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

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*
by (*simp add*: *trace-def*)

lemma *trace-id* [*simp*]:
trace (1_m *n*) = *n*
by (*simp add*: *trace-def*)

lemma *trace-comm*:
fixes *A B* :: *'a::comm-ring mat*
assumes *A*: *A* ∈ *carrier-mat n n* **and** *B*: *B* ∈ *carrier-mat n n*
shows *trace* (*A* ∗ *B*) = *trace* (*B* ∗ *A*)
proof (*simp add*: *trace-def*)
have (∑ *i* = *0*..<*n*. (*A* ∗ *B*) $$ (*i*, *i*)) = (∑ *i* = *0*..<*n*. ∑ *j* = *0*..<*n*. *A* $$ (*i*,*j*) ∗ *B* $$ (*j*,*i*))
apply (*rule sum.cong*) **using** *assms* **by** (*auto simp add*: *scalar-prod-def*)
also have ... = (∑ *j* = *0*..<*n*. ∑ *i* = *0*..<*n*. *A* $$ (*i*,*j*) ∗ *B* $$ (*j*,*i*))
by (*rule sum.swap*)
also have ... = (∑ *j* = *0*..<*n*. *col A j* · *row B j*)
by (*metis* (*no-types*, *lifting*) *A B atLeastLessThan-iff carrier-matD index-col index-row scalar-prod-def sum.cong*)
also have ... = (∑ *j* = *0*..<*n*. *row B j* · *col A j*)
apply (*rule sum.cong*) **apply** *auto*
apply (*subst comm-scalar-prod*[**where** *n*=*n*]) **apply** *auto*
using *assms* **by** *auto*
also have ... = (∑ *j* = *0*..<*n*. (*B* ∗ *A*) $$ (*j*, *j*))
apply (*rule sum.cong*) **using** *assms* **by** *auto*
finally show (∑ *i* = *0*..<*dim-row A*. (*A* ∗ *B*) $$ (*i*, *i*)) = (∑ *i* = *0*..<*dim-row B*. (*B* ∗ *A*) $$ (*i*, *i*))
using *A B* **by** *auto*
qed

lemma *trace-add-linear*:
fixes *A B* :: *'a::comm-ring mat*
assumes *A*: *A* ∈ *carrier-mat n n* **and** *B*: *B* ∈ *carrier-mat n n*
shows *trace* (*A* + *B*) = *trace A* + *trace B* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (∑ *i*=*0*..<*n*. *A*$$(*i*, *i*) + *B*$$(*i*, *i*)) **unfolding** *trace-def* **using** *A B* **by** *auto*
also have ... = (∑ *i*=*0*..<*n*. *A*$$(*i*, *i*)) + (∑ *i*=*0*..<*n*. *B*$$(*i*, *i*)) **by** (*auto simp add*: *sum.distrib*)
finally have *l*: *?lhs* = (∑ *i*=*0*..<*n*. *A*$$(*i*, *i*)) + (∑ *i*=*0*..<*n*. *B*$$(*i*, *i*)).
have *r*: *?rhs* = (∑ *i*=*0*..<*n*. *A*$$(*i*, *i*)) + (∑ *i*=*0*..<*n*. *B*$$(*i*, *i*)) **unfolding** *trace-def* **using** *A B* **by** *auto*
from *l r* **show** *?thesis* **by** *auto*
qed

lemma *trace-minus-linear*:
fixes *A B* :: *'a::comm-ring mat*
assumes *A*: *A* ∈ *carrier-mat n n* **and** *B*: *B* ∈ *carrier-mat n n*
shows *trace* (*A* − *B*) = *trace A* − *trace B* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (∑ *i*=*0*..<*n*. *A*$$(*i*, *i*) − *B*$$(*i*, *i*)) **unfolding** *trace-def* **using** *A B* **by** *auto*
also have ... = (∑ *i*=*0*..<*n*. *A*$$(*i*, *i*)) − (∑ *i*=*0*..<*n*. *B*$$(*i*, *i*)) **by** (*auto simp add*: *sum-subtractf*)

finally have *l*: *?lhs* = ($\sum$ *i*=*0*..<*n*. *A*$$(*i*, *i*)) − ($\sum$ *i*=*0*..<*n*. *B*$$(*i*, *i*))**.**
have *r*: *?rhs* = ($\sum$ *i*=*0*..<*n*. *A*$$(*i*, *i*)) − ($\sum$ *i*=*0*..<*n*. *B*$$(*i*, *i*)) **unfolding** *trace-def* **using** *A B* **by** *auto*
from *l r* **show** *?thesis* **by** *auto*
qed

lemma *trace-smult*:
assumes *A* ∈ *carrier-mat n n*
shows *trace* (*c* $\cdot_m$ *A*) = *c* ∗ *trace A*
proof −
have *trace* (*c* $\cdot_m$ *A*) = ($\sum$ *i* = *0*..<*dim-row A*. *c* ∗ *A* $$ (*i*, *i*)) **unfolding** *trace-def* **using** *assms* **by** *auto*
also have ... = *c* ∗ ($\sum$ *i* = *0*..<*dim-row A*. *A* $$ (*i*, *i*))
by (*simp add*: *sum-distrib-left*)
also have ... = *c* ∗ *trace A* **unfolding** *trace-def* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

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* · *w*) = *conjugate v* · *conjugate w*
using *assms* **by** (*simp add*: *scalar-prod-def sum-conjugate conjugate-dist-mul*)

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*
by (*simp add*: *scalar-prod-def conjugate-vec-def sum.neutral*)

lemma *conjugate-scalar-prod-Re* [*simp*]:
Re (*v* ·*c v*) ≥ *0*
by (*simp add*: *scalar-prod-def conjugate-vec-def sum-nonneg*)

lemma *self-cscalar-prod-geq-0*:
fixes *v* :: *'a*::*conjugatable-ordered-field vec*
shows *v* ·*c v* ≥ *0*
by (*auto simp add*: *scalar-prod-def*, *rule sum-nonneg*, *rule conjugate-square-positive*)

lemma *inner-prod-distrib-left*:
fixes *u v w* :: (*'a*::*conjugatable-field*) *vec*
assumes *dimu*: *u* ∈ *carrier-vec n* **and** *dimv*:*v* ∈ *carrier-vec n* **and** *dimw*: *w* ∈ *carrier-vec n*
shows *inner-prod* (*v* + *w*) *u* = *inner-prod v u* + *inner-prod w u* (**is** *?lhs* = *?rhs*)
proof −

have *dimcv*: *conjugate v* ∈ *carrier-vec n* **and** *dimcw*: *conjugate w* ∈ *carrier-vec n* **using** *assms* **by** *auto*
have *dimvw*: *conjugate* (*v* + *w*) ∈ *carrier-vec n* **using** *assms* **by** *auto*
have *u* · (*conjugate* (*v* + *w*)) = *u* · *conjugate v* + *u* · *conjugate w*
using *dimv dimw dimu dimcv dimcw*
by (*metis conjugate-add-vec scalar-prod-add-distrib*)
then show *?thesis* **by** *auto*
qed

lemma *inner-prod-distrib-right*:
fixes *u v w* :: (*'a*::*conjugatable-field*) *vec*
assumes *dimu*: *u* ∈ *carrier-vec n* **and** *dimv*:*v* ∈ *carrier-vec n* **and** *dimw*: *w* ∈ *carrier-vec n*
shows *inner-prod u* (*v* + *w*) = *inner-prod u v* + *inner-prod u w* (**is** *?lhs* = *?rhs*)
proof −
have *dimvw*: *v* + *w* ∈ *carrier-vec n* **using** *assms* **by** *auto*
have *dimcu*: *conjugate u* ∈ *carrier-vec n* **using** *assms* **by** *auto*
have (*v* + *w*) · (*conjugate u*) = *v* · *conjugate u* + *w* · *conjugate u*
apply (*simp add*: *comm-scalar-prod*[*OF dimvw dimcu*])
apply (*simp add*: *scalar-prod-add-distrib*[*OF dimcu dimv dimw*])
apply (*insert dimv dimw dimcu*, *simp add*: *comm-scalar-prod*[*of - n*])
done
then show *?thesis* **by** *auto*
qed

lemma *inner-prod-minus-distrib-right*:
fixes *u v w* :: (*'a*::*conjugatable-field*) *vec*
assumes *dimu*: *u* ∈ *carrier-vec n* **and** *dimv*:*v* ∈ *carrier-vec n* **and** *dimw*: *w* ∈ *carrier-vec n*
shows *inner-prod u* (*v* − *w*) = *inner-prod u v* − *inner-prod u w* (**is** *?lhs* = *?rhs*)
proof −
have *dimvw*: *v* − *w* ∈ *carrier-vec n* **using** *assms* **by** *auto*
have *dimcu*: *conjugate u* ∈ *carrier-vec n* **using** *assms* **by** *auto*
have (*v* − *w*) · (*conjugate u*) = *v* · *conjugate u* − *w* · *conjugate u*
apply (*simp add*: *comm-scalar-prod*[*OF dimvw dimcu*])
apply (*simp add*: *scalar-prod-minus-distrib*[*OF dimcu dimv dimw*])
apply (*insert dimv dimw dimcu*, *simp add*: *comm-scalar-prod*[*of - n*])
done
then show *?thesis* **by** *auto*
qed

lemma *inner-prod-smult-right*:
fixes *u v* :: *complex vec*
assumes *dimu*: *u* ∈ *carrier-vec n* **and** *dimv*:*v* ∈ *carrier-vec n*
shows *inner-prod* (*a* $\cdot_v$ *u*) *v* = *conjugate a* ∗ *inner-prod u v* (**is** *?lhs* = *?rhs*)
using *assms* **apply** (*simp add*: *scalar-prod-def conjugate-dist-mul*)
apply (*subst sum-distrib-left*) **by** (*rule sum.cong*, *auto*)

lemma *inner-prod-smult-left*:

fixes *u v* :: *complex vec*
assumes *dimu*: $u \in$ *carrier-vec n* **and** *dimv*: $v \in$ *carrier-vec n*
shows *inner-prod u* $(a \cdot_v v) = a *$ *inner-prod u v* (**is** *?lhs = ?rhs*)
using *assms* **apply** (*simp add*: *scalar-prod-def*)
apply (*subst sum-distrib-left*) **by** (*rule sum.cong*, *auto*)

lemma *inner-prod-smult-left-right*:
fixes *u v* :: *complex vec*
assumes *dimu*: $u \in$ *carrier-vec n* **and** *dimv*: $v \in$ *carrier-vec n*
shows *inner-prod* $(a \cdot_v u)$ $(b \cdot_v v)$ = *conjugate a* $*$ *b* $*$ *inner-prod u v* (**is** *?lhs = ?rhs*)
using *assms* **apply** (*simp add*: *scalar-prod-def*)
apply (*subst sum-distrib-left*) **by** (*rule sum.cong*, *auto*)

lemma *inner-prod-swap*:
fixes *x y* :: *complex vec*
assumes $y \in$ *carrier-vec n* **and** $x \in$ *carrier-vec n*
shows *inner-prod y x* = *conjugate* (*inner-prod x y*)
apply (*simp add*: *scalar-prod-def*)
apply (*rule sum.cong*) **using** *assms* **by** *auto*

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 −
have (*inner-prod* $(x+ a \cdot_v y)$ $(x+a \cdot_v y)$) = (*inner-prod* $(x+a \cdot_v y)$ *x*) + (*inner-prod* $(x+a \cdot_v y)$ $(a \cdot_v y)$)
apply (*subst inner-prod-distrib-right*) **using** *assms* **by** *auto*
also have ... = *inner-prod x x* + (*a*) $*$ (*inner-prod x y*) + *cnj a* $*$ ((*cnj* (*inner-prod x y*)) + (*a*) $*$ (*inner-prod y y*))
apply (*subst* (*1 2*) *inner-prod-distrib-left*[*of - n*]) **apply** (*auto simp add*: *assms*)
apply (*subst* (*1 2*) *inner-prod-smult-right*[*of - n*]) **apply** (*auto simp add*: *assms*)
apply (*subst inner-prod-smult-left*[*of - n*]) **apply** (*auto simp add*: *assms*)
apply (*subst inner-prod-swap*[*of y n x*]) **apply** (*auto simp add*: *assms*)
unfolding *distrib-left*
by *auto*
finally show *?thesis* **by** (*metis self-cscalar-prod-geq-0*)
qed

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* $\leq$ *inner-prod x x* $*$ *inner-prod y y*
proof −

define *cnj-a* **where** *cnj-a* = − (*inner-prod x y*)/ *cnj* (*inner-prod y y*)
define *a* **where** *a* = *cnj* (*cnj-a*)
have *cnj-rw*: (*cnj a*) = *cnj-a*
unfolding *a-def* **by** (*simp*)
have *rw-0*: *cnj* (*inner-prod x y*) + *a* ∗ (*inner-prod y y*) = *0*
unfolding *a-def cnj-a-def* **using** *assms*(*1*) *assms*(*2*) *conjugate-square-eq-0-vec* **by** *fastforce*
have *0* ≤ (*inner-prod x x* + *a* ∗ (*inner-prod x y*) + (*cnj a*) ∗ ((*cnj* (*inner-prod x y*)) + *a* ∗ (*inner-prod y y*)))
using *aux-Cauchy assms* **by** *auto*
also have ... = (*inner-prod x x* + *a* ∗ (*inner-prod x y*)) **unfolding** *rw-0* **by** *auto*
also have ... = (*inner-prod x x* − (*inner-prod x y*) ∗ *cnj* (*inner-prod x y*) / (*inner-prod y y*))
unfolding *a-def cnj-a-def* **by** *simp*
finally have *0* ≤ (*inner-prod x x* − (*inner-prod x y*) ∗ *cnj* (*inner-prod x y*) / (*inner-prod y y*)) **.**
hence *0* ≤ (*inner-prod x x* − (*inner-prod x y*) ∗ *cnj* (*inner-prod x y*) / (*inner-prod y y*)) ∗ (*inner-prod y y*)
by (*auto simp*: *less-eq-complex-def*)
also have ... = ((*inner-prod x x*)∗(*inner-prod y y*) − (*inner-prod x y*) ∗ *cnj* (*inner-prod x y*))
by (*smt* (*verit*) *add.inverse-neutral add-diff-cancel diff-0 diff-divide-eq-iff divide-cancel-right mult-eq-0-iff nonzero-mult-div-cancel-right rw-0*)
finally have (*inner-prod x y*) ∗ *cnj* (*inner-prod x y*) ≤ (*inner-prod x x*)∗(*inner-prod y y*) **by** *auto*
then show *?thesis*
apply (*subst inner-prod-swap*[*of y n x*]) **by** (*auto simp add*: *assms*)
qed

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* **by** (*simp add*: *mat-adjoint-def*)

lemma *adjoint-dim-col* [*simp*]:
dim-col (*adjoint A*) = *dim-row A* **by** (*simp add*: *mat-adjoint-def*)

lemma *adjoint-dim*:
A ∈ *carrier-mat n n* ⟹ *adjoint A* ∈ *carrier-mat n n*
using *adjoint-dim-col adjoint-dim-row* **by** *blast*

lemma *adjoint-def*:
adjoint A = *mat* (*dim-col A*) (*dim-row A*) (λ(*i*,*j*). *conjugate* (*A* $$ (*j*,*i*)))
unfolding *mat-adjoint-def mat-of-rows-def* **by** *auto*

lemma *adjoint-eval*:

```
  assumes i < dim-col A j < dim-row A
  shows (adjoint A) $$ (i,j) = conjugate (A $$ (j,i))
  using assms by (simp add: adjoint-def)

lemma adjoint-row:
  assumes i < dim-col A
  shows row (adjoint A) i = conjugate (col A i)
  apply (rule eq-vecI)
  using assms by (auto simp add: adjoint-eval)

lemma adjoint-col:
  assumes i < dim-row A
  shows col (adjoint A) i = conjugate (row A i)
  apply (rule eq-vecI)
  using assms by (auto simp add: adjoint-eval)
```

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 ∈ carrier-vec n w ∈ carrier-vec m A ∈ carrier-mat n m
  shows inner-prod v (A *v w) = inner-prod (adjoint A *v v) w (is ?lhs = ?rhs)
proof –
  from dims have ?lhs = (∑ i=0..<dim-vec v. (∑ j=0..<dim-vec w.
              conjugate (v$i) * A$$(i, j) * w$j))
    apply (simp add: scalar-prod-def sum-distrib-right )
    apply (rule sum.cong, simp)
    apply (rule sum.cong, auto)
    done
  moreover from assms have ?rhs = (∑ i=0..<dim-vec v. (∑ j=0..<dim-vec w.
              conjugate (v$i) * A$$(i, j) * w$j))
    apply (simp add: scalar-prod-def  adjoint-eval
                sum-conjugate conjugate-dist-mul sum-distrib-left)
    apply (subst sum.swap[where ?A = {0..<n}])
    apply (rule sum.cong, simp)
    apply (rule sum.cong, auto)
    done
  ultimately show ?thesis by simp
qed

lemma adjoint-one:
  shows adjoint (1m n) = (1m n::complex mat)
  apply (rule eq-matI)
  by (auto simp add: adjoint-eval)

lemma adjoint-scale:
  fixes A :: 'a::conjugatable-field mat
  shows adjoint (a ·m A) = (conjugate a) ·m adjoint A
  apply (rule eq-matI) using conjugatable-ring-class.conjugate-dist-mul
```

by (*auto simp add*: *adjoint-eval*)

lemma *adjoint-add*:
fixes *A B* :: *'a::conjugatable-field mat*
assumes *A* ∈ *carrier-mat n m B* ∈ *carrier-mat n m*
shows *adjoint* (*A* + *B*) = *adjoint A* + *adjoint B*
apply (*rule eq-matI*)
using *assms conjugatable-ring-class.conjugate-dist-add*
by(*auto simp add*: *adjoint-eval*)

lemma *adjoint-minus*:
fixes *A B* :: *'a::conjugatable-field mat*
assumes *A* ∈ *carrier-mat n m B* ∈ *carrier-mat n m*
shows *adjoint* (*A* − *B*) = *adjoint A* − *adjoint B*
apply (*rule eq-matI*)
using *assms* **apply**(*auto simp add*: *adjoint-eval*)
by (*metis add-uminus-conv-diff conjugate-dist-add conjugate-neg*)

lemma *adjoint-mult*:
fixes *A B* :: *'a::conjugatable-field mat*
assumes *A* ∈ *carrier-mat n m B* ∈ *carrier-mat m l*
shows *adjoint* (*A* ∗ *B*) = *adjoint B* ∗ *adjoint A*
proof (*rule eq-matI*, *auto simp add*: *adjoint-eval adjoint-row adjoint-col*)
fix *i j*
assume *i* < *dim-col B j* < *dim-row A*
show *conjugate* (*row A j* · *col B i*) = *conjugate* (*col B i*) · *conjugate* (*row A j*)
using *assms* **apply** (*simp add*: *conjugate-scalar-prod*)
apply (*subst comm-scalar-prod*[**where** *n*=*dim-row B*])
by (*auto simp add*: *carrier-vecI*)
qed

lemma *adjoint-adjoint*:
fixes *A* :: *'a::conjugatable-field mat*
shows *adjoint* (*adjoint A*) = *A*
by (*rule eq-matI*, *auto simp add*: *adjoint-eval*)

lemma *trace-adjoint-positive*:
fixes *A* :: *complex mat*
shows *trace* (*A* ∗ *adjoint A*) ≥ *0*
apply (*auto simp add*: *trace-def adjoint-col*)
apply (*rule sum-nonneg*) **by** *auto*

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*
by (*intro eq-matI*, *auto*)

lemma *add-carrier-mat'*:

$A \in carrier\text{-}mat\ nr\ nc \Longrightarrow B \in carrier\text{-}mat\ nr\ nc \Longrightarrow A + B \in carrier\text{-}mat\ nr\ nc$
by *simp*

lemma *minus-carrier-mat′*:
$A \in carrier\text{-}mat\ nr\ nc \Longrightarrow B \in carrier\text{-}mat\ nr\ nc \Longrightarrow A - B \in carrier\text{-}mat\ nr\ nc$
by *auto*

lemma *swap-plus-mat*:
fixes $A\ B\ C :: 'a::semiring\text{-}1\ mat$
assumes $A \in carrier\text{-}mat\ n\ n\ B \in carrier\text{-}mat\ n\ n\ C \in carrier\text{-}mat\ n\ n$
shows $A + B + C = A + C + B$
by (*metis assms assoc-add-mat comm-add-mat*)

lemma *uminus-mat*:
fixes $A :: complex\ mat$
assumes $A \in carrier\text{-}mat\ n\ n$
shows $-A = (-1) \cdot_m A$
by *auto*

ML-file *mat-alg.ML*
method-setup *mat-assoc* = ‹*mat-assoc-method*›
Normalization of expressions on matrices

lemma *mat-assoc-test*:
fixes $A\ B\ C\ D :: complex\ mat$
assumes $A \in carrier\text{-}mat\ n\ n\ B \in carrier\text{-}mat\ n\ n\ C \in carrier\text{-}mat\ n\ n\ D \in carrier\text{-}mat\ n\ n$
shows
$(A * B) * (C * D) = A * B * C * D$
$adjoint\ (A * adjoint\ B) * C = B * (adjoint\ A * C)$
$A * 1_m\ n * 1_m\ n * B * 1_m\ n = A * B$
$(A - B) + (B - C) = A + (-B) + B + (-C)$
$A + (B - C) = A + B - C$
$A - (B + C + D) = A - B - C - D$
$(A + B) * (B + C) = A * B + B * B + A * C + B * C$
$A - B = A + (-1) \cdot_m B$
$A * (B - C) * D = A * B * D - A * C * D$
$trace\ (A * B * C) = trace\ (B * C * A)$
$trace\ (A * B * C * D) = trace\ (C * D * A * B)$
$trace\ (A + B * C) = trace\ A + trace\ (C * B)$
$A + B = B + A$
$A + B + C = C + B + A$
$A + B + (C + D) = A + C + (B + D)$
using *assms* **by** (*mat-assoc n*)+

1.6 Hermitian matrices

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

definition *hermitian* :: *'a::conjugatable-field mat* ⇒ *bool* **where**
hermitian A ⟷ (*adjoint A* = *A*)

lemma *hermitian-one*:
shows *hermitian* ((*1*$_m$ *n*)::(*'a::conjugatable-field mat*))
unfolding *hermitian-def*
proof−
have *conjugate* (*1::'a*) = *1*
apply (*subst mult-1-right*[*symmetric*, *of conjugate 1*])
apply (*subst conjugate-id*[*symmetric*, *of conjugate 1* ∗ *1*])
apply (*subst conjugate-dist-mul*)
apply *auto*
done
then show *adjoint* ((*1*$_m$ *n*)::(*'a::conjugatable-field mat*)) = (*1*$_m$ *n*)
by (*auto simp add*: *adjoint-eval*)
qed

1.7 Inverse matrices

lemma *inverts-mat-symm*:
fixes *A B* :: *'a::field mat*
assumes *dim*: *A* ∈ *carrier-mat n n B* ∈ *carrier-mat n n*
and *AB*: *inverts-mat A B*
shows *inverts-mat B A*
proof −
have *A* ∗ *B* = *1*$_m$ *n* **using** *dim AB* **unfolding** *inverts-mat-def* **by** *auto*
with *dim* **have** *B* ∗ *A* = *1*$_m$ *n* **by** (*rule mat-mult-left-right-inverse*)
then show *inverts-mat B A* **using** *dim inverts-mat-def* **by** *auto*
qed

lemma *inverts-mat-unique*:
fixes *A B C* :: *'a::field mat*
assumes *dim*: *A* ∈ *carrier-mat n n B* ∈ *carrier-mat n n C* ∈ *carrier-mat n n*
and *AB*: *inverts-mat A B* **and** *AC*: *inverts-mat A C*
shows *B* = *C*
proof −
have *AB1*: *A* ∗ *B* = *1*$_m$ *n* **using** *AB dim* **unfolding** *inverts-mat-def* **by** *auto*
have *A* ∗ *C* = *1*$_m$ *n* **using** *AC dim* **unfolding** *inverts-mat-def* **by** *auto*
then have *CA1*: *C* ∗ *A* = *1*$_m$ *n* **using** *mat-mult-left-right-inverse*[*of A n C*] *dim*
by *auto*
then have *C* = *C* ∗ *1*$_m$ *n* **using** *dim* **by** *auto*
also have ... = *C* ∗ (*A* ∗ *B*) **using** *AB1* **by** *auto*
also have ... = (*C* ∗ *A*) ∗ *B* **using** *dim* **by** *auto*
also have ... = *1*$_m$ *n* ∗ *B* **using** *CA1* **by** *auto*
also have ... = *B* **using** *dim* **by** *auto*
finally show *B* = *C* **..**

qed

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* ∈ *carrier-mat* (*dim-row A*) (*dim-row A*) ∧ *inverts-mat A* (*adjoint A*)

lemma *unitaryD2*:
assumes *A* ∈ *carrier-mat n n*
shows *unitary A* ⟹ *inverts-mat* (*adjoint A*) *A*
using *assms adjoint-dim inverts-mat-symm unitary-def* **by** *blast*

lemma *unitary-simps* [*simp*]:
A ∈ *carrier-mat n n* ⟹ *unitary A* ⟹ *adjoint A* ∗ *A* = 1_m *n*
A ∈ *carrier-mat n n* ⟹ *unitary A* ⟹ *A* ∗ *adjoint A* = 1_m *n*
apply (*metis adjoint-dim-row carrier-matD*(*2*) *inverts-mat-def unitaryD2*)
by (*simp add*: *inverts-mat-def unitary-def*)

lemma *unitary-adjoint* [*simp*]:
assumes *A* ∈ *carrier-mat n n unitary A*
shows *unitary* (*adjoint A*)
unfolding *unitary-def*
using *adjoint-dim*[*OF assms*(*1*)] *assms* **by** (*auto simp add*: *unitaryD2*[*OF assms*] *adjoint-adjoint*)

lemma *unitary-one*:
shows *unitary* ((1_m *n*)::(*'a::conjugatable-field mat*))
unfolding *unitary-def*
proof −
define *I* **where** *I-def*[*simp*]: *I* ≡ ((1_m *n*)::(*'a::conjugatable-field mat*))
have *dim*: *I* ∈ *carrier-mat n n* **by** *auto*
have *hermitian I* **using** *hermitian-one* **by** *auto*
hence *adjoint I* = *I* **using** *hermitian-def* **by** *auto*
with *dim* **show** *I* ∈ *carrier-mat* (*dim-row I*) (*dim-row I*) ∧ *inverts-mat I* (*adjoint I*)
unfolding *inverts-mat-def* **using** *dim* **by** *auto*
qed

lemma *unitary-zero*:
fixes *A* :: *'a::conjugatable-field mat*
assumes *A* ∈ *carrier-mat 0 0*
shows *unitary A*
unfolding *unitary-def inverts-mat-def Let-def* **using** *assms* **by** *auto*

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 –
have *dimaP*: *adjoint P* ∈ *carrier-mat n n* **using** *dims* **by** *auto*
have *iv*: *inverts-mat P* (*adjoint P*) **using** *uP unitary-def* **by** *auto*
then have *P* ∗ (*adjoint P*) = 1_m *n* **using** *inverts-mat-def dims* **by** *auto*
then have *aPP*: *adjoint P* ∗ *P* = 1_m *n* **using** *mat-mult-left-right-inverse*[*OF dims*(*3*) *dimaP*] **by** *auto*
have *adjoint P* ∗ (*P* ∗ *A* ∗ *adjoint P*) ∗ *P* = (*adjoint P* ∗ *P*) ∗ *A* ∗ (*adjoint P* ∗ *P*)
using *dims dimaP* **by** (*mat-assoc n*)
also have ... = 1_m *n* ∗ *A* ∗ 1_m *n* **using** *aPP* **by** *auto*
also have ... = *A* **using** *dims* **by** *auto*
finally have *eqA*: *A* = *adjoint P* ∗ (*P* ∗ *A* ∗ *adjoint P*) ∗ *P* **..**
have *adjoint P* ∗ (*P* ∗ *B* ∗ *adjoint P*) ∗ *P* = (*adjoint P* ∗ *P*) ∗ *B* ∗ (*adjoint P* ∗ *P*)
using *dims dimaP* **by** (*mat-assoc n*)
also have ... = 1_m *n* ∗ *B* ∗ 1_m *n* **using** *aPP* **by** *auto*
also have ... = *B* **using** *dims* **by** *auto*
finally have *eqB*: *B* = *adjoint P* ∗ (*P* ∗ *B* ∗ *adjoint P*) ∗ *P* **..**
then show *?thesis* **using** *eqA eqB eq* **by** *auto*
qed

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*
unfolding *corthogonal-mat-def Let-def*
proof (*rule conjI*)
have *dima*: *adjoint U* ∈ *carrier-mat n n* **using** *dim* **by** *auto*
have *aUU*: *mat-adjoint U* ∗ *U* = (1_m *n*)
apply (*insert U*[*unfolded unitary-def*] *dim dima*, *drule conjunct2*)
apply (*drule inverts-mat-symm*[*of U*, *OF dim dima*], *unfold inverts-mat-def*, *auto*)
done
then show *diagonal-mat* (*mat-adjoint U* ∗ *U*)
by (*simp add*: *diagonal-mat-def*)
show ∀ *i*<*dim-col U*. (*mat-adjoint U* ∗ *U*) $$ (*i*, *i*) ≠ *0* **using** *dim* **by** (*simp add*: *aUU*)
qed

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 –
have *dim-pq*: *P* ∗ *Q* ∈ *carrier-mat n n* **using** *dim* **by** *auto*
have (*P* ∗ *Q*) ∗ *adjoint* (*P* ∗ *Q*) = *P* ∗ (*Q* ∗ *adjoint Q*) ∗ *adjoint P* **using** *dim*

by (*mat-assoc n*)
  also have ... = $P * (1_m\ n) * adjoint\ P$ **using** *uQ dim* **by** *auto*
  also have ... = $P * adjoint\ P$ **using** *dim* **by** (*mat-assoc n*)
  also have ... = $1_m\ n$ **using** *uP dim* **by** *simp*
  finally have $(P * Q) * adjoint\ (P * Q) = 1_m\ n$ **by** *auto*
  hence *inverts-mat* $(P * Q)$ (*adjoint* $(P * Q)$)
    using *inverts-mat-def dim-pq* **by** *auto*
  thus *unitary* $(P{*}Q)$ **using** *unitary-def dim-pq* **by** *auto*
qed

lemma *unitary-operator-keep-trace*:
  fixes *U A* :: *complex mat*
  assumes *dU*: $U \in$ *carrier-mat n n* **and** *dA*: $A \in$ *carrier-mat n n* **and** *u*: *unitary U*
  shows *trace A* = *trace* (*adjoint* $U * A * U$)
proof −
  have u': $U * adjoint\ U = 1_m\ n$ **using** *u* **unfolding** *unitary-def inverts-mat-def* **using** *dU* **by** *auto*
  have *trace* (*adjoint* $U * A * U$) = *trace* ($U *$ *adjoint* $U * A$) **using** *dU dA* **by** (*mat-assoc n*)
  also have ... = *trace A* **using** u' *dA* **by** *auto*
  finally show *?thesis* **by** *auto*
qed

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*
  unfolding *vec-norm-def* **by** (*insert self-cscalar-prod-geq-0*[*of v*], *simp add*: *less-eq-complex-def*)

lemma *vec-norm-zero*:
  fixes *v* :: *complex vec*
  assumes *dim*: $v \in$ *carrier-vec n*
  shows *vec-norm v* = *0* ⟷ $v = 0_v\ n$
  unfolding *vec-norm-def*
  by (*subst conjugate-square-eq-0-vec*[*OF dim*, *symmetric*], *rule csqrt-eq-0*)

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 −
  have *geq*: *vec-norm v* ≥ *0* **using** *vec-norm-geq-0* **by** *auto*
  have *neq*: *vec-norm v* ≠ *0*
    apply (*insert dim-v neq0*)

```
    apply (drule vec-norm-zero, auto)
    done
  show ?thesis using neq geq by (rule dual-order.not-eq-order-implies-strict)
qed

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) ·_v v)

lemma normalized-vec-dim[simp]:
  assumes (v::complex vec) ∈ carrier-vec n
  shows vec-normalize v ∈ carrier-vec n
  unfolding vec-normalize-def using assms by auto

lemma vec-eq-norm-smult-normalized:
  shows v = vec-norm v ·_v vec-normalize v
proof (cases v = 0_v (dim-vec v))
  define n where n = dim-vec v
  then have dimv: v ∈ carrier-vec n by auto
  then have dimnv: vec-normalize v ∈ carrier-vec n by auto
  {
    case True
    then have v0: v = 0_v n using n-def by auto
    then have n0: vec-norm v = 0 using vec-norm-def by auto
    have vec-norm v ·_v vec-normalize v = 0_v n
      unfolding smult-vec-def by (auto simp add: n0 carrier-vecD[OF dimnv])
    then show ?thesis using v0 by auto
    next
    case False
    then have v: v ≠ 0_v n using n-def by auto
    then have ge0: vec-norm v > 0 using vec-norm-ge-0 dimv by auto
     have vec-normalize v = (1 / vec-norm v) ·_v v using False vec-normalize-def
by auto
    then have vec-norm v ·_v vec-normalize v = (vec-norm v * (1 / vec-norm v))
·_v v
      using smult-smult-assoc by auto
    also have ... = v using ge0 by auto
    finally have v = vec-norm v ·_v vec-normalize v..
    then show v = vec-norm v ·_v vec-normalize v using v by auto
  }
qed

