Isabelle Marries Dirac: a Library for Quantum Computation and Quantum Information

Anthony Bordg, Hanna Lachnitt and Yijun He

March 17, 2025

Contents

Abstract

This work is an effort to formalise some quantum algorithms and results in quantum information theory. Formal methods being critical for the safety and security of algorithms and protocols, we foresee their widespread use for quantum computing in the future. We have developed a large library for quantum computing in Isabelle based on a matrix representation for quantum circuits, successfully formalising the no-cloning theorem, quantum teleportation, Deutsch's algorithm, the Deutsch-Jozsa algorithm and the quantum Prisoner's Dilemma.

1 Basic Results

theory *Basics*
imports
HOL.Set-Interval
HOL.Semiring-Normalization
HOL.Real-Vector-Spaces

HOL.Power
HOL.Complex
Jordan-Normal-Form.Jordan-Normal-Form
begin

1.1 Basic Set-Theoretic Results

lemma *set-2-atLeast0* [*simp*]: $\{0..<2::nat\} = \{0,1\}$ ⟨*proof*⟩

lemma *set-2*: $\{..<2::nat\} = \{0,1\}$ ⟨*proof*⟩

lemma *set-4-atLeast0* [*simp*]:$\{0..<4::nat\} = \{0,1,2,3\}$ ⟨*proof*⟩

lemma *set-4*: $\{..<4::nat\} = \{0,1,2,3\}$ ⟨*proof*⟩

lemma *set-4-disj* [*simp*]:
fixes *i*:: *nat*
assumes $i < 4$
shows $i = 0 \vee i = 1 \vee i = 2 \vee i = 3$
⟨*proof*⟩

lemma *set-8-atLeast0* [*simp*]: $\{0..<8::nat\} = \{0,1,2,3,4,5,6,7\}$ ⟨*proof*⟩

lemma *index-is-2* [*simp*]: $\forall i::nat.\ i \neq Suc\ 0 \longrightarrow i \neq 3 \longrightarrow 0 < i \longrightarrow i < 4 \longrightarrow i = 2$ ⟨*proof*⟩

lemma *index-sl-four* [*simp*]: $\forall i::nat.\ i < 4 \longrightarrow i = 0 \vee i = 1 \vee i = 2 \vee i = 3$ ⟨*proof*⟩

1.2 Basic Arithmetic Results

lemma *index-div-eq* [*simp*]:
fixes *i*::*nat*
shows $i \in \{a*b..<(a+1)*b\} \Longrightarrow i\ div\ b = a$
⟨*proof*⟩

lemma *index-mod-eq* [*simp*]:
fixes *i*::*nat*
shows $i \in \{a*b..<(a+1)*b\} \Longrightarrow i\ mod\ b = i-a*b$
⟨*proof*⟩

lemma *sqr-of-cmod-of-prod*:
shows $(cmod\ (z1 * z2))^2 = (cmod\ z1)^2 * (cmod\ z2)^2$
⟨*proof*⟩

lemma *less-power-add-imp-div-less* [*simp*]:
fixes *i m n*:: *nat*
assumes $i < 2\hat{}(m+n)$
shows $i\ div\ 2\hat{}n < 2\hat{}m$
⟨*proof*⟩

lemma *div-mult-mod-eq-minus*:
fixes *i j*:: *nat*
shows (*i div 2^n*) * *2^n* + *i mod 2^n* − (*j div 2^n*) * *2^n* − *j mod 2^n* = *i* − *j*
⟨*proof*⟩

lemma *neq-imp-neq-div-or-mod*:
fixes *i j*:: *nat*
assumes *i* ≠ *j*
shows *i div 2^n* ≠ *j div 2^n* ∨ *i mod 2^n* ≠ *j mod 2^n*
⟨*proof*⟩

lemma *index-one-mat-div-mod*:
assumes *i* < *2^(m+n)* **and** *j* < *2^(m+n)*
shows ((*1*$_m$(*2^m*) $$ (*i div 2^n*, *j div 2^n*))::*complex*) * *1*$_m$(*2^n*) $$ (*i mod 2^n*, *j mod 2^n*) = *1*$_m$(*2^(m+n)*) $$ (*i*, *j*)
⟨*proof*⟩

lemma *sqr-of-sqrt-2* [*simp*]:
fixes *z*:: *complex*
shows *z* * *2* / (*complex-of-real* (*sqrt 2*) * *complex-of-real* (*sqrt 2*)) = *z*
⟨*proof*⟩

lemma *two-div-sqrt-two* [*simp*]:
shows *2* * *complex-of-real* (*sqrt* (*1*/*2*)) = *complex-of-real* (*sqrt 2*)
⟨*proof*⟩

lemma *two-div-sqr-of-cmd-sqrt-two* [*simp*]:
shows *2* * (*cmod* (*1* / *complex-of-real* (*sqrt 2*)))^2 = *1*
⟨*proof*⟩

lemma *two-div-two* [*simp*]:
shows *2 div Suc* (*Suc 0*) = *1* ⟨*proof*⟩

lemma *two-mod-two* [*simp*]:
shows *2 mod Suc* (*Suc 0*) = *0* ⟨*proof*⟩

lemma *three-div-two* [*simp*]:
shows *3 div Suc* (*Suc 0*) = *1* ⟨*proof*⟩

lemma *three-mod-two* [*simp*]:
shows *3 mod Suc* (*Suc 0*) = *1* ⟨*proof*⟩

1.3 Basic Results on Matrices

lemma *index-matrix-prod* [*simp*]:
assumes *i* < *dim-row A* **and** *j* < *dim-col B* **and** *dim-col A* = *dim-row B*
shows (*A* * *B*) $$ (*i*,*j*) = (∑ *k*<*dim-row B*. (*A* $$ (*i*,*k*)) * (*B* $$ (*k*,*j*)))
⟨*proof*⟩

1.4 Basic Results on Sums

lemma *sum-insert* [*simp*]:
assumes $x \notin F$ **and** *finite F*
shows $(\sum y{\in}insert\ x\ F.\ P\ y) = (\sum y{\in}F.\ P\ y) + P\ x$
⟨*proof*⟩

lemma *sum-of-index-diff* [*simp*]:
fixes f:: $nat \Rightarrow {'a}$::*comm-monoid-add*
shows $(\sum i{\in}\{a..{<}a{+}b\}.\ f(i{-}a)) = (\sum i{\in}\{..{<}b\}.\ f(i))$
⟨*proof*⟩

1.5 Basic Results Involving the Exponential Function.

lemma *exp-of-real-cnj*:
fixes x ::*real*
shows $cnj\ (exp\ (\mathrm{i} * x)) = exp\ (-(\mathrm{i} * x))$
⟨*proof*⟩

lemma *exp-of-real-cnj2*:
fixes x ::*real*
shows $cnj\ (exp\ (-(\mathrm{i} * x))) = exp\ (\mathrm{i} * x)$
⟨*proof*⟩

lemma *exp-of-half-pi*:
fixes x:: *real*
assumes $x = pi/2$
shows $exp\ (\mathrm{i} * complex\text{-}of\text{-}real\ x) = \mathrm{i}$
⟨*proof*⟩

lemma *exp-of-minus-half-pi*:
fixes x:: *real*
assumes $x = pi/2$
shows $exp\ (-(\mathrm{i} * complex\text{-}of\text{-}real\ x)) = -\mathrm{i}$
⟨*proof*⟩

lemma *exp-of-real*:
fixes x:: *real*
shows $exp\ (\mathrm{i} * x) = cos\ x + \mathrm{i} * (sin\ x)$
⟨*proof*⟩

lemma *exp-of-real-inv*:
fixes x:: *real*
shows $exp\ (-(\mathrm{i} * x)) = cos\ x - \mathrm{i} * (sin\ x)$
⟨*proof*⟩

1.6 Basic Results with Trigonometric Functions.

1.6.1 Basic Inequalities

lemma *sin-squared-le-one*:
fixes *x*:: *real*
shows $(sin\ x)^2 \leq 1$
⟨*proof*⟩

lemma *cos-squared-le-one*:
fixes *x*:: *real*
shows $(cos\ x)^2 \leq 1$
⟨*proof*⟩

1.6.2 Basic Equalities

lemma *sin-of-quarter-pi*:
fixes *x*:: *real*
assumes *x* = *pi*/2
shows *sin* (*x*/2) = (*sqrt* 2)/2
⟨*proof*⟩

lemma *cos-of-quarter-pi*:
fixes *x*:: *real*
assumes *x* = *pi*/2
shows *cos* (*x*/2) = (*sqrt* 2)/2
⟨*proof*⟩

end

2 Binary Representation of Natural Numbers

theory *Binary-Nat*
imports
HOL.Nat
HOL.List
Basics
begin

primrec *bin-rep-aux*:: *nat* ⇒ *nat* ⇒ *nat list* **where**
bin-rep-aux 0 m = [*m*]
| *bin-rep-aux* (*Suc n*) *m* = *m div* 2^*n* # *bin-rep-aux n* (*m mod* 2^*n*)

lemma *length-of-bin-rep-aux*:
fixes *n m*:: *nat*
assumes *m* < 2^*n*
shows *length* (*bin-rep-aux n m*) = *n*+1
⟨*proof*⟩

lemma *bin-rep-aux-neq-nil*:
fixes *n m*:: *nat*
shows *bin-rep-aux n m* ≠ []
⟨*proof*⟩

lemma *last-of-bin-rep-aux*:
fixes *n m*:: *nat*
assumes *m* < *2^n* **and** *m* ≥ *0*
shows *last* (*bin-rep-aux n m*) = *0*
⟨*proof*⟩

lemma *mod-mod-power-cancel*:
fixes *m n p*:: *nat*
assumes *m* ≤ *n*
shows *p mod 2^n mod 2^m* = *p mod 2^m*
⟨*proof*⟩

lemma *bin-rep-aux-index*:
fixes *n m i*:: *nat*
assumes *n* ≥ *1* **and** *m* < *2^n* **and** *m* ≥ *0* **and** *i* ≤ *n*
shows *bin-rep-aux n m* ! *i* = (*m mod 2^(n−i)*) *div 2^(n−1−i)*
⟨*proof*⟩

lemma *bin-rep-aux-coeff*:
fixes *n m i*:: *nat*
assumes *m* < *2^n* **and** *i* ≤ *n* **and** *m* ≥ *0*
shows *bin-rep-aux n m* ! *i* = *0* ∨ *bin-rep-aux n m* ! *i* = *1*
⟨*proof*⟩

definition *bin-rep*:: *nat* ⇒ *nat* ⇒ *nat list* **where**
bin-rep n m = *butlast* (*bin-rep-aux n m*)

lemma *length-of-bin-rep*:
fixes *n m*:: *nat*
assumes *m* < *2^n*
shows *length* (*bin-rep n m*) = *n*
⟨*proof*⟩

lemma *bin-rep-coeff*:
fixes *n m i*:: *nat*
assumes *m* < *2^n* **and** *i* < *n* **and** *m* ≥ *0*
shows *bin-rep n m* ! *i* = *0* ∨ *bin-rep n m* ! *i* = *1*
⟨*proof*⟩

lemma *bin-rep-index*:
fixes *n m i*:: *nat*
assumes *n* ≥ *1* **and** *m* < *2^n* **and** *i* < *n* **and** *m* ≥ *0*
shows *bin-rep n m* ! *i* = (*m mod 2^(n−i)*) *div 2^(n−1−i)*
⟨*proof*⟩

lemma *bin-rep-eq*:
fixes *n m*:: *nat*
assumes $n \geq 1$ **and** $m \geq 0$ **and** $m < 2\hat{}n$ **and** $m \geq 0$
shows $m = (\sum i{<}n.\ bin\text{-}rep\ n\ m\ !\ i * 2\hat{}(n-1-i))$
⟨*proof*⟩

lemma *bin-rep-index-0*:
fixes *n m*:: *nat*
assumes $m < 2\hat{}n$ **and** $k > n$
shows $(bin\text{-}rep\ k\ m)\ !\ 0 = 0$
⟨*proof*⟩

lemma *bin-rep-index-0-geq*:
fixes *n m*:: *nat*
assumes $m \geq 2\hat{}n$ **and** $m < 2\hat{}(n+1)$
shows $bin\text{-}rep\ (n+1)\ m\ !\ 0 = 1$
⟨*proof*⟩

end

3 Qubits and Quantum Gates

theory *Quantum*
imports
Jordan-Normal-Form.Matrix
HOL−Library.Nonpos-Ints
Basics
Binary-Nat
begin

3.1 Qubits

In this theory *cpx* stands for *complex*.

definition *cpx-vec-length* :: *complex vec* ⇒ *real* (‹‖-‖›) **where**
$cpx\text{-}vec\text{-}length\ v \equiv sqrt(\sum i{<}dim\text{-}vec\ v.\ (cmod\ (v\ \$\ i))^2)$

lemma *cpx-length-of-vec-of-list* [*simp*]:
$\|vec\text{-}of\text{-}list\ l\| = sqrt(\sum i{<}length\ l.\ (cmod\ (l\ !\ i))^2)$
⟨*proof*⟩

lemma *norm-vec-index-unit-vec-is-0* [*simp*]:
assumes $j < n$ **and** $j \neq i$
shows $cmod\ ((unit\text{-}vec\ n\ i)\ \$\ j) = 0$
⟨*proof*⟩

lemma *norm-vec-index-unit-vec-is-1* [*simp*]:
assumes $j < n$ **and** $j = i$

shows *cmod* ((*unit-vec n i*) $ *j*) = *1*
⟨*proof*⟩

lemma *unit-cpx-vec-length* [*simp*]:
assumes $i < n$
shows $\|$*unit-vec n i*$\| = 1$
⟨*proof*⟩

lemma *smult-vec-length* [*simp*]:
assumes $x \geq 0$
shows $\|$*complex-of-real*(x) $\cdot_v$ $v\| = x * \|v\|$
⟨*proof*⟩

locale *state* =
fixes *n*:: *nat* **and** *v*:: *complex mat*
assumes *is-column* [*simp*]: *dim-col v = 1*
and *dim-row* [*simp*]: *dim-row v = 2^n*
and *is-normal* [*simp*]: $\|$*col v 0*$\| = 1$

Below the natural number n codes for the dimension of the complex vector space whose elements of norm 1 we call states.

lemma *unit-vec-of-right-length-is-state* [*simp*]:
assumes $i < 2\hat{}n$
shows *unit-vec* (*2^n*) *i* ∈ {*v*| *n v*::*complex vec. dim-vec v = 2^n* ∧ $\|v\| = 1$}
⟨*proof*⟩

definition *state-qbit* :: *nat* ⇒ *complex vec set* **where**
state-qbit n ≡ {*v*| *v*:: *complex vec. dim-vec v = 2^n* ∧ $\|v\| = 1$}

lemma (**in** *state*) *state-to-state-qbit* [*simp*]:
shows *col v 0* ∈ *state-qbit n*
⟨*proof*⟩

3.2 The Hermitian Conjugation

The Hermitian conjugate of a complex matrix is the complex conjugate of its transpose.

definition *dagger* :: *complex mat* ⇒ *complex mat* (‹-†›) **where**
$M^\dagger$ ≡ *mat* (*dim-col M*) (*dim-row M*) (λ(*i*,*j*). *cnj*(*M* $$ (*j*,*i*)))

We introduce the type of complex square matrices.

typedef *cpx-sqr-mat* = {*M* | *M*::*complex mat. square-mat M*}
⟨*proof*⟩

definition *cpx-sqr-mat-to-cpx-mat* :: *cpx-sqr-mat* => *complex mat* **where**
cpx-sqr-mat-to-cpx-mat M ≡ *Rep-cpx-sqr-mat M*

We introduce a coercion from the type of complex square matrices to the type of complex matrices.

declare [[*coercion cpx-sqr-mat-to-cpx-mat*]]

lemma *dim-row-of-dagger* [*simp*]:
dim-row $(M^\dagger)$ = *dim-col M*
⟨*proof*⟩

lemma *dim-col-of-dagger* [*simp*]:
dim-col $(M^\dagger)$ = *dim-row M*
⟨*proof*⟩

lemma *col-of-dagger* [*simp*]:
assumes $j <$ *dim-row M*
shows *col* $(M^\dagger)$ *j* = *vec* (*dim-col M*) ($\lambda i.$ *cnj* (*M* $$ (*j*,*i*)))
⟨*proof*⟩

lemma *row-of-dagger* [*simp*]:
assumes $i <$ *dim-col M*
shows *row* $(M^\dagger)$ *i* = *vec* (*dim-row M*) ($\lambda j.$ *cnj* (*M* $$ (*j*,*i*)))
⟨*proof*⟩

lemma *dagger-of-dagger-is-id*:
fixes *M* :: *complex Matrix.mat*
shows $(M^\dagger)^\dagger = M$
⟨*proof*⟩

lemma *dagger-of-sqr-is-sqr* [*simp*]:
square-mat $((M::$*cpx-sqr-mat*$)^\dagger)$
⟨*proof*⟩

lemma *dagger-of-id-is-id* [*simp*]:
$(1_m\ n)^\dagger = 1_m\ n$
⟨*proof*⟩

3.3 Unitary Matrices and Quantum Gates

definition *unitary* :: *complex mat* ⇒ *bool* **where**
unitary M ≡ $(M^\dagger) * M = 1_m$ (*dim-col M*) ∧ $M * (M^\dagger) = 1_m$ (*dim-row M*)

lemma *id-is-unitary* [*simp*]:
unitary $(1_m\ n)$
⟨*proof*⟩

locale *gate* =
fixes *n*:: *nat* **and** *A*:: *complex mat*
assumes *dim-row* [*simp*]: *dim-row A* = 2^*n*
and *square-mat* [*simp*]: *square-mat A*
and *unitary* [*simp*]: *unitary A*

We prove that a quantum gate is invertible and its inverse is given by its Hermitian conjugate.

lemma *mat-unitary-mat* [*intro*]:
assumes *unitary M*
shows *inverts-mat M* ($M^\dagger$)
⟨*proof*⟩

lemma *unitary-mat-mat* [*intro*]:
assumes *unitary M*
shows *inverts-mat* ($M^\dagger$) *M*
⟨*proof*⟩

lemma (**in** *gate*) *gate-is-inv*:
invertible-mat A
⟨*proof*⟩

3.4 Relations Between Complex Conjugation, Hermitian Conjugation, Transposition and Unitarity

notation *transpose-mat* (‹(-^{t})›)

lemma *col-tranpose* [*simp*]:
assumes *dim-row M* = *n* **and** *i* < *n*
shows *col* (M^t) *i* = *row M i*
⟨*proof*⟩

lemma *row-transpose* [*simp*]:
assumes *dim-col M* = *n* **and** *i* < *n*
shows *row* (M^t) *i* = *col M i*
⟨*proof*⟩

definition *cpx-mat-cnj* :: *complex mat* ⇒ *complex mat* (‹(-*)›) **where**
cpx-mat-cnj M ≡ *mat* (*dim-row M*) (*dim-col M*) (λ(*i*,*j*). *cnj* (*M* $$ (*i*,*j*)))

lemma *cpx-mat-cnj-id* [*simp*]:
(1_m *n*)* = 1_m *n*
⟨*proof*⟩

lemma *cpx-mat-cnj-cnj* [*simp*]:
($M^\star$)* = *M*
⟨*proof*⟩

lemma *dim-row-of-cjn-prod* [*simp*]:
dim-row (($M^\star$) ∗ ($N^\star$)) = *dim-row M*
⟨*proof*⟩

lemma *dim-col-of-cjn-prod* [*simp*]:
dim-col (($M^\star$) ∗ ($N^\star$)) = *dim-col N*
⟨*proof*⟩

lemma *cpx-mat-cnj-prod*:

assumes *dim-col M* = *dim-row N*
shows $(M * N)^\star = (M^\star) * (N^\star)$
⟨*proof*⟩

lemma *transpose-of-prod*:
fixes *M N*::*complex Matrix.mat*
assumes *dim-col M* = *dim-row N*
shows $(M * N)^t = N^t * (M^t)$
⟨*proof*⟩

lemma *transpose-cnj-is-dagger* [*simp*]:
$(M^t)^\star = (M^\dagger)$
⟨*proof*⟩

lemma *cnj-transpose-is-dagger* [*simp*]:
$(M^\star)^t = (M^\dagger)$
⟨*proof*⟩

lemma *dagger-of-transpose-is-cnj* [*simp*]:
$(M^t)^\dagger = (M^\star)$
⟨*proof*⟩

lemma *dagger-of-prod*:
fixes *M N*::*complex Matrix.mat*
assumes *dim-col M* = *dim-row N*
shows $(M * N)^\dagger = N^\dagger * (M^\dagger)$
⟨*proof*⟩

The product of two quantum gates is a quantum gate.

lemma *prod-of-gate-is-gate*:
assumes *gate n G1* **and** *gate n G2*
shows *gate n* (*G1* ∗ *G2*)
⟨*proof*⟩

lemma *left-inv-of-unitary-transpose* [*simp*]:
assumes *unitary U*
shows $(U^t)^\dagger * (U^t) = 1_m(\textit{dim-row } U)$
⟨*proof*⟩

lemma *right-inv-of-unitary-transpose* [*simp*]:
assumes *unitary U*
shows $U^t * ((U^t)^\dagger) = 1_m(\textit{dim-col } U)$
⟨*proof*⟩

lemma *transpose-of-unitary-is-unitary* [*simp*]:
assumes *unitary U*
shows *unitary* (U^t)
⟨*proof*⟩

3.5 The Inner Product

We introduce a coercion between complex vectors and (column) complex matrices.

definition *ket-vec* :: *complex vec* ⇒ *complex mat* (‹|-⟩›) **where**
$|v\rangle \equiv mat\ (dim\text{-}vec\ v)\ 1\ (\lambda(i,j).\ v\ \$\ i)$

lemma *ket-vec-index* [*simp*]:
assumes $i < dim\text{-}vec\ v$
shows $|v\rangle\ \$\$\ (i,0) = v\ \$\ i$
⟨*proof*⟩

lemma *ket-vec-col* [*simp*]:
$col\ |v\rangle\ 0 = v$
⟨*proof*⟩

lemma *smult-ket-vec* [*simp*]:
$|x \cdot_v v\rangle = x \cdot_m |v\rangle$
⟨*proof*⟩

lemma *smult-vec-length-bis* [*simp*]:
assumes $x \geq 0$
shows $\|col\ (complex\text{-}of\text{-}real(x) \cdot_m |v\rangle)\ 0\| = x * \|v\|$
⟨*proof*⟩

declare [[*coercion ket-vec*]]

definition *row-vec* :: *complex vec* ⇒ *complex mat* **where**
$row\text{-}vec\ v \equiv mat\ 1\ (dim\text{-}vec\ v)\ (\lambda(i,j).\ v\ \$\ j)$

definition *bra-vec* :: *complex vec* ⇒ *complex mat* **where**
$bra\text{-}vec\ v \equiv (row\text{-}vec\ v)^\star$

lemma *row-bra-vec* [*simp*]:
$row\ (bra\text{-}vec\ v)\ 0 = vec\ (dim\text{-}vec\ v)\ (\lambda i.\ cnj(v\ \$\ i))$
⟨*proof*⟩

