Quantum Hoare Logic

Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying,
Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan

March 17, 2025

Abstract

We formalize quantum Hoare logic as given in [1]. In particular, we specify the syntax and denotational semantics of a simple model of quantum programs. Then, we write down the rules of quantum Hoare logic for partial correctness, and show the soundness and completeness of the resulting proof system. As an application, we verify the correctness of Grover's algorithm.

Contents

1 Complex matrices **2**
- 1.1 Trace of a matrix 2
- 1.2 Conjugate of a vector 3
- 1.3 Inner product 3
- 1.4 Hermitian adjoint of a matrix 5
- 1.5 Algebraic manipulations on matrices 6
- 1.6 Hermitian matrices 7
- 1.7 Inverse matrices 8
- 1.8 Unitary matrices 8
- 1.9 Normalization of vectors 9
- 1.10 Spectral decomposition of normal complex matrices 10
- 1.11 Outer product 13
- 1.12 Semi-definite matrices 15
- 1.13 Löwner partial order 17
- 1.14 Density operators 20

2 Matrix limits **21**
- 2.1 Definition of limit of matrices 21
- 2.2 Existence of least upper bound for the Löwner order 24
- 2.3 Finite sum of matrices 27
- 2.4 Measurement 29

3 Quantum programs **30**
3.1 Syntax . . . 30
3.2 Denotational semantics . . . 31

4 Partial state **38**
4.1 Encodings . . . 40
4.2 Tensor product of vectors and matrices . . . 42
4.3 Extension of matrices . . . 46
4.4 Partial tensor product . . . 47
4.5 Partial extensions . . . 50
4.6 Commands on subset of variables . . . 54

5 Standard gates **54**

6 Partial and total correctness **56**
6.1 Weakest liberal preconditions . . . 57
6.2 Hoare triples for partial correctness . . . 60
6.3 Consequences of completeness . . . 61

7 Grover's algorithm **61**
7.1 Basic definitions . . . 61
7.2 Grover operator . . . 65
7.3 State of Grover's algorithm . . . 67
7.4 Grover's algorithm . . . 70
7.5 Correctness . . . 73

1 Complex matrices

theory *Complex-Matrix*
imports
Jordan-Normal-Form.Matrix
Jordan-Normal-Form.Conjugate
Jordan-Normal-Form.Jordan-Normal-Form-Existence
begin

1.1 Trace of a matrix

definition *trace* :: *'a::ring mat* ⇒ *'a* **where**
trace A = (∑ *i* ∈ {*0* ..< *dim-row A*}. *A* $$ (*i*,*i*))

lemma *trace-zero* [*simp*]:
trace (0_m *n n*) = *0*
⟨*proof*⟩

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

lemma *trace-comm*:
  fixes A B :: $'a$::*comm-ring mat*
  assumes A: $A \in$ *carrier-mat n n* **and** B: $B \in$ *carrier-mat n n*
  shows *trace* $(A * B)$ = *trace* $(B * A)$
⟨*proof*⟩

lemma *trace-add-linear*:
  fixes A B :: $'a$::*comm-ring mat*
  assumes A: $A \in$ *carrier-mat n n* **and** B: $B \in$ *carrier-mat n n*
  shows *trace* $(A + B)$ = *trace* A + *trace* B (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *trace-minus-linear*:
  fixes A B :: $'a$::*comm-ring mat*
  assumes A: $A \in$ *carrier-mat n n* **and** B: $B \in$ *carrier-mat n n*
  shows *trace* $(A - B)$ = *trace* A − *trace* B (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *trace-smult*:
  assumes $A \in$ *carrier-mat n n*
  shows *trace* $(c \cdot_m A) = c *$ *trace* A
⟨*proof*⟩

1.2 Conjugate of a vector

lemma *conjugate-scalar-prod*:
  fixes v w :: $'a$::*conjugatable-ring vec*
  assumes *dim-vec* v = *dim-vec* w
  shows *conjugate* $(v \cdot w)$ = *conjugate* v · *conjugate* w
  ⟨*proof*⟩

1.3 Inner product

abbreviation *inner-prod* :: $'a$ *vec* ⇒ $'a$ *vec* ⇒ $'a$ :: *conjugatable-ring*
  where *inner-prod* v w ≡ w ·c v

lemma *conjugate-scalar-prod-Im* [*simp*]:
  Im (v ·c v) = *0*
  ⟨*proof*⟩

lemma *conjugate-scalar-prod-Re* [*simp*]:
  Re (v ·c v) ≥ *0*
  ⟨*proof*⟩

lemma *self-cscalar-prod-geq-0*:
  fixes v :: $'a$::*conjugatable-ordered-field vec*
  shows v ·c v ≥ *0*
  ⟨*proof*⟩

lemma *inner-prod-distrib-left*:
fixes $u\ v\ w :: ('a::conjugatable\text{-}field)\ vec$
assumes *dimu*: $u \in carrier\text{-}vec\ n$ **and** *dimv*:$v \in carrier\text{-}vec\ n$ **and** *dimw*: $w \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ (v + w)\ u = inner\text{-}prod\ v\ u + inner\text{-}prod\ w\ u$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

lemma *inner-prod-distrib-right*:
fixes $u\ v\ w :: ('a::conjugatable\text{-}field)\ vec$
assumes *dimu*: $u \in carrier\text{-}vec\ n$ **and** *dimv*:$v \in carrier\text{-}vec\ n$ **and** *dimw*: $w \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ u\ (v + w) = inner\text{-}prod\ u\ v + inner\text{-}prod\ u\ w$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

lemma *inner-prod-minus-distrib-right*:
fixes $u\ v\ w :: ('a::conjugatable\text{-}field)\ vec$
assumes *dimu*: $u \in carrier\text{-}vec\ n$ **and** *dimv*:$v \in carrier\text{-}vec\ n$ **and** *dimw*: $w \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ u\ (v - w) = inner\text{-}prod\ u\ v - inner\text{-}prod\ u\ w$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

lemma *inner-prod-smult-right*:
fixes $u\ v :: complex\ vec$
assumes *dimu*: $u \in carrier\text{-}vec\ n$ **and** *dimv*:$v \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ (a \cdot_v u)\ v = conjugate\ a * inner\text{-}prod\ u\ v$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

lemma *inner-prod-smult-left*:
fixes $u\ v :: complex\ vec$
assumes *dimu*: $u \in carrier\text{-}vec\ n$ **and** *dimv*: $v \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ u\ (a \cdot_v v) = a * inner\text{-}prod\ u\ v$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

lemma *inner-prod-smult-left-right*:
fixes $u\ v :: complex\ vec$
assumes *dimu*: $u \in carrier\text{-}vec\ n$ **and** *dimv*: $v \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ (a \cdot_v u)\ (b \cdot_v v) = conjugate\ a * b\ \ * inner\text{-}prod\ u\ v$ (**is** $?lhs = ?rhs$)
⟨*proof*⟩

lemma *inner-prod-swap*:
fixes $x\ y :: complex\ vec$
assumes $y \in carrier\text{-}vec\ n$ **and** $x \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ y\ x = conjugate\ (inner\text{-}prod\ x\ y)$
⟨*proof*⟩

Cauchy-Schwarz theorem for complex vectors. This is analogous to aux_Cauchy and Cauchy_Schwarz_ineq in Generalizations2.thy in QR_Decomposition. Consider merging and moving to Isabelle library.

lemma *aux-Cauchy*:
fixes *x y* :: *complex vec*
assumes $x \in$ *carrier-vec n* **and** $y \in$ *carrier-vec n*
shows $0 \leq$ *inner-prod x x* + *a* ∗ (*inner-prod x y*) + (*cnj a*) ∗ ((*cnj* (*inner-prod x y*)) + *a* ∗ (*inner-prod y y*))
⟨*proof*⟩

lemma *Cauchy-Schwarz-complex-vec*:
fixes *x y* :: *complex vec*
assumes $x \in$ *carrier-vec n* **and** $y \in$ *carrier-vec n*
shows *inner-prod x y* ∗ *inner-prod y x* ≤ *inner-prod x x* ∗ *inner-prod y y*
⟨*proof*⟩

1.4 Hermitian adjoint of a matrix

abbreviation *adjoint* **where** *adjoint* ≡ *mat-adjoint*

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

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

lemma *adjoint-dim*:
A ∈ *carrier-mat n n* ⟹ *adjoint A* ∈ *carrier-mat n n*
⟨*proof*⟩

lemma *adjoint-def*:
adjoint A = *mat* (*dim-col A*) (*dim-row A*) (λ(*i*,*j*). *conjugate* (*A* $$ (*j*,*i*)))
⟨*proof*⟩

lemma *adjoint-eval*:
assumes *i* < *dim-col A j* < *dim-row A*
shows (*adjoint A*) $$ (*i*,*j*) = *conjugate* (*A* $$ (*j*,*i*))
⟨*proof*⟩

lemma *adjoint-row*:
assumes *i* < *dim-col A*
shows *row* (*adjoint A*) *i* = *conjugate* (*col A i*)
⟨*proof*⟩

lemma *adjoint-col*:
assumes *i* < *dim-row A*
shows *col* (*adjoint A*) *i* = *conjugate* (*row A i*)
⟨*proof*⟩

The identity <v, A w> = <A* v, w>

lemma *adjoint-def-alter*:
fixes *v w* :: *'a*::*conjugatable-field vec*
and *A* :: *'a*::*conjugatable-field mat*

assumes *dims*: $v \in$ *carrier-vec n* $w \in$ *carrier-vec m* $A \in$ *carrier-mat n m*
shows *inner-prod v* $(A *_v w) =$ *inner-prod* (*adjoint* $A *_v v$) *w* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *adjoint-one*:
shows *adjoint* $(1_m\ n) = (1_m\ n$::*complex mat*)
⟨*proof*⟩

lemma *adjoint-scale*:
fixes *A* :: *'a::conjugatable-field mat*
shows *adjoint* $(a \cdot_m A) =$ (*conjugate a*) $\cdot_m$ *adjoint A*
⟨*proof*⟩

lemma *adjoint-add*:
fixes *A B* :: *'a::conjugatable-field mat*
assumes $A \in$ *carrier-mat n m* $B \in$ *carrier-mat n m*
shows *adjoint* $(A + B) =$ *adjoint A* + *adjoint B*
⟨*proof*⟩

lemma *adjoint-minus*:
fixes *A B* :: *'a::conjugatable-field mat*
assumes $A \in$ *carrier-mat n m* $B \in$ *carrier-mat n m*
shows *adjoint* $(A - B) =$ *adjoint A* − *adjoint B*
⟨*proof*⟩

lemma *adjoint-mult*:
fixes *A B* :: *'a::conjugatable-field mat*
assumes $A \in$ *carrier-mat n m* $B \in$ *carrier-mat m l*
shows *adjoint* $(A * B) =$ *adjoint B* ∗ *adjoint A*
⟨*proof*⟩

lemma *adjoint-adjoint*:
fixes *A* :: *'a::conjugatable-field mat*
shows *adjoint* (*adjoint A*) = *A*
⟨*proof*⟩

lemma *trace-adjoint-positive*:
fixes *A* :: *complex mat*
shows *trace* (*A* ∗ *adjoint A*) ≥ *0*
⟨*proof*⟩

1.5 Algebraic manipulations on matrices

lemma *right-add-zero-mat*[*simp*]:
(*A* :: *'a* :: *monoid-add mat*) ∈ *carrier-mat nr nc* ⟹ $A + 0_m$ *nr nc* = *A*
⟨*proof*⟩

lemma *add-carrier-mat'*:
$A \in$ *carrier-mat nr nc* ⟹ $B \in$ *carrier-mat nr nc* ⟹ $A + B \in$ *carrier-mat nr*

nc
⟨*proof*⟩

lemma *minus-carrier-mat′*:
$A \in$ *carrier-mat nr nc* $\Longrightarrow$ $B \in$ *carrier-mat nr nc* $\Longrightarrow$ $A - B \in$ *carrier-mat nr nc*
⟨*proof*⟩

lemma *swap-plus-mat*:
fixes $A\ B\ C$ :: *′a::semiring-1 mat*
assumes $A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n* $C \in$ *carrier-mat n n*
shows $A + B + C = A + C + B$
⟨*proof*⟩

lemma *uminus-mat*:
fixes A :: *complex mat*
assumes $A \in$ *carrier-mat n n*
shows $-A = (-1) \cdot_m A$
⟨*proof*⟩

⟨*ML*⟩

lemma *mat-assoc-test*:
fixes $A\ B\ C\ D$ :: *complex mat*
assumes $A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n* $C \in$ *carrier-mat n n* $D \in$ *carrier-mat n n*
shows
$(A * B) * (C * D) = A * B * C * D$
adjoint $(A *$ *adjoint* $B) * C = B * ($*adjoint* $A * C)$
$A * 1_m\ n * 1_m\ n * B * 1_m\ n = A * B$
$(A - B) + (B - C) = A + (-B) + B + (-C)$
$A + (B - C) = A + B - C$
$A - (B + C + D) = A - B - C - D$
$(A + B) * (B + C) = A * B + B * B + A * C + B * C$
$A - B = A + (-1) \cdot_m B$
$A * (B - C) * D = A * B * D - A * C * D$
trace $(A * B * C) =$ *trace* $(B * C * A)$
trace $(A * B * C * D) =$ *trace* $(C * D * A * B)$
trace $(A + B * C) =$ *trace* $A +$ *trace* $(C * B)$
$A + B = B + A$
$A + B + C = C + B + A$
$A + B + (C + D) = A + C + (B + D)$
⟨*proof*⟩

1.6 Hermitian matrices

A Hermitian matrix is a matrix that is equal to its Hermitian adjoint.

definition *hermitian* :: *′a::conjugatable-field mat* $\Rightarrow$ *bool* **where**
hermitian $A \longleftrightarrow$ (*adjoint* $A = A$)

lemma *hermitian-one*:
  shows *hermitian* $((1_m\ n)::('a::conjugatable\text{-}field\ mat))$
  ⟨*proof*⟩

1.7 Inverse matrices

lemma *inverts-mat-symm*:
  fixes $A\ B$ :: *'a::field mat*
  assumes *dim*: $A \in carrier\text{-}mat\ n\ n\ B \in carrier\text{-}mat\ n\ n$
    and *AB*: *inverts-mat A B*
  shows *inverts-mat B A*
⟨*proof*⟩

lemma *inverts-mat-unique*:
  fixes $A\ B\ C$ :: *'a::field mat*
  assumes *dim*: $A \in carrier\text{-}mat\ n\ n\ B \in carrier\text{-}mat\ n\ n\ C \in carrier\text{-}mat\ n\ n$
    and *AB*: *inverts-mat A B* **and** *AC*: *inverts-mat A C*
  shows $B = C$
⟨*proof*⟩

1.8 Unitary matrices

A unitary matrix is a matrix whose Hermitian adjoint is also its inverse.

definition *unitary* :: *'a::conjugatable-field mat* ⇒ *bool* **where**
  unitary A ⟷ $A \in carrier\text{-}mat\ (dim\text{-}row\ A)\ (dim\text{-}row\ A) \wedge inverts\text{-}mat\ A\ (adjoint\ A)$

lemma *unitaryD2*:
  assumes $A \in carrier\text{-}mat\ n\ n$
  shows *unitary A* ⟹ *inverts-mat* (*adjoint A*) *A*
  ⟨*proof*⟩

lemma *unitary-simps* [*simp*]:
  $A \in carrier\text{-}mat\ n\ n \Longrightarrow unitary\ A \Longrightarrow adjoint\ A * A = 1_m\ n$
  $A \in carrier\text{-}mat\ n\ n \Longrightarrow unitary\ A \Longrightarrow A * adjoint\ A = 1_m\ n$
  ⟨*proof*⟩

lemma *unitary-adjoint* [*simp*]:
  assumes $A \in carrier\text{-}mat\ n\ n$ *unitary A*
  shows *unitary* (*adjoint A*)
  ⟨*proof*⟩

lemma *unitary-one*:
  shows *unitary* $((1_m\ n)::('a::conjugatable\text{-}field\ mat))$
  ⟨*proof*⟩

lemma *unitary-zero*:
  fixes *A* :: *'a::conjugatable-field mat*

assumes *A* ∈ *carrier-mat 0 0*
shows *unitary A*
⟨*proof*⟩

lemma *unitary-elim*:
assumes *dims*: *A* ∈ *carrier-mat n n B* ∈ *carrier-mat n n P* ∈ *carrier-mat n n*
and *uP*: *unitary P* **and** *eq*: *P* ∗ *A* ∗ *adjoint P* = *P* ∗ *B* ∗ *adjoint P*
shows *A* = *B*
⟨*proof*⟩

lemma *unitary-is-corthogonal*:
fixes *U* :: *'a::conjugatable-field mat*
assumes *dim*: *U* ∈ *carrier-mat n n*
and *U*: *unitary U*
shows *corthogonal-mat U*
⟨*proof*⟩

lemma *unitary-times-unitary*:
fixes *P Q* :: *'a:: conjugatable-field mat*
assumes *dim*: *P* ∈ *carrier-mat n n Q* ∈ *carrier-mat n n*
and *uP*: *unitary P* **and** *uQ*: *unitary Q*
shows *unitary* (*P* ∗ *Q*)
⟨*proof*⟩

lemma *unitary-operator-keep-trace*:
fixes *U A* :: *complex mat*
assumes *dU*: *U* ∈ *carrier-mat n n* **and** *dA*: *A* ∈ *carrier-mat n n* **and** *u*: *unitary U*
shows *trace A* = *trace* (*adjoint U* ∗ *A* ∗ *U*)
⟨*proof*⟩

1.9 Normalization of vectors

definition *vec-norm* :: *complex vec* ⇒ *complex* **where**
vec-norm v ≡ *csqrt* (*v* ·c *v*)

lemma *vec-norm-geq-0*:
fixes *v* :: *complex vec*
shows *vec-norm v* ≥ *0*
⟨*proof*⟩

lemma *vec-norm-zero*:
fixes *v* :: *complex vec*
assumes *dim*: *v* ∈ *carrier-vec n*
shows *vec-norm v* = *0* ⟷ *v* = 0_v *n*
⟨*proof*⟩

lemma *vec-norm-ge-0*:
fixes *v* :: *complex vec*

assumes *dim-v*: $v \in$ *carrier-vec n* **and** *neq0*: $v \neq 0_v$ *n*
shows *vec-norm* $v > 0$
⟨*proof*⟩

definition *vec-normalize* :: *complex vec* ⇒ *complex vec* **where**
vec-normalize $v = ($*if* $(v = 0_v$ (*dim-vec* $v))$ *then* v *else* 1 / (*vec-norm* $v) \cdot_v v)$

lemma *normalized-vec-dim*[*simp*]:
assumes (v::*complex vec*) $\in$ *carrier-vec n*
shows *vec-normalize* $v \in$ *carrier-vec n*
⟨*proof*⟩

lemma *vec-eq-norm-smult-normalized*:
shows $v =$ *vec-norm* $v \cdot_v$ *vec-normalize* v
⟨*proof*⟩

lemma *normalized-cscalar-prod*:
fixes *v w* :: *complex vec*
assumes *dim-v*: $v \in$ *carrier-vec n* **and** *dim-w*: $w \in$ *carrier-vec n*
shows v ∙c $w =$ (*vec-norm* v ∗ *vec-norm* w) ∗ (*vec-normalize* v ∙c *vec-normalize* w)
⟨*proof*⟩

lemma *normalized-vec-norm* :
fixes *v* :: *complex vec*
assumes *dim-v*: $v \in$ *carrier-vec n*
and *neq0*: $v \neq 0_v$ *n*
shows *vec-normalize* v ∙c *vec-normalize* $v = 1$
⟨*proof*⟩

lemma *normalize-zero*:
assumes $v \in$ *carrier-vec n*
shows *vec-normalize* $v = 0_v$ $n \longleftrightarrow v = 0_v$ n
⟨*proof*⟩

lemma *normalize-normalize*[*simp*]:
vec-normalize (*vec-normalize* v) = *vec-normalize* v
⟨*proof*⟩

1.10 Spectral decomposition of normal complex matrices

lemma *normalize-keep-corthogonal*:
fixes *vs* :: *complex vec list*
assumes *cor*: *corthogonal vs* **and** *dims*: *set vs* ⊆ *carrier-vec n*
shows *corthogonal* (*map vec-normalize vs*)
⟨*proof*⟩

lemma *normalized-corthogonal-mat-is-unitary*:
assumes *W*: *set ws* ⊆ *carrier-vec n*

```
    and orth: corthogonal ws
    and len: length ws = n
  shows unitary (mat-of-cols n (map vec-normalize ws)) (is unitary ?W)
⟨proof⟩

lemma normalize-keep-eigenvector:
  assumes ev: eigenvector A v e
    and dim: A ∈ carrier-mat n n v ∈ carrier-vec n
  shows eigenvector A (vec-normalize v) e
  ⟨proof⟩

lemma four-block-mat-adjoint:
  fixes A B C D :: 'a::conjugatable-field mat
  assumes dim: A ∈ carrier-mat nr1 nc1 B ∈ carrier-mat nr1 nc2
  C ∈ carrier-mat nr2 nc1 D ∈ carrier-mat nr2 nc2
  shows adjoint (four-block-mat A B C D)
    = four-block-mat (adjoint A) (adjoint C) (adjoint B) (adjoint D)
  ⟨proof⟩

fun unitary-schur-decomposition :: complex mat ⇒ complex list ⇒ complex mat ×
complex mat × complex mat where
  unitary-schur-decomposition A [] = (A, 1_m (dim-row A), 1_m (dim-row A))
| unitary-schur-decomposition A (e # es) = (let
       n = dim-row A;
       n1 = n − 1;
       v' = find-eigenvector A e;
       v = vec-normalize v';
       ws0 = gram-schmidt n (basis-completion v);
       ws = map vec-normalize ws0;
       W = mat-of-cols n ws;
       W' = corthogonal-inv W;
       A' = W' ∗ A ∗ W;
       (A1,A2,A0,A3) = split-block A' 1 1;
       (B,P,Q) = unitary-schur-decomposition A3 es;
       z-row = (0_m 1 n1);
       z-col = (0_m n1 1);
       one-1 = 1_m 1
     in (four-block-mat A1 (A2 ∗ P) A0 B,
     W ∗ four-block-mat one-1 z-row z-col P,
     four-block-mat one-1 z-row z-col Q ∗ W'))

theorem unitary-schur-decomposition:
  assumes A: (A::complex mat) ∈ carrier-mat n n
      and c: char-poly A = (∏ (e :: complex) ← es. [:− e, 1:])
      and B: unitary-schur-decomposition A es = (B,P,Q)
   shows similar-mat-wit A B P Q ∧ upper-triangular B ∧ diag-mat B = es ∧
unitary P ∧ (Q = adjoint P)
  ⟨proof⟩
```

lemma *complex-mat-char-poly-factorizable*:
  fixes A :: *complex mat*
  assumes $A \in$ *carrier-mat n n*
  shows $\exists$ *as. char-poly* $A = (\prod a \leftarrow as.\ [:- a, 1:]) \wedge$ *length as* $= n$
⟨*proof*⟩

lemma *complex-mat-has-unitary-schur-decomposition*:
  fixes A :: *complex mat*
  assumes $A \in$ *carrier-mat n n*
  shows $\exists$ *B P es. similar-mat-wit A B P* (*adjoint P*) $\wedge$ *unitary P*
    $\wedge$ *char-poly* $A = (\prod (e :: complex) \leftarrow es.\ [:- e, 1:]) \wedge$ *diag-mat B* $=$ *es*
⟨*proof*⟩

lemma *normal-upper-triangular-matrix-is-diagonal*:
  fixes A :: *'a::conjugatable-ordered-field mat*
  assumes $A \in$ *carrier-mat n n*
    and *tri*: *upper-triangular A*
    and *norm*: $A *$ *adjoint* $A =$ *adjoint* $A * A$
  shows *diagonal-mat A*
⟨*proof*⟩

lemma *normal-complex-mat-has-spectral-decomposition*:
  assumes *A*: (*A::complex mat*) $\in$ *carrier-mat n n*
    and *normal*: $A *$ *adjoint* $A =$ *adjoint* $A * A$
    and *c*: *char-poly* $A = (\prod (e :: complex) \leftarrow es.\ [:- e, 1:])$
    and *B*: *unitary-schur-decomposition A es* $= (B,P,Q)$
  shows *similar-mat-wit A B P* (*adjoint P*) $\wedge$ *diagonal-mat B* $\wedge$ *diag-mat B* $=$ *es*
$\wedge$ *unitary P*
⟨*proof*⟩

lemma *complex-mat-has-jordan-nf*:
  fixes A :: *complex mat*
  assumes $A \in$ *carrier-mat n n*
  shows $\exists$ *n-as. jordan-nf A n-as*
⟨*proof*⟩

lemma *hermitian-is-normal*:
  assumes *hermitian A*
  shows $A *$ *adjoint* $A =$ *adjoint* $A * A$
  ⟨*proof*⟩

lemma *hermitian-eigenvalue-real*:
  assumes *dim*: (*A::complex mat*) $\in$ *carrier-mat n n*
    and *hA*: *hermitian A*
    and *c*: *char-poly* $A = (\prod (e :: complex) \leftarrow es.\ [:- e, 1:])$
    and *B*: *unitary-schur-decomposition A es* $= (B,P,Q)$
  shows *similar-mat-wit A B P* (*adjoint P*) $\wedge$ *diagonal-mat B* $\wedge$ *diag-mat B* $=$ *es*
    $\wedge$ *unitary P* $\wedge$ ($\forall i < n.$ *B$$(i, i)* $\in$ *Reals*)
⟨*proof*⟩

lemma *hermitian-inner-prod-real*:
assumes *dimA*: (*A*::*complex mat*) ∈ *carrier-mat n n*
and *dimv*: *v* ∈ *carrier-vec n*
and *hA*: *hermitian A*
shows *inner-prod v* ($A *_v v$) ∈ *Reals*
⟨*proof*⟩

lemma *unit-vec-bracket*:
fixes *A* :: *complex mat*
assumes *dimA*: *A* ∈ *carrier-mat n n* **and** *i*: *i* < *n*
shows *inner-prod* (*unit-vec n i*) ($A *_v$ (*unit-vec n i*)) = *A*$$(*i*, *i*)
⟨*proof*⟩

lemma *spectral-decomposition-extract-diag*:
fixes *P B* :: *complex mat*
assumes *dimP*: *P* ∈ *carrier-mat n n* **and** *dimB*: *B* ∈ *carrier-mat n n*
and *uP*: *unitary P* **and** *dB*: *diagonal-mat B* **and** *i*: *i* < *n*
shows *inner-prod* (*col P i*) (*P* * *B* * (*adjoint P*) $*_v$ (*col P i*)) = *B*$$(*i*, *i*)
⟨*proof*⟩

lemma *hermitian-inner-prod-zero*:
fixes *A* :: *complex mat*
assumes *dimA*: *A* ∈ *carrier-mat n n* **and** *hA*: *hermitian A*
and *zero*: ∀ *v*∈*carrier-vec n*. *inner-prod v* ($A *_v v$) = *0*
shows *A* = 0_m *n n*
⟨*proof*⟩

lemma *complex-mat-decomposition-to-hermitian*:
fixes *A* :: *complex mat*
assumes *dim*: *A* ∈ *carrier-mat n n*
shows ∃ *B C*. *hermitian B* ∧ *hermitian C* ∧ *A* = *B* + i $\cdot_m$ *C* ∧ *B* ∈ *carrier-mat n n* ∧ *C* ∈ *carrier-mat n n*
⟨*proof*⟩

