Quantum and Classical Registers[*]

Dominique Unruh

March 17, 2025

Abstract

A formalization of the theory of quantum and classical registers as developed by Unruh [Unr24]. In a nutshell, a register refers to a part of a larger memory or system that can be accessed independently. Registers can be constructed from other registers and several (compatible) registers can be composed. For more details, see [Unr24]. This formalization develops both the generic theory of registers as well as specific instantiations for classical and quantum registers.

Note: This document assumes familiarity with the theoretical background developed in [Unr24]. [Unr24] also describes this formalization and mentions some of the design choices and challenges.

Some of the theories are autogenerated (*Laws_Classical*, *Laws_Quantum*, *Laws_Complement_Quantum*). Use the Python script *instantiate_laws.py* to recreate them after changing any of the theories starting with *Laws* or *Axioms*. See [Unr24] for an explanation of this mechanism and the reasons for it.

Contents

[*]Supported by the ERC consolidator grant CerQuS (819317), the PRG team grant Secure Quantum Technology (PRG946) from the Estonian Research Council, and the Estonian Centre of Exellence in IT (EXCITE) funded by ERDF.

1 Axioms of registers

```
theory Axioms
  imports Main
begin

class domain
instance prod :: (domain,domain) domain
  ⟨proof⟩

typedecl 'a update
axiomatization comp-update :: 'a::domain update ⇒ 'a update ⇒ 'a update where
  comp-update-assoc: comp-update (comp-update a b) c = comp-update a (comp-update b c)
axiomatization id-update :: 'a::domain update where
  id-update-left: comp-update id-update a = a and
  id-update-right: comp-update a id-update = a

axiomatization preregister :: ‹('a::domain update ⇒ 'b::domain update) ⇒ bool›
axiomatization where
  comp-preregister: preregister F ⟹ preregister G ⟹ preregister (G ∘ F) and
  id-preregister: ‹preregister id›
for F :: ‹'a::domain update ⇒ 'b::domain update› and G :: ‹'b update ⇒ 'c::domain update›

axiomatization where
  preregister-mult-right: ‹preregister (λa. comp-update a z)› and
  preregister-mult-left: ‹preregister (λa. comp-update z a)›
    for z :: 'a::domain update

axiomatization tensor-update :: ‹'a::domain update ⇒ 'b::domain update ⇒ ('a×'b) update›
 where tensor-extensionality: preregister F ⟹ preregister G ⟹ (⋀a b. F (tensor-update a b) = G (tensor-update
a b)) ⟹ F = G
  for F G :: ‹('a×'b) update ⇒ 'c::domain update›

axiomatization where tensor-update-mult: ‹comp-update (tensor-update a c) (tensor-update b d) = tensor-update
(comp-update a b) (comp-update c d)›
  for a b :: ‹'a::domain update› and c d :: ‹'b::domain update›

axiomatization register :: ‹('a update ⇒ 'b update) ⇒ bool›
axiomatization where
  register-preregister: register F ⟹ preregister F and
  register-comp: register F ⟹ register G ⟹ register (G ∘ F)  and
  register-mult: register F ⟹ comp-update (F a) (F b) = F (comp-update a b) and
  register-of-id: ‹register F ⟹ F id-update = id-update› and
  register-id: ‹register (id :: 'a update ⇒ 'a update)›
for F :: 'a::domain update ⇒ 'b::domain update and G :: 'b update ⇒ 'c::domain update

axiomatization where register-tensor-left: ‹register (λa. tensor-update a id-update)›
axiomatization where register-tensor-right: ‹register (λa. tensor-update id-update a)›

axiomatization register-pair ::
  ‹('a::domain update ⇒ 'c::domain update) ⇒ ('b::domain update ⇒ 'c update)
         ⇒ (('a×'b) update ⇒ 'c update)› where
  register-pair-is-register: ‹register F ⟹ register G ⟹ (⋀a b. comp-update (F a) (G b) = comp-update (G b)
(F a))
        ⟹ register (register-pair F G)› and
  register-pair-apply: ‹register F ⟹ register G ⟹ (⋀a b. comp-update (F a) (G b) = comp-update (G b) (F
a))
        ⟹ (register-pair F G) (tensor-update a b) = comp-update (F a) (G b)›

end
```

2 Generic laws about registers

theory *Laws*
  imports *Axioms*
begin

This notation is only used inside this file

notation *comp-update* (**infixl** $*_u$ *55*)
notation *tensor-update* (**infixr** $\otimes_u$ *70*)
notation *register-pair* (′(-;-′))

2.1 Elementary facts

declare *id-preregister*[*simp*]
declare *id-update-left*[*simp*]
declare *id-update-right*[*simp*]
declare *register-preregister*[*simp*]
declare *register-comp*[*simp*]
declare *register-of-id*[*simp*]
declare *register-tensor-left*[*simp*]
declare *register-tensor-right*[*simp*]
declare *preregister-mult-right*[*simp*]
declare *preregister-mult-left*[*simp*]
declare *register-id*[*simp*]

2.2 Preregisters

lemma *preregister-tensor-left*[*simp*]: ‹*preregister* (λ*b*::′*b*::*domain update. tensor-update a b*)›
  for *a* :: ‹′*a*::*domain update*›
⟨*proof*⟩

lemma *preregister-tensor-right*[*simp*]: ‹*preregister* (λ*a*::′*a*::*domain update. tensor-update a b*)›
  for *b* :: ‹′*b*::*domain update*›
⟨*proof*⟩

2.3 Registers

lemma *id-update-tensor-register*[*simp*]:
  assumes ‹*register F*›
  shows ‹*register* (λ*a*::′*a*::*domain update. id-update* $\otimes_u$ *F a*)›
  ⟨*proof*⟩

lemma *register-tensor-id-update*[*simp*]:
  assumes ‹*register F*›
  shows ‹*register* (λ*a*::′*a*::*domain update. F a* $\otimes_u$ *id-update*)›
  ⟨*proof*⟩

2.4 Tensor product of registers

definition *register-tensor* (**infixr** $\otimes_r$ *70*) **where**
  register-tensor F G = *register-pair* (λ*a. tensor-update* (*F a*) *id-update*) (λ*b. tensor-update id-update* (*G b*))

lemma *register-tensor-is-register*:
  fixes *F* :: ′*a*::*domain update* ⇒ ′*b*::*domain update* **and** *G* :: ′*c*::*domain update* ⇒ ′*d*::*domain update*
  shows *register F* ⟹ *register G* ⟹ *register* (*F* $\otimes_r$ *G*)
  ⟨*proof*⟩

lemma *register-tensor-apply*[*simp*]:
  fixes *F* :: ′*a*::*domain update* ⇒ ′*b*::*domain update* **and** *G* :: ′*c*::*domain update* ⇒ ′*d*::*domain update*
  assumes ‹*register F*› **and** ‹*register G*›
  shows (*F* $\otimes_r$ *G*) (*a* $\otimes_u$ *b*) = *F a* $\otimes_u$ *G b*
  ⟨*proof*⟩

definition *separating* (-::′*b*::*domain itself*) *A* ⟷

$(\forall\, F\ G :: {}'a{::}domain\ update \Rightarrow {}'b\ update.\ preregister\ F \longrightarrow preregister\ G \longrightarrow (\forall\, x{\in}A.\ F\ x = G\ x) \longrightarrow F = G)$

lemma *separating-UNIV*[*simp*]: ‹*separating TYPE*(-) *UNIV*›
  ⟨*proof*⟩

lemma *separating-mono*: ‹$A \subseteq B \Longrightarrow separating\ TYPE({}'a{::}domain)\ A \Longrightarrow separating\ TYPE({}'a)\ B$›
  ⟨*proof*⟩

lemma *register-eqI*: ‹$separating\ TYPE({}'b{::}domain)\ A \Longrightarrow preregister\ F \Longrightarrow preregister\ G \Longrightarrow (\bigwedge x.\ x{\in}A \Longrightarrow F\ x = G\ x) \Longrightarrow F = (G{::}\text{-} \Rightarrow {}'b\ update)$›
  ⟨*proof*⟩

lemma *separating-tensor*:
  fixes A :: ‹$'a{::}domain\ update\ set$› **and** B :: ‹$'b{::}domain\ update\ set$›
  assumes [*simp*]: ‹$separating\ TYPE({}'c{::}domain)\ A$›
  assumes [*simp*]: ‹$separating\ TYPE({}'c)\ B$›
  shows ‹$separating\ TYPE({}'c)\ \{a \otimes_u b \mid a\ b.\ a{\in}A \wedge b{\in}B\}$›
⟨*proof*⟩

lemma *register-tensor-distrib*:
  assumes [*simp*]: ‹*register F*› ‹*register G*› ‹*register H*› ‹*register L*›
  shows ‹$(F \otimes_r G)\ o\ (H \otimes_r L) = (F\ o\ H) \otimes_r (G\ o\ L)$›
  ⟨*proof*⟩

The following is easier to apply using the *rule*-method than *separating-tensor*

lemma *separating-tensor'*:
  fixes A :: ‹$'a{::}domain\ update\ set$› **and** B :: ‹$'b{::}domain\ update\ set$›
  assumes ‹$separating\ TYPE({}'c{::}domain)\ A$›
  assumes ‹$separating\ TYPE({}'c)\ B$›
  assumes ‹$C = \{a \otimes_u b \mid a\ b.\ a{\in}A \wedge b{\in}B\}$›
  shows ‹$separating\ TYPE({}'c)\ C$›
  ⟨*proof*⟩

lemma *tensor-extensionality3*:
  fixes $F\ G$ :: ‹$({}'a{::}domain \times {}'b{::}domain \times {}'c{::}domain)\ update \Rightarrow {}'d{::}domain\ update$›
  assumes [*simp*]: ‹*register F*› ‹*register G*›
  assumes $\bigwedge f\ g\ h.\ F\ (f \otimes_u g \otimes_u h) = G\ (f \otimes_u g \otimes_u h)$
  shows $F = G$
⟨*proof*⟩

lemma *tensor-extensionality3'*:
  fixes $F\ G$ :: ‹$(({}'a{::}domain \times {}'b{::}domain) \times {}'c{::}domain)\ update \Rightarrow {}'d{::}domain\ update$›
  assumes [*simp*]: ‹*register F*› ‹*register G*›
  assumes $\bigwedge f\ g\ h.\ F\ ((f \otimes_u g) \otimes_u h) = G\ ((f \otimes_u g) \otimes_u h)$
  shows $F = G$
⟨*proof*⟩

lemma *register-tensor-id*[*simp*]: ‹$id \otimes_r id = id$›
  ⟨*proof*⟩

2.5 Pairs and compatibility

definition *compatible* :: ‹$({}'a{::}domain\ update \Rightarrow {}'c{::}domain\ update)$
      $\Rightarrow ({}'b{::}domain\ update \Rightarrow {}'c\ update) \Rightarrow bool$› **where**
  ‹$compatible\ F\ G \longleftrightarrow register\ F \wedge register\ G \wedge (\forall\, a\ b.\ F\ a *_u G\ b = G\ b *_u F\ a)$›

lemma *compatibleI*:
  assumes *register F* **and** *register G*
  assumes ‹$\bigwedge a\ b.\ (F\ a) *_u (G\ b) = (G\ b) *_u (F\ a)$›
  shows *compatible F G*
  ⟨*proof*⟩

lemma *swap-registers*:

assumes *compatible R S*
shows $R\ a *_u S\ b = S\ b *_u R\ a$
⟨*proof*⟩

lemma *compatible-sym*: *compatible x y* $\Longrightarrow$ *compatible y x*
⟨*proof*⟩

lemma *pair-is-register*[*simp*]:
assumes *compatible F G*
shows *register* (*F*; *G*)
⟨*proof*⟩

lemma *register-pair-apply*:
assumes ‹*compatible F G*›
shows ‹$(F;\ G)\ (a \otimes_u b) = (F\ a) *_u (G\ b)$›
⟨*proof*⟩

lemma *register-pair-apply′*:
assumes ‹*compatible F G*›
shows ‹$(F;\ G)\ (a \otimes_u b) = (G\ b) *_u (F\ a)$›
⟨*proof*⟩

lemma *compatible-comp-left*[*simp*]: *compatible F G* $\Longrightarrow$ *register H* $\Longrightarrow$ *compatible* $(F \circ H)$ *G*
⟨*proof*⟩

lemma *compatible-comp-right*[*simp*]: *compatible F G* $\Longrightarrow$ *register H* $\Longrightarrow$ *compatible F* $(G \circ H)$
⟨*proof*⟩

lemma *compatible-comp-inner*[*simp*]:
compatible F G $\Longrightarrow$ *register H* $\Longrightarrow$ *compatible* $(H \circ F)$ $(H \circ G)$
⟨*proof*⟩

lemma *compatible-register1*: ‹*compatible F G* $\Longrightarrow$ *register F*›
⟨*proof*⟩
lemma *compatible-register2*: ‹*compatible F G* $\Longrightarrow$ *register G*›
⟨*proof*⟩

lemma *pair-o-tensor*:
assumes *compatible A B* **and** [*simp*]: ‹*register C*› **and** [*simp*]: ‹*register D*›
shows $(A;\ B)\ o\ (C \otimes_r D) = (A\ o\ C;\ B\ o\ D)$
⟨*proof*⟩

lemma *compatible-tensor-id-update-left*[*simp*]:
fixes *F* :: *′a*::*domain update* ⇒ *′c*::*domain update* **and** *G* :: *′b*::*domain update* ⇒ *′c*::*domain update*
assumes *compatible F G*
shows *compatible* $(\lambda a.\ \mathit{id\text{-}update} \otimes_u F\ a)$ $(\lambda a.\ \mathit{id\text{-}update} \otimes_u G\ a)$
⟨*proof*⟩

lemma *compatible-tensor-id-update-right*[*simp*]:
fixes *F* :: *′a*::*domain update* ⇒ *′c*::*domain update* **and** *G* :: *′b*::*domain update* ⇒ *′c*::*domain update*
assumes *compatible F G*
shows *compatible* $(\lambda a.\ F\ a \otimes_u \mathit{id\text{-}update})$ $(\lambda a.\ G\ a \otimes_u \mathit{id\text{-}update})$
⟨*proof*⟩

lemma *compatible-tensor-id-update-rl*[*simp*]:
assumes *register F* **and** *register G*
shows *compatible* $(\lambda a.\ F\ a \otimes_u \mathit{id\text{-}update})$ $(\lambda a.\ \mathit{id\text{-}update} \otimes_u G\ a)$
⟨*proof*⟩

lemma *compatible-tensor-id-update-lr*[*simp*]:
assumes *register F* **and** *register G*

shows *compatible* ($\lambda a.$ *id-update* $\otimes_u$ *F a*) ($\lambda a.$ *G a* $\otimes_u$ *id-update*)
⟨*proof*⟩

lemma *register-comp-pair*:
assumes [*simp*]: ‹*register F*› **and** [*simp*]: ‹*compatible G H*›
shows (*F o G*; *F o H*) = *F o* (*G*; *H*)
⟨*proof*⟩

lemma *swap-registers-left*:
assumes *compatible R S*
shows *R a* $*_u$ *S b* $*_u$ *c* = *S b* $*_u$ *R a* $*_u$ *c*
⟨*proof*⟩

lemma *swap-registers-right*:
assumes *compatible R S*
shows *c* $*_u$ *R a* $*_u$ *S b* = *c* $*_u$ *S b* $*_u$ *R a*
⟨*proof*⟩

lemmas *compatible-ac-rules* = *swap-registers comp-update-assoc*[*symmetric*] *swap-registers-right*

2.6 Fst and Snd

definition *Fst* **where** ‹*Fst a* = *a* $\otimes_u$ *id-update*›
definition *Snd* **where** ‹*Snd a* = *id-update* $\otimes_u$ *a*›

lemma *register-Fst*[*simp*]: ‹*register Fst*›
⟨*proof*⟩

lemma *register-Snd*[*simp*]: ‹*register Snd*›
⟨*proof*⟩

lemma *compatible-Fst-Snd*[*simp*]: ‹*compatible Fst Snd*›
⟨*proof*⟩

lemmas *compatible-Snd-Fst*[*simp*] = *compatible-Fst-Snd*[*THEN compatible-sym*]

definition ‹*swap* = (*Snd*; *Fst*)›

lemma *swap-apply*[*simp*]: *swap* (*a* $\otimes_u$ *b*) = (*b* $\otimes_u$ *a*)
⟨*proof*⟩

lemma *swap-o-Fst*: *swap o Fst* = *Snd*
⟨*proof*⟩
lemma *swap-o-Snd*: *swap o Snd* = *Fst*
⟨*proof*⟩

lemma *register-swap*[*simp*]: ‹*register swap*›
⟨*proof*⟩

lemma *pair-Fst-Snd*: ‹(*Fst*; *Snd*) = *id*›
⟨*proof*⟩

lemma *swap-o-swap*[*simp*]: ‹*swap o swap* = *id*›
⟨*proof*⟩

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

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

lemma *register-pair-Fst*:
assumes ‹*compatible F G*›

shows ‹$(F;G)$ *o* *Fst* = F›
⟨*proof*⟩

lemma *register-pair-Snd*:
assumes ‹*compatible* F G›
shows ‹$(F;G)$ *o* *Snd* = G›
⟨*proof*⟩

lemma *register-Fst-register-Snd*[*simp*]:
assumes ‹*register* F›
shows ‹(F *o* *Fst*; F *o* *Snd*) = F›
⟨*proof*⟩

lemma *register-Snd-register-Fst*[*simp*]:
assumes ‹*register* F›
shows ‹(F *o* *Snd*; F *o* *Fst*) = F *o* *swap*›
⟨*proof*⟩

lemma *compatible3*[*simp*]:
assumes [*simp*]: *compatible* F G **and** *compatible* G H **and** *compatible* F H
shows *compatible* (F; G) H
⟨*proof*⟩

lemma *compatible3'*[*simp*]:
assumes *compatible* F G **and** *compatible* G H **and** *compatible* F H
shows *compatible* F (G; H)
⟨*proof*⟩

lemma *pair-o-swap*[*simp*]:
assumes [*simp*]: *compatible* A B
shows (A; B) *o* *swap* = (B; A)
⟨*proof*⟩

2.7 Compatibility of register tensor products

lemma *compatible-register-tensor*:
fixes F :: ‹'*a*::*domain update* ⇒ '*e*::*domain update*› **and** G :: ‹'*b*::*domain update* ⇒ '*f*::*domain update*›
and F' :: ‹'*c*::*domain update* ⇒ '*e update*› **and** G' :: ‹'*d*::*domain update* ⇒ '*f update*›
assumes [*simp*]: ‹*compatible* F F'›
assumes [*simp*]: ‹*compatible* G G'›
shows ‹*compatible* $(F \otimes_r G)$ $(F' \otimes_r G')$›
⟨*proof*⟩