We introduce a definition called *bra* to see a vector as a column matrix.

definition *bra* :: *complex mat* ⇒ *complex mat* (‹⟨-|›) **where**
$\langle v| \equiv mat\ 1\ (dim\text{-}row\ v)\ (\lambda(i,j).\ cnj(v\ \$\$\ (j,i)))$

The relation between *bra*, *bra-vec* and *ket-vec* is given as follows.

lemma *bra-bra-vec* [*simp*]:
$bra\ (ket\text{-}vec\ v) = bra\text{-}vec\ v$
⟨*proof*⟩

lemma *row-bra* [*simp*]:
fixes *v*::*complex vec*
shows $row\ \langle v|\ 0 = vec\ (dim\text{-}vec\ v)\ (\lambda i.\ cnj\ (v\ \$\ i))$ ⟨*proof*⟩

We introduce the inner product of two complex vectors in $\mathbb{C}^n$.

definition *inner-prod* :: *complex vec* ⇒ *complex vec* ⇒ *complex* (‹⟨-|-⟩›) **where**
inner-prod u v ≡ $\sum$ *i* ∈ {*0*..< *dim-vec v*}. *cnj*(*u* $ *i*) * (*v* $ *i*)

lemma *inner-prod-with-row-bra-vec* [*simp*]:
assumes *dim-vec u* = *dim-vec v*
shows ⟨*u*|*v*⟩ = *row* (*bra-vec u*) *0* · *v*
⟨*proof*⟩

lemma *inner-prod-with-row-bra-vec-col-ket-vec* [*simp*]:
assumes *dim-vec u* = *dim-vec v*
shows ⟨*u*|*v*⟩ = (*row* ⟨*u*| *0*) · (*col* |*v*⟩ *0*)
⟨*proof*⟩

lemma *inner-prod-with-times-mat* [*simp*]:
assumes *dim-vec u* = *dim-vec v*
shows ⟨*u*|*v*⟩ = (⟨*u*| * |*v*⟩) $$ (*0*,*0*)
⟨*proof*⟩

lemma *orthogonal-unit-vec* [*simp*]:
assumes *i* < *n* **and** *j* < *n* **and** *i* ≠ *j*
shows ⟨*unit-vec n i*|*unit-vec n j*⟩ = *0*
⟨*proof*⟩

We prove that our inner product is linear in its second argument.

lemma *vec-index-is-linear* [*simp*]:
assumes *dim-vec u* = *dim-vec v* **and** *j* < *dim-vec u*
shows (*k* ·$_v$ *u* + *l* ·$_v$ *v*) $ *j* = *k* * (*u* $ *j*) + *l* * (*v* $ *j*)
⟨*proof*⟩

lemma *inner-prod-is-linear* [*simp*]:
fixes *u*::*complex vec* **and** *v*::*nat* ⇒ *complex vec* **and** *l*::*nat* ⇒ *complex*
assumes ∀ *i*∈{*0*, *1*}. *dim-vec u* = *dim-vec* (*v i*)
shows ⟨*u*|*l 0* ·$_v$ *v 0* + *l 1* ·$_v$ *v 1*⟩ = ($\sum$ *i*≤*1*. *l i* * ⟨*u*|*v i*⟩)
⟨*proof*⟩

lemma *inner-prod-cnj*:
assumes *dim-vec u* = *dim-vec v*
shows ⟨*v*|*u*⟩ = *cnj* (⟨*u*|*v*⟩)
⟨*proof*⟩

lemma *inner-prod-with-itself-Im* [*simp*]:
Im (⟨*u*|*u*⟩) = *0*
⟨*proof*⟩

lemma *inner-prod-with-itself-real* [*simp*]:
⟨*u*|*u*⟩ ∈ ℝ
⟨*proof*⟩

lemma *inner-prod-with-itself-eq0* [*simp*]:
assumes $u = 0_v$ (*dim-vec u*)
shows $\langle u|u\rangle = 0$
⟨*proof*⟩

lemma *inner-prod-with-itself-Re*:
Re ($\langle u|u\rangle$) ≥ 0
⟨*proof*⟩

lemma *inner-prod-with-itself-nonneg-reals*:
fixes *u*::*complex vec*
shows $\langle u|u\rangle \in$ *nonneg-Reals*
⟨*proof*⟩

lemma *inner-prod-with-itself-Re-non0*:
assumes $u \neq 0_v$ (*dim-vec u*)
shows *Re* ($\langle u|u\rangle$) > 0
⟨*proof*⟩

lemma *inner-prod-with-itself-nonneg-reals-non0*:
assumes $u \neq 0_v$ (*dim-vec u*)
shows $\langle u|u\rangle \neq 0$
⟨*proof*⟩

lemma *cpx-vec-length-inner-prod* [*simp*]:
$\|v\|^2 = \langle v|v\rangle$
⟨*proof*⟩

lemma *inner-prod-csqrt* [*simp*]:
csqrt $\langle v|v\rangle = \|v\|$
⟨*proof*⟩

3.6 Unitary Matrices and Length-Preservation

3.6.1 Unitary Matrices are Length-Preserving

The bra-vector $\langle A * v|$ is given by $\langle v| * A^\dagger$

lemma *dagger-of-ket-is-bra*:
fixes *v*:: *complex vec*
shows ($|v\rangle$)$^\dagger = \langle v|$
⟨*proof*⟩

lemma *bra-mat-on-vec*:
fixes *v*::*complex vec* **and** *A*::*complex mat*
assumes *dim-col A* = *dim-vec v*
shows $\langle A * v| = \langle v| * (A^\dagger)$
⟨*proof*⟩

lemma *mat-on-ket*:

fixes *v*:: *complex vec* **and** *A*:: *complex mat*
assumes *dim-col A* = *dim-vec v*
shows *A* ∗ |*v*⟩ = |*col* (*A* ∗ *v*) *0*⟩
⟨*proof*⟩

lemma *dagger-of-mat-on-ket*:
fixes *v*:: *complex vec* **and** *A* :: *complex mat*
assumes *dim-col A* = *dim-vec v*
shows (*A* ∗ |*v*⟩)† = ⟨*v*| ∗ (*A*†)
⟨*proof*⟩

definition *col-fst* :: *'a mat* ⇒ *'a vec* **where**
col-fst A = *vec* (*dim-row A*) (λ *i*. *A* $$ (*i*,*0*))

lemma *col-fst-is-col* [*simp*]:
col-fst M = *col M 0*
⟨*proof*⟩

We need to declare *col-fst* as a coercion from matrices to vectors in order to see a column matrix as a vector.

declare
[[*coercion-delete ket-vec*]]
[[*coercion col-fst*]]

lemma *unit-vec-to-col*:
assumes *dim-col A* = *n* **and** *i* < *n*
shows *col A i* = *A* ∗ |*unit-vec n i*⟩
⟨*proof*⟩

lemma *mult-ket-vec-is-ket-vec-of-mult*:
fixes *A*::*complex mat* **and** *v*::*complex vec*
assumes *dim-col A* = *dim-vec v*
shows |*A* ∗ |*v*⟩ ⟩ = *A* ∗ |*v*⟩
⟨*proof*⟩

lemma *unitary-is-sq-length-preserving* [*simp*]:
assumes *unitary U* **and** *dim-vec v* = *dim-col U*
shows ‖*U* ∗ |*v*⟩‖² = ‖*v*‖²
⟨*proof*⟩

lemma *col-ket-vec* [*simp*]:
assumes *dim-col M* = *1*
shows |*col M 0*⟩ = *M*
⟨*proof*⟩

lemma *state-col-ket-vec*:
assumes *state 1 v*
shows *state 1* |*col v 0*⟩
⟨*proof*⟩

lemma *col-ket-vec-index* [*simp*]:
assumes $i < dim\text{-}row\ v$
shows $|col\ v\ 0\rangle\ \$\$\ (i,0) = v\ \$\$\ (i,0)$
⟨*proof*⟩

lemma *col-index-of-mat-col* [*simp*]:
assumes $dim\text{-}col\ v = 1$ **and** $i < dim\text{-}row\ v$
shows $col\ v\ 0\ \$\ i = v\ \$\$\ (i,0)$
⟨*proof*⟩

lemma *unitary-is-sq-length-preserving-bis* [*simp*]:
assumes *unitary U* **and** $dim\text{-}row\ v = dim\text{-}col\ U$ **and** $dim\text{-}col\ v = 1$
shows $\|col\ (U * v)\ 0\|^2 = \|col\ v\ 0\|^2$
⟨*proof*⟩

A unitary matrix is length-preserving, i.e. it acts on a vector to produce another vector of the same length.

lemma *unitary-is-length-preserving-bis* [*simp*]:
fixes *U*::*complex mat* **and** *v*::*complex mat*
assumes *unitary U* **and** $dim\text{-}row\ v = dim\text{-}col\ U$ **and** $dim\text{-}col\ v = 1$
shows $\|col\ (U * v)\ 0\| = \|col\ v\ 0\|$
⟨*proof*⟩

lemma *unitary-is-length-preserving* [*simp*]:
fixes *U*:: *complex mat* **and** *v*:: *complex vec*
assumes *unitary U* **and** $dim\text{-}vec\ v = dim\text{-}col\ U$
shows $\|U * |v\rangle\| = \|v\|$
⟨*proof*⟩

3.6.2 Length-Preserving Matrices are Unitary

lemma *inverts-mat-sym*:
fixes *A B*:: *complex mat*
assumes *inverts-mat A B* **and** $dim\text{-}row\ B = dim\text{-}col\ A$ **and** *square-mat B*
shows *inverts-mat B A*
⟨*proof*⟩

lemma *sum-of-unit-vec-length*:
fixes *i j n*:: *nat* **and** *c*:: *complex*
assumes $i < n$ **and** $j < n$ **and** $i \neq j$
shows $\|unit\text{-}vec\ n\ i + c \cdot_v unit\text{-}vec\ n\ j\|^2 = 1 + cnj(c) * c$
⟨*proof*⟩

lemma *sum-of-unit-vec-to-col*:
assumes $dim\text{-}col\ A = n$ **and** $i < n$ **and** $j < n$
shows $col\ A\ i + c \cdot_v col\ A\ j = A * |unit\text{-}vec\ n\ i + c \cdot_v unit\text{-}vec\ n\ j\rangle$
⟨*proof*⟩

lemma *inner-prod-is-sesquilinear*:
fixes *u1 u2 v1 v2*:: *complex vec* **and** *c1 c2 c3 c4*:: *complex* **and** *n*:: *nat*
assumes *dim-vec u1* = *n* **and** *dim-vec u2* = *n* **and** *dim-vec v1* = *n* **and** *dim-vec v2* = *n*
shows $\langle c1 \cdot_v u1 + c2 \cdot_v u2 | c3 \cdot_v v1 + c4 \cdot_v v2 \rangle = cnj\ (c1) * c3 * \langle u1|v1 \rangle + cnj\ (c2) * c3 * \langle u2|v1 \rangle + cnj\ (c1) * c4 * \langle u1|v2 \rangle + cnj\ (c2) * c4 * \langle u2|v2 \rangle$
⟨*proof*⟩

A length-preserving matrix is unitary. So, unitary matrices are exactly the length-preserving matrices.

lemma *length-preserving-is-unitary*:
fixes *U*:: *complex mat*
assumes *square-mat U* **and** $\forall$ *v*::*complex vec*. *dim-vec v* = *dim-col U* $\longrightarrow$ $\|U * |v\rangle\| = \|v\|$
shows *unitary U*
⟨*proof*⟩

lemma *inner-prod-with-unitary-mat* [*simp*]:
assumes *unitary U* **and** *dim-vec u* = *dim-col U* **and** *dim-vec v* = *dim-col U*
shows $\langle U * |u\rangle | U * |v\rangle \rangle = \langle u|v \rangle$
⟨*proof*⟩

As a consequence we prove that columns and rows of a unitary matrix are orthonormal vectors.

lemma *unitary-unit-col* [*simp*]:
assumes *unitary U* **and** *dim-col U* = *n* **and** *i* < *n*
shows $\|col\ U\ i\| = 1$
⟨*proof*⟩

lemma *unitary-unit-row* [*simp*]:
assumes *unitary U* **and** *dim-row U* = *n* **and** *i* < *n*
shows $\|row\ U\ i\| = 1$
⟨*proof*⟩

lemma *orthogonal-col-of-unitary* [*simp*]:
assumes *unitary U* **and** *dim-col U* = *n* **and** *i* < *n* **and** *j* < *n* **and** *i* ≠ *j*
shows $\langle col\ U\ i | col\ U\ j \rangle = 0$
⟨*proof*⟩

lemma *orthogonal-row-of-unitary* [*simp*]:
fixes *U*::*complex mat*
assumes *unitary U* **and** *dim-row U* = *n* **and** *i* < *n* **and** *j* < *n* **and** *i* ≠ *j*
shows $\langle row\ U\ i | row\ U\ j \rangle = 0$
⟨*proof*⟩

As a consequence, we prove that a quantum gate acting on a state of a system of n qubits give another state of that same system.

lemma *gate-on-state-is-state* [*intro*, *simp*]:
assumes *a1*:*gate n A* **and** *a2*:*state n v*
shows *state n* (*A* ∗ *v*)
⟨*proof*⟩

3.7 A Few Well-known Quantum Gates

Any unitary operation on n qubits can be implemented exactly by composing single qubits and CNOT-gates (controlled-NOT gates). However, no straightforward method is known to implement these gates in a fashion which is resistant to errors. But, the Hadamard gate, the phase gate, the CNOT-gate and the $\pi/8$ gate are also universal for quantum computations, i.e. any quantum circuit on n qubits can be approximated to an arbitrary accuracy by using only these gates, and these gates can be implemented in a fault-tolerant way.

We introduce a coercion from real matrices to complex matrices.

definition *real-to-cpx-mat*:: *real mat* ⇒ *complex mat* **where**
real-to-cpx-mat A ≡ *mat* (*dim-row A*) (*dim-col A*) (λ(*i*,*j*). *A* $$ (*i*,*j*))

Our first quantum gate: the identity matrix! Arguably, not a very interesting one though!

definition *Id* :: *nat* ⇒ *complex mat* **where**
Id n ≡ 1_m (*2^n*)

lemma *id-is-gate* [*simp*]:
gate n (*Id n*)
⟨*proof*⟩

More interesting: the Pauli matrices.

definition *X* ::*complex mat* **where**
X ≡ *mat 2 2* (λ(*i*,*j*). *if i=j then 0 else 1*)

Be aware that *gate n A* means that the matrix A has dimension *2^n* ∗ *2^n*. For instance, with this convention a 2 X 2 matrix A which is unitary satisfies *gate 1 A* but not *gate 2 A* as one might have been expected.

lemma *dagger-of-X* [*simp*]:
$X^\dagger = X$
⟨*proof*⟩

lemma *X-inv* [*simp*]:
$X * X = 1_m\ 2$
⟨*proof*⟩

lemma *X-is-gate* [*simp*]:
gate 1 X
⟨*proof*⟩

definition Y ::*complex mat* **where**
$Y \equiv$ *mat 2 2* (λ(*i*,*j*). *if i=j then 0 else* (*if i=0 then* $-$i *else* i))

lemma *dagger-of-Y* [*simp*]:
$Y^\dagger = Y$
⟨*proof*⟩

lemma *Y-inv* [*simp*]:
$Y * Y = 1_m\ 2$
⟨*proof*⟩

lemma *Y-is-gate* [*simp*]:
gate 1 Y
⟨*proof*⟩

definition Z ::*complex mat* **where**
$Z \equiv$ *mat 2 2* (λ(*i*,*j*). *if i≠j then 0 else* (*if i=0 then 1 else* -1))

lemma *dagger-of-Z* [*simp*]:
$Z^\dagger = Z$
⟨*proof*⟩

lemma *Z-inv* [*simp*]:
$Z * Z = 1_m\ 2$
⟨*proof*⟩

lemma *Z-is-gate* [*simp*]:
gate 1 Z
⟨*proof*⟩

The Hadamard gate

definition H ::*complex mat* **where**
$H \equiv 1/sqrt(2) \cdot_m$ (*mat 2 2* (λ(*i*,*j*). *if i≠j then 1 else* (*if i=0 then 1 else* -1)))

lemma *H-without-scalar-prod*:
$H =$ *mat 2 2* (λ(*i*,*j*). *if i≠j then 1/sqrt(2) else* (*if i=0 then 1/sqrt(2) else* $-(1/sqrt(2))$))
⟨*proof*⟩

lemma *dagger-of-H* [*simp*]:
$H^\dagger = H$
⟨*proof*⟩

lemma *H-inv* [*simp*]:
$H * H = 1_m\ 2$
⟨*proof*⟩

lemma *H-is-gate* [*simp*]:

gate 1 H
⟨*proof*⟩

lemma *H-values*:
fixes *i j*:: *nat*
assumes $i < dim\text{-}row\ H$ **and** $j < dim\text{-}col\ H$ **and** $i \neq 1 \vee j \neq 1$
shows *H $$ (i,j) = 1/sqrt 2*
⟨*proof*⟩

lemma *H-values-right-bottom*:
fixes *i j*:: *nat*
assumes $i = 1 \wedge j = 1$
shows *H $$ (i,j) = − 1/sqrt 2*
⟨*proof*⟩

The controlled-NOT gate

definition *CNOT* ::*complex mat* **where**
CNOT ≡ *mat 4 4*
(λ(*i,j*). *if i=0 ∧ j=0 then 1 else*
(*if i=1 ∧ j=1 then 1 else*
(*if i=2 ∧ j=3 then 1 else*
(*if i=3 ∧ j=2 then 1 else 0*))))

lemma *dagger-of-CNOT* [*simp*]:
$CNOT^\dagger = CNOT$
⟨*proof*⟩

lemma *CNOT-inv* [*simp*]:
$CNOT * CNOT = 1_m\ 4$
⟨*proof*⟩

lemma *CNOT-is-gate* [*simp*]:
gate 2 CNOT
⟨*proof*⟩

The phase gate, also known as the S-gate

definition *S* ::*complex mat* **where**
S ≡ *mat 2 2* (λ(*i,j*). *if i=0 ∧ j=0 then 1 else* (*if i=1 ∧ j=1 then* i *else 0*))

The $\pi/8$ gate, also known as the T-gate

definition *T* ::*complex mat* **where**
T ≡ *mat 2 2* (λ(*i,j*). *if i=0 ∧ j=0 then 1 else* (*if i=1 ∧ j=1 then exp*(i∗(*pi*/4)) *else 0*))

A few relations between the Hadamard gate and the Pauli matrices

lemma *HXH-is-Z* [*simp*]:
$H * X * H = Z$
⟨*proof*⟩

lemma *HYH-is-minusY* [*simp*]:
$H * Y * H = - Y$
⟨*proof*⟩

lemma *HZH-is-X* [*simp*]:
shows $H * Z * H = X$
⟨*proof*⟩

3.8 The Bell States

We introduce below the so-called Bell states, also known as EPR pairs (EPR stands for Einstein, Podolsky and Rosen).

definition *bell00* ::*complex mat* (‹$|\beta_{00}\rangle$›) **where**
bell00 ≡ *1/sqrt(2)* ·$_m$ |*vec 4* (λ*i*. *if i=0* ∨ *i=3 then 1 else 0*)⟩

definition *bell01* ::*complex mat* (‹$|\beta_{01}\rangle$›) **where**
bell01 ≡ *1/sqrt(2)* ·$_m$ |*vec 4* (λ*i*. *if i=1* ∨ *i=2 then 1 else 0*)⟩

definition *bell10* ::*complex mat* (‹$|\beta_{10}\rangle$›) **where**
bell10 ≡ *1/sqrt(2)* ·$_m$ |*vec 4* (λ*i*. *if i=0 then 1 else if i=3 then −1 else 0*)⟩

definition *bell11* ::*complex mat* (‹$|\beta_{11}\rangle$›) **where**
bell11 ≡ *1/sqrt(2)* ·$_m$ |*vec 4* (λ*i*. *if i=1 then 1 else if i=2 then −1 else 0*)⟩

lemma
shows *bell00-is-state* [*simp*]:*state 2* $|\beta_{00}\rangle$ **and** *bell01-is-state* [*simp*]:*state 2* $|\beta_{01}\rangle$
and
bell10-is-state [*simp*]:*state 2* $|\beta_{10}\rangle$ **and** *bell11-is-state* [*simp*]:*state 2* $|\beta_{11}\rangle$
⟨*proof*⟩

lemma *bell00-index* [*simp*]:
shows $|\beta_{00}\rangle$ $$ (*0*,*0*) = *1/sqrt 2* **and** $|\beta_{00}\rangle$ $$ (*1*,*0*) = *0* **and** $|\beta_{00}\rangle$ $$ (*2*,*0*) = *0* **and**
$|\beta_{00}\rangle$ $$ (*3*,*0*) = *1/sqrt 2*
⟨*proof*⟩

lemma *bell01-index* [*simp*]:
shows $|\beta_{01}\rangle$ $$ (*0*,*0*) = *0* **and** $|\beta_{01}\rangle$ $$ (*1*,*0*) = *1/sqrt 2* **and** $|\beta_{01}\rangle$ $$ (*2*,*0*) = *1/sqrt 2* **and**
$|\beta_{01}\rangle$ $$ (*3*,*0*) = *0*
⟨*proof*⟩

lemma *bell10-index* [*simp*]:
shows $|\beta_{10}\rangle$ $$ (*0*,*0*) = *1/sqrt 2* **and** $|\beta_{10}\rangle$ $$ (*1*,*0*) = *0* **and** $|\beta_{10}\rangle$ $$ (*2*,*0*) = *0* **and**
$|\beta_{10}\rangle$ $$ (*3*,*0*) = *− 1/sqrt 2*
⟨*proof*⟩

lemma *bell-11-index* [*simp*]:
shows $|\beta_{11}\rangle$ $$ (0,0) = 0 **and** $|\beta_{11}\rangle$ $$ (1,0) = 1/sqrt 2 **and** $|\beta_{11}\rangle$ $$ (2,0) = − 1/sqrt 2 **and**
$|\beta_{11}\rangle$ $$ (3,0) = 0
⟨*proof*⟩

