Kraus Maps*

Dominique Unruh

October 13, 2025

Abstract

We formalize Kraus maps [1, Section 3], i.e., quantum channels of the form $\rho \mapsto \sum_x M_x \rho M_x^\dagger$ for suitable families of "Kraus operators" M_x. (In the finite-dimensional setting and the setting of separable Hilbert spaces, those are known to be equivalent to completely-positive maps, another common formalization of quantum channels.) Our results hold for arbitrary (i.e., not necessarily finite-dimensional or separable) Hilbert spaces.

Specifically, in theory `Kraus_Families`, we formalize the type (α, β, ξ) `kraus_family` of families of Kraus operators M_x (Kraus families for short), from trace-class operators on Hilbert space α to those on β, indexed by x of type ξ. This induces both a Kraus map $\rho \mapsto \sum_x M_x \rho M_x^\dagger$, as well as a quantum measurement with outcomes of type ξ.

We define and study various special Kraus families such as zero, identity, application of an operator, sum of two Kraus maps, sequential composition, infinite sum, random sampling, trace, tensor products, and complete measurements.

Furthermore, since working with explicit Kraus families can be cumbersome when the specific family of operators is not relevant, in theory `Kraus_Maps`, we define a Kraus map to be a function between two spaces that is of the form $\rho \mapsto \sum_x M_x \rho M_x^\dagger$ for some Kraus family, and restate our results in terms of such functions.

Contents

*Supported by the ERC consolidator grant CerQuS (819317), and the Estonian Cluster of Excellence "Foundations of the Universe" (TK202).

1 Miscelleanous missing theorems

theory *Misc-Kraus-Maps*
  imports
    Hilbert-Space-Tensor-Product.Hilbert-Space-Tensor-Product
    Hilbert-Space-Tensor-Product.Von-Neumann-Algebras
begin

unbundle *cblinfun-syntax*

lemma *abs-summable-norm*:
  assumes ‹*f abs-summable-on A*›
  shows ‹$(\lambda x.\ norm\ (f\ x))$ *abs-summable-on A*›
  ⟨*proof*⟩

lemma *abs-summable-on-add*:
  assumes ‹*f abs-summable-on A*› **and** ‹*g abs-summable-on A*›
  shows ‹$(\lambda x.\ f\ x + g\ x)$ *abs-summable-on A*›
⟨*proof*⟩

lemma *bdd-above-transform-mono-pos*:
  assumes *bdd*: ‹*bdd-above* $((\lambda x.\ g\ x)$ ‘ $M)$›
  assumes *gpos*: ‹$\bigwedge x.\ x \in M \Longrightarrow g\ x \geq 0$›
  assumes *mono*: ‹*mono-on* (*Collect* $((\leq)\ 0))\ f$›
  shows ‹*bdd-above* $((\lambda x.\ f\ (g\ x))$ ‘ $M)$›

⟨*proof*⟩

lemma *Ex-iffI*:
  assumes ‹$\bigwedge x.\ P\ x \Longrightarrow Q\ (f\ x)$›
  assumes ‹$\bigwedge x.\ Q\ x \Longrightarrow P\ (g\ x)$›
  shows ‹$Ex\ P \longleftrightarrow Ex\ Q$›
  ⟨*proof*⟩

lemma *has-sum-Sigma'-banach*:
  fixes A :: *'a set* **and** B :: *'a ⇒ 'b set*
    and f :: ‹*'a ⇒ 'b ⇒ 'c::banach*›
  assumes $((\lambda(x,y).\ f\ x\ y)$ *has-sum* $S)$ (*Sigma* A B)
  shows ‹$((\lambda x.$ *infsum* $(f\ x)\ (B\ x))$ *has-sum* $S)\ A$›
  ⟨*proof*⟩

lemma *infsum-Sigma-topological-monoid*:
  fixes A :: *'a set* **and** B :: *'a ⇒ 'b set*
    and f :: ‹*'a × 'b ⇒ 'c::{topological-comm-monoid-add, t3-space}*›
  assumes *summableAB*: f *summable-on* (*Sigma* A B)
  assumes *summableB*: ‹$\bigwedge x.\ x\in A \Longrightarrow (\lambda y.\ f\ (x,\ y))$ *summable-on* $(B\ x)$›
  shows *infsum* f (*Sigma* A B) $= (\sum_\infty x\in A.\ \sum_\infty y\in B\ x.\ f\ (x,\ y))$
⟨*proof*⟩

lemma *flip-eq-const*: ‹$(\lambda y.\ y = x) = ((=)\ x)$›
  ⟨*proof*⟩

lemma *sgn-ket*[*simp*]: ‹*sgn* (*ket* x) = *ket* x›
  ⟨*proof*⟩

lemma *tensor-op-in-tensor-vn*:
  assumes ‹$a \in A$› **and** ‹$b \in B$›
  shows ‹$a \otimes_o b \in A \otimes_{vN} B$›
⟨*proof*⟩

lemma *commutant-tensor-vn-subset*:
  assumes ‹*von-neumann-algebra* A› **and** ‹*von-neumann-algebra* B›
  shows ‹*commutant* $A \otimes_{vN}$ *commutant* $B \subseteq$ *commutant* $(A \otimes_{vN} B)$›
⟨*proof*⟩

lemma *commutant-span*[*simp*]: ‹*commutant* (*span* X) = *commutant* X›
⟨*proof*⟩

lemma *explicit-cblinfun-exists-0*[*simp*]: ‹*explicit-cblinfun-exists* (λ- -. *0*)›
  ⟨*proof*⟩

lemma *explicit-cblinfun-0*[*simp*]: ‹*explicit-cblinfun* (λ- -. *0*) = *0*›
  ⟨*proof*⟩

lemma *cnj-of-bool*[*simp*]: ‹*cnj* (*of-bool* *b*) = *of-bool* *b*›
⟨*proof*⟩

lemma *has-sum-single*:
fixes *f* :: ‹- ⇒ -::{*comm-monoid-add*,*t2-space*}›
assumes $\bigwedge j.\ j \neq i \Longrightarrow j{\in}A \Longrightarrow f\ j = 0$
assumes ‹*s* = (*if* *i*∈*A* *then* *f* *i* *else* *0*)›
shows *HAS-SUM* *f* *A* *s*
⟨*proof*⟩

lemma *classical-operator-None*[*simp*]: ‹*classical-operator* (λ-. *None*) = *0*›
⟨*proof*⟩

lemma *has-sum-in-in-closedsubspace*:
assumes ‹*has-sum-in* *T* *f* *A* *l*›
assumes ‹$\bigwedge x.\ x{\in}A \Longrightarrow f\ x \in S$›
assumes ‹*closedin* *T* *S*›
assumes ‹*csubspace* *S*›
shows ‹$l \in S$›
⟨*proof*⟩

lemma *has-sum-coordinatewise*:
‹(*f* *has-sum* *s*) *A* ⟷ (∀ *i*. ((λ*x*. *f* *x* *i*) *has-sum* *s* *i*) *A*)›
⟨*proof*⟩

lemma *one-dim-butterfly*:
‹*butterfly* *g* *h* = (*one-dim-iso* *g* ∗ *cnj* (*one-dim-iso* *h*)) $*_C$ *1*›
⟨*proof*⟩

lemma *one-dim-tc-butterfly*:
fixes *g* :: ‹*'a* :: *one-dim*› **and** *h* :: ‹*'b* :: *one-dim*›
shows ‹*tc-butterfly* *g* *h* = (*one-dim-iso* *g* ∗ *cnj* (*one-dim-iso* *h*)) $*_C$ *1*›
⟨*proof*⟩

lemma *one-dim-iso-of-real*[*simp*]: ‹*one-dim-iso* (*of-real* *x*) = *of-real* *x*›
⟨*proof*⟩

lemma *filter-insert-if*:
‹*Set.filter* *P* (*insert* *x* *M*) = (*if* *P* *x* *then* *insert* *x* (*Set.filter* *P* *M*) *else* *Set.filter* *P* *M*)›
⟨*proof*⟩

lemma *filter-empty*[*simp*]: ‹*Set.filter* *P* {} = {}›
⟨*proof*⟩

lemma *has-sum-in-cong-neutral*:

fixes $f\ g$ :: ‹$'a \Rightarrow 'b$::*comm-monoid-add*›
assumes ‹$\bigwedge x.\ x{\in}T{-}S \Longrightarrow g\ x = 0$›
assumes ‹$\bigwedge x.\ x{\in}S{-}T \Longrightarrow f\ x = 0$›
assumes ‹$\bigwedge x.\ x{\in}S{\cap}T \Longrightarrow f\ x = g\ x$›
shows *has-sum-in* $X\ f\ S\ x \longleftrightarrow$ *has-sum-in* $X\ g\ T\ x$
⟨*proof*⟩

lemma *infsum-in-cong-neutral*:
fixes $f\ g$ :: ‹$'a \Rightarrow 'b$::*comm-monoid-add*›
assumes ‹$\bigwedge x.\ x{\in}T{-}S \Longrightarrow g\ x = 0$›
assumes ‹$\bigwedge x.\ x{\in}S{-}T \Longrightarrow f\ x = 0$›
assumes ‹$\bigwedge x.\ x{\in}S{\cap}T \Longrightarrow f\ x = g\ x$›
shows ‹*infsum-in* $X\ f\ S$ = *infsum-in* $X\ g\ T$›
⟨*proof*⟩

lemma *filter-image*: ‹*Set.filter* P (f ‘ X) = f ‘ (*Set.filter* ($\lambda x.\ P\ (f\ x)$) X)›
⟨*proof*⟩

lemma *Sigma-image-left*: ‹(*SIGMA* x:f‘A. $B\ x$) = ($\lambda(x,y).\ (f\ x,\ y)$) ‘ (*SIGMA* x:A. $B\ (f\ x)$)›
⟨*proof*⟩

lemma *finite-subset-filter-image*:
assumes *finite* B
assumes ‹$B \subseteq$ *Set.filter* P (f ‘ A)›
shows $\exists\, C{\subseteq}A.$ *finite* $C \wedge B = f$ ‘ C
⟨*proof*⟩

definition ‹*card-le-1* $M \longleftrightarrow (\exists\, x.\ M \subseteq \{x\})$›

lemma *card-le-1-empty*[*iff*]: ‹*card-le-1* $\{\}$›
⟨*proof*⟩

lemma *card-le-1-signleton*[*iff*]: ‹*card-le-1* $\{x\}$›
⟨*proof*⟩

lemma *sgn-tensor-ell2*: ‹*sgn* $(h \otimes_s k)$ = *sgn* $h \otimes_s$ *sgn* k›
⟨*proof*⟩

lemma *is-ortho-set-tensor*:
assumes ‹*is-ortho-set* B›
assumes ‹*is-ortho-set* C›
shows ‹*is-ortho-set* $((\lambda(x,\ y).\ x \otimes_s y)$ ‘ $(B \times C))$›
⟨*proof*⟩

end

2 Kraus families

theory *Kraus-Families*
  imports
    Wlog.Wlog
    Hilbert-Space-Tensor-Product.Partial-Trace

    Misc-Kraus-Maps

  abbrevs
    $=kr = =_{kr}$ **and** $==kr = \equiv_{kr}$ **and** $*kr = *_{kr}$
begin

unbundle *cblinfun-syntax*

2.1 Kraus families

definition ‹$\mathit{kraus\text{-}family}\ \mathfrak{E} \longleftrightarrow \mathit{bdd\text{-}above}\ ((\lambda F.\ \sum(E,x){\in}F.\ E{*}\ o_{CL}\ E)\ `\ \{F.\ \mathit{finite}\ F \wedge F \subseteq \mathfrak{E}\}) \wedge 0 \notin \mathit{fst}\ `\ \mathfrak{E}$›
  for $\mathfrak{E}$ :: ‹$((\text{-}{::}\mathit{chilbert\text{-}space} \Rightarrow_{CL} \text{-}{::}\mathit{chilbert\text{-}space}) \times \text{-})\ \mathit{set}$›

typedef (**overloaded**) $('a{::}\mathit{chilbert\text{-}space}, 'b{::}\mathit{chilbert\text{-}space}, 'x)\ \mathit{kraus\text{-}family}$ =
  ‹$\mathit{Collect}\ \mathit{kraus\text{-}family} :: (('a \Rightarrow_{CL} 'b) \times 'x)\ \mathit{set}\ \mathit{set}$›
  ⟨*proof*⟩
setup-lifting *type-definition-kraus-family*

lemma *kraus-familyI*:
  assumes ‹$\mathit{bdd\text{-}above}\ ((\lambda F.\ \sum(E,x){\in}F.\ E{*}\ o_{CL}\ E)\ `\ \{F.\ \mathit{finite}\ F \wedge F \subseteq \mathfrak{E}\})$›
  assumes ‹$0 \notin \mathit{fst}\ `\ \mathfrak{E}$›
  shows ‹$\mathit{kraus\text{-}family}\ \mathfrak{E}$›
  ⟨*proof*⟩

lift-definition *kf-apply* :: ‹$('a{::}\mathit{chilbert\text{-}space}, 'b{::}\mathit{chilbert\text{-}space}, 'x)\ \mathit{kraus\text{-}family} \Rightarrow ('a,'a)\ \mathit{trace\text{-}class} \Rightarrow ('b,'b)\ \mathit{trace\text{-}class}$› **is**
  ‹$\lambda\mathfrak{E}\ \varrho.\ (\sum_\infty E{\in}\mathfrak{E}.\ \mathit{sandwich\text{-}tc}\ (\mathit{fst}\ E)\ \varrho)$› ⟨*proof*⟩
notation *kf-apply* (**infixr** ‹$*_{kr}$› *70*)

lemma *kraus-family-if-finite*[*iff*]: ‹$\mathit{kraus\text{-}family}\ \mathfrak{E}$› **if** ‹$\mathit{finite}\ \mathfrak{E}$› **and** ‹$0 \notin \mathit{fst}\ `\ \mathfrak{E}$›
⟨*proof*⟩

lemma *kf-apply-scaleC*:
  shows ‹$\mathit{kf\text{-}apply}\ \mathfrak{E}\ (c *_C x) = c *_C \mathit{kf\text{-}apply}\ \mathfrak{E}\ x$›
  ⟨*proof*⟩

lemma *kf-apply-abs-summable*:
  assumes ‹$\mathit{kraus\text{-}family}\ \mathfrak{E}$›
  shows ‹$(\lambda(E,x).\ \mathit{sandwich\text{-}tc}\ E\ \varrho)\ \mathit{abs\text{-}summable\text{-}on}\ \mathfrak{E}$›
⟨*proof*⟩

lemma *kf-apply-summable*:
  shows ‹(λ(*E*,*x*). *sandwich-tc E ϱ*) *summable-on* (*Rep-kraus-family* 𝔈)›
  ⟨*proof*⟩

lemma *kf-apply-has-sum*:
  shows ‹((λ(*E*,*x*). *sandwich-tc E ϱ*) *has-sum kf-apply* 𝔈 *ϱ*) (*Rep-kraus-family* 𝔈)›
  ⟨*proof*⟩

lemma *kf-apply-plus-right*:
  shows ‹*kf-apply* 𝔈 (*x* + *y*) = *kf-apply* 𝔈 *x* + *kf-apply* 𝔈 *y*›
  ⟨*proof*⟩

lemma *kf-apply-uminus-right*:
  shows ‹*kf-apply* 𝔈 (− *x*) = − *kf-apply* 𝔈 *x*›
  ⟨*proof*⟩

lemma *kf-apply-minus-right*:
  shows ‹*kf-apply* 𝔈 (*x* − *y*) = *kf-apply* 𝔈 *x* − *kf-apply* 𝔈 *y*›
  ⟨*proof*⟩

lemma *kf-apply-pos*:
  assumes ‹*ϱ* ≥ *0*›
  shows ‹*kf-apply* 𝔈 *ϱ* ≥ *0*›
  ⟨*proof*⟩

lemma *kf-apply-mono-right*:
  assumes ‹*ϱ* ≥ *τ*›
  shows ‹*kf-apply* 𝔈 *ϱ* ≥ *kf-apply* 𝔈 *τ*›
  ⟨*proof*⟩

lemma *kf-apply-geq-sum*:
  assumes ‹*ϱ* ≥ *0*› **and** ‹*M* ⊆ *Rep-kraus-family* 𝔈›
  shows ‹*kf-apply* 𝔈 *ϱ* ≥ (∑(*E*,-)∈*M*. *sandwich-tc E ϱ*)›
⟨*proof*⟩

lift-definition *kf-domain* :: ‹('*a*::*chilbert-space*,'*b*::*chilbert-space*,'*x*) *kraus-family* ⇒ '*x set*› **is**
  ‹λ𝔈. *snd* ‘ 𝔈›⟨*proof*⟩

lemma *kf-apply-clinear*[*iff*]: ‹*clinear* (*kf-apply* 𝔈)›
  ⟨*proof*⟩

lemma *kf-apply-0-right*[*iff*]: ‹*kf-apply* 𝔈 *0* = *0*›
  ⟨*proof*⟩

lift-definition *kf-operators* :: ‹('a::chilbert-space,'b::chilbert-space,'x) kraus-family ⇒ ('a $\Rightarrow_{CL}$ 'b) set› **is**
‹*image fst* :: ('a $\Rightarrow_{CL}$ 'b × 'x) set ⇒ ('a $\Rightarrow_{CL}$ 'b) set›⟨*proof*⟩

2.2 Bound and norm

lift-definition *kf-bound* :: ‹('a::chilbert-space, 'b::chilbert-space, 'x) kraus-family ⇒ ('a $\Rightarrow_{CL}$ 'a)› **is**
‹λ𝔈. *infsum-in cweak-operator-topology* (λ(E,x). E∗ o_{CL} E) 𝔈› ⟨*proof*⟩

lemma *kf-bound-def'*:
‹*kf-bound* 𝔈 = *Rep-cblinfun-wot* ($\sum_\infty$(E,x)∈*Rep-kraus-family* 𝔈. *compose-wot* (*adj-wot* (*Abs-cblinfun-wot* E)) (*Abs-cblinfun-wot* E))›
⟨*proof*⟩

definition ‹*kf-norm* 𝔈 = *norm* (*kf-bound* 𝔈)›

lemma *kf-norm-sum-bdd*: ‹*bdd-above* ((λF. *norm* ($\sum$(E,x)∈F. E∗ o_{CL} E)) ‘ {F. F ⊆ *Rep-kraus-family* 𝔈 ∧ *finite* F})›
⟨*proof*⟩

lemma *kf-norm-geq0*[*iff*]:
shows ‹*kf-norm* 𝔈 ≥ *0*›
⟨*proof*⟩

lemma *kf-bound-has-sum*:
shows ‹*has-sum-in cweak-operator-topology* (λ(E,x). E∗ o_{CL} E) (*Rep-kraus-family* 𝔈) (*kf-bound* 𝔈)›
⟨*proof*⟩

lemma *kraus-family-iff-summable*:
‹*kraus-family* 𝔈 ⟷ *summable-on-in cweak-operator-topology* (λ(E,x). E∗ o_{CL} E) 𝔈 ∧ *0* ∉ *fst* ‘ 𝔈›
⟨*proof*⟩

lemma *kraus-family-iff-summable'*:
‹*kraus-family* 𝔈 ⟷ (λ(E,x). *Abs-cblinfun-wot* (E∗ o_{CL} E)) *summable-on* 𝔈 ∧ *0* ∉ *fst* ‘ 𝔈›
⟨*proof*⟩

lemma *kf-bound-summable*:
shows ‹*summable-on-in cweak-operator-topology* (λ(E,x). E∗ o_{CL} E) (*Rep-kraus-family* 𝔈)›
⟨*proof*⟩

lemma *kf-bound-has-sum'*:
shows ‹((λ(E,x). *compose-wot* (*adj-wot* (*Abs-cblinfun-wot* E)) (*Abs-cblinfun-wot* E)) *has-sum* *Abs-cblinfun-wot* (*kf-bound* 𝔈)) (*Rep-kraus-family* 𝔈)›
⟨*proof*⟩

lemma *kf-bound-summable′*:
‹$((\lambda(E,x).$ *compose-wot* (*adj-wot* (*Abs-cblinfun-wot E*)) (*Abs-cblinfun-wot E*)) *summable-on Rep-kraus-family* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-bound-is-Sup*:
shows ‹*is-Sup* $((\lambda F.\ \sum(E,x)\in F.\ E* \ o_{CL}\ E)$ ‘ $\{F.$ *finite* $F \wedge F \subseteq$ *Rep-kraus-family* $\mathfrak{E}\})$ (*kf-bound* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-bound-leqI*:
assumes ‹$\bigwedge F.$ *finite* $F \Longrightarrow F \subseteq$ *Rep-kraus-family* $\mathfrak{E} \Longrightarrow (\sum(E,x)\in F.\ E* \ o_{CL}\ E) \leq B$›
shows ‹*kf-bound* $\mathfrak{E} \leq B$›
⟨*proof*⟩

lemma *kf-bound-pos*[*iff*]: ‹*kf-bound* $\mathfrak{E} \geq 0$›
⟨*proof*⟩

lemma *not-not-singleton-kf-norm-0*:
fixes $\mathfrak{E}$ :: ‹(*'a*::*chilbert-space*,*'b*::*chilbert-space*,*'x*) *kraus-family*›
assumes ‹¬ *class.not-singleton TYPE*(*'a*)›
shows ‹*kf-norm* $\mathfrak{E} = 0$›
⟨*proof*⟩

lemma *kf-norm-sum-leqI*:
assumes ‹$\bigwedge F.$ *finite* $F \Longrightarrow F \subseteq$ *Rep-kraus-family* $\mathfrak{E} \Longrightarrow$ *norm* $(\sum(E,x)\in F.\ E* \ o_{CL}\ E) \leq B$›
shows ‹*kf-norm* $\mathfrak{E} \leq B$›
⟨*proof*⟩

lemma *kf-bound-geq-sum*:
assumes ‹$M \subseteq$ *Rep-kraus-family* $\mathfrak{E}$›
shows ‹$(\sum(E,\text{-})\in M.\ E* \ o_{CL}\ E) \leq$ *kf-bound* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-norm-geq-norm-sum*:
assumes ‹$M \subseteq$ *Rep-kraus-family* $\mathfrak{E}$›
shows ‹*norm* $(\sum(E,\text{-})\in M.\ E* \ o_{CL}\ E) \leq$ *kf-norm* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-bound-finite*: ‹*kf-bound* $\mathfrak{E} = (\sum(E,x)\in$ *Rep-kraus-family* $\mathfrak{E}.\ E* \ o_{CL}\ E)$› **if** ‹*finite* (*Rep-kraus-family* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-norm-finite*: ‹*kf-norm* $\mathfrak{E} =$ *norm* $(\sum(E,x)\in$ *Rep-kraus-family* $\mathfrak{E}.\ E* \ o_{CL}\ E)$›

if ‹*finite* (*Rep-kraus-family* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-apply-bounded-pos*:
assumes ‹$\varrho \geq 0$›
shows ‹*norm* (*kf-apply* $\mathfrak{E}$ ϱ) $\leq$ *kf-norm* $\mathfrak{E}$ $*$ *norm* ϱ›
⟨*proof*⟩

lemma *kf-apply-bounded*:
— We suspect the factor 4 is not needed here but don't know how to prove that.
‹*norm* (*kf-apply* $\mathfrak{E}$ ϱ) $\leq$ *4* $*$ *kf-norm* $\mathfrak{E}$ $*$ *norm* ϱ›
⟨*proof*⟩

lemma *kf-apply-bounded-clinear*[*bounded-clinear*]:
shows ‹*bounded-clinear* (*kf-apply* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-bound-from-map*: ‹$\psi \bullet_C$ *kf-bound* $\mathfrak{E}$ φ = *trace-tc* ($\mathfrak{E}$ $*_{kr}$ *tc-butterfly* φ ψ)›
⟨*proof*⟩

lemma *trace-from-kf-bound*: ‹*trace-tc* ($\mathfrak{E}$ $*_{kr}$ ϱ) = *trace-tc* (*compose-tcr* (*kf-bound* $\mathfrak{E}$) ϱ)›
⟨*proof*⟩

lemma *kf-bound-selfadjoint*[*iff*]: ‹*selfadjoint* (*kf-bound* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-bound-leq-kf-norm-id*:
shows ‹*kf-bound* $\mathfrak{E}$ $\leq$ *kf-norm* $\mathfrak{E}$ $*_R$ *id-cblinfun*›
⟨*proof*⟩

