

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>	22
3	<i>Strong-Operator-Topology</i> – Strong operator topology on complex bounded operators	28
4	<i>Positive-Operators</i> – Positive bounded operators	33
5	<i>HS2Ell2</i> – Representing any Hilbert space as $\ell_2(X)$	40
6	<i>Weak-Operator-Topology</i> – Weak operator topology on complex bounded operators	42
7	<i>Misc-Tensor-Product-TTS</i> – Miscellaneous results missing from <code>Complex_Bounded_Operators</code>	50
7.1	Retrieving axioms	51
7.2	Auxiliary lemmas	51
7.3	<i>plus</i>	53
7.3.1	<i>minus</i>	53
7.3.2	<i>uminus</i>	53

7.4	<i>semigroup</i>	53
7.5	<i>abel-semigroup</i>	54
7.6	<i>comm-monoid</i>	54
7.7	<i>topological-space</i>	55
7.8	<i>sum</i>	55
7.9	<i>t2-space</i>	56
7.9.1	<i>continuous-on</i>	56
7.10	<i>scaleR</i>	56
7.11	<i>scaleC</i>	57
7.12	<i>ab-group-add</i>	57
7.13	<i>vector-space</i>	58
7.14	<i>complex-vector</i>	58
7.15	<i>open-uniformity</i>	59
7.16	<i>uniformity-dist</i>	59
7.17	<i>sgn</i>	59
7.18	<i>sgn-div-norm</i>	60
7.19	<i>dist-norm</i>	60
7.20	<i>complex-inner</i>	61
7.21	<i>is-ortho-set</i>	61
7.22	<i>metric-space</i>	62
7.23	<i>nhds</i>	62
7.24	<i>at-within</i>	63
7.25	<i>(has-sum)</i>	63
7.26	<i>filterlim</i>	64
7.27	<i>convergent</i>	64
7.28	<i>uniform-space.cauchy-filter</i>	64
7.29	<i>uniform-space.Cauchy</i>	64
7.30	<i>complete-space</i>	65
7.31	<i>chilbert-space</i>	65
7.32	<i>(hull)</i>	66
7.33	<i>csubspace</i>	66
7.34	<i>cspan</i>	67
7.34.1	<i>(islimpt)</i>	67
7.34.2	<i>closure</i>	68
7.35	<i>continuous</i>	68
7.36	<i>is-onb</i>	68
7.37	Transferring theorems	69
8	Stuff relying on the above lifting	70
9	<i>Eigenvalues</i> – Material related to eigenvalues and eigenspaces	71

10	<i>Compact-Operators</i> – Finite rank and compact operators	75
10.1	Finite rank operators	76
10.2	Compact operators	77
11	<i>Spectral-Theorem</i> – The spectral theorem for compact operators	82
11.1	Spectral decomp, compact op	83
12	<i>Trace-Class</i> – Trace-class operators	87
12.1	Auxiliary lemmas	87
12.2	Trace-norm and trace-class	88
12.3	Hilbert-Schmidt operators	90
12.4	Trace-norm and trace-class, continued	94
12.5	More Hilbert-Schmidt	110
12.6	Spectral Theorem	111
12.7	More Trace-Class	112
13	<i>Weak-Star-Topology</i> – Weak* topology on complex bounded operators	113
14	<i>Hilbert-Space-Tensor-Product</i> – Tensor product of Hilbert Spaces	119
14.1	Tensor product on - <i>ell2</i>	119
14.2	Tensor product of operators on - <i>ell2</i>	123
14.3	Tensor product of subspaces	131
15	<i>Partial-Trace</i> – The partial trace	133
16	<i>Von-Neumann-Algebras</i> – Von Neumann algebras and the double commutant theorem	135
16.1	Commutants	135
16.2	Double commutant theorem	137
16.3	Von Neumann Algebras	141
17	<i>Tensor-Product-Code</i> – Support for code generation	143
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*: $(\bigwedge x. x=y \implies P) \implies P$ *<proof>*

lemma *inv-prod-swap[simp]*: $\langle \text{inv prod.swap} = \text{prod.swap} \rangle$
<proof>

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

includes *lifting-syntax*

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

shows $\langle ((R \implies S) \implies \text{rel-filter } S \implies \text{rel-filter } R \implies (=)) \text{ filterlim filterlim} \rangle$
<proof>

definition *rel-topology* :: $\langle ('a \Rightarrow 'b \Rightarrow \text{bool}) \Rightarrow ('a \text{ topology} \Rightarrow 'b \text{ topology} \Rightarrow \text{bool}) \rangle$ **where**

$\langle \text{rel-topology } R \ S \ T \longleftrightarrow (\text{rel-fun } (\text{rel-set } R) (=)) (\text{openin } S) (\text{openin } T)$

$\wedge (\forall U. \text{openin } S \ U \longrightarrow \text{Domainp } (\text{rel-set } R) \ U) \wedge (\forall U. \text{openin } T \ U \longrightarrow \text{Rangep } (\text{rel-set } R) \ U) \rangle$

lemma *rel-topology-eq[relator-eq]*: $\langle \text{rel-topology } (=) = (=) \rangle$
<proof>

lemma *Rangep-conversep[simp]*: $\langle \text{Rangep } (R^{-1-1}) = \text{Domainp } R \rangle$
<proof>

lemma *Domainp-conversep[simp]*: $\langle \text{Domainp } (R^{-1-1}) = \text{Rangep } R \rangle$
<proof>

lemma *conversep-rel-fun*:

includes *lifting-syntax*

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

<proof>

lemma *rel-topology-conversep[simp]*: $\langle \text{rel-topology } (R^{-1-1}) = ((\text{rel-topology } R)^{-1-1}) \rangle$
<proof>

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

includes *lifting-syntax*

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

<proof>

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

includes *lifting-syntax*

shows $\langle (\text{rel-topology } R \implies \text{rel-set } R) \text{ topspace topspace} \rangle$

<proof>

lemma [*transfer-rule*]:

includes *lifting-syntax*

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

assumes $[transfer-rule]: \langle bi\text{-}unique\ S \rangle$
assumes $[transfer-rule]: \langle bi\text{-}total\ R \rangle$
assumes $[transfer-rule]: \langle bi\text{-}unique\ R \rangle$
shows $\langle (rel\text{-}topology\ R\ ==>\ rel\text{-}topology\ S\ ==>\ (R\ ==>\ S)\ ==>\ (=))\ continuous\text{-}map\ continuous\text{-}map \rangle$
 $\langle proof \rangle$

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

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

lemma *limit-in-closure*:
assumes $lim: \langle f \longrightarrow x \rangle\ F$
assumes $nt: \langle F \neq bot \rangle$
assumes $inA: \langle \forall_F\ x\ in\ F.\ f\ x \in A \rangle$
shows $\langle x \in closure\ A \rangle$
 $\langle proof \rangle$

lemma *filterlim-nhdsin-iff-limitin*:
 $\langle l \in\ topspace\ T \wedge filterlim\ f\ (nhdsin\ T\ l)\ F \iff limitin\ T\ f\ l\ F \rangle$
 $\langle proof \rangle$

lemma *pullback-topology-bi-cont*:
fixes $g :: \langle 'a \Rightarrow ('b \Rightarrow 'c::topological\text{-}space) \rangle$
and $f :: \langle 'a \Rightarrow 'a \Rightarrow 'a \rangle$ **and** $f' :: \langle 'c \Rightarrow 'c \Rightarrow 'c \rangle$
assumes $gf\text{-}f'g: \langle \bigwedge a\ b\ i.\ g\ (f\ a\ b)\ i = f'\ (g\ a\ i)\ (g\ b\ i) \rangle$
assumes $f'\text{-}cont: \langle \bigwedge a'\ b'.\ (case\text{-}prod\ f' \longrightarrow f'\ a'\ b')\ (nhds\ a' \times_F\ nhds\ b') \rangle$
defines $\langle T \equiv pullback\text{-}topology\ UNIV\ g\ euclidean \rangle$
shows $\langle LIM\ (x,y)\ nhdsin\ T\ a \times_F\ nhdsin\ T\ b.\ f\ x\ y :>\ nhdsin\ T\ (f\ a\ b) \rangle$
 $\langle proof \rangle$

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

lemma *has-sum-in-finite*:
assumes $finite\ F$
assumes $\langle sum\ f\ F \in\ topspace\ T \rangle$
shows $has\text{-}sum\text{-}in\ T\ f\ F\ (sum\ f\ F)$

⟨proof⟩

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

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

lemma hausdorff-OFCLASS-t2-space: ⟨OFCLASS('a::topological-space, t2-space-class)⟩ **if** ⟨Hausdorff-space (euclidean :: 'a topology)⟩
⟨proof⟩

lemma hausdorffI:

assumes ⟨ $\bigwedge x y. x \in \text{topspace } T \implies y \in \text{topspace } T \implies x \neq y \implies \exists U V. \text{openin } T U \wedge \text{openin } T V \wedge x \in U \wedge y \in V \wedge U \cap V = \{\}$ ⟩

shows ⟨Hausdorff-space T ⟩

⟨proof⟩

lemma hausdorff-euclidean[simp]: ⟨Hausdorff-space (euclidean :: -:t2-space topology)⟩
⟨proof⟩

lemma has-sum-in-unique:

assumes ⟨Hausdorff-space T ⟩

assumes ⟨has-sum-in $T f A l$ ⟩

assumes ⟨has-sum-in $T f A l'$ ⟩

shows ⟨ $l = l'$ ⟩

⟨proof⟩

lemma infsum-in-def':

assumes ⟨Hausdorff-space T ⟩

shows ⟨infsum-in $T f A = (\text{if summable-on-in } T f A \text{ then } (\text{THE } s. \text{has-sum-in } T f A s) \text{ else } 0)$ ⟩

⟨proof⟩

lemma has-sum-in-infsum-in:

assumes ⟨Hausdorff-space T ⟩ **and** summable: ⟨summable-on-in $T f A$ ⟩

shows ⟨has-sum-in $T f A$ (infsum-in $T f A$)⟩

⟨proof⟩

lemma nhdsin-mono:

assumes [simp]: ⟨ $\bigwedge x. \text{openin } T' x \implies \text{openin } T x$ ⟩

assumes [simp]: ⟨topspace $T = \text{topspace } T'$ ⟩

shows ⟨nhdsin $T a \leq \text{nhdsin } T' a$ ⟩

⟨proof⟩

lemma has-sum-in-cong:

assumes $\bigwedge x. x \in A \implies f x = g x$
shows $\text{has-sum-in } T f A x \longleftrightarrow \text{has-sum-in } T g A x$
 $\langle \text{proof} \rangle$

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 $\langle \text{infsum-in } T f A = \text{infsum-in } T g B \rangle$
 $\langle \text{proof} \rangle$

lemma *infsum-in-cong*:
assumes $\bigwedge x. x \in A \implies f x = g x$
shows $\text{infsum-in } T f A = \text{infsum-in } T g A$
 $\langle \text{proof} \rangle$

lemma *limitin-cong*: $\text{limitin } T f c F \longleftrightarrow \text{limitin } T g c F$ **if eventually** $(\lambda x. f x = g x) F$
 $\langle \text{proof} \rangle$

lemma *has-sum-in-reindex*:
assumes $\langle \text{inj-on } h A \rangle$
shows $\langle \text{has-sum-in } T g (h \text{ ` } A) x \longleftrightarrow \text{has-sum-in } T (g \circ h) A x \rangle$
 $\langle \text{proof} \rangle$

lemma *summable-on-in-reindex*:
assumes $\langle \text{inj-on } h A \rangle$
shows $\langle \text{summable-on-in } T g (h \text{ ` } A) \longleftrightarrow \text{summable-on-in } T (g \circ h) A \rangle$
 $\langle \text{proof} \rangle$

lemma *infsum-in-reindex*:
assumes $\langle \text{inj-on } h A \rangle$
shows $\langle \text{infsum-in } T g (h \text{ ` } A) = \text{infsum-in } T (g \circ h) A \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-in-reindex-bij-betw*:
assumes $\text{bij-betw } g A B$
shows $\text{has-sum-in } T (\lambda x. f (g x)) A s \longleftrightarrow \text{has-sum-in } T f B s$
 $\langle \text{proof} \rangle$

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

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

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

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

assumes *bij-betw* g A B
shows *infsun-in* T $(\lambda x. f (g x))$ $A = \text{infsun-in } T f B$
<proof>

lemma *limitin-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } S \rangle$
shows $\langle (\text{rel-topology } S \implies (R \implies S) \implies S \implies \text{rel-filter } R \implies (\longleftrightarrow))$
limitin limitin
<proof>

lemma *finite-subsets-at-top-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } R \rangle$
shows $\langle (\text{rel-set } R \implies \text{rel-filter } (\text{rel-set } R)) \text{ finite-subsets-at-top finite-subsets-at-top}$
<proof>

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*]: $\langle (S \implies S \implies S) (+) (+) \rangle$
assumes [*transfer-rule*]: $\langle S \ 0 \ 0 \rangle$
shows $\langle (R \implies S) \implies \text{rel-set } R \implies S \text{ sum sum} \rangle$
<proof>

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*]: $\langle (S \implies S \implies S) (+) (+) \rangle$
assumes [*transfer-rule*]: $\langle S \ 0 \ 0 \rangle$
shows $\langle (\text{rel-topology } S \implies (R \implies S) \implies (\text{rel-set } R) \implies S \implies (=))$
has-sum-in has-sum-in
<proof>

lemma *has-sum-in-topspace*: $\langle \text{has-sum-in } T f A s \implies s \in \text{topspace } T \rangle$
<proof>

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*]: $\langle (S \implies S \implies S) (+) (+) \rangle$
assumes [*transfer-rule*]: $\langle S \ 0 \ 0 \rangle$
shows $\langle (\text{rel-topology } S \implies (R \implies S) \implies (\text{rel-set } R) \implies (=)) \text{ summable-on-in}$

summable-on-in
 ⟨proof⟩

lemma *not-summable-infsum-in-0*: $\langle \neg \text{summable-on-in } T f A \implies \text{infsum-in } T f A = 0 \rangle$
 ⟨proof⟩

lemma *infsum-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*]: $\langle (S \implies S \implies S) (+) (+) \rangle$
assumes [*transfer-rule*]: $\langle S \ 0 \ 0 \rangle$
shows $\langle (\text{rel-topology } S \implies (R \implies S) \implies (\text{rel-set } R) \implies S) \text{ infsum-in infsum-in} \rangle$
 ⟨proof⟩

lemma *infsum-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } R \rangle$
shows $\langle ((R \implies (=)) \implies (\text{rel-set } R) \implies (=)) \text{ infsum infsum} \rangle$
 ⟨proof⟩

lemma *summable-on-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } R \rangle$
shows $\langle ((R \implies (=)) \implies (\text{rel-set } R) \implies (=)) \text{ Infinite-Sum.summable-on Infinite-Sum.summable-on} \rangle$
 ⟨proof⟩

lemma *abs-gbinomial*: $\langle \text{abs } (a \text{ gchoose } n) = (-1)^{\wedge(n - \text{nat } (\text{ceiling } a))} * (a \text{ gchoose } n) \rangle$
 ⟨proof⟩

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 $(\sum k \leq m. \text{abs } (a \text{ gchoose } k)) =$
 $(-1)^{\wedge a'} * ((-1)^{\wedge m} * (a - 1 \text{ gchoose } m))$
 $- (-1)^{\wedge a'} * \text{of-bool } (a' > 0) * ((-1)^{\wedge (a'-1)} * (a-1 \text{ gchoose } (a'-1)))$
 $+ (\sum k < a'. \text{abs } (a \text{ gchoose } k))$
 ⟨proof⟩

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⟩

lemma *gbinomial-summable-abs*:

fixes $a :: \text{real}$
assumes $\langle a \geq 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle \text{summable } (\lambda n. \text{abs } (a \text{ gchoose } n)) \rangle$
 $\langle \text{proof} \rangle$

lemma *summable-tendsto-times-n*:
fixes $f :: \langle \text{nat} \Rightarrow \text{real} \rangle$
assumes $\text{pos}: \langle \bigwedge n. f\ n \geq 0 \rangle$
assumes $\text{dec}: \langle \text{decseq } (\lambda n. (n+M) * f\ (n + M)) \rangle$
assumes $\text{sum}: \langle \text{summable } f \rangle$
shows $\langle (\lambda n. n * f\ n) \longrightarrow 0 \rangle$
 $\langle \text{proof} \rangle$

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

lemma *gbinomial-abs-sum*:
fixes $a :: \text{real}$
assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle (\lambda n. \text{abs } (a \text{ gchoose } n)) \text{ sums } 2 \rangle$
 $\langle \text{proof} \rangle$

lemma *sums-has-sum*:
fixes $s :: \langle 'a :: \text{banach} \rangle$
assumes $\text{sums}: \langle f \text{ sums } s \rangle$
assumes $\text{abs-sum}: \langle \text{summable } (\lambda n. \text{norm } (f\ n)) \rangle$
shows $\langle (f \text{ has-sum } s) \text{ UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *sums-has-sum-pos*:
fixes $s :: \text{real}$
assumes $\langle f \text{ sums } s \rangle$
assumes $\langle \bigwedge n. f\ n \geq 0 \rangle$
shows $\langle (f \text{ has-sum } s) \text{ UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *gbinomial-abs-has-sum*:
fixes $a :: \text{real}$
assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle ((\lambda n. \text{abs } (a \text{ gchoose } n)) \text{ has-sum } 2) \text{ UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *gbinomial-abs-has-sum-1*:
fixes $a :: \text{real}$
assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle (\lambda n. \text{abs } (a \text{ gchoose } n)) \text{ has-sum } 1 \rangle$ $(\text{UNIV} - \{0\})$
 $\langle \text{proof} \rangle$

lemma *gbinomial-abs-summable*:
fixes $a :: \text{real}$
assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle (\lambda n. (a \text{ gchoose } n)) \text{ abs-summable-on } \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *gbinomial-abs-summable-1*:
fixes $a :: \text{real}$
assumes $\langle a > 0 \rangle$ **and** $\langle a \leq 1 \rangle$
shows $\langle (\lambda n. (a \text{ gchoose } n)) \text{ abs-summable-on } \text{UNIV} - \{0\} \rangle$
 $\langle \text{proof} \rangle$

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

lemma *has-sum-sums*: $\langle f \text{ sums } s \rangle$ **if** $\langle (f \text{ has-sum } s) \text{ UNIV} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle P (\text{The } F) = Q (\text{The } F) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (\lambda i. - f i) \text{ summable-on } A \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (\lambda x. f x - g x) \text{ summable-on } A \rangle$
 $\langle \text{proof} \rangle$

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

<proof>

lemma *gbinomial-a-Suc-n*:

*<(a gchoose Suc n) = (a gchoose n) * (a-n) / Suc n>*
<proof>

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

assumes *<0 ∈ topspace T>*
assumes *<∧x. x ∈ A ⇒ f x = 0>*
shows *<has-sum-in T f A 0>*

<proof>

lemma *has-sum-diff*:

fixes *f g :: 'a ⇒ 'b::{topological-ab-group-add}*
assumes *<(f has-sum a) A>*
assumes *<(g has-sum b) A>*
shows *<(λx. f x - g x) has-sum (a - b) A>*
<proof>

lemma *has-sum-of-real*:

fixes *f :: 'a ⇒ real*
assumes *<(f has-sum a) A>*
shows *<(λx. of-real (f x)) has-sum (of-real a :: 'b::{real-algebra-1,real-normed-vector}) A>*
<proof>

lemma *summable-on-cdivide*:

fixes *f :: 'a ⇒ 'b :: {t2-space, topological-semigroup-mult, division-ring}*
assumes *<f summable-on A>*
shows *(λx. f x / c) summable-on A*
<proof>

lemma *norm-abs[simp]*: *<norm (abs x) = norm x>* **for** *x :: 'a :: {idom-abs-sgn, real-normed-div-algebra}*

<proof>

thm *abs-summable-product*

lemma *abs-summable-product*:

fixes *x :: 'a ⇒ 'b::real-normed-div-algebra*
assumes *x2-sum: (λi. (x i)²) abs-summable-on A*
and *y2-sum: (λi. (y i)²) abs-summable-on A*
shows *(λi. x i * y i) abs-summable-on A*

<proof>

lemma *Cauchy-Schwarz-ineq-infsum*:

fixes *x :: 'a ⇒ 'b::{real-normed-div-algebra}*
assumes *x2-sum: (λi. (x i)²) abs-summable-on A*
and *y2-sum: (λi. (y i)²) abs-summable-on A*
shows *<(∑_{∞ i ∈ A.} norm (x i * y i)) ≤ sqrt (∑_{∞ i ∈ A.} (norm (x i))²) * sqrt (∑_{∞ i ∈ A.} (norm*

$\langle (y \ i)^2 \rangle$
 $\langle proof \rangle$

lemma *continuous-map-pullback-both*:

assumes *cont*: $\langle continuous-map \ T1 \ T2 \ g' \rangle$

assumes *g'g*: $\langle \bigwedge x. f1 \ x \in \ topspace \ T1 \implies x \in A1 \implies g' \ (f1 \ x) = f2 \ (g \ x) \rangle$

assumes *top1*: $\langle f1 \ -' \ topspace \ T1 \cap A1 \subseteq g \ -' \ A2 \rangle$

shows $\langle continuous-map \ (pullback-topology \ A1 \ f1 \ T1) \ (pullback-topology \ A2 \ f2 \ T2) \ g \rangle$

$\langle proof \rangle$

lemma *onorm-case-prod-plus-leq*: $\langle onorm \ (case-prod \ plus \ :: \ - \Rightarrow \ 'a::real-normed-vector) \leq \ sqrt \ 2 \rangle$

$\langle proof \rangle$

lemma *bounded-linear-case-prod-plus[simp]*: $\langle bounded-linear \ (case-prod \ plus) \rangle$

$\langle proof \rangle$

lemma *pullback-topology-twice*:

assumes $\langle (f \ -' \ B) \cap A = C \rangle$

shows $\langle pullback-topology \ A \ f \ (pullback-topology \ B \ g \ T) = pullback-topology \ C \ (g \ o \ f) \ T \rangle$

$\langle proof \rangle$

lemma *pullback-topology-homeo-cong*:

assumes $\langle homeomorphic-map \ T \ S \ g \rangle$

assumes $\langle range \ f \subseteq \ topspace \ T \rangle$

shows $\langle pullback-topology \ A \ f \ T = pullback-topology \ A \ (g \ o \ f) \ S \rangle$

$\langle proof \rangle$

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

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

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

includes *lifting-syntax*

assumes $\langle bi-unique \ R \rangle$

shows $\langle (rel-topology \ R \ ==> \ rel-set \ (rel-set \ R)) \ opensets-in \ opensets-in \rangle$

$\langle proof \rangle$

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

includes *lifting-syntax*

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

shows $\langle (rel-topology \ R \ ==> \ (\longleftrightarrow)) \ Hausdorff-space \ Hausdorff-space \rangle$

$\langle proof \rangle$

lemma *sum-cmod-pos*:

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

shows $\langle (\sum x \in A. cmod \ (f \ x)) = cmod \ (\sum x \in A. f \ x) \rangle$

$\langle proof \rangle$

lemma *min-power-distrib-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 $x y :: \langle - :: \text{linordered-semidom} \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *card1I*:
assumes $a \in A$
assumes $\bigwedge x. x \in A \implies x = a$
shows $\langle \text{card } A = 1 \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

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

definition $\langle \text{rel-pred } T P Q = \text{rel-set } T (\text{Collect } P) (\text{Collect } Q) \rangle$

lemma *Collect-parametric[transfer-rule]*:
includes *lifting-syntax*
shows $\langle (\text{rel-pred } T \implies \text{rel-set } T) \text{ Collect Collect} \rangle$
 $\langle \text{proof} \rangle$

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*
 ⟨*proof*⟩

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*⟩

lemma *Domainp-rel-filter*:

assumes ⟨*Domainp r = S*⟩
shows ⟨*Domainp (rel-filter r) F* \longleftrightarrow (*F* \leq *principal (Collect S)*)⟩
 ⟨*proof*⟩

lemma *map-filter-on-cong*:

assumes [*simp*]: ⟨ $\forall_F x \text{ in } F. x \in D$ ⟩
assumes ⟨ $\bigwedge x. x \in D \implies f x = g x$ ⟩
shows ⟨*map-filter-on D f F = map-filter-on D g F*⟩
 ⟨*proof*⟩

lemma *filtermap-cong*:

assumes ⟨ $\forall_F x \text{ in } F. f x = g x$ ⟩
shows ⟨*filtermap f F = filtermap g F*⟩
 ⟨*proof*⟩

lemma *filtermap-INF-eq*:

assumes *inj-f*: ⟨*inj-on f X*⟩
assumes *B-nonempty*: ⟨*B* \neq $\{\}$ ⟩
assumes *F-bounded*: ⟨ $\bigwedge b. b \in B \implies F b \leq \text{principal } X$ ⟩
shows ⟨*filtermap f* ($\bigcap (F \text{ ‘ } B)$) = ($\bigcap b \in B. \text{filtermap } f (F b)$)⟩
 ⟨*proof*⟩

lemma *filtermap-inf-eq*:

assumes ⟨*inj-on f X*⟩
assumes ⟨*F1* \leq *principal X*⟩
assumes ⟨*F2* \leq *principal X*⟩
shows ⟨*filtermap f* (*F1* \sqcap *F2*) = *filtermap f F1* \sqcap *filtermap f F2*⟩
 ⟨*proof*⟩

definition ⟨*transfer-bounded-filter-Inf B M = Inf M* \sqcap *principal B*⟩

lemma *Inf-transfer-bounded-filter-Inf*: $\langle \text{Inf } M = \text{transfer-bounded-filter-Inf UNIV } M \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{Inf } M = \text{transfer-bounded-filter-Inf } B \ M \rangle$
 $\langle \text{proof} \rangle$

lemma *transfer-bounded-filter-Inf-parametric[transfer-rule]*:
includes *lifting-syntax*
fixes $r :: \langle 'rep \Rightarrow 'abs \Rightarrow \text{bool} \rangle$
assumes $[\text{transfer-rule}] : \langle \text{bi-unique } r \rangle$
shows $\langle (\text{rel-set } r \implies \text{rel-set } (\text{rel-filter } r)) \implies \text{rel-filter } r \rangle$
 $\text{transfer-bounded-filter-Inf transfer-bounded-filter-Inf}$
 $\langle \text{proof} \rangle$

definition $\langle \text{transfer-inf-principal } F \ M = F \sqcap \text{principal } M \rangle$

lemma *transfer-inf-principal-parametric[transfer-rule]*:
includes *lifting-syntax*
assumes $[\text{transfer-rule}] : \langle \text{bi-unique } T \rangle$
shows $\langle (\text{rel-filter } T \implies \text{rel-set } T \implies \text{rel-filter } T) \text{ transfer-inf-principal transfer-inf-principal} \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-map-is-continuous-at-point*:
assumes $\langle \text{continuous-map } T \ U \ f \rangle$
shows $\langle \text{filterlim } f \ (\text{nhdsin } U \ (f \ l)) \ (\text{atin } T \ l) \rangle$
 $\langle \text{proof} \rangle$

lemma *set-compr-2-image-collect*: $\langle \{f \ x \ y \mid x \ y. P \ x \ y\} = \text{case-prod } f \ ' \ \text{Collect } (\text{case-prod } P) \rangle$
 $\langle \text{proof} \rangle$

lemma *closure-image-closure*: $\langle \text{continuous-on } (\text{closure } S) \ f \implies \text{closure } (f \ ' \ \text{closure } S) = \text{closure } (f \ ' \ S) \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-reindex-bij-betw*:
assumes *bij-betw* $g \ A \ B$
shows $\langle ((\lambda x. f \ (g \ x)) \text{ has-sum } l) \ A \longleftrightarrow (f \ \text{has-sum } l) \ B \rangle$
 $\langle \text{proof} \rangle$

lemma *enum-inj*:
assumes $i < \text{CARD}('a)$ **and** $j < \text{CARD}('a)$

shows $\langle Enum.enum ! i :: 'a::enum \rangle = Enum.enum ! j \longleftrightarrow i = j$
 $\langle proof \rangle$

lemma *closedin-vimage*:

assumes $\langle closedin U S \rangle$
assumes $\langle continuous-map T U f \rangle$
shows $\langle closedin T (topspace T \cap (f - ' S)) \rangle$
 $\langle proof \rangle$

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

lemma *closedin-if-converge-inside*:

fixes $A :: 'a set$
assumes $AT: \langle A \subseteq topspace T \rangle$
assumes $xA: \langle \bigwedge (F::'a filter) f x. F \neq \perp \implies limitin T f x F \implies range f \subseteq A \implies x \in A \rangle$
shows $\langle closedin T A \rangle$
 $\langle proof \rangle$

lemma *cmod-mono*: $\langle 0 \leq a \implies a \leq b \implies cmod a \leq cmod b \rangle$
 $\langle proof \rangle$

lemma *has-sum-mono-neutral-complex*:

fixes $f :: 'a \Rightarrow complex$
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 $a \leq b$
 $\langle proof \rangle$

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

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 $\langle \forall x. \exists y. Q1 x y \wedge Q2 x y \wedge Q3 x y \rangle$
 $\langle proof \rangle$

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 $\langle \forall x. \exists y. Q1 x y \wedge Q2 x y \wedge Q3 x y \wedge Q4 x y \rangle$
 $\langle proof \rangle$

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 $\langle \forall x. \exists y. Q1 x y \wedge Q2 x y \wedge Q3 x y \wedge Q4 x y \wedge Q5 x y \rangle$
 $\langle proof \rangle$

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

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 $\langle is-Sup\ X\ s \rangle$
 $\langle proof \rangle$

lemma *is-Sup-unique*: $\langle is-Sup\ X\ a \implies is-Sup\ X\ b \implies a=b \rangle$
 $\langle proof \rangle$

lemma *has-Sup-bdd-above*: $\langle has-Sup\ X \implies bdd-above\ X \rangle$
 $\langle proof \rangle$

lemma *is-Sup-has-Sup*: $\langle is-Sup\ X\ s \implies has-Sup\ X \rangle$
 $\langle proof \rangle$

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 $\langle s = Sup\ X \rangle$
 $\langle proof \rangle$

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 $\langle is-Sup\ X\ (Sup\ X) \rangle$
 $\langle proof \rangle$

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 $\langle continuous-map\ T\ U\ f \rangle$
 $\langle proof \rangle$

lemma *SMT-choices*:

— Was included as SMT.choices in Isabelle and disappeared
 $\bigwedge Q. \forall x. \exists y\ ya. Q\ x\ y\ ya \implies \exists f\ fa. \forall x. Q\ x\ (f\ x)\ (fa\ x)$
 $\bigwedge Q. \forall x. \exists y\ ya\ yb. Q\ x\ y\ ya\ yb \implies \exists f\ fa\ fb. \forall x. Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)$
 $\bigwedge Q. \forall x. \exists y\ ya\ yb\ yc. Q\ x\ y\ ya\ yb\ yc \implies \exists f\ fa\ fb\ fc. \forall x. Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)$
 $\bigwedge Q. \forall x. \exists y\ ya\ yb\ yc\ yd. Q\ x\ y\ ya\ yb\ yc\ yd \implies$
 $\quad \exists f\ fa\ fb\ fc\ fd. \forall x. Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)$
 $\bigwedge Q. \forall x. \exists y\ ya\ yb\ yc\ yd\ ye. Q\ x\ y\ ya\ yb\ yc\ yd\ ye \implies$
 $\quad \exists f\ fa\ fb\ fc\ fd\ fe. \forall x. Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)\ (fe\ x)$
 $\bigwedge Q. \forall x. \exists y\ ya\ yb\ yc\ yd\ ye\ yf. Q\ x\ y\ ya\ yb\ yc\ yd\ ye\ yf \implies$

$\exists f \text{ fa fb fc fd fe ff}. \forall x. Q \ x \ (f \ x) \ (fa \ x) \ (fb \ x) \ (fc \ x) \ (fd \ x) \ (fe \ x) \ (ff \ x)$
 $\wedge Q. \forall x. \exists y \ ya \ yb \ yc \ yd \ ye \ yf \ yg. Q \ x \ y \ ya \ yb \ yc \ yd \ ye \ yf \ yg \implies$
 $\exists f \text{ fa fb fc fd fe ff fg}. \forall x. Q \ x \ (f \ x) \ (fa \ x) \ (fb \ x) \ (fc \ x) \ (fd \ x) \ (fe \ x) \ (ff \ x) \ (fg \ x)$
 <proof>

lemma *closedin-pullback-topology*:

$\text{closedin} \ (\text{pullback-topology} \ A \ f \ T) \ S \longleftrightarrow (\exists C. \text{closedin} \ T \ C \wedge S = f \text{--}'C \cap A)$
 <proof>

lemma *regular-space-pullback[intro]*:

assumes <regular-space T >
shows <regular-space $(\text{pullback-topology} \ A \ f \ T)$ >
 <proof>

lemma *t3-space-euclidean-regular[iff]*: <regular-space $(\text{euclidean} \ :: \ 'a::t3\text{-space} \ \text{topology})$ >
 <proof>

definition *increasing-filter* :: <'a::order filter \Rightarrow bool> **where**

— Definition suggested by [5]
 <increasing-filter $F \longleftrightarrow (\forall_F \ x \ \text{in} \ F. \forall_F \ y \ \text{in} \ F. y \geq x)$ >

lemma *increasing-filtermap*:

fixes $F \ :: \ \langle 'a::\text{order} \ \text{filter} \rangle$ **and** $f \ :: \ \langle 'a \Rightarrow 'b::\text{order} \rangle$ **and** $X \ :: \ \langle 'a \ \text{set} \rangle$
assumes *increasing*: <increasing-filter F >
assumes *mono*: <mono-on $X \ f$ >
assumes *ev-X*: <eventually $(\lambda x. x \in X) \ F$ >
shows <increasing-filter $(\text{filtermap} \ f \ F)$ >
 <proof>

lemma *increasing-finite-subsets-at-top[simp]*: <increasing-filter $(\text{finite-subsets-at-top} \ X)$ >
 <proof>

lemma *monotone-convergence*:

— Following [5]
fixes $f \ :: \ \langle 'b \Rightarrow 'a::\{\text{order-topology}, \ \text{conditionally-complete-linorder}\} \rangle$
assumes *bounded*: < $\forall_F \ x \ \text{in} \ F. f \ x \leq B$ >
assumes *increasing*: <increasing-filter $(\text{filtermap} \ f \ F)$ >
shows < $\exists l. (f \longrightarrow l) \ F$ >
 <proof>

lemma *monotone-convergence-complex*:

fixes $f \ :: \ \langle 'b \Rightarrow \text{complex} \rangle$
assumes *bounded*: < $\forall_F \ x \ \text{in} \ F. f \ x \leq B$ >
assumes *increasing*: <increasing-filter $(\text{filtermap} \ f \ F)$ >
shows < $\exists l. (f \longrightarrow l) \ F$ >
 <proof>

lemma *compact-closed-subset*:

assumes $\langle \text{compact } s \rangle$
assumes $\langle \text{closed } t \rangle$
assumes $\langle t \subseteq s \rangle$
shows $\langle \text{compact } t \rangle$
 $\langle \text{proof} \rangle$

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** $S :: \langle 'a::\text{metric-space set} \rangle$
 $\langle \text{proof} \rangle$

lemma *ex-norm1-not-singleton*:

shows $\langle \exists x::'a::\{\text{real-normed-vector, not-singleton}\}. \text{norm } x = 1 \rangle$
 $\langle \text{proof} \rangle$

lemma *is-Sup-approx-below*:

fixes $b :: \langle 'a::\text{linordered-ab-group-add} \rangle$
assumes $\langle \text{is-Sup } A \ b \rangle$
assumes $\langle \varepsilon > 0 \rangle$
shows $\langle \exists x \in A. b - \varepsilon \leq x \rangle$
 $\langle \text{proof} \rangle$

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 $\langle s \leq t \rangle$
 $\langle \text{proof} \rangle$

lemma *infsum-single*:

assumes $\bigwedge j. j \neq i \implies j \in A \implies f \ j = 0$
shows $\text{infsum } f \ A = (\text{if } i \in A \ \text{then } f \ i \ \text{else } 0)$
 $\langle \text{proof} \rangle$

lemma *suminf-eqI*:

fixes $x :: \langle -::\{\text{comm-monoid-add, t2-space}\} \rangle$
assumes $\langle f \ \text{sums } x \rangle$
shows $\langle \text{suminf } f = x \rangle$
 $\langle \text{proof} \rangle$

lemma *suminf-If-finite-set*:

fixes $f :: \langle - \Rightarrow -::\{\text{comm-monoid-add, t2-space}\} \rangle$
assumes $\langle \text{finite } F \rangle$
shows $\langle (\sum x \in F. f \ x) = (\sum x. \text{if } x \in F \ \text{then } f \ x \ \text{else } 0) \rangle$
 $\langle \text{proof} \rangle$

definition *separating-set* :: $\langle ('a \Rightarrow 'b) \Rightarrow \text{bool} \rangle \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$ **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*: $\langle S \subseteq T \Longrightarrow \text{separating-set } P \ S \Longrightarrow \text{separating-set } P \ T \rangle$
 $\langle \text{proof} \rangle$

lemma *separating-set-UNIV[simp]*: $\langle \text{separating-set } P \ \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle f = g \rangle$
 $\langle \text{proof} \rangle$

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 $\langle f = g \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{separating-set } P \ S \rangle$
 $\langle \text{proof} \rangle$

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 $y \leq x$
 $\langle \text{proof} \rangle$

lemma *bdd-above-mono2*:
assumes $\langle \text{bdd-above } (g \ ' B) \rangle$
assumes $\langle A \subseteq B \rangle$
assumes $\langle \bigwedge x. x \in A \Longrightarrow f \ x \leq g \ x \rangle$
shows $\langle \text{bdd-above } (f \ ' A) \rangle$
 $\langle \text{proof} \rangle$

