $(\text{rule gbinomial-summable-abs})$
using that by auto
ultimately have $\langle (\lambda n. n * \text{abs } (a+1 \text{ gchoose } n)) \longrightarrow 0 \rangle$
by $(\text{rule summable-tendsto-times-n})$
then have $\langle (\lambda n. \text{Suc } n * \text{abs } (a+1 \text{ gchoose } \text{Suc } n)) \longrightarrow 0 \rangle$
by $(\text{rule-tac LIMSEQ-ignore-initial-segment}[\text{where } k=1 \text{ and } a=0, \text{ simplified}])$
then have $\langle (\lambda n. \text{abs } (\text{Suc } n * (a+1 \text{ gchoose } \text{Suc } n))) \longrightarrow 0 \rangle$
by $(\text{simp add: abs-mult})$
then have $\langle (\lambda n. (a+1) * \text{abs } (a \text{ gchoose } n)) \longrightarrow 0 \rangle$
apply $(\text{subst } (\text{asm } \text{gbinomial-absorption}))$
by $(\text{simp add: abs-mult abs-a1})$
then have $\langle (\lambda n. \text{abs } (a \text{ gchoose } n)) \longrightarrow 0 \rangle$
using that(1) by force
then show $\langle (\lambda n. (a \text{ gchoose } n)) \longrightarrow 0 \rangle$
by $(\text{rule tendsto-rabs-zero-cancel})$
qed

from thesis1 thesis2 assms show ?thesis
using linorder-linear by blast
qed

lemma gbinomial-abs-sum:

fixes $a :: \text{real}$

assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$

shows $\langle (\lambda n. \text{abs } (a \text{ gchoose } n)) \text{ sums } 2 \rangle$

proof –

define a' **where** $\langle a' = \text{nat } (\text{ceiling } a) \rangle$

have $\langle a' = 1 \rangle$

using $a'\text{-def}$ $\text{assms}(1)$ $\text{assms}(2)$ **by** linarith

have $\text{lim: } \langle (\lambda n. (a - 1 \text{ gchoose } n)) \longrightarrow 0 \rangle$

by $(\text{simp add: assms}(1) \text{gbinomial-tendsto-0})$

have $\langle (\sum k \leq n. \text{abs } (a \text{ gchoose } k)) = (-1)^{a'} * ((-1)^n * (a - 1 \text{ gchoose } n)) - (-1)^{a'} * \text{of_bool } (0 < a') * ((-1)^{(a'-1)} * (a - 1 \text{ gchoose } (a' - 1))) + (\sum k < a'. |a \text{ gchoose } k|) \rangle$ **for** n

unfolding $a'\text{-def}$

apply $(\text{rule gbinomial-sum-lower-abs})$

using $\text{assms}(2)$ **by** linarith

also have $\langle \dots n = 2 - (-1)^n * (a - 1 \text{ gchoose } n) \rangle$ **for** n

using assms

by $(\text{auto simp add: } \langle a' = 1 \rangle)$

finally have $\langle (\sum k \leq n. \text{abs } (a \text{ gchoose } k)) = 2 - (-1)^n * (a - 1 \text{ gchoose } n) \rangle$ **for** n

by –

moreover have $\langle (\lambda n. 2 - (-1)^n * (a - 1 \text{ gchoose } n)) \longrightarrow 2 \rangle$

proof –

have $\langle (\lambda n. ((-1)^n * (a - 1 \text{ gchoose } n))) \longrightarrow 0 \rangle$

```

    by (rule tendsto-rabs-zero-cancel) (use lim in ⟨simp add: abs-mult tendsto-rabs-zero-iff⟩)
  then have ⟨(λn. 2 - (- 1) ^ n * (a - 1 gchoose n)) ⟶ 2 - 0⟩
    by (rule tendsto-diff[rotated]) simp
  then show ?thesis
    by simp
qed
ultimately have ⟨(λn. ∑ k≤n. abs (a gchoose k)) ⟶ 2⟩
  by auto
then show ?thesis
  using sums-def-le by blast
qed

lemma sums-has-sum:
  fixes s :: ⟨'a :: banach⟩
  assumes sums: ⟨f sums s⟩
  assumes abs-sum: ⟨summable (λn. norm (f n))⟩
  shows ⟨f has-sum s⟩ UNIV
proof (rule has-sumI-metric)
  fix e :: real assume ⟨0 < e⟩
  define e' where ⟨e' = e/2⟩
  then have ⟨e' > 0⟩
    using ⟨0 < e⟩ half-gt-zero by blast
  from suminf-exist-split[where r=e', OF ⟨0 < e'⟩ abs-sum]
  obtain N where ⟨norm (∑ i. norm (f (i + N))) < e'⟩
    by auto
  then have N: ⟨(∑ i. norm (f (i + N))) < e'⟩
    by auto
  then have N': ⟨norm (∑ i. f (i + N)) < e'⟩
    apply (rule dual-order.strict-trans2)
    by (auto intro!: summable-norm summable-iff-shift[THEN iffD2] abs-sum)

  define X where ⟨X = {..<N}⟩
  then have ⟨finite X⟩
    by auto
  moreover have ⟨dist (sum f Y) s < e⟩ if ⟨finite Y⟩ and ⟨X ⊆ Y⟩ for Y
  proof -
    have ⟨dist (sum f Y) s = norm (s - sum f {..<N} - sum f (Y - {..<N}))⟩
      by (metis X-def diff-diff-eq2 dist-norm norm-minus-commute sum.subset-diff that(1)
        that(2))
    also have ⟨... ≤ norm (s - sum f {..<N}) + norm (sum f (Y - {..<N}))⟩
      using norm-triangle-ineq4 by blast
    also have ⟨... = norm (∑ i. f (i + N)) + norm (sum f (Y - {..<N}))⟩
      apply (subst suminf-minus-initial-segment)
      using sums sums-summable apply blast
      using sums sums-unique by blast
    also have ⟨... < e' + norm (sum f (Y - {..<N}))⟩
      using N' by simp
    also have ⟨... ≤ e' + norm (∑ i∈Y - {..<N}. norm (f i))⟩
      apply (rule add-left-mono)

```

```

    by (smt (verit, best) real-norm-def sum-norm-le)
  also have ⟨... ≤ e' + (∑ i∈Y-{..<N}. norm (f i))⟩
    apply (rule add-left-mono)
    by (simp add: sum-nonneg)
  also have (∑ i∈Y-{..<N}. norm (f i)) = (∑ i|i+N∈Y. norm (f (i + N)))
    by (rule sum.reindex-bij-witness[of - λi. i + N λi. i - N]) auto
  also have ⟨e' + ... ≤ e' + (∑ i. norm (f (i + N)))⟩
    by (auto intro!: add-left-mono sum-le-suminf summable-iff-shift[THEN iffD2] abs-sum
finite-inverse-image ⟨finite Y⟩)
    also have ⟨... ≤ e' + e'⟩
      using N by simp
    also have ⟨... = e⟩
      by (simp add: e'-def)
    finally show ?thesis
      by -
  qed
  ultimately show ⟨∃ X. finite X ∧ X ⊆ UNIV ∧ (∀ Y. finite Y ∧ X ⊆ Y ∧ Y ⊆ UNIV →
dist (sum f Y) s < e)⟩
    by auto
  qed

```

lemma *sums-has-sum-pos*:

```

  fixes s :: real
  assumes ⟨f sums s⟩
  assumes ⟨∧ n. f n ≥ 0⟩
  shows ⟨(f has-sum s) UNIV⟩
  apply (rule sums-has-sum)
  apply (simp add: assms(1))
  using assms(1) assms(2) summable-def by auto

```

lemma *gbinomial-abs-has-sum*:

```

  fixes a :: real
  assumes ⟨a > 0⟩ and ⟨a ≤ 1⟩
  shows ⟨((λ n. abs (a gchoose n)) has-sum 2) UNIV⟩
  apply (rule sums-has-sum-pos)
  apply (rule gbinomial-abs-sum)
  using assms by auto

```

lemma *gbinomial-abs-has-sum-1*:

```

  fixes a :: real
  assumes ⟨a > 0⟩ and ⟨a ≤ 1⟩
  shows ⟨((λ n. abs (a gchoose n)) has-sum 1) (UNIV - {0})⟩
  proof -
  have ⟨((λ n. abs (a gchoose n)) has-sum 2 - (∑ n∈{0}. abs (a gchoose n))) (UNIV - {0})⟩
    apply (rule has-sum-Diff)
    apply (rule gbinomial-abs-has-sum)
    using assms apply auto[2]
    apply (rule has-sum-finite)
  by auto

```

then show *?thesis*
by *simp*
qed

lemma *gbinomial-abs-summable*:
fixes $a :: \text{real}$
assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle \lambda n. (a \text{ gchoose } n) \text{ abs-summable-on } UNIV \rangle$
using *assms* **by** (*auto intro!*: *has-sum-imp-summable gbinomial-abs-has-sum*)

lemma *gbinomial-abs-summable-1*:
fixes $a :: \text{real}$
assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle \lambda n. (a \text{ gchoose } n) \text{ abs-summable-on } UNIV - \{0\} \rangle$
using *assms* **by** (*auto intro!*: *has-sum-imp-summable gbinomial-abs-has-sum-1*)

lemma *has-sum-singleton[simp]*: $\langle (f \text{ has-sum } y) \{x\} \longleftrightarrow f x = y \rangle$ **for** $y :: \langle 'a :: \{ \text{comm-monoid-add, } t2\text{-space} \} \rangle$
using *has-sum-finite[of $\langle \{x\} \rangle$ infsumI[of $f \{x\} y$]* **by** *auto*

lemma *has-sum-sums*: $\langle f \text{ sums } s \rangle$ **if** $\langle (f \text{ has-sum } s) UNIV \rangle$
proof –
have $\langle \lambda n. \text{sum } f \{..<n\} \longrightarrow s \rangle$
proof (*unfold tendsto-def, intro allI impI*)
fix S **assume** $\langle \text{open } S \rangle$ **and** $\langle s \in S \rangle$
with $\langle (f \text{ has-sum } s) UNIV \rangle$
have $\langle \forall F. F \text{ in finite-subsets-at-top } UNIV. \text{sum } f F \in S \rangle$
using *has-sum-def tendsto-def* **by** *blast*
then
show $\langle \forall F. x \text{ in sequentially. sum } f \{..<x\} \in S \rangle$
using *eventually-compose-filterlim filterlim-lessThan-at-top* **by** *blast*
qed
then show *?thesis*
by (*simp add: sums-def*)
qed

lemma *The-eqI1*:
assumes $\langle \bigwedge x y. F x \implies F y \implies x = y \rangle$
assumes $\langle \exists z. F z \rangle$
assumes $\langle \bigwedge x. F x \implies P x = Q x \rangle$
shows $\langle P (\text{The } F) = Q (\text{The } F) \rangle$
by (*metis assms(1) assms(2) assms(3) theI*)

lemma *summable-on-uminus[intro!]*:
fixes $f :: \langle 'a \Rightarrow 'b :: \text{real-normed-vector} \rangle$
assumes $\langle f \text{ summable-on } A \rangle$
shows $\langle \lambda i. - f i \text{ summable-on } A \rangle$
apply (*subst asm-rl[of $\langle \lambda i. - f i = (\lambda i. (-1) *_R f i \rangle$]*)

apply *simp*
using *assms* **by** (*rule summable-on-scaleR-right*)

lemma *summable-on-diff*:

fixes $f\ g :: 'a \Rightarrow 'b::\text{real-normed-vector}$
assumes $\langle f \text{ summable-on } A \rangle$
assumes $\langle g \text{ summable-on } A \rangle$
shows $\langle (\lambda x. f\ x - g\ x) \text{ summable-on } A \rangle$
using *summable-on-add* [**where** $f=f$ **and** $g=\langle \lambda x. -\ g\ x \rangle$] *summable-on-uminus* [**where** $f=g$]
using *assms* **by** *auto*

lemma *gbinomial-1*: $\langle (1\ \text{gchoose } n) = \text{of-bool } (n \leq 1) \rangle$

proof –

consider $(0)\ \langle n=0 \rangle \mid (1)\ \langle n=1 \rangle \mid (\text{bigger})\ m$ **where** $\langle n = \text{Suc } (\text{Suc } m) \rangle$
by (*metis One-nat-def not0-implies-Suc*)

then show *?thesis*

proof *cases*

case *0*

then show *?thesis*

by *simp*

next

case *1*

then show *?thesis*

by *simp*

next

case *bigger*

then show *?thesis*

using *gbinomial-rec* [**where** $a=0$ **and** $k=\langle \text{Suc } m \rangle$]

by *simp*

qed

qed

lemma *gbinomial-a-Suc-n*:

$\langle (a\ \text{gchoose } \text{Suc } n) = (a\ \text{gchoose } n) * (a - n) / \text{Suc } n \rangle$
by (*simp add: gbinomial-prod-rev*)

lemma *has-sum-in-0* [*simp*]:

assumes $\langle 0 \in \text{topspace } T \rangle$

assumes $\langle \bigwedge x. x \in A \implies f\ x = 0 \rangle$

shows $\langle \text{has-sum-in } T\ f\ A\ 0 \rangle$

proof –

have $\langle \text{has-sum-in } T\ (\lambda -. 0)\ A\ 0 \rangle$

using *assms*

by (*simp add: has-sum-in-def sum.neutral-const* [*abs-def*])

then show *?thesis*

apply (*rule has-sum-in-cong* [*THEN iffD2, rotated*])

using *assms* by *simp*
qed

lemma *has-sum-diff*:

fixes $f g :: 'a \Rightarrow 'b :: \{\text{topological-ab-group-add}\}$
 assumes $\langle f \text{ has-sum } a \rangle A$
 assumes $\langle g \text{ has-sum } b \rangle A$
 shows $\langle (\lambda x. f x - g x) \text{ has-sum } (a - b) \rangle A$
 by (auto intro!: *has-sum-add has-sum-uminus*[*THEN iffD2*] *assms simp add: simp flip: add-uminus-conv-diff*)

lemma *has-sum-of-real*:

fixes $f :: 'a \Rightarrow \text{real}$
 assumes $\langle f \text{ has-sum } a \rangle A$
 shows $\langle (\lambda x. \text{of-real } (f x)) \text{ has-sum } (\text{of-real } a :: 'b :: \{\text{real-algebra-1, real-normed-vector}\}) \rangle A$
 apply (rule *has-sum-comm-additive*[*unfolded o-def*, where $f = \text{of-real}$])
 by (auto intro!: *additive.intro assms tendsto-of-real*)

lemma *summable-on-cdivide*:

fixes $f :: 'a \Rightarrow 'b :: \{\text{t2-space, topological-semigroup-mult, division-ring}\}$
 assumes $\langle f \text{ summable-on } A \rangle$
 shows $\langle \lambda x. f x / c \rangle \text{ summable-on } A$
 apply (subst *division-ring-class.divide-inverse*)
 using *assms summable-on-cmult-left* by blast

lemma *norm-abs*[*simp*]: $\langle \text{norm } (abs x) = \text{norm } x \rangle$ for $x :: 'a :: \{\text{idom-abs-sgn, real-normed-div-algebra}\}$

proof –

have $\langle \text{norm } x = \text{norm } (\text{sgn } x * abs x) \rangle$
 by (*simp add: sgn-mult-abs*)
 also have $\langle \dots = \text{norm } |x| \rangle$
 by (*simp add: norm-mult norm-sgn*)
 finally show ?thesis
 by *simp*

qed

thm *abs-summable-product*

lemma *abs-summable-product*:

fixes $x :: 'a \Rightarrow 'b :: \text{real-normed-div-algebra}$
 assumes *x2-sum*: $(\lambda i. (x i)^2) \text{ abs-summable-on } A$
 and *y2-sum*: $(\lambda i. (y i)^2) \text{ abs-summable-on } A$
 shows $(\lambda i. x i * y i) \text{ abs-summable-on } A$

proof (rule *nonneg-bdd-above-summable-on*)

show $\langle 0 \leq \text{norm } (x i * y i) \rangle$ for i
 by *simp*

show $\langle \text{bdd-above } (\text{sum } (\lambda i. \text{norm } (x i * y i))) \text{ } \{F. F \subseteq A \wedge \text{finite } F\} \rangle$

proof (rule *bdd-aboveI2*, *rename-tac F*)

fix F assume $\langle F \in \{F. F \subseteq A \wedge \text{finite } F\} \rangle$

then have *finite F* and $F \subseteq A$

by *auto*

```

have  $\text{norm } (x\ i * y\ i) \leq \text{norm } (x\ i * x\ i) + \text{norm } (y\ i * y\ i)$  for  $i$ 
  unfolding norm-mult
  by (smt mult-left-mono mult-nonneg-nonneg mult-right-mono norm-ge-zero)
hence  $(\sum_{i \in F}. \text{norm } (x\ i * y\ i)) \leq (\sum_{i \in F}. \text{norm } ((x\ i)^2) + \text{norm } ((y\ i)^2))$ 
  using [simp-trace]
  by (simp add: power2-eq-square sum-mono)
also have  $\dots = (\sum_{i \in F}. \text{norm } ((x\ i)^2)) + (\sum_{i \in F}. \text{norm } ((y\ i)^2))$ 
  by (simp add: sum.distrib)
also have  $\dots \leq (\sum_{\infty i \in A}. \text{norm } ((x\ i)^2)) + (\sum_{\infty i \in A}. \text{norm } ((y\ i)^2))$ 
  using x2-sum y2-sum  $\langle \text{finite } F \rangle \langle F \subseteq A \rangle$  by (auto intro!: finite-sum-le-infsum add-mono)
finally show  $\langle (\sum_{x a \in F}. \text{norm } (x\ x a * y\ x a)) \leq (\sum_{\infty i \in A}. \text{norm } ((x\ i)^2)) + (\sum_{\infty i \in A}. \text{norm } ((y\ i)^2)) \rangle$ 
  by simp
qed
qed

```

lemma *Cauchy-Schwarz-ineq-infsum:*

```

fixes  $x :: 'a \Rightarrow 'b :: \{\text{real-normed-div-algebra}\}$ 
assumes x2-sum:  $(\lambda i. (x\ i)^2)$  abs-summable-on  $A$ 
and y2-sum:  $(\lambda i. (y\ i)^2)$  abs-summable-on  $A$ 
shows  $\langle (\sum_{\infty i \in A}. \text{norm } (x\ i * y\ i)) \leq \text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (x\ i))^2) * \text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (y\ i))^2) \rangle$ 
proof -
  have  $\langle (\sum_{\infty i \in A}. \text{norm } (x\ i * y\ i)) \leq \text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (x\ i))^2) * \text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (y\ i))^2) \rangle$ 
  proof (rule infsum-le-finite-sums)
    show  $\langle (\lambda i. x\ i * y\ i)$  abs-summable-on  $A \rangle$ 
    using Misc-Tensor-Product.abs-summable-product x2-sum y2-sum by blast
    fix  $F$  assume  $\langle \text{finite } F \rangle$  and  $\langle F \subseteq A \rangle$ 

```

```

have sum1:  $\langle (\lambda i. (\text{norm } (x\ i))^2)$  summable-on  $A \rangle$ 
  by (metis (mono-tags, lifting) norm-power summable-on-cong x2-sum)
have sum2:  $\langle (\lambda i. (\text{norm } (y\ i))^2)$  summable-on  $A \rangle$ 
  by (metis (no-types, lifting) norm-power summable-on-cong y2-sum)

```

```

have  $\langle (\sum_{i \in F}. \text{norm } (x\ i * y\ i))^2 = (\sum_{i \in F}. \text{norm } (x\ i) * \text{norm } (y\ i))^2 \rangle$ 
  by (simp add: norm-mult)
also have  $\langle \dots \leq (\sum_{i \in F}. (\text{norm } (x\ i))^2) * (\sum_{i \in F}. (\text{norm } (y\ i))^2) \rangle$ 
  using Cauchy-Schwarz-ineq-sum by fastforce
also have  $\langle \dots \leq (\sum_{\infty i \in A}. (\text{norm } (x\ i))^2) * (\sum_{i \in F}. (\text{norm } (y\ i))^2) \rangle$ 
  using sum1  $\langle \text{finite } F \rangle \langle F \subseteq A \rangle$ 
  by (auto intro!: mult-right-mono finite-sum-le-infsum sum-nonneg)
also have  $\langle \dots \leq (\sum_{\infty i \in A}. (\text{norm } (x\ i))^2) * (\sum_{\infty i \in A}. (\text{norm } (y\ i))^2) \rangle$ 
  using sum2  $\langle \text{finite } F \rangle \langle F \subseteq A \rangle$ 
  by (auto intro!: mult-left-mono finite-sum-le-infsum infsum-nonneg)
also have  $\langle \dots = (\text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (x\ i))^2) * \text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (y\ i))^2))^2 \rangle$ 
  by (smt (verit, best) calculation real-sqrt-mult real-sqrt-pow2 zero-le-power2)
finally show  $\langle (\sum_{i \in F}. \text{norm } (x\ i * y\ i)) \leq \text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (x\ i))^2) * \text{sqrt } (\sum_{\infty i \in A}. (\text{norm } (y\ i))^2) \rangle$ 

```



```

(norm (y i))2
  apply (rule power2-le-imp-le)
  by (auto intro!: mult-nonneg-nonneg infsum-nonneg)
qed
then show ?thesis
  by -
qed

```

lemma *continuous-map-pullback-both*:

```

assumes cont: ⟨continuous-map T1 T2 g'⟩
assumes g'g: ⟨ $\bigwedge x. f1\ x \in \text{topspace } T1 \implies x \in A1 \implies g'\ (f1\ x) = f2\ (g\ x)$ ⟩
assumes top1: ⟨ $f1\ -' \text{topspace } T1 \cap A1 \subseteq g\ -' A2$ ⟩
shows ⟨continuous-map (pullback-topology A1 f1 T1) (pullback-topology A2 f2 T2) g'⟩
proof -
  from cont
  have ⟨continuous-map (pullback-topology A1 f1 T1) T2 (g' ∘ f1)⟩
    by (rule continuous-map-pullback)
  then have ⟨continuous-map (pullback-topology A1 f1 T1) T2 (f2 ∘ g)⟩
    apply (rule continuous-map-eq)
    by (simp add: g'g topspace-pullback-topology)
  then show ?thesis
    apply (rule continuous-map-pullback')
    by (simp add: top1 topspace-pullback-topology)
qed

```

lemma *onorm-case-prod-plus-leq*: ⟨ $\text{onorm } (\text{case-prod plus } :: - \implies 'a::\text{real-normed-vector}) \leq \text{sqrt } 2$ ⟩

```

  apply (rule onorm-bound)
  using norm-plus-leq-norm-prod by auto

```

lemma *bounded-linear-case-prod-plus*[*simp*]: ⟨*bounded-linear* (*case-prod plus*)⟩

```

  apply (rule bounded-linear-intro[where K=⟨sqrt 2⟩])
  by (auto simp add: scaleR-right-distrib norm-plus-leq-norm-prod mult.commute)

```

lemma *pullback-topology-twice*:

```

assumes ⟨ $f\ -' B \cap A = C$ ⟩
shows ⟨pullback-topology A f (pullback-topology B g T) = pullback-topology C (g ∘ f) T⟩
proof -
  have aux: ⟨ $S = A \iff S = B$ ⟩ if ⟨ $A = B$ ⟩ for A B S :: 'z
    using that by simp
  have *: ⟨ $(\exists V. (\text{openin } T\ U \wedge V = g\ -' U \cap B) \wedge S = f\ -' V \cap A) = (\text{openin } T\ U \wedge S = (g \circ f)\ -' U \cap C)$ ⟩ for S U
    apply (cases ⟨openin T U⟩)
    using assms
    by (auto intro!: aux simp: vimage-comp)
  then have *: ⟨ $(\exists V. (\exists U. \text{openin } T\ U \wedge V = g\ -' U \cap B) \wedge S = f\ -' V \cap A) = (\exists U. \text{openin } T\ U \wedge S = (g \circ f)\ -' U \cap C)$ ⟩ for S
    by metis
  show ?thesis

```

```

    by (auto intro!: * simp: topology-eq openin-pullback-topology)
qed

lemma pullback-topology-homeo-cong:
  assumes ⟨homeomorphic-map T S g⟩
  assumes ⟨range f ⊆ topspace T⟩
  shows ⟨pullback-topology A f T = pullback-topology A (g o f) S⟩
proof -
  have ⟨∃ Us. openin S Us ∧ f -‘ Ut ∩ A = (g o f) -‘ Us ∩ A⟩ if ⟨openin T Ut⟩ for Ut
  apply (rule exI[of - ⟨g -‘ Ut⟩])
  using assms that apply auto
  using homeomorphic-map-openness-eq apply blast
  by (smt (verit, best) homeomorphic-map-maps homeomorphic-maps-map openin-subset rangeI
subsetD)
  moreover have ⟨∃ Ut. openin T Ut ∧ (g o f) -‘ Us ∩ A = f -‘ Ut ∩ A⟩ if ⟨openin S Us⟩
for Us
  apply (rule exI[of - ⟨(g -‘ Us) ∩ topspace T⟩])
  using assms that apply auto
  by (meson continuous-map-open homeomorphic-imp-continuous-map)
  ultimately show ?thesis
  by (auto simp: topology-eq openin-pullback-topology)
qed

```

definition $\langle \text{opensets-in } T = \text{Collect } (\text{openin } T) \rangle$

— This behaves more nicely with the *transfer*-method (and friends) than *openin*. So when rewriting a subgoal, using, e.g., $\exists U \in \text{opensets } T. xxx$ instead of $\exists U. \text{openin } T U \longrightarrow xxx$ can make *transfer* work better.

lemma *opensets-in-parametric*[*transfer-rule*]:

```

  includes lifting-syntax
  assumes ⟨bi-unique R⟩
  shows ⟨rel-topology R ==> rel-set (rel-set R)⟩ opensets-in opensets-in
proof (intro rel-funI rel-setI)
  fix S T
  assume rel-topo: ⟨rel-topology R S T⟩
  fix U
  assume ⟨U ∈ opensets-in S⟩
  then show ⟨∃ V ∈ opensets-in T. rel-set R U V⟩
  by (smt (verit, del-insts) Domainp.cases mem-Collect-eq opensets-in-def rel-fun-def rel-topo
rel-topology-def)
next
  fix S T assume rel-topo: ⟨rel-topology R S T⟩
  fix U assume ⟨U ∈ opensets-in T⟩
  then show ⟨∃ V ∈ opensets-in S. rel-set R V U⟩
  by (smt (verit) RangeE mem-Collect-eq opensets-in-def rel-fun-def rel-topo rel-topology-def)
qed

```

lemma *hausdorff-parametric*[*transfer-rule*]:

```

  includes lifting-syntax

```

assumes $[transfer-rule]: \langle bi\text{-}unique\ R \rangle$
shows $\langle (rel\text{-}topology\ R ==> (\longleftrightarrow))\ Hausdorff\text{-}space\ Hausdorff\text{-}space \rangle$
proof –
have $Hausdorff\text{-}space\text{-}def'$: $\langle Hausdorff\text{-}space\ T \longleftrightarrow (\forall x \in topspace\ T. \forall y \in topspace\ T. x \neq y \rightarrow (\exists U \in opensets\text{-}in\ T. \exists V \in opensets\text{-}in\ T. x \in U \wedge y \in V \wedge U \cap V = \{\})) \rangle$
for $T :: \langle 'z\ topology \rangle$
unfolding $opensets\text{-}in\text{-}def\ Hausdorff\text{-}space\text{-}def\ disjnt\text{-}def\ Bex\text{-}def$ **by** $auto$
show $?thesis$
unfolding $Hausdorff\text{-}space\text{-}def'$
by $transfer\text{-}prover$
qed

lemma $sum\text{-}cmod\text{-}pos$:

assumes $\langle \bigwedge x. x \in A \implies f\ x \geq 0 \rangle$
shows $\langle (\sum x \in A. cmod\ (f\ x)) = cmod\ (\sum x \in A. f\ x) \rangle$
by $(metis\ (mono\text{-}tags,\ lifting)\ Re\text{-}sum\ assms\ cmod\text{-}Re\ sum.\text{cong}\ sum\text{-}nonneg)$

lemma $min\text{-}power\text{-}distrib\text{-}left$: $\langle (min\ x\ y) \wedge^n = min\ (x \wedge^n)\ (y \wedge^n) \rangle$ **if** $\langle x \geq 0 \rangle$ **and** $\langle y \geq 0 \rangle$
for $x\ y :: \langle - :: linordered\text{-}semidom \rangle$

by $(metis\ linorder\text{-}le\text{-}cases\ min.\text{absorb}\text{-}iff2\ min.\text{order}\text{-}iff\ power\text{-}mono\ that(1)\ that(2))$

lemma $abs\text{-}summable\text{-}times$:

fixes $f :: \langle 'a \Rightarrow 'c :: \{real\text{-}normed\text{-}algebra\} \rangle$ **and** $g :: \langle 'b \Rightarrow 'c \rangle$
assumes $sum\text{-}f$: $\langle f\ abs\text{-}summable\text{-}on\ A \rangle$
assumes $sum\text{-}g$: $\langle g\ abs\text{-}summable\text{-}on\ B \rangle$
shows $\langle (\lambda(i,j). f\ i * g\ j)\ abs\text{-}summable\text{-}on\ A \times B \rangle$

proof –

have $a1$: $\langle (\lambda j. norm\ (f\ i) * norm\ (g\ j))\ abs\text{-}summable\text{-}on\ B \rangle$ **if** $\langle i \in A \rangle$ **for** i
using $sum\text{-}g$ **by** $(simp\ add:\ summable\text{-}on\text{-}cmult\text{-}right)$

then have $a2$: $\langle (\lambda j. f\ i * g\ j)\ abs\text{-}summable\text{-}on\ B \rangle$ **if** $\langle i \in A \rangle$ **for** i
apply $(rule\ abs\text{-}summable\text{-}on\text{-}comparison\text{-}test)$

apply $(fact\ that)$
by $(simp\ add:\ norm\text{-}mult\text{-}ineq)$

from $sum\text{-}f$

have $\langle (\lambda x. \sum_{\infty} y \in B. norm\ (f\ x) * norm\ (g\ y))\ abs\text{-}summable\text{-}on\ A \rangle$

by $(auto\ simp\ add:\ infsum\text{-}cmult\text{-}right'\ infsum\text{-}nonneg\ intro!:\ summable\text{-}on\text{-}cmult\text{-}left)$

then have $b1$: $\langle (\lambda x. \sum_{\infty} y \in B. norm\ (f\ x * g\ y))\ abs\text{-}summable\text{-}on\ A \rangle$

apply $(rule\ abs\text{-}summable\text{-}on\text{-}comparison\text{-}test)$

using $a1\ a2$ **by** $(simp\text{-}all\ add:\ norm\text{-}mult\text{-}ineq\ infsum\text{-}mono\ infsum\text{-}nonneg)$

from $a2\ b1$ **show** $?thesis$

by $(intro\ abs\text{-}summable\text{-}on\text{-}Sigma\text{-}iff[THEN\ iffD2])\ auto$

qed

definition $\langle the\text{-}default\ def\ S = (if\ card\ S = 1\ then\ (THE\ x. x \in S)\ else\ def) \rangle$

lemma $card1I$:

assumes $a \in A$

assumes $\bigwedge x. x \in A \implies x = a$

shows $\langle \text{card } A = 1 \rangle$
by (*metis One-nat-def assms(1) assms(2) card-eq-Suc-0-ex1*)

lemma *the-default-CollectI*:
assumes $P a$
and $\bigwedge x. P x \implies x = a$
shows $P (\text{the-default } d (\text{Collect } P))$

proof –
have *card*: $\langle \text{card } (\text{Collect } P) = 1 \rangle$
apply (*rule card1I*)
using *assms* **by** *auto*
from *assms* **have** $\langle P (\text{THE } x. P x) \rangle$
by (*rule theI*)
then show *?thesis*
by (*simp add: the-default-def card*)

qed

lemma *the-default-singleton[simp]*: $\langle \text{the-default def } \{x\} = x \rangle$
unfolding *the-default-def* **by** *auto*

lemma *the-default-empty[simp]*: $\langle \text{the-default def } \{\} = \text{def} \rangle$
unfolding *the-default-def* **by** *auto*

lemma *the-default-The*: $\langle \text{the-default } z S = (\text{THE } x. x \in S) \rangle$ **if** $\langle \text{card } S = 1 \rangle$
by (*simp add: that the-default-def*)

lemma *the-default-parametricity[transfer-rule]*:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } T \rangle$
shows $\langle (T \implies \text{rel-set } T \implies T) \text{ the-default the-default} \rangle$

proof (*intro rel-funI, rename-tac def def' S S'*)
fix *def def'* **assume** [*transfer-rule*]: $\langle T \text{ def def}' \rangle$
fix $S S'$ **assume** [*transfer-rule*]: $\langle \text{rel-set } T S S' \rangle$
have *card-eq*: $\langle \text{card } S = \text{card } S' \rangle$

by *transfer-prover*

show $\langle T (\text{the-default def } S) (\text{the-default def}' S') \rangle$

proof (*cases* $\langle \text{card } S = 1 \rangle$)

case *True*

define *theS theS'* **where** [*no-atp*]: $\langle \text{theS} = (\text{THE } x. x \in S) \rangle$ **and** [*no-atp*]: $\langle \text{theS}' = (\text{THE } x. x \in S') \rangle$

from *True* **have** *cardS'*: $\langle \text{card } S' = 1 \rangle$

by (*simp add: card-eq*)

have $\langle \text{theS} \in S \rangle$

unfolding *theS-def*

by (*rule theI'*) (*use True in* $\langle \text{simp add: card-eq-Suc-0-ex1} \rangle$)

moreover have $\langle \text{theS}' \in S' \rangle$

unfolding *theS'-def*

by (*rule theI'*) (*use cardS' in* $\langle \text{simp add: card-eq-Suc-0-ex1} \rangle$)

```

ultimately have ⟨T theS theS'⟩
  using ⟨rel-set T S S'⟩ True cardS'
  by (auto simp: rel-set-def card-1-singleton-iff)
then show ?thesis
  by (simp add: True cardS' the-default-def theS-def theS'-def)
next
case False
then have cardS': ⟨card S' ≠ 1⟩
  by (simp add: card-eq)
show ?thesis
  using False cardS' ⟨T def def'⟩
  by (auto simp add: the-default-def)
qed
qed

definition ⟨rel-pred T P Q = rel-set T (Collect P) (Collect Q)⟩

lemma Collect-parametric[transfer-rule]:
  includes lifting-syntax
  shows ⟨(rel-pred T ==> rel-set T) Collect Collect⟩
  by (auto simp: rel-pred-def)

lemma fold-graph-finite:
  — Exists as comp-fun-commute-on.fold-graph-finite, but the comp-fun-commute-on-assumption
  is not needed.
  assumes fold-graph f z A y
  shows finite A
  using assms by induct simp-all

lemma fold-graph-parametric[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule, simp]: ⟨bi-unique T⟩
  shows ⟨((T ==> U ==> U) ==> U ==> rel-set T ==> rel-pred U)
    fold-graph fold-graph⟩
proof (intro rel-funI, rename-tac f f' z z' A A')
  fix f f' assume [transfer-rule, simp]: ⟨(T ==> U ==> U) f f'⟩
  fix z z' assume [transfer-rule, simp]: ⟨U z z'⟩
  fix A A' assume [transfer-rule, simp]: ⟨rel-set T A A'⟩
  have one-direction: ⟨∃ y'. fold-graph f' z' A' y' ∧ U y y'⟩ if ⟨fold-graph f z A y⟩
  and [transfer-rule]: ⟨U z z'⟩ ⟨(T ==> U ==> U) f f'⟩ ⟨rel-set T A A'⟩ ⟨bi-unique T⟩
  for f f' z z' A A' y and U :: ⟨'c1 ⇒ 'd1 ⇒ bool⟩ and T :: ⟨'a1 ⇒ 'b1 ⇒ bool⟩
  using ⟨fold-graph f z A y⟩ ⟨rel-set T A A'⟩
proof (induction arbitrary: A')
  case emptyI
  then show ?case
    by (metis ⟨U z z'⟩ empty-iff equals0I fold-graph.intros(1) rel-setD2)
next
case (insertI x A y)
from insertI have foldA: ⟨fold-graph f z A y⟩ and T-xA[transfer-rule]: ⟨rel-set T (insert x

```

A) A' and xA : $\langle x \notin A \rangle$
 by *simp-all*
 define DT RT where $\langle DT = Collect (Domainp T) \rangle$ and $\langle RT = Collect (Rangep T) \rangle$
 from TxA have $\langle x \in DT \rangle$
 by (*metis DT-def DomainPI insertCI mem-Collect-eq rel-set-def*)
 then obtain x' where [*transfer-rule*]: $\langle T x x' \rangle$
 unfolding DT -def by *blast*
 have $\langle x' \in A' \rangle$
 apply *transfer* by *simp*
 define A'' where $\langle A'' = A' - \{x'\} \rangle$
 then have A' -def: $\langle A' = insert x' A'' \rangle$
 using $\langle x' \in A' \rangle$ by *fastforce*
 have $\langle x' \notin A'' \rangle$
 unfolding A'' -def by *simp*
 have [*transfer-rule*]: $\langle rel-set T A A'' \rangle$
 apply (*subst asm-rl*[of $\langle A = (insert x A) - \{x\} \rangle$])
 using *insertI.hyps* apply *blast*
 unfolding A'' -def
 by *transfer-prover*
 from *insertI.IH*[*OF this*]
 obtain y'' where *foldA''*: $\langle fold-graph f' z' A'' y'' \rangle$ and [*transfer-rule*]: $\langle U y y'' \rangle$
 by *auto*
 define y' where $\langle y' = f' x' y'' \rangle$
 have $\langle fold-graph f' z' A' y' \rangle$
 unfolding A' -def y' -def
 using $\langle x' \notin A'' \rangle$, *foldA''*
 by (*rule fold-graph.intros*)
 moreover have $\langle U (f x y) y' \rangle$
 unfolding y' -def by *transfer-prover*
 ultimately show ?case
 by *auto*
 qed

show $\langle rel-pred U (fold-graph f z A) (fold-graph f' z' A') \rangle$
 unfolding *rel-pred-def rel-set-def bex-simps*
 apply *safe*
 subgoal
 by (*rule one-direction*[of $f z A - U z' T f'$]) *auto*
 subgoal
 by (*rule one-direction*[of $f' z' A' - \langle U^{-1-1} \rangle z \langle T^{-1-1} \rangle f$, *simplified*])
 (*auto simp flip: conversep-rel-fun*)
 done
 qed

lemma *Domainp-rel-filter*:
 assumes $\langle Domainp r = S \rangle$
 shows $\langle Domainp (rel-filter r) F \longleftrightarrow (F \leq principal (Collect S)) \rangle$
 proof (*intro iffI, elim Domainp.cases, hypsubst*)
 fix G

```

assume  $\langle \text{rel-filter } r \ F \ G \rangle$ 
then obtain  $Z$  where  $rZ: \langle \forall_F (x, y) \text{ in } Z. r \ x \ y \rangle$ 
  and  $ZF: \text{map-filter-on } \{(x, y). r \ x \ y\} \text{ fst } Z = F$ 
  and  $\text{map-filter-on } \{(x, y). r \ x \ y\} \text{ snd } Z = G$ 
  using  $\text{rel-filter.simps}$  by  $\text{blast}$ 
show  $\langle F \leq \text{principal } (\text{Collect } S) \rangle$ 
  using  $rZ$ 
  by ( $\text{auto simp flip: } ZF \text{ assms intro!: filter-leI elim!: eventually-mono}$ 
     $\text{simp: eventually-principal eventually-map-filter-on case-prod-unfold DomainPI}$ )
next
assume  $\text{asm: } \langle F \leq \text{principal } (\text{Collect } S) \rangle$ 
define  $Z$  where  $\langle Z = \text{inf } (\text{filtercomap } \text{fst } F) (\text{principal } \{(x, y). r \ x \ y\}) \rangle$ 
have  $rZ: \langle \forall_F (x, y) \text{ in } Z. r \ x \ y \rangle$ 
  by ( $\text{simp add: } Z\text{-def eventually-inf-principal}$ )
moreover
have  $\langle \forall_F x \text{ in } Z. P (\text{fst } x) \wedge (\text{case } x \text{ of } (x, xa) \Rightarrow r \ x \ xa) = \text{eventually } P \ F \rangle$  for  $P$ 
  using  $\text{asm}$  apply ( $\text{auto simp add: le-principal } Z\text{-def eventually-inf-principal eventually-filtercomap}$ )
  by ( $\text{smt (verit, del-insts) DomainpE assms eventually-elim2}$ )
then have  $\langle \text{map-filter-on } \{(x, y). r \ x \ y\} \text{ fst } Z = F \rangle$ 
  by ( $\text{simp add: filter-eq-iff eventually-map-filter-on } rZ$ )
ultimately show  $\langle \text{Domainp } (\text{rel-filter } r) \ F \rangle$ 
  by ( $\text{auto simp: Domainp-iff intro!: exI rel-filter.intros}$ )
qed

```

```

lemma  $\text{map-filter-on-cong}$ :
assumes [ $\text{simp}$ ]:  $\langle \forall_F x \text{ in } F. x \in D \rangle$ 
assumes  $\langle \bigwedge x. x \in D \Longrightarrow f \ x = g \ x \rangle$ 
shows  $\langle \text{map-filter-on } D \ f \ F = \text{map-filter-on } D \ g \ F \rangle$ 
apply ( $\text{rule filter-eq-iff[THEN iffD2, rule-format]}$ )
apply ( $\text{simp add: eventually-map-filter-on}$ )
apply ( $\text{rule eventually-subst}$ )
apply ( $\text{rule always-eventually}$ )
using  $\text{assms}(2)$  by  $\text{auto}$ 

```

```

lemma  $\text{filtermap-cong}$ :
assumes  $\langle \forall_F x \text{ in } F. f \ x = g \ x \rangle$ 
shows  $\langle \text{filtermap } f \ F = \text{filtermap } g \ F \rangle$ 
apply ( $\text{rule filter-eq-iff[THEN iffD2, rule-format]}$ )
apply ( $\text{simp add: eventually-filtermap}$ )
by ( $\text{smt (verit, del-insts) assms eventually-elim2}$ )

```

```

lemma  $\text{filtermap-INF-eq}$ :
assumes  $\text{inj-f: } \langle \text{inj-on } f \ X \rangle$ 
assumes  $B\text{-nonempty: } \langle B \neq \{\} \rangle$ 
assumes  $F\text{-bounded: } \langle \bigwedge b. b \in B \Longrightarrow F \ b \leq \text{principal } X \rangle$ 
shows  $\langle \text{filtermap } f \ (\bigcap (F \ ` B)) = (\bigcap b \in B. \text{filtermap } f \ (F \ b)) \rangle$ 
proof ( $\text{rule antisym}$ )

```

show $\langle \text{filtermap } f (\bigsqcap (F \text{ ' } B)) \leq (\bigsqcap b \in B. \text{filtermap } f (F b)) \rangle$
by (rule *filtermap-INF*)
define *f1* **where** $\langle f1 = \text{inv-into } X f \rangle$
have *f1f*: $\langle x \in X \implies f1 (f x) = x \rangle$ **for** *x*
by (simp add: *inj-f f1-def*)
have *ff1*: $\langle x \in f \text{ ' } X \implies x = f (f1 x) \rangle$ **for** *x*
by (simp add: *f1-def f-inv-into-f*)

have $\langle \text{filtermap } f (F b) \leq \text{principal } (f \text{ ' } X) \rangle$ **if** $\langle b \in B \rangle$ **for** *b*
by (metis *F-bounded filtermap-mono filtermap-principal that*)
then have $\langle (\bigsqcap b \in B. \text{filtermap } f (F b)) \leq (\bigsqcap b \in B. \text{principal } (f \text{ ' } X)) \rangle$
by (simp add: *INF-greatest INF-lower2*)
also have $\langle \dots = \text{principal } (f \text{ ' } X) \rangle$
by (simp add: *B-nonempty*)
finally have $\langle \forall_F x \text{ in } \bigsqcap b \in B. \text{filtermap } f (F b). x \in f \text{ ' } X \rangle$
using *B-nonempty le-principal by auto*
then have *: $\langle \forall_F x \text{ in } \bigsqcap b \in B. \text{filtermap } f (F b). x = f (f1 x) \rangle$
apply (rule *eventually-mono*)
by (simp add: *ff1*)

have $\langle \forall_F x \text{ in } F b. x \in X \rangle$ **if** $\langle b \in B \rangle$ **for** *b*
using *F-bounded le-principal that by blast*
then have **: $\langle \forall_F x \text{ in } F b. f1 (f x) = x \rangle$ **if** $\langle b \in B \rangle$ **for** *b*
apply (rule *eventually-mono*)
using that by (simp-all add: *f1f*)

have $\langle (\bigsqcap b \in B. \text{filtermap } f (F b)) = \text{filtermap } f (\text{filtermap } f1 (\bigsqcap b \in B. \text{filtermap } f (F b))) \rangle$
apply (simp add: *filtermap-filtermap*)
using * **by** (rule *filtermap-cong[where f=id, simplified]*)
also have $\langle \dots \leq \text{filtermap } f (\bigsqcap b \in B. \text{filtermap } f1 (\text{filtermap } f (F b))) \rangle$
apply (rule *filtermap-mono*)
by (rule *filtermap-INF*)
also have $\langle \dots = \text{filtermap } f (\bigsqcap b \in B. F b) \rangle$
apply (rule *arg-cong[where f=⟨filtermap -⟩*])
apply (rule *INF-cong, rule refl*)
unfolding *filtermap-filtermap*
using ** **by** (rule *filtermap-cong[where g=id, simplified]*)
finally show $\langle (\bigsqcap b \in B. \text{filtermap } f (F b)) \leq \text{filtermap } f (\bigsqcap (F \text{ ' } B)) \rangle$
by –

qed

lemma *filtermap-inf-eq*:
assumes $\langle \text{inj-on } f X \rangle$
assumes $\langle F1 \leq \text{principal } X \rangle$
assumes $\langle F2 \leq \text{principal } X \rangle$
shows $\langle \text{filtermap } f (F1 \sqcap F2) = \text{filtermap } f F1 \sqcap \text{filtermap } f F2 \rangle$
proof –
have $\langle \text{filtermap } f (F1 \sqcap F2) = \text{filtermap } f (\text{INF } F \in \{F1, F2\}. F) \rangle$
by *simp*


```

also have  $\langle \dots = (INF\ F \in \{F1, F2\}. \text{filtermap } f\ F) \rangle$ 
apply (rule filtermap-INF-eq[where  $X=X$ ])
using assms by auto
also have  $\langle \dots = \text{filtermap } f\ F1 \sqcap \text{filtermap } f\ F2 \rangle$ 
by simp
finally show ?thesis
by –
qed

```

definition $\langle \text{transfer-bounded-filter-Inf } B\ M = \text{Inf } M \sqcap \text{principal } B \rangle$

lemma *Inf-transfer-bounded-filter-Inf*: $\langle \text{Inf } M = \text{transfer-bounded-filter-Inf } UNIV\ M \rangle$
by (*metis inf-top.right-neutral top-eq-principal-UNIV transfer-bounded-filter-Inf-def*)

lemma *Inf-bounded-transfer-bounded-filter-Inf*:
assumes $\langle \bigwedge F. F \in M \implies F \leq \text{principal } B \rangle$
assumes $\langle M \neq \{\} \rangle$
shows $\langle \text{Inf } M = \text{transfer-bounded-filter-Inf } B\ M \rangle$
by (*simp add: Inf-less-eq assms(1) assms(2) inf-absorb1 transfer-bounded-filter-Inf-def*)

lemma *transfer-bounded-filter-Inf-parametric[transfer-rule]*:

```

includes lifting-syntax
fixes  $r :: \langle 'rep \Rightarrow 'abs \Rightarrow \text{bool} \rangle$ 
assumes [transfer-rule]:  $\langle \text{bi-unique } r \rangle$ 
shows  $\langle (\text{rel-set } r \implies \text{rel-set } (\text{rel-filter } r)) \implies \text{rel-filter } r \rangle$ 
      transfer-bounded-filter-Inf transfer-bounded-filter-Inf
proof (intro rel-funI, unfold transfer-bounded-filter-Inf-def)
fix  $BF\ BG$  assume BFBG[transfer-rule]:  $\langle \text{rel-set } r\ BF\ BG \rangle$ 
fix  $Fs\ Gs$  assume FsGs[transfer-rule]:  $\langle \text{rel-set } (\text{rel-filter } r)\ Fs\ Gs \rangle$ 
define  $D\ R$  where  $\langle D = \text{Collect } (\text{Domainp } r) \rangle$  and  $\langle R = \text{Collect } (\text{Rangep } r) \rangle$ 

```

```

have  $\langle \text{rel-set } r\ D\ R \rangle$ 
by (smt (verit) D-def Domainp-iff R-def RangePI Rangep.cases mem-Collect-eq rel-setI)
with  $\langle \text{bi-unique } r \rangle$ 
obtain  $f$  where  $\langle R = f ` D \rangle$  and [simp]:  $\langle \text{inj-on } f\ D \rangle$  and  $rf0: \langle x \in D \implies r\ x\ (f\ x) \rangle$  for  $x$ 
using bi-unique-rel-set-lemma
by metis
have  $rf: \langle r\ x\ y \longleftrightarrow x \in D \wedge f\ x = y \rangle$  for  $x\ y$ 
apply (auto simp: rf0)
using D-def apply auto[1]
using D-def assms bi-uniqueDr rf0 by fastforce

```

```

from BFBG
have  $\langle BF \subseteq D \rangle$ 
by (metis rel-setD1 rf subsetI)

```

have $G: \langle G = \text{filtermap } f\ F \rangle$ **if** $\langle \text{rel-filter } r\ F\ G \rangle$ **for** $F\ G$

```

using that proof cases
case (1 Z)
then have Z[simp]:  $\langle \forall F (x, y) \text{ in } Z. r x y \rangle$ 
  by –
then have  $\langle \text{filtermap } f F = \text{filtermap } f (\text{map-filter-on } \{(x, y). r x y\} \text{fst } Z) \rangle$ 
  using 1 by simp
also have  $\langle \dots = \text{map-filter-on } \{(x, y). r x y\} (f \circ \text{fst}) Z \rangle$ 
  unfolding map-filter-on-UNIV[symmetric]
  apply (subst map-filter-on-comp)
  using Z by simp-all
also have  $\langle \dots = G \rangle$ 
  apply (simp add: o-def rf)
  apply (subst map-filter-on-cong[where g=snd])
  using Z apply (rule eventually-mono)
  using 1 by (auto simp: rf)
finally show ?thesis
  by simp
qed

have rf-filter:  $\langle \text{rel-filter } r F G \iff F \leq \text{principal } D \wedge \text{filtermap } f F = G \rangle$  for F G
  apply (intro iffI conjI)
  apply (metis D-def DomainPI Domainp-rel-filter)
  using G apply simp
  by (metis D-def Domainp-iff Domainp-rel-filter G)

have FD:  $\langle F \leq \text{principal } D \rangle$  if  $\langle F \in Fs \rangle$  for F
  by (meson FsGs rel-setD1 rf-filter that)

from BFBG
have [simp]:  $\langle BG = f \text{‘ } BF \rangle$ 
  by (auto simp: rel-set-def rf)

from FsGs
have [simp]:  $\langle Gs = \text{filtermap } f \text{‘ } Fs \rangle$ 
  using G apply (auto simp: rel-set-def rf)
  by fastforce

show  $\langle \text{rel-filter } r (\bigcap Fs \sqcap \text{principal } BF) (\bigcap Gs \sqcap \text{principal } BG) \rangle$ 
proof (cases  $\langle Fs = \{\} \rangle$ )
  case True
  then have  $\langle Gs = \{\} \rangle$ 
    by transfer
  have  $\langle \text{rel-filter } r (\text{principal } BF) (\text{principal } BG) \rangle$ 
    by transfer-prover
  with True  $\langle Gs = \{\} \rangle$  show ?thesis
    by simp
next
  case False
  note False[simp]

```

```

then have [simp]: ⟨Gs ≠ {}⟩
  by transfer
have ⟨rel-filter r (∏ Fs ∩ principal BF) (filtermap f (∏ Fs ∩ principal BF))⟩
  apply (rule rf-filter[THEN iffD2])
  by (simp add: ⟨BF ⊆ D⟩ le-infI2)
then show ?thesis
  using FD ⟨BF ⊆ D⟩
  by (simp add: Inf-less-eq
    flip: filtermap-inf-eq[where X=D] filtermap-INF-eq[where X=D] flip: filtermap-principal)
qed
qed

```

definition ⟨transfer-inf-principal F M = F ∩ principal M⟩

```

lemma transfer-inf-principal-parametric[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: ⟨bi-unique T⟩
  shows ⟨(rel-filter T == => rel-set T == => rel-filter T) transfer-inf-principal transfer-inf-principal⟩
proof –
  have *: ⟨transfer-inf-principal F M = transfer-bounded-filter-Inf M {F}⟩ for F :: ⟨'z filter⟩
and M
  by (simp add: transfer-inf-principal-def[abs-def] transfer-bounded-filter-Inf-def)
  show ?thesis
  unfolding *
  apply transfer-prover-start
  apply transfer-step+
  by transfer-prover
qed

```

```

lemma continuous-map-is-continuous-at-point:
  assumes ⟨continuous-map T U f⟩
  shows ⟨filterlim f (nhdsin U (f l)) (atin T l)⟩
  by (metis assms atin-degenerate bot.extremum continuous-map-atin filterlim-iff-le-filtercomap
    filterlim-nhdsin-iff-limitin)

```

```

lemma set-compr-2-image-collect: ⟨{f x y |x y. P x y} = case-prod f ‘ Collect (case-prod P)⟩
  by fast

```

```

lemma closure-image-closure: ⟨continuous-on (closure S) f == => closure (f ‘ closure S) = closure
(f ‘ S)⟩
  by (smt (verit) closed-closure closure-closure closure-mono closure-subset image-closure-subset
    image-mono set-eq-subset)

```

```

lemma has-sum-reindex-bij-betw:
  assumes bij-betw g A B
  shows ((λx. f (g x)) has-sum l) A ⟷ (f has-sum l) B

```

proof –
have $\langle (\lambda x. f (g x)) \text{ has-sum } l \rangle A \longleftrightarrow \langle f \text{ has-sum } l \rangle (g \text{ ' } A) \rangle$
apply (rule *has-sum-reindex*[*symmetric, unfolded o-def*])
using *assms* *bij-betw-imp-inj-on* **by** *blast*
also have $\langle \dots \longleftrightarrow \langle f \text{ has-sum } l \rangle B \rangle$
using *assms* *bij-betw-imp-surj-on* **by** *blast*
finally show *?thesis* .
qed

lemma *enum-inj*:
assumes $i < \text{CARD}('a)$ **and** $j < \text{CARD}('a)$
shows $(\text{Enum.enum } ! i :: 'a::\text{enum}) = \text{Enum.enum } ! j \longleftrightarrow i = j$
using *inj-on-nth*[*OF enum-distinct, where* $I = \langle \{.. < \text{CARD}('a)\} \rangle$]
using *assms* **by** (*auto dest: inj-onD simp flip: card-UNIV-length-enum*)

lemma *closedin-vimage*:
assumes $\langle \text{closedin } U \ S \rangle$
assumes $\langle \text{continuous-map } T \ U \ f \rangle$
shows $\langle \text{closedin } T \ (\text{topspace } T \cap (f \text{ - ' } S)) \rangle$
by (*meson assms(1) assms(2) continuous-map-closedin-preimage-eq*)

lemma *join-forall*: $\langle (\forall x. P x) \wedge (\forall x. Q x) \longleftrightarrow (\forall x. P x \wedge Q x) \rangle$
by *auto*

lemma *closedin-if-converge-inside*:
fixes $A :: 'a \text{ set}$
assumes $AT: \langle A \subseteq \text{topspace } T \rangle$
assumes $xA: \langle \bigwedge (F::'a \text{ filter}) f x. F \neq \perp \implies \text{limitin } T f x F \implies \text{range } f \subseteq A \implies x \in A \rangle$
shows $\langle \text{closedin } T \ A \rangle$
proof (*cases* $\langle A = \{\} \rangle$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
then obtain a **where** $\langle a \in A \rangle$
by *auto*
define Ac **where** $\langle Ac = \text{topspace } T - A \rangle$
have $\langle \exists U. \text{openin } T \ U \wedge x \in U \wedge U \subseteq Ac \rangle$ **if** $\langle x \in Ac \rangle$ **for** x
proof (rule *ccontr*)
assume $\langle \nexists U. \text{openin } T \ U \wedge x \in U \wedge U \subseteq Ac \rangle$
then have $UA: \langle U \cap A \neq \{\} \rangle$ **if** $\langle \text{openin } T \ U \rangle$ **and** $\langle x \in U \rangle$ **for** U
by (*metis Ac-def Diff-mono Diff-triv openin-subset subset-refl that*)
have [*simp*]: $\langle x \in \text{topspace } T \rangle$
using *that* **by** (*simp add: Ac-def*)

define F **where** $\langle F = \text{nhdsin } T \ x \sqcap \text{principal } A \rangle$
have $\langle F \neq \perp \rangle$
apply (*subst filter-eq-iff*)
apply (*auto intro!: exI*[*of -* $\langle \lambda \cdot. \text{False} \rangle$]) *simp: F-def eventually-inf eventually-principal*

eventually-nhdsin
by (*meson UA disjoint-iff*)

define f **where** $\langle f\ y = (if\ y \in A\ then\ y\ else\ a) \rangle$ **for** y
with $\langle a \in A \rangle$ **have** $\langle range\ f \subseteq A \rangle$
by *force*

have $\langle \forall_F\ y\ in\ F.\ f\ y \in U \rangle$ **if** $\langle openin\ T\ U \rangle$ **and** $\langle x \in U \rangle$ **for** U
proof –
have $\langle eventually\ (\lambda x.\ x \in U)\ (nhdsin\ T\ x) \rangle$
using *eventually-nhdsin that by fastforce*
moreover have $\langle \exists R.\ (\forall x \in A.\ R\ x) \wedge (\forall x.\ x \in U \longrightarrow R\ x \longrightarrow f\ x \in U) \rangle$
apply (*rule exI[of - $\langle \lambda x.\ x \in A \rangle$]*)
by (*simp add: f-def*)
ultimately show *?thesis*
by (*auto simp add: F-def eventually-inf eventually-principal*)

qed
then have $\langle limitin\ T\ f\ x\ F \rangle$
unfolding *limitin-def* **by** *simp*
with $\langle F \neq \perp \rangle$ $\langle range\ f \subseteq A \rangle$ $x \in A$
have $\langle x \in A \rangle$
by *simp*
with that show *False*
by (*simp add: Ac-def*)

qed
then have $\langle openin\ T\ Ac \rangle$
apply (*rule-tac openin-subopen[THEN iffD2]*)
by *simp*
then show *?thesis*
by (*simp add: Ac-def AT closedin-def*)

qed

lemma *cmod-mono*: $\langle 0 \leq a \implies a \leq b \implies cmod\ a \leq cmod\ b \rangle$
by (*simp add: cmod-Re less-eq-complex-def*)

lemma *has-sum-mono-neutral-complex*:
fixes $f :: 'a \Rightarrow complex$
assumes $\langle f\ has-sum\ a \rangle$ **and** $\langle g\ has-sum\ b \rangle$
assumes $\langle \bigwedge x.\ x \in A \cap B \implies f\ x \leq g\ x \rangle$
assumes $\langle \bigwedge x.\ x \in A - B \implies f\ x \leq 0 \rangle$
assumes $\langle \bigwedge x.\ x \in B - A \implies g\ x \geq 0 \rangle$
shows $a \leq b$

proof –
have $\langle ((\lambda x.\ Re\ (f\ x))\ has-sum\ Re\ a)\ A \rangle$
using *assms(1) has-sum-Re has-sum-cong* **by** *blast*
moreover have $\langle ((\lambda x.\ Re\ (g\ x))\ has-sum\ Re\ b)\ B \rangle$
using *assms(2) has-sum-Re has-sum-cong* **by** *blast*
ultimately have $Re:\ \langle Re\ a \leq Re\ b \rangle$
apply (*rule has-sum-mono-neutral*)

using *assms(3-5)* **by** (*simp-all add: less-eq-complex-def*)
have $\langle (\lambda x. \text{Im } (f x)) \text{ has-sum Im } a \rangle A \rangle$
using *assms(1) has-sum-Im has-sum-cong* **by** *blast*
then have $\langle (\lambda x. \text{Im } (f x)) \text{ has-sum Im } a \rangle (A \cap B) \rangle$
apply (*rule has-sum-cong-neutral[THEN iffD1, rotated -1]*)
using *assms(3-5)* **by** (*auto simp add: less-eq-complex-def*)
moreover have $\langle (\lambda x. \text{Im } (g x)) \text{ has-sum Im } b \rangle B \rangle$
using *assms(2) has-sum-Im has-sum-cong* **by** *blast*
then have $\langle (\lambda x. \text{Im } (f x)) \text{ has-sum Im } b \rangle (A \cap B) \rangle$
apply (*rule has-sum-cong-neutral[THEN iffD1, rotated -1]*)
using *assms(3-5)* **by** (*auto simp add: less-eq-complex-def*)
ultimately have *Im*: $\langle \text{Im } a = \text{Im } b \rangle$
by (*rule has-sum-unique*)
from *Re Im show ?thesis*
using *less-eq-complexI* **by** *blast*
qed

lemma *choice2*: $\langle \exists f. (\forall x. Q1 x (f x)) \wedge (\forall x. Q2 x (f x)) \rangle$
if $\langle \forall x. \exists y. Q1 x y \wedge Q2 x y \rangle$
by (*meson that*)

lemma *choice3*: $\langle \exists f. (\forall x. Q1 x (f x)) \wedge (\forall x. Q2 x (f x)) \wedge (\forall x. Q3 x (f x)) \rangle$
if $\langle \forall x. \exists y. Q1 x y \wedge Q2 x y \wedge Q3 x y \rangle$
by (*meson that*)

lemma *choice4*: $\langle \exists f. (\forall x. Q1 x (f x)) \wedge (\forall x. Q2 x (f x)) \wedge (\forall x. Q3 x (f x)) \wedge (\forall x. Q4 x (f x)) \rangle$
if $\langle \forall x. \exists y. Q1 x y \wedge Q2 x y \wedge Q3 x y \wedge Q4 x y \rangle$
by (*meson that*)

lemma *choice5*: $\langle \exists f. (\forall x. Q1 x (f x)) \wedge (\forall x. Q2 x (f x)) \wedge (\forall x. Q3 x (f x)) \wedge (\forall x. Q4 x (f x)) \wedge (\forall x. Q5 x (f x)) \rangle$
if $\langle \forall x. \exists y. Q1 x y \wedge Q2 x y \wedge Q3 x y \wedge Q4 x y \wedge Q5 x y \rangle$
apply (*simp only: flip: all-conj-distrib*)
using *that* **by** (*rule choice*)

definition (*in order*) $\langle \text{is-Sup } X s \longleftrightarrow (\forall x \in X. x \leq s) \wedge (\forall y. (\forall x \in X. x \leq y) \longrightarrow s \leq y) \rangle$
definition (*in order*) $\langle \text{has-Sup } X \longleftrightarrow (\exists s. \text{is-Sup } X s) \rangle$

lemma (*in order*) *is-SupI*:
assumes $\langle \bigwedge x. x \in X \implies x \leq s \rangle$
assumes $\langle \bigwedge y. (\bigwedge x. x \in X \implies x \leq y) \implies s \leq y \rangle$
shows $\langle \text{is-Sup } X s \rangle$
using *assms* **by** (*auto simp add: is-Sup-def*)

lemma *is-Sup-unique*: $\langle \text{is-Sup } X a \implies \text{is-Sup } X b \implies a=b \rangle$
by (*simp add: Orderings.order-eq-iff is-Sup-def*)

lemma *has-Sup-bdd-above*: $\langle \text{has-Sup } X \implies \text{bdd-above } X \rangle$

by (metis bdd-above.unfold has-Sup-def is-Sup-def)

lemma *is-Sup-has-Sup*: $\langle is-Sup\ X\ s \implies has-Sup\ X \rangle$
using *has-Sup-def* **by** *blast*

class *Sup-order* = *order* + *Sup* + *sup* +
assumes *is-Sup-Sup*: $\langle has-Sup\ X \implies is-Sup\ X\ (Sup\ X) \rangle$
assumes *is-Sup-sup*: $\langle has-Sup\ \{x,y\} \implies is-Sup\ \{x,y\}\ (sup\ x\ y) \rangle$

lemma (in *Sup-order*) *is-Sup-eq-Sup*:
assumes $\langle is-Sup\ X\ s \rangle$
shows $\langle s = Sup\ X \rangle$
by (meson *assms local.dual-order.antisym local.has-Sup-def local.is-Sup-Sup local.is-Sup-def*)

lemma *is-Sup-cSup*:
fixes $X :: \langle 'a::conditionally-complete-lattice\ set \rangle$
assumes $\langle bdd-above\ X \rangle$ **and** $\langle X \neq \{\} \rangle$
shows $\langle is-Sup\ X\ (Sup\ X) \rangle$
using *assms* **by** (auto *intro!*: *cSup-upper cSup-least simp: is-Sup-def*)

lemma *continuous-map-iff-preserves-convergence*:
assumes $\langle \bigwedge F\ a.\ a \in\ topspace\ T \implies limitin\ T\ id\ a\ F \implies limitin\ U\ f\ (f\ a)\ F \rangle$
shows $\langle continuous-map\ T\ U\ f \rangle$
apply (rule *continuous-map-atin*[*THEN iffD2*], *intro ballI*)
using *assms*
by (*simp add: limitin-continuous-map*)

lemma *SMT-choices*:

— Was included as SMT.choices in Isabelle and disappeared

$\bigwedge Q.\ \forall x.\ \exists y\ ya.\ Q\ x\ y\ ya \implies \exists f\ fa.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)$
 $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb.\ Q\ x\ y\ ya\ yb \implies \exists f\ fa\ fb.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)$
 $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc.\ Q\ x\ y\ ya\ yb\ yc \implies \exists f\ fa\ fb\ fc.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)$
 $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd.\ Q\ x\ y\ ya\ yb\ yc\ yd \implies$
 $\quad \exists f\ fa\ fb\ fc\ fd.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)$
 $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd\ ye.\ Q\ x\ y\ ya\ yb\ yc\ yd\ ye \implies$
 $\quad \exists f\ fa\ fb\ fc\ fd\ fe.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)\ (fe\ x)$
 $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd\ ye\ yf.\ Q\ x\ y\ ya\ yb\ yc\ yd\ ye\ yf \implies$
 $\quad \exists f\ fa\ fb\ fc\ fd\ fe\ ff.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)\ (fe\ x)\ (ff\ x)$
 $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd\ ye\ yf\ yg.\ Q\ x\ y\ ya\ yb\ yc\ yd\ ye\ yf\ yg \implies$
 $\quad \exists f\ fa\ fb\ fc\ fd\ fe\ ff\ fg.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)\ (fe\ x)\ (ff\ x)\ (fg\ x)$
by *metis+*

lemma *closedin-pullback-topology*:

closedin (pullback-topology $A\ f\ T$) $S \iff (\exists C.\ closedin\ T\ C \wedge S = f^{-1}C \cap A)$

proof (rule *iffI*)

define $TT\ PT$ **where** $\langle TT = topspace\ T \rangle$ **and** $\langle PT = topspace\ (pullback-topology\ A\ f\ T) \rangle$

assume *closed*: $\langle closedin\ (pullback-topology\ A\ f\ T)\ S \rangle$

then have $\langle S \subseteq PT \rangle$
using *PT-def closedin-subset* **by** *blast*
from *closed* **have** $\langle \text{openin } (\text{pullback-topology } A \ f \ T) \ (PT - S) \rangle$
by *(auto intro!: simp: closedin-def PT-def)*
then obtain U **where** $\langle \text{openin } T \ U \rangle$ **and** $S\text{-fUA}$: $\langle PT - S = f -' U \cap A \rangle$
by *(auto simp: openin-pullback-topology)*
define C **where** $\langle C = TT - U \rangle$
have $\langle \text{closedin } T \ C \rangle$
using *C-def TT-def* $\langle \text{openin } T \ U \rangle$ **by** *blast*
moreover have $\langle S = f -' C \cap A \rangle$
using *S-fUA* $\langle S \subseteq PT \rangle$
apply *(simp only: C-def PT-def TT-def)*
by *(metis Diff-Diff-Int Diff-Int-distrib2 inf.absorb-iff2 topspace-pullback-topology vimage-Diff)*
ultimately show $\langle \exists C. \text{closedin } T \ C \wedge S = f -' C \cap A \rangle$
by *auto*
next
assume $\langle \exists U. \text{closedin } T \ U \wedge S = f -' U \cap A \rangle$
then show $\langle \text{closedin } (\text{pullback-topology } A \ f \ T) \ S \rangle$
apply *(simp add: openin-pullback-topology closedin-def topspace-pullback-topology)*
by *blast*
qed

lemma *regular-space-pullback[intro]*:
assumes $\langle \text{regular-space } T \rangle$
shows $\langle \text{regular-space } (\text{pullback-topology } A \ f \ T) \rangle$
proof *(unfold regular-space-def, intro allI impI)*
define $TT \ PT$ **where** $\langle TT = \text{topspace } T \rangle$ **and** $\langle PT = \text{topspace } (\text{pullback-topology } A \ f \ T) \rangle$
fix $S \ y$
assume *asm*: $\langle \text{closedin } (\text{pullback-topology } A \ f \ T) \ S \wedge y \in PT - S \rangle$
from *asm* **obtain** C **where** $\langle \text{closedin } T \ C \rangle$ **and** $S\text{-fCA}$: $\langle S = f -' C \cap A \rangle$
by *(auto simp: closedin-pullback-topology)*
from *asm S-fCA*
have $\langle f \ y \in TT - C \rangle$
by *(auto simp: PT-def TT-def topspace-pullback-topology)*
then obtain $U' \ V'$ **where** $\langle \text{openin } T \ U' \rangle$ **and** $\langle \text{openin } T \ V' \rangle$ **and** $\langle f \ y \in U' \rangle$ **and** $\langle C \subseteq V' \rangle$
and $\langle U' \cap V' = \{\} \rangle$
by *(metis TT-def* $\langle \text{closedin } T \ C \rangle$ *assms regular-space-def disjnt-def)*
define $U \ V$ **where** $\langle U = f -' U' \cap A \rangle$ **and** $\langle V = f -' V' \cap A \rangle$
have $\langle \text{openin } (\text{pullback-topology } A \ f \ T) \ U \rangle$
using *U-def* $\langle \text{openin } T \ U' \rangle$ *openin-pullback-topology* **by** *blast*
moreover have $\langle \text{openin } (\text{pullback-topology } A \ f \ T) \ V \rangle$
using *V-def* $\langle \text{openin } T \ V' \rangle$ *openin-pullback-topology* **by** *blast*
moreover have $\langle y \in U \rangle$
by *(metis DiffD1 Int-iff PT-def U-def* $\langle f \ y \in U' \rangle$ *asm topspace-pullback-topology vimageI)*
moreover have $\langle S \subseteq V \rangle$
using *S-fCA V-def* $\langle C \subseteq V' \rangle$ **by** *blast*
moreover have $\langle \text{disjnt } U \ V \rangle$
using *U-def V-def* $\langle U' \cap V' = \{\} \rangle$ *disjnt-def* **by** *blast*

ultimately show $\langle \exists U V. \text{openin} (\text{pullback-topology } A f T) U \wedge \text{openin} (\text{pullback-topology } A f T) V \wedge y \in U \wedge S \subseteq V \wedge \text{disjnt } U V \rangle$
apply (*rule-tac exI[of - U], rule-tac exI[of - V]*)
by auto
qed

lemma *t3-space-euclidean-regular[iff]*: $\langle \text{regular-space} (\text{euclidean} :: 'a::t3\text{-space topology}) \rangle$
using *t3-space*
apply (*simp add: regular-space-def disjnt-def*)
by fast

definition *increasing-filter* :: $\langle 'a::\text{order filter} \Rightarrow \text{bool} \rangle$ **where**
— Definition suggested by [5]
 $\langle \text{increasing-filter } F \longleftrightarrow (\forall_F x \text{ in } F. \forall_F y \text{ in } F. y \geq x) \rangle$

lemma *increasing-filtermap*:
fixes $F :: \langle 'a::\text{order filter} \rangle$ **and** $f :: \langle 'a \Rightarrow 'b::\text{order} \rangle$ **and** $X :: \langle 'a \text{ set} \rangle$
assumes *increasing*: $\langle \text{increasing-filter } F \rangle$
assumes *mono*: $\langle \text{mono-on } X f \rangle$
assumes *ev-X*: $\langle \text{eventually } (\lambda x. x \in X) F \rangle$
shows $\langle \text{increasing-filter} (\text{filtermap } f F) \rangle$
proof —
from *increasing*
have *incr*: $\langle \forall_F x \text{ in } F. \forall_F y \text{ in } F. x \leq y \rangle$
apply (*simp add: increasing-filter-def*)
by —
have $\langle \forall_F x \text{ in } F. \forall_F y \text{ in } F. f x \leq f y \rangle$
proof (*rule eventually-elim2[OF ev-X incr]*)
fix x
assume $\langle x \in X \rangle$
assume $\langle \forall_F y \text{ in } F. x \leq y \rangle$
then show $\langle \forall_F y \text{ in } F. f x \leq f y \rangle$
proof (*rule eventually-elim2[OF ev-X]*)
fix y **assume** $\langle y \in X \rangle$ **and** $\langle x \leq y \rangle$
with $\langle x \in X \rangle$ **show** $\langle f x \leq f y \rangle$
using *mono* **by** (*simp add: mono-on-def*)
qed
qed
then show $\langle \text{increasing-filter} (\text{filtermap } f F) \rangle$
by (*simp add: increasing-filter-def eventually-filtermap*)
qed

lemma *increasing-finite-subsets-at-top[simp]*: $\langle \text{increasing-filter} (\text{finite-subsets-at-top } X) \rangle$
apply (*simp add: increasing-filter-def eventually-finite-subsets-at-top*)
by force

lemma *monotone-convergence*:
— Following [5]

```

fixes  $f :: \langle 'b \Rightarrow 'a :: \{order-topology, conditionally-complete-linorder\} \rangle$ 
assumes  $bounded: \langle \forall_F x \text{ in } F. f x \leq B \rangle$ 
assumes  $increasing: \langle increasing-filter (filtermap f F) \rangle$ 
shows  $\langle \exists l. (f \longrightarrow l) F \rangle$ 
proof (cases  $\langle F \neq \perp \rangle$ )
  case True
    note [simp] = True
    define  $S l$  where  $\langle S x \longleftrightarrow (\forall_F y \text{ in } F. f y \geq x) \wedge x \leq B \rangle$ 
      and  $\langle l = Sup (Collect S) \rangle$  for  $x$ 
    from bounded increasing
    have  $ev-S: \langle eventually S (filtermap f F) \rangle$ 
      by (auto intro!: eventually-conj simp: S-def[abs-def] increasing-filter-def eventually-filtermap)
    have  $bdd-S: \langle bdd-above (Collect S) \rangle$ 
      by (auto simp: S-def)
    have  $S\text{-nonempty}: \langle Collect S \neq \{\} \rangle$ 
      using ev-S
      by (metis Collect-empty-eq-bot Set.empty-def True eventually-False filtermap-bot-iff)
    have  $\langle (f \longrightarrow l) F \rangle$ 
  proof (rule order-tendstoI; rename-tac x)
    fix  $x$ 
    assume  $\langle x < l \rangle$ 
    then obtain  $s$  where  $\langle S s \rangle$  and  $\langle x < s \rangle$ 
      using less-cSupD[OF S-nonempty] l-def
      by blast
    then
      show  $\langle \forall_F y \text{ in } F. x < f y \rangle$ 
      using S-def basic-trans-rules(22) eventually-mono by force
  next
    fix  $x$ 
    assume  $asm: \langle l < x \rangle$ 
    from ev-S
    show  $\langle \forall_F y \text{ in } F. f y < x \rangle$ 
      unfolding eventually-filtermap
      apply (rule eventually-mono)
      using asm
      by (metis bdd-S cSup-upper dual-order.strict-trans2 l-def mem-Collect-eq)
  qed
  then show  $\langle \exists l. (f \longrightarrow l) F \rangle$ 
    by (auto intro!: exI[of - l] simp: filterlim-def)
next
  case False
  then show  $\langle \exists l. (f \longrightarrow l) F \rangle$ 
    by (auto intro!: exI)
qed

```

```

lemma monotone-convergence-complex:
  fixes  $f :: \langle 'b \Rightarrow complex \rangle$ 
  assumes  $bounded: \langle \forall_F x \text{ in } F. f x \leq B \rangle$ 

```

```

assumes increasing:  $\langle \text{increasing-filter } (\text{filtermap } f \ F) \rangle$ 
shows  $\langle \exists l. (f \longrightarrow l) \ F \rangle$ 
proof –
  have inc-re:  $\langle \text{increasing-filter } (\text{filtermap } (\lambda x. \text{Re } (f \ x)) \ F) \rangle$ 
    using increasing-filtermap[OF increasing, where f=Re and X=UNIV]
    by (simp add: less-eq-complex-def[abs-def] mono-def monotone-def filtermap-filtermap)
  from bounded have  $\langle \forall_F \ x \ \text{in } \ F. \ \text{Re } (f \ x) \leq \text{Re } B \rangle$ 
    using eventually-mono less-eq-complex-def by fastforce
  from monotone-convergence[OF this inc-re]
  obtain re where lim-re:  $\langle ((\lambda x. \text{Re } (f \ x)) \longrightarrow \text{re}) \ F \rangle$ 
    by auto
  from bounded have  $\langle \forall_F \ x \ \text{in } \ F. \ \text{Im } (f \ x) = \text{Im } B \rangle$ 
    by (simp add: less-eq-complex-def[abs-def] eventually-mono)
  then have lim-im:  $\langle ((\lambda x. \text{Im } (f \ x)) \longrightarrow \text{Im } B) \ F \rangle$ 
    by (simp add: tendsto-eventually)
  from lim-re lim-im have  $\langle (f \longrightarrow \text{Complex } \text{re } (\text{Im } B)) \ F \rangle$ 
    by (simp add: tendsto-complex-iff)
  then show ?thesis
    by auto
qed

lemma compact-closed-subset:
  assumes  $\langle \text{compact } s \rangle$ 
  assumes  $\langle \text{closed } t \rangle$ 
  assumes  $\langle t \subseteq s \rangle$ 
  shows  $\langle \text{compact } t \rangle$ 
  by (metis assms(1) assms(2) assms(3) compact-Int-closed inf.absorb-iff2)

definition separable where  $\langle \text{separable } S \longleftrightarrow (\exists B. \text{countable } B \wedge S \subseteq \text{closure } B) \rangle$ 

lemma compact-imp-separable:  $\langle \text{separable } S \rangle$  if  $\langle \text{compact } S \rangle$  for  $S :: \langle 'a :: \text{metric-space set} \rangle$ 
proof –
  from that
  obtain K where  $\langle \text{finite } (K \ n) \rangle$  and K-cover-S:  $\langle S \subseteq (\bigcup k \in K \ n. \text{ball } k \ (1 / \text{of-nat } (n+1))) \rangle$ 
for  $n :: \text{nat}$ 
  proof (atomize-elim, intro choice2 allI)
    fix  $n$ 
    have  $\langle S \subseteq (\bigcup k \in UNIV. \text{ball } k \ (1 / \text{of-nat } (n+1))) \rangle$ 
      apply (auto intro!: simp: )
      by (smt (verit, del-insts) dist-eq-0-iff linordered-field-class.divide-pos-pos of-nat-less-0-iff)
    then show  $\langle \exists K. \text{finite } K \wedge S \subseteq (\bigcup k \in K. \text{ball } k \ (1 / \text{real } (n + 1))) \rangle$ 
      apply (simp add: compact-eq-Heine-Borel)
      by (meson Elementary-Metric-Spaces.open-ball compactE-image compact S)
  qed
define B where  $\langle B = (\bigcup n. K \ n) \rangle$ 
have  $\langle \text{countable } B \rangle$ 
  using B-def  $\langle \text{finite } (K \ -) \rangle$  uncountable-infinite by blast
have  $\langle S \subseteq \text{closure } B \rangle$ 
proof (intro subsetI closure-approachable[THEN iffD2, rule-format])

```

```

fix x assume ⟨x ∈ S⟩
fix e :: real assume ⟨e > 0⟩
define n :: nat where ⟨n = nat (ceiling (1/e))⟩
with ⟨e > 0⟩ have ne: ⟨1 / real (n+1) ≤ e⟩
proof -
  have ⟨1 / real (n+1) ≤ 1 / ceiling (1/e)⟩
    by (simp add: ⟨0 < e⟩ linordered-field-class.frac-le n-def)
  also have ⟨... ≤ 1 / (1/e)⟩
    by (smt (verit, del-insts) ⟨0 < e⟩ le-of-int-ceiling linordered-field-class.divide-pos-pos
linordered-field-class.frac-le)
  also have ⟨... = e⟩
    by simp
  finally show ?thesis
    by -
qed
have ⟨S ⊆ (⋃ k∈K n. ball k (1 / of-nat (n+1)))⟩
  using K-cover-S by presburger
then obtain k where ⟨k ∈ K n⟩ and x-ball: ⟨x ∈ ball k (1 / of-nat (n+1))⟩
  using ⟨x ∈ S⟩ by auto
from ⟨k ∈ K n⟩ have ⟨k ∈ B⟩
  using B-def by blast
moreover from x-ball have ⟨dist k x < e⟩
  by (smt (verit) ne mem-ball)
ultimately show ⟨∃ k∈B. dist k x < e⟩
  by fast
qed
show ⟨separable S⟩
  using ⟨S ⊆ closure B⟩ ⟨countable B⟩ separable-def by blast
qed

```

```

lemma ex-norm1-not-singleton:
  shows ⟨∃ x::'a::{real-normed-vector, not-singleton}. norm x = 1⟩
  apply (rule ex-norm1)
  by simp

```

```

lemma is-Sup-approx-below:
  fixes b :: ⟨'a::linordered-ab-group-add⟩
  assumes ⟨is-Sup A b⟩
  assumes ⟨ε > 0⟩
  shows ⟨∃ x∈A. b - ε ≤ x⟩
proof (rule ccontr)
  assume ⟨¬ (∃ x∈A. b - ε ≤ x)⟩
  then have ⟨b - ε ≥ x⟩ if ⟨x ∈ A⟩ for x
    using that by auto
  with assms
  have ⟨b ≤ b - ε⟩
    by (simp add: is-Sup-def)
  with ⟨ε > 0⟩
  show False

```

by *simp*
qed

lemma *sums-le-complex*:

fixes $f g :: \text{nat} \Rightarrow \text{complex}$
assumes $\langle \bigwedge n. f\ n \leq g\ n \rangle$
assumes $\langle f\ \text{sums}\ s \rangle$
assumes $\langle g\ \text{sums}\ t \rangle$
shows $\langle s \leq t \rangle$

proof –

have $\langle \text{Re}\ (f\ n) \leq \text{Re}\ (g\ n) \rangle$ **for** n
 by (*simp add: Re-mono assms(1)*)
moreover have $\langle (\lambda n. \text{Re}\ (f\ n))\ \text{sums}\ \text{Re}\ s \rangle$
 using *assms(2) sums-Re* **by** *auto*
moreover have $\langle (\lambda n. \text{Re}\ (g\ n))\ \text{sums}\ \text{Re}\ t \rangle$
 using *assms(3) sums-Re* **by** *auto*
ultimately have *re*: $\langle \text{Re}\ s \leq \text{Re}\ t \rangle$
 by (*rule sums-le*)
have $\langle \text{Im}\ (f\ n) = \text{Im}\ (g\ n) \rangle$ **for** n
 by (*simp add: assms(1) comp-Im-same*)
moreover have $\langle (\lambda n. \text{Im}\ (f\ n))\ \text{sums}\ \text{Im}\ s \rangle$
 using *assms(2) sums-Im* **by** *auto*
moreover have $\langle (\lambda n. \text{Im}\ (g\ n))\ \text{sums}\ \text{Im}\ t \rangle$
 using *assms(3) sums-Im* **by** *auto*
ultimately have *im*: $\langle \text{Im}\ s = \text{Im}\ t \rangle$
 using *sums-unique2* **by** *auto*
from *re im* **show** $\langle s \leq t \rangle$
 using *less-eq-complexI* **by** *auto*

qed

lemma *infsum-single*:

assumes $\bigwedge j. j \neq i \implies j \in A \implies f\ j = 0$
shows $\text{infsum}\ f\ A = (\text{if}\ i \in A\ \text{then}\ f\ i\ \text{else}\ 0)$
apply (*subst infsum-cong-neutral*[**where** $T = \langle A \cap \{i\} \rangle$ **and** $g = f$])
using *assms* **by** *auto*

lemma *suminf-eqI*:

fixes $x :: \langle - :: \{ \text{comm-monoid-add}, t2\text{-space} \} \rangle$
assumes $\langle f\ \text{sums}\ x \rangle$
shows $\langle \text{suminf}\ f = x \rangle$
using *assms*
by (*auto intro!: simp: sums-iff*)

lemma *suminf-If-finite-set*:

fixes $f :: \langle - \Rightarrow - :: \{ \text{comm-monoid-add}, t2\text{-space} \} \rangle$
assumes $\langle \text{finite}\ F \rangle$
shows $\langle (\sum x \in F. f\ x) = (\sum x. \text{if}\ x \in F\ \text{then}\ f\ x\ \text{else}\ 0) \rangle$
by (*auto intro!: suminf-eqI*[*symmetric*] *sums-If-finite-set simp: assms*)

definition *separating-set* :: $\langle ('a \Rightarrow 'b) \Rightarrow \text{bool} \Rightarrow 'a \text{ set} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{separating-set } P \ S \ \longleftrightarrow \ (\forall f \ g. \ P \ f \ \longrightarrow \ P \ g \ \longrightarrow \ (\forall x \in S. \ f \ x = g \ x) \ \longrightarrow \ f = g) \rangle$

lemma *separating-set-mono*: $\langle S \subseteq T \Longrightarrow \text{separating-set } P \ S \Longrightarrow \text{separating-set } P \ T \rangle$
unfolding *separating-set-def* **by** *fast*

lemma *separating-set-UNIV*[*simp*]: $\langle \text{separating-set } P \ \text{UNIV} \rangle$
by (*auto intro!*: *ext simp: separating-set-def*)

lemma *eq-from-separatingI*:
assumes $\langle \text{separating-set } P \ S \rangle$
assumes $\langle P \ f \rangle$ **and** $\langle P \ g \rangle$
assumes $\langle \bigwedge x. x \in S \Longrightarrow f \ x = g \ x \rangle$
shows $\langle f = g \rangle$
using *assms* **by** (*simp add: separating-set-def*)

lemma *eq-from-separatingI2*:
assumes $\langle \text{separating-set } P \ ((\lambda(x,y). \ h \ x \ y) \ ` \ (S \times T)) \rangle$
assumes $\langle P \ f \rangle$ **and** $\langle P \ g \rangle$
assumes $\langle \bigwedge x \ y. x \in S \Longrightarrow y \in T \Longrightarrow f \ (h \ x \ y) = g \ (h \ x \ y) \rangle$
shows $\langle f = g \rangle$
apply (*rule eq-from-separatingI*[*OF assms(1)*])
using *assms(2-4)* **by** *auto*

lemma *separating-setI*:
assumes $\langle \bigwedge f \ g. \ P \ f \Longrightarrow P \ g \Longrightarrow (\bigwedge x. x \in S \Longrightarrow f \ x = g \ x) \Longrightarrow f = g \rangle$
shows $\langle \text{separating-set } P \ S \rangle$
by (*simp add: assms separating-set-def*)

lemma *tendsto-le-complex*:
fixes $x \ y :: \text{complex}$
assumes $F: \neg \text{trivial-limit } F$
and $x: (f \ \longrightarrow \ x) \ F$
and $y: (g \ \longrightarrow \ y) \ F$
and $ev: \text{eventually } (\lambda x. \ g \ x \leq f \ x) \ F$
shows $y \leq x$

proof (*rule less-eq-complexI*)
note F
moreover have $\langle ((\lambda x. \ \text{Re } (f \ x)) \ \longrightarrow \ \text{Re } x) \ F \rangle$
by (*simp add: assms tendsto-Re*)
moreover have $\langle ((\lambda x. \ \text{Re } (g \ x)) \ \longrightarrow \ \text{Re } y) \ F \rangle$
by (*simp add: assms tendsto-Re*)
moreover from ev **have** $\text{eventually } (\lambda x. \ \text{Re } (g \ x) \leq \text{Re } (f \ x)) \ F$
apply (*rule eventually-mono*)
by (*simp add: less-eq-complex-def*)
ultimately show $\langle \text{Re } y \leq \text{Re } x \rangle$
by (*rule tendsto-le*)

next

```

note  $F$ 
moreover have  $\langle (\lambda x. \text{Im } (g x)) \longrightarrow \text{Im } y \rangle F \rangle$ 
  by (simp add: assms tendsto-Im)
moreover
have  $\langle (\lambda x. \text{Im } (g x)) \longrightarrow \text{Im } x \rangle F \rangle$ 
proof –
  have  $\langle (\lambda x. \text{Im } (f x)) \longrightarrow \text{Im } x \rangle F \rangle$ 
    by (simp add: assms tendsto-Im)
  moreover from ev have  $\langle \forall_F x \text{ in } F. \text{Im } (f x) = \text{Im } (g x) \rangle$ 
    apply (rule eventually-mono)
    by (simp add: less-eq-complex-def)
  ultimately show ?thesis
    by (rule Lim-transform-eventually)
qed
ultimately show  $\langle \text{Im } y = \text{Im } x \rangle$ 
  by (rule tendsto-unique)
qed

lemma bdd-above-mono2:
  assumes  $\langle \text{bdd-above } (g \text{ ' } B) \rangle$ 
  assumes  $\langle A \subseteq B \rangle$ 
  assumes  $\langle \bigwedge x. x \in A \implies f x \leq g x \rangle$ 
  shows  $\langle \text{bdd-above } (f \text{ ' } A) \rangle$ 
by (smt (verit, del-Insts) Set.basic-monos(7) assms(1) assms(2) assms(3) basic-trans-rules(23))
bdd-above.I2 bdd-above.unfold imageI

```

end

2 *Misc-Tensor-Product-BO* – Miscellaneous results missing from `Complex_Bounded_Operators`

```

theory Misc-Tensor-Product-BO
imports
  Complex-Bounded-Operators.Complex-L2
  Misc-Tensor-Product
  HOL-Library.Function-Algebras
begin

no-notation Set-Algebras.elt-set-eq (infix  $=_o$  50)

unbundle cblinfun-syntax

instance cblinfun :: (chilbert-space, chilbert-space) ordered-comm-monoid-add
  by intro-classes

lemma rank1-scaleR[simp]:  $\langle \text{rank1 } (c *_R a) \rangle$  if  $\langle \text{rank1 } a \rangle$  and  $\langle c \neq 0 \rangle$ 

```

by (simp add: rank1-scaleC scaleR-scaleC that(1) that(2))

lemma rank1-butterfly[simp]: $\langle \text{rank1 } (\text{butterfly } x \ y) \rangle$
 apply (cases $\langle y = 0 \rangle$)
 by (auto intro: exI[of - 0] simp: rank1-def butterfly-is-rank1)

definition $\langle \text{cfinite-dim } S \longleftrightarrow (\exists B. \text{finite } B \wedge S \subseteq \text{cspan } B) \rangle$

lemma cfinite-dim-subspace-has-basis:
 assumes $\langle \text{cfinite-dim } S \rangle$ and $\langle \text{csubspace } S \rangle$
 shows $\langle \exists B. \text{finite } B \wedge \text{cindependent } B \wedge \text{cspan } B = S \rangle$
proof –
 obtain B where $\langle \text{cindependent } B \rangle$ and $\langle \text{cspan } B = S \rangle$
 by (rule complex-vector.maximal-independent-subset[where $V=S$])
 (use $\langle \text{csubspace } S \rangle$ complex-vector.span-subspace in blast)
 from $\langle \text{cfinite-dim } S \rangle$
 obtain C where $\langle \text{finite } C \rangle$ and $\langle S \subseteq \text{cspan } C \rangle$
 using cfinite-dim-def by auto
 from $\langle \text{cspan } B = S \rangle$ and $\langle S \subseteq \text{cspan } C \rangle$
 have $\langle B \subseteq \text{cspan } C \rangle$
 using complex-vector.span-superset by force
 from $\langle \text{finite } C \rangle$ $\langle \text{cindependent } B \rangle$ this
 have $\langle \text{finite } B \rangle$
 by (rule complex-vector.independent-span-bound[THEN conjunct1])
 from this and $\langle \text{cindependent } B \rangle$ and $\langle \text{cspan } B = S \rangle$
 show ?thesis
 by auto
qed

lemma cfinite-dim-subspace-has-onb:
 assumes $\langle \text{cfinite-dim } S \rangle$ and $\langle \text{csubspace } S \rangle$
 shows $\langle \exists B. \text{finite } B \wedge \text{is-ortho-set } B \wedge \text{cspan } B = S \wedge (\forall x \in B. \text{norm } x = 1) \rangle$
proof –
 from assms
 obtain C where $\langle \text{finite } C \rangle$ and $\langle \text{cindependent } C \rangle$ and $\langle \text{cspan } C = S \rangle$
 using cfinite-dim-subspace-has-basis by blast
 obtain B where $\langle \text{finite } B \rangle$ and $\langle \text{is-ortho-set } B \rangle$ and $\langle \text{cspan } B = \text{cspan } C \rangle$
 and norm: $\langle x \in B \implies \text{norm } x = 1 \rangle$ for x
 using orthonormal-basis-of-cspan[OF $\langle \text{finite } C \rangle$]
 by blast
 with $\langle \text{cspan } C = S \rangle$ have $\langle \text{cspan } B = S \rangle$
 by simp
 with $\langle \text{finite } B \rangle$ and $\langle \text{is-ortho-set } B \rangle$ and norm
 show ?thesis
 by blast
qed

lemma cspan-finite-dim[intro]: $\langle \text{cfinite-dim } (\text{cspan } B) \rangle$ if $\langle \text{finite } B \rangle$
 using cfinite-dim-def that by auto

lift-definition *finite-dim-ccsubspace* :: $\langle 'a::\text{complex-normed-vector ccsubspace} \Rightarrow \text{bool} \rangle$ **is** *cfi-finite-dim*.

lemma *ccspan-finite-dim[intro]*: $\langle \text{finite-dim-ccsubspace} (\text{ccspan } B) \rangle$ **if** $\langle \text{finite } B \rangle$
using *ccspan-finite finite-dim-ccsubspace.rep-eq* **that** **by** *fastforce*

lemma *finite-dim-ccsubspace-zero[iff]*: $\langle \text{finite-dim-ccsubspace } 0 \rangle$

proof –

have *: $\langle \text{cfinite-dim} (\text{cspan } \{0\}) \rangle$

by *blast*

show *?thesis*

apply *transfer*

using * **by** *simp*

qed

lemma *finite-dim-ccsubspace-bot[iff]*: $\langle \text{finite-dim-ccsubspace } \perp \rangle$

using *finite-dim-ccsubspace-zero* **by** *auto*

lemma *compact-scaleC*:

fixes *s* :: $'a::\text{complex-normed-vector set}$

assumes *compact s*

shows *compact (scaleC c ' s)*

by (*auto intro!: compact-continuous-image assms continuous-at-imp-continuous-on*)

lemma *Proj-nearest*:

assumes $\langle x \in \text{space-as-set } S \rangle$

shows $\langle \text{dist} (\text{Proj } S m) m \leq \text{dist } x m \rangle$

proof –

have $\langle \text{is-projection-on} (\text{Proj } S) (\text{space-as-set } S) \rangle$

by (*simp add: Proj.rep-eq*)

then have $\langle \text{is-arg-min} (\lambda x. \text{dist } x m) (\lambda x. x \in \text{space-as-set } S) (\text{Proj } S m) \rangle$

by (*simp add: is-projection-on-def*)

with *assms* **show** *?thesis*

by (*auto simp: is-arg-min-def*)

qed

lemma *norm-cblinfun-bound-unit*:

assumes $\langle b \geq 0 \rangle$

assumes $\langle \bigwedge \psi. \text{norm } \psi = 1 \implies \text{norm} (a *_V \psi) \leq b \rangle$

shows $\langle \text{norm } a \leq b \rangle$

proof (*rule norm-cblinfun-bound*)

from *assms* **show** $\langle b \geq 0 \rangle$ **by** *simp*

fix *x*

```

show ⟨norm (a *V x) ≤ b * norm x⟩
proof (cases ⟨x = 0⟩)
  case True
  then show ?thesis by simp
next
case False
have ⟨norm (a *V x) = norm (a *V (norm x *C sgn x))⟩
  by simp
also have ⟨... = norm (a *V sgn x) * norm x⟩
  by (simp add: cblinfun.scaleC-right del: norm-scaleC-sgn)
also have ⟨... ≤ (b * norm (sgn x)) * norm x⟩
  by (simp add: assms(2) norm-sgn)
also have ⟨... = b * norm x⟩
  by (simp add: norm-sgn)
finally show ?thesis
  by -
qed
qed

```

lemma *cblinfun-norm-is-Sup-cinner*:

— [2], Proposition II.2.13

fixes $A :: \langle 'a :: \{not-singleton, chilbert-space\} \Rightarrow_{CL} 'a \rangle$

assumes *Aselfadj*: ⟨*selfadjoint* A ⟩

shows ⟨*is-Sup* (($\lambda \psi. cmod (\psi \cdot_C (A *_V \psi))$) ' { $\psi. norm \psi = 1$ }) (*norm* A)⟩

proof (*rule is-SupI*)

fix b **assume** ⟨ $b \in (\lambda \psi. cmod (\psi \cdot_C (A *_V \psi)))$ ' { $\psi. norm \psi = 1$ }⟩

then obtain ψ **where** ⟨*norm* $\psi = 1$ ⟩ **and** $b\text{-}\psi$: ⟨ $b = cmod (\psi \cdot_C (A *_V \psi))$ ⟩

by *blast*

have ⟨ $b \leq norm (A \psi)$ ⟩

using $b\text{-}\psi$ ⟨*norm* $\psi = 1$ ⟩

by (*metis complex-inner-class.Cauchy-Schwarz-ineq2 mult-cancel-right2*)

also have ⟨... ≤ *norm* A ⟩

using ⟨*norm* $\psi = 1$ ⟩

by (*metis mult-cancel-left2 norm-cblinfun*)

finally show ⟨ $b \leq norm A$ ⟩

by —

next

fix c **assume** *asm*: ⟨($\bigwedge b. b \in (\lambda \psi. cmod (\psi \cdot_C A \psi))$) ' { $\psi. norm \psi = 1$ } $\implies b \leq c$ ⟩

have *c-upper*: ⟨*cmod* ($\psi \cdot_C (A *_V \psi)$) ≤ c ⟩ **if** ⟨*norm* $\psi = 1$ ⟩ **for** ψ

using *that* **using** *asm*[of ⟨*cmod* ($\psi \cdot_C (A *_V \psi)$)⟩] **by** *auto*

have ⟨ $c \geq 0$ ⟩

by (*smt* (*z3*) *ex-norm1-not-singleton c-upper norm-ge-zero*)

have *: ⟨*Re* ($g \cdot_C A h$) ≤ c ⟩ **if** ⟨*norm* $g = 1$ ⟩ **and** ⟨*norm* $h = 1$ ⟩ **for** $g h$

proof —

have *c-upper'*: ⟨*cmod* ($\psi \cdot_C (A *_V \psi)$) ≤ $c * (norm \psi)^2$ ⟩ **for** ψ

apply (*cases* ⟨ $\psi = 0$ ⟩, *simp*)

apply (*subst* (2) *norm-scaleC-*sgn**[*symmetric*, of ψ])

```

apply (subst norm-scaleC-sgn[symmetric])
apply (simp only: cinner-scaleC-left cinner-scaleC-right cblinfun.scaleC-right)
using c-upper[of ⟨sgn ψ⟩]
by (simp add: norm-mult norm-sgn power2-eq-square)
from Aselfadj have Aselfadj':  $x \cdot_C (A *_V y) = (A *_V x) \cdot_C y$  for  $x y$ 
using cinner-adj-right[of  $x A y$ ] by (auto simp: selfadjoint-def)
from Aselfadj have Aselfadj'':  $(A *_V x) \cdot_C y = \text{cnj} ((A *_V y) \cdot_C x)$  for  $x y$ 
by (subst cinner-commute, subst Aselfadj') auto

have 1:  $\langle (h + g) \cdot_C A (h + g) = h \cdot_C A h + 2 * \text{Re} (g \cdot_C A h) + g \cdot_C A g \rangle$ 
by (simp add: cblinfun.c bilinear-simps algebra-simps
  Aselfadj' Aselfadj''[of  $h g$ ] complex-add-cnj del: cinner-commute')
from Aselfadj have 2:  $\langle (h - g) \cdot_C A (h - g) = h \cdot_C A h - 2 * \text{Re} (g \cdot_C A h) + g \cdot_C A g \rangle$ 
by (simp add: cblinfun.c bilinear-simps algebra-simps Aselfadj'
  Aselfadj''[of  $h g$ ] complex-add-cnj del: cinner-commute')
have  $\langle 4 * \text{Re} (g \cdot_C A h) = \text{Re} ((h + g) \cdot_C A (h + g)) - \text{Re} ((h - g) \cdot_C A (h - g)) \rangle$ 
by (smt (verit, ccfv-SIG) 1 2 Re-complex-of-real minus-complex.simps(1) plus-complex.sel(1))
also have  $\langle \dots \leq c * (\text{norm} (h + g))^2 - \text{Re} ((h - g) \cdot_C A (h - g)) \rangle$ 
using c-upper'[of  $\langle h + g \rangle$ ]
by (smt (verit, best) complex-Re-le-cmod)
also have  $\langle \dots \leq c * (\text{norm} (h + g))^2 + c * (\text{norm} (h - g))^2 \rangle$ 
unfolding diff-conv-add-uminus
by (rule add-left-mono)
  (use c-upper'[of  $\langle h - g \rangle$ ] in ⟨smt (verit) abs-Re-le-cmod add-uminus-conv-diff⟩)
also have  $\langle \dots = 2 * c * ((\text{norm} h)^2 + (\text{norm} g)^2) \rangle$ 
by (auto intro!: simp: polar-identity polar-identity-minus ring-distrib)
also have  $\langle \dots \leq 4 * c \rangle$ 
by (simp add: ⟨norm h = 1⟩ ⟨norm g = 1⟩)
finally show  $\langle \text{Re} (g \cdot_C (A *_V h)) \leq c \rangle$ 
by simp
qed
have *:  $\langle \text{cmod} (g \cdot_C A h) \leq c \rangle$  if  $\langle \text{norm} g = 1 \rangle$  and  $\langle \text{norm} h = 1 \rangle$  for  $g h$ 
proof –
define  $\gamma$  where  $\langle \gamma = (\text{if } g \cdot_C A h = 0 \text{ then } 1 \text{ else } \text{sgn} (g \cdot_C A h)) \rangle$ 
have  $\gamma$ :  $\langle \gamma * \text{cmod} (g \cdot_C A h) = g \cdot_C A h \rangle$ 
by (simp add:  $\gamma$ -def sgn-eq)
have  $\langle \text{norm } \gamma = 1 \rangle$ 
by (simp add:  $\gamma$ -def norm-sgn)
have  $\langle \text{cmod} (g \cdot_C A h) = \text{Re} (\text{complex-of-real} (\text{norm} (g \cdot_C A h))) \rangle$ 
by simp
also have  $\langle \dots = \text{Re} (g \cdot_C (A (h /_C \gamma))) \rangle$ 
using  $\gamma$   $\langle \text{cmod } \gamma = 1 \rangle$ 
by (smt (verit) Groups.mult-ac(2) Groups.mult-ac(3) cblinfun.scaleC-right cinner-scaleC-right
  left-inverse more-arith-simps(6) norm-eq-zero)
also have  $\langle \dots \leq c \rangle$ 
using  $\langle \text{norm } \gamma = 1 \rangle$ 
by (auto intro!: * simp: that norm-inverse)
finally show  $\langle \text{cmod} (g \cdot_C (A *_V h)) \leq c \rangle$ 
by –

```

qed
have $\langle \text{norm } (A \ h) \leq c \rangle$ **if** $\langle \text{norm } h = 1 \rangle$ **for** h
by (cases $\langle A \ h = 0 \rangle$)
 (use $*[OF - \text{that, of } \langle \text{sgn } (A \ h) \rangle]$ **in** $\langle \text{simp-all add: norm-sgn } \langle 0 \leq c \rangle \rangle$)
then show $\langle \text{norm } A \leq c \rangle$
using $\langle c \geq 0 \rangle$ **by** (auto intro!: norm-cblinfun-bound-unit)
qed

lemma *cblinfun-norm-approx-witness-cinner*:
fixes $A :: \langle 'a :: \{ \text{not-singleton, chilbert-space} \} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{selfadjoint } A \rangle$ **and** $\langle \varepsilon > 0 \rangle$
shows $\langle \exists \psi. \text{cmod } (\psi \cdot_C (A *_V \psi)) \geq \text{norm } A - \varepsilon \wedge \text{norm } \psi = 1 \rangle$
using *is-Sup-approx-below*[*OF cblinfun-norm-is-Sup-cinner*[*OF assms(1)*]] *assms(2)*
by *blast*

lemma *cblinfun-norm-approx-witness-cinner'*:
fixes $A :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{selfadjoint } A \rangle$ **and** $\langle \varepsilon > 0 \rangle$
shows $\langle \exists \psi. \text{cmod } (\psi \cdot_C A \ \psi) / (\text{norm } \psi)^2 \geq \text{norm } A - \varepsilon \rangle$
proof (cases $\langle \text{class.not-singleton TYPE('a)} \rangle$)
case *True*
obtain ψ **where** $\langle \text{cmod } (\psi \cdot_C A \ \psi) \geq \text{norm } A - \varepsilon \rangle$ **and** $\langle \text{norm } \psi = 1 \rangle$
apply *atomize-elim*
using *chilbert-space-axioms True assms*
by (*rule cblinfun-norm-approx-witness-cinner*[*internalize-sort' 'a*])
then have $\langle \text{cmod } (\psi \cdot_C A \ \psi) / (\text{norm } \psi)^2 \geq \text{norm } A - \varepsilon \rangle$
by *simp*
then show *?thesis*
by *auto*
next
case *False*
show *?thesis*
apply (*subst not-not-singleton-cblinfun-zero*[*OF False*])
apply *simp*
apply (*subst not-not-singleton-cblinfun-zero*[*OF False*])
using $\langle \varepsilon > 0 \rangle$ **by** *simp*
qed

lemma *has-sum-mono-neutral-cblinfun*:
fixes $f :: 'a \Rightarrow ('b :: \text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\langle (f \ \text{has-sum } a) \ A \rangle$ **and** $\langle (g \ \text{has-sum } b) \ B \rangle$
assumes $\langle \bigwedge x. x \in A \cap B \implies f \ x \leq g \ x \rangle$
assumes $\langle \bigwedge x. x \in A - B \implies f \ x \leq 0 \rangle$
assumes $\langle \bigwedge x. x \in B - A \implies g \ x \geq 0 \rangle$
shows $a \leq b$
proof –
from *assms*
have *sum-hfh*: $\langle ((\lambda x. h \cdot_C f \ x \ h) \ \text{has-sum } h \cdot_C a \ h) \ A \rangle$ **for** h
by (*intro has-sum-cinner-left has-sum-cblinfun-apply-left*)

```

from assms
have sum-hgh:  $\langle (\lambda x. h \cdot_C g x h) \text{ has-sum } h \cdot_C b h \rangle B$  for h
by (intro has-sum-cinner-left has-sum-cblinfun-apply-left)
from sum-hfh sum-hgh
have  $\langle h \cdot_C a h \leq h \cdot_C b h \rangle$  for h
apply (rule has-sum-mono-neutral-complex)
using assms
by (auto intro!: simp: less-eq-cblinfun-def)
then show  $\langle a \leq b \rangle$ 
by (simp add: less-eq-cblinfun-def)
qed

```

lemma *sums-mono-cblinfun*:

```

fixes f :: nat  $\Rightarrow$  (b::chilbert-space  $\Rightarrow_{CL}$  b)
assumes  $\langle f \text{ sums } a \rangle$  and g sums b
assumes  $\langle \bigwedge n. f n \leq g n \rangle$ 
shows  $a \leq b$ 
proof (rule cblinfun-leI)
fix h
from  $\langle f \text{ sums } a \rangle$ 
have sum1:  $\langle (\lambda n. h \cdot_C (f n *_{\mathcal{V}} h)) \text{ sums } (h \cdot_C (a *_{\mathcal{V}} h)) \rangle$ 
apply (rule bounded-linear.sums[rotated])
using bounded-clinear.bounded-linear bounded-clinear-cinner-right bounded-linear-compose
cblinfun.real.bounded-linear-left by blast
from  $\langle g \text{ sums } b \rangle$ 
have sum2:  $\langle (\lambda n. h \cdot_C (g n *_{\mathcal{V}} h)) \text{ sums } (h \cdot_C (b *_{\mathcal{V}} h)) \rangle$ 
apply (rule bounded-linear.sums[rotated])
by (metis bounded-linear-compose cblinfun.real.bounded-linear-left cblinfun.real.bounded-linear-right
cblinfun-cinner-right.rep-eq)
have  $\langle h \cdot_C (f n *_{\mathcal{V}} h) \leq h \cdot_C (g n *_{\mathcal{V}} h) \rangle$  for n
using assms(3) less-eq-cblinfun-def by auto
with sum1 sum2
show  $\langle h \cdot_C (a *_{\mathcal{V}} h) \leq h \cdot_C (b *_{\mathcal{V}} h) \rangle$ 
by (rule sums-le-complex[rotated])
qed

```

lemma *scaleC-scaleR-commute*: $\langle a *_{\mathcal{C}} b *_{\mathcal{R}} x = b *_{\mathcal{R}} a *_{\mathcal{C}} x \rangle$ **for** *x* :: $\langle \cdot \rangle$:*complex-normed-vector*
by (*simp add: scaleR-scaleC scaleC-left-commute*)

lemma *sandwich-scaleC-left*: $\langle \text{sandwich } (c *_{\mathcal{C}} e) = (c \text{ mod } c)^{\wedge 2} *_{\mathcal{C}} \text{sandwich } e \rangle$
by (*auto intro!: cblinfun-eqI simp: sandwich-apply cnj-x-x abs-complex-def*)

lemma *sandwich-scaleR-left*: $\langle \text{sandwich } (r *_{\mathcal{R}} e) = r^{\wedge 2} *_{\mathcal{R}} \text{sandwich } e \rangle$
by (*simp add: scaleR-scaleC sandwich-scaleC-left flip: of-real-power*)

lemma *infsun-product*:

```

fixes f ::  $\langle 'a \Rightarrow 'c :: \{ \text{topological-semigroup-mult, division-ring, banach} \} \rangle$ 
assumes  $\langle (\lambda(x, y). f x * g y) \text{ summable-on } X \times Y \rangle$ 

```

shows $\langle (\sum_{\infty} x \in X. f x) * (\sum_{\infty} y \in Y. g y) = (\sum_{\infty} (x,y) \in X \times Y. f x * g y) \rangle$
using *assms*
by (*simp add: infsum-cmult-right' infsum-cmult-left' flip: infsum-Sigma'-banach*)

lemma *infsum-product'*:

fixes $f :: \langle 'a \Rightarrow 'c :: \{banach, times, real-normed-algebra\} \rangle$ **and** $g :: \langle 'b \Rightarrow 'c \rangle$
assumes $\langle f \text{ abs-summable-on } X \rangle$
assumes $\langle g \text{ abs-summable-on } Y \rangle$
shows $\langle (\sum_{\infty} x \in X. f x) * (\sum_{\infty} y \in Y. g y) = (\sum_{\infty} (x,y) \in X \times Y. f x * g y) \rangle$
using *assms*

by (*simp add: abs-summable-times infsum-cmult-right infsum-cmult-left abs-summable-summable flip: infsum-Sigma'-banach*)

lemma *Proj-o-Proj-subspace-right*:

assumes $\langle A \geq B \rangle$
shows $\langle Proj A \text{ } o_{CL} \text{ } Proj B = Proj B \rangle$
by (*simp add: Proj-compose-cancelI assms*)

lemma *Proj-o-Proj-subspace-left*:

assumes $\langle A \leq B \rangle$
shows $\langle Proj A \text{ } o_{CL} \text{ } Proj B = Proj A \rangle$
by (*metis Proj-o-Proj-subspace-right adj-Proj adj-cblinfun-compose assms*)

lemma *orthogonal-spaces-SUP-left*:

assumes $\langle \bigwedge x. x \in X \implies \text{orthogonal-spaces } (A x) B \rangle$
shows $\langle \text{orthogonal-spaces } (\bigsqcup x \in X. A x) B \rangle$
by (*meson SUP-least assms orthogonal-spaces-leq-compl*)

lemma *orthogonal-spaces-SUP-right*:

assumes $\langle \bigwedge x. x \in X \implies \text{orthogonal-spaces } A (B x) \rangle$
shows $\langle \text{orthogonal-spaces } A (\bigsqcup x \in X. B x) \rangle$
by (*meson assms orthogonal-spaces-SUP-left orthogonal-spaces-sym*)

lemma *orthogonal-bot-left[simp]*: $\langle \text{orthogonal-spaces bot } S \rangle$

by (*simp add: orthogonal-spaces-def*)

lemma *infsum-bounded-linear-invertible*:

assumes $\langle \text{bounded-linear } h \rangle$
assumes $\langle \text{bounded-linear } h' \rangle$
assumes $\langle h' \circ h = id \rangle$
shows $\langle \text{infsum } (\lambda x. h (f x)) A = h (\text{infsum } f A) \rangle$

proof (*cases* $\langle f \text{ summable-on } A \rangle$)

case *True*

then show *?thesis*

using *assms(1) infsum-bounded-linear* **by** *blast*

next

case *False*

have $\langle \neg (\lambda x. h (f x)) \text{ summable-on } A \rangle$

```

proof (rule ccontr)
  assume  $\langle \neg \neg (\lambda x. h (f x)) \text{ summable-on } A \rangle$ 
  with  $\langle \text{bounded-linear } h' \rangle$  have  $\langle h' \circ h \circ f \text{ summable-on } A \rangle$ 
    by (auto intro: summable-on-bounded-linear simp: o-def)
  then have  $\langle f \text{ summable-on } A \rangle$ 
    by (simp add: assms(3))
  with False show False
    by blast
qed
then show ?thesis
  by (simp add: False assms(1) infsum-not-exists linear-simps(3))
qed

lemma cblinfun-eq-from-separatingI:
  fixes  $a b :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector} \rangle$ 
  assumes  $\langle \text{separating-set (bounded-clinear} :: ('a \Rightarrow 'b) \Rightarrow \text{bool}) } S \rangle$ 
  assumes  $\langle \bigwedge x. x \in S \implies a x = b x \rangle$ 
  shows  $\langle a = b \rangle$ 
  apply (rule cblinfun-eqI, rule fun-cong[where  $f = \langle \text{cblinfun-apply } - \rangle$ ])
  using assms(1) apply (rule eq-from-separatingI)
  using assms(2) by (auto intro!: bounded-cbilinear-apply-bounded-clinear cblinfun.bounded-cbilinear-axioms
    simp: )

lemma cblinfun-eq-from-separatingI2:
  fixes  $a b :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector} \rangle$ 
  assumes  $\langle \text{separating-set (bounded-clinear} :: ('a \Rightarrow 'b) \Rightarrow \text{bool}) } ((\lambda(x,y). h x y) \text{ ' } (S \times T)) \rangle$ 
  assumes  $\langle \bigwedge x y. x \in S \implies y \in T \implies a (h x y) = b (h x y) \rangle$ 
  shows  $\langle a = b \rangle$ 
  apply (rule cblinfun-eqI, rule fun-cong[where  $f = \langle \text{cblinfun-apply } - \rangle$ ])
  using assms(1) apply (rule eq-from-separatingI2)
  using assms(2) by (auto intro!: bounded-cbilinear-apply-bounded-clinear cblinfun.bounded-cbilinear-axioms
    simp: )

lemma separating-set-bounded-clinear-dense:
  assumes  $\langle \text{ccspan } S = \top \rangle$ 
  shows  $\langle \text{separating-set bounded-clinear } S \rangle$ 
  unfolding separating-set-def
  by (intro allI impI ext, rule bounded-clinear-eq-on-closure[where  $G=S$ ])
    (use assms ccspan.rep-eq in force)+)

lemma separating-set-ket:  $\langle \text{separating-set bounded-clinear (range ket)} \rangle$ 
  by (simp add: bounded-clinear-equal-ket separating-setI)

lemma separating-set-bounded-cbilinear-nested:
  assumes  $\langle \text{separating-set (bounded-clinear} :: (- \Rightarrow 'e::\text{complex-normed-vector}) \Rightarrow -) } ((\lambda(x, y). h x y) \text{ ' } (UNIV \times UNIV)) \rangle$ 
  assumes  $\langle \text{bounded-cbilinear } h \rangle$ 
  assumes  $\langle \text{separating-set (bounded-clinear} :: (- \Rightarrow 'e) \Rightarrow -) } A \rangle$ 
  assumes  $\langle \text{separating-set (bounded-clinear} :: (- \Rightarrow 'e) \Rightarrow -) } B \rangle$ 

```

```

  shows ‹separating-set (bounded-clinear :: (- => 'e) => -) ((λ(x,y). h x y) ‘(A × B))›
proof (rule separating-setI)
  fix f g :: ‹'a => 'e›
  assume [simp]: ‹bounded-clinear f› ‹bounded-clinear g›
  have [simp]: ‹bounded-clinear (λx. f (h x y))› for y
    apply (rule bounded-clinear-compose[OF ‹bounded-clinear f›])
    using assms(2) by (rule bounded-cbilinear.bounded-clinear-left)
  have [simp]: ‹bounded-clinear (λx. g (h x y))› for y
    apply (rule bounded-clinear-compose[OF ‹bounded-clinear g›])
    using assms(2) by (rule bounded-cbilinear.bounded-clinear-left)
  have [simp]: ‹bounded-clinear (λy. f (h x y))› for x
    apply (rule bounded-clinear-compose[OF ‹bounded-clinear f›])
    using assms(2) by (rule bounded-cbilinear.bounded-clinear-right)
  have [simp]: ‹bounded-clinear (λy. g (h x y))› for x
    apply (rule bounded-clinear-compose[OF ‹bounded-clinear g›])
    using assms(2) by (rule bounded-cbilinear.bounded-clinear-right)

  assume ‹z ∈ (λ(x, y). h x y) ‘(A × B) ==> f z = g z› for z
  then have ‹f (h x y) = g (h x y)› if ‹x ∈ A› and ‹y ∈ B› for x y
    using that by auto
  then have ‹(λx. f (h x y)) = (λx. g (h x y))› if ‹y ∈ B› for y
    by (intro eq-from-separatingI[OF assms(3)]) (use that in auto)
  then have ‹(λy. f (h x y)) = (λy. g (h x y))› for x
    apply (intro eq-from-separatingI[OF assms(4)])
    subgoal by simp
    subgoal by simp
    subgoal by meson
  done
  then have ‹f (h x y) = g (h x y)› for x y
    by meson
  with ‹bounded-clinear f› ‹bounded-clinear g›
  show ‹f = g›
    by (rule eq-from-separatingI2[where f=f and g=g and P=bounded-clinear and S=UNIV
and T=UNIV, rotated 1])
    (fact assms(1))
qed

```

lemma *separating-set-bounded-clinear-antilinear:*

```

  assumes ‹separating-set (bounded-clinear :: (- => 'e::complex-normed-vector conjugate-space)
=> -) A›
  shows ‹separating-set (bounded-antilinear :: (- => 'e) => -) A›
proof (rule separating-setI)
  fix f g :: ‹'a => 'e›
  assume ‹bounded-antilinear f›
  then have lin-f: ‹bounded-clinear (to-conjugate-space o f)›
    by (simp add: bounded-antilinear-o-bounded-antilinear)
  assume ‹bounded-antilinear g›
  then have lin-g: ‹bounded-clinear (to-conjugate-space o g)›

```



```

    by (simp add: bounded-antilinear-o-bounded-antilinear')
  assume ⟨f x = g x⟩ if ⟨x ∈ A⟩ for x
  then have ⟨(to-conjugate-space o f) x = (to-conjugate-space o g) x⟩ if ⟨x ∈ A⟩ for x
    by (simp add: that)
  with lin-f lin-g
  have ⟨to-conjugate-space o f = to-conjugate-space o g⟩
    by (rule eq-from-separatingI[OF assms])
  then show ⟨f = g⟩
    by (metis UNIV-I fun.inj-map-strong to-conjugate-space-inverse)
qed

```

lemma *separating-set-bounded-sesquilinear-nested*:

```

  assumes ⟨separating-set (bounded-clinear :: (- => 'e::complex-normed-vector) => -) ((λ(x, y).
  h x y) ' (UNIV × UNIV))⟩

```

```

  assumes ⟨bounded-sesquilinear h⟩

```

```

  assumes sep-A: ⟨separating-set (bounded-clinear :: (- => 'e conjugate-space) => -) A⟩

```

```

  assumes sep-B: ⟨separating-set (bounded-clinear :: (- => 'e) => -) B⟩

```

```

  shows ⟨separating-set (bounded-clinear :: (- => 'e) => -) ((λ(x,y). h x y) ' (A × B))⟩

```

proof (rule separating-setI)

```

  fix f g :: ⟨'a => 'e⟩

```

```

  assume [simp]: ⟨bounded-clinear f⟩ ⟨bounded-clinear g⟩

```

```

  have [simp]: ⟨bounded-antilinear (λx. f (h x y))⟩ for y

```

```

    apply (rule bounded-clinear-o-bounded-antilinear[OF ⟨bounded-clinear f⟩])

```

```

    using assms(2) by (rule bounded-sesquilinear.bounded-antilinear-left)

```

```

  have [simp]: ⟨bounded-antilinear (λx. g (h x y))⟩ for y

```

```

    apply (rule bounded-clinear-o-bounded-antilinear[OF ⟨bounded-clinear g⟩])

```

```

    using assms(2) by (rule bounded-sesquilinear.bounded-antilinear-left)

```

```

  have [simp]: ⟨bounded-clinear (λy. f (h x y))⟩ for x

```

```

    apply (rule bounded-clinear-compose[OF ⟨bounded-clinear f⟩])

```

```

    using assms(2) by (rule bounded-sesquilinear.bounded-clinear-right)

```

```

  have [simp]: ⟨bounded-clinear (λy. g (h x y))⟩ for x

```

```

    apply (rule bounded-clinear-compose[OF ⟨bounded-clinear g⟩])

```

```

    using assms(2) by (rule bounded-sesquilinear.bounded-clinear-right)

```

```

  from sep-A have sep-A': ⟨separating-set (bounded-antilinear :: (- => 'e) => -) A⟩

```

```

    by (rule separating-set-bounded-clinear-antilinear)

```

```

  assume ⟨z ∈ (λ(x, y). h x y) ' (A × B) ⟹ f z = g z⟩ for z

```

```

  then have ⟨f (h x y) = g (h x y)⟩ if ⟨x ∈ A⟩ and ⟨y ∈ B⟩ for x y

```

```

    using that by auto

```

```

  then have ⟨(λx. f (h x y)) = (λx. g (h x y))⟩ if ⟨y ∈ B⟩ for y

```

```

    by (intro eq-from-separatingI[OF sep-A']) (use that in auto)

```

```

  then have ⟨(λy. f (h x y)) = (λy. g (h x y))⟩ for x

```

```

    apply (intro eq-from-separatingI[OF sep-B])

```

```

    subgoal by simp

```

```

    subgoal by simp

```

```

    subgoal by meson

```

```

  done

```

```

  then have ⟨f (h x y) = g (h x y)⟩ for x y

```

```

    by meson

```

with $\langle \text{bounded-clinear } f \rangle \langle \text{bounded-clinear } g \rangle$
show $\langle f = g \rangle$
by (rule eq-from-separatingI2[**where** $f=f$ **and** $g=g$ **and** $P=\text{bounded-clinear}$ **and** $S=\text{UNIV}$
and $T=\text{UNIV}$, rotated 1])
 (fact *assms*(1))
qed

lemma *eq-on-ccsubspaces-Sup*:

fixes $a\ b :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector} \rangle$
assumes $\langle \bigwedge i\ h. i \in I \implies h \in \text{space-as-set } (X\ i) \implies a\ h = b\ h \rangle$
shows $\langle \bigwedge h. h \in \text{space-as-set } (\bigsqcup i \in I. X\ i) \implies a\ h = b\ h \rangle$
proof –
from *assms*
have $\langle X\ i \leq \text{kernel } (a - b) \rangle$ **if** $\langle i \in I \rangle$ **for** i
using that **by** (auto intro!: *ccsubspace-leI simp: kernel.rep-eq minus-cblinfun.rep-eq*)
then have $\langle (\bigsqcup i \in I. X\ i) \leq \text{kernel } (a - b) \rangle$
by (*simp add: SUP-least*)
then show $\langle h \in \text{space-as-set } (\bigsqcup i \in I. X\ i) \implies a\ h = b\ h \rangle$ **for** h
using *kernel-memberD less-eq-ccsubspace.rep-eq*
by (*metis (no-types, opaque-lifting) cblinfun.diff-left cblinfun.real.diff-right cblinfun.real.zero-left diff-eq-diff-eq double-diff mem-simps(6) subset-refl*)
qed

lemma *eq-on-ccsubspaces-sup*:

fixes $a\ b :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector} \rangle$
assumes $\langle \bigwedge h\ i. h \in \text{space-as-set } S \implies a\ h = b\ h \rangle$
assumes $\langle \bigwedge h\ i. h \in \text{space-as-set } T \implies a\ h = b\ h \rangle$
shows $\langle \bigwedge h. h \in \text{space-as-set } (S \sqcup T) \implies a\ h = b\ h \rangle$
apply (rule *eq-on-ccsubspaces-Sup*[**where** $I = \langle \{ \text{True}, \text{False} \} \rangle$ **and** $X = \langle \lambda i. \text{if } i \text{ then } T \text{ else } S \rangle$])
using *assms*
apply *presburger*
by *fastforce*

lemma *ccsubspace-contains-unit*:

assumes $\langle E \neq \perp \rangle$
shows $\langle \exists h \in \text{space-as-set } E. \text{norm } h = 1 \rangle$
proof –
from *assms* **have** $\langle \text{space-as-set } E \neq \{0\} \rangle$
by (*metis bot-ccsubspace.rep-eq space-as-set-inject*)
then obtain h_0 **where** $\langle h_0 \in \text{space-as-set } E \rangle$ **and** $\langle h_0 \neq 0 \rangle$
by *auto*
then have $\langle \text{sgn } h_0 \in \text{space-as-set } E \rangle$
using *ccsubspace-space-as-set*
by (auto intro!: *complex-vector.subspace-scale simp add: sgn-div-norm scaleR-scaleC*)
moreover from $\langle h_0 \neq 0 \rangle$ **have** $\langle \text{norm } (\text{sgn } h_0) = 1 \rangle$
by (*simp add: norm-sgn*)
ultimately show *?thesis*

by auto
qed

lemma *Proj-0-compl*: $\langle \text{Proj } S \ x = 0 \rangle$ **if** $\langle x \in \text{space-as-set } (-S) \rangle$
by (*simp add: kernel-memberD* that)

lemma *csubspace-has-basis*:
assumes $\langle \text{csubspace } S \rangle$
shows $\langle \exists B. \text{cindependent } B \wedge \text{cspan } B = S \rangle$
proof –
obtain *B* **where** $\langle \text{cindependent } B \rangle$ **and** $\langle \text{cspan } B = S \rangle$
by (*rule complex-vector.maximal-independent-subset*[**where** $V=S$])
(*use assms complex-vector.span-subspace in blast*)
then show *?thesis*
by auto
qed

lemma *inj-scaleC*:
fixes $A :: \langle 'a :: \text{complex-vector set} \rangle$
assumes $\langle c \neq 0 \rangle$
shows $\langle \text{inj-on } (\text{scaleC } c) \ A \rangle$
by (*meson assms inj-onI scaleC-left-imp-eq*)

definition *diagonal-operator* **where** $\langle \text{diagonal-operator } f =$
(*if bdd-above* ($\text{range } (\lambda x. \text{cmod } (f \ x))$) *then explicit-cblinfun* ($\lambda x \ y. \text{of-bool } (x=y) * f \ x$) *else*
 0) \rangle

lemma *diagonal-operator-exists*:
assumes $\langle \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (f \ x))) \rangle$
shows $\langle \text{explicit-cblinfun-exists } (\lambda x \ y. \text{of-bool } (x = y) * f \ x) \rangle$
proof –
from *assms* **obtain** *B* **where** $B: \langle \text{cmod } (f \ x) \leq B \rangle$ **for** *x*
by (*auto simp: bdd-above-def*)
show *?thesis*
proof (*rule explicit-cblinfun-exists-bounded*)
fix $S \ T :: \langle 'a \ \text{set} \rangle$ **and** $\psi :: \langle 'a \Rightarrow \text{complex} \rangle$
assume [*simp*]: $\langle \text{finite } S \rangle \langle \text{finite } T \rangle$
assume $\langle \psi \ a = 0 \rangle$ **if** $\langle a \notin T \rangle$ **for** *a*
have $\langle (\sum b \in S. (\text{cmod } (\sum a \in T. \psi \ a *_{\mathbb{C}} (\text{of-bool } (b = a) * f \ b)))^2$
 $= (\sum b \in S. (\text{cmod } (\text{of-bool } (b \in T) * \psi \ b * f \ b))^2 \rangle$
apply (*rule sum.cong[OF refl]*)
subgoal for *b*
apply (*subst sum-single*[**where** $i=b$])
by auto
by –
also have $\langle \dots = (\sum b \in S \cap T. (\text{cmod } (\psi \ b * f \ b))^2 \rangle$

```

    apply (rule sum.mono-neutral-cong-right)
    by auto
  also have ⟨... ≤ (∑ b∈T. (cmod (ψ b * f b))2)⟩
    by (simp add: sum-mono2)
  also have ⟨... ≤ (∑ b∈T. B2 * (cmod (ψ b))2)⟩
    by (rule sum-mono)
      (auto intro!: mult-left-mono power-mono B
        simp: norm-mult power-mult-distrib mult.commute[of B ^ 2])
  also have ⟨... = B2 * (∑ b∈T. (cmod (ψ b))2)⟩
    by (simp add: vector-space-over-itself.scale-sum-right)
  finally
  show ⟨(∑ b∈S. (cmod (∑ a∈T. ψ a *C (of-bool (b = a) * f b)))2)
    ≤ B2 * (∑ a∈T. (cmod (ψ a))2)⟩ .
qed
qed

```

lemma *diagonal-operator-ket*:

```

  assumes ⟨bdd-above (range (λx. cmod (f x)))⟩
  shows ⟨diagonal-operator f (ket x) = f x *C ket x⟩
  proof -
    have [simp]: ⟨has-ell2-norm (λb. of-bool (b = x) * f b)⟩
      by (auto intro!: finite-nonzero-values-imp-summable-on simp: has-ell2-norm-def)
    have ⟨Abs-ell2 (λb. of-bool (b = x) * f b) = f x *C ket x⟩
      by (rule Rep-ell2-inject[THEN iffD1])
        (auto simp: Abs-ell2-inverse scaleC-ell2.rep-eq ket.rep-eq)
    then show ?thesis
      by (auto intro!: simp: diagonal-operator-def assms explicit-cblinfun-ket diagonal-operator-exists)
  qed

```

lemma *diagonal-operator-invalid*:

```

  assumes ⟨¬ bdd-above (range (λx. cmod (f x)))⟩
  shows ⟨diagonal-operator f = 0⟩
  by (simp add: assms diagonal-operator-def)

```

lemma *diagonal-operator-adj*: ⟨diagonal-operator f* = diagonal-operator (λx. cnj (f x))⟩

```

  by (cases ⟨bdd-above (range (λx. cmod (f x)))⟩)
    (auto intro!: equal-ket cinner-ket-eqI
      simp: diagonal-operator-ket cinner-adj-right diagonal-operator-invalid)

```

lemma *diagonal-operator-comp*:

```

  assumes ⟨bdd-above (range (λx. cmod (f x)))⟩
  assumes ⟨bdd-above (range (λx. cmod (g x)))⟩
  shows ⟨diagonal-operator f oCL diagonal-operator g = diagonal-operator (λx. (f x * g x))⟩
  proof -
    have ⟨bdd-above (range (λx. cmod (f x * g x)))⟩
    proof -
      from assms(1) obtain F where ⟨cmod (f x) ≤ F⟩ for x

```

```

    by (auto simp: bdd-above-def)
  moreover from assms(2) obtain G where  $\langle cmod (g x) \leq G \rangle$  for x
    by (auto simp: bdd-above-def)
  ultimately have  $\langle cmod (f x * g x) \leq F * G \rangle$  for x
    by (smt (verit, del-Insts) mult-right-mono norm-ge-zero norm-mult ordered-comm-semiring-class.comm-mult-left-r)
  then show ?thesis
    by fast
qed
then show ?thesis
  by (auto intro!: equal-ket simp: diagonal-operator-ket assms cblinfun.scaleC-right)
qed

```

lemma *summable-on-bdd-above-real*: $\langle bdd-above (f 'M) \rangle$ if $\langle f \text{ summable-on } M \rangle$ for *f* :: $\langle 'a \Rightarrow \text{real} \rangle$

```

proof -
  from that have  $\langle f \text{ abs-summable-on } M \rangle$ 
    unfolding summable-on-iff-abs-summable-on-real[symmetric]
    by -
  then have  $\langle bdd-above (sum (\lambda x. norm (f x))) ' \{F. F \subseteq M \wedge \text{finite } F\} \rangle$ 
    unfolding abs-summable-iff-bdd-above by simp
  then have  $\langle bdd-above (sum (\lambda x. norm (f x))) ' (\lambda x. \{x\}) ' M \rangle$ 
    by (rule bdd-above-mono) auto
  then have  $\langle bdd-above ((\lambda x. norm (f x)) ' M) \rangle$ 
    by (simp add: image-image)
  then show ?thesis
    by (simp add: bdd-above-mono2)
qed

```

lemma *separating-set-clinear-cspan*:
assumes $\langle cspan S = UNIV \rangle$
shows $\langle \text{separating-set clinear } S \rangle$
using *assms*
by (*auto intro: complex-vector.linear-eq-on simp: separating-set-def*)

lemma *less-eq-cblinfunI*:
fixes *a b* :: $\langle 'a \Rightarrow_{CL} 'a::\text{hilbert-space} \rangle$
assumes $\langle \bigwedge h. h \cdot_C a h \leq h \cdot_C b h \rangle$
shows $\langle a \leq b \rangle$
using *assms*
by (*simp add: less-eq-cblinfun-def*)

end

3 Strong-Operator-Topology – Strong operator topology on complex bounded operators

theory *Strong-Operator-Topology*
imports *Misc-Tensor-Product-BO*

begin

unbundle *cblinfun-syntax*

typedef (**overloaded**) ('a,'b) *cblinfun-sot* = $\langle UNIV :: ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector}) \text{ set} \rangle ..$

setup-lifting *type-definition-cblinfun-sot*

instantiation *cblinfun-sot* :: (*complex-normed-vector*, *complex-normed-vector*) *complex-vector*

begin

lift-definition *scaleC-cblinfun-sot* :: $\langle \text{complex} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \rangle$
is $\langle \text{scaleC} \rangle .$

lift-definition *uminus-cblinfun-sot* :: $\langle ('a, 'b) \text{ cblinfun-sot} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \rangle$ **is** *uminus* .

lift-definition *zero-cblinfun-sot* :: $\langle ('a, 'b) \text{ cblinfun-sot} \rangle$ **is** *0* .

lift-definition *minus-cblinfun-sot* :: $\langle ('a, 'b) \text{ cblinfun-sot} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \rangle$ **is** *minus* .

lift-definition *plus-cblinfun-sot* :: $\langle ('a, 'b) \text{ cblinfun-sot} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \rangle$ **is** *plus* .

lift-definition *scaleR-cblinfun-sot* :: $\langle \text{real} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \Rightarrow ('a, 'b) \text{ cblinfun-sot} \rangle$ **is** *scaleR* .

instance

apply (*intro-classes*; *transfer*)

by (*auto simp add: scaleR-scaleC scaleC-add-right scaleC-add-left*)

end

instantiation *cblinfun-sot* :: (*complex-normed-vector*, *complex-normed-vector*) *topological-space*

begin

lift-definition *open-cblinfun-sot* :: $\langle ('a, 'b) \text{ cblinfun-sot set} \Rightarrow \text{bool} \rangle$ **is** $\langle \text{openin cstrong-operator-topology} \rangle$
.

instance

proof *intro-classes*

show $\langle \text{open } (UNIV :: ('a,'b) \text{ cblinfun-sot set}) \rangle$

apply *transfer*

by (*metis cstrong-operator-topology-topspace openin-topspace*)

show $\langle \text{open } S \Longrightarrow \text{open } T \Longrightarrow \text{open } (S \cap T) \rangle$ **for** $S T :: \langle ('a,'b) \text{ cblinfun-sot set} \rangle$

apply *transfer by auto*

show $\langle \forall S \in K. \text{open } S \Longrightarrow \text{open } (\bigcup K) \rangle$ **for** $K :: \langle ('a,'b) \text{ cblinfun-sot set set} \rangle$

apply *transfer by auto*

qed

end

lemma *transfer-nhds-cstrong-operator-topology[transfer-rule]*:

includes *lifting-syntax*

shows $\langle (\text{cr-cblinfun-sot} ==>> \text{rel-filter cr-cblinfun-sot}) (\text{nhdsin cstrong-operator-topology}) \text{ nhds} \rangle$

unfolding *nhds-def nhdsin-def*

apply (*simp add: cstrong-operator-topology-topospace*)
by *transfer-prover*

lemma *filterlim-cstrong-operator-topology*: $\langle \text{filterlim } f \text{ (nhdsin cstrong-operator-topology } l) = \text{limitin cstrong-operator-topology } f \text{ } l \rangle$
by (*auto simp: cstrong-operator-topology-topospace simp flip: filterlim-nhdsin-iff-limitin*)

lemma *hausdorff-sot[*simp*]*: $\langle \text{Hausdorff-space cstrong-operator-topology} \rangle$

proof (*rule hausdorffI*)

fix $a \ b :: \langle 'a \Rightarrow_{CL} 'b \rangle$

assume $\langle a \neq b \rangle$

then obtain ψ **where** $\langle a *_V \psi \neq b *_V \psi \rangle$

by (*meson cblinfun-eqI*)

then obtain $U' \ V'$ **where** $\langle \text{open } U' \rangle \langle \text{open } V' \rangle \langle a *_V \psi \in U' \rangle \langle b *_V \psi \in V' \rangle \langle U' \cap V' = \{\} \rangle$

by (*meson hausdorff*)

define $U \ V$ **where** $\langle U = \{f. \forall i \in \{\}. f *_V \psi \in U'\} \rangle$ **and** $\langle V = \{f. \forall i \in \{\}. f *_V \psi \in V'\} \rangle$

have $1: \langle \text{openin cstrong-operator-topology } U \rangle$

unfolding *U-def* **apply** (*rule cstrong-operator-topology-basis*)

using $\langle \text{open } U' \rangle$ **by** *auto*

have $2: \langle \text{openin cstrong-operator-topology } V \rangle$

unfolding *V-def* **apply** (*rule cstrong-operator-topology-basis*)

using $\langle \text{open } V' \rangle$ **by** *auto*

show $\langle \exists U \ V. \text{openin cstrong-operator-topology } U \wedge \text{openin cstrong-operator-topology } V \wedge a \in U \wedge b \in V \wedge U \cap V = \{\} \rangle$

by (*rule exI[of - U], rule exI[of - V]*)

(*use 1 2* $\langle a *_V \psi \in U' \rangle \langle b *_V \psi \in V' \rangle \langle U' \cap V' = \{\} \rangle$ **in** $\langle \text{auto simp: } U\text{-def } V\text{-def} \rangle$)

qed

instance *cblinfun-sot* :: (*complex-normed-vector, complex-normed-vector*) *t2-space*

proof *intro-classes*

fix $a \ b :: \langle 'a, 'b \rangle$ *cblinfun-sot*

show $\langle a \neq b \implies \exists U \ V. \text{open } U \wedge \text{open } V \wedge a \in U \wedge b \in V \wedge U \cap V = \{\} \rangle$

apply *transfer using hausdorff-sot*

by (*metis UNIV-I cstrong-operator-topology-topospace Hausdorff-space-def disjnt-def*)

qed

lemma *Domainp-cr-cblinfun-sot[*simp*]*: $\langle \text{Domainp cr-cblinfun-sot} = (\lambda-. \text{True}) \rangle$

by (*metis (no-types, opaque-lifting) DomainPI cblinfun-sot.left-total left-totalE*)

lemma *Rangep-cr-cblinfun-sot[*simp*]*: $\langle \text{Rangep cr-cblinfun-sot} = (\lambda-. \text{True}) \rangle$

by (*meson RangePI cr-cblinfun-sot-def*)

lemma *Rangep-set[*relator-domain*]*: $\text{Rangep (rel-set } T) = (\lambda A. \text{Ball } A \ (\text{Rangep } T))$

by (*metis (no-types, opaque-lifting) Domainp-conversep Domainp-set rel-set-conversep*)

```

lemma transfer-euclidean-cstrong-operator-topology[transfer-rule]:
  includes lifting-syntax
  shows ⟨(rel-topology cr-cblinfun-sot) cstrong-operator-topology euclidean⟩
proof (unfold rel-topology-def, intro conjI allI impI)
  show ⟨(rel-set cr-cblinfun-sot ==> (=)) (openin cstrong-operator-topology) (openin euclidean)⟩
  unfolding rel-fun-def rel-set-def open-openin [symmetric] cr-cblinfun-sot-def
  by (transfer, intro allI impI arg-cong[of - - openin x for x]) blast
next
  fix U :: ⟨('a ⇒CL 'b) set⟩
  assume ⟨openin cstrong-operator-topology U⟩
  show ⟨Domainp (rel-set cr-cblinfun-sot) U⟩
  by (simp add: Domainp-set)
next
  fix U :: ⟨('a, 'b) cblinfun-sot set⟩
  assume ⟨openin euclidean U⟩
  show ⟨Rangep (rel-set cr-cblinfun-sot) U⟩
  by (simp add: Rangep-set)
qed

lemma openin-cstrong-operator-topology: ⟨openin cstrong-operator-topology U ⟷ (∃ V. open V ∧ U = (*V) -' V)⟩
  by (simp add: cstrong-operator-topology-def openin-pullback-topology)

lemma cstrong-operator-topology-plus-cont: ⟨LIM (x,y) nhdsin cstrong-operator-topology a ×F nhdsin cstrong-operator-topology b.
  x + y :=> nhdsin cstrong-operator-topology (a + b)⟩
  unfolding cstrong-operator-topology-def
  by (rule pullback-topology-bi-cont[where f'=plus])
  (auto simp: case-prod-unfold tendsto-add-Pair cblinfun.add-left)

instance cblinfun-sot :: (complex-normed-vector, complex-normed-vector) topological-group-add
proof intro-classes
  show ⟨((λx. fst x + snd x) ⟶ a + b) (nhds a ×F nhds b)⟩ for a b :: ⟨('a,'b) cblinfun-sot⟩
  apply transfer
  using cstrong-operator-topology-plus-cont
  by (auto simp: case-prod-unfold)

  have *: ⟨continuous-map cstrong-operator-topology cstrong-operator-topology uminus⟩
  apply (subst continuous-on-cstrong-operator-topo-iff-coordinatewise)
  apply (rewrite at ⟨(λy. - y *V x)⟩ in ⟨∀x. ∃⟩ to ⟨(λy. y *V - x)⟩ DEADID.rel-mono-strong)
  by (auto simp: cstrong-operator-topology-continuous-evaluation cblinfun.minus-left cblinfun.minus-right)

  show ⟨(uminus ⟶ - a) (nhds a)⟩ for a :: ⟨('a,'b) cblinfun-sot⟩
  apply (subst tendsto-at-iff-tendsto-nhds[symmetric])
  apply (subst isCont-def[symmetric])
  apply (rule continuous-on-interior[where S=UNIV])
  apply (subst continuous-map-iff-continuous2[symmetric])

```



```

    apply transfer
    using * by auto
qed

lemma continuous-map-left-comp-sot[continuous-intros]:
  fixes b :: ⟨'b::complex-normed-vector ⇒CL 'c::complex-normed-vector⟩
    and f :: ⟨'a ⇒ 'd::complex-normed-vector ⇒CL 'b⟩
  assumes ⟨continuous-map T cstrong-operator-topology f⟩
  shows ⟨continuous-map T cstrong-operator-topology (λx. b oCL f x)⟩
proof -
  have *: ⟨open B ⇒ open ((*V) b - ' B)⟩ for B
    by (simp add: continuous-open-vimage)
  have **: ⟨((λa. b *V a ψ) - ' B ∩ UNIV) = (PiE UNIV (λi. if i=ψ then (λa. b *V a) - ' B
    else UNIV))⟩
    for ψ :: 'd and B
    by (auto simp: PiE-def Pi-def)
  have *: ⟨continuous-on UNIV (λ(a::'d ⇒ 'b). b *V (a ψ))⟩ for ψ
    unfolding continuous-on-open-vimage[OF open-UNIV]
    apply (intro allI impI)
    apply (subst **)
    apply (rule open-PiE)
    using * by auto
  have *: ⟨continuous-on UNIV (λ(a::'d ⇒ 'b) ψ. b *V a ψ)⟩
    apply (rule continuous-on-coordinatewise-then-product)
    by (rule *)
  have ⟨continuous-map cstrong-operator-topology cstrong-operator-topology
    (λx :: 'd ⇒CL 'b. b oCL x)⟩
    unfolding cstrong-operator-topology-def
    apply (rule continuous-map-pullback')
  subgoal
    apply (subst asm-rl[of ⟨(*V) ∘ (oCL) b = (λa x. b *V (a x)) ∘ (*V)⟩])
    subgoal by force
    subgoal by (rule continuous-map-pullback) (use * in auto)
  done
  subgoal using * by auto
done
from continuous-map-compose[OF assms this, unfolded o-def]
show ?thesis
  by -
qed

```

```

lemma continuous-cstrong-operator-topology-plus[continuous-intros]:
  assumes ⟨continuous-map T cstrong-operator-topology f⟩
  assumes ⟨continuous-map T cstrong-operator-topology g⟩
  shows ⟨continuous-map T cstrong-operator-topology (λx. f x + g x)⟩
  using assms
  by (auto intro!: continuous-map-add
    simp: continuous-on-cstrong-operator-topo-iff-coordinatewise cblinfun.add-left)

```

lemma *continuous-cstrong-operator-topology-uminus*[*continuous-intros*]:
assumes $\langle \text{continuous-map } T \text{ cstrong-operator-topology } f \rangle$
shows $\langle \text{continuous-map } T \text{ cstrong-operator-topology } (\lambda x. - f x) \rangle$
using *assms*
by (*auto simp add: continuous-on-cstrong-operator-topo-iff-coordinatewise cblinfun.minus-left*)

lemma *continuous-cstrong-operator-topology-minus*[*continuous-intros*]:
assumes $\langle \text{continuous-map } T \text{ cstrong-operator-topology } f \rangle$
assumes $\langle \text{continuous-map } T \text{ cstrong-operator-topology } g \rangle$
shows $\langle \text{continuous-map } T \text{ cstrong-operator-topology } (\lambda x. f x - g x) \rangle$
apply (*subst diff-conv-add-uminus*)
by (*intro continuous-intros assms*)

lemma *continuous-map-right-comp-sot*[*continuous-intros*]:
assumes $\langle \text{continuous-map } T \text{ cstrong-operator-topology } f \rangle$
shows $\langle \text{continuous-map } T \text{ cstrong-operator-topology } (\lambda x. f x \circ_{CL} a) \rangle$
apply (*rule continuous-map-compose[OF assms, unfolded o-def]*)
by (*simp add: continuous-on-cstrong-operator-topo-iff-coordinatewise cstrong-operator-topology-continuous-evaluation*)

lemma *continuous-map-scaleC-sot*[*continuous-intros*]:
assumes $\langle \text{continuous-map } T \text{ cstrong-operator-topology } f \rangle$
shows $\langle \text{continuous-map } T \text{ cstrong-operator-topology } (\lambda x. c *_C f x) \rangle$
apply (*subst asm-rl[of $\langle \text{scaleC } c = (o_{CL}) (c *_C \text{id-cblinfun}) \rangle]$)*)
apply *auto[1]*
using *assms* **by** (*rule continuous-map-left-comp-sot*)

lemma *continuous-scaleC-sot*[*continuous-intros*]:
fixes $f :: \langle 'a::\text{topological-space} \Rightarrow (-,-) \text{cblinfun-sot} \rangle$
assumes $\langle \text{continuous-on } X f \rangle$
shows $\langle \text{continuous-on } X (\lambda x. c *_C f x) \rangle$
apply (*rule continuous-on-compose[OF assms, unfolded o-def]*)
apply (*rule continuous-on-subset[rotated, where s=UNIV], simp*)
apply (*subst continuous-map-iff-continuous2[symmetric]*)
apply *transfer*
apply (*rule continuous-map-scaleC-sot*)
by *simp*

lemma *sot-closure-is-csubspace*[*simp*]:
fixes $A::(\text{'a}::\text{complex-normed-vector}, \text{'b}::\text{complex-normed-vector}) \text{cblinfun-sot set}$
assumes $\langle \text{csubspace } A \rangle$
shows $\langle \text{csubspace } (\text{closure } A) \rangle$
proof (*rule complex-vector.subspaceI*)
include *lattice-syntax*
show $0: \langle 0 \in \text{closure } A \rangle$
by (*simp add: assms closure-def complex-vector.subspace-0*)
show $\langle x + y \in \text{closure } A \rangle$ **if** $\langle x \in \text{closure } A \rangle \langle y \in \text{closure } A \rangle$ **for** $x y$
proof –

```

define FF where  $\langle FF = ((nhds\ x \sqcap principal\ A) \times_F (nhds\ y \sqcap principal\ A)) \rangle$ 
have nt:  $\langle FF \neq bot \rangle$ 
  by (simp add: prod-filter-eq-bot that(1) that(2) FF-def flip: closure-nhds-principal)
have  $\langle \forall_F\ x\ in\ FF.\ fst\ x \in A \rangle$ 
  unfolding FF-def
  by (smt (verit, ccfv-SIG) eventually-prod-filter fst-conv inf-sup-ord(2) le-principal)
moreover have  $\langle \forall_F\ x\ in\ FF.\ snd\ x \in A \rangle$ 
  unfolding FF-def
  by (smt (verit, ccfv-SIG) eventually-prod-filter snd-conv inf-sup-ord(2) le-principal)
ultimately have FF-plus:  $\langle \forall_F\ x\ in\ FF.\ fst\ x + snd\ x \in A \rangle$ 
  by (smt (verit, best) assms complex-vector.subspace-add eventually-elim2)

have  $\langle (fst \longrightarrow x) ((nhds\ x \sqcap principal\ A) \times_F (nhds\ y \sqcap principal\ A)) \rangle$ 
  apply (simp add: filterlim-def)
  using filtermap-fst-prod-filter
  using le-inf-iff by blast
moreover have  $\langle (snd \longrightarrow y) ((nhds\ x \sqcap principal\ A) \times_F (nhds\ y \sqcap principal\ A)) \rangle$ 
  apply (simp add: filterlim-def)
  using filtermap-snd-prod-filter
  using le-inf-iff by blast
ultimately have  $\langle (id \longrightarrow (x,y))\ FF \rangle$ 
  by (simp add: filterlim-def nhds-prod prod-filter-mono FF-def)

moreover note tendsto-add-Pair[of x y]
ultimately have  $\langle ((\lambda x.\ fst\ x + snd\ x) \circ id) \longrightarrow (\lambda x.\ fst\ x + snd\ x)\ (x,y)\ FF \rangle$ 
  unfolding filterlim-def nhds-prod
  by (smt (verit, best) filterlim-compose filterlim-def filterlim-filtermap fst-conv snd-conv
tendsto-compose-filtermap)

then have  $\langle (\lambda x.\ fst\ x + snd\ x) \longrightarrow (x+y)\ FF \rangle$ 
  by simp
then show  $\langle x + y \in closure\ A \rangle$ 
  using nt FF-plus by (rule limit-in-closure)
qed
show  $\langle c *_C\ x \in closure\ A \rangle$  if  $\langle x \in closure\ A \rangle$  for  $x\ c$ 
proof (cases c = 0)
  case False
  have  $(*_C)\ c \text{ ' } A \subseteq closure\ A$ 
    using csubspace-scaleC-invariant[of c A] assms False closure-subset[of A] by auto
  hence  $(*_C)\ c \text{ ' } closure\ A \subseteq closure\ A$ 
    by (intro image-closure-subset) (auto intro!: continuous-intros)
  thus ?thesis
    using that by blast
qed (use 0 in auto)
qed

lemma limitin-cstrong-operator-topology:
 $\langle limitin\ cstrong\ operator\ topology\ f\ l\ F \longleftrightarrow (\forall i.\ ((\lambda j.\ f\ j *_V\ i) \longrightarrow l *_V\ i)\ F) \rangle$ 
by (simp add: cstrong-operator-topology-def limitin-pullback-topology)

```

tendsto-coordinatewise)

lemma *cstrong-operator-topology-in-closureI*:

assumes $\langle \bigwedge M \varepsilon. \varepsilon > 0 \implies \text{finite } M \implies \exists a \in A. \forall v \in M. \text{norm } ((b-a) *_V v) \leq \varepsilon \rangle$

shows $\langle b \in \text{cstrong-operator-topology closure-of } A \rangle$

proof –

define $F :: \langle ('a \text{ set} \times \text{real}) \text{ filter} \rangle$ **where** $\langle F = \text{finite-subsets-at-top UNIV} \times_F \text{at-right } 0 \rangle$

obtain f **where** $fA: \langle f M \varepsilon \in A \rangle$ **and** $f: \langle v \in M \implies \text{norm } ((f M \varepsilon - b) *_V v) \leq \varepsilon \rangle$ **if** $\langle \text{finite } M \rangle$ **and** $\langle \varepsilon > 0 \rangle$ **for** $M \varepsilon v$

apply *atomize-elim*

apply (*intro allI choice2*)

using *assms*

by (*metis cblinfun.diff-left norm-minus-commute*)

have $F\text{-props}: \langle \forall_F (M, \varepsilon) \text{ in } F. \text{finite } M \wedge \varepsilon > 0 \rangle$

by (*auto intro!: eventually-prodI simp: F-def case-prod-unfold eventually-at-right-less*)

then have $\text{in}A: \langle \forall_F (M, \varepsilon) \text{ in } F. f M \varepsilon \in A \rangle$

apply (*rule eventually-rev-mp*)

using fA **by** (*auto intro!: always-eventually*)

have $\langle \text{limitin cstrong-operator-topology (case-prod } f) b F \rangle$

proof –

have $\langle \forall_F (M, \varepsilon) \text{ in } F. \text{norm } (f M \varepsilon *_V v - b *_V v) < e \rangle$ **if** $\langle e > 0 \rangle$ **for** $e v$

proof –

have $1: \langle \forall_F (M, \varepsilon) \text{ in } F. (\text{finite } M \wedge v \in M) \wedge (\varepsilon > 0 \wedge \varepsilon < e) \rangle$

apply (*unfold F-def case-prod-unfold, rule eventually-prodI*)

using *eventually-at-right that*

by (*auto simp add: eventually-finite-subsets-at-top*)

have $2: \langle \text{norm } (f M \varepsilon *_V v - b *_V v) < e \rangle$ **if** $\langle (\text{finite } M \wedge v \in M) \wedge (\varepsilon > 0 \wedge \varepsilon < e) \rangle$

for $M \varepsilon$

by (*smt (verit) cblinfun.diff-left f that*)

show *?thesis*

using 1 **apply** (*rule eventually-mono*)

using 2 **by** *auto*

qed

then have $\langle ((\lambda(M, \varepsilon). f M \varepsilon *_V v) \longrightarrow b *_V v) F \rangle$ **for** v

by (*simp add: tendsto-iff dist-norm case-prod-unfold*)

then show *?thesis*

by (*simp add: case-prod-unfold limitin-cstrong-operator-topology*)

qed

then show *?thesis*

apply (*rule limitin-closure-of*)

using $\text{in}A$ **by** (*auto simp: F-def case-prod-unfold prod-filter-eq-bot*)

qed

lemma *sot-weaker-than-norm-limitin*: $\langle \text{limitin cstrong-operator-topology } a A F \rangle$ **if** $\langle (a \longrightarrow A) F \rangle$

proof –

from that have $\langle (\lambda x. a x *_V \psi) \longrightarrow A \psi \rangle F \rangle$ **for** ψ
by *(auto intro!: cblinfun.tendsto)*
then show *?thesis*
by *(simp add: limitin-cstrong-operator-topology)*
qed

lemma *[transfer-rule]:*
includes *lifting-syntax*
shows $\langle (rel\text{-}set\ cr\text{-}cblinfun\text{-}sot \implies (=))\ csubspace\ csubspace \rangle$
unfolding *complex-vector.subspace-def*
by *transfer-prover*

lemma *[transfer-rule]:*
includes *lifting-syntax*
shows $\langle (rel\text{-}set\ cr\text{-}cblinfun\text{-}sot \implies (=))\ (closedin\ cstrong\text{-}operator\text{-}topology)\ closed \rangle$
apply *(simp add: closed-def[abs-def] closedin-def[abs-def] cstrong-operator-topology-topospace Compl-eq-Diff-UNIV)*
by *transfer-prover*

lemma *[transfer-rule]:*
includes *lifting-syntax*
shows $\langle (rel\text{-}set\ cr\text{-}cblinfun\text{-}sot \implies rel\text{-}set\ cr\text{-}cblinfun\text{-}sot)\ (Abstract\text{-}Topology.closure\text{-}of\ cstrong\text{-}operator\text{-}topology)\ closure \rangle$
apply *(subst closure-of-hull[where X=cstrong-operator-topology, unfolded cstrong-operator-topology-topospace, simplified, abs-def])*
apply *(subst closure-hull[abs-def])*
unfolding *hull-def*
by *transfer-prover*

lemma *sot-closure-is-csubspace'[simp]:*
fixes $A::('a::complex\text{-}normed\text{-}vector \Rightarrow_{CL} 'b::complex\text{-}normed\text{-}vector)\ set$
assumes $\langle csubspace\ A \rangle$
shows $\langle csubspace\ (cstrong\text{-}operator\text{-}topology\ closure\text{-}of\ A) \rangle$
using *sot-closure-is-csubspace[of $\langle Abs\text{-}cblinfun\text{-}sot\ 'A \rangle]$* *assms*
apply *(transfer fixing: A)*
by *simp*

lemma *has-sum-closed-cstrong-operator-topology:*
assumes $aA: \langle \bigwedge i. a\ i \in A \rangle$
assumes $closed: \langle closedin\ cstrong\text{-}operator\text{-}topology\ A \rangle$
assumes $subspace: \langle csubspace\ A \rangle$
assumes $has\text{-}sum: \langle \bigwedge \psi. ((\lambda i. a\ i *_V \psi)\ has\text{-}sum\ (b *_V \psi))\ I \rangle$
shows $\langle b \in A \rangle$

proof –

have $1: \langle range\ (sum\ a) \subseteq A \rangle$

proof –

have $\langle sum\ a\ X \in A \rangle$ **for** X

apply *(induction X rule:infinite-finite-induct)*

by *(auto simp add: subspace complex-vector.subspace-0 aA complex-vector.subspace-add)*

then show *?thesis*
by *auto*
qed

from *has-sum*
have $\langle (\lambda F. \sum i \in F. a \ i *_{\mathcal{V}} \psi) \longrightarrow b *_{\mathcal{V}} \psi \rangle$ (*finite-subsets-at-top I*) **for** ψ
using *has-sum-def by blast*
then have $\langle \text{limitin } \text{cstrong-operator-topology } (\lambda F. \sum i \in F. a \ i) \ b \rangle$ (*finite-subsets-at-top I*)
by (*auto simp add: limitin-cstrong-operator-topology cblinfun.sum-left*)
then show $\langle b \in A \rangle$
using *1 closed apply (rule limitin-closedin)*
by *simp*
qed

lemma *has-sum-in-cstrong-operator-topology*:
 $\langle \text{has-sum-in } \text{cstrong-operator-topology } f \ A \ l \longleftrightarrow (\forall \psi. ((\lambda i. f \ i *_{\mathcal{V}} \psi) \text{ has-sum } (l *_{\mathcal{V}} \psi)) \ A) \rangle$
by (*simp add: cblinfun.sum-left has-sum-in-def limitin-cstrong-operator-topology has-sum-def*)

lemma *summable-sot-absI*:

fixes $b :: \langle 'a \Rightarrow 'b :: \text{complex-normed-vector} \Rightarrow_{\mathcal{CL}} 'c :: \text{hilbert-space} \rangle$
assumes $\langle \bigwedge f \ f. \text{finite } F \implies (\sum n \in F. \text{norm } (b \ n *_{\mathcal{V}} f)) \leq K * \text{norm } f \rangle$
shows $\langle \text{summable-on-in } \text{cstrong-operator-topology } b \ \text{UNIV} \rangle$

proof –

obtain B' **where** B' : $\langle ((\lambda n. b \ n *_{\mathcal{V}} f) \text{ has-sum } (B' \ f)) \ \text{UNIV} \rangle$ **for** f

proof (*atomize-elim, intro choice allI*)

fix f

have $\langle (\lambda n. b \ n *_{\mathcal{V}} f) \text{ abs-summable-on } \text{UNIV} \rangle$

apply (*rule nonneg-bdd-above-summable-on*)

using *assms by (auto intro!: bdd-aboveI [where M = $\langle K * \text{norm } f \rangle$])*

then show $\langle \exists l. ((\lambda n. b \ n *_{\mathcal{V}} f) \text{ has-sum } l) \ \text{UNIV} \rangle$

by (*metis abs-summable-summable summable-on-def*)

qed

have $\langle \text{bounded-clinear } B' \rangle$

proof (*intro bounded-clinearI allI*)

fix $x \ y :: 'b$ **and** $c :: \text{complex}$

from B' [*of x*] B' [*of y*]

have $\langle ((\lambda n. b \ n *_{\mathcal{V}} x + b \ n *_{\mathcal{V}} y) \text{ has-sum } B' \ x + B' \ y) \ \text{UNIV} \rangle$

by (*simp add: has-sum-add*)

with B' [*of $\langle x + y \rangle$*]

show $\langle B' \ (x + y) = B' \ x + B' \ y \rangle$

by (*metis (no-types, lifting) cblinfun.add-right has-sum-cong infsunI*)

from B' [*of x*]

have $\langle ((\lambda n. c *_{\mathcal{C}} (b \ n *_{\mathcal{V}} x)) \text{ has-sum } c *_{\mathcal{C}} B' \ x) \ \text{UNIV} \rangle$

by (*metis cblinfun-scaleC-right.rep-eq has-sum-cblinfun-apply*)

with B' [*of $\langle c *_{\mathcal{C}} x \rangle$*]

show $\langle B' \ (c *_{\mathcal{C}} x) = c *_{\mathcal{C}} B' \ x \rangle$

by (*metis (no-types, lifting) cblinfun.scaleC-right has-sum-cong infsunI*)

show $\langle \text{norm } (B' \ x) \leq \text{norm } x * K \rangle$

```

proof –
  have *: ⟨(λn. b n *V x) abs-summable-on UNIV⟩
    apply (rule nonneg-bdd-above-summable-on)
    using assms by (auto intro!: bdd-aboveI[where M=⟨K * norm x⟩])
  have ⟨norm (B' x) ≤ (∑∞n. norm (b n *V x))⟩
    using - B'[of x] apply (rule norm-has-sum-bound)
    using * summable-iff-has-sum-infsum by blast
  also have ⟨(∑∞n. norm (b n *V x)) ≤ K * norm x⟩
    using * apply (rule infsum-le-finite-sums)
    using assms by simp
  finally show ?thesis
    by (simp add: mult commute)
qed
qed
define B where ⟨B = CBlinfun B'⟩
with ⟨bounded-clinear B'⟩ have BB': ⟨B *V f = B' f⟩ for f
  by (simp add: bounded-clinear-CBlinfun-apply)
have ⟨has-sum-in cstrong-operator-topology b UNIV B⟩
  using B' by (simp add: has-sum-in-cstrong-operator-topology BB')
then show ?thesis
  using summable-on-in-def by blast
qed

declare cstrong-operator-topology-topospace[simp]

lift-definition cblinfun-compose-sot :: ⟨('a::complex-normed-vector,'b::complex-normed-vector)
cblinfun-sot ⇒ ('c::complex-normed-vector,'a) cblinfun-sot ⇒ ('c,'b) cblinfun-sot⟩
  is cblinfun-compose .

lemma isCont-cblinfun-compose-sot-right[simp]: ⟨isCont (λF. cblinfun-compose-sot F G) x⟩
  apply (rule continuous-on-interior[where S=UNIV, rotated], simp)
  apply (rule continuous-map-iff-continuous2[THEN iffD1])
  apply transfer
  by (simp add: continuous-map-right-comp-sot)

lemma isCont-cblinfun-compose-sot-left[simp]: ⟨isCont (λF. cblinfun-compose-sot G F) x⟩
  apply (rule continuous-on-interior[where S=UNIV, rotated], simp)
  apply (rule continuous-map-iff-continuous2[THEN iffD1])
  apply transfer
  by (simp add: continuous-map-left-comp-sot)

lemma additive-cblinfun-compose-sot-right[simp]: ⟨additive (λF. cblinfun-compose-sot F G)⟩
  unfolding additive-def
  apply transfer
  by (simp add: cblinfun-compose-add-left)

lemma additive-cblinfun-compose-sot-left[simp]: ⟨additive (λF. cblinfun-compose-sot G F)⟩
  unfolding additive-def
  apply transfer

