

Quantum and Classical Registers

Dominique Unruh

December 14, 2021

Abstract

A formalization of the theory of quantum and classical registers as developed by Unruh [Unr21]. 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 [Unr21]. 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 [Unr21]. [Unr21] 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 [Unr21] for an explanation of this mechanism and the reasons for it.

Contents

1	Axioms of registers	2
2	Generic laws about registers	3
2.1	Elementary facts	4
2.2	Preregisters	4
2.3	Registers	4
2.4	Tensor product of registers	4
2.5	Pairs and compatibility	5
2.6	Fst and Snd	7
2.7	Compatibility of register tensor products	8
2.8	Associativity of the tensor product	8
2.9	Iso-registers	9
2.10	Compatibility simplification	11
2.11	Notation	11
3	Axioms of complements	11
4	Generic laws about complements	12
5	Classical instantiation of registers	14
6	Generic laws about registers, instantiated classically	16
6.1	Elementary facts	17
6.2	Preregisters	17
6.3	Registers	17
6.4	Tensor product of registers	17
6.5	Pairs and compatibility	18
6.6	Fst and Snd	20

6.7	Compatibility of register tensor products	21
6.8	Associativity of the tensor product	21
6.9	Iso-registers	22
6.10	Compatibility simplification	24
6.11	Notation	24
7	Miscellaneous facts	24
8	Derived facts about classical registers	27
9	Tensor products (finite dimensional)	28
10	Quantum instantiation of registers	33
11	Generic laws about registers, instantiated quantumly	34
11.1	Elementary facts	34
11.2	Preregisters	34
11.3	Registers	35
11.4	Tensor product of registers	35
11.5	Pairs and compatibility	36
11.6	Fst and Snd	37
11.7	Compatibility of register tensor products	39
11.8	Associativity of the tensor product	39
11.9	Iso-registers	40
11.10	Compatibility simplification	41
11.11	Notation	42
12	Quantum mechanics basics	42
12.1	Basic quantum states	42
12.1.1	EPR pair	42
12.1.2	Ket plus	42
12.2	Basic quantum gates	43
12.2.1	Pauli X	43
12.2.2	Pauli Z	43
12.2.3	Hadamard	43
12.2.4	CNOT	43
12.2.5	Qubit swap	44
13	Derived facts about quantum registers	44
14	Very simple Quantum Hoare logic	46
15	Tensor products as matrices	46
16	Quantum teleportation	48
17	Quantum instantiation of complements	50
18	Generic laws about complements, instantiated quantumly	51
19	More derived facts about quantum registers	54

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 \Rightarrow 'a update \Rightarrow '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 \Rightarrow 'b::domain update> \Rightarrow bool>

axiomatization where

comp-preregister: preregister F \Longrightarrow preregister G \Longrightarrow preregister (G \circ F) **and**
id-preregister: <preregister id>

for F :: <'a::domain update \Rightarrow 'b::domain update> **and** G :: <'b update \Rightarrow 'c::domain update>

axiomatization where

preregister-mult-right: <preregister (λ a. comp-update a z)> **and**
preregister-mult-left: <preregister (λ a. comp-update z a)>
for z :: 'a::domain update

axiomatization tensor-update :: <'a::domain update \Rightarrow 'b::domain update \Rightarrow ('a \times 'b) update>

where tensor-extensionality: preregister F \Longrightarrow preregister G \Longrightarrow (\bigwedge a b. F (tensor-update a b) = G (tensor-update a b)) \Longrightarrow F = G

for F G :: <'a \times 'b) update \Rightarrow '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 \Rightarrow 'b update> \Rightarrow bool>

axiomatization where

register-preregister: register F \Longrightarrow preregister F **and**
register-comp: register F \Longrightarrow register G \Longrightarrow register (G \circ F) **and**
register-mult: register F \Longrightarrow comp-update (F a) (F b) = F (comp-update a b) **and**
register-of-id: <register F \Longrightarrow F id-update = id-update> **and**
register-id: <register (id :: 'a update \Rightarrow 'a update)>

for F :: 'a::domain update \Rightarrow 'b::domain update **and** G :: 'b update \Rightarrow 'c::domain update

axiomatization where register-tensor-left: <register (λ a. tensor-update a id-update)>

axiomatization where register-tensor-right: <register (λ a. tensor-update id-update a)>

axiomatization register-pair ::

<'a::domain update \Rightarrow 'c::domain update> \Rightarrow ('b::domain update \Rightarrow 'c update)
 \Rightarrow (('a \times 'b) update \Rightarrow 'c update)> **where**

register-pair-is-register: <register F \Longrightarrow register G \Longrightarrow (\bigwedge a b. comp-update (F a) (G b) = comp-update (G b) (F a))

\Longrightarrow register (register-pair F G)> **and**

register-pair-apply: <register F \Longrightarrow register G \Longrightarrow (\bigwedge a b. comp-update (F a) (G b) = comp-update (G b) (F a))
 \Longrightarrow (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 (λb::'b::domain update. tensor-update a b)⟩
  for a :: ⟨'a::domain update⟩
  ⟨proof⟩

```

```

lemma preregister-tensor-right[simp]: ⟨preregister (λa::'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 (λa::'a::domain update. id-update ⊗u F a)⟩
  ⟨proof⟩

```

```

lemma register-tensor-id-update[simp]:
  assumes ⟨register F⟩
  shows ⟨register (λa::'a::domain update. F a ⊗u id-update)⟩
  ⟨proof⟩

```

2.4 Tensor product of registers

```

definition register-tensor (infixr ⊗r 70) where
  register-tensor F G = register-pair (λa. tensor-update (F a) id-update) (λb. 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 ⊗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 ⊗r G) (a ⊗u b) = F a ⊗u G b
  ⟨proof⟩

```

```

definition separating (-::'b::domain itself) A ↔
  (∀ F G :: 'a::domain 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::domain) A ⇒ separating TYPE('a) B⟩
  ⟨proof⟩

```