2.8 Associativity of the tensor product

definition *assoc* :: ‹(('*a*::*domain*×'*b*::*domain*)×'*c*::*domain*) *update* ⇒ ('*a*×('*b*×'*c*)) *update*› **where**
‹*assoc* = ((*Fst*; *Snd* *o* *Fst*); *Snd* *o* *Snd*)›

lemma *assoc-is-hom*[*simp*]: ‹*preregister assoc*›
⟨*proof*⟩

lemma *assoc-apply*[*simp*]: ‹*assoc* $((a \otimes_u b) \otimes_u c) = (a \otimes_u (b \otimes_u c))$›
⟨*proof*⟩

definition *assoc'* :: ‹('*a*×('*b*×'*c*)) *update* ⇒ (('*a*::*domain*×'*b*::*domain*)×'*c*::*domain*) *update*› **where**
‹*assoc'* = (*Fst* *o* *Fst*; (*Fst* *o* *Snd*; *Snd*))›

lemma *assoc'-is-hom*[*simp*]: ‹*preregister assoc'*›
⟨*proof*⟩

lemma *assoc'-apply*[*simp*]: ‹*assoc'* $(a \otimes_u (b \otimes_u c)) = ((a \otimes_u b) \otimes_u c)$›
⟨*proof*⟩

lemma *register-assoc*[*simp*]: ‹*register assoc*›
  ⟨*proof*⟩

lemma *register-assoc'*[*simp*]: ‹*register assoc'*›
  ⟨*proof*⟩

lemma *pair-o-assoc*[*simp*]:
  assumes [*simp*]: ‹*compatible F G*› ‹*compatible G H*› ‹*compatible F H*›
  shows ‹$(F;\ (G;\ H)) \circ$ *assoc* $= ((F;\ G);\ H)$›
⟨*proof*⟩

lemma *pair-o-assoc'*[*simp*]:
  assumes [*simp*]: ‹*compatible F G*› ‹*compatible G H*› ‹*compatible F H*›
  shows ‹$((F;\ G);\ H) \circ$ *assoc'* $= (F;\ (G;\ H))$›
⟨*proof*⟩

lemma *assoc'-o-assoc*[*simp*]: ‹*assoc' o assoc* = *id*›
  ⟨*proof*⟩

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

lemma *assoc-o-assoc'*[*simp*]: ‹*assoc o assoc'* = *id*›
  ⟨*proof*⟩

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

lemma *inv-assoc*[*simp*]: ‹*inv assoc* = *assoc'*›
  ⟨*proof*⟩

lemma *inv-assoc'*[*simp*]: ‹*inv assoc'* = *assoc*›
  ⟨*proof*⟩

lemma *bij-assoc*[*simp*]: ‹*bij assoc*›
  ⟨*proof*⟩

lemma *bij-assoc'*[*simp*]: ‹*bij assoc'*›
  ⟨*proof*⟩

2.9 Iso-registers

definition ‹*iso-register F* $\longleftrightarrow$ *register F* $\wedge$ ($\exists$ *G*. *register G* $\wedge$ *F o G* = *id* $\wedge$ *G o F* = *id*)›
  for *F* :: ‹-::*domain update* ⇒ -::*domain update*›

lemma *iso-registerI*:
  assumes ‹*register F*› ‹*register G*› ‹*F o G* = *id*› ‹*G o F* = *id*›
  shows ‹*iso-register F*›
  ⟨*proof*⟩

lemma *iso-register-inv*: ‹*iso-register F* $\Longrightarrow$ *iso-register* (*inv F*)›
  ⟨*proof*⟩

lemma *iso-register-inv-comp1*: ‹*iso-register F* $\Longrightarrow$ *inv F o F* = *id*›
  ⟨*proof*⟩

lemma *iso-register-inv-comp2*: ‹*iso-register F* $\Longrightarrow$ *F o inv F* = *id*›
  ⟨*proof*⟩

lemma *iso-register-id*[*simp*]: ‹*iso-register id*›
  ⟨*proof*⟩

lemma *iso-register-is-register*: ‹*iso-register F* $\Longrightarrow$ *register F*›
  ⟨*proof*⟩

lemma *iso-register-comp*[*simp*]:
  assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›
  shows ‹*iso-register* (*F o G*)›
⟨*proof*⟩

lemma *iso-register-tensor-is-iso-register*[*simp*]:
  assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›
  shows ‹*iso-register* ($F \otimes_r G$)›
⟨*proof*⟩

lemma *iso-register-bij*: ‹*iso-register F* $\Longrightarrow$ *bij F*›
  ⟨*proof*⟩

lemma *inv-register-tensor*[*simp*]:
  assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›
  shows ‹*inv* ($F \otimes_r G$) = *inv* $F \otimes_r$ *inv* G›
  ⟨*proof*⟩

lemma *iso-register-swap*[*simp*]: ‹*iso-register swap*›
  ⟨*proof*⟩

lemma *iso-register-assoc*[*simp*]: ‹*iso-register assoc*›
  ⟨*proof*⟩

lemma *iso-register-assoc′*[*simp*]: ‹*iso-register assoc′*›
  ⟨*proof*⟩

definition ‹*equivalent-registers F G* $\longleftrightarrow$ (*register F* $\wedge$ ($\exists$ *I*. *iso-register I* $\wedge$ *F o I* = *G*))›
  for *F G* :: ‹-::*domain update* $\Rightarrow$ -::*domain update*›

lemma *iso-register-equivalent-id*[*simp*]: ‹*equivalent-registers id F* $\longleftrightarrow$ *iso-register F*›
  ⟨*proof*⟩

lemma *equivalent-registersI*:
  assumes ‹*register F*›
  assumes ‹*iso-register I*›
  assumes ‹*F o I* = *G*›
  shows ‹*equivalent-registers F G*›
  ⟨*proof*⟩

lemma *equivalent-registers-refl*: ‹*equivalent-registers F F*› **if** ‹*register F*›
  ⟨*proof*⟩

lemma *equivalent-registers-register-left*: ‹*equivalent-registers F G* $\Longrightarrow$ *register F*›
  ⟨*proof*⟩

lemma *equivalent-registers-register-right*: ‹*register G*› **if** ‹*equivalent-registers F G*›
  ⟨*proof*⟩

lemma *equivalent-registers-sym*:
  assumes ‹*equivalent-registers F G*›
  shows ‹*equivalent-registers G F*›
  ⟨*proof*⟩

lemma *equivalent-registers-trans*[*trans*]:
  assumes ‹*equivalent-registers F G*› **and** ‹*equivalent-registers G H*›
  shows ‹*equivalent-registers F H*›
⟨*proof*⟩

lemma *equivalent-registers-assoc*[*simp*]:
  assumes [*simp*]: ‹*compatible F G*› ‹*compatible F H*› ‹*compatible G H*›
  shows ‹*equivalent-registers* (*F*;(*G*;*H*)) ((*F*;*G*);*H*)›
  ⟨*proof*⟩

lemma *equivalent-registers-pair-right*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes *eq*: ‹*equivalent-registers G H*›
  shows ‹*equivalent-registers* (*F*;*G*) (*F*;*H*)›
⟨*proof*⟩

lemma *equivalent-registers-pair-left*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes *eq*: ‹*equivalent-registers F H*›
  shows ‹*equivalent-registers* (*F*;*G*) (*H*;*G*)›
⟨*proof*⟩

lemma *equivalent-registers-comp*:
  assumes ‹*register H*›
  assumes ‹*equivalent-registers F G*›
  shows ‹*equivalent-registers* (*H o F*) (*H o G*)›
  ⟨*proof*⟩

lemma *equivalent-registers-compatible1*:
  assumes ‹*compatible F G*›
  assumes ‹*equivalent-registers* F F'›
  shows ‹*compatible* F' G›
  ⟨*proof*⟩

lemma *equivalent-registers-compatible2*:
  assumes ‹*compatible F G*›
  assumes ‹*equivalent-registers* G G'›
  shows ‹*compatible* F G'›
  ⟨*proof*⟩

2.10 Compatibility simplification

The simproc *compatibility-warn* produces helpful warnings for subgoals of the form *compatible x y* that are probably unsolvable due to missing declarations of variable compatibility facts. Same for subgoals of the form *register x*.

⟨*ML*⟩

named-theorems *register-attribute-rule-immediate*
named-theorems *register-attribute-rule*

lemmas [*register-attribute-rule*] = *conjunct1 conjunct2 iso-register-is-register iso-register-is-register*[*OF iso-register-inv*]
lemmas [*register-attribute-rule-immediate*] = *compatible-sym compatible-register1 compatible-register2*
  asm-rl[*of* ‹*compatible - -*›] *asm-rl*[*of* ‹*iso-register -*›] *asm-rl*[*of* ‹*register -*›] *iso-register-inv*

The following declares an attribute [*register*]. When the attribute is applied to a fact of the form *register F*, *iso-register F*, *compatible F G* or a conjunction of these, then those facts are added to the simplifier together with some derived theorems (e.g., *compatible F G* also adds *register F*).

In theory *Laws-Complement*, support for *is-unit-register F* and *complements F G* is added to this attribute.

⟨*ML*⟩

2.11 Notation

no-notation *comp-update* (**infixl** $*_u$ *55*)
no-notation *tensor-update* (**infixr** $\otimes_u$ *70*)

bundle *register-syntax* **begin**
notation *register-tensor* (**infixr** $\otimes_r$ *70*)
notation *register-pair* ('(-;-'))
end

end

3 Axioms of complements

theory *Axioms-Complement*
  imports *Laws With-Type.With-Type*
begin

typedecl ('*a*, '*b*) *complement-domain*
instance *complement-domain* :: (*domain*, *domain*) *domain*⟨*proof*⟩
typedecl ('*a*, '*b*) *complement-domain-simple*
instance *complement-domain-simple* :: (*domain*, *domain*) *domain*⟨*proof*⟩

⟨*ML*⟩

class *domain-with-simple-complement* = *domain*

— We need that there is at least one object in our category. We call is *some-domain*.
typedecl *some-domain*
instance *some-domain* :: *domain-with-simple-complement* ⟨*proof*⟩

axiomatization where
  complement-exists-simple: ‹*register F* ⟹ ∃ *G* :: ('*a*, '*b*) *complement-domain-simple update* ⇒ '*b update*. *compatible F G* ∧ *iso-register* (*F*;*G*)›
    for *F* :: ‹'*a*::*domain update* ⇒ '*b*::*domain-with-simple-complement update*›

axiomatization *cdc* :: ‹('*a*::*domain update* ⇒ '*b*::*domain update*) ⇒ ('*a*,'*b*) *complement-domain set*› **where**
  complement-exists: ‹*register F* ⟹ *let* '*c*::*domain* = *cdc F in*
                     ∃ *G* :: '*c update* ⇒ '*b update*. *compatible F G* ∧ *iso-register* (*F*;*G*)›
    for *F* :: ‹'*a*::*domain update* ⇒ '*b*::*domain update*›

axiomatization where *complement-unique*: ‹*compatible F G* ⟹ *iso-register* (*F*;*G*) ⟹ *compatible F H* ⟹ *iso-register* (*F*;*H*)
          ⟹ *equivalent-registers G H*›
      for *F* :: ‹'*a*::*domain update* ⇒ '*b*::*domain update*› **and** *G* :: ‹'*g*::*domain update* ⇒ '*b update*› **and** *H* :: ‹'*h*::*domain update* ⇒ '*b update*›

end

4 Generic laws about complements

theory *Laws-Complement*
  imports *Laws Axioms-Complement*
begin

unbundle *register-syntax*
notation *comp-update* (**infixl** $*_u$ *55*)
notation *tensor-update* (**infixr** $\otimes_u$ *70*)

definition ‹*complements F G* ⟷ *compatible F G* ∧ *iso-register* (*F*;*G*)›

lemma *complementsI*: ‹*compatible F G* ⟹ *iso-register* (*F*;*G*) ⟹ *complements F G*›
  ⟨*proof*⟩

lemma *complement-exists*:
  fixes *F* :: ‹'*a*::*domain update* ⇒ '*b*::*domain update*›

assumes ‹*register F*›
shows ‹*let 'c::domain = cdc F in*
    ∃ G :: 'c update ⇒ 'b update. complements F G›
⟨*proof*⟩

lemma *complements-sym*: ‹*complements G F*› **if** ‹*complements F G*›
⟨*proof*⟩

definition *complement* :: ‹(*'a::domain update ⇒ 'b::domain-with-simple-complement update*) ⇒ ((*'a,'b*) *complement-domain-simple update ⇒ 'b update*)› **where**
‹*complement F* = (*SOME G* :: (*'a, 'b*) *complement-domain-simple update ⇒ 'b update. compatible F G ∧ iso-register* (*F*;*G*))›

lemma *register-complement*[*simp*]: ‹*register* (*complement F*)› **if** ‹*register F*›
⟨*proof*⟩

lemma *complement-is-complement*[*simp*]:
assumes ‹*register F*›
shows ‹*complements F* (*complement F*)›
⟨*proof*⟩

lemma *complement-unique*:
assumes ‹*complements F G*›
assumes ‹*complements F G'*›
shows ‹*equivalent-registers G G'*›
⟨*proof*⟩

lemma *complement-unique'*:
assumes ‹*complements F G*›
shows ‹*equivalent-registers G* (*complement F*)›
⟨*proof*⟩

lemma *compatible-complement*[*simp*]: ‹*register F ⟹ compatible F* (*complement F*)›
⟨*proof*⟩

lemma *complements-register-tensor*:
assumes [*simp*]: ‹*register F*› ‹*register G*›
shows ‹*complements* ($F \otimes_r$ *G*) (*complement F* $\otimes_r$ *complement G*)›
⟨*proof*⟩

definition *is-unit-register* **where**
‹*is-unit-register U ⟷ complements U id*›

lemma *register-unit-register*[*simp*]: ‹*is-unit-register U ⟹ register U*›
⟨*proof*⟩

lemma *unit-register-compatible*[*simp*]: ‹*compatible U X*› **if** ‹*is-unit-register U*› ‹*register X*›
⟨*proof*⟩

lemma *unit-register-compatible'*[*simp*]: ‹*compatible X U*› **if** ‹*is-unit-register U*› ‹*register X*›
⟨*proof*⟩

lemma *compatible-complement-left*[*simp*]: ‹*register X ⟹ compatible* (*complement X*) *X*›
⟨*proof*⟩

lemma *compatible-complement-right*[*simp*]: ‹*register X ⟹ compatible X* (*complement X*)›
⟨*proof*⟩

lemma *unit-register-pair*[*simp*]: ‹*equivalent-registers X* (*U*; *X*)› **if** [*simp*]: ‹*is-unit-register U*› ‹*register X*›
⟨*proof*⟩

lemma *unit-register-compose-left*:

  assumes [*simp*]: ‹*is-unit-register U*›
  assumes [*simp*]: ‹*register A*›
  shows ‹*is-unit-register* (*A o U*)›
⟨*proof*⟩

lemma *unit-register-compose-right*:
  assumes [*simp*]: ‹*is-unit-register U*›
  assumes [*simp*]: ‹*iso-register A*›
  shows ‹*is-unit-register* (*U o A*)›
⟨*proof*⟩

lemma *unit-register-unique*:
  assumes ‹*is-unit-register F*›
  assumes ‹*is-unit-register G*›
  shows ‹*equivalent-registers F G*›
⟨*proof*⟩

lemma *unit-register-domains-isomorphic*:
  fixes *F* :: ‹′*a*::*domain update* ⇒ ′*c*::*domain update*›
  fixes *G* :: ‹′*b*::*domain update* ⇒ ′*d*::*domain update*›
  assumes ‹*is-unit-register F*›
  assumes ‹*is-unit-register G*›
  shows ‹∃ *I* :: ′*a update* ⇒ ′*b update*. *iso-register I*›
⟨*proof*⟩

lemma *id-complement-is-unit-register*[*simp*]: ‹*is-unit-register* (*complement id*)›
  ⟨*proof*⟩

type-synonym *unit-register-domain* = ‹(*some-domain*, *some-domain*) *complement-domain-simple*›
definition *unit-register* :: ‹*unit-register-domain update* ⇒ ′*a*::*domain update*› **where** ‹*unit-register* = (*SOME U*. *is-unit-register U*)›

lemma *unit-register-is-unit-register*[*simp*]: ‹*is-unit-register* (*unit-register* :: *unit-register-domain update* ⇒ ′*a*::*domain update*)›
⟨*proof*⟩

lemma *unit-register-domain-tensor-unit*:
  fixes *U* :: ‹′*a*::*domain update* ⇒ -›
  assumes ‹*is-unit-register U*›
  shows ‹∃ *I* :: ′*b*::*domain update* ⇒ (′*a*∗′*b*) *update*. *iso-register I*›

⟨*proof*⟩

lemma *compatible-complement-pair1*:
  assumes ‹*compatible F G*›
  shows ‹*compatible F* (*complement* (*F*;*G*))›
  ⟨*proof*⟩

lemma *compatible-complement-pair2*:
  assumes [*simp*]: ‹*compatible F G*›
  shows ‹*compatible G* (*complement* (*F*;*G*))›
⟨*proof*⟩

lemma *equivalent-complements*:
  assumes ‹*complements F G*›
  assumes ‹*equivalent-registers G G*′›
  shows ‹*complements F G*′›
  ⟨*proof*⟩

lemma *complements-complement-pair*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes *FG*′: ‹*complements* (*F*;*G*) *FG*′›

shows ‹*complements F* (*G*; *FG′*)›
⟨*proof*⟩

lemma *equivalent-registers-complement*:
assumes ‹*equivalent-registers F G*›
assumes ‹*complements F F′*›
assumes ‹*complements G G′*›
shows ‹*equivalent-registers F′ G′*›
⟨*proof*⟩

lemma *equivalent-registers-complement′*:
assumes ‹*equivalent-registers F G*›
shows ‹*equivalent-registers* (*complement F*) (*complement G*)›
⟨*proof*⟩

lemma *complements-complement-pair′*:
assumes [*simp*]: ‹*compatible F G*›
assumes *FG′*: ‹*complements* (*F*;*G*) *FG′*›
shows ‹*complements G* (*F*; *FG′*)›
⟨*proof*⟩

lemma *complements-chain*:
assumes [*simp*]: ‹*register F*› ‹*register G*›
shows ‹*complements* (*F o G*) (*complement F*; *F o complement G*)›
⟨*proof*⟩

lemma *complements-Fst-Snd*[*simp*]: ‹*complements Fst Snd*›
⟨*proof*⟩

lemma *complements-Snd-Fst*[*simp*]: ‹*complements Snd Fst*›
⟨*proof*⟩

lemma *compatible-unit-register*[*simp*]: ‹*register F* ⟹ *compatible F unit-register*›
⟨*proof*⟩

lemma *complements-id-unit-register*[*simp*]: ‹*complements id unit-register*›
⟨*proof*⟩

lemma *complements-iso-unit-register*: ‹*iso-register I* ⟹ *is-unit-register U* ⟹ *complements I U*›
⟨*proof*⟩

lemma *iso-register-complement-is-unit-register*[*simp*]:
assumes ‹*iso-register F*›
shows ‹*is-unit-register* (*complement F*)›
⟨*proof*⟩