lemma normalized-cscalar-prod:
  fixes v w :: complex vec
  assumes dim-v: v ∈ carrier-vec n and dim-w: w ∈ carrier-vec n
  shows v •c w = (vec-norm v * vec-norm w) * (vec-normalize v •c vec-normalize
w)
  unfolding vec-normalize-def apply (split if-split, split if-split)
proof (intro conjI impI)
  note dim0 = dim-v dim-w
```

have *dim*: $\textit{dim-vec}\ v = n\ \textit{dim-vec}\ w = n$ **using** *dim0* **by** *auto*
{
assume $w = 0_v\ n\ v = 0_v\ n$
then have *lhs*: $v \bullet c\ w = 0$ **by** *auto*
then moreover have *rhs*: $\textit{vec-norm}\ v * \textit{vec-norm}\ w * (v \bullet c\ w) = 0$ **by** *auto*
ultimately have $v \bullet c\ w = \textit{vec-norm}\ v * \textit{vec-norm}\ w * (v \bullet c\ w)$ **by** *auto*
}
with *dim* **show** $w = 0_v\ (\textit{dim-vec}\ w) \Longrightarrow v = 0_v\ (\textit{dim-vec}\ v) \Longrightarrow v \bullet c\ w = \textit{vec-norm}\ v * \textit{vec-norm}\ w * (v \bullet c\ w)$ **by** *auto*
{
assume *asm*: $w = 0_v\ n\ v \neq 0_v\ n$
then have *w0*: $\textit{conjugate}\ w = 0_v\ n$ **by** *auto*
with *dim0* **have** $(1\ /\ \textit{vec-norm}\ v \cdot_v v) \bullet c\ w = 0$ **by** *auto*
then moreover have *rhs*: $\textit{vec-norm}\ v * \textit{vec-norm}\ w * ((1\ /\ \textit{vec-norm}\ v \cdot_v v) \bullet c\ w) = 0$ **by** *auto*
moreover have $v \bullet c\ w = 0$ **using** *w0 dim0* **by** *auto*
ultimately have $v \bullet c\ w = \textit{vec-norm}\ v * \textit{vec-norm}\ w * ((1\ /\ \textit{vec-norm}\ v \cdot_v v) \bullet c\ w)$ **by** *auto*
}
with *dim* **show** $w = 0_v\ (\textit{dim-vec}\ w) \Longrightarrow v \neq 0_v\ (\textit{dim-vec}\ v) \Longrightarrow v \bullet c\ w = \textit{vec-norm}\ v * \textit{vec-norm}\ w * ((1\ /\ \textit{vec-norm}\ v \cdot_v v) \bullet c\ w)$ **by** *auto*
{
assume *asm*: $w \neq 0_v\ n\ v = 0_v\ n$
with *dim0* **have** $v \bullet c\ (1\ /\ \textit{vec-norm}\ w \cdot_v w) = 0$ **by** *auto*
then moreover have *rhs*: $\textit{vec-norm}\ v * \textit{vec-norm}\ w * (v \bullet c\ (1\ /\ \textit{vec-norm}\ w \cdot_v w)) = 0$ **by** *auto*
moreover have $v \bullet c\ w = 0$ **using** *asm dim0* **by** *auto*
ultimately have $v \bullet c\ w = \textit{vec-norm}\ v * \textit{vec-norm}\ w * (v \bullet c\ (1\ /\ \textit{vec-norm}\ w \cdot_v w))$ **by** *auto*
}
with *dim* **show** $w \neq 0_v\ (\textit{dim-vec}\ w) \Longrightarrow v = 0_v\ (\textit{dim-vec}\ v) \Longrightarrow v \bullet c\ w = \textit{vec-norm}\ v * \textit{vec-norm}\ w * (v \bullet c\ (1\ /\ \textit{vec-norm}\ w \cdot_v w))$ **by** *auto*
{
assume *asmw*: $w \neq 0_v\ n$ **and** *asmv*: $v \neq 0_v\ n$
have $\textit{vec-norm}\ w > 0$ **by** (*insert asmw dim0*, *rule vec-norm-ge-0*, *auto*)
then have *cw*: $\textit{conjugate}\ (1\ /\ \textit{vec-norm}\ w) = 1\ /\ \textit{vec-norm}\ w$
by (*simp add*: *complex-eq-iff complex-is-Real-iff less-complex-def*)
from *dim0* **have**
$((1\ /\ \textit{vec-norm}\ v \cdot_v v) \bullet c\ (1\ /\ \textit{vec-norm}\ w \cdot_v w)) = 1\ /\ \textit{vec-norm}\ v * (v \bullet c\ (1\ /\ \textit{vec-norm}\ w \cdot_v w))$ **by** *auto*
also have $\ldots = 1\ /\ \textit{vec-norm}\ v * (v \bullet (\textit{conjugate}\ (1\ /\ \textit{vec-norm}\ w) \cdot_v \textit{conjugate}\ w))$
by (*subst conjugate-smult-vec*, *auto*)
also have $\ldots = 1\ /\ \textit{vec-norm}\ v * \textit{conjugate}\ (1\ /\ \textit{vec-norm}\ w) * (v \bullet \textit{conjugate}\ w)$ **using** *dim* **by** *auto*
also have $\ldots = 1\ /\ \textit{vec-norm}\ v * (1\ /\ \textit{vec-norm}\ w) * (v \bullet c\ w)$ **using** *vec-norm-ge-0 cw* **by** *auto*
finally have *eq1*: $(1\ /\ \textit{vec-norm}\ v \cdot_v v) \bullet c\ (1\ /\ \textit{vec-norm}\ w \cdot_v w) = 1\ /\ \textit{vec-norm}\ v * (1\ /\ \textit{vec-norm}\ w) * (v \bullet c\ w)$ **.**

then have *vec-norm v ∗ vec-norm w ∗ ((1 / vec-norm v* $\cdot_v$ *v) ·c (1 / vec-norm w* $\cdot_v$ *w)) = (v ·c w)*
by (*subst eq1*, *insert vec-norm-ge-0*[*of v n*, *OF dim-v asmv*] *vec-norm-ge-0*[*of w n*, *OF dim-w asmw*], *auto*)
}
with *dim* **show** *w ≠* 0_v *(dim-vec w) ⟹ v ≠* 0_v *(dim-vec v) ⟹ v ·c w = vec-norm v ∗ vec-norm w ∗ ((1 / vec-norm v* $\cdot_v$ *v) ·c (1 / vec-norm w* $\cdot_v$ *w))* **by** *auto*
qed

lemma *normalized-vec-norm* :
fixes *v* :: *complex vec*
assumes *dim-v*: *v ∈ carrier-vec n*
and *neq0*: *v ≠* 0_v *n*
shows *vec-normalize v ·c vec-normalize v = 1*
unfolding *vec-normalize-def*
proof (*simp*, *rule conjI*)
show *v =* 0_v *(dim-vec v) ⟶ v ·c v = 1* **using** *neq0 dim-v* **by** *auto*
have *dim-a*: (*vec-normalize v*) *∈ carrier-vec n conjugate* (*vec-normalize v*) *∈ carrier-vec n* **using** *dim-v vec-normalize-def* **by** *auto*
note *dim = dim-v dim-a*
have *nvge0*: *vec-norm v > 0* **using** *vec-norm-ge-0 neq0 dim-v* **by** *auto*
then have *vvvv*: *v ·c v =* (*vec-norm v*) ∗ (*vec-norm v*) **unfolding** *vec-norm-def* **by** (*metis power2-csqrt power2-eq-square*)
from *nvge0* **have** *conjugate* (*vec-norm v*) *= vec-norm v*
by (*simp add*: *complex-eq-iff complex-is-Real-iff less-complex-def*)
then have *v ·c (1 / vec-norm v* $\cdot_v$ *v) = 1 / vec-norm v ∗ (v ·c v)*
by (*subst conjugate-smult-vec*, *auto*)
also have ... *= 1 / vec-norm v ∗ vec-norm v ∗ vec-norm v* **using** *vvvv* **by** *auto*
also have ... *= vec-norm v* **by** *auto*
finally have *v ·c (1 / vec-norm v* $\cdot_v$ *v) = vec-norm v*.
then show *v ≠* 0_v *(dim-vec v) ⟶ vec-norm v ≠ 0 ∧ v ·c (1 / vec-norm v* $\cdot_v$ *v) = vec-norm v*
using *neq0 nvge0* **by** *auto*
qed

lemma *normalize-zero*:
assumes *v ∈ carrier-vec n*
shows *vec-normalize v =* 0_v *n ⟷ v =* 0_v *n*
proof
show *v =* 0_v *n ⟹ vec-normalize v =* 0_v *n* **unfolding** *vec-normalize-def* **by** *auto*
next
have *v ≠* 0_v *n ⟹ vec-normalize v ≠* 0_v *n* **unfolding** *vec-normalize-def*
proof (*simp*, *rule impI*)
assume *asm*: *v ≠* 0_v *n*
then have *vec-norm v > 0* **using** *vec-norm-ge-0 assms* **by** *auto*
then have *nvge0*: *1 / vec-norm v > 0* **by** (*simp add*: *complex-is-Real-iff less-complex-def*)

have ∃ *k* < *n*. *v* $ *k* ≠ *0* **using** *asm assms* **by** *auto*
then obtain *k* **where** *kn*: *k* < *n* **and** *vkneq0*: *v* $ *k* ≠ *0* **by** *auto*
then have (*1* / *vec-norm v* $\cdot_v$ *v*) $ *k* = (*1* / *vec-norm v*) ∗ (*v* $ *k*)
using *assms carrier-vecD index-smult-vec*(*1*) **by** *blast*
with *nvge0 vkneq0* **have** (*1* / *vec-norm v* $\cdot_v$ *v*) $ *k* ≠ *0* **by** *auto*
then show *1* / *vec-norm v* $\cdot_v$ *v* ≠ 0_v *n* **using** *assms kn* **by** *fastforce*
qed
then show *vec-normalize v* = 0_v *n* ⟹ *v* = 0_v *n* **by** *auto*
qed

lemma *normalize-normalize*[*simp*]:
vec-normalize (*vec-normalize v*) = *vec-normalize v*
proof (*rule disjE*[*of v* = 0_v (*dim-vec v*) *v* ≠ 0_v (*dim-vec v*)], *auto*)
let *?n* = *dim-vec v*
{
assume *v* = 0_v *?n*
then have *vec-normalize v* = *v* **unfolding** *vec-normalize-def* **by** *auto*
then show *vec-normalize* (*vec-normalize v*) = *vec-normalize v* **by** *auto*
}
assume *neq0*: *v* ≠ 0_v *?n*
have *dim*: *v* ∈ *carrier-vec ?n* **by** *auto*
have *vec-norm* (*vec-normalize v*) = *1* **unfolding** *vec-norm-def*
using *normalized-vec-norm*[*OF dim neq0*] **by** *auto*
then show *vec-normalize* (*vec-normalize v*) = *vec-normalize v*
by (*subst* (*1*) *vec-normalize-def*, *simp*)
qed

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*)
unfolding *corthogonal-def*
proof (*rule allI*, *rule impI*, *rule allI*, *rule impI*, *goal-cases*)
case *c*: (*1 i j*)
let *?m* = *length vs*
have *len*: *length* (*map vec-normalize vs*) = *?m* **by** *auto*
have *dim*: ⋀*k*. *k* < *?m* ⟹ (*vs* ! *k*) ∈ *carrier-vec n* **using** *dims* **by** *auto*
have *map*: ⋀*k*. *k* < *?m* ⟹ *map vec-normalize vs* ! *k* = *vec-normalize* (*vs* ! *k*)
by *auto*

have *eq1*: ⋀*j k*. *j* < *?m* ⟹ *k* < *?m* ⟹ ((*vs* ! *j*) ∙c (*vs* ! *k*) = *0*) = (*j* ≠ *k*)
using *assms* **unfolding** *corthogonal-def* **by** *auto*
then have ⋀*k*. *k* < *?m* ⟹ (*vs* ! *k*) ∙c (*vs* ! *k*) ≠ *0* **by** *auto*
then have ⋀*k*. *k* < *?m* ⟹ (*vs* ! *k*) ≠ (0_v *n*) **using** *dim*
by (*auto simp add*: *conjugate-square-eq-0-vec*[*of - n*, *OF dim*])
then have *vnneq0*: ⋀*k*. *k* < *?m* ⟹ *vec-norm* (*vs* ! *k*) ≠ *0* **using** *vec-norm-zero*[*OF dim*] **by** *auto*

then have *i0*: *vec-norm* (*vs* ! *i*) ≠ *0* **and** *j0*: *vec-norm* (*vs* ! *j*) ≠ *0* **using** *c* **by** *auto*
have (*vs* ! *i*) ·c (*vs* ! *j*) = *vec-norm* (*vs* ! *i*) ∗ *vec-norm* (*vs* ! *j*) ∗ (*vec-normalize* (*vs* ! *i*) ·c *vec-normalize* (*vs* ! *j*))
by (*subst normalized-cscalar-prod*[*of vs* ! *i n vs* ! *j*], *auto*, *insert dim c*, *auto*)
with *i0 j0* **have** (*vec-normalize* (*vs* ! *i*) ·c *vec-normalize* (*vs* ! *j*) = *0*) = ((*vs* ! *i*) ·c (*vs* ! *j*) = *0*) **by** *auto*
with *eq1 c* **have** (*vec-normalize* (*vs* ! *i*) ·c *vec-normalize* (*vs* ! *j*) = *0*) = (*i* ≠ *j*) **by** *auto*
with *map c* **show** (*map vec-normalize vs* ! *i* ·c *map vec-normalize vs* ! *j* = *0*) = (*i* ≠ *j*) **by** *auto*
qed

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 −
define *vs* **where** *vs* = *map vec-normalize ws*
define *W* **where** *W* = *mat-of-cols n vs*
have *W′*: *set vs* ⊆ *carrier-vec n* **using** *assms vs-def* **by** *auto*
then have *W″*: ⋀*k*. *k* < *length vs* ⟹ *vs* ! *k* ∈ *carrier-vec n* **by** *auto*
have *orth′*: *corthogonal vs* **using** *assms normalize-keep-corthogonal vs-def* **by** *auto*
have *len′*[*simp*]: *length vs* = *n* **using** *assms vs-def* **by** *auto*
have *dimW*: *W* ∈ *carrier-mat n n* **using** *W-def len* **by** *auto*
have *adjoint W* ∈ *carrier-mat n n* **using** *dimW* **by** *auto*
then have *dimaW*: *mat-adjoint W* ∈ *carrier-mat n n* **by** *auto*
{
fix *i j* **assume** *i*: *i* < *n* **and** *j*: *j* < *n*
have *dimws*: (*ws* ! *i*) ∈ *carrier-vec n* (*ws* ! *j*) ∈ *carrier-vec n* **using** *W len i j* **by** *auto*
have (*ws* ! *i*) ·c (*ws* ! *i*) ≠ *0* (*ws* ! *j*) ·c (*ws* ! *j*) ≠ *0* **using** *orth corthogonal-def*[*of ws*] *len i j* **by** *auto*
then have *neq0*: (*ws* ! *i*) ≠ 0_v *n* (*ws* ! *j*) ≠ 0_v *n*
by (*auto simp add*: *conjugate-square-eq-0-vec*[*of ws* ! *i n*])
then have *vec-norm* (*ws* ! *i*) > *0* *vec-norm* (*ws* ! *j*) > *0* **using** *vec-norm-ge-0 dimws* **by** *auto*
then have *ge0*: *vec-norm* (*ws* ! *i*) ∗ *vec-norm* (*ws* ! *j*) > *0* **by** (*auto simp*: *less-complex-def*)
have *ws′*: *vs* ! *i* = *vec-normalize* (*ws* ! *i*)
vs ! *j* = *vec-normalize* (*ws* ! *j*)
using *len i j vs-def* **by** *auto*
have *ii1*: (*vs* ! *i*) ·c (*vs* ! *i*) = *1*
apply (*simp add*: *ws′*)
apply (*rule normalized-vec-norm*[*of ws* ! *i*], *rule dimws*, *rule neq0*)
done
have *ij0*: *i* ≠ *j* ⟹ (*ws* ! *i*) ·c (*ws* ! *j*) = *0* **using** *i j*

by (*insert orth*, *auto simp add*: *corthogonal-def*[*of ws*] *len*)
have $i \neq j \Longrightarrow (ws\ !\ i)\ \cdot c\ (ws\ !\ j) = (vec\text{-}norm\ (ws\ !\ i) * vec\text{-}norm\ (ws\ !\ j)) * ((vs\ !\ i)\ \cdot c\ (vs\ !\ j))$
apply (*auto simp add*: *ws'*)
apply (*rule normalized-cscalar-prod*)
apply (*rule dimws*, *rule dimws*)
done
with *ij0* **have** *ij0'*: $i \neq j \Longrightarrow (vs\ !\ i)\ \cdot c\ (vs\ !\ j) = 0$ **using** *ge0* **by** *auto*
have *cWk*: $\bigwedge k.\ k < n \Longrightarrow col\ W\ k = vs\ !\ k$ **unfolding** *W-def*
apply (*subst col-mat-of-cols*)
apply (*auto simp add*: *W''*)
done
have (*mat-adjoint W* $*$ *W*) $$ (*j*, *i*) = *row* (*mat-adjoint W*) *j* $\cdot$ *col W i*
by (*insert dimW i j dimaW*, *auto*)
also have ... = *conjugate* (*col W j*) $\cdot$ *col W i*
by (*insert dimW i j dimaW*, *auto simp add*: *mat-adjoint-def*)
also have ... = *col W i* $\cdot$ *conjugate* (*col W j*) **using** *comm-scalar-prod*[*of col W i n*] *dimW* **by** *auto*
also have ... = $(vs\ !\ i)\ \cdot c\ (vs\ !\ j)$ **using** *W-def col-mat-of-cols i j len cWk* **by** *auto*
finally have (*mat-adjoint W* $*$ *W*) $$ (*j*, *i*) = $(vs\ !\ i)\ \cdot c\ (vs\ !\ j)$**.**
then have (*mat-adjoint W* $*$ *W*) $$ (*j*, *i*) = (*if* (*j* = *i*) *then 1 else 0*)
by (*auto simp add*: *ii1 ij0'*)
}
note *maWW* = *this*
then have *mat-adjoint W* $*$ *W* = 1_m *n* **unfolding** *one-mat-def* **using** *dimW dimaW*
by (*auto simp add*: *maWW adjoint-def*)
then have *iv0*: *adjoint W* $*$ *W* = 1_m *n* **by** *auto*
have *dimaW*: *adjoint W* $\in$ *carrier-mat n n* **using** *dimaW* **by** *auto*
then have *iv1*: *W* $*$ *adjoint W* = 1_m *n* **using** *mat-mult-left-right-inverse dimW iv0* **by** *auto*
then show *unitary W* **unfolding** *unitary-def inverts-mat-def* **using** *dimW dimaW iv0 iv1* **by** *auto*
qed

lemma *normalize-keep-eigenvector*:
assumes *ev*: *eigenvector A v e*
and *dim*: *A* $\in$ *carrier-mat n n v* $\in$ *carrier-vec n*
shows *eigenvector A* (*vec-normalize v*) *e*
unfolding *eigenvector-def*
proof
show *vec-normalize v* $\in$ *carrier-vec* (*dim-row A*) **using** *dim* **by** *auto*
have *eg*: $A *_v v = e \cdot_v v$ **using** *ev dim eigenvector-def* **by** *auto*
have *vneq0*: $v \neq 0_v\ n$ **using** *ev dim* **unfolding** *eigenvector-def* **by** *auto*
then have *s0*: *vec-normalize v* $\neq 0_v$ *n*
by (*insert dim*, *subst normalize-zero*[*of v*], *auto*)
from *vneq0* **have** *vvge0*: *vec-norm v* > *0* **using** *vec-norm-ge-0 dim* **by** *auto*
have *s1*: $A *_v$ *vec-normalize v* = $e \cdot_v$ *vec-normalize v* **unfolding** *vec-normalize-def*

using *vneq0 dim*
apply (*auto*, *simp add*: *mult-mat-vec*)
apply (*subst eg*, *auto*)
done
with *s0 dim* **show** *vec-normalize v* $\neq$ *0*$_v$ (*dim-row A*) $\wedge$ *A* $*_v$ *vec-normalize v* = *e* $\cdot_v$ *vec-normalize v* **by** *auto*
qed

lemma *four-block-mat-adjoint*:
fixes *A B C D* :: *'a::conjugatable-field mat*
assumes *dim*: *A* $\in$ *carrier-mat nr1 nc1 B* $\in$ *carrier-mat nr1 nc2*
C $\in$ *carrier-mat nr2 nc1 D* $\in$ *carrier-mat nr2 nc2*
shows *adjoint* (*four-block-mat A B C D*)
= *four-block-mat* (*adjoint A*) (*adjoint C*) (*adjoint B*) (*adjoint D*)
by (*rule eq-matI*, *insert dim*, *auto simp add*: *adjoint-eval*)

fun *unitary-schur-decomposition* :: *complex mat* $\Rightarrow$ *complex list* $\Rightarrow$ *complex mat* $\times$ *complex mat* $\times$ *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*) $\in$ *carrier-mat n n*
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 Q* $\wedge$ *upper-triangular B* $\wedge$ *diag-mat B* = *es* $\wedge$ *unitary P* $\wedge$ (*Q* = *adjoint P*)
using *assms*
proof (*induct es arbitrary*: *n A B P Q*)
case *Nil*
with *degree-monic-char-poly*[*of A n*]

show *?case* **by** (*auto intro*: *similar-mat-wit-refl simp*: *diag-mat-def unitary-zero*)
next
case (*Cons e es n A C P Q*)
let *?n1* = *n* − *1*
from *Cons* **have** *A*: *A* ∈ *carrier-mat n n* **and** *dim*: *dim-row A* = *n* **by** *auto*
let *?cp* = *char-poly A*
from *Cons*(*3*)
have *cp*: *?cp* = [: −*e*, *1* :] ∗ (∏ *e* ← *es*. [:− *e*, *1*:]) **by** *auto*
have *mon*: *monic* (∏ *e*← *es*. [:− *e*, *1*:]) **by** (*rule monic-prod-list*, *auto*)
have *deg*: *degree ?cp* = *Suc* (*degree* (∏ *e*← *es*. [:− *e*, *1*:])) **unfolding** *cp*
by (*subst degree-mult-eq*, *insert mon*, *auto*)
with *degree-monic-char-poly*[*OF A*] **have** *n*: *n* ≠ *0* **by** *auto*
define *v′* **where** *v′* = *find-eigenvector A e*
define *v* **where** *v* = *vec-normalize v′*
define *b* **where** *b* = *basis-completion v*
define *ws0* **where** *ws0* = *gram-schmidt n b*
define *ws* **where** *ws* = *map vec-normalize ws0*
define *W* **where** *W* = *mat-of-cols n ws*
define *W′* **where** *W′* = *corthogonal-inv W*
define *A′* **where** *A′* = *W′* ∗ *A* ∗ *W*
obtain *A1 A2 A0 A3* **where** *splitA′*: *split-block A′ 1 1* = (*A1*,*A2*,*A0*,*A3*)
by (*cases split-block A′ 1 1*, *auto*)
obtain *B P′ Q′* **where** *schur*: *unitary-schur-decomposition A3 es* = (*B*,*P′*,*Q′*)
by (*cases unitary-schur-decomposition A3 es*, *auto*)
let *?P′* = *four-block-mat* (1_m *1*) (0_m *1 ?n1*) (0_m *?n1 1*) *P′*
let *?Q′* = *four-block-mat* (1_m *1*) (0_m *1 ?n1*) (0_m *?n1 1*) *Q′*
have *C*: *C* = *four-block-mat A1* (*A2* ∗ *P′*) *A0 B* **and** *P*: *P* = *W* ∗ *?P′* **and** *Q*: *Q* = *?Q′* ∗ *W′*
using *Cons*(*4*) **unfolding** *unitary-schur-decomposition.simps*
Let-def list.sel dim
v′-def[*symmetric*] *v-def*[*symmetric*] *b-def*[*symmetric*] *ws0-def*[*symmetric*] *ws-def*[*symmetric*] *W′-def*[*symmetric*] *W-def*[*symmetric*]
A′-def[*symmetric*] *split splitA′ schur* **by** *auto*
have *e*: *eigenvalue A e*
unfolding *eigenvalue-root-char-poly*[*OF A*] *cp* **by** *simp*
from *find-eigenvector*[*OF A e*] **have** *ev′*: *eigenvector A v′ e* **unfolding** *v′-def* **.**
then have *v′* ∈ *carrier-vec n* **unfolding** *eigenvector-def* **using** *A* **by** *auto*
with *ev′* **have** *ev*: *eigenvector A v e* **unfolding** *v-def* **using** *A dim normalize-keep-eigenvector* **by** *auto*
from *this*[*unfolded eigenvector-def*]
have *v*[*simp*]: *v* ∈ *carrier-vec n* **and** *v0*: *v* ≠ 0_v *n* **using** *A* **by** *auto*
interpret *cof-vec-space n TYPE*(*complex*) **.**
from *basis-completion*[*OF v v0*, *folded b-def*]
have *span-b*: *span* (*set b*) = *carrier-vec n* **and** *dist-b*: *distinct b*
and *indep*: ¬ *lin-dep* (*set b*) **and** *b*: *set b* ⊆ *carrier-vec n* **and** *hdb*: *hd b* = *v*
and *len-b*: *length b* = *n* **by** *auto*
from *hdb len-b n* **obtain** *vs* **where** *bv*: *b* = *v* # *vs* **by** (*cases b*, *auto*)
from *gram-schmidt-result*[*OF b dist-b indep refl*, *folded ws0-def*]
have *ws0*: *set ws0* ⊆ *carrier-vec n corthogonal ws0 length ws0* = *n*

by (*auto simp*: *len-b*)
then have *ws*: *set ws* ⊆ *carrier-vec n corthogonal ws length ws* = *n* **unfolding** *ws-def*
using *normalize-keep-corthogonal* **by** *auto*
have *ws0ne*: *ws0* ≠ [] **using** ‹*length ws0* = *n*› *n* **by** *auto*
from *gram-schmidt-hd*[*OF v*, *of vs*, *folded bv*] **have** *hdws0*: *hd ws0* = (*vec-normalize v′*) **unfolding** *ws0-def v-def* **.**
have *hd ws* = *vec-normalize* (*hd ws0*) **unfolding** *ws-def* **using** *hd-map*[*OF ws0ne*] **by** *auto*
then have *hdws*: *hd ws* = *v* **unfolding** *v-def* **using** *normalize-normalize*[*of v′*] *hdws0* **by** *auto*
have *orth-W*: *corthogonal-mat W* **using** *orthogonal-mat-of-cols ws* **unfolding** *W-def* **.**
have *W*: *W* ∈ *carrier-mat n n*
using *ws* **unfolding** *W-def* **using** *mat-of-cols-carrier*(*1*)[*of n ws*] **by** *auto*
have *W′*: *W′* ∈ *carrier-mat n n* **unfolding** *W′-def corthogonal-inv-def* **using** *W*
by (*auto simp*: *mat-of-rows-def*)
from *corthogonal-inv-result*[*OF orth-W*]
have *W′W*: *inverts-mat W′ W* **unfolding** *W′-def* **.**
hence *WW′*: *inverts-mat W W′* **using** *mat-mult-left-right-inverse*[*OF W′ W*] *W′ W*
unfolding *inverts-mat-def* **by** *auto*
have *A′*: *A′* ∈ *carrier-mat n n* **using** *W W′ A* **unfolding** *A′-def* **by** *auto*
have *A′A-wit*: *similar-mat-wit A′ A W′ W*
by (*rule similar-mat-witI*[*of - - n*], *insert W W′ A A′ W′W WW′*, *auto simp*: *A′-def*
inverts-mat-def)
hence *A′A*: *similar-mat A′ A* **unfolding** *similar-mat-def* **by** *blast*
from *similar-mat-wit-sym*[*OF A′A-wit*] **have** *simAA′*: *similar-mat-wit A A′ W W′* **by** *auto*
have *eigen*[*simp*]: $A *_v v = e \cdot_v v$ **and** *v0*: $v \neq 0_v\ n$
using *v-def v′-def find-eigenvector*[*OF A e*] *A normalize-keep-eigenvector*
unfolding *eigenvector-def* **by** *auto*
let *?f* = (λ *i*. *if i* = *0 then e else 0*)
have *col0*: *col A′ 0* = *vec n ?f*
unfolding *A′-def W′-def W-def*
using *corthogonal-col-ev-0*[*OF A v v0 eigen n hdws ws*]**.**
from *A′ n* **have** *dim-row A′* = *1* + *?n1 dim-col A′* = *1* + *?n1* **by** *auto*
from *split-block*[*OF splitA′ this*] **have** *A2*: *A2* ∈ *carrier-mat 1 ?n1*
and *A3*: *A3* ∈ *carrier-mat ?n1 ?n1*
and *A′block*: *A′* = *four-block-mat A1 A2 A0 A3* **by** *auto*
have *A1id*: *A1* = *mat 1 1* (λ -. *e*)
using *splitA′*[*unfolded split-block-def Let-def*] *arg-cong*[*OF col0*, *of* λ *v*. *v* $ *0*] *A′ n*
by (*auto simp*: *col-def*)
have *A1*: *A1* ∈ *carrier-mat 1 1* **unfolding** *A1id* **by** *auto*
{
fix *i*

assume $i < ?n1$
with *arg-cong*[*OF col0*, *of* λ *v*. *v* $ *Suc i*] *A′*
have *A′* $$ (*Suc i*, *0*) = *0* **by** *auto*
} **note** *A′0* = *this*
have *A0id*: *A0* = 0_m *?n1 1*
using *splitA′*[*unfolded split-block-def Let-def*] *A′0 A′* **by** *auto*
have *A0*: *A0* ∈ *carrier-mat ?n1 1* **unfolding** *A0id* **by** *auto*
from *cp char-poly-similar*[*OF A′A*]
have *cp*: *char-poly A′* = [: −*e*,*1* :] ∗ (∏ *e* ← *es*. [:− *e*, *1*:]) **by** *simp*
also have *char-poly A′* = *char-poly A1* ∗ *char-poly A3*
unfolding *A′block A0id*
by (*rule char-poly-four-block-zeros-col*[*OF A1 A2 A3*])
also have *char-poly A1* = [: −*e*,*1* :]
by (*simp add*: *A1id char-poly-defs det-def*)
finally have *cp*: *char-poly A3* = (∏ *e* ← *es*. [:− *e*, *1*:])
by (*metis mult-cancel-left pCons-eq-0-iff zero-neq-one*)
from *Cons*(*1*)[*OF A3 cp schur*]
have *simIH*: *similar-mat-wit A3 B P′ Q′* **and** *ut*: *upper-triangular B* **and** *diag*: *diag-mat B* = *es*
and *uP′*: *unitary P′* **and** *Q′P′*: *Q′* = *adjoint P′*
by *auto*
from *similar-mat-witD2*[*OF A3 simIH*]
have *B*: *B* ∈ *carrier-mat ?n1 ?n1* **and** *P′*: *P′* ∈ *carrier-mat ?n1 ?n1* **and** *Q′*: *Q′* ∈ *carrier-mat ?n1 ?n1*
and *PQ′*: *P′* ∗ *Q′* = 1_m *?n1* **by** *auto*
have *A0-eq*: *A0* = *P′* ∗ *A0* ∗ 1_m *1* **unfolding** *A0id* **using** *P′* **by** *auto*
have *simA′C*: *similar-mat-wit A′ C ?P′ ?Q′* **unfolding** *A′block C*
by (*rule similar-mat-wit-four-block*[*OF similar-mat-wit-refl*[*OF A1*] *simIH* - *A0-eq A1 A3 A0*],
insert PQ′ A2 P′ Q′, *auto*)
have *ut1*: *upper-triangular A1* **unfolding** *A1id* **by** *auto*
have *ut*: *upper-triangular C* **unfolding** *C A0id*
by (*intro upper-triangular-four-block*[*OF* - *B ut1 ut*], *auto simp*: *A1id*)
from *A1id* **have** *diagA1*: *diag-mat A1* = [*e*] **unfolding** *diag-mat-def* **by** *auto*
from *diag-four-block-mat*[*OF A1 B*] **have** *diag*: *diag-mat C* = *e* # *es* **unfolding** *diag diagA1 C* **by** *simp*

have *aW*: *adjoint W* ∈ *carrier-mat n n* **using** *W* **by** *auto*
have *aW′*: *adjoint W′* ∈ *carrier-mat n n* **using** *W′* **by** *auto*
have *unitary W* **using** *W-def ws-def ws0 normalized-corthogonal-mat-is-unitary* **by** *auto*
then have *ivWaW*: *inverts-mat W* (*adjoint W*) **using** *unitary-def W aW* **by** *auto*
with *WW′* **have** *W′aW*: *W′* = (*adjoint W*) **using** *inverts-mat-unique W W′ aW* **by** *auto*
then have *adjoint W′* = *W* **using** *adjoint-adjoint* **by** *auto*
with *ivWaW* **have** *inverts-mat W′* (*adjoint W′*) **using** *inverts-mat-symm W aW W′aW* **by** *auto*
then have *unitary W′* **using** *unitary-def W′* **by** *auto*

have *newP′*: $P' \in$ *carrier-mat* $(n - Suc\ 0)$ $(n - Suc\ 0)$ **using** P' **by** *auto*
have *rl*: $\bigwedge$ *x1 x2 x3 x4 y1 y2 y3 y4 f*. $x1 = y1 \Longrightarrow x2 = y2 \Longrightarrow x3 = y3 \Longrightarrow x4 = y4 \Longrightarrow f\ x1\ x2\ x3\ x4 = f\ y1\ y2\ y3\ y4$ **by** *simp*
have *Q′aP′*: $?Q' =$ *adjoint* $?P'$
apply (*subst four-block-mat-adjoint*, *auto simp add*: *newP′*)
apply (*rule rl*[**where** *f2* = *four-block-mat*])
apply (*auto simp add*: *eq-matI adjoint-eval Q′P′*)
done
have *adjoint P* = *adjoint ?P′* ∗ *adjoint W* **using** *W newP′ n*
apply (*simp add*: *P*)
apply (*subst adjoint-mult*[*of W*, *symmetric*])
apply (*auto simp add*: *W P′ carrier-matD*[*of W n n*])
done
also have ... = *?Q′* ∗ *W′* **using** *Q′aP′ W′aW* **by** *auto*
also have ... = *Q* **using** *Q* **by** *auto*
finally have *QaP*: *Q* = *adjoint P* **..**

from *similar-mat-wit-trans*[*OF simAA′ simA′C*, *folded P Q*] **have** *smw*: *similar-mat-wit A C P Q* **by** *blast*
then have *dimP*: $P \in$ *carrier-mat n n* **and** *dimQ*: $Q \in$ *carrier-mat n n* **unfolding** *similar-mat-wit-def* **using** *A* **by** *auto*
from *smw* **have** $P * Q = 1_m\ n$ **unfolding** *similar-mat-wit-def* **using** *A* **by** *auto*
then have *inverts-mat P Q* **using** *inverts-mat-def dimP* **by** *auto*
then have *uP*: *unitary P* **using** *QaP unitary-def dimP* **by** *auto*

from *ut similar-mat-wit-trans*[*OF simAA′ simA′C*, *folded P Q*] *diag uP QaP*
show *?case* **by** *blast*
qed

lemma *complex-mat-char-poly-factorizable*:
fixes *A* :: *complex mat*
assumes $A \in$ *carrier-mat n n*
shows ∃ *as*. *char-poly A* = (∏ *a* ← *as*. [:− *a*, *1*:]) ∧ *length as* = *n*
proof −
let *?ca* = *char-poly A*
have *ex0*: ∃ *bs*. *Polynomial.smult* (*lead-coeff ?ca*) (∏ *b*←*bs*. [:− *b*, *1*:]) = *?ca* ∧
length bs = *degree ?ca*
by (*simp add*: *fundamental-theorem-algebra-factorized*)
then obtain *bs* **where** *Polynomial.smult* (*lead-coeff ?ca*) (∏ *b*←*bs*. [:− *b*, *1*:]) = *?ca* ∧
length bs = *degree ?ca* **by** *auto*
moreover have *lead-coeff ?ca* = (*1*::*complex*)
using *assms degree-monic-char-poly* **by** *blast*
ultimately have *ex1*: *?ca* = (∏ *b*←*bs*. [:− *b*, *1*:]) ∧ *length bs* = *degree ?ca* **by** *auto*
moreover have *degree ?ca* = *n*
by (*simp add*: *assms degree-monic-char-poly*)

ultimately show *?thesis* **by** *auto*
qed

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 −
have $\exists$ *es. char-poly A* = ($\prod$ *e* $\leftarrow$ *es.* [:− *e*, *1*:]) $\wedge$ *length es* = *n*
using *assms* **by** (*simp add*: *complex-mat-char-poly-factorizable*)
then obtain *es* **where** *es*: *char-poly A* = ($\prod$ *e* $\leftarrow$ *es.* [:− *e*, *1*:]) $\wedge$ *length es* = *n* **by** *auto*
obtain *B P Q* **where** *B*: *unitary-schur-decomposition A es* = (*B*,*P*,*Q*) **by** (*cases unitary-schur-decomposition A es*, *auto*)

have *similar-mat-wit A B P Q* $\wedge$ *upper-triangular B* $\wedge$ *unitary P* $\wedge$ (*Q* = *adjoint P*) $\wedge$
char-poly A = ($\prod$ (*e* :: *complex*) $\leftarrow$ *es.* [:− *e*, *1*:]) $\wedge$ *diag-mat B* = *es* **using** *assms es B*
by (*auto simp add*: *unitary-schur-decomposition*)
then show *?thesis* **by** *auto*
qed

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 (*rule disjE*[*of n* = *0 n* > *0*], *blast*)
have *dim*: *dim-row A* = *n dim-col A* = *n* **using** *assms* **by** *auto*
from *norm* **have** *eq0*: $\bigwedge$*i j.* (*A* ∗ *adjoint A*)$$(*i*,*j*) = (*adjoint A* ∗ *A*)$$(*i*,*j*) **by** *auto*
have *nat-induct-strong*:
$\bigwedge$*P.* (*P*::*nat*⇒*bool*) *0* ⟹ ($\bigwedge$*i. i* < *n* ⟹ ($\bigwedge$*k. k* < *i* ⟹ *P k*) ⟹ *P i*) ⟹ ($\bigwedge$*i. i* < *n* ⟹ *P i*)
by (*metis dual-order.strict-trans infinite-descent0 linorder-neqE-nat*)
show *n* = *0* ⟹ *?thesis* **using** *dim* **unfolding** *diagonal-mat-def* **by** *auto*
show *n* > *0* ⟹ *?thesis* **unfolding** *diagonal-mat-def dim*
apply (*rule allI*, *rule impI*)
apply (*rule nat-induct-strong*)
proof (*rule allI*, *rule impI*, *rule impI*)
assume *asm*: *n* > *0*
from *tri upper-triangularD*[*of A 0 j*] *dim* **have** *z0*: $\bigwedge$*j. 0*< *j* ⟹ *j* < *n* ⟹ *A*$$(*j*, *0*) = *0*
by *auto*
then have *ada00*: (*adjoint A* ∗ *A*)$$(*0*,*0*) = *conjugate* (*A*$$(*0*,*0*)) ∗ *A*$$(*0*,*0*)
using *asm dim* **by** (*auto simp add*: *scalar-prod-def adjoint-eval sum.atLeast-Suc-lessThan*)

```
have aad00: (A ∗ adjoint A)$$(0,0) = (∑k=0..<n. A$$(0, k) ∗ conjugate
(A$$(0, k)))
  using asm dim by (auto simp add: scalar-prod-def adjoint-eval)
moreover have
  . . . = A$$(0,0) ∗ conjugate (A$$(0,0))
      + (∑k=1..<n. A$$(0, k) ∗ conjugate (A$$(0, k)))
   using dim asm by (subst sum.atLeast-Suc-lessThan[of 0 n λk. A$$(0, k) ∗
conjugate (A$$(0, k))], auto)
ultimately have f1tneq0: (∑k=(Suc 0)..<n. A$$(0, k) ∗ conjugate (A$$(0,
k))) = 0
  using eq0 ada00 by (simp)
have geq0: ⋀k. k < n ⟹ A$$(0, k) ∗ conjugate (A$$(0, k)) ≥ 0
  using conjugate-square-positive by auto
have ⋀k. 1 ≤ k ⟹ k < n ⟹ A$$(0, k) ∗ conjugate (A$$(0, k)) = 0
  by (rule sum-nonneg-0[of {1..<n}], auto, rule geq0, auto, rule f1tneq0)
with dim asm show
  case0: ⋀j. 0 < n ⟹ j < n ⟹ 0 ≠ j ⟹ A $$ (0, j) = 0
  by auto
{
  fix i
  assume asm: n > 0 i < n i > 0
    and ih: ⋀k. k < i ⟹ ∀j<n. k ≠ j ⟶ A $$ (k, j) = 0
  then have ⋀j. j<n ⟹ i ≠ j ⟹ A $$ (i, j) = 0
  proof −
    have inter-part: ⋀b m e. (b::nat) < e ⟹ b < m ⟹ m < e ⟹ {b..<m}
∪ {m..<e} = {b..<e} by auto
    then have
      ⋀b m e f. (b::nat) < e ⟹ b < m ⟹ m < e
        ⟹ (∑k=b..<e. f k) = (∑k∈{b..<m}∪{m..<e}. f k)
      using sum.union-disjoint by auto
    then have sum-part:
      ⋀b m e f. (b::nat) < e ⟹ b < m ⟹ m < e
                ⟹ (∑k=b..<e. f k) = (∑k=b..<m. f k) + (∑k=m..<e. f k)
      by (auto simp add: sum.union-disjoint)
    from tri upper-triangularD[of A j i] asm dim have
        zsi0: ⋀j. j < i ⟹ A$$(i, j) = 0 by auto
    from tri upper-triangularD[of A j i] asm dim have
        zsi1: ⋀k. i < k ⟹ k < n ⟹ A$$(k, i) = 0 by auto
    have
      (A ∗ adjoint A)$$(i, i)
      = (∑k=0..<n. conjugate (A$$(i, k)) ∗ A$$(i, k)) using asm dim
      apply (auto simp add: scalar-prod-def adjoint-eval)
      apply (rule sum.cong, auto)
      done
    also have
      . . . = (∑k=0..<i. conjugate (A$$(i, k)) ∗ A$$(i, k))
          + (∑k=i..<n. conjugate (A$$(i, k)) ∗ A$$(i, k))
      using asm
      by (auto simp add: sum-part[of 0 n i])
```

also have
... = (∑ *k*=*i*..<*n*. *conjugate* (*A*$$(*i*, *k*)) ∗ *A*$$(*i*, *k*))
using *zsi0*
by *auto*
also have
... = *conjugate* (*A*$$(*i*, *i*)) ∗ *A*$$(*i*, *i*)
+ (∑ *k*=(*Suc i*)..<*n*. *conjugate* (*A*$$(*i*, *k*)) ∗ *A*$$(*i*, *k*))
using *asm*
by (*auto simp add*: *sum.atLeast-Suc-lessThan*)
finally have
adaii: (*A* ∗ *adjoint A*)$$(*i*, *i*)
= *conjugate* (*A*$$(*i*, *i*)) ∗ *A*$$(*i*, *i*)
+ (∑ *k*=(*Suc i*)..<*n*. *conjugate* (*A*$$(*i*, *k*)) ∗ *A*$$(*i*, *k*)) **.**
have
(*adjoint A* ∗ *A*)$$(*i*, *i*) = (∑ *k*=*0*..<*n*. *conjugate* (*A*$$(*k*, *i*)) ∗ *A*$$(*k*, *i*))
using *asm dim* **by** (*auto simp add*: *scalar-prod-def adjoint-eval*)
also have
... = (∑ *k*=*0*..<*i*. *conjugate* (*A*$$(*k*, *i*)) ∗ *A*$$(*k*, *i*))
+ (∑ *k*=*i*..<*n*. *conjugate* (*A*$$(*k*, *i*)) ∗ *A*$$(*k*, *i*))
using *asm* **by** (*auto simp add*: *sum-part*[*of 0 n i*])
also have
... = (∑ *k*=*i*..<*n*. *conjugate* (*A*$$(*k*, *i*)) ∗ *A*$$(*k*, *i*))
using *asm ih* **by** *auto*
also have
... = *conjugate* (*A*$$(*i*, *i*)) ∗ *A*$$(*i*, *i*)
using *asm zsi1* **by** (*auto simp add*: *sum.atLeast-Suc-lessThan*)
finally have (*adjoint A* ∗ *A*)$$(*i*, *i*) = *conjugate* (*A*$$(*i*, *i*)) ∗ *A*$$(*i*, *i*) **.**
with *adaii eq0* **have**
fsitoneq0: (∑ *k*=(*Suc i*)..<*n*. *conjugate* (*A*$$(*i*, *k*)) ∗ *A*$$(*i*, *k*)) = *0* **by**
auto
have ⋀*k*. *k*<*n* ⟹ *i* < *k* ⟹ *conjugate* (*A*$$(*i*, *k*)) ∗ *A*$$(*i*, *k*) = *0*
by (*rule sum-nonneg-0*[*of* {(*Suc i*)..<*n*}], *auto*, *subst mult.commute*,
rule conjugate-square-positive, *rule fsitoneq0*)
then have ⋀*k*. *k*<*n* ⟹ *i*<*k* ⟹ *A* $$ (*i*, *k*) = *0* **by** *auto*
with *zsi0* **show** ⋀*j*. *j*<*n* ⟹ *i* ≠ *j* ⟹ *A* $$ (*i*, *j*) = *0*
by (*metis linorder-neqE-nat*)
qed
}
with *case0* **show** ⋀*i ia*.
0 < *n* ⟹
i < *n* ⟹
ia < *n* ⟹
(⋀*k*. *k* < *ia* ⟹ ∀ *j*<*n*. *k* ≠ *j* ⟶ *A* $$ (*k*, *j*) = *0*) ⟹
∀ *j*<*n*. *ia* ≠ *j* ⟶ *A* $$ (*ia*, *j*) = *0* **by** *auto*
qed
qed

lemma *normal-complex-mat-has-spectral-decomposition*:
assumes *A*: (*A*::*complex mat*) ∈ *carrier-mat n n*

and *normal*: $A * adjoint\ A\ = adjoint\ A * A$
and *c*: $char\text{-}poly\ A = (\prod\ (e :: complex) \leftarrow es.\ [:-\ e,\ 1:])$
and *B*: *unitary-schur-decomposition A es* = (*B*,*P*,*Q*)
shows *similar-mat-wit A B P* (*adjoint P*) ∧ *diagonal-mat B* ∧ *diag-mat B* = *es* ∧ *unitary P*
proof −
have *smw*: *similar-mat-wit A B P* (*adjoint P*)
and *ut*: *upper-triangular B*
and *uP*: *unitary P*
and *dB*: *diag-mat B* = *es*
and (*Q* = *adjoint P*)
using *assms* **by** (*auto simp add*: *unitary-schur-decomposition*)
from *smw* **have** *dimP*: *P* ∈ *carrier-mat n n* **and** *dimB*: *B* ∈ *carrier-mat n n*
and *dimaP*: *adjoint P* ∈ *carrier-mat n n*
unfolding *similar-mat-wit-def* **using** *A* **by** *auto*
have *dimaB*: *adjoint B* ∈ *carrier-mat n n* **using** *dimB* **by** *auto*
note *dims* = *dimP dimB dimaP dimaB*

have *inverts-mat P* (*adjoint P*) **using** *unitary-def uP dims* **by** *auto*
then have *iaPP*: *inverts-mat* (*adjoint P*) *P* **using** *inverts-mat-symm* **using** *dims* **by** *auto*
have *aPP*: $adjoint\ P * P = 1_m\ n$ **using** *dims iaPP* **unfolding** *inverts-mat-def* **by** *auto*
from *smw* **have** *A*: *A* = *P* ∗ *B* ∗ (*adjoint P*) **unfolding** *similar-mat-wit-def Let-def* **by** *auto*
then have *aA*: *adjoint A* = *P* ∗ *adjoint B* ∗ *adjoint P*
by (*insert A dimP dimB dimaP*, *auto simp add*: *adjoint-mult*[*of - n n - n*] *adjoint-adjoint*)
have *A* ∗ *adjoint A* = (*P* ∗ *B* ∗ *adjoint P*) ∗ (*P* ∗ *adjoint B* ∗ *adjoint P*) **using** *A aA* **by** *auto*
also have . . . = *P* ∗ *B* ∗ (*adjoint P* ∗ *P*) ∗ (*adjoint B* ∗ *adjoint P*) **using** *dims* **by** (*mat-assoc n*)
also have . . . = $P * B * 1_m\ n * (adjoint\ B * adjoint\ P)$ **using** *dims aPP* **by** (*auto*)
also have . . . = *P* ∗ *B* ∗ *adjoint B* ∗ *adjoint P* **using** *dims* **by** (*mat-assoc n*)
finally have *A* ∗ *adjoint A* = *P* ∗ *B* ∗ *adjoint B* ∗ *adjoint P*.
then have *adjoint P* ∗ (*A* ∗ *adjoint A*) ∗ *P* = (*adjoint P* ∗ *P*) ∗ *B* ∗ *adjoint B* ∗ (*adjoint P* ∗ *P*)
using *dims* **by** (*simp add*: *assoc-mult-mat*[*of - n n - n - n*])
also have . . . = $1_m\ n * B * adjoint\ B * 1_m\ n$ **using** *aPP* **by** *auto*
also have . . . = *B* ∗ *adjoint B* **using** *dims* **by** *auto*
finally have *eq0*: *adjoint P* ∗ (*A* ∗ *adjoint A*) ∗ *P* = *B* ∗ *adjoint B*.

have *adjoint A* ∗ *A* = (*P* ∗ *adjoint B* ∗ *adjoint P*) ∗ (*P* ∗ *B* ∗ *adjoint P*) **using** *A aA* **by** *auto*
also have . . . = *P* ∗ *adjoint B* ∗ (*adjoint P* ∗ *P*) ∗ (*B* ∗ *adjoint P*) **using** *dims* **by** (*mat-assoc n*)
also have . . . = $P * adjoint\ B * 1_m\ n * (B * adjoint\ P)$ **using** *dims aPP* **by** (*auto*)

also have ... = *P* ∗ *adjoint B* ∗ *B* ∗ *adjoint P* **using** *dims* **by** (*mat-assoc n*)
finally have *adjoint A* ∗ *A* = *P* ∗ *adjoint B* ∗ *B* ∗ *adjoint P* **by** *auto*
then have *adjoint P* ∗ (*adjoint A* ∗ *A*) ∗ *P* = (*adjoint P* ∗ *P*) ∗ *adjoint B* ∗ *B* ∗ (*adjoint P* ∗ *P*)
using *dims* **by** (*simp add*: *assoc-mult-mat*[*of - n n - n - n*])
also have ... = 1_m *n* ∗ *adjoint B* ∗ *B* ∗ 1_m *n* **using** *aPP* **by** *auto*
also have ... = *adjoint B* ∗ *B* **using** *dims* **by** *auto*
finally have *eq1*: *adjoint P* ∗ (*adjoint A* ∗ *A*) ∗ *P* = *adjoint B* ∗ *B*.

from *normal* **have** *adjoint P* ∗ (*adjoint A* ∗ *A*) ∗ *P* = *adjoint P* ∗ (*A* ∗ *adjoint A*) ∗ *P* **by** *auto*
with *eq0 eq1* **have** *B* ∗ *adjoint B* = *adjoint B* ∗ *B* **by** *auto*
with *ut dims* **have** *diagonal-mat B* **using** *normal-upper-triangular-matrix-is-diagonal* **by** *auto*
with *smw uP dB* **show** *similar-mat-wit A B P* (*adjoint P*) ∧ *diagonal-mat B* ∧ *diag-mat B* = *es* ∧ *unitary P* **by** *auto*
qed

lemma *complex-mat-has-jordan-nf*:
fixes *A* :: *complex mat*
assumes *A* ∈ *carrier-mat n n*
shows ∃ *n-as*. *jordan-nf A n-as*
proof −
have ∃ *as*. *char-poly A* = (∏ *a* ← *as*. [:− *a*, *1*:]) ∧ *length as* = *n*
using *assms* **by** (*simp add*: *complex-mat-char-poly-factorizable*)
then show *?thesis* **using** *assms*
by (*auto simp add*: *jordan-nf-iff-linear-factorization*)
qed

lemma *hermitian-is-normal*:
assumes *hermitian A*
shows *A* ∗ *adjoint A* = *adjoint A* ∗ *A*
using *assms* **by** (*auto simp add*: *hermitian-def*)

lemma *hermitian-eigenvalue-real*:
assumes *dim*: (*A*::*complex mat*) ∈ *carrier-mat n n*
and *hA*: *hermitian A*
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* (*adjoint P*) ∧ *diagonal-mat B* ∧ *diag-mat B* = *es* ∧ *unitary P* ∧ (∀ *i* < *n*. *B*$$(*i*, *i*) ∈ *Reals*)
proof −
have *normal*: *A* ∗ *adjoint A* = *adjoint A* ∗ *A* **using** *hA hermitian-is-normal* **by** *auto*
then have *schur*: *similar-mat-wit A B P* (*adjoint P*) ∧ *diagonal-mat B* ∧ *diag-mat B* = *es* ∧ *unitary P*
using *normal-complex-mat-has-spectral-decomposition*[*OF dim normal c B*] **by** (*simp*)
then have *similar-mat-wit A B P* (*adjoint P*)