```

by (simp add: cblinfun-compose-add-right)

lemma transfer-infsum-sot[transfer-rule]:

includes lifting-syntax

assumes [transfer-rule]: $\langle bi\text{-unique } R \rangle$

shows $\langle ((R \implies cr\text{-cblinfun-sot}) \implies rel\text{-set } R \implies cr\text{-cblinfun-sot}) (infsum\text{-in } cstrong\text{-operator-topology}) \text{ infsum} \rangle$

apply (simp add: infsum-euclidean-eq[abs-def, symmetric])

by transfer-prover

lemma transfer-summable-on-sot[transfer-rule]:

includes lifting-syntax

assumes [transfer-rule]: $\langle bi\text{-unique } R \rangle$

shows $\langle ((R \implies cr\text{-cblinfun-sot}) \implies rel\text{-set } R \implies (\longleftrightarrow)) (summable\text{-on-in } cstrong\text{-operator-topology}) \text{ (summable-on)} \rangle$

apply (simp add: summable-on-euclidean-eq[abs-def, symmetric])

by transfer-prover

lemma sandwich-sot-cont[continuous-intros]:

assumes $\langle continuous\text{-map } T \text{ } cstrong\text{-operator-topology } f \rangle$

shows $\langle continuous\text{-map } T \text{ } cstrong\text{-operator-topology } (\lambda x. sandwich\ A\ (f\ x)) \rangle$

apply (simp add: sandwich-apply)

by (intro continuous-intros assms)

lemma closed-map-sot-unitary-sandwich:

fixes $U :: \langle a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$

assumes $\langle unitary\ U \rangle$

shows $\langle closed\text{-map } cstrong\text{-operator-topology } cstrong\text{-operator-topology } (\lambda x. sandwich\ U\ x) \rangle$

apply (rule closed-eq-continuous-inverse-map[**where** $g = \langle sandwich\ (U^*) \rangle$, **THEN** iffD2])

using assms

by (auto intro!: continuous-intros

simp flip: sandwich-compose cblinfun-apply-cblinfun-compose)

unbundle no cblinfun-syntax

end

4 Positive-Operators – Positive bounded operators

theory Positive-Operators

imports

Ordinary-Differential-Equations.Cones

Misc-Tensor-Product-BO

Strong-Operator-Topology

begin

no-notation *Infinite-Set-Sum.abs-summable-on* (**infix** *abs'-summable'-on* 50)
hide-const (**open**) *Infinite-Set-Sum.abs-summable-on*
hide-fact (**open**) *Infinite-Set-Sum.abs-summable-on-Sigma-iff*

unbundle *cblinfun-syntax*

lemma *cinner-pos-if-pos*: $\langle f \cdot_C (A *_V f) \geq 0 \rangle$ **if** $\langle A \geq 0 \rangle$
using *less-eq-cblinfun-def* that **by force**

definition *sqrt-op* :: $\langle ('a :: \text{chilbert-space} \Rightarrow_{CL} 'a) \Rightarrow ('a \Rightarrow_{CL} 'a) \rangle$ **where**
 $\langle \text{sqrt-op } a = (\text{if } (\exists b :: 'a \Rightarrow_{CL} 'a. b \geq 0 \wedge b *_O_{CL} b = a) \text{ then } (\text{SOME } b. b \geq 0 \wedge b *_O_{CL} b = a) \text{ else } 0) \rangle$

lemma *sqrt-op-nonpos*: $\langle \text{sqrt-op } a = 0 \rangle$ **if** $\langle \neg a \geq 0 \rangle$

proof –

have $\langle \neg (\exists b. b \geq 0 \wedge b *_O_{CL} b = a) \rangle$
using *positive-cblinfun-squareI* that **by blast**
then show *?thesis*
by (*auto simp add: sqrt-op-def*)

qed

lemma *generalized-Cauchy-Schwarz*:

fixes *inner A*

assumes *Apos*: $\langle A \geq 0 \rangle$

defines *inner x y* $\equiv x \cdot_C (A *_V y)$

shows $\langle \text{complex-of-real } ((\text{norm } (\text{inner } x y))^2) \leq \text{inner } x x * \text{inner } y y \rangle$

proof (*cases* $\langle \text{inner } y y = 0 \rangle$)

case *True*

have [*simp*]: $\langle \text{inner } (s *_C x) y = \text{cnj } s * \text{inner } x y \rangle$ **for** $s x y$

by (*simp add: assms(2)*)

have [*simp*]: $\langle \text{inner } x (s *_C y) = s * \text{inner } x y \rangle$ **for** $s x y$

by (*simp add: assms(2) cblinfun.scaleC-right*)

have [*simp*]: $\langle \text{inner } (x - x') y = \text{inner } x y - \text{inner } x' y \rangle$ **for** $x x' y$

by (*simp add: cinner-diff-left inner-def*)

have [*simp*]: $\langle \text{inner } x (y - y') = \text{inner } x y - \text{inner } x y' \rangle$ **for** $x y y'$

by (*simp add: cblinfun.diff-right cinner-diff-right inner-def*)

have *Re0*: $\langle \text{Re } (\text{inner } x y) = 0 \rangle$ **for** x

proof –

have *: $\langle \text{Re } (\text{inner } x y) = (\text{inner } x y + \text{inner } y x) / 2 \rangle$

by (*smt (verit, del-insts) assms(1) assms(2) cinner-adj-left cinner-commute complex-Re-numeral complex-add-cnj field-sum-of-halves numeral-One numeral-plus-numeral of-real-divide of-real-numeral one-complex.simps(1) selfadjoint-def positive-selfadjointI semiring-norm(2)*)

have $\langle 0 \leq \text{Re } (\text{inner } (x - s *_C y) (x - s *_C y)) \rangle$ **for** s

by (*metis Re-mono assms(1) assms(2) cinner-pos-if-pos zero-complex.simps(1)*)

also have $\langle \dots s = \text{Re } (\text{inner } x x) - s * 2 * \text{Re } (\text{inner } x y) \rangle$ **for** s

apply (*auto simp: True*)

by (*smt (verit, ccfv-threshold) Re-complex-of-real assms(1) assms(2) cinner-adj-right cinner-commute complex-add-cnj diff-minus-eq-add minus-complex.simps(1) positive-selfadjointI selfadjoint-def uminus-complex.sel(1)*)

```

    finally show ⟨Re (inner x y) = 0⟩
      by (metis add-le-same-cancel1 ge-iff-diff-ge-0 nonzero-eq-divide-eq not-numeral-le-zero
zero-neq-numeral)
    qed
    have ⟨Im (inner x y) = Re (inner (imaginary-unit *C x) y)⟩
      by simp
    also have ⟨... = 0⟩
      by (rule Re0)
    finally have ⟨inner x y = 0⟩
      using Re0[of x]
      using complex-eq-iff zero-complex.simps(1) zero-complex.simps(2) by presburger
    then show ?thesis
      by (auto simp: True)
  next
  case False
  have inner-commute: ⟨inner x y = cnj (inner y x)⟩
    by (metis Apos cinner-adj-left cinner-commute' inner-def positive-selfadjointI selfadjoint-def)
  have [simp]: cnj (inner y y) = inner y y for y
    by (metis assms(1) cinner-adj-right cinner-commute' inner-def positive-selfadjointI selfad-
joint-def)
  define r where r = cnj (inner x y) / inner y y
  have 0 ≤ inner (x - scaleC r y) (x - scaleC r y)
    by (simp add: Apos inner-def cinner-pos-if-pos)
  also have ... = inner x x - r * inner x y - cnj r * inner y x + r * cnj r * inner y y
    unfolding cinner-diff-left cinner-diff-right cinner-scaleC-left cinner-scaleC-right inner-def
    by (smt (verit, ccfv-threshold) cblinfun.diff-right cblinfun.scaleC-right cblinfun-cinner-right.rep-eq
cinner-scaleC-left cinner-scaleC-right diff-add-eq diff-diff-eq2 mult.assoc)
  also have ... = inner x x - inner y x * cnj r
    unfolding r-def by auto
  also have ... = inner x x - inner x y * cnj (inner x y) / inner y y
    unfolding r-def
    by (metis assms(1) assms(2) cinner-adj-right cinner-commute complex-cnj-divide mult.commute
positive-selfadjointI times-divide-eq-left selfadjoint-def)
  finally have 0 ≤ inner x x - inner x y * cnj (inner x y) / inner y y .
  hence inner x y * cnj (inner x y) / inner y y ≤ inner x x
    by (simp add: le-diff-eq)
  hence ⟨(norm (inner x y)) ^ 2 / inner y y ≤ inner x x⟩
    using complex-norm-square by presburger
  then show ?thesis
    by (metis False assms(1) assms(2) cinner-pos-if-pos mult-right-mono nonzero-eq-divide-eq)
  qed

lemma sandwich-pos[intro]: ⟨sandwich b a ≥ 0⟩ if ⟨a ≥ 0⟩
  by (metis (no-types, opaque-lifting) positive-cblinfunI cblinfun-apply-cblinfun-compose cin-
ner-adj-left cinner-pos-if-pos sandwich-apply that)

lemma cblinfun-power-pos: ⟨cblinfun-power a n ≥ 0⟩ if ⟨a ≥ 0⟩
proof (cases ⟨even n⟩)
  case True

```

```

have ⟨0 ≤ (cblinfun-power a (n div 2))* oCL (cblinfun-power a (n div 2))⟩
  using positive-cblinfun-squareI by blast
also have ⟨... = cblinfun-power a (n div 2 + n div 2)⟩
  by (metis cblinfun-power-adj cblinfun-power-compose positive-selfadjointI that selfadjoint-def)
also from True have ⟨... = cblinfun-power a n⟩
  by (metis add-self-div-2 div-plus-div-distrib-dvd-right)
finally show ?thesis
  by -
next
case False
have ⟨0 ≤ sandwich (cblinfun-power a (n div 2)) a⟩
  using ⟨a ≥ 0⟩ by (rule sandwich-pos)
also have ⟨... = cblinfun-power a (n div 2 + 1 + n div 2)⟩
  unfolding sandwich-apply
  by (metis (no-types, lifting) One-nat-def cblinfun-compose-id-right cblinfun-power-0 cblin-
fun-power-Suc' cblinfun-power-adj cblinfun-power-compose positive-selfadjointI that selfadjoint-def)
also from False have ⟨... = cblinfun-power a n⟩
  by (smt (verit, del-insts) Suc-1 add.commute add.left-commute add-mult-distrib2 add-self-div-2
nat.simps(3) nonzero-mult-div-cancel-left odd-two-times-div-two-succ)
finally show ?thesis
  by -
qed

```

lemma *sqrt-op-existence*:

```

fixes A :: ⟨'a::chilbert-space ⇒CL 'a::chilbert-space⟩
assumes Apos: ⟨A ≥ 0⟩
shows ⟨∃ B. B ≥ 0 ∧ B oCL B = A ∧ (∀ F. A oCL F = F oCL A ⟶ B oCL F = F oCL B)
  ∧ B ∈ closure (cspan (range (cblinfun-power A)))⟩

```

proof –

```

define k S where ⟨k = norm A⟩ and ⟨S = A /R k - id-cblinfun⟩

```

```

have ⟨S ≤ 0⟩

```

```

proof (rule cblinfun-leI)

```

```

fix x :: 'a assume [simp]: ⟨norm x = 1⟩

```

```

with assms have aux1: ⟨complex-of-real (inverse (norm A)) * (x •C (A *V x)) ≤ 1⟩

```

```

by (smt (verit, del-insts) Reals-cnj-iff cinner-adj-left cinner-commute cinner-scaleR-left cin-

```

```

ner-scaleR-right cmod-Re complex-inner-class.Cauchy-Schwarz-ineq2 left-inverse less-eq-complex-def

```

```

linordered-field-class.inverse-nonnegative-iff-nonnegative mult-cancel-left2 mult-left-mono norm-cblinfun

```

```

norm-ge-zero norm-mult norm-of-real norm-one positive-selfadjointI reals-zero-comparable zero-less-one-class.zero-le-

```

```

selfadjoint-def)

```

```

show ⟨x •C (S *V x) ≤ x •C (0 *V x)⟩

```

```

by (auto simp: S-def cinner-diff-right cblinfun.diff-left scaleR-scaleC cdot-square-norm k-def
complex-of-real-mono-iff[where y=1, simplified])

```

```

  simp flip: assms of-real-inverse of-real-power of-real-mult power-mult-distrib power-inverse

```

```

  intro!: power-le-one aux1)

```

qed

```

have [simp]: ⟨S* = S⟩

```

using $\langle S \leq 0 \rangle$ *adj-0 comparable-selfadjoint' selfadjoint-def* **by** *blast*
have $\langle - \text{id-cblinfun} \leq S \rangle$
by (*simp add: S-def assms k-def scaleR-nonneg-nonneg*)
then have $\langle \text{norm } (S *_V f) \leq \text{norm } f \rangle$ **for** f
proof –
have 1: $\langle - S \geq 0 \rangle$
by (*simp add: $\langle S \leq 0 \rangle$*)
have 2: $\langle f \cdot_C (- S *_V f) \leq f \cdot_C f \rangle$
by (*metis $\langle - \text{id-cblinfun} \leq S \rangle$ id-cblinfun-apply less-eq-cblinfun-def minus-le-iff*)

have $\langle (\text{norm } (S *_V f))^4 = \text{complex-of-real } ((\text{cmod } ((- S *_V f) \cdot_C (- S *_V f)))^2) \rangle$
apply (*auto simp: power4-eq-xxxx cblinfun.minus-left complex-of-real-cmod power2-eq-square*
simp flip: power2-norm-eq-cinner)
by (*smt (verit, ccfv-SIG) complex-of-real-cmod mult.assoc norm-ge-zero norm-mult norm-of-real*
of-real-mult)
also have $\langle \dots \leq (- S *_V f) \cdot_C (- S *_V - S *_V f) * (f \cdot_C (- S *_V f)) \rangle$
apply (*rule generalized-Cauchy-Schwarz[where $A = \langle -S \rangle$ and $x = \langle -S *_V f \rangle$ and $y = f$]*)
by (*fact 1*)
also have $\langle \dots \leq (- S *_V f) \cdot_C (- S *_V - S *_V f) * (f \cdot_C f) \rangle$
using 2 **apply** (*rule mult-left-mono*)
using 1 *cinner-pos-if-pos* **by** *blast*
also have $\langle \dots \leq (- S *_V f) \cdot_C (- S *_V f) * (f \cdot_C f) \rangle$
apply (*rule mult-right-mono*)
apply (*metis $\langle - \text{id-cblinfun} \leq S \rangle$ id-cblinfun-apply less-eq-cblinfun-def neg-le-iff-le verit-minus-simplify(4)*)
by *simp*
also have $\langle \dots = (\text{norm } (-S *_V f))^2 * (\text{norm } f)^2 \rangle$
by (*simp add: cdot-square-norm*)
also have $\langle \dots = (\text{norm } (S *_V f))^2 * (\text{norm } f)^2 \rangle$
by (*simp add: cblinfun.minus-left*)
finally have $\langle \text{norm } (S *_V f) ^ 4 \leq (\text{norm } (S *_V f))^2 * (\text{norm } f)^2 \rangle$
using *complex-of-real-mono-iff* **by** *blast*
then have $\langle (\text{norm } (S *_V f))^2 \leq (\text{norm } f)^2 \rangle$
by (*smt (verit, best) $\langle \text{complex-of-real } (\text{norm } (S *_V f) ^ 4) = \text{complex-of-real } ((\text{cmod } ((- S *_V f) \cdot_C (- S *_V f)))^2) \rangle$ cblinfun.real.minus-left cinner-ge-zero cmod-Re mult-cancel-left*
mult-left-mono norm-minus-cancel of-real-eq-iff power2-eq-square power2-norm-eq-cinner' zero-less-norm-iff)
then show $\langle \text{norm } (S *_V f) \leq \text{norm } f \rangle$
by *auto*
qed
then have *norm-Snf*: $\langle \text{norm } (\text{cblinfun-power } S n *_V f) \leq \text{norm } f \rangle$ **for** $f n$
by (*induction n, auto simp: cblinfun-power-Suc' intro: order.trans*)
have *fSnf*: $\langle \text{cmod } (f \cdot_C (\text{cblinfun-power } S n *_V f)) \leq \text{cmod } (f \cdot_C f) \rangle$ **for** $f n$
by (*smt (z3) One-nat-def Re-complex-of-real Suc-1 cdot-square-norm cinner-ge-zero cmod-Re*
complex-inner-class.Cauchy-Schwarz-ineq2 mult.commute mult-cancel-right1 mult-left-mono norm-Snf
norm-ge-zero power-0 power-Suc)
from *norm-Snf* **have** *norm-Sn*: $\langle \text{norm } (\text{cblinfun-power } S n) \leq 1 \rangle$ **for** n
apply (*rule-tac norm-cblinfun-bound*)
by *auto*
define b **where** $\langle b = (\lambda n. (1/2 \text{ gchoose } n) *_R \text{cblinfun-power } S n) \rangle$
define $B0 B$ **where** $\langle B0 = \text{infsup } b \text{ UNIV} \rangle$ **and** $\langle B = \text{sqrt } k *_R B0 \rangle$

```

have sum-norm-b: ⟨(∑ n∈F. norm (b n)) ≤ 3⟩ (is ⟨?lhs ≤ ?rhs⟩) if ⟨finite F⟩ for F
proof -
  have [simp]: ⟨[1 / 2 :: real] = 1⟩
  by (simp add: ceiling-eq-iff)
  from ⟨finite F⟩ obtain d where ⟨F ⊆ {...d}⟩ and [simp]: ⟨d > 0⟩
  by (metis Icc-subset-Iic-iff atLeast0AtMost bot-nat-0.extremum bot-nat-0.not-eq-extremum
dual-order.trans finite-nat-iff-bounded-le less-one)

  have ⟨?lhs = (∑ n∈F. norm ((1 / 2 gchoose n) *R (cblinfun-power S n)))⟩
  by (simp add: b-def scaleR-cblinfun.rep-eq)
  also have ⟨... ≤ (∑ n∈F. abs ((1 / 2 gchoose n)))⟩
  apply (auto intro!: sum-mono)
  using norm-Sn
  by (metis norm-cmul-rule-thm norm-scaleR verit-prod-simplify(2))
  also have ⟨... ≤ (∑ n≤d. abs (1/2 gchoose n))⟩
  using ⟨F ⊆ {...d}⟩ by (auto intro!: mult-right-mono sum-mono2)
  also have ⟨... = (2 - (-1) ^ d * (-1 / 2) gchoose d)⟩
  apply (subst gbinomial-sum-lower-abs)
  by auto
  also have ⟨... ≤ (2 + norm (-1/2) gchoose d :: real)⟩
  apply (auto intro!: mult-right-mono)
  by (smt (verit) left-minus-one-mult-self mult.assoc mult-minus-left power2-eq-iff power2-eq-square)
  also have ⟨... ≤ 3⟩
  apply (subgoal-tac ⟨abs (-1/2) gchoose d :: real⟩ ≤ 1)
  apply (metis add-le-cancel-left is-num-normalize(1) mult commute mult-left-mono norm-ge-zero
numeral-Bit0 numeral-Bit1 one-add-one real-norm-def)
  apply (rule abs-gbinomial-leq1)
  by auto
  finally show ?thesis
  by -
qed

have has-sum-b: ⟨(b has-sum B0) UNIV⟩
  apply (auto intro!: has-sum-infsum abs-summable-summable[where f=b] bdd-aboveI[where
M=3] simp: B0-def abs-summable-iff-bdd-above)
  using sum-norm-b
  by simp

have ⟨B0 ≥ 0⟩
proof (rule positive-cblinfunI)
  fix f :: 'a assume [simp]: ⟨norm f = 1⟩
  from has-sum-b
  have sum1: ⟨(λn. f •C (b n *V f)) summable-on UNIV⟩
  apply (intro summable-on-cinner-left summable-on-cblinfun-apply-left)
  by (simp add: has-sum-imp-summable)
  have sum2: ⟨(λx. - (complex-of-real |1 / 2 gchoose x| * (f •C f))) summable-on UNIV -
{0}⟩
  apply (rule abs-summable-summable)

```

```

using gbinomial-abs-summable-1[of ⟨1/2⟩]
by (auto simp add: cnorm-eq-1[THEN iffD1])
from sum1 have sum3: ⟨(λn. complex-of-real (1 / 2 gchoose n) * (f •C (cblinfun-power S n
*_V f))) summable-on UNIV - {0}⟩
unfolding b-def
by (metis (no-types, lifting) cinner-scaleR-right finite.emptyI finite-insert
scaleR-cblinfun.rep-eq summable-on-cofin-subset summable-on-cong)

have aux: ⟨a ≥ - b⟩ if ⟨norm a ≤ norm b⟩ and ⟨a ∈ ℝ⟩ and ⟨b ≥ 0⟩ for a b :: complex
using cmod-eq-Re complex-is-Real-iff less-eq-complex-def that(1) that(2) that(3) by force

from has-sum-b
have ⟨f •C (B0 *_V f) = (∑∞n. f •C (b n *_V f))⟩
by (metis B0-def infsum-cblinfun-apply-left infsum-cinner-left summable-on-cblinfun-apply-left
summable-on-def)
moreover have ⟨... = (∑∞n∈UNIV-⟨0⟩. f •C (b n *_V f)) + f •C (b 0 *_V f)⟩
apply (subst infsum-Diff)
using sum1 by auto
moreover have ⟨... = f •C (b 0 *_V f) + (∑∞n∈UNIV-⟨0⟩. f •C ((1/2 gchoose n) *_R
cblinfun-power S n *_V f))⟩
unfolding b-def by simp
moreover have ⟨... = f •C (b 0 *_V f) + (∑∞n∈UNIV-⟨0⟩. of-real (1/2 gchoose n) * (f
•C (cblinfun-power S n *_V f)))⟩
by (simp add: scaleR-cblinfun.rep-eq)
moreover have ⟨... ≥ f •C (b 0 *_V f) - (∑∞n∈UNIV-⟨0⟩. of-real (abs (1/2 gchoose
n)) * (f •C f))⟩ (is ⟨- ≥ ...⟩)
proof -
have *: ⟨- (complex-of-real (abs (1 / 2 gchoose x)) * (f •C f))
≤ complex-of-real (1 / 2 gchoose x) * (f •C (cblinfun-power S x *_V f))⟩ for x
apply (rule aux)
by (auto simp: cblinfun-power-adj norm-mult fSnf selfadjoint-def
intro!: cinner-real cinner-selfadjoint-real mult-left-mono Reals-mult mult-nonneg-nonneg)
show ?thesis
apply (subst diff-conv-add-uminus) apply (rule add-left-mono)
apply (subst infsum-uminus[symmetric]) apply (rule infsum-mono-complex)
apply (rule sum2)
apply (rule sum3)
by (rule *)
qed
moreover have ⟨... = f •C (b 0 *_V f) - (∑∞n∈UNIV-⟨0⟩. of-real (abs (1/2 gchoose
n))) * (f •C f)⟩
by (simp add: infsum-cmult-left')
moreover have ⟨... = of-real (1 - (∑∞n∈UNIV-⟨0⟩. (abs (1/2 gchoose n)))) * (f •C
f)⟩
by (simp add: b-def left-diff-distrib infsum-of-real)
moreover have ⟨... ≥ 0 * (f •C f)⟩ (is ⟨- ≥ ...⟩)
apply (auto intro!: mult-nonneg-nonneg)
using gbinomial-abs-has-sum-1[where a=⟨1/2⟩]
by (auto simp add: infsumI)

```

```

moreover have  $\langle \dots = 0 \rangle$ 
  by simp
ultimately show  $\langle f \cdot_C (B0 *_{\mathcal{V}} f) \geq 0 \rangle$ 
  by force
qed
then have  $\langle B \geq 0 \rangle$ 
  by (simp add: B-def k-def scaleR-nonneg-nonneg)
then have  $\langle B = B* \rangle$ 
  by (simp add: positive-selfadjointI[unfolded selfadjoint-def])
have  $\langle B0 \circ_{CL} B0 = id\text{-cblinfun} + S \rangle$ 
proof (rule cblinfun-cinner-eqI)
  fix  $\psi$ 
  define  $s$  bb where  $\langle s = \psi \cdot_C ((B0 \circ_{CL} B0) *_{\mathcal{V}} \psi) \rangle$  and  $\langle bb\ k = (\sum_{n \leq k}. (b\ n *_{\mathcal{V}} \psi)) \cdot_C (b\ (k - n) *_{\mathcal{V}} \psi) \rangle$  for  $k$ 

  have  $\langle bb\ k = (\sum_{n \leq k}. of\text{-real} ((1 / 2\ gchoose\ (k - n)) * (1 / 2\ gchoose\ n)) * (\psi \cdot_C (cblinfun\text{-power}\ S\ k *_{\mathcal{V}} \psi))) \rangle$  for  $k$ 
  by (simp add: bb-def[abs-def] b-def cblinfun.scaleR-left cblinfun-power-adj mult.assoc flip: cinner-adj-right cblinfun-apply-cblinfun-compose)
  also have  $\langle \dots k = of\text{-real} (\sum_{n \leq k}. ((1 / 2\ gchoose\ n) * (1 / 2\ gchoose\ (k - n)))) * (\psi \cdot_C (cblinfun\text{-power}\ S\ k *_{\mathcal{V}} \psi)) \rangle$  for  $k$ 
  apply (subst mult.commute) by (simp add: sum-distrib-right)
  also have  $\langle \dots k = of\text{-real} (1\ gchoose\ k) * (\psi \cdot_C (cblinfun\text{-power}\ S\ k *_{\mathcal{V}} \psi)) \rangle$  for  $k$ 
  apply (simp only: atMost-atLeast0 gbinomial-Vandermonde)
  by simp
  also have  $\langle \dots k = of\text{-bool} (k \leq 1) * (\psi \cdot_C (cblinfun\text{-power}\ S\ k *_{\mathcal{V}} \psi)) \rangle$  for  $k$ 
  by (simp add: gbinomial-1)
  finally have bb-simp:  $\langle bb\ k = of\text{-bool} (k \leq 1) * (\psi \cdot_C (cblinfun\text{-power}\ S\ k *_{\mathcal{V}} \psi)) \rangle$  for  $k$ 
  by  $-$ 

have bb-sum:  $\langle bb\ \text{summable-on}\ UNIV \rangle$ 
  apply (rule summable-on-cong-neutral[where T={..1} and g=bb, THEN iffD2])
  by (auto simp: bb-simp)

from has-sum-b have bψ-sum:  $\langle (\lambda n. b\ n *_{\mathcal{V}} \psi)\ \text{summable-on}\ UNIV \rangle$ 
  by (simp add: has-sum-imp-summable summable-on-cblinfun-apply-left)

have b2-pos:  $\langle (b\ i *_{\mathcal{V}} \psi) \cdot_C (b\ j *_{\mathcal{V}} \psi) \geq 0 \rangle$  if  $\langle i \neq 0 \rangle \langle j \neq 0 \rangle$  for  $i\ j$ 
proof  $-$ 
  have gchoose-sign:  $\langle (-1) ^ (i+1) * ((1/2 :: real)\ gchoose\ i) \geq 0 \rangle$  if  $\langle i \neq 0 \rangle$  for  $i$ 
  proof  $-$ 
    obtain  $j$  where  $j: \langle Suc\ j = i \rangle$ 
    using  $\langle i \neq 0 \rangle$  not0-implies-Suc by blast
    show ?thesis
    proof (unfold j[symmetric], induction j)
      case  $0$ 
      then show ?case
        by simp
    next

```

```

    case (Suc j)
    have ⟨ $(-1)^{\wedge}(\text{Suc}(\text{Suc } j) + 1) * (1 / 2 \text{ gchoose } \text{Suc}(\text{Suc } j))$   

      =  $(-1)^{\wedge}(\text{Suc } j + 1) * (1 / 2 \text{ gchoose } \text{Suc } j) * ((-1) * (1/2 - \text{Suc } j) / (\text{Suc}$   

       $(\text{Suc } j)))$ ⟩
    apply (simp add: gbinomial-a-Suc-n)
    by (smt (verit, ccfv-threshold) divide-divide-eq-left' divide-divide-eq-right minus-divide-right)
    also have ⟨ $\dots \geq 0$ ⟩
    apply (rule mult-nonneg-nonneg)
    apply (rule Suc.IH)
    apply (rule divide-nonneg-pos)
    apply (rule mult-nonpos-nonpos)
    by auto
    finally show ?case
    by -
  qed
qed
from ⟨ $S \leq 0$ ⟩
have Sn-sign: ⟨ $\psi \cdot_C (\text{cblinfun-power } (-S) (i + j) *_V \psi) \geq 0$ ⟩
  by (auto intro!: cinner-pos-if-pos cblinfun-power-pos)
have *: ⟨ $(-1)^{\wedge}(i + (j + (i + j))) = (1::\text{complex})$ ⟩
  by (metis Parity.ring-1-class.power-minus-even even-add power-one)

have ⟨ $(b \ i *_V \psi) \cdot_C (b \ j *_V \psi)$   

  =  $\text{complex-of-real } (1 / 2 \text{ gchoose } i) * \text{complex-of-real } (1 / 2 \text{ gchoose } j)$   

  *  $(\psi \cdot_C (\text{cblinfun-power } S (i + j) *_V \psi))$ ⟩
  by (simp add: b-def cblinfun.scaleR-right cblinfun.scaleR-left cblinfun-power-adj  

    flip: cinner-adj-right cblinfun-apply-cblinfun-compose)
  also have ⟨ $\dots = \text{complex-of-real } ((-1)^{\wedge}(i+1) * (1 / 2 \text{ gchoose } i)) * \text{complex-of-real}$   

     $((-1)^{\wedge}(j+1) * (1 / 2 \text{ gchoose } j))$   

    *  $(\psi \cdot_C (\text{cblinfun-power } (-S) (i + j) *_V \psi))$ ⟩
  by (simp add: cblinfun.scaleR-left cblinfun-power-uminus * flip: power-add)
  also have ⟨ $\dots \geq 0$ ⟩
  apply (rule mult-nonneg-nonneg)
  apply (rule mult-nonneg-nonneg)
  using complex-of-real-nn-iff gchoose-sign that(1) apply blast
  using complex-of-real-nn-iff gchoose-sign that(2) apply blast
  by (fact Sn-sign)
  finally show ?thesis
  by -
qed

have ⟨ $s = (B0 *_V \psi) \cdot_C (B0 *_V \psi)$ ⟩
  by (metis ⟨ $0 \leq B0$ ⟩ cblinfun-apply-cblinfun-compose cinner-adj-left positive-selfadjointI  

  s-def selfadjoint-def)
  also have ⟨ $\dots = (\sum_{\infty n}. b \ n *_V \psi) \cdot_C (\sum_{\infty n}. b \ n *_V \psi)$ ⟩
  by (metis B0-def has-sum-b infsum-cblinfun-apply-left has-sum-imp-summable)
  also have ⟨ $\dots = (\sum_{\infty n}. bb \ n)$ ⟩
  using bψ-sum bψ-sum unfolding bb-def
  apply (rule Cauchy-cinner-product-infsum[symmetric])

```



```

using bψ-sum bψ-sum
apply (rule Cauchy-cinner-product-summable[where  $X = \langle \{0\} \rangle$  and  $Y = \langle \{0\} \rangle$ ])
using b2-pos by auto
also have  $\langle \dots = bb\ 0 + bb\ 1 \rangle$ 
apply (subst infsum-cong-neutral[where  $T = \langle \{..1\} \rangle$  and  $g = bb$ ])
by (auto simp: bb-simp)
also have  $\langle \dots = \psi \cdot_C ((id\text{-}cblinfun + S) *_V \psi) \rangle$ 
by (simp add: cblinfun-power-Suc cblinfun.add-left cinner-add-right bb-simp)
finally show  $\langle s = \psi \cdot_C ((id\text{-}cblinfun + S) *_V \psi) \rangle$ 
by –
qed
then have  $\langle B\ o_{CL}\ B = norm\ A *_C (id\text{-}cblinfun + S) \rangle$ 
apply (simp add: k-def B-def power2-eq-square scaleR-scaleC)
by (metis norm-imp-pos-and-ge of-real-power power2-eq-square real-sqrt-pow2)
also have  $\langle \dots = A \rangle$ 
by (metis (no-types, lifting) k-def S-def add commute cancel-comm-monoid-add-class.diff-cancel
diff-add-cancel norm-eq-zero of-real-1 of-real-mult right-inverse scaleC-diff-right scaleC-one scaleC-scaleC
scaleR-scaleC)
finally have  $B \geq A$ :  $\langle B\ o_{CL}\ B = A \rangle$ 
by –
have BF-comm:  $\langle B\ o_{CL}\ F = F\ o_{CL}\ B \rangle$  if  $\langle A\ o_{CL}\ F = F\ o_{CL}\ A \rangle$  for  $F$ 
proof –
have  $\langle S\ o_{CL}\ F = F\ o_{CL}\ S \rangle$ 
by (simp add: S-def that[symmetric] cblinfun-compose-minus-right cblinfun-compose-minus-left
flip: cblinfun-compose-assoc)
then have  $\langle cblinfun\ power\ S\ n\ o_{CL}\ F = F\ o_{CL}\ cblinfun\ power\ S\ n \rangle$  for  $n$ 
apply (induction n)
apply (simp-all add: cblinfun-power-Suc' cblinfun-compose-assoc)
by (simp flip: cblinfun-compose-assoc)
then have  $*$ :  $\langle b\ n\ o_{CL}\ F = F\ o_{CL}\ b\ n \rangle$  for  $n$ 
by (simp add: b-def)
have  $\langle (\sum_{\infty} n. b\ n)\ o_{CL}\ F = F\ o_{CL}\ (\sum_{\infty} n. b\ n) \rangle$ 
proof –
have [simp]:  $\langle b\ summable\ on\ UNIV \rangle$ 
using has-sum-b by (auto simp add: summable-on-def)
have  $\langle (\sum_{\infty} n. b\ n)\ o_{CL}\ F = (\sum_{\infty} n. (b\ n)\ o_{CL}\ F) \rangle$ 
apply (subst infsum-comm-additive[where  $f = \langle \lambda x. x\ o_{CL}\ F \rangle$ , symmetric])
by (auto simp: o-def isCont-cblinfun-compose-left)
also have  $\langle \dots = (\sum_{\infty} n. F\ o_{CL}\ (b\ n)) \rangle$ 
by (simp add: *)
also have  $\langle \dots = F\ o_{CL}\ (\sum_{\infty} n. b\ n) \rangle$ 
apply (subst infsum-comm-additive[where  $f = \langle \lambda x. F\ o_{CL}\ x \rangle$ , symmetric])
by (auto simp: o-def isCont-cblinfun-compose-right)
finally show ?thesis
by –
qed
then have  $\langle B0\ o_{CL}\ F = F\ o_{CL}\ B0 \rangle$ 
unfolding B0-def

```

```

    unfolding infsum-euclidean-eq[abs-def, symmetric]
    apply (transfer fixing: b F)
    by simp
  then show ?thesis
    by (auto simp: B-def)
qed
have B-closure: ⟨B ∈ closure (cspan (range (cblinfun-power A)))⟩
proof (cases ⟨k = 0⟩)
  case True
  then show ?thesis
    unfolding B-def using closure-subset complex-vector.span-zero by auto
next
  case False
  then have ⟨k ≠ 0⟩
    by –
  from has-sum-b
  have limit: ⟨sum b ⟶ B0⟩ (finite-subsets-at-top UNIV)
    by (simp add: has-sum-def)
  have ⟨cblinfun-power (A /R k - id-cblinfun) n ∈ cspan (range (cblinfun-power A))⟩ for n
  proof (induction n)
    case 0
    then show ?case
      by (auto intro!: complex-vector.span-base range-eqI[where x=0])
  next
    case (Suc n)
    define pow-n where ⟨pow-n = cblinfun-power (A /R k - id-cblinfun) n⟩
    have pow-n-span: ⟨pow-n ∈ cspan (range (cblinfun-power A))⟩
      using Suc by (simp add: pow-n-def)
    have A-pow-n-span: ⟨A oCL pow-n ∈ cspan (range (cblinfun-power A))⟩
    proof –
      from pow-n-span
      obtain F r where ⟨finite F⟩ and F-A: ⟨F ⊆ range (cblinfun-power A)⟩
        and pow-n-sum: ⟨pow-n = (∑ a∈F. r a *C a)⟩
        by (auto simp add: complex-vector.span-explicit)
      have ⟨A oCL a ∈ range (cblinfun-power A)⟩ if ⟨a ∈ F⟩ for a
      proof –
        from that obtain m where ⟨a = cblinfun-power A m⟩
          using F-A by auto
        then have ⟨A oCL a = cblinfun-power A (Suc m)⟩
          by (simp add: cblinfun-power-Suc')
        then show ?thesis
          by auto
      qed
    then have ⟨(∑ a∈F. r a *C (A oCL a)) ∈ cspan (range (cblinfun-power A))⟩
      by (meson basic-trans-rules(31) complex-vector.span-scale complex-vector.span-sum
        complex-vector.span-superset)
    moreover have ⟨A oCL pow-n = (∑ a∈F. r a *C (A oCL a))⟩
      by (simp add: pow-n-sum cblinfun-compose-sum-right flip: cblinfun.scaleC-left)
    ultimately show ?thesis

```

by *simp*
 qed
 have $\langle \text{cblinfun-power } (A /_R k - \text{id-cblinfun}) (Suc\ n) = (A\ o_{CL}\ \text{pow-n}) /_R k - \text{pow-n} \rangle$
 by (*simp add: cblinfun-power-Suc' cblinfun-compose-minus-left flip: pow-n-def*)
 also from *pow-n-span A-pow-n-span*
 have $\langle \dots \in \text{cspan } (\text{range } (\text{cblinfun-power } A)) \rangle$
 by (*auto intro!: complex-vector.span-diff complex-vector.span-scale simp: scaleR-scaleC*)
 finally show *?case*
 by –
 qed
 then have *b-range*: $\langle b\ n \in \text{cspan } (\text{range } (\text{cblinfun-power } A)) \rangle$ for *n*
 by (*simp add: b-def S-def scaleR-scaleC complex-vector.span-scale*)
 have *sum-bF*: $\langle \text{sum } b\ F \in \text{cspan } (\text{range } (\text{cblinfun-power } A)) \rangle$ if $\langle \text{finite } F \rangle$ for *F*
 using *that apply induction*
 using *b-range complex-vector.span-add complex-vector.span-zero* by *auto*
 have $\langle B0 \in \text{closure } (\text{cspan } (\text{range } (\text{cblinfun-power } A))) \rangle$
 using *limit apply (rule limit-in-closure)*
 using *sum-bF* by (*simp-all add: eventually-finite-subsets-at-top-weakI*)
 also have $\langle \dots = \text{closure } ((\lambda x. \text{inverse } (\text{sqrt } k) * _R x) \text{ `cspan } (\text{range } (\text{cblinfun-power } A))) \rangle$
 using $\langle k \neq 0 \rangle$ by (*simp add: scaleR-scaleC csubspace-scaleC-invariant*)
 also have $\langle \dots = (\lambda x. \text{inverse } (\text{sqrt } k) * _R x) \text{ `closure } (\text{cspan } (\text{range } (\text{cblinfun-power } A))) \rangle$
 by (*simp add: closure-scaleR*)
 finally show *?thesis*
 apply (*simp add: B-def image-def*)
 using $\langle k \neq 0 \rangle$ by *force*
 qed
 from $\langle B \geq 0 \rangle$ *B2A BF-comm B-closure*
 show *?thesis*
 by *metis*
 qed

lemma *wecken35hilfssatz*:

– Auxiliary lemma from [9]
 $\langle \exists P. \text{is-Proj } P \wedge (\forall F. F\ o_{CL}\ (W - T) = (W - T)\ o_{CL}\ F \longrightarrow F\ o_{CL}\ P = P\ o_{CL}\ F) \wedge (\forall f. W\ f = 0 \longrightarrow P\ f = f) \wedge (W = (2 * _C P - \text{id-cblinfun})\ o_{CL}\ T) \rangle$
 if *WT-comm*: $\langle W\ o_{CL}\ T = T\ o_{CL}\ W \rangle$ and $\langle W = W * \rangle$ and $\langle T = T * \rangle$
 and *WW-TT*: $\langle W\ o_{CL}\ W = T\ o_{CL}\ T \rangle$
 for *W T* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL}\ 'a \rangle$
proof (*rule exI, intro conjI allI impI*)
define *P* where $\langle P = \text{Proj } (\text{kernel } (W - T)) \rangle$
show $\langle \text{is-Proj } P \rangle$
 by (*simp add: P-def*)
show *thesis1*: $\langle F\ o_{CL}\ P = P\ o_{CL}\ F \rangle$ if $\langle F\ o_{CL}\ (W - T) = (W - T)\ o_{CL}\ F \rangle$ for *F*
proof –
 have *1*: $\langle F\ o_{CL}\ P = P\ o_{CL}\ F\ o_{CL}\ P \rangle$ if $\langle F\ o_{CL}\ (W - T) = (W - T)\ o_{CL}\ F \rangle$ for *F*
proof (*rule cblinfun-eqI*)

```

fix  $\psi$ 
have  $\langle P *_V \psi \in \text{space-as-set} (\text{kernel} (W - T)) \rangle$ 
  by (metis P-def Proj-range cblinfun-apply-in-image)
then have  $\langle (W - T) *_V P *_V \psi = 0 \rangle$ 
  using kernel-memberD by blast
then have  $\langle (W - T) *_V F *_V P *_V \psi = 0 \rangle$ 
  by (metis cblinfun.zero-right cblinfun-apply-cblinfun-compose that)
then have  $\langle F *_V P *_V \psi \in \text{space-as-set} (\text{kernel} (W - T)) \rangle$ 
  using kernel-memberI by blast
then have  $\langle P *_V (F *_V P *_V \psi) = F *_V P *_V \psi \rangle$ 
  using P-def Proj-fixes-image by blast
then show  $\langle (F \circ_{CL} P) *_V \psi = (P \circ_{CL} F \circ_{CL} P) *_V \psi \rangle$ 
  by simp
qed
have  $2: \langle F *_V \circ_{CL} (W - T) = (W - T) \circ_{CL} F *_V \rangle$ 
  by (metis  $\langle T = T *_V \rangle$   $\langle W = W *_V \rangle$  adj-cblinfun-compose adj-minus that)
have  $\langle F \circ_{CL} P = P \circ_{CL} F \circ_{CL} P \rangle$  and  $\langle F *_V \circ_{CL} P = P \circ_{CL} F *_V \circ_{CL} P \rangle$ 
  using  $1[OF \text{ that}] 1[OF 2]$  by auto
then show  $\langle F \circ_{CL} P = P \circ_{CL} F \rangle$ 
  by (metis P-def adj-Proj adj-cblinfun-compose cblinfun-assoc-left(1) double-adj)
qed
show thesis2:  $\langle P *_V f = f \rangle$  if  $\langle W *_V f = 0 \rangle$  for  $f$ 
proof –
  from that
  have  $\langle 0 = (W *_V f) \cdot_C (W *_V f) \rangle$ 
    by simp
  also from  $\langle W = W *_V \rangle$  have  $\langle \dots = f \cdot_C ((W \circ_{CL} W) *_V f) \rangle$ 
    by (simp add: that)
  also from WW-TT have  $\langle \dots = f \cdot_C ((T \circ_{CL} T) *_V f) \rangle$ 
    by simp
  also from  $\langle T = T *_V \rangle$  have  $\langle \dots = (T *_V f) \cdot_C (T *_V f) \rangle$ 
    by (metis cblinfun-apply-cblinfun-compose cinner-adj-left)
  finally have  $\langle T *_V f = 0 \rangle$ 
    by simp
  then have  $\langle (W - T) *_V f = 0 \rangle$ 
    by (simp add: cblinfun.diff-left that)
  then show  $\langle P *_V f = f \rangle$ 
    using P-def Proj-fixes-image kernel-memberI by blast
qed
show thesis3:  $\langle W = (2 *_C P - \text{id-cblinfun}) \circ_{CL} T \rangle$ 
proof –
  from WW-TT WT-comm have WT-binomial:  $\langle (W - T) \circ_{CL} (W + T) = 0 \rangle$ 
    by (simp add: cblinfun-compose-add-right cblinfun-compose-minus-left)
  have PWT:  $\langle P \circ_{CL} (W + T) = W + T \rangle$ 
  proof (rule cblinfun-eqI)
    fix  $\psi$ 
    from WT-binomial have  $\langle (W + T) *_V \psi \in \text{space-as-set} (\text{kernel} (W - T)) \rangle$ 
      by (metis cblinfun-apply-cblinfun-compose kernel-memberI zero-cblinfun.rep-eq)
    then show  $\langle (P \circ_{CL} (W + T)) *_V \psi = (W + T) *_V \psi \rangle$ 

```

by (metis *P-def Proj-idempotent Proj-range cblinfun-apply-cblinfun-compose cblinfun-fixes-range*)
 qed
 from *P-def* have $\langle (W - T) \circ_{CL} P = 0 \rangle$
 by (metis *Proj-range thesis1 cblinfun-apply-cblinfun-compose cblinfun-apply-in-image*
cblinfun-eqI kernel-memberD zero-cblinfun.rep-eq)
 with *PWT WT-comm thesis1* have $\langle 2 *_{C} T \circ_{CL} P = W + T \rangle$
 by (metis (no-types, lifting) *bounded-cbilinear.add-left bounded-cbilinear-cblinfun-compose*
cblinfun-compose-add-right cblinfun-compose-minus-left cblinfun-compose-minus-right eq-iff-diff-eq-0
scaleC-2)
 with *that(2) that(3)* show ?thesis
 by (smt (verit, ccfv-threshold) *P-def add-diff-cancel adj-Proj adj-cblinfun-compose adj-plus*
cblinfun-compose-id-right cblinfun-compose-minus-left cblinfun-compose-scaleC-left id-cblinfun-adjoint
scaleC-2)
 qed
 qed

lemma *sqrt-op-pos[simp]*: $\langle \text{sqrt-op } a \geq 0 \rangle$
proof (cases $\langle a \geq 0 \rangle$)
 case *True*
 from *sqrt-op-existence[OF True]*
 have *: $\langle \exists b::'a \Rightarrow_{CL} 'a. b \geq 0 \wedge b * \circ_{CL} b = a \rangle$
 by (metis *positive-selfadjointI selfadjoint-def*)
 then show ?thesis
 using * by (smt (verit, ccfv-threshold) *someI-ex sqrt-op-def*)
 next
 case *False*
 then show ?thesis
 by (simp add: *sqrt-op-nonpos*)
 qed

lemma *sqrt-op-square[simp]*:
 assumes $\langle a \geq 0 \rangle$
 shows $\langle \text{sqrt-op } a \circ_{CL} \text{sqrt-op } a = a \rangle$
proof –
 from *sqrt-op-existence[OF assms]*
 have *: $\langle \exists b::'a \Rightarrow_{CL} 'a. b \geq 0 \wedge b * \circ_{CL} b = a \rangle$
 by (metis *positive-selfadjointI selfadjoint-def*)
 have $\langle \text{sqrt-op } a \circ_{CL} \text{sqrt-op } a = (\text{sqrt-op } a) * \circ_{CL} \text{sqrt-op } a \rangle$
 by (metis *positive-selfadjointI selfadjoint-def sqrt-op-pos*)
 also have $\langle (\text{sqrt-op } a) * \circ_{CL} \text{sqrt-op } a = a \rangle$
 using * by (metis (mono-tags, lifting) *someI-ex sqrt-op-def*)
 finally show ?thesis
 by –
 qed

lemma *sqrt-op-unique*:
 — Proof follows [9]
 assumes $\langle b \geq 0 \rangle$ and $\langle b * \circ_{CL} b = a \rangle$
 shows $\langle b = \text{sqrt-op } a \rangle$

proof –
have $\langle a \geq 0 \rangle$
using *assms(2) positive-cblinfun-squareI* **by** *blast*
from *sqrt-op-existence[OF $\langle a \geq 0 \rangle$]*
obtain *sq* **where** $\langle sq \geq 0 \rangle$ **and** $\langle sq \circ_{CL} sq = a \rangle$ **and** *a-comm: $\langle a \circ_{CL} F = F \circ_{CL} a \implies sq \circ_{CL} F = F \circ_{CL} sq \rangle$* **for** *F*
by *metis*
have *eq-sq: $\langle b = sq \rangle$* **if** $\langle b \geq 0 \rangle$ **and** $\langle b^* \circ_{CL} b = a \rangle$ **for** *b*
proof –
have $\langle b \circ_{CL} a = a \circ_{CL} b \rangle$
by (*metis cblinfun-assoc-left(1) positive-selfadjointI selfadjoint-def that(1) that(2)*)
then have *b-sqrt-comm: $\langle b \circ_{CL} sq = sq \circ_{CL} b \rangle$*
using *a-comm* **by** *force*
from $\langle b \geq 0 \rangle$ **have** $\langle b = b^* \rangle$
by (*simp add: assms(1) positive-selfadjointI[unfolded selfadjoint-def]*)
have *sqrt-adj: $\langle sq = sq^* \rangle$*
by (*simp add: $\langle 0 \leq sq \rangle$ positive-selfadjointI[unfolded selfadjoint-def]*)
have *bb-sqrt: $\langle b \circ_{CL} b = sq \circ_{CL} sq \rangle$*
using $\langle b = b^* \rangle$ $\langle sq \circ_{CL} sq = a \rangle$ *that(2)* **by** *fastforce*

from *wecken35hilfssatz[OF b-sqrt-comm $\langle b = b^* \rangle$ sqrt-adj bb-sqrt]*
obtain *P* **where** $\langle is-Proj P \rangle$ **and** *b-P-sq: $\langle b = (2 *_C P - id-cblinfun) \circ_{CL} sq \rangle$*
and *P-comm: $\langle F \circ_{CL} (b - sq) = (b - sq) \circ_{CL} F \implies F \circ_{CL} P = P \circ_{CL} F \rangle$* **for** *F*
by *metis*

have *1: $\langle sandwich (id-cblinfun - P) b = (id-cblinfun - P) \circ_{CL} b \rangle$*
by (*smt (verit, del-insts) P-comm $\langle is-Proj P \rangle$ b-sqrt-comm cblinfun-assoc-left(1) cblinfun-compose-id-left cblinfun-compose-id-right cblinfun-compose-minus-left cblinfun-compose-minus-right cblinfun-compose-zero-left diff-0-right is-Proj-algebraic is-Proj-complement is-Proj-idempotent sandwich-apply*)
also have *2: $\langle \dots = - (id-cblinfun - P) \circ_{CL} sq \rangle$*
apply (*simp add: b-P-sq*)
by (*smt (verit, del-insts) $\langle 0 \leq sq \rangle$ $\langle is-Proj P \rangle$ add-diff-cancel-left' cancel-comm-monoid-add-class.diff-cancel cblinfun-compose-assoc cblinfun-compose-id-right cblinfun-compose-minus-right diff-diff-eq2 is-Proj-algebraic is-Proj-complement minus-diff-eq scaleC-2*)
also have $\langle \dots = - sandwich (id-cblinfun - P) sq \rangle$
by (*metis $\langle (id-cblinfun - P) \circ_{CL} b = - (id-cblinfun - P) \circ_{CL} sq \rangle$ calculation cblinfun-compose-uminus-left sandwich-apply*)
also have $\langle \dots \leq 0 \rangle$
by (*simp add: $\langle 0 \leq sq \rangle$ sandwich-pos*)
finally have $\langle sandwich (id-cblinfun - P) b \leq 0 \rangle$
by –
moreover from $\langle b \geq 0 \rangle$ **have** $\langle sandwich (id-cblinfun - P) b \geq 0 \rangle$
by (*simp add: sandwich-pos*)
ultimately have $\langle sandwich (id-cblinfun - P) b = 0 \rangle$
by *auto*
with *1 2* **have** $\langle (id-cblinfun - P) \circ_{CL} sq = 0 \rangle$
by (*metis add.inverse-neutral cblinfun-compose-uminus-left minus-diff-eq*)
with *b-P-sq* **show** $\langle b = sq \rangle$

by (metis (no-types, lifting) add.inverse-neutral add-diff-cancel-right' adj-cblinfun-compose cblinfun-compose-id-right cblinfun-compose-minus-left diff-0 diff-eq-diff-eq id-cblinfun-adjoint scaleC-2 sqrt-adj)
qed

from eq-sq have $\langle \text{sqrt-op } a = \text{sq} \rangle$
 by (simp add: $\langle 0 \leq a \rangle$ comparable-selfadjoint[unfolded selfadjoint-def])
 moreover from eq-sq have $\langle b = \text{sq} \rangle$
 by (simp add: assms(1) assms(2))
 ultimately show $\langle b = \text{sqrt-op } a \rangle$
 by simp
qed

lemma sqrt-op-in-closure: $\langle \text{sqrt-op } a \in \text{closure } (\text{cspan } (\text{range } (\text{cblinfun-power } a))) \rangle$
proof (cases $\langle a \geq 0 \rangle$)
 case True
 from sqrt-op-existence[OF True]
 obtain $B :: \langle 'a \Rightarrow_{CL} 'a \rangle$ where $\langle B \geq 0 \rangle$ and $\langle B \circ_{CL} B = a \rangle$
 and B-closure: $\langle B \in \text{closure } (\text{cspan } (\text{range } (\text{cblinfun-power } a))) \rangle$
 by metis
 then have $\langle \text{sqrt-op } a = B \rangle$
 by (metis positive-selfadjointI sqrt-op-unique selfadjoint-def)
 with B-closure show ?thesis
 by simp
 next
 case False
 then have $\langle \text{sqrt-op } a = 0 \rangle$
 by (simp add: sqrt-op-nonpos)
 also have $\langle 0 \in \text{closure } (\text{cspan } (\text{range } (\text{cblinfun-power } a))) \rangle$
 using closure-subset complex-vector.span-zero by blast
 finally show ?thesis
 by –
qed

lemma sqrt-op-commute:
 assumes $\langle A \geq 0 \rangle$
 assumes $\langle A \circ_{CL} F = F \circ_{CL} A \rangle$
 shows $\langle \text{sqrt-op } A \circ_{CL} F = F \circ_{CL} \text{sqrt-op } A \rangle$
 by (metis assms(1) assms(2) positive-selfadjointI sqrt-op-existence sqrt-op-unique selfadjoint-def)

lemma sqrt-op-0[simp]: $\langle \text{sqrt-op } 0 = 0 \rangle$
 apply (rule sqrt-op-unique[symmetric])
 by auto

lemma sqrt-op-scaleC:
 assumes $\langle c \geq 0 \rangle$ and $\langle a \geq 0 \rangle$
 shows $\langle \text{sqrt-op } (c *_C a) = \text{sqrt } c *_C \text{sqrt-op } a \rangle$
 apply (rule sqrt-op-unique[symmetric])

using *assms* **apply** (*auto simp: split-scaleC-pos-le positive-selfadjointI[unfolded selfadjoint-def]*)
by (*metis of-real-power power2-eq-square real-sqrt-pow2*)

definition *abs-op* :: $\langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{complex-inner} \Rightarrow 'a \Rightarrow_{CL} 'a \rangle$ **where** $\langle \text{abs-op } a = \text{sqrt-op } (a * o_{CL} a) \rangle$

lemma *abs-op-pos[simp]*: $\langle \text{abs-op } a \geq 0 \rangle$
by (*simp add: abs-op-def positive-cblinfun-squareI sqrt-op-pos*)

lemma *abs-op-0[simp]*: $\langle \text{abs-op } 0 = 0 \rangle$
unfolding *abs-op-def* **by** *auto*

lemma *abs-op-idem[simp]*: $\langle \text{abs-op } (\text{abs-op } a) = \text{abs-op } a \rangle$
by (*metis abs-op-def abs-op-pos sqrt-op-unique*)

lemma *abs-op-uminus[simp]*: $\langle \text{abs-op } (- a) = \text{abs-op } a \rangle$
by (*simp add: abs-op-def adj-uminus bounded-cbilinear.minus-left bounded-cbilinear.minus-right bounded-cbilinear-cblinfun-compose*)

lemma *selfbutter-pos[simp]*: $\langle \text{selfbutter } x \geq 0 \rangle$
by (*metis butterfly-def double-adj positive-cblinfun-squareI*)

lemma *abs-op-butterfly[simp]*: $\langle \text{abs-op } (\text{butterfly } x y) = (\text{norm } x / \text{norm } y) *_R \text{selfbutter } y \rangle$ **for**
 $x :: \langle 'a::\text{chilbert-space} \rangle$ **and** $y :: \langle 'b::\text{chilbert-space} \rangle$

proof (*cases* $\langle y=0 \rangle$)

case *False*

have $\langle \text{abs-op } (\text{butterfly } x y) = \text{sqrt-op } (\text{cinner } x x *_C \text{selfbutter } y) \rangle$

unfolding *abs-op-def* **by** *simp*

also have $\langle \dots = (\text{norm } x / \text{norm } y) *_R \text{selfbutter } y \rangle$

apply (*rule sqrt-op-unique[symmetric]*)

using *False* **by** (*auto intro!: scaleC-nonneg-nonneg simp: scaleR-scaleC power2-eq-square simp flip: power2-norm-eq-cinner*)

finally show *?thesis*

by $-$

next

case *True*

then show *?thesis*

by *simp*

qed

lemma *abs-op-nondegenerate*: $\langle a = 0 \rangle$ **if** $\langle \text{abs-op } a = 0 \rangle$

proof $-$

from *that*

have $\langle \text{sqrt-op } (a * o_{CL} a) = 0 \rangle$

by (*simp add: abs-op-def*)

then have $\langle 0 * o_{CL} 0 = (a * o_{CL} a) \rangle$

by (*metis cblinfun-compose-zero-right positive-cblinfun-squareI sqrt-op-square*)

then show $\langle a = 0 \rangle$

apply (*rule-tac op-square-nondegenerate*)
by *simp*
qed

lemma *abs-op-scaleC*: $\langle \text{abs-op } (c \cdot_C a) = |c| \cdot_C \text{abs-op } a \rangle$

proof –

define *aa* **where** $\langle aa = a \cdot_{o_{CL}} a \rangle$

have $\langle \text{abs-op } (c \cdot_C a) = \text{sqrt-op } (|c|^2 \cdot_C aa) \rangle$

by (*simp add: abs-op-def x-cnj-x aa-def*)

also have $\langle \dots = |c| \cdot_C \text{sqrt-op } aa \rangle$

by (*smt (verit, best) aa-def abs-complex-def abs-nn cblinfun-compose-scaleC-left cblinfun-compose-scaleC-right complex-cnj-complex-of-real o-apply positive-cblinfun-squareI power2-eq-square scaleC-adj scaleC-nonneg-nonneg scaleC-scaleC sqrt-op-pos sqrt-op-square sqrt-op-unique*)

also have $\langle \dots = |c| \cdot_C \text{abs-op } a \rangle$

by (*simp add: aa-def abs-op-def*)

finally show *?thesis*

by –

qed

lemma *kernel-abs-op[simp]*: $\langle \text{kernel } (\text{abs-op } a) = \text{kernel } a \rangle$

proof (*rule ccspace-eqI*)

fix *x*

have $\langle x \in \text{space-as-set } (\text{kernel } (\text{abs-op } a)) \longleftrightarrow \text{abs-op } a \ x = 0 \rangle$

using *kernel-memberD kernel-memberI* **by** *blast*

also have $\langle \dots \longleftrightarrow \text{abs-op } a \ x \cdot_C \text{abs-op } a \ x = 0 \rangle$

by *simp*

also have $\langle \dots \longleftrightarrow x \cdot_C ((\text{abs-op } a) \cdot_{o_{CL}} \text{abs-op } a) \ x = 0 \rangle$

by (*simp add: cinner-adj-right*)

also have $\langle \dots \longleftrightarrow x \cdot_C (a \cdot_{o_{CL}} a) \ x = 0 \rangle$

by (*simp add: abs-op-def positive-cblinfun-squareI positive-selfadjointI [unfolded selfadjoint-def]*)

also have $\langle \dots \longleftrightarrow a \ x \cdot_C a \ x = 0 \rangle$

by (*simp add: cinner-adj-right*)

also have $\langle \dots \longleftrightarrow a \ x = 0 \rangle$

by *simp*

also have $\langle \dots \longleftrightarrow x \in \text{space-as-set } (\text{kernel } a) \rangle$

using *kernel-memberD kernel-memberI* **by** *auto*

finally show $\langle x \in \text{space-as-set } (\text{kernel } (\text{abs-op } a)) \longleftrightarrow x \in \text{space-as-set } (\text{kernel } a) \rangle$

by –

qed

definition *polar-decomposition* **where**

– [1], 3.9 Polar Decomposition

$\langle \text{polar-decomposition } A = \text{cblinfun-extension } (\text{range } (\text{abs-op } A)) (\lambda \psi. A \cdot_V \text{inv } (\text{abs-op } A) \psi) \cdot_{o_{CL}} \text{Proj } (\text{abs-op } A \cdot_S \text{top}) \rangle$

for $A :: \langle 'a :: \text{chilbert-space} \Rightarrow_{o_{CL}} 'b :: \text{complex-inner} \rangle$

lemma

fixes $A :: \langle 'a :: \text{chilbert-space} \Rightarrow_{o_{CL}} 'b :: \text{chilbert-space} \rangle$

— [1], 3.9 Polar Decomposition

shows *polar-decomposition-correct*: $\langle \text{polar-decomposition } A \text{ } o_{CL} \text{ } \text{abs-op } A = A \rangle$
and *polar-decomposition-final-space*: $\langle \text{polar-decomposition } A \text{ } *_S \text{ } \text{top} = A \text{ } *_S \text{ } \text{top} \rangle$
and *polar-decomposition-initial-space*[*simp*]: $\langle \text{kernel } (\text{polar-decomposition } A) = \text{kernel } A \rangle$
and *polar-decomposition-partial-isometry*[*simp*]: $\langle \text{partial-isometry } (\text{polar-decomposition } A) \rangle$

proof –

have *abs-A-norm*: $\langle \text{norm } (\text{abs-op } A \text{ } h) = \text{norm } (A \text{ } h) \rangle$ **for** h

proof –

have $\langle \text{complex-of-real } ((\text{norm } (A \text{ } h))^2) = A \text{ } h \cdot_C A \text{ } h \rangle$
by (*simp add: cdot-square-norm*)

also have $\langle \dots = (A \text{ } * \text{ } o_{CL} \text{ } A) \text{ } h \cdot_C h \rangle$
by (*simp add: cinner-adj-left*)

also have $\langle \dots = ((\text{abs-op } A) \text{ } * \text{ } o_{CL} \text{ } \text{abs-op } A) \text{ } h \cdot_C h \rangle$
by (*simp add: abs-op-def positive-cblinfun-squareI positive-selfadjointI[unfolded selfadjoint-def]*)

also have $\langle \dots = \text{abs-op } A \text{ } h \cdot_C \text{abs-op } A \text{ } h \rangle$
by (*simp add: cinner-adj-left*)

also have $\langle \dots = \text{complex-of-real } ((\text{norm } (\text{abs-op } A \text{ } h))^2) \rangle$
using *cnorm-eq-square* **by** *blast*

finally show *?thesis*
by (*simp add: cdot-square-norm cnorm-eq*)

qed

define $W \text{ } W' \text{ } P$

where $\langle W = (\lambda \psi. A \text{ } *_V \text{ } \text{inv } (\text{abs-op } A) \text{ } \psi) \rangle$
and $\langle W' = \text{cblinfun-extension } (\text{range } (\text{abs-op } A)) \text{ } W \rangle$
and $\langle P = \text{Proj } (\text{abs-op } A \text{ } *_S \text{ } \text{top}) \rangle$

have *pdA*: $\langle \text{polar-decomposition } A = W' \text{ } o_{CL} \text{ } P \rangle$
by (*auto simp: polar-decomposition-def W'-def W-def P-def*)

have *AA-norm*: $\langle \text{norm } (W \text{ } \psi) = \text{norm } \psi \rangle$ **if** $\langle \psi \in \text{range } (\text{abs-op } A) \rangle$ **for** ψ

proof –

define h **where** $\langle h = \text{inv } (\text{abs-op } A) \text{ } \psi \rangle$

from that have *absA-h*: $\langle \text{abs-op } A \text{ } h = \psi \rangle$
by (*simp add: f-inv-into-f h-def*)

have $\langle \text{complex-of-real } ((\text{norm } (W \text{ } \psi))^2) = \text{complex-of-real } ((\text{norm } (A \text{ } h))^2) \rangle$
using *W-def h-def* **by** *blast*

also have $\langle \dots = A \text{ } h \cdot_C A \text{ } h \rangle$
by (*simp add: cdot-square-norm*)

also have $\langle \dots = (A \text{ } * \text{ } o_{CL} \text{ } A) \text{ } h \cdot_C h \rangle$
by (*simp add: cinner-adj-left*)

also have $\langle \dots = ((\text{abs-op } A) \text{ } * \text{ } o_{CL} \text{ } \text{abs-op } A) \text{ } h \cdot_C h \rangle$
by (*simp add: abs-op-def positive-cblinfun-squareI positive-selfadjointI[unfolded selfadjoint-def]*)

also have $\langle \dots = \text{abs-op } A \text{ } h \cdot_C \text{abs-op } A \text{ } h \rangle$
by (*simp add: cinner-adj-left*)

also have $\langle \dots = \text{complex-of-real } ((\text{norm } (\text{abs-op } A \text{ } h))^2) \rangle$
using *cnorm-eq-square* **by** *blast*

also have $\langle \dots = \text{complex-of-real } ((\text{norm } \psi)^2) \rangle$
using *absA-h* **by** *fastforce*
finally show $\langle \text{norm } (W \psi) = \text{norm } \psi \rangle$
by (*simp add: cdot-square-norm cnorm-eq*)
qed
then have *AA-norm'*: $\langle \text{norm } (W \psi) \leq 1 * \text{norm } \psi \rangle$ **if** $\langle \psi \in \text{range } (\text{abs-op } A) \rangle$ **for** ψ
using *that* **by** *simp*

have *W-absA*: $\langle W (\text{abs-op } A h) = A h \rangle$ **for** h
proof –
have $\langle A h = A h' \rangle$ **if** $\langle \text{abs-op } A h = \text{abs-op } A h' \rangle$ **for** $h h'$
proof –
from *that* **have** $\langle \text{norm } (\text{abs-op } A (h - h')) = 0 \rangle$
by (*simp add: cblinfun.diff-right*)
with *AA-norm* **have** $\langle \text{norm } (A (h - h')) = 0 \rangle$
by (*simp add: abs-A-norm*)
then show $\langle A h = A h' \rangle$
by (*simp add: cblinfun.diff-right*)
qed
then show *?thesis*
by (*metis W-def f-inv-into-f rangeI*)
qed

have *range-subspace*: $\langle \text{csubspace } (\text{range } (\text{abs-op } A)) \rangle$
by (*auto intro!: range-is-csubspace*)

have *exP*: $\langle \exists P. \text{is-Proj } P \wedge \text{range } ((*_V) P) = \text{closure } (\text{range } ((*_V) (\text{abs-op } A))) \rangle$
apply (*rule exI[of - $\langle \text{Proj } (\text{abs-op } A *_S \top) \rangle]$)*

by (*metis (no-types, opaque-lifting) Proj-is-Proj Proj-range Proj-range-closed cblinfun-image.rep-eq closure-closed space-as-set-top*)

have *add*: $\langle W (x + y) = W x + W y \rangle$ **if** *x-in*: $\langle x \in \text{range } (\text{abs-op } A) \rangle$ **and** *y-in*: $\langle y \in \text{range } (\text{abs-op } A) \rangle$ **for** $x y$
proof –
obtain $x' y'$ **where** $\langle x = \text{abs-op } A x' \rangle$ **and** $\langle y = \text{abs-op } A y' \rangle$
using *x-in y-in* **by** *blast*
then show *?thesis*
by (*simp flip: cblinfun.add-right add: W-absA*)
qed

have *scale*: $\langle W (c *_C x) = c *_C W x \rangle$ **if** *x-in*: $\langle x \in \text{range } (\text{abs-op } A) \rangle$ **for** $c x$
proof –
obtain x' **where** $\langle x = \text{abs-op } A x' \rangle$
using *x-in* **by** *blast*
then show *?thesis*
by (*simp flip: cblinfun.scaleC-right add: W-absA*)
qed

have $\langle \text{cblinfun-extension-exists } (\text{range } (\text{abs-op } A)) W \rangle$

```

using range-subspace exP add scale AA-norm'
by (rule cblinfun-extension-exists-proj)

then have W'-apply:  $\langle W' *_{\mathcal{V}} \psi = W \psi \rangle$  if  $\langle \psi \in \text{range} (\text{abs-op } A) \rangle$  for  $\psi$ 
by (simp add: W'-def cblinfun-extension-apply that)

have  $\langle \text{norm} (W' \psi) - \text{norm } \psi = 0 \rangle$  if  $\langle \psi \in \text{range} (\text{abs-op } A) \rangle$  for  $\psi$ 
by (simp add: W'-apply AA-norm that)
then have  $\langle \text{norm} (W' \psi) - \text{norm } \psi = 0 \rangle$  if  $\langle \psi \in \text{closure} (\text{range} (\text{abs-op } A)) \rangle$  for  $\psi$ 
apply (rule-tac continuous-constant-on-closure[where  $S = \langle \text{range} (\text{abs-op } A) \rangle$ ])
using that by (auto intro!: continuous-at-imp-continuous-on)
then have norm-W':  $\langle \text{norm} (W' \psi) = \text{norm } \psi \rangle$  if  $\langle \psi \in \text{space-as-set} (\text{abs-op } A *_{\mathcal{S}} \text{top}) \rangle$  for
 $\psi$ 
using cblinfun-image.rep-eq that by force

show correct:  $\langle \text{polar-decomposition } A \circ_{CL} \text{abs-op } A = A \rangle$ 
proof (rule cblinfun-eqI)
  fix  $\psi :: 'a$ 
  have  $\langle (\text{polar-decomposition } A \circ_{CL} \text{abs-op } A) *_{\mathcal{V}} \psi = W (P (\text{abs-op } A \psi)) \rangle$ 
    by (simp add: W'-apply P-def pdA Proj-fixes-image)
  also have  $\langle \dots = W (\text{abs-op } A \psi) \rangle$ 
    by (auto simp: P-def Proj-fixes-image)
  also have  $\langle \dots = A \psi \rangle$ 
    by (simp add: W-absA)

  finally show  $\langle (\text{polar-decomposition } A \circ_{CL} \text{abs-op } A) *_{\mathcal{V}} \psi = A *_{\mathcal{V}} \psi \rangle$ 
    by -
qed

show  $\langle \text{polar-decomposition } A *_{\mathcal{S}} \text{top} = A *_{\mathcal{S}} \text{top} \rangle$ 
proof (rule antisym)
  have *:  $\langle A *_{\mathcal{S}} \text{top} = \text{polar-decomposition } A *_{\mathcal{S}} \text{abs-op } A *_{\mathcal{S}} \text{top} \rangle$ 
    by (simp add: cblinfun-assoc-left(2) correct)
  also have  $\langle \dots \leq \text{polar-decomposition } A *_{\mathcal{S}} \text{top} \rangle$ 
    by (simp add: cblinfun-image-mono)
  finally show  $\langle A *_{\mathcal{S}} \text{top} \leq \text{polar-decomposition } A *_{\mathcal{S}} \text{top} \rangle$ 
    by -

  have  $\langle W' \psi \in \text{range } A \rangle$  if  $\langle \psi \in \text{range} (\text{abs-op } A) \rangle$  for  $\psi$ 
    using W'-apply W-def that by blast
  then have  $\langle W' \psi \in \text{closure} (\text{range } A) \rangle$  if  $\langle \psi \in \text{closure} (\text{range} (\text{abs-op } A)) \rangle$  for  $\psi$ 
    using *
  by (metis (mono-tags, lifting) P-def Proj-range Proj-fixes-image cblinfun-apply-cblinfun-compose
cblinfun-apply-in-image cblinfun-compose-image cblinfun-image.rep-eq pdA that top-ccsubspace.rep-eq)
  then have  $\langle W' \psi \in \text{space-as-set} (A *_{\mathcal{S}} \text{top}) \rangle$  if  $\langle \psi \in \text{space-as-set} (\text{abs-op } A *_{\mathcal{S}} \text{top}) \rangle$  for  $\psi$ 
    by (metis cblinfun-image.rep-eq that top-ccsubspace.rep-eq)
  then have  $\langle \text{polar-decomposition } A \psi \in \text{space-as-set} (A *_{\mathcal{S}} \text{top}) \rangle$  for  $\psi$ 
    by (metis P-def Proj-range cblinfun-apply-cblinfun-compose cblinfun-apply-in-image pdA)
  then show  $\langle \text{polar-decomposition } A *_{\mathcal{S}} \text{top} \leq A *_{\mathcal{S}} \text{top} \rangle$ 

```

using *
by (*metis* (*no-types*, *lifting*) *Proj-idempotent Proj-range cblinfun-compose-image dual-order.eq-iff polar-decomposition-def*)
qed

show $\langle \text{partial-isometry (polar-decomposition } A) \rangle$
apply (*rule partial-isometryI'*[**where** $V = \langle \text{abs-op } A *_S \text{ top} \rangle$])
by (*auto simp add: P-def Proj-fixes-image norm-W' pdA kernel-memberD*)

have $\langle \text{kernel (polar-decomposition } A) = - (\text{abs-op } A *_S \text{ top}) \rangle$
apply (*rule partial-isometry-initial'*[**where** $V = \langle \text{abs-op } A *_S \text{ top} \rangle$])
by (*auto simp add: P-def Proj-fixes-image norm-W' pdA kernel-memberD*)
also have $\langle \dots = \text{kernel (abs-op } A) \rangle$
by (*metis abs-op-pos kernel-compl-adj-range positive-selfadjointI selfadjoint-def*)
also have $\langle \dots = \text{kernel } A \rangle$
by (*simp add: kernel-abs-op*)
finally show $\langle \text{kernel (polar-decomposition } A) = \text{kernel } A \rangle$
by –
qed

lemma *polar-decomposition-correct'*: $\langle (\text{polar-decomposition } A) * o_{CL} A = \text{abs-op } A \rangle$
for $A :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$
proof –
have $\langle \text{polar-decomposition } A * o_{CL} A = (\text{polar-decomposition } A * o_{CL} \text{polar-decomposition } A) o_{CL} \text{abs-op } A \rangle$
by (*simp add: cblinfun-compose-assoc polar-decomposition-correct*)
also have $\langle \dots = \text{Proj} (- \text{kernel (polar-decomposition } A)) o_{CL} \text{abs-op } A \rangle$
by (*simp add: partial-isometry-adj-a-o-a polar-decomposition-partial-isometry*)
also have $\langle \dots = \text{Proj} (- \text{kernel } A) o_{CL} \text{abs-op } A \rangle$
by (*simp add: polar-decomposition-initial-space*)
also have $\langle \dots = \text{Proj} (- \text{kernel (abs-op } A)) o_{CL} \text{abs-op } A \rangle$
by *simp*
also have $\langle \dots = \text{Proj} (\text{abs-op } A *_S \text{ top}) o_{CL} \text{abs-op } A \rangle$
by (*metis abs-op-pos kernel-compl-adj-range ortho-involution positive-selfadjointI selfadjoint-def*)
also have $\langle \dots = \text{abs-op } A \rangle$
by (*simp add: Proj-fixes-image cblinfun-eqI*)
finally show *?thesis*
by –
qed

lemma *abs-op-adj*: $\langle \text{abs-op} (a*) = \text{sandwich (polar-decomposition } a) (\text{abs-op } a) \rangle$
proof –
have *pos*: $\langle \text{sandwich (polar-decomposition } a) (\text{abs-op } a) \geq 0 \rangle$
by (*simp add: sandwich-pos*)
have $\langle (\text{sandwich (polar-decomposition } a) (\text{abs-op } a)) * o_{CL} (\text{sandwich (polar-decomposition } a) (\text{abs-op } a))$
 $= \text{polar-decomposition } a o_{CL} (\text{abs-op } a) * o_{CL} \text{abs-op } a o_{CL} (\text{polar-decomposition } a) * \rangle$
apply (*simp add: sandwich-apply*)

by (*metis* (*no-types*, *lifting*) *cblinfun-assoc-left(1)* *polar-decomposition-correct* *polar-decomposition-correct'*)
also have $\langle \dots = a \circ_{CL} a^* \rangle$
by (*metis* *abs-op-pos* *adj-cblinfun-compose* *cblinfun-assoc-left(1)* *polar-decomposition-correct*
positive-selfadjointI *selfadjoint-def*)
finally have $\langle \text{sandwich } (polar-decomposition\ a) (abs-op\ a) = \text{sqr-op } (a \circ_{CL} a^*) \rangle$
using *pos* **by** (*simp* *add*: *sqr-op-unique*)
also have $\langle \dots = abs-op (a^*) \rangle$
by (*simp* *add*: *abs-op-def*)
finally show *?thesis*
by *simp*
qed

lemma *abs-opI*:
assumes $\langle a^* \circ_{CL} a = b^* \circ_{CL} b \rangle$
assumes $\langle a \geq 0 \rangle$
shows $\langle a = abs-op\ b \rangle$
by (*simp* *add*: *abs-op-def* *assms(1)* *assms(2)* *sqr-op-unique*)

lemma *abs-op-id-on-pos*: $\langle a \geq 0 \implies abs-op\ a = a \rangle$
using *abs-opI* **by** *force*

lemma *norm-abs-op[simp]*: $\langle norm (abs-op\ a) = norm\ a \rangle$
for $a :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$
proof –
have $\langle (norm (abs-op\ a))^2 = norm (abs-op\ a^* \circ_{CL} abs-op\ a) \rangle$
by *simp*
also have $\langle \dots = norm (a^* \circ_{CL} a) \rangle$
by (*simp* *add*: *abs-op-def* *positive-cblinfun-squareI* *positive-selfadjointI* [*unfolded selfadjoint-def*])
also have $\langle \dots = (norm\ a)^2 \rangle$
by *simp*
finally show *?thesis*
by *simp*
qed

lemma *partial-isometry-iff-square-proj*:
– [2], Exercise VIII.3.15
fixes $A :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$
shows $\langle \text{partial-isometry } A \longleftrightarrow \text{is-Proj } (A^* \circ_{CL} A) \rangle$
proof (*rule iffI*)
show $\langle \text{is-Proj } (A^* \circ_{CL} A) \rangle$ **if** $\langle \text{partial-isometry } A \rangle$
by (*simp* *add*: *partial-isometry-square-proj* *that*)
next
show $\langle \text{partial-isometry } A \rangle$ **if** $\langle \text{is-Proj } (A^* \circ_{CL} A) \rangle$
proof (*rule partial-isometryI*)
fix h
from *that* **have** $\langle norm (A^* \circ_{CL} A) \leq 1 \rangle$
using *norm-is-Proj* **by** *blast*

then have $\text{norm}A$: $\langle \text{norm } A \leq 1 \rangle$ **and** $\text{norm}A\text{adj}$: $\langle \text{norm } (A^*) \leq 1 \rangle$
by (*simp-all add: norm-AadjA abs-square-le-1*)
assume $\langle h \in \text{space-as-set } (- \text{kernel } A) \rangle$
also have $\langle \dots = \text{space-as-set } (- \text{kernel } (A^* \circ_{CL} A)) \rangle$
by (*metis (no-types, lifting) abs-opI is-Proj-algebraic kernel-abs-op positive-cblinfun-squareI*
that)
also have $\langle \dots = \text{space-as-set } ((A^* \circ_{CL} A) *_S \top) \rangle$
by (*simp add: kernel-compl-adj-range*)
finally have $\langle A^* *_V A *_V h = h \rangle$
by (*metis Proj-fixes-image Proj-on-own-range that cblinfun-apply-cblinfun-compose*)
then have $\langle \text{norm } h = \text{norm } (A^* *_V A *_V h) \rangle$
by *simp*
also have $\langle \dots \leq \text{norm } (A *_V h) \rangle$
by (*smt (verit) normAadj mult-left-le-one-le norm-cblinfun norm-ge-zero*)
also have $\langle \dots \leq \text{norm } h \rangle$
by (*smt (verit) normA mult-left-le-one-le norm-cblinfun norm-ge-zero*)
ultimately show $\langle \text{norm } (A *_V h) = \text{norm } h \rangle$
by *simp*
qed
qed

lemma *abs-op-square*: $\langle (\text{abs-op } A)^* \circ_{CL} \text{abs-op } A = A^* \circ_{CL} A \rangle$
by (*simp add: abs-op-def positive-cblinfun-squareI positive-selfadjointI [unfolded selfadjoint-def]*)

lemma *polar-decomposition-0*[*simp*]: $\langle \text{polar-decomposition } 0 = (0 :: 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space}) \rangle$
proof –

have $\langle \text{polar-decomposition } (0 :: 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space}) *_S \top = 0 *_S \top \rangle$
by (*simp add: polar-decomposition-final-space*)
then show *?thesis*
by *simp*
qed

lemma *polar-decomposition-unique*:

fixes $A :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
assumes *ker*: $\langle \text{kernel } X = \text{kernel } A \rangle$
assumes *comp*: $\langle X \circ_{CL} \text{abs-op } A = A \rangle$
shows $\langle X = \text{polar-decomposition } A \rangle$

proof –

have $\langle X \psi = \text{polar-decomposition } A \psi \rangle$ **if** $\langle \psi \in \text{space-as-set } (\text{kernel } A) \rangle$ **for** ψ

proof –

have $\langle \psi \in \text{space-as-set } (\text{kernel } X) \rangle$
by (*simp add: ker that*)
then have $\langle X \psi = 0 \rangle$
by (*simp add: kernel.rep-eq*)

moreover

have $\langle \psi \in \text{space-as-set } (\text{kernel } (\text{polar-decomposition } A)) \rangle$
by (*simp add: polar-decomposition-initial-space that*)
then have $\langle \text{polar-decomposition } A \psi = 0 \rangle$
by (*simp add: kernel.rep-eq del: polar-decomposition-initial-space*)

ultimately show *?thesis*
by *simp*
qed
then have 1: $\langle X \text{ } o_{CL} \text{ Proj } (\text{kernel } A) = \text{polar-decomposition } A \text{ } o_{CL} \text{ Proj } (\text{kernel } A) \rangle$
by (*metis* *assms*(1) *cblinfun-compose-Proj-kernel polar-decomposition-initial-space*)
have *: $\langle \text{abs-op } A *_S \top = - \text{kernel } A \rangle$
by (*metis* (*mono-tags*, *opaque-lifting*) *abs-op-pos kernel-abs-op kernel-compl-adj-range ortho-involution positive-selfadjointI selfadjoint-def*)

have $\langle X \text{ } o_{CL} \text{ abs-op } A = \text{polar-decomposition } A \text{ } o_{CL} \text{ abs-op } A \rangle$
by (*simp* *add: comp polar-decomposition-correct*)
then have $\langle X \psi = \text{polar-decomposition } A \psi \rangle$ **if** $\langle \psi \in \text{space-as-set } (\text{abs-op } A *_S \top) \rangle$ **for** ψ
by (*simp* *add: cblinfun-same-on-image that*)
then have 2: $\langle X \text{ } o_{CL} \text{ Proj } (- \text{kernel } A) = \text{polar-decomposition } A \text{ } o_{CL} \text{ Proj } (- \text{kernel } A) \rangle$
using *
by (*metis* (*no-types*, *opaque-lifting*) *Proj-idempotent cblinfun-eqI lift-cblinfun-comp(4) norm-Proj-apply*)
from 1 2 **have** $\langle X \text{ } o_{CL} \text{ Proj } (- \text{kernel } A) + X \text{ } o_{CL} \text{ Proj } (\text{kernel } A)$
 $= \text{polar-decomposition } A \text{ } o_{CL} \text{ Proj } (- \text{kernel } A) + \text{polar-decomposition } A \text{ } o_{CL} \text{ Proj } (\text{kernel } A) \rangle$
by *simp*
then show *?thesis*
by (*simp* *add: Proj-ortho-compl flip: cblinfun-compose-add-right*)
qed

lemma *norm-cblinfun-mono*:

— Would logically belong in *Complex-Bounded-Operators.Complex-Bounded-Linear-Function* but uses *sqrt-op* from this theory in the proof.

fixes $A \ B :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle A \geq 0 \rangle$
assumes $\langle A \leq B \rangle$
shows $\langle \text{norm } A \leq \text{norm } B \rangle$
proof —
have $\langle B \geq 0 \rangle$
using *assms* **by** *force*
have *sqrtA*: $\langle (\text{sqrt-op } A) * o_{CL} \text{ sqrt-op } A = A \rangle$
by (*simp* *add: \langle A \geq 0 \rangle positive-selfadjointI[unfolded selfadjoint-def]*)
have *sqrtB*: $\langle (\text{sqrt-op } B) * o_{CL} \text{ sqrt-op } B = B \rangle$
by (*simp* *add: \langle B \geq 0 \rangle positive-selfadjointI[unfolded selfadjoint-def]*)
have $\langle \text{norm } (\text{sqrt-op } A \psi) \leq \text{norm } (\text{sqrt-op } B \psi) \rangle$ **for** ψ
apply (*auto* *intro!: cnorm-le[THEN iffD2]*)
simp: sqrtA sqrtB
simp flip: cinner-adj-right cblinfun-apply-cblinfun-compose)
using *assms* *less-eq-cblinfun-def* **by** *auto*
then have $\langle \text{norm } (\text{sqrt-op } A) \leq \text{norm } (\text{sqrt-op } B) \rangle$
by (*meson* *dual-order.trans norm-cblinfun norm-cblinfun-bound norm-ge-zero*)
moreover have $\langle \text{norm } A = (\text{norm } (\text{sqrt-op } A))^2 \rangle$
by (*metis* *norm-AadjA sqrtA*)
moreover have $\langle \text{norm } B = (\text{norm } (\text{sqrt-op } B))^2 \rangle$
by (*metis* *norm-AadjA sqrtB*)

ultimately show $\langle \text{norm } A \leq \text{norm } B \rangle$
by force
qed

lemma sandwich-mono: $\langle \text{sandwich } A \ B \leq \text{sandwich } A \ C \rangle$ **if** $\langle B \leq C \rangle$
by (*metis cblinfun.real.diff-right diff-ge-0-iff-ge sandwich-pos that*)

lemma sums-pos-cblinfun:
fixes $f :: \text{nat} \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\langle f \text{ sums } a \rangle$
assumes $\langle \bigwedge n. f \ n \geq 0 \rangle$
shows $a \geq 0$
apply (*rule sums-mono-cblinfun[where f= λ -. 0] and g=f*)
using *assms by auto*

lemma has-sum-mono-cblinfun:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes ($f \text{ has-sum } x$) A **and** ($g \text{ has-sum } y$) A
assumes $\langle \bigwedge x. x \in A \implies f \ x \leq g \ x \rangle$
shows $x \leq y$
using *assms has-sum-mono-neutral-cblinfun by force*

lemma infsum-mono-cblinfun:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $f \text{ summable-on } A$ **and** $g \text{ summable-on } A$
assumes $\langle \bigwedge x. x \in A \implies f \ x \leq g \ x \rangle$
shows $\text{infsum } f \ A \leq \text{infsum } g \ A$
by (*meson assms has-sum-infsum has-sum-mono-cblinfun*)

lemma suminf-mono-cblinfun:
fixes $f :: \text{nat} \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\text{summable } f$ **and** $\text{summable } g$
assumes $\langle \bigwedge x. f \ x \leq g \ x \rangle$
shows $\text{suminf } f \leq \text{suminf } g$
using *assms sums-mono-cblinfun by blast*

lemma suminf-pos-cblinfun:
fixes $f :: \text{nat} \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\langle \text{summable } f \rangle$
assumes $\langle \bigwedge x. f \ x \geq 0 \rangle$
shows $\text{suminf } f \geq 0$
using *assms sums-mono-cblinfun by blast*

lemma infsum-mono-neutral-cblinfun:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $f \text{ summable-on } A$ **and** $g \text{ summable-on } B$
assumes $\langle \bigwedge x. x \in A \cap B \implies f \ x \leq g \ x \rangle$
assumes $\langle \bigwedge x. x \in A - B \implies f \ x \leq 0 \rangle$

assumes $\langle \bigwedge x. x \in B - A \implies g x \geq 0 \rangle$
shows $\text{infsum } f A \leq \text{infsum } g B$
by (*smt* (*verit*, *del-insts*) *assms*(1) *assms*(2) *assms*(3) *assms*(4) *assms*(5) *has-sum-infsum* *has-sum-mono-neutral-cblinfun*)

lemma *abs-op-geq*: $\langle \text{abs-op } a \geq a \rangle$ **if** $\langle \text{selfadjoint } a \rangle$

proof –

define $A P$ **where** $\langle A = \text{abs-op } a \rangle$ **and** $\langle P = \text{Proj } (\text{kernel } (A + a)) \rangle$
from *that* **have** [*simp*]: $\langle a^* = a \rangle$
by (*simp* *add*: *selfadjoint-def*)
have [*simp*]: $\langle A \geq 0 \rangle$
by (*simp* *add*: *A-def*)
then **have** [*simp*]: $\langle A^* = A \rangle$
using *positive-selfadjointI selfadjoint-def* **by** *fastforce*
have *aa-AA*: $\langle a \circ_{CL} a = A \circ_{CL} A \rangle$
by (*metis* *A-def* $\langle A^* = A \rangle$ *abs-op-square that selfadjoint-def*)
have [*simp*]: $\langle P^* = P \rangle$
by (*simp* *add*: *P-def adj-Proj*)
have *Aa-aA*: $\langle A \circ_{CL} a = a \circ_{CL} A \rangle$
by (*metis* (*full-types*) *A-def lift-cblinfun-comp*(2) *abs-op-def positive-cblinfun-squareI sqrt-op-commute that selfadjoint-def*)

have $\langle (A - a) \psi \cdot_C (A + a) \varphi = 0 \rangle$ **for** $\varphi \psi$
by (*simp* *add*: *adj-minus that* $\langle A^* = A \rangle$ *aa-AA Aa-aA cblinfun-compose-add-right cblinfun-compose-minus-left* *flip: cinner-adj-right cblinfun-apply-cblinfun-compose*)
then **have** $\langle (A - a) \psi \in \text{space-as-set } (\text{kernel } (A + a)) \rangle$ **for** ψ
by (*metis* $\langle A^* = A \rangle$ *adj-plus call-zero-iff cinner-adj-left kernel-memberI that selfadjoint-def*)
then **have** *P-fix*: $\langle P \circ_{CL} (A - a) = (A - a) \rangle$
by (*simp* *add*: *P-def Proj-fixes-image cblinfun-eqI*)
then **have** $\langle P \circ_{CL} (A - a) \circ_{CL} P = (A - a) \circ_{CL} P \rangle$
by *simp*
also **have** $\langle \dots = (P \circ_{CL} (A - a))^* \rangle$
by (*simp* *add*: *adj-minus* $\langle A^* = A \rangle$ *that* $\langle P^* = P \rangle$)
also **have** $\langle \dots = (A - a)^* \rangle$
by (*simp* *add*: *P-fix*)
also **have** $\langle \dots = A - a \rangle$
by (*simp* *add*: $\langle A^* = A \rangle$ *that adj-minus*)
finally **have** 1: $\langle P \circ_{CL} (A - a) \circ_{CL} P = A - a \rangle$
by –
have 2: $\langle P \circ_{CL} (A + a) \circ_{CL} P = 0 \rangle$
by (*simp* *add*: *P-def cblinfun-compose-assoc*)
have $\langle A - a = P \circ_{CL} (A - a) \circ_{CL} P + P \circ_{CL} (A + a) \circ_{CL} P \rangle$
by (*simp* *add*: 1 2)
also **have** $\langle \dots = \text{sandwich } P (2 *_{CL} A) \rangle$
by (*simp* *add*: *sandwich-apply cblinfun-compose-minus-left cblinfun-compose-minus-right cblinfun-compose-add-left cblinfun-compose-add-right scaleC-2* $\langle P^* = P \rangle$)
also **have** $\langle \dots \geq 0 \rangle$
by (*auto intro!*: *sandwich-pos scaleC-nonneg-nonneg simp: less-eq-complex-def*)

finally show $\langle A \geq a \rangle$
 by *auto*
 qed

lemma *abs-op-geq-neg*: $\langle \text{abs-op } a \geq - a \rangle$ if $\langle \text{selfadjoint } a \rangle$
 by (*metis abs-op-geq abs-op-uminus adj-uminus that selfadjoint-def*)

lemma *infsum-nonneg-cblinfun*:
 fixes $f :: 'a \Rightarrow 'b::\text{chilbert-space} \Rightarrow_{CL} 'b$
 assumes $\bigwedge x. x \in M \implies 0 \leq f x$
 shows $\text{infsum } f M \geq 0$
 apply (*cases* $\langle f \text{ summable-on } M \rangle$)
 apply (*subst infsum-0-simp[symmetric]*)
 apply (*rule infsum-mono-cblinfun*)
 using *assms* by (*auto simp: infsum-not-exists*)

lemma *adj-abs-op[simp]*: $\langle (\text{abs-op } a)^* = \text{abs-op } a \rangle$
 by (*simp add: positive-selfadjointI[unfolded selfadjoint-def]*)

lemma *cblinfun-image-less-eqI*:
 fixes $A :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector} \rangle$
 assumes $\langle \bigwedge h. h \in \text{space-as-set } S \implies A h \in \text{space-as-set } T \rangle$
 shows $\langle A *_S S \leq T \rangle$

proof –
 from *assms* have $\langle A \text{ ' space-as-set } S \subseteq \text{space-as-set } T \rangle$
 by *blast*
 then have $\langle \text{closure } (A \text{ ' space-as-set } S) \subseteq \text{closure } (\text{space-as-set } T) \rangle$
 by (*rule closure-mono*)
 also have $\langle \dots = \text{space-as-set } T \rangle$
 by *force*
 finally show *?thesis*
 apply (*transfer fixing: A*)
 by (*simp add: cblinfun-image.rep-eq ccsubspace-leI*)

qed

lemma *abs-op-plus-orthogonal*:
 assumes $\langle a^* o_{CL} b = 0 \rangle$ and $\langle a o_{CL} b^* = 0 \rangle$
 shows $\langle \text{abs-op } (a + b) = \text{abs-op } a + \text{abs-op } b \rangle$
 proof (*rule abs-opI[symmetric]*)
 have *ba*: $\langle b^* o_{CL} a = 0 \rangle$
 apply (*rule cblinfun-eqI, rule cinner-extensionality*)
 apply (*simp add: cinner-adj-right flip: cinner-adj-left*)
 by (*simp add: assms simp-a-oCL-b'*)
 have *abs-ab*: $\langle \text{abs-op } a o_{CL} \text{abs-op } b = 0 \rangle$
 proof –
 have $\langle \text{abs-op } b *_S \top = - \text{kernel } (\text{abs-op } b) \rangle$
 by (*simp add: kernel-compl-adj-range positive-selfadjointI*)

also have $\langle \dots = - \text{kernel } b \rangle$
by *simp*
also have $\langle \dots = (b^*) *_S \top \rangle$
by (*simp add: kernel-compl-adj-range*)
also have $\langle \dots \leq \text{kernel } a \rangle$
apply (*auto intro!: cblinfun-image-less-eqI kernel-memberI simp:*)
by (*simp add: assms flip: cblinfun-apply-cblinfun-compose*)
also have $\langle \dots = \text{kernel } (\text{abs-op } a) \rangle$
by *simp*
finally show $\langle \text{abs-op } a \text{ } o_{CL} \text{ abs-op } b = 0 \rangle$
by (*metis Proj-compose-cancelI cblinfun-compose-Proj-kernel cblinfun-compose-assoc cblinfun-compose-zero-left*)
qed
then have *abs-ba*: $\langle \text{abs-op } b \text{ } o_{CL} \text{ abs-op } a = 0 \rangle$
by (*metis abs-op-pos adj-0 adj-cblinfun-compose positive-selfadjointI selfadjoint-def*)
have $\langle (a + b)^* \text{ } o_{CL} (a + b) = (a^*) \text{ } o_{CL} a + (b^*) \text{ } o_{CL} b \rangle$
by (*simp add: cblinfun-compose-add-left cblinfun-compose-add-right adj-plus assms ba*)
also have $\langle \dots = (\text{abs-op } a + \text{abs-op } b)^* \text{ } o_{CL} (\text{abs-op } a + \text{abs-op } b) \rangle$
by (*simp add: cblinfun-compose-add-left cblinfun-compose-add-right adj-plus abs-ab abs-ba flip: abs-op-square*)
finally show $\langle (\text{abs-op } a + \text{abs-op } b)^* \text{ } o_{CL} (\text{abs-op } a + \text{abs-op } b) = (a + b)^* \text{ } o_{CL} (a + b) \rangle$
by *simp*
show $\langle 0 \leq \text{abs-op } a + \text{abs-op } b \rangle$
by *simp*
qed

definition *pos-op* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \Rightarrow 'a \Rightarrow_{CL} 'a \rangle$ **where**
 $\langle \text{pos-op } a = (\text{abs-op } a + a) /_R 2 \rangle$

definition *neg-op* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \Rightarrow 'a \Rightarrow_{CL} 'a \rangle$ **where**
 $\langle \text{neg-op } a = (\text{abs-op } a - a) /_R 2 \rangle$

lemma *pos-op-pos*:
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{pos-op } a \geq 0 \rangle$
using *abs-op-geq-neq[OF assms]*
apply (*simp add: pos-op-def*)
by (*smt (verit, best) add-le-cancel-right more-arith-simps(3) scaleR-nonneg-nonneg zero-le-divide-iff*)

lemma *neg-op-pos*:
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{neg-op } a \geq 0 \rangle$
using *abs-op-geq[OF assms]*
by (*simp add: neg-op-def scaleR-nonneg-nonneg*)

lemma *pos-op-neg-op-ortho*:
assumes $\langle \text{selfadjoint } a \rangle$

shows $\langle \text{pos-op } a \text{ } o_{CL} \text{ neg-op } a = 0 \rangle$
apply (*auto intro!*: *simp*: *pos-op-def neg-op-def cblinfun-compose-add-left cblinfun-compose-minus-right*)
by (*metis* (*no-types*, *opaque-lifting*) *Groups.add-ac(2)* *abs-op-def abs-op-pos abs-op-square*
assms cblinfun-assoc-left(1) *positive-cblinfun-squareI* *positive-selfadjointI* *selfadjoint-def sqrt-op-commute*)

lemma *pos-op-plus-neg-op*: $\langle \text{pos-op } a + \text{neg-op } a = \text{abs-op } a \rangle$
by (*simp add*: *pos-op-def neg-op-def scaleR-diff-right scaleR-add-right pth-8*)

lemma *pos-op-minus-neg-op*: $\langle \text{pos-op } a - \text{neg-op } a = a \rangle$
by (*simp add*: *pos-op-def neg-op-def scaleR-diff-right scaleR-add-right pth-8*)

lemma *pos-op-neg-op-unique*:
assumes *bca*: $\langle b - c = a \rangle$
assumes $\langle b \geq 0 \rangle$ **and** $\langle c \geq 0 \rangle$
assumes *bc*: $\langle b \text{ } o_{CL} \text{ } c = 0 \rangle$
shows $\langle b = \text{pos-op } a \rangle$ **and** $\langle c = \text{neg-op } a \rangle$
proof –
from *bc* **have** *cb*: $\langle c \text{ } o_{CL} \text{ } b = 0 \rangle$
by (*metis* *adj-0 adj-cblinfun-compose assms(2)* *assms(3)* *positive-selfadjointI* *selfadjoint-def*)

from $\langle b \geq 0 \rangle$ **have** [*simp*]: $\langle b^* = b \rangle$
by (*simp add*: *positive-selfadjointI[unfolded selfadjoint-def]*)
from $\langle c \geq 0 \rangle$ **have** [*simp*]: $\langle c^* = c \rangle$
by (*simp add*: *positive-selfadjointI[unfolded selfadjoint-def]*)
have *bc-abs*: $\langle b + c = \text{abs-op } a \rangle$
proof –
have $\langle (b + c)^* \text{ } o_{CL} \text{ } (b + c) = b \text{ } o_{CL} \text{ } b + c \text{ } o_{CL} \text{ } c \rangle$
by (*simp add*: *cblinfun-compose-add-left cblinfun-compose-add-right bc cb adj-plus*)
also have $\langle \dots = (b - c)^* \text{ } o_{CL} \text{ } (b - c) \rangle$
by (*simp add*: *cblinfun-compose-minus-left cblinfun-compose-minus-right bc cb adj-minus*)
also from *bca* **have** $\langle \dots = a^* \text{ } o_{CL} \text{ } a \rangle$
by *blast*
finally show *?thesis*
apply (*rule abs-opI*)
by (*simp add*: $\langle b \geq 0 \rangle \langle c \geq 0 \rangle$)

qed
from *arg-cong2[OF bca bc-abs, of plus]*
arg-cong2[OF pos-op-minus-neg-op[of a] pos-op-plus-neg-op[of a], of plus, symmetric]
show $\langle b = \text{pos-op } a \rangle$
by (*simp flip*: *scaleR-2*)
from *arg-cong2[OF bc-abs bca, of minus]*
arg-cong2[OF pos-op-plus-neg-op[of a] pos-op-minus-neg-op[of a], of minus, symmetric]
show $\langle c = \text{neg-op } a \rangle$
by (*simp flip*: *scaleR-2*)
qed

lemma *pos-imp-selfadjoint*: $\langle a \geq 0 \implies \text{selfadjoint } a \rangle$
using *positive-selfadjointI selfadjoint-def* **by** *blast*

lemma *abs-op-one-dim*: $\langle \text{abs-op } x = \text{one-dim-iso } (\text{abs } (\text{one-dim-iso } x :: \text{complex})) \rangle$
by (*metis (mono-tags, lifting) abs-opI abs-op-scaleC of-complex-def one-cblinfun-adj one-comp-one-cblinfun one-dim-iso-is-of-complex one-dim-iso-of-one one-dim-iso-of-zero one-dim-loewner-order one-dim-scaleC-1 zero-less-one-class.zero-le-one*)

lemma *pos-selfadjoint*: $\langle \text{selfadjoint } a \rangle$ **if** $\langle a \geq 0 \rangle$
using *adj-0 comparable-selfadjoint selfadjoint-def* **that** **by** *blast*

lemma *one-dim-loewner-order-strict*: $\langle A > B \iff \text{one-dim-iso } A > (\text{one-dim-iso } B :: \text{complex}) \rangle$
for $A B :: \langle 'a \Rightarrow_{CL} 'a :: \{\text{chilbert-space, one-dim}\} \rangle$
by (*auto simp: less-cblinfun-def one-dim-loewner-order*)

lemma *one-dim-cblinfun-zero-le-one*: $\langle 0 < (1 :: 'a :: \text{one-dim} \Rightarrow_{CL} 'a) \rangle$
by (*simp add: one-dim-loewner-order-strict*)

lemma *one-dim-cblinfun-one-pos*: $\langle 0 \leq (1 :: 'a :: \text{one-dim} \Rightarrow_{CL} 'a) \rangle$
by (*simp add: one-dim-loewner-order*)

lemma *Proj-pos[iff]*: $\langle \text{Proj } S \geq 0 \rangle$
apply (*rule positive-cblinfun-squareI[where B= $\langle \text{Proj } S \rangle$]*)
by (*simp add: adj-Proj*)

lemma *abs-op-Proj[simp]*: $\langle \text{abs-op } (\text{Proj } S) = \text{Proj } S \rangle$
by (*simp add: abs-op-id-on-pos*)

lemma *diagonal-operator-pos*:
assumes $\langle \bigwedge x. f x \geq 0 \rangle$
shows $\langle \text{diagonal-operator } f \geq 0 \rangle$
proof (*cases $\langle \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (f x))) \rangle$*)
case *True*
have [*simp*]: $\langle \text{csqrt } (f x) = \text{sqrt } (\text{cmod } (f x)) \rangle$ **for** x
by (*simp add: Extra-Ordered-Fields.complex-of-real-cmod assms abs-pos of-real-sqrt*)
have *bdd*: $\langle \text{bdd-above } (\text{range } (\lambda x. \text{sqrt } (\text{cmod } (f x)))) \rangle$
proof –
from *True* **obtain** B **where** $\langle \text{cmod } (f x) \leq B \rangle$ **for** x
by (*auto simp: bdd-above-def*)
then show *?thesis*
by (*auto intro!: bdd-aboveI[where M= $\langle \text{sqrt } B \rangle$] simp:*)
qed
show *?thesis*
apply (*rule positive-cblinfun-squareI[where B= $\langle \text{diagonal-operator } (\lambda x. \text{csqrt } (f x)) \rangle$]*)
by (*simp add: assms diagonal-operator-adj diagonal-operator-comp bdd complex-of-real-cmod abs-pos flip: of-real-mult*)

```

next
  case False
  then show ?thesis
  by (simp add: diagonal-operator-invalid)
qed

```

```

lemma abs-op-diagonal-operator:
  ⟨abs-op (diagonal-operator f) = diagonal-operator (λx. abs (f x))⟩
proof (cases ⟨bdd-above (range (λx. cmod (f x)))⟩)
  case True
  show ?thesis
  apply (rule abs-opI[symmetric])
  by (auto intro!: diagonal-operator-pos abs-nn simp: True diagonal-operator-adj diagonal-operator-comp
cnj-x-x)
next
  case False
  then show ?thesis
  by (simp add: diagonal-operator-invalid)
qed

```

```

unbundle no cblinfun-syntax

```

```

end

```

5 $HS2Ell2$ – Representing any Hilbert space as $\ell_2(X)$

```

theory HS2Ell2

```

```

  imports Complex-Bounded-Operators.Complex-L2 Misc-Tensor-Product-BO

```

```

begin

```

```

unbundle cblinfun-syntax

```

```

typedef (overloaded) 'a::⟨{hilbert-space, not-singleton}⟩ hilbert2ell2 = ⟨some-hilbert-basis
:: 'a set⟩
  using some-hilbert-basis-nonempty by auto

```

```

definition ell2-to-hilbert where ⟨ell2-to-hilbert = cblinfun-extension (range ket) (Rep-hilbert2ell2
o inv ket)⟩

```

```

lemma ell2-to-hilbert-ket: ⟨ell2-to-hilbert *V ket x = Rep-hilbert2ell2 x⟩

```

```

proof –

```

```

  have ⟨cblinfun-extension-exists (range ket) (Rep-hilbert2ell2 o inv ket)⟩

```

```

  proof (rule cblinfun-extension-exists-ortho[where B=1])

```

```

    fix x y :: 'b hilbert2ell2 ell2

```

```

    assume x ∈ range ket y ∈ range ket x ≠ y

```

```

    then obtain x' y' where x'-y': x = ket x' y = ket y' x' ≠ y'

```

```

    by auto

```

```

    have is-orthogonal (Rep-hilbert2ell2 x') (Rep-hilbert2ell2 y')

```

```

    by (meson Rep-chilbert2ell2 Rep-chilbert2ell2-inject ⟨x' ≠ y'⟩ is-ortho-set-def is-ortho-set-some-chilbert-basis)
    thus is-orthogonal ((Rep-chilbert2ell2 ∘ inv ket) x) ((Rep-chilbert2ell2 ∘ inv ket) y)
      using x'-y' by auto
  qed (auto simp: Rep-chilbert2ell2 is-normal-some-chilbert-basis)
  from cblinfun-extension-apply[OF this]
  have cblinfun-extension (range ket) (Rep-chilbert2ell2 ∘ inv ket) *V (ket x) =
    (Rep-chilbert2ell2 ∘ inv ket) (ket x)
    by blast
  thus ?thesis
    by (simp add: ell2-to-hilbert-def)
qed

```

lemma norm-ell2-to-hilbert: ⟨norm ell2-to-hilbert = 1⟩

```

proof (rule order.antisym)
  show ⟨norm ell2-to-hilbert ≤ 1⟩
    unfolding ell2-to-hilbert-def
  proof (rule cblinfun-extension-exists-ortho-norm[where B=1])
    fix x y :: 'b chilbert2ell2 ell2
    assume x ∈ range ket y ∈ range ket x ≠ y
    then obtain x' y' where x'-y': x = ket x' y = ket y' x' ≠ y'
      by auto
    have is-orthogonal (Rep-chilbert2ell2 x') (Rep-chilbert2ell2 y')
    by (meson Rep-chilbert2ell2 Rep-chilbert2ell2-inject ⟨x' ≠ y'⟩ is-ortho-set-def is-ortho-set-some-chilbert-basis)
    thus is-orthogonal ((Rep-chilbert2ell2 ∘ inv ket) x) ((Rep-chilbert2ell2 ∘ inv ket) y)
      using x'-y' by auto
  qed (auto simp: Rep-chilbert2ell2 is-normal-some-chilbert-basis)
  show ⟨norm ell2-to-hilbert ≥ 1⟩
    by (rule cblinfun-norm-geqI[where x=⟨ket undefined⟩])
      (auto simp: ell2-to-hilbert-ket Rep-chilbert2ell2 is-normal-some-chilbert-basis)
qed

```

lemma unitary-ell2-to-hilbert[simp]: ⟨unitary ell2-to-hilbert⟩

```

proof (rule surj-isometry-is-unitary)
  show ⟨isometry (ell2-to-hilbert :: 'a chilbert2ell2 ell2 ⇒CL -)⟩
  proof (rule orthogonal-on-basis-is-isometry)
    show ⟨ccspan (range ket) = top⟩
      by auto
    fix x y :: ⟨'a chilbert2ell2 ell2⟩
    assume ⟨x ∈ range ket⟩ ⟨y ∈ range ket⟩
    then obtain x' y' where [simp]: ⟨x = ket x'⟩ ⟨y = ket y'⟩
      by auto
    show ⟨(ell2-to-hilbert *V x) •C (ell2-to-hilbert *V y) = x •C y⟩
  proof (cases ⟨x'=y'⟩)
    case True
      hence Rep-chilbert2ell2 y' •C Rep-chilbert2ell2 y' = 1
        using Rep-chilbert2ell2 cnorm-eq-1 is-normal-some-chilbert-basis by blast
      then show ?thesis using True
        by (auto simp: ell2-to-hilbert-ket)
  next

```



```

    case False
    hence is-orthogonal (Rep-chilbert2ell2 x) (Rep-chilbert2ell2 y)
    by (metis Rep-chilbert2ell2 Rep-chilbert2ell2-inject is-ortho-set-def is-ortho-set-some-chilbert-basis)
    then show ?thesis
      using False by (auto simp: ell2-to-hilbert-ket cinner-ket)
  qed
qed
have ⟨cblinfun-apply ell2-to-hilbert ' range ket ⊇ some-chilbert-basis⟩
  by (metis Rep-chilbert2ell2-cases UNIV-I ell2-to-hilbert-ket image-eqI subsetI)
then have ⟨ell2-to-hilbert *S top ≥ ccspan some-chilbert-basis⟩ (is <- ≥ ...)
  by (smt (verit, del-insts) cblinfun-image-ccspan ccspan-mono ccspan-range-ket)
also have ⟨... = top⟩
  by simp
finally show ⟨ell2-to-hilbert *S top = top⟩
  by (simp add: top.extremum-unique)
qed

lemma ell2-to-hilbert-adj-ket: ⟨ell2-to-hilbert* *V ψ = ket (Abs-chilbert2ell2 ψ)⟩ if ⟨ψ ∈ some-chilbert-basis⟩
  using ell2-to-hilbert-ket unitary-ell2-to-hilbert
  by (metis (no-types, lifting) cblinfun-apply-cblinfun-compose cblinfun-id-cblinfun-apply that type-definition.Abs-inverse type-definition-chilbert2ell2 unitaryD1)

definition ⟨cr-chilbert2ell2-ell2 x y ↔ ell2-to-hilbert *V x = y⟩

lemma bi-unique-cr-chilbert2ell2-ell2[transfer-rule]: ⟨bi-unique cr-chilbert2ell2-ell2⟩
  by (metis (no-types, opaque-lifting) bi-unique-def cblinfun-apply-cblinfun-compose cr-chilbert2ell2-ell2-def id-cblinfun-apply unitaryD1 unitary-ell2-to-hilbert)
lemma bi-total-cr-chilbert2ell2-ell2[transfer-rule]: ⟨bi-total cr-chilbert2ell2-ell2⟩
  by (metis (no-types, opaque-lifting) bi-total-def cblinfun-apply-cblinfun-compose cr-chilbert2ell2-ell2-def id-cblinfun-apply unitaryD2 unitary-ell2-to-hilbert)

named-theorems c2l2l2

lemma c2l2l2-cinner[c2l2l2]:
  includes lifting-syntax
  shows ⟨(cr-chilbert2ell2-ell2 == => cr-chilbert2ell2-ell2 == => (=)) cinner cinner⟩
proof -
  have *: ⟨ket x •C ket y = (ell2-to-hilbert *V ket x) •C (ell2-to-hilbert *V ket y)⟩ for x y :: 'a chilbert2ell2
    by (metis Rep-chilbert2ell2 Rep-chilbert2ell2-inverse cinner-adj-right ell2-to-hilbert-adj-ket ell2-to-hilbert-ket)
  have ⟨x •C y = (ell2-to-hilbert *V x) •C (ell2-to-hilbert *V y)⟩ for x y :: 'a chilbert2ell2 ell2
    apply (rule fun-cong[where x=x])
    apply (rule bounded-antilinear-equal-ket)
    apply (intro bounded-linear-intros)
    apply (intro bounded-linear-intros)
    apply (rule fun-cong[where x=y])

```

```

    apply (rule bounded-clinear-equal-ket)
    apply (intro bounded-linear-intros)
    apply (intro bounded-linear-intros)
    by (simp add: *)
  then show ?thesis
    by (auto intro!: rel-funI simp: cr-chilbert2ell2-ell2-def)
qed

```

```

lemma c2l2l2-norm[c2l2l2]:
  includes lifting-syntax
  shows ⟨(cr-chilbert2ell2-ell2 == => (=)) norm norm⟩
  apply (subst norm-eq-sqrt-cinner[abs-def])
  apply (subst (2) norm-eq-sqrt-cinner[abs-def])
  using c2l2l2-cinner[transfer-rule] apply fail?
  by transfer-prover

```

```

lemma c2l2l2-scaleC[c2l2l2]:
  includes lifting-syntax
  shows ⟨((=) == => cr-chilbert2ell2-ell2 == => cr-chilbert2ell2-ell2) scaleC scaleC⟩
proof –
  have ⟨ell2-to-hilbert *V c *C x = c *C (ell2-to-hilbert *V x)⟩ for c and x :: ⟨'a chilbert2ell2 ell2⟩
    by (simp add: cblinfun.scaleC-right)
  then show ?thesis
    by (auto intro!: rel-funI simp: cr-chilbert2ell2-ell2-def)
qed

```

```

lemma c2l2l2-zero[c2l2l2]:
  includes lifting-syntax
  shows ⟨cr-chilbert2ell2-ell2 0 0⟩
  unfolding cr-chilbert2ell2-ell2-def by simp

```

```

lemma c2l2l2-is-ortho-set[c2l2l2]:
  includes lifting-syntax
  shows ⟨(rel-set cr-chilbert2ell2-ell2 == => (=)) is-ortho-set (is-ortho-set :: 'a:: {chilbert-space, not-singleton} set => bool)⟩
  unfolding is-ortho-set-def
  using c2l2l2[where 'a='a, transfer-rule] apply fail?
  by transfer-prover

```

```

lemma c2l2l2-ccspan[c2l2l2]:
  includes lifting-syntax
  shows ⟨(rel-set cr-chilbert2ell2-ell2 == => rel-ccsubspace cr-chilbert2ell2-ell2) ccspan ccspan⟩
proof (rule rel-funI, rename-tac A B)
  fix A and B :: ⟨'a set⟩
  assume ⟨rel-set cr-chilbert2ell2-ell2 A B⟩

```

```

then have ⟨ $B = \text{ell2-to-hilbert } 'A$ ⟩
by (metis (no-types, lifting) bi-unique-cr-chilbert2ell2-ell2 bi-unique-rel-set-lemma cr-chilbert2ell2-ell2-def image-cong)
then have ⟨ $\text{space-as-set } (\text{ccspan } B) = \text{ell2-to-hilbert } ' \text{space-as-set } (\text{ccspan } A)$ ⟩
by (subst space-as-set-image-commute[where  $V = \langle \text{ell2-to-hilbert} \rangle$ ])
      (auto intro: unitaryD2 simp: cblinfun-image-ccspan)
then have ⟨ $\text{rel-set } \text{cr-chilbert2ell2-ell2 } (\text{space-as-set } (\text{ccspan } A)) (\text{space-as-set } (\text{ccspan } B))$ ⟩
by (smt (verit, best) cr-chilbert2ell2-ell2-def image-iff rel-setI)
then show ⟨ $\text{rel-ccsubspace } \text{cr-chilbert2ell2-ell2 } (\text{ccspan } A) (\text{ccspan } B)$ ⟩
by (simp add: rel-ccsubspace-def)
qed

```

```

lemma ell2-to-hilbert-adj-ell2-to-hilbert [simp]:  $\text{ell2-to-hilbert} *_{\mathcal{V}} \text{ell2-to-hilbert} *_{\mathcal{V}} x = x$ 
using unitary-ell2-to-hilbert unfolding unitary-def
by (metis cblinfun-apply-cblinfun-compose cblinfun-id-cblinfun-apply)

```

```

lemma ell2-to-hilbert-ell2-to-hilbert-adj [simp]:  $\text{ell2-to-hilbert} *_{\mathcal{V}} \text{ell2-to-hilbert} *_{\mathcal{V}} x = x$ 
using unitary-ell2-to-hilbert unfolding unitary-def
by (metis cblinfun-apply-cblinfun-compose cblinfun-id-cblinfun-apply)

```

```

lemma bi-total-rel-ccsubspace-cr-chilbert2ell2-ell2 [transfer-rule]:
  ⟨ $\text{bi-total } (\text{rel-ccsubspace } \text{cr-chilbert2ell2-ell2})$ ⟩
apply (rule bi-totalI)
subgoal
by (rule left-total-rel-ccsubspace[where  $U = \text{ell2-to-hilbert}$  and  $V = \langle \text{ell2-to-hilbert} \rangle$ ])
      (auto simp: cr-chilbert2ell2-ell2-def)[3]
subgoal
by (rule right-total-rel-ccsubspace[where  $U = \langle \text{ell2-to-hilbert} \rangle$  and  $V = \langle \text{ell2-to-hilbert} \rangle$ ])
      (auto simp: cr-chilbert2ell2-ell2-def)
done

```

```

lemma c2l2l2-top[c2l2l2]:
includes lifting-syntax
shows ⟨ $(\text{rel-ccsubspace } \text{cr-chilbert2ell2-ell2}) \text{ top top}$ ⟩
unfolding rel-ccsubspace-def
by (simp add: UNIV-transfer bi-total-cr-chilbert2ell2-ell2)

```

```

lemma c2l2l2-is-onb[c2l2l2]:
includes lifting-syntax
shows ⟨ $(\text{rel-set } \text{cr-chilbert2ell2-ell2} ==> (=)) \text{ is-onb is-onb}$ ⟩
unfolding is-onb-def
using c2l2l2[where  $'a = 'a$ , transfer-rule] apply fail?
by transfer-prover

```

```

unbundle no cblinfun-syntax

```

```

end

```

6 Weak-Operator-Topology – Weak operator topology on complex bounded operators

theory *Weak-Operator-Topology*

imports *Misc-Tensor-Product Strong-Operator-Topology Positive-Operators Wlog.Wlog*

begin

unbundle *cblinfun-syntax*

definition *cweak-operator-topology*::('a::complex-normed-vector \Rightarrow_{CL} 'b::complex-inner) topology
where *cweak-operator-topology* = *pullback-topology UNIV* ($\lambda a (x,y). cinner\ x\ (a\ *_V\ y)$) *euclidean*

lemma *cweak-operator-topology-topospace[simp]*:

topospace cweak-operator-topology = *UNIV*

unfolding *cweak-operator-topology-def topspace-pullback-topology topspace-euclidean* **by** *auto*

lemma *cweak-operator-topology-basis*:

fixes *f*::('a::complex-normed-vector \Rightarrow_{CL} 'b::complex-inner) **and** *U*::'i \Rightarrow complex set **and** *x*::'i \Rightarrow 'b **and** *y*::'i \Rightarrow 'a

assumes *finite I* $\wedge i. i \in I \implies open\ (U\ i)$

shows *openin cweak-operator-topology* $\{f. \forall i \in I. cinner\ (x\ i)\ (f\ *_V\ y\ i) \in U\ i\}$

proof –

have *open* $\{g::('b \times 'a) \Rightarrow complex. \forall i \in I. g\ (x\ i,\ y\ i) \in U\ i\}$

by (*rule product-topology-basis'[OF assms]*)

moreover have $\{f. \forall i \in I. cinner\ (x\ i)\ (f\ *_V\ y\ i) \in U\ i\}$

= $(\lambda f (x,y). cinner\ x\ (f\ *_V\ y)) - \dots \cap UNIV$

by *auto*

ultimately show *?thesis*

unfolding *cweak-operator-topology-def* **by** (*subst openin-pullback-topology*) *auto*

qed

lemma *wot-weaker-than-sot*:

continuous-map cstrong-operator-topology cweak-operator-topology $(\lambda f. f)$

proof –

have *: $\langle continuous-on\ UNIV\ ((\lambda z. cinner\ x\ z) \circ (\lambda f. f\ y)) \rangle$ **for** *x* :: 'b **and** *y* :: 'a

apply (*rule continuous-on-compose*)

by (*auto intro: continuous-on-compose continuous-at-imp-continuous-on*)

have *: $\langle continuous-map\ euclidean\ euclidean\ (\lambda f (x::'b,\ y::'a). x \cdot_C f\ y) \rangle$

apply *simp*

apply (*rule continuous-on-coordinatewise-then-product*)

using * **by** *auto*

have *: $\langle continuous-map\ (pullback-topology\ UNIV\ (*_V)\ euclidean)\ euclidean\ ((\lambda f (x::'b,\ a::'a). x \cdot_C f\ a) \circ (*_V)) \rangle$

apply (*rule continuous-map-pullback*)

using * **by** *simp*

have *: $\langle continuous-map\ (pullback-topology\ UNIV\ (*_V)\ euclidean)\ euclidean\ ((\lambda a (x::'b,\ y::'a). x \cdot_C (a *_V y)) \circ (\lambda f. f)) \rangle$

apply (*subst asm-rl[of $\langle ((\lambda a (x, y). x \cdot_C (a *_V y)) \circ (\lambda f. f)) = (\lambda f (a,b). cinner\ a\ (f\ b))$*)

```

o (*V)⟩])
  using * by auto
  show ?thesis
  unfolding cstrong-operator-topology-def cweak-operator-topology-def
  apply (rule continuous-map-pullback')
  using * by auto
qed

```

lemma *cweak-operator-topology-weaker-than-euclidean*:

```

  continuous-map euclidean cweak-operator-topology (λf. f)
  by (metis (mono-tags, lifting) continuous-map-compose continuous-map-eq cstrong-operator-topology-weaker-than-euclidean wot-weaker-than-sot o-def)

```

lemma *cweak-operator-topology-cinner-continuous*:

```

  continuous-map cweak-operator-topology euclidean (λf. cinner x (f *V y))
proof –
  have continuous-map cweak-operator-topology euclidean ((λf. f (x,y)) o (λa (x,y). cinner x (a *V y)))
  unfolding cweak-operator-topology-def apply (rule continuous-map-pullback)
  using continuous-on-product-coordinates by fastforce
  then show ?thesis unfolding comp-def by simp
qed

```

lemma *continuous-on-cweak-operator-topo-iff-coordinatewise*:

```

  continuous-map T cweak-operator-topology f
  ↔ (∀ x y. continuous-map T euclidean (λz. cinner x (f z *V y)))
proof (intro iffI allI)
  fix x::'c and y::'b
  assume continuous-map T cweak-operator-topology f
  with continuous-map-compose[OF this cweak-operator-topology-cinner-continuous]
  have continuous-map T euclidean ((λf. cinner x (f *V y)) o f)
  by simp
  then show continuous-map T euclidean (λz. cinner x (f z *V y))
  unfolding comp-def by auto
next
  assume *: ⟨∀ x y. continuous-map T euclidean (λz. x •C (f z *V y))⟩
  then have *: continuous-map T euclidean ((λa (x,y). cinner x (a *V y)) o f)
  by (auto simp flip: euclidean-product-topology)
  show continuous-map T cweak-operator-topology f
  unfolding cweak-operator-topology-def
  apply (rule continuous-map-pullback')
  by (auto simp add: *)
qed

```

```

typedef (overloaded) ('a,'b) cblinfun-wot = ⟨UNIV :: ('a::complex-normed-vector ⇒CL 'b::complex-inner) set⟩ ..
setup-lifting type-definition-cblinfun-wot

```

instantiation *cblinfun-wot* :: (complex-normed-vector, complex-inner) complex-vector **begin**
lift-definition *scaleC-cblinfun-wot* :: $\langle \text{complex} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$

is $\langle \text{scaleC} \rangle$.

lift-definition *uminus-cblinfun-wot* :: $\langle ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** *uminus* .

lift-definition *zero-cblinfun-wot* :: $\langle ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** 0 .

lift-definition *minus-cblinfun-wot* :: $\langle ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** *minus* .

lift-definition *plus-cblinfun-wot* :: $\langle ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** *plus* .

lift-definition *scaleR-cblinfun-wot* :: $\langle \text{real} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** *scaleR* .

instance

apply (*intro-classes*; *transfer*)

by (*auto simp add: scaleR-scaleC scaleC-add-right scaleC-add-left*)

end

instantiation *cblinfun-wot* :: (complex-normed-vector, complex-inner) topological-space **begin**

lift-definition *open-cblinfun-wot* :: $\langle ('a, 'b) \text{ cblinfun-wot set} \Rightarrow \text{bool} \rangle$ **is** $\langle \text{openin cweak-operator-topology} \rangle$

.

instance

proof *intro-classes*

show $\langle \text{open (UNIV :: ('a, 'b) cblinfun-wot set)} \rangle$

apply *transfer*

by (*metis cweak-operator-topology-topspace openin-topspace*)

show $\langle \text{open } S \implies \text{open } T \implies \text{open } (S \cap T) \rangle$ **for** $S T :: \langle ('a, 'b) \text{ cblinfun-wot set} \rangle$

apply *transfer by auto*

show $\langle \forall S \in K. \text{open } S \implies \text{open } (\bigcup K) \rangle$ **for** $K :: \langle ('a, 'b) \text{ cblinfun-wot set set} \rangle$

apply *transfer by auto*

qed

end

lemma *transfer-nhds-cweak-operator-topology[transfer-rule]*:

includes *lifting-syntax*

shows $\langle (\text{cr-cblinfun-wot} \implies \text{rel-filter cr-cblinfun-wot}) (\text{nhdsin cweak-operator-topology}) \text{nhds} \rangle$

unfolding *nhds-def nhdsin-def*

apply (*simp add: cweak-operator-topology-topspace*)

by *transfer-prover*

lemma *limitin-cweak-operator-topology*:

$\langle \text{limitin cweak-operator-topology } f l F \iff (\forall a b. ((\lambda i. a \cdot_C (f i *_V b)) \longrightarrow a \cdot_C (l *_V b)) F) \rangle$

by (*simp add: cweak-operator-topology-def limitin-pullback-topology tendsto-coordinatewise*)

lemma *filterlim-cweak-operator-topology*: $\langle \text{filterlim } f (\text{nhdsin cweak-operator-topology } l) = \text{limitin cweak-operator-topology } f l \rangle$

```

by (auto simp: cweak-operator-topology-topospace simp flip: filterlim-nhdsin-iff-limitin)

instance cblinfun-wot :: (complex-normed-vector, complex-inner) t2-space
proof intro-classes
  fix a b :: ⟨('a,'b) cblinfun-wot⟩
  show ⟨a ≠ b ⟹ ∃ U V. open U ∧ open V ∧ a ∈ U ∧ b ∈ V ∧ U ∩ V = {}⟩
  proof transfer
    fix a b :: ⟨'a ⇒CL 'b⟩
    assume ⟨a ≠ b⟩
    then obtain φ ψ where ⟨φ ·C (a *V ψ) ≠ φ ·C (b *V ψ)⟩
      by (meson cblinfun-eqI cinner-extensionality)
    then obtain U' V' where ⟨open U'⟩ ⟨open V'⟩ and inU': ⟨φ ·C (a *V ψ) ∈ U'⟩ and inV':
      ⟨φ ·C (b *V ψ) ∈ V'⟩ and ⟨U' ∩ V' = {}⟩
    by (meson hausdorff)
    define U V :: ⟨('a ⇒CL 'b) set⟩ where ⟨U = {f. ∀ i ∈ {}. φ ·C (f *V ψ) ∈ U'}⟩ and ⟨V
  = {f. ∀ i ∈ {}. φ ·C (f *V ψ) ∈ V'}⟩
    have ⟨openin cweak-operator-topology U⟩
      unfolding U-def apply (rule cweak-operator-topology-basis)
      using ⟨open U'⟩ by auto
    moreover have ⟨openin cweak-operator-topology V⟩
      unfolding V-def apply (rule cweak-operator-topology-basis)
      using ⟨open V'⟩ by auto
    ultimately show ⟨∃ U V. openin cweak-operator-topology U ∧ openin cweak-operator-topology
  V ∧ a ∈ U ∧ b ∈ V ∧ U ∩ V = {}⟩
      apply (rule-tac exI[of - U])
      apply (rule-tac exI[of - V])
      using inU' inV' ⟨U' ∩ V' = {}⟩ by (auto simp: U-def V-def)
  qed
qed

lemma Domainp-cr-cblinfun-wot[simp]: ⟨Domainp cr-cblinfun-wot = (λ-. True)⟩
  by (metis (no-types, opaque-lifting) DomainPI cblinfun-wot.left-total left-totalE)

lemma Rangep-cr-cblinfun-wot[simp]: ⟨Rangep cr-cblinfun-wot = (λ-. True)⟩
  by (meson RangePI cr-cblinfun-wot-def)

lemma transfer-euclidean-cweak-operator-topology[transfer-rule]:
  includes lifting-syntax
  shows ⟨(rel-topology cr-cblinfun-wot) cweak-operator-topology euclidean⟩
proof (unfold rel-topology-def, intro conjI allI impI)
  show ⟨(rel-set cr-cblinfun-wot ==> (=)) (openin cweak-operator-topology) (openin euclidean)⟩
  apply (auto simp: rel-topology-def cr-cblinfun-wot-def rel-set-def intro!: rel-funI)
  apply transfer
  apply auto
  apply (meson openin-subopen subsetI)
  apply transfer
  apply auto
  by (meson openin-subopen subsetI)
next

```

```

fix U :: ⟨('a ⇒CL 'b) set⟩
assume ⟨openin cweak-operator-topology U⟩
show ⟨Domainp (rel-set cr-cblinfun-wot) U⟩
  by (simp add: Domainp-set)
next
fix U :: ⟨('a, 'b) cblinfun-wot set⟩
assume ⟨openin euclidean U⟩
show ⟨Rangep (rel-set cr-cblinfun-wot) U⟩
  by (simp add: Rangep-set)
qed

```

lemma *openin-cweak-operator-topology*: $\langle \text{openin cweak-operator-topology } U \iff (\exists V. \text{open } V \wedge U = (\lambda a (x,y). \text{cinner } x (a *_V y)) - 'V) \rangle$
 by (simp add: cweak-operator-topology-def openin-pullback-topology)

lemma *cweak-operator-topology-plus-cont*: $\langle \text{LIM } (x,y) \text{ nhdsin cweak-operator-topology } a \times_F \text{ nhdsin cweak-operator-topology } b. \rangle$
 $x + y :=> \text{nhdsin cweak-operator-topology } (a + b) \rangle$

```

proof -
show ?thesis
  unfolding cweak-operator-topology-def
  apply (rule-tac pullback-topology-bi-cont[where f'=plus])
  by (auto simp: case-prod-unfold tendsto-add-Pair cinner-add-right cblinfun.add-left)
qed

```

instance *cblinfun-wot* :: (complex-normed-vector, complex-inner) topological-group-add

proof *intro-classes*

```

show ⟨((λx. fst x + snd x) ⟶ a + b) (nhds a ×F nhds b)⟩ for a b :: ⟨('a,'b) cblinfun-wot⟩
  apply transfer
  using cweak-operator-topology-plus-cont
  by (auto simp: case-prod-unfold)

```

```

have *: ⟨continuous-map cweak-operator-topology cweak-operator-topology uminus⟩
  apply (subst continuous-on-cweak-operator-topo-iff-coordinatewise)
  apply (rewrite at ⟨(λz. x •C (- z *_V y))⟩ in ⟨∀ x y. □⟩ to ⟨(λz. - x •C (z *_V y))⟩) DEA-
  DID.rel-mono-strong)
  by (auto simp: cweak-operator-topology-cinner-continuous cblinfun.minus-left cblinfun.minus-right)
show ⟨(uminus ⟶ - a) (nhds a)⟩ for a :: ⟨('a,'b) cblinfun-wot⟩
  apply (subst tendsto-at-iff-tendsto-nhds[symmetric])
  apply (subst isCont-def[symmetric])
  apply (rule continuous-on-interior[where S=UNIV])
  apply (subst continuous-map-iff-continuous2[symmetric])
  apply transfer
  using * by auto
qed

```

lemma *continuous-map-left-comp-wot*:

$\langle \text{continuous-map cweak-operator-topology cweak-operator-topology } (\lambda a::'a::\text{complex-normed-vector} \Rightarrow_{CL} -. b \circ_{CL} a) \rangle$


```

for b :: ⟨'b::chilbert-space ⇒CL 'c::complex-inner⟩
proof -
have **: ⟨((λf::'b × 'a ⇒ complex. f (b* *V x, y)) - 'B ∩ UNIV)
= (PiE UNIV (λz. if z = (b* *V x, y) then B else UNIV))⟩
for x :: 'c and y :: 'a and B :: ⟨complex set⟩
by (auto simp: PiE-def Pi-def)
have *: ⟨continuous-on UNIV (λf::'b × 'a ⇒ complex. f (b* *V x, y))⟩ for x y
unfolding continuous-on-open-vimage[OF open-UNIV]
apply (intro allI impI)
apply (subst **)
apply (rule open-PiE)
by auto
have *: ⟨continuous-on UNIV (λ(f::'b × 'a ⇒ complex) (x, y). f (b* *V x, y))⟩
apply (rule continuous-on-coordinatewise-then-product)
using * by auto
show ?thesis
unfolding cweak-operator-topology-def
apply (rule continuous-map-pullback')
apply (subst asm-rl[of ⟨((λ(a::'a⇒CL'c) (x, y). x •C (a *V y)) ∘ (oCL) b) = (λf (x,y). f
(b* *V x,y)) ∘ (λa (x, y). x •C (a *V y))⟩]
apply (auto intro!: ext simp: cinner-adj-left)[1]
apply (rule continuous-map-pullback)
using * by auto
qed

```

lemma *continuous-map-scaleC-wot*: ⟨continuous-map cweak-operator-topology cweak-operator-topology

```

(scaleC c :: ('a::complex-normed-vector ⇒CL 'b::chilbert-space) ⇒ -)⟩
apply (subst asm-rl[of ⟨scaleC c = (oCL) (c *C id-cblinfun)⟩])
apply auto[1]
by (rule continuous-map-left-comp-wot)

```

lemma *continuous-scaleC-wot*: ⟨continuous-on X (scaleC c :: (-::complex-normed-vector, -::chilbert-space) cblinfun-wot ⇒ -)⟩

```

apply (rule continuous-on-subset[rotated, where s=UNIV], simp)
apply (subst continuous-map-iff-continuous2[symmetric])
apply transfer
by (rule continuous-map-scaleC-wot)

```

lemma *wot-closure-is-csubspace[simp]*:

```

fixes A::('a::complex-normed-vector, 'b::chilbert-space) cblinfun-wot set
assumes ⟨csubspace A⟩
shows ⟨csubspace (closure A)⟩

```

proof (rule complex-vector.subspaceI)

include *lattice-syntax*

show 0: ⟨0 ∈ closure A⟩

by (simp add: assms closure-def complex-vector.subspace-0)

show ⟨x + y ∈ closure A⟩ **if** ⟨x ∈ closure A⟩ ⟨y ∈ closure A⟩ **for** x y

proof -

```

define FF where  $\langle FF = ((nhds\ x \sqcap principal\ A) \times_F (nhds\ y \sqcap principal\ A)) \rangle$ 
have nt:  $\langle FF \neq bot \rangle$ 
  by (simp add: prod-filter-eq-bot that(1) that(2) FF-def flip: closure-nhds-principal)
have  $\langle \forall_F\ x\ in\ FF.\ fst\ x \in A \rangle$ 
  unfolding FF-def
  by (smt (verit, ccfv-SIG) eventually-prod-filter fst-conv inf-sup-ord(2) le-principal)
moreover have  $\langle \forall_F\ x\ in\ FF.\ snd\ x \in A \rangle$ 
  unfolding FF-def
  by (smt (verit, ccfv-SIG) eventually-prod-filter snd-conv inf-sup-ord(2) le-principal)
ultimately have FF-plus:  $\langle \forall_F\ x\ in\ FF.\ fst\ x + snd\ x \in A \rangle$ 
  by (smt (verit, best) assms complex-vector.subspace-add eventually-elim2)

have  $\langle (fst \longrightarrow x) ((nhds\ x \sqcap principal\ A) \times_F (nhds\ y \sqcap principal\ A)) \rangle$ 
  apply (simp add: filterlim-def)
  using filtermap-fst-prod-filter
  using le-inf-iff by blast
moreover have  $\langle (snd \longrightarrow y) ((nhds\ x \sqcap principal\ A) \times_F (nhds\ y \sqcap principal\ A)) \rangle$ 
  apply (simp add: filterlim-def)
  using filtermap-snd-prod-filter
  using le-inf-iff by blast
ultimately have  $\langle (id \longrightarrow (x,y))\ FF \rangle$ 
  by (simp add: filterlim-def nhds-prod prod-filter-mono FF-def)

moreover note tendsto-add-Pair[of x y]
ultimately have  $\langle ((\lambda x.\ fst\ x + snd\ x) \circ id) \longrightarrow (\lambda x.\ fst\ x + snd\ x)\ (x,y)\ FF \rangle$ 
  unfolding filterlim-def nhds-prod
  by (smt (verit, best) filterlim-compose filterlim-def filterlim-filtermap fst-conv snd-conv
tendsto-compose-filtermap)

then have  $\langle ((\lambda x.\ fst\ x + snd\ x) \longrightarrow (x+y))\ FF \rangle$ 
  by simp
then show  $\langle x + y \in closure\ A \rangle$ 
  using nt FF-plus by (rule limit-in-closure)
qed
show  $\langle c *_{C}\ x \in closure\ A \rangle$  if  $\langle x \in closure\ A \rangle$  for  $x\ c$ 
  using that
  using image-closure-subset[where S=A and T= $\langle closure\ A \rangle$  and  $f = \langle scaleC\ c \rangle$ , OF contin-
uous-scaleC-wot]
  apply auto
  by (metis 0 assms closure-subset csubspace-scaleC-invariant imageI in-mono scaleC-eq-0-iff)
qed

lemma [transfer-rule]:
includes lifting-syntax
shows  $\langle (rel\ set\ cr\ cblinfun\ wot ==> (=))\ csubspace\ csubspace \rangle$ 
unfolding complex-vector.subspace-def
by transfer-prover

```

```

lemma [transfer-rule]:
  includes lifting-syntax
  shows ⟨(rel-set cr-cblinfun-wot == => (=)) (closedin cweak-operator-topology) closed⟩
  apply (simp add: closed-def[abs-def] closedin-def[abs-def] cweak-operator-topology-topospace
Compl-eq-Diff-UNIV)
  by transfer-prover

```

```

lemma [transfer-rule]:
  includes lifting-syntax
  shows ⟨(rel-set cr-cblinfun-wot == => rel-set cr-cblinfun-wot) (Abstract-Topology.closure-of
cweak-operator-topology) closure⟩
  apply (subst closure-of-hull[where X=cweak-operator-topology, unfolded cweak-operator-topology-topospace,
simplified, abs-def])
  apply (subst closure-hull[abs-def])
  unfolding hull-def
  by transfer-prover

```

```

lemma wot-closure-is-csubspace'[simp]:
  fixes A::('a::complex-normed-vector =>CL 'b::chilbert-space) set
  assumes ⟨csubspace A⟩
  shows ⟨csubspace (cweak-operator-topology closure-of A)⟩
  using wot-closure-is-csubspace[of ⟨Abs-cblinfun-wot ' A⟩] assms
  apply (transfer fixing: A)
  by simp

```

```

lemma has-sum-closed-cweak-operator-topology:
  fixes A :: ⟨('b::complex-normed-vector =>CL 'c::complex-inner) set⟩
  assumes aA: ⟨ $\bigwedge i. a i \in A$ ⟩
  assumes closed: ⟨closedin cweak-operator-topology A⟩
  assumes subspace: ⟨csubspace A⟩
  assumes has-sum: ⟨ $\bigwedge \varphi \psi. ((\lambda i. \varphi \cdot_C (a i *_V \psi)) \text{has-sum } \varphi \cdot_C (b *_V \psi)) I$ ⟩
  shows ⟨ $b \in A$ ⟩

```

```

proof -
  have 1: ⟨range (sum a)  $\subseteq$  A⟩
  proof -
    have ⟨sum a X  $\in$  A⟩ for X
      apply (induction X rule:infinite-finite-induct)
      by (auto simp add: subspace complex-vector.subspace-0 aA complex-vector.subspace-add)
    then show ?thesis
      by auto
  qed

```

qed

from has-sum

```

have ⟨(( $\lambda F. \sum i \in F. \varphi \cdot_C (a i *_V \psi)$ )  $\longrightarrow$   $\varphi \cdot_C (b *_V \psi)$ ) (finite-subsets-at-top I)⟩ for  $\psi \varphi$ 
  using has-sum-def by blast

```

```

then have ⟨limitin cweak-operator-topology ( $\lambda F. \sum i \in F. a i$ ) b (finite-subsets-at-top I)⟩

```

```

  by (auto simp add: limitin-cweak-operator-topology cblinfun.sum-left cinner-sum-right)

```

```

then show ⟨ $b \in A$ ⟩

```

```

  using 1 closed apply (rule limitin-closedin)

```

by *simp*
qed

lemma *limitin-adj-wot*:

assumes $\langle \text{limitin cweak-operator-topology } f \ l \ F \rangle$
shows $\langle \text{limitin cweak-operator-topology } (\lambda i. (f \ i)^*) \ (l^*) \ F \rangle$

proof –

from *assms*

have $\langle (\lambda i. a \cdot_C (f \ i \ *V \ b)) \longrightarrow a \cdot_C (l \ *V \ b) \rangle \ F \rangle$ for $a \ b$

by (*simp add: limitin-cweak-operator-topology*)

then have $\langle (\lambda i. \text{cnj } (a \cdot_C (f \ i \ *V \ b))) \longrightarrow \text{cnj } (a \cdot_C (l \ *V \ b)) \rangle \ F \rangle$ for $a \ b$

using *tendsto-cnj by blast*

then have $\langle (\lambda i. \text{cnj } (((f \ i)^* \ *V \ a) \cdot_C \ b)) \longrightarrow \text{cnj } ((l^* \ *V \ a) \cdot_C \ b) \rangle \ F \rangle$ for $a \ b$

by (*simp add: cinner-adj-left*)

then have $\langle (\lambda i. b \cdot_C ((f \ i)^* \ *V \ a)) \longrightarrow b \cdot_C (l^* \ *V \ a) \rangle \ F \rangle$ for $a \ b$

by *simp*

then show *?thesis*

by (*simp add: limitin-cweak-operator-topology*)

qed

lemma *hausdorff-cweak-operator-topology[simp]*: $\langle \text{Hausdorff-space cweak-operator-topology} \rangle$

proof (*rule hausdorffI*)

fix $A \ B :: \langle 'a \Rightarrow_{CL} 'b \rangle$ assume $\langle A \neq B \rangle$

then obtain y where $\langle A \ *V \ y \neq B \ *V \ y \rangle$

by (*meson cblinfun-eqI*)

then obtain x where $\langle x \cdot_C (A \ *V \ y) \neq x \cdot_C (B \ *V \ y) \rangle$

using *cinner-extensionality by blast*

then obtain $U' \ V'$ where $\langle \text{open } U' \rangle \langle \text{open } V' \rangle$ and *inside*: $\langle x \cdot_C (A \ *V \ y) \in U' \rangle \langle x \cdot_C (B \ *V \ y) \in V' \rangle$ and *disj*: $\langle U' \cap V' = \{\} \rangle$

by (*meson separation-t2*)

define $U'' \ V''$ where $\langle U'' = \text{Pi}_E \ \text{UNIV} \ (\lambda i. \text{if } i=(x,y) \text{ then } U' \text{ else } \text{UNIV}) \rangle$ and $\langle V'' = \text{Pi}_E \ \text{UNIV} \ (\lambda i. \text{if } i=(x,y) \text{ then } V' \text{ else } \text{UNIV}) \rangle$

from $\langle \text{open } U' \rangle \langle \text{open } V' \rangle$

have $\langle \text{open } U'' \rangle$ and $\langle \text{open } V'' \rangle$

by (*auto simp: U''-def V''-def open-fun-def intro!: product-topology-basis*)

define $U \ V :: \langle 'a \Rightarrow_{CL} 'b \rangle \ \text{set}$ where $\langle U = (\lambda A \ (x, y). x \cdot_C (A \ *V \ y)) - ' U'' \rangle$ and $\langle V = (\lambda a \ (x, y). x \cdot_C (a \ *V \ y)) - ' V'' \rangle$

have *openin*: $\langle \text{openin cweak-operator-topology } U \rangle \langle \text{openin cweak-operator-topology } V \rangle$

using *U-def V-def open U'' open V'' openin-cweak-operator-topology by blast+*

have $\langle A \in U \rangle \langle B \in V \rangle$

using *inside by (auto simp: U-def V-def U''-def V''-def)*

have $\langle U \cap V = \{\} \rangle$

using *disj apply (auto simp: U-def V-def U''-def V''-def PiE-def Pi-iff)*

by (*metis disjoint-iff*)

show $\langle \exists U \ V. \text{openin cweak-operator-topology } U \wedge \text{openin cweak-operator-topology } V \wedge A \in U \wedge B \in V \wedge U \cap V = \{\} \rangle$

using $\langle A \in U \rangle \langle B \in V \rangle \langle U \cap V = \{\} \rangle$ *openin by blast*

qed

lemma *hermitian-limit-hermitian-wot*:

assumes $\langle F \neq \text{bot} \rangle$
assumes *herm*: $\langle \bigwedge i. (a\ i)^* = a\ i \rangle$
assumes *lim*: $\langle \text{limitin cweak-operator-topology } a\ A\ F \rangle$
shows $\langle A^* = A \rangle$
using - - $\langle F \neq \text{bot} \rangle$ *hausdorff-cweak-operator-topology*
apply (rule *limitin-Hausdorff-unique*)
using *lim* **apply** (rule *limitin-adj-wot*)
unfolding *herm* **by** (fact *lim*)

lemma *wot-weaker-than-sot-openin*:

$\langle \text{openin cweak-operator-topology } x \implies \text{openin cstrong-operator-topology } x \rangle$
using *wot-weaker-than-sot* **unfolding** *continuous-map-def* **by** *auto*

lemma *wot-weaker-than-sot-limitin*: $\langle \text{limitin cweak-operator-topology } a\ A\ F \rangle$ **if** $\langle \text{limitin cstrong-operator-topology } a\ A\ F \rangle$

using *that* **unfolding** *filterlim-cweak-operator-topology[symmetric]* *filterlim-cstrong-operator-topology[symmetric]*
apply (rule *filterlim-mono*)
apply (rule *nhdsin-mono*)
by (*auto simp: wot-weaker-than-sot-openin*)

lemma *hermitian-limit-hermitian-sot*:

assumes $\langle F \neq \text{bot} \rangle$
assumes $\langle \bigwedge i. (a\ i)^* = a\ i \rangle$
assumes $\langle \text{limitin cstrong-operator-topology } a\ A\ F \rangle$
shows $\langle A^* = A \rangle$
using *assms(1,2)* **apply** (rule *hermitian-limit-hermitian-wot[where a=a and F=F]*)
using *assms(3)* **by** (rule *wot-weaker-than-sot-limitin*)

lemma *hermitian-sum-hermitian-sot*:

assumes *herm*: $\langle \bigwedge i. (a\ i)^* = a\ i \rangle$
assumes *sum*: $\langle \text{has-sum-in cstrong-operator-topology } a\ X\ A \rangle$
shows $\langle A^* = A \rangle$

proof -

from *herm* **have** *herm-sum*: $\langle (\text{sum } a\ F)^* = \text{sum } a\ F \rangle$ **for** *F*
by (*simp add: sum-adj*)

show *?thesis*

using - *herm-sum sum[unfolded has-sum-in-def]*

apply (rule *hermitian-limit-hermitian-sot*)

by *simp*

qed

lemma *wot-is-norm-topology-findim[simp]*:

$\langle \text{cweak-operator-topology } :: ('a::\{\text{cfinite-dim}, \text{chilbert-space}\} \Rightarrow_{CL} 'b::\{\text{cfinite-dim}, \text{chilbert-space}\})$
topology) = *euclidean* \rangle

proof -

have $\langle \text{continuous-map euclidean cweak-operator-topology } \text{id} \rangle$

by (*simp add: id-def cweak-operator-topology-weaker-than-euclidean*)

moreover have $\langle \text{continuous-map cweak-operator-topology euclidean } (id :: 'a \Rightarrow_{CL} 'b \Rightarrow -) \rangle$
proof (rule continuous-map-iff-preserves-convergence)
fix l and $F :: \langle ('a \Rightarrow_{CL} 'b) \text{ filter} \rangle$
assume $lim\text{-wot}$: $\langle \text{limitin cweak-operator-topology } id \ l \ F \rangle$
obtain $A :: \langle 'a \text{ set} \rangle$ **where** $\langle is\text{-onb } A \rangle$
using $is\text{-onb-some-chilbert-basis}$ **by** $blast$
then have idA : $\langle id\text{-cblinfun} = (\sum x \in A. \text{selfbutter } x) \rangle$
using $butterflies\text{-sum-id-finite}$ **by** $blast$
obtain $B :: \langle 'b \text{ set} \rangle$ **where** $\langle is\text{-onb } B \rangle$
using $is\text{-onb-some-chilbert-basis}$ **by** $blast$
then have idB : $\langle id\text{-cblinfun} = (\sum x \in B. \text{selfbutter } x) \rangle$
using $butterflies\text{-sum-id-finite}$ **by** $blast$
from $lim\text{-wot}$ **have** $\langle ((\lambda x. b \cdot_C (x *_V a)) \longrightarrow b \cdot_C (l *_V a)) \ F \rangle$ **for** $a \ b$
by ($simp$ add: $limitin\text{-cweak-operator-topology}$)
then have $\langle ((\lambda x. (b \cdot_C (x *_V a)) *_C \text{butterfly } b \ a) \longrightarrow (b \cdot_C (l *_V a)) *_C \text{butterfly } b \ a)$
 $F \rangle$ **for** $a \ b$
by ($simp$ add: $tendsto\text{-scale}C$)
then have $\langle ((\lambda x. \text{selfbutter } b \ o_{CL} \ x \ o_{CL} \ \text{selfbutter } a) \longrightarrow (\text{selfbutter } b \ o_{CL} \ l \ o_{CL} \ \text{selfbutter}$
 $a)) \ F \rangle$ **for** $a \ b$
by ($simp$ add: $cblinfun\text{-comp-butterfly}$)
then have $\langle ((\lambda x. \sum b \in B. \text{selfbutter } b \ o_{CL} \ x \ o_{CL} \ \text{selfbutter } a) \longrightarrow (\sum b \in B. \text{selfbutter } b$
 $o_{CL} \ l \ o_{CL} \ \text{selfbutter } a)) \ F \rangle$ **for** a
by (rule $tendsto\text{-sum}$)
then have $\langle ((\lambda x. x \ o_{CL} \ \text{selfbutter } a) \longrightarrow (l \ o_{CL} \ \text{selfbutter } a)) \ F \rangle$ **for** a
by ($simp$ add: $flip$: $cblinfun\text{-compose-sum-left } idB$)
then have $\langle ((\lambda x. \sum a \in A. x \ o_{CL} \ \text{selfbutter } a) \longrightarrow (\sum a \in A. l \ o_{CL} \ \text{selfbutter } a)) \ F \rangle$
by (rule $tendsto\text{-sum}$)
then have $\langle id \longrightarrow l \rangle \ F$
by ($simp$ add: $flip$: $cblinfun\text{-compose-sum-right } idA \ id\text{-def}$)
then show $\langle \text{limitin euclidean } id \ (id \ l) \ F \rangle$
by $simp$
qed
ultimately show $?thesis$
by ($auto$ $simp$: $topology\text{-finer-continuous-id[symmetric]}$ $simp$ $flip$: $openin\text{-inject}$)
qed

lemma $sot\text{-is-norm-topology-fin-dim[simp]}$:

$\langle (cstrong\text{-operator-topology} :: ('a :: \{cfinite\text{-dim}, chilbert\text{-space}\} \Rightarrow_{CL} 'b :: \{cfinite\text{-dim}, chilbert\text{-space}\}))$
 $topology) = euclidean \rangle$
proof –
have 1: $\langle \text{continuous-map euclidean cstrong-operator-topology } (id :: 'a \Rightarrow_{CL} 'b \Rightarrow -) \rangle$
by ($simp$ add: $id\text{-def cstrong-operator-topology-weaker-than-euclidean}$)
have $\langle \text{continuous-map cstrong-operator-topology cweak-operator-topology } (id :: 'a \Rightarrow_{CL} 'b \Rightarrow -) \rangle$
by ($metis$ $eq\text{-id-iff wot-weaker-than-sot}$)
then have 2: $\langle \text{continuous-map cstrong-operator-topology euclidean } (id :: 'a \Rightarrow_{CL} 'b \Rightarrow -) \rangle$
by ($simp$ only: $wot\text{-is-norm-topology-findim}$)
from 1 2

```

  show ?thesis
  by (auto simp: topology-finer-continuous-id[symmetric] simp flip: openin-inject)
qed

lemma regular-space-wot: ⟨regular-space cweak-operator-topology⟩
proof -
  have ⟨regular-space (product-topology (λi::'b×'a. euclidean :: complex topology) UNIV)⟩
  by (simp add: regular-space-product-topology)
  then have ⟨regular-space (euclidean :: ('b×'a ⇒ complex) topology)⟩
  using euclidean-product-topology by metis
  then show ?thesis
  unfolding cweak-operator-topology-def
  by (rule-tac regular-space-pullback)
qed

instance cblinfun-wot :: (complex-normed-vector, complex-inner) t3-space
  apply intro-classes
  apply transfer
  using regular-space-wot
  by (auto simp add: regular-space-def disjnt-def)

instantiation cblinfun-wot :: (hilbert-space, hilbert-space) order begin
lift-definition less-eq-cblinfun-wot :: ⟨('a, 'b) cblinfun-wot ⇒ ('a, 'b) cblinfun-wot ⇒ bool⟩ is
less-eq.
lift-definition less-cblinfun-wot :: ⟨('a, 'b) cblinfun-wot ⇒ ('a, 'b) cblinfun-wot ⇒ bool⟩ is less.
instance
  apply (intro-classes; transfer')
  by auto
end

instance cblinfun-wot :: (hilbert-space, hilbert-space) ordered-comm-monoid-add
proof
  fix a b c :: ⟨('a, 'b) cblinfun-wot⟩
  assume ⟨a ≤ b⟩
  then show ⟨c + a ≤ c + b⟩
  apply transfer'
  by simp
qed

lemma limitin-wot-add:
  assumes ⟨limitin cweak-operator-topology f a F⟩
  assumes ⟨limitin cweak-operator-topology g b F⟩
  shows ⟨limitin cweak-operator-topology (λx. f x + g x) (a + b) F⟩
proof -
  have ⟨LIM x F. (f x, g x) :> nhdsin cweak-operator-topology a ×F nhdsin cweak-operator-topology
b)⟩

```

```

apply (rule filterlim-Pair)
using assms by (simp-all add: filterlim-cweak-operator-topology)
then have  $\langle \text{LIM } x \ F. \text{ case } (f \ x, \ g \ x) \text{ of } (x, \ y) \Rightarrow x + y \text{ :> nhdsin cweak-operator-topology } (a + b) \rangle$ 
apply (rule filterlim-compose[rotated])
by (rule cweak-operator-topology-plus-cont)
then show ?thesis
by (simp add: filterlim-cweak-operator-topology)
qed

```

lemma *monotone-convergence-wot*:

— [1], Proposition 43.1 (i), (ii), but translated to filters.

fixes $f :: \langle 'b \Rightarrow ('a \Rightarrow_{CL} 'a::\text{chilbert-space}) \rangle$

assumes *bounded*: $\langle \forall_F \ x \ \text{in } F. \ f \ x \leq B \rangle$

assumes *increasing*: $\langle \text{increasing-filter } (\text{filtermap } f \ F) \rangle$

shows $\langle \exists L. \ \text{limitin cweak-operator-topology } f \ L \ F \rangle$

proof —

wlog *nontrivial*: $\langle F \neq \perp \rangle$

using *negation* **by** (*auto intro!*: *limitin-trivial*)

wlog *pos*: $\langle \forall_F \ x \ \text{in } F. \ f \ x \geq 0 \rangle$ **generalizing** $f \ B$ **keeping** *bounded increasing nontrivial*

proof —

from *increasing*

have $\langle \forall_F \ y \ \text{in } F. \ \exists L. \ \text{limitin cweak-operator-topology } f \ L \ F \rangle$

unfolding *increasing-filter-def eventually-filtermap*

proof (*rule eventually-mono*)

fix $x0$

assume *f-lower*: $\langle \forall_F \ x \ \text{in } F. \ f \ x0 \leq f \ x \rangle$

define g **where** $\langle g \ x = f \ x - f \ x0 \rangle$ **for** x

from *bounded*

have *bounded-g*: $\langle \forall_F \ x \ \text{in } F. \ g \ x \leq B - f \ x0 \rangle$

apply (*rule eventually-mono*)

by (*simp add: g-def*)

from *f-lower*

have *pos-g*: $\langle \forall_F \ x \ \text{in } F. \ g \ x \geq 0 \rangle$

apply (*rule eventually-mono*)

by (*simp add: g-def*)

from *increasing*

have *increasing-g*: $\langle \text{increasing-filter } (\text{filtermap } g \ F) \rangle$

unfolding *increasing-filter-def eventually-filtermap*

apply (*rule eventually-mono*)

apply (*erule eventually-mono*)

by (*simp add: g-def[abs-def]*)

obtain L **where** $\langle \text{limitin cweak-operator-topology } g \ L \ F \rangle$

apply *atomize-elim*

using *pos-g bounded-g increasing-g nontrivial* **by** (*rule hypothesis*)

then have $\langle \text{limitin cweak-operator-topology } (\lambda x. \ g \ x + f \ x0) \ (L + f \ x0) \ F \rangle$

apply (*rule limitin-wot-add*)

by *simp*

then have $\langle \text{limitin cweak-operator-topology } f \ (L + f \ x0) \ F \rangle$


```

    by (auto intro!: simp: g-def[abs-def])
  then show  $\langle \exists L. \text{limitin cweak-operator-topology } f \ L \ F \rangle$ 
    by auto
qed
then show ?thesis
  by (simp add: nontrivial eventually-const)
qed

define surround where  $\langle \text{surround } \psi \ a = \psi \cdot_C (a *_{\mathcal{V}} \psi) \rangle$  for  $\psi :: 'a$  and  $a$ 
have mono-surround:  $\langle \text{mono } (\text{surround } \psi) \rangle$  for  $\psi$ 
  by (auto intro!: monoI simp: surround-def less-eq-cblinfun-def)
obtain  $l'$  where tendsto-l':  $\langle ((\lambda x. \text{surround } \psi (f x)) \longrightarrow l' \psi) \ F \rangle$  for  $\psi$ 
proof (atomize-elim, intro choice allI)
  fix  $\psi :: 'a$ 
  from bounded
  have surround-bound:  $\langle \forall_F x \text{ in } F. \text{surround } \psi (f x) \leq \text{surround } \psi \ B \rangle$ 
  unfolding surround-def
  apply (rule eventually-mono)
  by (simp add: less-eq-cblinfun-def)
moreover have  $\langle \text{increasing-filter } (\text{filtermap } (\lambda x. \text{surround } \psi (f x)) \ F) \rangle$ 
  using increasing-filtermap[OF increasing mono-surround]
  by (simp add: filtermap-filtermap)
ultimately obtain  $l'$  where  $\langle ((\lambda x. \text{surround } \psi (f x)) \longrightarrow l') \ F \rangle$ 
  apply atomize-elim
  by (auto intro!: monotone-convergence-complex increasing mono-surround
      simp: eventually-filtermap)
then show  $\langle \exists l'. ((\lambda x. \text{surround } \psi (f x)) \longrightarrow l') \ F \rangle$ 
  by auto
qed
define l where  $\langle l \ \varphi \ \psi = (l' (\varphi + \psi) - l' (\varphi - \psi) - i * l' (\varphi + i *_{\mathcal{C}} \psi) + i * l' (\varphi - i *_{\mathcal{C}} \psi)) / 4 \rangle$ 
for  $\varphi \ \psi :: 'a$ 
have polar:  $\langle \varphi \cdot_C a \ \psi = (\text{surround } (\varphi + \psi) \ a - \text{surround } (\varphi - \psi) \ a - i * \text{surround } (\varphi + i *_{\mathcal{C}} \psi) \ a + i * \text{surround } (\varphi - i *_{\mathcal{C}} \psi) \ a) / 4 \rangle$ 
for  $a :: \langle 'a \Rightarrow_{\mathcal{C}L} 'a \rangle$  and  $\varphi \ \psi$ 
  by (simp add: surround-def cblinfun.add-right cinner-add cblinfun.diff-right
      cinner-diff cblinfun.scaleC-right ring-distrib)
have tendsto-l:  $\langle ((\lambda x. \varphi \cdot_C f x \ \psi) \longrightarrow l \ \varphi \ \psi) \ F \rangle$  for  $\varphi \ \psi$ 
  by (auto intro!: tendsto-divide tendsto-add tendsto-diff tendsto-l' simp: l-def polar)
have l-bound:  $\langle \text{norm } (l \ \varphi \ \psi) \leq \text{norm } B * \text{norm } \varphi * \text{norm } \psi \rangle$  for  $\varphi \ \psi$ 
proof -
  from bounded pos
  have  $\langle \forall_F x \text{ in } F. \text{norm } (\varphi \cdot_C f x \ \psi) \leq \text{norm } B * \text{norm } \varphi * \text{norm } \psi \rangle$  for  $\varphi \ \psi$ 
  proof (rule eventually-elim2)
    fix  $x$ 
    assume  $\langle f x \leq B \rangle$  and  $\langle 0 \leq f x \rangle$ 
    have  $\langle \text{cmod } (\varphi \cdot_C (f x *_{\mathcal{V}} \psi)) \leq \text{norm } \varphi * \text{norm } (f x *_{\mathcal{V}} \psi) \rangle$ 
      using complex-inner-class.Cauchy-Schwarz-ineq2 by blast
    also have  $\langle \dots \leq \text{norm } \varphi * (\text{norm } (f x) * \text{norm } \psi) \rangle$ 
      by (simp add: mult-left-mono norm-cblinfun)
    also from  $\langle f x \leq B \rangle \langle 0 \leq f x \rangle$ 

```

```

have ⟨... ≤ norm φ * (norm B * norm ψ)⟩
  by (auto intro!: mult-left-mono mult-right-mono norm-cblinfun-mono simp: )
also have ⟨... = norm B * norm φ * norm ψ⟩
  by simp
finally show ⟨norm (φ •C f x ψ) ≤ norm B * norm φ * norm ψ⟩
  by -
qed
moreover from tendsto-l
have ⟨((λx. norm (φ •C f x ψ)) → norm (l φ ψ)) F⟩ for φ ψ
  using tendsto-norm by blast
ultimately show ?thesis
  using nontrivial tendsto-upperbound by blast
qed
have ⟨bounded-sesquilinear l⟩
proof (rule bounded-sesquilinear.intro)
fix φ φ' ψ ψ' and r :: complex
from tendsto-l have ⟨((λx. φ •C f x ψ + φ •C f x ψ') → l φ ψ + l φ ψ') F⟩
  by (simp add: tendsto-add)
moreover from tendsto-l have ⟨((λx. φ •C f x ψ + φ •C f x ψ') → l φ (ψ + ψ')) F⟩
  by (simp flip: cinner-add-right cblinfun.add-right)
ultimately show ⟨l φ (ψ + ψ') = l φ ψ + l φ ψ'⟩
  by (rule tendsto-unique[OF nontrivial, rotated])
from tendsto-l have ⟨((λx. φ •C f x ψ + φ' •C f x ψ) → l φ ψ + l φ' ψ) F⟩
  by (simp add: tendsto-add)
moreover from tendsto-l have ⟨((λx. φ •C f x ψ + φ' •C f x ψ) → l (φ + φ') ψ) F⟩
  by (simp flip: cinner-add-left cblinfun.add-left)
ultimately show ⟨l (φ + φ') ψ = l φ ψ + l φ' ψ⟩
  by (rule tendsto-unique[OF nontrivial, rotated])
from tendsto-l have ⟨((λx. r *C (φ •C f x ψ)) → r *C l φ ψ) F⟩
  using tendsto-scaleC by blast
moreover from tendsto-l have ⟨((λx. r *C (φ •C f x ψ)) → l φ (r *C ψ)) F⟩
  by (simp flip: cinner-scaleC-right cblinfun.scaleC-right)
ultimately show ⟨l φ (r *C ψ) = r *C l φ ψ⟩
  by (rule tendsto-unique[OF nontrivial, rotated])
from tendsto-l have ⟨((λx. cnj r *C (φ •C f x ψ)) → cnj r *C l φ ψ) F⟩
  using tendsto-scaleC by blast
moreover from tendsto-l have ⟨((λx. cnj r *C (φ •C f x ψ)) → l (r *C φ) ψ) F⟩
  by (simp flip: cinner-scaleC-left cblinfun.scaleC-left)
ultimately show ⟨l (r *C φ) ψ = cnj r *C l φ ψ⟩
  by (rule tendsto-unique[OF nontrivial, rotated])
show ⟨∃ K. ∀ a b. cmod (l a b) ≤ norm a * norm b * K⟩
  using l-bound by (auto intro!: exI[of - ⟨norm B⟩] simp: mult-ac)
qed
define L where ⟨L = (the-riesz-rep-sesqui l)*⟩
then have ⟨φ •C L ψ = l φ ψ⟩ for φ ψ
  by (auto intro!: ⟨bounded-sesquilinear l⟩ the-riesz-rep-sesqui-apply simp: cinner-adj-right)
with tendsto-l have ⟨((λx. φ •C f x ψ) → φ •C L ψ) F⟩ for φ ψ
  by simp
then have ⟨limitin cweak-operator-topology f L F⟩