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
    ⟨proof⟩

  lemma rank1-scaleR[simp]: ⟨rank1 (c *R a)⟩ if ⟨rank1 a⟩ and ⟨c ≠ 0⟩
    ⟨proof⟩

  lemma rank1-butterfly[simp]: ⟨rank1 (butterfly x y)⟩
    ⟨proof⟩

  definition ⟨cfinite-dim S ⟷ (∃ B. finite B ∧ S ⊆ cspan B)⟩

  lemma cfinite-dim-subspace-has-basis:
    assumes ⟨cfinite-dim S⟩ and ⟨csubspace S⟩
    shows ⟨∃ B. finite B ∧ cindependent B ∧ cspan B = S⟩
    ⟨proof⟩

  lemma cfinite-dim-subspace-has-onb:
    assumes ⟨cfinite-dim S⟩ and ⟨csubspace S⟩
    shows ⟨∃ B. finite B ∧ is-ortho-set B ∧ cspan B = S ∧ (∀ x∈B. norm x = 1)⟩
    ⟨proof⟩

  lemma cspan-finite-dim[intro]: ⟨cfinite-dim (cspan B)⟩ if ⟨finite B⟩
    ⟨proof⟩

  lift-definition finite-dim-ccsubspace :: ⟨'a::complex-normed-vector csubspace ⇒ bool⟩ is cfi-
  nite-dim⟨proof⟩

  lemma ccspan-finite-dim[intro]: ⟨finite-dim-ccsubspace (ccspan B)⟩ if ⟨finite B⟩
    ⟨proof⟩

  lemma finite-dim-ccsubspace-zero[iff]: ⟨finite-dim-ccsubspace 0⟩
    ⟨proof⟩

```

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

lemma *compact-scaleC*:
fixes $s :: 'a::\text{complex-normed-vector set}$
assumes $\text{compact } s$
shows $\text{compact } (\text{scaleC } c \ 's)$
 $\langle \text{proof} \rangle$

lemma *Proj-nearest*:
assumes $\langle x \in \text{space-as-set } S \rangle$
shows $\langle \text{dist } (\text{Proj } S \ m) \ m \leq \text{dist } x \ m \rangle$
 $\langle \text{proof} \rangle$

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

lemma *cblinfun-norm-is-Sup-cinner*:
— [2], Proposition II.2.13
fixes $A :: \langle 'a::\{\text{not-singleton, hilbert-space}\} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{selfadjoint } A \rangle$
shows $\langle \text{is-Sup } ((\lambda \psi. \text{cmod } (\psi \cdot_C (A *_V \psi))) \ ' \{\psi. \text{norm } \psi = 1\}) \ (\text{norm } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *cblinfun-norm-approx-witness-cinner*:
fixes $A :: \langle 'a::\{\text{not-singleton, hilbert-space}\} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{selfadjoint } A \rangle$ **and** $\langle \varepsilon > 0 \rangle$
shows $\langle \exists \psi. \text{cmod } (\psi \cdot_C (A *_V \psi)) \geq \text{norm } A - \varepsilon \wedge \text{norm } \psi = 1 \rangle$
 $\langle \text{proof} \rangle$

lemma *cblinfun-norm-approx-witness-cinner'*:
fixes $A :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{selfadjoint } A \rangle$ **and** $\langle \varepsilon > 0 \rangle$
shows $\langle \exists \psi. \text{cmod } (\psi \cdot_C A \ \psi) / (\text{norm } \psi)^2 \geq \text{norm } A - \varepsilon \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-mono-neutral-cblinfun*:
fixes $f :: 'a \Rightarrow ('b::\text{hilbert-space} \Rightarrow_{CL} 'b)$
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 $a \leq b$
 $\langle proof \rangle$

lemma *sums-mono-cblinfun*:
fixes $f :: nat \Rightarrow ('b :: \text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\langle f \text{ sums } a \rangle$ **and** $g \text{ sums } b$
assumes $\langle \bigwedge n. f n \leq g n \rangle$
shows $a \leq b$
 $\langle proof \rangle$

lemma *scaleC-scaleR-commute*: $\langle a *_C b *_R x = b *_R a *_C x \rangle$ **for** $x :: \langle - :: \text{complex-normed-vector} \rangle$
 $\langle proof \rangle$

lemma *sandwich-scaleC-left*: $\langle sandwich (c *_C e) = (c \text{ mod } c)^{\wedge 2} *_C sandwich e \rangle$
 $\langle proof \rangle$

lemma *sandwich-scaleR-left*: $\langle sandwich (r *_R e) = r^{\wedge 2} *_R sandwich e \rangle$
 $\langle proof \rangle$

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 $\langle (\sum_{\infty x \in X}. f x) * (\sum_{\infty y \in Y}. g y) = (\sum_{\infty (x,y) \in X \times Y}. f x * g y) \rangle$
 $\langle proof \rangle$

lemma *infsun-product'*:
fixes $f :: \langle 'a \Rightarrow 'c :: \{ \text{banach, times, real-normed-algebra} \} \rangle$ **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 $\langle (\sum_{\infty x \in X}. f x) * (\sum_{\infty y \in Y}. g y) = (\sum_{\infty (x,y) \in X \times Y}. f x * g y) \rangle$
 $\langle proof \rangle$

lemma *Proj-o-Proj-subspace-right*:
assumes $\langle A \geq B \rangle$
shows $\langle Proj A \text{ o}_{CL} Proj B = Proj B \rangle$
 $\langle proof \rangle$

lemma *Proj-o-Proj-subspace-left*:
assumes $\langle A \leq B \rangle$
shows $\langle Proj A \text{ o}_{CL} Proj B = Proj A \rangle$
 $\langle proof \rangle$

lemma *orthogonal-spaces-SUP-left*:
assumes $\langle \bigwedge x. x \in X \implies \text{orthogonal-spaces } (A x) B \rangle$
shows $\langle \text{orthogonal-spaces } (\bigsqcup_{x \in X}. A x) B \rangle$
 $\langle proof \rangle$

lemma *orthogonal-spaces-SUP-right*:
assumes $\langle \bigwedge x. x \in X \implies \text{orthogonal-spaces } A (B x) \rangle$
shows $\langle \text{orthogonal-spaces } A (\bigsqcup_{x \in X}. B x) \rangle$
 $\langle \text{proof} \rangle$

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

lemma *infsun-bounded-linear-invertible*:
assumes $\langle \text{bounded-linear } h \rangle$
assumes $\langle \text{bounded-linear } h' \rangle$
assumes $\langle h' \circ h = \text{id} \rangle$
shows $\langle \text{infsun } (\lambda x. h (f x)) A = h (\text{infsun } f A) \rangle$
 $\langle \text{proof} \rangle$

lemma *cblinfun-eq-from-separatingI*:
fixes $a b :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector} \rangle$
assumes $\langle \text{separating-set } (\text{bounded-clinear} :: ('a \Rightarrow 'b) \Rightarrow \text{bool}) S \rangle$
assumes $\langle \bigwedge x. x \in S \implies a x = b x \rangle$
shows $\langle a = b \rangle$
 $\langle \text{proof} \rangle$

lemma *cblinfun-eq-from-separatingI2*:
fixes $a b :: \langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector} \rangle$
assumes $\langle \text{separating-set } (\text{bounded-clinear} :: ('a \Rightarrow 'b) \Rightarrow \text{bool}) ((\lambda(x,y). h x y) \text{ ` } (S \times T)) \rangle$
assumes $\langle \bigwedge x y. x \in S \implies y \in T \implies a (h x y) = b (h x y) \rangle$
shows $\langle a = b \rangle$
 $\langle \text{proof} \rangle$

lemma *separating-set-bounded-clinear-dense*:
assumes $\langle \text{ccspan } S = \top \rangle$
shows $\langle \text{separating-set bounded-clinear } S \rangle$
 $\langle \text{proof} \rangle$

lemma *separating-set-ket*: $\langle \text{separating-set bounded-clinear } (\text{range ket}) \rangle$
 $\langle \text{proof} \rangle$

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

lemma *separating-set-bounded-linear-antilinear*:
assumes $\langle \text{separating-set (bounded-clinear :: } (- \Rightarrow 'e::\text{complex-normed-vector conjugate-space}) \Rightarrow -) A \rangle$
shows $\langle \text{separating-set (bounded-antilinear :: } (- \Rightarrow 'e) \Rightarrow -) A \rangle$
 $\langle \text{proof} \rangle$

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

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 $\langle \bigwedge h. h \in \text{space-as-set } (\bigsqcup_{i \in I} X i) \implies a h = b h \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \bigwedge h. h \in \text{space-as-set } (S \sqcup T) \implies a h = b h \rangle$
 $\langle \text{proof} \rangle$

lemma *ccsubspace-contains-unit*:
assumes $\langle E \neq \perp \rangle$
shows $\langle \exists h \in \text{space-as-set } E. \text{norm } h = 1 \rangle$
 $\langle \text{proof} \rangle$

lemma *Proj-0-compl*: $\langle \text{Proj } S x = 0 \rangle$ **if** $\langle x \in \text{space-as-set } (-S) \rangle$
 $\langle \text{proof} \rangle$

lemma *csubspace-has-basis*:
assumes $\langle \text{csubspace } S \rangle$
shows $\langle \exists B. \text{cindependent } B \wedge \text{cspan } B = S \rangle$
 $\langle \text{proof} \rangle$

lemma *inj-scaleC*:
fixes $A :: \langle 'a::\text{complex-vector set} \rangle$
assumes $\langle c \neq 0 \rangle$
shows $\langle \text{inj-on (scaleC } c) A \rangle$
 $\langle \text{proof} \rangle$

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

lemma *diagonal-operator-exists*:
assumes $\langle \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (f x))) \rangle$
shows $\langle \text{explicit-cblinfun-exists } (\lambda x y. \text{of-bool } (x = y) * f x) \rangle$
 $\langle \text{proof} \rangle$

lemma *diagonal-operator-ket*:
assumes $\langle \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (f x))) \rangle$
shows $\langle \text{diagonal-operator } f (\text{ket } x) = f x *_C \text{ket } x \rangle$
 $\langle \text{proof} \rangle$

lemma *diagonal-operator-invalid*:
assumes $\langle \neg \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (f x))) \rangle$
shows $\langle \text{diagonal-operator } f = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *diagonal-operator-adj*: $\langle \text{diagonal-operator } f^* = \text{diagonal-operator } (\lambda x. \text{cnj } (f x)) \rangle$
 $\langle \text{proof} \rangle$

lemma *diagonal-operator-comp*:
assumes $\langle \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (f x))) \rangle$
assumes $\langle \text{bdd-above } (\text{range } (\lambda x. \text{cmod } (g x))) \rangle$
shows $\langle \text{diagonal-operator } f \circ_{CL} \text{diagonal-operator } g = \text{diagonal-operator } (\lambda x. (f x * g x)) \rangle$
 $\langle \text{proof} \rangle$

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

lemma *separating-set-clinear-cspan*:
assumes $\langle \text{cspan } S = UNIV \rangle$
shows $\langle \text{separating-set clinear } S \rangle$
 $\langle \text{proof} \rangle$

lemma *less-eq-cblinfunI*:
fixes $a b :: \langle 'a \Rightarrow_{CL} 'a :: \text{chilbert-space} \rangle$
assumes $\langle \bigwedge h. h \cdot_C a h \leq h \cdot_C b h \rangle$
shows $\langle a \leq b \rangle$
 $\langle \text{proof} \rangle$

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 =  $\langle UNIV :: ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector}) \text{ set} \rangle$ 
   $\langle \text{proof} \rangle$ 
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  $\langle \text{scaleC} \rangle$   $\langle \text{proof} \rangle$ 
```

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

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

```
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  $\langle \text{proof} \rangle$ 
```

```
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  $\langle \text{proof} \rangle$ 
```

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

```
instance
```

```
   $\langle \text{proof} \rangle$ 
```

```
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  $\langle \text{openin cstrong-operator-topology} \rangle$ 
   $\langle \text{proof} \rangle$ 
```

```
instance
```

```
   $\langle \text{proof} \rangle$ 
```

```
end
```

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

```
  includes lifting-syntax
```

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

```
   $\langle \text{proof} \rangle$ 
```

lemma *filterlim-cstrong-operator-topology*: $\langle \text{filterlim } f \text{ (nhdsin cstrong-operator-topology } l) = \text{limitin cstrong-operator-topology } f \text{ } l \rangle$
 $\langle \text{proof} \rangle$

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

instance *cblinfun-sot* :: $(\text{complex-normed-vector}, \text{complex-normed-vector}) \text{ } t2\text{-space}$
 $\langle \text{proof} \rangle$

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

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

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

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

includes *lifting-syntax*

shows $\langle (\text{rel-topology cr-cblinfun-sot}) \text{ cstrong-operator-topology euclidean} \rangle$

$\langle \text{proof} \rangle$

lemma *openin-cstrong-operator-topology*: $\langle \text{openin cstrong-operator-topology } U \longleftrightarrow (\exists V. \text{open } V \wedge U = (*_V) -' V) \rangle$
 $\langle \text{proof} \rangle$

lemma *cstrong-operator-topology-plus-cont*: $\langle \text{LIM } (x, y) \text{ nhdsin cstrong-operator-topology } a \times_F \text{ nhdsin cstrong-operator-topology } b.$

$x + y \text{ } :> \text{ nhdsin cstrong-operator-topology } (a + b) \rangle$

$\langle \text{proof} \rangle$

instance *cblinfun-sot* :: $(\text{complex-normed-vector}, \text{complex-normed-vector}) \text{ } \text{topological-group-add}$
 $\langle \text{proof} \rangle$

lemma *continuous-map-left-comp-sot[continuous-intros]*:

fixes $b :: \langle 'b :: \text{complex-normed-vector} \Rightarrow_{CL} 'c :: \text{complex-normed-vector} \rangle$

and $f :: \langle 'a \Rightarrow 'd :: \text{complex-normed-vector} \Rightarrow_{CL} 'b \rangle$

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

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

$\langle \text{proof} \rangle$

lemma *continuous-cstrong-operator-topology-plus[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 $\langle \text{continuous-map } T \text{ cstrong-operator-topology } (\lambda x. f x + g x) \rangle$

⟨proof⟩

lemma *continuous-cstrong-operator-topology-uminus*[*continuous-intros*]:

assumes ⟨*continuous-map* T *cstrong-operator-topology* f ⟩

shows ⟨*continuous-map* T *cstrong-operator-topology* $(\lambda x. - f x)$ ⟩

⟨proof⟩

lemma *continuous-cstrong-operator-topology-minus*[*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* $(\lambda x. f x - g x)$ ⟩

⟨proof⟩

lemma *continuous-map-right-comp-sot*[*continuous-intros*]:

assumes ⟨*continuous-map* T *cstrong-operator-topology* f ⟩

shows ⟨*continuous-map* T *cstrong-operator-topology* $(\lambda x. f x \circ_{CL} a)$ ⟩

⟨proof⟩

lemma *continuous-map-scaleC-sot*[*continuous-intros*]:

assumes ⟨*continuous-map* T *cstrong-operator-topology* f ⟩

shows ⟨*continuous-map* T *cstrong-operator-topology* $(\lambda x. c *_C f x)$ ⟩

⟨proof⟩

lemma *continuous-scaleC-sot*[*continuous-intros*]:

fixes $f :: \langle 'a::\text{topological-space} \Rightarrow (-,-) \text{cblinfun-sot} \rangle$

assumes ⟨*continuous-on* X f ⟩

shows ⟨*continuous-on* X $(\lambda x. c *_C f x)$ ⟩

⟨proof⟩

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 ⟨*csubspace* A ⟩

shows ⟨*csubspace* $(\text{closure } A)$ ⟩

⟨proof⟩

include *lattice-syntax*

⟨proof⟩

lemma *limitin-cstrong-operator-topology*:

⟨*limitin cstrong-operator-topology* f l $F \iff (\forall i. ((\lambda j. f j *_V i) \longrightarrow l *_V i) F)$ ⟩

⟨proof⟩

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

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

shows ⟨ $b \in \text{cstrong-operator-topology closure-of } A$ ⟩

⟨proof⟩

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

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set } \text{cr-cblinfun-sot} \implies (=)) \ \text{csubspace } \text{csubspace} \rangle$
 $\langle \text{proof} \rangle$

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set } \text{cr-cblinfun-sot} \implies (=)) \ (\text{closedin } \text{cstrong-operator-topology}) \ \text{closed} \rangle$
 $\langle \text{proof} \rangle$

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set } \text{cr-cblinfun-sot} \implies \text{rel-set } \text{cr-cblinfun-sot}) \ (\text{Abstract-Topology.closure-of } \text{cstrong-operator-topology}) \ \text{closure} \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *has-sum-in-cstrong-operator-topology*:
 $\langle \text{has-sum-in } \text{cstrong-operator-topology } f \ A \ l \longleftrightarrow (\forall \psi. ((\lambda i. f \ i \ *_{V} \ \psi) \ \text{has-sum } (l \ *_{V} \ \psi)) \ A) \rangle$
 $\langle \text{proof} \rangle$

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

declare *cstrong-operator-topology-topospace* [*simp*]

lift-definition *cblinfun-compose-sot* :: $\langle 'a::\text{complex-normed-vector}, 'b::\text{complex-normed-vector} \rangle$
cblinfun-sot \Rightarrow $\langle 'c::\text{complex-normed-vector}, 'a \rangle$ *cblinfun-sot* \Rightarrow $\langle 'c, 'b \rangle$ *cblinfun-sot*
is *cblinfun-compose* $\langle \text{proof} \rangle$

lemma *isCont-cblinfun-compose-sot-right[simp]*: $\langle \text{isCont } (\lambda F. \text{cblinfun-compose-sot } F \ G) \ x \rangle$
 $\langle \text{proof} \rangle$

lemma *isCont-cblinfun-compose-sot-left[simp]*: $\langle \text{isCont } (\lambda F. \text{cblinfun-compose-sot } G \ F) \ x \rangle$
 $\langle \text{proof} \rangle$

lemma *additive-cblinfun-compose-sot-right[simp]*: $\langle \text{additive } (\lambda F. \text{cblinfun-compose-sot } F \ G) \rangle$
 $\langle \text{proof} \rangle$

lemma *additive-cblinfun-compose-sot-left[simp]*: $\langle \text{additive } (\lambda F. \text{cblinfun-compose-sot } G \ F) \rangle$
 $\langle \text{proof} \rangle$

lemma *transfer-infsum-sot[transfer-rule]*:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } R \rangle$
shows $\langle ((R \text{====} \Rightarrow \text{cr-cblinfun-sot}) \text{====} \Rightarrow \text{rel-set } R \text{====} \Rightarrow \text{cr-cblinfun-sot}) \ (\text{infsum-in cstrong-operator-topology}) \ \text{infsum} \rangle$
 $\langle \text{proof} \rangle$

lemma *transfer-summable-on-sot[transfer-rule]*:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } R \rangle$
shows $\langle ((R \text{====} \Rightarrow \text{cr-cblinfun-sot}) \text{====} \Rightarrow \text{rel-set } R \text{====} \Rightarrow (\leftarrow \rightarrow)) \ (\text{summable-on-in cstrong-operator-topology}) \ (\text{summable-on}) \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-sot-cont[continuous-intros]*:
assumes $\langle \text{continuous-map } T \ \text{cstrong-operator-topology } f \rangle$
shows $\langle \text{continuous-map } T \ \text{cstrong-operator-topology } (\lambda x. \text{sandwich } A \ (f \ x)) \rangle$
 $\langle \text{proof} \rangle$

lemma *closed-map-sot-unitary-sandwich*:
fixes $U :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
assumes $\langle \text{unitary } U \rangle$
shows $\langle \text{closed-map cstrong-operator-topology cstrong-operator-topology } (\lambda x. \text{sandwich } U \ x) \rangle$
 $\langle \text{proof} \rangle$

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** $\langle A \geq 0 \rangle$
 $\langle \text{proof} \rangle$

definition *sqrt-op* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \rangle \Rightarrow \langle 'a \Rightarrow_{CL} 'a \rangle$ **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** $\langle \neg a \geq 0 \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *sandwich-pos[intro]*: $\langle \text{sandwich } b a \geq 0 \rangle$ **if** $\langle a \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *cblinfun-power-pos*: $\langle \text{cblinfun-power } a n \geq 0 \rangle$ **if** $\langle a \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *sqrt-op-existence*:

fixes *A* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a::\text{hilbert-space} \rangle$

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

shows $\langle \exists B. B \geq 0 \wedge B o_{CL} B = A \wedge (\forall F. A o_{CL} F = F o_{CL} A \longrightarrow B o_{CL} F = F o_{CL} B) \wedge B \in \text{closure } (\text{cspan } (\text{range } (\text{cblinfun-power } A))) \rangle$

$\langle \text{proof} \rangle$

lemma *wecken35hilfssatz*:

— Auxiliary lemma from [9]

$\langle \exists P. \text{is-Proj } P \wedge (\forall F. F \circ_{CL} (W - T) = (W - T) \circ_{CL} F \longrightarrow F \circ_{CL} P = P \circ_{CL} F)$
 $\wedge (\forall f. W f = 0 \longrightarrow P f = f)$
 $\wedge (W = (\mathcal{Q} *_C P - \text{id-cblinfun}) \circ_{CL} T) \rangle$

if *WT-comm*: $\langle W \circ_{CL} T = T \circ_{CL} W \rangle$ **and** $\langle W = W^* \rangle$ **and** $\langle T = T^* \rangle$
and *WW-TT*: $\langle W \circ_{CL} W = T \circ_{CL} T \rangle$

for $W T :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \rangle$

$\langle \text{proof} \rangle$

lemma *sqrt-op-pos[simp]*: $\langle \text{sqrt-op } a \geq 0 \rangle$

$\langle \text{proof} \rangle$

lemma *sqrt-op-square[simp]*:

assumes $\langle a \geq 0 \rangle$

shows $\langle \text{sqrt-op } a \circ_{CL} \text{sqrt-op } a = a \rangle$

$\langle \text{proof} \rangle$

lemma *sqrt-op-unique*:

— Proof follows [9]

assumes $\langle b \geq 0 \rangle$ **and** $\langle b^* \circ_{CL} b = a \rangle$

shows $\langle b = \text{sqrt-op } a \rangle$

$\langle \text{proof} \rangle$

lemma *sqrt-op-in-closure*: $\langle \text{sqrt-op } a \in \text{closure } (\text{cspan } (\text{range } (\text{cblinfun-power } a))) \rangle$

$\langle \text{proof} \rangle$

lemma *sqrt-op-commute*:

assumes $\langle A \geq 0 \rangle$

assumes $\langle A \circ_{CL} F = F \circ_{CL} A \rangle$

shows $\langle \text{sqrt-op } A \circ_{CL} F = F \circ_{CL} \text{sqrt-op } A \rangle$

$\langle \text{proof} \rangle$

lemma *sqrt-op-0[simp]*: $\langle \text{sqrt-op } 0 = 0 \rangle$

$\langle \text{proof} \rangle$

lemma *sqrt-op-scaleC*:

assumes $\langle c \geq 0 \rangle$ **and** $\langle a \geq 0 \rangle$

shows $\langle \text{sqrt-op } (c *_C a) = \text{sqrt } c *_C \text{sqrt-op } a \rangle$

$\langle \text{proof} \rangle$

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

lemma *abs-op-pos[simp]*: $\langle \text{abs-op } a \geq 0 \rangle$

$\langle \text{proof} \rangle$

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

⟨proof⟩

lemma *abs-op-idem[simp]*: ⟨ $abs-op (abs-op a) = abs-op a$ ⟩
⟨proof⟩

lemma *abs-op-uminus[simp]*: ⟨ $abs-op (- a) = abs-op a$ ⟩
⟨proof⟩

lemma *selfbutter-pos[simp]*: ⟨ $selfbutter x \geq 0$ ⟩
⟨proof⟩

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

lemma *abs-op-nondegenerate*: ⟨ $a = 0$ ⟩ **if** ⟨ $abs-op a = 0$ ⟩
⟨proof⟩

lemma *abs-op-scaleC*: ⟨ $abs-op (c *_C a) = |c| *_C abs-op a$ ⟩
⟨proof⟩

lemma *kernel-abs-op[simp]*: ⟨ $kernel (abs-op a) = kernel a$ ⟩
⟨proof⟩

definition *polar-decomposition where*

— [1], 3.9 Polar Decomposition
⟨*polar-decomposition* $A = cblinfun-extension (range (abs-op A)) (\lambda\psi. A *_V inv (abs-op A) \psi)$ $o_{CL} Proj (abs-op A *_S top)$ ⟩
for $A :: \langle 'a::chilbert-space \Rightarrow_{CL} 'b::complex-inner \rangle$

lemma

fixes $A :: \langle 'a :: chilbert-space \Rightarrow_{CL} 'b :: chilbert-space \rangle$
— [1], 3.9 Polar Decomposition
shows *polar-decomposition-correct*: ⟨*polar-decomposition* $A o_{CL} abs-op A = A$ ⟩
and *polar-decomposition-final-space*: ⟨*polar-decomposition* $A *_S top = A *_S top$ ⟩
and *polar-decomposition-initial-space[simp]*: ⟨ $kernel (polar-decomposition A) = kernel A$ ⟩
and *polar-decomposition-partial-isometry[simp]*: ⟨*partial-isometry* (*polar-decomposition* A)⟩
⟨proof⟩

lemma *polar-decomposition-correct'*: ⟨(*polar-decomposition* A)* $o_{CL} A = abs-op A$ ⟩
for $A :: \langle 'a :: chilbert-space \Rightarrow_{CL} 'b :: chilbert-space \rangle$
⟨proof⟩

lemma *abs-op-adj*: ⟨ $abs-op (a*) = sandwich (polar-decomposition a) (abs-op a)$ ⟩
⟨proof⟩

lemma *abs-opI*:

assumes $\langle a^* \circ_{CL} a = b^* \circ_{CL} b \rangle$
assumes $\langle a \geq 0 \rangle$
shows $\langle a = \text{abs-op } b \rangle$
 $\langle \text{proof} \rangle$

lemma *abs-op-id-on-pos*: $\langle a \geq 0 \implies \text{abs-op } a = a \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-abs-op[simp]*: $\langle \text{norm } (\text{abs-op } a) = \text{norm } a \rangle$
for $a :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{partial-isometry } A \iff \text{is-Proj } (A^* \circ_{CL} A) \rangle$
 $\langle \text{proof} \rangle$

lemma *abs-op-square*: $\langle (\text{abs-op } A)^* \circ_{CL} \text{abs-op } A = A^* \circ_{CL} A \rangle$
 $\langle \text{proof} \rangle$

lemma *polar-decomposition-0[simp]*: $\langle \text{polar-decomposition } 0 = (0 :: 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space}) \rangle$
 $\langle \text{proof} \rangle$

lemma *polar-decomposition-unique*:
fixes $A :: \langle 'a :: \text{chilbert-space} \Rightarrow_{CL} 'b :: \text{chilbert-space} \rangle$
assumes $\langle \text{kernel } X = \text{kernel } A \rangle$
assumes $\langle X \circ_{CL} \text{abs-op } A = A \rangle$
shows $\langle X = \text{polar-decomposition } A \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{norm } A \leq \text{norm } B \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-mono*: $\langle \text{sandwich } A B \leq \text{sandwich } A C \rangle$ **if** $\langle B \leq C \rangle$
 $\langle \text{proof} \rangle$

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 $a \geq 0$
<proof>

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 $x \leq y$
<proof>

lemma *infsun-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 $\text{infsun } f A \leq \text{infsun } g A$
<proof>

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 $\text{suminf } f \leq \text{suminf } g$
<proof>

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 $\text{suminf } f \geq 0$
<proof>

lemma *infsun-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 $\text{infsun } f A \leq \text{infsun } g B$
<proof>

lemma *abs-op-geq:* $\langle \text{abs-op } a \geq a \rangle$ **if** $\langle \text{selfadjoint } a \rangle$
<proof>

lemma *abs-op-geq-neg:* $\langle \text{abs-op } a \geq -a \rangle$ **if** $\langle \text{selfadjoint } a \rangle$
<proof>

lemma *infsun-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 $\text{infsum } f M \geq 0$
 $\langle \text{proof} \rangle$

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

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 $\langle A *_S S \leq T \rangle$
 $\langle \text{proof} \rangle$

lemma *abs-op-plus-orthogonal*:
assumes $\langle a^* o_{CL} b = 0 \rangle$ **and** $\langle a o_{CL} b^* = 0 \rangle$
shows $\langle \text{abs-op } (a + b) = \text{abs-op } a + \text{abs-op } b \rangle$
 $\langle \text{proof} \rangle$

definition *pos-op* :: $\langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'a \Rightarrow 'a \Rightarrow_{CL} 'a \rangle$ **where**
 $\langle \text{pos-op } a = (\text{abs-op } a + a) /_R 2 \rangle$

definition *neg-op* :: $\langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'a \Rightarrow 'a \Rightarrow_{CL} 'a \rangle$ **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 $\langle \text{pos-op } a \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *neg-op-pos*:
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{neg-op } a \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *pos-op-neg-op-ortho*:
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{pos-op } a o_{CL} \text{neg-op } a = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *pos-op-plus-neg-op*: $\langle \text{pos-op } a + \text{neg-op } a = \text{abs-op } a \rangle$
 $\langle \text{proof} \rangle$

lemma *pos-op-minus-neg-op*: $\langle \text{pos-op } a - \text{neg-op } a = a \rangle$
 $\langle \text{proof} \rangle$

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** $\langle c = \text{neg-op } a \rangle$
 $\langle \text{proof} \rangle$

lemma *pos-imp-selfadjoint*: $\langle a \geq 0 \implies \text{selfadjoint } a \rangle$
 $\langle \text{proof} \rangle$

lemma *abs-op-one-dim*: $\langle \text{abs-op } x = \text{one-dim-iso } (\text{abs } (\text{one-dim-iso } x :: \text{complex})) \rangle$
 $\langle \text{proof} \rangle$

lemma *pos-selfadjoint*: $\langle \text{selfadjoint } a \rangle$ **if** $\langle a \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *one-dim-loewner-order-strict*: $\langle A > B \iff \text{one-dim-iso } A > (\text{one-dim-iso } B :: \text{complex}) \rangle$
for $A B :: \langle 'a \Rightarrow_{CL} 'a :: \{\text{chilbert-space, one-dim}\} \rangle$
 $\langle \text{proof} \rangle$

lemma *one-dim-cblinfun-zero-le-one*: $\langle 0 < (1 :: 'a :: \text{one-dim} \Rightarrow_{CL} 'a) \rangle$
 $\langle \text{proof} \rangle$

lemma *one-dim-cblinfun-one-pos*: $\langle 0 \leq (1 :: 'a :: \text{one-dim} \Rightarrow_{CL} 'a) \rangle$
 $\langle \text{proof} \rangle$

lemma *Proj-pos[iff]*: $\langle \text{Proj } S \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *abs-op-Proj[simp]*: $\langle \text{abs-op } (\text{Proj } S) = \text{Proj } S \rangle$
 $\langle \text{proof} \rangle$

lemma *diagonal-operator-pos*:
assumes $\langle \lambda x. f x \geq 0 \rangle$
shows $\langle \text{diagonal-operator } f \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *abs-op-diagonal-operator*:
 $\langle \text{abs-op } (\text{diagonal-operator } f) = \text{diagonal-operator } (\lambda x. \text{abs } (f x)) \rangle$
 $\langle \text{proof} \rangle$

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::⟨{*chilbert-space, not-singleton*}⟩ *chilbert2ell2* = ⟨*some-chilbert-basis*
 :: 'a set
 ⟨*proof*⟩

definition *ell2-to-hilbert* **where** ⟨*ell2-to-hilbert* = *cblinfun-extension* (*range ket*) (*Rep-chilbert2ell2*
 o *inv ket*)⟩

lemma *ell2-to-hilbert-ket*: ⟨*ell2-to-hilbert* *_V *ket* *x* = *Rep-chilbert2ell2* *x*⟩
 ⟨*proof*⟩

lemma *norm-ell2-to-hilbert*: ⟨*norm ell2-to-hilbert* = 1⟩
 ⟨*proof*⟩

lemma *unitary-ell2-to-hilbert[simp]*: ⟨*unitary ell2-to-hilbert*⟩
 ⟨*proof*⟩

lemma *ell2-to-hilbert-adj-ket*: ⟨*ell2-to-hilbert** *_V ψ = *ket* (*Abs-chilbert2ell2* ψ)⟩ **if** ⟨ $\psi \in$ *some-chilbert-basis*⟩
 ⟨*proof*⟩

definition ⟨*cr-chilbert2ell2-ell2* *x y* \longleftrightarrow *ell2-to-hilbert* *_V *x* = *y*⟩

lemma *bi-unique-cr-chilbert2ell2-ell2[transfer-rule]*: ⟨*bi-unique cr-chilbert2ell2-ell2*⟩
 ⟨*proof*⟩

lemma *bi-total-cr-chilbert2ell2-ell2[transfer-rule]*: ⟨*bi-total cr-chilbert2ell2-ell2*⟩
 ⟨*proof*⟩

named-theorems *c2l2l2*

lemma *c2l2l2-cinner[c2l2l2]*:
includes *lifting-syntax*
shows ⟨(*cr-chilbert2ell2-ell2* \implies *cr-chilbert2ell2-ell2* \implies (=)) *cinner cinner*⟩
 ⟨*proof*⟩

lemma *c2l2l2-norm[c2l2l2]*:
includes *lifting-syntax*
shows ⟨(*cr-chilbert2ell2-ell2* \implies (=)) *norm norm*⟩
 ⟨*proof*⟩

lemma *c2l2l2-scaleC[c2l2l2]*:

includes *lifting-syntax*
shows $\langle ((=) == => cr\text{-}chilbert2ell2\text{-}ell2 == => cr\text{-}chilbert2ell2\text{-}ell2) scaleC scaleC \rangle$
 $\langle proof \rangle$

lemma *c2l2l2-zero*[*c2l2l2*]:
includes *lifting-syntax*
shows $\langle cr\text{-}chilbert2ell2\text{-}ell2\ 0\ 0 \rangle$
 $\langle proof \rangle$

lemma *c2l2l2-is-ortho-set*[*c2l2l2*]:
includes *lifting-syntax*
shows $\langle (rel\text{-}set\ cr\text{-}chilbert2ell2\text{-}ell2 == => (=)) is\text{-}ortho\text{-}set\ (is\text{-}ortho\text{-}set :: 'a::\{chilbert\text{-}space, not\text{-}singleton\} set \Rightarrow bool) \rangle$
 $\langle proof \rangle$

lemma *c2l2l2-ccspan*[*c2l2l2*]:
includes *lifting-syntax*
shows $\langle (rel\text{-}set\ cr\text{-}chilbert2ell2\text{-}ell2 == => rel\text{-}ccsubspace\ cr\text{-}chilbert2ell2\text{-}ell2) ccspan\ ccspan \rangle$
 $\langle proof \rangle$

lemma *ell2-to-hilbert-adj-ell2-to-hilbert* [*simp*]: *ell2-to-hilbert** *_V *ell2-to-hilbert* *_V $x = x$
 $\langle proof \rangle$

lemma *ell2-to-hilbert-ell2-to-hilbert-adj* [*simp*]: *ell2-to-hilbert* *_V *ell2-to-hilbert** *_V $x = x$
 $\langle proof \rangle$

lemma *bi-total-rel-ccsubspace-cr-chilbert2ell2-ell2* [*transfer-rule*]:
 $\langle bi\text{-}total\ (rel\text{-}ccsubspace\ cr\text{-}chilbert2ell2\text{-}ell2) \rangle$
 $\langle proof \rangle$

lemma *c2l2l2-top*[*c2l2l2*]:
includes *lifting-syntax*
shows $\langle (rel\text{-}ccsubspace\ cr\text{-}chilbert2ell2\text{-}ell2) top\ top \rangle$
 $\langle proof \rangle$

lemma *c2l2l2-is-onb*[*c2l2l2*]:
includes *lifting-syntax*
shows $\langle (rel\text{-}set\ cr\text{-}chilbert2ell2\text{-}ell2 == => (=)) is\text{-}onb\ is\text{-}onb \rangle$
 $\langle proof \rangle$

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]*:

topspace cweak-operator-topology = *UNIV*

<proof>

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>

lemma *wot-weaker-than-sot*:

continuous-map cstrong-operator-topology cweak-operator-topology ($\lambda f. f$)

<proof>

lemma *cweak-operator-topology-weaker-than-euclidean*:

continuous-map euclidean cweak-operator-topology ($\lambda f. f$)

<proof>

lemma *cweak-operator-topology-cinner-continuous*:

continuous-map cweak-operator-topology euclidean ($\lambda f. cinner\ x\ (f\ *_V\ y)$)

<proof>

lemma *continuous-on-cweak-operator-topo-iff-coordinatewise*:

continuous-map T cweak-operator-topology f

$\longleftrightarrow (\forall x\ y. continuous-map\ T\ euclidean\ (\lambda z. cinner\ x\ (f\ z\ *_V\ y)))$

<proof>

typedef (**overloaded**) ('a,'b) *cblinfun-wot* = *<UNIV :: ('a::complex-normed-vector \Rightarrow_{CL} 'b::complex-inner) set>* *<proof>*

setup-lifting *type-definition-cblinfun-wot*

instantiation *cblinfun-wot* :: (complex-normed-vector, complex-inner) complex-vector **begin**

lift-definition *scaleC-cblinfun-wot* :: *<complex \Rightarrow ('a, 'b) cblinfun-wot \Rightarrow ('a, 'b) cblinfun-wot>*

is $\langle \text{scaleC} \rangle$ $\langle \text{proof} \rangle$
lift-definition $\text{uminus-cblinfun-wot} :: \langle ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** uminus
 $\langle \text{proof} \rangle$
lift-definition $\text{zero-cblinfun-wot} :: \langle ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** 0 $\langle \text{proof} \rangle$
lift-definition $\text{minus-cblinfun-wot} :: \langle ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** minus $\langle \text{proof} \rangle$
lift-definition $\text{plus-cblinfun-wot} :: \langle ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is** plus $\langle \text{proof} \rangle$
lift-definition $\text{scaleR-cblinfun-wot} :: \langle \text{real} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \Rightarrow ('a, 'b) \text{ cblinfun-wot} \rangle$ **is**
 scaleR $\langle \text{proof} \rangle$
instance
 $\langle \text{proof} \rangle$
end

instantiation $\text{cblinfun-wot} :: (\text{complex-normed-vector}, \text{complex-inner}) \text{ topological-space}$ **begin**
lift-definition $\text{open-cblinfun-wot} :: \langle ('a, 'b) \text{ cblinfun-wot set} \Rightarrow \text{bool} \rangle$ **is** $\langle \text{openin cweak-operator-topology} \rangle$
 $\langle \text{proof} \rangle$
instance
 $\langle \text{proof} \rangle$
end

lemma $\text{transfer-nhds-cweak-operator-topology}[\text{transfer-rule}]$:
includes lifting-syntax
shows $\langle (\text{cr-cblinfun-wot} ==> \text{rel-filter cr-cblinfun-wot}) (\text{nhdsin cweak-operator-topology}) \text{ nhds} \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{limitin-cweak-operator-topology}$:
 $\langle \text{limitin cweak-operator-topology } f \ l \ F \longleftrightarrow (\forall a \ b. ((\lambda i. a \cdot_C (f \ i \ *_V \ b)) \longrightarrow a \cdot_C (l \ *_V \ b)) \ F) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{filterlim-cweak-operator-topology}$: $\langle \text{filterlim } f \ (\text{nhdsin cweak-operator-topology } l) = \text{limitin cweak-operator-topology } f \ l \rangle$
 $\langle \text{proof} \rangle$

instance $\text{cblinfun-wot} :: (\text{complex-normed-vector}, \text{complex-inner}) \text{ t2-space}$
 $\langle \text{proof} \rangle$