```
    and uP: unitary P and dB: diag-mat B = es
    using assms by auto
  then have A: A = P ∗ B ∗ (adjoint P)
    and dimB: B ∈ carrier-mat n n and dimP: P ∈ carrier-mat n n
    unfolding similar-mat-wit-def Let-def using dim by auto
  then have dimaB: adjoint B ∈ carrier-mat n n by auto
  have adjoint A = adjoint (adjoint P) ∗ adjoint (P ∗ B)
    apply (subst A)
    apply (subst adjoint-mult[of P ∗ B n n adjoint P n])
      apply (insert dimB dimP, auto)
    done
  also have ... = P ∗ adjoint (P ∗ B) by (auto simp add: adjoint-adjoint)
  also have ... = P ∗ (adjoint B ∗ adjoint P) using dimB dimP by (auto simp
add: adjoint-mult)
   also have ... = P ∗ adjoint B ∗ adjoint P using dimB dimP by (subst as-
soc-mult-mat[symmetric, of P n n adjoint B n adjoint P n], auto)
  finally have aA: adjoint A = P ∗ adjoint B ∗ adjoint P .
  have A = adjoint A using hA hermitian-def[of A] by auto
  then have P ∗ B ∗ adjoint P = P ∗ adjoint B ∗ adjoint P using A aA by auto
  then have BaB: B = adjoint B using unitary-elim[OF dimB dimaB dimP] uP
by auto
  {
    fix i
    assume i < n
    then have B$$(i, i) = conjugate (B$$(i, i))
      apply (subst BaB)
      by (insert dimB, simp add: adjoint-eval)
    then have B$$(i, i) ∈ Reals unfolding conjugate-complex-def
      using Reals-cnj-iff by auto
  }
  then have ∀ i<n. B$$(i, i) ∈ Reals by auto
  with schur show ?thesis by auto
qed

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 −
  obtain es where es: char-poly A = (∏ (e :: complex) ← es. [:− e, 1:])
    using complex-mat-char-poly-factorizable dimA by auto
  obtain B P Q where unitary-schur-decomposition A es = (B,P,Q)
    by (cases unitary-schur-decomposition A es, auto)
  then have similar-mat-wit A B P (adjoint P) ∧ diagonal-mat B ∧ diag-mat B
= es
    ∧ unitary P ∧ (∀ i < n. B$$(i, i) ∈ Reals)
    using hermitian-eigenvalue-real dimA es hA by auto
  then have A: A = P ∗ B ∗ (adjoint P) and dB: diagonal-mat B
```

and *Bii*: $\bigwedge i.\ i < n \Longrightarrow B\$\$(i, i) \in Reals$
and *dimB*: $B \in$ *carrier-mat n n* **and** *dimP*: $P \in$ *carrier-mat n n* **and** *dimaP*: *adjoint P* $\in$ *carrier-mat n n*
unfolding *similar-mat-wit-def Let-def* **using** *dimA* **by** *auto*
define *w* **where** $w = (adjoint\ P) *_v v$
then have *dimw*: $w \in$ *carrier-vec n* **using** *dimaP dimv* **by** *auto*
from *A* **have** *inner-prod v* $(A *_v v) =$ *inner-prod v* $((P * B * (adjoint\ P)) *_v v)$ **by** *auto*
also have $\ldots =$ *inner-prod v* $((P * B) *_v ((adjoint\ P) *_v v))$ **using** *dimP dimB dimv*
by (*subst assoc-mult-mat-vec*[*of - n n adjoint P n*], *auto*)
also have $\ldots =$ *inner-prod v* $(P *_v (B *_v ((adjoint\ P) *_v v)))$ **using** *dimP dimB dimv dimaP*
by (*subst assoc-mult-mat-vec*[*of - n n B n*], *auto*)
also have $\ldots =$ *inner-prod w* $(B *_v w)$ **unfolding** *w-def*
apply (*rule adjoint-def-alter*[*OF - - dimP*])
apply (*insert mult-mat-vec-carrier*[*OF dimB mult-mat-vec-carrier*[*OF dimaP dimv*]], *auto simp add*: *dimv*)
done

also have $\ldots = (\sum i=0..<n.\ (\sum j=0..<n.$
$conjugate\ (w\$i) * B\$\$(i, j) * w\$j))$ **unfolding** *scalar-prod-def* **using** *dimw dimB*
apply (*simp add*: *scalar-prod-def sum-distrib-right*)
apply (*rule sum.cong*, *auto*, *rule sum.cong*, *auto*)
done
also have $\ldots = (\sum i=0..<n.\ B\$\$(i, i) *\ conjugate\ (w\$i) * w\$i)$
apply (*rule sum.cong*, *auto*)
apply (*simp add*: *sum.remove*)
apply (*insert dB*[*unfolded diagonal-mat-def*] *dimB*, *auto*)
done
finally have *sum*: *inner-prod v* $(A *_v v) = (\sum i=0..<n.\ B\$\$(i, i) *\ conjugate\ (w\$i) * w\$i)$ **.**
have $\bigwedge i.\ i < n \Longrightarrow B\$\$(i, i) *\ conjugate\ (w\$i) * w\$i \in Reals$ **using** *Bii* **by** (*simp add*: *Reals-cnj-iff*)
then have $(\sum i=0..<n.\ B\$\$(i, i) *\ conjugate\ (w\$i) * w\$i) \in Reals$ **by** *auto*
then show *?thesis* **using** *sum* **by** *auto*
qed

lemma *unit-vec-bracket*:
fixes *A* :: *complex mat*
assumes *dimA*: $A \in$ *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 −
define *w* **where** (*w*::*complex vec*) = *unit-vec n i*
have $A *_v w =$ *col A i* **using** *i dimA w-def* **by** *auto*
then have *1*: *inner-prod w* $(A *_v w) =$ *inner-prod w* (*col A i*) **using** *w-def* **by** *auto*
have *conjugate w = w* **unfolding** *w-def unit-vec-def conjugate-vec-def* **using** *i*

by *auto*
  then have *2*: *inner-prod w* (*col A i*) = *A*$$(*i*, *i*) **using** *i dimA w-def* **by** *auto*
  from *1 2* **show** *inner-prod w* (*A* $*_v$ *w*) = *A*$$(*i*, *i*) **by** *auto*
qed

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*$$(*i*, *i*)
proof −
  have *dimaP*: *adjoint P*∈ *carrier-mat n n* **using** *dimP* **by** *auto*
  have *uaP*: *unitary* (*adjoint P*) **using** *unitary-adjoint uP dimP* **by** *auto*
  then have *inverts-mat* (*adjoint P*) *P* **by** (*simp add*: *unitary-def adjoint-adjoint*)
  then have *iv*: (*adjoint P*) ∗ *P* = 1_m *n* **using** *dimaP inverts-mat-def* **by** *auto*
  define *v* **where** *v* = *col P i*
  then have *dimv*: *v* ∈ *carrier-vec n* **using** *dimP* **by** *auto*
  define *w* **where** (*w*::*complex vec*) = *unit-vec n i*
  then have *dimw*: *w* ∈ *carrier-vec n* **by** *auto*
  have *BaPv*: *B* $*_v$ (*adjoint P* $*_v$ *v*) ∈ *carrier-vec n* **using** *dimB dimaP dimv* **by** *auto*
  have (*adjoint P*) $*_v$ *v* = (*col* (*adjoint P* ∗ *P*) *i*)
    by (*simp add*: *col-mult2*[*OF dimaP dimP i*, *symmetric*] *v-def*)
  then have *aPv*: (*adjoint P*) $*_v$ *v* = *w*
    by (*auto simp add*: *iv i w-def*)
  have *inner-prod v* (*P* ∗ *B* ∗ (*adjoint P*) $*_v$ *v*) = *inner-prod v* ((*P* ∗ *B*) $*_v$ ((*adjoint P*) $*_v$ *v*)) **using** *dimP dimB dimv*
    by (*subst assoc-mult-mat-vec*[*of* - *n n adjoint P n*], *auto*)
  also have ... = *inner-prod v* (*P* $*_v$ (*B* $*_v$ ((*adjoint P*) $*_v$ *v*))) **using** *dimP dimB dimv dimaP*
    by (*subst assoc-mult-mat-vec*[*of* - *n n B n*], *auto*)
  also have ... = *inner-prod* (*adjoint P* $*_v$ *v*) (*B* $*_v$ (*adjoint P* $*_v$ *v*))
    by (*simp add*: *adjoint-def-alter*[*OF dimv BaPv dimP*])
  also have ... = *inner-prod w* (*B* $*_v$ *w*) **using** *aPv* **by** *auto*
  also have ... = *B*$$(*i*, *i*) **using** *w-def unit-vec-bracket dimB i* **by** *auto*
  finally show *inner-prod v* (*P* ∗ *B* ∗ (*adjoint P*) $*_v$ *v*) = *B*$$(*i*, *i*).
qed

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 −
  obtain *es* **where** *es*: *char-poly A* = (∏ (*e* :: *complex*) ← *es*. [:− *e*, *1*:])
    using *complex-mat-char-poly-factorizable dimA* **by** *auto*
  obtain *B P Q* **where** *unitary-schur-decomposition A es* = (*B*,*P*,*Q*)
    by (*cases unitary-schur-decomposition A es*, *auto*)
  then have *similar-mat-wit A B P* (*adjoint P*) ∧ *diagonal-mat B* ∧ *diag-mat B*

```
= es
    ∧ unitary P ∧ (∀ i < n. B$$(i, i) ∈ Reals)
    using hermitian-eigenvalue-real dimA es hA by auto
  then have A: A = P * B * (adjoint P) and dB: diagonal-mat B
    and Bii: ⋀i. i < n ⟹ B$$(i, i) ∈ Reals
    and dimB: B ∈ carrier-mat n n and dimP: P ∈ carrier-mat n n and dimaP:
adjoint P ∈ carrier-mat n n
    and uP: unitary P
    unfolding similar-mat-wit-def Let-def unitary-def using dimA by auto
  then have uaP: unitary (adjoint P) using unitary-adjoint by auto
  then have inverts-mat (adjoint P) P by (simp add: unitary-def adjoint-adjoint)
  then have iv: adjoint P * P = 1m n using dimaP inverts-mat-def by auto
  have B = 0m n n
  proof−
    {
      fix i assume i: i < n
      define v where v = col P i
      then have dimv: v ∈ carrier-vec n using v-def dimP by auto
      have inner-prod v (A *v v) = B$$(i, i) unfolding A v-def
        using spectral-decomposition-extract-diag[OF dimP dimB uP dB i] by auto
      moreover have inner-prod v (A *v v) = 0 using dimv zero by auto
      ultimately have B$$(i, i) = 0 by auto
    }
    note zB = this
     show B = 0m n n by (insert zB dB dimB, rule eq-matI, auto simp add:
diagonal-mat-def)
  qed
  then show A = 0m n n using A dimB dimP dimaP by auto
qed

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 ·m C ∧ B ∈ carrier-mat
n n ∧ C ∈ carrier-mat n n
proof −
  obtain B C where B: B = (1 / 2) ·m (A + adjoint A)
    and C: C = (−i / 2) ·m (A − adjoint A) by auto
  then have dimB: B ∈ carrier-mat n n and dimC: C ∈ carrier-mat n n using
dim by auto
  have hermitian B unfolding B hermitian-def using dim
    by (auto simp add: adjoint-eval)
  moreover have hermitian C unfolding C hermitian-def using dim
    apply (subst eq-matI)
       apply (auto simp add: adjoint-eval algebra-simps)
    done
  moreover have A = B + i ·m C using dim B C
    apply (subst eq-matI)
       apply (auto simp add: adjoint-eval algebra-simps)
```

done
ultimately show *?thesis* **using** *dimB dimC* **by** *auto*
qed

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*
unfolding *outer-prod-def* **using** *assms mat-of-cols-carrier mat-of-rows-carrier* **by** *auto*

lemma *mat-of-vec-mult-eq-scalar-prod*:
fixes *v w* :: *'a::conjugatable-field vec*
assumes *v* ∈ *carrier-vec n* **and** *w* ∈ *carrier-vec n*
shows *mat 1* (*dim-vec v*) (λ(*i*, *j*). (*conjugate v*) $ *j*) ∗ *mat* (*dim-vec w*) *1* (λ(*i*, *j*). *w* $ *i*)
= *mat 1 1* (λ*k*. *inner-prod v w*)
apply (*rule eq-matI*) **using** *assms* **apply** (*simp add*: *scalar-prod-def*) **apply** (*rule sum.cong*) **by** *auto*

lemma *one-dim-mat-mult-is-scale*:
fixes *A B* :: (*'a::conjugatable-field mat*)
assumes *B* ∈ *carrier-mat 1 n*
shows (*mat 1 1* (λ*k*. *a*)) ∗ *B* = *a* $\cdot_m$ *B*
apply (*rule eq-matI*) **using** *assms* **by** (*auto simp add*: *scalar-prod-def*)

lemma *outer-prod-mult-outer-prod*:
fixes *a b c d* :: *'a::conjugatable-field vec*
assumes *a*: *a* ∈ *carrier-vec d1* **and** *b*: *b* ∈ *carrier-vec d2*
and *c*: *c* ∈ *carrier-vec d2* **and** *d*: *d* ∈ *carrier-vec d3*
shows *outer-prod a b* ∗ *outer-prod c d* = *inner-prod b c* $\cdot_m$ *outer-prod a d*
proof −
let *?ma* = *mat* (*dim-vec a*) *1* (λ(*i*, *j*). *a* $ *i*)
let *?mb* = *mat 1* (*dim-vec b*) (λ(*i*, *j*). (*conjugate b*) $ *j*)
let *?mc* = *mat* (*dim-vec c*) *1* (λ(*i*, *j*). *c* $ *i*)
let *?md* = *mat 1* (*dim-vec d*) (λ(*i*, *j*). (*conjugate d*) $ *j*)
have (*?ma* ∗ *?mb*) ∗ (*?mc* ∗ *?md*) = *?ma* ∗ (*?mb* ∗ (*?mc* ∗ *?md*))
apply (*subst assoc-mult-mat*[*of ?ma d1 1 ?mb d2 ?mc* ∗ *?md d3*])
using *assms* **by** *auto*
also have … = *?ma* ∗ ((*?mb* ∗ *?mc*) ∗ *?md*)
apply (*subst assoc-mult-mat*[*symmetric*, *of ?mb 1 d2 ?mc 1 ?md d3*])
using *assms* **by** *auto*
also have … = *?ma* ∗ ((*mat 1 1* (λ*k*. *inner-prod b c*)) ∗ *?md*)

```
    apply (subst mat-of-vec-mult-eq-scalar-prod[of b d2 c]) using assms by auto
  also have ... = ?ma * (inner-prod b c ·m ?md)
    apply (subst one-dim-mat-mult-is-scale) using assms by auto
  also have ... = (inner-prod b c) ·m (?ma * ?md) using assms by auto
  finally show ?thesis unfolding outer-prod-def by auto
qed

lemma index-outer-prod:
  fixes v w :: 'a::conjugatable-field vec
  assumes v: v ∈ carrier-vec n and w: w ∈ carrier-vec m
    and ij: i < n j < m
  shows (outer-prod v w)$$(i, j) = v $ i * conjugate (w $ j)
  unfolding outer-prod-def using assms by (simp add: scalar-prod-def)

lemma mat-of-vec-mult-vec:
  fixes a b c :: 'a::conjugatable-field vec
  assumes a: a ∈ carrier-vec d and b: b ∈ carrier-vec d
  shows mat 1 d (λ(i, j). (conjugate a) $ j) *v b = vec 1 (λk. inner-prod a b)
  apply (rule eq-vecI)
   apply (simp add: scalar-prod-def carrier-vecD[OF a] carrier-vecD[OF b])
  apply (rule sum.cong) by auto

lemma mat-of-vec-mult-one-dim-vec:
  fixes a b :: 'a::conjugatable-field vec
  assumes a: a ∈ carrier-vec d
  shows mat d 1 (λ(i, j). a $ i) *v vec 1 (λk. c) = c ·v a
  apply (rule eq-vecI)
  by (auto simp add: scalar-prod-def carrier-vecD[OF a])

lemma outer-prod-mult-vec:
  fixes a b c :: 'a::conjugatable-field vec
  assumes a: a ∈ carrier-vec d1 and b: b ∈ carrier-vec d2
    and c: c ∈ carrier-vec d2
  shows outer-prod a b *v c = inner-prod b c ·v a
proof −
  have outer-prod a b *v c
    = mat d1 1 (λ(i, j). a $ i)
    * mat 1 d2 (λ(i, j). (conjugate b) $ j)
    *v c unfolding outer-prod-def using assms by auto
  also have ... = mat d1 1 (λ(i, j). a $ i)
    *v (mat 1 d2 (λ(i, j). (conjugate b) $ j)
    *v c) apply (subst assoc-mult-mat-vec) using assms by auto
  also have ... = mat d1 1 (λ(i, j). a $ i)
    *v vec 1 (λk. inner-prod b c) using mat-of-vec-mult-vec[of b] assms by auto
   also have ... = inner-prod b c ·v a using mat-of-vec-mult-one-dim-vec assms
by auto
  finally show ?thesis by auto
qed
```

```
lemma trace-outer-prod-right:
  fixes A :: 'a::conjugatable-field mat and v w :: 'a vec
  assumes A: A ∈ carrier-mat n n
    and v: v ∈ carrier-vec n and w: w ∈ carrier-vec n
  shows trace (A ∗ outer-prod v w) = inner-prod w (A ∗v v) (is ?lhs = ?rhs)
proof −
  define B where B = outer-prod v w
  then have B: B ∈ carrier-mat n n using assms by auto
  have trace(A ∗ B) = (∑ i = 0..<n. ∑ j = 0..<n. A $$ (i,j) ∗ B $$ (j,i))
    unfolding trace-def using A B by (simp add: scalar-prod-def)
  also have ... = (∑ i = 0..<n. ∑ j = 0..<n. A $$ (i,j) ∗ v $ j ∗ conjugate (w
$ i))
    unfolding B-def
    apply (rule sum.cong, simp, rule sum.cong, simp)
    by (insert v w, auto simp add: index-outer-prod)
  finally have ?lhs = (∑ i = 0..<n. ∑ j = 0..<n. A $$ (i,j) ∗ v $ j ∗ conjugate
(w $ i)) using B-def by auto
  moreover have ?rhs = (∑ i = 0..<n. ∑ j = 0..<n. A $$ (i,j) ∗ v $ j ∗ conjugate
(w $ i)) using A v w
    by (simp add: scalar-prod-def sum-distrib-right)
  ultimately show ?thesis by auto
qed

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 −
  have (1m n) ∗ (outer-prod v w) = outer-prod v w apply (subst left-mult-one-mat)
using outer-prod-dim assms by auto
  moreover have 1m n ∗v v = v using assms by auto
  ultimately show ?thesis using trace-outer-prod-right[of 1m n n v w] assms by
auto
qed

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 −
  define P where P = outer-prod b c
  then have dimP: P ∈ carrier-mat n m using assms by auto
  have inner-prod a (P ∗v d) = (∑ i=0..<n. (∑ j=0..<m. conjugate (a$i) ∗
P$$(i, j) ∗ d$j)) using assms dimP
    apply (simp add: scalar-prod-def sum-distrib-right)
    apply (rule sum.cong, auto)
    apply (rule sum.cong, auto)
```

done
also have ... = (∑ *i=0..<n.* (∑ *j=0..<m. conjugate* (*a$i*) ∗ *b$i* ∗ *conjugate*(*c$j*) ∗ *d$j*))
using *P-def b c* **by**(*simp add*: *index-outer-prod algebra-simps*)
finally have *eq*: *?lhs* = (∑ *i=0..<n.* (∑ *j=0..<m. conjugate* (*a$i*) ∗ *b$i* ∗ *conjugate*(*c$j*) ∗ *d$j*)) **using** *P-def* **by** *auto*

have *?rhs* = (∑ *i=0..<n. conjugate* (*a$i*) ∗ *b$i*) ∗ (∑ *j=0..<m. conjugate*(*c$j*) ∗ *d$j*) **using** *assms*
by (*auto simp add*: *scalar-prod-def algebra-simps*)
also have ... = (∑ *i=0..<n.* (∑ *j=0..<m. conjugate* (*a$i*) ∗ *b$i* ∗ *conjugate*(*c$j*) ∗ *d$j*))
using *assms* **by** (*simp add*: *sum-product algebra-simps*)
finally show *?lhs* = *?rhs* **using** *eq* **by** *auto*
qed

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 (*rule*)
assume *positive A*
then show *A* ∈ *carrier-mat* (*dim-col A*) (*dim-col A*) ∧
(∀ *v. dim-vec v* = *dim-col A* ∧ *vec-norm v* = *1* ⟶ *0* ≤ *inner-prod v* (*A* $*_v$ *v*))
unfolding *positive-def* **by** *auto*
next
define *n* **where** *n* = *dim-col A*
assume *A* ∈ *carrier-mat* (*dim-col A*) (*dim-col A*) ∧ (∀ *v. dim-vec v* = *dim-col A* ∧ *vec-norm v* = *1* ⟶ *0* ≤ *inner-prod v* (*A* $*_v$ *v*))
then have *A*: *A* ∈ *carrier-mat* (*dim-col A*) (*dim-col A*) **and** *geq0*: ∀ *v. dim-vec v* = *dim-col A* ∧ *vec-norm v* = *1* ⟶ *0* ≤ *inner-prod v* (*A* $*_v$ *v*) **by** *auto*
then have *dimA*: *A* ∈ *carrier-mat n n* **using** *n-def*[*symmetric*] **by** *auto*
{
fix *v* **assume** *dimv*: (*v*::*complex vec*) ∈ *carrier-vec n*
have *0* ≤ *inner-prod v* (*A* $*_v$ *v*)
proof (*cases v* = 0_v *n*)
case *True*
then show *0* ≤ *inner-prod v* (*A* $*_v$ *v*) **using** *dimA* **by** *auto*
next
case *False*
then have *1*: *vec-norm v* > *0* **using** *vec-norm-ge-0 dimv* **by** *auto*

then have *cnv*: *cnj* (*vec-norm v*) = *vec-norm v*
using *Reals-cnj-iff complex-is-Real-iff less-complex-def* **by** *auto*
define *w* **where** *w* = *vec-normalize v*
then have *dimw*: *w* ∈ *carrier-vec n* **using** *dimv* **by** *auto*
have *nvw*: *v* = *vec-norm v* $\cdot_v$ *w* **using** *w-def vec-eq-norm-smult-normalized* **by** *auto*
have *vec-norm w* = *1* **using** *normalized-vec-norm*[*OF dimv False*] *vec-norm-def w-def* **by** *auto*
then have *2*: *0* ≤ *inner-prod w* (*A* $*_v$ *w*) **using** *geq0 dimw dimA* **by** *auto*
have *inner-prod v* (*A* $*_v$ *v*) = *vec-norm v* ∗ *vec-norm v* ∗ *inner-prod w* (*A* $*_v$ *w*) **using** *dimA dimv dimw*
apply (*subst* (*1 2*) *nvw*)
apply (*subst mult-mat-vec*, *simp*, *simp*)
apply (*subst scalar-prod-smult-left*[*of* (*A* $*_v$ *w*) *conjugate* (*vec-norm v* $\cdot_v$ *w*) *vec-norm v*], *simp*)
apply (*simp add*: *conjugate-smult-vec cnv*)
done
also have ... ≥ *0* **using** *1 2* **by** *auto*
finally show *0* ≤ *inner-prod v* (*A* $*_v$ *v*) **by** *auto*
qed
}
then have *geq*: ∀ *v*. *dim-vec v* = *dim-col A* ⟶ *0* ≤ *inner-prod v* (*A* $*_v$ *v*) **using** *dimA* **by** *auto*
show *positive A* **unfolding** *positive-def*
by (*rule*, *simp add*: *A*, *rule geq*)
qed

lemma *positive-is-hermitian*:
fixes *A* :: *complex mat*
assumes *pA*: *positive A*
shows *hermitian A*
proof −
define *n* **where** *n* = *dim-col A*
then have *dimA*: *A* ∈ *carrier-mat n n* **using** *positive-def pA* **by** *auto*
obtain *B C* **where** *B*: *hermitian B* **and** *C*: *hermitian C* **and** *A*: *A* = *B* + i $\cdot_m$ *C*
and *dimB*: *B* ∈ *carrier-mat n n* **and** *dimC*: *C* ∈ *carrier-mat n n* **and** *dimiC*: i $\cdot_m$ *C* ∈ *carrier-mat n n*
using *complex-mat-decomposition-to-hermitian*[*OF dimA*] **by** *auto*
{
fix *v* :: *complex vec* **assume** *dimv*: *v* ∈ *carrier-vec n*
have *dimvA*: *dim-vec v* = *dim-col A* **using** *dimv dimA* **by** *auto*
have *inner-prod v* (*A* $*_v$ *v*) = *inner-prod v* (*B* $*_v$ *v*) + *inner-prod v* ((i $\cdot_m$ *C*) $*_v$ *v*)
unfolding *A* **using** *dimB dimiC dimv* **by** (*simp add*: *add-mult-distrib-mat-vec inner-prod-distrib-right*)
moreover have *inner-prod v* ((i $\cdot_m$ *C*) $*_v$ *v*) = i ∗ *inner-prod v* (*C* $*_v$ *v*) **using** *dimv dimC*
apply (*simp add*: *scalar-prod-def sum-distrib-left cong*: *sum.cong*)

apply (*rule sum.cong*, *auto*)
done
ultimately have *ABC*: *inner-prod v* ($A *_v v$) = *inner-prod v* ($B *_v v$) + i * *inner-prod v* ($C *_v v$) **by** *auto*
moreover have *inner-prod v* ($B *_v v$) ∈ *Reals* **using** *B dimB dimv hermitian-inner-prod-real* **by** *auto*
moreover have *inner-prod v* ($C *_v v$) ∈ *Reals* **using** *C dimC dimv hermitian-inner-prod-real* **by** *auto*
moreover have *inner-prod v* ($A *_v v$) ∈ *Reals* **using** *pA* **unfolding** *positive-def*

apply (*rule*)
apply (*fold n-def*)
apply (*simp add*: *complex-is-Real-iff*[*of inner-prod v* ($A *_v v$)])
apply (*auto simp add*: *dimvA less-complex-def less-eq-complex-def*)
done
ultimately have *inner-prod v* ($C *_v v$) = *0* **using** *of-real-Re* **by** *fastforce*
}
then have $C = 0_m$ *n n* **using** *hermitian-inner-prod-zero dimC C* **by** *auto*
then have *A* = *B* **using** *A dimC dimB* **by** *auto*
then show *hermitian A* **using** *B* **by** *auto*
qed

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*$$(*i*, *i*) ≥ *0*
proof −
have *hA*: *hermitian A* **using** *positive-is-hermitian pA* **by** *auto*
have *similar-mat-wit A B P* (*adjoint P*) ∧ *diagonal-mat B* ∧ *diag-mat B* = *es* ∧ *unitary P* ∧ (∀ *i* < *n*. *B*$$(*i*, *i*) ∈ *Reals*)
using *hermitian-eigenvalue-real dimA hA B c* **by** *auto*
then have *A*: *A* = *P* * *B* * (*adjoint P*) **and** *dB*: *diagonal-mat B*
and *Bii*: ⋀*i*. *i* < *n* ⟹ *B*$$(*i*, *i*) ∈ *Reals*
and *dimB*: *B* ∈ *carrier-mat n n* **and** *dimP*: *P* ∈ *carrier-mat n n* **and** *dimaP*: *adjoint P* ∈ *carrier-mat n n*
and *uP*: *unitary P*
unfolding *similar-mat-wit-def Let-def unitary-def* **using** *dimA* **by** *auto*
{
fix *i* **assume** *i*: *i* < *n*
define *v* **where** *v* = *col P i*
then have *dimv*: *v* ∈ *carrier-vec n* **using** *v-def dimP* **by** *auto*
have *inner-prod v* ($A *_v v$) = *B*$$(*i*, *i*) **unfolding** *A v-def*
using *spectral-decomposition-extract-diag*[*OF dimP dimB uP dB i*] **by** *auto*
moreover have *inner-prod v* ($A *_v v$) ≥ *0* **using** *dimv pA dimA positive-def* **by** *auto*
ultimately show *B*$$(*i*, *i*) ≥ *0* **by** *auto*
}

qed

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* (λ(*i*,*j*). (*if i* = *j then* (*B*$$(*i*, *i*)) ∗ (*D*$$(*i*, *i*)) *else 0*))
proof(*rule eq-matI*, *auto*)
  have *Bij*: ⋀*x y*. *x* < *n* ⟹ *y* < *n* ⟹ *x* ≠ *y* ⟹ *B*$$(*x*, *y*) = *0* **using** *dB diagonal-mat-def dimB* **by** *auto*
  have *Dij*: ⋀*x y*. *x* < *n* ⟹ *y* < *n* ⟹ *x* ≠ *y* ⟹ *D*$$(*x*, *y*) = *0* **using** *dD diagonal-mat-def dimD* **by** *auto*
{
  fix *i j* **assume** *ij*: *i* < *n j* < *n*
  have (*B* ∗ *D*) $$ (*i*, *j*) = (∑ *k*=*0*..<*n*. (*B* $$ (*i*, *k*)) ∗ (*D* $$ (*k*, *j*))) **using** *dimB dimD*
    by (*auto simp add*: *scalar-prod-def ij*)
  also have ... = *B*$$(*i*, *i*) ∗ *D*$$(*i*, *j*)
    apply (*simp add*: *sum.remove*[*of -i*] *ij*)
    apply (*simp add*: *Bij Dij ij*)
    done
  finally have (*B* ∗ *D*) $$ (*i*, *j*) = *B*$$(*i*, *i*) ∗ *D*$$(*i*, *j*)**.**
}
  note *BDij* = *this*
  from *BDij* **show** ⋀*j*. *j* < *n* ⟹ (*B* ∗ *D*) $$ (*j*, *j*) = *B* $$ (*j*, *j*) ∗ *D* $$ (*j*, *j*) **by** *auto*
  from *BDij* **show** ⋀*i j*. *i* < *n* ⟹ *j* < *n* ⟹ *i* ≠ *j* ⟹ (*B* ∗ *D*) $$ (*i*, *j*) = *0* **using** *Bij Dij* **by** *auto*
  from *assms* **show** *dim-row B* = *n dim-col D* = *n* **by** *auto*
qed

lemma *positive-only-if-decomp*:
  assumes *dimA*: *A* ∈ *carrier-mat n n* **and** *pA*: *positive A*
  shows ∃ *M* ∈ *carrier-mat n n*. *M* ∗ *adjoint M* = *A*
proof −
  from *pA* **have** *hA*: *hermitian A* **using** *positive-is-hermitian* **by** *auto*
  obtain *es* **where** *es*: *char-poly A* = (∏ (*e* :: *complex*) ← *es*. [:− *e*, *1*:])
    using *complex-mat-char-poly-factorizable dimA* **by** *auto*
  obtain *B P Q* **where** *schur*: *unitary-schur-decomposition A es* = (*B*,*P*,*Q*)
    by (*cases unitary-schur-decomposition A es*, *auto*)
  then have *similar-mat-wit A B P* (*adjoint P*) ∧ *diagonal-mat B* ∧ *diag-mat B* = *es*
    ∧ *unitary P* ∧ (∀ *i* < *n*. *B*$$(*i*, *i*) ∈ *Reals*)
    using *hermitian-eigenvalue-real dimA es hA* **by** *auto*
  then have *A*: *A* = *P* ∗ *B* ∗ (*adjoint P*) **and** *dB*: *diagonal-mat B*
    and *Bii*: ⋀*i*. *i* < *n* ⟹ *B*$$(*i*, *i*) ∈ *Reals*
    and *dimB*: *B* ∈ *carrier-mat n n* **and** *dimP*: *P* ∈ *carrier-mat n n* **and** *dimaP*: *adjoint P* ∈ *carrier-mat n n*
    unfolding *similar-mat-wit-def Let-def* **using** *dimA* **by** *auto*

have *Bii*: $\bigwedge i.\ i < n \Longrightarrow B\$\$(i, i) \geq 0$ **using** *pA dimA es schur positive-eigenvalue-positive*
by *auto*
define *D* **where** $D = mat\ n\ n\ (\lambda(i, j).\ (if\ (i = j)\ then\ csqrt\ (B\$\$(i, i))\ else\ 0))$
then have *dimD*: $D \in carrier\text{-}mat\ n\ n$ **and** *dimaD*: $adjoint\ D \in carrier\text{-}mat\ n\ n$ **using** *dimB* **by** *auto*
have *dD*: *diagonal-mat D* **using** *dB D-def* **unfolding** *diagonal-mat-def* **by** *auto*
then have *daD*: *diagonal-mat* (*adjoint D*) **by** (*simp add*: *adjoint-eval diagonal-mat-def*)
have *Dii*: $\bigwedge i.\ i < n \Longrightarrow D\$\$(i, i) = csqrt\ (B\$\$(i, i))$ **using** *dimD D-def* **by** *auto*
{
fix *i* **assume** *i*: $i < n$
define *c* **where** $c = csqrt\ (B\$\$(i, i))$
have *c*: $c \geq 0$ **using** *Bii i c-def* **by** (*auto simp*: *less-complex-def less-eq-complex-def*)
then have *conjugate c* = *c*
using *Reals-cnj-iff complex-is-Real-iff* **unfolding** *less-complex-def less-eq-complex-def*
by *auto*
then have $c * cnj\ c = B\$\$(i, i)$ **using** *c-def c* **unfolding** *conjugate-complex-def*
by (*metis power2-csqrt power2-eq-square*)
}
note *cBii = this*
have $D * adjoint\ D = mat\ n\ n\ (\lambda(i,j).\ (if\ (i = j)\ then\ B\$\$(i, i)\ else\ 0))$
apply (*simp add*: *diag-mat-mult-diag-mat*[*OF dimD dimaD dD daD*])
apply (*rule eq-matI*, *auto simp add*: *D-def adjoint-eval cBii*)
done
also have ... = *B* **using** *dimB dB*[*unfolded diagonal-mat-def*] **by** *auto*
finally have *DaDB*: $D * adjoint\ D = B$.
define *M* **where** $M = P * D$
then have *dimM*: $M \in carrier\text{-}mat\ n\ n$ **using** *dimP dimD* **by** *auto*
have $M * adjoint\ M = (P * D) * (adjoint\ D * adjoint\ P)$ **using** *M-def adjoint-mult*[*OF dimP dimD*] **by** *auto*
also have ... $= P * (D * adjoint\ D) * (adjoint\ P)$ **using** *dimP dimD* **by** (*mat-assoc n*)
also have ... $= P * B * (adjoint\ P)$ **using** *DaDB* **by** *auto*
finally have $M * adjoint\ M = A$ **using** *A* **by** *auto*
with *dimM* **show** $\exists M \in carrier\text{-}mat\ n\ n.\ M * adjoint\ M = A$ **by** *auto*
qed

lemma *positive-if-decomp*:
assumes *dimA*: $A \in carrier\text{-}mat\ n\ n$ **and** $\exists M.\ M * adjoint\ M = A$
shows *positive A*
proof −
from *assms* **obtain** *M* **where** *M*: $M * adjoint\ M = A$ **by** *auto*
define *m* **where** $m = dim\text{-}col\ M$
have *dimM*: $M \in carrier\text{-}mat\ n\ m$ **using** *M dimA m-def* **by** *auto*
{
fix *v* **assume** *dimv*: $(v::complex\ vec) \in carrier\text{-}vec\ n$
have *dimaM*: $adjoint\ M \in carrier\text{-}mat\ m\ n$ **using** *dimM* **by** *auto*
have *dimaMv*: $(adjoint\ M) *_v v \in carrier\text{-}vec\ m$ **using** *dimaM dimv* **by** *auto*

have *inner-prod v* $(A *_v v)$ = *inner-prod v* $(M *$ *adjoint M* $*_v v)$ **using** *M* **by** *auto*
also have ... = *inner-prod v* $(M *_v$ (*adjoint M* $*_v v))$ **using** *assoc-mult-mat-vec dimM dimaM dimv* **by** *auto*
also have ... = *inner-prod* (*adjoint M* $*_v v$) (*adjoint M* $*_v v$) **using** *adjoint-def-alter*[*OF dimv dimaMv dimM*] **by** *auto*
also have ... $\geq$ *0* **using** *self-cscalar-prod-geq-0* **by** *auto*
finally have *inner-prod v* $(A *_v v) \geq$ *0***.**
}
note *geq0* = *this*
from *dimA geq0* **show** *positive A* **using** *positive-def* **by** *auto*
qed

lemma *positive-iff-decomp*:
assumes *dimA*: *A* ∈ *carrier-mat n n*
shows *positive A* ⟷ (∃ *M*∈*carrier-mat n n*. *M* ∗ *adjoint M* = *A*)
proof
assume *pA*: *positive A*
then show ∃ *M*∈*carrier-mat n n*. *M* ∗ *adjoint M* = *A* **using** *positive-only-if-decomp assms* **by** *auto*
next
assume ∃ *M*∈*carrier-mat n n*. *M* ∗ *adjoint M* = *A*
then obtain *M* **where** *M*: *M* ∗ *adjoint M* = *A* **by** *auto*
then show *positive A* **using** *M positive-if-decomp assms* **by** *auto*
qed

lemma *positive-dim-eq*:
assumes *positive A*
shows *dim-row A* = *dim-col A*
using *carrier-matD*(*1*)[*of A dim-col A dim-col A*] *assms*[*unfolded positive-def*] **by** *simp*

lemma *positive-zero*:
positive (0_m *n n*)
by (*simp add*: *positive-def zero-mat-def mult-mat-vec-def scalar-prod-def*)

lemma *positive-one*:
positive (1_m *n*)
proof (*rule positive-if-decomp*)
show 1_m *n* ∈ *carrier-mat n n* **by** *auto*
have *adjoint* (1_m *n*) = 1_m *n* **using** *hermitian-one hermitian-def* **by** *auto*
then have 1_m *n* ∗ *adjoint* (1_m *n*) = 1_m *n* **by** *auto*
then show ∃ *M*. *M* ∗ *adjoint M* = 1_m *n* **by** *fastforce*
qed

lemma *positive-antisym*:
assumes *pA*: *positive A* **and** *pnA*: *positive* (−*A*)
shows *A* = 0_m (*dim-col A*) (*dim-col A*)
proof −