Adding support for *is-unit-register F* and *complements F G* to the [*register*] attribute

lemmas [*register-attribute-rule*] = *is-unit-register-def*[*THEN iffD1*] *complements-def*[*THEN iffD1*]
lemmas [*register-attribute-rule-immediate*] = *asm-rl*[*of* ‹*is-unit-register* -›]

no-notation *comp-update* (**infixl** $*_u$ *55*)
no-notation *tensor-update* (**infixr** $\otimes_u$ *70*)
unbundle *no register-syntax*

end

5 Classical instantiation of registers

theory *Axioms-Classical*
imports *Main*
begin

type-synonym *'a update* = ‹*'a* ⇀ *'a*›

lemma *id-update-left*: *Some* $\circ_m$ *a* = *a*
  ⟨*proof*⟩
lemma *id-update-right*: *a* $\circ_m$ *Some* = *a*
  ⟨*proof*⟩

lemma *comp-update-assoc*: (*a* $\circ_m$ *b*) $\circ_m$ *c* = *a* $\circ_m$ (*b* $\circ_m$ *c*)
  ⟨*proof*⟩

type-synonym (*'a,'b*) *preregister* = ‹*'a update* ⇒ *'b update*›
definition *preregister* :: ‹(*'a,'b*) *preregister* ⇒ *bool*› **where**
  ‹*preregister F* ⟷ (∃ *g s*. ∀ *a m*. *F a m* = (*case a* (*g m*) *of None* ⇒ *None* | *Some x* ⇒ *s x m*))›

lemma *id-preregister*: ‹*preregister id*›
  ⟨*proof*⟩

lemma *preregister-mult-right*: ‹*preregister* (λ*a*. *a* $\circ_m$ *z*)›
  ⟨*proof*⟩

lemma *preregister-mult-left*: ‹*preregister* (λ*a*. *z* $\circ_m$ *a*)›
  ⟨*proof*⟩

lemma *comp-preregister*: *preregister* (*G* ∘ *F*) **if** *preregister F* **and** ‹*preregister G*›
⟨*proof*⟩

definition *tensor-update* :: ‹*'a update* ⇒ *'b update* ⇒ (*'a*×*'b*) *update*› **where**
  ‹*tensor-update a b m* = (*case a* (*fst m*) *of None* ⇒ *None* | *Some x* ⇒ (*case b* (*snd m*) *of None* ⇒ *None* | *Some y* ⇒ *Some* (*x*,*y*)))›

lemma *tensor-update-mult*: ‹*tensor-update a c* $\circ_m$ *tensor-update b d* = *tensor-update* (*a* $\circ_m$ *b*) (*c* $\circ_m$ *d*)›
  ⟨*proof*⟩

definition *update1* :: ‹*'a* ⇒ *'a* ⇒ *'a update*› **where**
  ‹*update1 x y m* = (*if m*=*x then Some y else None*)›

lemma *update1-extensionality*:
  assumes ‹*preregister F*›
  assumes ‹*preregister G*›
  assumes *FGeq*: ‹⋀*x y*. *F* (*update1 x y*) = *G* (*update1 x y*)›
  shows *F* = *G*
⟨*proof*⟩

lemma *tensor-extensionality*:
  assumes ‹*preregister F*›
  assumes ‹*preregister G*›
  assumes *FGeq*: ‹⋀*a b*. *F* (*tensor-update a b*) = *G* (*tensor-update a b*)›
  shows *F* = *G*
⟨*proof*⟩

definition *valid-getter-setter g s* ⟷
  (∀ *b*. *b* = *s* (*g b*) *b*) ∧ (∀ *a b*. *g* (*s a b*) = *a*) ∧ (∀ *a a' b*. *s a* (*s a' b*) = *s a b*)

definition ‹*register-from-getter-setter g s a m* = (*case a* (*g m*) *of None* ⇒ *None* | *Some x* ⇒ *Some* (*s x m*))›

definition ‹*register-apply F a* = *the o F* (*Some o a*)›
definition ‹*setter F a m* = *register-apply F* (λ-. *a*) *m*› **for** *F* :: ‹*'a update* ⇒ *'b update*›
definition ‹*getter F m* = (*THE x*. *setter F x m* = *m*)› **for** *F* :: ‹*'a update* ⇒ *'b update*›

lemma
  assumes ‹*valid-getter-setter g s*›
  shows *getter-of-register-from-getter-setter*[*simp*]: ‹*getter* (*register-from-getter-setter g s*) = *g*›
    and *setter-of-register-from-getter-setter*[*simp*]: ‹*setter* (*register-from-getter-setter g s*) = *s*›

⟨*proof*⟩

definition *register* :: ‹(*'a,'b*) *preregister* ⇒ *bool*› **where**
  ‹*register F* ⟷ (∃ *g s*. *F* = *register-from-getter-setter g s* ∧ *valid-getter-setter g s*)›

lemma *register-of-id*: ‹*register F* ⟹ *F Some* = *Some*›
  ⟨*proof*⟩

lemma *register-id*: ‹*register id*›
  ⟨*proof*⟩

lemma *register-tensor-left*: ‹*register* (λ*a*. *tensor-update a Some*)›
  ⟨*proof*⟩

lemma *register-tensor-right*: ‹*register* (λ*a*. *tensor-update Some a*)›
  ⟨*proof*⟩

lemma *register-preregister*: *preregister F* **if** ‹*register F*›
⟨*proof*⟩

lemma *register-comp*: *register* (*G* ∘ *F*) **if** ‹*register F*› **and** ‹*register G*›
  for *F* :: (*'a,'b*) *preregister* **and** *G* :: (*'b,'c*) *preregister*
⟨*proof*⟩

lemma *register-mult*: *register F* ⟹ *F a* $\circ_m$ *F b* = *F* (*a* $\circ_m$ *b*)
  ⟨*proof*⟩

definition *register-pair* ::
  ‹(*'a update* ⇒ *'c update*) ⇒ (*'b update* ⇒ *'c update*) ⇒ ((*'a*×*'b*) *update* ⇒ *'c update*)› **where**
  ‹*register-pair F G* =
    register-from-getter-setter (λ*m*. (*getter F m*, *getter G m*)) (λ(*a*,*b*) *m*. *setter F a* (*setter G b m*))›

lemma *compatible-setter*:
  assumes [*simp*]: ‹*register F*› ‹*register G*›
  assumes *compat*: ‹⋀*a b*. *F a* $\circ_m$ *G b* = *G b* $\circ_m$ *F a*›
  shows ‹*setter F x o setter G y* = *setter G y o setter F x*›
⟨*proof*⟩

lemma *register-pair-apply*:
  assumes [*simp*]: ‹*register F*› ‹*register G*›
  assumes ‹⋀*a b*. *F a* $\circ_m$ *G b* = *G b* $\circ_m$ *F a*›
  shows ‹(*register-pair F G*) (*tensor-update a b*) = *F a* $\circ_m$ *G b*›
⟨*proof*⟩

lemma *register-pair-is-register*:
  fixes *F* :: ‹*'a update* ⇒ *'c update*› **and** *G*
  assumes [*simp*]: ‹*register F*› **and** [*simp*]: ‹*register G*›
  assumes *compat*: ‹⋀*a b*. *F a* $\circ_m$ *G b* = *G b* $\circ_m$ *F a*›
  shows ‹*register* (*register-pair F G*)›
⟨*proof*⟩

end

6 Generic laws about registers, instantiated classically

theory *Laws-Classical*
  imports *Axioms-Classical*
begin

This notation is only used inside this file

notation *map-comp* (**infixl** $*_u$ *55*)
notation *tensor-update* (**infixr** $\otimes_u$ *70*)
notation *register-pair* (*'*(-;-*'*))

6.1 Elementary facts

declare *id-preregister*[*simp*]
declare *id-update-left*[*simp*]
declare *id-update-right*[*simp*]
declare *register-preregister*[*simp*]
declare *register-comp*[*simp*]
declare *register-of-id*[*simp*]
declare *register-tensor-left*[*simp*]
declare *register-tensor-right*[*simp*]
declare *preregister-mult-right*[*simp*]
declare *preregister-mult-left*[*simp*]
declare *register-id*[*simp*]

6.2 Preregisters

lemma *preregister-tensor-left*[*simp*]: ‹*preregister* (λ*b*::′*b*::*type update. tensor-update a b*)›
  for *a* :: ‹′*a*::*type update*›
⟨*proof*⟩

lemma *preregister-tensor-right*[*simp*]: ‹*preregister* (λ*a*::′*a*::*type update. tensor-update a b*)›
  for *b* :: ‹′*b*::*type update*›
⟨*proof*⟩

6.3 Registers

lemma *id-update-tensor-register*[*simp*]:
  assumes ‹*register F*›
  shows ‹*register* (λ*a*::′*a*::*type update. Some* $\otimes_u$ *F a*)›
  ⟨*proof*⟩

lemma *register-tensor-id-update*[*simp*]:
  assumes ‹*register F*›
  shows ‹*register* (λ*a*::′*a*::*type update. F a* $\otimes_u$ *Some*)›
  ⟨*proof*⟩

6.4 Tensor product of registers

definition *register-tensor* (**infixr** $\otimes_r$ *70*) **where**
  register-tensor F G = *register-pair* (λ*a. tensor-update* (*F a*) *Some*) (λ*b. tensor-update Some* (*G b*))

lemma *register-tensor-is-register*:
  fixes *F* :: ′*a*::*type update* ⇒ ′*b*::*type update* **and** *G* :: ′*c*::*type update* ⇒ ′*d*::*type update*
  shows *register F* ⟹ *register G* ⟹ *register* (*F* $\otimes_r$ *G*)
  ⟨*proof*⟩

lemma *register-tensor-apply*[*simp*]:
  fixes *F* :: ′*a*::*type update* ⇒ ′*b*::*type update* **and** *G* :: ′*c*::*type update* ⇒ ′*d*::*type update*
  assumes ‹*register F*› **and** ‹*register G*›
  shows (*F* $\otimes_r$ *G*) (*a* $\otimes_u$ *b*) = *F a* $\otimes_u$ *G b*
  ⟨*proof*⟩

definition *separating* (-::′*b*::*type itself*) *A* ⟷
  (∀ *F G* :: ′*a*::*type update* ⇒ ′*b update. preregister F* ⟶ *preregister G* ⟶ (∀ *x*∈*A. F x* = *G x*) ⟶ *F* = *G*)

lemma *separating-UNIV*[*simp*]: ‹*separating TYPE*(-) *UNIV*›
  ⟨*proof*⟩

lemma *separating-mono*: ‹*A* ⊆ *B* ⟹ *separating TYPE*(′*a*::*type*) *A* ⟹ *separating TYPE*(′*a*) *B*›
  ⟨*proof*⟩

lemma *register-eqI*: ‹*separating TYPE*(′*b*::*type*) *A* ⟹ *preregister F* ⟹ *preregister G* ⟹ (⋀*x. x*∈*A* ⟹ *F x* = *G x*) ⟹ *F* = (*G*::- ⇒ ′*b update*)›
  ⟨*proof*⟩

lemma *separating-tensor*:
  fixes A :: ‹$'a$::*type update set*› **and** B :: ‹$'b$::*type update set*›
  assumes [*simp*]: ‹*separating TYPE*($'c$::*type*) A›
  assumes [*simp*]: ‹*separating TYPE*($'c$) B›
  shows ‹*separating TYPE*($'c$) $\{a \otimes_u b \mid a\ b.\ a{\in}A \wedge b{\in}B\}$›
⟨*proof*⟩

lemma *register-tensor-distrib*:
  assumes [*simp*]: ‹*register* F› ‹*register* G› ‹*register* H› ‹*register* L›
  shows ‹$(F \otimes_r G)\ o\ (H \otimes_r L) = (F\ o\ H) \otimes_r (G\ o\ L)$›
  ⟨*proof*⟩

The following is easier to apply using the *rule*-method than *separating-tensor*

lemma *separating-tensor'*:
  fixes A :: ‹$'a$::*type update set*› **and** B :: ‹$'b$::*type update set*›
  assumes ‹*separating TYPE*($'c$::*type*) A›
  assumes ‹*separating TYPE*($'c$) B›
  assumes ‹$C = \{a \otimes_u b \mid a\ b.\ a{\in}A \wedge b{\in}B\}$›
  shows ‹*separating TYPE*($'c$) C›
  ⟨*proof*⟩

lemma *tensor-extensionality3*:
  fixes $F\ G$ :: ‹($'a$::*type*×$'b$::*type*×$'c$::*type*) *update* ⇒ $'d$::*type update*›
  assumes [*simp*]: ‹*register* F› ‹*register* G›
  assumes $\bigwedge f\ g\ h.\ F\ (f \otimes_u g \otimes_u h) = G\ (f \otimes_u g \otimes_u h)$
  shows $F = G$
⟨*proof*⟩

lemma *tensor-extensionality3'*:
  fixes $F\ G$ :: ‹(($'a$::*type*×$'b$::*type*)×$'c$::*type*) *update* ⇒ $'d$::*type update*›
  assumes [*simp*]: ‹*register* F› ‹*register* G›
  assumes $\bigwedge f\ g\ h.\ F\ ((f \otimes_u g) \otimes_u h) = G\ ((f \otimes_u g) \otimes_u h)$
  shows $F = G$
⟨*proof*⟩

lemma *register-tensor-id*[*simp*]: ‹$id \otimes_r id = id$›
  ⟨*proof*⟩

6.5 Pairs and compatibility

definition *compatible* :: ‹($'a$::*type update* ⇒ $'c$::*type update*)
                ⇒ ($'b$::*type update* ⇒ $'c$ *update*) ⇒ *bool*› **where**
  ‹*compatible* $F\ G \longleftrightarrow$ *register* $F \wedge$ *register* $G \wedge (\forall a\ b.\ F\ a *_u G\ b = G\ b *_u F\ a)$›

lemma *compatibleI*:
  assumes *register* F **and** *register* G
  assumes ‹$\bigwedge a\ b.\ (F\ a) *_u (G\ b) = (G\ b) *_u (F\ a)$›
  shows *compatible* $F\ G$
  ⟨*proof*⟩

lemma *swap-registers*:
  assumes *compatible* $R\ S$
  shows $R\ a *_u S\ b = S\ b *_u R\ a$
  ⟨*proof*⟩

lemma *compatible-sym*: *compatible* $x\ y \Longrightarrow$ *compatible* $y\ x$
  ⟨*proof*⟩

lemma *pair-is-register*[*simp*]:
  assumes *compatible* $F\ G$
  shows *register* $(F;\ G)$
  ⟨*proof*⟩

lemma *register-pair-apply*:
  assumes ‹*compatible F G*›
  shows ‹$(F;\ G)\ (a \otimes_u b) = (F\ a) *_u (G\ b)$›
  ⟨*proof*⟩

lemma *register-pair-apply′*:
  assumes ‹*compatible F G*›
  shows ‹$(F;\ G)\ (a \otimes_u b) = (G\ b) *_u (F\ a)$›
  ⟨*proof*⟩

lemma *compatible-comp-left*[*simp*]: *compatible F G* $\Longrightarrow$ *register H* $\Longrightarrow$ *compatible* $(F \circ H)$ *G*
  ⟨*proof*⟩

lemma *compatible-comp-right*[*simp*]: *compatible F G* $\Longrightarrow$ *register H* $\Longrightarrow$ *compatible F* $(G \circ H)$
  ⟨*proof*⟩

lemma *compatible-comp-inner*[*simp*]:
  compatible F G $\Longrightarrow$ *register H* $\Longrightarrow$ *compatible* $(H \circ F)$ $(H \circ G)$
  ⟨*proof*⟩

lemma *compatible-register1*: ‹*compatible F G* $\Longrightarrow$ *register F*›
  ⟨*proof*⟩
lemma *compatible-register2*: ‹*compatible F G* $\Longrightarrow$ *register G*›
  ⟨*proof*⟩

lemma *pair-o-tensor*:
  assumes *compatible A B* **and** [*simp*]: ‹*register C*› **and** [*simp*]: ‹*register D*›
  shows $(A;\ B)\ o\ (C \otimes_r D) = (A\ o\ C;\ B\ o\ D)$
  ⟨*proof*⟩

lemma *compatible-tensor-id-update-left*[*simp*]:
  fixes F :: $'a$::*type update* $\Rightarrow$ $'c$::*type update* **and** G :: $'b$::*type update* $\Rightarrow$ $'c$::*type update*
  assumes *compatible F G*
  shows *compatible* $(\lambda a.\ Some \otimes_u F\ a)$ $(\lambda a.\ Some \otimes_u G\ a)$
  ⟨*proof*⟩

lemma *compatible-tensor-id-update-right*[*simp*]:
  fixes F :: $'a$::*type update* $\Rightarrow$ $'c$::*type update* **and** G :: $'b$::*type update* $\Rightarrow$ $'c$::*type update*
  assumes *compatible F G*
  shows *compatible* $(\lambda a.\ F\ a \otimes_u Some)$ $(\lambda a.\ G\ a \otimes_u Some)$
  ⟨*proof*⟩

lemma *compatible-tensor-id-update-rl*[*simp*]:
  assumes *register F* **and** *register G*
  shows *compatible* $(\lambda a.\ F\ a \otimes_u Some)$ $(\lambda a.\ Some \otimes_u G\ a)$
  ⟨*proof*⟩

lemma *compatible-tensor-id-update-lr*[*simp*]:
  assumes *register F* **and** *register G*
  shows *compatible* $(\lambda a.\ Some \otimes_u F\ a)$ $(\lambda a.\ G\ a \otimes_u Some)$
  ⟨*proof*⟩

lemma *register-comp-pair*:
  assumes [*simp*]: ‹*register F*› **and** [*simp*]: ‹*compatible G H*›
  shows $(F\ o\ G;\ F\ o\ H) = F\ o\ (G;\ H)$
⟨*proof*⟩

lemma *swap-registers-left*:
  assumes *compatible R S*
  shows $R\ a *_u S\ b *_u c = S\ b *_u R\ a *_u c$

⟨*proof*⟩

lemma *swap-registers-right*:
  assumes *compatible R S*
  shows $c *_u\ R\ a *_u\ S\ b = c *_u\ S\ b *_u\ R\ a$
  ⟨*proof*⟩

lemmas *compatible-ac-rules* = *swap-registers comp-update-assoc*[*symmetric*] *swap-registers-right*

6.6 Fst and Snd

definition *Fst* **where** ‹$Fst\ a = a \otimes_u\ Some$›
definition *Snd* **where** ‹$Snd\ a = Some \otimes_u\ a$›

lemma *register-Fst*[*simp*]: ‹*register Fst*›
  ⟨*proof*⟩

lemma *register-Snd*[*simp*]: ‹*register Snd*›
  ⟨*proof*⟩

lemma *compatible-Fst-Snd*[*simp*]: ‹*compatible Fst Snd*›
  ⟨*proof*⟩

lemmas *compatible-Snd-Fst*[*simp*] = *compatible-Fst-Snd*[*THEN compatible-sym*]

definition ‹*swap* = (*Snd*; *Fst*)›

lemma *swap-apply*[*simp*]: $swap\ (a \otimes_u\ b) = (b \otimes_u\ a)$
  ⟨*proof*⟩

lemma *swap-o-Fst*: *swap o Fst* = *Snd*
  ⟨*proof*⟩
lemma *swap-o-Snd*: *swap o Snd* = *Fst*
  ⟨*proof*⟩

lemma *register-swap*[*simp*]: ‹*register swap*›
  ⟨*proof*⟩

lemma *pair-Fst-Snd*: ‹(*Fst*; *Snd*) = *id*›
  ⟨*proof*⟩

lemma *swap-o-swap*[*simp*]: ‹*swap o swap* = *id*›
  ⟨*proof*⟩

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

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

lemma *register-pair-Fst*:
  assumes ‹*compatible F G*›
  shows ‹(*F*;*G*) *o Fst* = *F*›
  ⟨*proof*⟩

lemma *register-pair-Snd*:
  assumes ‹*compatible F G*›
  shows ‹(*F*;*G*) *o Snd* = *G*›
  ⟨*proof*⟩

lemma *register-Fst-register-Snd*[*simp*]:
  assumes ‹*register F*›
  shows ‹(*F o Fst*; *F o Snd*) = *F*›