```

lemma register-eqI: ⟨separating TYPE('b::domain) A ⇒ preregister F ⇒ preregister G ⇒ (∧x. x∈A ⇒ F
  x = G x) ⇒ F = (G::- ⇒ 'b update)⟩
  ⟨proof⟩

```

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

lemma register-tensor-distrib:
assumes $[simp]: \langle \text{register } F \rangle \langle \text{register } G \rangle \langle \text{register } H \rangle \langle \text{register } L \rangle$
shows $\langle (F \otimes_r G) o (H \otimes_r L) = (F o H) \otimes_r (G o L) \rangle$
 $\langle \text{proof} \rangle$

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

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

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

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

lemma register-tensor-id $[simp]: \langle \text{id} \otimes_r \text{id} = \text{id} \rangle$
 $\langle \text{proof} \rangle$

2.5 Pairs and compatibility

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

lemma compatibleI:
assumes $\text{register } F$ **and** $\text{register } G$
assumes $\langle \bigwedge a b. (F a) *_u (G b) = (G b) *_u (F a) \rangle$
shows $\text{compatible } F G$
 $\langle \text{proof} \rangle$

lemma swap-registers:
assumes $\text{compatible } R S$
shows $R a *_u S b = S b *_u R a$
 $\langle \text{proof} \rangle$

lemma compatible-sym: $\text{compatible } x y \implies \text{compatible } y x$
 $\langle \text{proof} \rangle$

lemma pair-is-register $[simp]:$
assumes $\text{compatible } F G$
shows $\text{register } (F; G)$
 $\langle \text{proof} \rangle$

lemma *register-pair-apply*:

assumes $\langle \text{compatible } F \ G \rangle$

shows $\langle (F; G) (a \otimes_u b) = (F a) *_u (G b) \rangle$

$\langle \text{proof} \rangle$

lemma *register-pair-apply'*:

assumes $\langle \text{compatible } F \ G \rangle$

shows $\langle (F; G) (a \otimes_u b) = (G b) *_u (F a) \rangle$

$\langle \text{proof} \rangle$

lemma *compatible-comp-left[simp]*: $\text{compatible } F \ G \implies \text{register } H \implies \text{compatible } (F \circ H) \ G$

$\langle \text{proof} \rangle$

lemma *compatible-comp-right[simp]*: $\text{compatible } F \ G \implies \text{register } H \implies \text{compatible } F \ (G \circ H)$

$\langle \text{proof} \rangle$

lemma *compatible-comp-inner[simp]*:

$\text{compatible } F \ G \implies \text{register } H \implies \text{compatible } (H \circ F) \ (H \circ G)$

$\langle \text{proof} \rangle$

lemma *compatible-register1*: $\langle \text{compatible } F \ G \implies \text{register } F \rangle$

$\langle \text{proof} \rangle$

lemma *compatible-register2*: $\langle \text{compatible } F \ G \implies \text{register } G \rangle$

$\langle \text{proof} \rangle$

lemma *pair-o-tensor*:

assumes $\text{compatible } A \ B$ **and** $[\text{simp}]$: $\langle \text{register } C \rangle$ **and** $[\text{simp}]$: $\langle \text{register } D \rangle$

shows $(A; B) \circ (C \otimes_r D) = (A \circ C; B \circ D)$

$\langle \text{proof} \rangle$

lemma *compatible-tensor-id-update-left[simp]*:

fixes $F :: 'a::\text{domain update} \Rightarrow 'c::\text{domain update}$ **and** $G :: 'b::\text{domain update} \Rightarrow 'c::\text{domain update}$

assumes $\text{compatible } F \ G$

shows $\text{compatible } (\lambda a. \text{id-update } \otimes_u F a) (\lambda a. \text{id-update } \otimes_u G a)$

$\langle \text{proof} \rangle$

lemma *compatible-tensor-id-update-right[simp]*:

fixes $F :: 'a::\text{domain update} \Rightarrow 'c::\text{domain update}$ **and** $G :: 'b::\text{domain update} \Rightarrow 'c::\text{domain update}$

assumes $\text{compatible } F \ G$

shows $\text{compatible } (\lambda a. F a \otimes_u \text{id-update}) (\lambda a. G a \otimes_u \text{id-update})$

$\langle \text{proof} \rangle$

lemma *compatible-tensor-id-update-rl[simp]*:

assumes $\text{register } F$ **and** $\text{register } G$

shows $\text{compatible } (\lambda a. F a \otimes_u \text{id-update}) (\lambda a. \text{id-update } \otimes_u G a)$

$\langle \text{proof} \rangle$

lemma *compatible-tensor-id-update-lr[simp]*:

assumes $\text{register } F$ **and** $\text{register } G$

shows $\text{compatible } (\lambda a. \text{id-update } \otimes_u F a) (\lambda a. G a \otimes_u \text{id-update})$

$\langle \text{proof} \rangle$

lemma *register-comp-pair*:

assumes $[\text{simp}]$: $\langle \text{register } F \rangle$ **and** $[\text{simp}]$: $\langle \text{compatible } G \ H \rangle$

shows $(F \circ G; F \circ H) = F \circ (G; H)$

$\langle \text{proof} \rangle$

lemma *swap-registers-left*:

assumes $\text{compatible } R \ S$

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** $\langle Fst\ a = a \otimes_u id\text{-update} \rangle$

definition *Snd* **where** $\langle Snd\ a = id\text{-update} \otimes_u a \rangle$

lemma *register-Fst[simp]*: $\langle register\ Fst \rangle$

⟨proof⟩

lemma *register-Snd[simp]*: $\langle register\ Snd \rangle$

⟨proof⟩

lemma *compatible-Fst-Snd[simp]*: $\langle compatible\ Fst\ Snd \rangle$

⟨proof⟩

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

definition $\langle swap = (Snd; Fst) \rangle$

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]*: $\langle register\ swap \rangle$

⟨proof⟩

lemma *pair-Fst-Snd*: $\langle (Fst; Snd) = id \rangle$

⟨proof⟩

lemma *swap-o-swap[simp]*: $\langle swap\ o\ swap = id \rangle$

⟨proof⟩

lemma *swap-swap[simp]*: $\langle swap\ (swap\ x) = x \rangle$

⟨proof⟩

lemma *inv-swap[simp]*: $\langle inv\ swap = swap \rangle$

⟨proof⟩

lemma *register-pair-Fst*:

assumes $\langle compatible\ F\ G \rangle$

shows $\langle (F;G)\ o\ Fst = F \rangle$

⟨proof⟩

lemma *register-pair-Snd*:

assumes $\langle compatible\ F\ G \rangle$

shows $\langle (F;G)\ o\ Snd = G \rangle$

⟨proof⟩

lemma *register-Fst-register-Snd[simp]*:

assumes $\langle register\ F \rangle$

shows $\langle (F\ o\ Fst; F\ o\ Snd) = F \rangle$

$\langle proof \rangle$

lemma *register-Snd-register-Fst*[simp]:
 assumes $\langle register\ F \rangle$
 shows $\langle (F\ o\ Snd; F\ o\ Fst) = F\ o\ swap \rangle$
 $\langle proof \rangle$

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

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

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

2.7 Compatibility of register tensor products

lemma *compatible-register-tensor*:
 fixes $F :: \langle 'a::domain\ update \Rightarrow 'e::domain\ update \rangle$ **and** $G :: \langle 'b::domain\ update \Rightarrow 'f::domain\ update \rangle$
 and $F' :: \langle 'c::domain\ update \Rightarrow 'e\ update \rangle$ **and** $G' :: \langle 'd::domain\ update \Rightarrow 'f\ update \rangle$
 assumes [simp]: $\langle compatible\ F\ F' \rangle$
 assumes [simp]: $\langle compatible\ G\ G' \rangle$
 shows $\langle compatible\ (F\ \otimes_r\ G)\ (F'\ \otimes_r\ G') \rangle$
 $\langle proof \rangle$

2.8 Associativity of the tensor product

definition *assoc* :: $\langle ('a::domain \times 'b::domain) \times 'c::domain \rangle\ update \Rightarrow ('a \times ('b \times 'c))\ update$ **where**
 $\langle assoc = ((Fst; Snd\ o\ Fst); Snd\ o\ Snd) \rangle$

lemma *assoc-is-hom*[simp]: $\langle preregister\ assoc \rangle$
 $\langle proof \rangle$

lemma *assoc-apply*[simp]: $\langle assoc\ ((a\ \otimes_u\ b)\ \otimes_u\ c) = (a\ \otimes_u\ (b\ \otimes_u\ c)) \rangle$
 $\langle proof \rangle$

definition *assoc'* :: $\langle ('a \times ('b \times 'c))\ update \Rightarrow (('a::domain \times 'b::domain) \times 'c::domain)\ update \rangle$ **where**
 $\langle assoc' = (Fst\ o\ Fst; (Fst\ o\ Snd; Snd)) \rangle$

lemma *assoc'-is-hom*[simp]: $\langle preregister\ assoc' \rangle$
 $\langle proof \rangle$

lemma *assoc'-apply*[simp]: $\langle assoc'\ (a\ \otimes_u\ (b\ \otimes_u\ c)) = ((a\ \otimes_u\ b)\ \otimes_u\ c) \rangle$
 $\langle proof \rangle$

lemma *register-assoc*[simp]: $\langle register\ assoc \rangle$
 $\langle proof \rangle$

lemma *register-assoc'*[simp]: $\langle register\ assoc' \rangle$
 $\langle proof \rangle$

lemma *pair-o-assoc*[simp]:
 assumes [simp]: $\langle compatible\ F\ G \rangle$ $\langle compatible\ G\ H \rangle$ $\langle compatible\ F\ H \rangle$
 shows $\langle (F; (G; H))\ o\ assoc = ((F; G); H) \rangle$
 $\langle proof \rangle$

lemma *pair-o-assoc'*[simp]:
assumes [simp]: $\langle \text{compatible } F \ G \rangle \langle \text{compatible } G \ H \rangle \langle \text{compatible } F \ H \rangle$
shows $\langle ((F; G); H) \circ \text{assoc}' = (F; (G; H)) \rangle$
 $\langle \text{proof} \rangle$

lemma *assoc'-o-assoc*[simp]: $\langle \text{assoc}' \circ \text{assoc} = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *assoc'-assoc*[simp]: $\langle \text{assoc}' (\text{assoc } x) = x \rangle$
 $\langle \text{proof} \rangle$

lemma *assoc-o-assoc'*[simp]: $\langle \text{assoc} \circ \text{assoc}' = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *assoc-assoc'*[simp]: $\langle \text{assoc} (\text{assoc}' x) = x \rangle$
 $\langle \text{proof} \rangle$

lemma *inv-assoc*[simp]: $\langle \text{inv } \text{assoc} = \text{assoc}' \rangle$
 $\langle \text{proof} \rangle$

lemma *inv-assoc'*[simp]: $\langle \text{inv } \text{assoc}' = \text{assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma [simp]: $\langle \text{bij } \text{assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma [simp]: $\langle \text{bij } \text{assoc}' \rangle$
 $\langle \text{proof} \rangle$

2.9 Iso-registers

definition $\langle \text{iso-register } F \iff \text{register } F \wedge (\exists G. \text{register } G \wedge F \circ G = \text{id} \wedge G \circ F = \text{id}) \rangle$
for $F :: \langle \text{--} :: \text{domain update} \Rightarrow \text{--} :: \text{domain update} \rangle$

lemma *iso-registerI*:
assumes $\langle \text{register } F \rangle \langle \text{register } G \rangle \langle F \circ G = \text{id} \rangle \langle G \circ F = \text{id} \rangle$
shows $\langle \text{iso-register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-inv*: $\langle \text{iso-register } F \implies \text{iso-register } (\text{inv } F) \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-inv-comp1*: $\langle \text{iso-register } F \implies \text{inv } F \circ F = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-inv-comp2*: $\langle \text{iso-register } F \implies F \circ \text{inv } F = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-id*[simp]: $\langle \text{iso-register } \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-is-register*: $\langle \text{iso-register } F \implies \text{register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-comp*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{iso-register } (F \circ G) \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-tensor-is-iso-register*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$

shows $\langle \text{iso-register } (F \otimes_r G) \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-bij*: $\langle \text{iso-register } F \implies \text{bij } F \rangle$
 $\langle \text{proof} \rangle$

lemma *inv-register-tensor*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{inv } (F \otimes_r G) = \text{inv } F \otimes_r \text{inv } G \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-swap*[simp]: $\langle \text{iso-register swap} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-assoc*[simp]: $\langle \text{iso-register assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-assoc'*[simp]: $\langle \text{iso-register assoc}' \rangle$
 $\langle \text{proof} \rangle$

definition $\langle \text{equivalent-registers } F G \iff (\text{register } F \wedge (\exists I. \text{iso-register } I \wedge F \circ I = G)) \rangle$
for $F G :: \text{--::domain update} \Rightarrow \text{--::domain update}$

lemma *iso-register-equivalent-id*[simp]: $\langle \text{equivalent-registers id } F \iff \text{iso-register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registersI*:
assumes $\langle \text{register } F \rangle$
assumes $\langle \text{iso-register } I \rangle$
assumes $\langle F \circ I = G \rangle$
shows $\langle \text{equivalent-registers } F G \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-register-left*: $\langle \text{equivalent-registers } F G \implies \text{register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-register-right*: $\langle \text{register } G \rangle$ **if** $\langle \text{equivalent-registers } F G \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-sym*:
assumes $\langle \text{equivalent-registers } F G \rangle$
shows $\langle \text{equivalent-registers } G F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-trans*[trans]:
assumes $\langle \text{equivalent-registers } F G \rangle$ **and** $\langle \text{equivalent-registers } G H \rangle$
shows $\langle \text{equivalent-registers } F H \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-assoc*[simp]:
assumes [simp]: $\langle \text{compatible } F G \rangle \langle \text{compatible } F H \rangle \langle \text{compatible } G H \rangle$
shows $\langle \text{equivalent-registers } (F;(G;H)) ((F;G);H) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-pair-right*:
assumes [simp]: $\langle \text{compatible } F G \rangle$
assumes *eq*: $\langle \text{equivalent-registers } G H \rangle$
shows $\langle \text{equivalent-registers } (F;G) (F;H) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-pair-left*:
assumes [simp]: $\langle \text{compatible } F G \rangle$
assumes *eq*: $\langle \text{equivalent-registers } F H \rangle$

shows $\langle \text{equivalent-registers } (F;G) (H;G) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-comp*:
assumes $\langle \text{register } H \rangle$
assumes $\langle \text{equivalent-registers } F G \rangle$
shows $\langle \text{equivalent-registers } (H \circ F) (H \circ G) \rangle$
 $\langle \text{proof} \rangle$

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 .

$\langle ML \rangle$

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 $\langle \text{compatible } - \rightarrow \rangle$] asm-rl[of $\langle \text{iso-register } \rightarrow \rangle$] asm-rl[of $\langle \text{register } \rightarrow \rangle$] 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.

$\langle ML \rangle$

2.11 Notation

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

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

bundle *no-register-notation* **begin**
no-notation *register-tensor* (**infixr** \otimes_r 70)
no-notation *register-pair* ($'(-; -)'$)
end

end

3 Axioms of complements

theory *Axioms-Complement*
imports *Laws*
begin

typedecl ($'a, 'b$) *complement-domain*
instance *complement-domain* :: (*domain, domain*) *domain* $\langle \text{proof} \rangle$
typedecl *some-domain*
instance *some-domain* :: *domain* $\langle \text{proof} \rangle$

axiomatization **where**

complement-exists: $\langle \text{register } F \implies \exists G :: ('a, 'b) \text{ complement-domain update} \Rightarrow 'b \text{ update. compatible } F G \wedge \text{iso-register } (F;G) \rangle$ **for** $F :: \langle 'a::\text{domain update} \Rightarrow 'b::\text{domain update} \rangle$

axiomatization where *complement-unique*: $\langle \text{compatible } F G \implies \text{iso-register } (F;G) \implies \text{compatible } F H \implies \text{iso-register } (F;H) \implies \text{equivalent-registers } G H \rangle$

for $F :: \langle 'a::\text{domain update} \Rightarrow 'b::\text{domain update} \rangle$ **and** $G :: \langle 'g::\text{domain update} \Rightarrow 'b \text{ update} \rangle$ **and** $H :: \langle 'h::\text{domain update} \Rightarrow 'b \text{ update} \rangle$

end

4 Generic laws about complements

theory *Laws-Complement*

imports *Laws Axioms-Complement*

begin

notation *comp-update* (**infixl** $*_u$ 55)

notation *tensor-update* (**infixr** \otimes_u 70)

definition $\langle \text{complements } F G \longleftrightarrow \text{compatible } F G \wedge \text{iso-register } (F;G) \rangle$

lemma *complementsI*: $\langle \text{compatible } F G \implies \text{iso-register } (F;G) \implies \text{complements } F G \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-sym*: $\langle \text{complements } G F \rangle$ **if** $\langle \text{complements } F G \rangle$
 $\langle \text{proof} \rangle$

definition *complement* :: $\langle ('a::\text{domain update} \Rightarrow 'b::\text{domain update}) \Rightarrow (('a, 'b) \text{ complement-domain update} \Rightarrow 'b \text{ update}) \rangle$ **where**
 $\langle \text{complement } F = (\text{SOME } G :: ('a, 'b) \text{ complement-domain update} \Rightarrow 'b \text{ update. compatible } F G \wedge \text{iso-register } (F;G)) \rangle$

lemma *register-complement[simp]*: $\langle \text{register } (\text{complement } F) \rangle$ **if** $\langle \text{register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *complement-is-complement*:
assumes $\langle \text{register } F \rangle$
shows $\langle \text{complements } F (\text{complement } F) \rangle$
 $\langle \text{proof} \rangle$

lemma *complement-unique*:
assumes $\langle \text{complements } F G \rangle$
shows $\langle \text{equivalent-registers } G (\text{complement } F) \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-complement[simp]*: $\langle \text{register } F \implies \text{compatible } F (\text{complement } F) \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-register-tensor*:
assumes $[\text{simp}]$: $\langle \text{register } F \rangle \langle \text{register } G \rangle$
shows $\langle \text{complements } (F \otimes_r G) (\text{complement } F \otimes_r \text{complement } G) \rangle$
 $\langle \text{proof} \rangle$

definition *is-unit-register where*
 $\langle \text{is-unit-register } U \longleftrightarrow \text{complements } U \text{ id} \rangle$

lemma *register-unit-register[simp]*: $\langle \text{is-unit-register } U \implies \text{register } U \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-compatible[simp]*: $\langle \text{compatible } U X \rangle$ **if** $\langle \text{is-unit-register } U \rangle \langle \text{register } X \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-compatible'*[simp]: $\langle \text{compatible } X \ U \rangle$ **if** $\langle \text{is-unit-register } U \rangle$ $\langle \text{register } X \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-complement-left*[simp]: $\langle \text{register } X \implies \text{compatible } (\text{complement } X) \ X \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-complement-right*[simp]: $\langle \text{register } X \implies \text{compatible } X \ (\text{complement } X) \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-pair*[simp]: $\langle \text{equivalent-registers } X \ (U; \ X) \rangle$ **if** [simp]: $\langle \text{is-unit-register } U \rangle$ $\langle \text{register } X \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-compose-left*:
assumes [simp]: $\langle \text{is-unit-register } U \rangle$
assumes [simp]: $\langle \text{register } A \rangle$
shows $\langle \text{is-unit-register } (A \ o \ U) \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-compose-right*:
assumes [simp]: $\langle \text{is-unit-register } U \rangle$
assumes [simp]: $\langle \text{iso-register } A \rangle$
shows $\langle \text{is-unit-register } (U \ o \ A) \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-unique*:
assumes $\langle \text{is-unit-register } F \rangle$
assumes $\langle \text{is-unit-register } G \rangle$
shows $\langle \text{equivalent-registers } F \ G \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-domains-isomorphic*:
fixes $F :: \langle 'a::\text{domain update} \Rightarrow 'c::\text{domain update} \rangle$
fixes $G :: \langle 'b::\text{domain update} \Rightarrow 'd::\text{domain update} \rangle$
assumes $\langle \text{is-unit-register } F \rangle$
assumes $\langle \text{is-unit-register } G \rangle$
shows $\langle \exists I :: 'a \ \text{update} \Rightarrow 'b \ \text{update}. \ \text{iso-register } I \rangle$
 $\langle \text{proof} \rangle$

lemma *id-complement-is-unit-register*[simp]: $\langle \text{is-unit-register } (\text{complement } id) \rangle$
 $\langle \text{proof} \rangle$

type-synonym *unit-register-domain* = $\langle (\text{some-domain}, \ \text{some-domain}) \ \text{complement-domain} \rangle$
definition *unit-register* :: $\langle \text{unit-register-domain update} \Rightarrow 'a::\text{domain update} \rangle$ **where** $\langle \text{unit-register} = (\text{SOME } U. \ \text{is-unit-register } U) \rangle$

lemma *unit-register-is-unit-register*[simp]: $\langle \text{is-unit-register } (\text{unit-register} :: \text{unit-register-domain update} \Rightarrow 'a::\text{domain update}) \rangle$
 $\langle \text{proof} \rangle$

lemma *unit-register-domain-tensor-unit*:
fixes $U :: \langle 'a::\text{domain update} \Rightarrow - \rangle$
assumes $\langle \text{is-unit-register } U \rangle$
shows $\langle \exists I :: 'b::\text{domain update} \Rightarrow ('a * 'b) \ \text{update}. \ \text{iso-register } I \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-complement-pair1*:
assumes $\langle \text{compatible } F \ G \rangle$
shows $\langle \text{compatible } F \ (\text{complement } (F; G)) \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-complement-pair2*:

assumes [simp]: $\langle \text{compatible } F \ G \rangle$
shows $\langle \text{compatible } G \ (\text{complement } (F;G)) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-complements*:
assumes $\langle \text{complements } F \ G \rangle$
assumes $\langle \text{equivalent-registers } G \ G' \rangle$
shows $\langle \text{complements } F \ G' \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-complement-pair*:
assumes [simp]: $\langle \text{compatible } F \ G \rangle$
shows $\langle \text{complements } F \ (G; \text{complement } (F;G)) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-complement*:
assumes $\langle \text{equivalent-registers } F \ G \rangle$
shows $\langle \text{equivalent-registers } (\text{complement } F) \ (\text{complement } G) \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-complement-pair'*:
assumes [simp]: $\langle \text{compatible } F \ G \rangle$
shows $\langle \text{complements } G \ (F; \text{complement } (F;G)) \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-chain*:
assumes [simp]: $\langle \text{register } F \rangle \langle \text{register } G \rangle$
shows $\langle \text{complements } (F \circ G) \ (\text{complement } F; F \circ \text{complement } G) \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-Fst-Snd*[simp]: $\langle \text{complements } Fst \ Snd \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-Snd-Fst*[simp]: $\langle \text{complements } Snd \ Fst \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-unit-register*[simp]: $\langle \text{register } F \implies \text{compatible } F \ \text{unit-register} \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-id-unit-register*[simp]: $\langle \text{complements } id \ \text{unit-register} \rangle$
 $\langle \text{proof} \rangle$

lemma *complements-iso-unit-register*: $\langle \text{iso-register } I \implies \text{is-unit-register } U \implies \text{complements } I \ U \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-complement-is-unit-register*[simp]:
assumes $\langle \text{iso-register } F \rangle$
shows $\langle \text{is-unit-register } (\text{complement } F) \rangle$
 $\langle \text{proof} \rangle$