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

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

lemma $\text{transfer-euclidean-cweak-operator-topology}[\text{transfer-rule}]$:
includes lifting-syntax
shows $\langle (\text{rel-topology cr-cblinfun-wot}) \text{ cweak-operator-topology euclidean} \rangle$

<proof>

lemma *openin-cweak-operator-topology*: $\langle \text{openin cweak-operator-topology } U \longleftrightarrow (\exists V. \text{open } V \wedge U = (\lambda a (x,y). \text{cinner } x (a *_V y)) - 'V) \rangle$
<proof>

lemma *cweak-operator-topology-plus-cont*: $\langle \text{LIM } (x,y) \text{nhdsin cweak-operator-topology } a \times_F \text{nhdsin cweak-operator-topology } b. x + y :> \text{nhdsin cweak-operator-topology } (a + b) \rangle$
<proof>

instance *cblinfun-wot* :: (complex-normed-vector, complex-inner) topological-group-add
<proof>

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 $b :: \langle 'b::\text{chilbert-space} \Rightarrow_{CL} 'c::\text{complex-inner} \rangle$
<proof>

lemma *continuous-map-scaleC-wot*: $\langle \text{continuous-map cweak-operator-topology cweak-operator-topology } (\text{scaleC } c :: ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{chilbert-space}) \Rightarrow -) \rangle$
<proof>

lemma *continuous-scaleC-wot*: $\langle \text{continuous-on } X (\text{scaleC } c :: (-::\text{complex-normed-vector}, -: \text{chilbert-space}) \text{cblinfun-wot} \Rightarrow -) \rangle$
<proof>

lemma *wot-closure-is-csubspace[simp]*:
fixes $A::('a::\text{complex-normed-vector}, 'b::\text{chilbert-space}) \text{cblinfun-wot set}$
assumes $\langle \text{csubspace } A \rangle$
shows $\langle \text{csubspace } (\text{closure } A) \rangle$
<proof>
include *lattice-syntax*
<proof>

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set cr-cblinfun-wot} ==> (=)) \text{csubspace csubspace} \rangle$
<proof>

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{rel-set cr-cblinfun-wot} ==> (=)) (\text{closedin cweak-operator-topology}) \text{closed} \rangle$
<proof>

lemma [*transfer-rule*]:

includes *lifting-syntax*
shows $\langle \text{rel-set cr-cblinfun-wot} \implies \text{rel-set cr-cblinfun-wot} \rangle$ (*Abstract-Topology.closure-of-cweak-operator-topology*) *closure*
 $\langle \text{proof} \rangle$

lemma *wot-closure-is-csubspace*[*simp*]:
fixes $A :: ('a :: \text{complex-normed-vector} \Rightarrow_{CL} 'b :: \text{hilbert-space}) \text{ set}$
assumes $\langle \text{csubspace } A \rangle$
shows $\langle \text{csubspace } (\text{cweak-operator-topology closure-of } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-closed-cweak-operator-topology*:
fixes $A :: \langle ('b :: \text{complex-normed-vector} \Rightarrow_{CL} 'c :: \text{complex-inner}) \text{ set} \rangle$
assumes $aA: \langle \bigwedge i. a \ i \in A \rangle$
assumes $\text{closed}: \langle \text{closedin cweak-operator-topology } A \rangle$
assumes $\text{subspace}: \langle \text{csubspace } A \rangle$
assumes $\text{has-sum}: \langle \bigwedge \varphi \ \psi. ((\lambda i. \varphi \cdot_C (a \ i \ *_V \ \psi))) \text{ has-sum } \varphi \cdot_C (b \ *_V \ \psi) \rangle I$
shows $\langle b \in A \rangle$
 $\langle \text{proof} \rangle$

lemma *limitin-adj-wot*:
assumes $\langle \text{limitin cweak-operator-topology } f \ l \ F \rangle$
shows $\langle \text{limitin cweak-operator-topology } (\lambda i. (f \ i)^*) \ (l^*) \ F \rangle$
 $\langle \text{proof} \rangle$

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

lemma *hermitian-limit-hermitian-wot*:
assumes $\langle F \neq \text{bot} \rangle$
assumes $\text{herm}: \langle \bigwedge i. (a \ i)^* = a \ i \rangle$
assumes $\text{lim}: \langle \text{limitin cweak-operator-topology } a \ A \ F \rangle$
shows $\langle A^* = A \rangle$
 $\langle \text{proof} \rangle$

lemma *wot-weaker-than-sot-openin*:
 $\langle \text{openin cweak-operator-topology } x \implies \text{openin cstrong-operator-topology } x \rangle$
 $\langle \text{proof} \rangle$

lemma *wot-weaker-than-sot-limitin*: $\langle \text{limitin cweak-operator-topology } a \ A \ F \rangle$ **if** $\langle \text{limitin cstrong-operator-topology } a \ A \ F \rangle$
 $\langle \text{proof} \rangle$

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 $\langle A^* = A \rangle$

<proof>

lemma *hermitian-sum-hermitian-sot*:

assumes *herm*: $\langle \bigwedge i. (a\ i)^* = a\ i \rangle$

assumes *sum*: $\langle \text{has-sum-in } \text{cstrong-operator-topology } a\ X\ A \rangle$

shows $\langle A^* = A \rangle$

<proof>

lemma *wot-is-norm-topology-findim[simp]*:

$\langle (\text{cweak-operator-topology} :: ('a::\{\text{cfinite-dim}, \text{chilbert-space}\} \Rightarrow_{CL} 'b::\{\text{cfinite-dim}, \text{chilbert-space}\})$
 $\text{topology}) = \text{euclidean} \rangle$

<proof>

lemma *sot-is-norm-topology-fin-dim[simp]*:

$\langle (\text{cstrong-operator-topology} :: ('a::\{\text{cfinite-dim}, \text{chilbert-space}\} \Rightarrow_{CL} 'b::\{\text{cfinite-dim}, \text{chilbert-space}\})$
 $\text{topology}) = \text{euclidean} \rangle$

<proof>

lemma *regular-space-wot*: $\langle \text{regular-space } \text{cweak-operator-topology} \rangle$

<proof>

instance *cblinfun-wot* :: $(\text{complex-normed-vector}, \text{complex-inner})\ \text{t3-space}$

<proof>

instantiation *cblinfun-wot* :: $(\text{chilbert-space}, \text{chilbert-space})\ \text{order } \text{begin}$

lift-definition *less-eq-cblinfun-wot* :: $\langle ('a, 'b)\ \text{cblinfun-wot} \Rightarrow ('a, 'b)\ \text{cblinfun-wot} \Rightarrow \text{bool} \rangle$ **is**
*less-eq**<proof>*

lift-definition *less-cblinfun-wot* :: $\langle ('a, 'b)\ \text{cblinfun-wot} \Rightarrow ('a, 'b)\ \text{cblinfun-wot} \Rightarrow \text{bool} \rangle$ **is**
*less**<proof>*

instance

<proof>

end

instance *cblinfun-wot* :: $(\text{chilbert-space}, \text{chilbert-space})\ \text{ordered-comm-monoid-add}$

<proof>

lemma *limitin-wot-add*:

assumes $\langle \text{limitin } \text{cweak-operator-topology } f\ a\ F \rangle$

assumes $\langle \text{limitin } \text{cweak-operator-topology } g\ b\ F \rangle$

shows $\langle \text{limitin } \text{cweak-operator-topology } (\lambda x. f\ x + g\ x)\ (a + b)\ F \rangle$

<proof>

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 $\text{bounded}: \langle \forall F \ x \ \text{in} \ F. \ f \ x \leq B \rangle$
assumes $\text{increasing}: \langle \text{increasing-filter} \ (\text{filtermap} \ f \ F) \rangle$
shows $\langle \exists L. \ \text{limitin} \ \text{cweak-operator-topology} \ f \ L \ F \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{summable-wot-boundedI}$:
fixes $f :: \langle 'b \Rightarrow ('a \Rightarrow_{CL} 'a::\text{chilbert-space}) \rangle$
assumes $\text{bounded}: \langle \bigwedge F. \ \text{finite} \ F \Longrightarrow F \subseteq X \Longrightarrow \text{sum} \ f \ F \leq B \rangle$
assumes $\text{pos}: \langle \bigwedge x. \ x \in X \Longrightarrow f \ x \geq 0 \rangle$
shows $\langle \text{summable-on-in} \ \text{cweak-operator-topology} \ f \ X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{summable-wot-boundedI}'$:
fixes $f :: \langle 'b \Rightarrow ('a::\text{chilbert-space}, 'a) \ \text{cblinfun-wot} \rangle$
assumes $\text{bounded}: \langle \bigwedge F. \ \text{finite} \ F \Longrightarrow F \subseteq X \Longrightarrow \text{sum} \ f \ F \leq B \rangle$
assumes $\text{pos}: \langle \bigwedge x. \ x \in X \Longrightarrow f \ x \geq 0 \rangle$
shows $\langle f \ \text{summable-on} \ X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{has-sum-mono-neutral-wot}$:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\langle \text{has-sum-in} \ \text{cweak-operator-topology} \ f \ A \ a \rangle$ **and** $\text{has-sum-in} \ \text{cweak-operator-topology} \ g \ B \ b$
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 $a \leq b$
 $\langle \text{proof} \rangle$

lemma has-sum-mono-wot :
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\text{has-sum-in} \ \text{cweak-operator-topology} \ f \ A \ x$ **and** $\text{has-sum-in} \ \text{cweak-operator-topology} \ g \ A \ y$
assumes $\langle \bigwedge x. \ x \in A \Longrightarrow f \ x \leq g \ x \rangle$
shows $x \leq y$
 $\langle \text{proof} \rangle$

lemma $\text{infsun-mono-neutral-wot}$:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space} \Rightarrow_{CL} 'b)$
assumes $\text{summable-on-in} \ \text{cweak-operator-topology} \ f \ A$ **and** $\text{summable-on-in} \ \text{cweak-operator-topology} \ g \ B$
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 $\langle \bigwedge x. x \in B-A \implies g x \geq 0 \rangle$
shows *infsun-in cweak-operator-topology* $f A \leq \text{infsun-in cweak-operator-topology } g B$
 $\langle \text{proof} \rangle$

lemma *has-sum-on-wot-transfer[transfer-rule]*:
includes *lifting-syntax*
shows $\langle ((=) \implies \text{cr-cblinfun-wot}) \implies (=) \implies \text{cr-cblinfun-wot} \implies (\longleftrightarrow) \rangle$
(has-sum-in cweak-operator-topology) HAS-SUM
 $\langle \text{proof} \rangle$

lemma *summable-on-wot-transfer[transfer-rule]*:
includes *lifting-syntax*
shows $\langle (((=) \implies \text{cr-cblinfun-wot}) \implies (=) \implies (\longleftrightarrow)) (\text{summable-on-in cweak-operator-topology}) \rangle$
(summable-on)
 $\langle \text{proof} \rangle$

lemma *Abs-cblinfun-wot-transfer[transfer-rule]*:
includes *lifting-syntax*
shows $\langle ((=) \implies \text{cr-cblinfun-wot}) \text{ id Abs-cblinfun-wot} \rangle$
 $\langle \text{proof} \rangle$

lemma *infsun-mono-neutral-wot'*:
fixes $f :: 'a \Rightarrow ('b::\text{hilbert-space}, 'b) \text{cblinfun-wot}$
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 $\text{infsun } f A \leq \text{infsun } g B$
 $\langle \text{proof} \rangle$

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 $\text{infsun } f M \geq 0$
 $\langle \text{proof} \rangle$

lemma *summable-on-Sigma-wotI*:
fixes $f :: 'a \times 'b \Rightarrow ('c::\text{hilbert-space}, 'c) \text{cblinfun-wot}$
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*: $\langle \bigwedge x \in A \implies (\lambda y. f(x, y)) \text{ summable-on } (B x) \rangle$
shows $\langle f \text{ summable-on Sigma } A B \rangle$
 $\langle \text{proof} \rangle$

lift-definition *compose-wot* :: $\langle ('b::\text{complex-inner}, 'c::\text{complex-inner}) \text{cblinfun-wot} \implies ('a::\text{complex-normed-vector}, 'b) \text{cblinfun-wot} \implies ('a, 'c) \text{cblinfun-wot} \rangle$ **is**
 $\text{cblinfun-compose} \langle \text{proof} \rangle$

lift-definition *adj-wot* :: $\langle ('a::\text{chilbert-space}, 'b::\text{complex-inner}) \text{cblinfun-wot} \Rightarrow ('b, 'a) \text{cblin-fun-wot} \rangle$ is *adj* \langle proof \rangle

lemma *infsum-wot-is-Sup*:

fixes $f :: \langle 'b \Rightarrow ('a \Rightarrow_{CL} 'a::\text{chilbert-space}) \rangle$
assumes *summable*: $\langle \text{summable-on-in cweak-operator-topology } f \ X \rangle$
— See also *summable-wot-boundedI* for proving this.
assumes *pos*: $\langle \bigwedge x. x \in X \implies f \ x \geq 0 \rangle$
defines $\langle S \equiv \text{infsum-in cweak-operator-topology } f \ X \rangle$
shows $\langle \text{is-Sup } ((\lambda F. \sum_{x \in F}. f \ x) \text{ ' } \{F. \text{finite } F \wedge F \subseteq X\}) \ S \rangle$
 \langle proof \rangle

lemma *has-sum-in-cweak-operator-topology-pointwise*:

$\langle \text{has-sum-in cweak-operator-topology } f \ X \ s \longleftrightarrow (\forall \psi \ \varphi. ((\lambda x. \psi \cdot_C f \ x \ \varphi) \text{has-sum } \psi \cdot_C \ s \ \varphi) \ X) \rangle$
 \langle proof \rangle

lemma *summable-wot-bdd-above*:

fixes $f :: \langle 'b \Rightarrow ('a \Rightarrow_{CL} 'a::\text{chilbert-space}) \rangle$
assumes *summable*: $\langle \text{summable-on-in cweak-operator-topology } f \ X \rangle$
— See also *summable-wot-boundedI* for proving this.
assumes *pos*: $\langle \bigwedge x. x \in X \implies f \ x \geq 0 \rangle$
shows $\langle \text{bdd-above } (\text{sum } f \text{ ' } \{F. \text{finite } F \wedge F \subseteq X\}) \rangle$
 \langle proof \rangle

lemma *summable-on-in-cweak-operator-topology-pointwise*:

assumes $\langle \text{summable-on-in cweak-operator-topology } f \ X \rangle$
shows $\langle (\lambda x. a \cdot_C f \ x \ b) \text{summable-on } X \rangle$
 \langle proof \rangle

lemma *infsum-in-cweak-operator-topology-pointwise*:

assumes $\langle \text{summable-on-in cweak-operator-topology } f \ X \rangle$
shows $\langle a \cdot_C (\text{infsum-in cweak-operator-topology } f \ X) \ b = (\sum_{\infty x \in X}. a \cdot_C f \ x \ b) \rangle$
 \langle proof \rangle

instance *cblinfun-wot* :: $(\text{complex-normed-vector}, \text{complex-inner}) \text{topological-ab-group-add}$

\langle proof \rangle

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 $\langle \text{has-sum-in cweak-operator-topology } (\lambda x. a \ o_{CL} \ f \ x) \ X \ (a \ o_{CL} \ s) \rangle$
 \langle proof \rangle

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>

lemma *summable-on-in-wot-compose-left:*

fixes $f :: \langle 'c \Rightarrow 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
assumes $\langle \text{summable-on-in cweak-operator-topology } f X \rangle$
shows $\langle \text{summable-on-in cweak-operator-topology } (\lambda x. a \ o_{CL} \ f \ x) \ X \rangle$
<proof>

lemma *summable-on-in-wot-compose-right:*

assumes $\langle \text{summable-on-in cweak-operator-topology } f X \rangle$
shows $\langle \text{summable-on-in cweak-operator-topology } (\lambda x. f \ x \ o_{CL} \ a) \ X \rangle$
<proof>

lemma *infsun-in-wot-compose-left:*

fixes $f :: \langle 'c \Rightarrow 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
assumes $\langle \text{summable-on-in cweak-operator-topology } f X \rangle$
shows $\langle \text{infsun-in cweak-operator-topology } (\lambda x. a \ o_{CL} \ f \ x) \ X = a \ o_{CL} \ (\text{infsun-in cweak-operator-topology } f \ X) \rangle$
<proof>

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 $\langle \text{infsun-in cweak-operator-topology } (\lambda x. f \ x \ o_{CL} \ a) \ X = (\text{infsun-in cweak-operator-topology } f \ X) \ o_{CL} \ a \rangle$
<proof>

lemma *infsun-wot-boundedI:*

fixes $f :: \langle 'b \Rightarrow ('a \Rightarrow_{CL} 'a::\text{hilbert-space}) \rangle$
assumes $\text{bounded: } \langle \bigwedge F. \text{finite } F \implies F \subseteq X \implies \text{sum } f \ F \leq B \rangle$
assumes $\text{pos: } \langle \bigwedge x. x \in X \implies f \ x \geq 0 \rangle$
shows $\langle \text{infsun-in cweak-operator-topology } f \ X \leq B \rangle$
<proof>

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
⟨ML⟩

7.2 Auxiliary lemmas
named-theorems unoverload-def

locale local-typedef = fixes  $S :: 'b$  set and  $s :: 's$  itself
  assumes Ex-type-definition-S:  $\exists (Rep :: 's \Rightarrow 'b) (Abs :: 'b \Rightarrow 's)$ . type-definition Rep Abs S
begin
definition Rep = fst (SOME (Rep :: 's  $\Rightarrow$  'b, Abs). type-definition Rep Abs S)
definition Abs = snd (SOME (Rep :: 's  $\Rightarrow$  'b, Abs). type-definition Rep Abs S)
lemma type-definition-S: type-definition Rep Abs S
  ⟨proof⟩
lemma rep-in-S[simp]:  $Rep\ x \in S$ 
  and rep-inverse[simp]:  $Abs\ (Rep\ x) = x$ 
  and Abs-inverse[simp]:  $y \in S \implies Rep\ (Abs\ y) = y$ 
  ⟨proof⟩
definition cr-S where  $cr-S \equiv \lambda s\ b.\ s = Rep\ b$ 
lemma Domainp-cr-S[transfer-domain-rule]:  $Domainp\ cr-S = (\lambda x.\ x \in S)$ 
  ⟨proof⟩
lemma right-total-cr-S[transfer-rule]: right-total cr-S
  ⟨proof⟩
lemma bi-unique-cr-S[transfer-rule]: bi-unique cr-S
  ⟨proof⟩
lemma left-unique-cr-S[transfer-rule]: left-unique cr-S
  ⟨proof⟩
lemma right-unique-cr-S[transfer-rule]: right-unique cr-S
  ⟨proof⟩
lemma cr-S-Rep[intro, simp]:  $cr-S\ (Rep\ a)\ a$  ⟨proof⟩
lemma cr-S-Abs[intro, simp]:  $a \in S \implies cr-S\ a\ (Abs\ a)$  ⟨proof⟩
lemma UNIV-transfer[transfer-rule]: ⟨rel-set cr-S S UNIV⟩
  ⟨proof⟩
end