2.3 Basic Kraus families

lemma *kf-emptyset*[*iff*]: ‹*kraus-family* {}›
⟨*proof*⟩

instantiation *kraus-family* :: (*chilbert-space*, *chilbert-space*, *type*) ‹*zero*› **begin**
lift-definition *zero-kraus-family* :: ‹('*a*,'*b*,'*x*) *kraus-family*› **is** ‹{}›
⟨*proof*⟩
instance⟨*proof*⟩
end

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

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

lemma *kf-norm-0*[*simp*]: ‹*kf-norm 0* = *0*›

⟨*proof*⟩

lift-definition *kf-of-op* :: ‹(*'a*::*chilbert-space* $\Rightarrow_{CL}$ *'b*::*chilbert-space*) ⇒ (*'a*, *'b*, *unit*) *kraus-family*›
is
‹λ*E*::*'a*$\Rightarrow_{CL}$*'b*. *if E* = *0 then* {} *else* {(*E*, ())}›
⟨*proof*⟩

lemma *kf-of-op0*[*simp*]: ‹*kf-of-op 0* = *0*›
⟨*proof*⟩

lemma *kf-of-op-norm*[*simp*]: ‹*kf-norm* (*kf-of-op E*) = (*norm E*)^2›
⟨*proof*⟩

lemma *kf-operators-in-kf-of-op*[*simp*]: ‹*kf-operators* (*kf-of-op U*) = (*if U* = *0 then* {} *else* {*U*})›
⟨*proof*⟩

lemma *kf-domain-of-op*[*simp*]: ‹*kf-domain* (*kf-of-op A*) = {()}› **if** ‹*A* ≠ *0*›
⟨*proof*⟩

definition ‹*kf-id* = *kf-of-op id-cblinfun*›

lemma *kf-domain-id*[*simp*]: ‹*kf-domain* (*kf-id* :: (*'a*::{*chilbert-space*,*not-singleton*},-,-) *kraus-family*) = {()}›
⟨*proof*⟩

lemma *kf-of-op-id*[*simp*]: ‹*kf-of-op id-cblinfun* = *kf-id*›
⟨*proof*⟩

lemma *kf-norm-id-leq1*: ‹*kf-norm kf-id* ≤ *1*›
⟨*proof*⟩

lemma *kf-norm-id-eq1*[*simp*]: ‹*kf-norm* (*kf-id* :: (*'a* :: {*chilbert-space*, *not-singleton*},*'a*,*unit*) *kraus-family*) = *1*›
⟨*proof*⟩

lemma *kf-operators-in-kf-id*[*simp*]: ‹*kf-operators kf-id* = (*if* (*id-cblinfun*::*'a*::*chilbert-space*$\Rightarrow_{CL}$-)=*0* *then* {} *else* {*id-cblinfun*::*'a*$\Rightarrow_{CL}$-})›
⟨*proof*⟩

instantiation *kraus-family* :: (*chilbert-space*, *chilbert-space*, *type*) *scaleR* **begin**
lift-definition *scaleR-kraus-family* :: ‹*real* ⇒ (*'a*::*chilbert-space*,*'b*::*chilbert-space*,*'x*) *kraus-family* ⇒ (*'a*,*'b*,*'x*) *kraus-family*› **is**
‹λ*r* 𝔈. *if r* ≤ *0 then* {} *else* (λ(*E*,*x*). (*sqrt r* $*_R$ *E*, *x*)) ‘ 𝔈›
⟨*proof*⟩
instance⟨*proof*⟩
end

lemma *kf-scale-apply*:

assumes ‹$r \geq 0$›
shows ‹*kf-apply* ($r *_R$ $\mathfrak{E}$) ϱ = $r *_R$ *kf-apply* $\mathfrak{E}$ ϱ›
⟨*proof*⟩

lemma *kf-bound-scale*[*simp*]: ‹*kf-bound* ($c *_R$ $\mathfrak{E}$) = $c *_R$ *kf-bound* $\mathfrak{E}$› **if** ‹$c \geq 0$›
⟨*proof*⟩

lemma *kf-norm-scale*[*simp*]: ‹*kf-norm* ($c *_R$ $\mathfrak{E}$) = c * *kf-norm* $\mathfrak{E}$› **if** ‹$c \geq 0$›
⟨*proof*⟩

lemma *kf-of-op-apply*: ‹*kf-apply* (*kf-of-op* E) ϱ = *sandwich-tc* E ϱ›
⟨*proof*⟩

lemma *kf-id-apply*[*simp*]: ‹*kf-apply kf-id* ϱ = ϱ›
⟨*proof*⟩

lemma *kf-scale-apply-neg*:
assumes ‹$r \leq 0$›
shows ‹*kf-apply* ($r *_R$ $\mathfrak{E}$) = *0*›
⟨*proof*⟩

lemma *kf-apply-0-left*[*iff*]: ‹*kf-apply 0* ϱ = *0*›
⟨*proof*⟩

lemma *kf-bound-of-op*[*simp*]: ‹*kf-bound* (*kf-of-op* A) = $A*$ o_{CL} A›
⟨*proof*⟩

lemma *kf-bound-id*[*simp*]: ‹*kf-bound kf-id* = *id-cblinfun*›
⟨*proof*⟩

2.4 Filtering

lift-definition *kf-filter* :: ‹($'x \Rightarrow$ *bool*) $\Rightarrow$ ($'a$::*chilbert-space*, $'b$::*chilbert-space*, $'x$) *kraus-family* $\Rightarrow$ ($'a$::*chilbert-space*, $'b$::*chilbert-space*, $'x$) *kraus-family*› **is**
‹λ(P::$'x \Rightarrow$ *bool*) $\mathfrak{E}$. *Set.filter* (λ(E,x). P x) $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-filter-false*[*simp*]: ‹*kf-filter* (λ-. *False*) $\mathfrak{E}$ = *0*›
⟨*proof*⟩

lemma *kf-filter-true*[*simp*]: ‹*kf-filter* (λ-. *True*) $\mathfrak{E}$ = $\mathfrak{E}$›
⟨*proof*⟩

definition ‹*kf-apply-on* $\mathfrak{E}$ S = *kf-apply* (*kf-filter* (λx. $x \in S$) $\mathfrak{E}$)›
notation *kf-apply-on* (- $*_{kr}$ @-/ - [*71*, *1000*, *70*] *70*)

lemma *kf-apply-on-pos*:
assumes ‹$\varrho \geq 0$›
shows ‹*kf-apply-on* $\mathfrak{E}$ X $\varrho \geq 0$›

$\langle proof \rangle$

lemma *kf-apply-on-bounded-clinear*[*bounded-clinear*]:
shows ‹*bounded-clinear* (*kf-apply-on* $\mathfrak{E}$ *X*)›
$\langle proof \rangle$

lemma *kf-filter-twice*:
‹*kf-filter P* (*kf-filter Q* $\mathfrak{E}$) = *kf-filter* ($\lambda x.\ P\ x \wedge Q\ x$) $\mathfrak{E}$›
$\langle proof \rangle$

lemma *kf-apply-on-union-has-sum*:
assumes ‹$\bigwedge X\ Y.\ X{\in}F \Longrightarrow Y{\in}F \Longrightarrow X{\neq}Y \Longrightarrow$ *disjnt X Y*›
shows ‹(($\lambda X.$ *kf-apply-on* $\mathfrak{E}$ $X\ \varrho$) *has-sum* (*kf-apply-on* $\mathfrak{E}$ ($\bigcup F$) ϱ)) *F*›
$\langle proof \rangle$

lemma *kf-apply-on-empty*[*simp*]: ‹*kf-apply-on E* {} ϱ = *0*›
$\langle proof \rangle$

lemma *kf-apply-on-union-eqI*:
assumes ‹$\bigwedge X\ Y.\ (X,Y){\in}F \Longrightarrow$ *kf-apply-on* $\mathfrak{E}$ $X\ \varrho$ = *kf-apply-on* $\mathfrak{F}$ $Y\ \varrho$›
assumes ‹$\bigwedge X\ Y\ X'\ Y'.\ (X,Y){\in}F \Longrightarrow (X',Y'){\in}F \Longrightarrow (X,Y){\neq}(X',Y') \Longrightarrow$ *disjnt X X'*›
assumes ‹$\bigwedge X\ Y\ X'\ Y'.\ (X,Y){\in}F \Longrightarrow (X',Y'){\in}F \Longrightarrow (X,Y){\neq}(X',Y') \Longrightarrow$ *disjnt Y Y'*›
assumes *XX*: ‹$XX = \bigcup$(*fst* ‘ *F*)› **and** *YY*: ‹$YY = \bigcup$(*snd* ‘ *F*)›
shows ‹*kf-apply-on* $\mathfrak{E}$ *XX* ϱ = *kf-apply-on* $\mathfrak{F}$ *YY* ϱ›
$\langle proof \rangle$

lemma *kf-apply-on-UNIV*[*simp*]: ‹*kf-apply-on* $\mathfrak{E}$ *UNIV* = *kf-apply* $\mathfrak{E}$›
$\langle proof \rangle$

lemma *kf-apply-on-CARD-1*[*simp*]: ‹($\lambda\mathfrak{E}.$ *kf-apply-on* $\mathfrak{E}$ {*x*::-::*CARD-1*}) = *kf-apply*›
$\langle proof \rangle$

lemma *kf-apply-on-union-summable-on*:
assumes ‹$\bigwedge X\ Y.\ X{\in}F \Longrightarrow Y{\in}F \Longrightarrow X{\neq}Y \Longrightarrow$ *disjnt X Y*›
shows ‹($\lambda X.$ *kf-apply-on* $\mathfrak{E}$ $X\ \varrho$) *summable-on F*›
$\langle proof \rangle$

lemma *kf-apply-on-union-infsum*:
assumes ‹$\bigwedge X\ Y.\ X{\in}F \Longrightarrow Y{\in}F \Longrightarrow X{\neq}Y \Longrightarrow$ *disjnt X Y*›
shows ‹($\sum_\infty X{\in}F.$ *kf-apply-on* $\mathfrak{E}$ $X\ \varrho$) = *kf-apply-on* $\mathfrak{E}$ ($\bigcup F$) ϱ›
$\langle proof \rangle$

lemma *kf-bound-filter*:
‹*kf-bound* (*kf-filter P* $\mathfrak{E}$) $\leq$ *kf-bound* $\mathfrak{E}$›
$\langle proof \rangle$

lemma *kf-norm-filter*:
‹*kf-norm* (*kf-filter* P $\mathfrak{E}$) $\leq$ *kf-norm* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-domain-filter*[*simp*]:
‹*kf-domain* (*kf-filter* P E) = *Set.filter* P (*kf-domain* E)›
⟨*proof*⟩

lemma *kf-filter-0-right*[*simp*]: ‹*kf-filter* P *0* = *0*›
⟨*proof*⟩

lemma *kf-apply-on-0-right*[*simp*]: ‹*kf-apply-on* $\mathfrak{E}$ X *0* = *0*›
⟨*proof*⟩

lemma *kf-apply-on-0-left*[*simp*]: ‹*kf-apply-on* *0* X ϱ = *0*›
⟨*proof*⟩

lemma *kf-apply-on-mono3*:
assumes ‹$\varrho \leq \sigma$›
shows ‹$\mathfrak{E} *_{kr} @X\ \varrho \leq \mathfrak{E} *_{kr} @X\ \sigma$›
⟨*proof*⟩

lemma *kf-apply-on-mono2*:
assumes ‹$X \subseteq Y$› **and** ‹$\varrho \geq$ *0*›
shows ‹$\mathfrak{E} *_{kr} @X\ \varrho \leq \mathfrak{E} *_{kr} @Y\ \varrho$›
⟨*proof*⟩

lemma *kf-operators-filter*: ‹*kf-operators* (*kf-filter* P $\mathfrak{E}$) $\subseteq$ *kf-operators* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-equal-if-filter-equal*:
assumes ‹$\bigwedge x$. *kf-filter* ((=)x) $\mathfrak{E}$ = *kf-filter* ((=)x) $\mathfrak{F}$›
shows ‹$\mathfrak{E} = \mathfrak{F}$›
⟨*proof*⟩

2.5 Equivalence

definition ‹*kf-eq-weak* $\mathfrak{E}$ $\mathfrak{F}$ $\longleftrightarrow$ *kf-apply* $\mathfrak{E}$ = *kf-apply* $\mathfrak{F}$›
definition ‹*kf-eq* $\mathfrak{E}$ $\mathfrak{F}$ $\longleftrightarrow$ ($\forall\, x$. *kf-apply-on* $\mathfrak{E}$ $\{x\}$ = *kf-apply-on* $\mathfrak{F}$ $\{x\}$)›

open-bundle *kraus-map-syntax* **begin**
notation *kf-eq-weak* (**infix** $=_{kr}$ *50*)
notation *kf-eq* (**infix** $\equiv_{kr}$ *50*)
notation *kf-apply* (**infixr** ‹$*_{kr}$› *70*)
notation *kf-apply-on* (- $*_{kr}$ @-/ - [*71*, *1000*, *70*] *70*)
end

lemma *kf-eq-weak-reflI*[*iff*]: ‹$x =_{kr}\ x$›
⟨*proof*⟩

lemma *kf-eq-weak-refl0*[*iff*]: ‹$0 =_{kr}\ 0$›
⟨*proof*⟩

lemma *kf-bound-cong*:
assumes ‹$\mathfrak{E} =_{kr}\ \mathfrak{F}$›
shows ‹*kf-bound* $\mathfrak{E}$ = *kf-bound* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-norm-cong*:
assumes ‹$\mathfrak{E} =_{kr}\ \mathfrak{F}$›
shows ‹*kf-norm* $\mathfrak{E}$ = *kf-norm* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-eq-weakI*:
assumes ‹$\bigwedge\varrho.\ \varrho \geq 0 \Longrightarrow \mathfrak{E} *_{kr}\ \varrho = \mathfrak{F} *_{kr}\ \varrho$›
shows ‹$\mathfrak{E} =_{kr}\ \mathfrak{F}$›
⟨*proof*⟩

lemma *kf-eqI*:
assumes ‹$\bigwedge x\ \varrho.\ \varrho \geq 0 \Longrightarrow \mathfrak{E} *_{kr}\ @\{x\}\ \varrho = \mathfrak{F} *_{kr}\ @\{x\}\ \varrho$›
shows ‹$\mathfrak{E} \equiv_{kr}\ \mathfrak{F}$›
⟨*proof*⟩

lemma *kf-eq-weak-trans*[*trans*]:
‹$F =_{kr}\ G \Longrightarrow G =_{kr}\ H \Longrightarrow F =_{kr}\ H$›
⟨*proof*⟩

lemma *kf-apply-eqI*:
assumes ‹$\mathfrak{E} =_{kr}\ \mathfrak{F}$›
shows ‹$\mathfrak{E} *_{kr}\ \varrho = \mathfrak{F} *_{kr}\ \varrho$›
⟨*proof*⟩

lemma *kf-eq-imp-eq-weak*:
assumes ‹$\mathfrak{E} \equiv_{kr}\ \mathfrak{F}$›
shows ‹$\mathfrak{E} =_{kr}\ \mathfrak{F}$›
⟨*proof*⟩

lemma *kf-filter-cong-eq*[*cong*]:
assumes ‹$\mathfrak{E} = \mathfrak{F}$›
assumes ‹$\bigwedge x.\ x \in$ *kf-domain* $\mathfrak{E} \Longrightarrow P\ x = Q\ x$›
shows ‹*kf-filter* $P\ \mathfrak{E}$ = *kf-filter* $Q\ \mathfrak{F}$›
⟨*proof*⟩

lemma *kf-filter-cong*:
  assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
  assumes ‹$\bigwedge x.\ x \in$ *kf-domain* $\mathfrak{E} \Longrightarrow P\ x = Q\ x$›
  shows ‹*kf-filter* $P\ \mathfrak{E} \equiv_{kr}$ *kf-filter* $Q\ \mathfrak{F}$›
⟨*proof*⟩

lemma *kf-filter-cong-weak*:
  assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
  assumes ‹$\bigwedge x.\ x \in$ *kf-domain* $\mathfrak{E} \Longrightarrow P\ x = Q\ x$›
  shows ‹*kf-filter* $P\ \mathfrak{E} =_{kr}$ *kf-filter* $Q\ \mathfrak{F}$›
  ⟨*proof*⟩

lemma *kf-eq-refl*[*iff*]: ‹$\mathfrak{E} \equiv_{kr} \mathfrak{E}$›
  ⟨*proof*⟩

lemma *kf-eq-trans*[*trans*]:
  assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
  assumes ‹$\mathfrak{F} \equiv_{kr} \mathfrak{G}$›
  shows ‹$\mathfrak{E} \equiv_{kr} \mathfrak{G}$›
  ⟨*proof*⟩

lemma *kf-eq-sym*[*sym*]:
  assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
  shows ‹$\mathfrak{F} \equiv_{kr} \mathfrak{E}$›
  ⟨*proof*⟩

lemma *kf-eq-weak-imp-eq-CARD-1*:
  fixes $\mathfrak{E}\ \mathfrak{F}$ :: ‹(*'a*::*chilbert-space*, *'b*::*chilbert-space*, *'x*::*CARD-1*) *kraus-family*›
  assumes ‹$\mathfrak{E} =_{kr} \mathfrak{F}$›
  shows ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
  ⟨*proof*⟩

lemma *kf-apply-on-eqI-filter*:
  assumes ‹*kf-filter* $(\lambda x.\ x \in X)\ \mathfrak{E} \equiv_{kr}$ *kf-filter* $(\lambda x.\ x \in X)\ \mathfrak{F}$›
  shows ‹$\mathfrak{E} *_{kr} @X\ \varrho = \mathfrak{F} *_{kr} @X\ \varrho$›
⟨*proof*⟩

lemma *kf-apply-on-eqI*:
  assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
  shows ‹$\mathfrak{E} *_{kr} @X\ \varrho = \mathfrak{F} *_{kr} @X\ \varrho$›
  ⟨*proof*⟩

lemma *kf-apply-eq0I*:
  assumes ‹$\mathfrak{E} =_{kr}$ *0*›
  shows ‹$\mathfrak{E} *_{kr} \varrho =$ *0*›
  ⟨*proof*⟩

lemma *kf-eq-weak0-imp-kf-eq0*:

assumes ‹$\mathfrak{E} =_{kr}$ *0*›
shows ‹$\mathfrak{E} \equiv_{kr}$ *0*›
⟨*proof*⟩

lemma *kf-apply-on-eq0I*:
assumes ‹$\mathfrak{E} =_{kr}$ *0*›
shows ‹$\mathfrak{E} *_{kr}$ @X ϱ = *0*›
⟨*proof*⟩

lemma *kf-filter-to-domain*[*simp*]:
‹*kf-filter* (λx. $x \in$ *kf-domain* $\mathfrak{E}$) $\mathfrak{E} = \mathfrak{E}$›
⟨*proof*⟩

lemma *kf-eq-0-iff-eq-0*: ‹$E =_{kr}$ *0* $\longleftrightarrow$ E = *0*›
⟨*proof*⟩

lemma *in-kf-domain-iff-apply-nonzero*:
‹$x \in$ *kf-domain* $\mathfrak{E}$ $\longleftrightarrow$ *kf-apply-on* $\mathfrak{E}$ $\{x\} \neq$ *0*›
⟨*proof*⟩

lemma *kf-domain-cong*:
assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
shows ‹*kf-domain* $\mathfrak{E}$ = *kf-domain* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-eq-weak-sym*[*sym*]:
assumes ‹$\mathfrak{E} =_{kr} \mathfrak{F}$›
shows ‹$\mathfrak{F} =_{kr} \mathfrak{E}$›
⟨*proof*⟩

lemma *kf-eqI-from-filter-eq-weak*:
assumes ‹$\bigwedge x$. *kf-filter* ((=) x) $E =_{kr}$ *kf-filter* ((=) x) F›
shows ‹$E \equiv_{kr} F$›
⟨*proof*⟩

lemma *kf-eq-weak-from-separatingI*:
fixes E :: ‹('*q*::*chilbert-space*,'*r*::*chilbert-space*,'*x*) *kraus-family*›
and F :: ‹('*q*,'*r*,'*y*) *kraus-family*›
assumes ‹*separating-set* (*bounded-clinear* :: (('*q*,'*q*) *trace-class* ⇒ ('*r*,'*r*) *trace-class*) ⇒ *bool*) S›
assumes ‹$\bigwedge\varrho$. $\varrho \in S \Longrightarrow E *_{kr} \varrho = F *_{kr} \varrho$›
shows ‹$E =_{kr} F$›
⟨*proof*⟩

lemma *kf-eq-weak-eq-trans*[*trans*]: ‹$a =_{kr} b \Longrightarrow b \equiv_{kr} c \Longrightarrow a =_{kr} c$›

