Tensor Products in Hilbert Spaces*

Dominique Unruh

March 10, 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

*Supported by the ERC consolidator grant CerQuS (819317), the PRG team grant Secure Quantum Technology (PRG946) from the Estonian Research Council, the Estonian Centre of Exellence in IT (EXCITE) funded by ERDF, and the Estonian Cluster of Excellence “Foundations of the Universe” (TK202).

1 *Misc-Tensor-Product* – Miscelleanous 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*: $(\bigwedge x.\ x{=}y \Longrightarrow P) \Longrightarrow P$ ⟨*proof*⟩

lemma *inv-prod-swap*[*simp*]: ‹*inv prod.swap* = *prod.swap*›
⟨*proof*⟩

lemma *filterlim-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique S*›
shows ‹((*R* ===> *S*) ===> *rel-filter S* ===> *rel-filter R* ===> (=)) *filterlim filterlim*›
⟨*proof*⟩

definition *rel-topology* :: ‹(′*a* ⇒ ′*b* ⇒ *bool*) ⇒ (′*a topology* ⇒ ′*b topology* ⇒ *bool*)› **where**
‹*rel-topology R S T* ⟷ (*rel-fun* (*rel-set R*) (=)) (*openin S*) (*openin T*)
∧ (∀ *U*. *openin S U* ⟶ *Domainp* (*rel-set R*) *U*) ∧ (∀ *U*. *openin T U* ⟶ *Rangep* (*rel-set R*) *U*)›

lemma *rel-topology-eq*[*relator-eq*]: ‹*rel-topology* (=) = (=)›
⟨*proof*⟩

lemma *Rangep-conversep*[*simp*]: ‹*Rangep* (R^{-1-1}) = *Domainp R*›
⟨*proof*⟩

lemma *Domainp-conversep*[*simp*]: ‹*Domainp* (R^{-1-1}) = *Rangep R*›
⟨*proof*⟩

lemma *conversep-rel-fun*:
includes *lifting-syntax*
shows ‹$(T$ ===> $U)^{-1-1}$ = (T^{-1-1}) ===> (U^{-1-1})›
⟨*proof*⟩

lemma *rel-topology-conversep*[*simp*]: ‹*rel-topology* (R^{-1-1}) = ((*rel-topology R*)$^{-1-1}$)›
⟨*proof*⟩

lemma *openin-parametric*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-topology R* ===> *rel-set R* ===> (=)) *openin openin*›
⟨*proof*⟩

lemma *topspace-parametric* [*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-topology R* ===> *rel-set R*) *topspace topspace*›
⟨*proof*⟩

lemma [*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-total S*›
  assumes [*transfer-rule*]: ‹*bi-unique S*›
  assumes [*transfer-rule*]: ‹*bi-total R*›
  assumes [*transfer-rule*]: ‹*bi-unique R*›
  shows ‹(*rel-topology R* ===> *rel-topology S* ===> (*R* ===> *S*) ===> (=)) *continuous-map continuous-map*›
  ⟨*proof*⟩

lemma *limitin-closedin*:
  assumes ‹*limitin T f c F*›
  assumes ‹*range f* ⊆ *S*›
  assumes ‹*closedin T S*›
  assumes ‹¬ *trivial-limit F*›
  shows ‹*c* ∈ *S*›
⟨*proof*⟩

lemma *closure-nhds-principal*: ‹*a* ∈ *closure A* ⟷ *inf* (*nhds a*) (*principal A*) ≠ *bot*›
⟨*proof*⟩

lemma *limit-in-closure*:
  assumes *lim*: ‹(*f* ⟶ *x*) *F*›
  assumes *nt*: ‹*F* ≠ *bot*›
  assumes *inA*: ‹$\forall_F$ *x in F*. *f x* ∈ *A*›
  shows ‹*x* ∈ *closure A*›
⟨*proof*⟩

lemma *filterlim-nhdsin-iff-limitin*:
  ‹*l* ∈ *topspace T* ∧ *filterlim f* (*nhdsin T l*) *F* ⟷ *limitin T f l F*›
  ⟨*proof*⟩

lemma *pullback-topology-bi-cont*:
  fixes *g* :: ‹*'a* ⇒ (*'b* ⇒ *'c*::*topological-space*)›
    and *f* :: ‹*'a* ⇒ *'a* ⇒ *'a*› **and** *f'* :: ‹*'c* ⇒ *'c* ⇒ *'c*›
  assumes *gf-f'g*: ‹⋀*a b i*. *g* (*f a b*) *i* = *f'* (*g a i*) (*g b i*)›
  assumes *f'-cont*: ‹⋀*a' b'*. (*case-prod f'* ⟶ *f' a' b'*) (*nhds a'* $\times_F$ *nhds b'*)›
  defines ‹*T* ≡ *pullback-topology UNIV g euclidean*›
  shows ‹*LIM* (*x*,*y*) *nhdsin T a* $\times_F$ *nhdsin T b*. *f x y* :> *nhdsin T* (*f a b*)›
⟨*proof*⟩

definition ‹*has-sum-in T f A x* ⟷ *limitin T* (*sum f*) *x* (*finite-subsets-at-top A*)›

lemma *has-sum-in-finite*:

assumes *finite F*
assumes ‹*sum f F* ∈ *topspace T*›
shows *has-sum-in T f F* (*sum f F*)
⟨*proof*⟩

definition ‹*summable-on-in T f A* ⟷ (∃ *x*. *has-sum-in T f A x*)›

definition ‹*infsum-in T f A* = (*let L* = *Collect* (*has-sum-in T f A*) *in if card L* = *1 then the-elem L else 0*)›

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

lemma *hausdorffI*:
assumes ‹⋀*x y*. *x* ∈ *topspace T* ⟹ *y* ∈ *topspace T* ⟹ *x* ≠ *y* ⟹ ∃ *U V*. *openin T U* ∧ *openin T V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* ∩ *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* = (*if summable-on-in T f A then* (*THE s*. *has-sum-in T f A s*) *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 *infsum-in-finite*:
assumes *finite F*
assumes ‹*Hausdorff-space T*›
assumes ‹*sum f F* ∈ *topspace T*›
shows *infsum-in T f F* = *sum f F*

⟨*proof*⟩

lemma *nhdsin-mono*:
assumes [*simp*]: ‹$\bigwedge x.$ *openin* T' $x \Longrightarrow$ *openin* T x›
assumes [*simp*]: ‹*topspace* T = *topspace* T'›
shows ‹*nhdsin* T $a \leq$ *nhdsin* T' a›
⟨*proof*⟩

lemma *has-sum-in-cong*:
assumes $\bigwedge x.$ $x \in A \Longrightarrow f\ x = g\ x$
shows *has-sum-in* T f A $x \longleftrightarrow$ *has-sum-in* T g A x
⟨*proof*⟩

lemma *infsum-in-eqI'*:
fixes f g :: ‹$'a \Rightarrow\ 'b$::*comm-monoid-add*›
assumes ‹$\bigwedge x.$ *has-sum-in* T f A $x \longleftrightarrow$ *has-sum-in* T g B x›
shows ‹*infsum-in* T f A = *infsum-in* T g B›
⟨*proof*⟩

lemma *infsum-in-cong*:
assumes $\bigwedge x.$ $x \in A \Longrightarrow f\ x = g\ x$
shows *infsum-in* T f A = *infsum-in* T g A
⟨*proof*⟩

lemma *limitin-cong*: *limitin* T f c $F \longleftrightarrow$ *limitin* T g c F **if** *eventually* $(\lambda x.\ f\ x = g\ x)$ F
⟨*proof*⟩

lemma *has-sum-in-reindex*:
assumes ‹*inj-on* h A›
shows ‹*has-sum-in* T g $(h$ ‘ $A)$ $x \longleftrightarrow$ *has-sum-in* T $(g \circ h)$ A x›
⟨*proof*⟩

lemma *summable-on-in-reindex*:
assumes ‹*inj-on* h A›
shows ‹*summable-on-in* T g $(h$ ‘ $A) \longleftrightarrow$ *summable-on-in* T $(g \circ h)$ A›
⟨*proof*⟩

lemma *infsum-in-reindex*:
assumes ‹*inj-on* h A›
shows ‹*infsum-in* T g $(h$ ‘ $A)$ = *infsum-in* T $(g \circ h)$ A›
⟨*proof*⟩

lemma *has-sum-in-reindex-bij-betw*:
assumes *bij-betw* g A B
shows *has-sum-in* T $(\lambda x.\ f\ (g\ x))$ A $s \longleftrightarrow$ *has-sum-in* T f B s
⟨*proof*⟩

lemma *has-sum-euclidean-iff*: ‹*has-sum-in euclidean f A s* $\longleftrightarrow$ (*f has-sum s*) *A*›
⟨*proof*⟩

lemma *summable-on-euclidean-eq*: ‹*summable-on-in euclidean f A* $\longleftrightarrow$ *f summable-on A*›
⟨*proof*⟩

lemma *infsum-euclidean-eq*: ‹*infsum-in euclidean f A* = *infsum f A*›
⟨*proof*⟩

lemma *infsum-in-reindex-bij-betw*:
assumes *bij-betw g A B*
shows *infsum-in T* ($\lambda x.$ *f* (*g x*)) *A* = *infsum-in T f B*
⟨*proof*⟩

lemma *limitin-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique S*›
shows ‹(*rel-topology S* ===> (*R* ===> *S*) ===> *S* ===> *rel-filter R* ===> ($\longleftrightarrow$)) *limitin limitin*›
⟨*proof*⟩

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

lemma *sum-parametric'*[*transfer-rule*]:
includes *lifting-syntax*
fixes *R* :: ‹*'a* ⇒ *'b* ⇒ *bool*› **and** *S* :: ‹*'c*::*comm-monoid-add* ⇒ *'d*::*comm-monoid-add* ⇒ *bool*›
assumes [*transfer-rule*]: ‹*bi-unique R*›
assumes [*transfer-rule*]: ‹(*S* ===> *S* ===> *S*) (+) (+)›
assumes [*transfer-rule*]: ‹*S 0 0*›
shows ‹((*R* ===> *S*) ===> *rel-set R* ===> *S*) *sum sum*›
⟨*proof*⟩

lemma *has-sum-in-parametric*[*transfer-rule*]:
includes *lifting-syntax*
fixes *R* :: ‹*'a* ⇒ *'b* ⇒ *bool*› **and** *S* :: ‹*'c*::*comm-monoid-add* ⇒ *'d*::*comm-monoid-add* ⇒ *bool*›
assumes [*transfer-rule*]: ‹*bi-unique R*›
assumes [*transfer-rule*]: ‹*bi-unique S*›
assumes [*transfer-rule*]: ‹(*S* ===> *S* ===> *S*) (+) (+)›
assumes [*transfer-rule*]: ‹*S 0 0*›
shows ‹(*rel-topology S* ===> (*R* ===> *S*) ===> (*rel-set R*) ===> *S* ===> (=)) *has-sum-in has-sum-in*›
⟨*proof*⟩

lemma *has-sum-in-topspace*: ‹*has-sum-in T f A s* ⟹ *s* ∈ *topspace T*›

⟨*proof*⟩

lemma *summable-on-in-parametric*[*transfer-rule*]:
includes *lifting-syntax*
fixes *R* :: ‹*'a* ⇒ *'b* ⇒ *bool*›
assumes [*transfer-rule*]: ‹*bi-unique R*›
assumes [*transfer-rule*]: ‹*bi-unique S*›
assumes [*transfer-rule*]: ‹(*S* ===> *S* ===> *S*) (+) (+)›
assumes [*transfer-rule*]: ‹*S 0 0*›
shows ‹(*rel-topology S* ===> (*R* ===> *S*) ===> (*rel-set R*) ===> (=)) *summable-on-in summable-on-in*›
⟨*proof*⟩

lemma *not-summable-infsum-in-0*: ‹¬ *summable-on-in T f A* ⟹ *infsum-in T f A* = *0*›
⟨*proof*⟩

lemma *infsum-in-parametric*[*transfer-rule*]:
includes *lifting-syntax*
fixes *R* :: ‹*'a* ⇒ *'b* ⇒ *bool*›
assumes [*transfer-rule*]: ‹*bi-unique R*›
assumes [*transfer-rule*]: ‹*bi-unique S*›
assumes [*transfer-rule*]: ‹(*S* ===> *S* ===> *S*) (+) (+)›
assumes [*transfer-rule*]: ‹*S 0 0*›
shows ‹(*rel-topology S* ===> (*R* ===> *S*) ===> (*rel-set R*) ===> *S*) *infsum-in infsum-in*›
⟨*proof*⟩

lemma *infsum-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique R*›
shows ‹((*R* ===> (=)) ===> (*rel-set R*) ===> (=)) *infsum infsum*›
⟨*proof*⟩

lemma *summable-on-transfer*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique R*›
shows ‹((*R* ===> (=)) ===> (*rel-set R*) ===> (=)) *Infinite-Sum.summable-on Infinite-Sum.summable-on*›
⟨*proof*⟩

lemma *abs-gbinomial*: ‹*abs* (*a gchoose n*) = (−*1*)^(*n* − *nat* (*ceiling a*)) ∗ (*a gchoose n*)›
⟨*proof*⟩

lemma *gbinomial-sum-lower-abs*:
fixes *a* :: ‹*'a*::{*floor-ceiling*}›
defines ‹*a'* ≡ *nat* (*ceiling a*)›
assumes ‹*of-nat m* ≥ *a*−*1*›
shows (∑ *k*≤*m*. *abs* (*a gchoose k*)) =
(−*1*)^*a'* ∗ ((−*1*) ^ *m* ∗ (*a* − *1 gchoose m*))
− (−*1*)^*a'* ∗ *of-bool* (*a'*>*0*) ∗ ((−*1*) ^(*a'*−*1*) ∗ (*a*−*1 gchoose* (*a'*−*1*)))

+ ($\sum$ *k*<*a'*. *abs* (*a gchoose k*))
⟨*proof*⟩

lemma *abs-gbinomial-leq1*:
fixes *a* :: ‹*'a* :: {*linordered-field*}›
assumes ‹*abs a* ≤ *1*›
shows ‹*abs* (*a gchoose b*) ≤ *1*›
⟨*proof*⟩

lemma *gbinomial-summable-abs*:
fixes *a* :: *real*
assumes ‹*a* ≥ *0*› **and** ‹*a* ≤ *1*›
shows ‹*summable* (λ*n*. *abs* (*a gchoose n*))›
⟨*proof*⟩

lemma *summable-tendsto-times-n*:
fixes *f* :: ‹*nat* ⇒ *real*›
assumes *pos*: ‹⋀*n*. *f n* ≥ *0*›
assumes *dec*: ‹*decseq* (λ*n*. (*n*+*M*) ∗ *f* (*n* + *M*))›
assumes *sum*: ‹*summable f*›
shows ‹(λ*n*. *n* ∗ *f n*) ⟶ *0*›
⟨*proof*⟩

lemma *gbinomial-tendsto-0*:
fixes *a* :: *real*
assumes ‹*a* > −*1*›
shows ‹(λ*n*. (*a gchoose n*)) ⟶ *0*›
⟨*proof*⟩

lemma *gbinomial-abs-sum*:
fixes *a* :: *real*
assumes ‹*a* > *0*› **and** ‹*a* ≤ *1*›
shows ‹(λ*n*. *abs* (*a gchoose n*)) *sums 2*›
⟨*proof*⟩

lemma *sums-has-sum*:
fixes *s* :: ‹*'a* :: *banach*›
assumes *sums*: ‹*f sums s*›
assumes *abs-sum*: ‹*summable* (λ*n*. *norm* (*f n*))›
shows ‹(*f has-sum s*) *UNIV*›
⟨*proof*⟩

lemma *sums-has-sum-pos*:
fixes *s* :: *real*

assumes ‹*f sums s*›
assumes ‹$\bigwedge n.\ f\ n \geq 0$›
shows ‹(*f has-sum s*) *UNIV*›
⟨*proof*⟩

lemma *gbinomial-abs-has-sum*:
fixes *a* :: *real*
assumes ‹$a > 0$› **and** ‹$a \leq 1$›
shows ‹((λ*n. abs* (*a gchoose n*)) *has-sum 2*) *UNIV*›
⟨*proof*⟩

lemma *gbinomial-abs-has-sum-1*:
fixes *a* :: *real*
assumes ‹$a > 0$› **and** ‹$a \leq 1$›
shows ‹((λ*n. abs* (*a gchoose n*)) *has-sum 1*) (*UNIV*−{*0*})›
⟨*proof*⟩

lemma *gbinomial-abs-summable*:
fixes *a* :: *real*
assumes ‹$a > 0$› **and** ‹$a \leq 1$›
shows ‹(λ*n.* (*a gchoose n*)) *abs-summable-on UNIV*›
⟨*proof*⟩

lemma *gbinomial-abs-summable-1*:
fixes *a* :: *real*
assumes ‹$a > 0$› **and** ‹$a \leq 1$›
shows ‹(λ*n.* (*a gchoose n*)) *abs-summable-on UNIV*−{*0*}›
⟨*proof*⟩

lemma *has-sum-singleton*[*simp*]: ‹(*f has-sum y*) {*x*} ⟷ *f x* = *y*› **for** *y* :: ‹*'a* :: {*comm-monoid-add*, *t2-space*}›
⟨*proof*⟩

lemma *has-sum-sums*: ‹*f sums s*› **if** ‹(*f has-sum s*) *UNIV*›
⟨*proof*⟩

lemma *The-eqI1*:
assumes ‹$\bigwedge x\ y.\ F\ x \Longrightarrow F\ y \Longrightarrow x = y$›
assumes ‹$\exists z.\ F\ z$›
assumes ‹$\bigwedge x.\ F\ x \Longrightarrow P\ x = Q\ x$›
shows ‹*P* (*The F*) = *Q* (*The F*)›
⟨*proof*⟩

lemma *summable-on-uminus*[*intro*!]:
fixes *f* :: ‹*'a* ⇒ *'b* :: *real-normed-vector*›
assumes ‹*f summable-on A*›
shows ‹(λ*i.* − *f i*) *summable-on A*›
⟨*proof*⟩

lemma *summable-on-diff*:
  fixes *f g* :: *'a* ⇒ *'b*::*real-normed-vector*
  assumes ‹*f summable-on A*›
  assumes ‹*g summable-on A*›
  shows ‹(*λx. f x* − *g x*) *summable-on A*›
  ⟨*proof*⟩

lemma *gbinomial-1*: ‹(*1 gchoose n*) = *of-bool* (*n*≤*1*)›
⟨*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 *summable-on-in-cong*:
  assumes ⋀*x. x*∈*A* ⟹ *f x* = *g x*
  shows *summable-on-in T f A* ⟷ *summable-on-in T g A*
  ⟨*proof*⟩

lemma *infsum-in-0*:
  assumes ‹*Hausdorff-space T*› **and** ‹*0* ∈ *topspace T*›
  assumes ‹⋀*x. x*∈*M* ⟹ *f x* = *0*›
  shows ‹*infsum-in T f M* = *0*›
⟨*proof*⟩

lemma *summable-on-in-finite*:
  fixes *f* :: ‹*'a* ⇒ *'b*::{*comm-monoid-add*,*topological-space*}›
  assumes *finite F*
  assumes ‹*sum f F* ∈ *topspace T*›
  shows *summable-on-in T f F*
  ⟨*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 :: $'a \Rightarrow$ *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 :: $'a \Rightarrow$ $'b$ :: {*t2-space*, *topological-semigroup-mult*, *division-ring*}
  assumes ‹f *summable-on* A›
  shows (λx. f x / c) *summable-on* A
  ⟨*proof*⟩

lemma *has-sum-in-weaker-topology*:
  assumes ‹*continuous-map* T U (λf. f)›
  assumes ‹*has-sum-in* T f A l›
  shows ‹*has-sum-in* U f A l›
  ⟨*proof*⟩

lemma *summable-on-in-weaker-topology*:
  assumes ‹*continuous-map* T U (λf. f)›
  assumes ‹*summable-on-in* T f A›
  shows ‹*summable-on-in* U f 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 :: $'a \Rightarrow$ $'b$::*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 :: $'a \Rightarrow$ $'b$::{*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 ‹($\sum_\infty i{\in}A$. *norm* (x i $*$ y i)) $\leq$ *sqrt* ($\sum_\infty i{\in}A$. $(\mathit{norm}\ (x\ i))^2$) $*$ *sqrt* ($\sum_\infty i{\in}A$. (*norm*
(y i))^2)›
⟨*proof*⟩

lemma *continuous-map-pullback-both*:
  assumes *cont*: ‹*continuous-map* $T1$ $T2$ g'›
  assumes $g'g$: ‹$\bigwedge x$. $f1$ x $\in$ *topspace* $T1$ $\Longrightarrow$ $x \in A1$ $\Longrightarrow$ g' ($f1$ x) = $f2$ (g x)›
  assumes *top1*: ‹$f1$ −‘ *topspace* $T1$ $\cap$ $A1$ $\subseteq$ g −‘ $A2$›
  shows ‹*continuous-map* (*pullback-topology* $A1$ $f1$ $T1$) (*pullback-topology* $A2$ $f2$ $T2$) g›
⟨*proof*⟩

lemma *onorm-case-prod-plus-leq*: ‹*onorm* (*case-prod plus* :: - ⇒ ′*a*::*real-normed-vector*) ≤ *sqrt* *2*›
⟨*proof*⟩

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

lemma *pullback-topology-twice*:
assumes ‹(*f* −‘ *B*) ∩ *A* = *C*›
shows ‹*pullback-topology A f* (*pullback-topology B g T*) = *pullback-topology C* (*g o f*) *T*›
⟨*proof*⟩

lemma *pullback-topology-homeo-cong*:
assumes ‹*homeomorphic-map T S g*›
assumes ‹*range f* ⊆ *topspace T*›
shows ‹*pullback-topology A f T* = *pullback-topology A* (*g o f*) *S*›
⟨*proof*⟩

definition ‹*opensets-in T* = *Collect* (*openin T*)›
— This behaves more nicely with the *transfer*-method (and friends) than *openin*. So when rewriting a subgoal, using, e.g., ∃ *U*∈*opensets T*. *xxx* instead of ∃ *U*. *openin T U* ⟶ *xxx* can make *transfer* work better.

lemma *opensets-in-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes ‹*bi-unique R*›
shows ‹(*rel-topology R* ===> *rel-set* (*rel-set R*)) *opensets-in opensets-in*›
⟨*proof*⟩

lemma *hausdorff-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique R*›
shows ‹(*rel-topology R* ===> (⟷)) *Hausdorff-space Hausdorff-space*›
⟨*proof*⟩

lemma *sum-cmod-pos*:
assumes ‹⋀*x*. *x*∈*A* ⟹ *f x* ≥ *0*›
shows ‹(∑ *x*∈*A*. *cmod* (*f x*)) = *cmod* (∑ *x*∈*A*. *f x*)›
⟨*proof*⟩

lemma *min-power-distrib-left*: ‹(*min x y*) ^ *n* = *min* (*x* ^ *n*) (*y* ^ *n*)› **if** ‹*x* ≥ *0*› **and** ‹*y* ≥ *0*› **for** *x y* :: ‹- :: *linordered-semidom*›
⟨*proof*⟩

lemma *abs-summable-times*:
fixes *f* :: ‹′*a* ⇒ ′*c*::{*real-normed-algebra*}› **and** *g* :: ‹′*b* ⇒ ′*c*›
assumes *sum-f*: ‹*f abs-summable-on A*›
assumes *sum-g*: ‹*g abs-summable-on B*›

shows ‹(λ(*i*,*j*). *f i* ∗ *g j*) *abs-summable-on A* × *B*›
⟨*proof*⟩

definition ‹*the-default def S* = (*if card S* = *1 then* (*THE x*. *x* ∈ *S*) *else def*)›

lemma *card1I*:
assumes *a* ∈ *A*
assumes ⋀*x*. *x* ∈ *A* ⟹ *x* = *a*
shows ‹*card A* = *1*›
⟨*proof*⟩

lemma *the-default-CollectI*:
assumes *P a*
and ⋀*x*. *P x* ⟹ *x* = *a*
shows *P* (*the-default d* (*Collect P*))
⟨*proof*⟩

lemma *the-default-singleton*[*simp*]: ‹*the-default def* {*x*} = *x*›
⟨*proof*⟩

lemma *the-default-empty*[*simp*]: ‹*the-default def* {} = *def*›
⟨*proof*⟩

lemma *the-default-The*: ‹*the-default z S* = (*THE x*. *x* ∈ *S*)› **if** ‹*card S* = *1*›
⟨*proof*⟩

lemma *the-default-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique T*›
shows ‹(*T* ===> *rel-set T* ===> *T*) *the-default the-default*›
⟨*proof*⟩

definition ‹*rel-pred T P Q* = *rel-set T* (*Collect P*) (*Collect Q*)›

lemma *Collect-parametric*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-pred T* ===> *rel-set T*) *Collect Collect*›
⟨*proof*⟩

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* ⟷ (*F* ≤ *principal* (*Collect S*))›
⟨*proof*⟩

lemma *map-filter-on-cong*:
assumes [*simp*]: ‹$\forall_F$ *x in F*. *x* ∈ *D*›
assumes ‹⋀*x*. *x* ∈ *D* ⟹ *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 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* ≠ {}›
assumes *F-bounded*: ‹⋀*b*. *b*∈*B* ⟹ *F b* ≤ *principal X*›
shows ‹*filtermap f* (⨅ (*F* ‘ *B*)) = (⨅*b*∈*B*. *filtermap f* (*F b*))›
⟨*proof*⟩

lemma *filtermap-inf-eq*:
assumes ‹*inj-on f X*›
assumes ‹*F1* ≤ *principal X*›
assumes ‹*F2* ≤ *principal X*›
shows ‹*filtermap f* (*F1* ⊓ *F2*) = *filtermap f F1* ⊓ *filtermap f F2*›
⟨*proof*⟩

definition ‹*transfer-bounded-filter-Inf B M* = *Inf M* ⊓ *principal B*›

lemma *Inf-transfer-bounded-filter-Inf*: ‹*Inf M* = *transfer-bounded-filter-Inf UNIV M*›
⟨*proof*⟩

lemma *Inf-bounded-transfer-bounded-filter-Inf*:
assumes ‹⋀*F*. *F* ∈ *M* ⟹ *F* ≤ *principal B*›
assumes ‹*M* ≠ {}›
shows ‹*Inf M* = *transfer-bounded-filter-Inf B M*›
⟨*proof*⟩

lemma *transfer-bounded-filter-Inf-parametric*[*transfer-rule*]:
includes *lifting-syntax*
fixes *r* :: ‹′*rep* ⇒ ′*abs* ⇒ *bool*›
assumes [*transfer-rule*]: ‹*bi-unique r*›
shows ‹(*rel-set r* ===> *rel-set* (*rel-filter r*) ===> *rel-filter r*)
transfer-bounded-filter-Inf transfer-bounded-filter-Inf›
⟨*proof*⟩

definition ‹*transfer-inf-principal F M* = *F* ⊓ *principal M*›

lemma *transfer-inf-principal-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique T*›
shows ‹(*rel-filter T* ===> *rel-set T* ===> *rel-filter T*) *transfer-inf-principal transfer-inf-principal*›
⟨*proof*⟩

lemma *continuous-map-is-continuous-at-point*:
assumes ‹*continuous-map T U f*›
shows ‹*filterlim f* (*nhdsin U* (*f l*)) (*atin T l*)›
⟨*proof*⟩

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

lemma *closure-image-closure*: ‹*continuous-on* (*closure S*) *f* ⟹ *closure* (*f* ‘ *closure S*) = *closure* (*f* ‘ *S*)›
⟨*proof*⟩

lemma *has-sum-reindex-bij-betw*:
assumes *bij-betw g A B*
shows ((λ*x*. *f* (*g x*)) *has-sum l*) *A* ⟷ (*f has-sum l*) *B*
⟨*proof*⟩

lemma *enum-inj*:
assumes *i* < *CARD*(′*a*) **and** *j* < *CARD*(′*a*)
shows (*Enum.enum* ! *i* :: ′*a*::*enum*) = *Enum.enum* ! *j* ⟷ *i* = *j*
⟨*proof*⟩

lemma *closedin-vimage*:
assumes ‹*closedin U S*›
assumes ‹*continuous-map T U f*›
shows ‹*closedin T* (*topspace T* ∩ (*f* −‘ *S*))›
⟨*proof*⟩

lemma *join-forall*: ‹$(\forall x.\ P\ x) \wedge (\forall x.\ Q\ x) \longleftrightarrow (\forall x.\ P\ x \wedge Q\ x)$›
⟨*proof*⟩

lemma *closedin-if-converge-inside*:
fixes A :: ‹*'a set*›
assumes AT: ‹$A \subseteq$ *topspace* T›
assumes xA: ‹$\bigwedge(F::'a\ filter)\ f\ x.\ F \neq \bot \Longrightarrow limitin\ T\ f\ x\ F \Longrightarrow range\ f \subseteq A \Longrightarrow x \in A$›
shows ‹*closedin T A*›
⟨*proof*⟩

lemma *cmod-mono*: ‹$0 \le a \Longrightarrow a \le b \Longrightarrow cmod\ a \le cmod\ b$›
⟨*proof*⟩

lemma *choice2*: ‹$\exists f.\ (\forall x.\ Q1\ x\ (f\ x)) \wedge (\forall x.\ Q2\ x\ (f\ x))$›
if ‹$\forall x.\ \exists y.\ Q1\ x\ y \wedge Q2\ x\ y$›
⟨*proof*⟩

lemma *choice3*: ‹$\exists f.\ (\forall x.\ Q1\ x\ (f\ x)) \wedge (\forall x.\ Q2\ x\ (f\ x)) \wedge (\forall x.\ Q3\ x\ (f\ x))$›
if ‹$\forall x.\ \exists y.\ Q1\ x\ y \wedge Q2\ x\ y \wedge Q3\ x\ y$›
⟨*proof*⟩

lemma *choice4*: ‹$\exists f.\ (\forall x.\ Q1\ x\ (f\ x)) \wedge (\forall x.\ Q2\ x\ (f\ x)) \wedge (\forall x.\ Q3\ x\ (f\ x)) \wedge (\forall x.\ Q4\ x\ (f\ x))$›
if ‹$\forall x.\ \exists y.\ Q1\ x\ y \wedge Q2\ x\ y \wedge Q3\ x\ y \wedge Q4\ x\ y$›
⟨*proof*⟩

lemma *choice5*: ‹$\exists f.\ (\forall x.\ Q1\ x\ (f\ x)) \wedge (\forall x.\ Q2\ x\ (f\ x)) \wedge (\forall x.\ Q3\ x\ (f\ x)) \wedge (\forall x.\ Q4\ x\ (f\ x)) \wedge (\forall x.\ Q5\ x\ (f\ x))$›
if ‹$\forall x.\ \exists y.\ Q1\ x\ y \wedge Q2\ x\ y \wedge Q3\ x\ y \wedge Q4\ x\ y \wedge Q5\ x\ y$›
⟨*proof*⟩

lemma *is-Sup-unique*: ‹$is\text{-}Sup\ X\ a \Longrightarrow is\text{-}Sup\ X\ b \Longrightarrow a=b$›
⟨*proof*⟩

lemma *has-Sup-bdd-above*: ‹$has\text{-}Sup\ X \Longrightarrow bdd\text{-}above\ X$›
⟨*proof*⟩

lemma *is-Sup-has-Sup*: ‹$is\text{-}Sup\ X\ s \Longrightarrow has\text{-}Sup\ X$›
⟨*proof*⟩

class *Sup-order* = *order* + *Sup* + *sup* +
assumes *is-Sup-Sup*: ‹$has\text{-}Sup\ X \Longrightarrow is\text{-}Sup\ X\ (Sup\ X)$›
assumes *is-Sup-sup*: ‹$has\text{-}Sup\ \{x,y\} \Longrightarrow is\text{-}Sup\ \{x,y\}\ (sup\ x\ y)$›

lemma (**in** *Sup-order*) *is-Sup-eq-Sup*:
assumes ‹*is-Sup X s*›
shows ‹$s = Sup\ X$›
⟨*proof*⟩

lemma *is-Sup-cSup*:
  fixes X :: ‹$'a$::*conditionally-complete-lattice set*›
  assumes ‹*bdd-above X*› **and** ‹$X \neq \{\}$›
  shows ‹*is-Sup X* (*Sup X*)›
  ⟨*proof*⟩

lemma *continuous-map-iff-preserves-convergence*:
  assumes ‹$\bigwedge F\ a.\ a \in topspace\ T \Longrightarrow limitin\ T\ id\ a\ F \Longrightarrow limitin\ U\ f\ (f\ a)\ F$›
  shows ‹*continuous-map T U f*›
  ⟨*proof*⟩

lemma *SMT-choices*:
  — Was included as SMT.choices in Isabelle and disappeared
  $\bigwedge Q.\ \forall x.\ \exists y\ ya.\ Q\ x\ y\ ya \Longrightarrow \exists f\ fa.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)$
  $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb.\ Q\ x\ y\ ya\ yb \Longrightarrow \exists f\ fa\ fb.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)$
  $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc.\ Q\ x\ y\ ya\ yb\ yc \Longrightarrow \exists f\ fa\ fb\ fc.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)$
  $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd.\ Q\ x\ y\ ya\ yb\ yc\ yd \Longrightarrow$
    $\exists f\ fa\ fb\ fc\ fd.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)$
  $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd\ ye.\ Q\ x\ y\ ya\ yb\ yc\ yd\ ye \Longrightarrow$
    $\exists f\ fa\ fb\ fc\ fd\ fe.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)\ (fe\ x)$
  $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd\ ye\ yf.\ Q\ x\ y\ ya\ yb\ yc\ yd\ ye\ yf \Longrightarrow$
    $\exists f\ fa\ fb\ fc\ fd\ fe\ ff.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)\ (fe\ x)\ (ff\ x)$
  $\bigwedge Q.\ \forall x.\ \exists y\ ya\ yb\ yc\ yd\ ye\ yf\ yg.\ Q\ x\ y\ ya\ yb\ yc\ yd\ ye\ yf\ yg \Longrightarrow$
    $\exists f\ fa\ fb\ fc\ fd\ fe\ ff\ fg.\ \forall x.\ Q\ x\ (f\ x)\ (fa\ x)\ (fb\ x)\ (fc\ x)\ (fd\ x)\ (fe\ x)\ (ff\ x)\ (fg\ x)$
  ⟨*proof*⟩

lemma *closedin-pullback-topology*:
  $closedin\ (pullback\text{-}topology\ A\ f\ T)\ S \longleftrightarrow (\exists C.\ closedin\ T\ C \wedge S = f-`C \cap A)$
⟨*proof*⟩

lemma *regular-space-pullback*[*intro*]:
  assumes ‹*regular-space T*›
  shows ‹*regular-space* (*pullback-topology A f T*)›
⟨*proof*⟩

lemma *t3-space-euclidean-regular*[*iff*]: ‹*regular-space* (*euclidean* :: $'a$::*t3-space topology*)›
  ⟨*proof*⟩

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

lemma *increasing-filtermap*:
  fixes F :: ‹$'a$::*order filter*› **and** f :: ‹$'a \Rightarrow 'b$::*order*› **and** X :: ‹$'a$ *set*›
  assumes *increasing*: ‹*increasing-filter F*›
  assumes *mono*: ‹*mono-on X f*›

assumes *ev-X*: ‹$\mathit{eventually}\ (\lambda x.\ x \in X)\ F$›
shows ‹*increasing-filter* (*filtermap f F*)›
⟨*proof*⟩

lemma *increasing-finite-subsets-at-top*[*simp*]: ‹*increasing-filter* (*finite-subsets-at-top X*)›
⟨*proof*⟩

lemma *monotone-convergence*:
— Following [5]
fixes f :: ‹$'b \Rightarrow{} 'a$::{*order-topology*, *conditionally-complete-linorder*}›
assumes *bounded*: ‹$\forall_F\ x\ \mathit{in}\ F.\ f\ x \leq B$›
assumes *increasing*: ‹*increasing-filter* (*filtermap f F*)›
shows ‹$\exists\, l.\ (f \longrightarrow l)\ F$›
⟨*proof*⟩

lemma *monotone-convergence-complex*:
fixes f :: ‹$'b \Rightarrow$ *complex*›
assumes *bounded*: ‹$\forall_F\ x\ \mathit{in}\ F.\ f\ x \leq B$›
assumes *increasing*: ‹*increasing-filter* (*filtermap f F*)›
shows ‹$\exists\, l.\ (f \longrightarrow l)\ F$›
⟨*proof*⟩

lemma *compact-closed-subset*:
assumes ‹*compact s*›
assumes ‹*closed t*›
assumes ‹$t \subseteq s$›
shows ‹*compact t*›
⟨*proof*⟩

definition *separable* **where** ‹$\mathit{separable}\ S \longleftrightarrow (\exists\, B.\ \mathit{countable}\ B \wedge S \subseteq \mathit{closure}\ B)$›

lemma *compact-imp-separable*: ‹*separable S*› **if** ‹*compact S*› **for** S :: ‹$'a$::*metric-space set*›
⟨*proof*⟩

lemma *infsum-single*:
assumes $\bigwedge j.\ j \neq i \Longrightarrow j{\in}A \Longrightarrow f\ j = 0$
shows $\mathit{infsum}\ f\ A = (\mathit{if}\ i{\in}A\ \mathit{then}\ f\ i\ \mathit{else}\ 0)$
⟨*proof*⟩

lemma *suminf-eqI*:
fixes x :: ‹-::{*comm-monoid-add*,*t2-space*}›
assumes ‹*f sums x*›
shows ‹$\mathit{suminf}\ f = x$›
⟨*proof*⟩

lemma *suminf-If-finite-set*:
fixes f :: ‹- $\Rightarrow$ -::{*comm-monoid-add*,*t2-space*}›
assumes ‹*finite F*›

shows ‹($\sum$ *x*∈*F*. *f x*) = ($\sum$ *x*. *if x*∈*F then f x else 0*)›
⟨*proof*⟩

lemma *tendsto-le-complex*:
fixes *x y* :: *complex*
assumes *F*: ¬ *trivial-limit F*
and *x*: (*f* ⟶ *x*) *F*
and *y*: (*g* ⟶ *y*) *F*
and *ev*: *eventually* (λ*x*. *g x* ≤ *f x*) *F*
shows *y* ≤ *x*
⟨*proof*⟩

lemma *bdd-above-mono2*:
assumes ‹*bdd-above* (*g* ‘ *B*)›
assumes ‹*A* ⊆ *B*›
assumes ‹⋀*x*. *x* ∈ *A* ⟹ *f x* ≤ *g x*›
shows ‹*bdd-above* (*f* ‘ *A*)›
⟨*proof*⟩

lemma *infsum-product*:
fixes *f* :: ‹′*a* ⇒ ′*c* :: {*topological-semigroup-mult*,*division-ring*,*banach*}›
assumes ‹(λ(*x*, *y*). *f x* ∗ *g y*) *summable-on X* × *Y*›
shows ‹($\sum_\infty$*x*∈*X*. *f x*) ∗ ($\sum_\infty$*y*∈*Y*. *g y*) = ($\sum_\infty$(*x*,*y*)∈*X*×*Y*. *f x* ∗ *g y*)›
⟨*proof*⟩

lemma *infsum-product′*:
fixes *f* :: ‹′*a* ⇒ ′*c* :: {*banach*,*times*,*real-normed-algebra*}› **and** *g* :: ‹′*b* ⇒ ′*c*›
assumes ‹*f abs-summable-on X*›
assumes ‹*g abs-summable-on Y*›
shows ‹($\sum_\infty$*x*∈*X*. *f x*) ∗ ($\sum_\infty$*y*∈*Y*. *g y*) = ($\sum_\infty$(*x*,*y*)∈*X*×*Y*. *f x* ∗ *g y*)›
⟨*proof*⟩

lemma *infsum-bounded-linear-invertible*:
assumes ‹*bounded-linear h*›
assumes ‹*bounded-linear h′*›
assumes ‹*h′ o h* = *id*›
shows ‹*infsum* (λ*x*. *h* (*f x*)) *A* = *h* (*infsum f A*)›
⟨*proof*⟩

lemma *summable-on-bdd-above-real*: ‹*bdd-above* (*f* ‘ *M*)› **if** ‹*f summable-on M*› **for** *f* :: ‹′*a* ⇒ *real*›
⟨*proof*⟩

end

2 *Strong-Operator-Topology* – Strong operator topology on complex bounded operators

theory *Strong-Operator-Topology*
imports
Complex-Bounded-Operators.Complex-Bounded-Linear-Function
Misc-Tensor-Product
begin

unbundle *cblinfun-syntax*

typedef (**overloaded**) (*'a,'b*) *cblinfun-sot* = ‹*UNIV* :: (*'a*::*complex-normed-vector* $\Rightarrow_{CL}$ *'b*::*complex-normed-vector*) *set*› ⟨*proof*⟩
setup-lifting *type-definition-cblinfun-sot*

instantiation *cblinfun-sot* :: (*complex-normed-vector*, *complex-normed-vector*) *complex-vector*
begin
lift-definition *scaleC-cblinfun-sot* :: ‹*complex* ⇒ (*'a*, *'b*) *cblinfun-sot* ⇒ (*'a*, *'b*) *cblinfun-sot*›
is ‹*scaleC*› ⟨*proof*⟩
lift-definition *uminus-cblinfun-sot* :: ‹(*'a*, *'b*) *cblinfun-sot* ⇒ (*'a*, *'b*) *cblinfun-sot*› **is** *uminus* ⟨*proof*⟩
lift-definition *zero-cblinfun-sot* :: ‹(*'a*, *'b*) *cblinfun-sot*› **is** *0* ⟨*proof*⟩
lift-definition *minus-cblinfun-sot* :: ‹(*'a*, *'b*) *cblinfun-sot* ⇒ (*'a*, *'b*) *cblinfun-sot* ⇒ (*'a*, *'b*) *cblinfun-sot*› **is** *minus* ⟨*proof*⟩
lift-definition *plus-cblinfun-sot* :: ‹(*'a*, *'b*) *cblinfun-sot* ⇒ (*'a*, *'b*) *cblinfun-sot* ⇒ (*'a*, *'b*) *cblinfun-sot*› **is** *plus* ⟨*proof*⟩
lift-definition *scaleR-cblinfun-sot* :: ‹*real* ⇒ (*'a*, *'b*) *cblinfun-sot* ⇒ (*'a*, *'b*) *cblinfun-sot*› **is** *scaleR* ⟨*proof*⟩
instance
⟨*proof*⟩
end

instantiation *cblinfun-sot* :: (*complex-normed-vector*, *complex-normed-vector*) *topological-space*
begin
lift-definition *open-cblinfun-sot* :: ‹(*'a*, *'b*) *cblinfun-sot set* ⇒ *bool*› **is** ‹*openin cstrong-operator-topology*› ⟨*proof*⟩
instance
⟨*proof*⟩
end

lemma *transfer-nhds-cstrong-operator-topology*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*cr-cblinfun-sot* ===> *rel-filter cr-cblinfun-sot*) (*nhdsin cstrong-operator-topology*) *nhds*›
⟨*proof*⟩

lemma *filterlim-cstrong-operator-topology*: ‹*filterlim f* (*nhdsin cstrong-operator-topology l*) = *limitin cstrong-operator-topology f l*›
⟨*proof*⟩

lemma *hausdorff-sot*[*simp*]: ‹*Hausdorff-space cstrong-operator-topology*›
⟨*proof*⟩