⟨*proof*⟩

lemma *register-Snd-register-Fst*[*simp*]:
assumes ‹*register F*›
shows ‹$(F\ o\ Snd;\ F\ o\ Fst) = F\ o\ swap$›
⟨*proof*⟩

lemma *compatible3*[*simp*]:
assumes [*simp*]: *compatible F G* **and** *compatible G H* **and** *compatible F H*
shows *compatible* $(F;\ G)\ H$
⟨*proof*⟩

lemma *compatible3'*[*simp*]:
assumes *compatible F G* **and** *compatible G H* **and** *compatible F H*
shows *compatible* $F\ (G;\ H)$
⟨*proof*⟩

lemma *pair-o-swap*[*simp*]:
assumes [*simp*]: *compatible A B*
shows $(A;\ B)\ o\ swap = (B;\ A)$
⟨*proof*⟩

6.7 Compatibility of register tensor products

lemma *compatible-register-tensor*:
fixes F :: ‹*'a::type update ⇒ 'e::type update*› **and** G :: ‹*'b::type update ⇒ 'f::type update*›
and F' :: ‹*'c::type update ⇒ 'e update*› **and** G' :: ‹*'d::type update ⇒ 'f update*›
assumes [*simp*]: ‹*compatible F F'*›
assumes [*simp*]: ‹*compatible G G'*›
shows ‹*compatible* $(F \otimes_r G)\ (F' \otimes_r G')$›
⟨*proof*⟩

6.8 Associativity of the tensor product

definition *assoc* :: ‹*(('a::type×'b::type)×'c::type) update ⇒ ('a×('b×'c)) update*› **where**
‹$assoc = ((Fst;\ Snd\ o\ Fst);\ Snd\ o\ Snd)$›

lemma *assoc-is-hom*[*simp*]: ‹*preregister assoc*›
⟨*proof*⟩

lemma *assoc-apply*[*simp*]: ‹$assoc\ ((a \otimes_u b) \otimes_u c) = (a \otimes_u (b \otimes_u c))$›
⟨*proof*⟩

definition *assoc'* :: ‹*('a×('b×'c)) update ⇒ (('a::type×'b::type)×'c::type) update*› **where**
‹$assoc' = (Fst\ o\ Fst;\ (Fst\ o\ Snd;\ Snd))$›

lemma *assoc'-is-hom*[*simp*]: ‹*preregister assoc'*›
⟨*proof*⟩

lemma *assoc'-apply*[*simp*]: ‹$assoc'\ (a \otimes_u (b \otimes_u c)) = ((a \otimes_u b) \otimes_u c)$›
⟨*proof*⟩

lemma *register-assoc*[*simp*]: ‹*register assoc*›
⟨*proof*⟩

lemma *register-assoc'*[*simp*]: ‹*register assoc'*›
⟨*proof*⟩

lemma *pair-o-assoc*[*simp*]:
assumes [*simp*]: ‹*compatible F G*› ‹*compatible G H*› ‹*compatible F H*›
shows ‹$(F;\ (G;\ H)) \circ assoc = ((F;\ G);\ H)$›
⟨*proof*⟩

lemma *pair-o-assoc*′[*simp*]:
  assumes [*simp*]: ‹*compatible F G*› ‹*compatible G H*› ‹*compatible F H*›
  shows ‹$((F;\ G);\ H) \circ$ *assoc*′ $= (F;\ (G;\ H))$›
⟨*proof*⟩

lemma *assoc*′*-o-assoc*[*simp*]: ‹*assoc*′ *o assoc* = *id*›
  ⟨*proof*⟩

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

lemma *assoc-o-assoc*′[*simp*]: ‹*assoc o assoc*′ = *id*›
  ⟨*proof*⟩

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

lemma *inv-assoc*[*simp*]: ‹*inv assoc* = *assoc*′›
  ⟨*proof*⟩

lemma *inv-assoc*′[*simp*]: ‹*inv assoc*′ = *assoc*›
  ⟨*proof*⟩

lemma *bij-assoc*[*simp*]: ‹*bij assoc*›
  ⟨*proof*⟩

lemma *bij-assoc*′[*simp*]: ‹*bij assoc*′›
  ⟨*proof*⟩

6.9 Iso-registers

definition ‹*iso-register F* ⟷ *register F* ∧ (∃ *G*. *register G* ∧ *F o G* = *id* ∧ *G o F* = *id*)›
  for *F* :: ‹-::*type update* ⇒ -::*type update*›

lemma *iso-registerI*:
  assumes ‹*register F*› ‹*register G*› ‹*F o G* = *id*› ‹*G o F* = *id*›
  shows ‹*iso-register F*›
  ⟨*proof*⟩

lemma *iso-register-inv*: ‹*iso-register F* ⟹ *iso-register* (*inv F*)›
  ⟨*proof*⟩

lemma *iso-register-inv-comp1*: ‹*iso-register F* ⟹ *inv F o F* = *id*›
  ⟨*proof*⟩

lemma *iso-register-inv-comp2*: ‹*iso-register F* ⟹ *F o inv F* = *id*›
  ⟨*proof*⟩

lemma *iso-register-id*[*simp*]: ‹*iso-register id*›
  ⟨*proof*⟩

lemma *iso-register-is-register*: ‹*iso-register F* ⟹ *register F*›
  ⟨*proof*⟩

lemma *iso-register-comp*[*simp*]:
  assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›
  shows ‹*iso-register* (*F o G*)›
⟨*proof*⟩

lemma *iso-register-tensor-is-iso-register*[*simp*]:
  assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›

shows ‹*iso-register* $(F \otimes_r G)$›
⟨*proof*⟩

lemma *iso-register-bij*: ‹*iso-register* $F \Longrightarrow$ *bij* F›
⟨*proof*⟩

lemma *inv-register-tensor*[*simp*]:
assumes [*simp*]: ‹*iso-register* F› ‹*iso-register* G›
shows ‹*inv* $(F \otimes_r G) =$ *inv* $F \otimes_r$ *inv* G›
⟨*proof*⟩

lemma *iso-register-swap*[*simp*]: ‹*iso-register swap*›
⟨*proof*⟩

lemma *iso-register-assoc*[*simp*]: ‹*iso-register assoc*›
⟨*proof*⟩

lemma *iso-register-assoc'*[*simp*]: ‹*iso-register assoc'*›
⟨*proof*⟩

definition ‹*equivalent-registers* F $G \longleftrightarrow$ (*register* $F \wedge (\exists I.$ *iso-register* $I \wedge F$ *o* $I = G))$›
for F G :: ‹-::*type update* ⇒ -::*type update*›

lemma *iso-register-equivalent-id*[*simp*]: ‹*equivalent-registers id* $F \longleftrightarrow$ *iso-register* F›
⟨*proof*⟩

lemma *equivalent-registersI*:
assumes ‹*register* F›
assumes ‹*iso-register* I›
assumes ‹F *o* $I = G$›
shows ‹*equivalent-registers* F G›
⟨*proof*⟩

lemma *equivalent-registers-refl*: ‹*equivalent-registers* F F› **if** ‹*register* F›
⟨*proof*⟩

lemma *equivalent-registers-register-left*: ‹*equivalent-registers* F $G \Longrightarrow$ *register* F›
⟨*proof*⟩

lemma *equivalent-registers-register-right*: ‹*register* G› **if** ‹*equivalent-registers* F G›
⟨*proof*⟩

lemma *equivalent-registers-sym*:
assumes ‹*equivalent-registers* F G›
shows ‹*equivalent-registers* G F›
⟨*proof*⟩

lemma *equivalent-registers-trans*[*trans*]:
assumes ‹*equivalent-registers* F G› **and** ‹*equivalent-registers* G H›
shows ‹*equivalent-registers* F H›
⟨*proof*⟩

lemma *equivalent-registers-assoc*[*simp*]:
assumes [*simp*]: ‹*compatible* F G› ‹*compatible* F H› ‹*compatible* G H›
shows ‹*equivalent-registers* $(F;(G;H))$ $((F;G);H)$›
⟨*proof*⟩

lemma *equivalent-registers-pair-right*:
assumes [*simp*]: ‹*compatible* F G›
assumes *eq*: ‹*equivalent-registers* G H›
shows ‹*equivalent-registers* $(F;G)$ $(F;H)$›
⟨*proof*⟩

lemma *equivalent-registers-pair-left*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes *eq*: ‹*equivalent-registers F H*›
  shows ‹*equivalent-registers* (*F*;*G*) (*H*;*G*)›
⟨*proof*⟩

lemma *equivalent-registers-comp*:
  assumes ‹*register H*›
  assumes ‹*equivalent-registers F G*›
  shows ‹*equivalent-registers* (*H o F*) (*H o G*)›
  ⟨*proof*⟩

lemma *equivalent-registers-compatible1*:
  assumes ‹*compatible F G*›
  assumes ‹*equivalent-registers F F′*›
  shows ‹*compatible F′ G*›
  ⟨*proof*⟩

lemma *equivalent-registers-compatible2*:
  assumes ‹*compatible F G*›
  assumes ‹*equivalent-registers G G′*›
  shows ‹*compatible F G′*›
  ⟨*proof*⟩

6.10 Compatibility simplification

The simproc *compatibility-warn* produces helpful warnings for subgoals of the form *compatible x y* that are probably unsolvable due to missing declarations of variable compatibility facts. Same for subgoals of the form *register x*.

⟨*ML*⟩

named-theorems *register-attribute-rule-immediate*
named-theorems *register-attribute-rule*

lemmas [*register-attribute-rule*] = *conjunct1 conjunct2 iso-register-is-register iso-register-is-register*[*OF iso-register-inv*]
lemmas [*register-attribute-rule-immediate*] = *compatible-sym compatible-register1 compatible-register2*
  asm-rl[*of* ‹*compatible - -*›] *asm-rl*[*of* ‹*iso-register -*›] *asm-rl*[*of* ‹*register -*›] *iso-register-inv*

The following declares an attribute [*register*]. When the attribute is applied to a fact of the form *register F*, *iso-register F*, *compatible F G* or a conjunction of these, then those facts are added to the simplifier together with some derived theorems (e.g., *compatible F G* also adds *register F*).

In theory *Laws-Complement*, support for *is-unit-register F* and *complements F G* is added to this attribute.

⟨*ML*⟩

6.11 Notation

no-notation *map-comp* (**infixl** $*_u$ *55*)
no-notation *tensor-update* (**infixr** $\otimes_u$ *70*)

bundle *register-syntax* **begin**
notation *register-tensor* (**infixr** $\otimes_r$ *70*)
notation *register-pair* (′(-;-′))
end

end

7 Miscellaneous facts

This theory proves various facts that are not directly related to this developments but do not occur in the imported theories.

theory *Misc*
  imports
    Complex-Bounded-Operators.Cblinfun-Code
    HOL−Library.Z2
    Jordan-Normal-Form.Matrix
    Hilbert-Space-Tensor-Product.Weak-Star-Topology
begin

— Remove notation that collides with the notation we use
no-notation *Order.top* (⊤ı)
unbundle *no m-inv-syntax*
unbundle *no vec-syntax*
unbundle *no inner-syntax*

— Import notation from Bounded Operator and Jordan Normal Form libraries
unbundle *cblinfun-syntax*
unbundle *jnf-syntax*

The following declares the ML antiquotation `fact`. In ML code, `@{fact f}` for a theorem/fact name f is replaced by an ML string containing a printable(!) representation of fact. (I.e., if you print that string using writeln, the user can ctrl-click on it.)

This is useful when constructing diagnostic messages in ML code, e.g., `"Use the theorem " ^ @{fact thmname} ^ "here."`

⟨*ML*⟩

instantiation *bit* :: *enum* **begin**
definition *enum-bit* = [*0*::*bit*,*1*]
definition *enum-all-bit P* ⟷ *P* (*0*::*bit*) ∧ *P 1*
definition *enum-ex-bit P* ⟷ *P* (*0*::*bit*) ∨ *P 1*
instance
⟨*proof*⟩
end

lemma *card-bit*[*simp*]: *CARD*(*bit*) = *2*
  ⟨*proof*⟩

instantiation *bit* :: *card-UNIV* **begin**
definition *finite-UNIV* = *Phantom*(*bit*) *True*
definition *card-UNIV* = *Phantom*(*bit*) *2*
instance
  ⟨*proof*⟩
end

lemma *mat-of-rows-list-carrier*[*simp*]:
  mat-of-rows-list n vs ∈ *carrier-mat* (*length vs*) *n*
  dim-row (*mat-of-rows-list n vs*) = *length vs*
  dim-col (*mat-of-rows-list n vs*) = *n*
  ⟨*proof*⟩

lemma *mat-of-rows-list-carrier2xn*[*iff*]:
  mat-of-rows-list n [*a*,*b*] ∈ *carrier-mat 2 n*
  ⟨*proof*⟩

lemma *mat-of-rows-list-carrier4xn*[*iff*]:
  mat-of-rows-list n [*a*,*b*,*c*,*d*] ∈ *carrier-mat 4 n*
  ⟨*proof*⟩

lemma *prod-cases3′* [*cases type*]:
  obtains (*fields*) *a b c* **where** $y = ((a,\ b),\ c)$
  ⟨*proof*⟩

We define the following abbreviations:

- *mutually f* $(x_1,x_2,\ldots,x_n)$ expands to the conjuction of all $f\ x_i\ x_j$ with $i \neq j$.
- *each f* $(x_1,x_2,\ldots,x_n)$ expands to the conjuction of all $f\ x_i$.

syntax *-mutually* :: *′a ⇒ args ⇒ ′b* (*mutually* - ′(-′))
syntax *-mutually2* :: *′a ⇒ ′b ⇒ args ⇒ args ⇒ ′c*

translations *mutually f* (*x*) => *CONST True*
translations *mutually f* (*-args x y*) => *f x y* ∧ *f y x*
translations *mutually f* (*-args x* (*-args x′ xs*)) => *-mutually2 f x* (*-args x′ xs*) (*-args x′ xs*)
translations *-mutually2 f x y zs* => *f x y* ∧ *f y x* ∧ *-mutually f zs*
translations *-mutually2 f x* (*-args y ys*) *zs* => *f x y* ∧ *f y x* ∧ *-mutually2 f x ys zs*

syntax *-each* :: *′a ⇒ args ⇒ ′b* (*each* - ′(-′))
translations *each f* (*x*) => *f x*
translations *-each f* (*-args x xs*) => *f x* ∧ *-each f xs*

lemma *dim-col-mat-adjoint*[*simp*]: *dim-col* (*mat-adjoint m*) = *dim-row m*
  ⟨*proof*⟩
lemma *dim-row-mat-adjoint*[*simp*]: *dim-row* (*mat-adjoint m*) = *dim-col m*
  ⟨*proof*⟩

lemma *invI*:
  assumes ‹*inj f*›
  assumes ‹*x* = *f y*›
  shows ‹*inv f x* = *y*›
  ⟨*proof*⟩

instantiation *prod* :: (*default*,*default*) *default* **begin**
definition ‹*default-prod* = (*default*, *default*)›
instance⟨*proof*⟩
end

instance *bit* :: *default*⟨*proof*⟩

lemma *surj-from-comp*:
  assumes ‹*surj* (*g o f*)›
  assumes ‹*inj g*›
  shows ‹*surj f*›
  ⟨*proof*⟩

lemma *double-exists*: ‹(∃ *x y*. *Q x y*) ⟷ (∃ *z*. *Q* (*fst z*) (*snd z*))›
  ⟨*proof*⟩

lemma *Ex-iffI*:
  assumes ‹⋀*x*. *P x* ⟹ *Q* (*f x*)›
  assumes ‹⋀*x*. *Q x* ⟹ *P* (*g x*)›

shows ‹*Ex P* ⟷ *Ex Q*›
⟨*proof*⟩

lemma *cspan-space-as-set*[*simp*]: ‹*cspan* (*space-as-set X*) = *space-as-set X*›
⟨*proof*⟩

thm *lift-cblinfun-comp*
lemma *lift-cblinfun-comp2*:
assumes ‹$a\ o_{CL}\ b = c$›
shows ‹$(d\ o_{CL}\ a)\ o_{CL}\ b = d\ o_{CL}\ c$›
⟨*proof*⟩
lemmas *lift-cblinfun-comp* = *lift-cblinfun-comp lift-cblinfun-comp2*

unbundle *no cblinfun-syntax*
unbundle *no jnf-syntax*

end

8 Derived facts about classical registers

theory *Classical-Extra*
imports *Laws-Classical Misc*
begin

lemma *register-from-getter-setter-of-getter-setter*[*simp*]: ‹*register-from-getter-setter* (*getter F*) (*setter F*) = *F*›
if ‹*register F*›
⟨*proof*⟩

lemma *valid-getter-setter-getter-setter*[*simp*]: ‹*valid-getter-setter* (*getter F*) (*setter F*)› **if** ‹*register F*›
⟨*proof*⟩

lemma *register-register-from-getter-setter*[*simp*]: ‹*register* (*register-from-getter-setter g s*)› **if** ‹*valid-getter-setter g s*›
⟨*proof*⟩

definition ‹*total-fun f* = (∀ *x*. *f x* ≠ *None*)›

lemma *register-total*:
assumes ‹*register F*›
assumes ‹*total-fun a*›
shows ‹*total-fun* (*F a*)›
⟨*proof*⟩

lemma *register-apply*:
assumes ‹*register F*›
shows ‹*Some o register-apply F a* = *F* (*Some o a*)›
⟨*proof*⟩

lemma *register-empty*:
assumes ‹*preregister F*›
shows ‹*F Map.empty* = *Map.empty*›
⟨*proof*⟩

lemma *compatible-setter*:
fixes *F* :: ‹('*a*,'*c*) *preregister*› **and** *G* :: ‹('*b*,'*c*) *preregister*›
assumes [*simp*]: ‹*register F*› ‹*register G*›
shows ‹*compatible F G* ⟷ (∀ *a b*. *setter F a o setter G b* = *setter G b o setter F a*)›
⟨*proof*⟩

lemma *register-from-getter-setter-compatibleI*[*intro*]:

assumes [*simp*]: ‹*valid-getter-setter g s*› ‹*valid-getter-setter g′ s′*›
assumes ‹$\bigwedge$*x y m. s x* (*s′ y m*) = *s′ y* (*s x m*)›
shows ‹*compatible* (*register-from-getter-setter g s*) (*register-from-getter-setter g′ s′*)›
⟨*proof*⟩

lemma *separating-update1*:
‹*separating TYPE*(-) {*update1 x y* | *x y. True*}›
⟨*proof*⟩

definition *permutation-register* (*p*::′*b*⇒′*a*) = *register-from-getter-setter p* (λ*a* -. *inv p a*)

lemma *permutation-register-register*[*simp*]:
fixes *p* :: ′*b* ⇒ ′*a*
assumes [*simp*]: *bij p*
shows *register* (*permutation-register p*)
⟨*proof*⟩

lemma *getter-permutation-register*: ‹*bij p* ⟹ *getter* (*permutation-register p*) = *p*›
⟨*proof*⟩

lemma *setter-permutation-register*: ‹*bij p* ⟹ *setter* (*permutation-register p*) *a m* = *inv p a*›
⟨*proof*⟩

definition *empty-var* :: ‹′*a*::{*CARD-1*} *update* ⇒ ′*b update*› **where**
empty-var = *register-from-getter-setter* (λ-. *undefined*) (λ- *m. m*)

lemma *valid-empty-var*[*simp*]: ‹*valid-getter-setter* (λ-. (*undefined*::-::*CARD-1*)) (λ- *m. m*)›
⟨*proof*⟩

lemma *register-empty-var*[*simp*]: ‹*register empty-var*›
⟨*proof*⟩

lemma *getter-empty-var*[*simp*]: ‹*getter empty-var m* = *undefined*›
⟨*proof*⟩

lemma *setter-empty-var*[*simp*]: ‹*setter empty-var a m* = *m*›
⟨*proof*⟩

lemma *empty-var-compatible*[*simp*]: ‹*compatible empty-var X*› **if** [*simp*]: ‹*register X*›
⟨*proof*⟩

lemma *empty-var-compatible′*[*simp*]: ‹*register X* ⟹ *compatible X empty-var*›
⟨*proof*⟩