3.9 The Bitwise Inner Product

definition *bitwise-inner-prod*:: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
bitwise-inner-prod n i j = (∑ *k*∈{0..<*n*}. (*bin-rep n i*) ! *k* ∗ (*bin-rep n j*) ! *k*)

abbreviation *bip*:: *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* (‹- ·_ -›) **where**
bip i n j ≡ *bitwise-inner-prod n i j*

lemma *bitwise-inner-prod-fst-el-0*:
assumes *i* < 2^*n* ∨ *j* < 2^*n*
shows (*i* ·$_{Suc\ n}$ *j*) = (*i mod* 2^*n*) ·$_n$ (*j mod* 2^*n*)
⟨*proof*⟩

lemma *bitwise-inner-prod-fst-el-is-1*:
fixes *n i j*:: *nat*
assumes *i* ≥ 2^*n* ∧ *j* ≥ 2^*n* **and** *i* < 2^(*n*+1) ∧ *j* < 2^(*n*+1)
shows (*i* ·$_{(n+1)}$ *j*) = 1 + ((*i mod* 2^*n*) ·$_n$ (*j mod* 2^*n*))
⟨*proof*⟩

lemma *bitwise-inner-prod-with-zero*:
assumes *m* < 2^*n*
shows (0 ·$_n$ *m*) = 0
⟨*proof*⟩

end

4 Complex Vectors

theory *Complex-Vectors*
imports
Quantum
VectorSpace.VectorSpace
begin

4.1 The Vector Space of Complex Vectors of Dimension n

definition *module-cpx-vec*:: *nat* ⇒ (*complex*, *complex vec*) *module* **where**
module-cpx-vec n ≡ *module-vec TYPE*(*complex*) *n*

definition *cpx-rng*:: *complex ring* **where**
cpx-rng ≡ (|*carrier* = *UNIV*, *mult* = (∗), *one* = 1, *zero* = 0, *add* = (+)|)

lemma *cpx-cring-is-field* [*simp*]:
field cpx-rng
⟨*proof*⟩

lemma *cpx-abelian-monoid* [*simp*]:
abelian-monoid cpx-rng
⟨*proof*⟩

lemma *vecspace-cpx-vec* [*simp*]:
vectorspace cpx-rng (*module-cpx-vec n*)
⟨*proof*⟩

lemma *module-cpx-vec* [*simp*]:
Module.module cpx-rng (*module-cpx-vec n*)
⟨*proof*⟩

definition *state-basis*:: *nat* ⇒ *nat* ⇒ *complex vec* **where**
state-basis n i ≡ *unit-vec* (*2*^*n*) *i*

definition *unit-vectors*:: *nat* ⇒ (*complex vec*) *set* **where**
unit-vectors n ≡ {*unit-vec n i* | *i*::*nat*. *0* ≤ *i* ∧ *i* < *n*}

lemma *unit-vectors-carrier-vec* [*simp*]:
unit-vectors n ⊆ *carrier-vec n*
⟨*proof*⟩

lemma (**in** *Module.module*) *finsum-over-singleton* [*simp*]:
assumes *f x* ∈ *carrier M*
shows *finsum M f* {*x*} = *f x*
⟨*proof*⟩

lemma *lincomb-over-singleton* [*simp*]:
assumes *x* ∈ *carrier-vec n* **and** *f* ∈ {*x*} → *UNIV*
shows *module.lincomb* (*module-cpx-vec n*) *f* {*x*} = *f x* ·$_v$ *x*
⟨*proof*⟩

lemma *dim-vec-lincomb* [*simp*]:
assumes *finite F* **and** *f*: *F* → *UNIV* **and** *F* ⊆ *carrier-vec n*
shows *dim-vec* (*module.lincomb* (*module-cpx-vec n*) *f F*) = *n*
⟨*proof*⟩

lemma *lincomb-vec-index* [*simp*]:
assumes *finite F* **and** *a2*:*i* < *n* **and** *F* ⊆ *carrier-vec n* **and** *f*: *F* → *UNIV*
shows *module.lincomb* (*module-cpx-vec n*) *f F* $ *i* = (∑ *v*∈*F*. *f v* ∗ (*v* $ *i*))
⟨*proof*⟩

lemma *unit-vectors-is-lin-indpt* [*simp*]:
module.lin-indpt cpx-rng (*module-cpx-vec n*) (*unit-vectors n*)
⟨*proof*⟩

lemma *unit-vectors-is-genset* [*simp*]:
module.gen-set cpx-rng (*module-cpx-vec n*) (*unit-vectors n*)
⟨*proof*⟩

lemma *unit-vectors-is-basis* [*simp*]:
vectorspace.basis cpx-rng (*module-cpx-vec n*) (*unit-vectors n*)
⟨*proof*⟩

lemma *state-qbit-is-lincomb* [*simp*]:
state-qbit n =
{*module.lincomb* (*module-cpx-vec* (*2^n*)) *a A*|*a A*.
finite A ∧ *A*⊆(*unit-vectors* (*2^n*)) ∧ *a*∈ *A* → *UNIV* ∧ ‖*module.lincomb* (*module-cpx-vec* (*2^n*)) *a A*‖ = *1*}
⟨*proof*⟩

end

5 Tensor Products

theory *Tensor*
imports
Complex-Vectors
Matrix-Tensor.Matrix-Tensor
Jordan-Normal-Form.Matrix
begin

There is already a formalization of tensor products in the Archive of Formal Proofs, namely Matrix_Tensor.thy in Tensor Product of Matrices[1] by T.V.H. Prathamesh, but it does not build on top of the formalization of vectors and matrices given in Matrices, Jordan Normal Forms, and Spectral Radius Theory[2] by René Thiemann and Akihisa Yamada. In the present theory our purpose consists in giving such a formalization. Of course, we will reuse Prathamesh's code as much as possible, and in order to achieve that we formalize some lemmas that translate back and forth between vectors (resp. matrices) seen as lists (resp. lists of lists) and vectors (resp. matrices) as formalized in [2].

5.1 The Kronecker Product of Complex Vectors

definition *tensor-vec*:: *complex Matrix.vec* ⇒ *complex Matrix.vec* ⇒ *complex Matrix.vec* (**infixl** ‹⊗› *63*)
where *tensor-vec u v* ≡ *vec-of-list* (*mult.vec-vec-Tensor* (∗) (*list-of-vec u*) (*list-of-vec v*))

5.2 The Tensor Product of Complex Matrices

To see a matrix in the sense of [2] as a matrix in the sense of [1], we convert it into its list of column vectors.

definition *mat-to-cols-list*:: *complex Matrix.mat* ⇒ *complex list list* **where**
mat-to-cols-list A = [[*A* $$ (*i*,*j*) . *i* <− [*0*..< *dim-row A*]] . *j* <− [*0*..< *dim-col A*]]

lemma *length-mat-to-cols-list* [*simp*]:
length (*mat-to-cols-list A*) = *dim-col A*
⟨*proof*⟩

lemma *length-cols-mat-to-cols-list* [*simp*]:
assumes *j* < *dim-col A*
shows *length* [*A* $$ (*i*,*j*) . *i* <− [*0*..< *dim-row A*]] = *dim-row A*
⟨*proof*⟩

lemma *length-row-mat-to-cols-list* [*simp*]:
assumes *i* < *dim-row A*
shows *length* (*row* (*mat-to-cols-list A*) *i*) = *dim-col A*
⟨*proof*⟩

lemma *length-col-mat-to-cols-list* [*simp*]:
assumes *j* < *dim-col A*
shows *length* (*col* (*mat-to-cols-list A*) *j*) = *dim-row A*
⟨*proof*⟩

lemma *mat-to-cols-list-is-not-Nil* [*simp*]:
assumes *dim-col A* > *0*
shows *mat-to-cols-list A* ≠ []
⟨*proof*⟩

Link between Matrix_Tensor.row_length and Matrix.dim_row

lemma *row-length-mat-to-cols-list* [*simp*]:
assumes *dim-col A* > *0*
shows *mult.row-length* (*mat-to-cols-list A*) = *dim-row A*
⟨*proof*⟩

mat-to-cols-list is a matrix in the sense of *Matrix.Matrix-Legacy*.

lemma *mat-to-cols-list-is-mat* [*simp*]:
assumes *dim-col A* > *0*
shows *mat* (*mult.row-length* (*mat-to-cols-list A*)) (*length* (*mat-to-cols-list A*)) (*mat-to-cols-list A*)
⟨*proof*⟩

definition *mat-of-cols-list*:: *nat* ⇒ *complex list list* ⇒ *complex Matrix.mat* **where**
mat-of-cols-list nr cs = *Matrix.mat nr* (*length cs*) (λ (*i*,*j*). *cs* ! *j* ! *i*)

lemma *index-mat-of-cols-list* [*simp*]:

assumes $i < nr$ **and** $j <$ *length cs*
shows *mat-of-cols-list nr cs* $$ $(i,j) = cs ! j ! i$
⟨*proof*⟩

lemma *mat-to-cols-list-to-mat* [*simp*]:
mat-of-cols-list (*dim-row A*) (*mat-to-cols-list A*) = *A*
⟨*proof*⟩

lemma *plus-mult-cpx* [*simp*]:
plus-mult 1 (∗) *0* (+) (*a-inv cpx-rng*)
⟨*proof*⟩

lemma *list-to-mat-to-cols-list* [*simp*]:
fixes *l*::*complex list list*
assumes *mat nr nc l*
shows *mat-to-cols-list* (*mat-of-cols-list nr l*) = *l*
⟨*proof*⟩

lemma *col-mat-of-cols-list* [*simp*]:
assumes $j <$ *length l*
shows *Matrix.col* (*mat-of-cols-list* (*length* (*l* ! *j*)) *l*) *j* = *vec-of-list* (*l* ! *j*)
⟨*proof*⟩

definition *tensor-mat*:: [*complex Matrix.mat*, *complex Matrix.mat*] ⇒ *complex Matrix.mat* (**infixl** ‹⨂› *63*) **where**
tensor-mat A B ≡
mat-of-cols-list (*dim-row A* ∗ *dim-row B*) (*mult.Tensor* (∗) (*mat-to-cols-list A*) (*mat-to-cols-list B*))

lemma *dim-row-tensor-mat* [*simp*]:
dim-row (*A* ⨂ *B*) = *dim-row A* ∗ *dim-row B*
⟨*proof*⟩

lemma *dim-col-tensor-mat* [*simp*]:
dim-col (*A* ⨂ *B*) = *dim-col A* ∗ *dim-col B*
⟨*proof*⟩

lemma *mat-to-cols-list-nth-nth-eq-index-mat*:
‹*mat-to-cols-list A* ! *j* ! *i* = *A* $$ (*i*, *j*)› **if** ‹*i* < *dim-row A*› ‹*j* < *dim-col A*›
⟨*proof*⟩

lemma *index-tensor-mat* [*simp*]:
assumes *a1*:*dim-row A* = *rA* **and** *a2*:*dim-col A* = *cA* **and** *a3*:*dim-row B* = *rB* **and** *a4*:*dim-col B* = *cB*
and *a5*:*i* < *rA* ∗ *rB* **and** *a6*:*j* < *cA* ∗ *cB* **and** *a7*:*cA* > *0* **and** *a8*:*cB* > *0*
shows (*A* ⨂ *B*) $$ (*i*,*j*) = *A* $$ (*i div rB*, *j div cB*) ∗ *B* $$ (*i mod rB*, *j mod cB*)
⟨*proof*⟩

To go from *Matrix.row* to *Matrix-Legacy.row*

lemma *Matrix-row-is-Legacy-row*:
assumes $i < dim\text{-}row\ A$
shows $Matrix.row\ A\ i = vec\text{-}of\text{-}list\ (row\ (mat\text{-}to\text{-}cols\text{-}list\ A)\ i)$
⟨*proof*⟩

To go from *Matrix-Legacy.row* to *Matrix.row*

lemma *Legacy-row-is-Matrix-row*:
assumes $i < mult.row\text{-}length\ A$
shows $row\ A\ i = list\text{-}of\text{-}vec\ (Matrix.row\ (mat\text{-}of\text{-}cols\text{-}list\ (mult.row\text{-}length\ A)\ A)\ i)$
⟨*proof*⟩

To go from *Matrix.col* to *Matrix-Legacy.col*

lemma *Matrix-col-is-Legacy-col*:
assumes $j < dim\text{-}col\ A$
shows $Matrix.col\ A\ j = vec\text{-}of\text{-}list\ (col\ (mat\text{-}to\text{-}cols\text{-}list\ A)\ j)$
⟨*proof*⟩

To go from *Matrix-Legacy.col* to *Matrix.col*

lemma *Legacy-col-is-Matrix-col*:
assumes $a1{:}j < length\ A$ **and** $a2{:}length\ (A\ !\ j) = mult.row\text{-}length\ A$
shows $col\ A\ j = list\text{-}of\text{-}vec\ (Matrix.col\ (mat\text{-}of\text{-}cols\text{-}list\ (mult.row\text{-}length\ A)\ A)\ j)$
⟨*proof*⟩

Link between *plus-mult.scalar-product* and $(\cdot)$

lemma *scalar-prod-is-Matrix-scalar-prod* [*simp*]:
fixes $u{::}complex\ list$ **and** $v{::}complex\ list$
assumes $length\ u = length\ v$
shows $plus\text{-}mult.scalar\text{-}product\ (*)\ 0\ (+)\ u\ v = (vec\text{-}of\text{-}list\ u) \cdot (vec\text{-}of\text{-}list\ v)$
⟨*proof*⟩

Link between $(*)$ and $\lambda f\ zer\ g\ M1.\ mat\text{-}multI\ zer\ g\ f\ (mult.row\text{-}length\ M1)\ M1$

lemma *matrix-mult-to-times-mat*:
assumes $dim\text{-}col\ A > 0$ **and** $dim\text{-}col\ B > 0$ **and** $dim\text{-}col\ (A{::}complex\ Matrix.mat) = dim\text{-}row\ B$
shows $A * B = mat\text{-}of\text{-}cols\text{-}list\ (dim\text{-}row\ A)\ (plus\text{-}mult.matrix\text{-}mult\ (*)\ 0\ (+)\ (mat\text{-}to\text{-}cols\text{-}list\ A)\ (mat\text{-}to\text{-}cols\text{-}list\ B))$
⟨*proof*⟩

lemma *mat-to-cols-list-times-mat* [*simp*]:
assumes $dim\text{-}col\ A = dim\text{-}row\ B$ **and** $dim\text{-}col\ A > 0$
shows $mat\text{-}to\text{-}cols\text{-}list\ (A * B) = plus\text{-}mult.matrix\text{-}mult\ (*)\ 0\ (+)\ (mat\text{-}to\text{-}cols\text{-}list\ A)\ (mat\text{-}to\text{-}cols\text{-}list\ B)$
⟨*proof*⟩

Finally, we prove that the tensor product of complex matrices is distributive over the multiplication of complex matrices.

lemma *mult-distr-tensor*:
assumes *a1*:*dim-col A* = *dim-row B* **and** *a2*:*dim-col C* = *dim-row D* **and** *a3*:*dim-col A* > *0* **and**
a4:*dim-col B* > *0* **and** *a5*:*dim-col C* > *0* **and** *a6*:*dim-col D* > *0*
shows $(A * B) \bigotimes (C * D) = (A \bigotimes C) * (B \bigotimes D)$
⟨*proof*⟩

lemma *tensor-mat-is-assoc*:
fixes *A B C*:: *complex Matrix.mat*
shows $A \bigotimes (B \bigotimes C) = (A \bigotimes B) \bigotimes C$
⟨*proof*⟩

end

6 Further Results on Tensor Products

theory *More-Tensor*
imports
Quantum
Tensor
Jordan-Normal-Form.Matrix
Basics
begin

lemma *tensor-prod-2* [*simp*]:
mult.vec-vec-Tensor (∗) [*x1*::*complex*,*x2*] [*x3*, *x4*] = [*x1* ∗ *x3*, *x1* ∗ *x4*, *x2* ∗ *x3*, *x2* ∗ *x4*]
⟨*proof*⟩

lemma *list-vec* [*simp*]:
assumes *v* ∈ *state-qbit 1*
shows *list-of-vec v* = [*v* $ *0*, *v* $ *1*]
⟨*proof*⟩

lemma *vec-tensor-prod-2* [*simp*]:
assumes *v* ∈ *state-qbit 1* **and** *w* ∈ *state-qbit 1*
shows *v* ⊗ *w* = *vec-of-list* [*v* $ *0* ∗ *w* $ *0*, *v* $ *0* ∗ *w* $ *1*, *v* $ *1* ∗ *w* $ *0*, *v* $ *1* ∗ *w* $ *1*]
⟨*proof*⟩

lemma *vec-dim-of-vec-of-list* [*simp*]:
assumes *length l* = *n*
shows *dim-vec* (*vec-of-list l*) = *n*
⟨*proof*⟩

lemma *vec-tensor-prod-2-bis* [*simp*]:
assumes *v* ∈ *state-qbit 1* **and** *w* ∈ *state-qbit 1*
shows *v* ⊗ *w* = *Matrix.vec 4* (λ*i*. *if i* = *0 then v* $ *0* ∗ *w* $ *0 else*

*if i = 3 then v $ 1 * w $ 1 else*
*if i = 1 then v $ 0 * w $ 1 else v $ 1 * w $ 0)*

⟨*proof*⟩

lemma *index-col-mat-of-cols-list* [*simp*]:
assumes *i < n* **and** *j < length l*
shows *Matrix.col* (*mat-of-cols-list n l*) *j $ i = l ! j ! i*
⟨*proof*⟩

lemma *multTensor2* [*simp*]:
assumes *a1*:*A = Matrix.mat 2 1* (λ(*i*,*j*). *if i = 0 then a0 else a1*) **and**
a2:*B = Matrix.mat 2 1* (λ(*i*,*j*). *if i = 0 then b0 else b1*)
shows *mult.Tensor* (*) (*mat-to-cols-list A*) (*mat-to-cols-list B*) = [[*a0*b0*, *a0*b1*, *a1*b0*, *a1*b1*]]
⟨*proof*⟩

lemma *multTensor2-bis* [*simp*]:
assumes *a1*:*dim-row A = 2* **and** *a2*:*dim-col A = 1* **and** *a3*:*dim-row B = 2* **and** *a4*:*dim-col B = 1*
shows *mult.Tensor* (*) (*mat-to-cols-list A*) (*mat-to-cols-list B*) =
[[*A $$ (0,0) * B $$ (0,0)*, *A $$ (0,0) * B $$ (1,0)*, *A $$ (1,0) * B $$ (0,0)*, *A $$ (1,0) * B $$ (1,0)*]]
⟨*proof*⟩

lemma *mat-tensor-prod-2-prelim* [*simp*]:
assumes *state 1 v* **and** *state 1 w*
shows *v* ⨂ *w = mat-of-cols-list 4*
[[*v $$ (0,0) * w $$ (0,0)*, *v $$ (0,0) * w $$ (1,0)*, *v $$ (1,0) * w $$ (0,0)*, *v $$ (1,0) * w $$ (1,0)*]]
⟨*proof*⟩

lemma *mat-tensor-prod-2-col* [*simp*]:
assumes *state 1 v* **and** *state 1 w*
shows *Matrix.col* (*v* ⨂ *w*) *0 = Matrix.col v 0* ⊗ *Matrix.col w 0*
⟨*proof*⟩

lemma *mat-tensor-prod-2* [*simp*]:
assumes *state 1 v* **and** *state 1 w*
shows *v* ⨂ *w = Matrix.mat 4 1* (λ(*i*,*j*). *if i = 0 then v $$ (0,0) * w $$ (0,0)*
else
*if i = 3 then v $$ (1,0) * w $$ (1,0) else*
*if i = 1 then v $$ (0,0) * w $$ (1,0) else*
*v $$ (1,0) * w $$ (0,0))*

⟨*proof*⟩

lemma *mat-tensor-prod-2-bis*:
assumes *state 1 v* **and** *state 1 w*
shows *v* ⨂ *w* = |*Matrix.vec 4* (λ*i*. *if i = 0 then v $$ (0,0) * w $$ (0,0) else*

*if i = 3 then v $$ (1,0) * w $$ (1,0) else*
*if i = 1 then v $$ (0,0) * w $$ (1,0) else*
*v $$ (1,0) * w $$ (0,0))⟩*

⟨*proof*⟩

lemma *eq-ket-vec*:
fixes *u v*:: *complex Matrix.vec*
assumes *u = v*
shows *|u⟩ = |v⟩*
⟨*proof*⟩

lemma *mat-tensor-ket-vec*:
assumes *state 1 v* **and** *state 1 w*
shows *v ⨂ w = |(Matrix.col v 0) ⊗ (Matrix.col w 0)⟩*
⟨*proof*⟩

The property of being a state (resp. a gate) is preserved by tensor product.