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

lemma *Domainp-cr-cblinfun-sot*[*simp*]: ‹*Domainp cr-cblinfun-sot* = (λ-. *True*)›
⟨*proof*⟩

lemma *Rangep-cr-cblinfun-sot*[*simp*]: ‹*Rangep cr-cblinfun-sot* = (λ-. *True*)›
⟨*proof*⟩

lemma *Rangep-set*[*relator-domain*]: *Rangep* (*rel-set T*) = (λ*A*. *Ball A* (*Rangep T*))
⟨*proof*⟩

lemma *transfer-euclidean-cstrong-operator-topology*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-topology cr-cblinfun-sot*) *cstrong-operator-topology euclidean*›
⟨*proof*⟩

lemma *openin-cstrong-operator-topology*: ‹*openin cstrong-operator-topology U* ⟷ (∃ *V*. *open V* ∧ *U* = ($*_V$) −‘ *V*)›
⟨*proof*⟩

lemma *cstrong-operator-topology-plus-cont*: ‹*LIM* (*x*,*y*) *nhdsin cstrong-operator-topology a* $\times_F$ *nhdsin cstrong-operator-topology b*.
x + *y* :> *nhdsin cstrong-operator-topology* (*a* + *b*)›
⟨*proof*⟩

instance *cblinfun-sot* :: (*complex-normed-vector*, *complex-normed-vector*) *topological-group-add*
⟨*proof*⟩

lemma *continuous-map-left-comp-sot*[*continuous-intros*]:
fixes *b* :: ‹*'b*::*complex-normed-vector* $\Rightarrow_{CL}$ *'c*::*complex-normed-vector*›
and *f* :: ‹*'a* ⇒ *'d*::*complex-normed-vector* $\Rightarrow_{CL}$ *'b*›
assumes ‹*continuous-map T cstrong-operator-topology f*›
shows ‹*continuous-map T cstrong-operator-topology* (λ*x*. *b* o_{CL} *f x*)›
⟨*proof*⟩

lemma *continuous-cstrong-operator-topology-plus*[*continuous-intros*]:
assumes ‹*continuous-map T cstrong-operator-topology f*›

assumes ‹*continuous-map T cstrong-operator-topology g*›
shows ‹*continuous-map T cstrong-operator-topology* ($\lambda x.\ f\ x + g\ x$)›
⟨*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* ($\lambda x.\ f\ x\ o_{CL}\ a$)›
⟨*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* :: ‹*'a::topological-space* ⇒ (-,-) *cblinfun-sot*›
assumes ‹*continuous-on X f*›
shows ‹*continuous-on X* ($\lambda x.\ c *_C f\ x$)›
⟨*proof*⟩

lemma *sot-closure-is-csubspace*[*simp*]:
fixes *A*::(*'a::complex-normed-vector*, *'b::complex-normed-vector*) *cblinfun-sot set*
assumes ‹*csubspace A*›
shows ‹*csubspace* (*closure A*)›
⟨*proof*⟩
include *lattice-syntax*
⟨*proof*⟩

lemma *limitin-cstrong-operator-topology*:
‹*limitin cstrong-operator-topology f l F* ⟷ ($\forall i.\ ((\lambda j.\ f\ j *_V i) \longrightarrow l *_V i)\ F$)›
⟨*proof*⟩

lemma *cstrong-operator-topology-in-closureI*:
assumes ‹$\bigwedge M\ \varepsilon.\ \varepsilon > 0 \Longrightarrow$ *finite M* $\Longrightarrow \exists a \in A.\ \forall v \in M.$ *norm* $((b-a) *_V v) \le \varepsilon$›
shows ‹$b \in$ *cstrong-operator-topology closure-of A*›
⟨*proof*⟩

lemma *sot-weaker-than-norm-limitin*: ‹*limitin cstrong-operator-topology a A F*› **if** ‹($a \longrightarrow A$) F›
⟨*proof*⟩

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-sot* ===> (=)) *csubspace csubspace*›
⟨*proof*⟩

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-sot* ===> (=)) (*closedin cstrong-operator-topology*) *closed*›
⟨*proof*⟩

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-sot* ===> *rel-set cr-cblinfun-sot*) (*Abstract-Topology.closure-of cstrong-operator-topology*) *closure*›
⟨*proof*⟩

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

lemma *has-sum-closed-cstrong-operator-topology*:
assumes *aA*: ‹$\bigwedge i.\ a\ i \in A$›
assumes *closed*: ‹*closedin cstrong-operator-topology A*›
assumes *subspace*: ‹*csubspace A*›
assumes *has-sum*: ‹$\bigwedge\psi.$ (($\lambda i.\ a\ i *_V \psi$) *has-sum* ($b *_V \psi$)) I›
shows ‹$b \in A$›
⟨*proof*⟩

lemma *has-sum-in-cstrong-operator-topology*:
‹*has-sum-in cstrong-operator-topology f A l* $\longleftrightarrow$ ($\forall\psi.$ (($\lambda i.\ f\ i *_V \psi$) *has-sum* ($l *_V \psi$)) A)›
⟨*proof*⟩

lemma *summable-sot-absI*:
fixes b :: ‹$'a \Rightarrow 'b$::*complex-normed-vector* $\Rightarrow_{CL}$ $'c$::*chilbert-space*›
assumes ‹$\bigwedge F\ f.$ *finite* $F \Longrightarrow (\sum n{\in}F.\ norm\ (b\ n *_V f)) \leq K * norm\ f$›
shows ‹*summable-on-in cstrong-operator-topology b UNIV*›
⟨*proof*⟩

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

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

lemma *isCont-cblinfun-compose-sot-right*[*simp*]: ‹*isCont* (*λF*. *cblinfun-compose-sot F G*) *x*›
  ⟨*proof*⟩

lemma *isCont-cblinfun-compose-sot-left*[*simp*]: ‹*isCont* (*λF*. *cblinfun-compose-sot G F*) *x*›
  ⟨*proof*⟩

lemma *additive-cblinfun-compose-sot-right*[*simp*]: ‹*additive* (*λF*. *cblinfun-compose-sot F G*)›
  ⟨*proof*⟩

lemma *additive-cblinfun-compose-sot-left*[*simp*]: ‹*additive* (*λF*. *cblinfun-compose-sot G F*)›
  ⟨*proof*⟩

lemma *transfer-infsum-sot*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-unique R*›
  shows ‹((*R* ===> *cr-cblinfun-sot*) ===> *rel-set R* ===> *cr-cblinfun-sot*) (*infsum-in cstrong-operator-topology*) *infsum*›
  ⟨*proof*⟩

lemma *transfer-summable-on-sot*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-unique R*›
  shows ‹((*R* ===> *cr-cblinfun-sot*) ===> *rel-set R* ===> (⟷)) (*summable-on-in cstrong-operator-topology*) (*summable-on*)›
  ⟨*proof*⟩

lemma *sandwich-sot-cont*[*continuous-intros*]:
  assumes ‹*continuous-map T cstrong-operator-topology f*›
  shows ‹*continuous-map T cstrong-operator-topology* (*λx*. *sandwich A* (*f x*))›
  ⟨*proof*⟩

lemma *closed-map-sot-unitary-sandwich*:
  fixes *U* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*›
  assumes ‹*unitary U*›
  shows ‹*closed-map cstrong-operator-topology cstrong-operator-topology* (*λx*. *sandwich U x*)›
  ⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

3 *Positive-Operators* – Positive bounded operators

theory *Positive-Operators*
  imports
    Ordinary-Differential-Equations.Cones
    Complex-Bounded-Operators.Complex-L2

    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*: ‹$f \cdot_C (A *_V f) \geq 0$› **if** ‹$A \geq 0$›
  ⟨*proof*⟩

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

lemma *sqrt-op-nonpos*: ‹$sqrt\text{-}op\ a = 0$› **if** ‹$\neg\ a \geq 0$›
⟨*proof*⟩

lemma *generalized-Cauchy-Schwarz*:
  fixes *inner A*
  assumes *Apos*: ‹$A \geq 0$›
  defines $inner\ x\ y \equiv x \cdot_C (A *_V y)$
  shows ‹$complex\text{-}of\text{-}real\ ((norm\ (inner\ x\ y))^2) \leq inner\ x\ x * inner\ y\ y$›
⟨*proof*⟩

lemma *sandwich-pos*[*intro*]: ‹$sandwich\ b\ a \geq 0$› **if** ‹$a \geq 0$›
  ⟨*proof*⟩

lemma *cblinfun-power-pos*: ‹$cblinfun\text{-}power\ a\ n \geq 0$› **if** ‹$a \geq 0$›
⟨*proof*⟩

lemma *sqrt-op-existence*:
  fixes A :: ‹$'a::chilbert\text{-}space \Rightarrow_{CL} 'a::chilbert\text{-}space$›
  assumes *Apos*: ‹$A \geq 0$›
  shows ‹$\exists B.\ B \geq 0 \wedge B\ o_{CL}\ B = A \wedge (\forall F.\ A\ o_{CL}\ F = F\ o_{CL}\ A \longrightarrow B\ o_{CL}\ F = F\ o_{CL}\ B)$
      $\wedge\ B \in closure\ (cspan\ (range\ (cblinfun\text{-}power\ A)))$›
⟨*proof*⟩

lemma *wecken35hilfssatz*:
— Auxiliary lemma from [9]
‹$\exists P.$ *is-Proj* $P \wedge (\forall F.\ F\ o_{CL}\ (W - T) = (W - T)\ o_{CL}\ F \longrightarrow F\ o_{CL}\ P = P\ o_{CL}\ F)$
$\wedge\ (\forall f.\ W\, f = 0 \longrightarrow P\, f = f)$
$\wedge\ (W = (2 *_C P -$ *id-cblinfun*$)\ o_{CL}\ T)$›
if *WT-comm*: ‹$W\ o_{CL}\ T = T\ o_{CL}\ W$› **and** ‹$W = W*$› **and** ‹$T = T*$›
and *WW-TT*: ‹$W\ o_{CL}\ W = T\ o_{CL}\ T$›
for $W\ T$ :: ‹'a::chilbert-space $\Rightarrow_{CL}$ 'a›
⟨*proof*⟩

lemma *sqrt-op-pos*[*simp*]: ‹*sqrt-op* $a \geq 0$›
⟨*proof*⟩

lemma *sqrt-op-square*[*simp*]:
assumes ‹$a \geq 0$›
shows ‹*sqrt-op* $a\ o_{CL}$ *sqrt-op* $a = a$›
⟨*proof*⟩

lemma *sqrt-op-unique*:
— Proof follows [9]
assumes ‹$b \geq 0$› **and** ‹$b*\ o_{CL}\ b = a$›
shows ‹$b =$ *sqrt-op* a›
⟨*proof*⟩

lemma *sqrt-op-in-closure*: ‹*sqrt-op* $a \in$ *closure* (*cspan* (*range* (*cblinfun-power* a)))›
⟨*proof*⟩

lemma *sqrt-op-commute*:
assumes ‹$A \geq 0$›
assumes ‹$A\ o_{CL}\ F = F\ o_{CL}\ A$›
shows ‹*sqrt-op* $A\ o_{CL}\ F = F\ o_{CL}$ *sqrt-op* A›
⟨*proof*⟩

lemma *sqrt-op-0*[*simp*]: ‹*sqrt-op* $0 = 0$›
⟨*proof*⟩

lemma *sqrt-op-scaleC*:
assumes ‹$c \geq 0$› **and** ‹$a \geq 0$›
shows ‹*sqrt-op* $(c *_C a) =$ *sqrt* $c *_C$ *sqrt-op* a›
⟨*proof*⟩

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

lemma *abs-op-pos*[*simp*]: ‹*abs-op* $a \geq 0$›
⟨*proof*⟩

lemma *abs-op-0*[*simp*]: ‹*abs-op 0 = 0*›
⟨*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* ≥ *0*›
⟨*proof*⟩

lemma *abs-op-butterfly*[*simp*]: ‹*abs-op* (*butterfly x y*) = (*norm x* / *norm y*) $*_R$ *selfbutter y*› **for** *x* :: ‹′*a*::*chilbert-space*› **and** *y* :: ‹′*b*::*chilbert-space*›
⟨*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*)) (λψ. *A* $*_V$ *inv* (*abs-op A*) ψ) o_{CL} *Proj* (*abs-op A* $*_S$ *top*)›
for *A* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*::*complex-inner*›

lemma
fixes *A* :: ‹′*a* :: *chilbert-space* $\Rightarrow_{CL}$ ′*b* :: *chilbert-space*›
— [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* :: ‹′*a* :: *chilbert-space* $\Rightarrow_{CL}$ ′*b* :: *chilbert-space*›
⟨*proof*⟩

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

lemma *abs-opI*:
  assumes ‹$a*\ o_{CL}\ a = b*\ o_{CL}\ b$›
  assumes ‹$a \geq 0$›
  shows ‹$a = abs\text{-}op\ b$›
  ⟨*proof*⟩

lemma *abs-op-id-on-pos*: ‹$a \geq 0 \Longrightarrow abs\text{-}op\ a = a$›
  ⟨*proof*⟩

lemma *norm-abs-op*[*simp*]: ‹*norm* (*abs-op a*) = *norm a*›
  for *a* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*›
⟨*proof*⟩

lemma *partial-isometry-iff-square-proj*:
  — [2], Exercise VIII.3.15
  fixes *A* :: ‹'*a* :: *chilbert-space* $\Rightarrow_{CL}$ '*b* :: *chilbert-space*›
  shows ‹*partial-isometry A* ⟷ *is-Proj* ($A*\ o_{CL}\ A$)›
⟨*proof*⟩

lemma *abs-op-square*: ‹(*abs-op A*)$*\ o_{CL}$ *abs-op A* = $A*\ o_{CL}\ A$›
  ⟨*proof*⟩

lemma *polar-decomposition-0*[*simp*]: ‹*polar-decomposition 0* = (*0* :: '*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*)›
⟨*proof*⟩

lemma *polar-decomposition-unique*:
  fixes *A* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*›
  assumes *ker*: ‹*kernel X* = *kernel A*›
  assumes *comp*: ‹$X\ o_{CL}$ *abs-op A* = *A*›
  shows ‹*X* = *polar-decomposition A*›
⟨*proof*⟩

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* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*a*›
  assumes ‹$A \geq 0$›
  assumes ‹$A \leq B$›
  shows ‹*norm A* ≤ *norm B*›
⟨*proof*⟩

lemma *sandwich-mono*: ‹*sandwich A B* ≤ *sandwich A C*› **if** ‹$B \leq C$›
  ⟨*proof*⟩

lemma *sums-pos-cblinfun*:
  fixes *f* :: *nat* ⇒ ('*b*::*chilbert-space* $\Rightarrow_{CL}$ '*b*)
  assumes ‹*f sums a*›

assumes ‹$\bigwedge n.$ *f n* $\geq$ *0*›
shows *a* $\geq$ *0*
⟨*proof*⟩

lemma *has-sum-mono-cblinfun*:
fixes *f* :: *'a* ⇒ (*'b*::*chilbert-space* $\Rightarrow_{CL}$ *'b*)
assumes (*f has-sum x*) *A* **and** (*g has-sum y*) *A*
assumes ‹$\bigwedge x.$ *x* ∈ *A* ⟹ *f x* ≤ *g x*›
shows *x* ≤ *y*
⟨*proof*⟩

lemma *infsum-mono-cblinfun*:
fixes *f* :: *'a* ⇒ (*'b*::*chilbert-space* $\Rightarrow_{CL}$ *'b*)
assumes *f summable-on A* **and** *g summable-on A*
assumes ‹$\bigwedge x.$ *x* ∈ *A* ⟹ *f x* ≤ *g x*›
shows *infsum f A* ≤ *infsum g A*
⟨*proof*⟩

lemma *suminf-mono-cblinfun*:
fixes *f* :: *nat* ⇒ (*'b*::*chilbert-space* $\Rightarrow_{CL}$ *'b*)
assumes *summable f* **and** *summable g*
assumes ‹$\bigwedge x.$ *f x* ≤ *g x*›
shows *suminf f* ≤ *suminf g*
⟨*proof*⟩

lemma *suminf-pos-cblinfun*:
fixes *f* :: *nat* ⇒ (*'b*::*chilbert-space* $\Rightarrow_{CL}$ *'b*)
assumes ‹*summable f*›
assumes ‹$\bigwedge x.$ *f x* ≥ *0*›
shows *suminf f* ≥ *0*
⟨*proof*⟩

lemma *infsum-mono-neutral-cblinfun*:
fixes *f* :: *'a* ⇒ (*'b*::*chilbert-space* $\Rightarrow_{CL}$ *'b*)
assumes *f summable-on A* **and** *g summable-on B*
assumes ‹$\bigwedge x.$ *x* ∈ *A*∩*B* ⟹ *f x* ≤ *g x*›
assumes ‹$\bigwedge x.$ *x* ∈ *A*−*B* ⟹ *f x* ≤ *0*›
assumes ‹$\bigwedge x.$ *x* ∈ *B*−*A* ⟹ *g x* ≥ *0*›
shows *infsum f A* ≤ *infsum g B*
⟨*proof*⟩

lemma *abs-op-geq*: ‹*abs-op a* ≥ *a*› **if** ‹*selfadjoint a*›
⟨*proof*⟩

lemma *abs-op-geq-neq*: ‹*abs-op a* ≥ − *a*› **if** ‹*selfadjoint a*›
⟨*proof*⟩

lemma *infsum-nonneg-cblinfun*:

fixes f :: $'a \Rightarrow 'b$::*chilbert-space* $\Rightarrow_{CL}$ $'b$
assumes $\bigwedge x.\ x \in M \Longrightarrow 0 \leq f\ x$
shows *infsum* $f\ M \geq 0$
⟨*proof*⟩

lemma *adj-abs-op*[*simp*]: ‹(*abs-op* a)∗ = *abs-op* a›
⟨*proof*⟩

lemma *cblinfun-image-less-eqI*:
fixes A :: ‹$'a$::*complex-normed-vector* $\Rightarrow_{CL}$ $'b$::*complex-normed-vector*›
assumes ‹$\bigwedge h.\ h \in$ *space-as-set* $S \Longrightarrow A\ h \in$ *space-as-set* T›
shows ‹$A *_S S \leq T$›
⟨*proof*⟩

lemma *abs-op-plus-orthogonal*:
assumes ‹$a* \, o_{CL}\ b = 0$› **and** ‹$a\ o_{CL}\ b* = 0$›
shows ‹*abs-op* $(a + b)$ = *abs-op* a + *abs-op* b›
⟨*proof*⟩

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

definition *neg-op* :: ‹$'a$::*chilbert-space* $\Rightarrow_{CL}$ $'a \Rightarrow 'a \Rightarrow_{CL} 'a$› **where**
‹*neg-op* a = (*abs-op* $a - a$) $/_R\ 2$›

lemma *pos-op-pos*:
assumes ‹*selfadjoint* a›
shows ‹*pos-op* $a \geq 0$›
⟨*proof*⟩

lemma *neg-op-pos*:
assumes ‹*selfadjoint* a›
shows ‹*neg-op* $a \geq 0$›
⟨*proof*⟩

lemma *pos-op-neg-op-ortho*:
assumes ‹*selfadjoint* a›
shows ‹*pos-op* $a\ o_{CL}$ *neg-op* $a = 0$›
⟨*proof*⟩

lemma *pos-op-plus-neg-op*: ‹*pos-op* a + *neg-op* a = *abs-op* a›
⟨*proof*⟩

lemma *pos-op-minus-neg-op*: ‹*pos-op* a − *neg-op* $a = a$›
⟨*proof*⟩

lemma *pos-op-neg-op-unique*:
  assumes *bca*: ‹$b - c = a$›
  assumes ‹$b \geq 0$› **and** ‹$c \geq 0$›
  assumes *bc*: ‹$b\ o_{CL}\ c = 0$›
  shows ‹$b = pos\text{-}op\ a$› **and** ‹$c = neg\text{-}op\ a$›
⟨*proof*⟩

lemma *pos-imp-selfadjoint*: ‹$a \geq 0 \Longrightarrow selfadjoint\ a$›
  ⟨*proof*⟩

lemma *abs-op-one-dim*: ‹$abs\text{-}op\ x = one\text{-}dim\text{-}iso\ (abs\ (one\text{-}dim\text{-}iso\ x :: complex))$›
  ⟨*proof*⟩

lemma *pos-selfadjoint*: ‹*selfadjoint a*› **if** ‹$a \geq 0$›
  ⟨*proof*⟩

lemma *one-dim-loewner-order-strict*: ‹$A > B \longleftrightarrow one\text{-}dim\text{-}iso\ A > (one\text{-}dim\text{-}iso\ B :: complex)$›
for $A\ B$ :: ‹$'a \Rightarrow_{CL} 'a::\{chilbert\text{-}space,\ one\text{-}dim\}$›
  ⟨*proof*⟩

lemma *one-dim-cblinfun-zero-le-one*: ‹$0 < (1 :: 'a::one\text{-}dim \Rightarrow_{CL} 'a)$›
  ⟨*proof*⟩
lemma *one-dim-cblinfun-one-pos*: ‹$0 \leq (1 :: 'a::one\text{-}dim \Rightarrow_{CL} 'a)$›
  ⟨*proof*⟩

lemma *Proj-pos*[*iff*]: ‹$Proj\ S \geq 0$›
  ⟨*proof*⟩

lemma *abs-op-Proj*[*simp*]: ‹$abs\text{-}op\ (Proj\ S) = Proj\ S$›
  ⟨*proof*⟩

lemma *diagonal-operator-pos*:
  assumes ‹$\bigwedge x.\ f\ x \geq 0$›
  shows ‹$diagonal\text{-}operator\ f \geq 0$›
⟨*proof*⟩

lemma *abs-op-diagonal-operator*:
  ‹$abs\text{-}op\ (diagonal\text{-}operator\ f) = diagonal\text{-}operator\ (\lambda x.\ abs\ (f\ x))$›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

4 *HS2Ell2* – Representing any Hilbert space as $\ell_2(X)$

theory *HS2Ell2*
imports *Complex-Bounded-Operators.Complex-L2*
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* ⟷ *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* ===> *cr-chilbert2ell2-ell2* ===> (=)) *cinner cinner*›
⟨*proof*⟩

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

lemma *c2l2l2-scaleC*[*c2l2l2*]:

includes *lifting-syntax*
shows ‹((=) ===> *cr-chilbert2ell2-ell2* ===> *cr-chilbert2ell2-ell2*) *scaleC scaleC*›
⟨*proof*⟩

lemma *c2l2l2-zero*[*c2l2l2*]:
includes *lifting-syntax*
shows ‹*cr-chilbert2ell2-ell2 0 0*›
⟨*proof*⟩

lemma *c2l2l2-is-ortho-set*[*c2l2l2*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-chilbert2ell2-ell2* ===> (=)) *is-ortho-set* (*is-ortho-set* :: ′*a*::{*chilbert-space*,*not-singleton*} *set* ⇒ *bool*)›
⟨*proof*⟩

lemma *c2l2l2-ccspan*[*c2l2l2*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-chilbert2ell2-ell2* ===> *rel-ccsubspace cr-chilbert2ell2-ell2*) *ccspan ccspan*›
⟨*proof*⟩

lemma *ell2-to-hilbert-adj-ell2-to-hilbert* [*simp*]: *ell2-to-hilbert*∗ $*_V$ *ell2-to-hilbert* $*_V$ *x* = *x*
⟨*proof*⟩

lemma *ell2-to-hilbert-ell2-to-hilbert-adj* [*simp*]: *ell2-to-hilbert* $*_V$ *ell2-to-hilbert*∗ $*_V$ *x* = *x*
⟨*proof*⟩

lemma *bi-total-rel-ccsubspace-cr-chilbert2ell2-ell2*[*transfer-rule*]:
‹*bi-total* (*rel-ccsubspace cr-chilbert2ell2-ell2*)›
⟨*proof*⟩

lemma *c2l2l2-top*[*c2l2l2*]:
includes *lifting-syntax*
shows ‹(*rel-ccsubspace cr-chilbert2ell2-ell2*) *top top*›
⟨*proof*⟩

lemma *c2l2l2-is-onb*[*c2l2l2*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-chilbert2ell2-ell2* ===> (=)) *is-onb is-onb*›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

5 *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* (λa (*x*,*y*). *cinner x* (*a* $*_V$ *y*)) *euclidean*

lemma *cweak-operator-topology-topspace*[*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* $\bigwedge i.\ i \in I \Longrightarrow$ *open* (*U i*)
shows *openin cweak-operator-topology* {*f*. $\forall$ *i*∈*I*. *cinner* (*x i*) (*f* $*_V$ *y i*) ∈ *U i*}
⟨*proof*⟩

lemma *wot-weaker-than-sot*:
continuous-map cstrong-operator-topology cweak-operator-topology (λf. *f*)
⟨*proof*⟩

lemma *cweak-operator-topology-weaker-than-euclidean*:
continuous-map euclidean cweak-operator-topology (λf. *f*)
⟨*proof*⟩

lemma *cweak-operator-topology-cinner-continuous*:
continuous-map cweak-operator-topology euclidean (λf. *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* (λz. *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 ‹*scaleC*› ⟨*proof*⟩
lift-definition *uminus-cblinfun-wot* :: ‹(′*a*, ′*b*) *cblinfun-wot* ⇒ (′*a*, ′*b*) *cblinfun-wot*› **is** *uminus* ⟨*proof*⟩
lift-definition *zero-cblinfun-wot* :: ‹(′*a*, ′*b*) *cblinfun-wot*› **is** *0* ⟨*proof*⟩
lift-definition *minus-cblinfun-wot* :: ‹(′*a*, ′*b*) *cblinfun-wot* ⇒ (′*a*, ′*b*) *cblinfun-wot* ⇒ (′*a*, ′*b*) *cblinfun-wot*› **is** *minus* ⟨*proof*⟩
lift-definition *plus-cblinfun-wot* :: ‹(′*a*, ′*b*) *cblinfun-wot* ⇒ (′*a*, ′*b*) *cblinfun-wot* ⇒ (′*a*, ′*b*) *cblinfun-wot*› **is** *plus* ⟨*proof*⟩
lift-definition *scaleR-cblinfun-wot* :: ‹*real* ⇒ (′*a*, ′*b*) *cblinfun-wot* ⇒ (′*a*, ′*b*) *cblinfun-wot*› **is** *scaleR* ⟨*proof*⟩
instance
⟨*proof*⟩
end

instantiation *cblinfun-wot* :: (*complex-normed-vector*, *complex-inner*) *topological-space* **begin**
lift-definition *open-cblinfun-wot* :: ‹(′*a*, ′*b*) *cblinfun-wot set* ⇒ *bool*› **is** ‹*openin cweak-operator-topology*› ⟨*proof*⟩
instance
⟨*proof*⟩
end

lemma *transfer-nhds-cweak-operator-topology*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*cr-cblinfun-wot* ===> *rel-filter cr-cblinfun-wot*) (*nhdsin cweak-operator-topology*) *nhds*›
⟨*proof*⟩

lemma *limitin-cweak-operator-topology*:
‹*limitin cweak-operator-topology f l F* ⟷ (∀ *a b*. ((λ*i*. $a \cdot_C (f\ i *_V b)$) ⟶ $a \cdot_C (l *_V b)$) *F*)›
⟨*proof*⟩

lemma *filterlim-cweak-operator-topology*: ‹*filterlim f* (*nhdsin cweak-operator-topology l*) = *limitin cweak-operator-topology f l*›
⟨*proof*⟩

instance *cblinfun-wot* :: (*complex-normed-vector*, *complex-inner*) *t2-space*
⟨*proof*⟩

lemma *Domainp-cr-cblinfun-wot*[*simp*]: ‹*Domainp cr-cblinfun-wot* = (λ-. *True*)›
⟨*proof*⟩

lemma *Rangep-cr-cblinfun-wot*[*simp*]: ‹*Rangep cr-cblinfun-wot* = (λ-. *True*)›
⟨*proof*⟩

lemma *transfer-euclidean-cweak-operator-topology*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-topology cr-cblinfun-wot*) *cweak-operator-topology euclidean*›

⟨*proof*⟩

lemma *openin-cweak-operator-topology*: ‹*openin cweak-operator-topology U* ⟷ (∃ *V*. *open V* ∧ *U* = (λ*a* (*x*,*y*). *cinner x* (*a* $*_V$ *y*)) −‘ *V*)›
⟨*proof*⟩

lemma *cweak-operator-topology-plus-cont*: ‹*LIM* (*x*,*y*) *nhdsin cweak-operator-topology a* $\times_F$ *nhdsin cweak-operator-topology b*.
x + *y* :> *nhdsin cweak-operator-topology* (*a* + *b*)›
⟨*proof*⟩

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

lemma *continuous-map-left-comp-wot*:
‹*continuous-map cweak-operator-topology cweak-operator-topology* (λ*a*::'*a*::*complex-normed-vector* $\Rightarrow_{CL}$ -. *b* o_{CL} *a*)›
for *b* :: ‹'*b*::*chilbert-space* $\Rightarrow_{CL}$ '*c*::*complex-inner*›
⟨*proof*⟩

lemma *continuous-map-scaleC-wot*: ‹*continuous-map cweak-operator-topology cweak-operator-topology*
(*scaleC c* :: ('*a*::*complex-normed-vector* $\Rightarrow_{CL}$ '*b*::*chilbert-space*) ⇒ -)›
⟨*proof*⟩

lemma *continuous-scaleC-wot*: ‹*continuous-on X* (*scaleC c* :: (-::*complex-normed-vector*,-::*chilbert-space*) *cblinfun-wot* ⇒ -)›
⟨*proof*⟩

lemma *wot-closure-is-csubspace*[*simp*]:
fixes *A*::('*a*::*complex-normed-vector*, '*b*::*chilbert-space*) *cblinfun-wot set*
assumes ‹*csubspace A*›
shows ‹*csubspace* (*closure A*)›
⟨*proof*⟩
include *lattice-syntax*
⟨*proof*⟩

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-wot* ===> (=)) *csubspace csubspace*›
⟨*proof*⟩

lemma [*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-wot* ===> (=)) (*closedin cweak-operator-topology*) *closed*›
⟨*proof*⟩

lemma [*transfer-rule*]:

includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-wot* ===> *rel-set cr-cblinfun-wot*) (*Abstract-Topology.closure-of cweak-operator-topology*) *closure*›
⟨*proof*⟩

lemma *wot-closure-is-csubspace′*[*simp*]:
fixes *A*::(′*a*::*complex-normed-vector* $\Rightarrow_{CL}$ ′*b*::*chilbert-space*) *set*
assumes ‹*csubspace A*›
shows ‹*csubspace* (*cweak-operator-topology closure-of A*)›
⟨*proof*⟩

lemma *has-sum-closed-cweak-operator-topology*:
fixes *A* :: ‹(′*b*::*complex-normed-vector* $\Rightarrow_{CL}$ ′*c*::*complex-inner*) *set*›
assumes *aA*: ‹⋀*i*. *a i* ∈ *A*›
assumes *closed*: ‹*closedin cweak-operator-topology A*›
assumes *subspace*: ‹*csubspace A*›
assumes *has-sum*: ‹⋀φ ψ. ((λ*i*. φ $\bullet_C$ (*a i* $*_V$ ψ)) *has-sum* φ $\bullet_C$ (*b* $*_V$ ψ)) *I*›
shows ‹*b* ∈ *A*›
⟨*proof*⟩

lemma *limitin-adj-wot*:
assumes ‹*limitin cweak-operator-topology f l F*›
shows ‹*limitin cweak-operator-topology* (λ*i*. (*f i*)∗) (*l*∗) *F*›
⟨*proof*⟩

lemma *hausdorff-cweak-operator-topology*[*simp*]: ‹*Hausdorff-space cweak-operator-topology*›
⟨*proof*⟩

lemma *hermitian-limit-hermitian-wot*:
assumes ‹*F* ≠ *bot*›
assumes *herm*: ‹⋀*i*. (*a i*)∗ = *a i*›
assumes *lim*: ‹*limitin cweak-operator-topology a A F*›
shows ‹*A*∗ = *A*›
⟨*proof*⟩

lemma *wot-weaker-than-sot-openin*:
‹*openin cweak-operator-topology x* ⟹ *openin cstrong-operator-topology x*›
⟨*proof*⟩

lemma *wot-weaker-than-sot-limitin*: ‹*limitin cweak-operator-topology a A F*› **if** ‹*limitin cstrong-operator-topology a A F*›
⟨*proof*⟩

lemma *hermitian-limit-hermitian-sot*:
assumes ‹*F* ≠ *bot*›
assumes ‹⋀*i*. (*a i*)∗ = *a i*›
assumes ‹*limitin cstrong-operator-topology a A F*›
shows ‹*A*∗ = *A*›

⟨*proof*⟩

lemma *hermitian-sum-hermitian-sot*:
assumes *herm*: ‹$\bigwedge i.$ (*a i*)∗ = *a i*›
assumes *sum*: ‹*has-sum-in cstrong-operator-topology a X A*›
shows ‹*A*∗ = *A*›
⟨*proof*⟩

lemma *wot-is-norm-topology-findim*[*simp*]:
‹(*cweak-operator-topology* :: ('*a*::{*cfinite-dim*,*chilbert-space*} $\Rightarrow_{CL}$ '*b*::{*cfinite-dim*,*chilbert-space*}) *topology*) = *euclidean*›
⟨*proof*⟩

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

‹(*cstrong-operator-topology* :: ('*a*::{*cfinite-dim*,*chilbert-space*} $\Rightarrow_{CL}$ '*b*::{*cfinite-dim*,*chilbert-space*}) *topology*) = *euclidean*›
⟨*proof*⟩

lemma *regular-space-wot*: ‹*regular-space cweak-operator-topology*›
⟨*proof*⟩

instance *cblinfun-wot* :: (*complex-normed-vector*, *complex-inner*) *t3-space*
⟨*proof*⟩

instantiation *cblinfun-wot* :: (*chilbert-space*, *chilbert-space*) *order* **begin**
lift-definition *less-eq-cblinfun-wot* :: ‹('*a*, '*b*) *cblinfun-wot* ⇒ ('*a*, '*b*) *cblinfun-wot* ⇒ *bool*› **is** *less-eq*⟨*proof*⟩
lift-definition *less-cblinfun-wot* :: ‹('*a*, '*b*) *cblinfun-wot* ⇒ ('*a*, '*b*) *cblinfun-wot* ⇒ *bool*› **is** *less*⟨*proof*⟩
instance
⟨*proof*⟩
end

instance *cblinfun-wot* :: (*chilbert-space*,*chilbert-space*) *ordered-comm-monoid-add*
⟨*proof*⟩

lemma *limitin-wot-add*:
assumes ‹*limitin cweak-operator-topology f a F*›
assumes ‹*limitin cweak-operator-topology g b F*›
shows ‹*limitin cweak-operator-topology* (λ*x*. *f x* + *g x*) (*a* + *b*) *F*›
⟨*proof*⟩

lemma *monotone-convergence-wot*:
— [1], Proposition 43.1 (i), (ii), but translated to filters.