1.11 Outer product

definition *outer-prod* :: *′a*::*conjugatable-field vec* ⇒ *′a vec* ⇒ *′a mat* **where**
outer-prod v w = *mat* (*dim-vec v*) *1* (λ(*i*, *j*). *v* $ *i*) * *mat 1* (*dim-vec w*) (λ(*i*, *j*). (*conjugate w*) $ *j*)

lemma *outer-prod-dim*[*simp*]:
fixes *v w* :: *′a*::*conjugatable-field vec*
assumes *v*: *v* ∈ *carrier-vec n* **and** *w*: *w* ∈ *carrier-vec m*
shows *outer-prod v w* ∈ *carrier-mat n m*
⟨*proof*⟩

lemma *mat-of-vec-mult-eq-scalar-prod*:
fixes *v w* :: *′a*::*conjugatable-field vec*

assumes $v \in$ *carrier-vec n* **and** $w \in$ *carrier-vec n*
shows *mat 1* (*dim-vec v*) ($\lambda(i, j)$. (*conjugate v*) \$ *j*) $*$ *mat* (*dim-vec w*) *1* ($\lambda(i, j)$. *w* \$ *i*)
= *mat 1 1* (λk. *inner-prod v w*)
⟨*proof*⟩

lemma *one-dim-mat-mult-is-scale*:
fixes *A B* :: ('*a*::*conjugatable-field mat*)
assumes $B \in$ *carrier-mat 1 n*
shows (*mat 1 1* (λk. *a*)) $*$ B = $a \cdot_m B$
⟨*proof*⟩

lemma *outer-prod-mult-outer-prod*:
fixes *a b c d* :: '*a*::*conjugatable-field vec*
assumes *a*: $a \in$ *carrier-vec d1* **and** *b*: $b \in$ *carrier-vec d2*
and *c*: $c \in$ *carrier-vec d2* **and** *d*: $d \in$ *carrier-vec d3*
shows *outer-prod a b* $*$ *outer-prod c d* = *inner-prod b c* $\cdot_m$ *outer-prod a d*
⟨*proof*⟩

lemma *index-outer-prod*:
fixes *v w* :: '*a*::*conjugatable-field vec*
assumes *v*: $v \in$ *carrier-vec n* **and** *w*: $w \in$ *carrier-vec m*
and *ij*: $i < n$ $j < m$
shows (*outer-prod v w*)\$\$(i, j) = *v* \$ *i* $*$ *conjugate* (*w* \$ *j*)
⟨*proof*⟩

lemma *mat-of-vec-mult-vec*:
fixes *a b c* :: '*a*::*conjugatable-field vec*
assumes *a*: $a \in$ *carrier-vec d* **and** *b*: $b \in$ *carrier-vec d*
shows *mat 1 d* ($\lambda(i, j)$. (*conjugate a*) \$ *j*) $*_v$ *b* = *vec 1* (λk. *inner-prod a b*)
⟨*proof*⟩

lemma *mat-of-vec-mult-one-dim-vec*:
fixes *a b* :: '*a*::*conjugatable-field vec*
assumes *a*: $a \in$ *carrier-vec d*
shows *mat d 1* ($\lambda(i, j)$. *a* \$ *i*) $*_v$ *vec 1* (λk. *c*) = $c \cdot_v a$
⟨*proof*⟩

lemma *outer-prod-mult-vec*:
fixes *a b c* :: '*a*::*conjugatable-field vec*
assumes *a*: $a \in$ *carrier-vec d1* **and** *b*: $b \in$ *carrier-vec d2*
and *c*: $c \in$ *carrier-vec d2*
shows *outer-prod a b* $*_v$ *c* = *inner-prod b c* $\cdot_v$ *a*
⟨*proof*⟩

lemma *trace-outer-prod-right*:
fixes *A* :: '*a*::*conjugatable-field mat* **and** *v w* :: '*a vec*
assumes *A*: $A \in$ *carrier-mat n n*
and *v*: $v \in$ *carrier-vec n* **and** *w*: $w \in$ *carrier-vec n*

shows *trace* (*A* ∗ *outer-prod v w*) = *inner-prod w* (*A* $*_v$ *v*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *trace-outer-prod*:
fixes *v w* :: (*'a::conjugatable-field vec*)
assumes *v*: *v* ∈ *carrier-vec n* **and** *w*: *w* ∈ *carrier-vec n*
shows *trace* (*outer-prod v w*) = *inner-prod w v* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *inner-prod-outer-prod*:
fixes *a b c d* :: *'a::conjugatable-field vec*
assumes *a*: *a* ∈ *carrier-vec n* **and** *b*: *b* ∈ *carrier-vec n*
and *c*: *c* ∈ *carrier-vec m* **and** *d*: *d* ∈ *carrier-vec m*
shows *inner-prod a* (*outer-prod b c* $*_v$ *d*) = *inner-prod a b* ∗ *inner-prod c d* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

1.12 Semi-definite matrices

definition *positive* :: *complex mat* ⇒ *bool* **where**
positive A ⟷
A ∈ *carrier-mat* (*dim-col A*) (*dim-col A*) ∧
(∀ *v*. *dim-vec v* = *dim-col A* ⟶ *inner-prod v* (*A* $*_v$ *v*) ≥ *0*)

lemma *positive-iff-normalized-vec*:
positive A ⟷
A ∈ *carrier-mat* (*dim-col A*) (*dim-col A*) ∧
(∀ *v*. (*dim-vec v* = *dim-col A* ∧ *vec-norm v* = *1*) ⟶ *inner-prod v* (*A* $*_v$ *v*) ≥ *0*)
⟨*proof*⟩

lemma *positive-is-hermitian*:
fixes *A* :: *complex mat*
assumes *pA*: *positive A*
shows *hermitian A*
⟨*proof*⟩

lemma *positive-eigenvalue-positive*:
assumes *dimA*: (*A::complex mat*) ∈ *carrier-mat n n*
and *pA*: *positive A*
and *c*: *char-poly A* = (∏ (*e* :: *complex*) ← *es*. [:− *e*, *1*:])
and *B*: *unitary-schur-decomposition A es* = (*B*,*P*,*Q*)
shows ⋀*i*. *i* < *n* ⟹ *B*$$(*i*, *i*) ≥ *0*
⟨*proof*⟩

lemma *diag-mat-mult-diag-mat*:
fixes *B D* :: *'a::semiring-0 mat*
assumes *dimB*: *B* ∈ *carrier-mat n n* **and** *dimD*: *D* ∈ *carrier-mat n n*
and *dB*: *diagonal-mat B* **and** *dD*: *diagonal-mat D*

shows $B * D = mat\ n\ n\ (\lambda(i,j).\ (if\ i = j\ then\ (B\$\$(i,\ i)) * (D\$\$(i,\ i))\ else\ 0))$
⟨*proof*⟩

lemma *positive-only-if-decomp*:
assumes *dimA*: $A \in carrier\text{-}mat\ n\ n$ **and** *pA*: *positive A*
shows $\exists M \in carrier\text{-}mat\ n\ n.\ M * adjoint\ M = A$
⟨*proof*⟩

lemma *positive-if-decomp*:
assumes *dimA*: $A \in carrier\text{-}mat\ n\ n$ **and** $\exists M.\ M * adjoint\ M = A$
shows *positive A*
⟨*proof*⟩

lemma *positive-iff-decomp*:
assumes *dimA*: $A \in carrier\text{-}mat\ n\ n$
shows $positive\ A \longleftrightarrow (\exists M{\in}carrier\text{-}mat\ n\ n.\ M * adjoint\ M = A)$
⟨*proof*⟩

lemma *positive-dim-eq*:
assumes *positive A*
shows $dim\text{-}row\ A = dim\text{-}col\ A$
⟨*proof*⟩

lemma *positive-zero*:
$positive\ (0_m\ n\ n)$
⟨*proof*⟩

lemma *positive-one*:
$positive\ (1_m\ n)$
⟨*proof*⟩

lemma *positive-antisym*:
assumes *pA*: *positive A* **and** *pnA*: $positive\ (-A)$
shows $A = 0_m\ (dim\text{-}col\ A)\ (dim\text{-}col\ A)$
⟨*proof*⟩

lemma *positive-add*:
assumes *pA*: *positive A* **and** *pB*: *positive B*
and *dimA*: $A \in carrier\text{-}mat\ n\ n$ **and** *dimB*: $B \in carrier\text{-}mat\ n\ n$
shows $positive\ (A + B)$
⟨*proof*⟩

lemma *positive-trace*:
assumes $A \in carrier\text{-}mat\ n\ n$ **and** *positive A*
shows $trace\ A \geq 0$
⟨*proof*⟩

lemma *positive-close-under-left-right-mult-adjoint*:
fixes *M A* :: *complex mat*

assumes *dM*: $M \in$ *carrier-mat n n* **and** *dA*: $A \in$ *carrier-mat n n*
and *pA*: *positive A*
shows *positive* ($M * A *$ *adjoint M*)
⟨*proof*⟩

lemma *positive-same-outer-prod*:
fixes *v w* :: *complex vec*
assumes *v*: $v \in$ *carrier-vec n*
shows *positive* (*outer-prod v v*)
⟨*proof*⟩

lemma *smult-smult-mat*:
fixes *k* :: *complex* **and** *l* :: *complex*
assumes $A \in$ *carrier-mat nr n*
shows $k \cdot_m (l \cdot_m A) = (k * l) \cdot_m A$ ⟨*proof*⟩

lemma *positive-smult*:
assumes $A \in$ *carrier-mat n n*
and *positive A*
and $c \geq 0$
shows *positive* $(c \cdot_m A)$
⟨*proof*⟩

Version of previous theorem for real numbers

lemma *positive-scale*:
fixes *c* :: *real*
assumes $A \in$ *carrier-mat n n*
and *positive A*
and $c \geq 0$
shows *positive* $(c \cdot_m A)$
⟨*proof*⟩

1.13 Löwner partial order

definition *lowner-le* :: *complex mat* ⇒ *complex mat* ⇒ *bool* (**infix** ‹$\leq_L$› *50*)
where
$A \leq_L B \longleftrightarrow$ *dim-row* $A =$ *dim-row* $B \wedge$ *dim-col* $A =$ *dim-col* $B \wedge$ *positive* $(B - A)$

lemma *lowner-le-refl*:
assumes $A \in$ *carrier-mat n n*
shows $A \leq_L A$
⟨*proof*⟩

lemma *lowner-le-antisym*:
assumes *A*: $A \in$ *carrier-mat n n* **and** *B*: $B \in$ *carrier-mat n n*
and *L1*: $A \leq_L B$ **and** *L2*: $B \leq_L A$
shows $A = B$
⟨*proof*⟩

lemma *lowner-le-inner-prod-le*:
fixes A B :: *complex mat* **and** v :: *complex vec*
assumes A: $A \in$ *carrier-mat n n* **and** B: $B \in$ *carrier-mat n n*
and v: $v \in$ *carrier-vec n*
and $A \leq_L B$
shows *inner-prod v* $(A *_v v) \leq$ *inner-prod v* $(B *_v v)$
⟨*proof*⟩

lemma *lowner-le-trans*:
fixes A B C :: *complex mat*
assumes A: $A \in$ *carrier-mat n n* **and** B: $B \in$ *carrier-mat n n* **and** C: $C \in$ *carrier-mat n n*
and *L1*: $A \leq_L B$ **and** *L2*: $B \leq_L C$
shows $A \leq_L C$
⟨*proof*⟩

lemma *lowner-le-imp-trace-le*:
assumes $A \in$ *carrier-mat n n* **and** $B \in$ *carrier-mat n n*
and $A \leq_L B$
shows *trace* $A \leq$ *trace* B
⟨*proof*⟩

lemma *lowner-le-add*:
assumes $A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n* $C \in$ *carrier-mat n n* $D \in$ *carrier-mat n n*
and $A \leq_L B$ $C \leq_L D$
shows $A + C \leq_L B + D$
⟨*proof*⟩

lemma *lowner-le-swap*:
assumes $A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n*
and $A \leq_L B$
shows $-B \leq_L -A$
⟨*proof*⟩

lemma *lowner-le-minus*:
assumes $A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n* $C \in$ *carrier-mat n n* $D \in$ *carrier-mat n n*
and $A \leq_L B$ $C \leq_L D$
shows $A - D \leq_L B - C$
⟨*proof*⟩

lemma *outer-prod-le-one*:
assumes $v \in$ *carrier-vec n*
and *inner-prod v v* $\leq$ *1*
shows *outer-prod v v* $\leq_L$ 1_m n
⟨*proof*⟩

lemma *zero-lowner-le-positiveD*:

fixes A :: *complex mat*
assumes dA: $A \in$ *carrier-mat* n n **and** *le*: 0_m n $n \leq_L A$
shows *positive* A
⟨*proof*⟩

lemma *zero-lowner-le-positiveI*:
fixes A :: *complex mat*
assumes dA: $A \in$ *carrier-mat* n n **and** *le*: *positive* A
shows 0_m n $n \leq_L A$
⟨*proof*⟩

lemma *lowner-le-trans-positiveI*:
fixes A B :: *complex mat*
assumes dA: $A \in$ *carrier-mat* n n **and** pA: *positive* A **and** *le*: $A \leq_L B$
shows *positive* B
⟨*proof*⟩

lemma *lowner-le-keep-under-measurement*:
fixes M A B :: *complex mat*
assumes dM: $M \in$ *carrier-mat* n n **and** dA: $A \in$ *carrier-mat* n n **and** dB: $B \in$ *carrier-mat* n n
and *le*: $A \leq_L B$
shows *adjoint* $M * A * M \leq_L$ *adjoint* $M * B * M$
⟨*proof*⟩

lemma *smult-distrib-left-minus-mat*:
fixes A B :: *'a::comm-ring-1 mat*
assumes $A \in$ *carrier-mat* n n $B \in$ *carrier-mat* n n
shows $c \cdot_m (B - A) = c \cdot_m B - c \cdot_m A$
⟨*proof*⟩

lemma *lowner-le-smultc*:
fixes c :: *complex*
assumes $c \geq 0$ $A \leq_L B$ $A \in$ *carrier-mat* n n $B \in$ *carrier-mat* n n
shows $c \cdot_m A \leq_L c \cdot_m B$
⟨*proof*⟩

lemma *lowner-le-smult*:
fixes c :: *real*
assumes $c \geq 0$ $A \leq_L B$ $A \in$ *carrier-mat* n n $B \in$ *carrier-mat* n n
shows $c \cdot_m A \leq_L c \cdot_m B$
⟨*proof*⟩

lemma *minus-smult-vec-distrib*:
fixes w :: *'a::comm-ring-1 vec*
shows $(a - b) \cdot_v w = a \cdot_v w - b \cdot_v w$
⟨*proof*⟩

lemma *smult-mat-mult-mat-vec-assoc*:

fixes A :: $'a$::*comm-ring-1 mat*
assumes A: $A \in$ *carrier-mat* n m **and** w: $w \in$ *carrier-vec* m
shows $a \cdot_m A *_v w = a \cdot_v (A *_v w)$
⟨*proof*⟩

lemma *mult-mat-vec-smult-vec-assoc*:
fixes A :: $'a$::*comm-ring-1 mat*
assumes A: $A \in$ *carrier-mat* n m **and** w: $w \in$ *carrier-vec* m
shows $A *_v (a \cdot_v w) = a \cdot_v (A *_v w)$
⟨*proof*⟩

lemma *outer-prod-left-right-mat*:
fixes A B :: *complex mat*
assumes *du*: $u \in$ *carrier-vec d2* **and** *dv*: $v \in$ *carrier-vec d3*
and *dA*: $A \in$ *carrier-mat d1 d2* **and** *dB*: $B \in$ *carrier-mat d3 d4*
shows $A * ($*outer-prod* u $v) * B = ($*outer-prod* $(A *_v u)$ (*adjoint* $B *_v v$))
⟨*proof*⟩

1.14 Density operators

definition *density-operator* :: *complex mat* ⇒ *bool* **where**
density-operator $A \longleftrightarrow$ *positive* $A \wedge$ *trace* $A = 1$

definition *partial-density-operator* :: *complex mat* ⇒ *bool* **where**
partial-density-operator $A \longleftrightarrow$ *positive* $A \wedge$ *trace* $A \leq 1$

lemma *pure-state-self-outer-prod-is-partial-density-operator*:
fixes v :: *complex vec*
assumes *dimv*: $v \in$ *carrier-vec* n **and** *nv*: *vec-norm* $v = 1$
shows *partial-density-operator* (*outer-prod* v v)
⟨*proof*⟩

lemma *lowner-le-trace*:
assumes A: $A \in$ *carrier-mat* n n
and B: $B \in$ *carrier-mat* n n
shows $A \leq_L B \longleftrightarrow (\forall \varrho \in$ *carrier-mat* n n. *partial-density-operator* $\varrho \longrightarrow$ *trace* $(A * \varrho) \leq$ *trace* $(B * \varrho))$
⟨*proof*⟩

lemma *lowner-le-traceI*:
assumes $A \in$ *carrier-mat* n n
and $B \in$ *carrier-mat* n n
and $\bigwedge \varrho$. $\varrho \in$ *carrier-mat* n $n \Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *trace* $(A * \varrho) \leq$ *trace* $(B * \varrho)$
shows $A \leq_L B$
⟨*proof*⟩

lemma *trace-pdo-eq-imp-eq*:

assumes *A*: $A \in$ *carrier-mat n n*
and *B*: $B \in$ *carrier-mat n n*
and *teq*: $\bigwedge\varrho$. $\varrho \in$ *carrier-mat n n* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *trace* $(A * \varrho) =$ *trace* $(B * \varrho)$
shows $A = B$
⟨*proof*⟩

lemma *lowner-le-traceD*:
assumes $A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n* $\varrho \in$ *carrier-mat n n*
and $A \leq_L B$
and *partial-density-operator* ϱ
shows *trace* $(A * \varrho) \leq$ *trace* $(B * \varrho)$
⟨*proof*⟩

lemma *sum-only-one-neq-0*:
assumes *finite A* **and** $j \in A$ **and** $\bigwedge i.\ i \in A \Longrightarrow i \neq j \Longrightarrow g\ i = 0$
shows *sum g A* $= g\ j$
⟨*proof*⟩

end

2 Matrix limits

theory *Matrix-Limit*
imports *Complex-Matrix*
begin

2.1 Definition of limit of matrices

definition *limit-mat* :: (*nat* ⇒ *complex mat*) ⇒ *complex mat* ⇒ *nat* ⇒ *bool* **where**
limit-mat X A m $\longleftrightarrow$ ($\forall$ *n*. *X n* $\in$ *carrier-mat m m* $\wedge$ *A* $\in$ *carrier-mat m m* $\wedge$
($\forall\ i < m.\ \forall\ j < m.\ (\lambda\ n.\ (X\ n)$ \$\$ $(i, j)) \longrightarrow (A$ \$\$ $(i, j)))))$

lemma *limit-mat-unique*:
assumes *limA*: *limit-mat X A m* **and** *limB*: *limit-mat X B m*
shows $A = B$
⟨*proof*⟩

lemma *limit-mat-const*:
fixes *A* :: *complex mat*
assumes $A \in$ *carrier-mat m m*
shows *limit-mat* $(\lambda k.\ A)$ *A m*
⟨*proof*⟩

lemma *limit-mat-scale*:
fixes *X* :: *nat* ⇒ *complex mat* **and** *A* :: *complex mat*
assumes *limX*: *limit-mat X A m*
shows *limit-mat* $(\lambda n.\ c \cdot_m X\ n)$ $(c \cdot_m A)$ *m*
⟨*proof*⟩

lemma *limit-mat-add*:
fixes X :: *nat* ⇒ *complex mat* **and** Y :: *nat* ⇒ *complex mat* **and** A :: *complex mat*
and m :: *nat* **and** B :: *complex mat*
assumes *limX*: *limit-mat X A m* **and** *limY*: *limit-mat Y B m*
shows *limit-mat* $(\lambda k.\ X\ k + Y\ k)\ (A + B)\ m$
⟨*proof*⟩

lemma *limit-mat-minus*:
fixes X :: *nat* ⇒ *complex mat* **and** Y :: *nat* ⇒ *complex mat* **and** A :: *complex mat*
and m :: *nat* **and** B :: *complex mat*
assumes *limX*: *limit-mat X A m* **and** *limY*: *limit-mat Y B m*
shows *limit-mat* $(\lambda k.\ X\ k - Y\ k)\ (A - B)\ m$
⟨*proof*⟩

lemma *limit-mat-mult*:
fixes X :: *nat* ⇒ *complex mat* **and** Y :: *nat* ⇒ *complex mat* **and** A :: *complex mat*
and m :: *nat* **and** B :: *complex mat*
assumes *limX*: *limit-mat X A m* **and** *limY*: *limit-mat Y B m*
shows *limit-mat* $(\lambda k.\ X\ k * Y\ k)\ (A * B)\ m$
⟨*proof*⟩

Adding matrix A to the sequence X

definition *mat-add-seq* :: *complex mat* ⇒ (*nat* ⇒ *complex mat*) ⇒ *nat* ⇒ *complex mat* **where**
mat-add-seq $A\ X = (\lambda n.\ A + X\ n)$

lemma *mat-add-limit*:
fixes X :: *nat* ⇒ *complex mat* **and** A :: *complex mat* **and** m :: *nat* **and** B :: *complex mat*
assumes *dimB*: $B \in$ *carrier-mat m m* **and** *limX*: *limit-mat X A m*
shows *limit-mat* (*mat-add-seq* B X) $(B + A)\ m$
⟨*proof*⟩

lemma *mat-minus-limit*:
fixes X :: *nat* ⇒ *complex mat* **and** A :: *complex mat* **and** m :: *nat* **and** B :: *complex mat*
assumes *dimB*: $B \in$ *carrier-mat m m* **and** *limX*: *limit-mat X A m*
shows *limit-mat* $(\lambda n.\ B - X\ n)\ (B - A)\ m$
⟨*proof*⟩