lemma *tensor-state2* [*simp*]:
assumes *state 1 u* **and** *state 1 v*
shows *state 2 (u ⨂ v)*
⟨*proof*⟩

lemma *sum-prod*:
fixes *f*::*nat ⇒ complex* **and** *g*::*nat ⇒ complex*
shows *(∑ i<a*b. f(i div b) * g(i mod b)) = (∑ i<a. f(i)) * (∑ j<b. g(j))*
⟨*proof*⟩

lemma *tensor-state* [*simp*]:
assumes *state m u* **and** *state n v*
shows *state (m + n) (u ⨂ v)*
⟨*proof*⟩

lemma *dim-row-of-tensor-gate*:
assumes *gate m G1* **and** *gate n G2*
shows *dim-row (G1 ⨂ G2) = 2^(m+n)*
⟨*proof*⟩

lemma *tensor-gate-sqr-mat*:
assumes *gate m G1* **and** *gate n G2*
shows *square-mat (G1 ⨂ G2)*
⟨*proof*⟩

lemma *dim-row-of-one-mat-less-pow*:
assumes *gate m G1* **and** *gate n G2* **and** *i < dim-row* $(1_m(\textit{dim-col G1} * \textit{dim-col G2}))$
shows *i < 2^(m+n)*
⟨*proof*⟩

lemma *dim-col-of-one-mat-less-pow*:

assumes *gate m G1* **and** *gate n G2* **and** $j <$ *dim-col* (1_m(*dim-col G1* $*$ *dim-col G2*))
shows $j < 2\hat{}(m+n)$
⟨*proof*⟩

lemma *index-tensor-gate-unitary1*:
assumes *gate m G1* **and** *gate n G2* **and** $i <$ *dim-row* (1_m(*dim-col G1* $*$ *dim-col G2*)) **and**
$j <$ *dim-col* (1_m(*dim-col G1* $*$ *dim-col G2*))
shows ((*G1* ⊗ *G2*)† $*$ (*G1* ⊗ *G2*)) $$ (*i*, *j*) = 1_m(*dim-col G1* $*$ *dim-col G2*) $$ (*i*, *j*)
⟨*proof*⟩

lemma *tensor-gate-unitary1* [*simp*]:
assumes *gate m G1* **and** *gate n G2*
shows (*G1* ⊗ *G2*)† $*$ (*G1* ⊗ *G2*) = 1_m(*dim-col G1* $*$ *dim-col G2*)
⟨*proof*⟩

lemma *index-tensor-gate-unitary2* [*simp*]:
assumes *gate m G1* **and** *gate n G2* **and** $i <$ *dim-row* (1_m (*dim-col G1* $*$ *dim-col G2*)) **and**
$j <$ *dim-col* (1_m (*dim-col G1* $*$ *dim-col G2*))
shows ((*G1* ⊗ *G2*) $*$ ((*G1* ⊗ *G2*)†)) $$ (*i*, *j*) = 1_m(*dim-col G1* $*$ *dim-col G2*) $$ (*i*, *j*)
⟨*proof*⟩

lemma *tensor-gate-unitary2* [*simp*]:
assumes *gate m G1* **and** *gate n G2*
shows (*G1* ⊗ *G2*) $*$ ((*G1* ⊗ *G2*)†) = 1_m(*dim-col G1* $*$ *dim-col G2*)
⟨*proof*⟩

lemma *tensor-gate* [*simp*]:
assumes *gate m G1* **and** *gate n G2*
shows *gate* (*m* + *n*) (*G1* ⊗ *G2*)
⟨*proof*⟩

end

7 Measurement

theory *Measurement*
imports
Quantum
begin

Given an element v such that *state n v*, its components *v* $ *i* (when v is seen as a vector, v being a matrix column) for $0 \leq i < n$ have to be understood as the coefficients of the representation of v in the basis given by the unit vectors of dimension 2^n, unless stated otherwise. Such a vector v is a state

for a quantum system of n qubits. In the literature on quantum computing, for $n = 1$, i.e. for a quantum system of 1 qubit, the elements of the so-called computational basis are denoted $|0\rangle$,$|1\rangle$, and these last elements might be understood for instance as $(1, 0)$,$(0, 1)$, i.e. as the zeroth and the first elements of a given basis ; for $n = 2$, i.e. for a quantum system of 2 qubits, the elements of the computational basis are denoted $|00\rangle$, $|01\rangle$, $|10\rangle$,$|11\rangle$, and they might be understood for instance as $(1, 0, 0, 0)$, $(0, 1, 0, 0)$, $(0, 0, 1, 0)$, $(0, 0, 0, 1)$; and so on for higher values of n. The idea behind these standard notations is that the labels on the vectors of the computational basis are the binary expressions of the natural numbers indexing the elements in a given ordered basis interpreting the computational basis in a specific context, another point of view is that the order of the basis corresponds to the lexicographic order for the labels. Those labels also represent the possible outcomes of a measurement of the n qubits of the system, while the squared modules of the corresponding coefficients represent the probabilities for those outcomes. The fact that the vector v has to be normalized expresses precisely the fact that the squared modules of the coefficients represent some probabilities and hence their sum should be 1. Note that in the case of a system with multiple qubits, i.e. $n \geq 2$, one can model the simultaneous measurement of multiple qubits by sequential measurements of single qubits. Indeed, this last process leads to the same probabilities for the various possible outcomes. Given a system with n-qubits and i the index of one qubit among the n qubits of the system, where $0 \leq i \leq n - 1$ (i.e. we start the indexing from 0), we want to find the indices of the states of the computational basis whose labels have a 1 at the ith spot (counting from 0). For instance, if $n = 3$ and $i = 2$ then 1,3,5,7 are the indices of the elements of the computational basis with a 1 at the 2nd spot, namely $|001\rangle$,$|011\rangle$,$|101\rangle$,$|111\rangle$. To achieve that we define the predicate *select-index* below.

definition *select-index* ::*nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
select-index n i j ≡ (*i*≤*n*−*1*) ∧ (*j*≤*2*^*n* − *1*) ∧ (*j mod 2*^(*n*−*i*) ≥ *2*^(*n*−*1*−*i*))

lemma *select-index-union*:
{*k*| *k*::*nat*. *select-index n i k*} ∪ {*k*| *k*::*nat*. (*k*<*2*^*n*) ∧ ¬ *select-index n i k*} = {*0*..<*2*^*n*::*nat*}
⟨*proof*⟩

lemma *select-index-inter*:
{*k*| *k*::*nat*. *select-index n i k*} ∩ {*k*| *k*::*nat*. (*k*<*2*^*n*) ∧ ¬ *select-index n i k*} = {}
⟨*proof*⟩

lemma *outcomes-sum* [*simp*]:
fixes *f* :: *nat* ⇒ *real*
shows
(∑*j*∈{*k*| *k*::*nat*. *select-index n i k*}. (*f j*)) +

$(\sum j{\in}\{k|\ k{::}nat.\ (k{<}2\hat{}n) \wedge \neg\ select\text{-}index\ n\ i\ k\}.\ (f\ j)) =$
$(\sum j{\in}\{0..{<}2\hat{}n{::}nat\}.\ (f\ j))$
⟨*proof*⟩

Given a state v of a n-qbit system, we compute the probability that a measure of qubit i has the outcome 1.

definition *prob1* ::*nat* ⇒ *complex mat* ⇒ *nat* ⇒ *real* **where**
prob1 n v i ≡ $\sum j{\in}\{k|\ k{::}nat.\ select\text{-}index\ n\ i\ k\}.\ (cmod(v\ \$\$\ (j,0)))^2$

definition *prob0* ::*nat* ⇒ *complex mat* ⇒ *nat* ⇒ *real* **where**
prob0 n v i ≡ $\sum j{\in}\{k|\ k{::}nat.\ (k{<}2\hat{}n) \wedge \neg\ select\text{-}index\ n\ i\ k\}.\ (cmod(v\ \$\$\ (j,0)))^2$

lemma
shows *prob1-geq-zero*:*prob1 n v i* ≥ *0* **and** *prob0-geq-zero*:*prob0 n v i* ≥ *0*
⟨*proof*⟩

lemma *prob-sum-is-one* [*simp*]:
assumes *state n v*
shows *prob1 n v i* + *prob0 n v i* = *1*
⟨*proof*⟩

lemma
assumes *state n v*
shows *prob1-leq-one*:*prob1 n v i* ≤ *1* **and** *prob0-leq-one*:*prob0 n v i* ≤ *1*
⟨*proof*⟩

lemma *prob0-is-prob*:
assumes *state n v*
shows *prob0 n v i* ≥ *0* ∧ *prob0 n v i* ≤ *1*
⟨*proof*⟩

lemma *prob1-is-prob*:
assumes *state n v*
shows *prob1 n v i* ≥ *0* ∧ *prob1 n v i* ≤ *1*
⟨*proof*⟩

Below we give the new state of a n-qubits system after a measurement of the ith qubit gave 0.

definition *post-meas0* ::*nat* ⇒ *complex mat* ⇒ *nat* ⇒ *complex mat* **where**
post-meas0 n v i ≡
of-real(*1*/*sqrt*(*prob0 n v i*)) $\cdot_m$ |*vec* (*2*^*n*) (λ*j*. *if* ¬ *select-index n i j then v* $$ (*j*,*0*) *else 0*)⟩

Note that a division by 0 never occurs. Indeed, if *sqrt*(*prob0 n v i*) would be 0 then *prob0 n v i* would be 0 and it would mean that the measurement of the ith qubit gave 1.

lemma *post-meas0-is-state* [*simp*]:
assumes *state n v* **and** *prob0 n v i* ≠ *0*

shows *state n* (*post-meas0 n v i*)
⟨*proof*⟩

Below we give the new state of a n-qubits system after a measurement of the ith qubit gave 1.

definition *post-meas1* ::*nat* ⇒ *complex mat* ⇒ *nat* ⇒ *complex mat* **where**
post-meas1 n v i ≡
of-real(*1*/*sqrt*(*prob1 n v i*)) ·$_m$ |*vec* (*2*^*n*) (λ*j*. *if select-index n i j then v* $$ (*j*,*0*) *else 0*)⟩

Note that a division by 0 never occurs. Indeed, if *sqrt*(*prob1 n v i*) would be 0 then *prob1 n v i* would be 0 and it would mean that the measurement of the ith qubit gave 0.

lemma *post-meas-1-is-state* [*simp*]:
assumes *state n v* **and** *prob1 n v i* ≠ *0*
shows *state n* (*post-meas1 n v i*)
⟨*proof*⟩

The measurement operator below takes a number of qubits n, a state v of a n-qubits system, a number i corresponding to the index (starting from 0) of one qubit among the n-qubits, and it computes a list whose first (resp. second) element is the pair made of the probability that the outcome of the measurement of the ith qubit is 0 (resp. 1) and the corresponding post-measurement state of the system. Of course, note that i should be strictly less than n and v should be a state of dimension n, i.e. state n v should hold".

definition *meas* ::*nat* ⇒ *complex mat* ⇒ *nat* ⇒ *-list* **where**
meas n v i ≡ [(*prob0 n v i*, *post-meas0 n v i*), (*prob1 n v i*, *post-meas1 n v i*)]

We want to determine the probability that the first n qubits of an n+1 qubit system are 0. For this we need to find the indices of the states of the computational basis whose labels do not have a 1 at spot $i = 0, ..., n$.

definition *prob0-fst-qubits*:: *nat* ⇒ *complex Matrix.mat* ⇒ *real* **where**
prob0-fst-qubits n v ≡
$\sum$ *j*∈{*k*| *k*::*nat*. (*k*<*2*^(*n*+*1*)) ∧ (∀ *i*∈{*0*..<*n*}. ¬ *select-index* (*n*+*1*) *i k*)}. (*cmod*(*v* $$ (*j*,*0*)))^2

lemma *select-index-div-2*:
fixes *n i j*::*nat*
assumes *i* < *2*^(*n*+*1*) **and** *j*<*n*
shows *select-index n j* (*i div 2*) = *select-index* (*n*+*1*) *j i*
⟨*proof*⟩

lemma *select-index-suc-even*:
fixes *n k i*:: *nat*
assumes *k* < *2*^*n* **and** *select-index n i k*
shows *select-index* (*Suc n*) *i* (*2*∗*k*)

⟨*proof*⟩

lemma *select-index-suc-odd*:
fixes *n k i*:: *nat*
assumes $k \leq 2\hat{}n - 1$ **and** *select-index n i k*
shows *select-index* (*Suc n*) *i* (2∗*k*+1)
⟨*proof*⟩

lemma *aux-range*:
fixes *k*:: *nat*
assumes $k < 2\hat{}(Suc\ n + 1)$ **and** $k \geq 2$
shows $k = 2 \vee k = 3 \vee (\exists l.\ l \geq 2 \wedge l \leq 2\hat{}(n+1) - 1 \wedge (k = 2*l \vee k = 2*l + 1))$
⟨*proof*⟩

lemma *select-index-with-1*:
fixes *n*:: *nat*
assumes $n \geq 1$
shows $\forall k.\ k < 2\hat{}(n+1) \longrightarrow k \geq 2 \longrightarrow (\exists i{<}n.$ *select-index* $(n+1)\ i\ k)$
⟨*proof*⟩

lemma *prob0-fst-qubits-index*:
fixes *n*:: *nat* **and** *v*:: *complex Matrix.mat*
shows $\{k|\ k{::}nat.\ (k{<}2\hat{}(n+1)) \wedge (\forall i{\in}\{0..{<}n\}.\ \neg$ *select-index* $(n+1)\ i\ k)\} = \{0,1\}$
⟨*proof*⟩

lemma *prob0-fst-qubits-eq*:
fixes *n*:: *nat*
shows *prob0-fst-qubits n v* = $(cmod(v\ \$\$\ (0,0)))^2 + (cmod(v\ \$\$\ (1,0)))^2$
⟨*proof*⟩

primrec *iter-post-meas0*:: *nat* ⇒ *nat* ⇒ *complex Matrix.mat* ⇒ *complex Matrix.mat* **where**
iter-post-meas0 n 0 v = *v*
| *iter-post-meas0 n* (*Suc m*) *v* = *post-meas0 n* (*iter-post-meas0 n m v*) *m*

definition *iter-prob0*:: *nat* ⇒ *nat* ⇒ *complex Matrix.mat* ⇒ *real* **where**
iter-prob0 n m v = $(\prod i{<}m.$ *prob0 n* (*iter-post-meas0 n i v*) *i*)

7.1 Measurements with Bell States

A Bell state is a remarkable state. Indeed, if one makes one measure, either of the first or the second qubit, then one gets either 0 with probability 1/2 or 1 with probability 1/2. Moreover, in the case of two successive measurements of the first and second qubit, the outcomes are correlated. Indeed, in the case of $|\beta_{00}\rangle$ or $|\beta_{10}\rangle$ (resp. $|\beta_{01}\rangle$ or $|\beta_{11}\rangle$) if one measures the second qubit after

a measurement of the first qubit (or the other way around) then one gets the same outcomes (resp. opposite outcomes), i.e. for instance the probability of measuring 0 for the second qubit after a measure with outcome 0 for the first qubit is 1 (resp. 0).

lemma *prob0-bell-fst* [*simp*]:
  assumes $v = |\beta_{00}\rangle \vee v = |\beta_{01}\rangle \vee v = |\beta_{10}\rangle \vee v = |\beta_{11}\rangle$
  shows *prob0 2 v 0 = 1/2*
⟨*proof*⟩

lemma *prob-1-bell-fst* [*simp*]:
  assumes $v = |\beta_{00}\rangle \vee v = |\beta_{01}\rangle \vee v = |\beta_{10}\rangle \vee v = |\beta_{11}\rangle$
  shows *prob1 2 v 0 = 1/2*
⟨*proof*⟩

lemma *prob0-bell-snd* [*simp*]:
  assumes $v = |\beta_{00}\rangle \vee v = |\beta_{01}\rangle \vee v = |\beta_{10}\rangle \vee v = |\beta_{11}\rangle$
  shows *prob0 2 v 1 = 1/2*
⟨*proof*⟩

lemma *prob-1-bell-snd* [*simp*]:
  assumes $v = |\beta_{00}\rangle \vee v = |\beta_{01}\rangle \vee v = |\beta_{10}\rangle \vee v = |\beta_{11}\rangle$
  shows *prob1 2 v 1 = 1/2*
⟨*proof*⟩

lemma *post-meas0-bell00-fst* [*simp*]:
  post-meas0 2 $|\beta_{00}\rangle$ *0 = |unit-vec 4 0⟩*
⟨*proof*⟩

lemma *post-meas0-bell00-snd* [*simp*]:
  post-meas0 2 $|\beta_{00}\rangle$ *1 = |unit-vec 4 0⟩*
⟨*proof*⟩

lemma *post-meas0-bell01-fst* [*simp*]:
  post-meas0 2 $|\beta_{01}\rangle$ *0 = |unit-vec 4 1⟩*
⟨*proof*⟩

lemma *post-meas0-bell01-snd* [*simp*]:
  post-meas0 2 $|\beta_{01}\rangle$ *1 = |unit-vec 4 2⟩*
⟨*proof*⟩

lemma *post-meas0-bell10-fst* [*simp*]:
  post-meas0 2 $|\beta_{10}\rangle$ *0 = |unit-vec 4 0⟩*
⟨*proof*⟩

lemma *post-meas0-bell10-snd* [*simp*]:
  post-meas0 2 $|\beta_{10}\rangle$ *1 = |unit-vec 4 0⟩*
⟨*proof*⟩

lemma *post-meas0-bell11-fst* [*simp*]:

post-meas0 2 $|\beta_{11}\rangle$ *0* = $|$*unit-vec 4 1*$\rangle$
⟨*proof*⟩

lemma *post-meas0-bell11-snd* [*simp*]:
post-meas0 2 $|\beta_{11}\rangle$ *1* = − $|$*unit-vec 4 2*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell00-fst* [*simp*]:
post-meas1 2 $|\beta_{00}\rangle$ *0* = $|$*unit-vec 4 3*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell00-snd* [*simp*]:
post-meas1 2 $|\beta_{00}\rangle$ *1* = $|$*unit-vec 4 3*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell01-fst* [*simp*]:
post-meas1 2 $|\beta_{01}\rangle$ *0* = $|$*unit-vec 4 2*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell01-snd* [*simp*]:
post-meas1 2 $|\beta_{01}\rangle$ *1* = $|$*unit-vec 4 1*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell10-fst* [*simp*]:
post-meas1 2 $|\beta_{10}\rangle$ *0* = − $|$*unit-vec 4 3*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell10-snd* [*simp*]:
post-meas1 2 $|\beta_{10}\rangle$ *1* = − $|$*unit-vec 4 3*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell11-fst* [*simp*]:
post-meas1 2 $|\beta_{11}\rangle$ *0* = − $|$*unit-vec 4 2*$\rangle$
⟨*proof*⟩

lemma *post-meas1-bell11-snd* [*simp*]:
post-meas1 2 $|\beta_{11}\rangle$ *1* = $|$*unit-vec 4 1*$\rangle$
⟨*proof*⟩

end

8 Quantum Entanglement

theory *Entanglement*
imports
Quantum
More-Tensor
begin

8.1 The Product States and Entangled States of a 2-qubits System

Below we add the condition that v and w are two-dimensional states, otherwise u can always be represented by the tensor product of the 1-dimensional vector *1* and u itself.

definition *prod-state2*:: *complex Matrix.mat* ⇒ *bool* **where**
prod-state2 u ≡ *if state 2 u then* ∃ *v w. state 1 v* ∧ *state 1 w* ∧ *u* = *v* ⨂ *w else undefined*

definition *entangled2*:: *complex Matrix.mat* ⇒ *bool* **where**
entangled2 u ≡ ¬ *prod-state2 u*

The Bell states are entangled states.

lemma *bell00-is-entangled2* [*simp*]:
entangled2 $|\beta_{00}\rangle$
⟨*proof*⟩

lemma *bell01-is-entangled2* [*simp*]:
entangled2 $|\beta_{01}\rangle$
⟨*proof*⟩

lemma *bell10-is-entangled2* [*simp*]:
entangled2 $|\beta_{10}\rangle$
⟨*proof*⟩

lemma *bell11-is-entangled2* [*simp*]:
entangled2 $|\beta_{11}\rangle$
⟨*proof*⟩

An entangled state is a state that cannot be broken down as the tensor product of smaller states.

definition *prod-state*:: *nat* ⇒ *complex Matrix.mat* ⇒ *bool* **where**
prod-state m u ≡ *if state m u then* ∃ *n p*::*nat*.∃ *v w. state n v* ∧ *state p w* ∧
n < *m* ∧ *p* < *m* ∧ *u* = *v* ⨂ *w else undefined*

definition *entangled*:: *nat* ⇒ *complex Matrix.mat* ⇒ *bool* **where**
entangled n v ≡ ¬ (*prod-state n v*)

lemma *sanity-check*:
¬(*entangled 2* (*mat-of-cols-list 2* [[*1*/*sqrt*(*2*), *1*/*sqrt*(*2*)]] ⨂ *mat-of-cols-list 2* [[*1*/*sqrt*(*2*), *1*/*sqrt*(*2*)]]))
⟨*proof*⟩

end

9 Quantum Teleportation

theory *Quantum-Teleportation*
imports
More-Tensor
Basics
Measurement
begin

definition *alice*:: *complex Matrix.mat* ⇒ *complex Matrix.mat* **where**
alice $\varphi \equiv (H \bigotimes Id\ 2) * ((CNOT \bigotimes Id\ 1) * (\varphi \bigotimes |\beta_{00}\rangle))$

abbreviation *M1*:: *complex Matrix.mat* **where**
M1 ≡ *mat-of-cols-list 8* [[*1*, *0*, *0*, *0*, *0*, *0*, *0*, *0*],
[*0*, *1*, *0*, *0*, *0*, *0*, *0*, *0*],
[*0*, *0*, *1*, *0*, *0*, *0*, *0*, *0*],
[*0*, *0*, *0*, *1*, *0*, *0*, *0*, *0*],
[*0*, *0*, *0*, *0*, *0*, *0*, *1*, *0*],
[*0*, *0*, *0*, *0*, *0*, *0*, *0*, *1*],
[*0*, *0*, *0*, *0*, *1*, *0*, *0*, *0*],
[*0*, *0*, *0*, *0*, *0*, *1*, *0*, *0*]]

lemma *tensor-prod-of-cnot-id-1*:
shows $(CNOT \bigotimes Id\ 1) = M1$
⟨*proof*⟩

abbreviation *M2*:: *complex Matrix.mat* **where**
M2 ≡ *mat-of-cols-list 8* [[*1/sqrt(2)*, *0*, *0*, *0*, *1/sqrt(2)*, *0*, *0*, *0*],
[*0*, *1/sqrt(2)*, *0*, *0*, *0*, *1/sqrt(2)*, *0*, *0*],
[*0*, *0*, *1/sqrt(2)*, *0*, *0*, *0*, *1/sqrt(2)*, *0*],
[*0*, *0*, *0*, *1/sqrt(2)*, *0*, *0*, *0*, *1/sqrt(2)*],
[*1/sqrt(2)*, *0*, *0*, *0*, *−1/sqrt(2)*, *0*, *0*, *0*],
[*0*, *1/sqrt(2)*, *0*, *0*, *0*, *−1/sqrt(2)*, *0*, *0*],
[*0*, *0*, *1/sqrt(2)*, *0*, *0*, *0*, *−1/sqrt(2)*, *0*],
[*0*, *0*, *0*, *1/sqrt(2)*, *0*, *0*, *0*, *−1/sqrt(2)*]]

lemma *tensor-prod-of-h-id-2*:
shows $(H \bigotimes Id\ 2) = M2$
⟨*proof*⟩

lemma *alice-step-1-state* [*simp*]:
assumes *state 1* φ
shows *state 3* $(\varphi \bigotimes |\beta_{00}\rangle)$
⟨*proof*⟩

lemma *alice-step-2-state*:
assumes *state 1* φ
shows *state 3* $((CNOT \bigotimes Id\ 1) * (\varphi \bigotimes |\beta_{00}\rangle))$