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 $\langle \text{is-unit-register } - \rangle$]

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

end

5 Classical instantiation of registerss

theory *Axioms-Classical*

imports *Main*
begin

type-synonym *'a update* = $\langle 'a \rightarrow 'a \rangle$

lemma *id-update-left*: *Some* \circ_m *a* = *a*
 \langle *proof* \rangle

lemma *id-update-right*: *a* \circ_m *Some* = *a*
 \langle *proof* \rangle

lemma *comp-update-assoc*: (*a* \circ_m *b*) \circ_m *c* = *a* \circ_m (*b* \circ_m *c*)
 \langle *proof* \rangle

type-synonym (*'a, 'b*) *preregister* = $\langle 'a \text{ update} \Rightarrow 'b \text{ update} \rangle$

definition *preregister* :: $\langle ('a, 'b) \text{ preregister} \Rightarrow \text{bool} \rangle$ **where**
 \langle *preregister* *F* $\longleftrightarrow (\exists g s. \forall a m. F a m = (\text{case } a (g m) \text{ of } \text{None} \Rightarrow \text{None} \mid \text{Some } x \Rightarrow s x m)) \rangle$

lemma *id-preregister*: \langle *preregister* *id* \rangle
 \langle *proof* \rangle

lemma *preregister-mult-right*: \langle *preregister* ($\lambda a. a \circ_m z$) \rangle
 \langle *proof* \rangle

lemma *preregister-mult-left*: \langle *preregister* ($\lambda a. z \circ_m a$) \rangle
 \langle *proof* \rangle

lemma *comp-preregister*: *preregister* (*G* \circ *F*) **if** *preregister* *F* **and** \langle *preregister* *G* \rangle
 \langle *proof* \rangle

definition *tensor-update* :: $\langle 'a \text{ update} \Rightarrow 'b \text{ update} \Rightarrow ('a \times 'b) \text{ update} \rangle$ **where**
 \langle *tensor-update* *a b m* = (*case* *a* (*fst* *m*) *of* *None* \Rightarrow *None* \mid *Some* *x* \Rightarrow (*case* *b* (*snd* *m*) *of* *None* \Rightarrow *None* \mid *Some* *y* \Rightarrow *Some* (*x, y*))) \rangle

lemma *tensor-update-mult*: \langle *tensor-update* *a c* \circ_m *tensor-update* *b d* = *tensor-update* (*a* \circ_m *b*) (*c* \circ_m *d*) \rangle
 \langle *proof* \rangle

definition *update1* :: $\langle 'a \Rightarrow 'a \Rightarrow 'a \text{ update} \rangle$ **where**
 \langle *update1* *x y m* = (*if* *m*=*x* *then* *Some* *y* *else* *None*) \rangle

lemma *update1-extensionality*:
assumes \langle *preregister* *F* \rangle
assumes \langle *preregister* *G* \rangle
assumes *F**Geq*: $\langle \bigwedge x y. F (\text{update1 } x y) = G (\text{update1 } x y) \rangle$
shows *F* = *G*
 \langle *proof* \rangle

lemma *tensor-extensionality*:
assumes \langle *preregister* *F* \rangle
assumes \langle *preregister* *G* \rangle
assumes *F**Geq*: $\langle \bigwedge a b. F (\text{tensor-update } a b) = G (\text{tensor-update } a b) \rangle$
shows *F* = *G*
 \langle *proof* \rangle

definition *valid-getter-setter* *g s* \longleftrightarrow
 $(\forall b. b = s (g b) b) \wedge (\forall a b. g (s a b) = a) \wedge (\forall a a' b. s a (s a' b) = s a b)$

definition \langle *register-from-getter-setter* *g s a m* = (*case* *a* (*g m*) *of* *None* \Rightarrow *None* \mid *Some* *x* \Rightarrow *Some* (*s x m*)) \rangle

definition \langle *register-apply* *F a* = *the* *o* *F* (*Some* *o a*) \rangle

definition \langle *setter* *F a m* = *register-apply* *F* ($\lambda-. a$) *m* \rangle **for** *F* :: $\langle 'a \text{ update} \Rightarrow 'b \text{ update} \rangle$

definition \langle *getter* *F m* = (*THE* *x. setter* *F x m* = *m*) \rangle **for** *F* :: $\langle 'a \text{ update} \Rightarrow 'b \text{ update} \rangle$

lemma
assumes \langle *valid-getter-setter* *g s* \rangle

shows *getter-of-register-from-getter-setter*[simp]: $\langle \text{getter } (\text{register-from-getter-setter } g \ s) = g \rangle$
and *setter-of-register-from-getter-setter*[simp]: $\langle \text{setter } (\text{register-from-getter-setter } g \ s) = s \rangle$
 $\langle \text{proof} \rangle$

definition *register* :: $\langle ('a, 'b) \text{ preregister} \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{register } F \longleftrightarrow (\exists g \ s. F = \text{register-from-getter-setter } g \ s \wedge \text{valid-getter-setter } g \ s) \rangle$

lemma *register-of-id*: $\langle \text{register } F \Longrightarrow F \text{ Some} = \text{Some} \rangle$
 $\langle \text{proof} \rangle$

lemma *register-id*: $\langle \text{register } \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *register-tensor-left*: $\langle \text{register } (\lambda a. \text{tensor-update } a \ \text{Some}) \rangle$
 $\langle \text{proof} \rangle$

lemma *register-tensor-right*: $\langle \text{register } (\lambda a. \text{tensor-update } \text{Some } a) \rangle$
 $\langle \text{proof} \rangle$

lemma *register-preregister*: *preregister* *F* **if** $\langle \text{register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *register-comp*: *register* (*G* \circ *F*) **if** $\langle \text{register } F \rangle$ **and** $\langle \text{register } G \rangle$
for *F* :: $\langle ('a, 'b) \text{ preregister} \rangle$ **and** *G* :: $\langle ('b, 'c) \text{ preregister} \rangle$
 $\langle \text{proof} \rangle$

lemma *register-mult*: *register* *F* $\Longrightarrow F \ a \ \circ_m \ F \ b = F \ (a \ \circ_m \ b)$
 $\langle \text{proof} \rangle$

definition *register-pair* ::
 $\langle ('a \ \text{update} \Rightarrow 'c \ \text{update}) \Rightarrow ('b \ \text{update} \Rightarrow 'c \ \text{update}) \Rightarrow (('a \times 'b) \ \text{update} \Rightarrow 'c \ \text{update}) \rangle$ **where**
 $\langle \text{register-pair } F \ G =$
 $\text{register-from-getter-setter } (\lambda m. (\text{getter } F \ m, \text{getter } G \ m)) (\lambda (a, b) \ m. \text{setter } F \ a \ (\text{setter } G \ b \ m)) \rangle$

lemma *compatible-setter*:
assumes [simp]: $\langle \text{register } F \rangle$ $\langle \text{register } G \rangle$
assumes *compat*: $\langle \bigwedge a \ b. F \ a \ \circ_m \ G \ b = G \ b \ \circ_m \ F \ a \rangle$
shows $\langle \text{setter } F \ x \ o \ \text{setter } G \ y = \text{setter } G \ y \ o \ \text{setter } F \ x \rangle$
 $\langle \text{proof} \rangle$

lemma *register-pair-apply*:
assumes [simp]: $\langle \text{register } F \rangle$ $\langle \text{register } G \rangle$
assumes $\langle \bigwedge a \ b. F \ a \ \circ_m \ G \ b = G \ b \ \circ_m \ F \ a \rangle$
shows $\langle (\text{register-pair } F \ G) (\text{tensor-update } a \ b) = F \ a \ \circ_m \ G \ b \rangle$
 $\langle \text{proof} \rangle$

lemma *register-pair-is-register*:
fixes *F* :: $\langle 'a \ \text{update} \Rightarrow 'c \ \text{update} \rangle$ **and** *G*
assumes [simp]: $\langle \text{register } F \rangle$ **and** [simp]: $\langle \text{register } G \rangle$
assumes *compat*: $\langle \bigwedge a \ b. F \ a \ \circ_m \ G \ b = G \ b \ \circ_m \ F \ a \rangle$
shows $\langle \text{register } (\text{register-pair } F \ G) \rangle$
 $\langle \text{proof} \rangle$

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]: $\langle \text{preregister } (\lambda b :: 'b::\text{type update. tensor-update } a \ b) \rangle$
for $a :: 'a::\text{type update}$
 $\langle \text{proof} \rangle$

lemma *preregister-tensor-right*[simp]: $\langle \text{preregister } (\lambda a :: 'a::\text{type update. tensor-update } a \ b) \rangle$
for $b :: 'b::\text{type update}$
 $\langle \text{proof} \rangle$

6.3 Registers

lemma *id-update-tensor-register*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle \text{register } (\lambda a :: 'a::\text{type update. Some } \otimes_u \ F \ a) \rangle$
 $\langle \text{proof} \rangle$

lemma *register-tensor-id-update*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle \text{register } (\lambda a :: 'a::\text{type update. } F \ a \ \otimes_u \ \text{Some}) \rangle$
 $\langle \text{proof} \rangle$

6.4 Tensor product of registers

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

lemma *register-tensor-is-register*:
fixes $F :: 'a::\text{type update} \Rightarrow 'b::\text{type update}$ **and** $G :: 'c::\text{type update} \Rightarrow 'd::\text{type update}$
shows $\text{register } F \ \Longrightarrow \ \text{register } G \ \Longrightarrow \ \text{register } (F \ \otimes_r \ G)$
 $\langle \text{proof} \rangle$

lemma *register-tensor-apply*[simp]:
fixes $F :: 'a::\text{type update} \Rightarrow 'b::\text{type update}$ **and** $G :: 'c::\text{type update} \Rightarrow 'd::\text{type update}$
assumes $\langle \text{register } F \rangle$ **and** $\langle \text{register } G \rangle$
shows $(F \ \otimes_r \ G) \ (a \ \otimes_u \ b) = F \ a \ \otimes_u \ G \ b$
 $\langle \text{proof} \rangle$

definition *separating* ($- :: 'b::\text{type itself}$) $A \ \longleftrightarrow$
 $(\forall F \ G :: 'a::\text{type update} \Rightarrow 'b \ \text{update. } \text{preregister } F \ \longrightarrow \ \text{preregister } G \ \longrightarrow \ (\forall x \in A. F \ x = G \ x) \ \longrightarrow \ F = G)$

lemma *separating-UNIV*[simp]: $\langle \text{separating } \text{TYPE}(-) \ \text{UNIV} \rangle$
 $\langle \text{proof} \rangle$

lemma *separating-mono*: $\langle A \subseteq B \ \Longrightarrow \ \text{separating } \text{TYPE}('a::\text{type}) \ A \ \Longrightarrow \ \text{separating } \text{TYPE}('a) \ B \rangle$
 $\langle \text{proof} \rangle$

lemma register-eqI: $\langle \text{separating TYPE('b::type)} A \implies \text{preregister } F \implies \text{preregister } G \implies (\bigwedge x. x \in A \implies F x = G x) \implies F = (G::- \Rightarrow 'b \text{ update}) \rangle$
 $\langle \text{proof} \rangle$

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

lemma register-tensor-distrib:
assumes $[\text{simp}]: \langle \text{register } F \rangle \langle \text{register } G \rangle \langle \text{register } H \rangle \langle \text{register } L \rangle$
shows $\langle (F \otimes_r G) \circ (H \otimes_r L) = (F \circ H) \otimes_r (G \circ L) \rangle$
 $\langle \text{proof} \rangle$

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

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

lemma tensor-extensionality3:
fixes $F G :: \langle ('a::\text{type} \times 'b::\text{type} \times 'c::\text{type}) \text{ update} \Rightarrow 'd::\text{type update} \rangle$
assumes $[\text{simp}]: \langle \text{register } F \rangle \langle \text{register } G \rangle$
assumes $\bigwedge f g h. F (f \otimes_u g \otimes_u h) = G (f \otimes_u g \otimes_u h)$
shows $F = G$
 $\langle \text{proof} \rangle$

lemma tensor-extensionality3':
fixes $F G :: \langle (('a::\text{type} \times 'b::\text{type}) \times 'c::\text{type}) \text{ update} \Rightarrow 'd::\text{type update} \rangle$
assumes $[\text{simp}]: \langle \text{register } F \rangle \langle \text{register } G \rangle$
assumes $\bigwedge f g h. F ((f \otimes_u g) \otimes_u h) = G ((f \otimes_u g) \otimes_u h)$
shows $F = G$
 $\langle \text{proof} \rangle$

lemma register-tensor-id $[\text{simp}]: \langle \text{id} \otimes_r \text{id} = \text{id} \rangle$
 $\langle \text{proof} \rangle$

6.5 Pairs and compatibility

definition compatible $:: \langle ('a::\text{type update} \Rightarrow 'c::\text{type update}) \Rightarrow ('b::\text{type update} \Rightarrow 'c \text{ update}) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{compatible } F G \iff \text{register } F \wedge \text{register } G \wedge (\forall a b. F a *_u G b = G b *_u F a) \rangle$

lemma compatibleI:
assumes $\text{register } F$ **and** $\text{register } G$
assumes $\langle \bigwedge a b. (F a) *_u (G b) = (G b) *_u (F a) \rangle$
shows $\text{compatible } F G$
 $\langle \text{proof} \rangle$

lemma swap-registers:
assumes $\text{compatible } R S$
shows $R a *_u S b = S b *_u R a$
 $\langle \text{proof} \rangle$

lemma compatible-sym: $\text{compatible } x y \implies \text{compatible } y x$
 $\langle \text{proof} \rangle$

lemma pair-is-register $[\text{simp}]:$

assumes *compatible F G*
shows *register (F; G)*
<proof>

lemma *register-pair-apply*:
assumes *<compatible F G>*
shows *<(F; G) (a ⊗_u b) = (F a) *_u (G b)>*
<proof>

lemma *register-pair-apply'*:
assumes *<compatible F G>*
shows *<(F; G) (a ⊗_u b) = (G b) *_u (F a)>*
<proof>

lemma *compatible-comp-left[simp]*: *compatible F G ⇒ register H ⇒ compatible (F ∘ H) G*
<proof>

lemma *compatible-comp-right[simp]*: *compatible F G ⇒ register H ⇒ compatible F (G ∘ H)*
<proof>

lemma *compatible-comp-inner[simp]*:
compatible F G ⇒ register H ⇒ compatible (H ∘ F) (H ∘ G)
<proof>

lemma *compatible-register1*: *<compatible F G ⇒ register F>*
<proof>

lemma *compatible-register2*: *<compatible F G ⇒ register G>*
<proof>

lemma *pair-o-tensor*:
assumes *compatible A B* **and** *[simp]: <register C>* **and** *[simp]: <register D>*
shows *(A; B) ∘ (C ⊗_r D) = (A ∘ C; B ∘ 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 (λa. Some ⊗_u F a) (λa. Some ⊗_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 (λa. F a ⊗_u Some) (λa. G a ⊗_u Some)*
<proof>

lemma *compatible-tensor-id-update-rl[simp]*:
assumes *register F* **and** *register G*
shows *compatible (λa. F a ⊗_u Some) (λa. Some ⊗_u G a)*
<proof>

lemma *compatible-tensor-id-update-lr[simp]*:
assumes *register F* **and** *register G*
shows *compatible (λa. Some ⊗_u F a) (λa. G a ⊗_u Some)*
<proof>

lemma *register-comp-pair*:
assumes *[simp]: <register F>* **and** *[simp]: <compatible G H>*
shows *(F ∘ G; F ∘ H) = F ∘ (G; H)*
<proof>

lemma *swap-registers-left*:
assumes *compatible R S*
shows $R a *_u S b *_u c = S b *_u R a *_u c$
 $\langle \text{proof} \rangle$

lemma *swap-registers-right*:
assumes *compatible R S*
shows $c *_u R a *_u S b = c *_u S b *_u R a$
 $\langle \text{proof} \rangle$

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

6.6 Fst and Snd

definition *Fst* **where** $\langle \text{Fst } a = a \otimes_u \text{Some} \rangle$

definition *Snd* **where** $\langle \text{Snd } a = \text{Some} \otimes_u a \rangle$

lemma *register-Fst[simp]*: $\langle \text{register } \text{Fst} \rangle$
 $\langle \text{proof} \rangle$

lemma *register-Snd[simp]*: $\langle \text{register } \text{Snd} \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-Fst-Snd[simp]*: $\langle \text{compatible } \text{Fst } \text{Snd} \rangle$
 $\langle \text{proof} \rangle$

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

definition $\langle \text{swap} = (\text{Snd}; \text{Fst}) \rangle$

lemma *swap-apply[simp]*: $\text{swap } (a \otimes_u b) = (b \otimes_u a)$
 $\langle \text{proof} \rangle$

lemma *swap-o-Fst*: $\text{swap } o \text{Fst} = \text{Snd}$
 $\langle \text{proof} \rangle$

lemma *swap-o-Snd*: $\text{swap } o \text{Snd} = \text{Fst}$
 $\langle \text{proof} \rangle$

lemma *register-swap[simp]*: $\langle \text{register } \text{swap} \rangle$
 $\langle \text{proof} \rangle$

lemma *pair-Fst-Snd*: $\langle (\text{Fst}; \text{Snd}) = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *swap-o-swap[simp]*: $\langle \text{swap } o \text{swap} = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *swap-swap[simp]*: $\langle \text{swap } (\text{swap } x) = x \rangle$
 $\langle \text{proof} \rangle$

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

lemma *register-pair-Fst*:
assumes $\langle \text{compatible } F \ G \rangle$
shows $\langle (F;G) o \text{Fst} = F \rangle$
 $\langle \text{proof} \rangle$

lemma *register-pair-Snd*:
assumes $\langle \text{compatible } F \ G \rangle$
shows $\langle (F;G) o \text{Snd} = G \rangle$
 $\langle \text{proof} \rangle$

lemma *register-Fst-register-Snd*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle (F \circ \text{Fst}; F \circ \text{Snd}) = F \rangle$
 $\langle \text{proof} \rangle$

lemma *register-Snd-register-Fst*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle (F \circ \text{Snd}; F \circ \text{Fst}) = F \circ \text{swap} \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible3*[simp]:
assumes [simp]: *compatible* $F\ G$ **and** *compatible* $G\ H$ **and** *compatible* $F\ H$
shows *compatible* $(F; G)\ H$
 $\langle \text{proof} \rangle$

lemma *compatible3'*[simp]:
assumes *compatible* $F\ G$ **and** *compatible* $G\ H$ **and** *compatible* $F\ H$
shows *compatible* $F\ (G; H)$
 $\langle \text{proof} \rangle$

lemma *pair-o-swap*[simp]:
assumes [simp]: *compatible* $A\ B$
shows $(A; B) \circ \text{swap} = (B; A)$
 $\langle \text{proof} \rangle$

6.7 Compatibility of register tensor products

lemma *compatible-register-tensor*:
fixes $F :: \langle 'a::\text{type update} \Rightarrow 'e::\text{type update} \rangle$ **and** $G :: \langle 'b::\text{type update} \Rightarrow 'f::\text{type update} \rangle$
and $F' :: \langle 'c::\text{type update} \Rightarrow 'e\ \text{update} \rangle$ **and** $G' :: \langle 'd::\text{type update} \Rightarrow 'f\ \text{update} \rangle$
assumes [simp]: $\langle \text{compatible } F\ F' \rangle$
assumes [simp]: $\langle \text{compatible } G\ G' \rangle$
shows $\langle \text{compatible } (F \otimes_r G)\ (F' \otimes_r G') \rangle$
 $\langle \text{proof} \rangle$

6.8 Associativity of the tensor product

definition *assoc* :: $\langle (('a::\text{type} \times 'b::\text{type}) \times 'c::\text{type})\ \text{update} \Rightarrow ('a \times ('b \times 'c))\ \text{update} \rangle$ **where**
 $\langle \text{assoc} = ((\text{Fst}; \text{Snd} \circ \text{Fst}); \text{Snd} \circ \text{Snd}) \rangle$

lemma *assoc-is-hom*[simp]: $\langle \text{preregister } \text{assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma *assoc-apply*[simp]: $\langle \text{assoc } ((a \otimes_u b) \otimes_u c) = (a \otimes_u (b \otimes_u c)) \rangle$
 $\langle \text{proof} \rangle$

definition *assoc'* :: $\langle ('a \times ('b \times 'c))\ \text{update} \Rightarrow (('a::\text{type} \times 'b::\text{type}) \times 'c::\text{type})\ \text{update} \rangle$ **where**
 $\langle \text{assoc}' = (\text{Fst} \circ \text{Fst}; (\text{Fst} \circ \text{Snd}; \text{Snd})) \rangle$

lemma *assoc'-is-hom*[simp]: $\langle \text{preregister } \text{assoc}' \rangle$
 $\langle \text{proof} \rangle$

lemma *assoc'-apply*[simp]: $\langle \text{assoc}' (a \otimes_u (b \otimes_u c)) = ((a \otimes_u b) \otimes_u c) \rangle$
 $\langle \text{proof} \rangle$

lemma *register-assoc*[simp]: $\langle \text{register } \text{assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma *register-assoc'*[simp]: $\langle \text{register } \text{assoc}' \rangle$
 $\langle \text{proof} \rangle$

lemma *pair-o-assoc*[simp]:
assumes [simp]: $\langle \text{compatible } F\ G \rangle$ $\langle \text{compatible } G\ H \rangle$ $\langle \text{compatible } F\ H \rangle$

shows $\langle (F; (G; H)) \circ \text{assoc} = ((F; G); H) \rangle$
<proof>

lemma *pair-o-assoc'*[simp]:
assumes [simp]: $\langle \text{compatible } F \ G \rangle \langle \text{compatible } G \ H \rangle \langle \text{compatible } F \ H \rangle$
shows $\langle ((F; G); H) \circ \text{assoc}' = (F; (G; H)) \rangle$
<proof>

lemma *assoc'-o-assoc*[simp]: $\langle \text{assoc}' \ o \ \text{assoc} = \text{id} \rangle$
<proof>

lemma *assoc'-assoc*[simp]: $\langle \text{assoc}' (\text{assoc } x) = x \rangle$
<proof>

lemma *assoc-o-assoc'*[simp]: $\langle \text{assoc } o \ \text{assoc}' = \text{id} \rangle$
<proof>

lemma *assoc-assoc'*[simp]: $\langle \text{assoc} (\text{assoc}' x) = x \rangle$
<proof>

lemma *inv-assoc*[simp]: $\langle \text{inv } \text{assoc} = \text{assoc}' \rangle$
<proof>

lemma *inv-assoc'*[simp]: $\langle \text{inv } \text{assoc}' = \text{assoc} \rangle$
<proof>

lemma [simp]: $\langle \text{bij } \text{assoc} \rangle$
<proof>

lemma [simp]: $\langle \text{bij } \text{assoc}' \rangle$
<proof>

6.9 Iso-registers

definition $\langle \text{iso-register } F \longleftrightarrow \text{register } F \wedge (\exists G. \text{register } G \wedge F \ o \ G = \text{id} \wedge G \ o \ F = \text{id}) \rangle$
for $F :: \langle \text{--}::\text{type update} \Rightarrow \text{--}::\text{type update} \rangle$

lemma *iso-registerI*:
assumes $\langle \text{register } F \rangle \langle \text{register } G \rangle \langle F \ o \ G = \text{id} \rangle \langle G \ o \ F = \text{id} \rangle$
shows $\langle \text{iso-register } F \rangle$
<proof>

lemma *iso-register-inv*: $\langle \text{iso-register } F \Longrightarrow \text{iso-register } (\text{inv } F) \rangle$
<proof>

lemma *iso-register-inv-comp1*: $\langle \text{iso-register } F \Longrightarrow \text{inv } F \ o \ F = \text{id} \rangle$
<proof>

lemma *iso-register-inv-comp2*: $\langle \text{iso-register } F \Longrightarrow F \ o \ \text{inv } F = \text{id} \rangle$
<proof>

lemma *iso-register-id*[simp]: $\langle \text{iso-register } \text{id} \rangle$
<proof>

lemma *iso-register-is-register*: $\langle \text{iso-register } F \Longrightarrow \text{register } F \rangle$
<proof>

lemma *iso-register-comp*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{iso-register } (F \ o \ G) \rangle$
<proof>

lemma *iso-register-tensor-is-iso-register*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{iso-register } (F \otimes_r G) \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-bij*: $\langle \text{iso-register } F \implies \text{bij } F \rangle$
 $\langle \text{proof} \rangle$

lemma *inv-register-tensor*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{inv } (F \otimes_r G) = \text{inv } F \otimes_r \text{inv } G \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-swap*[simp]: $\langle \text{iso-register swap} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-assoc*[simp]: $\langle \text{iso-register assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-assoc'*[simp]: $\langle \text{iso-register assoc}' \rangle$
 $\langle \text{proof} \rangle$

definition $\langle \text{equivalent-registers } F G \iff (\text{register } F \wedge (\exists I. \text{iso-register } I \wedge F \circ I = G)) \rangle$
for $F G :: \langle \text{--::type update} \implies \text{--::type update} \rangle$

lemma *iso-register-equivalent-id*[simp]: $\langle \text{equivalent-registers id } F \iff \text{iso-register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registersI*:
assumes $\langle \text{register } F \rangle$
assumes $\langle \text{iso-register } I \rangle$
assumes $\langle F \circ I = G \rangle$
shows $\langle \text{equivalent-registers } F G \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-register-left*: $\langle \text{equivalent-registers } F G \implies \text{register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-register-right*: $\langle \text{register } G \rangle$ **if** $\langle \text{equivalent-registers } F G \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-sym*:
assumes $\langle \text{equivalent-registers } F G \rangle$
shows $\langle \text{equivalent-registers } G F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-trans*[trans]:
assumes $\langle \text{equivalent-registers } F G \rangle$ **and** $\langle \text{equivalent-registers } G H \rangle$
shows $\langle \text{equivalent-registers } F H \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-assoc*[simp]:
assumes [simp]: $\langle \text{compatible } F G \rangle \langle \text{compatible } F H \rangle \langle \text{compatible } G H \rangle$
shows $\langle \text{equivalent-registers } (F;(G;H)) ((F;G);H) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-pair-right*:
assumes [simp]: $\langle \text{compatible } F G \rangle$
assumes *eq*: $\langle \text{equivalent-registers } G H \rangle$
shows $\langle \text{equivalent-registers } (F;G) (F;H) \rangle$
 $\langle \text{proof} \rangle$