Multiply matrix A by the sequence X

definition *mat-mult-seq* :: *complex mat* ⇒ (*nat* ⇒ *complex mat*) ⇒ *nat* ⇒ *complex mat* **where**
mat-mult-seq $A\ X = (\lambda n.\ A * X\ n)$

lemma *mat-mult-limit*:
  fixes X :: *nat* ⇒ *complex mat* **and** A B :: *complex mat* **and** m :: *nat*
  assumes *dimB*: $B \in$ *carrier-mat* m m **and** *limX*: *limit-mat* X A m
  shows *limit-mat* (*mat-mult-seq* B X) ($B * A$) m
  ⟨*proof*⟩

lemma *mult-mat-limit*:
  fixes X :: *nat* ⇒ *complex mat* **and** A B :: *complex mat* **and** m :: *nat*
  assumes *dimB*: $B \in$ *carrier-mat* m m **and** *limX*: *limit-mat* X A m
  shows *limit-mat* ($\lambda k.\ X\ k * B$) ($A * B$) m
  ⟨*proof*⟩

lemma *quadratic-form-mat*:
  fixes A :: *complex mat* **and** v :: *complex vec* **and** m :: *nat*
  assumes *dimv*: *dim-vec* $v = m$ **and** *dimA*: $A \in$ *carrier-mat* m m
  shows *inner-prod* v ($A *_v v$) = ($\sum$ *i=0..<m*. ($\sum$ *j=0..<m*. *conjugate* ($v\$i$) $*$ $A\$\$(i, j) * v\$j$))
⟨*proof*⟩

lemma *sum-subtractff*:
  fixes h g :: *nat* ⇒ *nat* ⇒'*a*::*ab-group-add*
  shows ($\sum x\in A.\ \sum y\in B.\ h\ x\ y - g\ x\ y$) = ($\sum x\in A.\ \sum y\in B.\ h\ x\ y$) − ($\sum x\in A.\ \sum y\in B.\ g\ x\ y$)
⟨*proof*⟩

lemma *sum-abs-complex*:
  fixes h :: *nat* ⇒ *nat* ⇒ *complex*
  shows *cmod* ($\sum x\in A.\sum y\in B.\ h\ x\ y$) ≤ ($\sum x\in A.\ \sum y\in B.$ *cmod*($h\ x\ y$))
⟨*proof*⟩

lemma *hermitian-mat-lim-is-hermitian*:
  fixes X :: *nat* ⇒ *complex mat* **and** A :: *complex mat* **and** m :: *nat*
  assumes *limX*: *limit-mat* X A m **and** *herX*: ∀ n. *hermitian* (X n)
  shows *hermitian* A
⟨*proof*⟩

lemma *quantifier-change-order-once*:
  fixes P :: *nat* ⇒ *nat* ⇒ *bool* **and** m :: *nat*
  shows $\forall j<m.\ \exists no.\ \forall n\geq no.\ P\ n\ j \Longrightarrow \exists no.\ \forall j<m.\ \forall n\geq no.\ P\ n\ j$
⟨*proof*⟩

lemma *quantifier-change-order-twice*:
  fixes P :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *bool* **and** m n :: *nat*
  shows $\forall i<m.\ \forall j<n.\ \exists\ no.\ \forall n\geq no.\ P\ n\ i\ j \Longrightarrow \exists no.\ \forall i<m.\ \forall j<n.\ \forall n\geq no.\ P\ n\ i\ j$
⟨*proof*⟩

lemma *pos-mat-lim-is-pos*:
  fixes X :: *nat* ⇒ *complex mat* **and** A :: *complex mat* **and** m :: *nat*

assumes *limX*: *limit-mat X A m* **and** *posX*: $\forall$ *n. positive* (*X n*)
shows *positive A*
⟨*proof*⟩

lemma *limit-mat-ignore-initial-segment*:
limit-mat g A d $\Longrightarrow$ *limit-mat* (λ*n. g* (*n* + *k*)) *A d*
⟨*proof*⟩

lemma *mat-trace-limit*:
limit-mat g A d $\Longrightarrow$ (λ*n. trace* (*g n*)) $\longlongrightarrow$ *trace A*
⟨*proof*⟩

2.2 Existence of least upper bound for the Löwner order

definition *lowner-is-lub* :: (*nat* $\Rightarrow$ *complex mat*) $\Rightarrow$ *complex mat* $\Rightarrow$ *bool* **where**
lowner-is-lub f M $\longleftrightarrow$ ($\forall$ *n. f n* $\le_L$ *M*) $\wedge$ ($\forall$ *M'*. ($\forall$ *n. f n* $\le_L$ *M'*) $\longrightarrow$ *M* $\le_L$ *M'*)

locale *matrix-seq* =
fixes *dim* :: *nat*
and *f* :: *nat* $\Rightarrow$ *complex mat*
assumes
dim: $\bigwedge$*n. f n* $\in$ *carrier-mat dim dim* **and**
pdo: $\bigwedge$*n. partial-density-operator* (*f n*) **and**
inc: $\bigwedge$*n. lowner-le* (*f n*) (*f* (*Suc n*))
begin

definition *lowner-is-lub* :: *complex mat* $\Rightarrow$ *bool* **where**
lowner-is-lub M $\longleftrightarrow$ ($\forall$ *n. f n* $\le_L$ *M*) $\wedge$ ($\forall$ *M'*. ($\forall$ *n. f n* $\le_L$ *M'*) $\longrightarrow$ *M* $\le_L$ *M'*)

lemma *lowner-is-lub-dim*:
assumes *lowner-is-lub M*
shows *M* $\in$ *carrier-mat dim dim*
⟨*proof*⟩

lemma *trace-adjoint-eq-u*:
fixes *A* :: *complex mat*
shows *trace* (*A* $*$ *adjoint A*) = ($\sum$ *i* $\in$ {*0* ..< *dim-row A*}. $\sum$ *j* $\in$ {*0* ..< *dim-col A*}. (*norm*(*A* $$ (*i*,*j*)))^2)
⟨*proof*⟩

lemma *trace-adjoint-element-ineq*:
fixes *A* :: *complex mat*
assumes *rindex*: *i* $\in$ {*0* ..< *dim-row A*}
and *cindex*: *j* $\in$ {*0* ..< *dim-col A*}
shows (*norm*(*A* $$ (*i*,*j*)))^2 $\le$ *trace* (*A* $*$ *adjoint A*)
⟨*proof*⟩

lemma *positive-is-normal*:

fixes *A* :: *complex mat*
assumes *pos*: *positive A*
shows *A* ∗ *adjoint A* = *adjoint A* ∗ *A*
⟨*proof*⟩

lemma *diag-mat-mul-diag-diag*:
fixes *A B* :: *complex mat*
assumes *dimA*: *A* ∈ *carrier-mat n n* **and** *dimB*: *B* ∈ *carrier-mat n n*
and *dA*: *diagonal-mat A* **and** *dB*: *diagonal-mat B*
shows *diagonal-mat* (*A* ∗ *B*)
⟨*proof*⟩

lemma *diag-mat-mul-diag-ele*:
fixes *A B* :: *complex mat*
assumes *dimA*: *A* ∈ *carrier-mat n n* **and** *dimB*: *B* ∈ *carrier-mat n n*
and *dA*: *diagonal-mat A* **and** *dB*: *diagonal-mat B*
shows ∀ *i*<*n*. (*A*∗*B*) $$ (*i*,*i*) = *A*$$(*i*, *i*) ∗ *B*$$(*i*, *i*)
⟨*proof*⟩

lemma *trace-square-less-square-trace*:
fixes *B* :: *complex mat*
assumes *dimB*: *B* ∈ *carrier-mat n n*
and *dB*: *diagonal-mat B* **and** *pB*: ⋀*i*. *i* < *n* ⟹ *B*$$(*i*, *i*) ≥ *0*
shows *trace* (*B*∗*B*) ≤ (*trace B*)^2
⟨*proof*⟩

lemma *trace-positive-eq*:
fixes *A* :: *complex mat*
assumes *pos*: *positive A*
shows *trace* (*A* ∗ *adjoint A*) ≤ (*trace A*)^2
⟨*proof*⟩

lemma *lowner-le-transitive*:
fixes *m n* :: *nat*
assumes *re*: *n* ≥ *m*
shows *positive* (*f n* − *f m*)
⟨*proof*⟩

The sequence of matrices converges pointwise.

lemma *inc-partial-density-operator-converge*:
assumes *i*: *i* ∈ {*0* ..<*dim*} **and** *j*: *j* ∈ {*0* ..<*dim*}
shows *convergent* (λ*n*. *f n* $$ (*i*, *j*))
⟨*proof*⟩

definition *mat-seq-minus* :: (*nat* ⇒ *complex mat*) ⇒ *complex mat* ⇒ *nat* ⇒ *complex mat* **where**
mat-seq-minus X A = (λ*n*. *X n* − *A*)

definition *minus-mat-seq* :: *complex mat* ⇒ (*nat* ⇒ *complex mat*) ⇒ *nat* ⇒ *complex mat* **where**
minus-mat-seq A X = (λ*n*. *A* − *X n*)

lemma *pos-mat-lim-is-pos-aux*:
fixes *X* :: *nat* ⇒ *complex mat* **and** *A* :: *complex mat* **and** *m* :: *nat*
assumes *limX*: *limit-mat X A m* **and** *posX*: ∃ *k*. ∀ *n*≥*k*. *positive* (*X n*)
shows *positive A*
⟨*proof*⟩

lemma *minus-mat-limit*:
fixes *X* :: *nat* ⇒ *complex mat* **and** *A* :: *complex mat* **and** *m* :: *nat* **and** *B* :: *complex mat*
assumes *dimB*: *B* ∈ *carrier-mat m m* **and** *limX*: *limit-mat X A m*
shows *limit-mat* (*mat-seq-minus X B*) (*A* − *B*) *m*
⟨*proof*⟩

lemma *mat-minus-limit*:
fixes *X* :: *nat* ⇒ *complex mat* **and** *A* :: *complex mat* **and** *m* :: *nat* **and** *B* :: *complex mat*
assumes *dimA*: *A* ∈ *carrier-mat m m* **and** *limX*: *limit-mat X A m*
shows *limit-mat* (*minus-mat-seq B X*) (*B* − *A*) *m*
⟨*proof*⟩

lemma *lowner-lub-form*:
lowner-is-lub (*mat dim dim* (λ (*i*, *j*). (*lim* (λ *n*. (*f n*) $$ (*i*, *j*)))))
⟨*proof*⟩

Lowner partial order is a complete partial order.

lemma *lowner-lub-exists*: ∃ *M*. *lowner-is-lub M*
⟨*proof*⟩

lemma *lowner-lub-unique*: ∃!*M*. *lowner-is-lub M*
⟨*proof*⟩

definition *lowner-lub* :: *complex mat* **where**
lowner-lub = (*THE M*. *lowner-is-lub M*)

lemma *lowner-lub-prop*: *lowner-is-lub lowner-lub*
⟨*proof*⟩

lemma *lowner-lub-is-limit*:
limit-mat f lowner-lub dim
⟨*proof*⟩

lemma *lowner-lub-trace*:
assumes ∀ *n*. *trace* (*f n*) ≤ *x*
shows *trace lowner-lub* ≤ *x*
⟨*proof*⟩

lemma *lowner-lub-is-positive*:
shows *positive lowner-lub*
⟨*proof*⟩

end

2.3 Finite sum of matrices

Add f in the interval [0, n)

fun *matrix-sum* :: *nat* ⇒ (*nat* ⇒ ′*b*::*semiring-1 mat*) ⇒ *nat* ⇒ ′*b mat* **where**
matrix-sum d f 0 = 0_m *d d*
| *matrix-sum d f* (*Suc n*) = *f n* + *matrix-sum d f n*

definition *matrix-inf-sum* :: *nat* ⇒ (*nat* ⇒ *complex mat*) ⇒ *complex mat* **where**
matrix-inf-sum d f = *matrix-seq.lowner-lub* (λ*n*. *matrix-sum d f n*)

lemma *matrix-sum-dim*:
fixes *f* :: *nat* ⇒ ′*b*::*semiring-1 mat*
shows (⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d d*) ⟹ *matrix-sum d f n* ∈ *carrier-mat d d*
⟨*proof*⟩

lemma *matrix-sum-cong*:
fixes *f* :: *nat* ⇒ ′*b*::*semiring-1 mat*
shows (⋀*k*. *k* < *n* ⟹ *f k* = *f*′ *k*) ⟹ *matrix-sum d f n* = *matrix-sum d f*′ *n*
⟨*proof*⟩

lemma *matrix-sum-add*:
fixes *f* :: *nat* ⇒ ′*b*::*semiring-1 mat* **and** *g* :: *nat* ⇒ ′*b*::*semiring-1 mat* **and** *h* :: *nat* ⇒ ′*b*::*semiring-1 mat*
shows (⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d d*) ⟹ (⋀*k*. *k* < *n* ⟹ *g k* ∈ *carrier-mat d d*) ⟹ (⋀*k*. *k* < *n* ⟹ *h k* ∈ *carrier-mat d d*) ⟹
(⋀*k*. *k* < *n* ⟹ *f k* = *g k* + *h k*) ⟹ *matrix-sum d f n* = *matrix-sum d g n* + *matrix-sum d h n*
⟨*proof*⟩

lemma *matrix-sum-smult*:
fixes *f* :: *nat* ⇒ ′*b*::*semiring-1 mat*
shows (⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d d*) ⟹
matrix-sum d (λ *k*. *c* $\cdot_m$ *f k*) *n* = *c* $\cdot_m$ *matrix-sum d f n*
⟨*proof*⟩

lemma *matrix-sum-remove*:
fixes *f* :: *nat* ⇒ ′*b*::*semiring-1 mat*
assumes *j*: *j* < *n*
and *df*: (⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d d*)
and *f*′: (⋀*k*. *f*′ *k* = (*if k* = *j then* 0_m *d d else f k*))
shows *matrix-sum d f n* = *f j* + *matrix-sum d f*′ *n*

⟨*proof*⟩

lemma *matrix-sum-Suc-remove-head*:
fixes f :: *nat* ⇒ *complex mat*
shows $(\bigwedge k.\ k < n + 1 \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow$
$matrix\text{-}sum\ d\ f\ (n + 1) = f\ 0 + matrix\text{-}sum\ d\ (\lambda k.\ f\ (k + 1))\ n$
⟨*proof*⟩

lemma *matrix-sum-positive*:
fixes f :: *nat* ⇒ *complex mat*
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow (\bigwedge k.\ k < n \Longrightarrow positive\ (f\ k))$
$\Longrightarrow positive\ (matrix\text{-}sum\ d\ f\ n)$
⟨*proof*⟩

lemma *matrix-sum-mult-right*:
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow A \in carrier\text{-}mat\ d\ d$
$\Longrightarrow matrix\text{-}sum\ d\ (\lambda k.\ (f\ k) * A)\ n = matrix\text{-}sum\ d\ (\lambda k.\ f\ k)\ n * A$
⟨*proof*⟩

lemma *matrix-sum-add-distrib*:
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow (\bigwedge k.\ k < n \Longrightarrow g\ k \in carrier\text{-}mat\ d\ d)$
$\Longrightarrow matrix\text{-}sum\ d\ (\lambda k.\ (f\ k) + (g\ k))\ n = matrix\text{-}sum\ d\ f\ n + matrix\text{-}sum\ d\ g\ n$
⟨*proof*⟩

lemma *matrix-sum-minus-distrib*:
fixes $f\ g$ :: *nat* ⇒ *complex mat*
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow (\bigwedge k.\ k < n \Longrightarrow g\ k \in carrier\text{-}mat\ d\ d)$
$\Longrightarrow matrix\text{-}sum\ d\ (\lambda k.\ (f\ k) - (g\ k))\ n = matrix\text{-}sum\ d\ f\ n - matrix\text{-}sum\ d\ g\ n$
⟨*proof*⟩

lemma *matrix-sum-shift-Suc*:
shows $(\bigwedge k.\ k < (Suc\ n) \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d)$
$\Longrightarrow matrix\text{-}sum\ d\ f\ (Suc\ n) = f\ 0 + matrix\text{-}sum\ d\ (\lambda k.\ f\ (Suc\ k))\ n$
⟨*proof*⟩

lemma *lowner-le-matrix-sum*:
fixes $f\ g$ :: *nat* ⇒ *complex mat*
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow (\bigwedge k.\ k < n \Longrightarrow g\ k \in carrier\text{-}mat\ d\ d)$
$\Longrightarrow (\bigwedge k.\ k < n \Longrightarrow f\ k \le_L g\ k)$
$\Longrightarrow matrix\text{-}sum\ d\ f\ n \le_L matrix\text{-}sum\ d\ g\ n$
⟨*proof*⟩

lemma *lowner-lub-add*:
assumes $matrix\text{-}seq\ d\ f\ matrix\text{-}seq\ d\ g\ \forall\ n.\ trace\ (f\ n + g\ n) \le 1$
shows $matrix\text{-}seq.lowner\text{-}lub\ (\lambda n.\ f\ n + g\ n) = matrix\text{-}seq.lowner\text{-}lub\ f + matrix\text{-}seq.lowner\text{-}lub\ g$

⟨*proof*⟩

lemma *lowner-lub-scale*:
fixes *c* :: *real*
assumes *matrix-seq d f* $\forall$ *n. trace* $(c \cdot_m f\ n) \leq 1$ $c \geq 0$
shows *matrix-seq.lowner-lub* $(\lambda n.\ c \cdot_m f\ n) = c \cdot_m$ *matrix-seq.lowner-lub f*
⟨*proof*⟩

lemma *trace-matrix-sum-linear*:
fixes *f* :: *nat* ⇒ *complex mat*
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in$ *carrier-mat d d*$) \Longrightarrow$ *trace* (*matrix-sum d f n*) = *sum* (λk. *trace* (*f k*)) {*0..<n*}
⟨*proof*⟩

lemma *matrix-sum-distrib-left*:
fixes *f* :: *nat* ⇒ *complex mat*
shows $P \in$ *carrier-mat d d* $\Longrightarrow (\bigwedge k.\ k < n \Longrightarrow f\ k \in$ *carrier-mat d d*$) \Longrightarrow$ *matrix-sum d* (λk. $P * (f\ k)$) $n = P *$ (*matrix-sum d f n*)
⟨*proof*⟩

2.4 Measurement

definition *measurement* :: *nat* ⇒ *nat* ⇒ (*nat* ⇒ *complex mat*) ⇒ *bool* **where**
measurement d n M ⟷ ($\forall j < n$. *M j* $\in$ *carrier-mat d d*)
$\wedge$ *matrix-sum d* (λj. (*adjoint* (*M j*)) $*$ *M j*) $n = 1_m\ d$

lemma *measurement-dim*:
assumes *measurement d n M*
shows $\bigwedge k.\ k < n \Longrightarrow (M\ k) \in$ *carrier-mat d d*
⟨*proof*⟩

lemma *measurement-id2*:
assumes *measurement d 2 M*
shows *adjoint* (*M 0*) $*$ *M 0* + *adjoint* (*M 1*) $*$ *M 1* $= 1_m\ d$
⟨*proof*⟩

Result of measurement on ρ by matrix M

definition *measurement-res* :: *complex mat* ⇒ *complex mat* ⇒ *complex mat* **where**
measurement-res M $\varrho = M * \varrho *$ *adjoint M*

lemma *add-positive-le-reduce1*:
assumes *dA*: $A \in$ *carrier-mat n n* **and** *dB*: $B \in$ *carrier-mat n n* **and** *dC*: $C \in$ *carrier-mat n n*
and *pB*: *positive B* **and** *le*: $A + B \leq_L C$
shows $A \leq_L C$
⟨*proof*⟩

lemma *add-positive-le-reduce2*:
assumes *dA*: $A \in$ *carrier-mat n n* **and** *dB*: $B \in$ *carrier-mat n n* **and** *dC*: $C \in$ *carrier-mat n n*

and *pB*: *positive* B **and** *le*: $B + A \leq_L C$
shows $A \leq_L C$
⟨*proof*⟩

lemma *measurement-le-one-mat*:
assumes *measurement d n f*
shows $\bigwedge j.\ j < n \Longrightarrow$ *adjoint* $(f\ j) * f\ j \leq_L 1_m\ d$
⟨*proof*⟩

lemma *pdo-close-under-measurement*:
fixes $M\ \varrho$ :: *complex mat*
assumes *dM*: $M \in$ *carrier-mat n n* **and** *dr*: $\varrho \in$ *carrier-mat n n*
and *pdor*: *partial-density-operator* ϱ
and *le*: *adjoint* $M * M \leq_L 1_m\ n$
shows *partial-density-operator* $(M * \varrho *$ *adjoint* $M)$
⟨*proof*⟩

lemma *trace-measurement*:
assumes *m*: *measurement d n M* **and** *dA*: $A \in$ *carrier-mat d d*
shows *trace* (*matrix-sum d* ($\lambda k.\ (M\ k) * A *$ *adjoint* $(M\ k)$) *n*) = *trace A*
⟨*proof*⟩

lemma *mat-inc-seq-positive-transform*:
assumes *dfn*: $\bigwedge n.\ f\ n \in$ *carrier-mat d d*
and *inc*: $\bigwedge n.\ f\ n \leq_L f\ (Suc\ n)$
shows $\bigwedge n.\ f\ n - f\ 0 \in$ *carrier-mat d d* **and** $\bigwedge n.\ (f\ n - f\ 0) \leq_L (f\ (Suc\ n) - f\ 0)$
⟨*proof*⟩

lemma *mat-inc-seq-lub*:
assumes *dfn*: $\bigwedge n.\ f\ n \in$ *carrier-mat d d*
and *inc*: $\bigwedge n.\ f\ n \leq_L f\ (Suc\ n)$
and *ub*: $\bigwedge n.\ f\ n \leq_L A$
shows $\exists B.$ *lowner-is-lub f B* $\wedge$ *limit-mat f B d*
⟨*proof*⟩

end

3 Quantum programs

theory *Quantum-Program*
imports *Matrix-Limit*
begin

3.1 Syntax

Datatype for quantum programs

datatype *com* =

SKIP
| *Utrans complex mat*
| *Seq com com* (‹-;;/ -› [*60*, *61*] *60*)
| *Measure nat nat* ⇒ *complex mat com list*
| *While nat* ⇒ *complex mat com*

A state corresponds to the density operator

type-synonym *state* = *complex mat*

List of dimensions of quantum states

locale *state-sig* =
fixes *dims* :: *nat list*
begin

definition *d* :: *nat* **where**
d = *prod-list dims*

Wellformedness of commands

fun *well-com* :: *com* ⇒ *bool* **where**
well-com SKIP = *True*
| *well-com* (*Utrans U*) = (*U* ∈ *carrier-mat d d* ∧ *unitary U*)
| *well-com* (*Seq S1 S2*) = (*well-com S1* ∧ *well-com S2*)
| *well-com* (*Measure n M S*) =
(*measurement d n M* ∧ *length S* = *n* ∧ *list-all well-com S*)
| *well-com* (*While M S*) =
(*measurement d 2 M* ∧ *well-com S*)

3.2 Denotational semantics

Denotation of going through the while loop n times

fun *denote-while-n-iter* :: *complex mat* ⇒ *complex mat* ⇒ (*state* ⇒ *state*) ⇒ *nat* ⇒ *state* ⇒ *state* **where**
denote-while-n-iter M0 M1 DS 0 ϱ = ϱ
| *denote-while-n-iter M0 M1 DS* (*Suc n*) ϱ = *denote-while-n-iter M0 M1 DS n* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*))

fun *denote-while-n* :: *complex mat* ⇒ *complex mat* ⇒ (*state* ⇒ *state*) ⇒ *nat* ⇒ *state* ⇒ *state* **where**
denote-while-n M0 M1 DS n ϱ = *M0* ∗ *denote-while-n-iter M0 M1 DS n* ϱ ∗ *adjoint M0*

fun *denote-while-n-comp* :: *complex mat* ⇒ *complex mat* ⇒ (*state* ⇒ *state*) ⇒ *nat* ⇒ *state* ⇒ *state* **where**
denote-while-n-comp M0 M1 DS n ϱ = *M1* ∗ *denote-while-n-iter M0 M1 DS n* ϱ ∗ *adjoint M1*

lemma *denote-while-n-iter-assoc*:
denote-while-n-iter M0 M1 DS (*Suc n*) ϱ = *DS* (*M1* ∗ (*denote-while-n-iter M0 M1 DS n* ϱ) ∗ *adjoint M1*)