fixes *f* :: ‹*'b* ⇒ (*'a* ⇒$_{CL}$ *'a::chilbert-space*)›
assumes *bounded*: ‹∀$_F$ *x in F*. *f x* ≤ *B*›
assumes *increasing*: ‹*increasing-filter* (*filtermap f F*)›
shows ‹∃ *L*. *limitin cweak-operator-topology f L F*›
⟨*proof*⟩

lemma *summable-wot-boundedI*:
fixes *f* :: ‹*'b* ⇒ (*'a* ⇒$_{CL}$ *'a::chilbert-space*)›
assumes *bounded*: ‹⋀*F*. *finite F* ⟹ *F* ⊆ *X* ⟹ *sum f F* ≤ *B*›
assumes *pos*: ‹⋀*x*. *x* ∈ *X* ⟹ *f x* ≥ *0*›
shows ‹*summable-on-in cweak-operator-topology f X*›
⟨*proof*⟩

lemma *summable-wot-boundedI'*:
fixes *f* :: ‹*'b* ⇒ (*'a::chilbert-space*, *'a*) *cblinfun-wot*›
assumes *bounded*: ‹⋀*F*. *finite F* ⟹ *F* ⊆ *X* ⟹ *sum f F* ≤ *B*›
assumes *pos*: ‹⋀*x*. *x* ∈ *X* ⟹ *f x* ≥ *0*›
shows ‹*f summable-on X*›
⟨*proof*⟩

lemma *has-sum-mono-neutral-wot*:
fixes *f* :: *'a* ⇒ (*'b::chilbert-space* ⇒$_{CL}$ *'b*)
assumes ‹*has-sum-in cweak-operator-topology f A a*› **and** *has-sum-in cweak-operator-topology g B b*
assumes ‹⋀*x*. *x* ∈ *A*∩*B* ⟹ *f x* ≤ *g x*›
assumes ‹⋀*x*. *x* ∈ *A*−*B* ⟹ *f x* ≤ *0*›
assumes ‹⋀*x*. *x* ∈ *B*−*A* ⟹ *g x* ≥ *0*›
shows *a* ≤ *b*
⟨*proof*⟩

lemma *has-sum-mono-wot*:
fixes *f* :: *'a* ⇒ (*'b::chilbert-space* ⇒$_{CL}$ *'b*)
assumes *has-sum-in cweak-operator-topology f A x* **and** *has-sum-in cweak-operator-topology g A y*
assumes ‹⋀*x*. *x* ∈ *A* ⟹ *f x* ≤ *g x*›
shows *x* ≤ *y*
⟨*proof*⟩

lemma *infsum-mono-neutral-wot*:
fixes *f* :: *'a* ⇒ (*'b::chilbert-space* ⇒$_{CL}$ *'b*)
assumes *summable-on-in cweak-operator-topology f A* **and** *summable-on-in cweak-operator-topology g B*
assumes ‹⋀*x*. *x* ∈ *A*∩*B* ⟹ *f x* ≤ *g x*›
assumes ‹⋀*x*. *x* ∈ *A*−*B* ⟹ *f x* ≤ *0*›

assumes ‹⋀*x*. *x* ∈ *B*−*A* ⟹ *g x* ≥ *0*›
shows *infsum-in cweak-operator-topology f A* ≤ *infsum-in cweak-operator-topology g B*
⟨*proof*⟩

lemma *has-sum-on-wot-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(((=) ===> *cr-cblinfun-wot*) ===> (=) ===> *cr-cblinfun-wot* ===> (⟷)) (*has-sum-in cweak-operator-topology*) *HAS-SUM*›
⟨*proof*⟩

lemma *summable-on-wot-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(((=) ===> *cr-cblinfun-wot*) ===> (=) ===> (⟷)) (*summable-on-in cweak-operator-topology*) (*summable-on*)›
⟨*proof*⟩

lemma *Abs-cblinfun-wot-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹((=) ===> *cr-cblinfun-wot*) *id Abs-cblinfun-wot*›
⟨*proof*⟩

lemma *infsum-mono-neutral-wot′*:
fixes *f* :: *′a* ⇒ (*′b*::*chilbert-space*, *′b*) *cblinfun-wot*
assumes *f summable-on A* **and** *g summable-on B*
assumes ‹⋀*x*. *x* ∈ *A*∩*B* ⟹ *f x* ≤ *g x*›
assumes ‹⋀*x*. *x* ∈ *A*−*B* ⟹ *f x* ≤ *0*›
assumes ‹⋀*x*. *x* ∈ *B*−*A* ⟹ *g x* ≥ *0*›
shows *infsum f A* ≤ *infsum g B*
⟨*proof*⟩

lemma *infsum-nonneg-wot′*:
fixes *f* :: *′a* ⇒ (*′c*::*chilbert-space*,*′c*) *cblinfun-wot*
assumes ⋀*x*. *x* ∈ *M* ⟹ *0* ≤ *f x*
shows *infsum f M* ≥ *0*
⟨*proof*⟩

lemma *summable-on-Sigma-wotI*:
fixes *f* :: ‹*′a* × *′b* ⇒ (*′c*::*chilbert-space*,*′c*) *cblinfun-wot*›
assumes ‹⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *B x* ⟹ *f* (*x*,*y*) ≥ *0*›
assumes *summableA*: ‹(λ*x*. $\sum_\infty$*y*∈*B x*. *f* (*x*,*y*)) *summable-on A*›
assumes *summableB*: ‹⋀*x*. *x*∈*A* ⟹ (λ*y*. *f* (*x*, *y*)) *summable-on* (*B x*)›
shows ‹*f summable-on Sigma A B*›
⟨*proof*⟩

lift-definition *compose-wot* :: ‹(*′b*::*complex-inner*,*′c*::*complex-inner*) *cblinfun-wot* ⇒ (*′a*::*complex-normed-vector*,*′b*) *cblinfun-wot* ⇒ (*′a*,*′c*) *cblinfun-wot*› **is**
cblinfun-compose⟨*proof*⟩

lift-definition *adj-wot* :: ‹(*'a*::*chilbert-space*, *'b*::*complex-inner*) *cblinfun-wot* ⇒ (*'b*, *'a*) *cblinfun-wot*› **is** *adj*⟨*proof*⟩

lemma *infsum-wot-is-Sup*:
  fixes *f* :: ‹*'b* ⇒ (*'a* $\Rightarrow_{CL}$ *'a*::*chilbert-space*)›
  assumes *summable*: ‹*summable-on-in cweak-operator-topology f X*›
    — See also *summable-wot-boundedI* for proving this.
  assumes *pos*: ‹⋀*x*. *x* ∈ *X* ⟹ *f x* ≥ *0*›
  defines ‹*S* ≡ *infsum-in cweak-operator-topology f X*›
  shows ‹*is-Sup* ((λ*F*. ∑*x*∈*F*. *f x*) ‘ {*F*. *finite F* ∧ *F* ⊆ *X*}) *S*›
⟨*proof*⟩

lemma *has-sum-in-cweak-operator-topology-pointwise*:
  ‹*has-sum-in cweak-operator-topology f X s* ⟷ (∀ *ψ* *φ*. ((λ*x*. *ψ* $\bullet_C$ *f x φ*) *has-sum* *ψ* $\bullet_C$ *s φ*) *X*)›
  ⟨*proof*⟩

lemma *summable-wot-bdd-above*:
  fixes *f* :: ‹*'b* ⇒ (*'a* $\Rightarrow_{CL}$ *'a*::*chilbert-space*)›
  assumes *summable*: ‹*summable-on-in cweak-operator-topology f X*›
    — See also *summable-wot-boundedI* for proving this.
  assumes *pos*: ‹⋀*x*. *x* ∈ *X* ⟹ *f x* ≥ *0*›
  shows ‹*bdd-above* (*sum f* ‘ {*F*. *finite F* ∧ *F* ⊆ *X*})›
  ⟨*proof*⟩

lemma *summable-on-in-cweak-operator-topology-pointwise*:
  assumes ‹*summable-on-in cweak-operator-topology f X*›
  shows ‹(λ*x*. *a* $\bullet_C$ *f x b*) *summable-on X*›
  ⟨*proof*⟩

lemma *infsum-in-cweak-operator-topology-pointwise*:
  assumes ‹*summable-on-in cweak-operator-topology f X*›
  shows ‹*a* $\bullet_C$ (*infsum-in cweak-operator-topology f X*) *b* = ($\sum_\infty$*x*∈*X*. *a* $\bullet_C$ *f x b*)›
  ⟨*proof*⟩

instance *cblinfun-wot* :: (*complex-normed-vector*, *complex-inner*) *topological-ab-group-add*
  ⟨*proof*⟩

lemma *has-sum-in-wot-compose-left*:
  fixes *f* :: ‹*'c* ⇒ *'a*::*complex-normed-vector* $\Rightarrow_{CL}$ *'b*::*chilbert-space*›
  assumes ‹*has-sum-in cweak-operator-topology f X s*›
  shows ‹*has-sum-in cweak-operator-topology* (λ*x*. *a* o_{CL} *f x*) *X* (*a* o_{CL} *s*)›
⟨*proof*⟩

lemma *has-sum-in-wot-compose-right*:
  fixes *f* :: ‹*'c* ⇒ *'a*::*complex-normed-vector* $\Rightarrow_{CL}$ *'b*::*complex-inner*›
  assumes ‹*has-sum-in cweak-operator-topology f X s*›
  shows ‹*has-sum-in cweak-operator-topology* (λ*x*. *f x* o_{CL} *a*) *X* (*s* o_{CL} *a*)›

⟨proof⟩

lemma *summable-on-in-wot-compose-left*:
fixes f :: ‹$'c \Rightarrow$ $'a$::*complex-normed-vector* $\Rightarrow_{CL}$ $'b$::*chilbert-space*›
assumes ‹*summable-on-in cweak-operator-topology f X*›
shows ‹*summable-on-in cweak-operator-topology* (λx. *a* o_{CL} *f x*) *X*›
⟨proof⟩

lemma *summable-on-in-wot-compose-right*:
assumes ‹*summable-on-in cweak-operator-topology f X*›
shows ‹*summable-on-in cweak-operator-topology* (λx. *f x* o_{CL} *a*) *X*›
⟨proof⟩

lemma *infsum-in-wot-compose-left*:
fixes f :: ‹$'c \Rightarrow$ $'a$::*complex-normed-vector* $\Rightarrow_{CL}$ $'b$::*chilbert-space*›
assumes ‹*summable-on-in cweak-operator-topology f X*›
shows ‹*infsum-in cweak-operator-topology* (λx. *a* o_{CL} *f x*) *X* = *a* o_{CL} (*infsum-in cweak-operator-topology f X*)›
⟨proof⟩

lemma *infsum-in-wot-compose-right*:
fixes f :: ‹$'c \Rightarrow$ $'a$::*complex-normed-vector* $\Rightarrow_{CL}$ $'b$::*complex-inner*›
assumes ‹*summable-on-in cweak-operator-topology f X*›
shows ‹*infsum-in cweak-operator-topology* (λx. *f x* o_{CL} *a*) *X* = (*infsum-in cweak-operator-topology f X*) o_{CL} *a*›
⟨proof⟩

lemma *infsum-wot-boundedI*:
fixes f :: ‹$'b \Rightarrow$ ($'a$ $\Rightarrow_{CL}$ $'a$::*chilbert-space*)›
assumes *bounded*: ‹$\bigwedge F$. *finite F* $\Longrightarrow$ $F \subseteq X$ $\Longrightarrow$ *sum f F* $\leq B$›
assumes *pos*: ‹$\bigwedge x$. $x \in X$ $\Longrightarrow$ *f x* $\geq$ *0*›
shows ‹*infsum-in cweak-operator-topology f X* $\leq B$›
⟨proof⟩

lemma *summable-imp-wot-summable*:
assumes ‹*f summable-on A*›
shows ‹*summable-on-in cweak-operator-topology f A*›
⟨proof⟩

lemma *triangle-ineq-wot*:
assumes ‹*f abs-summable-on A*›
shows ‹*norm* (*infsum-in cweak-operator-topology f A*) $\leq$ ($\sum_\infty x \in A$. *norm* (*f x*))›
⟨proof⟩

unbundle *no cblinfun-syntax*

end

6 *Misc-Tensor-Product-TTS* – Miscelleanous 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-L2

    Misc-Tensor-Product
    With-Type.With-Type
begin

unbundle *lattice-syntax* **and** *cblinfun-syntax*

6.1 Retrieving axioms

⟨*ML*⟩

6.2 Auxiliary lemmas

named-theorems *unoverload-def*

locale *local-typedef* = **fixes** *S* ::′*b set* **and** *s*::′*s itself*
  assumes *Ex-type-definition-S*: ∃ (*Rep*::′*s* ⇒ ′*b*) (*Abs*::′*b* ⇒ ′*s*). *type-definition Rep Abs S*
begin
definition *Rep* = *fst* (*SOME* (*Rep*::′*s* ⇒ ′*b*, *Abs*). *type-definition Rep Abs S*)
definition *Abs* = *snd* (*SOME* (*Rep*::′*s* ⇒ ′*b*, *Abs*). *type-definition Rep Abs S*)
lemma *type-definition-S*: *type-definition Rep Abs S*
  ⟨*proof*⟩
lemma *rep-in-S*[*simp*]: *Rep x* ∈ *S*
  and *rep-inverse*[*simp*]: *Abs* (*Rep x*) = *x*
  and *Abs-inverse*[*simp*]: *y* ∈ *S* ⟹ *Rep* (*Abs y*) = *y*
  ⟨*proof*⟩
definition *cr-S* **where** *cr-S* ≡ λ*s b*. *s* = *Rep b*
lemma *Domainp-cr-S*[*transfer-domain-rule*]: *Domainp cr-S* = (λ*x*. *x* ∈ *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*∈*S* ⟹ *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* :: ‹-::*cbanach ccsubspace*›
⟨*proof*⟩

definition ‹*transfer-ball-range A P* ⟷ (∀*f*. *range f* ⊆ *A* ⟶ *P f*)›

lemma *transfer-ball-range-parametric'*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*, *simp*]: ‹*right-unique T*› ‹*bi-total T*› ‹*bi-unique U*›
shows ‹(*rel-set U* ===> ((*T* ===> *U*) ===> (⟶)) ===> (⟶)) *transfer-ball-range transfer-ball-range*›
⟨*proof*⟩

lemma *transfer-ball-range-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*, *simp*]: ‹*bi-unique T*› ‹*bi-total T*› ‹*bi-unique U*›
shows ‹(*rel-set U* ===> ((*T* ===> *U*) ===> (⟷)) ===> (⟷)) *transfer-ball-range transfer-ball-range*›
⟨*proof*⟩

definition ‹*transfer-Times A B* = *A* × *B*›

lemma *transfer-Times-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set T* ===> *rel-set U* ===> *rel-set* (*rel-prod T U*)) *transfer-Times transfer-Times*›
⟨*proof*⟩

lemma *csubspace-nonempty*: ‹*csubspace X* ⟹ *X* ≠ {}›
⟨*proof*⟩

definition ‹*transfer-vimage-into f U s* = (*f* −‘ *U*) ∩ *s*›

lemma *transfer-vimage-into-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*› ‹*bi-unique B*›
shows ‹((*A* ===> *B*) ===> *rel-set B* ===> *rel-set A* ===> *rel-set A*) *transfer-vimage-into transfer-vimage-into*›
⟨*proof*⟩

lemma *make-parametricity-proof-friendly*:
shows ‹(∀x. P ⟶ Q x) ⟷ (P ⟶ (∀x. Q x))›
and ‹(∀x. x ∈ S ⟶ Q x) ⟷ (∀x∈S. Q x)›
and ‹(∀x⊆S. R x) ⟷ (∀x∈Pow S. R x)›
and ‹{x∈S. Q x} = Set.filter Q S›
and ‹{x. x ⊆ S ∧ R x} = Set.filter R (Pow S)›
and ‹⋀P. (∀f. range f ⊆ A ⟶ P f) = transfer-ball-range A P›
and ‹⋀A B. A × B = transfer-Times A B›
and ‹⋀B P. (∃A⊆B. P A) ⟷ (∃A∈Pow B. P A)›
and ‹⋀f U s. (f −‘ U) ∩ s = transfer-vimage-into f U s›
and ‹⋀M B. ⨅M ⊓ principal B = transfer-bounded-filter-Inf B M›
and ‹⋀F M. F ⊓ principal M = transfer-inf-principal F M›
⟨*proof*⟩

6.3 *plus*

locale *plus-ow* =
fixes *U plus*
assumes ‹∀x∈U. ∀y∈U. plus x y ∈ U›
lemma *plus-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set* A ===> (A ===> A ===> A) ===> (=))
plus-ow plus-ow›
⟨*proof*⟩

6.3.1 *minus*

locale *minus-ow* = **fixes** *U minus* **assumes** ‹∀x∈U. ∀y∈U. minus x y ∈ U›

lemma *minus-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set* A ===> (A ===> A ===> A) ===> (=))
minus-ow minus-ow›
⟨*proof*⟩

6.3.2 *uminus*

locale *uminus-ow* = **fixes** *U uminus* **assumes** ‹∀x∈U. uminus x ∈ U›

lemma *uminus-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set* A ===> (A ===> A) ===> (=))
uminus-ow uminus-ow›
⟨*proof*⟩

6.4 *semigroup*

locale *semigroup-ow* = *plus-ow U plus* **for** *U plus* +
  assumes ‹∀ *x*∈*U*. ∀ *y*∈*U*. ∀ *z*∈*U*. *plus x* (*plus y z*) = *plus* (*plus x y*) *z*›

lemma *semigroup-ow-parametricity*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-unique A*›
  shows ‹(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (=))
    semigroup-ow semigroup-ow›
  ⟨*proof*⟩

lemma *semigroup-ow-typeclass*[*simp*, *iff*]: ‹*semigroup-ow V* (+)›
  if ‹⋀*x y*. *x*∈*V* ⟹ *y*∈*V* ⟹ *x* + *y* ∈ *V*› **for** *V* :: ‹′*a* :: *semigroup-add set*›
  ⟨*proof*⟩

lemma *class-semigroup-add-ud*[*unoverload-def*]: ‹*class.semigroup-add* = *semigroup-ow UNIV*›
  ⟨*proof*⟩

6.5 *abel-semigroup*

locale *abel-semigroup-ow* = *semigroup-ow U plus* **for** *U plus* +
  assumes ‹∀ *x*∈*U*. ∀ *y*∈*U*. *plus x y* = *plus y x*›

lemma *abel-semigroup-ow-parametric*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-unique A*›
  shows ‹(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (=))
    abel-semigroup-ow abel-semigroup-ow›
  ⟨*proof*⟩

lemma *abel-semigroup-ow-typeclass*[*simp*, *iff*]: ‹*abel-semigroup-ow V* (+)›
  if ‹⋀*x y*. *x*∈*V* ⟹ *y*∈*V* ⟹ *x* + *y* ∈ *V*› **for** *V* :: ‹′*a* :: *ab-semigroup-add set*›
  ⟨*proof*⟩

lemma *class-ab-semigroup-add-ud*[*unoverload-def*]: ‹*class.ab-semigroup-add* = *abel-semigroup-ow UNIV*›
  ⟨*proof*⟩

6.6 *comm-monoid*

locale *comm-monoid-ow* = *abel-semigroup-ow U plus* **for** *U plus* +
  fixes *zero*
  assumes ‹*zero* ∈ *U*›
  assumes ‹∀ *x*∈*U*. *plus x zero* = *x*›

lemma *comm-monoid-ow-parametric*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-unique A*›
  shows ‹(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> *A* ===> (=))

comm-monoid-ow comm-monoid-ow›
⟨*proof*⟩

lemma *comm-monoid-ow-typeclass*[*simp*, *iff*]: ‹*comm-monoid-ow V* (+) *0*›
if ‹*0* ∈ *V*› **and** ‹⋀*x y*. *x*∈*V* ⟹ *y*∈*V* ⟹ *x* + *y* ∈ *V*› **for** *V* :: ‹'*a* :: *comm-monoid-add set*›
⟨*proof*⟩

lemma *class-comm-monoid-add-ud*[*unoverload-def*]: ‹*class.comm-monoid-add* = *comm-monoid-ow UNIV*›
⟨*proof*⟩

6.7 *topological-space*

locale *topological-space-ow* =
fixes *U open*
assumes ‹*open U*›
assumes ‹∀ *S*⊆*U*. ∀ *T*⊆*U*. *open S* ⟶ *open T* ⟶ *open* (*S* ∩ *T*)›
assumes ∀ *K*⊆*Pow U*. (∀ *S*∈*K*. *open S*) ⟶ *open* (⋃ *K*)

lemma *topological-space-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> (*rel-set A* ===> (=)) ===> (=))
topological-space-ow topological-space-ow›
⟨*proof*⟩

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

lemma *topological-space-ow-from-topology*[*simp*]: ‹*topological-space-ow* (*topspace T*) (*openin T*)›
⟨*proof*⟩

6.8 *sum*

definition ‹*sum-ow z plus f 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*]: ‹*bi-unique T*› ‹*bi-unique U*›
shows ‹(*T* ===> (*V* ===> *T* ===> *T*) ===> (*U* ===> *V*) ===> *rel-set U* ===> *T*)
sum-ow sum-ow›
⟨*proof*⟩

lemma (**in** *comm-monoid-set*) *comp-fun-commute-onI*: ‹*Finite-Set.comp-fun-commute-on UNIV* ((∗) ∘ *g*)›

⟨*proof*⟩

lemma (**in** *comm-monoid-set*) *F-via-the-default*: ‹*F g A* = *the-default def* (*Collect* (*fold-graph* ((∗) ∘ *g*) **1** *A*))›
if ‹*finite A*›
⟨*proof*⟩

lemma *sum-ud*[*unoverload-def*]: ‹*sum* = *sum-ow 0 plus*›
⟨*proof*⟩

6.9 *t2-space*

locale *t2-space-ow* = *topological-space-ow* +
assumes ‹∀ *x*∈*U*. ∀ *y*∈*U*. *x* ≠ *y* ⟶ (∃ *S*⊆*U*. ∃ *T*⊆*U*. *open S* ∧ *open T* ∧ *x* ∈ *S* ∧ *y* ∈ *T* ∧ *S* ∩ *T* = {})›

lemma *t2-space-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> (*rel-set A* ===> (=)) ===> (=))
t2-space-ow t2-space-ow›
⟨*proof*⟩

lemma *class-t2-space-ud*[*unoverload-def*]: ‹*class.t2-space* = *t2-space-ow UNIV*›
⟨*proof*⟩

lemma *t2-space-ow-from-topology*[*simp*, *iff*]: ‹*t2-space-ow* (*topspace T*) (*openin T*)› **if** ‹*Hausdorff-space T*›
⟨*proof*⟩

6.9.1 *continuous-on*

definition *continuous-on-ow* **where** ‹*continuous-on-ow A B opnA opnB s f*
⟷ (∀ *U*⊆*B*. *opnB U* ⟶ (∃ *V*⊆*A*. *opnA V* ∧ (*V* ∩ *s*) = (*f* −‘ *U*) ∩ *s*))›
for *f* :: ‹'*a* ⇒ '*b*›

lemma *continuous-on-ud*[*unoverload-def*]: ‹*continuous-on s f* ⟷ *continuous-on-ow UNIV UNIV open open s f*›
for *f* :: ‹'*a*::*topological-space* ⇒ '*b*::*topological-space*›
⟨*proof*⟩

lemma *continuous-on-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*› ‹*bi-unique B*›
shows ‹(*rel-set A* ===> *rel-set B* ===> (*rel-set A* ===> (⟷)) ===> (*rel-set B* ===> (⟷)) ===> *rel-set A* ===> (*A* ===> *B*) ===> (⟷)) *continuous-on-ow continuous-on-ow*›
⟨*proof*⟩

6.10 *scaleR*

locale *scaleR-ow* =
  fixes *U* **and** *scaleR* :: ‹*real* ⇒ ′*a* ⇒ ′*a*›
  assumes *scaleR-closed*: ‹∀ *a* ∈ *U*. *scaleR r a* ∈ *U*›

lemma *scaleR-ow-typeclass*[*simp*]: ‹*scaleR-ow UNIV scaleR*› **for** *scaleR*
  ⟨*proof*⟩

lemma *scaleR-ow-parametric*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-unique A*›
  shows ‹(*rel-set A* ===> ((=) ===> *A* ===> *A*) ===> (=))
    scaleR-ow scaleR-ow›
  ⟨*proof*⟩

6.11 *scaleC*

locale *scaleC-ow* = *scaleR-ow* +
  fixes *scaleC*
  assumes *scaleC-closed*: ‹∀ *a*∈*U*. *scaleC c a* ∈ *U*›
  assumes ‹∀ *a*∈*U*. *scaleR r a* = *scaleC* (*complex-of-real r*) *a*›

lemma *scaleC-ow-parametric*[*transfer-rule*]:
  includes *lifting-syntax*
  assumes [*transfer-rule*]: ‹*bi-unique A*›
  shows ‹(*rel-set A* ===> ((=) ===> *A* ===> *A*) ===> ((=) ===> *A* ===> *A*) ===> (=))
    scaleC-ow scaleC-ow›
  ⟨*proof*⟩

lemma *class-scaleC-ud*[*unoverload-def*]: ‹*class.scaleC* = *scaleC-ow UNIV*›
  ⟨*proof*⟩

6.12 *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 ‹∀ *a*∈*U*. *uminus a* ∈ *U*›
  assumes ∀ *a*∈*U*. *plus* (*uminus a*) *a* = *zero*
  assumes ∀ *a*∈*U*. ∀ *b*∈*U*. *minus a b* = *plus a* (*uminus b*)

lemma *ab-group-add-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*) ===> (=))
    ab-group-add-ow ab-group-add-ow›
  ⟨*proof*⟩

lemma *ab-group-add-ow-typeclass*[*simp*]:
‹*ab-group-add-ow V* (+) *0* (−) *uminus*›
if ‹*0* ∈ *V*› ‹∀ *x*∈*V*. −*x* ∈ *V*› ‹∀ *x*∈*V*. ∀ *y*∈*V*. *x* + *y* ∈ *V*›
for *V* :: ‹- :: *ab-group-add set*›
⟨*proof*⟩

lemma *class-ab-group-add-ud*[*unoverload-def*]: ‹*class.ab-group-add* = *ab-group-add-ow UNIV*›
⟨*proof*⟩

6.13 *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* ⇒ ′*a* ⇒ ′*a*
assumes
‹∀ *x*∈*U*. *scale a x* ∈ *U*›
∀ *x*∈*U*. ∀ *y*∈*U*. *scale a* (*plus x y*) = *plus* (*scale a x*) (*scale a y*)
∀ *x*∈*U*. *scale* (*a* + *b*) *x* = *plus* (*scale a x*) (*scale b x*)
∀ *x*∈*U*. *scale a* (*scale b x*) = *scale* (*a* ∗ *b*) *x*
∀ *x*∈*U*. *scale 1 x* = *x*

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

6.14 *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*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> ((=) ===> *A* ===> *A*) ===> ((=) ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===>
A ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A*) ===> (=))
complex-vector-ow complex-vector-ow›
⟨*proof*⟩

lemma *class-complex-vector-ud*[*unoverload-def*]: ‹*class.complex-vector* = *complex-vector-ow UNIV*›
⟨*proof*⟩

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

‹*vector-space-ow V* (+) *0* (−) *uminus* ($*_C$)›
if [*simp*]: ‹*csubspace V*›
for *V* :: ‹-::*complex-vector set*›
⟨*proof*⟩

lemma *complex-vector-ow-typeclass*[*simp*]:
‹*complex-vector-ow V* ($*_R$) ($*_C$) (+) *0* (−) *uminus*› **if** [*simp*]: ‹*csubspace V*›
⟨*proof*⟩

6.15 *open-uniformity*

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

lemma *open-uniformity-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> (*rel-set A* ===> (=)) ===> *rel-filter* (*rel-prod A A*) ===> (=))
open-uniformity-ow open-uniformity-ow›
⟨*proof*⟩

lemma *class-open-uniformity-ud*[*unoverload-def*]: ‹*class.open-uniformity* = *open-uniformity-ow UNIV*›
⟨*proof*⟩

lemma *open-uniformity-on-typeclass*[*simp*]:
fixes *V* :: ‹-::*open-uniformity set*›
assumes ‹*closed V*›
shows ‹*open-uniformity-ow V* (*openin* (*top-of-set V*)) (*uniformity-on V*)›
⟨*proof*⟩

6.16 *uniformity-dist*

locale *uniformity-dist-ow* = *dist dist* + *uniformity uniformity* **for** *U dist uniformity* +
assumes *uniformity-dist*: *uniformity* = $(\bigsqcap e \in \{0<..\}.$ *principal* $\{(x, y) \in U \times U.$ *dist x y* $< e\})$

lemma *class-uniformity-dist-ud*[*unoverload-def*]: ‹*class.uniformity-dist* = *uniformity-dist-ow UNIV*›
⟨*proof*⟩

lemma *uniformity-dist-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> *rel-filter* (*rel-prod A A*) ===> (=))
uniformity-dist-ow uniformity-dist-ow›
⟨*proof*⟩

lemma *uniformity-dist-on-typeclass*[*simp*]: ‹*uniformity-dist-ow V dist* (*uniformity-on V*)› **for** *V* :: ‹-::*uniformity-dist set*›
⟨*proof*⟩

6.17 *sgn*

locale *sgn-ow* =
fixes *U* **and** *sgn* :: ‹′*a* ⇒ ′*a*›
assumes *sgn-closed*: ‹∀ *a*∈*U*. *sgn a* ∈ *U*›

lemma *sgn-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> (*A* ===> *A*) ===> (=))
sgn-ow sgn-ow›
⟨*proof*⟩

6.18 *sgn-div-norm*

locale *sgn-div-norm-ow* = *scaleR-ow U scaleR* + *norm norm* + *sgn-ow U sgn* **for** *U sgn norm scaleR* +
assumes ∀ *x*∈*U*. *sgn x* = *scaleR* (*inverse* (*norm x*)) *x*

lemma *class-sgn-div-norm-ud*[*unoverload-def*]: ‹*class.sgn-div-norm* = *sgn-div-norm-ow UNIV*›
⟨*proof*⟩

lemma *sgn-div-norm-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> (*A* ===> *A*) ===> (*A* ===> (=)) ===> ((=) ===> *A* ===> *A*) ===> (=))
sgn-div-norm-ow sgn-div-norm-ow›
⟨*proof*⟩

lemma *sgn-div-norm-on-typeclass*[*simp*]:
fixes *V* :: ‹-::*sgn-div-norm set*›
assumes ‹$\bigwedge$*v r*. *v*∈*V* ⟹ *scaleR r v* ∈ *V*›
shows ‹*sgn-div-norm-ow V sgn norm* ($*_R$)›
⟨*proof*⟩

6.19 *dist-norm*

locale *dist-norm-ow* = *dist dist* + *norm norm* + *minus-ow U minus* **for** *U minus dist norm* +
assumes *dist-norm*: ∀ *x*∈*U*. ∀ *y*∈*U*. *dist x y* = *norm* (*minus x y*)

lemma *dist-norm-ud*[*unoverload-def*]: ‹*class.dist-norm* = *dist-norm-ow UNIV*›
⟨*proof*⟩

lemma *dist-norm-ow-parametric*[*transfer-rule*]:

includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> (=)) ===> (=))
dist-norm-ow dist-norm-ow›
⟨*proof*⟩

lemma *dist-norm-ow-typeclass*[*simp*]:
fixes *A* :: ‹-::*dist-norm set*›
assumes ‹$\bigwedge a\ b.\ [\![\ a \in A;\ b \in A\ ]\!] \Longrightarrow a - b \in A$›
shows ‹*dist-norm-ow A* (−) *dist norm*›
⟨*proof*⟩

6.20 *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 $\forall x{\in}U.\ \forall y{\in}U.$ *cinner x y* = *cnj* (*cinner y x*)
and $\forall x{\in}U.\ \forall y{\in}U.\ \forall z{\in}U.$ *cinner* (*plus x y*) *z* = *cinner x z* + *cinner y z*
and $\forall x{\in}U.\ \forall y{\in}U.$ *cinner* (*scaleC r x*) *y* = *cnj r* ∗ *cinner x y*
and $\forall x{\in}U.$ *0* ≤ *cinner x x*
and $\forall x{\in}U.$ *cinner x x* = *0* ⟷ *x* = *zero*
and $\forall x{\in}U.$ *norm x* = *sqrt* (*cmod* (*cinner x x*))

lemma *class-complex-inner-ud*[*unoverload-def*]: ‹*class.complex-inner* = *complex-inner-ow UNIV*›
⟨*proof*⟩

lemma *complex-inner-ow-parametricity*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique T*›
shows ‹(*rel-set T* ===> ((=) ===> *T* ===> *T*) ===> ((=) ===> *T* ===> *T*) ===> (*T* ===> *T* ===> *T*) ===> *T*
===> (*T* ===> *T* ===> *T*) ===> (*T* ===> *T*) ===> (*T* ===> *T* ===> (=)) ===> (*T* ===> (=))
===> (*T* ===> *T*) ===> *rel-filter* (*rel-prod T T*) ===> (*rel-set T* ===> (=))
===> (*T* ===> *T* ===> (=)) ===> (=)) *complex-inner-ow complex-inner-ow*›
⟨*proof*⟩

lemma *complex-inner-ow-typeclass*[*simp*]:
fixes *V* :: ‹-::*complex-inner set*›
assumes [*simp*]: ‹*closed V*› ‹*csubspace V*›
shows ‹*complex-inner-ow V* ($*_R$) ($*_C$) (+) *0* (−) *uminus dist norm sgn* (*uniformity-on V*) (*openin* (*top-of-set V*)) ($\cdot_C$)›
⟨*proof*⟩

6.21 *is-ortho-set*

definition *is-ortho-set-ow* **where** ‹*is-ortho-set-ow zero cinner S* ⟷
 ((∀ *x*∈*S*. ∀ *y*∈*S*. *x* ≠ *y* ⟶ *cinner x y* = *0*) ∧ *zero* ∉ *S*)›
 for *zero cinner*

lemma *is-ortho-set-ow-parametric*[*transfer-rule*]:
 includes *lifting-syntax*
 assumes [*transfer-rule*]: ‹*bi-unique A*›
 shows ‹(*A* ===> (*A* ===> *A* ===> (=)) ===> *rel-set A* ===> (=))
  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*⟩

6.22 *metric-space*

locale *metric-space-ow* = *uniformity-dist-ow U dist uniformity* + *open-uniformity-ow U open uniformity*
 for *U dist uniformity open* +
 assumes ∀ *x* ∈ *U*. ∀ *y* ∈ *U*. *dist x y* = *0* ⟷ *x* = *y*
  and ∀ *x*∈*U*. ∀ *y*∈*U*. ∀ *z*∈*U*. *dist x y* ≤ *dist x z* + *dist y z*

lemma *metric-space-ow-parametric*[*transfer-rule*]:
 includes *lifting-syntax*
 assumes [*transfer-rule*]: ‹*bi-unique A*›
 shows ‹(*rel-set A* ===> (*A* ===> *A* ===> (=)) ===> *rel-filter* (*rel-prod A A*) ===>
   (*rel-set A* ===> (=)) ===> (=))
  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*⟩

6.23 *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* ===> (*rel-set A* ===> (=)) ===> *A* ===> *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*]: ‹*nhds-ow* (*topspace T*) (*openin T*) *x* = *nhdsin T x*› **if** ‹*x* ∈ *topspace T*›
⟨*proof*⟩

6.24 *at-within*

definition ‹*at-within-ow U open a s* = *nhds-ow U open a* ⊓ *principal* (*s* − {*a*})›
for *U open a s*

lemma *at-within-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique T*›
shows ‹((*rel-set T*) ===> (*rel-set T* ===> (=)) ===> *T* ===> *rel-set T* ===> *rel-filter T*)
at-within-ow at-within-ow›
⟨*proof*⟩

lemma *at-within-ud*[*unoverload-def*]: ‹*at-within* = *at-within-ow UNIV open*›
⟨*proof*⟩

lemma *at-within-ow-topology*:
‹*at-within-ow* (*topspace T*) (*openin T*) *a S* = *nhdsin T a* ⊓ *principal* (*S* − {*a*})›
if ‹*a* ∈ *topspace T*›
⟨*proof*⟩

6.25 (*has-sum*)

definition ‹*has-sum-ow U plus zero open f A x* =
filterlim (*sum-ow zero plus f*) (*nhds-ow U* (λ*S*. *open S*) *x*)
(*finite-subsets-at-top A*)›
for *U plus zero open f A x*

lemma *has-sum-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique T*› ‹*bi-unique U*›
shows ‹(*rel-set T* ===> (*V* ===> *T* ===> *T*) ===> *T* ===> (*rel-set T* ===> (=)) ===> (*U* ===> *V*) ===> *rel-set U* ===> *T* ===> (=))
has-sum-ow has-sum-ow›
⟨*proof*⟩

lemma *has-sum-ud*[*unoverload-def*]: ‹*HAS-SUM* = *has-sum-ow UNIV plus* (*0*::′*a*::{*comm-monoid-add*,*topological-spa*
open›
⟨*proof*⟩

lemma *has-sum-ow-topology*:
assumes ‹*l* ∈ *topspace T*›
assumes ‹*0* ∈ *topspace T*›
assumes ‹⋀*x y*. *x* ∈ *topspace T* ⟹ *y* ∈ *topspace T* ⟹ *x* + *y* ∈ *topspace T*›
shows ‹*has-sum-ow* (*topspace T*) (+) *0* (*openin T*) *f S l* ⟷ *has-sum-in T f S l*›
⟨*proof*⟩

6.26 *filterlim*

6.27 *convergent*

definition *convergent-ow* **where**
‹*convergent-ow U open X* ⟷ (∃ *L*∈*U*. *filterlim X* (*nhds-ow U open L*) *sequentially*)›
for *U open*

lemma *convergent-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique T*›
shows ‹(*rel-set T* ===> (*rel-set T* ===> (=)) ===> ((=) ===> *T*) ===> (⟷))
convergent-ow convergent-ow›
⟨*proof*⟩

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

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

lemma *convergent-ow-topology*[*simp*]:
‹*convergent-ow* (*topspace T*) (*openin T*) *f* ⟷ (∃ *l*. *limitin T f l sequentially*)›
⟨*proof*⟩

lemma *convergent-ow-typeclass*[*simp*]:
‹*convergent-ow V* (*openin* (*top-of-set V*)) *f* ⟷ (∃ *l*. *limitin* (*top-of-set V*) *f l sequentially*)›
⟨*proof*⟩

6.28 *uniform-space.cauchy-filter*

lemma *cauchy-filter-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique T*
shows (*rel-filter* (*rel-prod T T*) ===> *rel-filter T* ===> (=))
uniform-space.cauchy-filter
uniform-space.cauchy-filter

⟨*proof*⟩

6.29 *uniform-space.Cauchy*

lemma *uniform-space-Cauchy-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: *bi-unique T*
shows (*rel-filter* (*rel-prod T T*) ===> ((=) ===> *T*) ===> (=))
uniform-space.Cauchy
uniform-space.Cauchy
⟨*proof*⟩

6.30 *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*⟩

6.31 *chilbert-space*

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

lemma *chilbert-space-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===> ((=) ===> *A* ===> *A*) ===> ((=) ===> *A* ===> *A*) ===> (*A* ===> *A* ===> *A*) ===>
A ===> (*A* ===> *A* ===> *A*) ===> (*A* ===> *A*) ===> (*A* ===> *A* ===> (=)) ===> (*A* ===> (=)) ===>
(*A* ===> *A*) ===> *rel-filter* (*rel-prod A A*) ===> (*rel-set A* ===> (=)) ===> (*A* ===> *A* ===> (=)) ===> (=))

chilbert-space-ow chilbert-space-ow›
⟨*proof*⟩

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

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

6.32 (*hull*)

definition ‹*hull-ow A S s* = ((λ*x*. *S x* ∧ *x* ⊆ *A*) *hull s*) ∩ *A*›

lemma *hull-ow-nondegenerate*: ‹*hull-ow A S s* = ((λ*x*. *S x* ∧ *x* ⊆ *A*) *hull s*)› **if** ‹*x* ⊆ *A*› **and** ‹*s* ⊆ *x*› **and** ‹*S x*›
⟨*proof*⟩

definition ‹*transfer-bounded-Inf B M* = *Inf M* ⊓ *B*›

lemma *transfer-bounded-Inf-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes ‹*bi-unique T*›
shows ‹(*rel-set T* ===> *rel-set* (*rel-set T*) ===> *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* ===> (*rel-set T* ===> (=)) ===> *rel-set T* ===> *rel-set T*)
hull-ow hull-ow
⟨*proof*⟩

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

6.33 *csubspace*

definition
‹*subspace-ow plus zero scale S* = (*zero* ∈ *S* ∧ (∀ *x*∈*S*. ∀ *y*∈*S*. *plus x y* ∈ *S*) ∧ (∀ *c*. ∀ *x*∈*S*. *scale c x* ∈ *S*))›
for *plus zero scale S*

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

includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique* *T*›
shows ‹((*T* ===> *T* ===> *T*) ===> *T* ===> ((=) ===> *T* ===> *T*) ===> *rel-set* *T* ===> (=))
subspace-ow subspace-ow›
⟨*proof*⟩

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

lemma *csubspace-ud*[*unoverload-def*]: ‹*csubspace* = *subspace-ow* (+) *0* ($*_C$)›
⟨*proof*⟩

6.34 *cspan*

definition
‹*span-ow U plus zero scale b* = *hull-ow U* (*subspace-ow plus zero scale*) *b*›
for *U plus zero scale b*

lemma *span-ow-on-typeclass*:
assumes ‹*csubspace U*›
assumes ‹*B* ⊆ *U*›
shows ‹*span-ow U plus 0 scaleC B* = *cspan B*›
⟨*proof*⟩

lemma (**in** *Modules.module*) *span-ud*[*unoverload-def*]: ‹*span* = *span-ow UNIV plus 0 scale*›
⟨*proof*⟩

lemmas *cspan-ud*[*unoverload-def*] = *complex-vector.span-ud*

lemma *span-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique* *T*›
shows ‹(*rel-set* *T* ===> (*T* ===> *T* ===> *T*) ===> *T* ===> ((=) ===> *T* ===> *T*) ===> *rel-set* *T* ===> *rel-set* *T*)
span-ow span-ow›
⟨*proof*⟩