```

lemma equivalent-registers-pair-left:
  assumes [simp]:  $\langle \text{compatible } F \ G \rangle$ 
  assumes eq:  $\langle \text{equivalent-registers } F \ H \rangle$ 
  shows  $\langle \text{equivalent-registers } (F;G) \ (H;G) \rangle$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma equivalent-registers-comp:
  assumes  $\langle \text{register } H \rangle$ 
  assumes  $\langle \text{equivalent-registers } F \ G \rangle$ 
  shows  $\langle \text{equivalent-registers } (H \circ F) \ (H \circ G) \rangle$ 
   $\langle \text{proof} \rangle$ 

```

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 .

$\langle ML \rangle$

```

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  $\langle \text{compatible } - \ - \rangle$ ] asm-rl[of  $\langle \text{iso-register } - \rangle$ ] asm-rl[of  $\langle \text{register } - \rangle$ ] 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.

$\langle ML \rangle$

6.11 Notation

```

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

```

```

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

```

```

bundle no-register-notation begin
no-notation register-tensor (infixr  $\otimes_r$  70)
no-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

```


begin

— Remove notation that collides with the notation we use

no-notation *Order.top* (\top)

no-notation *m-inv* (*inv* - [81] 80)

unbundle *no-vec-syntax*

unbundle *no-inner-syntax*

— Import notation from Bounded Operator and Jordan Normal Form libraries

unbundle *cblinfun-notation*

unbundle *jnf-notation*

abbreviation *butterket* $i\ j \equiv \text{butterfly } (ket\ i)\ (ket\ j)$

abbreviation *selfbutterket* $i \equiv \text{butterfly } (ket\ i)\ (ket\ i)$

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* $P \longleftrightarrow P\ (0::bit) \wedge P\ 1$

definition *enum-ex-bit* $P \longleftrightarrow P\ (0::bit) \vee P\ 1$

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 $n\ vs \in \text{carrier-mat } (length\ vs)\ n$

dim-row (*mat-of-rows-list* $n\ vs$) = *length* *vs*

dim-col (*mat-of-rows-list* $n\ vs$) = n

<proof>

lemma *apply-id-cblinfun[simp]*: $\langle (*_V)\ id\text{-cblinfun} = id \rangle$

<proof>

Overriding

`isaComplex-Bounded-Operators.Complex-Bounded-Linear-Function.sandwich`. The latter is the same function but defined as $a \Rightarrow_{CL}$ - which is less convenient for us.

definition *sandwich* **where** $\langle sandwich\ a\ b = a\ o_{CL}\ b\ o_{CL}\ a^* \rangle$

lemma *clinear-sandwich[simp]*: $\langle \text{clinear } (sandwich\ a) \rangle$

<proof>

lemma *sandwich-id[simp]*: $\langle sandwich\ id\text{-cblinfun} = id \rangle$

<proof>

lemma *mat-of-cblinfun-sandwich*:

fixes $a :: (-::\text{onb-enum}, -::\text{onb-enum}) \text{ cblinfun}$
shows $\langle \text{mat-of-cblinfun} (\text{sandwich } a \ b) = (\text{let } a' = \text{mat-of-cblinfun } a \text{ in } a' * \text{mat-of-cblinfun } b * \text{mat-adjoint } a') \rangle$
 $\langle \text{proof} \rangle$

lemma *prod-cases3'* [*cases type*]:

obtains (*fields*) $a \ b \ c$ **where** $y = ((a, b), c)$
 $\langle \text{proof} \rangle$

lemma *lift-cblinfun-comp*:

assumes $\langle a \ o_{CL} \ b = c \rangle$
shows $\langle a \ o_{CL} \ b = c \rangle$
and $\langle a \ o_{CL} \ (b \ o_{CL} \ d) = c \ o_{CL} \ d \rangle$
and $\langle a \ *_{S} \ (b \ *_{S} \ S) = c \ *_{S} \ S \rangle$
and $\langle a \ *_{V} \ (b \ *_{V} \ x) = c \ *_{V} \ x \rangle$
 $\langle \text{proof} \rangle$

We define the following abbreviations:

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

syntax *-mutually* $:: 'a \Rightarrow \text{args} \Rightarrow 'b \ (\text{mutually} \ - \ '(-))$

syntax *-mutually2* $:: 'a \Rightarrow 'b \Rightarrow \text{args} \Rightarrow \text{args} \Rightarrow 'c$

translations *mutually* $f \ (x) \Rightarrow \text{CONST } \text{True}$

translations *mutually* $f \ (-\text{args } x \ y) \Rightarrow f \ x \ y \ \wedge \ f \ y \ x$

translations *mutually* $f \ (-\text{args } x \ (-\text{args } x' \ xs)) \Rightarrow \text{-mutually2 } f \ x \ (-\text{args } x' \ xs) \ (-\text{args } x' \ xs)$

translations *-mutually2* $f \ x \ y \ zs \Rightarrow f \ x \ y \ \wedge \ f \ y \ x \ \wedge \ \text{-mutually } f \ zs$

translations *-mutually2* $f \ x \ (-\text{args } y \ ys) \ zs \Rightarrow f \ x \ y \ \wedge \ f \ y \ x \ \wedge \ \text{-mutually2 } f \ x \ ys \ zs$

syntax *-each* $:: 'a \Rightarrow \text{args} \Rightarrow 'b \ (\text{each} \ - \ '(-))$

translations *each* $f \ (x) \Rightarrow f \ x$

translations *-each* $f \ (-\text{args } x \ xs) \Rightarrow f \ x \ \wedge \ \text{-each } f \ xs$

lemma *enum-inj*:

assumes $i < \text{CARD}('a)$ **and** $j < \text{CARD}('a)$

shows $(\text{Enum.enum} ! i :: 'a::\text{enum}) = \text{Enum.enum} ! j \longleftrightarrow i = j$
 $\langle \text{proof} \rangle$

lemma [*simp*]: $\text{dim-col} (\text{mat-adjoint } m) = \text{dim-row } m$

$\langle \text{proof} \rangle$

lemma [*simp*]: $\text{dim-row} (\text{mat-adjoint } m) = \text{dim-col } m$

$\langle \text{proof} \rangle$

lemma *invI*:

assumes $\langle \text{inj } f \rangle$

assumes $\langle x = f \ y \rangle$

shows $\langle \text{inv } f \ x = y \rangle$

$\langle \text{proof} \rangle$

instantiation *prod* $:: (\text{default}, \text{default}) \text{ default } \text{begin}$

definition $\langle \text{default-prod} = (\text{default}, \text{default}) \rangle$

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

end

instance *bit* $:: \text{default} \langle \text{proof} \rangle$

lemma *surj-from-comp*:

assumes $\langle \text{surj } (g \ o \ f) \rangle$

assumes $\langle inj\ g \rangle$
shows $\langle surj\ f \rangle$
 $\langle proof \rangle$

lemma *double-exists*: $\langle (\exists x\ y.\ Q\ x\ y) \longleftrightarrow (\exists z.\ Q\ (fst\ z)\ (snd\ z)) \rangle$
 $\langle proof \rangle$

end

8 Derived facts about classical registers

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

lemma *register-from-getter-setter-of-getter-setter*[simp]: $\langle register\ from\ getter\ setter\ (getter\ F)\ (setter\ F) = F \rangle$
if $\langle register\ F \rangle$
 $\langle proof \rangle$

lemma *valid-getter-setter-getter-setter*[simp]: $\langle valid\ getter\ setter\ (getter\ F)\ (setter\ F) \rangle$ **if** $\langle register\ F \rangle$
 $\langle proof \rangle$

lemma *register-register-from-getter-setter*[simp]: $\langle register\ (register\ from\ getter\ setter\ g\ s) \rangle$ **if** $\langle valid\ getter\ setter\ g\ s \rangle$
 $\langle proof \rangle$

definition $\langle total\ fun\ f = (\forall x.\ f\ x \neq None) \rangle$

lemma *register-total*:
assumes $\langle register\ F \rangle$
assumes $\langle total\ fun\ a \rangle$
shows $\langle total\ fun\ (F\ a) \rangle$
 $\langle proof \rangle$

lemma *register-apply*:
assumes $\langle register\ F \rangle$
shows $\langle Some\ o\ register\ apply\ F\ a = F\ (Some\ o\ a) \rangle$
 $\langle proof \rangle$

lemma *register-empty*:
assumes $\langle preregister\ F \rangle$
shows $\langle F\ Map.empty = Map.empty \rangle$
 $\langle proof \rangle$

lemma *compatible-setter*:
fixes $F :: \langle ('a, 'c)\ preregister \rangle$ **and** $G :: \langle ('b, 'c)\ preregister \rangle$
assumes [simp]: $\langle register\ F \rangle \langle register\ G \rangle$
shows $\langle compatible\ F\ G \longleftrightarrow (\forall a\ b.\ setter\ F\ a\ o\ setter\ G\ b = setter\ G\ b\ o\ setter\ F\ a) \rangle$
 $\langle proof \rangle$

lemma *register-from-getter-setter-compatibleI*[intro]:
assumes [simp]: $\langle valid\ getter\ setter\ g\ s \rangle \langle valid\ getter\ setter\ g'\ s' \rangle$
assumes $\langle \bigwedge x\ y\ m.\ s\ x\ (s'\ y\ m) = s'\ y\ (s\ x\ m) \rangle$
shows $\langle compatible\ (register\ from\ getter\ setter\ g\ s)\ (register\ from\ getter\ setter\ g'\ s') \rangle$
 $\langle proof \rangle$

lemma *separating-update1*:
 $\langle separating\ TYPE(-)\ \{update1\ x\ y\ | x\ y.\ True\} \rangle$
 $\langle proof \rangle$

definition *permutation-register* $(p::'b \Rightarrow 'a) = register\ from\ getter\ setter\ p\ (\lambda a.\ inv\ p\ a)$

lemma *permutation-register-register*[simp]:

fixes $p :: 'b \Rightarrow 'a$
assumes $[simp]: \text{bij } p$
shows $\text{register } (\text{permutation-register } p)$
 $\langle \text{proof} \rangle$

lemma $\text{getter-permutation-register}: \langle \text{bij } p \implies \text{getter } (\text{permutation-register } p) = p \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{setter-permutation-register}: \langle \text{bij } p \implies \text{setter } (\text{permutation-register } p) a m = \text{inv } p a \rangle$
 $\langle \text{proof} \rangle$

definition $\text{empty-var} :: \langle 'a::\{\text{CARD-1}\} \text{ update} \Rightarrow 'b \text{ update} \rangle$ **where**
 $\text{empty-var} = \text{register-from-getter-setter } (\lambda-. \text{undefined}) (\lambda-. m. m)$

lemma $\text{valid-empty-var}[simp]: \langle \text{valid-getter-setter } (\lambda-. (\text{undefined}:::\text{CARD-1})) (\lambda-. m. m) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{register-empty-var}[simp]: \langle \text{register empty-var} \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{getter-empty-var}[simp]: \langle \text{getter empty-var } m = \text{undefined} \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{setter-empty-var}[simp]: \langle \text{setter empty-var } a m = m \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{empty-var-compatible}[simp]: \langle \text{compatible empty-var } X \rangle$ **if** $[simp]: \langle \text{register } X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{empty-var-compatible}'[simp]: \langle \text{register } X \implies \text{compatible } X \text{ empty-var} \rangle$
 $\langle \text{proof} \rangle$

Example $\text{record memory} =$
 $x :: \text{int*int}$
 $y :: \text{nat}$

definition $X = \text{register-from-getter-setter } x (\lambda a b. b(|x:=a|))$
definition $Y = \text{register-from-getter-setter } y (\lambda a b. b(|y:=a|))$

lemma $\text{validX}[simp]: \langle \text{valid-getter-setter } x (\lambda a b. b(|x:=a|)) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{registerX}[simp]: \langle \text{register } X \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{validY}[simp]: \langle \text{valid-getter-setter } y (\lambda a b. b(|y:=a|)) \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{registerY}[simp]: \langle \text{register } Y \rangle$
 $\langle \text{proof} \rangle$

lemma $\text{compatibleXY}[simp]: \langle \text{compatible } X Y \rangle$
 $\langle \text{proof} \rangle$

hide-const $(\text{open}) x y x\text{-update } y\text{-update } X Y$

end

9 Tensor products (finite dimensional)

theory *Finite-Tensor-Product*
imports *Complex-Bounded-Operators.Complex-L2 Misc*

begin

declare *cblinfun.scaleC-right*[simp]

unbundle *cblinfun-notation*

no-notation *m-inv* (*inv1* - [81] 80)

lift-definition *tensor-ell2* :: $\langle 'a::\text{finite ell2} \Rightarrow 'b::\text{finite ell2} \Rightarrow ('a \times 'b) \text{ ell2} \rangle$ (**infixr** \otimes_s 70) **is**
 $\langle \lambda \psi \varphi (i,j). \psi i * \varphi j \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-add2*: $\langle \text{tensor-ell2 } a (b + c) = \text{tensor-ell2 } a b + \text{tensor-ell2 } a c \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-add1*: $\langle \text{tensor-ell2 } (a + b) c = \text{tensor-ell2 } a c + \text{tensor-ell2 } b c \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-scaleC2*: $\langle \text{tensor-ell2 } a (c *_C b) = c *_C \text{tensor-ell2 } a b \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-scaleC1*: $\langle \text{tensor-ell2 } (c *_C a) b = c *_C \text{tensor-ell2 } a b \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-inner-prod*[simp]: $\langle \langle \text{tensor-ell2 } a b, \text{tensor-ell2 } c d \rangle = \langle a, c \rangle * \langle b, d \rangle \rangle$
 $\langle \text{proof} \rangle$

lemma *clinear-tensor-ell21*: *clinear* $(\lambda b. \text{tensor-ell2 } a b)$
 $\langle \text{proof} \rangle$

lemma *clinear-tensor-ell22*: *clinear* $(\lambda a. \text{tensor-ell2 } a b)$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-ket*[simp]: $\text{tensor-ell2 } (\text{ket } i) (\text{ket } j) = \text{ket } (i,j)$
 $\langle \text{proof} \rangle$

definition *tensor-op* :: $\langle ('a \text{ ell2}, 'b::\text{finite ell2}) \text{ cblinfun} \Rightarrow ('c \text{ ell2}, 'd::\text{finite ell2}) \text{ cblinfun} \Rightarrow (('a \times 'c) \text{ ell2}, ('b \times 'd) \text{ ell2}) \text{ cblinfun} \rangle$ (**infixr** \otimes_o 70) **where**
 $\langle \text{tensor-op } M N = (\text{SOME } P. \forall a c. P *_V (\text{ket } (a,c))) \rangle$
 $\langle = \text{tensor-ell2 } (M *_V \text{ket } a) (N *_V \text{ket } c) \rangle$

lemma *tensor-op-ket*:

fixes $a :: \langle 'a::\text{finite} \rangle$ **and** $b :: \langle 'b \rangle$ **and** $c :: \langle 'c::\text{finite} \rangle$ **and** $d :: \langle 'd \rangle$

shows $\langle \text{tensor-op } M N *_V (\text{ket } (a,c)) = \text{tensor-ell2 } (M *_V \text{ket } a) (N *_V \text{ket } c) \rangle$

$\langle \text{proof} \rangle$

lemma *tensor-op-ell2*: $\text{tensor-op } A B *_V \text{tensor-ell2 } \psi \varphi = \text{tensor-ell2 } (A *_V \psi) (B *_V \varphi)$
 $\langle \text{proof} \rangle$

lemma *comp-tensor-op*: $\text{tensor-op } a b \text{ } o_{CL} (\text{tensor-op } c d) = \text{tensor-op } (a \text{ } o_{CL} c) (b \text{ } o_{CL} d)$
for $a :: 'e::\text{finite ell2} \Rightarrow_{CL} 'c::\text{finite ell2}$ **and** $b :: 'f::\text{finite ell2} \Rightarrow_{CL} 'd::\text{finite ell2}$ **and**
 $c :: 'a::\text{finite ell2} \Rightarrow_{CL} 'e \text{ ell2}$ **and** $d :: 'b::\text{finite ell2} \Rightarrow_{CL} 'f \text{ ell2}$
 $\langle \text{proof} \rangle$

lemma *tensor-op-cbilinear*: $\langle \text{cbilinear } (\text{tensor-op } :: 'a::\text{finite ell2} \Rightarrow_{CL} 'b::\text{finite ell2} \Rightarrow 'c::\text{finite ell2} \Rightarrow_{CL} 'd::\text{finite ell2} \Rightarrow -) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-butter*: $\langle \text{tensor-op } (\text{butterket } i j) (\text{butterket } k l) = \text{butterket } (i,k) (j,l) \rangle$
for $i :: -$ **and** $j :: -: \text{finite}$ **and** $k :: -$ **and** $l :: -: \text{finite}$

<proof>

lemma *cspan-tensor-op*: $\langle \text{cspan } \{ \text{tensor-op } (\text{butterket } i \ j) \ (\text{butterket } k \ l) \mid i \ (j::\text{finite}) \ k \ (l::\text{finite}). \ \text{True} \} = \text{UNIV} \rangle$

<proof>

lemma *cindependent-tensor-op*: $\langle \text{cindependent } \{ \text{tensor-op } (\text{butterket } i \ j) \ (\text{butterket } k \ l) \mid i \ (j::\text{finite}) \ k \ (l::\text{finite}). \ \text{True} \} \rangle$

<proof>

lemma *tensor-extensionality*:

fixes $F \ G :: \langle ((\ 'a::\text{finite} \times \ 'b::\text{finite}) \ \text{ell2}) \Rightarrow_{CL} ((\ 'c::\text{finite} \times \ 'd::\text{finite}) \ \text{ell2}) \Rightarrow \ 'e::\text{complex-vector} \rangle$

assumes $[\text{simp}]$: *clinear* F *clinear* G

assumes *tensor-eq*: $(\bigwedge a \ b. \ F \ (\text{tensor-op } a \ b) = G \ (\text{tensor-op } a \ b))$

shows $F = G$

<proof>

lemma *tensor-id* $[\text{simp}]$: $\langle \text{tensor-op } \text{id-cblinfun } \text{id-cblinfun} = \text{id-cblinfun} \rangle$

<proof>

lemma *tensor-op-adjoint*: $\langle (\text{tensor-op } a \ b)^* = \text{tensor-op } (a^*) \ (b^*) \rangle$

<proof>

lemma *tensor-butterfly* $[\text{simp}]$: $\text{tensor-op } (\text{butterfly } \psi \ \psi') \ (\text{butterfly } \varphi \ \varphi') = \text{butterfly } (\text{tensor-ell2 } \psi \ \varphi) \ (\text{tensor-ell2 } \psi' \ \varphi')$

<proof>

definition *tensor-lift* :: $\langle ((\ 'a1::\text{finite } \text{ell2} \Rightarrow_{CL} \ 'a2::\text{finite } \text{ell2}) \Rightarrow (\ 'b1::\text{finite } \text{ell2} \Rightarrow_{CL} \ 'b2::\text{finite } \text{ell2}) \Rightarrow \ 'c) \Rightarrow ((\ 'a1 \times \ 'b1) \ \text{ell2} \Rightarrow_{CL} (\ 'a2 \times \ 'b2) \ \text{ell2}) \Rightarrow \ 'c::\text{complex-vector} \rangle$ **where**
 $\text{tensor-lift } F2 = (\text{SOME } G. \ \text{clinear } G \wedge (\forall a \ b. \ G \ (\text{tensor-op } a \ b) = F2 \ a \ b))$

lemma

fixes $F2 :: \ 'a::\text{finite } \text{ell2} \Rightarrow_{CL} \ 'b::\text{finite } \text{ell2}$

$\Rightarrow \ 'c::\text{finite } \text{ell2} \Rightarrow_{CL} \ 'd::\text{finite } \text{ell2}$

$\Rightarrow \ 'e::\text{complex-normed-vector}$

assumes *cbilinear* $F2$

shows *tensor-lift-clinear*: *clinear* $(\text{tensor-lift } F2)$

and *tensor-lift-correct*: $\langle (\lambda a \ b. \ \text{tensor-lift } F2 \ (\text{tensor-op } a \ b)) = F2 \rangle$

<proof>

lift-definition *assoc-ell20* :: $\langle ((\ 'a::\text{finite} \times \ 'b::\text{finite}) \times \ 'c::\text{finite}) \ \text{ell2} \Rightarrow (\ 'a \times (\ 'b \times \ 'c)) \ \text{ell2} \rangle$ **is**

$\langle \lambda f \ (a, (b, c)). \ f \ ((a, b), c) \rangle$

<proof>

lift-definition *assoc-ell20'* :: $\langle (\ 'a::\text{finite} \times (\ 'b::\text{finite} \times \ 'c::\text{finite})) \ \text{ell2} \Rightarrow ((\ 'a \times \ 'b) \times \ 'c) \ \text{ell2} \rangle$ **is**