```

by (simp add: limitin-cweak-operator-topology)
 then show ?thesis
 by auto
 qed

lemma *summable-wot-boundedI*:

fixes $f :: \langle 'b \Rightarrow ('a \Rightarrow_{CL} 'a::\text{chilbert-space}) \rangle$
 assumes *bounded*: $\langle \bigwedge F. \text{finite } F \Longrightarrow F \subseteq X \Longrightarrow \text{sum } f F \leq B \rangle$
 assumes *pos*: $\langle \bigwedge x. x \in X \Longrightarrow f x \geq 0 \rangle$
 shows $\langle \text{summable-on-in cweak-operator-topology } f X \rangle$

proof –

from *pos* **have** *incr*: $\langle \text{increasing-filter } (\text{filtermap } (\text{sum } f)) (\text{finite-subsets-at-top } X) \rangle$
 by (auto intro!: *increasing-filtermap*[**where** $X = \langle \{F. \text{finite } F \wedge F \subseteq X\} \rangle$] *mono-onI sum-mono2*)

show ?thesis

apply (simp add: *summable-on-in-def has-sum-in-def*)

by (*safe intro!*: *bounded incr monotone-convergence-wot*[**where** $B=B$] *eventually-finite-subsets-at-top-weakI*)

qed

lemma *summable-wot-boundedI'*:

fixes $f :: \langle 'b \Rightarrow ('a::\text{chilbert-space}, 'a) \text{ cblinfun-wot} \rangle$
 assumes *bounded*: $\langle \bigwedge F. \text{finite } F \Longrightarrow F \subseteq X \Longrightarrow \text{sum } f F \leq B \rangle$
 assumes *pos*: $\langle \bigwedge x. x \in X \Longrightarrow f x \geq 0 \rangle$
 shows $\langle f \text{ summable-on } X \rangle$
 apply (*subst summable-on-euclidean-eq*[*symmetric*])
 using *assms*
 apply (*transfer' fixing*: X)
 apply (*rule summable-wot-boundedI*)
 by auto

lemma *has-sum-mono-neutral-wot*:

fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
 assumes $\langle \text{has-sum-in cweak-operator-topology } f A a \rangle$ **and** *has-sum-in cweak-operator-topology*
 $g B b$
 assumes $\langle \bigwedge x. x \in A \cap B \Longrightarrow f x \leq g x \rangle$
 assumes $\langle \bigwedge x. x \in A - B \Longrightarrow f x \leq 0 \rangle$
 assumes $\langle \bigwedge x. x \in B - A \Longrightarrow g x \geq 0 \rangle$
 shows $a \leq b$

proof –

have *ψ-eq*: $\langle \psi \cdot_C a \psi \leq \psi \cdot_C b \psi \rangle$ **for** ψ

proof –

from *assms*(1)

have *sumA*: $\langle ((\lambda x. \psi \cdot_C f x \psi) \text{ has-sum } \psi \cdot_C a \psi) A \rangle$

by (*simp add*: *has-sum-in-def has-sum-def limitin-cweak-operator-topology*
cblinfun.sum-left cinner-sum-right)

from *assms*(2)

have *sumB*: $\langle ((\lambda x. \psi \cdot_C g x \psi) \text{ has-sum } \psi \cdot_C b \psi) B \rangle$

by (*simp add*: *has-sum-in-def has-sum-def limitin-cweak-operator-topology*)

```

      cblinfun.sum-left cinner-sum-right)
from sumA sumB
show ?thesis
  apply (rule has-sum-mono-neutral-complex)
  using assms(3-5)
  by (auto simp: less-eq-cblinfun-def)
qed
then show <a ≤ b>
  by (simp add: less-eq-cblinfun-def)
qed

```

lemma *has-sum-mono-wot*:

```

  fixes f :: 'a ⇒ ('b::chilbert-space ⇒CL 'b)
  assumes has-sum-in cweak-operator-topology f A x and has-sum-in cweak-operator-topology g
  A y
  assumes <∧x. x ∈ A ⇒ f x ≤ g x>
  shows x ≤ y
  using assms has-sum-mono-neutral-wot by force

```

lemma *infsun-mono-neutral-wot*:

```

  fixes f :: 'a ⇒ ('b::chilbert-space ⇒CL 'b)
  assumes summable-on-in cweak-operator-topology f A and summable-on-in cweak-operator-topology
  g B
  assumes <∧x. x ∈ A ∩ B ⇒ f x ≤ g x>
  assumes <∧x. x ∈ A - B ⇒ f x ≤ 0>
  assumes <∧x. x ∈ B - A ⇒ g x ≥ 0>
  shows infsun-in cweak-operator-topology f A ≤ infsun-in cweak-operator-topology g B
  using assms
  by (metis (mono-tags, lifting) has-sum-in-infsun-in has-sum-mono-neutral-wot hausdorff-cweak-operator-topology)

```

lemma *has-sum-on-wot-transfer[transfer-rule]*:

```

  includes lifting-syntax
  shows <(((=) ==> cr-cblinfun-wot) ==> (=) ==> cr-cblinfun-wot ==> (←→))
  (has-sum-in cweak-operator-topology) HAS-SUM>
  unfolding has-sum-euclidean-iff[abs-def, symmetric] has-sum-in-def[abs-def]
  by transfer-prover

```

lemma *summable-on-wot-transfer[transfer-rule]*:

```

  includes lifting-syntax
  shows <(((=) ==> cr-cblinfun-wot) ==> (=) ==> (←→)) (summable-on-in cweak-operator-topology)
  (summable-on)>
  apply (auto intro!: simp: summable-on-def[abs-def] summable-on-in-def[abs-def])
  by transfer-prover

```

lemma *Abs-cblinfun-wot-transfer[transfer-rule]*:

includes *lifting-syntax*
shows $\langle (=) == => cr\text{-}cblinfun\text{-}wot \rangle$ *id Abs-cblinfun-wot*
by (*auto intro!*: *rel-funI simp: cr-cblinfun-wot-def Abs-cblinfun-wot-inverse*)

lemma *infsun-mono-neutral-wot'*:
fixes $f :: 'a \Rightarrow ('b::\text{hilbert-space}, 'b) \text{cblinfun-wot}$
assumes f *summable-on A* **and** g *summable-on B*
assumes $\langle \bigwedge x. x \in A \cap B \implies f x \leq g x \rangle$
assumes $\langle \bigwedge x. x \in A - B \implies f x \leq 0 \rangle$
assumes $\langle \bigwedge x. x \in B - A \implies g x \geq 0 \rangle$
shows $\text{infsun } f A \leq \text{infsun } g B$
unfolding *infsun-euclidean-eq[symmetric]*
using *assms*
apply (*transfer' fixing: A B*)
apply (*rule infsun-mono-neutral-wot*)
by *auto*

lemma *infsun-nonneg-wot'*:
fixes $f :: 'a \Rightarrow ('c::\text{hilbert-space}, 'c) \text{cblinfun-wot}$
assumes $\bigwedge x. x \in M \implies 0 \leq f x$
shows $\text{infsun } f M \geq 0$
proof (*cases* $\langle f \text{ summable-on } M \rangle$)
case *True*
show *?thesis*
apply (*subst infsun-0[symmetric, OF refl]*)
apply (*rule infsun-mono-neutral-wot'[where A=M and B=M]*)
using *assms True by auto*
next
case *False*
then have $\langle \text{infsun } f M = 0 \rangle$
using *infsun-not-exists* **by** *blast*
then show *?thesis*
by *simp*
qed

lemma *summable-on-Sigma-wotI*:
fixes $f :: \langle 'a \times 'b \Rightarrow ('c::\text{hilbert-space}, 'c) \text{cblinfun-wot} \rangle$
assumes $\langle \bigwedge x y. x \in A \implies y \in B \implies f (x,y) \geq 0 \rangle$
assumes *summableA*: $\langle (\lambda x. \sum_{y \in B} f (x,y)) \text{ summable-on } A \rangle$
assumes *summableB*: $\langle \bigwedge x. x \in A \implies (\lambda y. f (x,y)) \text{ summable-on } (B x) \rangle$
shows $\langle f \text{ summable-on } \text{Sigma } A B \rangle$
proof (*rule summable-wot-boundedI'*)
show $\langle f x \geq 0 \rangle$ **if** $\langle x \in \text{Sigma } A B \rangle$ **for** x
using *assms that* **by** *blast*
show $\langle \text{sum } f F \leq (\sum_{x \in A} \sum_{y \in B} f (x,y)) \rangle$ **if** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq \text{Sigma } A B \rangle$ **for** F
proof –
define *FA* **where** $\langle FA = \text{fst } ' F \rangle$
define *FB* **where** $\langle FB x = \{y. (x,y) \in F\} \rangle$ **for** x

```

have F-FAB: ⟨F = Sigma FA FB⟩
  by (auto simp: FA-def FB-def image-iff Bex-def)
have [simp]: ⟨finite FA⟩ ⟨finite (FB x)⟩ for x
  using ⟨finite F⟩ by (auto intro!: finite-inverse-image injI simp: FA-def FB-def)
have FA-A: ⟨FA ⊆ A⟩
  using FA-def that(2) by auto
have FB-B: ⟨FB x ⊆ B x⟩ if ⟨x ∈ A⟩ for x
  using FB-def ⟨F ⊆ Sigma A B⟩ by auto
have ⟨sum f F = (∑ x∈FA. ∑ y∈FB x. f (x,y))⟩
  apply (subst sum.Sigma)
  by (auto simp: F-FAB)
also have ⟨... = (∑ x∈FA. ∑ ∞y∈FB x. f (x,y))⟩
  by fastforce
also have ⟨... ≤ (∑ x∈FA. ∑ ∞y∈B x. f (x,y))⟩
  apply (rule sum-mono)
  apply (rule infsum-mono-neutral-wot')
  using FA-A FB-B assms by auto
also have ⟨... = (∑ ∞x∈FA. ∑ ∞y∈B x. f (x,y))⟩
  by fastforce
also have ⟨... ≤ (∑ ∞x∈A. ∑ ∞y∈B x. f (x,y))⟩
  apply (rule infsum-mono-neutral-wot')
  using FA-A assms by (auto intro!: infsum-nonneg-wot')
finally show ⟨sum f F ≤ (∑ ∞x∈A. ∑ ∞y∈B x. f (x,y))⟩
  by -
qed
qed

```

lift-definition *compose-wot* :: ⟨('b::complex-inner, 'c::complex-inner) cblinfun-wot ⇒ ('a::complex-normed-vector, 'b) cblinfun-wot ⇒ ('a, 'c) cblinfun-wot⟩ **is** *cblinfun-compose*.

lift-definition *adj-wot* :: ⟨('a::hilbert-space, 'b::complex-inner) cblinfun-wot ⇒ ('b, 'a) cblinfun-wot⟩ **is** *adj*.

lemma *infsum-wot-is-Sup*:

```

fixes f :: ⟨'b ⇒ ('a ⇒CL 'a::hilbert-space)⟩
assumes summable: ⟨summable-on-in cweak-operator-topology f X⟩
  — See also summable-wot-boundedI for proving this.
assumes pos: ⟨∧x. x ∈ X ⇒ f x ≥ 0⟩
defines ⟨S ≡ infsum-in cweak-operator-topology f X⟩
shows ⟨is-Sup ((λF. ∑ x∈F. f x) ‘{F. finite F ∧ F ⊆ X}) S⟩
proof (rule is-SupI)
have has-sum: ⟨has-sum-in cweak-operator-topology f X S⟩
  unfolding S-def
  apply (rule has-sum-in-infsum-in)
  using assms by auto
show ⟨s ≤ S⟩ if ⟨s ∈ ((λF. ∑ x∈F. f x) ‘{F. finite F ∧ F ⊆ X})⟩ for s
proof -
from that obtain F where [simp]: ⟨finite F⟩ and ⟨F ⊆ X⟩ and s-def: ⟨s = (∑ x∈F. f x)⟩

```

```

    by auto
  show ?thesis
  proof (rule has-sum-mono-neutral-wot)
    show ⟨has-sum-in cweak-operator-topology f F s⟩
      by (auto intro!: has-sum-in-finite simp: s-def)
    show ⟨has-sum-in cweak-operator-topology f X S⟩
      by (fact has-sum)
    show ⟨f x ≤ f x⟩ for x
      by simp
    show ⟨f x ≤ 0⟩ if ⟨x ∈ F - X⟩ for x
      using ⟨F ⊆ X⟩ that by auto
    show ⟨f x ≥ 0⟩ if ⟨x ∈ X - F⟩ for x
      using that pos by auto
  qed
  qed
  show ⟨S ≤ y⟩
    if y-bound: ⟨ $\bigwedge x. x \in ((\lambda F. \sum x \in F. f x) \text{ ‘ } \{F. \text{finite } F \wedge F \subseteq X\}) \implies x \leq y$ ⟩ for y
  proof (rule cblinfun-leI, rename-tac  $\psi$ )
    fix  $\psi :: 'a$ 
    define g where ⟨g x =  $\psi \cdot_C \text{Rep-cblinfun-wot } x \psi$ ⟩ for x
    from has-sum have lim: ⟨ $(\lambda i. \psi \cdot_C ((\sum x \in i. f x) *_V \psi)) \longrightarrow \psi \cdot_C (S *_V \psi)$ ⟩ (finite-subsets-at-top X)
      by (simp add: has-sum-in-def limitin-cweak-operator-topology)
    have bound: ⟨ $\psi \cdot_C (\sum x \in F. f x) \psi \leq \psi \cdot_C y \psi$ ⟩ if ⟨finite F⟩ ⟨F ⊆ X⟩ for F
      using y-bound less-eq-cblinfun-def that(1) that(2) by fastforce
    show ⟨ $\psi \cdot_C (S *_V \psi) \leq \psi \cdot_C y \psi$ ⟩
      using finite-subsets-at-top-neq-bot tendsto-const lim apply (rule tendsto-le-complex)
      using bound by (auto intro!: eventually-finite-subsets-at-top-weakI)
  qed
  qed

lemma has-sum-in-cweak-operator-topology-pointwise:
  ⟨has-sum-in cweak-operator-topology f X s  $\longleftrightarrow (\forall \psi \varphi. ((\lambda x. \psi \cdot_C f x \varphi) \text{ has-sum } \psi \cdot_C s \varphi) X)$ ⟩
  by (simp add: has-sum-in-def has-sum-def limitin-cweak-operator-topology
    cblinfun.sum-left cinner-sum-right)

lemma summable-wot-bdd-above:
  fixes f :: ⟨'b  $\Rightarrow$  ('a  $\Rightarrow_{CL}$  'a::chilbert-space)⟩
  assumes summable: ⟨summable-on-in cweak-operator-topology f X⟩
    — See also summable-wot-boundedI for proving this.
  assumes pos: ⟨ $\bigwedge x. x \in X \implies f x \geq 0$ ⟩
  shows ⟨bdd-above (sum f ‘ {F. finite F  $\wedge$  F ⊆ X})⟩
  using infsum-wot-is-Sup[OF assms]
  by (auto intro!: simp: is-Sup-def bdd-above-def)

lemma summable-on-in-cweak-operator-topology-pointwise:
  assumes ⟨summable-on-in cweak-operator-topology f X⟩
  shows ⟨ $(\lambda x. a \cdot_C f x b)$  summable-on X⟩

```

using *assms*
by (*auto simp: summable-on-in-def summable-on-def has-sum-in-cweak-operator-topology-pointwise*)

lemma *infsun-in-cweak-operator-topology-pointwise*:
assumes $\langle \text{summable-on-in cweak-operator-topology } f \ X \rangle$
shows $\langle a \cdot_C (\text{infsun-in cweak-operator-topology } f \ X) \ b = (\sum_{\infty x \in X}. a \cdot_C f \ x \ b) \rangle$
by (*metis (mono-tags, lifting) assms has-sum-in-cweak-operator-topology-pointwise has-sum-in-infsun-in hausdorff-cweak-operator-topology infsunI*)

instance *cblinfun-wot* :: (*complex-normed-vector, complex-inner*) *topological-ab-group-add*
by *intro-classes*

lemma *has-sum-in-wot-compose-left*:
fixes $f :: \langle 'c \Rightarrow 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
assumes $\langle \text{has-sum-in cweak-operator-topology } f \ X \ s \rangle$
shows $\langle \text{has-sum-in cweak-operator-topology } (\lambda x. a \ o_{CL} \ f \ x) \ X \ (a \ o_{CL} \ s) \rangle$
proof (*rule has-sum-in-cweak-operator-topology-pointwise[THEN iffD2], intro allI, rename-tac g h*)
fix $g \ h$
from *assms* **have** $\langle ((\lambda x. (a*) \ g \cdot_C \ f \ x \ h) \ \text{has-sum } (a*) \ g \cdot_C \ s \ h) \ X) \rangle$
by (*metis has-sum-in-cweak-operator-topology-pointwise*)
then show $\langle ((\lambda x. g \cdot_C \ (a \ o_{CL} \ f \ x) \ h) \ \text{has-sum } g \cdot_C \ (a \ o_{CL} \ s) \ h) \ X \rangle$
by (*metis (no-types, lifting) cblinfun-apply-cblinfun-compose cinner-adj-left has-sum-cong*)
qed

lemma *has-sum-in-wot-compose-right*:
fixes $f :: \langle 'c \Rightarrow 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-inner} \rangle$
assumes $\langle \text{has-sum-in cweak-operator-topology } f \ X \ s \rangle$
shows $\langle \text{has-sum-in cweak-operator-topology } (\lambda x. f \ x \ o_{CL} \ a) \ X \ (s \ o_{CL} \ a) \rangle$
proof (*rule has-sum-in-cweak-operator-topology-pointwise[THEN iffD2], intro allI, rename-tac g h*)
fix $g \ h$
from *assms* **have** $\langle ((\lambda x. g \cdot_C \ f \ x \ (a \ h)) \ \text{has-sum } g \cdot_C \ s \ (a \ h)) \ X \rangle$
by (*metis has-sum-in-cweak-operator-topology-pointwise*)
then show $\langle ((\lambda x. g \cdot_C \ (f \ x \ o_{CL} \ a) \ h) \ \text{has-sum } g \cdot_C \ (s \ o_{CL} \ a) \ h) \ X \rangle$
by *simp*
qed

lemma *summable-on-in-wot-compose-left*:
fixes $f :: \langle 'c \Rightarrow 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
assumes $\langle \text{summable-on-in cweak-operator-topology } f \ X \rangle$
shows $\langle \text{summable-on-in cweak-operator-topology } (\lambda x. a \ o_{CL} \ f \ x) \ X \rangle$
using *has-sum-in-wot-compose-left assms*
by (*fastforce simp: summable-on-in-def*)

lemma *summable-on-in-wot-compose-right*:
assumes $\langle \text{summable-on-in cweak-operator-topology } f \ X \rangle$

shows $\langle \text{summable-on-in cweak-operator-topology } (\lambda x. f x \text{ } o_{CL} a) X \rangle$
using *has-sum-in-wot-compose-right assms*
by (*fastforce simp: summable-on-in-def*)

lemma *infsun-in-wot-compose-left:*

fixes $f :: \langle 'c \Rightarrow 'a :: \text{complex-normed-vector} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$
assumes $\langle \text{summable-on-in cweak-operator-topology } f X \rangle$
shows $\langle \text{infsun-in cweak-operator-topology } (\lambda x. a \text{ } o_{CL} f x) X = a \text{ } o_{CL} (\text{infsun-in cweak-operator-topology } f X) \rangle$
by (*metis (mono-tags, lifting) assms has-sum-in-infsun-in has-sum-in-unique hausdorff-cweak-operator-topology has-sum-in-wot-compose-left summable-on-in-wot-compose-left*)

lemma *infsun-in-wot-compose-right:*

fixes $f :: \langle 'c \Rightarrow 'a :: \text{complex-normed-vector} \Rightarrow_{CL} 'b :: \text{complex-inner} \rangle$
assumes $\langle \text{summable-on-in cweak-operator-topology } f X \rangle$
shows $\langle \text{infsun-in cweak-operator-topology } (\lambda x. f x \text{ } o_{CL} a) X = (\text{infsun-in cweak-operator-topology } f X) \text{ } o_{CL} a \rangle$
by (*metis (mono-tags, lifting) assms has-sum-in-infsun-in has-sum-in-unique hausdorff-cweak-operator-topology has-sum-in-wot-compose-right summable-on-in-wot-compose-right*)

lemma *infsun-wot-boundedI:*

fixes $f :: \langle 'b \Rightarrow ('a \Rightarrow_{CL} 'a :: \text{chilbert-space}) \rangle$
assumes *bounded*: $\langle \bigwedge F. \text{finite } F \Longrightarrow F \subseteq X \Longrightarrow \text{sum } f F \leq B \rangle$
assumes *pos*: $\langle \bigwedge x. x \in X \Longrightarrow f x \geq 0 \rangle$
shows $\langle \text{infsun-in cweak-operator-topology } f X \leq B \rangle$

proof (*rule less-eq-cblinfunI*)

fix h

have *summ*: $\langle \text{summable-on-in cweak-operator-topology } f X \rangle$

using *assms* **by** (*rule summable-wot-boundedI*)

then have $\langle h \cdot_C (\text{infsun-in cweak-operator-topology } f X *_V h) = (\sum_{\infty x \in X. h \cdot_C (f x *_V h)) \rangle$

by (*rule infsun-in-cweak-operator-topology-pointwise*)

also have $\langle \dots \leq h \cdot_C B h \rangle$

proof (*rule less-eq-complexI*)

from *summ* **have** *summ'*: $\langle (\lambda x. h \cdot_C (f x *_V h)) \text{ summable-on } X \rangle$

by (*auto intro!: summable-on-in-cweak-operator-topology-pointwise*)

have $*$: $\langle (\sum_{x \in F. h \cdot_C (f x *_V h)) \leq h \cdot_C B h \rangle$ **if** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq X \rangle$ **for** F

using *that bounded*

by (*simp add: less-eq-cblinfun-def flip: cinner-sum-right cblinfun.sum-left*)

show $\langle \text{Im } (\sum_{\infty x \in X. h \cdot_C (f x *_V h)) = \text{Im } (h \cdot_C (B *_V h)) \rangle$

proof –

from $*$ [*of* $\langle \{ \} \rangle$] **have** $\langle h \cdot_C B h \geq 0 \rangle$

by *simp*

then have $\langle \text{Im } (h \cdot_C B h) = 0 \rangle$

using *comp-Im-same zero-complex.sel(2)* **by** *presburger*

moreover then have $\langle \text{Im } (h \cdot_C (f x *_V h)) = 0 \rangle$ **if** $\langle x \in X \rangle$ **for** x

using $*$ [*of* $\langle \{ x \} \rangle$] **that**

by (*simp add: less-eq-complex-def*)

```

    ultimately show ⟨Im (∑∞ x∈X. h ·C (f x *V h)) = Im (h ·C (B *V h))⟩
      by (auto intro!: infsum-0 simp: summ' simp flip: infsum-Im)
  qed
  show ⟨Re (∑∞ x∈X. h ·C (f x *V h)) ≤ Re (h ·C (B *V h))⟩
  apply (auto intro!: summable-on-Re infsum-le-finite-sums simp: summ' simp flip: infsum-Re)
  using summ'
  by (metis * Re-sum less-eq-complex-def)
  qed
  finally show ⟨h ·C (infsum-in cweak-operator-topology f X *V h) ≤ h ·C (B *V h)⟩
  by -
  qed
  qed

  unbundle no cblinfun-syntax

end

```

7 Misc-Tensor-Product-TTS – Miscellaneous results missing from Complex_Bounded_Operators

Here specifically results obtained from lifting existing results using the types to sets mechanism ([6]).

```

theory Misc-Tensor-Product-TTS
  imports
    Complex-Bounded-Operators.Complex-Bounded-Linear-Function
    Misc-Tensor-Product
    Misc-Tensor-Product-BO
    With-Type.With-Type
begin

  unbundle lattice-syntax and cblinfun-syntax

```

7.1 Retrieving axioms

```

attribute-setup axiom = ⟨Scan.lift Parse.name-position >> (fn name-pos => Thm.rule-attribute
  []
  (fn context => fn - =>
    let val thy = Context.theory-of context
      val (full-name, -) = Name-Space.check context (Theory.axiom-table thy) name-pos
      in Thm.axiom thy full-name end))
  ⟨Retrieve an axiom by name. E.g., write @{thm [source] [[axiom HOL.refl]]}⟩

```

7.2 Auxiliary lemmas

```

named-theorems unoverload-def

```

```

locale local-typedef = fixes S ::'b set and s::'s itself
  assumes Ex-type-definition-S: ∃(Rep::'s ⇒ 'b) (Abs::'b ⇒ 's). type-definition Rep Abs S

```

```

begin
definition Rep = fst (SOME (Rep::'s ⇒ 'b, Abs). type-definition Rep Abs S)
definition Abs = snd (SOME (Rep::'s ⇒ 'b, Abs). type-definition Rep Abs S)
lemma type-definition-S: type-definition Rep Abs S
  unfolding Abs-def Rep-def split-beta'
  by (rule someI-ex) (use Ex-type-definition-S in auto)
lemma rep-in-S[simp]: Rep x ∈ S
  and rep-inverse[simp]: Abs (Rep x) = x
  and Abs-inverse[simp]: y ∈ S ⇒ Rep (Abs y) = y
  using type-definition-S
  unfolding type-definition-def by auto
definition cr-S where cr-S ≡ λs b. s = Rep b
lemma Domainp-cr-S[transfer-domain-rule]: Domainp cr-S = (λx. x ∈ S)
  by (metis Abs-inverse Domainp.simps cr-S-def rep-in-S)
lemma right-total-cr-S[transfer-rule]: right-total cr-S
  by (rule typedef-right-total[OF type-definition-S cr-S-def])
lemma bi-unique-cr-S[transfer-rule]: bi-unique cr-S
  by (rule typedef-bi-unique[OF type-definition-S cr-S-def])
lemma left-unique-cr-S[transfer-rule]: left-unique cr-S
  by (rule typedef-left-unique[OF type-definition-S cr-S-def])
lemma right-unique-cr-S[transfer-rule]: right-unique cr-S
  by (rule typedef-right-unique[OF type-definition-S cr-S-def])
lemma cr-S-Rep[intro, simp]: cr-S (Rep a) a by (simp add: cr-S-def)
lemma cr-S-Abs[intro, simp]: a ∈ S ⇒ cr-S a (Abs a) by (simp add: cr-S-def)
lemma UNIV-transfer[transfer-rule]: ⟨rel-set cr-S S UNIV⟩
  using Domainp-cr-S right-total-cr-S right-total-UNIV-transfer by fastforce
end

lemma complete-space-as-set[simp]: ⟨complete (space-as-set V)⟩ for V :: ⟨-::cbanach ccspace⟩
  by (simp add: complete-eq-closed)

definition ⟨transfer-ball-range A P ⟷ (∀f. range f ⊆ A ⟶ P f)⟩

lemma transfer-ball-range-parametric'[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule, simp]: ⟨right-unique T⟩ ⟨bi-total T⟩ ⟨bi-unique U⟩
  shows ⟨rel-set U ===> ((T ===> U) ===> (⟶)) ===> (⟶)⟩ transfer-ball-range
transfer-ball-range
proof (intro rel-funI impI, rename-tac A B P Q)
  fix A B P Q
  assume [transfer-rule]: ⟨rel-set U A B⟩
  assume TUPQ[transfer-rule]: ⟨((T ===> U) ===> (⟶)) P Q⟩
  assume ⟨transfer-ball-range A P⟩
  then have Pf: ⟨P f⟩ if ⟨range f ⊆ A⟩ for f
    unfolding transfer-ball-range-def using that by auto
  have ⟨Q g⟩ if ⟨range g ⊆ B⟩ for g
  proof -
    from that ⟨rel-set U A B⟩
    have ⟨Rangep (T ===> U) g⟩

```

```

apply (auto simp add: conversep-rel-fun Domainp-pred-fun-eq simp flip: Domainp-conversep)
apply (simp add: Domainp-conversep)
by (metis Rangep.simps range-subsetD rel-setD2)
then obtain f where TUFg[transfer-rule]:  $\langle (T \implies U) f g \rangle$ 
by blast
from that have  $\langle \text{range } f \subseteq A \rangle$ 
by transfer
then have  $\langle P f \rangle$ 
by (simp add: Pf)
then show  $\langle Q g \rangle$ 
by (metis TUFg TUPQ rel-funD)
qed
then show  $\langle \text{transfer-ball-range } B Q \rangle$ 
by (simp add: transfer-ball-range-def)
qed

```

```

lemma transfer-ball-range-parametric[transfer-rule]:
includes lifting-syntax
assumes [transfer-rule, simp]:  $\langle \text{bi-unique } T \rangle \langle \text{bi-total } T \rangle \langle \text{bi-unique } U \rangle$ 
shows  $\langle (\text{rel-set } U \implies ((T \implies U) \implies (\longleftrightarrow)) \implies (\longleftrightarrow)) \text{ transfer-ball-range} \text{ transfer-ball-range} \rangle$ 
proof –
have  $\langle (\text{rel-set } U \implies ((T \implies U) \implies (\longrightarrow)) \implies (\longrightarrow)) \text{ transfer-ball-range} \text{ transfer-ball-range} \rangle$ 
using assms(1) assms(2) assms(3) bi-unique-alt-def transfer-ball-range-parametric' by blast
then have 1:  $\langle (\text{rel-set } U \implies ((T \implies U) \implies (\longleftrightarrow)) \implies (\longrightarrow)) \text{ transfer-ball-range} \text{ transfer-ball-range} \rangle$ 
apply (rule rev-mp)
apply (intro rel-fun-mono')
by auto

have  $\langle (\text{rel-set } (U^{-1-1}) \implies ((T^{-1-1} \implies U^{-1-1}) \implies (\longrightarrow)) \implies (\longrightarrow)) \text{ transfer-ball-range} \text{ transfer-ball-range} \rangle$ 
apply (rule transfer-ball-range-parametric')
using assms(1) bi-unique-alt-def bi-unique-conversep apply blast
by auto
then have  $\langle (\text{rel-set } U \implies ((T \implies U) \implies (\longrightarrow)^{-1-1}) \implies (\longrightarrow)^{-1-1}) \text{ transfer-ball-range} \text{ transfer-ball-range} \rangle$ 
apply (rule-tac conversepD[where  $r = \langle (\text{rel-set } U \implies ((T \implies U) \implies (\longrightarrow)^{-1-1}) \implies (\longrightarrow)^{-1-1}) \implies (\longrightarrow)^{-1-1} \rangle$ ])
by (simp add: conversep-rel-fun del: conversep-iff)
then have 2:  $\langle (\text{rel-set } U \implies ((T \implies U) \implies (\longleftrightarrow)) \implies (\longrightarrow)^{-1-1}) \text{ transfer-ball-range} \text{ transfer-ball-range} \rangle$ 
apply (rule rev-mp)
apply (intro rel-fun-mono')
by (auto simp: rev-implies-def)

from 1 2 show ?thesis
apply (auto intro!: rel-funI simp: conversep-iff[abs-def])

```

apply (*smt* (*z3*) *rel-funE*)
by (*smt* (*verit*) *rel-funE* *rev-implies-def*)
qed

definition $\langle \text{transfer-Times } A \ B = A \times B \rangle$

lemma *transfer-Times-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set } T \text{ ===} \rangle \text{ rel-set } U \text{ ===} \rangle \text{ rel-set } (\text{rel-prod } T \ U) \rangle \text{ transfer-Times transfer-Times}$
by (*auto intro!*: *rel-funI simp add: transfer-Times-def rel-set-def*)

lemma *csubspace-nonempty*: $\langle \text{csubspace } X \implies X \neq \{\} \rangle$
using *complex-vector.subspace-0* **by** *auto*

definition $\langle \text{transfer-vimage-into } f \ U \ s = (f \text{ -' } U) \cap s \rangle$

lemma *transfer-vimage-into-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle \langle \text{bi-unique } B \rangle$
shows $\langle ((A \text{ ===} \rangle B) \text{ ===} \rangle \text{ rel-set } B \text{ ===} \rangle \text{ rel-set } A \text{ ===} \rangle \text{ rel-set } A \rangle \text{ transfer-vimage-into transfer-vimage-into}$
unfolding *transfer-vimage-into-def*
apply (*auto intro!*: *rel-funI simp: rel-set-def*)
by (*metis Int-iff apply-rsp' assms bi-unique-def vimage-eq*) $+$

lemma *make-parametricity-proof-friendly*:
shows $\langle (\forall x. P \longrightarrow Q \ x) \longleftrightarrow (P \longrightarrow (\forall x. Q \ x)) \rangle$
and $\langle (\forall x. x \in S \longrightarrow Q \ x) \longleftrightarrow (\forall x \in S. Q \ x) \rangle$
and $\langle (\forall x \subseteq S. R \ x) \longleftrightarrow (\forall x \in \text{Pow } S. R \ x) \rangle$
and $\langle \{x \in S. Q \ x\} = \text{Set.filter } Q \ S \rangle$
and $\langle \{x. x \subseteq S \wedge R \ x\} = \text{Set.filter } R \ (\text{Pow } S) \rangle$
and $\langle \bigwedge P. (\forall f. \text{range } f \subseteq A \longrightarrow P \ f) = \text{transfer-ball-range } A \ P \rangle$
and $\langle \bigwedge A \ B. A \times B = \text{transfer-Times } A \ B \rangle$
and $\langle \bigwedge B \ P. (\exists A \subseteq B. P \ A) \longleftrightarrow (\exists A \in \text{Pow } B. P \ A) \rangle$
and $\langle \bigwedge f \ U \ s. (f \text{ -' } U) \cap s = \text{transfer-vimage-into } f \ U \ s \rangle$
and $\langle \bigwedge M \ B. \bigcap M \cap \text{principal } B = \text{transfer-bounded-filter-Inf } B \ M \rangle$
and $\langle \bigwedge F \ M. F \cap \text{principal } M = \text{transfer-inf-principal } F \ M \rangle$
by (*auto simp: transfer-ball-range-def transfer-Times-def transfer-vimage-into-def transfer-bounded-filter-Inf-def transfer-inf-principal-def*)

7.3 *plus*

locale *plus-ow* =
fixes *U plus*
assumes $\langle \forall x \in U. \forall y \in U. \text{plus } x \ y \in U \rangle$

lemma *plus-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ ==== } (A \text{ ==== } A \text{ ==== } A) \text{ ==== } (=))$
plus-ow plus-ow
unfolding *plus-ow-def*
by *transfer-prover*

7.3.1 *minus*

locale *minus-ow* = **fixes** *U minus* **assumes** $\langle \forall x \in U. \forall y \in U. \text{minus } x \ y \in U \rangle$

lemma *minus-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ ==== } (A \text{ ==== } A \text{ ==== } A) \text{ ==== } (=))$
minus-ow minus-ow
unfolding *minus-ow-def*
by *transfer-prover*

7.3.2 *uminus*

locale *uminus-ow* = **fixes** *U uminus* **assumes** $\langle \forall x \in U. \text{uminus } x \in U \rangle$

lemma *uminus-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ ==== } (A \text{ ==== } A) \text{ ==== } (=))$
uminus-ow uminus-ow
unfolding *uminus-ow-def*
by *transfer-prover*

7.4 *semigroup*

locale *semigroup-ow* = *plus-ow U plus* **for** *U plus +*
assumes $\langle \forall x \in U. \forall y \in U. \forall z \in U. \text{plus } x \ (\text{plus } y \ z) = \text{plus } (\text{plus } x \ y) \ z \rangle$

lemma *semigroup-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ ==== } (A \text{ ==== } A \text{ ==== } A) \text{ ==== } (=))$
semigroup-ow semigroup-ow
unfolding *semigroup-ow-def semigroup-ow-axioms-def*
by *transfer-prover*

lemma *semigroup-ow-typeclass*[*simp, iff*]: $\langle \text{semigroup-ow } V \ (+) \rangle$
if $\langle \wedge x \ y. x \in V \implies y \in V \implies x + y \in V \rangle$ **for** $V :: \langle 'a :: \text{semigroup-add set} \rangle$
by (*auto intro!*: *plus-ow.intro semigroup-ow.intro semigroup-ow-axioms.intro simp: Groups.add-ac that*)

lemma *class-semigroup-add-ud*[*unoverload-def*]: $\langle \text{class.semigroup-add} = \text{semigroup-ow UNIV} \rangle$
by (*auto intro!*: *ext plus-ow.intro simp*: *class.semigroup-add-def semigroup-ow-def semigroup-ow-axioms-def*)

7.5 *abel-semigroup*

locale *abel-semigroup-ow* = *semigroup-ow U plus for U plus +*
assumes $\langle \forall x \in U. \forall y \in U. \text{plus } x \ y = \text{plus } y \ x \rangle$

lemma *abel-semigroup-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ ====} \rangle (A \text{ ====} \rangle A \text{ ====} \rangle A) \text{ ====} \rangle (=) \rangle$
abel-semigroup-ow abel-semigroup-ow
unfolding *abel-semigroup-ow-def abel-semigroup-ow-axioms-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *abel-semigroup-ow-typeclass*[*simp, iff*]: $\langle \text{abel-semigroup-ow } V (+) \rangle$
if $\langle \bigwedge x \ y. x \in V \implies y \in V \implies x + y \in V \rangle$ **for** $V :: \langle 'a :: \text{ab-semigroup-add set} \rangle$
by (*auto simp*: *abel-semigroup-ow-def abel-semigroup-ow-axioms-def Groups.add-ac that*)

lemma *class-ab-semigroup-add-ud*[*unoverload-def*]: $\langle \text{class.ab-semigroup-add} = \text{abel-semigroup-ow UNIV} \rangle$
by (*auto intro!*: *ext simp*: *class.ab-semigroup-add-def abel-semigroup-ow-def class-semigroup-add-ud abel-semigroup-ow-axioms-def class.ab-semigroup-add-axioms-def*)

7.6 *comm-monoid*

locale *comm-monoid-ow* = *abel-semigroup-ow U plus for U plus +*
fixes *zero*
assumes $\langle \text{zero} \in U \rangle$
assumes $\langle \forall x \in U. \text{plus } x \ \text{zero} = x \rangle$

lemma *comm-monoid-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ ====} \rangle (A \text{ ====} \rangle A \text{ ====} \rangle A) \text{ ====} \rangle A \text{ ====} \rangle (=) \rangle$
comm-monoid-ow comm-monoid-ow
unfolding *comm-monoid-ow-def comm-monoid-ow-axioms-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *comm-monoid-ow-typeclass*[*simp, iff*]: $\langle \text{comm-monoid-ow } V (+) \ 0 \rangle$
if $\langle 0 \in V \rangle$ **and** $\langle \bigwedge x \ y. x \in V \implies y \in V \implies x + y \in V \rangle$ **for** $V :: \langle 'a :: \text{comm-monoid-add set} \rangle$
by (*auto simp*: *comm-monoid-ow-def comm-monoid-ow-axioms-def that*)

lemma *class-comm-monoid-add-ud*[*unoverload-def*]: $\langle \text{class.comm-monoid-add} = \text{comm-monoid-ow UNIV} \rangle$
apply (*auto intro!*: *ext simp*: *class.comm-monoid-add-def comm-monoid-ow-def class-ab-semigroup-add-ud class.comm-monoid-add-axioms-def comm-monoid-ow-axioms-def*)

by (simp-all add: abel-semigroup-ow-def abel-semigroup-ow-axioms-def)

7.7 topological-space

locale topological-space-ow =

fixes U open

assumes $\langle \text{open } U \rangle$

assumes $\langle \forall S \subseteq U. \forall T \subseteq U. \text{open } S \longrightarrow \text{open } T \longrightarrow \text{open } (S \cap T) \rangle$

assumes $\langle \forall K \subseteq \text{Pow } U. (\forall S \in K. \text{open } S) \longrightarrow \text{open } (\bigcup K) \rangle$

lemma topological-space-ow-parametricity[transfer-rule]:

includes lifting-syntax

assumes [transfer-rule]: $\langle \text{bi-unique } A \rangle$

shows $\langle (\text{rel-set } A \text{ =====} \rangle (\text{rel-set } A \text{ =====} \rangle (=)) \text{ =====} \rangle (=)$

$\langle \text{topological-space-ow topological-space-ow} \rangle$

unfolding topological-space-ow-def make-parametricity-proof-friendly

by transfer-prover

lemma class-topological-space-ud[unoverload-def]: $\langle \text{class.topological-space} = \text{topological-space-ow UNIV} \rangle$

by (auto intro!: ext simp: class.topological-space-def topological-space-ow-def)

lemma topological-space-ow-from-topology[simp]: $\langle \text{topological-space-ow } (\text{topspace } T) (\text{openin } T) \rangle$

by (auto intro!: topological-space-ow.intro)

7.8 sum

definition $\langle \text{sum-ow } z \text{ plus } f \text{ } S =$

$(\text{if finite } S \text{ then the-default } z \text{ (Collect (fold-graph (plus o } f) z S)) \text{ else } z) \rangle$

for $U \text{ } z \text{ plus } S$

lemma sum-ow-parametric[transfer-rule]:

includes lifting-syntax

assumes [transfer-rule]: $\langle \text{bi-unique } T \rangle \langle \text{bi-unique } U \rangle$

shows $\langle (T \text{ =====} \rangle (V \text{ =====} \rangle T \text{ =====} \rangle T) \text{ =====} \rangle (U \text{ =====} \rangle V) \text{ =====} \rangle \text{rel-set } U \text{ =====} \rangle T) \rangle$

$\langle \text{sum-ow sum-ow} \rangle$

unfolding sum-ow-def

by transfer-prover

lemma (in comm-monoid-set) comp-fun-commute-onI: $\langle \text{Finite-Set.comp-fun-commute-on UNIV } ((*) \circ g) \rangle$

apply (rule Finite-Set.comp-fun-commute-on.intro)

by (simp add: o-def left-commute)

lemma (in comm-monoid-set) F-via-the-default: $\langle F \text{ } g \text{ } A = \text{the-default def (Collect (fold-graph } ((*) \circ g) \mathbf{1} A)) \rangle$

if $\langle \text{finite } A \rangle$

proof –

have $\langle y = x \rangle$ **if** $\langle \text{fold-graph } ((*) \circ g) \mathbf{1} A x \rangle$ **and** $\langle \text{fold-graph } ((*) \circ g) \mathbf{1} A y \rangle$ **for** $x y$
using that apply (rule *Finite-Set.comp-fun-commute-on.fold-graph-determ*[rotated 2, where $S=UNIV$])
by (*simp-all add: comp-fun-commute-onI*)
then have $\langle \text{Ex1 } (\text{fold-graph } ((*) \circ g) \mathbf{1} A) \rangle$
by (*meson finite-imp-fold-graph that*)
then have $\langle \text{card } (\text{Collect } (\text{fold-graph } ((*) \circ g) \mathbf{1} A)) = 1 \rangle$
using *card-eq-Suc-0-ex1* **by** *fastforce*
then show *?thesis*
using that by (*auto simp add: the-default-The eq-fold Finite-Set.fold-def*)
qed

lemma *sum-ud*[*unoverload-def*]: $\langle \text{sum} = \text{sum-ow } 0 \text{ plus} \rangle$
apply (*auto intro!: ext simp: sum-def sum-ow-def comm-monoid-set.F-via-the-default*)
apply (*subst comm-monoid-set.F-via-the-default*)
apply (*auto simp add: sum.comm-monoid-set-axioms*)
by (*metis comm-monoid-add-class.sum-def sum.infinite*)

7.9 *t2-space*

locale *t2-space-ow* = *topological-space-ow* +
assumes $\langle \forall x \in U. \forall y \in U. x \neq y \longrightarrow (\exists S \subseteq U. \exists T \subseteq U. \text{open } S \wedge \text{open } T \wedge x \in S \wedge y \in T \wedge S \cap T = \{\}) \rangle$

lemma *t2-space-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ ===} \rangle (\text{rel-set } A \text{ ===} \rangle (=)) \text{ ===} \rangle (=) \rangle$
 $\langle \text{t2-space-ow } \text{t2-space-ow} \rangle$
unfolding *t2-space-ow-def t2-space-ow-axioms-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *class-t2-space-ud*[*unoverload-def*]: $\langle \text{class.t2-space} = \text{t2-space-ow } UNIV \rangle$
by (*auto intro!: ext simp: class.t2-space-def class.t2-space-axioms-def t2-space-ow-def t2-space-ow-axioms-def class-topological-space-ud*)

lemma *t2-space-ow-from-topology*[*simp, iff*]: $\langle \text{t2-space-ow } (\text{topspace } T) (\text{openin } T) \rangle$ **if** $\langle \text{Hausdorff-space } T \rangle$
using that
apply (*auto intro!: t2-space-ow.intro simp: t2-space-ow-axioms-def Hausdorff-space-def disjoint-def*)
by (*metis openin-subset*)

7.9.1 *continuous-on*

definition *continuous-on-ow* **where** $\langle \text{continuous-on-ow } A B \text{ opnA opnB } s f \rangle$
 $\langle \longleftrightarrow (\forall U \subseteq B. \text{opnB } U \longrightarrow (\exists V \subseteq A. \text{opnA } V \wedge (V \cap s) = (f \text{ -' } U) \cap s)) \rangle$
for $f :: \langle 'a \Rightarrow 'b \rangle$

lemma *continuous-on-ud*[*unoverload-def*]: $\langle \text{continuous-on } s \text{ } f \longleftrightarrow \text{continuous-on-ow UNIV UNIV open open } s \text{ } f \rangle$
for $f :: \langle 'a::\text{topological-space} \Rightarrow 'b::\text{topological-space} \rangle$
unfolding *continuous-on-ow-def continuous-on-open-invariant* **by** *auto*

lemma *continuous-on-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle \langle \text{bi-unique } B \rangle$
shows $\langle (\text{rel-set } A \text{ } == == > \text{rel-set } B \text{ } == == > (\text{rel-set } A \text{ } == == > (\longleftrightarrow)) \text{ } == == > (\text{rel-set } B \text{ } == == > (\longleftrightarrow)) \text{ } == == > \text{rel-set } A \text{ } == == > (A \text{ } == == > B) \text{ } == == > (\longleftrightarrow)) \text{ } \text{continuous-on-ow continuous-on-ow} \rangle$
unfolding *continuous-on-ow-def make-parametricity-proof-friendly*
by *transfer-prover*

7.10 *scaleR*

locale *scaleR-ow* =
fixes U **and** $\text{scaleR} :: \langle \text{real} \Rightarrow 'a \Rightarrow 'a \rangle$
assumes *scaleR-closed*: $\langle \forall a \in U. \text{scaleR } r \text{ } a \in U \rangle$

lemma *scaleR-ow-typeclass*[*simp*]: $\langle \text{scaleR-ow UNIV scaleR} \rangle$ **for** scaleR
by (*simp add: scaleR-ow-def*)

lemma *scaleR-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ } == == > ((=) \text{ } == == > A \text{ } == == > A) \text{ } == == > (=)) \text{ } \text{scaleR-ow scaleR-ow} \rangle$
unfolding *scaleR-ow-def make-parametricity-proof-friendly*
by *transfer-prover*

7.11 *scaleC*

locale *scaleC-ow* = *scaleR-ow* +
fixes scaleC
assumes *scaleC-closed*: $\langle \forall a \in U. \text{scaleC } c \text{ } a \in U \rangle$
assumes $\langle \forall a \in U. \text{scaleR } r \text{ } a = \text{scaleC } (\text{complex-of-real } r) \text{ } a \rangle$

lemma *scaleC-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ } == == > ((=) \text{ } == == > A \text{ } == == > A) \text{ } == == > ((=) \text{ } == == > A \text{ } == == > A) \text{ } == == > (=)) \text{ } \text{scaleC-ow scaleC-ow} \rangle$
unfolding *scaleC-ow-def scaleC-ow-axioms-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *class-scaleC-ud*[*unoverload-def*]: $\langle \text{class.scaleC} = \text{scaleC-ow UNIV} \rangle$
by (*auto intro!: ext simp: class.scaleC-def scaleC-ow-def scaleR-ow-def scaleC-ow-axioms-def*)

7.12 *ab-group-add*

locale *ab-group-add-ow* = *comm-monoid-ow* *U* *plus zero + minus-ow U minus + uminus-ow U uminus*

for *U plus zero minus uminus +*
assumes $\langle \forall a \in U. \text{uminus } a \in U \rangle$
assumes $\forall a \in U. \text{plus } (\text{uminus } a) \ a = \text{zero}$
assumes $\forall a \in U. \forall b \in U. \text{minus } a \ b = \text{plus } a \ (\text{uminus } b)$

lemma *ab-group-add-ow-parametric[transfer-rule]*:

includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ =====} \rangle (A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle A \text{ =====} \rangle (A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle (A \text{ =====} \rangle A) \text{ =====} \rangle (=) \rangle$
ab-group-add-ow ab-group-add-ow
unfolding *ab-group-add-ow-def ab-group-add-ow-axioms-def*
apply *transfer-prover-start*
apply *transfer-step+*
by *transfer-prover*

lemma *ab-group-add-ow-typeclass[simp]*:

$\langle \text{ab-group-add-ow } V \ (+) \ 0 \ (-) \ \text{uminus} \rangle$
if $\langle 0 \in V \rangle \langle \forall x \in V. -x \in V \rangle \langle \forall x \in V. \forall y \in V. x + y \in V \rangle$
for *V :: <- :: ab-group-add set>*
using *that*
apply (*auto intro!*: *ab-group-add-ow.intro ab-group-add-ow-axioms.intro comm-monoid-ow-typeclass minus-ow.intro uminus-ow.intro*)
by *force*

lemma *class-ab-group-add-ud[unoverload-def]*: $\langle \text{class.ab-group-add} = \text{ab-group-add-ow } UNIV \rangle$

by (*auto intro!*: *ext simp: class.ab-group-add-def ab-group-add-ow-def class-comm-monoid-add-ud minus-ow-def uminus-ow-def ab-group-add-ow-axioms-def class.ab-group-add-axioms-def*)

7.13 *vector-space*

locale *vector-space-ow* = *ab-group-add-ow* *U plus zero minus uminus*

for *U plus zero minus uminus +*
fixes *scale :: 'f::field \Rightarrow 'a \Rightarrow 'a*
assumes
 $\langle \forall x \in U. \text{scale } a \ x \in U \rangle$
 $\forall x \in U. \forall y \in U. \text{scale } a \ (\text{plus } x \ y) = \text{plus } (\text{scale } a \ x) \ (\text{scale } a \ y)$
 $\forall x \in U. \text{scale } (a + b) \ x = \text{plus } (\text{scale } a \ x) \ (\text{scale } b \ x)$
 $\forall x \in U. \text{scale } a \ (\text{scale } b \ x) = \text{scale } (a * b) \ x$
 $\forall x \in U. \text{scale } 1 \ x = x$

lemma *vector-space-ow-parametric[transfer-rule]*:

includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{ =====} \rangle (A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle A \text{ =====} \rangle (A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle (A \text{ =====} \rangle A) \text{ =====} \rangle ((=) \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle (=) \rangle$

```

    vector-space-ow vector-space-ow)
unfolding vector-space-ow-def vector-space-ow-axioms-def
apply transfer-prover-start
    apply transfer-step+
by simp

```

7.14 complex-vector

locale *complex-vector-ow* = *vector-space-ow U plus zero minus uminus scaleC + scaleC-ow U scaleR scaleC*

for *U scaleR scaleC plus zero minus uminus*

lemma *complex-vector-ow-parametric[transfer-rule]*:

includes *lifting-syntax*

assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$

shows $\langle (\text{rel-set } A \text{ =====} \rangle ((=) \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle ((=) \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle$
 $(A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle$

$A \text{ =====} \rangle (A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle (A \text{ =====} \rangle A) \text{ =====} \rangle (=)$

complex-vector-ow complex-vector-ow

unfolding *complex-vector-ow-def make-parametricity-proof-friendly*

by *transfer-prover*

lemma *class-complex-vector-ud[unoverload-def]*: $\langle \text{class.complex-vector} = \text{complex-vector-ow UNIV} \rangle$

by (*auto intro!*: *ext simp: class.complex-vector-def vector-space-ow-def vector-space-ow-axioms-def class.complex-vector-axioms-def class.scaleC-def complex-vector-ow-def class.scaleC-ud class-ab-group-add-ud*)

lemma *vector-space-ow-typeclass[simp]*:

$\langle \text{vector-space-ow } V (+) 0 (-) \text{ uminus } (*_C) \rangle$

if [*simp*]: $\langle \text{csubspace } V \rangle$

for $V :: \langle \text{::complex-vector set} \rangle$

by (*auto intro!*: *vector-space-ow.intro ab-group-add-ow-typeclass scaleC-left.add*

vector-space-ow-axioms.intro complex-vector.subspace-neg scaleC-add-right

complex-vector.subspace-add complex-vector.subspace-scale complex-vector.subspace-0)

lemma *complex-vector-ow-typeclass[simp]*:

$\langle \text{complex-vector-ow } V (*_R) (*_C) (+) 0 (-) \text{ uminus} \rangle$ **if** [*simp*]: $\langle \text{csubspace } V \rangle$

by (*auto intro!*: *scaleC-ow-def simp add: complex-vector-ow-def scaleC-ow-def*

scaleC-ow-axioms-def scaleR-ow-def scaleR-scaleC complex-vector.subspace-scale)

7.15 open-uniformity

locale *open-uniformity-ow* = *open open + uniformity uniformity*

for A *open uniformity +*

assumes *open-uniformity*:

$\bigwedge U. U \subseteq A \implies \text{open } U \longleftrightarrow (\forall x \in U. \text{eventually } (\lambda(x', y). x' = x \longrightarrow y \in U) \text{ uniformity})$

lemma *open-uniformity-ow-parametric[transfer-rule]*:

includes *lifting-syntax*

assumes $[transfer\text{-}rule]: \langle bi\text{-}unique\ A \rangle$
shows $\langle (rel\text{-}set\ A\ ==>\ (rel\text{-}set\ A\ ==>\ (=))\ ==>\ rel\text{-}filter\ (rel\text{-}prod\ A\ A)\ ==>\ (=))$
 $\quad open\text{-}uniformity\text{-}ow\ open\text{-}uniformity\text{-}ow \rangle$
unfolding $open\text{-}uniformity\text{-}ow\text{-}def\ make\text{-}parametricity\text{-}proof\text{-}friendly$
by $transfer\text{-}prover$

lemma $class\text{-}open\text{-}uniformity\text{-}ud[unoverload\text{-}def]: \langle class.open\text{-}uniformity = open\text{-}uniformity\text{-}ow$
 $UNIV \rangle$
by $(auto\ intro!: ext\ simp: class.open\text{-}uniformity\text{-}def\ open\text{-}uniformity\text{-}ow\text{-}def)$

lemma $open\text{-}uniformity\text{-}on\text{-}typeclass[simp]:$
fixes $V :: \langle -: open\text{-}uniformity\ set \rangle$
assumes $\langle closed\ V \rangle$
shows $\langle open\text{-}uniformity\text{-}ow\ V\ (openin\ (top\text{-}of\text{-}set\ V))\ (uniformity\text{-}on\ V) \rangle$
proof $(rule\ open\text{-}uniformity\text{-}ow.intro, intro\ allI\ impI\ iffI\ ballI)$
fix U **assume** $\langle U \subseteq V \rangle$
assume $\langle openin\ (top\text{-}of\text{-}set\ V)\ U \rangle$
then obtain T **where** $\langle U = T \cap V \rangle$ **and** $\langle open\ T \rangle$
by $(metis\ Int\text{-}ac(3)\ openin\text{-}open)$
with $open\text{-}uniformity$
have $*$: $\langle \forall_F\ (x', y)\ in\ uniformity.\ x' = x \longrightarrow y \in T \rangle$ **if** $\langle x \in T \rangle$ **for** x
using $that$ **by** $blast$
have $\langle \forall_F\ (x', y)\ in\ uniformity\text{-}on\ V.\ x' = x \longrightarrow y \in U \rangle$ **if** $\langle x \in U \rangle$ **for** x
apply $(rule\ eventually\text{-}inf\text{-}principal[THEN\ iffD2])$
using $*[of\ x]$ **apply** $(rule\ eventually\text{-}rev\text{-}mp)$
using $\langle U = T \cap V \rangle$ **that** **by** $(auto\ intro!: always\text{-}eventually)$
then show $\langle \forall_F\ (x', y)\ in\ uniformity\text{-}on\ V.\ x' = x \longrightarrow y \in U \rangle$ **if** $\langle x \in U \rangle$ **for** x
using $that$ **by** $blast$
next
fix U **assume** $\langle U \subseteq V \rangle$
assume $asm: \langle \forall x \in U.\ \forall_F\ (x', y)\ in\ uniformity\text{-}on\ V.\ x' = x \longrightarrow y \in U \rangle$
from $asm[rule\text{-}format]$
have $\langle \forall_F\ (x', y)\ in\ uniformity.\ x' \in V \wedge y \in V \wedge x' = x \longrightarrow y \in U \cup -V \rangle$ **if** $\langle x \in U \rangle$ **for**
 x
unfolding $eventually\text{-}inf\text{-}principal$
apply $(rule\ eventually\text{-}rev\text{-}mp)$
using $that$ **by** $(auto\ intro!: always\text{-}eventually)$
then have $xU: \langle \forall_F\ (x', y)\ in\ uniformity.\ x' = x \longrightarrow y \in U \cup -V \rangle$ **if** $\langle x \in U \rangle$ **for** x
apply $(rule\ eventually\text{-}rev\text{-}mp)$
using $that$ $\langle U \subseteq V \rangle$ **by** $(auto\ intro!: always\text{-}eventually)$
have $\langle open\ (-V) \rangle$
using $assms$ **by** $auto$
with $open\text{-}uniformity$
have $\langle \forall_F\ (x', y)\ in\ uniformity.\ x' = x \longrightarrow y \in -V \rangle$ **if** $\langle x \in -V \rangle$ **for** x
using $that$ **by** $blast$
then have $xV: \langle \forall_F\ (x', y)\ in\ uniformity.\ x' = x \longrightarrow y \in U \cup -V \rangle$ **if** $\langle x \in -V \rangle$ **for** x
apply $(rule\ eventually\text{-}rev\text{-}mp)$
apply $(rule\ that)$
apply $(rule\ always\text{-}eventually)$

```

  by auto
  have ⟨ $\forall_F (x', y)$  in uniformity.  $x' = x \longrightarrow y \in U \cup -V$ ⟩ if ⟨ $x \in U \cup -V$ ⟩ for  $x$ 
  using  $xV$ [of  $x$ ]  $xU$ [of  $x$ ] that
  by auto
  then have ⟨open ( $U \cup -V$ )⟩
  using open-uniformity by blast
  then show ⟨openin (top-of-set  $V$ )  $U$ ⟩
  using ⟨ $U \subseteq V$ ⟩
  by (auto intro!: exI simp: openin-open)
qed

```

7.16 *uniformity-dist*

```

locale uniformity-dist-ow = dist dist + uniformity uniformity for  $U$  dist uniformity +
  assumes uniformity-dist: uniformity = ( $\prod e \in \{0 < ..\}$ . principal  $\{(x, y) \in U \times U. \text{dist } x \ y < e\}$ )

```

```

lemma class-uniformity-dist-ud[unoverload-def]: ⟨class.uniformity-dist = uniformity-dist-ow UNIV⟩
  by (auto intro!: ext simp: class-uniformity-dist-def uniformity-dist-ow-def)

```

```

lemma uniformity-dist-ow-parametric[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: ⟨bi-unique  $A$ ⟩
  shows ⟨(rel-set  $A$   $\implies$ ) ( $A$   $\implies$ )  $A$   $\implies$  (=)  $\implies$  rel-filter (rel-prod  $A$   $A$ )  $\implies$  (=)⟩
  uniformity-dist-ow uniformity-dist-ow

```

```

proof –
  have *: uniformity-dist-ow  $U$  dist uniformity  $\longleftrightarrow$ 
    uniformity = transfer-bounded-filter-Inf (transfer-Times  $U$   $U$ )
    ((le. principal (Set.filter ( $\lambda(x,y). \text{dist } x \ y < e$ ) (transfer-Times  $U$   $U$ ))) ‘ $\{0 < ..\}$ ’)
  for  $U$  dist uniformity
  unfolding uniformity-dist-ow-def make-parametricity-proof-friendly case-prod-unfold
    prod.collapse
  apply (subst Inf-bounded-transfer-bounded-filter-Inf[where  $B = \langle U \times U \rangle$ ])
  by (auto simp: transfer-Times-def)
  show ?thesis
  unfolding *[abs-def]
  by transfer-prover
qed

```

```

lemma uniformity-dist-on-typeclass[simp]: ⟨uniformity-dist-ow  $V$  dist (uniformity-on  $V$ )⟩ for  $V$ 
  :: ⟨uniformity-dist set⟩
  apply (auto simp add: uniformity-dist-ow-def uniformity-dist simp flip: INF-inf-const2)
  apply (subst asm-rl[of ⟨ $\bigwedge x. \text{Restr } \{(xa, y). \text{dist } xa \ y < x\} \ V = \{(xa, y). xa \in V \wedge y \in V \wedge \text{dist } xa \ y < x\}$ ⟩, rule-format])
  by auto

```

7.17 *sgn*

```

locale sgn-ow =

```

fixes U **and** $sgn :: \langle 'a \Rightarrow 'a \rangle$
assumes $sgn\text{-closed}$: $\langle \forall a \in U. sgn\ a \in U \rangle$

lemma $sgn\text{-ow-parametric}$ [$transfer\text{-rule}$]:
includes $lifting\text{-syntax}$
assumes [$transfer\text{-rule}$]: $\langle bi\text{-unique}\ A \rangle$
shows $\langle (rel\text{-set}\ A\ ==> (A\ ==> A)\ ==> (=))\ ==> (=) \rangle$
 $sgn\text{-ow}\ sgn\text{-ow}$
unfolding $sgn\text{-ow-def}$
by $transfer\text{-prover}$

7.18 $sgn\text{-div-norm}$

locale $sgn\text{-div-norm-ow} = scaleR\text{-ow}\ U\ scaleR + norm\ norm + sgn\text{-ow}\ U\ sgn$ **for** $U\ sgn\ norm$
 $scaleR +$
assumes $\forall x \in U. sgn\ x = scaleR\ (inverse\ (norm\ x))\ x$

lemma $class\text{-sgn-div-norm-ud}$ [$unoverload\text{-def}$]: $\langle class.\text{sgn-div-norm} = sgn\text{-div-norm-ow}\ UNIV \rangle$
by ($auto\ intro!$: $ext\ simp$: $class.\text{sgn-div-norm-def}\ sgn\text{-div-norm-ow-def}\ sgn\text{-div-norm-ow-axioms-def}\ unoverload\text{-def}\ sgn\text{-ow-def}$)

lemma $sgn\text{-div-norm-ow-parametric}$ [$transfer\text{-rule}$]:
includes $lifting\text{-syntax}$
assumes [$transfer\text{-rule}$]: $\langle bi\text{-unique}\ A \rangle$
shows $\langle (rel\text{-set}\ A\ ==> (A\ ==> A)\ ==> (A\ ==> (=))\ ==> ((=)\ ==> A\ ==> A)\ ==> (=) \rangle$
 $sgn\text{-div-norm-ow}\ sgn\text{-div-norm-ow}$
unfolding $sgn\text{-div-norm-ow-def}\ sgn\text{-div-norm-ow-axioms-def}\ make\text{-parametricity-proof-friendly}$
by $transfer\text{-prover}$

lemma $sgn\text{-div-norm-on-typeclass}$ [$simp$]:
fixes $V :: \langle \cdot :: sgn\text{-div-norm}\ set \rangle$
assumes $\langle \bigwedge v\ r. v \in V \implies scaleR\ r\ v \in V \rangle$
shows $\langle sgn\text{-div-norm-ow}\ V\ sgn\ norm\ (*_R) \rangle$
using $assms$
by ($auto\ simp\ add$: $sgn\text{-ow-def}\ sgn\text{-div-norm-ow-axioms-def}\ scaleR\text{-ow-def}\ sgn\text{-div-norm-ow-def}\ sgn\text{-div-norm}$)

7.19 $dist\text{-norm}$

locale $dist\text{-norm-ow} = dist\ dist + norm\ norm + minus\text{-ow}\ U\ minus$ **for** $U\ minus\ dist\ norm +$
assumes $dist\text{-norm}$: $\forall x \in U. \forall y \in U. dist\ x\ y = norm\ (minus\ x\ y)$

lemma $dist\text{-norm-ud}$ [$unoverload\text{-def}$]: $\langle class.\text{dist-norm} = dist\text{-norm-ow}\ UNIV \rangle$
by ($auto\ intro!$: $ext\ simp$: $class.\text{dist-norm-def}\ dist\text{-norm-ow-def}\ dist\text{-norm-ow-axioms-def}\ minus\text{-ow-def}\ unoverload\text{-def}$)

lemma $dist\text{-norm-ow-parametric}$ [$transfer\text{-rule}$]:
includes $lifting\text{-syntax}$

assumes [transfer-rule]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \implies (A \implies A \implies A) \implies (A \implies A \implies (=)) \implies (A \implies (=)) \implies (=)) \implies (=) \rangle$
dist-norm-ow dist-norm-ow
unfolding *dist-norm-ow-def dist-norm-ow-axioms-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *dist-norm-ow-typeclass[simp]*:
fixes $A :: \langle \text{dist-norm set} \rangle$
assumes $\langle \bigwedge a b. [a \in A; b \in A] \implies a - b \in A \rangle$
shows $\langle \text{dist-norm-ow } A \text{ (} - \text{) dist norm} \rangle$
by (*auto simp add: assms dist-norm-ow-def minus-ow-def dist-norm-ow-axioms-def dist-norm*)

7.20 complex-inner

locale *complex-inner-ow = complex-vector-ow U scaleR scaleC plus zero minus uminus*
+ dist-norm-ow U minus dist norm + sgn-div-norm-ow U sgn norm scaleR
+ uniformity-dist-ow U dist uniformity
+ open-uniformity-ow U open uniformity
for *U scaleR scaleC plus zero minus uminus dist norm sgn uniformity open +*
fixes *cinner :: 'a \Rightarrow 'a \Rightarrow complex*
assumes $\forall x \in U. \forall y \in U. \text{cinner } x y = \text{cnj } (\text{cinner } y x)$
and $\forall x \in U. \forall y \in U. \forall z \in U. \text{cinner } (\text{plus } x y) z = \text{cinner } x z + \text{cinner } y z$
and $\forall x \in U. \forall y \in U. \text{cinner } (\text{scaleC } r x) y = \text{cnj } r * \text{cinner } x y$
and $\forall x \in U. 0 \leq \text{cinner } x x$
and $\forall x \in U. \text{cinner } x x = 0 \iff x = \text{zero}$
and $\forall x \in U. \text{norm } x = \text{sqrt } (\text{cmod } (\text{cinner } x x))$

lemma *class-complex-inner-ud[unoverload-def]*: $\langle \text{class.complex-inner} = \text{complex-inner-ow UNIV} \rangle$
apply (*intro ext*)
by (*simp add: class-complex-inner-def class-complex-inner-axioms-def complex-inner-ow-def complex-inner-ow-axioms-def unoverload-def*)

lemma *complex-inner-ow-parametricity[transfer-rule]*:
includes *lifting-syntax*
assumes [transfer-rule]: $\langle \text{bi-unique } T \rangle$
shows $\langle (\text{rel-set } T \implies ((=) \implies T \implies T) \implies ((=) \implies T \implies T) \implies (T \implies T \implies T) \implies T \implies (T \implies T \implies T) \implies (T \implies T) \implies (T \implies T \implies (=)) \implies (T \implies T) \implies \text{rel-filter } (\text{rel-prod } T T) \implies (\text{rel-set } T \implies (=)) \implies (T \implies T \implies (=)) \implies (=) \rangle$
complex-inner-ow complex-inner-ow
unfolding *complex-inner-ow-def complex-inner-ow-axioms-def*
by *transfer-prover*

lemma *complex-inner-ow-typeclass[simp]*:
fixes $V :: \langle \text{complex-inner set} \rangle$
assumes [simp]: $\langle \text{closed } V \rangle \langle \text{csubspace } V \rangle$
shows $\langle \text{complex-inner-ow } V (*_R) (*_C) (+) 0 (-) \text{uminus dist norm sgn (uniformity-on } V) \rangle$

(*openin (top-of-set V)*) (\cdot_C)
apply (*auto intro!*: *complex-vector-ow-typeclass dist-norm-ow-typeclass sgn-div-norm-on-typeclass*
simp: *complex-inner-ow-def cinner-simps complex-vector.subspace-diff complex-inner-ow-axioms-def*
scaleR-scaleC complex-vector.subspace-scale
simp flip: *norm-eq-sqrt-cinner*)
by –

7.21 *is-ortho-set*

definition *is-ortho-set-ow* **where** $\langle \text{is-ortho-set-ow zero cinner } S \longleftrightarrow$
 $((\forall x \in S. \forall y \in S. x \neq y \longrightarrow \text{cinner } x \ y = 0) \wedge \text{zero} \notin S) \rangle$
for *zero cinner*

lemma *is-ortho-set-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (A \text{====} \Rightarrow (A \text{====} \Rightarrow A \text{====} \Rightarrow (=)) \text{====} \Rightarrow \text{rel-set } A \text{====} \Rightarrow (=)) \text{====} \Rightarrow$
 $\text{is-ortho-set-ow is-ortho-set-ow} \rangle$
unfolding *is-ortho-set-ow-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *is-ortho-set-ud*[*unoverload-def*]: $\langle \text{is-ortho-set} = \text{is-ortho-set-ow } 0 \text{ cinner} \rangle$
by (*auto simp*: *is-ortho-set-ow-def is-ortho-set-def*)

7.22 *metric-space*

locale *metric-space-ow* = *uniformity-dist-ow U dist uniformity + open-uniformity-ow U open*
uniformity
for *U dist uniformity open +*
assumes $\forall x \in U. \forall y \in U. \text{dist } x \ y = 0 \longleftrightarrow x = y$
and $\forall x \in U. \forall y \in U. \forall z \in U. \text{dist } x \ y \leq \text{dist } x \ z + \text{dist } y \ z$

lemma *metric-space-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \text{====} \Rightarrow (A \text{====} \Rightarrow A \text{====} \Rightarrow (=)) \text{====} \Rightarrow \text{rel-filter } (\text{rel-prod } A \ A) \text{====} \Rightarrow$
 $(\text{rel-set } A \text{====} \Rightarrow (=)) \text{====} \Rightarrow (=)) \text{====} \Rightarrow$
 $\text{metric-space-ow metric-space-ow} \rangle$
unfolding *metric-space-ow-def metric-space-ow-axioms-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *class-metric-space-ud*[*unoverload-def*]: $\langle \text{class.metric-space} = \text{metric-space-ow UNIV} \rangle$
by (*auto intro!*: *ext simp*: *class.metric-space-def class.metric-space-axioms-def metric-space-ow-def*
metric-space-ow-axioms-def unoverload-def)

lemma *metric-space-ow-typeclass*[*simp*]:
fixes $V :: \langle \text{metric-space set} \rangle$
assumes $\langle \text{closed } V \rangle$
shows $\langle \text{metric-space-ow } V \text{ dist } (\text{uniformity-on } V) (\text{openin } (\text{top-of-set } V)) \rangle$

by (auto simp: assms metric-space-ow-def metric-space-ow-axioms-def class.metric-space-axioms-def dist-triangle2)

7.23 nhds

definition *nhds-ow* where $\langle \text{nhds-ow } U \text{ open } a = (\text{INF } S \in \{S. S \subseteq U \wedge \text{open } S \wedge a \in S\}. \text{principal } S) \sqcap \text{principal } U \rangle$
for $U \text{ open}$

lemma *nhds-ow-parametric*[transfer-rule]:

includes *lifting-syntax*

assumes [transfer-rule]: $\langle \text{bi-unique } A \rangle$

shows $\langle (\text{rel-set } A \text{ ===>} (\text{rel-set } A \text{ ===>} (=)) \text{ ===>} A \text{ ===>} \text{rel-filter } A)$

nhds-ow nhds-ow

unfolding *nhds-ow-def*[folded transfer-bounded-filter-Inf-def] *make-parametricity-proof-friendly*

by *transfer-prover*

lemma *topological-space-nhds-ud*[unoverload-def]: $\langle \text{topological-space.nhds} = \text{nhds-ow UNIV} \rangle$

by (auto intro!: ext simp add: nhds-ow-def [[axiom topological-space.nhds-def-raw]])

lemma *nhds-ud*[unoverload-def]: $\langle \text{nhds} = \text{nhds-ow UNIV open} \rangle$

by (auto intro!: ext simp add: nhds-ow-def nhds-def)

lemma *nhds-ow-topology*[simp]: $\langle \text{nhds-ow } (\text{topspace } T) (\text{openin } T) x = \text{nhdsin } T x \rangle$ if $\langle x \in \text{topspace } T \rangle$

using that **apply** (auto intro!: ext simp add: nhds-ow-def nhdsin-def[abs-def])

apply (subst INF-inf-const2[symmetric])

using *openin-subset* by (auto intro!: INF-cong)

7.24 at-within

definition $\langle \text{at-within-ow } U \text{ open } a s = \text{nhds-ow } U \text{ open } a \sqcap \text{principal } (s - \{a\}) \rangle$

for $U \text{ open } a s$

lemma *at-within-ow-parametric*[transfer-rule]:

includes *lifting-syntax*

assumes [transfer-rule]: $\langle \text{bi-unique } T \rangle$

shows $\langle ((\text{rel-set } T \text{ ===>} (\text{rel-set } T \text{ ===>} (=)) \text{ ===>} T \text{ ===>} \text{rel-set } T \text{ ===>} \text{rel-filter } T)$

$\text{at-within-ow at-within-ow}$

unfolding *at-within-ow-def* *make-parametricity-proof-friendly* *transfer-inf-principal-def*[symmetric]

by *transfer-prover*

lemma *at-within-ud*[unoverload-def]: $\langle \text{at-within} = \text{at-within-ow UNIV open} \rangle$

by (auto intro!: ext simp: at-within-def at-within-ow-def unoverload-def)

lemma *at-within-ow-topology*:

$\langle \text{at-within-ow } (\text{topspace } T) (\text{openin } T) a S = \text{nhdsin } T a \sqcap \text{principal } (S - \{a\}) \rangle$

if $\langle a \in \text{topspace } T \rangle$

using that **unfolding** *at-within-ow-def* by (*simp add: nhds-ow-topology*)

7.25 (*has-sum*)

definition \langle has-sum-ow U plus zero open $f A x =$
 $filterlim (sum-ow zero plus f) (nhds-ow U (\lambda S. open S) x)$
 $(finite-subsets-at-top A)\rangle$
for U plus zero open $f A x$

lemma *has-sum-ow-parametric[transfer-rule]*:

includes *lifting-syntax*

assumes [*transfer-rule*]: \langle bi-unique $T\rangle$ \langle bi-unique $U\rangle$

shows \langle (rel-set $T \implies (V \implies T \implies T) \implies T \implies$ (rel-set $T \implies (=)$)
 $\implies (U \implies V) \implies rel-set U \implies T \implies (=)$)
 $has-sum-ow has-sum-ow\rangle$

unfolding *has-sum-ow-def*

by *transfer-prover*

lemma *has-sum-ud[unoverload-def]*: \langle HAS-SUM = has-sum-ow UNIV plus $(0::'a::\{comm-monoid-add,topological-space\})$ open \rangle

by (*auto intro!: ext simp: has-sum-def has-sum-ow-def unoverload-def*)

lemma *has-sum-ow-topology*:

assumes $\langle l \in topspace T\rangle$

assumes $\langle 0 \in topspace T\rangle$

assumes $\langle \bigwedge x y. x \in topspace T \implies y \in topspace T \implies x + y \in topspace T\rangle$

shows \langle has-sum-ow ($topspace T$) $(+)$ 0 (*openin* T) $f S l \iff$ has-sum-in $T f S l\rangle$

using *assms apply (simp add: has-sum-ow-def has-sum-in-def nhds-ow-topology sum-ud[symmetric])*

by (*metis filterlim-nhdsin-iff-limitin*)

7.26 *filterlim*

7.27 *convergent*

definition *convergent-ow where*

\langle convergent-ow U open $X \iff (\exists L \in U. filterlim X (nhds-ow U open L) sequentially)\rangle$

for U open

lemma *convergent-ow-parametric[transfer-rule]*:

includes *lifting-syntax*

assumes [*transfer-rule*]: \langle bi-unique $T\rangle$

shows \langle (rel-set $T \implies (rel-set T \implies (=)) \implies ((=) \implies T) \implies (\iff)$)
 $convergent-ow convergent-ow\rangle$

unfolding *convergent-ow-def*

by *transfer-prover*

lemma *convergent-ud[unoverload-def]*: \langle convergent = convergent-ow UNIV open \rangle

by (*auto simp: convergent-ow-def[abs-def] convergent-def[abs-def] unoverload-def*)

lemma *topological-space-convergent-ud*[*unoverload-def*]: $\langle \text{topological-space.convergent} = \text{convergent-ow UNIV} \rangle$

by (*auto intro!*: *ext simp*: [[*axiom topological-space.convergent-def-raw*]]
convergent-ow-def unoverload-def)

lemma *convergent-ow-topology*[*simp*]:

$\langle \text{convergent-ow (topspace } T) (\text{openin } T) f \longleftrightarrow (\exists l. \text{limitin } T f l \text{ sequentially}) \rangle$
by (*auto simp*: *convergent-ow-def simp flip*: *filterlim-nhdsin-iff-limitin*)

lemma *convergent-ow-typeclass*[*simp*]:

$\langle \text{convergent-ow } V (\text{openin (top-of-set } V)) f \longleftrightarrow (\exists l. \text{limitin (top-of-set } V) f l \text{ sequentially}) \rangle$
by (*simp flip*: *convergent-ow-topology*)

7.28 *uniform-space.cauchy-filter*

lemma *cauchy-filter-parametric*[*transfer-rule*]:

includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique T*
shows (*rel-filter (rel-prod T T) =====> rel-filter T =====> (=)*)
uniform-space.cauchy-filter
uniform-space.cauchy-filter
unfolding [[*axiom uniform-space.cauchy-filter-def-raw*]]
by *transfer-prover*

7.29 *uniform-space.Cauchy*

lemma *uniform-space-Cauchy-parametric*[*transfer-rule*]:

includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique T*
shows (*rel-filter (rel-prod T T) =====> ((=) =====> T) =====> (=)*)
uniform-space.Cauchy
uniform-space.Cauchy
unfolding [[*axiom uniform-space.Cauchy-uniform-raw*]]
using *filtermap-parametric*[*transfer-rule*] **apply** *fail?*
by *transfer-prover*

7.30 *complete-space*

locale *complete-space-ow* = *metric-space-ow U dist uniformity open*

for *U dist uniformity open* +

assumes $\langle \text{range } X \subseteq U \longrightarrow \text{uniform-space.Cauchy uniformity } X \longrightarrow \text{convergent-ow } U \text{ open } X \rangle$

lemma *class-complete-space-ud*[*unoverload-def*]: $\langle \text{class.complete-space} = \text{complete-space-ow UNIV} \rangle$

by (*auto intro!*: *ext simp*: *class.complete-space-def class.complete-space-axioms-def complete-space-ow-def complete-space-ow-axioms-def unoverload-def*)

lemma *complete-space-ow-parametric*[*transfer-rule*]:

includes *lifting-syntax*

```

assumes [transfer-rule]: bi-unique T
shows (rel-set T ===> (T ===> T ===> (=)) ===> rel-filter (rel-prod T T) ===>
(rel-set T ===> (=)) ===> (=))
  complete-space-ow complete-space-ow
unfolding complete-space-ow-def complete-space-ow-axioms-def make-parametricity-proof-friendly
by transfer-prover

```

```

lemma complete-space-ow-typeclass[simp]:
  fixes V :: <-::uniform-space set>
  assumes <complete V>
  shows <complete-space-ow V dist (uniformity-on V) (openin (top-of-set V))>
proof (rule complete-space-ow.intro)
  show <metric-space-ow V dist (uniformity-on V) (openin (top-of-set V))>
    apply (rule metric-space-ow-typeclass)
    by (simp add: assms complete-imp-closed)
  have < $\exists l. \text{limitin } (top\text{-of-set } V) X l \text{ sequentially}$ >
    if XV: < $\bigwedge n. X n \in V$ > and cauchy: <uniform-space.Cauchy (uniformity-on V) X> for X
  proof –
    from cauchy
    have <uniform-space.cauchy-filter (uniformity-on V) (filtermap X sequentially)>
      by (simp add: [[axiom uniform-space.Cauchy-uniform-raw]])
    then have <cauchy-filter (filtermap X sequentially)>
      by (auto simp: cauchy-filter-def [[axiom uniform-space.cauchy-filter-def-raw]])
    then have <Cauchy X>
      by (simp add: Cauchy-uniform)
    with <complete V> XV obtain l where l: <X  $\longrightarrow$  l> <l  $\in$  V>
    apply atomize-elim
    by (meson completeE)
  with XV l show ?thesis
    by (auto intro!: exI[of -] simp: convergent-def limitin-subtopology)
  qed
  then show <complete-space-ow-axioms V (uniformity-on V) (openin (top-of-set V))>
    apply (auto simp: complete-space-ow-axioms-def complete-imp-closed assms)
    by blast
qed

```

7.31 chilbert-space

locale chilbert-space-ow = complex-inner-ow + complete-space-ow

```

lemma chilbert-space-ow-parametric[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]: <bi-unique A>
  shows <(rel-set A ===> ((=) ===> A ===> A) ===> ((=) ===> A ===> A) ===>
(A ===> A ===> A) ===>
  A ===> (A ===> A ===> A) ===> (A ===> A) ===> (A ===> A ===> (=))
===> (A ===> (=)) ===>
  (A ===> A) ===> rel-filter (rel-prod A A) ===> (rel-set A ===> (=)) ===> (A
===> A ===> (=)) ===> (=)

```

chilbert-space-ow chilbert-space-ow
unfolding *chilbert-space-ow-def make-parametricity-proof-friendly*
by *transfer-prover*

lemma *chilbert-space-on-typeclass[simp]*:
fixes $V :: \langle - :: \text{complex-inner set} \rangle$
assumes $\langle \text{complete } V \rangle \langle \text{csubspace } V \rangle$
shows $\langle \text{chilbert-space-ow } V (*_R) (*_C) (+) 0 (-) \text{uminus dist norm sgn}$
 $(\text{uniformity-on } V) (\text{openin (top-of-set } V)) (\cdot_C) \rangle$
by (*auto intro!*: *chilbert-space-ow.intro complex-inner-ow-typeclass*
simp: assms complete-imp-closed)

lemma *class-chilbert-space-ud[unoverload-def]*:
 $\langle \text{class.chilbert-space} = \text{chilbert-space-ow UNIV} \rangle$
by (*auto intro!*: *ext simp add: class.chilbert-space-def chilbert-space-ow-def unoverload-def*)

7.32 (hull)

definition $\langle \text{hull-ow } A S s = ((\lambda x. S x \wedge x \subseteq A) \text{ hull } s) \cap A \rangle$

lemma *hull-ow-nondegenerate*: $\langle \text{hull-ow } A S s = ((\lambda x. S x \wedge x \subseteq A) \text{ hull } s) \rangle$ **if** $\langle x \subseteq A \rangle$ **and**
 $\langle s \subseteq x \rangle$ **and** $\langle S x \rangle$

proof –

have $\langle ((\lambda x. S x \wedge x \subseteq A) \text{ hull } s) \subseteq x \rangle$

apply (*rule hull-minimal*)

using that by *auto*

also note $\langle x \subseteq A \rangle$

finally show *?thesis*

unfolding *hull-ow-def* **by** *auto*

qed

definition $\langle \text{transfer-bounded-Inf } B M = \text{Inf } M \sqcap B \rangle$

lemma *transfer-bounded-Inf-parametric[transfer-rule]*:

includes *lifting-syntax*

assumes $\langle \text{bi-unique } T \rangle$

shows $\langle (\text{rel-set } T \text{ ===>} \text{ rel-set } (\text{rel-set } T) \text{ ===>} \text{ rel-set } T) \text{ transfer-bounded-Inf trans-fer-bounded-Inf} \rangle$

apply (*auto intro!*: *rel-funI simp: transfer-bounded-Inf-def rel-set-def Bex-def*)

apply (*metis (full-types) assms bi-uniqueDr*)

by (*metis (full-types) assms bi-uniqueDl*)

lemma *hull-ow-parametric[transfer-rule]*:

includes *lifting-syntax*

assumes [*transfer-rule*]: *bi-unique T*

shows $(\text{rel-set } T \text{ ===>} (\text{rel-set } T \text{ ===>} (=)) \text{ ===>} \text{ rel-set } T \text{ ===>} \text{ rel-set } T)$

hull-ow hull-ow

proof –

have *: $\langle \text{hull-ow } A S s = \text{transfer-bounded-Inf } A (\text{Set.filter } (\lambda x. S x \wedge s \subseteq x) (\text{Pow } A)) \rangle$ **for**

A S s
by (*auto simp add: hull-ow-def hull-def transfer-bounded-Inf-def*)
show *?thesis*
unfolding *
by *transfer-prover*
qed

lemma *hull-ow-ud[unoverload-def]: $\langle \text{hull} \rangle = \text{hull-ow UNIV}$*
unfolding *hull-def hull-ow-def* **by** *auto*

7.33 *csubspace*

definition

$\langle \text{subspace-ow plus zero scale } S = (\text{zero} \in S \wedge (\forall x \in S. \forall y \in S. \text{plus } x \ y \in S) \wedge (\forall c. \forall x \in S. \text{scale } c \ x \in S)) \rangle$
for *plus zero scale S*

lemma *subspace-ow-parametric[transfer-rule]:*

includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } T \rangle$
shows $\langle ((T \text{====>} T \text{====>} T) \text{====>} T \text{====>} ((=) \text{====>} T \text{====>} T) \text{====>} \text{rel-set } T \text{====>} (=) \rangle$
subspace-ow subspace-ow
unfolding *subspace-ow-def*
by *transfer-prover*

lemma *module-subspace-ud[unoverload-def]: $\langle \text{module.subspace} = \text{subspace-ow plus } 0 \rangle$*
by (*auto intro!: ext simp: [[axiom module.subspace-def-raw]] subspace-ow-def*)

lemma *csubspace-ud[unoverload-def]: $\langle \text{csubspace} = \text{subspace-ow } (+) \ 0 \ (*_C) \rangle$*
by (*simp add: csubspace-raw-def module-subspace-ud*)

7.34 *cspan*

definition

$\langle \text{span-ow } U \text{ plus zero scale } b = \text{hull-ow } U \ (\text{subspace-ow plus zero scale}) \ b \rangle$
for *U plus zero scale b*

lemma *span-ow-on-typeclass:*

assumes $\langle \text{csubspace } U \rangle$
assumes $\langle B \subseteq U \rangle$
shows $\langle \text{span-ow } U \text{ plus } 0 \text{ scale } C \ B = \text{cspan } B \rangle$

proof –

have $\langle \text{span-ow } U \text{ plus } 0 \text{ scale } C \ B = (\lambda x. \text{csubspace } x \wedge x \subseteq U) \ \text{hull } B \rangle$
using *assms*
by (*auto simp add: span-ow-def hull-ow-nondegenerate[where x=U] csubspace-raw-def simp flip: csubspace-ud*)
also have $\langle (\lambda x. \text{csubspace } x \wedge x \subseteq U) \ \text{hull } B = \text{cspan } B \rangle$
apply (*rule hull-unique*)

using *assms(2) complex-vector.span-superset* **apply** *force*
by (*simp-all add: assms complex-vector.span-minimal*)
finally show *?thesis*
by –
qed

lemma (**in** *Modules.module*) *span-ud[unoverload-def]*: $\langle \text{span} = \text{span-ow UNIV plus 0 scale} \rangle$
by (*auto intro!: ext simp: span-def span-ow-def*
module-subspace-ud hull-ow-ud)

lemmas *cspan-ud[unoverload-def]* = *complex-vector.span-ud*

lemma *span-ow-parametric[transfer-rule]*:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } T \rangle$
shows $\langle (\text{rel-set } T \text{ =====} \rangle (T \text{ =====} \rangle T \text{ =====} \rangle T) \text{ =====} \rangle T \text{ =====} \rangle ((=) \text{ =====} \rangle T \text{ =====} \rangle T) \text{ =====} \rangle \text{rel-set } T \text{ =====} \rangle \text{rel-set } T \rangle$
span-ow span-ow
unfolding *span-ow-def*
by *transfer-prover*

7.34.1 (*islimpt*)

definition $\langle \text{islimpt-ow } U \text{ open } x \ S \iff (\forall T \subseteq U. x \in T \longrightarrow \text{open } T \longrightarrow (\exists y \in S. y \in T \wedge y \neq x)) \rangle$
for *open*

lemma *islimpt-ow-parametric[transfer-rule]*:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } T \rangle$
shows $\langle (\text{rel-set } T \text{ =====} \rangle (\text{rel-set } T \text{ =====} \rangle (=)) \text{ =====} \rangle T \text{ =====} \rangle \text{rel-set } T \text{ =====} \rangle (\iff) \rangle$
islimpt-ow islimpt-ow
unfolding *islimpt-ow-def make-parametricity-proof-friendly*
by *transfer-prover*

definition $\langle \text{islimptin } T \ x \ S \iff x \in \text{topspace } T \wedge (\forall V. x \in V \longrightarrow \text{openin } T \ V \longrightarrow (\exists y \in S. y \in V \wedge y \neq x)) \rangle$

lemma *islimpt-ow-from-topology*: $\langle \text{islimpt-ow } (\text{topspace } T) \ (\text{openin } T) \ x \ S \iff \text{islimptin } T \ x \ S \vee x \notin \text{topspace } T \rangle$
apply (*cases* $\langle x \in \text{topspace } T \rangle$)
apply (*simp-all add: islimpt-ow-def islimptin-def Pow-def*)
by *blast+*

7.34.2 *closure*

definition $\langle \text{closure-ow } U \text{ open } S = S \cup \{x \in U. \text{islimpt-ow } U \text{ open } x \ S\} \rangle$ **for** *open*

lemma *closure-ow-with-typeclass[simp]*:
 $\langle \text{closure-ow } X \ (\text{openin } (\text{top-of-set } X)) \ S = (X \cap \text{closure } (X \cap S)) \cup S \rangle$

proof –

have $\langle \text{closure-ow } X \text{ (openin (top-of-set } X)) } S = (\text{top-of-set } X) \text{ closure-of } S \cup S \rangle$
apply (simp add: closure-ow-def islimpt-ow-def closure-of-def)
apply safe
apply (meson PowI openin-imp-subset)
by auto
also have $\langle \dots = (X \cap \text{closure } (X \cap S)) \cup S \rangle$
by (simp add: closure-of-subtopology)
finally show ?thesis
by –

qed

lemma closure-ow-parametric[transfer-rule]:

includes lifting-syntax
assumes [transfer-rule]: $\langle \text{bi-unique } T \rangle$
shows $\langle (\text{rel-set } T \text{ ==== } \Rightarrow (\text{rel-set } T \text{ ==== } (=)) \text{ ==== } \Rightarrow \text{rel-set } T \text{ ==== } \Rightarrow \text{rel-set } T) \text{ closure-ow closure-ow} \rangle$
unfolding closure-ow-def make-parametricity-proof-friendly
by transfer-prover

lemma closure-ow-from-topology: $\langle \text{closure-ow (topspace } T) \text{ (openin } T) } S = T \text{ closure-of } S \rangle$ **if** $\langle S \subseteq \text{topspace } T \rangle$

using that **apply** (auto simp: closure-ow-def islimpt-ow-from-topology in-closure-of)
apply (meson in-closure-of islimptin-def)
by (metis islimptin-def)

lemma closure-ud[unoverload-def]: $\langle \text{closure} = \text{closure-ow UNIV open} \rangle$

unfolding closure-def closure-ow-def islimpt-def islimpt-ow-def **by** auto

7.35 continuous

lemma continuous-on-ow-from-topology: $\langle \text{continuous-on-ow (topspace } T) \text{ (topspace } U) \text{ (openin } T) \text{ (openin } U) \text{ (topspace } T) } f \longleftrightarrow \text{continuous-map } T \ U \ f \rangle$

if $\langle f \text{ ' } \text{topspace } T \subseteq \text{topspace } U \rangle$

apply (simp add: continuous-on-ow-def continuous-map-def)

apply safe

apply (meson image-subset-iff that)

apply (smt (verit) Collect-mono-iff Int-def inf-absorb1 mem-Collect-eq openin-subopen openin-subset vimage-eq)

by blast

7.36 is-onb

definition

$\langle \text{is-onb-ow } U \text{ scaleC plus zero norm open cinner } E \longleftrightarrow \text{is-ortho-set-ow zero cinner } E \wedge (\forall b \in E. \text{norm } b = 1) \wedge$

$\text{closure-ow } U \text{ open (span-ow } U \text{ plus zero scaleC } E) = U \rangle$

for $U \text{ scaleC plus zero norm open cinner}$

```

lemma is-onb-ow-parametric[transfer-rule]:
  includes lifting-syntax
  assumes [transfer-rule]:  $\langle \text{bi-unique } A \rangle$ 
  shows  $\langle \text{rel-set } A \text{ =====} \rangle$ 
     $\langle \langle (=) \text{ =====} \rangle A \text{ =====} \rangle A \text{ =====} \rangle$ 
     $\langle A \text{ =====} \rangle A \text{ =====} \rangle A \text{ =====} \rangle$ 
     $A \text{ =====} \rangle$ 
     $\langle A \text{ =====} \rangle \langle (=) \text{ =====} \rangle \langle \text{rel-set } A \text{ =====} \rangle \langle (=) \text{ =====} \rangle \langle A \text{ =====} \rangle A \text{ =====} \rangle \langle (=) \text{ =====} \rangle$ 
  rel-set  $A \text{ =====} \rangle \langle (=) \rangle$ 
    is-onb-ow is-onb-ow
  unfolding is-onb-ow-def
  by transfer-prover

```

```

lemma is-onb-ud[unoverload-def]:
   $\langle \text{is-onb} = \text{is-onb-ow UNIV scaleC plus 0 norm open cinner} \rangle$ 
  unfolding is-onb-def is-onb-ow-def
  apply (subst asm-rl[of  $\langle \bigwedge E. \text{ccspan } E = \top \iff \text{closure } (\text{cspan } E) = \text{UNIV} \rangle$ , rule-format])
  apply (transfer, rule)
  unfolding unoverload-def
  apply transfer by auto

```

7.37 Transferring theorems

```

lemma closure-of-eqI:
  fixes  $f\ g :: 'a \Rightarrow 'b$  and  $T :: \langle 'a \text{ topology} \rangle$  and  $U :: \langle 'b \text{ topology} \rangle$ 
  assumes hausdorff:  $\langle \text{Hausdorff-space } U \rangle$ 
  assumes f-eq-g:  $\langle \bigwedge x. x \in S \implies f\ x = g\ x \rangle$ 
  assumes  $x :: \langle x \in T \text{ closure-of } S \rangle$ 
  assumes  $f :: \langle \text{continuous-map } T\ U\ f \rangle$  and  $g :: \langle \text{continuous-map } T\ U\ g \rangle$ 
  shows  $\langle f\ x = g\ x \rangle$ 
proof –
  have  $\langle \text{topspace } T \neq \{\} \rangle$ 
    by (metis assms(3) equals0D in-closure-of)
  have  $\langle \text{topspace } U \neq \{\} \rangle$ 
    using  $\langle \text{topspace } T \neq \{\} \rangle$  assms(5) continuous-map-image-subset-topspace by blast
  {
  assume  $\exists (\text{Rep} :: 't \Rightarrow 'a)$  Abs. type-definition Rep Abs (topspace T)
  then interpret  $T :: \text{local-typedef } \langle \text{topspace } T \rangle \langle \text{TYPE}('t) \rangle$ 
    by unfold-locales
  assume  $\exists (\text{Rep} :: 'u \Rightarrow 'b)$  Abs. type-definition Rep Abs (topspace U)
  then interpret  $U :: \text{local-typedef } \langle \text{topspace } U \rangle \langle \text{TYPE}('u) \rangle$ 
    by unfold-locales
  }
  note on-closure-eqI
  note this[unfolded unoverload-def]
  note this[unoverload-type 'b, unoverload-type 'a]
  note this[unfolded unoverload-def]
  note this[where  $'a = 't$  and  $'b = 'u$ ]

```

```

    note this[untransferred]
    note this[where f=f and g=g and S=⟨S ∩ topspace T⟩ and x=x and ?open=openin T
and opena=⟨openin U⟩]
    note this[simplified]
  }
  note * = this[cancel-type-definition, OF ⟨topspace T ≠ {}⟩, cancel-type-definition, OF ⟨topspace
U ≠ {}⟩]

  have 2: ⟨f ' topspace T ⊆ topspace U⟩
  by (meson assms(4) continuous-map-image-subset-topspace)
  have 3: ⟨g ' topspace T ⊆ topspace U⟩
  by (simp add: continuous-map-image-subset-topspace g)
  have 4: ⟨x ∈ topspace T⟩
  by (meson assms(3) in-closure-of)
  have 5: ⟨topological-space-ow (topspace T) (openin T)⟩
  by simp
  have 6: ⟨t2-space-ow (topspace U) (openin U)⟩
  by (simp add: hausdorff)
  from x have ⟨x ∈ T closure-of (S ∩ topspace T)⟩
  by (metis closure-of-restrict inf-commute)
  then have 7: ⟨x ∈ closure-ow (topspace T) (openin T) (S ∩ topspace T)⟩
  by (simp add: closure-ow-from-topology)
  have 8: ⟨continuous-on-ow (topspace T) (topspace U) (openin T) (openin U) (topspace T) f⟩
  by (meson 2 continuous-on-ow-from-topology f)
  have 9: ⟨continuous-on-ow (topspace T) (topspace U) (openin T) (openin U) (topspace T) g⟩
  by (simp add: 3 continuous-on-ow-from-topology g)

  show ?thesis
  apply (rule *)
  using 2 3 4 5 6 f-eq-g 7 8 9 by auto
qed

```

lemma *orthonormal-subspace-basis-exists*:

```

fixes S :: ⟨'a::hilbert-space set⟩
assumes ⟨is-ortho-set S⟩ and norm: ⟨ $\bigwedge x. x \in S \implies \text{norm } x = 1$ ⟩ and ⟨S ⊆ space-as-set V⟩
shows ⟨ $\exists B. B \supseteq S \wedge \text{is-ortho-set } B \wedge (\forall x \in B. \text{norm } x = 1) \wedge \text{ccspan } B = V$ ⟩

```

proof –

```

{
  assume  $\exists (Rep :: 't \Rightarrow 'a)$  Abs. type-definition Rep Abs (space-as-set V)
  then interpret T: local-typedef ⟨space-as-set V⟩ ⟨TYPE('t)⟩
  by unfold-locales

```

```

  note orthonormal-basis-exists
  note this[unfolded unoverload-def]
  note this[unoverload-type 'a]
  note this[unfolded unoverload-def]
  note this[where 'a='t]
  note this[untransferred]

```

```

    note this[where plus=plus and scaleC=scaleC and scaleR=scaleR and zero=0 and mi-
nus=minus
    and uminus=uminus and sgn=sgn and S=S and norm=norm and cinner=cinner and
dist=dist
    and ?open=⟨openin (top-of-set (space-as-set V))⟩
    and uniformity=⟨uniformity-on (space-as-set V)⟩]
    note this[simplified Domainp-rel-filter prod.Domainp-rel T.Domainp-cr-S]
  }
  note * = this[cancel-type-definition]
  have 1: ⟨uniformity-on (space-as-set V)
    ≤ principal (Collect (pred-prod (λx. x ∈ space-as-set V) (λx. x ∈ space-as-set V)))⟩
  by (auto simp: uniformity-dist intro!: le-infI2)
  have ⟨∃ B ∈ {A. ∀ x ∈ A. x ∈ space-as-set V}.
    S ⊆ B ∧ is-onb-ow (space-as-set V) (*C) (+) 0 norm (openin (top-of-set (space-as-set V)))
  (*C) B⟩
    apply (rule *)
    using ⟨S ⊆ space-as-set V⟩ ⟨is-ortho-set S⟩
    by (auto simp flip: unoverload-def
      intro!: complex-vector.subspace-scale real-vector.subspace-scale cspace-is-subspace
      cspace-nonempty complex-vector.subspace-add complex-vector.subspace-diff
      complex-vector.subspace-neg sgn-in-spaceI 1 norm)

  then obtain B where ⟨B ⊆ space-as-set V⟩ and ⟨S ⊆ B⟩
    and is-onb: ⟨is-onb-ow (space-as-set V) (*C) (+) 0 norm (openin (top-of-set (space-as-set
V)))⟩ (*C) B⟩
    by auto

  from ⟨B ⊆ space-as-set V⟩
  have [simp]: ⟨cspan B ∩ space-as-set V = cspan B⟩
  by (smt (verit) basic-trans-rules(8) ccspan.rep-eq ccspan-leqI ccspan-superset complex-vector.span-span
inf-absorb1 less-eq-ccsubspace.rep-eq)
  then have [simp]: ⟨space-as-set V ∩ cspan B = cspan B⟩
    by blast
  from ⟨B ⊆ space-as-set V⟩
  have [simp]: ⟨space-as-set V ∩ closure (cspan B) = closure (cspan B)⟩
  by (metis Int-absorb1 ccspan.rep-eq ccspan-leqI less-eq-ccsubspace.rep-eq)
  have [simp]: ⟨closure X ∪ X = closure X⟩ for X :: ⟨'z::topological-space set⟩
    using closure-subset by blast

  from is-onb have ⟨is-ortho-set B⟩
    by (auto simp: is-onb-ow-def unoverload-def)

  moreover from is-onb have ⟨norm x = 1⟩ if ⟨x ∈ B⟩ for x
    by (auto simp: is-onb-ow-def that)

  moreover from is-onb have ⟨closure (cspan B) = space-as-set V⟩
    by (simp add: is-onb-ow-def ⟨B ⊆ space-as-set V⟩
      closure-ow-with-typeclass span-ow-on-typeclass flip: unoverload-def)
  then have ⟨ccspan B = V⟩

```

```

by (simp add: ccspace.abs-eq space-as-set-inverse)

ultimately show ?thesis
using ‹ $S \subseteq B$ › by auto
qed

lemma has-sum-in-comm-additive-general:
  fixes f :: ‹'a  $\Rightarrow$  'b :: comm-monoid-add›
  and g :: ‹'b  $\Rightarrow$  'c :: comm-monoid-add›
  assumes T0[simp]: ‹ $0 \in \text{topspace } T$ › and Tplus[simp]: ‹ $\bigwedge x y. x \in \text{topspace } T \Longrightarrow y \in \text{topspace } T \Longrightarrow x+y \in \text{topspace } T$ ›
  assumes Uplus[simp]: ‹ $\bigwedge x y. x \in \text{topspace } U \Longrightarrow y \in \text{topspace } U \Longrightarrow x+y \in \text{topspace } U$ ›
  assumes grange: ‹ $g \text{ 'topspace } T \subseteq \text{topspace } U$ ›
  assumes g0: ‹ $g 0 = 0$ ›
  assumes frange: ‹ $f \text{ ' } S \subseteq \text{topspace } T$ ›
  assumes gcont: ‹ $\text{filterlim } g \text{ (nhdsin } U \text{ (} g \text{ l)) (atin } T \text{ l)}$ ›
  assumes gadd: ‹ $\bigwedge x y. x \in \text{topspace } T \Longrightarrow y \in \text{topspace } T \Longrightarrow g (x+y) = g x + g y$ ›
  assumes sumf: ‹ $\text{has-sum-in } T \text{ f } S \text{ l}$ ›
  shows ‹ $\text{has-sum-in } U \text{ (} g \text{ o } f \text{) } S \text{ (} g \text{ l)}$ ›
proof -
  define f' where ‹ $f' x = (\text{if } x \in S \text{ then } f x \text{ else } 0)$ › for x
  have ‹ $\text{topspace } T \neq \{\}$ ›
  using T0 by blast
  then have ‹ $\text{topspace } U \neq \{\}$ ›
  using grange by blast
  {
    assume  $\exists (Rep :: 't \Rightarrow 'b)$  Abs. type-definition Rep Abs (topspace T)
    then interpret T: local-typedef ‹topspace T› ‹TYPE('t)›
    by unfold-locales
    assume  $\exists (Rep :: 'u \Rightarrow 'c)$  Abs. type-definition Rep Abs (topspace U)
    then interpret U: local-typedef ‹topspace U› ‹TYPE('u)›
    by unfold-locales

    note [[show-types]]
    note has-sum-comm-additive-general
    note this[unfolded unoverload-def]
    note this[unoverload-type 'b, unoverload-type 'c]
    note this[where 'b='t and 'c='u and 'a='a]
    note this[unfolded unoverload-def]
    thm this[no-vars]
    note this[untransferred]
    note this[where f=g and g=f' and zero=0 and zeroa=0 and plus=plus and plusa=plus
      and ?open= $\langle \text{openin } U \rangle$  and opena= $\langle \text{openin } T \rangle$  and x=l and S=S and T= $\langle \text{topspace } T \rangle$ ]
    note this[simplified]
  }
  note * = this[cancel-type-definition, OF ‹topspace T  $\neq \{\}$ ›, cancel-type-definition, OF ‹topspace U  $\neq \{\}$ ›]

```

```

have f'T[simp]: ⟨f' x ∈ topspace T⟩ for x
  using frange f'-def by force
have [simp]: ⟨l ∈ topspace T⟩
  using sumf has-sum-in-topospace by blast
have [simp]: ⟨x ∈ topspace T ⇒ g x ∈ topspace U⟩ for x
  using grange by auto
have sumf'T: ⟨(∑ x∈F. f' x) ∈ topspace T⟩ if ⟨finite F⟩ for F
  using that apply induction
  by auto
have [simp]: ⟨(∑ x∈F. f x) ∈ topspace T⟩ if ⟨F ⊆ S⟩ for F
  using that apply (induction F rule:infinite-finite-induct)
  apply auto
  by (metis Tplus f'T f'-def)
have sum-gf: ⟨(∑ x∈F. g (f' x)) = g (∑ x∈F. f' x)⟩
  if ⟨finite F⟩ and ⟨F ⊆ S⟩ for F
proof -
  have ⟨(∑ x∈F. g (f' x)) = (∑ x∈F. g (f x))⟩
    apply (rule sum.cong)
    using frange that by (auto simp: f'-def)
  also have ⟨... = g (∑ x∈F. f x)⟩
    using ⟨finite F⟩ ⟨F ⊆ S⟩ apply induction
    using g0 frange apply auto
    apply (subst gadd)
    by (auto simp: f'-def)
  also have ⟨... = g (∑ x∈F. f' x)⟩
    apply (rule arg-cong[where f=g])
    apply (rule sum.cong)
    using that by (auto simp: f'-def)
  finally show ?thesis
    by -
qed
from sumf have sumf': ⟨has-sum-in T f' S l⟩
  apply (rule has-sum-in-cong[THEN iffD2, rotated])
  unfolding f'-def by auto
have [simp]: ⟨g l ∈ topspace U⟩
  using grange by auto
from gcont have contg': ⟨filterlim g (nhdsin U (g l)) (nhdsin T l □ principal (topospace T -
{l}))⟩
  apply (rule filterlim-cong[THEN iffD1, rotated -1])
  apply (rule refl)
  apply (simp add: atin-def)
  by (auto intro!: exI simp add: eventually-atin)
from T0 grange g0 have [simp]: ⟨0 ∈ topspace U⟩
  by auto

have [simp]:
  ⟨comm-monoid-ow (topospace T) (+) 0⟩
  ⟨comm-monoid-ow (topospace U) (+) 0⟩
  by (simp-all add: comm-monoid-ow-def abel-semigroup-ow-def

```

*semigroup-ow-def plus-ow-def semigroup-ow-axioms-def
comm-monoid-ow-axioms-def Groups.add-ac abel-semigroup-ow-axioms-def)*

have $\langle \text{has-sum-ow } (\text{topspace } U) (+) 0 (\text{openin } U) (g \circ f') S (g \ l) \rangle$
apply (rule *)
by (auto simp: topological-space-ow-from-topology sum-gf sumf'
sum-ud[symmetric] at-within-ow-topology has-sum-ow-topology
contg' sumf'T)

then have $\langle \text{has-sum-in } U (g \circ f') S (g \ l) \rangle$
apply (rule has-sum-ow-topology[THEN iffD1, rotated -1])
by simp-all
then have $\langle \text{has-sum-in } U (g \circ f') S (g \ l) \rangle$
by simp
then show ?thesis
apply (rule has-sum-in-cong[THEN iffD1, rotated])
unfolding f'-def **using** frange grange **by** auto

qed

lemma *has-sum-in-comm-additive:*
fixes $f :: \langle 'a \Rightarrow 'b :: \text{ab-group-add} \rangle$
and $g :: \langle 'b \Rightarrow 'c :: \text{ab-group-add} \rangle$
assumes $\langle \text{topspace } T = \text{UNIV} \rangle$ **and** $\langle \text{topspace } U = \text{UNIV} \rangle$
assumes $\langle \text{Modules.additive } g \rangle$
assumes $g\text{cont}: \langle \text{continuous-map } T \ U \ g \rangle$
assumes $\text{sumf}: \langle \text{has-sum-in } T \ f \ S \ l \rangle$
shows $\langle \text{has-sum-in } U (g \circ f) S (g \ l) \rangle$
apply (rule has-sum-in-comm-additive-general[where T=T and U=U])
using *assms*
by (auto simp: additive.zero Modules.additive-def intro!: continuous-map-is-continuous-at-point)

8 Stuff relying on the above lifting

definition $\langle \text{some-onb-of } X = (\text{SOME } B. \text{is-ortho-set } B \wedge (\forall b \in B. \text{norm } b = 1) \wedge \text{ccspan } B = X) \rangle$

lemma

fixes $X :: \langle 'a :: \text{hilbert-space ccspace} \rangle$
shows *some-onb-of-is-ortho-set*[iff]: $\langle \text{is-ortho-set } (\text{some-onb-of } X) \rangle$
and *some-onb-of-norm1*: $\langle b \in \text{some-onb-of } X \implies \text{norm } b = 1 \rangle$
and *some-onb-of-ccspan*[simp]: $\langle \text{ccspan } (\text{some-onb-of } X) = X \rangle$

proof –

let $?P = \langle \lambda B. \text{is-ortho-set } B \wedge (\forall b \in B. \text{norm } b = 1) \wedge \text{ccspan } B = X \rangle$
have $\langle \text{Ex } ?P \rangle$
using *orthonormal-subspace-basis-exists*[where S={ } and V=X]
by auto
then have $\langle ?P (\text{some-onb-of } X) \rangle$
by (simp add: *some-onb-of-def* *verit-sko-ex*)

```

then show is-ortho-set-some-onb-of:  $\langle \text{is-ortho-set (some-onb-of } X \rangle$ 
  and  $\langle b \in \text{some-onb-of } X \implies \text{norm } b = 1 \rangle$ 
  and  $\langle \text{ccspan (some-onb-of } X) = X \rangle$ 
  by auto
qed

lemma ccsubspace-as-whole-type:
  fixes  $X :: \langle 'a::\text{hilbert-space ccsubspace} \rangle$ 
  assumes  $\langle X \neq 0 \rangle$ 
  shows  $\langle \text{let } 'b::\text{type} = \text{some-onb-of } X \text{ in}$ 
     $\exists U::'b \text{ ell2} \Rightarrow_{CL} 'a. \text{isometry } U \wedge U *_S \top = X \rangle$ 
proof with-type-intro
  show  $\langle \text{some-onb-of } X \neq \{\} \rangle$ 
  using some-onb-of-ccspan[of X] assms
  by (auto simp del: some-onb-of-ccspan)
  fix  $\text{Rep} :: \langle 'b \Rightarrow 'a \rangle$  and  $\text{Abs}$ 
  assume  $\langle \text{bij-betw } \text{Rep } UNIV \text{ (some-onb-of } X) \rangle$ 
  then interpret type-definition Rep  $\langle \text{inv } \text{Rep} \rangle$   $\langle \text{some-onb-of } X \rangle$ 
  by (simp add: type-definition-bij-betw-iff)
  define  $U$  where  $\langle U = \text{cblinfun-extension (range ket) (Rep o inv ket)} \rangle$ 
  have [simp]:  $\langle \text{Rep } i \cdot_C \text{Rep } j = 0 \rangle$  if  $\langle i \neq j \rangle$  for  $i\ j$ 
  using Rep some-onb-of-is-ortho-set[unfolded is-ortho-set-def] that
  by (smt (verit) Rep-inverse)
  moreover have [simp]:  $\langle \text{norm (Rep } i) = 1 \rangle$  for  $i$ 
  using Rep[of i] some-onb-of-norm1
  by auto
  ultimately have  $\langle \text{cblinfun-extension-exists (range ket) (Rep o inv ket)} \rangle$ 
  apply (rule-tac cblinfun-extension-exists-ortho)
  by auto
  then have  $U\text{-ket[simp]$ :  $\langle U (\text{ket } i) = \text{Rep } i \rangle$  for  $i$ 
  by (auto simp: cblinfun-extension-apply U-def)
  have  $\langle \text{isometry } U \rangle$ 
  apply (rule orthogonal-on-basis-is-isometry[where B= $\langle \text{range ket} \rangle$ ])
  by (auto simp: cinner-ket simp flip: cnorm-eq-1)
  moreover have  $\langle U *_S \text{ccspan (range ket)} = X \rangle$ 
  apply (subst cblinfun-image-ccspan)
  by (simp add: Rep-range image-image)
  ultimately show  $\langle \exists U :: 'b \text{ ell2} \Rightarrow_{CL} 'a. \text{isometry } U \wedge U *_S \top = X \rangle$ 
  by auto
qed

lemma some-onb-of-0[simp]:  $\langle \text{some-onb-of } (0 :: 'a::\text{hilbert-space ccsubspace}) = \{\} \rangle$ 
proof –
  have no0:  $\langle 0 \notin \text{some-onb-of } (0 :: 'a \text{ ccsubspace}) \rangle$ 
  using some-onb-of-norm1
  by fastforce
  have  $\langle \text{ccspan (some-onb-of } 0) = (0 :: 'a \text{ ccsubspace}) \rangle$ 
  by simp
  then have  $\langle \text{some-onb-of } 0 \subseteq \text{space-as-set } (0 :: 'a \text{ ccsubspace}) \rangle$ 

```


by (*metis cspan-superset*)
 also have $\langle \dots = \{0\} \rangle$
 by *simp*
 finally show *?thesis*
 using *no0*
 by *blast*
qed

lemma *some-onb-of-finite-dim*:
 fixes $S :: \langle 'a::\text{chilbert-space ccspace} \rangle$
 assumes $\langle \text{finite-dim-ccspace } S \rangle$
 shows $\langle \text{finite (some-onb-of } S) \rangle$
proof –
 from *assms* obtain C where $CS: \langle \text{cspan } C = \text{space-as-set } S \rangle$ and $\langle \text{finite } C \rangle$
 by (*meson cfinite-dim-subspace-has-basis cspace-space-as-set finite-dim-ccspace.rep-eq*)
 then show $\langle \text{finite (some-onb-of } S) \rangle$
 using *cspan-superset complex-vector.independent-span-bound is-ortho-set-cindependent* by
fastforce
qed

lemma *some-onb-of-in-space[iff]*:
 fixes $S :: \langle 'a::\text{chilbert-space ccspace} \rangle$
 shows $\langle \text{some-onb-of } S \subseteq \text{space-as-set } S \rangle$
 using *cspan-superset* by *fastforce*

lemma *sum-some-onb-of-butterfly*:
 fixes $S :: \langle 'a::\text{chilbert-space ccspace} \rangle$
 assumes $\langle \text{finite-dim-ccspace } S \rangle$
 shows $\langle (\sum x \in \text{some-onb-of } S. \text{butterfly } x) = \text{Proj } S \rangle$
proof –
 obtain B where *onb-S-in-B*: $\langle \text{some-onb-of } S \subseteq B \rangle$ and $\langle \text{is-onb } B \rangle$
 apply *atomize-elim*
 apply (*rule orthonormal-basis-exists*)
 by (*simp-all add: some-onb-of-norm1*)
 have *S-ccspan*: $\langle S = \text{cspan (some-onb-of } S) \rangle$
 by *simp*

show *?thesis*

proof (*rule cblinfun-eq-gen-eqI[where G=B]*)

show $\langle \text{cspan } B = \top \rangle$

using $\langle \text{is-onb } B \rangle$ *is-onb-def* by *blast*

fix b assume $\langle b \in B \rangle$

show $\langle (\sum x \in \text{some-onb-of } S. \text{selfbutter } x) *_V b = \text{Proj } S *_V b \rangle$

proof (*cases* $\langle b \in \text{some-onb-of } S \rangle$)

case *True*

have $\langle (\sum x \in \text{some-onb-of } S. \text{selfbutter } x) *_V b = (\sum x \in \text{some-onb-of } S. \text{selfbutter } x *_V b) \rangle$

using *cblinfun.sum-left* by *blast*

```

also have ⟨... = b⟩
  apply (subst sum-single[where i=b])
  using True apply (auto intro!: simp add: assms some-onb-of-finite-dim)
  using is-ortho-set-def apply fastforce
  using cnorm-eq-1 some-onb-of-norm1 by force
also have ⟨... = Proj S *V b⟩
  apply (rule Proj-fixes-image[symmetric])
  using True some-onb-of-in-space by blast
finally show ?thesis
  by -
next
case False
have *: ⟨is-orthogonal x b⟩ if ⟨x ∈ some-onb-of S⟩ and ⟨x ≠ 0⟩ for x
proof -
  have ⟨x ∈ B⟩
    using onb-S-in-B that(1) by fastforce
  moreover note ⟨b ∈ B⟩
  moreover have ⟨x ≠ b⟩
    using False that(1) by blast
  moreover note ⟨is-onb B⟩
  ultimately show ⟨is-orthogonal x b⟩
    by (simp add: is-onb-def is-ortho-set-def)
qed
have ⟨(∑ x ∈ some-onb-of S. selfbutter x) *V b = (∑ x ∈ some-onb-of S. selfbutter x *V b)⟩
  using cblinfun.sum-left by blast
also have ⟨... = 0⟩
  by (auto intro!: sum.neutral simp: *)
also have ⟨... = Proj S *V b⟩
  apply (rule Proj-0-compl[symmetric])
  apply (subst S-ccspan)
  apply (rule mem-ortho-ccspanI)
  using * cinner-zero-right is-orthogonal-sym by blast
finally show ?thesis
  by -
qed
qed
qed

```

lemma *cdim-infinite-0*:

assumes ⟨ \neg *cfinite-dim S*⟩

shows ⟨*cdim S* = 0⟩

proof –

from *assms* **have** *not-fin-cspan*: ⟨ \neg *cfinite-dim (cspan S)*⟩

using *cfinite-dim-def cfinite-dim-subspace-has-basis complex-vector.span-superset* **by** *fastforce*

obtain *B* **where** ⟨*cindependent B*⟩ **and** ⟨*cspan S* = *cspan B*⟩

using *csubspace-has-basis* **by** *blast*

with *not-fin-cspan* **have** ⟨*infinite B*⟩

by *auto*

```

then have ⟨card B = 0⟩
  by force
have ⟨cdim (cspan S) = 0⟩
  apply (rule complex-vector.dim-unique[of B])
  apply (auto intro!: simp add: ⟨cspan S = cspan B⟩ complex-vector.span-superset)
  using ⟨cindependent B⟩ ⟨card B = 0⟩ by auto
then show ?thesis
  by simp
qed

```

lemma *some-onb-of-card*:

```

fixes S :: ⟨'a::chilbert-space ccspace⟩
shows ⟨card (some-onb-of S) = cdim (space-as-set S)⟩
proof (cases ⟨finite-dim-ccspace S⟩)
  case True
  show ?thesis
    apply (rule complex-vector.dim-eq-card[symmetric])
    apply (auto simp: is-ortho-set-cindependent)
    apply (metis True cspan-finite some-onb-of-ccspan complex-vector.span-clauses(1) some-onb-of-finite-dim)
    by (metis True cspan-finite some-onb-of-ccspan complex-vector.span-eq-iff cspace-space-as-set
some-onb-of-finite-dim)
  next
  case False
  then have ⟨cdim (space-as-set S) = 0⟩
    by (simp add: cdim-infinite-0 finite-dim-ccspace.rep-eq)
  moreover from False have ⟨infinite (some-onb-of S)⟩
    using cspan-finite-dim by fastforce
  ultimately show ?thesis
    by simp
qed

```

unbundle *no lattice-syntax* **and** *no cblinfun-syntax*

end

9 Eigenvalues – Material related to eigenvalues and eigenspaces

theory *Eigenvalues*

imports

Weak-Operator-Topology

Misc-Tensor-Product-TTS

begin

unbundle *cblinfun-syntax*

definition *normal-op* :: ⟨⟨'a::chilbert-space ⇒_{CL} 'a⟩ ⇒ bool⟩ **where**
 ⟨*normal-op* A ⟷ A o_{CL} A* = A* o_{CL} A⟩

definition *eigenvalues* :: $\langle 'a :: \text{complex-normed-vector} \Rightarrow_{CL} 'a \rangle \Rightarrow \text{complex set}$ **where**
 $\langle \text{eigenvalues } a = \{x. \text{eigenspace } x \ a \neq 0\} \rangle$

definition *invariant-subspace* :: $\langle 'a :: \text{complex-inner ccspace} \Rightarrow ('a \Rightarrow_{CL} 'a) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{invariant-subspace } S \ A \longleftrightarrow A *_S S \leq S \rangle$

lemma *invariant-subspaceI*: $\langle A *_S S \leq S \Longrightarrow \text{invariant-subspace } S \ A \rangle$
by (*simp add: invariant-subspace-def*)

definition *reducing-subspace* :: $\langle 'a :: \text{complex-inner ccspace} \Rightarrow ('a \Rightarrow_{CL} 'a) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{reducing-subspace } S \ A \longleftrightarrow \text{invariant-subspace } S \ A \wedge \text{invariant-subspace } (-S) \ A \rangle$

lemma *reducing-subspaceI*: $\langle A *_S S \leq S \Longrightarrow A *_S (-S) \leq -S \Longrightarrow \text{reducing-subspace } S \ A \rangle$
by (*simp add: reducing-subspace-def invariant-subspace-def*)

lemma *reducing-subspace-ortho*[*simp*]: $\langle \text{reducing-subspace } (-S) \ A \longleftrightarrow \text{reducing-subspace } S \ A \rangle$
for $S :: \langle 'a :: \text{chilbert-space ccspace} \rangle$
by (*auto simp: reducing-subspace-def*)

lemma *invariant-subspace-bot*[*simp*]: $\langle \text{invariant-subspace } \perp \ A \rangle$
by (*simp add: invariant-subspaceI*)

lemma *invariant-subspace-top*[*simp*]: $\langle \text{invariant-subspace } \top \ A \rangle$
by (*simp add: invariant-subspaceI*)

lemma *reducing-subspace-bot*[*simp*]: $\langle \text{reducing-subspace } \perp \ A \rangle$
by (*metis cblinfun-image-bot eq_refl orthogonal-bot orthogonal-spaces-leq-compl reducing-subspaceI*)

lemma *reducing-subspace-top*[*simp*]: $\langle \text{reducing-subspace } \top \ A \rangle$
by (*simp add: reducing-subspace-def*)

lemma *kernel-uminus*[*simp*]: $\text{kernel } (-A) = \text{kernel } A$
for $a :: \text{complex}$ **and** $A :: (-,-) \text{ cblinfun}$
by *transfer auto*

lemma *kernel-scaleC'*: $\text{kernel } (a *_C A) = (\text{if } a = 0 \text{ then } \top \text{ else } \text{kernel } A)$
for $a :: \text{complex}$ **and** $A :: (-,-) \text{ cblinfun}$
by (*cases a = 0*) *auto*

lemma *eigenvalues-0*[*simp*]: $\langle \text{eigenvalues } (0 :: 'a :: \{\text{not-singleton, complex-normed-vector}\} \Rightarrow_{CL} 'a) = \{0\} \rangle$
by (*auto simp: eigenvalues-def eigenspace-def kernel-scaleC'*)

lemma *nonzero-ccspace-contains-unit-vector*:
assumes $\langle S \neq 0 \rangle$
shows $\langle \exists \psi. \psi \in \text{space-as-set } S \wedge \text{norm } \psi = 1 \rangle$
proof –

```

from assms
obtain  $\psi$  where  $\langle \psi \in \text{space-as-set } S \rangle \langle \psi \neq 0 \rangle$ 
  by transfer (auto simp: complex-vector.subspace-0)
have  $\langle \text{sgn } \psi \in \text{space-as-set } S \rangle$ 
  using  $\psi$  by (simp add: complex-vector.subspace-scale scaleR-scaleC sgn-div-norm)
moreover have  $\langle \text{norm } (\text{sgn } \psi) = 1 \rangle$ 
  by (simp add:  $\langle \psi \neq 0 \rangle$  norm-sgn)
ultimately show ?thesis
  by auto
qed

```

```

lemma unit-eigenvector-ex:
  assumes  $\langle x \in \text{eigenvalues } a \rangle$ 
  shows  $\langle \exists h. \text{norm } h = 1 \wedge a h = x *_C h \rangle$ 
proof –
  from assms have  $\langle \text{eigenspace } x a \neq 0 \rangle$ 
    by (simp add: eigenvalues-def)
  then obtain  $\psi$  where  $\psi$ -ev:  $\langle \psi \in \text{space-as-set } (\text{eigenspace } x a) \rangle$  and  $\langle \psi \neq 0 \rangle$ 
    using nonzero-ccsubspace-contains-unit-vector by force
  define  $h$  where  $\langle h = \text{sgn } \psi \rangle$ 
  with  $\langle \psi \neq 0 \rangle$  have  $\langle \text{norm } h = 1 \rangle$ 
    by (simp add: norm-sgn)
  from  $\psi$ -ev have  $\langle h \in \text{space-as-set } (\text{eigenspace } x a) \rangle$ 
    by (simp add: h-def sgn-in-spaceI)
  then have  $\langle a *_V h = x *_C h \rangle$ 
    unfolding eigenspace-def
    by (transfer' fixing: x) simp
  with  $\langle \text{norm } h = 1 \rangle$  show ?thesis
    by auto
qed

```

```

lemma eigenvalue-norm-bound:
  assumes  $\langle e \in \text{eigenvalues } a \rangle$ 
  shows  $\langle \text{norm } e \leq \text{norm } a \rangle$ 
proof –
  from assms obtain  $h$  where  $\langle \text{norm } h = 1 \rangle$  and  $ah$ -eh:  $\langle a h = e *_C h \rangle$ 
    using unit-eigenvector-ex by blast
  have  $\langle \text{cmod } e = \text{norm } (e *_C h) \rangle$ 
    by (simp add:  $\langle \text{norm } h = 1 \rangle$ )
  also have  $\langle \dots = \text{norm } (a h) \rangle$ 
    using  $ah$ -eh by presburger
  also have  $\langle \dots \leq \text{norm } a \rangle$ 
    by (metis  $\langle \text{norm } h = 1 \rangle$  cblinfun.real.bounded-linear-right mult-cancel-left1 norm-cblinfun.rep-eq onorm)
  finally show  $\langle \text{cmod } e \leq \text{norm } a \rangle$ 
    by –
qed

```

```

lemma eigenvalue-selfadj-real:
  assumes ⟨e ∈ eigenvalues a⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨e ∈ ℝ⟩
proof -
  from assms obtain h where ⟨norm h = 1⟩ and ah-eh: ⟨a h = e *C h⟩
  using unit-eigenvector-ex by blast
  have ⟨e = h •C (e *C h)⟩
  by (metis ⟨norm h = 1⟩ cinner-simps(6) mult-cancel-left1 norm-one one-cinner-one power2-norm-eq-cinner
power2-norm-eq-cinner)
  also have ⟨... = h •C a h⟩
  by (simp add: ah-eh)
  also from assms(2) have ⟨... ∈ ℝ⟩
  using cinner-selfadjoint-real selfadjoint-def by blast
  finally show ⟨e ∈ ℝ⟩
  by -
qed

```

```

lemma is-Sup-imp-ex-tendsto:
  fixes X :: ⟨'a::{\linorder-topology,first-countable-topology} set⟩
  assumes sup: ⟨is-Sup X l⟩
  assumes ⟨X ≠ {}⟩
  shows ⟨∃f. range f ⊆ X ∧ f ⟶ l⟩
proof (cases ⟨∃x. x < l⟩)
  case True
  obtain A :: ⟨nat ⇒ 'a set⟩ where openA: ⟨open (A n)⟩ and lA: ⟨l ∈ A n⟩
  and fl: ⟨(∧n. f n ∈ A n) ⟹ f ⟶ l⟩ for n f
  by (rule Topological-Spaces.countable-basis[of l]) blast
  obtain f where fAX: ⟨f n ∈ A n ∩ X⟩ for n
  proof (atomize-elim, intro choice allI)
    fix n :: nat
    from True obtain x where ⟨x < l⟩
    by blast
    from open-left[OF openA lA this]
    obtain b where ⟨b < l⟩ and bl-A: ⟨{b<..l} ⊆ A n⟩
    by blast
    from sup ⟨b < l⟩ obtain x where ⟨x ∈ X⟩ and ⟨x > b⟩
    by (meson is-Sup-def leD leI)
    from ⟨x ∈ X⟩ sup have ⟨x ≤ l⟩
    by (simp add: is-Sup-def)
    from ⟨x ≤ l⟩ and ⟨x > b⟩ and bl-A
    have ⟨x ∈ A n⟩
    by fastforce
    with ⟨x ∈ X⟩
    show ⟨∃x. x ∈ A n ∩ X⟩
    by blast
  qed
  with fl have ⟨f ⟶ l⟩
  by auto

```

```

moreover from fAX have ⟨range f ⊆ X⟩
  by auto
ultimately show ?thesis
  by blast
next
case False
from ⟨X ≠ {}⟩ obtain x where ⟨x ∈ X⟩
  by blast
with ⟨is-Sup X l⟩ have ⟨x ≤ l⟩
  by (simp add: is-Sup-def)
with False have ⟨x = l⟩
  using basic-trans-rules(17) by auto
with ⟨x ∈ X⟩ have ⟨l ∈ X⟩
  by simp
define f where ⟨f n = l⟩ for n :: nat
then have ⟨f ⟶ l⟩
  by (auto intro!: simp: f-def[abs-def])
moreover from ⟨l ∈ X⟩ have ⟨range f ⊆ X⟩
  by (simp add: f-def)
ultimately show ?thesis
  by blast
qed

```

```

lemma eigenvaluesI:
  assumes ⟨A *V h = e *C h⟩
  assumes ⟨h ≠ 0⟩
  shows ⟨e ∈ eigenvalues A⟩
proof -
  from assms have ⟨h ∈ space-as-set (eigenspace e A)⟩
    by (simp add: eigenspace-def kernel.rep-eq cblinfun.diff-left)
  moreover from ⟨h ≠ 0⟩ have ⟨h ∉ space-as-set ⊥⟩
    by transfer simp
  ultimately have ⟨eigenspace e A ≠ ⊥⟩
    by fastforce
  then show ?thesis
    by (simp add: eigenvalues-def)
qed

```

```

lemma tendsto-diff-const-left-rewrite:
  fixes c d :: ⟨'a::topological-group-add, ab-group-add⟩
  assumes ⟨((λx. f x) ⟶ c - d) F⟩
  shows ⟨((λx. c - f x) ⟶ d) F⟩
  by (auto intro!: assms tendsto-eq-intros)

```

```

lemma not-not-singleton-no-eigenvalues:
  fixes a :: ⟨'a::complex-normed-vector ⇒CL 'a⟩
  assumes ⟨¬ class.not-singleton TYPE('a)⟩
  shows ⟨eigenvalues a = {}⟩
proof (rule equals0I)

```

fix e **assume** $\langle e \in \text{eigenvalues } a \rangle$
then have $\langle \text{eigenspace } e \ a \neq \perp \rangle$
by (*simp add: eigenvalues-def*)
then obtain h **where** $\langle \text{norm } h = 1 \rangle$ **and** $\langle h \in \text{space-as-set } (\text{eigenspace } e \ a) \rangle$
using *nonzero-ccsubspace-contains-unit-vector* **by** *auto*
from *assms* **have** $\langle h = 0 \rangle$
by (*rule not-not-singleton-zero*)
with $\langle \text{norm } h = 1 \rangle$
show *False*
by *simp*
qed

lemma *cblinfun-cinner-eq0I*:
fixes $a :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \bigwedge h. h \cdot_C a \ h = 0 \rangle$
shows $\langle a = 0 \rangle$
by (*rule cblinfun-cinner-eqI*) (*use assms in simp*)

lemma *normal-op-iff-adj-same-norms*:

— [2], Proposition II.2.16

fixes $a :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'a \rangle$

shows $\langle \text{normal-op } a \iff (\forall h. \text{norm } (a \ h) = \text{norm } ((a^*) \ h)) \rangle$

proof —

have *aux*: $\langle (\bigwedge h. a \ h = b \ h) \implies (\forall h. a \ h = (0 :: \text{complex})) \iff (\forall h. b \ h = (0 :: \text{real})) \rangle$ **for** a
 $:: \langle 'a \Rightarrow \text{complex} \rangle$ **and** $b :: \langle 'a \Rightarrow \text{real} \rangle$

by *simp*

have $\langle \text{normal-op } a \iff (a^* \ o_{CL} \ a) - (a \ o_{CL} \ a^*) = 0 \rangle$

using *normal-op-def* **by** *force*

also have $\langle \dots \iff (\forall h. h \cdot_C ((a^* \ o_{CL} \ a) - (a \ o_{CL} \ a^*)) \ h = 0) \rangle$

by (*auto intro!*: *cblinfun-cinner-eqI* *simp*: *cblinfun.diff-left cinner-diff-right*
simp flip: *cblinfun-apply-cblinfun-compose*)

also have $\langle \dots \iff (\forall h. (\text{norm } (a \ h))^2 - (\text{norm } ((a^*) \ h))^2 = 0) \rangle$

proof (*rule aux*)

fix h

have $\langle (\text{norm } (a \ *_V \ h))^2 - (\text{norm } (a^* \ *_V \ h))^2$

$= (a \ *_V \ h) \cdot_C (a \ *_V \ h) - (a^* \ *_V \ h) \cdot_C (a^* \ *_V \ h) \rangle$

by (*simp add*: *of-real-diff flip*: *cdot-square-norm of-real-power*)

also have $\langle \dots = h \cdot_C ((a^* \ o_{CL} \ a) - (a \ o_{CL} \ a^*)) \ h \rangle$

by (*simp add*: *cblinfun.diff-left cinner-diff-right cinner-adj-left*
cinner-adj-right flip: *cinner-adj-left*)

finally show $\langle h \cdot_C ((a^* \ o_{CL} \ a) - (a \ o_{CL} \ a^*)) \ h = (\text{norm } (a \ *_V \ h))^2 - (\text{norm } (a^* \ *_V \ h))^2 \rangle$

by *simp*

qed

also have $\langle \dots \iff (\forall h. \text{norm } (a \ h) = \text{norm } ((a^*) \ h)) \rangle$

by *simp*

finally show *?thesis*.

qed

lemma *normal-op-same-eigenspace-as-adj*:
— Shown inside the proof of [2, Proposition II.5.6]
assumes $\langle \text{normal-op } a \rangle$
shows $\langle \text{eigenspace } l \ a = \text{eigenspace } (\text{cnj } l) \ (a^*) \rangle$
proof –
from $\langle \text{normal-op } a \rangle$
have $\langle \text{normal-op } (a - l *_C \text{id-cblinfun}) \rangle$
by (*auto intro!*; *simp: normal-op-def cblinfun-compose-minus-left cblinfun-compose-minus-right adj-minus scaleC-diff-right*)
then have $\ast: \langle \text{norm } ((a - l *_C \text{id-cblinfun}) \ h) = \text{norm } (((a - l *_C \text{id-cblinfun}) \ast) \ h) \rangle$ **for** h
using *normal-op-iff-adj-same-norms* **by** *blast*
show *?thesis*
proof (*rule ccspace-eqI*)
fix h
have $\langle h \in \text{space-as-set } (\text{eigenspace } l \ a) \longleftrightarrow \text{norm } ((a - l *_C \text{id-cblinfun}) \ h) = 0 \rangle$
by (*simp add: eigenspace-def kernel-member-iff*)
also have $\langle \dots \longleftrightarrow \text{norm } (((a^*) - \text{cnj } l *_C \text{id-cblinfun}) \ h) = 0 \rangle$
by (*simp add: \ast adj-minus*)
also have $\langle \dots \longleftrightarrow h \in \text{space-as-set } (\text{eigenspace } (\text{cnj } l) \ (a^*)) \rangle$
by (*simp add: eigenspace-def kernel-member-iff*)
finally show $\langle h \in \text{space-as-set } (\text{eigenspace } l \ a) \longleftrightarrow h \in \text{space-as-set } (\text{eigenspace } (\text{cnj } l) \ (a^*)) \rangle$.
qed
qed

lemma *normal-op-adj-eigenvalues*:
assumes $\langle \text{normal-op } a \rangle$
shows $\langle \text{eigenvalues } (a^*) = \text{cnj } \text{'eigenvalues } a \rangle$
by (*auto intro!: complex-cnj-cnj[symmetric] image-eqI*
simp: eigenvalues-def assms normal-op-same-eigenspace-as-adj)

lemma *invariant-subspace-iff-PAP*:
— [2], Proposition II.3.7 (b)
 $\langle \text{invariant-subspace } S \ A \longleftrightarrow \text{Proj } S \ o_{CL} \ A \ o_{CL} \ \text{Proj } S = A \ o_{CL} \ \text{Proj } S \rangle$
proof –
define S' **where** $\langle S' = \text{space-as-set } S \rangle$
have $\langle \text{invariant-subspace } S \ A \longleftrightarrow (\forall h \in S'. \ A \ h \in S') \rangle$
proof *safe*
fix h **assume** $A: \text{invariant-subspace } S \ A$ **and** $h: h \in S'$
from h **have** $A *_V \ h \in \text{space-as-set } (A *_S \ S)$
using *cblinfun-apply-in-image* [of $h \ S \ A$] **unfolding** S' -*def* **by** *auto*
also have $\text{space-as-set } (A *_S \ S) \subseteq S'$
using A **unfolding** S' -*def* *invariant-subspace-def less-eq-ccspace-def* **by** *auto*
finally show $A *_V \ h \in S'$.
next
assume $\ast: \forall h \in S'. \ A *_V \ h \in S'$
hence $A *_S \ S \leq S$
unfolding S' -*def* **using** *cblinfun-image-less-eqI* **by** *blast*
thus *invariant-subspace* $S \ A$

unfolding *invariant-subspace-def less-eq-ccsubspace-def map-fun-def o-def id-def* .
qed
also have $\langle \dots \longleftrightarrow (\forall h. A *_V \text{Proj } S *_V h \in S') \rangle$
by (*metis (no-types, lifting) Proj-fixes-image Proj-range S'-def cblinfun-apply-in-image*)
also have $\langle \dots \longleftrightarrow (\forall h. \text{Proj } S *_V A *_V \text{Proj } S *_V h = A *_V \text{Proj } S *_V h) \rangle$
using *Proj-fixes-image S'-def space-as-setI-via-Proj* **by** *blast*
also have $\langle \dots \longleftrightarrow \text{Proj } S \text{ } o_{CL} A \text{ } o_{CL} \text{Proj } S = A \text{ } o_{CL} \text{Proj } S \rangle$
by (*auto intro!: cblinfun-eqI simp:*
simp flip: cblinfun-apply-cblinfun-compose cblinfun-compose-assoc)
finally show *?thesis*
by –
qed

lemma *reducing-iff-PA:*
– [2], Proposition II.3.7 (e)
 $\langle \text{reducing-subspace } S A \longleftrightarrow \text{Proj } S \text{ } o_{CL} A = A \text{ } o_{CL} \text{Proj } S \rangle$
proof (*rule iffI*)
assume *red:* $\langle \text{reducing-subspace } S A \rangle$
define *P* **where** $\langle P = \text{Proj } S \rangle$
from *red* **have** *AP:* $\langle P \text{ } o_{CL} A \text{ } o_{CL} P = A \text{ } o_{CL} P \rangle$
by (*simp add: invariant-subspace-iff-PAP reducing-subspace-def P-def*)
from *red* **have** $\langle \text{reducing-subspace } (- S) A \rangle$
by *simp*
then have $\langle (id\text{-cblinfun } - P) \text{ } o_{CL} A \text{ } o_{CL} (id\text{-cblinfun } - P) = A \text{ } o_{CL} (id\text{-cblinfun } - P) \rangle$
using *invariant-subspace-iff-PAP[of <- S>] reducing-subspace-def P-def Proj-ortho-compl*
by *metis*
then have $\langle P \text{ } o_{CL} A = P \text{ } o_{CL} A \text{ } o_{CL} P \rangle$
by (*simp add: cblinfun-compose-minus-left cblinfun-compose-minus-right*)
with *AP* **show** $\langle P \text{ } o_{CL} A = A \text{ } o_{CL} P \rangle$
by *simp*
next
define *P* **where** $\langle P = \text{Proj } S \rangle$
assume $\langle P \text{ } o_{CL} A = A \text{ } o_{CL} P \rangle$
then have $\langle P \text{ } o_{CL} A \text{ } o_{CL} P = A \text{ } o_{CL} P \text{ } o_{CL} P \rangle$
by *simp*
then have $\langle P \text{ } o_{CL} A \text{ } o_{CL} P = A \text{ } o_{CL} P \rangle$
by (*metis P-def Proj-idempotent cblinfun-assoc-left(1)*)
then have $\langle \text{invariant-subspace } S A \rangle$
by (*simp add: P-def invariant-subspace-iff-PAP*)
have $\langle (id\text{-cblinfun } - P) \text{ } o_{CL} A \text{ } o_{CL} (id\text{-cblinfun } - P) = A \text{ } o_{CL} (id\text{-cblinfun } - P) \rangle$
by (*metis (no-types, opaque-lifting) P-def Proj-idempotent Proj-ortho-compl <P o_{CL} A = A o_{CL} P> cblinfun-assoc-left(1) cblinfun-compose-id-left cblinfun-compose-minus-left cblinfun-compose-minus-right*)
then have $\langle \text{invariant-subspace } (- S) A \rangle$
by (*simp add: P-def Proj-ortho-compl invariant-subspace-iff-PAP*)
with $\langle \text{invariant-subspace } S A \rangle$
show $\langle \text{reducing-subspace } S A \rangle$
using *reducing-subspace-def* **by** *blast*
qed

lemma *reducing-iff-also-adj-invariant*:
— [2], Proposition II.3.7 (g)
shows $\langle \text{reducing-subspace } S \ A \longleftrightarrow \text{invariant-subspace } S \ A \wedge \text{invariant-subspace } S \ (A^*) \rangle$
proof (*intro iffI conjI; (erule conjE)?*)
assume $\langle \text{invariant-subspace } S \ A \rangle$ **and** $\langle \text{invariant-subspace } S \ (A^*) \rangle$
have $\langle \text{invariant-subspace } (- \ S) \ A \rangle$
proof (*intro invariant-subspaceI cblinfun-image-less-eqI*)
fix h **assume** $\langle h \in \text{space-as-set } (- \ S) \rangle$
show $\langle A \ *_V \ h \in \text{space-as-set } (- \ S) \rangle$
proof (*unfold uminus-ccsubspace.rep-eq, intro orthogonal-complementI*)
fix g **assume** $\langle g \in \text{space-as-set } S \rangle$
with $\langle \text{invariant-subspace } S \ (A^*) \rangle$ **have** $\langle (A^*) \ g \in \text{space-as-set } S \rangle$
by (*metis Proj-compose-cancellI Proj-range cblinfun-apply-in-image' cblinfun-fixes-range*
invariant-subspace-def space-as-setI-via-Proj)
have $\langle A \ h \cdot_C \ g = h \cdot_C \ (A^*) \ g \rangle$
by (*simp add: cinner-adj-right*)
also from $\langle (A^*) \ g \in \text{space-as-set } S \rangle$ **and** $\langle h \in \text{space-as-set } (- \ S) \rangle$
have $\langle \dots = 0 \rangle$
using *orthogonal-spaces-def orthogonal-spaces-leq-compl* **by** *blast*
finally show $\langle A \ h \cdot_C \ g = 0 \rangle$
by *blast*
qed
qed
with $\langle \text{invariant-subspace } S \ A \rangle$
show $\langle \text{reducing-subspace } S \ A \rangle$
using *reducing-subspace-def* **by** *blast*
next
assume $\langle \text{reducing-subspace } S \ A \rangle$
then show $\langle \text{invariant-subspace } S \ A \rangle$
using *reducing-subspace-def* **by** *blast*
show $\langle \text{invariant-subspace } S \ (A^*) \rangle$
by (*metis* $\langle \text{reducing-subspace } S \ A \rangle$ *adj-Proj adj-cblinfun-compose reducing-iff-PA reducing-subspace-def*)
qed

lemma *eigenspace-is-reducing*:
— [2], Proposition II.5.6
assumes $\langle \text{normal-op } a \rangle$
shows $\langle \text{reducing-subspace } (\text{eigenspace } l \ a) \ a \rangle$
proof (*unfold reducing-iff-also-adj-invariant invariant-subspace-def,*
intro conjI cblinfun-image-less-eqI subsetI)
fix h
assume $h\text{-eigen: } \langle h \in \text{space-as-set } (\text{eigenspace } l \ a) \rangle$
then have $\langle a \ h = l \ *_C \ h \rangle$
by (*simp add: eigenspace-memberD*)
also have $\langle \dots \in \text{space-as-set } (\text{eigenspace } l \ a) \rangle$
by (*simp add: Proj-fixes-image cblinfun.scaleC-right h-eigen space-as-setI-via-Proj*)
finally show $\langle a \ h \in \text{space-as-set } (\text{eigenspace } l \ a) \rangle$.

next
fix h
assume h -eigen: $\langle h \in \text{space-as-set} (\text{eigenspace } l \ a) \rangle$
then have $\langle h \in \text{space-as-set} (\text{eigenspace } (cnj \ l) \ (a*)) \rangle$
by (*simp add: assms normal-op-same-eigenspace-as-adj*)
then have $\langle (a*) \ h = cnj \ l \ *_C \ h \rangle$
by (*simp add: eigenspace-memberD*)
also have $\langle \dots \in \text{space-as-set} (\text{eigenspace } l \ a) \rangle$
by (*simp add: Proj-fixes-image cblinfun.scaleC-right h-eigen space-as-setI-via-Proj*)
finally show $\langle (a*) \ h \in \text{space-as-set} (\text{eigenspace } l \ a) \rangle$.
qed

lemma invariant-subspace-Inf:
assumes $\langle \bigwedge S. S \in M \implies \text{invariant-subspace } S \ a \rangle$
shows $\langle \text{invariant-subspace} (\bigcap M) \ a \rangle$
proof (*rule invariant-subspaceI*)
have $\langle a \ *_S \ \bigcap M \leq (\bigcap S \in M. a \ *_S \ S) \rangle$
using *cblinfun-image-INF-leq[where U=a and V=id and X=M]* **by** *simp*
also have $\langle \dots \leq (\bigcap S \in M. S) \rangle$
by (*rule INF-superset-mono, simp*) (*use assms in <auto simp: invariant-subspace-def>*)
also have $\langle \dots = \bigcap M \rangle$
by *simp*
finally show $\langle a \ *_S \ \bigcap M \leq \bigcap M \rangle$.
qed

lemma invariant-subspace-INF:
assumes $\langle \bigwedge x. x \in X \implies \text{invariant-subspace} (S \ x) \ a \rangle$
shows $\langle \text{invariant-subspace} (\bigcap x \in X. S \ x) \ a \rangle$
by (*smt (verit) assms imageE invariant-subspace-Inf*)

lemma invariant-subspace-Sup:
assumes $\langle \bigwedge S. S \in M \implies \text{invariant-subspace } S \ a \rangle$
shows $\langle \text{invariant-subspace} (\bigcup M) \ a \rangle$
proof –
have $\ast: \langle a \ ' \ \text{cspan} (\bigcup S \in M. \text{space-as-set } S) \subseteq \text{space-as-set} (\bigcup M) \rangle$
proof (*rule image-subsetI*)
fix h
assume $\langle h \in \text{cspan} (\bigcup S \in M. \text{space-as-set } S) \rangle$
then obtain $F \ r$ **where** $\langle h = (\sum g \in F. r \ g \ *_C \ g) \rangle$ **and** F -in-union: $\langle F \subseteq (\bigcup S \in M. \text{space-as-set } S) \rangle$
by (*auto intro!: simp: complex-vector.span-explicit*)
then have $\langle a \ h = (\sum g \in F. r \ g \ *_C \ a \ g) \rangle$
by (*simp add: cblinfun.scaleC-right cblinfun.sum-right*)
also have $\langle \dots \in \text{space-as-set} (\bigcup M) \rangle$
proof (*rule complex-vector.subspace-sum*)
show $\langle \text{csubspace} (\text{space-as-set} (\bigcup M)) \rangle$
by *simp*
fix g **assume** $\langle g \in F \rangle$
then obtain S **where** $\langle S \in M \rangle$ **and** $\langle g \in \text{space-as-set } S \rangle$

```

    using F-in-union by auto
  from assms[OF  $\langle S \in M \rangle$ ]  $\langle g \in \text{space-as-set } S \rangle$ 
  have  $\langle a \ g \in \text{space-as-set } S \rangle$ 
  by (simp add: Set.basic-monos(7) cblinfun-apply-in-image' invariant-subspace-def less-eq-ccsubspace.rep-eq)
  also from  $\langle S \in M \rangle$  have  $\langle \dots \subseteq \text{space-as-set } (\bigsqcup M) \rangle$ 
    by (meson Sup-upper less-eq-ccsubspace.rep-eq)
  finally show  $\langle r \ g \ *_C (a \ g) \in \text{space-as-set } (\bigsqcup M) \rangle$ 
    by (simp add: complex-vector.subspace-scale)
  qed
  finally show  $\langle a \ h \in \text{space-as-set } (\bigsqcup M) \rangle$ .
  qed
  have  $\langle \text{space-as-set } (a \ *_S \bigsqcup M) = \text{closure } (a \ ' \text{closure } (\text{cspan } (\bigsqcup S \in M. \text{space-as-set } S))) \rangle$ 
    by (metis Sup-ccsubspace.rep-eq cblinfun-image.rep-eq)
  also have  $\langle \dots = \text{closure } (a \ ' \text{cspan } (\bigsqcup S \in M. \text{space-as-set } S)) \rangle$ 
    by (rule closure-bounded-linear-image-subset-eq)
    (simp add: cblinfun.real.bounded-linear-right)
  also from * have  $\langle \dots \subseteq \text{closure } (\text{space-as-set } (\bigsqcup M)) \rangle$ 
    by (meson closure-mono)
  also have  $\langle \dots = \text{space-as-set } (\bigsqcup M) \rangle$ 
    by force
  finally have  $\langle a \ *_S \bigsqcup M \leq \bigsqcup M \rangle$ 
    by (simp add: less-eq-ccsubspace.rep-eq)
  then show ?thesis
    using invariant-subspaceI by blast
  qed

```

lemma *invariant-subspace-SUP*:

```

  assumes  $\langle \bigwedge x. x \in X \implies \text{invariant-subspace } (S \ x) \ a \rangle$ 
  shows  $\langle \text{invariant-subspace } (\bigsqcup x \in X. S \ x) \ a \rangle$ 
  by (metis (mono-tags, lifting) assms imageE invariant-subspace-Sup)

```

lemma *reducing-subspace-Inf*:

```

  fixes  $a :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'a \rangle$ 
  assumes  $\langle \bigwedge S. S \in M \implies \text{reducing-subspace } S \ a \rangle$ 
  shows  $\langle \text{reducing-subspace } (\prod M) \ a \rangle$ 
  using assms
  by (auto intro!: invariant-subspace-Inf invariant-subspace-SUP
    simp add: reducing-subspace-def uminus-Inf invariant-subspace-Inf)

```

lemma *reducing-subspace-INF*:

```

  fixes  $a :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'a \rangle$ 
  assumes  $\langle \bigwedge x. x \in X \implies \text{reducing-subspace } (S \ x) \ a \rangle$ 
  shows  $\langle \text{reducing-subspace } (\prod x \in X. S \ x) \ a \rangle$ 
  by (metis (mono-tags, lifting) assms imageE reducing-subspace-Inf)

```

lemma *reducing-subspace-Sup*:

```

  fixes  $a :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'a \rangle$ 
  assumes  $\langle \bigwedge S. S \in M \implies \text{reducing-subspace } S \ a \rangle$ 
  shows  $\langle \text{reducing-subspace } (\bigsqcup M) \ a \rangle$ 

```

using *assms*
by (*auto intro: invariant-subspace-Sup invariant-subspace-INF*
simp add: reducing-subspace-def uminus-Sup invariant-subspace-Inf)

lemma *reducing-subspace-SUP*:
fixes $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \bigwedge x. x \in X \implies \text{reducing-subspace} (S x) a \rangle$
shows $\langle \text{reducing-subspace} (\bigsqcup_{x \in X}. S x) a \rangle$
by (*metis (mono-tags, lifting) assms imageE reducing-subspace-Sup*)

lemma *selfadjoint-imp-normal*: $\langle \text{normal-op } a \rangle$ **if** $\langle \text{selfadjoint } a \rangle$
using that by (*simp add: selfadjoint-def normal-op-def*)

lemma *eigenspaces-orthogonal*:
— [2], Proposition II.5.7
assumes $\langle e \neq f \rangle$
assumes $\langle \text{normal-op } a \rangle$
shows $\langle \text{orthogonal-spaces} (\text{eigenspace } e a) (\text{eigenspace } f a) \rangle$
proof (*intro orthogonal-spaces-def [THEN iffD2] ballI*)
fix $g h$ **assume** $g\text{-eigen}: \langle g \in \text{space-as-set} (\text{eigenspace } e a) \rangle$ **and** $h\text{-eigen}: \langle h \in \text{space-as-set} (\text{eigenspace } f a) \rangle$
with $\langle \text{normal-op } a \rangle$ **have** $\langle g \in \text{space-as-set} (\text{eigenspace } (\text{cnj } e) (a*)) \rangle$
by (*simp add: normal-op-same-eigenspace-as-adj*)
then have $a\text{-adj-}g: \langle (a*) g = \text{cnj } e *_C g \rangle$
using *eigenspace-memberD* **by** *blast*
from $h\text{-eigen}$ **have** $a\text{-}h: \langle a h = f *_C h \rangle$
by (*simp add: eigenspace-memberD*)
have $\langle f * (g *_C h) = g *_C a h \rangle$
by (*simp add: a-h*)
also have $\langle \dots = (a*) g *_C h \rangle$
by (*simp add: cinner-adj-left*)
also have $\langle \dots = e * (g *_C h) \rangle$
using $a\text{-adj-}g$ **by** *auto*
finally have $\langle (f - e) * (g *_C h) = 0 \rangle$
by (*simp add: vector-space-over-itself.scale-left-diff-distrib*)
with $\langle e \neq f \rangle$ **show** $\langle g *_C h = 0 \rangle$
by *simp*
qed

definition *largest-eigenvalue* :: $\langle ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'a) \Rightarrow \text{complex} \rangle$ **where**
 $\langle \text{largest-eigenvalue } a =$
(if $\exists x. x \in \text{eigenvalues } a \wedge (\forall y \in \text{eigenvalues } a. \text{cmod } x \geq \text{cmod } y)$ then
SOME $x. x \in \text{eigenvalues } a \wedge (\forall y \in \text{eigenvalues } a. \text{cmod } x \geq \text{cmod } y)$ else 0) \rangle

lemma *largest-eigenvalue-0-aux*:
 $\langle \text{largest-eigenvalue} (0 :: 'a::\{\text{not-singleton, complex-normed-vector}\} \Rightarrow_{CL} 'a) = 0 \rangle$
proof —

```

let ?zero = ⟨0 :: 'a ⇒CL 'a⟩
define e where ⟨e = (SOME x. x ∈ eigenvalues ?zero ∧ (∀ y ∈ eigenvalues ?zero. cmod x ≥
cmod y))⟩
have ⟨∃ e. e ∈ eigenvalues ?zero ∧ (∀ y ∈ eigenvalues ?zero. cmod y ≤ cmod e)⟩ (is ⟨∃ e. ?P e⟩)
  by (auto intro!: exI[of - 0])
then have ⟨?P e⟩
  unfolding e-def
  by (rule someI-ex)
then have ⟨e = 0⟩
  by simp
then show ⟨largest-eigenvalue ?zero = 0⟩
  by (simp add: largest-eigenvalue-def)
qed

```

```

lemma largest-eigenvalue-0[simp]:
  ⟨largest-eigenvalue (0 :: 'a::complex-normed-vector ⇒CL 'a) = 0⟩
proof (cases ⟨class.not-singleton TYPE('a)⟩)
  case True
  show ?thesis
    using complex-normed-vector-axioms True
    by (rule largest-eigenvalue-0-aux[internalize-sort' 'a])
  next
  case False
  then have ⟨eigenvalues (0 :: 'a::complex-normed-vector ⇒CL 'a) = {}⟩
    by (rule not-not-singleton-no-eigenvalues)
  then show ?thesis
    by (auto simp add: largest-eigenvalue-def)
qed

```

hide-fact largest-eigenvalue-0-aux

```

lemma eigenvalues-nonneg:
  assumes ⟨a ≥ 0⟩ and ⟨v ∈ eigenvalues a⟩
  shows ⟨v ≥ 0⟩
proof -
  from assms obtain h where ⟨norm h = 1⟩ and ahvh: ⟨a *V h = v *C h⟩
  using unit-eigenvector-ex by blast
  have ⟨0 ≤ h ·C a h⟩
    by (simp add: assms(1) cinner-pos-if-pos)
  also have ⟨... = v * (h ·C h)⟩
    by (simp add: ahvh)
  also have ⟨... = v⟩
    using ⟨norm h = 1⟩ cnorm-eq-1 by auto
  finally show ⟨v ≥ 0⟩
    by blast
qed

```

unbundle no cblinfun-syntax

end

10 Compact-Operators – Finite rank and compact operators

theory *Compact-Operators*

imports *Misc-Tensor-Product-BO HS2Ell2*
Sqrt-Babylonian.Sqrt-Babylonian-Auxiliary Wlog. Wlog
HOL-Analysis.Abstract-Metric-Spaces
Strong-Operator-Topology
Misc-Tensor-Product-TTS
Eigenvalues

begin

unbundle *cblinfun-syntax*

10.1 Finite rank operators

definition *finite-rank* **where** $\langle \text{finite-rank } A \longleftrightarrow A \in \text{cspan } (\text{Collect rank1}) \rangle$

lemma *finite-rank-0[simp]*: $\langle \text{finite-rank } 0 \rangle$
by (*simp add: complex-vector.span-zero finite-rank-def*)

lemma *finite-rank-scaleC[simp]*: $\langle \text{finite-rank } (c *_C a) \rangle$ **if** $\langle \text{finite-rank } a \rangle$
using *complex-vector.span-scale finite-rank-def* **that** **by** *blast*

lemma *finite-rank-scaleR[simp]*: $\langle \text{finite-rank } (c *_R a) \rangle$ **if** $\langle \text{finite-rank } a \rangle$
by (*simp add: scaleR-scaleC that*)

lemma *finite-rank-uminus[simp]*: $\langle \text{finite-rank } (-a) = \text{finite-rank } a \rangle$
by (*metis add.inverse-inverse complex-vector.span-neg finite-rank-def*)

lemma *finite-rank-plus[simp]*: $\langle \text{finite-rank } (a + b) \rangle$ **if** $\langle \text{finite-rank } a \rangle$ **and** $\langle \text{finite-rank } b \rangle$
using *that* **by** (*auto simp: finite-rank-def complex-vector.span-add-eq2*)

lemma *finite-rank-minus[simp]*: $\langle \text{finite-rank } (a - b) \rangle$ **if** $\langle \text{finite-rank } a \rangle$ **and** $\langle \text{finite-rank } b \rangle$
using *complex-vector.span-diff finite-rank-def that(1) that(2)* **by** *blast*

lemma *finite-rank-butterfly[simp]*: $\langle \text{finite-rank } (\text{butterfly } x y) \rangle$
by (*cases* $\langle x \neq 0 \wedge y \neq 0 \rangle$)
(*auto intro: complex-vector.span-base complex-vector.span-zero simp add: finite-rank-def*)

lemma *finite-rank-sum-butterfly*:
fixes $a :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
assumes $\langle \text{finite-rank } a \rangle$
shows $\langle \exists x y (n::\text{nat}). a = (\sum_{i < n}. \text{butterfly } (x i) (y i)) \rangle$
proof –
from *assms*


```

have ⟨a ∈ cspan (Collect rank1)⟩
  by (simp add: finite-rank-def)
then obtain r t where ⟨finite t⟩ and t-rank1: ⟨t ⊆ Collect rank1⟩
  and a-sum: ⟨a = (∑ a∈t. r a *C a)⟩
  by (smt (verit, best) complex-vector.span-alt mem-Collect-eq)
from ⟨finite t⟩ obtain ι and n::nat where ι: ⟨bij-betw ι {..<n} t⟩
  using bij-betw-from-nat-into-finite by blast
define c where ⟨c i = r (ι i) *C ι i⟩ for i
from ι t-rank1
have c-rank1: ⟨rank1 (c i) ∨ c i = 0⟩ if ⟨i < n⟩ for i
  by (auto intro!: rank1-scaleC simp: c-def bij-betw-apply subset-iff that)
have ac-sum: ⟨a = (∑ i<n. c i)⟩
  by (smt (verit, best) a-sum ι c-def sum.cong sum.reindex-bij-betw)
from c-rank1
obtain x y where ⟨c i = butterfly (x i) (y i)⟩ if ⟨i < n⟩ for i
  apply atomize-elim
  apply (rule SMT-choices)
  using rank1-iff-butterfly by fastforce
with ac-sum show ?thesis
  by auto
qed

lemma finite-rank-sum: ⟨finite-rank (∑ x∈F. f x)⟩ if ⟨∧x. x∈F ⇒ finite-rank (f x)⟩
  using that by (induction F rule:infinite-finite-induct) (auto intro!: finite-rank-plus)

lemma rank1-finite-rank: ⟨finite-rank a⟩ if ⟨rank1 a⟩
  by (simp add: complex-vector.span-base finite-rank-def that)

lemma finite-rank-compose-left:
  assumes ⟨finite-rank B⟩
  shows ⟨finite-rank (A oCL B)⟩
proof -
  from assms have ⟨B ∈ cspan (Collect rank1)⟩
    by (simp add: finite-rank-def)
  then obtain F t where ⟨finite F⟩ and F-rank1: ⟨F ⊆ Collect rank1⟩ and ⟨B = (∑ x∈F. t
x *C x)⟩
    by (smt (verit, best) complex-vector.span-explicit mem-Collect-eq)
  then have ⟨A oCL B = (∑ x∈F. t x *C (A oCL x))⟩
    by (metis (mono-tags, lifting) cblinfun-compose-scaleC-right cblinfun-compose-sum-right
sum.cong)
  also have ⟨... ∈ cspan (Collect finite-rank)⟩
    by (intro complex-vector.span-sum complex-vector.span-scale)
    (use F-rank1 in ⟨auto intro!: complex-vector.span-base rank1-finite-rank rank1-compose-left⟩)
  also have ⟨... = Collect finite-rank⟩
    by (metis (no-types, lifting) complex-vector.span-superset cspan-eqI finite-rank-def mem-Collect-eq
subset-antisym subset-iff)
  finally show ?thesis
    by simp

```

qed

lemma *finite-rank-compose-right*:

assumes $\langle \text{finite-rank } A \rangle$

shows $\langle \text{finite-rank } (A \text{ } o_{CL} \text{ } B) \rangle$

proof –

from *assms* **have** $\langle A \in \text{cspan } (\text{Collect rank1}) \rangle$

by (*simp add: finite-rank-def*)

then obtain $F \ t$ **where** $\langle \text{finite } F \rangle$ **and** $F\text{-rank1}$: $\langle F \subseteq \text{Collect rank1} \rangle$ **and** $\langle A = (\sum_{x \in F}. t \ x \ *_C \ x) \rangle$

by (*smt (verit, best) complex-vector.span-explicit mem-Collect-eq*)

then have $\langle A \text{ } o_{CL} \text{ } B = (\sum_{x \in F}. t \ x \ *_C \ (x \text{ } o_{CL} \text{ } B)) \rangle$

by (*metis (mono-tags, lifting) cblinfun-compose-scaleC-left cblinfun-compose-sum-left sum.cong*)

also have $\langle \dots \in \text{cspan } (\text{Collect finite-rank}) \rangle$

by (*intro complex-vector.span-sum complex-vector.span-scale*)

(*use F-rank1 in <auto intro!: complex-vector.span-base rank1-finite-rank rank1-compose-right>*)

also have $\langle \dots = \text{Collect finite-rank} \rangle$

by (*metis (no-types, lifting) complex-vector.span-superset cspan-eqI finite-rank-def mem-Collect-eq subset-antisym subset-iff*)

finally show *?thesis*

by *simp*

qed

lemma *rank1-Proj-singleton[iff]*: $\langle \text{rank1 } (\text{Proj } (\text{ccspan } \{x\})) \rangle$

using *Proj-range rank1-def by blast*

lemma *finite-rank-Proj-singleton[iff]*: $\langle \text{finite-rank } (\text{Proj } (\text{ccspan } \{x\})) \rangle$

by (*simp add: rank1-finite-rank*)

lemma *finite-rank-Proj-finite-dim*:

fixes $S :: \langle 'a::\text{chilbert-space ccspace} \rangle$

assumes $\langle \text{finite-dim-ccspace } S \rangle$

shows $\langle \text{finite-rank } (\text{Proj } S) \rangle$

proof –

from *assms*

obtain B **where** $\langle \text{is-ortho-set } B \rangle$ **and** $\langle \text{finite } B \rangle$ **and** *spanB*: $\langle \text{cspan } B = \text{space-as-set } S \rangle$

unfolding *finite-dim-ccspace.rep-eq*

using *cfinite-dim-subspace-has-onb by force*

have $\langle \text{Proj } S = \text{Proj } (\text{ccspan } B) \rangle$

by (*metis Proj.rep-eq <finite B> cblinfun-apply-inject ccspan-finite spanB*)

moreover have $\langle \text{finite-rank } (\text{Proj } (\text{ccspan } B)) \rangle$

using $\langle \text{finite } B \rangle \langle \text{is-ortho-set } B \rangle$

proof *induction*

case *empty*

then show *?case*

by *simp*

next

case (*insert x F*)

```

then have ‹is-ortho-set F›
  by (meson is-ortho-set-antimono subset-insertI)
have ‹Proj (ccspan (insert x F)) = proj x + Proj (ccspan F)›
  by (subst Proj-orthog-ccspan-insert)
    (use insert in ‹auto simp: is-onb-def is-ortho-set-def›)
moreover have ‹finite-rank ...›
  by (rule finite-rank-plus)
    (auto intro!: ‹is-ortho-set F› insert)
ultimately show ?case
  by simp
qed
ultimately show ?thesis
  by simp
qed