lemma complete-space-as-set[simp]: ⟨complete (space-as-set V)⟩ for  $V :: \langle - :: cbanach\ ccspace \rangle$ 
  ⟨proof⟩

```

definition $\langle \text{transfer-ball-range } A P \longleftrightarrow (\forall f. \text{range } f \subseteq A \longrightarrow P f) \rangle$

lemma *transfer-ball-range-parametric*[transfer-rule]:

includes *lifting-syntax*

assumes [transfer-rule, simp]: $\langle \text{right-unique } T \rangle \langle \text{bi-total } T \rangle \langle \text{bi-unique } U \rangle$

shows $\langle \text{rel-set } U \text{ ===== } ((T \text{ ===== } U) \text{ ===== } (\longrightarrow)) \text{ ===== } (\longrightarrow) \rangle$ *transfer-ball-range*
transfer-ball-range
 $\langle \text{proof} \rangle$

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 $\langle \text{rel-set } U \text{ ===== } ((T \text{ ===== } U) \text{ ===== } (\longleftrightarrow)) \text{ ===== } (\longleftrightarrow) \rangle$ *transfer-ball-range*
transfer-ball-range
 $\langle \text{proof} \rangle$

definition $\langle \text{transfer-Times } A B = A \times B \rangle$

lemma *transfer-Times-parametricity*[transfer-rule]:

includes *lifting-syntax*

shows $\langle \text{rel-set } T \text{ ===== } \text{rel-set } U \text{ ===== } \text{rel-set } (\text{rel-prod } T U) \rangle$ *transfer-Times* *transfer-Times*
 $\langle \text{proof} \rangle$

lemma *csubspace-nonempty*: $\langle \text{csubspace } X \implies X \neq \{\} \rangle$

$\langle \text{proof} \rangle$

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 $\langle ((A \text{ ===== } B) \text{ ===== } \text{rel-set } B \text{ ===== } \text{rel-set } A \text{ ===== } \text{rel-set } A) \rangle$ *transfer-vimage-into*
transfer-vimage-into
 $\langle \text{proof} \rangle$

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 $\langle \bigwedge M B. \bigcap M \sqcap \text{principal } B = \text{transfer-bounded-filter-Inf } B M \rangle$
and $\langle \bigwedge F M. F \sqcap \text{principal } M = \text{transfer-inf-principal } F M \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

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

$\langle \text{proof} \rangle$

lemma *semigroup-ow-typeclass*[*simp, iff*]: $\langle \text{semigroup-ow } V (+) \rangle$
if $\langle \bigwedge x y. x \in V \implies y \in V \implies x + y \in V \rangle$ **for** $V :: \langle 'a :: \text{semigroup-add set} \rangle$
 $\langle \text{proof} \rangle$

lemma *class-semigroup-add-ud*[*unoverload-def*]: $\langle \text{class.semigroup-add} = \text{semigroup-ow } UNIV \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (\text{rel-set } A \text{ ====} \rangle (A \text{ ====} \rangle A \text{ ====} \rangle A) \text{ ====} \rangle (=) \rangle$
 $\langle \text{abel-semigroup-ow } \text{abel-semigroup-ow} \rangle$
 $\langle \text{proof} \rangle$

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** $V :: \langle 'a :: \text{ab-semigroup-add set} \rangle$
 $\langle \text{proof} \rangle$

lemma *class-ab-semigroup-add-ud*[*unoverload-def*]: $\langle \text{class.ab-semigroup-add} = \text{abel-semigroup-ow } UNIV \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (\text{rel-set } A \text{ ====} \rangle (A \text{ ====} \rangle A \text{ ====} \rangle A) \text{ ====} \rangle A \text{ ====} \rangle (=) \rangle$
 $\langle \text{comm-monoid-ow } \text{comm-monoid-ow} \rangle$
 $\langle \text{proof} \rangle$

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** $V :: \langle 'a :: \text{comm-monoid-add set} \rangle$
 $\langle \text{proof} \rangle$

lemma *class-comm-monoid-add-ud*[*unoverload-def*]: $\langle \text{class.comm-monoid-add} = \text{comm-monoid-ow } UNIV \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (\text{rel-set } A \text{ =====} \text{ rel-set } A \text{ =====} (=)) \text{ =====} (=) \rangle$
 topological-space-ow topological-space-ow
 $\langle \text{proof} \rangle$

lemma *class-topological-space-ud*[*unoverload-def*]: $\langle \text{class.topological-space} = \text{topological-space-ow UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *topological-space-ow-from-topology*[*simp*]: $\langle \text{topological-space-ow } (\text{topspace } T) (\text{openin } T) \rangle$
 $\langle \text{proof} \rangle$

7.8 sum

definition $\langle \text{sum-ow } z \text{ plus } f \text{ } S =$
 (if finite S then the-default z (Collect (fold-graph (plus o f) z S)) else z)
 for *U z 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 $\langle (T \text{ =====} (V \text{ =====} T \text{ =====} T) \text{ =====} (U \text{ =====} V) \text{ =====} \text{rel-set } U \text{ =====} T) \rangle$
 sum-ow sum-ow
 $\langle \text{proof} \rangle$

lemma (**in** *comm-monoid-set*) *comp-fun-commute-onI*: $\langle \text{Finite-Set.comp-fun-commute-on UNIV } ((*) \circ g) \rangle$
 $\langle \text{proof} \rangle$

lemma (**in** *comm-monoid-set*) *F-via-the-default*: $\langle F \text{ } g \text{ } A = \text{the-default def } (\text{Collect } (\text{fold-graph } ((*) \circ g) \mathbf{1} \text{ } A)) \rangle$
 if $\langle \text{finite } A \rangle$
 $\langle \text{proof} \rangle$

lemma *sum-ud*[*unoverload-def*]: $\langle \text{sum} = \text{sum-ow } 0 \text{ plus} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (\text{rel-set } A \text{ =====} \text{ rel-set } A \text{ =====} (=)) \text{ =====} (=) \rangle$
t2-space-ow t2-space-ow

$\langle \text{proof} \rangle$

lemma *class-t2-space-ud*[*unoverload-def*]: $\langle \text{class.t2-space} = \text{t2-space-ow UNIV} \rangle$

$\langle \text{proof} \rangle$

lemma *t2-space-ow-from-topology*[*simp, iff*]: $\langle \text{t2-space-ow (topspace } T) (\text{openin } T) \rangle$ **if** $\langle \text{Hausdorff-space } T \rangle$

$\langle \text{proof} \rangle$

7.9.1 *continuous-on*

definition *continuous-on-ow* **where** $\langle \text{continuous-on-ow } A B \text{ opnA opnB } s f$

$\longleftrightarrow (\forall U \subseteq B. \text{opnB } U \longrightarrow (\exists V \subseteq A. \text{opnA } V \wedge (V \cap s) = (f \text{ -' } U) \cap s)) \rangle$

for $f :: \langle 'a \Rightarrow 'b \rangle$

lemma *continuous-on-ud*[*unoverload-def*]: $\langle \text{continuous-on } s f \longleftrightarrow \text{continuous-on-ow UNIV UNIV open open } s f \rangle$

for $f :: \langle 'a::\text{topological-space} \Rightarrow 'b::\text{topological-space} \rangle$

$\langle \text{proof} \rangle$

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 $\langle (\text{rel-set } A \text{ =====} \text{ rel-set } B \text{ =====} \text{ rel-set } A \text{ =====} (\longleftrightarrow)) \text{ =====} (\text{rel-set } B \text{ =====} (\longleftrightarrow)) \text{ =====} \text{ rel-set } A \text{ =====} (A \text{ =====} B) \text{ =====} (\longleftrightarrow) \rangle$ *continuous-on-ow continuous-on-ow*

$\langle \text{proof} \rangle$

7.10 *scaleR*

locale *scaleR-ow* =

fixes U **and** *scaleR* :: $\langle \text{real} \Rightarrow 'a \Rightarrow 'a \rangle$

assumes *scaleR-closed*: $\langle \forall a \in U. \text{scaleR } r a \in U \rangle$

lemma *scaleR-ow-typeclass*[*simp*]: $\langle \text{scaleR-ow UNIV scaleR} \rangle$ **for** *scaleR*

$\langle \text{proof} \rangle$

lemma *scaleR-ow-parametric*[*transfer-rule*]:

includes *lifting-syntax*

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

shows $\langle \text{rel-set } A \implies ((=) \implies A \implies A) \implies (=) \rangle$
scaleR-ow scaleR-ow
 $\langle \text{proof} \rangle$

7.11 *scaleC*

locale *scaleC-ow* = *scaleR-ow* +
fixes *scaleC*
assumes *scaleC-closed*: $\langle \forall a \in U. \text{scaleC } c \ a \in U \rangle$
assumes $\langle \forall a \in U. \text{scaleR } r \ a = \text{scaleC } (\text{complex-of-real } r) \ a \rangle$

lemma *scaleC-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle \text{rel-set } A \implies ((=) \implies A \implies A) \implies ((=) \implies A \implies A) \implies (=) \rangle$
scaleC-ow scaleC-ow
 $\langle \text{proof} \rangle$

lemma *class-scaleC-ud*[*unoverload-def*]: $\langle \text{class.scaleC} = \text{scaleC-ow UNIV} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{rel-set } A \implies (A \implies A \implies A) \implies A \implies (A \implies A \implies A) \implies (A \implies A) \implies (=) \rangle$
ab-group-add-ow ab-group-add-ow
 $\langle \text{proof} \rangle$

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 $V :: \langle - :: \text{ab-group-add set} \rangle$
 $\langle \text{proof} \rangle$

lemma *class-ab-group-add-ud*[*unoverload-def*]: $\langle \text{class.ab-group-add} = \text{ab-group-add-ow UNIV} \rangle$
 $\langle \text{proof} \rangle$

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

$\langle \forall x \in U. \text{scale } a \ x \in U \rangle$

$\langle \forall x \in U. \forall y \in U. \text{scale } a \ (\text{plus } x \ y) = \text{plus} \ (\text{scale } a \ x) \ (\text{scale } a \ y) \rangle$

$\langle \forall x \in U. \text{scale} \ (a + b) \ x = \text{plus} \ (\text{scale } a \ x) \ (\text{scale } b \ x) \rangle$

$\langle \forall x \in U. \text{scale } a \ (\text{scale } b \ x) = \text{scale} \ (a * b) \ x \rangle$

$\langle \forall x \in U. \text{scale } 1 \ x = x \rangle$

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

$\langle \text{proof} \rangle$

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 $\langle (\text{rel-set } A \text{ =====} \rangle ((=) \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle ((=) \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle (A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle$

$A \text{ =====} \rangle (A \text{ =====} \rangle A \text{ =====} \rangle A) \text{ =====} \rangle (A \text{ =====} \rangle A) \text{ =====} \rangle (=) \rangle$

complex-vector-ow complex-vector-ow

$\langle \text{proof} \rangle$

lemma *class-complex-vector-ud*[*unoverload-def*]: $\langle \text{class.complex-vector} = \text{complex-vector-ow } UNIV \rangle$

$\langle \text{proof} \rangle$

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* :: $\langle \text{::complex-vector set} \rangle$

$\langle \text{proof} \rangle$

lemma *complex-vector-ow-typeclass*[*simp*]:

$\langle \text{complex-vector-ow } V \ (*_R) \ (*_C) \ (+) \ 0 \ (-) \ \text{uminus} \rangle$ **if** [*simp*]: $\langle \text{csubspace } V \rangle$

$\langle \text{proof} \rangle$

7.15 *open-uniformity*

locale *open-uniformity-ow* = *open open + uniformity uniformity*
for *A open uniformity +*
assumes *open-uniformity:*
 $\bigwedge U. U \subseteq A \implies \text{open } U \iff (\forall x \in U. \text{eventually } (\lambda(x', y). x' = x \longrightarrow y \in U) \text{ uniformity})$

lemma *open-uniformity-ow-parametric[transfer-rule]:*
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \implies (\text{rel-set } A \implies (=)) \implies \text{rel-filter } (\text{rel-prod } A \ A) \implies (=))$
 $\text{open-uniformity-ow open-uniformity-ow} \rangle$
 $\langle \text{proof} \rangle$

lemma *class-open-uniformity-ud[unoverload-def]:* $\langle \text{class.open-uniformity} = \text{open-uniformity-ow UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *open-uniformity-on-typeclass[simp]:*
fixes $V :: \langle \text{open-uniformity set} \rangle$
assumes $\langle \text{closed } V \rangle$
shows $\langle \text{open-uniformity-ow } V \text{ (openin (top-of-set } V)) \text{ (uniformity-on } V) \rangle$
 $\langle \text{proof} \rangle$

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]:* $\langle \text{class.uniformity-dist} = \text{uniformity-dist-ow UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *uniformity-dist-ow-parametric[transfer-rule]:*
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \implies (A \implies A \implies (=)) \implies \text{rel-filter } (\text{rel-prod } A \ A) \implies (=))$
 $\text{uniformity-dist-ow uniformity-dist-ow} \rangle$
 $\langle \text{proof} \rangle$

lemma *uniformity-dist-on-typeclass[simp]:* $\langle \text{uniformity-dist-ow } V \text{ dist (uniformity-on } V) \rangle$ **for** V
 $:: \langle \text{uniformity-dist set} \rangle$
 $\langle \text{proof} \rangle$

7.17 *sgn*

locale *sgn-ow* =
fixes U **and** *sgn* $:: \langle 'a \Rightarrow 'a \rangle$
assumes *sgn-closed:* $\langle \forall a \in U. \text{sgn } a \in U \rangle$

lemma *sgn-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \implies (A \implies A) \implies (=)) \implies (=) \rangle$
sgn-ow sgn-ow
 $\langle \text{proof} \rangle$

7.18 *sgn-div-norm*

locale *sgn-div-norm-ow* = *scaleR-ow U scaleR + norm norm + sgn-ow U sgn for U sgn norm scaleR +*
assumes $\forall x \in U. \text{sgn } x = \text{scaleR } (\text{inverse } (\text{norm } x)) \ x$

lemma *class-sgn-div-norm-ud*[*unoverload-def*]: $\langle \text{class.sgn-div-norm} = \text{sgn-div-norm-ow UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *sgn-div-norm-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (\text{rel-set } A \implies (A \implies A) \implies (A \implies (=)) \implies ((=) \implies A \implies A) \implies (=)) \implies (=) \rangle$
sgn-div-norm-ow sgn-div-norm-ow
 $\langle \text{proof} \rangle$

lemma *sgn-div-norm-on-typeclass*[*simp*]:
fixes $V :: \langle \text{sgn-div-norm set} \rangle$
assumes $\langle \bigwedge v \ r. v \in V \implies \text{scaleR } r \ v \in V \rangle$
shows $\langle \text{sgn-div-norm-ow } V \ \text{sgn } \text{norm } (*_R) \rangle$
 $\langle \text{proof} \rangle$

7.19 *dist-norm*

locale *dist-norm-ow* = *dist dist + norm norm + minus-ow U minus for U minus dist norm +*
assumes *dist-norm*: $\forall x \in U. \forall y \in U. \text{dist } x \ y = \text{norm } (\text{minus } x \ y)$

lemma *dist-norm-ud*[*unoverload-def*]: $\langle \text{class.dist-norm} = \text{dist-norm-ow UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *dist-norm-ow-parametric*[*transfer-rule*]:
includes *lifting-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
 $\langle \text{proof} \rangle$

lemma *dist-norm-ow-typeclass*[*simp*]:
fixes $A :: \langle \text{dist-norm set} \rangle$
assumes $\langle \bigwedge a \ b. \llbracket a \in A; b \in A \rrbracket \implies a - b \in A \rangle$

shows $\langle \text{dist-norm-ow } A \text{ } (-) \text{ dist norm} \rangle$
 $\langle \text{proof} \rangle$

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*]: $\langle \text{class.complex-inner} = \text{complex-inner-ow } UNIV \rangle$
 $\langle \text{proof} \rangle$

lemma *complex-inner-ow-parametricity*[*transfer-rule*]:

includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } T \rangle$
shows $\langle (\text{rel-set } T \text{ } \text{====} \rangle ((=) \text{====} \rangle T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle ((=) \text{====} \rangle T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle$
 $(T \text{ } \text{====} \rangle T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle T$
 $\text{====} \rangle (T \text{ } \text{====} \rangle T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle (T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle (T \text{ } \text{====} \rangle T \text{ } \text{====} \rangle$
 $(=)) \text{ } \text{====} \rangle (T \text{ } \text{====} \rangle (=))$
 $\text{====} \rangle (T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle \text{rel-filter } (\text{rel-prod } T \ T) \text{ } \text{====} \rangle (\text{rel-set } T \text{ } \text{====} \rangle (=))$
 $\text{====} \rangle (T \text{ } \text{====} \rangle T \text{ } \text{====} \rangle (=)) \text{ } \text{====} \rangle (=)) \text{ } \text{complex-inner-ow } \text{complex-inner-ow}$
 $\langle \text{proof} \rangle$

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 $\langle \text{complex-inner-ow } V \text{ } (*_R) \text{ } (*_C) \text{ } (+) \text{ } 0 \text{ } (-) \text{ } \text{uminus} \text{ } \text{dist norm} \text{ } \text{sgn} \text{ } (\text{uniformity-on } V)$
 $(\text{openin } (\text{top-of-set } V)) \text{ } (\cdot_C) \rangle$
 $\langle \text{proof} \rangle$

7.21 is-ortho-set

definition *is-ortho-set-ow* **where** $\langle \text{is-ortho-set-ow } \text{zero } \text{cinner } S \iff$
 $((\forall x \in S. \forall y \in S. x \neq y \implies \text{cinner } x \ y = 0) \wedge \text{zero} \notin S) \rangle$
for *zero* *cinner*

lemma *is-ortho-set-ow-parametric*[*transfer-rule*]:

includes *lifting-syntax*
assumes [*transfer-rule*]: $\langle \text{bi-unique } A \rangle$
shows $\langle (A \text{ } \text{====} \rangle (A \text{ } \text{====} \rangle A \text{ } \text{====} \rangle (=)) \text{ } \text{====} \rangle \text{rel-set } A \text{ } \text{====} \rangle (=)$

is-ortho-set-ow is-ortho-set-ow
 ⟨proof⟩

lemma *is-ortho-set-ud*[unoverload-def]: ⟨*is-ortho-set = is-ortho-set-ow 0 cinner*⟩
 ⟨proof⟩

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]: ⟨*bi-unique A*⟩
shows ⟨(*rel-set A* \implies (*A* \implies *A* \implies (=)) \implies *rel-filter (rel-prod A A)* \implies (*rel-set A* \implies (=)) \implies (=))
metric-space-ow metric-space-ow⟩
 ⟨proof⟩

lemma *class-metric-space-ud*[unoverload-def]: ⟨*class.metric-space = metric-space-ow UNIV*⟩
 ⟨proof⟩

lemma *metric-space-ow-typeclass*[simp]:

fixes *V :: <-::metric-space set>*
assumes ⟨*closed V*⟩
shows ⟨*metric-space-ow V dist (uniformity-on V) (openin (top-of-set V))*⟩
 ⟨proof⟩

7.23 nhds

definition *nhds-ow* **where** ⟨*nhds-ow U open a = (INF S∈{S. S ⊆ U ∧ open S ∧ a ∈ S}. principal S) □ principal U*⟩
for *U open*

lemma *nhds-ow-parametric*[transfer-rule]:

includes *lifting-syntax*
assumes [transfer-rule]: ⟨*bi-unique A*⟩
shows ⟨(*rel-set A* \implies (*rel-set A* \implies (=)) \implies *A* \implies *rel-filter A*)
nhds-ow nhds-ow⟩
 ⟨proof⟩

lemma *topological-space-nhds-ud*[unoverload-def]: ⟨*topological-space.nhds = nhds-ow UNIV*⟩
 ⟨proof⟩

lemma *nhds-ud*[unoverload-def]: ⟨*nhds = nhds-ow UNIV open*⟩
 ⟨proof⟩

lemma *nhds-ow-topology[simp]*: $\langle \text{nhds-ow } (\text{topspace } T) (\text{openin } T) x = \text{nhdsin } T x \rangle$ **if** $\langle x \in \text{topspace } T \rangle$
 $\langle \text{proof} \rangle$

7.24 *at-within*

definition $\langle \text{at-within-ow } U \text{ open } a \text{ } s = \text{nhds-ow } U \text{ open } a \sqcap \text{principal } (s - \{a\}) \rangle$
for $U \text{ open } a \text{ } s$

lemma *at-within-ow-parametric[transfer-rule]*:
includes *lifting-syntax*
assumes $[\text{transfer-rule}]$: $\langle \text{bi-unique } T \rangle$
shows $\langle ((\text{rel-set } T) \text{====} \Rightarrow (\text{rel-set } T) \text{====} \Rightarrow (=)) \text{====} \Rightarrow T \text{====} \Rightarrow \text{rel-set } T \text{====} \Rightarrow \text{rel-filter } T \rangle$
 $\langle \text{at-within-ow at-within-ow} \rangle$
 $\langle \text{proof} \rangle$

lemma *at-within-ud[unoverload-def]*: $\langle \text{at-within} = \text{at-within-ow UNIV open} \rangle$
 $\langle \text{proof} \rangle$

lemma *at-within-ow-topology*:
 $\langle \text{at-within-ow } (\text{topspace } T) (\text{openin } T) a \text{ } S = \text{nhdsin } T a \sqcap \text{principal } (S - \{a\}) \rangle$
if $\langle a \in \text{topspace } T \rangle$
 $\langle \text{proof} \rangle$

7.25 *(has-sum)*

definition $\langle \text{has-sum-ow } U \text{ plus zero open } f \text{ } A \text{ } x =$
 $\text{filterlim } (\text{sum-ow zero plus } f) (\text{nhds-ow } U (\lambda S. \text{open } S) x)$
 $(\text{finite-subsets-at-top } A) \rangle$
for $U \text{ plus zero open } f \text{ } A \text{ } x$

lemma *has-sum-ow-parametric[transfer-rule]*:
includes *lifting-syntax*
assumes $[\text{transfer-rule}]$: $\langle \text{bi-unique } T \rangle \langle \text{bi-unique } U \rangle$
shows $\langle (\text{rel-set } T \text{====} \Rightarrow (V \text{====} \Rightarrow T \text{====} \Rightarrow T) \text{====} \Rightarrow T \text{====} \Rightarrow (\text{rel-set } T \text{====} \Rightarrow (=))$
 $\text{====} \Rightarrow (U \text{====} \Rightarrow V) \text{====} \Rightarrow \text{rel-set } U \text{====} \Rightarrow T \text{====} \Rightarrow (=) \rangle$
 $\langle \text{has-sum-ow has-sum-ow} \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-ud[unoverload-def]*: $\langle \text{HAS-SUM} = \text{has-sum-ow UNIV plus } (0::'a::\{\text{comm-monoid-add, topological-space}\}) \text{ open} \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-ow-topology*:
assumes $\langle l \in \text{topspace } T \rangle$
assumes $\langle 0 \in \text{topspace } T \rangle$
assumes $\langle \bigwedge x y. x \in \text{topspace } T \implies y \in \text{topspace } T \implies x + y \in \text{topspace } T \rangle$

shows $\langle \text{has-sum-ow } (\text{topspace } T) (+) 0 (\text{openin } T) f S l \longleftrightarrow \text{has-sum-in } T f S l \rangle$
 $\langle \text{proof} \rangle$

7.26 *filterlim*

7.27 *convergent*

definition *convergent-ow where*

$\langle \text{convergent-ow } U \text{ open } X \longleftrightarrow (\exists L \in U. \text{filterlim } X (\text{nhds-ow } U \text{ open } L) \text{ sequentially}) \rangle$

for $U \text{ open}$

lemma *convergent-ow-parametric[transfer-rule]:*

includes *lifting-syntax*

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

shows $\langle (\text{rel-set } T \text{ ====} \rangle (\text{rel-set } T \text{ ====} \rangle (=) \text{ ====} \rangle ((=) \text{ ====} \rangle T) \text{ ====} \rangle (\longleftrightarrow) \rangle$
convergent-ow convergent-ow

$\langle \text{proof} \rangle$

lemma *convergent-ud[unoverload-def]:* $\langle \text{convergent} = \text{convergent-ow } UNIV \text{ open} \rangle$

$\langle \text{proof} \rangle$

lemma *topological-space-convergent-ud[unoverload-def]:* $\langle \text{topological-space.convergent} = \text{convergent-ow } UNIV \rangle$

$\langle \text{proof} \rangle$

lemma *convergent-ow-topology[simp]:*

$\langle \text{convergent-ow } (\text{topspace } T) (\text{openin } T) f \longleftrightarrow (\exists l. \text{limitin } T f l \text{ sequentially}) \rangle$

$\langle \text{proof} \rangle$

lemma *convergent-ow-typeclass[simp]:*

$\langle \text{convergent-ow } V (\text{openin } (\text{top-of-set } V)) f \longleftrightarrow (\exists l. \text{limitin } (\text{top-of-set } V) f l \text{ sequentially}) \rangle$

$\langle \text{proof} \rangle$

7.28 *uniform-space.cauchy-filter*

lemma *cauchy-filter-parametric[transfer-rule]:*

includes *lifting-syntax*

assumes $[transfer\text{-rule}]: \text{bi-unique } T$

shows $(\text{rel-filter } (\text{rel-prod } T T) \text{ ====} \rangle \text{rel-filter } T \text{ ====} \rangle (=) \rangle$
uniform-space.cauchy-filter

uniform-space.cauchy-filter

$\langle \text{proof} \rangle$

7.29 *uniform-space.Cauchy*

lemma *uniform-space-Cauchy-parametric[transfer-rule]:*

includes *lifting-syntax*

assumes $[transfer\text{-rule}]: \text{bi-unique } T$

shows $(\text{rel-filter } (\text{rel-prod } T T) \text{ ====} \rangle ((=) \text{ ====} \rangle T) \text{ ====} \rangle (=) \rangle$
uniform-space.Cauchy

uniform-space.Cauchy
 ⟨*proof*⟩

7.30 complete-space

locale *complete-space-ow* = *metric-space-ow* *U* *dist* *uniformity* *open*
for *U* *dist* *uniformity* *open* +
assumes ⟨*range* *X* ⊆ *U* → *uniform-space.Cauchy* *uniformity* *X* → *convergent-ow* *U* *open* *X*⟩

lemma *class-complete-space-ud*[*unoverload-def*]: ⟨*class.complete-space* = *complete-space-ow* *UNIV*⟩
 ⟨*proof*⟩

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*
 ⟨*proof*⟩

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*⟩

7.31 hilbert-space

locale *hilbert-space-ow* = *complex-inner-ow* + *complete-space-ow*

lemma *hilbert-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* ==> (=)) ==> (=)⟩
hilbert-space-ow *hilbert-space-ow*
 ⟨*proof*⟩

lemma *hilbert-space-on-typeclass*[*simp*]:
fixes *V* :: ⟨-::*complex-inner* *set*⟩
assumes ⟨*complete* *V*⟩ ⟨*csubspace* *V*⟩
shows ⟨*hilbert-space-ow* *V* (**_R*) (**_C*) (+) 0 (-) *uminus* *dist* *norm* *sgn*
 (*uniformity-on* *V*) (*openin* (*top-of-set* *V*)) (*•_C*)⟩
 ⟨*proof*⟩

lemma *class-chilbert-space-ud*[*unoverload-def*]:
 ‹*class.chilbert-space* = *chilbert-space-ow UNIV*›
 ‹*proof*›

7.32 (hull)

definition ‹*hull-ow* $A S s = ((\lambda x. S x \wedge x \subseteq A) \text{ hull } s) \cap A$ ›

lemma *hull-ow-nondegenerate*: ‹*hull-ow* $A S s = ((\lambda x. S x \wedge x \subseteq A) \text{ hull } s)$ › **if** ‹ $x \subseteq A$ › **and**
 ‹ $s \subseteq x$ › **and** ‹ $S x$ ›
 ‹*proof*›

definition ‹*transfer-bounded-Inf* $B M = \text{Inf } M \sqcap B$ ›

lemma *transfer-bounded-Inf-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes ‹*bi-unique* T ›
shows ‹(*rel-set* $T \implies \text{rel-set } (\text{rel-set } T) \implies \text{rel-set } T$) *transfer-bounded-Inf transfer-bounded-Inf*›
 ‹*proof*›

lemma *hull-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique* T ›
shows ‹(*rel-set* $T \implies (\text{rel-set } T \implies (=)) \implies \text{rel-set } T \implies \text{rel-set } T$)
hull-ow hull-ow›
 ‹*proof*›

lemma *hull-ow-ud*[*unoverload-def*]: ‹(*hull*) = *hull-ow UNIV*›
 ‹*proof*›

7.33 csubspace

definition
 ‹*subspace-ow plus zero scale* $S = (\text{zero} \in S \wedge (\forall x \in S. \forall y \in S. \text{plus } x y \in S) \wedge (\forall c. \forall x \in S. \text{scale } c x \in S))$ ›
for *plus zero scale* S

lemma *subspace-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique* T ›
shows ‹(($T \implies T \implies T$) $\implies T \implies ((=) \implies T \implies T) \implies \text{rel-set } T \implies (=)$)
subspace-ow subspace-ow›
 ‹*proof*›

lemma *module-subspace-ud*[*unoverload-def*]: ‹*module.subspace* = *subspace-ow plus 0*›
 ‹*proof*›

lemma *csubspace-ud*[*unoverload-def*]: $\langle \text{csubspace} = \text{subspace-ow } (+) \ 0 \ (*_C) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{span-ow } U \text{ plus } 0 \text{ scale } C \ B = \text{cspan } B \rangle$

$\langle \text{proof} \rangle$

lemma (**in** *Modules.module*) *span-ud*[*unoverload-def*]: $\langle \text{span} = \text{span-ow } UNIV \text{ plus } 0 \text{ scale} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle (\text{rel-set } T \text{ } \text{====} \rangle (T \text{ } \text{====} \rangle T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle T \text{ } \text{====} \rangle ((=) \text{ } \text{====} \rangle T \text{ } \text{====} \rangle T) \text{ } \text{====} \rangle \text{rel-set } T \text{ } \text{====} \rangle \text{rel-set } T \rangle$
span-ow span-ow

$\langle \text{proof} \rangle$

7.34.1 (*islimpt*)

definition $\langle \text{islimpt-ow } U \text{ open } x \ S \longleftrightarrow (\forall T \subseteq U. x \in T \longrightarrow \text{open } T \longrightarrow (\exists y \in S. y \in T \wedge y \neq x)) \rangle$
for *open*

lemma *islimpt-ow-parametric*[*transfer-rule*]:

includes *lifting-syntax*

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

shows $\langle (\text{rel-set } T \text{ } \text{====} \rangle (\text{rel-set } T \text{ } \text{====} \rangle (=) \text{ } \text{====} \rangle T \text{ } \text{====} \rangle \text{rel-set } T \text{ } \text{====} \rangle (\longleftrightarrow)) \rangle$
islimpt-ow islimpt-ow

$\langle \text{proof} \rangle$

definition $\langle \text{islimptin } T \ x \ S \longleftrightarrow x \in \text{topspace } T \wedge (\forall V. x \in V \longrightarrow \text{openin } T \ V \longrightarrow (\exists y \in S. y \in V \wedge y \neq x)) \rangle$

lemma *islimpt-ow-from-topology*: $\langle \text{islimpt-ow } (\text{topspace } T) \ (\text{openin } T) \ x \ S \longleftrightarrow \text{islimptin } T \ x \ S \vee x \notin \text{topspace } T \rangle$

$\langle \text{proof} \rangle$

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]*:

$\langle \text{closure-ow } X \text{ (openin (top-of-set } X)) S = (X \cap \text{closure } (X \cap S)) \cup S \rangle$
 $\langle \text{proof} \rangle$

lemma *closure-ow-parametric[transfer-rule]*:

includes *lifting-syntax*

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

shows $\langle (\text{rel-set } T \implies (\text{rel-set } T \implies (=)) \implies \text{rel-set } T \implies \text{rel-set } T) \text{ closure-ow} \rangle$

$\langle \text{proof} \rangle$

lemma *closure-ow-from-topology*: $\langle \text{closure-ow (topspace } T) \text{ (openin } T) S = T \text{ closure-of } S \rangle$ **if** $\langle S \subseteq \text{topspace } T \rangle$

$\langle \text{proof} \rangle$

lemma *closure-ud[unoverload-def]*: $\langle \text{closure} = \text{closure-ow UNIV open} \rangle$

$\langle \text{proof} \rangle$

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 $\langle f \text{ 'topspace } T \subseteq \text{topspace } U \rangle$

$\langle \text{proof} \rangle$

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 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 $\langle (\text{rel-set } A \implies$

$((=) \implies A \implies A) \implies$

$(A \implies A \implies A) \implies$

$A \implies$

$(A \implies (=)) \implies (\text{rel-set } A \implies (=)) \implies (A \implies A \implies (=)) \implies$

$\text{rel-set } A \implies (=) \rangle$

is-onb-ow is-onb-ow

$\langle \text{proof} \rangle$

lemma *is-onb-ud[unoverload-def]*:

⟨is-onb = is-onb-ow UNIV scaleC plus 0 norm open cinner⟩
 ⟨proof⟩

7.37 Transferring theorems

lemma *closure-of-eqI*:

fixes $f\ g :: \langle 'a \Rightarrow 'b \rangle$ **and** $T :: \langle 'a \text{ topology} \rangle$ **and** $U :: \langle 'b \text{ topology} \rangle$
assumes *hausdorff*: ⟨Hausdorff-space U⟩
assumes *f-eq-g*: ⟨ $\bigwedge x. x \in S \implies f\ x = g\ x$ ⟩
assumes *x*: ⟨ $x \in T \text{ closure-of } S$ ⟩
assumes *f*: ⟨continuous-map T U f⟩ **and** *g*: ⟨continuous-map T U g⟩
shows ⟨ $f\ x = g\ x$ ⟩

⟨proof⟩

lemma *orthonormal-subspace-basis-exists*:

fixes $S :: \langle 'a::\text{hilbert-space set} \rangle$
assumes ⟨is-ortho-set S⟩ **and** *norm*: ⟨ $\bigwedge x. x \in S \implies \text{norm } x = 1$ ⟩ **and** ⟨ $S \subseteq \text{space-as-set } V$ ⟩
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⟩

lemma *has-sum-in-comm-additive-general*:

fixes $f :: \langle 'a \Rightarrow 'b :: \text{comm-monoid-add} \rangle$
and $g :: \langle 'b \Rightarrow 'c :: \text{comm-monoid-add} \rangle$
assumes *T0[simp]*: ⟨ $0 \in \text{topspace } T$ ⟩ **and** *Tplus[simp]*: ⟨ $\bigwedge x\ y. x \in \text{topspace } T \implies y \in \text{topspace } T \implies x+y \in \text{topspace } T$ ⟩
assumes *Uplus[simp]*: ⟨ $\bigwedge x\ y. x \in \text{topspace } U \implies y \in \text{topspace } U \implies x+y \in \text{topspace } U$ ⟩
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*: ⟨filterlim g (nhdsin U (g l)) (atin T l)⟩
assumes *gadd*: ⟨ $\bigwedge x\ y. x \in \text{topspace } T \implies y \in \text{topspace } T \implies g\ (x+y) = g\ x + g\ y$ ⟩
assumes *sumf*: ⟨has-sum-in T f S l⟩
shows ⟨has-sum-in U (g o f) S (g l)⟩

⟨proof⟩

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 ⟨topspace T = UNIV⟩ **and** ⟨topspace U = UNIV⟩
assumes ⟨Modules.additive g⟩
assumes *gcont*: ⟨continuous-map T U g⟩
assumes *sumf*: ⟨has-sum-in T f S l⟩
shows ⟨has-sum-in U (g o f) S (g l)⟩

⟨proof⟩

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 $\text{some-onb-of-is-ortho-set}[\text{iff}]: \langle \text{is-ortho-set } (\text{some-onb-of } X) \rangle$
and $\text{some-onb-of-norm1}: \langle b \in \text{some-onb-of } X \implies \text{norm } b = 1 \rangle$
and $\text{some-onb-of-ccspan}[\text{simp}]: \langle \text{ccspan } (\text{some-onb-of } X) = X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{ccspace-as-whole-type}$:

fixes $X :: \langle 'a::\text{hilbert-space ccspace} \rangle$
assumes $\langle X \neq 0 \rangle$
shows $\langle \text{let } 'b::\text{type} = \text{some-onb-of } X \text{ in } \exists U::'b \text{ ell2} \Rightarrow_{CL} 'a. \text{isometry } U \wedge U *_S \top = X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{some-onb-of-0}[\text{simp}]: \langle \text{some-onb-of } (0 :: 'a::\text{hilbert-space ccspace}) = \{\} \rangle$

$\langle \text{proof} \rangle$

lemma $\text{some-onb-of-finite-dim}$:

fixes $S :: \langle 'a::\text{hilbert-space ccspace} \rangle$
assumes $\langle \text{finite-dim-ccspace } S \rangle$
shows $\langle \text{finite } (\text{some-onb-of } S) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{some-onb-of-in-space}[\text{iff}]:$

fixes $S :: \langle 'a::\text{hilbert-space ccspace} \rangle$
shows $\langle \text{some-onb-of } S \subseteq \text{space-as-set } S \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{sum-some-onb-of-butterfly}$:

fixes $S :: \langle 'a::\text{hilbert-space ccspace} \rangle$
assumes $\langle \text{finite-dim-ccspace } S \rangle$
shows $\langle (\sum x \in \text{some-onb-of } S. \text{butterfly } x \ x) = \text{Proj } S \rangle$
 $\langle \text{proof} \rangle$

lemma cdim-infinite-0 :

assumes $\langle \neg \text{finite-dim } S \rangle$
shows $\langle \text{cdim } S = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma some-onb-of-card :

fixes $S :: \langle 'a::\text{hilbert-space ccspace} \rangle$
shows $\langle \text{card} (\text{some-onb-of } S) = \text{cdim} (\text{space-as-set } S) \rangle$
 $\langle \text{proof} \rangle$

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* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \rangle \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{normal-op } A \longleftrightarrow A \circ_{CL} A^* = A^* \circ_{CL} A \rangle$

definition *eigenvalues* :: $\langle 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'a \rangle \Rightarrow \text{complex set} \rangle$ **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*: $\langle A *_S S \leq S \implies \text{invariant-subspace } S \ A \rangle$
 $\langle \text{proof} \rangle$

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*: $\langle A *_S S \leq S \implies A *_S (-S) \leq -S \implies \text{reducing-subspace } S \ A \rangle$
 $\langle \text{proof} \rangle$

lemma *reducing-subspace-ortho[simp]*: $\langle \text{reducing-subspace } (-S) \ A \longleftrightarrow \text{reducing-subspace } S \ A \rangle$
for $S :: \langle 'a::\text{hilbert-space ccspace} \rangle$
 $\langle \text{proof} \rangle$

lemma *invariant-subspace-bot[simp]*: $\langle \text{invariant-subspace } \perp \ A \rangle$
 $\langle \text{proof} \rangle$

lemma *invariant-subspace-top[simp]*: $\langle \text{invariant-subspace } \top \ A \rangle$
 $\langle \text{proof} \rangle$

lemma *reducing-subspace-bot[simp]*: $\langle \text{reducing-subspace } \perp \ A \rangle$
 $\langle \text{proof} \rangle$

lemma *reducing-subspace-top[simp]*: $\langle \text{reducing-subspace } \top A \rangle$
 $\langle \text{proof} \rangle$

lemma *kernel-uminus[simp]*: $\text{kernel } (-A) = \text{kernel } A$
for $a :: \text{complex}$ **and** $A :: (-,-) \text{ cblinfun}$
 $\langle \text{proof} \rangle$

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** $A :: (-,-) \text{ cblinfun}$
 $\langle \text{proof} \rangle$

lemma *eigenvalues-0[simp]*: $\langle \text{eigenvalues } (0 :: 'a::\{\text{not-singleton,complex-normed-vector}\}) \Rightarrow_{CL} 'a) = \{0\} \rangle$
 $\langle \text{proof} \rangle$

lemma *nonzero-ccsubspace-contains-unit-vector*:
assumes $\langle S \neq 0 \rangle$
shows $\langle \exists \psi. \psi \in \text{space-as-set } S \wedge \text{norm } \psi = 1 \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-eigenvector-ex*:
assumes $\langle x \in \text{eigenvalues } a \rangle$
shows $\langle \exists h. \text{norm } h = 1 \wedge a h = x *_C h \rangle$
 $\langle \text{proof} \rangle$

lemma *eigenvalue-norm-bound*:
assumes $\langle e \in \text{eigenvalues } a \rangle$
shows $\langle \text{norm } e \leq \text{norm } a \rangle$
 $\langle \text{proof} \rangle$

lemma *eigenvalue-selfadj-real*:
assumes $\langle e \in \text{eigenvalues } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle e \in \mathbb{R} \rangle$
 $\langle \text{proof} \rangle$

lemma *is-Sup-imp-ex-tendsto*:
fixes $X :: \langle 'a::\{\text{linorder-topology,first-countable-topology}\} \text{ set} \rangle$
assumes $\text{sup}: \langle \text{is-Sup } X l \rangle$
assumes $\langle X \neq \{\} \rangle$
shows $\langle \exists f. \text{range } f \subseteq X \wedge f \longrightarrow l \rangle$
 $\langle \text{proof} \rangle$

lemma *eigenvaluesI*:
assumes $\langle A *_V h = e *_C h \rangle$
assumes $\langle h \neq 0 \rangle$
shows $\langle e \in \text{eigenvalues } A \rangle$

$\langle \text{proof} \rangle$

lemma *tendsto-diff-const-left-rewrite*:

fixes $c\ d :: \langle 'a :: \{ \text{topological-group-add}, \text{ab-group-add} \} \rangle$

assumes $\langle ((\lambda x. f\ x) \longrightarrow c - d)\ F \rangle$

shows $\langle ((\lambda x. c - f\ x) \longrightarrow d)\ F \rangle$

$\langle \text{proof} \rangle$

lemma *not-not-singleton-no-eigenvalues*:

fixes $a :: \langle 'a :: \text{complex-normed-vector} \Rightarrow_{CL} 'a \rangle$

assumes $\langle \neg \text{class.not-singleton TYPE('a)} \rangle$

shows $\langle \text{eigenvalues } a = \{ \} \rangle$

$\langle \text{proof} \rangle$

lemma *cblinfun-cinner-eq0I*:

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

assumes $\langle \bigwedge h. h \cdot_C a\ h = 0 \rangle$

shows $\langle a = 0 \rangle$

$\langle \text{proof} \rangle$

lemma *normal-op-iff-adj-same-norms*:

— [2], Proposition II.2.16

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

shows $\langle \text{normal-op } a \longleftrightarrow (\forall h. \text{norm } (a\ h) = \text{norm } ((a^*)\ h)) \rangle$

$\langle \text{proof} \rangle$

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 $\langle \text{eigenspace } l\ a = \text{eigenspace } (cnj\ l)\ (a^*) \rangle$

$\langle \text{proof} \rangle$

lemma *normal-op-adj-eigenvalues*:

assumes $\langle \text{normal-op } a \rangle$

shows $\langle \text{eigenvalues } (a^*) = cnj\ \text{'eigenvalues } a \rangle$

$\langle \text{proof} \rangle$

lemma *invariant-subspace-iff-PAP*:

— [2], Proposition II.3.7 (b)

$\langle \text{invariant-subspace } S\ A \longleftrightarrow \text{Proj } S\ o_{CL}\ A\ o_{CL}\ \text{Proj } S = A\ o_{CL}\ \text{Proj } S \rangle$

$\langle \text{proof} \rangle$

lemma *reducing-iff-PA*:

— [2], Proposition II.3.7 (e)

$\langle \text{reducing-subspace } S\ A \longleftrightarrow \text{Proj } S\ o_{CL}\ A = A\ o_{CL}\ \text{Proj } S \rangle$

$\langle \text{proof} \rangle$

lemma *reducing-iff-also-adj-invariant*:

— [2], Proposition II.3.7 (g)
shows $\langle \text{reducing-subspace } S \ A \longleftrightarrow \text{invariant-subspace } S \ A \wedge \text{invariant-subspace } S \ (A^*) \rangle$
 $\langle \text{proof} \rangle$

lemma *eigenspace-is-reducing*:
— [2], Proposition II.5.6
assumes $\langle \text{normal-op } a \rangle$
shows $\langle \text{reducing-subspace } (\text{eigenspace } l \ a) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *invariant-subspace-Inf*:
assumes $\langle \bigwedge S. S \in M \implies \text{invariant-subspace } S \ a \rangle$
shows $\langle \text{invariant-subspace } (\bigcap M) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *invariant-subspace-INF*:
assumes $\langle \bigwedge x. x \in X \implies \text{invariant-subspace } (S \ x) \ a \rangle$
shows $\langle \text{invariant-subspace } (\bigcap x \in X. S \ x) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *invariant-subspace-Sup*:
assumes $\langle \bigwedge S. S \in M \implies \text{invariant-subspace } S \ a \rangle$
shows $\langle \text{invariant-subspace } (\bigcup M) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *invariant-subspace-SUP*:
assumes $\langle \bigwedge x. x \in X \implies \text{invariant-subspace } (S \ x) \ a \rangle$
shows $\langle \text{invariant-subspace } (\bigcup x \in X. S \ x) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *reducing-subspace-Inf*:
fixes $a :: \langle 'a :: \text{chilbert-space} \implies_{CL} 'a \rangle$
assumes $\langle \bigwedge S. S \in M \implies \text{reducing-subspace } S \ a \rangle$
shows $\langle \text{reducing-subspace } (\bigcap M) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *reducing-subspace-INF*:
fixes $a :: \langle 'a :: \text{chilbert-space} \implies_{CL} 'a \rangle$
assumes $\langle \bigwedge x. x \in X \implies \text{reducing-subspace } (S \ x) \ a \rangle$
shows $\langle \text{reducing-subspace } (\bigcap x \in X. S \ x) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *reducing-subspace-Sup*:
fixes $a :: \langle 'a :: \text{chilbert-space} \implies_{CL} 'a \rangle$
assumes $\langle \bigwedge S. S \in M \implies \text{reducing-subspace } S \ a \rangle$
shows $\langle \text{reducing-subspace } (\bigcup M) \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *reducing-subspace-SUP*:

fixes $a :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \bigwedge x. x \in X \implies \text{reducing-subspace } (S\ x)\ a \rangle$
shows $\langle \text{reducing-subspace } (\bigsqcup_{x \in X}. S\ x)\ a \rangle$
 $\langle \text{proof} \rangle$

lemma *selfadjoint-imp-normal*: $\langle \text{normal-op } a \rangle$ **if** $\langle \text{selfadjoint } a \rangle$
 $\langle \text{proof} \rangle$

lemma *eigenspaces-orthogonal*:

— [2], Proposition II.5.7

assumes $\langle e \neq f \rangle$

assumes $\langle \text{normal-op } a \rangle$

shows $\langle \text{orthogonal-spaces } (\text{eigenspace } e\ a)\ (\text{eigenspace } f\ a) \rangle$

$\langle \text{proof} \rangle$

definition *largest-eigenvalue* :: $\langle ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'a) \Rightarrow \text{complex} \rangle$ **where**

$\langle \text{largest-eigenvalue } a =$

$(\text{if } \exists x. x \in \text{eigenvalues } a \wedge (\forall y \in \text{eigenvalues } a. \text{cmod } x \geq \text{cmod } y) \text{ then}$

$\text{SOME } x. x \in \text{eigenvalues } a \wedge (\forall y \in \text{eigenvalues } a. \text{cmod } x \geq \text{cmod } y) \text{ else } 0) \rangle$

lemma *largest-eigenvalue-0-aux*:

$\langle \text{largest-eigenvalue } (0 :: 'a::\{\text{not-singleton, complex-normed-vector}\} \Rightarrow_{CL} 'a) = 0 \rangle$

$\langle \text{proof} \rangle$

lemma *largest-eigenvalue-0[simp]*:

$\langle \text{largest-eigenvalue } (0 :: 'a::\text{complex-normed-vector} \Rightarrow_{CL} 'a) = 0 \rangle$

$\langle \text{proof} \rangle$

hide-fact *largest-eigenvalue-0-aux*

lemma *eigenvalues-nonneg*:

assumes $\langle a \geq 0 \rangle$ **and** $\langle v \in \text{eigenvalues } a \rangle$

shows $\langle v \geq 0 \rangle$

$\langle \text{proof} \rangle$

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]*: $\langle \text{finite-rank } 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-scaleC[simp]*: $\langle \text{finite-rank } (c *_{\mathbb{C}} a) \rangle$ **if** $\langle \text{finite-rank } a \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-scaleR[simp]*: $\langle \text{finite-rank } (c *_{\mathbb{R}} a) \rangle$ **if** $\langle \text{finite-rank } a \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-uminus[simp]*: $\langle \text{finite-rank } (-a) = \text{finite-rank } a \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-plus[simp]*: $\langle \text{finite-rank } (a + b) \rangle$ **if** $\langle \text{finite-rank } a \rangle$ **and** $\langle \text{finite-rank } b \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-minus[simp]*: $\langle \text{finite-rank } (a - b) \rangle$ **if** $\langle \text{finite-rank } a \rangle$ **and** $\langle \text{finite-rank } b \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-butterfly[simp]*: $\langle \text{finite-rank } (\text{butterfly } x \ y) \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-sum-butterfly*:
fixes $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{\text{CL}} 'b::\text{chilbert-space} \rangle$
assumes $\langle \text{finite-rank } a \rangle$
shows $\langle \exists x \ y \ (n::\text{nat}). a = (\sum i < n. \text{butterfly } (x \ i) \ (y \ i)) \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-sum*: $\langle \text{finite-rank } (\sum x \in F. f \ x) \rangle$ **if** $\langle \bigwedge x. x \in F \implies \text{finite-rank } (f \ x) \rangle$
 $\langle \text{proof} \rangle$

lemma *rank1-finite-rank*: $\langle \text{finite-rank } a \rangle$ **if** $\langle \text{rank1 } a \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-compose-left*:
assumes $\langle \text{finite-rank } B \rangle$
shows $\langle \text{finite-rank } (A \ o_{\text{CL}} \ B) \rangle$

<proof>

lemma *finite-rank-compose-right*:

assumes *<finite-rank A>*

shows *<finite-rank (A o_{CL} B)>*

<proof>

lemma *rank1-Proj-singleton[iff]*: *<rank1 (Proj (ccspan {x}))>*

<proof>

lemma *finite-rank-Proj-singleton[iff]*: *<finite-rank (Proj (ccspan {x}))>*

<proof>

lemma *finite-rank-Proj-finite-dim*:

fixes *S :: '<a::chilbert-space ccspace>*

assumes *<finite-dim-ccspace S>*

shows *<finite-rank (Proj S)>*

<proof>

lemma *finite-rank-Proj-finite*:

fixes *F :: '<a::chilbert-space set>*

assumes *<finite F>*

shows *<finite-rank (Proj (ccspan F))>*

<proof>

lemma *finite-rank-cfinite-dim[simp]*: *<finite-rank (a :: 'a :: {cfinite-dim, chilbert-space} \Rightarrow_{CL} 'b*
:: complex-normed-vector)>

<proof>

lemma *finite-rank-cspan-butterflies*:

<finite-rank a \longleftrightarrow a \in cspan (range (case-prod butterfly))>

for *a :: '<a::chilbert-space \Rightarrow_{CL} 'b::chilbert-space>*

<proof>

lemma *finite-rank-comp-left*: *<finite-rank (a o_{CL} b)>* **if** *<finite-rank a>*

for *a b :: '<::chilbert-space \Rightarrow_{CL} ::chilbert-space>*

<proof>

lemma *finite-rank-comp-right*: *<finite-rank (a o_{CL} b)>* **if** *<finite-rank b>*

for *a b :: '<::chilbert-space \Rightarrow_{CL} ::chilbert-space>*

<proof>

10.2 Compact operators

definition *compact-map* **where** *<compact-map f \longleftrightarrow clinear f \wedge compact (closure (f ' cball 0*
1))>

lemma *bounded-clinear f* **if** $\langle \text{compact-map } f \rangle$

— [2], Proposition II.4.2 (a)

thm *bounded-clinear-def*

$\langle \text{proof} \rangle$

lift-definition *compact-op* :: $\langle ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{complex-normed-vector}) \Rightarrow \text{bool} \rangle$ **is** *compact-map* $\langle \text{proof} \rangle$

lemma *compact-op-def2*: $\langle \text{compact-op } a \longleftrightarrow \text{compact } (\text{closure } (a \text{ ' cball } 0 \ 1)) \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-0[simp]*: $\langle \text{compact-op } 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-scaleC[simp]*: $\langle \text{compact-op } (c *_{\mathbb{C}} a) \rangle$ **if** $\langle \text{compact-op } a \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-scaleR[simp]*: $\langle \text{compact-op } (c *_{\mathbb{R}} a) \rangle$ **if** $\langle \text{compact-op } a \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-uminus[simp]*: $\langle \text{compact-op } (-a) = \text{compact-op } a \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-plus[simp]*: $\langle \text{compact-op } (a + b) \rangle$ **if** $\langle \text{compact-op } a \rangle$ **and** $\langle \text{compact-op } b \rangle$
 $\langle \text{proof} \rangle$

lemma *csubspace-compact-op*: $\langle \text{csubspace } (\text{Collect } \text{compact-op}) \rangle$
— [2], Proposition II.4.2 (b)
 $\langle \text{proof} \rangle$

lemma *compact-op-minus[simp]*: $\langle \text{compact-op } (a - b) \rangle$ **if** $\langle \text{compact-op } a \rangle$ **and** $\langle \text{compact-op } b \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-sgn[simp]*: $\langle \text{compact-op } (\text{sgn } a) = \text{compact-op } a \rangle$
 $\langle \text{proof} \rangle$

lemma *closed-compact-op*:

shows $\langle \text{closed } (\text{Collect } (\text{compact-op} :: ('a::\text{complex-normed-vector} \Rightarrow_{CL} 'b::\text{hilbert-space}) \Rightarrow \text{bool})) \rangle$

— [2], Proposition II.4.2 (b)

$\langle \text{proof} \rangle$

lemma *rank1-compact-op*: $\langle \text{compact-op } a \rangle$ **if** $\langle \text{rank1 } a \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-compact-op*: $\langle \text{compact-op } a \rangle$ **if** $\langle \text{finite-rank } a \rangle$
 $\langle \text{proof} \rangle$

lemma *bounded-products-sot-lim-imp-lim*:
— Implicit in the proof of [2], Proposition II.4.4 (c)
fixes $A :: \langle 'a :: \text{complex-normed-vector} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$
assumes $\text{lim-PA} : \langle \text{limit in cstrong-operator-topology } (\lambda x. P x \circ_{CL} A) A F \rangle$
and $\langle \text{compact-op } A \rangle$
and $P\text{-leq-B} : \langle \bigwedge x. \text{norm } (P x) \leq B \rangle$
shows $\langle ((\lambda x. P x \circ_{CL} A) \longrightarrow A) F \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-finite-rank*:
fixes $A :: \langle 'a :: \text{complex-normed-vector} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$
shows $\langle \text{compact-op } A \longleftrightarrow A \in \text{closure } (\text{Collect finite-rank}) \rangle$
— [2], Proposition II.4.4 (c)
 $\langle \text{proof} \rangle$

typedef (**overloaded**) $('a :: \text{hilbert-space}, 'b :: \text{complex-normed-vector}) \text{ compact-op} =$
 $\langle \text{Collect compact-op} :: ('a \Rightarrow_{CL} 'b) \text{ set} \rangle$
morphisms *from-compact-op Abs-compact-op*
 $\langle \text{proof} \rangle$
setup-lifting *type-definition-compact-op*

instantiation $\text{compact-op} :: (\text{hilbert-space}, \text{complex-normed-vector}) \text{ complex-normed-vector}$ **begin**

lift-definition $\text{scaleC-compact-op} :: \langle \text{complex} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** scaleC $\langle \text{proof} \rangle$

lift-definition $\text{uminus-compact-op} :: \langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** uminus $\langle \text{proof} \rangle$

lift-definition $\text{zero-compact-op} :: \langle ('a, 'b) \text{ compact-op} \rangle$ **is** 0 $\langle \text{proof} \rangle$

lift-definition $\text{minus-compact-op} :: \langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** minus $\langle \text{proof} \rangle$

lift-definition $\text{plus-compact-op} :: \langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** plus $\langle \text{proof} \rangle$

lift-definition $\text{sgn-compact-op} :: \langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** sgn $\langle \text{proof} \rangle$

lift-definition $\text{norm-compact-op} :: \langle ('a, 'b) \text{ compact-op} \Rightarrow \text{real} \rangle$ **is** norm $\langle \text{proof} \rangle$

lift-definition $\text{scaleR-compact-op} :: \langle \text{real} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \rangle$ **is** scaleR $\langle \text{proof} \rangle$

lift-definition $\text{dist-compact-op} :: \langle ('a, 'b) \text{ compact-op} \Rightarrow ('a, 'b) \text{ compact-op} \Rightarrow \text{real} \rangle$ **is** dist $\langle \text{proof} \rangle$

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 $\text{open-compact-op} :: ('a, 'b) \text{ compact-op set} \Rightarrow \text{bool}$

where [*code del*]: $\text{open-compact-op } S = (\forall x \in S. \forall_F (x', y) \text{ in uniformity. } x' = x \longrightarrow y \in S)$

instance

$\langle \text{proof} \rangle$

end

lemma *from-compact-op-plus*: $\langle \text{from-compact-op } (a + b) = \text{from-compact-op } a + \text{from-compact-op } b \rangle$
 $\langle \text{proof} \rangle$

lemma *from-compact-op-scaleC*: $\langle \text{from-compact-op } (c *_C a) = c *_C \text{from-compact-op } a \rangle$
 $\langle \text{proof} \rangle$

lemma *from-compact-op-norm[simp]*: $\langle \text{norm } (\text{from-compact-op } a) = \text{norm } a \rangle$
 $\langle \text{proof} \rangle$

lemma *compact-op-butterfly[simp]*: $\langle \text{compact-op } (\text{butterfly } x \ y) \rangle$
 $\langle \text{proof} \rangle$

lift-definition *butterfly-co* :: $\langle 'a::\text{complex-normed-vector} \Rightarrow 'b::\text{hilbert-space} \Rightarrow ('b, 'a) \text{compact-op} \rangle$ **is** *butterfly*
 $\langle \text{proof} \rangle$

lemma *butterfly-co-add-left*: $\langle \text{butterfly-co } (a + a') \ b = \text{butterfly-co } a \ b + \text{butterfly-co } a' \ b \rangle$
 $\langle \text{proof} \rangle$

lemma *butterfly-co-add-right*: $\langle \text{butterfly-co } a \ (b + b') = \text{butterfly-co } a \ b + \text{butterfly-co } a \ b' \rangle$
 $\langle \text{proof} \rangle$

lemma *butterfly-co-scaleR-left[simp]*: $\text{butterfly-co } (r *_R \psi) \ \varphi = r *_C \text{butterfly-co } \psi \ \varphi$
 $\langle \text{proof} \rangle$

lemma *butterfly-co-scaleR-right[simp]*: $\text{butterfly-co } \psi \ (r *_R \varphi) = r *_C \text{butterfly-co } \psi \ \varphi$
 $\langle \text{proof} \rangle$

lemma *butterfly-co-scaleC-left[simp]*: $\text{butterfly-co } (r *_C \psi) \ \varphi = r *_C \text{butterfly-co } \psi \ \varphi$
 $\langle \text{proof} \rangle$

lemma *butterfly-co-scaleC-right[simp]*: $\text{butterfly-co } \psi \ (r *_C \varphi) = \text{conj } r *_C \text{butterfly-co } \psi \ \varphi$
 $\langle \text{proof} \rangle$

lemma *finite-rank-separating-on-compact-op*:

fixes $F \ G :: \langle ('a::\text{hilbert-space}, 'b::\text{hilbert-space}) \text{compact-op} \Rightarrow 'c::\text{complex-normed-vector} \rangle$

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 $\langle \text{bounded-clinear } G \rangle$

shows $\langle F = G \rangle$

$\langle \text{proof} \rangle$

lemma *trunc-ell2-as-Proj*: $\langle \text{trunc-ell2 } S \ \psi = \text{Proj } (\text{ccspan } (\text{ket } 'S)) \ \psi \rangle$
 $\langle \text{proof} \rangle$

lemma *unitary-between-bij-betw*:

assumes $\langle \text{is-onb } A \rangle \ \langle \text{is-onb } B \rangle$

shows $\langle \text{bij-betw } ((*_V) \text{ (unitary-between } A B)) A B \rangle$
 $\langle \text{proof} \rangle$

lemma *tendsto-finite-subsets-at-top-image:*

assumes $\langle \text{inj-on } g X \rangle$

shows $\langle (f \longrightarrow x) \text{ (finite-subsets-at-top } (g \text{ ' } X)) \longleftrightarrow ((\lambda S. f (g \text{ ' } S)) \longrightarrow x) \text{ (finite-subsets-at-top } X) \rangle$

$\langle \text{proof} \rangle$

lemma *Proj-onb-limit:*

shows $\langle \text{is-onb } A \implies ((\lambda S. \text{Proj } (\text{ccspan } S) \psi) \longrightarrow \psi) \text{ (finite-subsets-at-top } A) \rangle$

$\langle \text{proof} \rangle$

lemma *is-ortho-setD:*

assumes *is-ortho-set* $S x \in S y \in S x \neq y$

shows $x \cdot_C y = 0$

$\langle \text{proof} \rangle$

lemma *finite-rank-dense-compact:*

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 $\langle \text{closure } (\text{cspan } ((\lambda(\xi,\eta). \text{butterfly } \xi \eta) \text{ ' } (A \times B))) = \text{Collect compact-op} \rangle$

$\langle \text{proof} \rangle$

lemma *compact-op-comp-left:* $\langle \text{compact-op } (a \text{ } o_{CL} \text{ } b) \rangle$ **if** $\langle \text{compact-op } a \rangle$

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

$\langle \text{proof} \rangle$

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 $\langle \text{finite-dim-ccsubspace } (\text{eigenspace } e \text{ } a) \rangle$

$\langle \text{proof} \rangle$

lemma *eigenvalue-in-the-limit-compact-op:*

— [2], Proposition II.4.14

assumes $\langle \text{compact-op } T \rangle$

assumes $\langle l \neq 0 \rangle$

assumes *normh:* $\langle \bigwedge n. \text{norm } (h \text{ } n) = 1 \rangle$

assumes *Tl-lim:* $\langle (\lambda n. (T - l *_C \text{id-cblinfun}) (h \text{ } n)) \longrightarrow 0 \rangle$

shows $\langle l \in \text{eigenvalues } T \rangle$

$\langle \text{proof} \rangle$

lemma *norm-is-eigenvalue:*

— [2], Proposition II.5.9