⟨*proof*⟩

lemma *alice-state* [*simp*]:
assumes *state 1* φ
shows *state 3* (*alice* φ)
⟨*proof*⟩

lemma *alice-step-1*:
assumes *state 1* φ **and** $\alpha = \varphi$ $$ (*0*,*0*) **and** $\beta = \varphi$ $$ (*1*,*0*)
shows ($\varphi \bigotimes |\beta_{00}\rangle$) = *mat-of-cols-list 8* [[α/*sqrt*(*2*),*0*,*0*,α/*sqrt*(*2*),β/*sqrt*(*2*),*0*,*0*,β/*sqrt*(*2*)]]
⟨*proof*⟩

lemma *alice-step-2*:
assumes *state 1* φ **and** $\alpha = \varphi$ $$ (*0*,*0*) **and** $\beta = \varphi$ $$ (*1*,*0*)
shows (*CNOT* $\bigotimes$ *Id 1*) $*$ ($\varphi \bigotimes |\beta_{00}\rangle$) = *mat-of-cols-list 8* [[α/*sqrt*(*2*),*0*,*0*,α/*sqrt*(*2*),*0*,β/*sqrt*(*2*),β/*sqrt*(*2*),
⟨*proof*⟩

lemma *alice-result*:
assumes *state 1* φ **and** $\alpha = \varphi$ $$ (*0*,*0*) **and** $\beta = \varphi$ $$ (*1*,*0*)
shows *alice* φ = *mat-of-cols-list 8* [[α/*2*, β/*2*, β/*2*, α/*2*, α/*2*, $-\beta$/*2*, $-\beta$/*2*,
α/*2*]]
⟨*proof*⟩

An application of function *alice* to a state φ of a 1-qubit system results in the following cases.

definition *alice-meas*:: *complex Matrix.mat* ⇒ *-list* **where**
alice-meas φ = [
((*prob0 3* (*alice* φ) *0*) $*$ (*prob0 3* (*post-meas0 3* (*alice* φ) *0*) *1*), *post-meas0 3*
(*post-meas0 3* (*alice* φ) *0*) *1*)
, ((*prob0 3* (*alice* φ) *0*) $*$ (*prob1 3* (*post-meas0 3* (*alice* φ) *0*) *1*), *post-meas1 3*
(*post-meas0 3* (*alice* φ) *0*) *1*)
, ((*prob1 3* (*alice* φ) *0*) $*$ (*prob0 3* (*post-meas1 3* (*alice* φ) *0*) *1*), *post-meas0 3*
(*post-meas1 3* (*alice* φ) *0*) *1*)
, ((*prob1 3* (*alice* φ) *0*) $*$ (*prob1 3* (*post-meas1 3* (*alice* φ) *0*) *1*), *post-meas1 3*
(*post-meas1 3* (*alice* φ) *0*) *1*)
]

definition *alice-pos*:: *complex Matrix.mat* ⇒ *complex Matrix.mat* ⇒ *bool* **where**
alice-pos φ *q* ≡ *q* = *mat-of-cols-list 8* [[φ $$ (*0*,*0*), φ $$ (*1*,*0*), *0*, *0*, *0*, *0*, *0*, *0*]]
∨
q = *mat-of-cols-list 8* [[*0*, *0*, φ $$ (*1*,*0*), φ $$ (*0*,*0*), *0*, *0*, *0*, *0*]] ∨
q = *mat-of-cols-list 8* [[*0*, *0*, *0*, *0*, φ $$ (*0*,*0*), − φ $$ (*1*,*0*), *0*, *0*]]
∨
q = *mat-of-cols-list 8* [[*0*, *0*, *0*, *0*, *0*, *0*, − φ $$ (*1*,*0*), φ $$ (*0*,*0*)]]

lemma *phi-vec-length*:
assumes *state 1* φ
shows *cmod*(φ $$ (*0*,*0*))^2 + *cmod*(φ $$ (*Suc 0*,*0*))^2 = *1*
⟨*proof*⟩

lemma *select-index-3-subsets* [*simp*]:
shows $\{j::nat.\ select\text{-}index\ 3\ 0\ j\} = \{4,5,6,7\} \wedge$
$\{j::nat.\ j < 8 \wedge \neg\ select\text{-}index\ 3\ 0\ j\} = \{0,1,2,3\} \wedge$
$\{j::nat.\ select\text{-}index\ 3\ 1\ j\} = \{2,3,6,7\} \wedge$
$\{j::nat.\ j < 8 \wedge \neg\ select\text{-}index\ 3\ 1\ j\} = \{0,1,4,5\}$
⟨*proof*⟩

lemma *prob-index-0-alice*:
assumes *state 1 φ*
shows *prob0 3 (alice φ) 0 = 1/2 ∧ prob1 3 (alice φ) 0 = 1/2*
⟨*proof*⟩

lemma *post-meas0-index-0-alice*:
assumes *state 1 φ* **and** $\alpha = \varphi$ $$ $(0,0)$ **and** $\beta = \varphi$ $$ $(1,0)$
shows *post-meas0 3 (alice φ) 0 =*
mat-of-cols-list 8 $[[\alpha/sqrt(2),\ \beta/sqrt(2),\ \beta/sqrt(2),\ \alpha/sqrt(2),\ 0,\ 0,\ 0,\ 0]]$
⟨*proof*⟩

lemma *post-meas1-index-0-alice*:
assumes *state 1 φ* **and** $\alpha = \varphi$ $$ $(0,0)$ **and** $\beta = \varphi$ $$ $(1,0)$
shows *post-meas1 3 (alice φ) 0 = mat-of-cols-list 8* $[[0,0,0,0,\alpha/sqrt(2),-\beta/sqrt(2),-\beta/sqrt(2),\alpha/sqrt(2)]]$
⟨*proof*⟩

lemma *post-meas0-index-0-alice-state* [*simp*]:
assumes *state 1 φ*
shows *state 3 (post-meas0 3 (alice φ) 0)*
⟨*proof*⟩

lemma *post-meas1-index-0-alice-state* [*simp*]:
assumes *state 1 φ*
shows *state 3 (post-meas1 3 (alice φ) 0)*
⟨*proof*⟩

lemma *Alice-case* [*simp*]:
assumes *state 1 φ* **and** *state 3 q* **and** *List.member (alice-meas φ) (p, q)*
shows *alice-pos φ q*
⟨*proof*⟩

datatype *bit = zero | one*

definition *alice-out*:: *complex Matrix.mat => complex Matrix.mat => bit × bit*
where
alice-out φ q ≡
if q = mat-of-cols-list 8 $[[\varphi$ $$ $(0,0),\ \varphi$ $$ $(1,0),\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]]$ *then (zero, zero) else*
if q = mat-of-cols-list 8 $[[0,\ 0,\ \varphi$ $$ $(1,0),\ \varphi$ $$ $(0,0),\ 0,\ 0,\ 0,\ 0]]$ *then (zero, one) else*

if q = mat-of-cols-list 8 [[0, 0, 0, 0, φ $$ (0,0), − φ $$ (1,0), 0, 0]] then (one, zero) else
if q = mat-of-cols-list 8 [[0, 0, 0, 0, 0, 0, − φ $$ (1,0), φ $$ (0,0)]] then (one, one) else
undefined

definition *bob*:: *complex Matrix.mat => bit × bit ⇒ complex Matrix.mat* **where**
bob q b ≡
if (fst b, snd b) = (zero, zero) then q else
if (fst b, snd b) = (zero, one) then (Id 2 ⨂ X) ∗ q else
if (fst b, snd b) = (one, zero) then (Id 2 ⨂ Z) ∗ q else
if (fst b, snd b) = (one, one) then (Id 2 ⨂ Z ∗ X) ∗ q else
undefined

lemma *alice-out-unique* [*simp*]:
assumes *state 1 φ*
shows *Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, φ $$ (Suc 0, 0), φ $$ (0, 0), 0, 0, 0, 0]]!j!i) ≠*
Matrix.mat 8 (Suc 0) (λ(i,j). [[φ $$ (0, 0), φ $$ (Suc 0, 0), 0, 0, 0, 0, 0, 0]]!j!i) ∧
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, 0, 0, φ $$ (0, 0), −φ $$ (Suc 0, 0), 0, 0]]!j!i) ≠
Matrix.mat 8 (Suc 0) (λ(i,j). [[φ $$ (0, 0), φ $$ (Suc 0, 0), 0, 0, 0, 0, 0, 0]]!j!i) ∧
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, 0, 0, 0, 0, −φ $$ (Suc 0, 0), φ $$ (0, 0)]]!j!i) ≠
Matrix.mat 8 (Suc 0) (λ(i,j). [[φ $$ (0, 0), φ $$ (Suc 0, 0), 0, 0, 0, 0, 0, 0]]!j!i) ∧
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, 0, 0, φ $$ (0, 0), −φ $$ (Suc 0, 0), 0, 0]]!j!i) ≠
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, φ $$ (Suc 0, 0), φ $$ (0, 0), 0, 0, 0, 0]]!j!i) ∧
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, 0, 0, 0, 0, −φ $$ (Suc 0, 0), φ $$ (0, 0)]]!j!i) ≠
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, φ $$ (Suc 0, 0), φ $$ (0, 0), 0, 0, 0, 0]]!j!i) ∧
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, 0, 0, 0, 0, −φ $$ (Suc 0, 0), φ $$ (0, 0)]]!j!i) ≠
Matrix.mat 8 (Suc 0) (λ(i,j). [[0, 0, 0, 0, φ $$ (0, 0), −φ $$ (Suc 0, 0), 0, 0]]!j!i)
⟨*proof*⟩

abbreviation *M3*:: *complex Matrix.mat* **where**
M3 ≡ mat-of-cols-list 8 [[0, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0],

$[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1],$
$[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0]]$

lemma *tensor-prod-of-id-2-x*:
shows $(Id\ 2 \bigotimes X) = M3$
⟨*proof*⟩

abbreviation *M4*:: *complex Matrix.mat* **where**
$M4 \equiv mat\text{-}of\text{-}cols\text{-}list\ 8\ [[0,\ -1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0],$
$[1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 0,\ -1,\ 0,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 0,\ 0,\ 0,\ -1,\ 0,\ 0],$
$[0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ -1],$
$[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0]]$

abbreviation *ZX*:: *complex Matrix.mat* **where**
$ZX \equiv mat\text{-}of\text{-}cols\text{-}list\ 2\ [[0,\ -1],\ [1,\ 0]]$

lemma *l-inv-of-ZX*:
shows $ZX^\dagger * ZX = 1_m\ 2$
⟨*proof*⟩

lemma *r-inv-of-ZX*:
shows $ZX * (ZX^\dagger) = 1_m\ 2$
⟨*proof*⟩

lemma *ZX-is-gate* [*simp*]:
shows *gate 1 ZX*
⟨*proof*⟩

lemma *prod-of-ZX*:
shows $Z * X = ZX$
⟨*proof*⟩

lemma *tensor-prod-of-id-2-y*:
shows $(Id\ 2 \bigotimes Z * X) = M4$
⟨*proof*⟩

abbreviation *M5*:: *complex Matrix.mat* **where**
$M5 \equiv mat\text{-}of\text{-}cols\text{-}list\ 8\ [[1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0],$
$[0,\ -1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 0,\ -1,\ 0,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 0,\ 0,\ 1,\ 0,\ 0,\ 0],$
$[0,\ 0,\ 0,\ 0,\ 0,\ -1,\ 0,\ 0],$
$[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1,\ 0],$
$[0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ -1]]$

lemma *tensor-prod-of-id-2-z*:
shows $(Id\ 2 \bigotimes Z) = M5$
⟨*proof*⟩

lemma *teleportation*:
assumes *state 1* φ **and** *state 3* q **and** *List.member* $(alice\text{-}meas\ \varphi)\ (p,\ q)$
shows $\exists r.\ state\ 2\ r \wedge bob\ q\ (alice\text{-}out\ \varphi\ q) = r \bigotimes \varphi$
⟨*proof*⟩

end

10 The Deutsch Algorithm

theory *Deutsch*
imports
More-Tensor
Measurement
begin

Given a function $f : 0, 1 \mapsto 0, 1$, Deutsch's algorithm decides if this function is constant or balanced with a single $f(x)$ circuit to evaluate the function for multiple values of x simultaneously. The algorithm makes use of quantum parallelism and quantum interference.

A constant function with values in 0,1 returns either always 0 or always 1. A balanced function is 0 for half of the inputs and 1 for the other half.

locale *deutsch* =
fixes *f*:: *nat* ⇒ *nat*
assumes *dom*: $f \in (\{0,1\} \rightarrow_E \{0,1\})$

context *deutsch*
begin

definition *is-swap*:: *bool* **where**
$is\text{-}swap = (\forall x \in \{0,1\}.\ f\ x = 1 - x)$

lemma *is-swap-values*:
assumes *is-swap*
shows $f\ 0 = 1$ **and** $f\ 1 = 0$
⟨*proof*⟩

lemma *is-swap-sum-mod-2*:
assumes *is-swap*
shows $(f\ 0 + f\ 1)\ mod\ 2 = 1$
⟨*proof*⟩

definition *const*:: *nat* ⇒ *bool* **where**
const n = (∀ *x* ∈ {*0*,*1*}.(*f x* = *n*))

definition *is-const*:: *bool* **where**
is-const ≡ *const 0* ∨ *const 1*

definition *is-balanced*:: *bool* **where**
is-balanced ≡ (∀ *x* ∈ {*0*,*1*}.(*f x* = *x*)) ∨ *is-swap*

lemma *f-values*: (*f 0* = *0* ∨ *f 0* = *1*) ∧ (*f 1* = *0* ∨ *f 1* = *1*)
⟨*proof*⟩

lemma *f-cases*:
shows *is-const* ∨ *is-balanced*
⟨*proof*⟩

lemma *const-0-sum-mod-2*:
assumes *const 0*
shows (*f 0* + *f 1*) *mod 2* = *0*
⟨*proof*⟩

lemma *const-1-sum-mod-2*:
assumes *const 1*
shows (*f 0* + *f 1*) *mod 2* = *0*
⟨*proof*⟩

lemma *is-const-sum-mod-2*:
assumes *is-const*
shows (*f 0* + *f 1*) *mod 2* = *0*
⟨*proof*⟩

lemma *id-sum-mod-2*:
assumes *f* = *id*
shows (*f 0* + *f 1*) *mod 2* = *1*
⟨*proof*⟩

lemma *is-balanced-sum-mod-2*:
assumes *is-balanced*
shows (*f 0* + *f 1*) *mod 2* = *1*
⟨*proof*⟩

lemma *f-ge-0*: ∀ *x*. (*f x* ≥ *0*) ⟨*proof*⟩

end

The Deutsch's Transform U_f.

definition (**in** *deutsch*) *deutsch-transform*:: *complex Matrix.mat* (‹U_f›) **where**
U_f ≡ *mat-of-cols-list 4* [[*1* − *f*(*0*), *f*(*0*), *0*, *0*],

$[f(0),\ 1 - f(0),\ 0,\ 0],$
$[0,\ 0,\ 1 - f(1),\ f(1)],$
$[0,\ 0,\ f(1),\ 1 - f(1)]]$

lemma (**in** *deutsch*) *deutsch-transform-dim* [*simp*]:
shows *dim-row* $U_f = 4$ **and** *dim-col* $U_f = 4$
⟨*proof*⟩

lemma (**in** *deutsch*) *deutsch-transform-coeff-is-zero* [*simp*]:
shows U_f $$ $(0,2) = 0$ **and** U_f $$ $(0,3) = 0$
and U_f $$ $(1,2) = 0$ **and** U_f $$$(1,3) = 0$
and U_f $$ $(2,0) = 0$ **and** U_f $$$(2,1) = 0$
and U_f $$ $(3,0) = 0$ **and** U_f $$ $(3,1) = 0$
⟨*proof*⟩

lemma (**in** *deutsch*) *deutsch-transform-coeff* [*simp*]:
shows U_f $$ $(0,1) = f(0)$ **and** U_f $$ $(1,0) = f(0)$
and U_f $$$(2,3) = f(1)$ **and** U_f $$ $(3,2) = f(1)$
and U_f $$ $(0,0) = 1 - f(0)$ **and** U_f $$$(1,1) = 1 - f(0)$
and U_f $$ $(2,2) = 1 - f(1)$ **and** U_f $$ $(3,3) = 1 - f(1)$
⟨*proof*⟩

abbreviation (**in** *deutsch*) V_f:: *complex Matrix.mat* **where**
$V_f \equiv$ *Matrix.mat 4 4* $(\lambda(i,j).$
if $i=0 \wedge j=0$ *then* $1 - f(0)$ *else*
(*if* $i=0 \wedge j=1$ *then* $f(0)$ *else*
(*if* $i=1 \wedge j=0$ *then* $f(0)$ *else*
(*if* $i=1 \wedge j=1$ *then* $1 - f(0)$ *else*
(*if* $i=2 \wedge j=2$ *then* $1 - f(1)$ *else*
(*if* $i=2 \wedge j=3$ *then* $f(1)$ *else*
(*if* $i=3 \wedge j=2$ *then* $f(1)$ *else*
(*if* $i=3 \wedge j=3$ *then* $1 - f(1)$ *else* 0))))))))

lemma (**in** *deutsch*) *deutsch-transform-alt-rep-coeff-is-zero* [*simp*]:
shows V_f $$ $(0,2) = 0$ **and** V_f $$ $(0,3) = 0$
and V_f $$ $(1,2) = 0$ **and** V_f $$$(1,3) = 0$
and V_f $$ $(2,0) = 0$ **and** V_f $$$(2,1) = 0$
and V_f $$ $(3,0) = 0$ **and** V_f $$ $(3,1) = 0$
⟨*proof*⟩

lemma (**in** *deutsch*) *deutsch-transform-alt-rep-coeff* [*simp*]:
shows V_f $$ $(0,1) = f(0)$ **and** V_f $$ $(1,0) = f(0)$
and V_f $$$(2,3) = f(1)$ **and** V_f $$ $(3,2) = f(1)$
and V_f $$ $(0,0) = 1 - f(0)$ **and** V_f $$$(1,1) = 1 - f(0)$
and V_f $$ $(2,2) = 1 - f(1)$ **and** V_f $$ $(3,3) = 1 - f(1)$
⟨*proof*⟩

lemma (**in** *deutsch*) *deutsch-transform-alt-rep*:
shows $U_f = V_f$

⟨*proof*⟩

U_f is a gate.

lemma (**in** *deutsch*) *transpose-of-deutsch-transform*:
shows $(U_f)^t = U_f$
⟨*proof*⟩

lemma (**in** *deutsch*) *adjoint-of-deutsch-transform*:
shows $(U_f)^\dagger = U_f$
⟨*proof*⟩

lemma (**in** *deutsch*) *deutsch-transform-is-gate*:
shows *gate 2* U_f
⟨*proof*⟩

Two qubits are prepared. The first one in the state $|0\rangle$, the second one in the state $|1\rangle$.

abbreviation *zero* **where** *zero* ≡ *unit-vec 2 0*
abbreviation *one* **where** *one* ≡ *unit-vec 2 1*

lemma *ket-zero-is-state*:
shows *state 1* $|zero\rangle$
⟨*proof*⟩

lemma *ket-one-is-state*:
shows *state 1* $|one\rangle$
⟨*proof*⟩

lemma *ket-zero-to-mat-of-cols-list* [*simp*]: $|zero\rangle$ = *mat-of-cols-list 2* [[*1*, *0*]]
⟨*proof*⟩

lemma *ket-one-to-mat-of-cols-list* [*simp*]: $|one\rangle$ = *mat-of-cols-list 2* [[*0*, *1*]]
⟨*proof*⟩

Applying the Hadamard gate to the state $|0\rangle$ results in the new state $\psi_{00} = \frac{(|0\rangle + |1\rangle)}{\sqrt{2}}$

abbreviation ψ_{00}:: *complex Matrix.mat* **where**
ψ_{00} ≡ *mat-of-cols-list 2* [[*1/sqrt(2)*, *1/sqrt(2)*]]

lemma *H-on-ket-zero*:
shows $(H * |zero\rangle) = \psi_{00}$
⟨*proof*⟩

lemma *H-on-ket-zero-is-state*:
shows *state 1* $(H * |zero\rangle)$
⟨*proof*⟩

Applying the Hadamard gate to the state $|0\rangle$ results in the new state ψ_{01} =

$\frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$.

abbreviation ψ_{01}:: *complex Matrix.mat* **where**
$\psi_{01} \equiv$ *mat-of-cols-list 2* [[*1/sqrt(2)*, *−1/sqrt(2)*]]

lemma *H-on-ket-one*:
shows $(H * |one\rangle) = \psi_{01}$
⟨*proof*⟩

lemma *H-on-ket-one-is-state*:
shows *state 1* $(H * |one\rangle)$
⟨*proof*⟩

Then, the state $\psi_1 = \frac{(|00\rangle - |01\rangle + |10\rangle - |11\rangle)}{2}$ is obtained by taking the tensor product of the states $\psi_{00} = \frac{(|0\rangle + |1\rangle)}{\sqrt{2}}$ and $\psi_{01} = \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$.