⟨*proof*⟩

lemma *denote-while-n-iter-dim*:
$\varrho \in$ *carrier-mat m m* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *M1* $\in$ *carrier-mat m m* $\Longrightarrow$ *adjoint M1* $*$ *M1* $\leq_L$ 1_m *m*
$\Longrightarrow$ ($\bigwedge\varrho$. $\varrho \in$ *carrier-mat m m* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *DS* $\varrho \in$ *carrier-mat m m* $\wedge$ *partial-density-operator* (*DS* ϱ))
$\Longrightarrow$ *denote-while-n-iter M0 M1 DS n* $\varrho \in$ *carrier-mat m m* $\wedge$ *partial-density-operator* (*denote-while-n-iter M0 M1 DS n* ϱ)
⟨*proof*⟩

lemma *pdo-denote-while-n-iter*:
$\varrho \in$ *carrier-mat m m* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *M1* $\in$ *carrier-mat m m* $\Longrightarrow$ *adjoint M1* $*$ *M1* $\leq_L$ 1_m *m*
$\Longrightarrow$ ($\bigwedge\varrho$. $\varrho \in$ *carrier-mat m m* $\wedge$ *partial-density-operator* $\varrho \Longrightarrow$ *partial-density-operator* (*DS* ϱ))
$\Longrightarrow$ ($\bigwedge\varrho$. $\varrho \in$ *carrier-mat m m* $\wedge$ *partial-density-operator* $\varrho \Longrightarrow$ *DS* $\varrho \in$ *carrier-mat m m*)
$\Longrightarrow$ *partial-density-operator* (*denote-while-n-iter M0 M1 DS n* ϱ)
⟨*proof*⟩

Denotation of while is simply the infinite sum of denote_while_n

definition *denote-while* :: *complex mat* $\Rightarrow$ *complex mat* $\Rightarrow$ (*state* $\Rightarrow$ *state*) $\Rightarrow$ *state* $\Rightarrow$ *state* **where**
denote-while M0 M1 DS ϱ = *matrix-inf-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ)

lemma *denote-while-n-dim*:
assumes $\varrho \in$ *carrier-mat d d*
M0 $\in$ *carrier-mat d d*
M1 $\in$ *carrier-mat d d*
partial-density-operator ϱ
$\bigwedge\varrho'$. $\varrho' \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho' \Longrightarrow$ *positive* (*DS* ϱ') $\wedge$ *trace* (*DS* ϱ') $\leq$ *trace* ϱ' $\wedge$ *DS* $\varrho' \in$ *carrier-mat d d*
shows *denote-while-n M0 M1 DS n* ϱ $\in$ *carrier-mat d d*
⟨*proof*⟩

lemma *denote-while-n-sum-dim*:
assumes $\varrho \in$ *carrier-mat d d*
M0 $\in$ *carrier-mat d d*
M1 $\in$ *carrier-mat d d*
partial-density-operator ϱ
$\bigwedge\varrho'$. $\varrho' \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho' \Longrightarrow$ *positive* (*DS* ϱ') $\wedge$ *trace* (*DS* ϱ') $\leq$ *trace* ϱ' $\wedge$ *DS* $\varrho' \in$ *carrier-mat d d*
shows *matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n* $\in$ *carrier-mat d d*
⟨*proof*⟩

lemma *trace-decrease-mul-adj*:
assumes *pdo*: *partial-density-operator* ϱ **and** *dimr*: $\varrho \in$ *carrier-mat d d*

and *dimx*: $x \in$ *carrier-mat d d* **and** *un*: *adjoint* $x * x \leq_L$ 1_m *d*
shows *trace* $(x * \varrho *$ *adjoint* $x) \leq$ *trace* ϱ
⟨*proof*⟩

lemma *denote-while-n-positive*:
assumes *dim0*: *M0* $\in$ *carrier-mat d d* **and** *dim1*: *M1* $\in$ *carrier-mat d d* **and** *un*: *adjoint M1* $*$ *M1* $\leq_L$ 1_m *d*
and *DS*: $\bigwedge\varrho$. $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *positive* (*DS* ϱ) $\wedge$ *trace* (*DS* ϱ) $\leq$ *trace* ϱ $\wedge$ *DS* $\varrho \in$ *carrier-mat d d*
shows *partial-density-operator* $\varrho \wedge \varrho \in$ *carrier-mat d d* $\Longrightarrow$ *positive* (*denote-while-n M0 M1 DS n* ϱ)
⟨*proof*⟩

lemma *denote-while-n-sum-positive*:
assumes *dim0*: *M0* $\in$ *carrier-mat d d* **and** *dim1*: *M1* $\in$ *carrier-mat d d* **and** *un*: *adjoint M1* $*$ *M1* $\leq_L$ 1_m *d*
and *DS*: $\bigwedge\varrho$. $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *positive* (*DS* ϱ) $\wedge$ *trace* (*DS* ϱ) $\leq$ *trace* ϱ $\wedge$ *DS* $\varrho \in$ *carrier-mat d d*
and *pdo*: *partial-density-operator* ϱ **and** *r*: $\varrho \in$ *carrier-mat d d*
shows *positive* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n*)
⟨*proof*⟩

lemma *trace-measure2-id*:
assumes *dM0*: *M0* $\in$ *carrier-mat n n* **and** *dM1*: *M1* $\in$ *carrier-mat n n*
and *id*: *adjoint M0* $*$ *M0* $+$ *adjoint M1* $*$ *M1* $=$ 1_m *n*
and *dA*: *A* $\in$ *carrier-mat n n*
shows *trace* (*M0* $*$ *A* $*$ *adjoint M0*) $+$ *trace* (*M1* $*$ *A* $*$ *adjoint M1*) $=$ *trace A*
⟨*proof*⟩

lemma *measurement-lowner-le-one1*:
assumes *dim0*: *M0* $\in$ *carrier-mat d d* **and** *dim1*: *M1* $\in$ *carrier-mat d d* **and** *id*: *adjoint M0* $*$ *M0* $+$ *adjoint M1* $*$ *M1* $=$ 1_m *d*
shows *adjoint M1* $*$ *M1* $\leq_L$ 1_m *d*
⟨*proof*⟩

lemma *denote-while-n-sum-trace*:
assumes *dim0*: *M0* $\in$ *carrier-mat d d* **and** *dim1*: *M1* $\in$ *carrier-mat d d* **and** *id*: *adjoint M0* $*$ *M0* $+$ *adjoint M1* $*$ *M1* $=$ 1_m *d*
and *DS*: $\bigwedge\varrho$. $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *positive* (*DS* ϱ) $\wedge$ *trace* (*DS* ϱ) $\leq$ *trace* ϱ $\wedge$ *DS* $\varrho \in$ *carrier-mat d d*
and *r*: $\varrho \in$ *carrier-mat d d*
and *pdor*: *partial-density-operator* ϱ
shows *trace* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n*) $\leq$ *trace* ϱ
⟨*proof*⟩

lemma *denote-while-n-sum-partial-density*:
assumes *dim0*: *M0* $\in$ *carrier-mat d d* **and** *dim1*: *M1* $\in$ *carrier-mat d d* **and** *id*: *adjoint M0* $*$ *M0* $+$ *adjoint M1* $*$ *M1* $=$ 1_m *d*
and *DS*: $\bigwedge\varrho$. $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *positive*

$(DS\ \varrho) \wedge trace\ (DS\ \varrho) \leq trace\ \varrho \wedge DS\ \varrho \in carrier\text{-}mat\ d\ d$
and *pdo*: $partial\text{-}density\text{-}operator\ \varrho$ **and** *r*: $\varrho \in carrier\text{-}mat\ d\ d$
shows $(partial\text{-}density\text{-}operator\ (matrix\text{-}sum\ d\ (\lambda n.\ denote\text{-}while\text{-}n\ M0\ M1\ DS\ n\ \varrho)\ n))$
⟨*proof*⟩

lemma *denote-while-n-sum-lowner-le*:
assumes *dim0*: $M0 \in carrier\text{-}mat\ d\ d$ **and** *dim1*: $M1 \in carrier\text{-}mat\ d\ d$ **and** *id*: $adjoint\ M0 * M0 + adjoint\ M1 * M1 = 1_m\ d$
and *DS*: $\bigwedge\varrho.\ \varrho \in carrier\text{-}mat\ d\ d \Longrightarrow partial\text{-}density\text{-}operator\ \varrho \Longrightarrow positive\ (DS\ \varrho) \wedge trace\ (DS\ \varrho) \leq trace\ \varrho \wedge DS\ \varrho \in carrier\text{-}mat\ d\ d$
and *pdo*: $partial\text{-}density\text{-}operator\ \varrho$ **and** *dimr*: $\varrho \in carrier\text{-}mat\ d\ d$
shows $(matrix\text{-}sum\ d\ (\lambda n.\ denote\text{-}while\text{-}n\ M0\ M1\ DS\ n\ \varrho)\ n \leq_L matrix\text{-}sum\ d\ (\lambda n.\ denote\text{-}while\text{-}n\ M0\ M1\ DS\ n\ \varrho)\ (Suc\ n))$
⟨*proof*⟩

lemma *lowner-is-lub-matrix-sum*:
assumes *dim0*: $M0 \in carrier\text{-}mat\ d\ d$ **and** *dim1*: $M1 \in carrier\text{-}mat\ d\ d$ **and** *id*: $adjoint\ M0 * M0 + adjoint\ M1 * M1 = 1_m\ d$
and *DS*: $\bigwedge\varrho.\ \varrho \in carrier\text{-}mat\ d\ d \Longrightarrow partial\text{-}density\text{-}operator\ \varrho \Longrightarrow positive\ (DS\ \varrho) \wedge trace\ (DS\ \varrho) \leq trace\ \varrho \wedge DS\ \varrho \in carrier\text{-}mat\ d\ d$
and *pdo*: $partial\text{-}density\text{-}operator\ \varrho$ **and** *dimr*: $\varrho \in carrier\text{-}mat\ d\ d$
shows $matrix\text{-}seq.lowner\text{-}is\text{-}lub\ (matrix\text{-}sum\ d\ (\lambda n.\ denote\text{-}while\text{-}n\ M0\ M1\ DS\ n\ \varrho))\ (matrix\text{-}seq.lowner\text{-}lub\ (matrix\text{-}sum\ d\ (\lambda n.\ denote\text{-}while\text{-}n\ M0\ M1\ DS\ n\ \varrho)))$
⟨*proof*⟩

lemma *denote-while-dim-positive*:
assumes *dim0*: $M0 \in carrier\text{-}mat\ d\ d$ **and** *dim1*: $M1 \in carrier\text{-}mat\ d\ d$ **and** *id*: $adjoint\ M0 * M0 + adjoint\ M1 * M1 = 1_m\ d$
and *DS*: $\bigwedge\varrho.\ \varrho \in carrier\text{-}mat\ d\ d \Longrightarrow partial\text{-}density\text{-}operator\ \varrho \Longrightarrow positive\ (DS\ \varrho) \wedge trace\ (DS\ \varrho) \leq trace\ \varrho \wedge DS\ \varrho \in carrier\text{-}mat\ d\ d$
and *pdo*: $partial\text{-}density\text{-}operator\ \varrho$ **and** *dimr*: $\varrho \in carrier\text{-}mat\ d\ d$
shows
$denote\text{-}while\ M0\ M1\ DS\ \varrho \in carrier\text{-}mat\ d\ d \wedge positive\ (denote\text{-}while\ M0\ M1\ DS\ \varrho) \wedge trace\ (denote\text{-}while\ M0\ M1\ DS\ \varrho) \leq trace\ \varrho$
⟨*proof*⟩

definition *denote-measure* :: $nat \Rightarrow (nat \Rightarrow complex\ mat) \Rightarrow ((state \Rightarrow state)\ list) \Rightarrow state \Rightarrow state$ **where**
$denote\text{-}measure\ n\ M\ DS\ \varrho = matrix\text{-}sum\ d\ (\lambda k.\ (DS!k)\ ((M\ k) * \varrho * adjoint\ (M\ k)))\ n$

lemma *denote-measure-dim*:
assumes $\varrho \in carrier\text{-}mat\ d\ d$
$measurement\ d\ n\ M$
$\bigwedge\varrho'\ k.\ \varrho' \in carrier\text{-}mat\ d\ d \Longrightarrow k < n \Longrightarrow (DS!k)\ \varrho' \in carrier\text{-}mat\ d\ d$
shows
$denote\text{-}measure\ n\ M\ DS\ \varrho \in carrier\text{-}mat\ d\ d$
⟨*proof*⟩

lemma *measure-well-com*:
  assumes *well-com* (*Measure n M S*)
  shows $\bigwedge k.\ k < n \Longrightarrow$ *well-com* (*S* ! *k*)
  ⟨*proof*⟩

Semantics of commands

fun *denote* :: *com* ⇒ *state* ⇒ *state* **where**
  denote SKIP $\varrho = \varrho$
| *denote* (*Utrans U*) $\varrho = U * \varrho *$ *adjoint U*
| *denote* (*Seq S1 S2*) ϱ = *denote S2* (*denote S1* ϱ)
| *denote* (*Measure n M S*) ϱ = *denote-measure n M* (*map denote S*) ϱ
| *denote* (*While M S*) ϱ = *denote-while* (*M 0*) (*M 1*) (*denote S*) ϱ

lemma *denote-measure-expand*:
  assumes *m*: $m \leq n$ **and** *wc*: *well-com* (*Measure n M S*)
  shows *denote* (*Measure m M S*) ϱ = *matrix-sum d* (λk. *denote* (*S*!*k*) ((*M k*) $* \varrho$ $*$ *adjoint* (*M k*))) *m*
  ⟨*proof*⟩

lemma *matrix-sum-trace-le*:
  fixes *f* :: *nat* ⇒ *complex mat* **and** *g* :: *nat* ⇒ *complex mat*
  assumes ($\bigwedge k.\ k < n \Longrightarrow f\ k \in$ *carrier-mat d d*)
    ($\bigwedge k.\ k < n \Longrightarrow g\ k \in$ *carrier-mat d d*)
    ($\bigwedge k.\ k < n \Longrightarrow$ *trace* (*f k*) $\leq$ *trace* (*g k*))
  shows *trace* (*matrix-sum d f n*) $\leq$ *trace* (*matrix-sum d g n*)
⟨*proof*⟩

lemma *map-denote-positive-trace-dim*:
  assumes *well-com* (*Measure x1 x2a x3a*)
    x4 $\in$ *carrier-mat d d*
    partial-density-operator x4
    $\bigwedge$*x3aa* ϱ. *x3aa* $\in$ *set x3a* $\Longrightarrow$ *well-com x3aa* $\Longrightarrow$ $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ
      $\Longrightarrow$ *positive* (*denote x3aa* ϱ) $\wedge$ *trace* (*denote x3aa* ϱ) $\leq$ *trace* ϱ $\wedge$ *denote x3aa* $\varrho \in$ *carrier-mat d d*
  shows $\forall k < x1$. *positive* ((*map denote x3a* ! *k*) (*x2a k* $*$ *x4* $*$ *adjoint* (*x2a k*)))
      $\wedge$ ((*map denote x3a* ! *k*) (*x2a k* $*$ *x4* $*$ *adjoint* (*x2a k*))) $\in$ *carrier-mat d d*
      $\wedge$ *trace* ((*map denote x3a* ! *k*) (*x2a k* $*$ *x4* $*$ *adjoint* (*x2a k*))) $\leq$ *trace* (*x2a k* $*$ *x4* $*$ *adjoint* (*x2a k*))
⟨*proof*⟩

lemma *denote-measure-positive-trace-dim*:
  assumes *well-com* (*Measure x1 x2a x3a*)
    x4 $\in$ *carrier-mat d d*
    partial-density-operator x4
    $\bigwedge$*x3aa* ϱ. *x3aa* $\in$ *set x3a* $\Longrightarrow$ *well-com x3aa* $\Longrightarrow$ $\varrho \in$ *carrier-mat d d* $\Longrightarrow$

partial-density-operator ϱ
$\Longrightarrow$ *positive* (*denote x3aa* ϱ) $\wedge$ *trace* (*denote x3aa* ϱ) $\leq$ *trace* ϱ $\wedge$ *denote x3aa* ϱ $\in$ *carrier-mat d d*
shows *positive* (*denote* (*Measure x1 x2a x3a*) *x4*) $\wedge$ *trace* (*denote* (*Measure x1 x2a x3a*) *x4*) $\leq$ *trace x4*
$\wedge$ (*denote* (*Measure x1 x2a x3a*) *x4*) $\in$ *carrier-mat d d*
⟨*proof*⟩

lemma *denote-positive-trace-dim*:
well-com S $\Longrightarrow$ ϱ $\in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ
$\Longrightarrow$ (*positive* (*denote S* ϱ) $\wedge$ *trace* (*denote S* ϱ) $\leq$ *trace* ϱ $\wedge$ *denote S* ϱ $\in$ *carrier-mat d d*)
⟨*proof*⟩

lemma *denote-dim-pdo*:
well-com S $\Longrightarrow$ ϱ $\in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ
$\Longrightarrow$ (*denote S* ϱ $\in$ *carrier-mat d d*) $\wedge$ (*partial-density-operator* (*denote S* ϱ))
⟨*proof*⟩

lemma *denote-dim*:
well-com S $\Longrightarrow$ ϱ $\in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ
$\Longrightarrow$ (*denote S* ϱ $\in$ *carrier-mat d d*)
⟨*proof*⟩

lemma *denote-trace*:
well-com S $\Longrightarrow$ ϱ $\in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ
$\Longrightarrow$ *trace* (*denote S* ϱ) $\leq$ *trace* ϱ
⟨*proof*⟩

lemma *denote-partial-density-operator*:
assumes *well-com S partial-density-operator* ϱ ϱ $\in$ *carrier-mat d d*
shows *partial-density-operator* (*denote S* ϱ)
⟨*proof*⟩

lemma *denote-while-n-sum-mat-seq*:
assumes ϱ $\in$ *carrier-mat d d* **and**
x1 0 $\in$ *carrier-mat d d* **and**
x1 1 $\in$ *carrier-mat d d* **and**
partial-density-operator ϱ **and**
wc: *well-com x2* **and** *mea*: *measurement d 2 x1*
shows *matrix-seq d* (*matrix-sum d* (λn. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ))
⟨*proof*⟩

lemma *denote-while-n-add*:
assumes *M0*: *x1 0* $\in$ *carrier-mat d d* **and**
M1: *x1 1* $\in$ *carrier-mat d d* **and**
wc: *well-com x2* **and** *mea*: *measurement d 2 x1* **and**

DS: ($\bigwedge \varrho_1\ \varrho_2.\ \varrho_1 \in$ *carrier-mat d d* $\Longrightarrow \varrho_2 \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho_1 \Longrightarrow$
partial-density-operator $\varrho_2 \Longrightarrow$ *trace* $(\varrho_1 + \varrho_2) \leq 1 \Longrightarrow$ *denote x2* $(\varrho_1 + \varrho_2)$ = *denote x2* ϱ_1 + *denote x2* ϱ_2)
shows $\varrho_1 \in$ *carrier-mat d d* $\Longrightarrow \varrho_2 \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho_1 \Longrightarrow$ *partial-density-operator* $\varrho_2 \Longrightarrow$ *trace* $(\varrho_1 + \varrho_2) \leq 1 \Longrightarrow$
denote-while-n (*x1 0*) (*x1 1*) (*denote x2*) *k* $(\varrho_1 + \varrho_2)$ = *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *k* ϱ_1 + *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *k* ϱ_2
⟨*proof*⟩

lemma *denote-while-add*:
assumes *r1*: $\varrho_1 \in$ *carrier-mat d d* **and**
r2: $\varrho_2 \in$ *carrier-mat d d* **and**
M0: *x1 0* $\in$ *carrier-mat d d* **and**
M1: *x1 1* $\in$ *carrier-mat d d* **and**
pdo1: *partial-density-operator* ϱ_1 **and**
pdo2: *partial-density-operator* ϱ_2 **and** *tr12*: *trace* $(\varrho_1 + \varrho_2) \leq 1$ **and**
wc: *well-com x2* **and** *mea*: *measurement d 2 x1* **and**
DS: ($\bigwedge \varrho_1\ \varrho_2.\ \varrho_1 \in$ *carrier-mat d d* $\Longrightarrow \varrho_2 \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho_1 \Longrightarrow$
partial-density-operator $\varrho_2 \Longrightarrow$ *trace* $(\varrho_1 + \varrho_2) \leq 1 \Longrightarrow$ *denote x2* $(\varrho_1 + \varrho_2)$ = *denote x2* ϱ_1 + *denote x2* ϱ_2)
shows
denote-while (*x1 0*) (*x1 1*) (*denote x2*) $(\varrho_1 + \varrho_2)$ = *denote-while* (*x1 0*) (*x1 1*) (*denote x2*) ϱ_1 + *denote-while* (*x1 0*) (*x1 1*) (*denote x2*) ϱ_2
⟨*proof*⟩

lemma *denote-add*:
well-com S $\Longrightarrow \varrho_1 \in$ *carrier-mat d d* $\Longrightarrow \varrho_2 \in$ *carrier-mat d d* $\Longrightarrow$
partial-density-operator $\varrho_1 \Longrightarrow$ *partial-density-operator* $\varrho_2 \Longrightarrow$ *trace* $(\varrho_1 + \varrho_2) \leq 1 \Longrightarrow$
denote S $(\varrho_1 + \varrho_2)$ = *denote S* ϱ_1 + *denote S* ϱ_2
⟨*proof*⟩

lemma *mulfact*:
fixes *c*:: *real* **and** *a*:: *complex* **and** *b*:: *complex*
assumes $c \geq 0$ $a \leq b$
shows $c * a \leq c * b$
⟨*proof*⟩

lemma *denote-while-n-scale*:
fixes *c*:: *real*
assumes $c \geq 0$
measurement d 2 x1 well-com x2
($\bigwedge \varrho.\ \varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *trace* $(c \cdot_m \varrho) \leq 1 \Longrightarrow$
denote x2 $(c \cdot_m \varrho) = c \cdot_m$ *denote x2* ϱ)
shows $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* $\varrho \Longrightarrow$ *trace* $(c \cdot_m \varrho) \leq$