Example

record *memory* =
x :: *int*∗*int*
y :: *nat*

definition *X* = *register-from-getter-setter x* (λ*a b. b*⦇*x*:=*a*⦈)
definition *Y* = *register-from-getter-setter y* (λ*a b. b*⦇*y*:=*a*⦈)

lemma *validX*[*simp*]: ‹*valid-getter-setter x* (λ*a b. b*⦇*x*:=*a*⦈)›
⟨*proof*⟩

lemma *registerX*[*simp*]: ‹*register X*›
⟨*proof*⟩

lemma *validY*[*simp*]: ‹*valid-getter-setter y* (λ*a b. b*⦇*y*:=*a*⦈)›
⟨*proof*⟩

lemma *registerY*[*simp*]: ‹*register Y*›
⟨*proof*⟩

lemma *compatibleXY*[*simp*]: ‹*compatible X Y*›
  ⟨*proof*⟩

hide-const (**open**) *x y x-update y-update X Y*

end

9 Quantum instantiation of registers

theory *Axioms-Quantum*
  imports *Jordan-Normal-Form.Matrix-Impl HOL−Library.Rewrite*
     Complex-Bounded-Operators.Complex-L2
     Hilbert-Space-Tensor-Product.Hilbert-Space-Tensor-Product
     Hilbert-Space-Tensor-Product.Weak-Star-Topology
     Hilbert-Space-Tensor-Product.Partial-Trace
     Hilbert-Space-Tensor-Product.Von-Neumann-Algebras
     With-Type.With-Type
     Misc
begin

unbundle *cblinfun-syntax*
unbundle *no m-inv-syntax*

type-synonym *'a update* = ‹(*'a ell2*, *'a ell2*) *cblinfun*›

definition *preregister* :: ‹(*'a update* ⇒ *'b update*) ⇒ *bool*› **where**
  ‹*preregister F* ⟷ *bounded-clinear F* ∧ *continuous-map weak-star-topology weak-star-topology F*›

lemma *preregister-mult-right*: ‹*preregister* (*λa. a* o_{CL} *z*)›
  ⟨*proof*⟩

lemma *preregister-mult-left*: ‹*preregister* (*λa. z* o_{CL} *a*)›
  ⟨*proof*⟩

lemma *comp-preregister*: *preregister F* ⟹ *preregister G* ⟹ *preregister* (*G* ∘ *F*)
  ⟨*proof*⟩

lemma *id-preregister*: ‹*preregister id*›
  ⟨*proof*⟩

lemma *tensor-extensionality*:
  ‹*preregister F* ⟹ *preregister G* ⟹ (⋀*a b. F* (*tensor-op a b*) = *G* (*tensor-op a b*)) ⟹ *F* = *G*›
  ⟨*proof*⟩

definition *register* :: ‹(*'a update* ⇒ *'b update*) ⇒ *bool*› **where**
  register F ⟷
    bounded-clinear F
   ∧ *continuous-map weak-star-topology weak-star-topology F*
   ∧ *F id-cblinfun* = *id-cblinfun*
   ∧ (∀ *a b. F*(*a* o_{CL} *b*) = *F a* o_{CL} *F b*)
   ∧ (∀ *a. F* (*a*∗) = (*F a*)∗)

lemma *register-of-id*: ‹*register F* ⟹ *F id-cblinfun* = *id-cblinfun*›
  ⟨*proof*⟩

lemma *register-id*: ‹*register id*›
  ⟨*proof*⟩

lemma *register-preregister*: *register F* ⟹ *preregister F*
  ⟨*proof*⟩

lemma *register-comp*: *register F* $\Longrightarrow$ *register G* $\Longrightarrow$ *register* $(G \circ F)$
⟨*proof*⟩

lemma *register-mult*: *register F* $\Longrightarrow$ *cblinfun-compose* $(F\ a)$ $(F\ b)$ = *F* (*cblinfun-compose a b*)
⟨*proof*⟩

lemma *register-tensor-left*: ‹*register* (λ*a*. *tensor-op a id-cblinfun*)›
⟨*proof*⟩

lemma *register-tensor-right*: ‹*register* (λ*a*. *tensor-op id-cblinfun a*)›
⟨*proof*⟩

definition *register-pair* ::
‹('*a update* ⇒ '*c update*) ⇒ ('*b update* ⇒ '*c update*)
⇒ (('*a*×'*b*) *update* ⇒ '*c update*)› **where**
‹*register-pair F G* = (*if register F* ∧ *register G* ∧ (∀ *a b*. *F a* o_{CL} *G b* = *G b* o_{CL} *F a*) *then*
SOME R. (∀ *a b*. *register R* ∧ *R* (*a* $\otimes_o$ *b*) = *F a* o_{CL} *G b*) *else* (λ-. *0*))›

lemma *cbilinear-F-comp-G*[*simp*]: ‹*clinear F* $\Longrightarrow$ *clinear G* $\Longrightarrow$ *cbilinear* (λ*a b*. *F a* o_{CL} *G b*)›
⟨*proof*⟩

lemma *register-projector*:
assumes *register F*
assumes *is-Proj a*
shows *is-Proj* (*F a*)
⟨*proof*⟩

lemma *register-unitary*:
assumes *register F*
assumes *unitary a*
shows *unitary* (*F a*)
⟨*proof*⟩

definition ‹*register-decomposition-basis* Φ = (*SOME B*. *is-ortho-set B* ∧ (∀ *b*∈*B*. *norm b* = *1*) ∧ *ccspan B* = Φ (*butterfly* (*ket undefined*) (*ket undefined*)) $*_S$ ⊤)›
for Φ :: ‹'*a update* ⇒ '*b update*›

lemma *register-decomposition*:
fixes Φ :: ‹'*a update* ⇒ '*b update*›
assumes [*simp*]: ‹*register* Φ›
shows ‹*let* '*c*::*type* = *register-decomposition-basis* Φ *in*
(∃ *U* :: ('*a* × '*c*) *ell2* $\Rightarrow_{CL}$ '*b ell2*. *unitary U* ∧
(∀ ϑ. Φ ϑ = *sandwich U* (ϑ $\otimes_o$ *id-cblinfun*)))›
— Proof based on [Daw21]
⟨*proof*⟩

lemma *register-bounded-clinear*: ‹*register F* $\Longrightarrow$ *bounded-clinear F*›
⟨*proof*⟩

lemma *clinear-register*: ‹*register F* $\Longrightarrow$ *clinear F*›
⟨*proof*⟩

lemma *weak-star-cont-register*: ‹*register F* $\Longrightarrow$ *continuous-map weak-star-topology weak-star-topology F*›
⟨*proof*⟩

lemma *register-inv-weak-star-continuous*:
assumes ‹*register F*›
shows ‹*continuous-map* (*subtopology weak-star-topology* (*range F*)) *weak-star-topology* (*inv F*)›
⟨*proof*⟩

lemma *register-inj*: ‹*inj-on F X*› **if** [*simp*]: ‹*register F*›
⟨*proof*⟩

lemma *register-norm*: ‹*norm* (*F* *a*) = *norm* *a*› **if** ‹*register* *F*›
⟨*proof*⟩

lemma *unitary-sandwich-register*: ‹*unitary* *a* ⟹ *register* (*sandwich* *a*)›
  ⟨*proof*⟩

lemma *register-adj*: ‹*register* *F* ⟹ *F* (*a*∗) = (*F* *a*)∗›
  ⟨*proof*⟩

lemma *right-register-tensor-ex*: ‹∃ *T* :: (′*a* × ′*b*) *update* ⇒ (′*a* × ′*c*) *update*.
        register *T* ∧ (∀ *a* *b*. *T* (*a* $\otimes_o$ *b*) = *a* $\otimes_o$ *F* *b*)› **if** ‹*register* *F*›
⟨*proof*⟩

lemma
  fixes *F* :: ‹′*a* *update* ⇒ ′*c* *update*› **and** *G*
  assumes [*simp*]: ‹*register* *F*› ‹*register* *G*›
  assumes *FG-comm*: ‹⋀*a* *b*. *F* *a* o_{CL} *G* *b* = *G* *b* o_{CL} *F* *a*›
  shows *register-pair-apply*: ‹(*register-pair* *F* *G*) (*tensor-op* *a* *b*) = *F* *a* o_{CL} *G* *b*›
    and *register-pair-is-register*: ‹*register* (*register-pair* *F* *G*)›
⟨*proof*⟩

unbundle *no* *cblinfun-syntax*

end

10 Generic laws about registers, instantiated quantumly

theory *Laws-Quantum*
  imports *Axioms-Quantum*
begin

This notation is only used inside this file

notation *cblinfun-compose* (**infixl** $*_u$ *55*)
notation *tensor-op* (**infixr** $\otimes_u$ *70*)
notation *register-pair* (′(-;-′))

10.1 Elementary facts

declare *id-preregister*[*simp*]
declare *cblinfun-compose-id-left*[*simp*]
declare *cblinfun-compose-id-right*[*simp*]
declare *register-preregister*[*simp*]
declare *register-comp*[*simp*]
declare *register-of-id*[*simp*]
declare *register-tensor-left*[*simp*]
declare *register-tensor-right*[*simp*]
declare *preregister-mult-right*[*simp*]
declare *preregister-mult-left*[*simp*]
declare *register-id*[*simp*]

10.2 Preregisters

lemma *preregister-tensor-left*[*simp*]: ‹*preregister* (λ*b*::′*b*::*type* *update*. *tensor-op* *a* *b*)›
  for *a* :: ‹′*a*::*type* *update*›
⟨*proof*⟩

lemma *preregister-tensor-right*[*simp*]: ‹*preregister* (λ*a*::′*a*::*type* *update*. *tensor-op* *a* *b*)›
  for *b* :: ‹′*b*::*type* *update*›

⟨*proof*⟩

10.3 Registers

lemma *id-update-tensor-register*[*simp*]:
  assumes ‹*register F*›
  shows ‹*register* (*λa*::*'a*::*type update. id-cblinfun* $\otimes_u$ *F a*)›
  ⟨*proof*⟩

lemma *register-tensor-id-update*[*simp*]:
  assumes ‹*register F*›
  shows ‹*register* (*λa*::*'a*::*type update. F a* $\otimes_u$ *id-cblinfun*)›
  ⟨*proof*⟩

10.4 Tensor product of registers

definition *register-tensor* (**infixr** $\otimes_r$ *70*) **where**
  register-tensor F G = register-pair (*λa. tensor-op* (*F a*) *id-cblinfun*) (*λb. tensor-op id-cblinfun* (*G b*))

lemma *register-tensor-is-register*:
  fixes *F* :: *'a*::*type update* ⇒ *'b*::*type update* **and** *G* :: *'c*::*type update* ⇒ *'d*::*type update*
  shows *register F* ⟹ *register G* ⟹ *register* (*F* $\otimes_r$ *G*)
  ⟨*proof*⟩

lemma *register-tensor-apply*[*simp*]:
  fixes *F* :: *'a*::*type update* ⇒ *'b*::*type update* **and** *G* :: *'c*::*type update* ⇒ *'d*::*type update*
  assumes ‹*register F*› **and** ‹*register G*›
  shows (*F* $\otimes_r$ *G*) (*a* $\otimes_u$ *b*) = *F a* $\otimes_u$ *G b*
  ⟨*proof*⟩

definition *separating* (-::*'b*::*type itself*) *A* ⟷
  (∀ *F G* :: *'a*::*type update* ⇒ *'b update. preregister F* ⟶ *preregister G* ⟶ (∀ *x*∈*A*. *F x* = *G x*) ⟶ *F* = *G*)

lemma *separating-UNIV*[*simp*]: ‹*separating TYPE*(-) *UNIV*›
  ⟨*proof*⟩

lemma *separating-mono*: ‹*A* ⊆ *B* ⟹ *separating TYPE*(*'a*::*type*) *A* ⟹ *separating TYPE*(*'a*) *B*›
  ⟨*proof*⟩

lemma *register-eqI*: ‹*separating TYPE*(*'b*::*type*) *A* ⟹ *preregister F* ⟹ *preregister G* ⟹ (⋀*x*. *x*∈*A* ⟹ *F x* = *G x*) ⟹ *F* = (*G*::- ⇒ *'b update*)›
  ⟨*proof*⟩

lemma *separating-tensor*:
  fixes *A* :: ‹*'a*::*type update set*› **and** *B* :: ‹*'b*::*type update set*›
  assumes [*simp*]: ‹*separating TYPE*(*'c*::*type*) *A*›
  assumes [*simp*]: ‹*separating TYPE*(*'c*) *B*›
  shows ‹*separating TYPE*(*'c*) {*a* $\otimes_u$ *b* | *a b*. *a*∈*A* ∧ *b*∈*B*}›
⟨*proof*⟩

lemma *register-tensor-distrib*:
  assumes [*simp*]: ‹*register F*› ‹*register G*› ‹*register H*› ‹*register L*›
  shows ‹(*F* $\otimes_r$ *G*) *o* (*H* $\otimes_r$ *L*) = (*F o H*) $\otimes_r$ (*G o L*)›
  ⟨*proof*⟩

The following is easier to apply using the *rule*-method than *separating-tensor*

lemma *separating-tensor'*:
  fixes *A* :: ‹*'a*::*type update set*› **and** *B* :: ‹*'b*::*type update set*›
  assumes ‹*separating TYPE*(*'c*::*type*) *A*›
  assumes ‹*separating TYPE*(*'c*) *B*›
  assumes ‹*C* = {*a* $\otimes_u$ *b* | *a b*. *a*∈*A* ∧ *b*∈*B*}›
  shows ‹*separating TYPE*(*'c*) *C*›
  ⟨*proof*⟩

lemma *tensor-extensionality3*:
  fixes $F\ G$ :: ‹$('a{::}type\times 'b{::}type\times 'c{::}type)$ *update* ⇒ $'d{::}type$ *update*›
  assumes [*simp*]: ‹*register F*› ‹*register G*›
  assumes $\bigwedge f\ g\ h.\ F\ (f \otimes_u g \otimes_u h) = G\ (f \otimes_u g \otimes_u h)$
  shows $F = G$
⟨*proof*⟩

lemma *tensor-extensionality3'*:
  fixes $F\ G$ :: ‹$(('a{::}type\times 'b{::}type)\times 'c{::}type)$ *update* ⇒ $'d{::}type$ *update*›
  assumes [*simp*]: ‹*register F*› ‹*register G*›
  assumes $\bigwedge f\ g\ h.\ F\ ((f \otimes_u g) \otimes_u h) = G\ ((f \otimes_u g) \otimes_u h)$
  shows $F = G$
⟨*proof*⟩

lemma *register-tensor-id*[*simp*]: ‹$id \otimes_r id = id$›
  ⟨*proof*⟩

10.5 Pairs and compatibility

definition *compatible* :: ‹$('a{::}type$ *update* ⇒ $'c{::}type$ *update*)
      ⇒ ($'b{::}type$ *update* ⇒ $'c$ *update*) ⇒ *bool*› **where**
  ‹*compatible F G* ⟷ *register F* ∧ *register G* ∧ $(\forall\ a\ b.\ F\ a *_u G\ b = G\ b *_u F\ a)$›

lemma *compatibleI*:
  assumes *register F* **and** *register G*
  assumes ‹$\bigwedge a\ b.\ (F\ a) *_u (G\ b) = (G\ b) *_u (F\ a)$›
  shows *compatible F G*
  ⟨*proof*⟩

lemma *swap-registers*:
  assumes *compatible R S*
  shows $R\ a *_u S\ b = S\ b *_u R\ a$
  ⟨*proof*⟩

lemma *compatible-sym*: *compatible x y* ⟹ *compatible y x*
  ⟨*proof*⟩

lemma *pair-is-register*[*simp*]:
  assumes *compatible F G*
  shows *register* $(F;\ G)$
  ⟨*proof*⟩

lemma *register-pair-apply*:
  assumes ‹*compatible F G*›
  shows ‹$(F;\ G)\ (a \otimes_u b) = (F\ a) *_u (G\ b)$›
  ⟨*proof*⟩

lemma *register-pair-apply'*:
  assumes ‹*compatible F G*›
  shows ‹$(F;\ G)\ (a \otimes_u b) = (G\ b) *_u (F\ a)$›
  ⟨*proof*⟩

lemma *compatible-comp-left*[*simp*]: *compatible F G* ⟹ *register H* ⟹ *compatible* $(F \circ H)\ G$
  ⟨*proof*⟩

lemma *compatible-comp-right*[*simp*]: *compatible F G* ⟹ *register H* ⟹ *compatible* $F\ (G \circ H)$
  ⟨*proof*⟩

lemma *compatible-comp-inner*[*simp*]:
  compatible F G ⟹ *register H* ⟹ *compatible* $(H \circ F)\ (H \circ G)$

⟨*proof*⟩

lemma *compatible-register1*: ‹*compatible F G* $\Longrightarrow$ *register F*›
⟨*proof*⟩
lemma *compatible-register2*: ‹*compatible F G* $\Longrightarrow$ *register G*›
⟨*proof*⟩

lemma *pair-o-tensor*:
assumes *compatible A B* **and** [*simp*]: ‹*register C*› **and** [*simp*]: ‹*register D*›
shows $(A;\ B)\ o\ (C \otimes_r D) = (A\ o\ C;\ B\ o\ D)$
⟨*proof*⟩

lemma *compatible-tensor-id-update-left*[*simp*]:
fixes *F* :: *'a::type update* ⇒ *'c::type update* **and** *G* :: *'b::type update* ⇒ *'c::type update*
assumes *compatible F G*
shows *compatible* ($\lambda a.$ *id-cblinfun* $\otimes_u$ *F a*) ($\lambda a.$ *id-cblinfun* $\otimes_u$ *G a*)
⟨*proof*⟩

lemma *compatible-tensor-id-update-right*[*simp*]:
fixes *F* :: *'a::type update* ⇒ *'c::type update* **and** *G* :: *'b::type update* ⇒ *'c::type update*
assumes *compatible F G*
shows *compatible* ($\lambda a.$ *F a* $\otimes_u$ *id-cblinfun*) ($\lambda a.$ *G a* $\otimes_u$ *id-cblinfun*)
⟨*proof*⟩

lemma *compatible-tensor-id-update-rl*[*simp*]:
assumes *register F* **and** *register G*
shows *compatible* ($\lambda a.$ *F a* $\otimes_u$ *id-cblinfun*) ($\lambda a.$ *id-cblinfun* $\otimes_u$ *G a*)
⟨*proof*⟩

lemma *compatible-tensor-id-update-lr*[*simp*]:
assumes *register F* **and** *register G*
shows *compatible* ($\lambda a.$ *id-cblinfun* $\otimes_u$ *F a*) ($\lambda a.$ *G a* $\otimes_u$ *id-cblinfun*)
⟨*proof*⟩

lemma *register-comp-pair*:
assumes [*simp*]: ‹*register F*› **and** [*simp*]: ‹*compatible G H*›
shows $(F\ o\ G;\ F\ o\ H) = F\ o\ (G;\ H)$
⟨*proof*⟩

lemma *swap-registers-left*:
assumes *compatible R S*
shows $R\ a *_u S\ b *_u c = S\ b *_u R\ a *_u c$
⟨*proof*⟩

lemma *swap-registers-right*:
assumes *compatible R S*
shows $c *_u R\ a *_u S\ b = c *_u S\ b *_u R\ a$
⟨*proof*⟩

lemmas *compatible-ac-rules* = *swap-registers cblinfun-compose-assoc*[*symmetric*] *swap-registers-right*