6.34.1 (*islimpt*)

definition ‹*islimpt-ow U open x S* ⟷ (∀ *T*⊆*U*. *x*∈*T* ⟶ *open* *T* ⟶ (∃ *y*∈*S*. *y*∈*T* ∧ *y*≠*x*))›
for *open*

lemma *islimpt-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique* *T*›
shows ‹(*rel-set* *T* ===> (*rel-set* *T* ===> (=)) ===> *T* ===> *rel-set* *T* ===> (⟷))
islimpt-ow islimpt-ow›
⟨*proof*⟩

definition ‹*islimptin T x S ⟷ x ∈ topspace T ∧ (∀ V. x ∈ V ⟶ openin T V ⟶ (∃ y∈S. y ∈ V ∧ y ≠ x))*›

lemma *islimpt-ow-from-topology*: ‹*islimpt-ow* (*topspace T*) (*openin T*) *x S ⟷ islimptin T x S ∨ x ∉ topspace T*›
⟨*proof*⟩

6.34.2 *closure*

definition ‹*closure-ow U open S = S ∪ {x∈U. islimpt-ow U open x S}*› **for** *open*

lemma *closure-ow-with-typeclass*[*simp*]:
‹*closure-ow X* (*openin* (*top-of-set X*)) *S* = (*X ∩ closure* (*X ∩ S*)) *∪ S*›
⟨*proof*⟩

lemma *closure-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique T*›
shows ‹(*rel-set T* ===> (*rel-set T* ===> (=)) ===> *rel-set T* ===> *rel-set T*) *closure-ow closure-ow*›
⟨*proof*⟩

lemma *closure-ow-from-topology*: ‹*closure-ow* (*topspace T*) (*openin T*) *S = T closure-of S*› **if** ‹*S ⊆ topspace T*›
⟨*proof*⟩

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

6.35 *continuous*

lemma *continuous-on-ow-from-topology*: ‹*continuous-on-ow* (*topspace T*) (*topspace U*) (*openin T*) (*openin U*) (*topspace T*) *f ⟷ continuous-map T U f*›
if ‹*f ‘ topspace T ⊆ topspace U*›
⟨*proof*⟩

6.36 *is-onb*

definition
‹*is-onb-ow U scaleC plus zero norm open cinner E ⟷ is-ortho-set-ow zero cinner E ∧* (∀ *b∈E. norm b = 1*) ∧
closure-ow U open (*span-ow U plus zero scaleC E*) = *U*›
for *U scaleC plus zero norm open cinner*

lemma *is-onb-ow-parametric*[*transfer-rule*]:
includes *lifting-syntax*
assumes [*transfer-rule*]: ‹*bi-unique A*›
shows ‹(*rel-set A* ===>

((=) ===> A ===> A) ===>
(A ===> A ===> A) ===>
A ===>
(A ===> (=)) ===> (rel-set A ===> (=)) ===> (A ===> A ===> (=)) ===> rel-set A ===> (=))
is-onb-ow is-onb-ow›
⟨*proof*⟩

lemma *is-onb-ud*[*unoverload-def*]:
‹*is-onb* = *is-onb-ow UNIV scaleC plus 0 norm open cinner*›
⟨*proof*⟩

6.37 Transferring theorems

lemma *closure-of-eqI*:
fixes *f g* :: ‹′*a* ⇒ ′*b*› **and** *T* :: ‹′*a topology*› **and** *U* :: ‹′*b topology*›
assumes *hausdorff*: ‹*Hausdorff-space U*›
assumes *f-eq-g*: ‹⋀*x*. *x* ∈ *S* ⟹ *f x* = *g x*›
assumes *x*: ‹*x* ∈ *T 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* :: ‹′*a*::*chilbert-space set*›
assumes ‹*is-ortho-set S*› **and** *norm*: ‹⋀*x*. *x*∈*S* ⟹ *norm x* = *1*› **and** ‹*S* ⊆ *space-as-set V*›
shows ‹∃ *B*. *B* ⊇ *S* ∧ *is-ortho-set B* ∧ (∀ *x*∈*B*. *norm x* = *1*) ∧ *ccspan B* = *V*›
⟨*proof*⟩

lemma *has-sum-in-comm-additive-general*:
fixes *f* :: ‹′*a* ⇒ ′*b* :: *comm-monoid-add*›
and *g* :: ‹′*b* ⇒ ′*c* :: *comm-monoid-add*›
assumes *T0*[*simp*]: ‹*0* ∈ *topspace T*› **and** *Tplus*[*simp*]: ‹⋀*x y*. *x* ∈ *topspace T* ⟹ *y* ∈ *topspace T* ⟹ *x*+*y* ∈ *topspace T*›
assumes *Uplus*[*simp*]: ‹⋀*x y*. *x* ∈ *topspace U* ⟹ *y* ∈ *topspace U* ⟹ *x*+*y* ∈ *topspace U*›
assumes *grange*: ‹*g* ‘ *topspace T* ⊆ *topspace U*›
assumes *g0*: ‹*g 0* = *0*›
assumes *frange*: ‹*f* ‘ *S* ⊆ *topspace T*›
assumes *gcont*: ‹*filterlim g* (*nhdsin U* (*g l*)) (*atin T l*)›
assumes *gadd*: ‹⋀*x y*. *x* ∈ *topspace T* ⟹ *y* ∈ *topspace T* ⟹ *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* :: ‹′*a* ⇒ ′*b* :: *ab-group-add*›
and *g* :: ‹′*b* ⇒ ′*c* :: *ab-group-add*›
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*⟩

7 Stuff relying on the above lifting

definition ‹*some-onb-of X* = (*SOME B. is-ortho-set B* ∧ (∀ *b*∈*B. norm b* = *1*) ∧ *ccspan B* = *X*)›

lemma
fixes *X* :: ‹′*a*::*chilbert-space ccsubspace*›
shows *some-onb-of-is-ortho-set*[*iff*]: ‹*is-ortho-set* (*some-onb-of X*)›
and *some-onb-of-norm1*: ‹*b* ∈ *some-onb-of X* ⟹ *norm b* = *1*›
and *some-onb-of-ccspan*[*simp*]: ‹*ccspan* (*some-onb-of X*) = *X*›
⟨*proof*⟩

lemma *ccsubspace-as-whole-type*:
fixes *X* :: ‹′*a*::*chilbert-space ccsubspace*›
assumes ‹*X* ≠ *0*›
shows ‹*let* ′*b*::*type* = *some-onb-of X in*
∃ *U*::′*b ell2* $\Rightarrow_{CL}$ ′*a. isometry U* ∧ *U* $*_S$ ⊤ = *X*›
⟨*proof*⟩

lemma *some-onb-of-0*[*simp*]: ‹*some-onb-of* (*0* :: ′*a*::*chilbert-space ccsubspace*) = {}›
⟨*proof*⟩

lemma *some-onb-of-finite-dim*:
fixes *S* :: ‹′*a*::*chilbert-space ccsubspace*›
assumes ‹*finite-dim-ccsubspace S*›
shows ‹*finite* (*some-onb-of S*)›
⟨*proof*⟩

lemma *some-onb-of-in-space*[*iff*]:
fixes *S* :: ‹′*a*::*chilbert-space ccsubspace*›
shows ‹*some-onb-of S* ⊆ *space-as-set S*›
⟨*proof*⟩

lemma *sum-some-onb-of-butterfly*:
fixes *S* :: ‹′*a*::*chilbert-space ccsubspace*›
assumes ‹*finite-dim-ccsubspace S*›
shows ‹(∑ *x*∈*some-onb-of S. butterfly x x*) = *Proj S*›
⟨*proof*⟩

lemma *cdim-infinite-0*:
  assumes ‹¬ *cfinite-dim S*›
  shows ‹*cdim S* = *0*›
⟨*proof*⟩

lemma *some-onb-of-card*:
  fixes *S* :: ‹′*a*::*chilbert-space ccsubspace*›
  shows ‹*card* (*some-onb-of S*) = *cdim* (*space-as-set S*)›
⟨*proof*⟩

unbundle *no lattice-syntax* **and** *no cblinfun-syntax*

end

8 *Eigenvalues* – Material related to eigenvalues and eigenspaces

theory *Eigenvalues*
  imports
    Weak-Operator-Topology
    Misc-Tensor-Product-TTS
begin

unbundle *cblinfun-syntax*

definition *normal-op* :: ‹(′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*a*) ⇒ *bool*› **where**
  ‹*normal-op A* ⟷ *A* o_{CL} *A*∗ = *A*∗ o_{CL} *A*›

definition *eigenvalues* :: ‹(′*a*::*complex-normed-vector* $\Rightarrow_{CL}$ ′*a*) ⇒ *complex set*› **where**
  ‹*eigenvalues a* = {*x*. *eigenspace x a* ≠ *0*}›

definition *invariant-subspace* :: ‹′*a*::*complex-inner ccsubspace* ⇒ (′*a* $\Rightarrow_{CL}$ ′*a*) ⇒ *bool*› **where**
  ‹*invariant-subspace S A* ⟷ *A* $*_S$ *S* ≤ *S*›

lemma *invariant-subspaceI*: ‹*A* $*_S$ *S* ≤ *S* ⟹ *invariant-subspace S A*›
  ⟨*proof*⟩

definition *reducing-subspace* :: ‹′*a*::*complex-inner ccsubspace* ⇒ (′*a* $\Rightarrow_{CL}$ ′*a*) ⇒ *bool*› **where**
  ‹*reducing-subspace S A* ⟷ *invariant-subspace S A* ∧ *invariant-subspace* (−*S*) *A*›

lemma *reducing-subspaceI*: ‹*A* $*_S$ *S* ≤ *S* ⟹ *A* $*_S$ (−*S*) ≤ −*S* ⟹ *reducing-subspace S A*›
  ⟨*proof*⟩

lemma *reducing-subspace-ortho*[*simp*]: ‹*reducing-subspace* (−*S*) *A* ⟷ *reducing-subspace S A*›
  for *S* :: ‹′*a*::*chilbert-space ccsubspace*›
  ⟨*proof*⟩

lemma *invariant-subspace-bot*[*simp*]: ‹*invariant-subspace* ⊥ *A*›

⟨*proof*⟩

lemma *invariant-subspace-top*[*simp*]: ‹*invariant-subspace* ⊤ *A*›
⟨*proof*⟩

lemma *reducing-subspace-bot*[*simp*]: ‹*reducing-subspace* ⊥ *A*›
⟨*proof*⟩

lemma *reducing-subspace-top*[*simp*]: ‹*reducing-subspace* ⊤ *A*›
⟨*proof*⟩

lemma *kernel-uminus*[*simp*]: *kernel* (−*A*) = *kernel A*
for *a* :: *complex* **and** *A* :: (-,-) *cblinfun*
⟨*proof*⟩

lemma *kernel-scaleC'*: *kernel* ($a *_C A$) = (*if a* = *0 then* ⊤ *else kernel A*)
for *a* :: *complex* **and** *A* :: (-,-) *cblinfun*
⟨*proof*⟩

lemma *eigenvalues-0*[*simp*]: ‹*eigenvalues* (*0* :: *'a*::{*not-singleton*,*complex-normed-vector*} $\Rightarrow_{CL}$ *'a*) = {*0*}›
⟨*proof*⟩

lemma *nonzero-ccsubspace-contains-unit-vector*:
assumes ‹*S* ≠ *0*›
shows ‹∃ ψ. ψ ∈ *space-as-set S* ∧ *norm* ψ = *1*›
⟨*proof*⟩

lemma *unit-eigenvector-ex*:
assumes ‹*x* ∈ *eigenvalues a*›
shows ‹∃ *h*. *norm h* = *1* ∧ *a h* = $x *_C h$›
⟨*proof*⟩

lemma *eigenvalue-norm-bound*:
assumes ‹*e* ∈ *eigenvalues a*›
shows ‹*norm e* ≤ *norm a*›
⟨*proof*⟩

lemma *eigenvalue-selfadj-real*:
assumes ‹*e* ∈ *eigenvalues a*›
assumes ‹*selfadjoint a*›
shows ‹*e* ∈ ℝ›
⟨*proof*⟩

lemma *is-Sup-imp-ex-tendsto*:
fixes *X* :: ‹*'a*::{*linorder-topology*,*first-countable-topology*} *set*›
assumes *sup*: ‹*is-Sup X l*›
assumes ‹*X* ≠ {}›

shows ‹$\exists f.$ *range* $f \subseteq X \wedge f \longrightarrow l$›
⟨*proof*⟩

lemma *eigenvaluesI*:
assumes ‹$A *_V h = e *_C h$›
assumes ‹$h \neq 0$›
shows ‹$e \in$ *eigenvalues* A›
⟨*proof*⟩

lemma *tendsto-diff-const-left-rewrite*:
fixes $c\ d$ :: ‹$'a$::{*topological-group-add*, *ab-group-add*}›
assumes ‹$((\lambda x.\ f\ x) \longrightarrow c - d)\ F$›
shows ‹$((\lambda x.\ c - f\ x) \longrightarrow d)\ F$›
⟨*proof*⟩

lemma *not-not-singleton-no-eigenvalues*:
fixes a :: ‹$'a$::*complex-normed-vector* $\Rightarrow_{CL}$ $'a$›
assumes ‹$\neg$ *class.not-singleton* *TYPE*($'a$)›
shows ‹*eigenvalues* a = {}›
⟨*proof*⟩

lemma *cblinfun-cinner-eq0I*:
fixes a :: ‹$'a$::*chilbert-space* $\Rightarrow_{CL}$ $'a$›
assumes ‹$\bigwedge h.\ h \bullet_C a\ h = 0$›
shows ‹$a = 0$›
⟨*proof*⟩

lemma *normal-op-iff-adj-same-norms*:
— [2], Proposition II.2.16
fixes a :: ‹$'a$::*chilbert-space* $\Rightarrow_{CL}$ $'a$›
shows ‹*normal-op* $a \longleftrightarrow (\forall h.$ *norm* $(a\ h) =$ *norm* $((a*)\ h))$›
⟨*proof*⟩

lemma *normal-op-same-eigenspace-as-adj*:
— Shown inside the proof of [2, Proposition II.5.6]
assumes ‹*normal-op* a›
shows ‹*eigenspace* $l\ a$ = *eigenspace* (*cnj* l) $(a*\ )$›
⟨*proof*⟩

lemma *normal-op-adj-eigenvalues*:
assumes ‹*normal-op* a›
shows ‹*eigenvalues* $(a*)$ = *cnj* ‘ *eigenvalues* a›
⟨*proof*⟩

lemma *invariant-subspace-iff-PAP*:
— [2], Proposition II.3.7 (b)
‹*invariant-subspace* $S\ A \longleftrightarrow$ *Proj* $S\ o_{CL}\ A\ o_{CL}$ *Proj* $S = A\ o_{CL}$ *Proj* S›
⟨*proof*⟩

lemma *reducing-iff-PA*:
— [2], Proposition II.3.7 (e)
‹*reducing-subspace S A* ⟷ *Proj S* o_{CL} *A* = *A* o_{CL} *Proj S*›
⟨*proof*⟩

lemma *reducing-iff-also-adj-invariant*:
— [2], Proposition II.3.7 (g)
shows ‹*reducing-subspace S A* ⟷ *invariant-subspace S A* ∧ *invariant-subspace S* (*A*∗)›
⟨*proof*⟩

lemma *eigenspace-is-reducing*:
— [2], Proposition II.5.6
assumes ‹*normal-op a*›
shows ‹*reducing-subspace* (*eigenspace l a*) *a*›
⟨*proof*⟩

lemma *invariant-subspace-Inf*:
assumes ‹⋀*S*. *S* ∈ *M* ⟹ *invariant-subspace S a*›
shows ‹*invariant-subspace* (⨅*M*) *a*›
⟨*proof*⟩

lemma *invariant-subspace-INF*:
assumes ‹⋀*x*. *x* ∈ *X* ⟹ *invariant-subspace* (*S x*) *a*›
shows ‹*invariant-subspace* (⨅*x*∈*X*. *S x*) *a*›
⟨*proof*⟩

lemma *invariant-subspace-Sup*:
assumes ‹⋀*S*. *S* ∈ *M* ⟹ *invariant-subspace S a*›
shows ‹*invariant-subspace* (⨆*M*) *a*›
⟨*proof*⟩

lemma *invariant-subspace-SUP*:
assumes ‹⋀*x*. *x* ∈ *X* ⟹ *invariant-subspace* (*S x*) *a*›
shows ‹*invariant-subspace* (⨆*x*∈*X*. *S x*) *a*›
⟨*proof*⟩

lemma *reducing-subspace-Inf*:
fixes *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'a*›
assumes ‹⋀*S*. *S* ∈ *M* ⟹ *reducing-subspace S a*›
shows ‹*reducing-subspace* (⨅*M*) *a*›
⟨*proof*⟩

lemma *reducing-subspace-INF*:
fixes *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'a*›
assumes ‹⋀*x*. *x* ∈ *X* ⟹ *reducing-subspace* (*S x*) *a*›
shows ‹*reducing-subspace* (⨅*x*∈*X*. *S x*) *a*›
⟨*proof*⟩

lemma *reducing-subspace-Sup*:
  fixes *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'a*›
  assumes ‹$\bigwedge$*S*. *S* ∈ *M* ⟹ *reducing-subspace S a*›
  shows ‹*reducing-subspace* ($\bigsqcup$ *M*) *a*›
  ⟨*proof*⟩

lemma *reducing-subspace-SUP*:
  fixes *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'a*›
  assumes ‹$\bigwedge$*x*. *x* ∈ *X* ⟹ *reducing-subspace* (*S x*) *a*›
  shows ‹*reducing-subspace* ($\bigsqcup$*x*∈*X*. *S x*) *a*›
  ⟨*proof*⟩

lemma *selfadjoint-imp-normal*: ‹*normal-op a*› **if** ‹*selfadjoint a*›
  ⟨*proof*⟩

lemma *eigenspaces-orthogonal*:
  — [2], Proposition II.5.7
  assumes ‹*e* ≠ *f*›
  assumes ‹*normal-op a*›
  shows ‹*orthogonal-spaces* (*eigenspace e a*) (*eigenspace f a*)›
⟨*proof*⟩

definition *largest-eigenvalue* :: ‹(*'a::complex-normed-vector* $\Rightarrow_{CL}$ *'a*) ⇒ *complex*› **where**
  ‹*largest-eigenvalue a* =
    (*if* ∃ *x*. *x* ∈ *eigenvalues a* ∧ (∀ *y* ∈ *eigenvalues a*. *cmod x* ≥ *cmod y*) *then*
    SOME x. *x* ∈ *eigenvalues a* ∧ (∀ *y* ∈ *eigenvalues a*. *cmod x* ≥ *cmod y*) *else 0*)›

lemma *largest-eigenvalue-0-aux*:
  ‹*largest-eigenvalue* (*0* :: *'a::{not-singleton,complex-normed-vector}* $\Rightarrow_{CL}$ *'a*) = *0*›
⟨*proof*⟩

lemma *largest-eigenvalue-0*[*simp*]:
  ‹*largest-eigenvalue* (*0* :: *'a::complex-normed-vector* $\Rightarrow_{CL}$ *'a*) = *0*›
⟨*proof*⟩

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

lemma *eigenvalues-nonneg*:
  assumes ‹*a* ≥ *0*› **and** ‹*v* ∈ *eigenvalues a*›
  shows ‹*v* ≥ *0*›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

9 *Compact-Operators* – Finite rank and compact operators

theory *Compact-Operators*
  imports
    Sqrt-Babylonian.Sqrt-Babylonian-Auxiliary
    Wlog.Wlog
    HOL−Analysis.Abstract-Metric-Spaces

    HS2Ell2
    Strong-Operator-Topology
    Misc-Tensor-Product-TTS
    Eigenvalues
begin

unbundle *cblinfun-syntax*

9.1 Finite rank operators

definition *finite-rank* **where** ‹*finite-rank A* ⟷ *A* ∈ *cspan* (*Collect rank1*)›

lemma *finite-rank-0*[*simp*]: ‹*finite-rank 0*›
  ⟨*proof*⟩

lemma *finite-rank-scaleC*[*simp*]: ‹*finite-rank* ($c *_C a$)› **if** ‹*finite-rank a*›
  ⟨*proof*⟩

lemma *finite-rank-scaleR*[*simp*]: ‹*finite-rank* ($c *_R a$)› **if** ‹*finite-rank a*›
  ⟨*proof*⟩

lemma *finite-rank-uminus*[*simp*]: ‹*finite-rank* (−*a*) = *finite-rank a*›
  ⟨*proof*⟩

lemma *finite-rank-plus*[*simp*]: ‹*finite-rank* (*a* + *b*)› **if** ‹*finite-rank a*› **and** ‹*finite-rank b*›
  ⟨*proof*⟩

lemma *finite-rank-minus*[*simp*]: ‹*finite-rank* (*a* − *b*)› **if** ‹*finite-rank a*› **and** ‹*finite-rank b*›
  ⟨*proof*⟩

lemma *finite-rank-butterfly*[*simp*]: ‹*finite-rank* (*butterfly x y*)›
  ⟨*proof*⟩

lemma *finite-rank-sum-butterfly*:
  fixes *a* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*::*chilbert-space*›
  assumes ‹*finite-rank a*›
  shows ‹∃ *x y* (*n*::*nat*). *a* = (∑ *i*<*n*. *butterfly* (*x i*) (*y i*))›
⟨*proof*⟩

lemma *finite-rank-sum*: ‹*finite-rank* (∑ *x*∈*F*. *f x*)› **if** ‹⋀*x*. *x*∈*F* ⟹ *finite-rank* (*f x*)›

⟨*proof*⟩

lemma *rank1-finite-rank*: ‹*finite-rank a*› **if** ‹*rank1 a*›
⟨*proof*⟩

lemma *finite-rank-compose-left*:
assumes ‹*finite-rank B*›
shows ‹*finite-rank* $(A\ o_{CL}\ B)$›
⟨*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 ccsubspace*›
assumes ‹*finite-dim-ccsubspace 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* ⟷ *a* ∈ *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*⟩

9.2 Compact operators

definition *compact-map* **where** ‹*compact-map f* ⟷ *clinear f* ∧ *compact* (*closure* (*f* ‘ *cball 0 1*))›

lemma ‹*bounded-clinear f*› **if** ‹*compact-map f*›
— [2], Proposition II.4.2 (a)
thm *bounded-clinear-def*
⟨*proof*⟩

lift-definition *compact-op* :: ‹(′*a*::*complex-normed-vector* $\Rightarrow_{CL}$ ′*b*::*complex-normed-vector*) ⇒ *bool*› **is** *compact-map*⟨*proof*⟩

lemma *compact-op-def2*: ‹*compact-op a* ⟷ *compact* (*closure* (*a* ‘ *cball 0 1*))›
⟨*proof*⟩

lemma *compact-op-0*[*simp*]: ‹*compact-op 0*›
⟨*proof*⟩

lemma *compact-op-scaleC*[*simp*]: ‹*compact-op* (*c* $*_C$ *a*)› **if** ‹*compact-op a*›
⟨*proof*⟩

lemma *compact-op-scaleR*[*simp*]: ‹*compact-op* (*c* $*_R$ *a*)› **if** ‹*compact-op a*›
⟨*proof*⟩

lemma *compact-op-uminus*[*simp*]: ‹*compact-op* (−*a*) = *compact-op a*›
⟨*proof*⟩

lemma *compact-op-plus*[*simp*]: ‹*compact-op* (*a* + *b*)› **if** ‹*compact-op a*› **and** ‹*compact-op b*›
⟨*proof*⟩

lemma *csubspace-compact-op*: ‹*csubspace* (*Collect compact-op*)›
— [2], Proposition II.4.2 (b)
⟨*proof*⟩

lemma *compact-op-minus*[*simp*]: ‹*compact-op* (*a* − *b*)› **if** ‹*compact-op a*› **and** ‹*compact-op b*›
⟨*proof*⟩

lemma *compact-op-sgn*[*simp*]: ‹*compact-op* (*sgn a*) = *compact-op a*›
⟨*proof*⟩

lemma *closed-compact-op*:
shows ‹*closed* (*Collect* (*compact-op* :: (′*a*::*complex-normed-vector* $\Rightarrow_{CL}$ ′*b*::*chilbert-space*) ⇒ *bool*))›

— [2], Proposition II.4.2 (b)
⟨*proof*⟩

lemma *rank1-compact-op*: ‹*compact-op a*› **if** ‹*rank1 a*›
⟨*proof*⟩

lemma *finite-rank-compact-op*: ‹*compact-op a*› **if** ‹*finite-rank a*›
⟨*proof*⟩

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

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

typedef (**overloaded**) ($'a$::*chilbert-space*,$'b$::*complex-normed-vector*) *compact-op* =
‹*Collect compact-op* :: ($'a \Rightarrow_{CL}$ $'b$) *set*›
morphisms *from-compact-op Abs-compact-op*
⟨*proof*⟩
setup-lifting *type-definition-compact-op*

instantiation *compact-op* :: (*chilbert-space*, *complex-normed-vector*) *complex-normed-vector* **begin**
lift-definition *scaleC-compact-op* :: ‹*complex* ⇒ ($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op*› **is** *scaleC* ⟨*proof*⟩
lift-definition *uminus-compact-op* :: ‹($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op*› **is** *uminus* ⟨*proof*⟩
lift-definition *zero-compact-op* :: ‹($'a$, $'b$) *compact-op*› **is** *0* ⟨*proof*⟩
lift-definition *minus-compact-op* :: ‹($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op*› **is** *minus* ⟨*proof*⟩
lift-definition *plus-compact-op* :: ‹($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op*› **is** *plus* ⟨*proof*⟩
lift-definition *sgn-compact-op* :: ‹($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op*› **is** *sgn* ⟨*proof*⟩
lift-definition *norm-compact-op* :: ‹($'a$, $'b$) *compact-op* ⇒ *real*› **is** *norm* ⟨*proof*⟩
lift-definition *scaleR-compact-op* :: ‹*real* ⇒ ($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op*› **is** *scaleR* ⟨*proof*⟩
lift-definition *dist-compact-op* :: ‹($'a$, $'b$) *compact-op* ⇒ ($'a$, $'b$) *compact-op* ⇒ *real*› **is** *dist* ⟨*proof*⟩
definition [*code del*]:

‹(*uniformity* :: ((′*a*, ′*b*) *compact-op* × (′*a*, ′*b*) *compact-op*) *filter*) = (*INF e*∈{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})›
definition *open-compact-op* :: (′*a*, ′*b*) *compact-op set* ⇒ *bool*
where [*code del*]: *open-compact-op S* = (∀ *x*∈*S*. ∀ $_F$ (*x*′, *y*) *in uniformity*. *x*′ = *x* ⟶ *y* ∈ *S*)
instance
⟨*proof*⟩
end

lemma *from-compact-op-plus*: ‹*from-compact-op* (*a* + *b*) = *from-compact-op a* + *from-compact-op b*›
⟨*proof*⟩

lemma *from-compact-op-scaleC*: ‹*from-compact-op* (*c* $*_C$ *a*) = *c* $*_C$ *from-compact-op a*›
⟨*proof*⟩

lemma *from-compact-op-norm*[*simp*]: ‹*norm* (*from-compact-op a*) = *norm a*›
⟨*proof*⟩

lemma *compact-op-butterfly*[*simp*]: ‹*compact-op* (*butterfly x y*)›
⟨*proof*⟩

lift-definition *butterfly-co* :: ‹′*a*::*complex-normed-vector* ⇒ ′*b*::*chilbert-space* ⇒ (′*b*,′*a*) *compact-op*› **is** *butterfly*
⟨*proof*⟩

lemma *butterfly-co-add-left*: ‹*butterfly-co* (*a* + *a*′) *b* = *butterfly-co a b* + *butterfly-co a*′ *b*›
⟨*proof*⟩

lemma *butterfly-co-add-right*: ‹*butterfly-co a* (*b* + *b*′) = *butterfly-co a b* + *butterfly-co a b*′›
⟨*proof*⟩

lemma *butterfly-co-scaleR-left*[*simp*]: *butterfly-co* (*r* $*_R$ *ψ*) *φ* = *r* $*_C$ *butterfly-co ψ φ*
⟨*proof*⟩

lemma *butterfly-co-scaleR-right*[*simp*]: *butterfly-co ψ* (*r* $*_R$ *φ*) = *r* $*_C$ *butterfly-co ψ φ*
⟨*proof*⟩

lemma *butterfly-co-scaleC-left*[*simp*]: *butterfly-co* (*r* $*_C$ *ψ*) *φ* = *r* $*_C$ *butterfly-co ψ φ*
⟨*proof*⟩

lemma *butterfly-co-scaleC-right*[*simp*]: *butterfly-co ψ* (*r* $*_C$ *φ*) = *cnj r* $*_C$ *butterfly-co ψ φ*
⟨*proof*⟩

lemma *finite-rank-separating-on-compact-op*:
fixes *F G* :: ‹(′*a*::*chilbert-space*,′*b*::*chilbert-space*) *compact-op* ⇒ ′*c*::*complex-normed-vector*›
assumes ‹⋀*x*. *finite-rank* (*from-compact-op x*) ⟹ *F x* = *G x*›
assumes ‹*bounded-clinear F*›
assumes ‹*bounded-clinear G*›

shows ‹$F = G$›
⟨*proof*⟩

lemma *trunc-ell2-as-Proj*: ‹*trunc-ell2* S ψ = *Proj* (*ccspan* (*ket* ‘ S)) ψ›
⟨*proof*⟩

lemma *unitary-between-bij-betw*:
assumes ‹*is-onb* A› ‹*is-onb* B›
shows ‹*bij-betw* (($*_V$) (*unitary-between* A B)) A B›
⟨*proof*⟩

lemma *tendsto-finite-subsets-at-top-image*:
assumes ‹*inj-on* g X›
shows ‹($f \longrightarrow x$) (*finite-subsets-at-top* (g ‘ X)) $\longleftrightarrow$ ((λS. f (g ‘ S)) $\longrightarrow x$) (*finite-subsets-at-top* X)›
⟨*proof*⟩

lemma *Proj-onb-limit*:
shows ‹*is-onb* $A \Longrightarrow$ ((λS. *Proj* (*ccspan* S) ψ) $\longrightarrow \psi$) (*finite-subsets-at-top* A)›
⟨*proof*⟩

lemma *is-ortho-setD*:
assumes *is-ortho-set* S $x \in S$ $y \in S$ $x \neq y$
shows $x \cdot_C y = 0$
⟨*proof*⟩

lemma *finite-rank-dense-compact*:
fixes A :: ‹'*a*::*chilbert-space set*› **and** B :: ‹'*b*::*chilbert-space set*›
assumes ‹*is-onb* A› **and** ‹*is-onb* B›
shows ‹*closure* (*cspan* (($\lambda(\xi,\eta)$. *butterfly* ξ η) ‘ ($A \times B$))) = *Collect compact-op*›
⟨*proof*⟩

lemma *compact-op-comp-left*: ‹*compact-op* (a o_{CL} b)› **if** ‹*compact-op* a›
for a b :: ‹-::*chilbert-space* $\Rightarrow_{CL}$ -::*chilbert-space*›
⟨*proof*⟩

lemma *compact-op-eigenspace-finite-dim*:
fixes a :: ‹'*a* $\Rightarrow_{CL}$ '*a*::*chilbert-space*›
assumes ‹*compact-op* a›
assumes ‹$e \neq 0$›
shows ‹*finite-dim-ccsubspace* (*eigenspace* e a)›
⟨*proof*⟩

lemma *eigenvalue-in-the-limit-compact-op*:
— [2], Proposition II.4.14
assumes ‹*compact-op* T›

assumes ‹$l \neq 0$›
assumes *normh*: ‹$\bigwedge n.$ *norm* $(h\ n) = 1$›
assumes *Tl-lim*: ‹$(\lambda n.\ (T - l *_C$ *id-cblinfun*$)\ (h\ n)) \longrightarrow 0$›
shows ‹$l \in$ *eigenvalues* T›
⟨*proof*⟩

lemma *norm-is-eigenvalue*:
— [2], Proposition II.5.9
fixes a :: ‹$'a \Rightarrow_{CL} 'a$::{*not-singleton*, *chilbert-space*}›
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹*norm a* ∈ *eigenvalues a* ∨ − *norm a* ∈ *eigenvalues a*›
⟨*proof*⟩

lemma
fixes a :: ‹$'a \Rightarrow_{CL} 'a$::{*not-singleton*, *chilbert-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 :: ‹$'a \Rightarrow_{CL} 'a$::*chilbert-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 :: ‹$'a \Rightarrow_{CL} 'a$::*chilbert-space*›
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹*cmod* (*largest-eigenvalue a*) = *norm a*›
⟨*proof*⟩

lemma *compact-op-comp-right*: ‹*compact-op* $(a\ o_{CL}\ b)$› **if** ‹*compact-op b*›
for *a b* :: ‹-::*chilbert-space* $\Rightarrow_{CL}$ -::*chilbert-space*›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

10 *Spectral-Theorem* – The spectral theorem for compact operators

theory *Spectral-Theorem*
  imports *Compact-Operators Positive-Operators Eigenvalues*
begin

unbundle *cblinfun-syntax*

10.1 Spectral decomp, compact op

fun *spectral-dec-val* :: ‹('a::chilbert-space $\Rightarrow_{CL}$ 'a) ⇒ nat ⇒ complex›
  — The eigenvalues in the spectral decomposition
  and *spectral-dec-space* :: ‹('a $\Rightarrow_{CL}$ 'a) ⇒ nat ⇒ 'a ccsubspace›
  — The eigenspaces in the spectral decomposition
  and *spectral-dec-op* :: ‹('a $\Rightarrow_{CL}$ 'a) ⇒ nat ⇒ ('a $\Rightarrow_{CL}$ 'a)›
  — A sequence of operators mostly for the proof of spectral composition. But see also *spectral-dec-op-spectral-dec-proj* below.
  where ‹*spectral-dec-val a n = largest-eigenvalue (spectral-dec-op a n)*›
  | ‹*spectral-dec-space a n = (if spectral-dec-val a n = 0 then 0 else eigenspace (spectral-dec-val a n) (spectral-dec-op a n))*›
  | ‹*spectral-dec-op a (Suc n) = spectral-dec-op a n* o_{CL} *Proj (− spectral-dec-space a n)*›
  | ‹*spectral-dec-op a 0 = a*›

definition *spectral-dec-proj* :: ‹('a::chilbert-space $\Rightarrow_{CL}$ 'a) ⇒ nat ⇒ ('a $\Rightarrow_{CL}$ 'a)› **where**
  — Projectors in the spectral decomposition
  ‹*spectral-dec-proj a n = Proj (spectral-dec-space a n)*›

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 ‹*selfadjoint a*›
  shows ‹*selfadjoint (spectral-dec-op a n)*›
⟨*proof*⟩

lemma *spectral-dec-op-compact*:
  assumes ‹*compact-op a*›
  shows ‹*compact-op (spectral-dec-op a n)*›
  ⟨*proof*⟩

lemma *spectral-dec-val-eigenvalue-of-spectral-dec-op*:
  fixes *a* :: ‹'a::{chilbert-space, not-singleton} $\Rightarrow_{CL}$ 'a›
  assumes ‹*compact-op a*›
  assumes ‹*selfadjoint a*›

shows ‹*spectral-dec-val a n* ∈ *eigenvalues* (*spectral-dec-op a n*)›
⟨*proof*⟩

lemma *spectral-dec-proj-finite-rank*:
assumes ‹*compact-op a*›
shows ‹*finite-rank* (*spectral-dec-proj a n*)›
⟨*proof*⟩

lemma *norm-spectral-dec-op*:
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹*norm* (*spectral-dec-op a n*) = *cmod* (*spectral-dec-val a n*)›
⟨*proof*⟩

lemma *spectral-dec-op-decreasing-eigenspaces*:
assumes ‹$n \geq m$› **and** ‹$e \neq 0$›
assumes ‹*selfadjoint a*›
shows ‹*eigenspace e* (*spectral-dec-op a n*) ≤ *eigenspace e* (*spectral-dec-op a m*)›
⟨*proof*⟩

lemma *spectral-dec-val-not-not-singleton*:
fixes *a* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*a*›
assumes ‹¬ *class.not-singleton TYPE*('*a*)›
shows ‹*spectral-dec-val a n* = *0*›
⟨*proof*⟩

lemma *spectral-dec-val-eigenvalue-aux*:
— [2], Theorem II.5.1
fixes *a* :: ‹'*a*::{*chilbert-space*, *not-singleton*} $\Rightarrow_{CL}$ '*a*›
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
assumes *eigen-neq0*: ‹*spectral-dec-val a n* ≠ *0*›
shows ‹*spectral-dec-val a n* ∈ *eigenvalues a*›
⟨*proof*⟩

lemma *spectral-dec-val-eigenvalue*:
— [2], Theorem II.5.1
fixes *a* :: ‹('*a*::*chilbert-space* $\Rightarrow_{CL}$ '*a*)›
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
assumes *eigen-neq0*: ‹*spectral-dec-val a n* ≠ *0*›
shows ‹*spectral-dec-val a n* ∈ *eigenvalues a*›
⟨*proof*⟩

hide-fact *spectral-dec-val-eigenvalue-aux*

lemma *spectral-dec-val-decreasing*:
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›

assumes ‹$n \geq m$›
shows ‹*cmod* (*spectral-dec-val a n*) $\leq$ *cmod* (*spectral-dec-val a m*)›
⟨*proof*⟩

lemma *spectral-dec-val-distinct-aux*:
fixes *a* :: ‹(*'a*::{*chilbert-space*, *not-singleton*} $\Rightarrow_{CL}$ *'a*)›
assumes ‹$n \neq m$›
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
assumes *neq0*: ‹*spectral-dec-val a n* $\neq$ *0*›
shows ‹*spectral-dec-val a n* $\neq$ *spectral-dec-val a m*›
⟨*proof*⟩

lemma *spectral-dec-val-distinct*:
fixes *a* :: ‹*'a*::*chilbert-space* $\Rightarrow_{CL}$ *'a*›
assumes ‹$n \neq m$›
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
assumes *neq0*: ‹*spectral-dec-val a n* $\neq$ *0*›
shows ‹*spectral-dec-val a n* $\neq$ *spectral-dec-val a m*›
⟨*proof*⟩

hide-fact *spectral-dec-val-distinct-aux*

lemma *spectral-dec-val-tendsto-0*:

assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹*spectral-dec-val a* $\longrightarrow$ *0*›
⟨*proof*⟩

lemma *spectral-dec-op-tendsto*:
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹*spectral-dec-op a* $\longrightarrow$ *0*›
⟨*proof*⟩

lemma *spectral-dec-op-spectral-dec-proj*:
‹*spectral-dec-op a n* = *a* − ($\sum$ *i*<*n*. *spectral-dec-val a i* $*_C$ *spectral-dec-proj a i*)›
⟨*proof*⟩

lemma *sequential-tendsto-reorder*:
assumes ‹*inj g*›
assumes ‹*f* $\longrightarrow$ *l*›
shows ‹(*f o g*) $\longrightarrow$ *l*›
⟨*proof*⟩

lemma *spectral-dec-sums*:
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹(λn. *spectral-dec-val a n* $*_C$ *spectral-dec-proj a n*) *sums* *a*›
⟨*proof*⟩

lemma *spectral-dec-val-real*:
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹*spectral-dec-val a n* $\in \mathbb{R}$›
⟨*proof*⟩

lemma *spectral-dec-space-orthogonal*:
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
assumes ‹$n \neq m$›
shows ‹*orthogonal-spaces* (*spectral-dec-space a n*) (*spectral-dec-space a m*)›
⟨*proof*⟩

lemma *spectral-dec-proj-pos*: ‹*spectral-dec-proj a n* $\geq$ *0*›
⟨*proof*⟩

lemma
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows *spectral-dec-tendsto-pos-op*: ‹(λn. *max 0* (*spectral-dec-val a n*) $*_C$ *spectral-dec-proj a n*) *sums* *pos-op a*› (**is** *?thesis1*)
and *spectral-dec-tendsto-neg-op*: ‹(λn. − *min* (*spectral-dec-val a n*) *0* $*_C$ *spectral-dec-proj a n*) *sums* *neg-op a*› (**is** *?thesis2*)
⟨*proof*⟩

lemma *spectral-dec-tendsto-abs-op*:
assumes ‹*compact-op a*›
assumes ‹*selfadjoint a*›
shows ‹(λn. *cmod* (*spectral-dec-val a n*) $*_R$ *spectral-dec-proj a n*) *sums* *abs-op a*›
⟨*proof*⟩

definition *spectral-dec-vecs* :: ‹('*a* $\Rightarrow_{CL}$ '*a*) $\Rightarrow$ '*a*::*chilbert-space set*› **where**
‹*spectral-dec-vecs a* = ($\bigcup$ *n*. *scaleC* (*csqrt* (*spectral-dec-val a n*)) ‘ *some-onb-of* (*spectral-dec-space a n*))›

lemma *spectral-dec-vecs-ortho*:
assumes ‹*selfadjoint a*› **and** ‹*compact-op a*›
shows ‹*is-ortho-set* (*spectral-dec-vecs a*)›