abbreviation ψ_1:: *complex Matrix.mat* **where**
$\psi_1 \equiv$ *mat-of-cols-list 4* [[*1/2*, *−1/2*, *1/2*, *−1/2*]]

lemma ψ_0-*to*-ψ_1:
shows $(\psi_{00} \bigotimes \psi_{01}) = \psi_1$
⟨*proof*⟩

lemma ψ_1-*is-state*:
shows *state 2* ψ_1
⟨*proof*⟩

Next, the gate U_f is applied to the state $\psi_1 = \frac{(|00\rangle - |01\rangle + |10\rangle - |11\rangle)}{2}$ and $\psi_2 = \frac{(|0f(0) \oplus 0\rangle - |0f(0) \oplus 1\rangle + |1f(1) \oplus 0\rangle - |1f(1) \oplus 1\rangle)}{2}$ is obtained. This simplifies to $\psi_2 = \frac{(|0f(0)\rangle - |0\overline{f(0)}\rangle + |1f(1)\rangle - |1\overline{f(1)}\rangle)}{2}$

abbreviation (**in** *deutsch*) ψ_2:: *complex Matrix.mat* **where**
$\psi_2 \equiv$ *mat-of-cols-list 4* [[*(1 − f(0))/2 − f(0)/2*,
f(0)/2 − (1 − f(0))/2,
(1 − f(1))/2 − f(1)/2,
f(1)/2 − (1− f(1))/2]]

lemma (**in** *deutsch*) ψ_1-*to*-ψ_2:
shows $U_f * \psi_1 = \psi_2$
⟨*proof*⟩

lemma (**in** *deutsch*) ψ_2-*is-state*:
shows *state 2* ψ_2
⟨*proof*⟩

lemma *H-tensor-Id-1*:

defines *d:v* ≡ *mat-of-cols-list 4* [[*1/sqrt(2)*, *0*, *1/sqrt(2)*, *0*],
[*0*, *1/sqrt(2)*, *0*, *1/sqrt(2)*],
[*1/sqrt(2)*, *0*, *−1/sqrt(2)*, *0*],
[*0*, *1/sqrt(2)*, *0*, *−1/sqrt(2)*]]
shows (*H* ⨂ *Id 1*) = *v*
⟨*proof*⟩

lemma *H-tensor-Id-1-is-gate*:
shows *gate 2* (*H* ⨂ *Id 1*)
⟨*proof*⟩

Applying the Hadamard gate to the first qubit of ψ_2 results in $\psi_3 = \pm|f(0)\oplus f(1)\rangle \left[\frac{(|0\rangle - |1\rangle)}{\sqrt{2}}\right]$

abbreviation (**in** *deutsch*) ψ_3:: *complex Matrix.mat* **where**
ψ_3 ≡ *mat-of-cols-list 4*
[[(*1−f(0)*)/(*2∗sqrt(2)*) − *f(0)*/(*2∗sqrt(2)*) + (*1−f(1)*)/(*2∗sqrt(2)*) − *f(1)*/(*2∗sqrt(2)*),
f(0)/(*2∗sqrt(2)*) − (*1−f(0)*)/(*2∗sqrt(2)*) + (*f(1)*/(*2∗sqrt(2)*) − (*1−f(1)*)/(*2∗sqrt(2)*)),
(*1−f(0)*)/(*2∗sqrt(2)*) − *f(0)*/(*2∗sqrt(2)*) − (*1−f(1)*)/(*2∗sqrt(2)*) + *f(1)*/(*2∗sqrt(2)*),
f(0)/(*2∗sqrt(2)*) − (*1−f(0)*)/(*2∗sqrt(2)*) − *f(1)*/(*2∗sqrt(2)*) + (*1−f(1)*)/(*2∗sqrt(2)*)]]

lemma (**in** *deutsch*) ψ_2-*to*-ψ_3:
shows (*H* ⨂ *Id 1*) ∗ ψ_2 = ψ_3
⟨*proof*⟩

lemma (**in** *deutsch*) ψ_3-*is-state*:
shows *state 2* ψ_3
⟨*proof*⟩

Finally, all steps are put together. The result depends on the function f. If f is constant the first qubit of $\pm|f(0) \oplus f(1)\rangle \left[\frac{(|0\rangle - |1\rangle)}{\sqrt{2}}\right]$ is 0, if it is is_balanced it is 1. The algorithm only uses one evaluation of f(x) and will always succeed.

definition (**in** *deutsch*) *deutsch-algo*:: *complex Matrix.mat* **where**
deutsch-algo ≡ (*H* ⨂ *Id 1*) ∗ (*U*$_f$ ∗ ((*H* ∗ |*zero*⟩) ⨂ (*H* ∗ |*one*⟩)))

lemma (**in** *deutsch*) *deutsch-algo-result* [*simp*]:
shows *deutsch-algo* = ψ_3
⟨*proof*⟩

lemma (**in** *deutsch*) *deutsch-algo-result-is-state*:
shows *state 2 deutsch-algo*
⟨*proof*⟩

If the function is constant then the measurement of the first qubit should result in the state $|0\rangle$ with probability 1.

lemma (**in** *deutsch*) *prob0-deutsch-algo-const*:

assumes *is-const*
shows *prob0 2 deutsch-algo 0 = 1*
⟨*proof*⟩

lemma (**in** *deutsch*) *prob1-deutsch-algo-const*:
assumes *is-const*
shows *prob1 2 deutsch-algo 0 = 0*
⟨*proof*⟩

If the function is balanced the measurement of the first qubit should result in the state $|1\rangle$ with probability 1.

lemma (**in** *deutsch*) *prob0-deutsch-algo-balanced*:
assumes *is-balanced*
shows *prob0 2 deutsch-algo 0 = 0*
⟨*proof*⟩

lemma (**in** *deutsch*) *prob1-deutsch-algo-balanced*:
assumes *is-balanced*
shows *prob1 2 deutsch-algo 0 = 1*
⟨*proof*⟩

Eventually, the measurement of the first qubit results in $f(0) \oplus f(1)$.

definition (**in** *deutsch*) *deutsch-algo-eval*:: *real* **where**
deutsch-algo-eval ≡ *prob1 2 deutsch-algo 0*

lemma (**in** *deutsch*) *sum-mod-2-cases*:
shows (*f 0 + f 1*) *mod 2 = 0* ⟶ *is-const*
and (*f 0 + f 1*) *mod 2 = 1* ⟶ *is-balanced*
⟨*proof*⟩

lemma (**in** *deutsch*) *deutsch-algo-eval-is-sum-mod-2*:
shows *deutsch-algo-eval* = (*f 0 + f 1*) *mod 2*
⟨*proof*⟩

If the algorithm returns 0 then one concludes that the input function is constant and if it returns 1 then the function is balanced.

theorem (**in** *deutsch*) *deutsch-algo-is-correct*:
shows *deutsch-algo-eval = 0* ⟶ *is-const* **and** *deutsch-algo-eval = 1* ⟶ *is-balanced*
⟨*proof*⟩

end

11 The Deutsch-Jozsa Algorithm

theory *Deutsch-Jozsa*
imports
Deutsch
More-Tensor

Binary-Nat
begin

Given a function $f : 0,1^n \mapsto 0,1$, the Deutsch-Jozsa algorithm decides if this function is constant or balanced with a single $f(x)$ circuit to evaluate the function for multiple values of x simultaneously. The algorithm makes use of quantum parallelism and quantum interference.

A constant function with values in 0,1 returns either always 0 or always 1. A balanced function is 0 for half of the inputs and 1 for the other half.

locale *bob-fun* =
  fixes *f*:: *nat* ⇒ *nat* **and** *n*:: *nat*
  assumes *dom*: *f* ∈ ({(*i*::*nat*). *i* < 2^*n*} →$_E$ {*0*,*1*})
  assumes *dim*: *n* ≥ *1*

context *bob-fun*
begin

definition *const*:: *nat* ⇒ *bool* **where**
const c = (∀ *x*∈{*i*::*nat*. *i*<2^*n*}. *f x* = *c*)

definition *is-const*:: *bool* **where**
is-const ≡ *const 0* ∨ *const 1*

definition *is-balanced*:: *bool* **where**
is-balanced ≡ ∃ *A B* ::*nat set*. *A* ⊆ {*i*::*nat*. *i* < 2^*n*} ∧ *B* ⊆ {*i*::*nat*. *i* < 2^*n*}
  ∧ *card A* = 2^(*n*−*1*) ∧ *card B* = 2^(*n*−*1*)
  ∧ (∀ *x*∈*A*. *f x* = *0*) ∧ (∀ *x*∈*B*. *f x* = *1*)

lemma *is-balanced-inter*:
  fixes *A B*:: *nat set*
  assumes ∀ *x* ∈ *A*. *f x* = *0* **and** ∀ *x* ∈ *B*. *f x* = *1*
  shows *A* ∩ *B* = {}
  ⟨*proof*⟩

lemma *is-balanced-union*:
  fixes *A B*:: *nat set*
  assumes *A* ⊆ {*i*::*nat*. *i* < 2^*n*} **and** *B* ⊆ {*i*::*nat*. *i* < 2^*n*}
    and *card A* = 2^(*n*−*1*) **and** *card B* = 2^(*n*−*1*)
    and *A* ∩ *B* = {}
  shows *A* ∪ *B* = {*i*::*nat*. *i* < 2^*n*}
⟨*proof*⟩

lemma *f-ge-0*: ∀ *x*. *f x* ≥ *0* ⟨*proof*⟩

lemma *f-dom-not-zero*:
  shows *f* ∈ ({*i*::*nat*. *n* ≥ *1* ∧ *i* < 2^*n*} →$_E$ {*0*,*1*})
  ⟨*proof*⟩

lemma *f-values*: $\forall x \in \{(i::nat).\ i < 2\hat{}n\}\ .\ f\ x = 0 \vee f\ x = 1$
⟨*proof*⟩

end

The input function has to be constant or balanced.

locale *jozsa* = *bob-fun* +
assumes *const-or-balanced*: *is-const* ∨ *is-balanced*

Introduce two customised rules: disjunctions with four disjuncts and induction starting from one instead of zero.

lemma *disj-four-cases*:
assumes $A \vee B \vee C \vee D$ **and** $A \Longrightarrow P$ **and** $B \Longrightarrow P$ **and** $C \Longrightarrow P$ **and** $D \Longrightarrow P$
shows P
⟨*proof*⟩

The unitary transform U_f.

definition (**in** *jozsa*) *jozsa-transform*:: *complex Matrix.mat* (‹U_f›) **where**
$U_f \equiv$ *Matrix.mat* $(2\hat{}(n+1))$ $(2\hat{}(n+1))$ $(\lambda(i,j).$
if $i = j$ *then* $(1-f(i\ div\ 2))$ *else*
if $i = j + 1 \wedge$ *odd* i *then* $f(i\ div\ 2)$ *else*
if $i = j - 1 \wedge$ *even* $i \wedge j \geq 1$ *then* $f(i\ div\ 2)$ *else* $0)$

lemma (**in** *jozsa*) *jozsa-transform-dim* [*simp*]:
shows *dim-row* $U_f = 2\hat{}(n+1)$ **and** *dim-col* $U_f = 2\hat{}(n+1)$
⟨*proof*⟩

lemma (**in** *jozsa*) *jozsa-transform-coeff-is-zero* [*simp*]:
assumes $i <$ *dim-row* $U_f \wedge j <$ *dim-col* U_f
shows $(i \neq j \wedge \neg(i=j+1 \wedge$ *odd* $i) \wedge \neg\ (i=j-1 \wedge$ *even* $i \wedge j \geq 1)) \longrightarrow U_f$ \$\$ $(i,j) = 0$
⟨*proof*⟩

lemma (**in** *jozsa*) *jozsa-transform-coeff* [*simp*]:
assumes $i <$ *dim-row* $U_f \wedge j <$ *dim-col* U_f
shows $i = j \longrightarrow U_f$ \$\$ $(i,j) = 1 - f\ (i\ div\ 2)$
and $i = j + 1 \wedge$ *odd* $i \longrightarrow U_f$ \$\$ $(i,j) = f\ (i\ div\ 2)$
and $j \geq 1 \wedge i = j - 1 \wedge$ *even* $i \longrightarrow U_f$ \$\$ $(i,j) = f\ (i\ div\ 2)$
⟨*proof*⟩

lemma (**in** *jozsa*) U_f*-mult-without-empty-summands-sum-even*:
fixes $i\ j\ A$
assumes $i <$ *dim-row* U_f **and** $j <$ *dim-col* A **and** *even* i **and** *dim-col* $U_f =$ *dim-row* A
shows $(\sum k \in \{0..<$ *dim-row* $A\}.\ U_f$ \$\$ $(i,k) *$ A \$\$ $(k,j)) = (\sum k \in \{i,i+1\}.\ U_f$ \$\$ $(i,k) *$ A \$\$ $(k,j))$
⟨*proof*⟩

lemma (**in** *jozsa*) U_f*-mult-without-empty-summands-even*:
fixes *i j A*
assumes $i <$ *dim-row* U_f **and** $j <$ *dim-col A* **and** *even i* **and** *dim-col* $U_f =$ *dim-row A*
shows ($U_f * A$) $$ (*i*,*j*) = ($\sum k \in \{i, i+1\}$. U_f $$ (*i*,*k*) * *A* $$ (*k*,*j*))
⟨*proof*⟩

lemma (**in** *jozsa*) U_f*-mult-without-empty-summands-sum-odd*:
fixes *i j A*
assumes $i <$ *dim-row* U_f **and** $j <$ *dim-col A* **and** *odd i* **and** *dim-col* $U_f =$ *dim-row A*
shows ($\sum k \in \{0..<$ *dim-row A*$\}$. U_f $$ (*i*,*k*) * *A* $$ (*k*,*j*)) =($\sum k \in \{i-1, i\}$. U_f $$ (*i*,*k*) * *A* $$ (*k*,*j*))
⟨*proof*⟩

lemma (**in** *jozsa*) U_f*-mult-without-empty-summands-odd*:
fixes *i j A*
assumes $i <$ *dim-row* U_f **and** $j <$ *dim-col A* **and** *odd i* **and** *dim-col* $U_f =$ *dim-row A*
shows ($U_f * A$) $$ (*i*,*j*) = ($\sum k \in \{i-1, i\}$. U_f $$ (*i*,*k*) * *A* $$ (*k*,*j*))
⟨*proof*⟩

U_f is a gate.

lemma (**in** *jozsa*) *transpose-of-jozsa-transform*:
shows $(U_f)^t = U_f$
⟨*proof*⟩

lemma (**in** *jozsa*) *adjoint-of-jozsa-transform*:
shows $(U_f)^\dagger = U_f$
⟨*proof*⟩

lemma (**in** *jozsa*) *jozsa-transform-is-unitary-index-even*:
fixes *i j*:: *nat*
assumes $i <$ *dim-row* U_f **and** $j <$ *dim-col* U_f **and** *even i*
shows ($U_f * U_f$) $$ (*i*,*j*) = 1_m (*dim-col* U_f) $$ (*i*,*j*)
⟨*proof*⟩

lemma (**in** *jozsa*) *jozsa-transform-is-unitary-index-odd*:
fixes *i j*:: *nat*
assumes $i <$ *dim-row* U_f **and** $j <$ *dim-col* U_f **and** *odd i*
shows ($U_f * U_f$) $$ (*i*,*j*) = 1_m (*dim-col* U_f) $$ (*i*,*j*)
⟨*proof*⟩

lemma (**in** *jozsa*) *jozsa-transform-is-gate*:
shows *gate* (*n*+1) U_f
⟨*proof*⟩

N-fold application of the tensor product

fun *iter-tensor*:: *complex Matrix.mat* ⇒ *nat* ⇒ *complex Matrix.mat* (‹- ⊗^ ›) 75)

where
  $A \otimes^{(Suc\ 0)} = A$
| $A \otimes^{(Suc\ k)} = A \bigotimes (A \otimes^k)$

lemma *one-tensor-is-id* [*simp*]:
  fixes A
  shows $A \otimes^1 = A$
  ⟨*proof*⟩

lemma *iter-tensor-suc*:
  fixes n
  assumes $n \geq 1$
  shows $A \otimes^{(Suc\ n)} = A \bigotimes (A \otimes^n)$
  ⟨*proof*⟩

lemma *dim-row-of-iter-tensor* [*simp*]:
  fixes A n
  assumes $n \geq 1$
  shows *dim-row*$(A \otimes^n) =$ (*dim-row* $A)\hat{}n$
  ⟨*proof*⟩

lemma *dim-col-of-iter-tensor* [*simp*]:
  fixes A n
  assumes $n \geq 1$
  shows *dim-col*$(A \otimes^n) =$ (*dim-col* $A)\hat{}n$
  ⟨*proof*⟩

lemma *iter-tensor-values*:
  fixes A n i j
  assumes $n \geq 1$ **and** $i <$ *dim-row* $(A \bigotimes (A \otimes^n))$ **and** $j <$ *dim-col* $(A \bigotimes (A \otimes^n))$
  shows $(A \otimes^{(Suc\ n)})$ \$\$ $(i,j) = (A \bigotimes (A \otimes^n))$ \$\$ (i,j)
  ⟨*proof*⟩

lemma *iter-tensor-mult-distr*:
  assumes $n \geq 1$ **and** *dim-col* $A =$ *dim-row* B **and** *dim-col* $A > 0$ **and** *dim-col* $B > 0$
  shows $(A \otimes^{(Suc\ n)}) * (B \otimes^{(Suc\ n)}) = (A * B) \bigotimes ((A \otimes^n) * (B \otimes^n))$
⟨*proof*⟩

lemma *index-tensor-mat-with-vec2-row-cond*:
  fixes A B:: *complex Matrix.mat* **and** i:: *nat*
  assumes $i < 2 *$ (*dim-row* B) **and** $i \geq$ *dim-row* B **and** *dim-col* $B > 0$ **and** *dim-row* $A = 2$ **and** *dim-col* $A = 1$
  shows $(A \bigotimes B)$ \$\$ $(i,0) = (A$ \$\$ $(1,0)) * (B$ \$\$ $(i-$*dim-row* $B,0))$
⟨*proof*⟩

lemma *iter-tensor-of-gate-is-gate*:
  fixes A:: *complex Matrix.mat* **and** n m:: *nat*

assumes *gate m A* **and** $n \geq 1$
shows *gate* $(m*n)$ $(A \otimes^n)$
⟨*proof*⟩

lemma *iter-tensor-of-state-is-state*:
fixes *A*:: *complex Matrix.mat* **and** *n m*:: *nat*
assumes *state m A* **and** $n \geq 1$
shows *state* $(m*n)$ $(A \otimes^n)$
⟨*proof*⟩

We prepare n+1 qubits. The first n qubits in the state $|0\rangle$, the last one in the state $|1\rangle$.

abbreviation ψ_{10}:: *nat* ⇒ *complex Matrix.mat* **where**
ψ_{10} *n* ≡ *Matrix.mat* (*2^n*) *1* (λ(*i*,*j*). *1*/(*sqrt 2*)^*n*)

lemma ψ_{10}-*values*:
fixes *i j n*
assumes *i* < *dim-row* (ψ_{10} *n*) **and** *j* < *dim-col* (ψ_{10} *n*)
shows (ψ_{10} *n*) $$ (*i*,*j*) = *1*/(*sqrt 2*)^*n*
⟨*proof*⟩

$H^{\otimes n}$ is applied to $|0\rangle^{\otimes n}$.

lemma *H-on-ket-zero*:
shows (*H* ∗ |*zero*⟩) = ψ_{10} *1*
⟨*proof*⟩

lemma ψ_{10}-*tensor*:
assumes $n \geq 1$
shows (ψ_{10} *1*) ⨂ (ψ_{10} *n*) = (ψ_{10} (*Suc n*))
⟨*proof*⟩

lemma ψ_{10}-*tensor-is-state*:
assumes $n \geq 1$
shows *state n* (|*zero*⟩ $\otimes^n$)
⟨*proof*⟩

lemma *iter-tensor-of-H-is-gate*:
assumes $n \geq 1$
shows *gate n* ($H \otimes^n$)
⟨*proof*⟩

lemma *iter-tensor-of-H-on-zero-tensor*:
assumes $n \geq 1$
shows ($H \otimes^n$) ∗ (|*zero*⟩ $\otimes^n$) = ψ_{10} *n*
⟨*proof*⟩

lemma ψ_{10}-*is-state*:
assumes $n \geq 1$
shows *state n* (ψ_{10} *n*)