```

fixes a :: ⟨'a ⇒CL 'a::{not-singleton, hilbert-space}⟩
assumes ⟨compact-op a⟩
assumes ⟨selfadjoint a⟩
shows ⟨norm a ∈ eigenvalues a ∨ - norm a ∈ eigenvalues a⟩
⟨proof⟩

```

```

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⟩

```

```

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⟩

```

```

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⟩
⟨proof⟩

```

```

lemma compact-op-comp-right: ⟨compact-op (a oCL b)⟩ if ⟨compact-op b⟩
for a b :: ⟨-::hilbert-space ⇒CL -::hilbert-space⟩
⟨proof⟩

```

```

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* :: $\langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow \text{complex} \rangle$
 — The eigenvalues in the spectral decomposition
and *spectral-dec-space* :: $\langle 'a \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow 'a \text{ ccspace} \rangle$
 — The eigenspaces in the spectral decomposition
and *spectral-dec-op* :: $\langle 'a \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow_{CL} 'a) \rangle$
 — A sequence of operators mostly for the proof of spectral composition. But see also *spectral-dec-op-spectral-dec-proj* below.
where $\langle \text{spectral-dec-val } a \ n = \text{largest-eigenvalue } (\text{spectral-dec-op } a \ n) \rangle$
 $\mid \langle \text{spectral-dec-space } a \ n = (\text{if } \text{spectral-dec-val } a \ n = 0 \text{ then } 0 \text{ else } \text{eigenspace } (\text{spectral-dec-val } a \ n) \ (\text{spectral-dec-op } a \ n)) \rangle$
 $\mid \langle \text{spectral-dec-op } a \ (\text{Suc } n) = \text{spectral-dec-op } a \ n \ o_{CL} \ \text{Proj } (- \ \text{spectral-dec-space } a \ n) \rangle$
 $\mid \langle \text{spectral-dec-op } a \ 0 = a \rangle$

definition *spectral-dec-proj* :: $\langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'a \rangle \Rightarrow \text{nat} \Rightarrow ('a \Rightarrow_{CL} 'a) \rangle$ **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 $\langle \text{selfadjoint } (\text{spectral-dec-op } a \ n) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-op-compact*:
assumes $\langle \text{compact-op } a \rangle$
shows $\langle \text{compact-op } (\text{spectral-dec-op } a \ n) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-val-eigenvalue-of-spectral-dec-op*:
fixes $a :: \langle 'a :: \{\text{hilbert-space}, \text{not-singleton}\} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{spectral-dec-val } a \ n \in \text{eigenvalues } (\text{spectral-dec-op } a \ n) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-proj-finite-rank*:
assumes $\langle \text{compact-op } a \rangle$
shows $\langle \text{finite-rank } (\text{spectral-dec-proj } a \ n) \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-spectral-dec-op*:

assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{norm } (\text{spectral-dec-op } a \ n) = \text{cmod } (\text{spectral-dec-val } a \ n) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-op-decreasing-eigenspaces:*

assumes $\langle n \geq m \rangle$ **and** $\langle e \neq 0 \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{eigenspace } e \ (\text{spectral-dec-op } a \ n) \leq \text{eigenspace } e \ (\text{spectral-dec-op } a \ m) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-val-not-not-singleton:*

fixes $a :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \neg \text{class.not-singleton } \text{TYPE}('a) \rangle$
shows $\langle \text{spectral-dec-val } a \ n = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-val-eigenvalue-aux:*

— [2], Theorem II.5.1
fixes $a :: \langle 'a :: \{ \text{hilbert-space}, \text{not-singleton} \} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
assumes *eigen-neq0*: $\langle \text{spectral-dec-val } a \ n \neq 0 \rangle$
shows $\langle \text{spectral-dec-val } a \ n \in \text{eigenvalues } a \rangle$
 $\langle \text{proof} \rangle$

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 *eigen-neq0*: $\langle \text{spectral-dec-val } a \ n \neq 0 \rangle$
shows $\langle \text{spectral-dec-val } a \ n \in \text{eigenvalues } a \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{cmod } (\text{spectral-dec-val } a \ n) \leq \text{cmod } (\text{spectral-dec-val } a \ m) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-val-distinct-aux:*

fixes $a :: \langle ('a :: \{ \text{hilbert-space}, \text{not-singleton} \} \Rightarrow_{CL} 'a) \rangle$
assumes $\langle n \neq m \rangle$
assumes $\langle \text{compact-op } a \rangle$

assumes $\langle \text{selfadjoint } a \rangle$
assumes $\text{neq0}: \langle \text{spectral-dec-val } a \ n \neq 0 \rangle$
shows $\langle \text{spectral-dec-val } a \ n \neq \text{spectral-dec-val } a \ m \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-val-distinct*:
fixes $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle n \neq m \rangle$
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
assumes $\text{neq0}: \langle \text{spectral-dec-val } a \ n \neq 0 \rangle$
shows $\langle \text{spectral-dec-val } a \ n \neq \text{spectral-dec-val } a \ m \rangle$
 $\langle \text{proof} \rangle$

hide-fact *spectral-dec-val-distinct-aux*

lemma *spectral-dec-val-tendsto-0*:

assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{spectral-dec-val } a \longrightarrow 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-op-tendsto*:

assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{spectral-dec-op } a \longrightarrow 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-op-spectral-dec-proj*:

$\langle \text{spectral-dec-op } a \ n = a - (\sum_{i < n. \text{spectral-dec-val } a \ i *_{CL} \text{spectral-dec-proj } a \ i) \rangle$
 $\langle \text{proof} \rangle$

lemma *sequential-tendsto-reorder*:

assumes $\langle \text{inj } g \rangle$
assumes $\langle f \longrightarrow l \rangle$
shows $\langle (f \circ g) \longrightarrow l \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-sums*:

assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle (\lambda n. \text{spectral-dec-val } a \ n *_{CL} \text{spectral-dec-proj } a \ n) \ \text{sums } a \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-val-real*:
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle \text{spectral-dec-val } a \ n \in \mathbb{R} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{orthogonal-spaces } (\text{spectral-dec-space } a \ n) \ (\text{spectral-dec-space } a \ m) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-proj-pos*: $\langle \text{spectral-dec-proj } a \ n \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows *spectral-dec-tendsto-pos-op*: $\langle (\lambda n. \max 0 \ (\text{spectral-dec-val } a \ n) \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a \ n) \ \text{sums } \text{pos-op } a \rangle$ (**is** *?thesis1*)
and *spectral-dec-tendsto-neg-op*: $\langle (\lambda n. - \min \ (\text{spectral-dec-val } a \ n) \ 0 \ *_{\mathbb{C}} \ \text{spectral-dec-proj } a \ n) \ \text{sums } \text{neg-op } a \rangle$ (**is** *?thesis2*)
 $\langle \text{proof} \rangle$

lemma *spectral-dec-tendsto-abs-op*:
assumes $\langle \text{compact-op } a \rangle$
assumes $\langle \text{selfadjoint } a \rangle$
shows $\langle (\lambda n. \text{cmod } (\text{spectral-dec-val } a \ n) \ *_{\mathbb{R}} \ \text{spectral-dec-proj } a \ n) \ \text{sums } \text{abs-op } a \rangle$
 $\langle \text{proof} \rangle$

definition *spectral-dec-vecs* :: $\langle ('a \Rightarrow_{\mathbb{C}L} 'a) \Rightarrow 'a::\text{chilbert-space set} \rangle$ **where**
 $\langle \text{spectral-dec-vecs } a = (\bigcup n. \text{scaleC } (\text{csqrt } (\text{spectral-dec-val } a \ n)) \ \text{'some-onb-of } (\text{spectral-dec-space } a \ n)) \rangle$

lemma *spectral-dec-vecs-ortho*:
assumes $\langle \text{selfadjoint } a \rangle$ **and** $\langle \text{compact-op } a \rangle$
shows $\langle \text{is-ortho-set } (\text{spectral-dec-vecs } a) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-val-nonneg*:
assumes $\langle a \geq 0 \rangle$
assumes $\langle \text{compact-op } a \rangle$
shows $\langle \text{spectral-dec-val } a \ n \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-space-finite-dim[intro]*:

assumes $\langle \text{compact-op } a \rangle$
shows $\langle \text{finite-dim-ccsubspace } (\text{spectral-dec-space } a \ n) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-space-0*:
assumes $\langle \text{spectral-dec-val } a \ n = 0 \rangle$
shows $\langle \text{spectral-dec-space } a \ n = 0 \rangle$
 $\langle \text{proof} \rangle$

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 $h :: \langle 'a :: \{\text{hilbert-space}\} \rangle$
assumes $\langle \text{is-onb } E \rangle$
shows *parseval-abs-summable*: $\langle (\lambda e. (\text{cmod } (e \cdot_C h))^2) \text{ abs-summable-on } E \rangle$
 $\langle \text{proof} \rangle$

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 $\langle ((\lambda e. (\text{norm } (A *_V e))^2) \text{ has-sum } t) E \longleftrightarrow ((\lambda (e,f). (\text{cmod } (f \cdot_C (A *_V e))))^2) \text{ has-sum } t) (E \times F) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle ((\lambda e. (\text{norm } (A *_{\mathcal{V}} e))^2) \text{ has-sum } t) E \longleftrightarrow ((\lambda f. (\text{norm } (A *_{\mathcal{V}} f))^2) \text{ has-sum } t) F \rangle$
 $\langle \text{proof} \rangle$

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 $\langle ((\lambda e. \text{cmod } (e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e))) \text{ has-sum } t) E \longleftrightarrow ((\lambda f. \text{cmod } (f \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} f))) \text{ has-sum } t) F \rangle$

— [1], Corollary 18.2

$\langle \text{proof} \rangle$

definition *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 $\langle \text{trace-class } A \rangle$

$\langle \text{proof} \rangle$

lemma *trace-class-iff-summable*:

assumes $\langle \text{is-onb } E \rangle$

shows $\langle \text{trace-class } A \longleftrightarrow (\lambda e. e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e)) \text{ abs-summable-on } E \rangle$

$\langle \text{proof} \rangle$

lemma *trace-class-0[simp]*: $\langle \text{trace-class } 0 \rangle$

$\langle \text{proof} \rangle$

lemma *trace-class-uminus*: $\langle \text{trace-class } t \Longrightarrow \text{trace-class } (-t) \rangle$

$\langle \text{proof} \rangle$

lemma *trace-class-uminus-iff[simp]*: $\langle \text{trace-class } (-a) = \text{trace-class } a \rangle$

$\langle \text{proof} \rangle$

definition *trace-norm* **where** $\langle \text{trace-norm } A = (\text{if } \text{trace-class } A \text{ then } (\sum_{\infty e \in \text{some-chilbert-basis.}} \text{cmod } (e \cdot_{\mathcal{C}} (\text{abs-op } A *_{\mathcal{V}} e))) \text{ else } 0) \rangle$

definition *trace* **where** $\langle \text{trace } A = (\text{if } \text{trace-class } A \text{ then } (\sum_{\infty e \in \text{some-chilbert-basis.}} e \cdot_{\mathcal{C}} (A *_{\mathcal{V}} e)) \text{ else } 0) \rangle$

lemma *trace-0[simp]*: $\langle \text{trace } 0 = 0 \rangle$

$\langle \text{proof} \rangle$

lemma *trace-class-abs-op[simp]*: $\langle \text{trace-class } (\text{abs-op } A) = \text{trace-class } A \rangle$

$\langle \text{proof} \rangle$

lemma *trace-abs-op[simp]*: $\langle \text{trace } (\text{abs-op } A) = \text{trace-norm } A \rangle$

<proof>

lemma *trace-norm-pos*: $\langle \text{trace-norm } A = \text{trace } A \rangle$ **if** $\langle A \geq 0 \rangle$
<proof>

lemma *trace-norm-alt-def*:

assumes $\langle \text{is-onb } B \rangle$

shows $\langle \text{trace-norm } A = (\text{if } \text{trace-class } A \text{ then } (\sum_{\infty} e \in B. \text{cmod } (e \cdot_C (\text{abs-op } A *_V e))) \text{ else } 0) \rangle$

<proof>

lemma *trace-class-finite-dim[simp]*: $\langle \text{trace-class } A \rangle$ **for** $A :: \langle 'a :: \{ \text{cfinite-dim, chilbert-space} \} \Rightarrow_{CL} 'b :: \text{complex-inner} \rangle$

<proof>

lemma *trace-class-scaleC*: $\langle \text{trace-class } (c *_C a) \rangle$ **if** $\langle \text{trace-class } a \rangle$
<proof>

lemma *trace-scaleC*: $\langle \text{trace } (c *_C a) = c * \text{trace } a \rangle$
<proof>

lemma *trace-uminus*: $\langle \text{trace } (- a) = - \text{trace } a \rangle$
<proof>

lemma *trace-norm-0[simp]*: $\langle \text{trace-norm } 0 = 0 \rangle$
<proof>

lemma *trace-norm-nneg[simp]*: $\langle \text{trace-norm } a \geq 0 \rangle$
<proof>

lemma *trace-norm-scaleC*: $\langle \text{trace-norm } (c *_C a) = \text{norm } c * \text{trace-norm } a \rangle$
<proof>

lemma *trace-norm-nondegenerate*: $\langle a = 0 \rangle$ **if** $\langle \text{trace-class } a \rangle$ **and** $\langle \text{trace-norm } a = 0 \rangle$
<proof>

typedef (**overloaded**) ($'a :: \text{chilbert-space}, 'b :: \text{chilbert-space}$) *trace-class* = $\langle \text{Collect } \text{trace-class} :: ('a \Rightarrow_{CL} 'b) \text{ set} \rangle$

morphisms *from-trace-class Abs-trace-class*

<proof>

setup-lifting *type-definition-trace-class*

lemma *trace-class-from-trace-class[simp]*: $\langle \text{trace-class } (\text{from-trace-class } t) \rangle$
<proof>

lemma *trace-pos*: $\langle \text{trace } a \geq 0 \rangle$ **if** $\langle a \geq 0 \rangle$
<proof>

lemma *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 *trace-adj* and then hide this fact.
 $\langle \text{proof} \rangle$

12.3 Hilbert-Schmidt operators

definition *hilbert-schmidt* **where** $\langle \text{hilbert-schmidt } a \iff \text{trace-class } (a^* \circ_{CL} a) \rangle$

definition *hilbert-schmidt-norm* **where** $\langle \text{hilbert-schmidt-norm } a = \text{sqrt } (\text{trace-norm } (a^* \circ_{CL} a)) \rangle$

lemma *hilbert-schmidtI*: $\langle \text{hilbert-schmidt } a \rangle$ **if** $\langle \text{trace-class } (a^* \circ_{CL} a) \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-0[simp]*: $\langle \text{hilbert-schmidt } 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-norm-pos[simp]*: $\langle \text{hilbert-schmidt-norm } a \geq 0 \rangle$
 $\langle \text{proof} \rangle$

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 $\langle ((\lambda x. (\text{norm } (a *_V x))^2) \text{ has-sum } (\text{hilbert-schmidt-norm } a)^2) B \rangle$

$\langle \text{proof} \rangle$

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 $\langle (\lambda x. (\text{norm } (a *_V x))^2) \text{ summable-on } B \rangle$

$\langle \text{proof} \rangle$

lemma *summable-hilbert-schmidt-norm-square-converse*:

assumes $\langle \text{is-onb } B \rangle$

assumes $\langle (\lambda x. (\text{norm } (a *_V x))^2) \text{ summable-on } B \rangle$

shows $\langle \text{hilbert-schmidt } a \rangle$

$\langle \text{proof} \rangle$

lemma *infsun-hilbert-schmidt-norm-square*:

— [1], Proposition 18.6 (a)

assumes $\langle \text{is-onb } B \rangle$ **and** $\langle \text{hilbert-schmidt } a \rangle$

shows $\langle (\sum_{x \in B} (\text{norm } (a *_V x))^2) = ((\text{hilbert-schmidt-norm } a)^2) \rangle$

$\langle \text{proof} \rangle$

lemma

— [1], Proposition 18.6 (d)

assumes $\langle \text{hilbert-schmidt } b \rangle$

shows *hilbert-schmidt-comp-right*: $\langle \text{hilbert-schmidt } (a \text{ } o_{CL} \text{ } b) \rangle$
and *hilbert-schmidt-norm-comp-right*: $\langle \text{hilbert-schmidt-norm } (a \text{ } o_{CL} \text{ } b) \leq \text{norm } a * \text{hilbert-schmidt-norm } b \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-adj[simp]*:
— Implicit in [1], Proposition 18.6 (b)
assumes $\langle \text{hilbert-schmidt } a \rangle$
shows $\langle \text{hilbert-schmidt } (a^*) \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-norm-adj[simp]*:
— [1], Proposition 18.6 (b)
shows $\langle \text{hilbert-schmidt-norm } (a^*) = \text{hilbert-schmidt-norm } a \rangle$
 $\langle \text{proof} \rangle$

lemma
— [1], Proposition 18.6 (d)
fixes $a :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$ **and** b
assumes $\langle \text{hilbert-schmidt } a \rangle$
shows *hilbert-schmidt-comp-left*: $\langle \text{hilbert-schmidt } (a \text{ } o_{CL} \text{ } b) \rangle$
 $\langle \text{proof} \rangle$

lemma
— [1], Proposition 18.6 (d)
fixes $a :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$ **and** b
assumes $\langle \text{hilbert-schmidt } a \rangle$
shows *hilbert-schmidt-norm-comp-left*: $\langle \text{hilbert-schmidt-norm } (a \text{ } o_{CL} \text{ } b) \leq \text{norm } b * \text{hilbert-schmidt-norm } a \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-scaleC*: $\langle \text{hilbert-schmidt } (c *_{C} a) \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-scaleR*: $\langle \text{hilbert-schmidt } (r *_{R} a) \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-uminus*: $\langle \text{hilbert-schmidt } (- a) \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
 $\langle \text{proof} \rangle$

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 $t \ u :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$
— [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*.
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-minus*: $\langle \text{hilbert-schmidt } (a - b) \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$ **and** $\langle \text{hilbert-schmidt } b \rangle$

for $a\ b :: \langle 'a :: \text{hilbert-space} \Rightarrow_{CL} 'b :: \text{hilbert-space} \rangle$
 $\langle \text{proof} \rangle$

typedef (overloaded) $('a :: \text{hilbert-space}, 'b :: \text{complex-inner})$ *hilbert-schmidt* = $\langle \text{Collect hilbert-schmidt} :: ('a \Rightarrow_{CL} 'b) \text{ set} \rangle$
 $\langle \text{proof} \rangle$

setup-lifting *type-definition-hilbert-schmidt*

instantiation *hilbert-schmidt* :: $(\text{hilbert-space}, \text{hilbert-space})$

$\{ \text{zero}, \text{scaleC}, \text{uminus}, \text{plus}, \text{minus}, \text{dist-norm}, \text{sgn-div-norm}, \text{uniformity-dist}, \text{open-uniformity} \}$ **begin**

lift-definition *zero-hilbert-schmidt* :: $\langle ('a, 'b) \text{ hilbert-schmidt} \rangle$ **is** 0 $\langle \text{proof} \rangle$

lift-definition *norm-hilbert-schmidt* :: $\langle ('a, 'b) \text{ hilbert-schmidt} \Rightarrow \text{real} \rangle$ **is** *hilbert-schmidt-norm*
 $\langle \text{proof} \rangle$

lift-definition *scaleC-hilbert-schmidt* :: $\langle \text{complex} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \rangle$
is *scaleC*
 $\langle \text{proof} \rangle$

lift-definition *scaleR-hilbert-schmidt* :: $\langle \text{real} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \rangle$
is *scaleR*
 $\langle \text{proof} \rangle$

lift-definition *uminus-hilbert-schmidt* :: $\langle ('a, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \rangle$ **is** *uminus*
 $\langle \text{proof} \rangle$

lift-definition *minus-hilbert-schmidt* :: $\langle ('a, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \rangle$ **is** *minus*
 $\langle \text{proof} \rangle$

lift-definition *plus-hilbert-schmidt* :: $\langle ('a, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'b) \text{ hilbert-schmidt} \rangle$ **is** *plus*
 $\langle \text{proof} \rangle$

definition $\langle \text{dist } a\ b = \text{norm } (a - b) \rangle$ **for** $a\ b :: \langle ('a, 'b) \text{ hilbert-schmidt} \rangle$

definition $\langle \text{sgn } x = \text{inverse } (\text{norm } x) *_{\mathbb{R}} x \rangle$ **for** $x :: \langle ('a, 'b) \text{ hilbert-schmidt} \rangle$

definition $\langle \text{uniformity} = (\text{INF } e \in \{ 0 < .. \} . \text{principal } \{ (x :: ('a, 'b) \text{ hilbert-schmidt}, y) . \text{dist } x\ y < e \}) \rangle$

definition $\langle \text{open } U = (\forall x \in U . \forall_F (x', y) \text{ in } \text{INF } e \in \{ 0 < .. \} . \text{principal } \{ (x, y) . \text{norm } (x - y) < e \} . x' = x \longrightarrow y \in U) \rangle$ **for** $U :: \langle ('a, 'b) \text{ hilbert-schmidt set} \rangle$

instance

$\langle \text{proof} \rangle$

end

lift-definition *hs-compose* :: $\langle ('b :: \text{hilbert-space}, 'c :: \text{complex-inner}) \text{ hilbert-schmidt} \Rightarrow ('a :: \text{hilbert-space}, 'b) \text{ hilbert-schmidt} \Rightarrow ('a, 'c) \text{ hilbert-schmidt} \rangle$ **is**

cblinfun-compose

$\langle \text{proof} \rangle$

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** $?thesis3$)
 $\langle \text{proof} \rangle$

lemma trace-exists :

— [1], Proposition 18.9
assumes $\langle \text{is-onb } B \rangle$ **and** $\langle \text{trace-class } A \rangle$
shows $\langle (\lambda e. e \cdot_C (A *_V e)) \text{ summable-on } B \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{trace } (a + b) = \text{trace } a + \text{trace } b \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{hs-times-hs-trace-class}$:

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 $\text{trace-class-iff-hs-times-hs}$ because here the types of B, C are more general.
 $\langle \text{proof} \rangle$

instantiation $\text{hilbert-schmidt} :: (\text{hilbert-space}, \text{hilbert-space}) \text{ complex-vector}$ **begin**

instance

$\langle \text{proof} \rangle$

end

instantiation $\text{hilbert-schmidt} :: (\text{hilbert-space}, \text{hilbert-space}) \text{ complex-inner}$ **begin**

lift-definition $\text{cinner-hilbert-schmidt} :: \langle ('a, 'b) \text{hilbert-schmidt} \Rightarrow ('a, 'b) \text{hilbert-schmidt} \Rightarrow \text{complex} \rangle$ **is**

$\langle \lambda b c. \text{trace } (b * \circ_{CL} c) \rangle$ $\langle \text{proof} \rangle$

instance

$\langle \text{proof} \rangle$

end

lemma $\text{hilbert-schmidt-norm-triangle-ineq}$:

— [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 $a\ b :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
assumes $\langle \text{hilbert-schmidt } a \rangle \langle \text{hilbert-schmidt } b \rangle$
shows $\langle \text{hilbert-schmidt-norm } (a + b) \leq \text{hilbert-schmidt-norm } a + \text{hilbert-schmidt-norm } b \rangle$
 $\langle \text{proof} \rangle$

lift-definition $\text{adj-hs} :: \langle ('a::\text{hilbert-space}, 'b::\text{hilbert-space}) \text{hilbert-schmidt} \Rightarrow ('b, 'a) \text{hilbert-schmidt} \rangle$
is adj
 $\langle \text{proof} \rangle$

lemma adj-hs-plus : $\langle \text{adj-hs } (x + y) = \text{adj-hs } x + \text{adj-hs } y \rangle$
 $\langle \text{proof} \rangle$

lemma adj-hs-minus : $\langle \text{adj-hs } (x - y) = \text{adj-hs } x - \text{adj-hs } y \rangle$
 $\langle \text{proof} \rangle$

lemma norm-adj-hs[simp] : $\langle \text{norm } (\text{adj-hs } x) = \text{norm } x \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{hilbert-schmidt-norm-geq-norm}$:
— [1], Proposition 18.6 (c)
assumes $\langle \text{hilbert-schmidt } a \rangle$
shows $\langle \text{norm } a \leq \text{hilbert-schmidt-norm } a \rangle$
 $\langle \text{proof} \rangle$

12.4 Trace-norm and trace-class, continued

lemma $\text{trace-class-comp-left}$: $\langle \text{trace-class } (a\ o_{CL}\ b) \rangle$ **if** $\langle \text{trace-class } a \rangle$ **for** $a :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
— [1], Theorem 18.11 (a)
 $\langle \text{proof} \rangle$

lemma $\text{trace-class-comp-right}$: $\langle \text{trace-class } (a\ o_{CL}\ b) \rangle$ **if** $\langle \text{trace-class } b \rangle$ **for** $a :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
— [1], Theorem 18.11 (a)
 $\langle \text{proof} \rangle$

lemma

fixes $B :: \langle 'a::\text{hilbert-space set} \rangle$ **and** $A :: \langle 'a \Rightarrow_{CL} 'a \rangle$ **and** $b :: \langle 'b::\text{hilbert-space} \Rightarrow_{CL} 'c::\text{hilbert-space} \rangle$ **and** $c :: \langle 'c \Rightarrow_{CL} 'b \rangle$

shows trace-alt-def :

— [1], Proposition 18.9

$\langle \text{is-onb } B \implies \text{trace } A = (\text{if } \text{trace-class } A \text{ then } (\sum_{e \in B} e \cdot_C (A *_{\mathcal{V}} e)) \text{ else } 0) \rangle$

and trace-hs-times-hs : $\langle \text{hilbert-schmidt } c \implies \text{hilbert-schmidt } b \implies \text{trace } (c\ o_{CL}\ b) =$
 $((\text{of-real } (\text{hilbert-schmidt-norm } ((c*) + b)))^2 - (\text{of-real } (\text{hilbert-schmidt-norm } ((c*) -$
 $b)))^2 -$

$$\begin{aligned} & i * (\text{of-real } (\text{hilbert-schmidt-norm } (((c*) + i *_{\mathcal{C}} b))))^2 + \\ & i * (\text{of-real } (\text{hilbert-schmidt-norm } (((c*) - i *_{\mathcal{C}} b))))^2) / 4 \rangle \end{aligned}$$

$\langle \text{proof} \rangle$

lemma *trace-ket-sum*:

fixes $A :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2} \rangle$
assumes $\langle \text{trace-class } A \rangle$
shows $\langle \text{trace } A = (\sum_{\infty} e. \text{ket } e \cdot_C (A *_V \text{ket } e)) \rangle$
<proof>

lemma *trace-one-dim[simp]*: $\langle \text{trace } A = \text{one-dim-iso } A \rangle$ **for** $A :: \langle 'a::\text{one-dim} \Rightarrow_{CL} 'a \rangle$
<proof>

lemma *trace-has-sum*:

assumes $\langle \text{is-onb } E \rangle$
assumes $\langle \text{trace-class } t \rangle$
shows $\langle ((\lambda e. e \cdot_C (t *_V e)) \text{has-sum trace } t) E \rangle$
<proof>

lemma *trace-sandwich-isometry[simp]*: $\langle \text{trace } (\text{sandwich } U A) = \text{trace } A \rangle$ **if** $\langle \text{isometry } U \rangle$
<proof>

lemma *circularity-of-trace*:

— [1], Theorem 18.11 (e)
fixes $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$ **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>

lemma *trace-butterfly-comp*: $\langle \text{trace } (\text{butterfly } x y \circ_{CL} a) = y \cdot_C (a *_V x) \rangle$
<proof>

lemma *trace-butterfly*: $\langle \text{trace } (\text{butterfly } x y) = y \cdot_C x \rangle$
<proof>

lemma *trace-butterfly-comp'*: $\langle \text{trace } (a \circ_{CL} \text{butterfly } x y) = y \cdot_C (a *_V x) \rangle$
<proof>

lemma *trace-norm-adj[simp]*: $\langle \text{trace-norm } (a^*) = \text{trace-norm } a \rangle$
— [1], Theorem 18.11 (f)
<proof>

lemma *trace-class-adj[simp]*: $\langle \text{trace-class } (a^*) \rangle$ **if** $\langle \text{trace-class } a \rangle$
<proof>

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*

<proof>

lift-definition *selfadjoint-tc* :: $\langle 'a::\text{chilbert-space}, 'a \text{ trace-class} \Rightarrow \text{bool} \rangle$ **is** *selfadjoint**<proof>*

lemma *selfadjoint-tc-def'*: $\langle \text{selfadjoint-tc } a \longleftrightarrow \text{adj-tc } a = a \rangle$
<proof>

lemma *trace-class-finite-dim*^[simp]: $\langle \text{trace-class } A \text{ for } A :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\{\text{cfinite-dim}, \text{chilbert-space}\} \rangle$
<proof>

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>

lemma *trace-class-minus*^[simp]: $\langle \text{trace-class } t \Longrightarrow \text{trace-class } u \Longrightarrow \text{trace-class } (t - u) \rangle$

for $t\ u :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$

<proof>

lemma *trace-plus*:

assumes $\langle \text{trace-class } a \rangle$ $\langle \text{trace-class } b \rangle$

shows $\langle \text{trace } (a + b) = \text{trace } a + \text{trace } b \rangle$

<proof>

hide-fact *trace-plus-prelim*

lemma *trace-class-sum*:

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

assumes $\langle \bigwedge i. i \in I \Longrightarrow \text{trace-class } (a\ i) \rangle$

shows $\langle \text{trace-class } (\sum_{i \in I}. a\ i) \rangle$

<proof>

lemma

assumes $\langle \bigwedge i. i \in I \Longrightarrow \text{trace-class } (a\ i) \rangle$

shows *trace-sum*: $\langle \text{trace } (\sum_{i \in I}. a\ i) = (\sum_{i \in I}. \text{trace } (a\ i)) \rangle$

<proof>

lemma *cmod-trace-times*: $\langle \text{cmod } (\text{trace } (a\ o_{CL}\ b)) \leq \text{norm } a * \text{trace-norm } b \rangle$ **if** $\langle \text{trace-class } b \rangle$

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

— [1], Theorem 18.11 (e)

<proof>

lemma *trace-leq-trace-norm*^[simp]: $\langle \text{cmod } (\text{trace } a) \leq \text{trace-norm } a \rangle$

<proof>

lemma *trace-norm-triangle*:

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

assumes $[simp]$: $\langle trace\text{-}class\ a \rangle \langle trace\text{-}class\ b \rangle$
shows $\langle trace\text{-}norm\ (a + b) \leq trace\text{-}norm\ a + trace\text{-}norm\ b \rangle$
— [1], Theorem 18.11 (a)
 $\langle proof \rangle$

instantiation $trace\text{-}class :: (chilbert\text{-}space, chilbert\text{-}space) \{complex\text{-}vector\}$ **begin**

lift-definition $zero\text{-}trace\text{-}class :: \langle ('a, 'b)\ trace\text{-}class \rangle$ **is** 0 $\langle proof \rangle$
lift-definition $minus\text{-}trace\text{-}class :: \langle ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \rangle$
is $minus$ $\langle proof \rangle$
lift-definition $uminus\text{-}trace\text{-}class :: \langle ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \rangle$ **is** $uminus$ $\langle proof \rangle$
lift-definition $plus\text{-}trace\text{-}class :: \langle ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \rangle$
is $plus$ $\langle proof \rangle$
lift-definition $scaleC\text{-}trace\text{-}class :: \langle complex \Rightarrow ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \rangle$ **is** $scaleC$
 $\langle proof \rangle$
lift-definition $scaleR\text{-}trace\text{-}class :: \langle real \Rightarrow ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \rangle$ **is** $scaleR$
 $\langle proof \rangle$
instance
 $\langle proof \rangle$
end

lemma $from\text{-}trace\text{-}class\ 0[simp]$: $\langle from\text{-}trace\text{-}class\ 0 = 0 \rangle$
 $\langle proof \rangle$

instantiation $trace\text{-}class :: (chilbert\text{-}space, chilbert\text{-}space) \{complex\text{-}normed\text{-}vector\}$ **begin**

lift-definition $norm\text{-}trace\text{-}class :: \langle ('a, 'b)\ trace\text{-}class \Rightarrow real \rangle$ **is** $trace\text{-}norm$ $\langle proof \rangle$
definition $sgn\text{-}trace\text{-}class :: \langle ('a, 'b)\ trace\text{-}class \Rightarrow ('a, 'b)\ trace\text{-}class \rangle$ **where** $\langle sgn\text{-}trace\text{-}class\ a = a /_R\ norm\ a \rangle$
definition $dist\text{-}trace\text{-}class :: \langle ('a, 'b)\ trace\text{-}class \Rightarrow - \Rightarrow - \rangle$ **where** $\langle dist\text{-}trace\text{-}class\ a\ b = norm\ (a - b) \rangle$
definition $[code\ del]: uniformity\text{-}trace\text{-}class = (INF\ e \in \{0 < ..\}. principal\ \{(x :: ('a, 'b)\ trace\text{-}class, y). dist\ x\ y < e\})$
definition $[code\ del]: open\text{-}trace\text{-}class\ U = (\forall x \in U. \forall_F (x', y)\ in\ INF\ e \in \{0 < ..\}. principal\ \{(x, y). dist\ x\ y < e\}. x' = x \longrightarrow y \in U)$ **for** $U :: ('a, 'b)\ trace\text{-}class\ set$
instance
 $\langle proof \rangle$
end

lemma $trace\text{-}norm\text{-}comp\text{-}right$:

fixes $a :: \langle 'b :: chilbert\text{-}space \Rightarrow_{CL}\ 'c :: chilbert\text{-}space \rangle$ **and** $b :: \langle 'a :: chilbert\text{-}space \Rightarrow_{CL}\ 'b \rangle$
assumes $\langle trace\text{-}class\ b \rangle$
shows $\langle trace\text{-}norm\ (a\ o_{CL}\ b) \leq norm\ a * trace\text{-}norm\ b \rangle$
— [1], Theorem 18.11 (g)