⟨*proof*⟩

lemma *kf-eq-eq-weak-trans*[*trans*]: ‹$a \equiv_{kr} b \Longrightarrow b =_{kr} c \Longrightarrow a =_{kr} c$›
⟨*proof*⟩

instantiation *kraus-family* :: (*chilbert-space*,*chilbert-space*,*type*) *preorder* **begin**
definition *less-eq-kraus-family* **where** ‹$\mathfrak{E} \le \mathfrak{F} \longleftrightarrow (\forall x.\ \forall \varrho{\ge}0.$ *kf-apply-on* $\mathfrak{E}$ $\{x\}$ $\varrho \le$ *kf-apply-on* $\mathfrak{F}$ $\{x\}$ $\varrho)$›
definition *less-kraus-family* **where** ‹$\mathfrak{E} < \mathfrak{F} \longleftrightarrow \mathfrak{E} \le \mathfrak{F} \wedge \neg\ \mathfrak{E} \equiv_{kr} \mathfrak{F}$›
lemma *kf-antisym*: ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F} \longleftrightarrow \mathfrak{E} \le \mathfrak{F} \wedge \mathfrak{F} \le \mathfrak{E}$›
for $\mathfrak{E}$ $\mathfrak{F}$ :: ‹('*a*, '*b*, '*c*) *kraus-family*›
⟨*proof*⟩
instance
⟨*proof*⟩
end

lemma *kf-apply-on-mono1*:
assumes ‹$\mathfrak{E} \le \mathfrak{F}$› **and** ‹$\varrho \ge 0$›
shows ‹$\mathfrak{E} *_{kr} @X\ \varrho \le \mathfrak{F} *_{kr} @X\ \varrho$›
⟨*proof*⟩

lemma *kf-apply-mono-left*: ‹$\mathfrak{E} \le \mathfrak{F} \Longrightarrow \varrho \ge 0 \Longrightarrow \mathfrak{E} *_{kr} \varrho \le \mathfrak{F} *_{kr} \varrho$›
⟨*proof*⟩

lemma *kf-apply-mono*:
assumes ‹$\varrho \ge 0$›
assumes ‹$\mathfrak{E} \le \mathfrak{F}$› **and** ‹$\varrho \le \sigma$›
shows ‹$\mathfrak{E} *_{kr} \varrho \le \mathfrak{F} *_{kr} \sigma$›
⟨*proof*⟩

lemma *kf-apply-on-mono*:
assumes ‹$\varrho \ge 0$›
assumes ‹$\mathfrak{E} \le \mathfrak{F}$› **and** ‹$X \subseteq Y$› **and** ‹$\varrho \le \sigma$›
shows ‹$\mathfrak{E} *_{kr} @X\ \varrho \le \mathfrak{F} *_{kr} @Y\ \sigma$›
⟨*proof*⟩

lemma *kf-one-dim-is-id*[*simp*]:
fixes $\mathfrak{E}$ :: ‹('*a*::*one-dim*, '*a*::*one-dim*, '*x*) *kraus-family*›
shows ‹$\mathfrak{E} =_{kr}$ *kf-norm* $\mathfrak{E} *_R$ *kf-id*›
⟨*proof*⟩

2.6 Mapping and flattening

definition *kf-similar-elements* :: ‹('*a*::*chilbert-space*, '*b*::*chilbert-space*, '*x*) *kraus-family* ⇒ ('*a* $\Rightarrow_{CL}$ '*b*) ⇒ ('*a* $\Rightarrow_{CL}$ '*b* × '*x*) *set*› **where**
— All elements of the Kraus family that are equal up to rescaling (and belong to the same output)
‹*kf-similar-elements* $\mathfrak{E}$ $E = \{(F,x) \in$ *Rep-kraus-family* $\mathfrak{E}.\ (\exists r{>}0.\ E = r *_R F)\}$›

definition *kf-element-weight* **where**
— The total weight (norm of the square) of all similar elements
‹*kf-element-weight* $\mathfrak{E}$ $E = (\sum_\infty (F,\text{-}) \in$ *kf-similar-elements* $\mathfrak{E}$ E. *norm* $(F* \, o_{CL} \, F))$›

lemma *kf-element-weight-geq0*[*simp*]: ‹*kf-element-weight* $\mathfrak{E}$ $E \geq 0$›
⟨*proof*⟩

lemma *kf-similar-elements-abs-summable*:
fixes $\mathfrak{E}$ :: ‹(*'a*::*chilbert-space*,*'b*::*chilbert-space*,*'x*) *kraus-family*›
shows ‹$(\lambda(F,\text{-}).\ F* \, o_{CL} \, F)$ *abs-summable-on* (*kf-similar-elements* $\mathfrak{E}$ E)›
⟨*proof*⟩

lemma *kf-similar-elements-kf-operators*:
assumes ‹$(F,x) \in$ *kf-similar-elements* $\mathfrak{E}$ E›
shows ‹$F \in$ *span* (*kf-operators* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-element-weight-neq0*: ‹*kf-element-weight* $\mathfrak{E}$ $E \neq 0$›
if ‹$(E,x) \in$ *Rep-kraus-family* $\mathfrak{E}$› **and** ‹$E \neq 0$›
⟨*proof*⟩

lemma *kf-element-weight-0-left*[*simp*]: ‹*kf-element-weight* *0* $E = 0$›
⟨*proof*⟩

lemma *kf-element-weight-0-right*[*simp*]: ‹*kf-element-weight* E *0* $= 0$›
⟨*proof*⟩

lemma *kf-element-weight-scale*:
assumes ‹$r > 0$›
shows ‹*kf-element-weight* $\mathfrak{E}$ $(r *_R E) =$ *kf-element-weight* $\mathfrak{E}$ E›
⟨*proof*⟩

lemma *kf-element-weight-kf-operators*:
assumes ‹*kf-element-weight* $\mathfrak{E}$ $E \neq 0$›
shows ‹$E \in$ *span* (*kf-operators* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-map-aux*:
fixes f :: ‹*'x* $\Rightarrow$ *'y*› **and** $\mathfrak{E}$ :: ‹(*'a*::*chilbert-space*, *'b*::*chilbert-space*, *'x*) *kraus-family*›
defines ‹$B \equiv$ *kf-bound* $\mathfrak{E}$›
defines ‹*filtered* $y \equiv$ *kf-filter* $(\lambda x.\ f\ x = y)$ $\mathfrak{E}$›
defines ‹*flattened* $\equiv \{(E,\ y).$ *norm* $(E* \, o_{CL} \, E) =$ *kf-element-weight* (*filtered* y) $E \wedge E \neq 0\}$›
defines ‹*good* $\equiv (\lambda(E,y).$ (*norm* $E)^2 =$ *kf-element-weight* (*filtered* y) $E \wedge E \neq 0)$›
shows ‹*has-sum-in cweak-operator-topology* $(\lambda(E,\text{-}).\ E* \, o_{CL} \, E)$ *flattened* B› (**is** *?has-sum*)
and ‹*snd* ‘ (*SIGMA* $(E,\ y)$:*Collect good*. *kf-similar-elements* (*filtered* y) E)
$= \{(F,\ x).\ (F,\ x) \in$ *Rep-kraus-family* $\mathfrak{E} \wedge F \neq 0\}$› (**is** *?snd-sigma*)
and ‹*inj-on snd* (*SIGMA* p:*Collect good*. *kf-similar-elements* (*filtered* (*snd* p)) (*fst* p))› (**is**

?inj-snd)
⟨*proof*⟩

lift-definition *kf-map* :: ‹(*'x* ⇒ *'y*) ⇒ (*'a*::*chilbert-space*, *'b*::*chilbert-space*, *'x*) *kraus-family* ⇒ (*'a*, *'b*, *'y*) *kraus-family*› **is**
‹λ*f* 𝔈. {(*E*, *y*). *norm* (*E*∗ o_{CL} *E*) = *kf-element-weight* (*kf-filter* (λ*x*. *f x* = *y*) 𝔈) *E* ∧ *E* ≠ *0*}›
⟨*proof*⟩

lemma
fixes 𝔈 :: ‹(*'a*::*chilbert-space*,*'b*::*chilbert-space*,*'x*) *kraus-family*›
shows *kf-apply-map*[*simp*]: ‹*kf-apply* (*kf-map f* 𝔈) = *kf-apply* 𝔈›
and *kf-map-bound*: ‹*kf-bound* (*kf-map f* 𝔈) = *kf-bound* 𝔈›
and *kf-map-norm*[*simp*]: ‹*kf-norm* (*kf-map f* 𝔈) = *kf-norm* 𝔈›
⟨*proof*⟩

abbreviation ‹*kf-flatten* ≡ *kf-map* (λ-. ())›

Like *kf-map*, but with a much simpler definition. However, only makes sense for injective functions.

lift-definition *kf-map-inj* :: ‹(*'x* ⇒ *'y*) ⇒ (*'a*::*chilbert-space*, *'b*::*chilbert-space*, *'x*) *kraus-family* ⇒ (*'a*, *'b*, *'y*) *kraus-family*› **is**
‹λ*f* 𝔈. (λ(*E*,*x*). (*E*, *f x*)) ‘ 𝔈›
⟨*proof*⟩

lemma *kf-element-weight-map-inj*:
assumes ‹*inj-on f* (*kf-domain* 𝔈)›
shows ‹*kf-element-weight* (*kf-map-inj f* 𝔈) *E* = *kf-element-weight* 𝔈 *E*›
⟨*proof*⟩

lemma *kf-eq-weak-kf-map-left*: ‹*kf-map f F* $=_{kr}$ *G*› **if** ‹*F* $=_{kr}$ *G*›
⟨*proof*⟩

lemma *kf-eq-weak-kf-map-right*: ‹*F* $=_{kr}$ *kf-map f G*› **if** ‹*F* $=_{kr}$ *G*›
⟨*proof*⟩

lemma *kf-filter-map*:
fixes 𝔈 :: ‹(*'a*::*chilbert-space*,*'b*::*chilbert-space*,*'x*) *kraus-family*›
shows ‹*kf-filter P* (*kf-map f* 𝔈) = *kf-map f* (*kf-filter* (λ*x*. *P* (*f x*)) 𝔈)›
⟨*proof*⟩

lemma *kf-filter-map-inj*:
fixes 𝔈 :: ‹(*'a*::*chilbert-space*,*'b*::*chilbert-space*,*'x*) *kraus-family*›
shows ‹*kf-filter P* (*kf-map-inj f* 𝔈) = *kf-map-inj f* (*kf-filter* (λ*x*. *P* (*f x*)) 𝔈)›
⟨*proof*⟩

lemma *kf-map-kf-map-inj-comp*:
assumes ‹*inj-on f* (*kf-domain* 𝔈)›

shows ‹*kf-map g* (*kf-map-inj f* 𝔈) = *kf-map* (*g o f*) 𝔈›
⟨*proof*⟩

lemma *kf-element-weight-eqweak0*:
assumes ‹𝔈 $=_{kr}$ *0*›
shows ‹*kf-element-weight* 𝔈 *E* = *0*›
⟨*proof*⟩

lemma *kf-map-inj-kf-map-comp*:
assumes ‹*inj-on g* (*f* ‘ *kf-domain* 𝔈)›
shows ‹*kf-map-inj g* (*kf-map f* 𝔈) = *kf-map* (*g o f*) 𝔈›
⟨*proof*⟩

lemma *kf-apply-map-inj*[*simp*]:
assumes ‹*inj-on f* (*kf-domain* 𝔈)›
shows ‹*kf-map-inj f* 𝔈 $*_{kr}$ ϱ = 𝔈 $*_{kr}$ ϱ›
⟨*proof*⟩

lemma *kf-map-inj-eq-kf-map*:
assumes ‹*inj-on f* (*kf-domain* 𝔈)›
shows ‹*kf-map-inj f* 𝔈 $\equiv_{kr}$ *kf-map f* 𝔈›
⟨*proof*⟩

lemma *kf-map-inj-id*[*simp*]: ‹*kf-map-inj id* 𝔈 = 𝔈›
⟨*proof*⟩

lemma *kf-map-id*: ‹*kf-map id* 𝔈 $\equiv_{kr}$ 𝔈›
⟨*proof*⟩

lemma *kf-map-inj-bound*[*simp*]:
fixes 𝔈 :: ‹(*'a::chilbert-space,'b::chilbert-space,'x*) *kraus-family*›
assumes ‹*inj-on f* (*kf-domain* 𝔈)›
shows ‹*kf-bound* (*kf-map-inj f* 𝔈) = *kf-bound* 𝔈›
⟨*proof*⟩

lemma *kf-map-inj-norm*[*simp*]:
fixes 𝔈 :: ‹(*'a::chilbert-space,'b::chilbert-space,'x*) *kraus-family*›
assumes ‹*inj-on f* (*kf-domain* 𝔈)›
shows ‹*kf-norm* (*kf-map-inj f* 𝔈) = *kf-norm* 𝔈›
⟨*proof*⟩

lemma *kf-map-cong-weak*:
assumes ‹𝔈 $=_{kr}$ 𝔉›
shows ‹*kf-map f* 𝔈 $=_{kr}$ *kf-map g* 𝔉›
⟨*proof*⟩

lemma *kf-flatten-cong-weak*:

assumes ‹$\mathfrak{E} =_{kr} \mathfrak{F}$›
shows ‹*kf-flatten* $\mathfrak{E} =_{kr}$ *kf-flatten* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-flatten-cong*:
assumes ‹$\mathfrak{E} =_{kr} \mathfrak{F}$›
shows ‹*kf-flatten* $\mathfrak{E} \equiv_{kr}$ *kf-flatten* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-map-twice*:
‹*kf-map* f (*kf-map* g $\mathfrak{E}$) $\equiv_{kr}$ *kf-map* $(f \circ g)$ $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-map-cong*:
assumes ‹$\bigwedge x.\ x \in$ *kf-domain* $\mathfrak{E} \Longrightarrow f\ x = g\ x$›
assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{F}$›
shows ‹*kf-map* f $\mathfrak{E} \equiv_{kr}$ *kf-map* g $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-map-inj-cong-eq*:
assumes ‹$\bigwedge x.\ x \in$ *kf-domain* $\mathfrak{E} \Longrightarrow f\ x = g\ x$›
assumes ‹$\mathfrak{E} = \mathfrak{F}$›
shows ‹*kf-map-inj* f $\mathfrak{E}$ = *kf-map-inj* g $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-domain-map*[*simp*]:
‹*kf-domain* (*kf-map* f $\mathfrak{E}$) = f ‘ *kf-domain* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-apply-on-map*[*simp*]:
‹(*kf-map* f E) $*_{kr}$ @X ϱ = E $*_{kr}$ @(f −‘ X) ϱ›
⟨*proof*⟩

lemma *kf-apply-on-map-inj*[*simp*]:
assumes ‹*inj-on* f ((f −‘ X) ∩ *kf-domain* E)›
shows ‹*kf-map-inj* f E $*_{kr}$ @X ϱ = E $*_{kr}$ @(f −‘ X) ϱ›
⟨*proof*⟩

lemma *kf-map0*[*simp*]: ‹*kf-map* f *0* = *0*›
⟨*proof*⟩

lemma *kf-map-inj-kr-eq-weak*:
assumes ‹*inj-on* f (*kf-domain* $\mathfrak{E}$)›
shows ‹*kf-map-inj* f $\mathfrak{E} =_{kr} \mathfrak{E}$›

⟨*proof*⟩

lemma *kf-map-inj-0*[*simp*]: ‹*kf-map-inj f 0 = 0*›
⟨*proof*⟩

lemma *kf-domain-map-inj*[*simp*]: ‹*kf-domain* (*kf-map-inj f* 𝔈) = *f* ‘ *kf-domain* 𝔈›
⟨*proof*⟩

lemma *kf-operators-kf-map*:
‹*kf-operators* (*kf-map f* 𝔈) ⊆ *span* (*kf-operators* 𝔈)›
⟨*proof*⟩

lemma *kf-operators-kf-map-inj*[*simp*]: ‹*kf-operators* (*kf-map-inj f* 𝔈) = *kf-operators* 𝔈›
⟨*proof*⟩

2.7 Addition

lift-definition *kf-plus* :: ‹(*'a*::*chilbert-space*, *'b*::*chilbert-space*, *'x*) *kraus-family* ⇒ (*'a*,*'b*,*'y*) *kraus-family* ⇒ (*'a*,*'b*,*'x*+*'y*) *kraus-family*› **is**
‹λ𝔈 𝔉. (λ(*E*,*x*). (*E*, *Inl x*)) ‘ 𝔈 ∪ (λ(*F*,*y*). (*F*, *Inr y*)) ‘ 𝔉›
⟨*proof*⟩

instantiation *kraus-family* :: (*chilbert-space*, *chilbert-space*, *type*) *plus* **begin**
definition *plus-kraus-family* **where** ‹𝔈 + 𝔉 = *kf-map* (λ*xy*. *case xy of Inl x* ⇒ *x* | *Inr y* ⇒ *y*) (*kf-plus* 𝔈 𝔉)›
instance⟨*proof*⟩
end

lemma *kf-plus-apply*:
fixes 𝔈 :: ‹(*'a*::*chilbert-space*, *'b*::*chilbert-space*, *'x*) *kraus-family*›
and 𝔉 :: ‹(*'a*, *'b*, *'y*) *kraus-family*›
shows ‹*kf-apply* (*kf-plus* 𝔈 𝔉) ϱ = *kf-apply* 𝔈 ϱ + *kf-apply* 𝔉 ϱ›
⟨*proof*⟩

lemma *kf-plus-apply'*: ‹(𝔈 + 𝔉) $*_{kr}$ ϱ = 𝔈 $*_{kr}$ ϱ + 𝔉 $*_{kr}$ ϱ›
⟨*proof*⟩

lemma *kf-plus-0-left*[*simp*]: ‹*kf-plus 0* 𝔈 = *kf-map-inj Inr* 𝔈›
⟨*proof*⟩

lemma *kf-plus-0-right*[*simp*]: ‹*kf-plus* 𝔈 *0* = *kf-map-inj Inl* 𝔈›
⟨*proof*⟩

lemma *kf-plus-0-left'*[*simp*]: ‹*0* + 𝔈 $\equiv_{kr}$ 𝔈›
⟨*proof*⟩

lemma *kf-plus-0-right'*: ‹𝔈 + *0* $\equiv_{kr}$ 𝔈›
⟨*proof*⟩

lemma *kf-plus-bound*: ‹*kf-bound* (*kf-plus* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-bound* $\mathfrak{E}$ + *kf-bound* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-plus-bound'*: ‹*kf-bound* ($\mathfrak{E}$ + $\mathfrak{F}$) = *kf-bound* $\mathfrak{E}$ + *kf-bound* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-norm-triangle*: ‹*kf-norm* (*kf-plus* $\mathfrak{E}$ $\mathfrak{F}$) ≤ *kf-norm* $\mathfrak{E}$ + *kf-norm* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-norm-triangle'*: ‹*kf-norm* ($\mathfrak{E}$ + $\mathfrak{F}$) ≤ *kf-norm* $\mathfrak{E}$ + *kf-norm* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-plus-map-both*:
‹*kf-plus* (*kf-map* *f* $\mathfrak{E}$) (*kf-map* *g* $\mathfrak{F}$) = *kf-map* (*map-sum* *f* *g*) (*kf-plus* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

2.8 Composition

lemma *kf-comp-dependent-raw-norm-aux*:
fixes $\mathfrak{E}$ :: ‹*'a* ⇒ (*'e*::*chilbert-space*, *'f*::*chilbert-space*, *'g*) *kraus-family*›
and $\mathfrak{F}$ :: ‹(*'b*::*chilbert-space*, *'e*, *'a*) *kraus-family*›
assumes *B*: ‹⋀*x*. *x* ∈ *kf-domain* $\mathfrak{F}$ ⟹ *kf-norm* ($\mathfrak{E}$ *x*) ≤ *B*›
assumes [*simp*]: ‹*B* ≥ *0*›
assumes ‹*finite C*›
assumes *C-subset*: ‹*C* ⊆ (λ((*F*,*y*), (*E*,*x*)). (*E* o_{CL} *F*, (*F*,*E*,*y*,*x*))) ‘ (*SIGMA* (*F*,*y*):*Rep-kraus-family* $\mathfrak{F}$. *Rep-kraus-family* ($\mathfrak{E}$ *y*))›
shows ‹(∑(*E*,*x*)∈*C*. *E*∗ o_{CL} *E*) ≤ (*B* ∗ *kf-norm* $\mathfrak{F}$) $*_R$ *id-cblinfun*›
⟨*proof*⟩

lift-definition *kf-comp-dependent-raw* :: ‹(*'x* ⇒ (*'b*::*chilbert-space*,*'c*::*chilbert-space*,*'y*) *kraus-family*) ⇒ (*'a*::*chilbert-space*,*'b*,*'x*) *kraus-family*
⇒ (*'a*, *'c*, (*'a* $\Rightarrow_{CL}$ *'b*) × (*'b* $\Rightarrow_{CL}$ *'c*) × *'x* × *'y*) *kraus-family*› **is**
‹λ$\mathfrak{E}$ $\mathfrak{F}$. *if bdd-above* ((*kf-norm o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$) *then*
Set.filter (λ(*EF*,-). *EF*≠*0*) ((λ((*F*,*y*), (*E*::*'b*$\Rightarrow_{CL}$*'c*,*x*::*'y*)). (*E* o_{CL} *F*, (*F*,*E*,*y*,*x*))) ‘ (*SIGMA* (*F*::*'a*$\Rightarrow_{CL}$*'b*,*y*::*'x*):*Rep-kraus-family* $\mathfrak{F}$. (*Rep-kraus-family* ($\mathfrak{E}$ *y*))))
else {}›
⟨*proof*⟩

lemma *kf-comp-dependent-raw-norm-leq*:
fixes $\mathfrak{E}$ :: ‹*'a* ⇒ (*'b*::*chilbert-space*, *'c*::*chilbert-space*, *'d*) *kraus-family*›
and $\mathfrak{F}$ :: ‹(*'e*::*chilbert-space*, *'b*, *'a*) *kraus-family*›
assumes ‹⋀*x*. *x* ∈ *kf-domain* $\mathfrak{F}$ ⟹ *kf-norm* ($\mathfrak{E}$ *x*) ≤ *B*›
assumes ‹*B* ≥ *0*›
shows ‹*kf-norm* (*kf-comp-dependent-raw* $\mathfrak{E}$ $\mathfrak{F}$) ≤ *B* ∗ *kf-norm* $\mathfrak{F}$›
⟨*proof*⟩

hide-fact *kf-comp-dependent-raw-norm-aux*

definition ‹*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$ = *kf-map* (λ(*F*,*E*,*y*,*x*). (*y*,*x*)) (*kf-comp-dependent-raw* $\mathfrak{E}$ $\mathfrak{F}$)›

definition ‹*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$ = *kf-comp-dependent* (λ-. $\mathfrak{E}$) $\mathfrak{F}$›

lemma *kf-comp-dependent-norm-leq*:
  assumes ‹$\bigwedge x.\ x \in$ *kf-domain* $\mathfrak{F} \Longrightarrow$ *kf-norm* ($\mathfrak{E}$ x) $\leq B$›
  assumes ‹$B \geq 0$›
  shows ‹*kf-norm* (*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$) $\leq B *$ *kf-norm* $\mathfrak{F}$›
  ⟨*proof*⟩

lemma *kf-comp-norm-leq*:
  shows ‹*kf-norm* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) $\leq$ *kf-norm* $\mathfrak{E}$ $*$ *kf-norm* $\mathfrak{F}$›
  ⟨*proof*⟩