10.6 Fst and Snd

definition *Fst* **where** ‹*Fst a* = *a* $\otimes_u$ *id-cblinfun*›
definition *Snd* **where** ‹*Snd a* = *id-cblinfun* $\otimes_u$ *a*›

lemma *register-Fst*[*simp*]: ‹*register Fst*›
⟨*proof*⟩

lemma *register-Snd*[*simp*]: ‹*register Snd*›
⟨*proof*⟩

lemma *compatible-Fst-Snd*[*simp*]: ‹*compatible Fst Snd*›

⟨*proof*⟩

lemmas *compatible-Snd-Fst*[*simp*] = *compatible-Fst-Snd*[*THEN compatible-sym*]

definition ‹*swap* = (*Snd*; *Fst*)›

lemma *swap-apply*[*simp*]: *swap* ($a \otimes_u b$) = ($b \otimes_u a$)
⟨*proof*⟩

lemma *swap-o-Fst*: *swap o Fst* = *Snd*
⟨*proof*⟩
lemma *swap-o-Snd*: *swap o Snd* = *Fst*
⟨*proof*⟩

lemma *register-swap*[*simp*]: ‹*register swap*›
⟨*proof*⟩

lemma *pair-Fst-Snd*: ‹(*Fst*; *Snd*) = *id*›
⟨*proof*⟩

lemma *swap-o-swap*[*simp*]: ‹*swap o swap* = *id*›
⟨*proof*⟩

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

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

lemma *register-pair-Fst*:
assumes ‹*compatible F G*›
shows ‹(*F*;*G*) *o Fst* = *F*›
⟨*proof*⟩

lemma *register-pair-Snd*:
assumes ‹*compatible F G*›
shows ‹(*F*;*G*) *o Snd* = *G*›
⟨*proof*⟩

lemma *register-Fst-register-Snd*[*simp*]:
assumes ‹*register F*›
shows ‹(*F o Fst*; *F o Snd*) = *F*›
⟨*proof*⟩

lemma *register-Snd-register-Fst*[*simp*]:
assumes ‹*register F*›
shows ‹(*F o Snd*; *F o Fst*) = *F o swap*›
⟨*proof*⟩

lemma *compatible3*[*simp*]:
assumes [*simp*]: *compatible F G* **and** *compatible G H* **and** *compatible F H*
shows *compatible* (*F*; *G*) *H*
⟨*proof*⟩

lemma *compatible3'*[*simp*]:
assumes *compatible F G* **and** *compatible G H* **and** *compatible F H*
shows *compatible F* (*G*; *H*)
⟨*proof*⟩

lemma *pair-o-swap*[*simp*]:
assumes [*simp*]: *compatible A B*
shows (*A*; *B*) *o swap* = (*B*; *A*)

⟨*proof*⟩

10.7 Compatibility of register tensor products

lemma *compatible-register-tensor*:
  fixes F :: ‹$'a$::*type update* ⇒ $'e$::*type update*› **and** G :: ‹$'b$::*type update* ⇒ $'f$::*type update*›
    and F' :: ‹$'c$::*type update* ⇒ $'e$ *update*› **and** G' :: ‹$'d$::*type update* ⇒ $'f$ *update*›
  assumes [*simp*]: ‹*compatible* F F'›
  assumes [*simp*]: ‹*compatible* G G'›
  shows ‹*compatible* ($F \otimes_r G$) ($F' \otimes_r G'$)›
⟨*proof*⟩

10.8 Associativity of the tensor product

definition *assoc* :: ‹(($'a$::*type*×$'b$::*type*)×$'c$::*type*) *update* ⇒ ($'a$×($'b$×$'c$)) *update*› **where**
  ‹*assoc* = ((*Fst*; *Snd o Fst*); *Snd o Snd*)›

lemma *assoc-is-hom*[*simp*]: ‹*preregister assoc*›
  ⟨*proof*⟩

lemma *assoc-apply*[*simp*]: ‹*assoc* (($a \otimes_u b$) $\otimes_u c$) = ($a \otimes_u$ ($b \otimes_u c$))›
  ⟨*proof*⟩

definition *assoc'* :: ‹($'a$×($'b$×$'c$)) *update* ⇒ (($'a$::*type*×$'b$::*type*)×$'c$::*type*) *update*› **where**
  ‹*assoc'* = (*Fst o Fst*; (*Fst o Snd*; *Snd*))›

lemma *assoc'-is-hom*[*simp*]: ‹*preregister assoc'*›
  ⟨*proof*⟩

lemma *assoc'-apply*[*simp*]: ‹*assoc'* ($a \otimes_u$ ($b \otimes_u c$)) = (($a \otimes_u b$) $\otimes_u c$)›
  ⟨*proof*⟩

lemma *register-assoc*[*simp*]: ‹*register assoc*›
  ⟨*proof*⟩

lemma *register-assoc'*[*simp*]: ‹*register assoc'*›
  ⟨*proof*⟩

lemma *pair-o-assoc*[*simp*]:
  assumes [*simp*]: ‹*compatible* F G› ‹*compatible* G H› ‹*compatible* F H›
  shows ‹(F; (G; H)) ∘ *assoc* = ((F; G); H)›
⟨*proof*⟩

lemma *pair-o-assoc'*[*simp*]:
  assumes [*simp*]: ‹*compatible* F G› ‹*compatible* G H› ‹*compatible* F H›
  shows ‹((F; G); H) ∘ *assoc'* = (F; (G; H))›
⟨*proof*⟩

lemma *assoc'-o-assoc*[*simp*]: ‹*assoc' o assoc* = *id*›
  ⟨*proof*⟩

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

lemma *assoc-o-assoc'*[*simp*]: ‹*assoc o assoc'* = *id*›
  ⟨*proof*⟩

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

lemma *inv-assoc*[*simp*]: ‹*inv assoc* = *assoc'*›
  ⟨*proof*⟩

lemma *inv-assoc'*[*simp*]: ‹*inv assoc'* = *assoc*›

⟨*proof*⟩

lemma *bij-assoc*[*simp*]: ‹*bij assoc*›
⟨*proof*⟩

lemma *bij-assoc'*[*simp*]: ‹*bij assoc'*›
⟨*proof*⟩

10.9 Iso-registers

definition ‹*iso-register F* $\longleftrightarrow$ *register F* $\wedge$ ($\exists$ *G*. *register G* $\wedge$ *F o G* = *id* $\wedge$ *G o F* = *id*)›
for *F* :: ‹-::*type update* $\Rightarrow$ -::*type update*›

lemma *iso-registerI*:
assumes ‹*register F*› ‹*register G*› ‹*F o G* = *id*› ‹*G o F* = *id*›
shows ‹*iso-register F*›
⟨*proof*⟩

lemma *iso-register-inv*: ‹*iso-register F* $\Longrightarrow$ *iso-register* (*inv F*)›
⟨*proof*⟩

lemma *iso-register-inv-comp1*: ‹*iso-register F* $\Longrightarrow$ *inv F o F* = *id*›
⟨*proof*⟩

lemma *iso-register-inv-comp2*: ‹*iso-register F* $\Longrightarrow$ *F o inv F* = *id*›
⟨*proof*⟩

lemma *iso-register-id*[*simp*]: ‹*iso-register id*›
⟨*proof*⟩

lemma *iso-register-is-register*: ‹*iso-register F* $\Longrightarrow$ *register F*›
⟨*proof*⟩

lemma *iso-register-comp*[*simp*]:
assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›
shows ‹*iso-register* (*F o G*)›
⟨*proof*⟩

lemma *iso-register-tensor-is-iso-register*[*simp*]:
assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›
shows ‹*iso-register* (*F* $\otimes_r$ *G*)›
⟨*proof*⟩

lemma *iso-register-bij*: ‹*iso-register F* $\Longrightarrow$ *bij F*›
⟨*proof*⟩

lemma *inv-register-tensor*[*simp*]:
assumes [*simp*]: ‹*iso-register F*› ‹*iso-register G*›
shows ‹*inv* (*F* $\otimes_r$ *G*) = *inv F* $\otimes_r$ *inv G*›
⟨*proof*⟩

lemma *iso-register-swap*[*simp*]: ‹*iso-register swap*›
⟨*proof*⟩

lemma *iso-register-assoc*[*simp*]: ‹*iso-register assoc*›
⟨*proof*⟩

lemma *iso-register-assoc'*[*simp*]: ‹*iso-register assoc'*›
⟨*proof*⟩

definition ‹*equivalent-registers F G* $\longleftrightarrow$ (*register F* $\wedge$ ($\exists$ *I*. *iso-register I* $\wedge$ *F o I* = *G*))›

for *F G* :: ‹-::*type update* ⇒ -::*type update*›

lemma *iso-register-equivalent-id*[*simp*]: ‹*equivalent-registers id F* ⟷ *iso-register F*›
⟨*proof*⟩

lemma *equivalent-registersI*:
assumes ‹*register F*›
assumes ‹*iso-register I*›
assumes ‹*F o I* = *G*›
shows ‹*equivalent-registers F G*›
⟨*proof*⟩

lemma *equivalent-registers-refl*: ‹*equivalent-registers F F*› **if** ‹*register F*›
⟨*proof*⟩

lemma *equivalent-registers-register-left*: ‹*equivalent-registers F G* ⟹ *register F*›
⟨*proof*⟩

lemma *equivalent-registers-register-right*: ‹*register G*› **if** ‹*equivalent-registers F G*›
⟨*proof*⟩

lemma *equivalent-registers-sym*:
assumes ‹*equivalent-registers F G*›
shows ‹*equivalent-registers G F*›
⟨*proof*⟩

lemma *equivalent-registers-trans*[*trans*]:
assumes ‹*equivalent-registers F G*› **and** ‹*equivalent-registers G H*›
shows ‹*equivalent-registers F H*›
⟨*proof*⟩

lemma *equivalent-registers-assoc*[*simp*]:
assumes [*simp*]: ‹*compatible F G*› ‹*compatible F H*› ‹*compatible G H*›
shows ‹*equivalent-registers* (*F*;(*G*;*H*)) ((*F*;*G*);*H*)›
⟨*proof*⟩

lemma *equivalent-registers-pair-right*:
assumes [*simp*]: ‹*compatible F G*›
assumes *eq*: ‹*equivalent-registers G H*›
shows ‹*equivalent-registers* (*F*;*G*) (*F*;*H*)›
⟨*proof*⟩

lemma *equivalent-registers-pair-left*:
assumes [*simp*]: ‹*compatible F G*›
assumes *eq*: ‹*equivalent-registers F H*›
shows ‹*equivalent-registers* (*F*;*G*) (*H*;*G*)›
⟨*proof*⟩

lemma *equivalent-registers-comp*:
assumes ‹*register H*›
assumes ‹*equivalent-registers F G*›
shows ‹*equivalent-registers* (*H o F*) (*H o G*)›
⟨*proof*⟩

lemma *equivalent-registers-compatible1*:
assumes ‹*compatible F G*›
assumes ‹*equivalent-registers F F'*›
shows ‹*compatible F' G*›
⟨*proof*⟩

lemma *equivalent-registers-compatible2*:
assumes ‹*compatible F G*›
assumes ‹*equivalent-registers G G'*›

shows ‹*compatible F G′*›
⟨*proof*⟩

10.10 Compatibility simplification

The simproc *compatibility-warn* produces helpful warnings for subgoals of the form *compatible x y* that are probably unsolvable due to missing declarations of variable compatibility facts. Same for subgoals of the form *register x*.

⟨*ML*⟩

named-theorems *register-attribute-rule-immediate*
named-theorems *register-attribute-rule*

lemmas [*register-attribute-rule*] = *conjunct1 conjunct2 iso-register-is-register iso-register-is-register*[*OF iso-register-inv*]
lemmas [*register-attribute-rule-immediate*] = *compatible-sym compatible-register1 compatible-register2*
asm-rl[*of* ‹*compatible - -*›] *asm-rl*[*of* ‹*iso-register -*›] *asm-rl*[*of* ‹*register -*›] *iso-register-inv*

The following declares an attribute [*register*]. When the attribute is applied to a fact of the form *register F*, *iso-register F*, *compatible F G* or a conjunction of these, then those facts are added to the simplifier together with some derived theorems (e.g., *compatible F G* also adds *register F*).
In theory *Laws-Complement*, support for *is-unit-register F* and *complements F G* is added to this attribute.

⟨*ML*⟩

10.11 Notation

no-notation *cblinfun-compose* (**infixl** $*_u$ *55*)
no-notation *tensor-op* (**infixr** $\otimes_u$ *70*)

bundle *register-syntax* **begin**
notation *register-tensor* (**infixr** $\otimes_r$ *70*)
notation *register-pair* ('(-;-'))
end

end

11 Quantum mechanics basics

theory *Quantum*
imports
Misc
Hilbert-Space-Tensor-Product.Hilbert-Space-Tensor-Product
HOL−Library.Z2
Jordan-Normal-Form.Matrix-Impl
Real-Impl.Real-Impl
HOL−Library.Code-Target-Numeral
begin

unbundle *cblinfun-syntax*

type-synonym ('*a*,'*b*) *matrix* = ‹('*a ell2*, '*b ell2*) *cblinfun*›

11.1 Basic quantum states

11.1.1 EPR pair

definition *vector-β00* = *vec-of-list* [*1/sqrt 2::complex*, *0*, *0*, *1/sqrt 2*]
definition *β00* :: ‹(*bit*×*bit*) *ell2*› **where** [*code del*]: *β00* = *basis-enum-of-vec vector-β00*
lemma *vec-of-basis-enum-β00*[*simp*]: *vec-of-basis-enum β00* = *vector-β00*
⟨*proof*⟩

lemma *vec-of-ell2-β00*[*simp*, *code*]: *vec-of-ell2 β00 = vector-β00*
  ⟨*proof*⟩

lemma *norm-β00*[*simp*]: *norm β00 = 1*
  ⟨*proof*⟩

11.1.2 Ket plus

definition *vector-ketplus = vec-of-list [1/sqrt 2::complex, 1/sqrt 2]*
definition *ketplus* :: ‹*bit ell2*› (|+⟩) **where** [*code del*]: ‹*ketplus = basis-enum-of-vec vector-ketplus*›
lemma *vec-of-basis-enum-ketplus*[*simp*]: *vec-of-basis-enum ketplus = vector-ketplus*
  ⟨*proof*⟩
lemma *vec-of-ell2-ketplus*[*simp*, *code*]: *vec-of-ell2 ketplus = vector-ketplus*
  ⟨*proof*⟩

11.2 Basic quantum gates

11.2.1 Pauli X

definition *matrix-pauliX = mat-of-rows-list 2 [[0::complex, 1], [1, 0]]*
definition *pauliX* :: ‹(*bit*, *bit*) *matrix*› **where** [*code del*]: *pauliX = cblinfun-of-mat matrix-pauliX*
lemma *mat-of-cblinfun-pauliX*[*simp*, *code*]: *mat-of-cblinfun pauliX = matrix-pauliX*
  ⟨*proof*⟩

derive (*eq*) *ceq bit*

instantiation *bit* :: *ccompare* **begin**
definition *CCOMPARE*(*bit*) = *Some* (*λb1 b2. case* (*b1*, *b2*) *of* (*0*, *0*) ⇒ *order.Eq* | (*0*, *1*) ⇒ *order.Lt* | (*1*, *0*) ⇒ *order.Gt* | (*1*, *1*) ⇒ *order.Eq*)
instance
  ⟨*proof*⟩
end

derive (*dlist*) *set-impl bit*

lemma *pauliX-adjoint*[*simp*]: *pauliX∗ = pauliX*
  ⟨*proof*⟩
lemma *pauliXX*[*simp*]: *pauliX* o_{CL} *pauliX = id-cblinfun*
  ⟨*proof*⟩

11.2.2 Pauli Z

definition *matrix-pauliZ = mat-of-rows-list 2 [[1::complex, 0], [0, −1]]*
definition *pauliZ* :: ‹(*bit*, *bit*) *matrix*› **where** [*code del*]: *pauliZ = cblinfun-of-mat matrix-pauliZ*
lemma *mat-of-cblinfun-pauliZ*[*simp*, *code*]: *mat-of-cblinfun pauliZ = matrix-pauliZ*
  ⟨*proof*⟩
lemma *pauliZ-adjoint*[*simp*]: *pauliZ∗ = pauliZ*
  ⟨*proof*⟩
lemma *pauliZZ*[*simp*]: *pauliZ* o_{CL} *pauliZ = id-cblinfun*
  ⟨*proof*⟩

11.2.3 Hadamard

definition *matrix-hadamard = mat-of-rows-list 2 [[1/sqrt 2::complex, 1/sqrt 2], [1/sqrt 2, −1/sqrt 2]]*
definition *hadamard* :: ‹(*bit*,*bit*) *matrix*› **where** [*code del*]: *hadamard = cblinfun-of-mat matrix-hadamard*

lemma *mat-of-cblinfun-hadamard*[*simp*, *code*]: *mat-of-cblinfun hadamard = matrix-hadamard*
  ⟨*proof*⟩

lemma *hada-adj*[*simp*]: *hadamard∗ = hadamard*
  ⟨*proof*⟩

11.2.4 CNOT

definition *matrix-CNOT = mat-of-rows-list 4 [[1::complex,0,0,0], [0,1,0,0], [0,0,0,1], [0,0,1,0]]*

definition *CNOT* :: ‹(*bit*∗*bit*, *bit*∗*bit*) *matrix*› **where** [*code del*]: *CNOT* = *cblinfun-of-mat matrix-CNOT*

lemma *mat-of-cblinfun-CNOT*[*simp*, *code*]: *mat-of-cblinfun CNOT* = *matrix-CNOT*
⟨*proof*⟩

lemma *CNOT-adj*[*simp*]: *CNOT*∗ = *CNOT*
⟨*proof*⟩

lemma *cnot-apply*[*simp*]: ‹*CNOT* $*_V$ *ket* (*i*,*j*) = *ket* (*i*,*j*+*i*)›
⟨*proof*⟩