```

```

lemma finite-rank-Proj-finite:
  fixes F :: ‹'a::chilbert-space set›
  assumes ‹finite F›
  shows ‹finite-rank (Proj (ccspan F))›
proof -
  obtain B where ‹is-ortho-set B› and ‹finite B› and ‹cspan B = cspan F›
  by (meson assms orthonormal-basis-of-cspan)
  have ‹Proj (ccspan F) = Proj (ccspan B)›
  by (simp add: ‹cspan B = cspan F› ccspan.abs-eq)
  moreover have ‹finite-rank (Proj (ccspan B))›
  using ‹finite B› ‹is-ortho-set B›
proof induction
  case empty
  then show ?case
  by simp
next
  case (insert x F)
  then have ‹is-ortho-set F›
  by (meson is-ortho-set-antimono subset-insertI)
  have ‹Proj (ccspan (insert x F)) = proj x + Proj (ccspan F)›
  by (subst Proj-orthog-ccspan-insert)
    (use insert in ‹auto simp: is-onb-def is-ortho-set-def›)
  moreover have ‹finite-rank ...›
  by (rule finite-rank-plus) (auto intro!: ‹is-ortho-set F› insert)
  ultimately show ?case
  by simp
qed
ultimately show ?thesis
  by simp
qed

```

```

lemma finite-rank-cfinite-dim[simp]: ‹finite-rank (a :: 'a :: {cfinite-dim,chilbert-space}) ⇒CL 'b
:: complex-normed-vector›
proof -

```

```

obtain  $B :: \langle 'a \text{ set} \rangle$  where  $\langle is\text{-onb } B \rangle$ 
  using  $is\text{-onb-some-chilbert-basis}$  by  $blast$ 
from  $\langle is\text{-onb } B \rangle$  have  $[simp]: \langle finite\ B \rangle$ 
  by  $(auto\ intro!: c\text{independent-}c\text{finite-dim-finite } is\text{-ortho-set-}c\text{independent } simp\ add: is\text{-onb-def})$ 
have  $[simp]: \langle cspan\ B = UNIV \rangle$ 
proof  $-$ 
  from  $\langle is\text{-onb } B \rangle$  have  $\langle ccspan\ B = \top \rangle$ 
    using  $is\text{-onb-def}$  by  $blast$ 
  then have  $\langle closure\ (cspan\ B) = UNIV \rangle$ 
    by  $(metis\ ccspan.rep\text{-}eq\ space\text{-}as\text{-}set\text{-}top)$ 
  then show  $?thesis$ 
    by  $simp$ 
qed
have  $a\text{-sum}: \langle a = (\sum b \in B. a\ o_{CL}\ selfbutter\ b) \rangle$ 
proof  $(rule\ cblinfun\text{-}eq\text{-}on\text{-}UNIV\text{-}span[OF\ \langle cspan\ B = UNIV \rangle])$ 
  fix  $s$  assume  $[simp]: \langle s \in B \rangle$ 
  with  $\langle is\text{-onb } B \rangle$  have  $\langle norm\ s = 1 \rangle$ 
    by  $(simp\ add: is\text{-onb-def})$ 
  have  $1: \langle j \neq s \implies j \in B \implies (a\ o_{CL}\ selfbutter\ j) *_{\mathbb{V}} s = 0 \rangle$  for  $j$ 
    using  $\langle is\text{-onb } B \rangle \langle s \in B \rangle cblinfun.scaleC\text{-}right\ is\text{-onb-def}\ is\text{-ortho-set-def}\ scaleC\text{-}eq\text{-}0\text{-iff}$ 
    by  $fastforce$ 
  have  $2: \langle a *_{\mathbb{V}} s = (if\ s \in B\ then\ (a\ o_{CL}\ selfbutter\ s) *_{\mathbb{V}} s\ else\ 0) \rangle$ 
    using  $\langle norm\ s = 1 \rangle \langle s \in B \rangle$  by  $(simp\ add: cnorm\text{-}eq\text{-}1)$ 
  show  $\langle a *_{\mathbb{V}} s = (\sum b \in B. a\ o_{CL}\ selfbutter\ b) *_{\mathbb{V}} s \rangle$ 
    by  $(subst\ cblinfun.sum\text{-}left, subst\ sum\text{-}single[where\ i=s])\ (use\ 1\ 2\ in\ auto)$ 
qed
have  $\langle finite\text{-}rank\ (\sum b \in B. a\ o_{CL}\ selfbutter\ b) \rangle$ 
  by  $(auto\ intro!: finite\text{-}rank\text{-}sum\ simp: cblinfun\text{-}comp\text{-}butterfly)$ 
with  $a\text{-sum}$  show  $?thesis$ 
  by  $simp$ 
qed

lemma  $finite\text{-}rank\text{-}cspan\text{-}butterflies:$ 
   $\langle finite\text{-}rank\ a \iff a \in cspan\ (range\ (case\text{-}prod\ butterfly)) \rangle$ 
  for  $a :: \langle 'a::chilbert\text{-}space \Rightarrow_{CL}\ 'b::chilbert\text{-}space \rangle$ 
proof  $-$ 
  have  $\langle (Collect\ finite\text{-}rank :: ('a \Rightarrow_{CL}\ 'b)\ set) = cspan\ (Collect\ rank1) \rangle$ 
    using  $finite\text{-}rank\text{-}def$  by  $fastforce$ 
  also have  $\langle \dots = cspan\ (insert\ 0\ (Collect\ rank1)) \rangle$ 
    by  $force$ 
  also have  $\langle \dots = cspan\ (range\ (case\text{-}prod\ butterfly)) \rangle$ 
    by  $(rule\ arg\text{-}cong[where\ f=cspan])$ 
     $(use\ butterfly\text{-}0\text{-}left\ in\ \langle force\ simp: image\text{-}iff\ rank1\text{-}iff\text{-}butterfly\ simp\ del: butterfly\text{-}0\text{-}left \rangle)$ 
  finally show  $?thesis$ 
    by  $auto$ 
qed

lemma  $finite\text{-}rank\text{-}comp\text{-}left: \langle finite\text{-}rank\ (a\ o_{CL}\ b) \rangle$  if  $\langle finite\text{-}rank\ a \rangle$ 

```

```

  for a b :: ⟨-::chilbert-space ⇒CL -::chilbert-space⟩
proof –
  from that
  have ⟨a ∈ cspan (range (case-prod butterfly))⟩
    by (simp add: finite-rank-cspan-butterflies)
  then have ⟨a oCL b ∈ (λa. a oCL b) ‘ cspan (range (case-prod butterfly))⟩
    by fast
  also have ⟨... = cspan ((λa. a oCL b) ‘ range (case-prod butterfly))⟩
    by (simp add: clinear-cblinfun-compose-left complex-vector.linear-span-image)
  also have ⟨... ⊆ cspan (range (case-prod butterfly))⟩
    by (force intro!: complex-vector.span-mono
        simp add: image-image case-prod-unfold butterfly-comp-cblinfun image-def)
  finally show ?thesis
  using finite-rank-cspan-butterflies by blast
qed

```

```

lemma finite-rank-comp-right: ⟨finite-rank (a oCL b)⟩ if ⟨finite-rank b⟩
  for a b :: ⟨-::chilbert-space ⇒CL -::chilbert-space⟩
proof –
  from that
  have ⟨b ∈ cspan (range (case-prod butterfly))⟩
    by (simp add: finite-rank-cspan-butterflies)
  then have ⟨a oCL b ∈ ((oCL) a) ‘ cspan (range (case-prod butterfly))⟩
    by fast
  also have ⟨... = cspan (((oCL) a) ‘ range (case-prod butterfly))⟩
    by (simp add: clinear-cblinfun-compose-right complex-vector.linear-span-image)
  also have ⟨... ⊆ cspan (range (case-prod butterfly))⟩
    by (force intro!: complex-vector.span-mono
        simp add: image-image case-prod-unfold cblinfun-comp-butterfly image-def)
  finally show ?thesis
  using finite-rank-cspan-butterflies by blast
qed

```

10.2 Compact operators

definition compact-map **where** ⟨compact-map f ⟷ clinear f ∧ compact (closure (f ‘ cball 0 1))⟩

lemma ⟨bounded-clinear f⟩ **if** ⟨compact-map f⟩

— [2], Proposition II.4.2 (a)

thm bounded-clinear-def

proof (unfold bounded-clinear-def bounded-clinear-axioms-def, intro conjI)

show ⟨clinear f⟩

using compact-map-def that **by** blast

have ⟨compact (closure (f ‘ cball 0 1))⟩

using compact-map-def that **by** blast

then have ⟨bounded (f ‘ cball 0 1)⟩

by (meson bounded-subset closure-subset compact-imp-bounded)

```

then obtain  $K$  where  $*$ :  $\langle \text{norm } (f x) \leq K \rangle$  if  $\langle \text{norm } x \leq 1 \rangle$  for  $x$ 
  by  $(\text{force simp: bounded-iff dist-norm ball-def})$ 
have  $\langle \text{norm } (f x) \leq \text{norm } x * K \rangle$  for  $x$ 
proof  $(\text{cases } \langle x = 0 \rangle)$ 
  case True
    then show  $?thesis$ 
    using  $\langle \text{linear } f \rangle$   $\text{complex-vector.linear-0}$  by force
  next
  case False
    have  $\langle \text{norm } (f x) = \text{norm } (f (\text{norm } x *_{\mathbb{C}} \text{sgn } x)) \rangle$ 
    by simp
    also have  $\langle \dots = \text{norm } x * \text{norm } (f (\text{sgn } x)) \rangle$ 
    by  $(\text{smt } (\text{verit, best}) \langle \text{linear } f \rangle \text{complex-vector.linear-scale norm-ge-zero norm-of-real norm-scaleC})$ 
    also have  $\langle \dots \leq \text{norm } x * K \rangle$ 
    by  $(\text{simp add: } * \text{mult-left-mono norm-sgn})$ 
    finally show  $?thesis$ 
    by  $-$ 
  qed
then show  $\langle \exists K. \forall x. \text{norm } (f x) \leq \text{norm } x * K \rangle$ 
by blast
qed

```

lift-definition $\text{compact-op} :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{\mathbb{C}L} 'b::\text{complex-normed-vector} \rangle \Rightarrow \text{bool}$ **is** compact-map .

lemma compact-op-def2 : $\langle \text{compact-op } a \longleftrightarrow \text{compact } (\text{closure } (a \text{ ' cball } 0 \ 1)) \rangle$
by $\text{transfer } (\text{use bounded-clinear.clinear compact-map-def in blast})$

lemma $\text{compact-op-0[simp]}$: $\langle \text{compact-op } 0 \rangle$
by $(\text{simp add: compact-op-def2 image-constant[where } x=0] \text{mem-cball-leI[where } x=0])$

lemma $\text{compact-op-scaleC[simp]}$: $\langle \text{compact-op } (c *_{\mathbb{C}} a) \rangle$ **if** $\langle \text{compact-op } a \rangle$
proof $-$
have $\langle \text{compact } (\text{closure } (a \text{ ' cball } 0 \ 1)) \rangle$
using compact-op-def2 **that** **by** *blast*
then have $*$: $\langle \text{compact } (\text{scaleC } c \text{ ' closure } (a \text{ ' cball } 0 \ 1)) \rangle$
using compact-scaleC **by** *blast*
have $\langle \text{closure } ((c *_{\mathbb{C}} a) \text{ ' cball } 0 \ 1) = \text{closure } (\text{scaleC } c \text{ ' } a \text{ ' cball } 0 \ 1) \rangle$
by $(\text{metis } (\text{no-types, lifting}) \text{cblinfun.scaleC-left image-cong image-image})$
also have $\langle \dots = \text{scaleC } c \text{ ' closure } (a \text{ ' cball } 0 \ 1) \rangle$
using closure-scaleC **by** *blast*
finally have $\langle \text{compact } (\text{closure } ((c *_{\mathbb{C}} a) \text{ ' cball } 0 \ 1)) \rangle$
using $*$ **by** *simp*
then show $?thesis$
using compact-op-def2 **by** *blast*
qed

lemma $\text{compact-op-scaleR[simp]}$: $\langle \text{compact-op } (c *_{\mathbb{R}} a) \rangle$ **if** $\langle \text{compact-op } a \rangle$

by (simp add: scaleR-scaleC that)

lemma compact-op-uminus[simp]: $\langle \text{compact-op } (-a) = \text{compact-op } a \rangle$
by (metis compact-op-scaleC scaleC-minus1-left verit-minus-simplify(4))

lemma compact-op-plus[simp]: $\langle \text{compact-op } (a + b) \rangle$ if $\langle \text{compact-op } a \rangle$ and $\langle \text{compact-op } b \rangle$
proof –

have $\langle \text{compact } (\text{closure } (a \text{ ‘ cball } 0 \ 1)) \rangle$

using compact-op-def2 that by blast

moreover have $\langle \text{compact } (\text{closure } (b \text{ ‘ cball } 0 \ 1)) \rangle$

using compact-op-def2 that by blast

ultimately have compact-sum:

$\langle \text{compact } \{x + y \mid x \in \text{closure } ((*_V) a \text{ ‘ cball } 0 \ 1) \wedge y \in \text{closure } ((*_V) b \text{ ‘ cball } 0 \ 1)\} \rangle$ (is $\langle \text{compact } ?sum \rangle$)

by (rule compact-sums)

have $\langle \text{compact } (\text{closure } ((a + b) \text{ ‘ cball } 0 \ 1)) \rangle$

proof –

have $\langle ((*_V) (a + b) \text{ ‘ cball } 0 \ 1) \subseteq ?sum \rangle$

using cblinfun.real.add-left closure-subset image-subset-iff by blast

then have $\langle \text{closure } ((*_V) (a + b) \text{ ‘ cball } 0 \ 1) \subseteq \text{closure } ?sum \rangle$

by (meson closure-mono)

also have $\langle \dots = ?sum \rangle$

using compact-sum

by (auto intro!: closure-closed compact-imp-closed)

finally show ?thesis

by (rule compact-closed-subset[rotated 2]) (use compact-sum in auto)

qed

then show ?thesis

using compact-op-def2 by blast

qed

lemma csubspace-compact-op: $\langle \text{csubspace } (\text{Collect compact-op}) \rangle$

– [2], Proposition II.4.2 (b)

by (simp add: complex-vector.subspace-def)

lemma compact-op-minus[simp]: $\langle \text{compact-op } (a - b) \rangle$ if $\langle \text{compact-op } a \rangle$ and $\langle \text{compact-op } b \rangle$
by (metis compact-op-plus compact-op-uminus that(1) that(2) uminus-add-conv-diff)

lemma compact-op-sgn[simp]: $\langle \text{compact-op } (\text{sgn } a) = \text{compact-op } a \rangle$

proof (cases $\langle a = 0 \rangle$)

case True

then show ?thesis

by simp

next

case False

have $\langle \text{compact-op } (\text{sgn } a) \rangle$ if $\langle \text{compact-op } a \rangle$

by (simp add: sgn-cblinfun-def that)

moreover have $\langle \text{compact-op } (\text{norm } a *_R \text{sgn } a) \rangle$ if $\langle \text{compact-op } (\text{sgn } a) \rangle$

by (simp add: that)

moreover have $\langle \text{norm } a *_{\mathbb{R}} \text{sgn } a = a \rangle$
by (*simp add: False sgn-div-norm*)
ultimately show *?thesis*
by auto
qed

lemma closed-compact-op:
shows $\langle \text{closed } (\text{Collect } (\text{compact-op} :: ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{chilbert-space}) \Rightarrow \text{bool})) \rangle$
— [2], Proposition II.4.2 (b)

proof (*intro closed-sequential-limits[THEN iffD2] allI impI conjI*)
fix T **and** $A :: \langle 'a \Rightarrow_{CL} 'b \rangle$
assume *asm*: $\langle (\forall n. T\ n \in \text{Collect } \text{compact-op}) \wedge T \longrightarrow A \rangle$
have $\langle \text{Met-TC.mtotally-bounded } (A \text{ ' cball } 0\ 1) \rangle$
proof (*unfold Met-TC.mtotally-bounded-def, intro allI impI*)
fix $\varepsilon :: \text{real}$ **assume** $\langle \varepsilon > 0 \rangle$
define δ **where** $\langle \delta = \varepsilon/3 \rangle$
then have $\langle \delta > 0 \rangle$
using $\langle \varepsilon > 0 \rangle$ **by** *simp*
from *asm*[*unfolded LIMSEQ-def, THEN conjunct2, rule-format, OF* $\langle \delta > 0 \rangle$]
obtain n **where** *dist-TA*: $\langle \text{dist } (T\ n) A < \delta \rangle$
by auto
from *asm* **have** $\langle \text{compact-op } (T\ n) \rangle$
by simp
then have $\langle \text{Met-TC.mtotally-bounded } (T\ n \text{ ' cball } 0\ 1) \rangle$
by (*subst Met-TC.mtotally-bounded-eq-compact-closure-of*)
(*auto intro!: simp: compact-op-def2 Met-TC.mtotally-bounded-eq-compact-closure-of*)
then obtain K **where** $\langle \text{finite } K \rangle$ **and** $K\text{-}T$: $\langle K \subseteq T\ n \text{ ' cball } 0\ 1 \rangle$ **and**
 TK : $\langle T\ n \text{ ' cball } 0\ 1 \subseteq (\bigcup k \in K. \text{Met-TC.mball } k\ \delta) \rangle$
unfolding *Met-TC.mtotally-bounded-def* **using** $\langle \delta > 0 \rangle$ **by meson**
from $\langle \text{finite } K \rangle$ **and** $K\text{-}T$ **obtain** H **where** $\langle \text{finite } H \rangle$ **and** $\langle H \subseteq \text{cball } 0\ 1 \rangle$
and KTH : $\langle K = T\ n \text{ ' } H \rangle$
by (*meson finite-subset-image*)
from TK **have** TH : $\langle T\ n \text{ ' cball } 0\ 1 \subseteq (\bigcup h \in H. \text{ball } (T\ n *_{\mathbb{V}} h)\ \delta) \rangle$
by (*simp add: KTH*)
have $\langle A \text{ ' cball } 0\ 1 \subseteq (\bigcup h \in H. \text{ball } (A\ h)\ \varepsilon) \rangle$
proof (*rule subsetI*)
fix x **assume** $\langle x \in (*_{\mathbb{V}}) A \text{ ' cball } 0\ 1 \rangle$
then obtain l **where** $\langle l \in \text{cball } 0\ 1 \rangle$ **and** xl : $\langle x = A\ l \rangle$
by blast
then have $\langle T\ n\ l \in T\ n \text{ ' cball } 0\ 1 \rangle$
by auto
with TH **obtain** h **where** $\langle h \in H \rangle$ **and** $\langle T\ n\ l \in \text{ball } (T\ n\ h)\ \delta \rangle$
by blast
then have *dist-Tlh*: $\langle \text{dist } (T\ n\ l) (T\ n\ h) < \delta \rangle$
by (*simp add: dist-commute*)
have $\langle \text{dist } (A\ h) (A\ l) < \varepsilon \rangle$
proof —
have *norm-h*: $\langle \text{norm } h \leq 1 \rangle$


```

    using ⟨ $H \subseteq \text{cball } 0 \ 1$ ⟩ ⟨ $h \in H$ ⟩ mem-cball-0 by blast
  have norm-l: ⟨ $\text{norm } l \leq 1$ ⟩
    using ⟨ $l \in \text{cball } 0 \ 1$ ⟩ by auto
  have ⟨ $\text{dist } (A \ h) \ (T \ n \ h) < \delta$ ⟩
  proof -
    have ⟨ $\text{dist } (T \ n \ *_V \ h) \ (A \ *_V \ h) \leq \text{norm } h * \text{dist } (T \ n) \ A$ ⟩
      using norm-cblinfun[of  $T \ n - A \ h$ ] by (simp add: dist-norm cblinfun.diff-left mult-ac)
    also have ⟨ $\dots \leq 1 * \text{dist } (T \ n) \ A$ ⟩
      by (rule mult-right-mono) (use norm-h in auto)
    also have ⟨ $\text{dist } (T \ n) \ A < \delta$ ⟩
      by fact
    finally show ?thesis
      by (simp add: dist-commute)
  qed
  moreover have ⟨ $\text{dist } (T \ n \ h) \ (T \ n \ l) < \delta$ ⟩
    using dist-Tlh by (metis dist-commute)
  moreover from dist-TA norm-l have ⟨ $\text{dist } (T \ n \ l) \ (A \ l) < \delta$ ⟩
  proof -
    have ⟨ $\text{dist } (T \ n \ *_V \ l) \ (A \ *_V \ l) \leq \text{norm } l * \text{dist } (T \ n) \ A$ ⟩
      using norm-cblinfun[of  $T \ n - A \ l$ ] by (simp add: dist-norm cblinfun.diff-left mult-ac)
    also have ⟨ $\dots \leq 1 * \text{dist } (T \ n) \ A$ ⟩
      by (rule mult-right-mono) (use norm-l in auto)
    also have ⟨ $\text{dist } (T \ n) \ A < \delta$ ⟩
      by fact
    finally show ?thesis
      by (simp add: dist-commute)
  qed
  ultimately show ?thesis
    unfolding  $\delta$ -def
    by (rule dist-triangle-third)
  qed
  then show ⟨ $x \in (\bigcup_{h \in H}. \text{ball } (A \ h) \ \varepsilon)$ ⟩
    using ⟨ $h \in H$ ⟩ by (auto intro!: simp: xl)
  qed
  then show ⟨ $\exists K. \text{finite } K \wedge K \subseteq (*_V) \ A \ \text{'cball } 0 \ 1 \wedge$   

     $(*_V) \ A \ \text{'cball } 0 \ 1 \subseteq (\bigcup_{x \in K}. \text{Met-TC.mball } x \ \varepsilon)$ ⟩
    using ⟨ $H \subseteq \text{cball } 0 \ 1$ ⟩
    by (force intro!: exI[of - ⟨ $A \ \text{'H}$ ⟩] ⟨finite H⟩ simp: ball-def)
  qed
  then have ⟨Met-TC.mtotally-bounded (closure ( $A \ \text{'cball } 0 \ 1$ ))⟩
    using Met-TC.mtotally-bounded-closure-of by auto
  then have ⟨compact (closure ( $A \ \text{'cball } 0 \ 1$ ))⟩
    by (simp-all add: Met-TC.mtotally-bounded-eq-compact-closure-of complete-UNIV-cuspace)
  then show ⟨ $A \in \text{Collect compact-op}$ ⟩
    using compact-op-def2 by blast
  qed

lemma rank1-compact-op: ⟨compact-op a⟩ if ⟨rank1 a⟩
proof -

```

```

wlog ⟨a ≠ 0⟩
  using negation by simp
with that obtain ψ where im-a: ⟨a *S ⊤ = cspan {ψ}⟩ and ⟨ψ ≠ 0⟩
  using rank1-def by fastforce
define c where ⟨c = norm a / norm ψ⟩
have compact-ψc: ⟨compact ((λx. x *C ψ) ‘ cball 0 c)⟩
proof -
  have ⟨continuous-on (cball 0 c) (λx. x *C ψ)⟩
    by (auto intro!: continuous-at-imp-continuous-on)
  moreover have ⟨compact (cball (0::complex) c)⟩
    by (simp add: compact-eq-bounded-closed)
  ultimately show ?thesis
    by (rule compact-continuous-image)
qed
have ⟨a ‘ cball 0 1 ⊆ (λx. x *C ψ) ‘ cball 0 c⟩
proof (rule subsetI)
  fix φ
  assume asm: ⟨φ ∈ a ‘ cball 0 1⟩
  then have ⟨φ ∈ space-as-set (a *S ⊤)⟩
    using cblinfun-apply-in-image by blast
  also have ⟨... = cspan {ψ}⟩
    by (simp add: cspan.rep-eq im-a)
  finally obtain d where d: ⟨φ = d *C ψ⟩
    by (metis complex-vector.span-breakdown-eq complex-vector.span-empty eq-iff-diff-eq-0 singletonD)
  from asm obtain γ where ⟨φ = a γ⟩ and ⟨norm γ ≤ 1⟩
    by force
  have ⟨cmod d * norm ψ = norm φ⟩
    by (simp add: d)
  also have ⟨... ≤ norm a * norm γ⟩
    using ⟨φ = a *V γ⟩ complex-of-real-mono norm-cblinfun by blast
  also have ⟨... ≤ norm a⟩
    by (metis ⟨norm γ ≤ 1⟩ mult.commute mult-left-le-one-le norm-ge-zero)
  finally have ⟨cmod d ≤ c⟩
    by (smt (verit, ccfv-threshold) ⟨ψ ≠ 0⟩ c-def linordered-field-class.pos-divide-le-eq nonzero-eq-divide-eq norm-le-zero-iff)
  then show ⟨φ ∈ (λx. x *C ψ) ‘ cball 0 c⟩
    by (auto simp: d)
qed
with compact-ψc have cl-in-cl: ⟨closure (a ‘ cball 0 1) ⊆ ((λx. x *C ψ) ‘ cball 0 c)⟩
  using closure-mono[of - ⟨(λx. x *C ψ) ‘ cball 0 c⟩] compact-ψc
  by (simp add: compact-imp-closed)
with compact-ψc show ⟨compact-op a⟩
  using compact-closed-subset compact-op-def2 by blast
qed

lemma finite-rank-compact-op: ⟨compact-op a⟩ if ⟨finite-rank a⟩
proof -
  from that obtain t r where ⟨finite t⟩ and ⟨t ⊆ Collect rank1⟩

```

and $a\text{-decomp}$: $\langle a = (\sum_{x \in t}. r\ x *_{\mathcal{C}}\ x) \rangle$
by (*auto simp: finite-rank-def complex-vector.span-explicit*)
from $\langle \text{finite } t \rangle \langle t \subseteq \text{Collect rank1} \rangle$ **show** $\langle \text{compact-op } a \rangle$
by (*unfold a-decomp, induction*)
(auto intro!: compact-op-plus compact-op-scaleC intro: rank1-compact-op)
qed

lemma *bounded-products-sot-lim-imp-lim*:

— Implicit in the proof of [2], Proposition II.4.4 (c)

fixes $A :: \langle 'a :: \text{complex-normed-vector} \Rightarrow_{\mathcal{CL}} 'b :: \text{chilbert-space} \rangle$

assumes lim-PA : $\langle \text{limitin cstrong-operator-topology } (\lambda x. P\ x\ o_{\mathcal{CL}}\ A)\ A\ F \rangle$

and $\langle \text{compact-op } A \rangle$

and $P\text{-leq-}B$: $\langle \bigwedge x. \text{norm } (P\ x) \leq B \rangle$

shows $\langle ((\lambda x. P\ x\ o_{\mathcal{CL}}\ A) \longrightarrow A)\ F \rangle$

proof —

wlog $\langle F \neq \perp \rangle$

using *negation by simp*

wlog $\langle B \neq 0 \rangle$

proof —

from *negation assms* **have** $P0$: $\langle P\ x = 0 \rangle$ **for** x

by *auto*

from lim-PA **have** $\langle ((\lambda x. 0) \longrightarrow \text{Abs-cblinfun-sot } A)\ F \rangle$

unfolding *limitin-canonical-iff [symmetric]*

by (*transfer fixing: P F*) (*use P0 in simp*)

moreover **have** $\langle ((\lambda x. 0) \longrightarrow 0)\ F \rangle$

by *simp*

ultimately **have** $\langle \text{Abs-cblinfun-sot } A = 0 \rangle$

using $\langle F \neq \perp \rangle$ *tendsto-unique by blast*

then **have** $\langle A = 0 \rangle$

by (*metis Abs-cblinfun-sot-inverse cstrong-operator-topology-topospace lim-PA*
limitin-def zero-cblinfun-sot.rep-eq)

with $P0$ **show** *?thesis*

by *simp*

qed

have $\langle B > 0 \rangle$

proof —

from $P\text{-leq-}B$ [*of undefined*] **have** $\langle B \geq 0 \rangle$

by (*smt (verit, del-Insts) norm-ge-zero*)

with $\langle B \neq 0 \rangle$

show *?thesis*

by *simp*

qed

show *?thesis*

proof (*rule metric-space-class.tendstoI*)

fix $\varepsilon :: \text{real}$ **assume** $\langle \varepsilon > 0 \rangle$

define $\delta\ \gamma\ T$ **where** $\langle \delta = \varepsilon/4 \rangle$ **and** $\langle \gamma = \min \delta\ (\delta/B) \rangle$ **and** $\langle T\ x = P\ x\ o_{\mathcal{CL}}\ A \rangle$ **for** x

then **have** $\langle \delta > 0 \rangle$

using $\langle \varepsilon > 0 \rangle$ **by** *simp*

```

then have ⟨ $\gamma > 0$ ⟩
  using ⟨ $B > 0$ ⟩ by (simp add:  $\gamma$ -def)
from ⟨compact-op A⟩ have ⟨Met-TC.mtotally-bounded (A ‘ cball 0 1)⟩
  by (subst Met-TC.mtotally-bounded-eq-compact-closure-of)
  (auto intro!: simp: compact-op-def2 Met-TC.mtotally-bounded-eq-compact-closure-of)
then obtain K where ⟨finite K⟩ and K-T: ⟨ $K \subseteq A$  ‘ cball 0 1⟩ and
  AK: ⟨ $A$  ‘ cball 0 1  $\subseteq (\bigcup_{k \in K} \text{Met-TC.mball } k \ \gamma)$ ⟩
  unfolding Met-TC.mtotally-bounded-def using ⟨ $\gamma > 0$ ⟩ by meson
from ⟨finite K⟩ and K-T obtain H where ⟨finite H⟩ and ⟨ $H \subseteq \text{cball } 0 \ 1$ ⟩
  and KAH: ⟨ $K = A$  ‘ H⟩
  by (meson finite-subset-image)
from AK have AH: ⟨ $A$  ‘ cball 0 1  $\subseteq (\bigcup_{h \in H} \text{ball } (A *_{\vee} h) \ \gamma)$ ⟩
  by (simp add: KAH)
have ⟨ $\forall_F x \text{ in } F. \forall h \in H. \text{dist } (T x h) (A h) < \delta$ ⟩
  using lim-PA ⟨ $\delta > 0$ ⟩
  by (auto intro!: eventually-ball-finite ⟨finite H⟩
    simp: limitin-cstrong-operator-topology T-def metric-space-class.tendsto-iff)
then show ⟨ $\forall_F x \text{ in } F. \text{dist } (T x) A < \varepsilon$ ⟩
proof (rule eventually-mono)
  fix x
  assume asm: ⟨ $\forall h \in H. \text{dist } (T x *_{\vee} h) (A *_{\vee} h) < \delta$ ⟩
  have ⟨ $\text{dist } (T x l) (A l) \leq 3 * \delta$ ⟩ if ⟨norm l = 1⟩ for l
  proof -
    from that have ⟨ $A l \in A$  ‘ cball 0 1⟩
      by auto
    with AH obtain h where ⟨ $h \in H$ ⟩ and Al $\gamma$ : ⟨ $A l \in \text{ball } (A h) \ \gamma$ ⟩
      by blast
    then have dist-Alh: ⟨ $\text{dist } (A l) (A h) < \gamma$ ⟩
      by (simp add: dist-commute)
    have ⟨ $\text{dist } (A l) (A h) < \delta$ ⟩
      using dist-Alh by (simp add:  $\gamma$ -def)
    moreover from asm have ⟨ $\text{dist } (A h) (T x h) < \delta$ ⟩
      by (simp add: ⟨ $h \in H$ ⟩ dist-commute)
    moreover have ⟨ $\text{dist } (T x h) (T x l) < \delta$ ⟩
    proof -
      have ⟨ $\text{dist } (T x h) (T x l) \leq \text{norm } (P x) * \text{dist } (A h) (A l)$ ⟩
        by (metis T-def cblinfun.real.diff-right cblinfun-apply-cblinfun-compose
          dist-norm norm-cblinfun)
      also from Al $\gamma$  P-leq-B have ⟨ $\dots < B * \gamma$ ⟩
        by (smt (verit, ccfv-SIG) ⟨ $B \neq 0$ ⟩ linordered-semiring-strict-class.mult-le-less-imp-less
          linordered-semiring-strict-class.mult-strict-mono' mem-ball norm-ge-zero zero-le-dist)
      also have ⟨ $\dots \leq B * (\delta / B)$ ⟩
        by (smt (verit, best)  $\gamma$ -def ⟨ $0 < B$ ⟩ mult-left-mono)
      also have ⟨ $\dots \leq \delta$ ⟩
        by (simp add: ⟨ $B \neq 0$ ⟩)
      finally show ?thesis
        by -
    qed
  ultimately show ?thesis

```

```

    by (smt (verit) dist-commute dist-triangle2)
  qed
  then have ⟨dist (T x) A ≤ 3 * δ⟩
    unfolding dist-norm using ⟨δ > 0⟩
  by (auto intro!: norm-cblinfun-bound-unit simp: cblinfun.diff-left)
  then show ⟨dist (T x) A < ε⟩
    by (rule order.strict-trans1) (use ⟨ε > 0⟩ in ⟨simp add: δ-def⟩)
  qed
  qed
  qed

```

lemma compact-op-finite-rank:

```

  fixes A :: ⟨'a::complex-normed-vector ⇒CL 'b::hilbert-space⟩
  shows ⟨compact-op A ⟷ A ∈ closure (Collect finite-rank)⟩
  — [2], Proposition II.4.4 (c)
  proof (rule iffI)
    assume ⟨A ∈ closure (Collect finite-rank)⟩
    then have ⟨A ∈ closure (Collect compact-op)⟩
      by (metis closure-sequential finite-rank-compact-op mem-Collect-eq)
    also have ⟨... = Collect compact-op⟩
      by (simp add: closed-compact-op)
    finally show ⟨compact-op A⟩
      by simp
  next
    assume ⟨compact-op A⟩
    then have ⟨compact (closure (A ‘ cball 0 1))⟩
      using compact-op-def2 by blast
    then have sep-A-ball: ⟨separable (closure (A ‘ cball 0 1))⟩
      using compact-imp-separable by blast
    define L where ⟨L = closure (range A)⟩
    obtain B :: ⟨nat ⇒ -⟩ where ⟨L ⊆ closure (range B)⟩
  proof atomize-elim
    from sep-A-ball obtain B0 where ⟨countable B0⟩
      and A-B0: ⟨A ‘ cball 0 1 ⊆ closure B0⟩
      by (meson closure-subset order-trans separable-def)
    define B1 where ⟨B1 = (⋃ n::nat. scaleR n ‘ B0)⟩
    from ⟨countable B0⟩ have ⟨countable B1⟩
      by (auto intro!: countable-UN countable-image simp: B1-def)
    have ⟨range A = (⋃ n::nat. A ‘ scaleR n ‘ cball (0::'a) 1)⟩
  proof -
    have ⟨UNIV = (⋃ n::nat. scaleR n ‘ cball (0::'a) 1)⟩
  proof (intro antisym subsetI UNIV-I)
    fix x :: 'a
    have norm x < 1 + real-of-int [norm x] 1 + real-of-int [norm x] > 0
      using norm-ge-zero[of x] by linarith+
    hence ⟨x ∈ scaleR (nat (ceiling (norm x)) + 1) ‘ cball (0::'a) 1⟩
      by (intro image-eqI[where x=⟨x /R (nat (ceiling (norm x)) + 1)⟩])
        (auto simp: divide-simps)
  qed
  qed
  qed

```

```

    then show  $\langle x \in (\bigcup x::nat. (*_R) (real\ x) \text{ ' } cball\ 0\ 1) \rangle$ 
      by blast
  qed
  then show ?thesis
    by fastforce
  qed
  also have  $\langle \dots = (\bigcup n::nat. scaleR\ n \text{ ' } A \text{ ' } cball\ 0\ 1) \rangle$ 
    by (auto simp: cblinfun.scaleR-right image-comp fun-eq-iff)
  also have  $\langle \dots \subseteq (\bigcup n::nat. scaleR\ n \text{ ' } closure\ B0) \rangle$ 
    using A-B0 by fastforce
  also have  $\langle \dots \subseteq closure\ (\bigcup n::nat. scaleR\ n \text{ ' } B0) \rangle$ 
    by (metis (mono-tags, lifting) SUP-le-iff closure-closure closure-mono closure-scaleR closure-subset)
  also have  $\langle \dots = closure\ B1 \rangle$ 
    using B1-def by fastforce
  finally have  $\langle L \subseteq closure\ B1 \rangle$ 
    by (simp add: L-def closure-minimal)
  with  $\langle countable\ B1 \rangle$ 
  show  $\langle \exists B :: nat \Rightarrow -. L \subseteq closure\ (range\ B) \rangle$ 
    by (metis L-def closure-eq-empty empty-not-UNIV image-is-empty range-from-nat-into subset-empty)
  qed
  define P T where  $\langle P\ n = Proj\ (ccspan\ (B \text{ ' } \{..n\})) \rangle$ 
    and  $\langle T\ n = P\ n\ o_{CL}\ A \rangle$  for n
  have  $\langle limitin\ cstrong\ operator\ topology\ T\ A\ sequentially \rangle$ 
  proof (intro limitin-cstrong-operator-topology [THEN iffD2, rule-format] metric-LIMSEQ-I)

    fix h and  $\varepsilon :: real$  assume  $\langle \varepsilon > 0 \rangle$ 
    define Ah where  $\langle Ah = A\ h \rangle$ 
    have  $\langle Ah \in closure\ (range\ B) \rangle$ 
      by (metis L-def Ah-def  $\langle L \subseteq closure\ (range\ B) \rangle$  cblinfun-apply-in-image cblinfun-image.rep-eq subsetD top-ccsubspace.rep-eq)
    then obtain x where  $\langle x \in range\ B \rangle$  and  $\langle dist\ x\ Ah < \varepsilon \rangle$ 
      using  $\langle \varepsilon > 0 \rangle$  unfolding closure-approachable by blast
    then obtain n0 where x-n0:  $\langle x = B\ n0 \rangle$ 
      by blast
    have  $\langle dist\ (P\ n\ *_V\ Ah)\ Ah < \varepsilon \rangle$  if  $\langle n \geq n0 \rangle$  for n
    proof -
      have  $\langle x \in space-as-set\ (P\ n\ *_S\ \top) \rangle$ 
        using  $\langle n \geq n0 \rangle$ 
        by (auto intro!: ccspan-superset' simp: P-def x-n0)
      from Proj-nearest [OF this, of Ah]
      have  $\langle dist\ (P\ n\ *_V\ Ah)\ Ah \leq dist\ x\ Ah \rangle$ 
        by (simp add: P-def)
      with  $\langle dist\ x\ Ah < \varepsilon \rangle$  show ?thesis
        by auto
    qed
  then show  $\langle \exists n0. \forall n \geq n0. dist\ (T\ n\ *_V\ h)\ (A\ *_V\ h) < \varepsilon \rangle$ 
    unfolding T-def Ah-def by auto

```

qed
then have $\langle (\lambda x. P x \text{ } o_{CL} A) \longrightarrow A \rangle$ *sequentially*
unfolding *T-def*
by (*auto intro!*: *bounded-products-sot-lim-imp-lim*[**where** *B=1*] $\langle \text{compact-op } A \rangle$ *norm-is-Proj*
simp: *P-def*)
moreover have $\langle \text{finite-rank } (P x \text{ } o_{CL} A) \rangle$ **for** *x*
by (*auto intro!*: *finite-rank-compose-right* *finite-rank-Proj-finite* *simp*: *P-def*)
ultimately show $\langle A \in \text{closure } (\text{Collect } \text{finite-rank}) \rangle$
using *closure-sequential* **by force**
qed

typedef (**overloaded**) (*'a::chilbert-space, 'b::complex-normed-vector*) *compact-op* =
 $\langle \text{Collect } \text{compact-op} :: ('a \Rightarrow_{CL} 'b) \text{ set} \rangle$
morphisms *from-compact-op* *Abs-compact-op*
by (*auto intro!*: *exI*[*of - 0*])
setup-lifting *type-definition-compact-op*

instantiation *compact-op* :: (*chilbert-space, complex-normed-vector*) *complex-normed-vector* **begin**
lift-definition *scaleC-compact-op* :: $\langle \text{complex} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is**
scaleC **by** *simp*
lift-definition *uminus-compact-op* :: $\langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** *uminus* **by**
simp
lift-definition *zero-compact-op* :: $\langle ('a, 'b) \text{ compact-op} \rangle$ **is** *0* **by** *simp*
lift-definition *minus-compact-op* :: $\langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is**
minus **by** *simp*
lift-definition *plus-compact-op* :: $\langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is**
plus **by** *simp*
lift-definition *sgn-compact-op* :: $\langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** *sgn* **by** *simp*
lift-definition *norm-compact-op* :: $\langle ('a, 'b) \text{ compact-op} \Rightarrow \text{real} \rangle$ **is** *norm* .
lift-definition *scaleR-compact-op* :: $\langle \text{real} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** *scaleR*
by *simp*
lift-definition *dist-compact-op* :: $\langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow \text{real} \rangle$ **is** *dist* .
definition [*code del*]:
 $\langle (\text{uniformity} :: (('a, 'b) \text{ compact-op} \times ('a, 'b) \text{ compact-op}) \text{ filter}) = (\text{INF } e \in \{0 <.. \}. \text{ principal } \{(x, y). \text{ dist } x y < e\}) \rangle$
definition *open-compact-op* :: $(('a, 'b) \text{ compact-op } \text{ set} \Rightarrow \text{bool})$
where [*code del*]: *open-compact-op* *S* = $(\forall x \in S. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in S)$
instance
proof
show $((*_R) r :: ('a, 'b) \text{ compact-op} \Rightarrow -) = (*_C) (\text{complex-of-real } r)$ **for** *r*
by (*rule ext, transfer*) (*simp add: scaleR-scaleC*)
show $a + b + c = a + (b + c)$
for $a \ b \ c :: ('a, 'b) \text{ compact-op}$
by *transfer simp*
show $a + b = b + a$
for $a \ b :: ('a, 'b) \text{ compact-op}$
by *transfer simp*
show $0 + a = a$

```

for  $a :: ('a, 'b)$  compact-op
by transfer simp
show  $-(a :: ('a, 'b)$  compact-op)  $+ a = 0$ 
for  $a :: ('a, 'b)$  compact-op
by transfer simp
show  $a - b = a + - b$ 
for  $a b :: ('a, 'b)$  compact-op
by transfer simp
show  $a *_C (x + y) = a *_C x + a *_C y$ 
for  $a :: \text{complex}$  and  $x y :: ('a, 'b)$  compact-op
by transfer (simp add: scaleC-add-right)
show  $(a + b) *_C x = a *_C x + b *_C x$ 
for  $a b :: \text{complex}$  and  $x :: ('a, 'b)$  compact-op
by transfer (simp add: scaleC-left.add)
show  $a *_C b *_C x = (a * b) *_C x$ 
for  $a b :: \text{complex}$  and  $x :: ('a, 'b)$  compact-op
by transfer simp
show  $1 *_C x = x$ 
for  $x :: ('a, 'b)$  compact-op
by transfer simp
show  $\text{dist } x y = \text{norm } (x - y)$ 
for  $x y :: ('a, 'b)$  compact-op
by transfer (simp add: dist-norm)
show  $a *_R (x + y) = a *_R x + a *_R y$ 
for  $a :: \text{real}$  and  $x y :: ('a, 'b)$  compact-op
by transfer (simp add: scaleR-right-distrib)
show  $(a + b) *_R x = a *_R x + b *_R x$ 
for  $a b :: \text{real}$  and  $x :: ('a, 'b)$  compact-op
by transfer (simp add: scaleR-left.add)
show  $a *_R b *_R x = (a * b) *_R x$ 
for  $a b :: \text{real}$  and  $x :: ('a, 'b)$  compact-op
by transfer simp
show  $1 *_R x = x$ 
for  $x :: ('a, 'b)$  compact-op
by transfer simp
show  $\text{sgn } x = \text{inverse } (\text{norm } x) *_R x$ 
for  $x :: ('a, 'b)$  compact-op
by transfer (simp add: sgn-div-norm)
show  $\text{uniformity} = (\text{INF } e \in \{0 < ..\}). \text{principal } \{(x, y). \text{dist } (x :: ('a, 'b)$  compact-op)  $y < e\}$ 
using uniformity-compact-op-def by blast
show  $\text{open } U = (\forall x \in U. \forall_F (x', y) \text{ in } \text{uniformity}. x' = x \longrightarrow y \in U)$ 
for  $U :: ('a, 'b)$  compact-op set
by (simp add: open-compact-op-def)
show  $(\text{norm } x = 0) \longleftrightarrow (x = 0)$ 
for  $x :: ('a, 'b)$  compact-op
by transfer simp
show  $\text{norm } (x + y) \leq \text{norm } x + \text{norm } y$ 
for  $x y :: ('a, 'b)$  compact-op
by transfer (use norm-triangle-ineq in blast)

```



```

show  $\text{norm } (a *_R x) = |a| * \text{norm } x$ 
for  $a :: \text{real}$  and  $x :: ('a, 'b) \text{compact-op}$ 
by transfer simp
show  $\text{norm } (a *_C x) = \text{cmod } a * \text{norm } x$ 
for  $a :: \text{complex}$  and  $x :: ('a, 'b) \text{compact-op}$ 
by transfer simp
qed
end

```

```

lemma from-compact-op-plus:  $\langle \text{from-compact-op } (a + b) = \text{from-compact-op } a + \text{from-compact-op } b \rangle$ 
by transfer simp

```

```

lemma from-compact-op-scaleC:  $\langle \text{from-compact-op } (c *_C a) = c *_C \text{from-compact-op } a \rangle$ 
by transfer simp

```

```

lemma from-compact-op-norm[simp]:  $\langle \text{norm } (\text{from-compact-op } a) = \text{norm } a \rangle$ 
by transfer simp

```

```

lemma compact-op-butterfly[simp]:  $\langle \text{compact-op } (\text{butterfly } x \ y) \rangle$ 
by (simp add: finite-rank-compact-op)

```

```

lift-definition butterfly-co ::  $\langle 'a :: \text{complex-normed-vector} \Rightarrow 'b :: \text{chilbert-space} \Rightarrow ('b, 'a) \text{compact-op} \rangle$ 
is butterfly
by simp

```

```

lemma butterfly-co-add-left:  $\langle \text{butterfly-co } (a + a') \ b = \text{butterfly-co } a \ b + \text{butterfly-co } a' \ b \rangle$ 
by transfer (rule butterfly-add-left)

```

```

lemma butterfly-co-add-right:  $\langle \text{butterfly-co } a \ (b + b') = \text{butterfly-co } a \ b + \text{butterfly-co } a \ b' \rangle$ 
by transfer (rule butterfly-add-right)

```

```

lemma butterfly-co-scaleR-left[simp]:  $\text{butterfly-co } (r *_R \psi) \ \varphi = r *_C \text{butterfly-co } \psi \ \varphi$ 
by transfer (rule butterfly-scaleR-left)

```

```

lemma butterfly-co-scaleR-right[simp]:  $\text{butterfly-co } \psi \ (r *_R \varphi) = r *_C \text{butterfly-co } \psi \ \varphi$ 
by transfer (rule butterfly-scaleR-right)

```

```

lemma butterfly-co-scaleC-left[simp]:  $\text{butterfly-co } (r *_C \psi) \ \varphi = r *_C \text{butterfly-co } \psi \ \varphi$ 
by transfer (rule butterfly-scaleC-left)

```

```

lemma butterfly-co-scaleC-right[simp]:  $\text{butterfly-co } \psi \ (r *_C \varphi) = \text{cnj } r *_C \text{butterfly-co } \psi \ \varphi$ 
by transfer (rule butterfly-scaleC-right)

```

```

lemma finite-rank-separating-on-compact-op:
fixes  $F \ G :: \langle ('a :: \text{chilbert-space}, 'b :: \text{chilbert-space}) \text{compact-op} \Rightarrow 'c :: \text{complex-normed-vector} \rangle$ 
assumes  $\langle \bigwedge x. \text{finite-rank } (\text{from-compact-op } x) \implies F \ x = G \ x \rangle$ 
assumes  $\langle \text{bounded-clinear } F \rangle$ 

```

```

assumes ⟨bounded-clinear G⟩
shows ⟨F = G⟩
proof –
define FG where ⟨FG x = F x - G x⟩ for x
from ⟨bounded-clinear F⟩ and ⟨bounded-clinear G⟩
have ⟨bounded-clinear FG⟩
  by (auto simp: FG-def[abs-def] intro!: bounded-clinear-sub)
then have contFG': ⟨continuous-map euclidean euclidean FG⟩
  by (simp add: Complex-Vector-Spaces.bounded-clinear.bounded-linear linear-continuous-on)
have ⟨continuous-on (Collect compact-op) (FG o Abs-compact-op)⟩
proof
  fix a :: 'a ⇒CL 'b and e :: real
  assume 0 < e and a-compact: a ∈ Collect compact-op
  have dist-rw: ⟨dist x' a = dist (Abs-compact-op x') (Abs-compact-op a)⟩ if ⟨compact-op x'⟩
for x'
  by (metis Abs-compact-op-inverse a-compact dist-compact-op.rep-eq mem-Collect-eq that)

  from ⟨bounded-clinear FG⟩
  have ⟨continuous-on UNIV FG⟩
    using contFG' continuous-map-iff-continuous2 by blast
  then have ⟨∃ d>0. ∀ x'. dist x' (Abs-compact-op a) < d ⟶ dist (FG x') (FG (Abs-compact-op
a)) ≤ e⟩
    using ⟨e > 0⟩ by (force simp: continuous-on-iff)
  then have ⟨∃ d>0. ∀ x'. compact-op x' ⟶ dist (Abs-compact-op x') (Abs-compact-op a) <
d ⟶
    dist (FG (Abs-compact-op x')) (FG (Abs-compact-op a)) ≤ e⟩
    by blast
  then show ∃ d>0. ∀ x'∈Collect compact-op. dist x' a < d ⟶ dist ((FG o Abs-compact-op)
x') ((FG o Abs-compact-op) a) ≤ e
    by (simp add: dist-rw o-def)
qed
then have contFG: ⟨continuous-on (closure (Collect finite-rank)) (FG o Abs-compact-op)⟩
  by (auto simp: compact-op-finite-rank[abs-def])

have FG0: ⟨finite-rank a ⟹ (FG o Abs-compact-op) a = 0⟩ for a
  by (metis (no-types, lifting) Abs-compact-op-inverse FG-def assms(1) closure-subset comp-apply
compact-op-finite-rank eq-iff-diff-eq-0 mem-Collect-eq subset-eq)

have ⟨(FG o Abs-compact-op) a = 0⟩ if ⟨compact-op a⟩ for a
  using contFG FG0
  by (rule continuous-constant-on-closure) (use that in ⟨auto simp: compact-op-finite-rank⟩)

then have ⟨FG a = 0⟩ for a
  by (metis Abs-compact-op-cases comp-apply mem-Collect-eq)

then show ⟨F = G⟩
  by (auto simp: FG-def[abs-def] fun-eq-iff)
qed

```

lemma *trunc-ell2-as-Proj*: $\langle \text{trunc-ell2 } S \ \psi = \text{Proj } (\text{ccspan } (\text{ket } ' S)) \ \psi \rangle$
proof (*rule cinner-ket-eqI*)
fix x
have $*$: $\langle \text{Proj } (\text{ccspan } (\text{ket } ' S)) (\text{ket } x) = 0 \rangle$ **if** $\langle x \notin S \rangle$
by (*auto intro!*; *Proj-0-compl mem-ortho-ccspanI simp: that*)
have $\langle \text{ket } x \cdot_C \text{trunc-ell2 } S \ \psi = \text{of-bool } (x \in S) * (\text{ket } x \cdot_C \psi) \rangle$
by (*simp add: cinner-ket-left trunc-ell2.rep-eq*)
also have $\langle \dots = \text{Proj } (\text{ccspan } (\text{ket } ' S)) (\text{ket } x) \cdot_C \psi \rangle$
by (*cases* $\langle x \in S \rangle$) (*auto simp add: * ccspan-superset' Proj-fixes-image*)
also have $\langle \dots = \text{ket } x \cdot_C (\text{Proj } (\text{ccspan } (\text{ket } ' S)) *_V \psi) \rangle$
by (*simp add: adj-Proj flip: cinner-adj-left*)
finally show $\langle \text{ket } x \cdot_C \text{trunc-ell2 } S \ \psi = \text{ket } x \cdot_C (\text{Proj } (\text{ccspan } (\text{ket } ' S)) *_V \psi) \rangle$.
qed

lemma *unitary-between-bij-betw*:
assumes $\langle \text{is-onb } A \rangle \langle \text{is-onb } B \rangle$
shows $\langle \text{bij-betw } ((*_V) (\text{unitary-between } A \ B)) \ A \ B \rangle$
using *bij-between-bases-bij[OF assms]*
by (*rule bij-betw-cong[THEN iffD1, rotated]*)
(simp add: assms(1) assms(2) unitary-between-apply)

lemma *tendsto-finite-subsets-at-top-image*:
assumes $\langle \text{inj-on } g \ X \rangle$
shows $\langle (f \longrightarrow x) (\text{finite-subsets-at-top } (g \ ' X)) \longleftrightarrow ((\lambda S. f (g \ ' S)) \longrightarrow x) (\text{finite-subsets-at-top } X) \rangle$
by (*simp add: filterlim-def assms o-def*)
flip: filtermap-image-finite-subsets-at-top filtermap-compose

lemma *Proj-onb-limit*:
shows $\langle \text{is-onb } A \implies ((\lambda S. \text{Proj } (\text{ccspan } S) \ \psi) \longrightarrow \psi) (\text{finite-subsets-at-top } A) \rangle$
proof –
have *main*: $\langle ((\lambda S. \text{Proj } (\text{ccspan } S) \ \psi) \longrightarrow \psi) (\text{finite-subsets-at-top } A) \rangle$ **if** $\langle \text{is-onb } A \rangle$
for $\psi :: \langle 'b :: \{\text{hilbert-space, not-singleton}\} \rangle$ **and** A
proof –
define U **where** $\langle U = \text{unitary-between } (\text{ell2-to-hilbert} * \ ' A) (\text{range } \text{ket}) \rangle$
have [*simp*]: $\langle \text{unitary } U \rangle$
by (*simp add: U-def that unitary-between-unitary unitary-image-onb*)
have *lim1*: $\langle ((\lambda S. \text{trunc-ell2 } S (U *_V \text{ell2-to-hilbert} * *_V \psi)) \longrightarrow U *_V \text{ell2-to-hilbert} * *_V \psi) (\text{finite-subsets-at-top } \text{UNIV}) \rangle$
by (*rule trunc-ell2-lim-at-UNIV*)
have *lim2*: $\langle ((\lambda S. \text{ell2-to-hilbert} *_V U * *_V \text{trunc-ell2 } S (U *_V \text{ell2-to-hilbert} * *_V \psi)) \longrightarrow \text{ell2-to-hilbert} *_V U * *_V U *_V \text{ell2-to-hilbert} * *_V \psi) (\text{finite-subsets-at-top } \text{UNIV}) \rangle$
by (*intro cblinfun.tendsto lim1 auto*)
have $*$: $\langle \text{ell2-to-hilbert} *_V U * *_V \text{trunc-ell2 } S (U *_V \text{ell2-to-hilbert} * *_V \psi) \rangle$
 $= \text{Proj } (\text{ccspan } ((\text{ell2-to-hilbert } o \ U * o \ \text{ket } ' S)) \ \psi) \langle \text{is } \langle ?lhs = ?rhs \rangle \rangle$ **for** S
proof –
have $\langle ?lhs = (\text{sandwich } \text{ell2-to-hilbert} *_V \ \text{sandwich } (U *) *_V \ \text{Proj } (\text{ccspan } (\text{ket } ' S))) *_V$

```

ψ⟩
  by (simp add: trunc-ell2-as-Proj sandwich-apply)
also have ⟨... = Proj (ell2-to-hilbert *S U* *S ccspace (ket ' S)) *V ψ⟩
  by (simp add: Proj-sandwich)
also have ⟨... = Proj (ccspan (ell2-to-hilbert ' U* ' ket ' S)) *V ψ⟩
  by (simp add: cblinfun-image-ccspan)
also have ⟨... = ?rhs⟩
  by (simp add: image-comp)
finally show ?thesis
  by -
qed
have **: ⟨ell2-to-hilbert *V U* *V U *V ell2-to-hilbert* *V ψ = ψ⟩
  by (simp add: lift-cblinfun-comp[OF unitaryD1] lift-cblinfun-comp[OF unitaryD2])
have **: ⟨range (ell2-to-hilbert o U* o ket) = A⟩ (is ⟨?lhs = -⟩)
proof -
  have ⟨bij-betw U (ell2-to-hilbert* ' A) (range ket)⟩
    by (auto intro!: unitary-between-bij-betw that unitary-image-onb simp add: U-def)
  then have bijUadj: ⟨bij-betw (U*) (range ket) (ell2-to-hilbert* ' A)⟩
  by (metis ⟨unitary U⟩ bij-betw-imp-surj-on inj-imp-bij-betw-inv unitary-adj-inv unitary-inj)
  have ⟨?lhs = ell2-to-hilbert ' U* ' range ket⟩
    by (simp add: image-comp)
  also from this and bijUadj have ⟨... = ell2-to-hilbert ' (ell2-to-hilbert* ' A)⟩
    by (metis bij-betw-imp-surj-on)
  also have ⟨... = A⟩
    by (metis image-inv-f-f unitary-adj unitary-adj-inv unitary-ell2-to-hilbert unitary-inj)
  finally show ?thesis
    by -
qed
from lim2 have lim3: ⟨((λS. Proj (ccspan ((ell2-to-hilbert o U* o ket) ' S)) ψ) → ψ)
(finite-subsets-at-top UNIV)⟩
  unfolding * ** by -
  then have lim4: ⟨((λS. Proj (ccspan S) ψ) → ψ) (finite-subsets-at-top (range (ell2-to-hilbert
o U* o ket)))⟩
  by (rule tendsto-finite-subsets-at-top-image[THEN iffD2, rotated])
  (intro inj-compose unitary-inj unitary-ell2-to-hilbert unitary-adj[THEN iffD2] ⟨unitary
U⟩ inj-ket)
  then show ?thesis
    unfolding *** by -
qed
assume ⟨is-onb A⟩
show ?thesis
proof (cases ⟨class.not-singleton TYPE('a)⟩)
  case True
  show ?thesis
    using hilbert-space-class.hilbert-space-axioms True ⟨is-onb A⟩
    by (rule main[internalize-sort' 'b2])
next
  case False
  then have ⟨ψ = 0⟩

```

```

    by (rule not-not-singleton-zero)
  then show ?thesis
    by simp
qed
qed

```

```

lemma is-ortho-setD:
  assumes is-ortho-set S x ∈ S y ∈ S x ≠ y
  shows x ⋅C y = 0
  using assms unfolding is-ortho-set-def by blast

```

```

lemma finite-rank-dense-compact:
  fixes A :: ⟨'a::hilbert-space set⟩ and B :: ⟨'b::hilbert-space set⟩
  assumes ⟨is-onb A⟩ and ⟨is-onb B⟩
  shows ⟨closure (cspan ((λ(ξ,η). butterfly ξ η) ' (A × B))) = Collect compact-op⟩
proof (rule Set.equalityI)
  show ⟨closure (cspan ((λ(ξ,η). butterfly ξ η) ' (A × B))) ⊆ Collect compact-op⟩
  proof -
    have ⟨closure (cspan ((λ(ξ,η). butterfly ξ η) ' (A × B))) ⊆ closure (Collect finite-rank)⟩
    proof (rule closure-mono; safe)
      fix x assume x ∈ cspan ((λ(ξ, η). butterfly ξ η) ' (A × B))
      thus finite-rank x
        by (induction rule: complex-vector.span-induct-alt) auto
    qed
    also have ⟨... = Collect compact-op⟩
      by (simp add: Set.set-eqI compact-op-finite-rank)
    finally show ?thesis
      by -
  qed
  show ⟨Collect compact-op ⊆ closure (cspan ((λ(ξ,η). butterfly ξ η) ' (A × B)))⟩
  proof -
    have ⟨Collect (compact-op :: 'b⇒CL'a ⇒ -) = closure (cspan (Collect rank1))⟩
      by (simp add: compact-op-finite-rank[abs-def] finite-rank-def[abs-def])
    also have ⟨... ⊆ closure (cspan (closure (cspan ((λ(ξ,η). butterfly ξ η) ' (A × B))))⟩
    proof (rule closure-mono, rule complex-vector.span-mono, rule subsetI)
      fix x :: ⟨'b ⇒CL 'a⟩ assume ⟨x ∈ Collect rank1⟩
      then obtain a b where xab: ⟨x = butterfly a b⟩
        using rank1-iff-butterfly by fastforce
      define f where ⟨f F G = butterfly (Proj (ccspan F) a) (Proj (ccspan G) b)⟩ for F G
      have lim: ⟨(case-prod f ⟶ x) (finite-subsets-at-top A ×F finite-subsets-at-top B)⟩
    proof (rule tendstoI, subst dist-norm)
      fix e :: real assume ⟨e > 0⟩
      define d where ⟨d = (if norm a = 0 ∧ norm b = 0 then 1
        else e / (max (norm a) (norm b) / 4)⟩
      have [simp]: d > 0
        unfolding d-def using ⟨e > 0⟩
        by (auto intro!: divide-pos-pos simp: less-max-iff-disj)
      have d: ⟨norm a * d + norm a * d + norm b * d < e⟩
    proof -

```

have $\langle x * d \leq e / 4 \rangle$ **if** $x \in \{ \text{norm } a, \text{norm } b \}$ **for** x
proof (*cases* $x = 0$)
 case *False*
 have $d : d = e / (\max (\text{norm } a) (\text{norm } b)) / 4$
 using *False* x **by** (*auto simp: d-def*)
 have $d \leq e / x / 4$
 unfolding d **by** (*intro divide-left-mono divide-right-mono*)
 (*use* $x \langle d > 0 \rangle \langle e > 0 \rangle$ *False in* *auto simp: less-max-iff-disj*)
 thus *?thesis*
 using *False* x **by** (*auto simp: field-simps*)
qed (*use* $\langle e > 0 \rangle$ **in** *auto*)
hence $\text{norm } a * d \leq e / 4$ $\text{norm } b * d \leq e / 4$
 by *blast+*
hence $\langle \text{norm } a * d + \text{norm } a * d + \text{norm } b * d \leq 3 * e / 4 \rangle$
 by *linarith*
also have $\langle \dots < e \rangle$
 by (*simp add:* $\langle 0 < e \rangle$)
finally show *?thesis* .
qed
from *Proj-onb-limit*[**where** $\psi=a$, *OF* *assms(1)*]
have $\langle \forall_F F$ *in* *finite-subsets-at-top* $A. \text{norm} (\text{Proj} (\text{ccspan } F) a - a) < d \rangle$
 by (*metis* *Lim-null* $\langle 0 < d \rangle$ *order-tendstoD(2)* *tendsto-norm-zero-iff*)
moreover from *Proj-onb-limit*[**where** $\psi=b$, *OF* *assms(2)*]
have $\langle \forall_F G$ *in* *finite-subsets-at-top* $B. \text{norm} (\text{Proj} (\text{ccspan } G) b - b) < d \rangle$
 by (*metis* *Lim-null* $\langle 0 < d \rangle$ *order-tendstoD(2)* *tendsto-norm-zero-iff*)
ultimately have *FG-close*: $\langle \forall_F (F,G)$ *in* *finite-subsets-at-top* $A \times_F$ *finite-subsets-at-top*

B.

$\text{norm} (\text{Proj} (\text{ccspan } F) a - a) < d \wedge \text{norm} (\text{Proj} (\text{ccspan } G) b - b) < d$
 unfolding *case-prod-beta*
 by (*rule* *eventually-prodI*)
have *fFG-dist*: $\langle \text{norm} (f F G - x) < e \rangle$
 if $\langle \text{norm} (\text{Proj} (\text{ccspan } F) a - a) < d \rangle$ **and** $\langle \text{norm} (\text{Proj} (\text{ccspan } G) b - b) < d \rangle$
 and $\langle F \subseteq A \rangle$ **and** $\langle G \subseteq B \rangle$ **for** $F G$
proof –
 have *a-split*: $\langle a = \text{Proj} (\text{ccspan } F) *_{\mathcal{V}} a + \text{Proj} (\text{ccspan} (A-F)) *_{\mathcal{V}} a \rangle$
 proof –
 have A : *is-ortho-set* A $\text{ccspan } A = \top$
 using *assms* **unfolding** *is-onb-def* **by** *auto*
 have $\text{Proj} (\text{ccspan} (F \cup A)) = \text{Proj} (\text{ccspan } F) + \text{Proj} (\text{ccspan} (A-F))$
 by (*subst* *Proj-orthog-ccspan-union* [*symmetric*])
 (*use* *that* **in** *auto* *intro!*: *is-ortho-setD*[*OF* $A(1)$])
 also have $F \cup A = A$
 using *that* **by** *blast*
 finally show *?thesis*
 using $A(2)$ **by** (*simp* *flip: cblinfun.add-left*)
qed

 have *b-split*: $\langle b = \text{Proj} (\text{ccspan } G) *_{\mathcal{V}} b + \text{Proj} (\text{ccspan} (B-G)) *_{\mathcal{V}} b \rangle$
 proof –

```

have B: is-ortho-set B ccspace B = ⊤
  using assms unfolding is-onb-def by auto
have Proj (ccspace (G ∪ B)) = Proj (ccspace G) + Proj (ccspace (B-G))
  by (subst Proj-ortho-ccspace-union [symmetric])
    (use that in ⟨auto intro!: is-ortho-setD[OF B(1)]⟩)
also have G ∪ B = B
  using that by blast
finally show ?thesis
  using B(2) by (simp flip: cblinfun.add-left)
qed

have n1: ⟨norm (f F (B-G)) ≤ norm a * d⟩ for F
proof -
  have ⟨norm (f F (B-G)) ≤ norm a * norm (Proj (ccspace (B-G)) b)⟩
  by (auto intro!: mult-right-mono is-Proj-reduces-norm simp add: f-def norm-butterfly)
  also have ⟨... ≤ norm a * norm (Proj (ccspace G) b - b)⟩
  by (metis add-diff-cancel-left' b-split less-eq-real-def norm-minus-commute)
  also have ⟨... ≤ norm a * d⟩
  by (meson less-eq-real-def mult-left-mono norm-ge-zero that(2))
  finally show ?thesis
  by -
qed

have n2: ⟨norm (f (A-F) G) ≤ norm b * d⟩ for G
proof -
  have ⟨norm (f (A-F) G) ≤ norm b * norm (Proj (ccspace (A-F)) a)⟩
  by (auto intro!: mult-right-mono is-Proj-reduces-norm simp add: f-def norm-butterfly
    mult.commute)
  also have ⟨... ≤ norm b * norm (Proj (ccspace F) a - a)⟩
  by (smt (verit, best) a-split add-diff-cancel-left' minus-diff-eq norm-minus-cancel)
  also have ⟨... ≤ norm b * d⟩
  by (meson less-eq-real-def mult-left-mono norm-ge-zero that(1))
  finally show ?thesis
  by -
qed

have ⟨norm (f F G - x) = norm (- f F (B-G) - f (A-F) (B-G) - f (A-F) G)⟩
  unfolding xab
  by (subst a-split, subst b-split)
    (simp add: f-def butterfly-add-right butterfly-add-left)
also have ⟨... ≤ norm (f F (B-G)) + norm (f (A-F) (B-G)) + norm (f (A-F)
G)⟩
  by (smt (verit, best) norm-minus-cancel norm-triangle-ineq4)
also have ⟨... ≤ norm a * d + norm a * d + norm b * d⟩
  using n1 n2
  by (meson add-mono-thms-linordered-semiring(1))
also have ⟨... < e⟩
  by (fact d)
finally show ?thesis
  by -
qed

```

have $\forall_F (F, G)$ in *finite-subsets-at-top* $A \times_F$ *finite-subsets-at-top* B .
 $(\text{finite } F \wedge F \subseteq A) \wedge \text{finite } G \wedge G \subseteq B$
unfolding *case-prod-unfold* **by** (*intro eventually-prodI*) *auto*
thus $\langle \forall_F FG$ in *finite-subsets-at-top* $A \times_F$ *finite-subsets-at-top* B .
 $\text{norm } ((\text{case } FG \text{ of } (F, G) \Rightarrow f F G) - x) < e \rangle$
using *FG-close* **by** *eventually-elim* (*use fFG-dist in auto*)
qed
have *nontriv*: $\langle \text{finite-subsets-at-top } A \times_F \text{ finite-subsets-at-top } B \neq \perp \rangle$
by (*simp add: prod-filter-eq-bot*)
have *inside*: $\langle \forall_F x$ in *finite-subsets-at-top* $A \times_F$ *finite-subsets-at-top* B .
 $\text{case-prod } f x \in \text{cspan } ((\lambda(\xi, \eta). \text{butterfly } \xi \eta) ' (A \times B)) \rangle$
proof (*rule eventually-mp*[**where** $P = \langle \lambda(F, G). \text{finite } F \wedge \text{finite } G \rangle$])
show $\langle \forall_F (F, G)$ in *finite-subsets-at-top* $A \times_F$ *finite-subsets-at-top* B . *finite* $F \wedge$ *finite* $G \rangle$
by (*smt* (*verit*) *case-prod-conv eventually-finite-subsets-at-top-weakI eventually-prod-filter*)
have *f-in-span*: $\langle f F G \in \text{cspan } ((\lambda(\xi, \eta). \text{butterfly } \xi \eta) ' (A \times B)) \rangle$ **if** [*simp*]: $\langle \text{finite } F \rangle$
 $\langle \text{finite } G \rangle$ **and** $\langle F \subseteq A \rangle \langle G \subseteq B \rangle$ **for** $F G$
proof –
have $\langle \text{Proj } (\text{ccspan } F) a \in \text{cspan } F \rangle$
by (*metis* *Proj-range cblinfun-apply-in-image ccspan-finite that(1)*)
then obtain r **where** *ProjFsum*: $\langle \text{Proj } (\text{ccspan } F) a = (\sum_{x \in F}. r x *_C x) \rangle$
using *complex-vector.span-finite[OF finite F]* **by** *auto*
have $\langle \text{Proj } (\text{ccspan } G) b \in \text{cspan } G \rangle$
by (*metis* *Proj-range cblinfun-apply-in-image ccspan-finite that(2)*)
then obtain s **where** *ProjGsum*: $\langle \text{Proj } (\text{ccspan } G) b = (\sum_{x \in G}. s x *_C x) \rangle$
using *complex-vector.span-finite[OF finite G]* **by** *auto*
have $\langle \text{butterfly } \xi \eta \in (\lambda(\xi, \eta). \text{butterfly } \xi \eta) ' (A \times B) \rangle$
if $\langle \eta \in G \rangle$ **and** $\langle \xi \in F \rangle$ **for** $\eta \xi$
using $\langle F \subseteq A \rangle \langle G \subseteq B \rangle$ **that** **by** *auto*
then show *?thesis*
by (*auto intro!*: *complex-vector.span-sum complex-vector.span-scale*
complex-vector.span-base[**where** $a = \langle \text{butterfly } - - \rangle$]
simp add: f-def ProjFsum ProjGsum butterfly-sum-left butterfly-sum-right)
qed
have $\forall_F (F, G)$ in *finite-subsets-at-top* $A \times_F$ *finite-subsets-at-top* B .
 $(\text{finite } F \wedge F \subseteq A) \wedge \text{finite } G \wedge G \subseteq B$
unfolding *case-prod-unfold* **by** (*intro eventually-prodI*) *auto*
thus $\langle \forall_F x$ in *finite-subsets-at-top* $A \times_F$ *finite-subsets-at-top* B .
 $(\text{case } x \text{ of } (F, G) \Rightarrow \text{finite } F \wedge \text{finite } G) \longrightarrow (\text{case } x \text{ of } (F, G) \Rightarrow f F G) \in$
 $\text{cspan } ((\lambda(\xi, \eta). \text{butterfly } \xi \eta) ' (A \times B)) \rangle$
by *eventually-elim* (*auto intro!*: *f-in-span*)
qed
show $\langle x \in \text{closure } (\text{cspan } ((\lambda(\xi, \eta). \text{butterfly } \xi \eta) ' (A \times B))) \rangle$
using *lim nontriv inside* **by** (*rule limit-in-closure*)
qed
also have $\langle \dots = \text{closure } (\text{cspan } ((\lambda(\xi, \eta). \text{butterfly } \xi \eta) ' (A \times B))) \rangle$
by (*simp add: complex-vector.span-eq-iff*[*THEN iffD2*])
finally show *?thesis*
by –
qed

qed

lemma *compact-op-comp-left*: $\langle \text{compact-op } (a \text{ } o_{CL} \text{ } b) \rangle$ **if** $\langle \text{compact-op } a \rangle$
for $a \text{ } b :: \langle \text{--} :: \text{chilbert-space} \Rightarrow_{CL} \text{--} :: \text{chilbert-space} \rangle$

proof –

from *that* **have** $\langle a \in \text{closure } (\text{Collect finite-rank}) \rangle$

using *compact-op-finite-rank* **by** *blast*

then **have** $\langle a \text{ } o_{CL} \text{ } b \in (\lambda a. a \text{ } o_{CL} \text{ } b) \text{ ' closure } (\text{Collect finite-rank}) \rangle$

by *blast*

also **have** $\langle \dots \subseteq \text{closure } ((\lambda a. a \text{ } o_{CL} \text{ } b) \text{ ' Collect finite-rank}) \rangle$

by (*auto intro!*: *closure-bounded-linear-image-subset bounded-clinear.bounded-linear bounded-clinear-cblinfun-compo*)

also **have** $\langle \dots \subseteq \text{closure } (\text{Collect finite-rank}) \rangle$

by (*auto intro!*: *closure-mono finite-rank-comp-left*)

finally **show** $\langle \text{compact-op } (a \text{ } o_{CL} \text{ } b) \rangle$

using *compact-op-finite-rank* **by** *blast*

qed

lemma *compact-op-eigenspace-finite-dim*:

fixes $a :: \langle 'a \Rightarrow_{CL} 'a :: \text{chilbert-space} \rangle$

assumes $\langle \text{compact-op } a \rangle$

assumes $\langle e \neq 0 \rangle$

shows $\langle \text{finite-dim-ccsubspace } (\text{eigenspace } e \text{ } a) \rangle$

proof –

define S **where** $\langle S = \text{space-as-set } (\text{eigenspace } e \text{ } a) \rangle$

obtain B **where** $\langle \text{ccspan } B = \text{eigenspace } e \text{ } a \rangle$ **and** $\langle \text{is-ortho-set } B \rangle$

and *norm-B*: $\langle x \in B \implies \text{norm } x = 1 \rangle$ **for** x

using *orthonormal-subspace-basis-exists* [**where** $S = \langle \{ \} \rangle$ **and** $V = \langle \text{eigenspace } e \text{ } a \rangle$]

by (*auto simp*: *S-def*)

then **have** *span-BS*: $\langle \text{closure } (\text{cspan } B) = S \rangle$

by (*metis S-def ccspan.rep-eq*)

have $\langle \text{finite } B \rangle$

proof (*rule ccontr*)

assume $\langle \text{infinite } B \rangle$

then **obtain** $b :: \langle \text{nat} \Rightarrow 'a \rangle$ **where** *range-b*: $\langle \text{range } b \subseteq B \rangle$ **and** $\langle \text{inj } b \rangle$

by (*meson infinite-countable-subset*)

define f **where** $\langle f \text{ } n = a \text{ } (b \text{ } n) \rangle$ **for** n

have *range-f*: $\langle \text{range } f \subseteq \text{closure } (a \text{ ' cball } 0 \text{ } 1) \rangle$

using *norm-B range-b*

by (*auto intro!*: *closure-subset*[*THEN subsetD*] *imageI simp*: *f-def*)

from $\langle \text{compact-op } a \rangle$ **have** *compact*: $\langle \text{compact } (\text{closure } (a \text{ ' cball } 0 \text{ } 1)) \rangle$

using *compact-op-def2* **by** *blast*

obtain $l \text{ } r$ **where** $\langle \text{strict-mono } r \rangle$ **and** *fr-lim*: $\langle (f \text{ } o \text{ } r) \longrightarrow l \rangle$

using *range-f compact*[*unfolded compact-def, rule-format, of f*]

by *fast*

define $d :: \text{real}$ **where** $\langle d = \text{cmod } e * \text{sqrt } 2 \rangle$

from $\langle e \neq 0 \rangle$ **have** $\langle d > 0 \rangle$

by (*auto intro!*: *Rings.linordered-semiring-strict-class.mult-pos-pos simp*: *d-def*)

have *aux*: $\langle \exists n \geq N. P \text{ } n \rangle$ **if** $\langle P \text{ } (\text{Suc } N) \rangle$ **for** $P \text{ } N$

```

    using Suc-n-not-le-n nat-le-linear that by blast
  have <dist (f (r n)) (f (r (Suc n))) = d> for n
  proof -
    have ortho: <is-orthogonal (b (r n)) (b (r (Suc n)))>
    proof -
      have <b (r n) ≠ b (r (Suc n))>
        by (metis Suc-n-not-n <inj b> <strict-mono r> injD strict-mono-eq)
      moreover from range-b have <b (r n) ∈ B> and <b (r (Suc n)) ∈ B>
        by fast+
      ultimately show ?thesis
        using <is-ortho-set B>
        by (auto intro!: simp: is-ortho-set-def)
    qed
  have normb: <norm (b n) = 1> for n
    by (metis <inj b> image-subset-iff inj-image-mem-iff norm-B range-b range-eqI)
  have <f (r n) = e *C b (r n)> for n
  proof -
    from range-b span-BS
    have <b (r n) ∈ S>
      using complex-vector.span-superset closure-subset
      by (blast dest: range-subsetD[where i = <b n>])
    then show ?thesis
      by (auto intro!: dest!: eigenspace-memberD simp: S-def f-def)
    qed
  then have <(dist (f (r n)) (f (r (Suc n))))2 = (cmod e * dist (b (r n)) (b (r (Suc n))))2>
    by (simp add: dist-norm flip: scaleC-diff-right)
  also from ortho have <... = (cmod e * sqrt 2)2>
    by (simp add: dist-norm polar-identity-minus power-mult-distrib normb)
  finally show ?thesis
    by (simp add: d-def)
  qed
  with <d > 0> have <¬ Cauchy (f o r)>
    by (auto intro!: exI[of - <d/2>] aux
      simp: Cauchy-altdef2 dist-commute simp del: less-divide-eq-numeral1)
  with fr-lim show False
    using LIMSEQ-imp-Cauchy by blast
  qed
  with span-BS show ?thesis
    using S-def cspan-finite-dim finite-dim-ccsubspace.rep-eq by fastforce
  qed

lemma eigenvalue-in-the-limit-compact-op:
  — [2], Proposition II.4.14
  assumes <compact-op T>
  assumes <l ≠ 0>
  assumes normh: <∧n. norm (h n) = 1>
  assumes Tl-lim: <(λn. (T - l *C id-cblinfun) (h n)) → 0>
  shows <l ∈ eigenvalues T>
  proof -

```

```

from assms(1)
have compact-Tball: ⟨compact (closure (T ‘cball 0 1))⟩
  using compact-op-def2 by blast
have ⟨T (h n) ∈ closure (T ‘cball 0 1)⟩ for n
  by (smt (z3) assms(3) closure-subset image-subset-iff mem-cball-0)
then obtain n f where lim-Thn: ⟨(λk. T (h (n k))) ⟶ f⟩ and ⟨strict-mono n⟩
  using compact-Tball[unfolded compact-def, rule-format, where f=⟨T o h⟩, unfolded o-def]
  by fast
have lThk-lim: ⟨(λk. (l *C id-cblinfun - T) (h (n k))) ⟶ 0⟩
proof -
  have ⟨(λn. (l *C id-cblinfun - T) (h n)) ⟶ 0⟩
    using Tl-lim[THEN tendsto-minus]
    by (simp add: cblinfun.diff-left)
  with ⟨strict-mono n⟩ show ?thesis
    by (rule LIMSEQ-subseq-LIMSEQ[unfolded o-def, rotated])
qed
have ⟨h (n k) = inverse l *C ((l *C id-cblinfun - T) (h (n k)) + T (h (n k)))⟩ for k
  by (metis assms(2) cblinfun.real.add-left cblinfun.scaleC-left diff-add-cancel divideC-field-splits-simps-1 (5)
id-cblinfun.rep-eq scaleC-zero-right)
moreover have ⟨... ⟶ inverse l *C (0 + f)⟩
  by (intro tendsto-intros lThk-lim lim-Thn)
ultimately have lim-hn: ⟨(λk. h (n k)) ⟶ inverse l *C f⟩
  by simp
have ⟨f ≠ 0⟩
proof -
  from lim-hn have ⟨(λk. norm (h (n k))) ⟶ norm (inverse l *C f)⟩
    by (rule isCont-tendsto-compose[unfolded o-def, rotated]) fastforce
  moreover have ⟨(λ-. 1) ⟶ 1⟩
    by simp
  ultimately have ⟨norm (inverse l *C f) = 1⟩
    unfolding normh
    using LIMSEQ-unique by blast
  then show ⟨f ≠ 0⟩
    by force
qed
from lim-hn have ⟨(λk. T (h (n k))) ⟶ T (inverse l *C f)⟩
  by (rule isCont-tendsto-compose[rotated]) force
with lim-Thn have ⟨f = T (inverse l *C f)⟩
  using LIMSEQ-unique by blast
with ⟨l ≠ 0⟩ have ⟨l *C f = T f⟩
  by (metis cblinfun.scaleC-right divideC-field-simps(2))
with ⟨f ≠ 0⟩ show ⟨l ∈ eigenvalues T⟩
  by (auto intro!: eigenvaluesI[where h=f])
qed

```

lemma *norm-is-eigenvalue*:

— [2], Proposition II.5.9

fixes *a* :: ⟨'*a* ⇒_{*C*} *L* 'a::{*not-singleton, chilbert-space*}⟩

```

assumes ⟨compact-op a⟩
assumes ⟨selfadjoint a⟩
shows ⟨norm a ∈ eigenvalues a ∨ - norm a ∈ eigenvalues a⟩
proof -
  wlog ⟨a ≠ 0⟩
  using negation by auto
obtain h e where h-lim: ⟨(λi. h i •C a (h i)) ⟶ e⟩ and normh: ⟨norm (h i) = 1⟩
  and norme: ⟨cmod e = norm a⟩ for i
proof atomize-elim
  have sgn-cmod: ⟨sgn x * cmod x = x⟩ for x
    by (simp add: complex-of-real-cmod sgn-mult-abs)
  from cblinfun-norm-is-Sup-cinner[OF ⟨selfadjoint a⟩]
obtain f where range-f: ⟨range f ⊆ ((λψ. cmod (ψ •C (a *V ψ))) ‘ {ψ. norm ψ = 1})⟩
  and f-lim: ⟨f ⟶ norm a⟩
    by (atomize-elim, rule is-Sup-imp-ex-tendsto) (auto simp: ex-norm1-not-singleton)
obtain h0 where normh0: ⟨norm (h0 i) = 1⟩ and f-h0: ⟨f i = cmod (h0 i •C a (h0 i))⟩
for i
  by (atomize-elim, rule choice2) (use range-f in auto)
from f-h0 f-lim have h0lim-cmod: ⟨(λi. cmod (h0 i •C a (h0 i))) ⟶ norm a⟩
  by presburger
have sgn-sphere: ⟨sgn (h0 i •C a (h0 i)) ∈ insert 0 (sphere 0 1)⟩ for i
  using normh0 by (auto intro!: left-inverse simp: sgn-div-norm)
have compact: ⟨compact (insert 0 (sphere (0::complex) 1))⟩
  by simp
obtain r l where ⟨strict-mono r⟩ and l-sphere: ⟨l ∈ insert 0 (sphere 0 1)⟩
  and h0lim-sgn: ⟨((λi. sgn (h0 i •C a (h0 i))) ∘ r) ⟶ l⟩
  using compact[unfolded compact-def, rule-format, OF sgn-sphere]
  by fast
define h and e where ⟨h i = h0 (r i)⟩ and ⟨e = l * norm a⟩ for i
have hlim-cmod: ⟨(λi. cmod (h i •C a (h i))) ⟶ norm a⟩
  using LIMSEQ-subseq-LIMSEQ[OF h0lim-cmod ⟨strict-mono r⟩]
  unfolding h-def o-def by auto
with h0lim-sgn have ⟨(λi. sgn (h i •C a (h i)) * cmod (h i •C a (h i))) ⟶ e⟩
  by (auto intro!: tendsto-mult tendsto-of-real simp: o-def h-def e-def)
then have hlim: ⟨(λi. h i •C a (h i)) ⟶ e⟩
  by (simp add: sgn-cmod)
have ⟨e ≠ 0⟩
proof (rule ccontr, unfold not-not)
  assume ⟨e = 0⟩
  from hlim have ⟨(λi. cmod (h i •C a (h i))) ⟶ cmod e⟩
    by (rule tendsto-compose[where g=cmod, rotated])
    (smt (verit, del-Insts) ⟨e = 0⟩ diff-zero dist-norm metric-LIM-imp-LIM
      norm-ge-zero norm-zero real-norm-def tendsto-ident-at)
  with ⟨e = 0⟩ hlim-cmod have ⟨norm a = 0⟩
    using LIMSEQ-unique by fastforce
  with ⟨a ≠ 0⟩ show False
    by simp
qed
then have norme: ⟨norm e = norm a⟩

```

```

    using l-sphere by (simp add: e-def norm-mult)
  show  $\langle \exists h e. (\lambda i. h i \cdot_C (a *_V h i)) \longrightarrow e \wedge (\forall i. \text{norm } (h i) = 1) \wedge \text{cmod } e = \text{norm } a \rangle$ 
    using norme normh0 hlim
    by (auto intro!: exI[of - h] exI[of - e] simp: h-def)
qed
have  $\langle e \in \mathbb{R} \rangle$ 
proof -
  from h-lim[THEN tendsto-Im]
  have *:  $\langle (\lambda i. \text{Im } (h i \cdot_C a (h i))) \longrightarrow \text{Im } e \rangle$ 
    by -
  have **:  $\langle \text{Im } (h i \cdot_C a (h i)) = 0 \rangle$  for  $i$ 
    using assms(2) selfadjoint-def cinner-selfadjoint-real complex-is-Real-iff by auto
  have  $\langle \text{Im } e = 0 \rangle$ 
    using - * by (rule tendsto-unique) (use ** in auto)
  then show ?thesis
    using complex-is-Real-iff by presburger
qed
define  $e'$  where  $\langle e' = \text{Re } e \rangle$ 
with  $\langle e \in \mathbb{R} \rangle$  have  $ee'$ :  $\langle e = \text{of-real } e' \rangle$ 
  by simp
have  $\langle e' \in \text{eigenvalues } a \rangle$ 
proof -
  have [trans]:  $\langle f \longrightarrow 0 \rangle$  if  $\langle \bigwedge x. f x \leq g x \rangle$  and  $\langle g \longrightarrow 0 \rangle$  and  $\langle \bigwedge x. f x \geq 0 \rangle$  for  $f g$ 
  ::  $\langle \text{nat} \Rightarrow \text{real} \rangle$ 
  by (rule real-tendsto-sandwich[where h=g and f= $\lambda \cdot. 0$ ]) (use that in auto)
  have [simp]:  $a * = a$ 
    using assms(2) by (simp add: selfadjoint-def)
  have [simp]:  $\text{Re } (h x \cdot_C h x) = 1$  for  $x$ 
    using normh[of x] by (simp flip: power2-norm-eq-cinner')
  have  $\langle (\text{norm } ((a - e' *_R \text{id-cblinfun}) (h n)))^2 = (\text{norm } (a (h n)))^2 - 2 * e' * \text{Re } (h n \cdot_C a (h n)) + e'^2 \rangle$  for  $n$ 
    by (simp add: power2-norm-eq-cinner' algebra-simps cblinfun.cbilinear-simps
      cblinfun.scaleR-left power2-eq-square[of e'] flip: cinner-adj-right)
  also have  $\langle \dots n \leq e'^2 - 2 * e' * \text{Re } (h n \cdot_C a (h n)) + e'^2 \rangle$  for  $n$ 
  proof -
    from norme have  $\langle e'^2 = (\text{norm } a)^2 \rangle$ 
      by (auto simp: ee' power2-eq-iff abs-if split: if-splits)
    then have  $\langle (\text{norm } (a *_V h n))^2 \leq e'^2 \rangle$ 
      using norm-cblinfun[of a h n] by (simp add: normh)
    then show ?thesis
      by auto
  qed
  also have  $\langle \dots \longrightarrow 0 \rangle$ 
    apply (subst asm-rl[of  $\langle (\lambda n. e'^2 - 2 * e' * \text{Re } (h n \cdot_C a (h n)) + e'^2) = (\lambda n. 2 * e' * (e' - \text{Re } (h n \cdot_C (a *_V h n)))) \rangle$ ])
  subgoal
    by (auto simp: fun-eq-iff right-diff-distrib power2-eq-square)[1]
  subgoal
    using h-lim[THEN tendsto-Re]

```

```

    by (auto intro!: tendsto-mult-right-zero tendsto-diff-const-left-rewrite simp: ee')
  done
  finally have ⟨(λn. (a - e' *R id-cblinfun) (h n)) ⟶ 0⟩
    by (simp add: tendsto-norm-zero-iff)
  then show ⟨e' ∈ eigenvalues a⟩
    unfolding scaleR-scaleC
    by (rule eigenvalue-in-the-limit-compact-op[rotated -1])
      (use ⟨a ≠ 0⟩ norme in ⟨auto intro!: normh simp: assms ee'⟩)
qed
from ⟨e ∈ ℝ⟩ norme
have ⟨e = norm a ∨ e = - norm a⟩
  by (smt (verit, best) in-Reals-norm of-real-Re)
with ⟨e' ∈ eigenvalues a⟩ show ?thesis
  using ee' by presburger
qed

lemma
  fixes a :: ⟨'a ⇒CL 'a::{not-singleton, hilbert-space}⟩
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  shows largest-eigenvalue-norm-aux: ⟨largest-eigenvalue a ∈ {norm a, - norm a}⟩
    and largest-eigenvalue-ex: ⟨largest-eigenvalue a ∈ eigenvalues a⟩
proof -
  define l where ⟨l = (SOME x. x ∈ eigenvalues a ∧ (∀ y ∈ eigenvalues a. cmod x ≥ cmod y))⟩
  from norm-is-eigenvalue[OF assms]
  obtain e where ⟨e ∈ {of-real (norm a), - of-real (norm a)}⟩ and ⟨e ∈ eigenvalues a⟩
    by auto
  then have norme: ⟨norm e = norm a⟩
    by auto
  have ⟨e ∈ eigenvalues a ∧ (∀ y ∈ eigenvalues a. cmod y ≤ cmod e)⟩ (is ⟨?P e⟩)
    by (auto intro!: ⟨e ∈ eigenvalues a⟩ simp: eigenvalue-norm-bound norme)
  then have *: ⟨l ∈ eigenvalues a ∧ (∀ y ∈ eigenvalues a. cmod y ≤ cmod l)⟩
    unfolding l-def largest-eigenvalue-def by (rule someI)
  then have l-def': ⟨l = largest-eigenvalue a⟩
    by (metis (mono-tags, lifting) l-def largest-eigenvalue-def)
  from * have ⟨l ∈ eigenvalues a⟩
    by (simp add: l-def)
  then show ⟨largest-eigenvalue a ∈ eigenvalues a⟩
    by (simp add: l-def')
  have ⟨norm l ≥ norm a⟩
    using * norme ⟨e ∈ eigenvalues a⟩ by auto
  moreover have ⟨norm l ≤ norm a⟩
    using * eigenvalue-norm-bound by blast
  ultimately have ⟨norm l = norm a⟩
    by linarith
  moreover have ⟨l ∈ ℝ⟩
    using ⟨l ∈ eigenvalues a⟩ assms(2) eigenvalue-selfadj-real by blast
  ultimately have ⟨l ∈ {norm a, - norm a}⟩
    by (smt (verit, ccfv-SIG) in-Reals-norm insertCI l-def of-real-Re)

```

```

    then show ⟨largest-eigenvalue a ∈ {norm a, - norm a}⟩
      by (simp add: l-def')
qed

lemma largest-eigenvalue-norm:
  fixes a :: ⟨'a ⇒CL 'a::hilbert-space⟩
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨largest-eigenvalue a ∈ {norm a, - norm a}⟩
proof (cases ⟨class.not-singleton TYPE('a)⟩)
  case True
  show ?thesis
    using hilbert-space-class.hilbert-space-axioms True assms
    by (rule largest-eigenvalue-norm-aux[internalize-sort' 'a])
  next
  case False
  then have ⟨a = 0⟩
    by (rule not-not-singleton-cblinfun-zero)
  then show ?thesis
    by simp
qed

```

hide-fact largest-eigenvalue-norm-aux

```

lemma cmod-largest-eigenvalue:
  fixes a :: ⟨'a ⇒CL 'a::hilbert-space⟩
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨cmod (largest-eigenvalue a) = norm a⟩
  using largest-eigenvalue-norm[OF assms] by auto

lemma compact-op-comp-right: ⟨compact-op (a oCL b)⟩ if ⟨compact-op b⟩
  for a b :: ⟨-::hilbert-space ⇒CL -::hilbert-space⟩
proof -
  from that have ⟨b ∈ closure (Collect finite-rank)⟩
  using compact-op-finite-rank by blast
  then have ⟨a oCL b ∈ cblinfun-compose a ‘ closure (Collect finite-rank)⟩
    by blast
  also have ⟨... ⊆ closure (cblinfun-compose a ‘ Collect finite-rank)⟩
    by (auto intro!: closure-bounded-linear-image-subset bounded-clinear.bounded-linear bounded-clinear-cblinfun-comp)
  also have ⟨... ⊆ closure (Collect finite-rank)⟩
    by (auto intro!: closure-mono finite-rank-comp-right)
  finally show ⟨compact-op (a oCL b)⟩
    using compact-op-finite-rank by blast
qed

```

unbundle no cblinfun-syntax

end

11 Spectral-Theorem – The spectral theorem for compact operators

theory *Spectral-Theorem*

imports *Compact-Operators Positive-Operators Eigenvalues*

begin

unbundle *cblinfun-syntax*

11.1 Spectral decomp, compact op

fun *spectral-dec-val* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow \text{complex}$

— The eigenvalues in the spectral decomposition

and *spectral-dec-space* :: $\langle 'a \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow 'a \text{ ccspace}$

— The eigenspaces in the spectral decomposition

and *spectral-dec-op* :: $\langle 'a \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow_{CL} 'a)$

— A sequence of operators mostly for the proof of spectral composition. But see also *spectral-dec-op-spectral-dec-proj* below.

where $\langle \text{spectral-dec-val } a \ n = \text{largest-eigenvalue } (\text{spectral-dec-op } a \ n) \rangle$

$\langle \text{spectral-dec-space } a \ n = (\text{if } \text{spectral-dec-val } a \ n = 0 \text{ then } 0 \text{ else } \text{eigenspace } (\text{spectral-dec-val } a \ n) \ (\text{spectral-dec-op } a \ n)) \rangle$

$\langle \text{spectral-dec-op } a \ (\text{Suc } n) = \text{spectral-dec-op } a \ n \ o_{CL} \ \text{Proj } (- \ \text{spectral-dec-space } a \ n) \rangle$

$\langle \text{spectral-dec-op } a \ 0 = a \rangle$

definition *spectral-dec-proj* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow_{CL} 'a)$ **where**

— Projectors in the spectral decomposition

$\langle \text{spectral-dec-proj } a \ n = \text{Proj } (\text{spectral-dec-space } a \ n) \rangle$

declare *spectral-dec-val.simps*[*simp del*]

declare *spectral-dec-space.simps*[*simp del*]

lemmas *spectral-dec-def* = *spectral-dec-val.simps*

lemmas *spectral-dec-space-def* = *spectral-dec-space.simps*

lemma *spectral-dec-op-selfadj*:

assumes $\langle \text{selfadjoint } a \rangle$

shows $\langle \text{selfadjoint } (\text{spectral-dec-op } a \ n) \rangle$

proof (*induction n*)

case 0

with *assms* **show** ?*case*

by *simp*

next

case (*Suc n*)

define *E T* **where** $\langle E = \text{spectral-dec-space } a \ n \rangle$ **and** $\langle T = \text{spectral-dec-op } a \ n \rangle$

from *Suc* **have** $\langle \text{normal-op } T \rangle$

by (*auto intro!*: *selfadjoint-imp-normal simp: T-def*)

then have $\langle \text{reducing-subspace } E \ T \rangle$

by (*auto intro!*: *eigenspace-is-reducing simp: spectral-dec-space-def E-def T-def*)

then have $\langle \text{reducing-subspace } (- \ E) \ T \rangle$


```

  by simp
  then have *: ⟨Proj (- E) oCL T oCL Proj (- E) = T oCL Proj (- E)⟩
  by (simp add: invariant-subspace-iff-PAP reducing-subspace-def)
  show ?case
  using Suc
  apply (simp add: flip: T-def E-def *)
  by (simp add: selfadjoint-def adj-Proj cblinfun-compose-assoc)
qed

```

```

lemma spectral-dec-op-compact:
  assumes ⟨compact-op a⟩
  shows ⟨compact-op (spectral-dec-op a n)⟩
  apply (induction n)
  by (auto intro!: compact-op-comp-left assms)

```

```

lemma spectral-dec-val-eigenvalue-of-spectral-dec-op:
  fixes a :: ⟨'a::{chilbert-space, not-singleton} ⇒CL 'a⟩
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨spectral-dec-val a n ∈ eigenvalues (spectral-dec-op a n)⟩
  by (auto intro!: largest-eigenvalue-ex spectral-dec-op-compact spectral-dec-op-selfadj assms
      simp: spectral-dec-def)

```

```

lemma spectral-dec-proj-finite-rank:
  assumes ⟨compact-op a⟩
  shows ⟨finite-rank (spectral-dec-proj a n)⟩
  apply (cases ⟨spectral-dec-val a n = 0⟩)
  by (auto intro!: finite-rank-Proj-finite-dim compact-op-eigenspace-finite-dim spectral-dec-op-compact
      assms
      simp: spectral-dec-proj-def spectral-dec-space-def)

```

```

lemma norm-spectral-dec-op:
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨norm (spectral-dec-op a n) = cmod (spectral-dec-val a n)⟩
  by (simp add: spectral-dec-def cmod-largest-eigenvalue spectral-dec-op-compact spectral-dec-op-selfadj
      assms)

```

```

lemma spectral-dec-op-decreasing-eigenspaces:
  assumes ⟨n ≥ m⟩ and ⟨e ≠ 0⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨eigenspace e (spectral-dec-op a n) ≤ eigenspace e (spectral-dec-op a m)⟩
proof -
  have *: ⟨eigenspace e (spectral-dec-op a (Suc n)) ≤ eigenspace e (spectral-dec-op a n)⟩ for n
  proof (intro ccspace-leI subsetI)
    fix h
    assume asm: ⟨h ∈ space-as-set (eigenspace e (spectral-dec-op a (Suc n)))⟩
    have ⟨orthogonal-spaces (eigenspace e (spectral-dec-op a (Suc n))) (kernel (spectral-dec-op a

```

```

(Suc n))›
  using spectral-dec-op-selfadj[of a ⟨Suc n⟩] ⟨e ≠ 0⟩ ⟨selfadjoint a⟩
  by (auto intro!: eigenspaces-orthogonal selfadjoint-imp-normal spectral-dec-op-selfadj
      simp: spectral-dec-space-def simp flip: eigenspace-0)
  then have ⟨eigenspace e (spectral-dec-op a (Suc n)) ≤ - kernel (spectral-dec-op a (Suc n))⟩
    using orthogonal-spaces-leq-compl by blast
  also have ⟨... ≤ - spectral-dec-space a n⟩
    by (auto intro!: cccsubspace-leI kernel-memberI simp: Proj-0-compl)
  finally have ⟨h ∈ space-as-set (- spectral-dec-space a n)⟩
    using asm by (simp add: Set.basic-monos(γ) less-eq-cccsubspace.rep-eq)
  then have ⟨spectral-dec-op a n h = spectral-dec-op a (Suc n) h⟩
    by (simp add: Proj-fixes-image)
  also have ⟨... = e *C h⟩
    using asm eigenspace-memberD by blast
  finally show ⟨h ∈ space-as-set (eigenspace e (spectral-dec-op a n))⟩
    by (simp add: eigenspace-memberI)
qed
define k where ⟨k = n - m⟩
from * have ⟨eigenspace e (spectral-dec-op a (m + k)) ≤ eigenspace e (spectral-dec-op a m)⟩
  by (induction k) (auto simp del: spectral-dec-op.simps intro: order.trans)
then show ?thesis
  using ⟨n ≥ m⟩ by (simp add: k-def)
qed

```

```

lemma spectral-dec-val-not-not-singleton:
  fixes a :: ⟨'a::chilbert-space ⇒CL 'a⟩
  assumes ⟨¬ class.not-singleton TYPE('a)⟩
  shows ⟨spectral-dec-val a n = 0⟩
proof -
  from assms have ⟨spectral-dec-op a n = 0⟩
    by (rule not-not-singleton-cblinfun-zero)
  then have ⟨largest-eigenvalue (spectral-dec-op a n) = 0⟩
    by simp
  then show ?thesis
    by (simp add: spectral-dec-def)
qed

```

```

lemma spectral-dec-val-eigenvalue-aux:
  — [2], Theorem II.5.1
  fixes a :: ⟨'a::{chilbert-space, not-singleton} ⇒CL 'a⟩
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  assumes eigen-neq0: ⟨spectral-dec-val a n ≠ 0⟩
  shows ⟨spectral-dec-val a n ∈ eigenvalues a⟩
proof -
  define e where ⟨e = spectral-dec-val a n⟩
  with assms have ⟨e ≠ 0⟩
    by fastforce

```

```

from spectral-dec-op-decreasing-eigenspaces[where  $m=0$  and  $a=a$  and  $n=n$ ,  $OF - \langle e \neq 0 \rangle$ 
 $\langle \text{selfadjoint } a \rangle$ 
have 1:  $\langle \text{eigenspace } e \text{ (spectral-dec-op } a \ n) \leq \text{eigenspace } e \ a \rangle$ 
by simp
have 2:  $\langle \text{spectral-dec-space } a \ n \neq \perp \rangle$ 
proof –
have  $\langle \text{spectral-dec-val } a \ n \in \text{eigenvalues (spectral-dec-op } a \ n) \rangle$ 
by (simp add: assms(1) assms(2) spectral-dec-val.simps spectral-dec-op-compact spectral-dec-op-selfadj largest-eigenvalue-ex)
then show ?thesis
using  $\langle e \neq 0 \rangle$  by (simp add: eigenvalues-def spectral-dec-space.simps e-def)
qed
from 1 2 have  $\langle \text{eigenspace } e \ a \neq \perp \rangle$ 
by (auto simp: spectral-dec-space-def bot-unique simp flip: e-def simp: \langle e \neq 0 \rangle)
then show  $\langle e \in \text{eigenvalues } a \rangle$ 
by (simp add: eigenvalues-def)
qed

```

```

lemma spectral-dec-val-eigenvalue:
– [2], Theorem II.5.1
fixes  $a :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'a \rangle$ 
assumes  $\langle \text{compact-op } a \rangle$ 
assumes  $\langle \text{selfadjoint } a \rangle$ 
assumes  $\text{eigen-neq0: } \langle \text{spectral-dec-val } a \ n \neq 0 \rangle$ 
shows  $\langle \text{spectral-dec-val } a \ n \in \text{eigenvalues } a \rangle$ 
proof (cases \langle class.not-singleton TYPE('a) \rangle)
case True
show ?thesis
using hilbert-space-axioms True assms
by (rule spectral-dec-val-eigenvalue-aux[internalize-sort' 'a])
next
case False
then have  $\langle \text{spectral-dec-val } a \ n = 0 \rangle$ 
by (rule spectral-dec-val-not-not-singleton)
with assms show ?thesis
by simp
qed

```

hide-fact *spectral-dec-val-eigenvalue-aux*

```

lemma spectral-dec-val-decreasing:
assumes  $\langle \text{compact-op } a \rangle$ 
assumes  $\langle \text{selfadjoint } a \rangle$ 
assumes  $\langle n \geq m \rangle$ 
shows  $\langle \text{cmod (spectral-dec-val } a \ n) \leq \text{cmod (spectral-dec-val } a \ m) \rangle$ 
proof –
have  $\langle \text{norm (spectral-dec-op } a \ (Suc \ n)) \leq \text{norm (spectral-dec-op } a \ n) \rangle$  for  $n$ 
apply simp
by (smt (verit) Proj-partial-isometry cblinfun-compose-zero-right mult-cancel-left2 norm-cblinfun-compose)

```

norm-le-zero-iff norm-partial-isometry
then have $*$: $\langle \text{cmod} (\text{spectral-dec-val } a \text{ (Suc } n)) \leq \text{cmod} (\text{spectral-dec-val } a \text{ } n) \rangle$ **for** n
by (*simp add: cmod-largest-eigenvalue spectral-dec-op-compact assms spectral-dec-op-selfadj*
spectral-dec-def
del: spectral-dec-op.simps)
define k **where** $\langle k = n - m \rangle$
have $\langle \text{cmod} (\text{spectral-dec-val } a \text{ (} m + k)) \leq \text{cmod} (\text{spectral-dec-val } a \text{ } m) \rangle$
apply (*induction k arbitrary: m*)
apply *simp*
by (*metis * add-Suc-right order-trans-rules(23)*)
with $\langle n \geq m \rangle$ **show** *?thesis*
by (*simp add: k-def*)
qed

lemma *spectral-dec-val-distinct-aux*:
fixes $a :: \langle 'a :: \{\text{chilbert-space, not-singleton}\} \Rightarrow_{CL} 'a \rangle$
assumes $\langle n \neq m \rangle$
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
assumes *neq0*: $\langle \text{spectral-dec-val } a \text{ } n \neq 0 \rangle$
shows $\langle \text{spectral-dec-val } a \text{ } n \neq \text{spectral-dec-val } a \text{ } m \rangle$
proof (*rule ccontr*)
assume $\langle \neg \text{spectral-dec-val } a \text{ } n \neq \text{spectral-dec-val } a \text{ } m \rangle$
then have *eq*: $\langle \text{spectral-dec-val } a \text{ } n = \text{spectral-dec-val } a \text{ } m \rangle$
by *blast*
wlog *nm*: $\langle n > m \rangle$ **goal** *False* **generalizing** $n \ m$ **keeping** *eq neq0*
using *hypothesis[of n m] negation assms eq neq0*
by *auto*
define e **where** $\langle e = \text{spectral-dec-val } a \text{ } n \rangle$
with *neq0* **have** $\langle e \neq 0 \rangle$
by *simp*

have $\langle \text{spectral-dec-space } a \text{ } n \neq \perp \rangle$
proof –
have $\langle e \in \text{eigenvalues} (\text{spectral-dec-op } a \text{ } n) \rangle$
by (*auto intro!: spectral-dec-val-eigenvalue-of-spectral-dec-op assms simp: e-def*)
then show *?thesis*
by (*simp add: spectral-dec-space-def eigenvalues-def e-def neq0*)
qed
then obtain h **where** $\langle \text{norm } h = 1 \rangle$ **and** $h \in \text{space-as-set} (\text{spectral-dec-space } a \text{ } n) \rangle$
using *ccsubspace-contains-unit* **by** *blast*
have *T-Sucm-h*: $\langle \text{spectral-dec-op } a \text{ (Suc } m) \ h = 0 \rangle$
proof –
have $\langle \text{spectral-dec-space } a \text{ } n = \text{eigenspace } e \text{ (spectral-dec-op } a \text{ } n) \rangle$
by (*simp add: spectral-dec-space-def e-def neq0*)
also have $\langle \dots \leq \text{eigenspace } e \text{ (spectral-dec-op } a \text{ } m) \rangle$
using $\langle n > m \rangle \langle e \neq 0 \rangle$ *assms*
by (*auto intro!: spectral-dec-op-decreasing-eigenspaces simp:*)

```

also have ⟨... = spectral-dec-space a m⟩
  using neq0 by (simp add: spectral-dec-space-def e-def eq)
finally have ⟨h ∈ space-as-set (spectral-dec-space a m)⟩
  using h-En
  by (simp add: basic-trans-rules(31) less-eq-ccsubspace.rep-eq)
then show ⟨spectral-dec-op a (Suc m) h = 0⟩
  by (simp add: Proj-0-compl)
qed
have ⟨spectral-dec-op a (Suc m + k) h = 0⟩ if ⟨k ≤ n - m - 1⟩ for k
proof (insert that, induction k)
  case 0
  from T-Sucm-h show ?case
  by simp
next
  case (Suc k)
  define mk1 where ⟨mk1 = Suc (m + k)⟩
  from Suc.prem1 have ⟨mk1 ≤ n⟩
  using mk1-def by linarith
  have ⟨eigenspace e (spectral-dec-op a n) ≤ eigenspace e (spectral-dec-op a mk1)⟩
  using ⟨mk1 ≤ n⟩ ⟨e ≠ 0⟩ ⟨selfadjoint a⟩
  by (rule spectral-dec-op-decreasing-eigenspaces)
  with h-En have h-mk1: ⟨h ∈ space-as-set (eigenspace e (spectral-dec-op a mk1))⟩
  by (auto simp: e-def spectral-dec-space-def less-eq-ccsubspace.rep-eq neq0)
  have ⟨Proj (- spectral-dec-space a mk1) *V h = 0 ∨ Proj (- spectral-dec-space a mk1) *V
h = h⟩
  proof (cases ⟨e = spectral-dec-val a mk1⟩)
  case True
  from h-mk1 have ⟨Proj (- spectral-dec-space a mk1) h = 0⟩
  using ⟨e ≠ 0⟩ by (simp add: Proj-0-compl True spectral-dec-space-def)
  then show ?thesis
  by simp
  next
  case False
  have ⟨orthogonal-spaces (eigenspace e (spectral-dec-op a mk1)) (spectral-dec-space a mk1)⟩
  by (simp add: False assms eigenspaces-orthogonal spectral-dec-space.simps spectral-dec-op-selfadj
selfadjoint-imp-normal)
  with h-mk1 have ⟨h ∈ space-as-set (- spectral-dec-space a mk1)⟩
  using less-eq-ccsubspace.rep-eq orthogonal-spaces-leq-compl by blast
  then have ⟨Proj (- spectral-dec-space a mk1) h = h⟩
  by (rule Proj-fixes-image)
  then show ?thesis
  by simp
  qed
with Suc show ?case
  by (auto simp: mk1-def)
qed
from this[where k=⟨n - m - 1⟩]
have ⟨spectral-dec-op a n h = 0⟩
  using ⟨n > m⟩

```

```

    by (simp del: spectral-dec-op.simps)
  moreover from h-En have ⟨spectral-dec-op a n h = e *C h⟩
    by (simp add: neq0 e-def eigenspace-memberD spectral-dec-space-def)
  ultimately show False
    using ⟨norm h = 1⟩ ⟨e ≠ 0⟩
    by force
qed

```

```

lemma spectral-dec-val-distinct:
  fixes a :: ⟨'a::hilbert-space ⇒CL 'a⟩
  assumes ⟨n ≠ m⟩
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  assumes neq0: ⟨spectral-dec-val a n ≠ 0⟩
  shows ⟨spectral-dec-val a n ≠ spectral-dec-val a m⟩
proof (cases ⟨class.not-singleton TYPE('a)⟩)
  case True
  show ?thesis
    using hilbert-space-axioms True assms
    by (rule spectral-dec-val-distinct-aux[internalize-sort' 'a])
  next
  case False
  then have ⟨spectral-dec-val a n = 0⟩
    by (rule spectral-dec-val-not-not-singleton)
  with assms show ?thesis
    by simp
qed

```

hide-fact *spectral-dec-val-distinct-aux*

```

lemma spectral-dec-val-tendsto-0:
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨spectral-dec-val a ⟶ 0⟩
proof (cases ⟨∃ n. spectral-dec-val a n = 0⟩)
  case True
  then obtain n where ⟨spectral-dec-val a n = 0⟩
    by auto
  then have ⟨spectral-dec-val a m = 0⟩ if ⟨m ≥ n⟩ for m
    using spectral-dec-val-decreasing[OF assms that]
    by simp
  then show ⟨spectral-dec-val a ⟶ 0⟩
    by (auto intro!: tendsto-eventually eventually-sequentiallyI)
  next
  case False
  define E where ⟨E = spectral-dec-val a⟩
  from False have ⟨E n ∈ eigenvalues a⟩ for n
    by (simp add: spectral-dec-val-eigenvalue assms E-def)

```

```

then have ‹eigenspace (E n) a ≠ 0› for n
  by (simp add: eigenvalues-def)
then obtain e where e-E: ‹e n ∈ space-as-set (eigenspace (E n) a)›
  and norm-e: ‹norm (e n) = 1› for n
  apply atomize-elim
  using cccsubspace-contains-unit
  by (auto intro!: choice2)
then obtain h n where ‹strict-mono n› and aen-lim: ‹(λj. a (e (n j))) → h›
proof atomize-elim
  from ‹compact-op a›
  have compact: ‹compact (closure (a ‘ cball 0 1))›
    by (simp add: compact-op-def2)
  from norm-e have ‹a (e n) ∈ closure (a ‘ cball 0 1)› for n
    using closure-subset[of ‹a ‘ cball 0 1›] by auto
  with compact[unfolded compact-def, rule-format, of ‹λn. a (e n)›]
  show ‹∃ n h. strict-mono n ∧ (λj. a (e (n j))) → h›
    by (auto simp: o-def)
qed
have ortho-en: ‹is-orthogonal (e (n j)) (e (n k))› if ‹j ≠ k› for j k
proof -
  have ‹n j ≠ n k›
    by (simp add: ‹strict-mono n› strict-mono-eq that)
  then have ‹E (n j) ≠ E (n k)›
    unfolding E-def
    apply (rule spectral-dec-val-distinct)
    using False assms by auto
  then have ‹orthogonal-spaces (eigenspace (E (n j)) a) (eigenspace (E (n k)) a)›
    apply (rule eigenspaces-orthogonal)
    by (simp add: assms(2) selfadjoint-imp-normal)
  with e-E show ?thesis
    using orthogonal-spaces-def by blast
qed
have aEe: ‹a (e n) = E n *C e n› for n
  by (simp add: e-E eigenspace-memberD)
obtain α where E-lim: ‹(λn. norm (E n)) → α›
  by (rule decseq-convergent[where X=‹λn. cmod (E n)› and B=0])
  (use spectral-dec-val-decreasing[OF assms] in ‹auto intro!: simp: decseq-def E-def›)
then have ‹α ≥ 0›
  apply (rule LIMSEQ-le-const)
  by simp
have aen-diff: ‹norm (a (e (n j)) - a (e (n k))) ≥ α * sqrt 2› if ‹j ≠ k› for j k
proof -
  from E-lim and spectral-dec-val-decreasing[OF assms, folded E-def]
  have E-geq-α: ‹cmod (E n) ≥ α› for n
    apply (rule-tac decseq-ge[unfolded decseq-def, rotated])
    by auto
  have ‹(norm (a (e (n j)) - a (e (n k))))2 = (cmod (E (n j)))2 + (cmod (E (n k)))2›
    by (simp add: polar-identity-minus aEe that ortho-en norm-e)
  also have ‹... ≥ α2 + α2› (is ‹- ≥ ...›)

```

```

    apply (rule add-mono)
    using E-geq- $\alpha \geq 0$  by auto
  also have  $\langle \dots = (\alpha * \text{sqrt } 2)^2 \rangle$ 
    by (simp add: algebra-simps)
  finally show ?thesis
    apply (rule power2-le-imp-le)
    by simp
qed
have  $\langle \alpha = 0 \rangle$ 
proof -
  have  $\langle \alpha * \text{sqrt } 2 < \varepsilon \rangle$  if  $\langle \varepsilon > 0 \rangle$  for  $\varepsilon$ 
  proof -
    from  $\langle \text{strict-mono } n \rangle$  have cauchy:  $\langle \text{Cauchy } (\lambda k. a (e (n k))) \rangle$ 
      using LIMSEQ-imp-Cauchy aen-lim by blast
    obtain k where k:  $\langle \forall m \geq k. \forall n a \geq k. \text{dist } (a *_{\vee} e (n m)) (a *_{\vee} e (n na)) < \varepsilon \rangle$ 
      apply atomize-elim
      using cauchy[unfolded Cauchy-def, rule-format, OF  $\langle \varepsilon > 0 \rangle$ ]
      by simp
    define j where  $\langle j = \text{Suc } k \rangle$ 
    then have  $\langle j \neq k \rangle$ 
      by simp
    from k have  $\langle \text{dist } (a (e (n j))) (a (e (n k))) < \varepsilon \rangle$ 
      by (simp add: j-def)
    with aen-diff[OF  $\langle j \neq k \rangle$ ] show  $\langle \alpha * \text{sqrt } 2 < \varepsilon \rangle$ 
      by (simp add: Cauchy-def dist-norm)
  qed
  with  $\langle \alpha \geq 0 \rangle$  show  $\langle \alpha = 0 \rangle$ 
    by (smt (verit) linordered-semiring-strict-class.mult-pos-pos real-sqrt-le-0-iff)
qed
with E-lim show ?thesis
  by (auto intro!: tendsto-norm-zero-cancel simp: E-def)
qed

```

```

lemma spectral-dec-op-tendsto:
  assumes  $\langle \text{compact-op } a \rangle$ 
  assumes  $\langle \text{selfadjoint } a \rangle$ 
  shows  $\langle \text{spectral-dec-op } a \longrightarrow 0 \rangle$ 
  apply (rule tendsto-norm-zero-cancel)
  using spectral-dec-val-tendsto-0[OF assms]
  apply (simp add: norm-spectral-dec-op assms)
  using tendsto-norm-zero by blast

```

```

lemma spectral-dec-op-spectral-dec-proj:
   $\langle \text{spectral-dec-op } a \ n = a - (\sum i < n. \text{spectral-dec-val } a \ i *_{\mathcal{C}} \text{spectral-dec-proj } a \ i) \rangle$ 
proof (induction n)
  case 0
  show ?case
    by simp
next

```



```

case (Suc n)
have ⟨spectral-dec-op a (Suc n) = spectral-dec-op a n oCL Proj (– spectral-dec-space a n)⟩
  by simp
also have ⟨... = spectral-dec-op a n – spectral-dec-val a n *C spectral-dec-proj a n⟩ (is ⟨?lhs
= ?rhs⟩)
proof –
  have ⟨?lhs h = ?rhs h⟩ if ⟨h ∈ space-as-set (spectral-dec-space a n)⟩ for h
  proof –
    have ⟨?lhs h = 0⟩
      by (simp add: Proj-0-compl that)
    have ⟨spectral-dec-op a n *V h = spectral-dec-val a n *C h⟩
      by (smt (verit, best) Proj-fixes-image ⟨(spectral-dec-op a n oCL Proj (– spectral-dec-space a
n)) *V h = 0⟩ cblinfun-apply-cblinfun-compose complex-vector.scale-eq-0-iff eigenspace-memberD
spectral-dec-space.elims kernel-Proj kernel-cblinfun-compose kernel-memberD kernel-memberI or-
tho-involution that)
    also have ⟨... = spectral-dec-val a n *C (spectral-dec-proj a n *V h)⟩
      by (simp add: Proj-fixes-image spectral-dec-proj-def that)
    finally
      have ⟨?rhs h = 0⟩
        by (simp add: cblinfun.diff-left)
      with ⟨?lhs h = 0⟩ show ?thesis
        by simp
  qed
moreover have ⟨?lhs h = ?rhs h⟩ if ⟨h ∈ space-as-set (– spectral-dec-space a n)⟩ for h
  by (simp add: Proj-0-compl Proj-fixes-image cblinfun.diff-left spectral-dec-proj-def that)
ultimately have ⟨?lhs h = ?rhs h⟩
  if ⟨h ∈ space-as-set (spectral-dec-space a n ⊔ – spectral-dec-space a n)⟩ for h
  using that by (rule eq-on-ccsubspaces-sup)
then show ⟨?lhs = ?rhs⟩
  by (auto intro!: cblinfun-eqI simp add: )
qed
also have ⟨... = a – (∑ i<Suc n. spectral-dec-val a i *C spectral-dec-proj a i)⟩
  by (simp add: Suc.IH)
finally show ?case
  by –
qed

```

lemma sequential-tendsto-reorder:

```

assumes ⟨inj g⟩
assumes ⟨f ⟶ l⟩
shows ⟨(f o g) ⟶ l⟩
proof (intro lim-explicit[THEN iffD2] impI allI)
fix S assume ⟨open S⟩ and ⟨l ∈ S⟩
with ⟨f ⟶ l⟩
obtain M where M: ⟨f m ∈ S⟩ if ⟨m ≥ M⟩ for m
  using tendsto-obtains-N by blast
define N where ⟨N = Max (g – ‘{..M}) + 1⟩
have N: ⟨g n ≥ M⟩ if ⟨n ≥ N⟩ for n

```

proof –
from $\langle \text{inj } g \rangle$ **have** $\langle \text{finite } (g - \{..<M\}) \rangle$
using *finite-vimageI* **by** *blast*
then have $\langle N > n \rangle$ **if** $\langle n \in g - \{..<M\} \rangle$ **for** n
using *N-def that*
by (*simp add: less-Suc-eq-le*)
then have $\langle N > n \rangle$ **if** $\langle g n < M \rangle$ **for** n
by (*simp add: that*)
with that show $\langle g n \geq M \rangle$
using *linorder-not-less* **by** *blast*
qed
from $N M$ **show** $\langle \exists N. \forall n \geq N. (f \circ g) n \in S \rangle$
by *auto*
qed

lemma *spectral-dec-sums*:
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle (\lambda n. \text{spectral-dec-val } a n *_C \text{spectral-dec-proj } a n) \text{ sums } a \rangle$
proof –
from *spectral-dec-op-tendsto[OF assms]*
have $\langle (\lambda n. a - \text{spectral-dec-op } a n) \longrightarrow a \rangle$
by (*simp add: tendsto-diff-const-left-rewrite*)
moreover from *spectral-dec-op-spectral-dec-proj[of a]*
have $\langle a - \text{spectral-dec-op } a n = (\sum_{i < n. \text{spectral-dec-val } a i *_C \text{spectral-dec-proj } a i) \rangle$ **for** n
by *simp*
ultimately show *?thesis*
by (*simp add: sums-def*)
qed

lemma *spectral-dec-val-real*:
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{spectral-dec-val } a n \in \mathbf{R} \rangle$
by (*metis Reals-0 assms(1) assms(2) eigenvalue-selfadj-real spectral-dec-val-eigenvalue*)

lemma *spectral-dec-space-orthogonal*:
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
assumes $\langle n \neq m \rangle$
shows $\langle \text{orthogonal-spaces } (\text{spectral-dec-space } a n) (\text{spectral-dec-space } a m) \rangle$
proof (*cases* $\langle \text{spectral-dec-val } a n = 0 \vee \text{spectral-dec-val } a m = 0 \rangle$)
case *True*
then show *?thesis*

by (auto intro!: simp: spectral-dec-space-def)
 next
 case False
 have ⟨spectral-dec-space a n ≤ eigenspace (spectral-dec-val a n) a⟩
 using ⟨selfadjoint a⟩
 by (metis False spectral-dec-space.elims spectral-dec-op.simps(2) spectral-dec-op-decreasing-eigenspaces zero-le)
 moreover have ⟨spectral-dec-space a m ≤ eigenspace (spectral-dec-val a m) a⟩
 using ⟨selfadjoint a⟩
 by (metis False spectral-dec-space.elims spectral-dec-op.simps(2) spectral-dec-op-decreasing-eigenspaces zero-le)
 moreover have ⟨orthogonal-spaces (eigenspace (spectral-dec-val a n) a) (eigenspace (spectral-dec-val a m) a)⟩
 apply (intro eigenspaces-orthogonal selfadjoint-imp-normal assms spectral-dec-val-distinct)
 using False by simp
 ultimately show ?thesis
 by (meson order.trans orthocomplemented-lattice-class.compl-mono orthogonal-spaces-leq-compl)

qed

lemma spectral-dec-proj-pos: ⟨spectral-dec-proj a n ≥ 0⟩
 by (auto intro!: simp: spectral-dec-proj-def)

lemma

assumes ⟨compact-op a⟩
 assumes ⟨selfadjoint a⟩
 shows spectral-dec-tendsto-pos-op: ⟨(λn. max 0 (spectral-dec-val a n) *_C spectral-dec-proj a n) sums pos-op a⟩ (is ?thesis1)
 and spectral-dec-tendsto-neg-op: ⟨(λn. - min (spectral-dec-val a n) 0 *_C spectral-dec-proj a n) sums neg-op a⟩ (is ?thesis2)

proof –

define I J where ⟨I = {n. spectral-dec-val a n ≥ 0}⟩
 and ⟨J = {n. spectral-dec-val a n ≤ 0}⟩
 define R S where ⟨R = (⋃_{n∈I}. spectral-dec-space a n)⟩
 and ⟨S = (⋃_{n∈J}. spectral-dec-space a n)⟩
 define aR aS where ⟨aR = a o_{CL} Proj R⟩ and ⟨aS = - a o_{CL} Proj S⟩
 have spectral-dec-cases: ⟨(0 < spectral-dec-val a n ⇒ P) ⇒
 (spectral-dec-val a n < 0 ⇒ P) ⇒
 (spectral-dec-val a n = 0 ⇒ P) ⇒ P⟩ for n P
 apply atomize-elim
 using reals-zero-comparable[OF spectral-dec-val-real[OF assms, of n]]
 by auto
 have PRP: ⟨spectral-dec-proj a n o_{CL} Proj R = spectral-dec-proj a n⟩ if ⟨n ∈ I⟩ for n
 by (auto intro!: Proj-o-Subspace-left
 simp add: R-def SUP-upper that spectral-dec-proj-def)
 have PR0: ⟨spectral-dec-proj a n o_{CL} Proj R = 0⟩ if ⟨n ∉ I⟩ for n
 apply (cases rule: spectral-dec-cases[of n])
 using that

by (*auto intro!*: *orthogonal-spaces-SUP-right spectral-dec-space-orthogonal assms*
simp: *spectral-dec-proj-def R-def I-def*
simp flip: *orthogonal-projectors-orthogonal-spaces*)
have *PSP*: $\langle \text{spectral-dec-proj } a \ n \ o_{CL} \ \text{Proj } S = \text{spectral-dec-proj } a \ n \rangle$ **if** $\langle n \in J \rangle$ **for** n
by (*auto intro!*: *Proj-o-Subspace-left*
simp add: *S-def SUP-upper that spectral-dec-proj-def*)
have *PS0*: $\langle \text{spectral-dec-proj } a \ n \ o_{CL} \ \text{Proj } S = 0 \rangle$ **if** $\langle n \notin J \rangle$ **for** n
apply (*cases rule*: *spectral-dec-cases[of n]*)
using *that*
by (*auto intro!*: *orthogonal-spaces-SUP-right spectral-dec-space-orthogonal assms*
simp: *spectral-dec-proj-def S-def J-def*
simp flip: *orthogonal-projectors-orthogonal-spaces*)
from *spectral-dec-sums[OF assms]*
have $\langle (\lambda n. (\text{spectral-dec-val } a \ n \ *_C \ \text{spectral-dec-proj } a \ n) \ o_{CL} \ \text{Proj } R) \ \text{sums } aR \rangle$
unfolding *aR-def*
apply (*rule bounded-linear.sums[rotated]*)
by (*intro bounded-clinear.bounded-linear bounded-clinear-cblinfun-compose-left*)
then have *sum-aR*: $\langle (\lambda n. \max 0 \ (\text{spectral-dec-val } a \ n) \ *_C \ \text{spectral-dec-proj } a \ n) \ \text{sums } aR \rangle$
apply (*rule sums-cong[THEN iffD1, rotated]*)
by (*simp add*: *I-def PR0 PRP max-def*)
from *sum-aR* **have** $\langle aR \geq 0 \rangle$
apply (*rule sums-pos-cblinfun*)
by (*auto intro!*: *spectral-dec-proj-pos scaleC-nonneg-nonneg simp: max-def*)
from *spectral-dec-sums[OF assms]*
have $\langle (\lambda n. \text{spectral-dec-val } a \ n \ *_C \ \text{spectral-dec-proj } a \ n \ o_{CL} \ \text{Proj } S) \ \text{sums } - \ aS \rangle$
unfolding *aS-def minus-minus cblinfun-compose-uminus-left*
apply (*rule bounded-linear.sums[rotated]*)
by (*intro bounded-clinear.bounded-linear bounded-clinear-cblinfun-compose-left*)
then have *sum-aS'*: $\langle (\lambda n. \min (\text{spectral-dec-val } a \ n) \ 0 \ *_C \ \text{spectral-dec-proj } a \ n) \ \text{sums } - \ aS \rangle$
apply (*rule sums-cong[THEN iffD1, rotated]*)
by (*simp add*: *J-def PS0 PSP min-def*)
then have *sum-aS*: $\langle (\lambda n. - \min (\text{spectral-dec-val } a \ n) \ 0 \ *_C \ \text{spectral-dec-proj } a \ n) \ \text{sums } aS \rangle$
using *sums-minus* **by** *fastforce*
from *sum-aS* **have** $\langle aS \geq 0 \rangle$
by (*rule sums-pos-cblinfun*)
(*auto intro!*: *spectral-dec-proj-pos scaleC-nonpos-nonneg simp: max-def min-def*)
from *sum-aR* *sum-aS'*
have $\langle (\lambda n. \max 0 \ (\text{spectral-dec-val } a \ n) \ *_C \ \text{spectral-dec-proj } a \ n$
 $\quad + \min (\text{spectral-dec-val } a \ n) \ 0 \ *_C \ \text{spectral-dec-proj } a \ n) \ \text{sums } (aR - aS) \rangle$
using *sums-add* **by** *fastforce*
then have $\langle (\lambda n. \text{spectral-dec-val } a \ n \ *_C \ \text{spectral-dec-proj } a \ n) \ \text{sums } (aR - aS) \rangle$
proof (*rule sums-cong[THEN iffD1, rotated]*)
fix n
have $\langle \max 0 \ (\text{spectral-dec-val } a \ n) + \min (\text{spectral-dec-val } a \ n) \ 0$
 $\quad = \text{spectral-dec-val } a \ n \rangle$
apply (*cases rule*: *spectral-dec-cases[of n]*)
by (*auto intro!*: *simp: max-def min-def*)
then
show $\langle \max 0 \ (\text{spectral-dec-val } a \ n) \ *_C \ \text{spectral-dec-proj } a \ n +$

```

      min (spectral-dec-val a n) 0 *C spectral-dec-proj a n =
      spectral-dec-val a n *C spectral-dec-proj a n
    by (metis scaleC-left.add)
  qed
  with spectral-dec-sums[OF assms]
  have ⟨aR - aS = a⟩
    using sums-unique2 by blast
  have ⟨aR oCL aS = 0⟩
    by (metis (no-types, opaque-lifting) Proj-idempotent ⟨0 ≤ aR⟩ ⟨aR - aS = a⟩ aR-def
    add-cancel-left-left add-minus-cancel adj-0 adj-Proj adj-cblinfun-compose assms(2) cblinfun-compose-minus-right
    comparable-selfadjoint lift-cblinfun-comp(2) selfadjoint-def uminus-add-conv-diff)
  have ⟨aR = pos-op a⟩ and ⟨aS = neg-op a⟩
    by (intro pos-op-neg-op-unique[where b=aR and c=aS]
    ⟨aR - aS = a⟩ ⟨0 ≤ aR⟩ ⟨0 ≤ aS⟩ ⟨aR oCL aS = 0⟩)+
  with sum-aR and sum-aS
  show ?thesis1 and ?thesis2
    by auto
  qed

```

```

lemma spectral-dec-tendsto-abs-op:
  assumes ⟨compact-op a⟩
  assumes ⟨selfadjoint a⟩
  shows ⟨(λn. cmod (spectral-dec-val a n) *R spectral-dec-proj a n) sums abs-op a⟩
proof -
  from spectral-dec-tendsto-pos-op[OF assms] spectral-dec-tendsto-neg-op[OF assms]
  have ⟨(λn. max 0 (spectral-dec-val a n) *C spectral-dec-proj a n
    + - min (spectral-dec-val a n) 0 *C spectral-dec-proj a n) sums (pos-op a + neg-op a)⟩
    using sums-add by blast
  then have ⟨(λn. cmod (spectral-dec-val a n) *R spectral-dec-proj a n) sums (pos-op a +
    neg-op a)⟩
    apply (rule sums-cong[THEN iffD1, rotated])
    using spectral-dec-val-real[OF assms]
  apply (simp add: complex-is-Real-iff cmod-def max-def min-def less-eq-complex-def scaleR-scaleC
    flip: scaleC-add-right)
    by (metis complex-surj zero-complex.code)
  then show ?thesis
    by (simp add: pos-op-plus-neg-op)
  qed

```

definition *spectral-dec-vecs* :: ⟨('a ⇒_{CL} 'a) ⇒ 'a::chilbert-space set⟩ **where**
 ⟨*spectral-dec-vecs* a = (⋃ n. scaleC (csqrt (spectral-dec-val a n)) ‘some-onb-of (spectral-dec-space a n))⟩

```

lemma spectral-dec-vecs-ortho:
  assumes ⟨selfadjoint a⟩ and ⟨compact-op a⟩
  shows ⟨is-ortho-set (spectral-dec-vecs a)⟩
proof (unfold is-ortho-set-def, intro conjI ballI impI)
  show ⟨0 ∉ spectral-dec-vecs a⟩
  proof (rule notI)

```

```

    assume ⟨0 ∈ spectral-dec-vecs a⟩
    then obtain n v where v0: ⟨0 = csqrt (spectral-dec-val a n) *C v⟩ and v-in: ⟨v ∈ some-onb-of
(spectral-dec-space a n)⟩
      by (auto simp: spectral-dec-vecs-def)
      from v-in have ⟨v ≠ 0⟩
        using some-onb-of-norm1 by fastforce
      from v-in have ⟨spectral-dec-space a n ≠ 0⟩
        using some-onb-of-0 by force
      then have ⟨spectral-dec-val a n ≠ 0⟩
        by (meson spectral-dec-space.elims)
      with v0 ⟨v ≠ 0⟩ show False
        by force
  qed
  fix g h assume g: ⟨g ∈ spectral-dec-vecs a⟩ and h: ⟨h ∈ spectral-dec-vecs a⟩ and ⟨g ≠ h⟩
  from g obtain ng g' where gg': ⟨g = csqrt (spectral-dec-val a ng) *C g'⟩ and g'-in: ⟨g' ∈
some-onb-of (spectral-dec-space a ng)⟩
    by (auto simp: spectral-dec-vecs-def)
  from h obtain nh h' where hh': ⟨h = csqrt (spectral-dec-val a nh) *C h'⟩ and h'-in: ⟨h' ∈
some-onb-of (spectral-dec-space a nh)⟩
    by (auto simp: spectral-dec-vecs-def)
  have ⟨is-orthogonal g' h'⟩
  proof (cases ⟨ng = nh⟩)
    case True
      with h'-in have ⟨h' ∈ some-onb-of (spectral-dec-space a nh)⟩
        by simp
      with g'-in True ⟨g ≠ h⟩ gg' hh'
      show ?thesis
        using is-ortho-set-def by fastforce
    case False
      then have ⟨orthogonal-spaces (spectral-dec-space a ng) (spectral-dec-space a nh)⟩
        by (auto intro!: spectral-dec-space-orthogonal-assms simp: )
      with h'-in g'-in show ⟨is-orthogonal g' h'⟩
        using orthogonal-spaces-ccspan by force
  qed
  then show ⟨is-orthogonal g h⟩
    by (simp add: gg' hh')
  qed

lemma spectral-dec-val-nonneg:
  assumes ⟨a ≥ 0⟩
  assumes ⟨compact-op a⟩
  shows ⟨spectral-dec-val a n ≥ 0⟩
proof -
  define v where ⟨v = spectral-dec-val a n⟩
  wlog non0: ⟨spectral-dec-val a n ≠ 0⟩ generalizing v keeping v-def
  using negation by force
  have [simp]: ⟨selfadjoint a⟩
  using adj-0-assms(1) comparable-selfadjoint-selfadjoint-def by blast

```

```

have ⟨v ∈ eigenvalues a⟩
  by (auto intro!: non0 spectral-dec-val-eigenvalue assms simp: v-def)
then show ⟨spectral-dec-val a n ≥ 0⟩
  using assms(1) eigenvalues-nonneg v-def by blast
qed

```

```

lemma spectral-dec-space-finite-dim[intro]:
  assumes ⟨compact-op a⟩
  shows ⟨finite-dim-ccsubspace (spectral-dec-space a n)⟩
  by (auto intro!: compact-op-eigenspace-finite-dim spectral-dec-op-compact assms simp: spectral-dec-space-def)

```

```

lemma spectral-dec-space-0:
  assumes ⟨spectral-dec-val a n = 0⟩
  shows ⟨spectral-dec-space a n = 0⟩
  by (simp add: assms spectral-dec-space-def)

```

```

unbundle no cblinfun-syntax

```

```

end

```

12 Trace-Class – Trace-class operators

```

theory Trace-Class
  imports Complex-Bounded-Operators.Complex-L2 HS2Ell2
    Weak-Operator-Topology Positive-Operators Compact-Operators
    Spectral-Theorem
begin

```

```

hide-fact (open) Infinite-Set-Sum.abs-summable-on-Sigma-iff
hide-fact (open) Infinite-Set-Sum.abs-summable-on-comparison-test
hide-const (open) Determinants.trace
hide-fact (open) Determinants.trace-def

```

```

unbundle cblinfun-syntax

```

12.1 Auxiliary lemmas

```

lemma
  fixes h :: ⟨'a::{chilbert-space}⟩
  assumes ⟨is-onb E⟩
  shows parseval-abs-summable: (λe. (cmod (e •C h))2) abs-summable-on E
proof (cases h = 0)
  case True
  then show ?thesis by simp
next
  case False
  then have ⟨ $\sum_{\infty e \in E. (cmod (e \cdot_C h))^2} \neq 0$ ⟩

```

```

    using assms by (simp add: parseval-identity is-onb-def)
  then show ?thesis
    using infsun-not-exists by auto
qed

```

lemma *basis-image-square-has-sum1*:

— Half of [1, Proposition 18.1], other half in *basis-image-square-has-sum1*.

fixes $E :: \langle 'a::\text{complex-inner set} \rangle$ **and** $F :: \langle 'b::\text{hilbert-space set} \rangle$

assumes $\langle \text{is-onb } E \rangle$ **and** $\langle \text{is-onb } F \rangle$

shows $\langle ((\lambda e. (\text{norm } (A *_V e))^2) \text{ has-sum } t) E \longleftrightarrow ((\lambda(e,f). (\text{cmod } (f \cdot_C (A *_V e))))^2) \text{ has-sum } t) (E \times F) \rangle$

proof (*rule iffI*)

assume *asm*: $\langle ((\lambda e. (\text{norm } (A *_V e))^2) \text{ has-sum } t) E \rangle$

have *sum1*: $\langle t = (\sum_{\infty e \in E. (\text{norm } (A *_V e))^2) \rangle$

using *asm infsunI* **by** *blast*

have *abs1*: $\langle (\lambda e. (\text{norm } (A *_V e))^2) \text{ abs-summable-on } E \rangle$

using *asm summable-on-def* **by** *auto*

have *sum2*: $\langle t = (\sum_{\infty e \in E. \sum_{\infty f \in F. (\text{cmod } (f \cdot_C (A *_V e))))^2} \rangle$

apply (*subst sum1*)

apply (*rule infsun-cong*)

using *assms(2)*

by (*simp add: is-onb-def flip: parseval-identity*)

have *abs2*: $\langle (\lambda e. \sum_{\infty f \in F. (\text{cmod } (f \cdot_C (A *_V e))))^2) \text{ abs-summable-on } E \rangle$

using - *abs1* **apply** (*rule summable-on-cong[THEN iffD2]*)

apply (*subst parseval-identity*)

using *assms(2)* **by** (*auto simp: is-onb-def*)

have *abs3*: $\langle (\lambda(x, y). (\text{cmod } (y \cdot_C (A *_V x))))^2) \text{ abs-summable-on } E \times F \rangle$

thm *abs-summable-on-Sigma-iff*

apply (*rule abs-summable-on-Sigma-iff[THEN iffD2]*, *rule conjI*)

using *abs2* **apply** (*auto simp del: real-norm-def*)

using *assms(2) parseval-abs-summable* **apply** *blast*

by *auto*

have *sum3*: $\langle t = (\sum_{\infty (e,f) \in E \times F. (\text{cmod } (f \cdot_C (A *_V e))))^2 \rangle$

apply (*subst sum2*)

apply (*subst infsun-Sigma'-banach[symmetric]*)

using *abs3 abs-summable-summable* **apply** *blast*

by *auto*

then show $\langle ((\lambda(e,f). (\text{cmod } (f \cdot_C (A *_V e))))^2) \text{ has-sum } t) (E \times F) \rangle$

using *abs3 abs-summable-summable has-sum-infsun* **by** *blast*

next

assume *asm*: $\langle ((\lambda(e,f). (\text{cmod } (f \cdot_C (A *_V e))))^2) \text{ has-sum } t) (E \times F) \rangle$

have *abs3*: $\langle (\lambda(x, y). (\text{cmod } (y \cdot_C (A *_V x))))^2) \text{ abs-summable-on } E \times F \rangle$

using *asm summable-on-def summable-on-iff-abs-summable-on-real*

by *blast*

have *sum3*: $\langle t = (\sum_{\infty (e,f) \in E \times F. (\text{cmod } (f \cdot_C (A *_V e))))^2 \rangle$

using *asm infsunI* **by** *blast*

have *sum2*: $\langle t = (\sum_{\infty e \in E. \sum_{\infty f \in F. (\text{cmod } (f \cdot_C (A *_V e))))^2 \rangle$

by (*metis (mono-tags, lifting) asm infsun-Sigma'-banach infsun-cong sum3 summable-iff-has-sum-infsun*)

have *abs2*: $\langle (\lambda e. \sum_{\infty f \in F. (\text{cmod } (f \cdot_C (A *_V e))))^2) \text{ abs-summable-on } E \rangle$


```

by (smt (verit, del-insts) abs3 summable-on-Sigma-banach summable-on-cong summable-on-iff-abs-summable-on-re
have sum1: ⟨t = (∑∞ e∈E. (norm (A *V e))2)⟩
  apply (subst sum2)
  apply (rule infsum-cong)
  using assms
  by (auto intro!: simp: parseval-identity is-onb-def)
have abs1: ⟨(λe. (norm (A *V e))2) abs-summable-on E⟩
  using assms abs2
  by (auto intro!: simp: parseval-identity is-onb-def)
show ⟨(λe. (norm (A *V e))2) has-sum t⟩ E⟩
  using abs1 sum1 by auto
qed

```

lemma *basis-image-square-has-sum2*:

— Half of [1, Proposition 18.1], other half in *basis-image-square-has-sum1*.

fixes $E :: \langle 'a::\text{hilbert-space set} \rangle$ **and** $F :: \langle 'b::\text{hilbert-space set} \rangle$

assumes $\langle \text{is-onb } E \rangle$ **and** $\langle \text{is-onb } F \rangle$

shows $\langle ((\lambda e. (\text{norm } (A *_{V} e))^2) \text{ has-sum } t) E \longleftrightarrow ((\lambda f. (\text{norm } (A *_{V} f))^2) \text{ has-sum } t) F \rangle$

proof —

have $\langle ((\lambda e. (\text{norm } (A *_{V} e))^2) \text{ has-sum } t) E \longleftrightarrow ((\lambda(e,f). (\text{cmod } (f \cdot_C (A *_{V} e)))^2) \text{ has-sum } t) (E \times F) \rangle$

using *basis-image-square-has-sum1* *assms* **by** *blast*

also have $\langle \dots \longleftrightarrow ((\lambda(e,f). (\text{cmod } ((A *_{V} f) \cdot_C e))^2) \text{ has-sum } t) (E \times F) \rangle$

apply (*subst cinner-adj-left*)

by (*rule refl*)

also have $\langle \dots \longleftrightarrow ((\lambda(f,e). (\text{cmod } ((A *_{V} f) \cdot_C e))^2) \text{ has-sum } t) (F \times E) \rangle$

apply (*subst asm-rl[of ⟨F × E = prod.swap ‘(E × F)⟩]*)

apply *force*

apply (*subst has-sum-reindex*)

by (*auto simp: o-def*)

also have $\langle \dots \longleftrightarrow ((\lambda f. (\text{norm } (A *_{V} f))^2) \text{ has-sum } t) F \rangle$

apply (*subst cinner-commute, subst complex-mod-cnj*)

using *basis-image-square-has-sum1* *assms*

by *blast*

finally show *?thesis*

by —

qed

12.2 Trace-norm and trace-class

lemma *trace-norm-basis-invariance*:

assumes $\langle \text{is-onb } E \rangle$ **and** $\langle \text{is-onb } F \rangle$

shows $\langle ((\lambda e. \text{cmod } (e \cdot_C (\text{abs-op } A *_{V} e))) \text{ has-sum } t) E \longleftrightarrow ((\lambda f. \text{cmod } (f \cdot_C (\text{abs-op } A *_{V} f))) \text{ has-sum } t) F \rangle$

— [1], Corollary 18.2

proof —

define B **where** $\langle B = \text{sqrt-op } (\text{abs-op } A) \rangle$

have $\langle \text{complex-of-real } (\text{cmod } (e \cdot_C (\text{abs-op } A *_{V} e))) = (B *_{V} B *_{V} e) \cdot_C e \rangle$ **for** e

apply (*simp add: B-def positive-selfadjointI[unfolding selfadjoint-def] flip: cblinfun-apply-cblinfun-compose*)

by $\langle \text{metis } \text{abs-op-pos } \text{abs-pos } \text{cinner-commute } \text{cinner-pos-if-pos } \text{complex-cnj-complex-of-real } \text{complex-of-real-cmod} \rangle$
also have $\langle \dots e = \text{complex-of-real } ((\text{norm } (B *_{\mathcal{V}} e))^2) \rangle$ **for** e
apply $\langle \text{subst } \text{cdot-square-norm}[\text{symmetric}] \rangle$
apply $\langle \text{subst } \text{cinner-adj-left}[\text{symmetric}] \rangle$
by $\langle \text{simp add: } B\text{-def} \rangle$
finally have $*$: $\langle \text{cmod } (e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e)) = (\text{norm } (B *_{\mathcal{V}} e))^2 \rangle$ **for** e
by $\langle \text{metis } \text{Re-complex-of-real} \rangle$

have $\langle ((\lambda e. \text{cmod } (e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e))) \text{ has-sum } t) E \longleftrightarrow ((\lambda e. (\text{norm } (B *_{\mathcal{V}} e))^2) \text{ has-sum } t) E \rangle$
by $\langle \text{simp add: } * \rangle$
also have $\langle \dots = ((\lambda f. (\text{norm } (B *_{\mathcal{V}} f))^2) \text{ has-sum } t) F \rangle$
apply $\langle \text{subst } \text{basis-image-square-has-sum2}[\text{where } F=F] \rangle$
by $\langle \text{simp-all add: } \text{assms} \rangle$
also have $\langle \dots = ((\lambda f. (\text{norm } (B *_{\mathcal{V}} f))^2) \text{ has-sum } t) F \rangle$
using $\text{basis-image-square-has-sum2 } \text{assms}(2)$ **by** blast
also have $\langle \dots = ((\lambda e. \text{cmod } (e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e))) \text{ has-sum } t) F \rangle$
by $\langle \text{simp add: } * \rangle$
finally show $?thesis$
by simp
qed

definition $\text{trace-class} :: \langle 'a::\text{chilbert-space} \Rightarrow_{\mathcal{C}L} 'b::\text{complex-inner} \rangle \Rightarrow \text{bool}$
where $\langle \text{trace-class } A \longleftrightarrow (\exists E. \text{is-onb } E \wedge (\lambda e. e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e)) \text{ abs-summable-on } E) \rangle$

lemma trace-classI :
assumes $\langle \text{is-onb } E \rangle$ **and** $\langle (\lambda e. e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e)) \text{ abs-summable-on } E \rangle$
shows $\langle \text{trace-class } A \rangle$
using $\text{assms}(1) \text{ assms}(2) \text{ trace-class-def}$ **by** blast

lemma $\text{trace-class-iff-summable}$:
assumes $\langle \text{is-onb } E \rangle$
shows $\langle \text{trace-class } A \longleftrightarrow (\lambda e. e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e)) \text{ abs-summable-on } E \rangle$
apply $\langle \text{auto intro!: } \text{trace-classI } \text{assms } \text{simp: } \text{trace-class-def} \rangle$
using $\text{assms } \text{summable-on-def } \text{trace-norm-basis-invariance}$ **by** blast

lemma $\text{trace-class-0}[\text{simp}]$: $\langle \text{trace-class } 0 \rangle$
unfolding trace-class-def
by $\langle \text{auto intro!: } \text{exI}[\text{of - some-chilbert-basis}] \text{ simp: } \text{is-onb-def } \text{is-normal-some-chilbert-basis} \rangle$

lemma $\text{trace-class-uminus}$: $\langle \text{trace-class } t \Longrightarrow \text{trace-class } (-t) \rangle$
by $\langle \text{auto simp add: } \text{trace-class-def} \rangle$

lemma $\text{trace-class-uminus-iff}[\text{simp}]$: $\langle \text{trace-class } (-a) = \text{trace-class } a \rangle$
by $\langle \text{auto simp add: } \text{trace-class-def} \rangle$

definition trace-norm where $\langle \text{trace-norm } A = (\text{if trace-class } A \text{ then } (\sum_{\infty} e \in \text{some-chilbert-basis. } e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e))) \text{ else } 0 \rangle$

definition trace where $\langle \text{trace } A = (\text{if trace-class } A \text{ then } (\sum_{\infty} e \in \text{some-chilbert-basis. } e \cdot_C (A *_{\mathcal{V}} e))) \text{ else } 0 \rangle$

lemma trace-0[simp]: $\langle \text{trace } 0 = 0 \rangle$
unfolding trace-def by simp

lemma trace-class-abs-op[simp]: $\langle \text{trace-class } (\text{abs-op } A) = \text{trace-class } A \rangle$
unfolding trace-class-def
by simp

lemma trace-abs-op[simp]: $\langle \text{trace } (\text{abs-op } A) = \text{trace-norm } A \rangle$

proof (cases $\langle \text{trace-class } A \rangle$)

case True

have pos: $\langle e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e) \geq 0 \rangle$ **for** e

by (simp add: cinner-pos-if-pos)

then have abs: $\langle e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e) = \text{abs } (e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e)) \rangle$ **for** e

by (simp add: abs-pos)

have $\langle \text{trace } (\text{abs-op } A) = (\sum_{\infty} e \in \text{some-chilbert-basis. } e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e)) \rangle$

by (simp add: trace-def True)

also have $\langle \dots = (\sum_{\infty} e \in \text{some-chilbert-basis. } \text{complex-of-real } (\text{cmod } (e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e)))) \rangle$

using pos abs complex-of-real-cmod **by** presburger

also have $\langle \dots = \text{complex-of-real } (\sum_{\infty} e \in \text{some-chilbert-basis. } \text{cmod } (e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e))) \rangle$

by (simp add: infsum-of-real)

also have $\langle \dots = \text{trace-norm } A \rangle$

by (simp add: trace-norm-def True)

finally show ?thesis

by –

next

case False

then show ?thesis

by (simp add: trace-def trace-norm-def)

qed

lemma trace-norm-pos: $\langle \text{trace-norm } A = \text{trace } A \rangle$ **if** $\langle A \geq 0 \rangle$

by (metis abs-op-id-on-pos that trace-abs-op)

lemma trace-norm-alt-def:

assumes $\langle \text{is-onb } B \rangle$

shows $\langle \text{trace-norm } A = (\text{if trace-class } A \text{ then } (\sum_{\infty} e \in B. \text{cmod } (e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e))) \text{ else } 0) \rangle$

by (metis (mono-tags, lifting) assms infsum-eqI' is-onb-some-chilbert-basis trace-norm-basis-invariance trace-norm-def)

lemma trace-class-finite-dim[simp]: $\langle \text{trace-class } A \rangle$ **for** $A :: \langle 'a :: \{ \text{cfinite-dim, chilbert-space} \} \Rightarrow_{CL}$

```

'b::complex-inner
  apply (subst trace-class-iff-summable[of some-chilbert-basis])
  by (auto intro!: summable-on-finite)

lemma trace-class-scaleC: ‹trace-class (c *C a)› if ‹trace-class a›
proof –
  from that obtain B where ‹is-onb B› and ‹(λx. x *C (abs-op a *V x)) abs-summable-on B›
  by (auto simp: trace-class-def)
  then show ?thesis
  by (auto intro!: exI[of - B] summable-on-cmult-right simp: trace-class-def ‹is-onb B› abs-op-scaleC
norm-mult)
qed

lemma trace-scaleC: ‹trace (c *C a) = c * trace a›
proof –
  consider (trace-class) ‹trace-class a› | (c0) ‹c = 0› | (non-trace-class) ‹¬ trace-class a› ‹c ≠ 0›
  by auto
  then show ?thesis
proof cases
  case trace-class
  then have ‹trace-class (c *C a)›
  by (rule trace-class-scaleC)
  then have ‹trace (c *C a) = (∑∞ e ∈ some-chilbert-basis. e *C (c *C a *V e))›
  unfolding trace-def by simp
  also have ‹... = c * (∑∞ e ∈ some-chilbert-basis. e *C (a *V e))›
  by (auto simp: infsum-cmult-right)
  also from trace-class have ‹... = c * trace a›
  by (simp add: Trace-Class.trace-def)
  finally show ?thesis
  by –
next
  case c0
  then show ?thesis
  by simp
next
  case non-trace-class
  then have ‹¬ trace-class (c *C a)›
  by (metis divideC-field-simps(2) trace-class-scaleC)
  with non-trace-class show ?thesis
  by (simp add: trace-def)
qed
qed

lemma trace-uminus: ‹trace (- a) = - trace a›
by (metis mult-minus1 scaleC-minus1-left trace-scaleC)

lemma trace-norm-0[simp]: ‹trace-norm 0 = 0›
by (auto simp: trace-norm-def)

```

lemma *trace-norm-nneg*[simp]: $\langle \text{trace-norm } a \geq 0 \rangle$
apply (*cases* $\langle \text{trace-class } a \rangle$)
by (*auto simp: trace-norm-def infsum-nonneg*)

lemma *trace-norm-scaleC*: $\langle \text{trace-norm } (c *_C a) = \text{norm } c * \text{trace-norm } a \rangle$
proof –
consider (*trace-class*) $\langle \text{trace-class } a \rangle$ | (*c0*) $\langle c = 0 \rangle$ | (*non-trace-class*) $\langle \neg \text{trace-class } a \rangle$ $\langle c \neq 0 \rangle$
by *auto*
then show ?thesis
proof *cases*
case *trace-class*
then have $\langle \text{trace-class } (c *_C a) \rangle$
by (*rule trace-class-scaleC*)
then have $\langle \text{trace-norm } (c *_C a) = (\sum_{\infty} e \in \text{some-chilbert-basis. norm } (e \cdot_C (\text{abs-op } (c *_C a) *_V e))) \rangle$
unfolding *trace-norm-def* **by** *simp*
also have $\langle \dots = \text{norm } c * (\sum_{\infty} e \in \text{some-chilbert-basis. norm } (e \cdot_C (\text{abs-op } a *_V e))) \rangle$
by (*auto simp: infsum-cmult-right' abs-op-scaleC norm-mult*)
also from *trace-class* **have** $\langle \dots = \text{norm } c * \text{trace-norm } a \rangle$
by (*simp add: trace-norm-def*)
finally show ?thesis
by –
next
case *c0*
then show ?thesis
by *simp*
next
case *non-trace-class*
then have $\langle \neg \text{trace-class } (c *_C a) \rangle$
by (*metis divideC-field-simps(2) trace-class-scaleC*)
with *non-trace-class* **show** ?thesis
by (*simp add: trace-norm-def*)
qed
qed

lemma *trace-norm-nondegenerate*: $\langle a = 0 \rangle$ **if** $\langle \text{trace-class } a \rangle$ **and** $\langle \text{trace-norm } a = 0 \rangle$
proof (*rule ccontr*)
assume $\langle a \neq 0 \rangle$
then have $\langle \text{abs-op } a \neq 0 \rangle$
using *abs-op-nondegenerate* **by** *blast*
then obtain x **where** $\langle x \cdot_C (\text{abs-op } a *_V x) \neq 0 \rangle$
by (*metis cblinfun.zero-left cblinfun-cinner-eqI cinner-zero-right*)
then have $\langle \text{norm } x \neq 0 \rangle$
by *auto*
then have $\langle \text{sgn } x \cdot_C (\text{abs-op } a *_V \text{sgn } x) \neq 0 \rangle$ **and** [simp]: $\langle \text{norm } (\text{sgn } x) = 1 \rangle$
unfolding *sgn-div-norm* **using** x **by** (*auto simp: cblinfun.scaleR-right*)

```

obtain  $B$  where  $\text{sgn}x\text{-}B$ :  $\langle \{ \text{sgn } x \} \subseteq B \rangle$  and  $\langle \text{is-onb } B \rangle$ 
  apply  $\text{atomize-elim}$  apply ( $\text{rule orthonormal-basis-exists}$ )
  using  $\langle \text{norm } x \neq 0 \rangle$  by ( $\text{auto simp: is-ortho-set-def sgn-div-norm}$ )

from  $\langle \text{is-onb } B \rangle$  that
have  $\text{summable}$ :  $\langle (\lambda e. e \cdot_C (\text{abs-op } a *_V e)) \text{ abs-summable-on } B \rangle$ 
  using  $\text{trace-class-iff-summable}$  by  $\text{fastforce}$ 

from  $\text{that}$  have  $\langle 0 = \text{trace-norm } a \rangle$ 
  by  $\text{simp}$ 
also from  $\langle \text{is-onb } B \rangle$  have  $\langle \text{trace-norm } a = (\sum_{\infty e \in B. cmod (e \cdot_C (\text{abs-op } a *_V e))) \rangle$ 
  by ( $\text{smt (verit, ccfv-SIG) abs-norm-cancel infsum-cong infsum-not-exists real-norm-def trace-class-def}$ 
 $\text{trace-norm-alt-def}$ )
also have  $\langle \dots \geq (\sum_{\infty e \in \{ \text{sgn } x \}. cmod (e \cdot_C (\text{abs-op } a *_V e))) \rangle$  (is  $\langle - \geq \dots \rangle$ )
  apply ( $\text{rule infsum-mono2}$ )
  using  $\text{summable sgn}x\text{-}B$  by  $\text{auto}$ 
also from  $\text{ax}x'$  have  $\langle \dots > 0 \rangle$ 
  by ( $\text{simp add: is-orthogonal-sym } \text{ax}x'$ )
finally show  $\text{False}$ 
  by  $\text{simp}$ 
qed

```

```

typedef (overloaded) ( $'a::\text{hilbert-space}$ ,  $'b::\text{hilbert-space}$ )  $\text{trace-class} = \langle \text{Collect trace-class} ::$ 
 $(\text{'a} \Rightarrow_{CL} \text{'b}) \text{ set} \rangle$ 
  morphisms  $\text{from-trace-class}$   $\text{Abs-trace-class}$ 
  by ( $\text{auto intro: exI[of - 0]}$ )
setup-lifting  $\text{type-definition-trace-class}$ 

```

```

lemma  $\text{trace-class-from-trace-class}$ [ $\text{simp}$ ]:  $\langle \text{trace-class (from-trace-class } t \rangle$ 
  using  $\text{from-trace-class}$  by  $\text{blast}$ 

```

```

lemma  $\text{trace-pos}$ :  $\langle \text{trace } a \geq 0 \rangle$  if  $\langle a \geq 0 \rangle$ 
  by ( $\text{metis abs-op-def complex-of-real-nn-iff sqrt-op-unique that trace-abs-op trace-norm-nneg}$ )

```

```

lemma  $\text{trace-adj-prelim}$ :  $\langle \text{trace } (a^*) = \text{cnj } (\text{trace } a) \rangle$  if  $\langle \text{trace-class } a \rangle$  and  $\langle \text{trace-class } (a^*) \rangle$ 
  — We will later strengthen this as  $\text{trace-adj}$  and then hide this fact.
  by ( $\text{simp add: trace-def that flip: cinner-adj-right infsum-cn}$ )

```

12.3 Hilbert-Schmidt operators

```

definition  $\text{hilbert-schmidt}$  where  $\langle \text{hilbert-schmidt } a \iff \text{trace-class } (a^* o_{CL} a) \rangle$ 

```

```

definition  $\text{hilbert-schmidt-norm}$  where  $\langle \text{hilbert-schmidt-norm } a = \text{sqrt } (\text{trace-norm } (a^* o_{CL} a)) \rangle$ 

```

```

lemma  $\text{hilbert-schmidtI}$ :  $\langle \text{hilbert-schmidt } a \rangle$  if  $\langle \text{trace-class } (a^* o_{CL} a) \rangle$ 
  using  $\text{that unfolding hilbert-schmidt-def}$  by  $\text{simp}$ 

```

```

lemma  $\text{hilbert-schmidt-0}$ [ $\text{simp}$ ]:  $\langle \text{hilbert-schmidt } 0 \rangle$ 

```

unfolding *hilbert-schmidt-def* **by** *simp*

lemma *hilbert-schmidt-norm-pos*[*simp*]: $\langle \text{hilbert-schmidt-norm } a \geq 0 \rangle$
by (*auto simp: hilbert-schmidt-norm-def*)

lemma *has-sum-hilbert-schmidt-norm-square*:

— [1], Proposition 18.6 (a)

assumes $\langle \text{is-onb } B \rangle$ **and** $\langle \text{hilbert-schmidt } a \rangle$

shows $\langle ((\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ has-sum } (\text{hilbert-schmidt-norm } a)^2) B \rangle$

proof —

from $\langle \text{hilbert-schmidt } a \rangle$

have $\langle \text{trace-class } (a *_{\mathcal{O}CL} a) \rangle$

using *hilbert-schmidt-def* **by** *blast*

with $\langle \text{is-onb } B \rangle$ **have** $\langle ((\lambda x. \text{cmod } (x \cdot_{\mathcal{C}} ((a *_{\mathcal{O}CL} a) *_{\mathcal{V}} x))) \text{ has-sum } \text{trace-norm } (a *_{\mathcal{O}CL} a)) B \rangle$

by (*metis* (*no-types*, *lifting*) *abs-op-def* *has-sum-cong* *has-sum-infsum* *positive-cblinfun-squareI* *sqrt-op-unique* *trace-class-def* *trace-norm-alt-def* *trace-norm-basis-invariance*)

then show *?thesis*

by (*auto simp: cinner-adj-right cdot-square-norm of-real-power norm-power hilbert-schmidt-norm-def*)

qed

lemma *summable-hilbert-schmidt-norm-square*:

— [1], Proposition 18.6 (a)

assumes $\langle \text{is-onb } B \rangle$ **and** $\langle \text{hilbert-schmidt } a \rangle$

shows $\langle (\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ summable-on } B \rangle$

using *assms(1)* *assms(2)* *has-sum-hilbert-schmidt-norm-square* *summable-on-def* **by** *blast*

lemma *summable-hilbert-schmidt-norm-square-converse*:

assumes $\langle \text{is-onb } B \rangle$

assumes $\langle (\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ summable-on } B \rangle$

shows $\langle \text{hilbert-schmidt } a \rangle$

proof —

from *assms(2)*

have $\langle (\lambda x. \text{cmod } (x \cdot_{\mathcal{C}} ((a *_{\mathcal{O}CL} a) *_{\mathcal{V}} x))) \text{ summable-on } B \rangle$

by (*metis* (*no-types*, *lifting*) *cblinfun-apply-cblinfun-compose* *cinner-adj-right* *cinner-pos-if-pos* *cmod-Re* *positive-cblinfun-squareI* *power2-norm-eq-cinner'* *summable-on-cong*)

then have $\langle \text{trace-class } (a *_{\mathcal{O}CL} a) \rangle$

by (*metis* (*no-types*, *lifting*) *abs-op-def* *assms(1)* *positive-cblinfun-squareI* *sqrt-op-unique* *summable-on-cong* *trace-class-def*)

then show *?thesis*

using *hilbert-schmidtI* **by** *blast*

qed

lemma *infsum-hilbert-schmidt-norm-square*:

— [1], Proposition 18.6 (a)

assumes $\langle \text{is-onb } B \rangle$ **and** $\langle \text{hilbert-schmidt } a \rangle$

shows $\langle (\sum_{\infty} x \in B. (\text{norm } (a *_{\mathcal{V}} x))^2) = ((\text{hilbert-schmidt-norm } a)^2) \rangle$

using *assms* *has-sum-hilbert-schmidt-norm-square* *infsumI* **by** *blast*

lemma

— [1], Proposition 18.6 (d)

assumes $\langle \text{hilbert-schmidt } b \rangle$

shows $\text{hilbert-schmidt-comp-right}$: $\langle \text{hilbert-schmidt } (a \text{ } o_{CL} \ b) \rangle$

and $\text{hilbert-schmidt-norm-comp-right}$: $\langle \text{hilbert-schmidt-norm } (a \text{ } o_{CL} \ b) \leq \text{norm } a * \text{hilbert-schmidt-norm } b \rangle$

proof —

define $B :: \langle 'a \text{ set} \rangle$ **where** $\langle B = \text{some-chilbert-basis} \rangle$

have $[simp]$: $\langle \text{is-onb } B \rangle$

by $(\text{simp add: } B\text{-def})$

have leq : $\langle (\text{norm } ((a \text{ } o_{CL} \ b) *_{V} \ x))^2 \leq (\text{norm } a)^2 * (\text{norm } (b *_{V} \ x))^2 \rangle$ **for** x

by $(\text{metis cblinfun-apply-cblinfun-compose norm-cblinfun norm-ge-zero power-mono power-mult-distrib})$

have $\langle (\lambda x. (\text{norm } (b *_{V} \ x))^2) \text{ summable-on } B \rangle$

using $\langle \text{is-onb } B \rangle$ $\text{summable-hilbert-schmidt-norm-square assms}$ **by** blast

then have $sum2$: $\langle (\lambda x. (\text{norm } a)^2 * (\text{norm } (b *_{V} \ x))^2) \text{ summable-on } B \rangle$

using $\text{summable-on-cmult-right}$ **by** blast

then have $\langle (\lambda x. ((\text{norm } a)^2 * (\text{norm } (b *_{V} \ x))^2)) \text{ abs-summable-on } B \rangle$

by auto

then have $\langle (\lambda x. (\text{norm } ((a \text{ } o_{CL} \ b) *_{V} \ x))^2) \text{ abs-summable-on } B \rangle$

apply $(\text{rule abs-summable-on-comparison-test})$

using leq **by** force

then have $sum5$: $\langle (\lambda x. (\text{norm } ((a \text{ } o_{CL} \ b) *_{V} \ x))^2) \text{ summable-on } B \rangle$

by auto

then show $[simp]$: $\langle \text{hilbert-schmidt } (a \text{ } o_{CL} \ b) \rangle$

using $\langle \text{is-onb } B \rangle$

by $(\text{rule summable-hilbert-schmidt-norm-square-converse}[\text{rotated}])$

have $\langle (\text{hilbert-schmidt-norm } (a \text{ } o_{CL} \ b))^2 = (\sum_{\infty x \in B. (\text{norm } ((a \text{ } o_{CL} \ b) *_{V} \ x))^2) \rangle$

apply $(\text{rule infsum-hilbert-schmidt-norm-square}[\text{symmetric}])$

by simp-all

also have $\langle \dots \leq (\sum_{\infty x \in B. (\text{norm } a)^2 * (\text{norm } (b *_{V} \ x))^2) \rangle$

using $sum5 \ sum2 \ leq$ **by** $(\text{rule infsum-mono})$

also have $\langle \dots = (\text{norm } a)^2 * (\sum_{\infty x \in B. (\text{norm } (b *_{V} \ x))^2) \rangle$

by $(\text{simp add: infsum-cmult-right}')$

also have $\langle \dots = (\text{norm } a)^2 * (\text{hilbert-schmidt-norm } b)^2 \rangle$

by $(\text{simp add: assms infsum-hilbert-schmidt-norm-square})$

finally show $\langle \text{hilbert-schmidt-norm } (a \text{ } o_{CL} \ b) \leq \text{norm } a * \text{hilbert-schmidt-norm } b \rangle$

apply $(\text{rule-tac power2-le-imp-le})$

by $(\text{auto intro!: mult-nonneg-nonneg simp: power-mult-distrib})$

qed

lemma $\text{hilbert-schmidt-adj}[simp]$:

— Implicit in [1], Proposition 18.6 (b)

assumes $\langle \text{hilbert-schmidt } a \rangle$

shows $\langle \text{hilbert-schmidt } (a^*) \rangle$

proof –
from *assms*
have $\langle (\lambda e. (\text{norm } (a *_{\mathcal{V}} e))^2) \text{ summable-on some-chilbert-basis} \rangle$
using *is-onb-some-chilbert-basis summable-hilbert-schmidt-norm-square* **by** *blast*
then have $\langle (\lambda e. (\text{norm } (a *_{\mathcal{V}} e))^2) \text{ summable-on some-chilbert-basis} \rangle$
by (*metis basis-image-square-has-sum2 is-onb-some-chilbert-basis summable-on-def*)
then show *?thesis*
using *is-onb-some-chilbert-basis summable-hilbert-schmidt-norm-square-converse* **by** *blast*
qed

lemma *hilbert-schmidt-norm-adj[simp]*:

– [1], Proposition 18.6 (b)

shows $\langle \text{hilbert-schmidt-norm } (a*) = \text{hilbert-schmidt-norm } a \rangle$

proof (*cases* $\langle \text{hilbert-schmidt } a \rangle$)

case *True*

then have $\langle ((\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ has-sum } (\text{hilbert-schmidt-norm } a)^2) \text{ some-chilbert-basis} \rangle$

by (*simp add: has-sum-hilbert-schmidt-norm-square*)

then have 1: $\langle ((\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ has-sum } (\text{hilbert-schmidt-norm } a)^2) \text{ some-chilbert-basis} \rangle$

by (*metis basis-image-square-has-sum2 is-onb-some-chilbert-basis*)

from *True*

have $\langle \text{hilbert-schmidt } (a*) \rangle$

by *simp*

then have 2: $\langle ((\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ has-sum } (\text{hilbert-schmidt-norm } (a*))^2) \text{ some-chilbert-basis} \rangle$

by (*simp add: has-sum-hilbert-schmidt-norm-square*)

from 1 2 **show** *?thesis*

by (*metis abs-of-nonneg hilbert-schmidt-norm-pos infsumI real-sqrt-abs*)

next

case *False*

then have $\langle \neg \text{hilbert-schmidt } (a*) \rangle$

using *hilbert-schmidt-adj* **by** *fastforce*

then show *?thesis*

by (*metis False hilbert-schmidt-def hilbert-schmidt-norm-def trace-norm-def*)

qed

lemma

– [1], Proposition 18.6 (d)

fixes $a :: \langle 'a :: \text{chilbert-space} \Rightarrow_{\text{CL}} 'b :: \text{chilbert-space} \rangle$ **and** b

assumes $\langle \text{hilbert-schmidt } a \rangle$

shows *hilbert-schmidt-comp-left*: $\langle \text{hilbert-schmidt } (a \circ_{\text{CL}} b) \rangle$

apply (*subst asm-rl[of* $\langle a \circ_{\text{CL}} b = (b *_{\text{CL}} a*) * \rangle$, *simp*)

by (*auto intro!: assms hilbert-schmidt-comp-right hilbert-schmidt-adj simp del: adj-cblinfun-compose*)

lemma

– [1], Proposition 18.6 (d)

fixes $a :: \langle 'a :: \text{chilbert-space} \Rightarrow_{\text{CL}} 'b :: \text{chilbert-space} \rangle$ **and** b

assumes $\langle \text{hilbert-schmidt } a \rangle$

shows *hilbert-schmidt-norm-comp-left*: $\langle \text{hilbert-schmidt-norm } (a \text{ } o_{CL} \text{ } b) \leq \text{norm } b * \text{hilbert-schmidt-norm } a \rangle$

apply (*subst asm-rl*[of $\langle a \text{ } o_{CL} \text{ } b = (b * o_{CL} \text{ } a) * \rangle$], *simp*)

using *hilbert-schmidt-norm-comp-right*[of $\langle a * \rangle \langle b * \rangle$]

by (*auto intro!*: *assms hilbert-schmidt-adj simp del: adj-cblinfun-compose*)

lemma *hilbert-schmidt-scaleC*: $\langle \text{hilbert-schmidt } (c *_{C} a) \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
using *hilbert-schmidt-def that trace-class-scaleC* **by** *fastforce*

lemma *hilbert-schmidt-scaleR*: $\langle \text{hilbert-schmidt } (r *_{R} a) \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
by (*simp add: hilbert-schmidt-scaleC scaleR-scaleC that*)

lemma *hilbert-schmidt-uminus*: $\langle \text{hilbert-schmidt } (- a) \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
by (*metis hilbert-schmidt-scaleC scaleC-minus1-left that*)

lemma *hilbert-schmidt-plus*: $\langle \text{hilbert-schmidt } (t + u) \rangle$ **if** $\langle \text{hilbert-schmidt } t \rangle$ **and** $\langle \text{hilbert-schmidt } u \rangle$

for $t \ u :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$

— [1], Proposition 18.6 (e). We use a different proof than Conway: Our proof of *trace-class-plus* below was easy to adapt to Hilbert-Schmidt operators, so we adapted that one. However, Conway’s proof would most likely work as well, and possibly additionally allow us to weaken the sort of *'b* to *complex-inner*.

proof –

define *II* :: $\langle 'a \Rightarrow_{CL} ('a \times 'a) \rangle$ **where** $\langle II = \text{cblinfun-left} + \text{cblinfun-right} \rangle$

define *JJ* :: $\langle ('b \times 'b) \Rightarrow_{CL} 'b \rangle$ **where** $\langle JJ = \text{cblinfun-left} * + \text{cblinfun-right} * \rangle$

define *t2 u2* **where** $\langle t2 = t * o_{CL} \text{ } t \rangle$ **and** $\langle u2 = u * o_{CL} \text{ } u \rangle$

define *tu* :: $\langle ('a \times 'a) \Rightarrow_{CL} ('b \times 'b) \rangle$ **where** $\langle tu = (\text{cblinfun-left } o_{CL} \text{ } t \text{ } o_{CL} \text{ } \text{cblinfun-left} *) + (\text{cblinfun-right } o_{CL} \text{ } u \text{ } o_{CL} \text{ } \text{cblinfun-right} *) \rangle$

define *tu2* :: $\langle ('a \times 'a) \Rightarrow_{CL} ('a \times 'a) \rangle$ **where** $\langle tu2 = (\text{cblinfun-left } o_{CL} \text{ } t2 \text{ } o_{CL} \text{ } \text{cblinfun-left} *) + (\text{cblinfun-right } o_{CL} \text{ } u2 \text{ } o_{CL} \text{ } \text{cblinfun-right} *) \rangle$

have *t-plus-u*: $\langle t + u = JJ \text{ } o_{CL} \text{ } tu \text{ } o_{CL} \text{ } II \rangle$

apply (*simp add: II-def JJ-def tu-def cblinfun-compose-add-left cblinfun-compose-add-right cblinfun-compose-assoc*)

by (*simp flip: cblinfun-compose-assoc*)

have *tu-tu2*: $\langle tu * o_{CL} \text{ } tu = tu2 \rangle$

by (*simp add: tu-def tu2-def t2-def u2-def cblinfun-compose-add-left cblinfun-compose-add-right cblinfun-compose-assoc adj-plus isometryD[THEN simp-a-oCL-b] cblinfun-right-left-ortho[THEN simp-a-oCL-b] cblinfun-left-right-ortho[THEN simp-a-oCL-b]*)

have $\langle \text{trace-class } tu2 \rangle$

proof (*rule trace-classI*)

define *BL BR B* :: $\langle ('a \times 'a) \text{ set} \rangle$ **where** $\langle BL = \text{some-chilbert-basis} \times \{0\} \rangle$
and $\langle BR = \{0\} \times \text{some-chilbert-basis} \rangle$
and $\langle B = BL \cup BR \rangle$

have $\langle BL \cap BR = \{ \} \rangle$

using *is-ortho-set-some-chilbert-basis*

by (*auto simp: BL-def BR-def is-ortho-set-def*)

show $\langle \text{is-onb } B \rangle$

by (*simp add: BL-def BR-def B-def is-onb-prod*)

```

have ⟨tu2 ≥ 0⟩
by (auto intro!: positive-cblinfunI simp: t2-def u2-def cinner-adj-right tu2-def cblinfun.add-left
cinner-pos-if-pos)
then have abs-tu2: ⟨abs-op tu2 = tu2⟩
  by (metis abs-opI)
have abs-t2: ⟨abs-op t2 = t2⟩
  by (metis abs-opI positive-cblinfun-squareI t2-def)
have abs-u2: ⟨abs-op u2 = u2⟩
  by (metis abs-opI positive-cblinfun-squareI u2-def)

from that(1)
have ⟨(λx. x •C (abs-op t2 *V x)) abs-summable-on some-chilbert-basis⟩
by (simp add: hilbert-schmidt-def t2-def trace-class-iff-summable[OF is-onb-some-chilbert-basis])
then have ⟨(λx. x •C (t2 *V x)) abs-summable-on some-chilbert-basis⟩
  by (simp add: abs-t2)
then have sum-BL: ⟨(λx. x •C (tu2 *V x)) abs-summable-on BL⟩
  apply (subst asm-rl[of ⟨BL = (λx. (x,0)) ‘some-chilbert-basis’⟩])
  by (auto simp: BL-def summable-on-reindex inj-on-def o-def tu2-def cblinfun.add-left)
from that(2)
have ⟨(λx. x •C (abs-op u2 *V x)) abs-summable-on some-chilbert-basis⟩
by (simp add: hilbert-schmidt-def u2-def trace-class-iff-summable[OF is-onb-some-chilbert-basis])
then have ⟨(λx. x •C (u2 *V x)) abs-summable-on some-chilbert-basis⟩
  by (simp add: abs-u2)
then have sum-BR: ⟨(λx. x •C (tu2 *V x)) abs-summable-on BR⟩
  apply (subst asm-rl[of ⟨BR = (λx. (0,x)) ‘some-chilbert-basis’⟩])
  by (auto simp: BR-def summable-on-reindex inj-on-def o-def tu2-def cblinfun.add-left)
from sum-BL sum-BR
show ⟨(λx. x •C (abs-op tu2 *V x)) abs-summable-on B⟩
  using ⟨BL ∩ BR = { }⟩
  by (auto intro!: summable-on-Un-disjoint simp: B-def abs-tu2)
qed
then have ⟨hilbert-schmidt tu⟩
  by (auto simp flip: tu-tu2 intro!: hilbert-schmidtI)
with t-plus-u
show ⟨hilbert-schmidt (t + u)⟩
  by (auto intro: hilbert-schmidt-comp-left hilbert-schmidt-comp-right)
qed

lemma hilbert-schmidt-minus: ⟨hilbert-schmidt (a - b)⟩ if ⟨hilbert-schmidt a⟩ and ⟨hilbert-schmidt
b⟩
  for a b :: ⟨'a::chilbert-space ⇒CL 'b::chilbert-space⟩
  using hilbert-schmidt-plus hilbert-schmidt-uminus that(1) that(2) by fastforce

typedef (overloaded) ('a::chilbert-space,'b::complex-inner) hilbert-schmidt = ⟨Collect hilbert-schmidt
:: ('a ⇒CL 'b) set⟩
  by (auto intro!: exI[of - 0])
setup-lifting type-definition-hilbert-schmidt

instantiation hilbert-schmidt :: (chilbert-space, chilbert-space)

```

```

{zero,scaleC,uminus,plus,minus,dist-norm,sgn-div-norm,uniformity-dist,open-uniformity} be-
gin
lift-definition zero-hilbert-schmidt :: ⟨('a,'b) hilbert-schmidt⟩ is 0 by auto
lift-definition norm-hilbert-schmidt :: ⟨('a,'b) hilbert-schmidt ⇒ real⟩ is hilbert-schmidt-norm
.
lift-definition scaleC-hilbert-schmidt :: ⟨complex ⇒ ('a,'b) hilbert-schmidt ⇒ ('a,'b) hilbert-schmidt⟩
is scaleC
  by (simp add: hilbert-schmidt-scaleC)
lift-definition scaleR-hilbert-schmidt :: ⟨real ⇒ ('a,'b) hilbert-schmidt ⇒ ('a,'b) hilbert-schmidt⟩
is scaleR
  by (simp add: hilbert-schmidt-scaleR)
lift-definition uminus-hilbert-schmidt :: ⟨('a,'b) hilbert-schmidt ⇒ ('a,'b) hilbert-schmidt⟩ is
uminus
  by (simp add: hilbert-schmidt-uminus)
lift-definition minus-hilbert-schmidt :: ⟨('a,'b) hilbert-schmidt ⇒ ('a,'b) hilbert-schmidt ⇒ ('a,'b)
hilbert-schmidt⟩ is minus
  by (simp add: hilbert-schmidt-minus)
lift-definition plus-hilbert-schmidt :: ⟨('a,'b) hilbert-schmidt ⇒ ('a,'b) hilbert-schmidt ⇒ ('a,'b)
hilbert-schmidt⟩ is plus
  by (simp add: hilbert-schmidt-plus)
definition ⟨dist a b = norm (a - b)⟩ for a b :: ⟨('a,'b) hilbert-schmidt⟩
definition ⟨sgn x = inverse (norm x) *R x⟩ for x :: ⟨('a,'b) hilbert-schmidt⟩
definition ⟨uniformity = (INF e∈{0<..}. principal {(x::('a,'b) hilbert-schmidt, y). dist x y <
e})⟩
definition ⟨open U = (∀ x∈U. ∀F (x', y) in INF e∈{0<..}. principal {(x, y). norm (x - y) <
e}. x' = x → y ∈ U)⟩ for U :: ⟨('a,'b) hilbert-schmidt set⟩
instance
proof intro-classes
  show ⟨(*R) r = ((*C) (complex-of-real r)) :: ('a,'b) hilbert-schmidt ⇒ -⟩ for r :: real
    apply (rule ext)
    apply transfer
    by (auto simp: scaleR-scaleC)
  show ⟨dist x y = norm (x - y)⟩ for x y :: ⟨('a,'b) hilbert-schmidt⟩
    by (simp add: dist-hilbert-schmidt-def)
  show ⟨sgn x = inverse (norm x) *R x⟩ for x :: ⟨('a,'b) hilbert-schmidt⟩
    by (simp add: Trace-Class.sgn-hilbert-schmidt-def)
  show ⟨uniformity = (INF e∈{0<..}. principal {(x::('a,'b) hilbert-schmidt, y). dist x y < e})⟩
    using Trace-Class.uniformity-hilbert-schmidt-def by blast
  show ⟨open U = (∀ x∈U. ∀F (x', y) in uniformity. x' = x → y ∈ U)⟩ for U :: ⟨('a,'b)
hilbert-schmidt set⟩
    by (simp add: uniformity-hilbert-schmidt-def open-hilbert-schmidt-def dist-hilbert-schmidt-def)
qed
end

lift-definition hs-compose :: ⟨('b::hilbert-space,'c::complex-inner) hilbert-schmidt
⇒ ('a::hilbert-space,'b) hilbert-schmidt ⇒ ('a,'c) hilbert-schmidt⟩ is
  cblinfun-compose
  by (simp add: hilbert-schmidt-comp-right)

```

lemma

— [1], 18.8 Proposition

fixes $A :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$

shows $\text{trace-class-iff-sqrt-hs}: \langle \text{trace-class } A \longleftrightarrow \text{hilbert-schmidt } (\text{sqrt-op } (\text{abs-op } A)) \rangle$ (**is** $?thesis1$)

and $\text{trace-class-iff-hs-times-hs}: \langle \text{trace-class } A \longleftrightarrow (\exists B (C :: 'a \Rightarrow_{CL} 'a). \text{hilbert-schmidt } B \wedge \text{hilbert-schmidt } C \wedge A = B \circ_{CL} C) \rangle$ (**is** $?thesis2$)

and $\text{trace-class-iff-abs-hs-times-hs}: \langle \text{trace-class } A \longleftrightarrow (\exists B (C :: 'a \Rightarrow_{CL} 'a). \text{hilbert-schmidt } B \wedge \text{hilbert-schmidt } C \wedge \text{abs-op } A = B \circ_{CL} C) \rangle$ (**is** $?thesis3$)

proof —

define $Sq\ W$ **where** $\langle Sq = \text{sqrt-op } (\text{abs-op } A) \rangle$ **and** $\langle W = \text{polar-decomposition } A \rangle$

have $\text{trace-class-sqrt-hs}: \langle \text{hilbert-schmidt } Sq \rangle$ **if** $\langle \text{trace-class } A \rangle$

proof (*rule hilbert-schmidtI*)

from *that*

have $\langle \text{trace-class } (\text{abs-op } A) \rangle$

by *simp*

then show $\langle \text{trace-class } (Sq * \circ_{CL} Sq) \rangle$

by (*auto simp: Sq-def positive-selfadjointI[unfolding selfadjoint-def]*)

qed

have $\text{sqrt-hs-hs-times-hs}: \langle \exists B (C :: 'a \Rightarrow_{CL} 'a). \text{hilbert-schmidt } B \wedge \text{hilbert-schmidt } C \wedge A = B \circ_{CL} C \rangle$

if $\langle \text{hilbert-schmidt } Sq \rangle$

proof —

have $\langle A = W \circ_{CL} \text{abs-op } A \rangle$

by (*simp add: polar-decomposition-correct W-def*)

also have $\langle \dots = (W \circ_{CL} Sq) \circ_{CL} Sq \rangle$

by (*metis Sq-def abs-op-pos cblinfun-compose-assoc positive-selfadjointI sqrt-op-pos sqrt-op-square*)

finally have $\langle A = (W \circ_{CL} Sq) \circ_{CL} Sq \rangle$

by —

then show $?thesis$

apply (*rule-tac exI[of - $\langle W \circ_{CL} Sq \rangle$], rule-tac exI[of - Sq]*)

using *that by (auto simp add: hilbert-schmidt-comp-right)*

qed

have $\text{hs-times-hs-abs-hs-times-hs}: \langle \exists B (C :: 'a \Rightarrow_{CL} 'a). \text{hilbert-schmidt } B \wedge \text{hilbert-schmidt } C \wedge \text{abs-op } A = B \circ_{CL} C \rangle$

if $\langle \exists B (C :: 'a \Rightarrow_{CL} 'a). \text{hilbert-schmidt } B \wedge \text{hilbert-schmidt } C \wedge A = B \circ_{CL} C \rangle$

proof —

from *that obtain B and C :: $\langle 'a \Rightarrow_{CL} 'a \rangle$ where $\langle \text{hilbert-schmidt } B \rangle$ and $\langle \text{hilbert-schmidt } C \rangle$ and $ABC: \langle A = B \circ_{CL} C \rangle$*

by *auto*

from $\langle \text{hilbert-schmidt } B \rangle$

have $\text{hs-WB}: \langle \text{hilbert-schmidt } (W * \circ_{CL} B) \rangle$

by (*simp add: hilbert-schmidt-comp-right*)

have $\langle \text{abs-op } A = W * \circ_{CL} A \rangle$

by (*simp add: W-def polar-decomposition-correct*)

also have $\langle \dots = (W * \circ_{CL} B) \circ_{CL} C \rangle$

by (*metis ABC cblinfun-compose-assoc*)

finally have $\langle \text{abs-op } A = (W * \circ_{CL} B) \circ_{CL} C \rangle$

by —

with *hs-WB* $\langle \text{hilbert-schmidt } C \rangle$
show *?thesis*
by *auto*
qed
have *abs-hs-times-hs-trace-class*: $\langle \text{trace-class } A \rangle$
if $\langle \exists B (C :: 'a \Rightarrow_{CL} 'a). \text{hilbert-schmidt } B \wedge \text{hilbert-schmidt } C \wedge \text{abs-op } A = B \circ_{CL} C \rangle$
proof –
from *that obtain B and C* $:: \langle 'a \Rightarrow_{CL} 'a \rangle$ **where** $\langle \text{hilbert-schmidt } B \rangle$ **and** $\langle \text{hilbert-schmidt } C \rangle$ **and** *ABC*: $\langle \text{abs-op } A = B \circ_{CL} C \rangle$
by *auto*
from $\langle \text{hilbert-schmidt } B \rangle$
have $\langle \text{hilbert-schmidt } (B*) \rangle$
by *simp*
then have $\langle (\lambda e. (\text{norm } (B* *_{\mathcal{V}} e))^2) \text{abs-summable-on some-chilbert-basis} \rangle$
by *(metis is-onb-some-chilbert-basis summable-hilbert-schmidt-norm-square summable-on-iff-abs-summable-on-rea)*
moreover
from $\langle \text{hilbert-schmidt } C \rangle$
have $\langle (\lambda e. (\text{norm } (C *_{\mathcal{V}} e))^2) \text{abs-summable-on some-chilbert-basis} \rangle$
by *(metis is-onb-some-chilbert-basis summable-hilbert-schmidt-norm-square summable-on-iff-abs-summable-on-rea)*
ultimately have $\langle (\lambda e. \text{norm } (B* *_{\mathcal{V}} e) * \text{norm } (C *_{\mathcal{V}} e)) \text{abs-summable-on some-chilbert-basis} \rangle$
apply *(rule-tac abs-summable-product)*
by *(metis (no-types, lifting) power2-eq-square summable-on-cong)+*
then have $\langle (\lambda e. \text{cinner } e (\text{abs-op } A *_{\mathcal{V}} e)) \text{abs-summable-on some-chilbert-basis} \rangle$
proof *(rule Infinite-Sum.abs-summable-on-comparison-test)*
fix $e :: 'a$ **assume** $\langle e \in \text{some-chilbert-basis} \rangle$
have $\langle \text{norm } (e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e)) = \text{norm } ((B* *_{\mathcal{V}} e) \cdot_C (C *_{\mathcal{V}} e)) \rangle$
by *(simp add: ABC cinner-adj-left)*
also have $\langle \dots \leq \text{norm } (B* *_{\mathcal{V}} e) * \text{norm } (C *_{\mathcal{V}} e) \rangle$
by *(rule Cauchy-Schwarz-ineq2)*
also have $\langle \dots = \text{norm } (\text{norm } (B* *_{\mathcal{V}} e) * \text{norm } (C *_{\mathcal{V}} e)) \rangle$
by *simp*
finally show $\langle \text{cmod } (e \cdot_C (\text{abs-op } A *_{\mathcal{V}} e)) \leq \text{norm } (\text{norm } (B* *_{\mathcal{V}} e) * \text{norm } (C *_{\mathcal{V}} e)) \rangle$
by –
qed
then show $\langle \text{trace-class } A \rangle$
apply *(rule trace-classI[rotated])* **by** *simp*
qed
from *trace-class-sqrt-hs sqrt-hs-hs-times-hs hs-times-hs-abs-hs-times-hs abs-hs-times-hs-trace-class*
show *?thesis1* **and** *?thesis2* **and** *?thesis3*
unfolding *Sq-def* **by** *metis+*
qed

lemma *trace-exists*:

– [1], Proposition 18.9

assumes $\langle \text{is-onb } B \rangle$ **and** $\langle \text{trace-class } A \rangle$

shows $\langle (\lambda e. e \cdot_C (A *_{\mathcal{V}} e)) \text{summable-on } B \rangle$

proof –

obtain $b \ c :: \langle 'a \Rightarrow_{CL} 'a \rangle$ **where** $\langle \text{hilbert-schmidt } b \rangle \langle \text{hilbert-schmidt } c \rangle$ **and** *Abc*: $\langle A = c*$

$o_{CL} b$
by (*metis abs-op-pos adj-cblinfun-compose assms(2) double-adj hilbert-schmidt-comp-left hilbert-schmidt-comp-right polar-decomposition-correct polar-decomposition-correct' positive-selfadjointI[unfolded selfadjoint-def] trace-class-iff-hs-times-hs*)

have $\langle (\lambda e. (\text{norm } (b *_{\mathcal{V}} e))^2) \text{ summable-on } B \rangle$
using $\langle \text{hilbert-schmidt } b \rangle \text{ assms}(1) \text{ summable-hilbert-schmidt-norm-square}$ **by** *auto*
moreover have $\langle (\lambda e. (\text{norm } (c *_{\mathcal{V}} e))^2) \text{ summable-on } B \rangle$
using $\langle \text{hilbert-schmidt } c \rangle \text{ assms}(1) \text{ summable-hilbert-schmidt-norm-square}$ **by** *auto*
ultimately have $\langle (\lambda e. ((\text{norm } (b *_{\mathcal{V}} e))^2 + (\text{norm } (c *_{\mathcal{V}} e))^2) / 2) \text{ summable-on } B \rangle$
by (*auto intro!: summable-on-cdivide summable-on-add*)

then have $\langle (\lambda e. (((\text{norm } (b *_{\mathcal{V}} e))^2 + (\text{norm } (c *_{\mathcal{V}} e))^2) / 2) \text{ abs-summable-on } B \rangle$
by *simp*

then have $\langle (\lambda e. e \cdot_C (A *_{\mathcal{V}} e)) \text{ abs-summable-on } B \rangle$

proof (*rule abs-summable-on-comparison-test*)

fix e **assume** $\langle e \in B \rangle$

obtain γ **where** $\langle \text{cmod } \gamma = 1 \rangle$ **and** $\gamma: \langle \gamma * ((b *_{\mathcal{V}} e) \cdot_C (c *_{\mathcal{V}} e)) = \text{abs } ((b *_{\mathcal{V}} e) \cdot_C (c *_{\mathcal{V}} e)) \rangle$

apply *atomize-elim*

apply (*cases* $\langle (b *_{\mathcal{V}} e) \cdot_C (c *_{\mathcal{V}} e) \neq 0 \rangle$)

apply (*rule exI[of - cnpj (sgn ((b *_ℳ e) ·_C (c *_ℳ e)))]*)

apply (*auto simp add: norm-sgn intro!: norm-one*)

by (*metis (no-types, lifting) abs-mult-sgn cblinfun.scaleC-right cblinfun-mult-right.rep-eq cdot-square-norm complex-norm-square complex-scaleC-def mult.comm-neutral norm-one norm-sgn one-cinner-one*)

have $\langle \text{cmod } (e \cdot_C (A *_{\mathcal{V}} e)) = \text{Re } (\text{abs } (e \cdot_C (A *_{\mathcal{V}} e))) \rangle$

by (*metis abs-nn cmod-Re norm-abs*)

also have $\langle \dots = \text{Re } (\text{abs } ((b *_{\mathcal{V}} e) \cdot_C (c *_{\mathcal{V}} e))) \rangle$

by (*metis (mono-tags, lifting) Abs abs-nn cblinfun-apply-cblinfun-compose cinner-adj-left cinner-commute' complex-mod-cnj complex-of-real-cmod norm-abs*)

also have $\langle \dots = \text{Re } (((b *_{\mathcal{V}} e) \cdot_C (\gamma *_{\mathcal{C}} (c *_{\mathcal{V}} e)))) \rangle$

by (*simp add: γ*)

also have $\langle \dots \leq ((\text{norm } (b *_{\mathcal{V}} e))^2 + (\text{norm } (\gamma *_{\mathcal{C}} (c *_{\mathcal{V}} e)))^2) / 2 \rangle$

by (*smt (z3) field-sum-of-halves norm-ge-zero polar-identity-minus zero-le-power-eq-numeral*)

also have $\langle \dots = ((\text{norm } (b *_{\mathcal{V}} e))^2 + (\text{norm } (c *_{\mathcal{V}} e))^2) / 2 \rangle$

by (*simp add: cmod $\gamma = 1$*)

also have $\langle \dots \leq \text{norm } (((\text{norm } (b *_{\mathcal{V}} e))^2 + (\text{norm } (c *_{\mathcal{V}} e))^2) / 2) \rangle$

by *simp*

finally show $\langle \text{cmod } (e \cdot_C (A *_{\mathcal{V}} e)) \leq \text{norm } (((\text{norm } (b *_{\mathcal{V}} e))^2 + (\text{norm } (c *_{\mathcal{V}} e))^2) / 2) \rangle$

by $-$

qed

then show *?thesis*

by (*metis abs-summable-summable*)

qed

lemma *trace-plus-prelim*:

assumes $\langle \text{trace-class } a \rangle \langle \text{trace-class } b \rangle \langle \text{trace-class } (a+b) \rangle$
 — We will later strengthen this as *trace-plus* and then hide this fact.
shows $\langle \text{trace } (a + b) = \text{trace } a + \text{trace } b \rangle$
by (*auto simp add: asms infsum-add trace-def cblinfun.add-left cinner-add-right*
intro!: infsum-add trace-exists)

lemma *hs-times-hs-trace-class*:

fixes $B :: \langle 'b::\text{hilbert-space} \Rightarrow_{CL} 'c::\text{hilbert-space} \rangle$ **and** $C :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
assumes $\langle \text{hilbert-schmidt } B \rangle$ **and** $\langle \text{hilbert-schmidt } C \rangle$
shows $\langle \text{trace-class } (B \circ_{CL} C) \rangle$

— Not an immediate consequence of *trace-class-iff-hs-times-hs* because here the types of B, C are more general.

proof —

define $A Sq W$ **where** $\langle A = B \circ_{CL} C \rangle$ **and** $\langle Sq = \text{sqrt-op } (abs-op A) \rangle$ **and** $\langle W = \text{polar-decomposition } A \rangle$

from $\langle \text{hilbert-schmidt } B \rangle$

have $hs-WB: \langle \text{hilbert-schmidt } (W^* \circ_{CL} B) \rangle$

by (*simp add: hilbert-schmidt-comp-right*)

have $\langle abs-op A = W^* \circ_{CL} A \rangle$

by (*simp add: W-def polar-decomposition-correct'*)

also have $\langle \dots = (W^* \circ_{CL} B) \circ_{CL} C \rangle$

by (*metis A-def cblinfun-compose-assoc*)

finally have $abs-op-A: \langle abs-op A = (W^* \circ_{CL} B) \circ_{CL} C \rangle$

by —

from $\langle \text{hilbert-schmidt } (W^* \circ_{CL} B) \rangle$

have $\langle \text{hilbert-schmidt } (B^* \circ_{CL} W) \rangle$

by (*simp add: asms(1) hilbert-schmidt-comp-left*)

then have $\langle (\lambda e. (\text{norm } ((B^* \circ_{CL} W) *_V e))^2) \text{ abs-summable-on some-chilbert-basis} \rangle$

by (*metis is-onb-some-chilbert-basis summable-hilbert-schmidt-norm-square summable-on-iff-abs-summable-on-real*)

moreover from $\langle \text{hilbert-schmidt } C \rangle$

have $\langle (\lambda e. (\text{norm } (C *_V e))^2) \text{ abs-summable-on some-chilbert-basis} \rangle$

by (*metis is-onb-some-chilbert-basis summable-hilbert-schmidt-norm-square summable-on-iff-abs-summable-on-real*)

ultimately have $\langle (\lambda e. \text{norm } ((B^* \circ_{CL} W) *_V e) * \text{norm } (C *_V e)) \text{ abs-summable-on some-chilbert-basis} \rangle$

apply (*rule-tac abs-summable-product*)

by (*metis (no-types, lifting) power2-eq-square summable-on-cong*)+

then have $\langle (\lambda e. \text{cinner } e (abs-op A *_V e)) \text{ abs-summable-on some-chilbert-basis} \rangle$

proof (*rule Infinite-Sum.abs-summable-on-comparison-test*)

fix $e :: 'a$ **assume** $\langle e \in \text{some-chilbert-basis} \rangle$

have $\langle \text{norm } (e \cdot_C (abs-op A *_V e)) = \text{norm } (((B^* \circ_{CL} W) *_V e) \cdot_C (C *_V e)) \rangle$

by (*simp add: abs-op-A cinner-adj-left cinner-adj-right*)

also have $\langle \dots \leq \text{norm } ((B^* \circ_{CL} W) *_V e) * \text{norm } (C *_V e) \rangle$

by (*rule Cauchy-Schwarz-ineq2*)

also have $\langle \dots = \text{norm } (\text{norm } ((B^* \circ_{CL} W) *_V e) * \text{norm } (C *_V e)) \rangle$


```

    by simp
    finally show ⟨cmod (e ·C (abs-op A *V e)) ≤ norm (norm ((B* oCL W) *V e) * norm (C
*V e))⟩
    by -
  qed
  then show ⟨trace-class A⟩
  apply (rule trace-classI[rotated]) by simp
qed

```

instantiation *hilbert-schmidt* :: (*chilbert-space*, *chilbert-space*) *complex-vector* **begin**
instance

```

proof intro-classes
  fix a b c :: ⟨('a,'b) hilbert-schmidt⟩
  show ⟨a + b + c = a + (b + c)⟩
    apply transfer by auto
  show ⟨a + b = b + a⟩
    apply transfer by auto
  show ⟨0 + a = a⟩
    apply transfer by auto
  show ⟨- a + a = 0⟩
    apply transfer by auto
  show ⟨a - b = a + - b⟩
    apply transfer by auto
  show ⟨r *C (a + b) = r *C a + r *C b⟩ for r :: complex
    apply transfer
    using scaleC-add-right
    by auto
  show ⟨(r + r') *C a = r *C a + r' *C a⟩ for r r' :: complex
    apply transfer
    by (simp add: scaleC-add-left)
  show ⟨r *C r' *C a = (r * r') *C a⟩ for r r'
    apply transfer by auto
  show ⟨1 *C a = a⟩
    apply transfer by auto
  qed
end

```

instantiation *hilbert-schmidt* :: (*chilbert-space*, *chilbert-space*) *complex-inner* **begin**

lift-definition *cinner-hilbert-schmidt* :: ⟨('a,'b) *hilbert-schmidt* ⇒ ('a,'b) *hilbert-schmidt* ⇒ *complex*⟩ **is**

⟨λb c. trace (b* o_{CL} c)⟩ .

instance

```

proof intro-classes
  fix x y z :: ⟨('a,'b) hilbert-schmidt⟩
  show ⟨x ·C y = conj (y ·C x)⟩
  proof (transfer; unfold mem-Collect-eq)
    fix x y :: ⟨'a ⇒CL 'b⟩
    assume hs-xy: ⟨hilbert-schmidt x⟩ ⟨hilbert-schmidt y⟩

```

```

then have tc: ⟨trace-class ((y* oCL x)*)⟩ ⟨trace-class (y* oCL x)⟩
  by (auto intro!: hs-times-hs-trace-class)
have ⟨trace (x* oCL y) = trace ((y* oCL x)*)⟩
  by simp
also have ⟨... = cnj (trace (y* oCL x))⟩
  using tc trace-adj-prelim by blast
finally show ⟨trace (x* oCL y) = cnj (trace (y* oCL x))⟩
  by -
qed
show ⟨(x + y) •C z = x •C z + y •C z⟩
proof (transfer; unfold mem-Collect-eq)
  fix x y z :: ⟨'a ⇒CL 'b⟩
  assume [simp]: ⟨hilbert-schmidt x⟩ ⟨hilbert-schmidt y⟩ ⟨hilbert-schmidt z⟩
  have [simp]: ⟨trace-class ((x + y)* oCL z)⟩ ⟨trace-class (x* oCL z)⟩ ⟨trace-class (y* oCL z)⟩
    by (auto intro!: hs-times-hs-trace-class hilbert-schmidt-adj hilbert-schmidt-plus)
  then have [simp]: ⟨trace-class ((x* oCL z) + (y* oCL z))⟩
    by (simp add: adj-plus cblinfun-compose-add-left)
  show ⟨trace ((x + y)* oCL z) = trace (x* oCL z) + trace (y* oCL z)⟩
    by (simp add: trace-plus-prelim adj-plus cblinfun-compose-add-left hs-times-hs-trace-class)
qed
show ⟨r *C x •C y = cnj r * (x •C y)⟩ for r
  apply transfer
  by (simp add: trace-scaleC)
show ⟨0 ≤ x •C x⟩
  apply transfer
  by (simp add: positive-cblinfun-squareI trace-pos)
show ⟨(x •C x = 0) = (x = 0)⟩
proof (transfer; unfold mem-Collect-eq)
  fix x :: ⟨'a ⇒CL 'b⟩
  assume [simp]: ⟨hilbert-schmidt x⟩
  have ⟨trace (x* oCL x) = 0 ⟷ trace (abs-op (x* oCL x)) = 0⟩
    by (metis abs-op-def positive-cblinfun-squareI sqrt-op-unique)
  also have ⟨... ⟷ trace-norm (x* oCL x) = 0⟩
    by simp
  also have ⟨... ⟷ x* oCL x = 0⟩
    by (metis ⟨hilbert-schmidt x⟩ hilbert-schmidt-def trace-norm-0 trace-norm-nondegenerate)
  also have ⟨... ⟷ x = 0⟩
    using cblinfun-compose-zero-right op-square-nondegenerate by blast
  finally show ⟨trace (x* oCL x) = 0 ⟷ x = 0⟩
    by -
qed
show ⟨norm x = sqrt (cmod (x •C x))⟩
  apply transfer
  apply (auto simp: hilbert-schmidt-norm-def)
  by (metis Re-complex-of-real cmod-Re positive-cblinfun-squareI trace-norm-pos trace-pos)
qed
end

```

lemma hilbert-schmidt-norm-triangle-ineq:

— [1], Proposition 18.6 (e). We do not use their proof but get it as a simple corollary of the instantiation of *hilbert-schmidt* as a inner product space. The proof by Conway would probably allow us to weaken the sort of *'b* to *complex-inner*.

```

fixes a b :: ⟨'a::hilbert-space ⇒CL 'b::hilbert-space⟩
assumes ⟨hilbert-schmidt a⟩ ⟨hilbert-schmidt b⟩
shows ⟨hilbert-schmidt-norm (a + b) ≤ hilbert-schmidt-norm a + hilbert-schmidt-norm b⟩
proof –
define a' b' where ⟨a' = Abs-hilbert-schmidt a⟩ and ⟨b' = Abs-hilbert-schmidt b⟩
have [transfer-rule]: ⟨cr-hilbert-schmidt a a'⟩
by (simp add: Abs-hilbert-schmidt-inverse a'-def assms(1) cr-hilbert-schmidt-def)
have [transfer-rule]: ⟨cr-hilbert-schmidt b b'⟩
by (simp add: Abs-hilbert-schmidt-inverse assms(2) b'-def cr-hilbert-schmidt-def)
have ⟨norm (a' + b') ≤ norm a' + norm b'⟩
by (rule norm-triangle-ineq)
then show ?thesis
apply transfer
by –
qed

```

lift-definition *adj-hs* :: ⟨('a::hilbert-space, 'b::hilbert-space) hilbert-schmidt ⇒ ('b, 'a) hilbert-schmidt⟩
is *adj*
by *auto*

lemma *adj-hs-plus*: ⟨*adj-hs* (x + y) = *adj-hs* x + *adj-hs* y⟩
apply *transfer*
by (*simp add: adj-plus*)

lemma *adj-hs-minus*: ⟨*adj-hs* (x - y) = *adj-hs* x - *adj-hs* y⟩
apply *transfer*
by (*simp add: adj-minus*)

lemma *norm-adj-hs[simp]*: ⟨*norm* (*adj-hs* x) = *norm* x⟩
apply *transfer*
by *simp*

lemma *hilbert-schmidt-norm-geq-norm*:

— [1], Proposition 18.6 (c)
assumes ⟨*hilbert-schmidt* a⟩
shows ⟨*norm* a ≤ *hilbert-schmidt-norm* a⟩

```

proof –
have ⟨norm (a x) ≤ hilbert-schmidt-norm a⟩ if ⟨norm x = 1⟩ for x
proof –
obtain B where ⟨x ∈ B⟩ and ⟨is-onb B⟩
using orthonormal-basis-exists[of ⟨{x}⟩] ⟨norm x = 1⟩
by force
have ⟨(norm (a x))2 = (∑∞x∈{x}. (norm (a x))2)⟩
by simp
also have ⟨... ≤ (∑∞x∈B. (norm (a x))2)⟩
apply (rule infsum-mono-neutral)

```

```

    by (auto intro!: summable-hilbert-schmidt-norm-square ⟨is-onb B⟩ assms ⟨x ∈ B⟩)
  also have ⟨... = (hilbert-schmidt-norm a)2⟩
    using infsum-hilbert-schmidt-norm-square[OF ⟨is-onb B⟩ assms]
  by -
  finally show ?thesis
    by force
qed
then show ?thesis
  by (auto intro!: norm-cblinfun-bound-unit)
qed

```

12.4 Trace-norm and trace-class, continued

lemma *trace-class-comp-left*: $\langle \text{trace-class } (a \circ_{CL} b) \rangle$ **if** $\langle \text{trace-class } a \rangle$ **for** $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$

— [1], Theorem 18.11 (a)

proof —

```

  from ⟨trace-class a⟩
  obtain C :: ⟨'a ⇒CL 'b⟩ and B where ⟨hilbert-schmidt C⟩ and ⟨hilbert-schmidt B⟩ and aCB:
    ⟨a = C oCL B⟩
  by (auto simp: trace-class-iff-hs-times-hs)
  from ⟨hilbert-schmidt B⟩ have ⟨hilbert-schmidt (B oCL b)⟩
  by (simp add: hilbert-schmidt-comp-left)
  with ⟨hilbert-schmidt C⟩ have ⟨trace-class (C oCL (B oCL b))⟩
  using hs-times-hs-trace-class by blast
  then show ?thesis
    by (simp flip: aCB cblinfun-compose-assoc)
qed

```

lemma *trace-class-comp-right*: $\langle \text{trace-class } (a \circ_{CL} b) \rangle$ **if** $\langle \text{trace-class } b \rangle$ **for** $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$

— [1], Theorem 18.11 (a)

proof —

```

  from ⟨trace-class b⟩
  obtain C :: ⟨'c ⇒CL 'a⟩ and B where ⟨hilbert-schmidt C⟩ and ⟨hilbert-schmidt B⟩ and aCB:
    ⟨b = C oCL B⟩
  by (auto simp: trace-class-iff-hs-times-hs)
  from ⟨hilbert-schmidt C⟩ have ⟨hilbert-schmidt (a oCL C)⟩
  by (simp add: hilbert-schmidt-comp-right)
  with ⟨hilbert-schmidt B⟩ have ⟨trace-class ((a oCL C) oCL B)⟩
  using hs-times-hs-trace-class by blast
  then show ?thesis
    by (simp flip: aCB add: cblinfun-compose-assoc)
qed

```

lemma

fixes $B :: \langle 'a::\text{chilbert-space set} \rangle$ **and** $A :: \langle 'a \Rightarrow_{CL} 'a \rangle$ **and** $b :: \langle 'b::\text{chilbert-space} \Rightarrow_{CL} 'c::\text{chilbert-space} \rangle$ **and** $c :: \langle 'c \Rightarrow_{CL} 'b \rangle$
shows *trace-alt-def*:

— [1], Proposition 18.9
 $\langle is\text{-}onb\ B \implies trace\ A = (if\ trace\text{-}class\ A\ then\ (\sum_{\infty e \in B}. e \cdot_C (A *_{\mathcal{V}} e))\ else\ 0) \rangle$
and *trace-hs-times-hs*: $\langle hilbert\text{-}schmidt\ c \implies hilbert\text{-}schmidt\ b \implies trace\ (c\ o_{CL}\ b) =$
 $((of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) + b)))^2 - (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) -$
 $b)))^2 -$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ (((c*) + i *_{\mathcal{C}}\ b))))^2 +$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ (((c*) - i *_{\mathcal{C}}\ b))))^2 / 4 \rangle$

proof —
have *ecbe-has-sum*: $\langle ((\lambda e. e \cdot_C ((c\ o_{CL}\ b) *_{\mathcal{V}} e))\ has\text{-}sum$
 $((of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) + b)))^2 - (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) -$
 $b)))^2 -$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) + i *_{\mathcal{C}}\ b)))^2 +$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) - i *_{\mathcal{C}}\ b)))^2 / 4) B \rangle$
if $\langle is\text{-}onb\ B \rangle$ **and** $\langle hilbert\text{-}schmidt\ c \rangle$ **and** $\langle hilbert\text{-}schmidt\ b \rangle$ **for** $B :: \langle 'y :: hilbert\text{-}space\ set \rangle$
and $c :: \langle 'x :: hilbert\text{-}space \implies_{CL}\ 'y \rangle$ **and** b
apply (*simp flip: cinner-adj-left*[*of c*])
apply (*subst cdot-norm*)
using **that** **by** (*auto simp add: field-class.field-divide-inverse infsum-cmult-left'*
simp del: Num.inverse-eq-divide-numeral
simp flip: cblinfun.add-left cblinfun.diff-left cblinfun.scaleC-left of-real-power
intro!: has-sum-cmult-left has-sum-cmult-right has-sum-add has-sum-diff has-sum-of-real
has-sum-hilbert-schmidt-norm-square hilbert-schmidt-plus hilbert-schmidt-minus hilbert-schmidt-scaleC)

then **have** *ecbe-infsum*: $\langle (\sum_{\infty e \in B}. e \cdot_C ((c\ o_{CL}\ b) *_{\mathcal{V}} e)) =$
 $((of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) + b)))^2 - (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) -$
 $b)))^2 -$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) + i *_{\mathcal{C}}\ b)))^2 +$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) - i *_{\mathcal{C}}\ b)))^2 / 4) \rangle$
if $\langle is\text{-}onb\ B \rangle$ **and** $\langle hilbert\text{-}schmidt\ c \rangle$ **and** $\langle hilbert\text{-}schmidt\ b \rangle$ **for** $B :: \langle 'y :: hilbert\text{-}space\ set \rangle$
and $c :: \langle 'x :: hilbert\text{-}space \implies_{CL}\ 'y \rangle$ **and** b
using *infsumI* **that**(1) **that**(2) **that**(3) **by** *blast*

show $\langle trace\ (c\ o_{CL}\ b) =$
 $((of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) + b)))^2 - (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) -$
 $b)))^2 -$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) + i *_{\mathcal{C}}\ b)))^2 +$
 $i * (of\text{-}real\ (hilbert\text{-}schmidt\text{-}norm\ ((c*) - i *_{\mathcal{C}}\ b)))^2 / 4 \rangle$
if $\langle hilbert\text{-}schmidt\ c \rangle$ **and** $\langle hilbert\text{-}schmidt\ b \rangle$

proof —
from **that** **have** *tc-cb*[*simp*]: $\langle trace\text{-}class\ (c\ o_{CL}\ b) \rangle$
by (*rule hs-times-hs-trace-class*)
show *?thesis*
using *ecbe-infsum*[*OF is-onb-some-hilbert-basis hilbert-schmidt c hilbert-schmidt b*]
apply (*simp only: trace-def*)
by *simp*

qed

show $\langle trace\ A = (if\ trace\text{-}class\ A\ then\ (\sum_{\infty e \in B}. e \cdot_C (A *_{\mathcal{V}} e))\ else\ 0) \rangle$ **if** $\langle is\text{-}onb\ B \rangle$
proof (*cases* $\langle trace\text{-}class\ A \rangle$)

```

case True
with that
obtain b c :: ⟨'a ⇒CL 'a⟩ where hs-b: ⟨hilbert-schmidt b⟩ and hs-c: ⟨hilbert-schmidt c⟩ and
Acb: ⟨A = c oCL b⟩
  by (metis trace-class-iff-hs-times-hs)
  have [simp]: ⟨trace-class (c oCL b)⟩
  using Acb True by auto

show ⟨trace A = (if trace-class A then (∑∞ e∈B. e •C (A *V e)) else 0)⟩
  using ecbe-infsum[OF is-onb-some-chilbert-basis hs-c hs-b]
  using ecbe-infsum[OF ⟨is-onb B⟩ hs-c hs-b]
  by (simp only: Acb trace-def)
next
case False
then show ?thesis
  by (simp add: trace-def)
qed
qed

```

```

lemma trace-ket-sum:
  fixes A :: ⟨'a ell2 ⇒CL 'a ell2⟩
  assumes ⟨trace-class A⟩
  shows ⟨trace A = (∑∞ e. ket e •C (A *V ket e))⟩
  apply (subst infsum-reindex[where h=ket, unfolded o-def, symmetric])
  by (auto simp: trace-class A trace-alt-def[OF is-onb-ket] is-onb-ket)

```

```

lemma trace-one-dim[simp]: ⟨trace A = one-dim-iso A⟩ for A :: ⟨'a::one-dim ⇒CL 'a⟩
proof –
  have onb: ⟨is-onb {1 :: 'a}⟩
    by auto
  have ⟨trace A = 1 •C (A *V 1)⟩
    apply (subst trace-alt-def)
    apply (fact onb)
    by simp
  also have ⟨... = one-dim-iso A⟩
    by (simp add: cinner-cblinfun-def one-dim-iso-def)
  finally show ?thesis
    by –
qed

```

```

lemma trace-has-sum:
  assumes ⟨is-onb E⟩
  assumes ⟨trace-class t⟩
  shows ⟨((λe. e •C (t *V e)) has-sum trace t) E⟩
  using assms(1) assms(2) trace-alt-def trace-exists by fastforce

```

```

lemma trace-sandwich-isometry[simp]: ⟨trace (sandwich U A) = trace A⟩ if ⟨isometry U⟩

```

```

proof (cases ⟨trace-class A⟩)
  case True
  note True[simp]
  have ⟨is-ortho-set ((*V) U ‘ some-chilbert-basis)⟩
    unfolding is-ortho-set-def
    apply auto
  apply (metis (no-types, opaque-lifting) cblinfun-apply-cblinfun-compose cblinfun-id-cblinfun-apply
cinner-adj-right is-ortho-set-def is-ortho-set-some-chilbert-basis isometry-def that)
  by (metis is-normal-some-chilbert-basis isometry-preserves-norm norm-zero that zero-neq-one)
  moreover have ⟨x ∈ ((*V) U ‘ some-chilbert-basis) ⇒ norm x = 1⟩ for x
    using is-normal-some-chilbert-basis isometry-preserves-norm that by fastforce
  ultimately obtain B where BU: ⟨B ⊇ U ‘ some-chilbert-basis⟩ and ⟨is-onb B⟩
  apply atomize-elim
  by (rule orthonormal-basis-exists)

have xUy: ⟨x ·C U y = 0⟩ if xBU: ⟨x ∈ B – U ‘ some-chilbert-basis⟩ for x y
proof –
  from that ⟨is-onb B⟩ ⟨isometry U⟩
  have ⟨x ·C z = 0⟩ if ⟨z ∈ U ‘ some-chilbert-basis⟩ for z
    using that by (metis BU Diff-iff-in-mono is-onb-def is-ortho-set-def)
  then have ⟨x ∈ orthogonal-complement (closure (cspan (U ‘ some-chilbert-basis)))⟩
  by (metis orthogonal-complementI orthogonal-complement-of-closure orthogonal-complement-of-cspan)
  then have ⟨x ∈ space-as-set (– ccspan (U ‘ some-chilbert-basis))⟩
    by (simp add: ccspan.rep-eq uminus-ccsubspace.rep-eq)
  then have ⟨x ∈ space-as-set (– (U *S top))⟩
    by (metis cblinfun-image-ccspan ccspan-some-chilbert-basis)
  moreover have ⟨U y ∈ space-as-set (U *S top)⟩
    by simp
  ultimately show ?thesis
    apply (transfer fixing: x y)
    using orthogonal-complement-orthoI by blast
qed

have [simp]: ⟨trace-class (sandwich U A)⟩
  by (simp add: sandwich.rep-eq trace-class-comp-left trace-class-comp-right)
have ⟨trace (sandwich U A) = (∑∞ e∈B. e ·C ((sandwich U *V A) *V e))⟩
  using ⟨is-onb B⟩ trace-alt-def by fastforce
also have ⟨... = (∑∞ e∈U ‘ some-chilbert-basis. e ·C ((sandwich U *V A) *V e))⟩
  apply (rule infsum-cong-neutral)
  using BU xUy by (auto simp: sandwich-apply)
also have ⟨... = (∑∞ e∈some-chilbert-basis. U e ·C ((sandwich U *V A) *V U e))⟩
  apply (subst infsum-reindex)
  apply (metis cblinfun-apply-cblinfun-compose cblinfun-id-cblinfun-apply inj-on-inverseI isom-
etry-def that)
  by (auto simp: o-def)
also have ⟨... = (∑∞ e∈some-chilbert-basis. e ·C A e)⟩
  apply (rule infsum-cong)
  apply (simp add: sandwich-apply flip: cinner-adj-right)
  by (metis cblinfun-apply-cblinfun-compose cblinfun-id-cblinfun-apply isometry-def that)

```

also have $\langle \dots = \text{trace } A \rangle$
by (*simp add: trace-def*)
finally show *?thesis*
by –
next
case *False*
note *False[simp]*
then have [*simp*]: $\langle \neg \text{trace-class } (\text{sandwich } U A) \rangle$
by (*smt (verit, ccfu-SIG) cblinfun-assoc-left(1) cblinfun-compose-id-left cblinfun-compose-id-right isometryD sandwich.rep-eq that trace-class-comp-left trace-class-comp-right*)
show *?thesis*
by (*simp add: trace-def*)
qed

lemma *circularity-of-trace*:

– [1], Theorem 18.11 (e)
fixes $a :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$ **and** $b :: \langle 'b \Rightarrow_{CL} 'a \rangle$
– The proof from [1] only work for square operators, we generalize it
assumes $\langle \text{trace-class } a \rangle$
– Actually, $\text{trace-class } (a \circ_{CL} b) \wedge \text{trace-class } (b \circ_{CL} a)$ is sufficient here, see [3] but the proof is more involved. Only $\text{trace-class } (a \circ_{CL} b)$ is not sufficient, see [4].
shows $\langle \text{trace } (a \circ_{CL} b) = \text{trace } (b \circ_{CL} a) \rangle$
proof –

define $a' b' :: \langle ('a \times 'b) \Rightarrow_{CL} ('a \times 'b) \rangle$
where $\langle a' = \text{cblinfun-right } o_{CL} a \circ_{CL} \text{cblinfun-left} * \rangle$
and $\langle b' = \text{cblinfun-left } o_{CL} b \circ_{CL} \text{cblinfun-right} * \rangle$

have $\langle \text{trace-class } a' \rangle$
by (*simp add: a'-def assms trace-class-comp-left trace-class-comp-right*)

have *circ'*: $\langle \text{trace } (a' \circ_{CL} b') = \text{trace } (b' \circ_{CL} a') \rangle$

proof –
from $\langle \text{trace-class } a' \rangle$
obtain $B C :: \langle ('a \times 'b) \Rightarrow_{CL} ('a \times 'b) \rangle$ **where** $\langle \text{hilbert-schmidt } B \rangle$ **and** $\langle \text{hilbert-schmidt } C \rangle$
and aCB : $\langle a' = C * \circ_{CL} B \rangle$
by (*metis abs-op-pos adj-cblinfun-compose double-adj hilbert-schmidt-comp-left hilbert-schmidt-comp-right polar-decomposition-correct polar-decomposition-correct' positive-selfadjointI[unfolded selfadjoint-def] trace-class-iff-hs-times-hs*)
have *hs-iB*: $\langle \text{hilbert-schmidt } (i *_C B) \rangle$
by (*metis Abs-hilbert-schmidt-inverse Rep-hilbert-schmidt hilbert-schmidt B mem-Collect-eq scaleC-hilbert-schmidt.rep-eq*)
have $*$: $\langle \text{Re } (\text{trace } (C * \circ_{CL} B)) = \text{Re } (\text{trace } (C \circ_{CL} B *)) \rangle$ **if** $\langle \text{hilbert-schmidt } B \rangle$ $\langle \text{hilbert-schmidt } C \rangle$ **for** $B C :: \langle ('a \times 'b) \Rightarrow_{CL} ('a \times 'b) \rangle$
proof –
from *that*
obtain $B' C'$ **where** $\langle B = \text{Rep-hilbert-schmidt } B' \rangle$ **and** $\langle C = \text{Rep-hilbert-schmidt } C' \rangle$
by (*meson Rep-hilbert-schmidt-cases mem-Collect-eq*)


```

then have [transfer-rule]: ⟨cr-hilbert-schmidt B B'⟩ ⟨cr-hilbert-schmidt C C'⟩
  by (simp-all add: cr-hilbert-schmidt-def)

have ⟨Re (trace (C* oCL B)) = Re (C' ·C B')⟩
  apply transfer by simp
also have ⟨... = (1/4) * ((norm (C' + B'))2 - (norm (C' - B'))2)⟩
  by (simp add: cdot-norm)
also have ⟨... = (1/4) * ((norm (adj-hs C' + adj-hs B'))2 - (norm (adj-hs C' - adj-hs
B'))2)⟩
  by (simp add: flip: adj-hs-plus adj-hs-minus)
also have ⟨... = Re (adj-hs C' ·C adj-hs B')⟩
  by (simp add: cdot-norm)
also have ⟨... = Re (trace (C oCL B*))⟩
  apply transfer by simp
finally show ?thesis
  by -
qed
have **: ⟨trace (C* oCL B) = cnj (trace (C oCL B*))⟩ if ⟨hilbert-schmidt B⟩ ⟨hilbert-schmidt
C⟩ for B C :: ⟨('a × 'b) ⇒CL ('a × 'b)⟩
  using *[OF ⟨hilbert-schmidt B⟩ ⟨hilbert-schmidt C⟩]
  using *[OF hilbert-schmidt-scaleC[of - i, OF ⟨hilbert-schmidt B⟩] ⟨hilbert-schmidt C⟩]
  apply (auto simp: trace-scaleC cblinfun-compose-uminus-right trace-uminus)
  by (smt (verit, best) cnj.code complex.collapse)

have ⟨trace (b' oCL a') = trace ((b' oCL C*) oCL B)⟩
  by (simp add: aCB cblinfun-assoc-left(1))
also from ** ⟨hilbert-schmidt B⟩ ⟨hilbert-schmidt C⟩ have ⟨... = cnj (trace ((C oCL b'*)
oCL B*))⟩
  by (metis adj-cblinfun-compose double-adj hilbert-schmidt-comp-left)
also have ⟨... = cnj (trace (C oCL (B oCL b')*))⟩
  by (simp add: cblinfun-assoc-left(1))
also from ** ⟨hilbert-schmidt B⟩ ⟨hilbert-schmidt C⟩ have ⟨... = trace (C* oCL (B oCL
b'))⟩
  by (simp add: hilbert-schmidt-comp-left)
also have ⟨... = trace (a' oCL b')⟩
  by (simp add: aCB cblinfun-compose-assoc)
finally show ?thesis
  by simp
qed

have ⟨trace (a oCL b) = trace (sandwich cblinfun-right (a oCL b) :: ('a × 'b) ⇒CL ('a × 'b))⟩
  by simp
also have ⟨... = trace (sandwich cblinfun-right (a oCL (cblinfun-left* oCL (cblinfun-left ::
⇒CL ('a × 'b))) oCL b) :: ('a × 'b) ⇒CL ('a × 'b))⟩
  by simp
also have ⟨... = trace (a' oCL b')⟩
  by (simp only: a'-def b'-def sandwich-apply cblinfun-compose-assoc)
also have ⟨... = trace (b' oCL a')⟩
  by (rule circ')

```

also have $\langle \dots = \text{trace} (\text{sandwich cblinfun-left} (b \text{ } o_{CL} (\text{cblinfun-right} * \text{ } o_{CL} (\text{cblinfun-right} :: \Rightarrow_{CL} ('a \times 'b))) \text{ } o_{CL} a) :: ('a \times 'b) \Rightarrow_{CL} ('a \times 'b)) \rangle$
by (*simp only: a'-def b'-def sandwich-apply cblinfun-compose-assoc*)
also have $\langle \dots = \text{trace} (\text{sandwich cblinfun-left} (b \text{ } o_{CL} a) :: ('a \times 'b) \Rightarrow_{CL} ('a \times 'b)) \rangle$
by *simp*
also have $\langle \dots = \text{trace} (b \text{ } o_{CL} a) \rangle$
by *simp*
finally show $\langle \text{trace} (a \text{ } o_{CL} b) = \text{trace} (b \text{ } o_{CL} a) \rangle$
by –
qed

lemma trace-butterfly-comp: $\langle \text{trace} (\text{butterfly } x \text{ } y \text{ } o_{CL} a) = y \cdot_C (a *_{\mathcal{V}} x) \rangle$
proof –
have $\langle \text{trace} (\text{butterfly } x \text{ } y \text{ } o_{CL} a) = \text{trace} (\text{vector-to-cblinfun } y * \text{ } o_{CL} (a \text{ } o_{CL} (\text{vector-to-cblinfun } x :: \text{complex} \Rightarrow_{CL} -))) \rangle$
unfolding *butterfly-def*
by (*metis cblinfun-compose-assoc circularity-of-trace trace-class-finite-dim*)
also have $\langle \dots = y \cdot_C (a *_{\mathcal{V}} x) \rangle$
by (*simp add: one-dim-iso-cblinfun-comp*)
finally show *?thesis*
by –
qed

lemma trace-butterfly: $\langle \text{trace} (\text{butterfly } x \text{ } y) = y \cdot_C x \rangle$
using *trace-butterfly-comp* [**where** *a=id-cblinfun*] **by** *auto*

lemma trace-butterfly-comp': $\langle \text{trace} (a \text{ } o_{CL} \text{ butterfly } x \text{ } y) = y \cdot_C (a *_{\mathcal{V}} x) \rangle$
by (*simp add: cblinfun-comp-butterfly trace-butterfly*)

lemma trace-norm-adj[*simp*]: $\langle \text{trace-norm} (a *) = \text{trace-norm } a \rangle$
– [1], Theorem 18.11 (f)
proof –
have $\langle \text{of-real} (\text{trace-norm} (a *)) = \text{trace} (\text{sandwich} (\text{polar-decomposition } a) *_{\mathcal{V}} \text{abs-op } a) \rangle$
by (*metis abs-op-adj trace-abs-op*)
also have $\langle \dots = \text{trace} ((\text{polar-decomposition } a) * \text{ } o_{CL} (\text{polar-decomposition } a) \text{ } o_{CL} \text{abs-op } a) \rangle$
by (*metis (no-types, lifting) abs-op-adj cblinfun-compose-assoc circularity-of-trace double-adj polar-decomposition-correct polar-decomposition-correct' sandwich.rep-eq trace-class-abs-op trace-def*)
also have $\langle \dots = \text{trace} (\text{abs-op } a) \rangle$
by (*simp add: cblinfun-compose-assoc polar-decomposition-correct polar-decomposition-correct'*)
also have $\langle \dots = \text{of-real} (\text{trace-norm } a) \rangle$
by *simp*
finally show *?thesis*
by *simp*
qed

lemma trace-class-adj[*simp*]: $\langle \text{trace-class} (a *) \rangle$ **if** $\langle \text{trace-class } a \rangle$
proof (*rule ccontr*)

```

assume asm:  $\langle \neg \text{trace-class } (a^*) \rangle$ 
then have  $\langle \text{trace-norm } (a^*) = 0 \rangle$ 
  by (simp add: trace-norm-def)
then have  $\langle \text{trace-norm } a = 0 \rangle$ 
  by (metis trace-norm-adj)
then have  $\langle a = 0 \rangle$ 
  using that trace-norm-nondegenerate by blast
then have  $\langle \text{trace-class } (a^*) \rangle$ 
  by simp
with asm show False
  by simp
qed

```

lift-definition *adj-tc* :: $\langle ('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{ trace-class} \Rightarrow ('b, 'a) \text{ trace-class} \rangle$
is *adj*
by *simp*

lift-definition *selfadjoint-tc* :: $\langle ('a::\text{chilbert-space}, 'a) \text{ trace-class} \Rightarrow \text{bool} \rangle$ **is** *selfadjoint*.

lemma *selfadjoint-tc-def'*: $\langle \text{selfadjoint-tc } a \iff \text{adj-tc } a = a \rangle$
apply *transfer*
using *selfadjoint-def by blast*

lemma *trace-class-finite-dim*^[*simp*]: $\langle \text{trace-class } A \rangle$ **for** $A :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\{\text{cfinite-dim, chilbert-space}\} \rangle$
by (*metis double-adj trace-class-adj trace-class-finite-dim*)

lemma *trace-class-plus*^[*simp*]:

fixes $t\ u :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$

assumes $\langle \text{trace-class } t \rangle$ **and** $\langle \text{trace-class } u \rangle$

shows $\langle \text{trace-class } (t + u) \rangle$

— [1], Theorem 18.11 (a). However, we use a completely different proof that does not need the fact that trace class operators can be diagonalized with countably many diagonal elements.

proof —

define $II :: \langle 'a \Rightarrow_{CL} ('a \times 'a) \rangle$ **where** $\langle II = \text{cblinfun-left} + \text{cblinfun-right} \rangle$

define $JJ :: \langle ('b \times 'b) \Rightarrow_{CL} 'b \rangle$ **where** $\langle JJ = \text{cblinfun-left*} + \text{cblinfun-right*} \rangle$

define $tu :: \langle ('a \times 'a) \Rightarrow_{CL} ('b \times 'b) \rangle$ **where** $\langle tu = (\text{cblinfun-left } o_{CL} t\ o_{CL} \text{cblinfun-left*}) + (\text{cblinfun-right } o_{CL} u\ o_{CL} \text{cblinfun-right*}) \rangle$

have *t-plus-u*: $\langle t + u = JJ\ o_{CL} tu\ o_{CL} II \rangle$

apply (*simp add: II-def JJ-def tu-def cblinfun-compose-add-left cblinfun-compose-add-right cblinfun-compose-assoc*)

by (*simp flip: cblinfun-compose-assoc*)

have $\langle \text{trace-class } tu \rangle$

proof (*rule trace-classI*)

define $BL\ BR\ B :: \langle ('a \times 'a) \text{ set} \rangle$ **where** $\langle BL = \text{some-chilbert-basis} \times \{0\} \rangle$

and $\langle BR = \{0\} \times \text{some-chilbert-basis} \rangle$

and $\langle B = BL \cup BR \rangle$

have $\langle BL \cap BR = \{ \} \rangle$

using *is-ortho-set-some-chilbert-basis*

by (*auto simp: BL-def BR-def is-ortho-set-def*)

show $\langle is-onb\ B \rangle$
by (*simp add: BL-def BR-def B-def is-onb-prod*)
have $abs-tu: \langle abs-op\ tu = (cblinfun-left\ o_{CL}\ abs-op\ t\ o_{CL}\ cblinfun-left*) + (cblinfun-right\ o_{CL}\ abs-op\ u\ o_{CL}\ cblinfun-right*) \rangle$
using [*show-consts*]
proof –
have $\langle ((cblinfun-left\ o_{CL}\ abs-op\ t\ o_{CL}\ cblinfun-left*) + (cblinfun-right\ o_{CL}\ abs-op\ u\ o_{CL}\ cblinfun-right*)) *_{o_{CL}} ((cblinfun-left\ o_{CL}\ abs-op\ t\ o_{CL}\ cblinfun-left*) + (cblinfun-right\ o_{CL}\ abs-op\ u\ o_{CL}\ cblinfun-right*)) \rangle$
 $= tu *_{o_{CL}} tu \rangle$
proof –
have $tt[THEN\ simp-a-oCL-b,\ simp]: \langle (abs-op\ t) *_{o_{CL}} abs-op\ t = t *_{o_{CL}} t \rangle$
by (*simp add: abs-op-def positive-cblinfun-squareI positive-selfadjointI[unfolded selfadjoint-def]*)
have $uu[THEN\ simp-a-oCL-b,\ simp]: \langle (abs-op\ u) *_{o_{CL}} abs-op\ u = u *_{o_{CL}} u \rangle$
by (*simp add: abs-op-def positive-cblinfun-squareI positive-selfadjointI[unfolded selfadjoint-def]*)
note *isometryD[THEN simp-a-oCL-b, simp]*
note *cblinfun-right-left-ortho[THEN simp-a-oCL-b, simp]*
note *cblinfun-left-right-ortho[THEN simp-a-oCL-b, simp]*
show *?thesis*
using $tt[of\ \langle cblinfun-left* :: ('a \times 'a) \Rightarrow_{CL}\ 'a \rangle]\ uu[of\ \langle cblinfun-right* :: ('a \times 'a) \Rightarrow_{CL}\ 'a \rangle]$
by (*simp add: tu-def cblinfun-compose-add-right cblinfun-compose-add-left adj-plus cblinfun-compose-assoc*)
qed
moreover **have** $\langle (cblinfun-left\ o_{CL}\ abs-op\ t\ o_{CL}\ cblinfun-left*) + (cblinfun-right\ o_{CL}\ abs-op\ u\ o_{CL}\ cblinfun-right*) \geq 0 \rangle$
apply (*rule positive-cblinfunI*)
by (*auto simp: cblinfun.add-left cinner-pos-if-pos*)
ultimately **show** *?thesis*
by (*rule abs-opI[symmetric]*)
qed
from *assms(1)*
have $\langle (\lambda x. x \cdot_C (abs-op\ t *_{V}\ x))\ abs-summable-on\ some-chilbert-basis \rangle$
by (*metis is-onb-some-chilbert-basis summable-on-iff-abs-summable-on-complex trace-class-abs-op trace-exists*)
then **have** *sum-BL: $\langle (\lambda x. x \cdot_C (abs-op\ tu *_{V}\ x))\ abs-summable-on\ BL \rangle$*
apply (*subst asm-rl[of $\langle BL = (\lambda x. (x, 0))\ 'some-chilbert-basis \rangle$]*)
by (*auto simp: BL-def summable-on-reindex inj-on-def o-def abs-tu cblinfun.add-left*)
from *assms(2)*
have $\langle (\lambda x. x \cdot_C (abs-op\ u *_{V}\ x))\ abs-summable-on\ some-chilbert-basis \rangle$
by (*metis is-onb-some-chilbert-basis summable-on-iff-abs-summable-on-complex trace-class-abs-op trace-exists*)
then **have** *sum-BR: $\langle (\lambda x. x \cdot_C (abs-op\ tu *_{V}\ x))\ abs-summable-on\ BR \rangle$*
apply (*subst asm-rl[of $\langle BR = (\lambda x. (0, x))\ 'some-chilbert-basis \rangle$]*)
by (*auto simp: BR-def summable-on-reindex inj-on-def o-def abs-tu cblinfun.add-left*)
from *sum-BL sum-BR*

```

    show ⟨(λx. x ·C (abs-op tu *V x)) abs-summable-on B⟩
      using ⟨BL ∩ BR = {}⟩
      by (auto intro!: summable-on-Un-disjoint simp: B-def)
  qed
  with t-plus-u
  show ⟨trace-class (t + u)⟩
    by (simp add: trace-class-comp-left trace-class-comp-right)
  qed

lemma trace-class-minus[simp]: ⟨trace-class t ⟹ trace-class u ⟹ trace-class (t - u)⟩
  for t u :: ⟨'a::hilbert-space ⟹CL 'b::hilbert-space⟩
  by (metis trace-class-plus trace-class-uminus uminus-add-conv-diff)

lemma trace-plus:
  assumes ⟨trace-class a⟩ ⟨trace-class b⟩
  shows ⟨trace (a + b) = trace a + trace b⟩
  by (simp add: assms(1) assms(2) trace-plus-prelim)
hide-fact trace-plus-prelim

lemma trace-class-sum:
  fixes a :: ⟨'a ⟹ 'b::hilbert-space ⟹CL 'c::hilbert-space⟩
  assumes ⟨∧i. i∈I ⟹ trace-class (a i)⟩
  shows ⟨trace-class (∑ i∈I. a i)⟩
  using assms apply (induction I rule:infinite-finite-induct)
  by auto

lemma
  assumes ⟨∧i. i∈I ⟹ trace-class (a i)⟩
  shows trace-sum: ⟨trace (∑ i∈I. a i) = (∑ i∈I. trace (a i))⟩
  using assms apply (induction I rule:infinite-finite-induct)
  by (auto simp: trace-plus trace-class-sum)

lemma cmod-trace-times: ⟨cmod (trace (a oCL b)) ≤ norm a * trace-norm b⟩ if ⟨trace-class b⟩
  for b :: ⟨'a::hilbert-space ⟹CL 'b::hilbert-space⟩
  — [1], Theorem 18.11 (e)
proof —
  define W where ⟨W = polar-decomposition b⟩

  have ⟨norm W ≤ 1⟩
    by (metis W-def norm-partial-isometry norm-zero order-refl polar-decomposition-partial-isometry
      zero-less-one-class.zero-le-one)
  have hs1: ⟨hilbert-schmidt (sqrt-op (abs-op b))⟩
    using that trace-class-iff-sqrt-hs by blast
  then have hs2: ⟨hilbert-schmidt (sqrt-op (abs-op b) oCL W* oCL a*)⟩
    by (simp add: hilbert-schmidt-comp-left)

  from ⟨trace-class b⟩
  have ⟨trace-class (a oCL b)⟩
    using trace-class-comp-right by blast

```

then have $sum1$: $\langle (\lambda e. e \cdot_C ((a \ o_{CL} \ b) *_{V} e)) \text{ abs-summable-on some-chilbert-basis} \rangle$
by (*metis is-onb-some-chilbert-basis summable-on-iff-abs-summable-on-complex trace-exists*)

have $sum5$: $\langle (\lambda x. (norm (sqrt-op (abs-op b) *_{V} x))^2) \text{ summable-on some-chilbert-basis} \rangle$
using *summable-hilbert-schmidt-norm-square[OF is-onb-some-chilbert-basis hs1]*
by (*simp add: power2-eq-square*)

have $sum4$: $\langle (\lambda x. (norm ((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} x))^2) \text{ summable-on some-chilbert-basis} \rangle$
using *summable-hilbert-schmidt-norm-square[OF is-onb-some-chilbert-basis hs2]*
by (*simp add: power2-eq-square*)

have $sum3$: $\langle (\lambda e. norm ((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} e) * norm (sqrt-op (abs-op b) *_{V} e)) \text{ summable-on some-chilbert-basis} \rangle$
apply (*rule abs-summable-summable*)
apply (*rule abs-summable-product*)
by (*intro sum4 sum5 summable-on-iff-abs-summable-on-real[THEN iffD1]*) $+$

have $sum2$: $\langle (\lambda e. ((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} e) \cdot_C (sqrt-op (abs-op b) *_{V} e)) \text{ abs-summable-on some-chilbert-basis} \rangle$
using $sum3$ [*THEN summable-on-iff-abs-summable-on-real[THEN iffD1]*]
apply (*rule abs-summable-on-comparison-test*)
by (*simp add: complex-inner-class.Cauchy-Schwarz-ineq2*)

from $\langle \text{trace-class } b \rangle$
have $\langle cmod (\text{trace } (a \ o_{CL} \ b)) = cmod (\sum_{\infty} e \in \text{some-chilbert-basis}. e \cdot_C ((a \ o_{CL} \ b) *_{V} e)) \rangle$
by (*simp add: trace-class-comp-right trace-def*)
also have $\langle \dots \leq (\sum_{\infty} e \in \text{some-chilbert-basis}. cmod (e \cdot_C ((a \ o_{CL} \ b) *_{V} e))) \rangle$
using $sum1$ **by** (*rule norm-infsum-bound*)
also have $\langle \dots = (\sum_{\infty} e \in \text{some-chilbert-basis}. cmod (((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} e) \cdot_C (sqrt-op (abs-op b) *_{V} e))) \rangle$
apply (*simp add: positive-selfadjointI[unfolded selfadjoint-def] flip: cinner-adj-right cblinfun-apply-cblinfun-compose*)
by (*metis (full-types) W-def abs-op-def cblinfun-compose-assoc polar-decomposition-correct sqrt-op-pos sqrt-op-square*)
also have $\langle \dots \leq (\sum_{\infty} e \in \text{some-chilbert-basis}. norm ((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} e) * norm (sqrt-op (abs-op b) *_{V} e)) \rangle$
using $sum2 \ sum3$ **apply** (*rule infsum-mono*)
using *complex-inner-class.Cauchy-Schwarz-ineq2* **by** *blast*
also have $\langle \dots = (\sum_{\infty} e \in \text{some-chilbert-basis}. norm (norm ((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} e) * norm (sqrt-op (abs-op b) *_{V} e))) \rangle$
by *simp*
also have $\langle \dots \leq sqrt (\sum_{\infty} e \in \text{some-chilbert-basis}. (norm (norm ((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} e)))^2) * sqrt (\sum_{\infty} e \in \text{some-chilbert-basis}. (norm (norm (sqrt-op (abs-op b) *_{V} e)))^2) \rangle$
apply (*rule Cauchy-Schwarz-ineq-infsum*)
using $sum4 \ sum5$ **by** *auto*
also have $\langle \dots = sqrt (\sum_{\infty} e \in \text{some-chilbert-basis}. (norm ((sqrt-op (abs-op b) \ o_{CL} \ W * \ o_{CL} \ a*) *_{V} e))^2) * sqrt (\sum_{\infty} e \in \text{some-chilbert-basis}. (norm (sqrt-op (abs-op b) *_{V} e))^2) \rangle$

```

  by simp
  also have ⟨... = hilbert-schmidt-norm (sqrt-op (abs-op b) oCL W* oCL a*) * hilbert-schmidt-norm
  (sqrt-op (abs-op b))⟩
    apply (subst infsum-hilbert-schmidt-norm-square, simp, fact hs2)
    apply (subst infsum-hilbert-schmidt-norm-square, simp, fact hs1)
    by simp
  also have ⟨... ≤ hilbert-schmidt-norm (sqrt-op (abs-op b)) * norm (W* oCL a*) * hilbert-schmidt-norm
  (sqrt-op (abs-op b))⟩
    by (metis cblinfun-assoc-left(1) hilbert-schmidt-norm-comp-left hilbert-schmidt-norm-pos mult.commute
  mult-right-mono that trace-class-iff-sqrt-hs)
    also have ⟨... ≤ hilbert-schmidt-norm (sqrt-op (abs-op b)) * norm (W*) * norm (a*) *
  hilbert-schmidt-norm (sqrt-op (abs-op b))⟩
      by (metis (no-types, lifting) ab-semigroup-mult-class.mult-ac(1) hilbert-schmidt-norm-pos
  mult-right-mono norm-cblinfun-compose ordered-comm-semiring-class.comm-mult-left-mono)
    also have ⟨... ≤ hilbert-schmidt-norm (sqrt-op (abs-op b)) * norm (a*) * hilbert-schmidt-norm
  (sqrt-op (abs-op b))⟩
      by (metis ⟨norm W ≤ 1⟩ hilbert-schmidt-norm-pos mult.right-neutral mult-left-mono mult-right-mono
  norm-adj norm-ge-zero)
    also have ⟨... = norm a * (hilbert-schmidt-norm (sqrt-op (abs-op b)))2⟩
      by (simp add: power2-eq-square)
    also have ⟨... = norm a * trace-norm b⟩
      apply (simp add: hilbert-schmidt-norm-def positive-selfadjointI[unfolded selfadjoint-def])
      by (metis abs-op-idem of-real-eq-iff trace-abs-op)
  finally show ?thesis
    by -
qed

```

lemma *trace-leq-trace-norm*[simp]: $\langle \text{cmod } (\text{trace } a) \leq \text{trace-norm } a \rangle$

proof (cases ⟨trace-class a⟩)

case True

then have $\langle \text{cmod } (\text{trace } a) \leq \text{norm } (\text{id-cblinfun } :: 'a \Rightarrow_{CL} 'a) * \text{trace-norm } a \rangle$

using *cmod-trace-times*[**where** $a = \langle \text{id-cblinfun } :: 'a \Rightarrow_{CL} 'a \rangle$ **and** $b = a$]

by *simp*

also have $\langle \dots \leq \text{trace-norm } a \rangle$

apply (rule *mult-left-le-one-le*)

by (auto intro!: *mult-left-le-one-le simp: norm-cblinfun-id-le*)

finally show ?thesis

by -

next

case False

then show ?thesis

by (simp add: *trace-def*)

qed

lemma *trace-norm-triangle*:

fixes $a b :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$

assumes [simp]: $\langle \text{trace-class } a \rangle \langle \text{trace-class } b \rangle$

shows $\langle \text{trace-norm } (a + b) \leq \text{trace-norm } a + \text{trace-norm } b \rangle$

— [1], Theorem 18.11 (a)

```

proof –
  define w where  $\langle w = \text{polar-decomposition } (a+b) \rangle$ 
  have  $\langle \text{norm } (w^*) \leq 1 \rangle$ 
  by (metis dual-order.refl norm-adj norm-partial-isometry norm-zero polar-decomposition-partial-isometry
w-def zero-less-one-class.zero-le-one)
  have  $\langle \text{trace-norm } (a + b) = \text{cmod } (\text{trace } (\text{abs-op } (a+b))) \rangle$ 
  by simp
  also have  $\langle \dots = \text{cmod } (\text{trace } (w^* \text{ o}_{CL} (a+b))) \rangle$ 
  by (simp add: polar-decomposition-correct' w-def)
  also have  $\langle \dots \leq \text{cmod } (\text{trace } (w^* \text{ o}_{CL} a)) + \text{cmod } (\text{trace } (w^* \text{ o}_{CL} b)) \rangle$ 
  by (simp add: cblinfun-compose-add-right norm-triangle-ineq trace-class-comp-right trace-plus)
  also have  $\langle \dots \leq (\text{norm } (w^*) * \text{trace-norm } a) + (\text{norm } (w^*) * \text{trace-norm } b) \rangle$ 
  by (smt (verit, best) assms(1) assms(2) cmod-trace-times)
  also have  $\langle \dots \leq \text{trace-norm } a + \text{trace-norm } b \rangle$ 
  using  $\langle \text{norm } (w^*) \leq 1 \rangle$ 
  by (smt (verit, ccfv-SIG) mult-le-cancel-right2 trace-norm-nneg)
  finally show ?thesis
  by –
qed

instantiation trace-class :: (chilbert-space, chilbert-space) {complex-vector} begin

lift-definition zero-trace-class ::  $\langle ('a, 'b) \text{ trace-class} \rangle$  is 0 by auto
lift-definition minus-trace-class ::  $\langle ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \rangle$ 
is minus by auto
lift-definition uminus-trace-class ::  $\langle ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \rangle$  is uminus by simp
lift-definition plus-trace-class ::  $\langle ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \rangle$ 
is plus by auto
lift-definition scaleC-trace-class ::  $\langle \text{complex} \Rightarrow ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \rangle$  is scaleC
  by (metis (no-types, opaque-lifting) cblinfun-compose-id-right cblinfun-compose-scaleC-right
mem-Collect-eq trace-class-comp-left)
lift-definition scaleR-trace-class ::  $\langle \text{real} \Rightarrow ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \rangle$  is scaleR
  by (metis (no-types, opaque-lifting) cblinfun-compose-id-right cblinfun-compose-scaleC-right
mem-Collect-eq scaleR-scaleC trace-class-comp-left)
instance
proof standard
  fix a b c ::  $\langle ('a, 'b) \text{ trace-class} \rangle$ 
  show  $\langle a + b + c = a + (b + c) \rangle$ 
  apply transfer by auto
  show  $\langle a + b = b + a \rangle$ 
  apply transfer by auto
  show  $\langle 0 + a = a \rangle$ 
  apply transfer by auto
  show  $\langle - a + a = 0 \rangle$ 
  apply transfer by auto
  show  $\langle a - b = a + - b \rangle$ 
  apply transfer by auto
  show  $\langle (*_R) r = ((*_C) (\text{complex-of-real } r)) :: - \Rightarrow ('a, 'b) \text{ trace-class} \rangle$  for r :: real
  by (metis (mono-tags, opaque-lifting) Trace-Class.scaleC-trace-class-def Trace-Class.scaleR-trace-class-def)

```



```

id-apply map-fun-def o-def scaleR-scaleC)
  show ⟨r *C (a + b) = r *C a + r *C b⟩ for r :: complex
    apply transfer
    by (metis (no-types, lifting) scaleC-add-right)
  show ⟨(r + r') *C a = r *C a + r' *C a⟩ for r r' :: complex
    apply transfer
    by (metis (no-types, lifting) scaleC-add-left)
  show ⟨r *C r' *C a = (r * r') *C a⟩ for r r' :: complex
    apply transfer by auto
  show ⟨1 *C a = a⟩
    apply transfer by auto
qed
end

```

```

lemma from-trace-class-0[simp]: ⟨from-trace-class 0 = 0⟩
  by (simp add: zero-trace-class.rep-eq)

```

```

instantiation trace-class :: (chilbert-space, chilbert-space) {complex-normed-vector} begin

```

```

lift-definition norm-trace-class :: ⟨('a,'b) trace-class ⇒ real⟩ is trace-norm .
definition sgn-trace-class :: ⟨('a,'b) trace-class ⇒ ('a,'b) trace-class⟩ where ⟨sgn-trace-class a
= a /R norm a⟩
definition dist-trace-class :: ⟨('a,'b) trace-class ⇒ - ⇒ -⟩ where ⟨dist-trace-class a b = norm
(a - b)⟩
definition [code del]: uniformity-trace-class = (INF e∈{0<..}. principal {(x::('a,'b) trace-class,
y). dist x y < e})
definition [code del]: open-trace-class U = (∀ x∈U. ∀F (x', y) in INF e∈{0<..}. principal {(x,
y). dist x y < e}. x' = x ⟶ y ∈ U) for U :: ('a,'b) trace-class set
instance
proof standard
  fix a b :: ⟨('a,'b) trace-class⟩
  show ⟨dist a b = norm (a - b)⟩
    by (metis (no-types, lifting) Trace-Class.dist-trace-class-def)
  show ⟨uniformity = (INF e∈{0<..}. principal {(x :: ('a,'b) trace-class, y). dist x y < e})⟩
    by (simp add: uniformity-trace-class-def)
  show ⟨open U = (∀ x∈U. ∀F (x', y) in uniformity. x' = x ⟶ y ∈ U)⟩ for U :: ⟨('a,'b)
trace-class set⟩
    by (smt (verit, del-insts) case-prod-beta' eventually-mono open-trace-class-def uniformity-trace-class-def)
  show ⟨(norm a = 0) = (a = 0)⟩
    apply transfer
    by (auto simp add: trace-norm-nondegenerate)
  show ⟨norm (a + b) ≤ norm a + norm b⟩
    apply transfer
    by (auto simp: trace-norm-triangle)
  show ⟨norm (r *C a) = cmod r * norm a⟩ for r
    apply transfer
    by (auto simp: trace-norm-scaleC)
  then show ⟨norm (r *R a) = |r| * norm a⟩ for r
    by (metis norm-of-real scaleR-scaleC)

```

```

show ⟨sgn a = a /R norm a⟩
  by (simp add: sgn-trace-class-def)
qed
end

```

lemma *trace-norm-comp-right*:

```

fixes a :: ⟨'b::hilbert-space ⇒CL 'c::hilbert-space⟩ and b :: ⟨'a::hilbert-space ⇒CL 'b⟩
assumes ⟨trace-class b⟩
shows ⟨trace-norm (a oCL b) ≤ norm a * trace-norm b⟩
  — [1], Theorem 18.11 (g)
proof —
  define w w1 s where ⟨w = polar-decomposition b⟩ and ⟨w1 = polar-decomposition (a oCL b)⟩
  and ⟨s = w1* oCL a oCL w⟩
  have abs-ab: ⟨abs-op (a oCL b) = s oCL abs-op b⟩
  by (auto simp: w1-def w-def s-def cblinfun-compose-assoc polar-decomposition-correct polar-decomposition-correct')
  have norm-s-t: ⟨norm s ≤ norm a⟩
  proof —
  have ⟨norm s ≤ norm (w1* oCL a) * norm w⟩
  by (simp add: norm-cblinfun-compose s-def)
  also have ⟨... ≤ norm (w1*) * norm a * norm w⟩
  by (metis mult.commute mult-left-mono norm-cblinfun-compose norm-ge-zero)
  also have ⟨... ≤ norm a⟩
  by (metis (no-types, opaque-lifting) dual-order.refl mult.commute mult.right-neutral mult-zero-left norm-adj norm-ge-zero norm-partial-isometry norm-zero polar-decomposition-partial-isometry w1-def w-def)
  finally show ?thesis
  by —
qed
have ⟨trace-norm (a oCL b) = cmod (trace (abs-op (a oCL b)))⟩
  by simp
also have ⟨... = cmod (trace (s oCL abs-op b))⟩
  using abs-ab by presburger
also have ⟨... ≤ norm s * trace-norm (abs-op b)⟩
  using assms by (simp add: cmod-trace-times)
also from norm-s-t have ⟨... ≤ norm a * trace-norm b⟩
  by (metis abs-op-idem mult-right-mono of-real-eq-iff trace-abs-op trace-norm-nneg)
finally show ?thesis
  by —
qed

```

lemma *trace-norm-comp-left*:

— [1], Theorem 18.11 (g)

```

fixes a :: ⟨'b::chilbert-space ⇒CL 'c::chilbert-space⟩ and b :: ⟨'a::chilbert-space ⇒CL 'b⟩
assumes [simp]: ⟨trace-class a⟩
shows ⟨trace-norm (a oCL b) ≤ trace-norm a * norm b⟩
proof –
  have ⟨trace-norm (b* oCL a*) ≤ norm (b*) * trace-norm (a*)⟩
    apply (rule trace-norm-comp-right)
    by simp
  then have ⟨trace-norm ((b* oCL a*)*) ≤ norm b * trace-norm a⟩
    by (simp del: adj-cblinfun-compose)
  then show ?thesis
    by (simp add: mult.commute)
qed

lemma bounded-clinear-trace-duality: ⟨trace-class t ⇒ bounded-clinear (λa. trace (t oCL a))⟩
  apply (rule bounded-clinearI[where K=⟨trace-norm t⟩])
  apply (auto simp add: cblinfun-compose-add-right trace-class-comp-left trace-plus trace-scaleC)[2]
  by (metis circularity-of-trace order-trans trace-leq-trace-norm trace-norm-comp-right)

lemma trace-class-butterfly[simp]: ⟨trace-class (butterfly x y)⟩ for x :: ⟨'a::chilbert-space⟩ and y
  :: ⟨'b::chilbert-space⟩
  unfolding butterfly-def
  apply (rule trace-class-comp-left)
  by simp

lemma trace-adj: ⟨trace (a*) = cnj (trace a)⟩
  by (metis Complex-Inner-Product0.complex-inner-1-right cinner-zero-right double-adj is-onb-some-chilbert-basis
  is-orthogonal-sym trace-adj-prelim trace-alt-def trace-class-adj)
hide-fact trace-adj-prelim

lemma cmod-trace-times': ⟨cmod (trace (a oCL b)) ≤ norm b * trace-norm a⟩ if ⟨trace-class a⟩
  – [1], Theorem 18.11 (e)
  apply (subst asm-rl[of ⟨a oCL b = (b* oCL a*)*⟩], simp)
  apply (subst trace-adj)
  using cmod-trace-times[of ⟨a*⟩ ⟨b*⟩]
  by (auto intro!: that trace-class-adj hilbert-schmidt-comp-right hilbert-schmidt-adj simp del:
  adj-cblinfun-compose)

lift-definition iso-trace-class-compact-op-dual' :: ⟨('a::chilbert-space,'b::chilbert-space) trace-class
  ⇒ ('b,'a) compact-op ⇒CL complex⟩ is
  ⟨λt c. trace (from-compact-op c oCL t)⟩
proof (rename-tac t)
  include lifting-syntax
  fix t :: ⟨'a ⇒CL 'b⟩
  assume ⟨t ∈ Collect trace-class⟩
  then have [simp]: ⟨trace-class t⟩
    by simp
  have ⟨cmod (trace (from-compact-op x oCL t)) ≤ norm x * trace-norm t⟩ for x
    by (metis ⟨trace-class t⟩ cmod-trace-times from-compact-op-norm)

```

then show $\langle \text{bounded-clinear } (\lambda c. \text{trace } (\text{from-compact-op } c \text{ } o_{CL} \ t)) \rangle$
apply (rule-tac bounded-clinearI[**where** $K = \langle \text{trace-norm } t \rangle$])
by (auto simp: from-compact-op-plus from-compact-op-scaleC cblinfun-compose-add-right
cblinfun-compose-add-left trace-plus trace-class-comp-right trace-scaleC)

qed

lemma iso-trace-class-compact-op-dual'-apply: $\langle \text{iso-trace-class-compact-op-dual}' \ t \ c = \text{trace } (\text{from-compact-op } c \text{ } o_{CL} \ \text{from-trace-class } t) \rangle$
by (simp add: iso-trace-class-compact-op-dual'.rep-eq)

lemma iso-trace-class-compact-op-dual'-plus: $\langle \text{iso-trace-class-compact-op-dual}' \ (a + b) = \text{iso-trace-class-compact-op-dual}' \ a + \text{iso-trace-class-compact-op-dual}' \ b \rangle$
apply transfer
by (simp add: cblinfun-compose-add-right trace-class-comp-right trace-plus)

lemma iso-trace-class-compact-op-dual'-scaleC: $\langle \text{iso-trace-class-compact-op-dual}' \ (c *_{C} \ a) = c *_{C} \ \text{iso-trace-class-compact-op-dual}' \ a \rangle$
apply transfer
by (simp add: trace-scaleC)

lemma iso-trace-class-compact-op-dual'-bounded-clinear[bounded-clinear, simp]:

— [1], Theorem 19.1

$\langle \text{bounded-clinear } (\text{iso-trace-class-compact-op-dual}' :: ('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \ \text{trace-class} \Rightarrow -) \rangle$

proof —

let $?iso = \langle \text{iso-trace-class-compact-op-dual}' :: ('a, 'b) \ \text{trace-class} \Rightarrow - \rangle$

have $\langle \text{norm } (?iso \ t) \leq \text{norm } t \rangle$ **for** t

proof (rule norm-cblinfun-bound)

show $\langle \text{norm } t \geq 0 \rangle$ **by** simp

fix c

show $\langle \text{cmod } (\text{iso-trace-class-compact-op-dual}' \ t *_{V} \ c) \leq \text{norm } t * \text{norm } c \rangle$

apply (transfer fixing: c)

apply simp

by (metis cmod-trace-times from-compact-op-norm ordered-field-class.sign-simps(5))

qed

then show $\langle \text{bounded-clinear } ?iso \rangle$

apply (rule-tac bounded-clinearI[**where** $K = I$])

by (auto simp: iso-trace-class-compact-op-dual'-plus iso-trace-class-compact-op-dual'-scaleC)

qed

lemma iso-trace-class-compact-op-dual'-surjective[simp]:

$\langle \text{surj } (\text{iso-trace-class-compact-op-dual}' :: ('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \ \text{trace-class} \Rightarrow -) \rangle$

proof —

let $?iso = \langle \text{iso-trace-class-compact-op-dual}' :: ('a, 'b) \ \text{trace-class} \Rightarrow - \rangle$

have $\langle \exists A. \Phi = ?iso \ A \rangle$ **for** $\Phi :: \langle ('b, 'a) \ \text{compact-op} \Rightarrow_{CL} \ \text{complex} \rangle$

proof —

define p **where** $\langle p \ x \ y = \Phi \ (\text{butterfly-co } y \ x) \rangle$ **for** $x \ y$

have $\text{norm-}p$: $\langle \text{norm } (p \ x \ y) \leq \text{norm } \Phi * \text{norm } x * \text{norm } y \rangle$ **for** $x \ y$

```

proof –
  have  $\langle \text{norm } (p \ x \ y) \leq \text{norm } \Phi * \text{norm } (\text{butterfly-co } y \ x) \rangle$ 
    by (auto simp: p-def norm-cblinfun)
  also have  $\langle \dots = \text{norm } \Phi * \text{norm } (\text{butterfly } y \ x) \rangle$ 
    apply transfer by simp
  also have  $\langle \dots = \text{norm } \Phi * \text{norm } x * \text{norm } y \rangle$ 
    by (simp add: norm-butterfly)
  finally show ?thesis
    by –
qed
have [simp]:  $\langle \text{bounded-sesquilinear } p \rangle$ 
  apply (rule bounded-sesquilinear.intro)
  using norm-p
  by (auto
    intro!: exI[of -  $\langle \text{norm } \Phi \rangle$ ]
    simp add: p-def butterfly-co-add-left butterfly-co-add-right complex-vector.linear-add
    cblinfun.scaleC-right cblinfun.scaleC-left ab-semigroup-mult-class.mult-ac)
define A where  $\langle A = (\text{the-riesz-rep-sesqui } p) * \rangle$ 
then have xAy:  $\langle x \cdot_C (A \ y) = p \ x \ y \rangle$  for x y
  by (simp add: cinner-adj-right the-riesz-rep-sesqui-apply)
have  $\Phi C$ :  $\langle \Phi \ C = \text{trace } (\text{from-compact-op } C \ o_{CL} \ A) \rangle$  if  $\langle \text{finite-rank } (\text{from-compact-op } C) \rangle$ 
for C
proof –
  from that
  obtain x y and n :: nat where C-sum:  $\langle \text{from-compact-op } C = (\sum_{i < n}. \text{butterfly } (y \ i) \ (x \ i)) \rangle$ 
  apply atomize-elim by (rule finite-rank-sum-butterfly)
  then have  $\langle C = (\sum_{i < n}. \text{butterfly-co } (y \ i) \ (x \ i)) \rangle$ 
  apply transfer by simp
  then have  $\langle \Phi \ C = (\sum_{i < n}. \Phi *_V \text{butterfly-co } (y \ i) \ (x \ i)) \rangle$ 
  using cblinfun.sum-right by blast
  also have  $\langle \dots = (\sum_{i < n}. p \ (x \ i) \ (y \ i)) \rangle$ 
  using p-def by presburger
  also have  $\langle \dots = (\sum_{i < n}. (x \ i) \cdot_C (A \ (y \ i))) \rangle$ 
  using xAy by presburger
  also have  $\langle \dots = (\sum_{i < n}. \text{trace } (\text{butterfly } (y \ i) \ (x \ i) \ o_{CL} \ A)) \rangle$ 
  by (simp add: trace-butterfly-comp)
  also have  $\langle \dots = \text{trace } ((\sum_{i < n}. \text{butterfly } (y \ i) \ (x \ i)) \ o_{CL} \ A) \rangle$ 
  by (metis (mono-tags, lifting) cblinfun.compose-sum-left sum.cong trace-class-butterfly
trace-class-comp-left trace-sum)
  also have  $\langle \dots = \text{trace } (\text{from-compact-op } C \ o_{CL} \ A) \rangle$ 
  using C-sum by presburger
  finally show ?thesis
    by –
qed
have  $\langle \text{trace-class } A \rangle$ 
proof (rule trace-classI)
  show  $\langle \text{is-onb some-chilbert-basis} \rangle$ 
  by simp

```

```

define  $W$  where  $\langle W = \text{polar-decomposition } A \rangle$ 
have  $\langle \text{norm } (W^*) \leq 1 \rangle$ 
  by (metis W-def nle-le norm-adj norm-partial-isometry norm-zero not-one-le-zero polar-decomposition-partial-isometry)
    have  $\langle (\sum_{x \in E}. \text{cmod } (x \cdot_C (\text{abs-op } A *_V x))) \leq \text{norm } \Phi \rangle$  if  $\langle \text{finite } E \rangle$  and  $\langle E \subseteq \text{some-chilbert-basis} \rangle$  for  $E$ 
    proof –
      define  $CE$  where  $\langle CE = (\sum_{x \in E}. (\text{butterfly } x x)) \rangle$ 
      from  $\langle E \subseteq \text{some-chilbert-basis} \rangle$ 
      have  $\langle \text{norm } CE \leq 1 \rangle$ 
      by (auto intro!: sum-butterfly-is-Proj norm-is-Proj is-normal-some-chilbert-basis simp: CE-def is-ortho-set-antimono)
      have  $\langle (\sum_{x \in E}. \text{cmod } (x \cdot_C (\text{abs-op } A *_V x))) = \text{cmod } (\sum_{x \in E}. x \cdot_C (\text{abs-op } A *_V x)) \rangle$ 
      apply (rule sum-cmod-pos)
      by (simp add: cinner-pos-if-pos)
      also have  $\langle \dots = \text{cmod } (\sum_{x \in E}. (W *_V x) \cdot_C (A *_V x)) \rangle$ 
      apply (rule arg-cong, rule sum.cong, simp)
by (metis W-def cblinfun-apply-cblinfun-compose cinner-adj-right polar-decomposition-correct')
      also have  $\langle \dots = \text{cmod } (\sum_{x \in E}. \Phi (\text{butterfly-co } x (W x))) \rangle$ 
      apply (rule arg-cong, rule sum.cong, simp)
      by (simp flip: p-def xAy)
      also have  $\langle \dots = \text{cmod } (\Phi (\sum_{x \in E}. \text{butterfly-co } x (W x))) \rangle$ 
      by (simp add: cblinfun.sum-right)
      also have  $\langle \dots \leq \text{norm } \Phi * \text{norm } (\sum_{x \in E}. \text{butterfly-co } x (W x)) \rangle$ 
      using norm-cblinfun by blast
      also have  $\langle \dots = \text{norm } \Phi * \text{norm } (\sum_{x \in E}. \text{butterfly } x (W x)) \rangle$ 
      apply transfer by simp
      also have  $\langle \dots = \text{norm } \Phi * \text{norm } (\sum_{x \in E}. (\text{butterfly } x x \circ_{CL} W^*)) \rangle$ 
      apply (rule arg-cong, rule sum.cong, simp)
      by (simp add: butterfly-comp-cblinfun)
      also have  $\langle \dots = \text{norm } \Phi * \text{norm } (CE \circ_{CL} W^*) \rangle$ 
      by (simp add: CE-def cblinfun-compose-sum-left)
      also have  $\langle \dots \leq \text{norm } \Phi \rangle$ 
      apply (rule mult-left-le, simp-all)
      using  $\langle \text{norm } CE \leq 1 \rangle$   $\langle \text{norm } (W^*) \leq 1 \rangle$ 
      by (metis mult-le-one norm-cblinfun-compose norm-ge-zero order-trans)
      finally show ?thesis
      by –
qed
then show  $\langle (\lambda x. x \cdot_C (\text{abs-op } A *_V x)) \text{abs-summable-on some-chilbert-basis} \rangle$ 
apply (rule-tac nonneg-bdd-above-summable-on)
by (auto intro!: bdd-aboveI2)
qed
then obtain  $A'$  where  $A'$ :  $\langle A = \text{from-trace-class } A' \rangle$ 
using from-trace-class-cases by blast
from  $\Phi C$  have  $\Phi C'$ :  $\langle \Phi C = ?\text{iso } A' C \rangle$  if  $\langle \text{finite-rank } (\text{from-compact-op } C) \rangle$  for  $C$ 
by (simp add: that iso-trace-class-compact-op-dual'-apply A')
have  $\langle \Phi = ?\text{iso } A' \rangle$ 
apply (unfold cblinfun-apply-inject[symmetric])

```

```

    apply (rule finite-rank-separating-on-compact-op)
    using  $\Phi C'$  by (auto intro!: cblinfun.bounded-clinear-right)
  then show ?thesis
    by auto
qed
then show ?thesis
  by auto
qed

lemma iso-trace-class-compact-op-dual'-isometric[simp]:
  — [1], Theorem 19.1
   $\langle \text{norm } (iso\text{-trace-class-compact-op-dual}' t) = \text{norm } t \rangle$  for  $t :: \langle 'a::\text{chilbert-space}, 'b::\text{chilbert-space} \rangle$ 
  trace-class
proof —
  let ?iso =  $\langle iso\text{-trace-class-compact-op-dual}' :: ('a,'b) \text{ trace-class} \Rightarrow - \rangle$ 
  have  $\langle \text{norm } (?iso t) \leq \text{norm } t \rangle$  for  $t$ 
  proof (rule norm-cblinfun-bound)
    show  $\langle \text{norm } t \geq 0 \rangle$  by simp
    fix  $c$ 
    show  $\langle \text{cmod } (iso\text{-trace-class-compact-op-dual}' t *_{\vee} c) \leq \text{norm } t * \text{norm } c \rangle$ 
      apply (transfer fixing: c)
      apply simp
      by (metis cmod-trace-times from-compact-op-norm ordered-field-class.sign-simps(5))
  qed
  moreover have  $\langle \text{norm } (?iso t) \geq \text{norm } t \rangle$  for  $t$ 
  proof —
    define  $s$  where  $\langle s E = (\sum e \in E. \text{cmod } (e \cdot_C (\text{abs-op } (\text{from-trace-class } t) *_{\vee} e))) \rangle$  for  $E$ 
    have bound:  $\langle \text{norm } (?iso t) \geq s E \rangle$  if  $\langle \text{finite } E \rangle$  and  $\langle E \subseteq \text{some-chilbert-basis} \rangle$  for  $E$ 
    proof —

```

Partial duplication from the proof of *iso-trace-class-compact-op-dual'-surjective*. In Conway's text, this subproof occurs only once. However, it did not become clear to use how this works: It seems that Conway's proof only implies that *iso-trace-class-compact-op-dual'* is isometric on the subset of trace-class operators A constructed in that proof, but not necessarily on others (if *iso-trace-class-compact-op-dual'* were non-injective, there might be others)

```

    define  $A \Phi$  where  $\langle A = \text{from-trace-class } t \rangle$  and  $\langle \Phi = ?iso t \rangle$ 
    define  $W$  where  $\langle W = \text{polar-decomposition } A \rangle$ 
    have  $\langle \text{norm } (W*) \leq 1 \rangle$ 
      by (metis W-def nle-le norm-adj norm-partial-isometry norm-zero not-one-le-zero polar-decomposition-partial-isometry)
    define  $CE$  where  $\langle CE = (\sum x \in E. (\text{butterfly } x x)) \rangle$ 
    from  $\langle E \subseteq \text{some-chilbert-basis} \rangle$ 
    have  $\langle \text{norm } CE \leq 1 \rangle$ 
      by (auto intro!: sum-butterfly-is-Proj norm-is-Proj is-normal-some-chilbert-basis simp: CE-def is-ortho-set-antimono)
    have  $\langle s E = (\sum x \in E. \text{cmod } (x \cdot_C (\text{abs-op } A *_{\vee} x))) \rangle$ 
      using A-def s-def by blast

```

```

also have ⟨... = cmod (∑ x∈E. x •C (abs-op A *V x))⟩
  apply (rule sum-cmod-pos)
  by (simp add: cinner-pos-if-pos)
also have ⟨... = cmod (∑ x∈E. (W *V x) •C (A *V x))⟩
  apply (rule arg-cong, rule sum.cong, simp)
by (metis W-def cblinfun-apply-cblinfun-compose cinner-adj-right polar-decomposition-correct')
also have ⟨... = cmod (∑ x∈E. Φ (butterfly-co x (W x)))⟩
  apply (rule arg-cong, rule sum.cong, simp)
by (auto simp: Φ-def iso-trace-class-compact-op-dual'-apply butterfly-co.rep-eq trace-butterfly-comp
  simp flip: A-def)
also have ⟨... = cmod (Φ (∑ x∈E. butterfly-co x (W x)))⟩
  by (simp add: cblinfun.sum-right)
also have ⟨... ≤ norm Φ * norm (∑ x∈E. butterfly-co x (W x))⟩
  using norm-cblinfun by blast
also have ⟨... = norm Φ * norm (∑ x∈E. butterfly x (W x))⟩
  apply transfer by simp
also have ⟨... = norm Φ * norm (∑ x∈E. (butterfly x x oCL W*))⟩
  apply (rule arg-cong, rule sum.cong, simp)
  by (simp add: butterfly-comp-cblinfun)
also have ⟨... = norm Φ * norm (CE oCL W*)⟩
  by (simp add: CE-def cblinfun-compose-sum-left)
also have ⟨... ≤ norm Φ⟩
  apply (rule mult-left-le, simp-all)
  using ⟨norm CE ≤ 1⟩ ⟨norm (W*) ≤ 1⟩
  by (metis mult-le-one norm-cblinfun-compose norm-ge-zero order-trans)
finally show ?thesis
  by (simp add: Φ-def)
qed
have ⟨trace-class (from-trace-class t)⟩ and ⟨norm t = trace-norm (from-trace-class t)⟩
  using from-trace-class
  by (auto simp add: norm-trace-class.rep-eq)
then have ⟨((λe. cmod (e •C (abs-op (from-trace-class t) *V e))) has-sum norm t) some-chilbert-basis)⟩

  by (metis (no-types, lifting) has-sum-cong has-sum-infsum is-onb-some-chilbert-basis trace-class-def
  trace-norm-alt-def trace-norm-basis-invariance)
  then have lim: ⟨(s ⟶ norm t) (finite-subsets-at-top some-chilbert-basis)⟩
    by (simp add: filterlim-iff has-sum-def s-def)
  show ?thesis
    using - - lim apply (rule tendsto-le)
    by (auto intro!: tendsto-const eventually-finite-subsets-at-top-weakI bound)
qed
ultimately show ?thesis
  using nle-le by blast
qed

instance trace-class :: (chilbert-space, chilbert-space) cbanach
proof
  let ?UNIVc = ⟨UNIV :: (('b,'a) compact-op ⇒CL complex) set⟩

```



```

let ?UNIVt = ⟨UNIV :: ('a,'b) trace-class set⟩
let ?iso = ⟨iso-trace-class-compact-op-dual' :: ('a,'b) trace-class ⇒ -⟩
have lin-inv[simp]: ⟨bounded-clinear (inv ?iso)⟩
  apply (rule bounded-clinear-inv[where b=1])
  by auto
have [simp]: ⟨inj ?iso⟩
proof (rule injI)
  fix x y assume ⟨?iso x = ?iso y⟩
  then have ⟨norm (?iso (x - y)) = 0⟩
  by (metis (no-types, opaque-lifting) add-diff-cancel-left diff-self iso-trace-class-compact-op-dual'-isometric
iso-trace-class-compact-op-dual'-plus norm-eq-zero ordered-field-class.sign-simps(12))
  then have ⟨norm (x - y) = 0⟩
  by simp
  then show ⟨x = y⟩
  by simp
qed
have norm-inv[simp]: ⟨norm (inv ?iso x) = norm x⟩ for x
  by (metis iso-trace-class-compact-op-dual'-isometric iso-trace-class-compact-op-dual'-surjective
surj-f-inv-f)
have ⟨complete ?UNIVc⟩
  by (simp add: complete-UNIV)
then have ⟨complete (inv ?iso ' ?UNIVc)⟩
  apply (rule complete-isometric-image[rotated 4, where e=1])
  by (auto simp: bounded-clinear.bounded-linear)
then have ⟨complete ?UNIVt⟩
  by (simp add: inj-imp-surj-inv)
then show ⟨Cauchy X ⇒ convergent X⟩ for X :: ⟨nat ⇒ ('a, 'b) trace-class⟩
  by (simp add: complete-def convergent-def)
qed

```

lemma *trace-norm-geq-cinner-abs-op*: $\langle \psi \cdot_C (\text{abs-op } t *_V \psi) \leq \text{trace-norm } t \rangle$ **if** $\langle \text{trace-class } t \rangle$
and $\langle \text{norm } \psi = 1 \rangle$

proof –

have $\langle \exists B. \{\psi\} \subseteq B \wedge \text{is-onb } B \rangle$

apply (rule orthonormal-basis-exists)

using $\langle \text{norm } \psi = 1 \rangle$

by auto

then obtain B **where** $\langle \text{is-onb } B \rangle$ **and** $\langle \psi \in B \rangle$

by auto

have $\langle \psi \cdot_C (\text{abs-op } t *_V \psi) = (\sum_{\infty \psi \in \{\psi\}} \psi \cdot_C (\text{abs-op } t *_V \psi)) \rangle$

by simp

also have $\langle \dots \leq (\sum_{\infty \psi \in B} \psi \cdot_C (\text{abs-op } t *_V \psi)) \rangle$

apply (rule infsum-mono-neutral-complex)

using $\langle \psi \in B \rangle \langle \text{is-onb } B \rangle$ *that*

by (auto simp add: trace-exists cinner-pos-if-pos)

also have $\langle \dots = \text{trace-norm } t \rangle$

```

    using ⟨is-onb B⟩ that
    by (metis trace-abs-op trace-alt-def trace-class-abs-op)
  finally show ?thesis
    by -
qed

lemma norm-leq-trace-norm: ⟨norm t ≤ trace-norm t⟩ if ⟨trace-class t⟩
  for t :: ⟨'a::hilbert-space ⇒CL 'b::hilbert-space⟩
proof -
  wlog not-singleton: ⟨class.not-singleton TYPE('a)⟩
    using not-not-singleton-cblinfun-zero[of t] negation by simp
  note cblinfun-norm-approx-witness' = cblinfun-norm-approx-witness[internalize-sort' 'a, OF
  complex-normed-vector-axioms not-singleton]

  show ?thesis
  proof (rule field-le-epsilon)
    fix ε :: real assume ⟨ε > 0⟩

    define δ :: real where
      ⟨δ = min (sqrt (ε / 2)) (ε / (4 * (norm (sqrt-op (abs-op t)) + 1)))⟩
    have ⟨δ > 0⟩
      using ⟨ε > 0⟩ apply (auto simp add: δ-def)
      by (smt (verit) norm-not-less-zero zero-less-divide-iff)
    have δ-small: ⟨δ2 + 2 * norm (sqrt-op (abs-op t)) * δ ≤ ε⟩
    proof -
      define n where ⟨n = norm (sqrt-op (abs-op t))⟩
      then have ⟨n ≥ 0⟩
        by simp
      have δ: ⟨δ = min (sqrt (ε / 2)) (ε / (4 * (n + 1)))⟩
        by (simp add: δ-def n-def)

      have ⟨δ2 + 2 * n * δ ≤ ε / 2 + 2 * n * δ⟩
        apply (rule add-right-mono)
        apply (subst δ) apply (subst min-power-distrib-left)
        using ⟨ε > 0⟩ ⟨n ≥ 0⟩ by auto
      also have ⟨... ≤ ε / 2 + 2 * n * (ε / (4 * (n + 1)))⟩
        apply (intro add-left-mono mult-left-mono)
        by (simp-all add: δ ⟨n ≥ 0⟩)
      also have ⟨... = ε / 2 + 2 * (n / (n+1)) * (ε / 4)⟩
        by simp
      also have ⟨... ≤ ε / 2 + 2 * 1 * (ε / 4)⟩
        apply (intro add-left-mono mult-left-mono mult-right-mono)
        using ⟨n ≥ 0⟩ ⟨ε > 0⟩ by auto
      also have ⟨... = ε⟩
        by simp
      finally show ⟨δ2 + 2 * n * δ ≤ ε⟩
        by -
    qed
  qed

```

```

from  $\langle \delta > 0 \rangle$  obtain  $\psi$  where  $\psi \varepsilon$ :  $\langle \text{norm} (\text{sqrt-op} (\text{abs-op } t)) - \delta \leq \text{norm} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi) \rangle$  and  $\langle \text{norm } \psi = 1 \rangle$ 
  apply atomize-elim by (rule cblinfun-norm-approx-witness')

have aux1:  $\langle 2 * \text{complex-of-real } x = \text{complex-of-real} (2 * x) \rangle$  for  $x$ 
  by simp

have  $\langle \text{complex-of-real} (\text{norm } t) = \text{norm} (\text{abs-op } t) \rangle$ 
  by simp
also have  $\langle \dots = (\text{norm} (\text{sqrt-op} (\text{abs-op } t)))^2 \rangle$ 
  by (simp add: positive-selfadjointI[unfolded selfadjoint-def] flip: norm-AadjA)
also have  $\langle \dots \leq (\text{norm} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi) + \delta)^2 \rangle$ 
  by (smt (verit)  $\psi \varepsilon$  complex-of-real-mono norm-triangle-ineq4 norm-triangle-sub pos2 power-strict-mono)
also have  $\langle \dots = (\text{norm} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi))^2 + \delta^2 + 2 * \text{norm} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi) * \delta \rangle$ 
  by (simp add: power2-sum)
also have  $\langle \dots \leq (\text{norm} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi))^2 + \delta^2 + 2 * \text{norm} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi) * \delta \rangle$ 
  apply (rule complex-of-real-mono-iff[THEN iffD2])
  by (smt (z3)  $\langle 0 < \delta \rangle \langle \text{norm } \psi = 1 \rangle$  more-arith-simps(11) mult-less-cancel-right-disj norm-cblinfun one-power2 power2-eq-square)
also have  $\langle \dots \leq (\text{norm} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi))^2 + \varepsilon \rangle$ 
  apply (rule complex-of-real-mono-iff[THEN iffD2])
  using  $\delta$ -small by auto
also have  $\langle \dots = ((\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi) \cdot_{\mathcal{C}} (\text{sqrt-op} (\text{abs-op } t) *_{\mathcal{V}} \psi)) + \varepsilon \rangle$ 
  by (simp add: cdot-square-norm)
also have  $\langle \dots = (\psi \cdot_{\mathcal{C}} (\text{abs-op } t *_{\mathcal{V}} \psi)) + \varepsilon \rangle$ 
  by (simp add: positive-selfadjointI[unfolded selfadjoint-def] flip: cinner-adj-right cblinfun-apply-cblinfun-compose)
also have  $\langle \dots \leq \text{trace-norm } t + \varepsilon \rangle$ 
  using  $\langle \text{norm } \psi = 1 \rangle$   $\langle \text{trace-class } t \rangle$  by (auto simp add: trace-norm-geq-cinner-abs-op)
finally show  $\langle \text{norm } t \leq \text{trace-norm } t + \varepsilon \rangle$ 
  using complex-of-real-mono-iff by blast
qed
qed

lemma clinear-from-trace-class[iff]:  $\langle \text{clinear from-trace-class} \rangle$ 
  apply (rule clinearI; transfer)
  by auto

lemma bounded-clinear-from-trace-class[bounded-clinear]:
   $\langle \text{bounded-clinear} (\text{from-trace-class} :: ('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{trace-class} \Rightarrow -) \rangle$ 
proof (cases  $\langle \text{class.not-singleton TYPE('a) \rangle$ )
  case True
  show ?thesis
  apply (rule bounded-clinearI[where K=1]; transfer)
  by (auto intro!: norm-leq-trace-norm[internalize-sort 'a] chilbert-space-axioms True)
next

```

```

case False
then have zero:  $\langle A = 0 \rangle$  for  $A :: \langle 'a \Rightarrow_{CL} 'b \rangle$ 
  by (rule not-not-singleton-cblinfun-zero)
show ?thesis
  apply (rule bounded-clinearI[where K=1])
  by (subst zero, simp)+
qed

```

```

instantiation trace-class :: (chilbert-space, chilbert-space) order begin
lift-definition less-eq-trace-class ::  $\langle ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \Rightarrow \text{bool} \rangle$  is
  less-eq.
lift-definition less-trace-class ::  $\langle ('a, 'b) \text{ trace-class} \Rightarrow ('a, 'b) \text{ trace-class} \Rightarrow \text{bool} \rangle$  is
  less.
instance
  apply intro-classes
  apply (auto simp add: less-eq-trace-class.rep-eq less-trace-class.rep-eq)
  by (simp add: from-trace-class-inject)
end

```

```

lift-definition compose-tcl ::  $\langle ('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{ trace-class} \Rightarrow ('c::\text{chilbert-space} \Rightarrow_{CL} 'a) \Rightarrow ('c, 'b) \text{ trace-class} \rangle$  is
 $\langle \text{cblinfun-compose} :: 'a \Rightarrow_{CL} 'b \Rightarrow 'c \Rightarrow_{CL} 'a \Rightarrow 'c \Rightarrow_{CL} 'b \rangle$ 
  by (simp add: trace-class-comp-left)

```

```

lift-definition compose-tcr ::  $\langle ('a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space}) \Rightarrow ('c::\text{chilbert-space}, 'a) \text{ trace-class} \Rightarrow ('c, 'b) \text{ trace-class} \rangle$  is
 $\langle \text{cblinfun-compose} :: 'a \Rightarrow_{CL} 'b \Rightarrow 'c \Rightarrow_{CL} 'a \Rightarrow 'c \Rightarrow_{CL} 'b \rangle$ 
  by (simp add: trace-class-comp-right)

```

```

lemma norm-compose-tcl:  $\langle \text{norm} (\text{compose-tcl } a \ b) \leq \text{norm } a * \text{norm } b \rangle$ 
  by (auto intro!: trace-norm-comp-left simp: norm-trace-class.rep-eq compose-tcl.rep-eq)

```

```

lemma norm-compose-tcr:  $\langle \text{norm} (\text{compose-tcr } a \ b) \leq \text{norm } a * \text{norm } b \rangle$ 
  by (auto intro!: trace-norm-comp-right simp: norm-trace-class.rep-eq compose-tcr.rep-eq)

```

```

interpretation compose-tcl: bounded-cbilinear compose-tcl
proof (intro bounded-cbilinear.intro exI[of - 1] allI)
  fix a a' ::  $\langle ('a, 'b) \text{ trace-class} \rangle$  and b b' ::  $\langle 'c \Rightarrow_{CL} 'a \rangle$  and r :: complex
  show  $\langle \text{compose-tcl } (a + a') \ b = \text{compose-tcl } a \ b + \text{compose-tcl } a' \ b \rangle$ 
    apply transfer
    by (simp add: cblinfun-compose-add-left)
  show  $\langle \text{compose-tcl } a \ (b + b') = \text{compose-tcl } a \ b + \text{compose-tcl } a \ b' \rangle$ 
    apply transfer
    by (simp add: cblinfun-compose-add-right)
  show  $\langle \text{compose-tcl } (r *_{\mathbb{C}} a) \ b = r *_{\mathbb{C}} \text{compose-tcl } a \ b \rangle$ 
    apply transfer
    by simp

```

show $\langle \text{compose-tcl } a (r *_C b) = r *_C \text{ compose-tcl } a b \rangle$
apply *transfer*
by *simp*
show $\langle \text{norm } (\text{compose-tcl } a b) \leq \text{norm } a * \text{norm } b * 1 \rangle$
by (*simp add: norm-compose-tcl*)
qed

interpretation *compose-tcr: bounded-cbilinear compose-tcr*

proof (*intro bounded-cbilinear.intro exI[of - 1] allI*)
fix $a a' :: \langle 'a \Rightarrow_{CL} 'b \rangle$ **and** $b b' :: \langle ('c, 'a) \text{ trace-class} \rangle$ **and** $r :: \text{complex}$
show $\langle \text{compose-tcr } (a + a') b = \text{compose-tcr } a b + \text{compose-tcr } a' b \rangle$
apply *transfer*
by (*simp add: cblinfun-compose-add-left*)
show $\langle \text{compose-tcr } a (b + b') = \text{compose-tcr } a b + \text{compose-tcr } a b' \rangle$
apply *transfer*
by (*simp add: cblinfun-compose-add-right*)
show $\langle \text{compose-tcr } (r *_C a) b = r *_C \text{ compose-tcr } a b \rangle$
apply *transfer*
by *simp*
show $\langle \text{compose-tcr } a (r *_C b) = r *_C \text{ compose-tcr } a b \rangle$
apply *transfer*
by *simp*
show $\langle \text{norm } (\text{compose-tcr } a b) \leq \text{norm } a * \text{norm } b * 1 \rangle$
by (*simp add: norm-compose-tcr*)
qed

lemma *trace-norm-sandwich*: $\langle \text{trace-norm } (\text{sandwich } e t) \leq (\text{norm } e)^{\wedge 2} * \text{trace-norm } t \rangle$ **if** $\langle \text{trace-class } t \rangle$

apply (*simp add: sandwich-apply*)
by (*smt (z3) Groups.mult-ac(2) more-arith-simps(11) mult-left-mono norm-adj norm-ge-zero power2-eq-square that trace-class-comp-right trace-norm-comp-left trace-norm-comp-right*)

lemma *trace-class-sandwich*: $\langle \text{trace-class } b \implies \text{trace-class } (\text{sandwich } a b) \rangle$
by (*simp add: sandwich-apply trace-class-comp-right trace-class-comp-left*)

definition $\langle \text{sandwich-tc } e t = \text{compose-tcl } (\text{compose-tcr } e t) (e*) \rangle$

lemma *sandwich-tc-transfer[transfer-rule]*:

includes *lifting-syntax*
shows $\langle ((=) \implies \text{cr-trace-class} \implies \text{cr-trace-class}) (\lambda e. (*_V) (\text{sandwich } e)) \text{ sandwich-tc} \rangle$
by (*auto intro!: rel-funI simp: sandwich-tc-def cr-trace-class-def compose-tcl.rep-eq compose-tcr.rep-eq sandwich-apply*)

lemma *from-trace-class-sandwich-tc*:

$\langle \text{from-trace-class } (\text{sandwich-tc } e t) = \text{sandwich } e (\text{from-trace-class } t) \rangle$
apply *transfer*
by (*rule sandwich-apply*)

lemma *norm-sandwich-tc*: $\langle \text{norm } (\text{sandwich-tc } e t) \leq (\text{norm } e)^{\wedge 2} * \text{norm } t \rangle$

by (simp add: norm-trace-class.rep-eq from-trace-class-sandwich-tc trace-norm-sandwich)

lemma sandwich-tc-pos: $\langle \text{sandwich-tc } e \ t \geq 0 \rangle$ **if** $\langle t \geq 0 \rangle$
using that **apply** (transfer fixing: e)
by (simp add: sandwich-pos)

lemma sandwich-tc-scaleC-right: $\langle \text{sandwich-tc } e \ (c *_{\mathcal{C}} t) = c *_{\mathcal{C}} \text{sandwich-tc } e \ t \rangle$
apply (transfer fixing: e c)
by (simp add: cblinfun.scaleC-right)

lemma sandwich-tc-plus: $\langle \text{sandwich-tc } e \ (t + u) = \text{sandwich-tc } e \ t + \text{sandwich-tc } e \ u \rangle$
by (simp add: sandwich-tc-def compose-tcr.add-right compose-tcl.add-left)

lemma sandwich-tc-minus: $\langle \text{sandwich-tc } e \ (t - u) = \text{sandwich-tc } e \ t - \text{sandwich-tc } e \ u \rangle$
by (simp add: sandwich-tc-def compose-tcr.diff-right compose-tcl.diff-left)

lemma sandwich-tc-uminus-right: $\langle \text{sandwich-tc } e \ (-t) = - \text{sandwich-tc } e \ t \rangle$
by (metis (no-types, lifting) add.right-inverse arith-simps(50) diff-0 group-cancel.sub1 sandwich-tc-minus)

lemma trace-comp-pos:

fixes $a \ b :: \langle 'a :: \text{chilbert-space} \Rightarrow_{\mathcal{CL}} 'a \rangle$

assumes $\langle \text{trace-class } b \rangle$

assumes $\langle a \geq 0 \rangle$ **and** $\langle b \geq 0 \rangle$

shows $\langle \text{trace } (a \ o_{\mathcal{CL}} \ b) \geq 0 \rangle$

proof –

obtain $c :: \langle 'a \Rightarrow_{\mathcal{CL}} 'a \rangle$ **where** $\langle a = c * o_{\mathcal{CL}} \ c \rangle$

by (metis assms(2) positive-selfadjointI sqrt-op-pos sqrt-op-square selfadjoint-def)

then have $\langle \text{trace } (a \ o_{\mathcal{CL}} \ b) = \text{trace } (\text{sandwich } c \ b) \rangle$

by (simp add: sandwich-apply assms(1) cblinfun-assoc-left(1) circularity-of-trace trace-class-comp-right)

also have $\langle \dots \geq 0 \rangle$

by (auto intro: trace-pos sandwich-pos assms)

finally show ?thesis

by –

qed

lemma trace-norm-one-dim: $\langle \text{trace-norm } x = \text{cmod } (\text{one-dim-iso } x) \rangle$

apply (rule of-real-eq-iff[**where** 'a=complex, **THEN** iffD1])

apply (simp add: abs-op-one-dim flip: trace-abs-op)

by (simp add: abs-complex-def)

lemma trace-norm-bounded:

fixes $A \ B :: \langle 'a :: \text{chilbert-space} \Rightarrow_{\mathcal{CL}} 'a \rangle$

assumes $\langle A \geq 0 \rangle$ **and** $\langle \text{trace-class } B \rangle$

assumes $\langle A \leq B \rangle$

shows $\langle \text{trace-class } A \rangle$

proof –
have $\langle (\lambda x. x \cdot_C (B *_V x)) \text{ abs-summable-on some-chilbert-basis} \rangle$
by $\langle \text{metis assms}(2) \text{ is-onb-some-chilbert-basis summable-on-iff-abs-summable-on-complex trace-exists} \rangle$
then have $\langle (\lambda x. x \cdot_C (A *_V x)) \text{ abs-summable-on some-chilbert-basis} \rangle$
apply $\langle \text{rule abs-summable-on-comparison-test} \rangle$
using $\langle A \geq 0 \rangle \langle A \leq B \rangle$
by $\langle \text{auto intro!; cmod-mono cinner-pos-if-pos simp: abs-op-id-on-pos less-eq-cblinfun-def} \rangle$
then show $\langle ?thesis \rangle$
by $\langle \text{auto intro!; trace-classI}[OF \text{ is-onb-some-chilbert-basis}] \text{ simp: abs-op-id-on-pos} \langle A \geq 0 \rangle \rangle$
qed

lemma *trace-norm-cblinfun-mono*:
fixes $A B :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle A \geq 0 \rangle$ **and** $\langle \text{trace-class } B \rangle$
assumes $\langle A \leq B \rangle$
shows $\langle \text{trace-norm } A \leq \text{trace-norm } B \rangle$

proof –
from *assms* **have** $\langle \text{trace-class } A \rangle$
by $\langle \text{rule trace-norm-bounded} \rangle$
from $\langle A \leq B \rangle$ **and** $\langle A \geq 0 \rangle$
have $\langle B \geq 0 \rangle$
by *simp*
have $\langle \text{cmod } (x \cdot_C (\text{abs-op } A *_V x)) \leq \text{cmod } (x \cdot_C (\text{abs-op } B *_V x)) \rangle$ **for** x
using $\langle A \leq B \rangle$
unfolding *less-eq-cblinfun-def*
using $\langle A \geq 0 \rangle \langle B \geq 0 \rangle$
by $\langle \text{auto intro!; cmod-mono cinner-pos-if-pos simp: abs-op-id-on-pos} \rangle$
then show $\langle \text{trace-norm } A \leq \text{trace-norm } B \rangle$
using $\langle \text{trace-class } A \rangle \langle \text{trace-class } B \rangle$
by $\langle \text{auto intro!; infsum-mono simp add: trace-norm-def trace-class-iff-summable}[OF \text{ is-onb-some-chilbert-basis}] \rangle$
qed

lemma *norm-cblinfun-mono-trace-class*:
fixes $A B :: \langle 'a::\text{chilbert-space}, 'a \rangle \text{ trace-class} \rangle$
assumes $\langle A \geq 0 \rangle$
assumes $\langle A \leq B \rangle$
shows $\langle \text{norm } A \leq \text{norm } B \rangle$
using *assms*
apply *transfer*
apply $\langle \text{rule trace-norm-cblinfun-mono} \rangle$
by *auto*

lemma *trace-norm-butterfly*: $\langle \text{trace-norm } (\text{butterfly } a \ b) = (\text{norm } a) * (\text{norm } b) \rangle$
for $a \ b :: \langle - :: \text{chilbert-space} \rangle$
proof –

```

have ⟨trace-norm (butterfly a b) = trace (abs-op (butterfly a b))⟩
  by (simp flip: trace-abs-op)
also have ⟨... = (norm a / norm b) * trace (selfbutter b)⟩
  by (simp add: abs-op-butterfly scaleR-scaleC trace-scaleC del: trace-abs-op)
also have ⟨... = (norm a / norm b) * trace ((vector-to-cblinfun b :: complex ⇒CL -)* oCL
vector-to-cblinfun b)⟩
  apply (subst butterfly-def)
  apply (subst circularity-of-trace)
  by simp-all
also have ⟨... = (norm a / norm b) * (b ·C b)⟩
  by simp
also have ⟨... = (norm a) * (norm b)⟩
  by (simp add: cdot-square-norm power2-eq-square)
finally show ?thesis
  using of-real-eq-iff by blast
qed

```

lemma *from-trace-class-sum*:

```

shows ⟨from-trace-class (∑ x∈M. f x) = (∑ x∈M. from-trace-class (f x))⟩
  apply (induction M rule:infinite-finite-induct)
  by (simp-all add: plus-trace-class.rep-eq)

```

lemma *has-sum-mono-neutral-traceclass*:

```

fixes f :: 'a ⇒ ('b::hilbert-space, 'b) trace-class
assumes ⟨f has-sum a⟩ and ⟨g has-sum b⟩ B
assumes ⟨∧x. x ∈ A∩B ⇒ f x ≤ g x⟩
assumes ⟨∧x. x ∈ A-B ⇒ f x ≤ 0⟩
assumes ⟨∧x. x ∈ B-A ⇒ g x ≥ 0⟩
shows a ≤ b

```

proof –

```

from assms(1)
have ⟨(λx. from-trace-class (f x)) has-sum from-trace-class a⟩ A
  apply (rule Infinite-Sum.has-sum-bounded-linear[rotated])
  by (intro bounded-clinear-from-trace-class bounded-clinear.bounded-linear)
moreover
from assms(2)
have ⟨(λx. from-trace-class (g x)) has-sum from-trace-class b⟩ B
  apply (rule Infinite-Sum.has-sum-bounded-linear[rotated])
  by (intro bounded-clinear-from-trace-class bounded-clinear.bounded-linear)
ultimately have ⟨from-trace-class a ≤ from-trace-class b⟩
  apply (rule has-sum-mono-neutral-cblinfun)
  using assms by (auto simp: less-eq-trace-class.rep-eq)
then show ?thesis
  by (auto simp: less-eq-trace-class.rep-eq)

```

qed

lemma *has-sum-mono-traceclass*:

fixes $f :: 'a \Rightarrow ('b::\text{hilbert-space}, 'b) \text{ trace-class}$
assumes $(f \text{ has-sum } x) A$ **and** $(g \text{ has-sum } y) A$
assumes $\langle \bigwedge x. x \in A \implies f x \leq g x \rangle$
shows $x \leq y$
using $\text{assms has-sum-mono-neutral-traceclass}$ **by force**

lemma $\text{infsum-mono-traceclass}$:
fixes $f :: 'a \Rightarrow ('b::\text{hilbert-space}, 'b) \text{ trace-class}$
assumes $f \text{ summable-on } A$ **and** $g \text{ summable-on } A$
assumes $\langle \bigwedge x. x \in A \implies f x \leq g x \rangle$
shows $\text{infsum } f A \leq \text{infsum } g A$
by $(\text{meson assms has-sum-infsum has-sum-mono-traceclass})$

lemma $\text{infsum-mono-neutral-traceclass}$:
fixes $f :: 'a \Rightarrow ('b::\text{hilbert-space}, 'b) \text{ trace-class}$
assumes $f \text{ summable-on } A$ **and** $g \text{ summable-on } B$
assumes $\langle \bigwedge x. x \in A \cap B \implies f x \leq g x \rangle$
assumes $\langle \bigwedge x. x \in A - B \implies f x \leq 0 \rangle$
assumes $\langle \bigwedge x. x \in B - A \implies g x \geq 0 \rangle$
shows $\text{infsum } f A \leq \text{infsum } g B$
using $\text{assms}(1) \text{ assms}(2) \text{ assms}(3) \text{ assms}(4) \text{ assms}(5) \text{ has-sum-mono-neutral-traceclass summable-iff-has-sum-infsum}$
by blast

instance $\text{trace-class} :: (\text{hilbert-space}, \text{hilbert-space}) \text{ ordered-complex-vector}$
apply $(\text{intro-classes}; \text{transfer})$
by $(\text{auto intro!}: \text{scaleC-left-mono scaleC-right-mono})$

lemma $\text{Abs-trace-class-geq0I}$: $\langle 0 \leq \text{Abs-trace-class } t \rangle$ **if** $\langle \text{trace-class } t \rangle$ **and** $\langle t \geq 0 \rangle$
using that by $(\text{simp add}: \text{zero-trace-class.abs-eq less-eq-trace-class.abs-eq eq-onp-def})$

lift-definition $\text{tc-compose} :: \langle ('b::\text{hilbert-space}, 'c::\text{hilbert-space}) \text{ trace-class} \implies ('a::\text{hilbert-space}, 'b) \text{ trace-class} \implies ('a, 'c) \text{ trace-class} \rangle$ **is**
 cblinfun-compose
by $(\text{simp add}: \text{trace-class-comp-left})$

lemma norm-tc-compose :
 $\langle \text{norm } (\text{tc-compose } a b) \leq \text{norm } a * \text{norm } b \rangle$
proof transfer
fix $a :: \langle 'c \Rightarrow_{CL} 'b \rangle$ **and** $b :: \langle 'a \Rightarrow_{CL} 'c \rangle$
assume $\langle a \in \text{Collect trace-class} \rangle$ **and** $\text{tc-b}: \langle b \in \text{Collect trace-class} \rangle$
then have $\langle \text{trace-norm } (a \circ_{CL} b) \leq \text{trace-norm } a * \text{norm } b \rangle$
by $(\text{simp add}: \text{trace-norm-comp-left})$
also have $\langle \dots \leq \text{trace-norm } a * \text{trace-norm } b \rangle$
using tc-b **by** $(\text{auto intro!}: \text{mult-left-mono norm-leq-trace-norm})$
finally show $\langle \text{trace-norm } (a \circ_{CL} b) \leq \text{trace-norm } a * \text{trace-norm } b \rangle$
by $-$
qed

lift-definition *trace-tc* :: $\langle ('a::\text{chilbert-space}, 'a) \text{ trace-class} \Rightarrow \text{complex} \rangle$ **is** *trace*.

lemma *trace-tc-plus*: $\langle \text{trace-tc } (a + b) = \text{trace-tc } a + \text{trace-tc } b \rangle$
apply transfer by (*simp add: trace-plus*)

lemma *trace-tc-scaleC*: $\langle \text{trace-tc } (c *_C a) = c *_C \text{trace-tc } a \rangle$
apply transfer by (*simp add: trace-scaleC*)

lemma *trace-tc-norm*: $\langle \text{norm } (\text{trace-tc } a) \leq \text{norm } a \rangle$
apply transfer by *auto*

lemma *bounded-clinear-trace-tc*[*bounded-clinear, simp*]: $\langle \text{bounded-clinear } \text{trace-tc} \rangle$
apply (*rule bounded-clinearI*[**where** $K=1$])
by (*auto simp: trace-tc-scaleC trace-tc-plus intro!: trace-tc-norm*)

lemma *norm-tc-pos*: $\langle \text{norm } A = \text{trace-tc } A \rangle$ **if** $\langle A \geq 0 \rangle$
using that apply transfer by (*simp add: trace-norm-pos*)

lemma *norm-tc-pos-Re*: $\langle \text{norm } A = \text{Re } (\text{trace-tc } A) \rangle$ **if** $\langle A \geq 0 \rangle$
using *norm-tc-pos*[*OF that*]
by (*metis Re-complex-of-real*)

lemma *from-trace-class-pos*: $\langle \text{from-trace-class } A \geq 0 \longleftrightarrow A \geq 0 \rangle$
by (*simp add: less-eq-trace-class.rep-eq*)

lemma *infsum-tc-norm-bounded-abs-summable*:

fixes $A :: \langle 'a \Rightarrow ('b::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{ trace-class} \rangle$

assumes *pos*: $\langle \bigwedge x. x \in M \Longrightarrow A x \geq 0 \rangle$

assumes *bound-B*: $\langle \bigwedge F. \text{finite } F \Longrightarrow F \subseteq M \Longrightarrow \text{norm } (\sum_{x \in F} A x) \leq B \rangle$

shows $\langle A \text{ abs-summable-on } M \rangle$

proof –

have $\langle (\sum_{x \in F} \text{norm } (A x)) = \text{norm } (\sum_{x \in F} A x) \rangle$ **if** $\langle F \subseteq M \rangle$ **for** F

proof –

have $\langle \text{complex-of-real } (\sum_{x \in F} \text{norm } (A x)) = (\sum_{x \in F} \text{complex-of-real } (\text{trace-norm } (\text{from-trace-class } (A x)))) \rangle$

by (*simp add: norm-trace-class.rep-eq trace-norm-pos*)

also have $\langle \dots = (\sum_{x \in F} \text{trace } (\text{from-trace-class } (A x))) \rangle$

using that pos by (*auto intro!: sum.cong simp add: trace-norm-pos less-eq-trace-class.rep-eq*)

also have $\langle \dots = \text{trace } (\text{from-trace-class } (\sum_{x \in F} A x)) \rangle$

by (*simp add: from-trace-class-sum trace-sum*)

also have $\langle \dots = \text{norm } (\sum_{x \in F} A x) \rangle$

by (*smt (verit, ccfv-threshold) calculation norm-of-real norm-trace-class.rep-eq sum-norm-le trace-leq-trace-norm*)

finally show *?thesis*

using of-real-eq-iff by *blast*

qed

with *bound-B* **have** *bound-B'*: $\langle (\sum_{x \in F} \text{norm } (A x)) \leq B \rangle$ **if** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq M \rangle$ **for** F

by (*metis that(1) that(2)*)

```

then show ⟨A abs-summable-on M⟩
  apply (rule-tac nonneg-bdd-above-summable-on)
  by (auto intro!: bdd-aboveI)
qed

lemma trace-norm-uminus[simp]: ⟨trace-norm (-a) = trace-norm a⟩
  by (metis abs-op-uminus of-real-eq-iff trace-abs-op)

lemma trace-norm-triangle-minus:
  fixes a b :: ⟨'a::hilbert-space ⇒CL 'b::hilbert-space⟩
  assumes [simp]: ⟨trace-class a⟩ ⟨trace-class b⟩
  shows ⟨trace-norm (a - b) ≤ trace-norm a + trace-norm b⟩
  using trace-norm-triangle[of a ⟨-b⟩]
  by auto

lemma trace-norm-abs-op[simp]: ⟨trace-norm (abs-op t) = trace-norm t⟩
  by (simp add: trace-norm-def)

lemma
  fixes t :: ⟨'a ⇒CL 'a::hilbert-space⟩
  shows cblinfun-decomp-4pos: ⟨
    ∃ t1 t2 t3 t4.
      t = t1 - t2 + i *C t3 - i *C t4
      ∧ t1 ≥ 0 ∧ t2 ≥ 0 ∧ t3 ≥ 0 ∧ t4 ≥ 0⟩ (is ?thesis1)
  and trace-class-decomp-4pos: ⟨trace-class t ⇒
    ∃ t1 t2 t3 t4.
      t = t1 - t2 + i *C t3 - i *C t4
      ∧ trace-class t1 ∧ trace-class t2 ∧ trace-class t3 ∧ trace-class t4
      ∧ trace-norm t1 ≤ trace-norm t ∧ trace-norm t2 ≤ trace-norm t ∧ trace-norm t3
      ≤ trace-norm t ∧ trace-norm t4 ≤ trace-norm t
      ∧ t1 ≥ 0 ∧ t2 ≥ 0 ∧ t3 ≥ 0 ∧ t4 ≥ 0⟩ (is ⟨- ⇒ ?thesis2⟩)

proof -
  define th ta where ⟨th = (1/2) *C (t + t*)⟩ and ⟨ta = (-i/2) *C (t - t*)⟩
  have th-herm: ⟨th* = th⟩
    by (simp add: adj-plus th-def)
  have ⟨ta* = ta⟩
    by (simp add: adj-minus ta-def scaleC-diff-right adj-uminus)
  have ⟨t = th + i *C ta⟩
    by (smt (verit, ccfv-SIG) add.commute add.inverse-inverse complex-i-mult-minus com-
      plex-vector.vector-space-assms(1) complex-vector.vector-space-assms(3) diff-add-cancel group-cancel.add2
      i-squared scaleC-half-double ta-def th-def times-divide-eq-right)
  define t1 t2 where ⟨t1 = (abs-op th + th) /R 2⟩ and ⟨t2 = (abs-op th - th) /R 2⟩
  have ⟨t1 ≥ 0⟩
    using abs-op-geq-req[unfolded selfadjoint-def, OF ⟨th* = th⟩] ordered-field-class.sign-simps(15)
    by (fastforce simp add: t1-def intro!: scaleR-nonneg-nonneg)
  have ⟨t2 ≥ 0⟩
    using abs-op-geq[unfolded selfadjoint-def, OF ⟨th* = th⟩] ordered-field-class.sign-simps(15)
    by (fastforce simp add: t2-def intro!: scaleR-nonneg-nonneg)
  have ⟨th = t1 - t2⟩

```

```

apply (simp add: t1-def t2-def)
  by (metis (no-types, opaque-lifting) Extra-Ordered-Fields.sign-simps(8) diff-add-cancel or-
    dered-field-class.sign-simps(2) ordered-field-class.sign-simps(27) scaleR-half-double)
define t3 t4 where ⟨t3 = (abs-op ta + ta) /R 2⟩ and ⟨t4 = (abs-op ta - ta) /R 2⟩
have ⟨t3 ≥ 0⟩
  using abs-op-geq-neq[unfolded selfadjoint-def, OF ⟨ta* = ta⟩] ordered-field-class.sign-simps(15)
  by (fastforce simp add: t3-def intro!: scaleR-nonneg-nonneg)
have ⟨t4 ≥ 0⟩
  using abs-op-geq[unfolded selfadjoint-def, OF ⟨ta* = ta⟩] ordered-field-class.sign-simps(15)
  by (fastforce simp add: t4-def intro!: scaleR-nonneg-nonneg)
have ⟨ta = t3 - t4⟩
  apply (simp add: t3-def t4-def)
  by (metis (no-types, opaque-lifting) Extra-Ordered-Fields.sign-simps(8) diff-add-cancel or-
    dered-field-class.sign-simps(2) ordered-field-class.sign-simps(27) scaleR-half-double)
have decomp: ⟨t = t1 - t2 + i *C t3 - i *C t4⟩
  by (simp add: ⟨t = th + i *C ta⟩ ⟨th = t1 - t2⟩ ⟨ta = t3 - t4⟩ scaleC-diff-right)
from decomp ⟨t1 ≥ 0⟩ ⟨t2 ≥ 0⟩ ⟨t3 ≥ 0⟩ ⟨t4 ≥ 0⟩
show ?thesis1
  by auto
show ?thesis2 if ⟨trace-class t⟩
proof -
  have ⟨trace-class th⟩ ⟨trace-class ta⟩
    by (auto simp add: th-def ta-def
      intro!: ⟨trace-class t⟩ trace-class-scaleC trace-class-plus trace-class-minus trace-class-uminus
      trace-class-adj)
  then have tc: ⟨trace-class t1⟩ ⟨trace-class t2⟩ ⟨trace-class t3⟩ ⟨trace-class t4⟩
    by (auto simp add: t1-def t2-def t3-def t4-def scaleR-scaleC intro!: trace-class-scaleC)
  have tn-th: ⟨trace-norm th ≤ trace-norm t⟩
    using trace-norm-triangle[of t ⟨t*⟩]
    by (auto simp add: that th-def trace-norm-scaleC)
  have tn-ta: ⟨trace-norm ta ≤ trace-norm t⟩
    using trace-norm-triangle-minus[of t ⟨t*⟩]
    by (auto simp add: that ta-def trace-norm-scaleC)
  have tn1: ⟨trace-norm t1 ≤ trace-norm t⟩
    using trace-norm-triangle[of ⟨abs-op th⟩ th] tn-th
    by (auto simp add: ⟨trace-class th⟩ t1-def trace-norm-scaleC scaleR-scaleC)
  have tn2: ⟨trace-norm t2 ≤ trace-norm t⟩
    using trace-norm-triangle-minus[of ⟨abs-op th⟩ th] tn-th
    by (auto simp add: ⟨trace-class th⟩ t2-def trace-norm-scaleC scaleR-scaleC)
  have tn3: ⟨trace-norm t3 ≤ trace-norm t⟩
    using trace-norm-triangle[of ⟨abs-op ta⟩ ta] tn-ta
    by (auto simp add: ⟨trace-class ta⟩ t3-def trace-norm-scaleC scaleR-scaleC)
  have tn4: ⟨trace-norm t4 ≤ trace-norm t⟩
    using trace-norm-triangle-minus[of ⟨abs-op ta⟩ ta] tn-ta
    by (auto simp add: ⟨trace-class ta⟩ t4-def trace-norm-scaleC scaleR-scaleC)
from decomp tc ⟨t1 ≥ 0⟩ ⟨t2 ≥ 0⟩ ⟨t3 ≥ 0⟩ ⟨t4 ≥ 0⟩ tn1 tn2 tn3 tn4
show ?thesis2
  by auto
qed

```

qed

lemma *trace-class-decomp-4pos'*:

fixes $t :: \langle 'a :: \text{chilbert-space}, 'a \rangle \text{trace-class}$

shows $\langle \exists t1\ t2\ t3\ t4.$

$$t = t1 - t2 + i *_C t3 - i *_C t4$$

$$\wedge \text{norm } t1 \leq \text{norm } t \wedge \text{norm } t2 \leq \text{norm } t \wedge \text{norm } t3 \leq \text{norm } t \wedge \text{norm } t4 \leq \text{norm } t$$

t

$$\wedge t1 \geq 0 \wedge t2 \geq 0 \wedge t3 \geq 0 \wedge t4 \geq 0 \rangle$$

proof –

from *trace-class-decomp-4pos*[of $\langle \text{from-trace-class } t \rangle$, *OF trace-class-from-trace-class*]

obtain $t1'\ t2'\ t3'\ t4'$

where $*$: $\langle \text{from-trace-class } t = t1' - t2' + i *_C t3' - i *_C t4'$

$$\wedge \text{trace-class } t1' \wedge \text{trace-class } t2' \wedge \text{trace-class } t3' \wedge \text{trace-class } t4'$$

$$\wedge \text{trace-norm } t1' \leq \text{trace-norm } (\text{from-trace-class } t) \wedge \text{trace-norm } t2' \leq \text{trace-norm } (\text{from-trace-class } t)$$

$\wedge \text{trace-norm } t3' \leq \text{trace-norm } (\text{from-trace-class } t) \wedge \text{trace-norm } t4' \leq \text{trace-norm } (\text{from-trace-class } t)$

$$\wedge t1' \geq 0 \wedge t2' \geq 0 \wedge t3' \geq 0 \wedge t4' \geq 0 \rangle$$

by *auto*

then obtain $t1\ t2\ t3\ t4$ **where**

$t1234$: $\langle t1' = \text{from-trace-class } t1 \rangle \langle t2' = \text{from-trace-class } t2 \rangle \langle t3' = \text{from-trace-class } t3 \rangle \langle t4' = \text{from-trace-class } t4 \rangle$

by (*metis from-trace-class-cases mem-Collect-eq*)

with $*$ **have** $n1234$: $\langle \text{norm } t1 \leq \text{norm } t \rangle \langle \text{norm } t2 \leq \text{norm } t \rangle \langle \text{norm } t3 \leq \text{norm } t \rangle \langle \text{norm } t4 \leq \text{norm } t \rangle$

by (*metis norm-trace-class.rep-eq*) $+$

have $t\text{-decomp}$: $\langle t = t1 - t2 + i *_C t3 - i *_C t4 \rangle$

using $*$ **unfolding** $t1234$

by (*auto simp: from-trace-class-inject*

$$\text{simp flip: scaleC-trace-class.rep-eq plus-trace-class.rep-eq minus-trace-class.rep-eq}$$

have $pos1234$: $\langle t1 \geq 0 \rangle \langle t2 \geq 0 \rangle \langle t3 \geq 0 \rangle \langle t4 \geq 0 \rangle$

using $*$ **unfolding** $t1234$

by (*auto simp: less-eq-trace-class-def*)

from $t\text{-decomp } pos1234\ n1234$

show $?thesis$

by *blast*

qed

thm *bounded-clinear-trace-duality*

lemma *bounded-clinear-trace-duality'*: $\langle \text{trace-class } t \implies \text{bounded-clinear } (\lambda a. \text{trace } (a\ o_{CL}\ t)) \rangle$

for $t :: \langle - :: \text{chilbert-space} \Rightarrow_{CL} - :: \text{chilbert-space} \rangle$

apply (*rule bounded-clinearI*[**where** $K = \langle \text{trace-norm } t \rangle$])

apply (*auto simp add: cblinfun-compose-add-left trace-class-comp-right trace-plus trace-scaleC*)[2]

by (*metis circularity-of-trace order-trans trace-leq-trace-norm trace-norm-comp-right*)

lemma *infsum-nonneg-traceclass*:

fixes $f :: 'a \Rightarrow ('b :: \text{chilbert-space}, 'b) \text{trace-class}$

assumes $\bigwedge x. x \in M \implies 0 \leq f\ x$

shows $\text{infsum } f\ M \geq 0$

apply (*cases* $\langle f \text{ summable-on } M \rangle$)
apply (*subst* *infsun-0-simp*[*symmetric*])
apply (*rule* *infsun-mono-neutral-traceclass*)
using *assms* **by** (*auto simp: infsun-not-exists*)

lemma *sandwich-tc-compose*: $\langle \text{sandwich-tc } (A \text{ } o_{CL} \text{ } B) = \text{sandwich-tc } A \text{ } o \text{ } \text{sandwich-tc } B \rangle$
apply (*rule* *ext*)
apply (*rule* *from-trace-class-inject*[*THEN iffD1*])
apply (*transfer* *fixing: A B*)
by (*simp add: sandwich-compose*)

lemma *sandwich-tc-0-left*[*simp*]: $\langle \text{sandwich-tc } 0 = 0 \rangle$
by (*auto intro!: ext simp add: sandwich-tc-def compose-tcl.zero-left compose-tcr.zero-left*)

lemma *sandwich-tc-0-right*[*simp*]: $\langle \text{sandwich-tc } e \text{ } 0 = 0 \rangle$
by (*auto intro!: ext simp add: sandwich-tc-def compose-tcl.zero-left compose-tcr.zero-right*)

lemma *sandwich-tc-scaleC-left*: $\langle \text{sandwich-tc } (c \text{ } *_C \text{ } e) \text{ } t = (c \text{ } \text{mod } c) \hat{\text{2}} \text{ } *_C \text{ } \text{sandwich-tc } e \text{ } t \rangle$
apply (*rule* *from-trace-class-inject*[*THEN iffD1*])
by (*simp add: from-trace-class-sandwich-tc scaleC-trace-class.rep-eq sandwich-scaleC-left*)

lemma *sandwich-tc-scaleR-left*: $\langle \text{sandwich-tc } (r \text{ } *_R \text{ } e) \text{ } t = r \hat{\text{2}} \text{ } *_R \text{ } \text{sandwich-tc } e \text{ } t \rangle$
by (*simp add: scaleR-scaleC sandwich-tc-scaleC-left flip: of-real-power*)

lemma *bounded-cbilinear-tc-compose*: $\langle \text{bounded-cbilinear } \text{tc-compose} \rangle$
unfolding *bounded-cbilinear-def*
apply *transfer*
apply (*auto intro!: exI*[*of - 1*] *simp: cblinfun-compose-add-left cblinfun-compose-add-right*)
by (*meson norm-leq-trace-norm dual-order.trans mult-right-mono trace-norm-comp-right trace-norm-nneg*)

lemmas *bounded-clinear-tc-compose-left*[*bounded-clinear*] = *bounded-cbilinear.bounded-clinear-left*[*OF bounded-cbilinear-tc-compose*]

lemmas *bounded-clinear-tc-compose-right*[*bounded-clinear*] = *bounded-cbilinear.bounded-clinear-right*[*OF bounded-cbilinear-tc-compose*]

lift-definition *tc-butterfly* :: $\langle 'a::\text{chilbert-space} \Rightarrow 'b::\text{chilbert-space} \Rightarrow ('b, 'a) \text{ trace-class} \rangle$
is *butterfly*
by *simp*

lemma *norm-tc-butterfly*: $\langle \text{norm } (\text{tc-butterfly } \psi \text{ } \varphi) = \text{norm } \psi \text{ } * \text{norm } \varphi \rangle$
apply (*transfer* *fixing: $\psi \text{ } \varphi$*)
by (*simp add: trace-norm-butterfly*)

lemma *comp-tc-butterfly*[*simp*]: $\langle \text{tc-compose } (\text{tc-butterfly } a \text{ } b) \text{ } (\text{tc-butterfly } c \text{ } d) = (b \text{ } \cdot_C \text{ } c) \text{ } *_C \text{ } \text{tc-butterfly } a \text{ } d \rangle$
apply *transfer'*
by *simp*

lemma *tc-butterfly-pos*[*simp*]: $\langle 0 \leq \text{tc-butterfly } \psi \text{ } \psi \rangle$

apply *transfer*
by *simp*

lift-definition *rank1-tc* :: $\langle 'a::\text{chilbert-space}, 'b::\text{chilbert-space} \rangle \text{trace-class} \Rightarrow \text{bool}$ **is** *rank1*.
lift-definition *finite-rank-tc* :: $\langle 'a::\text{chilbert-space}, 'b::\text{chilbert-space} \rangle \text{trace-class} \Rightarrow \text{bool}$ **is** *finite-rank*.

lemma *finite-rank-tc-0[iff]*: $\langle \text{finite-rank-tc } 0 \rangle$
apply *transfer* **by** *simp*

lemma *finite-rank-tc-plus*: $\langle \text{finite-rank-tc } (a + b) \rangle$
if $\langle \text{finite-rank-tc } a \rangle$ **and** $\langle \text{finite-rank-tc } b \rangle$
using *that* **apply** *transfer*
by *simp*

lemma *finite-rank-tc-scale*: $\langle \text{finite-rank-tc } (c *_{\mathbb{C}} a) \rangle$ **if** $\langle \text{finite-rank-tc } a \rangle$
using *that* **apply** *transfer* **by** *simp*

lemma *csubspace-finite-rank-tc*: $\langle \text{csubspace } (\text{Collect } \text{finite-rank-tc}) \rangle$
apply *(rule complex-vector.subspaceI)*
by *(auto intro!: finite-rank-tc-plus finite-rank-tc-scale)*

lemma *rank1-trace-class*: $\langle \text{trace-class } a \rangle$ **if** $\langle \text{rank1 } a \rangle$
for $a\ b :: \langle 'a::\text{chilbert-space} \Rightarrow_{\mathbb{C}L} 'b::\text{chilbert-space} \rangle$
using *that* **by** *(auto intro!: simp: rank1-iff-butterfly)*

lemma *finite-rank-trace-class*: $\langle \text{trace-class } a \rangle$ **if** $\langle \text{finite-rank } a \rangle$
for $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{\mathbb{C}L} 'b::\text{chilbert-space} \rangle$

proof –

from $\langle \text{finite-rank } a \rangle$ **obtain** $F\ f$ **where** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq \text{Collect } \text{rank1} \rangle$
and $a\text{-def}$: $\langle a = (\sum_{x \in F}. f\ x *_{\mathbb{C}} x) \rangle$
by *(smt (verit, ccfv-threshold) complex-vector.span-explicit finite-rank-def mem-Collect-eq)*
then show $\langle \text{trace-class } a \rangle$
unfolding $a\text{-def}$
apply *induction*
by *(auto intro!: trace-class-plus trace-class-scaleC intro: rank1-trace-class)*

qed

lemma *trace-minus*:

assumes $\langle \text{trace-class } a \rangle$ $\langle \text{trace-class } b \rangle$
shows $\langle \text{trace } (a - b) = \text{trace } a - \text{trace } b \rangle$
by *(metis (no-types, lifting) add-uminus-conv-diff assms(1) assms(2) trace-class-uminus trace-plus trace-uminus)*

lemma *trace-cblinfun-mono*:

fixes $A\ B :: \langle 'a::\text{chilbert-space} \Rightarrow_{\mathbb{C}L} 'a \rangle$
assumes $\langle \text{trace-class } A \rangle$ **and** $\langle \text{trace-class } B \rangle$
assumes $\langle A \leq B \rangle$
shows $\langle \text{trace } A \leq \text{trace } B \rangle$

proof –
have $sumA$: $\langle (\lambda e. e \cdot_C (A *_V e)) \text{ summable-on some-chilbert-basis} \rangle$
by $(auto \text{ intro!}: \text{trace-exists } assms)$
moreover have $sumB$: $\langle (\lambda e. e \cdot_C (B *_V e)) \text{ summable-on some-chilbert-basis} \rangle$
by $(auto \text{ intro!}: \text{trace-exists } assms)$
moreover have $\langle x \cdot_C (A *_V x) \leq x \cdot_C (B *_V x) \rangle$ **for** x
using $assms(3)$ $\text{less-eq-cblinfun-def}$ **by** $blast$
ultimately have $\langle (\sum_{\infty e \in \text{some-chilbert-basis}. e \cdot_C (A *_V e)} \leq (\sum_{\infty e \in \text{some-chilbert-basis}. e \cdot_C (B *_V e)}) \rangle$
by $(rule \text{ infsum-mono-complex})$
then show $?thesis$
by $(metis \text{ assms}(1) \text{ assms}(2) \text{ assms}(3) \text{ diff-ge-0-iff-ge } \text{trace-minus } \text{trace-pos})$
qed

lemma trace-tc-mono :
assumes $\langle A \leq B \rangle$
shows $\langle \text{trace-tc } A \leq \text{trace-tc } B \rangle$
using $assms$
apply transfer
by $(simp \text{ add}: \text{trace-cblinfun-mono})$

lemma $\text{trace-tc-0}[simp]$: $\langle \text{trace-tc } 0 = 0 \rangle$
apply $\text{transfer}'$ **by** $simp$

lemma $\text{cspan-tc-transfer}[\text{transfer-rule}]$:
includes lifting-syntax
shows $\langle (\text{rel-set } \text{cr-trace-class} ==> \text{rel-set } \text{cr-trace-class}) \text{ cspan } \text{cspan} \rangle$

proof $(intro \text{ rel-funI } \text{rel-setI})$
fix $B :: \langle ('a \Rightarrow_{CL} 'b) \text{ set} \rangle$ **and** C
assume $\langle \text{rel-set } \text{cr-trace-class } B \ C \rangle$
then have BC : $\langle B = \text{from-trace-class } 'C \rangle$
by $(auto \text{ intro!}: \text{simp}: \text{cr-trace-class-def } \text{image-iff } \text{rel-set-def})$

show $\langle \exists t \in \text{cspan } C. \text{cr-trace-class } a \ t \rangle$ **if** $\langle a \in \text{cspan } B \rangle$ **for** a
proof –
from that obtain $F \ f$ **where** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq B \rangle$ **and** $a\text{-sum}$: $\langle a = (\sum_{x \in F}. f \ x *_C \ x) \rangle$
by $(auto \text{ simp}: \text{complex-vector.span-explicit})$
from $\langle F \subseteq B \rangle$
obtain $F' \ \text{where}$ $\langle F' \subseteq C \rangle$ **and** FF' : $\langle F = \text{from-trace-class } 'F' \rangle$
by $(auto \text{ elim!}: \text{subset-imageE } \text{simp}: BC)$
define t **where** $\langle t = (\sum_{x \in F'}. f \ (\text{from-trace-class } x) *_C \ x) \rangle$
have $\langle a = \text{from-trace-class } t \rangle$
by $(simp \text{ add}: a\text{-sum } t\text{-def } \text{from-trace-class-sum } \text{scaleC-trace-class.rep-eq } FF' \ \text{sum.reindex } o\text{-def } \text{from-trace-class-inject } \text{inj-on-def})$
moreover have $\langle t \in \text{cspan } C \rangle$
by $(metis \text{ (no-types, lifting) } \langle F' \subseteq C \rangle \text{complex-vector.span-clauses}(4) \text{complex-vector.span-sum } \text{complex-vector.span-superset } \text{subsetD } t\text{-def})$
ultimately show $\langle \exists t \in \text{cspan } C. \text{cr-trace-class } a \ t \rangle$
by $(auto \text{ simp}: \text{cr-trace-class-def})$

qed

show $\langle \exists a \in \text{cspan } B. \text{ cr-trace-class } a \rangle$ if $\langle t \in \text{cspan } C \rangle$ for t

proof –

from that obtain $F f$ where $\langle \text{finite } F \rangle$ and $\langle F \subseteq C \rangle$ and $t\text{-sum}: \langle t = (\sum x \in F. f x *_C x) \rangle$

by (auto simp: complex-vector.span-explicit)

define a where $\langle a = (\sum x \in F. f x *_C \text{ from-trace-class } x) \rangle$

then have $\langle a = \text{from-trace-class } t \rangle$

by (simp add: t-sum a-def from-trace-class-sum scaleC-trace-class.rep-eq)

moreover have $\langle a \in \text{cspan } B \rangle$

using BC $\langle F \subseteq C \rangle$

by (auto intro!: complex-vector.span-base complex-vector.span-sum complex-vector.span-scale simp: a-def)

ultimately show ?thesis

by (auto simp: cr-trace-class-def)

qed

qed

lemma finite-rank-tc-def': $\langle \text{finite-rank-tc } A \longleftrightarrow A \in \text{cspan } (\text{Collect rank1-tc}) \rangle$

apply transfer'

apply (auto simp: finite-rank-def)

apply (metis (no-types, lifting) Collect-cong rank1-trace-class)

by (metis (no-types, lifting) Collect-cong rank1-trace-class)

lemma tc-butterfly-add-left: $\langle \text{tc-butterfly } (a + a') b = \text{tc-butterfly } a b + \text{tc-butterfly } a' b \rangle$

apply transfer

by (rule butterfly-add-left)

lemma tc-butterfly-add-right: $\langle \text{tc-butterfly } a (b + b') = \text{tc-butterfly } a b + \text{tc-butterfly } a b' \rangle$

apply transfer

by (rule butterfly-add-right)

lemma tc-butterfly-sum-left: $\langle \text{tc-butterfly } (\sum i \in M. \psi i) \varphi = (\sum i \in M. \text{tc-butterfly } (\psi i) \varphi) \rangle$

apply transfer

by (rule butterfly-sum-left)

lemma tc-butterfly-sum-right: $\langle \text{tc-butterfly } \psi (\sum i \in M. \varphi i) = (\sum i \in M. \text{tc-butterfly } \psi (\varphi i)) \rangle$

apply transfer

by (rule butterfly-sum-right)

lemma tc-butterfly-scaleC-left[simp]: $\text{tc-butterfly } (c *_C \psi) \varphi = c *_C \text{tc-butterfly } \psi \varphi$

apply transfer by simp

lemma tc-butterfly-scaleC-right[simp]: $\text{tc-butterfly } \psi (c *_C \varphi) = \text{cnj } c *_C \text{tc-butterfly } \psi \varphi$

apply transfer by simp

lemma bounded-sesquilinear-tc-butterfly[iff]: $\langle \text{bounded-sesquilinear } (\lambda a b. \text{tc-butterfly } b a) \rangle$

by (auto intro!: bounded-sesquilinear.intro exI[of - 1])

simp: tc-butterfly-add-left tc-butterfly-add-right norm-tc-butterfly)

lemma *trace-norm-plus-orthogonal:*

assumes $\langle \text{trace-class } a \rangle$ **and** $\langle \text{trace-class } b \rangle$
assumes $\langle a * o_{CL} b = 0 \rangle$ **and** $\langle a o_{CL} b * = 0 \rangle$
shows $\langle \text{trace-norm } (a + b) = \text{trace-norm } a + \text{trace-norm } b \rangle$

proof –

have $\langle \text{trace-norm } (a + b) = \text{trace } (\text{abs-op } (a + b)) \rangle$
by *simp*
also have $\langle \dots = \text{trace } (\text{abs-op } a + \text{abs-op } b) \rangle$
by (*simp add: abs-op-plus-orthogonal assms*)
also have $\langle \dots = \text{trace } (\text{abs-op } a) + \text{trace } (\text{abs-op } b) \rangle$
by (*simp add: assms trace-plus*)
also have $\langle \dots = \text{trace-norm } a + \text{trace-norm } b \rangle$
by *simp*
finally show *?thesis*
using *of-real-eq-iff* **by** *blast*

qed

lemma *norm-tc-plus-orthogonal:*

assumes $\langle \text{tc-compose } (\text{adj-tc } a) b = 0 \rangle$ **and** $\langle \text{tc-compose } a (\text{adj-tc } b) = 0 \rangle$
shows $\langle \text{norm } (a + b) = \text{norm } a + \text{norm } b \rangle$
using *assms apply transfer*
by (*auto intro!: trace-norm-plus-orthogonal*)

lemma *trace-norm-sum-exchange:*

fixes $t :: \langle - \Rightarrow (-::\text{chilbert-space} \Rightarrow_{CL} -::\text{chilbert-space}) \rangle$
assumes $\langle \bigwedge i. i \in F \Rightarrow \text{trace-class } (t i) \rangle$
assumes $\langle \bigwedge i j. i \in F \Rightarrow j \in F \Rightarrow i \neq j \Rightarrow (t i) * o_{CL} t j = 0 \rangle$
assumes $\langle \bigwedge i j. i \in F \Rightarrow j \in F \Rightarrow i \neq j \Rightarrow t i o_{CL} (t j) * = 0 \rangle$
shows $\langle \text{trace-norm } (\sum_{i \in F} t i) = (\sum_{i \in F} \text{trace-norm } (t i)) \rangle$

proof (*insert assms, induction F rule:infinite-finite-induct*)

case (*infinite A*)

then show *?case*

by *simp*

next

case *empty*

show *?case*

by *simp*

next

case (*insert x F*)

have $\langle \text{trace-norm } (\sum_{i \in \text{insert } x F} t i) = \text{trace-norm } (t x + (\sum_{x \in F} t x)) \rangle$

by (*simp add: insert*)

also have $\langle \dots = \text{trace-norm } (t x) + \text{trace-norm } (\sum_{x \in F} t x) \rangle$

proof (*rule trace-norm-plus-orthogonal*)

show $\langle \text{trace-class } (t x) \rangle$

by (*simp add: insert.premis*)

```

show ⟨trace-class ( $\sum x \in F. t x$ )⟩
  by (simp add: trace-class-sum insert.prems)
show ⟨ $t x * o_{CL} (\sum x \in F. t x) = 0$ ⟩
  by (auto intro!: sum.neutral insert.prems simp: cblinfun-compose-sum-right sum-adj insert.hyps)
show ⟨ $t x o_{CL} (\sum x \in F. t x) * = 0$ ⟩
  by (auto intro!: sum.neutral insert.prems simp: cblinfun-compose-sum-right sum-adj insert.hyps)
qed
also have ⟨ $\dots = \text{trace-norm } (t x) + (\sum x \in F. \text{trace-norm } (t x))$ ⟩
  apply (subst insert.IH)
  by (simp-all add: insert.prems)
also have ⟨ $\dots = (\sum i \in \text{insert } x F. \text{trace-norm } (t i))$ ⟩
  by (simp add: insert)
finally show ?case
  by –
qed