lemma *kf-comp-dependent-raw-apply*:
  fixes $\mathfrak{E}$ :: ‹$'y \Rightarrow$ ($'a$::*chilbert-space*, $'b$::*chilbert-space*, $'x$) *kraus-family*›
    and $\mathfrak{F}$ :: ‹($'c$::*chilbert-space*, $'a$, $'y$) *kraus-family*›
  assumes ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
  shows ‹*kf-comp-dependent-raw* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ ϱ
      $= (\sum_\infty(F,y)\in$*Rep-kraus-family* $\mathfrak{F}$. $\mathfrak{E}$ y $*_{kr}$ *sandwich-tc* F ϱ)›
⟨*proof*⟩

lemma *kf-comp-dependent-apply*:
  fixes $\mathfrak{E}$ :: ‹$'y \Rightarrow$ ($'a$::*chilbert-space*, $'b$::*chilbert-space*, $'x$) *kraus-family*›
    and $\mathfrak{F}$ :: ‹($'c$::*chilbert-space*, $'a$, $'y$) *kraus-family*›
  assumes ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
  shows ‹*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ ϱ
    $= (\sum_\infty(F,y)\in$*Rep-kraus-family* $\mathfrak{F}$. $\mathfrak{E}$ y $*_{kr}$ *sandwich-tc* F ϱ)›
  ⟨*proof*⟩

lemma *kf-comp-apply*:
  shows ‹*kf-apply* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-apply* $\mathfrak{E}$ ∘ *kf-apply* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-comp-cong-weak*: ‹*kf-comp* F G $=_{kr}$ *kf-comp* F' G'›
  if ‹$F =_{kr} F'$› **and** ‹$G =_{kr} G'$›
  ⟨*proof*⟩

lemma *kf-comp-dependent-raw-assoc*:
  fixes $\mathfrak{E}$ :: ‹$'f \Rightarrow$ ($'c$::*chilbert-space*,$'d$::*chilbert-space*,$'e$) *kraus-family*›
    and $\mathfrak{F}$ :: ‹$'g \Rightarrow$ ($'b$::*chilbert-space*,$'c$::*chilbert-space*,$'f$) *kraus-family*›
    and $\mathfrak{G}$ :: ‹($'a$::*chilbert-space*,$'b$::*chilbert-space*,$'g$) *kraus-family*›
  defines ‹*reorder* :: $'a \Rightarrow_{CL} 'c \times 'c \Rightarrow_{CL} 'd \times ('a \Rightarrow_{CL} 'b \times 'b \Rightarrow_{CL} 'c \times 'g \times 'f) \times 'e$
        $\Rightarrow 'a \Rightarrow_{CL} 'b \times 'b \Rightarrow_{CL} 'd \times 'g \times 'b \Rightarrow_{CL} 'c \times 'c \Rightarrow_{CL} 'd \times 'f \times 'e \equiv$
      $\lambda(FG::'a \Rightarrow_{CL} 'c,\ E::'c \Rightarrow_{CL} 'd,\ (G::'a \Rightarrow_{CL} 'b,\ F::'b \Rightarrow_{CL} 'c,\ g::'g,\ f::'f),\ e::'e).$
      $(G,\ E\ o_{CL}\ F,\ g,\ F,\ E,\ f,\ e)$›
  assumes ‹*bdd-above* (*range* (*kf-norm o* $\mathfrak{E}$))›
  assumes ‹*bdd-above* (*range* (*kf-norm o* $\mathfrak{F}$))›
  shows ‹*kf-comp-dependent-raw* (λg::$'g$. *kf-comp-dependent-raw* $\mathfrak{E}$ ($\mathfrak{F}$ g)) $\mathfrak{G}$
    = *kf-map-inj reorder* (*kf-comp-dependent-raw* (λ(-,-,-,f). $\mathfrak{E}$ f) (*kf-comp-dependent-raw* $\mathfrak{F}$

$\mathfrak{G}$))›
(**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *kf-filter-comp-dependent*:
fixes $\mathfrak{F}$ :: ‹′*e* ⇒ (′*b*::*chilbert-space*,′*c*::*chilbert-space*,′*f*) *kraus-family*›
and $\mathfrak{E}$ :: ‹(′*a*::*chilbert-space*,′*b*::*chilbert-space*,′*e*) *kraus-family*›
assumes ‹*bdd-above* ((*kf-norm o* $\mathfrak{F}$) ‘ *kf-domain* $\mathfrak{E}$)›
shows ‹*kf-filter* (λ(*e*,*f*). *F e f* ∧ *E e*) (*kf-comp-dependent* $\mathfrak{F}$ $\mathfrak{E}$)
= *kf-comp-dependent* (λ*e*. *kf-filter* (*F e*) ($\mathfrak{F}$ *e*)) (*kf-filter E* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-comp-assoc-weak*:
fixes $\mathfrak{E}$ :: ‹(′*c*::*chilbert-space*,′*d*::*chilbert-space*,′*e*) *kraus-family*›
and $\mathfrak{F}$ :: ‹(′*b*::*chilbert-space*,′*c*::*chilbert-space*,′*f*) *kraus-family*›
and $\mathfrak{G}$ :: ‹(′*a*::*chilbert-space*,′*b*::*chilbert-space*,′*g*) *kraus-family*›
shows ‹*kf-comp* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) $\mathfrak{G}$ $=_{kr}$ *kf-comp* $\mathfrak{E}$ (*kf-comp* $\mathfrak{F}$ $\mathfrak{G}$)›
⟨*proof*⟩

lemma *kf-comp-dependent-raw-cong-left*:
assumes ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
assumes ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}'$) ‘ *kf-domain* $\mathfrak{F}$)›
assumes ‹⋀*x*. *x* ∈ *snd* ‘ *Rep-kraus-family* $\mathfrak{F}$ ⟹ $\mathfrak{E}$ *x* = $\mathfrak{E}'$ *x*›
shows ‹*kf-comp-dependent-raw* $\mathfrak{E}$ $\mathfrak{F}$ = *kf-comp-dependent-raw* $\mathfrak{E}'$ $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-comp-dependent-cong-left*:
assumes ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
assumes ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}'$) ‘ *kf-domain* $\mathfrak{F}$)›
assumes ‹⋀*x*. *x* ∈ *kf-domain* $\mathfrak{F}$ ⟹ $\mathfrak{E}$ *x* = $\mathfrak{E}'$ *x*›
shows ‹*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$ = *kf-comp-dependent* $\mathfrak{E}'$ $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-domain-comp-dependent-raw-subset*:
‹*kf-domain* (*kf-comp-dependent-raw* $\mathfrak{E}$ $\mathfrak{F}$) ⊆ *UNIV* × *UNIV* × (*SIGMA x*:*kf-domain* $\mathfrak{F}$. *kf-domain* ($\mathfrak{E}$ *x*))›
⟨*proof*⟩

lemma *kf-domain-comp-dependent-subset*:
‹*kf-domain* (*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$) ⊆ (*SIGMA x*:*kf-domain* $\mathfrak{F}$. *kf-domain* ($\mathfrak{E}$ *x*))›
⟨*proof*⟩

lemma *kf-domain-comp-subset*: ‹*kf-domain* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) ⊆ *kf-domain* $\mathfrak{F}$ × *kf-domain* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-apply-comp-dependent-cong*:
fixes $\mathfrak{E}$ :: ‹′*f* ⇒ (′*b*::*chilbert-space*,′*c*::*chilbert-space*,′*e1*) *kraus-family*›

and $\mathfrak{E}'$ :: ‹$'f \Rightarrow$ ($'b$::*chilbert-space*,$'c$::*chilbert-space*,$'e2$) *kraus-family*›
and $\mathfrak{F}$ $\mathfrak{F}'$ :: ‹($'a$::*chilbert-space*,$'b$::*chilbert-space*,$'f$) *kraus-family*›
assumes *bdd*: ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
assumes *bdd'*: ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}'$) ‘ *kf-domain* $\mathfrak{F}'$)›
assumes ‹$f \in$ *kf-domain* $\mathfrak{F} \Longrightarrow$ *kf-apply-on* ($\mathfrak{E}$ f) E = *kf-apply-on* ($\mathfrak{E}'$ f) E'›
assumes ‹*kf-apply-on* $\mathfrak{F}$ $\{f\}$ = *kf-apply-on* $\mathfrak{F}'$ $\{f\}$›
shows ‹*kf-apply-on* (*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$) ($\{f\}\times E$) = *kf-apply-on* (*kf-comp-dependent* $\mathfrak{E}'$ $\mathfrak{F}'$) ($\{f\}\times E'$)›
⟨*proof*⟩

lemma *kf-comp-dependent-cong-weak*:
fixes $\mathfrak{E}$ :: ‹$'f \Rightarrow$ ($'b$::*chilbert-space*,$'c$::*chilbert-space*,$'e1$) *kraus-family*›
and $\mathfrak{E}'$ :: ‹$'f \Rightarrow$ ($'b$::*chilbert-space*,$'c$::*chilbert-space*,$'e2$) *kraus-family*›
and $\mathfrak{F}$ $\mathfrak{F}'$ :: ‹($'a$::*chilbert-space*,$'b$::*chilbert-space*,$'f$) *kraus-family*›
assumes *bdd*: ‹*bdd-above* ((*kf-norm o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
assumes *eq*: ‹$\bigwedge x.\ x \in$ *kf-domain* $\mathfrak{F} \Longrightarrow \mathfrak{E}\ x =_{kr} \mathfrak{E}'\ x$›
assumes ‹$\mathfrak{F} \equiv_{kr} \mathfrak{F}'$›
shows ‹*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$ $=_{kr}$ *kf-comp-dependent* $\mathfrak{E}'$ $\mathfrak{F}'$›
⟨*proof*⟩

lemma *kf-comp-dependent-assoc*:
fixes $\mathfrak{E}$ :: ‹$'g \Rightarrow 'f \Rightarrow$ ($'c$::*chilbert-space*,$'d$::*chilbert-space*,$'e$) *kraus-family*›
and $\mathfrak{F}$ :: ‹$'g \Rightarrow$ ($'b$::*chilbert-space*,$'c$::*chilbert-space*,$'f$) *kraus-family*›
and $\mathfrak{G}$ :: ‹($'a$::*chilbert-space*,$'b$::*chilbert-space*,$'g$) *kraus-family*›
assumes *bdd-E*: ‹*bdd-above* ((*kf-norm o case-prod* $\mathfrak{E}$) ‘ (*SIGMA* x:*kf-domain* $\mathfrak{G}$. *kf-domain* ($\mathfrak{F}$ x)))›
assumes *bdd-F*: ‹*bdd-above* ((*kf-norm o* $\mathfrak{F}$) ‘ *kf-domain* $\mathfrak{G}$)›
shows ‹(*kf-comp-dependent* (λg. *kf-comp-dependent* ($\mathfrak{E}$ g) ($\mathfrak{F}$ g)) $\mathfrak{G}$) $\equiv_{kr}$
kf-map ($\lambda((g,f),e)$. (g,f,e)) (*kf-comp-dependent* ($\lambda(g,f)$. $\mathfrak{E}$ g f) (*kf-comp-dependent* $\mathfrak{F}$ $\mathfrak{G}$))›
(**is** ‹*?lhs* $\equiv_{kr}$ *?rhs*›)
⟨*proof*⟩

lemma *kf-comp-dependent-assoc-weak*:
fixes $\mathfrak{E}$ :: ‹$'g \Rightarrow 'f \Rightarrow$ ($'c$::*chilbert-space*,$'d$::*chilbert-space*,$'e$) *kraus-family*›
and $\mathfrak{F}$ :: ‹$'g \Rightarrow$ ($'b$::*chilbert-space*,$'c$::*chilbert-space*,$'f$) *kraus-family*›
and $\mathfrak{G}$ :: ‹($'a$::*chilbert-space*,$'b$::*chilbert-space*,$'g$) *kraus-family*›
assumes *bdd-E*: ‹*bdd-above* ((*kf-norm o case-prod* $\mathfrak{E}$) ‘ (*SIGMA* x:*kf-domain* $\mathfrak{G}$. *kf-domain* ($\mathfrak{F}$ x)))›
assumes *bdd-F*: ‹*bdd-above* ((*kf-norm o* $\mathfrak{F}$) ‘ *kf-domain* $\mathfrak{G}$)›
shows ‹*kf-comp-dependent* (λg. *kf-comp-dependent* ($\mathfrak{E}$ g) ($\mathfrak{F}$ g)) $\mathfrak{G}$ $=_{kr}$
kf-comp-dependent ($\lambda(g,f)$. $\mathfrak{E}$ g f) (*kf-comp-dependent* $\mathfrak{F}$ $\mathfrak{G}$)›
⟨*proof*⟩

lemma *kf-comp-dependent-comp-assoc-weak*:
fixes $\mathfrak{E}$ :: ‹($'c$::*chilbert-space*,$'d$::*chilbert-space*,$'e$) *kraus-family*›
and $\mathfrak{F}$ :: ‹$'g \Rightarrow$ ($'b$::*chilbert-space*,$'c$::*chilbert-space*,$'f$) *kraus-family*›

and $\mathfrak{G}$ :: ‹('a::chilbert-space,'b::chilbert-space,'g) kraus-family›
assumes ‹*bdd-above* ((*kf-norm* *o* $\mathfrak{F}$) ‘ *kf-domain* $\mathfrak{G}$)›
shows ‹*kf-comp-dependent* (λg. *kf-comp* $\mathfrak{E}$ ($\mathfrak{F}$ g)) $\mathfrak{G}$ $=_{kr}$
kf-comp $\mathfrak{E}$ (*kf-comp-dependent* $\mathfrak{F}$ $\mathfrak{G}$)›
⟨*proof*⟩

lemma *kf-comp-comp-dependent-assoc-weak*:
fixes $\mathfrak{E}$ :: ‹'f ⇒ ('c::chilbert-space,'d::chilbert-space,'e) kraus-family›
and $\mathfrak{F}$ :: ‹('b::chilbert-space,'c::chilbert-space,'f) kraus-family›
and $\mathfrak{G}$ :: ‹('a::chilbert-space,'b::chilbert-space,'g) kraus-family›
assumes *bdd-E*: ‹*bdd-above* ((*kf-norm* *o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
shows ‹*kf-comp* (*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$) $\mathfrak{G}$ $=_{kr}$
kf-comp-dependent (λ(-,f). $\mathfrak{E}$ f) (*kf-comp* $\mathfrak{F}$ $\mathfrak{G}$)›
⟨*proof*⟩

lemma *kf-comp-assoc*:
fixes $\mathfrak{E}$ :: ‹('c::chilbert-space,'d::chilbert-space,'e) kraus-family›
and $\mathfrak{F}$ :: ‹('b::chilbert-space,'c::chilbert-space,'f) kraus-family›
and $\mathfrak{G}$ :: ‹('a::chilbert-space,'b::chilbert-space,'g) kraus-family›
shows ‹*kf-comp* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) $\mathfrak{G}$ $\equiv_{kr}$
kf-map (λ((g,f),e). (g,f,e)) (*kf-comp* $\mathfrak{E}$ (*kf-comp* $\mathfrak{F}$ $\mathfrak{G}$))›
⟨*proof*⟩

lemma *kf-comp-dependent-cong*:
fixes $\mathfrak{E}$ $\mathfrak{E}'$ :: ‹'f ⇒ ('b::chilbert-space,'c::chilbert-space,'e) kraus-family›
and $\mathfrak{F}$ $\mathfrak{F}'$ :: ‹('a::chilbert-space,'b::chilbert-space,'f) kraus-family›
assumes *bdd*: ‹*bdd-above* ((*kf-norm* *o* $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
assumes ‹$\bigwedge x$. $x \in$ *kf-domain* $\mathfrak{F} \Longrightarrow \mathfrak{E}\ x \equiv_{kr} \mathfrak{E}'\ x$›
assumes ‹$\mathfrak{F} \equiv_{kr} \mathfrak{F}'$›
shows ‹*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$ $\equiv_{kr}$ *kf-comp-dependent* $\mathfrak{E}'$ $\mathfrak{F}'$›
⟨*proof*⟩

lemma *kf-comp-cong*:
fixes $\mathfrak{E}$ $\mathfrak{E}'$ :: ‹('b::chilbert-space,'c::chilbert-space,'e) kraus-family›
and $\mathfrak{F}$ $\mathfrak{F}'$ :: ‹('a::chilbert-space,'b::chilbert-space,'f) kraus-family›
assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{E}'$›
assumes ‹$\mathfrak{F} \equiv_{kr} \mathfrak{F}'$›
shows ‹*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$ $\equiv_{kr}$ *kf-comp* $\mathfrak{E}'$ $\mathfrak{F}'$›
⟨*proof*⟩

lemma *kf-bound-comp-dependent-raw-of-op*:
shows ‹*kf-bound* (*kf-comp-dependent-raw* $\mathfrak{E}$ (*kf-of-op* U))
= *sandwich* (U∗) (*kf-bound* ($\mathfrak{E}$ ()))›
⟨*proof*⟩

lemma *kf-bound-comp-dependent-of-op*:
shows ‹*kf-bound* (*kf-comp-dependent* $\mathfrak{E}$ (*kf-of-op* U)) = *sandwich* (U∗) (*kf-bound* ($\mathfrak{E}$ ()))›

⟨*proof*⟩

lemma *kf-bound-comp-of-op*:
shows ‹*kf-bound* (*kf-comp* $\mathfrak{E}$ (*kf-of-op* *U*)) = *sandwich* (*U*∗) (*kf-bound* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *kf-norm-comp-dependent-of-op-coiso*:
assumes ‹*isometry* (*U*∗)›
shows ‹*kf-norm* (*kf-comp-dependent* $\mathfrak{E}$ (*kf-of-op* *U*)) = *kf-norm* ($\mathfrak{E}$ ())›
⟨*proof*⟩

lemma *kf-norm-comp-of-op-coiso*:
assumes ‹*isometry* (*U*∗)›
shows ‹*kf-norm* (*kf-comp* $\mathfrak{E}$ (*kf-of-op* *U*)) = *kf-norm* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-bound-comp-dependend-raw-iso*:
assumes ‹*isometry* *U*›
shows ‹*kf-bound* (*kf-comp-dependent-raw* (λ-. *kf-of-op* *U*) $\mathfrak{E}$)
= *kf-bound* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-bound-comp-dependent-iso*:
assumes ‹*isometry* *U*›
shows ‹*kf-bound* (*kf-comp-dependent* (λ-. *kf-of-op* *U*) $\mathfrak{E}$) = *kf-bound* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-bound-comp-iso*:
assumes ‹*isometry* *U*›
shows ‹*kf-bound* (*kf-comp* (*kf-of-op* *U*) $\mathfrak{E}$) = *kf-bound* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-norm-comp-dependent-iso*:
assumes ‹*isometry* *U*›
shows ‹*kf-norm* (*kf-comp-dependent* (λ-. *kf-of-op* *U*) $\mathfrak{E}$) = *kf-norm* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-norm-comp-iso*:
assumes ‹*isometry* *U*›
shows ‹*kf-norm* (*kf-comp* (*kf-of-op* *U*) $\mathfrak{E}$) = *kf-norm* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-comp-dependent-raw-0-right*[*simp*]: ‹*kf-comp-dependent-raw* $\mathfrak{E}$ *0* = *0*›
⟨*proof*⟩

lemma *kf-comp-dependent-raw-0-left*[*simp*]: ‹*kf-comp-dependent-raw* *0* $\mathfrak{E}$ = *0*›
⟨*proof*⟩

lemma *kf-comp-dependent-0-left*[*simp*]: ‹*kf-comp-dependent* (λ-. *0*) *E* = *0*›
⟨*proof*⟩

lemma *kf-comp-dependent-0-right*[*simp*]: ‹*kf-comp-dependent E 0* = *0*›
⟨*proof*⟩

lemma *kf-comp-0-left*[*simp*]: ‹*kf-comp 0 E* = *0*›
⟨*proof*⟩

lemma *kf-comp-0-right*[*simp*]: ‹*kf-comp E 0* = *0*›
⟨*proof*⟩

lemma *kf-filter-comp*:
fixes 𝔉 :: ‹(*'b*::*chilbert-space*,*'c*::*chilbert-space*,*'f*) *kraus-family*›
and 𝔈 :: ‹(*'a*::*chilbert-space*,*'b*::*chilbert-space*,*'e*) *kraus-family*›
shows ‹*kf-filter* (λ(*e*,*f*). *F f* ∧ *E e*) (*kf-comp* 𝔉 𝔈)
= *kf-comp* (*kf-filter F* 𝔉) (*kf-filter E* 𝔈)›
⟨*proof*⟩

lemma *kf-comp-dependent-invalid*:
assumes ‹¬ *bdd-above* ((*kf-norm o* 𝔈) ‘ *kf-domain* 𝔉)›
shows ‹*kf-comp-dependent* 𝔈 𝔉 = *0*›
⟨*proof*⟩

lemma *kf-comp-dependent-map-left*:
‹*kf-comp-dependent* (λ*x*. *kf-map* (*f x*) (*E x*)) *F*
$\equiv_{kr}$ *kf-map* (λ(*x*,*y*). (*x*, *f x y*)) (*kf-comp-dependent E F*)›
⟨*proof*⟩

lemma *kf-comp-dependent-map-right*:
‹*kf-comp-dependent E* (*kf-map f F*)
$\equiv_{kr}$ *kf-map* (λ(*x*,*y*). (*f x*, *y*)) (*kf-comp-dependent* (λ*x*. *E* (*f x*)) *F*)›
⟨*proof*⟩

lemma *kf-comp-dependent-raw-map-inj-right*:
‹*kf-comp-dependent-raw E* (*kf-map-inj f F*)
= *kf-map-inj* (λ(*E*,*F*,*x*,*y*). (*E*, *F*, *f x*, *y*)) (*kf-comp-dependent-raw* (λ*x*. *E* (*f x*)) *F*)›
⟨*proof*⟩

lemma *kf-comp-dependent-map-inj-right*:
assumes ‹*inj-on f* (*kf-domain F*)›
shows ‹*kf-comp-dependent E* (*kf-map-inj f F*)
= *kf-map-inj* (λ(*x*,*y*). (*f x*, *y*)) (*kf-comp-dependent* (λ*x*. *E* (*f x*)) *F*)›
⟨*proof*⟩

lemma *kf-comp-dependent-map-right-weak*:

‹*kf-comp-dependent* E (*kf-map* f F)
$=_{kr}$ *kf-comp-dependent* (λx. E (f x)) F›
⟨*proof*⟩

lemma *kf-comp-map-left*:
‹*kf-comp* (*kf-map* f E) F $\equiv_{kr}$ *kf-map* ($\lambda(x,y)$. (x, f y)) (*kf-comp* E F)›
⟨*proof*⟩

lemma *kf-comp-map-right*:
‹*kf-comp* E (*kf-map* f F) $\equiv_{kr}$ *kf-map* ($\lambda(x,y)$. (f x, y)) (*kf-comp* E F)›
⟨*proof*⟩

lemma *kf-comp-map-both*:
‹*kf-comp* (*kf-map* e E) (*kf-map* f F) $\equiv_{kr}$ *kf-map* ($\lambda(x,y)$. (f x, e y)) (*kf-comp* E F)›
⟨*proof*⟩

lemma *kf-apply-commute*:
assumes ‹*kf-operators* $\mathfrak{F}$ $\subseteq$ *commutant* (*kf-operators* $\mathfrak{E}$)›
shows ‹*kf-apply* $\mathfrak{F}$ *o* *kf-apply* $\mathfrak{E}$ = *kf-apply* $\mathfrak{E}$ *o* *kf-apply* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-comp-commute-weak*:
assumes ‹*kf-operators* $\mathfrak{F}$ $\subseteq$ *commutant* (*kf-operators* $\mathfrak{E}$)›
shows ‹*kf-comp* $\mathfrak{F}$ $\mathfrak{E}$ $=_{kr}$ *kf-comp* $\mathfrak{E}$ $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-comp-commute*:
assumes ‹*kf-operators* $\mathfrak{F}$ $\subseteq$ *commutant* (*kf-operators* $\mathfrak{E}$)›
shows ‹*kf-comp* $\mathfrak{F}$ $\mathfrak{E}$ $\equiv_{kr}$ *kf-map* *prod.swap* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-comp-apply-on-singleton*:
‹*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ @$\{x\}$ ϱ = $\mathfrak{E}$ $*_{kr}$ @$\{$*snd* $x\}$ ($\mathfrak{F}$ $*_{kr}$ @$\{$*fst* $x\}$ ϱ)›
⟨*proof*⟩

lemma *kf-comp-dependent-apply-on-singleton*:
assumes ‹*bdd-above* ((*kf-norm* $\circ$ $\mathfrak{E}$) ‘ *kf-domain* $\mathfrak{F}$)›
shows ‹*kf-comp-dependent* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ @$\{x\}$ ϱ = $\mathfrak{E}$ (*fst* x) $*_{kr}$ @$\{$*snd* $x\}$ ($\mathfrak{F}$ $*_{kr}$ @$\{$*fst* $x\}$ ϱ)›
⟨*proof*⟩

lemma *kf-comp-id-left*: ‹*kf-comp* *kf-id* $\mathfrak{E}$ $\equiv_{kr}$ *kf-map* (λx. (x,())) $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-comp-id-right*: ‹*kf-comp* $\mathfrak{E}$ *kf-id* $\equiv_{kr}$ *kf-map* (λx. ((),x)) $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-comp-dependent-raw-kf-plus-left*:
fixes $\mathfrak{D}$ :: ‹$'f$ ⇒ ($'b$::*chilbert-space*, $'c$::*chilbert-space*, $'d$) *kraus-family*›
fixes $\mathfrak{E}$ :: ‹$'f$ ⇒ ($'b$::*chilbert-space*, $'c$::*chilbert-space*, $'e$) *kraus-family*›

fixes $\mathfrak{F}$ :: ‹('a::chilbert-space, 'b, 'f) kraus-family›
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{D}$ x)) ‘ kf-domain $\mathfrak{F}$)›
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{E}$ x)) ‘ kf-domain $\mathfrak{F}$)›
shows ‹kf-comp-dependent-raw (λx. kf-plus ($\mathfrak{D}$ x) ($\mathfrak{E}$ x)) $\mathfrak{F}$ =
kf-map-inj (λx. case x of Inl (F,D,f,d) ⇒ (F, D, f, Inl d) | Inr (F,E,f,e) ⇒ (F, E, f, Inr e))
(kf-plus (kf-comp-dependent-raw $\mathfrak{D}$ $\mathfrak{F}$) (kf-comp-dependent-raw $\mathfrak{E}$ $\mathfrak{F}$))›
⟨proof⟩

lemma *kf-comp-dependent-kf-plus-left*:
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{D}$ x)) ‘ kf-domain $\mathfrak{F}$)›
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{E}$ x)) ‘ kf-domain $\mathfrak{F}$)›
shows ‹kf-comp-dependent (λx. kf-plus ($\mathfrak{D}$ x) ($\mathfrak{E}$ x)) $\mathfrak{F}$ =
kf-map-inj (λx. case x of Inl (f,d) ⇒ (f, Inl d) | Inr (f,e) ⇒ (f, Inr e)) (kf-plus (kf-comp-dependent $\mathfrak{D}$ $\mathfrak{F}$) (kf-comp-dependent $\mathfrak{E}$ $\mathfrak{F}$))›
⟨proof⟩

lemma *kf-map-inj-twice*:
shows ‹kf-map-inj f (kf-map-inj g $\mathfrak{E}$) = kf-map-inj (f o g) $\mathfrak{E}$›
⟨proof⟩

lemma *kf-comp-dependent-kf-plus-left'*:
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{D}$ x)) ‘ kf-domain $\mathfrak{F}$)›
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{E}$ x)) ‘ kf-domain $\mathfrak{F}$)›
shows ‹kf-plus (kf-comp-dependent $\mathfrak{D}$ $\mathfrak{F}$) (kf-comp-dependent $\mathfrak{E}$ $\mathfrak{F}$) =
kf-map-inj (λ(f,de). case de of Inl d ⇒ Inl (f,d) | Inr e ⇒ Inr (f,e)) (kf-comp-dependent (λx. kf-plus ($\mathfrak{D}$ x) ($\mathfrak{E}$ x)) $\mathfrak{F}$)›
⟨proof⟩

lemma *kf-comp-dependent-plus-left*:
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{D}$ x)) ‘ kf-domain $\mathfrak{F}$)›
assumes ‹bdd-above ((λx. kf-norm ($\mathfrak{E}$ x)) ‘ kf-domain $\mathfrak{F}$)›
shows ‹kf-comp-dependent (λx. $\mathfrak{D}$ x + $\mathfrak{E}$ x) $\mathfrak{F}$ $\equiv_{kr}$ kf-comp-dependent $\mathfrak{D}$ $\mathfrak{F}$ + kf-comp-dependent $\mathfrak{E}$ $\mathfrak{F}$›
⟨proof⟩

2.9 Infinite sums

lift-definition *kf-infsum* :: ‹('a ⇒ ('b::chilbert-space,'c::chilbert-space,'x) kraus-family) ⇒ 'a set ⇒ ('b,'c,'a×'x) kraus-family› **is**
‹$\lambda\mathfrak{E}$ A. if summable-on-in cweak-operator-topology (λa. kf-bound ($\mathfrak{E}$ a)) A
then (λ(a,(E,x)). (E,(a,x))) ‘ Sigma A (λa. Rep-kraus-family ($\mathfrak{E}$ a)) else {}›
⟨proof⟩

definition *kf-summable* :: ‹('a ⇒ ('b::chilbert-space,'c::chilbert-space,'x) kraus-family) ⇒ 'a set ⇒ bool› **where**

‹*kf-summable* $\mathfrak{E}$ *A* $\longleftrightarrow$ *summable-on-in cweak-operator-topology* (λa. *kf-bound* ($\mathfrak{E}$ *a*)) *A*›

lemma *kf-summable-from-abs-summable*:
assumes *sum*: ‹(λa. *kf-norm* ($\mathfrak{E}$ *a*)) *summable-on A*›
shows ‹*kf-summable* $\mathfrak{E}$ *A*›
⟨*proof*⟩

lemma *kf-infsum-apply*:
assumes ‹*kf-summable* $\mathfrak{E}$ *A*›
shows ‹*kf-infsum* $\mathfrak{E}$ *A* $*_{kr}$ ϱ = ($\sum_\infty a{\in}A$. $\mathfrak{E}$ *a* $*_{kr}$ ϱ)›
⟨*proof*⟩

lemma *kf-infsum-apply-summable*:
assumes ‹*kf-summable* $\mathfrak{E}$ *A*›
shows ‹(λa. $\mathfrak{E}$ *a* $*_{kr}$ ϱ) *summable-on A*›
⟨*proof*⟩

lemma *kf-bound-infsum*:
fixes *f* :: ‹*'a* ⇒ (*'b*::*chilbert-space*,*'c*::*chilbert-space*,*'x*) *kraus-family*›
assumes ‹*kf-summable f A*›
shows ‹*kf-bound* (*kf-infsum f A*) = *infsum-in cweak-operator-topology* (λa. *kf-bound* (*f a*)) *A*›
⟨*proof*⟩

lemma *kf-norm-infsum*:
assumes *sum*: ‹(λa. *kf-norm* ($\mathfrak{E}$ *a*)) *summable-on A*›
shows ‹*kf-norm* (*kf-infsum* $\mathfrak{E}$ *A*) ≤ ($\sum_\infty a{\in}A$. *kf-norm* ($\mathfrak{E}$ *a*))›
⟨*proof*⟩

lemma *kf-filter-infsum*:
assumes ‹*kf-summable* $\mathfrak{E}$ *A*›
shows ‹*kf-filter P* (*kf-infsum* $\mathfrak{E}$ *A*)
= *kf-infsum* (λa. *kf-filter* (λx. *P* (*a*,*x*)) ($\mathfrak{E}$ *a*)) {*a*∈*A*. ∃ *x*. *P* (*a*,*x*)}›
(**is** ‹*?lhs* = *?rhs*›)
⟨*proof*⟩

lemma *kf-infsum-empty*[*simp*]: ‹*kf-infsum* $\mathfrak{E}$ {} = *0*›
⟨*proof*⟩

lemma *kf-infsum-singleton*[*simp*]: ‹*kf-infsum* $\mathfrak{E}$ {*a*} = *kf-map-inj* (λx. (*a*,*x*)) ($\mathfrak{E}$ *a*)›
⟨*proof*⟩

lemma *kf-infsum-invalid*:
assumes ‹¬ *kf-summable* $\mathfrak{E}$ *A*›
shows ‹*kf-infsum* $\mathfrak{E}$ *A* = *0*›
⟨*proof*⟩

lemma *kf-infsum-cong*:
fixes $\mathfrak{E}$ $\mathfrak{F}$ :: ‹*'a* ⇒ (*'b*::*chilbert-space*, *'c*::*chilbert-space*, *'x*) *kraus-family*›

assumes ‹$\bigwedge a.\ a \in A \Longrightarrow \mathfrak{E}\ a \equiv_{kr} \mathfrak{F}\ a$›
shows ‹*kf-infsum* $\mathfrak{E}$ $A \equiv_{kr}$ *kf-infsum* $\mathfrak{F}$ A›
⟨*proof*⟩

2.10 Trace-preserving maps

definition ‹*kf-trace-preserving* $\mathfrak{E} \longleftrightarrow (\forall\, \varrho.\ \textit{trace-tc}\ (\mathfrak{E} *_{kr} \varrho) = \textit{trace-tc}\ \varrho)$›

definition ‹*kf-trace-reducing* $\mathfrak{E} \longleftrightarrow (\forall\, \varrho{\geq}0.\ \textit{trace-tc}\ (\mathfrak{E} *_{kr} \varrho) \leq \textit{trace-tc}\ \varrho)$›

lemma *kf-trace-reducing-iff-norm-leq1*: ‹*kf-trace-reducing* $\mathfrak{E} \longleftrightarrow$ *kf-norm* $\mathfrak{E} \leq 1$›
⟨*proof*⟩

lemma *kf-trace-preserving-iff-bound-id*: ‹*kf-trace-preserving* $\mathfrak{E} \longleftrightarrow$ *kf-bound* $\mathfrak{E}$ = *id-cblinfun*›
⟨*proof*⟩

lemma *kf-trace-norm-preserving*: ‹*kf-norm* $\mathfrak{E} \leq 1$› **if** ‹*kf-trace-preserving* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-trace-norm-preserving-eq*:
fixes $\mathfrak{E}$ :: ‹('*a*::{*chilbert-space*,*not-singleton*}, '*b*::*chilbert-space*, '*c*) *kraus-family*›
assumes ‹*kf-trace-preserving* $\mathfrak{E}$›
shows ‹*kf-norm* $\mathfrak{E} = 1$›
⟨*proof*⟩

lemma *kf-trace-preserving-map*[*simp*]: ‹*kf-trace-preserving* (*kf-map* f $\mathfrak{E}$) $\longleftrightarrow$ *kf-trace-preserving* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-trace-reducing-map*[*simp*]: ‹*kf-trace-reducing* (*kf-map* f $\mathfrak{E}$) $\longleftrightarrow$ *kf-trace-reducing* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-trace-preserving-id*[*iff*]: ‹*kf-trace-preserving kf-id*›
⟨*proof*⟩

lemma *kf-trace-reducing-id*[*iff*]: ‹*kf-trace-reducing kf-id*›
⟨*proof*⟩

2.11 Sampling

lift-definition *kf-sample* :: ‹('*x* ⇒ *real*) ⇒ ('*a*::*chilbert-space*, '*a*, '*x*) *kraus-family*› **is**
‹$\lambda p.$ *if* $(\forall\, x.\ p\ x \geq 0) \wedge p$ *summable-on UNIV then Set.filter* $(\lambda(E,\text{-}).\ E{\neq}0)$ (*range* ($\lambda x.$ (*sqrt* (p x) $*_R$ *id-cblinfun*, x))) *else* {}›
⟨*proof*⟩

lemma *kf-sample-norm*:
fixes p :: ‹'*x* ⇒ *real*›
assumes ‹$\bigwedge x.\ p\ x \geq 0$›
assumes ‹p *summable-on UNIV*›

shows ‹*kf-norm* (*kf-sample p* :: (*'a*::{*chilbert-space*,*not-singleton*},*'a*,*'x*) *kraus-family*)
= ($\sum_\infty$*x*. *p x*)›
⟨*proof*⟩

2.12 Trace

lift-definition *kf-trace* :: ‹*'a set* ⇒ (*'a*::*chilbert-space*, *'b*::*one-dim*, *'a*) *kraus-family*› **is**
‹λ*B*. *if is-onb B then* (λ*x*. (*vector-to-cblinfun x*∗, *x*)) ‘ *B else* {}›
⟨*proof*⟩

lemma *kf-trace-is-trace*:
assumes ‹*is-onb B*›
shows ‹*kf-trace B* $*_{kr}$ *ϱ* = *one-dim-iso* (*trace-tc ϱ*)›
⟨*proof*⟩

lemma *kf-eq-weak-kf-trace*:
assumes ‹*is-onb A*› **and** ‹*is-onb B*›
shows ‹*kf-trace A* $=_{kr}$ *kf-trace B*›
⟨*proof*⟩

lemma *trace-is-kf-trace*:
assumes ‹*is-onb B*›
shows ‹*trace-tc t* = *one-dim-iso* (*kf-trace B* $*_{kr}$ *t*)›
⟨*proof*⟩

lemma *kf-trace-bound*[*simp*]:
assumes ‹*is-onb B*›
shows ‹*kf-bound* (*kf-trace B*) = *id-cblinfun*›
⟨*proof*⟩

lemma *kf-trace-norm-eq1* [*simp*]:
fixes *B* :: ‹*'a*::{*chilbert-space*, *not-singleton*} *set*›
assumes ‹*is-onb B*›
shows ‹*kf-norm* (*kf-trace B*) = *1*›
⟨*proof*⟩

lemma *kf-trace-norm-leq1* [*simp*]:
fixes *B* :: ‹*'a*::*chilbert-space set*›
assumes ‹*is-onb B*›
shows ‹*kf-norm* (*kf-trace B*) ≤ *1*›
⟨*proof*⟩

2.13 Constant maps

lift-definition *kf-constant-onedim* :: ‹(*'b*,*'b*) *trace-class* ⇒ (*'a*::*one-dim*, *'b*::*chilbert-space*, *unit*) *kraus-family*› **is**
‹λ*t*::(*'b*,*'b*) *trace-class*. *if t* ≥ *0 then*
Set.filter (λ(*E*,-). *E*≠*0*) ((λ*v*. (*vector-to-cblinfun v*,())) ‘ *spectral-dec-vecs-tc t*)
else {}›

⟨*proof*⟩

definition *kf-constant* :: ‹(*'b*,*'b*) *trace-class* ⇒ (*'a*::*chilbert-space*, *'b*::*chilbert-space*, *unit*) *kraus-family*›
where
‹*kf-constant* ϱ = *kf-flatten* (*kf-comp* (*kf-constant-onedim* ϱ :: (*complex*,-,-) *kraus-family*) (*kf-trace* *some-chilbert-basis*))›

lemma *kf-constant-onedim-invalid*: ‹$\neg\ \varrho \geq 0 \Longrightarrow$ *kf-constant-onedim* $\varrho = 0$›
⟨*proof*⟩

lemma *kf-constant-invalid*: ‹$\neg\ \varrho \geq 0 \Longrightarrow$ *kf-constant* $\varrho = 0$›
⟨*proof*⟩

lemma *kf-constant-onedim-apply*:
assumes ‹$\varrho \geq 0$›
shows ‹*kf-apply* (*kf-constant-onedim* ϱ) σ = *one-dim-iso* $\sigma *_C \varrho$›
⟨*proof*⟩

lemma *kf-constant-apply*:
assumes ‹$\varrho \geq 0$›
shows ‹*kf-apply* (*kf-constant* ϱ) σ = *trace-tc* $\sigma *_C \varrho$›
⟨*proof*⟩

lemma *kf-bound-constant-onedim*[*simp*]:
fixes ϱ :: ‹(*'a*::*chilbert-space*, *'a*) *trace-class*›
assumes ‹$\varrho \geq 0$›
shows ‹*kf-bound* (*kf-constant-onedim* ϱ) = *norm* $\varrho *_R$ *id-cblinfun*›
⟨*proof*⟩

lemma *kf-bound-constant*[*simp*]:
fixes ϱ :: ‹(*'a*::*chilbert-space*, *'a*) *trace-class*›
assumes ‹$\varrho \geq 0$›
shows ‹*kf-bound* (*kf-constant* ϱ) = *norm* $\varrho *_R$ *id-cblinfun*›
⟨*proof*⟩

lemma *kf-norm-constant-onedim*[*simp*]:
assumes ‹$\varrho \geq 0$›
shows ‹*kf-norm* (*kf-constant-onedim* ϱ) = *norm* ϱ›
⟨*proof*⟩

lemma *kf-norm-constant*:
assumes ‹$\varrho \geq 0$›
shows ‹*kf-norm* (*kf-constant* ϱ :: (*'a*::{*chilbert-space*,*not-singleton*},*'b*::*chilbert-space*,-) *kraus-family*) = *norm* ϱ›
⟨*proof*⟩

lemma *kf-norm-constant-leq*:
shows ‹*kf-norm* (*kf-constant* ϱ) ≤ *norm* ϱ›
⟨*proof*⟩

lemma *kf-comp-constant-right*:
  assumes [*iff*]: ‹$t \geq 0$›
  shows ‹*kf-map fst* (*kf-comp* E (*kf-constant* t)) $\equiv_{kr}$ *kf-constant* ($E *_{kr} t$)›
⟨*proof*⟩

lemma *kf-comp-constant-right-weak*:
  assumes [*iff*]: ‹$t \geq 0$›
  shows ‹*kf-comp* E (*kf-constant* t) $=_{kr}$ *kf-constant* ($E *_{kr} t$)›
  ⟨*proof*⟩

2.14 Tensor products

lemma *kf-tensor-raw-bound-aux*:
  fixes $\mathfrak{E}$ :: ‹($'a$ *ell2* $\Rightarrow_{CL}$ $'b$ *ell2* $\times$ $'x$) *set*› **and** $\mathfrak{F}$ :: ‹($'c$ *ell2* $\Rightarrow_{CL}$ $'d$ *ell2* $\times$ $'y$) *set*›
  assumes ‹$\bigwedge S$. *finite* $S \Longrightarrow S \subseteq \mathfrak{E} \Longrightarrow (\sum(E, x)\in S.\ E{*}\ o_{CL}\ E) \leq B$›
  assumes ‹$\bigwedge S$. *finite* $S \Longrightarrow S \subseteq \mathfrak{F} \Longrightarrow (\sum(E, x)\in S.\ E{*}\ o_{CL}\ E) \leq C$›
  assumes ‹*finite* U›
  assumes ‹$U \subseteq ((\lambda((E, x), F, y).\ (E \otimes_o F, E, F, x, y))$ ‘ $(\mathfrak{E} \times \mathfrak{F}))$›
  shows ‹$(\sum(G, z)\in U.\ G{*}\ o_{CL}\ G) \leq B \otimes_o C$›
⟨*proof*⟩

lift-definition *kf-tensor-raw* :: ‹($'a$ *ell2*, $'b$ *ell2*, $'x$) *kraus-family* $\Rightarrow$ ($'c$ *ell2*, $'d$ *ell2*, $'y$) *kraus-family* $\Rightarrow$
      (($'a \times 'c$) *ell2*, ($'b \times 'd$) *ell2*, (($'a$ *ell2*$\Rightarrow_{CL}$$'b$ *ell2*)$\times$($'c$ *ell2*$\Rightarrow_{CL}$$'d$ *ell2*)$\times 'x \times 'y$)) *kraus-family*›
is
  ‹$\lambda\mathfrak{E}\ \mathfrak{F}.\ (\lambda((E,x), (F,y)).\ (E \otimes_o F, (E,F,x,y)))$ ‘ $(\mathfrak{E}\times\mathfrak{F})$›
⟨*proof*⟩

lemma *kf-apply-tensor-raw-as-infsum*:
  ‹*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ ϱ = $(\sum_\infty((E,x),(F,y))\in$*Rep-kraus-family* $\mathfrak{E}$ $\times$ *Rep-kraus-family* $\mathfrak{F}$. *sandwich-tc* ($E \otimes_o F$) ϱ)›
⟨*proof*⟩

lemma *kf-apply-tensor-raw*:
  shows ‹*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ *tc-tensor* ϱ σ = *tc-tensor* ($\mathfrak{E} *_{kr} \varrho$) ($\mathfrak{F} *_{kr} \sigma$)›
⟨*proof*⟩

hide-fact *kf-tensor-raw-bound-aux*

definition ‹*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$ = *kf-map* ($\lambda(E, F, x, y).\ (x,y)$) (*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$)›

lemma *kf-apply-tensor*:
  ‹*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ *tc-tensor* ϱ σ = *tc-tensor* ($\mathfrak{E} *_{kr} \varrho$) ($\mathfrak{F} *_{kr} \sigma$)›
  ⟨*proof*⟩

lemma *kf-apply-tensor-as-infsum*:
  ‹*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$ $*_{kr}$ ϱ = $(\sum_\infty((E,x),(F,y))\in$*Rep-kraus-family* $\mathfrak{E}$ $\times$ *Rep-kraus-family* $\mathfrak{F}$. *sand-*

wich-tc $(E \otimes_o F)\ \varrho)$›
  ⟨*proof*⟩

lemma *kf-bound-tensor-raw*:
  ‹*kf-bound* (*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-bound* $\mathfrak{E}$ $\otimes_o$ *kf-bound* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-bound-tensor*:
  ‹*kf-bound* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-bound* $\mathfrak{E}$ $\otimes_o$ *kf-bound* $\mathfrak{F}$›
  ⟨*proof*⟩

lemma *kf-norm-tensor*:
  ‹*kf-norm* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-norm* $\mathfrak{E}$ $*$ *kf-norm* $\mathfrak{F}$›
  ⟨*proof*⟩

lemma *kf-tensor-cong-weak*:
  assumes ‹$\mathfrak{E} =_{kr} \mathfrak{E}'$›
  assumes ‹$\mathfrak{F} =_{kr} \mathfrak{F}'$›
  shows ‹*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$ $=_{kr}$ *kf-tensor* $\mathfrak{E}'$ $\mathfrak{F}'$›
⟨*proof*⟩

lemma *kf-filter-tensor*:
  ‹*kf-filter* ($\lambda(x,y)$. $P\ x \land Q\ y$) (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-tensor* (*kf-filter* P $\mathfrak{E}$) (*kf-filter* Q $\mathfrak{F}$)›
  ⟨*proof*⟩

lemma *kf-filter-tensor-singleton*:
  ‹*kf-filter* ((=) x) (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-tensor* (*kf-filter* ((=) (*fst* x)) $\mathfrak{E}$) (*kf-filter* ((=) (*snd* x)) $\mathfrak{F}$)›
  ⟨*proof*⟩

lemma *kf-tensor-cong*:
  fixes $\mathfrak{E}$ $\mathfrak{E}'$ :: ‹('*a ell2*, '*b ell2*, '*x*) *kraus-family*›
    and $\mathfrak{F}$ $\mathfrak{F}'$ :: ‹('*c ell2*, '*d ell2*, '*y*) *kraus-family*›
  assumes ‹$\mathfrak{E} \equiv_{kr} \mathfrak{E}'$›
  assumes ‹$\mathfrak{F} \equiv_{kr} \mathfrak{F}'$›
  shows ‹*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$ $\equiv_{kr}$ *kf-tensor* $\mathfrak{E}'$ $\mathfrak{F}'$›
⟨*proof*⟩

lemma *kf-tensor-compose-distrib-weak*:
  shows ‹*kf-tensor* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) (*kf-comp* $\mathfrak{G}$ $\mathfrak{H}$)
    $=_{kr}$ *kf-comp* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{G}$) (*kf-tensor* $\mathfrak{F}$ $\mathfrak{H}$)›
  ⟨*proof*⟩

lemma *kf-tensor-compose-distrib*:
  shows ‹*kf-tensor* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) (*kf-comp* $\mathfrak{G}$ $\mathfrak{H}$)
    $\equiv_{kr}$ *kf-map* ($\lambda((e,g),(f,h))$. $((e,f),(g,h))$) (*kf-comp* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{G}$) (*kf-tensor* $\mathfrak{F}$ $\mathfrak{H}$))›