```
  define n where n = dim-col A
  from pA have dimA: A ∈ carrier-mat n n and dimnA: −A ∈ carrier-mat n n
    using positive-def n-def by auto
  from pA have hA: hermitian A using positive-is-hermitian by auto
  obtain es where es: char-poly A = (∏ (e :: complex) ← es. [:− e, 1:])
    using complex-mat-char-poly-factorizable dimA by auto
  obtain B P Q where schur: unitary-schur-decomposition A es = (B,P,Q)
    by (cases unitary-schur-decomposition A es, auto)
  then have similar-mat-wit A B P (adjoint P) ∧ diagonal-mat B ∧ unitary P
    using hermitian-eigenvalue-real dimA es hA by auto
 then have A: A = P ∗ B ∗ (adjoint P) and dB: diagonal-mat B and uP: unitary
P
    and dimB: B ∈ carrier-mat n n and dimnB: −B ∈ carrier-mat n n
    and dimP: P ∈ carrier-mat n n and dimaP: adjoint P ∈ carrier-mat n n
    unfolding similar-mat-wit-def Let-def using dimA by auto
 from es schur have geq0: ⋀i. i < n ⟹ B$$(i, i) ≥ 0 using positive-eigenvalue-positive
dimA pA by auto
  from A have nA: −A = P ∗ (−B) ∗ (adjoint P) using mult-smult-assoc-mat
dimB dimP dimaP by auto
  from dB have dnB: diagonal-mat (−B) by (simp add: diagonal-mat-def)
  {
    fix i assume i: i < n
    define v where v = col P i
    then have dimv: v ∈ carrier-vec n using v-def dimP by auto
    have inner-prod v ((−A) ∗v v) = (−B)$$(i, i) unfolding nA v-def
     using spectral-decomposition-extract-diag[OF dimP dimnB uP dnB i] by auto
    moreover have inner-prod v ((−A) ∗v v) ≥ 0 using dimv pnA dimnA posi-
tive-def by auto
    ultimately have B$$(i, i) ≤ 0 using dimB i by auto
    moreover have B$$(i, i) ≥ 0 using i geq0 by auto
    ultimately have B$$(i, i) = 0
      by (metis antisym)
  }
  then have B = 0m n n using dimB dB[unfolded diagonal-mat-def]
    by (subst eq-matI, auto)
  then show A = 0m n n using A dimB dimP dimaP by auto
qed

lemma positive-add:
  assumes pA: positive A and pB: positive B
    and dimA: A ∈ carrier-mat n n and dimB: B ∈ carrier-mat n n
  shows positive (A + B)
  unfolding positive-def
proof
  have dimApB: A + B ∈ carrier-mat n n using dimA dimB by auto
  then show A + B ∈ carrier-mat (dim-col (A + B)) (dim-col (A + B)) using
carrier-matD[of A+B] by auto
  {
    fix v assume dimv: (v::complex vec) ∈ carrier-vec n
```

have *1*: *inner-prod v* ($A *_v v$) ≥ *0* **using** *dimv pA*[*unfolded positive-def*] *dimA* **by** *auto*
have *2*: *inner-prod v* ($B *_v v$) ≥ *0* **using** *dimv pB*[*unfolded positive-def*] *dimB* **by** *auto*
have *inner-prod v* (($A + B$) $*_v v$) = *inner-prod v* ($A *_v v$) + *inner-prod v* ($B *_v v$)
using *dimA dimB dimv* **by** (*simp add*: *add-mult-distrib-mat-vec inner-prod-distrib-right*)

also have ... ≥ *0* **using** *1 2* **by** *auto*
finally have *inner-prod v* (($A + B$) $*_v v$) ≥ *0***.**
}
note *geq0* = *this*
then have ⋀*v*. *dim-vec v* = *n* ⟹ *0* ≤ *inner-prod v* (($A + B$) $*_v v$) **by** *auto*
then show ∀ *v*. *dim-vec v* = *dim-col* ($A + B$) ⟶ *0* ≤ *inner-prod v* (($A + B$) $*_v v$) **using** *dimApB* **by** *auto*
qed

lemma *positive-trace*:
assumes *A* ∈ *carrier-mat n n* **and** *positive A*
shows *trace A* ≥ *0*
using *assms positive-iff-decomp trace-adjoint-positive* **by** *auto*

lemma *positive-close-under-left-right-mult-adjoint*:
fixes *M A* :: *complex mat*
assumes *dM*: *M* ∈ *carrier-mat n n* **and** *dA*: *A* ∈ *carrier-mat n n*
and *pA*: *positive A*
shows *positive* ($M * A *$ *adjoint M*)
unfolding *positive-def*
proof (*rule*, *simp add*: *mult-carrier-mat*[*OF mult-carrier-mat*[*OF dM dA*] *adjoint-dim*[*OF dM*]] *carrier-matD*[*OF dM*], *rule*, *rule*)
have *daM*: *adjoint M* ∈ *carrier-mat n n* **using** *dM* **by** *auto*
fix *v*::*complex vec* **assume** *dim-vec v* = *dim-col* ($M * A *$ *adjoint M*)
then have *dv*: *v* ∈ *carrier-vec n* **using** *assms* **by** *auto*
then have *adjoint M* $*_v$ *v* ∈ *carrier-vec n* **using** *daM* **by** *auto*
have *assoc*: $M * A *$ *adjoint M* $*_v v = M *_v$ ($A *_v$ (*adjoint M* $*_v v$))
using *dA dM daM dv* **by** (*auto simp add*: *assoc-mult-mat-vec*[*of - n n - n*])
have *inner-prod v* ($M * A *$ *adjoint M* $*_v v$) = *inner-prod* (*adjoint M* $*_v v$) ($A *_v$ (*adjoint M* $*_v v$))
apply (*subst assoc*)
apply (*subst adjoint-def-alter*[**where** *?A* = *M*])
by (*auto simp add*: *dv dA daM dM carrier-matD*[*OF dM*] *mult-mat-vec-carrier*[*of - n n*])
also have ... ≥ *0* **using** *dA dv daM pA positive-def* **by** *auto*
finally show *inner-prod v* ($M * A *$ *adjoint M* $*_v v$) ≥ *0* **by** *auto*
qed

lemma *positive-same-outer-prod*:
fixes *v w* :: *complex vec*
assumes *v*: *v* ∈ *carrier-vec n*

shows *positive* (*outer-prod v v*)
proof −
have *d1*: *adjoint* (*mat* (*dim-vec v*) *1* (λ(*i*, *j*). *v* $ *i*)) ∈ *carrier-mat 1 n* **using** *assms* **by** *auto*
have *d2*: *mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*) ∈ *carrier-mat 1 n* **using** *assms* **by** *auto*
have *dv*: *dim-vec v* = *n* **using** *assms* **by** *auto*
have *mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*) = *adjoint* (*mat* (*dim-vec v*) *1* (λ(*i*, *j*). *v* $ *i*)) (**is** *?r* = *adjoint ?l*)
apply (*rule eq-matI*)
subgoal for *i j* **by** (*simp add*: *dv adjoint-eval*)
using *d1 d2* **by** *auto*
then have *outer-prod v v* = *?l* ∗ *adjoint ?l* **unfolding** *outer-prod-def* **by** *auto*
then have ∃ *M*. *M* ∗ *adjoint M* = *outer-prod v v* **by** *auto*
then show *positive* (*outer-prod v v*) **using** *positive-if-decomp*[*OF outer-prod-dim*[*OF v v*]] **by** *auto*
qed

lemma *smult-smult-mat*:
fixes *k* :: *complex* **and** *l* :: *complex*
assumes *A* ∈ *carrier-mat nr n*
shows *k* $\cdot_m$ (*l* $\cdot_m$ *A*) = (*k* ∗ *l*) $\cdot_m$ *A* **by** *auto*

lemma *positive-smult*:
assumes *A* ∈ *carrier-mat n n*
and *positive A*
and *c* ≥ *0*
shows *positive* (*c* $\cdot_m$ *A*)
proof −
have *sc*: *csqrt c* ≥ *0* **using** *assms*(*3*) **by** (*fastforce simp*: *less-eq-complex-def*)
obtain *M* **where** *dimM*: *M* ∈ *carrier-mat n n* **and** *A*: *M* ∗ *adjoint M* = *A* **using** *assms*(*1*−*2*) *positive-iff-decomp* **by** *auto*
have *c* $\cdot_m$ *A* = *c* $\cdot_m$ (*M* ∗ *adjoint M*) **using** *A* **by** *auto*
have *ccsq*: *conjugate* (*csqrt c*) = (*csqrt c*) **using** *sc Reals-cnj-iff*[*of csqrt c*] *complex-is-Real-iff*
by (*auto simp*: *less-eq-complex-def*)
have *MM*: (*M* ∗ *adjoint M*) ∈ *carrier-mat n n* **using** *A assms* **by** *fastforce*
have *leftd*: *c* $\cdot_m$ (*M* ∗ *adjoint M*) ∈ *carrier-mat n n* **using** *A assms* **by** *fastforce*
have *rightd*: (*csqrt c* $\cdot_m$ *M*) ∗ (*adjoint* (*csqrt c* $\cdot_m$ *M*))∈ *carrier-mat n n* **using** *A assms* **by** *fastforce*
have (*csqrt c* $\cdot_m$ *M*) ∗ (*adjoint* (*csqrt c* $\cdot_m$ *M*)) = (*csqrt c* $\cdot_m$ *M*) ∗ ((*conjugate* (*csqrt c*)) $\cdot_m$ *adjoint M*)
using *adjoint-scale assms*(*1*) **by** (*metis adjoint-scale*)
also have ... = (*csqrt c* $\cdot_m$ *M*) ∗ (*csqrt c* $\cdot_m$ *adjoint M*) **using** *sc ccsq* **by** *fastforce*
also have ... = *csqrt c* $\cdot_m$ (*M* ∗ (*csqrt c* $\cdot_m$ *adjoint M*))
using *mult-smult-assoc-mat index-smult-mat*(*2*,*3*) **by** *fastforce*
also have ... = *csqrt c* $\cdot_m$ ((*csqrt c*) $\cdot_m$ (*M* ∗ *adjoint M*))
using *mult-smult-distrib* **by** *fastforce*

```
  also have ... = c ·m (M ∗ adjoint M)
    using smult-smult-mat[of M ∗ adjoint M n n (csqrt c) (csqrt c)] MM sc
    by (metis power2-csqrt power2-eq-square )
  also have ... = c ·m A using A by auto
  finally have (csqrt c ·m M) ∗ (adjoint (csqrt c ·m M)) = c ·m A by auto
  moreover have c ·m A ∈ carrier-mat n n using assms(1) by auto
  moreover have csqrt c ·m M ∈ carrier-mat n n using dimM by auto
  ultimately show ?thesis using positive-iff-decomp by auto
qed
```

Version of previous theorem for real numbers

```
lemma positive-scale:
  fixes c :: real
  assumes A ∈ carrier-mat n n
    and positive A
    and c ≥ 0
  shows positive (c ·m A)
  apply (rule positive-smult) using assms by (auto simp: less-eq-complex-def)
```

1.13 Löwner partial order

```
definition lowner-le :: complex mat ⇒ complex mat ⇒ bool (infix ‹≤L› 50)
where
  A ≤L B ⟷ dim-row A = dim-row B ∧ dim-col A = dim-col B ∧ positive (B −
A)

lemma lowner-le-refl:
  assumes A ∈ carrier-mat n n
  shows A ≤L A
  unfolding lowner-le-def
  apply (simp add: minus-r-inv-mat[OF assms])
  by (rule positive-zero)

lemma lowner-le-antisym:
  assumes A: A ∈ carrier-mat n n and B: B ∈ carrier-mat n n
    and L1: A ≤L B and L2: B ≤L A
  shows A = B
proof −
  from L1 have P1: positive (B − A) by (simp add: lowner-le-def)
  from L2 have P2: positive (A − B) by (simp add: lowner-le-def)
  have A − B = − (B − A) using A B by auto
  then have P3: positive (− (B − A)) using P2 by auto
  have BA: B − A ∈ carrier-mat n n using A B by auto
  have B − A = 0m n n using BA by (subst positive-antisym[OF P1 P3], auto)
 then have B + (−A) + A = 0m n n + A using A B minus-add-uminus-mat[OF
B A] by auto
  then have B + (−A + A) = 0m n n + A using A B by auto
  then show A = B using A B BA uminus-l-inv-mat[OF A] by auto
qed
```

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 −
from *assms* **have** *positive* ($B-A$) **by** (*auto simp add*: *lowner-le-def*)
with *assms* **have** *geq*: *inner-prod v* (($B-A$) $*_v$ *v*) $\geq$ *0*
unfolding *positive-def* **by** *auto*
have *inner-prod v* (($B-A$) $*_v$ *v*) = *inner-prod v* ($B *_v v$) − *inner-prod v* ($A *_v$ *v*)
unfolding *minus-add-uminus-mat*[*OF B A*]
by (*subst add-mult-distrib-mat-vec*[*OF B - v*], *insert A B v*, *auto simp add*: *inner-prod-distrib-right*[*OF v*])
then show *?thesis* **using** *geq* **by** *auto*
qed

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 $A \leq_L C$
unfolding *lowner-le-def*
proof (*auto simp add*: *carrier-matD*[*OF A*] *carrier-matD*[*OF C*])
have *dim*: $C - A \in$ *carrier-mat n n* **using** *A C* **by** *auto*
{
fix *v* **assume** *v*: (*v*::*complex vec*) $\in$ *carrier-vec n*
from *L1* **have** *inner-prod v* ($A *_v v$) $\leq$ *inner-prod v* ($B *_v v$) **using** *lowner-le-inner-prod-le A B v* **by** *auto*
also from *L2* **have** ... $\leq$ *inner-prod v* ($C *_v v$) **using** *lowner-le-inner-prod-le B C v* **by** *auto*
finally have *inner-prod v* ($A *_v v$) $\leq$ *inner-prod v* ($C *_v v$)**.**
then have *inner-prod v* ($C *_v v$) − *inner-prod v* ($A *_v v$) $\geq$ *0* **by** *auto*
then have *inner-prod v* (($C - A$) $*_v$ *v*) $\geq$ *0* **using** *A C v*
apply (*subst minus-add-uminus-mat*[*OF C A*])
apply (*subst add-mult-distrib-mat-vec*[*OF C - v*], *simp*)
apply (*simp add*: *inner-prod-distrib-right*[*OF v*])
done
}
note *leq* = *this*
show *positive* ($C - A$) **unfolding** *positive-def*
apply (*rule*, *simp add*: *carrier-matD*[*OF A*] *dim*)
apply (*subst carrier-matD*[*OF dim*], *insert leq*, *auto*)
done
qed

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 −
have *positive* $(B - A)$ **using** *assms lowner-le-def* **by** *auto*
moreover have $B - A \in$ *carrier-mat n n* **using** *assms* **by** *auto*
ultimately have *trace* $(B - A) \geq 0$ **using** *positive-trace* **by** *auto*
moreover have *trace* $(B - A) =$ *trace* $B -$ *trace* A **using** *trace-minus-linear assms* **by** *auto*
ultimately have *trace* $B -$ *trace* $A \geq 0$ **by** *auto*
then show *trace* $A \leq$ *trace* B **by** *auto*
qed

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 −
have $B + D - (A + C) = B - A + (D - C)$ **using** *assms* **by** *auto*
then have *positive* $(B + D - (A + C))$ **using** *assms* **unfolding** *lowner-le-def* **using** *positive-add*
by (*metis minus-carrier-mat*)
then show $A + C \leq_L B + D$ **unfolding** *lowner-le-def* **using** *assms* **by** *fastforce*
qed

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 −
have *positive* $(B - A)$ **using** *assms lowner-le-def* **by** *fastforce*
moreover have $B - A = (-A) - (-B)$ **using** *assms* **by** *fastforce*
ultimately have *positive* $((-A) - (-B))$ **by** *auto*
then show *?thesis* **using** *lowner-le-def assms* **by** *fastforce*
qed

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 −
have *positive* $(D - C)$ **using** *assms lowner-le-def* **by** *auto*
then have $-D \leq_L -C$ **using** *lowner-le-swap assms* **by** *auto*
then have $A + (-D) \leq_L B + (-C)$ **using** *lowner-le-add*[*of A n B*] *assms* **by** *auto*
moreover have $A + (-D) = A - D$ **and** $B + (-C) = B - C$ **by** *auto*
ultimately show *?thesis* **by** *auto*

```
qed

lemma outer-prod-le-one:
  assumes v ∈ carrier-vec n
    and inner-prod v v ≤ 1
  shows outer-prod v v ≤L 1m n
proof −
  let ?o = outer-prod v v
  have do: ?o ∈ carrier-mat n n using assms by auto
  {
    fix u :: complex vec assume dim-vec u = n
    then have du: u ∈ carrier-vec n by auto
   have r: inner-prod u u ∈ Reals apply (simp add: scalar-prod-def carrier-vecD[OF
du])
      using complex-In-mult-cnj-zero complex-is-Real-iff by blast
    have geq0: inner-prod u u ≥ 0
      using self-cscalar-prod-geq-0 by auto

    have inner-prod u (?o *v u) = inner-prod u v * inner-prod v u
      apply (subst inner-prod-outer-prod)
      using du assms by auto
   also have ... ≤ inner-prod u u * inner-prod v v using Cauchy-Schwarz-complex-vec
du assms by auto
    also have ... ≤ inner-prod u u using assms(2) r geq0
      by (simp add: mult-right-le-one-le less-eq-complex-def)
    finally have le: inner-prod u (?o *v u) ≤ inner-prod u u.

    have inner-prod u ((1m n − ?o) *v u) = inner-prod u ((1m n *v u) − ?o *v
u)
      apply (subst minus-mult-distrib-mat-vec) using do du by auto
    also have ... = inner-prod u u − inner-prod u (?o *v u)
      apply (subst inner-prod-minus-distrib-right)
      using du do by auto
    also have ... ≥ 0 using le by auto
    finally have inner-prod u ((1m n − ?o) *v u) ≥ 0 by auto
  }
  then have positive (1m n − outer-prod v v)
    unfolding positive-def using do by auto
  then show ?thesis unfolding lowner-le-def using do by auto
qed

lemma zero-lowner-le-positiveD:
  fixes A :: complex mat
  assumes dA: A ∈ carrier-mat n n and le: 0m n n ≤L A
  shows positive A
  using assms unfolding lowner-le-def by (subgoal-tac A − 0m n n = A, auto)

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*
using *assms* **unfolding** *lowner-le-def* **by** (*subgoal-tac* $A - 0_m$ *n n* $= A$, *auto*)

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 –
have *dB*: $B \in$ *carrier-mat n n* **using** *le dA lowner-le-def* **by** *auto*
have 0_m *n n* $\leq_L$ *A* **using** *zero-lowner-le-positiveI dA pA* **by** *auto*
then have 0_m *n n* $\leq_L$ *B* **using** *dA dB le* **by** (*simp add*: *lowner-le-trans*[*of* - *n A B*])
then show *?thesis* **using** *dB zero-lowner-le-positiveD* **by** *auto*
qed

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* $M * B * M$
unfolding *lowner-le-def*
proof (*rule conjI*, *fastforce*)+
have *daM*: *adjoint* $M \in$ *carrier-mat n n* **using** *dM* **by** *auto*
have *dBmA*: $B - A \in$ *carrier-mat n n* **using** *dB dA* **by** *fastforce*
have *positive* $(B - A)$ **using** *le lowner-le-def* **by** *auto*
then have *p*: *positive* (*adjoint* $M * (B - A) * M$)
using *positive-close-under-left-right-mult-adjoint*[*OF daM dBmA*] *adjoint-adjoint*[*of M*] **by** *auto*
moreover have *e*: *adjoint* $M * (B - A) * M =$ *adjoint* $M * B * M -$ *adjoint* $M * A * M$ **using** *dM dB dA* **by** (*mat-assoc n*)
ultimately show *positive* (*adjoint* $M * B * M -$ *adjoint* $M * A * M$) **by** *auto*
qed

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 $c \cdot_m (B - A) = c \cdot_m B - c \cdot_m A$
using *assms* **by** (*auto simp add*: *minus-add-uminus-mat add-smult-distrib-left-mat*)

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 –
have *eqBA*: $c \cdot_m (B - A) = c \cdot_m B - c \cdot_m A$
using *assms* **by** (*auto simp add*: *smult-distrib-left-minus-mat*)

have *positive* $(B - A)$ **using** *assms*(*2*) **unfolding** *lowner-le-def* **by** *auto*
then have *positive* $(c \cdot_m (B - A))$ **using** *positive-smult*[*of* $B-A$ *n c*] *assms* **by** *fastforce*
moreover have $c \cdot_m A \in$ *carrier-mat n n* **using** *index-smult-mat*(*2*,*3*) *assms*(*3*) **by** *auto*
moreover have $c \cdot_m B \in$ *carrier-mat n n* **using** *index-smult-mat*(*2*,*3*) *assms*(*4*) **by** *auto*
ultimately show *?thesis* **unfolding** *lowner-le-def* **using** *eqBA* **by** *fastforce*
qed

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 $c \cdot_m A \leq_L c \cdot_m B$
apply (*rule lowner-le-smultc*) **using** *assms* **by** (*auto simp*: *less-eq-complex-def*)

lemma *minus-smult-vec-distrib*:
fixes *w* :: *'a*::*comm-ring-1 vec*
shows $(a - b) \cdot_v w = a \cdot_v w - b \cdot_v w$
apply (*rule eq-vecI*)
by (*auto simp add*: *scalar-prod-def algebra-simps*)

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 $a \cdot_m A *_v w = a \cdot_v (A *_v w)$
apply (*rule eq-vecI*)
apply (*simp add*: *scalar-prod-def carrier-matD*[*OF A*] *carrier-vecD*[*OF w*])
apply (*subst sum-distrib-left*) **apply** (*rule sum.cong*, *simp*)
by *auto*

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 $A *_v (a \cdot_v w) = a \cdot_v (A *_v w)$
apply (*rule eq-vecI*)
apply (*simp add*: *scalar-prod-def carrier-matD*[*OF A*] *carrier-vecD*[*OF w*])
apply (*subst sum-distrib-left*) **apply** (*rule sum.cong*, *simp*)
by *auto*

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* $B *_v v$))
unfolding *outer-prod-def*
proof −
have *eq1*: $A * (mat\ (dim\text{-}vec\ u)\ 1\ (\lambda(i, j).\ u \$ i)) = mat\ (dim\text{-}vec\ (A *_v u))\ 1\ (\lambda(i, j).\ (A *_v u) \$ i)$

apply (*rule eq-matI*)
by (*auto simp add*: *dA du scalar-prod-def*)
have *conj*: *conjugate a* ∗ *b* = *conjugate* ((*a*::*complex*) ∗ *conjugate b*) **for** *a b* **by** *auto*
have *eq2*: *mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*) ∗ *B* = *mat 1* (*dim-vec* (*adjoint B* $*_v$ *v*)) (λ(*i*, *y*). *conjugate* (*adjoint B* $*_v$ *v*) $ *y*)
apply (*rule eq-matI*)
apply (*auto simp add*: *carrier-matD*[*OF dB*] *carrier-vecD*[*OF dv*] *scalar-prod-def adjoint-def conjugate-vec-def sum-conjugate*)
apply (*rule sum.cong*)
by (*auto simp add*: *conj*)
have *A* ∗ (*mat* (*dim-vec u*) *1* (λ(*i*, *j*). *u* $ *i*) ∗ *mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*)) ∗ *B* =
(*A* ∗ (*mat* (*dim-vec u*) *1* (λ(*i*, *j*). *u* $ *i*))) ∗(*mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*)) ∗ *B*
using *dA du dv dB assoc-mult-mat*[*OF dA*, *of mat* (*dim-vec u*) *1* (λ(*i*, *j*). *u* $ *i*) *1 mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*)] **by** *fastforce*
also have ... = (*A* ∗ (*mat* (*dim-vec u*) *1* (λ(*i*, *j*). *u* $ *i*))) ∗((*mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*)) ∗ *B*)
using *dA du dv dB assoc-mult-mat*[*OF* - - *dB*, *of* (*A* ∗ (*mat* (*dim-vec u*) *1* (λ(*i*, *j*). *u* $ *i*))) *d1 1*] **by** *fastforce*
finally show *A* ∗ (*mat* (*dim-vec u*) *1* (λ(*i*, *j*). *u* $ *i*) ∗ *mat 1* (*dim-vec v*) (λ(*i*, *y*). *conjugate v* $ *y*)) ∗ *B* =
mat (*dim-vec* (*A* $*_v$ *u*)) *1* (λ(*i*, *j*). (*A* $*_v$ *u*) $ *i*) ∗ *mat 1* (*dim-vec* (*adjoint B* $*_v$ *v*)) (λ(*i*, *y*). *conjugate* (*adjoint B* $*_v$ *v*) $ *y*)
using *eq1 eq2* **by** *auto*
qed

1.14 Density operators

definition *density-operator* :: *complex mat* ⇒ *bool* **where**
density-operator A ⟷ *positive A* ∧ *trace A* = *1*

definition *partial-density-operator* :: *complex mat* ⇒ *bool* **where**
partial-density-operator A ⟷ *positive A* ∧ *trace A* ≤ *1*

lemma *pure-state-self-outer-prod-is-partial-density-operator*:
fixes *v* :: *complex vec*
assumes *dimv*: *v* ∈ *carrier-vec n* **and** *nv*: *vec-norm v* = *1*
shows *partial-density-operator* (*outer-prod v v*)
unfolding *partial-density-operator-def*
proof
have *dimov*: *outer-prod v v* ∈ *carrier-mat n n* **using** *dimv* **by** *auto*
show *positive* (*outer-prod v v*) **unfolding** *positive-def*
proof (*rule*, *simp add*: *carrier-matD*(*2*)[*OF dimov*] *dimov*, *rule allI*, *rule impI*)
fix *w* **assume** *dim-vec* (*w*::*complex vec*) = *dim-col* (*outer-prod v v*)
then have *dimw*: *w* ∈ *carrier-vec n* **using** *dimov carrier-vecI* **by** *auto*
then have *inner-prod w* ((*outer-prod v v*) $*_v$ *w*) = *inner-prod w v* ∗ *inner-prod v w*

using *inner-prod-outer-prod dimw dimv* **by** *auto*
also have ... = *inner-prod w v* ∗ *conjugate* (*inner-prod w v*) **using** *dimw dimv*
apply (*subst conjugate-scalar-prod*[*of v conjugate w*], *simp*)
apply (*subst conjugate-vec-sprod-comm*[*of conjugate v - conjugate w*], *auto*)
apply (*rule carrier-vec-conjugate*[*OF dimv*])
apply (*rule carrier-vec-conjugate*[*OF dimw*])
done
also have ... ≥ *0* **by** (*auto simp*: *less-eq-complex-def*)
finally show *inner-prod w* ((*outer-prod v v*) $*_v$ *w*) ≥ *0*.
qed
have *eq*: *trace* (*outer-prod v v*) = (∑ *i=0..<n. v$i* ∗ *conjugate*(*v$i*)) **unfolding** *trace-def*
apply (*subst carrier-matD*(*1*)[*OF dimov*])
apply (*simp add*: *index-outer-prod*[*OF dimv dimv*])
done
have *vec-norm v* = *csqrt* (∑ *i=0..<n. v$i* ∗ *conjugate*(*v$i*)) **unfolding** *vec-norm-def* **using** *dimv*
by (*simp add*: *scalar-prod-def*)
then have (∑ *i=0..<n. v$i* ∗ *conjugate*(*v$i*)) = *1* **using** *nv* **by** *auto*
with *eq* **show** *trace* (*outer-prod v v*) ≤ *1* **by** *auto*
qed

lemma *lowner-le-trace*:
assumes *A*: *A* ∈ *carrier-mat n n*
and *B*: *B* ∈ *carrier-mat n n*
shows *A* $\leq_L$ *B* ⟷ (∀ *ϱ*∈*carrier-mat n n. partial-density-operator ϱ* ⟶ *trace* (*A* ∗ *ϱ*) ≤ *trace* (*B* ∗ *ϱ*))
proof (*rule iffI*)
have *dimBmA*: *B* − *A* ∈ *carrier-mat n n* **using** *A B* **by** *auto*
{
assume *A* $\leq_L$ *B*
then have *pBmA*: *positive* (*B* − *A*) **using** *lowner-le-def* **by** *auto*
moreover have *B* − *A* ∈ *carrier-mat n n* **using** *assms* **by** *auto*
ultimately have ∃ *M*∈*carrier-mat n n. M* ∗ *adjoint M* = *B* − *A* **using** *positive-iff-decomp*[*of B* − *A*] **by** *auto*
then obtain *M* **where** *dimM*: *M* ∈ *carrier-mat n n* **and** *M*: *M* ∗ *adjoint M* = *B* − *A* **by** *auto*
{
fix *ϱ* **assume** *dimr*: *ϱ* ∈ *carrier-mat n n* **and** *pdr*: *partial-density-operator ϱ*
have *eq*: *trace*(*B* ∗ *ϱ*) − *trace*(*A* ∗ *ϱ*) = *trace*((*B* − *A*) ∗ *ϱ*) **using** *A B dimr*
apply (*subst minus-mult-distrib-mat*, *auto*)
apply (*subst trace-minus-linear*, *auto*)
done
have *pr*: *positive ϱ* **using** *pdr partial-density-operator-def* **by** *auto*
then have ∃ *P*∈*carrier-mat n n. ϱ* = *P* ∗ *adjoint P* **using** *positive-iff-decomp dimr* **by** *auto*
then obtain *P* **where** *dimP*: *P* ∈ *carrier-mat n n* **and** *P*: *ϱ* = *P* ∗ *adjoint P* **by** *auto*

have *trace*$((B - A) * \varrho) = $ *trace*$(M *$ *adjoint* $M * (P *$ *adjoint* $P))$ **using** P M **by** *auto*
also have ... = *trace*((*adjoint* $P * M) *$ *adjoint* (*adjoint* $P * M$)) **using** *dimM dimP* **by** (*mat-assoc n*)
also have ... $\geq$ *0* **using** *trace-adjoint-positive* **by** *auto*
finally have *trace*$((B - A) * \varrho) \geq$ *0*.
with *eq* **have** *trace* $(B * \varrho) -$ *trace* $(A * \varrho) \geq$ *0* **by** *auto*
}
then show $\forall \varrho \in$*carrier-mat n n*. *partial-density-operator* $\varrho \longrightarrow$ *trace* $(A * \varrho)$ $\leq$ *trace* $(B * \varrho)$ **by** *auto*
}

{
assume *asm*: $\forall \varrho \in$*carrier-mat n n*. *partial-density-operator* $\varrho \longrightarrow$ *trace* $(A * \varrho)$ $\leq$ *trace* $(B * \varrho)$
have *positive* $(B - A)$
proof $-$
{
fix v **assume** *dim-vec* (v::*complex vec*) = *dim-col* $(B - A) \wedge$ *vec-norm* v = *1*
then have *dimv*: $v \in$ *carrier-vec n* **and** *nv*: *vec-norm* v = *1*
using *carrier-matD*[*OF dimBmA*] **by** (*auto intro*: *carrier-vecI*)
have *dimov*: *outer-prod v v* $\in$ *carrier-mat n n* **using** *dimv* **by** *auto*
then have *partial-density-operator* (*outer-prod v v*)
using *dimv nv pure-state-self-outer-prod-is-partial-density-operator* **by** *auto*
then have *leq*: *trace*$(A *$ (*outer-prod v v*)) $\leq$ *trace*$(B *$ (*outer-prod v v*)) **using** *asm dimov* **by** *auto*
have *trace*$((B - A) *$ (*outer-prod v v*)) = *trace*$(B *$ (*outer-prod v v*)) $-$ *trace*$(A *$ (*outer-prod v v*)) **using** A B *dimov*
apply (*subst minus-mult-distrib-mat*, *auto*)
apply (*subst trace-minus-linear*, *auto*)
done
then have *trace*$((B - A) *$ (*outer-prod v v*)) $\geq$ *0* **using** *leq* **by** *auto*
then have *inner-prod v* $((B - A) *_v v) \geq$ *0* **using** *trace-outer-prod-right*[*OF dimBmA dimv dimv*] **by** *auto*
}
then show *positive* $(B - A)$ **using** *positive-iff-normalized-vec*[*of* $B - A$] *dimBmA* A **by** *simp*
qed
then show $A \leq_L B$ **using** *lowner-le-def* A B **by** *auto*
}
qed

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 $A \leq_L B$

using *lowner-le-trace assms* **by** *auto*

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 −
  from *teq* **have** $A \leq_L B$ **using** *lowner-le-trace*[*OF A B*] *teq* **by** *auto*
  moreover from *teq* **have** $B \leq_L A$ **using** *lowner-le-trace*[*OF B A*] *teq* **by** *auto*
  ultimately show $A = B$ **using** *lowner-le-antisym A B* **by** *auto*
qed

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* $(B * \varrho)$
  using *lowner-le-trace assms* **by** *blast*

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 −
  have $\{j\} \subseteq A$ **using** *assms* **by** *auto*
  moreover have $\forall i{\in}A - \{j\}.\ g\ i = 0$ **using** *assms* **by** *simp*
  ultimately have *sum g A* = *sum g* $\{j\}$ **using** *assms*
    by (*auto simp add*: *comm-monoid-add-class.sum.mono-neutral-right*[*of A* $\{j\}$ *g*])
  moreover have *sum g* $\{j\}$ = *g j* **by** *simp*
  ultimately show *?thesis* **by** *auto*
qed

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 −
have *dim*: $A \in$ *carrier-mat m m* $B \in$ *carrier-mat m m* **using** *limA limB limit-mat-def* **by** *auto*
{
fix *i j* **assume** *i*: $i < m$ **and** *j*: $j < m$
have $(\lambda\ n.\ (X\ n)\ \$\$\ (i, j)) \longrightarrow (A\ \$\$\ (i, j))$ **using** *limit-mat-def limA i j* **by** *auto*
moreover have $(\lambda\ n.\ (X\ n)\ \$\$\ (i, j)) \longrightarrow (B\ \$\$\ (i, j))$ **using** *limit-mat-def limB i j* **by** *auto*
ultimately have $(A\ \$\$\ (i, j)) = (B\ \$\$\ (i, j))$ **using** *LIMSEQ-unique* **by** *auto*
}
then show $A = B$ **using** *mat-eq-iff dim* **by** *auto*
qed

lemma *limit-mat-const*:
fixes A :: *complex mat*
assumes $A \in$ *carrier-mat m m*
shows *limit-mat* $(\lambda k.\ A)\ A\ m$
unfolding *limit-mat-def* **using** *assms* **by** *auto*

lemma *limit-mat-scale*:
fixes X :: *nat* $\Rightarrow$ *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 −
have *dimA*: $A \in$ *carrier-mat m m* **using** *limX limit-mat-def* **by** *auto*
have *dimX*: $\bigwedge n.\ X\ n \in$ *carrier-mat m m* **using** *limX* **unfolding** *limit-mat-def* **by** *auto*
have $\bigwedge i\ j.\ i < m \Longrightarrow j < m \Longrightarrow (\lambda n.\ (c \cdot_m X\ n)\ \$\$\ (i, j)) \longrightarrow (c \cdot_m A)\ \$\$\ (i, j)$
proof −
fix *i j* **assume** *i*: $i < m$ **and** *j*: $j < m$
have $(\lambda n.\ (X\ n)\ \$\$\ (i, j)) \longrightarrow A\$\$(i, j)$ **using** *limX limit-mat-def i j* **by** *auto*
moreover have $(\lambda n.\ c) \longrightarrow c$ **by** *auto*
ultimately have $(\lambda n.\ c * (X\ n)\ \$\$\ (i, j)) \longrightarrow c * A\$\$(i, j)$
using *tendsto-mult*[*of* $\lambda n.\ c\ c$] *limX limit-mat-def* **by** *auto*
moreover have $(c \cdot_m X\ n)\ \$\$\ (i, j) = c * (X\ n)\ \$\$\ (i, j)$ **for** *n*
using *index-smult-mat(1)*[*of i X n j c*] *i j dimX*[*of n*] **by** *auto*
moreover have $(c \cdot_m A)\ \$\$\ (i, j) = c * A\ \$\$\ (i, j)$
using *index-smult-mat(1)*[*of i A j c*] *i j dimA* **by** *auto*
ultimately show $(\lambda n.\ (c \cdot_m X\ n)\ \$\$\ (i, j)) \longrightarrow (c \cdot_m A)\ \$\$\ (i, j)$ **by** *auto*
qed
then show *?thesis* **unfolding** *limit-mat-def* **using** *dimA dimX* **by** *auto*
qed