⟨*proof*⟩

abbreviation ψ_{11}:: *complex Matrix.mat* **where**
$\psi_{11} \equiv$ *Matrix.mat 2 1* (λ(*i*,*j*). *if i=0 then 1/sqrt(2) else −1/sqrt(2)*)

lemma *H-on-ket-one-is-*ψ_{11}:
shows $(H * |one\rangle) = \psi_{11}$
⟨*proof*⟩

abbreviation ψ_1:: *nat* ⇒ *complex Matrix.mat* **where**
$\psi_1\ n \equiv$ *Matrix.mat* (*2^(n+1)*) *1* (λ(*i*,*j*). *if even i then 1/(sqrt 2)^(n+1) else −1/(sqrt 2)^(n+1)*)

lemma ψ_1*-values-even*[*simp*]:
fixes *i j n*
assumes $i <$ *dim-row* $(\psi_1\ n)$ **and** $j <$ *dim-col* $(\psi_1\ n)$ **and** *even i*
shows $(\psi_1\ n)$ $$ (*i*,*j*) = *1/(sqrt 2)^(n+1)*
⟨*proof*⟩

lemma ψ_1*-values-odd* [*simp*]:
fixes *i j n*
assumes $i <$ *dim-row* $(\psi_1\ n)$ **and** $j <$ *dim-col* $(\psi_1\ n)$ **and** *odd i*
shows $(\psi_1\ n)$ $$ (*i*,*j*) = *−1/(sqrt 2)^(n+1)*
⟨*proof*⟩

lemma ψ_{10}*-tensor-*ψ_{11}*-is-*ψ_1:
assumes $n \geq 1$
shows $(\psi_{10}\ n) \bigotimes \psi_{11} = \psi_1\ n$
⟨*proof*⟩

lemma ψ_1*-is-state*:
assumes $n \geq 1$
shows *state* *(n+1)* $(\psi_1\ n)$
⟨*proof*⟩

abbreviation (**in** *jozsa*) ψ_2:: *complex Matrix.mat* **where**
$\psi_2 \equiv$ *Matrix.mat* *(2^(n+1))* *1* (λ(*i*,*j*). *if even i then (−1)^f(i div 2)/(sqrt 2)^(n+1)*

else (−1)^(f(i div 2)+1)/(sqrt 2)^(n+1))

lemma (**in** *jozsa*) ψ_2*-values-even* [*simp*]:
fixes *i j*
assumes $i <$ *dim-row* ψ_2 **and** $j <$ *dim-col* ψ_2 **and** *even i*
shows ψ_2 $$ (*i*,*j*) = *(−1)^f(i div 2)/(sqrt 2)^(n+1)*
⟨*proof*⟩

lemma (**in** *jozsa*) ψ_2*-values-odd* [*simp*]:
fixes *i j*
assumes $i <$ *dim-row* ψ_2 **and** $j <$ *dim-col* ψ_2 **and** *odd i*

shows ψ_2 $$ (*i*,*j*) = (−1)^(*f*(*i div 2*)+1)/(*sqrt 2*)^(*n*+1)
⟨*proof*⟩

lemma (**in** *jozsa*) ψ_2-*values-odd-hidden* [*simp*]:
assumes *2*∗*k*+1 < *dim-row* ψ_2 **and** *j* < *dim-col* ψ_2
shows ψ_2 $$ (*2*∗*k*+1,*j*) = ((−1)^(*f*((*2*∗*k*+1) *div 2*)+1))/(*sqrt 2*)^(*n*+1)
⟨*proof*⟩

lemma (**in** *jozsa*) *snd-rep-of-*ψ_2:
assumes *i* < *dim-row* ψ_2
shows ((*1*−*f*(*i div 2*)) + −*f*(*i div 2*)) ∗ *1*/(*sqrt 2*)^(*n*+1) = (−1)^*f*(*i div 2*)/(*sqrt 2*)^(*n*+1)
and (−(*1*−*f*(*i div 2*))+(*f*(*i div 2*)))∗ *1*/(*sqrt 2*)^(*n*+1) = (−1)^(*f*(*i div 2*)+1)/(*sqrt 2*)^(*n*+1)
⟨*proof*⟩

lemma (**in** *jozsa*) *jozsa-transform-times-*ψ_1*-is-*ψ_2:
shows U_f ∗ (ψ_1 *n*) = ψ_2
⟨*proof*⟩

lemma (**in** *jozsa*) ψ_2-*is-state*:
shows *state* (*n*+1) ψ_2
⟨*proof*⟩

$H^{\hat{}}_{\otimes}$ *n* is the result of taking the nth tensor product of H

abbreviation *iter-tensor-of-H-rep*:: *nat* ⇒ *complex Matrix.mat* (‹$H^{\hat{}}_{\otimes}$ -›) **where** *iter-tensor-of-H-rep n* ≡ *Matrix.mat* (*2*^*n*) (*2*^*n*) (λ(*i*,*j*).(−1)^(*i* ·*n j*)/(*sqrt 2*)^*n*)

lemma *tensor-of-H-values* [*simp*]:
fixes *n i j*:: *nat*
assumes *i* < *dim-row* ($H^{\hat{}}_{\otimes}$ *n*) **and** *j* < *dim-col* ($H^{\hat{}}_{\otimes}$ *n*)
shows ($H^{\hat{}}_{\otimes}$ *n*) $$ (*i*,*j*) = (−1)^(*i* ·*n j*)/(*sqrt 2*)^*n*
⟨*proof*⟩

lemma *dim-row-of-iter-tensor-of-H* [*simp*]:
assumes *n* ≥ *1*
shows *1* < *dim-row* ($H^{\hat{}}_{\otimes}$ *n*)
⟨*proof*⟩

lemma *iter-tensor-of-H-fst-pos*:
fixes *n i j*:: *nat*
assumes *i* < *2*^*n* ∨ *j* < *2*^*n* **and** *i* < *2*^(*n*+1) ∧ *j* < *2*^(*n*+1)
shows ($H^{\hat{}}_{\otimes}$ (*Suc n*)) $$ (*i*,*j*) = *1*/*sqrt*(*2*) ∗ (($H^{\hat{}}_{\otimes}$ *n*) $$ (*i mod 2*^*n*, *j mod 2*^*n*))
⟨*proof*⟩

lemma *iter-tensor-of-H-fst-neg*:
fixes *n i j*:: *nat*
assumes *i* ≥ *2*^*n* ∧ *j* ≥ *2*^*n* **and** *i* < *2*^(*n*+1) ∧ *j* < *2*^(*n*+1)

shows $(H^{\hat{}}_{\otimes}\ (Suc\ n))\ \$\$\ (i,j) = -1/sqrt(2) * (H^{\hat{}}_{\otimes}\ n)\ \$\$\ (i\ mod\ 2^{\hat{}}n,\ j\ mod\ 2^{\hat{}}n)$
⟨*proof*⟩

lemma *H-tensor-iter-tensor-of-H*:
fixes *n*:: *nat*
shows $(H \bigotimes H^{\hat{}}_{\otimes}\ n) = H^{\hat{}}_{\otimes}\ (Suc\ n)$
⟨*proof*⟩

We prove that $Matrix.mat\ 2^n\ 2^n\ (\lambda x.\ complex\text{-}of\text{-}real\ (case\ x\ of\ (i,\ j) \Rightarrow (-1)^{i\ \cdot n\ j}\ /\ (sqrt\ 2)^n))$ is indeed the matrix representation of $H\ \otimes^n$, the iterated tensor product of the Hadamard gate H.

lemma *one-tensor-of-H-is-H*:
shows $(H^{\hat{}}_{\otimes}\ 1) = H$
⟨*proof*⟩

lemma *iter-tensor-of-H-rep-is-correct*:
fixes *n*:: *nat*
assumes $n \geq 1$
shows $(H\ \otimes^n) = H^{\hat{}}_{\otimes}\ n$
⟨*proof*⟩

$HId^{\hat{}}_{\otimes}\ 1$ is the result of taking the tensor product of the nth tensor of H and Id 1

abbreviation *tensor-of-H-tensor-Id*:: *nat* ⇒ *complex Matrix.mat* (‹*HId*$^{\hat{}}_{\otimes}$ -›) **where** *tensor-of-H-tensor-Id n* ≡ *Matrix.mat* $(2\hat{}(n+1))\ (2\hat{}(n+1))\ (\lambda(i,j).$
if $(i\ mod\ 2 = j\ mod\ 2)\ then\ (-1)\hat{}((i\ div\ 2) \cdot_n (j\ div\ 2))/(sqrt\ 2)\hat{}n\ else\ 0)$

lemma *mod-2-is-both-even-or-odd*:
$((even\ i \wedge even\ j) \vee (odd\ i \wedge odd\ j)) \longleftrightarrow (i\ mod\ 2 = j\ mod\ 2)$
⟨*proof*⟩

lemma *HId-values* [*simp*]:
assumes $n \geq 1$ **and** $i < dim\text{-}row\ (HId^{\hat{}}_{\otimes}\ n)$ **and** $j < dim\text{-}col\ (HId^{\hat{}}_{\otimes}\ n)$
shows $even\ i \wedge even\ j \longrightarrow (HId^{\hat{}}_{\otimes}\ n)\ \$\$\ (i,j) = (-1)\hat{}((i\ div\ 2) \cdot_n (j\ div\ 2))/(sqrt\ 2)\hat{}n$
and $odd\ i \wedge odd\ j \longrightarrow (HId^{\hat{}}_{\otimes}\ n)\ \$\$\ (i,j) = (-1)\hat{}((i\ div\ 2) \cdot_n (j\ div\ 2))/(sqrt\ 2)\hat{}n$
and $(i\ mod\ 2 = j\ mod\ 2) \longrightarrow (HId^{\hat{}}_{\otimes}\ n)\ \$\$\ (i,j) = (-1)\hat{}((i\ div\ 2) \cdot_n (j\ div\ 2))/(sqrt\ 2)\hat{}n$
and $\neg(i\ mod\ 2 = j\ mod\ 2) \longrightarrow (HId^{\hat{}}_{\otimes}\ n)\ \$\$\ (i,j) = 0$
⟨*proof*⟩

lemma *iter-tensor-of-H-tensor-Id-is-HId*:
shows $(H^{\hat{}}_{\otimes}\ n) \bigotimes Id\ 1 = HId^{\hat{}}_{\otimes}\ n$
⟨*proof*⟩

lemma *HId-is-gate*:
assumes $n \geq 1$

shows *gate* (*n*+1) (*HId*^⊗ *n*)
⟨*proof*⟩

State ψ_3 is obtained by the multiplication of *Matrix.mat* 2^{n+1} 2^{n+1} (λx. *complex-of-real* (*case x of* (*i*, *j*) ⇒ *if i mod 2* = *j mod 2 then* $(-1)^{i \text{ div } 2 \cdot_n j \text{ div } 2}$ / $(sqrt\ 2)^n$ *else 0*)) and ψ_2

abbreviation (**in** *jozsa*) ψ_3:: *complex Matrix.mat* **where**
ψ_3 ≡ *Matrix.mat* (2^(*n*+1)) *1* (λ(*i*,*j*).
if even i
then (∑ *k*<2^*n*. (−1)^(*f*(*k*) + ((*i div 2*) ·*n k*))/((*sqrt 2*)^*n* ∗ (*sqrt 2*)^(*n*+1)))
else (∑ *k*<2^*n*. (−1)^(*f*(*k*)+ *1* + ((*i div 2*) ·*n k*)) /((*sqrt 2*)^*n* ∗ (*sqrt 2*)^(*n*+1))))

lemma (**in** *jozsa*) ψ_3-*values*:
assumes *i* < *dim-row* ψ_3
shows *odd i* ⟶ ψ_3 $$ (*i*,*0*) = (∑ *k*<2^*n*. (−1)^(*f*(*k*) + *1* + ((*i div 2*) ·*n k*))/((*sqrt 2*)^*n* ∗ (*sqrt 2*)^(*n*+1)))
⟨*proof*⟩

lemma (**in** *jozsa*) ψ_3-*dim* [*simp*]:
shows *1* < *dim-row* ψ_3
⟨*proof*⟩

lemma *sum-every-odd-summand-is-zero*:
fixes *n*:: *nat*
assumes *n* ≥ *1*
shows ∀ *f*::(*nat* ⇒ *complex*).(∀ *i*. *i*<2^(*n*+1) ∧ *odd i* ⟶ *f i* = *0*) ⟶
(∑ *k*∈{*0*..<2^(*n*+1)}. *f k*) = (∑ *k*∈{*0*..<2^*n*}. *f* (*2*∗*k*))
⟨*proof*⟩

lemma *sum-every-even-summand-is-zero*:
fixes *n*:: *nat*
assumes *n* ≥ *1*
shows ∀ *f*::(*nat* ⇒ *complex*).(∀ *i*. *i*<2^(*n*+1) ∧ *even i* ⟶ *f i* = *0*) ⟶
(∑ *k*∈{*0*..<2^(*n*+1)}. *f k*) = (∑ *k*∈{*0*..< 2^*n*}. *f* (*2*∗*k*+*1*))
⟨*proof*⟩

lemma (**in** *jozsa*) *iter-tensor-of-H-times-*ψ_2*-is-*ψ_3:
shows ((*H*^⊗ *n*) ⨂ *Id 1*) ∗ ψ_2 = ψ_3
⟨*proof*⟩

lemma (**in** *jozsa*) ψ_3-*is-state*:
shows *state* (*n*+*1*) ψ_3
⟨*proof*⟩

Finally, all steps are put together. The result depends on the function f. If f is constant the first n qubits are 0, if f is balanced there is at least one qubit in state 1 among the first n qubits. The algorithm only uses one evaluation of f(x) and will always succeed.

definition (**in** *jozsa*) *jozsa-algo*:: *complex Matrix.mat* **where**
jozsa-algo ≡ ((*H* ⊗^ *n*) ⨂ *Id 1*) ∗ (*U*$_f$ ∗ (((*H* ⊗^ *n*) ∗ (|*zero*⟩ ⊗^ *n*)) ⨂ (*H* ∗ |*one*⟩))))

lemma (**in** *jozsa*) *jozsa-algo-result* [*simp*]:
shows *jozsa-algo* = ψ_3
⟨*proof*⟩

lemma (**in** *jozsa*) *jozsa-algo-result-is-state*:
shows *state* (*n*+*1*) *jozsa-algo*
⟨*proof*⟩

lemma (**in** *jozsa*) *prob0-fst-qubits-of-jozsa-algo*:
shows (*prob0-fst-qubits n jozsa-algo*) = (∑ *j*∈{*0*,*1*}. (*cmod*(*jozsa-algo* $$ (*j*,*0*)))^2)
⟨*proof*⟩

General lemmata required to compute probabilities.

lemma *aux-comp-with-sqrt2*:
shows (*sqrt 2*)^ *n* ∗ (*sqrt 2*)^ *n* = *2*^ *n*
⟨*proof*⟩

lemma *aux-comp-with-sqrt2-bis* [*simp*]:
shows *2*^ *n*/(*sqrt*(*2*)^ *n* ∗ *sqrt*(*2*)^(*n*+*1*)) = *1*/*sqrt 2*
⟨*proof*⟩

lemma *aux-ineq-with-card*:
fixes *g*:: *nat* ⇒ *nat* **and** *A*:: *nat set*
assumes *finite A*
shows (∑ *k*∈*A*. (−*1*)^(*g k*)) ≤ *card A* **and** (∑ *k*∈*A*. (−*1*)^(*g k*)) ≥ −*card A*
⟨*proof*⟩

lemma *aux-comp-with-cmod*:
fixes *g*:: *nat* ⇒ *nat*
assumes (∀ *x*<*2*^ *n*. *g x* = *0*) ∨ (∀ *x*<*2*^ *n*. *g x* = *1*)
shows (*cmod* (∑ *k*<*2*^ *n*. (−*1*)^(*g k*)))^2 = *2*^(*2*∗*n*)
⟨*proof*⟩

lemma *cmod-less*:
fixes *a n*:: *int*
assumes *a* < *n* **and** *a* > −*n*
shows *cmod a* < *n*
⟨*proof*⟩

lemma *square-less*:
fixes *a n*:: *real*
assumes *a* < *n* **and** *a* > −*n*
shows $a^2 < n^2$
⟨*proof*⟩

lemma *cmod-square-real* [*simp*]:

fixes n:: *real*
shows $(cmod\ n)^2 = n^2$
⟨*proof*⟩

lemma *aux-comp-sum-divide-cmod*:
fixes n:: *nat* **and** g:: $nat \Rightarrow int$ **and** a:: *real*
shows $(cmod(complex\text{-}of\text{-}real(\sum k{<}n.\ g\ k\ /\ a)))^2 = (cmod\ (\sum k{<}n.\ g\ k)\ /\ a)^2$
⟨*proof*⟩

The function is constant if and only if the first n qubits are 0. So, if the function is constant, then the probability of measuring 0 for the first n qubits is 1.

lemma (**in** *jozsa*) *prob0-jozsa-algo-of-const-0*:
assumes *const 0*
shows *prob0-fst-qubits n jozsa-algo = 1*
⟨*proof*⟩

lemma (**in** *jozsa*) *prob0-jozsa-algo-of-const-1*:
assumes *const 1*
shows *prob0-fst-qubits n jozsa-algo = 1*
⟨*proof*⟩

If the probability of measuring 0 for the first n qubits is 1, then the function is constant.

lemma (**in** *jozsa*) *max-value-of-not-const-less*:
assumes ¬ *const 0* **and** ¬ *const 1*
shows $(cmod\ (\sum k{::}nat{<}2\hat{}n.\ (-(1{::}nat))\hat{}(f\ k)))^2 < (2{::}nat)\hat{}(2{*}n)$
⟨*proof*⟩

lemma (**in** *jozsa*) *max-value-of-not-const-less-bis*:
assumes ¬ *const 0* **and** ¬ *const 1*
shows $(cmod\ (\sum k{::}nat{<}2\hat{}n.\ (-(1{::}nat))\hat{}(f\ k\ +\ 1)))^2 < (2{::}nat)\hat{}(2{*}n)$
⟨*proof*⟩

lemma (**in** *jozsa*) *f-const-has-max-value*:
assumes *const 0* ∨ *const 1*
shows $(cmod\ (\sum k{<}(2{::}nat)\hat{}n.\ (-1)\hat{}(f\ k)))^2 = (2{::}nat)\hat{}(2{*}n)$
and $(cmod\ (\sum k{<}(2{::}nat)\hat{}n.\ (-1)\hat{}(f\ k\ +\ 1)))^2 = (2{::}nat)\hat{}(2{*}n)$
⟨*proof*⟩

lemma (**in** *jozsa*) *prob0-fst-qubits-leq*:
shows $(cmod\ (\sum k{<}(2{::}nat)\hat{}n.\ (-1)\hat{}(f\ k)))^2 \leq (2{::}nat)\hat{}(2{*}n)$
and $(cmod\ (\sum k{<}(2{::}nat)\hat{}n.\ (-1)\hat{}(f\ k\ +\ 1)))^2 \leq (2{::}nat)\hat{}(2{*}n)$
⟨*proof*⟩

lemma (**in** *jozsa*) *prob0-jozsa-algo-1-is-const*:
assumes *prob0-fst-qubits n jozsa-algo = 1*
shows *const 0* ∨ *const 1*
⟨*proof*⟩

The function is balanced if and only if at least one qubit among the first n qubits is not zero. So, if the function is balanced then the probability of measuring 0 for the first n qubits is 0.

lemma *sum-union-disjoint-finite-set*:
fixes *C*::*nat set* **and** *g*::*nat* ⇒ *int*
assumes *finite C*
shows ∀ *A B*. *A* ∩ *B* = {} ∧ *A* ∪ *B* = *C* ⟶ (∑ *k*∈*C*. *g k*) = (∑ *k*∈*A*. *g k*) + (∑ *k*∈*B*. *g k*)
⟨*proof*⟩

lemma (**in** *jozsa*) *balanced-pos-and-neg-terms-cancel-out1*:
assumes *is-balanced*
shows (∑ *k*<(*2*::*nat*)^*n*. (−(*1*::*nat*))^(*f k*)) = *0*
⟨*proof*⟩

lemma (**in** *jozsa*) *balanced-pos-and-neg-terms-cancel-out2*:
assumes *is-balanced*
shows (∑ *k*<(*2*::*nat*)^*n*. (−(*1*::*nat*))^(*f k* + *1*)) = *0*
⟨*proof*⟩

lemma (**in** *jozsa*) *prob0-jozsa-algo-of-balanced*:
assumes *is-balanced*
shows *prob0-fst-qubits n jozsa-algo* = *0*
⟨*proof*⟩

If the probability that the first n qubits are 0 is 0, then the function is balanced.

lemma (**in** *jozsa*) *balanced-prob0-jozsa-algo*:
assumes *prob0-fst-qubits n jozsa-algo* = *0*
shows *is-balanced*
⟨*proof*⟩

We prove the correctness of the algorithm.

definition (**in** *jozsa*) *jozsa-algo-eval*:: *real* **where**
jozsa-algo-eval ≡ *prob0-fst-qubits n jozsa-algo*

theorem (**in** *jozsa*) *jozsa-algo-is-correct*:
shows *jozsa-algo-eval* = *1* ⟷ *is-const*
and *jozsa-algo-eval* = *0* ⟷ *is-balanced*
⟨*proof*⟩

end

12 The No-Cloning Theorem

theory *No-Cloning*
imports
Quantum

Tensor
begin

12.1 The Cauchy-Schwarz Inequality

lemma *inner-prod-expand*:
assumes *dim-vec a = dim-vec b* **and** *dim-vec a = dim-vec c* **and** *dim-vec a = dim-vec d*
shows $\langle a + b|c + d\rangle = \langle a|c\rangle + \langle a|d\rangle + \langle b|c\rangle + \langle b|d\rangle$
⟨*proof*⟩

lemma *inner-prod-distrib-left*:
assumes *dim-vec a = dim-vec b*
shows $\langle c \cdot_v a|b\rangle = cnj(c) * \langle a|b\rangle$
⟨*proof*⟩

lemma *inner-prod-distrib-right*:
assumes *dim-vec a = dim-vec b*
shows $\langle a|c \cdot_v b\rangle = c * \langle a|b\rangle$
⟨*proof*⟩

lemma *cauchy-schwarz-ineq*:
assumes *dim-vec v = dim-vec w*
shows $(cmod(\langle v|w\rangle))^2 \leq Re\ (\langle v|v\rangle * \langle w|w\rangle)$
⟨*proof*⟩

lemma *cauchy-schwarz-eq* [*simp*]:
assumes $v = (l \cdot_v w)$
shows $(cmod(\langle v|w\rangle))^2 = Re\ (\langle v|v\rangle * \langle w|w\rangle)$
⟨*proof*⟩

lemma *cauchy-schwarz-col* [*simp*]:
assumes *dim-vec v = dim-vec w* **and** $(cmod(\langle v|w\rangle))^2 = Re\ (\langle v|v\rangle * \langle w|w\rangle)$
shows $\exists l.\ v = (l \cdot_v w) \vee w = (l \cdot_v v)$
⟨*proof*⟩

12.2 The No-Cloning Theorem

lemma *eq-from-inner-prod* [*simp*]:
assumes *dim-vec v = dim-vec w* **and** $\langle v|w\rangle = 1$ **and** $\langle v|v\rangle = 1$ **and** $\langle w|w\rangle = 1$
shows $v = w$
⟨*proof*⟩

lemma *hermite-cnj-of-tensor*:
shows $(A \bigotimes B)^\dagger = (A^\dagger) \bigotimes (B^\dagger)$
⟨*proof*⟩

locale *quantum-machine* =
fixes *n*:: *nat* **and** *s*:: *complex Matrix.vec* **and** *U*:: *complex Matrix.mat*
assumes *dim-vec* [*simp*]: *dim-vec s = 2^n*

and *dim-col* [*simp*]: *dim-col U = 2^n * 2^n*
and *square* [*simp*]: *square-mat U* **and** *unitary* [*simp*]: *unitary U*

lemma *inner-prod-of-unit-vec*:
fixes *n i*:: *nat*
assumes *i < n*
shows ⟨*unit-vec n i*| *unit-vec n i*⟩ = *1*
⟨*proof*⟩

theorem (**in** *quantum-machine*) *no-cloning*:
assumes [*simp*]: *dim-vec v = 2^n* **and** [*simp*]: *dim-vec w = 2^n* **and**
cloning1: ⋀*s*. *U* * (|*v*⟩ ⊗ |*s*⟩) = |*v*⟩ ⊗ |*v*⟩ **and**
cloning2: ⋀*s*. *U* * (|*w*⟩ ⊗ |*s*⟩) = |*w*⟩ ⊗ |*w*⟩ **and**
⟨*v*|*v*⟩ = *1* **and** ⟨*w*|*w*⟩ = *1*
shows *v = w* ∨ ⟨*v*|*w*⟩ = *0*
⟨*proof*⟩

end

13 Quantum Prisoner's Dilemma

theory *Quantum-Prisoners-Dilemma*
imports
More-Tensor
Measurement
Basics
begin

In the 2-parameter strategic space of Eisert, Wilkens and Lewenstein [EWL], Prisoner's Dilemma ceases to pose a dilemma if quantum strategies are allowed for. Indeed, Alice and Bob both choosing to defect is no longer a Nash equilibrium. However, a new Nash equilibrium appears which is at the same time Pareto optimal. Moreover, there exists a quantum strategy which always gives reward if played against any classical strategy. Below the parameter γ can be seen as a measure of the game's entanglement. The game behaves classically if $\gamma = 0$, and for the maximally entangled case ($\gamma = 2*\pi$) the dilemma disappears as pointed out above.