⟨*proof*⟩

lemma *kf-tensor-compose-distrib′*:
shows ‹*kf-comp* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{G}$) (*kf-tensor* $\mathfrak{F}$ $\mathfrak{H}$)
$\equiv_{kr}$ *kf-map* (λ((*e*,*f*),(*g*,*h*)). ((*e*,*g*),(*f*,*h*))) (*kf-tensor* (*kf-comp* $\mathfrak{E}$ $\mathfrak{F}$) (*kf-comp* $\mathfrak{G}$ $\mathfrak{H}$))›
⟨*proof*⟩

definition *kf-tensor-right* :: ‹(′*extra ell2*, ′*extra ell2*) *trace-class* ⇒ (′*qu ell2*, (′*qu*×′*extra*) *ell2*, *unit*) *kraus-family*› **where**
— *kf-tensor-right* ϱ maps σ to σ $\otimes_o$ ϱ
‹*kf-tensor-right* ϱ = *kf-map-inj* (λ-. ()) (*kf-comp* (*kf-tensor* *kf-id* (*kf-constant-onedim* ϱ)) (*kf-of-op* (*tensor-ell2-right* (*ket* ()))))›
definition *kf-tensor-left* :: ‹(′*extra ell2*, ′*extra ell2*) *trace-class* ⇒ (′*qu ell2*, (′*extra*×′*qu*) *ell2*, *unit*) *kraus-family*› **where**
— *kf-tensor-right* ϱ maps σ to ϱ $\otimes_o$ σ
‹*kf-tensor-left* ϱ = *kf-map-inj* (λ-. ()) (*kf-comp* (*kf-tensor* (*kf-constant-onedim* ϱ) *kf-id*) (*kf-of-op* (*tensor-ell2-left* (*ket* ()))))›

lemma *kf-apply-tensor-right*[*simp*]:
assumes ‹ϱ ≥ *0*›
shows ‹*kf-tensor-right* ϱ $*_{kr}$ σ = *tc-tensor* σ ϱ›
⟨*proof*⟩
lemma *kf-apply-tensor-left*[*simp*]:
assumes ‹ϱ ≥ *0*›
shows ‹*kf-tensor-left* ϱ $*_{kr}$ σ = *tc-tensor* ϱ σ›
⟨*proof*⟩

lemma *kf-bound-tensor-right*[*simp*]:
assumes ‹ϱ ≥ *0*›
shows ‹*kf-bound* (*kf-tensor-right* ϱ) = *norm* ϱ $*_C$ *id-cblinfun*›
⟨*proof*⟩
lemma *kf-bound-tensor-left*[*simp*]:
assumes ‹ϱ ≥ *0*›
shows ‹*kf-bound* (*kf-tensor-left* ϱ) = *norm* ϱ $*_C$ *id-cblinfun*›
⟨*proof*⟩

lemma *kf-norm-tensor-right*[*simp*]:
assumes ‹ϱ ≥ *0*›
shows ‹*kf-norm* (*kf-tensor-right* ϱ) = *norm* ϱ›
⟨*proof*⟩
lemma *kf-norm-tensor-left*[*simp*]:
assumes ‹ϱ ≥ *0*›
shows ‹*kf-norm* (*kf-tensor-left* ϱ) = *norm* ϱ›
⟨*proof*⟩

lemma *kf-trace-preserving-tensor*:

assumes ‹*kf-trace-preserving* $\mathfrak{E}$› **and** ‹*kf-trace-preserving* $\mathfrak{F}$›
shows ‹*kf-trace-preserving* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-trace-reducing-tensor*:
assumes ‹*kf-trace-reducing* $\mathfrak{E}$› **and** ‹*kf-trace-reducing* $\mathfrak{F}$›
shows ‹*kf-trace-reducing* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-tensor-map-left*:
‹*kf-tensor* (*kf-map f* $\mathfrak{E}$) $\mathfrak{F}$ $\equiv_{kr}$ *kf-map* (*apfst f*) (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-tensor-map-right*:
‹*kf-tensor* $\mathfrak{E}$ (*kf-map f* $\mathfrak{F}$) $\equiv_{kr}$ *kf-map* (*apsnd f*) (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-tensor-map-both*:
‹*kf-tensor* (*kf-map f* $\mathfrak{E}$) (*kf-map g* $\mathfrak{F}$) $\equiv_{kr}$ *kf-map* (*map-prod f g*) (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-tensor-raw-map-inj-both*:
‹*kf-tensor-raw* (*kf-map-inj f* $\mathfrak{E}$) (*kf-map-inj g* $\mathfrak{F}$) = *kf-map-inj* ($\lambda(E,F,x,y)$. $(E,F,f\ x,g\ y)$) (*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-domain-tensor-raw-subset*:
‹*kf-domain* (*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$) $\subseteq$ *kf-operators* $\mathfrak{E}$ $\times$ *kf-operators* $\mathfrak{F}$ $\times$ *kf-domain* $\mathfrak{E}$ $\times$ *kf-domain* $\mathfrak{F}$›
⟨*proof*⟩

lemma *kf-tensor-map-inj-both*:
assumes ‹*inj-on f* (*kf-domain* $\mathfrak{E}$)›
assumes ‹*inj-on g* (*kf-domain* $\mathfrak{F}$)›
shows ‹*kf-tensor* (*kf-map-inj f* $\mathfrak{E}$) (*kf-map-inj g* $\mathfrak{F}$) = *kf-map-inj* (*map-prod f g*) (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *kf-operators-tensor-raw*:
shows ‹*kf-operators* (*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$) = $\{E \otimes_o F \mid E\ F.\ E \in$ *kf-operators* $\mathfrak{E} \wedge F \in$ *kf-operators* $\mathfrak{F}\}$›
⟨*proof*⟩

lemma *kf-operators-tensor*:
shows ‹*kf-operators* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$) $\subseteq$ *span* $\{E \otimes_o F \mid E\ F.\ E \in$ *kf-operators* $\mathfrak{E} \wedge F \in$ *kf-operators* $\mathfrak{F}\}$›
⟨*proof*⟩

lemma *kf-domain-tensor*: ‹*kf-domain* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$) = *kf-domain* $\mathfrak{E}$ $\times$ *kf-domain* $\mathfrak{F}$›

⟨*proof*⟩

lemma *kf-tensor-raw-0-left*[*simp*]: ‹*kf-tensor-raw 0* $\mathfrak{E}$ = *0*›
⟨*proof*⟩

lemma *kf-tensor-raw-0-right*[*simp*]: ‹*kf-tensor-raw* $\mathfrak{E}$ *0* = *0*›
⟨*proof*⟩

lemma *kf-tensor-0-left*[*simp*]: ‹*kf-tensor 0* $\mathfrak{E}$ = *0*›
⟨*proof*⟩

lemma *kf-tensor-0-right*[*simp*]: ‹*kf-tensor* $\mathfrak{E}$ *0* = *0*›
⟨*proof*⟩

lemma *kf-tensor-of-op*:
‹*kf-tensor* (*kf-of-op A*) (*kf-of-op B*) = *kf-map* (λ(). ((),())) (*kf-of-op* (*A* $\otimes_o$ *B*))›
⟨*proof*⟩

2.15 Partial trace

definition *kf-partial-trace-right* :: ‹(('a×'b) *ell2*, 'a *ell2*, 'b) *kraus-family*› **where**
‹*kf-partial-trace-right* = *kf-map* (λ((-,*b*),-). *inv ket b*)
(*kf-comp* (*kf-of-op* (*tensor-ell2-right* (*ket* ())*))
(*kf-tensor kf-id* (*kf-trace* (*range ket*))))›

definition *kf-partial-trace-left* :: ‹(('a×'b) *ell2*, 'b *ell2*, 'a) *kraus-family*› **where**
‹*kf-partial-trace-left* = *kf-map-inj snd* (*kf-comp kf-partial-trace-right* (*kf-of-op swap-ell2*))›

lemma *partial-trace-is-kf-partial-trace*:
fixes *t* :: ‹(('a × 'b) *ell2*, ('a × 'b) *ell2*) *trace-class*›
shows ‹*partial-trace t* = *kf-partial-trace-right* $*_{kr}$ *t*›
⟨*proof*⟩

lemma *partial-trace-ignores-kraus-family*:
assumes ‹*kf-trace-preserving* $\mathfrak{E}$›
shows ‹*partial-trace* (*kf-tensor* $\mathfrak{F}$ $\mathfrak{E}$ $*_{kr}$ ϱ) = $\mathfrak{F}$ $*_{kr}$ *partial-trace* ϱ›
⟨*proof*⟩

lemma *kf-partial-trace-bound*[*simp*]:
shows ‹*kf-bound kf-partial-trace-right* = *id-cblinfun*›
⟨*proof*⟩

lemma *kf-partial-trace-norm*[*simp*]:
shows ‹*kf-norm kf-partial-trace-right* = *1*›
⟨*proof*⟩

lemma *kf-partial-trace-right-apply-singleton*:
‹*kf-partial-trace-right* $*_{kr}$ @{*x*} ϱ = *sandwich-tc* (*tensor-ell2-right* (*ket x*)*) ϱ›
⟨*proof*⟩

lemma *kf-partial-trace-left-apply-singleton*:
‹*kf-partial-trace-left* $*_{kr}$ @{x} ϱ = *sandwich-tc* (*tensor-ell2-left* (*ket x*)∗) ϱ›
⟨*proof*⟩

lemma *kf-domain-partial-trace-right*[*simp*]: ‹*kf-domain kf-partial-trace-right* = *UNIV*›
⟨*proof*⟩

lemma *kf-domain-partial-trace-left*[*simp*]: ‹*kf-domain kf-partial-trace-left* = *UNIV*›
⟨*proof*⟩

2.16 Complete measurement

lemma *complete-measurement-aux*:
fixes B **and** F :: ‹(′*a*::*chilbert-space* $\Rightarrow_{CL}$ ′*a* × ′*a*) *set*›
defines ‹*family* ≡ (λ*x*. (*selfbutter* (*sgn x*), *x*)) ‘ *B*›
assumes ‹*finite F*› **and** *FB*: ‹*F* ⊆ *family*› **and** ‹*is-ortho-set B*›
shows ‹(∑(*E*, *x*)∈*F*. *E*∗ o_{CL} *E*) ≤ *id-cblinfun*›
⟨*proof*⟩

lemma *complete-measurement-is-kraus-family*:
assumes ‹*is-ortho-set B*›
shows ‹*kraus-family* ((λ*x*. (*selfbutter* (*sgn x*), *x*)) ‘ *B*)›
⟨*proof*⟩

lift-definition *kf-complete-measurement* :: ‹′*a set* ⇒ (′*a*::*chilbert-space*, ′*a*, ′*a*) *kraus-family*› **is**
‹λ*B*. *if is-ortho-set B then* (λ*x*. (*selfbutter* (*sgn x*), *x*)) ‘ *B else* {}›
⟨*proof*⟩

definition *kf-complete-measurement-ket* :: ‹(′*a ell2*, ′*a ell2*, ′*a*) *kraus-family*› **where**
‹*kf-complete-measurement-ket* = *kf-map-inj* (*inv ket*) (*kf-complete-measurement* (*range ket*))›

lemma *kf-complete-measurement-domain*[*simp*]:
assumes ‹*is-ortho-set B*›
shows ‹*kf-domain* (*kf-complete-measurement B*) = *B*›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-domain*[*simp*]:
‹*kf-domain kf-complete-measurement-ket* = *UNIV*›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-kf-map*:
‹*kf-complete-measurement-ket* $\equiv_{kr}$ *kf-map* (*inv ket*) (*kf-complete-measurement* (*range ket*))›
⟨*proof*⟩

lemma *kf-bound-complete-measurement*:

assumes ‹*is-ortho-set B*›
shows ‹*kf-bound* (*kf-complete-measurement B*) $\leq$ *id-cblinfun*›
⟨*proof*⟩

lemma *kf-norm-complete-measurement*:
assumes ‹*is-ortho-set B*›
shows ‹*kf-norm* (*kf-complete-measurement B*) $\leq$ *1*›
⟨*proof*⟩

lemma *kf-complete-measurement-invalid*:
assumes ‹¬ *is-ortho-set B*›
shows ‹*kf-complete-measurement B* = *0*›
⟨*proof*⟩

lemma *kf-complete-measurement-idem*:
‹*kf-comp* (*kf-complete-measurement B*) (*kf-complete-measurement B*)
$\equiv_{kr}$ *kf-map* (λ*b*. (*b*,*b*)) (*kf-complete-measurement B*)›
⟨*proof*⟩

lemma *kf-complete-measurement-idem-weak*:
‹*kf-comp* (*kf-complete-measurement B*) (*kf-complete-measurement B*)
$=_{kr}$ *kf-complete-measurement B*›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-idem*:
‹*kf-comp kf-complete-measurement-ket kf-complete-measurement-ket*
$\equiv_{kr}$ *kf-map* (λ*b*. (*b*,*b*)) *kf-complete-measurement-ket*›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-idem-weak*:
‹*kf-comp kf-complete-measurement-ket kf-complete-measurement-ket*
$=_{kr}$ *kf-complete-measurement-ket*›
⟨*proof*⟩

lemma *kf-complete-measurement-apply*:
assumes [*simp*]: ‹*is-ortho-set B*›
shows ‹*kf-complete-measurement B* $*_{kr}$ *t* = ($\sum_\infty$*x*∈*B*. *sandwich-tc* (*selfbutter* (*sgn x*)) *t*)›
⟨*proof*⟩

lemma *kf-complete-measurement-has-sum*:
assumes ‹*is-ortho-set B*›
shows ‹((λ*x*. *sandwich-tc* (*selfbutter* (*sgn x*)) ϱ) *has-sum kf-complete-measurement B* $*_{kr}$ ϱ) *B*›
⟨*proof*⟩

lemma *kf-complete-measurement-has-sum-onb*:
assumes ‹*is-onb B*›

shows ‹((λx. *sandwich-tc* (*selfbutter* x) ϱ) *has-sum kf-complete-measurement* $B *_{kr}$ ϱ) B›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-has-sum*:
‹((λx. *sandwich-tc* (*selfbutter* (*ket* x)) ϱ) *has-sum kf-complete-measurement-ket* $*_{kr}$ ϱ) *UNIV*›
⟨*proof*⟩

lemma *kf-complete-measurement-apply-onb*:
assumes ‹*is-onb* B›
shows ‹*kf-complete-measurement* $B *_{kr}$ t = ($\sum_\infty x{\in}B$. *sandwich-tc* (*selfbutter* x) t)›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-apply*: ‹*kf-complete-measurement-ket* $*_{kr}$ t = ($\sum_\infty x$. *sandwich-tc* (*selfbutter* (*ket* x)) t)›
⟨*proof*⟩

lemma *kf-bound-complete-measurement-onb*[*simp*]:
assumes ‹*is-onb* B›
shows ‹*kf-bound* (*kf-complete-measurement* B) = *id-cblinfun*›
⟨*proof*⟩

lemma *kf-bound-complete-measurement-ket*[*simp*]:
‹*kf-bound kf-complete-measurement-ket* = *id-cblinfun*›
⟨*proof*⟩

lemma *kf-norm-complete-measurement-onb*[*simp*]:
fixes B :: ‹$'a$::{*not-singleton*, *chilbert-space*} *set*›
assumes ‹*is-onb* B›
shows ‹*kf-norm* (*kf-complete-measurement* B) = *1*›
⟨*proof*⟩

lemma *kf-norm-complete-measurement-ket*[*simp*]:
‹*kf-norm kf-complete-measurement-ket* = *1*›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-diagonal-operator*[*simp*]:
‹*kf-complete-measurement-ket* $*_{kr}$ *diagonal-operator-tc* f = *diagonal-operator-tc* f›
⟨*proof*⟩

lemma *kf-operators-complete-measurement*:
‹*kf-operators* (*kf-complete-measurement* B) = (*selfbutter o sgn*) ‘ B› **if** ‹*is-ortho-set* B›
⟨*proof*⟩

lemma *kf-operators-complete-measurement-invalid*:
‹*kf-operators* (*kf-complete-measurement* B) = {}› **if** ‹$\neg$ *is-ortho-set* B›
⟨*proof*⟩

lemma *kf-operators-complete-measurement-ket*:
‹*kf-operators kf-complete-measurement-ket* = *range* (λ*c*. *butterfly* (*ket c*) (*ket c*))›
⟨*proof*⟩

lemma *kf-complete-measurement-apply-butterfly*:
assumes ‹*is-ortho-set B*› **and** ‹$b \in B$›
shows ‹*kf-complete-measurement B* $*_{kr}$ *tc-butterfly b b* = *tc-butterfly b b*›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-apply-butterfly*:
‹*kf-complete-measurement-ket* $*_{kr}$ *tc-butterfly* (*ket x*) (*ket x*) = *tc-butterfly* (*ket x*) (*ket x*)›
⟨*proof*⟩

lemma *kf-map-eq-kf-map-inj-singleton*:
assumes ‹*card-le-1* (*Rep-kraus-family* 𝔈)›
shows ‹*kf-map f* 𝔈 = *kf-map-inj f* 𝔈›
⟨*proof*⟩

lemma *kf-map-eq-kf-map-inj-singleton'*:
assumes ‹⋀*y*. *card-le-1* (*Rep-kraus-family* (*kf-filter* ((=)*y*) 𝔈))›
assumes ‹*inj-on f* (*kf-domain* 𝔈)›
shows ‹*kf-map f* 𝔈 = *kf-map-inj f* 𝔈›
⟨*proof*⟩

lemma *kf-filter-singleton-kf-complete-measurement*:
assumes ‹$x \in B$› **and** ‹*is-ortho-set B*›
shows ‹*kf-filter* ((=)*x*) (*kf-complete-measurement B*) = *kf-map-inj* (λ-. *x*) (*kf-of-op* (*selfbutter* (*sgn x*)))›
⟨*proof*⟩

lemma *kf-filter-singleton-kf-complete-measurement'*:
assumes ‹$x \in B$› **and** ‹*is-ortho-set B*›
shows ‹*kf-filter* ((=)*x*) (*kf-complete-measurement B*) = *kf-map* (λ-. *x*) (*kf-of-op* (*selfbutter* (*sgn x*)))›
⟨*proof*⟩

lemma *kf-filter-disjoint*:
assumes ‹⋀*x*. $x \in$ *kf-domain* 𝔈 ⟹ *P x* = *False*›
shows ‹*kf-filter P* 𝔈 = *0*›
⟨*proof*⟩

lemma *kf-complete-measurement-tensor*:

assumes ‹*is-ortho-set B*› **and** ‹*is-ortho-set C*›
shows ‹*kf-map* ($\lambda(b,c)$. $b \otimes_s c$) (*kf-tensor* (*kf-complete-measurement B*) (*kf-complete-measurement C*))
= *kf-complete-measurement* (($\lambda(b,c)$. $b \otimes_s c$) ‘ ($B \times C$))›
⟨*proof*⟩

lemma *card-le-1-kf-filter*: ‹*card-le-1* (*Rep-kraus-family* (*kf-filter P* $\mathfrak{E}$))› **if** ‹*card-le-1* (*Rep-kraus-family* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *card-le-1-kf-map-inj*[*iff*]: ‹*card-le-1* (*Rep-kraus-family* (*kf-map-inj f* $\mathfrak{E}$))› **if** ‹*card-le-1* (*Rep-kraus-family* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *card-le-1-kf-map*[*iff*]: ‹*card-le-1* (*Rep-kraus-family* (*kf-map f* $\mathfrak{E}$))› **if** ‹*card-le-1* (*Rep-kraus-family* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *card-le-1-kf-tensor-raw*[*iff*]: ‹*card-le-1* (*Rep-kraus-family* (*kf-tensor-raw* $\mathfrak{E}$ $\mathfrak{F}$))› **if** ‹*card-le-1* (*Rep-kraus-family* $\mathfrak{E}$)› **and** ‹*card-le-1* (*Rep-kraus-family* $\mathfrak{F}$)›
⟨*proof*⟩

lemma *card-le-1-kf-tensor*[*iff*]: ‹*card-le-1* (*Rep-kraus-family* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$))› **if** ‹*card-le-1* (*Rep-kraus-family* $\mathfrak{E}$)› **and** ‹*card-le-1* (*Rep-kraus-family* $\mathfrak{F}$)›
⟨*proof*⟩

lemma *card-le-1-kf-filter-complete-measurement*: ‹*card-le-1* (*Rep-kraus-family* (*kf-filter* ((=)*x*) (*kf-complete-measurement B*)))›
⟨*proof*⟩

lemma *kf-complete-measurement-ket-tensor*:
shows ‹*kf-tensor* (*kf-complete-measurement-ket* :: (-,-,'*a*) *kraus-family*) (*kf-complete-measurement-ket* :: (-,-,'*b*) *kraus-family*)
= *kf-complete-measurement-ket*›
⟨*proof*⟩