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* (λ*k*. *X k* + *Y k*) (*A* + *B*) *m*
proof −
have *dimA*: *A* ∈ *carrier-mat m m* **using** *limX limit-mat-def* **by** *auto*
have *dimB*: *B* ∈ *carrier-mat m m* **using** *limY limit-mat-def* **by** *auto*
have *dimX*: ⋀*n*. *X n* ∈ *carrier-mat m m* **using** *limX* **unfolding** *limit-mat-def* **by** *auto*
have *dimY*: ⋀*n*. *Y n* ∈ *carrier-mat m m* **using** *limY* **unfolding** *limit-mat-def* **by** *auto*
then have *dimXAB*: ∀ *n*. *X n* + *Y n* ∈ *carrier-mat m m* ∧ *A* + *B* ∈ *carrier-mat m m* **using** *dimA dimB dimX dimY*
by (*simp*)

have (⋀*i j*. *i* < *m* ⟹ *j* < *m* ⟹ (λ*n*. (*X n* + *Y n*) $$ (*i*, *j*)) ⟶ (*A* + *B*) $$ (*i*, *j*))
proof −
fix *i j* **assume** *i*: *i* < *m* **and** *j*: *j* < *m*
have (λ*n*. (*X n*) $$ (*i*, *j*)) ⟶ *A*$$(*i*, *j*) **using** *limX limit-mat-def i j* **by** *auto*
moreover have (λ*n*. (*Y n*) $$ (*i*, *j*)) ⟶ *B*$$(*i*, *j*) **using** *limY limit-mat-def i j* **by** *auto*
ultimately have (λ*n*. (*X n*)$$(*i*, *j*) + (*Y n*) $$ (*i*, *j*)) ⟶ (*A*$$(*i*, *j*) + *B*$$(*i*, *j*))
using *tendsto-add*[*of* λ*n*. (*X n*) $$ (*i*, *j*) *A* $$ (*i*, *j*)] **by** *auto*
moreover have (*X n* + *Y n*) $$ (*i*, *j*) = (*X n*)$$(*i*, *j*) + (*Y n*) $$ (*i*, *j*) **for** *n*
using *i j dimX dimY index-add-mat*(*1*)[*of i Y n j X n*] **by** *fastforce*
moreover have (*A* + *B*) $$ (*i*, *j*) = *A*$$(*i*, *j*) + *B*$$(*i*, *j*)
using *i j dimA dimB* **by** *fastforce*
ultimately show (λ*n*. (*X n* + *Y n*) $$ (*i*, *j*)) ⟶ (*A* + *B*) $$ (*i*, *j*) **by** *auto*
qed
then show *?thesis*
unfolding *limit-mat-def* **using** *dimXAB* **by** *auto*
qed

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* (λ*k*. *X k* − *Y k*) (*A* − *B*) *m*
proof −
have *dimA*: *A* ∈ *carrier-mat m m* **using** *limX limit-mat-def* **by** *auto*
have *dimB*: *B* ∈ *carrier-mat m m* **using** *limY limit-mat-def* **by** *auto*
have *dimX*: ⋀*n*. *X n* ∈ *carrier-mat m m* **using** *limX* **unfolding** *limit-mat-def* **by** *auto*

have *dimY*: $\bigwedge n.\ Y\ n \in$ *carrier-mat m m* **using** *limY* **unfolding** *limit-mat-def* **by** *auto*
have $-1 \cdot_m Y\ n = -\ Y\ n$ **for** *n* **using** *dimY* **by** *auto*
moreover have $-1 \cdot_m B = -\ B$ **using** *dimB* **by** *auto*
ultimately have *limit-mat* $(\lambda n.\ -\ Y\ n)\ (-\ B)\ m$ **using** *limit-mat-scale*[*OF limY*, *of* -1] **by** *auto*
then have *limit-mat* $(\lambda n.\ X\ n + (-\ Y\ n))\ (A + (-\ B))\ m$ **using** *limit-mat-add limX* **by** *auto*
moreover have $X\ n + (-\ Y\ n) = X\ n - Y\ n$ **for** *n* **using** *dimX dimY* **by** *auto*
moreover have $A + (-\ B) = A - B$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

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 −
have *dimA*: $A \in$ *carrier-mat m m* **using** *limX limit-mat-def* **by** *auto*
have *dimB*: $B \in$ *carrier-mat m m* **using** *limY limit-mat-def* **by** *auto*
have *dimX*: $\bigwedge n.\ X\ n \in$ *carrier-mat m m* **using** *limX* **unfolding** *limit-mat-def* **by** *auto*
have *dimY*: $\bigwedge n.\ Y\ n \in$ *carrier-mat m m* **using** *limY* **unfolding** *limit-mat-def* **by** *auto*
then have *dimXAB*: $\forall n.\ X\ n * Y\ n \in$ *carrier-mat m m* $\wedge\ A * B \in$ *carrier-mat m m* **using** *dimA dimB dimX dimY*
by *fastforce*

have $(\bigwedge i\ j.\ i < m \Longrightarrow j < m \Longrightarrow (\lambda n.\ (X\ n * Y\ n)\ \$\$\ (i, j)) \longrightarrow (A * B)\ \$\$\ (i, j))$
proof −
fix *i j* **assume** *i*: $i < m$ **and** *j*: $j < m$
have *eqn*: $(X\ n * Y\ n)\ \$\$\ (i, j) = (\sum k=0..<m.\ (X\ n)\$\$(i, k) * (Y\ n)\$\$(k, j))$ **for** *n*
using *i j dimX*[*of n*] *dimY*[*of n*] **by** (*auto simp add*: *scalar-prod-def*)
have *eq*: $(A * B)\ \$\$\ (i, j) = (\sum k=0..<m.\ A\$\$(i,k) * B\$\$(k,j))$
using *i j dimB dimA* **by** (*auto simp add*: *scalar-prod-def*)
have $(\lambda n.\ (X\ n)\ \$\$\ (i, k)) \longrightarrow A\$\$(i, k)$ **if** $k < m$ **for** *k* **using** *limX limit-mat-def that i* **by** *auto*
moreover have $(\lambda n.\ (Y\ n)\ \$\$\ (k, j)) \longrightarrow B\$\$(k, j)$ **if** $k < m$ **for** *k* **using** *limY limit-mat-def that j* **by** *auto*
ultimately have $(\lambda n.\ (X\ n)\$\$(i, k) * (Y\ n)\$\$(k,j)) \longrightarrow A\$\$(i, k) * B\$\$(k, j)$ **if** $k < m$ **for** *k*
using *tendsto-mult*[*of* $\lambda n.\ (X\ n)\ \$\$\ (i, k)\ A\$\$(i, k)$ - $\lambda n.\ (Y\ n)\$\$(k, j)\ B\$\$(k, j)$] *that* **by** *auto*
then have $(\lambda n.\ (\sum k=0..<m.\ (X\ n)\$\$(i,k) * (Y\ n)\$\$(k,j))) \longrightarrow (\sum k=0..<m.\ A\$\$(i,k) * B\$\$(k,j))$

using *tendsto-sum*[*of* {*0..*<*m*} λ*k n.* (*X n*)$$(*i*,*k*) ∗ (*Y n*)$$(*k*,*j*) λ*k. A*$$(*i*, *k*) ∗ *B*$$(*k*, *j*)] **by** *auto*
then show (λ*n.* (*X n* ∗ *Y n*) $$ (*i*, *j*)) ⟶ (*A* ∗ *B*) $$ (*i*, *j*) **using** *eqn eq* **by** *auto*
qed
then show *?thesis*
unfolding *limit-mat-def* **using** *dimXAB* **by** *fastforce*
qed

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 = (λ*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* ∈ *carrier-mat m m* **and** *limX*: *limit-mat X A m*
shows *limit-mat* (*mat-add-seq B X*) (*B* + *A*) *m*
unfolding *mat-add-seq-def* **using** *limit-mat-add limit-mat-const*[*OF dimB*] *limX* **by** *auto*

lemma *mat-minus-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* (λ*n. B* − *X n*) (*B* − *A*) *m*
using *limit-mat-minus limit-mat-const*[*OF dimB*] *limX* **by** *auto*

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 = (λ*n. A* ∗ *X n*)

lemma *mat-mult-limit*:
fixes *X* :: *nat* ⇒ *complex mat* **and** *A B* :: *complex mat* **and** *m* :: *nat*
assumes *dimB*: *B* ∈ *carrier-mat m m* **and** *limX*: *limit-mat X A m*
shows *limit-mat* (*mat-mult-seq B X*) (*B* ∗ *A*) *m*
unfolding *mat-mult-seq-def* **using** *limit-mat-mult limit-mat-const*[*OF dimB*] *limX* **by** *auto*

lemma *mult-mat-limit*:
fixes *X* :: *nat* ⇒ *complex mat* **and** *A B* :: *complex mat* **and** *m* :: *nat*
assumes *dimB*: *B* ∈ *carrier-mat m m* **and** *limX*: *limit-mat X A m*
shows *limit-mat* (λ*k. X k* ∗ *B*) (*A* ∗ *B*) *m*
unfolding *mat-mult-seq-def* **using** *limit-mat-mult limit-mat-const*[*OF dimB*] *limX* **by** *auto*

lemma *quadratic-form-mat*:

fixes *A* :: *complex mat* **and** *v* :: *complex vec* **and** *m* :: *nat*
assumes *dimv*: *dim-vec v* = *m* **and** *dimA*: *A* ∈ *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 −
have *inner-prod v* $(A *_v v) = (\sum i{=}0..{<}m.\ (\sum j{=}0..{<}m.$
conjugate $(v\$i) * A\$\$(i, j) * v\$j))$
unfolding *scalar-prod-def* **using** *dimv dimA*
apply (*simp add*: *scalar-prod-def sum-distrib-right*)
apply (*rule sum.cong*, *auto*, *rule sum.cong*, *auto*)
done
then show *?thesis* **by** *auto*
qed

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 −
have $\forall\ x \in A.\ (\sum y{\in}B.\ h\ x\ y - g\ x\ y) = (\sum y{\in}B.\ h\ x\ y) - (\sum y{\in}B.\ g\ x\ y)$
proof −
{
fix *x* **assume** *x*: *x* ∈ *A*
have $(\sum y{\in}B.\ h\ x\ y - g\ x\ y) = (\sum y{\in}B.\ h\ x\ y) - (\sum y{\in}B.\ g\ x\ y)$
using *sum-subtractf* **by** *auto*
}
then show *?thesis* **using** *sum-subtractf* **by** *blast*
qed
then have $(\sum x{\in}A.\sum y{\in}B.\ h\ x\ y - g\ x\ y) = (\sum x{\in}A.\ ((\sum y{\in}B.\ h\ x\ y) - (\sum y{\in}B.\ g\ x\ y)))$ **by** *auto*
also have ... $= (\sum x{\in}A.\ \sum y{\in}B.\ h\ x\ y) - (\sum x{\in}A.\ \sum y{\in}B.\ g\ x\ y)$
by (*simp add*: *sum-subtractf*)
finally have $(\sum x{\in}A.\ \sum y{\in}B.\ h\ x\ y - g\ x\ y) = (\sum x{\in}A.\ sum\ (h\ x)\ B) - (\sum x{\in}A.\ sum\ (g\ x)\ B)$ **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *sum-abs-complex*:
fixes *h* :: *nat* ⇒ *nat* ⇒ *complex*
shows *cmod* $(\sum x{\in}A.\sum y{\in}B.\ h\ x\ y) \le (\sum x{\in}A.\ \sum y{\in}B.\ cmod(h\ x\ y))$
proof −
have *B*: $\forall\ x \in A.\ cmod(\ \sum y{\in}B\ .h\ x\ y) \le (\sum y{\in}B.\ cmod(h\ x\ y))$ **using** *sum-abs norm-sum* **by** *blast*
have *cmod* $(\sum x{\in}A.\sum y{\in}B.\ h\ x\ y) \le (\sum x{\in}A.\ \ cmod(\ \sum y{\in}B\ .h\ x\ y))$ **using** *sum-abs norm-sum* **by** *blast*
also have ... $\le (\sum x{\in}A.\ \sum y{\in}B.\ cmod(h\ x\ y))$ **using** *sum-abs norm-sum B*
by (*simp add*: *sum-mono*)
finally have *cmod* $(\sum x{\in}A.\ \sum y{\in}B.\ h\ x\ y) \le (\sum x{\in}A.\ \sum y{\in}B.\ cmod\ (h\ x\ y))$ **by** *auto*

then show *?thesis* **by** *auto*
qed

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 −
have *dimX*: ∀ *n*. *X n* ∈ *carrier-mat m m* **using** *limX* **unfolding** *limit-mat-def* **by** *auto*
have *dimA* : *A* ∈ *carrier-mat m m* **using** *limX* **unfolding** *limit-mat-def* **by** *auto*

from *herX* **have** *herXn*: ∀ *n*. *adjoint* (*X n*) = (*X n*) **unfolding** *hermitian-def* **by** *auto*
from *limX* **have** *limXn*: ∀ *i*<*m*. ∀ *j*<*m*. (λ*n*. *X n* $$ (*i*, *j*)) ⟶ *A* $$ (*i*, *j*) **unfolding** *limit-mat-def* **by** *auto*
have ∀ *i*<*m*. ∀ *j*<*m*.(*adjoint A*)$$ (*i*, *j*) = *A*$$ (*i*, *j*)
proof −
{
fix *i j* **assume** *i*: *i* < *m* **and** *j*: *j* < *m*
have *aij*: (*adjoint A*)$$ (*i*, *j*) = *conjugate* (*A* $$ (*j*,*i*)) **using** *adjoint-eval i j dimA* **by** *blast*
have *ij*: (λ*n*. *X n* $$ (*i*, *j*)) ⟶ *A* $$ (*i*, *j*) **using** *limXn i j* **by** *auto*
have *ji*: (λ*n*. *X n* $$ (*j*, *i*)) ⟶ *A* $$ (*j*, *i*) **using** *limXn i j* **by** *auto*
then have ∀ *r*>*0*. ∃ *no*. ∀ *n*≥*no*. *dist* (*conjugate* (*X n* $$ (*j*, *i*))) (*conjugate* (*A* $$ (*j*, *i*))) < *r*
proof −
{
fix *r* :: *real* **assume** *r* : *r* > *0*
have ∃ *no*. ∀ *n*≥*no*. *cmod* (*X n* $$ (*j*, *i*) − *A* $$ (*j*, *i*)) < *r* **using** *ji r* **unfolding** *LIMSEQ-def dist-norm* **by** *auto*
then obtain *no* **where** *Xji*: ∀ *n*≥*no*. *cmod* (*X n* $$ (*j*, *i*) − *A* $$ (*j*, *i*)) < *r* **by** *auto*
then have ∀ *n*≥*no*. *cmod* (*conjugate* (*X n* $$ (*j*, *i*) − *A* $$ (*j*, *i*))) < *r*
using *complex-mod-cnj conjugate-complex-def* **by** *presburger*
then have ∀ *n*≥*no*. *dist* (*conjugate* (*X n* $$ (*j*, *i*))) (*conjugate* (*A* $$ (*j*, *i*))) < *r* **unfolding** *dist-norm* **by** *auto*
then have ∃ *no*. ∀ *n*≥*no*. *dist* (*conjugate* (*X n* $$ (*j*, *i*))) (*conjugate* (*A* $$ (*j*, *i*))) < *r* **by** *auto*
}
then show *?thesis* **by** *auto*
qed
then have *conjX*: (λ*n*. *conjugate* (*X n* $$ (*j*, *i*))) ⟶ *conjugate* (*A* $$ (*j*, *i*)) **unfolding** *LIMSEQ-def* **by** *auto*

from *herXn* **have** ∀ *n*. *conjugate* (*X n* $$ (*j*,*i*)) = *X n*$$ (*i*, *j*) **using** *adjoint-eval i j dimX*
by (*metis adjoint-dim-col carrier-matD*(*1*))
then have (λ*n*. *X n* $$ (*i*, *j*)) ⟶ *conjugate* (*A* $$ (*j*, *i*)) **using** *conjX*

```
by auto
        then have conjugate (A $$ (j,i)) = A$$ (i, j) using ij by (simp add:
LIMSEQ-unique)
      then have (adjoint A)$$ (i, j) = A$$ (i, j) using adjoint-eval i j by (simp
add:aij)
    }
    then show ?thesis by auto
  qed
  then have hermitian A using hermitian-def dimA
    by (metis adjoint-dim carrier-matD(1) carrier-matD(2) eq-matI)
  then show ?thesis by auto
qed

lemma quantifier-change-order-once:
  fixes P :: nat ⇒ nat ⇒ bool and m :: nat
  shows ∀ j<m. ∃ no. ∀ n≥no. P n j ⟹ ∃ no. ∀ j<m. ∀ n≥no. P n j
proof (induct m)
    case 0
    then show ?case by auto
  next
    case (Suc m)
    then show ?case
    proof −
      have mm: ∃ no. ∀ j<m. ∀ n≥no. P n j using Suc by auto
      then obtain M where MM: ∀ j<m. ∀ n≥M. P n j by auto
      have sucm: ∃ no. ∀ n≥no. P n m using Suc(2) by auto
      then obtain N where NN: ∀ n≥N. P n m by auto
      let ?N = max M N
      from MM NN have ∀ j<Suc m. ∀ n≥?N. P n j
        by (metis less-antisym max.boundedE)
      then have ∃ no. ∀ j<Suc m. ∀ n≥no. P n j by blast
      then show ?thesis by auto
    qed
  qed

lemma quantifier-change-order-twice:
  fixes P :: nat ⇒ nat ⇒ nat ⇒ bool and m n :: nat
  shows ∀ i<m. ∀ j<n. ∃  no. ∀ n≥no. P n i j ⟹ ∃ no. ∀ i<m. ∀ j<n. ∀ n≥no. P
n i j
proof −
  assume fact: ∀ i<m. ∀ j<n. ∃  no. ∀ n≥no. P n i j
  have one: ∀ i<m. ∃ no.∀ j<n. ∀ n≥no. P n i j
    using fact quantifier-change-order-once by auto
  have two: ∀ i<m. ∃ no.∀ j<n. ∀ n≥no. P n i j ⟹ ∃ no. ∀ i<m. ∀ j<n. ∀ n≥no.
P n i j
  proof (induct m)
    case 0
    then show ?case by auto
  next
```

```
    case (Suc m)
    then show ?case
    proof –
      obtain M where MM: ∀ i<m. ∀ j<n. ∀ n≥M. P n i j using Suc by auto
      obtain N where NN: ∀ j<n. ∀ n≥N. P n m j using Suc(2) by blast
      let ?N = max M N
      from MM NN have ∀ i<Suc m. ∀ j<n. ∀ n≥?N. P n i j
        by (metis less-antisym max.boundedE)
      then have ∃ no. ∀ i<Suc m. ∀ j<n. ∀ n≥no. P n i j by blast
      then show ?thesis by auto
    qed
  qed
  with fact show ?thesis using one by auto
qed

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: ∀ n. positive (X n)
  shows positive A
proof (rule ccontr)
  have  dimX : ∀ n. X n ∈ carrier-mat m m using limX unfolding limit-mat-def
by auto
  have dimA : A ∈ carrier-mat m m using limX unfolding limit-mat-def by auto
  have herX : ∀  n. hermitian (X n) using posX positive-is-hermitian by auto
  then have herA : hermitian A  using hermitian-mat-lim-is-hermitian limX by
auto
  then have herprod: ∀  v. dim-vec v = dim-col A ⟶ inner-prod v (A *_v v) ∈
Reals
    using hermitian-inner-prod-real dimA by auto

  assume npA:  ¬ positive A
  from npA have ¬ (A ∈ carrier-mat (dim-col A) (dim-col A)) ∨ ¬ (∀ v. dim-vec
v = dim-col A ⟶ 0 ≤ inner-prod v (A *_v v))
    unfolding positive-def by blast
  then have evA: ∃  v. dim-vec v = dim-col A ∧ ¬ inner-prod v (A *_v v) ≥ 0
using dimA by blast
  then have ∃  v. dim-vec v = dim-col A ∧  inner-prod v (A *_v v) < 0
  proof –
     obtain v where vA: dim-vec v = dim-col A ∧ ¬ inner-prod v (A *_v v) ≥ 0
using evA by auto
    from vA herprod have ¬ 0 ≤ inner-prod v (A *_v v) ∧ inner-prod v (A *_v v)
∈ Reals by auto
    then have inner-prod v (A *_v v) < 0
     using complex-is-Real-iff by (auto simp: less-complex-def less-eq-complex-def)
    then have  ∃  v. dim-vec v = dim-col A ∧  inner-prod v (A *_v v) < 0 using
vA by auto
    then show ?thesis by auto
  qed
```

then obtain *v* **where** *neg*: *dim-vec v* = *dim-col A* ∧ *inner-prod v* (*A* $*_v$ *v*) < *0* **by** *auto*

have *nzero*: *v* ≠ 0_v *m*
proof (*rule ccontr*)
assume *nega*: ¬ *v* ≠ 0_v *m*
have *zero*: *v* = 0_v *m* **using** *nega* **by** *auto*
have (*A* $*_v$ *v*) = 0_v *m* **unfolding** *mult-mat-vec-def* **using** *zero* **using** *dimA* **by** *auto*
then have *zerov*: *inner-prod v* (*A* $*_v$ *v*) = *0* **by** (*simp add*: *zero*)
from *neg zerov* **have** ¬ *v* ≠ 0_v *m* ⟹ *False* **using** *dimA* **by** *auto*
with *nega* **show** *False* **by** *auto*
qed

have *invgeq*: *inner-prod v v* > *0*
proof −
have *inner-prod v v* = *vec-norm v* ∗ *vec-norm v* **unfolding** *vec-norm-def* **by** (*metis carrier-matD*(*2*) *carrier-vec-dim-vec dimA mult-cancel-left1 neg normalized-cscalar-prod normalized-vec-norm nzero vec-norm-def*)
moreover have *vec-norm v* > *0* **using** *nzero vec-norm-ge-0 neg dimA* **by** (*metis carrier-matD*(*2*) *carrier-vec-dim-vec*)
ultimately have *inner-prod v v* > *0* **by** (*auto simp*: *less-eq-complex-def less-complex-def*)
then show *?thesis* **by** *auto*
qed

have *invv*: *inner-prod v v* = (∑ *i* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*) ∗ (*v* $ *i*)))
proof −
{
have ∀ *i* < *m*. *conjugate* (*v* $ *i*) ∗ (*v* $ *i*) ≥ *0* **using** *conjugate-square-smaller-0* **by** (*simp add*: *less-eq-complex-def*)
then have *vi*: ∀ *i* < *m*. *conjugate* (*v* $ *i*) ∗ (*v* $ *i*) = *cmod* (*conjugate* (*v* $ *i*) ∗ (*v* $ *i*)) **using** *cmod-eq-Re* **by** (*simp add*: *complex.expand*)

have *inner-prod v v*= (∑ *i* = *0*..<*m*. ((*v* $ *i*) ∗ *conjugate* (*v* $ *i*))) **unfolding** *scalar-prod-def conjugate-vec-def* **using** *neg dimA* **by** *auto*
also have . . . = (∑ *i* = *0*..<*m*. (*conjugate* (*v* $ *i*) ∗ (*v* $ *i*))) **by** (*meson mult.commute*)
also have . . . = (∑ *i* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*) ∗ (*v* $ *i*))) **using** *vi* **by** *auto*
finally have *inner-prod v v* = (∑ *i* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*) ∗ (*v* $ *i*))) **by** *auto*
}
then show *?thesis* **by** *auto*
qed

let *?r* = *inner-prod v* (*A* $*_v$ *v*) **have** *rl*: *?r* < *0* **using** *neg* **by** *auto*
have *vAv*: *inner-prod v* (*A* $*_v$ *v*) = (∑ *i*=*0*..<*m*. (∑ *j*=*0*..<*m*.

conjugate (*v*\$*i*) ∗ *A*\$\$(*i*, *j*) ∗ *v*\$*j*)) **using** *quadratic-form-mat dimA neg* **by** *auto*
from *limX* **have** *limij*: ∀ *i*<*m*. ∀ *j*<*m*. (λ*n*. *X n* \$\$ (*i*, *j*)) ⟶ *A* \$\$ (*i*, *j*) **unfolding** *limit-mat-def* **by** *auto*
then have *limXv*: (λ *n*. *inner-prod v* ((*X n*) $*_v$ *v*)) ⟶ *inner-prod v* (*A* $*_v$ *v*)
proof −
have *XAless*: *cmod* (*inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) ≤
(∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*. *cmod* (*conjugate* (*v* \$ *i*)) ∗ *cmod* (*X n* \$\$ (*i*, *j*) − *A* \$\$ (*i*, *j*)) ∗ *cmod* (*v* \$ *j*)) **for** *n*
proof −
have ∀ *i* < *m*. ∀ *j* < *m*. *conjugate* (*v*\$*i*) ∗ *X n* \$\$(*i*, *j*) ∗ *v*\$*j* − *conjugate* (*v*\$*i*) ∗ *A*\$\$(*i*, *j*) ∗ *v*\$*j* =
conjugate (*v*\$*i*) ∗ (*X n* \$\$(*i*, *j*)−*A*\$\$(*i*, *j*)) ∗ *v*\$*j*
by (*simp add*: *mult.commute right-diff-distrib*)
then have *ele*: ∀ *i* < *m*.(∑ *j*=*0*..<*m*.(*conjugate* (*v*\$*i*) ∗ *X n* \$\$(*i*, *j*) ∗ *v*\$*j* − *conjugate* (*v*\$*i*) ∗ *A*\$\$(*i*, *j*) ∗ *v*\$*j*)) = (∑ *j*=*0*..<*m*.(
conjugate (*v*\$*i*) ∗ (*X n* \$\$(*i*, *j*)−*A*\$\$(*i*, *j*)) ∗ *v*\$*j*)) **by** *auto*
have ∀ *i* < *m*. ∀ *j* < *m*. *cmod*(*conjugate* (*v* \$ *i*) ∗ (*X n* \$\$ (*i*, *j*) − *A* \$\$ (*i*, *j*)) ∗ *v* \$ *j*) =
cmod(*conjugate* (*v* \$ *i*)) ∗ *cmod* (*X n* \$\$ (*i*, *j*) − *A* \$\$ (*i*, *j*)) ∗ *cmod*(*v* \$ *j*)
by (*simp add*: *norm-mult*)
then have *less*: ∀ *i* < *m*.(∑ *j* = *0*..<*m*. *cmod*(*conjugate* (*v* \$ *i*) ∗ (*X n* \$\$ (*i*, *j*) − *A* \$\$ (*i*, *j*)) ∗ *v* \$ *j*)) =
(∑ *j* = *0*..<*m*. *cmod*(*conjugate* (*v* \$ *i*)) ∗ *cmod* (*X n* \$\$ (*i*, *j*) − *A* \$\$ (*i*, *j*)) ∗ *cmod*(*v* \$ *j*)) **by** *auto*

have *inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*) = (∑ *i*=*0*..<*m*. (∑ *j*=*0*..<*m*.
conjugate (*v*\$*i*) ∗ *X n* \$\$(*i*, *j*) ∗ *v*\$*j*)) − (∑ *i*=*0*..<*m*. (∑ *j*=*0*..<*m*.
conjugate (*v*\$*i*) ∗ *A*\$\$(*i*, *j*) ∗ *v*\$*j*)) **using** *quadratic-form-mat neg dimA dimX* **by** *auto*
also have ... = (∑ *i*=*0*..<*m*. (∑ *j*=*0*..<*m*.(
conjugate (*v*\$*i*) ∗ *X n* \$\$(*i*, *j*) ∗ *v*\$*j* − *conjugate* (*v*\$*i*) ∗ *A*\$\$(*i*, *j*) ∗ *v*\$*j*)))
using *sum-subtractff*[*of* λ *i j*. *conjugate* (*v* \$ *i*) ∗ *X n* \$\$ (*i*, *j*) ∗ *v* \$ *j* λ *i j*. *conjugate* (*v* \$ *i*) ∗ *A* \$\$ (*i*, *j*) ∗ *v* \$ *j* {*0*..<*m*}] **by** *auto*
also have ... = (∑ *i*=*0*..<*m*. (∑ *j*=*0*..<*m*.(
conjugate (*v*\$*i*) ∗ (*X n* \$\$(*i*, *j*)−*A*\$\$(*i*, *j*)) ∗ *v*\$*j*))) **using** *ele* **by** *auto*
finally have *minusXA*: *inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*) = (∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*. *conjugate* (*v* \$ *i*) ∗ (*X n* \$\$ (*i*, *j*) − *A* \$\$ (*i*, *j*)) ∗ *v* \$ *j*) **by** *auto*

from *minusXA* **have** *cmod* (*inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) =
cmod (∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*. *conjugate* (*v* \$ *i*) ∗ (*X n* \$\$ (*i*, *j*) − *A* \$\$ (*i*, *j*)) ∗ *v* \$ *j*) **by** *auto*
also have ... ≤ (∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*. *cmod*(*conjugate* (*v* \$ *i*) ∗ (*X*

n $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *v* $ *j*))
using *sum-abs-complex* **by** *simp*
also have . . . = (∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*. *cmod*(*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod*(*v* $ *j*))
using *less* **by** *auto*
finally show *?thesis* **by** *auto*
qed

from *limij* **have** *limijm*: ∀ *i*<*m*. ∀ *j*<*m*. ∀ *r*>*0*. ∃ *no*. ∀ *n*≥*no*. *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) < *r*
unfolding *LIMSEQ-def dist-norm* **by** *auto*
from *limX* **have** *mg*: *m* > *0* **using** *limit-mat-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *mat-eq-iff neq0-conv not-less0 npA posX*)

have *cmoda*: ∃ *no*. ∀ *n*≥*no*. (∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod* (*v* $ *j*)) < *r*
if *r*: *r* > *0* **for** *r*
proof −
let *?u* = (∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*.((*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*))))
have *ug*: *?u* > *0*
proof −
have *ur*: *?u* = (∑ *i* = *0*..<*m*. (*cmod* (*conjugate* (*v* $ *i*)) ∗ (∑ *j* = *0*..<*m*.(*cmod* (*v* $ *j*))))) **by** (*simp add*: *sum-distrib-left*)
have (∑ *j* = *0*..<*m*.(*cmod* (*v* $ *j*))) ≥ *cmod* (*v* $ *i*) **if** *i*: *i* < *m* **for** *i*
using *member-le-sum*[*of i* {*0*..<*m*} λ *j*. *cmod* (*v*$*j*)] *cmod-def i* **by** *simp*
then have ∀ *i* < *m*. (*cmod* (*conjugate* (*v* $ *i*)) ∗ (∑ *j* = *0*..<*m*.(*cmod* (*v* $ *j*)))) ≥ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *i*))
by (*simp add*: *mult-left-mono*)
then have *?u* ≥ (∑ *i* = *0*..<*m*. (*cmod* (*conjugate* (*v* $ *i*)) ∗*cmod* (*v* $ *i*)))
using *ur sum-mono*[*of* {*0*..<*m*} λ *i*. *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *i*) λ *i*. *cmod* (*conjugate* (*v* $ *i*)) ∗ (∑ *j* = *0*..<*m*. *cmod* (*v* $ *j*))]
by *auto*
moreover have (∑ *i* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*) ∗*cmod* (*v* $ *i*))) = (∑ *i* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*) ∗ (*v* $ *i*)))
using *norm-ge-zero norm-mult norm-of-real* **by** (*metis* (*no-types*, *opaque-lifting*) *abs-of-nonneg*)
moreover have (∑ *i* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*) ∗ (*v* $ *i*))) = *inner-prod v v* **using** *invv* **by** *auto*
ultimately have *?u* ≥ *inner-prod v v*
by (*metis* (*no-types*, *lifting*) *Im-complex-of-real Re-complex-of-real invv less-eq-complex-def norm-mult sum.cong*)
then have *?u* > *0* **using** *invgeq* **by** (*auto simp*: *less-eq-complex-def less-complex-def*)
then show *?thesis* **by** *auto*
qed

let *?s* = *r* / (*2* ∗ *?u*)

have *sgz*: *?s* > *0* **using** *ug rl*
by (*smt* (*verit*) *divide-pos-pos dual-order.strict-iff-order linordered-semiring-strict-class.mult-pos-pos zero-less-norm-iff r*)
from *limijm* **have** *sij*: ∃ *no*. ∀ *n*≥*no*. *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) < *?s* **if** *i*: *i* < *m* **and** *j*: *j* < *m* **for** *i j*
proof −
obtain *N* **where** *Ns*: ∀ *n*≥*N*. *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) < *?s* **using** *sgz limijm i j* **by** *blast*
then show *?thesis* **by** *auto*
qed
then have ∃ *no*. ∀ *i*<*m*. ∀ *j*<*m*. ∀ *n*≥*no*. *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) < *?s*
using *quantifier-change-order-twice*[*of m m* λ *n i j*. (*cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*))<*?s*)] **by** *auto*
then obtain *N* **where** *Nno*: ∀ *i*<*m*. ∀ *j*<*m*. ∀ *n*≥*N*. *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) < *?s* **by** *auto*
then have *mmN*: *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod* (*v* $ *j*)
≤ *?s* ∗ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*))
if *i*: *i* < *m* **and** *j*: *j* < *m* **and** *n*: *n* ≥ *N* **for** *i j n*
proof −
have *geq*: *cmod* (*conjugate* (*v* $ *i*)) ≥ *0* ∧ *cmod* (*v* $ *j*)≥*0* **by** *simp*
then have *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ≤*cmod* (*conjugate* (*v* $ *i*)) ∗ *?s* **using** *Nno i j n*
by (*smt* (*verit*) *mult-left-mono*)
then have *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod* (*v* $ *j*)
≤ *cmod* (*conjugate* (*v* $ *i*)) ∗*?s* ∗ *cmod* (*v* $ *j*) **using** *geq mult-right-mono* **by** *blast*
also have . . . = *?s* ∗ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*)) **by** *simp*
finally show *?thesis* **by** *auto*
qed
then have (∑ *i* = *0*..<*m*. ∑ *j* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod* (*v* $ *j*)) < *r*
if *n*: *n* ≥ *N* **for** *n*
proof −
have *mmX*: ∀ *i*<*m*. ∀ *j*<*m*. *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod* (*v* $ *j*)
≤ *?s* ∗ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*)) **using** *n mmN* **by** *blast*
have (∑ *j* = *0*..<*m*. *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod* (*v* $ *j*))
≤ (∑ *j* = *0*..<*m*.(*?s* ∗ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*)))) **if** *i*: *i* < *m* **for** *i*
proof −
have ∀ *j*<*m*. *cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i*, *j*) − *A* $$ (*i*, *j*)) ∗ *cmod* (*v* $ *j*)
≤ *?s* ∗ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*)) **using** *mmX i* **by** *auto*

then show *?thesis*
using *sum-mono*[*of* {*0..<m*} *λ j. cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i, j*) − *A* $$ (*i, j*)) ∗ *cmod* (*v* $ *j*) *λ j.* (*?s* ∗ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*)))]
atLeastLessThan-iff **by** *blast*
qed
then have ($\sum$ *i* = *0..<m.* $\sum$ *j* = *0..<m. cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i, j*) − *A* $$ (*i, j*)) ∗ *cmod* (*v* $ *j*))
≤ ($\sum$ *i* = *0..<m.* $\sum$ *j* = *0..<m.*(*?s* ∗ (*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*)))) **using** *sum-mono atLeastLessThan-iff*
by (*metis* (*no-types, lifting*))
also have ... = *?s* ∗ ($\sum$ *i* = *0..<m.* $\sum$ *j* = *0..<m.*((*cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*v* $ *j*)))) **by** (*simp add*: *sum-distrib-left*)
also have ... = *r* / *2* **using** *nonzero-mult-divide-mult-cancel-right sgz* **by** *fastforce*
finally show *?thesis* **using** *r* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed
then have *XnAv*:∃ *no.* ∀ *n*≥*no. cmod* (*inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) < *r* **if** *r*: *r* > *0* **for** *r*
proof −
obtain *no* **where** *nno*: ∀ *n*≥*no.* ($\sum$ *i* = *0..<m.* $\sum$ *j* = *0..<m. cmod* (*conjugate* (*v* $ *i*)) ∗ *cmod* (*X n* $$ (*i, j*) − *A* $$ (*i, j*)) ∗ *cmod* (*v* $ *j*)) < *r*
using *r cmoda neg* **by** *auto*
then have ∀ *n*≥*no. cmod* (*inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) < *r* **using** *XAless neg* **by** (*smt* (*verit*))
then show *?thesis* **by** *auto*
qed
then have (*λn. inner-prod v* (*X n* $*_v$ *v*)) ⟶ *inner-prod v* (*A* $*_v$ *v*) **unfolding** *LIMSEQ-def dist-norm* **by** *auto*
then show *?thesis* **by** *auto*
qed

from *limXv* **have** ∀ *r*>*0.* ∃ *no.* ∀ *n*≥*no. cmod* (*inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) < *r* **unfolding** *LIMSEQ-def dist-norm* **by** *auto*
then have ∃ *no.* ∀ *n*≥*no. cmod* (*inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) < −*?r* **using** *rl*
by (*auto simp*: *less-eq-complex-def less-complex-def*)
then obtain *N* **where** *Ng*: ∀ *n*≥*N. cmod* (*inner-prod v* (*X n* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) < −*?r* **by** *auto*
then have *XN*: *cmod* (*inner-prod v* (*X N* $*_v$ *v*) − *inner-prod v* (*A* $*_v$ *v*)) < −*?r* **by** *auto*

from *posX* **have** *positive* (*X N*) **by** *auto*
then have *XNv*:*inner-prod v* (*X N* $*_v$ *v*) ≥ *0*
by (*metis Complex-Matrix.positive-def carrier-matD*(*2*) *dimA dimX neg*)

from *rl XNv* **have** *XX*: *cmod* (*inner-prod v* (*X N* $*_v$ *v*) − *inner-prod v* (*A* $*_v$

v)) = *cmod*(*inner-prod v* (*X N* $*_v$ *v*)) − *cmod*(*inner-prod v* (*A* $*_v$ *v*))
using *XN cmod-eq-Re* **by** (*auto simp*: *less-eq-complex-def less-complex-def*)
then have *YY*: *cmod*(*inner-prod v* (*X N* $*_v$ *v*)) − *cmod*(*inner-prod v* (*A* $*_v$ *v*)) < −*?r* **using** *XN* **by** *auto*
then have *cmod*(*inner-prod v* (*X N* $*_v$ *v*)) − *cmod*(*inner-prod v* (*A* $*_v$ *v*)) < *cmod*(*inner-prod v* (*A* $*_v$ *v*))
using *rl cmod-eq-Re* **by** (*auto simp*: *less-eq-complex-def less-complex-def*)
then have *cmod*(*inner-prod v* (*X N* $*_v$ *v*)) < *0* **using** *XNv XX YY cmod-eq-Re*
by (*auto simp*: *less-eq-complex-def less-complex-def*)
then have *False* **using** *XNv* **by** *simp*
with *npA* **show** *False* **by** *auto*
qed

lemma *limit-mat-ignore-initial-segment*:
limit-mat g A d ⟹ *limit-mat* (λ*n*. *g* (*n* + *k*)) *A d*
proof −
assume *asm*: *limit-mat g A d*
then have *lim*: ∀ *i* < *d*. ∀ *j* < *d*. (λ *n*. (*g n*) $$ (*i*, *j*)) ⟶ (*A* $$ (*i*, *j*)) **using** *limit-mat-def* **by** *auto*
then have *limk*: ∀ *i* < *d*. ∀ *j* < *d*. (λ *n*. (*g* (*n* + *k*)) $$ (*i*, *j*)) ⟶ (*A* $$ (*i*, *j*))
proof −
{
fix *i j* **assume** *dims*: *i* < *d j* < *d*
then have (λ *n*. (*g n*) $$ (*i*, *j*)) ⟶ (*A* $$ (*i*, *j*)) **using** *lim* **by** *auto*
then have (λ *n*. (*g* (*n* + *k*)) $$ (*i*, *j*)) ⟶ (*A* $$ (*i*, *j*)) **using** *LIMSEQ-ignore-initial-segment* **by** *auto*
}
then show ∀ *i* < *d*. ∀ *j* < *d*. (λ *n*. (*g* (*n* + *k*)) $$ (*i*, *j*)) ⟶ (*A* $$ (*i*, *j*)) **by** *auto*
qed
have ∀ *n*. *g n* ∈ *carrier-mat d d* **using** *asm* **unfolding** *limit-mat-def* **by** *auto*
then have ∀ *n*. *g* (*n* + *k*) ∈ *carrier-mat d d* **by** *auto*
moreover have *A* ∈ *carrier-mat d d* **using** *asm limit-mat-def* **by** *auto*
ultimately show *limit-mat* (λ*n*. *g* (*n* + *k*)) *A d* **using** *limit-mat-def limk* **by** *auto*
qed

lemma *mat-trace-limit*:
limit-mat g A d ⟹ (λ*n*. *trace* (*g n*)) ⟶ *trace A*
proof −
assume *lim*: *limit-mat g A d*
then have *dgn*: *g n* ∈ *carrier-mat d d* **for** *n* **using** *limit-mat-def* **by** *auto*
from *lim* **have** *dA*: *A* ∈ *carrier-mat d d* **using** *limit-mat-def* **by** *auto*
have *trg*: *trace* (*g n*) = (∑*k*=*0*..<*d*. (*g n*)$$(*k*, *k*)) **for** *n* **unfolding** *trace-def* **using** *carrier-matD*[*OF dgn*] **by** *auto*
have ∀*k* < *d*. (λ*n*. (*g n*)$$(*k*, *k*)) ⟶ *A*$$(*k*, *k*) **using** *limit-mat-def lim* **by** *auto*
then have (λ*n*. (∑*k*=*0*..<*d*. (*g n*)$$(*k*, *k*))) ⟶ (∑*k*=*0*..<*d*. *A*$$(*k*, *k*))

using *tendsto-sum*[**where** *?I* = {*0*..<*d*} **and** *?f* = (λ*k n*. (*g n*)$$(*k*, *k*))] **by** *auto*
then show (λ*n*. *trace* (*g n*)) ⟶ *trace A* **unfolding** *trace-def*
using *trg carrier-matD*[*OF dgn*] *carrier-matD*[*OF dA*] **by** *auto*
qed