⟨*proof*⟩

lemma *spectral-dec-val-nonneg*:
  assumes ‹$a \geq 0$›
  assumes ‹*compact-op a*›
  shows ‹*spectral-dec-val a n* ≥ 0›
⟨*proof*⟩

lemma *spectral-dec-space-finite-dim*[*intro*]:
  assumes ‹*compact-op a*›
  shows ‹*finite-dim-ccsubspace* (*spectral-dec-space a n*)›
  ⟨*proof*⟩

lemma *spectral-dec-space-0*:
  assumes ‹*spectral-dec-val a n* = *0*›
  shows ‹*spectral-dec-space a n* = *0*›
  ⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

11 *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*

11.1 Auxiliary lemmas

lemma
  fixes *h* :: ‹*'a*::{*chilbert-space*}›
  assumes ‹*is-onb E*›
  shows *parseval-abs-summable*: ‹($\lambda e.$ (*cmod* ($e \cdot_C h$))^2) *abs-summable-on E*›
⟨*proof*⟩

lemma *basis-image-square-has-sum1*:
  — Half of [1, Proposition 18.1], other half in *basis-image-square-has-sum1*.
  fixes *E* :: ‹*'a*::*complex-inner set*› **and** *F* :: ‹*'b*::*chilbert-space set*›

assumes ‹*is-onb E*› **and** ‹*is-onb F*›
shows ‹$((\lambda e.\ (norm\ (A *_V\ e))^2)\ has\text{-}sum\ t)\ E \longleftrightarrow ((\lambda(e,f).\ (cmod\ (f \cdot_C\ (A *_V\ e)))^2)\ has\text{-}sum\ t)\ (E \times F)$›
⟨*proof*⟩

lemma *basis-image-square-has-sum2*:
— Half of [1, Proposition 18.1], other half in *basis-image-square-has-sum1*.
fixes *E* :: ‹*'a::chilbert-space set*› **and** *F* :: ‹*'b::chilbert-space set*›
assumes ‹*is-onb E*› **and** ‹*is-onb F*›
shows ‹$((\lambda e.\ (norm\ (A *_V\ e))^2)\ has\text{-}sum\ t)\ E \longleftrightarrow ((\lambda f.\ (norm\ (A* *_V\ f))^2)\ has\text{-}sum\ t)\ F$›
⟨*proof*⟩

11.2 Trace-norm and trace-class

lemma *trace-norm-basis-invariance*:
assumes ‹*is-onb E*› **and** ‹*is-onb F*›
shows ‹$((\lambda e.\ cmod\ (e \cdot_C\ (abs\text{-}op\ A *_V\ e)))\ has\text{-}sum\ t)\ E \longleftrightarrow ((\lambda f.\ cmod\ (f \cdot_C\ (abs\text{-}op\ A *_V\ f)))\ has\text{-}sum\ t)\ F$›
— [1], Corollary 18.2
⟨*proof*⟩

definition *trace-class* :: ‹$('a::chilbert\text{-}space \Rightarrow_{CL} 'b::complex\text{-}inner) \Rightarrow bool$›
where ‹$trace\text{-}class\ A \longleftrightarrow (\exists E.\ is\text{-}onb\ E \wedge (\lambda e.\ e \cdot_C\ (abs\text{-}op\ A *_V\ e))\ abs\text{-}summable\text{-}on\ E)$›

lemma *trace-classI*:
assumes ‹*is-onb E*› **and** ‹$(\lambda e.\ e \cdot_C\ (abs\text{-}op\ A *_V\ e))\ abs\text{-}summable\text{-}on\ E$›
shows ‹*trace-class A*›
⟨*proof*⟩

lemma *trace-class-iff-summable*:
assumes ‹*is-onb E*›
shows ‹$trace\text{-}class\ A \longleftrightarrow (\lambda e.\ e \cdot_C\ (abs\text{-}op\ A *_V\ e))\ abs\text{-}summable\text{-}on\ E$›
⟨*proof*⟩

lemma *trace-class-0*[*simp*]: ‹*trace-class 0*›
⟨*proof*⟩

lemma *trace-class-uminus*: ‹$trace\text{-}class\ t \Longrightarrow trace\text{-}class\ (-t)$›
⟨*proof*⟩

lemma *trace-class-uminus-iff*[*simp*]: ‹$trace\text{-}class\ (-a) = trace\text{-}class\ a$›
⟨*proof*⟩

definition *trace-norm* **where** ‹$trace\text{-}norm\ A = (if\ trace\text{-}class\ A\ then\ (\sum_\infty e \in some\text{-}chilbert\text{-}basis.\ cmod\ (e \cdot_C\ (abs\text{-}op\ A *_V\ e)))\ else\ 0)$›

definition *trace* **where** ‹$trace\ A = (if\ trace\text{-}class\ A\ then\ (\sum_\infty e \in some\text{-}chilbert\text{-}basis.\ e \cdot_C\ (A$

$*_V$ *e*)) *else 0*)›

lemma *trace-0*[*simp*]: ‹*trace 0* = *0*›
⟨*proof*⟩

lemma *trace-class-abs-op*[*simp*]: ‹*trace-class* (*abs-op A*) = *trace-class A*›
⟨*proof*⟩

lemma *trace-abs-op*[*simp*]: ‹*trace* (*abs-op A*) = *trace-norm A*›
⟨*proof*⟩

lemma *trace-norm-pos*: ‹*trace-norm A* = *trace A*› **if** ‹*A* ≥ *0*›
⟨*proof*⟩

lemma *trace-norm-alt-def*:
assumes ‹*is-onb B*›
shows ‹*trace-norm A* = (*if trace-class A then* ($\sum_\infty$*e*∈*B*. *cmod* (*e* $\cdot_C$ (*abs-op A* $*_V$ *e*))) *else 0*)›
⟨*proof*⟩

lemma *trace-class-finite-dim*[*simp*]: ‹*trace-class A*› **for** *A* :: ‹*'a*::{*cfinite-dim*,*chilbert-space*} $\Rightarrow_{CL}$ *'b*::*complex-inner*›
⟨*proof*⟩

lemma *trace-class-scaleC*: ‹*trace-class* (*c* $*_C$ *a*)› **if** ‹*trace-class a*›
⟨*proof*⟩

lemma *trace-scaleC*: ‹*trace* (*c* $*_C$ *a*) = *c* ∗ *trace a*›
⟨*proof*⟩

lemma *trace-uminus*: ‹*trace* (− *a*) = − *trace a*›
⟨*proof*⟩

lemma *trace-norm-0*[*simp*]: ‹*trace-norm 0* = *0*›
⟨*proof*⟩

lemma *trace-norm-nneg*[*simp*]: ‹*trace-norm a* ≥ *0*›
⟨*proof*⟩

lemma *trace-norm-scaleC*: ‹*trace-norm* (*c* $*_C$ *a*) = *norm c* ∗ *trace-norm a*›
⟨*proof*⟩

lemma *trace-norm-nondegenerate*: ‹*a* = *0*› **if** ‹*trace-class a*› **and** ‹*trace-norm a* = *0*›
⟨*proof*⟩

typedef (**overloaded**) (*'a*::*chilbert-space*, *'b*::*chilbert-space*) *trace-class* = ‹*Collect trace-class* :: (*'a* $\Rightarrow_{CL}$ *'b*) *set*›

morphisms *from-trace-class Abs-trace-class*
⟨*proof*⟩
setup-lifting *type-definition-trace-class*

lemma *trace-class-from-trace-class*[*simp*]: ‹*trace-class* (*from-trace-class t*)›
⟨*proof*⟩

lemma *trace-pos*: ‹*trace* $a \geq 0$› **if** ‹$a \geq 0$›
⟨*proof*⟩

lemma *trace-adj-prelim*: ‹*trace* ($a*$) = *cnj* (*trace a*)› **if** ‹*trace-class a*› **and** ‹*trace-class* ($a*$)›
— We will later strengthen this as *trace-adj* and then hide this fact.
⟨*proof*⟩

11.3 Hilbert-Schmidt operators

definition *hilbert-schmidt* **where** ‹*hilbert-schmidt a* ⟷ *trace-class* ($a*$ o_{CL} a)›

definition *hilbert-schmidt-norm* **where** ‹*hilbert-schmidt-norm a* = *sqrt* (*trace-norm* ($a*$ o_{CL} a))›

lemma *hilbert-schmidtI*: ‹*hilbert-schmidt a*› **if** ‹*trace-class* ($a*$ o_{CL} a)›
⟨*proof*⟩

lemma *hilbert-schmidt-0*[*simp*]: ‹*hilbert-schmidt 0*›
⟨*proof*⟩

lemma *hilbert-schmidt-norm-pos*[*simp*]: ‹*hilbert-schmidt-norm* $a \geq 0$›
⟨*proof*⟩

lemma *has-sum-hilbert-schmidt-norm-square*:
— [1], Proposition 18.6 (a)
assumes ‹*is-onb B*› **and** ‹*hilbert-schmidt a*›
shows ‹((λx. (*norm* ($a *_V x$))^2) *has-sum* (*hilbert-schmidt-norm a*)^2) *B*›
⟨*proof*⟩

lemma *summable-hilbert-schmidt-norm-square*:
— [1], Proposition 18.6 (a)
assumes ‹*is-onb B*› **and** ‹*hilbert-schmidt a*›
shows ‹(λx. (*norm* ($a *_V x$))^2) *summable-on B*›
⟨*proof*⟩

lemma *summable-hilbert-schmidt-norm-square-converse*:
assumes ‹*is-onb B*›
assumes ‹(λx. (*norm* ($a *_V x$))^2) *summable-on B*›
shows ‹*hilbert-schmidt a*›
⟨*proof*⟩

lemma *infsum-hilbert-schmidt-norm-square*:

— [1], Proposition 18.6 (a)
assumes ‹*is-onb B*› **and** ‹*hilbert-schmidt a*›
shows ‹$(\sum_\infty x \in B.\ (norm\ (a *_V\ x))^2) = ((hilbert\text{-}schmidt\text{-}norm\ a)^2)$›
⟨*proof*⟩

lemma
— [1], Proposition 18.6 (d)
assumes ‹*hilbert-schmidt b*›
shows *hilbert-schmidt-comp-right*: ‹*hilbert-schmidt* $(a\ o_{CL}\ b)$›
and *hilbert-schmidt-norm-comp-right*: ‹*hilbert-schmidt-norm* $(a\ o_{CL}\ b) \leq$ *norm a* $*$ *hilbert-schmidt-norm b*›
⟨*proof*⟩

lemma *hilbert-schmidt-adj*[*simp*]:
— Implicit in [1], Proposition 18.6 (b)
assumes ‹*hilbert-schmidt a*›
shows ‹*hilbert-schmidt* $(a*)$›
⟨*proof*⟩

lemma *hilbert-schmidt-norm-adj*[*simp*]:
— [1], Proposition 18.6 (b)
shows ‹*hilbert-schmidt-norm* $(a*)$ = *hilbert-schmidt-norm a*›
⟨*proof*⟩

lemma
— [1], Proposition 18.6 (d)
fixes *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*› **and** *b*
assumes ‹*hilbert-schmidt a*›
shows *hilbert-schmidt-comp-left*: ‹*hilbert-schmidt* $(a\ o_{CL}\ b)$›
⟨*proof*⟩

lemma
— [1], Proposition 18.6 (d)
fixes *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*› **and** *b*
assumes ‹*hilbert-schmidt a*›
shows *hilbert-schmidt-norm-comp-left*: ‹*hilbert-schmidt-norm* $(a\ o_{CL}\ b) \leq$ *norm b* $*$ *hilbert-schmidt-norm a*›
⟨*proof*⟩

lemma *hilbert-schmidt-scaleC*: ‹*hilbert-schmidt* $(c *_C\ a)$› **if** ‹*hilbert-schmidt a*›
⟨*proof*⟩

lemma *hilbert-schmidt-scaleR*: ‹*hilbert-schmidt* $(r *_R\ a)$› **if** ‹*hilbert-schmidt a*›
⟨*proof*⟩

lemma *hilbert-schmidt-uminus*: ‹*hilbert-schmidt* $(-\ a)$› **if** ‹*hilbert-schmidt a*›
⟨*proof*⟩

lemma *hilbert-schmidt-plus*: ‹*hilbert-schmidt* $(t + u)$› **if** ‹*hilbert-schmidt t*› **and** ‹*hilbert-schmidt u*›
  for *t u* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*›
— [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 possible additionally allow us to weaken the sort of '*b* to *complex-inner*.
⟨*proof*⟩

lemma *hilbert-schmidt-minus*: ‹*hilbert-schmidt* $(a - b)$› **if** ‹*hilbert-schmidt a*› **and** ‹*hilbert-schmidt b*›
  for *a b* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*›
  ⟨*proof*⟩

typedef (**overloaded**) ('*a*::*chilbert-space*,'*b*::*complex-inner*) *hilbert-schmidt* = ‹*Collect hilbert-schmidt* :: ('*a* $\Rightarrow_{CL}$ '*b*) *set*›
  ⟨*proof*⟩
setup-lifting *type-definition-hilbert-schmidt*

instantiation *hilbert-schmidt* :: (*chilbert-space*, *chilbert-space*)
  {*zero*,*scaleC*,*uminus*,*plus*,*minus*,*dist-norm*,*sgn-div-norm*,*uniformity-dist*,*open-uniformity*} **begin**
lift-definition *zero-hilbert-schmidt* :: ‹('*a*,'*b*) *hilbert-schmidt*› **is** *0* ⟨*proof*⟩
lift-definition *norm-hilbert-schmidt* :: ‹('*a*,'*b*) *hilbert-schmidt* ⇒ *real*› **is** *hilbert-schmidt-norm*
⟨*proof*⟩
lift-definition *scaleC-hilbert-schmidt* :: ‹*complex* ⇒ ('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt*› **is** *scaleC*
  ⟨*proof*⟩
lift-definition *scaleR-hilbert-schmidt* :: ‹*real* ⇒ ('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt*› **is** *scaleR*
  ⟨*proof*⟩
lift-definition *uminus-hilbert-schmidt* :: ‹('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt*› **is** *uminus*
  ⟨*proof*⟩
lift-definition *minus-hilbert-schmidt* :: ‹('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt*› **is** *minus*
  ⟨*proof*⟩
lift-definition *plus-hilbert-schmidt* :: ‹('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt*› **is** *plus*
  ⟨*proof*⟩
definition ‹*dist a b* = *norm* $(a - b)$› **for** *a b* :: ‹('*a*,'*b*) *hilbert-schmidt*›
definition ‹*sgn x* = *inverse* (*norm x*) $*_R$ *x*› **for** *x* :: ‹('*a*,'*b*) *hilbert-schmidt*›
definition ‹*uniformity* = (*INF e*∈{*0*<..}. *principal* {(*x*::('*a*,'*b*) *hilbert-schmidt*, *y*). *dist x y* < *e*})›
definition ‹*open U* = (∀ *x*∈*U*. $\forall_F$ (*x'*, *y*) *in INF e*∈{*0*<..}. *principal* {(*x*, *y*). *norm* $(x - y)$ < *e*}. *x'* = *x* ⟶ *y* ∈ *U*)› **for** *U* :: ‹('*a*,'*b*) *hilbert-schmidt set*›
instance
⟨*proof*⟩

end

lift-definition *hs-compose* :: ‹('*b*::*chilbert-space*,'*c*::*complex-inner*) *hilbert-schmidt* ⇒ ('*a*::*chilbert-space*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*c*) *hilbert-schmidt*› **is** *cblinfun-compose*
⟨*proof*⟩

lemma
— [1], 18.8 Proposition
fixes *A* :: ‹'*a* :: *chilbert-space* $\Rightarrow_{CL}$ '*b* :: *chilbert-space*›
shows *trace-class-iff-sqrt-hs*: ‹*trace-class A* ⟷ *hilbert-schmidt* (*sqrt-op* (*abs-op A*))› (**is** *?thesis1*)
and *trace-class-iff-hs-times-hs*: ‹*trace-class A* ⟷ (∃ *B* (*C*::'*a*$\Rightarrow_{CL}$'*a*). *hilbert-schmidt B* ∧ *hilbert-schmidt C* ∧ *A* = *B* o_{CL} *C*)› (**is** *?thesis2*)
and *trace-class-iff-abs-hs-times-hs*: ‹*trace-class A* ⟷ (∃ *B* (*C*::'*a*$\Rightarrow_{CL}$'*a*). *hilbert-schmidt B* ∧ *hilbert-schmidt C* ∧ *abs-op A* = *B* o_{CL} *C*)› (**is** *?thesis3*)
⟨*proof*⟩

lemma *trace-exists*:
— [1], Proposition 18.9
assumes ‹*is-onb B*› **and** ‹*trace-class A*›
shows ‹(λ*e*. *e* $\cdot_C$ (*A* $*_V$ *e*)) *summable-on B*›
⟨*proof*⟩

lemma *trace-plus-prelim*:
assumes ‹*trace-class a*› ‹*trace-class b*› ‹*trace-class* (*a*+*b*)›
— We will later strengthen this as *trace-plus* and then hide this fact.
shows ‹*trace* (*a* + *b*) = *trace a* + *trace b*›
⟨*proof*⟩

lemma *hs-times-hs-trace-class*:
fixes *B* :: ‹'*b*::*chilbert-space* $\Rightarrow_{CL}$ '*c*::*chilbert-space*› **and** *C* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*›
assumes ‹*hilbert-schmidt B*› **and** ‹*hilbert-schmidt C*›
shows ‹*trace-class* (*B* o_{CL} *C*)›
— Not an immediate consequence of *trace-class-iff-hs-times-hs* because here the types of *B*, *C* are more general.
⟨*proof*⟩

instantiation *hilbert-schmidt* :: (*chilbert-space*, *chilbert-space*) *complex-vector* **begin**
instance
⟨*proof*⟩
end

instantiation *hilbert-schmidt* :: (*chilbert-space*, *chilbert-space*) *complex-inner* **begin**
lift-definition *cinner-hilbert-schmidt* :: ‹('*a*,'*b*) *hilbert-schmidt* ⇒ ('*a*,'*b*) *hilbert-schmidt* ⇒ *complex*› **is**

‹λb c. *trace* ($b*$ o_{CL} c)› ⟨*proof*⟩
instance
⟨*proof*⟩
end

lemma *hilbert-schmidt-norm-triangle-ineq*:
— [1], Proposition 18.6 (e). We do not use their proof but get it as a simple corollary of the instantiation of *hilbert-schmidt* as a inner product space. The proof by Conway would probably allow us to weaken the sort of *'b* to *complex-inner*.
fixes *a b* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
assumes ‹*hilbert-schmidt a*› ‹*hilbert-schmidt b*›
shows ‹*hilbert-schmidt-norm* (*a* + *b*) ≤ *hilbert-schmidt-norm a* + *hilbert-schmidt-norm b*›
⟨*proof*⟩

lift-definition *adj-hs* :: ‹(*'a::chilbert-space*,*'b::chilbert-space*) *hilbert-schmidt* ⇒ (*'b*,*'a*) *hilbert-schmidt*›
is *adj*
⟨*proof*⟩

lemma *adj-hs-plus*: ‹*adj-hs* (*x* + *y*) = *adj-hs x* + *adj-hs y*›
⟨*proof*⟩

lemma *adj-hs-minus*: ‹*adj-hs* (*x* − *y*) = *adj-hs x* − *adj-hs y*›
⟨*proof*⟩

lemma *norm-adj-hs*[*simp*]: ‹*norm* (*adj-hs x*) = *norm x*›
⟨*proof*⟩

lemma *hilbert-schmidt-norm-geq-norm*:
— [1], Proposition 18.6 (c)
assumes ‹*hilbert-schmidt a*›
shows ‹*norm a* ≤ *hilbert-schmidt-norm a*›
⟨*proof*⟩

11.4 Trace-norm and trace-class, continued

lemma *trace-class-comp-left*: ‹*trace-class* (*a* o_{CL} *b*)› **if** ‹*trace-class a*› **for** *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
— [1], Theorem 18.11 (a)
⟨*proof*⟩

lemma *trace-class-comp-right*: ‹*trace-class* (*a* o_{CL} *b*)› **if** ‹*trace-class b*› **for** *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
— [1], Theorem 18.11 (a)
⟨*proof*⟩

lemma
fixes *B* :: ‹*'a::chilbert-space set*› **and** *A* :: ‹*'a* $\Rightarrow_{CL}$ *'a*› **and** *b* :: ‹*'b::chilbert-space* $\Rightarrow_{CL}$ *'c::chilbert-space*› **and** *c* :: ‹*'c* $\Rightarrow_{CL}$ *'b*›
shows *trace-alt-def*:

— [1], Proposition 18.9
‹*is-onb B* ⟹ *trace A* = (*if trace-class A then* ($\sum_\infty$*e*∈*B*. *e* $\cdot_C$ (*A* $*_V$ *e*)) *else 0*)›
and *trace-hs-times-hs*: ‹*hilbert-schmidt c* ⟹ *hilbert-schmidt b* ⟹ *trace* (*c* o_{CL} *b*) =
((*of-real* (*hilbert-schmidt-norm* ((*c*∗) + *b*)))^2 − (*of-real* (*hilbert-schmidt-norm* ((*c*∗) − *b*)))^2 −
i ∗ (*of-real* (*hilbert-schmidt-norm* (((*c*∗) + i $*_C$ *b*))))^2 +
i ∗ (*of-real* (*hilbert-schmidt-norm* (((*c*∗) − i $*_C$ *b*))))^2) / *4*›
⟨*proof*⟩

lemma *trace-ket-sum*:
fixes *A* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'a ell2*›
assumes ‹*trace-class A*›
shows ‹*trace A* = ($\sum_\infty$*e*. *ket e* $\cdot_C$ (*A* $*_V$ *ket e*))›
⟨*proof*⟩

lemma *trace-one-dim*[*simp*]: ‹*trace A* = *one-dim-iso A*› **for** *A* :: ‹*'a*::*one-dim* $\Rightarrow_{CL}$ *'a*›
⟨*proof*⟩

lemma *trace-has-sum*:
assumes ‹*is-onb E*›
assumes ‹*trace-class t*›
shows ‹((λ*e*. *e* $\cdot_C$ (*t* $*_V$ *e*)) *has-sum trace t*) *E*›
⟨*proof*⟩

lemma *trace-sandwich-isometry*[*simp*]: ‹*trace* (*sandwich U A*) = *trace A*› **if** ‹*isometry U*›
⟨*proof*⟩

lemma *circularity-of-trace*:
— [1], Theorem 18.11 (e)
fixes *a* :: ‹*'a*::*chilbert-space* $\Rightarrow_{CL}$ *'b*::*chilbert-space*› **and** *b* :: ‹*'b* $\Rightarrow_{CL}$ *'a*›
— The proof from [1] only work for square operators, we generalize it
assumes ‹*trace-class a*›
— Actually, *trace-class* (*a* o_{CL} *b*) ∧ *trace-class* (*b* o_{CL} *a*) is sufficient here, see [3] but the proof is more involved. Only *trace-class* (*a* o_{CL} *b*) is not sufficient, see [4].
shows ‹*trace* (*a* o_{CL} *b*) = *trace* (*b* o_{CL} *a*)›
⟨*proof*⟩

lemma *trace-butterfly-comp*: ‹*trace* (*butterfly x y* o_{CL} *a*) = *y* $\cdot_C$ (*a* $*_V$ *x*)›
⟨*proof*⟩

lemma *trace-butterfly*: ‹*trace* (*butterfly x y*) = *y* $\cdot_C$ *x*›
⟨*proof*⟩

lemma *trace-butterfly-comp'*: ‹*trace* (*a* o_{CL} *butterfly x y*) = *y* $\cdot_C$ (*a* $*_V$ *x*)›
⟨*proof*⟩

lemma *trace-norm-adj*[*simp*]: ‹*trace-norm* (*a*∗) = *trace-norm a*›
— [1], Theorem 18.11 (f)
⟨*proof*⟩

lemma *trace-class-adj*[*simp*]: ‹*trace-class* (*a*∗)› **if** ‹*trace-class a*›
⟨*proof*⟩

lift-definition *adj-tc* :: ‹(′*a*::*chilbert-space*, ′*b*::*chilbert-space*) *trace-class* ⇒ (′*b*,′*a*) *trace-class*›
is *adj*
⟨*proof*⟩

lift-definition *selfadjoint-tc* :: ‹(′*a*::*chilbert-space*, ′*a*) *trace-class* ⇒ *bool*› **is** *selfadjoint*⟨*proof*⟩

lemma *selfadjoint-tc-def* ′: ‹*selfadjoint-tc a* ⟷ *adj-tc a* = *a*›
⟨*proof*⟩

lemma *trace-class-finite-dim*′[*simp*]: ‹*trace-class A*› **for** *A* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*::{*cfinite-dim*,*chilbert-space*}›
⟨*proof*⟩

lemma *trace-class-plus*[*simp*]:
fixes *t u* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*::*chilbert-space*›
assumes ‹*trace-class t*› **and** ‹*trace-class u*›
shows ‹*trace-class* (*t* + *u*)›
— [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*]: ‹*trace-class t* ⟹ *trace-class u* ⟹ *trace-class* (*t* − *u*)›
for *t u* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*::*chilbert-space*›
⟨*proof*⟩

lemma *trace-plus*:
assumes ‹*trace-class a*› ‹*trace-class b*›
shows ‹*trace* (*a* + *b*) = *trace a* + *trace b*›
⟨*proof*⟩
hide-fact *trace-plus-prelim*

lemma *trace-class-sum*:
fixes *a* :: ‹′*a* ⇒ ′*b*::*chilbert-space* $\Rightarrow_{CL}$ ′*c*::*chilbert-space*›
assumes ‹⋀*i*. *i*∈*I* ⟹ *trace-class* (*a i*)›
shows ‹*trace-class* (∑*i*∈*I*. *a i*)›
⟨*proof*⟩

lemma
assumes ‹⋀*i*. *i*∈*I* ⟹ *trace-class* (*a i*)›
shows *trace-sum*: ‹*trace* (∑*i*∈*I*. *a i*) = (∑*i*∈*I*. *trace* (*a i*))›
⟨*proof*⟩

lemma *cmod-trace-times*: ‹*cmod* (*trace* (*a* o_{CL} *b*)) ≤ *norm a* ∗ *trace-norm b*› **if** ‹*trace-class b*›

for *b* :: ‹*'a*::*chilbert-space* $\Rightarrow_{CL}$ *'b*::*chilbert-space*›
— [1], Theorem 18.11 (e)
⟨*proof*⟩

lemma *trace-leq-trace-norm*[*simp*]: ‹*cmod* (*trace a*) ≤ *trace-norm a*›
⟨*proof*⟩

lemma *trace-norm-triangle*:
fixes *a b* :: ‹*'a*::*chilbert-space* $\Rightarrow_{CL}$ *'b*::*chilbert-space*›
assumes [*simp*]: ‹*trace-class a*› ‹*trace-class b*›
shows ‹*trace-norm* (*a* + *b*) ≤ *trace-norm a* + *trace-norm b*›
— [1], Theorem 18.11 (a)
⟨*proof*⟩

instantiation *trace-class* :: (*chilbert-space*, *chilbert-space*) {*complex-vector*} **begin**

lift-definition *zero-trace-class* :: ‹(*'a*,*'b*) *trace-class*› **is** *0* ⟨*proof*⟩
lift-definition *minus-trace-class* :: ‹(*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class*› **is** *minus* ⟨*proof*⟩
lift-definition *uminus-trace-class* :: ‹(*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class*› **is** *uminus* ⟨*proof*⟩
lift-definition *plus-trace-class* :: ‹(*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class*› **is** *plus* ⟨*proof*⟩
lift-definition *scaleC-trace-class* :: ‹*complex* ⇒ (*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class*› **is** *scaleC* ⟨*proof*⟩
lift-definition *scaleR-trace-class* :: ‹*real* ⇒ (*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class*› **is** *scaleR* ⟨*proof*⟩
instance
⟨*proof*⟩
end

lemma *from-trace-class-0*[*simp*]: ‹*from-trace-class 0* = *0*›
⟨*proof*⟩

lemma *not-not-singleton-tc-zero*:
‹*x* = *0*› **if** ‹¬ *class.not-singleton TYPE*(*'a*)› **for** *x* :: ‹(*'a*::*chilbert-space*,*'b*::*chilbert-space*) *trace-class*›
⟨*proof*⟩

instantiation *trace-class* :: (*chilbert-space*, *chilbert-space*) {*complex-normed-vector*} **begin**

lift-definition *norm-trace-class* :: ‹(*'a*,*'b*) *trace-class* ⇒ *real*› **is** *trace-norm* ⟨*proof*⟩
definition *sgn-trace-class* :: ‹(*'a*,*'b*) *trace-class* ⇒ (*'a*,*'b*) *trace-class*› **where** ‹*sgn-trace-class a* = *a* $/_R$ *norm a*›
definition *dist-trace-class* :: ‹(*'a*,*'b*) *trace-class* ⇒ - ⇒ -› **where** ‹*dist-trace-class a b* = *norm* (*a* − *b*)›
definition [*code del*]: *uniformity-trace-class* = (*INF e*∈{*0*<..}. *principal* {(*x*::(*'a*,*'b*) *trace-class*, *y*). *dist x y* < *e*})
definition [*code del*]: *open-trace-class U* = (∀ *x*∈*U*. ∀$_F$ (*x'*, *y*) *in INF e*∈{*0*<..}. *principal* {(*x*, *y*). *dist x y* < *e*}. *x'* = *x* ⟶ *y* ∈ *U*) **for** *U* :: (*'a*,*'b*) *trace-class set*

instance
⟨*proof*⟩
end

lemma *trace-norm-comp-right*:
fixes *a* :: ‹′*b*::*chilbert-space* $\Rightarrow_{CL}$ ′*c*::*chilbert-space*› **and** *b* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*›
assumes ‹*trace-class b*›
shows ‹*trace-norm* (*a* o_{CL} *b*) ≤ *norm a* ∗ *trace-norm b*›
— [1], Theorem 18.11 (g)
⟨*proof*⟩

lemma *trace-norm-comp-left*:
— [1], Theorem 18.11 (g)
fixes *a* :: ‹′*b*::*chilbert-space* $\Rightarrow_{CL}$ ′*c*::*chilbert-space*› **and** *b* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*›
assumes [*simp*]: ‹*trace-class a*›
shows ‹*trace-norm* (*a* o_{CL} *b*) ≤ *trace-norm a* ∗ *norm b*›
⟨*proof*⟩

lemma *bounded-clinear-trace-duality*: ‹*trace-class t* ⟹ *bounded-clinear* (λ*a*. *trace* (*t* o_{CL} *a*))›
⟨*proof*⟩

lemma *trace-class-butterfly*[*simp*]: ‹*trace-class* (*butterfly x y*)› **for** *x* :: ‹′*a*::*chilbert-space*› **and** *y* :: ‹′*b*::*chilbert-space*›
⟨*proof*⟩

lemma *trace-adj*: ‹*trace* (*a*∗) = *cnj* (*trace a*)›
⟨*proof*⟩
hide-fact *trace-adj-prelim*

lemma *cmod-trace-times′*: ‹*cmod* (*trace* (*a* o_{CL} *b*)) ≤ *norm b* ∗ *trace-norm a*› **if** ‹*trace-class a*›
— [1], Theorem 18.11 (e)
⟨*proof*⟩

lift-definition *iso-trace-class-compact-op-dual′* :: ‹(′*a*::*chilbert-space*,′*b*::*chilbert-space*) *trace-class* ⇒ (′*b*,′*a*) *compact-op* $\Rightarrow_{CL}$ *complex*› **is**
‹λ*t c*. *trace* (*from-compact-op c* o_{CL} *t*)›
⟨*proof*⟩
include *lifting-syntax*
⟨*proof*⟩

lemma *iso-trace-class-compact-op-dual′-apply*: ‹*iso-trace-class-compact-op-dual′ t c* = *trace* (*from-compact-op c* o_{CL} *from-trace-class t*)›
⟨*proof*⟩

lemma *iso-trace-class-compact-op-dual′-plus*: ‹*iso-trace-class-compact-op-dual′* (*a* + *b*) = *iso-trace-class-compact-op-dual′* *a* + *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* $*_C$ *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* (*iso-trace-class-compact-op-dual′* :: (′*a*::*chilbert-space*,′*b*::*chilbert-space*) *trace-class* ⇒ -)›
⟨*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*: ‹ψ $\cdot_C$ (*abs-op* *t* $*_V$ ψ) ≤ *trace-norm* *t*› **if** ‹*trace-class* *t*› **and** ‹*norm* ψ = *1*›
⟨*proof*⟩

lemma *norm-leq-trace-norm*: ‹*norm* *t* ≤ *trace-norm* *t*› **if** ‹*trace-class* *t*›
for *t* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*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* :: ‹('a::chilbert-space ⇒$_{CL}$ 'b::chilbert-space) ⇒ ('c::chilbert-space, 'a) trace-class ⇒ ('c,'b) trace-class› **is**
‹*cblinfun-compose* :: 'a ⇒$_{CL}$ 'b ⇒ 'c ⇒$_{CL}$ 'a ⇒ 'c ⇒$_{CL}$ 'b›
⟨*proof*⟩

lemma *norm-compose-tcl*: ‹norm (compose-tcl a b) ≤ norm a * norm b›
⟨*proof*⟩

lemma *norm-compose-tcr*: ‹norm (compose-tcr a b) ≤ norm a * norm b›
⟨*proof*⟩

interpretation *compose-tcl*: *bounded-cbilinear compose-tcl*
⟨*proof*⟩

interpretation *compose-tcr*: *bounded-cbilinear compose-tcr*
⟨*proof*⟩

lemma *trace-norm-sandwich*: ‹trace-norm (sandwich e t) ≤ (norm e)^2 * trace-norm t› **if** ‹trace-class t›
⟨*proof*⟩

lemma *trace-class-sandwich*: ‹trace-class b ⟹ trace-class (sandwich a b)›
⟨*proof*⟩

definition ‹sandwich-tc e t = compose-tcl (compose-tcr e t) (e*)›

lemma *sandwich-tc-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹((=) ===> cr-trace-class ===> cr-trace-class) (λe. (*$_V$) (sandwich e)) sandwich-tc›
⟨*proof*⟩

lemma *from-trace-class-sandwich-tc*:
‹from-trace-class (sandwich-tc e t) = sandwich e (from-trace-class t)›
⟨*proof*⟩

lemma *norm-sandwich-tc*: ‹*norm* (*sandwich-tc e t*) ≤ (*norm e*)^*2* ∗ *norm t*›
⟨*proof*⟩

lemma *sandwich-tc-pos*: ‹*sandwich-tc e t* ≥ *0*› **if** ‹*t* ≥ *0*›
⟨*proof*⟩

lemma *sandwich-tc-scaleC-right*: ‹*sandwich-tc e* (*c* $*_C$ *t*) = *c* $*_C$ *sandwich-tc e t*›
⟨*proof*⟩

lemma *sandwich-tc-plus*: ‹*sandwich-tc e* (*t* + *u*) = *sandwich-tc e t* + *sandwich-tc e u*›
⟨*proof*⟩

lemma *sandwich-tc-minus*: ‹*sandwich-tc e* (*t* − *u*) = *sandwich-tc e t* − *sandwich-tc e u*›
⟨*proof*⟩

lemma *sandwich-tc-uminus-right*: ‹*sandwich-tc e* (− *t*) = − *sandwich-tc e t*›
⟨*proof*⟩

lemma *trace-comp-pos*:
fixes *a b* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*a*›
assumes ‹*trace-class b*›
assumes ‹*a* ≥ *0*› **and** ‹*b* ≥ *0*›
shows ‹*trace* (*a* o_{CL} *b*) ≥ *0*›
⟨*proof*⟩

lemma *trace-norm-one-dim*: ‹*trace-norm x* = *cmod* (*one-dim-iso x*)›
⟨*proof*⟩

lemma *trace-norm-bounded*:
fixes *A B* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*a*›
assumes ‹*A* ≥ *0*› **and** ‹*trace-class B*›
assumes ‹*A* ≤ *B*›
shows ‹*trace-class A*›
⟨*proof*⟩

lemma *trace-norm-cblinfun-mono*:
fixes *A B* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*a*›
assumes ‹*A* ≥ *0*› **and** ‹*trace-class B*›
assumes ‹*A* ≤ *B*›
shows ‹*trace-norm A* ≤ *trace-norm B*›
⟨*proof*⟩

lemma *norm-cblinfun-mono-trace-class*:
fixes *A B* :: ‹('*a*::*chilbert-space*, '*a*) *trace-class*›

assumes ‹$A \geq 0$›
assumes ‹$A \leq B$›
shows ‹*norm* $A \leq$ *norm* B›
⟨*proof*⟩

lemma *trace-norm-butterfly*: ‹*trace-norm* (*butterfly* a b) = (*norm* a) $*$ (*norm* b)›
for a b :: ‹- :: *chilbert-space*›
⟨*proof*⟩

lemma *from-trace-class-sum*:
shows ‹*from-trace-class* ($\sum x{\in}M.\ f\ x$) = ($\sum x{\in}M.$ *from-trace-class* ($f\ x$))›
⟨*proof*⟩

lemma *has-sum-mono-neutral-traceclass*:
fixes f :: $'a \Rightarrow$ ($'b$::*chilbert-space*, $'b$) *trace-class*
assumes ‹(f *has-sum* a) A› **and** (g *has-sum* b) B
assumes ‹$\bigwedge x.\ x \in A{\cap}B \Longrightarrow f\ x \leq g\ x$›
assumes ‹$\bigwedge x.\ x \in A{-}B \Longrightarrow f\ x \leq 0$›
assumes ‹$\bigwedge x.\ x \in B{-}A \Longrightarrow g\ x \geq 0$›
shows $a \leq b$
⟨*proof*⟩

lemma *has-sum-mono-traceclass*:
fixes f :: $'a \Rightarrow$ ($'b$::*chilbert-space*, $'b$) *trace-class*
assumes (f *has-sum* x) A **and** (g *has-sum* y) A
assumes ‹$\bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x$›
shows $x \leq y$
⟨*proof*⟩

lemma *infsum-mono-traceclass*:
fixes f :: $'a \Rightarrow$ ($'b$::*chilbert-space*, $'b$) *trace-class*
assumes f *summable-on* A **and** g *summable-on* A
assumes ‹$\bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x$›
shows *infsum* f $A \leq$ *infsum* g A
⟨*proof*⟩

lemma *infsum-mono-neutral-traceclass*:
fixes f :: $'a \Rightarrow$ ($'b$::*chilbert-space*, $'b$) *trace-class*
assumes f *summable-on* A **and** g *summable-on* B
assumes ‹$\bigwedge x.\ x \in A{\cap}B \Longrightarrow f\ x \leq g\ x$›
assumes ‹$\bigwedge x.\ x \in A{-}B \Longrightarrow f\ x \leq 0$›
assumes ‹$\bigwedge x.\ x \in B{-}A \Longrightarrow g\ x \geq 0$›
shows *infsum* f $A \leq$ *infsum* g B
⟨*proof*⟩

instance *trace-class* :: (*chilbert-space*, *chilbert-space*) *ordered-complex-vector*
⟨*proof*⟩