⟨proof⟩

lemma *trace-norm-comp-left*:

— [1], Theorem 18.11 (g)

fixes $a :: \langle 'b::\text{hilbert-space} \Rightarrow_{CL} 'c::\text{hilbert-space} \rangle$ **and** $b :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b \rangle$

assumes [simp]: ⟨trace-class a ⟩

shows ⟨trace-norm $(a \circ_{CL} b) \leq \text{trace-norm } a * \text{norm } b$ ⟩

⟨proof⟩

lemma *bounded-clinear-trace-duality*: ⟨trace-class $t \implies \text{bounded-clinear } (\lambda a. \text{trace } (t \circ_{CL} a))$ ⟩

⟨proof⟩

lemma *trace-class-butterfly*[simp]: ⟨trace-class $(\text{butterfly } x \ y)$ ⟩ **for** $x :: \langle 'a::\text{hilbert-space} \rangle$ **and** $y :: \langle 'b::\text{hilbert-space} \rangle$

⟨proof⟩

lemma *trace-adj*: ⟨trace $(a^*) = \text{cnj } (\text{trace } a)$ ⟩

⟨proof⟩

hide-fact *trace-adj-prelim*

lemma *cmod-trace-times'*: ⟨cmod $(\text{trace } (a \circ_{CL} b)) \leq \text{norm } b * \text{trace-norm } a$ ⟩ **if** ⟨trace-class a ⟩

— [1], Theorem 18.11 (e)

⟨proof⟩

lift-definition *iso-trace-class-compact-op-dual'*: ⟨ $('a::\text{hilbert-space}, 'b::\text{hilbert-space})$ trace-class $\implies ('b, 'a)$ compact-op \Rightarrow_{CL} complex⟩ **is**

⟨ $\lambda t c. \text{trace } (\text{from-compact-op } c \circ_{CL} t)$ ⟩

⟨proof⟩

include *lifting-syntax*

⟨proof⟩

lemma *iso-trace-class-compact-op-dual'-apply*: ⟨iso-trace-class-compact-op-dual' $t \ c = \text{trace } (\text{from-compact-op } c \circ_{CL} \text{from-trace-class } t)$ ⟩

⟨proof⟩

lemma *iso-trace-class-compact-op-dual'-plus*: ⟨iso-trace-class-compact-op-dual' $(a + b) = \text{iso-trace-class-compact-op-dual}' a + \text{iso-trace-class-compact-op-dual}' b$ ⟩

⟨proof⟩

lemma *iso-trace-class-compact-op-dual'-scaleC*: ⟨iso-trace-class-compact-op-dual' $(c *_C a) = c *_C \text{iso-trace-class-compact-op-dual}' a$ ⟩

⟨proof⟩

lemma *iso-trace-class-compact-op-dual'-bounded-clinear*[bounded-clinear, simp]:

— [1], Theorem 19.1

⟨bounded-clinear $(\text{iso-trace-class-compact-op-dual}' :: ('a::\text{hilbert-space}, 'b::\text{hilbert-space})$ trace-class $\implies -)$ ⟩

⟨proof⟩

lemma *iso-trace-class-compact-op-dual'-surjective*[simp]:
 ⟨surj (iso-trace-class-compact-op-dual' :: ('a::chilbert-space, 'b::chilbert-space) trace-class ⇒ -)⟩
 ⟨proof⟩

lemma *iso-trace-class-compact-op-dual'-isometric*[simp]:
 — [1], Theorem 19.1
 ⟨norm (iso-trace-class-compact-op-dual' t) = norm t⟩ **for** t :: ⟨('a::chilbert-space, 'b::chilbert-space)
 trace-class⟩
 ⟨proof⟩

instance *trace-class* :: (chilbert-space, chilbert-space) cbanach
 ⟨proof⟩

lemma *trace-norm-geq-cinner-abs-op*: ⟨ $\psi \cdot_C (\text{abs-op } t *_V \psi) \leq \text{trace-norm } t$ ⟩ **if** ⟨trace-class t⟩
and ⟨norm $\psi = 1$ ⟩
 ⟨proof⟩

lemma *norm-leq-trace-norm*: ⟨norm t ≤ trace-norm t⟩ **if** ⟨trace-class t⟩
for t :: ⟨'a::chilbert-space ⇒_{CL} 'b::chilbert-space⟩
 ⟨proof⟩

lemma *clinear-from-trace-class*[iff]: ⟨clinear from-trace-class⟩
 ⟨proof⟩

lemma *bounded-clinear-from-trace-class*[bounded-clinear]:
 ⟨bounded-clinear (from-trace-class :: ('a::chilbert-space, 'b::chilbert-space) trace-class ⇒ -)⟩
 ⟨proof⟩

instantiation *trace-class* :: (chilbert-space, chilbert-space) order **begin**

lift-definition *less-eq-trace-class* :: ⟨('a, 'b) trace-class ⇒ ('a, 'b) trace-class ⇒ bool⟩ **is**
 less-eq⟨proof⟩

lift-definition *less-trace-class* :: ⟨('a, 'b) trace-class ⇒ ('a, 'b) trace-class ⇒ bool⟩ **is**
 less⟨proof⟩

instance

⟨proof⟩

end

lift-definition *compose-tcl* :: ⟨('a::chilbert-space, 'b::chilbert-space) trace-class ⇒ ('c::chilbert-space
 ⇒_{CL} 'a) ⇒ ('c, 'b) trace-class⟩ **is**
 ⟨cblinfun-compose :: 'a ⇒_{CL} 'b ⇒ 'c ⇒_{CL} 'a ⇒ 'c ⇒_{CL} 'b⟩
 ⟨proof⟩

lift-definition *compose-tcr* :: $\langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle \Rightarrow \langle 'c::\text{hilbert-space}, 'a \rangle \text{trace-class} \Rightarrow \langle 'c, 'b \rangle \text{trace-class}$ **is**
 $\langle \text{cblinfun-compose} :: 'a \Rightarrow_{CL} 'b \Rightarrow 'c \Rightarrow_{CL} 'a \Rightarrow 'c \Rightarrow_{CL} 'b \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-compose-tcl*: $\langle \text{norm} (\text{compose-tcl } a \ b) \leq \text{norm } a * \text{norm } b \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-compose-tcr*: $\langle \text{norm} (\text{compose-tcr } a \ b) \leq \text{norm } a * \text{norm } b \rangle$
 $\langle \text{proof} \rangle$

interpretation *compose-tcl*: *bounded-cbilinear compose-tcl*
 $\langle \text{proof} \rangle$

interpretation *compose-tcr*: *bounded-cbilinear compose-tcr*
 $\langle \text{proof} \rangle$

lemma *trace-norm-sandwich*: $\langle \text{trace-norm} (\text{sandwich } e \ t) \leq (\text{norm } e)^2 * \text{trace-norm } t \rangle$ **if**
 $\langle \text{trace-class } t \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-class-sandwich*: $\langle \text{trace-class } b \implies \text{trace-class} (\text{sandwich } a \ b) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle ((=) \implies \text{cr-trace-class} \implies \text{cr-trace-class}) (\lambda e. (*_V) (\text{sandwich } e)) \text{sandwich-tc} \rangle$
 $\langle \text{proof} \rangle$

lemma *from-trace-class-sandwich-tc*:
 $\langle \text{from-trace-class} (\text{sandwich-tc } e \ t) = \text{sandwich } e (\text{from-trace-class } t) \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-sandwich-tc*: $\langle \text{norm} (\text{sandwich-tc } e \ t) \leq (\text{norm } e)^2 * \text{norm } t \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-pos*: $\langle \text{sandwich-tc } e \ t \geq 0 \rangle$ **if** $\langle t \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-scaleC-right*: $\langle \text{sandwich-tc } e (c *_C t) = c *_C \text{sandwich-tc } e \ t \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-plus*: $\langle \text{sandwich-tc } e (t + u) = \text{sandwich-tc } e \ t + \text{sandwich-tc } e \ u \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-minus*: $\langle \text{sandwich-tc } e (t - u) = \text{sandwich-tc } e t - \text{sandwich-tc } e u \rangle$
<proof>

lemma *sandwich-tc-uminus-right*: $\langle \text{sandwich-tc } e (- t) = - \text{sandwich-tc } e t \rangle$
<proof>

lemma *trace-comp-pos*:
fixes $a b :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{trace-class } b \rangle$
assumes $\langle a \geq 0 \rangle$ **and** $\langle b \geq 0 \rangle$
shows $\langle \text{trace } (a \circ_{CL} b) \geq 0 \rangle$
<proof>

lemma *trace-norm-one-dim*: $\langle \text{trace-norm } x = \text{cmod } (\text{one-dim-iso } x) \rangle$
<proof>

lemma *trace-norm-bounded*:
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 $\langle \text{trace-class } A \rangle$
<proof>

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>

lemma *norm-cblinfun-mono-trace-class*:
fixes $A B :: \langle ('a::\text{chilbert-space}, 'a) \text{ trace-class} \rangle$
assumes $\langle A \geq 0 \rangle$
assumes $\langle A \leq B \rangle$
shows $\langle \text{norm } A \leq \text{norm } B \rangle$
<proof>

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>

lemma *from-trace-class-sum*:
shows $\langle \text{from-trace-class } (\sum_{x \in M}. f x) = (\sum_{x \in M}. \text{from-trace-class } (f x)) \rangle$
<proof>

lemma *has-sum-mono-neutral-traceclass*:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space}, 'b) \text{trace-class}$
assumes $\langle f \text{ has-sum } a \rangle A$ **and** $\langle g \text{ has-sum } b \rangle B$
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 $a \leq b$
 $\langle \text{proof} \rangle$

lemma *has-sum-mono-traceclass*:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space}, 'b) \text{trace-class}$
assumes $\langle f \text{ has-sum } x \rangle A$ **and** $\langle g \text{ has-sum } y \rangle A$
assumes $\langle \bigwedge x. x \in A \implies f x \leq g x \rangle$
shows $x \leq y$
 $\langle \text{proof} \rangle$

lemma *infsun-mono-traceclass*:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space}, 'b) \text{trace-class}$
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 $\text{infsun } f A \leq \text{infsun } g A$
 $\langle \text{proof} \rangle$

lemma *infsun-mono-neutral-traceclass*:
fixes $f :: 'a \Rightarrow ('b::\text{chilbert-space}, 'b) \text{trace-class}$
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 $\text{infsun } f A \leq \text{infsun } g B$
 $\langle \text{proof} \rangle$

instance *trace-class* :: $(\text{chilbert-space}, \text{chilbert-space}) \text{ordered-complex-vector}$
 $\langle \text{proof} \rangle$

lemma *Abs-trace-class-geq0I*: $\langle 0 \leq \text{Abs-trace-class } t \rangle$ **if** $\langle \text{trace-class } t \rangle$ **and** $\langle t \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lift-definition *tc-compose* :: $\langle ('b::\text{chilbert-space}, 'c::\text{chilbert-space}) \text{trace-class} \implies ('a::\text{chilbert-space}, 'b) \text{trace-class} \implies ('a, 'c) \text{trace-class} \rangle$ **is**
 cblinfun-compose
 $\langle \text{proof} \rangle$

lemma *norm-tc-compose*:
 $\langle \text{norm } (\text{tc-compose } a b) \leq \text{norm } a * \text{norm } b \rangle$
 $\langle \text{proof} \rangle$

lift-definition *trace-tc* :: $\langle 'a::\text{chilbert-space}, 'a \rangle \text{trace-class} \Rightarrow \text{complex}$ **is** *trace* $\langle \text{proof} \rangle$

lemma *trace-tc-plus*: $\langle \text{trace-tc } (a + b) = \text{trace-tc } a + \text{trace-tc } b \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-tc-scaleC*: $\langle \text{trace-tc } (c *_C a) = c *_C \text{trace-tc } a \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-tc-norm*: $\langle \text{norm } (\text{trace-tc } a) \leq \text{norm } a \rangle$
 $\langle \text{proof} \rangle$

lemma *bounded-clinear-trace-tc*[*bounded-clinear, simp*]: $\langle \text{bounded-clinear } \text{trace-tc} \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-tc-pos*: $\langle \text{norm } A = \text{trace-tc } A \rangle$ **if** $\langle A \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-tc-pos-Re*: $\langle \text{norm } A = \text{Re } (\text{trace-tc } A) \rangle$ **if** $\langle A \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *from-trace-class-pos*: $\langle \text{from-trace-class } A \geq 0 \iff A \geq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *infsun-tc-norm-bounded-abs-summable*:

fixes $A :: \langle 'a \Rightarrow ('b::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{trace-class} \rangle$

assumes *pos*: $\langle \bigwedge x. x \in M \implies A x \geq 0 \rangle$

assumes *bound-B*: $\langle \bigwedge F. \text{finite } F \implies F \subseteq M \implies \text{norm } (\sum_{x \in F}. A x) \leq B \rangle$

shows $\langle A \text{abs-summable-on } M \rangle$

$\langle \text{proof} \rangle$

lemma *trace-norm-uminus*[*simp*]: $\langle \text{trace-norm } (-a) = \text{trace-norm } a \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-norm-triangle-minus*:

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 $\langle \text{trace-norm } (a - b) \leq \text{trace-norm } a + \text{trace-norm } b \rangle$

$\langle \text{proof} \rangle$

lemma *trace-norm-abs-op*[*simp*]: $\langle \text{trace-norm } (\text{abs-op } t) = \text{trace-norm } t \rangle$
 $\langle \text{proof} \rangle$

lemma

fixes $t :: \langle 'a \Rightarrow_{CL} 'a::\text{chilbert-space} \rangle$

shows *cblinfun-decomp-4pos*: \langle

$\exists t1 t2 t3 t4.$

$t = t1 - t2 + i *_C t3 - i *_C t4$

$\wedge t1 \geq 0 \wedge t2 \geq 0 \wedge t3 \geq 0 \wedge t4 \geq 0 \rangle$ (**is** *?thesis1*)

and *trace-class-decomp-4pos*: $\langle \text{trace-class } t \implies$
 $\exists t1\ t2\ t3\ t4.$
 $t = t1 - t2 + i *_C t3 - i *_C t4$
 $\wedge \text{trace-class } t1 \wedge \text{trace-class } t2 \wedge \text{trace-class } t3 \wedge \text{trace-class } t4$
 $\wedge \text{trace-norm } t1 \leq \text{trace-norm } t \wedge \text{trace-norm } t2 \leq \text{trace-norm } t \wedge \text{trace-norm } t3$
 $\leq \text{trace-norm } t \wedge \text{trace-norm } t4 \leq \text{trace-norm } t$
 $\wedge t1 \geq 0 \wedge t2 \geq 0 \wedge t3 \geq 0 \wedge t4 \geq 0 \rangle$ (**is** $\langle - \implies ?thesis2 \rangle$)
 $\langle \text{proof} \rangle$

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
 $\wedge t1 \geq 0 \wedge t2 \geq 0 \wedge t3 \geq 0 \wedge t4 \geq 0 \rangle$
 $\langle \text{proof} \rangle$

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 $t :: \langle - :: \text{chilbert-space} \Rightarrow_{CL} - :: \text{chilbert-space} \rangle$
 $\langle \text{proof} \rangle$

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 $\text{infsum } f\ M \geq 0$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-compose*: $\langle \text{sandwich-tc } (A \ o_{CL} \ B) = \text{sandwich-tc } A \ o \ \text{sandwich-tc } B \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-0-left[simp]*: $\langle \text{sandwich-tc } 0 = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-0-right[simp]*: $\langle \text{sandwich-tc } e \ 0 = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-scaleC-left*: $\langle \text{sandwich-tc } (c *_C \ e) \ t = (c \ \text{mod } c) \hat{\ }^2 *_C \ \text{sandwich-tc } e \ t \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-tc-scaleR-left*: $\langle \text{sandwich-tc } (r *_R \ e) \ t = r \hat{\ }^2 *_R \ \text{sandwich-tc } e \ t \rangle$
 $\langle \text{proof} \rangle$

lemma *bounded-cbilinear-tc-compose*: $\langle \text{bounded-cbilinear } \text{tc-compose} \rangle$
 $\langle \text{proof} \rangle$

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*
 $\langle \text{proof} \rangle$

lemma *norm-tc-butterfly*: $\langle \text{norm } (tc\text{-butterfly } \psi \ \varphi) = \text{norm } \psi * \text{norm } \varphi \rangle$
 $\langle \text{proof} \rangle$

lemma *comp-tc-butterfly[simp]*: $\langle tc\text{-compose } (tc\text{-butterfly } a \ b) \ (tc\text{-butterfly } c \ d) = (b \cdot_C \ c) *_C \ tc\text{-butterfly } a \ d \rangle$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-pos[simp]*: $\langle 0 \leq tc\text{-butterfly } \psi \ \psi \rangle$
 $\langle \text{proof} \rangle$

lift-definition *rank1-tc* :: $\langle ('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{ trace-class} \Rightarrow \text{bool} \rangle$ **is** *rank1* $\langle \text{proof} \rangle$

lift-definition *finite-rank-tc* :: $\langle ('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{ trace-class} \Rightarrow \text{bool} \rangle$ **is** *finite-rank* $\langle \text{proof} \rangle$

lemma *finite-rank-tc-0[iff]*: $\langle \text{finite-rank-tc } 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-tc-plus*: $\langle \text{finite-rank-tc } (a + b) \rangle$
if $\langle \text{finite-rank-tc } a \rangle$ **and** $\langle \text{finite-rank-tc } b \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-tc-scale*: $\langle \text{finite-rank-tc } (c *_C \ a) \rangle$ **if** $\langle \text{finite-rank-tc } a \rangle$
 $\langle \text{proof} \rangle$

lemma *csubspace-finite-rank-tc*: $\langle \text{csubspace } (\text{Collect } \text{finite-rank-tc}) \rangle$
 $\langle \text{proof} \rangle$

lemma *rank1-trace-class*: $\langle \text{trace-class } a \rangle$ **if** $\langle \text{rank1 } a \rangle$
for $a \ b :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-trace-class*: $\langle \text{trace-class } a \rangle$ **if** $\langle \text{finite-rank } a \rangle$
for $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-minus*:
assumes $\langle \text{trace-class } a \rangle \langle \text{trace-class } b \rangle$
shows $\langle \text{trace } (a - b) = \text{trace } a - \text{trace } b \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-cblinfun-mono*:
fixes $A \ B :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'a \rangle$
assumes $\langle \text{trace-class } A \rangle$ **and** $\langle \text{trace-class } B \rangle$

assumes $\langle A \leq B \rangle$
shows $\langle \text{trace } A \leq \text{trace } B \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-tc-mono*:
assumes $\langle A \leq B \rangle$
shows $\langle \text{trace-tc } A \leq \text{trace-tc } B \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-tc-0[simp]*: $\langle \text{trace-tc } 0 = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *cspan-tc-transfer[transfer-rule]*:
includes *lifting-syntax*
shows $\langle (\text{rel-set } \text{cr-trace-class} \text{ ===} \rangle \text{ rel-set cr-trace-class}) \text{ cspan cspan} \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-tc-def'*: $\langle \text{finite-rank-tc } A \longleftrightarrow A \in \text{cspan } (\text{Collect rank1-tc}) \rangle$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-add-left*: $\langle \text{tc-butterfly } (a + a') \text{ } b = \text{tc-butterfly } a \text{ } b + \text{tc-butterfly } a' \text{ } b \rangle$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-add-right*: $\langle \text{tc-butterfly } a \text{ } (b + b') = \text{tc-butterfly } a \text{ } b + \text{tc-butterfly } a \text{ } b' \rangle$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-sum-left*: $\langle \text{tc-butterfly } (\sum_{i \in M}. \psi \text{ } i) \text{ } \varphi = (\sum_{i \in M}. \text{tc-butterfly } (\psi \text{ } i) \text{ } \varphi) \rangle$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-sum-right*: $\langle \text{tc-butterfly } \psi \text{ } (\sum_{i \in M}. \varphi \text{ } i) = (\sum_{i \in M}. \text{tc-butterfly } \psi \text{ } (\varphi \text{ } i)) \rangle$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-scaleC-left[simp]*: $\text{tc-butterfly } (c *_{\mathcal{C}} \psi) \text{ } \varphi = c *_{\mathcal{C}} \text{tc-butterfly } \psi \text{ } \varphi$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-scaleC-right[simp]*: $\text{tc-butterfly } \psi \text{ } (c *_{\mathcal{C}} \varphi) = \text{cnj } c *_{\mathcal{C}} \text{tc-butterfly } \psi \text{ } \varphi$
 $\langle \text{proof} \rangle$

lemma *bounded-sesquilinear-tc-butterfly[iff]*: $\langle \text{bounded-sesquilinear } (\lambda a \text{ } b. \text{tc-butterfly } b \text{ } a) \rangle$
 $\langle \text{proof} \rangle$

lemma *trace-norm-plus-orthogonal*:
assumes $\langle \text{trace-class } a \rangle$ **and** $\langle \text{trace-class } b \rangle$
assumes $\langle a *_{\mathcal{O}_{\mathcal{L}}} b = 0 \rangle$ **and** $\langle a \text{ }_{\mathcal{O}_{\mathcal{L}}} b * = 0 \rangle$
shows $\langle \text{trace-norm } (a + b) = \text{trace-norm } a + \text{trace-norm } b \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-tc-plus-orthogonal*:

assumes $\langle tc\text{-compose } (adj\text{-tc } a) \ b = 0 \rangle$ **and** $\langle tc\text{-compose } a \ (adj\text{-tc } b) = 0 \rangle$
shows $\langle norm \ (a + b) = norm \ a + norm \ b \rangle$
 $\langle proof \rangle$

lemma *trace-norm-sum-exchange*:

fixes $t :: \langle - \Rightarrow (-::chilbert\text{-space} \Rightarrow_{CL} -::chilbert\text{-space}) \rangle$
assumes $\langle \bigwedge i. i \in F \Longrightarrow trace\text{-class } (t \ i) \rangle$
assumes $\langle \bigwedge i \ j. i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow (t \ i)^* \ o_{CL} \ t \ j = 0 \rangle$
assumes $\langle \bigwedge i \ j. i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow t \ i \ o_{CL} \ (t \ j)^* = 0 \rangle$
shows $\langle trace\text{-norm } (\sum_{i \in F}. t \ i) = (\sum_{i \in F}. trace\text{-norm } (t \ i)) \rangle$
 $\langle proof \rangle$

lemma *norm-tc-sum-exchange*:

assumes $\langle \bigwedge i \ j. i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow tc\text{-compose } (adj\text{-tc } (t \ i)) \ (t \ j) = 0 \rangle$
assumes $\langle \bigwedge i \ j. i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow tc\text{-compose } (t \ i) \ (adj\text{-tc } (t \ j)) = 0 \rangle$
shows $\langle norm \ (\sum_{i \in F}. t \ i) = (\sum_{i \in F}. norm \ (t \ i)) \rangle$
 $\langle proof \rangle$

instantiation *trace-class* :: $(one\text{-dim}, one\text{-dim})$ *complex-inner* **begin**

lift-definition *cinner-trace-class* :: $\langle ('a, 'b) \ trace\text{-class} \Rightarrow ('a, 'b) \ trace\text{-class} \Rightarrow complex \rangle$ **is**
 $\langle (\cdot_C) \rangle \langle proof \rangle$

instance

$\langle proof \rangle$

end

instantiation *trace-class* :: $(one\text{-dim}, one\text{-dim})$ *one-dim* **begin**

lift-definition *one-trace-class* :: $\langle ('a, 'b) \ trace\text{-class} \rangle$ **is** 1

$\langle proof \rangle$

lift-definition *times-trace-class* :: $\langle ('a, 'b) \ trace\text{-class} \Rightarrow ('a, 'b) \ trace\text{-class} \Rightarrow ('a, 'b) \ trace\text{-class} \rangle$

is $\langle (*) \rangle$

$\langle proof \rangle$

lift-definition *divide-trace-class* :: $\langle ('a, 'b) \ trace\text{-class} \Rightarrow ('a, 'b) \ trace\text{-class} \Rightarrow ('a, 'b) \ trace\text{-class} \rangle$

is $\langle (/) \rangle$

$\langle proof \rangle$

lift-definition *inverse-trace-class* :: $\langle ('a, 'b) \ trace\text{-class} \Rightarrow ('a, 'b) \ trace\text{-class} \rangle$ **is** $\langle Fields.inverse \rangle$

$\langle proof \rangle$

definition *canonical-basis-trace-class* :: $\langle ('a, 'b) \ trace\text{-class} \ list \rangle$ **where** $\langle canonical\text{-basis-trace-class} = [1] \rangle$

definition *canonical-basis-length-trace-class* :: $\langle ('a, 'b) \ trace\text{-class} \ itself \Rightarrow nat \rangle$ **where** $\langle canonical\text{-basis-length-trace-class} = 1 \rangle$

instance

$\langle proof \rangle$

end

lemma *from-trace-class-one-dim-iso*[simp]: $\langle from\text{-trace-class} = one\text{-dim-iso} \rangle$

<proof>

lemma *trace-tc-one-dim-iso*[simp]: $\langle \text{trace-tc} = \text{one-dim-iso} \rangle$
<proof>

lemma *compose-tcr-id-left*[simp]: $\langle \text{compose-tcr id-cblinfun } t = t \rangle$
<proof>

lemma *compose-tcl-id-right*[simp]: $\langle \text{compose-tcl } t \text{ id-cblinfun} = t \rangle$
<proof>

lemma *sandwich-tc-id-cblinfun*[simp]: $\langle \text{sandwich-tc id-cblinfun } t = t \rangle$
<proof>

lemma *bounded-clinear-sandwich-tc*[bounded-clinear]: $\langle \text{bounded-clinear} (\text{sandwich-tc } e) \rangle$
<proof>

lemma *trace-class-Proj*: $\langle \text{trace-class} (\text{Proj } S) \iff \text{finite-dim-ccsubspace } S \rangle$
<proof>

lemma *not-trace-class-trace0*: $\langle \text{trace } a = 0 \rangle \text{ if } \langle \neg \text{trace-class } a \rangle$
<proof>

lemma *trace-Proj*: $\langle \text{trace} (\text{Proj } S) = \text{cdim} (\text{space-as-set } S) \rangle$
<proof>

lemma *trace-tc-pos*: $\langle t \geq 0 \implies \text{trace-tc } t \geq 0 \rangle$
<proof>

lift-definition *tc-apply* :: $\langle ('a :: \text{hilbert-space}, 'b :: \text{hilbert-space}) \text{ trace-class} \implies 'a \implies 'b \rangle$ is *cblin-fun-apply**<proof>*

lemma *bounded-cbilinear-tc-apply*: $\langle \text{bounded-cbilinear } \text{tc-apply} \rangle$
<proof>

lift-definition *diagonal-operator-tc* :: $\langle ('a \implies \text{complex}) \implies ('a \text{ ell2}, 'a \text{ ell2}) \text{ trace-class} \rangle$ is
 $\langle \lambda f. \text{if } f \text{ abs-summable-on UNIV then diagonal-operator } f \text{ else } 0 \rangle$
<proof>

lemma *from-trace-class-diagonal-operator-tc*:
assumes $\langle f \text{ abs-summable-on UNIV} \rangle$
shows $\langle \text{from-trace-class} (\text{diagonal-operator-tc } f) = \text{diagonal-operator } f \rangle$
<proof>

lemma *tc-butterfly-scaleC-summable*:
fixes $f :: \langle 'a \implies \text{complex} \rangle$
assumes $\langle f \text{ abs-summable-on } A \rangle$

shows $\langle (\lambda x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) \text{ summable-on } A \rangle$
 $\langle \text{proof} \rangle$

lemma *tc-butterfly-scaleC-has-sum:*

fixes $f :: \langle 'a \Rightarrow \text{complex} \rangle$

assumes $\langle f \text{ abs-summable-on } UNIV \rangle$

shows $\langle (\lambda x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) \text{ has-sum diagonal-operator-tc } f \rangle UNIV$

$\langle \text{proof} \rangle$

lemma *diagonal-operator-tc-invalid:* $\langle \neg f \text{ abs-summable-on } UNIV \implies \text{diagonal-operator-tc } f = 0 \rangle$

$\langle \text{proof} \rangle$

lemma *tc-butterfly-scaleC-infsum:*

fixes $f :: \langle 'a \Rightarrow \text{complex} \rangle$

shows $\langle (\sum_{\infty} x. f x *_C \text{tc-butterfly } (\text{ket } x) (\text{ket } x)) = \text{diagonal-operator-tc } f \rangle$

$\langle \text{proof} \rangle$

lemma *from-trace-class-abs-summable:* $\langle f \text{ abs-summable-on } X \implies (\lambda x. \text{from-trace-class } (f x)) \text{ abs-summable-on } X \rangle$

$\langle \text{proof} \rangle$

lemma *from-trace-class-summable:* $\langle f \text{ summable-on } X \implies (\lambda x. \text{from-trace-class } (f x)) \text{ summable-on } X \rangle$

$\langle \text{proof} \rangle$

lemma *from-trace-class-infsum:*

assumes $\langle f \text{ summable-on } UNIV \rangle$

shows $\langle \text{from-trace-class } (\sum_{\infty} x. f x) = (\sum_{\infty} x. \text{from-trace-class } (f x)) \rangle$

$\langle \text{proof} \rangle$

lemma *cspan-trace-class:*

$\langle \text{cspan } (\text{Collect trace-class } :: ('a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space}) \text{ set}) = \text{Collect trace-class} \rangle$

$\langle \text{proof} \rangle$

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 $\langle \exists L. (f \longrightarrow L) F \rangle$

$\langle \text{proof} \rangle$

lemma *nonneg-bdd-above-summable-on-tc:*

fixes $f :: \langle 'a \Rightarrow ('c::\text{chilbert-space}, 'c) \text{ trace-class} \rangle$

assumes $pos: \langle \bigwedge x. x \in A \implies f x \geq 0 \rangle$
assumes $bdd: \langle bdd\text{-above } (trace\text{-tc } ' \{F. F \subseteq A \wedge finite\ F\}) \rangle$
shows $\langle f \text{ summable-on } A \rangle$
 $\langle proof \rangle$

lemma *summable-Sigma-positive-tc*:
fixes $f :: \langle 'a \Rightarrow 'b \Rightarrow ('c, 'c::chilbert\text{-space}) \text{ trace-class} \rangle$
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 $\langle (\lambda(x, y). f x y) \text{ summable-on } (SIGMA x:X. Y x) \rangle$
 $\langle proof \rangle$

lemma *infsum-Sigma-positive-tc*:
fixes $f :: \langle 'a \Rightarrow 'b \Rightarrow ('c::chilbert\text{-space}, 'c) \text{ trace-class} \rangle$
assumes $\langle \bigwedge x. x \in X \implies f x \text{ summable-on } Y x \rangle$
assumes $\langle \bigwedge x y. x \in X \implies y \in Y x \implies f x y \geq 0 \rangle$
shows $\langle (\sum_{\infty} x \in X. \sum_{\infty} y \in Y x. f x y) = (\sum_{\infty} (x,y) \in Sigma\ X\ Y. f x y) \rangle$
 $\langle proof \rangle$

lemma *infsum-swap-positive-tc*:
fixes $f :: \langle 'a \Rightarrow 'b \Rightarrow ('c::chilbert\text{-space}, 'c) \text{ trace-class} \rangle$
assumes $\langle \bigwedge x. x \in X \implies f x \text{ summable-on } Y \rangle$
assumes $\langle \bigwedge y. y \in Y \implies (\lambda x. f x y) \text{ summable-on } X \rangle$
assumes $\langle \bigwedge x y. x \in X \implies y \in Y \implies f x y \geq 0 \rangle$
shows $\langle (\sum_{\infty} x \in X. \sum_{\infty} y \in Y. f x y) = (\sum_{\infty} y \in Y. \sum_{\infty} x \in X. f x y) \rangle$
 $\langle proof \rangle$

lemma *separating-density-ops*:
assumes $\langle B > 0 \rangle$
shows $\langle separating\text{-set clinear } \{t :: ('a::chilbert\text{-space}, 'a) \text{ trace-class. } 0 \leq t \wedge norm\ t \leq B\} \rangle$
 $\langle proof \rangle$

lemma *summable-abs-summable-tc*:
fixes $f :: \langle 'a \Rightarrow ('b::chilbert\text{-space}, 'b) \text{ trace-class} \rangle$
assumes $\langle f \text{ summable-on } X \rangle$
assumes $\langle \bigwedge x. x \in X \implies f x \geq 0 \rangle$
shows $\langle f \text{ abs-summable-on } X \rangle$
 $\langle proof \rangle$

12.5 More Hilbert-Schmidt

lemma *trace-class-hilbert-schmidt*: $\langle hilbert\text{-schmidt } a \rangle$ **if** $\langle trace\text{-class } a \rangle$
for $a :: \langle 'a::chilbert\text{-space} \Rightarrow_{CL} 'b::chilbert\text{-space} \rangle$
 $\langle proof \rangle$

lemma *finite-rank-hilbert-schmidt*: $\langle \text{hilbert-schmidt } a \rangle$ **if** $\langle \text{finite-rank } a \rangle$
for $a :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
 $\langle \text{proof} \rangle$

lemma *hilbert-schmidt-compact*: $\langle \text{compact-op } a \rangle$ **if** $\langle \text{hilbert-schmidt } a \rangle$
for $a :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
— [1], Corollary 18.7. (Only the second part. The first part is stated inside this proof though.)
 $\langle \text{proof} \rangle$

lemma *trace-class-compact*: $\langle \text{compact-op } a \rangle$ **if** $\langle \text{trace-class } a \rangle$
for $a :: \langle 'a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space} \rangle$
 $\langle \text{proof} \rangle$

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* :: $\langle ('a::\text{hilbert-space}, 'a) \text{ trace-class} \Rightarrow \text{nat} \Rightarrow ('a, 'a) \text{ trace-class} \rangle$
is
spectral-dec-proj
 $\langle \text{proof} \rangle$

lift-definition *spectral-dec-val-tc* :: $\langle ('a::\text{hilbert-space}, 'a) \text{ trace-class} \Rightarrow \text{nat} \Rightarrow \text{complex} \rangle$ **is**
spectral-dec-val $\langle \text{proof} \rangle$

lemma *spectral-dec-proj-tc-finite-rank*:
assumes $\langle \text{adj-tc } a = a \rangle$
shows $\langle \text{finite-rank-tc } (\text{spectral-dec-proj-tc } a \ n) \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-summable-tc*:
assumes $\langle \text{selfadjoint-tc } a \rangle$
shows $\langle (\lambda n. \text{spectral-dec-val-tc } a \ n *_{CL} \text{spectral-dec-proj-tc } a \ n) \text{ abs-summable-on } UNIV \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-has-sum-tc*:
assumes $\langle \text{selfadjoint-tc } a \rangle$
shows $\langle ((\lambda n. \text{spectral-dec-val-tc } a \ n *_{CL} \text{spectral-dec-proj-tc } a \ n) \text{ has-sum } a) UNIV \rangle$
 $\langle \text{proof} \rangle$

lemma *spectral-dec-sums-tc*:
assumes $\langle \text{selfadjoint-tc } a \rangle$
shows $\langle (\lambda n. \text{spectral-dec-val-tc } a \ n *_{CL} \text{spectral-dec-proj-tc } a \ n) \text{ sums } a \rangle$
 $\langle \text{proof} \rangle$

lift-definition *spectral-dec-vecs-tc* :: $\langle 'a, 'a \text{ trace-class} \Rightarrow 'a::\text{hilbert-space set} \rangle$ **is**
spectral-dec-vecs⟨*proof*⟩

lemma *compact-from-trace-class*[*iff*]: $\langle \text{compact-op (from-trace-class } t) \rangle$
 ⟨*proof*⟩

lemma *sum-some-onb-of-tc-butterfly*:
assumes $\langle \text{finite-dim-ccsubspace } S \rangle$
shows $\langle (\sum x \in \text{some-onb-of } S. \text{tc-butterfly } x \ x) = \text{Abs-trace-class (Proj } S) \rangle$
 ⟨*proof*⟩

lemma *butterfly-spectral-dec-vec-tc-has-sum*:
assumes $\langle t \geq 0 \rangle$
shows $\langle (\lambda v. \text{tc-butterfly } v \ v) \text{ has-sum } t \rangle (\text{spectral-dec-vecs-tc } t) \rangle$
 ⟨*proof*⟩

lemma *spectral-dec-vec-tc-norm-summable*:
assumes $\langle t \geq 0 \rangle$
shows $\langle (\lambda v. (\text{norm } v)^2) \text{ summable-on (spectral-dec-vecs-tc } t) \rangle$
 ⟨*proof*⟩

12.7 More Trace-Class

lemma *finite-rank-tc-dense-aux*: $\langle \text{closure (Collect finite-rank-tc :: ('a::hilbert-space, 'a) trace-class set)} = \text{UNIV} \rangle$
 ⟨*proof*⟩

lemma *finite-rank-tc-dense*: $\langle \text{closure (Collect finite-rank-tc :: ('a::hilbert-space, 'b::hilbert-space) trace-class set)} = \text{UNIV} \rangle$
 ⟨*proof*⟩

hide-fact *finite-rank-tc-dense-aux*

lemma *onb-butterflies-span-trace-class*:
fixes $A :: \langle 'a::\text{hilbert-space set} \rangle$ **and** $B :: \langle 'b::\text{hilbert-space set} \rangle$
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*⟩

lemma *separating-set-tc-butterfly*: $\langle \text{separating-set bounded-clinear } ((\lambda(g, h). \text{tc-butterfly } g \ h) \ ` (\text{UNIV} \times \text{UNIV})) \rangle$
 ⟨*proof*⟩

lemma *separating-set-tc-butterfly-nested*:
assumes $\langle \text{separating-set (bounded-clinear :: (-} \Rightarrow 'c::\text{complex-normed-vector}) \Rightarrow -) \ A \rangle$

assumes $\langle \text{separating-set } (\text{bounded-clinear} :: (- \Rightarrow 'c \text{ conjugate-space}) \Rightarrow -) B \rangle$
shows $\langle \text{separating-set } (\text{bounded-clinear} :: (- \Rightarrow 'c) \Rightarrow -) ((\lambda(g,h). \text{tc-butterfly } g \ h) ' (A \times B)) \rangle$
 $\langle \text{proof} \rangle$

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* :: $\langle ('a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space}) \text{ topology} \rangle$

where $\langle \text{weak-star-topology} = \text{pullback-topology UNIV } (\lambda x. \lambda t \in \text{Collect trace-class. trace } (t \ o_{CL} \ x))$