1 $\Longrightarrow$
denote-while-n (*x1 0*) (*x1 1*) (*denote x2*) *n* (*complex-of-real c* $\cdot_m$ ϱ) = *c* $\cdot_m$ (*denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ)
⟨*proof*⟩

lemma *denote-while-scale*:
fixes *c*:: *real*
assumes $\varrho \in$ *carrier-mat d d*
partial-density-operator ϱ
trace (*c* $\cdot_m$ ϱ) $\leq$ *1* *c* $\geq$ *0*
measurement d 2 x1 well-com x2
($\bigwedge\varrho$. $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ $\Longrightarrow$ *trace* (*c* $\cdot_m$ ϱ) $\leq$ *1* $\Longrightarrow$
denote x2 (*c* $\cdot_m$ ϱ) = *c* $\cdot_m$ *denote x2* ϱ)
shows *denote-while* (*x1 0*) (*x1 1*) (*denote x2*) (*c* $\cdot_m$ ϱ) = *c* $\cdot_m$ *denote-while* (*x1 0*) (*x1 1*) (*denote x2*) ϱ
⟨*proof*⟩

lemma *denote-scale*:
fixes *c* :: *real*
assumes *c*$\geq$*0*
shows *well-com S* $\Longrightarrow$ $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ $\Longrightarrow$
trace (*c* $\cdot_m$ ϱ) $\leq$ *1* $\Longrightarrow$ *denote S* (*c* $\cdot_m$ ϱ) = *c* $\cdot_m$ *denote S* ϱ
⟨*proof*⟩

lemma *limit-mat-denote-while-n*:
assumes *wc*: *well-com* (*While M S*) **and** *dr*: $\varrho \in$ *carrier-mat d d* **and** *pdor*: *partial-density-operator* ϱ
shows *limit-mat* (*matrix-sum d* (λ*k*. *denote-while-n* (*M 0*) (*M 1*) (*denote S*) *k* ϱ)) (*denote* (*While M S*) ϱ) *d*
⟨*proof*⟩

end

end

4 Partial state

theory *Partial-State*
imports *Quantum-Program Deep-Learning.Tensor-Matricization*
begin

lemma *nths-intersection-eq*:
assumes {*0..*<*length xs*} $\subseteq$ *A*
shows *nths xs B* = *nths xs* (*A* $\cap$ *B*)
⟨*proof*⟩

lemma *nths-minus-eq*:
assumes {*0..*<*length xs*} $\subseteq$ *A*

shows *nths xs* $(-B)$ = *nths xs* $(A - B)$
⟨*proof*⟩

lemma *nths-split-complement-eq*:
assumes $A \cap B = \{\}$
and $\{0..<$*length xs*$\} \subseteq A \cup B$
shows *nths xs A* = *nths xs* $(-B)$
⟨*proof*⟩

lemma *lt-set-card-lt*:
fixes A :: *nat set*
assumes *finite A* **and** $x \in A$
shows *card* $\{y.\ y \in A \wedge y < x\}$ < *card A*
⟨*proof*⟩

definition *ind-in-set* **where**
ind-in-set A x = *card* $\{i.\ i \in A \wedge i < x\}$

lemma *bij-ind-in-set-bound*:
fixes M :: *nat* **and** *v0* :: *nat set*
assumes $\bigwedge x.\ f\ x$ = *card* $\{y.\ y \in v0 \wedge y < x\}$
shows *bij-betw f* $(\{0..<M\} \cap v0)$ $\{0..<$*card* $(\{0..<M\} \cap v0)\}$
⟨*proof*⟩

lemma *ind-in-set-bound*:
fixes A :: *nat set* **and** $M\ N$ :: *nat*
assumes $N \geq M$
shows *ind-in-set A N* $\notin$ (*ind-in-set A* ‘ $(\{0..<M\} \cap A)$)
⟨*proof*⟩

lemma *bij-minus-subset*:
bij-betw f A B $\Longrightarrow C \subseteq A \Longrightarrow (f\ ‘\ A) - (f\ ‘\ C) = f\ ‘\ (A - C)$
⟨*proof*⟩

lemma *ind-in-set-minus-subset-bound*:
fixes $A\ B$ :: *nat set* **and** M :: *nat*
assumes $B \subseteq A$
shows (*ind-in-set A* ‘ $(\{0..<M\} \cap A)$) − (*ind-in-set A* ‘ B) = (*ind-in-set A* ‘ $(\{0..<M\} \cap A)$) ∩ (*ind-in-set A* ‘ $(A - B)$)
⟨*proof*⟩

lemma *ind-in-set-iff*:
fixes $A\ B$ :: *nat set*
assumes $x \in A$ **and** $B \subseteq A$
shows *ind-in-set A x* ∈ (*ind-in-set A* ‘ B) = $(x \in B)$
⟨*proof*⟩

lemma *nths-reencode-eq*:

assumes $B \subseteq A$
shows *nths* (*nths xs A*) (*ind-in-set A* ‘ *B*) = *nths xs B*
⟨*proof*⟩

lemma *nths-reencode-eq-comp*:
assumes $B \subseteq A$
shows *nths* (*nths xs A*) (− *ind-in-set A* ‘ *B*) = *nths xs* (*A* − *B*)
⟨*proof*⟩

lemma *nths-prod-list-split*:
fixes *A* :: *nat set* **and** *xs* :: *nat list*
assumes $B \subseteq A$
shows *prod-list* (*nths xs A*) = (*prod-list* (*nths xs B*)) ∗ (*prod-list* (*nths xs* (*A* − *B*)))
⟨*proof*⟩

4.1 Encodings

lemma *digit-encode-take*:
take n (*digit-encode ds a*) = *digit-encode* (*take n ds*) *a*
⟨*proof*⟩

lemma *nths-minus-upt-eq-drop*:
nths l (−{..<*n*}) = *drop n l*
⟨*proof*⟩

lemma *digit-encode-drop*:
drop n (*digit-encode ds a*) = *digit-encode* (*drop n ds*) (*a div* (*prod-list* (*take n ds*)))
⟨*proof*⟩

List of active variables in the partial state

locale *partial-state* = *state-sig* +
fixes *vars* :: *nat set*

context *partial-state*
begin

Dimensions of active variables

abbreviation *avars* :: *nat set* **where**
avars ≡ {*0*..<*length dims*}

definition *dims1* :: *nat list* **where**
dims1 = *nths dims vars*

definition *dims2* :: *nat list* **where**
dims2 = *nths dims* (−*vars*)

lemma *dims1-alter*:
assumes $avars \subseteq A$

shows *dims1* = *nths dims* (*A* ∩ *vars*)
⟨*proof*⟩

lemma *dims2-alter*:
assumes *avars* ⊆ *A*
shows *dims2* = *nths dims* (*A*−*vars*)
⟨*proof*⟩

Total dimension for the active variables

definition *d1* :: *nat* **where**
d1 = *prod-list dims1*

Total dimension for the non-active variables

definition *d2* :: *nat* **where**
d2 = *prod-list dims2*

Translating dimension in d to dimension in d1

definition *encode1* :: *nat* ⇒ *nat* **where**
encode1 i = *digit-decode dims1* (*nths* (*digit-encode dims i*) *vars*)

lemma *encode1-alter*:
assumes *avars* ⊆ *A*
shows *encode1 i* = *digit-decode dims1* (*nths* (*digit-encode dims i*) (*A* ∩ *vars*))
⟨*proof*⟩

Translating dimension in d to dimension in d2

definition *encode2* :: *nat* ⇒ *nat* **where**
encode2 i = *digit-decode dims2* (*nths* (*digit-encode dims i*) (−*vars*))

lemma *encode2-alter*:
assumes *avars* ⊆ *A*
shows *encode2 i* = *digit-decode dims2* (*nths* (*digit-encode dims i*) (*A*−*vars*))
⟨*proof*⟩

lemma *encode1-lt* [*simp*]:
assumes *i* < *d*
shows *encode1 i* < *d1*
⟨*proof*⟩

lemma *encode2-lt* [*simp*]:
assumes *i* < *d*
shows *encode2 i* < *d2*
⟨*proof*⟩

Given dimensions in d1 and d2, form dimension in d

fun *encode12* :: *nat* × *nat* ⇒ *nat* **where**
encode12 (*i*, *j*) = *digit-decode dims* (*weave vars* (*digit-encode dims1 i*) (*digit-encode dims2 j*))
declare *encode12.simps* [*simp del*]

lemma *encode12-inv*:
  assumes $k < d$
  shows *encode12* (*encode1 k*, *encode2 k*) = *k*
  ⟨*proof*⟩

lemma *encode12-inv1*:
  assumes $i < d1$ $j < d2$
  shows *encode1* (*encode12* (*i*, *j*)) = *i*
  ⟨*proof*⟩

lemma *encode12-inv2*:
  assumes $i < d1$ $j < d2$
  shows *encode2* (*encode12* (*i*, *j*)) = *j*
  ⟨*proof*⟩

lemma *encode12-lt*:
  assumes $i < d1$ $j < d2$
  shows *encode12* (*i*, *j*) < *d*
  ⟨*proof*⟩

lemma *sum-encode*: ($\sum i = 0..<d1.$ $\sum j = 0..<d2.$ *f i j*) = *sum* (λ*k*. *f* (*encode1 k*) (*encode2 k*)) {*0..<d*}
  ⟨*proof*⟩

4.2 Tensor product of vectors and matrices

Given vector v1 of dimension d1, and vector v2 of dimension d2, form the tensor vector of dimension d1 * d2 = d

definition *tensor-vec* :: *'a::times vec* ⇒ *'a vec* ⇒ *'a vec* **where**
  tensor-vec v1 v2 = *Matrix.vec d* (λ*i*. *v1* $ *encode1 i* ∗ *v2* $ *encode2 i*)

lemma *tensor-vec-dim* [*simp*]:
  dim-vec (*tensor-vec v1 v2*) = *d*
  ⟨*proof*⟩

lemma *tensor-vec-carrier*:
  tensor-vec v1 v2 ∈ *carrier-vec d*
  ⟨*proof*⟩

lemma *tensor-vec-eval*:
  assumes $i < d$
  shows *tensor-vec v1 v2* $ *i* = *v1* $ *encode1 i* ∗ *v2* $ *encode2 i*
  ⟨*proof*⟩

lemma *tensor-vec-add1*:
  fixes *v1 v2 v3* :: *'a::comm-ring vec*
  assumes *v1* ∈ *carrier-vec d1*
    and *v2* ∈ *carrier-vec d1*

and *v3* ∈ *carrier-vec d2*
shows *tensor-vec* (*v1* + *v2*) *v3* = *tensor-vec v1 v3* + *tensor-vec v2 v3*
⟨*proof*⟩

lemma *tensor-vec-add2*:
fixes *v1 v2 v3* :: *'a::comm-ring vec*
assumes *v1* ∈ *carrier-vec d1*
and *v2* ∈ *carrier-vec d2*
and *v3* ∈ *carrier-vec d2*
shows *tensor-vec v1* (*v2* + *v3*) = *tensor-vec v1 v2* + *tensor-vec v1 v3*
⟨*proof*⟩

Given d1-by-d1 matrix m1, and d2-by-d2 matrix m2, form d-by-d matrix

definition *tensor-mat* :: *'a::comm-ring-1 mat* ⇒ *'a mat* ⇒ *'a mat* **where**
tensor-mat m1 m2 = *Matrix.mat d d* (λ(*i*,*j*).
m1 $$ (*encode1 i*, *encode1 j*) ∗ *m2* $$ (*encode2 i*, *encode2 j*))

lemma *tensor-mat-dim-row* [*simp*]:
dim-row (*tensor-mat m1 m2*) = *d*
⟨*proof*⟩

lemma *tensor-mat-dim-col* [*simp*]:
dim-col (*tensor-mat m1 m2*) = *d*
⟨*proof*⟩

lemma *tensor-mat-carrier*:
tensor-mat m1 m2 ∈ *carrier-mat d d*
⟨*proof*⟩

lemma *tensor-mat-eval*:
assumes *i* < *d j* < *d*
shows *tensor-mat m1 m2* $$ (*i*, *j*) = *m1* $$ (*encode1 i*, *encode1 j*) ∗ *m2* $$ (*encode2 i*, *encode2 j*)
⟨*proof*⟩

lemma *tensor-mat-zero1*:
shows *tensor-mat* (0_m *d1 d1*) *m1* = 0_m *d d*
⟨*proof*⟩

lemma *tensor-mat-zero2*:
shows *tensor-mat m1* (0_m *d2 d2*) = 0_m *d d*
⟨*proof*⟩

lemma *tensor-mat-add1*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
and *m3* ∈ *carrier-mat d2 d2*
shows *tensor-mat* (*m1* + *m2*) *m3* = *tensor-mat m1 m3* + *tensor-mat m2 m3*
⟨*proof*⟩

lemma *tensor-mat-add2*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d2\ d2$
and $m3 \in carrier\text{-}mat\ d2\ d2$
shows $tensor\text{-}mat\ m1\ (m2 + m3) = tensor\text{-}mat\ m1\ m2 + tensor\text{-}mat\ m1\ m3$
⟨*proof*⟩

lemma *tensor-mat-minus1*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d1\ d1$
and $m3 \in carrier\text{-}mat\ d2\ d2$
shows $tensor\text{-}mat\ (m1 - m2)\ m3 = tensor\text{-}mat\ m1\ m3 - tensor\text{-}mat\ m2\ m3$
⟨*proof*⟩

lemma *tensor-mat-matrix-sum2*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d2\ d2)$
$\Longrightarrow matrix\text{-}sum\ d\ (\lambda k.\ tensor\text{-}mat\ m1\ (f\ k))\ n = tensor\text{-}mat\ m1\ (matrix\text{-}sum\ d2\ f\ n)$
⟨*proof*⟩

lemma *tensor-mat-scale1*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d2\ d2$
shows $tensor\text{-}mat\ (a \cdot_m m1)\ m2 = a \cdot_m tensor\text{-}mat\ m1\ m2$
⟨*proof*⟩

lemma *tensor-mat-scale2*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d2\ d2$
shows $tensor\text{-}mat\ m1\ (a \cdot_m m2) = a \cdot_m tensor\text{-}mat\ m1\ m2$
⟨*proof*⟩

lemma *tensor-mat-trace*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d2\ d2$
shows $trace\ (tensor\text{-}mat\ m1\ m2) = trace\ m1 * trace\ m2$
⟨*proof*⟩

lemma *tensor-mat-id*:
$tensor\text{-}mat\ (1_m\ d1)\ (1_m\ d2) = 1_m\ d$
⟨*proof*⟩

lemma *tensor-mat-mult-vec*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d2\ d2$
and $v1 \in carrier\text{-}vec\ d1$
and $v2 \in carrier\text{-}vec\ d2$

shows *tensor-vec* $(m1 *_v v1)$ $(m2 *_v v2)$ = *tensor-mat m1 m2* $*_v$ *tensor-vec v1 v2*
⟨*proof*⟩

lemma *tensor-mat-mult*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
and *m3* ∈ *carrier-mat d2 d2*
and *m4* ∈ *carrier-mat d2 d2*
shows *tensor-mat* (*m1* ∗ *m2*) (*m3* ∗ *m4*) = *tensor-mat m1 m3* ∗ *tensor-mat m2 m4*
⟨*proof*⟩

lemma *tensor-mat-adjoint*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
shows *adjoint* (*tensor-mat m1 m2*) = *tensor-mat* (*adjoint m1*) (*adjoint m2*)
⟨*proof*⟩

lemma *tensor-mat-hermitian*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
and *hermitian m1*
and *hermitian m2*
shows *hermitian* (*tensor-mat m1 m2*)
⟨*proof*⟩

lemma *tensor-mat-unitary*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
and *unitary m1*
and *unitary m2*
shows *unitary* (*tensor-mat m1 m2*)
⟨*proof*⟩

lemma *tensor-mat-positive*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
and *positive m1*
and *positive m2*
shows *positive* (*tensor-mat m1 m2*)
⟨*proof*⟩

lemma *tensor-mat-positive-le*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
and *positive m1*
and *positive m2*
and $m1 \leq_L A$

and $m2 \leq_L B$
shows $tensor\text{-}mat\ m1\ m2 \leq_L tensor\text{-}mat\ A\ B$
⟨*proof*⟩

lemma *tensor-mat-le-one*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d2\ d2$
and *positive m1*
and *positive m2*
and $m1 \leq_L 1_m\ d1$
and $m2 \leq_L 1_m\ d2$
shows $tensor\text{-}mat\ m1\ m2 \leq_L 1_m\ d$
⟨*proof*⟩

4.3 Extension of matrices

definition *mat-extension* :: *'a::comm-ring-1 mat* ⇒ *'a mat* **where**
$mat\text{-}extension\ m = tensor\text{-}mat\ m\ (1_m\ d2)$

lemma *mat-extension-carrier*:
$mat\text{-}extension\ m \in carrier\text{-}mat\ d\ d$
⟨*proof*⟩

lemma *mat-extension-add*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d1\ d1$
shows $mat\text{-}extension\ (m1 + m2) = mat\text{-}extension\ m1 + mat\text{-}extension\ m2$
⟨*proof*⟩

lemma *mat-extension-trace*:
assumes $m \in carrier\text{-}mat\ d1\ d1$
shows $trace\ (mat\text{-}extension\ m) = d2 * trace\ m$
⟨*proof*⟩

lemma *mat-extension-id*:
$mat\text{-}extension\ (1_m\ d1) = 1_m\ d$
⟨*proof*⟩

lemma *mat-extension-mult*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d1\ d1$
shows $mat\text{-}extension\ (m1 * m2) = mat\text{-}extension\ m1 * mat\text{-}extension\ m2$
⟨*proof*⟩

lemma *mat-extension-hermitian*:
assumes $m \in carrier\text{-}mat\ d1\ d1$
and *hermitian m*
shows *hermitian (mat-extension m)*
⟨*proof*⟩

lemma *mat-extension-unitary*:
  assumes *m* ∈ *carrier-mat d1 d1*
    and *unitary m*
  shows *unitary* (*mat-extension m*)
  ⟨*proof*⟩

end

abbreviation *tensor-mat* ≡ *partial-state.tensor-mat*
abbreviation *mat-extension* ≡ *partial-state.mat-extension*

context *state-sig*
begin

Swapping the order of matrices, as well as switching vars, should have no effect

lemma *tensor-mat-comm*:
  assumes *vars1* ∩ *vars2* = {}
    and {*0..<length dims*} ⊆ *vars1* ∪ *vars2*
    and *m1* ∈ *carrier-mat* (*prod-list* (*nths dims vars1*)) (*prod-list* (*nths dims vars1*))
    and *m2* ∈ *carrier-mat* (*prod-list* (*nths dims vars2*)) (*prod-list* (*nths dims vars2*))
  shows *tensor-mat dims vars1 m1 m2* = *tensor-mat dims vars2 m2 m1*
⟨*proof*⟩
end

4.4 Partial tensor product

In this context, we assume two disjoint sets of variables, and define the tensor product of two matrices on these variables

locale *partial-state2* = *state-sig* +
  fixes *vars1* :: *nat set*
    and *vars2* :: *nat set*
  assumes *disjoint*: *vars1* ∩ *vars2* = {}

begin

definition *vars0* :: *nat set* **where**
  vars0 = *vars1* ∪ *vars2*

definition *dims0* :: *nat list* **where**
  dims0 = *nths dims vars0*

definition *dims1* :: *nat list* **where**
  dims1 = *nths dims vars1*

definition *dims2* :: *nat list* **where**
  dims2 = *nths dims vars2*

definition *d0* :: *nat* **where**
d0 = *prod-list dims0*

definition *d1* :: *nat* **where**
d1 = *prod-list dims1*

definition *d2* :: *nat* **where**
d2 = *prod-list dims2*

lemma *dims-product*:
d0 = *d1* ∗ *d2*
⟨*proof*⟩

Locations of variables in vars1 relative to vars0. For example: if vars0 = 0,1,2,4,5,6,9 and vars1 = 1,4,6,9, then vars1' should be 1,3,5,6.

definition *vars1'* :: *nat set* **where**
vars1' = (*ind-in-set vars0*) ' *vars1*

definition *vars2'* :: *nat set* **where**
vars2' = (*ind-in-set vars0*) ' *vars2*

lemma *vars1'I*:
x ∈ *vars1* ⟹ *card* {*y*∈*vars0*. *y* < *x*} ∈ *vars1'*
⟨*proof*⟩

lemma *vars1'D*:
i ∈ *vars1'* ⟹ ∃ *x*∈*vars1*. *card* {*y*∈*vars0*. *y* < *x*} = *i*
⟨*proof*⟩

Main property of vars1'

lemma *ind-in-set-bij*:
bij-betw (*ind-in-set vars0*) ({*0*..<*length dims*} ∩ *vars0*) {*0*..<*card* ({*0*..<*length dims*} ∩ *vars0*)}
⟨*proof*⟩

lemma *length-dims0*:
length dims0 = *card* ({*0*..<*length dims*} ∩ *vars0*)
⟨*proof*⟩

lemma *length-dims0-minus-vars2'-is-vars1'*:
{*0*..<*length dims0*} − *vars2'* = {*0*..<*length dims0*} ∩ *vars1'*
⟨*proof*⟩

lemma *length-dims0-minus-vars1'-is-vars2'*:
{*0*..<*length dims0*} − *vars1'* = {*0*..<*length dims0*} ∩ *vars2'*
⟨*proof*⟩

lemma *nths-vars1'*:
nths dims0 vars1' = *dims1*

⟨*proof*⟩

lemma *nths-vars1′-comp*:
nths dims0 (−*vars2′*) = *dims1*
⟨*proof*⟩

lemma *nths-vars2′*:
nths dims0 (−*vars1′*) = *dims2*
⟨*proof*⟩

lemma *nths-vars2′-comp*:
nths dims0 (*vars2′*) = *dims2*
⟨*proof*⟩

lemma *ptensor-encode1-encode2*:
partial-state.encode1 dims0 vars1′ = *partial-state.encode2 dims0 vars2′*
⟨*proof*⟩

lemma *ptensor-encode2-encode1*:
partial-state.encode1 dims0 vars2′ = *partial-state.encode2 dims0 vars1′*
⟨*proof*⟩

Given vector v1 of dimension d1, and vector v2 of dimension d2, form the tensor vector of dimension d1 * d2 = d0

definition *ptensor-vec* :: ′*a*::*times vec* ⇒ ′*a vec* ⇒ ′*a vec* **where**
ptensor-vec v1 v2 = *partial-state.tensor-vec dims0 vars1′ v1 v2*

lemma *ptensor-vec-dim* [*simp*]:
dim-vec (*ptensor-vec v1 v2*) = *d0*
⟨*proof*⟩

lemma *ptensor-vec-carrier*:
ptensor-vec v1 v2 ∈ *carrier-vec d0*
⟨*proof*⟩

lemma *ptensor-vec-add*:
fixes *v1 v2 v3* :: ′*a*::*comm-ring vec*
assumes *v1* ∈ *carrier-vec d1*
and *v2* ∈ *carrier-vec d1*
and *v3* ∈ *carrier-vec d2*
shows *ptensor-vec* (*v1* + *v2*) *v3* = *ptensor-vec v1 v3* + *ptensor-vec v2 v3*
⟨*proof*⟩

definition *ptensor-mat* :: ′*a*::*comm-ring-1 mat* ⇒ ′*a mat* ⇒ ′*a mat* **where**
ptensor-mat m1 m2 = *partial-state.tensor-mat dims0 vars1′ m1 m2*

lemma *ptensor-mat-dim-row* [*simp*]:
dim-row (*ptensor-mat m1 m2*) = *d0*
⟨*proof*⟩

lemma *ptensor-mat-dim-col* [*simp*]:
dim-col (*ptensor-mat m1 m2*) = *d0*
⟨*proof*⟩

lemma *ptensor-mat-carrier*:
ptensor-mat m1 m2 ∈ *carrier-mat d0 d0*
⟨*proof*⟩

lemma *ptensor-mat-add*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
and *m3* ∈ *carrier-mat d2 d2*
shows *ptensor-mat* (*m1* + *m2*) *m3* = *ptensor-mat m1 m3* + *ptensor-mat m2 m3*
⟨*proof*⟩

lemma *ptensor-mat-trace*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
shows *trace* (*ptensor-mat m1 m2*) = *trace m1* ∗ *trace m2*
⟨*proof*⟩

lemma *ptensor-mat-id*:
ptensor-mat (1_m *d1*) (1_m *d2*) = 1_m *d0*
⟨*proof*⟩

lemma *ptensor-mat-mult*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
and *m3* ∈ *carrier-mat d2 d2*
and *m4* ∈ *carrier-mat d2 d2*
shows *ptensor-mat* (*m1* ∗ *m2*) (*m3* ∗ *m4*) = *ptensor-mat m1 m3* ∗ *ptensor-mat m2 m4*
⟨*proof*⟩

lemma *ptensor-mat-mult-vec*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *v1* ∈ *carrier-vec d1*
and *m2* ∈ *carrier-mat d2 d2*
and *v2* ∈ *carrier-vec d2*
shows *ptensor-vec* (*m1* $*_v$ *v1*) (*m2* $*_v$ *v2*) = *ptensor-mat m1 m2* $*_v$ *ptensor-vec v1 v2*
⟨*proof*⟩

4.5 Partial extensions

definition *pmat-extension* :: *'a::comm-ring-1 mat* ⇒ *'a mat* **where**
pmat-extension m = *ptensor-mat m* (1_m *d2*)

lemma *pmat-extension-carrier*:
pmat-extension m ∈ *carrier-mat d0 d0*
⟨*proof*⟩

lemma *pmat-extension-add*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
shows *pmat-extension* (*m1* + *m2*) = *pmat-extension m1* + *pmat-extension m2*
⟨*proof*⟩

lemma *pmat-extension-trace*:
assumes *m* ∈ *carrier-mat d1 d1*
shows *trace* (*pmat-extension m*) = *d2* ∗ *trace m*
⟨*proof*⟩

lemma *pmat-extension-id*:
pmat-extension (1_m *d1*) = 1_m *d0*
⟨*proof*⟩

lemma *pmat-extension-mult*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
shows *pmat-extension* (*m1* ∗ *m2*) = *pmat-extension m1* ∗ *pmat-extension m2*
⟨*proof*⟩

end

context *state-sig*
begin

abbreviation *ptensor-vec* ≡ *partial-state2.ptensor-vec*
abbreviation *ptensor-mat* ≡ *partial-state2.ptensor-mat*
abbreviation *pmat-extension* ≡ *partial-state2.pmat-extension*

Key property: commutativity of tensor product

lemma *ptensor-mat-comm*:
fixes *m1 m2* :: *complex mat*
assumes *vars1* ∩ *vars2* = {}
shows *ptensor-mat dims vars1 vars2 m1 m2* = *ptensor-mat dims vars2 vars1 m2 m1*
⟨*proof*⟩