$\langle \lambda f \ ((a, b), c). \ f \ (a, (b, c)) \rangle$

<proof>

lift-definition *assoc-ell2* :: $\langle ((\ 'a::\text{finite} \times \ 'b::\text{finite}) \times \ 'c::\text{finite}) \ \text{ell2} \Rightarrow_{CL} (\ 'a \times (\ 'b \times \ 'c)) \ \text{ell2} \rangle$

is *assoc-ell20*

<proof>

lift-definition *assoc-ell2'* :: $\langle (\ 'a::\text{finite} \times (\ 'b::\text{finite} \times \ 'c::\text{finite})) \ \text{ell2} \Rightarrow_{CL} ((\ 'a \times \ 'b) \times \ 'c) \ \text{ell2} \rangle$ **is**

assoc-ell20'

<proof>

lemma *assoc-ell2-tensor*: $\langle \text{assoc-ell2} \ *_{\vee} \ \text{tensor-ell2} \ (\text{tensor-ell2 } a \ b) \ c = \text{tensor-ell2 } a \ (\text{tensor-ell2 } b \ c) \rangle$

<proof>

lemma *assoc-ell2'-tensor*: $\langle \text{assoc-ell2}' \ *_{\vee} \ \text{tensor-ell2} \ a \ (\text{tensor-ell2 } b \ c) = \text{tensor-ell2} \ (\text{tensor-ell2 } a \ b) \ c \rangle$

<proof>

lemma *adjoint-assoc-ell2[simp]*: $\langle \text{assoc-ell2}^* = \text{assoc-ell2}' \rangle$
 $\langle \text{proof} \rangle$

lemma *adjoint-assoc-ell2'[simp]*: $\langle \text{assoc-ell2}'^* = \text{assoc-ell2} \rangle$
 $\langle \text{proof} \rangle$

lift-definition *swap-ell20* :: $\langle ('a::\text{finite} \times 'b::\text{finite}) \text{ ell2} \Rightarrow ('b \times 'a) \text{ ell2} \rangle$ **is**
 $\langle \lambda f (a,b). f (b,a) \rangle$
 $\langle \text{proof} \rangle$

lift-definition *swap-ell2* :: $\langle ('a::\text{finite} \times 'b::\text{finite}) \text{ ell2} \Rightarrow_{CL} ('b \times 'a) \text{ ell2} \rangle$
is *swap-ell20*
 $\langle \text{proof} \rangle$

lemma *swap-ell2-tensor[simp]*: $\langle \text{swap-ell2} *_V \text{ tensor-ell2 } a \ b = \text{tensor-ell2 } b \ a \rangle$
 $\langle \text{proof} \rangle$

lemma *adjoint-swap-ell2[simp]*: $\langle \text{swap-ell2}^* = \text{swap-ell2} \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-extensionality*:
assumes $\langle \bigwedge s \ t. a *_V (s \otimes_s t) = b *_V (s \otimes_s t) \rangle$
shows $a = b$
 $\langle \text{proof} \rangle$

lemma *assoc-ell2'-assoc-ell2[simp]*: $\langle \text{assoc-ell2}' \ o_{CL} \ \text{assoc-ell2} = \text{id-cblinfun} \rangle$
 $\langle \text{proof} \rangle$

lemma *assoc-ell2-assoc-ell2'[simp]*: $\langle \text{assoc-ell2} \ o_{CL} \ \text{assoc-ell2}' = \text{id-cblinfun} \rangle$
 $\langle \text{proof} \rangle$

lemma *unitary-assoc-ell2[simp]*: $\langle \text{unitary } \text{assoc-ell2} \rangle$
 $\langle \text{proof} \rangle$

lemma *unitary-assoc-ell2'[simp]*: $\langle \text{unitary } \text{assoc-ell2}' \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-op-left-add*: $\langle (x + y) \otimes_o b = x \otimes_o b + y \otimes_o b \rangle$
for $x \ y :: \langle 'a::\text{finite } \text{ell2} \Rightarrow_{CL} 'c::\text{finite } \text{ell2} \rangle$ **and** $b :: \langle 'b::\text{finite } \text{ell2} \Rightarrow_{CL} 'd::\text{finite } \text{ell2} \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-op-right-add*: $\langle b \otimes_o (x + y) = b \otimes_o x + b \otimes_o y \rangle$
for $x \ y :: \langle 'a::\text{finite } \text{ell2} \Rightarrow_{CL} 'c::\text{finite } \text{ell2} \rangle$ **and** $b :: \langle 'b::\text{finite } \text{ell2} \Rightarrow_{CL} 'd::\text{finite } \text{ell2} \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-op-scaleC-left*: $\langle (c *_C x) \otimes_o b = c *_C (x \otimes_o b) \rangle$
for $x :: \langle 'a::\text{finite } \text{ell2} \Rightarrow_{CL} 'c::\text{finite } \text{ell2} \rangle$ **and** $b :: \langle 'b::\text{finite } \text{ell2} \Rightarrow_{CL} 'd::\text{finite } \text{ell2} \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-op-scaleC-right*: $\langle b \otimes_o (c *_C x) = c *_C (b \otimes_o x) \rangle$
for $x :: \langle 'a::\text{finite } \text{ell2} \Rightarrow_{CL} 'c::\text{finite } \text{ell2} \rangle$ **and** $b :: \langle 'b::\text{finite } \text{ell2} \Rightarrow_{CL} 'd::\text{finite } \text{ell2} \rangle$
 $\langle \text{proof} \rangle$

lemma *clinear-tensor-left[simp]*: $\langle \text{clinear } (\lambda a. a \otimes_o b :: \text{finite } \text{ell2} \Rightarrow_{CL} \text{finite } \text{ell2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *clinear-tensor-right[simp]*: $\langle \text{clinear } (\lambda b. a \otimes_o b :: \text{finite } \text{ell2} \Rightarrow_{CL} \text{finite } \text{ell2}) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-ell2-nonzero*: $\langle a \otimes_s b \neq 0 \rangle$ **if** $\langle a \neq 0 \rangle$ **and** $\langle b \neq 0 \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-op-nonzero*:

fixes $a :: \langle 'a::\text{finite ell2} \Rightarrow_{CL} 'c::\text{finite ell2} \rangle$ **and** $b :: \langle 'b::\text{finite ell2} \Rightarrow_{CL} 'd::\text{finite ell2} \rangle$

assumes $\langle a \neq 0 \rangle$ **and** $\langle b \neq 0 \rangle$

shows $\langle a \otimes_o b \neq 0 \rangle$

<proof>

lemma *inj-tensor-ell2-left*: $\langle \text{inj } (\lambda a::'a::\text{finite ell2}. a \otimes_s b) \rangle$ **if** $\langle b \neq 0 \rangle$ **for** $b :: \langle 'b::\text{finite ell2} \rangle$

<proof>

lemma *inj-tensor-ell2-right*: $\langle \text{inj } (\lambda b::'b::\text{finite ell2}. a \otimes_s b) \rangle$ **if** $\langle a \neq 0 \rangle$ **for** $a :: \langle 'a::\text{finite ell2} \rangle$

<proof>

lemma *inj-tensor-left*: $\langle \text{inj } (\lambda a::'a::\text{finite ell2} \Rightarrow_{CL} 'c::\text{finite ell2}. a \otimes_o b) \rangle$ **if** $\langle b \neq 0 \rangle$ **for** $b :: \langle 'b::\text{finite ell2} \Rightarrow_{CL} 'd::\text{finite ell2} \rangle$

<proof>

lemma *inj-tensor-right*: $\langle \text{inj } (\lambda b::'b::\text{finite ell2} \Rightarrow_{CL} 'c::\text{finite ell2}. a \otimes_o b) \rangle$ **if** $\langle a \neq 0 \rangle$ **for** $a :: \langle 'a::\text{finite ell2} \Rightarrow_{CL} 'd::\text{finite ell2} \rangle$

<proof>

lemma *tensor-ell2-almost-injective*:

assumes $\langle \text{tensor-ell2 } a \ b = \text{tensor-ell2 } c \ d \rangle$

assumes $\langle a \neq 0 \rangle$

shows $\langle \exists \gamma. b = \gamma *_{\mathcal{C}} d \rangle$

<proof>

lemma *tensor-op-almost-injective*:

fixes $a \ c :: \langle 'a::\text{finite ell2} \Rightarrow_{CL} 'b::\text{finite ell2} \rangle$

and $b \ d :: \langle 'c::\text{finite ell2} \Rightarrow_{CL} 'd::\text{finite ell2} \rangle$

assumes $\langle \text{tensor-op } a \ b = \text{tensor-op } c \ d \rangle$

assumes $\langle a \neq 0 \rangle$

shows $\langle \exists \gamma. b = \gamma *_{\mathcal{C}} d \rangle$

<proof>

lemma *tensor-ell2-0-left[simp]*: $\langle \text{tensor-ell2 } 0 \ x = 0 \rangle$

<proof>

lemma *tensor-ell2-0-right[simp]*: $\langle \text{tensor-ell2 } x \ 0 = 0 \rangle$

<proof>

lemma *tensor-op-0-left[simp]*: $\langle \text{tensor-op } 0 \ x = (0 :: ('a::\text{finite}*'b::\text{finite}) \text{ ell2} \Rightarrow_{CL} ('c::\text{finite}*'d::\text{finite}) \text{ ell2}) \rangle$

<proof>

lemma *tensor-op-0-right[simp]*: $\langle \text{tensor-op } x \ 0 = (0 :: ('a::\text{finite}*'b::\text{finite}) \text{ ell2} \Rightarrow_{CL} ('c::\text{finite}*'d::\text{finite}) \text{ ell2}) \rangle$

<proof>

lemma *bij-tensor-ell2-one-dim-left*:

assumes $\langle \psi \neq 0 \rangle$

shows $\langle \text{bij } (\lambda x::'b::\text{finite ell2}. (\psi :: 'a::\text{CARD-1 ell2}) \otimes_s x) \rangle$

<proof>

lemma *bij-tensor-op-one-dim-left*:

assumes $\langle a \neq 0 \rangle$

shows $\langle \text{bij } (\lambda x::'c::\text{finite ell2} \Rightarrow_{CL} 'd::\text{finite ell2}. (a :: 'a::\{\text{CARD-1,enum}\} \text{ ell2} \Rightarrow_{CL} 'b::\{\text{CARD-1,enum}\} \text{ ell2}) \otimes_o x) \rangle$

<proof>

lemma *swap-ell2-selfinv[simp]*: $\langle \text{swap-ell2 } o_{CL} \ \text{swap-ell2} = \text{id-cblinfun} \rangle$

<proof>

lemma *bij-tensor-op-one-dim-right*:

assumes $\langle b \neq 0 \rangle$

shows $\langle \text{bij } (\lambda x :: 'c :: \text{finite } \text{ell2} \Rightarrow_{CL} 'd :: \text{finite } \text{ell2}. x \otimes_o (b :: 'a :: \{CARD-1, \text{enum}\} \text{ell2} \Rightarrow_{CL} 'b :: \{CARD-1, \text{enum}\} \text{ell2})) \rangle$

(**is** $\langle \text{bij } ?f \rangle$)

<proof>

lemma *overlapping-tensor*:

fixes $a23 :: \langle ('a2 :: \text{finite} * 'a3 :: \text{finite}) \text{ell2} \Rightarrow_{CL} ('b2 :: \text{finite} * 'b3 :: \text{finite}) \text{ell2} \rangle$

and $b12 :: \langle ('a1 :: \text{finite} * 'a2) \text{ell2} \Rightarrow_{CL} ('b1 :: \text{finite} * 'b2) \text{ell2} \rangle$

assumes $\text{eq} : \langle \text{butterfly } \psi \ \psi' \otimes_o a23 = \text{assoc-ell2 } o_{CL} (b12 \otimes_o \text{butterfly } \varphi \ \varphi') o_{CL} \text{assoc-ell2} \rangle$

assumes $\langle \psi \neq 0 \rangle \langle \psi' \neq 0 \rangle \langle \varphi \neq 0 \rangle \langle \varphi' \neq 0 \rangle$

shows $\langle \exists c. \text{butterfly } \psi \ \psi' \otimes_o a23 = \text{butterfly } \psi \ \psi' \otimes_o c \otimes_o \text{butterfly } \varphi \ \varphi' \rangle$

<proof>

lemma *norm-tensor-ell2*: $\langle \text{norm } (a \otimes_s b) = \text{norm } a * \text{norm } b \rangle$

<proof>

lemma *bounded-cbilinear-tensor-ell2*[*bounded-cbilinear*]: $\langle \text{bounded-cbilinear } (\otimes_s) \rangle$

<proof>

end

10 Quantum instantiation of registers

theory *Axioms-Quantum*

imports *Jordan-Normal-Form.Matrix-Impl HOL-Library.Rewrite*

Complex-Bounded-Operators.Complex-L2

Finite-Tensor-Product

begin

unbundle *cblinfun-notation*

no-notation *m-inv* (*inv1* - [81] 80)

type-synonym $'a \text{ update} = \langle ('a \text{ ell2}, 'a \text{ ell2}) \text{cblinfun} \rangle$

lemma *preregister-mult-right*: $\langle \text{clinear } (\lambda a. a \ o_{CL} \ z) \rangle$

<proof>

lemma *preregister-mult-left*: $\langle \text{clinear } (\lambda a. z \ o_{CL} \ a) \rangle$

<proof>

definition *register* :: $\langle ('a :: \text{finite } \text{update} \Rightarrow 'b :: \text{finite } \text{update}) \Rightarrow \text{bool} \rangle$ **where**

$\text{register } F \longleftrightarrow$

$\text{clinear } F$

$\wedge F \ \text{id-cblinfun} = \text{id-cblinfun}$

$\wedge (\forall a \ b. F(a \ o_{CL} \ b) = F \ a \ o_{CL} \ F \ b)$

$\wedge (\forall a. F(a*) = (F \ a)*)$

lemma *register-of-id*: $\langle \text{register } F \Longrightarrow F \ \text{id-cblinfun} = \text{id-cblinfun} \rangle$

<proof>

lemma *register-id*: $\langle \text{register } \text{id} \rangle$

<proof>

lemma *register-preregister*: $\text{register } F \Longrightarrow \text{clinear } F$

<proof>

lemma *register-comp*: $\text{register } F \Longrightarrow \text{register } G \Longrightarrow \text{register } (G \circ F)$

<proof>

lemma *register-mult*: $\text{register } F \implies \text{cblinfun-compose } (F \ a) \ (F \ b) = F \ (\text{cblinfun-compose } a \ b)$
 ⟨proof⟩

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

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

definition *register-pair* ::
 ⟨('a::finite update ⇒ 'c::finite update) ⇒ ('b::finite 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
 tensor-lift (λa b. F a o_{CL} G b) else (λ-. 0))⟩

lemma *cbilinear-F-comp-G[simp]*: ⟨clinear F ⇒ clinear G ⇒ cbilinear (λa b. F a o_{CL} G b)⟩
 ⟨proof⟩

lemma *register-pair-apply*:
assumes [simp]: ⟨register F⟩ ⟨register G⟩
assumes ⟨∧ a b. F a o_{CL} G b = G b o_{CL} F a⟩
shows ⟨register-pair F G⟩ (tensor-op a b) = F a o_{CL} G b⟩
 ⟨proof⟩

lemma *register-pair-is-register*:
fixes F :: ⟨'a::finite update ⇒ 'c::finite update⟩ **and** G
assumes [simp]: ⟨register F⟩ **and** [simp]: ⟨register G⟩
assumes ⟨∧ a b. F a o_{CL} G b = G b o_{CL} F a⟩
shows ⟨register (register-pair F G)⟩
 ⟨proof⟩

end

11 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** ⊗_u 70)
notation *register-pair* (('(-;-'))

11.1 Elementary facts

declare *complex-vector.linear-id*[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]

11.2 Preregisters

lemma *preregister-tensor-left*[simp]: ⟨clinear (λb::'b::finite update. tensor-op a b)⟩
 for a :: ⟨'a::finite update⟩

<proof>

lemma *preregister-tensor-right*[simp]: $\langle \text{clinear } (\lambda a::'a::\text{finite update. tensor-op } a \ b) \rangle$
for $b :: \langle 'b::\text{finite update} \rangle$
<proof>

11.3 Registers

lemma *id-update-tensor-register*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle \text{register } (\lambda a::'a::\text{finite update. id-cblinfun } \otimes_u F \ a) \rangle$
<proof>

lemma *register-tensor-id-update*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle \text{register } (\lambda a::'a::\text{finite update. } F \ a \ \otimes_u \ \text{id-cblinfun}) \rangle$
<proof>

11.4 Tensor product of registers

definition *register-tensor* (**infixr** \otimes_r 70) **where**
 $\text{register-tensor } F \ G = \text{register-pair } (\lambda a. \text{tensor-op } (F \ a) \ \text{id-cblinfun}) \ (\lambda b. \text{tensor-op } \text{id-cblinfun } (G \ b))$

lemma *register-tensor-is-register*:
fixes $F :: 'a::\text{finite update} \Rightarrow 'b::\text{finite update}$ **and** $G :: 'c::\text{finite update} \Rightarrow 'd::\text{finite update}$
shows $\text{register } F \ \Longrightarrow \ \text{register } G \ \Longrightarrow \ \text{register } (F \ \otimes_r \ G)$
<proof>

lemma *register-tensor-apply*[simp]:
fixes $F :: 'a::\text{finite update} \Rightarrow 'b::\text{finite update}$ **and** $G :: 'c::\text{finite update} \Rightarrow 'd::\text{finite update}$
assumes $\langle \text{register } F \rangle$ **and** $\langle \text{register } G \rangle$
shows $(F \ \otimes_r \ G) \ (a \ \otimes_u \ b) = F \ a \ \otimes_u \ G \ b$
<proof>

definition *separating* ($-::'b::\text{finite itself}$) $A \ \longleftrightarrow$
 $(\forall F \ G :: 'a::\text{finite update} \Rightarrow 'b \ \text{update. clinear } F \ \longrightarrow \ \text{clinear } G \ \longrightarrow \ (\forall x \in A. F \ x = G \ x) \ \longrightarrow \ F = G)$

lemma *separating-UNIV*[simp]: $\langle \text{separating } \text{TYPE}(-) \ \text{UNIV} \rangle$
<proof>

lemma *separating-mono*: $\langle A \subseteq B \ \Longrightarrow \ \text{separating } \text{TYPE}('a::\text{finite}) \ A \ \Longrightarrow \ \text{separating } \text{TYPE}('a) \ B \rangle$
<proof>

lemma *register-eqI*: $\langle \text{separating } \text{TYPE}('b::\text{finite}) \ A \ \Longrightarrow \ \text{clinear } F \ \Longrightarrow \ \text{clinear } G \ \Longrightarrow \ (\bigwedge x. x \in A \ \Longrightarrow \ F \ x = G \ x) \ \Longrightarrow \ F = (G::-\Rightarrow 'b \ \text{update}) \rangle$
<proof>

lemma *separating-tensor*:
fixes $A :: \langle 'a::\text{finite update set} \rangle$ **and** $B :: \langle 'b::\text{finite update set} \rangle$
assumes [simp]: $\langle \text{separating } \text{TYPE}('c::\text{finite}) \ A \rangle$
assumes [simp]: $\langle \text{separating } \text{TYPE}('c) \ B \rangle$
shows $\langle \text{separating } \text{TYPE}('c) \ \{a \ \otimes_u \ b \mid a \ b. \ a \in A \ \wedge \ b \in B\} \rangle$
<proof>

lemma *register-tensor-distrib*:
assumes [simp]: $\langle \text{register } F \rangle \ \langle \text{register } G \rangle \ \langle \text{register } H \rangle \ \langle \text{register } L \rangle$
shows $\langle (F \ \otimes_r \ G) \ o \ (H \ \otimes_r \ L) = (F \ o \ H) \ \otimes_r \ (G \ o \ L) \rangle$
<proof>