11.2.5 Qubit swap

definition *matrix-Uswap* = *mat-of-rows-list 4* [[*1*::*complex*, *0*, *0*, *0*], [*0*,*0*,*1*,*0*], [*0*,*1*,*0*,*0*], [*0*,*0*,*0*,*1*]]
definition *Uswap* :: ‹(*bit*×*bit*, *bit*×*bit*) *matrix*› **where**
[*code del*]: ‹*Uswap* = *cblinfun-of-mat matrix-Uswap*›

lemma *mat-of-cblinfun-Uswap*[*simp*, *code*]: *mat-of-cblinfun Uswap* = *matrix-Uswap*
⟨*proof*⟩

lemma *dim-col-Uswap*[*simp*]: *dim-col matrix-Uswap* = *4*
⟨*proof*⟩
lemma *dim-row-Uswap*[*simp*]: *dim-row matrix-Uswap* = *4*
⟨*proof*⟩
lemma *Uswap-adjoint*[*simp*]: *Uswap*∗ = *Uswap*
⟨*proof*⟩
lemma *Uswap-involution*[*simp*]: *Uswap* o_{CL} *Uswap* = *id-cblinfun*
⟨*proof*⟩
lemma *unitary-Uswap*[*simp*]: *unitary Uswap*
⟨*proof*⟩

lemma *Uswap-apply*[*simp*]: ‹*Uswap* $*_V$ *s* $\otimes_s$ *t* = *t* $\otimes_s$ *s*›
⟨*proof*⟩

unbundle *no cblinfun-syntax*

end

12 Derived facts about quantum registers

theory *Quantum-Extra*
imports
Laws-Quantum
Quantum
begin

no-notation *meet* (**infixl** ⊓ı *70*)
no-notation *Group.mult* (**infixl** ⊗ı *70*)
no-notation *Order.top* (⊤ı)
unbundle *lattice-syntax*
unbundle *register-syntax*
unbundle *cblinfun-syntax*

lemma *zero-not-register*[*simp*]: ‹~ *register* (λ-. *0*)›
⟨*proof*⟩

lemma *register-pair-is-register-converse*:
‹*register* (*F*;*G*) ⟹ *register F*› ‹*register* (*F*;*G*) ⟹ *register G*›
⟨*proof*⟩

lemma *register-id'*[*simp*]: ‹*register* (λ*x*. *x*)›
⟨*proof*⟩

lemma *compatible-proj-intersect*:

assumes *compatible R S* **and** *is-Proj a* **and** *is-Proj b*
shows $(R\ a *_S \top) \sqcap (S\ b *_S \top) = ((R\ a\ o_{CL}\ S\ b) *_S \top)$
⟨*proof*⟩

lemma *compatible-proj-mult*:
assumes *compatible R S* **and** *is-Proj a* **and** *is-Proj b*
shows *is-Proj* $(R\ a\ o_{CL}\ S\ b)$
⟨*proof*⟩

lemma *sandwich-tensor*:
fixes a :: ‹$'a$ *ell2* $\Rightarrow_{CL}$ $'a$ *ell2*› **and** b :: ‹$'b$ *ell2* $\Rightarrow_{CL}$ $'b$ *ell2*›
assumes [*simp*]: ‹*unitary a*› ‹*unitary b*›
shows $(*_V)$ (*sandwich* $(a \otimes_o b)$) = *sandwich* $a \otimes_r$ *sandwich* b
⟨*proof*⟩

lemma *sandwich-grow-left*:
fixes a :: ‹$'a$ *ell2* $\Rightarrow_{CL}$ $'a$ *ell2*›
assumes *unitary a*
shows *sandwich* $a \otimes_r$ *id* = *sandwich* $(a \otimes_o$ *id-cblinfun*$)$
⟨*proof*⟩

lemma *register-sandwich*: ‹*register F* $\Longrightarrow$ *F* (*sandwich a b*) = *sandwich* (*F a*) (*F b*)›
⟨*proof*⟩

lemma *assoc-ell2-sandwich*: ‹*assoc* = *sandwich assoc-ell2*›
⟨*proof*⟩

lemma *assoc-ell2'-sandwich*: ‹*assoc'* = *sandwich* (*assoc-ell2*∗)›
⟨*proof*⟩

lemma *swap-sandwich*: *swap* = *sandwich Uswap*
⟨*proof*⟩

lemma *id-tensor-sandwich*:
fixes a :: $'a$::*finite ell2* $\Rightarrow_{CL}$ $'b$::*finite ell2*
assumes *unitary a*
shows *id* $\otimes_r$ *sandwich a* = *sandwich* (*id-cblinfun* $\otimes_o$ *a*)
⟨*proof*⟩

lemma *compatible-selfbutter-join*:
assumes [*register*]: *compatible R S*
shows R (*selfbutter* ψ) o_{CL} S (*selfbutter* φ) = $(R;\ S)$ (*selfbutter* $(\psi \otimes_s \varphi)$)
⟨*proof*⟩

lemma *register-mult'*:
assumes ‹*register F*›
shows ‹$F\ a *_V F\ b *_V c = F\ (a\ o_{CL}\ b) *_V c$›
⟨*proof*⟩

lemma *register-scaleC*:
assumes ‹*register F*› **shows** ‹$F\ (c *_C a) = c *_C F\ a$›
⟨*proof*⟩

lemma *register-adjoint*: $F\ (a*) = (F\ a)*$ **if** ‹*register F*›
⟨*proof*⟩

lemma *register-finite-dim*: ‹*register F* $\longleftrightarrow$ *clinear F* $\wedge$ *F id-cblinfun* = *id-cblinfun* $\wedge$ $(\forall\ a\ b.\ F\ (a\ o_{CL}\ b) = F\ a\ o_{CL}\ F\ b) \wedge (\forall\ a.\ F\ (a*) = F\ a*)$›
for F :: ‹$'a$::*finite update* $\Rightarrow$ $'b$::*finite update*›

⟨*proof*⟩

unbundle *no lattice-syntax*
unbundle *no register-syntax*
unbundle *no cblinfun-syntax*

end

13 Very simple Quantum Hoare logic

theory *QHoare*
  imports *Quantum-Extra*
begin

unbundle *register-syntax*
unbundle *cblinfun-syntax*
unbundle *lattice-syntax*
no-notation *Order.top* ($\top_1$)

locale *qhoare* =
  fixes *memory-type* :: *'mem itself*
begin

definition *apply U R* = *R U* **for** *R* :: ‹*'a update* ⇒ *'mem update*›
definition *ifthen R x* = *R* (*butterfly* (*ket x*) (*ket x*)) **for** *R* :: ‹*'a update* ⇒ *'mem update*›
definition *program S* = *fold* (o_{CL}) *S id-cblinfun* **for** *S* :: ‹*'mem update list*›

definition *hoare* :: ‹*'mem ell2 ccsubspace* ⇒ (*'mem ell2* $\Rightarrow_{CL}$ *'mem ell2*) *list* ⇒ *'mem ell2 ccsubspace* ⇒ *bool*›
where
  hoare C p D ⟷ (∀ *ψ*∈*space-as-set C*. *program p* $*_V$ *ψ* ∈ *space-as-set D*) **for** *C p D*

definition *EQ* :: (*'a update* ⇒ *'mem update*) ⇒ *'a ell2* ⇒ *'mem ell2 ccsubspace* (**infix** $=_q$ *75*) **where**
  EQ R ψ = *R* (*selfbutter ψ*) $*_S$ ⊤

lemma *program-skip*[*simp*]: *program* [] = *id-cblinfun*
  ⟨*proof*⟩

lemma *program-seq*: *program* (*p1*@*p2*) = *program p2* o_{CL} *program p1*
  ⟨*proof*⟩

lemma *hoare-seq*[*trans*]: *hoare C p1 D* ⟹ *hoare D p2 E* ⟹ *hoare C* (*p1*@*p2*) *E*
  ⟨*proof*⟩

lemma *hoare-weaken-left*[*trans*]: ‹*A* ≤ *B* ⟹ *hoare B p C* ⟹ *hoare A p C*›
  ⟨*proof*⟩

lemma *hoare-weaken-right*[*trans*]: ‹*hoare A p B* ⟹ *B* ≤ *C* ⟹ *hoare A p C*›
  ⟨*proof*⟩

lemma *hoare-skip*: *C* ≤ *D* ⟹ *hoare C* [] *D*
  ⟨*proof*⟩

lemma *hoare-apply*:
  assumes *R U* $*_S$ *pre* ≤ *post*
  shows *hoare pre* [*apply U R*] *post*
⟨*proof*⟩

lemma *hoare-ifthen*:
  fixes *R* :: ‹*'a update* ⇒ *'mem update*›
  assumes *R* (*selfbutter* (*ket x*)) $*_S$ *pre* ≤ *post*
  shows *hoare pre* [*ifthen R x*] *post*
⟨*proof*⟩
end

```
unbundle no register-syntax
unbundle no cblinfun-syntax
unbundle no lattice-syntax

end
```

14 Quantum teleportation

```
theory Teleport
  imports
    Real-Impl.Real-Impl
    HOL−Library.Code-Target-Numeral
    HOL−Library.Word
    Hilbert-Space-Tensor-Product.Tensor-Product-Code
    QHoare
begin

hide-const (open) Finite-Cartesian-Product.vec
hide-type (open) Finite-Cartesian-Product.vec
hide-const (open) Finite-Cartesian-Product.mat
hide-const (open) Finite-Cartesian-Product.row
hide-const (open) Finite-Cartesian-Product.column
no-notation Group.mult (infixl ⊗ı 70)
no-notation Order.top (⊤ı)
unbundle no vec-syntax
unbundle no inner-syntax
unbundle register-syntax
unbundle cblinfun-syntax

locale teleport-locale = qhoare TYPE('mem) +
  fixes X :: bit update ⇒ 'mem update
    and Φ :: (bit∗bit) update ⇒ 'mem update
    and A :: 'atype update ⇒ 'mem update
    and B :: 'btype update ⇒ 'mem update
  assumes compat[register]: mutually compatible (X,Φ,A,B)
begin

abbreviation Φ1 ≡ Φ ∘ Fst
abbreviation Φ2 ≡ Φ ∘ Snd
abbreviation XΦ2 ≡ (X;Φ2)
abbreviation XΦ1 ≡ (X;Φ1)
abbreviation XΦ ≡ (X;Φ)
abbreviation XAB ≡ ((X;A); B)
abbreviation AB ≡ (A;B)
abbreviation Φ2AB ≡ ((Φ o Snd; A); B)

definition teleport a b = [
    apply CNOT XΦ1,
    apply hadamard X,
    ifthen Φ1 a,
    ifthen X b,
    apply (if a=1 then pauliX else id-cblinfun) Φ2,
    apply (if b=1 then pauliZ else id-cblinfun) Φ2
  ]


lemma Φ-XΦ: ‹Φ a = XΦ (id-cblinfun ⊗o a)›
  ⟨proof⟩
lemma XΦ1-XΦ: ‹XΦ1 a = XΦ (assoc (a ⊗o id-cblinfun))›
  ⟨proof⟩
lemma XΦ2-XΦ: ‹XΦ2 a = XΦ ((id ⊗r swap) (assoc (a ⊗o id-cblinfun)))›
  ⟨proof⟩
```

lemma *Φ2-XΦ*: ‹*Φ2 a* = *XΦ* (*id-cblinfun* $\otimes_o$ (*id-cblinfun* $\otimes_o$ *a*))›
  ⟨*proof*⟩
lemma *X-XΦ*: ‹*X a* = *XΦ* (*a* $\otimes_o$ *id-cblinfun*)›
  ⟨*proof*⟩
lemma *Φ1-XΦ*: ‹*Φ1 a* = *XΦ* (*id-cblinfun* $\otimes_o$ (*a* $\otimes_o$ *id-cblinfun*))›
  ⟨*proof*⟩
lemmas *to-XΦ* = *Φ-XΦ XΦ1-XΦ XΦ2-XΦ Φ2-XΦ X-XΦ Φ1-XΦ*

lemma *X-XΦ1*: ‹*X a* = *XΦ1* (*a* $\otimes_o$ *id-cblinfun*)›
  ⟨*proof*⟩
lemmas *to-XΦ1* = *X-XΦ1*

lemma *XAB-to-XΦ2-AB*: ‹*XAB a* = (*XΦ2*;*AB*) ((*swap* $\otimes_r$ *id*) (*assoc′* (*id-cblinfun* $\otimes_o$ *assoc a*)))›
  ⟨*proof*⟩

lemma *XΦ2-to-XΦ2-AB*: ‹*XΦ2 a* = (*XΦ2*;*AB*) (*a* $\otimes_o$ *id-cblinfun*)›
  ⟨*proof*⟩

schematic-goal *Φ2AB-to-XΦ2-AB*: *Φ2AB a* = (*XΦ2*;*AB*) *?b*
  ⟨*proof*⟩

lemmas *to-XΦ2-AB* = *XAB-to-XΦ2-AB XΦ2-to-XΦ2-AB Φ2AB-to-XΦ2-AB*

lemma *teleport*:
  assumes [*simp*]: *norm ψ* = *1*
  shows *hoare* (*XAB* $=_q$ *ψ* ⊓ *Φ* $=_q$ *β00*) (*teleport a b*) (*Φ2AB* $=_q$ *ψ*)
⟨*proof*⟩

end

locale *concrete-teleport-vars* **begin**

type-synonym *a-state* = *64 word*
type-synonym *b-state* = *1000000 word*
type-synonym *mem* = *a-state* ∗ *bit* ∗ *bit* ∗ *b-state* ∗ *bit*
type-synonym *′a var* = ‹*′a update* ⇒ *mem update*›

definition *A* :: *a-state var* **where** ‹*A a* = *a* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun*›
definition *X* :: ‹*bit var*› **where** ‹*X a* = *id-cblinfun* $\otimes_o$ *a* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun*›
definition *Φ1* :: ‹*bit var*› **where** ‹*Φ1 a* = *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *a* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun*›
definition *B* :: ‹*b-state var*› **where** ‹*B a* = *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *a* $\otimes_o$ *id-cblinfun*›
definition *Φ2* :: ‹*bit var*› **where** ‹*Φ2 a* = *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *id-cblinfun* $\otimes_o$ *a*›

end

interpretation *teleport-concrete*:
  concrete-teleport-vars +
  teleport-locale concrete-teleport-vars.X
                  ‹(*concrete-teleport-vars.Φ1*; *concrete-teleport-vars.Φ2*)›
                  concrete-teleport-vars.A
                  concrete-teleport-vars.B
  ⟨*proof*⟩

thm *teleport*
thm *teleport-def*

unbundle *no register-syntax*
unbundle *no cblinfun-syntax*

end

15 Quantum instantiation of complements

theory *Axioms-Complement-Quantum*
  imports
    Laws-Quantum
    Quantum-Extra
    Hilbert-Space-Tensor-Product.Weak-Star-Topology
    Hilbert-Space-Tensor-Product.Partial-Trace
    With-Type.With-Type
    Hilbert-Space-Tensor-Product.Misc-Tensor-Product-TTS
begin

unbundle *no m-inv-syntax*
unbundle *cblinfun-syntax*
unbundle *lattice-syntax*
unbundle *register-syntax*
no-notation *Lattice.join* (**infixl** ⊔ı *65*)
no-notation *elt-set-eq* (**infix** *=o 50*)
no-notation *eq-closure-of* (*closure′-of*ı)
hide-const (**open**) *Order.top*
declare [[*eta-contract*]]

lemma *register-decomposition-converse*:
  assumes ‹*unitary U*›
  shows ‹*register* (*λx. sandwich U* (*id-cblinfun* $\otimes_o$ *x*))›
  ⟨*proof*⟩

lemma *iso-register-decomposition*:

assumes [*simp*]: ‹*iso-register F*›
shows ‹$\exists$ *U*. *unitary U* $\wedge$ *F* = *sandwich U*›
⟨*proof*⟩

lemma *complement-exists*:
fixes *F* :: ‹*'a update* $\Rightarrow$ *'b update*›
assumes ‹*register F*›
shows ‹*let 'c::type = register-decomposition-basis F in*
$\exists$ *G* :: *'c update* $\Rightarrow$ *'b update. compatible F G* $\wedge$ *iso-register* (*F*;*G*)›
⟨*proof*⟩

lemma *commutant-exchange*:
fixes *F* :: ‹*'a update* $\Rightarrow$ *'b update*›
assumes ‹*iso-register F*›
shows ‹*commutant* (*F* ‘ *X*) = *F* ‘ *commutant X*›
⟨*proof*⟩

lemma *complement-range*:
assumes [*simp*]: ‹*compatible F G*› **and** [*simp*]: ‹*iso-register* (*F*;*G*)›
shows ‹*range G* = *commutant* (*range F*)›
⟨*proof*⟩

lemma *register-inv-G-o-F*:
assumes [*simp*]: ‹*register F*› **and** [*simp*]: ‹*register G*› **and** *range-FG*: ‹*range F* $\subseteq$ *range G*›
shows ‹*register* (*inv G* $\circ$ *F*)›
⟨*proof*⟩

lemma *same-range-equivalent*:
fixes *F* :: ‹*'a update* $\Rightarrow$ *'c update*› **and** *G* :: ‹*'b update* $\Rightarrow$ *'c update*›
assumes [*simp*]: ‹*register F*› **and** [*simp*]: ‹*register G*›
assumes *range-FG*: ‹*range F* = *range G*›
shows ‹*equivalent-registers F G*›
⟨*proof*⟩

lemma *complement-unique*:
assumes *compatible F G* **and** ‹*iso-register* (*F*;*G*)›
assumes *compatible F H* **and** ‹*iso-register* (*F*;*H*)›
shows ‹*equivalent-registers G H*›
⟨*proof*⟩