Key property: associativity of tensor product

lemma *ind-in-set-mono*:
fixes *a b* :: *nat* **and** *A* :: *nat set*
assumes *a* ∈ *A b* ∈ *A a* < *b*
shows *ind-in-set A a* < *ind-in-set A b*
⟨*proof*⟩

lemma *ind-in-set-inj*:
  fixes *a b* :: *nat* **and** *A* :: *nat set*
  assumes $a \in A$ $b \in A$ *ind-in-set* A a = *ind-in-set* A b
  shows $a = b$
⟨*proof*⟩

lemma *ind-in-set-mono2*:
  fixes *a b* :: *nat* **and** *A* :: *nat set*
  assumes $a \in A$ $b \in A$ *ind-in-set* A a < *ind-in-set* A b
  shows $a < b$
  ⟨*proof*⟩

lemma *ind-in-set-bij-betw*:
  fixes *A B* :: *nat set*
  assumes $B \subseteq A$ $c \in B$
  shows *bij-betw* (*ind-in-set* A) $\{i \in B.\ i < c\}$ $\{i \in$ *ind-in-set* A ‘ $B.\ i <$ *ind-in-set* $A\ c\}$
  ⟨*proof*⟩

lemma *ind-in-set-assoc*:
  fixes *A B C* :: *nat set*
  assumes $C \subseteq B$ $B \subseteq A$
  shows *ind-in-set* (*ind-in-set* A ‘ B) ‘ (*ind-in-set* A ‘ C) = *ind-in-set* B ‘ C
⟨*proof*⟩

lemma *nths-reencode-eq3*:
  fixes *A B C* :: *nat set*
  assumes $C \subseteq B$ $B \subseteq A$
  shows *nths* (*nths xs* (*ind-in-set* A ‘ B)) (*ind-in-set* B ‘ C) = *nths xs* (*ind-in-set* A ‘ C)
  ⟨*proof*⟩

lemma *nths-assoc-three-A*:
  fixes *A B C* :: *nat set*
  assumes $A \cap B = \{\}$
    and $(A \cup B) \cap C = \{\}$
  shows *nths* (*nths xs* (*ind-in-set* $(A \cup B \cup C)$ ‘ $(A \cup B)$)) (*ind-in-set* $(A \cup B)$ ‘ A)
      = *nths xs* (*ind-in-set* $(A \cup B \cup C)$ ‘ A)
  ⟨*proof*⟩

lemma *nths-assoc-three-B*:
  fixes *A B C* :: *nat set*
  assumes $A \cap B = \{\}$
    and $(A \cup B) \cap C = \{\}$
  shows *nths* (*nths xs* (*ind-in-set* $(A \cup B \cup C)$ ‘ $(A \cup B)$)) (*ind-in-set* $(A \cup B)$ ‘ B)
      = *nths* (*nths xs* (*ind-in-set* $(A \cup B \cup C)$ ‘ $(B \cup C)$)) (*ind-in-set* $(B \cup C)$ ‘

B)
⟨*proof*⟩

lemma *nths-assoc-three-C*:
fixes *A B C* :: *nat set*
assumes $A \cap B = \{\}$
and $(A \cup B) \cap C = \{\}$
shows *nths* (*nths xs* (*ind-in-set* $(A \cup B \cup C)$ ‘ $(B \cup C)$)) (*ind-in-set* $(B \cup C)$ ‘ *C*)
= *nths xs* (*ind-in-set* $(A \cup B \cup C)$ ‘ *C*)
⟨*proof*⟩

lemma *valid-index-ind-in-set*:
assumes *is* ◁ *nths dims A B* ⊆ *A*
shows *nths is* (*ind-in-set A* ‘ *B*) ◁ *nths dims B*
⟨*proof*⟩

lemma *ind-in-set-id*:
fixes *A* :: *nat set*
assumes *finite A*
shows *ind-in-set A* ‘ *A* = {*0*..< *card A*}
⟨*proof*⟩

lemma *nths-complement-ind-in-set*:
fixes *A B* :: *nat set*
assumes $A \cap B = \{\}$
card $(A \cup B)$ = *length xs*
shows *nths xs* (− *ind-in-set* $(A \cup B)$ ‘ *A*) = *nths xs* (*ind-in-set* $(A \cup B)$ ‘ *B*)
⟨*proof*⟩

lemma *ind-in-set-inj′*:
fixes *A B* :: *nat set*
assumes $B \subseteq A$
shows *inj-on* (*ind-in-set A*) *B*
⟨*proof*⟩

lemma *ind-in-set-less*:
fixes *x* :: *nat* **and** *A* :: *nat set*
assumes *finite A* $x \in A$
shows *ind-in-set A x* < *card A*
⟨*proof*⟩

lemma *ptensor-mat-assoc*:
assumes *vars1* ∩ *vars2* = {}
and (*vars1* ∪ *vars2*) ∩ *vars3* = {}
and *vars1* ∪ *vars2* ∪ *vars3* ⊆ {*0*..< *length dims*}
shows *ptensor-mat dims* (*vars1* ∪ *vars2*) *vars3* (*ptensor-mat dims vars1 vars2 m1 m2*) *m3* =
ptensor-mat dims vars1 (*vars2* ∪ *vars3*) *m1* (*ptensor-mat dims vars2 vars3*

m2 m3)
⟨*proof*⟩

Some simple consequences of associativity

lemma *pmat-extension-assoc*:
assumes *vars1* ∩ *vars2* = {}
and (*vars1* ∪ *vars2*) ∩ *vars3* = {}
and *vars1* ∪ *vars2* ∪ *vars3* ⊆ {*0..*< *length dims*}
shows *pmat-extension dims vars1* (*vars2* ∪ *vars3*) *m* =
pmat-extension dims (*vars1* ∪ *vars2*) *vars3* (*pmat-extension dims vars1 vars2 m*)
⟨*proof*⟩

end

4.6 Commands on subset of variables

context *state-sig*
begin

definition *Utrans-P* :: *nat set* ⇒ *complex mat* ⇒ *com* **where**
Utrans-P vars U = *Utrans* (*mat-extension dims vars U*)

lemma *well-com-Utrans-P*:
assumes *U* ∈ *carrier-mat* (*prod-list* (*nths dims vars*)) (*prod-list* (*nths dims vars*))
and *unitary U*
shows *well-com* (*Utrans-P vars U*)
⟨*proof*⟩

definition *Measure-P* :: *nat set* ⇒ *nat* ⇒ (*nat* ⇒ *complex mat*) ⇒ *com list* ⇒ *com* **where**
Measure-P vars n Ps Cs = *Measure n* (λ*n. mat-extension dims vars* (*Ps n*)) *Cs*

definition *While-P* :: *nat set* ⇒ *complex mat* ⇒ *complex mat* ⇒ *com* ⇒ *com* **where**
While-P vars M0 M1 C = *While* (λ*n.*
if n = *0 then mat-extension dims vars M0*
else if n = *1 then mat-extension dims vars M1*
else undefined) *C*

end

end

5 Standard gates

theory *Gates*
imports *Complex-Matrix*
begin

Pauli matrices

definition *sigma-x* :: *complex mat* **where**
sigma-x = *mat-of-rows-list 2* [[*0*, *1*], [*1*, *0*]]

definition *sigma-y* :: *complex mat* **where**
sigma-y = *mat-of-rows-list 2* [[*0*, −i], [i, *0*]]

definition *sigma-z* :: *complex mat* **where**
sigma-z = *mat-of-rows-list 2* [[*1*, *0*], [*0*, −*1*]]

Hadamard matrices

definition *hadamard* :: *complex mat* **where**
hadamard = *mat 2 2* (λ(*i*, *j*). *if* (*i* = *0* ∨ *j* = *0*) *then 1* / *csqrt 2 else* − *1* / *sqrt 2*)

lemma *hadamard-dim*:
hadamard ∈ *carrier-mat 2 2*
⟨*proof*⟩

lemma *hermitian-hadamard*:
hermitian hadamard
⟨*proof*⟩

lemma *csqrt-2-sq*:
complex-of-real (*sqrt 2*) ∗ *complex-of-real* (*sqrt 2*) = *2*
⟨*proof*⟩

lemma *sum-le-2*:
⋀(*f*::*nat*⇒*complex*). *sum f* {*0*..<*2*} = *f 0* + *f 1*
⟨*proof*⟩

lemma *unitary-hadamard*:
unitary hadamard
⟨*proof*⟩

The matrix [0 0 .. 0 1 1 0 .. 0 0 0 1 .. 0 0 . . .. . . 0 0 .. 1 0] implements i := i + 1 in the last variable.

definition *mat-incr* :: *nat* ⇒ *complex mat* **where**
mat-incr n = *mat n n* (λ(*i*,*j*). *if i* = *0 then* (*if j* = *n* − *1 then 1 else 0*) *else* (*if i* = *j* + *1 then 1 else 0*))

lemma *mat-incr-dim*:
mat-incr n ∈ *carrier-mat n n*
⟨*proof*⟩

lemma *adjoint-mat-incr*:
adjoint (*mat-incr n*) = *mat n n* (λ(*i*,*j*). *if j* = *0 then* (*if i* = *n* − *1 then 1 else 0*) *else* (*if j* = *i* + *1 then 1 else 0*))
⟨*proof*⟩

lemma *mat-incr-mult-adjoint-mat-incr*:
shows *mat-incr n* ∗ (*adjoint* (*mat-incr n*)) = 1_m *n*
⟨*proof*⟩

lemma *unitary-mat-incr*:
unitary (*mat-incr n*)
⟨*proof*⟩

end

6 Partial and total correctness

theory *Quantum-Hoare*
imports *Quantum-Program*
begin

context *state-sig*
begin

definition *density-states* :: *state set* **where**
density-states = {ϱ ∈ *carrier-mat d d*. *partial-density-operator* ϱ}

lemma *denote-density-states*:
ϱ ∈ *density-states* ⟹ *well-com S* ⟹ *denote S* ϱ ∈ *density-states*
⟨*proof*⟩

definition *is-quantum-predicate* :: *complex mat* ⇒ *bool* **where**
is-quantum-predicate P ⟷ *P* ∈ *carrier-mat d d* ∧ *positive P* ∧ *P* $\leq_L$ 1_m *d*

lemma *trace-measurement2*:
assumes *m*: *measurement n 2 M* **and** *dA*: *A* ∈ *carrier-mat n n*
shows *trace* ((*M 0*) ∗ *A* ∗ *adjoint* (*M 0*)) + *trace* ((*M 1*) ∗ *A* ∗ *adjoint* (*M 1*))
= *trace A*
⟨*proof*⟩

lemma *qp-close-under-unitary-operator*:
fixes *U P* :: *complex mat*
assumes *dU*: *U* ∈ *carrier-mat d d*
and *u*: *unitary U*
and *qp*: *is-quantum-predicate P*
shows *is-quantum-predicate* (*adjoint U* ∗ *P* ∗ *U*)
⟨*proof*⟩

lemma *qps-after-measure-is-qp*:
assumes *m*: *measurement d n M* **and** *qpk*: ⋀*k*. *k* < *n* ⟹ *is-quantum-predicate*
(*P k*)
shows *is-quantum-predicate* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n*)
⟨*proof*⟩

definition *hoare-total-correct* :: *complex mat* ⇒ *com* ⇒ *complex mat* ⇒ *bool* (‹$\models_t$ {(*1*-)}/ (-)/ {(*1*-)}› *50*) **where**
$\models_t$ {*P*} *S* {*Q*} ⟷ (∀ ϱ∈*density-states*. *trace* (*P* ∗ ϱ) ≤ *trace* (*Q* ∗ *denote S* ϱ))

definition *hoare-partial-correct* :: *complex mat* ⇒ *com* ⇒ *complex mat* ⇒ *bool* (‹$\models_p$ {(*1*-)}/ (-)/ {(*1*-)}› *50*) **where**
$\models_p$ {*P*} *S* {*Q*} ⟷ (∀ ϱ∈*density-states*. *trace* (*P* ∗ ϱ) ≤ *trace* (*Q* ∗ *denote S* ϱ) + (*trace* ϱ − *trace* (*denote S* ϱ)))

lemma *total-implies-partial*:
assumes *S*: *well-com S*
and *total*: $\models_t$ {*P*} *S* {*Q*}
shows $\models_p$ {*P*} *S* {*Q*}
⟨*proof*⟩

lemma *predicate-prob-positive*:
assumes 0_m *d d* $\leq_L$ *P*
and ϱ ∈ *density-states*
shows *0* ≤ *trace* (*P* ∗ ϱ)
⟨*proof*⟩

lemma *total-pre-zero*:
assumes *S*: *well-com S*
and *Q*: *is-quantum-predicate Q*
shows $\models_t$ {0_m *d d*} *S* {*Q*}
⟨*proof*⟩

lemma *partial-post-identity*:
assumes *S*: *well-com S*
and *P*: *is-quantum-predicate P*
shows $\models_p$ {*P*} *S* {1_m *d*}
⟨*proof*⟩

6.1 Weakest liberal preconditions

definition *is-weakest-liberal-precondition* :: *complex mat* ⇒ *com* ⇒ *complex mat* ⇒ *bool* **where**
is-weakest-liberal-precondition W S P ⟷
is-quantum-predicate W ∧ $\models_p$ {*W*} *S* {*P*} ∧ (∀ *Q*. *is-quantum-predicate Q* ⟶ $\models_p$ {*Q*} *S* {*P*} ⟶ *Q* $\leq_L$ *W*)

definition *wlp-measure* :: *nat* ⇒ (*nat* ⇒ *complex mat*) ⇒ ((*complex mat* ⇒ *complex mat*) *list*) ⇒ *complex mat* ⇒ *complex mat* **where**
wlp-measure n M WS P = *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ ((*WS*!*k*) *P*) ∗ (*M k*)) *n*

fun *wlp-while-n* :: *complex mat* ⇒ *complex mat* ⇒ (*complex mat* ⇒ *complex mat*) ⇒ *nat* ⇒ *complex mat* ⇒ *complex mat* **where**
wlp-while-n M0 M1 WS 0 P = 1_m *d*
| *wlp-while-n M0 M1 WS* (*Suc n*) *P* = *adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ (*WS* (*wlp-while-n M0 M1 WS n P*)) ∗ *M1*

lemma *measurement2-leq-one-mat*:
assumes *dP*: *P* ∈ *carrier-mat d d* **and** *dQ*: *Q* ∈ *carrier-mat d d*
and *leP*: *P* $\leq_L$ 1_m *d* **and** *leQ*: *Q* $\leq_L$ 1_m *d* **and** *m*: *measurement d 2 M*
shows (*adjoint* (*M 0*) ∗ *P* ∗ (*M 0*) + *adjoint* (*M 1*) ∗ *Q* ∗ (*M 1*)) $\leq_L$ 1_m *d*
⟨*proof*⟩

lemma *wlp-while-n-close*:
assumes *close*: ⋀*P*. *is-quantum-predicate P* ⟹ *is-quantum-predicate* (*WS P*)
and *m*: *measurement d 2 M* **and** *qpP*: *is-quantum-predicate P*
shows *is-quantum-predicate* (*wlp-while-n* (*M 0*) (*M 1*) *WS k P*)
⟨*proof*⟩

lemma *wlp-while-n-mono*:
assumes ⋀*P*. *is-quantum-predicate P* ⟹ *is-quantum-predicate* (*WS P*)
and ⋀*P Q*. *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ *P* $\leq_L$ *Q* ⟹ *WS P* $\leq_L$ *WS Q*
and *measurement d 2 M*
and *is-quantum-predicate P*
and *is-quantum-predicate Q*
and *P* $\leq_L$ *Q*
shows (*wlp-while-n* (*M 0*) (*M 1*) *WS k P*) $\leq_L$ (*wlp-while-n* (*M 0*) (*M 1*) *WS k Q*)
⟨*proof*⟩

definition *wlp-while* :: *complex mat* ⇒ *complex mat* ⇒ (*complex mat* ⇒ *complex mat*) ⇒ *complex mat* ⇒ *complex mat* **where**
wlp-while M0 M1 WS P = (*THE Q*. *limit-mat* (λ*n*. *wlp-while-n M0 M1 WS n P*) *Q d*)

lemma *wlp-while-exists*:
assumes ⋀*P*. *is-quantum-predicate P* ⟹ *is-quantum-predicate* (*WS P*)
and ⋀*P Q*. *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ *P* $\leq_L$ *Q* ⟹ *WS P* $\leq_L$ *WS Q*
and *m*: *measurement d 2 M*
and *qpP*: *is-quantum-predicate P*
shows *is-quantum-predicate* (*wlp-while* (*M 0*) (*M 1*) *WS P*)
∧ (∀ *n*. (*wlp-while* (*M 0*) (*M 1*) *WS P*) $\leq_L$ (*wlp-while-n* (*M 0*) (*M 1*) *WS n P*))
∧ (∀ *W'*. (∀ *n*. *W'* $\leq_L$ (*wlp-while-n* (*M 0*) (*M 1*) *WS n P*)) ⟶ *W'* $\leq_L$ (*wlp-while* (*M 0*) (*M 1*) *WS P*))
∧ *limit-mat* (λ*n*. *wlp-while-n* (*M 0*) (*M 1*) *WS n P*) (*wlp-while* (*M 0*) (*M 1*) *WS P*) *d*

⟨*proof*⟩

lemma *wlp-while-mono*:
assumes $\bigwedge P$. *is-quantum-predicate* $P \Longrightarrow$ *is-quantum-predicate* (*WS* P)
and $\bigwedge P\ Q$. *is-quantum-predicate* $P \Longrightarrow$ *is-quantum-predicate* $Q \Longrightarrow P \leq_L Q \Longrightarrow$ *WS* $P \leq_L$ *WS* Q
and *measurement d 2 M*
and *is-quantum-predicate P*
and *is-quantum-predicate Q*
and $P \leq_L Q$
shows *wlp-while* (*M 0*) (*M 1*) *WS* $P \leq_L$ *wlp-while* (*M 0*) (*M 1*) *WS Q*
⟨*proof*⟩

fun *wlp* :: *com* ⇒ *complex mat* ⇒ *complex mat* **where**
wlp SKIP P = *P*
| *wlp* (*Utrans U*) *P* = *adjoint U* ∗ *P* ∗ *U*
| *wlp* (*Seq S1 S2*) *P* = *wlp S1* (*wlp S2 P*)
| *wlp* (*Measure n M S*) *P* = *wlp-measure n M* (*map wlp S*) *P*
| *wlp* (*While M S*) *P* = *wlp-while* (*M 0*) (*M 1*) (*wlp S*) *P*

lemma *wlp-measure-expand-m*:
assumes *m*: $m \leq n$ **and** *wc*: *well-com* (*Measure n M S*)
shows *wlp* (*Measure m M S*) *P* = *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ (*M k*)) *m*
⟨*proof*⟩

lemma *wlp-measure-expand*:
assumes *wc*: *well-com* (*Measure n M S*)
shows *wlp* (*Measure n M S*) *P* = *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ (*M k*)) *n*
⟨*proof*⟩

lemma *wlp-mono-and-close*:
shows *well-com S* ⟹ *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ $P \leq_L Q$
⟹ *is-quantum-predicate* (*wlp S P*) ∧ *wlp S P* $\leq_L$ *wlp S Q*
⟨*proof*⟩

lemma *wlp-close*:
assumes *wc*: *well-com S* **and** *qp*: *is-quantum-predicate P*
shows *is-quantum-predicate* (*wlp S P*)
⟨*proof*⟩

lemma *wlp-soundness*:
well-com S ⟹
($\bigwedge P$. (*is-quantum-predicate P* ⟹
(∀ ϱ ∈ *density-states*. *trace* (*wlp S P* ∗ ϱ) = *trace* (*P* ∗ (*denote S* ϱ)) + *trace* ϱ − *trace* (*denote S* ϱ))))
⟨*proof*⟩

lemma *denote-while-split*:
  assumes *wc*: *well-com* (*While M S*) **and** *dsr*: $\varrho \in$ *density-states*
  shows *denote* (*While M S*) ϱ = (*M 0*) $*$ ϱ $*$ *adjoint* (*M 0*) + *denote* (*While M S*) (*denote S* (*M 1* $*$ ϱ $*$ *adjoint* (*M 1*)))
⟨*proof*⟩

lemma *wlp-while-split*:
  assumes *wc*: *well-com* (*While M S*) **and** *qpP*: *is-quantum-predicate P*
  shows *wlp* (*While M S*) *P* = *adjoint* (*M 0*) $*$ *P* $*$ (*M 0*) + *adjoint* (*M 1*) $*$ (*wlp S* (*wlp* (*While M S*) *P*)) $*$ (*M 1*)
⟨*proof*⟩

lemma *wlp-is-weakest-liberal-precondition*:
  assumes *well-com S* **and** *is-quantum-predicate P*
  shows *is-weakest-liberal-precondition* (*wlp S P*) *S P*
  ⟨*proof*⟩

6.2 Hoare triples for partial correctness

inductive *hoare-partial* :: *complex mat* ⇒ *com* ⇒ *complex mat* ⇒ *bool* (‹$\vdash_p$ ({(*1*-)}/ (-)/ {(*1*-)})› *50*) **where**
  is-quantum-predicate P ⟹ $\vdash_p$ {*P*} *SKIP* {*P*}
| *is-quantum-predicate P* ⟹ $\vdash_p$ {*adjoint U* $*$ *P* $*$ *U*} *Utrans U* {*P*}
| *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ *is-quantum-predicate R* ⟹
   $\vdash_p$ {*P*} *S1* {*Q*} ⟹ $\vdash_p$ {*Q*} *S2* {*R*} ⟹
   $\vdash_p$ {*P*} *Seq S1 S2* {*R*}
| (⋀*k*. *k* < *n* ⟹ *is-quantum-predicate* (*P k*)) ⟹ *is-quantum-predicate Q* ⟹
   (⋀*k*. *k* < *n* ⟹ $\vdash_p$ {*P k*} *S* ! *k* {*Q*}) ⟹
   $\vdash_p$ {*matrix-sum d* (λ*k*. *adjoint* (*M k*) $*$ *P k* $*$ *M k*) *n*} *Measure n M S* {*Q*}
| *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹
   $\vdash_p$ {*Q*} *S* {*adjoint* (*M 0*) $*$ *P* $*$ *M 0* + *adjoint* (*M 1*) $*$ *Q* $*$ *M 1*} ⟹
   $\vdash_p$ {*adjoint* (*M 0*) $*$ *P* $*$ *M 0* + *adjoint* (*M 1*) $*$ *Q* $*$ *M 1*} *While M S* {*P*}
| *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ *is-quantum-predicate P'* ⟹ *is-quantum-predicate Q'* ⟹
   $P \le_L P'$ ⟹ $\vdash_p$ {*P'*} *S* {*Q'*} ⟹ $Q' \le_L Q$ ⟹ $\vdash_p$ {*P*} *S* {*Q*}

theorem *hoare-partial-sound*:
  $\vdash_p$ {*P*} *S* {*Q*} ⟹ *well-com S* ⟹ $\models_p$ {*P*} *S* {*Q*}
⟨*proof*⟩

lemma *wlp-complete*:
  well-com S ⟹ *is-quantum-predicate P* ⟹ $\vdash_p$ {*wlp S P*} *S* {*P*}
⟨*proof*⟩

theorem *hoare-partial-complete*:
  $\models_p$ {*P*} *S* {*Q*} ⟹ *well-com S* ⟹ *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ $\vdash_p$ {*P*} *S* {*Q*}