$(\text{product-topology } (\lambda-. \text{euclidean}) \ (\text{Collect trace-class})) \rangle$

lemma *open-map-product-topology-reindex*:

fixes $\pi :: \langle 'b \Rightarrow 'a \rangle$

assumes *bij- π* : $\langle \text{bij-betw } \pi \ B \ A \rangle$ **and** *ST*: $\langle \bigwedge x. x \in B \Longrightarrow S \ x = T \ (\pi \ x) \rangle$

assumes *g-def*: $\langle \bigwedge f. g \ f = \text{restrict } (f \ o \ \pi) \ B \rangle$

shows $\langle \text{open-map } (\text{product-topology } T \ A) \ (\text{product-topology } S \ B) \ g \rangle$

$\langle \text{proof} \rangle$

lemma *homeomorphic-map-product-topology-reindex*:

fixes $\pi :: \langle 'b \Rightarrow 'a \rangle$

assumes *bij- π* : $\langle \text{bij-betw } \pi \ B \ A \rangle$ **and** *ST*: $\langle \bigwedge x. x \in B \Longrightarrow S \ x = T \ (\pi \ x) \rangle$

assumes *g-def*: $\langle \bigwedge f. g \ f = \text{restrict } (f \ o \ \pi) \ B \rangle$

shows $\langle \text{homeomorphic-map } (\text{product-topology } T \ A) \ (\text{product-topology } S \ B) \ g \rangle$

$\langle \text{proof} \rangle$

lemma *weak-star-topology-def'*:

$\langle \text{weak-star-topology} = \text{pullback-topology UNIV } (\lambda x \ t. \text{trace } (\text{from-trace-class } t \ o_{CL} \ x)) \ \text{euclidean} \rangle$

$\langle \text{proof} \rangle$

lemma *weak-star-topology-topospace[simp]*:

$\text{topospace weak-star-topology} = \text{UNIV}$

$\langle \text{proof} \rangle$

lemma *weak-star-topology-basis'*:

fixes $f::('a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space})$ **and** $U::'i \Rightarrow \text{complex set}$ **and** $t::'i \Rightarrow ('b, 'a)$
trace-class

assumes $\text{finite } I \wedge i. i \in I \implies \text{open } (U i)$

shows $\text{openin weak-star-topology } \{f. \forall i \in I. \text{trace } (\text{from-trace-class } (t i) o_{CL} f) \in U i\}$

<proof>

lemma *weak-star-topology-basis:*

fixes $f::('a::\text{hilbert-space} \Rightarrow_{CL} 'b::\text{hilbert-space})$ **and** $U::'i \Rightarrow \text{complex set}$ **and** $t::'i \Rightarrow ('b$
 $\Rightarrow_{CL} 'a)$

assumes $\text{finite } I \wedge i. i \in I \implies \text{open } (U i)$

assumes $tc: \langle \wedge i. i \in I \implies \text{trace-class } (t i) \rangle$

shows $\text{openin weak-star-topology } \{f. \forall i \in I. \text{trace } (t i o_{CL} f) \in U i\}$

<proof>

lemma *wot-weaker-than-weak-star:*

continuous-map weak-star-topology cweak-operator-topology $(\lambda f. f)$

<proof>

lemma *wot-weaker-than-weak-star':*

$\langle \text{openin cweak-operator-topology } U \implies \text{openin weak-star-topology } U \rangle$

<proof>

lemma *weak-star-topology-continuous-duality':*

shows *continuous-map weak-star-topology euclidean* $(\lambda x. \text{trace } (\text{from-trace-class } t o_{CL} x))$

<proof>

lemma *weak-star-topology-continuous-duality:*

assumes $\langle \text{trace-class } t \rangle$

shows *continuous-map weak-star-topology euclidean* $(\lambda x. \text{trace } (t o_{CL} x))$

<proof>

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

$\longleftrightarrow (\forall t. \text{trace-class } t \longrightarrow \text{continuous-map } T \text{ euclidean } (\lambda x. \text{trace } (t o_{CL} f x)))$

<proof>

lemma *weak-star-topology-weaker-than-euclidean:*

continuous-map euclidean weak-star-topology $(\lambda f. f)$

<proof>

typedef (**overloaded**) $('a, 'b)$ *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* *<proof>*

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*

<proof>

instantiation *cblinfun-weak-star* :: (*complex-normed-vector*, *complex-normed-vector*) *complex-vector*
begin
lift-definition *scaleC-cblinfun-weak-star* :: $\langle \text{complex} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \rangle$
is scaleC $\langle \text{proof} \rangle$
lift-definition *uminus-cblinfun-weak-star* :: $\langle ('a, 'b) \text{ cblinfun-weak-star} \Rightarrow ('a, 'b) \text{ cblinfun-weak-star} \rangle$
is uminus $\langle \text{proof} \rangle$
lift-definition *zero-cblinfun-weak-star* :: $\langle ('a, 'b) \text{ cblinfun-weak-star} \rangle$ **is 0** $\langle \text{proof} \rangle$
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** $\langle \text{proof} \rangle$
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** $\langle \text{proof} \rangle$
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** $\langle \text{proof} \rangle$
instance
 $\langle \text{proof} \rangle$
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** $\langle \text{openin weak-star-topology} \rangle$ $\langle \text{proof} \rangle$
instance
 $\langle \text{proof} \rangle$
end

lemma *transfer-nhds-weak-star-topology*[*transfer-rule*]:
includes *lifting-syntax*
shows $\langle (\text{cr-cblinfun-weak-star} ==> \text{rel-filter cr-cblinfun-weak-star}) (\text{nhdsin weak-star-topology}) \text{ nhds} \rangle$
 $\langle \text{proof} \rangle$

lemma *limitin-weak-star-topology'*:
 $\langle \text{limitin weak-star-topology } f \text{ l } F \longleftrightarrow (\forall t. ((\lambda j. \text{trace } (\text{from-trace-class } t \text{ o}_{CL} f j)) \longrightarrow \text{trace } (\text{from-trace-class } t \text{ o}_{CL} l)) F) \rangle$
 $\langle \text{proof} \rangle$

lemma *limitin-weak-star-topology*:
 $\langle \text{limitin weak-star-topology } f \text{ l } F \longleftrightarrow (\forall t. \text{trace-class } t \longrightarrow ((\lambda j. \text{trace } (t \text{ o}_{CL} f j)) \longrightarrow \text{trace } (t \text{ o}_{CL} l)) F) \rangle$
 $\langle \text{proof} \rangle$

lemma *filterlim-weak-star-topology*:
 $\langle \text{filterlim } f (\text{nhdsin weak-star-topology } l) = \text{limitin weak-star-topology } f \text{ l} \rangle$
 $\langle \text{proof} \rangle$

lemma *openin-weak-star-topology'*: $\langle \text{openin weak-star-topology } U \longleftrightarrow (\exists V. \text{open } V \wedge U = (\lambda x. \text{trace } (\text{from-trace-class } t \text{ o}_{CL} x)) -' V) \rangle$
 $\langle \text{proof} \rangle$

lemma *hausdorff-weak-star*[simp]: $\langle \text{Hausdorff-space weak-star-topology} \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma *transfer-euclidean-weak-star-topology*[transfer-rule]:
includes *lifting-syntax*
shows $\langle (\text{rel-topology cr-cblinfun-weak-star}) \text{ weak-star-topology euclidean} \rangle$
 $\langle \text{proof} \rangle$

instance *cblinfun-weak-star* :: $(\text{chilbert-space}, \text{chilbert-space}) \text{ t2-space}$
 $\langle \text{proof} \rangle$

lemma *weak-star-topology-plus-cont*: $\langle \text{LIM } (x,y) \text{ nhdsin weak-star-topology } a \times_F \text{ nhdsin weak-star-topology } b. \langle x + y \rangle \text{ nhdsin weak-star-topology } (a + b) \rangle$
 $\langle \text{proof} \rangle$

instance *cblinfun-weak-star* :: $(\text{chilbert-space}, \text{chilbert-space}) \text{ topological-group-add}$
 $\langle \text{proof} \rangle$

lemma *continuous-map-left-comp-weak-star*:
 $\langle \text{continuous-map weak-star-topology weak-star-topology } (\lambda a::'a::\text{chilbert-space} \Rightarrow_{CL} -. b \text{ o}_{CL} a) \rangle$
for $b :: \langle 'b::\text{chilbert-space} \Rightarrow_{CL} 'c::\text{chilbert-space} \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-map-right-comp-weak-star*:
 $\langle \text{continuous-map weak-star-topology weak-star-topology } (\lambda b::'b::\text{chilbert-space} \Rightarrow_{CL} -. b \text{ o}_{CL} a) \rangle$
for $a :: \langle 'a::\text{chilbert-space} \Rightarrow_{CL} 'b::\text{chilbert-space} \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-map-scaleC-weak-star*: $\langle \text{continuous-map weak-star-topology weak-star-topology } (\text{scaleC } c) \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-scaleC-weak-star*: $\langle \text{continuous-on } X \text{ (scaleC } c :: (-,-) \text{ cblinfun-weak-star } \Rightarrow -) \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-star-closure-is-csubspace[simp]*:
fixes $A::('a::\text{chilbert-space}, 'b::\text{chilbert-space}) \text{ cblinfun-weak-star set}$
assumes $\langle \text{csubspace } A \rangle$
shows $\langle \text{csubspace (closure } A) \rangle$
 $\langle \text{proof} \rangle$
include *lattice-syntax*
 $\langle \text{proof} \rangle$

lemma *transfer-csubspace-cblinfun-weak-star[transfer-rule]*:
includes *lifting-syntax*
shows $\langle (\text{rel-set cr-cblinfun-weak-star } ==> (=)) \text{ csubspace csubspace} \rangle$
 $\langle \text{proof} \rangle$

lemma *transfer-closed-cblinfun-weak-star[transfer-rule]*:
includes *lifting-syntax*
shows $\langle (\text{rel-set cr-cblinfun-weak-star } ==> (=)) \text{ (closedin weak-star-topology) closed} \rangle$
 $\langle \text{proof} \rangle$

lemma *transfer-closure-cblinfun-weak-star[transfer-rule]*:
includes *lifting-syntax*
shows $\langle (\text{rel-set cr-cblinfun-weak-star } ==> \text{rel-set cr-cblinfun-weak-star}) \text{ (Abstract-Topology.closure-of weak-star-topology) closure} \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{csubspace (weak-star-topology closure-of } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-closed-weak-star-topology*:
assumes $aA: \langle \bigwedge i. a \ i \in A \rangle$
assumes $\text{closed}: \langle \text{closedin weak-star-topology } A \rangle$
assumes $\text{subspace}: \langle \text{csubspace } A \rangle$
assumes $\text{has-sum}: \langle \bigwedge t. \text{trace-class } t \implies ((\lambda i. \text{trace } (t \ o_{CL} \ a \ i)) \text{ has-sum trace } (t \ o_{CL} \ b)) \ I \rangle$
shows $\langle b \in A \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-in-weak-star*:
 $\langle \text{has-sum-in weak-star-topology } f \ A \ l \longleftrightarrow$
 $(\forall t. \text{trace-class } t \longrightarrow ((\lambda i. \text{trace } (t \ o_{CL} \ f \ i)) \text{ has-sum trace } (t \ o_{CL} \ l)) \ A) \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-butterfly-ket*: $\langle \text{has-sum-in weak-star-topology } (\lambda i. \text{butterfly } (\text{ket } i) (\text{ket } i)) \text{ UNIV } \text{id-cblinfun} \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-weak-star-cont*[simp]:
 $\langle \text{continuous-map weak-star-topology weak-star-topology } (\text{sandwich } A) \rangle$
 $\langle \text{proof} \rangle$

lemma *has-sum-butterfly-ket-a*: $\langle \text{has-sum-in weak-star-topology } (\lambda i. \text{butterfly } (a *_{\mathbb{V}} \text{ket } i) (\text{ket } i)) \text{ UNIV } a \rangle$
 $\langle \text{proof} \rangle$

lemma *finite-rank-weak-star-dense*[simp]: $\langle \text{weak-star-topology closure-of } (\text{Collect finite-rank}) = (\text{UNIV} :: ('a \text{ ell2} \Rightarrow_{\text{CL}} 'b::\text{chilbert-space}) \text{ set}) \rangle$
 $\langle \text{proof} \rangle$

lemma *butterkets-weak-star-dense*[simp]:
 $\langle \text{weak-star-topology closure-of cspan } ((\lambda(\xi,\eta). \text{butterfly } (\text{ket } \xi) (\text{ket } \eta)) ' \text{UNIV}) = \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-star-linear-eq-butterfly-ketI*:
fixes $F G :: ('a \text{ ell2} \Rightarrow_{\text{CL}} 'b \text{ ell2}) \Rightarrow 'c::\text{complex-vector}$
assumes *linear F and linear 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 $F = G$
 $\langle \text{proof} \rangle$

lemma *continuous-map-scaleC-weak-star*[continuous-intros]:
assumes $\langle \text{continuous-map } T \text{ weak-star-topology } f \rangle$
shows $\langle \text{continuous-map } T \text{ weak-star-topology } (\lambda x. \text{scaleC } c (f x)) \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-map-uminus-weak-star*[continuous-intros]:
assumes $\langle \text{continuous-map } T \text{ weak-star-topology } f \rangle$
shows $\langle \text{continuous-map } T \text{ weak-star-topology } (\lambda x. - f x) \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-map-add-weak-star*[continuous-intros]:
assumes $\langle \text{continuous-map } T \text{ weak-star-topology } f \rangle$
assumes $\langle \text{continuous-map } T \text{ weak-star-topology } g \rangle$
shows $\langle \text{continuous-map } T \text{ weak-star-topology } (\lambda x. f x + g x) \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-map-minus-weak-star*[*continuous-intros*]:
assumes $\langle \text{continuous-map } T \text{ weak-star-topology } f \rangle$
assumes $\langle \text{continuous-map } T \text{ weak-star-topology } g \rangle$
shows $\langle \text{continuous-map } T \text{ weak-star-topology } (\lambda x. f x - g x) \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-star-topology-is-norm-topology-fin-dim*[*simp*]:
 $\langle (\text{weak-star-topology} :: ('a::\{\text{cfinite-dim}, \text{chilbert-space}\}) \Rightarrow_{CL} 'b::\{\text{cfinite-dim}, \text{chilbert-space}\})$
 $\text{topology}) = \text{euclidean} \rangle$
 $\langle \text{proof} \rangle$

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*
 $\langle \text{proof} \rangle$

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**
 $\langle \lambda \psi \varphi (i,j). \psi i * \varphi j \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-add1*: $\langle \text{tensor-ell2 } (a + b) c = \text{tensor-ell2 } a c + \text{tensor-ell2 } b c \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-add2*: $\langle \text{tensor-ell2 } a (b + c) = \text{tensor-ell2 } a b + \text{tensor-ell2 } a c \rangle$
<proof>

lemma *tensor-ell2-scaleC1*: $\langle \text{tensor-ell2 } (c *_C a) b = c *_C \text{tensor-ell2 } a b \rangle$
<proof>

lemma *tensor-ell2-scaleC2*: $\langle \text{tensor-ell2 } a (c *_C b) = c *_C \text{tensor-ell2 } a b \rangle$
<proof>

lemma *tensor-ell2-diff1*: $\langle \text{tensor-ell2 } (a - b) c = \text{tensor-ell2 } a c - \text{tensor-ell2 } b c \rangle$
<proof>

lemma *tensor-ell2-diff2*: $\langle \text{tensor-ell2 } a (b - c) = \text{tensor-ell2 } a b - \text{tensor-ell2 } a c \rangle$
<proof>

lemma *tensor-ell2-inner-prod[simp]*: $\langle \text{tensor-ell2 } a b \cdot_C \text{tensor-ell2 } c d = (a \cdot_C c) * (b \cdot_C d) \rangle$
<proof>

lemma *norm-tensor-ell2*: $\langle \text{norm } (a \otimes_s b) = \text{norm } a * \text{norm } b \rangle$
<proof>

lemma *clinear-tensor-ell21*: *clinear* $(\lambda b. a \otimes_s b)$
<proof>

lemma *bounded-clinear-tensor-ell21*: *bounded-clinear* $(\lambda b. a \otimes_s b)$
<proof>

lemma *clinear-tensor-ell22*: *clinear* $(\lambda a. a \otimes_s b)$
<proof>

lemma *bounded-clinear-tensor-ell22*: *bounded-clinear* $(\lambda a. \text{tensor-ell2 } a b)$
<proof>

lemma *tensor-ell2-ket*: *tensor-ell2* $(\text{ket } i) (\text{ket } j) = \text{ket } (i, j)$
<proof>

lemma *tensor-ell2-0-left[simp]*: $\langle 0 \otimes_s x = 0 \rangle$
<proof>

lemma *tensor-ell2-0-right[simp]*: $\langle x \otimes_s 0 = 0 \rangle$
<proof>

lemma *tensor-ell2-sum-left*: $\langle (\sum x \in X. a x) \otimes_s b = (\sum x \in X. a x \otimes_s b) \rangle$
<proof>

lemma *tensor-ell2-sum-right*: $\langle a \otimes_s (\sum x \in X. b x) = (\sum x \in X. a \otimes_s b x) \rangle$
<proof>

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 $\langle \text{closure } (\text{cspan } \{a \otimes_s b \mid a \in S \wedge b \in T\}) = \text{UNIV} \rangle$

$\langle \text{proof} \rangle$

definition *assoc-ell2* $:: \langle (('a \times 'b) \times 'c) \text{ ell2} \Rightarrow_{CL} ('a \times ('b \times 'c)) \text{ ell2} \rangle$ **where**

$\langle \text{assoc-ell2} = \text{classical-operator } (\text{Some } o \ (\lambda((a,b),c). (a,(b,c)))) \rangle$

lemma *unitary-assoc-ell2[simp]*: $\langle \text{unitary } \text{assoc-ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *assoc-ell2-tensor*: $\langle \text{assoc-ell2} *_{\mathcal{V}} ((a \otimes_s b) \otimes_s c) = (a \otimes_s (b \otimes_s c)) \rangle$

$\langle \text{proof} \rangle$

lemma *assoc-ell2'-tensor*: $\langle \text{assoc-ell2} *_{\mathcal{V}} \text{tensor-ell2 } a \ (\text{tensor-ell2 } b \ c) = \text{tensor-ell2 } (\text{tensor-ell2 } a \ b) \ c \rangle$

$\langle \text{proof} \rangle$

lemma *assoc-ell2'-inv*: $\text{assoc-ell2 } o_{CL} \ \text{assoc-ell2} * = \text{id-cblinfun}$

$\langle \text{proof} \rangle$

lemma *assoc-ell2-inv*: $\text{assoc-ell2} * \ o_{CL} \ \text{assoc-ell2} = \text{id-cblinfun}$

$\langle \text{proof} \rangle$

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]*: $\langle \text{unitary } \text{swap-ell2} \rangle$

$\langle \text{proof} \rangle$

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** $b :: \langle 'b \text{ ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *swap-ell2-ket[simp]*: $\langle (\text{swap-ell2} :: ('a \times 'b) \text{ ell2} \Rightarrow_{CL} -) *_{\mathcal{V}} \text{ket } (x,y) = \text{ket } (y,x) \rangle$

$\langle \text{proof} \rangle$

lemma *adjoint-swap-ell2[simp]*: $\langle \text{swap-ell2} * = \text{swap-ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-ell2-extensionality*:

assumes $(\bigwedge s \ t. a *_{\mathcal{V}} (s \otimes_s t) = b *_{\mathcal{V}} (s \otimes_s t))$

shows $a = b$

$\langle \text{proof} \rangle$

lemma *tensor-ell2-nonzero*: $\langle a \otimes_s b \neq 0 \rangle$ **if** $\langle a \neq 0 \rangle$ **and** $\langle b \neq 0 \rangle$

$\langle \text{proof} \rangle$

lemma *swap-ell2-selfinv[simp]*: $\langle \text{swap-ell2 } o_{CL} \text{ swap-ell2} = \text{id-cblinfun} \rangle$
 $\langle \text{proof} \rangle$

lemma *bounded-cbilinear-tensor-ell2[bounded-cbilinear]*: $\langle \text{bounded-cbilinear } (\otimes_s) \rangle$
 $\langle \text{proof} \rangle$

lemma *ket-pair-split*: $\langle \text{ket } x = \text{tensor-ell2 } (\text{ket } (\text{fst } x)) (\text{ket } (\text{snd } x)) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-is-ortho-set*:
assumes $\langle \text{is-ortho-set } A \rangle \langle \text{is-ortho-set } B \rangle$
shows $\langle \text{is-ortho-set } \{a \otimes_s b \mid a \in A \wedge b \in B\} \rangle$
 $\langle \text{proof} \rangle$

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**
 $\langle \text{ccspan } B = \top \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-is-onb*:
assumes $\langle \text{is-onb } A \rangle \langle \text{is-onb } B \rangle$
shows $\langle \text{is-onb } \{a \otimes_s b \mid a \in A \wedge b \in B\} \rangle$
 $\langle \text{proof} \rangle$

lemma *continuous-tensor-ell2*: $\langle \text{continuous-on UNIV } (\lambda(x::'a \text{ ell2}, y::'b \text{ ell2}). x \otimes_s y) \rangle$
 $\langle \text{proof} \rangle$

lemma *summable-on-tensor-ell2-right*: $\langle \varphi \text{ summable-on } A \implies (\lambda x. \psi \otimes_s \varphi x) \text{ summable-on } A \rangle$
 $\langle \text{proof} \rangle$

lemma *summable-on-tensor-ell2-left*: $\langle \varphi \text{ summable-on } A \implies (\lambda x. \varphi x \otimes_s \psi) \text{ summable-on } A \rangle$
 $\langle \text{proof} \rangle$

lift-definition *tensor-ell2-left* :: $\langle 'a \text{ ell2} \Rightarrow ('b \text{ ell2} \Rightarrow_{CL} ('a \times 'b) \text{ ell2}) \rangle$ **is**
 $\langle \lambda \psi \varphi. \psi \otimes_s \varphi \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-left-apply[simp]*: $\langle \text{tensor-ell2-left } \psi *_V \varphi = \psi \otimes_s \varphi \rangle$
 $\langle \text{proof} \rangle$

lift-definition *tensor-ell2-right* :: $\langle 'a \text{ ell2} \Rightarrow ('b \text{ ell2} \Rightarrow_{CL} ('b \times 'a) \text{ ell2}) \rangle$ **is**
 $\langle \lambda \psi \varphi. \varphi \otimes_s \psi \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-right-apply[simp]*: $\langle \text{tensor-ell2-right } \psi *_V \varphi = \varphi \otimes_s \psi \rangle$
 $\langle \text{proof} \rangle$

lemma *isometry-tensor-ell2-right*: $\langle \text{isometry } (\text{tensor-ell2-right } \psi) \rangle$ **if** $\langle \text{norm } \psi = 1 \rangle$
 $\langle \text{proof} \rangle$

lemma *isometry-tensor-ell2-left*: $\langle \text{isometry } (\text{tensor-ell2-left } \psi) \rangle$ **if** $\langle \text{norm } \psi = 1 \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-right-scale*: $\langle \text{tensor-ell2-right } (a *_C \psi) = a *_C \text{tensor-ell2-right } \psi \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-left-scale*: $\langle \text{tensor-ell2-left } (a *_C \psi) = a *_C \text{tensor-ell2-left } \psi \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-right-0[simp]*: $\langle \text{tensor-ell2-right } 0 = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-left-0[simp]*: $\langle \text{tensor-ell2-left } 0 = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-right-adj-apply[simp]*: $\langle (\text{tensor-ell2-right } \psi^*) *_V (\alpha \otimes_s \beta) = (\psi \cdot_C \beta) *_C \alpha \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-left-adj-apply[simp]*: $\langle (\text{tensor-ell2-left } \psi^*) *_V (\alpha \otimes_s \beta) = (\psi \cdot_C \alpha) *_C \beta \rangle$
 $\langle \text{proof} \rangle$

lemma *infsun-tensor-ell2-right*: $\langle \psi \otimes_s (\sum_{\infty x \in A} \varphi x) = (\sum_{\infty x \in A} \psi \otimes_s \varphi x) \rangle$
 $\langle \text{proof} \rangle$

lemma *infsun-tensor-ell2-left*: $\langle (\sum_{\infty x \in A} \varphi x) \otimes_s \psi = (\sum_{\infty x \in A} \varphi x \otimes_s \psi) \rangle$
 $\langle \text{proof} \rangle$

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 $a = b$

$\langle \text{proof} \rangle$

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 $\langle A = B \rangle$

$\langle \text{proof} \rangle$

lemma *unitary-tensor-ell2-right-CARD-1*:

fixes $\psi :: \langle 'a :: \{ \text{CARD-1}, \text{enum} \} \text{ell2} \rangle$

assumes $\langle \text{norm } \psi = 1 \rangle$

shows $\langle \text{unitary } (\text{tensor-ell2-right } \psi) \rangle$

$\langle \text{proof} \rangle$

14.2 Tensor product of operators on - ell2

definition *tensor-op* :: $\langle ('a \text{ ell2}, 'b \text{ ell2}) \text{cblinfun} \Rightarrow ('c \text{ ell2}, 'd \text{ ell2}) \text{cblinfun}$

$\Rightarrow (('a \times 'c) \text{ ell2}, ('b \times 'd) \text{ ell2}) \text{cblinfun} \rangle$ (**infixr** \otimes_o 70) **where**

$\langle \text{tensor-op } M N = \text{cblinfun-extension } (\text{range ket}) (\lambda k. \text{case } (\text{inv ket } k) \text{ of } (x, y) \Rightarrow \text{tensor-ell2}$

$(M *_V \text{ket } x) (N *_V \text{ket } y)\rangle$

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*: $\langle \text{norm } (M \otimes_o N) = \text{norm } M * \text{norm } N \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-ket*: $\langle \text{tensor-op } M N *_V (\text{ket } (a,c)) = \text{tensor-ell2 } (M *_V \text{ket } a) (N *_V \text{ket } c) \rangle$

$\langle \text{proof} \rangle$

lemma *comp-tensor-op*: $\langle \text{tensor-op } a b \text{ } o_{CL} (\text{tensor-op } c d) = \text{tensor-op } (a \text{ } o_{CL} c) (b \text{ } o_{CL} d) \rangle$

for $a :: \langle 'e \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$ **and** $b :: \langle 'f \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$ **and**

$c :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'e \text{ ell2} \rangle$ **and** $d :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'f \text{ ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-left-add*: $\langle (x + y) \otimes_o b = x \otimes_o b + y \otimes_o b \rangle$

for $x y :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$ **and** $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-right-add*: $\langle b \otimes_o (x + y) = b \otimes_o x + b \otimes_o y \rangle$

for $x y :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$ **and** $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-scaleC-left*: $\langle (c *_C x) \otimes_o b = c *_C (x \otimes_o b) \rangle$

for $x :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$ **and** $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-scaleC-right*: $\langle b \otimes_o (c *_C x) = c *_C (b \otimes_o x) \rangle$

for $x :: \langle 'a \text{ ell2} \Rightarrow_{CL} 'c \text{ ell2} \rangle$ **and** $b :: \langle 'b \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-bounded-cbilinear[simp]*: $\langle \text{bounded-cbilinear tensor-op} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-cbilinear[simp]*: $\langle \text{cbilinear tensor-op} \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-butter*: $\langle \text{butterfly } (\text{ket } i) (\text{ket } j) \otimes_o \text{butterfly } (\text{ket } k) (\text{ket } l) = \text{butterfly } (\text{ket } (i,k))$

$(\text{ket } (j,l)) \rangle$

$\langle \text{proof} \rangle$

lemma *cspan-tensor-op-butter*: $\langle \text{cspan } \{ \text{tensor-op } (\text{butterfly } (\text{ket } i) (\text{ket } j)) (\text{butterfly } (\text{ket } k) (\text{ket } l)) \mid (i::\text{finite}) (j::\text{finite}) (k::\text{finite}) (l::\text{finite}). \text{True} \} = \text{UNIV} \rangle$

$\langle \text{proof} \rangle$

lemma *cindependent-tensor-op-butter*: $\langle \text{cindependent } \{ \text{tensor-op } (\text{butterfly } (\text{ket } i) (\text{ket } j)) (\text{butterfly } (\text{ket } k) (\text{ket } l)) \mid (i::\text{finite}) (j::\text{finite}) (k::\text{finite}) (l::\text{finite}). \text{True} \} = \text{UNIV} \rangle$

$(ket\ k)\ (ket\ l) \mid i\ j\ k\ l.\ True\}$
 $\langle proof \rangle$

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
 $\langle \lambda a.\ a \otimes_o id\text{-cblinfun} \rangle$
 $\langle proof \rangle$

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
 $\langle \lambda a.\ id\text{-cblinfun} \otimes_o a \rangle$
 $\langle proof \rangle$

lemma *sandwich-tensor-ell2-right*: $\langle sandwich\ (tensor\text{-ell2}\text{-right}\ \psi^*)\ *_V\ a \otimes_o b = (\psi \cdot_C (b *_V \psi)) *_C a \rangle$
 $\langle proof \rangle$

lemma *sandwich-tensor-ell2-left*: $\langle sandwich\ (tensor\text{-ell2}\text{-left}\ \psi^*)\ *_V\ a \otimes_o b = (\psi \cdot_C (a *_V \psi)) *_C b \rangle$
 $\langle proof \rangle$

lemma *tensor-op-adjoint*: $\langle (tensor\text{-op}\ a\ b)^* = tensor\text{-op}\ (a^*)\ (b^*) \rangle$
 $\langle proof \rangle$

lemma *has-sum-id-tensor-butterfly-ket*: $\langle ((\lambda i.\ (id\text{-cblinfun} \otimes_o butterfly\ (ket\ i)\ (ket\ i)) *_V \psi)\ has\text{-sum}\ \psi)\ UNIV \rangle$
 $\langle proof \rangle$

lemma *tensor-op-dense*: $\langle cstrong\text{-operator}\text{-topology}\ closure\text{-of}\ (cspan\ \{a \otimes_o b \mid a\ b.\ True\}) = UNIV \rangle$
— [7, p.185 (10)], but we prove it directly.
 $\langle proof \rangle$

lemma *tensor-extensionality-finite*:
fixes $F\ G :: \langle (((('a::finite \times 'b::finite)\ ell2) \Rightarrow_{CL}\ (('c::finite \times 'd::finite)\ ell2)) \Rightarrow 'e::complex\text{-vector}) \rangle$
assumes $[simp]:\ clinear\ F\ clinear\ G$
assumes *tensor-eq*: $(\bigwedge a\ b.\ F\ (tensor\text{-op}\ a\ b) = G\ (tensor\text{-op}\ a\ b))$
shows $F = G$
 $\langle proof \rangle$

lemma *tensor-id[simp]*: $\langle tensor\text{-op}\ id\text{-cblinfun}\ id\text{-cblinfun} = id\text{-cblinfun} \rangle$
 $\langle proof \rangle$

lemma *tensor-butterfly*: $tensor\text{-op}\ (butterfly\ \psi\ \psi')\ (butterfly\ \varphi\ \varphi') = butterfly\ (tensor\text{-ell2}\ \psi\ \varphi)\ (tensor\text{-ell2}\ \psi'\ \varphi')$
 $\langle proof \rangle$

definition *tensor-lift* :: $\langle ('a1::\text{finite ell2} \Rightarrow_{CL} 'a2::\text{finite ell2}) \Rightarrow ('b1::\text{finite ell2} \Rightarrow_{CL} 'b2::\text{finite ell2}) \Rightarrow 'c \rangle$
 $\Rightarrow \langle (('a1 \times 'b1) \text{ ell2} \Rightarrow_{CL} ('a2 \times 'b2) \text{ ell2}) \Rightarrow 'c::\text{complex-normed-vector} \rangle$

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 *cbilinear 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>

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>

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>

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>

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>

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>

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>

lemma *tensor-op-almost-injective*:

fixes *a c* :: $\langle 'a \text{ ell2} \Rightarrow_{CL} 'b \text{ ell2} \rangle$

and *b d* :: $\langle 'c \text{ ell2} \Rightarrow_{CL} 'd \text{ ell2} \rangle$

assumes $\langle \text{tensor-op } a b = \text{tensor-op } c d \rangle$

assumes $\langle a \neq 0 \rangle$

shows $\langle \exists \gamma. b = \gamma *_C d \rangle$
 $\langle \text{proof} \rangle$

lemma *linear-tensor-left[simp]*: $\langle \text{linear } (\lambda a. a \otimes_o b :: - \text{ell2} \Rightarrow_{CL} - \text{ell2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *linear-tensor-right[simp]*: $\langle \text{linear } (\lambda b. a \otimes_o b :: - \text{ell2} \Rightarrow_{CL} - \text{ell2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-op-0-left[simp]*: $\langle \text{tensor-op } 0 \ x = (0 :: ('a*'b) \text{ell2} \Rightarrow_{CL} ('c*'d) \text{ell2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-op-0-right[simp]*: $\langle \text{tensor-op } x \ 0 = (0 :: ('a*'b) \text{ell2} \Rightarrow_{CL} ('c*'d) \text{ell2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *bij-tensor-ell2-one-dim-left*:

assumes $\langle \psi \neq 0 \rangle$

shows $\langle \text{bij } (\lambda x::'b \text{ell2}. (\psi :: 'a::\text{CARD-1} \text{ell2}) \otimes_s x) \rangle$

$\langle \text{proof} \rangle$

lemma *bij-tensor-op-one-dim-left*:

fixes $a :: \langle 'a::\{\text{CARD-1}, \text{enum}\} \text{ell2} \Rightarrow_{CL} 'b::\{\text{CARD-1}, \text{enum}\} \text{ell2} \rangle$

assumes $\langle a \neq 0 \rangle$

shows $\langle \text{bij } (\lambda x::'c \text{ell2} \Rightarrow_{CL} 'd \text{ell2}. a \otimes_o x) \rangle$

$\langle \text{proof} \rangle$

lemma *bij-tensor-op-one-dim-right*:

assumes $\langle b \neq 0 \rangle$

shows $\langle \text{bij } (\lambda x::'c \text{ell2} \Rightarrow_{CL} 'd \text{ell2}. x \otimes_o (b :: 'a::\{\text{CARD-1}, \text{enum}\} \text{ell2} \Rightarrow_{CL} 'b::\{\text{CARD-1}, \text{enum}\} \text{ell2})) \rangle$

(**is** $\langle \text{bij } ?f \rangle$)

$\langle \text{proof} \rangle$

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 $\text{eq: } \langle \text{butterfly } \psi \ \psi' \otimes_o a23 = \text{assoc-ell2 } o_{CL} (b12 \otimes_o \text{butterfly } \varphi \ \varphi') o_{CL} \text{assoc-ell2*} \rangle$

assumes $\langle \psi \neq 0 \rangle \langle \psi' \neq 0 \rangle \langle \varphi \neq 0 \rangle \langle \varphi' \neq 0 \rangle$

shows $\langle \exists c. \text{butterfly } \psi \ \psi' \otimes_o a23 = \text{butterfly } \psi \ \psi' \otimes_o c \otimes_o \text{butterfly } \varphi \ \varphi' \rangle$

$\langle \text{proof} \rangle$

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]

$\langle \text{proof} \rangle$

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>

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>

lemma *swap-tensor-op[simp]*: $\langle \text{swap-ell2 } o_{CL} (a \otimes_o b) o_{CL} \text{ swap-ell2} = b \otimes_o a \rangle$
<proof>

lemma *swap-tensor-op-sandwich[simp]*: $\langle \text{sandwich swap-ell2 } (a \otimes_o b) = b \otimes_o a \rangle$
<proof>

lemma *swap-ell2-commute-tensor-op*:
 $\langle \text{swap-ell2 } o_{CL} (a \otimes_o b) = (b \otimes_o a) o_{CL} \text{ swap-ell2} \rangle$
<proof>

lemma *trace-class-tensor-op-swap*: $\langle \text{trace-class } (a \otimes_o b) \longleftrightarrow \text{trace-class } (b \otimes_o a) \rangle$
<proof>

lemma *trace-class-tensor-iff*: $\langle \text{trace-class } (a \otimes_o b) \longleftrightarrow (\text{trace-class } a \wedge \text{trace-class } b) \vee a = 0 \vee b = 0 \rangle$
<proof>

lemma *trace-tensor*: $\langle \text{trace } (a \otimes_o b) = \text{trace } a * \text{trace } b \rangle$
— [8, Lemma 32]
<proof>

lemma *isometry-tensor-op*: $\langle \text{isometry } (U \otimes_o V) \rangle$ **if** $\langle \text{isometry } U \rangle$ **and** $\langle \text{isometry } V \rangle$
<proof>

lemma *is-Proj-tensor-op*: $\langle \text{is-Proj } a \implies \text{is-Proj } b \implies \text{is-Proj } (a \otimes_o b) \rangle$
<proof>

lemma *isometry-tensor-id-right[simp]*:
fixes $U :: \langle 'a \text{ ell2 } \Rightarrow_{CL} 'b \text{ ell2} \rangle$
shows $\langle \text{isometry } (U \otimes_o (\text{id-cblinfun} :: 'c \text{ ell2 } \Rightarrow_{CL} -)) \longleftrightarrow \text{isometry } U \rangle$
<proof>

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>

lemma *unitary-tensor-id-right[simp]*: $\langle \text{unitary } (U \otimes_o \text{id-cblinfun}) \longleftrightarrow \text{unitary } U \rangle$
<proof>

lemma *unitary-tensor-id-left[simp]*: $\langle \text{unitary } (\text{id-cblinfun} \otimes_o U) \longleftrightarrow \text{unitary } U \rangle$