```

lemma *norm-tc-sum-exchange*:

```

assumes ⟨ $\bigwedge i j. i \in F \implies j \in F \implies i \neq j \implies \text{tc-compose } (\text{adj-tc } (t i)) (t j) = 0$ ⟩
assumes ⟨ $\bigwedge i j. i \in F \implies j \in F \implies i \neq j \implies \text{tc-compose } (t i) (\text{adj-tc } (t j)) = 0$ ⟩
shows ⟨ $\text{norm } (\sum i \in F. t i) = (\sum i \in F. \text{norm } (t i))$ ⟩
using assms apply transfer
by (auto intro!: trace-norm-sum-exchange)

```

instantiation *trace-class* :: (*one-dim, one-dim*) *complex-inner* **begin**

lift-definition *cinner-trace-class* :: ⟨('a, 'b) *trace-class* \Rightarrow ('a, 'b) *trace-class* \Rightarrow *complex*⟩ **is** ⟨(\cdot_C)⟩.

instance

proof *intro-classes*

fix $x y z :: \langle ('a, 'b) \text{ trace-class} \rangle$

show ⟨ $x \cdot_C y = \text{cnj } y \cdot_C x$ ⟩

apply *transfer'*

by *simp*

show ⟨ $(x + y) \cdot_C z = x \cdot_C z + y \cdot_C z$ ⟩

apply *transfer'*

by (*simp add: cinner-simps*)

show ⟨ $r *_C x \cdot_C y = \text{cnj } r * (x \cdot_C y)$ ⟩ **for** r

apply (*transfer' fixing: r*)

using *cinner-simps* **by** *blast*

show ⟨ $0 \leq x \cdot_C x$ ⟩

apply *transfer'*

by *simp*

show ⟨ $(x \cdot_C x = 0) = (x = 0)$ ⟩

apply *transfer'*

by *auto*

show ⟨ $\text{norm } x = \text{sqrt } (\text{cmod } (x \cdot_C x))$ ⟩

proof *transfer'*

```

fix x :: ⟨'a ⇒CL 'b⟩
define c :: complex where ⟨c = one-dim-iso x⟩
then have xc: ⟨x = c *C 1⟩
  by simp
have ⟨trace-norm x = norm c⟩
  by (simp add: trace-norm-one-dim xc)
also have ⟨norm c = sqrt (cmod (x *C x))⟩
  by (metis inner-real-def norm-eq-sqrt-cinner norm-one norm-scaleC real-inner-1-right xc)
finally show ⟨trace-norm x = sqrt (cmod (x *C x)) ⟩
  by (simp add: cinner-cblinfun-def)
qed
qed
end

```

instantiation *trace-class* :: (one-dim, one-dim) one-dim **begin**

lift-definition *one-trace-class* :: ⟨('a, 'b) trace-class⟩ **is** 1

by auto

lift-definition *times-trace-class* :: ⟨('a, 'b) trace-class ⇒ ('a, 'b) trace-class ⇒ ('a, 'b) trace-class⟩
is ⟨(*)⟩

by auto

lift-definition *divide-trace-class* :: ⟨('a, 'b) trace-class ⇒ ('a, 'b) trace-class ⇒ ('a, 'b) trace-class⟩
is ⟨(/)⟩

by auto

lift-definition *inverse-trace-class* :: ⟨('a, 'b) trace-class ⇒ ('a, 'b) trace-class⟩ **is** ⟨Fields.inverse⟩
by auto

definition *canonical-basis-trace-class* :: ⟨('a, 'b) trace-class list⟩ **where** ⟨canonical-basis-trace-class = [1]⟩

definition *canonical-basis-length-trace-class* :: ⟨('a, 'b) trace-class itself ⇒ nat⟩ **where** ⟨canonical-basis-length-trace-class = 1⟩

instance

proof *intro-classes*

fix x y z :: ⟨('a, 'b) trace-class⟩

have [simp]: ⟨1 ≠ (0 :: ('a, 'b) trace-class)⟩

using one-trace-class.rep-eq by force

then have [simp]: ⟨0 ≠ (1 :: ('a, 'b) trace-class)⟩

by force

show ⟨distinct (canonical-basis :: (-,-) trace-class list)⟩

by (simp add: canonical-basis-trace-class-def)

show ⟨cindependent (set canonical-basis :: (-,-) trace-class set)⟩

by (simp add: canonical-basis-trace-class-def)

show ⟨canonical-basis-length TYPE((('a, 'b) trace-class) = length (canonical-basis :: (-,-) trace-class list)⟩

by (simp add: canonical-basis-length-trace-class-def canonical-basis-trace-class-def)

show ⟨x ∈ set canonical-basis ⇒ norm x = 1⟩

apply (simp add: canonical-basis-trace-class-def)

by (smt (verit, ccfv-threshold) one-trace-class-def cinner-trace-class.abs-eq cnorm-eq-1 one-cinner-one one-trace-class.rsp)

show ⟨canonical-basis = [1 :: ('a, 'b) trace-class]⟩

```

    by (simp add: canonical-basis-trace-class-def)
  show ⟨a *C 1 * b *C 1 = (a * b) *C (1 :: ('a,'b) trace-class)⟩ for a b :: complex
    apply (transfer' fixing: a b)
    by simp
  show ⟨x div y = x * inverse y⟩
    apply transfer'
    by (simp add: divide-cblinfun-def)
  show ⟨inverse (a *C 1 :: ('a,'b) trace-class) = 1 /C a⟩ for a :: complex
    apply transfer'
    by simp
  show ⟨is-ortho-set (set canonical-basis :: ('a,'b) trace-class set)⟩
    by (simp add: is-ortho-set-def canonical-basis-trace-class-def)
  show ⟨cspan (set canonical-basis :: ('a,'b) trace-class set) = UNIV⟩
  proof (intro Set.set-eqI iffI UNIV-I)
    fix x :: ⟨('a,'b) trace-class⟩
    have ⟨∃ c. y = c *C 1⟩ for y :: ⟨'a ⇒CL 'b⟩
      apply (rule exI[where x=⟨one-dim-iso y⟩])
      by simp
    then obtain c where ⟨x = c *C 1⟩
      apply transfer'
      by auto
    then show ⟨x ∈ cspan (set canonical-basis)⟩
      by (auto intro!: complex-vector.span-base complex-vector.span-clauses
        simp: canonical-basis-trace-class-def)
  qed
qed
end

```

```

lemma from-trace-class-one-dim-iso[simp]: ⟨from-trace-class = one-dim-iso⟩
proof (rule ext)
  fix x :: ⟨('a, 'b) trace-class⟩
  have ⟨from-trace-class x = from-trace-class (one-dim-iso x *C 1)⟩
    by simp
  also have ⟨... = one-dim-iso x *C from-trace-class 1⟩
    using scaleC-trace-class.rep-eq by blast
  also have ⟨... = one-dim-iso x *C 1⟩
    by (simp add: one-trace-class.rep-eq)
  also have ⟨... = one-dim-iso x⟩
    by simp
  finally show ⟨from-trace-class x = one-dim-iso x⟩
    by -
qed

```

```

lemma trace-tc-one-dim-iso[simp]: ⟨trace-tc = one-dim-iso⟩
  by (simp add: trace-tc.rep-eq[abs-def])

```

```

lemma compose-tcr-id-left[simp]: ⟨compose-tcr id-cblinfun t = t⟩
  by (auto intro!: from-trace-class-inject[THEN iffD1] simp: compose-tcr.rep-eq)

```

```

lemma compose-tcl-id-right[simp]: ⟨compose-tcl t id-cblinfun = t⟩
  by (auto intro!: from-trace-class-inject[THEN iffD1] simp: compose-tcl.rep-eq)

lemma sandwich-tc-id-cblinfun[simp]: ⟨sandwich-tc id-cblinfun t = t⟩
  by (simp add: from-trace-class-inverse sandwich-tc-def)

lemma bounded-clinear-sandwich-tc[bounded-clinear]: ⟨bounded-clinear (sandwich-tc e)⟩
  using norm-sandwich-tc[of e]
  by (auto intro!: bounded-clinearI[where K=⟨(norm e)2⟩]
    simp: sandwich-tc-plus sandwich-tc-scaleC-right cross3-simps)

lemma trace-class-Proj: ⟨trace-class (Proj S) ⟷ finite-dim-ccsubspace S⟩
proof –
  define C where ⟨C = some-onb-of S⟩
  then obtain B where ⟨is-onb B⟩ and ⟨C ⊆ B⟩
    using orthonormal-basis-exists some-onb-of-norm1 by blast
  have card-C: ⟨card C = cdim (space-as-set S)⟩
    by (simp add: C-def some-onb-of-card)
  have S-C: ⟨S = ccspan C⟩
    by (simp add: C-def)

from ⟨is-onb B⟩
have ⟨trace-class (Proj S) ⟷ ((λx. x •C (abs-op (Proj S) *V x)) abs-summable-on B)⟩
  by (rule trace-class-iff-summable)
also have ⟨... ⟷ ((λx. cmod (x •C (Proj S *V x))) abs-summable-on B)⟩
  by simp
also have ⟨... ⟷ ((λx. 1::real) abs-summable-on C)⟩
proof (rule summable-on-cong-neutral)
  fix x :: 'a
  show ⟨norm 1 = 0⟩ if ⟨x ∈ C - B⟩
    using that ⟨C ⊆ B⟩ by auto
  show ⟨norm (cmod (x •C (Proj S *V x))) = norm (1::real)⟩ if ⟨x ∈ B ∩ C⟩
    apply (subst Proj-fixes-image)
    using C-def Int-absorb1 that ⟨is-onb B⟩
    by (auto simp: is-onb-def cnorm-eq-1)
  show ⟨norm (cmod (x •C (Proj S *V x))) = 0⟩ if ⟨x ∈ B - C⟩
    apply (subst Proj-0-compl)
    apply (subst S-C)
    apply (rule mem-ortho-ccspanI)
    using that ⟨is-onb B⟩ ⟨C ⊆ B⟩
    by (force simp: is-onb-def is-ortho-set-def)+
qed
also have ⟨... ⟷ finite C⟩
  using infsum-diverge-constant[where A=C and c=⟨1::real⟩]
  by auto
also have ⟨... ⟷ finite-dim-ccsubspace S⟩
  by (metis C-def S-C ccspan-finite-dim some-onb-of-finite-dim)
finally show ?thesis

```

```

    by –
qed

lemma not-trace-class-trace0: ⟨trace a = 0⟩ if ⟨¬ trace-class a⟩
  using that by (simp add: trace-def)

lemma trace-Proj: ⟨trace (Proj S) = cdim (space-as-set S)⟩
proof (cases ⟨finite-dim-ccsubspace S⟩)
  case True
  define C where ⟨C = some-onb-of S⟩
  then obtain B where ⟨is-onb B⟩ and ⟨C ⊆ B⟩
  using orthonormal-basis-exists some-onb-of-norm1 by blast
  have [simp]: ⟨finite C⟩
  using C-def True some-onb-of-finite-dim by blast
  have card-C: ⟨card C = cdim (space-as-set S)⟩
  by (simp add: C-def some-onb-of-card)
  have S-C: ⟨S = ccspan C⟩
  by (simp add: C-def)

  from True have ⟨trace-class (Proj S)⟩
  by (simp add: trace-class-Proj)
  with ⟨is-onb B⟩ have ⟨((λe. e •C (Proj S *V e)) has-sum trace (Proj S)) B⟩
  by (rule trace-has-sum)
  then have ⟨((λe. 1) has-sum trace (Proj S)) C⟩
  proof (rule has-sum-cong-neutral[THEN iffD1, rotated -1])
    fix x :: 'a
    show ⟨1 = 0⟩ if ⟨x ∈ C – B⟩
      using that ⟨C ⊆ B⟩ by auto
    show ⟨x •C (Proj S *V x) = 1⟩ if ⟨x ∈ B ∩ C⟩
      apply (subst Proj-fixes-image)
      using C-def Int-absorb1 that ⟨is-onb B⟩
      by (auto simp: is-onb-def cnorm-eq-1)
    show ⟨is-orthogonal x (Proj S *V x)⟩ if ⟨x ∈ B – C⟩
      apply (subst Proj-0-compl)
      apply (subst S-C)
      apply (rule mem-ortho-ccspanI)
      using that ⟨is-onb B⟩ ⟨C ⊆ B⟩
      by (force simp: is-onb-def is-ortho-set-def)+
  qed
  then have ⟨trace (Proj S) = card C⟩
  using has-sum-constant[OF ⟨finite C⟩, of 1]
  apply simp
  using has-sum-unique by blast
  also have ⟨... = cdim (space-as-set S)⟩
  using card-C by presburger
  finally show ?thesis
  by –
next

```

```

case False
then have ⟨¬ trace-class (Proj S)⟩
  using trace-class-Proj by blast
then have ⟨trace (Proj S) = 0⟩
  by (rule not-trace-class-trace0)
moreover from False have ⟨cdim (space-as-set S) = 0⟩
  apply transfer
  by (simp add: cdim-infinite-0)
ultimately show ?thesis
  by simp
qed

lemma trace-tc-pos: ⟨t ≥ 0 ⟹ trace-tc t ≥ 0⟩
  using trace-tc-mono by fastforce

lift-definition tc-apply :: ⟨('a::hilbert-space,'b::hilbert-space) trace-class ⟹ 'a ⟹ 'b⟩ is cblin-
fun-apply.

lemma bounded-cbilinear-tc-apply: ⟨bounded-cbilinear tc-apply⟩
  apply (rule bounded-cbilinear.intro; transfer)
  apply (auto intro!: exI[of - 1] cblinfun.add-left cblinfun.add-right cblinfun.scaleC-right)
  by (smt (verit, del-insts) mult-right-mono norm-cblinfun norm-ge-zero norm-leq-trace-norm)

lift-definition diagonal-operator-tc :: ⟨('a ⟹ complex) ⟹ ('a ell2, 'a ell2) trace-class⟩ is
⟨λf. if f abs-summable-on UNIV then diagonal-operator f else 0⟩
proof (rule CollectI)
fix f :: ⟨'a ⟹ complex⟩
show ⟨trace-class (if f abs-summable-on UNIV then diagonal-operator f else 0)⟩
proof (cases ⟨f abs-summable-on UNIV⟩)
case True
have bdd: ⟨bdd-above (range (λx. cmod (f x)))⟩
proof (rule bdd-aboveI2)
fix x
have ⟨cmod (f x) = (∑∞x∈{x}. cmod (f x))⟩
  by simp
also have ⟨... ≤ (∑∞x. cmod (f x))⟩
  apply (rule infsum-mono-neutral)
  by (auto intro!: True)
finally show ⟨cmod (f x) ≤ (∑∞x. cmod (f x))⟩
  by -
qed
have ⟨trace-class (diagonal-operator f)⟩
  by (auto intro!: trace-classI[OF is-onb-ket] summable-on-reindex[THEN iffD2] True
    simp: abs-op-diagonal-operator o-def diagonal-operator-ket bdd)
with True show ?thesis
  by simp
next
case False
then show ?thesis

```


by *simp*
qed
qed

lemma *from-trace-class-diagonal-operator-tc*:
assumes $\langle f \text{ abs-summable-on } UNIV \rangle$
shows $\langle \text{from-trace-class } (\text{diagonal-operator-tc } f) = \text{diagonal-operator } f \rangle$
apply (*transfer fixing: f*)
using *assms* **by** *simp*

lemma *tc-butterfly-scaleC-summable*:
fixes $f :: \langle 'a \Rightarrow \text{complex} \rangle$
assumes $\langle f \text{ abs-summable-on } A \rangle$
shows $\langle (\lambda x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) \text{ summable-on } A \rangle$
proof –
define M **where** $\langle M = (\sum_{\infty} x \in A. \text{norm } (f x)) \rangle$
have $\langle (\sum_{x \in F. \text{cmod } (f x) * \text{norm } (\text{tc-butterfly } (\text{ket } x) (\text{ket } x))) \leq M \rangle$ **if** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq A \rangle$ **for** F
proof –
have $\langle (\sum_{x \in F. \text{norm } (f x) * \text{norm } (\text{tc-butterfly } (\text{ket } x) (\text{ket } x))) = (\sum_{x \in F. \text{norm } (f x)) \rangle$
by (*simp add: norm-tc-butterfly*)
also have $\langle \dots \leq (\sum_{\infty} x \in A. \text{norm } (f x)) \rangle$
using *assms finite-sum-le-infsum norm-ge-zero that(1) that(2)* **by** *blast*
also have $\langle \dots = M \rangle$
by (*simp add: M-def*)
finally show *?thesis*
by –
qed
then have $\langle (\lambda x. \text{norm } (f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x))) \text{ abs-summable-on } A \rangle$
apply (*intro nonneg-bdd-above-summable-on bdd-aboveI*)
by *auto*
then show *?thesis*
by (*auto intro: abs-summable-summable*)
qed

lemma *tc-butterfly-scaleC-has-sum*:
fixes $f :: \langle 'a \Rightarrow \text{complex} \rangle$
assumes $\langle f \text{ abs-summable-on } UNIV \rangle$
shows $\langle ((\lambda x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) \text{ has-sum diagonal-operator-tc } f) UNIV \rangle$
proof –
define D **where** $\langle D = (\sum_{\infty} x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) \rangle$
have *bdd-f*: $\langle \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (f x))) \rangle$
by (*metis assms summable-on-bdd-above-real*)

have $\langle \text{ket } y \cdot_C \text{from-trace-class } D (\text{ket } z) = \text{ket } y \cdot_C \text{from-trace-class } (\text{diagonal-operator-tc } f) (\text{ket } z) \rangle$ **for** $y z$
proof –

```

have blin-tc-apply: ⟨bounded-linear (λa. tc-apply a (ket z))⟩
by (intro bounded-clinear.bounded-linear bounded-cbilinear.bounded-clinear-left bounded-cbilinear-tc-apply)
have summ: ⟨(λx. f x *C tc-butterfly (ket x) (ket x)) summable-on UNIV⟩
by (intro tc-butterfly-scaleC-summable assms)

have ⟨(λx. f x *C tc-butterfly (ket x) (ket x)) has-sum D⟩ UNIV
by (simp add: D-def summ)
with blin-tc-apply have ⟨(λx. tc-apply (f x *C tc-butterfly (ket x) (ket x)) (ket z)) has-sum
tc-apply D (ket z)⟩ UNIV
by (rule Infinite-Sum.has-sum-bounded-linear)
then have ⟨(λx. ket y •C tc-apply (f x *C tc-butterfly (ket x) (ket x)) (ket z)) has-sum ket
y •C tc-apply D (ket z)⟩ UNIV
by (smt (verit, best) has-sum-cong has-sum-imp-summable has-sum-infsum infsumI inf-
sum-cinner-left summable-on-cinner-left)
then have ⟨(λx. of-bool (x=y) * of-bool (y=z) * f y) has-sum ket y •C tc-apply D (ket z)⟩
UNIV
apply (rule has-sum-cong[THEN iffD2, rotated])
by (auto intro!: simp: tc-apply.rep-eq scaleC-trace-class.rep-eq tc-butterfly.rep-eq)
then have ⟨(λx. of-bool (y=z) * f y) has-sum ket y •C tc-apply D (ket z)⟩ {y}
apply (rule has-sum-cong-neutral[THEN iffD2, rotated -1])
by auto
then have ⟨ket y •C tc-apply D (ket z) = of-bool (y=z) * f y⟩
by simp
also have ⟨... = ket y •C from-trace-class (diagonal-operator-tc f) (ket z)⟩
by (simp add: diagonal-operator-tc.rep-eq assms diagonal-operator-ket bdd-f)
finally show ?thesis
by (simp add: tc-apply.rep-eq)
qed
then have ⟨from-trace-class D = from-trace-class (diagonal-operator-tc f)⟩
by (auto intro!: equal-ket cinner-ket-eqI)
then have ⟨D = diagonal-operator-tc f⟩
by (simp add: from-trace-class-inject)
with tc-butterfly-scaleC-summable[OF assms]
show ?thesis
using D-def by force
qed

lemma diagonal-operator-tc-invalid: ⟨¬ f abs-summable-on UNIV ⟹ diagonal-operator-tc f =
0⟩
apply (transfer fixing: f) by simp

lemma tc-butterfly-scaleC-infsum:
fixes f :: ⟨'a ⇒ complex⟩
shows ⟨(∑∞ x. f x *C tc-butterfly (ket x) (ket x)) = diagonal-operator-tc f⟩
proof (cases ⟨f abs-summable-on UNIV⟩)
case True
then show ?thesis

```

```

    using infsunI tc-butterfly-scaleC-has-sum by fastforce
next
case False
then have [simp]:  $\langle \text{diagonal-operator-tc } f = 0 \rangle$ 
  apply (transfer fixing: f) by simp
have  $\langle \neg (\lambda x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) \text{ summable-on } UNIV \rangle$ 
proof (rule notI)
  assume  $\langle (\lambda x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) \text{ summable-on } UNIV \rangle$ 
  then have  $\langle (\lambda x. \text{trace-tc } (f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x))) \text{ summable-on } UNIV \rangle$ 
    apply (rule summable-on-bounded-linear[rotated])
    by (simp add: bounded-clinear.bounded-linear)
  then have  $\langle f \text{ summable-on } UNIV \rangle$ 
    apply (rule summable-on-cong[THEN iffD1, rotated])
    apply (transfer' fixing: f)
    by (simp add: trace-scaleC trace-butterfly)
  with False
  show False
    by (metis summable-on-iff-abs-summable-on-complex)
qed
then have [simp]:  $\langle (\sum_{\infty} x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) = 0 \rangle$ 
  using infsun-not-exists by blast
show ?thesis
  by simp
qed

```

```

lemma from-trace-class-abs-summable:  $\langle f \text{ abs-summable-on } X \implies (\lambda x. \text{from-trace-class } (f x)) \text{ abs-summable-on } X \rangle$ 
  apply (rule abs-summable-on-comparison-test[where g= $\langle f \rangle$ ])
  by (simp-all add: norm-leq-trace-norm norm-trace-class.rep-eq)

```

```

lemma from-trace-class-summable:  $\langle f \text{ summable-on } X \implies (\lambda x. \text{from-trace-class } (f x)) \text{ summable-on } X \rangle$ 
  apply (rule Infinite-Sum.summable-on-bounded-linear[where h=from-trace-class])
  by (simp-all add: bounded-clinear.bounded-linear bounded-clinear-from-trace-class)

```

```

lemma from-trace-class-infsun:
  assumes  $\langle f \text{ summable-on } UNIV \rangle$ 
  shows  $\langle \text{from-trace-class } (\sum_{\infty} x. f x) = (\sum_{\infty} x. \text{from-trace-class } (f x)) \rangle$ 
  apply (rule infsun-bounded-linear-strong[symmetric])
  using assms
  by (auto intro!: bounded-clinear.bounded-linear bounded-clinear-from-trace-class from-trace-class-summable)

```

```

lemma cspan-trace-class:
   $\langle \text{cspan } (\text{Collect trace-class } :: ('a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space}) \text{ set}) = \text{Collect trace-class} \rangle$ 
proof (intro Set.set-eqI iffI)
  fix  $x :: 'a \Rightarrow_{CL} 'b$ 
  show  $\langle x \in \text{Collect trace-class} \implies x \in \text{cspan } (\text{Collect trace-class}) \rangle$ 
    by (simp add: complex-vector.span-clauses)
  assume  $\langle x \in \text{cspan } (\text{Collect trace-class}) \rangle$ 

```

```

then obtain  $F f$  where  $x\text{-def}$ :  $\langle x = (\sum a \in F. f a *_C a) \rangle$  and  $\langle F \subseteq \text{Collect trace-class} \rangle$ 
  by (auto intro!: simp: complex-vector.span-explicit)
then have  $\langle \text{trace-class } x \rangle$ 
  by (auto intro!: trace-class-sum trace-class-scaleC simp:  $x\text{-def}$ )
then show  $\langle x \in \text{Collect trace-class} \rangle$ 
  by simp
qed

lemma monotone-convergence-tc:
  fixes  $f :: \langle 'b \Rightarrow ('a, 'a::\text{chilbert-space}) \text{trace-class} \rangle$ 
  assumes bounded:  $\langle \forall_F x \text{ in } F. \text{trace-tc } (f x) \leq B \rangle$ 
  assumes pos:  $\langle \forall_F x \text{ in } F. f x \geq 0 \rangle$ 
  assumes increasing:  $\langle \text{increasing-filter } (\text{filtermap } f F) \rangle$ 
  shows  $\langle \exists L. (f \longrightarrow L) F \rangle$ 
proof –
  wlog  $\langle F \neq \perp \rangle$ 
  using negation by simp
  then have  $\langle \text{filtermap } f F \neq \perp \rangle$ 
  by (simp add: filtermap-bot-iff)
  have  $\langle \text{mono-on } \{t::('a, 'a) \text{trace-class}. t \geq 0\} \text{trace-tc} \rangle$ 
  by (simp add: ord.mono-onI trace-tc-mono)
  with increasing
  have  $\langle \text{increasing-filter } (\text{filtermap } (\text{trace-tc } o f) F) \rangle$ 
  unfolding filtermap-compose
  apply (rule increasing-filtermap)
  by (auto intro!: pos simp: eventually-filtermap)
  then obtain  $l$  where  $l$ :  $\langle ((\lambda x. \text{trace-tc } (f x)) \longrightarrow l) F \rangle$ 
  apply atomize-elim
  apply (rule monotone-convergence-complex)
  using bounded by (simp-all add: o-def)
  have  $\langle \text{cauchy-filter } (\text{filtermap } f F) \rangle$ 
  proof (rule cauchy-filter-metricI)
  fix  $e :: \text{real}$  assume  $\langle e > 0 \rangle$ 
  define  $d$  where  $\langle d = e/4 \rangle$ 
  have  $\langle \forall_F x \text{ in } \text{filtermap } f F. \text{dist } (\text{trace-tc } x) l < d \rangle$ 
  unfolding eventually-filtermap
  using  $l$  apply (rule tendstoD)
  using  $\langle e > 0 \rangle$  by (simp add: d-def)
  then obtain  $P1$  where ev-P1:  $\langle \text{eventually } P1 (\text{filtermap } f F) \rangle$  and  $P1$ :  $\langle P1 x \implies \text{dist } (\text{trace-tc } x) l < d \rangle$  for  $x$ 
  by blast
  from increasing obtain  $P2$  where ev-P2:  $\langle \text{eventually } P2 (\text{filtermap } f F) \rangle$  and
   $P2$ :  $\langle P2 x \implies (\forall_F z \text{ in } \text{filtermap } f F. z \geq x) \rangle$  for  $x$ 
  using increasing-filter-def by blast
  define  $P$  where  $\langle P x \iff P1 x \wedge P2 x \rangle$  for  $x$ 
  with ev-P1 ev-P2 have ev-P:  $\langle \text{eventually } P (\text{filtermap } f F) \rangle$ 
  by (auto intro!: eventually-conj simp: P-def[abs-def])
  have  $\langle \text{dist } x y < e \rangle$  if  $\langle P x \rangle$  and  $\langle P y \rangle$  for  $x y$ 
  proof –

```

from $\langle P \ x \rangle$ **have** $\langle \forall_F \ z \text{ in } \text{filtermap } f \ F. \ z \geq x \rangle$
by (*simp add: P-def P2*)
moreover from $\langle P \ y \rangle$ **have** $\langle \forall_F \ z \text{ in } \text{filtermap } f \ F. \ z \geq y \rangle$
by (*simp add: P-def P2*)
ultimately have $\langle \forall_F \ z \text{ in } \text{filtermap } f \ F. \ z \geq x \wedge z \geq y \wedge P1 \ z \rangle$
using *ev-P1* **by** (*auto intro!: eventually-conj*)
from *eventually-happens'*[*OF* $\langle \text{filtermap } f \ F \neq \perp \rangle$ *this*]
obtain z **where** $\langle z \geq x \rangle$ **and** $\langle z \geq y \rangle$ **and** $\langle P1 \ z \rangle$
by *auto*
have $\langle \text{dist } x \ y \leq \text{norm } (z - x) + \text{norm } (z - y) \rangle$
by (*metis (no-types, lifting) diff-add-cancel diff-add-eq-diff-diff-swap dist-trace-class-def norm-minus-commute norm-triangle-sub*)
also from $\langle x \leq z \rangle \langle y \leq z \rangle$ **have** $\langle \dots = (\text{trace-tc } z - \text{trace-tc } x) + (\text{trace-tc } z - \text{trace-tc } y) \rangle$
by (*metis (no-types, lifting) cross3-simps(16) diff-left-mono diff-self norm-tc-pos of-real-add trace-tc-plus*)
also from $\langle x \leq z \rangle \langle y \leq z \rangle$ **have** $\langle \dots = \text{norm } (\text{trace-tc } z - \text{trace-tc } x) + \text{norm } (\text{trace-tc } z - \text{trace-tc } y) \rangle$
by (*simp add: Extra-Ordered-Fields.complex-of-real-cmod trace-tc-mono abs-pos*)
also have $\langle \dots = \text{dist } (\text{trace-tc } z) (\text{trace-tc } x) + \text{dist } (\text{trace-tc } z) (\text{trace-tc } y) \rangle$
using *dist-complex-def* **by** *presburger*
also have $\langle \dots \leq (\text{dist } (\text{trace-tc } z) \ l + \text{dist } (\text{trace-tc } x) \ l) + (\text{dist } (\text{trace-tc } z) \ l + \text{dist } (\text{trace-tc } y) \ l) \rangle$
apply (*intro complex-of-real-mono add-mono*)
by (*simp-all add: dist-triangle2*)
also from $P1 \ \langle P1 \ z \rangle$ **that have** $\langle \dots < 4 * d \rangle$
by (*smt (verit, best) P-def complex-of-real-strict-mono-iff*)
also have $\langle \dots = e \rangle$
by (*simp add: d-def*)
finally show *?thesis*
by *simp*
qed
with *ev-P* **show** $\langle \exists P. \text{eventually } P \ (\text{filtermap } f \ F) \wedge (\forall x \ y. P \ x \wedge P \ y \longrightarrow \text{dist } x \ y < e) \rangle$
by *blast*
qed
then have $\langle \text{convergent-filter } (\text{filtermap } f \ F) \rangle$
using *cauchy-filter-convergent* **by** *fastforce*
then show $\langle \exists L. (f \longrightarrow L) \ F \rangle$
by (*simp add: convergent-filter-iff filterlim-def*)
qed

lemma *nonneg-bdd-above-summable-on-tc*:
fixes $f :: \langle 'a \Rightarrow ('c::\text{hilbert-space}, 'c) \text{trace-class} \rangle$
assumes *pos*: $\langle \bigwedge x. x \in A \implies f \ x \geq 0 \rangle$
assumes *bdd*: $\langle \text{bdd-above } (\text{trace-tc } ' \ \text{sum } f \ ' \ \{F. F \subseteq A \wedge \text{finite } F\}) \rangle$
shows $\langle f \ \text{summable-on } A \rangle$
proof –
have *pos'*: $\langle (\sum x \in F. f \ x) \geq 0 \rangle$ **if** $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq A \rangle$ **for** F
using *that pos*

by (simp add: basic-trans-rules(31) sum-nonneg)
 from pos have incr: $\langle \text{increasing-filter } (\text{filtermap } (\text{sum } f) (\text{finite-subsets-at-top } A)) \rangle$
 by (auto intro!: increasing-filtermap[where $X = \langle \{F. \text{finite } F \wedge F \subseteq A\} \rangle$] mono-onI sum-mono2)
 from bdd obtain B where B: $\langle \text{trace-tc } (\text{sum } f X) \leq B \rangle$ if $\langle \text{finite } X \rangle$ and $\langle X \subseteq A \rangle$ for X
 apply atomize-elim
 by (auto simp: bdd-above-def)
 show ?thesis
 apply (simp add: summable-on-def has-sum-def)
 by (safe intro!: pos' incr monotone-convergence-tc[where $B=B$] B
 eventually-finite-subsets-at-top-weakI)
 qed

lemma summable-Sigma-positive-tc:

fixes f :: $\langle 'a \Rightarrow 'b \Rightarrow ('c, 'c::\text{chilbert-space}) \text{trace-class} \rangle$
 assumes $\langle \bigwedge x. x \in X \implies f x \text{ summable-on } Y x \rangle$
 assumes $\langle (\lambda x. \sum_{\infty} y \in Y x. f x y) \text{ summable-on } X \rangle$
 assumes $\langle \bigwedge x y. x \in X \implies y \in Y x \implies f x y \geq 0 \rangle$
 shows $\langle (\lambda(x, y). f x y) \text{ summable-on } (\text{SIGMA } x:X. Y x) \rangle$
 proof -
 have $\langle \text{trace-tc } (\sum_{(x,y) \in F} f x y) \leq \text{trace-tc } (\sum_{\infty} x \in X. \sum_{\infty} y \in Y x. f x y) \rangle$ if $\langle F \subseteq \text{Sigma } X Y \rangle$
 and $\langle \text{finite } F \rangle$ for F
 proof -
 define g where $\langle g x y = (\text{if } (x,y) \in \text{Sigma } X Y \text{ then } f x y \text{ else } 0) \rangle$ for x y
 have g-pos[iff]: $\langle g x y \geq 0 \rangle$ for x y
 using assms by (auto intro!: simp: g-def)
 have g-summable: $\langle g x \text{ summable-on } Y x \rangle$ for x
 by (metis assms(1) g-def mem-Sigma-iff summable-on-0 summable-on-cong)
 have sum-g-summable: $\langle (\lambda x. \sum_{\infty} y \in Y x. g x y) \text{ summable-on } X \rangle$
 by (metis (mono-tags, lifting) SigmaI g-def assms(2) infsum-cong summable-on-cong)
 have $\langle (\sum_{(x,y) \in F} f x y) = (\sum_{(x,y) \in F} g x y) \rangle$
 by (smt (verit, ccfv-SIG) g-def split-cong subsetD sum.cong that(1))
 also have $\langle (\sum_{(x,y) \in F} g x y) \leq (\sum_{(x,y) \in \text{fst } 'F \times \text{snd } 'F} g x y) \rangle$
 using that assms apply (auto intro!: sum-mono2 assms simp: image-iff)
 by force+
 also have $\langle \dots = (\sum_{x \in \text{fst } 'F} \sum_{y \in \text{snd } 'F} g x y) \rangle$
 by (metis (no-types, lifting) finite-imageI sum.Sigma sum.cong that(2))
 also have $\langle \dots = (\sum_{x \in \text{fst } 'F} \sum_{\infty} y \in \text{snd } 'F} g x y) \rangle$
 by (metis finite-imageI infsum-finite that(2))
 also have $\langle \dots \leq (\sum_{x \in \text{fst } 'F} \sum_{\infty} y \in Y x} g x y) \rangle$
 apply (intro sum-mono infsum-mono-neutral-traceclass)
 using assms that
 apply (auto intro!: g-summable)
 by (simp add: g-def)
 also have $\langle \dots = (\sum_{\infty} x \in \text{fst } 'F} \sum_{\infty} y \in Y x} g x y) \rangle$
 using that by (auto intro!: infsum-finite[symmetric] simp:)
 also have $\langle \dots \leq (\sum_{\infty} x \in X} \sum_{\infty} y \in Y x} g x y) \rangle$
 apply (rule infsum-mono-neutral-traceclass)
 using that assms by (auto intro!: infsum-nonneg-traceclass sum-g-summable)

```

also have  $\langle \dots = (\sum_{\infty x \in X}. \sum_{\infty y \in Y} x. f x y) \rangle$ 
by (smt (verit, ccfv-threshold) g-def infsum-cong mem-Sigma-iff)
finally show ?thesis
using trace-tc-mono by blast
qed
then show ?thesis
apply (rule-tac nonneg-bdd-above-summable-on-tc)
by (auto intro!: assms bdd-aboveI2)
qed

lemma infsum-Sigma-positive-tc:
  fixes  $f :: \langle 'a \Rightarrow 'b \Rightarrow ('c::\text{chilbert-space}, 'c) \text{ trace-class} \rangle$ 
  assumes  $\langle \bigwedge x. x \in X \Longrightarrow f x \text{ summable-on } Y x \rangle$ 
  assumes  $\langle \bigwedge x y. x \in X \Longrightarrow y \in Y x \Longrightarrow f x y \geq 0 \rangle$ 
  shows  $\langle (\sum_{\infty x \in X}. \sum_{\infty y \in Y} x. f x y) = (\sum_{\infty (x,y) \in \text{Sigma } X Y}. f x y) \rangle$ 
proof (cases  $\langle \lambda x. \sum_{\infty y \in Y} x. f x y \text{ summable-on } X \rangle$ )
  case True
    show ?thesis
    apply (rule infsum-Sigma'-banach)
    apply (rule summable-Sigma-positive-tc)
    using assms True by auto
  next
    case False
    then have  $1: \langle (\sum_{\infty x \in X}. \sum_{\infty y \in Y} x. f x y) = 0 \rangle$ 
    using infsum-not-exists by blast
    from False have  $\langle \neg (\lambda(x,y). f x y) \text{ summable-on Sigma } X Y \rangle$ 
    using summable-on-Sigma-banach by blast
    then have  $2: \langle (\sum_{\infty (x,y) \in \text{Sigma } X Y}. f x y) = 0 \rangle$ 
    using infsum-not-exists by blast
    from  $1\ 2$  show ?thesis
    by simp
  qed

lemma infsum-swap-positive-tc:
  fixes  $f :: \langle 'a \Rightarrow 'b \Rightarrow ('c::\text{chilbert-space}, 'c) \text{ trace-class} \rangle$ 
  assumes  $\langle \bigwedge x. x \in X \Longrightarrow f x \text{ summable-on } Y \rangle$ 
  assumes  $\langle \bigwedge y. y \in Y \Longrightarrow (\lambda x. f x y) \text{ summable-on } X \rangle$ 
  assumes  $\langle \bigwedge x y. x \in X \Longrightarrow y \in Y \Longrightarrow f x y \geq 0 \rangle$ 
  shows  $\langle (\sum_{\infty x \in X}. \sum_{\infty y \in Y}. f x y) = (\sum_{\infty y \in Y}. \sum_{\infty x \in X}. f x y) \rangle$ 
proof -
  have  $\langle (\sum_{\infty x \in X}. \sum_{\infty y \in Y}. f x y) = (\sum_{\infty (x,y) \in X \times Y}. f x y) \rangle$ 
  apply (rule infsum-Sigma-positive-tc)
  using assms by auto
  also have  $\langle \dots = (\sum_{\infty (y,x) \in Y \times X}. f x y) \rangle$ 
  apply (subst product-swap[symmetric])
  by (simp add: infsum-reindex o-def)
  also have  $\langle \dots = (\sum_{\infty y \in Y}. \sum_{\infty x \in X}. f x y) \rangle$ 
  apply (rule infsum-Sigma-positive-tc[symmetric])

```

```

    using assms by auto
  finally show ?thesis
    by -
qed

```

lemma *separating-density-ops*:

```

  assumes  $\langle B > 0 \rangle$ 
  shows  $\langle \text{separating-set clinear } \{t :: ('a::\text{chilbert-space}, 'a) \text{ trace-class. } 0 \leq t \wedge \text{norm } t \leq B\} \rangle$ 
proof -
  define S where  $\langle S = \{t :: ('a, 'a) \text{ trace-class. } 0 \leq t \wedge \text{norm } t \leq B\} \rangle$ 
  have  $\langle \text{cspan } S = \text{UNIV} \rangle$ 
  proof (intro Set.set-eqI iffI UNIV-I)
    fix  $t :: \langle ('a, 'a) \text{ trace-class} \rangle$ 
    from trace-class-decomp-4pos'
    obtain  $t_1 t_2 t_3 t_4$  where t-decomp:  $\langle t = t_1 - t_2 + i *_{\mathbb{C}} t_3 - i *_{\mathbb{C}} t_4 \rangle$ 
      and pos:  $\langle t_1 \geq 0 \rangle \langle t_2 \geq 0 \rangle \langle t_3 \geq 0 \rangle \langle t_4 \geq 0 \rangle$ 
      by fast
    have  $\langle t' \in \text{cspan } S \rangle$  if  $\langle t' \geq 0 \rangle$  for  $t'$ 
    proof -
      define  $t''$  where  $\langle t'' = (B / \text{norm } t') *_{\mathbb{R}} t' \rangle$ 
      have  $\langle t'' \in S \rangle$ 
        using  $\langle B > 0 \rangle$ 
        by (simp add: S-def that zero-le-scaleR-iff t''-def)
      have  $t'-t''$ :  $\langle t' = (\text{norm } t' / B) *_{\mathbb{R}} t'' \rangle$ 
        using  $\langle B > 0 \rangle$  t''-def by auto
      show  $\langle t' \in \text{cspan } S \rangle$ 
        apply (subst t'-t'')
        using  $\langle t'' \in S \rangle$ 
        by (simp add: scaleR-scaleC complex-vector.span-clauses)
    qed
  with pos have  $\langle t_1 \in \text{cspan } S \rangle$  and  $\langle t_2 \in \text{cspan } S \rangle$  and  $\langle t_3 \in \text{cspan } S \rangle$  and  $\langle t_4 \in \text{cspan } S \rangle$ 
    by auto
  then show  $\langle t \in \text{cspan } S \rangle$ 
    by (auto intro!: complex-vector.span-diff complex-vector.span-add complex-vector.span-scale
      intro: complex-vector.span-base simp: t-decomp)
  qed
  then show  $\langle \text{separating-set clinear } S \rangle$ 
    by (rule separating-set-clinear-cspan)
qed

```

lemma *summable-abs-summable-tc*:

```

  fixes  $f :: \langle 'a \Rightarrow ('b::\text{chilbert-space}, 'b) \text{ trace-class} \rangle$ 
  assumes  $\langle f \text{ summable-on } X \rangle$ 
  assumes  $\langle \bigwedge x. x \in X \implies f x \geq 0 \rangle$ 
  shows  $\langle f \text{ abs-summable-on } X \rangle$ 
proof -
  from assms(1) have  $\langle (\lambda x. \text{trace-tc } (f x)) \text{ summable-on } X \rangle$ 

```


apply (*rule summable-on-bounded-linear*[rotated])
by (*simp add: bounded-clinear.bounded-linear*)
then have $\langle (\lambda x. \text{Re } (\text{trace-}tc \ (f \ x))) \text{ summable-on } X \rangle$
using *summable-on-Re* **by** *blast*
then show $\langle (\lambda x. \text{norm } (f \ x)) \text{ summable-on } X \rangle$
by (*metis (mono-tags, lifting) assms(2) norm-tc-pos-Re summable-on-cong*)
qed

12.5 More Hilbert-Schmidt

lemma *trace-class-hilbert-schmidt*: $\langle \text{hilbert-schmidt } a \rangle$ **if** $\langle \text{trace-class } a \rangle$
for $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
by (*auto intro!: trace-class-comp-right that simp: hilbert-schmidt-def*)

lemma *finite-rank-hilbert-schmidt*: $\langle \text{hilbert-schmidt } a \rangle$ **if** $\langle \text{finite-rank } a \rangle$
for $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
using *finite-rank-comp-right finite-rank-trace-class hilbert-schmidtI* **that** **by** *blast*

lemma *hilbert-schmidt-compact*: $\langle \text{compact-op } a \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
for $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
— [1], Corollary 18.7. (Only the second part. The first part is stated inside this proof though.)

proof —

have $\langle \exists b. \text{finite-rank } b \wedge \text{hilbert-schmidt-norm } (b - a) < \varepsilon \rangle$ **if** $\langle \varepsilon > 0 \rangle$ **for** ε

proof —

have $\langle \varepsilon^2 > 0 \rangle$

using *that* **by** *force*

obtain $B :: \langle 'a \text{ set} \rangle$ **where** $\langle \text{is-onb } B \rangle$

using *is-onb-some-chilbert-basis* **by** *blast*

with $\langle \text{hilbert-schmidt } a \rangle$ **have** $a\text{-sum-}B: \langle (\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ summable-on } B \rangle$

by (*auto intro!: summable-hilbert-schmidt-norm-square*)

then have $\langle ((\lambda x. (\text{norm } (a *_{\mathcal{V}} x))^2) \text{ has-sum } (\sum_{\infty x \in B. (\text{norm } (a *_{\mathcal{V}} x))^2)) B \rangle$

using *has-sum-ifs* **by** *blast*

from *tendsto-iff*[*THEN iffD1, rule-format, OF this*[*unfolded has-sum-def*] $\langle \varepsilon^2 > 0 \rangle$]

obtain F **where** [*simp*]: $\langle \text{finite } F \rangle$ **and** $\langle F \subseteq B \rangle$

and $F\text{bound}: \langle \text{dist } (\sum_{x \in F. (\text{norm } (a *_{\mathcal{V}} x))^2) (\sum_{\infty x \in B. (\text{norm } (a *_{\mathcal{V}} x))^2) < \varepsilon^2 \rangle$

apply *atomize-elim*

by (*auto intro!: simp: eventually-finite-subsets-at-top*)

define $p \ b$ **where** $\langle p = (\sum_{x \in F. \text{selfbutter } x} \rangle$ **and** $\langle b = a \ o_{CL} \ p \rangle$

have [*simp*]: $\langle p \ x = x \rangle$ **if** $\langle x \in F \rangle$ **for** x

apply (*simp add: p-def cblinfun.sum-left*)

apply (*subst sum-single*[**where** $i=x$])

using $\langle F \subseteq B \rangle$ **that** $\langle \text{is-onb } B \rangle$

by (*auto intro!: simp: cnorm-eq-1 is-onb-def is-ortho-set-def*)

have [*simp*]: $\langle p \ x = 0 \rangle$ **if** $\langle x \in B - F \rangle$ **for** x

using $\langle F \subseteq B \rangle$ **that** $\langle \text{is-onb } B \rangle$

apply (*auto intro!: sum.neutral simp add: p-def cblinfun.sum-left is-onb-def is-ortho-set-def*)

by *auto*

have $\langle \text{finite-rank } p \rangle$

by (*simp add: finite-rank-sum p-def*)

```

then have ‹finite-rank b›
  by (simp add: b-def finite-rank-comp-right)
with ‹hilbert-schmidt a› have ‹hilbert-schmidt (b - a)›
  by (auto intro!: hilbert-schmidt-minus intro: finite-rank-hilbert-schmidt)
then have ‹(hilbert-schmidt-norm (b - a))2 = (∑∞ x∈B. (norm ((b - a) *V x))2)›
  by (simp add: infsum-hilbert-schmidt-norm-square ‹is-onb B›)
also have ‹... = (∑∞ x∈B-F. (norm (a *V x))2)›
  by (auto intro!: infsum-cong-neutral
      simp: b-def cblinfun.diff-left)
also have ‹... = (∑∞ x∈B. (norm (a *V x))2) - (∑x∈F. (norm (a *V x))2)›
  apply (subst infsum-Diff)
  using ‹F ⊆ B› a-sum-B by auto
also have ‹... < ε2›
  using Fbound
  by (simp add: dist-norm)
finally show ?thesis
  using ‹finite-rank b›
  using power-less-imp-less-base that by fastforce
qed
then have ‹∃ b. finite-rank b ∧ dist b a < ε› if ‹ε > 0› for ε
  apply (rule ex-mono[rule-format, rotated])
  apply (auto intro!: that simp: dist-norm)
  using hilbert-schmidt-minus ‹hilbert-schmidt a› finite-rank-hilbert-schmidt hilbert-schmidt-norm-geq-norm
  by fastforce
then show ?thesis
  by (simp add: compact-op-finite-rank closure-approachable)
qed

```

```

lemma trace-class-compact: ‹compact-op a› if ‹trace-class a›
  for a :: ‹'a::hilbert-space ⇒CL 'b::hilbert-space›
  by (simp add: hilbert-schmidt-compact that trace-class-hilbert-schmidt)

```

12.6 Spectral Theorem

The spectral theorem for trace class operators. A corollary of the one for compact operators (*Hilbert-Space-Tensor-Product.Spectral-Theorem*) but not an immediate one.

lift-definition *spectral-dec-proj-tc* :: ‹('a::hilbert-space, 'a) trace-class ⇒ nat ⇒ ('a, 'a) trace-class› is

```

spectral-dec-proj
  using finite-rank-trace-class spectral-dec-proj-finite-rank trace-class-compact by blast

```

lift-definition *spectral-dec-val-tc* :: ‹('a::hilbert-space, 'a) trace-class ⇒ nat ⇒ complex› is *spectral-dec-val*.

```

lemma spectral-dec-proj-tc-finite-rank:
  assumes ‹adj-tc a = a›
  shows ‹finite-rank-tc (spectral-dec-proj-tc a n)›
  using assms apply transfer

```

by (simp add: spectral-dec-proj-finite-rank trace-class-compact)

lemma spectral-dec-summable-tc:

assumes $\langle \text{selfadjoint-tc } a \rangle$

shows $\langle (\lambda n. \text{spectral-dec-val-tc } a \ n \ *_{\mathbb{C}} \ \text{spectral-dec-proj-tc } a \ n) \ \text{abs-summable-on } \text{UNIV} \rangle$

proof (intro nonneg-bounded-partial-sums-imp-summable-on norm-ge-zero eventually-finite-subsets-at-top-weakI)

define a' where $\langle a' = \text{from-trace-class } a \rangle$

then have [transfer-rule]: $\langle \text{cr-trace-class } a' \ a \rangle$

by (simp add: cr-trace-class-def)

have $\langle \text{compact-op } a' \rangle$

by (auto intro!: trace-class-compact simp: a'-def)

have $\langle \text{selfadjoint } a' \rangle$

using a'-def assms selfadjoint-tc.rep-eq by blast

fix $F :: \langle \text{nat set} \rangle$ assume $\langle \text{finite } F \rangle$

define R where $\langle R = (\bigsqcup_{n \in F}. \text{spectral-dec-space } a' \ n) \rangle$

have $\langle (\sum_{x \in F}. \text{norm } (\text{spectral-dec-val-tc } a \ x \ *_{\mathbb{C}} \ \text{spectral-dec-proj-tc } a \ x))$

$= \text{norm } (\sum_{x \in F}. \text{spectral-dec-val-tc } a \ x \ *_{\mathbb{C}} \ \text{spectral-dec-proj-tc } a \ x) \rangle$

proof (rule norm-tc-sum-exchange[symmetric]; transfer; rename-tac $n \ m \ F$)

fix $n \ m :: \text{nat}$ assume $\langle n \neq m \rangle$

then have *: $\langle \text{Proj } (\text{spectral-dec-space } a' \ n) \ o_{\mathbb{C}L} \ \text{Proj } (\text{spectral-dec-space } a' \ m) = 0 \rangle$ if

$\langle \text{spectral-dec-val } a' \ n \neq 0 \rangle$ and $\langle \text{spectral-dec-val } a' \ m \neq 0 \rangle$

by (auto intro!: orthogonal-projectors-orthogonal-spaces[THEN iffD1] spectral-dec-space-orthogonal

$\langle \text{compact-op } a' \rangle \langle \text{selfadjoint } a' \rangle \text{simp: })$

show $\langle (\text{spectral-dec-val } a' \ n \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a' \ n) \ *_{\mathbb{C}L} \ \text{spectral-dec-val } a' \ m \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a' \ m = 0 \rangle$

by (auto intro!: * simp: spectral-dec-proj-def adj-Proj)

show $\langle \text{spectral-dec-val } a' \ n \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a' \ n \ o_{\mathbb{C}L} \ (\text{spectral-dec-val } a' \ m \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a' \ m) \ *_{\mathbb{C}} = 0 \rangle$

by (auto intro!: * simp: spectral-dec-proj-def adj-Proj)

qed

also have $\langle \dots = \text{trace-norm } (\sum_{x \in F}. \text{spectral-dec-val } a' \ x \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a' \ x) \rangle$

by (metis (no-types, lifting) a'-def spectral-dec-proj-tc.rep-eq spectral-dec-val-tc.rep-eq from-trace-class-sum norm-trace-class.rep-eq scaleC-trace-class.rep-eq sum.cong)

also have $\langle \dots = \text{trace-norm } (\sum x. \text{if } x \in F \text{ then } \text{spectral-dec-val } a' \ x \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a' \ x \ \text{else } 0) \rangle$

by (simp add: $\langle \text{finite } F \rangle \text{suminf-If-finite-set}$)

also have $\langle \dots = \text{trace-norm } (\sum x. (\text{spectral-dec-val } a' \ x \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a' \ x) \ o_{\mathbb{C}L} \ \text{Proj } R) \rangle$

proof –

have $\langle \text{spectral-dec-proj } a' \ n = \text{spectral-dec-proj } a' \ n \ o_{\mathbb{C}L} \ \text{Proj } R \rangle$ if $\langle n \in F \rangle$ for n

by (auto intro!: Proj-o-Proj-subspace-left[symmetric] SUP-upper that simp: spectral-dec-proj-def R-def)

moreover have $\langle \text{spectral-dec-proj } a' \ n \ o_{\mathbb{C}L} \ \text{Proj } R = 0 \rangle$ if $\langle n \notin F \rangle$ for n

using that

by (auto intro!: orthogonal-spaces-SUP-right spectral-dec-space-orthogonal $\langle \text{compact-op } a' \rangle \langle \text{selfadjoint } a' \rangle$

simp: spectral-dec-proj-def R-def

simp flip: orthogonal-projectors-orthogonal-spaces)

ultimately show *?thesis*
by (*auto intro!*: *arg-cong*[**where** *f=trace-norm*] *suminf-cong*)
qed
also have $\langle \dots = \text{trace-norm} ((\sum x. \text{spectral-dec-val } a' x *_C \text{spectral-dec-proj } a' x) \text{ } o_{CL} \text{ Proj } R) \rangle$
apply (*intro arg-cong*[**where** *f=trace-norm*] *bounded-linear.suminf*[*symmetric*]
bounded-clinear.bounded-linear bounded-clinear-cblinfun-compose-left sums-summable)
using $\langle \text{compact-op } a' \rangle \langle \text{selfadjoint } a' \rangle \text{ spectral-dec-sums}$ **by** *blast*
also have $\langle \dots = \text{trace-norm} (a' \text{ } o_{CL} \text{ Proj } R) \rangle$
using *spectral-dec-sums*[*OF* $\langle \text{compact-op } a' \rangle \langle \text{selfadjoint } a' \rangle$] *sums-unique* **by** *fastforce*
also have $\langle \dots \leq \text{trace-norm } a' * \text{norm} (\text{Proj } R) \rangle$
by (*auto intro!*: *trace-norm-comp-left simp: a'-def*)
also have $\langle \dots \leq \text{trace-norm } a' \rangle$
by (*simp add: mult-left-le norm-Proj-leq1*)
finally show $\langle (\sum x \in F. \text{norm} (\text{spectral-dec-val-tc } a x *_C \text{spectral-dec-proj-tc } a x)) \leq \text{trace-norm } a' \rangle$
by –
qed

lemma *spectral-dec-has-sum-tc*:

assumes $\langle \text{selfadjoint-tc } a \rangle$
shows $\langle ((\lambda n. \text{spectral-dec-val-tc } a n *_C \text{spectral-dec-proj-tc } a n) \text{ has-sum } a) \text{ UNIV} \rangle$
proof –
define *a' b b' where* $\langle a' = \text{from-trace-class } a \rangle$
and $\langle b = (\sum_{\infty} n. \text{spectral-dec-val-tc } a n *_C \text{spectral-dec-proj-tc } a n) \rangle$ **and** $\langle b' = \text{from-trace-class } b \rangle$
have [*simp*]: $\langle \text{compact-op } a' \rangle$
by (*auto intro!*: *trace-class-compact simp: a'-def*)
have [*simp*]: $\langle \text{selfadjoint } a' \rangle$
using *a'-def assms selfadjoint-tc.rep-eq* **by** *blast*
have [*simp*]: $\langle \text{trace-class } b' \rangle$
by (*simp add: b'-def*)
from *spectral-dec-summable-tc*[*OF assms*]
have *has-sum-b*: $\langle ((\lambda n. \text{spectral-dec-val-tc } a n *_C \text{spectral-dec-proj-tc } a n) \text{ has-sum } b) \text{ UNIV} \rangle$
by (*metis abs-summable-summable b-def summable-iff-has-sum-infsum*)
then have $\langle ((\lambda F. \sum n \in F. \text{spectral-dec-val-tc } a n *_C \text{spectral-dec-proj-tc } a n) \longrightarrow b) (\text{finite-subsets-at-top UNIV}) \rangle$
by (*simp add: has-sum-def*)
then have $\langle ((\lambda F. \text{norm} ((\sum n \in F. \text{spectral-dec-val-tc } a n *_C \text{spectral-dec-proj-tc } a n) - b)) \longrightarrow 0) (\text{finite-subsets-at-top UNIV}) \rangle$
using *LIM-zero tendsto-norm-zero* **by** *blast*
then have $\langle ((\lambda F. \text{norm} ((\sum n \in F. \text{spectral-dec-val-tc } a n *_C \text{spectral-dec-proj-tc } a n) - b)) \longrightarrow 0) (\text{filtermap } (\lambda n. \{..<n\}) \text{ sequentially}) \rangle$
by (*meson filterlim-compose filterlim-filtermap filterlim-lessThan-at-top*)
then have $\langle ((\lambda m. \text{norm} ((\sum n \in \{..<m\}. \text{spectral-dec-val-tc } a n *_C \text{spectral-dec-proj-tc } a n) - b)) \longrightarrow 0) \text{ sequentially} \rangle$
by (*simp add: filterlim-filtermap*)
then have $\langle ((\lambda m. \text{trace-norm} ((\sum n \in \{..<m\}. \text{spectral-dec-val } a' n *_C \text{spectral-dec-proj } a' n)) \longrightarrow 0) \text{ sequentially}) \rangle$

– b') $\longrightarrow 0$) *sequentially*
 unfolding a' -def b' -def
 by *transfer*
 then have $\langle (\lambda m. \text{norm } ((\sum n \in \{..<m\}. \text{spectral-dec-val } a' n *_{\mathbb{C}} \text{spectral-dec-proj } a' n) - b'))$
 $\longrightarrow 0$) *sequentially*
 apply (rule *tendsto-0-le*[**where** $K=1$])
 by (*auto intro!*; *eventually-sequentiallyI norm-leq-trace-norm trace-class-minus*
 trace-class-sum trace-class-scaleC spectral-dec-proj-finite-rank
 intro: finite-rank-trace-class)
 then have $\langle (\lambda n. \text{spectral-dec-val } a' n *_{\mathbb{C}} \text{spectral-dec-proj } a' n) \text{ sums } b' \rangle$
 using *LIM-zero-cancel sums-def tendsto-norm-zero-iff* **by** *blast*
 moreover have $\langle (\lambda n. \text{spectral-dec-val } a' n *_{\mathbb{C}} \text{spectral-dec-proj } a' n) \text{ sums } a' \rangle$
 using $\langle \text{compact-op } a' \rangle \langle \text{selfadjoint } a' \rangle$ **by** (rule *spectral-dec-sums*)
 ultimately have $\langle a = b \rangle$
 using a' -def b' -def *from-trace-class-inject sums-unique2* **by** *blast*
 with *has-sum-b* **show** *?thesis*
 by *simp*
qed

lemma *spectral-dec-sums-tc*:

assumes $\langle \text{selfadjoint-tc } a \rangle$
shows $\langle (\lambda n. \text{spectral-dec-val-tc } a n *_{\mathbb{C}} \text{spectral-dec-proj-tc } a n) \text{ sums } a \rangle$
using *assms has-sum-imp-sums spectral-dec-has-sum-tc* **by** *blast*

lift-definition *spectral-dec-vecs-tc* :: $\langle ('a, 'a) \text{ trace-class} \Rightarrow 'a::\text{hilbert-space set} \rangle$ **is**
spectral-dec-vecs.

lemma *compact-from-trace-class*[*iff*]: $\langle \text{compact-op } (\text{from-trace-class } t) \rangle$
by (*auto intro!*; *simp: trace-class-compact*)

lemma *sum-some-onb-of-tc-butterfly*:

assumes $\langle \text{finite-dim-ccsubspace } S \rangle$
shows $\langle (\sum x \in \text{some-onb-of } S. \text{tc-butterfly } x x) = \text{Abs-trace-class } (\text{Proj } S) \rangle$
by (*metis (mono-tags, lifting) assms from-trace-class-inverse from-trace-class-sum sum.cong*
sum-some-onb-of-butterfly tc-butterfly.rep-eq)

lemma *butterfly-spectral-dec-vec-tc-has-sum*:

assumes $\langle t \geq 0 \rangle$
shows $\langle ((\lambda v. \text{tc-butterfly } v v) \text{ has-sum } t) (\text{spectral-dec-vecs-tc } t) \rangle$

proof –

define t' **where** $\langle t' = \text{from-trace-class } t \rangle$
note *power2-csqrt*[*unfolded power2-eq-square, simp*]
note *Reals-cnj-iff*[*simp*]
have [*simp*]: $\langle \text{compact-op } t' \rangle$
by (*simp add: t'-def*)
from *assms* **have** $\langle \text{selfadjoint-tc } t \rangle$
apply *transfer*
apply (rule *comparable-selfadjoint*[*of 0*])

```

by simp-all
have spectral-real[simp]: ⟨spectral-dec-val t' n ∈ ℝ⟩ for n
  apply (rule spectral-dec-val-real)
  using ⟨selfadjoint-tc t⟩ by (auto intro!: trace-class-compact simp: selfadjoint-tc.rep-eq t'-def)

have *: ⟨((λ(n,v). tc-butterfly v v) has-sum t) (SIGMA n:UNIV. (*C) (csqrt (spectral-dec-val t' n))) ' some-onb-of (spectral-dec-space t' n))⟩
proof (rule has-sum-SigmaI[where g=⟨λn. spectral-dec-val t' n *C spectral-dec-proj-tc t n⟩])
  have ⟨spectral-dec-val t' n ≥ 0⟩ for n
    by (simp add: assms from-trace-class-pos spectral-dec-val-nonneg t'-def)
  then have [simp]: ⟨cnj (csqrt (spectral-dec-val t' n)) * csqrt (spectral-dec-val t' n) = spectral-dec-val t' n⟩ for n
    apply (auto simp add: csqrt-of-real-nonneg less-eq-complex-def)
    by (metis of-real-Re of-real-mult spectral-real sqrt-sqrt)
  have sum: ⟨(∑ y∈(λx. csqrt (spectral-dec-val t' n) *C x) ' some-onb-of (spectral-dec-space t' n). tc-butterfly y y) = spectral-dec-val t' n *C spectral-dec-proj-tc t n⟩ for n
    proof (cases ⟨spectral-dec-val t' n = 0⟩)
      case True
        then show ?thesis
          by (metis (mono-tags, lifting) csqrt-0 imageE scaleC-eq-0-iff sum.neutral tc-butterfly-scaleC-left)
      next
      case False
        then have ⟨inj-on (λx. csqrt (spectral-dec-val t' n) *C x) X⟩ for X :: ⟨'a set⟩
          by (meson csqrt-eq-0 inj-scaleC)
        then show ?thesis
          by (simp add: sum.reindex False spectral-dec-space-finite-dim sum-some-onb-of-tc-butterfly spectral-dec-proj-def spectral-dec-proj-tc-def flip: scaleC-sum-right t'-def)
    qed
  then show ⟨((λy. case (n, y) of (n, v) ⇒ tc-butterfly v v) has-sum spectral-dec-val t' n *C spectral-dec-proj-tc t n)
    ((*C) (csqrt (spectral-dec-val t' n))) ' some-onb-of (spectral-dec-space t' n))⟩ for n
    by (auto intro!: has-sum-finiteI finite-imageI some-onb-of-finite-dim spectral-dec-space-finite-dim simp: t'-def)
  show ⟨((λn. spectral-dec-val t' n *C spectral-dec-proj-tc t n) has-sum t) UNIV⟩
    by (auto intro!: spectral-dec-has-sum-tc ⟨selfadjoint-tc t⟩ simp: t'-def simp flip: spectral-dec-val-tc.rep-eq)
  show ⟨(λ(n, v). tc-butterfly v v) summable-on (SIGMA n:UNIV. (*C) (csqrt (spectral-dec-val t' n))) ' some-onb-of (spectral-dec-space t' n))⟩
    proof -
      have inj: ⟨inj-on ((*C) (csqrt (spectral-dec-val t' n))) (some-onb-of (spectral-dec-space t' n))⟩ for n
    proof (cases ⟨spectral-dec-val t' n = 0⟩)
      case True
        then have ⟨spectral-dec-space t' n = 0⟩
          using spectral-dec-space-0 by blast
        then have ⟨some-onb-of (spectral-dec-space t' n) = {}⟩
          using some-onb-of-0 by auto
        then show ?thesis
          by simp
    qed
  
```

```

next
  case False
  then show ?thesis
    by (auto intro!: inj-scaleC)
qed
have 1: ⟨(λx. tc-butterfly x x) abs-summable-on (λxa. csqrt (spectral-dec-val t' n) *C xa) ‘
some-onb-of (spectral-dec-space t' n)⟩ for n
  by (auto intro!: summable-on-finite some-onb-of-finite-dim spectral-dec-space-finite-dim
simp: t'-def)

  have ⟨(λn. cmod (spectral-dec-val t' n) * (∑∞ h∈some-onb-of (spectral-dec-space t' n).
norm (tc-butterfly h h))) abs-summable-on UNIV⟩
  proof -
    have *: ⟨(∑∞ h∈some-onb-of (spectral-dec-space t' n). norm (tc-butterfly h h)) = norm
(spectral-dec-proj-tc t n)⟩ for n
    proof -
      have ⟨(∑∞ h∈some-onb-of (spectral-dec-space t' n). norm (tc-butterfly h h))
= (∑∞ h∈some-onb-of (spectral-dec-space t' n). 1)⟩
      by (simp add: infsum-cong norm-tc-butterfly some-onb-of-norm1)
    also have ⟨... = card (some-onb-of (spectral-dec-space t' n))⟩
      by simp
    also have ⟨... = cdim (space-as-set (spectral-dec-space t' n))⟩
      by (simp add: some-onb-of-card)
    also have ⟨... = norm (spectral-dec-proj-tc t n)⟩
      unfolding t'-def
      apply transfer
      by (metis of-real-eq-iff of-real-of-nat-eq spectral-dec-proj-def spectral-dec-proj-pos
trace-Proj trace-norm-pos)
    finally show ?thesis
      by -
  qed
show ?thesis
  apply (simp add: *)
  by (metis (no-types, lifting) ⟨selfadjoint-tc t⟩ norm-scaleC spectral-dec-summable-tc
spectral-dec-val-tc.rep-eq summable-on-cong t'-def)
qed
then have 2: ⟨(λn. ∑∞ v∈(*C) (csqrt (spectral-dec-val t' n)) ‘ some-onb-of (spectral-dec-space
t' n).
norm (tc-butterfly v v)) abs-summable-on UNIV⟩
  apply (subst infsum-reindex)
  by (auto intro!: inj simp: o-def infsum-cmult-right' norm-mult simp del: real-norm-def)
show ?thesis
  apply (rule abs-summable-summable)
  apply (rule abs-summable-on-Sigma-iff[THEN iffD2])
  using 1 2 by auto
qed
qed
have ⟨((λv. tc-butterfly v v) has-sum t) (∪ n. (*C) (csqrt (spectral-dec-val t' n)) ‘ some-onb-of
(spectral-dec-space t' n))⟩

```

```

proof –
  have **: ⟨(⋃ n. (*C) (csqrt (spectral-dec-val t' n)) ‘ some-onb-of (spectral-dec-space t' n)) =
    snd ‘ (SIGMA n:UNIV. (*C) (csqrt (spectral-dec-val t' n)) ‘ some-onb-of (spectral-dec-space
t' n))⟩
  by force
  have inj: ⟨inj-on snd (SIGMA n:UNIV. (λx. csqrt (spectral-dec-val t' n) *C x) ‘ some-onb-of
(spectral-dec-space t' n))⟩
  proof (rule inj-onI)
    fix nh assume nh: ⟨nh ∈ (SIGMA n:UNIV. (λx. csqrt (spectral-dec-val t' n) *C x) ‘
some-onb-of (spectral-dec-space t' n))⟩
    fix mg assume mg: ⟨mg ∈ (SIGMA m:UNIV. (λx. csqrt (spectral-dec-val t' m) *C x) ‘
some-onb-of (spectral-dec-space t' m))⟩
    assume ⟨snd nh = snd mg⟩
    from nh obtain n h where nh': ⟨nh = (n, csqrt (spectral-dec-val t' n) *C h)⟩ and h: ⟨h
∈ some-onb-of (spectral-dec-space t' n)⟩
    by blast
    from mg obtain m g where mg': ⟨mg = (m, csqrt (spectral-dec-val t' m) *C g)⟩ and g:
⟨g ∈ some-onb-of (spectral-dec-space t' m)⟩
    by blast
    have ⟨n = m⟩
    proof (rule ccontr)
      assume [simp]: ⟨n ≠ m⟩
      from h have val-not-0: ⟨spectral-dec-val t' n ≠ 0⟩
      using some-onb-of-0 spectral-dec-space-0 by fastforce
      from ⟨snd nh = snd mg⟩ nh' mg' have eq: ⟨csqrt (spectral-dec-val t' n) *C h = csqrt
(spectral-dec-val t' m) *C g⟩
      by simp
      from ⟨n ≠ m⟩ have ⟨orthogonal-spaces (spectral-dec-space t' n) (spectral-dec-space t' m)⟩
      apply (rule spectral-dec-space-orthogonal[rotated -1])
      using ⟨selfadjoint-tc t⟩ by (auto intro!: trace-class-compact simp: t'-def selfad-
joint-tc.rep-eq)
      with h g have ⟨is-orthogonal h g⟩
      using orthogonal-spaces-ccspan by fastforce
      then have ⟨is-orthogonal (csqrt (spectral-dec-val t' n) *C h) (csqrt (spectral-dec-val t' m)
*C g)⟩
      by force
      with eq have val-h-0: ⟨csqrt (spectral-dec-val t' n) *C h = 0⟩
      by simp
      with val-not-0 have ⟨h = 0⟩
      by fastforce
      with h show False
      using some-onb-of-is-ortho-set
      by (auto simp: is-ortho-set-def)
    qed
    with ⟨snd nh = snd mg⟩ nh' mg' show ⟨nh = mg⟩
    by simp
  qed
from * show ?thesis
apply (subst **)

```



```

  apply (rule has-sum-reindex[THEN iffD2, rotated])
  by (auto intro!: inj simp: o-def case-prod-unfold)
qed
then show ?thesis
  by (simp add: spectral-dec-vecs-tc.rep-eq spectral-dec-vecs-def flip: t'-def)
qed

```

lemma *spectral-dec-vec-tc-norm-summable*:

```

  assumes  $\langle t \geq 0 \rangle$ 
  shows  $\langle (\lambda v. (\text{norm } v)^2) \text{ summable-on } (\text{spectral-dec-vecs-tc } t) \rangle$ 
proof –
  from butterfly-spectral-dec-vec-tc-has-sum[OF assms]
  have  $\langle (\lambda v. \text{tc-butterfly } v \ v) \text{ summable-on } (\text{spectral-dec-vecs-tc } t) \rangle$ 
  using has-sum-imp-summable by blast
  then have  $\langle (\lambda v. \text{trace-tc } (\text{tc-butterfly } v \ v)) \text{ summable-on } (\text{spectral-dec-vecs-tc } t) \rangle$ 
  apply (rule summable-on-bounded-linear[rotated])
  by (simp add: bounded-clinear.bounded-linear)
  moreover have *:  $\langle \text{trace-tc } (\text{tc-butterfly } v \ v) = \text{of-real } ((\text{norm } v)^2) \rangle$  for  $v :: 'a$ 
  by (metis norm-tc-butterfly norm-tc-pos power2-eq-square tc-butterfly-pos)
  ultimately have  $\langle (\lambda v. \text{complex-of-real } ((\text{norm } v)^2)) \text{ summable-on } (\text{spectral-dec-vecs-tc } t) \rangle$ 
  by simp
  then show ?thesis
  by (smt (verit, ccfv-SIG) norm-of-real summable-on-cong summable-on-iff-abs-summable-on-complex
  zero-le-power2)
qed

```

12.7 More Trace-Class

lemma *finite-rank-tc-dense-aux*: $\langle \text{closure } (\text{Collect finite-rank-tc } :: ('a::\text{hilbert-space}, 'a) \text{ trace-class set}) = \text{UNIV} \rangle$

proof (intro order-top-class.top-le subsetI)

```

  fix  $a :: \langle ('a, 'a) \text{ trace-class} \rangle$ 
  wlog selfadj:  $\langle \text{selfadjoint-tc } a \rangle$  goal  $\langle a \in \text{closure } (\text{Collect finite-rank-tc}) \rangle$  generalizing  $a$ 
proof –
  define  $b \ c$  where  $\langle b = a + \text{adj-tc } a \rangle$  and  $\langle c = i *_C (a - \text{adj-tc } a) \rangle$ 
  have  $\langle \text{adj-tc } b = b \rangle$ 
  unfolding b-def
  apply transfer
  by (simp add: adj-plus)
  have  $\langle \text{adj-tc } c = c \rangle$ 
  unfolding c-def
  apply transfer
  apply (simp add: adj-minus)
  by (metis minus-diff-eq scaleC-right.minus)
  have abc:  $\langle a = (1/2) *_C b + (-i/2) *_C c \rangle$ 
  apply (simp add: b-def c-def)
  by (metis (no-types, lifting) cross3-simps(8) diff-add-cancel group-cancel.add2 scaleC-add-right
  scaleC-half-double)

```

have $\langle b \in \text{closure } (\text{Collect finite-rank-tc}) \rangle$ **and** $\langle c \in \text{closure } (\text{Collect finite-rank-tc}) \rangle$
using $\langle \text{adj-tc } b = b \rangle$ $\langle \text{adj-tc } c = c \rangle$ *hypothesis selfadjoint-tc-def'* **by** *auto*
with *abc* **have** $\langle a \in \text{cspan } (\text{closure } (\text{Collect finite-rank-tc})) \rangle$
by (*metis* *complex-vector.span-add* *complex-vector.span-clauses(1)* *complex-vector.span-clauses(4)*)
also have $\langle \dots \subseteq \text{closure } (\text{cspan } (\text{Collect finite-rank-tc})) \rangle$
by (*simp* *add: closure-mono* *complex-vector.span-minimal* *complex-vector.span-superset*)
also have $\langle \dots = \text{closure } (\text{Collect finite-rank-tc}) \rangle$
by (*metis* *Set.basic-monos(1)* *complex-vector.span-minimal* *complex-vector.span-superset*
csubspace-finite-rank-tc subset-antisym)
finally show *?thesis*
by –
qed
then have $\langle (\lambda n. \text{spectral-dec-val-tc } a \ n \ *_C \ \text{spectral-dec-proj-tc } a \ n) \ \text{sums } a \rangle$
by (*simp* *add: spectral-dec-sums-tc*)
moreover from *selfadj*
have $\langle \text{finite-rank-tc } (\sum i < n. \text{spectral-dec-val-tc } a \ i \ *_C \ \text{spectral-dec-proj-tc } a \ i) \rangle$ **for** *n*
apply (*induction n*)
by (*auto* *intro!: finite-rank-tc-plus* *spectral-dec-proj-tc-finite-rank* *finite-rank-tc-scale*
simp: selfadjoint-tc-def')
ultimately show $\langle a \in \text{closure } (\text{Collect finite-rank-tc}) \rangle$
unfolding *sums-def* *closure-sequential*
apply (*auto* *intro!: simp: sums-def* *closure-sequential*)
by *meson*
qed

lemma *finite-rank-tc-dense*: $\langle \text{closure } (\text{Collect finite-rank-tc} :: ('a::\text{hilbert-space}, 'b::\text{hilbert-space})$
trace-class set) = UNIV \rangle

proof –

have $\langle \text{UNIV} = \text{closure } (\text{Collect finite-rank-tc} :: ('a \times 'b, 'a \times 'b) \text{ trace-class set}) \rangle$

by (*rule* *finite-rank-tc-dense-aux[symmetric]*)

define *l r* **and** *corner* :: $\langle ('a \times 'b, 'a \times 'b) \text{ trace-class} \Rightarrow \rightarrow \rangle$ **where**

$\langle l = \text{cblinfun-left} \rangle$ **and** $\langle r = \text{cblinfun-right} \rangle$ **and**

$\langle \text{corner } t = \text{compose-tcl } (\text{compose-tcr } (r^*) \ t) \ l \rangle$ **for** *t*

have [*iff*]: $\langle \text{bounded-clinear } \text{corner} \rangle$

by (*auto* *intro: bounded-clinear-compose* *compose-tcl.bounded-clinear-left* *compose-tcr.bounded-clinear-right*

simp: corner-def[abs-def])

have $\langle \text{UNIV} = \text{corner } ' \text{UNIV} \rangle$

proof (*intro* *UNIV-eq-I* *range-eqI*)

fix *t*

have $\langle \text{from-trace-class } (\text{corner } (\text{compose-tcl } (\text{compose-tcr } r \ t) \ (l^*)))$

$= (r^* \ o_{CL} \ r) \ o_{CL} \ \text{from-trace-class } t \ o_{CL} \ (l^* \ o_{CL} \ l) \rangle$

by (*simp* *add: corner-def* *compose-tcl.rep-eq* *compose-tcr.rep-eq* *cblinfun-compose-assoc*)

also have $\langle \dots = \text{from-trace-class } t \rangle$

by (*simp* *add: l-def* *r-def*)

finally show $\langle t = \text{corner } (\text{compose-tcl } (\text{compose-tcr } r \ t) \ (l^*)) \rangle$

by (*metis* *from-trace-class-inject*)

qed

```

also have ⟨... = corner ‘ closure (Collect finite-rank-tc)⟩
  by (simp add: finite-rank-tc-dense-aux)
also have ⟨... ⊆ closure (corner ‘ Collect finite-rank-tc)⟩
  by (auto intro!: bounded-clinear.bounded-linear closure-bounded-linear-image-subset)
also have ⟨... ⊆ closure (Collect finite-rank-tc)⟩
proof (intro closure-mono subsetI CollectI)
  fix t assume ⟨t ∈ corner ‘ Collect finite-rank-tc⟩
  then obtain u where ⟨finite-rank-tc u⟩ and tu: ⟨t = corner u⟩
  by blast
  show ⟨finite-rank-tc t⟩
  using ⟨finite-rank-tc u⟩
  by (auto intro!: finite-rank-compose-right[of - l] finite-rank-compose-left[of - ⟨r*⟩]
    simp add: corner-def tu finite-rank-tc.rep-eq compose-tcl.rep-eq compose-tcr.rep-eq)
qed
finally show ?thesis
  by blast
qed

```

hide-fact *finite-rank-tc-dense-aux*

lemma *onb-butterflies-span-trace-class*:

fixes $A :: \langle 'a::\text{chilbert-space set} \rangle$ **and** $B :: \langle 'b::\text{chilbert-space set} \rangle$

assumes $\langle \text{is-onb } A \rangle$ **and** $\langle \text{is-onb } B \rangle$

shows $\langle \text{ccspan } ((\lambda(x, y). \text{tc-butterfly } x \ y) \ (A \times B)) = \top \rangle$

proof —

have $\langle \text{closure } (\text{cspan } ((\lambda(x, y). \text{tc-butterfly } x \ y) \ (A \times B))) \supseteq \text{Collect rank1-tc} \rangle$

proof (rule subsetI)

— This subproof is almost identical to the corresponding one in *finite-rank-dense-compact*, and lengthy. Can they be merged into one subproof?

fix $x :: \langle ('b, 'a) \text{ trace-class} \rangle$ **assume** $\langle x \in \text{Collect rank1-tc} \rangle$

then obtain $a \ b$ **where** $xab: \langle x = \text{tc-butterfly } a \ b \rangle$

apply transfer using rank1-iff-butterfly by fastforce

define f **where** $\langle f \ F \ G = \text{tc-butterfly } (\text{Proj } (\text{ccspan } F) \ a) \ (\text{Proj } (\text{ccspan } G) \ b) \rangle$ **for** $F \ G$

have $\text{lim}: \langle (\text{case-prod } f \ \longrightarrow \ x) \ (\text{finite-subsets-at-top } A \ \times_F \ \text{finite-subsets-at-top } B) \rangle$

proof (rule tendstoI, subst dist-norm)

fix $e :: \text{real}$ **assume** $\langle e > 0 \rangle$

define d **where** $\langle d = (\text{if } \text{norm } a = 0 \ \wedge \ \text{norm } b = 0 \ \text{then } 1$
 $\text{else } e / (\max (\text{norm } a) (\text{norm } b)) / 4) \rangle$

have $d: \langle \text{norm } a * d + \text{norm } a * d + \text{norm } b * d < e \rangle$

proof —

have $\langle \text{norm } a * d \leq e/4 \rangle$

using $\langle e > 0 \rangle$ **apply** (auto simp: d-def)

apply (simp add: divide-le-eq)

by (smt (z3) *Extra-Ordered-Fields.mult-sign-intros(3)* $\langle 0 < e \rangle$ *antisym-conv divide-le-eq less-imp-le linordered-field-class.mult-imp-div-pos-le mult-left-mono nice-ordered-field-class.dense-le nice-ordered-field-class.divide-nonneg-neg nice-ordered-field-class.divide-nonpos-pos nle-le nonzero-mult-div-cancel-left norm-imp-pos-and-ge ordered-field-class.sign-simps(5) split-mult-pos-le*)

```

moreover have  $\langle \text{norm } b * d \leq e/4 \rangle$ 
  using  $\langle e > 0 \rangle$  apply (auto simp: d-def)
  apply (simp add: divide-le-eq)
  by (smt (verit) linordered-field-class.mult-imp-div-pos-le mult-left-mono norm-le-zero-iff
ordered-field-class.sign-simps(5))
  ultimately have  $\langle \text{norm } a * d + \text{norm } a * d + \text{norm } b * d \leq 3 * e / 4 \rangle$ 
    by linarith
  also have  $\langle \dots < e \rangle$ 
    by (simp add:  $\langle 0 < e \rangle$ )
  finally show ?thesis
    by -
qed
have [simp]:  $\langle d > 0 \rangle$ 
  using  $\langle e > 0 \rangle$  apply (auto simp: d-def)
apply (smt (verit, best) nice-ordered-field-class.divide-pos-pos norm-eq-zero norm-not-less-zero)
  by (smt (verit) linordered-field-class.divide-pos-pos zero-less-norm-iff)
from Proj-onb-limit[where  $\psi=a$ , OF assms(1)]
have  $\langle \forall_F F \text{ in finite-subsets-at-top } A. \text{norm } (\text{Proj } (\text{ccspan } F) a - a) < d \rangle$ 
  by (metis Lim-null  $\langle 0 < d \rangle$  order-tendstoD(2) tendsto-norm-zero-iff)
moreover from Proj-onb-limit[where  $\psi=b$ , OF assms(2)]
have  $\langle \forall_F G \text{ in finite-subsets-at-top } B. \text{norm } (\text{Proj } (\text{ccspan } G) b - b) < d \rangle$ 
  by (metis Lim-null  $\langle 0 < d \rangle$  order-tendstoD(2) tendsto-norm-zero-iff)
ultimately have FG-close:  $\langle \forall_F (F, G) \text{ in finite-subsets-at-top } A \times_F \text{ finite-subsets-at-top}$ 
B.
   $\text{norm } (\text{Proj } (\text{ccspan } F) a - a) < d \wedge \text{norm } (\text{Proj } (\text{ccspan } G) b - b) < d \rangle$ 
  unfolding case-prod-beta
  by (rule eventually-prodI)
have fFG-dist:  $\langle \text{norm } (f F G - x) < e \rangle$ 
  if  $\langle \text{norm } (\text{Proj } (\text{ccspan } F) a - a) < d \rangle$  and  $\langle \text{norm } (\text{Proj } (\text{ccspan } G) b - b) < d \rangle$ 
  and  $\langle F \subseteq A \rangle$  and  $\langle G \subseteq B \rangle$  for  $F G$ 
proof -
  have a-split:  $\langle a = \text{Proj } (\text{ccspan } F) a + \text{Proj } (\text{ccspan } (A-F)) a \rangle$ 
  using assms apply (simp add: is-onb-def is-ortho-set-def that Proj-orthog-ccspan-union
flip: cblinfun.add-left)
  apply (subst Proj-orthog-ccspan-union[symmetric])
  apply (metis DiffD1 DiffD2 in-mono that(3))
  using  $\langle F \subseteq A \rangle$  by (auto intro!: simp: Un-absorb1)
  have b-split:  $\langle b = \text{Proj } (\text{ccspan } G) b + \text{Proj } (\text{ccspan } (B-G)) b \rangle$ 
  using assms apply (simp add: is-onb-def is-ortho-set-def that Proj-orthog-ccspan-union
flip: cblinfun.add-left)
  apply (subst Proj-orthog-ccspan-union[symmetric])
  apply (metis DiffD1 DiffD2 in-mono that(4))
  using  $\langle G \subseteq B \rangle$  by (auto intro!: simp: Un-absorb1)
  have n1:  $\langle \text{norm } (f F (B-G)) \leq \text{norm } a * d \rangle$  for  $F$ 
proof -
  have  $\langle \text{norm } (f F (B-G)) \leq \text{norm } a * \text{norm } (\text{Proj } (\text{ccspan } (B-G)) b) \rangle$ 
  by (auto intro!: mult-right-mono is-Proj-reduces-norm simp add: f-def norm-tc-butterfly)
  also have  $\langle \dots \leq \text{norm } a * \text{norm } (\text{Proj } (\text{ccspan } G) b - b) \rangle$ 
  by (metis add-diff-cancel-left' b-split less-eq-real-def norm-minus-commute)

```

```

    also have ⟨... ≤ norm a * d⟩
      by (meson less-eq-real-def mult-left-mono norm-ge-zero that(2))
    finally show ?thesis
      by -
  qed
  have n2: ⟨norm (f (A-F) G) ≤ norm b * d⟩ for G
  proof -
    have ⟨norm (f (A-F) G) ≤ norm b * norm (Proj (ccspan (A-F)) a)⟩
      by (auto intro!: mult-right-mono is-Proj-reduces-norm simp add: f-def norm-tc-butterfly
mult.commute)
    also have ⟨... ≤ norm b * norm (Proj (ccspan F) a - a)⟩
      by (smt (verit, best) a-split add-diff-cancel-left' minus-diff-eq norm-minus-cancel)
    also have ⟨... ≤ norm b * d⟩
      by (meson less-eq-real-def mult-left-mono norm-ge-zero that(1))
    finally show ?thesis
      by -
  qed
  have ⟨norm (f F G - x) = norm (- f F (B-G) - f (A-F) (B-G) - f (A-F) G)⟩
    unfolding xab
    apply (subst a-split, subst b-split)
    by (simp add: f-def tc-butterfly-add-right tc-butterfly-add-left)
  also have ⟨... ≤ norm (f F (B-G)) + norm (f (A-F) (B-G)) + norm (f (A-F) G)⟩
    by (smt (verit, best) norm-minus-cancel norm-triangle-ineq4)
  also have ⟨... ≤ norm a * d + norm a * d + norm b * d⟩
    using n1 n2
    by (meson add-mono-thms-linordered-semiring(1))
  also have ⟨... < e⟩
    by (fact d)
  finally show ?thesis
    by -
  qed
  show ⟨∀ F FG in finite-subsets-at-top A ×F finite-subsets-at-top B. norm (case-prod f FG
- x) < e⟩
    apply (rule eventually-elim2)
    apply (rule eventually-prodI[where P=⟨λF. finite F ∧ F ⊆ A⟩ and Q=⟨λG. finite G
∧ G ⊆ B⟩])
    apply auto[2]
    apply (rule FG-close)
    using fFG-dist by fastforce
  qed
  have nontriv: ⟨finite-subsets-at-top A ×F finite-subsets-at-top B ≠ ⊥⟩
    by (simp add: prod-filter-eq-bot)
  have inside: ⟨∀ F x in finite-subsets-at-top A ×F finite-subsets-at-top B.
case-prod f x ∈ cspan ((λ(ξ,η). tc-butterfly ξ η) ' (A × B))⟩
  proof (rule eventually-mp[where P=⟨λ(F,G). finite F ∧ finite G⟩])
    show ⟨∀ F (F,G) in finite-subsets-at-top A ×F finite-subsets-at-top B. finite F ∧ finite G⟩
      by (smt (verit) case-prod-conv eventually-finite-subsets-at-top-weakI eventually-prod-filter)
    have f-in-span: ⟨f F G ∈ cspan ((λ(ξ,η). tc-butterfly ξ η) ' (A × B))⟩ if [simp]: ⟨finite F⟩
⟨finite G⟩ and ⟨F ⊆ A⟩ ⟨G ⊆ B⟩ for F G

```

proof –

have $\langle \text{Proj } (\text{ccspan } F) \ a \in \text{cspan } F \rangle$
by $(\text{metis } \text{Proj-range } \text{cblinfun-apply-in-image } \text{ccspan-finite } \text{that}(1))$
then obtain r **where** $\text{ProjFsum}: \langle \text{Proj } (\text{ccspan } F) \ a = (\sum_{x \in F}. r \ x \ *_C \ x) \rangle$
apply atomize-elim
using $\text{complex-vector.span-finite}[OF \ \langle \text{finite } F \rangle]$
by auto

have $\langle \text{Proj } (\text{ccspan } G) \ b \in \text{cspan } G \rangle$
by $(\text{metis } \text{Proj-range } \text{cblinfun-apply-in-image } \text{ccspan-finite } \text{that}(2))$
then obtain s **where** $\text{ProjGsum}: \langle \text{Proj } (\text{ccspan } G) \ b = (\sum_{x \in G}. s \ x \ *_C \ x) \rangle$
apply atomize-elim
using $\text{complex-vector.span-finite}[OF \ \langle \text{finite } G \rangle]$
by auto

have $\langle \text{tc-butterfly } \xi \ \eta \in (\lambda(\xi, \eta). \text{tc-butterfly } \xi \ \eta) \ ' (A \times B) \rangle$
if $\langle \eta \in G \rangle$ **and** $\langle \xi \in F \rangle$ **for** $\eta \ \xi$
using $\langle F \subseteq A \rangle \ \langle G \subseteq B \rangle$ **that** **by** $(\text{auto } \text{intro!}: \text{pair-imageI})$
then show $?thesis$
by $(\text{auto } \text{intro!}: \text{complex-vector.span-sum } \text{complex-vector.span-scale}$
 $\text{intro}: \text{complex-vector.span-base}[\text{where } a = \langle \text{tc-butterfly } - \ - \rangle]$
 $\text{simp } \text{add}: \text{f-def } \text{ProjFsum } \text{ProjGsum } \text{tc-butterfly-sum-left } \text{tc-butterfly-sum-right})$

qed

show $\langle \forall_F \ x \ \text{in } \text{finite-subsets-at-top } A \times_F \ \text{finite-subsets-at-top } B.$
 $(\text{case } x \ \text{of } (F, G) \Rightarrow \text{finite } F \wedge \text{finite } G) \longrightarrow (\text{case } x \ \text{of } (F, G) \Rightarrow \text{f } F \ G) \in$
 $\text{cspan } ((\lambda(\xi, \eta). \text{tc-butterfly } \xi \ \eta) \ ' (A \times B)) \rangle$
apply $(\text{rule } \text{eventually-mono})$
apply $(\text{rule } \text{eventually-prodI}[\text{where } P = \langle \lambda F. \text{finite } F \wedge F \subseteq A \rangle \ \text{and } Q = \langle \lambda G. \text{finite } G \wedge$
 $G \subseteq B \rangle])$
by $(\text{auto } \text{intro!}: \text{f-in-span})$

qed

show $\langle x \in \text{closure } (\text{cspan } ((\lambda(\xi, \eta). \text{tc-butterfly } \xi \ \eta) \ ' (A \times B))) \rangle$
using $\text{lim } \text{nontriv } \text{inside}$ **by** $(\text{rule } \text{limit-in-closure})$

qed

moreover have $\langle \text{cspan } (\text{Collect } \text{rank1-tc} :: ('b, 'a) \ \text{trace-class set}) = \text{Collect } \text{finite-rank-tc}$
using $\text{finite-rank-tc-def}'$ **by** fastforce

moreover have $\langle \text{closure } (\text{Collect } \text{finite-rank-tc} :: ('b, 'a) \ \text{trace-class set}) = \text{UNIV} \rangle$
by $(\text{rule } \text{finite-rank-tc-dense})$

ultimately have $\langle \text{closure } (\text{cspan } ((\lambda(x, y). \text{tc-butterfly } x \ y) \ ' (A \times B))) \supseteq \text{UNIV} \rangle$
by $(\text{smt } (\text{verit}, \text{del-insts}) \ \text{Un-UNIV-left } \text{closed-sum-closure-left } \text{closed-sum-cspan } \text{closure-closure}$
 $\text{closure-is-csubspace } \text{complex-vector.span-eq-iff } \text{complex-vector.subspace-span } \text{subset-Un-eq})$

then show $?thesis$
by $(\text{metis } \text{ccspan.abs-eq } \text{ccspan-UNIV } \text{closure-UNIV } \text{complex-vector.span-UNIV } \text{top.extremum-uniqueI})$

qed

lemma $\text{separating-set-tc-butterfly}: \langle \text{separating-set } \text{bounded-clinear } ((\lambda(g, h). \text{tc-butterfly } g \ h) \ ' (UNIV \times UNIV)) \rangle$
apply $(\text{rule } \text{separating-set-mono}[\text{where } S = \langle \lambda(g, h). \text{tc-butterfly } g \ h) \ ' (\text{some-chilbert-basis } \times$
 $\text{some-chilbert-basis}) \rangle])$
by $(\text{auto } \text{intro!}: \text{separating-set-bounded-clinear-dense } \text{onb-butterflies-span-trace-class})$

```

lemma separating-set-tc-butterfly-nested:
  assumes ‹separating-set (bounded-clinear :: (- => 'c::complex-normed-vector) => -) A›
  assumes ‹separating-set (bounded-clinear :: (- => 'c conjugate-space) => -) B›
  shows ‹separating-set (bounded-clinear :: (- => 'c) => -) ((λ(g,h). tc-butterfly g h) ‘(A × B))›
proof –
  from separating-set-tc-butterfly
  have ‹separating-set bounded-clinear ((λ(g,h). tc-butterfly g h) ‘prod.swap ‘(UNIV × UNIV))›
    by simp
  then have ‹separating-set bounded-clinear ((λ(g,h). tc-butterfly h g) ‘(UNIV × UNIV))›
    unfolding image-image by simp
  then have ‹separating-set (bounded-clinear :: (- => 'c) => -) ((λ(g,h). tc-butterfly h g) ‘(B × A))›
    apply (rule separating-set-bounded-sesquilinear-nested)
    apply (rule bounded-sesquilinear-tc-butterfly)
    using assms by auto
  then have ‹separating-set (bounded-clinear :: (- => 'c) => -) ((λ(g,h). tc-butterfly h g) ‘prod.swap ‘(A × B))›
    by (smt (verit, del-insts) SigmaE SigmaI eq-from-separatingI image-iff pair-in-swap-image separating-setI)
    then show ?thesis
    unfolding image-image by simp
qed

```

```

unbundle no cblinfun-syntax

```

```

end

```

13 Weak-Star-Topology – Weak* topology on complex bounded operators

```

theory Weak-Star-Topology

```

```

  imports Trace-Class Weak-Operator-Topology Misc-Tensor-Product-TTS

```

```

begin

```

```

unbundle cblinfun-syntax

```

```

definition weak-star-topology :: ‹('a::hilbert-space =>CL 'b::hilbert-space) topology›

```

```

  where ‹weak-star-topology = pullback-topology UNIV (λx. λt∈Collect trace-class. trace (t oCL x))›

```

```

    ‹(product-topology (λ-. euclidean) (Collect trace-class))›

```

```

lemma open-map-product-topology-reindex:

```

```

  fixes  $\pi :: \langle 'b \Rightarrow 'a \rangle$ 

```

```

  assumes bij-π: ‹bij-betw π B A› and ST: ‹ $\bigwedge x. x \in B \implies S x = T (\pi x)$ ›

```

```

  assumes g-def: ‹ $\bigwedge f. g f = \text{restrict } (f \circ \pi) B$ ›

```

```

  shows ‹open-map (product-topology T A) (product-topology S B) g›

```

proof –

define $\pi' g'$ **where** $\langle \pi' = \text{inv-into } B \pi \rangle$ **and** $\langle g' f = \text{restrict } (f \circ \pi') A \rangle$ **for** $f :: \langle 'b \Rightarrow 'c \rangle$

have $\text{bij-}g: \langle \text{bij-betw } g (Pi_E A V) (Pi_E B (V \circ \pi)) \rangle$ **for** V

apply (rule *bij-betw-byWitness*[**where** $f'=g'$])

subgoal

unfolding $g'\text{-def } g\text{-def } \pi'\text{-def}$

by (smt (verit, best) *PiE-restrict bij- π bij-betw-imp-surj-on bij-betw-inv-into-right comp-eq-dest-lhs inv-into-into restrict-def restrict-ext*)

subgoal

unfolding $g'\text{-def } g\text{-def } \pi'\text{-def}$

by (smt (verit, ccfv-SIG) *PiE-restrict bij- π bij-betwE bij-betw-inv-into-left comp-apply restrict-apply restrict-ext*)

subgoal

unfolding $g'\text{-def } g\text{-def } \pi'\text{-def}$

using *PiE-mem bij- π bij-betw-imp-surj-on* **by** *fastforce*

subgoal

unfolding $g'\text{-def } g\text{-def } \pi'\text{-def}$

by (smt (verit, best) *PiE-mem bij- π bij-betw-iff-bijections bij-betw-inv-into-left comp-def image-subset-iff restrict-PiE-iff*)

done

have $\text{open-}gU: \langle \text{openin } (\text{product-topology } S B) (g ' U) \rangle$ **if** $\langle \text{openin } (\text{product-topology } T A) U \rangle$

for U

proof –

from *product-topology-open-contains-basis*[*OF that*]

obtain V **where** $xAV: \langle x \in Pi_E A (V x) \rangle$ **and** $\text{open}V: \langle \text{openin } (T a) (V x a) \rangle$ **and** $\text{finite}V: \langle \text{finite } \{a. V x a \neq \text{topspace } (T a)\} \rangle$

and $AVU: \langle Pi_E A (V x) \subseteq U \rangle$ **if** $\langle x \in U \rangle$ **for** $x a$

apply *atomize-elim*

apply (rule *choice4*)

by *meson*

define V' **where** $\langle V' x b = (\text{if } b \in B \text{ then } V x (\pi b) \text{ else } \text{topspace } (S b)) \rangle$ **for** $b x$

have $PiEV': \langle Pi_E B (V x \circ \pi) = Pi_E B (V' x) \rangle$ **for** x

by (*metis* (*mono-tags, opaque-lifting*) *PiE-cong V'-def comp-def*)

from $xAV AVU$ **have** $AVU': \langle (\bigcup x \in U. Pi_E A (V x)) = U \rangle$

by *blast*

have $\text{open}Vb: \langle \text{openin } (S b) (V' x b) \rangle$ **if** [*simp*]: $\langle x \in U \rangle$ **for** $x b$

by (*auto simp: ST V'-def intro!: openV*)

have $\langle \text{bij-betw } \pi' \{a \in A. V x a \neq \text{topspace } (T a)\} \{b \in B. (V x \circ \pi) b \neq \text{topspace } (S b)\} \rangle$ **for**

x

apply (rule *bij-betw-byWitness*[**where** $f'=\pi$])

apply *simp*

apply (*metis* $\pi'\text{-def } \text{bij-}\pi$ *bij-betw-inv-into-right*)

using $\pi'\text{-def } \text{bij-}\pi$ *bij-betw-imp-inj-on* **apply** *fastforce*

apply (smt (verit, best) *ST $\pi'\text{-def } \text{bij-}\pi$ bij-betw-imp-surj-on comp-apply f-inv-into-f image-Collect-subsetI inv-into-into mem-Collect-eq*)

using *ST bij- π bij-betwE* **by** *fastforce*

then have $\langle \text{finite } \{b \in B. (V x \circ \pi) b \neq \text{topspace } (S b)\} \rangle$ **if** $\langle x \in U \rangle$ **for** x

apply (rule *bij-betw-finite*[*THEN iffD1*])


```

    using that finiteV
    by simp
    also have ⟨{b∈B. (V x ∘ π) b ≠ topspace (S b)} = {b. V' x b ≠ topspace (S b)}⟩ if ⟨x ∈
U⟩ for x
    by (auto simp: V'-def)
    finally have finiteVπ: ⟨finite {b. V' x b ≠ topspace (S b)}⟩ if ⟨x ∈ U⟩ for x
    using that by -
    from openVb finiteVπ
    have ⟨openin (product-topology S B) (PiE B (V' x))⟩ if [simp]: ⟨x ∈ U⟩ for x
    by (auto intro!: product-topology-basis)
    with bij-g PiEV' have ⟨openin (product-topology S B) (g ' (PiE A (V x)))⟩ if ⟨x ∈ U⟩ for x
    by (metis bij-betw-imp-surj-on that)
    then have ⟨openin (product-topology S B) (⋃x∈U. (g ' (PiE A (V x))))⟩
    by blast
    with AVU' show ⟨openin (product-topology S B) (g ' U)⟩
    by (metis image-UN)
qed
show ⟨open-map (product-topology T A) (product-topology S B) g⟩
by (simp add: open-gU open-map-def)
qed

```

lemma *homeomorphic-map-product-topology-reindex*:

```

fixes π :: ⟨'b ⇒ 'a⟩
assumes bij-π: ⟨bij-betw π B A⟩ and ST: ⟨⋀x. x∈B ⇒ S x = T (π x)⟩
assumes g-def: ⟨⋀f. g f = restrict (f ∘ π) B⟩
shows ⟨homeomorphic-map (product-topology T A) (product-topology S B) g⟩
proof (rule bijective-open-imp-homeomorphic-map)
show open-map: ⟨open-map (product-topology T A) (product-topology S B) g⟩
using assms by (rule open-map-product-topology-reindex)
define π' g' where ⟨π' = inv-into B π⟩ and ⟨g' f = restrict (f ∘ π') A⟩ for f :: ⟨'b ⇒ 'c⟩
have ⟨bij-betw π' A B⟩
by (simp add: π'-def bij-π bij-betw-inv-into)

```

```

have l1: ⟨x ∈ (λx. restrict (x ∘ π) B) ' (ΠE i∈A. topspace (T i))⟩ if ⟨x ∈ (ΠE i∈B. topspace
(S i))⟩ for x

```

```

proof -
have ⟨g' x ∈ (ΠE i∈A. topspace (T i))⟩
by (smt (z3) g'-def PiE-mem π'-def assms(1) assms(2) bij-betw-imp-surj-on bij-betw-inv-into-right
comp-apply inv-into-into restrict-PiE-iff that)
moreover have ⟨x = restrict (g' x ∘ π) B⟩
by (smt (verit) PiE-restrict π'-def assms(1) bij-betwE bij-betw-inv-into-left comp-apply
restrict-apply restrict-ext that g'-def)
ultimately show ?thesis
by (intro rev-image-eqI)

```

```

qed
show topspace: ⟨g ' topspace (product-topology T A) = topspace (product-topology S B)⟩
using l1 assms unfolding g-def [abs-def] topspace-product-topology
by (auto simp: bij-betw-def)

```

```

show ⟨inj-on g (topspace (product-topology T A))⟩
  apply (simp add: g-def[abs-def])
  by (smt (verit) PiE-ext assms(1) bij-betw-iff-bijections comp-apply inj-on-def restrict-apply')

have open-map-g': ⟨open-map (product-topology S B) (product-topology T A) g'⟩
  using ⟨bij-betw π' A B⟩ apply (rule open-map-product-topology-reindex)
  apply (metis ST π'-def big-π bij-betw-imp-surj-on bij-betw-inv-into-right inv-into-into)
  using g'-def by blast
have g'g: ⟨g' (g x) = x⟩ if ⟨x ∈ topspace (product-topology T A)⟩ for x
  using that unfolding g'-def g-def topspace-product-topology
  by (smt (verit) PiE-restrict ⟨bij-betw π' A B⟩ π'-def big-π bij-betwE
    bij-betw-inv-into-right comp-def restrict-apply' restrict-ext)
have gg': ⟨g (g' x) = x⟩ if ⟨x ∈ topspace (product-topology S B)⟩ for x
  unfolding g'-def g-def
  by (metis (no-types, lifting) g'-def f-inv-into-f g'g g-def inv-into-into that topspace)

from open-map-g'
have ⟨openin (product-topology T A) (g' ' U)⟩ if ⟨openin (product-topology S B) U⟩ for U
  using open-map-def that by blast
also have ⟨g' ' U = (g - ' U) ∩ (topspace (product-topology T A))⟩ if ⟨openin (product-topology
S B) U⟩ for U
proof -
  from that
  have U-top: ⟨U ⊆ topspace (product-topology S B)⟩
    using openin-subset by blast
  from topspace
  have topspace': ⟨topspace (product-topology T A) = g' ' topspace (product-topology S B)⟩
    by (metis bij-betw-byWitness bij-betw-def calculation g'g gg' openin-subset openin-topspace)
  show ?thesis
    unfolding topspace'
    using U-top gg'
    by auto
qed
finally have open-gU2: ⟨openin (product-topology T A) ((g - ' U) ∩ (topspace (product-topology
T A)))⟩
  if ⟨openin (product-topology S B) U⟩ for U
  using that by blast

then show ⟨continuous-map (product-topology T A) (product-topology S B) g'⟩
  by (smt (verit, best) g'g image-iff open-eq-continuous-inverse-map open-map-g' topspace)
qed

lemma weak-star-topology-def':
  ⟨weak-star-topology = pullback-topology UNIV (λx t. trace (from-trace-class t oCL x)) eu-
  clidean⟩
proof -
  define fg where ⟨fx = (λt ∈ Collect trace-class. trace (t oCL x))⟩ and ⟨gf' = f' o from-trace-class⟩
  for x :: ⟨'a ⇒CL 'b⟩ and f' :: ⟨'b ⇒CL 'a ⇒ complex⟩

```

```

have ⟨homeomorphic-map (product-topology (λ-. euclidean) (Collect trace-class)) (product-topology
(λ-. euclidean) UNIV) g⟩
  unfolding g-def[abs-def]
  apply (rule homeomorphic-map-product-topology-reindex[where π=from-trace-class])
  subgoal
  by (smt (verit, best) UNIV-I bij-betwI' from-trace-class from-trace-class-cases from-trace-class-inject)
  by auto
then have homeo-g: ⟨homeomorphic-map (product-topology (λ-. euclidean) (Collect trace-class))
euclidean g⟩
  by (simp add: euclidean-product-topology)
have ⟨weak-star-topology = pullback-topology UNIV f (product-topology (λ-. euclidean) (Collect
trace-class))⟩
  by (simp add: weak-star-topology-def pullback-topology-homeo-cong homeo-g f-def[abs-def])
also have ⟨... = pullback-topology UNIV (g o f) euclidean⟩
  by (subst pullback-topology-homeo-cong)
  (auto simp add: homeo-g f-def[abs-def] split: if-splits)
also have ⟨... = pullback-topology UNIV (λx t. trace (from-trace-class t oCL x)) euclidean⟩
  by (auto simp: f-def[abs-def] g-def[abs-def] o-def)
finally show ?thesis
  by -
qed

```

```

lemma weak-star-topology-topospace[simp]:
  topspace weak-star-topology = UNIV
  unfolding weak-star-topology-def topspace-pullback-topology topspace-euclidean by auto

```

```

lemma weak-star-topology-basis':
  fixes f::('a::hilbert-space ⇒CL 'b::hilbert-space) and U::'i ⇒ complex set and t::'i ⇒ ('b,'a)
trace-class
  assumes finite I ∧ i. i ∈ I ⇒ open (U i)
  shows openin weak-star-topology {f. ∀ i ∈ I. trace (from-trace-class (t i) oCL f) ∈ U i}
proof -
  have 1: open {g. ∀ i ∈ I. g (t i) ∈ U i}
  using assms by (rule product-topology-basis')
  show ?thesis
  unfolding weak-star-topology-def'
  apply (subst openin-pullback-topology)
  apply (intro exI conjI)
  using 1 by auto
qed

```

```

lemma weak-star-topology-basis:
  fixes f::('a::hilbert-space ⇒CL 'b::hilbert-space) and U::'i ⇒ complex set and t::'i ⇒ ('b
⇒CL 'a)
  assumes finite I ∧ i. i ∈ I ⇒ open (U i)
  assumes tc: ⟨∧ i. i ∈ I ⇒ trace-class (t i)⟩
  shows openin weak-star-topology {f. ∀ i ∈ I. trace (t i oCL f) ∈ U i}
proof -
  obtain t' where tt': ⟨t i = from-trace-class (t' i)⟩ if ⟨i ∈ I⟩ for i

```

by (atomize-elim, rule choice) (use tc from-trace-class-cases in blast)
 show ?thesis
 using assms by (auto simp: tt' o-def intro!: weak-star-topology-basis')
 qed

lemma wot-weaker-than-weak-star:

continuous-map weak-star-topology cweak-operator-topology (λf. f)
 unfolding weak-star-topology-def cweak-operator-topology-def
 proof (rule continuous-map-pullback-both)
 define g' :: ⟨('b ⇒_{CL} 'a ⇒ complex) ⇒ 'b × 'a ⇒ complex⟩ where
 ⟨g' f = (λ(x,y). f (butterfly y x))⟩ for f
 show ⟨(λx. λt∈Collect trace-class. trace (t o_{CL} x)) - 'topspace (product-topology (λ-. euclidean) (Collect trace-class)) ∩ UNIV
 ⊆ (λf. f) - 'UNIV⟩
 by simp
 show ⟨g' (λt∈Collect trace-class. trace (t o_{CL} x)) = (λ(xa, y). xa •_C (x *_V y))⟩
 if ⟨(λt∈Collect trace-class. trace (t o_{CL} x)) ∈ topspace (product-topology (λ-. euclidean) (Collect trace-class))⟩
 for x
 by (auto intro!: ext simp: g'-def trace-butterfly-comp)
 show ⟨continuous-map (product-topology (λ-. euclidean) (Collect trace-class)) euclidean g'⟩
 apply (subst euclidean-product-topology[symmetric])
 apply (rule continuous-map-coordinatewise-then-product)
 subgoal for i
 unfolding g'-def case-prod-unfold
 by (metis continuous-map-product-projection mem-Collect-eq trace-class-butterfly)
 subgoal
 by (auto simp: g'-def[abs-def])
 done
 qed

lemma wot-weaker-than-weak-star':

⟨openin cweak-operator-topology U ⇒ openin weak-star-topology U⟩
 using wot-weaker-than-weak-star[where 'a='a and 'b='b]
 by (auto simp: continuous-map-def weak-star-topology-tospace)

lemma weak-star-topology-continuous-duality':

shows continuous-map weak-star-topology euclidean (λx. trace (from-trace-class t o_{CL} x))
 proof –
 have continuous-map weak-star-topology euclidean ((λf. f t) o (λx t. trace (from-trace-class t o_{CL} x)))
 unfolding weak-star-topology-def' apply (rule continuous-map-pullback)
 using continuous-on-product-coordinates by fastforce
 then show ?thesis unfolding comp-def by simp
 qed

lemma weak-star-topology-continuous-duality:

assumes ⟨trace-class t⟩
 shows continuous-map weak-star-topology euclidean (λx. trace (t o_{CL} x))

by (metis assms from-trace-class-cases mem-Collect-eq weak-star-topology-continuous-duality')

lemma continuous-on-weak-star-topo-iff-coordinatewise:

fixes $f :: \langle 'a \Rightarrow 'b :: \text{hilbert-space} \Rightarrow_{CL} 'c :: \text{hilbert-space} \rangle$

shows continuous-map T weak-star-topology f

$\longleftrightarrow (\forall t. \text{trace-class } t \longrightarrow \text{continuous-map } T \text{ euclidean } (\lambda x. \text{trace } (t \circ_{CL} f x)))$

proof (intro iffI allI impI)

fix $t :: \langle 'c \Rightarrow_{CL} 'b \rangle$

assume $\langle \text{trace-class } t \rangle$

assume continuous-map T weak-star-topology f

with continuous-map-compose[OF this weak-star-topology-continuous-duality, OF $\langle \text{trace-class } t \rangle$]

have continuous-map T euclidean $((\lambda x. \text{trace } (t \circ_{CL} x)) \circ f)$

by simp

then show continuous-map T euclidean $(\lambda x. \text{trace } (t \circ_{CL} f x))$

unfolding comp-def by auto

next

assume $\langle \forall t. \text{trace-class } t \longrightarrow \text{continuous-map } T \text{ euclidean } (\lambda x. \text{trace } (t \circ_{CL} f x)) \rangle$

then have $\langle \text{continuous-map } T \text{ euclidean } (\lambda x. \text{trace } (\text{from-trace-class } t \circ_{CL} f x)) \rangle$ for t

by auto

then have *: continuous-map T euclidean $((\lambda x t. \text{trace } (\text{from-trace-class } t \circ_{CL} x)) \circ f)$

by (auto simp flip: euclidean-product-topology simp: o-def)

show continuous-map T weak-star-topology f

unfolding weak-star-topology-def'

apply (rule continuous-map-pullback')

by (auto simp add: *)

qed

lemma weak-star-topology-weaker-than-euclidean:

continuous-map euclidean weak-star-topology $(\lambda f. f)$

apply (subst continuous-on-weak-star-topo-iff-coordinatewise)

by (auto intro!: linear-continuous-on bounded-clinear.bounded-linear bounded-clinear-trace-duality)

typedef (overloaded) $('a, 'b) \text{ cblinfun-weak-star} = \langle UNIV :: ('a :: \text{complex-normed-vector} \Rightarrow_{CL} 'b :: \text{complex-normed-vector}) \text{ set} \rangle$

morphisms from-weak-star to-weak-star ..

setup-lifting type-definition-cblinfun-weak-star

lift-definition id-weak-star :: $\langle ('a :: \text{complex-normed-vector}, 'a) \text{ cblinfun-weak-star} \rangle$ is id-cblinfun

.

instantiation cblinfun-weak-star :: $(\text{complex-normed-vector}, \text{complex-normed-vector}) \text{ complex-vector}$

begin

lift-definition scaleC-cblinfun-weak-star :: $\langle \text{complex} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b)$

cblinfun-weak-star

is $\langle \text{scaleC} \rangle$.

lift-definition uminus-cblinfun-weak-star :: $\langle ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \rangle$

is uminus .

lift-definition *zero-cblinfun-weak-star* :: $\langle ('a, 'b) \text{ cblinfun-weak-star} \rangle$ **is** *0* .
lift-definition *minus-cblinfun-weak-star* :: $\langle ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \rangle$ **is** *minus* .
lift-definition *plus-cblinfun-weak-star* :: $\langle ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \rangle$ **is** *plus* .
lift-definition *scaleR-cblinfun-weak-star* :: $\langle \text{real} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \rangle$ **is** *scaleR* .
instance
 by (*intro-classes; transfer*) (*auto simp add: scaleR-scaleC scaleC-add-right scaleC-add-left*)
end

instantiation *cblinfun-weak-star* :: (*hilbert-space, hilbert-space*) *topological-space* **begin**
lift-definition *open-cblinfun-weak-star* :: $\langle ('a, 'b) \text{ cblinfun-weak-star set} \Rightarrow \text{bool} \rangle$ **is** $\langle \text{openin weak-star-topology} \rangle$.
instance
proof *intro-classes*
 show $\langle \text{open} (UNIV :: ('a, 'b) \text{ cblinfun-weak-star set}) \rangle$
 by *transfer (metis weak-star-topology-topspace openin-topspace)*
 show $\langle \text{open } S \Longrightarrow \text{open } T \Longrightarrow \text{open} (S \cap T) \rangle$ **for** $S T :: \langle ('a, 'b) \text{ cblinfun-weak-star set} \rangle$
 by *transfer auto*
 show $\langle \forall S \in K. \text{open } S \Longrightarrow \text{open} (\bigcup K) \rangle$ **for** $K :: \langle ('a, 'b) \text{ cblinfun-weak-star set set} \rangle$
 by *transfer auto*
qed
end

lemma *transfer-nhds-weak-star-topology[transfer-rule]*:
 includes *lifting-syntax*
 shows $\langle (\text{cr-cblinfun-weak-star} ==> \text{rel-filter cr-cblinfun-weak-star}) (\text{nhdsin weak-star-topology nhds}) \rangle$
proof –
 have (*cr-cblinfun-weak-star ==> rel-filter cr-cblinfun-weak-star*)
 ($\lambda a. \bigcap (\text{principal } \{S. \text{openin weak-star-topology } S \wedge a \in S\})$)
 ($\lambda a. \bigcap (\text{principal } \{S. \text{open } S \wedge a \in S\})$)
 by *transfer-prover*
 thus *?thesis*
 unfolding *nhds-def nhdsin-def weak-star-topology-topspace* **by** *simp*
qed

lemma *limitin-weak-star-topology'*:
 $\langle \text{limitin weak-star-topology } f l F \longleftrightarrow (\forall t. ((\lambda j. \text{trace} (\text{from-trace-class } t \text{ o}_{CL} f j)) \longrightarrow \text{trace} (\text{from-trace-class } t \text{ o}_{CL} l)) F) \rangle$
 by (*simp add: weak-star-topology-def' limitin-pullback-topology tendsto-coordinatewise*)

lemma *limitin-weak-star-topology*:
 $\langle \text{limitin weak-star-topology } f l F \longleftrightarrow (\forall t. \text{trace-class } t \longrightarrow ((\lambda j. \text{trace} (t \text{ o}_{CL} f j)) \longrightarrow \text{trace} (t \text{ o}_{CL} l)) F) \rangle$
 by (*smt (z3) eventually-mono from-trace-class from-trace-class-cases limitin-weak-star-topology' mem-Collect-eq tendsto-def*)

lemma *filterlim-weak-star-topology*:
 ‹*filterlim* *f* (*nhdsin* *weak-star-topology* *l*) = *limitin* *weak-star-topology* *f* *l*›
 by (*auto simp: weak-star-topology-topospace simp flip: filterlim-nhdsin-iff-limitin*)

lemma *openin-weak-star-topology'*: ‹*openin* *weak-star-topology* *U* \longleftrightarrow ($\exists V$. *open* *V* \wedge *U* = (λx *t*. *trace* (*from-trace-class* *t* *o*_{CL} *x*)) - ' *V*)›
 by (*simp add: weak-star-topology-def' openin-pullback-topology*)

lemma *hausdorff-weak-star*[*simp*]: ‹*Hausdorff-space* *weak-star-topology*›
 by (*metis cweak-operator-topology-topospace hausdorff-cweak-operator-topology*
Hausdorff-space-def weak-star-topology-topospace wot-weaker-than-weak-star')

lemma *Domainp-cr-cblinfun-weak-star*[*simp*]: ‹*Domainp* *cr-cblinfun-weak-star* = (λ -. *True*)›
 by (*metis (no-types, opaque-lifting) DomainPI cblinfun-weak-star.left-total left-totalE*)

lemma *Rangep-cr-cblinfun-weak-star*[*simp*]: ‹*Rangep* *cr-cblinfun-weak-star* = (λ -. *True*)›
 by (*meson RangePI cr-cblinfun-weak-star-def*)

lemma *transfer-euclidean-weak-star-topology*[*transfer-rule*]:
 includes *lifting-syntax*
 shows ‹(*rel-topology* *cr-cblinfun-weak-star*) *weak-star-topology* *euclidean*›
proof (*unfold rel-topology-def, intro conjI allI impI*)
 show ‹(*rel-set* *cr-cblinfun-weak-star* \implies (=)) (*openin* *weak-star-topology*) (*openin* *euclidean*)›
 unfolding *rel-fun-def* *rel-set-def* *open-openin* [*symmetric*] *cr-cblinfun-weak-star-def*
 by (*transfer, intro allI impI arg-cong[of - - openin x for x] blast*)
next
 fix *U* :: ‹('a \Rightarrow _{CL} 'b) *set*›
 assume ‹*openin* *weak-star-topology* *U*›
 show ‹*Domainp* (*rel-set* *cr-cblinfun-weak-star*) *U*›
 by (*simp add: Domainp-set*)
next
 fix *U* :: ‹('a, 'b) *cblinfun-weak-star* *set*›
 assume ‹*openin* *euclidean* *U*›
 show ‹*Rangep* (*rel-set* *cr-cblinfun-weak-star*) *U*›
 by (*simp add: Rangep-set*)
qed

instance *cblinfun-weak-star* :: (*hilbert-space*, *hilbert-space*) *t2-space*
 apply (*rule hausdorff-OFCLASS-t2-space*)
 apply *transfer*
 by (*rule hausdorff-weak-star*)

lemma *weak-star-topology-plus-cont*: $\langle LIM (x,y) \text{ nhdsin weak-star-topology } a \times_F \text{ nhdsin weak-star-topology } b. \rangle$

$x + y :> \text{nhdsin weak-star-topology } (a + b)$

proof –

have *trace-plus*: $\langle \text{trace } (t \text{ } o_{CL} (a + b)) = \text{trace } (t \text{ } o_{CL} a) + \text{trace } (t \text{ } o_{CL} b) \rangle$ **if** $\langle \text{trace-class } t \rangle$
for $t :: \langle 'b \Rightarrow_{CL} 'a \rangle$ **and** $a \ b$

by (*auto simp: cblinfun-compose-add-right trace-plus that trace-class-comp-left*)

show *?thesis*

unfolding *weak-star-topology-def'*

by (*rule pullback-topology-bi-cont[where f'=plus]*)

(*auto simp: trace-plus case-prod-unfold tendsto-add-Pair*)

qed

instance *cblinfun-weak-star* :: (*hilbert-space, hilbert-space*) *topological-group-add*

proof *intro-classes*

show $\langle ((\lambda x. \text{fst } x + \text{snd } x) \longrightarrow a + b) (\text{nhds } a \times_F \text{nhds } b) \rangle$ **for** $a \ b :: \langle ('a, 'b) \text{ cblinfun-weak-star} \rangle$

apply *transfer*

using *weak-star-topology-plus-cont*

by (*auto simp: case-prod-unfold*)

have $\langle \text{continuous-map weak-star-topology euclidean } (\lambda x. \text{trace } (t \text{ } o_{CL} - x)) \rangle$ **if** $\langle \text{trace-class } t \rangle$
for $t :: \langle 'b \Rightarrow_{CL} 'a \rangle$

using *weak-star-topology-continuous-duality[of <-t>]*

by (*auto simp: cblinfun-compose-uminus-left cblinfun-compose-uminus-right intro!: that trace-class-uminus*)

then have $*$: $\langle \text{continuous-map weak-star-topology weak-star-topology } (\text{uminus} :: ('a \Rightarrow_{CL} 'b) \Rightarrow -) \rangle$

by (*auto simp: continuous-on-weak-star-topo-iff-coordinatewise*)

show $\langle (\text{uminus} \longrightarrow - a) (\text{nhds } a) \rangle$ **for** $a :: \langle ('a, 'b) \text{ cblinfun-weak-star} \rangle$

apply (*subst tendsto-at-iff-tendsto-nhds[symmetric]*)

apply (*subst isCont-def[symmetric]*)

apply (*rule continuous-on-interior[where S=UNIV]*)

apply (*subst continuous-map-iff-continuous2[symmetric]*)

apply *transfer*

using $*$ **by** *auto*

qed

lemma *continuous-map-left-comp-weak-star*:

$\langle \text{continuous-map weak-star-topology weak-star-topology } (\lambda a :: 'a :: \text{hilbert-space} \Rightarrow_{CL} -. b \text{ } o_{CL} a) \rangle$

for $b :: \langle 'b :: \text{hilbert-space} \Rightarrow_{CL} 'c :: \text{hilbert-space} \rangle$

proof (*unfold weak-star-topology-def, rule continuous-map-pullback-both*)

define $g' :: \langle ('b \Rightarrow_{CL} 'a \Rightarrow \text{complex}) \Rightarrow ('c \Rightarrow_{CL} 'a \Rightarrow \text{complex}) \rangle$ **where**

$\langle g' f = (\lambda t \in \text{Collect trace-class. } f (t \text{ } o_{CL} b)) \rangle$ **for** f

show $\langle (\lambda x. \lambda t \in \text{Collect trace-class. } \text{trace } (t \text{ } o_{CL} x)) - ' \text{topspace } (\text{product-topology } (\lambda -. \text{euclidean}) (\text{Collect trace-class})) \cap \text{UNIV} \rangle$

$\subseteq (o_{CL}) b - ' \text{UNIV} \rangle$


```

    by simp
    show ⟨g' (λt∈Collect trace-class. trace (t oCL x)) = (λt∈Collect trace-class. trace (t oCL (b
oCL x)))⟩ for x
    by (auto intro!: ext simp: g'-def[abs-def] cblinfun-compose-assoc trace-class-comp-left)
    show ⟨continuous-map (product-topology (λ-. euclidean) (Collect trace-class))
(product-topology (λ-. euclidean) (Collect trace-class)) g'⟩
    apply (rule continuous-map-coordinatewise-then-product)
    subgoal for i
      unfolding g'-def
      apply (subst restrict-apply')
      subgoal by simp
      subgoal by (metis continuous-map-product-projection mem-Collect-eq trace-class-comp-left)
      done
    subgoal by (auto simp: g'-def[abs-def])
    done
qed

```

lemma *continuous-map-right-comp-weak-star*:

⟨continuous-map weak-star-topology weak-star-topology (λb::'b::hilbert-space ⇒_{CL} -. b o_{CL} a)⟩

```

    for a :: ⟨'a::hilbert-space ⇒CL 'b::hilbert-space⟩
proof (subst weak-star-topology-def, subst weak-star-topology-def, rule continuous-map-pullback-both)
    define g' :: ⟨('c ⇒CL 'b ⇒ complex) ⇒ ('c ⇒CL 'a ⇒ complex)⟩ where
      ⟨g' f = (λt∈Collect trace-class. f (a oCL t))⟩ for f
    show ⟨(λx. λt∈Collect trace-class. trace (t oCL x)) - 'topspace (product-topology (λ-. eu-
clidean) (Collect trace-class)) ∩ UNIV
      ⊆ (λb. b oCL a) - 'UNIV⟩
    by simp
    have *: trace (a oCL y oCL x) = trace (y oCL (x oCL a)) if trace-class y for x :: 'b ⇒CL 'c
and y :: 'c ⇒CL 'a
    by (simp add: circularity-of-trace simp-a-oCL-b that trace-class-comp-left)
    show ⟨g' (λt∈Collect trace-class. trace (t oCL x)) = (λt∈Collect trace-class. trace (t oCL (x
oCL a)))⟩ for x
    by (auto intro!: ext simp: g'-def[abs-def] trace-class-comp-right *)
    show ⟨continuous-map (product-topology (λ-. euclidean) (Collect trace-class))
(product-topology (λ-. euclidean) (Collect trace-class)) g'⟩
    apply (rule continuous-map-coordinatewise-then-product)
    subgoal for i
      unfolding g'-def mem-Collect-eq
      apply (subst restrict-apply')
      subgoal by simp
      subgoal
        by (metis continuous-map-product-projection mem-Collect-eq trace-class-comp-right)
      done
    subgoal by (auto simp: g'-def[abs-def])
    done
qed

```

lemma *continuous-map-scaleC-weak-star*: ⟨continuous-map weak-star-topology weak-star-topology

```

(scaleC c)
  apply (subst asm-rl[of ‹scaleC c = (oCL) (c *C id-cblinfun)›])
  subgoal by auto
  subgoal by (rule continuous-map-left-comp-weak-star)
  done

lemma continuous-scaleC-weak-star: ‹continuous-on X (scaleC c :: (-,-) cblinfun-weak-star ⇒
-›)
  apply (rule continuous-on-subset[rotated, where s=UNIV])
  subgoal by simp
  subgoal
    apply (subst continuous-map-iff-continuous2[symmetric])
    apply transfer
    by (rule continuous-map-scaleC-weak-star)
  done

lemma weak-star-closure-is-csubspace[simp]:
  fixes A::('a::hilbert-space, 'b::hilbert-space) cblinfun-weak-star set
  assumes ‹csubspace A›
  shows ‹csubspace (closure A)›
proof (rule complex-vector.subspaceI)
  include lattice-syntax
  show 0: ‹0 ∈ closure A›
  by (simp add: assms closure-def complex-vector.subspace-0)
  show ‹x + y ∈ closure A› if ‹x ∈ closure A› ‹y ∈ closure A› for x y
  proof -
    define FF where ‹FF = ((nhds x ⊓ principal A) ×F (nhds y ⊓ principal A))›
    have nt: ‹FF ≠ bot›
    by (simp add: prod-filter-eq-bot that(1) that(2) FF-def flip: closure-nhds-principal)
    have ‹∀F x in FF. fst x ∈ A›
    unfolding FF-def
    by (smt (verit, ccfv-SIG) eventually-prod-filter fst-conv inf-sup-ord(2) le-principal)
    moreover have ‹∀F x in FF. snd x ∈ A›
    unfolding FF-def
    by (smt (verit, ccfv-SIG) eventually-prod-filter snd-conv inf-sup-ord(2) le-principal)
    ultimately have FF-plus: ‹∀F x in FF. fst x + snd x ∈ A›
    by (smt (verit, best) assms complex-vector.subspace-add eventually-elim2)

    have ‹(fst ⟶ x) ((nhds x ⊓ principal A) ×F (nhds y ⊓ principal A))›
    apply (simp add: filterlim-def)
    using filtermap-fst-prod-filter
    using le-inf-iff by blast
    moreover have ‹(snd ⟶ y) ((nhds x ⊓ principal A) ×F (nhds y ⊓ principal A))›
    apply (simp add: filterlim-def)
    using filtermap-snd-prod-filter
    using le-inf-iff by blast
    ultimately have ‹(id ⟶ (x,y)) FF›
    by (simp add: filterlim-def nhds-prod prod-filter-mono FF-def)
  end
end

```

moreover note *tendsto-add-Pair*[of $x\ y$]
ultimately have $\langle ((\lambda x. \text{fst } x + \text{snd } x) \circ \text{id}) \longrightarrow (\lambda x. \text{fst } x + \text{snd } x) (x,y) \text{ FF} \rangle$
unfolding *filterlim-def nhds-prod*
by (*smt (verit, best) filterlim-compose filterlim-def filterlim-filtermap fst-conv snd-conv tendsto-compose-filtermap*)

then have $\langle ((\lambda x. \text{fst } x + \text{snd } x) \longrightarrow (x+y)) \text{ FF} \rangle$
by *simp*
then show $\langle x + y \in \text{closure } A \rangle$
using *nt FF-plus* **by** (*rule limit-in-closure*)

qed
show $\langle c *_C x \in \text{closure } A \rangle$ **if** $\langle x \in \text{closure } A \rangle$ **for** $x\ c$
proof (*cases c = 0*)
case *False*
have $(*_C) c \text{ 'closure } A \subseteq \text{closure } A$
using *csubspace-scaleC-invariant*[of $c\ A$] $\langle \text{csubspace } A \rangle$ *False closure-subset*[of A]
by (*intro image-closure-subset continuous-scaleC-weak-star closed-closure*) *auto*
thus *?thesis*
using *that* **by** *blast*
qed (*use 0 in auto*)
qed

lemma *transfer-csubspace-cblinfun-weak-star*[*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set } cr\text{-cblinfun-weak-star} ==> (=)) \text{ csubspace csubspace} \rangle$
unfolding *complex-vector.subspace-def*
by *transfer-prover*

lemma *transfer-closed-cblinfun-weak-star*[*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set } cr\text{-cblinfun-weak-star} ==> (=)) (\text{closedin weak-star-topology}) \text{ closed} \rangle$
proof –
have $(\text{rel-set } cr\text{-cblinfun-weak-star} ==> (=))$
 $(\lambda S. \text{openin weak-star-topology } (UNIV - S))$
 $(\lambda S. \text{open } (UNIV - S))$
by *transfer-prover*
thus *?thesis*
by (*simp add: closed-def[abs-def] closedin-def[abs-def] Compl-eq-Diff-UNIV*)
qed

lemma *transfer-closure-cblinfun-weak-star*[*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set } cr\text{-cblinfun-weak-star} ==> \text{rel-set } cr\text{-cblinfun-weak-star}) (\text{Abstract-Topology.closure-of weak-star-topology}) \text{ closure} \rangle$
apply (*subst closure-of-hull*[**where** $X = \text{weak-star-topology}$, *unfolded weak-star-topology-topospace, simplified, abs-def*])
apply (*subst closure-hull*[*abs-def*])
unfolding *hull-def*

by *transfer-prover*

lemma *weak-star-closure-is-csubspace* [*simp*]:
fixes $A::('a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space}) \text{ set}$
assumes $\langle \text{csubspace } A \rangle$
shows $\langle \text{csubspace } (\text{weak-star-topology closure-of } A) \rangle$
using *weak-star-closure-is-csubspace* [of $\langle \text{to-weak-star ' } A \rangle$] *assms*
apply (*transfer fixing: A*)
by *simp*

lemma *has-sum-closed-weak-star-topology*:
assumes $aA: \langle \bigwedge i. a \ i \in A \rangle$
assumes *closed*: $\langle \text{closedin weak-star-topology } A \rangle$
assumes *subspace*: $\langle \text{csubspace } A \rangle$
assumes *has-sum*: $\langle \bigwedge t. \text{trace-class } t \implies ((\lambda i. \text{trace } (t \ o_{CL} \ a \ i)) \text{ has-sum trace } (t \ o_{CL} \ b)) \ I \rangle$
shows $\langle b \in A \rangle$

proof –
have $1: \langle \text{range } (\text{sum } a) \subseteq A \rangle$
proof –
have $\langle \text{sum } a \ X \in A \rangle$ for X
 apply (*induction X rule:infinite-finite-induct*)
 by (*auto simp add: subspace complex-vector.subspace-0 aA complex-vector.subspace-add*)
 then show *?thesis*
 by *auto*
qed

from *has-sum*
have $\langle ((\lambda F. \sum i \in F. \text{trace } (t \ o_{CL} \ a \ i)) \longrightarrow \text{trace } (t \ o_{CL} \ b)) \ (\text{finite-subsets-at-top } I) \rangle$ **if**
 $\langle \text{trace-class } t \rangle$ **for** t
 by (*auto intro: that simp: has-sum-def*)
then have $\langle \text{limitin weak-star-topology } (\lambda F. \sum i \in F. a \ i) \ b \ (\text{finite-subsets-at-top } I) \rangle$
 by (*auto simp add: limitin-weak-star-topology cblinfun-compose-sum-right trace-sum trace-class-comp-left*)
then show $\langle b \in A \rangle$
 using 1 *closed* **apply** (*rule limitin-closedin*)
 by *simp*
qed

lemma *has-sum-in-weak-star*:
 $\langle \text{has-sum-in weak-star-topology } f \ A \ l \longleftrightarrow$
 $(\forall t. \text{trace-class } t \longrightarrow ((\lambda i. \text{trace } (t \ o_{CL} \ f \ i)) \text{ has-sum trace } (t \ o_{CL} \ l)) \ A) \rangle$
proof –
have $*$: $\langle \text{trace } (t \ o_{CL} \ \text{sum } f \ F) = \text{sum } (\lambda i. \text{trace } (t \ o_{CL} \ f \ i)) \ F \rangle$ **if** $\langle \text{trace-class } t \rangle$
 for $t \ F$
 by (*simp-all add: cblinfun-compose-sum-right that trace-class-comp-left trace-sum*)
show *?thesis*
 by (*simp add: * has-sum-def has-sum-in-def limitin-weak-star-topology*)
qed

lemma *has-sum-butterfly-ket*: $\langle \text{has-sum-in weak-star-topology } (\lambda i. \text{butterfly } (\text{ket } i) \ (\text{ket } i)) \ UNIV \rangle$

```

id-cblinfun›
proof (rule has-sum-in-weak-star[THEN iffD2, rule-format])
  fix t :: ⟨'a ell2 ⇒CL 'a ell2⟩
  assume [simp]: ⟨trace-class t⟩
  from trace-has-sum[OF is-onb-ket ⟨trace-class t⟩]
  have ⟨(λi. ket i •C (t *V ket i)) has-sum trace t⟩ UNIV›
    apply (subst (asm) has-sum-reindex)
    by (auto simp: o-def)
  then show ⟨(λi. trace (t oCL butterfly (ket i) (ket i))) has-sum trace (t oCL id-cblinfun)⟩
    UNIV›
    by (simp add: trace-butterfly-comp')
qed

lemma sandwich-weak-star-cont[simp]:
  ⟨continuous-map weak-star-topology weak-star-topology (sandwich A)⟩
  using continuous-map-compose[OF continuous-map-left-comp-weak-star continuous-map-right-comp-weak-star]
  by (auto simp: o-def sandwich-apply[abs-def])

lemma has-sum-butterfly-ket-a: ⟨has-sum-in weak-star-topology (λi. butterfly (a *V ket i) (ket
i)) UNIV a⟩
proof –
  have ⟨has-sum-in weak-star-topology ((λb. a oCL b) ∘ (λi. butterfly (ket i) (ket i))) UNIV (a
oCL id-cblinfun)⟩
    apply (rule has-sum-in-comm-additive)
  by (auto intro!: has-sum-butterfly-ket continuous-map-is-continuous-at-point limitin-continuous-map
continuous-map-left-comp-weak-star cblinfun-compose-add-right
simp: Modules.additive-def)
  then show ?thesis
    by (auto simp: o-def cblinfun-comp-butterfly)
qed

lemma finite-rank-weak-star-dense[simp]: ⟨weak-star-topology closure-of (Collect finite-rank) =
(UNIV :: ('a ell2 ⇒CL 'b::hilbert-space) set)⟩
proof –
  have ⟨x ∈ weak-star-topology closure-of (Collect finite-rank)⟩ for x :: ⟨'a ell2 ⇒CL 'b⟩
proof (rule limitin-closure-of)
  define f :: ⟨'a ⇒ 'a ell2 ⇒CL 'b⟩ where ⟨f = (λi. butterfly (x *V ket i) (ket i))⟩
  have ⟨has-sum-in weak-star-topology f UNIV x⟩
    using f-def has-sum-butterfly-ket-a by blast
  then show ⟨limitin weak-star-topology (sum f) x (finite-subsets-at-top UNIV)⟩
    using has-sum-in-def by blast
  show ⟨∀ F F in finite-subsets-at-top UNIV.
(∑ i∈F. butterfly (x *V ket i) (ket i)) ∈ Collect finite-rank⟩
    by (auto intro!: finite-rank-sum simp: f-def)
  show ⟨finite-subsets-at-top UNIV ≠ ⊥⟩
    by simp
qed
then show ?thesis

```

by auto
qed

lemma *butterkets-weak-star-dense[simp]*:

$\langle \text{weak-star-topology closure-of cspan } ((\lambda(\xi, \eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) \text{ 'UNIV}) = \text{UNIV} \rangle$

proof –

from *continuous-map-image-closure-subset[OF weak-star-topology-weaker-than-euclidean]*

have $\langle \text{weak-star-topology closure-of } (\text{cspan } ((\lambda(\xi, \eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) \text{ 'UNIV}))$
 $\supseteq \text{closure } (\text{cspan } ((\lambda(\xi, \eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) \text{ 'UNIV})) \rangle$ (**is** $\langle - \supseteq \dots \rangle$)

by auto

moreover

have $\langle \dots = \text{Collect compact-op} \rangle$

unfolding *finite-rank-dense-compact[OF is-onb-ket is-onb-ket, symmetric]*

by (*simp add: image-image case-prod-beta flip: map-prod-image*)

moreover have $\langle \dots \supseteq \text{Collect finite-rank} \rangle$

by (*metis closure-subset compact-op-finite-rank mem-Collect-eq subsetI subset-antisym*)

ultimately have *: $\langle \text{weak-star-topology closure-of } (\text{cspan } ((\lambda(\xi, \eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) \text{ 'UNIV})) \supseteq \text{Collect finite-rank} \rangle$

by *blast*

have $\langle \text{weak-star-topology closure-of cspan } ((\lambda(\xi, \eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) \text{ 'UNIV})$

$= \text{weak-star-topology closure-of } (\text{weak-star-topology closure-of cspan } ((\lambda(\xi, \eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) \text{ 'UNIV})) \rangle$

by *simp*

also have $\langle \dots \supseteq \text{weak-star-topology closure-of Collect finite-rank} \rangle$ (**is** $\langle - \supseteq \dots \rangle$)

using * *closure-of-mono* by *blast*

also have $\langle \dots = \text{UNIV} \rangle$

by *simp*

finally show *?thesis*

by auto

qed

lemma *weak-star-clinear-eq-butterfly-ketI*:

fixes $F G :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \Rightarrow 'c :: \text{complex-vector} \rangle$

assumes *clinear F and clinear G*

and $\langle \text{continuous-map weak-star-topology } T F \rangle$ **and** $\langle \text{continuous-map weak-star-topology } T G \rangle$

and $\langle \text{Hausdorff-space } T \rangle$

assumes $\bigwedge i j. F (\text{butterfly } (\text{ket } i) (\text{ket } j)) = G (\text{butterfly } (\text{ket } i) (\text{ket } j))$

shows $F = G$

proof –

have $FG: \langle F x = G x \rangle$ **if** $\langle x \in \text{cspan } ((\lambda(\xi, \eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) \text{ 'UNIV}) \rangle$ **for** x

by (*smt (verit) assms(1) assms(2) assms(6) complex-vector.linear-eq-on imageE split-def that*)

show *?thesis*

apply (*rule ext*)

using $\langle \text{Hausdorff-space } T \rangle FG$

apply (*rule closure-of-eqI*[**where** $f=F$ **and** $g=G$ **and** $S=\langle cspan ((\lambda(\xi,\eta). butterfly (ket \xi) (ket \eta)) \text{ 'UNIV})\rangle$])
using *assms butterkets-weak-star-dense* **by** *auto*
qed

lemma *continuous-map-scaleC-weak-star'*[*continuous-intros*]:
assumes $\langle continuous-map\ T\ weak-star-topology\ f \rangle$
shows $\langle continuous-map\ T\ weak-star-topology\ (\lambda x. scaleC\ c\ (f\ x)) \rangle$
using *continuous-map-compose*[*OF assms continuous-map-scaleC-weak-star*]
by (*simp add: o-def*)

lemma *continuous-map-uminus-weak-star*[*continuous-intros*]:
assumes $\langle continuous-map\ T\ weak-star-topology\ f \rangle$
shows $\langle continuous-map\ T\ weak-star-topology\ (\lambda x. -\ f\ x) \rangle$
apply (*subst scaleC-minus1-left*[*abs-def,symmetric*])
by (*intro continuous-map-scaleC-weak-star' assms*)

lemma *continuous-map-add-weak-star*[*continuous-intros*]:
assumes $\langle continuous-map\ T\ weak-star-topology\ f \rangle$
assumes $\langle continuous-map\ T\ weak-star-topology\ g \rangle$
shows $\langle continuous-map\ T\ weak-star-topology\ (\lambda x. f\ x + g\ x) \rangle$
proof –
have $\langle continuous-map\ T\ euclidean\ (\lambda x. trace\ (t\ o_{CL}\ f\ x)) \rangle$ **if** $\langle trace-class\ t \rangle$ **for** t
using *assms(1) continuous-on-weak-star-topo-iff-coordinatewise* **that** **by** *auto*
moreover **have** $\langle continuous-map\ T\ euclidean\ (\lambda x. trace\ (t\ o_{CL}\ g\ x)) \rangle$ **if** $\langle trace-class\ t \rangle$ **for** t
using *assms(2) continuous-on-weak-star-topo-iff-coordinatewise* **that** **by** *auto*
ultimately **show** *?thesis*
by (*auto intro! continuous-map-add simp add: continuous-on-weak-star-topo-iff-coordinatewise cblinfun-compose-add-right trace-class-comp-left trace-plus*)
qed

lemma *continuous-map-minus-weak-star*[*continuous-intros*]:
assumes $\langle continuous-map\ T\ weak-star-topology\ f \rangle$
assumes $\langle continuous-map\ T\ weak-star-topology\ g \rangle$
shows $\langle continuous-map\ T\ weak-star-topology\ (\lambda x. f\ x - g\ x) \rangle$
by (*subst diff-conv-add-uminus*) (*intro assms continuous-intros*)

lemma *weak-star-topology-is-norm-topology-fin-dim*[*simp*]:
 $\langle (weak-star-topology :: ('a::\{cfinite-dim,chilbert-space\} \Rightarrow_{CL} 'b::\{cfinite-dim,chilbert-space\}) topology) = euclidean \rangle$
proof –
have *1*: $\langle continuous-map\ euclidean\ weak-star-topology\ (id :: 'a \Rightarrow_{CL} 'b \Rightarrow -) \rangle$
by (*simp add: id-def weak-star-topology-weaker-than-euclidean*)
have $\langle continuous-map\ weak-star-topology\ cweak-operator-topology\ (id :: 'a \Rightarrow_{CL} 'b \Rightarrow -) \rangle$
by (*simp only: id-def wot-weaker-than-weak-star*)
then **have** *2*: $\langle continuous-map\ weak-star-topology\ euclidean\ (id :: 'a \Rightarrow_{CL} 'b \Rightarrow -) \rangle$
by (*simp only: wot-is-norm-topology-findim*)
from *1 2*
show *?thesis*

by (auto simp: topology-finer-continuous-id[symmetric] simp flip: openin-inject)
qed

lemma *infsum-mono-wot*:

fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$

assumes *summable-on-in cweak-operator-topology f A* and *summable-on-in cweak-operator-topology g A*

assumes $\langle \bigwedge x. x \in A \implies f x \leq g x \rangle$

shows *infsum-in cweak-operator-topology f A \leq infsum-in cweak-operator-topology g A*

by (meson *assms has-sum-in-infsum-in has-sum-mono-wot hausdorff-cweak-operator-topology*)

unbundle *no cblinfun-syntax*

end

14 Hilbert-Space-Tensor-Product – Tensor product of Hilbert Spaces

theory *Hilbert-Space-Tensor-Product*

imports *Complex-Bounded-Operators.Complex-L2 Misc-Tensor-Product*

Strong-Operator-Topology Polynomial-Interpolation.Ring-Hom

Positive-Operators Weak-Star-Topology Spectral-Theorem Trace-Class

begin

unbundle *cblinfun-syntax*

hide-const (open) *Determinants.trace*

hide-fact (open) *Determinants.trace-def*

14.1 Tensor product on - ell2

lift-definition *tensor-ell2* :: $\langle 'a \text{ ell2} \Rightarrow 'b \text{ ell2} \Rightarrow ('a \times 'b) \text{ ell2} \rangle$ (infixr \otimes_s 70) is
 $\langle \lambda \psi \varphi (i,j). \psi i * \varphi j \rangle$

proof –

fix $\psi :: \langle 'a \Rightarrow \text{complex} \rangle$ and $\varphi :: \langle 'b \Rightarrow \text{complex} \rangle$

assume $\langle \text{has-ell2-norm } \psi \rangle \langle \text{has-ell2-norm } \varphi \rangle$

from $\langle \text{has-ell2-norm } \varphi \rangle$ have $\varphi\text{-sum}: \langle \lambda j. (\psi i * \varphi j)^2 \rangle \text{abs-summable-on UNIV}$ for i

by (metis *ell2-norm-smult(1) has-ell2-norm-def*)

have *double-sum*: $\langle \lambda i. \sum_{\infty} j. \text{cmod } ((\psi i * \varphi j)^2) \rangle \text{abs-summable-on UNIV}$

unfolding *norm-mult power-mult-distrib infsum-cmult-right'*

by (rule *summable-on-cmult-left*) (use $\langle \text{has-ell2-norm } \psi \rangle$ in $\langle \text{auto simp: has-ell2-norm-def} \rangle$)

have $\langle \lambda (i,j). (\psi i * \varphi j)^2 \rangle \text{abs-summable-on UNIV} \times \text{UNIV}$

by (rule *abs-summable-on-Sigma-iff[THEN iffD2]*) (use $\varphi\text{-sum double-sum}$ in *auto*)

then show $\langle \text{has-ell2-norm } (\lambda (i, j). \psi i * \varphi j) \rangle$

by (auto simp *add: has-ell2-norm-def case-prod-beta*)

qed

lemma *tensor-ell2-add1*: $\langle \text{tensor-ell2 } (a + b) \ c = \text{tensor-ell2 } a \ c + \text{tensor-ell2 } b \ c \rangle$
by *transfer* (*auto simp: case-prod-beta vector-space-over-itself.scale-left-distrib*)

lemma *tensor-ell2-add2*: $\langle \text{tensor-ell2 } a \ (b + c) = \text{tensor-ell2 } a \ b + \text{tensor-ell2 } a \ c \rangle$
by *transfer* (*auto simp: case-prod-beta algebra-simps*)

lemma *tensor-ell2-scaleC1*: $\langle \text{tensor-ell2 } (c *_C a) \ b = c *_C \text{tensor-ell2 } a \ b \rangle$
by *transfer* (*auto simp: case-prod-beta*)

lemma *tensor-ell2-scaleC2*: $\langle \text{tensor-ell2 } a \ (c *_C b) = c *_C \text{tensor-ell2 } a \ b \rangle$
by *transfer* (*auto simp: case-prod-beta*)

lemma *tensor-ell2-diff1*: $\langle \text{tensor-ell2 } (a - b) \ c = \text{tensor-ell2 } a \ c - \text{tensor-ell2 } b \ c \rangle$
by *transfer* (*auto simp: case-prod-beta ordered-field-class.sign-simps*)

lemma *tensor-ell2-diff2*: $\langle \text{tensor-ell2 } a \ (b - c) = \text{tensor-ell2 } a \ b - \text{tensor-ell2 } a \ c \rangle$
by *transfer* (*auto simp: case-prod-beta ordered-field-class.sign-simps*)

lemma *tensor-ell2-inner-prod*[*simp*]: $\langle \text{tensor-ell2 } a \ b \cdot_C \text{tensor-ell2 } c \ d = (a \cdot_C c) * (b \cdot_C d) \rangle$
apply (*rule local-defE*[**where** $y = \langle \text{tensor-ell2 } a \ b \rangle$], *rename-tac ab*)
apply (*rule local-defE*[**where** $y = \langle \text{tensor-ell2 } c \ d \rangle$], *rename-tac cd*)
proof (*transfer, hypsubst-thin*)
fix $a \ c :: \langle 'a \Rightarrow \text{complex} \rangle$ **and** $b \ d :: \langle 'b \Rightarrow \text{complex} \rangle$

assume *assms*: $\langle \text{has-ell2-norm } (\lambda(i, j). a \ i * b \ j) \rangle \langle \text{has-ell2-norm } (\lambda(i, j). c \ i * d \ j) \rangle$

have $*$: $\langle (\lambda xy. \text{cnj } (a \ (\text{fst } xy) * b \ (\text{snd } xy)) * (c \ (\text{fst } xy) * d \ (\text{snd } xy))) \text{abs-summable-on } UNIV \rangle$
apply (*rule abs-summable-product*)
subgoal
by (*metis* (*mono-tags, lifting*) *assms(1) complex-mod-cnj has-ell2-norm-def norm-power split-def summable-on-cong*)
subgoal
by (*metis* (*mono-tags, lifting*) *assms(2) case-prod-unfold has-ell2-norm-def summable-on-cong*)
done

then have $*$: $\langle (\lambda(x, y). \text{cnj } (a \ x * b \ y) * (c \ x * d \ y)) \text{summable-on } UNIV \times UNIV \rangle$
using *abs-summable-summable* **by** (*auto simp: case-prod-unfold*)

have $\langle (\sum_{\infty i. \text{cnj } (\text{case } i \ \text{of } (i, j) \Rightarrow a \ i * b \ j) * (\text{case } i \ \text{of } (i, j) \Rightarrow c \ i * d \ j)) = (\sum_{\infty (i, j) \in UNIV \times UNIV. \text{cnj } (a \ i * b \ j) * (c \ i * d \ j)}) \rangle$ (**is** $\langle ?lhs = - \rangle$)
by (*simp add: case-prod-unfold*)

also have $\langle \dots = (\sum_{\infty i. \sum_{\infty j. \text{cnj } (a \ i * b \ j) * (c \ i * d \ j)}) \rangle$
by (*subst infsum-Sigma'-banach[symmetric]*) (*use * in auto*)

also have $\langle \dots = (\sum_{\infty i. \text{cnj } (a \ i) * c \ i) * (\sum_{\infty j. \text{cnj } (b \ j) * (d \ j)}) \rangle$ (**is** $\langle - = ?rhs \rangle$)
by (*subst infsum-cmult-left'[symmetric]*)
(auto intro!: infsum-cong simp flip: infsum-cmult-right')

finally show $\langle ?lhs = ?rhs \rangle$.

qed

lemma *norm-tensor-ell2*: $\langle \text{norm } (a \otimes_s b) = \text{norm } a * \text{norm } b \rangle$
by (*simp add: norm-eq-sqrt-cinner*[**where** 'a= $\langle \text{::type } \text{ell2} \rangle$ '] *norm-mult real-sqrt-mult*)

lemma *cllinear-tensor-ell21*: *cllinear* $(\lambda b. a \otimes_s b)$
by (*rule cllinearI; transfer*)
(*auto simp add: case-prod-beta cond-case-prod-eta algebra-simps fun-eq-iff*)

lemma *bounded-cllinear-tensor-ell21*: *bounded-cllinear* $(\lambda b. a \otimes_s b)$
by (*auto intro!: bounded-cllinear.intro cllinear-tensor-ell21*
simp: bounded-cllinear-axioms-def norm-tensor-ell2 mult.commute[*of norm a*])

lemma *cllinear-tensor-ell22*: *cllinear* $(\lambda a. a \otimes_s b)$
by (*rule cllinearI; transfer*) (*auto simp: case-prod-beta algebra-simps*)

lemma *bounded-cllinear-tensor-ell22*: *bounded-cllinear* $(\lambda a. \text{tensor-ell2 } a \ b)$
by (*auto intro!: bounded-cllinear.intro cllinear-tensor-ell22*
simp: bounded-cllinear-axioms-def norm-tensor-ell2)

lemma *tensor-ell2-ket*: *tensor-ell2* $(\text{ket } i) (\text{ket } j) = \text{ket } (i, j)$
by *transfer auto*

lemma *tensor-ell2-0-left*[*simp*]: $\langle 0 \otimes_s x = 0 \rangle$
by *transfer auto*

lemma *tensor-ell2-0-right*[*simp*]: $\langle x \otimes_s 0 = 0 \rangle$
by *transfer auto*

lemma *tensor-ell2-sum-left*: $\langle (\sum x \in X. a \ x) \otimes_s b = (\sum x \in X. a \ x \otimes_s b) \rangle$
by (*induction X rule:infinite-finite-induct*) (*auto simp: tensor-ell2-add1*)

lemma *tensor-ell2-sum-right*: $\langle a \otimes_s (\sum x \in X. b \ x) = (\sum x \in X. a \otimes_s b \ x) \rangle$
by (*induction X rule:infinite-finite-induct*) (*auto simp: tensor-ell2-add2*)

lemma *tensor-ell2-dense*:

fixes $S :: \langle 'a \ \text{ell2 set} \rangle$ **and** $T :: \langle 'b \ \text{ell2 set} \rangle$

assumes $\langle \text{closure } (\text{cspan } S) = \text{UNIV} \rangle$ **and** $\langle \text{closure } (\text{cspan } T) = \text{UNIV} \rangle$

shows $\langle \text{closure } (\text{cspan } \{a \otimes_s b \mid a \ b. a \in S \wedge b \in T\}) = \text{UNIV} \rangle$

proof –

define ST **where** $\langle ST = \{a \otimes_s b \mid a \ b. a \in S \wedge b \in T\} \rangle$

from *assms* **have** 1: $\langle \text{bounded-cllinear } F \implies \text{bounded-cllinear } G \implies (\forall x \in S. F \ x = G \ x) \implies F = G \rangle$ **for** $F \ G :: \langle 'a \ \text{ell2} \implies \text{complex} \rangle$

using *bounded-cllinear-eq-on-closure*[*of F G S*] **by** *auto*

from *assms* **have** 2: $\langle \text{bounded-cllinear } F \implies \text{bounded-cllinear } G \implies (\forall x \in T. F \ x = G \ x) \implies F = G \rangle$ **for** $F \ G :: \langle 'b \ \text{ell2} \implies \text{complex} \rangle$

using *bounded-cllinear-eq-on-closure*[*of F G T*] **by** *auto*

have $\langle F = G \rangle$

if [*simp*]: $\langle \text{bounded-cllinear } F \rangle \langle \text{bounded-cllinear } G \rangle$ **and** *eq*: $\langle \forall x \in ST. F \ x = G \ x \rangle$

```

for F G :: ⟨('a×'b) ell2 ⇒ complex⟩
proof -
from eq have eq': ⟨F (s ⊗s t) = G (s ⊗s t)⟩ if ⟨s ∈ S⟩ and ⟨t ∈ T⟩ for s t
  using ST-def that by blast
have eq'': ⟨F (s ⊗s ket t) = G (s ⊗s ket t)⟩ if ⟨s ∈ S⟩ for s t
  by (rule fun-cong[where x=⟨ket t⟩], rule 2)
  (use eq' that in ⟨auto simp: bounded-clinear-compose bounded-clinear-tensor-ell21⟩)
have eq''': ⟨F (ket s ⊗s ket t) = G (ket s ⊗s ket t)⟩ for s t
  by (rule fun-cong[where x=⟨ket s⟩], rule 1)
  (use eq'' in ⟨auto simp: bounded-clinear-compose bounded-clinear-tensor-ell21
    intro: bounded-clinear-compose[OF - bounded-clinear-tensor-ell22]⟩)
show F = G
  by (rule bounded-clinear-equal-ket) (use eq''' in ⟨auto simp: tensor-ell2-ket⟩)
qed
then show ⟨closure (cspan ST) = UNIV⟩
  using separating-dense-span by blast
qed

definition assoc-ell2 :: ⟨(('a×'b)×'c) ell2 ⇒CL ('a×('b×'c)) ell2⟩ where
  ⟨assoc-ell2 = classical-operator (Some o (λ((a,b),c). (a,(b,c))))⟩

lemma unitary-assoc-ell2[simp]: ⟨unitary assoc-ell2⟩
  unfolding assoc-ell2-def
  by (rule unitary-classical-operator, rule o-bij[of ⟨(λ(a,(b,c)). ((a,b),c))⟩] auto)

lemma assoc-ell2-tensor: ⟨assoc-ell2 *V ((a ⊗s b) ⊗s c) = (a ⊗s (b ⊗s c))⟩
proof -
  note [simp] = bounded-clinear-compose[OF bounded-clinear-tensor-ell21]
    bounded-clinear-compose[OF bounded-clinear-tensor-ell22]
    bounded-clinear-cblinfun-apply
  have ⟨assoc-ell2 *V ((ket a ⊗s ket b) ⊗s ket c) = (ket a ⊗s (ket b ⊗s ket c))⟩ for a :: 'a and
    b :: 'b and c :: 'c
    by (simp add: inj-def assoc-ell2-def classical-operator-ket classical-operator-exists-inj tensor-ell2-ket)
  then have ⟨assoc-ell2 *V ((ket a ⊗s ket b) ⊗s c) = (ket a ⊗s (ket b ⊗s c))⟩ for a :: 'a and
    b :: 'b
    apply -
    apply (rule fun-cong[where x=c])
    apply (rule bounded-clinear-equal-ket)
    by auto
  then have ⟨assoc-ell2 *V ((ket a ⊗s b) ⊗s c) = (ket a ⊗s (b ⊗s c))⟩ for a :: 'a
    apply -
    apply (rule fun-cong[where x=b])
    apply (rule bounded-clinear-equal-ket)
    by auto
  then show ⟨assoc-ell2 *V ((a ⊗s b) ⊗s c) = (a ⊗s (b ⊗s c))⟩
    apply -
    apply (rule fun-cong[where x=a])
    apply (rule bounded-clinear-equal-ket)