lemma *Abs-trace-class-geq0I*: ‹$0 \leq$ *Abs-trace-class t*› **if** ‹*trace-class t*› **and** ‹$t \geq 0$›
⟨*proof*⟩

lift-definition *tc-compose* :: ‹('*b*::*chilbert-space*, '*c*::*chilbert-space*) *trace-class*
⇒ ('*a*::*chilbert-space*, '*b*) *trace-class* ⇒ ('*a*,'*c*) *trace-class*› **is**
cblinfun-compose
⟨*proof*⟩

lemma *norm-tc-compose*:
‹*norm* (*tc-compose a b*) $\leq$ *norm a* $*$ *norm b*›
⟨*proof*⟩

lift-definition *trace-tc* :: ‹('*a*::*chilbert-space*, '*a*) *trace-class* ⇒ *complex*› **is** *trace*⟨*proof*⟩

lemma *trace-tc-plus*: ‹*trace-tc* $(a + b) =$ *trace-tc a* $+$ *trace-tc b*›
⟨*proof*⟩

lemma *trace-tc-scaleC*: ‹*trace-tc* $(c *_C a) = c *_C$ *trace-tc a*›
⟨*proof*⟩

lemma *trace-tc-norm*: ‹*norm* (*trace-tc a*) $\leq$ *norm a*›
⟨*proof*⟩

lemma *bounded-clinear-trace-tc*[*bounded-clinear*, *simp*]: ‹*bounded-clinear trace-tc*›
⟨*proof*⟩

lemma *norm-tc-pos*: ‹*norm A* = *trace-tc A*› **if** ‹$A \geq 0$›
⟨*proof*⟩

lemma *norm-tc-pos-Re*: ‹*norm A* = *Re* (*trace-tc A*)› **if** ‹$A \geq 0$›
⟨*proof*⟩

lemma *from-trace-class-pos*: ‹*from-trace-class A* $\geq 0 \longleftrightarrow A \geq 0$›
⟨*proof*⟩

lemma *infsum-tc-norm-bounded-abs-summable*:
fixes *A* :: ‹'*a* ⇒ ('*b*::*chilbert-space*, '*b*::*chilbert-space*) *trace-class*›
assumes *pos*: ‹$\bigwedge x.\ x \in M \Longrightarrow A\ x \geq 0$›
assumes *bound-B*: ‹$\bigwedge F.$ *finite* $F \Longrightarrow F \subseteq M \Longrightarrow$ *norm* $(\sum x{\in}F.\ A\ x) \leq B$›
shows ‹*A abs-summable-on M*›
⟨*proof*⟩

lemma *trace-norm-uminus*[*simp*]: ‹*trace-norm* $(-a) =$ *trace-norm a*›
⟨*proof*⟩

lemma *trace-norm-triangle-minus*:

fixes a b :: ‹$'a$::*chilbert-space* $\Rightarrow_{CL}$ $'b$::*chilbert-space*›
assumes [*simp*]: ‹*trace-class a*› ‹*trace-class b*›
shows ‹*trace-norm* $(a - b) \leq$ *trace-norm* a + *trace-norm* b›
⟨*proof*⟩

lemma *trace-norm-abs-op*[*simp*]: ‹*trace-norm* (*abs-op t*) = *trace-norm t*›
⟨*proof*⟩

lemma
fixes t :: ‹$'a \Rightarrow_{CL}$ $'a$::*chilbert-space*›
shows *cblinfun-decomp-4pos*: ‹
$\exists$ *t1 t2 t3 t4*.
$t = t1 - t2 + i *_C t3 - i *_C t4$
$\wedge\ t1 \geq 0 \wedge t2 \geq 0 \wedge t3 \geq 0 \wedge t4 \geq 0$› (**is** *?thesis1*)
and *trace-class-decomp-4pos*: ‹*trace-class t* $\Longrightarrow$
$\exists$ *t1 t2 t3 t4*.
$t = t1 - t2 + i *_C t3 - i *_C t4$
$\wedge$ *trace-class t1* $\wedge$ *trace-class t2* $\wedge$ *trace-class t3* $\wedge$ *trace-class t4*
$\wedge$ *trace-norm t1* $\leq$ *trace-norm t* $\wedge$ *trace-norm t2* $\leq$ *trace-norm t* $\wedge$ *trace-norm t3*
$\leq$ *trace-norm t* $\wedge$ *trace-norm t4* $\leq$ *trace-norm t*
$\wedge\ t1 \geq 0 \wedge t2 \geq 0 \wedge t3 \geq 0 \wedge t4 \geq 0$› (**is** ‹- $\Longrightarrow$ *?thesis2*›)
⟨*proof*⟩

lemma *trace-class-decomp-4pos'*:
fixes t :: ‹($'a$::*chilbert-space*,$'a$) *trace-class*›
shows ‹$\exists$ *t1 t2 t3 t4*.
$t = t1 - t2 + i *_C t3 - i *_C t4$
$\wedge$ *norm t1* $\leq$ *norm t* $\wedge$ *norm t2* $\leq$ *norm t* $\wedge$ *norm t3* $\leq$ *norm t* $\wedge$ *norm t4* $\leq$ *norm*
t
$\wedge\ t1 \geq 0 \wedge t2 \geq 0 \wedge t3 \geq 0 \wedge t4 \geq 0$›
⟨*proof*⟩

thm *bounded-clinear-trace-duality*
lemma *bounded-clinear-trace-duality'*: ‹*trace-class t* $\Longrightarrow$ *bounded-clinear* (λa. *trace* (a o_{CL} t))›
for t :: ‹-::*chilbert-space* $\Rightarrow_{CL}$ -::*chilbert-space*›
⟨*proof*⟩

lemma *infsum-nonneg-traceclass*:
fixes f :: $'a \Rightarrow$ ($'b$::*chilbert-space*, $'b$) *trace-class*
assumes $\bigwedge x.\ x \in M \Longrightarrow 0 \leq f\ x$
shows *infsum f M* ≥ 0
⟨*proof*⟩

lemma *sandwich-tc-compose*: ‹*sandwich-tc* (A o_{CL} B) = *sandwich-tc A o sandwich-tc B*›
⟨*proof*⟩

lemma *sandwich-tc-0-left*[*simp*]: ‹*sandwich-tc 0* = *0*›
⟨*proof*⟩

lemma *sandwich-tc-0-right*[*simp*]: ‹*sandwich-tc e 0 = 0*›
⟨*proof*⟩

lemma *sandwich-tc-scaleC-left*: ‹*sandwich-tc* (*c* $*_C$ *e*) *t* = (*cmod c*)^*2* $*_C$ *sandwich-tc e t*›
⟨*proof*⟩

lemma *sandwich-tc-scaleR-left*: ‹*sandwich-tc* (*r* $*_R$ *e*) *t* = *r*^*2* $*_R$ *sandwich-tc e t*›
⟨*proof*⟩

lemma *bounded-cbilinear-tc-compose*: ‹*bounded-cbilinear tc-compose*›
⟨*proof*⟩
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* :: ‹*'a*::*chilbert-space* ⇒ *'b*::*chilbert-space* ⇒ (*'b*,*'a*) *trace-class*›
is *butterfly*
⟨*proof*⟩

lemma *norm-tc-butterfly*: ‹*norm* (*tc-butterfly* ψ φ) = *norm* ψ * *norm* φ›
⟨*proof*⟩

lemma *trace-tc-butterfly*: ‹*trace-tc* (*tc-butterfly x y*) = *y* $\bullet_C$ *x*›
⟨*proof*⟩

lemma *comp-tc-butterfly*[*simp*]: ‹*tc-compose* (*tc-butterfly a b*) (*tc-butterfly c d*) = (*b* $\bullet_C$ *c*) $*_C$ *tc-butterfly a d*›
⟨*proof*⟩

lemma *tc-butterfly-pos*[*simp*]: ‹*0* ≤ *tc-butterfly* ψ ψ›
⟨*proof*⟩

lift-definition *rank1-tc* :: ‹(*'a*::*chilbert-space*, *'b*::*chilbert-space*) *trace-class* ⇒ *bool*› **is** *rank1*⟨*proof*⟩
lift-definition *finite-rank-tc* :: ‹(*'a*::*chilbert-space*, *'b*::*chilbert-space*) *trace-class* ⇒ *bool*› **is** *finite-rank*⟨*proof*⟩

lemma *finite-rank-tc-0*[*iff*]: ‹*finite-rank-tc 0*›
⟨*proof*⟩

lemma *finite-rank-tc-plus*: ‹*finite-rank-tc* (*a* + *b*)›
if ‹*finite-rank-tc a*› **and** ‹*finite-rank-tc b*›
⟨*proof*⟩

lemma *finite-rank-tc-scale*: ‹*finite-rank-tc* (*c* $*_C$ *a*)› **if** ‹*finite-rank-tc a*›
⟨*proof*⟩

lemma *csubspace-finite-rank-tc*: ‹*csubspace* (*Collect finite-rank-tc*)›
⟨*proof*⟩

lemma *rank1-trace-class*: ‹*trace-class a*› **if** ‹*rank1 a*›
for *a b* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
⟨*proof*⟩

lemma *finite-rank-trace-class*: ‹*trace-class a*› **if** ‹*finite-rank a*›
for *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
⟨*proof*⟩

lemma *trace-minus*:
assumes ‹*trace-class a*› ‹*trace-class b*›
shows ‹*trace* (*a* − *b*) = *trace a* − *trace b*›
⟨*proof*⟩

lemma *trace-cblinfun-mono*:
fixes *A B* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'a*›
assumes ‹*trace-class A*› **and** ‹*trace-class B*›
assumes ‹*A* ≤ *B*›
shows ‹*trace A* ≤ *trace B*›
⟨*proof*⟩

lemma *trace-tc-mono*:
assumes ‹*A* ≤ *B*›
shows ‹*trace-tc A* ≤ *trace-tc B*›
⟨*proof*⟩

lemma *trace-tc-0*[*simp*]: ‹*trace-tc 0* = *0*›
⟨*proof*⟩

lemma *cspan-tc-transfer*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-trace-class* ===> *rel-set cr-trace-class*) *cspan cspan*›
⟨*proof*⟩

lemma *finite-rank-tc-def'*: ‹*finite-rank-tc A* ⟷ *A* ∈ *cspan* (*Collect rank1-tc*)›
⟨*proof*⟩

lemma *tc-butterfly-add-left*: ‹*tc-butterfly* (*a* + *a'*) *b* = *tc-butterfly a b* + *tc-butterfly a' b*›
⟨*proof*⟩

lemma *tc-butterfly-add-right*: ‹*tc-butterfly a* (*b* + *b'*) = *tc-butterfly a b* + *tc-butterfly a b'*›
⟨*proof*⟩

lemma *tc-butterfly-sum-left*: ‹*tc-butterfly* (∑*i*∈*M*. *ψ i*) *φ* = (∑*i*∈*M*. *tc-butterfly* (*ψ i*) *φ*)›
⟨*proof*⟩

lemma *tc-butterfly-sum-right*: ‹*tc-butterfly ψ* (∑*i*∈*M*. *φ i*) = (∑*i*∈*M*. *tc-butterfly ψ* (*φ i*))›
⟨*proof*⟩

lemma *tc-butterfly-scaleC-left*[*simp*]: *tc-butterfly* $(c *_C \psi)$ $\varphi = c *_C$ *tc-butterfly* ψ φ
⟨*proof*⟩

lemma *tc-butterfly-scaleC-right*[*simp*]: *tc-butterfly* ψ $(c *_C \varphi)$ = *cnj* $c *_C$ *tc-butterfly* ψ φ
⟨*proof*⟩

lemma *bounded-sesquilinear-tc-butterfly*[*iff*]: ‹*bounded-sesquilinear* (λ*a b. tc-butterfly b a*)›
⟨*proof*⟩

lemma *trace-norm-plus-orthogonal*:
assumes ‹*trace-class a*› **and** ‹*trace-class b*›
assumes ‹$a* \; o_{CL} \; b = 0$› **and** ‹$a \; o_{CL} \; b* = 0$›
shows ‹*trace-norm* (*a* + *b*) = *trace-norm a* + *trace-norm b*›
⟨*proof*⟩

lemma *norm-tc-plus-orthogonal*:
assumes ‹*tc-compose* (*adj-tc a*) *b* = *0*› **and** ‹*tc-compose a* (*adj-tc b*) = *0*›
shows ‹*norm* (*a* + *b*) = *norm a* + *norm b*›
⟨*proof*⟩

lemma *trace-norm-sum-exchange*:
fixes *t* :: ‹- ⇒ (-::*chilbert-space* $\Rightarrow_{CL}$ -::*chilbert-space*)›
assumes ‹$\bigwedge i.\; i \in F \Longrightarrow$ *trace-class* (*t i*)›
assumes ‹$\bigwedge i\, j.\; i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow (t\; i)* \; o_{CL} \; t\; j = 0$›
assumes ‹$\bigwedge i\, j.\; i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow t\; i \; o_{CL} \; (t\; j)* = 0$›
shows ‹*trace-norm* ($\sum i{\in}F.\; t\; i$) = ($\sum i{\in}F.$ *trace-norm* (*t i*))›
⟨*proof*⟩

lemma *norm-tc-sum-exchange*:
assumes ‹$\bigwedge i\, j.\; i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow$ *tc-compose* (*adj-tc* (*t i*)) (*t j*) = *0*›
assumes ‹$\bigwedge i\, j.\; i \in F \Longrightarrow j \in F \Longrightarrow i \neq j \Longrightarrow$ *tc-compose* (*t i*) (*adj-tc* (*t j*)) = *0*›
shows ‹*norm* ($\sum i{\in}F.\; t\; i$) = ($\sum i{\in}F.$ *norm* (*t i*))›
⟨*proof*⟩

instantiation *trace-class* :: (*one-dim*, *one-dim*) *complex-inner* **begin**
lift-definition *cinner-trace-class* :: ‹('*a*, '*b*) *trace-class* ⇒ ('*a*, '*b*) *trace-class* ⇒ *complex*› **is** ‹$(\cdot_C)$›⟨*proof*⟩
instance
⟨*proof*⟩
end

instantiation *trace-class* :: (*one-dim*, *one-dim*) *one-dim* **begin**
lift-definition *one-trace-class* :: ‹('*a*, '*b*) *trace-class*› **is** *1*
⟨*proof*⟩

lift-definition *times-trace-class* :: ‹(′a, ′b) trace-class ⇒ (′a, ′b) trace-class ⇒ (′a, ′b) trace-class›
is ‹(∗)›
⟨*proof*⟩
lift-definition *divide-trace-class* :: ‹(′a, ′b) trace-class ⇒ (′a, ′b) trace-class ⇒ (′a, ′b) trace-class›
is ‹(/)›
⟨*proof*⟩
lift-definition *inverse-trace-class* :: ‹(′a, ′b) trace-class ⇒ (′a, ′b) trace-class› **is** ‹*Fields.inverse*›
⟨*proof*⟩
definition *canonical-basis-trace-class* :: ‹(′a, ′b) trace-class list› **where** ‹*canonical-basis-trace-class* = [*1*]›
definition *canonical-basis-length-trace-class* :: ‹(′a, ′b) trace-class itself ⇒ nat› **where** ‹*canonical-basis-length-trace-class* = *1*›
instance
⟨*proof*⟩
end

lemma *from-trace-class-one-dim-iso*[*simp*]: ‹*from-trace-class* = *one-dim-iso*›
⟨*proof*⟩

lemma *trace-tc-one-dim-iso*[*simp*]: ‹*trace-tc* = *one-dim-iso*›
⟨*proof*⟩

lemma *compose-tcr-id-left*[*simp*]: ‹*compose-tcr id-cblinfun t* = *t*›
⟨*proof*⟩

lemma *compose-tcl-id-right*[*simp*]: ‹*compose-tcl t id-cblinfun* = *t*›
⟨*proof*⟩

lemma *sandwich-tc-id-cblinfun*[*simp*]: ‹*sandwich-tc id-cblinfun t* = *t*›
⟨*proof*⟩

lemma *bounded-clinear-sandwich-tc*[*bounded-clinear*]: ‹*bounded-clinear* (*sandwich-tc e*)›
⟨*proof*⟩

lemma *trace-class-Proj*: ‹*trace-class* (*Proj S*) ⟷ *finite-dim-ccsubspace S*›
⟨*proof*⟩

lemma *not-trace-class-trace0*: ‹*trace a* = *0*› **if** ‹¬ *trace-class a*›
⟨*proof*⟩

lemma *trace-Proj*: ‹*trace* (*Proj S*) = *cdim* (*space-as-set S*)›
⟨*proof*⟩

lemma *trace-tc-pos*: ‹*t* ≥ *0* ⟹ *trace-tc t* ≥ *0*›
⟨*proof*⟩

lift-definition *tc-apply* :: ‹(′a::chilbert-space,′b::chilbert-space) trace-class ⇒ ′a ⇒ ′b› **is** *cblin-*

fun-apply⟨*proof*⟩

lemma *bounded-cbilinear-tc-apply*: ‹*bounded-cbilinear tc-apply*›
⟨*proof*⟩

lift-definition *diagonal-operator-tc* :: ‹(*'a* ⇒ *complex*) ⇒ (*'a ell2*, *'a ell2*) *trace-class*› **is**
‹*λf. if f abs-summable-on UNIV then diagonal-operator f else 0*›
⟨*proof*⟩

lemma *from-trace-class-diagonal-operator-tc*:
assumes ‹*f abs-summable-on UNIV*›
shows ‹*from-trace-class* (*diagonal-operator-tc f*) = *diagonal-operator f*›
⟨*proof*⟩

lemma *tc-butterfly-scaleC-summable*:
fixes *f* :: ‹*'a* ⇒ *complex*›
assumes ‹*f abs-summable-on A*›
shows ‹(*λx. f x* $*_C$ *tc-butterfly* (*ket x*) (*ket x*)) *summable-on A*›
⟨*proof*⟩

lemma *tc-butterfly-scaleC-has-sum*:
fixes *f* :: ‹*'a* ⇒ *complex*›
assumes ‹*f abs-summable-on UNIV*›
shows ‹((*λx. f x* $*_C$ *tc-butterfly* (*ket x*) (*ket x*)) *has-sum diagonal-operator-tc f*) *UNIV*›
⟨*proof*⟩

lemma *diagonal-operator-tc-invalid*: ‹¬ *f abs-summable-on UNIV* ⟹ *diagonal-operator-tc f* = *0*›
⟨*proof*⟩

lemma *tc-butterfly-scaleC-infsum*:
fixes *f* :: ‹*'a* ⇒ *complex*›
shows ‹($\sum_\infty$*x. f x* $*_C$ *tc-butterfly* (*ket x*) (*ket x*)) = *diagonal-operator-tc f*›
⟨*proof*⟩

lemma *from-trace-class-abs-summable*: ‹*f abs-summable-on X* ⟹ (*λx. from-trace-class* (*f x*)) *abs-summable-on X*›
⟨*proof*⟩

lemma *from-trace-class-summable*: ‹*f summable-on X* ⟹ (*λx. from-trace-class* (*f x*)) *summable-on X*›
⟨*proof*⟩

lemma *from-trace-class-infsum*:
assumes ‹*f summable-on UNIV*›

shows ‹*from-trace-class* ($\sum_\infty$*x*. *f x*) = ($\sum_\infty$*x*. *from-trace-class* (*f x*))›
⟨*proof*⟩

lemma *cspan-trace-class*:
‹*cspan* (*Collect trace-class* :: (*'a*::*chilbert-space* $\Rightarrow_{CL}$ *'b*::*chilbert-space*) *set*) = *Collect trace-class*›
⟨*proof*⟩

lemma *monotone-convergence-tc*:
fixes *f* :: ‹*'b* ⇒ (*'a*, *'a*::*chilbert-space*) *trace-class*›
assumes *bounded*: ‹$\forall_F$ *x in F*. *trace-tc* (*f x*) ≤ *B*›
assumes *pos*: ‹$\forall_F$ *x in F*. *f x* ≥ *0*›
assumes *increasing*: ‹*increasing-filter* (*filtermap f F*)›
shows ‹∃ *L*. (*f* ⟶ *L*) *F*›
⟨*proof*⟩

lemma *nonneg-bdd-above-summable-on-tc*:
fixes *f* :: ‹*'a* ⇒ (*'c*::*chilbert-space*, *'c*) *trace-class*›
assumes *pos*: ‹⋀*x*. *x*∈*A* ⟹ *f x* ≥ *0*›
assumes *bdd*: ‹*bdd-above* (*trace-tc* ‘ *sum f* ‘ {*F*. *F*⊆*A* ∧ *finite F*})›
shows ‹*f summable-on A*›
⟨*proof*⟩

lemma *summable-Sigma-positive-tc*:
fixes *f* :: ‹*'a* ⇒ *'b* ⇒ (*'c*, *'c*::*chilbert-space*) *trace-class*›
assumes ‹⋀*x*. *x*∈*X* ⟹ *f x summable-on Y x*›
assumes ‹(λ*x*. $\sum_\infty$*y*∈*Y x*. *f x y*) *summable-on X*›
assumes ‹⋀*x y*. *x* ∈ *X* ⟹ *y* ∈ *Y x* ⟹ *f x y* ≥ *0*›
shows ‹(λ(*x*, *y*). *f x y*) *summable-on* (*SIGMA x*:*X*. *Y x*)›
⟨*proof*⟩

lemma *infsum-Sigma-positive-tc*:
fixes *f* :: ‹*'a* ⇒ *'b* ⇒ (*'c*::*chilbert-space*, *'c*) *trace-class*›
assumes ‹⋀*x*. *x*∈*X* ⟹ *f x summable-on Y x*›
assumes ‹⋀*x y*. *x* ∈ *X* ⟹ *y* ∈ *Y x* ⟹ *f x y* ≥ *0*›
shows ‹($\sum_\infty$*x*∈*X*. $\sum_\infty$*y*∈*Y x*. *f x y*) = ($\sum_\infty$(*x*,*y*)∈*Sigma X Y*. *f x y*)›
⟨*proof*⟩

lemma *infsum-swap-positive-tc*:
fixes *f* :: ‹*'a* ⇒ *'b* ⇒ (*'c*::*chilbert-space*, *'c*) *trace-class*›
assumes ‹⋀*x*. *x*∈*X* ⟹ *f x summable-on Y*›
assumes ‹⋀*y*. *y*∈*Y* ⟹ (λ*x*. *f x y*) *summable-on X*›
assumes ‹⋀*x y*. *x* ∈ *X* ⟹ *y* ∈ *Y* ⟹ *f x y* ≥ *0*›
shows ‹($\sum_\infty$*x*∈*X*. $\sum_\infty$*y*∈*Y*. *f x y*) = ($\sum_\infty$*y*∈*Y*. $\sum_\infty$*x*∈*X*. *f x y*)›
⟨*proof*⟩

lemma *separating-density-ops*:
  assumes ‹$B > 0$›
  shows ‹*separating-set clinear* {*t* :: (*'a::chilbert-space*, *'a*) *trace-class*. $0 \leq t \wedge$ *norm* $t \leq B$}›
⟨*proof*⟩

lemma *summable-abs-summable-tc*:
  fixes *f* :: ‹*'a* ⇒ (*'b::chilbert-space*,*'b*) *trace-class*›
  assumes ‹*f summable-on X*›
  assumes ‹$\bigwedge x.\ x \in X \Longrightarrow f\ x \geq 0$›
  shows ‹*f abs-summable-on X*›
⟨*proof*⟩

lemma *sandwich-tc-eq0-D*:
  assumes *eq0*: ‹$\bigwedge \varrho.\ \varrho \geq 0 \Longrightarrow$ *norm* $\varrho \leq B \Longrightarrow$ *sandwich-tc a* $\varrho = 0$›
  assumes *Bpos*: ‹$B > 0$›
  shows ‹$a = 0$›
⟨*proof*⟩

lemma *sandwich-tc-butterfly*: ‹*sandwich-tc c* (*tc-butterfly a b*) = *tc-butterfly* (*c a*) (*c b*)›
  ⟨*proof*⟩

lemma *tc-butterfly-0-left*[*simp*]: ‹*tc-butterfly 0 t* = *0*›
  ⟨*proof*⟩

lemma *tc-butterfly-0-right*[*simp*]: ‹*tc-butterfly t 0* = *0*›
  ⟨*proof*⟩

11.5 More Hilbert-Schmidt

lemma *trace-class-hilbert-schmidt*: ‹*hilbert-schmidt a*› **if** ‹*trace-class a*›
  for *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
  ⟨*proof*⟩

lemma *finite-rank-hilbert-schmidt*: ‹*hilbert-schmidt a*› **if** ‹*finite-rank a*›
  for *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
  ⟨*proof*⟩

lemma *hilbert-schmidt-compact*: ‹*compact-op a*› **if** ‹*hilbert-schmidt a*›
  for *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
  — [1], Corollary 18.7. (Only the second part. The first part is stated inside this proof though.)
⟨*proof*⟩

lemma *trace-class-compact*: ‹*compact-op a*› **if** ‹*trace-class a*›
  for *a* :: ‹*'a::chilbert-space* $\Rightarrow_{CL}$ *'b::chilbert-space*›
  ⟨*proof*⟩

11.6 Spectral Theorem

The spectral theorem for trace class operators. A corollary of the one for compact operators (*Hilbert-Space-Tensor-Product.Spectral-Theorem*) but not an immediate one.

lift-definition *spectral-dec-proj-tc* :: ‹('a::chilbert-space, 'a) trace-class ⇒ nat ⇒ ('a, 'a) trace-class› **is**
  spectral-dec-proj
  ⟨*proof*⟩

lift-definition *spectral-dec-val-tc* :: ‹('a::chilbert-space, 'a) trace-class ⇒ nat ⇒ complex› **is**
  spectral-dec-val⟨*proof*⟩

lemma *spectral-dec-proj-tc-finite-rank*:
  assumes ‹*adj-tc a = a*›
  shows ‹*finite-rank-tc (spectral-dec-proj-tc a n)*›
  ⟨*proof*⟩

lemma *spectral-dec-summable-tc*:
  assumes ‹*selfadjoint-tc a*›
  shows ‹(λn. *spectral-dec-val-tc a n* $*_C$ *spectral-dec-proj-tc a n*) *abs-summable-on UNIV*›
⟨*proof*⟩

lemma *spectral-dec-has-sum-tc*:
  assumes ‹*selfadjoint-tc a*›
  shows ‹((λn. *spectral-dec-val-tc a n* $*_C$ *spectral-dec-proj-tc a n*) *has-sum a*) *UNIV*›
⟨*proof*⟩

lemma *spectral-dec-sums-tc*:
  assumes ‹*selfadjoint-tc a*›
  shows ‹(λn. *spectral-dec-val-tc a n* $*_C$ *spectral-dec-proj-tc a n*) *sums a*›
  ⟨*proof*⟩

lift-definition *spectral-dec-vecs-tc* :: ‹('a,'a) trace-class ⇒ 'a::chilbert-space set› **is**
  spectral-dec-vecs⟨*proof*⟩

lemma *compact-from-trace-class*[*iff*]: ‹*compact-op (from-trace-class t)*›
  ⟨*proof*⟩

lemma *sum-some-onb-of-tc-butterfly*:
  assumes ‹*finite-dim-ccsubspace S*›
  shows ‹($\sum$ *x*∈*some-onb-of S. tc-butterfly x x*) = *Abs-trace-class (Proj S)*›
  ⟨*proof*⟩

lemma *butterfly-spectral-dec-vec-tc-has-sum*:
  assumes ‹$t \geq 0$›
  shows ‹((λv. *tc-butterfly v v*) *has-sum t*) (*spectral-dec-vecs-tc t*)›
⟨*proof*⟩

lemma *spectral-dec-vec-tc-norm-summable*:
  assumes ‹$t \geq 0$›
  shows ‹(λv. (*norm v*)^2) *summable-on* (*spectral-dec-vecs-tc t*)›
⟨*proof*⟩

11.7 More Trace-Class

lemma *finite-rank-tc-dense-aux*: ‹*closure* (*Collect finite-rank-tc* :: (*'a*::*chilbert-space*, *'a*) *trace-class set*) = *UNIV*›
⟨*proof*⟩

lemma *finite-rank-tc-dense*: ‹*closure* (*Collect finite-rank-tc* :: (*'a*::*chilbert-space*, *'b*::*chilbert-space*) *trace-class set*) = *UNIV*›
⟨*proof*⟩

hide-fact *finite-rank-tc-dense-aux*

lemma *ccspan-finite-rank-tc*[*simp*]: ‹*ccspan* (*Collect finite-rank-tc*) = ⊤›
  ⟨*proof*⟩

lemma *ccspan-rank1-tc*[*simp*]: ‹*ccspan* (*Collect rank1-tc*) = ⊤›
  ⟨*proof*⟩

lemma *onb-butterflies-span-trace-class*:
  fixes *A* :: ‹*'a*::*chilbert-space set*› **and** *B* :: ‹*'b*::*chilbert-space set*›
  assumes ‹*is-onb A*› **and** ‹*is-onb B*›
  shows ‹*ccspan* ((λ(*x*, *y*). *tc-butterfly x y*) ‘ (*A*×*B*)) = ⊤›
⟨*proof*⟩

lemma *separating-set-tc-butterfly*: ‹*separating-set bounded-clinear* ((λ(*g*,*h*). *tc-butterfly g h*) ‘ (*UNIV* × *UNIV*))›
  ⟨*proof*⟩

lemma *separating-set-tc-butterfly-nested*:
  assumes ‹*separating-set* (*bounded-clinear* :: (- ⇒ *'c*::*complex-normed-vector*) ⇒ -) *A*›
  assumes ‹*separating-set* (*bounded-clinear* :: (- ⇒ *'c conjugate-space*) ⇒ -) *B*›
  shows ‹*separating-set* (*bounded-clinear* :: (- ⇒ *'c*) ⇒ -) ((λ(*g*,*h*). *tc-butterfly g h*) ‘ (*A* × *B*))›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

12 *Weak-Star-Topology* – Weak* topology on complex bounded operators

theory *Weak-Star-Topology*
  imports *Trace-Class Weak-Operator-Topology Misc-Tensor-Product-TTS*
begin

unbundle *cblinfun-syntax*

definition *weak-star-topology* :: ‹('a::chilbert-space $\Rightarrow_{CL}$'b::chilbert-space) topology›
  where ‹weak-star-topology = pullback-topology UNIV (λx. λt∈Collect trace-class. trace (t o_{CL} x))
(product-topology (λ-. euclidean) (Collect trace-class))›

lemma *open-map-product-topology-reindex*:
  fixes π :: ‹'b ⇒ 'a›
  assumes *bij-π*: ‹bij-betw π B A› **and** *ST*: ‹⋀x. x∈B ⟹ S x = T (π x)›
  assumes *g-def*: ‹⋀f. g f = restrict (f o π) B›
  shows ‹open-map (product-topology T A) (product-topology S B) g›
⟨*proof*⟩

lemma *homeomorphic-map-product-topology-reindex*:
  fixes π :: ‹'b ⇒ 'a›
  assumes *big-π*: ‹bij-betw π B A› **and** *ST*: ‹⋀x. x∈B ⟹ S x = T (π x)›
  assumes *g-def*: ‹⋀f. g f = restrict (f o π) B›
  shows ‹homeomorphic-map (product-topology T A) (product-topology S B) g›
⟨*proof*⟩

lemma *weak-star-topology-def'*:
  ‹weak-star-topology = pullback-topology UNIV (λx t. trace (from-trace-class t o_{CL} x)) euclidean›
⟨*proof*⟩

lemma *weak-star-topology-topspace*[*simp*]:
  topspace weak-star-topology = UNIV
  ⟨*proof*⟩

lemma *weak-star-topology-basis'*:
  fixes *f*::('a::chilbert-space $\Rightarrow_{CL}$ 'b::chilbert-space) **and** *U*::'i ⇒ *complex set* **and** *t*::'i ⇒ ('b,'a) *trace-class*
  assumes *finite I* ⋀*i. i* ∈ *I* ⟹ *open* (*U i*)
  shows *openin weak-star-topology* {*f*. ∀ *i*∈*I*. *trace* (*from-trace-class* (*t i*) o_{CL} *f*) ∈ *U i*}
⟨*proof*⟩

lemma *weak-star-topology-basis*:
  fixes *f*::('a::chilbert-space $\Rightarrow_{CL}$ 'b::chilbert-space) **and** *U*::'i ⇒ *complex set* **and** *t*::'i ⇒ ('b $\Rightarrow_{CL}$ 'a)
  assumes *finite I* ⋀*i. i* ∈ *I* ⟹ *open* (*U i*)
  assumes *tc*: ‹⋀i. i ∈ I ⟹ trace-class (t i)›

shows *openin weak-star-topology* $\{f.\ \forall\, i{\in}I.\ trace\ (t\ i\ o_{CL}\ f) \in U\ i\}$
⟨*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′*:
‹*openin cweak-operator-topology U* $\Longrightarrow$ *openin weak-star-topology U*›
⟨*proof*⟩

lemma *weak-star-topology-continuous-duality′*:
shows *continuous-map weak-star-topology euclidean* $(\lambda x.\ trace\ (from\text{-}trace\text{-}class\ t\ o_{CL}\ x))$
⟨*proof*⟩

lemma *weak-star-topology-continuous-duality*:
assumes ‹*trace-class t*›
shows *continuous-map weak-star-topology euclidean* $(\lambda x.\ trace\ (t\ o_{CL}\ x))$
⟨*proof*⟩

lemma *continuous-on-weak-star-topo-iff-coordinatewise*:
fixes *f* :: ‹′*a* ⇒ ′*b*::*chilbert-space* $\Rightarrow_{CL}$ ′*c*::*chilbert-space*›
shows *continuous-map T weak-star-topology f*
$\longleftrightarrow (\forall\, t.\ trace\text{-}class\ t \longrightarrow continuous\text{-}map\ T\ euclidean\ (\lambda x.\ trace\ (t\ o_{CL}\ f\ x)))$
⟨*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* = ‹*UNIV* :: (′*a*::*complex-normed-vector* $\Rightarrow_{CL}$ ′*b*::*complex-normed-vector*) *set*›
morphisms *from-weak-star to-weak-star* ⟨*proof*⟩
setup-lifting *type-definition-cblinfun-weak-star*

lift-definition *id-weak-star* :: ‹(′*a*::*complex-normed-vector*, ′*a*) *cblinfun-weak-star*› **is** *id-cblinfun*
⟨*proof*⟩

instantiation *cblinfun-weak-star* :: (*complex-normed-vector*, *complex-normed-vector*) *complex-vector*
begin
lift-definition *scaleC-cblinfun-weak-star* :: ‹*complex* ⇒ (′*a*, ′*b*) *cblinfun-weak-star* ⇒ (′*a*, ′*b*) *cblinfun-weak-star*›
is ‹*scaleC*› ⟨*proof*⟩
lift-definition *uminus-cblinfun-weak-star* :: ‹(′*a*, ′*b*) *cblinfun-weak-star* ⇒ (′*a*, ′*b*) *cblinfun-weak-star*› **is** *uminus* ⟨*proof*⟩
lift-definition *zero-cblinfun-weak-star* :: ‹(′*a*, ′*b*) *cblinfun-weak-star*› **is** *0* ⟨*proof*⟩
lift-definition *minus-cblinfun-weak-star* :: ‹(′*a*, ′*b*) *cblinfun-weak-star* ⇒ (′*a*, ′*b*) *cblinfun-weak-star* ⇒ (′*a*, ′*b*) *cblinfun-weak-star*› **is** *minus* ⟨*proof*⟩

lift-definition *plus-cblinfun-weak-star* :: ‹(′a, ′b) *cblinfun-weak-star* ⇒ (′a, ′b) *cblinfun-weak-star* ⇒ (′a, ′b) *cblinfun-weak-star*› **is** *plus* ⟨*proof*⟩
lift-definition *scaleR-cblinfun-weak-star* :: ‹*real* ⇒ (′a, ′b) *cblinfun-weak-star* ⇒ (′a, ′b) *cblinfun-weak-star*› **is** *scaleR* ⟨*proof*⟩
instance
⟨*proof*⟩
end

instantiation *cblinfun-weak-star* :: (*chilbert-space*, *chilbert-space*) *topological-space* **begin**
lift-definition *open-cblinfun-weak-star* :: ‹(′a, ′b) *cblinfun-weak-star set* ⇒ *bool*› **is** ‹*openin weak-star-topology*› ⟨*proof*⟩
instance
⟨*proof*⟩
end

lemma *transfer-nhds-weak-star-topology*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*cr-cblinfun-weak-star* ===> *rel-filter cr-cblinfun-weak-star*) (*nhdsin weak-star-topology*) *nhds*›
⟨*proof*⟩

lemma *limitin-weak-star-topology′*:
‹*limitin weak-star-topology f l F* ⟷ (∀ *t*. ((λ*j*. *trace* (*from-trace-class t* o_{CL} *f j*)) ⟶ *trace* (*from-trace-class t* o_{CL} *l*)) *F*)›
⟨*proof*⟩

lemma *limitin-weak-star-topology*:
‹*limitin weak-star-topology f l F* ⟷ (∀ *t*. *trace-class t* ⟶ ((λ*j*. *trace* (*t* o_{CL} *f j*)) ⟶ *trace* (*t* o_{CL} *l*)) *F*)›
⟨*proof*⟩

lemma *filterlim-weak-star-topology*:
‹*filterlim f* (*nhdsin weak-star-topology l*) = *limitin weak-star-topology f l*›
⟨*proof*⟩

lemma *openin-weak-star-topology′*: ‹*openin weak-star-topology U* ⟷ (∃ *V*. *open V* ∧ *U* = (λ*x t*. *trace* (*from-trace-class t* o_{CL} *x*)) −‘ *V*)›
⟨*proof*⟩

lemma *hausdorff-weak-star*[*simp*]: ‹*Hausdorff-space weak-star-topology*›
⟨*proof*⟩

lemma *Domainp-cr-cblinfun-weak-star*[*simp*]: ‹*Domainp cr-cblinfun-weak-star* = (λ-. *True*)›
⟨*proof*⟩

lemma *Rangep-cr-cblinfun-weak-star*[*simp*]: ‹*Rangep cr-cblinfun-weak-star* = (λ-. *True*)›
⟨*proof*⟩

lemma *transfer-euclidean-weak-star-topology*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-topology cr-cblinfun-weak-star*) *weak-star-topology euclidean*›
⟨*proof*⟩

instance *cblinfun-weak-star* :: (*chilbert-space*, *chilbert-space*) *t2-space*
⟨*proof*⟩

lemma *weak-star-topology-plus-cont*: ‹*LIM* (*x*,*y*) *nhdsin weak-star-topology a* $\times_F$ *nhdsin weak-star-topology b*.
x + *y* :> *nhdsin weak-star-topology* (*a* + *b*)›
⟨*proof*⟩

instance *cblinfun-weak-star* :: (*chilbert-space*, *chilbert-space*) *topological-group-add*
⟨*proof*⟩

lemma *continuous-map-left-comp-weak-star*:
‹*continuous-map weak-star-topology weak-star-topology* (λ*a*::'*a*::*chilbert-space* $\Rightarrow_{CL}$ -. *b* o_{CL} *a*)›

for *b* :: ‹'*b*::*chilbert-space* $\Rightarrow_{CL}$ '*c*::*chilbert-space*›
⟨*proof*⟩

lemma *continuous-map-right-comp-weak-star*:
‹*continuous-map weak-star-topology weak-star-topology* (λ*b*::'*b*::*chilbert-space* $\Rightarrow_{CL}$ -. *b* o_{CL} *a*)›

for *a* :: ‹'*a*::*chilbert-space* $\Rightarrow_{CL}$ '*b*::*chilbert-space*›
⟨*proof*⟩

lemma *continuous-map-scaleC-weak-star*: ‹*continuous-map weak-star-topology weak-star-topology* (*scaleC c*)›
⟨*proof*⟩

lemma *continuous-scaleC-weak-star*: ‹*continuous-on X* (*scaleC c* :: (-,-) *cblinfun-weak-star* ⇒ -)›
⟨*proof*⟩

lemma *weak-star-closure-is-csubspace*[*simp*]:
fixes *A*::('*a*::*chilbert-space*, '*b*::*chilbert-space*) *cblinfun-weak-star set*
assumes ‹*csubspace A*›
shows ‹*csubspace* (*closure A*)›
⟨*proof*⟩
include *lattice-syntax*