⟨*proof*⟩

6.3 Consequences of completeness

lemma *hoare-patial-seq-assoc-sem*:
shows $\models_p \{A\}\ (S1\ ;;\ S2)\ ;;\ S3\ \{B\} \longleftrightarrow \models_p \{A\}\ S1\ ;;\ (S2\ ;;\ S3)\ \{B\}$
⟨*proof*⟩

lemma *hoare-patial-seq-assoc*:
assumes *well-com S1* **and** *well-com S2* **and** *well-com S3*
and *is-quantum-predicate A* **and** *is-quantum-predicate B*
shows $\vdash_p \{A\}\ (S1\ ;;\ S2)\ ;;\ S3\ \{B\} \longleftrightarrow \vdash_p \{A\}\ S1\ ;;\ (S2\ ;;\ S3)\ \{B\}$
⟨*proof*⟩

end

end

7 Grover's algorithm

theory *Grover*
imports *Partial-State Gates Quantum-Hoare*
begin

7.1 Basic definitions

locale *grover-state* =
fixes n :: *nat*
and f :: *nat* ⇒ *bool*
assumes *n*: $n > 1$
and *dimM*: $card\ \{i.\ i < (2::nat)\ \widehat{}\ n \wedge f\ i\} > 0$
$card\ \{i.\ i < (2::nat)\ \widehat{}\ n \wedge f\ i\} < (2::nat)\ \widehat{}\ n$
begin

definition *N* **where**
$N = (2::nat)\ \widehat{}\ n$

definition *M* **where**
$M = card\ \{i.\ i < N \wedge f\ i\}$

lemma *N-ge-0* [*simp*]: $0 < N$ ⟨*proof*⟩

lemma *M-ge-0* [*simp*]: $0 < M$ ⟨*proof*⟩

lemma *M-neq-0* [*simp*]: $M \neq 0$ ⟨*proof*⟩

lemma *M-le-N* [*simp*]: $M < N$ ⟨*proof*⟩

lemma *M-not-ge-N* [*simp*]: $\neg\ M \geq N$ ⟨*proof*⟩

definition ψ :: *complex vec* **where**
$\psi = Matrix.vec\ N\ (\lambda i.\ 1\ /\ sqrt\ N)$

lemma ψ*-dim* [*simp*]:
$\psi \in carrier\text{-}vec\ N$
$dim\text{-}vec\ \psi = N$
⟨*proof*⟩

lemma ψ*-eval*:
$i < N \Longrightarrow \psi\ \$\ i = 1\ /\ sqrt\ N$
⟨*proof*⟩

lemma ψ*-inner*:
$inner\text{-}prod\ \psi\ \psi = 1$
⟨*proof*⟩

lemma ψ*-norm*:
$vec\text{-}norm\ \psi = 1$
⟨*proof*⟩

definition α :: *complex vec* **where**
$\alpha = Matrix.vec\ N\ (\lambda i.\ if\ f\ i\ then\ 0\ else\ 1\ /\ sqrt\ (N - M))$

lemma α*-dim* [*simp*]:
$\alpha \in carrier\text{-}vec\ N$
$dim\text{-}vec\ \alpha = N$
⟨*proof*⟩

lemma α*-eval*:
$i < N \Longrightarrow \alpha\ \$\ i = (if\ f\ i\ then\ 0\ else\ 1\ /\ sqrt\ (N - M))$
⟨*proof*⟩

lemma α*-inner*:
$inner\text{-}prod\ \alpha\ \alpha = 1$
⟨*proof*⟩

definition β :: *complex vec* **where**
$\beta = Matrix.vec\ N\ (\lambda i.\ if\ f\ i\ then\ 1\ /\ sqrt\ M\ else\ 0)$

lemma β*-dim* [*simp*]:
$\beta \in carrier\text{-}vec\ N$
$dim\text{-}vec\ \beta = N$
⟨*proof*⟩

lemma β*-eval*:
$i < N \Longrightarrow \beta\ \$\ i = (if\ f\ i\ then\ 1\ /\ sqrt\ M\ else\ 0)$
⟨*proof*⟩

lemma β*-inner*:
$inner\text{-}prod\ \beta\ \beta = 1$
⟨*proof*⟩

lemma *alpha-beta-orth*:
$inner\text{-}prod\ \alpha\ \beta = 0$
⟨*proof*⟩

lemma *beta-alpha-orth*:
$inner\text{-}prod\ \beta\ \alpha = 0$
⟨*proof*⟩

definition ϑ :: *real* **where**
$\vartheta = 2 * arccos\ (sqrt\ ((N - M) / N))$

lemma *cos-theta-div-2*:
$cos\ (\vartheta / 2) = sqrt\ ((N - M) / N)$
⟨*proof*⟩

lemma *sin-theta-div-2*:
$sin\ (\vartheta / 2) = sqrt\ (M / N)$
⟨*proof*⟩

lemma ϑ*-neq-0*:
$\vartheta \neq 0$
⟨*proof*⟩

abbreviation *ccos* **where** $ccos\ \varphi \equiv complex\text{-}of\text{-}real\ (cos\ \varphi)$
abbreviation *csin* **where** $csin\ \varphi \equiv complex\text{-}of\text{-}real\ (sin\ \varphi)$

lemma ψ*-eq*:
$\psi = ccos\ (\vartheta / 2) \cdot_v \alpha + csin\ (\vartheta / 2) \cdot_v \beta$
⟨*proof*⟩

lemma *psi-inner-alpha*:
$inner\text{-}prod\ \psi\ \alpha = ccos\ (\vartheta / 2)$
⟨*proof*⟩

lemma *psi-inner-beta*:
$inner\text{-}prod\ \psi\ \beta = csin\ (\vartheta / 2)$
⟨*proof*⟩

definition *alpha-l* :: *nat* ⇒ *complex* **where**
$alpha\text{-}l\ l = ccos\ ((l + 1 / 2) * \vartheta)$

lemma *alpha-l-real*:
$alpha\text{-}l\ l \in Reals$
⟨*proof*⟩

lemma *cnj-alpha-l*:
  conjugate (*alpha-l l*) = *alpha-l l*
  ⟨*proof*⟩

definition *beta-l* :: *nat* ⇒ *complex* **where**
  beta-l l = *csin* ((*l* + *1* / *2*) ∗ *ϑ*)

lemma *beta-l-real*:
  beta-l l ∈ *Reals*
  ⟨*proof*⟩

lemma *cnj-beta-l*:
  conjugate (*beta-l l*) = *beta-l l*
  ⟨*proof*⟩

lemma *csin-ccos-squared-add*:
  ccos (*a*::*real*) ∗ *ccos a* + *csin a* ∗ *csin a* = *1*
  ⟨*proof*⟩

lemma *alpha-l-beta-l-add-norm*:
  alpha-l l ∗ *alpha-l l* + *beta-l l* ∗ *beta-l l* = *1*
  ⟨*proof*⟩

definition *psi-l* **where**
  psi-l l = (*alpha-l l*) $\cdot_v$ *α* + (*beta-l l*) $\cdot_v$ *β*

lemma *psi-l-dim*:
  psi-l l ∈ *carrier-vec N*
  ⟨*proof*⟩

lemma *inner-psi-l*:
  inner-prod (*psi-l l*) (*psi-l l*) = *1*
⟨*proof*⟩

abbreviation *proj* :: *complex vec* ⇒ *complex mat* **where**
  proj v ≡ *outer-prod v v*

definition *psi′-l* **where**
  psi′-l l = (*alpha-l l*) $\cdot_v$ *α* − (*beta-l l*) $\cdot_v$ *β*

lemma *psi′-l-dim*:
  psi′-l l ∈ *carrier-vec N*
  ⟨*proof*⟩

definition *proj-psi′-l* **where**
  proj-psi′-l l = *proj* (*psi′-l l*)

lemma *proj-psi′-dim*:
  proj-psi′-l l ∈ *carrier-mat N N*

⟨*proof*⟩

lemma *psi-inner-psi′-l*:
inner-prod ψ (*psi′-l l*) = (*alpha-l l* ∗ *ccos* (ϑ / *2*) − *beta-l l* ∗ *csin* (ϑ / *2*))
⟨*proof*⟩

lemma *double-ccos-square*:
2 ∗ *ccos* (*a*::*real*) ∗ *ccos a* = *ccos* (*2* ∗ *a*) + *1*
⟨*proof*⟩

lemma *double-csin-square*:
2 ∗ *csin* (*a*::*real*) ∗ *csin a* = *1* − *ccos* (*2* ∗ *a*)
⟨*proof*⟩

lemma *csin-double*:
2 ∗ *csin* (*a*::*real*) ∗ *ccos a* = *csin*(*2* ∗ *a*)
⟨*proof*⟩

lemma *ccos-add*:
ccos (*x* + *y*) = *ccos x* ∗ *ccos y* − *csin x* ∗ *csin y*
⟨*proof*⟩

lemma *alpha-l-Suc-l-derive*:
2 ∗ (*alpha-l l* ∗ *ccos* (ϑ / *2*) − *beta-l l* ∗ *csin* (ϑ / *2*)) ∗ *ccos* (ϑ / *2*) − *alpha-l l* = *alpha-l* (*l* + *1*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *csin-add*:
csin (*x* + *y*) = *ccos x* ∗ *csin y* + *csin x* ∗ *ccos y*
⟨*proof*⟩

lemma *beta-l-Suc-l-derive*:
2 ∗ (*alpha-l l* ∗ *ccos* (ϑ / *2*) − (*beta-l l*) ∗ *csin* (ϑ / *2*)) ∗ *csin* (ϑ / *2*) + *beta-l l* = *beta-l* (*l* + *1*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *psi-l-Suc-l-derive*:
2 ∗ (*alpha-l l* ∗ *ccos* (ϑ / *2*) − *beta-l l* ∗ *csin* (ϑ / *2*)) $\cdot_v$ ψ − *psi′-l l* = *psi-l* (*l* + *1*)
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

7.2 Grover operator

Oracle O

definition *proj-O* :: *complex mat* **where**
proj-O = *mat N N* (λ(*i*, *j*). *if i* = *j then* (*if f i then 1 else 0*) *else 0*)

lemma *proj-O-dim*:
  $proj\text{-}O \in carrier\text{-}mat\ N\ N$
  ⟨*proof*⟩

lemma *proj-O-mult-alpha*:
  $proj\text{-}O *_v \alpha = zero\text{-}vec\ N$
  ⟨*proof*⟩

lemma *proj-O-mult-beta*:
  $proj\text{-}O *_v \beta = \beta$
  ⟨*proof*⟩

definition *mat-O* :: *complex mat* **where**
  $mat\text{-}O = mat\ N\ N\ (\lambda(i,j).\ if\ i = j\ then\ (if\ f\ i\ then\ -1\ else\ 1)\ else\ 0)$

lemma *mat-O-dim*:
  $mat\text{-}O \in carrier\text{-}mat\ N\ N$
  ⟨*proof*⟩

lemma *mat-O-mult-alpha*:
  $mat\text{-}O *_v \alpha = \alpha$
  ⟨*proof*⟩

lemma *mat-O-mult-beta*:
  $mat\text{-}O *_v \beta = -\ \beta$
  ⟨*proof*⟩

lemma *hermitian-mat-O*:
  hermitian mat-O
  ⟨*proof*⟩

lemma *unitary-mat-O*:
  unitary mat-O
⟨*proof*⟩

definition *mat-Ph* :: *complex mat* **where**
  $mat\text{-}Ph = mat\ N\ N\ (\lambda(i,j).\ if\ i = j\ then\ if\ i = 0\ then\ 1\ else\ -1\ else\ 0)$

lemma *hermitian-mat-Ph*:
  hermitian mat-Ph
  ⟨*proof*⟩

lemma *unitary-mat-Ph*:
  unitary mat-Ph
⟨*proof*⟩

definition *mat-G′* :: *complex mat* **where**
  $mat\text{-}G' = mat\ N\ N\ (\lambda(i,j).\ if\ i = j\ then\ 2\ /\ N\ -\ 1\ else\ 2\ /\ N)$

Geometrically, the Grover operator G is a rotation

definition *mat-G* :: *complex mat* **where**
mat-G = *mat-G′* ∗ *mat-O*

end

7.3 State of Grover's algorithm

The dimensions are [2, 2, ..., 2, n]. We work with a very special case as in the paper

locale *grover-state-sig* = *grover-state* + *state-sig* +
fixes *R* :: *nat*
fixes *K* :: *nat*
assumes *dims-def*: *dims* = *replicate n 2* @ [*K*]
assumes *R*: *R* = *pi* / (*2* ∗ ϑ) − *1* / *2*
assumes *K*: *K* > *R*

begin

lemma *K-gt-0*:
K > *0*
⟨*proof*⟩

Bits q0 to q_(n-1)

definition *vars1* :: *nat set* **where**
vars1 = {*0* ..< *n*}

Bit r

definition *vars2* :: *nat set* **where**
vars2 = {*n*}

lemma *length-dims*:
length dims = *n* + *1*
⟨*proof*⟩

lemma *dims-nth-lt-n*:
l < *n* ⟹ *nth dims l* = *2*
⟨*proof*⟩

lemma *nths-Suc-n-dims*:
nths dims {*0*..<(*Suc n*)} = *dims*
⟨*proof*⟩

interpretation *ps2-P*: *partial-state2 dims vars1 vars2*
⟨*proof*⟩

interpretation *ps-P*: *partial-state ps2-P.dims0 ps2-P.vars1′*⟨*proof*⟩

abbreviation *tensor-P* **where**
tensor-P A B ≡ *ps2-P.ptensor-mat A B*

lemma *tensor-P-dim*:
tensor-P A B ∈ *carrier-mat d d*
⟨*proof*⟩

lemma *dims-nths-le-n*:
assumes $l \leq n$
shows *nths dims* {*0..<l*} = *replicate l 2*
⟨*proof*⟩

lemma *dims-nths-one-lt-n*:
assumes $l < n$
shows *nths dims* {*l*} = [*2*]
⟨*proof*⟩

lemma *dims-vars1*:
nths dims vars1 = *replicate n 2*
⟨*proof*⟩

lemma *nths-rep-2-n*:
nths (*replicate n 2*) {*n*} = []
⟨*proof*⟩

lemma *dims-vars2*:
nths dims vars2 = [*K*]
⟨*proof*⟩

lemma *d-vars1*:
prod-list (*nths dims vars1*) = *N*
⟨*proof*⟩

lemma *ps2-P-dims0*:
ps2-P.dims0 = *dims*
⟨*proof*⟩

lemma *ps2-P-vars1'*:
ps2-P.vars1' = *vars1*
⟨*proof*⟩

lemma *ps2-P-d0*:
ps2-P.d0 = *d*
⟨*proof*⟩

lemma *ps2-P-d1*:
ps2-P.d1 = *N*
⟨*proof*⟩

lemma *ps2-P-d2*:
  ps2-P.d2 = K
  ⟨*proof*⟩

lemma *ps-P-d*:
  ps-P.d = d
  ⟨*proof*⟩

lemma *ps-P-d1*:
  ps-P.d1 = N
  ⟨*proof*⟩

lemma *ps-P-d2*:
  ps-P.d2 = K
  ⟨*proof*⟩

lemma *nths-uminus-vars1*:
  nths dims (− *vars1*) = *nths dims vars2*
  ⟨*proof*⟩

lemma *tensor-P-mult*:
  assumes *m1* ∈ *carrier-mat* (*2*^*n*) (*2*^*n*)
    and *m2* ∈ *carrier-mat* (*2*^*n*) (*2*^*n*)
    and *m3* ∈ *carrier-mat K K*
    and *m4* ∈ *carrier-mat K K*
  shows (*tensor-P m1 m3*) ∗ (*tensor-P m2 m4*) = *tensor-P* (*m1* ∗ *m2*) (*m3* ∗ *m4*)
⟨*proof*⟩

lemma *mat-ext-vars1*:
  shows *mat-extension dims vars1 A* = *tensor-P A* (1_m *K*)
  ⟨*proof*⟩

lemma *Utrans-P-is-tensor-P1*:
  Utrans-P vars1 A = *Utrans* (*tensor-P A* (1_m *K*))
  ⟨*proof*⟩

lemma *nths-dims-uminus-vars2*:
  nths dims (−*vars2*) = *nths dims vars1*
⟨*proof*⟩

lemma *mat-ext-vars2*:
  assumes *A* ∈ *carrier-mat K K*
  shows *mat-extension dims vars2 A* = *tensor-P* (1_m *N*) *A*
⟨*proof*⟩

lemma *Utrans-P-is-tensor-P2*:
  assumes *A* ∈ *carrier-mat K K*
  shows *Utrans-P vars2 A* = *Utrans* (*tensor-P* (1_m *N*) *A*)

⟨*proof*⟩

7.4 Grover's algorithm

Apply hadamard operator to first n variables

definition *hadamard-on-i* :: *nat* ⇒ *complex mat* **where**
hadamard-on-i i = *pmat-extension dims* {*i*} (*vars1* − {*i*}) *hadamard*
declare *hadamard-on-i-def* [*simp*]

fun *hadamard-n* :: *nat* ⇒ *com* **where**
hadamard-n 0 = *SKIP*
| *hadamard-n* (*Suc i*) = *hadamard-n i* ;; *Utrans* (*tensor-P* (*hadamard-on-i i*) (1_m *K*))

Body of the loop

definition *D* :: *com* **where**
D = *Utrans-P vars1 mat-O* ;;
hadamard-n n ;;
Utrans-P vars1 mat-Ph ;;
hadamard-n n ;;
Utrans-P vars2 (*mat-incr K*)

lemma *unitary-ex-mat-O*:
unitary (*tensor-P mat-O* (1_m *K*))
⟨*proof*⟩

lemma *unitary-ex-mat-Ph*:
unitary (*tensor-P mat-Ph* (1_m *K*))
⟨*proof*⟩

lemma *unitary-hadamard-on-i*:
assumes $k < n$
shows *unitary* (*hadamard-on-i k*)
⟨*proof*⟩

lemma *unitary-exhadamard-on-i*:
assumes $k < n$
shows *unitary* (*tensor-P* (*hadamard-on-i k*) (1_m *K*))
⟨*proof*⟩

lemma *hadamard-on-i-dim*:
assumes $k < n$
shows *hadamard-on-i k* ∈ *carrier-mat N N*
⟨*proof*⟩

lemma *well-com-hadamard-k*:
$k \leq n \Longrightarrow$ *well-com* (*hadamard-n k*)
⟨*proof*⟩

lemma *well-com-hadamard-n*:
well-com (*hadamard-n n*)
⟨*proof*⟩

lemma *well-com-mat-O*:
well-com (*Utrans-P vars1 mat-O*)
⟨*proof*⟩

lemma *well-com-mat-Ph*:
well-com (*Utrans-P vars1 mat-Ph*)
⟨*proof*⟩

lemma *unitary-exmat-incr*:
unitary (*tensor-P* (1_m *N*) (*mat-incr K*))
⟨*proof*⟩

lemma *well-com-mat-incr*:
well-com (*Utrans-P vars2* (*mat-incr K*))
⟨*proof*⟩

lemma *well-com-D*: *well-com D*
⟨*proof*⟩

Test at while loop

definition *M0* :: *complex mat* **where**
M0 = *mat K K* (λ(*i*,*j*). *if i* = *j* ∧ *i* ≥ *R then 1 else 0*)

lemma *hermitian-M0*:
hermitian M0
⟨*proof*⟩

lemma *M0-dim*:
M0 ∈ *carrier-mat K K*
⟨*proof*⟩

lemma *M0-mult-M0*:
M0 ∗ *M0* = *M0*
⟨*proof*⟩

definition *M1* :: *complex mat* **where**
M1 = *mat K K* (λ(*i*,*j*). *if i* = *j* ∧ *i* < *R then 1 else 0*)

lemma *M1-dim*:
M1 ∈ *carrier-mat K K*
⟨*proof*⟩

lemma *hermitian-M1*:
hermitian M1
⟨*proof*⟩

lemma *M1-mult-M1*:
$M1 * M1 = M1$
⟨*proof*⟩

lemma *M1-add-M0*:
$M1 + M0 = 1_m\ K$
⟨*proof*⟩

Test at the end

definition *testN* :: *nat* ⇒ *complex mat* **where**
$testN\ k = mat\ N\ N\ (\lambda(i,j).\ if\ i = k \wedge j = k\ then\ 1\ else\ 0)$

lemma *hermitian-testN*:
hermitian (*testN k*)
⟨*proof*⟩

lemma *testN-mult-testN*:
$testN\ k * testN\ k = testN\ k$
⟨*proof*⟩

lemma *testN-dim*:
$testN\ k \in carrier\text{-}mat\ N\ N$
⟨*proof*⟩

definition *test-fst-k* :: *nat* ⇒ *complex mat* **where**
$test\text{-}fst\text{-}k\ k = mat\ N\ N\ (\lambda(i, j).\ if\ (i = j \wedge i < k)\ then\ 1\ else\ 0)$

lemma *sum-test-k*:
assumes $m \leq N$
shows $matrix\text{-}sum\ N\ (\lambda k.\ testN\ k)\ m = test\text{-}fst\text{-}k\ m$
⟨*proof*⟩

lemma *test-fst-kN*:
$test\text{-}fst\text{-}k\ N = 1_m\ N$
⟨*proof*⟩

lemma *matrix-sum-tensor-P1*:
$(\bigwedge k.\ k < m \Longrightarrow g\ k \in carrier\text{-}mat\ N\ N) \Longrightarrow (A \in carrier\text{-}mat\ K\ K) \Longrightarrow$
$matrix\text{-}sum\ d\ (\lambda k.\ tensor\text{-}P\ (g\ k)\ A)\ m = tensor\text{-}P\ (matrix\text{-}sum\ N\ g\ m)\ A$
⟨*proof*⟩

Grover's algorithm. Assume we start in the zero state

definition *Grover* :: *com* **where**
Grover = *hadamard-n n* ;;
While-P vars2 M0 M1 D ;;
Measure-P vars1 N testN (*replicate N SKIP*)

lemma *well-com-if*:

well-com (*Measure-P vars1 N testN* (*replicate N SKIP*))
⟨*proof*⟩

lemma *well-com-while*:
well-com (*While-P vars2 M0 M1 D*)
⟨*proof*⟩

lemma *well-com-Grover*:
well-com Grover
⟨*proof*⟩

7.5 Correctness

Pre-condition: assume in the zero state

definition *ket-pre* :: *complex vec* **where**
ket-pre = *Matrix.vec N* (λ*k*. *if k* = *0 then 1 else 0*)

lemma *ket-pre-dim*:
ket-pre ∈ *carrier-vec N* ⟨*proof*⟩

definition *pre* :: *complex mat* **where**
pre = *proj ket-pre*

lemma *pre-dim*:
pre ∈ *carrier-mat N N*
⟨*proof*⟩

lemma *norm-pre*:
inner-prod ket-pre ket-pre = *1*
⟨*proof*⟩

lemma *pre-trace*:
trace pre = *1*
⟨*proof*⟩

lemma *positive-pre*:
positive pre
⟨*proof*⟩

lemma *pre-le-one*:
pre $\leq_L$ 1_m *N*
⟨*proof*⟩

Post-condition: should be in a state i with f i = 1

definition *post* :: *complex mat* **where**
post = *mat N N* (λ(*i*, *j*). *if* (*i* = *j* ∧ *f i*) *then 1 else 0*)

lemma *post-dim*:
post ∈ *carrier-mat N N*

⟨proof⟩

lemma *hermitian-post*:
hermitian post
⟨proof⟩

Hoare triples of initialization

definition *ket-zero* :: *complex vec* **where**
ket-zero = *Matrix.vec 2* (λ*k. if k* = *0 then 1 else 0*)

lemma *ket-zero-dim*:
ket-zero ∈ *carrier-vec 2* ⟨proof⟩

definition *proj-zero* **where**
proj-zero = *proj ket-zero*

definition *ket-one* **where**
ket-one = *Matrix.vec 2* (λ*k. if k* = *1 then 1 else 0*)

definition *proj-one* **where**
proj-one = *proj ket-one*

definition *ket-plus* **where**
ket-plus = *Matrix.vec 2* (λ*k.1* / *csqrt 2*)

lemma *ket-plus-dim*:
ket-plus ∈ *carrier-vec 2* ⟨proof⟩

lemma *ket-plus-eval* [*simp*]:
i < *2* ⟹ *ket-plus* $ *i* = *1* / *csqrt 2*
⟨proof⟩

lemma *csqrt-2-sq* [*simp*]:
complex-of-real (*sqrt 2*) ∗ *complex-of-real* (*sqrt 2*) = *2*
⟨proof⟩

lemma *ket-plus-tensor-n*:
partial-state.tensor-vec [*2*, *2*] {*0*} *ket-plus ket-plus* = *Matrix.vec 4* (λ*k. 1* / *2*)
⟨proof⟩

definition *proj-plus* **where**
proj-plus = *proj ket-plus*

lemma *hadamard-on-zero*:
hadamard $*_v$ *ket-zero* = *ket-plus*
⟨proof⟩

fun *exH-k* :: *nat* ⇒ *complex mat* **where**
exH-k 0 = *hadamard-on-i 0*

| *exH-k* (*Suc k*) = *exH-k k* ∗ *hadamard-on-i* (*Suc k*)

fun *H-k* :: *nat* ⇒ *complex mat* **where**
H-k 0 = *hadamard*
| *H-k* (*Suc k*) = *ptensor-mat dims* {*0*..<*Suc k*} {*Suc k*} (*H-k k*) *hadamard*

lemma *H-k-dim*:
$k < n \Longrightarrow$ *H-k k* ∈ *carrier-mat* (*2*^(*Suc k*)) (*2*^(*Suc k*))
⟨*proof*⟩

lemma *exH-k-eq-H-k*:
$k < n \Longrightarrow$ *exH-k k* = *pmat-extension dims* {*0*..<(*Suc k*)} {(*Suc k*)..<*n*} (*H-k k*)
⟨*proof*⟩