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

lemma *separating-tensor'*:
fixes $A :: \langle 'a::\text{finite update set} \rangle$ **and** $B :: \langle 'b::\text{finite update set} \rangle$
assumes $\langle \text{separating } \text{TYPE}('c::\text{finite}) \ A \rangle$

assumes $\langle \text{separating TYPE}(c) B \rangle$
assumes $\langle C = \{a \otimes_u b \mid a \in A \wedge b \in B\} \rangle$
shows $\langle \text{separating TYPE}(c) C \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-extensionality3*:

fixes $F G :: \langle ('a::\text{finite} \times 'b::\text{finite} \times 'c::\text{finite}) \text{ update} \Rightarrow 'd::\text{finite update} \rangle$
assumes $[\text{simp}] : \langle \text{register } F \rangle \langle \text{register } G \rangle$
assumes $\bigwedge f g h. F (f \otimes_u g \otimes_u h) = G (f \otimes_u g \otimes_u h)$
shows $F = G$
 $\langle \text{proof} \rangle$

lemma *tensor-extensionality3'*:

fixes $F G :: \langle ('a::\text{finite} \times 'b::\text{finite}) \times 'c::\text{finite} \text{ update} \Rightarrow 'd::\text{finite update} \rangle$
assumes $[\text{simp}] : \langle \text{register } F \rangle \langle \text{register } G \rangle$
assumes $\bigwedge f g h. F ((f \otimes_u g) \otimes_u h) = G ((f \otimes_u g) \otimes_u h)$
shows $F = G$
 $\langle \text{proof} \rangle$

lemma *register-tensor-id* $[\text{simp}]$: $\langle \text{id} \otimes_r \text{id} = \text{id} \rangle$
 $\langle \text{proof} \rangle$

11.5 Pairs and compatibility

definition *compatible* $:: \langle ('a::\text{finite update} \Rightarrow 'c::\text{finite update})$
 $\Rightarrow ('b::\text{finite update} \Rightarrow 'c \text{ update}) \Rightarrow \text{bool} \rangle$ **where**
 $\langle \text{compatible } F G \iff \text{register } F \wedge \text{register } G \wedge (\forall a b. F a *_u G b = G b *_u F a) \rangle$

lemma *compatibleI*:

assumes *register F and register G*
assumes $\langle \bigwedge a b. (F a) *_u (G b) = (G b) *_u (F a) \rangle$
shows *compatible F G*
 $\langle \text{proof} \rangle$

lemma *swap-registers*:

assumes *compatible R S*
shows $R a *_u S b = S b *_u R a$
 $\langle \text{proof} \rangle$

lemma *compatible-sym*: *compatible x y \implies compatible y x*
 $\langle \text{proof} \rangle$

lemma *pair-is-register* $[\text{simp}]$:

assumes *compatible F G*
shows *register (F; G)*
 $\langle \text{proof} \rangle$

lemma *register-pair-apply*:

assumes $\langle \text{compatible } F G \rangle$
shows $\langle (F; G) (a \otimes_u b) = (F a) *_u (G b) \rangle$
 $\langle \text{proof} \rangle$

lemma *register-pair-apply'*:

assumes $\langle \text{compatible } F G \rangle$
shows $\langle (F; G) (a \otimes_u b) = (G b) *_u (F a) \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-comp-left* $[\text{simp}]$: *compatible F G \implies register H \implies compatible (F o H) G*
 $\langle \text{proof} \rangle$

lemma *compatible-comp-right* $[\text{simp}]$: *compatible F G \implies register H \implies compatible F (G o H)*

<proof>

lemma *compatible-comp-inner[simp]*:
compatible $F\ G \implies$ *register* $H \implies$ *compatible* $(H \circ F)\ (H \circ G)$
<proof>

lemma *compatible-register1*: *<compatible* $F\ G \implies$ *register* F *>*
<proof>

lemma *compatible-register2*: *<compatible* $F\ G \implies$ *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 $F :: 'a::\text{finite_update} \Rightarrow 'c::\text{finite_update}$ **and** $G :: 'b::\text{finite_update} \Rightarrow 'c::\text{finite_update}$
assumes *compatible* $F\ G$
shows *compatible* $(\lambda a.\ \text{id-cblinfun} \otimes_u F\ a)\ (\lambda a.\ \text{id-cblinfun} \otimes_u G\ a)$
<proof>

lemma *compatible-tensor-id-update-right[simp]*:
fixes $F :: 'a::\text{finite_update} \Rightarrow 'c::\text{finite_update}$ **and** $G :: 'b::\text{finite_update} \Rightarrow 'c::\text{finite_update}$
assumes *compatible* $F\ G$
shows *compatible* $(\lambda a.\ F\ a \otimes_u \text{id-cblinfun})\ (\lambda a.\ G\ a \otimes_u \text{id-cblinfun})$
<proof>

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

lemma *compatible-tensor-id-update-lr[simp]*:
assumes *register* F **and** *register* G
shows *compatible* $(\lambda a.\ \text{id-cblinfun} \otimes_u F\ a)\ (\lambda a.\ G\ a \otimes_u \text{id-cblinfun})$
<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 $R\ a *_{\text{u}} S\ b *_{\text{u}} c = S\ b *_{\text{u}} R\ a *_{\text{u}} c$
<proof>

lemma *swap-registers-right*:
assumes *compatible* $R\ S$
shows $c *_{\text{u}} R\ a *_{\text{u}} S\ b = c *_{\text{u}} S\ b *_{\text{u}} R\ a$
<proof>

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

11.6 Fst and Snd

definition *Fst* **where** *<Fst* $a = a \otimes_u \text{id-cblinfun}$ *>*

definition *Snd* **where** *<Snd* $a = \text{id-cblinfun} \otimes_u a$ *>*

lemma *register-Fst[simp]*: *<register* Fst *>*
<proof>

lemma *register-Snd*[simp]: $\langle \text{register Snd} \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible-Fst-Snd*[simp]: $\langle \text{compatible Fst Snd} \rangle$
 $\langle \text{proof} \rangle$

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

definition $\langle \text{swap} = (\text{Snd}; \text{Fst}) \rangle$

lemma *swap-apply*[simp]: $\text{swap } (a \otimes_u b) = (b \otimes_u a)$
 $\langle \text{proof} \rangle$

lemma *swap-o-Fst*: $\text{swap } o \text{ Fst} = \text{Snd}$
 $\langle \text{proof} \rangle$

lemma *swap-o-Snd*: $\text{swap } o \text{ Snd} = \text{Fst}$
 $\langle \text{proof} \rangle$

lemma *register-swap*[simp]: $\langle \text{register swap} \rangle$
 $\langle \text{proof} \rangle$

lemma *pair-Fst-Snd*: $\langle (\text{Fst}; \text{Snd}) = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *swap-o-swap*[simp]: $\langle \text{swap } o \text{ swap} = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *swap-swap*[simp]: $\langle \text{swap } (\text{swap } x) = x \rangle$
 $\langle \text{proof} \rangle$

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

lemma *register-pair-Fst*:
assumes $\langle \text{compatible } F \ G \rangle$
shows $\langle (F; G) o \text{Fst} = F \rangle$
 $\langle \text{proof} \rangle$

lemma *register-pair-Snd*:
assumes $\langle \text{compatible } F \ G \rangle$
shows $\langle (F; G) o \text{Snd} = G \rangle$
 $\langle \text{proof} \rangle$

lemma *register-Fst-register-Snd*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle (F o \text{Fst}; F o \text{Snd}) = F \rangle$
 $\langle \text{proof} \rangle$

lemma *register-Snd-register-Fst*[simp]:
assumes $\langle \text{register } F \rangle$
shows $\langle (F o \text{Snd}; F o \text{Fst}) = F o \text{swap} \rangle$
 $\langle \text{proof} \rangle$

lemma *compatible3*[simp]:
assumes [simp]: *compatible* $F \ G$ **and** *compatible* $G \ H$ **and** *compatible* $F \ H$
shows *compatible* $(F; G) \ H$
 $\langle \text{proof} \rangle$

lemma *compatible3'*[simp]:
assumes *compatible* $F \ G$ **and** *compatible* $G \ H$ **and** *compatible* $F \ H$
shows *compatible* $F \ (G; H)$
 $\langle \text{proof} \rangle$

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

11.7 Compatibility of register tensor products

lemma *compatible-register-tensor*:
fixes $F :: \langle 'a::\text{finite update} \Rightarrow 'e::\text{finite update} \rangle$ **and** $G :: \langle 'b::\text{finite update} \Rightarrow 'f::\text{finite update} \rangle$
and $F' :: \langle 'c::\text{finite update} \Rightarrow 'e \text{ update} \rangle$ **and** $G' :: \langle 'd::\text{finite update} \Rightarrow 'f \text{ update} \rangle$
assumes [simp]: *compatible F F'*
assumes [simp]: *compatible G G'*
shows $\langle \text{compatible } (F \otimes_r G) (F' \otimes_r G') \rangle$
 ⟨*proof*⟩

11.8 Associativity of the tensor product

definition *assoc* :: $\langle ('a::\text{finite} \times 'b::\text{finite}) \times 'c::\text{finite} \rangle \text{ update} \Rightarrow ('a \times ('b \times 'c)) \text{ update}$ **where**
 $\langle \text{assoc} = ((Fst; Snd \circ Fst); Snd \circ Snd) \rangle$

lemma *assoc-is-hom*[simp]: *cllinear assoc*
 ⟨*proof*⟩

lemma *assoc-apply*[simp]: $\langle \text{assoc } ((a \otimes_u b) \otimes_u c) = (a \otimes_u (b \otimes_u c)) \rangle$
 ⟨*proof*⟩

definition *assoc'* :: $\langle ('a \times ('b \times 'c)) \text{ update} \Rightarrow (('a::\text{finite} \times 'b::\text{finite}) \times 'c::\text{finite}) \text{ update} \rangle$ **where**
 $\langle \text{assoc}' = (Fst \circ Fst; (Fst \circ Snd; Snd)) \rangle$

lemma *assoc'-is-hom*[simp]: *cllinear assoc'*
 ⟨*proof*⟩

lemma *assoc'-apply*[simp]: $\langle \text{assoc}' (a \otimes_u (b \otimes_u c)) = ((a \otimes_u b) \otimes_u c) \rangle$
 ⟨*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 $\langle (F; (G; H)) \circ \text{assoc} = ((F; G); H) \rangle$
 ⟨*proof*⟩

lemma *pair-o-assoc'*[simp]:
assumes [simp]: *compatible F G* *compatible G H* *compatible F H*
shows $\langle ((F; G); H) \circ \text{assoc}' = (F; (G; H)) \rangle$
 ⟨*proof*⟩

lemma *assoc'-o-assoc*[simp]: $\langle \text{assoc}' \circ \text{assoc} = \text{id} \rangle$
 ⟨*proof*⟩

lemma *assoc'-assoc*[simp]: $\langle \text{assoc}' (\text{assoc } x) = x \rangle$
 ⟨*proof*⟩

lemma *assoc-o-assoc'*[simp]: $\langle \text{assoc} \circ \text{assoc}' = \text{id} \rangle$
 ⟨*proof*⟩

lemma *assoc-assoc'*[simp]: $\langle \text{assoc} (\text{assoc}' x) = x \rangle$
 ⟨*proof*⟩

lemma *inv-assoc*[simp]: $\langle \text{inv assoc} = \text{assoc}' \rangle$
 $\langle \text{proof} \rangle$

lemma *inv-assoc'*[simp]: $\langle \text{inv assoc}' = \text{assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma [simp]: $\langle \text{bij assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma [simp]: $\langle \text{bij assoc}' \rangle$
 $\langle \text{proof} \rangle$

11.9 Iso-registers

definition $\langle \text{iso-register } F \longleftrightarrow \text{register } F \wedge (\exists G. \text{register } G \wedge F \circ G = \text{id} \wedge G \circ F = \text{id}) \rangle$
for $F :: \langle \text{finite update} \Rightarrow \text{finite update} \rangle$

lemma *iso-registerI*:
assumes $\langle \text{register } G \rangle \langle F \circ G = \text{id} \rangle \langle G \circ F = \text{id} \rangle$
shows $\langle \text{iso-register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-inv*: $\langle \text{iso-register } F \Longrightarrow \text{iso-register } (\text{inv } F) \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-inv-comp1*: $\langle \text{iso-register } F \Longrightarrow \text{inv } F \circ F = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-inv-comp2*: $\langle \text{iso-register } F \Longrightarrow F \circ \text{inv } F = \text{id} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-id*[simp]: $\langle \text{iso-register id} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-is-register*: $\langle \text{iso-register } F \Longrightarrow \text{register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-comp*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{iso-register } (F \circ G) \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-tensor-is-iso-register*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{iso-register } (F \otimes_r G) \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-bij*: $\langle \text{iso-register } F \Longrightarrow \text{bij } F \rangle$
 $\langle \text{proof} \rangle$

lemma *inv-register-tensor*[simp]:
assumes [simp]: $\langle \text{iso-register } F \rangle \langle \text{iso-register } G \rangle$
shows $\langle \text{inv } (F \otimes_r G) = \text{inv } F \otimes_r \text{inv } G \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-swap*[simp]: $\langle \text{iso-register swap} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-assoc*[simp]: $\langle \text{iso-register assoc} \rangle$
 $\langle \text{proof} \rangle$

lemma *iso-register-assoc*[simp]: $\langle \text{iso-register assoc} \rangle$
 $\langle \text{proof} \rangle$

definition $\langle \text{equivalent-registers } F \ G \longleftrightarrow (\text{register } F \wedge (\exists I. \text{iso-register } I \wedge F \circ I = G)) \rangle$
for $F \ G :: \langle \text{--::finite update} \Rightarrow \text{--::finite update} \rangle$

lemma *iso-register-equivalent-id*[simp]: $\langle \text{equivalent-registers id } F \longleftrightarrow \text{iso-register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registersI*:
assumes $\langle \text{register } F \rangle$
assumes $\langle \text{iso-register } I \rangle$
assumes $\langle F \circ I = G \rangle$
shows $\langle \text{equivalent-registers } F \ G \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-register-left*: $\langle \text{equivalent-registers } F \ G \Longrightarrow \text{register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-register-right*: $\langle \text{register } G \rangle$ **if** $\langle \text{equivalent-registers } F \ G \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-sym*:
assumes $\langle \text{equivalent-registers } F \ G \rangle$
shows $\langle \text{equivalent-registers } G \ F \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-trans*[trans]:
assumes $\langle \text{equivalent-registers } F \ G \rangle$ **and** $\langle \text{equivalent-registers } G \ H \rangle$
shows $\langle \text{equivalent-registers } F \ H \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-assoc*[simp]:
assumes [simp]: $\langle \text{compatible } F \ G \rangle$ $\langle \text{compatible } F \ H \rangle$ $\langle \text{compatible } G \ H \rangle$
shows $\langle \text{equivalent-registers } (F;(G;H)) ((F;G);H) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-pair-right*:
assumes [simp]: $\langle \text{compatible } F \ G \rangle$
assumes *eq*: $\langle \text{equivalent-registers } G \ H \rangle$
shows $\langle \text{equivalent-registers } (F;G) (F;H) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-pair-left*:
assumes [simp]: $\langle \text{compatible } F \ G \rangle$
assumes *eq*: $\langle \text{equivalent-registers } F \ H \rangle$
shows $\langle \text{equivalent-registers } (F;G) (H;G) \rangle$
 $\langle \text{proof} \rangle$

lemma *equivalent-registers-comp*:
assumes $\langle \text{register } H \rangle$
assumes $\langle \text{equivalent-registers } F \ G \rangle$
shows $\langle \text{equivalent-registers } (H \circ F) (H \circ G) \rangle$
 $\langle \text{proof} \rangle$

11.10 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 .

$\langle ML \rangle$

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>

11.11 Notation

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

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

bundle *no-register-notation* **begin**
no-notation *register-tensor* (**infixr** \otimes_r 70)
no-notation *register-pair* ('(-;-)')
end

end

12 Quantum mechanics basics

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

type-synonym ('a,'b) *matrix* = (<'a *ell2*, 'b *ell2*> *cblinfun*>

12.1 Basic quantum states

12.1.1 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>

12.1.2 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>

12.2 Basic quantum gates

12.2.1 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 [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>

12.2.2 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 [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>

12.2.3 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 [simp, code]: *mat-of-cblinfun hadamard = matrix-hadamard*
<proof>

lemma *hada-adj*[simp]: *hadamard* = hadamard*
<proof>

12.2.4 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 [simp, code]: *mat-of-cblinfun CNOT = matrix-CNOT*
<proof>

lemma [simp]: *CNOT* = CNOT*
<proof>

lemma *cnot-apply*[simp]: *<CNOT *_V ket (i, j) = ket (i, j+i)>*
<proof>

12.2.5 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* ⊗_{*s*} *t* = *t* ⊗_{*s*} *s*⟩
⟨*proof*⟩

end

13 Derived facts about quantum registers

theory *Quantum-Extra*

imports

Laws-Quantum

Quantum

begin

no-notation *meet* (**infixl** \sqcap 70)

no-notation *Group.mult* (**infixl** \otimes 70)

no-notation *Order.top* (\top 1)

unbundle *register-notation*

unbundle *cblinfun-notation*

lemma *zero-not-register[simp]*: ⟨ \sim *register* (λ . 0)⟩
⟨*proof*⟩

lemma *register-pair-is-register-converse*:
⟨*register* (*F*; *G*) \implies *register F*⟩ ⟨*register* (*F*; *G*) \implies *register G*⟩
⟨*proof*⟩

lemma *register-id'[simp]*: ⟨*register* ($\lambda x. x$)⟩
⟨*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*⟩

lemma *compatible-proj-intersect*:

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

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

lemma unitary-sandwich-register: $\langle unitary a \implies register (sandwich a) \rangle$
 $\langle proof \rangle$

lemma sandwich-tensor:
fixes $a :: \langle 'a::finite ell2 \Rightarrow_{CL} 'a ell2 \rangle$ **and** $b :: \langle 'b::finite ell2 \Rightarrow_{CL} 'b ell2 \rangle$
assumes $\langle unitary a \rangle \langle unitary b \rangle$
shows $sandwich (a \otimes_o b) = sandwich a \otimes_r sandwich b$
 $\langle proof \rangle$