```

by auto
qed

lemma *assoc-ell2'-tensor*: $\langle \text{assoc-ell2} *_{\mathcal{V}} \text{tensor-ell2 } a (\text{tensor-ell2 } b \ c) = \text{tensor-ell2 } (\text{tensor-ell2 } a \ b) \ c \rangle$

by (metis (no-types, opaque-lifting) *assoc-ell2-tensor cblinfun-apply-cblinfun-compose id-cblinfun.rep-eq unitaryD1 unitary-assoc-ell2*)

lemma *assoc-ell2'-inv*: $\text{assoc-ell2 } o_{CL} \ \text{assoc-ell2} * = \text{id-cblinfun}$
by (auto intro: *equal-ket*)

lemma *assoc-ell2-inv*: $\text{assoc-ell2} * \ o_{CL} \ \text{assoc-ell2} = \text{id-cblinfun}$
by (auto intro: *equal-ket*)

definition *swap-ell2* :: $\langle ('a \times 'b) \ \text{ell2} \Rightarrow_{CL} ('b \times 'a) \ \text{ell2} \rangle$ **where**
 $\langle \text{swap-ell2} = \text{classical-operator } (\text{Some } o \ \text{prod.swap}) \rangle$

lemma *unitary-swap-ell2[simp]*: $\langle \text{unitary } \text{swap-ell2} \rangle$
unfolding *swap-ell2-def* **by** (rule *unitary-classical-operator*) auto

lemma *swap-ell2-tensor[simp]*: $\langle \text{swap-ell2} *_{\mathcal{V}} (a \otimes_s b) = b \otimes_s a \rangle$ **for** $a :: \langle 'a \ \text{ell2} \rangle$ **and** $b :: \langle 'b \ \text{ell2} \rangle$

proof –

note [simp] = *bounded-clinear-compose[OF bounded-clinear-tensor-ell21]*

bounded-clinear-compose[OF bounded-clinear-tensor-ell22]

bounded-clinear-cblinfun-apply

have $\langle \text{swap-ell2} *_{\mathcal{V}} (\text{ket } a \otimes_s \text{ket } b) = (\text{ket } b \otimes_s \text{ket } a) \rangle$ **for** $a :: 'a$ **and** $b :: 'b$

by (simp add: *inj-def swap-ell2-def classical-operator-ket classical-operator-exists-inj tensor-ell2-ket*)

then have $\langle \text{swap-ell2} *_{\mathcal{V}} (\text{ket } a \otimes_s b) = (b \otimes_s \text{ket } a) \rangle$ **for** $a :: 'a$

apply –

apply (rule *fun-cong[where x=b]*)

apply (rule *bounded-clinear-equal-ket*)

by auto

then show $\langle \text{swap-ell2} *_{\mathcal{V}} (a \otimes_s b) = (b \otimes_s a) \rangle$

apply –

apply (rule *fun-cong[where x=a]*)

apply (rule *bounded-clinear-equal-ket*)

by auto

qed

lemma *swap-ell2-ket[simp]*: $\langle (\text{swap-ell2} :: ('a \times 'b) \ \text{ell2} \Rightarrow_{CL} \ -) *_{\mathcal{V}} \text{ket } (x,y) = \text{ket } (y,x) \rangle$
by (metis *swap-ell2-tensor tensor-ell2-ket*)

lemma *adjoint-swap-ell2[simp]*: $\langle \text{swap-ell2} * = \text{swap-ell2} \rangle$
by (simp add: *swap-ell2-def inv-map-total*)

lemma *tensor-ell2-extensionality*:

assumes $\langle \bigwedge s t. a *_V (s \otimes_s t) = b *_V (s \otimes_s t) \rangle$
shows $a = b$
using *assms* **by** (*auto intro: equal-ket simp flip: tensor-ell2-ket*)

lemma *tensor-ell2-nonzero*: $\langle a \otimes_s b \neq 0 \rangle$ **if** $\langle a \neq 0 \rangle$ **and** $\langle b \neq 0 \rangle$
by (*use that in transfer*) (*auto simp: fun-eq-iff*)

lemma *swap-ell2-selfinv[simp]*: $\langle \text{swap-ell2 } o_{CL} \text{ swap-ell2} = \text{id-cblinfun} \rangle$
by (*metis adjoint-swap-ell2 unitary-def unitary-swap-ell2*)

lemma *bounded-cbilinear-tensor-ell2[bounded-cbilinear]*: $\langle \text{bounded-cbilinear } (\otimes_s) \rangle$
proof *standard*
fix $a a' :: 'a \text{ ell2}$ **and** $b b' :: 'b \text{ ell2}$ **and** $r :: \text{complex}$
show $\langle \text{tensor-ell2 } (a + a') b = \text{tensor-ell2 } a b + \text{tensor-ell2 } a' b \rangle$
by (*meson tensor-ell2-add1*)
show $\langle \text{tensor-ell2 } a (b + b') = \text{tensor-ell2 } a b + \text{tensor-ell2 } a b' \rangle$
by (*simp add: tensor-ell2-add2*)
show $\langle \text{tensor-ell2 } (r *_C a) b = r *_C \text{tensor-ell2 } a b \rangle$
by (*simp add: tensor-ell2-scaleC1*)
show $\langle \text{tensor-ell2 } a (r *_C b) = r *_C \text{tensor-ell2 } a b \rangle$
by (*simp add: tensor-ell2-scaleC2*)
show $\langle \exists K. \forall a b. \text{norm } (\text{tensor-ell2 } a b) \leq \text{norm } a * \text{norm } b * K \rangle$
by (*rule exI[of - 1]*) (*simp add: norm-tensor-ell2*)

qed

lemma *ket-pair-split*: $\langle \text{ket } x = \text{tensor-ell2 } (\text{ket } (\text{fst } x)) (\text{ket } (\text{snd } x)) \rangle$
by (*simp add: tensor-ell2-ket*)

lemma *tensor-ell2-is-ortho-set*:
assumes $\langle \text{is-ortho-set } A \rangle$ $\langle \text{is-ortho-set } B \rangle$
shows $\langle \text{is-ortho-set } \{a \otimes_s b \mid a \in A \wedge b \in B\} \rangle$
unfolding *is-ortho-set-def*
proof *safe*
fix $a a' b b'$
assume $ab: a \in A a' \in A b \in B b' \in B a \otimes_s b \neq a' \otimes_s b'$
hence $a \neq a' \vee b \neq b'$
by *auto*
hence *is-orthogonal* $a a' \vee$ *is-orthogonal* $b b'$
using *assms is-ortho-setD ab* **by** *metis*
thus *is-orthogonal* $(a \otimes_s b) (a' \otimes_s b')$
by *auto*

next
fix $a b$
assume $ab: a \in A b \in B 0 = a \otimes_s b$
hence $a \neq 0 b \neq 0$
using *assms unfolding is-ortho-set-def* **by** *blast+*
thus *False* **using** ab

using *tensor-ell2-nonzero*[of *a b*] **by** *simp*
qed

lemma *tensor-ell2-dense'*: $\langle \text{ccspan } \{a \otimes_s b \mid a \in A \wedge b \in B\} = \top \rangle$ **if** $\langle \text{ccspan } A = \top \rangle$ **and**
 $\langle \text{ccspan } B = \top \rangle$
proof –
from *that have* *Adense*: $\langle \text{closure } (\text{cspan } A) = \text{UNIV} \rangle$
by (*transfer' fixing: A*) *simp*
from *that have* *Bdense*: $\langle \text{closure } (\text{cspan } B) = \text{UNIV} \rangle$
by (*transfer' fixing: B*) *simp*
show $\langle \text{ccspan } \{a \otimes_s b \mid a \in A \wedge b \in B\} = \top \rangle$
by (*transfer fixing: A B*) (*use Adense Bdense in <rule tensor-ell2-dense>*)
qed

lemma *tensor-ell2-is-onb*:
assumes $\langle \text{is-onb } A \rangle \langle \text{is-onb } B \rangle$
shows $\langle \text{is-onb } \{a \otimes_s b \mid a \in A \wedge b \in B\} \rangle$
proof (*subst is-onb-def, intro conjI ballI*)
show $\langle \text{is-ortho-set } \{a \otimes_s b \mid a \in A \wedge b \in B\} \rangle$
by (*rule tensor-ell2-is-ortho-set*) (*use assms in <auto simp: is-onb-def>*)
show $\langle \text{ccspan } \{a \otimes_s b \mid a \in A \wedge b \in B\} = \top \rangle$
by (*rule tensor-ell2-dense'*) (*use <is-onb A> <is-onb B> in <simp-all add: is-onb-def>*)
show $\langle ab \in \{a \otimes_s b \mid a \in A \wedge b \in B\} \implies \text{norm } ab = 1 \rangle$ **for** *ab*
using $\langle \text{is-onb } A \rangle \langle \text{is-onb } B \rangle$ **by** (*auto simp: is-onb-def norm-tensor-ell2*)
qed

lemma *continuous-tensor-ell2*: $\langle \text{continuous-on UNIV } (\lambda(x::'a \text{ ell2}, y::'b \text{ ell2}). x \otimes_s y) \rangle$
proof –
have *cont*: $\langle \text{continuous-on UNIV } (\lambda t. t \otimes_s x) \rangle$ **for** $x :: \langle 'b \text{ ell2} \rangle$
by (*intro linear-continuous-on bounded-clinear.bounded-linear bounded-clinear-tensor-ell22*)
have *lip*: $\langle \text{local-lipschitz } (\text{UNIV} :: 'a \text{ ell2 set}) (\text{UNIV} :: 'b \text{ ell2 set}) (\otimes_s) \rangle$
proof (*rule local-lipschitzI*)
fix $t :: \langle 'a \text{ ell2} \rangle$ **and** $x :: \langle 'b \text{ ell2} \rangle$
define $u \ L :: \text{real where } \langle u = 1 \rangle$ **and** $\langle L = \text{norm } t + u \rangle$
have $\langle u > 0 \rangle$
by (*simp add: u-def*)
have [*simp*]: $\langle L \geq 0 \rangle$
by (*simp add: L-def u-def*)
have $\langle \text{norm } s \leq L \rangle$ **if** $\langle s \in \text{cball } t \ u \rangle$ **for** $s :: \langle 'a \text{ ell2} \rangle$
using *that unfolding L-def mem-cball by norm*
have $\langle L\text{-lipschitz-on } (\text{cball } x \ u) ((\otimes_s) s) \rangle$ **if** $\langle s \in \text{cball } t \ u \rangle$ **for** $s :: \langle 'a \text{ ell2} \rangle$
by (*rule lipschitz-onI*)
*(auto intro!: mult-right-mono *[OF that]*
simp add: dist-norm norm-tensor-ell2 simp flip: tensor-ell2-diff2)
with $\langle u > 0 \rangle$ **show** $\langle \exists u > 0. \exists L. \forall s \in \text{cball } t \ u \cap \text{UNIV}. L\text{-lipschitz-on } (\text{cball } x \ u \cap \text{UNIV})$
 $((\otimes_s) s) \rangle$
by *force*
qed
show *?thesis*

by (subst UNIV-Times-UNIV[symmetric]) (use lip cont in ⟨rule Lipschitz.continuous-on-TimesI⟩)
qed

lemma summable-on-tensor-ell2-right: ⟨ φ summable-on $A \implies (\lambda x. \psi \otimes_s \varphi x)$ summable-on A ⟩
by (rule summable-on-bounded-linear[where h=⟨ $\lambda x. \psi \otimes_s x$ ⟩]) (intro bounded-linear-intros)

lemma summable-on-tensor-ell2-left: ⟨ φ summable-on $A \implies (\lambda x. \varphi x \otimes_s \psi)$ summable-on A ⟩
by (rule summable-on-bounded-linear[where h=⟨ $\lambda x. x \otimes_s \psi$ ⟩]) (intro bounded-linear-intros)

lift-definition tensor-ell2-left :: ⟨ $'a$ ell2 $\implies ('b$ ell2 $\implies_{CL} ('a \times 'b)$ ell2)⟩ is
⟨ $\lambda \psi \varphi. \psi \otimes_s \varphi$ ⟩
by (simp add: bounded-cbilinear.bounded-clinear-right bounded-cbilinear-tensor-ell2)

lemma tensor-ell2-left-apply[simp]: ⟨tensor-ell2-left $\psi *_{\mathcal{V}} \varphi = \psi \otimes_s \varphi$ ⟩
by (transfer fixing: $\psi \varphi$) simp

lift-definition tensor-ell2-right :: ⟨ $'a$ ell2 $\implies ('b$ ell2 $\implies_{CL} ('b \times 'a)$ ell2)⟩ is
⟨ $\lambda \psi \varphi. \varphi \otimes_s \psi$ ⟩
by (simp add: bounded-clinear-tensor-ell2)

lemma tensor-ell2-right-apply[simp]: ⟨tensor-ell2-right $\psi *_{\mathcal{V}} \varphi = \varphi \otimes_s \psi$ ⟩
by (transfer fixing: $\psi \varphi$) simp

lemma isometry-tensor-ell2-right: ⟨isometry (tensor-ell2-right ψ)⟩ if ⟨norm $\psi = 1$ ⟩
by (rule norm-preserving-isometry) (simp add: norm-tensor-ell2 that)

lemma isometry-tensor-ell2-left: ⟨isometry (tensor-ell2-left ψ)⟩ if ⟨norm $\psi = 1$ ⟩
by (rule norm-preserving-isometry) (simp add: norm-tensor-ell2 that)

lemma tensor-ell2-right-scale: ⟨tensor-ell2-right ($a *_{\mathcal{C}} \psi$) = $a *_{\mathcal{C}}$ tensor-ell2-right ψ ⟩
by transfer (auto simp: tensor-ell2-scaleC2)

lemma tensor-ell2-left-scale: ⟨tensor-ell2-left ($a *_{\mathcal{C}} \psi$) = $a *_{\mathcal{C}}$ tensor-ell2-left ψ ⟩
by transfer (auto simp: tensor-ell2-scaleC1)

lemma tensor-ell2-right-0[simp]: ⟨tensor-ell2-right $0 = 0$ ⟩
by (auto intro!: cblinfun-eqI)

lemma tensor-ell2-left-0[simp]: ⟨tensor-ell2-left $0 = 0$ ⟩
by (auto intro!: cblinfun-eqI)

lemma tensor-ell2-right-adj-apply[simp]: ⟨(tensor-ell2-right ψ^*) $*_{\mathcal{V}} (\alpha \otimes_s \beta) = (\psi \cdot_{\mathcal{C}} \beta) *_{\mathcal{C}} \alpha$ ⟩
by (rule cinner-extensionality) (simp add: cinner-adj-right)

lemma tensor-ell2-left-adj-apply[simp]: ⟨(tensor-ell2-left ψ^*) $*_{\mathcal{V}} (\alpha \otimes_s \beta) = (\psi \cdot_{\mathcal{C}} \alpha) *_{\mathcal{C}} \beta$ ⟩
by (rule cinner-extensionality) (simp add: cinner-adj-right)

lemma infsum-tensor-ell2-right: ⟨ $\psi \otimes_s (\sum_{\infty x \in A. \varphi x}) = (\sum_{\infty x \in A. \psi \otimes_s \varphi x)$ ⟩

proof –

consider (summable) ⟨ φ summable-on A ⟩ | (summable) ⟨ $\psi \neq 0$ ⟩ ⟨ $(\lambda x. \psi \otimes_s \varphi x)$ summable-on A ⟩

```

| (ψ 0) ⟨ψ = 0⟩
| (not-summable) ⟨¬ φ summable-on A⟩ ⟨¬ (λx. ψ ⊗s φ x) summable-on A⟩
by auto
then show ?thesis
proof cases
case summable
then show ?thesis
by (rule infsum-bounded-linear[symmetric, unfolded o-def, rotated])
(intro bounded-linear-intros)
next
case summable'
then have *: ⟨(ψ /R (norm ψ)2) ·C ψ = 1⟩
by (simp add: scaleR-scaleC cdot-square-norm)
from summable'(2) have ⟨(λx. (tensor-ell2-left (ψ /R (norm ψ)2))* *V (ψ ⊗s φ x))
summable-on A⟩
by (rule summable-on-bounded-linear[unfolded o-def, rotated])
(intro bounded-linear-intros)
with * have ⟨φ summable-on A⟩
by simp
then show ?thesis
by (rule infsum-bounded-linear[symmetric, unfolded o-def, rotated])
(intro bounded-linear-intros)
next
case ψ 0
then show ?thesis
by simp
next
case not-summable
then show ?thesis
by (simp add: infsum-not-exists)
qed
qed

```

lemma *infsum-tensor-ell2-left*: $\langle (\sum_{\infty x \in A}. \varphi x) \otimes_s \psi = (\sum_{\infty x \in A}. \varphi x \otimes_s \psi) \rangle$

proof –

from *infsum-tensor-ell2-right*

have $\langle \text{swap-ell2} *_{V} (\psi \otimes_s (\sum_{\infty x \in A}. \varphi x)) = \text{swap-ell2} *_{V} (\sum_{\infty x \in A}. \psi \otimes_s \varphi x) \rangle$

by *metis*

then show *?thesis*

by (*simp add: invertible-cblinfun-isometry flip: infsum-cblinfun-apply-invertible*)

qed

lemma *tensor-ell2-extensionality3*:

assumes $\langle \bigwedge s t u. a *_{V} (s \otimes_s t \otimes_s u) = b *_{V} (s \otimes_s t \otimes_s u) \rangle$

shows $a = b$

by (*rule equal-ket*) (*use assms in* $\langle \text{auto simp flip: tensor-ell2-ket} \rangle$)

lemma *cblinfun-cinner-tensor-eqI*:

assumes $\langle \bigwedge \psi \varphi. (\psi \otimes_s \varphi) \cdot_C (A *_{V} (\psi \otimes_s \varphi)) = (\psi \otimes_s \varphi) \cdot_C (B *_{V} (\psi \otimes_s \varphi)) \rangle$


```

shows ⟨A = B⟩
proof -
  define C where ⟨C = A - B⟩
  from assms have assmC: ⟨(ψ ⊗s φ) •C (C *V (ψ ⊗s φ)) = 0⟩ for ψ φ
    by (simp add: C-def cblinfun.diff-left cinner-simps(3))

  have ⟨(x ⊗s y) •C (C *V (z ⊗s w)) = 0⟩ for x y z w

  proof -
    define d e f g h j k l m n p q
      where defs: ⟨d = (x ⊗s y) •C (C *V z ⊗s w)⟩
        ⟨e = (z ⊗s y) •C (C *V x ⊗s y)⟩
        ⟨f = (x ⊗s w) •C (C *V x ⊗s y)⟩
        ⟨g = (z ⊗s w) •C (C *V x ⊗s y)⟩
        ⟨h = (x ⊗s y) •C (C *V z ⊗s y)⟩
        ⟨j = (x ⊗s w) •C (C *V z ⊗s y)⟩
        ⟨k = (z ⊗s w) •C (C *V z ⊗s y)⟩
        ⟨l = (z ⊗s w) •C (C *V x ⊗s w)⟩
        ⟨m = (x ⊗s y) •C (C *V x ⊗s w)⟩
        ⟨n = (z ⊗s y) •C (C *V x ⊗s w)⟩
        ⟨p = (z ⊗s y) •C (C *V z ⊗s w)⟩
        ⟨q = (x ⊗s w) •C (C *V z ⊗s w)⟩

    have constraint: ⟨cnj α * e + cnj β * f + cnj β * cnj α * g + α * h + α * cnj β * j +
      α * cnj β * cnj α * k + β * m + β * cnj α * n + β * cnj β * cnj α * l +
      β * α * d + β * α * cnj α * p + β * α * cnj β * q = 0⟩
      (is ⟨?lhs = -⟩) for α β
    proof -
      from assms
      have ⟨0 = ((x + α *C z) ⊗s (y + β *C w)) •C (C *V ((x + α *C z) ⊗s (y + β *C w)))⟩
        by (simp add: assmC)
      also have ⟨... = ?lhs⟩
        by (simp add: tensor-ell2-add1 tensor-ell2-add2 cinner-add-right cinner-add-left
          cblinfun.add-right tensor-ell2-scaleC1 tensor-ell2-scaleC2 semiring-class.distrib-left
          cblinfun.scaleC-right assmC defs flip: add.assoc mult.assoc)
      finally show ?thesis
        by simp
    qed

    have aux1: ⟨a = 0 ⟹ b = 0 ⟹ a + b = 0⟩ for a b :: complex
      by auto
    have aux2: ⟨a = 0 ⟹ b = 0 ⟹ a - b = 0⟩ for a b :: complex
      by auto
    have aux4: ⟨2 * a = 0 ⟷ a = 0⟩ for a :: complex
      by auto
    have aux5: ⟨8 = 2 * 2 * (2::complex)⟩
      by simp

    from constraint[of 1 0]

```

```

have 1: ⟨e + h = 0⟩
  by simp
from constraint[of i 0]
have 2: ⟨h = e⟩
  by simp
from 1 2
have [simp]: ⟨e = 0⟩ ⟨h = 0⟩
  by auto
from constraint[of 0 1]
have 3: ⟨f + m = 0⟩
  by simp
from constraint[of 0 i]
have 4: ⟨m = f⟩
  by simp
from 3 4
have [simp]: ⟨m = 0⟩ ⟨f = 0⟩
  by auto
from constraint[of 1 1]
have 5: ⟨g + j + k + n + l + d + p + q = 0⟩
  by simp
from constraint[of 1 ⟨-1⟩]
have 6: ⟨-g - j - k - n + l - d - p + q = 0⟩
  by simp
from aux1[OF 5 6]
have 7: ⟨l + q = 0⟩
  by algebra
from aux2[OF 5 7]
have 8: ⟨g + j + k + n + d + p = 0⟩
  by (simp add: algebra-simps)
from constraint[of 1 i]
have 9: ⟨-(i * g) - i * j - i * k + i * n + l + i * d + i * p + q = 0⟩
  by simp
from constraint[of 1 ⟨-i⟩]
have 10: ⟨i * g + i * j + i * k - i * n + l - i * d - i * p + q = 0⟩
  by simp
from aux2[OF 9 10]
have 11: ⟨n + d + p - k - j - g = 0⟩
  using i-squared by algebra
from aux2[OF 8 11]
have 12: ⟨g + j + k = 0⟩
  by algebra
from aux1[OF 8 11]
have 13: ⟨n + d + p = 0⟩
  by algebra
from constraint[of i 1]
have 14: ⟨i * j - i * g + k - i * n - i * l + i * d + p + i * q = 0⟩
  by simp
from constraint[of i ⟨-1⟩]
have 15: ⟨i * g - i * j - k + i * n - i * l - i * d - p + i * q = 0⟩

```

```

    by simp
  from aux1[OF 14 15]
  have [simp]:  $\langle q = l \rangle$ 
    by simp
  from 7
  have [simp]:  $\langle q = 0 \rangle \langle l = 0 \rangle$ 
    by auto
  from 14
  have 16:  $\langle i * j - i * g + k - i * n + i * d + p = 0 \rangle$ 
    by simp
  from constraint[of  $\langle -i \rangle$  1]
  have 17:  $\langle i * g - i * j + k + i * n - i * d + p = 0 \rangle$ 
    by simp
  from aux1[OF 16 17]
  have [simp]:  $\langle k = -p \rangle$ 
    by algebra
  from aux2[OF 16 17]
  have 18:  $\langle j + d - n - g = 0 \rangle$ 
    using i-squared by algebra
  from constraint[of  $\langle -i \rangle$  1]
  have 19:  $\langle i * g - i * j + i * n - i * d = 0 \rangle$ 
    by (simp add: algebra-simps)
  from constraint[of  $\langle -i \rangle$   $\langle -1 \rangle$ ]
  have 20:  $\langle i * j - i * g - i * n + i * d = 0 \rangle$ 
    by (simp add: algebra-simps)
  from constraint[of i i]
  have 21:  $\langle j - g + n - d + 2 * i * p = 0 \rangle$ 
    by (simp add: algebra-simps)
  from constraint[of i  $\langle -i \rangle$ ]
  have 22:  $\langle g - j - n + d - 2 * i * p = 0 \rangle$ 
    by (simp add: algebra-simps)
  from constraint[of 2 1]
  have 23:  $\langle g + j + n + d = 0 \rangle$ 
    using 12 13  $\langle k = -p \rangle$  by algebra
  from aux2[OF 23 18]
  have [simp]:  $\langle g = -n \rangle$ 
    by algebra
  from 23
  have [simp]:  $\langle j = -d \rangle$ 
    by (simp add: add-eq-0-iff2)
  have  $8 * (i * p) + (4 * (i * d) + 4 * (i * n)) = 0$ 
    using constraint[of 2 i] by simp
  hence 24:  $\langle 2 * p + d + n = 0 \rangle$ 
    using complex-i-not-zero by algebra
  from aux2[OF 24 13]
  have [simp]:  $\langle p = 0 \rangle$ 
    by simp
  then have [simp]:  $\langle k = 0 \rangle$ 
    by auto

```

```

from 12
have ⟨g = - j⟩
  by simp
from 21
have ⟨d = - g⟩
  by auto

show ⟨d = 0⟩
  using refl[of d]
  apply (subst (asm) ⟨d = - g⟩)
  apply (subst (asm) ⟨g = - j⟩)
  apply (subst (asm) ⟨j = - d⟩)
  by simp
qed
then show ?thesis
  by (auto intro!: equal-ket cinner-ket-eqI
      simp: C-def cblinfun.diff-left cinner-diff-right
      simp flip: tensor-ell2-ket)
qed

lemma unitary-tensor-ell2-right-CARD-1:
  fixes ψ :: ⟨'a :: {CARD-1,enum} ell2⟩
  assumes ⟨norm ψ = 1⟩
  shows ⟨unitary (tensor-ell2-right ψ)⟩
proof (rule unitaryI)
  show ⟨tensor-ell2-right ψ* oCL tensor-ell2-right ψ = id-cblinfun⟩
    by (simp add: assms isometry-tensor-ell2-right)
  have *: ⟨(ψ •C φ) * (φ •C ψ) = φ •C φ⟩ for φ
  proof -
    define ψ' φ' where ⟨ψ' = 1 •C ψ⟩ and ⟨φ' = 1 •C φ⟩
    have ψ: ⟨ψ = ψ' *C 1⟩
      by (metis ψ'-def one-cinner-a-scaleC-one)
    have φ: ⟨φ = φ' *C 1⟩
      by (metis φ'-def one-cinner-a-scaleC-one)
    show ?thesis
      unfolding ψ φ
      by (metis (no-types, lifting) Groups.mult-ac(1) ψ assms cinner-simps(5) cinner-simps(6)
          norm-one of-complex-def of-complex-inner-1 power2-norm-eq-cinner)
  qed
  show ⟨tensor-ell2-right ψ oCL tensor-ell2-right ψ* = id-cblinfun⟩
    by (rule cblinfun-cinner-tensor-eqI) (simp add: *)
qed

```

14.2 Tensor product of operators on - ell2

definition *tensor-op* :: ⟨('a ell2, 'b ell2) cblinfun ⇒ ('c ell2, 'd ell2) cblinfun
 ⇒ (('a×'c) ell2, ('b×'d) ell2) cblinfun⟩ (**infixr** \otimes_o 70) **where**
 ⟨*tensor-op* M N = cblinfun-extension (range ket) (λk. case (inv ket k) of (x,y) ⇒ *tensor-ell2*
 (M *_V ket x) (N *_V ket y))⟩

lemma

— Loosely following [7, Section IV.1]

fixes $a :: \langle 'a \rangle$ **and** $b :: \langle 'b \rangle$ **and** $c :: \langle 'c \rangle$ **and** $d :: \langle 'd \rangle$ **and** $M :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \rangle$ **and** $N :: \langle 'c \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

shows *tensor-op-ell2*: $\langle (M \otimes_o N) *_V (\psi \otimes_s \varphi) = (M *_V \psi) \otimes_s (N *_V \varphi) \rangle$

and *tensor-op-norm*: $\langle \text{norm } (M \otimes_o N) = \text{norm } M * \text{norm } N \rangle$

proof —

define $S1 :: \langle ('a \times 'd) \text{ ell2 set} \rangle$ **and** $f1 \ g1 \ extg1$

where $\langle S1 = \text{range ket} \rangle$

and $\langle f1 \ k = (\text{case } (\text{inv ket } k) \text{ of } (x,y) \Rightarrow \text{tensor-ell2 } (M *_V \text{ket } x) (\text{ket } y)) \rangle$

and $\langle g1 = \text{cconstruct } S1 \ f1 \rangle$ **and** $\langle extg1 = \text{cblinfun-extension } (\text{cspan } S1) \ g1 \rangle$

for k

define $S2 :: \langle ('a \times 'c) \text{ ell2 set} \rangle$ **and** $f2 \ g2 \ extg2$

where $\langle S2 = \text{range ket} \rangle$

and $\langle f2 \ k = (\text{case } (\text{inv ket } k) \text{ of } (x,y) \Rightarrow \text{tensor-ell2 } (\text{ket } x) (N *_V \text{ket } y)) \rangle$

and $\langle g2 = \text{cconstruct } S2 \ f2 \rangle$ **and** $\langle extg2 = \text{cblinfun-extension } (\text{cspan } S2) \ g2 \rangle$

for k

define *tensorMN* **where** $\langle \text{tensorMN} = \text{extg1 } o_{CL} \ \text{extg2} \rangle$

have *extg1-ket*: $\langle \text{extg1 } *_V \text{ket } (x,y) = (M *_V \text{ket } x) \otimes_s \text{ket } y \rangle$

and *norm-extg1*: $\langle \text{norm } \text{extg1} \leq \text{norm } M \rangle$ **for** $x \ y$

proof —

have [*simp*]: $\langle \text{cindependent } S1 \rangle$

using *S1-def cindependent-ket* **by** *blast*

have [*simp*]: $\langle \text{closure } (\text{cspan } S1) = UNIV \rangle$

by (*simp add: S1-def*)

have [*simp*]: $\langle \text{ket } (x, y) \in \text{cspan } S1 \rangle$ **for** $x \ y$

by (*simp add: S1-def complex-vector.span-base*)

have *g1-f1*: $\langle g1 \ (\text{ket } (x,y)) = f1 \ (\text{ket } (x,y)) \rangle$ **for** $x \ y$

by (*metis S1-def cindependent S1 complex-vector.construct-basis g1-def rangeI*)

have [*simp*]: $\langle \text{clinear } g1 \rangle$

unfolding *g1-def* **using** $\langle \text{cindependent } S1 \rangle$ **by** (*rule complex-vector.linear-construct*)

then have *g1-add*: $\langle g1 \ (x + y) = g1 \ x + g1 \ y \rangle$ **if** $\langle x \in \text{cspan } S1 \rangle$ **and** $\langle y \in \text{cspan } S1 \rangle$ **for** $x \ y$

using *clinear-iff* **by** *blast*

from $\langle \text{clinear } g1 \rangle$ **have** *g1-scale*: $\langle g1 \ (c *_C x) = c *_C g1 \ x \rangle$ **if** $\langle x \in \text{cspan } S1 \rangle$ **for** $x \ c$

by (*simp add: complex-vector.linear-scale*)

have *g1-bounded*: $\langle \text{norm } (g1 \ \psi) \leq \text{norm } M * \text{norm } \psi \rangle$ **if** $\langle \psi \in \text{cspan } S1 \rangle$ **for** ψ

proof —

from that obtain $t \ r$ **where** $\langle \text{finite } t \rangle$ **and** $\langle t \subseteq \text{range ket} \rangle$ **and** $\psi\text{-tr}$: $\langle \psi = (\sum a \in t. r \ a *_C a) \rangle$

by (*smt (verit) complex-vector.span-explicit mem-Collect-eq S1-def*)

define $X \ Y$ **where** $\langle X = \text{fst } ' \text{inv ket } ' \ t \rangle$ **and** $\langle Y = \text{snd } ' \text{inv ket } ' \ t \rangle$

have *g1-ket*: $\langle g1 \ (\text{ket } (x,y)) = (M *_V \text{ket } x) \otimes_s \text{ket } y \rangle$ **for** $x \ y$

by (*simp add: g1-def S1-def complex-vector.construct-basis f1-def*)

define ξ **where** $\langle \xi \ y = (\sum x \in X. \text{if } (\text{ket } (x,y)) \in t \text{ then } r \ (\text{ket } (x,y)) *_C \text{ket } x \ \text{else } 0) \rangle$ **for**

y

```

have  $\psi\xi$ :  $\langle \psi = (\sum y \in Y. \xi y \otimes_s \text{ket } y) \rangle$ 
proof -
  have  $\langle (\sum y \in Y. \xi y \otimes_s \text{ket } y) = (\sum xy \in X \times Y. \text{if } \text{ket } xy \in t \text{ then } r (\text{ket } xy) *_C \text{ket } xy$ 
else 0)  $\rangle$ 
    unfolding  $\xi$ -def tensor-ell2-sum-left
    by (subst sum.swap)
    (auto simp: sum.cartesian-product tensor-ell2-scaleC1 tensor-ell2-ket intro!: sum.cong)
  also have  $\langle \dots = (\sum xy \in \text{ket } \langle (X \times Y). \text{if } xy \in t \text{ then } r xy *_C xy \text{ else } 0) \rangle$ 
    by (subst sum.reindex) (auto simp add: inj-on-def)
  also have  $\langle \dots = \psi \rangle$ 
    unfolding  $\psi$ -tr
proof (rule sum.mono-neutral-cong-right, goal-cases)
  case 2
  show  $t \subseteq \text{ket } \langle (X \times Y) \rangle$ 
  proof
    fix  $x$  assume  $x \in t$ 
    with  $\langle t \subseteq \text{range } \text{ket} \rangle$  obtain  $a b$  where  $ab: x = \text{ket } (a, b)$ 
    by fast
    also have  $\text{ket } (a, b) \in \text{ket } \langle (X \times Y) \rangle$ 
    by (metis X-def Y-def  $\langle x \in t \rangle$  ab f-inv-into-f fst-conv image-eq1
ket-injective mem-Sigma-iff rangeI snd-conv)
    finally show  $x \in \text{ket } \langle (X \times Y) \rangle$  .
  qed
qed (auto simp add: X-def Y-def  $\langle \text{finite } t \rangle$ )
finally show ?thesis
  by simp
qed
have  $\langle (\text{norm } (g1 \ \psi))^2 = (\text{norm } (\sum y \in Y. (M *_V \xi y) \otimes_s \text{ket } y))^2 \rangle$ 
  by (auto simp:  $\psi\xi$  complex-vector.linear-sum  $\xi$ -def tensor-ell2-sum-left
complex-vector.linear-scale g1-ket tensor-ell2-scaleC1
complex-vector.linear-0 tensor-ell2-ket
intro!: sum.cong arg-cong[where  $f = \text{norm}$ ])
also have  $\langle \dots = (\sum y \in Y. (\text{norm } ((M *_V \xi y) \otimes_s \text{ket } y))^2 \rangle$ 
  unfolding Y-def by (rule pythagorean-theorem-sum) (use  $\langle \text{finite } t \rangle$  in auto)
also have  $\langle \dots = (\sum y \in Y. (\text{norm } (M *_V \xi y))^2 \rangle$ 
  by (simp add: norm-tensor-ell2)
also have  $\langle \dots \leq (\sum y \in Y. (\text{norm } M * \text{norm } (\xi y))^2 \rangle$ 
  by (meson norm-cblinfun norm-ge-zero power-mono sum-mono)
also have  $\langle \dots = (\text{norm } M)^2 * (\sum y \in Y. (\text{norm } (\xi y \otimes_s \text{ket } y))^2 \rangle$ 
  by (simp add: power-mult-distrib norm-tensor-ell2 flip: sum-distrib-left)
also have  $\langle \dots = (\text{norm } M)^2 * (\text{norm } (\sum y \in Y. \xi y \otimes_s \text{ket } y))^2 \rangle$ 
  unfolding Y-def
  by (subst pythagorean-theorem-sum) (use  $\langle \text{finite } t \rangle$  in auto)
also have  $\langle \dots = (\text{norm } M)^2 * (\text{norm } \psi)^2 \rangle$ 
  using  $\psi\xi$  by fastforce
finally show  $\langle \text{norm } (g1 \ \psi) \leq \text{norm } M * \text{norm } \psi \rangle$ 
  by (metis mult-nonneg-nonneg norm-ge-zero power2-le-imp-le power-mult-distrib)
qed

```

```

have extg1-exists: ⟨cblinfun-extension-exists (cspan S1) g1⟩
  by (rule cblinfun-extension-exists-bounded-dense[where B=⟨norm M⟩])
    (use g1-add g1-scale g1-bounded in auto)

then show ⟨extg1 *V ket (x,y) = (M *V ket x) ⊗s ket y⟩ for x y
  by (simp add: extg1-def cblinfun-extension-apply g1-f1 f1-def)

from g1-add g1-scale g1-bounded
show ⟨norm extg1 ≤ norm M⟩
  by (auto simp: extg1-def intro!: cblinfun-extension-norm-bounded-dense)
qed

have extg1-apply: ⟨extg1 *V (ψ ⊗s φ) = (M *V ψ) ⊗s φ⟩ for ψ φ
proof -
  have 1: ⟨bounded-clinear (λa. extg1 *V (a ⊗s ket y))⟩ for y
    by (intro bounded-clinear-cblinfun-apply bounded-clinear-tensor-ell22)
  have 2: ⟨bounded-clinear (λa. (M *V a) ⊗s ket y)⟩ for y :: 'd
  by (auto intro!: bounded-clinear-tensor-ell22[THEN bounded-clinear-compose] bounded-clinear-cblinfun-apply)
  have 3: ⟨bounded-clinear (λa. extg1 *V (ψ ⊗s a))⟩
    by (intro bounded-clinear-cblinfun-apply bounded-clinear-tensor-ell21)
  have 4: ⟨bounded-clinear ((⊗s) (M *V ψ))⟩
  by (auto intro!: bounded-clinear-tensor-ell21[THEN bounded-clinear-compose] bounded-clinear-cblinfun-apply)

  have eq-ket: ⟨extg1 *V tensor-ell2 ψ (ket y) = tensor-ell2 (M *V ψ) (ket y)⟩ for y
    by (rule bounded-clinear-eq-on-closure[where t=ψ and G=⟨range ket⟩])
      (use 1 2 extg1-ket in ⟨auto simp: tensor-ell2-ket⟩)
  show ?thesis
    by (rule bounded-clinear-eq-on-closure[where t=φ and G=⟨range ket⟩])
      (use 3 4 eq-ket in auto)
qed

have extg2-ket: ⟨extg2 *V ket (x,y) = ket x ⊗s (N *V ket y)⟩
and norm-extg2: ⟨norm extg2 ≤ norm N⟩ for x y
proof -
  have [simp]: ⟨cindependent S2⟩
    using S2-def cindependent-ket by blast
  have [simp]: ⟨closure (cspan S2) = UNIV⟩
    by (simp add: S2-def)
  have [simp]: ⟨ket (x, y) ∈ cspan S2⟩ for x y
    by (simp add: S2-def complex-vector.span-base)
  have g2-f2: ⟨g2 (ket (x,y)) = f2 (ket (x,y))⟩ for x y
    by (metis S2-def ⟨cindependent S2⟩ complex-vector.construct-basis g2-def rangeI)
  have [simp]: ⟨clinear g2⟩
    unfolding g2-def using ⟨cindependent S2⟩ by (rule complex-vector.linear-construct)
  then have g2-add: ⟨g2 (x + y) = g2 x + g2 y⟩ if ⟨x ∈ cspan S2⟩ and ⟨y ∈ cspan S2⟩ for
x y
    using clinear-iff by blast
  from ⟨clinear g2⟩ have g2-scale: ⟨g2 (c *C x) = c *C g2 x⟩ if ⟨x ∈ cspan S2⟩ for x c
    by (simp add: complex-vector.linear-scale)

```

have $g2\text{-bounded}$: $\langle \text{norm } (g2 \ \psi) \leq \text{norm } N * \text{norm } \psi \rangle$ **if** $\langle \psi \in \text{cspan } S2 \rangle$ **for** ψ
proof –
from that obtain $t \ r$ **where** $\langle \text{finite } t \rangle$ **and** $\langle t \subseteq \text{range } \text{ket} \rangle$ **and** $\psi\text{-tr}$: $\langle \psi = (\sum_{a \in t} r \ a *_{\mathbb{C}} a) \rangle$
by (*smt (verit) complex-vector.span-explicit mem-Collect-eq S2-def*)
define $X \ Y$ **where** $\langle X = \text{fst } ' \text{inv } \text{ket } ' \ t \rangle$ **and** $\langle Y = \text{snd } ' \text{inv } \text{ket } ' \ t \rangle$
have $g2\text{-ket}$: $\langle g2 \ (\text{ket } (x,y)) = \text{ket } x \otimes_s (N *_{\mathbb{V}} \text{ket } y) \rangle$ **for** $x \ y$
by (*auto simp add: f2-def complex-vector.construct-basis g2-def S2-def*)
define ξ **where** $\langle \xi \ x = (\sum_{y \in Y} \text{if } (\text{ket } (x,y) \in t) \text{ then } r \ (\text{ket } (x,y)) *_{\mathbb{C}} \text{ket } y \text{ else } 0) \rangle$ **for**
 x
have $\psi\xi$: $\langle \psi = (\sum_{x \in X} \text{ket } x \otimes_s \xi \ x) \rangle$
proof –
have $\langle (\sum_{x \in X} \text{ket } x \otimes_s \xi \ x) = (\sum_{xy \in X \times Y} \text{if } \text{ket } xy \in t \text{ then } r \ (\text{ket } xy) *_{\mathbb{C}} \text{ket } xy \text{ else } 0) \rangle$
by (*auto simp: \xi-def tensor-ell2-sum-right sum.cartesian-product tensor-ell2-scaleC2 tensor-ell2-ket intro!: sum.cong*)
also have $\langle \dots = (\sum_{xy \in \text{ket } ' (X \times Y)} \text{if } xy \in t \text{ then } r \ xy *_{\mathbb{C}} xy \text{ else } 0) \rangle$
by (*subst sum.reindex*) (*auto simp add: inj-on-def*)
also have $\langle \dots = \psi \rangle$
unfolding $\psi\text{-tr}$
proof (*rule sum.mono-neutral-cong-right, goal-cases*)
case 2
show $t \subseteq \text{ket } ' (X \times Y)$
proof
fix x **assume** $x \in t$
with $\langle t \subseteq \text{range } \text{ket} \rangle$ **obtain** $a \ b$ **where** ab : $x = \text{ket } (a, b)$
by *fast*
also have $\text{ket } (a, b) \in \text{ket } ' (X \times Y)$
by (*metis X-def Y-def \langle x \in t \rangle ab f-inv-into-f fst-conv image-eq1 ket-injective mem-Sigma-iff rangeI snd-conv*)
finally show $x \in \text{ket } ' (X \times Y)$.
qed
qed (*auto simp add: X-def Y-def \langle finite t \rangle*)
finally show *?thesis*
by *simp*
qed
have $\langle (\text{norm } (g2 \ \psi))^2 = (\text{norm } (\sum_{x \in X} \text{ket } x \otimes_s (N *_{\mathbb{V}} \xi \ x)))^2 \rangle$
by (*auto simp: \psi\xi complex-vector.linear-sum \xi-def tensor-ell2-sum-right complex-vector.linear-scale g2-ket tensor-ell2-scaleC2 complex-vector.linear-0 tensor-ell2-ket intro!: sum.cong arg-cong[where f=norm]*)
also have $\langle \dots = (\sum_{x \in X} (\text{norm } (\text{ket } x \otimes_s (N *_{\mathbb{V}} \xi \ x)))^2 \rangle$
unfolding $X\text{-def}$ **by** (*rule pythagorean-theorem-sum*) (*use \langle finite t \rangle in auto*)
also have $\langle \dots = (\sum_{x \in X} (\text{norm } (N *_{\mathbb{V}} \xi \ x))^2 \rangle$
by (*simp add: norm-tensor-ell2*)
also have $\langle \dots \leq (\sum_{x \in X} (\text{norm } N * \text{norm } (\xi \ x))^2 \rangle$
by (*meson norm-cblinfun norm-ge-zero power-mono sum-mono*)
also have $\langle \dots = (\text{norm } N)^2 * (\sum_{x \in X} (\text{norm } (\text{ket } x \otimes_s \xi \ x))^2 \rangle$

by (simp add: power-mult-distrib norm-tensor-ell2 flip: sum-distrib-left)
 also have $\langle \dots = (\text{norm } N)^2 * (\text{norm } (\sum x \in X. \text{ket } x \otimes_s \xi x))^2 \rangle$
 unfolding X-def by (subst pythagorean-theorem-sum) (use <finite t> in auto)
 also have $\langle \dots = (\text{norm } N)^2 * (\text{norm } \psi)^2 \rangle$
 using $\psi\xi$ by fastforce
 finally show $\langle \text{norm } (g2 \ \psi) \leq \text{norm } N * \text{norm } \psi \rangle$
 by (metis mult-nonneg-nonneg norm-ge-zero power2-le-imp-le power-mult-distrib)

qed

have extg2-exists: $\langle \text{cblinfun-extension-exists } (cspan \ S2) \ g2 \rangle$
 by (rule cblinfun-extension-exists-bounded-dense[where B= $\langle \text{norm } N \rangle$])
 (use g2-add g2-scale g2-bounded in auto)

then show $\langle \text{extg2 } *_{\mathcal{V}} \text{ket } (x,y) = \text{ket } x \otimes_s N *_{\mathcal{V}} \text{ket } y \rangle$ for $x \ y$
 by (simp add: extg2-def cblinfun-extension-apply g2-f2 f2-def)

from g2-add g2-scale g2-bounded
 show $\langle \text{norm } \text{extg2} \leq \text{norm } N \rangle$
 by (auto simp: extg2-def intro!: cblinfun-extension-norm-bounded-dense)

qed

have extg2-apply: $\langle \text{extg2 } *_{\mathcal{V}} (\psi \otimes_s \varphi) = \psi \otimes_s (N *_{\mathcal{V}} \varphi) \rangle$ for $\psi \ \varphi$

proof –

have 1: $\langle \text{bounded-clinear } (\lambda a. \text{extg2 } *_{\mathcal{V}} (\text{ket } x \otimes_s a)) \rangle$ for x
 by (intro bounded-clinear-cblinfun-apply bounded-clinear-tensor-ell21)

have 2: $\langle \text{bounded-clinear } (\lambda a. \text{ket } x \otimes_s (N *_{\mathcal{V}} a)) \rangle$ for $x :: 'a$
 by (auto intro!: bounded-clinear-tensor-ell21 [THEN bounded-clinear-compose] bounded-clinear-cblinfun-apply)

have 3: $\langle \text{bounded-clinear } (\lambda a. \text{extg2 } *_{\mathcal{V}} (a \otimes_s \varphi)) \rangle$
 by (intro bounded-clinear-cblinfun-apply bounded-clinear-tensor-ell22)

have 4: $\langle \text{bounded-clinear } (\lambda a. a \otimes_s (N *_{\mathcal{V}} \varphi)) \rangle$
 by (auto intro!: bounded-clinear-tensor-ell22 [THEN bounded-clinear-compose] bounded-clinear-cblinfun-apply)

have eq-ket: $\langle \text{extg2 } *_{\mathcal{V}} (\text{ket } x \otimes_s \varphi) = \text{ket } x \otimes_s (N *_{\mathcal{V}} \varphi) \rangle$ for x
 by (rule bounded-clinear-eq-on-closure[where t= φ and G= $\langle \text{range } \text{ket} \rangle$])
 (use 1 2 extg2-ket in <auto simp: tensor-ell2-ket>)

show ?thesis
 by (rule bounded-clinear-eq-on-closure[where t= ψ and G= $\langle \text{range } \text{ket} \rangle$])
 (use 3 4 eq-ket in auto)

qed

have tensorMN-apply: $\langle \text{tensorMN } *_{\mathcal{V}} (\psi \otimes_s \varphi) = (M *_{\mathcal{V}} \psi) \otimes_s (N *_{\mathcal{V}} \varphi) \rangle$ for $\psi \ \varphi$
 by (simp add: extg1-apply extg2-apply tensorMN-def)

have $\langle \text{cblinfun-extension-exists } (\text{range } \text{ket}) \ (\lambda k. \text{case inv ket } k \text{ of } (x, y) \Rightarrow (M *_{\mathcal{V}} \text{ket } x) \otimes_s (N *_{\mathcal{V}} \text{ket } y)) \rangle$
 by (rule cblinfun-extension-existsI[where B=tensorMN])
 (use tensorMN-apply[of <ket -> <ket ->] in <auto simp: tensor-ell2-ket>)

then have otimes-ket: $\langle (M \otimes_o N) *_{\mathcal{V}} (\text{ket } (a,c)) = (M *_{\mathcal{V}} \text{ket } a) \otimes_s (N *_{\mathcal{V}} \text{ket } c) \rangle$ for $a \ c$

by (*simp add: tensor-op-def cblinfun-extension-apply*)

have *tensorMN-otimes*: $\langle M \otimes_o N = \text{tensorMN} \rangle$
by (*rule equal-ket*)
 (*use tensorMN-apply*[of $\langle \text{ket } \rightarrow \text{ket } \rightarrow \rangle$] **in** *auto simp: otimes-ket tensor-ell2-ket*)

show *otimes-apply*: $\langle (M \otimes_o N) *_V (\psi \otimes_s \varphi) = (M *_V \psi) \otimes_s (N *_V \varphi) \rangle$ **for** $\psi \varphi$
by (*simp add: tensorMN-apply tensorMN-otimes*)

show $\langle \text{norm } (M \otimes_o N) = \text{norm } M * \text{norm } N \rangle$
proof (*rule order.antisym*)

show $\langle \text{norm } (M \otimes_o N) \leq \text{norm } M * \text{norm } N \rangle$
unfolding *tensorMN-otimes tensorMN-def*
by (*smt* (*verit, best*) *mult-mono norm-cblinfun-compose norm-extg1 norm-extg2 norm-ge-zero*)
have $\langle \text{norm } (M \otimes_o N) \geq \text{norm } M * \text{norm } N * \varepsilon \rangle$ **if** $\langle \varepsilon < 1 \rangle$ **and** $\langle \varepsilon > 0 \rangle$ **for** ε
proof –

obtain ψa **where** $1: \langle \text{norm } (M *_V \psi a) \geq \text{norm } M * \text{sqrt } \varepsilon \rangle$ **and** $\langle \text{norm } \psi a = 1 \rangle$
by (*atomize-elim, rule cblinfun-norm-approx-witness-mult*[**where** $\varepsilon = \text{sqrt } \varepsilon$ **and** $A = M$])
 (*use* $\langle \varepsilon < 1 \rangle$ **in** *auto*)

obtain ψb **where** $2: \langle \text{norm } (N *_V \psi b) \geq \text{norm } N * \text{sqrt } \varepsilon \rangle$ **and** $\langle \text{norm } \psi b = 1 \rangle$
by (*atomize-elim, rule cblinfun-norm-approx-witness-mult*[**where** $\varepsilon = \text{sqrt } \varepsilon$ **and** $A = N$])
 (*use* $\langle \varepsilon < 1 \rangle$ **in** *auto*)

have $\langle \text{norm } ((M \otimes_o N) *_V (\psi a \otimes_s \psi b)) / \text{norm } (\psi a \otimes_s \psi b) = \text{norm } ((M \otimes_o N) *_V (\psi a \otimes_s \psi b)) \rangle$
using $\langle \text{norm } \psi a = 1 \rangle \langle \text{norm } \psi b = 1 \rangle$
by (*simp add: norm-tensor-ell2*)

also have $\langle \dots = \text{norm } (M *_V \psi a) * \text{norm } (N *_V \psi b) \rangle$
by (*simp add: norm-tensor-ell2 otimes-apply*)

also from $1 \ 2$ **have** $\langle \dots \geq (\text{norm } M * \text{sqrt } \varepsilon) * (\text{norm } N * \text{sqrt } \varepsilon) \rangle$ (**is** $\langle - \geq \dots \rangle$)
by (*rule mult-mono'*) (*use* $\langle \varepsilon > 0 \rangle$ **in** *auto*)

also have $\langle \dots = \text{norm } M * \text{norm } N * \varepsilon \rangle$
using $\langle \varepsilon > 0 \rangle$ **by** *force*

finally show *?thesis*
using *cblinfun-norm-geqI* **by** *blast*

qed

then show $\langle \text{norm } (M \otimes_o N) \geq \text{norm } M * \text{norm } N \rangle$
by (*metis field-le-mult-one-interval mult commute*)

qed

qed

lemma *tensor-op-ket*: $\langle \text{tensor-op } M \ N *_V (\text{ket } (a, c)) = \text{tensor-ell2 } (M *_V \text{ket } a) (N *_V \text{ket } c) \rangle$
by (*metis tensor-ell2-ket tensor-op-ell2*)

lemma *comp-tensor-op*: $(\text{tensor-op } a \ b) \ o_{CL} (\text{tensor-op } c \ d) = \text{tensor-op } (a \ o_{CL} \ c) (b \ o_{CL} \ d)$
for $a :: 'e \ \text{ell2} \Rightarrow_{CL} 'c \ \text{ell2}$ **and** $b :: 'f \ \text{ell2} \Rightarrow_{CL} 'd \ \text{ell2}$ **and**
 $c :: 'a \ \text{ell2} \Rightarrow_{CL} 'e \ \text{ell2}$ **and** $d :: 'b \ \text{ell2} \Rightarrow_{CL} 'f \ \text{ell2}$
by (*rule equal-ket*) (*auto simp flip: tensor-ell2-ket simp: tensor-op-ell2*)

lemma *tensor-op-left-add*: $\langle (x + y) \otimes_o b = x \otimes_o b + y \otimes_o b \rangle$

for $x\ y :: \langle 'a\ ell2 \Rightarrow_{CL} 'c\ ell2 \rangle$ **and** $b :: \langle 'b\ ell2 \Rightarrow_{CL} 'd\ ell2 \rangle$
by (*auto intro!*: *equal-ket simp: tensor-op-ket plus-cblinfun.rep-eq tensor-ell2-add1*)

lemma *tensor-op-right-add*: $\langle b \otimes_o (x + y) = b \otimes_o x + b \otimes_o y \rangle$
for $x\ y :: \langle 'a\ ell2 \Rightarrow_{CL} 'c\ ell2 \rangle$ **and** $b :: \langle 'b\ ell2 \Rightarrow_{CL} 'd\ ell2 \rangle$
by (*auto intro!*: *equal-ket simp: tensor-op-ket plus-cblinfun.rep-eq tensor-ell2-add2*)

lemma *tensor-op-scaleC-left*: $\langle (c *_C x) \otimes_o b = c *_C (x \otimes_o b) \rangle$
for $x :: \langle 'a\ ell2 \Rightarrow_{CL} 'c\ ell2 \rangle$ **and** $b :: \langle 'b\ ell2 \Rightarrow_{CL} 'd\ ell2 \rangle$
by (*auto intro!*: *equal-ket simp: tensor-op-ket tensor-ell2-scaleC1*)

lemma *tensor-op-scaleC-right*: $\langle b \otimes_o (c *_C x) = c *_C (b \otimes_o x) \rangle$
for $x :: \langle 'a\ ell2 \Rightarrow_{CL} 'c\ ell2 \rangle$ **and** $b :: \langle 'b\ ell2 \Rightarrow_{CL} 'd\ ell2 \rangle$
by (*auto intro!*: *equal-ket simp: tensor-op-ket tensor-ell2-scaleC2*)

lemma *tensor-op-bounded-cbilinear[simp]*: $\langle \text{bounded-cbilinear tensor-op} \rangle$
by (*auto intro!*: *bounded-cbilinear.intro exI[of - 1]*
simp: tensor-op-left-add tensor-op-right-add tensor-op-scaleC-left tensor-op-scaleC-right tensor-op-norm)

lemma *tensor-op-cbilinear[simp]*: $\langle \text{cbilinear tensor-op} \rangle$
by (*simp add: bounded-cbilinear.add-left bounded-cbilinear.add-right cbilinear-def clinearI tensor-op-scaleC-left tensor-op-scaleC-right*)

lemma *tensor-butter*: $\langle \text{butterfly (ket } i) \text{ (ket } j) \otimes_o \text{ butterfly (ket } k) \text{ (ket } l) = \text{butterfly (ket } (i,k) \text{) (ket } (j,l) \text{)} \rangle$
by (*rule equal-ket*)
(auto simp flip: tensor-ell2-ket simp: tensor-op-ell2 butterfly-def tensor-ell2-scaleC1 tensor-ell2-scaleC2)

lemma *cspan-tensor-op-butter*: $\langle \text{cspan } \{ \text{tensor-op (butterfly (ket } i) \text{ (ket } j) \text{) (butterfly (ket } k) \text{ (ket } l) \text{)} \mid (i:::\text{finite}) (j:::\text{finite}) (k:::\text{finite}) (l:::\text{finite}). \text{ True} \} = \text{UNIV} \rangle$
unfolding *tensor-butter*
by (*subst cspan-butterfly-ket[symmetric]*) (*metis surj-pair*)

lemma *cindependent-tensor-op-butter*: $\langle \text{cindependent } \{ \text{tensor-op (butterfly (ket } i) \text{ (ket } j) \text{) (butterfly (ket } k) \text{ (ket } l) \text{)} \mid i\ j\ k\ l. \text{ True} \} \rangle$
unfolding *tensor-butter*
using *cindependent-butterfly-ket*
by (*smt (z3) Collect-mono-iff complex-vector.independent-mono*)

lift-definition *right-amplification* :: $\langle ('a\ ell2 \Rightarrow_{CL} 'b\ ell2) \Rightarrow_{CL} (('a \times 'c)\ ell2 \Rightarrow_{CL} ('b \times 'c)\ ell2) \rangle$ **is**
 $\langle \lambda a. a \otimes_o \text{id-cblinfun} \rangle$
by (*simp add: bounded-cbilinear.bounded-clinear-left*)

lift-definition *left-amplification* :: $\langle ('a\ ell2 \Rightarrow_{CL} 'b\ ell2) \Rightarrow_{CL} (('c \times 'a)\ ell2 \Rightarrow_{CL} ('c \times 'b)\ ell2) \rangle$ **is**
 $\langle \lambda a. \text{id-cblinfun} \otimes_o a \rangle$

by (simp add: bounded-cbilinear.bounded-clinear-right)

lemma sandwich-tensor-ell2-right: $\langle \text{sandwich } (\text{tensor-ell2-right } \psi^*) *_V a \otimes_o b = (\psi \cdot_C (b *_V \psi)) *_C a \rangle$

by (rule cblinfun-eqI) (simp add: sandwich-apply tensor-op-ell2)

lemma sandwich-tensor-ell2-left: $\langle \text{sandwich } (\text{tensor-ell2-left } \psi^*) *_V a \otimes_o b = (\psi \cdot_C (a *_V \psi)) *_C b \rangle$

by (rule cblinfun-eqI) (simp add: sandwich-apply tensor-op-ell2)

lemma tensor-op-adjoint: $\langle (\text{tensor-op } a b)^* = \text{tensor-op } (a^*) (b^*) \rangle$

by (rule cinner-ket-adjointI[symmetric])

(auto simp flip: tensor-ell2-ket simp: tensor-op-ell2 cinner-adj-left)

lemma has-sum-id-tensor-butterfly-ket: $\langle ((\lambda i. (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i)) *_V \psi) \text{ has-sum } \psi) \text{ UNIV} \rangle$

proof –

have *: $\langle (\sum i \in F. (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i)) *_V \psi) = \text{trunc-ell2 } (\text{UNIV} \times F) \psi \rangle$ **if** $\langle \text{finite } F \rangle$ **for** F

proof (rule Rep-ell2-inject[THEN iffD1], rule ext, rename-tac xy)

fix xy :: $\langle b \times a \rangle$

obtain x y **where** xy: $\langle xy = (x, y) \rangle$

by fastforce

have $\langle \text{Rep-ell2 } (\sum i \in F. (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i)) *_V \psi) xy = \text{ket } xy \cdot_C (\sum i \in F. (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i)) *_V \psi) \rangle$

by (simp add: cinner-ket-left)

also have $\langle \dots = (\sum i \in F. \text{ket } xy \cdot_C ((\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i)) *_V \psi)) \rangle$

using cinner-sum-right by blast

also have $\langle \dots = (\sum i \in F. \text{ket } xy \cdot_C ((\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i))^* *_V \psi)) \rangle$

by (simp add: tensor-op-adjoint)

also have $\langle \dots = (\sum i \in F. ((\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i)) *_V \text{ket } xy) \cdot_C \psi) \rangle$

by (meson cinner-adj-right)

also have $\langle \dots = \text{of-bool } (y \in F) * (\text{ket } xy \cdot_C \psi) \rangle$

by (subst sum-single[where i=y])

(auto simp: xy tensor-op-ell2 cinner-ket that simp flip: tensor-ell2-ket)

also have $\langle \dots = \text{of-bool } (y \in F) * (\text{Rep-ell2 } \psi xy) \rangle$

by (simp add: cinner-ket-left)

also have $\langle \dots = \text{Rep-ell2 } (\text{trunc-ell2 } (\text{UNIV} \times F) \psi) xy \rangle$

by (simp add: trunc-ell2.rep-eq xy)

finally show $\langle \text{Rep-ell2 } (\sum i \in F. (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } i) (\text{ket } i)) *_V \psi) xy = \dots \rangle$

by –

qed

have $\langle ((\lambda F. \text{trunc-ell2 } F \psi) \longrightarrow \text{trunc-ell2 } \text{UNIV } \psi) (\text{filtermap } ((\times) \text{UNIV}) (\text{finite-subsets-at-top } \text{UNIV})) \rangle$

by (rule trunc-ell2-lim-general)

(auto simp add: filterlim-def le-filter-def eventually-finite-subsets-at-top eventually-filtermap intro!: exI[where x=snd ‘-’])

then have $\langle (\lambda F. \text{trunc-ell2 } (UNIV \times F) \psi) \longrightarrow \psi \text{ (finite-subsets-at-top UNIV)} \rangle$
by (*simp add: filterlim-def filtermap-filtermap*)
then have $\langle (\lambda F. (\sum i \in F. (id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ i) (ket\ i)) *_V \psi)) \longrightarrow \psi \text{ (finite-subsets-at-top UNIV)} \rangle$
by (*rule Lim-transform-eventually*)
*(simp add: * eventually-finite-subsets-at-top-weakI)*
then show *?thesis*
by (*simp add: has-sum-def*)
qed

lemma tensor-op-dense: $\langle \text{cstrong-operator-topology closure-of } (cspan \{a \otimes_o b \mid a\ b.\ True\}) = UNIV \rangle$

— [7, p.185 (10)], but we prove it directly.

proof (*intro order.antisym subset-UNIV subsetI*)

fix $c :: \langle ('a \times 'b) \text{ ell2} \Rightarrow_{CL} ('c \times 'd) \text{ ell2} \rangle$

define c' **where** $\langle c' \ i \ j = (tensor\text{-ell2-right } (ket\ i)) *_O_{CL} \ c \ *_O_{CL} \ tensor\text{-ell2-right } (ket\ j) \rangle$ **for** $i \ j$

define $AB :: \langle ('a \times 'b) \text{ ell2} \Rightarrow_{CL} ('c \times 'd) \text{ ell2} \rangle \text{ set}$ **where**

$\langle AB = \text{cstrong-operator-topology closure-of } (cspan \{a \otimes_o b \mid a\ b.\ True\}) \rangle$

have [*simp*]: $\langle \text{closedin cstrong-operator-topology } AB \rangle$

by (*simp add: AB-def*)

have [*simp*]: $\langle \text{csubspace } AB \rangle$

using *AB-def sot-closure-is-csubspace'* **by** *blast*

have $*$: $\langle c' \ i \ j \otimes_o \text{ butterfly } (ket\ i) (ket\ j) = (id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ i) (ket\ i)) *_O_{CL} \ c \ *_O_{CL} \ (id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ j) (ket\ j)) \rangle$ **for** $i \ j$

proof (*rule equal-ket, rule cinner-ket-eqI, rename-tac a b*)

fix $a :: \langle 'a \times 'b \rangle$ **and** $b :: \langle 'c \times 'd \rangle$

obtain $bi \ bj \ ai \ aj$ **where** $b: \langle b = (bi, bj) \rangle$ **and** $a: \langle a = (ai, aj) \rangle$

by (*meson surj-pair*)

have $\langle ket\ b \cdot_C ((c' \ i \ j \otimes_o \text{ butterfly } (ket\ i) (ket\ j)) *_V \ ket\ a) = of\text{-bool } (j = aj \wedge bj = i) * ((ket\ bi \otimes_s \ ket\ i) \cdot_C (c *_V \ ket\ ai \otimes_s \ ket\ aj)) \rangle$

by (*auto simp add: a b tensor-op-ell2 cinner-ket c'-def cinner-adj-right*)

simp flip: tensor-ell2-ket)

also have $\langle \dots = ket\ b \cdot_C ((id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ i) (ket\ i)) *_O_{CL} \ c \ *_O_{CL} \ id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ j) (ket\ j)) *_V \ ket\ a \rangle$

apply (*subst asm-rl[of $\langle id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ i) (ket\ i) = (id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ i) (ket\ i)) * \rangle$]*)

subgoal

by (*simp add: tensor-op-adjoint*)

subgoal

by (*auto simp: a b tensor-op-ell2 cinner-adj-right cinner-ket*)

simp flip: tensor-ell2-ket)

done

finally show $\langle ket\ b \cdot_C ((c' \ i \ j \otimes_o \text{ butterfly } (ket\ i) (ket\ j)) *_V \ ket\ a) =$

$ket\ b \cdot_C ((id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ i) (ket\ i)) *_O_{CL} \ c \ *_O_{CL} \ id\text{-cblinfun } \otimes_o \text{ butterfly } (ket\ j) (ket\ j)) *_V \ ket\ a \rangle$

```

    by -
qed

have ⟨c' i j ⊗o butterfly (ket i) (ket j) ∈ AB⟩ for i j
proof -
  have ⟨c' i j ⊗o butterfly (ket i) (ket j) ∈ {a ⊗o b | a b. True}⟩
    by auto
  also have ⟨... ⊆ cspan ...⟩
    by (simp add: complex-vector.span-superset)
  also have ⟨... ⊆ cstrong-operator-topology closure-of ...⟩
    by (rule closure-of-subset) simp
  also have ⟨... = AB⟩
    by (simp add: AB-def)
  finally show ?thesis
    by simp
qed
with * have AB1: ⟨(id-cblinfun ⊗o butterfly (ket i) (ket i)) oCL c oCL (id-cblinfun ⊗o butterfly
(ket j) (ket j)) ∈ AB⟩ for i j
  by simp
  have ⟨((λi. ((id-cblinfun ⊗o butterfly (ket i) (ket i)) oCL c oCL (id-cblinfun ⊗o butterfly (ket
j) (ket j))) *V ψ)
    has-sum (c oCL (id-cblinfun ⊗o butterfly (ket j) (ket j))) *V ψ) UNIV⟩ for j ψ
    by (simp add: has-sum-id-tensor-butterfly-ket)
  then have AB2: ⟨(c oCL (id-cblinfun ⊗o butterfly (ket j) (ket j))) ∈ AB⟩ for j
    by (rule has-sum-closed-cstrong-operator-topology[rotated -1]) (use AB1 in auto)

  have ⟨((λj. (c oCL (id-cblinfun ⊗o butterfly (ket j) (ket j))) *V ψ) has-sum c *V ψ) UNIV⟩
for ψ
  by (simp add: has-sum-cblinfun-apply has-sum-id-tensor-butterfly-ket)
  then show AB3: ⟨c ∈ AB⟩
    by (rule has-sum-closed-cstrong-operator-topology[rotated -1]) (use AB2 in auto)
qed

lemma tensor-extensionality-finite:
fixes F G :: ⟨(((a::finite × b::finite) ell2) ⇒CL ((c::finite × d::finite) ell2)) ⇒ 'e::complex-vector⟩
assumes [simp]: clinear F clinear G
assumes tensor-eq: (∧ a b. F (tensor-op a b) = G (tensor-op a b))
shows F = G
proof (rule ext, rule complex-vector.linear-eq-on-span[where f=F and g=G])
  show ⟨cllinear F⟩ and ⟨cllinear G⟩
    using assms by (simp-all add: cbilinear-def)
  show ⟨x ∈ cspan {tensor-op (butterfly (ket i) (ket j)) (butterfly (ket k) (ket l)) | i j k l. True}⟩

    for x :: ⟨('a × 'b) ell2 ⇒CL ('c × 'd) ell2⟩
    using cspan-tensor-op-butterfly by auto
  show ⟨F x = G x⟩ if ⟨x ∈ {tensor-op (butterfly (ket i) (ket j)) (butterfly (ket k) (ket l)) | i j k
l. True}⟩ for x
    using that by (auto simp: tensor-eq)

```

qed

lemma *tensor-id[simp]*: $\langle \text{tensor-op id-cblinfun id-cblinfun} = \text{id-cblinfun} \rangle$
by (*rule equal-ket*) (*auto simp flip: tensor-ell2-ket simp: tensor-op-ell2*)

lemma *tensor-butterfly*: $\text{tensor-op} (\text{butterfly } \psi \ \psi') (\text{butterfly } \varphi \ \varphi') = \text{butterfly} (\text{tensor-ell2 } \psi \ \varphi) (\text{tensor-ell2 } \psi' \ \varphi')$
by (*rule equal-ket*)
(*auto simp flip: tensor-ell2-ket simp: tensor-op-ell2 butterfly-def tensor-ell2-scaleC1 tensor-ell2-scaleC2*)

definition *tensor-lift* :: $\langle (('a1::\text{finite ell2}} \Rightarrow_{CL} 'a2::\text{finite ell2}}) \Rightarrow ('b1::\text{finite ell2}} \Rightarrow_{CL} 'b2::\text{finite ell2}}) \Rightarrow 'c \rangle$
 $\Rightarrow (((('a1 \times 'b1) \text{ ell2}} \Rightarrow_{CL} ('a2 \times 'b2) \text{ ell2}}) \Rightarrow 'c::\text{complex-normed-vector}) \rangle$

where

tensor-lift F2 = (*SOME G. clinear G* \wedge ($\forall a \ b. G (\text{tensor-op } a \ b) = F2 \ a \ b$))

lemma

fixes *F2* :: $'a::\text{finite ell2}} \Rightarrow_{CL} 'b::\text{finite ell2}} \Rightarrow 'c::\text{finite ell2}} \Rightarrow_{CL} 'd::\text{finite ell2}} \Rightarrow 'e::\text{complex-normed-vector}$

assumes *cilinear F2*

shows *tensor-lift-clinear: clinear (tensor-lift F2)*

and *tensor-lift-correct: $\langle (\lambda a \ b. \text{tensor-lift } F2 \ (a \otimes_o \ b)) = F2 \rangle$*

proof –

define *F2'* *t4* φ **where**

$\langle F2' = \text{tensor-lift } F2 \rangle$ **and**

$\langle t4 = (\lambda(i,j,k,l). \text{tensor-op} (\text{butterfly} (\text{ket } i) (\text{ket } j)) (\text{butterfly} (\text{ket } k) (\text{ket } l))) \rangle$ **and**

$\langle \varphi \ m = (\text{let } (i,j,k,l) = \text{inv } t4 \ m \ \text{in } F2 (\text{butterfly} (\text{ket } i) (\text{ket } j)) (\text{butterfly} (\text{ket } k) (\text{ket } l))) \rangle$

for *m*

have *t4inj: x = y if t4 x = t4 y for x y*

proof (*rule ccontr*)

obtain *i j k l* **where** *x: x = (i,j,k,l)* **by** (*meson prod-cases4*)

obtain *i' j' k' l'* **where** *y: y = (i',j',k',l')* **by** (*meson prod-cases4*)

have *1: bra (i,k) *_V t4 x *_V ket (j,l) = 1*

by (*auto simp: t4-def x tensor-op-ell2 butterfly-def cinner-ket simp flip: tensor-ell2-ket*)

assume $\langle x \neq y \rangle$

then have *2: bra (i,k) *_V t4 y *_V ket (j,l) = 0*

by (*auto simp: t4-def x y tensor-op-ell2 butterfly-def cinner-ket simp flip: tensor-ell2-ket*)

from *1 2* **that**

show *False*

by *auto*

qed

have $\langle \varphi (\text{tensor-op} (\text{butterfly} (\text{ket } i) (\text{ket } j)) (\text{butterfly} (\text{ket } k) (\text{ket } l))) = F2 (\text{butterfly} (\text{ket } i) (\text{ket } j)) (\text{butterfly} (\text{ket } k) (\text{ket } l)) \rangle$ **for** *i j k l*

apply (*subst asm-rl[$\langle \text{tensor-op} (\text{butterfly} (\text{ket } i) (\text{ket } j)) (\text{butterfly} (\text{ket } k) (\text{ket } l)) = t4 (i,j,k,l) \rangle$]*)

subgoal **by** (*simp add: t4-def*)

```

subgoal by (auto simp add: injI t4inj inv-f-f  $\varphi$ -def)
done

have *:  $\langle \text{range } t_4 = \{ \text{tensor-op } (\text{butterfly } (\text{ket } i) (\text{ket } j)) (\text{butterfly } (\text{ket } k) (\text{ket } l)) \mid i \ j \ k \ l. \text{True} \} \rangle$ 
by (force simp: case-prod-beta t4-def image-iff)

have cblinfun-extension-exists (range t4)  $\varphi$ 
by (rule cblinfun-extension-exists-finite-dim)
      (use * cindependent-tensor-op-butter cspan-tensor-op-butter in auto)

then obtain  $G$  where  $G$ :  $\langle G *_{\mathcal{V}} (t_4 (i,j,k,l)) = F_2 (\text{butterfly } (\text{ket } i) (\text{ket } j)) (\text{butterfly } (\text{ket } k) (\text{ket } l)) \rangle$  for  $i \ j \ k \ l$ 
unfolding cblinfun-extension-exists-def
by (metis (no-types, lifting) t4inj  $\varphi$ -def f-inv-into-f rangeI split-conv)

have *:  $\langle G *_{\mathcal{V}} \text{tensor-op } (\text{butterfly } (\text{ket } i) (\text{ket } j)) (\text{butterfly } (\text{ket } k) (\text{ket } l)) = F_2 (\text{butterfly } (\text{ket } i) (\text{ket } j)) (\text{butterfly } (\text{ket } k) (\text{ket } l)) \rangle$  for  $i \ j \ k \ l$ 
using  $G$  by (auto simp: t4-def)
have *:  $\langle G *_{\mathcal{V}} \text{tensor-op } a (\text{butterfly } (\text{ket } k) (\text{ket } l)) = F_2 a (\text{butterfly } (\text{ket } k) (\text{ket } l)) \rangle$  for  $a \ k \ l$ 
apply (rule complex-vector.linear-eq-on-span[where  $g = \langle \lambda a. F_2 a \rightarrow$  and  $B = \langle \{ \text{butterfly } (\text{ket } k) (\text{ket } l) \mid k \ l. \text{True} \} \rangle$ ])
unfolding cspan-butterfly-ket
using * apply (auto intro!: clinear-compose[unfolded o-def, where  $f = \langle \lambda a. \text{tensor-op } a \rightarrow$  and  $g = \langle (*_{\mathcal{V}}) G \rangle$ ])
apply (metis cbilinear-def tensor-op-cbilinear)
using assms unfolding cbilinear-def by blast
have  $G$ -F2:  $\langle G *_{\mathcal{V}} \text{tensor-op } a \ b = F_2 a \ b \rangle$  for  $a \ b$ 
apply (rule complex-vector.linear-eq-on-span[where  $g = \langle F_2 a \rangle$  and  $B = \langle \{ \text{butterfly } (\text{ket } k) (\text{ket } l) \mid k \ l. \text{True} \} \rangle$ ])
unfolding cspan-butterfly-ket
using * apply (auto simp: cblinfun.add-right clinearI
      intro!: clinear-compose[unfolded o-def, where  $f = \langle \text{tensor-op } a \rangle$  and  $g = \langle (*_{\mathcal{V}}) G \rangle$ ])
apply (meson cbilinear-def tensor-op-cbilinear)
using assms unfolding cbilinear-def by blast

have  $\langle \text{clinear } F_2' \wedge (\forall a \ b. F_2' (\text{tensor-op } a \ b) = F_2 a \ b) \rangle$ 
unfolding F2'-def tensor-lift-def
apply (rule someI[where  $x = \langle (*_{\mathcal{V}}) G \rangle$  and  $P = \langle \lambda G. \text{clinear } G \wedge (\forall a \ b. G (\text{tensor-op } a \ b) = F_2 a \ b) \rangle$ ])
using  $G$ -F2 by (simp add: cblinfun.add-right clinearI)

then show  $\langle \text{clinear } F_2' \rangle$  and  $\langle (\lambda a \ b. \text{tensor-lift } F_2 (\text{tensor-op } a \ b)) = F_2 \rangle$ 
unfolding F2'-def by auto
qed

```


lemma *tensor-op-nonzero*:

fixes $a :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$ **and** $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$
assumes $\langle a \neq 0 \rangle$ **and** $\langle b \neq 0 \rangle$
shows $\langle a \otimes_o b \neq 0 \rangle$

proof –

from $\langle a \neq 0 \rangle$ **obtain** i **where** $i: \langle a *_V \text{ ket } i \neq 0 \rangle$
by (*metis cblinfun.zero-left equal-ket*)
from $\langle b \neq 0 \rangle$ **obtain** j **where** $j: \langle b *_V \text{ ket } j \neq 0 \rangle$
by (*metis cblinfun.zero-left equal-ket*)
from $i j$ **have** $ijneq0: \langle (a *_V \text{ ket } i) \otimes_s (b *_V \text{ ket } j) \neq 0 \rangle$
by (*simp add: tensor-ell2-nonzero*)
have $\langle (a *_V \text{ ket } i) \otimes_s (b *_V \text{ ket } j) = (a \otimes_o b) *_V \text{ ket } (i,j) \rangle$
by (*simp add: tensor-op-ket*)
with $ijneq0$ **show** $\langle a \otimes_o b \neq 0 \rangle$
by *force*

qed

lemma *inj-tensor-ell2-left*: $\langle \text{inj } (\lambda a::'a \text{ ell2}. a \otimes_s b) \rangle$ **if** $\langle b \neq 0 \rangle$ **for** $b :: \langle 'b \text{ ell2} \rangle$

proof (*rule injI, rule ccontr*)

fix $x y :: \langle 'a \text{ ell2} \rangle$
assume $eq: \langle x \otimes_s b = y \otimes_s b \rangle$
assume $neq: \langle x \neq y \rangle$
define a **where** $\langle a = x - y \rangle$
from neq $a\text{-def}$ **have** $neq0: \langle a \neq 0 \rangle$
by *auto*
with $\langle b \neq 0 \rangle$ **have** $\langle a \otimes_s b \neq 0 \rangle$
by (*simp add: tensor-ell2-nonzero*)
then **have** $\langle x \otimes_s b \neq y \otimes_s b \rangle$
unfolding $a\text{-def}$
by (*metis add-cancel-left-left diff-add-cancel tensor-ell2-add1*)
with eq **show** *False*
by *auto*

qed

lemma *inj-tensor-ell2-right*: $\langle \text{inj } (\lambda b::'b \text{ ell2}. a \otimes_s b) \rangle$ **if** $\langle a \neq 0 \rangle$ **for** $a :: \langle 'a \text{ ell2} \rangle$

proof (*rule injI, rule ccontr*)

fix $x y :: \langle 'b \text{ ell2} \rangle$
assume $eq: \langle a \otimes_s x = a \otimes_s y \rangle$
assume $neq: \langle x \neq y \rangle$
define b **where** $\langle b = x - y \rangle$
from neq $b\text{-def}$ **have** $neq0: \langle b \neq 0 \rangle$
by *auto*
with $\langle a \neq 0 \rangle$ **have** $\langle a \otimes_s b \neq 0 \rangle$
by (*simp add: tensor-ell2-nonzero*)
then **have** $\langle a \otimes_s x \neq a \otimes_s y \rangle$
unfolding $b\text{-def}$
by (*metis add-cancel-left-left diff-add-cancel tensor-ell2-add2*)
with eq **show** *False*
by *auto*

qed

lemma inj-tensor-left: $\langle \text{inj } (\lambda a :: 'a \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2}. a \otimes_o b) \rangle$ **if** $\langle b \neq 0 \rangle$ **for** $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

proof (rule injI, rule ccontr)

fix $x y :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$

assume $eq: \langle x \otimes_o b = y \otimes_o b \rangle$

assume $neg: \langle x \neq y \rangle$

define a **where** $\langle a = x - y \rangle$

from neg a -def **have** $neg0: \langle a \neq 0 \rangle$

by auto

with $\langle b \neq 0 \rangle$ **have** $\langle a \otimes_o b \neq 0 \rangle$

by (simp add: tensor-op-nonzero)

then **have** $\langle x \otimes_o b \neq y \otimes_o b \rangle$

unfolding a -def

by (metis add-cancel-left-left diff-add-cancel tensor-op-left-add)

with eq **show** False

by auto

qed

lemma inj-tensor-right: $\langle \text{inj } (\lambda b :: 'b \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2}. a \otimes_o b) \rangle$ **if** $\langle a \neq 0 \rangle$ **for** $a :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

proof (rule injI, rule ccontr)

fix $x y :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$

assume $eq: \langle a \otimes_o x = a \otimes_o y \rangle$

assume $neg: \langle x \neq y \rangle$

define b **where** $\langle b = x - y \rangle$

from neg b -def **have** $neg0: \langle b \neq 0 \rangle$

by auto

with $\langle a \neq 0 \rangle$ **have** $\langle a \otimes_o b \neq 0 \rangle$

by (simp add: tensor-op-nonzero)

then **have** $\langle a \otimes_o x \neq a \otimes_o y \rangle$

unfolding b -def

by (metis add-cancel-left-left diff-add-cancel tensor-op-right-add)

with eq **show** False

by auto

qed

lemma tensor-ell2-almost-injective:

assumes $\langle \text{tensor-ell2 } a \ b = \text{tensor-ell2 } c \ d \rangle$

assumes $\langle a \neq 0 \rangle$

shows $\langle \exists \gamma. b = \gamma *_C d \rangle$

proof –

from $\langle a \neq 0 \rangle$ **obtain** i **where** $i: \langle \text{cinner } (\text{ket } i) \ a \neq 0 \rangle$

by (metis cinner-eq-zero-iff cinner-ket-left ell2-pointwise-ortho)

have $\langle \text{cinner } (\text{ket } i \otimes_s \text{ket } j) \ (a \otimes_s b) = \text{cinner } (\text{ket } i \otimes_s \text{ket } j) \ (c \otimes_s d) \rangle$ **for** j

using $assms$ **by** simp

then **have** $eq2: \langle (\text{cinner } (\text{ket } i) \ a) * (\text{cinner } (\text{ket } j) \ b) = (\text{cinner } (\text{ket } i) \ c) * (\text{cinner } (\text{ket } j) \ d) \rangle$ **for** j

```

    by (metis tensor-ell2-inner-prod)
  then obtain  $\gamma$  where  $\langle \text{cinner } (ket\ i)\ c = \gamma * \text{cinner } (ket\ i)\ a \rangle$ 
    by (metis i eq-divide-eq)
  with eq2 have  $\langle \text{cinner } (ket\ i)\ a * (\text{cinner } (ket\ j)\ b) = (\text{cinner } (ket\ i)\ a) * (\gamma * \text{cinner } (ket\ j)\ d) \rangle$  for  $j$ 
    by simp
  then have  $\langle \text{cinner } (ket\ j)\ b = \text{cinner } (ket\ j)\ (\gamma *_{\mathcal{C}} d) \rangle$  for  $j$ 
    using  $i$  by force
  then have  $\langle b = \gamma *_{\mathcal{C}} d \rangle$ 
    by (simp add: cinner-ket-eqI)
  then show ?thesis
    by auto
qed

```

lemma tensor-op-almost-injective:

```

  fixes  $a\ c :: \langle 'a\ ell2 \Rightarrow_{\mathcal{C}L} 'b\ ell2 \rangle$ 
    and  $b\ d :: \langle 'c\ ell2 \Rightarrow_{\mathcal{C}L} 'd\ ell2 \rangle$ 
  assumes  $\langle \text{tensor-op } a\ b = \text{tensor-op } c\ d \rangle$ 
  assumes  $\langle a \neq 0 \rangle$ 
  shows  $\langle \exists \gamma. b = \gamma *_{\mathcal{C}} d \rangle$ 
proof (cases  $\langle d = 0 \rangle$ )
  case False
  from  $\langle a \neq 0 \rangle$  obtain  $\psi$  where  $\psi: \langle a *_{\mathcal{V}} \psi \neq 0 \rangle$ 
    by (metis cblinfun.zero-left cblinfun-eqI)
  have  $\langle (a \otimes_o b) (\psi \otimes_s \varphi) = (c \otimes_o d) (\psi \otimes_s \varphi) \rangle$  for  $\varphi$ 
    using assms by simp
  then have eq2:  $\langle (a \psi) \otimes_s (b \varphi) = (c \psi) \otimes_s (d \varphi) \rangle$  for  $\varphi$ 
    by (simp add: tensor-op-ell2)
  then have eq2':  $\langle (d \varphi) \otimes_s (c \psi) = (b \varphi) \otimes_s (a \psi) \rangle$  for  $\varphi$ 
    by (metis swap-ell2-tensor)
  from False obtain  $\varphi 0$  where  $\varphi 0: \langle d \varphi 0 \neq 0 \rangle$ 
    by (metis cblinfun.zero-left cblinfun-eqI)
  obtain  $\gamma$  where  $\langle c \psi = \gamma *_{\mathcal{C}} a \psi \rangle$ 
    apply atomize-elim
    using eq2'  $\varphi 0$  by (rule tensor-ell2-almost-injective)
  with eq2 have  $\langle (a \psi) \otimes_s (b \varphi) = (a \psi) \otimes_s (\gamma *_{\mathcal{C}} d \varphi) \rangle$  for  $\varphi$ 
    by (simp add: tensor-ell2-scaleC1 tensor-ell2-scaleC2)
  then have  $\langle b \varphi = \gamma *_{\mathcal{C}} d \varphi \rangle$  for  $\varphi$ 
    by (smt (verit, best)  $\psi$  complex-vector.scale-cancel-right tensor-ell2-almost-injective tensor-ell2-nonzero tensor-ell2-scaleC2)
  then have  $\langle b = \gamma *_{\mathcal{C}} d \rangle$ 
    by (simp add: cblinfun-eqI)
  then show ?thesis
    by auto
next
  case True
  then have  $\langle c \otimes_o d = 0 \rangle$ 
    by (metis add-cancel-right-left tensor-op-right-add)

```

```

then have ⟨a ⊗o b = 0⟩
  using assms(1) by presburger
with ⟨a ≠ 0⟩ have ⟨b = 0⟩
  by (meson tensor-op-nonzero)
then show ?thesis
  by auto
qed

lemma clinear-tensor-left[simp]: ⟨clinear (λa. a ⊗o b :: - ell2 ⇒CL - ell2)⟩
  apply (rule clinearI)
  apply (rule tensor-op-left-add)
  by (rule tensor-op-scaleC-left)

lemma clinear-tensor-right[simp]: ⟨clinear (λb. a ⊗o b :: - ell2 ⇒CL - ell2)⟩
  apply (rule clinearI)
  apply (rule tensor-op-right-add)
  by (rule tensor-op-scaleC-right)

lemma tensor-op-0-left[simp]: ⟨tensor-op 0 x = (0 :: ('a*'b) ell2 ⇒CL ('c*'d) ell2)⟩
  apply (rule equal-ket)
  by (auto simp flip: tensor-ell2-ket simp: tensor-op-ell2)

lemma tensor-op-0-right[simp]: ⟨tensor-op x 0 = (0 :: ('a*'b) ell2 ⇒CL ('c*'d) ell2)⟩
  apply (rule equal-ket)
  by (auto simp flip: tensor-ell2-ket simp: tensor-op-ell2)

lemma bij-tensor-ell2-one-dim-left:
  assumes ⟨ψ ≠ 0⟩
  shows ⟨bij (λx::'b ell2. (ψ :: 'a::CARD-1 ell2) ⊗s x)⟩
proof (rule bijI)
  show ⟨inj (λx::'b ell2. (ψ :: 'a::CARD-1 ell2) ⊗s x)⟩
    using assms by (rule inj-tensor-ell2-right)
  have ⟨∃x. ψ ⊗s x = φ⟩ for φ :: ⟨('a*'b) ell2⟩
proof (use assms in transfer)
  fix ψ :: ⟨'a ⇒ complex⟩ and φ :: ⟨'a*'b ⇒ complex⟩
  assume ⟨has-ell2-norm φ⟩ and ⟨ψ ≠ (λ-. 0)⟩
  define c where ⟨c = ψ undefined⟩
  then have ⟨ψ a = c⟩ for a
    by (subst everything-the-same[of - undefined]) simp
  with ⟨ψ ≠ (λ-. 0)⟩ have ⟨c ≠ 0⟩
    by auto

define x where ⟨x j = φ (undefined, j) / c⟩ for j
have ⟨(λ(i, j). ψ i * x j) = φ⟩
proof (rule ext, safe)
  fix a :: 'a and b :: 'b
  show ψ a * x b = φ (a, b)
    using ⟨c ≠ 0⟩ by (simp add: c-def x-def everything-the-same[of a undefined])
qed

```

```

moreover have ⟨has-ell2-norm x⟩
proof –
  have ⟨(λ(i,j). (φ (i,j))2) abs-summable-on UNIV⟩
    using ⟨has-ell2-norm φ⟩ has-ell2-norm-def by auto
  then have ⟨(λ(i,j). (φ (i,j))2) abs-summable-on Pair undefined ‘ UNIV⟩
    using summable-on-subset-banach by blast
  then have ⟨(λj. (φ (undefined,j))2) abs-summable-on UNIV⟩
    by (subst (asm) summable-on-reindex) (auto simp: o-def inj-def)
  then have ⟨(λj. (φ (undefined,j) / c)2) abs-summable-on UNIV⟩
    by (simp add: divide-inverse power-mult-distrib norm-mult summable-on-cmult-left)
  then have ⟨(λj. (x j)2) abs-summable-on UNIV⟩
    by (simp add: x-def)
  then show ?thesis
    using has-ell2-norm-def by blast
qed
ultimately show ⟨∃ x ∈ Collect has-ell2-norm. (λ(i, j). ψ i * x j) = φ⟩
  by (intro bexI[where x=x]) auto
qed

then show ⟨surj (λx::'b ell2. (ψ :: 'a::CARD-1 ell2) ⊗s x)⟩
  by (metis surj-def)
qed

lemma bij-tensor-op-one-dim-left:
  fixes a :: ⟨'a::{CARD-1,enum} ell2 ⇒CL 'b::{CARD-1,enum} ell2⟩
  assumes ⟨a ≠ 0⟩
  shows ⟨bij (λx::'c ell2 ⇒CL 'd ell2. a ⊗o x)⟩
proof –
  have [simp]: ⟨bij (Pair (undefined::'a))⟩
    by (rule o-bij[of snd]) auto
  have [simp]: ⟨bij (Pair (undefined::'b))⟩
    by (rule o-bij[of snd]) auto
  define t where ⟨t x = a ⊗o x⟩ for x :: ⟨'c ell2 ⇒CL 'd ell2⟩
  define u :: ⟨'c ell2 ⇒CL ('a × 'c) ell2⟩ where ⟨u = classical-operator (Some o Pair undefined)⟩
  define v :: ⟨'d ell2 ⇒CL ('b × 'd) ell2⟩ where ⟨v = classical-operator (Some o Pair undefined)⟩
  have [simp]: ⟨unitary u⟩ ⟨unitary v⟩
    by (simp-all add: u-def v-def)
  have u-ket[simp]: ⟨u *V ket x = ket (undefined, x)⟩ for x
    by (simp add: u-def classical-operator-ket classical-operator-exists-inj inj-def)
  have uadj-ket[simp]: ⟨u* *V ket (z, x) = ket x⟩ for x z
    by (subst everything-the-same[of - undefined])
      (metis (no-types, opaque-lifting) u-ket cinner-adj-right cinner-ket-eqI cinner-ket-same
        orthogonal-ket prod.inject)
  have v-ket[simp]: ⟨v *V ket x = ket (undefined, x)⟩ for x
    by (simp add: v-def classical-operator-ket classical-operator-exists-inj inj-def)
  have [simp]: ⟨v *V x = ket undefined ⊗s x⟩ for x
    by (rule fun-cong[where x=x], rule bounded-clinear-equal-ket)
      (auto simp add: bounded-clinear-tensor-ell21 cblinfun.bounded-clinear-right tensor-ell2-ket)
  define a' :: complex where ⟨a' = one-dim-iso a⟩

```

```

from assms have ⟨ $a' \neq 0$ ⟩
  using a'-def one-dim-iso-of-zero' by auto
have a-a': ⟨ $a = \text{of-complex } a'$ ⟩
  by (simp add: a'-def)
have ⟨ $t \ x \ *_V \ \text{ket } (i,j) = (a' \ *_C \ v \ o_{CL} \ x \ o_{CL} \ u^*) \ *_V \ \text{ket } (i,j)$ ⟩ for  $x \ i \ j$ 
  apply (simp add: t-def)
  apply (simp add: ket-CARD-1-is-1 tensor-op-ell2 flip: tensor-ell2-ket)
  by (metis a'-def one-cblinfun-apply-one one-dim-scaleC-1 scaleC-cblinfun.rep-eq tensor-ell2-scaleC1)

then have t: ⟨ $t \ x = (a' \ *_C \ v \ o_{CL} \ x \ o_{CL} \ u^*)$ ⟩ for  $x$ 
  by (intro equal-ket) auto
define s where ⟨ $s \ y = (\text{inverse } a' \ *_C \ (v)^* \ o_{CL} \ y \ o_{CL} \ u)$ ⟩ for  $y$ 
have ⟨ $s \ (t \ x) = (a' \ *_C \ \text{inverse } a') \ *_C \ (((v)^* \ o_{CL} \ v) \ o_{CL} \ x \ o_{CL} \ (u^* \ o_{CL} \ u))$ ⟩ for  $x$ 
  apply (simp add: s-def t cblinfun-compose-assoc)
  by (simp flip: cblinfun-compose-assoc)?
also have ⟨ $\dots \ x = x$ ⟩ for  $x$ 
  using ⟨ $a' \neq 0$ ⟩ by simp
finally have ⟨ $s \ o \ t = \text{id}$ ⟩
  by auto
have ⟨ $t \ (s \ y) = (a' \ *_C \ \text{inverse } a') \ *_C \ ((v \ o_{CL} \ (v)^*) \ o_{CL} \ y \ o_{CL} \ (u \ o_{CL} \ u^*))$ ⟩ for  $y$ 
  apply (simp add: s-def t cblinfun-compose-assoc)
  by (simp flip: cblinfun-compose-assoc)?
also have ⟨ $\dots \ y = y$ ⟩ for  $y$ 
  using ⟨ $a' \neq 0$ ⟩ by simp
finally have ⟨ $t \ o \ s = \text{id}$ ⟩
  by auto
from ⟨ $s \ o \ t = \text{id}$ ⟩ ⟨ $t \ o \ s = \text{id}$ ⟩
show ⟨bij t⟩
  using o-bij by blast
qed

lemma bij-tensor-op-one-dim-right:
  assumes ⟨ $b \neq 0$ ⟩
  shows ⟨bij ( $\lambda x::'c \ \text{ell2} \Rightarrow_{CL} 'd \ \text{ell2} . x \ \otimes_o \ (b :: 'a::\{\text{CARD-1,enum}\} \ \text{ell2} \Rightarrow_{CL} 'b::\{\text{CARD-1,enum}\} \ \text{ell2})$ )⟩
    (is ⟨bij ?f⟩)
proof -
  let ?sf = ⟨ $(\lambda x . \ \text{swap-ell2} \ o_{CL} \ (?f \ x) \ o_{CL} \ \text{swap-ell2})$ ⟩
  let ?s = ⟨ $(\lambda x . \ \text{swap-ell2} \ o_{CL} \ x \ o_{CL} \ \text{swap-ell2})$ ⟩
  let ?g = ⟨ $(\lambda x::'c \ \text{ell2} \Rightarrow_{CL} 'd \ \text{ell2} . (b :: 'a::\{\text{CARD-1,enum}\} \ \text{ell2} \Rightarrow_{CL} 'b::\{\text{CARD-1,enum}\} \ \text{ell2}) \ \otimes_o \ x)$ ⟩
  have ⟨?sf = ?g⟩
    by (auto intro!: ext tensor-ell2-extensionality simp add: swap-ell2-tensor tensor-op-ell2)
  have ⟨bij ?g⟩
    using assms by (rule bij-tensor-op-one-dim-left)
  have ⟨?s o ?sf = ?f⟩
    apply (auto intro!: ext simp: cblinfun-assoc-left)
    by (auto simp: cblinfun-assoc-right)?
  also have ⟨bij ?s⟩

```

```

apply (rule o-bij[where  $g = \langle \lambda x. \text{swap-ell2 } o_{CL} x o_{CL} \text{swap-ell2} \rangle$ ])
apply (auto intro!: ext simp: cblinfun-assoc-left)
by (auto simp: cblinfun-assoc-right)?
show  $\langle \text{bij } ?f \rangle$ 
apply (subst  $\langle ?s o ?sf = ?f \rangle$ [symmetric], subst  $\langle ?sf = ?g \rangle$ )
using  $\langle \text{bij } ?g \rangle \langle \text{bij } ?s \rangle$  by (rule bij-comp)
qed

lemma overlapping-tensor:
fixes  $a23 :: \langle ('a2*'a3) \text{ell2} \Rightarrow_{CL} ('b2*'b3) \text{ell2} \rangle$ 
and  $b12 :: \langle ('a1*'a2) \text{ell2} \Rightarrow_{CL} ('b1*'b2) \text{ell2} \rangle$ 
assumes eq:  $\langle \text{butterfly } \psi \psi' \otimes_o a23 = \text{assoc-ell2 } o_{CL} (b12 \otimes_o \text{butterfly } \varphi \varphi') o_{CL} \text{assoc-ell2} \rangle$ 
assumes  $\langle \psi \neq 0 \rangle \langle \psi' \neq 0 \rangle \langle \varphi \neq 0 \rangle \langle \varphi' \neq 0 \rangle$ 
shows  $\langle \exists c. \text{butterfly } \psi \psi' \otimes_o a23 = \text{butterfly } \psi \psi' \otimes_o c \otimes_o \text{butterfly } \varphi \varphi' \rangle$ 
proof –
let  $?id1 = \langle \text{id-cblinfun} :: \text{unit ell2} \Rightarrow_{CL} \text{unit ell2} \rangle$ 
note id-cblinfun-eq-1[simp del]
define  $d$  where  $\langle d = \text{butterfly } \psi \psi' \otimes_o a23 \rangle$ 

define  $\psi_n \psi_n' a23_n$  where  $\langle \psi_n = \psi /_C \text{norm } \psi \rangle$  and  $\langle \psi_n' = \psi' /_C \text{norm } \psi' \rangle$  and  $\langle a23_n = \text{norm } \psi *_C \text{norm } \psi' *_C a23 \rangle$ 
have [simp]:  $\langle \text{norm } \psi_n = 1 \rangle \langle \text{norm } \psi_n' = 1 \rangle$ 
using  $\langle \psi \neq 0 \rangle \langle \psi' \neq 0 \rangle$  by (auto simp:  $\psi_n$ -def  $\psi_n'$ -def norm-inverse)
have  $n1: \langle \text{butterfly } \psi_n \psi_n' \otimes_o a23_n = \text{butterfly } \psi \psi' \otimes_o a23 \rangle$ 
by (auto simp:  $\psi_n$ -def  $\psi_n'$ -def  $a23_n$ -def tensor-op-scaleC-left tensor-op-scaleC-right field-simps)

define  $\varphi_n \varphi_n' b12_n$  where  $\langle \varphi_n = \varphi /_C \text{norm } \varphi \rangle$  and  $\langle \varphi_n' = \varphi' /_C \text{norm } \varphi' \rangle$  and  $\langle b12_n = \text{norm } \varphi *_C \text{norm } \varphi' *_C b12 \rangle$ 
have [simp]:  $\langle \text{norm } \varphi_n = 1 \rangle \langle \text{norm } \varphi_n' = 1 \rangle$ 
using  $\langle \varphi \neq 0 \rangle \langle \varphi' \neq 0 \rangle$  by (auto simp:  $\varphi_n$ -def  $\varphi_n'$ -def norm-inverse)
have  $n2: \langle b12_n \otimes_o \text{butterfly } \varphi_n \varphi_n' = b12 \otimes_o \text{butterfly } \varphi \varphi' \rangle$ 
by (auto simp:  $\varphi_n$ -def  $\varphi_n'$ -def  $b12_n$ -def tensor-op-scaleC-left tensor-op-scaleC-right field-simps)

define  $c' :: \langle (\text{unit}*'a2*\text{unit}) \text{ell2} \Rightarrow_{CL} (\text{unit}*'b2*\text{unit}) \text{ell2} \rangle$ 
where  $\langle c' = (\text{vector-to-cblinfun } \psi_n \otimes_o \text{id-cblinfun} \otimes_o \text{vector-to-cblinfun } \varphi_n) *_C o_{CL} d$ 
 $o_{CL} (\text{vector-to-cblinfun } \psi_n' \otimes_o \text{id-cblinfun} \otimes_o \text{vector-to-cblinfun } \varphi_n') \rangle$ 

define  $c'' :: \langle 'a2 \text{ell2} \Rightarrow_{CL} 'b2 \text{ell2} \rangle$ 
where  $\langle c'' = \text{inv } (\lambda c''. \text{id-cblinfun} \otimes_o c'' \otimes_o \text{id-cblinfun}) c' \rangle$ 

have *:  $\langle \text{bij } (\lambda c''::'a2 \text{ell2} \Rightarrow_{CL} 'b2 \text{ell2}. ?id1 \otimes_o c'' \otimes_o ?id1) \rangle$ 
by (subst asm-rl[of  $\langle - = (\lambda x. \text{id-cblinfun} \otimes_o x) o (\lambda c''. c'' \otimes_o \text{id-cblinfun}) \rangle$ ])
(auto intro!: bij-comp bij-tensor-op-one-dim-left bij-tensor-op-one-dim-right)

have  $c'-c'': \langle c' = ?id1 \otimes_o c'' \otimes_o ?id1 \rangle$ 
unfolding  $c''$ -def
by (rule surj-f-inv-f[where  $y = c'$ , symmetric]) (use * in  $\langle \text{rule bij-is-surj} \rangle$ )

define  $c :: \langle 'a2 \text{ell2} \Rightarrow_{CL} 'b2 \text{ell2} \rangle$ 

```

where $\langle c = c'' /_C \text{norm } \psi /_C \text{norm } \psi' /_C \text{norm } \varphi /_C \text{norm } \varphi' \rangle$

have $\text{aux}: \langle \text{assoc-ell2}^* \circ_{CL} (\text{assoc-ell2} \circ_{CL} x \circ_{CL} \text{assoc-ell2}^*) \circ_{CL} \text{assoc-ell2} = x \rangle$ **for** x
apply (*simp add: cblinfun-assoc-left*)
by (*simp add: cblinfun-assoc-right*)?

have $\text{aux2}: \langle (\text{assoc-ell2} \circ_{CL} ((x \otimes_o y) \otimes_o z) \circ_{CL} \text{assoc-ell2}^*) = x \otimes_o (y \otimes_o z) \rangle$ **for** $x y z$
by (*intro equal-ket*) (*auto simp flip: tensor-ell2-ket simp: assoc-ell2'-tensor assoc-ell2-tensor tensor-op-ell2*)

have $\langle d = (\text{butterfly } \psi_n \psi_n \otimes_o \text{id-cblinfun}) \circ_{CL} d \circ_{CL} (\text{butterfly } \psi_n' \psi_n' \otimes_o \text{id-cblinfun}) \rangle$
by (*auto simp: d-def n1[symmetric] comp-tensor-op cnorm-eq-1[THEN iffD1]*)

also have $\langle \dots = (\text{butterfly } \psi_n \psi_n \otimes_o \text{id-cblinfun}) \circ_{CL} \text{assoc-ell2} \circ_{CL} (b12_n \otimes_o \text{butterfly } \varphi_n \varphi_n') \rangle$
 $\circ_{CL} \text{assoc-ell2}^* \circ_{CL} (\text{butterfly } \psi_n' \psi_n' \otimes_o \text{id-cblinfun})$
by (*auto simp: d-def eq n2 cblinfun-assoc-left*)

also have $\langle \dots = (\text{butterfly } \psi_n \psi_n \otimes_o \text{id-cblinfun}) \circ_{CL} \text{assoc-ell2} \circ_{CL} ((\text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n \varphi_n') \circ_{CL} (b12_n \otimes_o \text{butterfly } \varphi_n \varphi_n')) \circ_{CL} (\text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n' \varphi_n') \rangle$
 $\circ_{CL} \text{assoc-ell2}^* \circ_{CL} (\text{butterfly } \psi_n' \psi_n' \otimes_o \text{id-cblinfun})$
by (*auto simp: comp-tensor-op cnorm-eq-1[THEN iffD1]*)

also have $\langle \dots = (\text{butterfly } \psi_n \psi_n \otimes_o \text{id-cblinfun}) \circ_{CL} \text{assoc-ell2} \circ_{CL} ((\text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n \varphi_n') \circ_{CL} (\text{assoc-ell2}^* \circ_{CL} d \circ_{CL} \text{assoc-ell2})) \circ_{CL} (\text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n' \varphi_n') \rangle$
 $\circ_{CL} \text{assoc-ell2}^* \circ_{CL} (\text{butterfly } \psi_n' \psi_n' \otimes_o \text{id-cblinfun})$
by (*auto simp: d-def n2 eq aux*)

also have $\langle \dots = ((\text{butterfly } \psi_n \psi_n \otimes_o \text{id-cblinfun}) \circ_{CL} (\text{assoc-ell2} \circ_{CL} (\text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n \varphi_n') \circ_{CL} \text{assoc-ell2}^*)) \circ_{CL} d \circ_{CL} ((\text{assoc-ell2} \circ_{CL} (\text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n' \varphi_n') \circ_{CL} \text{assoc-ell2}^*)) \circ_{CL} (\text{butterfly } \psi_n' \psi_n' \otimes_o \text{id-cblinfun}) \rangle$
by (*auto simp: sandwich-def cblinfun-assoc-left*)

also have $\langle \dots = (\text{butterfly } \psi_n \psi_n \otimes_o \text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n \varphi_n') \circ_{CL} d \circ_{CL} (\text{butterfly } \psi_n' \psi_n' \otimes_o \text{id-cblinfun} \otimes_o \text{butterfly } \varphi_n' \varphi_n') \rangle$
apply (*simp only: tensor-id[symmetric] comp-tensor-op aux2*)
by (*simp add: cnorm-eq-1[THEN iffD1]*)

also have $\langle \dots = (\text{vector-to-cblinfun } \psi_n \otimes_o \text{id-cblinfun} \otimes_o \text{vector-to-cblinfun } \varphi_n) \circ_{CL} c' \circ_{CL} (\text{vector-to-cblinfun } \psi_n' \otimes_o \text{id-cblinfun} \otimes_o \text{vector-to-cblinfun } \varphi_n')^* \rangle$
apply (*simp add: c'-def butterfly-def-one-dim[where 'c=unit ell2] cblinfun-assoc-left comp-tensor-op tensor-op-adjoint cnorm-eq-1[THEN iffD1]*)
by (*simp add: cblinfun-assoc-right comp-tensor-op*)

also have $\langle \dots = \text{butterfly } \psi_n \psi_n' \otimes_o c'' \otimes_o \text{butterfly } \varphi_n \varphi_n' \rangle$
by (*simp add: c'-c'' comp-tensor-op tensor-op-adjoint butterfly-def-one-dim[symmetric]*)

also have $\langle \dots = \text{butterfly } \psi \psi' \otimes_o c \otimes_o \text{butterfly } \varphi \varphi' \rangle$
by (*simp add: ψ_n -def ψ_n' -def φ_n -def φ_n' -def c-def tensor-op-scaleC-left tensor-op-scaleC-right*)

finally have $d\text{-c}: \langle d = \text{butterfly } \psi \psi' \otimes_o c \otimes_o \text{butterfly } \varphi \varphi' \rangle$
by –

then show *?thesis*
by (*auto simp: d-def*)

qed

lemma *tensor-op-pos*: $\langle a \otimes_o b \geq 0 \rangle$ **if** [*simp*]: $\langle a \geq 0 \rangle \langle b \geq 0 \rangle$
for $a :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2} \rangle$ **and** $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \rangle$
— [8, Lemma 18]
proof —
have $\langle (\text{sqrt-op } a \otimes_o \text{sqrt-op } b) *_{o_{CL}} (\text{sqrt-op } a \otimes_o \text{sqrt-op } b) = a \otimes_o b \rangle$
by (*simp add: tensor-op-adjoint comp-tensor-op positive-selfadjointI [unfolded selfadjoint-def]*)
then show $\langle a \otimes_o b \geq 0 \rangle$
by (*metis positive-cblinfun-squareI*)
qed

lemma *abs-op-tensor*: $\langle \text{abs-op } (a \otimes_o b) = \text{abs-op } a \otimes_o \text{abs-op } b \rangle$
— [8, Lemma 18]
proof —
have $\langle (\text{abs-op } a \otimes_o \text{abs-op } b) *_{o_{CL}} (\text{abs-op } a \otimes_o \text{abs-op } b) = (a \otimes_o b) *_{o_{CL}} (a \otimes_o b) \rangle$
by (*simp add: tensor-op-adjoint comp-tensor-op abs-op-def positive-cblinfun-squareI positive-selfadjointI [unfolded selfadjoint-def]*)
then show *?thesis*
by (*metis abs-opI abs-op-pos tensor-op-pos*)
qed

lemma *trace-class-tensor*: $\langle \text{trace-class } (a \otimes_o b) \rangle$ **if** $\langle \text{trace-class } a \rangle$ **and** $\langle \text{trace-class } b \rangle$
— [8, Lemma 32]
proof —
from $\langle \text{trace-class } a \rangle$
have $a: \langle (\lambda x. \text{ket } x \cdot_C (\text{abs-op } a *_{\mathcal{V}} \text{ket } x)) \text{abs-summable-on UNIV} \rangle$
by (*auto simp add: trace-class-iff-summable [OF is-onb-ket] summable-on-reindex o-def*)
from $\langle \text{trace-class } b \rangle$
have $b: \langle (\lambda y. \text{ket } y \cdot_C (\text{abs-op } b *_{\mathcal{V}} \text{ket } y)) \text{abs-summable-on UNIV} \rangle$
by (*auto simp add: trace-class-iff-summable [OF is-onb-ket] summable-on-reindex o-def*)
from $a \ b$ **have** $\langle (\lambda(x,y). (\text{ket } x \cdot_C (\text{abs-op } a *_{\mathcal{V}} \text{ket } x)) * (\text{ket } y \cdot_C (\text{abs-op } b *_{\mathcal{V}} \text{ket } y))) \text{abs-summable-on UNIV} \times \text{UNIV} \rangle$
by (*rule abs-summable-times*)
then have $\langle (\lambda(x,y). (\text{ket } x \otimes_s \text{ket } y) \cdot_C ((\text{abs-op } a \otimes_o \text{abs-op } b) *_{\mathcal{V}} (\text{ket } x \otimes_s \text{ket } y))) \text{abs-summable-on UNIV} \times \text{UNIV} \rangle$
by (*simp add: tensor-op-ell2 case-prod-unfold flip: tensor-ell2-ket*)
then have $\langle (\lambda xy. \text{ket } xy \cdot_C ((\text{abs-op } a \otimes_o \text{abs-op } b) *_{\mathcal{V}} \text{ket } xy)) \text{abs-summable-on UNIV} \rangle$
by (*simp add: case-prod-beta tensor-ell2-ket*)
then have $\langle (\lambda xy. \text{ket } xy \cdot_C (\text{abs-op } (a \otimes_o b) *_{\mathcal{V}} \text{ket } xy)) \text{abs-summable-on UNIV} \rangle$
by (*simp add: abs-op-tensor*)
then show $\langle \text{trace-class } (a \otimes_o b) \rangle$
by (*auto simp add: trace-class-iff-summable [OF is-onb-ket] summable-on-reindex o-def*)
qed

lemma *swap-tensor-op*[*simp*]: $\langle \text{swap-ell2 } o_{CL} (a \otimes_o b) o_{CL} \text{swap-ell2} = b \otimes_o a \rangle$
by (*auto intro!: equal-ket simp add: tensor-op-ell2 simp flip: tensor-ell2-ket*)

lemma *swap-tensor-op-sandwich*[*simp*]: $\langle \text{sandwich swap-ell2 } (a \otimes_o b) = b \otimes_o a \rangle$
by (*simp add: sandwich-apply*)

```

lemma swap-ell2-commute-tensor-op:
  ⟨swap-ell2 oCL (a ⊗o b) = (b ⊗o a) oCL swap-ell2⟩
  by (auto intro!: tensor-ell2-extensionality simp: tensor-op-ell2)

lemma trace-class-tensor-op-swap: ⟨trace-class (a ⊗o b) ⟷ trace-class (b ⊗o a)⟩
proof (rule iffI)
  assume ⟨trace-class (a ⊗o b)⟩
  then have ⟨trace-class (swap-ell2 oCL (a ⊗o b) oCL swap-ell2)⟩
    using trace-class-comp-left trace-class-comp-right by blast
  then show ⟨trace-class (b ⊗o a)⟩
    by simp
next
  assume ⟨trace-class (b ⊗o a)⟩
  then have ⟨trace-class (swap-ell2 oCL (b ⊗o a) oCL swap-ell2)⟩
    using trace-class-comp-left trace-class-comp-right by blast
  then show ⟨trace-class (a ⊗o b)⟩
    by simp
qed

lemma trace-class-tensor-iff: ⟨trace-class (a ⊗o b) ⟷ (trace-class a ∧ trace-class b) ∨ a = 0
  ∨ b = 0⟩
proof (intro iffI)
  show ⟨trace-class a ∧ trace-class b ∨ a = 0 ∨ b = 0 ⟹ trace-class (a ⊗o b)⟩
    by (auto simp add: trace-class-tensor)
  show ⟨trace-class a ∧ trace-class b ∨ a = 0 ∨ b = 0⟩ if ⟨trace-class (a ⊗o b)⟩
  proof (cases ⟨a = 0 ∨ b = 0⟩)
    case True
      then show ?thesis
        by simp
    next
      case False
        then have ⟨a ≠ 0⟩ and ⟨b ≠ 0⟩
          by auto
        have *: ⟨trace-class a⟩ if ⟨trace-class (a ⊗o b)⟩ and ⟨b ≠ 0⟩ for a :: ⟨'e ell2 ⇒CL 'g ell2⟩
          and b :: ⟨'f ell2 ⇒CL 'h ell2⟩
          proof –
            from ⟨b ≠ 0⟩ have ⟨abs-op b ≠ 0⟩
              using abs-op-nondegenerate by blast
            then obtain ψ0 where ψ0: ⟨ψ0 •C (abs-op b *V ψ0) ≠ 0⟩
              by (metis cblinfun.zero-left cblinfun.cinner-eqI cinner-zero-right)
            define ψ where ⟨ψ = sgn ψ0⟩
            with ψ0 have ⟨ψ •C (abs-op b *V ψ) ≠ 0⟩ and ⟨norm ψ = 1⟩
              by (auto simp add: ψ-def norm-sgn sgn-div-norm cblinfun.scaleR-right field-simps)
            then have ⟨∃ B. {ψ} ⊆ B ∧ is-onb B⟩
              by (intro orthonormal-basis-exists) auto
            then obtain B where [simp]: ⟨is-onb B⟩ and ⟨ψ ∈ B⟩
              by auto
  
```

```

define A :: ‹'e ell2 set› where ‹A = range ket›
then have [simp]: ‹is-onb A› by simp
with ‹is-onb B› have ‹is-onb { $\alpha \otimes_s \beta \mid \alpha \in A \wedge \beta \in B$ }›
  by (simp add: tensor-ell2-is-onb)
with ‹trace-class (a  $\otimes_o$  b)›
have ‹( $\lambda \gamma. \gamma \cdot_C (abs-op (a \otimes_o b) *_V \gamma)$ ) abs-summable-on { $\alpha \otimes_s \beta \mid \alpha \in A \wedge \beta \in B$ }›
  using trace-class-iff-summable by auto
then have ‹( $\lambda \gamma. \gamma \cdot_C (abs-op (a \otimes_o b) *_V \gamma)$ ) abs-summable-on ( $\lambda \alpha. \alpha \otimes_s \psi$ ) ‹A›
  by (rule summable-on-subset-banach) (use ‹ $\psi \in B$ › in blast)
then have ‹( $\lambda \alpha. (\alpha \otimes_s \psi) \cdot_C (abs-op (a \otimes_o b) *_V (\alpha \otimes_s \psi))$ ) abs-summable-on A›
proof (subst (asm) summable-on-reindex)
  show inj-on ( $\lambda \alpha. \alpha \otimes_s \psi$ ) A
  by (metis UNIV-I ‹norm  $\psi = 1$ › inj-on-subset inj-tensor-ell2-left norm-le-zero-iff
not-one-le-zero subset-iff)
qed (simp-all add: o-def)
then have ‹( $\lambda \alpha. norm (\alpha \cdot_C (abs-op a *_V \alpha)) * norm (\psi \cdot_C (abs-op b *_V \psi))$ ) summable-on
A›
  by (simp add: abs-op-tensor tensor-op-ell2 norm-mult)
then have ‹( $\lambda \alpha. \alpha \cdot_C (abs-op a *_V \alpha)$ ) abs-summable-on A›
  by (rule summable-on-cmult-left'[THEN iffD1, rotated])
  (use ‹ $\psi \cdot_C (abs-op b *_V \psi) \neq 0$ › norm-eq-zero in ‹blast›)
then show ‹trace-class a›
  using ‹is-onb A› trace-classI by blast
qed
from *[of a b] ‹b  $\neq 0$ › ‹trace-class (a  $\otimes_o$  b)› have ‹trace-class a›
  by simp
have ‹trace-class (b  $\otimes_o$  a)›
  using that trace-class-tensor-op-swap by blast
from *[of b a] ‹a  $\neq 0$ › ‹trace-class (b  $\otimes_o$  a)› have ‹trace-class b›
  by simp
from ‹trace-class a› ‹trace-class b› show ?thesis
  by simp
qed
qed

```

lemma trace-tensor: ‹trace (a \otimes_o b) = trace a * trace b›

— [8, Lemma 32]

proof —

consider (tc) ‹trace-class a› ‹trace-class b› | (zero) ‹a = 0 \vee b = 0› | (nota) ‹a $\neq 0$ › ‹b $\neq 0$ › ‹ \neg trace-class a› | (notb) ‹a $\neq 0$ › ‹b $\neq 0$ › ‹ \neg trace-class b›

by blast

then show ?thesis

proof cases

case tc

then have *: ‹trace-class (a \otimes_o b)›

by (simp add: trace-class-tensor)

have sum: ‹($\lambda(x, y). ket (x, y) \cdot_C ((a \otimes_o b) *_V ket (x, y))$) summable-on UNIV›

using trace-exists[OF is-onb-ket *]

```

    by (simp-all add: o-def case-prod-unfold summable-on-reindex)

  have ⟨trace a * trace b = (∑∞x. ∑∞y. ket x •C (a *V ket x) * (ket y •C (b *V ket y)))⟩
    apply (simp add: trace-ket-sum tc flip: infsum-cmult-left')
    by (simp flip: infsum-cmult-right')
  also have ⟨... = (∑∞x. ∑∞y. ket (x,y) •C ((a ⊗o b) *V ket (x,y)))⟩
    by (simp add: tensor-op-ell2 flip: tensor-ell2-ket)
  also have ⟨... = (∑∞xy∈UNIV. ket xy •C ((a ⊗o b) *V ket xy))⟩
    apply (simp add: sum infsum-Sigma'-banach)
    by (simp add: case-prod-unfold)
  also have ⟨... = trace (a ⊗o b)⟩
    by (simp add: * trace-ket-sum)
  finally show ?thesis
    by simp
next
case zero
then show ?thesis by auto
next
case nota
then have [simp]: ⟨trace a = 0⟩
  unfolding trace-def by simp
from nota have ⟨¬ trace-class (a ⊗o b)⟩
  by (simp add: trace-class-tensor-iff)
then have [simp]: ⟨trace (a ⊗o b) = 0⟩
  unfolding trace-def by simp
show ?thesis
  by simp
next
case notb
then have [simp]: ⟨trace b = 0⟩
  unfolding trace-def by simp
from notb have ⟨¬ trace-class (a ⊗o b)⟩
  by (simp add: trace-class-tensor-iff)
then have [simp]: ⟨trace (a ⊗o b) = 0⟩
  unfolding trace-def by simp
show ?thesis
  by simp
qed
qed

```

lemma isometry-tensor-op: ⟨isometry (U ⊗_o V)⟩ **if** ⟨isometry U⟩ **and** ⟨isometry V⟩
 unfolding isometry-def **using that by** (simp add: tensor-op-adjoint comp-tensor-op)

lemma is-Proj-tensor-op: ⟨is-Proj a ⟹ is-Proj b ⟹ is-Proj (a ⊗_o b)⟩
 by (simp add: comp-tensor-op is-Proj-algebraic tensor-op-adjoint)

lemma isometry-tensor-id-right[simp]:
 fixes U :: ⟨'a ell2 ⇒_{CL} 'b ell2⟩
 shows ⟨isometry (U ⊗_o (id-cblinfun :: 'c ell2 ⇒_{CL} -)) ⟷ isometry U⟩

proof (*rule iffI*)
 assume $\langle \text{isometry } U \rangle$
 then show $\langle \text{isometry } (U \otimes_o \text{id-cblinfun}) \rangle$
 unfolding *isometry-def*
 by (*auto simp add: tensor-op-adjoint comp-tensor-op*)
next
 let $?id = \langle \text{id-cblinfun} :: 'c \text{ ell2} \Rightarrow_{CL} - \rangle$
 assume *asm*: $\langle \text{isometry } (U \otimes_o ?id) \rangle$
 then have $\langle (U * o_{CL} U) \otimes_o ?id = \text{id-cblinfun} \otimes_o ?id \rangle$
 by (*simp add: isometry-def tensor-op-adjoint comp-tensor-op*)
 then have $\langle U * o_{CL} U = \text{id-cblinfun} \rangle$
 by (*rule inj-tensor-left[of ?id, unfolded inj-def, rule-format, rotated]*) *simp*
 then show $\langle \text{isometry } U \rangle$
 by (*simp add: isometry-def*)
qed

lemma *isometry-tensor-id-left[simp]*:
 fixes $U :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \rangle$
 shows $\langle \text{isometry } ((\text{id-cblinfun} :: 'c \text{ ell2} \Rightarrow_{CL} -) \otimes_o U) \longleftrightarrow \text{isometry } U \rangle$

proof (*rule iffI*)
 assume $\langle \text{isometry } U \rangle$
 then show $\langle \text{isometry } (\text{id-cblinfun} \otimes_o U) \rangle$
 unfolding *isometry-def*
 by (*auto simp add: tensor-op-adjoint comp-tensor-op*)
next
 let $?id = \langle \text{id-cblinfun} :: 'c \text{ ell2} \Rightarrow_{CL} - \rangle$
 assume *asm*: $\langle \text{isometry } (?id \otimes_o U) \rangle$
 then have $\langle ?id \otimes_o (U * o_{CL} U) = ?id \otimes_o \text{id-cblinfun} \rangle$
 by (*simp add: isometry-def tensor-op-adjoint comp-tensor-op*)
 then have $\langle U * o_{CL} U = \text{id-cblinfun} \rangle$
 by (*rule inj-tensor-right[of ?id, unfolded inj-def, rule-format, rotated]*) *simp*
 then show $\langle \text{isometry } U \rangle$
 by (*simp add: isometry-def*)
qed

lemma *unitary-tensor-id-right[simp]*: $\langle \text{unitary } (U \otimes_o \text{id-cblinfun}) \longleftrightarrow \text{unitary } U \rangle$
 unfolding *unitary-twosided-isometry*
 by (*simp add: tensor-op-adjoint*)

lemma *unitary-tensor-id-left[simp]*: $\langle \text{unitary } (\text{id-cblinfun} \otimes_o U) \longleftrightarrow \text{unitary } U \rangle$
 unfolding *unitary-twosided-isometry*
 by (*simp add: tensor-op-adjoint*)

lemma *sandwich-tensor-op*: $\langle \text{sandwich } (a \otimes_o b) (c \otimes_o d) = \text{sandwich } a \ c \otimes_o \text{sandwich } b \ d \rangle$
 by (*simp add: sandwich-apply tensor-op-adjoint flip: cblinfun-compose-assoc comp-tensor-op*)

lemma *sandwich-assoc-ell2-tensor-op[simp]*: $\langle \text{sandwich } \text{assoc-ell2} ((a \otimes_o b) \otimes_o c) = a \otimes_o (b \otimes_o c) \rangle$

by (*auto intro!*: *tensor-ell2-extensionality3*
simp: *sandwich-apply assoc-ell2'-tensor assoc-ell2-tensor tensor-op-ell2*)

lemma *unitary-tensor-op*: $\langle \text{unitary } (a \otimes_o b) \rangle$ **if** [*simp*]: $\langle \text{unitary } a \rangle \langle \text{unitary } b \rangle$
by (*auto intro!*: *unitaryI simp add: tensor-op-adjoint comp-tensor-op*)

lemma *tensor-ell2-right-butterfly*: $\langle \text{tensor-ell2-right } \psi \text{ } o_{CL} \text{ tensor-ell2-right } \varphi^* = \text{id-cblinfun} \otimes_o \text{ butterfly } \psi \varphi \rangle$

by (*auto intro!*: *equal-ket cinner-ket-eqI simp: tensor-op-ell2 simp flip: tensor-ell2-ket*)

lemma *tensor-ell2-left-butterfly*: $\langle \text{tensor-ell2-left } \psi \text{ } o_{CL} \text{ tensor-ell2-left } \varphi^* = \text{butterfly } \psi \varphi \otimes_o \text{id-cblinfun} \rangle$

by (*auto intro!*: *equal-ket cinner-ket-eqI simp: tensor-op-ell2 simp flip: tensor-ell2-ket*)

lift-definition *tc-tensor* :: $\langle ('a \text{ ell2}, 'b \text{ ell2}) \text{ trace-class} \Rightarrow ('c \text{ ell2}, 'd \text{ ell2}) \text{ trace-class} \Rightarrow$
 $(('a \times 'c) \text{ ell2}, ('b \times 'd) \text{ ell2}) \text{ trace-class} \rangle$ **is**

tensor-op

by (*auto intro!*: *trace-class-tensor*)

lemma *trace-norm-tensor*: $\langle \text{trace-norm } (a \otimes_o b) = \text{trace-norm } a * \text{trace-norm } b \rangle$

by (*rule of-real-hom.injectivity[where 'a=complex]*)

(*simp add: abs-op-tensor trace-tensor flip: trace-abs-op*)

lemma *bounded-cbilinear-tc-tensor*: $\langle \text{bounded-cbilinear } \text{tc-tensor} \rangle$

unfolding *bounded-cbilinear-def*

by *transfer*

(*auto intro!*: *exI[of - 1]*)

simp: trace-norm-tensor tensor-op-left-add tensor-op-right-add tensor-op-scaleC-left

tensor-op-scaleC-right)

lemmas *bounded-clinear-tc-tensor-left[bounded-clinear]* = *bounded-cbilinear.bounded-clinear-left[OF bounded-cbilinear-tc-tensor]*

lemmas *bounded-clinear-tc-tensor-right[bounded-clinear]* = *bounded-cbilinear.bounded-clinear-right[OF bounded-cbilinear-tc-tensor]*

lemma *tc-tensor-scaleC-left*: $\langle \text{tc-tensor } (c *_C a) b = c *_C \text{tc-tensor } a b \rangle$

by *transfer'* (*simp add: tensor-op-scaleC-left*)

lemma *tc-tensor-scaleC-right*: $\langle \text{tc-tensor } a (c *_C b) = c *_C \text{tc-tensor } a b \rangle$

by *transfer'* (*simp add: tensor-op-scaleC-right*)

lemma *comp-tc-tensor*: $\langle \text{tc-compose } (\text{tc-tensor } a b) (\text{tc-tensor } c d) = \text{tc-tensor } (\text{tc-compose } a c) (\text{tc-compose } b d) \rangle$

by *transfer'* (*rule comp-tensor-op*)

lemma *norm-tc-tensor*: $\langle \text{norm } (\text{tc-tensor } a b) = \text{norm } a * \text{norm } b \rangle$

by (*transfer'*, *rule of-real-hom.injectivity[where 'a=complex]*)

(*simp add: abs-op-tensor trace-tensor flip: trace-abs-op*)

lemma *tc-tensor-pos*: $\langle \text{tc-tensor } a b \geq 0 \rangle$ **if** $\langle a \geq 0 \rangle$ **and** $\langle b \geq 0 \rangle$

for $a :: \langle 'a \text{ ell2}, 'a \text{ ell2} \rangle \text{ trace-class}$ and $b :: \langle 'b \text{ ell2}, 'b \text{ ell2} \rangle \text{ trace-class}$
 using that by *transfer'* (rule *tensor-op-pos*)

interpretation *tensor-op-cbilinear*: *bounded-cbilinear tensor-op*
 by *simp*

lemma *tensor-op-mono-left*:

fixes $a :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2} \rangle$ and $c :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \rangle$
assumes $\langle a \leq b \rangle$ and $\langle c \geq 0 \rangle$
shows $\langle a \otimes_o c \leq b \otimes_o c \rangle$

proof –

have $\langle b - a \geq 0 \rangle$
 by (*simp add: assms(1)*)
with $\langle c \geq 0 \rangle$ **have** $\langle (b - a) \otimes_o c \geq 0 \rangle$
 by (*intro tensor-op-pos*)
then have $\langle b \otimes_o c - a \otimes_o c \geq 0 \rangle$
 by (*simp add: tensor-op-cbilinear.diff-left*)
then show *?thesis*
 by *simp*

qed

lemma *tensor-op-mono-right*:

fixes $a :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2} \rangle$ and $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \rangle$
assumes $\langle b \leq c \rangle$ and $\langle a \geq 0 \rangle$
shows $\langle a \otimes_o b \leq a \otimes_o c \rangle$

proof –

have $\langle c - b \geq 0 \rangle$
 by (*simp add: assms(1)*)
with $\langle a \geq 0 \rangle$ **have** $\langle a \otimes_o (c - b) \geq 0 \rangle$
 by (*intro tensor-op-pos*)
then have $\langle a \otimes_o c - a \otimes_o b \geq 0 \rangle$
 by (*simp add: tensor-op-cbilinear.diff-right*)
then show *?thesis*
 by *simp*

qed

lemma *tensor-op-mono*:

fixes $a :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2} \rangle$ and $c :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \rangle$
assumes $\langle a \leq b \rangle$ and $\langle c \leq d \rangle$ and $\langle b \geq 0 \rangle$ and $\langle c \geq 0 \rangle$
shows $\langle a \otimes_o c \leq b \otimes_o d \rangle$

proof –

have $\langle a \otimes_o c \leq b \otimes_o c \rangle$
 using $\langle a \leq b \rangle$ and $\langle c \geq 0 \rangle$
 by (*rule tensor-op-mono-left*)
also have $\langle \dots \leq b \otimes_o d \rangle$
 using $\langle c \leq d \rangle$ and $\langle b \geq 0 \rangle$
 by (*rule tensor-op-mono-right*)
finally show *?thesis*

by –
qed

lemma sandwich-tc-tensor: $\langle \text{sandwich-tc } (E \otimes_o F) (tc\text{-tensor } t \ u) = tc\text{-tensor } (\text{sandwich-tc } E \ t) (\text{sandwich-tc } F \ u) \rangle$
by (transfer fixing: $E \ F$) (simp add: sandwich-tensor-op)

lemma tensor-tc-butterfly: $tc\text{-tensor } (tc\text{-butterfly } \psi \ \psi') (tc\text{-butterfly } \varphi \ \varphi') = tc\text{-butterfly } (tensor\text{-ell2 } \psi \ \varphi) (tensor\text{-ell2 } \psi' \ \varphi')$
by (transfer fixing: $\varphi \ \varphi' \ \psi \ \psi'$) (simp add: tensor-butterfly)

lemma separating-set-bounded-clinear-tc-tensor:
shows $\langle \text{separating-set bounded-clinear } ((\lambda(\varrho, \sigma). tc\text{-tensor } \varrho \ \sigma) \ ' (UNIV \times UNIV)) \rangle$
proof –
have $\langle \top = ccspace ((\lambda(x, y). tc\text{-butterfly } x \ y) \ ' (range \ ket \times range \ ket)) \rangle$
by (simp add: onb-butterflies-span-trace-class)
also have $\langle \dots = ccspace ((\lambda(x, y, z, w). tc\text{-butterfly } (x \otimes_s y) (z \otimes_s w)) \ ' (range \ ket \times range \ ket \times range \ ket \times range \ ket)) \rangle$
by (auto intro!: arg-cong[where f=ccspan] image-eqI simp: tensor-ell2-ket)
also have $\langle \dots = ccspace ((\lambda(x, y, z, w). tc\text{-tensor } (tc\text{-butterfly } x \ z) (tc\text{-butterfly } y \ w)) \ ' (range \ ket \times range \ ket \times range \ ket \times range \ ket)) \rangle$
by (simp add: tensor-tc-butterfly)
also have $\langle \dots \leq ccspace ((\lambda(\varrho, \sigma). tc\text{-tensor } \varrho \ \sigma) \ ' (UNIV \times UNIV)) \rangle$
by (auto intro!: ccspace-mono)
finally show ?thesis
by (intro separating-set-bounded-clinear-dense) (use top-le in blast)
qed

lemma separating-set-bounded-clinear-tc-tensor-nested:
assumes $\langle \text{separating-set } (bounded-clinear \ :: \ (- \Rightarrow \ 'e::\text{complex-normed-vector}) \Rightarrow \ -) \ A \rangle$
assumes $\langle \text{separating-set } (bounded-clinear \ :: \ (- \Rightarrow \ 'e::\text{complex-normed-vector}) \Rightarrow \ -) \ B \rangle$
shows $\langle \text{separating-set } (bounded-clinear \ :: \ (- \Rightarrow \ 'e::\text{complex-normed-vector}) \Rightarrow \ -) \ ((\lambda(\varrho, \sigma). tc\text{-tensor } \varrho \ \sigma) \ ' (A \times B)) \rangle$
using separating-set-bounded-clinear-tc-tensor bounded-cbilinear-tc-tensor assms
by (rule separating-set-bounded-cbilinear-nested)

lemma tc-tensor-0-left[simp]: $\langle tc\text{-tensor } 0 \ x = 0 \rangle$
by transfer' simp

lemma tc-tensor-0-right[simp]: $\langle tc\text{-tensor } x \ 0 = 0 \rangle$
by transfer' simp

14.3 Tensor product of subspaces

definition tensor-ccsubspace (infixr \otimes_S 70) where

$\langle \text{tensor-ccsubspace } A \ B = \text{ccspan } \{ \psi \otimes_s \varphi \mid \psi \ \varphi. \ \psi \in \text{space-as-set } A \wedge \varphi \in \text{space-as-set } B \} \rangle$

lemma *tensor-ccsubspace-via-Proj*: $\langle A \otimes_S B = (\text{Proj } A \otimes_o \text{Proj } B) *_S \top \rangle$

proof (*rule antisym*)

have $\langle \psi \otimes_s \varphi \in \text{space-as-set } ((\text{Proj } A \otimes_o \text{Proj } B) *_S \top) \rangle$ **if** $\langle \psi \in \text{space-as-set } A \rangle$ **and** $\langle \varphi \in \text{space-as-set } B \rangle$ **for** $\psi \ \varphi$

by (*metis Proj-fixes-image cblinfun-apply-in-image tensor-op-ell2 that(1) that(2)*)

then show $\langle A \otimes_S B \leq (\text{Proj } A \otimes_o \text{Proj } B) *_S \top \rangle$

by (*auto intro!: ccspan-leqI simp: tensor-ccsubspace-def*)

have $*$: $\langle \text{ccspan } \{ \psi \otimes_s \varphi \mid (\psi :: 'a \ \text{ell2}) (\varphi :: 'b \ \text{ell2}). \ \text{True} \} = \top \rangle$

using *tensor-ell2-dense* [**where** $A = \langle \text{UNIV} :: 'a \ \text{ell2} \ \text{set} \rangle$ **and** $B = \langle \text{UNIV} :: 'b \ \text{ell2} \ \text{set} \rangle$]

by *auto*

have $\langle (\text{Proj } A \otimes_o \text{Proj } B) *_V \psi \otimes_s \varphi \in \text{space-as-set } (A \otimes_S B) \rangle$ **for** $\psi \ \varphi$

unfolding *tensor-op-ell2 tensor-ccsubspace-def*

by (*smt (verit) Proj-range cblinfun-apply-in-image ccspan-superset mem-Collect-eq subsetD*)

then show $\langle (\text{Proj } A \otimes_o \text{Proj } B) *_S \top \leq A \otimes_S B \rangle$

by (*auto intro!: ccspan-leqI simp: cblinfun-image-ccspan simp flip: **)

qed

lemma *tensor-ccsubspace-top[simp]*: $\langle \top \otimes_S \top = \top \rangle$

by (*simp add: tensor-ccsubspace-via-Proj*)

lemma *tensor-ccsubspace-0-left[simp]*: $\langle 0 \otimes_S X = 0 \rangle$

by (*simp add: tensor-ccsubspace-via-Proj*)

lemma *tensor-ccsubspace-0-right[simp]*: $\langle X \otimes_S 0 = 0 \rangle$

by (*simp add: tensor-ccsubspace-via-Proj*)

lemma *tensor-ccsubspace-image*: $\langle (A *_S T) \otimes_S (B *_S U) = (A \otimes_o B) *_S (T \otimes_S U) \rangle$

proof –

have $\langle (A *_S T) \otimes_S (B *_S U) = \text{ccspan } ((\lambda(\psi, \varphi). \ \psi \otimes_s \varphi) \text{ ` } (\text{space-as-set } (A *_S T) \times \text{space-as-set } (B *_S U))) \rangle$

by (*simp add: tensor-ccsubspace-def set-compr-2-image-collect ccspan.rep-eq*)

also have $\langle \dots = \text{ccspan } ((\lambda(\psi, \varphi). \ \psi \otimes_s \varphi) \text{ ` } \text{closure } ((A \text{ ` } \text{space-as-set } T) \times (B \text{ ` } \text{space-as-set } U))) \rangle$

by (*simp add: cblinfun-image.rep-eq closure-Times*)

also have $\langle \dots = \text{ccspan } (\text{closure } ((\lambda(\psi, \varphi). \ \psi \otimes_s \varphi) \text{ ` } ((A \text{ ` } \text{space-as-set } T) \times (B \text{ ` } \text{space-as-set } U)))) \rangle$

by (*subst closure-image-closure[symmetric]*)

(*use continuous-on-subset continuous-tensor-ell2 in auto*)

also have $\langle \dots = \text{ccspan } ((\lambda(\psi, \varphi). \ \psi \otimes_s \varphi) \text{ ` } ((A \text{ ` } \text{space-as-set } T) \times (B \text{ ` } \text{space-as-set } U))) \rangle$

by *simp*

also have $\langle \dots = (A \otimes_o B) *_S \text{ccspan } ((\lambda(\psi, \varphi). \ \psi \otimes_s \varphi) \text{ ` } (\text{space-as-set } T \times \text{space-as-set } U)) \rangle$

by (*simp add: cblinfun-image-ccspan image-image tensor-op-ell2 case-prod-beta*)

flip: map-prod-image)

also have $\langle \dots = (A \otimes_o B) *_S (T \otimes_S U) \rangle$

by (*simp add: tensor-ccsubspace-def set-compr-2-image-collect*)

finally show *?thesis*
by –
qed

lemma *tensor-ccsubspace-bot-left[simp]*: $\langle \perp \otimes_S S = \perp \rangle$
by (*simp add: tensor-ccsubspace-via-Proj*)
lemma *tensor-ccsubspace-bot-right[simp]*: $\langle S \otimes_S \perp = \perp \rangle$
by (*simp add: tensor-ccsubspace-via-Proj*)

lemma *swap-ell2-tensor-ccsubspace*: $\langle \text{swap-ell2 } *_S (S \otimes_S T) = T \otimes_S S \rangle$
by (*force intro!: arg-cong[where f=ccspan]*
simp add: tensor-ccsubspace-def cblinfun-image-ccspan image-image set-compr-2-image-collect)

lemma *tensor-ccsubspace-right1dim-member*:
assumes $\langle \psi \in \text{space-as-set } (S \otimes_S \text{ccspan}\{\varphi\}) \rangle$
shows $\langle \exists \psi'. \psi = \psi' \otimes_s \varphi \rangle$
proof (*cases* $\langle \varphi = 0 \rangle$)
case *True*
with *assms* **show** *?thesis*
by *simp*

next
case *False*
have $\langle \{\psi \otimes_s \varphi' \mid \psi \varphi'. \psi \in \text{space-as-set } S \wedge \varphi' \in \text{space-as-set } (\text{ccspan } \{\varphi\})\}$
 $= \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
proof –
have $\langle \psi \in \text{space-as-set } S \implies \exists \psi'. \psi \otimes_s c *_C \varphi = \psi' \otimes_s \varphi \wedge \psi' \in \text{space-as-set } S \rangle$ **for** $c \psi$
by (*rule exI[where x= $c *_C \psi$]*)
(auto simp: tensor-ell2-scaleC2 tensor-ell2-scaleC1
complex-vector.subspace-scale)
moreover have $\langle \psi \in \text{space-as-set } S \implies$
 $\exists \psi' \varphi'. \psi \otimes_s \varphi = \psi' \otimes_s \varphi' \wedge \psi' \in \text{space-as-set } S \wedge \varphi' \in \text{range } (\lambda k. k *_C \varphi) \rangle$ **for** ψ
by (*rule exI[where x= ψ], rule exI[where x= φ]*)
(auto intro!: range-eqI[where x= $\langle 1::\text{complex} \rangle$])
ultimately show *?thesis*
by (*auto simp: ccspan-finite complex-vector.span-singleton*)

qed
moreover have $\langle \text{csubspace } \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
proof (*rule complex-vector.subspaceI*)
show $\langle 0 \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
by (*auto intro!: exI[where x= 0]*)
show $\langle x + y \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
if $x: \langle x \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
and $y: \langle y \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$ **for** $x y$
using *that complex-vector.subspace-add tensor-ell2-add1*
unfolding *mem-Collect-eq* **by** (*metis csubspace-space-as-set*)
show $\langle c *_C x \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
if $x: \langle x \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$ **for** $c x$
using *that complex-vector.subspace-scale tensor-ell2-scaleC2 tensor-ell2-scaleC1*
unfolding *mem-Collect-eq* **by** (*metis csubspace-space-as-set*)

qed
moreover have $\langle \text{closed } \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
proof (*rule closed-sequential-limits*[*THEN iffD2, rule-format*])
fix $x \ l$
assume *asm*: $\langle (\forall n. x \ n \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\}) \wedge x \longrightarrow l \rangle$
then obtain ψ' **where** *x-def*: $\langle x \ n = \psi' \ n \otimes_s \varphi \rangle$ **and** $\psi'-S$: $\langle \psi' \ n \in \text{space-as-set } S \rangle$ **for** n
unfolding *mem-Collect-eq* **by** *metis*
from *asm* **have** $\langle x \longrightarrow l \rangle$
by *simp*
have $\langle \text{Cauchy } \psi' \rangle$
proof (*rule CauchyI*)
fix $e :: \text{real}$ **assume** $\langle e > 0 \rangle$
define d **where** $\langle d = e * \text{norm } \varphi \rangle$
with *False* $\langle e > 0 \rangle$ **have** $\langle d > 0 \rangle$
by *auto*
from $\langle x \longrightarrow l \rangle$
have $\langle \text{Cauchy } x \rangle$
using *LIMSEQ-imp-Cauchy* **by** *blast*
then obtain M **where** $\langle \forall m \geq M. \forall n \geq M. \text{norm } (x \ m - x \ n) < d \rangle$
using *Cauchy-iff* $\langle 0 < d \rangle$ **by** *blast*
then show $\langle \exists M. \forall m \geq M. \forall n \geq M. \text{norm } (\psi' \ m - \psi' \ n) < e \rangle$
by (*intro exI*[*of - M*])
(use False in auto simp add: x-def norm-tensor-ell2 d-def simp flip: tensor-ell2-diff1)
qed
then obtain l' **where** $\langle \psi' \longrightarrow l' \rangle$
using *convergent-eq-Cauchy* **by** *blast*
with $\psi'-S$ **have** $l'-S$: $\langle l' \in \text{space-as-set } S \rangle$
by (*metis* $\langle \text{Cauchy } \psi' \rangle$ *completeE complete-space-as-set limI*)
from $\langle \psi' \longrightarrow l' \rangle$ **have** $\langle x \longrightarrow l' \otimes_s \varphi \rangle$
by (*auto intro: tendsto-eq-intros simp: x-def*[*abs-def*])
with $\langle x \longrightarrow l \rangle$ **have** $\langle l = l' \otimes_s \varphi \rangle$
using *LIMSEQ-unique* **by** *blast*
then show $\langle l \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
using $l'-S$ **by** *auto*
qed
ultimately have $\langle \text{space-as-set } (\text{ccspan } \{\psi \otimes_s \varphi' \mid \psi \varphi'. \psi \in \text{space-as-set } S \wedge \varphi' \in \text{space-as-set } (\text{ccspan } \{\varphi\})\})$
 $= \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
by (*simp add: ccspan.rep-eq complex-vector.span-eq-iff*[*THEN iffD2*])
with *assms* **have** $\langle \psi \in \{\psi \otimes_s \varphi \mid \psi. \psi \in \text{space-as-set } S\} \rangle$
by (*simp add: tensor-ccsubspace-def*)
then show $\langle \exists \psi'. \psi = \psi' \otimes_s \varphi \rangle$
by *auto*
qed
lemma *tensor-ccsubspace-left1dim-member*:
assumes $\langle \psi \in \text{space-as-set } (\text{ccspan}\{\varphi\} \otimes_S S) \rangle$
shows $\langle \exists \psi'. \psi = \varphi \otimes_s \psi' \rangle$
proof –

```

from assms
have  $\langle \text{swap-ell2 } *V \psi \in \text{space-as-set } (\text{swap-ell2 } *S (\text{ccspan } \{\varphi\} \otimes_S S)) \rangle$ 
by (metis rev-image-eqI space-as-set-image-commute swap-ell2-selfinv)
then have  $\langle \text{swap-ell2 } \psi \in \text{space-as-set } (S \otimes_S \text{ccspan}\{\varphi\}) \rangle$ 
  by (simp add: swap-ell2-tensor-ccsubspace)
then obtain  $\psi'$  where  $\psi': \langle \text{swap-ell2 } \psi = \psi' \otimes_s \varphi \rangle$ 
  using tensor-ccsubspace-right1dim-member by blast
have  $\langle \psi = \text{swap-ell2 } *V \text{swap-ell2 } *V \psi \rangle$ 
  by (simp flip: cblinfun-apply-cblinfun-compose)
also have  $\langle \dots = \text{swap-ell2 } *V (\psi' \otimes_s \varphi) \rangle$ 
  by (simp add: \psi')
also have  $\langle \dots = \varphi \otimes_s \psi' \rangle$ 
  by simp
finally show ?thesis
  by auto
qed

lemma tensor-ell2-mem-tensor-ccsubspace-left:
  assumes  $\langle a \otimes_s b \in \text{space-as-set } (S \otimes_S T) \rangle$  and  $\langle b \neq 0 \rangle$ 
  shows  $\langle a \in \text{space-as-set } S \rangle$ 
proof (cases  $\langle a = 0 \rangle$ )
  case True
  then show ?thesis
    by simp
  next
  case False
  have  $\langle \text{norm } (\text{Proj } S \ a) * \text{norm } (\text{Proj } T \ b) = \text{norm } ((\text{Proj } S \ a) \otimes_s (\text{Proj } T \ b)) \rangle$ 
    by (simp add: norm-tensor-ell2)
  also have  $\langle \dots = \text{norm } (\text{Proj } (S \otimes_S T) (a \otimes_s b)) \rangle$ 
    by (simp add: tensor-ccsubspace-via-Proj Proj-on-own-range is-Proj-tensor-op tensor-op-ell2)
  also from assms have  $\langle \dots = \text{norm } (a \otimes_s b) \rangle$ 
    by (simp add: Proj-fixes-image)
  also have  $\langle \dots = \text{norm } a * \text{norm } b \rangle$ 
    by (simp add: norm-tensor-ell2)
  finally have prod-eq:  $\langle \text{norm } (\text{Proj } S \ *V \ a) * \text{norm } (\text{Proj } T \ *V \ b) = \text{norm } a * \text{norm } b \rangle$ 
    by –
  with False  $\langle b \neq 0 \rangle$  have Tb-non0:  $\langle \text{norm } (\text{Proj } T \ *V \ b) \neq 0 \rangle$ 
    by fastforce
  have  $\langle \text{norm } (\text{Proj } S \ a) = \text{norm } a \rangle$ 
proof (rule ccontr)
  assume asm:  $\langle \text{norm } (\text{Proj } S \ *V \ a) \neq \text{norm } a \rangle$ 
  have Sa-leq:  $\langle \text{norm } (\text{Proj } S \ *V \ a) \leq \text{norm } a \rangle$ 
    by (simp add: is-Proj-reduces-norm)
  have Tb-leq:  $\langle \text{norm } (\text{Proj } T \ *V \ b) \leq \text{norm } b \rangle$ 
    by (simp add: is-Proj-reduces-norm)
  from asm Sa-leq have  $\langle \text{norm } (\text{Proj } S \ *V \ a) < \text{norm } a \rangle$ 
    by simp
  then have  $\langle \text{norm } (\text{Proj } S \ *V \ a) * \text{norm } (\text{Proj } T \ *V \ b) < \text{norm } a * \text{norm } (\text{Proj } T \ *V \ b) \rangle$ 

```

```

    using Tb-non0 by auto
    also from Tb-leq have ⟨... ≤ norm a * norm b⟩
    using False by force
    also note prod-eq
    finally show False
    by simp
qed
then show ⟨a ∈ space-as-set S⟩
using norm-Proj-apply by blast
qed

```

```

lemma tensor-ell2-mem-tensor-ccsubspace-right:
  assumes ⟨a ⊗s b ∈ space-as-set (S ⊗S T)⟩ and ⟨a ≠ 0⟩
  shows ⟨b ∈ space-as-set T⟩
proof -
  have ⟨swap-ell2 *V (a ⊗s b) ∈ space-as-set (swap-ell2 *S (S ⊗S T))⟩
  using assms(1) cblinfun-apply-in-image' by blast
  then have ⟨b ⊗s a ∈ space-as-set (T ⊗S S)⟩
  by (simp add: swap-ell2-tensor-ccsubspace)
  then show ⟨b ∈ space-as-set T⟩
  using ⟨a ≠ 0⟩ by (rule tensor-ell2-mem-tensor-ccsubspace-left)
qed

```

```

lemma tensor-ell2-in-tensor-ccsubspace: ⟨a ⊗s b ∈ space-as-set (A ⊗S B)⟩ if ⟨a ∈ space-as-set A⟩ and ⟨b ∈ space-as-set B⟩
— Converse is tensor-ell2-mem-tensor-ccsubspace-left and ...-right.
using that by (auto intro!: ccspan-superset[THEN subsetD] simp add: tensor-ccsubspace-def)

```

```

lemma tensor-ccsubspace-INF-left-top:
  fixes S :: ⟨'a ⇒ 'b ell2 ccsubspace⟩
  shows ⟨(INF x∈X. S x) ⊗S (⊔::'c ell2 ccsubspace) = (INF x∈X. S x ⊗S ⊔)⟩
proof (rule antisym[rotated])
  let ?top = ⟨⊔ :: 'c ell2 ccsubspace⟩
  have *: ⟨ψ ⊗s φ ∈ space-as-set (⊓x∈X. S x ⊗S ?top)⟩
  if ⟨ψ ∈ space-as-set (⊓x∈X. S x)⟩ for ψ φ
  proof -
    from that(1) have ⟨ψ ∈ space-as-set (S x)⟩ if ⟨x ∈ X⟩ for x
    using that by (simp add: Inf-ccsubspace.rep-eq)
    then have ⟨ψ ⊗s φ ∈ space-as-set (S x ⊗S ⊔)⟩ if ⟨x ∈ X⟩ for x
    using ccspan-superset that by (force simp: tensor-ccsubspace-def)
    then show ?thesis
    by (simp add: Inf-ccsubspace.rep-eq)
  qed
  show ⟨(INF x∈X. S x) ⊗S ?top ≤ (INF x∈X. S x ⊗S ?top)⟩
  by (subst tensor-ccsubspace-def, rule ccspan-leqI) (use * in auto)

  show ⟨(⊓x∈X. S x ⊗S ?top) ≤ (⊓x∈X. S x) ⊗S ?top⟩
  proof (rule ccsubspace-leI-unit)
    fix ψ

```

assume asm : $\langle \psi \in \text{space-as-set } (\prod_{x \in X}. S\ x \otimes_S ?top) \rangle$
obtain ψ' **where** $\psi'b$ - b : $\langle \psi' b \otimes_s \text{ket } b = (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b)) *_V \psi \rangle$ **for**

b

proof (*atomize-elim, rule choice, intro allI*)
fix $b :: 'c$
have $\langle (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b)) *_V \psi \in \text{space-as-set } (\top \otimes_S \text{ccspan } \{\text{ket } b\}) \rangle$
by (*simp add: butterfly-eq-proj tensor-ccsubspace-via-Proj*)
then show $\langle \exists \psi'. \psi' \otimes_s \text{ket } b = (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b)) *_V \psi \rangle$
by (*metis tensor-ccsubspace-right1dim-member*)
qed

have $\langle \psi' b \in \text{space-as-set } (S\ x) \rangle$ **if** $\langle x \in X \rangle$ **for** $x\ b$
proof –
from asm **have** ψ - ST : $\langle \psi \in \text{space-as-set } (S\ x \otimes_S ?top) \rangle$
by (*meson INF-lower Set.basic-monos(γ) less-eq-ccsubspace.rep-eq that*)
have $\langle \psi' b \otimes_s \text{ket } b = (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b)) *_V \psi \rangle$
by (*simp add: $\psi'b$ - b*)
also from ψ - ST
have $\langle \dots \in \text{space-as-set } (((\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b))) *_S (S\ x \otimes_S ?top)) \rangle$
by (*meson cblinfun-apply-in-image'*)
also have $\langle \dots = \text{space-as-set } (((\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b)) o_{CL} (\text{Proj } (S\ x) \otimes_o \text{id-cblinfun})) *_S \top) \rangle$
by (*simp add: cblinfun-compose-image tensor-ccsubspace-via-Proj*)
also have $\langle \dots = \text{space-as-set } ((\text{Proj } (S\ x) \otimes_o (\text{butterfly } (\text{ket } b) (\text{ket } b) o_{CL} \text{id-cblinfun})) *_S \top) \rangle$
by (*simp add: comp-tensor-op*)
also have $\langle \dots = \text{space-as-set } ((\text{Proj } (S\ x) \otimes_o (\text{id-cblinfun } o_{CL} \text{butterfly } (\text{ket } b) (\text{ket } b))) *_S \top) \rangle$
by *simp*
also have $\langle \dots = \text{space-as-set } (((\text{Proj } (S\ x) \otimes_o \text{id-cblinfun}) o_{CL} (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b))) *_S \top) \rangle$
by (*simp add: comp-tensor-op*)
also have $\langle \dots \subseteq \text{space-as-set } ((\text{Proj } (S\ x) \otimes_o \text{id-cblinfun}) *_S \top) \rangle$
by (*metis cblinfun-compose-image cblinfun-image-mono less-eq-ccsubspace.rep-eq top-greatest*)
also have $\langle \dots = \text{space-as-set } (S\ x \otimes_S ?top) \rangle$
by (*simp add: tensor-ccsubspace-via-Proj*)
finally have $\langle \psi' b \otimes_s \text{ket } b \in \text{space-as-set } (S\ x \otimes_S ?top) \rangle$
by –
then show $\langle \psi' b \in \text{space-as-set } (S\ x) \rangle$
using *tensor-ell2-mem-tensor-ccsubspace-left*
by (*metis ket-nonzero*)
qed

then have $\langle \psi' b \in \text{space-as-set } (\prod_{x \in X}. S\ x) \rangle$ **if** $\langle x \in X \rangle$ **for** $x\ b$
using that **by** (*simp add: Inf-ccsubspace.rep-eq*)

then have $*$: $\langle \psi' b \otimes_s \text{ket } b \in \text{space-as-set } ((\prod_{x \in X}. S\ x) \otimes_S ?top) \rangle$ **for** b
by (*auto intro!: ccspan-superset[THEN set-mp]*)
simp add: tensor-ccsubspace-def Inf-ccsubspace.rep-eq

have $\langle \psi \in \text{space-as-set } (\text{ccspan } (\text{range } (\lambda b. \psi' b \otimes_s \text{ket } b))) \rangle$ **(is** $\langle \psi \in ?rhs \rangle$)
proof –
define γ **where** $\langle \gamma F = (\sum b \in F. (\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } b) (\text{ket } b)) *_V \psi) \rangle$ **for** F
have γ -*rhs*: $\langle \gamma F \in ?rhs \rangle$ **for** F
using *ccspan-superset* **by** (*force intro!*: *complex-vector.subspace-sum simp add: γ -def*
 $\psi' b$ - b)
have γ -*trunc*: $\langle \gamma F = \text{trunc-ell2 } (UNIV \times F) \psi \rangle$ **if** $\langle \text{finite } F \rangle$ **for** F
proof (*rule cinner-ket-eqI*)
fix $x :: \langle 'b \times 'c \rangle$ **obtain** $x1\ x2$ **where** x -*def*: $\langle x = (x1, x2) \rangle$
by *force*
have $*$: $\langle \text{ket } x \cdot_C ((\text{id-cblinfun } \otimes_o \text{butterfly } (\text{ket } j) (\text{ket } j)) *_V \psi) = \text{of-bool } (j=x2) *$
Rep-ell2 $\psi\ x$ **for** j
apply (*simp add: x -def tensor-op-ell2 tensor-op-adjoint cinner-ket*
flip: tensor-ell2-ket cinner-adj-left)
by (*simp add: tensor-ell2-ket cinner-ket-left*)
have $\langle \text{ket } x \cdot_C \gamma F = \text{of-bool } (x2 \in F) *_C \text{Rep-ell2 } \psi\ x \rangle$
using *that*
apply (*simp add: x -def γ -def complex-vector.linear-sum[of $\langle \text{cinner } - \rangle$] bounded-clinear-cinner-right*
bounded-clinear.clinear sum-single[where $i=x2$] tensor-op-adjoint tensor-op-ell2
cinner-ket
flip: tensor-ell2-ket cinner-adj-left)
by (*simp add: tensor-ell2-ket cinner-ket-left*)
moreover **have** $\langle \text{ket } x \cdot_C \text{trunc-ell2 } (UNIV \times F) \psi = \text{of-bool } (x2 \in F) *_C \text{Rep-ell2 } \psi\ x \rangle$
by (*simp add: trunc-ell2.rep-eq cinner-ket-left x -def*)
ultimately **show** $\langle \text{ket } x \cdot_C \gamma F = \text{ket } x \cdot_C \text{trunc-ell2 } (UNIV \times F) \psi \rangle$
by *simp*
qed
have $\langle \gamma \longrightarrow \psi \rangle$ (*finite-subsets-at-top UNIV*)
proof (*rule tendsto-iff[THEN iffD2, rule-format]*)
fix $e :: \text{real}$ **assume** $\langle e > 0 \rangle$
from *trunc-ell2-lim-at-UNIV*[of ψ]
have $\langle \forall F\ F \text{ in finite-subsets-at-top } UNIV. \text{dist } (\text{trunc-ell2 } F \psi) \psi < e \rangle$
by (*simp add: $\langle 0 < e \rangle$ tendstoD*)
then **obtain** M **where** $\langle \text{finite } M \rangle$ **and** *less-e*: $\langle \text{finite } F \implies F \supseteq M \implies \text{dist } (\text{trunc-ell2}$
 $F \psi) \psi < e \rangle$ **for** F
by (*metis (mono-tags, lifting) eventually-finite-subsets-at-top subset-UNIV*)
define M' **where** $\langle M' = \text{snd } ' M \rangle$
have $\langle \text{finite } M' \rangle$
using M' -*def* $\langle \text{finite } M \rangle$ **by** *blast*
have $\langle \text{dist } (\gamma F') \psi < e \rangle$ **if** $\langle \text{finite } F' \rangle$ **and** $\langle F' \supseteq M' \rangle$ **for** F'
proof –
have $\langle \text{dist } (\gamma F') \psi = \text{norm } (\text{trunc-ell2 } (- (UNIV \times F')) \psi) \rangle$
using *that* **by** (*simp only: γ -trunc dist-norm trunc-ell2-uminus norm-minus-commute*)
also **have** $\langle \dots \leq \text{norm } (\text{trunc-ell2 } (- ((\text{fst } ' M) \times F')) \psi) \rangle$
by (*meson Compl-anti-mono Set.basic-monos(1) Sigma-mono subset-UNIV trunc-ell2-norm-mono*)
also **have** $\langle \dots = \text{dist } (\text{trunc-ell2 } ((\text{fst } ' M) \times F') \psi) \psi \rangle$
apply (*simp add: trunc-ell2-uminus dist-norm*)

```

    using norm-minus-commute by blast
  also have ⟨... < e⟩
    apply (rule less-e)
    subgoal
      using ⟨finite F'⟩ ⟨finite M⟩ by force
    subgoal
      using ⟨F' ⊇ M'⟩ M'-def by force
    done
  finally show ?thesis
    by -
  qed
  then show ⟨∀ F F' in finite-subsets-at-top UNIV. dist (γ F') ψ < e⟩
    using ⟨finite M'⟩ by (auto simp add: eventually-finite-subsets-at-top)
  qed
  then show ⟨ψ ∈ ?rhs⟩
    by (rule Lim-in-closed-set[rotated -1]) (use γ-rhs in auto)
  qed
  also from * have ⟨... ⊆ space-as-set ((∏ x∈X. S x) ⊗S ?top)⟩
    by (meson ccspace-leqI image-subset-iff less-eq-ccsubspace.rep-eq)

  finally show ⟨ψ ∈ space-as-set ((∏ x∈X. S x) ⊗S ?top)⟩
    by -
  qed
qed

```

lemma tensor-ccsubspace-INF-right-top:
fixes $S :: 'a \Rightarrow 'b \text{ ell2 ccspace}$
shows $\langle (\top :: 'c \text{ ell2 ccspace}) \otimes_S (\text{INF } x \in X. S x) = (\text{INF } x \in X. \top \otimes_S S x) \rangle$
proof -
have $\langle (\text{INF } x \in X. S x) \otimes_S (\top :: 'c \text{ ell2 ccspace}) = (\text{INF } x \in X. S x \otimes_S \top) \rangle$
by (rule tensor-ccsubspace-INF-left-top)
then have $\langle \text{swap-ell2 } * _S ((\text{INF } x \in X. S x) \otimes_S (\top :: 'c \text{ ell2 ccspace})) = \text{swap-ell2 } * _S (\text{INF } x \in X. S x \otimes_S \top) \rangle$
by simp
then show ?thesis
by (cases ⟨X = {}⟩)
 (simp-all add: swap-ell2-tensor-ccsubspace)
qed

lemma tensor-ccsubspace-INF-left: $\langle (\text{INF } x \in X. S x) \otimes_S T = (\text{INF } x \in X. S x \otimes_S T) \rangle$ **if** $\langle X \neq \{\} \rangle$
proof (cases ⟨T=0⟩)
case True
then show ?thesis
 using that by simp
next
case False
from ccspace-as-whole-type[OF False]
have ⟨let 't::type = some-onb-of T in