2.2 Existence of least upper bound for the Löwner order

definition *lowner-is-lub* :: (*nat* ⇒ *complex mat*) ⇒ *complex mat* ⇒ *bool* **where**
lowner-is-lub f M ⟷ (∀ *n*. *f n* $\le_L$ *M*) ∧ (∀ *M′*. (∀ *n*. *f n* $\le_L$ *M′*) ⟶ *M* $\le_L$ *M′*)

locale *matrix-seq* =
fixes *dim* :: *nat*
and *f* :: *nat* ⇒ *complex mat*
assumes
dim: ⋀*n*. *f n* ∈ *carrier-mat dim dim* **and**
pdo: ⋀*n*. *partial-density-operator* (*f n*) **and**
inc: ⋀*n*. *lowner-le* (*f n*) (*f* (*Suc n*))
begin

definition *lowner-is-lub* :: *complex mat* ⇒ *bool* **where**
lowner-is-lub M ⟷ (∀ *n*. *f n* $\le_L$ *M*) ∧ (∀ *M′*. (∀ *n*. *f n* $\le_L$ *M′*) ⟶ *M* $\le_L$ *M′*)

lemma *lowner-is-lub-dim*:
assumes *lowner-is-lub M*
shows *M* ∈ *carrier-mat dim dim*
proof −
have *f 0* $\le_L$ *M* **using** *assms lowner-is-lub-def* **by** *auto*
then have *1*: *dim-row* (*f 0*) = *dim-row M* ∧ *dim-col* (*f 0*) = *dim-col M*
using *lowner-le-def* **by** *auto*
moreover have *2*: *f 0* ∈ *carrier-mat dim dim*
using *dim* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *trace-adjoint-eq-u*:
fixes *A* :: *complex mat*
shows *trace* (*A* ∗ *adjoint A*) = (∑ *i* ∈ {*0* ..< *dim-row A*}. ∑ *j* ∈ {*0* ..< *dim-col A*}. (*norm*(*A* $$ (*i*,*j*)))^2)
proof −
have *trace* (*A* ∗ *adjoint A*) = (∑ *i* ∈ {*0* ..< *dim-row A*}. *row A i* · *conjugate* (*row A i*))
by (*simp add*: *trace-def cmod-def adjoint-def scalar-prod-def*)
also have … = (∑ *i* ∈ {*0* ..< *dim-row A*}. ∑ *j* ∈ {*0* ..< *dim-col A*}. (*norm*(*A* $$ (*i*,*j*)))^2)
proof (*simp add*: *scalar-prod-def cmod-def*)
have *cnjmul*: ∀ *i ia*. *A* $$ (*i*, *ia*) ∗ *cnj* (*A* $$ (*i*, *ia*)) =
((*complex-of-real* (*Re* (*A* $$ (*i*, *ia*))))^2 + (*complex-of-real* (*Im* (*A*

```
$$ (i, ia))))^2)
        by (simp add: complex-mult-cnj)
      then have ∀ i. (∑ ia = 0..<dim-col A. A $$ (i, ia) ∗ cnj (A $$ (i, ia))) =
                      (∑ ia = 0..<dim-col A.  ((complex-of-real (Re (A $$ (i, ia))))^2
+ (complex-of-real (Im (A $$ (i, ia))))^2))
        by auto
      then show(∑ i = 0..<dim-row A. ∑ ia = 0..<dim-col A. A $$ (i, ia) ∗ cnj
(A $$ (i, ia))) =
        (∑ x = 0..<dim-row A. ∑ xa = 0..<dim-col A. (complex-of-real (Re (A $$
(x, xa))))^2) +
        (∑ x = 0..<dim-row A. ∑ xa = 0..<dim-col A. (complex-of-real (Im (A $$
(x, xa))))^2)
        by auto
    qed
  finally show ?thesis .
qed

lemma trace-adjoint-element-ineq:
  fixes A :: complex mat
  assumes rindex: i ∈ {0 ..< dim-row A}
     and  cindex: j ∈ {0 ..< dim-col A}
  shows (norm(A $$ (i,j)))^2 ≤ trace (A ∗ adjoint A)
proof (simp add: trace-adjoint-eq-u less-eq-complex-def)
   have ineqi: (cmod (A $$ (i, j)))^2 ≤ (∑ xa = 0..<dim-col A. (cmod (A $$ (i,
xa)))^2)
    using cindex member-le-sum[of j  {0 ..< dim-col A} λ x. (cmod (A $$ (i, x)))^2]
by auto
  also have ineqj: . . .  ≤ (∑ x = 0..<dim-row A. ∑ xa = 0..<dim-col A. (cmod
(A $$ (x, xa)))^2)
    using rindex member-le-sum[of i  {0 ..< dim-row A} λ x. ∑ xa = 0..<dim-col
A. (cmod (A $$ (x, xa)))^2]
    by (simp add: sum-nonneg)
  then show (cmod (A $$ (i, j)))^2 ≤ (∑ x = 0..<dim-row A. ∑ xa = 0..<dim-col
A. (cmod (A $$ (x, xa)))^2)
  using ineqi by linarith
 qed

lemma positive-is-normal:
  fixes A :: complex mat
  assumes pos: positive A
  shows A ∗ adjoint A = adjoint A ∗ A
proof −
  have hA: hermitian A using positive-is-hermitian pos by auto
  then show ?thesis by (simp add: hA hermitian-is-normal)
qed

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 −
  have AB: A ∗ B = mat n n (λ(i,j). (if (i = j) then (A$$(i, i)) ∗ (B$$(i, i))
else 0))
    using diag-mat-mult-diag-mat[of A n B] dimA dimB dA dB by auto
  then have dAB: ∀ i<n. ∀ j<n. i ≠ j ⟶ (A∗B) $$ (i,j) = 0
  proof −
    {
      fix i j assume i: i < n and j: j < n and ij: i ≠ j
      have (A∗B) $$ (i,j) = 0 using AB i j ij by auto
    }
    then show ?thesis by auto
  qed
  then show ?thesis using diagonal-mat-def dAB dimA dimB
   by (metis carrier-matD(1) carrier-matD(2) index-mult-mat(2) index-mult-mat(3))
qed

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$$(i, i)
proof −
  have AB: A ∗ B = mat n n (λ(i,j). if i = j then (A$$(i, i)) ∗ (B$$(i, i)) else 0)
    using diag-mat-mult-diag-mat[of A n B] dimA dimB dA dB by auto
  then show ?thesis
    using AB by auto
qed

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$$(i, i) ≥ 0
    shows trace (B∗B) ≤ (trace B)^2
proof −
  have tB:  trace B = (∑  i ∈ {0 ..<n}. B $$ (i,i)) using assms trace-def[of B]
carrier-mat-def by auto
  then have tBtB: (trace B)^2 = (∑  i ∈ {0 ..<n}.∑  j ∈ {0 ..<n}. B $$ (i,i)∗B
$$ (j,j))
  proof −
    show ?thesis
      by (metis (no-types) semiring-normalization-rules(29) sum-product tB)
  qed
 have BB: ⋀i. i < n ⟹ (B∗B) $$ (i,i) = (B$$(i, i))^2 using diag-mat-mul-diag-ele[of
B n B] dimB dB
       by (metis numeral-1-eq-Suc-0 power-Suc0-right power-add-numeral semir-
ing-norm(2))
   have tBB:  trace (B∗B) = (∑  i ∈ {0 ..<n}. (B∗B) $$ (i,i)) using assms
```

```
trace-def[of B∗B] carrier-mat-def by auto
  also have ... = (∑ i ∈ {0 ..<n}. (B$$(i, i))^2) using BB by auto
  finally have BBt: trace (B ∗ B) = (∑ i = 0..<n. (B $$ (i, i))^2) by auto
  have lesseq: ∀ i ∈ {0 ..<n}. (B $$ (i, i))^2 ≤ (∑ j ∈ {0 ..<n}. B $$ (i,i)∗B $$
(j,j))
  proof −
    {
      fix i assume i: i < n
      have (∑ j = 0..<n. B $$ (i, i) ∗ B $$ (j, j)) = (B $$ (i, i))^2 + sum (λ j.
(B $$ (i, i) ∗ B $$ (j, j))) ({0 ..<n} − {i})
       by (metis (no-types, lifting) BB atLeastLessThan-iff dB diag-mat-mul-diag-ele
dimB finite-atLeastLessThan i not-le not-less-zero sum.remove)
      moreover have (sum (λ j. (B $$ (i, i) ∗ B $$ (j, j))) ({0 ..<n} − {i})) ≥ 0
      proof (cases {0..<n} − {i} ≠ {})
        case True
        then show ?thesis using pB i sum-nonneg[of {0..<n} − {i} λ j. (B $$ (i,
i) ∗ B $$ (j, j))] by auto
       next
         case False
         have (∑ j∈{0..<n} − {i}. B $$ (i, i) ∗ B $$ (j, j)) = 0 using False by
fastforce
       then show ?thesis by auto
     qed
     ultimately have (∑ j = 0..<n. B $$ (i, i) ∗ B $$ (j, j)) ≥ (B $$ (i, i))^2 by
auto
   }
   then show ?thesis by auto
 qed
  from tBtB BBt lesseq have trace (B∗B) ≤ (trace B)^2
    using sum-mono[of {0..<n} λ i. (B $$ (i, i))^2 λ i. (∑ j = 0..<n. B $$ (i, i)
∗ B $$ (j, j))]
    by (metis (no-types, lifting))
  then show ?thesis by auto
qed


lemma trace-positive-eq:
  fixes A :: complex mat
  assumes pos: positive A
  shows trace (A ∗ adjoint A) ≤ (trace A)^2
proof −
  from assms have normal: A ∗ adjoint A = adjoint A ∗ A by (rule posi-
tive-is-normal)
  moreover
  from assms positive-dim-eq obtain n where cA: A ∈ carrier-mat n n by auto
  moreover
  from assms complex-mat-char-poly-factorizable cA obtain es where charpo:
char-poly A = (∏ a ← es. [:− a, 1:]) ∧ length es = n by auto
  moreover
  obtain B P Q where B: unitary-schur-decomposition A es = (B,P,Q) by (cases
```

unitary-schur-decomposition A es, *auto*)
ultimately have
smw: *similar-mat-wit A B P* (*adjoint P*)
and *ut*: *diagonal-mat B*
and *uP*: *unitary P*
and *dB*: *diag-mat B* = *es*
and *QaP*: *Q* = *adjoint P*
using *normal-complex-mat-has-spectral-decomposition*[*of A n es B P Q*] *unitary-schur-decomposition* **by** *auto*
from *smw cA QaP uP* **have** *cB*: *B* ∈ *carrier-mat n n* **and** *cP*: *P* ∈ *carrier-mat n n* **and** *cQ*: *Q* ∈ *carrier-mat n n*
unfolding *similar-mat-wit-def Let-def unitary-def* **by** *auto*
then have *caP*: *adjoint P* ∈ *carrier-mat n n* **using** *adjoint-dim*[*of P n*] **by** *auto*
from *smw QaP cA* **have** *A*: *A* = *P* ∗ *B* ∗ *adjoint P* **and** *traceA*: *trace A* = *trace* (*P* ∗ *B* ∗ *Q*) **and** *PB*: *P* ∗ *Q* = 1_m *n* ∧ *Q* ∗ *P* = 1_m *n*
unfolding *similar-mat-wit-def* **by** *auto*
have *traceAB*: *trace* (*P* ∗ *B* ∗ *Q*) = *trace* ((*Q*∗*P*)∗*B*)
using *cQ cP cB* **by** (*mat-assoc n*)
also have *traceelim*: ... = *trace B* **using** *traceAB PB cA cB cP cQ left-mult-one-mat*[*of P*∗*Q n n*]
using *similar-mat-wit-sym* **by** *auto*
finally have *traceAB*: *trace A* = *trace B* **using** *traceA* **by** *auto*
from *A cB cP* **have** *aAa*: *adjoint A* = *adjoint*((*P* ∗ *B*) ∗ *adjoint P*) **by** *auto*
have *aA*: *adjoint A* = *P* ∗ *adjoint B* ∗ *adjoint P*
unfolding *aAa* **using** *cP cB* **by** (*mat-assoc n*)
have *hA*: *hermitian A* **using** *pos positive-is-hermitian* **by** *auto*
then have *AaA*: *A* = *adjoint A* **using** *hA hermitian-def*[*of A*] **by** *auto*
then have *PBaP*: *P* ∗ *B* ∗ *adjoint P* = *P* ∗ *adjoint B* ∗ *adjoint P* **using** *A aA* **by** *auto*
then have *BaB*: *B* = *adjoint B* **using** *unitary-elim*[*of B n adjoint B P*] *uP cP cB adjoint-dim*[*of B n*] **by** *auto*
have *aPP*: *adjoint P* ∗ *P* = 1_m *n* **using** *uP PB QaP* **by** *blast*
have *A* ∗ *A* = *P* ∗ *B* ∗ (*adjoint P* ∗ *P*) ∗ *B* ∗ *adjoint P*
unfolding *A* **using** *cP cB* **by** (*mat-assoc n*)
also have ... = *P* ∗ *B* ∗ *B* ∗ *adjoint P*
unfolding *aPP* **using** *cP cB* **by** (*mat-assoc n*)
finally have *AA*: *A* ∗ *A* = *P* ∗ *B* ∗ *B* ∗ *adjoint P* **by** *auto*
then have *tAA*: *trace* (*A*∗*A*) = *trace* (*P* ∗ *B* ∗ *B* ∗ *adjoint P*) **by** *auto*
also have *tBB*: ... = *trace* (*adjoint P* ∗ *P* ∗ *B* ∗ *B*) **using** *cP cB* **by** (*mat-assoc n*)
also have ... = *trace* (*B* ∗ *B*) **using** *uP unitary-def*[*of P*] *inverts-mat-def*[*of P adjoint P*]
using *PB QaP cB* **by** *auto*
finally have *traceAABB*: *trace* (*A* ∗ *A*) = *trace* (*B* ∗ *B*) **by** *auto*
have *BP*: ⋀*i*. *i* < *n* ⟹ *B*$$(*i*, *i*) ≥ *0*
proof −
{
fix *i* **assume** *i*: *i* < *n*
then have *B*$$(*i*, *i*) ≥ *0* **using** *positive-eigenvalue-positive*[*of A n es B P Q*

i] *cA pos charpo B* **by** *auto*
then show $B\$\$(i, i) \geq 0$ **by** *auto*
}
qed
have *Brel*: *trace* $(B{*}B) \leq (trace\ B)^2$ **using** *trace-square-less-square-trace*[*of B n*] *cB ut BP* **by** *auto*
from *AaA traceAABB traceAB Brel* **have** *trace* $(A{*}adjoint\ A) \leq (trace\ A)^2$ **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *lowner-le-transitive*:
fixes *m n* :: *nat*
assumes *re*: $n \geq m$
shows *positive* $(f\ n - f\ m)$
proof −
from *re* **show** *positive* $(f\ n - f\ m)$
proof (*induct n*)
case *0*
then show *?case* **using** *positive-zero*
by (*metis dim le-0-eq minus-r-inv-mat*)
next
case (*Suc n*)
then show *?case*
proof (*cases Suc n* = *m*)
case *True*
then show *?thesis* **using** *positive-zero*
by (*metis dim minus-r-inv-mat*)
next
case *False*
then show *?thesis*
proof −
from *False Suc* **have** *nm*: $n \geq m$ **by** *linarith*
from *Suc nm* **have** *pnm*: *positive* $(f\ n - f\ m)$ **by** *auto*
from *inc* **have** *positive* $(f\ (Suc\ n) - f\ n)$ **unfolding** *lowner-le-def* **by** *auto*
then have *pf*: *positive* $((f\ (Suc\ n) - f\ n) + (f\ n - f\ m))$ **using** *positive-add dim pnm*
by (*meson minus-carrier-mat*)
have $(f\ (Suc\ n) - f\ n) + (f\ n - f\ m) = f\ (Suc\ n) + ((- f\ n) + f\ n) + (- f\ m)$
using *local.dim* **by** (*mat-assoc dim*, *auto*)
also have $\ldots = f\ (Suc\ n) + 0_m\ dim\ dim + (- f\ m)$
using *local.dim* **by** (*subst uminus-l-inv-mat*[**where** *nc*=*dim* **and** *nr*=*dim*], *auto*)
also have $\ldots = f\ (Suc\ n) - f\ m$
using *local.dim* **by** (*mat-assoc dim*, *auto*)
finally have *re*: $f\ (Suc\ n) - f\ n + (f\ n - f\ m) = f\ (Suc\ n) - f\ m$ **.**
from *pf re* **have** *positive* $(f\ (Suc\ n) - f\ m)$ **by** *auto*
then show *?thesis* **by** *auto*

qed
qed
qed
qed

The sequence of matrices converges pointwise.

lemma *inc-partial-density-operator-converge*:
assumes *i*: $i \in \{0 ..< dim\}$ **and** *j*: $j \in \{0 ..< dim\}$
shows *convergent* ($\lambda n.\ f\ n$ $$ (i, j))
proof−
have *tracefn*: *trace* ($f\ n$) $\geq 0 \wedge$ *trace* ($f\ n$) ≤ 1 **for** *n*
proof −
from *pdo* **show** *?thesis*
unfolding *partial-density-operator-def* **using** *positive-trace*[*of f n*]
using *dim* **by** *blast*
qed
from *tracefn* **have** *normf*: *norm*(*trace* ($f\ n$)) $\leq$ *norm*(*trace* (*f* (*Suc n*))) $\wedge$ *norm*(*trace* ($f\ n$)) ≤ 1 **for** *n*
proof −
have *trless*: *trace* ($f\ n$) $\leq$ *trace* (*f* (*Suc n*))
using *pdo inc dim positive-trace*[*of f*(*Suc n*) − *f n*] *trace-minus-linear*[*of f* (*Suc n*) *dim f n*]
unfolding *partial-density-operator-def lowner-le-def*
using *Complex-Matrix.positive-def* **by** *force*
moreover from *trless tracefn* **have** *norm*(*trace* ($f\ n$)) $\leq$ *norm*(*trace* (*f* (*Suc n*))) **unfolding** *cmod-def*
by (*simp add*: *less-eq-complex-def less-complex-def*)
moreover from *trless tracefn* **have** *norm*(*trace* ($f\ n$)) ≤ 1 **using** *pdo partial-density-operator-def cmod-def*
by (*simp add*: *less-eq-complex-def less-complex-def*)
ultimately show *?thesis* **by** *auto*
qed
then have *inctrace*: *incseq* (λ *n*. *norm*(*trace* ($f\ n$))) **by** (*simp add*: *incseq-SucI*)
then have *tr-sup*: (λ *n*. *norm*(*trace* ($f\ n$))) $\longrightarrow$ (*SUP i*. *norm* (*trace* ($f\ i$)))
using *LIMSEQ-incseq-SUP*[*of* λ *n*. *norm*(*trace* ($f\ n$))] *pdo partial-density-operator-def normf* **by** (*meson bdd-aboveI2*)
then have *tr-cauchy*: *Cauchy* (λ *n*. *norm*(*trace* ($f\ n$))) **using** *Cauchy-convergent-iff convergent-def* **by** *blast*
then have *tr-cauchy-def*: $\forall e>0.\ \exists M.\ \forall m\geq M.\ \forall n\geq M.$ *dist*(*norm*(*trace* ($f\ n$))) (*norm*(*trace* ($f\ m$))) $< e$ **unfolding** *Cauchy-def* **by** *blast*
moreover have $\forall$ *m n*. *dist*(*norm*(*trace* ($f\ m$))) (*norm*(*trace* ($f\ n$))) = *norm*(*trace* ($f\ m$) − *trace* ($f\ n$))
using *tracefn cmod-eq-Re dist-real-def* **by** (*auto simp*: *less-eq-complex-def less-complex-def*)
ultimately have *norm-trace*: $\forall e>0.\exists M.\ \forall m\geq M.\ \forall n\geq M.$ *norm*((*trace* ($f\ n$)) − (*trace* ($f\ m$))) $< e$ **by** *auto*

have *eq-minus*: $\forall$ *m n*. *trace* ($f\ m$) − *trace* ($f\ n$) = *trace* ($f\ m - f\ n$) **using** *trace-minus-linear dim* **by** *metis*
from *eq-minus norm-trace* **have** *norm-trace-cauchy*: $\forall e>0.\exists M.\ \forall m\geq M.\ \forall n\geq M.$

norm((*trace* (*f n* − *f m*))) < *e* **by** *auto*
then have *norm-trace-cauchy-iff*: ∀ *e*>*0*.∃ *M*. ∀ *m*≥*M*. ∀ *n*≥*m*. *norm*((*trace* (*f n* − *f m*))) < *e*
by (*meson order-trans-rules*(*23*))
then have *norm-square*: ∀ *e*>*0*.∃ *M*. ∀ *m*≥*M*. ∀ *n*≥*m*. $(norm((trace\ (f\ n - f\ m))))^2 < e^2$
by (*metis abs-of-nonneg norm-ge-zero order-less-le real-sqrt-abs real-sqrt-less-iff*)

have *tr-re*: ∀ *m*. ∀ *n* ≥ *m*. *trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*)) ≤ $((trace\ (f\ n - f\ m)))^2$
using *trace-positive-eq lowner-le-transitive* **by** *auto*
have *tr-re-g*: ∀ *m*. ∀ *n* ≥ *m*. *trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*)) ≥ *0*
using *lowner-le-transitive positive-trace trace-adjoint-positive* **by** *auto*
have *norm-trace-fmn*: *norm*(*trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*))) ≤ $(norm(trace\ (f\ n - f\ m)))^2$ **if** *nm*: *n* ≥ *m* **for** *m n*
proof −
have *mnA*: *trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*)) ≤ $(trace\ (f\ n - f\ m))^2$ **using** *tr-re nm* **by** *auto*
have *mnB*: *trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*)) ≥ *0* **using** *tr-re-g nm* **by** *auto*
from *mnA mnB* **show** *?thesis*
by (*smt* (*verit*) *cmod-eq-Re less-eq-complex-def norm-power zero-complex.sel*(*1*) *zero-complex.sel*(*2*))
qed
then have *cauchy-adj*: ∃ *M*. ∀ *m*≥*M*. ∀ *n*≥*m*. *norm*(*trace* ((*f n*− *f m*) ∗ *adjoint* (*f n* − *f m*))) < e^2 **if** *e*: *e* > *0* **for** *e*
proof −
have ∃ *M*. ∀ *m*≥*M*. ∀ *n*≥*m*. $(cmod\ (trace\ (f\ n - f\ m)))^2 < e^2$ **using** *norm-square e* **by** *auto*
then obtain *M* **where** ∀ *m*≥*M*. ∀ *n*≥*m*. $(cmod\ (trace\ (f\ n - f\ m)))^2 < e^2$ **by** *auto*
then have ∀ *m*≥*M*. ∀ *n*≥*m*. *norm*(*trace* ((*f n*− *f m*) ∗ *adjoint* (*f n* − *f m*))) < e^2 **using** *norm-trace-fmn* **by** *fastforce*
then show *?thesis* **by** *auto*
qed

have *norm-minus*: ∀ *m*. ∀ *n* ≥ *m*. $(norm\ ((f\ n - f\ m)\ \$\$\ (i, j)))^2$ ≤ *trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*))
using *trace-adjoint-element-ineq i j*
by (*smt* (*verit*) *adjoint-dim-row carrier-matD*(*1*) *index-minus-mat*(*2*) *index-mult-mat*(*2*) *lowner-le-transitive matrix-seq-axioms matrix-seq-def positive-is-normal*)
then have *norm-minus-le*: $(norm\ ((f\ n - f\ m)\ \$\$\ (i, j)))^2$ ≤ *norm* (*trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*))) **if** *nm*: *n* ≥ *m* **for** *n m*
proof −
have $(norm\ ((f\ n - f\ m)\ \$\$\ (i, j)))^2$ ≤ (*trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*))) **using** *norm-minus nm* **by** *auto*
also have . . . = *norm* (*trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*))) **using** *tr-re-g nm*
by (*smt* (*verit*) *Re-complex-of-real less-eq-complex-def matrix-seq.trace-adjoint-eq-u*

matrix-seq-axioms mult-cancel-left2 norm-one norm-scaleR of-real-def of-real-hom.hom-zero)
finally show *?thesis* **by** (*auto simp*: *less-eq-complex-def less-complex-def*)
qed

from *norm-minus-le cauchy-adj* **have** *cauchy-ij*: $\exists M. \forall m \geq M. \forall n \geq m.$ (*norm* ((*f n* − *f m*) $$ (*i*, *j*)))^2 < e^2 **if** *e*: *e* > *0* **for** *e*
proof −
have $\exists M. \forall m \geq M. \forall n \geq m.$ *norm*(*trace* ((*f n*− *f m*) ∗ *adjoint* (*f n* − *f m*))) < e^2 **using** *cauchy-adj e* **by** *auto*
then obtain *M* **where** $\forall m \geq M. \forall n \geq m.$ *norm*(*trace* ((*f n* − *f m*) ∗ *adjoint* (*f n* − *f m*))) < e^2 **by** *auto*
then have $\forall m \geq M. \forall n \geq m.$ (*norm* ((*f n* − *f m*) $$ (*i*, *j*)))^2 < e^2 **using** *norm-minus-le* **by** *fastforce*
then show *?thesis* **by** *auto*
qed
then have *cauchy-ij-norm*: $\exists M. \forall m \geq M. \forall n \geq m.$ (*norm* ((*f n* − *f m*) $$ (*i*, *j*))) < *e* **if** *e*: *e* > *0* **for** *e*
proof −
have $\exists M. \forall m \geq M. \forall n \geq m.$ (*norm* ((*f n* − *f m*) $$ (*i*, *j*)))^2 < e^2 **using** *cauchy-ij e* **by** *auto*
then obtain *M* **where** *mn*: $\forall m \geq M. \forall n \geq m.$ (*norm* ((*f n* − *f m*) $$ (*i*, *j*)))^2 < e^2 **by** *auto*
have (*norm* ((*f n* − *f m*) $$ (*i*, *j*))) < *e* **if** *m*: *m* ≥ *M* **and** *n*: *n* ≥ *m* **for** *m n* :: *nat*
proof −
from *m n mn* **have** (*norm* ((*f n*− *f m*) $$ (*i*, *j*)))^2 < e^2 **by** *auto*
then show *?thesis*
using *e power-less-imp-less-base* **by** *fastforce*
qed
then show *?thesis* **by** *auto*
qed

have *cauchy-final*: $\exists M. \forall m \geq M. \forall n \geq M.$ *norm* ((*f m*) $$ (*i*, *j*) − (*f n*) $$ (*i*, *j*)) < *e* **if** *e*: *e* > *0* **for** *e*
proof −
obtain *M* **where** *mnm*: $\forall m \geq M. \forall n \geq m.$ *norm* ((*f n* − *f m*) $$ (*i*, *j*)) < *e* **using** *cauchy-ij-norm e* **by** *auto*
have *norm* ((*f m*) $$ (*i*, *j*) − (*f n*) $$ (*i*, *j*)) < *e* **if** *m*: *m* ≥ *M* **and** *n*: *n* ≥ *M* **for** *m n*
proof (*cases n* ≥ *m*)
case *True*
then show *?thesis*
proof −
from *mnm m True* **have** *norm* ((*f n*) $$ (*i*, *j*) − (*f m*) $$ (*i*, *j*)) < *e*
by (*metis atLeastLessThan-iff carrier-matD*(*1*) *carrier-matD*(*2*) *dim i index-minus-mat*(*1*) *j*)
then have *norm* ((*f m*) $$ (*i*, *j*) − (*f n*) $$ (*i*, *j*)) < *e* **by** (*simp add*: *norm-minus-commute*)
then show *?thesis* **by** *auto*

qed
next
case *False*
then show *?thesis*
proof −
from *False n mnm* **have** *norm*: *norm* ((*f m* − *f n*) $$ (*i*, *j*)) < *e* **by** *auto*
have *minus*: (*f m* − *f n*) $$ (*i*, *j*) = *f m* $$ (*i*, *j*) −*f n* $$ (*i*, *j*)
by (*metis atLeastLessThan-iff carrier-matD*(*1*) *carrier-matD*(*2*) *dim i index-minus-mat*(*1*) *j*)
also have ... = − (*f n* − *f m*) $$ (*i*, *j*) **using** *dim*
by (*metis atLeastLessThan-iff carrier-matD*(*1*) *carrier-matD*(*2*) *i index-minus-mat*(*1*) *j minus-diff-eq*)
finally have *fmn*: (*f m* − *f n*) $$ (*i*, *j*) = − (*f n* − *f m*) $$ (*i*, *j*) **by** *auto*
then have *norm* ((− (*f n* − *f m*)) $$ (*i*, *j*)) < *e* **using** *norm*
by (*metis* (*no-types*, *lifting*) *atLeastLessThan-iff carrier-matD*(*1*) *carrier-matD*(*2*) *i*
index-minus-mat(*2*) *index-minus-mat*(*3*) *index-uminus-mat*(*1*) *j matrix-seq-axioms matrix-seq-def*)
then have *norm* (((*f n* − *f m*)) $$ (*i*, *j*)) < *e* **using** *fmn norm* **by** *auto*
then have *norm* (*f n* $$ (*i*, *j*) − *f m* $$ (*i*, *j*)) < *e*
by (*metis minus norm norm-minus-commute*)
then have *norm* (*f m* $$ (*i*, *j*) − *f n* $$ (*i*, *j*)) < *e* **by** (*simp add*: *norm-minus-commute*)
then show *?thesis* **by** *auto*
qed
qed
then show *?thesis* **by** *auto*
qed

from *cauchy-final* **have** *Cauchy* (λ *n*. *f n* $$ (*i*, *j*)) **by** (*simp add*: *Cauchy-def dist-norm*)
then show *?thesis* **by** (*simp add*: *Cauchy-convergent-iff*)
qed

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 −
from *posX* **obtain** *k* **where** *posk*: ∀ *n*≥*k*. *positive* (*X n*) **by** *auto*

let *?Y* = λn. *X* (*n* + *k*)
have *posY*: ∀ *n*. *positive* (*?Y n*) **using** *posk* **by** *auto*

from *limX* **have** *dimXA*: ∀ *n*. *X* (*n* + *k*) ∈ *carrier-mat m m* ∧ *A* ∈ *carrier-mat m m*
unfolding *limit-mat-def* **by** *auto*

have (λn. *X* (*n* + *k*) $$ (*i*, *j*)) ⟶ *A* $$ (*i*, *j*) **if** *i*: *i* < *m* **and** *j*: *j* < *m* **for** *i j*
proof −
have (λn. *X n* $$ (*i*, *j*)) ⟶ *A* $$ (*i*, *j*) **using** *limX limit-mat-def i j* **by** *auto*
then have *limseqX*: ∀ *r*>*0*. ∃ *no*. ∀ *n*≥*no*. *dist* (*X n* $$ (*i*, *j*)) (*A* $$ (*i*, *j*)) < *r* **unfolding** *LIMSEQ-def* **by** *auto*
then have ∃ *no*. ∀ *n*≥*no*. *dist* (*X* (*n* + *k*) $$ (*i*, *j*)) (*A* $$ (*i*, *j*)) < *r* **if** *r*: *r* > *0* **for** *r*
proof −
obtain *no* **where** ∀ *n*≥*no*. *dist* (*X n* $$ (*i*, *j*)) (*A* $$ (*i*, *j*)) < *r* **using** *limseqX r* **by** *auto*
then have ∀ *n*≥*no*. *dist* (*X* (*n* + *k*) $$ (*i*, *j*)) (*A* $$ (*i*, *j*)) < *r* **by** *auto*
then show *?thesis* **by** *auto*
qed
then show *?thesis* **unfolding** *LIMSEQ-def* **by** *auto*
qed
then have *limXA*: *limit-mat* (λn. *X* (*n* + *k*)) *A m* **unfolding** *limit-mat-def* **using** *dimXA* **by** *auto*

from *posY limXA* **have** *positive A* **using** *pos-mat-lim-is-pos*[*of ?Y A m*] **by** *auto*
then show *?thesis* **by** *auto*
qed

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 −
have *dimXAB*: ∀ *n*. *X n* − *B* ∈ *carrier-mat m m* ∧ *A* − *B* ∈ *carrier-mat m m* **using** *index-minus-mat dimB* **by** *auto*
have (λn. (*X n* − *B*) $$ (*i*, *j*)) ⟶ (*A* − *B*) $$ (*i*, *j*) **if** *i*: *i* < *m* **and** *j*: *j* < *m* **for** *i j*
proof −
from *limX i j* **have** (λn. (*X n*) $$ (*i*, *j*)) ⟶ (*A*) $$ (*i*, *j*) **unfolding** *limit-mat-def* **by** *auto*
then have *X*: ∀ *r*>*0*. ∃ *no*. ∀ *n*≥*no*. *dist* (*X n* $$ (*i*, *j*)) (*A* $$ (*i*, *j*)) < *r* **unfolding** *LIMSEQ-def* **by** *auto*
then have *XB*: ∃ *no*. ∀ *n*≥*no*. *dist* ((*X n* − *B*) $$ (*i*, *j*)) ((*A* − *B*) $$ (*i*, *j*)) < *r* **if** *r*: *r* > *0* **for** *r*
proof −

obtain *no* **where** $\forall n \geq no.$ *dist* $(X\ n\ \$\$\ (i, j))\ (A\ \$\$\ (i, j)) < r$ **using** *r X* **by** *auto*

then have *dist*: $\forall n \geq no.$ *norm* $(X\ n\ \$\$\ (i, j) - A\ \$\$\ (i, j)) < r$ **unfolding** *dist-norm* **by** *auto*

then have *norm* $((X\ n - B)\ \$\$\ (i, j) - (A - B)\ \$\$\ (i, j)) < r$ **if** n: $n \geq no$ **for** n

proof −

have $(X\ n - B)\ \$\$\ (i, j) - (A - B)\ \$\$\ (i, j) = (X\ n)\ \$\$\ (i, j) - A\ \$\$\ (i, j)$

using *dimB i j* **by** *auto*

then have *norm* $((X\ n - B)\ \$\$\ (i, j) - (A - B)\ \$\$\ (i, j)) =$ *norm* $((X\ n)\ \$\$\ (i, j) - A\ \$\$\ (i, j))$ **by** *auto*

then show *?thesis* **using** *dist n* **by** *auto*

qed

then show *?thesis* **using** *dist-norm* **by** *metis*

qed

then show *?thesis* **unfolding** *LIMSEQ-def* **by** *auto*

qed

then show *?thesis*

unfolding *limit-mat-def mat-seq-minus-def* **using** *dimXAB* **by** *auto*

qed

lemma *mat-minus-limit*:

fixes X :: *nat* ⇒ *complex mat* **and** A :: *complex mat* **and** m :: *nat* **and** B :: *complex mat*

assumes *dimA*: $A \in$ *carrier-mat m m* **and** *limX*: *limit-mat X A m*

shows *limit-mat* (*minus-mat-seq B X*) $(B - A)$ *m*

proof−

have *dimX* : $\forall n.\ X\ n \in$ *carrier-mat m m* **using** *limX* **unfolding** *limit-mat-def* **by** *auto*

then have *dimXAB*: $\forall n.\ B - X\ n \in$ *carrier-mat m m* $\wedge\ B - A \in$ *carrier-mat m m* **using** *index-minus-mat dimA*

by (*simp add*: *minus-carrier-mat*)

have $(\lambda n.\ (B - X\ n)\ \$\$\ (i, j)) \longrightarrow (B - A)\ \$\$\ (i, j)$ **if** i: $i < m$ **and** j: $j < m$ **for** *i j*

proof −

from *limX i j* **have** $(\lambda n.\ (X\ n)\ \$\$\ (i, j)) \longrightarrow (A)\ \$\$\ (i, j)$ **unfolding** *limit-mat-def* **by** *auto*

then have X: $\forall r>0.\ \exists no.\ \forall n \geq no.$ *dist* $(X\ n\ \$\$\ (i, j))\ (A\ \$\$\ (i, j)) < r$ **unfolding** *LIMSEQ-def* **by** *auto*

then have *XB*: $\exists no.\ \forall n \geq no.$ *dist* $((B - X\ n)\ \$\$\ (i, j))\ ((B - A)\ \$\$\ (i, j)) < r$ **if** r: $r > 0$ **for** r

proof −

obtain *no* **where** $\forall n \geq no.$ *dist* $(X\ n\ \$\$\ (i, j))\ (A\ \$\$\ (i, j)) < r$ **using** *r X* **by** *auto*

then have *dist*: $\forall n \geq no.$ *norm* $(X\ n\ \$\$\ (i, j) - A\ \$\$\ (i, j)) < r$ **unfolding** *dist-norm* **by** *auto*

then have *norm* $((B - X\ n)\ \$\$\ (i, j) - (B - A)\ \$\$\ (i, j)) < r$ **if** n: $n \geq no$

for *n*
proof −
have $(B - X\ n)\ \$\$\ (i, j) - (B - A)\ \$\$\ (i, j) = - ((X\ n)\ \$\$\ (i, j) - A\ \$\$\ (i, j))$
using *dimA i j*
by (*smt* (*verit*) *cancel-ab-semigroup-add-class.diff-right-commute cancel-comm-monoid-add-class.diff-cancel carrier-matD*(*1*) *carrier-matD*(*2*) *diff-add-cancel dimX index-minus-mat*(*1*) *minus-diff-eq*)
then have *norm* $((B - X\ n)\ \$\$\ (i, j) - (B - A)\ \$\$\ (i, j))$ = *norm* $((X\ n)\ \$\$\ (i, j) - A\ \$\$\ (i, j))$
by (*metis norm-minus-cancel*)
then show *?thesis* **using** *dist n* **by** *auto*
qed
then show *?thesis* **using** *dist-norm* **by** *metis*
qed
then show *?thesis* **unfolding** *LIMSEQ-def* **by** *auto*
qed
then have *limit-mat* (*minus-mat-seq B X*) $(B - A)$ *m*
unfolding *limit-mat-def minus-mat-seq-def* **using** *dimXAB* **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *lowner-lub-form*:
lowner-is-lub (*mat dim dim* (λ (i, j). (*lim* (λ *n*. (*f n*) \$\$ (i, j)))))
proof −
from *inc-partial-density-operator-converge*
have *conf*: $\forall$ $i \in \{0 ..<dim\}$. $\forall$ $j \in \{0 ..<dim\}$. *convergent* (λ *n*. *f n* \$\$ (i, j)) **by** *auto*
let *?A* = *mat dim dim* (λ (i, j). (*lim* (λ *n*. (*f n*) \$\$ (i, j))))
have *dim-A*: *?A* $\in$ *carrier-mat dim dim* **by** *auto*
have *lim-A*: (λn. *f n* \$\$ (i, j)) $\longrightarrow$ *mat dim dim* ($\lambda(i, j)$. *lim* (λn. *f n* \$\$ (i, j))) \$\$ (i, j)
if *i*: $i < dim$ **and** *j*: $j < dim$ **for** *i j*
proof −
from *i j* **have** *ij*: *mat dim dim* ($\lambda(i, j)$. *lim* (λn. *f n* \$\$ (i, j))) \$\$ (i, j) = *lim* (λn. *f n* \$\$ (i, j))
by (*metis case-prod-conv index-mat*(*1*))
have *convergent* (λn. *f n* \$\$ (i, j)) **using** *conf i j* **by** *auto*
then have (λn. *f n* \$\$ (i, j)) $\longrightarrow$ *lim* (λn. *f n* \$\$ (i, j)) **using** *convergent-LIMSEQ-iff* **by** *auto*
then show *?thesis* **using** *ij* **by** *auto*
qed

from *dim dim-A lim-A* **have** *lim-mat-A*: *limit-mat f ?A dim* **unfolding** *limit-mat-def* **by** *auto*

have *is-ub*: *f n* $\leq_L$ *?A* **for** *n*
proof −
have $\forall$ $m \geq n$. *positive* (*f m* − *f n*) **using** *lowner-le-transitive* **by** *auto*

then have *le*: $\forall\ m \geq n.\ f\ n \leq_L f\ m$ **unfolding** *lowner-le-def* **using** *dim*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
have *dimn*: $f\ n \in$ *carrier-mat dim dim* **using** *dim* **by** *auto*
then have *limAf*: *limit-mat* (*mat-seq-minus f* (*f n*)) (*?A* − *f n*) *dim* **using** *minus-mat-limit lim-mat-A* **by** *auto*

have $\forall\, m \geq n.$ *positive* (*f m* − *f n*) **using** *lowner-le-transitive* **by** *auto*
then have $\exists\, k.\ \forall\, m \geq k.$ *positive* (*f m* − *f n*) **by** *auto*
then have *posAf*: $\exists\ k.\ \forall\ m \geq k.$ *positive* ((*mat-seq-minus f* (*f n*)) *m*) **unfolding** *mat-seq-minus-def* **by** *auto*

from *limAf posAf* **have** *positive* (*?A* − *f n*) **using** *pos-mat-lim-is-pos-aux* **by** *auto*
then have $f\ n \leq_L$ *mat dim dim* ($\lambda(i,\ j).$ *lim* ($\lambda n.$ *f n* $$ (*i*, *j*))) **unfolding** *lowner-le-def* **using** *dim* **by** *auto*
then show *?thesis* **by** *auto*
qed

have *is-lub*: $?A \leq_L M'$ **if** *ub*: $\forall\, n.\ f\ n \leq_L M'$ **for** M'
proof −
have *dim-M*: $M' \in$ *carrier-mat dim dim* **using** *ub* **unfolding** *lowner-le-def* **using** *dim*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *carrier-mat-triv*)
from *ub* **have** *posAf*: $\forall\ n.$ *positive* (*minus-mat-seq* M' *f n*) **unfolding** *minus-mat-seq-def lowner-le-def* **by** *auto*
have *limAf*: *limit-mat* (*minus-mat-seq* M' *f*) (M' − *?A*) *dim*
using *mat-minus-limit dim-A lim-mat-A* **by** *auto*
from *posAf limAf* **have** *positive* (M' − *?A*) **using** *pos-mat-lim-is-pos-aux* **by** *auto*
then have $?A \leq_L M'$ **unfolding** *lowner-le-def* **using** *dim dim-A dim-M* **by** *auto*
then show *?thesis* **by** *auto*
qed

from *is-ub is-lub* **show** *?thesis* **unfolding** *lowner-is-lub-def* **by** *auto*
qed