2.17 Reconstruction

lemma *kf-reconstruction-is-bounded-clinear*:
assumes ‹$\bigwedge\varrho$. ((λa. *sandwich-tc* (*f a*) ϱ) *has-sum* $\mathfrak{E}$ ϱ) *A*›
shows ‹*bounded-clinear* $\mathfrak{E}$›
⟨*proof*⟩

lemma *kf-reconstruction-is-kraus-family*:
assumes *sum*: ‹$\bigwedge\varrho$. ((λa. *sandwich-tc* (*f a*) ϱ) *has-sum* $\mathfrak{E}$ ϱ) *A*›
defines ‹$F \equiv$ *Set.filter* ($\lambda(E,-)$. $E{\neq}0$) ((λa. (*f a*, *a*)) ‘ *A*)›
shows ‹*kraus-family F*›

⟨*proof*⟩

lemma *kf-reconstruction*:
  assumes *sum*: ‹$\bigwedge\varrho$. ((λa. *sandwich-tc* (f a) ϱ) *has-sum* $\mathfrak{E}$ ϱ) A›
  defines ‹$F \equiv$ *Abs-kraus-family* (*Set.filter* ($\lambda(E,\text{-})$. $E{\neq}0$) ((λa. (f a, a)) ‘ A))›
  shows ‹*kf-apply* $F = \mathfrak{E}$›
⟨*proof*⟩

2.18 Cleanup

unbundle *no cblinfun-syntax*
unbundle *no kraus-map-syntax*

end

3 Kraus maps

theory *Kraus-Maps*
  imports *Kraus-Families*
begin

3.1 Kraus maps

unbundle *kraus-map-syntax*
unbundle *cblinfun-syntax*

definition *kraus-map* :: ‹(('a::chilbert-space,'a) *trace-class* ⇒ ('b::chilbert-space,'b) *trace-class*) ⇒ *bool*› **where**
  kraus-map-def-raw: ‹*kraus-map* $\mathfrak{E} \longleftrightarrow$ ($\exists$ *EE* :: ('a,'b,unit) *kraus-family*. $\mathfrak{E}$ = *kf-apply EE*)›

lemma *kraus-map-def*: ‹*kraus-map* $\mathfrak{E} \longleftrightarrow$ ($\exists$ *EE* :: ('a::chilbert-space,'b::chilbert-space,'x) *kraus-family*. $\mathfrak{E}$ = *kf-apply EE*)›
  — Has a more general type than the original definition
⟨*proof*⟩

lemma *kraus-mapI*:
  assumes ‹$\mathfrak{E}$ = *kf-apply* $\mathfrak{E}'$›
  shows ‹*kraus-map* $\mathfrak{E}$›
  ⟨*proof*⟩

lemma *kraus-map-bounded-clinear*:
  ‹*bounded-clinear* $\mathfrak{E}$› **if** ‹*kraus-map* $\mathfrak{E}$›
  ⟨*proof*⟩

lemma *kraus-map-pos*:
  assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹$\varrho \geq 0$›
  shows ‹$\mathfrak{E}\ \varrho \geq 0$›
⟨*proof*⟩

lemma *kraus-map-mono*:
  assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹$\varrho \geq \tau$›
  shows ‹$\mathfrak{E}\ \varrho \geq \mathfrak{E}\ \tau$›
  ⟨*proof*⟩

lemma *kraus-map-kf-apply*[*iff*]: ‹*kraus-map* (*kf-apply* $\mathfrak{E}$)›
  ⟨*proof*⟩

definition *km-some-kraus-family* :: ‹(('*a*::*chilbert-space*, '*a*) *trace-class* ⇒ ('*b*::*chilbert-space*, '*b*) *trace-class*) ⇒ ('*a*, '*b*, *unit*) *kraus-family*› **where**
  ‹*km-some-kraus-family* $\mathfrak{E}$ = (*if kraus-map* $\mathfrak{E}$ *then SOME* $\mathfrak{F}$. $\mathfrak{E}$ = *kf-apply* $\mathfrak{F}$ *else 0*)›

lemma *kf-apply-km-some-kraus-family*[*simp*]:
  assumes ‹*kraus-map* $\mathfrak{E}$›
  shows ‹*kf-apply* (*km-some-kraus-family* $\mathfrak{E}$) = $\mathfrak{E}$›
  ⟨*proof*⟩

lemma *km-some-kraus-family-invalid*:
  assumes ‹¬ *kraus-map* $\mathfrak{E}$›
  shows ‹*km-some-kraus-family* $\mathfrak{E}$ = *0*›
  ⟨*proof*⟩

definition *km-operators-in* :: ‹(('*a*::*chilbert-space*,'*a*) *trace-class* ⇒ ('*b*::*chilbert-space*,'*b*) *trace-class*) ⇒ ('*a* $\Rightarrow_{CL}$ '*b*) *set* ⇒ *bool*› **where**
  ‹*km-operators-in* $\mathfrak{E}$ *S* ⟷ (∃ $\mathfrak{F}$ :: ('*a*,'*b*,*unit*) *kraus-family*. *kf-apply* $\mathfrak{F}$ = $\mathfrak{E}$ ∧ *kf-operators* $\mathfrak{F}$ ⊆ *S*)›

lemma *km-operators-in-mono*: ‹*S* ⊆ *T* ⟹ *km-operators-in* $\mathfrak{E}$ *S* ⟹ *km-operators-in* $\mathfrak{E}$ *T*›
  ⟨*proof*⟩

lemma *km-operators-in-kf-apply*:
  assumes ‹*span* (*kf-operators* $\mathfrak{E}$) ⊆ *S*›
  shows ‹*km-operators-in* (*kf-apply* $\mathfrak{E}$) *S*›
⟨*proof*⟩

lemma *km-operators-in-kf-apply-flattened*:
  fixes $\mathfrak{E}$ :: ‹('*a*::*chilbert-space*,'*b*::*chilbert-space*,'*x*::*CARD-1*) *kraus-family*›
  assumes ‹*kf-operators* $\mathfrak{E}$ ⊆ *S*›
  shows ‹*km-operators-in* (*kf-apply* $\mathfrak{E}$) *S*›
⟨*proof*⟩

lemma *km-commute*:
  assumes ‹*km-operators-in* $\mathfrak{E}$ *S*›
  assumes ‹*km-operators-in* $\mathfrak{F}$ *T*›
  assumes ‹*S* ⊆ *commutant T*›
  shows ‹$\mathfrak{F}$ *o* $\mathfrak{E}$ = $\mathfrak{E}$ *o* $\mathfrak{F}$›
⟨*proof*⟩

lemma *km-operators-in-UNIV*:

assumes ‹*kraus-map* $\mathfrak{E}$›
shows ‹*km-operators-in* $\mathfrak{E}$ *UNIV*›
⟨*proof*⟩

lemma *separating-kraus-map-bounded-clinear*:
fixes *S* :: ‹(′*a*::*chilbert-space*,′*a*) *trace-class set*›
assumes ‹*separating-set* (*bounded-clinear* :: (- ⇒ (′*b*::*chilbert-space*,′*b*) *trace-class*) ⇒ -) *S*›
shows ‹*separating-set* (*kraus-map* :: (- ⇒ (′*b*::*chilbert-space*,′*b*) *trace-class*) ⇒ -) *S*›
⟨*proof*⟩

3.2 Bound and norm

definition *km-bound* :: ‹((′*a*::*chilbert-space*, ′*a*) *trace-class* ⇒ (′*b*::*chilbert-space*, ′*b*) *trace-class*) ⇒ (′*a*, ′*a*) *cblinfun*› **where**
‹*km-bound* $\mathfrak{E}$ = (*if* ∃$\mathfrak{E}'$ :: (-, -, *unit*) *kraus-family*. $\mathfrak{E}$ = *kf-apply* $\mathfrak{E}'$ *then kf-bound* (*SOME* $\mathfrak{E}'$ :: (-, -, *unit*) *kraus-family*. $\mathfrak{E}$ = *kf-apply* $\mathfrak{E}'$) *else 0*)›

lemma *km-bound-kf-bound*:
assumes ‹$\mathfrak{E}$ = *kf-apply* $\mathfrak{F}$›
shows ‹*km-bound* $\mathfrak{E}$ = *kf-bound* $\mathfrak{F}$›
⟨*proof*⟩

definition *km-norm* :: ‹((′*a*::*chilbert-space*, ′*a*) *trace-class* ⇒ (′*b*::*chilbert-space*, ′*b*) *trace-class*) ⇒ *real*› **where**
‹*km-norm* $\mathfrak{E}$ = *norm* (*km-bound* $\mathfrak{E}$)›

lemma *km-norm-kf-norm*:
assumes ‹$\mathfrak{E}$ = *kf-apply* $\mathfrak{F}$›
shows ‹*km-norm* $\mathfrak{E}$ = *kf-norm* $\mathfrak{F}$›
⟨*proof*⟩

lemma *km-bound-invalid*:
assumes ‹¬ *kraus-map* $\mathfrak{E}$›
shows ‹*km-bound* $\mathfrak{E}$ = *0*›
⟨*proof*⟩

lemma *km-norm-invalid*:
assumes ‹¬ *kraus-map* $\mathfrak{E}$›
shows ‹*km-norm* $\mathfrak{E}$ = *0*›
⟨*proof*⟩

lemma *km-norm-geq0*[*iff*]: ‹*km-norm* $\mathfrak{E}$ ≥ *0*›
⟨*proof*⟩

lemma *kf-bound-pos*[*iff*]: ‹*km-bound* $\mathfrak{E}$ ≥ *0*›
⟨*proof*⟩

lemma *km-bounded-pos*:
  assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹$\varrho \geq 0$›
  shows ‹*norm* ($\mathfrak{E}$ ϱ) $\leq$ *km-norm* $\mathfrak{E}$ $*$ *norm* ϱ›
⟨*proof*⟩

lemma *km-bounded*:
  assumes ‹*kraus-map* $\mathfrak{E}$›
  shows ‹*norm* ($\mathfrak{E}$ ϱ) $\leq$ *4* $*$ *km-norm* $\mathfrak{E}$ $*$ *norm* ϱ›
⟨*proof*⟩

lemma *km-bound-from-map*:
  assumes ‹*kraus-map* $\mathfrak{E}$›
  shows ‹$\psi \bullet_C$ *km-bound* $\mathfrak{E}$ φ = *trace-tc* ($\mathfrak{E}$ (*tc-butterfly* φ ψ))›
  ⟨*proof*⟩

lemma *trace-from-km-bound*:
  assumes ‹*kraus-map* $\mathfrak{E}$›
  shows ‹*trace-tc* ($\mathfrak{E}$ ϱ) = *trace-tc* (*compose-tcr* (*km-bound* $\mathfrak{E}$) ϱ)›
  ⟨*proof*⟩

lemma *km-bound-selfadjoint*[*iff*]: ‹*selfadjoint* (*km-bound* $\mathfrak{E}$)›
  ⟨*proof*⟩

lemma *km-bound-leq-km-norm-id*: ‹*km-bound* $\mathfrak{E}$ $\leq$ *km-norm* $\mathfrak{E}$ $*_R$ *id-cblinfun*›
  ⟨*proof*⟩

lemma *kf-norm-km-some-kraus-family*[*simp*]: ‹*kf-norm* (*km-some-kraus-family* $\mathfrak{E}$) = *km-norm* $\mathfrak{E}$›
  ⟨*proof*⟩

3.3 Basic Kraus maps

Zero map and constant maps. Addition and rescaling and composition of maps.

lemma *kraus-map-0*[*iff*]: ‹*kraus-map 0*›
  ⟨*proof*⟩

lemma *kraus-map-0'*[*iff*]: ‹*kraus-map* (λ-. *0*)›
  ⟨*proof*⟩
lemma *km-bound-0*[*simp*]: ‹*km-bound 0* = *0*›
  ⟨*proof*⟩

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

lemma *km-some-kraus-family-0*[*simp*]: ‹*km-some-kraus-family 0* = *0*›
  ⟨*proof*⟩

lemma *kraus-map-id*[*iff*]: ‹*kraus-map id*›
  ⟨*proof*⟩

lemma *km-bound-id*[*simp*]: ‹*km-bound id* = *id-cblinfun*›
  ⟨*proof*⟩

lemma *km-norm-id-leq1*[*iff*]: ‹*km-norm id* ≤ *1*›
  ⟨*proof*⟩

lemma *km-norm-id-eq1*[*simp*]: ‹*km-norm* (*id* :: (*'a* :: {*chilbert-space*, *not-singleton*}, *'a*) *trace-class* ⇒ -) = *1*›
  ⟨*proof*⟩

lemma *km-operators-in-id*[*iff*]: ‹*km-operators-in id* {*id-cblinfun*}›
  ⟨*proof*⟩

lemma *kraus-map-add*[*iff*]:
  assumes ‹*kraus-map* 𝔈› **and** ‹*kraus-map* 𝔉›
  shows ‹*kraus-map* (λϱ. 𝔈 ϱ + 𝔉 ϱ)›
⟨*proof*⟩

lemma *kraus-map-plus'*[*iff*]:
  assumes ‹*kraus-map* 𝔈› **and** ‹*kraus-map* 𝔉›
  shows ‹*kraus-map* (𝔈 + 𝔉)›
  ⟨*proof*⟩

lemma *km-bound-plus*:
  assumes ‹*kraus-map* 𝔈› **and** ‹*kraus-map* 𝔉›
  shows ‹*km-bound* (𝔈 + 𝔉) = *km-bound* 𝔈 + *km-bound* 𝔉›
⟨*proof*⟩

lemma *km-norm-triangle*:
  assumes ‹*kraus-map* 𝔈› **and** ‹*kraus-map* 𝔉›
  shows ‹*km-norm* (𝔈 + 𝔉) ≤ *km-norm* 𝔈 + *km-norm* 𝔉›
  ⟨*proof*⟩

lemma *kraus-map-constant*[*iff*]: ‹*kraus-map* (λσ. *trace-tc* σ $*_C$ ϱ)› **if** ‹ϱ ≥ *0*›
  ⟨*proof*⟩

lemma *kraus-map-constant-invalid*:
  ‹¬ *kraus-map* (λσ :: (*'a*::{*chilbert-space*,*not-singleton*},*'a*) *trace-class*. *trace-tc* σ $*_C$ ϱ)› **if** ‹∼ ϱ ≥ *0*›
⟨*proof*⟩

lemma *kraus-map-scale*:
  assumes ‹*kraus-map* 𝔈› **and** ‹*c* ≥ *0*›
  shows ‹*kraus-map* (λϱ. *c* $*_R$ 𝔈 ϱ)›
⟨*proof*⟩

lemma *km-bound-scale*[*simp*]: ‹*km-bound* ($\lambda\varrho$. $c *_R \mathfrak{E}\ \varrho$) = $c *_R$ *km-bound* $\mathfrak{E}$› **if** ‹$c \geq 0$›
⟨*proof*⟩

lemma *km-norm-scale*[*simp*]: ‹*km-norm* ($\lambda\varrho$. $c *_R \mathfrak{E}\ \varrho$) = $c *$ *km-norm* $\mathfrak{E}$› **if** ‹$c \geq 0$›
⟨*proof*⟩

lemma *kraus-map-sandwich*[*iff*]: ‹*kraus-map* (*sandwich-tc* A)›
⟨*proof*⟩

lemma *km-bound-sandwich*[*simp*]: ‹*km-bound* (*sandwich-tc* A) = $A* \ o_{CL}\ A$›
⟨*proof*⟩

lemma *km-norm-sandwich*[*simp*]: ‹*km-norm* (*sandwich-tc* A) = (*norm* $A)^2$›
⟨*proof*⟩

lemma *km-operators-in-sandwich*: ‹*km-operators-in* (*sandwich-tc* U) $\{U\}$›
⟨*proof*⟩

lemma *km-constant-bound*[*simp*]: ‹*km-bound* ($\lambda\sigma$. *trace-tc* $\sigma *_C \varrho$) = *norm* $\varrho *_R$ *id-cblinfun*› **if** ‹$\varrho \geq 0$›
⟨*proof*⟩

lemma *km-constant-norm*[*simp*]: ‹*km-norm* ($\lambda\sigma$::(*'a*::{*chilbert-space*,*not-singleton*},*'a*) *trace-class*. *trace-tc* $\sigma *_C \varrho$) = *norm* ϱ› **if** ‹$\varrho \geq 0$›
⟨*proof*⟩

lemma *km-constant-norm-leq*[*simp*]: ‹*km-norm* ($\lambda\sigma$::(*'a*::*chilbert-space*,*'a*) *trace-class*. *trace-tc* $\sigma *_C \varrho$) $\leq$ *norm* ϱ›
⟨*proof*⟩

lemma *kraus-map-comp*:
assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹*kraus-map* $\mathfrak{F}$›
shows ‹*kraus-map* ($\mathfrak{E}$ *o* $\mathfrak{F}$)›
⟨*proof*⟩

lemma *km-comp-norm-leq*:
assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹*kraus-map* $\mathfrak{F}$›
shows ‹*km-norm* ($\mathfrak{E}$ *o* $\mathfrak{F}$) $\leq$ *km-norm* $\mathfrak{E}$ $*$ *km-norm* $\mathfrak{F}$›
⟨*proof*⟩

lemma *km-bound-comp-sandwich*:
assumes ‹*kraus-map* $\mathfrak{E}$›
shows ‹*km-bound* ($\lambda\varrho$. $\mathfrak{E}$ (*sandwich-tc* U ϱ)) = *sandwich* ($U*$) (*km-bound* $\mathfrak{E}$)›
⟨*proof*⟩

lemma *km-norm-comp-sandwich-coiso*:
  assumes ‹*isometry* $(U*)$›
  shows ‹*km-norm* $(\lambda\varrho.\ \mathfrak{E}\ (sandwich\text{-}tc\ U\ \varrho)) = km\text{-}norm\ \mathfrak{E}$›
⟨*proof*⟩

lemma *km-bound-comp-sandwich-iso*:
  assumes ‹*isometry U*›
  shows ‹*km-bound* $(\lambda\varrho.\ sandwich\text{-}tc\ U\ (\mathfrak{E}\ \varrho)) = km\text{-}bound\ \mathfrak{E}$›
⟨*proof*⟩

lemma *km-norm-comp-sandwich-iso*:
  assumes ‹*isometry U*›
  shows ‹*km-norm* $(\lambda\varrho.\ sandwich\text{-}tc\ U\ (\mathfrak{E}\ \varrho)) = km\text{-}norm\ \mathfrak{E}$›
⟨*proof*⟩

lemma *kraus-map-sum*:
  assumes ‹$\bigwedge x.\ x \in A \Longrightarrow kraus\text{-}map\ (\mathfrak{E}\ x)$›
  shows ‹*kraus-map* $(\sum x \in A.\ \mathfrak{E}\ x)$›
  ⟨*proof*⟩

lemma *km-bound-sum*:
  assumes ‹$\bigwedge x.\ x \in A \Longrightarrow kraus\text{-}map\ (\mathfrak{E}\ x)$›
  shows ‹*km-bound* $(\sum x \in A.\ \mathfrak{E}\ x) = (\sum x \in A.\ km\text{-}bound\ (\mathfrak{E}\ x))$›
⟨*proof*⟩

3.4 Infinite sums

lemma
  assumes ‹$\bigwedge\varrho.\ ((\lambda a.\ sandwich\text{-}tc\ (f\ a)\ \varrho)\ has\text{-}sum\ \mathfrak{E}\ \varrho)\ A$›
  defines ‹$EE \equiv Set.filter\ (\lambda(E,\text{-}).\ E \neq 0)\ ((\lambda a.\ (f\ a,\ a))\ `\ A)$›
  shows *kraus-mapI-sum*: ‹*kraus-map* $\mathfrak{E}$›
    and *kraus-map-sum-kraus-family*: ‹*kraus-family EE*›
    and *kraus-map-sum-kf-apply*: ‹$\mathfrak{E} = kf\text{-}apply\ (Abs\text{-}kraus\text{-}family\ EE)$›
⟨*proof*⟩

lemma *kraus-map-infsum-sandwich*:
  assumes ‹$\bigwedge\varrho.\ (\lambda a.\ sandwich\text{-}tc\ (f\ a)\ \varrho)\ summable\text{-}on\ A$›
  shows ‹*kraus-map* $(\lambda\varrho.\ \sum_\infty a \in A.\ sandwich\text{-}tc\ (f\ a)\ \varrho)$›
  ⟨*proof*⟩

lemma *kraus-map-sum-sandwich*: ‹*kraus-map* $(\lambda\varrho.\ \sum a \in A.\ sandwich\text{-}tc\ (f\ a)\ \varrho)$›
  ⟨*proof*⟩

lemma *kraus-map-as-infsum*:
  assumes ‹*kraus-map* $\mathfrak{E}$›
  shows ‹$\exists M. \forall \varrho.$ (($\lambda E.$ *sandwich-tc* E ϱ) *has-sum* $\mathfrak{E}$ ϱ) M›
⟨*proof*⟩

definition *km-summable* :: ‹('a ⇒ ('b::*chilbert-space*,'b) *trace-class* ⇒ ('c::*chilbert-space*,'c) *trace-class*) ⇒ 'a *set* ⇒ *bool*› **where**
  ‹*km-summable* $\mathfrak{E}$ A $\longleftrightarrow$ *summable-on-in cweak-operator-topology* ($\lambda a.$ *km-bound* ($\mathfrak{E}$ a)) A›

lemma *km-summable-kf-summable*:
  assumes ‹$\bigwedge a\ \varrho.\ a \in A \Longrightarrow \mathfrak{E}\ a\ \varrho = \mathfrak{F}\ a *_{kr} \varrho$›
  shows ‹*km-summable* $\mathfrak{E}$ A $\longleftrightarrow$ *kf-summable* $\mathfrak{F}$ A›
⟨*proof*⟩

lemma *km-summable-summable*:
  assumes *km*: ‹$\bigwedge a.\ a{\in}A \Longrightarrow$ *kraus-map* ($\mathfrak{E}$ a)›
  assumes *sum*: ‹*km-summable* $\mathfrak{E}$ A›
  shows ‹($\lambda a.$ $\mathfrak{E}$ a ϱ) *summable-on* A›
⟨*proof*⟩

lemma *kraus-map-infsum*:
  assumes *km*: ‹$\bigwedge a.\ a{\in}A \Longrightarrow$ *kraus-map* ($\mathfrak{E}$ a)›
  assumes *sum*: ‹*km-summable* $\mathfrak{E}$ A›
  shows ‹*kraus-map* ($\lambda\varrho.\ \sum_\infty a{\in}A.\ \mathfrak{E}\ a\ \varrho$)›
⟨*proof*⟩

lemma *km-bound-infsum*:
  assumes *km*: ‹$\bigwedge a.\ a{\in}A \Longrightarrow$ *kraus-map* ($\mathfrak{E}$ a)›
  assumes *sum*: ‹*km-summable* $\mathfrak{E}$ A›
  shows ‹*km-bound* ($\lambda\varrho.\ \sum_\infty a{\in}A.\ \mathfrak{E}\ a\ \varrho$) = *infsum-in cweak-operator-topology* ($\lambda a.$ *km-bound* ($\mathfrak{E}$ a)) A›
⟨*proof*⟩

lemma *km-norm-infsum*:
  assumes *km*: ‹$\bigwedge a.\ a{\in}A \Longrightarrow$ *kraus-map* ($\mathfrak{E}$ a)›
  assumes *sum*: ‹($\lambda a.$ *km-norm* ($\mathfrak{E}$ a)) *summable-on* A›
  shows ‹*km-norm* ($\lambda\varrho.\ \sum_\infty a{\in}A.\ \mathfrak{E}\ a\ \varrho$) $\leq$ ($\sum_\infty a{\in}A.$ *km-norm* ($\mathfrak{E}$ a))›
⟨*proof*⟩

lemma *kraus-map-has-sum*:
  assumes ‹$\bigwedge x.\ x \in A \Longrightarrow$ *kraus-map* ($\mathfrak{E}$ x)›
  assumes ‹*km-summable* $\mathfrak{E}$ A›
  assumes ‹($\mathfrak{E}$ *has-sum* $\mathfrak{F}$) A›
  shows ‹*kraus-map* $\mathfrak{F}$›