<proof>

lemma sandwich-tensor-op: *<sandwich (a \otimes_o b) (c \otimes_o d) = sandwich a c \otimes_o sandwich b d>*
<proof>

lemma sandwich-assoc-ell2-tensor-op[simp]: *<sandwich assoc-ell2 ((a \otimes_o b) \otimes_o c) = a \otimes_o (b \otimes_o c)>*
<proof>

lemma unitary-tensor-op: *<unitary (a \otimes_o b)>* **if** [simp]: *<unitary a>* *<unitary b>*
<proof>

lemma tensor-ell2-right-butterfly: *<tensor-ell2-right ψ o_{CL} tensor-ell2-right φ^* = id-cblinfun \otimes_o butterfly ψ φ >*
<proof>

lemma tensor-ell2-left-butterfly: *<tensor-ell2-left ψ o_{CL} tensor-ell2-left φ^* = butterfly ψ φ \otimes_o id-cblinfun>*
<proof>

lift-definition tc-tensor :: *<'a ell2, 'b ell2>* *trace-class* \Rightarrow *'c ell2, 'd ell2>* *trace-class* \Rightarrow
(('a \times 'c) ell2, ('b \times 'd) ell2) trace-class> **is**
tensor-op
<proof>

lemma trace-norm-tensor: *<trace-norm (a \otimes_o b) = trace-norm a * trace-norm b>*
<proof>

lemma bounded-cbilinear-tc-tensor: *<bounded-cbilinear tc-tensor>*
<proof>

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: *<tc-tensor (c $*_C$ a) b = c $*_C$ tc-tensor a b>*
<proof>

lemma tc-tensor-scaleC-right: *<tc-tensor a (c $*_C$ b) = c $*_C$ tc-tensor a b>*
<proof>

lemma comp-tc-tensor: *<tc-compose (tc-tensor a b) (tc-tensor c d) = tc-tensor (tc-compose a c) (tc-compose b d)>*
<proof>

lemma norm-tc-tensor: *<norm (tc-tensor a b) = norm a * norm b>*
<proof>

lemma *tc-tensor-pos*: $\langle tc\text{-tensor } a \ b \geq 0 \rangle$ **if** $\langle a \geq 0 \rangle$ **and** $\langle b \geq 0 \rangle$
for $a :: \langle ('a \ ell2, 'a \ ell2) \ trace\text{-class} \rangle$ **and** $b :: \langle ('b \ ell2, 'b \ ell2) \ trace\text{-class} \rangle$
 $\langle proof \rangle$

interpretation *tensor-op-cbilinear*: *bounded-cbilinear tensor-op*
 $\langle proof \rangle$

lemma *tensor-op-mono-left*:
fixes $a :: \langle 'a \ ell2 \Rightarrow_{CL} 'a \ ell2 \rangle$ **and** $c :: \langle 'b \ ell2 \Rightarrow_{CL} 'b \ ell2 \rangle$
assumes $\langle a \leq b \rangle$ **and** $\langle c \geq 0 \rangle$
shows $\langle a \otimes_o c \leq b \otimes_o c \rangle$
 $\langle proof \rangle$

lemma *tensor-op-mono-right*:
fixes $a :: \langle 'a \ ell2 \Rightarrow_{CL} 'a \ ell2 \rangle$ **and** $b :: \langle 'b \ ell2 \Rightarrow_{CL} 'b \ ell2 \rangle$
assumes $\langle b \leq c \rangle$ **and** $\langle a \geq 0 \rangle$
shows $\langle a \otimes_o b \leq a \otimes_o c \rangle$
 $\langle proof \rangle$

lemma *tensor-op-mono*:
fixes $a :: \langle 'a \ ell2 \Rightarrow_{CL} 'a \ ell2 \rangle$ **and** $c :: \langle 'b \ ell2 \Rightarrow_{CL} 'b \ ell2 \rangle$
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 $\langle a \otimes_o c \leq b \otimes_o d \rangle$
 $\langle proof \rangle$

lemma *sandwich-tc-tensor*: $\langle sandwich\text{-tc } (E \otimes_o F) (tc\text{-tensor } t \ u) = tc\text{-tensor } (sandwich\text{-tc } E \ t) (sandwich\text{-tc } F \ u) \rangle$
 $\langle proof \rangle$

lemma *tensor-tc-butterfly*: $tc\text{-tensor } (tc\text{-butterfly } \psi \ \psi') (tc\text{-butterfly } \varphi \ \varphi') = tc\text{-butterfly } (tensor\text{-ell2 } \psi \ \varphi) (tensor\text{-ell2 } \psi' \ \varphi')$
 $\langle proof \rangle$

lemma *separating-set-bounded-clinear-tc-tensor*:
shows $\langle separating\text{-set } bounded\text{-clinear } ((\lambda(\varrho, \sigma). tc\text{-tensor } \varrho \ \sigma) \ ' (UNIV \times UNIV)) \rangle$
 $\langle proof \rangle$

lemma *separating-set-bounded-clinear-tc-tensor-nested*:
assumes $\langle separating\text{-set } (bounded\text{-clinear } :: (- \Rightarrow 'e::complex\text{-normed-vector}) \Rightarrow -) \ A \rangle$
assumes $\langle separating\text{-set } (bounded\text{-clinear } :: (- \Rightarrow 'e::complex\text{-normed-vector}) \Rightarrow -) \ B \rangle$
shows $\langle separating\text{-set } (bounded\text{-clinear } :: (- \Rightarrow 'e::complex\text{-normed-vector}) \Rightarrow -) \ ((\lambda(\varrho, \sigma). tc\text{-tensor } \varrho \ \sigma) \ ' (A \times B)) \rangle$
 $\langle proof \rangle$

lemma *tc-tensor-0-left[simp]*: $\langle tc\text{-tensor } 0 \ x = 0 \rangle$
 $\langle proof \rangle$

lemma *tc-tensor-0-right[simp]*: $\langle tc\text{-tensor } x \ 0 = 0 \rangle$
 $\langle proof \rangle$

14.3 Tensor product of subspaces

definition *tensor-ccsubspace* (**infixr** \otimes_S 70) **where**
 $\langle tensor\text{-ccsubspace } A \ B = ccspace \ \{ \psi \otimes_s \varphi \mid \psi \ \varphi. \ \psi \in space\text{-as-set } A \ \wedge \ \varphi \in space\text{-as-set } B \} \rangle$

lemma *tensor-ccsubspace-via-Proj*: $\langle A \otimes_S B = (Proj \ A \ \otimes_o \ Proj \ B) \ *_S \ \top \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-top[simp]*: $\langle \top \otimes_S \top = \top \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-0-left[simp]*: $\langle 0 \otimes_S X = 0 \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-0-right[simp]*: $\langle X \otimes_S 0 = 0 \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-image*: $\langle (A \ *_S \ T) \otimes_S (B \ *_S \ U) = (A \ \otimes_o \ B) \ *_S \ (T \ \otimes_S \ U) \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-bot-left[simp]*: $\langle \perp \otimes_S S = \perp \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-bot-right[simp]*: $\langle S \otimes_S \perp = \perp \rangle$
 $\langle proof \rangle$

lemma *swap-ell2-tensor-ccsubspace*: $\langle swap\text{-ell2} \ *_S \ (S \ \otimes_S \ T) = T \ \otimes_S \ S \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-right1dim-member*:
assumes $\langle \psi \in space\text{-as-set } (S \ \otimes_S \ ccspace\{\varphi\}) \rangle$
shows $\langle \exists \psi'. \ \psi = \psi' \otimes_s \varphi \rangle$
 $\langle proof \rangle$

lemma *tensor-ccsubspace-left1dim-member*:
assumes $\langle \psi \in space\text{-as-set } (ccspace\{\varphi\} \ \otimes_S \ S) \rangle$
shows $\langle \exists \psi'. \ \psi = \varphi \otimes_s \psi' \rangle$
 $\langle proof \rangle$

lemma *tensor-ell2-mem-tensor-ccsubspace-left*:
assumes $\langle a \ \otimes_s \ b \in space\text{-as-set } (S \ \otimes_S \ T) \rangle$ **and** $\langle b \neq 0 \rangle$
shows $\langle a \in space\text{-as-set } S \rangle$
 $\langle proof \rangle$

lemma *tensor-ell2-mem-tensor-ccsubspace-right*:
assumes $\langle a \otimes_s b \in \text{space-as-set } (S \otimes_S T) \rangle$ **and** $\langle a \neq 0 \rangle$
shows $\langle b \in \text{space-as-set } T \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-in-tensor-ccsubspace*: $\langle a \otimes_s b \in \text{space-as-set } (A \otimes_S B) \rangle$ **if** $\langle a \in \text{space-as-set } A \rangle$ **and** $\langle b \in \text{space-as-set } B \rangle$
— Converse is *tensor-ell2-mem-tensor-ccsubspace-left* and *...-right*.
 $\langle \text{proof} \rangle$

lemma *tensor-ccsubspace-INF-left-top*:
fixes $S :: \langle 'a \Rightarrow 'b \text{ ell2 ccspace} \rangle$
shows $\langle (\text{INF } x \in X. S x) \otimes_S (\top :: 'c \text{ ell2 ccspace}) = (\text{INF } x \in X. S x \otimes_S \top) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ccsubspace-INF-right-top*:
fixes $S :: \langle 'a \Rightarrow 'b \text{ ell2 ccspace} \rangle$
shows $\langle (\top :: 'c \text{ ell2 ccspace}) \otimes_S (\text{INF } x \in X. S x) = (\text{INF } x \in X. \top \otimes_S S x) \rangle$
 $\langle \text{proof} \rangle$

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** $\langle X \neq \{\} \rangle$
 $\langle \text{proof} \rangle$

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** $\langle X \neq \{\} \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ccsubspace-ccspan*: $\langle \text{ccspan } X \otimes_S \text{ccspan } Y = \text{ccspan } \{x \otimes_s y \mid x y. x \in X \wedge y \in Y\} \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ccsubspace-mono*: $\langle A \otimes_S B \leq C \otimes_S D \rangle$ **if** $\langle A \leq C \rangle$ **and** $\langle B \leq D \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \exists \varphi \delta. (\forall n :: \text{nat}. \varphi n \in \text{space-as-set } A) \wedge (\forall n. \delta n \in \text{space-as-set } B) \wedge ((\lambda n. \varphi n \otimes_s \delta n) \text{ has-sum } \psi) \text{ UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *ortho-tensor-ccsubspace-right*: $\langle - (\top \otimes_S A) = \top \otimes_S (- A) \rangle$
 $\langle \text{proof} \rangle$

lemma *ortho-tensor-ccsubspace-left*: $\langle - (A \otimes_S \top) = (- A) \otimes_S \top \rangle$
 $\langle \text{proof} \rangle$

lemma *kernel-tensor-id-left*: $\langle \text{kernel } (id\text{-cblinfun } \otimes_o A) = \top \otimes_S \text{kernel } A \rangle$
 $\langle \text{proof} \rangle$

lemma *kernel-tensor-id-right*: $\langle \text{kernel } (A \otimes_o id\text{-cblinfun}) = \text{kernel } A \otimes_S \top \rangle$
 $\langle \text{proof} \rangle$

lemma *eigenspace-tensor-id-left*: $\langle \text{eigenspace } c (id\text{-cblinfun } \otimes_o A) = \top \otimes_S \text{eigenspace } c A \rangle$
 $\langle \text{proof} \rangle$

lemma *eigenspace-tensor-id-right*: $\langle \text{eigenspace } c (A \otimes_o id\text{-cblinfun}) = \text{eigenspace } c A \otimes_S \top \rangle$
 $\langle \text{proof} \rangle$

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* :: $\langle ('a \times 'c) \text{ ell2}, ('b \times 'c) \text{ ell2} \rangle \text{ trace-class} \Rightarrow ('a \text{ ell2}, 'b \text{ ell2})$
 $\text{trace-class} \rangle$ **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).

$\langle \text{proof} \rangle$

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*: $\langle \text{norm } (\text{partial-trace } t) \leq \text{norm } t \rangle$

$\langle \text{proof} \rangle$

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)⟩
 ⟨proof⟩

lemma *trace-partial-trace-compose-eq-trace-compose-tensor-id*:

⟨trace (from-trace-class (partial-trace t) o_{CL} x) = trace (from-trace-class t o_{CL} (x ⊗_o id-cblinfun))⟩
 ⟨proof⟩

lemma *right-amplification-weak-star-cont[simp]*:

⟨continuous-map weak-star-topology weak-star-topology (λa. a ⊗_o id-cblinfun)⟩
 — Logically does not belong in this theory but uses the partial trace in the proof.
 ⟨proof⟩

lemma *left-amplification-weak-star-cont[simp]*:

⟨continuous-map weak-star-topology weak-star-topology (λb. id-cblinfun ⊗_o b :: ('c × 'a) ell2 ⇒_{CL} ('c × 'b) ell2)⟩
 — Logically does not belong in this theory but uses the partial trace in the proof.
 ⟨proof⟩

lemma *partial-trace-plus*: ⟨partial-trace (t + u) = partial-trace t + partial-trace u⟩

⟨proof⟩

lemma *partial-trace-scaleC*: ⟨partial-trace (c *_C t) = c *_C partial-trace t⟩

⟨proof⟩

lemma *partial-trace-tensor*: ⟨partial-trace (tc-tensor t u) = trace-tc u *_C t⟩

⟨proof⟩

lemma *bounded-clinear-partial-trace[bounded-clinear, iff]*: ⟨bounded-clinear partial-trace⟩

⟨proof⟩

lemma *vector-sandwich-partial-trace-has-sum*:

⟨((λz. ((x ⊗_s ket z) •_C (from-trace-class ρ *_V (y ⊗_s ket z))))
 has-sum x •_C (from-trace-class (partial-trace ρ) *_V y)) UNIV)⟩
 ⟨proof⟩

lemma *vector-sandwich-partial-trace*:

⟨x •_C (from-trace-class (partial-trace ρ) *_V y) =
 (∑_{∞ z}. ((x ⊗_s ket z) •_C (from-trace-class ρ *_V (y ⊗_s ket z))))⟩
 ⟨proof⟩

unbundle *no cblinfun-syntax*

end

16 Von-Neumann-Algebras – Von Neumann algebras and the double commutant theorem

theory *Von-Neumann-Algebras*
imports *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 *sandwich-unitary-commutant*:

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

assumes [*simp*]: $\langle \text{unitary } U \rangle$

shows $\langle \text{sandwich } U \text{ ' } \text{commutant } X = \text{commutant } (\text{sandwich } U \text{ ' } X) \rangle$

$\langle \text{proof} \rangle$

lemma *commutant-tensor1*: $\langle \text{commutant } (\text{range } (\lambda a. a \otimes_o \text{id-cblinfun})) = \text{range } (\lambda b. \text{id-cblinfun} \otimes_o b) \rangle$

$\langle \text{proof} \rangle$

lemma *csubspace-commutant*[*simp*]: $\langle \text{csubspace } (\text{commutant } X) \rangle$

$\langle \text{proof} \rangle$

lemma *closed-commutant*[*simp*]: $\langle \text{closed } (\text{commutant } X) \rangle$

$\langle \text{proof} \rangle$

lemma *closed-csubspace-commutant*[*simp*]: $\langle \text{closed-csubspace } (\text{commutant } X) \rangle$

$\langle \text{proof} \rangle$

lemma *commutant-mult*: $\langle a \circ_{CL} b \in \text{commutant } X \rangle$ **if** $\langle a \in \text{commutant } X \rangle$ **and** $\langle b \in \text{commutant } X \rangle$

$\langle \text{proof} \rangle$

lemma *double-commutant-grows*[*simp*]: $\langle X \subseteq \text{commutant } (\text{commutant } X) \rangle$

$\langle \text{proof} \rangle$

lemma *commutant-antimono*: $\langle X \subseteq Y \implies \text{commutant } X \supseteq \text{commutant } Y \rangle$

$\langle \text{proof} \rangle$

lemma *triple-commutant[simp]*: $\langle \text{commutant } (\text{commutant } (\text{commutant } X)) = \text{commutant } X \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-adj*: $\langle \text{adj } ' \text{commutant } X = \text{commutant } (\text{adj } ' X) \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-empty[simp]*: $\langle \text{commutant } \{\} = \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-weak-star-closed[simp]*: $\langle \text{closedin weak-star-topology } (\text{commutant } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *cspan-in-double-commutant*: $\langle \text{cspan } X \subseteq \text{commutant } (\text{commutant } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-star-closure-in-double-commutant*: $\langle \text{weak-star-topology closure-of } X \subseteq \text{commutant } (\text{commutant } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *weak-star-closure-cspan-in-double-commutant*: $\langle \text{weak-star-topology closure-of cspan } X \subseteq \text{commutant } (\text{commutant } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-memberI*:
assumes $\langle \bigwedge y. y \in X \implies x \circ_{CL} y = y \circ_{CL} x \rangle$
shows $\langle x \in \text{commutant } X \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-sot-closed*: $\langle \text{closedin cstrong-operator-topology } (\text{commutant } A) \rangle$
— [2], Exercise IX.6.2
 $\langle \text{proof} \rangle$

lemma *commutant-tensor1'*: $\langle \text{commutant } (\text{range } (\lambda a. \text{id-cblinfun } \otimes_o a)) = \text{range } (\lambda b. b \otimes_o \text{id-cblinfun}) \rangle$
 $\langle \text{proof} \rangle$

lemma *closed-map-sot-tensor-op-id-right*:
 $\langle \text{closed-map cstrong-operator-topology cstrong-operator-topology } (\lambda a. a \otimes_o \text{id-cblinfun} :: ('a \times 'b) \text{ ell2} \implies_{CL} ('a \times 'b) \text{ ell2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *id-in-commutant[iff]*: $\langle \text{id-cblinfun} \in \text{commutant } A \rangle$
 $\langle \text{proof} \rangle$

lemma *double-commutant-hull*: $\langle \text{commutant} (\text{commutant } X) = (\lambda X. \text{commutant} (\text{commutant } X) = X) \text{ hull } X \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-adj-closed*: $\langle (\bigwedge x. x \in X \implies x^* \in X) \implies x \in \text{commutant } X \implies x^* \in \text{commutant } X \rangle$
 $\langle \text{proof} \rangle$

lemma *double-commutant-Un-left*: $\langle \text{commutant} (\text{commutant} (\text{commutant} (\text{commutant } X) \cup Y)) = \text{commutant} (\text{commutant} (X \cup Y)) \rangle$
 $\langle \text{proof} \rangle$

lemma *double-commutant-Un-right*: $\langle \text{commutant} (\text{commutant} (X \cup \text{commutant} (\text{commutant } Y))) = \text{commutant} (\text{commutant} (X \cup Y)) \rangle$
 $\langle \text{proof} \rangle$

lemma *amplification-double-commutant-commute*:
 $\langle \text{commutant} (\text{commutant} ((\lambda a. a \otimes_o \text{id-cblinfun}) ' X))$
 $= (\lambda a. a \otimes_o \text{id-cblinfun}) ' \text{commutant} (\text{commutant } X) \rangle$
— [7], Corollary IV.1.5
 $\langle \text{proof} \rangle$

lemma *amplification-double-commutant-commute'*:
 $\langle \text{commutant} (\text{commutant} ((\lambda a. \text{id-cblinfun} \otimes_o a) ' X))$
 $= (\lambda a. \text{id-cblinfun} \otimes_o a) ' \text{commutant} (\text{commutant } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-cspan*: $\langle \text{commutant} (\text{cspan } A) = \text{commutant } A \rangle$
 $\langle \text{proof} \rangle$

lemma *double-commutant-grows'*: $\langle x \in X \implies x \in \text{commutant} (\text{commutant } X) \rangle$
 $\langle \text{proof} \rangle$

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'* :: $\langle \text{nat} \Rightarrow 'a \text{ ell2 ccspace list} \Rightarrow ('a \times \text{nat}) \text{ ell2 ccspace} \rangle$ **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* :: $\langle \text{nat} \Rightarrow ('a \times \text{nat}) \text{ ell2 ccspace} \rangle$ **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** $\langle n < m \rangle$
 $\langle \text{proof} \rangle$

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** $\langle n < m \rangle$
 $\langle \text{proof} \rangle$

lemma *Proj-inflation-carrier*: $\langle \text{Proj} (\text{inflation-carrier } n) = \text{inflation-op} (\text{replicate } n \text{id-cblinfun}) \rangle$
 $\langle \text{proof} \rangle$

lemma *inflation-op-carrierI*:

assumes $\langle \text{Proj} (\text{inflation-carrier } n) \text{ o}_{CL} a \text{ o}_{CL} \text{Proj} (\text{inflation-carrier } n) = a \rangle$
shows $\langle a \in \text{inflation-op-carrier } n \rangle$
 $\langle \text{proof} \rangle$

lemma *inflation-op-compose*: $\langle \text{inflation-op}' n \text{ ops1} \text{ o}_{CL} \text{inflation-op}' n \text{ ops2} = \text{inflation-op}' n (\text{map2} \text{cblinfun-compose} \text{ops1} \text{ops2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *inflation-op-in-carrier*: $\langle \text{inflation-op} \text{ops} \in \text{inflation-op-carrier } n \rangle$ **if** $\langle \text{length} \text{ops} \leq n \rangle$
 $\langle \text{proof} \rangle$

lemma *inflation-op'-apply-tensor-outside*: $\langle n < m \implies \text{inflation-op}' m \text{ as} *_{\mathcal{V}} (v \otimes_s \text{ket } n) = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *inflation-op'-compose-tensor-outside*: $\langle n < m \implies \text{inflation-op}' m \text{ as} \text{ o}_{CL} \text{tensor-ell2-right} (\text{ket } n) = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *inflation-state'-apply-tensor-outside*: $\langle n < m \implies (a \otimes_o \text{butterfly} \psi (\text{ket } n)) *_{\mathcal{V}} \text{inflation-state}' m \text{ vs} = 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *inflation-op-apply-inflation-state*: $\langle \text{inflation-op}' n \text{ ops} *_{\mathcal{V}} \text{inflation-state}' n \text{ vecs} = \text{infla-} \rangle$

tion-state' n (*map2 cblinfun-apply ops vecs*)
 ⟨*proof*⟩

lemma *inflation-state-in-carrier*: ⟨*inflation-state vecs* \in *space-as-set (inflation-carrier n)*⟩ **if**
 ⟨*length vecs* + $m \leq n$ ⟩
 ⟨*proof*⟩

lemma *inflation-op'-apply-tensor-outside'*: ⟨ $n \geq \text{length } as + m \implies \text{inflation-op}' m \text{ as } *_V (v \otimes_s$
 $\text{ket } n) = 0$ ⟩
 ⟨*proof*⟩

lemma *Proj-inflation-carrier-outside*: ⟨*Proj (inflation-carrier n) *_V ($\psi \otimes_s \text{ket } i) = 0$* ⟩ **if** ⟨ $i \geq$
 n ⟩
 ⟨*proof*⟩

lemma *inflation-state'-is-orthogonal-outside*: ⟨ $n < m \implies \text{is-orthogonal } (a \otimes_s \text{ket } n) (\text{inflation-state}'$
 $m \text{ vs})$ ⟩
 ⟨*proof*⟩

lemma *inflation-op-adj*: ⟨*(inflation-op' n ops)* = inflation-op' n (map adj ops)*⟩
 ⟨*proof*⟩

lemma *inflation-state0*:
assumes ⟨ $\bigwedge v. v \in \text{set } f \implies v = 0$ ⟩
shows ⟨*inflation-state' n f = 0*⟩
 ⟨*proof*⟩

lemma *inflation-state-plus*:
assumes ⟨*length f = length g*⟩
shows ⟨*inflation-state' n f + inflation-state' n g = inflation-state' n (map2 plus f g)*⟩
 ⟨*proof*⟩

lemma *inflation-state-minus*:
assumes ⟨*length f = length g*⟩
shows ⟨*inflation-state' n f - inflation-state' n g = inflation-state' n (map2 minus f g)*⟩
 ⟨*proof*⟩

lemma *inflation-state-scaleC*:
shows ⟨ $c *_C \text{inflation-state}' n f = \text{inflation-state}' n (\text{map } (\text{scaleC } c) f)$ ⟩
 ⟨*proof*⟩

lemma *inflation-op-compose-tensor-ell2-right*:
assumes ⟨ $i \geq n$ ⟩ **and** ⟨ $i < n + \text{length } f$ ⟩
shows ⟨*inflation-op' n f* $o_{CL} \text{tensor-ell2-right } (\text{ket } i) = \text{tensor-ell2-right } (\text{ket } i) o_{CL} (f!(i-n))$ ⟩
 ⟨*proof*⟩

lemma *inflation-op-apply*:
assumes ⟨ $i \geq n$ ⟩ **and** ⟨ $i < n + \text{length } f$ ⟩

shows $\langle \text{inflation-op}' n f *_V (\psi \otimes_s \text{ket } i) = (f!(i-n) *_V \psi) \otimes_s \text{ket } i \rangle$
 $\langle \text{proof} \rangle$

lemma *norm-inflation-state*:

$\langle \text{norm } (\text{inflation-state}' n f) = \text{sqrt } (\sum v \leftarrow f. (\text{norm } v)^2) \rangle$
 $\langle \text{proof} \rangle$

lemma *cstrong-operator-topology-in-closure-algebraicI*:

— [2], Proposition IX.5.3

assumes *space*: $\langle \text{csubspace } A \rangle$

assumes *mult*: $\langle \bigwedge a a'. a \in A \implies a' \in A \implies a \text{ } o_{CL} \text{ } a' \in A \rangle$

assumes *one*: $\langle \text{id-cblinfun } \in A \rangle$

assumes *main*: $\langle \bigwedge n S. S \leq \text{inflation-carrier } n \implies (\bigwedge a. a \in A \implies \text{inflation-op } (\text{replicate } n \text{ } a) *_S S \leq S) \implies$

$\text{inflation-op } (\text{replicate } n \text{ } b) *_S S \leq S \rangle$

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

$\langle \text{proof} \rangle$

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 $\langle (\lambda a. \text{inflation-op } (\text{replicate } n \text{ } a)) \text{ 'commutant } (\text{commutant } A)$

$\subseteq \text{commutant}' (\text{commutant}' ((\lambda a. \text{inflation-op } (\text{replicate } n \text{ } a)) \text{ ' } A)) \rangle$

$\langle \text{proof} \rangle$

lemma *double-commutant-theorem-aux*:

— Basically the double commutant theorem, except that we restricted to spaces of the form $'a \text{ ell2}$

— [2], Proposition IX.6.4

fixes *A* :: $\langle ('a \text{ ell2} \Rightarrow_{CL} 'a \text{ ell2}) \text{ set} \rangle$

assumes *csubspace* $\langle \text{csubspace } A \rangle$

assumes $\langle \bigwedge a a'. a \in A \implies a' \in A \implies a \text{ } o_{CL} \text{ } a' \in A \rangle$

assumes *id-cblinfun* $\langle \text{id-cblinfun } \in A \rangle$

assumes $\langle \bigwedge a. a \in A \implies a^* \in A \rangle$

shows $\langle \text{commutant } (\text{commutant } A) = \text{cstrong-operator-topology closure-of } A \rangle$

$\langle \text{proof} \rangle$

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* :: $\langle ('a :: \{ \text{hilbert-space}, \text{not-singleton} \} \Rightarrow_{CL} 'a) \text{ set} \rangle$

assumes *subspace*: $\langle \text{csubspace } A \rangle$

assumes *mult*: $\langle \bigwedge a a'. a \in A \implies a' \in A \implies a \text{ } o_{CL} \text{ } a' \in A \rangle$

assumes *id*: $\langle \text{id-cblinfun } \in A \rangle$

assumes *adj*: $\langle \bigwedge a. a \in A \implies a^* \in A \rangle$

shows $\langle \text{commutant } (\text{commutant } A) = \text{cstrong-operator-topology closure-of } A \rangle$

⟨proof⟩

lemma *double-commutant-theorem*:

— [2], Proposition IX.6.4

fixes $A :: \langle 'a :: \{\text{hilbert-space}\} \Rightarrow_{CL} 'a \text{ set} \rangle$

assumes *subspace*: $\langle \text{csubspace } A \rangle$

assumes *mult*: $\langle \bigwedge a \ a'. \ a \in A \implies a' \in A \implies a \ o_{CL} \ a' \in A \rangle$

assumes *id*: $\langle \text{id-cblinfun} \in A \rangle$

assumes *adj*: $\langle \bigwedge a. \ a \in A \implies a^* \in A \rangle$

shows $\langle \text{commutant } (\text{commutant } A) = \text{cstrong-operator-topology closure-of } A \rangle$

⟨proof⟩

hide-fact *double-commutant-theorem-aux double-commutant-theorem-aux2*

lemma *double-commutant-theorem-span*:

fixes $A :: \langle 'a :: \{\text{hilbert-space}\} \Rightarrow_{CL} 'a \text{ set} \rangle$

assumes *mult*: $\langle \bigwedge a \ a'. \ a \in A \implies a' \in A \implies a \ o_{CL} \ a' \in A \rangle$

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⟩

16.3 Von Neumann Algebras

definition *one-algebra* :: $\langle 'a \Rightarrow_{CL} 'a :: \text{hilbert-space} \text{ set} \rangle$ **where**

$\langle \text{one-algebra} = \text{range } (\lambda c. \ c *_{\mathcal{C}} \text{id-cblinfun}) \rangle$

definition *von-neumann-algebra* **where** $\langle \text{von-neumann-algebra } A \iff (\forall a \in A. \ a^* \in A) \wedge \text{commutant } (\text{commutant } A) = A \rangle$

definition *von-neumann-factor* **where** $\langle \text{von-neumann-factor } A \iff \text{von-neumann-algebra } A \wedge A \cap \text{commutant } A = \text{one-algebra} \rangle$

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}

⟨proof⟩

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⟩

lemma *commutant-UNIV*: $\langle \text{commutant } (\text{UNIV} :: ('a \Rightarrow_{CL} 'a :: \text{hilbert-space} \text{ set})) = \text{one-algebra} \rangle$

⟨proof⟩

lemma *von-neumann-algebra-UNIV*: $\langle \text{von-neumann-algebra } \text{UNIV} \rangle$

$\langle \text{proof} \rangle$

lemma *von-neumann-factor-UNIV*: $\langle \text{von-neumann-factor UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *von-neumann-algebra-UNION*:
assumes $\langle \bigwedge x. x \in X \implies \text{von-neumann-algebra } (A\ x) \rangle$
shows $\langle \text{von-neumann-algebra } (\text{commutant } (\text{commutant } (\bigcup_{x \in X}. A\ x))) \rangle$
 $\langle \text{proof} \rangle$

lemma *von-neumann-algebra-union*:
assumes $\langle \text{von-neumann-algebra } A \rangle$
assumes $\langle \text{von-neumann-algebra } B \rangle$
shows $\langle \text{von-neumann-algebra } (\text{commutant } (\text{commutant } (A \cup B))) \rangle$
 $\langle \text{proof} \rangle$

lemma *von-neumann-algebra-commutant*: $\langle \text{von-neumann-algebra } (\text{commutant } A) \rangle$ **if** $\langle \text{von-neumann-algebra } A \rangle$
 $\langle \text{proof} \rangle$

lemma *von-neumann-algebra-def-sot*:
 $\langle \text{von-neumann-algebra } \mathfrak{F} \longleftrightarrow$
 $(\forall a \in \mathfrak{F}. a^* \in \mathfrak{F}) \wedge \text{csubspace } \mathfrak{F} \wedge (\forall a \in \mathfrak{F}. \forall b \in \mathfrak{F}. a \circ_{CL} b \in \mathfrak{F}) \wedge \text{id-cblinfun} \in \mathfrak{F} \wedge$
 $\text{closedin cstrong-operator-topology } \mathfrak{F} \rangle$
 $\langle \text{proof} \rangle$

lemma *double-commutant-hull'*:
assumes $\langle \bigwedge x. x \in X \implies x^* \in X \rangle$
shows $\langle \text{commutant } (\text{commutant } X) = \text{von-neumann-algebra hull } X \rangle$
 $\langle \text{proof} \rangle$

lemma *commutant-one-algebra*: $\langle \text{commutant one-algebra} = \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

definition *tensor-vn (infixr \otimes_{vN} 70) where*
 $\langle \text{tensor-vn } X\ Y = \text{commutant } (\text{commutant } ((\lambda a. a \otimes_{\circ} \text{id-cblinfun}) \text{ ` } X \cup (\lambda a. \text{id-cblinfun} \otimes_{\circ} a) \text{ ` } Y)) \rangle$

lemma *von-neumann-algebra-adj-image*: $\langle \text{von-neumann-algebra } X \implies \text{adj ` } X = X \rangle$
 $\langle \text{proof} \rangle$

lemma *von-neumann-algebra-tensor-vn*:
assumes $\langle \text{von-neumann-algebra } X \rangle$
assumes $\langle \text{von-neumann-algebra } Y \rangle$
shows $\langle \text{von-neumann-algebra } (X \otimes_{vN} Y) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-vn-one-one*[simp]: $\langle \text{one-algebra} \otimes_{vN} \text{one-algebra} = \text{one-algebra} \rangle$
 $\langle \text{proof} \rangle$

lemma *sandwich-swap-tensor-vn*: $\langle \text{sandwich swap-ell2 } '(X \otimes_{vN} Y) = Y \otimes_{vN} X \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-vn-one-left*: $\langle \text{one-algebra} \otimes_{vN} X = (\lambda x. \text{id-cblinfun} \otimes_o x) 'X \rangle$ **if** $\langle \text{von-neumann-algebra } X \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-vn-one-right*: $\langle X \otimes_{vN} \text{one-algebra} = (\lambda x. x \otimes_o \text{id-cblinfun}) 'X \rangle$ **if** $\langle \text{von-neumann-algebra } X \rangle$
 $\langle \text{proof} \rangle$

lemma *double-commutant-in-vn-algI*: $\langle \text{commutant } (\text{commutant } X) \subseteq Y \rangle$
if $\langle \text{von-neumann-algebra } Y \rangle$ **and** $\langle X \subseteq Y \rangle$
 $\langle \text{proof} \rangle$

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 $\langle x \circ_{CL} y \in M \rangle$
 $\langle \text{proof} \rangle$

lemma *von-neumann-algebra-id*:
assumes $\langle \text{von-neumann-algebra } M \rangle$
shows $\langle \text{id-cblinfun} \in M \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-vn-UNIV*[simp]: $\langle \text{UNIV} \otimes_{vN} \text{UNIV} = (\text{UNIV} :: (('a \times 'b) \text{ ell2} \Rightarrow_{CL} \text{-}) \text{ set}) \rangle$
 $\langle \text{proof} \rangle$

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 existing 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 $\text{tensor-unpack } A B i = \text{tensor-unpack } A B j \longleftrightarrow i = j$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-bound1[simp]*: $i < A * B \implies \text{fst } (\text{tensor-unpack } A B i) < A$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-bound2[simp]*: $i < A * B \implies \text{snd } (\text{tensor-unpack } A B i) < B$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-fstfst*: $\langle \text{fst } (\text{tensor-unpack } A B (\text{fst } (\text{tensor-unpack } (A * B) C i)))$
 $= \text{fst } (\text{tensor-unpack } A (B * C) i) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-sndsnd*: $\langle \text{snd } (\text{tensor-unpack } B C (\text{snd } (\text{tensor-unpack } A (B * C) i)))$
 $= \text{snd } (\text{tensor-unpack } (A * B) C i) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-fstsnd*: $\langle \text{fst } (\text{tensor-unpack } B C (\text{snd } (\text{tensor-unpack } A (B * C) i)))$
 $= \text{snd } (\text{tensor-unpack } A B (\text{fst } (\text{tensor-unpack } (A * B) C i))) \rangle$
 $\langle \text{proof} \rangle$

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]*: $\langle \text{dim-vec } (\text{tensor-state-jnf } \psi \varphi) = \text{dim-vec } \psi * \text{dim-vec } \varphi \rangle$
 $\langle \text{proof} \rangle$

lemma *enum-prod-nth-tensor-unpack*:
assumes $\langle i < \text{CARD}'a * \text{CARD}'b \rangle$
shows $(\text{Enum.enum } ! i :: 'a :: \text{enum} \times 'b :: \text{enum}) =$
 $(\text{let } (i1, i2) = \text{tensor-unpack } \text{CARD}'a \text{ CARD}'b i \text{ in}$
 $(\text{Enum.enum } ! i1, \text{Enum.enum } ! i2))$
 $\langle \text{proof} \rangle$

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
 $\langle \text{vec-of-basis-enum } \psi \ \$ \ i1 \ * \ \text{vec-of-basis-enum } \varphi \ \$ \ i2 \rangle$
 $\langle \text{proof} \rangle$

lemma *vec-of-basis-enum-tensor-state*:

fixes $\psi :: \langle 'a::\text{enum } \text{ell2} \rangle$ **and** $\varphi :: \langle 'b::\text{enum } \text{ell2} \rangle$
shows $\langle \text{vec-of-basis-enum } (\psi \otimes_s \varphi) = \text{tensor-state-jnf } (\text{vec-of-basis-enum } \psi) (\text{vec-of-basis-enum } \varphi) \rangle$
 $\langle \text{proof} \rangle$

lemma *mat-of-cblinfun-tensor-op-index*:

fixes $a :: \langle 'a::\text{enum } \text{ell2} \Rightarrow_{CL} 'b::\text{enum } \text{ell2} \rangle$ **and** $b :: \langle 'c::\text{enum } \text{ell2} \Rightarrow_{CL} 'd::\text{enum } \text{ell2} \rangle$
assumes $[simp]: \langle i < \text{CARD}('b) * \text{CARD}('d) \rangle$
assumes $[simp]: \langle j < \text{CARD}('a) * \text{CARD}('c) \rangle$
shows $\langle \text{mat-of-cblinfun } (\text{tensor-op } a \ b) \ \$\$ \ (i,j) =$
 $(\text{let } (i1,i2) = \text{tensor-unpack } \text{CARD}('b) \ \text{CARD}('d) \ i \ \text{in}$
 $\text{let } (j1,j2) = \text{tensor-unpack } \text{CARD}('a) \ \text{CARD}('c) \ j \ \text{in}$
 $\text{mat-of-cblinfun } a \ \$\$ \ (i1,j1) * \text{mat-of-cblinfun } b \ \$\$ \ (i2,j2)) \rangle$
 $\langle \text{proof} \rangle$

definition *tensor-op-jnf* $A \ B =$

$(\text{let } r1 = \text{dim-row } A \ \text{in}$
 $\text{let } c1 = \text{dim-col } A \ \text{in}$
 $\text{let } r2 = \text{dim-row } B \ \text{in}$
 $\text{let } c2 = \text{dim-col } B \ \text{in}$
 $\text{mat } (r1 * r2) \ (c1 * c2)$
 $(\lambda(i,j). \text{let } (i1,i2) = \text{tensor-unpack } r1 \ r2 \ i \ \text{in}$
 $\text{let } (j1,j2) = \text{tensor-unpack } c1 \ c2 \ j \ \text{in}$
 $(A \ \$\$ \ (i1,j1)) * (B \ \$\$ \ (i2,j2))))$

lemma *tensor-op-jnf-dim* $[simp]:$

$\langle \text{dim-row } (\text{tensor-op-jnf } a \ b) = \text{dim-row } a * \text{dim-row } b \rangle$
 $\langle \text{dim-col } (\text{tensor-op-jnf } a \ b) = \text{dim-col } a * \text{dim-col } b \rangle$
 $\langle \text{proof} \rangle$

lemma *mat-of-cblinfun-tensor-op*:

fixes $a :: \langle 'a::\text{enum } \text{ell2} \Rightarrow_{CL} 'b::\text{enum } \text{ell2} \rangle$ **and** $b :: \langle 'c::\text{enum } \text{ell2} \Rightarrow_{CL} 'd::\text{enum } \text{ell2} \rangle$
shows $\langle \text{mat-of-cblinfun } (\text{tensor-op } a \ b) = \text{tensor-op-jnf } (\text{mat-of-cblinfun } a) (\text{mat-of-cblinfun } b) \rangle$
 $\langle \text{proof} \rangle$

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*)

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*)

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. Künar 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.