Lowner partial order is a complete partial order.

lemma *lowner-lub-exists*: $\exists\, M.$ *lowner-is-lub M*
using *lowner-lub-form* **by** *auto*

lemma *lowner-lub-unique*: $\exists!M.$ *lowner-is-lub M*
proof (*rule HOL.ex-ex1I*)
show $\exists\, M.$ *lowner-is-lub M*
by (*rule lowner-lub-exists*)
next
fix *M N*
assume *M*: *lowner-is-lub M* **and** *N*: *lowner-is-lub N*
have *Md*: $M \in$ *carrier-mat dim dim* **using** *M* **by** (*rule lowner-is-lub-dim*)

have *Nd*: $N \in$ *carrier-mat dim dim* **using** *N* **by** (*rule lowner-is-lub-dim*)
have *MN*: $M \leq_L N$ **using** *M N* **by** (*simp add*: *lowner-is-lub-def*)
have *NM*: $N \leq_L M$ **using** *M N* **by** (*simp add*: *lowner-is-lub-def*)
show $M = N$ **using** *MN NM* **by** (*auto intro*: *lowner-le-antisym*[*OF Md Nd*])
qed

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

lemma *lowner-lub-prop*: *lowner-is-lub lowner-lub*
unfolding *lowner-lub-def*
apply (*rule HOL.theI'*)
by (*rule lowner-lub-unique*)

lemma *lowner-lub-is-limit*:
limit-mat f lowner-lub dim
proof −
define *A* **where** *A* = *lowner-lub*
then have *A* = (*THE M*. *lowner-is-lub M*) **using** *lowner-lub-def* **by** *auto*
then have *Af*: *A* = (*mat dim dim* (λ (*i*, *j*). (*lim* (λ *n*. (*f n*) $$ (*i*, *j*)))))
using *lowner-lub-form lowner-lub-unique* **by** *auto*
show *limit-mat f A dim* **unfolding** *Af limit-mat-def*
apply (*auto simp add*: *dim*)
proof −
fix *i j* **assume** *dims*: $i < dim$ $j < dim$
then have *convergent* (λ*n*. *f n* $$ (*i*, *j*)) **using** *inc-partial-density-operator-converge*
by *auto*
then show (λ*n*. *f n* $$ (*i*, *j*)) ⟶ *lim* (λ*n*. *f n* $$ (*i*, *j*)) **using** *convergent-LIMSEQ-iff* **by** *auto*
qed
qed

lemma *lowner-lub-trace*:
assumes ∀ *n*. *trace* (*f n*) ≤ *x*
shows *trace lowner-lub* ≤ *x*
proof −
have ∀ *n*. *trace* (*f n*) ≥ *0* **using** *positive-trace pdo* **unfolding** *partial-density-operator-def*
using *dim* **by** *blast*
then have *Re*: ∀ *n*. *Re* (*trace* (*f n*)) ≥ *0* ∧ *Im* (*trace* (*f n*)) = *0*
by (*auto simp*: *less-eq-complex-def less-complex-def*)
then have *lex*: ∀ *n*. *Re* (*trace* (*f n*)) ≤ *Re x* ∧ *Im x* = *0* **using** *assms*
by (*auto simp*: *less-eq-complex-def less-complex-def*)

have *limit-mat f lowner-lub dim* **using** *lowner-lub-is-limit* **by** *auto*
then have *conv*: (λ*n*. *trace* (*f n*)) ⟶ *trace lowner-lub* **using** *mat-trace-limit*
by *auto*
then have (λ*n*. *Re* (*trace* (*f n*))) ⟶ *Re* (*trace lowner-lub*)
by (*simp add*: *tendsto-Re*)
then have *Rell*: *Re* (*trace lowner-lub*) ≤ *Re x*

using *lex Lim-bounded*[*of* (*λn. Re* (*trace* (*f n*))) *Re* (*trace lowner-lub*) *0 Re x*]
by *simp*

from *conv* **have** (*λn. Im* (*trace* (*f n*))) ⟶ *Im* (*trace lowner-lub*)
by (*simp add*: *tendsto-Im*)
then have *Imll*: *Im* (*trace lowner-lub*) = *0* **using** *Re*
by (*simp add*: *Lim-bounded Lim-bounded2 dual-order.antisym*)

from *Rell Imll lex* **show** *?thesis* **by** (*simp add*: *less-eq-complex-def less-complex-def*)
qed

lemma *lowner-lub-is-positive*:
shows *positive lowner-lub*
using *lowner-lub-is-limit pos-mat-lim-is-pos pdo* **unfolding** *partial-density-operator-def*
by *auto*

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* (*λn. 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 (*induct n*)
case *0*
show *?case* **by** *auto*
next
case (*Suc n*)
then have *f n* ∈ *carrier-mat d d* **by** *auto*
then show *?case* **using** *Suc* **by** *auto*
qed

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 (*induct n*)
case *0*
show *?case* **by** *auto*
next
case (*Suc n*)

then show *?case* **unfolding** *matrix-sum.simps* **by** *auto*
qed

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 $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow (\bigwedge k.\ k < n \Longrightarrow g\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow (\bigwedge k.\ k < n \Longrightarrow h\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow$
$(\bigwedge k.\ k < n \Longrightarrow f\ k = g\ k + h\ k) \Longrightarrow matrix\text{-}sum\ d\ f\ n = matrix\text{-}sum\ d\ g\ n + matrix\text{-}sum\ d\ h\ n$
proof (*induct n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof −
have *gh*: *matrix-sum d g n* ∈ *carrier-mat d d* ∧ *matrix-sum d h n* ∈ *carrier-mat d d*
using *matrix-sum-dim Suc*(*3*, *4*) **by** (*simp add*: *matrix-sum-dim*)

have *nSuc*: *n* < *Suc n* **by** *auto*
have *sumf*: *matrix-sum d f n* = *matrix-sum d g n* + *matrix-sum d h n* **using** *Suc* **by** *auto*
have *matrix-sum d f* (*Suc n*) = *matrix-sum d g* (*Suc n*) + *matrix-sum d h* (*Suc n*)
unfolding *matrix-sum.simps Suc*(*5*)[*OF nSuc*] *sumf*
apply (*mat-assoc d*) **using** *gh Suc* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

lemma *matrix-sum-smult*:
fixes *f* :: *nat* ⇒ *'b*::*semiring-1 mat*
shows $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d) \Longrightarrow$
$matrix\text{-}sum\ d\ (\lambda\ k.\ c \cdot_m f\ k)\ n = c \cdot_m matrix\text{-}sum\ d\ f\ n$
proof (*induct n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case*
apply *auto*
using *add-smult-distrib-left-mat Suc matrix-sum-dim*
by (*metis lessI less-SucI*)
qed

lemma *matrix-sum-remove*:
fixes *f* :: *nat* ⇒ *'b*::*semiring-1 mat*

assumes *j*: $j < n$
and *df*: $(\bigwedge k.\ k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d)$
and *f'*: $(\bigwedge k.\ f'\ k = (if\ k = j\ then\ 0_m\ d\ d\ else\ f\ k))$
shows $matrix\text{-}sum\ d\ f\ n = f\ j + matrix\text{-}sum\ d\ f'\ n$
proof −
have *df'*: $\bigwedge k.\ k < n \Longrightarrow f'\ k \in carrier\text{-}mat\ d\ d$ **using** *f' df* **by** *auto*
have *dsf*: $k < n \Longrightarrow matrix\text{-}sum\ d\ f\ k \in carrier\text{-}mat\ d\ d$ **for** *k* **using** *matrix-sum-dim*[*OF df*] **by** *auto*
have *dsf'*: $k < n \Longrightarrow matrix\text{-}sum\ d\ f'\ k \in carrier\text{-}mat\ d\ d$ **for** *k* **using** *matrix-sum-dim*[*OF df'*] **by** *auto*
have *flj*: $\bigwedge k.\ k < j \Longrightarrow f'\ k = f\ k$ **using** *j f'* **by** *auto*
then have $matrix\text{-}sum\ d\ f\ j = matrix\text{-}sum\ d\ f'\ j$ **using** *matrix-sum-cong*[*of j f' f, OF flj*] *df df' j* **by** *auto*
then have *eqj*: $matrix\text{-}sum\ d\ f\ (Suc\ j) = f\ j + matrix\text{-}sum\ d\ f'\ (Suc\ j)$ **unfolding** *matrix-sum.simps*
by (*subst* (*1*) *f'*, *simp add*: *df dsf' j*)
have *lm*: $(j + 1) + l \le n \Longrightarrow matrix\text{-}sum\ d\ f\ ((j + 1) + l) = f\ j + matrix\text{-}sum\ d\ f'\ ((j + 1) + l)$ **for** *l*
proof (*induct l*)
case *0*
show *?case* **using** *j eqj* **by** *auto*
next
case (*Suc l*) **then have** *eq*: $matrix\text{-}sum\ d\ f\ ((j + 1) + l) = f\ j + matrix\text{-}sum\ d\ f'\ ((j + 1) + l)$ **by** *auto*
have *s*: $((j + 1) + Suc\ l) = Suc\ ((j + 1) + l)$ **by** *simp*
have *eqf'*: $f'\ (j + 1 + l) = f\ (j + 1 + l)$ **using** *f' Suc* **by** *auto*
have *dims*: $f\ (j + 1 + l) \in carrier\text{-}mat\ d\ d$ $f\ j \in carrier\text{-}mat\ d\ d$ $matrix\text{-}sum\ d\ f'\ (j + 1 + l) \in carrier\text{-}mat\ d\ d$ **using** *df df' dsf' Suc* **by** *auto*
show *?case* **apply** (*subst* (*1 2*) *s*) **unfolding** *matrix-sum.simps*
apply (*subst eq*, *subst eqf'*)
apply (*mat-assoc d*) **using** *dims* **by** *auto*
qed
have *p*: $(j + 1) + (n - j - 1) \le n$ **using** *j* **by** *auto*
show *?thesis* **using** *lm*[*OF p*] *j* **by** *auto*
qed

lemma *matrix-sum-Suc-remove-head*:
fixes $f :: nat \Rightarrow complex\ mat$
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 (*induct n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then have *dSS*: $\bigwedge k.\ k < Suc\ (Suc\ n) \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d$ **by** *auto*
have *ds*: $matrix\text{-}sum\ d\ (\lambda k.\ f\ (k + 1))\ n \in carrier\text{-}mat\ d\ d$ **using** *matrix-sum-dim*[*OF dSS, of n* $\lambda k.\ k + 1$] **by** *auto*
have $matrix\text{-}sum\ d\ f\ (Suc\ n + 1) = f\ (n + 1) + matrix\text{-}sum\ d\ f\ (n + 1)$ **by**

auto
  also have ... $= f\ (n + 1) + (f\ 0 + matrix\text{-}sum\ d\ (\lambda k.\ f\ (k + 1))\ n)$ **using** *Suc* **by** *auto*
  also have ... $= f\ 0 + (f\ (n + 1) + matrix\text{-}sum\ d\ (\lambda k.\ f\ (k + 1))\ n)$
    using *ds* **apply** (*mat-assoc d*) **using** *dSS* **by** *auto*
  finally show *?case* **by** *auto*
qed

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 (*induct n*)
  case *0*
  show *?case* **using** *positive-zero* **by** *auto*
next
  case (*Suc n*)
  then have *dfn*: $f\ n \in carrier\text{-}mat\ d\ d$ **and** *psn*: *positive* (*matrix-sum d f n*) **and** *pn*: *positive* (*f n*) **and** *d*: $k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d$ **for** *k* **by** *auto*
  then have *dsn*: $matrix\text{-}sum\ d\ f\ n \in carrier\text{-}mat\ d\ d$ **using** *matrix-sum-dim* **by** *auto*
  show *?case* **unfolding** *matrix-sum.simps* **using** *positive-add*[*OF pn psn dfn dsn*] **by** *auto*
qed

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 (*induct n*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc n*)
  then have $k < n \Longrightarrow f\ k \in carrier\text{-}mat\ d\ d$ **and** *dfn*: $f\ n \in carrier\text{-}mat\ d\ d$ **for** *k* **by** *auto*
  then have *dsfn*: $matrix\text{-}sum\ d\ f\ n \in carrier\text{-}mat\ d\ d$ **using** *matrix-sum-dim* **by** *auto*
  have $(f\ n + matrix\text{-}sum\ d\ f\ n) * A = f\ n * A + matrix\text{-}sum\ d\ f\ n * A$
    apply (*mat-assoc d*) **using** *Suc dsfn* **by** *auto*
  also have ... $= f\ n * A + matrix\text{-}sum\ d\ (\lambda k.\ f\ k * A)\ n$ **using** *Suc* **by** *auto*
  finally show *?case* **by** *auto*
qed

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 (*induct n*)
  case *0*

then show *?case* **by** *auto*
next
case (*Suc n*)
then have *dfn*: *f n* ∈ *carrier-mat d d* **and** *dgn*: *g n* ∈ *carrier-mat d d*
and *dfk*: *k* < *n* ⟹ *f k* ∈ *carrier-mat d d* **and** *dgk*: *k* < *n* ⟹ *g k* ∈ *carrier-mat d d*
and *eq*: *matrix-sum d* (λ*k*. *f k* + *g k*) *n* = *matrix-sum d f n* + *matrix-sum d g n* **for** *k* **by** *auto*
have *dsf*: *matrix-sum d f n* ∈ *carrier-mat d d* **using** *matrix-sum-dim dfk* **by** *auto*
have *dsg*: *matrix-sum d g n* ∈ *carrier-mat d d* **using** *matrix-sum-dim dgk* **by** *auto*
show *?case* **unfolding** *matrix-sum.simps eq*
using *dfn dgn dsf dsg* **by** (*mat-assoc d*)
qed

lemma *matrix-sum-minus-distrib*:
fixes *f g* :: *nat* ⇒ *complex mat*
shows (⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d d*) ⟹ (⋀*k*. *k* < *n* ⟹ *g k* ∈ *carrier-mat d d*)
⟹ *matrix-sum d* (λ*k*. (*f k*) − (*g k*)) *n* = *matrix-sum d f n* − *matrix-sum d g n*
proof −
have *eq*: −*1* $\cdot_m$ *g k* = − *g k* **for** *k* **by** *auto*
assume *dfk*: ⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d d* **and** *dgk*: ⋀*k*. *k* < *n* ⟹ (*g k*) ∈ *carrier-mat d d*
then have *k* < *n* ⟹ (*f k*) − (*g k*) = (*f k*) + (− (*g k*)) **for** *k* **by** *auto*
then have *matrix-sum d* (λ*k*. (*f k*) − (*g k*)) *n* = *matrix-sum d* (λ*k*. (*f k*) + (− (*g k*))) *n*
using *matrix-sum-cong*[*of n* λ*k*. (*f k*) − (*g k*)] *dfk dgk* **by** *auto*
also have . . . = *matrix-sum d f n* + *matrix-sum d* (λ*k*. − (*g k*)) *n*
using *matrix-sum-add-distrib*[*of n f*] *dfk dgk* **by** *auto*
also have . . . = *matrix-sum d f n* − *matrix-sum d g n*
apply (*subgoal-tac matrix-sum d* (λ*k*. − (*g k*)) *n* = − *matrix-sum d g n*, *auto*)
apply (*subgoal-tac* − *1* $\cdot_m$ *matrix-sum d g n* = − *matrix-sum d g n*)
by (*simp add*: *matrix-sum-smult*[*of n g d* −*1*, *OF dgk*, *simplified eq*, *simplified*], *auto*)
finally show *?thesis* **.**
qed

lemma *matrix-sum-shift-Suc*:
shows (⋀*k*. *k* < (*Suc n*) ⟹ *f k* ∈ *carrier-mat d d*)
⟹ *matrix-sum d f* (*Suc n*) = *f 0* + *matrix-sum d* (λ*k*. *f* (*Suc k*)) *n*
proof (*induct n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
have *dfk*: *k* < *Suc* (*Suc n*) ⟹ *f k* ∈ *carrier-mat d d* **for** *k* **using** *Suc* **by** *auto*
have *dsSk*: *k* < *Suc n* ⟹ *matrix-sum d* (λ*k*. *f* (*Suc k*)) *n* ∈ *carrier-mat d d* **for** *k* **using** *matrix-sum-dim*[*of* - λ*k*. *f* (*Suc k*)] *dfk* **by** *fastforce*

have *matrix-sum d f (Suc (Suc n))* = *f (Suc n)* + *matrix-sum d f (Suc n)* **by** *auto*
also have ... = *f (Suc n)* + *f 0* + *matrix-sum d* (λ*k. f (Suc k)*) *n* **using** *Suc dsSk assoc-add-mat*[*of f (Suc n) d d f 0*] **by** *fastforce*
also have ... = *f 0* + (*f (Suc n)* + *matrix-sum d* (λ*k. f (Suc k)*) *n*) **apply** (*mat-assoc d*) **using** *dsSk dfk* **by** *auto*
also have ... = *f 0* + *matrix-sum d* (λ*k. f (Suc k)*) (*Suc n*) **by** *auto*
finally show *?case* **.**
qed

lemma *lowner-le-matrix-sum*:
fixes *f g* :: *nat* ⇒ *complex mat*
shows (⋀*k. k* < *n* ⟹ *f k* ∈ *carrier-mat d d*) ⟹ (⋀*k. k* < *n* ⟹ *g k* ∈ *carrier-mat d d*)
⟹ (⋀*k. k* < *n* ⟹ *f k* $\leq_L$ *g k*)
⟹ *matrix-sum d f n* $\leq_L$ *matrix-sum d g n*
proof (*induct n*)
case *0*
show *?case* **unfolding** *matrix-sum.simps* **using** *lowner-le-refl*[*of* 0_m *d d d*] **by** *auto*
next
case (*Suc n*)
then have *dfn*: *f n* ∈ *carrier-mat d d* **and** *dgn*: *g n* ∈ *carrier-mat d d* **and** *le1*: *f n* $\leq_L$ *g n* **by** *auto*
then have *le2*: *matrix-sum d f n* $\leq_L$ *matrix-sum d g n* **using** *Suc* **by** *auto*
have *k* < *n* ⟹ *f k* ∈ *carrier-mat d d* **for** *k* **using** *Suc* **by** *auto*
then have *dsf*: *matrix-sum d f n* ∈ *carrier-mat d d* **using** *matrix-sum-dim* **by** *auto*
have *k* < *n* ⟹ *g k* ∈ *carrier-mat d d* **for** *k* **using** *Suc* **by** *auto*
then have *dsg*: *matrix-sum d g n* ∈ *carrier-mat d d* **using** *matrix-sum-dim* **by** *auto*
show *?case* **unfolding** *matrix-sum.simps* **using** *lowner-le-add dfn dsf dgn dsg le1 le2* **by** *auto*
qed

lemma *lowner-lub-add*:
assumes *matrix-seq d f matrix-seq d g* ∀ *n. trace* (*f n* + *g n*) ≤ *1*
shows *matrix-seq.lowner-lub* (λ*n. f n* + *g n*) = *matrix-seq.lowner-lub f* + *matrix-seq.lowner-lub g*
proof −
have *msf*: *matrix-seq.lowner-is-lub f* (*matrix-seq.lowner-lub f*) **using** *assms*(*1*) *matrix-seq.lowner-lub-prop* **by** *auto*
then have *limit-mat f* (*matrix-seq.lowner-lub f*) *d* **using** *matrix-seq.lowner-lub-is-limit assms* **by** *auto*
then have *lim1*: ∀ *i*<*d*. ∀ *j*<*d*. (λ*n. f n* $$ (*i, j*)) ⟶ (*matrix-seq.lowner-lub f*) $$ (*i, j*) **using** *limit-mat-def assms* **by** *auto*

have *msg*: *matrix-seq.lowner-is-lub g* (*matrix-seq.lowner-lub g*) **using** *assms*(*2*) *matrix-seq.lowner-lub-prop* **by** *auto*

then have *limit-mat g* (*matrix-seq.lowner-lub g*) *d* **using** *matrix-seq.lowner-lub-is-limit assms* **by** *auto*
then have *lim2*: ∀ *i*<*d*. ∀ *j*<*d*. (λ*n*. *g n* $$ (*i*, *j*)) ⟶ (*matrix-seq.lowner-lub g*) $$ (*i*, *j*) **using** *limit-mat-def assms* **by** *auto*

have ∀ *n*. *f n* + *g n* ∈ *carrier-mat d d* **using** *assms* **unfolding** *matrix-seq-def* **by** *fastforce*
moreover have ∀ *n*. *partial-density-operator* (*f n* + *g n*) **using** *assms*
unfolding *matrix-seq-def partial-density-operator-def* **using** *positive-add* **by** *blast*
moreover have (*f n* + *g n*) $\leq_L$ (*f* (*Suc n*) + *g* (*Suc n*)) **for** *n*
using *assms*
unfolding *matrix-seq-def* **using** *lowner-le-add*[*of f n d f* (*Suc n*) *g n g* (*Suc n*)] **by** *auto*
ultimately have *msfg*: *matrix-seq d* (λ*n*. *f n* + *g n*) **using** *assms* **unfolding** *matrix-seq-def* **by** *auto*
then have *mslfg*: *matrix-seq.lowner-is-lub* (λ*n*. *f n* + *g n*) (*matrix-seq.lowner-lub* (λ*n*. *f n* + *g n*))
using *matrix-seq.lowner-lub-prop* **by** *auto*
then have *limit-mat* (λ*n*. *f n* + *g n*) (*matrix-seq.lowner-lub* (λ*n*. *f n* + *g n*)) *d* **using** *matrix-seq.lowner-lub-is-limit msfg* **by** *auto*
then have *lim3*: ∀ *i*<*d*. ∀ *j*<*d*. (λ*n*. (*f n* + *g n*) $$ (*i*, *j*)) ⟶ (*matrix-seq.lowner-lub* (λ*n*. *f n* + *g n*)) $$ (*i*, *j*) **using** *limit-mat-def assms* **by** *auto*

have ∀ *i*<*d*. ∀ *j*<*d*. ∀ *n*. (*f n* + *g n*) $$ (*i*, *j*) = *f n* $$ (*i*, *j*) + *g n* $$ (*i*, *j*) **using** *assms* **unfolding** *matrix-seq-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *index-add-mat*(*1*))
then have *add*: ∀ *i*<*d*. ∀ *j*<*d*. (λ*n*. *f n* $$ (*i*, *j*) + *g n* $$ (*i*, *j*)) ⟶ (*matrix-seq.lowner-lub* (λ*n*. *f n* + *g n*)) $$ (*i*, *j*) **using** *lim3* **by** *auto*
have *matrix-seq.lowner-lub f* $$ (*i*, *j*) + *matrix-seq.lowner-lub g* $$ (*i*, *j*) = *matrix-seq.lowner-lub* (λ*n*. *f n* + *g n*) $$ (*i*, *j*)
if *i*: *i* < *d* **and** *j*: *j* < *d* **for** *i j*
proof −
have (λ*n*. *f n* $$ (*i*, *j*)) ⟶ *matrix-seq.lowner-lub f* $$ (*i*, *j*) **using** *lim1 i j* **by** *auto*
moreover have (λ*n*. *g n* $$ (*i*, *j*)) ⟶ *matrix-seq.lowner-lub g* $$ (*i*, *j*) **using** *lim2 i j* **by** *auto*
ultimately have (λ*n*. *f n* $$ (*i*, *j*) + *g n* $$ (*i*, *j*)) ⟶ *matrix-seq.lowner-lub f* $$ (*i*, *j*) + *matrix-seq.lowner-lub g* $$ (*i*, *j*)
using *tendsto-add*[*of* λ*n*. *f n* $$ (*i*, *j*) *matrix-seq.lowner-lub f* $$ (*i*, *j*) *sequentially* λ*n*. *g n* $$ (*i*, *j*) *matrix-seq.lowner-lub g* $$ (*i*, *j*)] **by** *auto*
moreover have (λ*n*. *f n* $$ (*i*, *j*) + *g n* $$ (*i*, *j*)) ⟶ *matrix-seq.lowner-lub* (λ*n*. *f n* + *g n*) $$ (*i*, *j*) **using** *add i j* **by** *auto*
ultimately show *?thesis* **using** *LIMSEQ-unique* **by** *auto*
qed
moreover have *matrix-seq.lowner-lub f* ∈ *carrier-mat d d* **using** *matrix-seq.lowner-is-lub-dim assms*(*1*) *msf* **unfolding** *matrix-seq-def* **by** *auto*
moreover have *matrix-seq.lowner-lub g* ∈ *carrier-mat d d* **using** *matrix-seq.lowner-is-lub-dim assms*(*2*) *msg* **unfolding** *matrix-seq-def* **by** *auto*

moreover have *matrix-seq.lowner-lub* ($\lambda n.$ *f n* + *g n*) $\in$ *carrier-mat d d* **using** *matrix-seq.lowner-is-lub-dim msfg mslfg* **unfolding** *matrix-seq-def* **by** *auto*
ultimately show *?thesis* **unfolding** *matrix-seq-def* **using** *mat-eq-iff* **by** *auto*
qed

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 −
have *msf*: *matrix-seq.lowner-is-lub f* (*matrix-seq.lowner-lub f*)
using *assms*(*1*) *matrix-seq.lowner-lub-prop* **by** *auto*
then have *limit-mat f* (*matrix-seq.lowner-lub f*) *d*
using *matrix-seq.lowner-lub-is-limit assms* **by** *auto*
then have *lim1*: $\forall\, i<d.\ \forall\, j<d.$ ($\lambda n.$ *f n* $$ (*i*, *j*)) $\longrightarrow$ (*matrix-seq.lowner-lub f*) $$ (*i*, *j*)
using *limit-mat-def assms* **by** *auto*

have *dimcf*: $\forall\, n.\ c \cdot_m f\ n \in$ *carrier-mat d d* **using** *assms* **unfolding** *matrix-seq-def* **by** *fastforce*
moreover have $\forall\, n.$ *partial-density-operator* ($c \cdot_m f\ n$) **using** *assms* **unfolding** *matrix-seq-def partial-density-operator-def* **using** *positive-scale* **by** *blast*
moreover have $\forall\, n.\ c \cdot_m f\ n \leq_L c \cdot_m f$ (*Suc n*) **using** *lowner-le-smult assms*(*1*,*3*)
unfolding *matrix-seq-def partial-density-operator-def* **by** *blast*
ultimately have *mscf*: *matrix-seq d* ($\lambda n.\ c \cdot_m f\ n$) **unfolding** *matrix-seq-def* **by** *auto*
then have *mslfg*: *matrix-seq.lowner-is-lub* ($\lambda n.\ c \cdot_m f\ n$) (*matrix-seq.lowner-lub* ($\lambda n.\ c \cdot_m f\ n$))
using *matrix-seq.lowner-lub-prop* **by** *auto*
then have *limit-mat* ($\lambda n.\ c \cdot_m f\ n$) (*matrix-seq.lowner-lub* ($\lambda n.\ c \cdot_m f\ n$)) *d*
using *matrix-seq.lowner-lub-is-limit mscf* **by** *auto*
then have *lim3*: $\forall\, i<d.\ \forall\, j<d.$ ($\lambda n.$ ($c \cdot_m f\ n$) $$ (*i*, *j*)) $\longrightarrow$ (*matrix-seq.lowner-lub* ($\lambda n.\ c \cdot_m f\ n$)) $$ (*i*, *j*)
using *limit-mat-def assms* **by** *auto*

from *mslfg mscf* **have** *dleft*: *matrix-seq.lowner-lub* ($\lambda n.\ c \cdot_m f\ n$) $\in$ *carrier-mat d d*
using *matrix-seq.lowner-is-lub-dim* **by** *auto*
have *dllf*: *matrix-seq.lowner-lub f* $\in$ *carrier-mat d d*
using *matrix-seq.lowner-is-lub-dim assms*(*1*) *msf* **unfolding** *matrix-seq-def* **by** *auto*
then have *dright*: $c \cdot_m$ *matrix-seq.lowner-lub f* $\in$ *carrier-mat d d* **using** *index-smult-mat*(*2*,*3*) **by** *auto*
have $\forall\ i<d.\ \forall\ j<d.\ \forall\ n.$ ($c \cdot_m f\ n$) $$ (*i*, *j*) = *c* ∗ *f n* $$ (*i*, *j*)
using *assms*(*1*) **unfolding** *matrix-seq-def* **using** *index-smult-mat*(*1*)
by (*metis carrier-matD*(*1*−*2*))
then have *smult*: $\forall\, i<d.\ \forall\, j<d.$ ($\lambda n.$ *c* ∗ *f n* $$ (*i*, *j*)) $\longrightarrow$ (*matrix-seq.lowner-lub*

$(\lambda n.\ c \cdot_m f\ n))$ $$ (i, j)
  using *lim3* **by** *auto*
 have *ij*: $(c \cdot_m$ *matrix-seq.lowner-lub f*$)$ $$ $(i, j) = ($*matrix-seq.lowner-lub* $(\lambda n.\ c \cdot_m f\ n))$ $$ (i, j)
  if *i*: $i < d$ **and** *j*: $j < d$ **for** *i j*
 proof −
  have $(\lambda n.\ f\ n$ $$ $(i, j)) \longrightarrow$ *matrix-seq.lowner-lub f* $$ (i, j) **using** *lim1 i j* **by** *auto*
  moreover have $\forall i<d.\ \forall j<d.(c \cdot_m$ *matrix-seq.lowner-lub f*$)$ $$ $(i, j) = c *$ *matrix-seq.lowner-lub f* $$ (i, j)
   using *index-smult-mat dllf* **by** *fastforce*
   ultimately have $\forall i<d.\ \forall j<d.\ (\lambda n.\ c * f\ n$ $$ $(i, j)) \longrightarrow (c \cdot_m$ *matrix-seq.lowner-lub f*$)$ $$ (i, j)
    using *tendsto-intros(18)*[*of* $\lambda n.\ c$ *c sequentially* $\lambda n.\ f\ n$ $$ (i, j) *matrix-seq.lowner-lub f* $$ (i, j)] *i j*
   by (*simp add*: *lim1 tendsto-mult-left*)
  then show *?thesis* **using** *smult i j LIMSEQ-unique* **by** *metis*
 qed

 from *dleft dright ij* **show** *?thesis*
  using *mat-eq-iff*[*of matrix-seq.lowner-lub* $(\lambda n.\ c \cdot_m f\ n)$ $c \cdot_m$ *matrix-seq.lowner-lub f*]
  by (*metis* (*mono-tags*) *carrier-matD*(*1*) *carrier-matD*(*2*))
qed

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* $(\lambda k.$ *trace* $(f\ k))$ $\{0..<n\}$
proof (*induct n*)
 case *0*
 show *?case* **by** *auto*
next
 case (*Suc n*)
 then have $\bigwedge k.\ k < n \Longrightarrow f\ k \in$ *carrier-mat d d* **by** *auto*
 then have *ds*: *matrix-sum d f n* ∈ *carrier-mat d d* **using** *matrix-sum-dim* **by** *auto*
 have *trace* (*matrix-sum d f* (*Suc n*)) = *trace* (*f n*) + *trace* (*matrix-sum d f n*)
  unfolding *matrix-sum.simps* **apply** (*mat-assoc d*) **using** *ds Suc* **by** *auto*
 also have ... = *sum* (*trace* ∘ *f*) $\{0..<n\}$ + (*trace* ∘ *f*) *n* **using** *Suc* **by** *auto*
 also have ... = *sum* (*trace* ∘ *f*) $\{0..<$*Suc n*$\}$ **by** *auto*
 finally show *?case* **by** *auto*
qed

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* $(\lambda k.\ P * (f\ k))\ n = P *$ (*matrix-sum d f n*)
proof (*induct n*)

case *0*
show *?case* **unfolding** *matrix-sum.simps* **using** *0* **by** *auto*
next
case (*Suc n*)
then have $\bigwedge k.\ k < n \Longrightarrow f\ k \in$ *carrier-mat d d* **by** *auto*
then have *ds*: *matrix-sum d f n* $\in$ *carrier-mat d d* **using** *matrix-sum-dim* **by** *auto*
then have *dPf*: $\bigwedge k.\ k < n \Longrightarrow P * f\ k \in$ *carrier-mat d d* **using** *Suc* **by** *auto*
then have *matrix-sum d* ($\lambda k.\ P * f\ k$) *n* $\in$ *carrier-mat d d* **using** *matrix-sum-dim*[*OF dPf*] **by** *auto*
have *matrix-sum d* ($\lambda k.\ P * f\ k$) (*Suc n*) $=$ $P * f\ n\ +$ *matrix-sum d* ($\lambda k.\ P * f\ k$) *n* **unfolding** *matrix-sum.simps* **using** *Suc*(*2*) **by** *auto*
also have ... $= P * f\ n + P *$ *matrix-sum d f n* **using** *Suc* **by** *auto*
also have ... $= P *$ (*f n* $+$ *matrix-sum d f n*) **apply** (*mat-assoc d*) **using** *ds dPf Suc* **by** *auto*
finally show *matrix-sum d* ($\lambda k.\ P * f\ k$) (*Suc n*) $= P *$ (*matrix-sum d f* (*Suc n*)) **by** *auto*
qed

2.4 Measurement

definition *measurement* :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ (*nat* $\Rightarrow$ *complex mat*) $\Rightarrow$ *bool* **where**
measurement d n M $\longleftrightarrow$ ($\forall j < n.$ *M j* $\in$ *carrier-mat d d*)
$\wedge$ *matrix-sum d* ($\lambda j.$ (*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*
using *assms* **unfolding** *measurement-def* **by** *auto*

lemma *measurement-id2*:
assumes *measurement d 2 M*
shows *adjoint* (*M 0*) $*$ *M 0* $+$ *adjoint* (*M 1*) $*$ *M 1* $= 1_m\ d$
proof −
have *ssz*: (*Suc* (*Suc 0*)) $=$ *2* **by** *auto*
have *M 0* $\in$ *carrier-mat d d M 1* $\in$ *carrier-mat d d* **using** *assms measurement-def* **by** *auto*
then have *adjoint* (*M 0*) $*$ *M 0* $+$ *adjoint* (*M 1*) $*$ *M 1* $=$ *matrix-sum d* ($\lambda j.$ (*adjoint* (*M j*)) $*$ *M j*) (*Suc* (*Suc 0*))
by *auto*
also have ... $=$ *matrix-sum d* ($\lambda j.$ (*adjoint* (*M j*)) $*$ *M j*) (*2*::*nat*) **by** (*subst ssz*, *auto*)
also have ... $= 1_m\ d$ **using** *measurement-def*[*of d 2 M*] *assms* **by** *auto*
finally show *?thesis* **by** *auto*
qed