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

lemma register-sandwich: $\langle register F \implies F (sandwich a b) = sandwich (F a) (F b) \rangle$
 $\langle proof \rangle$

lemma assoc-ell2-sandwich: $\langle assoc = sandwich assoc-ell2 \rangle$
 $\langle proof \rangle$

lemma assoc-ell2'-sandwich: $\langle assoc' = sandwich assoc-ell2' \rangle$
 $\langle proof \rangle$

lemma swap-sandwich: $swap = sandwich Uswap$
 $\langle proof \rangle$

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

lemma compatible-selfbutter-join:
assumes $[register]: compatible R S$
shows $R (selfbutter \psi) \circ_{CL} S (selfbutter \varphi) = (R; S) (selfbutter (\psi \otimes_s \varphi))$
 $\langle proof \rangle$

lemma register-mult':
assumes $\langle register F \rangle$
shows $\langle F a *_V F b *_V c = F (a \circ_{CL} b) *_V c \rangle$
 $\langle proof \rangle$

lemma register-scaleC:
assumes $\langle register F \rangle$ **shows** $\langle F (c *_C a) = c *_C F a \rangle$
 $\langle proof \rangle$

lemma register-bounded-clinear: $\langle register F \implies bounded-clinear F \rangle$
 $\langle proof \rangle$

lemma register-adjoint: $F (a*) = (F a)*$ **if** $\langle register F \rangle$
 $\langle proof \rangle$

end

14 Very simple Quantum Hoare logic

```

theory QHoare
  imports Quantum-Extra
begin

no-notation Order.top ( $\top$ )

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

definition apply  $U R = R U$  for  $R :: \langle 'a \text{ update} \Rightarrow 'mem \text{ update} \rangle$ 
definition ifthen  $R x = R (\text{butterket } x)$  for  $R :: \langle 'a \text{ update} \Rightarrow 'mem \text{ update} \rangle$ 
definition program  $S = \text{fold } (o_{CL}) S \text{ id-cblinfun}$  for  $S :: \langle 'mem \text{ update list} \rangle$ 

definition hoare ::  $\langle 'mem \text{ ell2 ccspace} \Rightarrow ('mem \text{ ell2} \Rightarrow_{CL} 'mem \text{ ell2}) \text{ list} \Rightarrow 'mem \text{ ell2 ccspace} \Rightarrow \text{bool} \rangle$ 
where
  hoare  $C p D \longleftrightarrow (\forall \psi \in \text{space-as-set } C. \text{program } p *_{\vee} \psi \in \text{space-as-set } D)$  for  $C p D$ 

definition EQ ::  $( 'a \text{ update} \Rightarrow 'mem \text{ update} ) \Rightarrow 'a \text{ ell2} \Rightarrow 'mem \text{ ell2 ccspace}$  (infix  $=_q$  75) where
  EQ  $R \psi = R (\text{selfbutter } \psi) *_{\mathcal{S}} \top$ 

lemma program-skip[simp]:  $\text{program } [] = \text{id-cblinfun}$ 
  <proof>

lemma program-seq:  $\text{program } (p1 @ p2) = \text{program } p2 \ o_{CL} \ \text{program } p1$ 
  <proof>

lemma hoare-seq[trans]:  $\text{hoare } C p1 D \Longrightarrow \text{hoare } D p2 E \Longrightarrow \text{hoare } C (p1 @ p2) E$ 
  <proof>

lemma hoare-weaken-left[trans]:  $\langle A \leq B \Longrightarrow \text{hoare } B p C \Longrightarrow \text{hoare } A p C \rangle$ 
  <proof>

lemma hoare-weaken-right[trans]:  $\langle \text{hoare } A p B \Longrightarrow B \leq C \Longrightarrow \text{hoare } A p C \rangle$ 
  <proof>

lemma hoare-skip:  $C \leq D \Longrightarrow \text{hoare } C [] D$ 
  <proof>

lemma hoare-apply:
  assumes  $R U *_{\mathcal{S}} \text{pre} \leq \text{post}$ 
  shows  $\text{hoare } \text{pre} [\text{apply } U R] \text{post}$ 
  <proof>

lemma hoare-ifthen:
  fixes  $R :: \langle 'a \text{ update} \Rightarrow 'mem \text{ update} \rangle$ 
  assumes  $R (\text{selfbutter } (\text{ket } x)) *_{\mathcal{S}} \text{pre} \leq \text{post}$ 
  shows  $\text{hoare } \text{pre} [\text{ifthen } R x] \text{post}$ 
  <proof>

end

end

```

15 Tensor products as matrices

```

theory Finite-Tensor-Product-Matrices
  imports Finite-Tensor-Product
begin

```

definition *tensor-pack* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \times \text{nat}) \Rightarrow \text{nat}$
where *tensor-pack* $X Y = (\lambda(x, y). x * Y + y)$

definition *tensor-unpack* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \times \text{nat})$
where *tensor-unpack* $X Y xy = (xy \text{ div } Y, xy \text{ mod } Y)$

lemma *tensor-unpack-inj*:
assumes $i < A * B$ **and** $j < A * B$
shows *tensor-unpack* $A B i = \text{tensor-unpack } A B j \longleftrightarrow i = j$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-bound1[simp]*: $i < A * B \implies \text{fst } (\text{tensor-unpack } A B i) < A$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-bound2[simp]*: $i < A * B \implies \text{snd } (\text{tensor-unpack } A B i) < B$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-fstfst*: $\langle \text{fst } (\text{tensor-unpack } A B (\text{fst } (\text{tensor-unpack } (A * B) C i)))$
 $= \text{fst } (\text{tensor-unpack } A (B * C) i) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-sndsnd*: $\langle \text{snd } (\text{tensor-unpack } B C (\text{snd } (\text{tensor-unpack } A (B * C) i)))$
 $= \text{snd } (\text{tensor-unpack } (A * B) C i) \rangle$
 $\langle \text{proof} \rangle$

lemma *tensor-unpack-fstsnd*: $\langle \text{fst } (\text{tensor-unpack } B C (\text{snd } (\text{tensor-unpack } A (B * C) i)))$
 $= \text{snd } (\text{tensor-unpack } A B (\text{fst } (\text{tensor-unpack } (A * B) C i))) \rangle$
 $\langle \text{proof} \rangle$

definition *tensor-state-jnf* $\psi \varphi = (\text{let } d1 = \text{dim-vec } \psi \text{ in let } d2 = \text{dim-vec } \varphi \text{ in}$
 $\text{vec } (d1 * d2) (\lambda i. \text{let } (i1, i2) = \text{tensor-unpack } d1 d2 i \text{ in } (\text{vec-index } \psi i1) * (\text{vec-index } \varphi i2)))$

lemma *tensor-state-jnf-dim[simp]*: $\langle \text{dim-vec } (\text{tensor-state-jnf } \psi \varphi) = \text{dim-vec } \psi * \text{dim-vec } \varphi \rangle$
 $\langle \text{proof} \rangle$

lemma *enum-prod-nth-tensor-unpack*:
assumes $\langle i < \text{CARD}('a) * \text{CARD}('b) \rangle$
shows $(\text{Enum.enum } ! i :: 'a :: \text{enum} \times 'b :: \text{enum}) =$
 $(\text{let } (i1, i2) = \text{tensor-unpack } \text{CARD}('a) \text{CARD}('b) i \text{ in}$
 $(\text{Enum.enum } ! i1, \text{Enum.enum } ! i2))$
 $\langle \text{proof} \rangle$

lemma *vec-of-basis-enum-tensor-state-index*:
fixes $\psi :: \langle 'a :: \text{enum ell2} \rangle$ **and** $\varphi :: \langle 'b :: \text{enum ell2} \rangle$
assumes [simp]: $\langle i < \text{CARD}('a) * \text{CARD}('b) \rangle$
shows $\langle \text{vec-of-basis-enum } (\psi \otimes_s \varphi) \$ i = (\text{let } (i1, i2) = \text{tensor-unpack } \text{CARD}('a) \text{CARD}('b) i \text{ in}$
 $\text{vec-of-basis-enum } \psi \$ i1 * \text{vec-of-basis-enum } \varphi \$ i2) \rangle$
 $\langle \text{proof} \rangle$

lemma *vec-of-basis-enum-tensor-state*:
fixes $\psi :: \langle 'a :: \text{enum ell2} \rangle$ **and** $\varphi :: \langle 'b :: \text{enum ell2} \rangle$
shows $\langle \text{vec-of-basis-enum } (\psi \otimes_s \varphi) = \text{tensor-state-jnf } (\text{vec-of-basis-enum } \psi) (\text{vec-of-basis-enum } \varphi) \rangle$
 $\langle \text{proof} \rangle$

lemma *mat-of-cblinfun-tensor-op-index*:
fixes $a :: \langle 'a :: \text{enum ell2} \Rightarrow_{CL} 'b :: \text{enum ell2} \rangle$ **and** $b :: \langle 'c :: \text{enum ell2} \Rightarrow_{CL} 'd :: \text{enum ell2} \rangle$
assumes [simp]: $\langle i < \text{CARD}('b) * \text{CARD}('d) \rangle$
assumes [simp]: $\langle j < \text{CARD}('a) * \text{CARD}('c) \rangle$
shows $\langle \text{mat-of-cblinfun } (\text{tensor-op } a b) \$\$ (i, j) =$
 $(\text{let } (i1, i2) = \text{tensor-unpack } \text{CARD}('b) \text{CARD}('d) i \text{ in}$
 $\text{let } (j1, j2) = \text{tensor-unpack } \text{CARD}('a) \text{CARD}('c) j \text{ in}$
 $\text{mat-of-cblinfun } a \$\$ (i1, j1) * \text{mat-of-cblinfun } b \$\$ (i2, j2)) \rangle$
 $\langle \text{proof} \rangle$

definition *tensor-op-jnf* $A B =$
 (let $r1 = \text{dim-row } A$ in
 let $c1 = \text{dim-col } A$ in
 let $r2 = \text{dim-row } B$ in
 let $c2 = \text{dim-col } B$ in
 mat ($r1*r2$) ($c1*c2$)
 ($\lambda(i,j).$ let ($i1,i2$) = *tensor-unpack* $r1 r2 i$ in
 let ($j1,j2$) = *tensor-unpack* $c1 c2 j$ in
 ($A \text{ \&\amp; } (i1,j1) * (B \text{ \&\amp; } (i2,j2))$)))

lemma *tensor-op-jnf-dim[simp]*:
 $\langle \text{dim-row } (\text{tensor-op-jnf } a b) = \text{dim-row } a * \text{dim-row } b \rangle$
 $\langle \text{dim-col } (\text{tensor-op-jnf } a b) = \text{dim-col } a * \text{dim-col } b \rangle$
 $\langle \text{proof} \rangle$

lemma *mat-of-cblinfun-tensor-op*:
fixes $a :: \langle 'a::\text{enum } ell2 \Rightarrow_{CL} 'b::\text{enum } ell2 \rangle$ **and** $b :: \langle 'c::\text{enum } ell2 \Rightarrow_{CL} 'd::\text{enum } ell2 \rangle$
shows $\langle \text{mat-of-cblinfun } (\text{tensor-op } a b) = \text{tensor-op-jnf } (\text{mat-of-cblinfun } a) (\text{mat-of-cblinfun } b) \rangle$
 $\langle \text{proof} \rangle$

lemma *mat-of-cblinfun-assoc-ell2'[simp]*:
 $\langle \text{mat-of-cblinfun } (\text{assoc-ell2}' :: (('a::\text{enum} \times 'b::\text{enum} \times 'c::\text{enum})) ell2 \Rightarrow_{CL} -) = \text{one-mat } (CARD('a)*CARD('b)*CARD('c)) \rangle$
(is *mat-of-cblinfun ?assoc = -*
 $\langle \text{proof} \rangle$

lemma *assoc-ell2'-inv*: $\text{assoc-ell2 } o_{CL} \text{ assoc-ell2}' = \text{id-cblinfun}$
 $\langle \text{proof} \rangle$

lemma *assoc-ell2-inv*: $\text{assoc-ell2}' o_{CL} \text{ assoc-ell2} = \text{id-cblinfun}$
 $\langle \text{proof} \rangle$

lemma *mat-of-cblinfun-assoc-ell2[simp]*:
 $\langle \text{mat-of-cblinfun } (\text{assoc-ell2} :: (('a::\text{enum} \times 'b::\text{enum}) \times 'c::\text{enum}) ell2 \Rightarrow_{CL} -) = \text{one-mat } (CARD('a)*CARD('b)*CARD('c)) \rangle$
(is *mat-of-cblinfun ?assoc = -*
 $\langle \text{proof} \rangle$

end

16 Quantum teleportation

theory *Teleport*

imports

QHoare
Real-Impl.Real-Impl
HOL-Library.Code-Target-Numeral
Finite-Tensor-Product-Matrices
HOL-Library.Word