lemma *mult-exH-k-left*:
assumes *Suc k* < *n*
shows *hadamard-on-i* (*Suc k*) ∗ *exH-k k* = *exH-k* (*Suc k*)
⟨*proof*⟩

lemma *exH-eq-H*:
exH-k (*n* − *1*) = *H-k* (*n* − *1*)
⟨*proof*⟩

fun *ket-zero-k* :: *nat* ⇒ *complex vec* **where**
ket-zero-k 0 = *ket-zero*
| *ket-zero-k* (*Suc k*) = *ptensor-vec dims* {*0*..<(*Suc k*)} {*Suc k*} (*ket-zero-k k*) *ket-zero*

lemma *ket-zero-k-dim*:
assumes *k* < *n*
shows *ket-zero-k k* ∈ *carrier-vec* (*2*^(*Suc k*))
⟨*proof*⟩

fun *ket-plus-k* **where**
ket-plus-k 0 = *ket-plus*
| *ket-plus-k* (*Suc k*) = *ptensor-vec dims* {*0*..<(*Suc k*)} {*Suc k*} (*ket-plus-k k*) *ket-plus*

lemma *ket-plus-k-dim*:
assumes *k* < *n*
shows *ket-plus-k k* ∈ *carrier-vec* (*2*^(*Suc k*))
⟨*proof*⟩

lemma *H-k-ket-zero-k*:
$k < n \Longrightarrow$ (*H-k k*) $*_v$ (*ket-zero-k k*) = (*ket-plus-k k*)
⟨*proof*⟩

lemma *encode1-replicate-2*:
partial-state.encode1 (*replicate* (*Suc k*) *2*) $\{0..<k\}$ *i* = *i mod* (*2* ^ *k*)
⟨*proof*⟩

lemma *encode2-replicate-2*:
assumes *i* < *2* ^ *Suc k*
shows *partial-state.encode2* (*replicate* (*Suc k*) *2*) $\{0..<k\}$ *i* = *i div* (*2* ^ *k*)
⟨*proof*⟩

lemma *ket-zero-k-decode*:
$k < n \Longrightarrow$ *ket-zero-k k* = *Matrix.vec* (*2*^(*Suc k*)) (λ*k*. *if k* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *ket-plus-k-decode*:
$k < n \Longrightarrow$ *ket-plus-k k* = *Matrix.vec* (*2*^(*Suc k*)) (λ*l*. *1* / *csqrt* (*2*^(*Suc k*)))
⟨*proof*⟩

lemma *exH-k-mult-pre-is-psi*:
exH-k (*n* − *1*) $*_v$ *ket-pre* = ψ
⟨*proof*⟩

definition *ket-k* :: *nat* ⇒ *complex vec* **where**
ket-k x = *Matrix.vec K* (λ*k*. *if k* = *x then 1 else 0*)

lemma *ket-k-dim*:
ket-k k ∈ *carrier-vec K*
⟨*proof*⟩

lemma *mat-incr-mult-ket-k*:
$k < K \Longrightarrow$ (*mat-incr K*) $*_v$ (*ket-k k*) = (*ket-k* ((*k* + *1*) *mod K*))
⟨*proof*⟩

definition *proj-k* **where**
proj-k x = *proj* (*ket-k x*)

lemma *proj-k-dim*:
proj-k k ∈ *carrier-mat K K*
⟨*proof*⟩

lemma *norm-ket-k-lt-K*:
$k < K \Longrightarrow$ *inner-prod* (*ket-k k*) (*ket-k k*) = *1*
⟨*proof*⟩

lemma *norm-ket-k-ge-K*:
$k \geq K \Longrightarrow$ *inner-prod* (*ket-k k*) (*ket-k k*) = *0*
⟨*proof*⟩

lemma *norm-ket-k*:
inner-prod (*ket-k k*) (*ket-k k*) ≤ *1*

⟨*proof*⟩

lemma *proj-k-mat*:
assumes $k < K$
shows *proj-k* k = *mat* K K ($\lambda(i, j)$. *if* ($i = j \wedge i = k$) *then 1 else 0*)
⟨*proof*⟩

lemma *positive-proj-k*:
positive (*proj-k* k)
⟨*proof*⟩

lemma *proj-k-le-one*:
(*proj-k* k) $\leq_L$ 1_m K
⟨*proof*⟩

definition *proj-psi* **where**
proj-psi = *proj* ψ

lemma *proj-psi-dim*:
proj-psi $\in$ *carrier-mat* N N
⟨*proof*⟩

lemma *norm-psi*:
inner-prod ψ ψ = *1*
⟨*proof*⟩

lemma *proj-psi-mat*:
proj-psi = *mat* N N (λk. *1* / N)
⟨*proof*⟩

lemma *hermitian-proj-psi*:
hermitian proj-psi
⟨*proof*⟩

lemma *hermitian-exproj-psi*:
hermitian (*tensor-P proj-psi* (1_m K))
⟨*proof*⟩

lemma *proj-psi-is-projection*:
proj-psi * *proj-psi* = *proj-psi*
⟨*proof*⟩

lemma *proj-psi-trace*:
trace (*proj-psi*) = *1*
⟨*proof*⟩

lemma *positive-proj-psi*:
positive (*proj-psi*)
⟨*proof*⟩

lemma *proj-psi-le-one*:
$(proj\text{-}psi) \leq_L 1_m\ N$
⟨*proof*⟩

lemma *hermitian-hadamard-on-k*:
assumes $k < n$
shows *hermitian* (*hadamard-on-i k*)
⟨*proof*⟩

lemma *hermitian-H-k*:
$k < n \Longrightarrow$ *hermitian* (*H-k k*)
⟨*proof*⟩

lemma *unitary-H-k*:
$k < n \Longrightarrow$ *unitary* (*H-k k*)
⟨*proof*⟩

lemma *exH-k-dim*:
shows $k < n \Longrightarrow$ *exH-k k* $\in$ *carrier-mat N N*
⟨*proof*⟩

lemma *exH-n-dim*:
shows *exH-k* $(n - 1) \in$ *carrier-mat N N*
⟨*proof*⟩

lemma *unitary-exH-k*:
shows $k < n \Longrightarrow$ *unitary* (*exH-k k*)
⟨*proof*⟩

lemma *hermitian-exH-n*:
hermitian (*exH-k* $(n - 1)$)
⟨*proof*⟩

lemma *exH-k-mult-psi-is-pre*:
exH-k $(n - 1) *_v \psi =$ *ket-pre*
⟨*proof*⟩

fun *exexH-k* :: *nat* ⇒ *complex mat* **where**
exexH-k k = *tensor-P* (*exH-k k*) $(1_m\ K)$

lemma *unitary-exexH-k*:
$k < n \Longrightarrow$ *unitary* (*exexH-k k*)
⟨*proof*⟩

lemma *exexH-k-dim*:
$k < n \Longrightarrow$ *exexH-k k* $\in$ *carrier-mat d d*
⟨*proof*⟩

lemma *hoare-seq-utrans*:
fixes P :: *complex mat*
assumes *unitary U1* **and** *unitary U2* **and** *is-quantum-predicate P*
and *dU1*: $U1 \in$ *carrier-mat d d* **and** *dU2*: $U2 \in$ *carrier-mat d d*
shows
$\vdash_p$
$\{$*adjoint* $(U2 * U1) * P * (U2 * U1)\}$
Utrans U1;; *Utrans U2*
$\{P\}$
⟨*proof*⟩

lemma *qp-close-after-exexH-k*:
fixes P :: *complex mat*
assumes *is-quantum-predicate P*
shows $k < n \Longrightarrow$ *is-quantum-predicate* (*adjoint* (*exexH-k k*) $*$ P $*$ *exexH-k k*)
⟨*proof*⟩

lemma *hoare-hadamard-n*:
fixes P :: *complex mat*
shows *is-quantum-predicate P* $\Longrightarrow k < n \Longrightarrow$
$\vdash_p$
$\{$*adjoint* (*exexH-k k*) $*$ P $*$ *exexH-k k*$\}$
hadamard-n (*Suc k*)
$\{P\}$
⟨*proof*⟩

lemma *qp-pre*:
is-quantum-predicate (*tensor-P pre* (*proj-k 0*))
⟨*proof*⟩

lemma *qp-init-post*:
is-quantum-predicate (*tensor-P proj-psi* (*proj-k 0*))
⟨*proof*⟩

lemma *tensor-P-adjoint-left-right*:
assumes $m1 \in$ *carrier-mat N N* **and** $m2 \in$ *carrier-mat K K* **and** $m3 \in$ *carrier-mat N N* **and** $m4 \in$ *carrier-mat K K*
shows *adjoint* (*tensor-P m1 m2*) $*$ *tensor-P m3 m4* $*$ *tensor-P m1 m2* $=$ *tensor-P* (*adjoint m1* $*$ *m3* $*$ *m1*) (*adjoint m2* $*$ *m4* $*$ *m2*)
⟨*proof*⟩

abbreviation *exH-n* **where**
exH-n $\equiv$ *exH-k* $(n - 1)$

lemma *hoare-triple-init*:
$\vdash_p$
$\{$*tensor-P pre* (*proj-k 0*)$\}$
hadamard-n n
$\{$*tensor-P proj-psi* (*proj-k 0*)$\}$

⟨*proof*⟩

Hoare triples of while loop

definition *proj-psi-l* **where**
proj-psi-l l = *proj* (*psi-l l*)

lemma *positive-psi-l*:
$k < K \Longrightarrow$ *positive* (*proj-psi-l k*)
⟨*proof*⟩

lemma *hermitian-proj-psi-l*:
$k < K \Longrightarrow$ *hermitian* (*proj-psi-l k*)
⟨*proof*⟩

definition P' **where**
P' = *tensor-P* (*proj-psi-l R*) (*proj-k R*)

lemma *proj-psi-l-dim*:
proj-psi-l l ∈ *carrier-mat N N*
⟨*proof*⟩

definition Q :: *complex mat* **where**
Q = *matrix-sum d* (λ*l*. *tensor-P* (*proj-psi-l l*) (*proj-k l*)) *R*

lemma *psi-l-le-id*:
shows *proj-psi-l l* $\leq_L$ 1_m *N*
⟨*proof*⟩

lemma *positive-proj-psi-l*:
shows *positive* (*proj-psi-l l*)
⟨*proof*⟩

definition *proj-fst-k* :: *nat* ⇒ *complex mat* **where**
proj-fst-k k = *mat K K* (λ(*i*, *j*). *if* ($i = j \wedge i < k$) *then 1 else 0*)

lemma *hermitian-proj-fst-k*:
adjoint (*proj-fst-k k*) = *proj-fst-k k*
⟨*proof*⟩

lemma *proj-fst-k-is-projection*:
proj-fst-k k ∗ *proj-fst-k k* = *proj-fst-k k*
⟨*proof*⟩

lemma *positive-proj-fst-k*:
positive (*proj-fst-k k*)
⟨*proof*⟩

lemma *proj-fst-k-le-one*:
proj-fst-k k $\leq_L$ 1_m *K*

⟨*proof*⟩

lemma *sum-proj-k*:
assumes $m \le K$
shows *matrix-sum K* $(\lambda k.$ *proj-k k*$)$ $m =$ *proj-fst-k m*
⟨*proof*⟩

lemma *proj-psi-proj-k-le-exproj-k*:
shows *tensor-P* (*proj-psi-l k*) (*proj-k l*) $\le_L$ *tensor-P* (1_m *N*) (*proj-k l*)
⟨*proof*⟩

definition *Q1* :: *complex mat* **where**
Q1 = *matrix-sum d* ($\lambda l.$ *tensor-P* (*proj-psi'-l l*) (*proj-k l*)) *R*

lemma *tensor-P-left-right-partial1*:
assumes *m1* ∈ *carrier-mat N N* **and** *m2* ∈ *carrier-mat N N* **and** *m3* ∈ *carrier-mat K K* **and** *m4* ∈ *carrier-mat N N*
shows *tensor-P m1* (1_m *K*) ∗ *tensor-P m2 m3* ∗ *tensor-P m4* (1_m *K*) = *tensor-P* (*m1* ∗ *m2* ∗ *m4*) *m3*
⟨*proof*⟩

lemma *tensor-P-left-right-partial2*:
assumes *m1* ∈ *carrier-mat K K* **and** *m2* ∈ *carrier-mat K K* **and** *m3* ∈ *carrier-mat N N* **and** *m4* ∈ *carrier-mat K K*
shows *tensor-P* (1_m *N*) *m1* ∗ *tensor-P m3 m2* ∗ *tensor-P* (1_m *N*) *m4* = *tensor-P m3* (*m1* ∗ *m2* ∗ *m4*)
⟨*proof*⟩

lemma *matrix-sum-mult-left-right*:
fixes *A B* :: *complex mat*
assumes *dg*: ($\bigwedge k.\ k < l \Longrightarrow g\ k \in$ *carrier-mat m m*)
and *dA*: *A* ∈ *carrier-mat m m* **and** *dB*: *B* ∈ *carrier-mat m m*
shows *matrix-sum m* ($\lambda k.$ *A* ∗ *g k* ∗ *B*) *l* = *A* ∗ *matrix-sum m g l* ∗ *B*
⟨*proof*⟩

lemma *mat-O-split*:
mat-O = 1_m *N* − *2* $\cdot_m$ *proj-O*
⟨*proof*⟩

lemma *mat-O-mult-psi'-l*:
mat-O $*_v$ (*psi'-l l*) = *psi-l l*
⟨*proof*⟩

lemma *mat-O-times-Q1*:
adjoint (*tensor-P mat-O* (1_m *K*)) ∗ *Q1* ∗ (*tensor-P mat-O* (1_m *K*)) = *Q*
⟨*proof*⟩

definition *Q2* **where**
Q2 = *matrix-sum d* ($\lambda l.$ *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k l*)) *R*

lemma *Q2-dim*:
Q2 ∈ *carrier-mat d d*
⟨*proof*⟩

lemma *Q2-le-one*:
Q2 $\leq_L$ *1*$_m$ *d*
⟨*proof*⟩

lemma *qp-Q2*:
is-quantum-predicate Q2
⟨*proof*⟩

lemma *pre-mat*:
pre = *mat N N* (λ(*i*, *j*). *if i* = *j* ∧ *i* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *mat-Ph-split*:
mat-Ph = *2* $\cdot_m$ *pre* − *1*$_m$ *N*
⟨*proof*⟩

lemma *H-Ph-H*:
exexH-k (*n*−*1*) ∗ *tensor-P mat-Ph* (*1*$_m$ *K*) ∗ *exexH-k* (*n* − *1*) = *2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*
⟨*proof*⟩

lemma *hermitian-proj-psi-minus-1*:
hermitian (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*)
⟨*proof*⟩

lemma *unitary-proj-psi-minus-1*:
unitary (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*)
⟨*proof*⟩

lemma *proj-psi-minus-1-mult-psi′-l*:
(*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) $*_v$ *psi′-l l* = *psi-l* (*l* + *1*)
⟨*proof*⟩

lemma *proj-psi-minus-1-mult-psi-Suc-l*:
(*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) $*_v$ *psi-l* (*l* + *1*) = *psi′-l l*
⟨*proof*⟩

lemma *exproj-psi-minus-1-tensor*:
(*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*)) − *1*$_m$ *d* = *tensor-P* (*2* $\cdot_m$ *proj-psi* − (*1*$_m$ *N*)) (*1*$_m$ *K*)
⟨*proof*⟩

lemma *unitary-exproj-psi-minus-1*:
unitary (*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*)

⟨*proof*⟩

lemma *proj-psi-minus-1-Q2*:
$adjoint\ (2 \cdot_m tensor\text{-}P\ proj\text{-}psi\ (1_m\ K) - 1_m\ d) * Q2 * (2 \cdot_m tensor\text{-}P\ proj\text{-}psi\ (1_m\ K) - 1_m\ d) = Q1$
⟨*proof*⟩

lemma *qp-Q1*:
is-quantum-predicate Q1
⟨*proof*⟩

lemma *qp-Q*:
is-quantum-predicate Q
⟨*proof*⟩

lemma *hoare-triple-D1*:
$\vdash_p$
{*Q*}
Utrans-P vars1 mat-O
{*Q1*}
⟨*proof*⟩

lemma *hoare-triple-D2*:
$\vdash_p$
{*Q1*}
hadamard-n n ;;
Utrans-P vars1 mat-Ph ;;
hadamard-n n
{*Q2*}
⟨*proof*⟩

definition *exM0* **where**
$exM0 = tensor\text{-}P\ (1_m\ N)\ M0$

lemma *M0-mult-ket-k-R*:
$M0 *_v ket\text{-}k\ R = ket\text{-}k\ R$
⟨*proof*⟩

lemma *exP0-P′*:
$adjoint\ exM0 * P' * exM0 = P'$
⟨*proof*⟩

definition *exM1* **where**
$exM1 = tensor\text{-}P\ (1_m\ N)\ M1$

lemma *M1-mult-ket-k*:
assumes $k < R$
shows $M1 *_v ket\text{-}k\ k = ket\text{-}k\ k$
⟨*proof*⟩

lemma *exP1-Q*:
adjoint exM1 $*$ *Q* $*$ *exM1* = *Q*
⟨*proof*⟩

lemma *qp-P'*:
is-quantum-predicate P'
⟨*proof*⟩

lemma *P'-add-Q*:
P' + *Q* = *matrix-sum d* (λ*l. tensor-P* (*proj-psi-l l*) (*proj-k l*)) (*R* + *1*)
⟨*proof*⟩

lemma *positive-Qk*:
positive (*tensor-P* (*proj-psi-l l*) (*proj-k l*))
⟨*proof*⟩

lemma *P'-Q-dim*:
P' + *Q* ∈ *carrier-mat d d*
⟨*proof*⟩

lemma *P'-add-Q-le-one*:
P' + *Q* $\leq_L$ 1_m *d*
⟨*proof*⟩

lemma *qp-P'-Q*:
is-quantum-predicate (*P'* + *Q*)
⟨*proof*⟩

lemma *Q2-leq-lemma*:
tensor-P (1_m *N*) (*mat-incr K*) $*$ *Q2* $*$ *adjoint* (*tensor-P* (1_m *N*) (*mat-incr K*))
$\leq_L$ *P'* + *Q*
⟨*proof*⟩

lemma *Q2-leq*:
Q2 $\leq_L$ *adjoint* (*tensor-P* (1_m *N*) (*mat-incr K*)) $*$ (*P'* + *Q*) $*$ *tensor-P* (1_m *N*)
(*mat-incr K*)
⟨*proof*⟩

lemma *hoare-triple-D3*:
$\vdash_p$
{*Q2*}
Utrans-P vars2 (*mat-incr K*)
{*adjoint exM0* $*$ *P'* $*$ *exM0* + *adjoint exM1* $*$ *Q* $*$ *exM1*}
⟨*proof*⟩

lemma *qp-D3-post*:
is-quantum-predicate (*adjoint exM0* $*$ *P'* $*$ *exM0* + *adjoint exM1* $*$ *Q* $*$ *exM1*)
⟨*proof*⟩

lemma *hoare-triple-D*:
$\vdash_p$
$\{Q\}$
D
$\{adjoint\ exM0 * P' * exM0 + adjoint\ exM1 * Q * exM1\}$
⟨*proof*⟩

lemma *psi-is-psi-l0*:
$\psi = psi\text{-}l\ 0$
⟨*proof*⟩

lemma *proj-psi-is-proj-psi-l0*:
$proj\text{-}psi = proj\text{-}psi\text{-}l\ 0$
⟨*proof*⟩

lemma *lowner-le-Q*:
$tensor\text{-}P\ proj\text{-}psi\ (proj\text{-}k\ 0) \le_L adjoint\ exM0 * P' * exM0 + adjoint\ exM1 * Q * exM1$
⟨*proof*⟩

lemma *hoare-triple-while*:
$\vdash_p$
$\{adjoint\ exM0 * P' * exM0 + adjoint\ exM1 * Q * exM1\}$
While-P vars2 M0 M1 D
$\{P'\}$
⟨*proof*⟩

lemma *R-and-a-half-ϑ*:
$(R + 1/2) * \vartheta = pi\ /\ 2$
⟨*proof*⟩

lemma *psi-lR-is-beta*:
$psi\text{-}l\ R = \beta$
⟨*proof*⟩

lemma *post-mult-beta*:
$post *_v \beta = \beta$
⟨*proof*⟩

lemma *post-mult-post*:
$post * post = post$
⟨*proof*⟩

lemma *post-mult-proj-psi-lR*:
$post * proj\text{-}psi\text{-}l\ R = proj\text{-}psi\text{-}l\ R$
⟨*proof*⟩

lemma *proj-psi-lR-mult-post*:

proj-psi-l R $*$ *post* = *proj-psi-l R*
⟨*proof*⟩

lemma *proj-psi-lR-mult-proj-psi-lR*:
proj-psi-l R $*$ *proj-psi-l R* = *proj-psi-l R*
⟨*proof*⟩

lemma *proj-psi-lR-le-post*:
proj-psi-l R $\leq_L$ *post*
⟨*proof*⟩

lemma *P′-le-post-R*:
$P' \leq_L$ (*tensor-P post* (*proj-k R*))
⟨*proof*⟩

lemma *positive-post*:
positive post
⟨*proof*⟩

lemma *lowner-le-P′*:
$P' \leq_L$ *tensor-P post* (1_m *K*)
⟨*proof*⟩

lemma *post-mult-testNk*:
assumes *f k*
shows *post* $*$ (*testN k*) = *testN k*
⟨*proof*⟩

lemma *post-mult-testNk-neg*:
assumes ¬ *f k*
shows *post* $*$ *testN k* = 0_m *N N*
⟨*proof*⟩

lemma *testN-post1*:
f k ⟹ *adjoint* (*testN k*) $*$ *post* $*$ *testN k* = *testN k*
⟨*proof*⟩

lemma *testN-post2*:
¬ *f k* ⟹ *adjoint* (*testN k*) $*$ *post* $*$ *testN k* = 0_m *N N*
⟨*proof*⟩

definition *post-fst-k* :: *nat* ⇒ *complex mat* **where**
post-fst-k k = *mat N N* (λ(*i*, *j*). *if* (*i* = *j* ∧ *f i* ∧ *i* < *k*) *then 1 else 0*)

lemma *post-fst-kN*:
post-fst-k N = *post*
⟨*proof*⟩

lemma *post-fst-k-Suc*:

$f\ i \Longrightarrow post\text{-}fst\text{-}k\ (Suc\ i) = testN\ i + post\text{-}fst\text{-}k\ i$
⟨*proof*⟩

lemma *post-fst-k-Suc-neg*:
$\neg\ f\ i \Longrightarrow post\text{-}fst\text{-}k\ (Suc\ i) = post\text{-}fst\text{-}k\ i$
⟨*proof*⟩

lemma *testN-sum*:
$matrix\text{-}sum\ N\ (\lambda k.\ adjoint\ (testN\ k) * post * testN\ k)\ N = post$
⟨*proof*⟩

lemma *tensor-P-testN-sum*:
$matrix\text{-}sum\ d\ (\lambda k.\ adjoint\ (tensor\text{-}P\ (testN\ k)\ (1_m\ K)) * tensor\text{-}P\ post\ (1_m\ K)$
$*\ tensor\text{-}P\ (testN\ k)\ (1_m\ K))\ N =$
$tensor\text{-}P\ post\ (1_m\ K)$
⟨*proof*⟩

lemma *post-le-one*:
$post \leq_L 1_m\ N$
⟨*proof*⟩

lemma *qp-post*:
$is\text{-}quantum\text{-}predicate\ (tensor\text{-}P\ post\ (1_m\ K))$
⟨*proof*⟩

lemma *hoare-triple-if*:
$\vdash_p$
$\{tensor\text{-}P\ post\ (1_m\ K)\}$
$Measure\text{-}P\ vars1\ N\ testN\ (replicate\ N\ SKIP)$
$\{tensor\text{-}P\ post\ (1_m\ K)\}$
⟨*proof*⟩

theorem *grover-partial-deduct*:
$\vdash_p$
$\{tensor\text{-}P\ pre\ (proj\text{-}k\ 0)\}$
Grover
$\{tensor\text{-}P\ post\ (1_m\ K)\}$
⟨*proof*⟩

theorem *grover-partial-correct*:
$\models_p$
$\{tensor\text{-}P\ pre\ (proj\text{-}k\ 0)\}$
Grover
$\{tensor\text{-}P\ post\ (1_m\ K)\}$
⟨*proof*⟩
end

end

References

[1] M. Ying. Floyd–Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems*, 33(6):19:1–19:49, 2011.