⟨*proof*⟩

lemma *transfer-csubspace-cblinfun-weak-star*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-weak-star* ===> (=)) *csubspace csubspace*›
⟨*proof*⟩

lemma *transfer-closed-cblinfun-weak-star*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-weak-star* ===> (=)) (*closedin weak-star-topology*) *closed*›
⟨*proof*⟩

lemma *transfer-closure-cblinfun-weak-star*[*transfer-rule*]:
includes *lifting-syntax*
shows ‹(*rel-set cr-cblinfun-weak-star* ===> *rel-set cr-cblinfun-weak-star*) (*Abstract-Topology.closure-of weak-star-topology*) *closure*›
⟨*proof*⟩

lemma *weak-star-closure-is-csubspace′*[*simp*]:
fixes *A*::(*′a*::*chilbert-space* $\Rightarrow_{CL}$ *′b*::*chilbert-space*) *set*
assumes ‹*csubspace A*›
shows ‹*csubspace* (*weak-star-topology closure-of A*)›
⟨*proof*⟩

lemma *has-sum-closed-weak-star-topology*:
assumes *aA*: ‹$\bigwedge$*i*. *a i* ∈ *A*›
assumes *closed*: ‹*closedin weak-star-topology A*›
assumes *subspace*: ‹*csubspace A*›
assumes *has-sum*: ‹$\bigwedge$*t*. *trace-class t* ⟹ ((λ*i*. *trace* (*t* o_{CL} *a i*)) *has-sum trace* (*t* o_{CL} *b*)) *I*›
shows ‹*b* ∈ *A*›
⟨*proof*⟩

lemma *has-sum-in-weak-star*:
‹*has-sum-in weak-star-topology f A l* ⟷
(∀ *t*. *trace-class t* ⟶ ((λ*i*. *trace* (*t* o_{CL} *f i*)) *has-sum trace* (*t* o_{CL} *l*)) *A*)›
⟨*proof*⟩

lemma *has-sum-butterfly-ket*: ‹*has-sum-in weak-star-topology* (λ*i*. *butterfly* (*ket i*) (*ket i*)) *UNIV id-cblinfun*›
⟨*proof*⟩

lemma *sandwich-weak-star-cont*[*simp*]:
‹*continuous-map weak-star-topology weak-star-topology* (*sandwich A*)›
⟨*proof*⟩

lemma *has-sum-butterfly-ket-a*: ‹*has-sum-in weak-star-topology* (λ*i*. *butterfly* (*a* $*_V$ *ket i*) (*ket i*)) *UNIV a*›
⟨*proof*⟩

lemma *finite-rank-weak-star-dense*[*simp*]: ‹*weak-star-topology closure-of* (*Collect finite-rank*) = (*UNIV* :: (*'a ell2* $\Rightarrow_{CL}$ *'b::chilbert-space*) *set*)›
⟨*proof*⟩

lemma *butterkets-weak-star-dense*[*simp*]:
‹*weak-star-topology closure-of cspan* ((λ(ξ,η). *butterfly* (*ket* ξ) (*ket* η)) ‘ *UNIV*) = *UNIV*›

⟨*proof*⟩

lemma *weak-star-clinear-eq-butterfly-ketI*:
fixes *F G* :: ‹(*'a ell2* $\Rightarrow_{CL}$ *'b ell2*) ⇒ *'c::complex-vector*›
assumes *clinear F* **and** *clinear G*
and ‹*continuous-map weak-star-topology T F*› **and** ‹*continuous-map weak-star-topology T G*›
and ‹*Hausdorff-space T*›
assumes ⋀*i j*. *F* (*butterfly* (*ket i*) (*ket j*)) = *G* (*butterfly* (*ket i*) (*ket j*))
shows *F* = *G*
⟨*proof*⟩

lemma *continuous-map-scaleC-weak-star'*[*continuous-intros*]:
assumes ‹*continuous-map T weak-star-topology f*›
shows ‹*continuous-map T weak-star-topology* (λ*x*. *scaleC c* (*f x*))›
⟨*proof*⟩

lemma *continuous-map-uminus-weak-star*[*continuous-intros*]:
assumes ‹*continuous-map T weak-star-topology f*›
shows ‹*continuous-map T weak-star-topology* (λ*x*. − *f x*)›
⟨*proof*⟩

lemma *continuous-map-add-weak-star*[*continuous-intros*]:
assumes ‹*continuous-map T weak-star-topology f*›
assumes ‹*continuous-map T weak-star-topology g*›
shows ‹*continuous-map T weak-star-topology* (λ*x*. *f x* + *g x*)›
⟨*proof*⟩

lemma *continuous-map-minus-weak-star*[*continuous-intros*]:
assumes ‹*continuous-map T weak-star-topology f*›
assumes ‹*continuous-map T weak-star-topology g*›
shows ‹*continuous-map T weak-star-topology* (λ*x*. *f x* − *g x*)›
⟨*proof*⟩

lemma *weak-star-topology-is-norm-topology-fin-dim*[*simp*]:
‹(*weak-star-topology* :: (*'a*::{*cfinite-dim*,*chilbert-space*} $\Rightarrow_{CL}$ *'b*::{*cfinite-dim*,*chilbert-space*}) *topology*) = *euclidean*›
⟨*proof*⟩

lemma *infsum-mono-wot*:
fixes f :: $'a \Rightarrow ('b$::*chilbert-space* $\Rightarrow_{CL}$ $'b)$
assumes *summable-on-in cweak-operator-topology f A* **and** *summable-on-in cweak-operator-topology g A*
assumes ‹$\bigwedge x.\ x \in A \Longrightarrow f\ x \le g\ x$›
shows *infsum-in cweak-operator-topology f A* $\le$ *infsum-in cweak-operator-topology g A*
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

13 *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*

13.1 Tensor product on - *ell2*

lift-definition *tensor-ell2* :: ‹$'a$ *ell2* $\Rightarrow$ $'b$ *ell2* $\Rightarrow$ $('a \times 'b)$ *ell2*› (**infixr** $\otimes_s$ *70*) **is**
‹$\lambda\psi\ \varphi\ (i,j).\ \psi\ i * \varphi\ j$›
⟨*proof*⟩

lemma *tensor-ell2-add1*: ‹*tensor-ell2* $(a + b)$ c = *tensor-ell2* a c + *tensor-ell2* b c›
⟨*proof*⟩

lemma *tensor-ell2-add2*: ‹*tensor-ell2* a $(b + c)$ = *tensor-ell2* a b + *tensor-ell2* a c›
⟨*proof*⟩

lemma *tensor-ell2-scaleC1*: ‹*tensor-ell2* $(c *_C a)$ b = $c *_C$ *tensor-ell2* a b›
⟨*proof*⟩

lemma *tensor-ell2-scaleC2*: ‹*tensor-ell2* a $(c *_C b)$ = $c *_C$ *tensor-ell2* a b›
⟨*proof*⟩

lemma *tensor-ell2-diff1*: ‹*tensor-ell2* $(a - b)$ c = *tensor-ell2* a c − *tensor-ell2* b c›

⟨proof⟩

lemma *tensor-ell2-diff2*: ‹*tensor-ell2 a* (*b* − *c*) = *tensor-ell2 a b* − *tensor-ell2 a c*›
⟨proof⟩

lemma *tensor-ell2-inner-prod*[*simp*]: ‹*tensor-ell2 a b* $\cdot_C$ *tensor-ell2 c d* = (*a* $\cdot_C$ *c*) ∗ (*b* $\cdot_C$ *d*)›
⟨proof⟩

lemma *norm-tensor-ell2*: ‹*norm* (*a* $\otimes_s$ *b*) = *norm a* ∗ *norm b*›
⟨proof⟩

lemma *clinear-tensor-ell21*: *clinear* (λ*b*. *a* $\otimes_s$ *b*)
⟨proof⟩

lemma *bounded-clinear-tensor-ell21*: *bounded-clinear* (λ*b*. *a* $\otimes_s$ *b*)
⟨proof⟩

lemma *clinear-tensor-ell22*: *clinear* (λ*a*. *a* $\otimes_s$ *b*)
⟨proof⟩

lemma *bounded-clinear-tensor-ell22*: *bounded-clinear* (λ*a*. *tensor-ell2 a b*)
⟨proof⟩

lemma *tensor-ell2-ket*: *tensor-ell2* (*ket i*) (*ket j*) = *ket* (*i*,*j*)
⟨proof⟩

lemma *tensor-ell2-0-left*[*simp*]: ‹*0* $\otimes_s$ *x* = *0*›
⟨proof⟩

lemma *tensor-ell2-0-right*[*simp*]: ‹*x* $\otimes_s$ *0* = *0*›
⟨proof⟩

lemma *tensor-ell2-sum-left*: ‹(∑*x*∈*X*. *a x*) $\otimes_s$ *b* = (∑*x*∈*X*. *a x* $\otimes_s$ *b*)›
⟨proof⟩

lemma *tensor-ell2-sum-right*: ‹*a* $\otimes_s$ (∑*x*∈*X*. *b x*) = (∑*x*∈*X*. *a* $\otimes_s$ *b x*)›
⟨proof⟩

lemma *tensor-ell2-dense*:
fixes *S* :: ‹'*a ell2 set*› **and** *T* :: ‹'*b ell2 set*›
assumes ‹*closure* (*cspan S*) = *UNIV*› **and** ‹*closure* (*cspan T*) = *UNIV*›
shows ‹*closure* (*cspan* {*a*$\otimes_s$*b* | *a b*. *a*∈*S* ∧ *b*∈*T*}) = *UNIV*›
⟨proof⟩

definition *assoc-ell2* :: ‹(('*a*×'*b*)×'*c*) *ell2* $\Rightarrow_{CL}$ ('*a*×('*b*×'*c*)) *ell2*› **where**
‹*assoc-ell2* = *classical-operator* (*Some o* (λ((*a*,*b*),*c*). (*a*,(*b*,*c*))))›

lemma *unitary-assoc-ell2*[*simp*]: ‹*unitary assoc-ell2*›
⟨proof⟩

lemma *assoc-ell2-tensor*: ‹$assoc\text{-}ell2 *_V ((a \otimes_s b) \otimes_s c) = (a \otimes_s (b \otimes_s c))$›
⟨*proof*⟩

lemma *assoc-ell2'-tensor*: ‹$assoc\text{-}ell2* *_V tensor\text{-}ell2\ a\ (tensor\text{-}ell2\ b\ c) = tensor\text{-}ell2\ (tensor\text{-}ell2\ a\ b)\ c$›
⟨*proof*⟩

lemma *assoc-ell2'-inv*: $assoc\text{-}ell2\ o_{CL}\ assoc\text{-}ell2* = id\text{-}cblinfun$
⟨*proof*⟩

lemma *assoc-ell2-inv*: $assoc\text{-}ell2*\ o_{CL}\ assoc\text{-}ell2 = id\text{-}cblinfun$
⟨*proof*⟩

definition *swap-ell2* :: ‹$('a\times'b)\ ell2 \Rightarrow_{CL} ('b\times'a)\ ell2$› **where**
‹$swap\text{-}ell2 = classical\text{-}operator\ (Some\ o\ prod.swap)$›

lemma *unitary-swap-ell2*[*simp*]: ‹*unitary swap-ell2*›
⟨*proof*⟩

lemma *swap-ell2-tensor*[*simp*]: ‹$swap\text{-}ell2 *_V (a \otimes_s b) = b \otimes_s a$› **for** *a* :: ‹*'a ell2*› **and** *b* :: ‹*'b ell2*›
⟨*proof*⟩

lemma *swap-ell2-ket*[*simp*]: ‹$(swap\text{-}ell2 :: ('a\times'b)\ ell2 \Rightarrow_{CL} \text{-}) *_V ket\ (x,y) = ket\ (y,x)$›
⟨*proof*⟩

lemma *adjoint-swap-ell2*[*simp*]: ‹$swap\text{-}ell2* = swap\text{-}ell2$›
⟨*proof*⟩

lemma *tensor-ell2-extensionality*:
assumes $(\bigwedge s\ t.\ a *_V (s \otimes_s t) = b *_V (s \otimes_s t))$
shows $a = b$
⟨*proof*⟩

lemma *tensor-ell2-nonzero*: ‹$a \otimes_s b \neq 0$› **if** ‹$a \neq 0$› **and** ‹$b \neq 0$›
⟨*proof*⟩

lemma *swap-ell2-selfinv*[*simp*]: ‹$swap\text{-}ell2\ o_{CL}\ swap\text{-}ell2 = id\text{-}cblinfun$›
⟨*proof*⟩

lemma *bounded-cbilinear-tensor-ell2*[*bounded-cbilinear*]: ‹$bounded\text{-}cbilinear\ (\otimes_s)$›
⟨*proof*⟩

lemma *ket-pair-split*: ‹$ket\ x = tensor\text{-}ell2\ (ket\ (fst\ x))\ (ket\ (snd\ x))$›
⟨*proof*⟩

lemma *tensor-ell2-is-ortho-set*:
  assumes ‹*is-ortho-set A*› ‹*is-ortho-set B*›
  shows ‹*is-ortho-set* $\{a \otimes_s b \mid a\ b.\ a \in A \wedge b \in B\}$›
  ⟨*proof*⟩

lemma *tensor-ell2-dense′*: ‹*ccspan* $\{a \otimes_s b \mid a\ b.\ a \in A \wedge b \in B\} = \top$› **if** ‹*ccspan* $A = \top$› **and** ‹*ccspan* $B = \top$›
⟨*proof*⟩

lemma *tensor-ell2-is-onb*:
  assumes ‹*is-onb A*› ‹*is-onb B*›
  shows ‹*is-onb* $\{a \otimes_s b \mid a\ b.\ a \in A \wedge b \in B\}$›
⟨*proof*⟩

lemma *continuous-tensor-ell2*: ‹*continuous-on UNIV* $(\lambda(x{::}'a\ ell2,\ y{::}'b\ ell2).\ x \otimes_s y)$›
⟨*proof*⟩

lemma *summable-on-tensor-ell2-right*: ‹φ *summable-on* $A \Longrightarrow (\lambda x.\ \psi \otimes_s \varphi\ x)$ *summable-on* A›
  ⟨*proof*⟩

lemma *summable-on-tensor-ell2-left*: ‹φ *summable-on* $A \Longrightarrow (\lambda x.\ \varphi\ x \otimes_s \psi)$ *summable-on* A›
  ⟨*proof*⟩

lift-definition *tensor-ell2-left* :: ‹$'a\ ell2 \Rightarrow ('b\ ell2 \Rightarrow_{CL} ('a \times 'b)\ ell2)$› **is**
  ‹$\lambda\psi\ \varphi.\ \psi \otimes_s \varphi$›
  ⟨*proof*⟩

lemma *tensor-ell2-left-apply*[*simp*]: ‹*tensor-ell2-left* $\psi *_V \varphi = \psi \otimes_s \varphi$›
  ⟨*proof*⟩

lift-definition *tensor-ell2-right* :: ‹$'a\ ell2 \Rightarrow ('b\ ell2 \Rightarrow_{CL} ('b \times 'a)\ ell2)$› **is**
  ‹$\lambda\psi\ \varphi.\ \varphi \otimes_s \psi$›
  ⟨*proof*⟩

lemma *tensor-ell2-right-apply*[*simp*]: ‹*tensor-ell2-right* $\psi *_V \varphi = \varphi \otimes_s \psi$›
  ⟨*proof*⟩

lemma *isometry-tensor-ell2-right*: ‹*isometry* (*tensor-ell2-right* ψ)› **if** ‹*norm* $\psi = 1$›
  ⟨*proof*⟩

lemma *isometry-tensor-ell2-left*: ‹*isometry* (*tensor-ell2-left* ψ)› **if** ‹*norm* $\psi = 1$›
  ⟨*proof*⟩

lemma *tensor-ell2-right-scale*: ‹*tensor-ell2-right* $(a *_C \psi) = a *_C$ *tensor-ell2-right* ψ›
  ⟨*proof*⟩
lemma *tensor-ell2-left-scale*: ‹*tensor-ell2-left* $(a *_C \psi) = a *_C$ *tensor-ell2-left* ψ›
  ⟨*proof*⟩

lemma *tensor-ell2-right-0*[*simp*]: ‹*tensor-ell2-right 0 = 0*›
⟨*proof*⟩

lemma *tensor-ell2-left-0*[*simp*]: ‹*tensor-ell2-left 0 = 0*›
⟨*proof*⟩

lemma *tensor-ell2-right-adj-apply*[*simp*]: ‹$(\textit{tensor-ell2-right}\ \psi*)\ *_V\ (\alpha \otimes_s \beta) = (\psi \bullet_C \beta) *_C \alpha$›
⟨*proof*⟩
lemma *tensor-ell2-left-adj-apply*[*simp*]: ‹$(\textit{tensor-ell2-left}\ \psi*)\ *_V\ (\alpha \otimes_s \beta) = (\psi \bullet_C \alpha) *_C \beta$›
⟨*proof*⟩

lemma *infsum-tensor-ell2-right*: ‹$\psi \otimes_s (\sum_\infty x{\in}A.\ \varphi\ x) = (\sum_\infty x{\in}A.\ \psi \otimes_s \varphi\ x)$›
⟨*proof*⟩

lemma *infsum-tensor-ell2-left*: ‹$(\sum_\infty x{\in}A.\ \varphi\ x) \otimes_s \psi = (\sum_\infty x{\in}A.\ \varphi\ x \otimes_s \psi)$›
⟨*proof*⟩

lemma *tensor-ell2-extensionality3*:
assumes $(\bigwedge s\ t\ u.\ a *_V (s \otimes_s t \otimes_s u) = b *_V (s \otimes_s t \otimes_s u))$
shows $a = b$
⟨*proof*⟩

lemma *cblinfun-cinner-tensor-eqI*:
assumes ‹$\bigwedge\psi\ \varphi.\ (\psi \otimes_s \varphi) \bullet_C (A *_V (\psi \otimes_s \varphi)) = (\psi \otimes_s \varphi) \bullet_C (B *_V (\psi \otimes_s \varphi))$›
shows ‹$A = B$›
⟨*proof*⟩

lemma *unitary-tensor-ell2-right-CARD-1*:
fixes ψ :: ‹*'a* :: {*CARD-1*,*enum*} *ell2*›
assumes ‹*norm* ψ = *1*›
shows ‹*unitary* (*tensor-ell2-right* ψ)›
⟨*proof*⟩

13.2 Tensor product of operators on - *ell2*

definition *tensor-op* :: ‹(*'a ell2*, *'b ell2*) *cblinfun* ⇒ (*'c ell2*, *'d ell2*) *cblinfun*
⇒ ((*'a*×*'c*) *ell2*, (*'b*×*'d*) *ell2*) *cblinfun*› (**infixr** $\otimes_o$ *70*) **where**
‹*tensor-op M N* = *cblinfun-extension* (*range ket*) (λ*k*. *case* (*inv ket k*) *of* (*x*,*y*) ⇒ *tensor-ell2* ($M *_V$ *ket x*) ($N *_V$ *ket y*))›

lemma
— Loosely following [7, Section IV.1]
fixes *a* :: ‹*'a*› **and** *b* :: ‹*'b*› **and** *c* :: ‹*'c*› **and** *d* :: ‹*'d*› **and** *M* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'b ell2*› **and** *N* :: ‹*'c ell2* $\Rightarrow_{CL}$ *'d ell2*›
shows *tensor-op-ell2*: ‹$(M \otimes_o N) *_V (\psi \otimes_s \varphi) = (M *_V \psi) \otimes_s (N *_V \varphi)$›
and *tensor-op-norm*: ‹*norm* $(M \otimes_o N)$ = *norm M* ∗ *norm N*›
⟨*proof*⟩

lemma *tensor-op-ket*: ‹*tensor-op M N* $*_V$ (*ket* (*a*,*c*)) = *tensor-ell2* (*M* $*_V$ *ket a*) (*N* $*_V$ *ket c*)›
⟨*proof*⟩

lemma *comp-tensor-op*: (*tensor-op a b*) o_{CL} (*tensor-op c d*) = *tensor-op* (*a* o_{CL} *c*) (*b* o_{CL} *d*)
for *a* :: *'e ell2* $\Rightarrow_{CL}$ *'c ell2* **and** *b* :: *'f ell2* $\Rightarrow_{CL}$ *'d ell2* **and**
c :: *'a ell2* $\Rightarrow_{CL}$ *'e ell2* **and** *d* :: *'b ell2* $\Rightarrow_{CL}$ *'f ell2*
⟨*proof*⟩

lemma *tensor-op-left-add*: ‹(x + y) $\otimes_o$ b = x $\otimes_o$ b + y $\otimes_o$ b›
for *x y* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'c ell2*› **and** *b* :: ‹*'b ell2* $\Rightarrow_{CL}$ *'d ell2*›
⟨*proof*⟩

lemma *tensor-op-right-add*: ‹b $\otimes_o$ (x + y) = b $\otimes_o$ x + b $\otimes_o$ y›
for *x y* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'c ell2*› **and** *b* :: ‹*'b ell2* $\Rightarrow_{CL}$ *'d ell2*›
⟨*proof*⟩

lemma *tensor-op-scaleC-left*: ‹(c $*_C$ x) $\otimes_o$ b = c $*_C$ (x $\otimes_o$ b)›
for *x* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'c ell2*› **and** *b* :: ‹*'b ell2* $\Rightarrow_{CL}$ *'d ell2*›
⟨*proof*⟩

lemma *tensor-op-scaleC-right*: ‹b $\otimes_o$ (c $*_C$ x) = c $*_C$ (b $\otimes_o$ x)›
for *x* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'c ell2*› **and** *b* :: ‹*'b ell2* $\Rightarrow_{CL}$ *'d ell2*›
⟨*proof*⟩

lemma *tensor-op-bounded-cbilinear*[*simp*]: ‹*bounded-cbilinear tensor-op*›
⟨*proof*⟩

lemma *tensor-op-cbilinear*[*simp*]: ‹*cbilinear tensor-op*›
⟨*proof*⟩

lemma *tensor-butter*: ‹*butterfly* (*ket i*) (*ket j*) $\otimes_o$ *butterfly* (*ket k*) (*ket l*) = *butterfly* (*ket* (*i*,*k*)) (*ket* (*j*,*l*))›
⟨*proof*⟩

lemma *cspan-tensor-op-butter*: ‹*cspan* {*tensor-op* (*butterfly* (*ket i*) (*ket j*)) (*butterfly* (*ket k*) (*ket l*))| (*i*::-::*finite*) (*j*::-::*finite*) (*k*::-::*finite*) (*l*::-::*finite*). *True*} = *UNIV*›
⟨*proof*⟩

lemma *cindependent-tensor-op-butter*: ‹*cindependent* {*tensor-op* (*butterfly* (*ket i*) (*ket j*)) (*butterfly* (*ket k*) (*ket l*))| *i j k l*. *True*}›
⟨*proof*⟩

lift-definition *right-amplification* :: ‹(*'a ell2* $\Rightarrow_{CL}$ *'b ell2*) $\Rightarrow_{CL}$ ((*'a*×*'c*) *ell2* $\Rightarrow_{CL}$ (*'b*×*'c*) *ell2*)› **is**
‹λ*a*. *a* $\otimes_o$ *id-cblinfun*›
⟨*proof*⟩

lift-definition *left-amplification* :: ‹(*'a ell2* $\Rightarrow_{CL}$ *'b ell2*) $\Rightarrow_{CL}$ ((*'c*×*'a*) *ell2* $\Rightarrow_{CL}$ (*'c*×*'b*) *ell2*)› **is**

‹$\lambda a.$ *id-cblinfun* $\otimes_o$ *a*›
⟨*proof*⟩

lemma *sandwich-tensor-ell2-right*: ‹*sandwich* (*tensor-ell2-right* ψ∗) $*_V$ *a* $\otimes_o$ *b* = (ψ $\cdot_C$ (*b* $*_V$ ψ)) $*_C$ *a*›
⟨*proof*⟩
lemma *sandwich-tensor-ell2-left*: ‹*sandwich* (*tensor-ell2-left* ψ∗) $*_V$ *a* $\otimes_o$ *b* = (ψ $\cdot_C$ (*a* $*_V$ ψ)) $*_C$ *b*›
⟨*proof*⟩

lemma *tensor-op-adjoint*: ‹(*tensor-op a b*)∗ = *tensor-op* (*a*∗) (*b*∗)›
⟨*proof*⟩

lemma *has-sum-id-tensor-butterfly-ket*: ‹(($\lambda i.$ (*id-cblinfun* $\otimes_o$ *butterfly* (*ket i*) (*ket i*)) $*_V$ ψ) *has-sum* ψ) *UNIV*›
⟨*proof*⟩

lemma *tensor-op-dense*: ‹*cstrong-operator-topology closure-of* (*cspan* {*a* $\otimes_o$ *b* | *a b. True*}) = *UNIV*›
— [7, p.185 (10)], but we prove it directly.
⟨*proof*⟩

lemma *tensor-extensionality-finite*:
fixes *F G* :: ‹((('*a*::*finite* × '*b*::*finite*) *ell2*) $\Rightarrow_{CL}$ (('*c*::*finite* × '*d*::*finite*) *ell2*)) ⇒ '*e*::*complex-vector*›
assumes [*simp*]: *clinear F clinear G*
assumes *tensor-eq*: (⋀*a b. F* (*tensor-op a b*) = *G* (*tensor-op a b*))
shows *F* = *G*
⟨*proof*⟩

lemma *tensor-id*[*simp*]: ‹*tensor-op id-cblinfun id-cblinfun* = *id-cblinfun*›
⟨*proof*⟩

lemma *tensor-butterfly*: *tensor-op* (*butterfly* ψ ψ') (*butterfly* φ φ') = *butterfly* (*tensor-ell2* ψ φ) (*tensor-ell2* ψ' φ')
⟨*proof*⟩

definition *tensor-lift* :: ‹(('*a1*::*finite ell2* $\Rightarrow_{CL}$ '*a2*::*finite ell2*) ⇒ ('*b1*::*finite ell2* $\Rightarrow_{CL}$ '*b2*::*finite ell2*) ⇒ '*c*)
⇒ ((('*a1*×'*b1*) *ell2* $\Rightarrow_{CL}$ ('*a2*×'*b2*) *ell2*) ⇒ '*c*::*complex-normed-vector*)›
where
tensor-lift F2 = (*SOME G. clinear G* ∧ (∀ *a b. G* (*tensor-op a b*) = *F2 a b*))

lemma
fixes *F2* :: '*a*::*finite ell2* $\Rightarrow_{CL}$ '*b*::*finite ell2*
⇒ '*c*::*finite ell2* $\Rightarrow_{CL}$ '*d*::*finite ell2*

$\Rightarrow$ *'e::complex-normed-vector*
assumes *cbilinear F2*
shows *tensor-lift-clinear*: *clinear* (*tensor-lift F2*)
and *tensor-lift-correct*: ‹(λ*a b. tensor-lift F2* (*a* $\otimes_o$ *b*)) = *F2*›
⟨*proof*⟩

lemma *tensor-op-nonzero*:
fixes *a* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'c ell2*› **and** *b* :: ‹*'b ell2* $\Rightarrow_{CL}$ *'d ell2*›
assumes ‹*a* $\neq$ *0*› **and** ‹*b* $\neq$ *0*›
shows ‹*a* $\otimes_o$ *b* $\neq$ *0*›
⟨*proof*⟩

lemma *inj-tensor-ell2-left*: ‹*inj* (λ*a::'a ell2. a* $\otimes_s$ *b*)› **if** ‹*b* $\neq$ *0*› **for** *b* :: ‹*'b ell2*›
⟨*proof*⟩

lemma *inj-tensor-ell2-right*: ‹*inj* (λ*b::'b ell2. a* $\otimes_s$ *b*)› **if** ‹*a* $\neq$ *0*› **for** *a* :: ‹*'a ell2*›
⟨*proof*⟩

lemma *inj-tensor-left*: ‹*inj* (λ*a::'a ell2* $\Rightarrow_{CL}$ *'c ell2. a* $\otimes_o$ *b*)› **if** ‹*b* $\neq$ *0*› **for** *b* :: ‹*'b ell2* $\Rightarrow_{CL}$ *'d ell2*›
⟨*proof*⟩

lemma *inj-tensor-right*: ‹*inj* (λ*b::'b ell2* $\Rightarrow_{CL}$ *'c ell2. a* $\otimes_o$ *b*)› **if** ‹*a* $\neq$ *0*› **for** *a* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'d ell2*›
⟨*proof*⟩

lemma *tensor-ell2-almost-injective*:
assumes ‹*tensor-ell2 a b* = *tensor-ell2 c d*›
assumes ‹*a* $\neq$ *0*›
shows ‹$\exists\gamma$. *b* = $\gamma *_C$ *d*›
⟨*proof*⟩

lemma *tensor-op-almost-injective*:
fixes *a c* :: ‹*'a ell2* $\Rightarrow_{CL}$ *'b ell2*›
and *b d* :: ‹*'c ell2* $\Rightarrow_{CL}$ *'d ell2*›
assumes ‹*tensor-op a b* = *tensor-op c d*›
assumes ‹*a* $\neq$ *0*›
shows ‹$\exists\gamma$. *b* = $\gamma *_C$ *d*›
⟨*proof*⟩

lemma *clinear-tensor-left*[*simp*]: ‹*clinear* (λ*a. a* $\otimes_o$ *b* :: *- ell2* $\Rightarrow_{CL}$ *- ell2*)›
⟨*proof*⟩

lemma *clinear-tensor-right*[*simp*]: ‹*clinear* (λ*b. a* $\otimes_o$ *b* :: *- ell2* $\Rightarrow_{CL}$ *- ell2*)›
⟨*proof*⟩

lemma *tensor-op-0-left*[*simp*]: ‹*tensor-op 0 x* = (*0* :: (*'a*∗*'b*) *ell2* $\Rightarrow_{CL}$ (*'c*∗*'d*) *ell2*)›

⟨proof⟩

lemma *tensor-op-0-right*[*simp*]: ‹*tensor-op x 0* = (*0* :: ('*a*∗'*b*) *ell2* $\Rightarrow_{CL}$ ('*c*∗'*d*) *ell2*)›
⟨proof⟩

lemma *bij-tensor-ell2-one-dim-left*:
assumes ‹$\psi \neq 0$›
shows ‹*bij* (λ*x*::'*b ell2*. (ψ :: '*a*::*CARD-1 ell2*) $\otimes_s$ *x*)›
⟨proof⟩

lemma *bij-tensor-op-one-dim-left*:
fixes *a* :: ‹'*a*::{*CARD-1*,*enum*} *ell2* $\Rightarrow_{CL}$ '*b*::{*CARD-1*,*enum*} *ell2*›
assumes ‹*a* ≠ *0*›
shows ‹*bij* (λ*x*::'*c ell2* $\Rightarrow_{CL}$ '*d ell2*. *a* $\otimes_o$ *x*)›
⟨proof⟩

lemma *bij-tensor-op-one-dim-right*:
assumes ‹*b* ≠ *0*›
shows ‹*bij* (λ*x*::'*c ell2* $\Rightarrow_{CL}$ '*d ell2*. *x* $\otimes_o$ (*b* :: '*a*::{*CARD-1*,*enum*} *ell2* $\Rightarrow_{CL}$ '*b*::{*CARD-1*,*enum*} *ell2*))›
(**is** ‹*bij ?f*›)
⟨proof⟩

lemma *overlapping-tensor*:
fixes *a23* :: ‹('*a2*∗'*a3*) *ell2* $\Rightarrow_{CL}$ ('*b2*∗'*b3*) *ell2*›
and *b12* :: ‹('*a1*∗'*a2*) *ell2* $\Rightarrow_{CL}$ ('*b1*∗'*b2*) *ell2*›
assumes *eq*: ‹*butterfly* ψ ψ' $\otimes_o$ *a23* = *assoc-ell2* o_{CL} (*b12* $\otimes_o$ *butterfly* φ φ') o_{CL} *assoc-ell2*∗›
assumes ‹$\psi \neq 0$› ‹$\psi' \neq 0$› ‹$\varphi \neq 0$› ‹$\varphi' \neq 0$›
shows ‹∃ *c*. *butterfly* ψ ψ' $\otimes_o$ *a23* = *butterfly* ψ ψ' $\otimes_o$ *c* $\otimes_o$ *butterfly* φ φ'›
⟨proof⟩

lemma *tensor-op-pos*: ‹*a* $\otimes_o$ *b* ≥ *0*› **if** [*simp*]: ‹*a* ≥ *0*› ‹*b* ≥ *0*›
for *a* :: ‹'*a ell2* $\Rightarrow_{CL}$ '*a ell2*› **and** *b* :: ‹'*b ell2* $\Rightarrow_{CL}$ '*b ell2*›
— [8, Lemma 18]
⟨proof⟩

lemma *abs-op-tensor*: ‹*abs-op* (*a* $\otimes_o$ *b*) = *abs-op a* $\otimes_o$ *abs-op b*›
— [8, Lemma 18]
⟨proof⟩

lemma *trace-class-tensor*: ‹*trace-class* (*a* $\otimes_o$ *b*)› **if** ‹*trace-class a*› **and** ‹*trace-class b*›
— [8, Lemma 32]
⟨proof⟩

lemma *swap-tensor-op*[*simp*]: ‹*swap-ell2* o_{CL} (*a* $\otimes_o$ *b*) o_{CL} *swap-ell2* = *b* $\otimes_o$ *a*›
⟨proof⟩

lemma *swap-tensor-op-sandwich*[*simp*]: ‹*sandwich swap-ell2* (*a* $\otimes_o$ *b*) = *b* $\otimes_o$ *a*›

⟨*proof*⟩

lemma *swap-ell2-commute-tensor-op*:
‹*swap-ell2* o_{CL} (*a* $\otimes_o$ *b*) = (*b* $\otimes_o$ *a*) o_{CL} *swap-ell2*›
⟨*proof*⟩

lemma *trace-class-tensor-op-swap*: ‹*trace-class* (*a* $\otimes_o$ *b*) ⟷ *trace-class* (*b* $\otimes_o$ *a*)›
⟨*proof*⟩

lemma *trace-class-tensor-iff*: ‹*trace-class* (*a* $\otimes_o$ *b*) ⟷ (*trace-class a* ∧ *trace-class b*) ∨ *a* = *0* ∨ *b* = *0*›
⟨*proof*⟩

lemma *trace-tensor*: ‹*trace* (*a* $\otimes_o$ *b*) = *trace a* ∗ *trace b*›
— [8, Lemma 32]
⟨*proof*⟩

lemma *isometry-tensor-op*: ‹*isometry* (*U* $\otimes_o$ *V*)› **if** ‹*isometry U*› **and** ‹*isometry V*›
⟨*proof*⟩

lemma *is-Proj-tensor-op*: ‹*is-Proj a* ⟹ *is-Proj b* ⟹ *is-Proj* (*a* $\otimes_o$ *b*)›
⟨*proof*⟩

lemma *isometry-tensor-id-right*[*simp*]:
fixes *U* :: ‹'*a ell2* $\Rightarrow_{CL}$ '*b ell2*›
shows ‹*isometry* (*U* $\otimes_o$ (*id-cblinfun* :: '*c ell2* $\Rightarrow_{CL}$ -)) ⟷ *isometry U*›
⟨*proof*⟩

lemma *isometry-tensor-id-left*[*simp*]:
fixes *U* :: ‹'*a ell2* $\Rightarrow_{CL}$ '*b ell2*›
shows ‹*isometry* ((*id-cblinfun* :: '*c ell2* $\Rightarrow_{CL}$ -) $\otimes_o$ *U*) ⟷ *isometry U*›
⟨*proof*⟩

lemma *unitary-tensor-id-right*[*simp*]: ‹*unitary* (*U* $\otimes_o$ *id-cblinfun*) ⟷ *unitary U*›
⟨*proof*⟩

lemma *unitary-tensor-id-left*[*simp*]: ‹*unitary* (*id-cblinfun* $\otimes_o$ *U*) ⟷ *unitary U*›
⟨*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* $\varphi*$ = *id-cblinfun* $\otimes_o$ *butterfly* ψ φ›
⟨*proof*⟩
lemma *tensor-ell2-left-butterfly*: ‹*tensor-ell2-left* ψ o_{CL} *tensor-ell2-left* $\varphi*$ = *butterfly* ψ φ $\otimes_o$ *id-cblinfun*›
⟨*proof*⟩

lift-definition *tc-tensor* :: ‹('*a ell2*, '*b ell2*) *trace-class* ⇒ ('*c ell2*, '*d ell2*) *trace-class* ⇒ (('*a* × '*c*) *ell2*, ('*b* × '*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*: ‹*tc-tensor a b* ≥ *0*› **if** ‹*a* ≥ *0*› **and** ‹*b* ≥ *0*›
for *a* :: ‹('*a ell2*,'*a ell2*) *trace-class*› **and** *b* :: ‹('*b ell2*,'*b ell2*) *trace-class*›
⟨*proof*⟩

interpretation *tensor-op-cbilinear*: *bounded-cbilinear tensor-op*
⟨*proof*⟩

lemma *tensor-op-mono-left*:
fixes *a* :: ‹'*a ell2* $\Rightarrow_{CL}$ '*a ell2*› **and** *c* :: ‹'*b ell2* $\Rightarrow_{CL}$ '*b ell2*›
assumes ‹*a* ≤ *b*› **and** ‹*c* ≥ *0*›

shows ‹$a \otimes_o c \leq b \otimes_o c$›
⟨*proof*⟩

lemma *tensor-op-mono-right*:
fixes a :: ‹$'a$ *ell2* $\Rightarrow_{CL}$ $'a$ *ell2*› **and** b :: ‹$'b$ *ell2* $\Rightarrow_{CL}$ $'b$ *ell2*›
assumes ‹$b \leq c$› **and** ‹$a \geq 0$›
shows ‹$a \otimes_o b \leq a \otimes_o c$›
⟨*proof*⟩

lemma *tensor-op-mono*:
fixes a :: ‹$'a$ *ell2* $\Rightarrow_{CL}$ $'a$ *ell2*› **and** c :: ‹$'b$ *ell2* $\Rightarrow_{CL}$ $'b$ *ell2*›
assumes ‹$a \leq b$› **and** ‹$c \leq d$› **and** ‹$b \geq 0$› **and** ‹$c \geq 0$›
shows ‹$a \otimes_o c \leq b \otimes_o d$›
⟨*proof*⟩

lemma *sandwich-tc-tensor*: ‹*sandwich-tc* ($E \otimes_o F$) (*tc-tensor* t u) = *tc-tensor* (*sandwich-tc* E t) (*sandwich-tc* F u)›
⟨*proof*⟩

lemma *tensor-tc-butterfly*: *tc-tensor* (*tc-butterfly* ψ ψ') (*tc-butterfly* φ φ') = *tc-butterfly* (*tensor-ell2* ψ φ) (*tensor-ell2* ψ' φ')
⟨*proof*⟩

lemma *separating-set-bounded-clinear-tc-tensor*:
shows ‹*separating-set bounded-clinear* (($\lambda(\varrho,\sigma)$. *tc-tensor* ϱ σ) ‘ (*UNIV* × *UNIV*))›
⟨*proof*⟩

lemma *separating-set-bounded-clinear-tc-tensor-nested*:
assumes ‹*separating-set* (*bounded-clinear* :: (- => $'e$::*complex-normed-vector*) ⇒ -) A›
assumes ‹*separating-set* (*bounded-clinear* :: (- => $'e$::*complex-normed-vector*) ⇒ -) B›
shows ‹*separating-set* (*bounded-clinear* :: (- => $'e$::*complex-normed-vector*) ⇒ -) (($\lambda(\varrho,\sigma)$. *tc-tensor* ϱ σ) ‘ (A × B))›
⟨*proof*⟩

lemma *tc-tensor-0-left*[*simp*]: ‹*tc-tensor* 0 x = 0›
⟨*proof*⟩
lemma *tc-tensor-0-right*[*simp*]: ‹*tc-tensor* x 0 = 0›
⟨*proof*⟩

lemma *sandwich-tensor-ell2-right'*: ‹*sandwich* (*tensor-ell2-right* ψ) $*_V$ a = $a \otimes_o$ *selfbutter* ψ›
⟨*proof*⟩

lemma *sandwich-tensor-ell2-left'*: ‹*sandwich* (*tensor-ell2-left* ψ) $*_V$ a = *selfbutter* $\psi \otimes_o a$›