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** \otimes_1 70)
no-notation *Order.top* (\top_1)
unbundle *no-vec-syntax*
unbundle *no-inner-syntax*


```

locale teleport-locale = qhoare TYPE('mem::finite) +
  fixes X :: bit update  $\Rightarrow$  'mem::finite update
    and  $\Phi$  :: (bit*bit) update  $\Rightarrow$  'mem update
    and A :: 'atype::finite update  $\Rightarrow$  'mem update
    and B :: 'btype::finite update  $\Rightarrow$  'mem update
  assumes compat[register]: mutually compatible (X, $\Phi$ ,A,B)
begin

abbreviation  $\Phi 1 \equiv \Phi \circ Fst$ 
abbreviation  $\Phi 2 \equiv \Phi \circ Snd$ 
abbreviation  $X\Phi 2 \equiv (X;\Phi 2)$ 
abbreviation  $X\Phi 1 \equiv (X;\Phi 1)$ 
abbreviation  $X\Phi \equiv (X;\Phi)$ 
abbreviation  $XAB \equiv ((X;A); B)$ 
abbreviation  $AB \equiv (A;B)$ 
abbreviation  $\Phi 2AB \equiv ((\Phi \circ Snd; A); B)$ 

definition teleport a b = [
  apply CNOT X $\Phi 1$ ,
  apply hadamard X,
  ifthen  $\Phi 1$  a,
  ifthen X b,
  apply (if a=1 then pauliX else id-cblinfun)  $\Phi 2$ ,
  apply (if b=1 then pauliZ else id-cblinfun)  $\Phi 2$ 
]

lemma  $\Phi$ -X $\Phi$ :  $\langle \Phi a = X\Phi (id-cblinfun \otimes_o a) \rangle$ 
  <proof>
lemma X $\Phi 1$ -X $\Phi$ :  $\langle X\Phi 1 a = X\Phi (assoc (a \otimes_o id-cblinfun)) \rangle$ 
  <proof>
lemma X $\Phi 2$ -X $\Phi$ :  $\langle X\Phi 2 a = X\Phi ((id \otimes_r swap) (assoc (a \otimes_o id-cblinfun))) \rangle$ 
  <proof>
lemma  $\Phi 2$ -X $\Phi$ :  $\langle \Phi 2 a = X\Phi (id-cblinfun \otimes_o (id-cblinfun \otimes_o a)) \rangle$ 
  <proof>
lemma X-X $\Phi$ :  $\langle X a = X\Phi (a \otimes_o id-cblinfun) \rangle$ 
  <proof>
lemma  $\Phi 1$ -X $\Phi$ :  $\langle \Phi 1 a = X\Phi (id-cblinfun \otimes_o (a \otimes_o id-cblinfun)) \rangle$ 
  <proof>
lemmas to-X $\Phi = \Phi$ -X $\Phi$  X $\Phi 1$ -X $\Phi$  X $\Phi 2$ -X $\Phi$   $\Phi 2$ -X $\Phi$  X-X $\Phi$   $\Phi 1$ -X $\Phi$ 

lemma X-X $\Phi 1$ :  $\langle X a = X\Phi 1 (a \otimes_o id-cblinfun) \rangle$ 
  <proof>
lemmas to-X $\Phi 1 = X$ -X $\Phi 1$ 

lemma XAB-to-X $\Phi 2$ -AB:  $\langle XAB a = (X\Phi 2;AB) ((swap \otimes_r id) (assoc' (id-cblinfun \otimes_o assoc a))) \rangle$ 
  <proof>

lemma X $\Phi 2$ -to-X $\Phi 2$ -AB:  $\langle X\Phi 2 a = (X\Phi 2;AB) (a \otimes_o id-cblinfun) \rangle$ 
  <proof>

schematic-goal  $\Phi 2AB$ -to-X $\Phi 2$ -AB:  $\Phi 2AB a = (X\Phi 2;AB) ?b$ 
  <proof>

lemmas to-X $\Phi 2$ -AB = XAB-to-X $\Phi 2$ -AB X $\Phi 2$ -to-X $\Phi 2$ -AB  $\Phi 2AB$ -to-X $\Phi 2$ -AB

lemma teleport:
  assumes [simp]: norm  $\psi = 1$ 
  shows hoare (XAB =q  $\psi \sqcap \Phi =_q \beta 00$ ) (teleport a b) ( $\Phi 2AB =_q \psi$ )
  <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* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun*⟩

definition *X* :: ⟨*bit var*⟩ where ⟨*X a* = *id-cblinfun* ⊗_o *a* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun*⟩

definition Φ1 :: ⟨*bit var*⟩ where ⟨Φ1 *a* = *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *a* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun*⟩

definition *B* :: ⟨*b-state var*⟩ where ⟨*B a* = *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *a* ⊗_o *id-cblinfun*⟩

definition Φ2 :: ⟨*bit var*⟩ where ⟨Φ2 *a* = *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun* ⊗_o *id-cblinfun* ⊗_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*

end

17 Quantum instantiation of complements

theory *Axioms-Complement-Quantum*

imports *Laws-Quantum Finite-Tensor-Product Quantum-Extra*

begin

no-notation *m-inv* (*inv1* - [81] 80)

no-notation *Lattice.join* (infixl \sqcup_1 65)

typedef ('*a*::*finite*, '*b*::*finite*) *complement-domain* = ⟨{..*if* *CARD*('b) *div* *CARD*('a) ≠ 0 *then* *CARD*('b) *div* *CARD*('a) *else* 1}⟩

⟨*proof*⟩

instance *complement-domain* :: (*finite*, *finite*) *finite*

⟨*proof*⟩

lemma *CARD-complement-domain*:

assumes ⟨*CARD*('b::*finite*) = *n* * *CARD*('a::*finite*)⟩

shows ⟨*CARD*('a, 'b) *complement-domain* = *n*⟩

⟨*proof*⟩

lemma *register-decomposition*:

fixes Φ :: ⟨'*a*::*finite update* ⇒ '*b*::*finite update*⟩

assumes [*simp*]: ⟨*register* Φ⟩

shows ⟨∃ *U* :: ('a × ('a, 'b) *complement-domain*) *ell2* ⇒_{CL} 'b *ell2*. *unitary* *U* ∧
(∀ *ϑ*. Φ *ϑ* = *sandwich* *U* (*ϑ* ⊗_o *id-cblinfun*))⟩

— Proof based on [Daw21]

⟨*proof*⟩

lemma *register-decomposition-converse*:

assumes ⟨*unitary* *U*⟩

shows ⟨*register* (λ*x*. *sandwich* *U* (*id-cblinfun* ⊗_o *x*))⟩

<proof>

lemma *register-inj*: $\langle inj\ F \rangle$ **if** $\langle register\ F \rangle$
<proof>

lemma *iso-register-decomposition*:
assumes $[simp]: \langle iso-register\ F \rangle$
shows $\langle \exists U. unitary\ U \wedge F = sandwich\ U \rangle$
<proof>

lemma *complement-exists*:
fixes $F :: \langle 'a::finite\ update \Rightarrow 'b::finite\ update \rangle$
assumes $\langle register\ F \rangle$
shows $\langle \exists G :: ('a, 'b)\ complement-domain\ update \Rightarrow 'b\ update. compatible\ F\ G \wedge iso-register\ (F;G) \rangle$
<proof>

definition $\langle commutant\ F = \{x. \forall y \in F. x\ o_{CL}\ y = y\ o_{CL}\ x\} \rangle$

lemma *commutant-exchange*:
fixes $F :: \langle 'a::finite\ update \Rightarrow 'b::finite\ update \rangle$
assumes $\langle iso-register\ F \rangle$
shows $\langle commutant\ (F\ 'X) = F\ 'commutant\ X \rangle$
<proof>

lemma *commutant-tensor1*: $\langle commutant\ (range\ (\lambda a. a\ \otimes_o\ id-cblinfun)) = range\ (\lambda b. id-cblinfun\ \otimes_o\ b) \rangle$
<proof>

lemma *complement-range*:
assumes $[simp]: \langle compatible\ F\ G \rangle$ **and** $[simp]: \langle iso-register\ (F;G) \rangle$
shows $\langle range\ G = commutant\ (range\ F) \rangle$
<proof>

lemma *same-range-equivalent*:
fixes $F :: \langle 'a::finite\ update \Rightarrow 'c::finite\ update \rangle$ **and** $G :: \langle 'b::finite\ update \Rightarrow 'c::finite\ update \rangle$
assumes $[simp]: \langle register\ F \rangle$ **and** $[simp]: \langle register\ G \rangle$
assumes $\langle range\ F = range\ G \rangle$
shows $\langle equivalent-registers\ F\ G \rangle$
<proof>

lemma *complement-unique*:
assumes $\langle compatible\ F\ G \rangle$ **and** $\langle iso-register\ (F;G) \rangle$
assumes $\langle compatible\ F\ H \rangle$ **and** $\langle iso-register\ (F;H) \rangle$
shows $\langle equivalent-registers\ G\ H \rangle$
<proof>

end

18 Generic laws about complements, instantiated quantumly

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

notation *cblinfun-compose* (**infixl** $*_u$ 55)

notation *tensor-op* (**infixr** \otimes_u 70)

definition $\langle complements\ F\ G \longleftrightarrow compatible\ F\ G \wedge iso-register\ (F;G) \rangle$

lemma *complementsI*: $\langle compatible\ F\ G \Longrightarrow iso-register\ (F;G) \Longrightarrow complements\ F\ G \rangle$
<proof>

lemma *complements-sym*: $\langle complements\ G\ F \rangle$ **if** $\langle complements\ F\ G \rangle$

⟨proof⟩

definition *complement* :: ⟨('a::finite update ⇒ 'b::finite update) ⇒ (('a,'b) complement-domain update ⇒ 'b update)⟩ **where**

⟨complement F = (SOME G :: ('a, 'b) complement-domain 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*:

assumes ⟨register F⟩

shows ⟨complements F (complement F)⟩

⟨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 ⊗_r G) (complement F ⊗_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 $\langle is\text{-}unit\text{-}register\ G \rangle$
shows $\langle equivalent\text{-}registers\ F\ G \rangle$
 $\langle proof \rangle$

lemma *unit-register-domains-isomorphic*:

fixes $F :: \langle 'a::finite\ update \Rightarrow 'c::finite\ update \rangle$
fixes $G :: \langle 'b::finite\ update \Rightarrow 'd::finite\ update \rangle$
assumes $\langle is\text{-}unit\text{-}register\ F \rangle$
assumes $\langle is\text{-}unit\text{-}register\ G \rangle$
shows $\langle \exists I :: 'a\ update \Rightarrow 'b\ update.\ iso\text{-}register\ I \rangle$
 $\langle proof \rangle$

lemma *id-complement-is-unit-register[simp]*: $\langle is\text{-}unit\text{-}register\ (complement\ id) \rangle$
 $\langle proof \rangle$

type-synonym *unit-register-domain* = $\langle (unit,\ unit)\ complement\text{-}domain \rangle$

definition *unit-register* :: $\langle unit\text{-}register\text{-}domain\ update \Rightarrow 'a::finite\ update \rangle$ **where** $\langle unit\text{-}register = (SOME\ U.\ is\text{-}unit\text{-}register\ U) \rangle$

lemma *unit-register-is-unit-register[simp]*: $\langle is\text{-}unit\text{-}register\ (unit\text{-}register :: unit\text{-}register\text{-}domain\ update \Rightarrow 'a::finite\ update) \rangle$
 $\langle proof \rangle$

lemma *unit-register-domain-tensor-unit*:

fixes $U :: \langle 'a::finite\ update \Rightarrow \rightarrow \rangle$
assumes $\langle is\text{-}unit\text{-}register\ U \rangle$
shows $\langle \exists I :: 'b::finite\ update \Rightarrow ('a * 'b)\ update.\ iso\text{-}register\ I \rangle$

$\langle proof \rangle$

lemma *compatible-complement-pair1*:

assumes $\langle compatible\ F\ G \rangle$
shows $\langle compatible\ F\ (complement\ (F;G)) \rangle$
 $\langle proof \rangle$

lemma *compatible-complement-pair2*:

assumes [simp]: $\langle compatible\ F\ G \rangle$
shows $\langle compatible\ G\ (complement\ (F;G)) \rangle$
 $\langle proof \rangle$

lemma *equivalent-complements*:

assumes $\langle complements\ F\ G \rangle$
assumes $\langle equivalent\text{-}registers\ G\ G' \rangle$
shows $\langle complements\ F\ G' \rangle$
 $\langle proof \rangle$

lemma *complements-complement-pair*:

assumes [simp]: $\langle compatible\ F\ G \rangle$
shows $\langle complements\ F\ (G;\ complement\ (F;G)) \rangle$
 $\langle proof \rangle$

lemma *equivalent-registers-complement*:

assumes $\langle equivalent\text{-}registers\ F\ G \rangle$
shows $\langle equivalent\text{-}registers\ (complement\ F)\ (complement\ G) \rangle$
 $\langle proof \rangle$

lemma *complements-complement-pair'*:

assumes [simp]: $\langle compatible\ F\ G \rangle$
shows $\langle complements\ G\ (F;\ complement\ (F;G)) \rangle$
 $\langle proof \rangle$

lemma *complements-chain*:

assumes [*simp*]: $\langle \text{register } F \rangle \langle \text{register } G \rangle$

shows $\langle \text{complements } (F \circ G) \text{ (complement } F; F \circ \text{complement } G) \rangle$

$\langle \text{proof} \rangle$

lemma *complements-Fst-Snd*[*simp*]: $\langle \text{complements } Fst \ Snd \rangle$

$\langle \text{proof} \rangle$

lemma *complements-Snd-Fst*[*simp*]: $\langle \text{complements } Snd \ Fst \rangle$

$\langle \text{proof} \rangle$

lemma *compatible-unit-register*[*simp*]: $\langle \text{register } F \implies \text{compatible } F \ \text{unit-register} \rangle$

$\langle \text{proof} \rangle$

lemma *complements-id-unit-register*[*simp*]: $\langle \text{complements } id \ \text{unit-register} \rangle$

$\langle \text{proof} \rangle$

lemma *complements-iso-unit-register*: $\langle \text{iso-register } I \implies \text{is-unit-register } U \implies \text{complements } I \ U \rangle$

$\langle \text{proof} \rangle$

lemma *iso-register-complement-is-unit-register*[*simp*]:

assumes $\langle \text{iso-register } F \rangle$

shows $\langle \text{is-unit-register (complement } F) \rangle$

$\langle \text{proof} \rangle$

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 $\langle \text{is-unit-register } - \rangle$]

no-notation *cblinfun-compose* (**infixl** $*_u$ 55)

no-notation *tensor-op* (**infixr** \otimes_u 70)

end

19 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

definition *empty-var* :: $\langle 'a::\{CARD-1,enum\} \ \text{update} \Rightarrow 'b::\text{finite update} \rangle$ **where**

empty-var $a = \text{one-dim-iso } a \ *_C \ \text{id-cblinfun}$

lemma *is-unit-register-empty-var*[*register*]: $\langle \text{is-unit-register } \text{empty-var} \rangle$

$\langle \text{proof} \rangle$

instance *complement-domain* :: $(\text{type}, \text{type}) \ \text{default} \ \langle \text{proof} \rangle$

end

theory *Pure-States*

imports *Quantum-Extra2* *HOL-Eisbach.Eisbach*

begin

definition $\langle \text{pure-state-target-vector } F \ \eta_F = (\text{if } \text{ket default} \in \text{range } (\text{cblinfun-apply } (F \ (\text{butterfly } \eta_F \ \eta_F))))$

then ket default

else (SOME η' . $\text{norm } \eta' = 1 \wedge \eta' \in \text{range } (\text{cblinfun-apply } (F \ (\text{butterfly } \eta_F \ \eta_F)))) \rangle$

lemma *pure-state-target-vector-eqI*:

assumes $\langle F (\text{butterfly } \eta_F \eta_F) = G (\text{butterfly } \eta_G \eta_G) \rangle$
shows $\langle \text{pure-state-target-vector } F \eta_F = \text{pure-state-target-vector } G \eta_G \rangle$
 $\langle \text{proof} \rangle$

lemma *pure-state-target-vector-ket-default*: $\langle \text{pure-state-target-vector } F \eta_F = \text{ket default} \rangle$ **if** $\langle \text{ket default} \in \text{range} (\text{cblinfun-apply } (F (\text{butterfly } \eta_F \eta_F))) \rangle$
 $\langle \text{proof} \rangle$

lemma

assumes $[\text{simp}]$: $\langle \eta_F \neq 0 \rangle$ $\langle \text{register } F \rangle$
shows *pure-state-target-vector-in-range*: $\langle \text{pure-state-target-vector } F \eta_F \in \text{range} ((*_V) (F (\text{selfbutter } \eta_F))) \rangle$ **(is ?range)**
and *pure-state-target-vector-norm*: $\langle \text{norm } (\text{pure-state-target-vector } F \eta_F) = 1 \rangle$ **(is ?norm)**
 $\langle \text{proof} \rangle$

lemma *pure-state-target-vector-correct*:

assumes $[\text{simp}]$: $\langle \eta \neq 0 \rangle$ $\langle \text{register } F \rangle$
shows $\langle F (\text{selfbutter } \eta) *_V \text{pure-state-target-vector } F \eta \neq 0 \rangle$
 $\langle \text{proof} \rangle$

definition $\langle \text{pure-state}' F \psi \eta_F = F (\text{butterfly } \psi \eta_F) *_V \text{pure-state-target-vector } F \eta_F \rangle$

abbreviation $\langle \text{pure-state } F \psi \equiv \text{pure-state}' F \psi (\text{ket default}) \rangle$

nonterminal *pure-tensor*

syntax *-pure-tensor* :: $\langle 'a \Rightarrow 'b \Rightarrow \text{pure-tensor} \Rightarrow \text{pure-tensor} \rangle$ $(- \otimes_p - [1000, 0, 0] 1000)$
syntax *-pure-tensor2* :: $\langle 'a \Rightarrow 'b \Rightarrow 'c \Rightarrow 'd \Rightarrow \text{pure-tensor} \rangle$ $(- \otimes_p - - [1000, 0, 1000, 0] 1000)$
syntax *-pure-tensor1* :: $\langle 'a \Rightarrow 'b \Rightarrow \text{pure-tensor} \rangle$
syntax *-pure-tensor-start* :: $\langle \text{pure-tensor} \Rightarrow 'a \rangle$ ('(-)')

translations

-pure-tensor $F \psi G \varphi \rightarrow \text{CONST pure-state } (F; G) (\psi \otimes_s \varphi)$
-pure-tensor $F \psi (\text{CONST pure-state } G \varphi) \rightarrow \text{CONST pure-state } (F; G) (\psi \otimes_s \varphi)$
-pure-tensor-start $x \rightarrow x$

-pure-tensor-start $(\text{-pure-tensor2 } F \psi G \varphi) \leftarrow \text{CONST pure-state } (F; G) (\psi \otimes_s \varphi)$
-pure-tensor $F \psi (\text{-pure-tensor2 } G \varphi H \eta) \leftarrow \text{-pure-tensor2 } F \psi (G; H) (\varphi \otimes_s \eta)$

term $\langle (F \psi \otimes_p G \varphi \otimes_p H z) \rangle$

term $\langle \text{pure-state } (F; G) (a \otimes_s b) \rangle$

lemma *register-pair-butterfly-tensor*: $\langle (F; G) (\text{butterfly } (a \otimes_s b) (c \otimes_s d)) = F (\text{butterfly } a c) \text{o}_{CL} G (\text{butterfly } b d) \rangle$

if $[\text{simp}]$: $\langle \text{compatible } F G \rangle$
 $\langle \text{proof} \rangle$

lemma *pure-state-eqI*:

assumes $\langle F (\text{selfbutter } \eta_F) = G (\text{selfbutter } \eta_G) \rangle$
assumes $\langle F (\text{butterfly } \psi \eta_F) = G (\text{butterfly } \varphi \eta_G) \rangle$
shows $\langle \text{pure-state}' F \psi \eta_F = \text{pure-state}' G \varphi \eta_G \rangle$
 $\langle \text{proof} \rangle$

definition $\langle \text{regular-register } F \longleftrightarrow \text{register } F \wedge (\exists a. (F; \text{complement } F) (\text{selfbutterket default } \otimes_o a) = \text{selfbutterket default}) \rangle$

lemma *regular-registerI*:

assumes $[\text{simp}]$: $\langle \text{register } F \rangle$
assumes $[\text{simp}]$: $\langle \text{complements } F G \rangle$
assumes *eq*: $\langle (F; G) (\text{selfbutterket default } \otimes_o a) = \text{selfbutterket default} \rangle$
shows $\langle \text{regular-register } F \rangle$
 $\langle \text{proof} \rangle$

lemma *regular-register-pair*:

assumes [simp]: $\langle \text{compatible } F \ G \rangle$

assumes $\langle \text{regular-register } F \rangle$ and $\langle \text{regular-register } G \rangle$

shows $\langle \text{regular-register } (F;G) \rangle$

$\langle \text{proof} \rangle$

lemma *regular-register-comp*: $\langle \text{regular-register } (F \circ G) \rangle$ **if** $\langle \text{regular-register } F \rangle$ $\langle \text{regular-register } G \rangle$

$\langle \text{proof} \rangle$

lemma *regular-iso-register*:

assumes $\langle \text{regular-register } F \rangle$

assumes [register]: $\langle \text{iso-register } F \rangle$

shows $\langle F \ (\text{selfbuttket default}) = \text{selfbuttket default} \rangle$

$\langle \text{proof} \rangle$

lemma *pure-state-nested*:

assumes [simp]: $\langle \text{compatible } F \ G \rangle$

assumes $\langle \text{regular-register } H \rangle$

assumes $\langle \text{iso-register } H \rangle$

shows $\langle \text{pure-state } (F;G) \ (\text{pure-state } H \ h \otimes_s g) = \text{pure-state } ((F \circ H);G) \ (h \otimes_s g) \rangle$

$\langle \text{proof} \rangle$

lemma *state-apply1*:

assumes [register]: $\langle \text{compatible } F \ G \rangle$

shows $\langle F \ U \ *_V \ (F \ \psi \ \otimes_p \ G \ \varphi) = (F \ (U \ \psi) \ \otimes_p \ G \ \varphi) \rangle$

$\langle \text{proof} \rangle$

lemma *Fst-regular*[simp]: $\langle \text{regular-register } Fst \rangle$

$\langle \text{proof} \rangle$

lemma *Snd-regular*[simp]: $\langle \text{regular-register } Snd \rangle$

$\langle \text{proof} \rangle$

lemma *id-regular*[simp]: $\langle \text{regular-register } id \rangle$

$\langle \text{proof} \rangle$

lemma *swap-regular*[simp]: $\langle \text{regular-register } swap \rangle$

$\langle \text{proof} \rangle$

lemma *assoc-regular*[simp]: $\langle \text{regular-register } assoc \rangle$

$\langle \text{proof} \rangle$

lemma *assoc'-regular*[simp]: $\langle \text{regular-register } assoc' \rangle$

$\langle \text{proof} \rangle$

lemma *cspan-pure-state'*:

assumes $\langle \text{iso-register } F \rangle$

assumes $\langle \text{cspan } (g \ ' X) = UNIV \rangle$

assumes $\eta\text{-cond}$: $\langle F \ (\text{selfbuttket } \eta) \ *_V \ \text{pure-state-target-vector } F \ \eta \neq 0 \rangle$

shows $\langle \text{cspan } ((\lambda z. \text{pure-state}' F \ (g \ z)) \ ' X) = UNIV \rangle$

$\langle \text{proof} \rangle$

lemma *cspan-pure-state*:

assumes [simp]: $\langle \text{iso-register } F \rangle$

assumes $\langle \text{cspan } (g \ ' X) = UNIV \rangle$

shows $\langle \text{cspan } ((\lambda z. \text{pure-state } F \ (g \ z)) \ ' X) = UNIV \rangle$

$\langle \text{proof} \rangle$

lemma *pure-state-bounded-clinear*:

assumes [register]: $\langle \text{compatible } F \ G \rangle$

shows $\langle \text{bounded-clinear } (\lambda \psi. (F \ \psi \ \otimes_p \ G \ \varphi)) \rangle$

$\langle \text{proof} \rangle$

lemma *pure-state-bounded-clinear-right*:
assumes [*register*]: $\langle \text{compatible } F \ G \rangle$
shows $\langle \text{bounded-clinear } (\lambda\varphi. (F \ \psi \ \otimes_p \ G \ \varphi)) \rangle$
 $\langle \text{proof} \rangle$

lemma *pure-state-clinear*:
assumes [*register*]: $\langle \text{compatible } F \ G \rangle$
shows $\langle \text{clinear } (\lambda\psi. (F \ \psi \ \otimes_p \ G \ \varphi)) \rangle$
 $\langle \text{proof} \rangle$

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 $F_1(\psi_1) \otimes_p F_2(\psi_2) \otimes_p \dots \otimes_p F_n(\psi_n)$ 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 $F :: \langle \text{bit update} \Rightarrow 'c::\{\text{finite,default}\} \text{update} \rangle$
and $G :: \langle \text{bit update} \Rightarrow 'c \text{ update} \rangle$
assumes [*register*]: $\langle \text{compatible } F \ G \rangle$
shows $\langle (F;G) \ \text{CNOT} \ o_{CL} \ (G;F) \ \text{CNOT} \ o_{CL} \ (F;G) \ \text{CNOT} = (F;G) \ \text{swap-ell2} \rangle$
 $\langle \text{proof} \rangle$

end

theory *Check-Autogenerated-Files*

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

$\langle ML \rangle$

end

References

- [Daw21] Matthew Daws. Answer to mathoverflow question “unital *-homomorphisms between matrices”. <https://mathoverflow.net/a/390180>, 2021. Last accessed 2021-10-12.
- [Unr21] Dominique Unruh. Quantum and classical registers. [arXiv:2105.10914](https://arxiv.org/abs/2105.10914) [cs.LO], 2021.