$(\text{INF } x \in X. S x) \otimes_S T = (\text{INF } x \in X. S x \otimes_S T)$
proof *with-type-mp*
with-type-case
from *with-type-mp.premise*
obtain $U :: \langle 't \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$ **where** [*simp*]: $\langle \text{isometry } U \rangle$ **and** $imU: \langle U *_S \top = T \rangle$
by *auto*
have $\langle (id\text{-cblinfun } \otimes_o U) *_S ((\prod x \in X. S x) \otimes_S \top) = (id\text{-cblinfun } \otimes_o U) *_S (\prod x \in X. S x \otimes_S \top) \rangle$
by (*rule arg-cong*[**where** $f = \langle \lambda x. - *_S x \rangle$], *rule tensor-ccsubspace-INF-left-top*)
then show $\langle (\prod x \in X. S x) \otimes_S T = (\prod x \in X. S x \otimes_S T) \rangle$
using that by (*simp add: imU flip: tensor-ccsubspace-image*)
qed
from *this*[*cancel-with-type*]
show *?thesis*
by –
qed

lemma *tensor-ccsubspace-INF-right*: $\langle (\text{INF } x \in X. T \otimes_S S x) = (\text{INF } x \in X. T \otimes_S S x) \rangle$ **if** $\langle X \neq \{\} \rangle$
proof –
from that have $\langle (\text{INF } x \in X. S x) \otimes_S T = (\text{INF } x \in X. S x \otimes_S T) \rangle$
by (*rule tensor-ccsubspace-INF-left*)
then have $\langle \text{swap-ell2} *_S ((\text{INF } x \in X. S x) \otimes_S T) = \text{swap-ell2} *_S (\text{INF } x \in X. S x \otimes_S T) \rangle$
by *simp*
then show *?thesis*
by (*cases* $\langle X = \{\} \rangle$)
(simp-all add: swap-ell2-tensor-ccsubspace)
qed

lemma *tensor-ccsubspace-ccspan*: $\langle \text{ccspan } X \otimes_S \text{ccspan } Y = \text{ccspan } \{x \otimes_s y \mid x y. x \in X \wedge y \in Y\} \rangle$
proof (*rule antisym*)
show $\langle \text{ccspan } \{x \otimes_s y \mid x y. x \in X \wedge y \in Y\} \leq \text{ccspan } X \otimes_S \text{ccspan } Y \rangle$
using *ccspan-superset*[*of X*] *ccspan-superset*[*of Y*]
by (*auto intro!*: *ccspan-mono Collect-mono ex-mono simp add: tensor-ccsubspace-def*)
next
have $\langle \{\psi \otimes_s \varphi \mid \psi \varphi. \psi \in \text{space-as-set } (\text{ccspan } X) \wedge \varphi \in \text{space-as-set } (\text{ccspan } Y)\} \subseteq \text{closure } \{x \otimes_s y \mid x y. x \in \text{cspan } X \wedge y \in \text{cspan } Y\} \rangle$
proof (*rule subsetI*)
fix γ
assume $\langle \gamma \in \{\psi \otimes_s \varphi \mid \psi \varphi. \psi \in \text{space-as-set } (\text{ccspan } X) \wedge \varphi \in \text{space-as-set } (\text{ccspan } Y)\} \rangle$
then obtain $\psi \varphi$ **where** $\psi: \langle \psi \in \text{space-as-set } (\text{ccspan } X) \rangle$ **and** $\varphi: \langle \varphi \in \text{space-as-set } (\text{ccspan } Y) \rangle$ **and** $\gamma\text{-def}: \langle \gamma = \psi \otimes_s \varphi \rangle$
by *blast*
from ψ
obtain ψ' **where** *lim1*: $\langle \psi' \longrightarrow \psi \rangle$ **and** $\psi'X: \langle \psi' n \in \text{cspan } X \rangle$ **for** n
using *closure-sequential unfolding ccspan.rep-eq* **by** *blast*
from φ
obtain φ' **where** *lim2*: $\langle \varphi' \longrightarrow \varphi \rangle$ **and** $\varphi'Y: \langle \varphi' n \in \text{cspan } Y \rangle$ **for** n

```

    using closure-sequential unfolding ccspan.rep-eq by blast
  interpret tensor: bounded-cbilinear tensor-ell2
  by (rule bounded-cbilinear-tensor-ell2)
  from lim1 lim2 have ⟨λn. ψ' n ⊗s φ' n ⟶ ψ ⊗s φ⟩
  by (rule tensor.tendsto)
  moreover have ⟨ψ' n ⊗s φ' n ∈ {x ⊗s y | x y. x ∈ cspan X ∧ y ∈ cspan Y}⟩ for n
  using ψ'X φ'Y by auto
  ultimately show ⟨γ ∈ closure {x ⊗s y | x y. x ∈ cspan X ∧ y ∈ cspan Y}⟩
  unfolding γ-def
  by (meson closure-sequential)
qed
also have ⟨closure {x ⊗s y | x y. x ∈ cspan X ∧ y ∈ cspan Y}
  ⊆ closure (cspan {x ⊗s y | x y. x ∈ X ∧ y ∈ Y})⟩
proof (intro closure-mono subsetI)
  fix γ
  assume ⟨γ ∈ {x ⊗s y | x y. x ∈ cspan X ∧ y ∈ cspan Y}⟩
  then obtain x y where γ-def: ⟨γ = x ⊗s y⟩ and ⟨x ∈ cspan X⟩ and ⟨y ∈ cspan Y⟩
  by blast
  from ⟨x ∈ cspan X⟩
  obtain X' x' where ⟨finite X'⟩ and ⟨X' ⊆ X⟩ and x-def: ⟨x = (∑ i ∈ X'. x' i *C i)⟩
  using complex-vector.span-explicit[of X] by auto
  from ⟨y ∈ cspan Y⟩
  obtain Y' y' where ⟨finite Y'⟩ and ⟨Y' ⊆ Y⟩ and y-def: ⟨y = (∑ j ∈ Y'. y' j *C j)⟩
  using complex-vector.span-explicit[of Y] by auto
  from x-def y-def γ-def
  have ⟨γ = (∑ i ∈ X'. x' i *C i) ⊗s (∑ j ∈ Y'. y' j *C j)⟩
  by simp
  also have ⟨... = (∑ i ∈ X'. ∑ j ∈ Y'. (x' i *C i) ⊗s (y' j *C j))⟩
  by (smt (verit) sum.cong tensor-ell2-sum-left tensor-ell2-sum-right)
  also have ⟨... = (∑ i ∈ X'. ∑ j ∈ Y'. (x' i * y' j) *C (i ⊗s j))⟩
  by (metis (no-types, lifting) scaleC-scaleC sum.cong tensor-ell2-scaleC1 tensor-ell2-scaleC2)
  also have ⟨... ∈ cspan {x ⊗s y | x y. x ∈ X ∧ y ∈ Y}⟩
  using ⟨X' ⊆ X⟩ ⟨Y' ⊆ Y⟩
  by (auto intro!: complex-vector.span-sum complex-vector.span-scale
    complex-vector.span-base[of ⟨- ⊗s -⟩])
  finally show ⟨γ ∈ cspan {x ⊗s y | x y. x ∈ X ∧ y ∈ Y}⟩
  by -
qed
also have ⟨... = space-as-set (ccspan {x ⊗s y | x y. x ∈ X ∧ y ∈ Y})⟩
  using ccspan.rep-eq by blast
  finally show ⟨ccspan X ⊗S ccspan Y ≤ ccspan {x ⊗s y | x y. x ∈ X ∧ y ∈ Y}⟩
  by (auto intro!: ccspan-leqI simp add: tensor-ccsubspace-def)
qed

lemma tensor-ccsubspace-mono: ⟨A ⊗S B ≤ C ⊗S D⟩ if ⟨A ≤ C⟩ and ⟨B ≤ D⟩
  apply (auto intro!: ccspan-mono simp add: tensor-ccsubspace-def)
  using that
  by (auto simp add: less-eq-ccsubspace-def)

```

lemma *tensor-ccsubspace-element-as-infsum*:
fixes $A :: \langle 'a \text{ ell2 ccspace} \rangle$ **and** $B :: \langle 'b \text{ ell2 ccspace} \rangle$
assumes $\langle \psi \in \text{space-as-set } (A \otimes_S B) \rangle$
shows $\langle \exists \varphi \delta. (\forall n::\text{nat}. \varphi n \in \text{space-as-set } A) \wedge (\forall n. \delta n \in \text{space-as-set } B)$
 $\wedge ((\lambda n. \varphi n \otimes_s \delta n) \text{ has-sum } \psi) \text{ UNIV} \rangle$
proof –
obtain A' **where** $\text{span}A' : \langle \text{ccspan } A' = A \rangle$ **and** $\text{ortho}A' : \langle \text{is-ortho-set } A' \rangle$ **and** $\text{norm}A' : \langle a \in A' \implies \text{norm } a = 1 \rangle$ **for** a
using *some-onb-of-ccspan some-onb-of-is-ortho-set some-onb-of-norm1*
by *blast*
obtain B' **where** $\text{span}B' : \langle \text{ccspan } B' = B \rangle$ **and** $\text{ortho}B' : \langle \text{is-ortho-set } B' \rangle$ **and** $\text{norm}B' : \langle b \in B' \implies \text{norm } b = 1 \rangle$ **for** b
using *some-onb-of-ccspan some-onb-of-is-ortho-set some-onb-of-norm1*
by *blast*
define AB' **where** $\langle AB' = \{a \otimes_s b \mid a b. a \in A' \wedge b \in B'\} \rangle$
define $ABnon0$ **where** $\langle ABnon0 = \{ab \in AB'. (ab \cdot_C \psi) *_C ab \neq 0\} \rangle$
have $ABnon0\text{-def}' : \langle ABnon0 = \{ab \in AB'. (\text{norm } (ab \cdot_C \psi))^2 \neq 0\} \rangle$
by *(auto simp: ABnon0-def)*
have $\langle \text{is-ortho-set } AB' \rangle$
by *(simp add: AB'-def orthoA' orthoB' tensor-ell2-is-ortho-set)*
have $\text{norm}AB' : \langle ab \in AB' \implies \text{norm } ab = 1 \rangle$ **for** ab
by *(auto simp add: AB'-def norm-tensor-ell2 normA' normB')*
have $\text{span}AB' : \langle \text{ccspan } AB' = A \otimes_S B \rangle$
by *(simp add: tensor-ccsubspace-ccspan AB'-def flip: spanA' spanB')*
have $\text{sum1} : \langle ((\lambda ab. (ab \cdot_C \psi) *_C ab) \text{ has-sum } \psi) AB' \rangle$
apply *(rule basis-projections-reconstruct-has-sum)*
by *(simp-all add: spanAB' is-ortho-set AB' normAB' assms)*
have $\langle (\lambda ab. (\text{norm } (ab \cdot_C \psi))^2) \text{ summable-on } AB' \rangle$
by *(rule parseval-identity-summable)*
(simp-all add: spanAB' is-ortho-set AB' normAB' assms)
then have $\langle \text{countable } ABnon0 \rangle$
using $ABnon0\text{-def}'$ *summable-countable-real* **by** *blast*
obtain f **and** $N :: \langle \text{nat set} \rangle$ **where** $\text{bij-}f : \langle \text{bij-betw } f N ABnon0 \rangle$
using $\langle \text{countable } ABnon0 \rangle$ *countableE-bij* **by** *blast*
then obtain $\varphi 0 \delta 0$ **where** $f\text{-def} : \langle f n = \varphi 0 n \otimes_s \delta 0 n \rangle$ **and** $\varphi 0A' : \langle \varphi 0 n \in A' \rangle$ **and** $\delta 0B' : \langle \delta 0 n \in B' \rangle$ **if** $\langle n \in N \rangle$ **for** n
apply *atomize-elim*
apply *(subst all-conj-distrib[symmetric] choice-iff[symmetric])+*
apply *(simp add: bij-betw-def ABnon0-def)*
using $AB'\text{-def}$ $\langle \text{bij-betw } f N ABnon0 \rangle$ *bij-betwE mem-Collect-eq* **by** *blast*
define c **where** $\langle c n = (\varphi 0 n \otimes_s \delta 0 n) \cdot_C \psi \rangle$ **for** n
from sum1 **have** $\langle ((\lambda ab. (ab \cdot_C \psi) *_C ab) \text{ has-sum } \psi) ABnon0 \rangle$
by *(rule has-sum-cong-neutral[THEN iffD1, rotated -1]) (auto simp: ABnon0-def)*
then have $\langle ((\lambda n. (f n \cdot_C \psi) *_C f n) \text{ has-sum } \psi) N \rangle$
by *(rule has-sum-reindex-bij-betw[OF bij-f, THEN iffD2])*
then have $\text{sum2} : \langle ((\lambda n. c n *_C (\varphi 0 n \otimes_s \delta 0 n)) \text{ has-sum } \psi) N \rangle$
by *(rule has-sum-cong[THEN iffD1, rotated]) (simp add: f-def c-def)*
define $\varphi \delta$ **where** $\langle \varphi n = (\text{if } n \in N \text{ then } c n *_C \varphi 0 n \text{ else } 0) \rangle$ **and** $\langle \delta n = (\text{if } n \in N \text{ then } \delta 0 n \text{ else } 0) \rangle$ **for** n

then have 1: $\langle \varphi \ n \in \text{space-as-set } A \rangle$ **and** 2: $\langle \delta \ n \in \text{space-as-set } B \rangle$ **for** n
using $\varphi 0A' \ \delta 0B' \ \text{span}A' \ \text{span}B' \ \text{ccspan-superset}$
by (*auto intro!*: *complex-vector.subspace-scale simp*: $\varphi\text{-def}$ $\delta\text{-def}$)
from sum2 **have** sum3 : $\langle ((\lambda n. \varphi \ n \otimes_s \delta \ n) \text{ has-sum } \psi) \text{ UNIV} \rangle$
by (*rule has-sum-cong-neutral*[*THEN iffD2, rotated -1*])
*(auto simp: $\varphi\text{-def}$ $\delta\text{-def}$ *tensor-ell2-scaleC1*)*
from 1 2 sum3 **show** *?thesis*
by *auto*
qed

lemma *ortho-tensor-ccsubspace-right*: $\langle - \ (\top \otimes_S A) = \top \otimes_S (- A) \rangle$

proof –

have [*simp*]: $\langle \text{is-Proj } (id\text{-cblinfun} \otimes_o \text{Proj } X) \rangle$ **for** X
by (*metis Proj-is-Proj Proj-top is-Proj-tensor-op*)

have $\langle \text{Proj } (- \ (\top \otimes_S A)) = id\text{-cblinfun} - \text{Proj } (\top \otimes_S A) \rangle$

by (*simp add: Proj-ortho-compl*)

also have $\langle \dots = id\text{-cblinfun} - (id\text{-cblinfun} \otimes_o \text{Proj } A) \rangle$

by (*simp add: tensor-ccsubspace-via-Proj Proj-on-own-range*)

also have $\langle \dots = id\text{-cblinfun} \otimes_o (id\text{-cblinfun} - \text{Proj } A) \rangle$

by (*metis cblinfun.diff-right left-amplification.rep-eq tensor-id*)

also have $\langle \dots = \text{Proj } \top \otimes_o \text{Proj } (- A) \rangle$

by (*simp add: Proj-ortho-compl*)

also have $\langle \dots = \text{Proj } (\top \otimes_S (- A)) \rangle$

by (*simp add: tensor-ccsubspace-via-Proj Proj-on-own-range*)

finally show *?thesis*

using *Proj-inj* **by** *blast*

qed

lemma *ortho-tensor-ccsubspace-left*: $\langle - \ (A \otimes_S \top) = (- A) \otimes_S \top \rangle$

proof –

have $\langle - \ (A \otimes_S \top) = \text{swap-ell2} \ *_S \ (- \ (\top \otimes_S A)) \rangle$

by (*simp add: unitary-image-ortho-compl swap-ell2-tensor-ccsubspace*)

also have $\langle \dots = \text{swap-ell2} \ *_S \ (\top \otimes_S (- A)) \rangle$

by (*simp add: ortho-tensor-ccsubspace-right*)

also have $\langle \dots = (- A) \otimes_S \top \rangle$

by (*simp add: swap-ell2-tensor-ccsubspace*)

finally show *?thesis*

by –

qed

lemma *kernel-tensor-id-left*: $\langle \text{kernel } (id\text{-cblinfun} \otimes_o A) = \top \otimes_S \text{kernel } A \rangle$

proof –

have $\langle \text{kernel } (id\text{-cblinfun} \otimes_o A) = - \ ((id\text{-cblinfun} \otimes_o A) \ * \ * \ * \ \top) \rangle$

by (*rule kernel-compl-adj-range*)

also have $\langle \dots = - \ (id\text{-cblinfun} \ *_S \ \top \otimes_S A \ * \ * \ \top) \rangle$

by (*metis cblinfun-image-id id-cblinfun-adjoint tensor-ccsubspace-image tensor-ccsubspace-top tensor-op-adjoint*)

also have $\langle \dots = \top \otimes_S (- \ (A \ * \ * \ \top)) \rangle$

by (simp add: ortho-tensor-ccsubspace-right)
 also have $\langle \dots = \mathbb{T} \otimes_S \text{kernel } A \rangle$
 by (simp add: kernel-compl-adj-range)
 finally show ?thesis
 by –
 qed

lemma kernel-tensor-id-right: $\langle \text{kernel } (A \otimes_o \text{id-cblinfun}) = \text{kernel } A \otimes_S \mathbb{T} \rangle$
proof –
 have ker-swap: $\langle \text{kernel } \text{swap-ell2} = 0 \rangle$
 by (simp add: kernel-isometry)
 have $\langle \text{kernel } (\text{id-cblinfun} \otimes_o A) = \mathbb{T} \otimes_S \text{kernel } A \rangle$
 by (rule kernel-tensor-id-left)
 from this[THEN arg-cong, of $\langle \text{cblinfun-image } \text{swap-ell2} \rangle$]
 show ?thesis
 by (simp add: swap-ell2-tensor-ccsubspace cblinfun-image-kernel-unitary
 flip: swap-ell2-commute-tensor-op kernel-cblinfun-compose[OF ker-swap])
 qed

lemma eigenspace-tensor-id-left: $\langle \text{eigenspace } c (\text{id-cblinfun} \otimes_o A) = \mathbb{T} \otimes_S \text{eigenspace } c A \rangle$
proof –
 have $\langle \text{eigenspace } c (\text{id-cblinfun} \otimes_o A) = \text{kernel } (\text{id-cblinfun} \otimes_o (A - c *_C \text{id-cblinfun})) \rangle$
 unfolding eigenspace-def
 by (metis (no-types, lifting) complex-vector.scale-minus-left tensor-id tensor-op-right-add
 tensor-op-scaleC-right uminus-add-conv-diff)
 also have $\langle \text{kernel } (\text{id-cblinfun} \otimes_o (A - c *_C \text{id-cblinfun})) = \mathbb{T} \otimes_S \text{kernel } (A - c *_C \text{id-cblinfun}) \rangle$
 by (simp add: kernel-tensor-id-left)
 also have $\langle \dots = \mathbb{T} \otimes_S \text{eigenspace } c A \rangle$
 by (simp add: eigenspace-def)
 finally show ?thesis
 by –
 qed

lemma eigenspace-tensor-id-right: $\langle \text{eigenspace } c (A \otimes_o \text{id-cblinfun}) = \text{eigenspace } c A \otimes_S \mathbb{T} \rangle$
proof –
 have $\langle \text{eigenspace } c (\text{id-cblinfun} \otimes_o A) = \mathbb{T} \otimes_S \text{eigenspace } c A \rangle$
 by (rule eigenspace-tensor-id-left)
 from this[THEN arg-cong, of $\langle \text{cblinfun-image } \text{swap-ell2} \rangle$]
 show ?thesis
 by (simp add: swap-ell2-commute-tensor-op cblinfun-image-eigenspace-unitary swap-ell2-tensor-ccsubspace)
 qed

unbundle no cblinfun-syntax

end

15 Partial-Trace – The partial trace

theory *Partial-Trace*

imports *Trace-Class Hilbert-Space-Tensor-Product*

begin

unbundle *cblinfun-syntax*

hide-fact (open) *Infinite-Set-Sum.abs-summable-on-Sigma-iff*

hide-fact (open) *Infinite-Set-Sum.abs-summable-on-comparison-test*

hide-const (open) *Determinants.trace*

hide-fact (open) *Determinants.trace-def*

definition *partial-trace* :: $\langle ('a \times 'c) \text{ ell2}, ('b \times 'c) \text{ ell2} \rangle \text{ trace-class} \Rightarrow ('a \text{ ell2}, 'b \text{ ell2}) \text{ trace-class}$ **where**

$\langle \text{partial-trace } t = (\sum_{\infty} j. \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) t) (\text{tensor-ell2-right } (\text{ket } j))) \rangle$

lemma *partial-trace-def'*: $\langle \text{partial-trace } t = (\sum_{\infty} j. \text{sandwich-tc } ((\text{tensor-ell2-right } (\text{ket } j))^*) t) \rangle$

— We cannot use this as the definition of *partial-trace* because this definition has a more restricted type (*t* is a square operator).

by (*auto intro!*: *simp: partial-trace-def sandwich-tc-def*)

lemma *partial-trace-abs-summable*:

$\langle (\lambda j. \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) t) (\text{tensor-ell2-right } (\text{ket } j))) \text{ abs-summable-on } UNIV \rangle$

and *partial-trace-has-sum*:

$\langle ((\lambda j. \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) t) (\text{tensor-ell2-right } (\text{ket } j))) \text{ has-sum } \text{partial-trace } t) UNIV \rangle$

and *partial-trace-norm-reducing*: $\langle \text{norm } (\text{partial-trace } t) \leq \text{norm } t \rangle$

proof –

define *t'* **where** $\langle t' = \text{from-trace-class } t \rangle$

define *s* **where** $\langle s \ k = \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } k))^*) t) (\text{tensor-ell2-right } (\text{ket } k)) \rangle$ **for** *k*

have *bound*: $\langle (\sum_{k \in F} \text{norm } (s \ k)) \leq \text{norm } t \rangle$

if $\langle F \in \{F. F \subseteq UNIV \wedge \text{finite } F\} \rangle$

for *F* :: $\langle 'a \text{ set} \rangle$

proof –

from *that* **have** [*simp*]: $\langle \text{finite } F \rangle$

by *force*

define *tk* **where** $\langle tk \ k = \text{tensor-ell2-right } (\text{ket } k)^* \circ_{CL} t' \circ_{CL} \text{tensor-ell2-right } (\text{ket } k) \rangle$ **for** *k*

have *tc-t'* [*simp*]: $\langle \text{trace-class } t' \rangle$

by (*simp add*: *t'-def*)

then **have** *tc-tk* [*simp*]: $\langle \text{trace-class } (tk \ k) \rangle$ **for** *k*

by (*simp add*: *tk-def trace-class-comp-left trace-class-comp-right*)

define *uk* **where** $\langle uk \ k = (\text{polar-decomposition } (tk \ k))^* \rangle$ **for** *k*

define *u* **where** $\langle u = (\sum_{k \in F} uk \ k \otimes_o \text{butterfly } (\text{ket } k) (\text{ket } k)) \rangle$

define *B* :: $\langle 'b \text{ ell2 set} \rangle$ **where** $\langle B = \text{range } ket \rangle$

```

have aux1: ⟨tensor-ell2-right (ket x)* *V u *V a = 0⟩ if ⟨x ∉ F⟩ for x a
proof –
  have *: ⟨u* oCL tensor-ell2-right (ket x) = 0⟩
    by (auto intro!: equal-ket simp: u-def sum-adj tensor-op-adjoint tensor-ell2-right-apply
      cblinfun.sum-left tensor-op-ell2 cinner-ket sum-single[where i=x] ⟨x ∉ F⟩)
  have ⟨tensor-ell2-right (ket x)* oCL u = 0⟩
    by (rule adj-inject[THEN iffD1]) (use * in simp)
  then show ?thesis
    by (simp flip: cblinfun-apply-cblinfun-compose)
qed

have aux2: ⟨uk x *V tensor-ell2-right (ket x)* *V a = tensor-ell2-right (ket x)* *V u *V a⟩
if ⟨x ∈ F⟩ for x a
proof –
  have *: ⟨tensor-ell2-right (ket x) oCL (uk x)* = u* oCL tensor-ell2-right (ket x)⟩
    by (auto intro!: equal-ket simp: u-def sum-adj tensor-op-adjoint tensor-ell2-right-apply
      cblinfun.sum-left tensor-op-ell2 ⟨x ∈ F⟩ cinner-ket sum-single[where i=x])
  have ⟨uk x oCL tensor-ell2-right (ket x)* = tensor-ell2-right (ket x)* oCL u⟩
    by (rule adj-inject[THEN iffD1]) (use * in simp)
  then show ?thesis
    by (simp flip: cblinfun-apply-cblinfun-compose)
qed

have sum1: ⟨(λ(x, y). ket (y, x) •C (u *V t' *V ket (y, x))) summable-on UNIV⟩
proof –
  have ⟨trace-class (u oCL t')⟩
    by (simp add: trace-class-comp-right)
  then have ⟨(λyx. yx •C ((u oCL t') *V yx)) summable-on (range ket)⟩
    using is-onb-ket trace-exists by blast
  then have ⟨(λyx. ket yx •C ((u oCL t') *V ket yx)) summable-on UNIV⟩
    apply (subst summable-on-reindex-bij-betw[where g=ket and A=UNIV and B=⟨range
ket⟩])
    using bij-betw-def inj-ket by blast
  then show ?thesis
    by (subst summable-on-reindex-bij-betw[where g=prod.swap and A=UNIV, symmetric])
auto
qed

have norm-u: ⟨norm u ≤ 1⟩
proof –
  define u2 uk2 where ⟨u2 = u* oCL u⟩ and ⟨uk2 k = (uk k)* oCL uk k⟩ for k
  have *: ⟨(∑ i∈F. (uk i* oCL uk k) ⊗o (ket i •C ket k) *C butterfly (ket i) (ket k))
    = (uk k* oCL uk k) ⊗o butterfly (ket k) (ket k)⟩ if [simp]: ⟨k ∈ F⟩ for k
    apply (subst sum-single[where i=k])
    by (auto simp: cinner-ket)
  have **: ⟨(∑ ka∈F. (uk2 ka oCL uk2 k) ⊗o (ket ka •C ket k) *C butterfly (ket ka) (ket k))
    = (uk2 k oCL uk2 k) ⊗o butterfly (ket k) (ket k)⟩ if [simp]: ⟨k ∈ F⟩ for k
    apply (subst sum-single[where i=k])

```

```

  by (auto simp: cinner-ket)
have proj-uk2: ⟨is-Proj (uk2 k)⟩ for k
  unfolding uk2-def
  apply (rule partial-isometry-square-proj)
  by (auto intro!: partial-isometry-square-proj partial-isometry-adj simp: uk-def)
have u2-explicit: ⟨u2 = (∑ k∈F. uk2 k ⊗o butterfly (ket k) (ket k))⟩
  by (simp add: u2-def u-def sum-adj tensor-op-adjoint cblinfun-compose-sum-right
    cblinfun-compose-sum-left tensor-butter comp-tensor-op * uk2-def)
have ⟨u2* = u2⟩
  by (simp add: u2-def)
moreover have ⟨u2 oCL u2 = u2⟩
  by (simp add: u2-explicit cblinfun-compose-sum-right cblinfun-compose-sum-left
    comp-tensor-op ** proj-uk2 is-Proj-idempotent)
ultimately have ⟨is-Proj u2⟩
  by (simp add: is-Proj-I)
then have ⟨norm u2 ≤ 1⟩
  using norm-is-Proj by blast
then show ⟨norm u ≤ 1⟩
  by (simp add: power-le-one-iff norm-AAadj u2-def)
qed

have ⟨(∑ k∈F. norm (s k))
  = (∑ k∈F. trace-norm (tk k))⟩
  by (simp add: s-def tk-def norm-trace-class.rep-eq compose-tcl.rep-eq compose-tcr.rep-eq
    t'-def)
  also have ⟨... = cmod (∑ k∈F. trace (uk k oCL tk k))⟩
  by (smt (verit, best) norm-of-real of-real-hom.hom-sum polar-decomposition-correct' sum.cong
    sum-nonneg trace-abs-op trace-norm-nneg uk-def)
  also have ⟨... = cmod (∑ k∈F. trace (tensor-ell2-right (ket k)* oCL u oCL t' oCL tensor-ell2-right (ket k)))⟩
  apply (rule arg-cong[where f=cmod], rule sum.cong[OF refl], rule arg-cong[where f=trace])
  by (auto intro!: equal-ket simp: tk-def aux2)
  also have ⟨... = cmod (∑ k∈F. ∑∞j. ket j •C ((tensor-ell2-right (ket k)* oCL u oCL t' oCL tensor-ell2-right (ket k)) *V ket j))⟩
  by (auto intro!: sum.cong simp: trace-ket-sum trace-class-comp-left trace-class-comp-right)
  also have ⟨... = cmod (∑∞k∈F. ∑∞j. ket j •C ((tensor-ell2-right (ket k)* oCL u oCL t' oCL tensor-ell2-right (ket k)) *V ket j))⟩
  by (simp add: ⟨finite F⟩)
  also have ⟨... = cmod (∑∞k. ∑∞j. ket j •C ((tensor-ell2-right (ket k)* oCL u oCL t' oCL tensor-ell2-right (ket k)) *V ket j))⟩
  apply (rule arg-cong[where f=cmod])
  apply (rule infsum-cong-neutral)
  by (auto simp: aux1)
  also have ⟨... = cmod (∑∞k. ∑∞j. ket (j,k) •C ((u oCL t') *V ket (j,k)))⟩
  apply (rule arg-cong[where f=cmod], rule infsum-cong, rule infsum-cong)
  by (simp add: tensor-ell2-right-apply cinner-adj-right tensor-ell2-ket)
  also have ⟨... = cmod (∑∞(k,j). ket (j,k) •C ((u oCL t') *V ket (j,k)))⟩
  apply (rule arg-cong[where f=cmod])
  apply (subst infsum-Sigma'-banach)

```



```

    using sum1 by auto
  also have ⟨... = cmod (∑∞jk. ket jk ·C ((u oCL t') *V ket jk))⟩
    apply (subst infsum-reindex-bij-betw[where g=prod.swap and A=UNIV, symmetric])
    by auto
  also have ⟨... = cmod (trace (u oCL t'))⟩
    by (simp add: trace-ket-sum trace-class-comp-right)
  also have ⟨... ≤ trace-norm (u oCL t')⟩
    using trace-leq-trace-norm by blast
  also have ⟨... ≤ norm u * trace-norm t'⟩
    by (simp add: trace-norm-comp-right)
  also have ⟨... ≤ trace-norm t'⟩
    using norm-u
    by (metis more-arith-simps(5) mult-right-mono trace-norm-nneg)
  also have ⟨... = norm t⟩
    by (simp add: norm-trace-class.rep-eq t'-def)
  finally show ⟨(∑k∈F. norm (s k)) ≤ norm t⟩
    by -
qed

show abs-summable: ⟨s abs-summable-on UNIV⟩
  by (intro nonneg-bdd-above-summable-on bdd-aboveI2[where M=⟨norm t⟩] norm-ge-zero
bound)

from abs-summable
show has-sum: ⟨(s has-sum partial-trace t) UNIV⟩
  by (simp add: abs-summable-summable partial-trace-def s-def[abs-def] t'-def)

show ⟨norm (partial-trace t) ≤ norm t⟩
proof -
  have ⟨norm (partial-trace t) ≤ (∑∞k. norm (s k))⟩
    using - has-sum apply (rule norm-has-sum-bound)
    using abs-summable has-sum-infsum by blast
  also from bound have ⟨(∑∞k. norm (s k)) ≤ norm t⟩
    by (simp add: abs-summable infsum-le-finite-sums)
  finally show ?thesis
    by -
qed
qed

lemma partial-trace-abs-summable':
  ⟨(λj. sandwich-tc ((tensor-ell2-right (ket j))* t) abs-summable-on UNIV)
and partial-trace-has-sum':
  ⟨((λj. sandwich-tc ((tensor-ell2-right (ket j))* t) has-sum partial-trace t) UNIV)
using partial-trace-abs-summable partial-trace-has-sum
by (auto intro!: simp: sandwich-tc-def sandwich-apply)

```

lemma *trace-partial-trace-compose-eq-trace-compose-tensor-id*:
 $\langle \text{trace } (\text{from-trace-class } (\text{partial-trace } t) \text{ } o_{CL} x) = \text{trace } (\text{from-trace-class } t \text{ } o_{CL} (x \otimes_o \text{id-cblinfun})) \rangle$

proof –
define *s* **where** $\langle s = \text{trace } (\text{from-trace-class } (\text{partial-trace } t) \text{ } o_{CL} x) \rangle$
define *s'* **where** $\langle s' e = \text{ket } e \cdot_C ((\text{from-trace-class } (\text{partial-trace } t) \text{ } o_{CL} x) *_V \text{ket } e) \rangle$ **for** *e*
define *u* **where** $\langle u j = \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) t) (\text{tensor-ell2-right } (\text{ket } j)) \rangle$ **for** *j*
define *u'* **where** $\langle u' e j = \text{ket } e \cdot_C (\text{from-trace-class } (u j) *_V x *_V \text{ket } e) \rangle$ **for** *e j*
have $\langle (u \text{ has-sum } \text{partial-trace } t) \text{ } UNIV \rangle$
using *partial-trace-has-sum*[of *t*]
by (*simp add: u-def*[*abs-def*])
then have $\langle ((\lambda u. \text{from-trace-class } u *_V x *_V \text{ket } e) \text{ } o \text{ } u \text{ has-sum } \text{from-trace-class } (\text{partial-trace } t) *_V x *_V \text{ket } e) \text{ } UNIV \rangle$ **for** *e*
proof (*rule has-sum-comm-additive*[rotated –1])
show $\langle \text{Modules.additive } (\lambda u. \text{from-trace-class } u *_V x *_V \text{ket } e) \rangle$
by (*simp add: Modules.additive-def cblinfun.add-left plus-trace-class.rep-eq*)
have *bounded-clinear*: $\langle \text{bounded-clinear } (\lambda u. \text{from-trace-class } u *_V x *_V \text{ket } e) \rangle$
proof (*rule bounded-clinearI*[**where** $K = \langle \text{norm } (x *_V \text{ket } e) \rangle$])
show $\langle \text{from-trace-class } (b1 + b2) *_V x *_V \text{ket } e = \text{from-trace-class } b1 *_V x *_V \text{ket } e + \text{from-trace-class } b2 *_V x *_V \text{ket } e \rangle$ **for** *b1 b2*
by (*simp add: plus-cblinfun.rep-eq plus-trace-class.rep-eq*)
show $\langle \text{from-trace-class } (r *_C b) *_V x *_V \text{ket } e = r *_C (\text{from-trace-class } b *_V x *_V \text{ket } e) \rangle$
for *b r*
by (*simp add: scaleC-trace-class.rep-eq*)
show $\langle \text{norm } (\text{from-trace-class } t *_V x *_V \text{ket } e) \leq \text{norm } t * \text{norm } (x *_V \text{ket } e) \rangle$ **for** *t*
proof –
have $\langle \text{norm } (\text{from-trace-class } t *_V x *_V \text{ket } e) \leq \text{norm } (\text{from-trace-class } t) * \text{norm } (x *_V \text{ket } e) \rangle$
by (*simp add: norm-cblinfun*)
also have $\langle \dots \leq \text{norm } t * \text{norm } (x *_V \text{ket } e) \rangle$
by (*auto intro!: mult-right-mono simp add: norm-leq-trace-norm norm-trace-class.rep-eq*)
finally show *?thesis*
by –
qed
qed
have $\langle \text{isCont } (\lambda u. \text{from-trace-class } u *_V x *_V \text{ket } e) (\text{partial-trace } t) \rangle$
using *bounded-clinear clinear-continuous-at* **by** *auto*
then show $\langle (\lambda u. \text{from-trace-class } u *_V x *_V \text{ket } e) \text{ } \text{partial-trace } t \rightarrow \text{from-trace-class } (\text{partial-trace } t) *_V x *_V \text{ket } e \rangle$
by (*simp add: isCont-def*)
qed
then have $\langle ((\lambda v. \text{ket } e \cdot_C v) \text{ } o ((\lambda u. \text{from-trace-class } u *_V x *_V \text{ket } e) \text{ } o \text{ } u) \text{ has-sum } \text{ket } e \cdot_C (\text{from-trace-class } (\text{partial-trace } t) *_V x *_V \text{ket } e)) \text{ } UNIV \rangle$ **for** *e*
proof (*rule has-sum-comm-additive*[rotated –1])
show $\langle \text{Modules.additive } (\lambda v. \text{ket } e \cdot_C v) \rangle$
by (*simp add: Modules.additive-def cinner-simps*(2))
have *bounded-clinear*: $\langle \text{bounded-clinear } (\lambda v. \text{ket } e \cdot_C v) \rangle$
using *bounded-clinear-cinner-right* **by** *auto*
then have $\langle \text{isCont } (\lambda v. \text{ket } e \cdot_C v) \text{ } l \rangle$ **for** *l*

by *simp*
then show $\langle (\lambda v. \text{ket } e \cdot_C v) -l \rightarrow \text{ket } e \cdot_C l \rangle$ **for** l
 by (*simp add: isContD*)
qed
then have *has-sum-u'*: $\langle ((\lambda j. u' e j) \text{ has-sum } s' e) \text{ UNIV} \rangle$ **for** e
 by (*simp add: o-def u'-def s'-def*)
then have *infsum-u'*: $\langle s' e = \text{infsum } (u' e) \text{ UNIV} \rangle$ **for** e
 by (*metis infsumI*)
have *tc-u-x[simp]*: $\langle \text{trace-class } (\text{from-trace-class } (u j) o_{CL} x) \rangle$ **for** j
 by (*simp add: trace-class-comp-left*)

have *summable-u'-pairs*: $\langle (\lambda(e, j). u' e j) \text{ summable-on } \text{UNIV} \times \text{UNIV} \rangle$
proof –
have $\langle \text{trace-class } (\text{from-trace-class } t o_{CL} (x \otimes_o \text{id-cblinfun})) \rangle$
 by (*simp add: trace-class-comp-left*)
from *trace-exists[OF is-onb-ket this]*
have $\langle (\lambda e j. \text{ket } ej \cdot_C (\text{from-trace-class } t *_V (x \otimes_o \text{id-cblinfun}) *_V \text{ket } ej)) \text{ summable-on } \text{UNIV} \rangle$
 by (*simp-all add: summable-on-reindex o-def*)
then show *?thesis*
 by (*simp-all add: o-def u'-def[abs-def] u-def*
trace-class-comp-left trace-class-comp-right Abs-trace-class-inverse tensor-ell2-right-apply

ket-pair-split tensor-op-ell2 case-prod-unfold cinner-adj-right
compose-tcl.rep-eq compose-tcr.rep-eq)
qed

have *u'-tensor*: $\langle u' e j = \text{ket } (e, j) \cdot_C ((\text{from-trace-class } t o_{CL} (x \otimes_o \text{id-cblinfun})) *_V \text{ket } (e, j)) \rangle$
for $e j$
 by (*simp add: u'-def u-def tensor-op-ell2 tensor-ell2-right-apply Abs-trace-class-inverse*
trace-class-comp-left trace-class-comp-right cinner-adj-right compose-tcl.rep-eq compose-tcr.rep-eq
flip: tensor-ell2-ket)

have $\langle ((\lambda e. e \cdot_C ((\text{from-trace-class } (\text{partial-trace } t) o_{CL} x) *_V e)) \text{ has-sum } s) (\text{range } \text{ket}) \rangle$
unfolding *s-def*
apply (*rule trace-has-sum*)
 by (*auto simp: trace-class-comp-left*)
then have $\langle (s' \text{ has-sum } s) \text{ UNIV} \rangle$
apply (*subst (asm) has-sum-reindex*)
 by (*auto simp: o-def s'-def[abs-def]*)
then have $\langle s = \text{infsum } s' \text{ UNIV} \rangle$
 by (*simp add: infsumI*)
also have $\langle \dots = \text{infsum } (\lambda e. \text{infsum } (u' e) \text{ UNIV}) \text{ UNIV} \rangle$
 using *infsum-u'* **by** *presburger*
also have $\langle \dots = (\sum_{\infty} (e, j) \in \text{UNIV}. u' e j) \rangle$
apply (*subst infsum-Sigma'-banach*)
apply (*rule summable-u'-pairs*)
 by *simp*
also have $\langle \dots = \text{trace } (\text{from-trace-class } t o_{CL} (x \otimes_o \text{id-cblinfun})) \rangle$

unfolding u' -tensor
by (simp add: trace-ket-sum cond-case-prod-eta trace-class-comp-left)
finally show ?thesis
by (simp add: s-def)
qed

lemma right-amplification-weak-star-cont[simp]:
 $\langle \text{continuous-map weak-star-topology weak-star-topology } (\lambda a. a \otimes_{\circ} \text{id-cblinfun}) \rangle$
— Logically does not belong in this theory but uses the partial trace in the proof.
proof (unfold weak-star-topology-def', rule continuous-map-pullback-both)
show $\langle S \subseteq f - 'UNIV \rangle$ **for** $S :: \langle 'x \text{ set} \rangle$ **and** $f :: \langle 'x \Rightarrow 'y \rangle$
by simp
define $g' :: \langle (('b \text{ ell2}, 'a \text{ ell2}) \text{ trace-class} \Rightarrow \text{complex}) \Rightarrow (('b \times 'c) \text{ ell2}, ('a \times 'c) \text{ ell2}) \text{ trace-class} \Rightarrow \text{complex} \rangle$ **where**
 $\langle g' \tau t = \tau (\text{partial-trace } t) \rangle$ **for** τt
have $\langle \text{continuous-on } UNIV \ g' \rangle$
by (simp add: continuous-on-coordinatewise-then-product g'-def)
then show $\langle \text{continuous-map euclidean euclidean } g' \rangle$
using continuous-map-iff-continuous2 **by** blast
show $\langle g' (\lambda t. \text{trace } (\text{from-trace-class } t \ o_{CL} \ x)) =$
 $(\lambda t. \text{trace } (\text{from-trace-class } t \ o_{CL} \ x \otimes_{\circ} \text{id-cblinfun})) \rangle$ **for** x
by (auto intro!: ext simp: g'-def trace-partial-trace-compose-eq-trace-compose-tensor-id)
qed

lemma left-amplification-weak-star-cont[simp]:
 $\langle \text{continuous-map weak-star-topology weak-star-topology } (\lambda b. \text{id-cblinfun} \otimes_{\circ} b :: ('c \times 'a) \text{ ell2} \Rightarrow_{CL} ('c \times 'b) \text{ ell2}) \rangle$
— Logically does not belong in this theory but uses the partial trace in the proof.
proof —
have $\langle \text{continuous-map weak-star-topology weak-star-topology } ($
 $(\lambda x. x \ o_{CL} \ \text{swap-ell2}) \ o (\lambda x. \text{swap-ell2} \ o_{CL} \ x) \ o (\lambda a. a \otimes_{\circ} \text{id-cblinfun} :: ('a \times 'c) \text{ ell2} \Rightarrow_{CL} ('b \times 'c) \text{ ell2})) \rangle$
by (auto intro!: continuous-map-compose[**where** $X' = \text{weak-star-topology}$]
continuous-map-left-comp-weak-star continuous-map-right-comp-weak-star)
then show ?thesis
by (auto simp: o-def)
qed

lemma partial-trace-plus: $\langle \text{partial-trace } (t + u) = \text{partial-trace } t + \text{partial-trace } u \rangle$
proof —
from partial-trace-has-sum[of t] **and** partial-trace-has-sum[of u]
have $\langle ((\lambda j. \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) \ t) (\text{tensor-ell2-right } (\text{ket } j)))$
 $+ \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) \ u) (\text{tensor-ell2-right } (\text{ket } j)))$
has-sum
 $\text{partial-trace } t + \text{partial-trace } u) \ UNIV \rangle$ (**is** $\langle (?f \text{ has-sum } -) \ - \rangle$)
by (rule has-sum-add)

moreover have $\langle ?f j = \text{compose-tcl} (\text{compose-tcr} ((\text{tensor-ell2-right} (\text{ket } j))^*) (t + u)) (\text{tensor-ell2-right} (\text{ket } j)) \rangle$ **(is** $\langle ?f j = ?g j \rangle$ **for** j
by $(\text{simp add: compose-tcl.add-left compose-tcr.add-right})$
ultimately have $\langle (?g \text{ has-sum partial-trace } t + \text{partial-trace } u) \text{ UNIV} \rangle$
by simp
moreover have $\langle (?g \text{ has-sum partial-trace } (t + u)) \text{ UNIV} \rangle$
by $(\text{simp add: partial-trace-has-sum})$
ultimately show $?thesis$
using has-sum-unique **by** blast
qed

lemma $\text{partial-trace-scaleC}$: $\langle \text{partial-trace} (c *_C t) = c *_C \text{partial-trace } t \rangle$
by $(\text{simp add: partial-trace-def infsum-scaleC-right compose-tcr.scaleC-right compose-tcl.scaleC-left})$

lemma $\text{partial-trace-tensor}$: $\langle \text{partial-trace} (\text{tc-tensor } t u) = \text{trace-tc } u *_C t \rangle$

proof –

define $t' u'$ **where** $\langle t' = \text{from-trace-class } t \rangle$ **and** $\langle u' = \text{from-trace-class } u \rangle$
have 1 : $\langle (\lambda j. \text{ket } j \cdot_C (\text{from-trace-class } u *_V \text{ket } j)) \text{ summable-on UNIV} \rangle$
using trace-exists **[where** $B = \langle \text{range ket} \rangle$ **and** $A = \langle \text{from-trace-class } u \rangle$
by $(\text{simp add: summable-on-reindex o-def})$
have $\langle \text{partial-trace} (\text{tc-tensor } t u) =$
 $(\sum_{\infty} j. \text{compose-tcl} (\text{compose-tcr} (\text{tensor-ell2-right} (\text{ket } j))^*) (\text{tc-tensor } t u)) (\text{tensor-ell2-right}$
 $(\text{ket } j)) \rangle$

by $(\text{simp add: partial-trace-def})$

also have $\langle \dots = (\sum_{\infty} j. (\text{ket } j \cdot_C (\text{from-trace-class } u *_V \text{ket } j)) *_C t) \rangle$

proof –

have $*$: $\langle \text{tensor-ell2-right} (\text{ket } j)^* o_{CL} t' \otimes_o u' o_{CL} \text{tensor-ell2-right} (\text{ket } j) =$
 $(\text{ket } j \cdot_C (u' *_V \text{ket } j)) *_C t' \rangle$ **for** j

by $(\text{auto intro!}: \text{cblinfun-eqI simp: tensor-op-ell2})$

show $?thesis$

apply $(\text{rule infsum-cong})$

by $(\text{auto intro!}: \text{from-trace-class-inject}[THEN \text{iffD1}] \text{simp flip: } t'\text{-def } u'\text{-def}$

$\text{simp: } * \text{compose-tcl.rep-eq compose-tcr.rep-eq tc-tensor.rep-eq scaleC-trace-class.rep-eq})$

qed

also have $\langle \dots = \text{trace-tc } u *_C t \rangle$

by $(\text{auto intro!}: \text{infsum-scaleC-left simp: trace-tc-def trace-alt-def}[OF \text{is-onb-ket}] \text{infsum-reindex o-def 1})$

finally show $?thesis$

by –

qed

lemma $\text{bounded-clinear-partial-trace}$ $[\text{bounded-clinear, iff}]$: $\langle \text{bounded-clinear partial-trace} \rangle$

apply $(\text{rule bounded-clinearI}[\text{where } K=I])$

by $(\text{auto simp add: partial-trace-plus partial-trace-scaleC partial-trace-norm-reducing})$

lemma $\text{vector-sandwich-partial-trace-has-sum}$:

$\langle ((\lambda z. ((x \otimes_s \text{ket } z) \cdot_C (\text{from-trace-class } \varrho *_V (y \otimes_s \text{ket } z))))$

$\text{has-sum } x \cdot_C (\text{from-trace-class} (\text{partial-trace } \varrho) *_V y)) \text{ UNIV} \rangle$

proof –

```

define  $x\rho y$  where  $\langle x\rho y = x \cdot_C (\text{from-trace-class } (\text{partial-trace } \rho) *_V y) \rangle$ 
have  $\langle ((\lambda j. \text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) \rho) (\text{tensor-ell2-right } (\text{ket } j))))$ 
   $\text{has-sum } \text{partial-trace } \rho) \text{ UNIV} \rangle$ 
using  $\text{partial-trace-has-sum}$  by force
then have  $\langle ((\lambda j. x \cdot_C (\text{from-trace-class } (\text{compose-tcl } (\text{compose-tcr } ((\text{tensor-ell2-right } (\text{ket } j))^*) \rho) (\text{tensor-ell2-right } (\text{ket } j)))) *_V y))$ 
   $\text{has-sum } x\rho y) \text{ UNIV} \rangle$ 
unfolding  $x\rho y\text{-def}$ 
apply  $(\text{rule } \text{Infinite-Sum.has-sum-bounded-linear}[\text{rotated}])$ 
by  $(\text{intro } \text{bounded-clinear.bounded-linear } \text{bounded-linear-intros})$ 
then have  $\langle ((\lambda j. x \cdot_C (\text{tensor-ell2-right } (\text{ket } j))^* *_V \text{from-trace-class } \rho *_V y \otimes_s \text{ket } j)) \text{has-sum } x\rho y) \text{ UNIV} \rangle$ 
by  $(\text{simp add: } \text{compose-tcl.rep-eq } \text{compose-tcr.rep-eq})$ 
then show  $?thesis$ 
by  $(\text{metis } (\text{no-types, lifting}) \text{cinner-adj-right } \text{has-sum-cong } \text{tensor-ell2-right-apply } x\rho y\text{-def})$ 
qed

```

lemma $\text{vector-sandwich-partial-trace}$:

```

 $\langle x \cdot_C (\text{from-trace-class } (\text{partial-trace } \rho) *_V y) =$ 
   $(\sum_{\infty z. ((x \otimes_s \text{ket } z) \cdot_C (\text{from-trace-class } \rho *_V (y \otimes_s \text{ket } z)))) \rangle$ 
by  $(\text{metis } (\text{mono-tags, lifting}) \text{infsumI } \text{vector-sandwich-partial-trace-has-sum})$ 

```

unbundle $\text{no } \text{cblinfun-syntax}$

end

16 Von-Neumann-Algebras – Von Neumann algebras and the double commutant theorem

theory $\text{Von-Neumann-Algebras}$

imports $\text{Hilbert-Space-Tensor-Product}$

begin

unbundle cblinfun-syntax

16.1 Commutants

definition $\langle \text{commutant } F = \{x. \forall y \in F. x \circ_{CL} y = y \circ_{CL} x\} \rangle$

lemma $\text{sandwich-unitary-commutant}$:

fixes $U :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$

assumes $[\text{simp}]: \langle \text{unitary } U \rangle$

shows $\langle \text{sandwich } U \text{ 'commutant } X = \text{commutant } (\text{sandwich } U \text{ ' } X) \rangle$

proof $(\text{rule } \text{Set.set-eqI})$

fix x

let $?comm = \langle \lambda a b. a \circ_{CL} b = b \circ_{CL} a \rangle$

have $\langle x \in \text{sandwich } U \text{ ' commutant } X \longleftrightarrow \text{sandwich } (U*) \ x \in \text{commutant } X \rangle$
apply (*subst inj-image-mem-iff*[*symmetric*, **where** $f = \langle \text{sandwich } (U*) \rangle$])
by (*auto intro!*: *inj-sandwich-isometry simp: image-image*
simp flip: cblinfun-apply-cblinfun-compose sandwich-compose)
also have $\langle \dots \longleftrightarrow (\forall y \in X. ?\text{comm } (\text{sandwich } (U*) \ x) \ y) \rangle$
by (*simp add: commutant-def*)
also have $\langle \dots \longleftrightarrow (\forall y \in X. ?\text{comm } x \ (\text{sandwich } U \ y)) \rangle$
apply (*rule ball-cong, simp*)
apply (*simp add: sandwich-apply*)
by (*smt (verit) assms cblinfun-assoc-left(1) cblinfun-compose-id-left cblinfun-compose-id-right*
unitaryD1 unitaryD2)
also have $\langle \dots \longleftrightarrow (\forall y \in \text{sandwich } U \text{ ' } X. ?\text{comm } x \ y) \rangle$
by fast
also have $\langle \dots \longleftrightarrow x \in \text{commutant } (\text{sandwich } U \text{ ' } X) \rangle$
by (*simp add: commutant-def*)
finally show $\langle x \in (*_V) \ (\text{sandwich } U) \text{ ' commutant } X \longleftrightarrow x \in \text{commutant } ((*_V) \ (\text{sandwich } U) \text{ ' } X) \rangle$
by -
qed

lemma commutant-tensor1: $\langle \text{commutant } (\text{range } (\lambda a. a \otimes_o \text{id-cblinfun})) = \text{range } (\lambda b. \text{id-cblinfun} \otimes_o b) \rangle$

proof (*rule Set.set-eqI, rule iffI*)
fix $x :: \langle ('a \times 'b) \ \text{ell2} \Rightarrow_{CL} ('a \times 'b) \ \text{ell2} \rangle$
fix $\gamma :: 'a$
assume $\langle x \in \text{commutant } (\text{range } (\lambda a. a \otimes_o \text{id-cblinfun})) \rangle$
then have *comm*: $\langle (a \otimes_o \text{id-cblinfun}) *_V x *_V \psi = x *_V (a \otimes_o \text{id-cblinfun}) *_V \psi \rangle$ **for** $a \ \psi$
by (*metis (mono-tags, lifting) commutant-def mem-Collect-eq rangeI cblinfun-apply-cblinfun-compose*)

define op where $\langle op = \text{classical-operator } (\lambda i. \text{Some } (\gamma, i :: 'b)) \rangle$

have [*simp*]: $\langle \text{classical-operator-exists } (\lambda i. \text{Some } (\gamma, i)) \rangle$

apply (*rule classical-operator-exists-inj*)

using inj-map-def by blast

define x' where $\langle x' = op *_V x *_V op \rangle$

have x' : $\langle \text{cinner } (\text{ket } j) \ (x' *_V \text{ket } l) = \text{cinner } (\text{ket } (\gamma, j)) \ (x *_V \text{ket } (\gamma, l)) \rangle$ **for** $j \ l$

by (*simp add: x'-def op-def classical-operator-ket cinner-adj-right*)

have $\langle \text{cinner } (\text{ket } (i, j)) \ (x *_V \text{ket } (k, l)) = \text{cinner } (\text{ket } (i, j)) \ ((\text{id-cblinfun} \otimes_o x) *_V \text{ket } (k, l)) \rangle$
for $i \ j \ k \ l$

proof -

have $\langle \text{cinner } (\text{ket } (i, j)) \ (x *_V \text{ket } (k, l))$
 $= \text{cinner } ((\text{butterfly } (\text{ket } i) \ (\text{ket } \gamma) \otimes_o \text{id-cblinfun}) *_V \text{ket } (\gamma, j)) \ (x *_V (\text{butterfly } (\text{ket } k)$
 $(\text{ket } \gamma) \otimes_o \text{id-cblinfun}) *_V \text{ket } (\gamma, l)) \rangle$

by (*auto simp: tensor-op-ket tensor-ell2-ket*)

also have $\langle \dots = \text{cinner } (\text{ket } (\gamma, j)) \ ((\text{butterfly } (\text{ket } \gamma) \ (\text{ket } i) \otimes_o \text{id-cblinfun}) *_V x *_V$
 $(\text{butterfly } (\text{ket } k) \ (\text{ket } \gamma) \otimes_o \text{id-cblinfun}) *_V \text{ket } (\gamma, l)) \rangle$

by (*metis (no-types, lifting) cinner-adj-left butterfly-adjoint id-cblinfun-adjoint tensor-op-adjoint*)

also have $\langle \dots = \text{cinner } (\text{ket } (\gamma, j)) \ (x *_V (\text{butterfly } (\text{ket } \gamma) \ (\text{ket } i) \otimes_o \text{id-cblinfun} \ o_{CL}$
 $\text{butterfly } (\text{ket } k) \ (\text{ket } \gamma) \otimes_o \text{id-cblinfun}) *_V \text{ket } (\gamma, l)) \rangle$

```

    unfolding comm by (simp add: cblinfun-apply-cblinfun-compose)
  also have ⟨... = cinner (ket i) (ket k) * cinner (ket (γ,j)) (x *V ket (γ,l))⟩
    by (simp add: comp-tensor-op tensor-op-ket tensor-op-scaleC-left cinner-ket tensor-ell2-ket)
  also have ⟨... = cinner (ket i) (ket k) * cinner (ket j) (x' *V ket l)⟩
    by (simp add: x')
  also have ⟨... = cinner (ket (i,j)) ((id-cblinfun ⊗o x') *V ket (k,l))⟩
    apply (simp add: tensor-op-ket)
    by (simp flip: tensor-ell2-ket)
  finally show ?thesis by -
qed
then have ⟨x = (id-cblinfun ⊗o x')⟩
  by (auto intro!: equal-ket cinner-ket-eqI)
then show ⟨x ∈ range (λb. id-cblinfun ⊗o b)⟩
  by auto
next
fix x :: ⟨('a × 'b) ell2 ⇒CL ('a × 'b) ell2⟩
assume ⟨x ∈ range (λb. id-cblinfun ⊗o b)⟩
then obtain b where x: ⟨x = id-cblinfun ⊗o b⟩
  by auto
then show ⟨x ∈ commutant (range (λa. a ⊗o id-cblinfun))⟩
  by (auto simp: x commutant-def comp-tensor-op)
qed

lemma csubspace-commutant[simp]: ⟨csubspace (commutant X)⟩
  by (auto simp add: complex-vector.subspace-def commutant-def cblinfun-compose-add-right
  cblinfun-compose-add-left)

lemma closed-commutant[simp]: ⟨closed (commutant X)⟩
proof (subst closed-sequential-limits, intro allI impI, erule conjE)
  fix s :: ⟨nat ⇒ -⟩ and l
  assume s-comm: ⟨∀ n. s n ∈ commutant X⟩
  assume ⟨s ⟶ l⟩
  have ⟨l oCL x - x oCL l = 0⟩ if ⟨x ∈ X⟩ for x
  proof -
    from ⟨s ⟶ l⟩
    have ⟨(λn. s n oCL x - x oCL s n) ⟶ l oCL x - x oCL l⟩
      apply (rule isCont-tendsto-compose[rotated])
      by (intro continuous-intros)
    then have ⟨(λ-. 0) ⟶ l oCL x - x oCL l⟩
      using s-comm that by (auto simp add: commutant-def)
    then show ?thesis
      by (simp add: LIMSEQ-const-iff that)
  qed
then show ⟨l ∈ commutant X⟩
  by (simp add: commutant-def)
qed

lemma closed-csubspace-commutant[simp]: ⟨closed-csubspace (commutant X)⟩
  apply (rule closed-csubspace.intro) by simp-all

```


lemma *commutant-mult*: $\langle a \circ_{CL} b \in \text{commutant } X \rangle$ **if** $\langle a \in \text{commutant } X \rangle$ **and** $\langle b \in \text{commutant } X \rangle$

using *that*
apply (*auto simp: commutant-def cblinfun-compose-assoc*)
by (*simp flip: cblinfun-compose-assoc*)

lemma *double-commutant-grows*[*simp*]: $\langle X \subseteq \text{commutant } (\text{commutant } X) \rangle$
by (*auto simp add: commutant-def*)

lemma *commutant-antimono*: $\langle X \subseteq Y \implies \text{commutant } X \supseteq \text{commutant } Y \rangle$
by (*auto simp add: commutant-def*)

lemma *triple-commutant*[*simp*]: $\langle \text{commutant } (\text{commutant } (\text{commutant } X)) = \text{commutant } X \rangle$
by (*auto simp: commutant-def*)

lemma *commutant-adj*: $\langle \text{adj } 'a \text{ commutant } X = \text{commutant } (\text{adj } 'a X) \rangle$
apply (*auto intro!: image-eqI double-adj[symmetric] simp: commutant-def simp flip: adj-cblinfun-compose*)
by (*metis adj-cblinfun-compose double-adj*)

lemma *commutant-empty*[*simp*]: $\langle \text{commutant } \{\} = UNIV \rangle$
by (*simp add: commutant-def*)

lemma *commutant-weak-star-closed*[*simp*]: $\langle \text{closedin weak-star-topology } (\text{commutant } X) \rangle$

proof –

have *comm-inter*: $\langle \text{commutant } X = (\bigcap x \in X. \text{commutant } \{x\}) \rangle$
by (*auto simp: commutant-def*)

have *comm-x*: $\langle \text{commutant } \{x\} = (\lambda y. x \circ_{CL} y - y \circ_{CL} x) - \{0\} \rangle$ **for** $x :: 'a \Rightarrow_{CL} 'a$
by (*auto simp add: commutant-def vimage-def*)

have *cont*: $\langle \text{continuous-map weak-star-topology weak-star-topology } (\lambda y. x \circ_{CL} y - y \circ_{CL} x) \rangle$
for $x :: 'a \Rightarrow_{CL} 'a$

apply (*rule continuous-intros*)

by (*simp-all add: continuous-map-left-comp-weak-star continuous-map-right-comp-weak-star*)

have $\langle \text{closedin weak-star-topology } ((\lambda y. x \circ_{CL} y - y \circ_{CL} x) - \{0\}) \rangle$ **for** $x :: 'a \Rightarrow_{CL} 'a$

using *closedin-vimage*[**where** $U = \langle \text{weak-star-topology} \rangle$ **and** $S = \langle \{0\} \rangle$ **and** $T = \langle \text{weak-star-topology} \rangle$]

using *cont* **by** (*auto simp add: closedin-Hausdorff-singleton*)

then show *?thesis*

apply (*cases* $\langle X = \{\} \rangle$)

using *closedin-topospace*[*of weak-star-topology*]

by (*auto simp add: comm-inter comm-x*)

qed

lemma *cspan-in-double-commutant*: $\langle \text{cspan } X \subseteq \text{commutant } (\text{commutant } X) \rangle$
by (*simp add: complex-vector.span-minimal*)

lemma *weak-star-closure-in-double-commutant*: $\langle \text{weak-star-topology closure-of } X \subseteq \text{commutant} (\text{commutant } X) \rangle$

by (*simp add: closure-of-minimal*)

lemma *weak-star-closure-cspan-in-double-commutant*: $\langle \text{weak-star-topology closure-of cspan } X \subseteq \text{commutant} (\text{commutant } X) \rangle$

by (*simp add: closure-of-minimal cspan-in-double-commutant*)

lemma *commutant-memberI*:

assumes $\langle \bigwedge y. y \in X \implies x \circ_{CL} y = y \circ_{CL} x \rangle$

shows $\langle x \in \text{commutant } X \rangle$

using *assms* **by** (*simp add: commutant-def*)

lemma *commutant-sot-closed*: $\langle \text{closedin cstrong-operator-topology} (\text{commutant } A) \rangle$

— [2], Exercise IX.6.2

proof (*cases* $\langle A = \{\} \rangle$)

case *True*

then show *?thesis*

apply *simp*

by (*metis closedin-topspace cstrong-operator-topology-topspace*)

next

case *False*

have *closed-a*: $\langle \text{closedin cstrong-operator-topology} (\text{commutant } \{a\}) \rangle$ **for** $a :: \langle 'a \Rightarrow_{CL} 'a \rangle$

proof —

have *comm-a*: $\langle \text{commutant } \{a\} = (\lambda b. a \circ_{CL} b - b \circ_{CL} a) - \{0\} \rangle$

by (*auto simp: commutant-def*)

have *closed-0*: $\langle \text{closedin cstrong-operator-topology} \{0\} \rangle$

apply (*rule closedin-Hausdorff-singleton*)

by *simp-all*

have *cont*: $\langle \text{continuous-map cstrong-operator-topology cstrong-operator-topology} (\lambda b. a \circ_{CL} b - b \circ_{CL} a) \rangle$

by (*intro continuous-intros continuous-map-left-comp-sot continuous-map-right-comp-sot*)

show *?thesis*

using *closedin-vimage[OF closed-0 cont]*

by (*simp add: comm-a*)

qed

have ***: $\langle \text{commutant } A = (\bigcap a \in A. \text{commutant } \{a\}) \rangle$

by (*auto simp add: commutant-def*)

show *?thesis*

by (*auto intro!: closedin-Inter simp: * False closed-a*)

qed

lemma *commutant-tensor1'*: $\langle \text{commutant} (\text{range} (\lambda a. \text{id-cblinfun} \otimes_o a)) = \text{range} (\lambda b. b \otimes_o \text{id-cblinfun}) \rangle$

proof —

have $\langle \text{commutant} (\text{range} (\lambda a. \text{id-cblinfun} \otimes_o a)) = \text{commutant} (\text{sandwich swap-ell2 } \langle \text{range} (\lambda a. a \otimes_o \text{id-cblinfun}) \rangle) \rangle$

by (*metis (no-types, lifting) image-cong range-composition swap-tensor-op-sandwich*)

also have $\langle \dots = \text{sandwich swap-ell2 } \text{'commutant (range (\lambda a. a \otimes_o \text{id-cblinfun}))} \rangle$
by (*simp add: sandwich-unitary-commutant*)
also have $\langle \dots = \text{sandwich swap-ell2 } \text{'range (\lambda a. \text{id-cblinfun} \otimes_o a) \rangle$
by (*simp add: commutant-tensor1*)
also have $\langle \dots = \text{range (\lambda b. b \otimes_o \text{id-cblinfun})} \rangle$
by force
finally show *?thesis*
by –
qed

lemma *closed-map-sot-tensor-op-id-right:*

$\langle \text{closed-map cstrong-operator-topology cstrong-operator-topology } (\lambda a. a \otimes_o \text{id-cblinfun} :: ('a \times 'b) \text{ ell2} \Rightarrow_{CL} ('a \times 'b) \text{ ell2}) \rangle$

proof (*unfold closed-map-def, intro allI impI*)

fix $U :: \langle ('a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2}) \text{ set} \rangle$

assume *closed-U*: $\langle \text{closedin cstrong-operator-topology } U \rangle$

have *aux1*: $\langle \text{range } f \subseteq X \iff (\forall x. f x \in X) \rangle$ **for** $f :: \langle 'x \Rightarrow 'y \rangle$ **and** X

by *blast*

have $\langle l \in (\lambda a. a \otimes_o \text{id-cblinfun}) \text{' } U \rangle$ **if** *range*: $\langle \text{range } (\lambda x. f x) \subseteq (\lambda a. a \otimes_o \text{id-cblinfun}) \text{' } U \rangle$
and *limit*: $\langle \text{limitin cstrong-operator-topology } f l F \rangle$ **and** $\langle F \neq \perp \rangle$

for f **and** $l :: \langle ('a \times 'b) \text{ ell2} \Rightarrow_{CL} ('a \times 'b) \text{ ell2} \rangle$ **and** $F :: \langle (('a \times 'b) \text{ ell2} \Rightarrow_{CL} ('a \times 'b) \text{ ell2}) \text{ filter} \rangle$

proof –

from *range* **obtain** f' **where** $f'U$: $\langle \text{range } f' \subseteq U \rangle$ **and** *f-def*: $\langle f y = f' y \otimes_o \text{id-cblinfun} \rangle$
for y

apply *atomize-elim*

apply (*subst aux1*)

apply (*rule choice2*)

by *auto*

have $\langle l \in \text{commutant (range (\lambda a. \text{id-cblinfun} \otimes_o a))} \rangle$

proof (*rule commutant-memberI*)

fix $c :: \langle ('a \times 'b) \text{ ell2} \Rightarrow_{CL} ('a \times 'b) \text{ ell2} \rangle$

assume $\langle c \in \text{range } (\lambda a. \text{id-cblinfun} \otimes_o a) \rangle$

then obtain c' **where** *c-def*: $\langle c = \text{id-cblinfun} \otimes_o c' \rangle$

by *blast*

from *limit* **have** 1: $\langle \text{limitin cstrong-operator-topology } ((\lambda z. z \circ_{CL} c) \circ f) (l \circ_{CL} c) F \rangle$

apply(*rule continuous-map-limit[rotated]*)

by (*simp add: continuous-map-right-comp-sot*)

from *limit* **have** 2: $\langle \text{limitin cstrong-operator-topology } ((\lambda z. c \circ_{CL} z) \circ f) (c \circ_{CL} l) F \rangle$

apply(*rule continuous-map-limit[rotated]*)

by (*simp add: continuous-map-left-comp-sot*)

have 3: $\langle f x \circ_{CL} c = c \circ_{CL} f x \rangle$ **for** x

by (*simp add: f-def c-def comp-tensor-op*)

from 1 2 **show** $\langle l \circ_{CL} c = c \circ_{CL} l \rangle$

unfolding 3 *o-def*

by (*meson hausdorff-sot limitin-Hausdorff-unique that(3)*)

```

qed
then have ⟨l ∈ range (λa. a ⊗o id-cblinfun)⟩
  by (simp add: commutant-tensor1')
then obtain l' where l-def: ⟨l = l' ⊗o id-cblinfun⟩
  by blast
have ⟨limitin cstrong-operator-topology f' l' F⟩
proof (rule limitin-cstrong-operator-topology[THEN iffD2], rule allI)
  fix ψ fix b :: 'b
  have ⟨((λx. f x *V (ψ ⊗s ket b)) → l *V (ψ ⊗s ket b)) F⟩
    using limitin-cstrong-operator-topology that(2) by auto
  then have ⟨((λx. (f' x *V ψ) ⊗s ket b) → (l' *V ψ) ⊗s ket b) F⟩
    by (simp add: f-def l-def tensor-op-ell2)
  then have ⟨((λx. (tensor-ell2-right (ket b))* *V ((f' x *V ψ) ⊗s ket b))
    → (tensor-ell2-right (ket b))* *V ((l' *V ψ) ⊗s ket b)) F⟩
    apply (rule cblinfun.tendsto[rotated])
    by simp
  then show ⟨((λx. f' x *V ψ) → l' *V ψ) F⟩
    by (simp add: tensor-ell2-right-adj-apply)
qed
with closed-U f'U ⟨F ≠ ⊥⟩ have ⟨l' ∈ U⟩
  by (simp add: Misc-Tensor-Product.limitin-closedin)
then show ⟨l ∈ (λa. a ⊗o id-cblinfun) 'U⟩
  by (simp add: l-def)
qed
then show ⟨closedin cstrong-operator-topology ((λa. a ⊗o id-cblinfun :: ('a × 'b) ell2 ⇒CL
('a × 'b) ell2) 'U)⟩
  apply (rule-tac closedin-if-converge-inside)
  by simp-all
qed

lemma id-in-commutant[iff]: ⟨id-cblinfun ∈ commutant A⟩
  by (simp add: commutant-memberI)

lemma double-commutant-hull: ⟨commutant (commutant X) = (λX. commutant (commutant
X) = X) hull X⟩
  by (smt (verit) commutant-antimono double-commutant-grows hull-unique triple-commutant)

lemma commutant-adj-closed: ⟨(λx. x ∈ X ⇒ x* ∈ X) ⇒ x ∈ commutant X ⇒ x* ∈
commutant X⟩
  by (metis (no-types, opaque-lifting) commutant-adj commutant-antimono double-adj imageI
subset-iff)

lemma double-commutant-Un-left: ⟨commutant (commutant (commutant (commutant X) ∪ Y))
= commutant (commutant (X ∪ Y))⟩
  apply (simp add: double-commutant-hull cong: arg-cong[where f=⟨Hull.hull -⟩])
  using hull-Un-left by fastforce

lemma double-commutant-Un-right: ⟨commutant (commutant (X ∪ commutant (commutant
Y))) = commutant (commutant (X ∪ Y))⟩

```

by (metis Un-ac(3) double-commutant-Un-left)

lemma *amplification-double-commutant-commute*:

$\langle \text{commutant} (\text{commutant} ((\lambda a. a \otimes_{\circ} \text{id-cblinfun}) 'X))$
 $= (\lambda a. a \otimes_{\circ} \text{id-cblinfun}) ' \text{commutant} (\text{commutant } X) \rangle$

— [7], Corollary IV.1.5

proof —

define $\pi :: \langle ('a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2}) \Rightarrow (('a \times 'b) \text{ ell2} \Rightarrow_{CL} ('a \times 'b) \text{ ell2}) \rangle$ **where**

$\langle \pi a = a \otimes_{\circ} \text{id-cblinfun} \rangle$ **for** a

define $U :: \langle 'b \Rightarrow 'a \text{ ell2} \Rightarrow_{CL} ('a \times 'b) \text{ ell2} \rangle$ **where** $\langle U i = \text{tensor-ell2-right} (\text{ket } i) \rangle$ **for** i
 $:: 'b$

write *commutant* ($\langle -' \rangle$ [120] 120)

— Notation X' for X'

write *id-cblinfun* ($\langle 1 \rangle$)

have $*$: $\langle (\pi 'X)'' \subseteq \text{range } \pi \rangle$ **for** X

proof (*rule subsetI*)

fix x **assume** *asm*: $\langle x \in (\pi 'X)'' \rangle$

fix t

define y **where** $\langle y = U t *_{oCL} x o_{CL} U t \rangle$

have $\langle \text{ket} (k,l) \cdot_C (x *_V \text{ket} (m,n)) = \text{ket} (k,l) \cdot_C (\pi y *_V \text{ket} (m,n)) \rangle$ **for** $k l m n$

proof —

have *comm*: $\langle x o_{CL} (U i o_{CL} U j *) = (U i o_{CL} U j *) o_{CL} x \rangle$ **for** $i j$

proof —

have $\langle U i o_{CL} U j * = \text{id-cblinfun} \otimes_{\circ} \text{butterfly} (\text{ket } i) (\text{ket } j) \rangle$

by (*simp add: U-def tensor-ell2-right-butterfly*)

also have $\langle \dots \in (\pi 'X)' \rangle$

by (*simp add: π -def commutant-def comp-tensor-op*)

finally show *?thesis*

using *asm*

by (*simp add: commutant-def*)

qed

have $\langle \text{ket} (k,l) \cdot_C (x *_V \text{ket} (m,n)) = \text{ket } k \cdot_C (U l * *_V x *_V U n *_V \text{ket } m) \rangle$

by (*simp add: cinner-adj-right U-def tensor-ell2-ket*)

also have $\langle \dots = \text{ket } k \cdot_C (U l * *_V x *_V U n *_V U t * *_V U t *_V \text{ket } m) \rangle$

using *U-def* **by** *fastforce*

also have $\langle \dots = \text{ket } k \cdot_C (U l * *_V U n *_V U t * *_V x *_V U t *_V \text{ket } m) \rangle$

using *simp-a-oCL-b'[OF comm]*

by *simp*

also have $\langle \dots = \text{of-bool} (l=n) * (\text{ket } k \cdot_C (U t * *_V x *_V U t *_V \text{ket } m)) \rangle$

using *U-def* **by** *fastforce*

also have $\langle \dots = \text{of-bool} (l=n) * (\text{ket } k \cdot_C (y *_V \text{ket } m)) \rangle$

using *y-def* **by** *force*

also have $\langle \dots = \text{ket} (k,l) \cdot_C (\pi y *_V \text{ket} (m,n)) \rangle$

by (*simp add: π -def tensor-op-ell2 flip: tensor-ell2-ket*)

finally show *?thesis*

by —

qed

then have $\langle x = \pi y \rangle$

by (*metis cinner-ket-eqI equal-ket surj-pair*)
 then show $\langle x \in \text{range } \pi \rangle$
 by *simp*
 qed
 have **: $\langle \pi '(Y') = (\pi 'Y)' \cap \text{range } \pi \rangle$ for Y
 using *inj-tensor-left[of id-cblinfun]*
 apply (*auto simp add: commutant-def π -def comp-tensor-op*
 intro!: image-eqI)
 using *injD* by *fastforce*
 have 1: $\langle (\pi 'X)'' \subseteq \pi '(X'') \rangle$ for X
 proof –
 have $\langle (\pi 'X)'' \subseteq (\pi 'X)'' \cap \text{range } \pi \rangle$
 by (*simp add: **)
 also have $\langle \dots \subseteq ((\pi 'X)')' \cap \text{range } \pi \rangle$
 by (*simp add: commutant-antimono inf.coboundedI1*)
 also have $\langle \dots = \pi '(X'') \rangle$
 by (*simp add: ***)
 finally show ?thesis
 by –
 qed
 have $\langle x \text{ } o_{CL} \ y = y \text{ } o_{CL} \ x \rangle$ if $\langle x \in \pi '(X'') \rangle$ and $\langle y \in (\pi 'X)'\rangle$ for $x \ y$
 proof (*intro equal-ket cinner-ket-eqI*)
 fix $i \ j :: \langle a \times b \rangle$
 from that obtain w where $\langle w \in X'' \rangle$ and x -def: $\langle x = w \otimes_o \mathbf{1} \rangle$
 by (*auto simp: π -def*)
 obtain $i1 \ i2$ where i -def: $\langle i = (i1, i2) \rangle$ by *force*
 obtain $j1 \ j2$ where j -def: $\langle j = (j1, j2) \rangle$ by *force*
 define y_0 where $\langle y_0 = U \ i2* \ o_{CL} \ y \ o_{CL} \ U \ j2 \rangle$

 have $\langle y_0 \in X' \rangle$
 proof (*rule commutant-memberI*)
 fix z assume $\langle z \in X \rangle$
 then have $\langle z \otimes_o \mathbf{1} \in \pi 'X \rangle$
 by (*auto simp: π -def*)
 have $\langle y_0 \text{ } o_{CL} \ z = U \ i2* \ o_{CL} \ y \ o_{CL} \ (z \otimes_o \mathbf{1}) \text{ } o_{CL} \ U \ j2 \rangle$
 by (*auto intro!: equal-ket simp add: y_0 -def U -def tensor-op-ell2*)
 also have $\langle \dots = U \ i2* \ o_{CL} \ (z \otimes_o \mathbf{1}) \text{ } o_{CL} \ y \ o_{CL} \ U \ j2 \rangle$
 using $\langle z \otimes_o \mathbf{1} \in \pi 'X \rangle$ and $\langle y \in (\pi 'X)'\rangle$
 apply (*auto simp add: commutant-def*)
 by (*simp add: cblinfun-compose-assoc*)
 also have $\langle \dots = z \text{ } o_{CL} \ y_0 \rangle$
 by (*auto intro!: equal-ket cinner-ket-eqI*
 simp add: y_0 -def U -def tensor-op-ell2 tensor-op-adjoint simp flip: cinner-adj-left)
 finally show $\langle y_0 \text{ } o_{CL} \ z = z \text{ } o_{CL} \ y_0 \rangle$
 by –
 qed
 have $\langle \text{ket } i \cdot_C ((x \text{ } o_{CL} \ y) *_V \text{ket } j) = \text{ket } i1 \cdot_C (U \ i2* *_V (w \otimes_o \mathbf{1}) *_V y *_V U \ j2 *_V \text{ket } j1) \rangle$

by (*simp add: U-def i-def j-def tensor-ell2-ket cinner-adj-right x-def*)
 also have $\langle \dots = \text{ket } i1 \cdot_C (U \ i2* \ *V (w \otimes_o \ \mathbf{1}) \ *V (U \ i2 \ o_{CL} \ U \ i2*) \ *V \ y \ *V \ U \ j2 \ *V \ \text{ket } j1) \rangle$
 by (*simp add: U-def tensor-ell2-right-butterfly tensor-op-adjoint tensor-op-ell2 flip: cinner-adj-left*)
 also have $\langle \dots = \text{ket } i1 \cdot_C (w \ *V \ y_0 \ *V \ \text{ket } j1) \rangle$
 by (*simp add: y_0-def tensor-op-adjoint tensor-op-ell2 U-def flip: cinner-adj-left*)
 also have $\langle \dots = \text{ket } i1 \cdot_C (y_0 \ *V \ w \ *V \ \text{ket } j1) \rangle$
 using $\langle y_0 \in X \ \prime \rangle \langle w \in X \ \prime \prime \rangle$
 apply (*subst (asm) (2) commutant-def*)
 using *lift-cblinfun-comp(4)* by force
 also have $\langle \dots = \text{ket } i1 \cdot_C (U \ i2* \ *V \ y \ *V (U \ j2 \ o_{CL} \ U \ j2*) \ *V (w \otimes_o \ \mathbf{1}) \ *V \ U \ j2 \ *V \ \text{ket } j1) \rangle$
 by (*simp add: y_0-def tensor-op-adjoint tensor-op-ell2 U-def flip: cinner-adj-left*)
 also have $\langle \dots = \text{ket } i1 \cdot_C (U \ i2* \ *V \ y \ *V (w \otimes_o \ \mathbf{1}) \ *V \ U \ j2 \ *V \ \text{ket } j1) \rangle$
 by (*simp add: U-def tensor-ell2-right-butterfly tensor-op-adjoint tensor-op-ell2 flip: cinner-adj-left*)
 also have $\langle \dots = \text{ket } i \cdot_C ((y \ o_{CL} \ x) \ *V \ \text{ket } j) \rangle$
 by (*simp add: U-def i-def j-def tensor-ell2-ket cinner-adj-right x-def*)
 finally show $\langle \text{ket } i \cdot_C ((x \ o_{CL} \ y) \ *V \ \text{ket } j) = \text{ket } i \cdot_C ((y \ o_{CL} \ x) \ *V \ \text{ket } j) \rangle$
 by –
 qed
 then have 2: $\langle (\pi \ \prime \ X) \prime \prime \supseteq \pi \ \prime \ (X \ \prime \prime) \rangle$
 by (*auto intro!: commutant-memberI*)
 from 1 2 show ?thesis
 by (*auto simp flip: π -def*)
 qed

lemma *amplification-double-commutant-commute'*:
 $\langle \text{commutant} (\text{commutant} ((\lambda a. \text{id-cblinfun} \otimes_o a) \ \prime \ X)) \rangle$
 $= (\lambda a. \text{id-cblinfun} \otimes_o a) \ \prime \ \text{commutant} (\text{commutant } X) \rangle$
proof –
 have $\langle \text{commutant} (\text{commutant} ((\lambda a. \text{id-cblinfun} \otimes_o a) \ \prime \ X)) \rangle$
 $= \text{commutant} (\text{commutant} (\text{sandwich swap-ell2} \ \prime \ (\lambda a. a \otimes_o \text{id-cblinfun}) \ \prime \ X)) \rangle$
 by (*simp add: swap-tensor-op-sandwich image-image*)
 also have $\langle \dots = \text{sandwich swap-ell2} \ \prime \ \text{commutant} ((\lambda a. a \otimes_o \text{id-cblinfun}) \ \prime \ X) \rangle$
 by (*simp add: sandwich-unitary-commutant*)
 also have $\langle \dots = \text{sandwich swap-ell2} \ \prime \ (\lambda a. a \otimes_o \text{id-cblinfun}) \ \prime \ \text{commutant} (\text{commutant } X) \rangle$
 by (*simp add: amplification-double-commutant-commute*)
 also have $\langle \dots = (\lambda a. \text{id-cblinfun} \otimes_o a) \ \prime \ \text{commutant} (\text{commutant } X) \rangle$
 by (*simp add: swap-tensor-op-sandwich image-image*)
 finally show ?thesis
 by –
 qed

lemma *commutant-cspan*: $\langle \text{commutant} (\text{cspan } A) = \text{commutant } A \rangle$
 by (*meson basic-trans-rules(24) commutant-antimono complex-vector.span-superset cspan-in-double-commutant dual-order.trans*)

lemma *double-commutant-grows'*: $\langle x \in X \implies x \in \text{commutant} (\text{commutant } X) \rangle$
using *double-commutant-grows by blast*

16.2 Double commutant theorem

fun *inflation-op'* :: $\langle \text{nat} \Rightarrow ('a \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2}) \text{ list} \Rightarrow ('a \times \text{nat}) \text{ ell2} \Rightarrow_{CL} ('b \times \text{nat}) \text{ ell2} \rangle$
where

$\langle \text{inflation-op}' n \text{ Nil} = 0 \rangle$
 $| \langle \text{inflation-op}' n (a \# as) = (a \otimes_o \text{butterfly} (\text{ket } n) (\text{ket } n)) + \text{inflation-op}' (n+1) as \rangle$

abbreviation $\langle \text{inflation-op} \equiv \text{inflation-op}' 0 \rangle$

fun *inflation-state'* :: $\langle \text{nat} \Rightarrow 'a \text{ ell2} \text{ list} \Rightarrow ('a \times \text{nat}) \text{ ell2} \rangle$ **where**

$\langle \text{inflation-state}' n \text{ Nil} = 0 \rangle$
 $| \langle \text{inflation-state}' n (a \# as) = (a \otimes_s \text{ket } n) + \text{inflation-state}' (n+1) as \rangle$

abbreviation $\langle \text{inflation-state} \equiv \text{inflation-state}' 0 \rangle$

fun *inflation-space'* :: $\langle \text{nat} \Rightarrow 'a \text{ ell2} \text{ ccspace} \text{ list} \Rightarrow ('a \times \text{nat}) \text{ ell2} \text{ ccspace} \rangle$ **where**

$\langle \text{inflation-space}' n \text{ Nil} = 0 \rangle$
 $| \langle \text{inflation-space}' n (S \# Ss) = (S \otimes_S \text{ccspan} \{\text{ket } n\}) + \text{inflation-space}' (n+1) Ss \rangle$

abbreviation $\langle \text{inflation-space} \equiv \text{inflation-space}' 0 \rangle$

definition *inflation-carrier* :: $\langle \text{nat} \Rightarrow ('a \times \text{nat}) \text{ ell2} \text{ ccspace} \rangle$ **where**

$\langle \text{inflation-carrier } n = \text{inflation-space} (\text{replicate } n \top) \rangle$

definition *inflation-op-carrier* :: $\langle \text{nat} \Rightarrow (('a \times \text{nat}) \text{ ell2} \Rightarrow_{CL} ('b \times \text{nat}) \text{ ell2}) \text{ set} \rangle$ **where**

$\langle \text{inflation-op-carrier } n = \{ \text{Proj} (\text{inflation-carrier } n) \text{ o}_{CL} a \text{ o}_{CL} \text{Proj} (\text{inflation-carrier } n) \mid a. \text{True} \} \rangle$

lemma *inflation-op-compose-outside*: $\langle \text{inflation-op}' m \text{ ops} \text{ o}_{CL} (a \otimes_o \text{butterfly} (\text{ket } n) (\text{ket } n)) = 0 \rangle$ **if** $\langle n < m \rangle$

using that apply (*induction ops arbitrary: m*)

by (*auto simp: cblinfun-compose-add-left comp-tensor-op cinner-ket*)

lemma *inflation-op-compose-outside-rev*: $\langle (a \otimes_o \text{butterfly} (\text{ket } n) (\text{ket } n)) \text{ o}_{CL} \text{inflation-op}' m \text{ ops} = 0 \rangle$ **if** $\langle n < m \rangle$

using that apply (*induction ops arbitrary: m*)

by (*auto simp: cblinfun-compose-add-right comp-tensor-op cinner-ket*)

lemma *Proj-inflation-carrier*: $\langle \text{Proj} (\text{inflation-carrier } n) = \text{inflation-op} (\text{replicate } n \text{id-cblinfun}) \rangle$

proof –

have $\langle \text{Proj} (\text{inflation-space}' m (\text{replicate } n \top)) = \text{inflation-op}' m (\text{replicate } n \text{id-cblinfun}) \rangle$ **for** m

proof (*induction n arbitrary: m*)

case 0

then show *?case*


```

    by simp
  next
  case (Suc n)
  have *: ⟨orthogonal-spaces (( $\top$  :: 'b ell2 ccspace)  $\otimes_S$  ccspan {ket m}) (inflation-space'
(Suc m) (replicate n  $\top$ ))⟩
  by (auto simp add: orthogonal-projectors-orthogonal-spaces Suc tensor-ccsubspace-via-Proj
Proj-on-own-range is-Proj-tensor-op inflation-op-compose-outside-rev butterfly-is-Proj
simp flip: butterfly-eq-proj)
  show ?case
  apply (simp add: Suc * Proj-sup)
  by (metis (no-types, opaque-lifting) Proj-is-Proj Proj-on-own-range Proj-top
butterfly-eq-proj is-Proj-tensor-op norm-ket tensor-ccsubspace-via-Proj)
qed
then show ?thesis
by (force simp add: inflation-carrier-def)
qed

```

lemma *inflation-op-carrierI*:

```

  assumes ⟨Proj (inflation-carrier n)  $o_{CL}$  a  $o_{CL}$  Proj (inflation-carrier n) = a⟩
  shows ⟨a  $\in$  inflation-op-carrier n⟩
  using assms by (auto intro!: exI[of - a] simp add: inflation-op-carrier-def)

```

lemma *inflation-op-compose*: ⟨inflation-op' n ops1 o_{CL} inflation-op' n ops2 = inflation-op' n
(map2 cblinfun-compose ops1 ops2)⟩

proof (induction ops2 arbitrary: ops1 n)

case Nil

then show ?case by simp

next

case (Cons op ops2)

note IH = this

fix ops1 :: ⟨('c ell2 \Rightarrow_{CL} 'b ell2) list⟩

show ⟨inflation-op' n ops1 o_{CL} inflation-op' n (op # ops2) =
inflation-op' n (map2 (o_{CL}) ops1 (op # ops2))⟩

proof (cases ops1)

case Nil

then show ?thesis

by simp

next

case (Cons a list)

then show ?thesis

by (simp add: cblinfun-compose-add-right cblinfun-compose-add-left tensor-op-ell2
inflation-op-compose-outside comp-tensor-op inflation-op-compose-outside-rev
flip: IH)

qed

qed

lemma *inflation-op-in-carrier*: ⟨inflation-op ops \in inflation-op-carrier n⟩ if ⟨length ops \leq n⟩

apply (rule inflation-op-carrierI)

using that

by (*simp add: Proj-inflation-carrier inflation-op-carrier-def inflation-op-compose zip-replicate1 zip-replicate2 o-def*)

lemma *inflation-op'-apply-tensor-outside*: $\langle n < m \implies \text{inflation-op}' m \text{ as } *_V (v \otimes_s \text{ket } n) = 0 \rangle$
apply (*induction as arbitrary: m*)
by (*auto simp: cblinfun.add-left tensor-op-ell2 cinner-ket*)

lemma *inflation-op'-compose-tensor-outside*: $\langle n < m \implies \text{inflation-op}' m \text{ as } o_{CL} \text{ tensor-ell2-right } (\text{ket } n) = 0 \rangle$
apply (*rule cblinfun-eqI*)
by (*simp add: inflation-op'-apply-tensor-outside*)

lemma *inflation-state'-apply-tensor-outside*: $\langle n < m \implies (a \otimes_o \text{butterfly } \psi (\text{ket } n)) *_V \text{inflation-state}' m \text{ vs } = 0 \rangle$
apply (*induction vs arbitrary: m*)
by (*auto simp: cblinfun.add-right tensor-op-ell2 cinner-ket*)

lemma *inflation-op-apply-inflation-state*: $\langle \text{inflation-op}' n \text{ ops } *_V \text{inflation-state}' n \text{ vecs} = \text{inflation-state}' n (\text{map2 cblinfun-apply ops vecs}) \rangle$

proof (*induction vecs arbitrary: ops n*)

case *Nil*

then show *?case by simp*

next

case (*Cons v vecs*)

note *IH = this*

fix *ops :: <'b ell2 \Rightarrow_{CL} 'a ell2> list*

show $\langle \text{inflation-op}' n \text{ ops } *_V \text{inflation-state}' n (v \# \text{vecs}) = \text{inflation-state}' n (\text{map2 } (*_V) \text{ ops } (v \# \text{vecs})) \rangle$

proof (*cases ops*)

case *Nil*

then show *?thesis*

by *simp*

next

case (*Cons a list*)

then show *?thesis*

by (*simp add: cblinfun.add-right cblinfun.add-left tensor-op-ell2*

inflation-op'-apply-tensor-outside inflation-state'-apply-tensor-outside flip: IH)

qed

qed

lemma *inflation-state-in-carrier*: $\langle \text{inflation-state } \text{vecs} \in \text{space-as-set } (\text{inflation-carrier } n) \rangle$ **if** $\langle \text{length } \text{vecs} + m \leq n \rangle$

apply (*rule space-as-setI-via-Proj*)

using *that*

by (*simp add: Proj-inflation-carrier inflation-op-apply-inflation-state zip-replicate1 o-def*)

lemma *inflation-op'-apply-tensor-outside'*: $\langle n \geq \text{length } \text{as} + m \implies \text{inflation-op}' m \text{ as } *_V (v \otimes_s \text{ket } n) = 0 \rangle$

apply (*induction as arbitrary: m*)
by (*auto simp: cblinfun.add-left tensor-op-ell2 cinner-ket*)

lemma *Proj-inflation-carrier-outside*: $\langle \text{Proj} (\text{inflation-carrier } n) *_V (\psi \otimes_s \text{ket } i) = 0 \rangle$ **if** $\langle i \geq n \rangle$
by (*simp add: Proj-inflation-carrier inflation-op'-apply-tensor-outside' that*)

lemma *inflation-state'-is-orthogonal-outside*: $\langle n < m \implies \text{is-orthogonal} (a \otimes_s \text{ket } n) (\text{inflation-state}' m \text{ vs}) \rangle$
apply (*induction vs arbitrary: m*)
by (*auto simp: cinner-add-right*)

lemma *inflation-op-adj*: $\langle (\text{inflation-op}' n \text{ ops})^* = \text{inflation-op}' n (\text{map adj ops}) \rangle$
apply (*induction ops arbitrary: n*)
by (*simp-all add: adj-plus tensor-op-adjoint*)

lemma *inflation-state0*:
assumes $\langle \bigwedge v. v \in \text{set } f \implies v = 0 \rangle$
shows $\langle \text{inflation-state}' n f = 0 \rangle$
using *assms* **apply** (*induction f arbitrary: n*)
apply *simp*
using *tensor-ell2-0-left* **by** *force*

lemma *inflation-state-plus*:
assumes $\langle \text{length } f = \text{length } g \rangle$
shows $\langle \text{inflation-state}' n f + \text{inflation-state}' n g = \text{inflation-state}' n (\text{map2 plus } f g) \rangle$
using *assms* **apply** (*induction f g arbitrary: n rule: list-induct2*)
by (*auto simp: algebra-simps tensor-ell2-add1*)

lemma *inflation-state-minus*:
assumes $\langle \text{length } f = \text{length } g \rangle$
shows $\langle \text{inflation-state}' n f - \text{inflation-state}' n g = \text{inflation-state}' n (\text{map2 minus } f g) \rangle$
using *assms* **apply** (*induction f g arbitrary: n rule: list-induct2*)
by (*auto simp: algebra-simps tensor-ell2-diff1*)

lemma *inflation-state-scaleC*:
shows $\langle c *_C \text{inflation-state}' n f = \text{inflation-state}' n (\text{map } (\text{scaleC } c) f) \rangle$
apply (*induction f arbitrary: n*)
by (*auto simp: algebra-simps tensor-ell2-scaleC1*)

lemma *inflation-op-compose-tensor-ell2-right*:
assumes $\langle i \geq n \rangle$ **and** $\langle i < n + \text{length } f \rangle$
shows $\langle \text{inflation-op}' n f \text{ o}_{CL} \text{tensor-ell2-right } (\text{ket } i) = \text{tensor-ell2-right } (\text{ket } i) \text{ o}_{CL} (f!(i-n)) \rangle$
proof (*insert assms, induction f arbitrary: n*)
case *Nil*
then show *?case*
by *simp*
next

```

case (Cons a f)
show ?case
proof (cases ⟨i = n⟩)
  case True
  have ⟨a ⊗o butterfly (ket n) (ket n) oCL tensor-ell2-right (ket n) = tensor-ell2-right (ket n)
oCL a⟩
  apply (rule cblinfun-eqI)
  by (simp add: tensor-op-ell2 cinner-ket)
  with True show ?thesis
  by (simp add: cblinfun-compose-add-left inflation-op'-compose-tensor-outside)
next
case False
with Cons.prem1 have 1: ⟨Suc n ≤ i⟩
  by presburger
have 2: ⟨a ⊗o butterfly (ket n) (ket n) oCL tensor-ell2-right (ket i) = 0⟩
  apply (rule cblinfun-eqI)
  using False by (simp add: tensor-op-ell2 cinner-ket)
show ?thesis
  using Cons.prem1
  by (simp add: cblinfun-compose-add-left Cons.IH[where n=⟨Suc n⟩] 2)
qed
qed

```

lemma *inflation-op-apply*:

```

assumes ⟨i ≥ n⟩ and ⟨i < n + length f⟩
shows ⟨inflation-op' n f *V (ψ ⊗s ket i) = (f!(i-n) *V ψ) ⊗s ket i⟩
by (simp add: inflation-op-compose-tensor-ell2-right assms
  flip: tensor-ell2-right-apply cblinfun-apply-cblinfun-compose)

```

lemma *norm-inflation-state*:

```

⟨norm (inflation-state' n f) = sqrt (∑ v←f. (norm v)2)⟩
proof -
  have ⟨(norm (inflation-state' n f))2 = (∑ v←f. (norm v)2)⟩
  proof (induction f arbitrary: n)
    case Nil
    then show ?case by simp
  next
  case (Cons v f)
  have ⟨(norm (inflation-state' n (v # f)))2 = (norm (v ⊗s ket n + inflation-state' (Suc n)
f))2⟩
  by simp
  also have ⟨... = (norm (v ⊗s ket n))2 + (norm (inflation-state' (Suc n) f))2⟩
  apply (rule pythagorean-theorem)
  apply (rule inflation-state'-is-orthogonal-outside)
  by simp
  also have ⟨... = (norm (v ⊗s ket n))2 + (∑ v←f. (norm v)2)⟩
  by (simp add: Cons.IH)
  also have ⟨... = (norm v)2 + (∑ v←f. (norm v)2)⟩
  by (simp add: norm-tensor-ell2)

```

also have $\langle \dots = (\sum v \leftarrow v \# f. (\text{norm } v)^2) \rangle$
 by *simp*
 finally show *?case*
 by –
 qed
 then show *?thesis*
 by (*simp add: real-sqrt-unique*)
 qed

lemma *cstrong-operator-topology-in-closure-algebraicI*:

— [2], Proposition IX.5.3
 assumes *space*: $\langle \text{csubspace } A \rangle$
 assumes *mult*: $\langle \bigwedge a \ a'. \ a \in A \implies a' \in A \implies a \ o_{CL} \ a' \in A \rangle$
 assumes *one*: $\langle \text{id-cblinfun} \in A \rangle$
 assumes *main*: $\langle \bigwedge n \ S. \ S \leq \text{inflation-carrier } n \implies (\bigwedge a. \ a \in A \implies \text{inflation-op } (\text{replicate } n \ a) \ *_S \ S \leq S) \implies$
 $\text{inflation-op } (\text{replicate } n \ b) \ *_S \ S \leq S \rangle$
 shows $\langle b \in \text{cstrong-operator-topology-closure-of } A \rangle$
proof (*rule cstrong-operator-topology-in-closureI*)
 fix *F* :: $\langle 'a \ \text{ell2 set} \rangle$ and $\varepsilon :: \text{real}$
 assume $\langle \text{finite } F \rangle$ and $\langle \varepsilon > 0 \rangle$
 obtain *f* where $\langle \text{set } f = F \rangle$ and $\langle \text{distinct } f \rangle$
 using $\langle \text{finite } F \rangle$ *finite-distinct-list* by *blast*
 define *n* *M'* *M* where $\langle n = \text{length } f \rangle$
 and $\langle M' = ((\lambda a. \ \text{inflation-state } (\text{map } (\text{cblinfun-apply } a) \ f)) \ 'A) \rangle$
 and $\langle M = \text{ccspan } M' \rangle$
 have *M-carrier*: $\langle M \leq \text{inflation-carrier } n \rangle$
proof –
 have $\langle M' \subseteq \text{space-as-set } (\text{inflation-carrier } n) \rangle$
 by (*auto intro!: inflation-state-in-carrier simp add: M'-def n-def*)
 then show *?thesis*
 by (*simp add: M-def ccspan-leqI*)
 qed

have $\langle \text{inflation-op } (\text{replicate } n \ a) \ *_S \ M \leq M \rangle$ if $\langle a \in A \rangle$ for *a*

proof (*unfold M-def, rule cblinfun-image-ccspan-leqI*)

fix *v* assume $\langle v \in M' \rangle$
 then obtain *a'* where $\langle a' \in A \rangle$ and *v-def*: $\langle v = \text{inflation-state } (\text{map } (\text{cblinfun-apply } a') \ f) \rangle$
 using *M'-def* by *blast*
 then have $\langle \text{inflation-op } (\text{replicate } n \ a) \ *_V \ v = \text{inflation-state } (\text{map } ((*_V) \ (a \ o_{CL} \ a')) \ f) \rangle$
 by (*simp add: v-def n-def inflation-op-apply-inflation-state map2-map-map*
 flip: cblinfun-apply-cblinfun-compose map-replicate-const)
 also have $\langle \dots \in M' \rangle$
 using *M'-def* $\langle a' \in A \rangle \langle a \in A \rangle$ *mult*
 by *simp*
 also have $\langle \dots \subseteq \text{space-as-set } (\text{ccspan } M') \rangle$
 by (*simp add: ccspan-superset*)
 finally show $\langle \text{inflation-op } (\text{replicate } n \ a) \ *_V \ v \in \text{space-as-set } (\text{ccspan } M') \rangle$

```

    by -
  qed
  then have b-invariant: ⟨inflation-op (replicate n b) *S M ≤ M⟩
    using M-carrier by (simp add: main)
  have f-M: ⟨inflation-state f ∈ space-as-set M⟩
  proof -
    have ⟨inflation-state f = inflation-state (map (cblinfun-apply id-cblinfun) f)⟩
      by simp
    also have ⟨... ∈ M'⟩
      using M'-def one by blast
    also have ⟨... ⊆ space-as-set M⟩
      by (simp add: M-def ccspan-superset)
    finally show ?thesis
      by -
  qed
  have ⟨csubspace M'⟩
  proof (rule complex-vector.subspaceI)
    fix c x y
    show ⟨0 ∈ M'⟩
      apply (auto intro!: image-eqI[where x=0] simp add: M'-def)
      apply (subst inflation-state0)
      by (auto simp add: space complex-vector.subspace-0)
    show ⟨x ∈ M' ⟹ y ∈ M' ⟹ x + y ∈ M'⟩
      by (auto intro!: image-eqI[where x=⟨- + -⟩]
          simp add: M'-def inflation-state-plus map2-map-map
          cblinfun.add-left[abs-def] space complex-vector.subspace-add)
    show ⟨c *C x ∈ M' ⟩ if ⟨x ∈ M'⟩
  proof -
    from that
    obtain a where ⟨a ∈ A⟩ and ⟨x = inflation-state (map ((*V) a) f)⟩
      by (auto simp add: M'-def)
    then have ⟨c *C x = inflation-state (map ((*V) (c *C a)) f)⟩
      by (simp add: inflation-state-scaleC o-def scaleC-cblinfun.rep-eq)
    moreover have ⟨c *C a ∈ A⟩
      by (simp add: ⟨a ∈ A⟩ space complex-vector.subspace-scale)
    ultimately show ?thesis
      unfolding M'-def
      by (rule image-eqI)
  qed
  qed
  then have M-closure-M': ⟨space-as-set M = closure M'⟩
    by (metis M-def ccspan.rep-eq complex-vector.span-eq-iff)
  have ⟨inflation-state (map (cblinfun-apply b) f) ∈ space-as-set M⟩
  proof -
    have ⟨map2 (*V) (replicate n b) f = map ((*V) b) f⟩
      using map2-map-map[where h=cblinfun-apply and g=id and f=⟨λ-. b⟩ and xs=f]
      by (simp add: n-def flip: map-replicate-const)
    then have ⟨inflation-state (map (cblinfun-apply b) f) = inflation-op (replicate n b) *V
inflation-state f⟩

```

```

    by (simp add: inflation-op-apply-inflation-state)
  also have ⟨... ∈ space-as-set (inflation-op (replicate n b) *S M)⟩
    by (simp add: f-M cblinfun-apply-in-image')
  also have ⟨... ⊆ space-as-set M⟩
    using b-invariant less-eq-ccsubspace.rep-eq by blast
  finally show ?thesis
    by -
qed
then obtain m where ⟨m ∈ M'⟩ and m-close: ⟨norm (m - inflation-state (map (cblinfun-apply
b) f)) ≤ ε⟩
  apply atomize-elim
  apply (simp add: M-closure-M' closure-approachable dist-norm)
  using ⟨ε > 0⟩ by fastforce
from ⟨m ∈ M'⟩
obtain a where ⟨a ∈ A⟩ and m-def: ⟨m = inflation-state (map (cblinfun-apply a) f)⟩
  by (auto simp add: M'-def)
have ⟨(∑ v←f. (norm ((a - b) *V v))2) ≤ ε2⟩
proof -
  have ⟨(∑ v←f. (norm ((a - b) *V v))2) = (norm (inflation-state (map (cblinfun-apply (a
- b)) f))2)⟩
    apply (simp add: norm-inflation-state o-def)
    apply (subst real-sqrt-pow2)
    apply (rule sum-list-nonneg)
    by (auto simp: sum-list-nonneg)
  also have ⟨... = (norm (m - inflation-state (map (cblinfun-apply b) f))2)⟩
    by (simp add: m-def inflation-state-minus map2-map-map cblinfun.diff-left[abs-def])
  also have ⟨... ≤ ε2⟩
    by (simp add: m-close power-mono)
  finally show ?thesis
    by -
qed
then have ⟨(norm ((a - b) *V v))2 ≤ ε2⟩ if ⟨v ∈ F⟩ for v
  using that apply (simp flip: sum.distinct-set-conv-list add: ⟨distinct f⟩)
  by (smt (verit) ⟨finite F⟩ ⟨set f = F⟩ sum-nonneg-leq-bound zero-le-power2)
then show ⟨∃ a ∈ A. ∀ f ∈ F. norm ((b - a) *V f) ≤ ε⟩
  using ⟨0 < ε⟩ ⟨a ∈ A⟩
  by (metis cblinfun.real.diff-left norm-minus-commute power2-le-imp-le power-eq-0-iff power-zero-numeral
realpow-pos-nth-unique zero-compare-simps(12))
qed

```

lemma *commutant-inflation*:

— One direction of [2], Proposition IX.6.2.

fixes n

defines $\langle \bigwedge X. \text{commutant}' X \equiv \text{commutant } X \cap \text{inflation-op-carrier } n \rangle$

shows $\langle (\lambda a. \text{inflation-op (replicate } n \ a)) \text{ 'commutant (commutant } A)$

$\subseteq \text{commutant}' (\text{commutant}' ((\lambda a. \text{inflation-op (replicate } n \ a)) \text{ ' } A)) \rangle$

proof (unfold *commutant'*-def, rule subsetI, rule IntI)

fix b

assume $\langle b \in (\lambda a. \text{inflation-op (replicate } n \ a)) \text{ 'commutant (commutant } A) \rangle$

then obtain $b0$ **where** $b\text{-def}$: $\langle b = \text{inflation-op } (\text{replicate } n \ b0) \rangle$ **and** $b0\text{-}A''$: $\langle b0 \in \text{commutant } (\text{commutant } A) \rangle$
by *auto*
show $\langle b \in \text{inflation-op-carrier } n \rangle$
by (*simp add*: $b\text{-def inflation-op-in-carrier}$)
show $\langle b \in \text{commutant } (\text{commutant } ((\lambda a. \text{inflation-op } (\text{replicate } n \ a)) \ 'A) \cap \text{inflation-op-carrier } n) \rangle$
proof (*rule commutant-memberI*)
fix c
assume $\langle c \in \text{commutant } ((\lambda a. \text{inflation-op } (\text{replicate } n \ a)) \ 'A) \cap \text{inflation-op-carrier } n \rangle$
then have $c\text{-comm}$: $\langle c \in \text{commutant } ((\lambda a. \text{inflation-op } (\text{replicate } n \ a)) \ 'A) \rangle$
and $c\text{-carr}$: $\langle c \in \text{inflation-op-carrier } n \rangle$
by *auto*
define c' **where** $\langle c' \ i \ j = (\text{tensor-ell2-right } (\text{ket } i))^* \ o_{CL} \ c \ o_{CL} \ \text{tensor-ell2-right } (\text{ket } j) \rangle$
for $i \ j$
have $\langle c' \ i \ j \ o_{CL} \ a = a \ o_{CL} \ c' \ i \ j \rangle$ **if** $\langle a \in A \rangle$ **and** $\langle i < n \rangle$ **and** $\langle j < n \rangle$ **for** $a \ i \ j$
proof –
from $c\text{-comm}$ **have** $\langle c \ o_{CL} \ \text{inflation-op } (\text{replicate } n \ a) = \text{inflation-op } (\text{replicate } n \ a) \ o_{CL} \ c \rangle$
using *that by* (*auto simp*: commutant-def)
then have $\langle (\text{tensor-ell2-right } (\text{ket } i))^* \ o_{CL} \ c \ o_{CL} \ (\text{inflation-op } (\text{replicate } n \ a) \ o_{CL} \ \text{tensor-ell2-right } (\text{ket } j)) \rangle$
 $= \langle (\text{inflation-op } (\text{replicate } n \ (a^*)) \ o_{CL} \ (\text{tensor-ell2-right } (\text{ket } i)))^* \ o_{CL} \ c \ o_{CL} \ \text{tensor-ell2-right } (\text{ket } j) \rangle$
apply (*simp add*: inflation-op-adj)
by (*metis* (no-types , lifting) $\text{lift-cblinfun-comp}(2)$)
then show *?thesis*
apply (*subst* (asm) $\text{inflation-op-compose-tensor-ell2-right}$)
apply (*simp*, *simp add*: that)
apply (*subst* (asm) $\text{inflation-op-compose-tensor-ell2-right}$)
apply (*simp*, *simp add*: that)
by (*simp add*: $\text{that } c'\text{-def cblinfun-compose-assoc}$)
qed
then have $\langle c' \ i \ j \in \text{commutant } A \rangle$ **if** $\langle i < n \rangle$ **and** $\langle j < n \rangle$ **for** $i \ j$
using *that by* (*simp add*: commutant-memberI)
with $b0\text{-}A''$ **have** $b0\text{-}c'$: $\langle b0 \ o_{CL} \ c' \ i \ j = c' \ i \ j \ o_{CL} \ b0 \rangle$ **if** $\langle i < n \rangle$ **and** $\langle j < n \rangle$ **for** $i \ j$
using *that by* (*simp add*: commutant-def)

from $c\text{-carr}$ **obtain** c'' **where** c'' : $\langle c = \text{Proj } (\text{inflation-carrier } n) \ o_{CL} \ c'' \ o_{CL} \ \text{Proj } (\text{inflation-carrier } n) \rangle$
by (*auto simp add*: $\text{inflation-op-carrier-def}$)

have $c0$: $\langle c \ *_{\mathcal{V}} \ (\psi \otimes_s \ \text{ket } i) = 0 \rangle$ **if** $\langle i \geq n \rangle$ **for** $i \ \psi$
using *that by* (*simp add*: $c'' \ \text{Proj-inflation-carrier-outside}$)
have $\text{cadj}0$: $\langle c \ *_{\mathcal{V}} \ (\psi \otimes_s \ \text{ket } j) = 0 \rangle$ **if** $\langle j \geq n \rangle$ **for** $j \ \psi$
using *that by* (*simp add*: $c'' \ \text{adj-Proj Proj-inflation-carrier-outside}$)

have $\langle \text{inflation-op } (\text{replicate } n \ b0) \ o_{CL} \ c = c \ o_{CL} \ \text{inflation-op } (\text{replicate } n \ b0) \rangle$
proof (*rule equal-ket*, *rule cinner-ket-eqI*)


```

fix ii jj
obtain  $i' j' :: 'a$  and  $i j :: nat$  where  $ii\text{-def}: \langle ii = (i', i) \rangle$  and  $jj\text{-def}: \langle jj = (j', j) \rangle$ 
  by force
show  $\langle ket\ ii \cdot_C ((inflation\text{-op}\ (replicate\ n\ b0)\ o_{CL}\ c) *_V\ ket\ jj) =$ 
   $ket\ ii \cdot_C ((c\ o_{CL}\ inflation\text{-op}\ (replicate\ n\ b0)) *_V\ ket\ jj) \rangle$ 
proof (cases  $\langle i < n \wedge j < n \rangle$ )
  case True
    have  $\langle ket\ ii \cdot_C ((inflation\text{-op}\ (replicate\ n\ b0)\ o_{CL}\ c) *_V\ ket\ jj) = ((b0 *_V\ ket\ i') \otimes_s$ 
       $ket\ i) \cdot_C (c *_V\ ket\ j' \otimes_s\ ket\ j) \rangle$ 
    using True by (simp add:  $ii\text{-def}\ jj\text{-def}\ inflation\text{-op}\text{-adj}\ inflation\text{-op}\text{-apply}\ flip: tensor\text{-ell2}\text{-inner}\text{-prod}$ 
       $flip: tensor\text{-ell2}\text{-ket}\ cinner\text{-adj}\text{-left}[\text{where } G = \langle inflation\text{-op } \rightarrow \rangle]$ )
    also have  $\langle \dots = (ket\ i' \otimes_s\ ket\ i) \cdot_C (c *_V\ (b0 *_V\ ket\ j') \otimes_s\ ket\ j) \rangle$ 
    using  $b0\text{-}c'$  apply (simp add:  $c'\text{-def}\ flip: tensor\text{-ell2}\text{-right}\text{-apply}\ cinner\text{-adj}\text{-right}$ )
    by (metis (no-types, lifting) True  $simp\text{-}a\text{-}o_{CL}\text{-}b'$ )
    also have  $\langle \dots = ket\ ii \cdot_C ((c\ o_{CL}\ inflation\text{-op}\ (replicate\ n\ b0)) *_V\ ket\ jj) \rangle$ 
  by (simp add: True  $ii\text{-def}\ jj\text{-def}\ inflation\text{-op}\text{-adj}\ inflation\text{-op}\text{-apply}\ flip: tensor\text{-ell2}\text{-inner}\text{-prod}$ 
     $flip: tensor\text{-ell2}\text{-ket}\ cinner\text{-adj}\text{-left}[\text{where } G = \langle inflation\text{-op } \rightarrow \rangle]$ )
  finally show ?thesis
    by -
  next
  case False
  then show ?thesis
    apply (auto simp add:  $ii\text{-def}\ jj\text{-def}\ inflation\text{-op}\text{-adj}\ c0\ inflation\text{-op}'\text{-apply}\text{-tensor}\text{-outside}'$ 
       $simp\ flip: tensor\text{-ell2}\text{-ket}\ cinner\text{-adj}\text{-left}[\text{where } G = \langle inflation\text{-op } \rightarrow \rangle]$ )
    by (simp add:  $cadj0\ flip: cinner\text{-adj}\text{-left}[\text{where } G = c]$ )
qed
qed
then show  $\langle b\ o_{CL}\ c = c\ o_{CL}\ b \rangle$ 
  by (simp add:  $b\text{-def}$ )
qed
qed

```

lemma *double-commutant-theorem-aux:*

— Basically the double commutant theorem, except that we restricted to spaces of the form $'a\ ell2$

— [2], Proposition IX.6.4

fixes $A :: \langle ('a\ ell2 \Rightarrow_{CL} 'a\ ell2)\ set \rangle$

assumes $\langle csubspace\ A \rangle$

assumes $\langle \bigwedge a\ a'. a \in A \implies a' \in A \implies a\ o_{CL}\ a' \in A \rangle$

assumes $\langle id\text{-}cblinfun \in A \rangle$

assumes $\langle \bigwedge a. a \in A \implies a * \in A \rangle$

shows $\langle commutant\ (commutant\ A) = cstrong\text{-operator}\text{-topology}\ \text{closure}\text{-of}\ A \rangle$

proof (intro *Set.set-eqI* iffI)

show $\langle x \in commutant\ (commutant\ A) \rangle$ **if** $\langle x \in cstrong\text{-operator}\text{-topology}\ \text{closure}\text{-of}\ A \rangle$ **for** x

using *closure-of-minimal commutant-sot-closed double-commutant-grows that* **by** *blast*

next

show $\langle b \in cstrong\text{-operator}\text{-topology}\ \text{closure}\text{-of}\ A \rangle$ **if** $b \in A''$: $\langle b \in commutant\ (commutant\ A) \rangle$
for b

proof (*rule cstrong-operator-topology-in-closure-algebraicI*)
show $\langle \text{csubspace } A \rangle$ **and** $\langle a \in A \implies a' \in A \implies a \text{ } o_{CL} \text{ } a' \in A \rangle$ **and** $\langle \text{id-cblinfun} \in A \rangle$ **for**
 $a \text{ } a'$
using *assms by auto*
fix $n \text{ } M$
assume *asm*: $\langle a \in A \implies \text{inflation-op} (\text{replicate } n \text{ } a) *_S M \leq M \rangle$ **for** a
assume *M-carrier*: $\langle M \leq \text{inflation-carrier } n \rangle$
define *commutant'* **where** $\langle \text{commutant}' X = \text{commutant } X \cap \text{inflation-op-carrier } n \rangle$ **for** X
 $:: \langle (('a \times \text{nat}) \text{ ell2} \implies_{CL} ('a \times \text{nat}) \text{ ell2}) \text{ set} \rangle$
define An **where** $\langle An = (\lambda a. \text{inflation-op} (\text{replicate } n \text{ } a)) ' A \rangle$
have $*$: $\langle \text{Proj } M \text{ } o_{CL} (\text{inflation-op} (\text{replicate } n \text{ } a) \text{ } o_{CL} \text{ } \text{Proj } M) = \text{inflation-op} (\text{replicate } n$
 $a) \text{ } o_{CL} \text{ } \text{Proj } M \rangle$ **if** $\langle a \in A \rangle$ **for** a
apply (*rule Proj-compose-cancelI*)
using *asm that by (simp add: cblinfun-compose-image)*
have $\langle \text{Proj } M \text{ } o_{CL} \text{inflation-op} (\text{replicate } n \text{ } a) = \text{inflation-op} (\text{replicate } n \text{ } a) \text{ } o_{CL} \text{ } \text{Proj } M \rangle$ **if**
 $\langle a \in A \rangle$ **for** a
proof –
have $\langle \text{Proj } M \text{ } o_{CL} \text{inflation-op} (\text{replicate } n \text{ } a) = (\text{inflation-op} (\text{replicate } n \text{ } (a*))) \text{ } o_{CL} \text{ } \text{Proj}$
 $M \rangle*$
by (*simp add: inflation-op-adj adj-Proj*)
also have $\langle \dots = (\text{Proj } M \text{ } o_{CL} \text{inflation-op} (\text{replicate } n \text{ } (a*))) \text{ } o_{CL} \text{ } \text{Proj } M \rangle*$
apply (*subst *[symmetric]*)
by (*simp-all add: that assms flip: cblinfun-compose-assoc*)
also have $\langle \dots = \text{Proj } M \text{ } o_{CL} \text{inflation-op} (\text{replicate } n \text{ } a) \text{ } o_{CL} \text{ } \text{Proj } M \rangle$
by (*simp add: inflation-op-adj adj-Proj cblinfun-compose-assoc*)
also have $\langle \dots = \text{inflation-op} (\text{replicate } n \text{ } a) \text{ } o_{CL} \text{ } \text{Proj } M \rangle$
apply (*subst *[symmetric]*)
by (*simp-all add: that flip: cblinfun-compose-assoc*)
finally show *?thesis*
by –
qed
then have $\langle \text{Proj } M \in \text{commutant}' An \rangle$
using *M-carrier*
apply (*auto intro!: inflation-op-carrierI simp add: An-def commutant-def commutant'-def*)
by (*metis Proj-compose-cancelI Proj-range adj-Proj adj-cblinfun-compose*)
from $b-A''$ **have** $\langle \text{inflation-op} (\text{replicate } n \text{ } b) \in \text{commutant}' (\text{commutant}' An) \rangle$
using *commutant-inflation[of n A, folded commutant'-def]*
by (*auto simp add: An-def commutant'-def*)
with $\langle \text{Proj } M \in \text{commutant}' An \rangle$
have $*$: $\langle \text{inflation-op} (\text{replicate } n \text{ } b) \text{ } o_{CL} \text{ } \text{Proj } M = \text{Proj } M \text{ } o_{CL} \text{inflation-op} (\text{replicate } n \text{ } b) \rangle$
by (*simp add: commutant-def commutant'-def*)
show $\langle \text{inflation-op} (\text{replicate } n \text{ } b) *_S M \leq M \rangle$
proof –
have $\langle \text{inflation-op} (\text{replicate } n \text{ } b) *_S M = (\text{inflation-op} (\text{replicate } n \text{ } b) \text{ } o_{CL} \text{ } \text{Proj } M) *_S \top \rangle$
by (*metis lift-cblinfun-comp(3) Proj-range*)
also have $\langle \dots = (\text{Proj } M \text{ } o_{CL} \text{inflation-op} (\text{replicate } n \text{ } b)) *_S \top \rangle$
by (*simp add: **)
also have $\langle \dots \leq M \rangle$
by (*metis lift-cblinfun-comp(3) Proj-image-leq*)

```

    finally show ?thesis
      by -
    qed
  qed
qed

```

lemma *double-commutant-theorem-aux2*:

— Basically the double commutant theorem, except that we restricted to spaces of typeclass *not-singleton*

— [2], Proposition IX.6.4

```

fixes A :: ⟨('a::{chilbert-space,not-singleton}) ⇒CL 'a⟩ set
assumes subspace: ⟨csubspace A⟩
assumes mult: ⟨∧ a a'. a ∈ A ⇒ a' ∈ A ⇒ a oCL a' ∈ A⟩
assumes id: ⟨id-cblinfun ∈ A⟩
assumes adj: ⟨∧ a. a ∈ A ⇒ a* ∈ A⟩
shows ⟨commutant (commutant A) = cstrong-operator-topology closure-of A⟩

```

proof —

```

define A' :: ⟨('a chilbert2ell2 ell2 ⇒CL 'a chilbert2ell2 ell2) set⟩
  where ⟨A' = sandwich (ell2-to-hilbert*) ' A⟩
have subspace: ⟨csubspace A'⟩
  using subspace by (auto intro!: complex-vector.linear-subspace-image simp: A'-def)
have mult: ⟨∧ a a'. a ∈ A' ⇒ a' ∈ A' ⇒ a oCL a' ∈ A'⟩
  using mult by (auto simp add: A'-def sandwich-arg-compose unitary-ell2-to-hilbert)
have id: ⟨id-cblinfun ∈ A'⟩
  using id by (auto intro!: image-eqI simp add: A'-def sandwich-isometry-id unitary-ell2-to-hilbert)
have adj: ⟨∧ a. a ∈ A' ⇒ a* ∈ A'⟩
  using adj by (auto intro!: image-eqI simp: A'-def simp flip: sandwich-apply-adj)
have homeo: ⟨homeomorphic-map cstrong-operator-topology cstrong-operator-topology
  ((*V) (sandwich ell2-to-hilbert*))⟩
  by (auto intro!: continuous-intros homeomorphic-maps-imp-map[where g=⟨sandwich (ell2-to-hilbert*)⟩]
  simp: homeomorphic-maps-def unitary-ell2-to-hilbert
  simp flip: cblinfun-apply-cblinfun-compose sandwich-compose)
have ⟨commutant (commutant A) = cstrong-operator-topology closure-of A'⟩
  using subspace mult id adj by (rule double-commutant-theorem-aux)
then have ⟨sandwich ell2-to-hilbert ' commutant (commutant A) = sandwich ell2-to-hilbert '
  (cstrong-operator-topology closure-of A')⟩
  by simp
then show ?thesis
  by (simp add: A'-def unitary-ell2-to-hilbert sandwich-unitary-commutant image-image homeo
  flip: cblinfun-apply-cblinfun-compose sandwich-compose
  homeomorphic-map-closure-of[where Y=cstrong-operator-topology])

```

qed

lemma *double-commutant-theorem*:

— [2], Proposition IX.6.4

```

fixes A :: ⟨('a::{chilbert-space}) ⇒CL 'a⟩ set
assumes subspace: ⟨csubspace A⟩
assumes mult: ⟨∧ a a'. a ∈ A ⇒ a' ∈ A ⇒ a oCL a' ∈ A⟩
assumes id: ⟨id-cblinfun ∈ A⟩

```

```

assumes adj:  $\langle \bigwedge a. a \in A \implies a^* \in A \rangle$ 
shows  $\langle \text{commutant } (\text{commutant } A) = \text{cstrong-operator-topology closure-of } A \rangle$ 
proof (cases  $\langle UNIV = \{0 :: 'a\} \rangle$ )
  case True
    then have  $\langle (x :: 'a) = 0 \rangle$  for x
      by auto
    then have UNIV-0:  $\langle UNIV = \{0 :: 'a \Rightarrow_{CL} 'a\} \rangle$ 
      by (auto intro!; cblinfun-eqI)
    have  $\langle 0 \in \text{commutant } (\text{commutant } A) \rangle$ 
      using complex-vector.subspace-0 csubspace-commutant by blast
    then have 1:  $\langle \text{commutant } (\text{commutant } A) = UNIV \rangle$ 
      using UNIV-0
      by force
    have  $\langle 0 \in A \rangle$ 
      by (simp add: assms(1) complex-vector.subspace-0)
    then have  $\langle 0 \in \text{cstrong-operator-topology closure-of } A \rangle$ 
      by (simp add: assms(1) complex-vector.subspace-0)
    then have 2:  $\langle \text{cstrong-operator-topology closure-of } A = UNIV \rangle$ 
      using UNIV-0
      by force
    from 1 2 show ?thesis
      by simp
  next
    case False
      note aux2 = double-commutant-theorem-aux2 [where 'a =  $\langle 'z :: \{\text{chilbert-space, not-singleton}\} \rangle$ ,
      rule-format, internalize-sort  $\langle 'z :: \{\text{chilbert-space, not-singleton}\} \rangle$ ]
      have hilbert:  $\langle \text{class.chilbert-space } (*_R) (*_C) (+) (0 :: 'a) (-) \text{uminus dist norm sgn uniformity open } (*_C) \rangle$ 
        by (rule chilbert-space-class.chilbert-space-axioms)
      from False
      have not-singleton:  $\langle \text{class.not-singleton TYPE('a)} \rangle$ 
        by (rule class-not-singletonI-monoid-add)
      show ?thesis
        apply (rule aux2)
        using assms hilbert not-singleton by auto

```

qed

hide-fact *double-commutant-theorem-aux double-commutant-theorem-aux2*

lemma *double-commutant-theorem-span*:

fixes *A* :: $\langle ('a :: \{\text{chilbert-space}\} \Rightarrow_{CL} 'a) \text{ set} \rangle$

assumes *mult*: $\langle \bigwedge a a'. a \in A \implies a' \in A \implies a \circ_{CL} a' \in A \rangle$

assumes *id*: $\langle \text{id-cblinfun} \in A \rangle$

assumes *adj*: $\langle \bigwedge a. a \in A \implies a^* \in A \rangle$

shows $\langle \text{commutant } (\text{commutant } A) = \text{cstrong-operator-topology closure-of } (\text{cspan } A) \rangle$

proof –

have $\langle \text{commutant } (\text{commutant } A) = \text{commutant } (\text{commutant } (\text{cspan } A)) \rangle$

by (*simp add: commutant-cspan*)

also have $\langle \dots = \text{cstrong-operator-topology closure-of } (\text{cspan } A) \rangle$

apply (*rule double-commutant-theorem*)
using *assms*
apply (*auto simp: cspan-compose-closed cspan-adj-closed*)
using *complex-vector.span-clauses(1)* **by** *blast*
finally show *?thesis*
by –
qed

16.3 Von Neumann Algebras

definition *one-algebra* :: $\langle 'a \Rightarrow_{CL} 'a::\text{chilbert-space} \rangle$ **set** **where**
 $\langle \text{one-algebra} = \text{range } (\lambda c. c *_C \text{id-cblinfun}) \rangle$

definition *von-neumann-algebra* **where** $\langle \text{von-neumann-algebra } A \longleftrightarrow (\forall a \in A. a^* \in A) \wedge \text{commutant } (\text{commutant } A) = A \rangle$

definition *von-neumann-factor* **where** $\langle \text{von-neumann-factor } A \longleftrightarrow \text{von-neumann-algebra } A \wedge A \cap \text{commutant } A = \text{one-algebra} \rangle$

lemma *von-neumann-algebraI*: $\langle (\bigwedge a. a \in A \implies a^* \in A) \implies \text{commutant } (\text{commutant } A) \subseteq A \implies \text{von-neumann-algebra } A \rangle$ **for** \mathfrak{F}

apply (*auto simp: von-neumann-algebra-def*)
using *double-commutant-grows* **by** *blast*

lemma *von-neumann-factorI*:

assumes $\langle \text{von-neumann-algebra } A \rangle$
assumes $\langle A \cap \text{commutant } A \subseteq \text{one-algebra} \rangle$
shows $\langle \text{von-neumann-factor } A \rangle$

proof –

have *1*: $\langle A \supseteq \text{one-algebra} \rangle$
apply (*subst asm-rl[of A = commutant (commutant A)]*)
using *assms(1) von-neumann-algebra-def* **apply** *blast*
by (*auto simp: commutant-def one-algebra-def*)

have *2*: $\langle \text{commutant } A \supseteq \text{one-algebra} \rangle$
by (*auto simp: commutant-def one-algebra-def*)

from *1 2 assms* **show** *?thesis*
by (*auto simp add: von-neumann-factor-def*)

qed

lemma *commutant-UNIV*: $\langle \text{commutant } (\text{UNIV} :: ('a \Rightarrow_{CL} 'a::\text{chilbert-space}) \text{ set}) = \text{one-algebra} \rangle$

proof –

have *1*: $\langle c *_C \text{id-cblinfun} \in \text{commutant } \text{UNIV} \rangle$ **for** *c*
by (*simp add: commutant-def*)

moreover have *2*: $\langle x \in \text{range } (\lambda c. c *_C \text{id-cblinfun}) \rangle$ **if** *x-comm*: $\langle x \in \text{commutant } \text{UNIV} \rangle$ **for**
x :: $\langle 'a \Rightarrow_{CL} 'a \rangle$

proof –

obtain *B* :: $\langle 'a \text{ set} \rangle$ **where** $\langle \text{is-onb } B \rangle$
using *is-onb-some-chilbert-basis* **by** *blast*

```

have ⟨ $\exists c. x *_V \psi = c *_C \psi$ ⟩ for  $\psi$ 
proof -
  have ⟨ $norm (x *_V \psi) = norm ((x \ o_{CL} \ selfbutter (sgn \ \psi)) *_V \ \psi)$ ⟩
    by (simp add: cnorm-eq-1)
  also have ⟨ $\dots = norm ((selfbutter (sgn \ \psi) \ o_{CL} \ x) *_V \ \psi)$ ⟩
    using x-comm by (simp add: commutant-def del: butterfly-apply)
  also have ⟨ $\dots = norm (proj \ \psi *_V (x *_V \ \psi))$ ⟩
    by (simp add: butterfly-sgn-eq-proj)
  finally have ⟨ $x *_V \ \psi \in \ space\text{-as}\text{-set} (ccspan \ \{\psi\})$ ⟩
    by (metis norm-Prod-apply)
  then show ?thesis
    by (auto simp add: ccspan-finite complex-vector.span-singleton)
qed
then obtain f where f: ⟨ $x *_V \ \psi = f \ \psi *_C \ \psi$ ⟩ for  $\psi$ 
  apply atomize-elim apply (rule choice) by auto

have ⟨ $f \ \psi = f \ \varphi$ ⟩ if ⟨ $\psi \in B$ ⟩ and ⟨ $\varphi \in B$ ⟩ for  $\psi \ \varphi$ 
proof (cases ⟨ $\psi = \varphi$ ⟩)
  case True
    then show ?thesis by simp
next
  case False
    with that ⟨is-onb B⟩
    have [simp]: ⟨ $\psi \cdot_C \ \varphi = 0$ ⟩
      by (auto simp: is-onb-def is-ortho-set-def)
    then have [simp]: ⟨ $\varphi \cdot_C \ \psi = 0$ ⟩
      using is-orthogonal-sym by blast
    from that ⟨is-onb B⟩ have [simp]: ⟨ $\psi \neq 0$ ⟩
      by (auto simp: is-onb-def)
    from that ⟨is-onb B⟩ have [simp]: ⟨ $\varphi \neq 0$ ⟩
      by (auto simp: is-onb-def)

have ⟨ $f (\psi + \varphi) *_C \ \psi + f (\psi + \varphi) *_C \ \varphi = f (\psi + \varphi) *_C (\psi + \varphi)$ ⟩
  by (simp add: complex-vector.vector-space-assms(1))
also have ⟨ $\dots = x *_V (\psi + \varphi)$ ⟩
  by (simp add: f)
also have ⟨ $\dots = x *_V \ \psi + x *_V \ \varphi$ ⟩
  by (simp add: cblinfun.add-right)
also have ⟨ $\dots = f \ \psi *_C \ \psi + f \ \varphi *_C \ \varphi$ ⟩
  by (simp add: f)
finally have *: ⟨ $f (\psi + \varphi) *_C \ \psi + f (\psi + \varphi) *_C \ \varphi = f \ \psi *_C \ \psi + f \ \varphi *_C \ \varphi$ ⟩
  by -
have ⟨ $f (\psi + \varphi) = f \ \psi$ ⟩
  using *[THEN arg-cong[where f=⟨cinner  $\psi$ ⟩]]
  by (simp add: cinner-add-right)
moreover have ⟨ $f (\psi + \varphi) = f \ \varphi$ ⟩
  using *[THEN arg-cong[where f=⟨cinner  $\varphi$ ⟩]]
  by (simp add: cinner-add-right)
ultimately show ⟨ $f \ \psi = f \ \varphi$ ⟩

```

```

    by simp
  qed
  then obtain c where ⟨f ψ = c⟩ if ⟨ψ ∈ B⟩ for ψ
    by meson
  then have ⟨x *V ψ = (c *C id-cblinfun) *V ψ⟩ if ⟨ψ ∈ B⟩ for ψ
    by (simp add: f that)
  then have ⟨x = c *C id-cblinfun⟩
    apply (rule cblinfun-eq-gen-eqI[where G=B])
    using ⟨is-onb B⟩ by (auto simp: is-onb-def)
  then show ⟨x ∈ range (λc. c *C id-cblinfun)⟩
    by (auto)
  qed

  from 1 2 show ?thesis
    by (auto simp: one-algebra-def)
  qed

lemma von-neumann-algebra-UNIV: ⟨von-neumann-algebra UNIV⟩
  by (auto simp: von-neumann-algebra-def commutant-def)

lemma von-neumann-factor-UNIV: ⟨von-neumann-factor UNIV⟩
  by (simp add: von-neumann-factor-def commutant-UNIV von-neumann-algebra-UNIV)

lemma von-neumann-algebra-UNION:
  assumes ⟨∧x. x ∈ X ⇒ von-neumann-algebra (A x)⟩
  shows ⟨von-neumann-algebra (commutant (commutant (∪x∈X. A x)))⟩
  proof (rule von-neumann-algebraI)
    show ⟨commutant (commutant (commutant (∪x∈X. A x)))⟩
      ⊆ commutant (commutant (∪x∈X. A x))
      by (meson commutant-antimono double-commutant-grows)
  next
    fix a
    assume ⟨a ∈ commutant (commutant (∪x∈X. A x))⟩
    then have ⟨a* ∈ adj ‘ commutant (commutant (∪x∈X. A x))⟩
      by simp
    also have ⟨... = commutant (commutant (∪x∈X. adj ‘ A x))⟩
      by (simp add: commutant-adj image-UN)
    also have ⟨... ⊆ commutant (commutant (∪x∈X. A x))⟩
      using assms by (auto simp: von-neumann-algebra-def intro!: commutant-antimono)
    finally show ⟨a* ∈ commutant (commutant (∪x∈X. A x))⟩
      by -
  qed

lemma von-neumann-algebra-union:
  assumes ⟨von-neumann-algebra A⟩
  assumes ⟨von-neumann-algebra B⟩
  shows ⟨von-neumann-algebra (commutant (commutant (A ∪ B)))⟩
  using von-neumann-algebra-UNION[where X=⟨{True,False}⟩ and A=⟨λx. if x then A else

```

$B\rangle]$
by $(\text{auto simp: assms Un-ac}(\beta))$

lemma *von-neumann-algebra-commutant*: $\langle \text{von-neumann-algebra (commutant } A) \rangle$ **if** $\langle \text{von-neumann-algebra } A \rangle$

proof $(\text{rule von-neumann-algebraI})$

show $\langle a^* \in \text{commutant } A \rangle$ **if** $\langle a \in \text{commutant } A \rangle$ **for** a

by $(\text{smt (verit) Set.basic-monos}(\gamma) \langle \text{von-neumann-algebra } A \rangle \text{commutant-adj commutant-antimono double-adj image-iff image-subsetI that von-neumann-algebra-def})$

show $\langle \text{commutant (commutant (commutant } A)) \subseteq \text{commutant } A \rangle$

by *simp*

qed

lemma *von-neumann-algebra-def-sot*:

$\langle \text{von-neumann-algebra } \mathfrak{F} \longleftrightarrow$

$(\forall a \in \mathfrak{F}. a^* \in \mathfrak{F}) \wedge \text{csubspace } \mathfrak{F} \wedge (\forall a \in \mathfrak{F}. \forall b \in \mathfrak{F}. a \text{ } o_{CL} \text{ } b \in \mathfrak{F}) \wedge \text{id-cblinfun} \in \mathfrak{F} \wedge$
 $\text{closedin cstrong-operator-topology } \mathfrak{F} \rangle$

proof $(\text{unfold von-neumann-algebra-def, intro iffI conjI; elim conjE; assumption?})$

assume *comm*: $\langle \text{commutant (commutant } \mathfrak{F}) = \mathfrak{F} \rangle$

from *comm* **show** $\langle \text{closedin cstrong-operator-topology } \mathfrak{F} \rangle$

by $(\text{metis commutant-sot-closed})$

from *comm* **show** $\langle \text{csubspace } \mathfrak{F} \rangle$

by $(\text{metis csubspace-commutant})$

from *comm* **show** $\langle \forall a \in \mathfrak{F}. \forall b \in \mathfrak{F}. a \text{ } o_{CL} \text{ } b \in \mathfrak{F} \rangle$

using *commutant-mult* **by** *blast*

from *comm* **show** $\langle \text{id-cblinfun} \in \mathfrak{F} \rangle$

by *blast*

next

assume *adj*: $\langle \forall a \in \mathfrak{F}. a^* \in \mathfrak{F} \rangle$

assume *subspace*: $\langle \text{csubspace } \mathfrak{F} \rangle$

assume *closed*: $\langle \text{closedin cstrong-operator-topology } \mathfrak{F} \rangle$

assume *mult*: $\langle \forall a \in \mathfrak{F}. \forall b \in \mathfrak{F}. a \text{ } o_{CL} \text{ } b \in \mathfrak{F} \rangle$

assume *id*: $\langle \text{id-cblinfun} \in \mathfrak{F} \rangle$

have $\langle \text{commutant (commutant } \mathfrak{F}) = \text{cstrong-operator-topology closure-of } \mathfrak{F} \rangle$

apply $(\text{rule double-commutant-theorem})$

thm *double-commutant-theorem*[of \mathfrak{F}]

using *subspace subspace mult id adj*

by *simp-all*

also from *closed* **have** $\langle \dots = \mathfrak{F} \rangle$

by $(\text{simp add: closure-of-eq})$

finally show $\langle \text{commutant (commutant } \mathfrak{F}) = \mathfrak{F} \rangle$

by $-$

qed

lemma *double-commutant-hull'*:

assumes $\langle \bigwedge x. x \in X \implies x^* \in X \rangle$

shows $\langle \text{commutant (commutant } X) = \text{von-neumann-algebra hull } X \rangle$

proof (rule antisym)
show $\langle \text{commutant} (\text{commutant } X) \subseteq \text{von-neumann-algebra hull } X \rangle$
apply (subst double-commutant-hull)
apply (rule hull-antimono)
by (simp add: von-neumann-algebra-def)
show $\langle \text{von-neumann-algebra hull } X \subseteq \text{commutant} (\text{commutant } X) \rangle$
apply (rule hull-minimal)
by (simp-all add: von-neumann-algebra-def assms commutant-adj-closed)
qed

lemma commutant-one-algebra: $\langle \text{commutant one-algebra} = \text{UNIV} \rangle$
by (metis commutant-UNIV commutant-empty triple-commutant)

definition tensor-vn (infixr \otimes_{vN} 70) **where**
 $\langle \text{tensor-vn } X Y = \text{commutant} (\text{commutant} ((\lambda a. a \otimes_o \text{id-cblinfun}) ' X \cup (\lambda a. \text{id-cblinfun} \otimes_o a) ' Y)) \rangle$

lemma von-neumann-algebra-adj-image: $\langle \text{von-neumann-algebra } X \implies \text{adj} ' X = X \rangle$
by (auto simp: von-neumann-algebra-def intro!: image-eqI[**where** $x = \langle - * \rangle$])

lemma von-neumann-algebra-tensor-vn:
assumes $\langle \text{von-neumann-algebra } X \rangle$
assumes $\langle \text{von-neumann-algebra } Y \rangle$
shows $\langle \text{von-neumann-algebra} (X \otimes_{vN} Y) \rangle$
proof (rule von-neumann-algebraI)
have $\langle \text{adj} ' (X \otimes_{vN} Y) = \text{commutant} (\text{commutant} ((\lambda a. a \otimes_o \text{id-cblinfun}) ' \text{adj} ' X \cup (\lambda a. \text{id-cblinfun} \otimes_o a) ' \text{adj} ' Y)) \rangle$
by (simp add: tensor-vn-def commutant-adj image-image image-Un tensor-op-adjoint)
also have $\langle \dots = \text{commutant} (\text{commutant} ((\lambda a. a \otimes_o \text{id-cblinfun}) ' X \cup (\lambda a. \text{id-cblinfun} \otimes_o a) ' Y)) \rangle$
using assms **by** (simp add: von-neumann-algebra-adj-image)
also have $\langle \dots = X \otimes_{vN} Y \rangle$
by (simp add: tensor-vn-def)
finally show $\langle a * \in X \otimes_{vN} Y \rangle$ **if** $\langle a \in X \otimes_{vN} Y \rangle$ **for** a
using that **by** blast
show $\langle \text{commutant} (\text{commutant} (X \otimes_{vN} Y)) \subseteq X \otimes_{vN} Y \rangle$
by (simp add: tensor-vn-def)
qed

lemma tensor-vn-one-one[simp]: $\langle \text{one-algebra} \otimes_{vN} \text{one-algebra} = \text{one-algebra} \rangle$
apply (simp add: tensor-vn-def one-algebra-def image-image
tensor-op-scaleC-left tensor-op-scaleC-right)
by (simp add: commutant-one-algebra commutant-UNIV flip: one-algebra-def)

lemma sandwich-swap-tensor-vn: $\langle \text{sandwich swap-ell2} ' (X \otimes_{vN} Y) = Y \otimes_{vN} X \rangle$
by (simp add: tensor-vn-def sandwich-unitary-commutant image-Un image-image Un-commute)

lemma tensor-vn-one-left: $\langle \text{one-algebra} \otimes_{vN} X = (\lambda x. \text{id-cblinfun} \otimes_o x) ' X \rangle$ **if** $\langle \text{von-neumann-algebra}$

X
proof –
have $\langle \text{one-algebra } \otimes_{vN} X = \text{commutant} \text{ (commutant } ((\lambda a. \text{id-cblinfun } \otimes_o a) ' X)) \rangle$
apply (*simp add: tensor-vn-def one-algebra-def image-image*)
by (*metis (no-types, lifting) Un-commute Un-empty-right commutant-UNIV commutant-empty double-commutant-Un-right image-cong one-algebra-def tensor-id tensor-op-scaleC-left*)
also have $\langle \dots = (\lambda a. \text{id-cblinfun } \otimes_o a) ' \text{commutant (commutant } X) \rangle$
by (*simp add: amplification-double-commutant-commute'*)
also have $\langle \dots = (\lambda a. \text{id-cblinfun } \otimes_o a) ' X \rangle$
using *that von-neumann-algebra-def by blast*
finally show *?thesis*
by –
qed
lemma *tensor-vn-one-right*: $\langle X \otimes_{vN} \text{one-algebra} = (\lambda x. x \otimes_o \text{id-cblinfun}) ' X \rangle$ **if** $\langle \text{von-neumann-algebra } X \rangle$
proof –
have $\langle X \otimes_{vN} \text{one-algebra} = \text{sandwich swap-ell2} ' (\text{one-algebra } \otimes_{vN} X) \rangle$
by (*simp add: sandwich-swap-tensor-vn*)
also have $\langle \dots = \text{sandwich swap-ell2} ' (\lambda x. \text{id-cblinfun } \otimes_o x) ' X \rangle$
by (*simp add: tensor-vn-one-left that*)
also have $\langle \dots = (\lambda x. x \otimes_o \text{id-cblinfun}) ' X \rangle$
by (*simp add: image-image*)
finally show *?thesis*
by –
qed
lemma *double-commutant-in-vn-algI*: $\langle \text{commutant (commutant } X) \subseteq Y \rangle$
if $\langle \text{von-neumann-algebra } Y \rangle$ **and** $\langle X \subseteq Y \rangle$
by (*metis commutant-antimono that(1) that(2) von-neumann-algebra-def*)
lemma *von-neumann-algebra-compose*:
assumes $\langle \text{von-neumann-algebra } M \rangle$
assumes $\langle x \in M \rangle$ **and** $\langle y \in M \rangle$
shows $\langle x \circ_{CL} y \in M \rangle$
using *assms apply (auto simp: von-neumann-algebra-def commutant-def)*
by (*metis (no-types, lifting) assms(1) commutant-mult von-neumann-algebra-def*)
lemma *von-neumann-algebra-id*:
assumes $\langle \text{von-neumann-algebra } M \rangle$
shows $\langle \text{id-cblinfun} \in M \rangle$
using *assms by (auto simp: von-neumann-algebra-def)*
lemma *tensor-vn-UNIV[simp]*: $\langle UNIV \otimes_{vN} UNIV = (UNIV :: ((a \times b) \text{ ell2} \Rightarrow_{CL} -) \text{ set}) \rangle$
proof –
have $\langle (UNIV \otimes_{vN} UNIV :: ((a \times b) \text{ ell2} \Rightarrow_{CL} -) \text{ set}) = \text{commutant (range } (\lambda a. a \otimes_o \text{id-cblinfun}) \cup \text{range } (\lambda a. \text{id-cblinfun } \otimes_o a)) \rangle$
(is $\langle - = ?rhs \rangle$)
by (*simp add: tensor-vn-def commutant-cspan*)

```

also have  $\langle \dots \supseteq \text{commutant } (\text{commutant } \{a \otimes_o b \mid a \ b. \ \text{True}\}) \rangle$  (is  $\langle - \supseteq \dots \rangle$ )
proof (rule double-commutant-in-vn-algI)
  show  $\langle \text{von-neumann-algebra } ?rhs \rangle$ 
  by (metis calculation von-neumann-algebra-UNIV von-neumann-algebra-tensor-vn)
show  $\langle \{a \otimes_o b \mid (a :: 'a \ \text{ell2} \Rightarrow_{CL} -) \ (b :: 'b \ \text{ell2} \Rightarrow_{CL} -). \ \text{True}\} \subseteq ?rhs \rangle$ 
proof (rule subsetI)
  fix  $x :: \langle ('a \times 'b) \ \text{ell2} \Rightarrow_{CL} ('a \times 'b) \ \text{ell2} \rangle$ 
  assume  $\langle x \in \{a \otimes_o b \mid a \ b. \ \text{True}\} \rangle$ 
  then obtain  $a \ b$  where  $\langle x = a \otimes_o b \rangle$ 
  by auto
  then have  $\langle x = (a \otimes_o \text{id-cblinfun}) \ o_{CL} (\text{id-cblinfun} \otimes_o b) \rangle$ 
  by (simp add: comp-tensor-op)
  also have  $\langle \dots \in ?rhs \rangle$ 
proof -
  have  $\langle a \otimes_o \text{id-cblinfun} \in ?rhs \rangle$ 
  by (auto intro!: double-commutant-grows')
  moreover have  $\langle \text{id-cblinfun} \otimes_o b \in ?rhs \rangle$ 
  by (auto intro!: double-commutant-grows')
  ultimately show ?thesis
  using commutant-mult by blast
qed
finally show  $\langle x \in ?rhs \rangle$ 
by -
qed
qed
also have  $\langle \dots = \text{cstrong-operator-topology closure-of } (\text{cspan } \{a \otimes_o b \mid a \ b. \ \text{True}\}) \rangle$ 
apply (rule double-commutant-theorem-span)
apply (auto simp: comp-tensor-op tensor-op-adjoint)
using tensor-id[symmetric] by blast+
also have  $\langle \dots = UNIV \rangle$ 
using tensor-op-dense by blast
finally show ?thesis
by auto
qed
unbundle no cblinfun-syntax

```

end

17 Tensor-Product-Code – Support for code generation

```

theory Tensor-Product-Code
imports Hilbert-Space-Tensor-Product
  Complex-Bounded-Operators.Cblinfun-Code
begin

```

Automatic evaluation of formulas involving finite dimensional tensor products. Builds upon *Complex-Bounded-Operators.Cblinfun-Code* and reduces computations to the ex-

isting procedures from `Jordan_Normal_Form`.

unbundle `cblinfun-syntax` **and** `jnf-syntax`
hide-const (**open**) `Finite-Cartesian-Product.vec`
hide-const (**open**) `Finite-Cartesian-Product.mat`

definition `tensor-pack` :: $\text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \times \text{nat}) \Rightarrow \text{nat}$
where `tensor-pack` $X Y = (\lambda(x, y). x * Y + y)$

definition `tensor-unpack` :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \times \text{nat})$
where `tensor-unpack` $X Y xy = (xy \text{ div } Y, xy \text{ mod } Y)$

lemma `tensor-unpack-inj`:
assumes $i < A * B$ **and** $j < A * B$
shows `tensor-unpack` $A B i = \text{tensor-unpack } A B j \longleftrightarrow i = j$
by (`metis div-mult-mod-eq prod.sel(1) prod.sel(2) tensor-unpack-def`)

lemma `tensor-unpack-bound1[simp]`: $i < A * B \implies \text{fst } (\text{tensor-unpack } A B i) < A$
unfolding `tensor-unpack-def`

by (`auto intro!: less-mult-imp-div-less`)

lemma `tensor-unpack-bound2[simp]`: $i < A * B \implies \text{snd } (\text{tensor-unpack } A B i) < B$
unfolding `tensor-unpack-def`

by (`auto intro!: mod-less-divisor Nat.gr0I`)

lemma `tensor-unpack-fstfst`: $\langle \text{fst } (\text{tensor-unpack } A B (\text{fst } (\text{tensor-unpack } (A * B) C i))) \rangle$
 $= \text{fst } (\text{tensor-unpack } A (B * C) i \rangle$

unfolding `tensor-unpack-def` **by** (`auto simp flip: div-mult2-eq simp: mult.commute`)

lemma `tensor-unpack-sndsnd`: $\langle \text{snd } (\text{tensor-unpack } B C (\text{snd } (\text{tensor-unpack } A (B * C) i))) \rangle$
 $= \text{snd } (\text{tensor-unpack } (A * B) C i \rangle$

unfolding `tensor-unpack-def` **by** (`auto simp: mod-mod-cancel`)

lemma `tensor-unpack-fstsnd`: $\langle \text{fst } (\text{tensor-unpack } B C (\text{snd } (\text{tensor-unpack } A (B * C) i))) \rangle$
 $= \text{snd } (\text{tensor-unpack } A B (\text{fst } (\text{tensor-unpack } (A * B) C i))) \rangle$

unfolding `tensor-unpack-def`

by (`cases` $\langle C = 0 \rangle$) (`simp-all add: mult.commute [of B C] mod-mult2-eq [of i C B]`)

definition `tensor-state-jnf` $\psi \varphi = (\text{let } d1 = \text{dim-vec } \psi \text{ in let } d2 = \text{dim-vec } \varphi \text{ in}$
 $\text{vec } (d1 * d2) (\lambda i. \text{let } (i1, i2) = \text{tensor-unpack } d1 d2 i \text{ in } (\text{vec-index } \psi i1) * (\text{vec-index } \varphi i2)))$

lemma `tensor-state-jnf-dim[simp]`: $\langle \text{dim-vec } (\text{tensor-state-jnf } \psi \varphi) = \text{dim-vec } \psi * \text{dim-vec } \varphi \rangle$
unfolding `tensor-state-jnf-def` `Let-def` **by** `simp`

lemma `enum-prod-nth-tensor-unpack`:

assumes $\langle i < \text{CARD}'a \rangle * \text{CARD}'b \rangle$

shows $(\text{Enum.enum } ! i :: 'a :: \text{enum} \times 'b :: \text{enum}) =$

$(\text{let } (i1, i2) = \text{tensor-unpack } \text{CARD}'a \text{ CARD}'b i \text{ in}$
 $(\text{Enum.enum } ! i1, \text{Enum.enum } ! i2))$

using `assms`

by (`simp add: enum-prod-def product-nth tensor-unpack-def`)

lemma *vec-of-basis-enum-tensor-state-index*:
fixes $\psi :: \langle 'a::\text{enum ell2} \rangle$ **and** $\varphi :: \langle 'b::\text{enum ell2} \rangle$
assumes [*simp*]: $\langle i < \text{CARD}('a) * \text{CARD}('b) \rangle$
shows $\langle \text{vec-of-basis-enum } (\psi \otimes_s \varphi) \$ i = (\text{let } (i1,i2) = \text{tensor-unpack } \text{CARD}('a) \text{ CARD}('b)$
i in
 $\text{vec-of-basis-enum } \psi \$ i1 * \text{vec-of-basis-enum } \varphi \$ i2) \rangle$
proof –
define *i1 i2* **where** *i1* = *fst* (*tensor-unpack* *CARD*('a) *CARD*('b) *i*)
and *i2* = *snd* (*tensor-unpack* *CARD*('a) *CARD*('b) *i*)
have [*simp*]: *i1* < *CARD*('a) *i2* < *CARD*('b)
using *assms i1-def tensor-unpack-bound1* **apply** *presburger*
using *assms i2-def tensor-unpack-bound2* **by** *presburger*

have $\langle \text{vec-of-basis-enum } (\psi \otimes_s \varphi) \$ i = \text{Rep-ell2 } (\psi \otimes_s \varphi) (\text{enum-class.enum ! } i) \rangle$
by (*simp add: vec-of-basis-enum-ell2-component*)
also have $\langle \dots = \text{Rep-ell2 } \psi (\text{Enum.enum!}i1) * \text{Rep-ell2 } \varphi (\text{Enum.enum!}i2) \rangle$
apply (*transfer fixing: i i1 i2*)
by (*simp add: enum-prod-nth-tensor-unpack case-prod-beta i1-def i2-def*)
also have $\langle \dots = \text{vec-of-basis-enum } \psi \$ i1 * \text{vec-of-basis-enum } \varphi \$ i2 \rangle$
by (*simp add: vec-of-basis-enum-ell2-component*)
finally show *?thesis*
by (*simp add: case-prod-beta i1-def i2-def*)
qed

lemma *vec-of-basis-enum-tensor-state*:
fixes $\psi :: \langle 'a::\text{enum ell2} \rangle$ **and** $\varphi :: \langle 'b::\text{enum ell2} \rangle$
shows $\langle \text{vec-of-basis-enum } (\psi \otimes_s \varphi) = \text{tensor-state-jnf } (\text{vec-of-basis-enum } \psi) (\text{vec-of-basis-enum } \varphi) \rangle$
apply (*rule eq-vecI, simp-all*)
apply (*subst vec-of-basis-enum-tensor-state-index, simp-all*)
by (*simp add: tensor-state-jnf-def case-prod-beta Let-def*)

lemma *mat-of-cblinfun-tensor-op-index*:
fixes $a :: \langle 'a::\text{enum ell2} \Rightarrow_{CL} 'b::\text{enum ell2} \rangle$ **and** $b :: \langle 'c::\text{enum ell2} \Rightarrow_{CL} 'd::\text{enum ell2} \rangle$
assumes [*simp*]: $\langle i < \text{CARD}('b) * \text{CARD}('d) \rangle$
assumes [*simp*]: $\langle j < \text{CARD}('a) * \text{CARD}('c) \rangle$
shows $\langle \text{mat-of-cblinfun } (\text{tensor-op } a \ b) \$\$ (i,j) =$
 $(\text{let } (i1,i2) = \text{tensor-unpack } \text{CARD}('b) \text{ CARD}('d) \ i \ \text{in}$
 $\text{let } (j1,j2) = \text{tensor-unpack } \text{CARD}('a) \ \text{CARD}('c) \ j \ \text{in}$
 $\text{mat-of-cblinfun } a \ \$\$ (i1,j1) * \text{mat-of-cblinfun } b \ \$\$ (i2,j2)) \rangle$
proof –
define *i1 i2 j1 j2*
where *i1* = *fst* (*tensor-unpack* *CARD*('b) *CARD*('d) *i*)
and *i2* = *snd* (*tensor-unpack* *CARD*('b) *CARD*('d) *i*)
and *j1* = *fst* (*tensor-unpack* *CARD*('a) *CARD*('c) *j*)

```

    and j2 = snd (tensor-unpack CARD('a) CARD('c) j)
  have [simp]: i1 < CARD('b) i2 < CARD('d) j1 < CARD('a) j2 < CARD('c)
    using assms i1-def tensor-unpack-bound1 apply presburger
    using assms i2-def tensor-unpack-bound2 apply blast
    using assms(2) j1-def tensor-unpack-bound1 apply blast
    using assms(2) j2-def tensor-unpack-bound2 by presburger

  have ⟨mat-of-cblinfun (tensor-op a b) $$ (i,j)
    = Rep-ell2 (tensor-op a b *V ket (Enum.enum!j)) (Enum.enum ! i)⟩
    by (simp add: mat-of-cblinfun-ell2-component)
  also have ⟨... = Rep-ell2 ((a *V ket (Enum.enum!j1)) ⊗s (b *V ket (Enum.enum!j2)))
    (Enum.enum!i)⟩
    by (simp add: tensor-op-ell2 enum-prod-nth-tensor-unpack[where i=j] Let-def case-prod-beta
    j1-def[symmetric] j2-def[symmetric] flip: tensor-ell2-ket)
  also have ⟨... = vec-of-basis-enum ((a *V ket (Enum.enum!j1)) ⊗s b *V ket (Enum.enum!j2))
    $ i⟩
    by (simp add: vec-of-basis-enum-ell2-component)
  also have ⟨... = vec-of-basis-enum (a *V ket (enum-class.enum ! j1)) $ i1 *
    vec-of-basis-enum (b *V ket (enum-class.enum ! j2)) $ i2⟩
    by (simp add: case-prod-beta vec-of-basis-enum-tensor-state-index i1-def[symmetric] i2-def[symmetric])
  also have ⟨... = Rep-ell2 (a *V ket (enum-class.enum ! j1)) (enum-class.enum ! i1) *
    Rep-ell2 (b *V ket (enum-class.enum ! j2)) (enum-class.enum ! i2)⟩
    by (simp add: vec-of-basis-enum-ell2-component)
  also have ⟨... = mat-of-cblinfun a $$ (i1, j1) * mat-of-cblinfun b $$ (i2, j2)⟩
    by (simp add: mat-of-cblinfun-ell2-component)
  finally show ?thesis
    by (simp add: i1-def[symmetric] i2-def[symmetric] j1-def[symmetric] j2-def[symmetric]
    case-prod-beta)
qed

```

definition *tensor-op-jnf* $A B =$

```

  (let r1 = dim-row A in
  let c1 = dim-col A in
  let r2 = dim-row B in
  let c2 = dim-col B in
  mat (r1 * r2) (c1 * c2)
  (λ(i,j). let (i1,i2) = tensor-unpack r1 r2 i in
    let (j1,j2) = tensor-unpack c1 c2 j in
    (A $$ (i1,j1)) * (B $$ (i2,j2))))

```

lemma *tensor-op-jnf-dim*[simp]:

```

⟨dim-row (tensor-op-jnf a b) = dim-row a * dim-row b⟩
⟨dim-col (tensor-op-jnf a b) = dim-col a * dim-col b⟩
unfolding tensor-op-jnf-def Let-def by simp-all

```

lemma *mat-of-cblinfun-tensor-op*:

```

fixes a :: ⟨'a::enum ell2 ⇒CL 'b::enum ell2⟩ and b :: ⟨'c::enum ell2 ⇒CL 'd::enum ell2⟩

```

shows $\langle \text{mat-of-cblinfun } (\text{tensor-op } a \ b) = \text{tensor-op-jnf } (\text{mat-of-cblinfun } a) (\text{mat-of-cblinfun } b) \rangle$

apply (*rule eq-matI, simp-all add: canonical-basis-length*)

apply (*subst mat-of-cblinfun-tensor-op-index, simp-all*)

by (*simp add: tensor-op-jnf-def case-prod-beta Let-def canonical-basis-length*)

lemma *mat-of-cblinfun-assoc-ell2* [*simp*]:

$\langle \text{mat-of-cblinfun } (\text{assoc-ell2} * :: ((a::\text{enum} \times (b::\text{enum} \times (c::\text{enum}))) \ \text{ell2} \Rightarrow_{CL} -)) = \text{one-mat } (\text{CARD}(a) * \text{CARD}(b) * \text{CARD}(c)) \rangle$

(*is mat-of-cblinfun ?assoc = -*)

proof (*rule mat-eq-iff[THEN iffD2], intro conjI allI impI*)

show $\langle \text{dim-row } (\text{mat-of-cblinfun } ?\text{assoc}) = \text{dim-row } (1_m (\text{CARD}(a) * \text{CARD}(b) * \text{CARD}(c))) \rangle$

by (*simp add: canonical-basis-length*)

show $\langle \text{dim-col } (\text{mat-of-cblinfun } ?\text{assoc}) = \text{dim-col } (1_m (\text{CARD}(a) * \text{CARD}(b) * \text{CARD}(c))) \rangle$

by (*simp add: canonical-basis-length*)

fix *i j*

let *?i = Enum.enum ! i :: ('a × 'b) × 'c* **and** *?j = Enum.enum ! j :: 'a × ('b × 'c)*

assume $\langle i < \text{dim-row } (1_m (\text{CARD}(a) * \text{CARD}(b) * \text{CARD}(c))) \rangle$

then have *iB* [*simp*]: $\langle i < \text{CARD}(a) * \text{CARD}(b) * \text{CARD}(c) \rangle$ **by** *simp*

then have *iB'* [*simp*]: $\langle i < \text{CARD}(a) * (\text{CARD}(b) * \text{CARD}(c)) \rangle$ **by** *linarith*

assume $\langle j < \text{dim-col } (1_m (\text{CARD}(a) * \text{CARD}(b) * \text{CARD}(c))) \rangle$

then have *jB* [*simp*]: $\langle j < \text{CARD}(a) * \text{CARD}(b) * \text{CARD}(c) \rangle$ **by** *simp*

then have *jB'* [*simp*]: $\langle j < \text{CARD}(a) * (\text{CARD}(b) * \text{CARD}(c)) \rangle$ **by** *linarith*

define *i1 i23 i2 i3*

where *i1* = *fst* (*tensor-unpack* *CARD(a)* (*CARD(b)*CARD(c)*) *i*)

and *i23* = *snd* (*tensor-unpack* *CARD(a)* (*CARD(b)*CARD(c)*) *i*)

and *i2* = *fst* (*tensor-unpack* *CARD(b)* *CARD(c)* *i23*)

and *i3* = *snd* (*tensor-unpack* *CARD(b)* *CARD(c)* *i23*)

define *j12 j1 j2 j3*

where *j12* = *fst* (*tensor-unpack* (*CARD(a)*CARD(b)*) *CARD(c)* *j*)

and *j1* = *fst* (*tensor-unpack* *CARD(a)* *CARD(b)* *j12*)

and *j2* = *snd* (*tensor-unpack* *CARD(a)* *CARD(b)* *j12*)

and *j3* = *snd* (*tensor-unpack* (*CARD(a)*CARD(b)*) *CARD(c)* *j*)

have [*simp*]: *j12* < *CARD(a)*CARD(b)* *i23* < *CARD(b)*CARD(c)*

using *j12-def jB tensor-unpack-bound1* **apply** *presburger*

using *i23-def iB' tensor-unpack-bound2* **by** *blast*

have *j1'*: $\langle \text{fst } (\text{tensor-unpack } \text{CARD}(a) (\text{CARD}(b) * \text{CARD}(c)) \ j) = j1 \rangle$

by (*simp add: j1-def j12-def tensor-unpack-fstfst*)

let *?i1 = Enum.enum ! i1 :: 'a* **and** *?i2 = Enum.enum ! i2 :: 'b* **and** *?i3 = Enum.enum ! i3*

```

:: 'c
let ?j1 = Enum.enum ! j1 :: 'a and ?j2 = Enum.enum ! j2 :: 'b and ?j3 = Enum.enum ! j3
:: 'c

have i: ⟨?i = ((?i1, ?i2), ?i3)⟩
  by (auto simp add: enum-prod-nth-tensor-unpack case-prod-beta
      tensor-unpack-fstfst tensor-unpack-fstsnd tensor-unpack-sndsnd i1-def i2-def i23-def
i3-def)
have j: ⟨?j = (?j1, (?j2, ?j3))⟩
  by (auto simp add: enum-prod-nth-tensor-unpack case-prod-beta
      tensor-unpack-fstfst tensor-unpack-fstsnd tensor-unpack-sndsnd j1-def j2-def j12-def j3-def)
have ijeq: ⟨(?i1, ?i2, ?i3) = (?j1, ?j2, ?j3) ⟷ i = j⟩
  unfolding i1-def i2-def i3-def j1-def j2-def j3-def apply simp
  apply (subst enum-inj, simp, simp)
  apply (subst enum-inj, simp, simp)
  apply (subst enum-inj, simp, simp)
  apply (subst tensor-unpack-inj[symmetric, where i=i and j=j and A=CARD('a) and
B=CARD('b)*CARD('c)], simp, simp)
  unfolding prod-eq-iff
  apply (subst tensor-unpack-inj[symmetric, where i=⟨snd (tensor-unpack CARD('a) (CARD('b)
* CARD('c)) i)⟩ and A=CARD('b) and B=CARD('c)], simp, simp)
  by (simp add: i1-def[symmetric] j1-def[symmetric] i2-def[symmetric] j2-def[symmetric]
i3-def[symmetric] j3-def[symmetric]
i23-def[symmetric] j12-def[symmetric] j1'
prod-eq-iff tensor-unpack-fstsnd tensor-unpack-sndsnd)

have ⟨mat-of-cblinfun ?assoc $$ (i, j) = Rep-ell2 (assoc-ell2* *V ket ?j) ?i⟩
  by (subst mat-of-cblinfun-ell2-component, auto)
also have ⟨... = Rep-ell2 ((ket ?j1 ⊗s ket ?j2) ⊗s ket ?j3) ?i⟩
  by (simp add: j assoc-ell2'-tensor-flip: tensor-ell2-ket)
also have ⟨... = (if (?i1, ?i2, ?i3) = (?j1, ?j2, ?j3) then 1 else 0)⟩
  by (auto simp add: ket.rep-eq i tensor-ell2-ket)
also have ⟨... = (if i=j then 1 else 0)⟩
  using ijeq by simp
finally
show ⟨mat-of-cblinfun ?assoc $$ (i, j) =
      1m (CARD('a) * CARD('b) * CARD('c)) $$ (i, j)⟩
  by auto
qed

lemma mat-of-cblinfun-assoc-ell2[simp]:
  ⟨mat-of-cblinfun (assoc-ell2 :: (('a::enum×'b::enum)×'c::enum) ell2 ⇒CL -) = one-mat
(CARD('a)*CARD('b)*CARD('c))⟩
  (is mat-of-cblinfun ?assoc = -)
proof -
let ?assoc' = assoc-ell2* :: (('a::enum×'b::enum×'c::enum)) ell2 ⇒CL -)
have one-mat (CARD('a)*CARD('b)*CARD('c)) = mat-of-cblinfun (?assoc oCL ?assoc')
  by (simp add: mult.assoc mat-of-cblinfun-id)

```



```

also have ⟨... = mat-of-cblinfun ?assoc * mat-of-cblinfun ?assoc⟩
  using mat-of-cblinfun-compose by blast
also have ⟨... = mat-of-cblinfun ?assoc * one-mat (CARD('a)*CARD('b)*CARD('c))⟩
  by simp
also have ⟨... = mat-of-cblinfun ?assoc⟩
  apply (rule right-mult-one-mat')
  by (simp add: canonical-basis-length)
finally show ?thesis
  by simp
qed

```

unbundle no cblinfun-syntax **and** no jnf-syntax

end

References

- [1] J. B. Conway. *A course in operator theory*. Number 21 in Graduate studies in mathematics. American Mathematical Society, Providence, RI, 2000.
- [2] J. B. Conway. *A course in functional analysis*, volume 96. Springer Science & Business Media, 2013.
- [3] Faisal and Y. Choi. $\text{tr}(ab) = \text{tr}(ba)$? Mathoverflow answer, <https://mathoverflow.net/a/76389/101775>, 2011–2014.
- [4] A. Henriques and A. Taghavi. $\text{tr}(ab) = \text{tr}(ba)$? Mathoverflow question, <https://mathoverflow.net/questions/76386/trab-trba>, 2011–2014.
- [5] E. W. ([https://math.stackexchange.com/users/86856/eric wofsey](https://math.stackexchange.com/users/86856/eric-wofsey)). A bounded increasing net converges formulated using filters. Mathematics Stack Exchange (Answer). URL:<https://math.stackexchange.com/q/4749216> (version: 2023-08-07).
- [6] O. Kunar and A. Popescu. From types to sets by local type definition in higher-order logic. *Journal of Automated Reasoning*, 62(2):237260, June 2018.
- [7] M. Takesaki. *Theory of operator algebras I*. Springer, New York, NY, Nov. 2011.
- [8] D. Unruh. Quantum references. [arXiv:2105.10914v3 \[cs.LO\]](https://arxiv.org/abs/2105.10914v3), 2024.
- [9] F. Wecken. Zur theorie linearer operatoren. *Math. Ann.*, 110:722–725, 1935.