⟨*proof*⟩

13.3 Tensor product of subspaces

definition *tensor-ccsubspace* (**infixr** $\otimes_S$ *70*) **where**
‹*tensor-ccsubspace A B* = *ccspan* $\{\psi \otimes_s \varphi \mid \psi\ \varphi.\ \psi \in$ *space-as-set A* $\wedge\ \varphi \in$ *space-as-set B*$\}$›

lemma *tensor-ccsubspace-via-Proj*: ‹$A \otimes_S B = ($*Proj A* $\otimes_o$ *Proj B*$) *_S \top$›
⟨*proof*⟩

lemma *tensor-ccsubspace-top*[*simp*]: ‹$\top \otimes_S \top = \top$›
⟨*proof*⟩

lemma *tensor-ccsubspace-0-left*[*simp*]: ‹$0 \otimes_S X = 0$›
⟨*proof*⟩

lemma *tensor-ccsubspace-0-right*[*simp*]: ‹$X \otimes_S 0 = 0$›
⟨*proof*⟩

lemma *tensor-ccsubspace-image*: ‹$(A *_S T) \otimes_S (B *_S U) = (A \otimes_o B) *_S (T \otimes_S U)$›
⟨*proof*⟩

lemma *tensor-ccsubspace-bot-left*[*simp*]: ‹$\bot \otimes_S S = \bot$›
⟨*proof*⟩
lemma *tensor-ccsubspace-bot-right*[*simp*]: ‹$S \otimes_S \bot = \bot$›
⟨*proof*⟩

lemma *swap-ell2-tensor-ccsubspace*: ‹*swap-ell2* $*_S (S \otimes_S T) = T \otimes_S S$›
⟨*proof*⟩

lemma *tensor-ccsubspace-right1dim-member*:
assumes ‹$\psi \in$ *space-as-set* $(S \otimes_S$ *ccspan*$\{\varphi\})$›
shows ‹$\exists\,\psi'.\ \psi = \psi' \otimes_s \varphi$›
⟨*proof*⟩

lemma *tensor-ccsubspace-left1dim-member*:
assumes ‹$\psi \in$ *space-as-set* (*ccspan*$\{\varphi\} \otimes_S S)$›
shows ‹$\exists\,\psi'.\ \psi = \varphi \otimes_s \psi'$›
⟨*proof*⟩

lemma *tensor-ell2-mem-tensor-ccsubspace-left*:
assumes ‹$a \otimes_s b \in$ *space-as-set* $(S \otimes_S T)$› **and** ‹$b \neq 0$›
shows ‹$a \in$ *space-as-set S*›
⟨*proof*⟩

lemma *tensor-ell2-mem-tensor-ccsubspace-right*:
assumes ‹$a \otimes_s b \in$ *space-as-set* $(S \otimes_S T)$› **and** ‹$a \neq 0$›
shows ‹$b \in$ *space-as-set T*›

⟨*proof*⟩

lemma *tensor-ell2-in-tensor-ccsubspace*: ‹$a \otimes_s b \in$ *space-as-set* $(A \otimes_S B)$› **if** ‹$a \in$ *space-as-set* A› **and** ‹$b \in$ *space-as-set* B›
— Converse is *tensor-ell2-mem-tensor-ccsubspace-left* and . . . *-right*.
⟨*proof*⟩

lemma *tensor-ccsubspace-INF-left-top*:
fixes S :: ‹$'a \Rightarrow 'b$ *ell2 ccsubspace*›
shows ‹(*INF* $x{\in}X$. $S\ x$) $\otimes_S$ ($\top$::$'c$ *ell2 ccsubspace*) = (*INF* $x{\in}X$. $S\ x \otimes_S \top$)›
⟨*proof*⟩

lemma *tensor-ccsubspace-INF-right-top*:
fixes S :: ‹$'a \Rightarrow 'b$ *ell2 ccsubspace*›
shows ‹($\top$::$'c$ *ell2 ccsubspace*) $\otimes_S$ (*INF* $x{\in}X$. $S\ x$) = (*INF* $x{\in}X$. $\top \otimes_S S\ x$)›
⟨*proof*⟩

lemma *tensor-ccsubspace-INF-left*: ‹(*INF* $x{\in}X$. $S\ x$) $\otimes_S T$ = (*INF* $x{\in}X$. $S\ x \otimes_S T$)› **if** ‹$X \neq \{\}$›
⟨*proof*⟩

lemma *tensor-ccsubspace-INF-right*: ‹(*INF* $x{\in}X$. $T \otimes_S S\ x$) = (*INF* $x{\in}X$. $T \otimes_S S\ x$)› **if** ‹$X \neq \{\}$›
⟨*proof*⟩

lemma *tensor-ccsubspace-ccspan*: ‹*ccspan* $X \otimes_S$ *ccspan* Y = *ccspan* $\{x \otimes_s y \mid x\ y.\ x \in X \wedge y \in Y\}$›
⟨*proof*⟩

lemma *tensor-ccsubspace-mono*: ‹$A \otimes_S B \leq C \otimes_S D$› **if** ‹$A \leq C$› **and** ‹$B \leq D$›
⟨*proof*⟩

lemma *tensor-ccsubspace-element-as-infsum*:
fixes A :: ‹$'a$ *ell2 ccsubspace*› **and** B :: ‹$'b$ *ell2 ccsubspace*›
assumes ‹$\psi \in$ *space-as-set* $(A \otimes_S B)$›
shows ‹$\exists\varphi\ \delta$. ($\forall n$::*nat*. $\varphi\ n \in$ *space-as-set* A) $\wedge$ ($\forall n$. $\delta\ n \in$ *space-as-set* B)
$\wedge$ ((λn. $\varphi\ n \otimes_s \delta\ n$) *has-sum* ψ) *UNIV*›
⟨*proof*⟩

lemma *ortho-tensor-ccsubspace-right*: ‹$-$ ($\top \otimes_S A$) = $\top \otimes_S$ ($-$ A)›
⟨*proof*⟩

lemma *ortho-tensor-ccsubspace-left*: ‹$-$ ($A \otimes_S \top$) = ($-$ A) $\otimes_S \top$›
⟨*proof*⟩

lemma *kernel-tensor-id-left*: ‹*kernel* (*id-cblinfun* $\otimes_o A$) = $\top \otimes_S$ *kernel* A›
⟨*proof*⟩

lemma *kernel-tensor-id-right*: ‹*kernel* ($A \otimes_o$ *id-cblinfun*) = *kernel* $A \otimes_S \top$›

⟨*proof*⟩

lemma *eigenspace-tensor-id-left*: ‹*eigenspace c* (*id-cblinfun* $\otimes_o$ *A*) = ⊤ $\otimes_S$ *eigenspace c A*›
⟨*proof*⟩

lemma *eigenspace-tensor-id-right*: ‹*eigenspace c* (*A* $\otimes_o$ *id-cblinfun*) = *eigenspace c A* $\otimes_S$ ⊤›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

14 *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* :: ‹(('*a* × '*c*) *ell2*, ('*b* × '*c*) *ell2*) *trace-class* ⇒ ('*a ell2*, '*b ell2*) *trace-class*› **where**
‹*partial-trace t* = ($\sum_\infty$*j. compose-tcl* (*compose-tcr* ((*tensor-ell2-right* (*ket j*))*) *t*) (*tensor-ell2-right* (*ket j*)))›

lemma *partial-trace-def'*: ‹*partial-trace t* = ($\sum_\infty$*j. sandwich-tc* ((*tensor-ell2-right* (*ket j*))*) *t*)›
— We cannot use this as the definition of *partial-trace* because this definition has a more restricted type (*t* is a square operator).
  ⟨*proof*⟩

lemma *partial-trace-abs-summable*:
‹(λ*j. compose-tcl* (*compose-tcr* ((*tensor-ell2-right* (*ket j*))*) *t*) (*tensor-ell2-right* (*ket j*))) *abs-summable-on UNIV*›
  and *partial-trace-has-sum*:
  ‹((λ*j. compose-tcl* (*compose-tcr* ((*tensor-ell2-right* (*ket j*))*) *t*) (*tensor-ell2-right* (*ket j*))) *has-sum partial-trace t*) *UNIV*›
  and *partial-trace-norm-reducing*: ‹*norm* (*partial-trace t*) ≤ *norm t*›
⟨*proof*⟩

lemma *partial-trace-abs-summable'*:
‹(λ*j. sandwich-tc* ((*tensor-ell2-right* (*ket j*))*) *t*) *abs-summable-on UNIV*›
  and *partial-trace-has-sum'*:
‹((λ*j. 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* $\otimes_o$ *id-cblinfun*))›
⟨*proof*⟩

lemma *right-amplification-weak-star-cont*[*simp*]:
‹*continuous-map weak-star-topology weak-star-topology* (λ*a*. *a* $\otimes_o$ *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* (λ*b*. *id-cblinfun* $\otimes_o$ *b* :: ('*c*×'*a*) *ell2* $\Rightarrow_{CL}$ ('*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*:
‹((λ*z*. ((*x* $\otimes_s$ *ket z*) $\cdot_C$ (*from-trace-class* ϱ $*_V$ (*y* $\otimes_s$ *ket z*))))
has-sum x $\cdot_C$ (*from-trace-class* (*partial-trace* ϱ) $*_V$ *y*)) *UNIV*›
⟨*proof*⟩

lemma *vector-sandwich-partial-trace*:
‹*x* $\cdot_C$ (*from-trace-class* (*partial-trace* ϱ) $*_V$ *y*) =
($\sum_\infty$*z*. ((*x* $\otimes_s$ *ket z*) $\cdot_C$ (*from-trace-class* ϱ $*_V$ (*y* $\otimes_s$ *ket z*))))›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

15 *Von-Neumann-Algebras* – Von Neumann algebras and the double commutant theorem

theory *Von-Neumann-Algebras*
  imports *Hilbert-Space-Tensor-Product*
begin

unbundle *cblinfun-syntax*

15.1 Commutants

definition ‹*commutant F* = {*x*. ∀ *y*∈*F*. *x* o_{CL} *y* = *y* o_{CL} *x*}›

lemma *sandwich-unitary-commutant*:
  fixes *U* :: ‹′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*b*::*chilbert-space*›
  assumes [*simp*]: ‹*unitary U*›
  shows ‹*sandwich U* ‘ *commutant X* = *commutant* (*sandwich U* ‘ *X*)›
⟨*proof*⟩

lemma *commutant-tensor1*: ‹*commutant* (*range* (λ*a*. *a* $\otimes_o$ *id-cblinfun*)) = *range* (λ*b*. *id-cblinfun* $\otimes_o$ *b*)›
⟨*proof*⟩

lemma *csubspace-commutant*[*simp*]: ‹*csubspace* (*commutant X*)›
  ⟨*proof*⟩

lemma *closed-commutant*[*simp*]: ‹*closed* (*commutant X*)›
⟨*proof*⟩

lemma *closed-csubspace-commutant*[*simp*]: ‹*closed-csubspace* (*commutant X*)›
  ⟨*proof*⟩

lemma *commutant-mult*: ‹*a* o_{CL} *b* ∈ *commutant X*› **if** ‹*a* ∈ *commutant X*› **and** ‹*b* ∈ *commutant X*›
  ⟨*proof*⟩

lemma *double-commutant-grows*[*simp*]: ‹*X* ⊆ *commutant* (*commutant X*)›
  ⟨*proof*⟩

lemma *commutant-antimono*: ‹*X* ⊆ *Y* ⟹ *commutant X* ⊇ *commutant Y*›
  ⟨*proof*⟩

lemma *triple-commutant*[*simp*]: ‹*commutant* (*commutant* (*commutant X*)) = *commutant X*›
  ⟨*proof*⟩

lemma *commutant-adj*: ‹*adj* ‘ *commutant X* = *commutant* (*adj* ‘ *X*)›

⟨*proof*⟩

lemma *commutant-empty*[*simp*]: ‹*commutant* {} = *UNIV*›
⟨*proof*⟩

lemma *commutant-weak-star-closed*[*simp*]: ‹*closedin weak-star-topology* (*commutant X*)›
⟨*proof*⟩

lemma *cspan-in-double-commutant*: ‹*cspan X* ⊆ *commutant* (*commutant X*)›
⟨*proof*⟩

lemma *weak-star-closure-in-double-commutant*: ‹*weak-star-topology closure-of X* ⊆ *commutant* (*commutant X*)›
⟨*proof*⟩

lemma *weak-star-closure-cspan-in-double-commutant*: ‹*weak-star-topology closure-of cspan X* ⊆ *commutant* (*commutant X*)›
⟨*proof*⟩

lemma *commutant-memberI*:
assumes ‹$\bigwedge y.\ y \in X \Longrightarrow x\ o_{CL}\ y = y\ o_{CL}\ x$›
shows ‹*x* ∈ *commutant X*›
⟨*proof*⟩

lemma *commutant-sot-closed*: ‹*closedin cstrong-operator-topology* (*commutant A*)›
— [2], Exercise IX.6.2
⟨*proof*⟩

lemma *commutant-tensor1'*: ‹*commutant* (*range* (λ*a*. *id-cblinfun* $\otimes_o$ *a*)) = *range* (λ*b*. *b* $\otimes_o$ *id-cblinfun*)›
⟨*proof*⟩

lemma *closed-map-sot-tensor-op-id-right*:
‹*closed-map cstrong-operator-topology cstrong-operator-topology* (λ*a*. *a* $\otimes_o$ *id-cblinfun* :: (*'a* × *'b*) *ell2* $\Rightarrow_{CL}$ (*'a* × *'b*) *ell2*)›
⟨*proof*⟩

lemma *id-in-commutant*[*iff*]: ‹*id-cblinfun* ∈ *commutant A*›
⟨*proof*⟩

lemma *double-commutant-hull*: ‹*commutant* (*commutant X*) = (λ*X*. *commutant* (*commutant X*) = *X*) *hull X*›
⟨*proof*⟩

lemma *commutant-adj-closed*: ‹$(\bigwedge x.\ x \in X \Longrightarrow x{*} \in X) \Longrightarrow x \in$ *commutant X* $\Longrightarrow x{*} \in$ *commutant X*›

⟨proof⟩

lemma *double-commutant-Un-left*: ‹$commutant\ (commutant\ (commutant\ (commutant\ X) \cup Y)) = commutant\ (commutant\ (X \cup Y))$›
⟨proof⟩

lemma *double-commutant-Un-right*: ‹$commutant\ (commutant\ (X \cup commutant\ (commutant\ Y))) = commutant\ (commutant\ (X \cup Y))$›
⟨proof⟩

lemma *amplification-double-commutant-commute*:
‹$commutant\ (commutant\ ((\lambda a.\ a \otimes_o id\text{-}cblinfun)\ `\ X))$
$= (\lambda a.\ a \otimes_o id\text{-}cblinfun)\ `\ commutant\ (commutant\ X)$›
— [7], Corollary IV.1.5
⟨proof⟩

lemma *amplification-double-commutant-commute'*:
‹$commutant\ (commutant\ ((\lambda a.\ id\text{-}cblinfun \otimes_o a)\ `\ X))$
$= (\lambda a.\ id\text{-}cblinfun \otimes_o a)\ `\ commutant\ (commutant\ X)$›
⟨proof⟩

lemma *commutant-cspan*: ‹$commutant\ (cspan\ A) = commutant\ A$›
⟨proof⟩

lemma *double-commutant-grows'*: ‹$x \in X \Longrightarrow x \in commutant\ (commutant\ X)$›
⟨proof⟩

15.2 Double commutant theorem

fun *inflation-op'* :: ‹$nat \Rightarrow ('a\ ell2 \Rightarrow_{CL} 'b\ ell2)\ list \Rightarrow ('a \times nat)\ ell2 \Rightarrow_{CL} ('b \times nat)\ ell2$›
where
‹$inflation\text{-}op'\ n\ Nil = 0$›
| ‹$inflation\text{-}op'\ n\ (a\#as) = (a \otimes_o butterfly\ (ket\ n)\ (ket\ n)) + inflation\text{-}op'\ (n+1)\ as$›

abbreviation ‹$inflation\text{-}op \equiv inflation\text{-}op'\ 0$›

fun *inflation-state'* :: ‹$nat \Rightarrow 'a\ ell2\ list \Rightarrow ('a \times nat)\ ell2$› **where**
‹$inflation\text{-}state'\ n\ Nil = 0$›
| ‹$inflation\text{-}state'\ n\ (a\#as) = (a \otimes_s ket\ n) + inflation\text{-}state'\ (n+1)\ as$›

abbreviation ‹$inflation\text{-}state \equiv inflation\text{-}state'\ 0$›

fun *inflation-space'* :: ‹$nat \Rightarrow 'a\ ell2\ ccsubspace\ list \Rightarrow ('a \times nat)\ ell2\ ccsubspace$› **where**
‹$inflation\text{-}space'\ n\ Nil = 0$›
| ‹$inflation\text{-}space'\ n\ (S\#Ss) = (S \otimes_S ccspan\ \{ket\ n\}) + inflation\text{-}space'\ (n+1)\ Ss$›

abbreviation ‹$inflation\text{-}space \equiv inflation\text{-}space'\ 0$›

definition *inflation-carrier* :: ‹*nat* ⇒ (*'a*×*nat*) *ell2 ccsubspace*› **where**
‹*inflation-carrier n* = *inflation-space* (*replicate n* ⊤)›

definition *inflation-op-carrier* :: ‹*nat* ⇒ ((*'a*×*nat*) *ell2* $\Rightarrow_{CL}$ (*'b*×*nat*) *ell2*) *set*› **where**
‹*inflation-op-carrier n* = { *Proj* (*inflation-carrier n*) o_{CL} *a* o_{CL} *Proj* (*inflation-carrier n*) | *a*. *True* }›

lemma *inflation-op-compose-outside*: ‹*inflation-op' m ops* o_{CL} (*a* $\otimes_o$ *butterfly* (*ket n*) (*ket n*)) = *0*› **if** ‹*n* < *m*›
⟨*proof*⟩

lemma *inflation-op-compose-outside-rev*: ‹(*a* $\otimes_o$ *butterfly* (*ket n*) (*ket n*)) o_{CL} *inflation-op' m ops* = *0*› **if** ‹*n* < *m*›
⟨*proof*⟩

lemma *Proj-inflation-carrier*: ‹*Proj* (*inflation-carrier n*) = *inflation-op* (*replicate n id-cblinfun*)›
⟨*proof*⟩

lemma *inflation-op-carrierI*:
assumes ‹*Proj* (*inflation-carrier n*) o_{CL} *a* o_{CL} *Proj* (*inflation-carrier n*) = *a*›
shows ‹*a* ∈ *inflation-op-carrier n*›
⟨*proof*⟩

lemma *inflation-op-compose*: ‹*inflation-op' n ops1* o_{CL} *inflation-op' n ops2* = *inflation-op' n* (*map2 cblinfun-compose ops1 ops2*)›
⟨*proof*⟩

lemma *inflation-op-in-carrier*: ‹*inflation-op ops* ∈ *inflation-op-carrier n*› **if** ‹*length ops* ≤ *n*›
⟨*proof*⟩

lemma *inflation-op'-apply-tensor-outside*: ‹*n* < *m* ⟹ *inflation-op' m as* $*_V$ (*v* $\otimes_s$ *ket n*) = *0*›
⟨*proof*⟩

lemma *inflation-op'-compose-tensor-outside*: ‹*n* < *m* ⟹ *inflation-op' m as* o_{CL} *tensor-ell2-right* (*ket n*) = *0*›
⟨*proof*⟩

lemma *inflation-state'-apply-tensor-outside*: ‹*n* < *m* ⟹ (*a* $\otimes_o$ *butterfly* ψ (*ket n*)) $*_V$ *inflation-state' m vs* = *0*›
⟨*proof*⟩

lemma *inflation-op-apply-inflation-state*: ‹*inflation-op' n ops* $*_V$ *inflation-state' n vecs* = *inflation-state' n* (*map2 cblinfun-apply ops vecs*)›
⟨*proof*⟩

lemma *inflation-state-in-carrier*: ‹*inflation-state vecs* ∈ *space-as-set* (*inflation-carrier n*)› **if** ‹*length vecs* + *m* ≤ *n*›
⟨*proof*⟩

lemma *inflation-op'-apply-tensor-outside'*: ‹$n \geq$ *length as* + $m \Longrightarrow$ *inflation-op' m as* $*_V$ ($v \otimes_s$ *ket n*) = *0*›
⟨*proof*⟩

lemma *Proj-inflation-carrier-outside*: ‹*Proj* (*inflation-carrier n*) $*_V$ ($\psi \otimes_s$ *ket i*) = *0*› **if** ‹$i \geq n$›
⟨*proof*⟩

lemma *inflation-state'-is-orthogonal-outside*: ‹$n < m \Longrightarrow$ *is-orthogonal* ($a \otimes_s$ *ket n*) (*inflation-state' m vs*)›
⟨*proof*⟩

lemma *inflation-op-adj*: ‹(*inflation-op' n ops*)∗ = *inflation-op' n* (*map adj ops*)›
⟨*proof*⟩

lemma *inflation-state0*:
assumes ‹$\bigwedge v.\ v \in$ *set f* $\Longrightarrow v = 0$›
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 ‹$c *_C$ *inflation-state' n f* = *inflation-state' n* (*map* (*scaleC c*) *f*)›
⟨*proof*⟩

lemma *inflation-op-compose-tensor-ell2-right*:
assumes ‹$i \geq n$› **and** ‹$i < n$ + *length f*›
shows ‹*inflation-op' n f* o_{CL} *tensor-ell2-right* (*ket i*) = *tensor-ell2-right* (*ket i*) o_{CL} ($f!(i-n)$)›
⟨*proof*⟩

lemma *inflation-op-apply*:
assumes ‹$i \geq n$› **and** ‹$i < n$ + *length f*›
shows ‹*inflation-op' n f* $*_V$ ($\psi \otimes_s$ *ket i*) = ($f!(i-n) *_V \psi$) $\otimes_s$ *ket i*›
⟨*proof*⟩

lemma *norm-inflation-state*:
‹*norm* (*inflation-state' n f*) = *sqrt* ($\sum v \leftarrow f.$ (*norm v*)^2)›
⟨*proof*⟩

lemma *cstrong-operator-topology-in-closure-algebraicI*:
— [2], Proposition IX.5.3
assumes *space*: ‹*csubspace A*›
assumes *mult*: ‹$\bigwedge a\ a'.\ a \in A \Longrightarrow a' \in A \Longrightarrow a\ o_{CL}\ a' \in A$›
assumes *one*: ‹*id-cblinfun* ∈ *A*›
assumes *main*: ‹$\bigwedge n\ S.\ S \le$ *inflation-carrier* $n \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow$ *inflation-op* (*replicate* *n* *a*) $*_S\ S \le S) \Longrightarrow$
inflation-op (*replicate* *n* *b*) $*_S\ S \le S$›
shows ‹*b* ∈ *cstrong-operator-topology closure-of A*›
⟨*proof*⟩

lemma *commutant-inflation*:
— One direction of [2], Proposition IX.6.2.
fixes *n*
defines ‹$\bigwedge X.$ *commutant′ X* ≡ *commutant X* ∩ *inflation-op-carrier n*›
shows ‹(*λa. inflation-op* (*replicate n a*)) ‘ *commutant* (*commutant A*)
⊆ *commutant′* (*commutant′* ((*λa. inflation-op* (*replicate n a*)) ‘ *A*))›
⟨*proof*⟩

lemma *double-commutant-theorem-aux*:
— Basically the double commutant theorem, except that we restricted to spaces of the form *′a ell2*
— [2], Proposition IX.6.4
fixes *A* :: ‹(*′a ell2* $\Rightarrow_{CL}$ *′a ell2*) *set*›
assumes ‹*csubspace A*›
assumes ‹$\bigwedge a\ a'.\ a \in A \Longrightarrow a' \in A \Longrightarrow a\ o_{CL}\ a' \in A$›
assumes ‹*id-cblinfun* ∈ *A*›
assumes ‹$\bigwedge a.\ a \in A \Longrightarrow a* \in A$›
shows ‹*commutant* (*commutant A*) = *cstrong-operator-topology closure-of A*›
⟨*proof*⟩

lemma *double-commutant-theorem-aux2*:
— Basically the double commutant theorem, except that we restricted to spaces of typeclass *not-singleton*
— [2], Proposition IX.6.4
fixes *A* :: ‹(*′a*::{*chilbert-space*,*not-singleton*} $\Rightarrow_{CL}$ *′a*) *set*›
assumes *subspace*: ‹*csubspace A*›
assumes *mult*: ‹$\bigwedge a\ a'.\ a \in A \Longrightarrow a' \in A \Longrightarrow a\ o_{CL}\ a' \in A$›
assumes *id*: ‹*id-cblinfun* ∈ *A*›
assumes *adj*: ‹$\bigwedge a.\ a \in A \Longrightarrow a* \in A$›
shows ‹*commutant* (*commutant A*) = *cstrong-operator-topology closure-of A*›
⟨*proof*⟩

lemma *double-commutant-theorem*:
— [2], Proposition IX.6.4
fixes *A* :: ‹(*′a*::{*chilbert-space*} $\Rightarrow_{CL}$ *′a*) *set*›
assumes *subspace*: ‹*csubspace A*›

assumes *mult*: ‹$\bigwedge a\ a'.\ a \in A \Longrightarrow a' \in A \Longrightarrow a\ o_{CL}\ a' \in A$›
assumes *id*: ‹*id-cblinfun* $\in A$›
assumes *adj*: ‹$\bigwedge a.\ a \in A \Longrightarrow a* \in A$›
shows ‹*commutant* (*commutant* A) = *cstrong-operator-topology closure-of* A›
⟨*proof*⟩

hide-fact *double-commutant-theorem-aux double-commutant-theorem-aux2*

lemma *double-commutant-theorem-span*:
fixes A :: ‹('a::{chilbert-space} $\Rightarrow_{CL}$ 'a) *set*›
assumes *mult*: ‹$\bigwedge a\ a'.\ a \in A \Longrightarrow a' \in A \Longrightarrow a\ o_{CL}\ a' \in A$›
assumes *id*: ‹*id-cblinfun* $\in A$›
assumes *adj*: ‹$\bigwedge a.\ a \in A \Longrightarrow a* \in A$›
shows ‹*commutant* (*commutant* A) = *cstrong-operator-topology closure-of* (*cspan* A)›
⟨*proof*⟩

15.3 Von Neumann Algebras

definition *one-algebra* :: ‹('a $\Rightarrow_{CL}$ 'a::chilbert-space) *set*› **where**
‹*one-algebra* = *range* ($\lambda c.\ c *_C$ *id-cblinfun*)›

definition *von-neumann-algebra* **where** ‹*von-neumann-algebra* $A \longleftrightarrow (\forall a{\in}A.\ a* \in A) \wedge$ *commutant* (*commutant* A) = A›
definition *von-neumann-factor* **where** ‹*von-neumann-factor* $A \longleftrightarrow$ *von-neumann-algebra* $A \wedge$ $A \cap$ *commutant* A = *one-algebra*›

lemma *von-neumann-algebraI*: ‹($\bigwedge a.\ a{\in}A \Longrightarrow a* \in A$) $\Longrightarrow$ *commutant* (*commutant* A) $\subseteq A$ $\Longrightarrow$ *von-neumann-algebra* A› **for** $\mathfrak{F}$
⟨*proof*⟩

lemma *von-neumann-factorI*:
assumes ‹*von-neumann-algebra* A›
assumes ‹$A \cap$ *commutant* $A \subseteq$ *one-algebra*›
shows ‹*von-neumann-factor* A›
⟨*proof*⟩

lemma *commutant-UNIV*: ‹*commutant* (*UNIV* :: ('a $\Rightarrow_{CL}$ 'a::chilbert-space) *set*) = *one-algebra*›

⟨*proof*⟩

lemma *von-neumann-algebra-UNIV*: ‹*von-neumann-algebra UNIV*›
⟨*proof*⟩

lemma *von-neumann-factor-UNIV*: ‹*von-neumann-factor UNIV*›
⟨*proof*⟩

lemma *von-neumann-algebra-UNION*:

assumes ‹⋀*x*. *x* ∈ *X* ⟹ *von-neumann-algebra* (*A x*)›
shows ‹*von-neumann-algebra* (*commutant* (*commutant* (⋃*x*∈*X*. *A x*)))›
⟨*proof*⟩

lemma *von-neumann-algebra-union*:
assumes ‹*von-neumann-algebra A*›
assumes ‹*von-neumann-algebra B*›
shows ‹*von-neumann-algebra* (*commutant* (*commutant* (*A* ∪ *B*)))›
⟨*proof*⟩

lemma *von-neumann-algebra-commutant*: ‹*von-neumann-algebra* (*commutant A*)› **if** ‹*von-neumann-algebra A*›
⟨*proof*⟩

lemma *von-neumann-algebra-def-sot*:
‹*von-neumann-algebra* 𝔉 ⟷
(∀ *a*∈𝔉. *a*∗ ∈ 𝔉) ∧ *csubspace* 𝔉 ∧ (∀ *a*∈𝔉. ∀ *b*∈𝔉. *a* o_{CL} *b* ∈ 𝔉) ∧ *id-cblinfun* ∈ 𝔉 ∧
closedin cstrong-operator-topology 𝔉›
⟨*proof*⟩

lemma *double-commutant-hull′*:
assumes ‹⋀*x*. *x* ∈ *X* ⟹ *x*∗ ∈ *X*›
shows ‹*commutant* (*commutant X*) = *von-neumann-algebra hull X*›
⟨*proof*⟩

lemma *commutant-one-algebra*: ‹*commutant one-algebra* = *UNIV*›
⟨*proof*⟩

definition *tensor-vn* (**infixr** $\otimes_{vN}$ *70*) **where**
‹*tensor-vn X Y* = *commutant* (*commutant* ((λ*a*. *a* $\otimes_o$ *id-cblinfun*) ‘ *X* ∪ (λ*a*. *id-cblinfun* $\otimes_o$ *a*) ‘ *Y*))›

lemma *von-neumann-algebra-adj-image*: ‹*von-neumann-algebra X* ⟹ *adj* ‘ *X* = *X*›
⟨*proof*⟩

lemma *von-neumann-algebra-tensor-vn*:
assumes ‹*von-neumann-algebra X*›
assumes ‹*von-neumann-algebra Y*›
shows ‹*von-neumann-algebra* (*X* $\otimes_{vN}$ *Y*)›
⟨*proof*⟩

lemma *tensor-vn-one-one*[*simp*]: ‹*one-algebra* $\otimes_{vN}$ *one-algebra* = *one-algebra*›
⟨*proof*⟩

lemma *sandwich-swap-tensor-vn*: ‹*sandwich swap-ell2* ‘ (*X* $\otimes_{vN}$ *Y*) = *Y* $\otimes_{vN}$ *X*›
⟨*proof*⟩

lemma *tensor-vn-one-left*: ‹*one-algebra* $\otimes_{vN}$ *X* = (λ*x*. *id-cblinfun* $\otimes_o$ *x*) ‘ *X*› **if** ‹*von-neumann-algebra X*›
⟨*proof*⟩
lemma *tensor-vn-one-right*: ‹*X* $\otimes_{vN}$ *one-algebra* = (λ*x*. *x* $\otimes_o$ *id-cblinfun*) ‘ *X*› **if** ‹*von-neumann-algebra X*›
⟨*proof*⟩

lemma *double-commutant-in-vn-algI*: ‹*commutant* (*commutant X*) ⊆ *Y*›
 if ‹*von-neumann-algebra Y*› **and** ‹*X* ⊆ *Y*›
 ⟨*proof*⟩

lemma *von-neumann-algebra-compose*:
 assumes ‹*von-neumann-algebra M*›
 assumes ‹*x* ∈ *M*› **and** ‹*y* ∈ *M*›
 shows ‹*x* o_{CL} *y* ∈ *M*›
 ⟨*proof*⟩

lemma *von-neumann-algebra-id*:
 assumes ‹*von-neumann-algebra M*›
 shows ‹*id-cblinfun* ∈ *M*›
 ⟨*proof*⟩

lemma *tensor-vn-UNIV*[*simp*]: ‹*UNIV* $\otimes_{vN}$ *UNIV* = (*UNIV* :: ((*'a*×*'b*) *ell2* $\Rightarrow_{CL}$ -) *set*)›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

16 *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* :: *nat* ⇒ *nat* ⇒ (*nat* × *nat*) ⇒ *nat*
 where *tensor-pack X Y* = (λ(*x*, *y*). *x* ∗ *Y* + *y*)

definition *tensor-unpack* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ (*nat* × *nat*)
  where *tensor-unpack X Y xy* = (*xy div Y*, *xy mod Y*)

lemma *tensor-unpack-inj*:
  assumes *i* < *A* ∗ *B* **and** *j* < *A* ∗ *B*
  shows *tensor-unpack A B i* = *tensor-unpack A B j* ⟷ *i* = *j*
  ⟨*proof*⟩

lemma *tensor-unpack-bound1*[*simp*]: *i* < *A* ∗ *B* ⟹ *fst* (*tensor-unpack A B i*) < *A*
  ⟨*proof*⟩
lemma *tensor-unpack-bound2*[*simp*]: *i* < *A* ∗ *B* ⟹ *snd* (*tensor-unpack A B i*) < *B*
  ⟨*proof*⟩

lemma *tensor-unpack-fstfst*: ‹*fst* (*tensor-unpack A B* (*fst* (*tensor-unpack* (*A* ∗ *B*) *C i*)))
    = *fst* (*tensor-unpack A* (*B* ∗ *C*) *i*)›
  ⟨*proof*⟩
lemma *tensor-unpack-sndsnd*: ‹*snd* (*tensor-unpack B C* (*snd* (*tensor-unpack A* (*B* ∗ *C*) *i*)))
    = *snd* (*tensor-unpack* (*A* ∗ *B*) *C i*)›
  ⟨*proof*⟩
lemma *tensor-unpack-fstsnd*: ‹*fst* (*tensor-unpack B C* (*snd* (*tensor-unpack A* (*B* ∗ *C*) *i*)))
    = *snd* (*tensor-unpack A B* (*fst* (*tensor-unpack* (*A* ∗ *B*) *C i*)))›
  ⟨*proof*⟩

definition *tensor-state-jnf* ψ φ = (*let d1* = *dim-vec* ψ *in let d2* = *dim-vec* φ *in*
  vec (*d1*∗*d2*) (λ*i*. *let* (*i1*,*i2*) = *tensor-unpack d1 d2 i in* (*vec-index* ψ *i1*) ∗ (*vec-index* φ *i2*)))

lemma *tensor-state-jnf-dim*[*simp*]: ‹*dim-vec* (*tensor-state-jnf* ψ φ) = *dim-vec* ψ ∗ *dim-vec* φ›
  ⟨*proof*⟩

lemma *enum-prod-nth-tensor-unpack*:
  assumes ‹*i* < *CARD*(′*a*) ∗ *CARD*(′*b*)›
  shows (*Enum.enum* ! *i* :: ′*a*::*enum*×′*b*::*enum*) =
      (*let* (*i1*,*i2*) = *tensor-unpack CARD*(′*a*) *CARD*(′*b*) *i in*
        (*Enum.enum* ! *i1*, *Enum.enum* ! *i2*))
  ⟨*proof*⟩

lemma *vec-of-basis-enum-tensor-state-index*:
  fixes ψ :: ‹′*a*::*enum ell2*› **and** φ :: ‹′*b*::*enum ell2*›
  assumes [*simp*]: ‹*i* < *CARD*(′*a*) ∗ *CARD*(′*b*)›
  shows ‹*vec-of-basis-enum* (ψ $\otimes_s$ φ) $ *i* = (*let* (*i1*,*i2*) = *tensor-unpack CARD*(′*a*) *CARD*(′*b*)
i in
   vec-of-basis-enum ψ $ *i1* ∗ *vec-of-basis-enum* φ $ *i2*)›
⟨*proof*⟩

lemma *vec-of-basis-enum-tensor-state*:

fixes *ψ* :: ‹*'a*::*enum ell2*› **and** *φ* :: ‹*'b*::*enum ell2*›
shows ‹*vec-of-basis-enum* (*ψ* $\otimes_s$ *φ*) = *tensor-state-jnf* (*vec-of-basis-enum ψ*) (*vec-of-basis-enum φ*)›
⟨*proof*⟩

lemma *mat-of-cblinfun-tensor-op-index*:
fixes *a* :: ‹*'a*::*enum ell2* $\Rightarrow_{CL}$ *'b*::*enum ell2*› **and** *b* :: ‹*'c*::*enum ell2* $\Rightarrow_{CL}$ *'d*::*enum ell2*›
assumes [*simp*]: ‹*i* < *CARD*(*'b*) ∗ *CARD*(*'d*)›
assumes [*simp*]: ‹*j* < *CARD*(*'a*) ∗ *CARD*(*'c*)›
shows ‹*mat-of-cblinfun* (*tensor-op a b*) $$ (*i*,*j*) =
(*let* (*i1*,*i2*) = *tensor-unpack CARD*(*'b*) *CARD*(*'d*) *i in*
let (*j1*,*j2*) = *tensor-unpack CARD*(*'a*) *CARD*(*'c*) *j in*
mat-of-cblinfun a $$ (*i1*,*j1*) ∗ *mat-of-cblinfun b* $$ (*i2*,*j2*))›
⟨*proof*⟩

definition *tensor-op-jnf A B* =
(*let r1* = *dim-row A in*
let c1 = *dim-col A in*
let r2 = *dim-row B in*
let c2 = *dim-col B in*
mat (*r1* ∗ *r2*) (*c1* ∗ *c2*)
(λ(*i*,*j*). *let* (*i1*,*i2*) = *tensor-unpack r1 r2 i in*
let (*j1*,*j2*) = *tensor-unpack c1 c2 j in*
(*A* $$ (*i1*,*j1*)) ∗ (*B* $$ (*i2*,*j2*))))

lemma *tensor-op-jnf-dim*[*simp*]:
‹*dim-row* (*tensor-op-jnf a b*) = *dim-row a* ∗ *dim-row b*›
‹*dim-col* (*tensor-op-jnf a b*) = *dim-col a* ∗ *dim-col b*›
⟨*proof*⟩

lemma *mat-of-cblinfun-tensor-op*:
fixes *a* :: ‹*'a*::*enum ell2* $\Rightarrow_{CL}$ *'b*::*enum ell2*› **and** *b* :: ‹*'c*::*enum ell2* $\Rightarrow_{CL}$ *'d*::*enum ell2*›
shows ‹*mat-of-cblinfun* (*tensor-op a b*) = *tensor-op-jnf* (*mat-of-cblinfun a*) (*mat-of-cblinfun b*)›
⟨*proof*⟩

lemma *mat-of-cblinfun-assoc-ell2'*[*simp*]:
‹*mat-of-cblinfun* (*assoc-ell2*∗ :: ((*'a*::*enum*×(*'b*::*enum*×*'c*::*enum*)) *ell2* $\Rightarrow_{CL}$ -)) = *one-mat* (*CARD*(*'a*)∗*CARD*(*'b*)∗*CARD*(*'c*))›
(**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* $\Rightarrow_{CL}$ -)) = *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. tr(ab) = tr(ba)? Mathoverflow answer, https://mathoverflow.net/a/76389/101775, 2011–2014.

[4] A. Henriques and A. Taghavi. tr(ab) = tr(ba)? Mathoverflow question, https://mathoverflow.net/questions/76386/trab-trba, 2011–2014.

[5] E. W. (https://math.stackexchange.com/users/86856/eric wofsey). A bounded increasing net converges formulated using filters. Mathematics Stack Exchange (Answer). URL:https://math.stackexchange.com/q/4749216 (version: 2023-08-07).

[6] O. Kunar and A. Popescu. From types to sets by local type definition in higher-order logic. *Journal of Automated Reasoning*, 62(2):237260, June 2018.

[7] M. Takesaki. *Theory of operator algebras I.* Springer, New York, NY, Nov. 2011.

[8] D. Unruh. Quantum references. arXiv:2105.10914v3 [cs.LO], 2024.

[9] F. Wecken. Zur theorie linearer operatoren. *Math. Ann.*, 110:722–725, 1935.