15.1 Finite dimensional complement

typedef (*'a*, *'b::finite*) *complement-domain-simple* = ‹{..< *if CARD('b) div CARD('a)* $\neq$ *0 then CARD('b) div CARD('a) else 1*}›
⟨*proof*⟩

instance *complement-domain-simple* :: (*type*, *finite*) *finite*
⟨*proof*⟩

lemma *CARD-complement-domain*:
assumes ‹*CARD('b::finite)* = *n* $*$ *CARD('a)*›
shows ‹*CARD*((*'a*,*'b*) *complement-domain-simple*) = *n*›
⟨*proof*⟩

lemma *register-decomposition-finite-aux*:
fixes Φ :: ‹*'a::finite update* $\Rightarrow$ *'b::finite update*›
assumes [*simp*]: ‹*register* Φ›

shows ‹∃ *U* :: (*'a* × (*'a*, *'b*) *complement-domain-simple*) *ell2* $\Rightarrow_{CL}$ *'b ell2*. *unitary U* ∧
 (∀ *ϑ*. Φ *ϑ* = *sandwich U* (*ϑ* $\otimes_o$ *id-cblinfun*))›
— Proof based on [Daw21]
⟨*proof*⟩

lemma *register-decomposition-finite*:
 fixes Φ :: ‹*'a update* ⇒ *'b*::*finite update*›
 assumes [*simp*]: ‹*register* Φ›
 shows ‹∃ *U* :: (*'a* × (*'a*, *'b*) *complement-domain-simple*) *ell2* $\Rightarrow_{CL}$ *'b ell2*. *unitary U* ∧
 (∀ *ϑ*. Φ *ϑ* = *sandwich U* (*ϑ* $\otimes_o$ *id-cblinfun*))›
⟨*proof*⟩

hide-fact *register-decomposition-finite-aux*

lemma *complement-exists-simple*:
 fixes *F* :: ‹*'a update* ⇒ *'b*::*finite update*›
 assumes ‹*register F*›
 shows ‹∃ *G* :: (*'a*, *'b*) *complement-domain-simple update* ⇒ *'b update*. *compatible F G* ∧ *iso-register* (*F*;*G*)›
⟨*proof*⟩

unbundle *no cblinfun-syntax*
unbundle *no lattice-syntax*
unbundle *no register-syntax*

end

16 Generic laws about complements, instantiated quantumly

theory *Laws-Complement-Quantum*
 imports *Laws-Quantum Axioms-Complement-Quantum*
begin

unbundle *register-syntax*
notation *cblinfun-compose* (**infixl** $*_u$ *55*)
notation *tensor-op* (**infixr** $\otimes_u$ *70*)

definition ‹*complements F G* ⟷ *compatible F G* ∧ *iso-register* (*F*;*G*)›

lemma *complementsI*: ‹*compatible F G* ⟹ *iso-register* (*F*;*G*) ⟹ *complements F G*›
 ⟨*proof*⟩

lemma *complement-exists*:
 fixes *F* :: ‹*'a*::*type update* ⇒ *'b*::*type update*›
 assumes ‹*register F*›
 shows ‹*let 'c*::*type* = *register-decomposition-basis F in*
 ∃ *G* :: *'c update* ⇒ *'b update*. *complements F G*›
 ⟨*proof*⟩

lemma *complements-sym*: ‹*complements G F*› **if** ‹*complements F G*›
⟨*proof*⟩

definition *complement* :: ‹(*'a*::*type update* ⇒ *'b*::*finite update*) ⇒ ((*'a*,*'b*) *complement-domain-simple update* ⇒ *'b update*)› **where**
 ‹*complement F* = (*SOME G* :: (*'a*, *'b*) *complement-domain-simple update* ⇒ *'b update*. *compatible F G* ∧ *iso-register* (*F*;*G*))›

lemma *register-complement*[*simp*]: ‹*register* (*complement F*)› **if** ‹*register F*›
 ⟨*proof*⟩

lemma *complement-is-complement*[*simp*]:
 assumes ‹*register F*›
 shows ‹*complements F* (*complement F*)›
 ⟨*proof*⟩

lemma *complement-unique*:
  assumes ‹*complements* F G›
  assumes ‹*complements* F G'›
  shows ‹*equivalent-registers* G G'›
  ⟨*proof*⟩

lemma *complement-unique'*:
  assumes ‹*complements* F G›
  shows ‹*equivalent-registers* G (*complement* F)›
  ⟨*proof*⟩

lemma *compatible-complement*[*simp*]: ‹*register* $F \Longrightarrow$ *compatible* F (*complement* F)›
  ⟨*proof*⟩

lemma *complements-register-tensor*:
  assumes [*simp*]: ‹*register* F› ‹*register* G›
  shows ‹*complements* ($F \otimes_r G$) (*complement* $F \otimes_r$ *complement* G)›
⟨*proof*⟩

definition *is-unit-register* **where**
  ‹*is-unit-register* $U \longleftrightarrow$ *complements* U *id*›

lemma *register-unit-register*[*simp*]: ‹*is-unit-register* $U \Longrightarrow$ *register* U›
  ⟨*proof*⟩

lemma *unit-register-compatible*[*simp*]: ‹*compatible* U X› **if** ‹*is-unit-register* U› ‹*register* X›
  ⟨*proof*⟩

lemma *unit-register-compatible'*[*simp*]: ‹*compatible* X U› **if** ‹*is-unit-register* U› ‹*register* X›
  ⟨*proof*⟩

lemma *compatible-complement-left*[*simp*]: ‹*register* $X \Longrightarrow$ *compatible* (*complement* X) X›
  ⟨*proof*⟩

lemma *compatible-complement-right*[*simp*]: ‹*register* $X \Longrightarrow$ *compatible* X (*complement* X)›
  ⟨*proof*⟩

lemma *unit-register-pair*[*simp*]: ‹*equivalent-registers* X $(U;\ X)$› **if** [*simp*]: ‹*is-unit-register* U› ‹*register* X›
⟨*proof*⟩

lemma *unit-register-compose-left*:
  assumes [*simp*]: ‹*is-unit-register* U›
  assumes [*simp*]: ‹*register* A›
  shows ‹*is-unit-register* (A *o* U)›
⟨*proof*⟩

lemma *unit-register-compose-right*:
  assumes [*simp*]: ‹*is-unit-register* U›
  assumes [*simp*]: ‹*iso-register* A›
  shows ‹*is-unit-register* (U *o* A)›
⟨*proof*⟩

lemma *unit-register-unique*:
  assumes ‹*is-unit-register* F›
  assumes ‹*is-unit-register* G›
  shows ‹*equivalent-registers* F G›
⟨*proof*⟩

lemma *unit-register-domains-isomorphic*:
  fixes F :: ‹*'a::type update* $\Rightarrow$ *'c::type update*›
  fixes G :: ‹*'b::type update* $\Rightarrow$ *'d::type update*›

  assumes ‹*is-unit-register F*›
  assumes ‹*is-unit-register G*›
  shows ‹$\exists$ *I* :: *'a update* $\Rightarrow$ *'b update. iso-register I*›
⟨*proof*⟩

lemma *id-complement-is-unit-register*[*simp*]: ‹*is-unit-register* (*complement id*)›
  ⟨*proof*⟩

type-synonym *unit-register-domain* = ‹(*unit*, *unit*) *complement-domain-simple*›
definition *unit-register* :: ‹*unit-register-domain update* $\Rightarrow$ *'a*::*type update*› **where** ‹*unit-register* = (*SOME U*. *is-unit-register U*)›

lemma *unit-register-is-unit-register*[*simp*]: ‹*is-unit-register* (*unit-register* :: *unit-register-domain update* $\Rightarrow$ *'a*::*type update*)›
⟨*proof*⟩

lemma *unit-register-domain-tensor-unit*:
  fixes *U* :: ‹*'a*::*type update* $\Rightarrow$ -›
  assumes ‹*is-unit-register U*›
  shows ‹$\exists$ *I* :: *'b*::*type update* $\Rightarrow$ (*'a*$*$*'b*) *update. iso-register I*›

⟨*proof*⟩

lemma *compatible-complement-pair1*:
  assumes ‹*compatible F G*›
  shows ‹*compatible F* (*complement* (*F*;*G*))›
  ⟨*proof*⟩

lemma *compatible-complement-pair2*:
  assumes [*simp*]: ‹*compatible F G*›
  shows ‹*compatible G* (*complement* (*F*;*G*))›
⟨*proof*⟩

lemma *equivalent-complements*:
  assumes ‹*complements F G*›
  assumes ‹*equivalent-registers G G'*›
  shows ‹*complements F G'*›
  ⟨*proof*⟩

lemma *complements-complement-pair*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes *FG'*: ‹*complements* (*F*;*G*) *FG'*›
  shows ‹*complements F* (*G*; *FG'*)›
⟨*proof*⟩

lemma *equivalent-registers-complement*:
  assumes ‹*equivalent-registers F G*›
  assumes ‹*complements F F'*›
  assumes ‹*complements G G'*›
  shows ‹*equivalent-registers F' G'*›
  ⟨*proof*⟩

lemma *equivalent-registers-complement'*:
  assumes ‹*equivalent-registers F G*›
  shows ‹*equivalent-registers* (*complement F*) (*complement G*)›
  ⟨*proof*⟩

lemma *complements-complement-pair'*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes *FG'*: ‹*complements* (*F*;*G*) *FG'*›
  shows ‹*complements G* (*F*; *FG'*)›
⟨*proof*⟩

lemma *complements-chain*:
  assumes [*simp*]: ‹*register F*› ‹*register G*›
  shows ‹*complements* (*F o G*) (*complement F*; *F o complement G*)›
⟨*proof*⟩

lemma *complements-Fst-Snd*[*simp*]: ‹*complements Fst Snd*›
  ⟨*proof*⟩

lemma *complements-Snd-Fst*[*simp*]: ‹*complements Snd Fst*›
  ⟨*proof*⟩

lemma *compatible-unit-register*[*simp*]: ‹*register F* ⟹ *compatible F unit-register*›
  ⟨*proof*⟩

lemma *complements-id-unit-register*[*simp*]: ‹*complements id unit-register*›
  ⟨*proof*⟩

lemma *complements-iso-unit-register*: ‹*iso-register I* ⟹ *is-unit-register U* ⟹ *complements I U*›
  ⟨*proof*⟩

lemma *iso-register-complement-is-unit-register*[*simp*]:
  assumes ‹*iso-register F*›
  shows ‹*is-unit-register* (*complement F*)›
  ⟨*proof*⟩

Adding support for *is-unit-register F* and *complements F G* to the [*register*] attribute

lemmas [*register-attribute-rule*] = *is-unit-register-def*[*THEN iffD1*] *complements-def*[*THEN iffD1*]
lemmas [*register-attribute-rule-immediate*] = *asm-rl*[*of* ‹*is-unit-register -*›]

no-notation *cblinfun-compose* (**infixl** $*_u$ *55*)
no-notation *tensor-op* (**infixr** $\otimes_u$ *70*)
unbundle *no register-syntax*

end

17 More derived facts about quantum registers

This theory contains some derived facts that cannot be placed in theory *Quantum-Extra* because they depend on *Laws-Complement-Quantum*.

theory *Quantum-Extra2*
  imports
    Laws-Complement-Quantum
    Quantum
begin

unbundle *register-syntax*

definition *empty-var* :: ‹*'a*::{*CARD-1*,*enum*} *update* ⇒ *'b update*› **where**
  empty-var a = *one-dim-iso a* $*_C$ *id-cblinfun*

lemma *is-unit-register-empty-var*[*register*]: ‹*is-unit-register empty-var*› (**is** ‹*is-unit-register ?e*›)
⟨*proof*⟩

instance *complement-domain-simple* :: (*type*, *type*) *default* ⟨*proof*⟩

unbundle *no register-syntax*

end
theory *Pure-States*
  imports *Quantum-Extra2 HOL−Eisbach.Eisbach*

begin

unbundle *cblinfun-syntax*
unbundle *register-syntax*

definition ‹*pure-state-target-vector F* η_F = (*if ket default* ∈ *range* (*cblinfun-apply* (*F* (*butterfly* η_F η_F)))
  then ket default
  else (*SOME* η'. *norm* η' = *1* ∧ η' ∈ *range* (*cblinfun-apply* (*F* (*butterfly* η_F η_F)))))›

lemma *pure-state-target-vector-eqI*:
  assumes ‹*F* (*butterfly* η_F η_F) = *G* (*butterfly* η_G η_G)›
  shows ‹*pure-state-target-vector F* η_F = *pure-state-target-vector G* η_G›
  ⟨*proof*⟩

lemma *pure-state-target-vector-ket-default*: ‹*pure-state-target-vector F* η_F = *ket default*› **if** ‹*ket default* ∈ *range* (*cblinfun-apply* (*F* (*butterfly* η_F η_F)))›
  ⟨*proof*⟩

lemma
  assumes [*simp*]: ‹η_F ≠ *0*› ‹*register F*›
  shows *pure-state-target-vector-in-range*: ‹*pure-state-target-vector F* η_F ∈ *range* (($*_V$) (*F* (*selfbutter* η_F)))› (**is** *?range*)
    and *pure-state-target-vector-norm*: ‹*norm* (*pure-state-target-vector F* η_F) = *1*› (**is** *?norm*)
⟨*proof*⟩

lemma *pure-state-target-vector-correct*:
  assumes [*simp*]: ‹η ≠ *0*› ‹*register F*›
  shows ‹*F* (*selfbutter* η) $*_V$ *pure-state-target-vector F* η ≠ *0*›
⟨*proof*⟩

definition ‹*pure-state' F* ψ η_F = *F* (*butterfly* ψ η_F) $*_V$ *pure-state-target-vector F* η_F›

abbreviation ‹*pure-state F* ψ ≡ *pure-state' F* ψ (*ket default*)›

nonterminal *pure-tensor*
syntax *-pure-tensor* :: ‹*'a* ⇒ *'b* ⇒ *pure-tensor* ⇒ *pure-tensor*› (- - $\otimes_p$ - [*1000*, *0*, *0*] *1000*)
syntax *-pure-tensor2* :: ‹*'a* ⇒ *'b* ⇒ *'c* ⇒ *'d* ⇒ *pure-tensor*› (- - $\otimes_p$ - - [*1000*, *0*, *1000*, *0*] *1000*)
syntax *-pure-tensor1* :: ‹*'a* ⇒ *'b* ⇒ *pure-tensor*›
syntax *-pure-tensor-start* :: ‹*pure-tensor* ⇒ *'a*› ('(-'))

translations
  -pure-tensor2 F ψ *G* φ ⇀ *CONST pure-state* (*F*; *G*) (ψ $\otimes_s$ φ)
  -pure-tensor F ψ (*CONST pure-state G* φ) ⇀ *CONST pure-state* (*F*; *G*) (ψ $\otimes_s$ φ)
  -pure-tensor-start x ⇀ *x*

  -pure-tensor-start (*-pure-tensor2 F* ψ *G* φ) ↽ *CONST pure-state* (*F*; *G*) (ψ $\otimes_s$ φ)
  -pure-tensor F ψ (*-pure-tensor2 G* φ *H* η) ↽ *-pure-tensor2 F* ψ (*G*;*H*) (φ $\otimes_s$ η)

term ‹(*F* ψ $\otimes_p$ *G* φ $\otimes_p$ *H z*)›
term ‹*pure-state* (*F*; *G*) (*a* $\otimes_s$ *b*)›

lemma *register-pair-butterfly-tensor*: ‹(*F*; *G*) (*butterfly* (*a* $\otimes_s$ *b*) (*c* $\otimes_s$ *d*)) = *F* (*butterfly a c*) o_{CL} *G* (*butterfly b d*)›
  if [*simp*]: ‹*compatible F G*›
  ⟨*proof*⟩

lemma *pure-state-eqI*:
  assumes ‹*F* (*selfbutter* η_F) = *G* (*selfbutter* η_G)›
  assumes ‹*F* (*butterfly* ψ η_F) = *G* (*butterfly* φ η_G)›
  shows ‹*pure-state' F* ψ η_F = *pure-state' G* φ η_G›
⟨*proof*⟩

definition ‹*regular-register F* $\longleftrightarrow$ *register F* $\wedge$ ($\exists$ *a*. (*F*; *complement F*) (*selfbutter* (*ket default*) $\otimes_o$ *a*) = *selfbutter* (*ket default*))›

lemma *regular-registerI*:
  assumes [*simp*]: ‹*register F*›
  assumes [*simp*]: ‹*complements F G*›
  assumes *eq*: ‹(*F*; *G*) (*selfbutter* (*ket default*) $\otimes_o$ *a*) = *selfbutter* (*ket default*)›
  shows ‹*regular-register F*›
⟨*proof*⟩

lemma *regular-register-pair*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes ‹*regular-register F*› **and** ‹*regular-register G*›
  shows ‹*regular-register* (*F*;*G*)›
⟨*proof*⟩

lemma *regular-register-comp*: ‹*regular-register* (*F o G*)› **if** ‹*regular-register F*› ‹*regular-register G*›
⟨*proof*⟩

lemma *regular-iso-register*:
  assumes ‹*regular-register F*›
  assumes [*register*]: ‹*iso-register F*›
  shows ‹*F* (*selfbutter* (*ket default*)) = *selfbutter* (*ket default*)›
⟨*proof*⟩

lemma *pure-state-nested*:
  assumes [*simp*]: ‹*compatible F G*›
  assumes ‹*regular-register H*›
  assumes ‹*iso-register H*›
  shows ‹*pure-state* (*F*;*G*) (*pure-state H h* $\otimes_s$ *g*) = *pure-state* ((*F o H*);*G*) (*h* $\otimes_s$ *g*)›
⟨*proof*⟩

lemma *state-apply1*:
  assumes [*register*]: ‹*compatible F G*›
  shows ‹*F U* $*_V$ (*F* ψ $\otimes_p$ *G* φ) = (*F* (*U* ψ) $\otimes_p$ *G* φ)›
⟨*proof*⟩

lemma *Fst-regular*[*simp*]: ‹*regular-register Fst*›
  ⟨*proof*⟩

lemma *Snd-regular*[*simp*]: ‹*regular-register Snd*›
  ⟨*proof*⟩

lemma *id-regular*[*simp*]: ‹*regular-register id*›
  ⟨*proof*⟩

lemma *swap-regular*[*simp*]: ‹*regular-register swap*›
  ⟨*proof*⟩

lemma *assoc-regular*[*simp*]: ‹*regular-register assoc*›
  ⟨*proof*⟩

lemma *assoc'-regular*[*simp*]: ‹*regular-register assoc'*›
  ⟨*proof*⟩

lemma *cspan-pure-state'*:
  assumes ‹*iso-register F*›
  assumes ‹*cspan* (*g* ‘ *X*) = *UNIV*›
  assumes η*-cond*: ‹*F* (*selfbutter* η) $*_V$ *pure-state-target-vector F* $\eta \neq 0$›
  shows ‹*cspan* ((λ*z*. *pure-state' F* (*g z*) η) ‘ *X*) = *UNIV*›
⟨*proof*⟩

lemma *cspan-pure-state*:
  assumes [*simp*]: ‹*iso-register F*›
  assumes ‹*cspan* (*g* ‘ *X*) = *UNIV*›
  shows ‹*cspan* ((λ*z*. *pure-state F* (*g z*)) ‘ *X*) = *UNIV*›
  ⟨*proof*⟩

lemma *pure-state-bounded-clinear*:
  assumes [*register*]: ‹*compatible F G*›
  shows ‹*bounded-clinear* (λψ. (*F* ψ $\otimes_p$ *G* φ))›
⟨*proof*⟩

lemma *pure-state-bounded-clinear-right*:
  assumes [*register*]: ‹*compatible F G*›
  shows ‹*bounded-clinear* (λφ. (*F* ψ $\otimes_p$ *G* φ))›
⟨*proof*⟩

lemma *pure-state-clinear*:
  assumes [*register*]: ‹*compatible F G*›
  shows ‹*clinear* (λψ. (*F* ψ $\otimes_p$ *G* φ))›
  ⟨*proof*⟩

method *pure-state-flatten-nested* =
  (*subst pure-state-nested*, (*auto*; *fail*)[*3*])+

The following method *pure-state-eq* tries to solve a equality where both sides are of the form $F_1(\psi_1)$ $\otimes_p$ $F_2(\psi_2)$ $\otimes_p$ ... $\otimes_p$ $F_n(\psi_n)$ by reordering the registers and unfolding nested register pairs. (For the unfolding of nested pairs, it is necessary that the corresponding *compatible F G* facts are provable by the simplifier.)

If the some of the pure states ψ_i themselves are $\otimes_p$-tensors, they will be flattened if possible. (If all necessary conditions can be proven, such as *regular-register* etc.)

The method may either succeed, fail, or reduce the equality to a hopefully simpler one.

method *pure-state-eq* =
  (*pure-state-flatten-nested?*,
    rule pure-state-eqI;
    auto simp: *register-pair-butterfly-tensor compatible-ac-rules default-prod-def*
    simp flip: *tensor-ell2-ket*)

lemma *example*:
  fixes *F* :: ‹*bit update* ⇒ *'c*::{*finite*,*default*} *update*›
    and *G* :: ‹*bit update* ⇒ *'c update*›
  assumes [*register*]: ‹*compatible F G*›
  shows ‹(*F*;*G*) *CNOT* o_{CL} (*G*;*F*) *CNOT* o_{CL} (*F*;*G*) *CNOT* = (*F*;*G*) *swap-ell2*›
⟨*proof*⟩

unbundle *no cblinfun-syntax*
unbundle *no register-syntax*

end

theory *Check-Autogenerated-Files*

  imports *Laws-Classical Laws-Quantum Laws-Complement-Quantum*
begin

⟨*ML*⟩

end

References

[Daw21] Matthew Daws. Answer to mathoverlow question “unital *-homomorphisms between matrices”. https://mathoverflow.net/a/390180, 2021. Last accessed 2021-10-12.

[Unr24] Dominique Unruh. Quantum references. arXiv:2105.10914 [cs.LO], 2021–2024.