⟨*proof*⟩

lemma *km-summable-iff-sums-to-kraus-map*:
  assumes ‹$\bigwedge a.\ a{\in}A \Longrightarrow$ *kraus-map* ($\mathfrak{E}$ *a*)›
  shows ‹*km-summable* $\mathfrak{E}$ *A* $\longleftrightarrow$ ($\exists\,\mathfrak{F}$. ($\forall\,t$. ((λx. $\mathfrak{E}$ *x t*) *has-sum* $\mathfrak{F}$ *t*) *A*) $\wedge$ *kraus-map* $\mathfrak{F}$)›
⟨*proof*⟩

3.5 Tensor products

definition *km-tensor-exists* :: ‹(('*a ell2*, '*b ell2*) *trace-class* ⇒ ('*c ell2*, '*d ell2*) *trace-class*)
  ⇒ (('*e ell2*, '*f ell2*) *trace-class* ⇒ ('*g ell2*, '*h ell2*) *trace-class*) ⇒ *bool*›
where
  ‹*km-tensor-exists* $\mathfrak{E}$ $\mathfrak{F}$ $\longleftrightarrow$ ($\exists\,\mathfrak{E}\mathfrak{F}$. *bounded-clinear* $\mathfrak{E}\mathfrak{F}$ $\wedge$ ($\forall\,\varrho\ \sigma$. $\mathfrak{E}\mathfrak{F}$ (*tc-tensor* ϱ σ) = *tc-tensor* ($\mathfrak{E}$ ϱ) ($\mathfrak{F}$ σ)))›

definition *km-tensor* :: ‹(('*a ell2*, '*c ell2*) *trace-class* ⇒ ('*e ell2*, '*g ell2*) *trace-class*)
  ⇒ (('*b ell2*, '*d ell2*) *trace-class* ⇒ ('*f ell2*, '*h ell2*) *trace-class*)
  ⇒ (('*a* × '*b*) *ell2*, ('*c* × '*d*) *ell2*) *trace-class* ⇒ (('*e* × '*f*) *ell2*, ('*g* × '*h*) *ell2*) *trace-class*› **where**
  ‹*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$ = (*if km-tensor-exists* $\mathfrak{E}$ $\mathfrak{F}$
    then SOME $\mathfrak{E}\mathfrak{F}$. *bounded-clinear* $\mathfrak{E}\mathfrak{F}$ $\wedge$ ($\forall\,\varrho\ \sigma$. $\mathfrak{E}\mathfrak{F}$ (*tc-tensor* ϱ σ) = *tc-tensor* ($\mathfrak{E}$ ϱ) ($\mathfrak{F}$ σ))
    else 0)›

lemma *km-tensor-invalid*:
  assumes ‹¬ *km-tensor-exists* $\mathfrak{E}$ $\mathfrak{F}$›
  shows ‹*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$ = *0*›
  ⟨*proof*⟩

lemma *km-tensor-exists-bounded-clinear*[*iff*]:
  assumes ‹*km-tensor-exists* $\mathfrak{E}$ $\mathfrak{F}$›
  shows ‹*bounded-clinear* (*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
  ⟨*proof*⟩

lemma *km-tensor-apply*[*simp*]:
  assumes ‹*km-tensor-exists* $\mathfrak{E}$ $\mathfrak{F}$›
  shows ‹*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$ (*tc-tensor* ϱ σ) = *tc-tensor* ($\mathfrak{E}$ ϱ) ($\mathfrak{F}$ σ)›
  ⟨*proof*⟩

lemma *km-tensor-unique*:
  assumes ‹*bounded-clinear* $\mathfrak{E}\mathfrak{F}$›
  assumes ‹$\bigwedge\varrho\ \sigma$. $\mathfrak{E}\mathfrak{F}$ (*tc-tensor* ϱ σ) = *tc-tensor* ($\mathfrak{E}$ ϱ) ($\mathfrak{F}$ σ)›
  shows ‹$\mathfrak{E}\mathfrak{F}$ = *km-tensor* $\mathfrak{E}$ $\mathfrak{F}$›
⟨*proof*⟩

lemma *km-tensor-kf-tensor*: ‹*km-tensor* (*kf-apply* $\mathfrak{E}$) (*kf-apply* $\mathfrak{F}$) = *kf-apply* (*kf-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
  ⟨*proof*⟩

lemma *km-tensor-kraus-map*:
  assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹*kraus-map* $\mathfrak{F}$›
  shows ‹*kraus-map* (*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$)›
⟨*proof*⟩

lemma *km-tensor-kraus-map-exists*:
  assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹*kraus-map* $\mathfrak{F}$›
  shows ‹*km-tensor-exists* $\mathfrak{E}$ $\mathfrak{F}$›
⟨*proof*⟩

lemma *km-tensor-as-infsum*:
  assumes ‹$\bigwedge\varrho$. ((λi. *sandwich-tc* (*E i*) ϱ) *has-sum* $\mathfrak{E}$ ϱ) *I*›
  assumes ‹$\bigwedge\varrho$. ((λj. *sandwich-tc* (*F j*) ϱ) *has-sum* $\mathfrak{F}$ ϱ) *J*›
  shows ‹*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$ ϱ = ($\sum_\infty(i,j)\in I\times J$. *sandwich-tc* (*E i* $\otimes_o$ *F j*) ϱ)›
⟨*proof*⟩

lemma *km-bound-tensor*:
  assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹*kraus-map* $\mathfrak{F}$›
  shows ‹*km-bound* (*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$) = *km-bound* $\mathfrak{E}$ $\otimes_o$ *km-bound* $\mathfrak{F}$›
⟨*proof*⟩

lemma *km-norm-tensor*:
  assumes ‹*kraus-map* $\mathfrak{E}$› **and** ‹*kraus-map* $\mathfrak{F}$›
  shows ‹*km-norm* (*km-tensor* $\mathfrak{E}$ $\mathfrak{F}$) = *km-norm* $\mathfrak{E}$ $*$ *km-norm* $\mathfrak{F}$›
⟨*proof*⟩

lemma *km-tensor-compose-distrib*:
  assumes ‹*km-tensor-exists* $\mathfrak{E}$ $\mathfrak{G}$› **and** ‹*km-tensor-exists* $\mathfrak{F}$ $\mathfrak{H}$›
  shows ‹*km-tensor* ($\mathfrak{E}$ *o* $\mathfrak{F}$) ($\mathfrak{G}$ *o* $\mathfrak{H}$) = *km-tensor* $\mathfrak{E}$ $\mathfrak{G}$ *o* *km-tensor* $\mathfrak{F}$ $\mathfrak{H}$›
  ⟨*proof*⟩

lemma *kraus-map-tensor-right*[*simp*]:
  assumes ‹$\varrho \geq 0$›
  shows ‹*kraus-map* ($\lambda\sigma$. *tc-tensor* σ ϱ)›
  ⟨*proof*⟩
lemma *kraus-map-tensor-left*[*simp*]:
  assumes ‹$\varrho \geq 0$›
  shows ‹*kraus-map* ($\lambda\sigma$. *tc-tensor* ϱ σ)›
  ⟨*proof*⟩

lemma *km-bound-tensor-right*[*simp*]:
  assumes ‹$\varrho \geq 0$›
  shows ‹*km-bound* ($\lambda\sigma$. *tc-tensor* σ ϱ) = *norm* ϱ $*_C$ *id-cblinfun*›
  ⟨*proof*⟩
lemma *km-bound-tensor-left*[*simp*]:
  assumes ‹$\varrho \geq 0$›
  shows ‹*km-bound* ($\lambda\sigma$. *tc-tensor* ϱ σ) = *norm* ϱ $*_C$ *id-cblinfun*›

⟨*proof*⟩

lemma *kf-norm-tensor-right*[*simp*]:
assumes ‹$\varrho \geq 0$›
shows ‹*km-norm* (*λσ. tc-tensor σ ϱ*) = *norm ϱ*›
⟨*proof*⟩

lemma *kf-norm-tensor-left*[*simp*]:
assumes ‹$\varrho \geq 0$›
shows ‹*km-norm* (*λσ. tc-tensor ϱ σ*) = *norm ϱ*›
⟨*proof*⟩

lemma *km-operators-in-tensor*:
assumes ‹*km-operators-in* 𝔈 *S*›
assumes ‹*km-operators-in* 𝔉 *T*›
shows ‹*km-operators-in* (*km-tensor* 𝔈 𝔉) (*span* {$s \otimes_o t$ | *s t. s*∈*S* ∧ *t*∈*T*})›
⟨*proof*⟩

lemma *km-tensor-sandwich-tc*:
‹*km-tensor* (*sandwich-tc A*) (*sandwich-tc B*) = *sandwich-tc* ($A \otimes_o B$)›
⟨*proof*⟩

3.6 Trace and partial trace

definition ‹*km-trace-preserving* 𝔈 ⟷ (∃𝔉::(-,-,*unit*) *kraus-family*. 𝔈 = *kf-apply* 𝔉 ∧ *kf-trace-preserving* 𝔉)›
lemma *km-trace-preserving-def′*: ‹*km-trace-preserving* 𝔈 ⟷ (∃𝔉::(-, -, ′*c*) *kraus-family*. 𝔈 = *kf-apply* 𝔉 ∧ *kf-trace-preserving* 𝔉)›
— Has a more general type than *km-trace-preserving-def*
⟨*proof*⟩

definition *km-trace-reducing-def*: ‹*km-trace-reducing* 𝔈 ⟷ (∃𝔉::(-,-,*unit*) *kraus-family*. 𝔈 = *kf-apply* 𝔉 ∧ *kf-trace-reducing* 𝔉)›
lemma *km-trace-reducing-def′*: ‹*km-trace-reducing* 𝔈 ⟷ (∃𝔉::(-, -, ′*c*) *kraus-family*. 𝔈 = *kf-apply* 𝔉 ∧ *kf-trace-reducing* 𝔉)›
⟨*proof*⟩

lemma *km-trace-preserving-apply*[*simp*]: ‹*km-trace-preserving* (*kf-apply* 𝔈) = *kf-trace-preserving* 𝔈›
⟨*proof*⟩

lemma *km-trace-reducing-apply*[*simp*]: ‹*km-trace-reducing* (*kf-apply* 𝔈) = *kf-trace-reducing* 𝔈›
⟨*proof*⟩

lemma *km-trace-preserving-iff*: ‹*km-trace-preserving* 𝔈 ⟷ *kraus-map* 𝔈 ∧ (∀ ϱ. *trace-tc* (𝔈 ϱ) = *trace-tc* ϱ)›
⟨*proof*⟩

lemma *km-trace-reducing-iff*: ‹*km-trace-reducing* $\mathfrak{E}$ ⟷ *kraus-map* $\mathfrak{E}$ ∧ (∀ ϱ≥*0*. *trace-tc* ($\mathfrak{E}$ ϱ) ≤ *trace-tc* ϱ)›
⟨*proof*⟩

lemma *km-trace-preserving-imp-reducing*:
assumes ‹*km-trace-preserving* $\mathfrak{E}$›
shows ‹*km-trace-reducing* $\mathfrak{E}$›
⟨*proof*⟩

lemma *km-trace-preserving-id*[*iff*]: ‹*km-trace-preserving id*›
⟨*proof*⟩

lemma *km-trace-reducing-iff-norm-leq1*: ‹*km-trace-reducing* $\mathfrak{E}$ ⟷ *kraus-map* $\mathfrak{E}$ ∧ *km-norm* $\mathfrak{E}$ ≤ *1*›
⟨*proof*⟩

lemma *km-trace-preserving-iff-bound-id*: ‹*km-trace-preserving* $\mathfrak{E}$ ⟷ *kraus-map* $\mathfrak{E}$ ∧ *km-bound* $\mathfrak{E}$ = *id-cblinfun*›
⟨*proof*⟩

lemma *km-trace-preserving-iff-bound-id′*:
fixes $\mathfrak{E}$ :: ‹(′*a*::{*chilbert-space*, *not-singleton*}, ′*a*) *trace-class* ⇒ -›
shows ‹*km-trace-preserving* $\mathfrak{E}$ ⟷ *km-bound* $\mathfrak{E}$ = *id-cblinfun*›
⟨*proof*⟩

lemma *km-trace-norm-preserving*: ‹*km-norm* $\mathfrak{E}$ ≤ *1*› **if** ‹*km-trace-preserving* $\mathfrak{E}$›
⟨*proof*⟩

lemma *km-trace-norm-preserving-eq*:
fixes $\mathfrak{E}$ :: ‹(′*a*::{*chilbert-space*,*not-singleton*},′*a*) *trace-class* ⇒ (′*b*::*chilbert-space*,′*b*) *trace-class*›
assumes ‹*km-trace-preserving* $\mathfrak{E}$›
shows ‹*km-norm* $\mathfrak{E}$ = *1*›
⟨*proof*⟩

lemma *kraus-map-trace*: ‹*kraus-map* (*one-dim-iso o trace-tc*)›
⟨*proof*⟩

lemma *trace-preserving-trace-kraus-map*[*iff*]: ‹*km-trace-preserving* (*one-dim-iso o trace-tc*)›
⟨*proof*⟩

lemma *km-trace-bound*[*simp*]: ‹*km-bound* (*one-dim-iso o trace-tc*) = *id-cblinfun*›
⟨*proof*⟩

lemma *km-trace-norm-eq1*[*simp*]: ‹*km-norm* (*one-dim-iso o trace-tc* :: (′*a*::{*chilbert-space*,*not-singleton*},′*a*) *trace-class* ⇒ -) = *1*›
⟨*proof*⟩

lemma *km-trace-norm-leq1*[*simp*]: ‹*km-norm* (*one-dim-iso o trace-tc*) ≤ *1*›
⟨*proof*⟩

lemma *kraus-map-partial-trace*[*iff*]: ‹*kraus-map partial-trace*›
⟨*proof*⟩

lemma *partial-trace-ignores-kraus-map*:
assumes ‹*km-trace-preserving* 𝔈›
assumes ‹*kraus-map* 𝔉›
shows ‹*partial-trace* (*km-tensor* 𝔉 𝔈 ϱ) = 𝔉 (*partial-trace* ϱ)›
⟨*proof*⟩

lemma *km-partial-trace-bound*[*simp*]: ‹*km-bound partial-trace* = *id-cblinfun*›
⟨*proof*⟩

lemma *km-partial-trace-norm*[*simp*]:
shows ‹*km-norm partial-trace* = *1*›
⟨*proof*⟩

lemma *km-trace-preserving-tensor*:
assumes ‹*km-trace-preserving* 𝔈› **and** ‹*km-trace-preserving* 𝔉›
shows ‹*km-trace-preserving* (*km-tensor* 𝔈 𝔉)›
⟨*proof*⟩

lemma *km-trace-reducing-tensor*:
assumes ‹*km-trace-reducing* 𝔈› **and** ‹*km-trace-reducing* 𝔉›
shows ‹*km-trace-reducing* (*km-tensor* 𝔈 𝔉)›
⟨*proof*⟩

3.7 Complete measurements

definition ‹*km-complete-measurement B* ϱ = ($\sum_\infty$*x*∈*B*. *sandwich-tc* (*selfbutter* (*sgn x*)) ϱ)›
abbreviation ‹*km-complete-measurement-ket* ≡ *km-complete-measurement* (*range ket*)›

lemma *km-complete-measurement-kf-complete-measurement*: ‹*km-complete-measurement B* = *kf-apply* (*kf-complete-measurement B*)› **if** ‹*is-ortho-set B*›
⟨*proof*⟩

lemma *km-complete-measurement-ket-kf-complete-measurement-ket*: ‹*km-complete-measurement-ket* = *kf-apply kf-complete-measurement-ket*›
⟨*proof*⟩

lemma *km-complete-measurement-has-sum*:
assumes ‹*is-ortho-set B*›
shows ‹((λ*x*. *sandwich-tc* (*selfbutter* (*sgn x*)) ϱ) *has-sum km-complete-measurement B* ϱ) *B*›

⟨*proof*⟩

lemma *km-complete-measurement-ket-has-sum*:
‹((λ*x*. *sandwich-tc* (*selfbutter* (*ket x*)) ϱ) *has-sum km-complete-measurement-ket* ϱ) *UNIV*›
⟨*proof*⟩

lemma *km-bound-complete-measurement*:
assumes ‹*is-ortho-set B*›
shows ‹*km-bound* (*km-complete-measurement B*) ≤ *id-cblinfun*›
⟨*proof*⟩

lemma *km-norm-complete-measurement*:
assumes ‹*is-ortho-set B*›
shows ‹*km-norm* (*km-complete-measurement B*) ≤ *1*›
⟨*proof*⟩

lemma *km-bound-complete-measurement-onb*[*simp*]:
assumes ‹*is-onb B*›
shows ‹*km-bound* (*km-complete-measurement B*) = *id-cblinfun*›
⟨*proof*⟩

lemma *km-bound-complete-measurement-ket*[*simp*]: ‹*km-bound km-complete-measurement-ket* = *id-cblinfun*›
⟨*proof*⟩

lemma *km-norm-complete-measurement-onb*[*simp*]:
fixes *B* :: ‹'*a*::{*not-singleton*, *chilbert-space*} *set*›
assumes ‹*is-onb B*›
shows ‹*km-norm* (*km-complete-measurement B*) = *1*›
⟨*proof*⟩

lemma *km-norm-complete-measurement-ket*[*simp*]:
shows ‹*km-norm km-complete-measurement-ket* = *1*›
⟨*proof*⟩

lemma *kraus-map-complete-measurement*:
assumes ‹*is-ortho-set B*›
shows ‹*kraus-map* (*km-complete-measurement B*)›
⟨*proof*⟩

lemma *kraus-map-complete-measurement-ket*[*iff*]:
shows ‹*kraus-map km-complete-measurement-ket*›
⟨*proof*⟩

lemma *km-complete-measurement-idem*[*simp*]:
assumes ‹*is-ortho-set B*›
shows ‹*km-complete-measurement B* (*km-complete-measurement B* ϱ) = *km-complete-measurement B* ϱ›
⟨*proof*⟩

lemma *km-complete-measurement-ket-idem*[*simp*]:
‹*km-complete-measurement-ket* (*km-complete-measurement-ket* ϱ) = *km-complete-measurement-ket* ϱ›
⟨*proof*⟩

lemma *km-complete-measurement-has-sum-onb*:
assumes ‹*is-onb* B›
shows ‹((λx. *sandwich-tc* (*selfbutter* x) ϱ) *has-sum km-complete-measurement* B ϱ) B›
⟨*proof*⟩

lemma *km-complete-measurement-ket-diagonal-operator*[*simp*]:
‹*km-complete-measurement-ket* (*diagonal-operator-tc* f) = *diagonal-operator-tc* f›
⟨*proof*⟩

lemma *km-operators-complete-measurement*:
assumes ‹*is-ortho-set* B›
shows ‹*km-operators-in* (*km-complete-measurement* B) (*span* (*selfbutter* ‘ B))›
⟨*proof*⟩

lemma *km-operators-complete-measurement-ket*:
shows ‹*km-operators-in km-complete-measurement-ket* (*span* (*range* (λc. (*selfbutter* (*ket* c)))))›
⟨*proof*⟩

lemma *km-complete-measurement-ket-butterket*[*simp*]:
‹*km-complete-measurement-ket* (*tc-butterfly* (*ket* c) (*ket* c)) = *tc-butterfly* (*ket* c) (*ket* c)›
⟨*proof*⟩

lemma *km-complete-measurement-tensor*:
assumes ‹*is-ortho-set* B› **and** ‹*is-ortho-set* C›
shows ‹*km-tensor* (*km-complete-measurement* B) (*km-complete-measurement* C)
= *km-complete-measurement* (($\lambda(b,c)$. $b \otimes_s c$) ‘ ($B \times C$))›
⟨*proof*⟩

lemma *km-complete-measurement-ket-tensor*:
shows ‹*km-tensor* (*km-complete-measurement-ket* :: ('*a ell2*, -) *trace-class* ⇒ -) (*km-complete-measurement-ket* :: ('*b ell2*, -) *trace-class* ⇒ -)
= *km-complete-measurement-ket*›
⟨*proof*⟩

lemma *km-tensor-0-left*[*simp*]: ‹*km-tensor* (*0* :: ('*a ell2*, '*b ell2*) *trace-class* ⇒ ('*c ell2*, '*d ell2*) *trace-class*) $\mathfrak{E}$ = *0*›
⟨*proof*⟩

lemma *km-tensor-0-right*[*simp*]: ‹*km-tensor* $\mathfrak{E}$ (*0* :: ('*a ell2*, '*b ell2*) *trace-class* ⇒ ('*c ell2*, '*d ell2*) *trace-class*) = *0*›
⟨*proof*⟩

unbundle *no kraus-map-syntax*
unbundle *no cblinfun-syntax*

end

References

[1] K. Kraus. *States, effects, and operations: fundamental notions of quantum theory*, volume 190 of *LNP*. Springer, 1983. Defines Kraus maps in Section 3, Theorem 1. Only considers separable Hilbert spaces.