13.1 The Set-Up

locale *prisoner* =
fixes γ:: *real*
assumes $\gamma \leq pi/2$ **and** $\gamma \geq 0$

abbreviation (**in** *prisoner*) *J* :: *complex Matrix.mat* **where**
J ≡ *mat-of-cols-list 4* [[*cos*(γ/*2*), *0*, *0*, i∗*sin*(γ/*2*)],
[*0*, *cos*(γ/*2*), −i∗*sin*(γ/*2*), *0*],
[*0*, −i∗*sin*(γ/*2*), *cos*(γ/*2*), *0*],

$[\mathrm{i}*sin(\gamma/2),\ 0,\ 0,\ cos(\gamma/2)]]$

abbreviation (**in** *prisoner*) ψ_1 :: *complex Matrix.mat* **where**
$\psi_1 \equiv$ *mat-of-cols-list 4* $[[cos(\gamma/2),\ 0,\ 0,\ \mathrm{i}*sin(\gamma/2)]]$

lemma (**in** *prisoner*) *psi-one*:
shows $J * |unit\text{-}vec\ 4\ 0\rangle = \psi_1$
⟨*proof*⟩

locale *strategic-space-2p* = *prisoner* +
fixes ϑ_A:: *real*
and φ_A:: *real*
and ϑ_B:: *real*
and φ_B:: *real*
assumes $0 \le \vartheta_A \wedge \vartheta_A \le pi$
and $0 \le \varphi_A \wedge \varphi_A \le pi/2$
and $0 \le \vartheta_B \wedge \vartheta_B \le pi$
and $0 \le \varphi_B \wedge \varphi_B \le pi/2$

abbreviation (**in** *strategic-space-2p*) U_A :: *complex Matrix.mat* **where**
$U_A \equiv$ *mat-of-cols-list 2* $[[exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2),\ -sin(\vartheta_A/2)],$
$[sin(\vartheta_A/2),\ exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2)]]$

abbreviation (**in** *strategic-space-2p*) U_B :: *complex Matrix.mat* **where**
$U_B \equiv$ *mat-of-cols-list 2* $[[exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2),\ -sin(\vartheta_B/2)],$
$[sin(\vartheta_B/2),\ exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2)]]$

abbreviation (**in** *strategic-space-2p*) ψ_2 :: *complex Matrix.mat* **where**
$\psi_2 \equiv$
mat-of-cols-list 4 $[[exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * cos(\gamma/2) + sin(\vartheta_A/2)$
$* sin(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2),$
$exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * -sin(\vartheta_B/2) * cos(\gamma/2) + sin(\vartheta_A/2) *$
$exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2),$
$-sin(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * cos(\gamma/2) + exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2)$
$* sin(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2),$
$sin(\vartheta_A/2) * sin(\vartheta_B/2) * cos(\gamma/2) + exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) *$
$exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2)]]$

abbreviation (**in** *strategic-space-2p*) U_{AB} :: *complex Matrix.mat* **where**
$U_{AB} \equiv$
mat-of-cols-list 4 $[[exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2),\ exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2)$
$* -sin(\vartheta_B/2),$
$-sin(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2),\ -sin(\vartheta_A/2) *$
$-sin(\vartheta_B/2)],$
$[exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * sin(\vartheta_B/2),\ exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) *$
$exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2),$
$-sin(\vartheta_A/2) * sin(\vartheta_B/2),\ -sin(\vartheta_A/2) * exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2)],$
$[sin(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2),\ -sin(\vartheta_A/2) * sin(\vartheta_B/2),$
$exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * exp\ (\mathrm{i}*\varphi_B)*cos(\vartheta_B/2),$

$exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * -sin(\vartheta_B/2)],$
$[sin(\vartheta_A/2) * sin(\vartheta_B/2),\ sin(\vartheta_A/2) * exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2),$
$exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * sin(\vartheta_B/2),\ exp\ (-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2)$
$* exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2)]]$

lemma (**in** *strategic-space-2p*) U_A-*tensor-*U_B:
shows $(U_A \bigotimes U_B) = U_{AB}$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *psi-two*:
shows $(U_A \bigotimes U_B) * \psi_1 = \psi_2$
⟨*proof*⟩

abbreviation (**in** *prisoner*) *J-cnj* :: *complex Matrix.mat* **where**
J-cnj ≡ *mat-of-cols-list 4* $[[cos(\gamma/2),\ 0,\ 0,\ -\mathrm{i}*sin(\gamma/2)],$
$[0,\ cos(\gamma/2),\ \mathrm{i}*sin(\gamma/2),\ 0],$
$[0,\ \mathrm{i}*sin(\gamma/2),\ cos(\gamma/2),\ 0],$
$[-\mathrm{i}*sin(\gamma/2),\ 0,\ 0,\ cos(\gamma/2)]]$

lemma (**in** *prisoner*) *hermite-cnj-of-J* [*simp*]:
shows $J^\dagger = J\text{-}cnj$
⟨*proof*⟩

abbreviation (**in** *strategic-space-2p*) ψ_f :: *complex Matrix.mat* **where**
ψ_f ≡ *mat-of-cols-list 4* [[
$cos(\gamma/2) * (exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * cos(\gamma/2) + sin(\vartheta_A/2)$
$* sin(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2))$
$+ (-\mathrm{i}*sin(\gamma/2)) * (sin(\vartheta_A/2) * sin(\vartheta_B/2) * cos(\gamma/2) + exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2)$
$* exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2)),$

$cos(\gamma/2) * (exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * -sin(\vartheta_B/2) * cos(\gamma/2) + sin(\vartheta_A/2) * exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2)$
$* \mathrm{i}*sin(\gamma/2))$
$+ (\mathrm{i}*sin(\gamma/2)) * (-sin(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * cos(\gamma/2) + exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2)$
$* sin(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2)),$

$(\mathrm{i}*sin(\gamma/2)) * (exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * -sin(\vartheta_B/2) * cos(\gamma/2) + sin(\vartheta_A/2) *$
$exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2))$
$+ cos(\gamma/2) * (-sin(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * cos(\gamma/2) + exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2)$
$* sin(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2)),$

$(-\mathrm{i}*sin(\gamma/2)) * (exp(\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) * exp(\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * cos(\gamma/2) + sin(\vartheta_A/2)$
$* sin(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2))$
$+ cos(\gamma/2) * (sin(\vartheta_A/2) * sin(\vartheta_B/2) * cos(\gamma/2) + exp(-\mathrm{i}*\varphi_A)*cos(\vartheta_A/2) *$
$exp(-\mathrm{i}*\varphi_B)*cos(\vartheta_B/2) * \mathrm{i}*sin(\gamma/2))$
]]

lemma (**in** *strategic-space-2p*) *psi-f*:
shows $(J^\dagger) * \psi_2 = \psi_f$
⟨*proof*⟩

lemma (**in** *prisoner*) *unit-vec-4-0-ket-is-state*:
shows *state 2* $|$*unit-vec 4 0*$\rangle$
⟨*proof*⟩

lemma *cos-sin-squared-add-cpx*:
complex-of-real (*cos* (*γ*/*2*)) ∗ *complex-of-real* (*cos* (*γ*/*2*)) −
i∗*complex-of-real* (*sin* (*γ*/*2*)) ∗ (i∗*complex-of-real* (*sin* (*γ*/*2*))) = *1*
⟨*proof*⟩

lemma *sin-cos-squared-add-cpx*:
i∗*complex-of-real* (*sin* (*γ*/*2*)) ∗ (i∗*complex-of-real* (*sin* (*γ*/*2*))) −
complex-of-real (*cos* (*γ*/*2*)) ∗ *complex-of-real* (*cos* (*γ*/*2*)) = −*1*
⟨*proof*⟩

lemma (**in** *prisoner*) *J-cnj-times-J*:
shows $J^\dagger * J = 1_m\ 4$
⟨*proof*⟩

lemma (**in** *prisoner*) *J-times-J-cnj*:
shows $J * (J^\dagger) = 1_m\ 4$
⟨*proof*⟩

lemma (**in** *prisoner*) *J-is-gate*:
shows *gate 2 J*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *psi-one-is-state*:
shows *state 2* ψ_1
⟨*proof*⟩

abbreviation (**in** *strategic-space-2p*) U_A*-cnj* :: *complex Matrix.mat* **where**
U_A*-cnj* ≡ *mat-of-cols-list 2* [[(*exp*(−i∗φ_A))∗*cos*(ϑ_A/*2*), *sin*(ϑ_A/*2*)],
[−*sin*(ϑ_A/*2*), (*exp* (i∗φ_A))∗*cos*(ϑ_A/*2*)]]

abbreviation (**in** *strategic-space-2p*) U_B*-cnj* :: *complex Matrix.mat* **where**
U_B*-cnj* ≡ *mat-of-cols-list 2* [[(*exp*(−i∗φ_B))∗*cos*(ϑ_B/*2*), *sin*(ϑ_B/*2*)],
[−*sin*(ϑ_B/*2*), (*exp*(i∗φ_B))∗*cos*(ϑ_B/*2*)]]

lemma (**in** *strategic-space-2p*) *hermite-cnj-of-*U_A:
shows $U_A{}^\dagger = U_A$*-cnj*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *hermite-cnj-of-*U_B:
shows $U_B{}^\dagger = U_B$*-cnj*
⟨*proof*⟩

lemma *exp-sin-cos-squared-add*:
fixes *x y* :: *real*

shows *exp* (− (i ∗ *x*)) ∗ *cos* (*y*) ∗ (*exp* (i ∗ *x*) ∗ *cos* (*y*)) + *sin*(*y*) ∗ *sin*(*y*) = *1*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) U_A-*cnj-times-*U_A:
shows $U_A{}^\dagger * U_A = 1_m\ 2$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) U_A-*times-*U_A-*cnj*:
shows $U_A * (U_A{}^\dagger) = 1_m\ 2$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) U_B-*cnj-times-*U_B:
shows $U_B{}^\dagger * U_B = 1_m\ 2$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) U_B-*times-*U_B-*cnj*:
shows $U_B * (U_B{}^\dagger) = 1_m\ 2$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) U_A-*is-gate*:
shows *gate 1* U_A
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) U_B-*is-gate*:
shows *gate 1* U_B
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) U_{AB}-*is-gate*:
shows *gate 2* $(U_A \bigotimes U_B)$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *psi-two-is-state*:
shows *state 2* ψ_2
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *J-cnj-is-gate*:
shows *gate 2* $(J^\dagger)$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *psi-f-is-state*:
shows *state 2* ψ_f
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *equation-one*:
shows $(J^\dagger) * ((U_A \bigotimes U_B) * (J * |$*unit-vec 4 0*$\rangle)) = \psi_f$
⟨*proof*⟩

abbreviation (**in** *strategic-space-2p*) *prob00* :: *complex Matrix.mat* ⇒ *real* **where**

prob00 v ≡ *if state 2 v then* (*cmod* (*v* $$ (*0*,*0*)))^2 *else undefined*

abbreviation (**in** *strategic-space-2p*) *prob01* :: *complex Matrix.mat* ⇒ *real* **where**
prob01 v ≡ *if state 2 v then* (*cmod* (*v* $$ (*1*,*0*)))^2 *else undefined*

abbreviation (**in** *strategic-space-2p*) *prob10* :: *complex Matrix.mat* ⇒ *real* **where**
prob10 v ≡ *if state 2 v then* (*cmod* (*v* $$ (*2*,*0*)))^2 *else undefined*

abbreviation (**in** *strategic-space-2p*) *prob11* :: *complex Matrix.mat* ⇒ *real* **where**
prob11 v ≡ *if state 2 v then* (*cmod* (*v* $$ (*3*,*0*)))^2 *else undefined*

definition (**in** *strategic-space-2p*) *alice-payoff* :: *real* **where**
alice-payoff ≡ *3* ∗ (*prob00* ψ_f) + *1* ∗ (*prob11* ψ_f) + *0* ∗ (*prob01* ψ_f) + *5* ∗ (*prob10* ψ_f)

definition (**in** *strategic-space-2p*) *bob-payoff* :: *real* **where**
bob-payoff ≡ *3* ∗ (*prob00* ψ_f) + *1* ∗ (*prob11* ψ_f) + *5* ∗ (*prob01* ψ_f) + *0* ∗ (*prob10* ψ_f)

definition (**in** *strategic-space-2p*) *is-nash-eq* :: *bool* **where**
is-nash-eq ≡
(∀ *tA pA*. *strategic-space-2p* γ *tA pA* ϑ_B φ_B ⟶
  alice-payoff ≥ *strategic-space-2p.alice-payoff* γ *tA pA* ϑ_B φ_B)
∧
(∀ *tB pB*. *strategic-space-2p* γ ϑ_A φ_A *tB pB* ⟶
  bob-payoff ≥ *strategic-space-2p.bob-payoff* γ ϑ_A φ_A *tB pB*)

definition (**in** *strategic-space-2p*) *is-pareto-opt* :: *bool* **where**
is-pareto-opt ≡ ∀ *tA pA tB pB*. *strategic-space-2p* γ *tA pA tB pB* ⟶
((*strategic-space-2p.alice-payoff* γ *tA pA tB pB* > *alice-payoff* ⟶
  strategic-space-2p.bob-payoff γ *tA pA tB pB* < *bob-payoff*) ∧
 (*strategic-space-2p.bob-payoff* γ *tA pA tB pB* > *bob-payoff* ⟶
  strategic-space-2p.alice-payoff γ *tA pA tB pB* < *alice-payoff*))

lemma (**in** *strategic-space-2p*) *sum-of-prob*:
  fixes *v* :: *complex Matrix.mat*
  assumes *state 2 v*
  shows (*prob00 v*) + (*prob11 v*) + (*prob01 v*) + (*prob10 v*) = *1*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *sum-payoff-le-6*:
  fixes *tA pA tB pB* :: *real*
  shows *alice-payoff* + *bob-payoff* ≤ *6*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *coop-is-pareto-opt*:
  assumes *alice-payoff* = *3* ∧ *bob-payoff* = *3*
  shows *is-pareto-opt*
  ⟨*proof*⟩

13.2 The Separable Case

lemma (**in** *strategic-space-2p*) *separable-case-CC*:
assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = 0 \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow$ *alice-payoff* $= 3 \wedge$ *bob-payoff* $= 3$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-case-DD*:
assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi \longrightarrow$ *alice-payoff* $= 1 \wedge$ *bob-payoff* $= 1$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-case-DC*:
assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow$ *alice-payoff* $= 5 \wedge$ *bob-payoff* $= 0$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-alice-payoff-*D_B:

assumes $\gamma = 0$ **and** $\varphi_B = 0 \wedge \vartheta_B = pi$
shows *alice-payoff* ≤ 1
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-bob-payoff-*D_A:

assumes $\gamma = 0$ **and** $\varphi_A = 0 \wedge \vartheta_A = pi$
shows *bob-payoff* ≤ 1
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-case-DD-alice-opt*:
assumes $\gamma = 0$ **and** $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi$
shows $\bigwedge tA\ pA.$ *strategic-space-2p* $\gamma\ tA\ pA\ \vartheta_B\ \varphi_B \longrightarrow$ *strategic-space-2p.alice-payoff* $\gamma\ tA\ pA\ \vartheta_B\ \varphi_B \leq$ *alice-payoff*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-case-DD-bob-opt*:
assumes $\gamma = 0$ **and** $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi$
shows $\bigwedge tB\ pB.$ *strategic-space-2p* $\gamma\ \vartheta_A\ \varphi_A\ tB\ pB \longrightarrow$ *strategic-space-2p.bob-payoff* $\gamma\ \vartheta_A\ \varphi_A\ tB\ pB \leq$ *bob-payoff*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-case-DD-is-nash-eq*:
assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi \longrightarrow$ *is-nash-eq*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-case-CC-is-not-nash-eq*:

assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = 0 \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow \neg$ *is-nash-eq*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *separable-case-CC-is-pareto-optimal*:
assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = 0 \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow$ *is-pareto-opt*
⟨*proof*⟩

13.3 The Maximally Entangled Case

lemma *exp-to-sin*:
fixes x:: *real*
shows *exp* (i $*$ x) $-$ *exp* ($-$(i $*$ x)) $=$ $2 *$ i $*$ (*sin* x)
⟨*proof*⟩

lemma *exp-to-cos*:
fixes x:: *real*
shows *exp* (i $*$ x) $+$ *exp* ($-$(i $*$ x)) $=$ $2 *$ (*cos* x)
⟨*proof*⟩

lemma *cmod-real-prod-squared*:
fixes x y:: *real*
shows (*cmod* (*complex-of-real* x $*$ *complex-of-real* y))^2 $= x^2 * y^2$
⟨*proof*⟩

lemma *quantum-payoff-simp*:
fixes x y:: *real*
shows $3 *$ (*cmod* (*complex-of-real* (*sin* x) $*$ *complex-of-real* (*cos* y)))^2 $+$
(*cmod* (*complex-of-real* (*cos* x) $*$ *complex-of-real* (*cos* y)))^2 $=$
$2 *$ (*sin* x)^2 $*$ (*cos* y)^2 $+$ (*cos* y)^2
⟨*proof*⟩

lemma *quantum-payoff-le-3*:
fixes x y:: *real*
shows $2 *$ (*sin* x)^2 $*$ (*cos* y)^2 $+$ (*cos* y)^2 ≤ 3
⟨*proof*⟩

lemma *sqrt-two-squared-cpx*: *complex-of-real* (*sqrt* 2) $*$ *complex-of-real* (*sqrt* 2) $=$ 2
⟨*proof*⟩

lemma *hidden-sqrt-two-squared-cpx*: *complex-of-real* (*sqrt* 2) $*$ (*complex-of-real* (*sqrt* 2) $*$ x) / 4 $=$ $x/2$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-DD*:

assumes $\gamma = pi/2$

shows $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi \longrightarrow$ *alice-payoff* $= 1 \wedge$ *bob-payoff* $= 1$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-QQ*:

assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \vartheta_A = 0 \wedge \varphi_B = pi/2 \wedge \vartheta_B = 0 \longrightarrow$ *alice-payoff* $= 3 \wedge$ *bob-payoff* $= 3$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-QD*:

assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \vartheta_A = 0 \wedge \varphi_B = 0 \wedge \vartheta_B = pi \longrightarrow$ *alice-payoff* $= 5 \wedge$ *bob-payoff* $= 0$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-alice-payoff-*Q_B:

assumes $\gamma = pi/2$
shows $\varphi_B = pi/2 \wedge \vartheta_B = 0 \longrightarrow$ *alice-payoff* ≤ 3
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-bob-payoff-*Q_A:

assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \vartheta_A = 0 \longrightarrow$ *bob-payoff* ≤ 3
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-DD-is-not-nash-eq*:
assumes $\gamma = pi/2$
shows $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi \longrightarrow \neg$*is-nash-eq*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-alice-opt*:
assumes $\gamma = pi/2$ **and** $\varphi_A = pi/2 \wedge \vartheta_A = 0 \wedge \varphi_B = pi/2 \wedge \vartheta_B = 0$
shows $\bigwedge tA\ pA.$ *strategic-space-2p* γ *tA pA* ϑ_B $\varphi_B \longrightarrow$ *strategic-space-2p.alice-payoff* γ *tA pA* ϑ_B $\varphi_B \leq$ *alice-payoff*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-bob-opt*:
assumes $\gamma = pi/2$ **and** $\varphi_A = pi/2 \wedge \vartheta_A = 0 \wedge \varphi_B = pi/2 \wedge \vartheta_B = 0$
shows $\bigwedge tB\ pB.$ *strategic-space-2p* γ ϑ_A φ_A *tB pB* $\longrightarrow$ *strategic-space-2p.bob-payoff* γ ϑ_A φ_A *tB pB* $\leq$ *bob-payoff*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-QQ-is-nash-eq*:
assumes $\gamma = pi/2$

shows $\varphi_A = pi/2 \wedge \vartheta_A = 0 \wedge \varphi_B = pi/2 \wedge \vartheta_B = 0 \longrightarrow$ *is-nash-eq*
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-QQ-is-pareto-optimal*:
assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \vartheta_A = 0 \wedge \varphi_B = pi/2 \wedge \vartheta_B = 0 \longrightarrow$ *is-pareto-opt*
⟨*proof*⟩

13.4 The Unfair Strategy Case

lemma *half-sqrt-two-squared*: $2 * (sqrt\ 2\ /\ 2)^2 = 1$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-MD*:

assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \vartheta_A = pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = pi \longrightarrow$ *alice-payoff* $= 3 \wedge$ *bob-payoff* $= 1/2$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-MC*:

assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \vartheta_A = pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow$ *alice-payoff* $= 3 \wedge$ *bob-payoff* $= 1/2$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *max-entangled-MH*:

assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \vartheta_A = pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = pi/2 \longrightarrow$ *alice-payoff* $= 1 \wedge$ *bob-payoff* $= 1$
⟨*proof*⟩

abbreviation M :: *complex Matrix.mat* **where**
$M \equiv$ *mat-of-cols-list* 2 $[[\mathrm{i} * sqrt(2)/2, -1 * sqrt(2)/2],$
$[1 * sqrt(2)/2, -\mathrm{i} * sqrt(2)/2]]$

lemma (**in** *strategic-space-2p*) *M-correct*:
assumes $\varphi_A = pi/2 \wedge \vartheta_A = pi/2$
shows $U_A = M$
⟨*proof*⟩

lemma *hidden-sqrt-two-squared-cpx2*:
fixes $x\ y$ :: *complex*
shows $(sqrt\ 2) * ((sqrt\ 2) * (x * y))\ /\ 2 = x * y$
⟨*proof*⟩

lemma (**in** *strategic-space-2p*) *unfair-strategy-no-benefit*:

assumes $\gamma = pi/2$
shows $\varphi_A = pi/2 \wedge \varphi_B = 0 \wedge \vartheta_A = \vartheta_B \longrightarrow alice\text{-}payoff = 1 \wedge bob\text{-}payoff = 1$
⟨*proof*⟩

end

14 Acknowledgements

The work has been jointly supported by the Cambridge Mathematics Placements (CMP) Programme and the ERC Advanced Grant ALEXANDRIA (Project GA 742178).

References

[1] J. Boender, F. Kammüller, and R. Nagarajan. Formalization of quantum protocols using coq. In *QPL*, 2015.

[2] J. Eisert, M. Wilkens, and M. Lewenstein. Quantum Games and Quantum Strategies. *Physical Review Letters*, 83:3077–3080, Oct. 1999.

[3] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.