Result of measurement on ρ by matrix M

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

```
lemma add-positive-le-reduce1:
  assumes dA: A ∈ carrier-mat n n and dB: B ∈ carrier-mat n n and dC: C ∈
carrier-mat n n
    and pB: positive B and le: A + B ≤L C
  shows A ≤L C
  unfolding lowner-le-def positive-def
proof (auto simp add: carrier-matD[OF dA] carrier-matD[OF dC] simp del: less-eq-complex-def)
  have eq: C − (A + B) = (C − A + (−B)) using dA dB dC by auto
  have positive (C − (A + B)) using le lowner-le-def dA dB dC by auto
  with eq have p: positive (C − A + (−B)) by auto
  fix v :: complex vec assume n = dim-vec v
  then have dv: v ∈ carrier-vec n by auto
  have ge: inner-prod v (B *v v) ≥ 0 using pB dv dB positive-def by auto
  have 0 ≤ inner-prod v ((C − A + (−B)) *v v) using p positive-def dv dA dB
dC by auto
  also have ... = inner-prod v ((C − A)*v v + (−B) *v v)
     using dv dA dB dC add-mult-distrib-mat-vec[OF minus-carrier-mat[OF dA]]
by auto
  also have ... = inner-prod v ((C − A) *v v) + inner-prod v ((−B) *v v)
    apply (subst inner-prod-distrib-right)
     by (rule dv, auto simp add: mult-mat-vec-carrier[OF minus-carrier-mat[OF
dA]] mult-mat-vec-carrier[OF uminus-carrier-mat[OF dB]] dv)
  also have ... = inner-prod v ((C − A) *v v) − inner-prod v (B *v v) using dB
dv by auto
  also have ... ≤ inner-prod v ((C − A) *v v) using ge by auto
  finally show 0 ≤ inner-prod v ((C − A) *v v).
qed

lemma add-positive-le-reduce2:
  assumes dA: A ∈ carrier-mat n n and dB: B ∈ carrier-mat n n and dC: C ∈
carrier-mat n n
    and pB: positive B and le: B + A ≤L C
  shows A ≤L C
  apply (subgoal-tac B + A = A + B) using add-positive-le-reduce1[of A n B C]
assms by auto

lemma measurement-le-one-mat:
  assumes measurement d n f
  shows ⋀j. j < n ⟹ adjoint (f j) * f j ≤L 1m d
proof −
  fix j assume j: j < n
  define M where M = adjoint (f j) * f j
  have df: k < n ⟹ f k ∈ carrier-mat d d for k using assms measurement-dim
by auto
  have daf: k < n ⟹ adjoint (f k) * f k ∈ carrier-mat d d for k
  proof −
    assume k < n
     then have f k ∈ carrier-mat d d adjoint (f k) ∈ carrier-mat d d using df
adjoint-dim by auto
```

then show *adjoint* (*f k*) ∗ *f k* ∈ *carrier-mat d d* **by** *auto*
qed
have *pafj*: $k < n \Longrightarrow$ *positive* (*adjoint* (*f k*) ∗ (*f k*)) **for** *k*
apply (*subst* (*2*) *adjoint-adjoint*[*of f k*, *symmetric*])
by (*metis adjoint-adjoint daf positive-if-decomp*)
define *f′* **where** ⋀*k*. *f′ k* = (*if k* = *j then* 0_m *d d else adjoint* (*f k*) ∗ *f k*)
have *pf′*: $k < n \Longrightarrow$ *positive* (*f′ k*) **for** *k* **unfolding** *f′-def* **using** *positive-zero pafj j* **by** *auto*
have *df′*: $k < n \Longrightarrow$ *f′ k* ∈ *carrier-mat d d* **for** *k* **using** *daf j zero-carrier-mat f′-def* **by** *auto*
then have *dsf′*: *matrix-sum d f′ n* ∈ *carrier-mat d d* **using** *matrix-sum-dim*[*of n f′ d*] **by** *auto*
have *psf′*: *positive* (*matrix-sum d f′ n*) **using** *matrix-sum-positive pafj df′ pf′* **by** *auto*
have *M* + *matrix-sum d f′ n* = *matrix-sum d* (λ*k*. *adjoint* (*f k*) ∗ *f k*) *n*
using *matrix-sum-remove*[*OF j* , *of* (λ*k*. *adjoint* (*f k*) ∗ *f k*), *OF daf*, *of f′*] *f′-def* **unfolding** *M-def* **by** *auto*
also have ... = 1_m *d* **using** *measurement-def assms* **by** *auto*
finally have *M* + *matrix-sum d f′ n* = 1_m *d*.
moreover have 1_m *d* $\le_L$ 1_m *d* **using** *lowner-le-refl*[*of - d*] **by** *auto*
ultimately have (*M* + *matrix-sum d f′ n*) $\le_L$ 1_m *d* **by** *auto*
then show *M* $\le_L$ 1_m *d* **unfolding** *M-def* **using** *add-positive-le-reduce1*[*OF - dsf′ one-carrier-mat psf′*] *daf j* **by** *auto*
qed

lemma *pdo-close-under-measurement*:
fixes *M* ϱ :: *complex mat*
assumes *dM*: *M* ∈ *carrier-mat n n* **and** *dr*: ϱ ∈ *carrier-mat n n*
and *pdor*: *partial-density-operator* ϱ
and *le*: *adjoint M* ∗ *M* $\le_L$ 1_m *n*
shows *partial-density-operator* (*M* ∗ ϱ ∗ *adjoint M*)
unfolding *partial-density-operator-def*
proof
show *positive* (*M* ∗ ϱ ∗ *adjoint M*)
using *positive-close-under-left-right-mult-adjoint*[*OF dM dr*] *pdor partial-density-operator-def* **by** *auto*
next
have *daM*: *adjoint M* ∈ *carrier-mat n n* **using** *dM* **by** *auto*
then have *daMM*: *adjoint M* ∗ *M* ∈ *carrier-mat n n* **using** *dM* **by** *auto*
have *trace* (*M* ∗ ϱ ∗ *adjoint M*) = *trace* (*adjoint M* ∗ *M* ∗ ϱ)
using *dM dr* **by** (*mat-assoc n*)
also have ... ≤ *trace* (1_m *n* ∗ ϱ)
using *lowner-le-trace*[**where** *?B* = 1_m *n* **and** *?A* = *adjoint M* ∗ *M*, *OF daMM one-carrier-mat*] *le dr pdor* **by** *auto*
also have ... = *trace* ϱ **using** *dr* **by** *auto*
also have ... ≤ *1* **using** *pdor partial-density-operator-def* **by** *auto*
finally show *trace* (*M* ∗ ϱ ∗ *adjoint M*) ≤ *1* **by** *auto*
qed

lemma *trace-measurement*:
assumes *m*: *measurement d n M* **and** *dA*: *A* ∈ *carrier-mat d d*
shows *trace* (*matrix-sum d* (*λk*. (*M k*) ∗ *A* ∗ *adjoint* (*M k*)) *n*) = *trace A*
proof −
have *dMk*: *k* < *n* ⟹ (*M k*) ∈ *carrier-mat d d* **for** *k* **using** *m* **unfolding** *measurement-def* **by** *auto*
then have *daMk*: *k* < *n* ⟹ *adjoint* (*M k*) ∈ *carrier-mat d d* **for** *k* **using** *m adjoint-dim* **unfolding** *measurement-def* **by** *auto*
have *d1*: *k* < *n* ⟹ *M k* ∗ *A* ∗ *adjoint* (*M k*) ∈ *carrier-mat d d***for** *k* **using** *dMk daMk dA* **by** *fastforce*
then have *ds1*: *k* < *n* ⟹ *matrix-sum d* (*λk*. *M k* ∗ *A* ∗ *adjoint* (*M k*)) *k* ∈ *carrier-mat d d* **for** *k*
using *matrix-sum-dim*[*of k λk*. *M k* ∗ *A* ∗ *adjoint* (*M k*) *d*] **by** *auto*
have *d2*: *k* < *n* ⟹ *adjoint* (*M k*) ∗*M k* ∗ *A* ∈ *carrier-mat d d* **for** *k* **using** *daMk dMk dA* **by** *fastforce*
then have *ds2*: *k* < *n* ⟹ *matrix-sum d* (*λk*. *adjoint* (*M k*) ∗*M k* ∗ *A*) *k* ∈ *carrier-mat d d* **for** *k*
using *matrix-sum-dim*[*of k λk*. *adjoint* (*M k*) ∗*M k* ∗ *A d*] **by** *auto*
have *daMMk*: *k* < *n* ⟹ *adjoint* (*M k*) ∗ *M k* ∈ *carrier-mat d d* **for** *k* **using** *dMk* **by** *fastforce*
have *k* ≤ *n* ⟹ *trace* (*matrix-sum d* (*λk*. (*M k*) ∗ *A* ∗ *adjoint* (*M k*)) *k*) = *trace* (*matrix-sum d* (*λk*. *adjoint* (*M k*) ∗ (*M k*) ∗ *A*) *k*) **for** *k*
proof (*induct k*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc k*)
then have *k*: *k* < *n* **by** *auto*
have *trace* (*M k* ∗ *A* ∗ *adjoint* (*M k*)) = *trace* (*adjoint* (*M k*) ∗ *M k* ∗ *A*)
using *dA* **apply** (*mat-assoc d*) **using** *dMk k* **by** *auto*
then show *?case* **unfolding** *matrix-sum.simps* **using** *ds1 ds2 d1 d2 k Suc daMk dMk dA*
by (*subst trace-add-linear*[*of - d*], *auto*)+
qed
then have *trace* (*matrix-sum d* (*λk*. (*M k*) ∗ *A* ∗ *adjoint* (*M k*)) *n*) = *trace* (*matrix-sum d* (*λk*. *adjoint* (*M k*) ∗ (*M k*) ∗ *A*) *n*) **by** *auto*
also have ... = *trace* (*matrix-sum d* (*λk*. *adjoint* (*M k*) ∗ (*M k*)) *n* ∗ *A*) **using** *matrix-sum-mult-right*[*OF daMMk*, *of n id A*] *dA* **by** *auto*
also have ... = *trace A* **using** *m dA* **unfolding** *measurement-def* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *mat-inc-seq-positive-transform*:
assumes *dfn*: ⋀*n*. *f n* ∈ *carrier-mat d d*
and *inc*: ⋀*n*. *f n* $\leq_L$ *f* (*Suc n*)
shows ⋀*n*. *f n* − *f 0* ∈ *carrier-mat d d* **and** ⋀*n*. (*f n* − *f 0*) $\leq_L$ (*f* (*Suc n*) − *f 0*)
proof −
show ⋀*n*. *f n* − *f 0* ∈ *carrier-mat d d* **using** *dfn* **by** *fastforce*

have *f 0* $\leq_L$ *f 0* **using** *lowner-le-refl[of f 0 d] dfn* **by** *auto*
then show (*f n* − *f 0*) $\leq_L$ (*f* (*Suc n*) − *f 0*) **for** *n*
using *lowner-le-minus[of f n d f (Suc n) f 0 f 0] dfn inc* **by** *fastforce*
qed

lemma *mat-inc-seq-lub*:
assumes *dfn*: ⋀*n*. *f n* ∈ *carrier-mat d d*
and *inc*: ⋀*n*. *f n* $\leq_L$ *f* (*Suc n*)
and *ub*: ⋀*n*. *f n* $\leq_L$ *A*
shows ∃ *B*. *lowner-is-lub f B* ∧ *limit-mat f B d*
proof −
have *dmfn0*: ⋀*n*. *f n* − *f 0* ∈ *carrier-mat d d* **and** *incm0*: ⋀*n*. (*f n* − *f 0*) $\leq_L$ (*f* (*Suc n*) − *f 0*)
using *mat-inc-seq-positive-transform[OF dfn, of id] assms* **by** *auto*
define *c* **where** *c* = *1* / (*trace* (*A* − *f 0*) + *1*)
have *f 0* $\leq_L$ *A* **using** *ub* **by** *auto*
then have *dA*: *A* ∈ *carrier-mat d d* **using** *ub* **unfolding** *lowner-le-def* **using** *dfn[of 0]* **by** *fastforce*
then have *dAmf0*: *A* − *f 0* ∈ *carrier-mat d d* **using** *dfn[of 0]* **by** *auto*
have *positive* (*A* − *f 0*) **using** *ub lowner-le-def* **by** *auto*
then have *tgeq0*: *trace* (*A* − *f 0*) ≥ *0* **using** *positive-trace dAmf0* **by** *auto*
then have *trace* (*A* − *f 0*) + *1* > *0* **by** (*auto simp*: *less-eq-complex-def less-complex-def*)
then have *gtc*: *c* > *0* **unfolding** *c-def* **using** *complex-is-Real-iff*
by (*auto simp*: *less-eq-complex-def less-complex-def*)
then have *gtci*: (*1* / *c*) > *0* **using** *complex-is-Real-iff*
by (*auto simp*: *less-eq-complex-def less-complex-def*)

have *trace* (*c* $\cdot_m$ (*A* − *f 0*)) = *c* ∗ *trace* (*A* − *f 0*)
using *trace-smult dAmf0* **by** *auto*
also have . . . = (*1* / (*trace* (*A* − *f 0*) + *1*)) ∗ *trace* (*A* − *f 0*) **unfolding** *c-def* **by** *auto*
also have . . . < *1* **using** *tgeq0* **by** (*simp add*: *complex-is-Real-iff less-eq-complex-def less-complex-def*)
finally have *lt1*: *trace* (*c* $\cdot_m$ (*A* − *f 0*)) < *1*.

have *le0*: − *f 0* $\leq_L$ − *f 0* **using** *lowner-le-refl[of − f 0 d] dfn* **by** *auto*

have *dmf0*: − *f 0* ∈ *carrier-mat d d* **using** *dfn* **by** *auto*
have *mf0smcle*: (*c* $\cdot_m$ (*X* − *f 0*)) $\leq_L$ (*c* $\cdot_m$ (*Y* − *f 0*)) **if** *X* $\leq_L$ *Y* **and** *X* ∈ *carrier-mat d d* **and** *Y* ∈ *carrier-mat d d* **for** *X Y*
proof −
have (*X* − *f 0*) $\leq_L$ (*Y* − *f 0*)
using *lowner-le-minus[of X d Y f 0 f 0] that dfn lowner-le-refl* **by** *auto*
then show *?thesis* **using** *lowner-le-smultc[of c (X − f 0) Y − f 0 d]* **using** *that dfn gtc* **by** *fastforce*
qed
have (*c* $\cdot_m$ (*f n* − *f 0*)) $\leq_L$ (*c* $\cdot_m$ (*A* − *f 0*)) **for** *n*
using *mf0smcle ub dfn dA* **by** *auto*
then have *trace* (*c* $\cdot_m$ (*f n* − *f 0*)) ≤ *trace* (*c* $\cdot_m$ (*A* − *f 0*)) **for** *n*

using *lowner-le-imp-trace-le*[*of c* $\cdot_m$ (*f n* − *f 0*) *d*] *dmfn0 dAmf0* **by** *auto*
then have *trlt1*: *trace* (*c* $\cdot_m$ (*f n* − *f 0*)) < *1* **for** *n* **using** *lt1*
unfolding *less-eq-complex-def less-complex-def*
by (*metis add.commute add-less-cancel-right add-mono-thms-linordered-field*(*3*))

have *f 0* $\le_L$ *f n* **for** *n*
proof (*induct n*)
case *0*
then show *?case* **using** *dfn lowner-le-refl* **by** *auto*
next
case (*Suc n*)
then show *?case* **using** *dfn lowner-le-trans*[*of f 0 d f n*] *inc* **by** *auto*
qed
then have *positive* (*f n* − *f 0*) **for** *n* **using** *lowner-le-def* **by** *auto*
then have *p*: *positive* (*c* $\cdot_m$ (*f n* − *f 0*)) **for** *n*
by (*intro positive-smult*, *insert gtc dmfn0*, *auto*)

have *inc′*: *c* $\cdot_m$ (*f n* − *f 0*) $\le_L$ *c* $\cdot_m$ (*f* (*Suc n*) − *f 0*) **for** *n*
using *incm0 lowner-le-smultc*[*of c f n* − *f 0*] *gtc dmfn0* **by** *fastforce*

define *g* **where** *g n* = *c* $\cdot_m$ (*f n* − *f 0*) **for** *n*
then have *positive* (*g n*) **and** *trace* (*g n*) < *1* **and** (*g n*) $\le_L$ (*g* (*Suc n*)) **and** *dgn*: (*g n*) ∈ *carrier-mat d d* **for** *n*
unfolding *g-def* **using** *p trlt1 inc′ dmfn0* **by** *auto*
then have *ms*: *matrix-seq d g* **unfolding** *matrix-seq-def partial-density-operator-def*
by (*simp add*: *less-eq-complex-def less-complex-def dual-order.strict-iff-not*)
then have *uniM*: ∃!*M*. *matrix-seq.lowner-is-lub g M* **using** *matrix-seq.lowner-lub-unique* **by** *auto*
then obtain *M* **where** *M*: *matrix-seq.lowner-is-lub g M* **by** *auto*
then have *leg*: *g n* $\le_L$ *M* **and** *lubg*: ⋀*M′*. (∀ *n*. *g n* $\le_L$ *M′*) ⟶ *M* $\le_L$ *M′* **for** *n*
unfolding *matrix-seq.lowner-is-lub-def*[*OF ms*] **by** *auto*
have *M* = *matrix-seq.lowner-lub g*
using *matrix-seq.lowner-lub-def*[*OF ms*] *M uniM theI-unique*[*of matrix-seq.lowner-is-lub g*] **by** *auto*
then have *limg*: *limit-mat g M d* **using** *M matrix-seq.lowner-lub-is-limit*[*OF ms*] **by** *auto*
then have *dM*: *M* ∈ *carrier-mat d d* **unfolding** *limit-mat-def* **by** *auto*

define *B* **where** *B* = *f 0* + (*1* / *c*) $\cdot_m$ *M*
have *eqinv*: *f 0* + (*1* / *c*) $\cdot_m$ (*c* $\cdot_m$ (*X* − *f 0*)) = *X* **if** *X* ∈ *carrier-mat d d* **for** *X*
proof −
have *f 0* + (*1* / *c*) $\cdot_m$ (*c* $\cdot_m$ (*X* − *f 0*)) = *f 0* + (*1* / *c* ∗ *c*) $\cdot_m$ (*X* − *f 0*)
apply (*subgoal-tac* (*1* / *c*) $\cdot_m$ (*c* $\cdot_m$ (*X* − *f 0*)) = (*1* / *c* ∗ *c*) $\cdot_m$ (*X* − *f 0*), *simp*)
using *smult-smult-mat dfn that* **by** *auto*
also have . . . = *f 0* + *1* $\cdot_m$ (*X* − *f 0*) **using** *gtc* **by** *auto*
also have . . . = *f 0* + (*X* − *f 0*) **by** *auto*

also have ... $=$ ($-$ *f 0*) $+$ *f 0* $+$ *X* **apply** (*mat-assoc d*) **using** *that dfn* **by** *auto*
also have ... $=$ 0_m *d d* $+$ *X* **using** *dfn uminus-l-inv-mat*[*of f 0 d d*] **by** *fastforce*
also have ... $=$ *X* **using** *that* **by** *auto*
finally show *?thesis* **by** *auto*
qed
have *limit-mat* (λn. (*1* / *c*) $\cdot_m$ *g n*) ((*1* / *c*) $\cdot_m$ *M*) *d* **using** *limit-mat-scale*[*OF limg*] *gtci* **by** *auto*
then have *limit-mat* (λn. *f 0* $+$ (*1* / *c*) $\cdot_m$ *g n*) (*f 0* $+$ (*1* / *c*) $\cdot_m$ *M*) *d*
using *mat-add-limit*[*of f 0*] *limg dfn* **unfolding** *mat-add-seq-def* **by** *auto*
then have *limf*: *limit-mat f B d* **using** *eqinv*[*OF dfn*] **unfolding** *B-def g-def* **by** *auto*

have *f0acmcile*: (*f 0* $+$ (*1* / *c*) $\cdot_m$ *X*) $\leq_L$ (*f 0* $+$ (*1* / *c*) $\cdot_m$ *Y*) **if** *X* $\leq_L$ *Y* **and** *X* $\in$ *carrier-mat d d* **and** *Y* $\in$ *carrier-mat d d* **for** *X Y*
proof $-$
have ((*1* / *c*) $\cdot_m$ *X*) $\leq_L$ ((*1* / *c*) $\cdot_m$ *Y*)
using *lowner-le-smultc*[*of 1*/*c*] *that gtci* **by** *fastforce*
then show (*f 0* $+$ (*1* / *c*) $\cdot_m$ *X*) $\leq_L$ (*f 0* $+$ (*1* / *c*) $\cdot_m$ *Y*)
using *lowner-le-add*[*of - d -* (*1* / *c*) $\cdot_m$ *X* (*1* / *c*) $\cdot_m$ *Y*]
that gtci dfn lowner-le-refl[*of f 0*, *OF dfn*] **by** *fastforce*
qed

have (*f 0* $+$ (*1* / *c*) $\cdot_m$ *g n*) $\leq_L$ (*f 0* $+$ (*1* / *c*) $\cdot_m$ *M*) **for** *n*
using *f0acmcile*[*OF leg dgn dM*] **by** *auto*
then have *lubf*: *f n* $\leq_L$ *B* **for** *n* **using** *eqinv*[*OF dfn*] *g-def B-def* **by** *auto*

{
fix *B′* **assume** *asm*: ∀ *n*. *f n* $\leq_L$ *B′*
then have *f 0* $\leq_L$ *B′* **by** *auto*
then have *dB′*: *B′* $\in$ *carrier-mat d d* **unfolding** *lowner-le-def* **using** *dfn*[*of 0*] **by** *auto*
have *f n* $\leq_L$ *B′* **for** *n* **using** *asm* **by** *auto*
then have (*c* $\cdot_m$ (*f n* $-$ *f 0*)) $\leq_L$ (*c* $\cdot_m$ (*B′* $-$ *f 0*)) **for** *n*
using *mf0smcle*[*of f n B′*] *dfn dB′* **by** *auto*
then have *g n* $\leq_L$ (*c* $\cdot_m$ (*B′* $-$ *f 0*)) **for** *n* **using** *g-def* **by** *auto*
then have *M* $\leq_L$ (*c* $\cdot_m$ (*B′* $-$ *f 0*)) **using** *lubg* **by** *auto*
then have (*f 0* $+$ (*1* / *c*) $\cdot_m$ *M*) $\leq_L$ (*f 0* $+$ (*1* / *c*) $\cdot_m$ (*c* $\cdot_m$ (*B′* $-$ *f 0*)))
using *f0acmcile*[*of M* (*c* $\cdot_m$ (*B′* $-$ *f 0*)), *OF - dM*] **using** *dB′ dfn* **by** *fastforce*
then have *B* $\leq_L$ *B′* **unfolding** *B-def* **using** *eqinv*[*OF dB′*] **by** *auto*
}
with *limf lubf* **have** ((∀ *n*. *f n* $\leq_L$ *B*) ∧ (∀ *M′*. (∀ *n*. *f n* $\leq_L$ *M′*) ⟶ *B* $\leq_L$ *M′*)) ∧ *limit-mat f B d* **by** *auto*
then show *?thesis* **unfolding** *lowner-is-lub-def* **by** *auto*
qed

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 (*induct n arbitrary*: *ϱ*)
case *0*
show *?case* **by** *auto*
next
case (*Suc n*)
show *?case*
apply (*subst denote-while-n-iter.simps*)
apply (*subst Suc*, *auto*)
done
qed

lemma *denote-while-n-iter-dim*:
ϱ ∈ *carrier-mat m m* ⟹ *partial-density-operator ϱ* ⟹ *M1* ∈ *carrier-mat m m* ⟹ *adjoint M1* ∗ *M1* $\leq_L$ 1_m *m*
⟹ (⋀*ϱ*. *ϱ* ∈ *carrier-mat m m* ⟹ *partial-density-operator ϱ* ⟹ *DS ϱ* ∈ *carrier-mat m m* ∧ *partial-density-operator* (*DS ϱ*))
⟹ *denote-while-n-iter M0 M1 DS n ϱ* ∈ *carrier-mat m m* ∧ *partial-density-operator* (*denote-while-n-iter M0 M1 DS n ϱ*)
proof (*induct n arbitrary*: *ϱ*)
case *0*
then show *?case* **unfolding** *denote-while-n-iter.simps* **by** *auto*
next
case (*Suc n*)
then have *dr*: *ϱ* ∈ *carrier-mat m m* **and** *dM1*: *M1* ∈ *carrier-mat m m* **by** *auto*
have *dMr*: *M1* ∗ *ϱ* ∗ *adjoint M1* ∈ *carrier-mat m m* **using** *dr dM1* **by** *fastforce*
have *pdoMr*: *partial-density-operator* (*M1* ∗ *ϱ* ∗ *adjoint M1*) **using** *pdo-close-under-measurement Suc* **by** *auto*
from *Suc dMr pdoMr* **have** *d*: *DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*) ∈ *carrier-mat m m* **and** *partial-density-operator* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)) **by** *auto*
then show *?case* **unfolding** *denote-while-n-iter.simps*
using *Suc* **by** *auto*
qed

lemma *pdo-denote-while-n-iter*:
ϱ ∈ *carrier-mat m m* ⟹ *partial-density-operator ϱ* ⟹ *M1* ∈ *carrier-mat m m* ⟹ *adjoint M1* ∗ *M1* $\leq_L$ 1_m *m*
⟹ (⋀*ϱ*. *ϱ* ∈ *carrier-mat m m* ∧ *partial-density-operator ϱ* ⟹ *partial-density-operator* (*DS ϱ*))

⟹ (⋀ϱ. ϱ ∈ *carrier-mat m m* ∧ *partial-density-operator* ϱ ⟹ *DS* ϱ ∈ *carrier-mat m m*)
⟹ *partial-density-operator* (*denote-while-n-iter M0 M1 DS n* ϱ)
proof (*induct n arbitrary*: ϱ)
case *0*
then show *?case* **unfolding** *denote-while-n-iter.simps* **by** *auto*
next
case (*Suc n*)
have *partial-density-operator* (*M1* ∗ ϱ ∗ *adjoint M1*) **using** *Suc pdo-close-under-measurement* **by** *auto*
moreover have *M1* ∗ ϱ ∗ *adjoint M1* ∈ *carrier-mat m m* **using** *Suc* **by** *auto*
ultimately have *p*: *partial-density-operator* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)) **and** *d*: *DS* (*M1* ∗ ϱ ∗ *adjoint M1*) ∈ *carrier-mat m m* **using** *Suc* **by** *auto*
show *?case* **unfolding** *denote-while-n-iter.simps* **using** *Suc*(*1*)[*OF d p Suc*(*4*) *Suc*(*5*)] *Suc* **by** *auto*
qed

Denotation of while is simply the infinite sum of denote_while_n

definition *denote-while* :: *complex mat* ⇒ *complex mat* ⇒ (*state* ⇒ *state*) ⇒ *state* ⇒ *state* **where**
denote-while M0 M1 DS ϱ = *matrix-inf-sum d* (λ*n*. *denote-while-n M0 M1 DS n* ϱ)

lemma *denote-while-n-dim*:
assumes ϱ ∈ *carrier-mat d d*
M0 ∈ *carrier-mat d d*
M1 ∈ *carrier-mat d d*
partial-density-operator ϱ
⋀ϱ′. ϱ′ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ′ ⟹ *positive* (*DS* ϱ′) ∧ *trace* (*DS* ϱ′) ≤ *trace* ϱ′ ∧ *DS* ϱ′ ∈ *carrier-mat d d*
shows *denote-while-n M0 M1 DS n* ϱ ∈ *carrier-mat d d*
proof (*induction n arbitrary*: ϱ)
case *0*
then show *?case*
proof −
have *M0* ∗ ϱ ∗ *adjoint M0* ∈ *carrier-mat d d*
using *assms assoc-mult-mat* **by** *auto*
then show *?thesis* **by** *auto*
qed
next
case (*Suc n*)
then show *?case*
proof −
have *denote-while-n M0 M1 DS n* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)) ∈ *carrier-mat d d*
using *Suc assms* **by** *auto*
then show *?thesis* **by** *auto*
qed
qed

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* ϱ' $\Longrightarrow$ *positive* (*DS* ϱ') $\wedge$ *trace* (*DS* ϱ') $\leq$ *trace* ϱ' $\wedge$ *DS* $\varrho' \in$ *carrier-mat d d*
  shows *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $\in$ *carrier-mat d d*
proof (*induct n*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc n*)
  then show *?case*
  proof −
    have *denote-while-n M0 M1 DS n* $\varrho \in$ *carrier-mat d d*
      using *denote-while-n-dim assms* **by** *auto*
     then have *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) (*Suc n*) $\in$ *carrier-mat d d*
      using *Suc* **by** *auto*
    then show *?thesis* **by** *auto*
  qed
qed

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 −
  have *ad*: *adjoint* $x * x \in$ *carrier-mat d d* **using** *adjoint-dim index-mult-mat dimx* **by** *auto*
  have *trace* ($x * \varrho *$ *adjoint* x) = *trace* ((*adjoint* $x * x$) $* \varrho$) **using** *dimx dimr* **by** (*mat-assoc d*)
  also have ... $\leq$ *trace* ($1_m\ d * \varrho$) **using** *lowner-le-trace un ad dimr pdo* **by** *auto*
  also have ... = *trace* ϱ **using** *dimr* **by** *auto*
  ultimately show *?thesis* **by** *auto*
qed

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* $\varrho \wedge$ *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 (*induction n arbitrary*: ϱ)
  case *0*
  then show *?case* **using** *positive-close-under-left-right-mult-adjoint dim0* **unfold-**

ing *partial-density-operator-def* **by** *auto*
next
  case (*Suc n*)
  then show *?case*
  proof −
   have *pdoM*: *partial-density-operator* (*M1* ∗ *ϱ* ∗ *adjoint M1*) **using** *pdo-close-under-measurement Suc dim1 un* **by** *auto*
    moreover have *cM*: *M1* ∗ *ϱ* ∗ *adjoint M1* ∈ *carrier-mat d d* **using** *Suc dim1 adjoint-dim index-mult-mat* **by** *auto*
     ultimately have *DSM1*: *positive* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)) ∧ *trace* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)) ≤ *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) ∧ *DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*) ∈ *carrier-mat d d*
      using *DS* **by** *auto*
   moreover have *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) ≤ *trace ϱ* **using** *trace-decrease-mul-adj Suc dim1 un* **by** *auto*
    ultimately have *partial-density-operator* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)) **using** *Suc* **unfolding** *partial-density-operator-def* **by** *auto*
     then have *positive* (*M0* ∗ *denote-while-n-iter M0 M1 DS n* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)) ∗ *adjoint M0*) **using** *Suc DSM1* **by** *auto*
    then have *positive* (*denote-while-n M0 M1 DS* (*Suc n*) *ϱ*) **by** *auto*
    then show *?thesis* **by** *auto*
  qed
qed

lemma *denote-while-n-sum-positive*:
  assumes *dim0*: *M0* ∈ *carrier-mat d d* **and** *dim1*: *M1* ∈ *carrier-mat d d* **and** *un*: *adjoint M1* ∗ *M1* $\leq_L$ 1_m *d*
      and *DS*: ⋀*ϱ*. *ϱ* ∈ *carrier-mat d d* ⟹ *partial-density-operator ϱ* ⟹ *positive* (*DS ϱ*) ∧ *trace* (*DS ϱ*) ≤ *trace ϱ* ∧ *DS ϱ* ∈ *carrier-mat d d*
      and *pdo*: *partial-density-operator ϱ* **and** *r*: *ϱ* ∈ *carrier-mat d d*
    shows *positive* (*matrix-sum d* (*λn*. *denote-while-n M0 M1 DS n ϱ*) *n*)
proof −
   have ⋀*k*. *k* < *n* ⟹ *positive* (*denote-while-n M0 M1 DS k ϱ*) **using** *assms denote-while-n-positive* **by** *auto*
  moreover have ⋀*k*. *k* < *n* ⟹ *denote-while-n M0 M1 DS k ϱ* ∈ *carrier-mat d d* **using** *denote-while-n-dim assms* **by** *auto*
  ultimately show *?thesis* **using** *matrix-sum-positive* **by** *auto*
qed

lemma *trace-measure2-id*:
  assumes *dM0*: *M0* ∈ *carrier-mat n n* **and** *dM1*: *M1* ∈ *carrier-mat n n*
    and *id*: *adjoint M0* ∗ *M0* + *adjoint M1* ∗ *M1* = 1_m *n*
    and *dA*: *A* ∈ *carrier-mat n n*
  shows *trace* (*M0* ∗ *A* ∗ *adjoint M0*) + *trace* (*M1* ∗ *A* ∗ *adjoint M1*) = *trace A*
proof −
   have *trace* (*M0* ∗ *A* ∗ *adjoint M0*) + *trace* (*M1* ∗ *A* ∗ *adjoint M1*) = *trace* ((*adjoint M0* ∗ *M0* + *adjoint M1* ∗ *M1*) ∗ *A*)
    using *assms* **by** (*mat-assoc n*)
  also have ... = *trace* (1_m *n* ∗ *A*) **using** *id* **by** *auto*

also have ... = *trace A* **using** *dA* **by** *auto*
finally show *?thesis*.
qed

lemma *measurement-lowner-le-one1*:
assumes *dim0*: *M0* ∈ *carrier-mat d d* **and** *dim1*: *M1* ∈ *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 −
have *paM0*: *positive* (*adjoint M0* ∗ *M0*)
apply (*subgoal-tac adjoint M0* ∗ *adjoint* (*adjoint M0*) = *adjoint M0* ∗ *M0*)
subgoal using *positive-if-decomp*[*of adjoint M0* ∗ *M0*] *dim0 adjoint-dim*[*OF dim0*] **by** *fastforce*
using *adjoint-adjoint*[*of M0*] **by** *auto*
have *le1*: *adjoint M0* ∗ *M0* + *adjoint M1* ∗ *M1* $\leq_L$ 1_m *d* **using** *id lowner-le-refl*[*of* 1_m *d*] **by** *fastforce*
show *adjoint M1* ∗ *M1* $\leq_L$ 1_m *d*
using *add-positive-le-reduce2*[*OF - - - paM0 le1*] *dim0 dim1* **by** *fastforce*
qed

lemma *denote-while-n-sum-trace*:
assumes *dim0*: *M0* ∈ *carrier-mat d d* **and** *dim1*: *M1* ∈ *carrier-mat d d* **and** *id*: *adjoint M0* ∗ *M0* + *adjoint M1* ∗ *M1* = 1_m *d*
and *DS*: ⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *positive* (*DS* ϱ) ∧ *trace* (*DS* ϱ) ≤ *trace* ϱ ∧ *DS* ϱ ∈ *carrier-mat d d*
and *r*: ϱ ∈ *carrier-mat d d*
and *pdor*: *partial-density-operator* ϱ
shows *trace* (*matrix-sum d* (λ*n*. *denote-while-n M0 M1 DS n* ϱ) *n*) ≤ *trace* ϱ
proof −
have *un*: *adjoint M1* ∗ *M1* $\leq_L$ 1_m *d* **using** *measurement-lowner-le-one1* **using** *dim0 dim1 id* **by** *auto*
have *DS'*: (*DS* ϱ ∈ *carrier-mat d d*) ∧ *partial-density-operator* (*DS* ϱ) **if** ϱ ∈ *carrier-mat d d* **and** *partial-density-operator* ϱ **for** ϱ
proof −
have *res*: *positive* (*DS* ϱ) ∧ *trace* (*DS* ϱ) ≤ *trace* ϱ ∧ *DS* ϱ ∈ *carrier-mat d d* **using** *DS that* **by** *auto*
moreover have *trace* ϱ ≤ *1* **using** *that partial-density-operator-def* **by** *auto*
ultimately have *trace* (*DS* ϱ) ≤ *1* **by** *auto*
with *res* **show** *?thesis* **unfolding** *partial-density-operator-def* **by** *auto*
qed
have *dWk*: *denote-while-n-iter M0 M1 DS k* ϱ ∈ *carrier-mat d d* **for** *k*
using *denote-while-n-iter-dim*[*OF r pdor dim1 un*] *DS'* *dim0 dim1* **by** *auto*
have *pdoWk*: *partial-density-operator* (*denote-while-n-iter M0 M1 DS k* ϱ) **for** *k*
using *pdo-denote-while-n-iter*[*OF r pdor dim1 un*] *DS'* *dim0 dim1* **by** *auto*
have *dW0k*: *denote-while-n M0 M1 DS k* ϱ ∈ *carrier-mat d d* **for** *k* **using** *denote-while-n-dim r dim0 dim1 pdor* **by** *auto*
then have *dsW0k*: *matrix-sum d* (λ*n*. *denote-while-n M0 M1 DS n* ϱ) *k* ∈ *carrier-mat d d* **for** *k*
using *matrix-sum-dim*[*of k* λ*k*. *denote-while-n M0 M1 DS k* ϱ] **by** *auto*

have (*denote-while-n-comp M0 M1 DS n ϱ*) ∈ *carrier-mat d d* **for** *n* **unfolding** *denote-while-n-comp.simps* **using** *dim1 dWk* **by** *auto*
moreover have
pdoW1k: *partial-density-operator* (*denote-while-n-comp M0 M1 DS n ϱ*) **for** *n* **unfolding** *denote-while-n-comp.simps*
using *pdo-close-under-measurement*[*OF dim1 dWk pdoWk un*] **by** *auto*
ultimately have *trace* (*DS* (*denote-while-n-comp M0 M1 DS n ϱ*)) ≤ *trace* (*denote-while-n-comp M0 M1 DS n ϱ*) **for** *n*
using *DS* **by** *auto*
moreover have *trace* (*denote-while-n-iter M0 M1 DS* (*Suc n*) *ϱ*) = *trace* (*DS* (*denote-while-n-comp M0 M1 DS n ϱ*)) **for** *n*
using *denote-while-n-iter-assoc*[*folded denote-while-n-comp.simps*] **by** *auto*
ultimately have *leq3*: *trace* (*denote-while-n-iter M0 M1 DS* (*Suc n*) *ϱ*) ≤ *trace* (*denote-while-n-comp M0 M1 DS n ϱ*) **for** *n* **by** *auto*

have *mainleq*: *trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc n*)) + *trace* (*denote-while-n-comp M0 M1 DS n ϱ*) ≤ *trace ϱ* **for** *n*
proof (*induct n*)
case *0*
then show *?case* **unfolding** *matrix-sum.simps denote-while-n.simps denote-while-n-comp.simps denote-while-n-iter.simps*
apply (*subgoal-tac M0* ∗ *ϱ* ∗ *adjoint M0* + 0_m *d d* = *M0* ∗ *ϱ* ∗ *adjoint M0*)
using *trace-measure2-id*[*OF dim0 dim1 id r*] *dim0* **apply** *simp*
using *dim0* **by** *auto*
next
case (*Suc n*)

have *eq1*: *trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc* (*Suc n*)))
= *trace* (*denote-while-n M0 M1 DS* (*Suc n*) *ϱ*) + *trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc n*))
unfolding *matrix-sum.simps*
using *trace-add-linear dW0k*[*of Suc n*] *dsW0k*[*of Suc n*] **by** *auto*

have *eq2*: *trace* (*denote-while-n M0 M1 DS* (*Suc n*) *ϱ*) + *trace* (*denote-while-n-comp M0 M1 DS* (*Suc n*) *ϱ*)
= *trace* (*denote-while-n-iter M0 M1 DS* (*Suc n*) *ϱ*)
unfolding *denote-while-n.simps denote-while-n-comp.simps* **using** *trace-measure2-id*[*OF dim0 dim1 id dWk*[*of Suc n*]] **by** *auto*

have *trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc* (*Suc n*))) + *trace* (*denote-while-n-comp M0 M1 DS* (*Suc n*) *ϱ*)
= *trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc n*)) + *trace* (*denote-while-n M0 M1 DS* (*Suc n*) *ϱ*) + *trace* (*denote-while-n-comp M0 M1 DS* (*Suc n*) *ϱ*)
using *eq1* **by** *auto*
also have ... = *trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc n*)) + *trace* (*denote-while-n-iter M0 M1 DS* (*Suc n*) *ϱ*)

using *eq2* **by** *auto*
also have ... $\leq$ *trace* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) (*Suc n*)) + *trace* (*denote-while-n-comp M0 M1 DS n* ϱ)
using *leq3* **by** *auto*
also have ... $\leq$ *trace* ϱ **using** *Suc* **by** *auto*
finally show *?case*.
qed

have *reduce-le-complex*: (*b*::*complex*) $\geq$ *0* $\Longrightarrow$ *a* + *b* $\leq$ *c* $\Longrightarrow$ *a* $\leq$ *c* **for** *a b c*
by (*simp add*: *less-eq-complex-def*)
have *positive* (*denote-while-n-comp M0 M1 DS n* ϱ) **for** *n* **using** *pdoW1k* **unfolding** *partial-density-operator-def* **by** *auto*
then have *trace* (*denote-while-n-comp M0 M1 DS n* ϱ) $\geq$ *0* **for** *n* **using** *positive-trace*
using ‹$\bigwedge n$. *denote-while-n-comp M0 M1 DS n* $\varrho \in$ *carrier-mat d d*› **by** *blast*
then have *trace* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) (*Suc n*)) $\leq$ *trace* ϱ **for** *n*
using *mainleq reduce-le-complex*[*of trace* (*denote-while-n-comp M0 M1 DS n* ϱ)] **by** *auto*
moreover have *trace* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *0*) $\leq$ *trace* ϱ
unfolding *matrix-sum.simps*
using *trace-zero positive-trace pdor* **unfolding** *partial-density-operator-def*
using *r* **by** *auto*
ultimately show *trace* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n*) $\leq$ *trace* ϱ **for** *n*
apply (*induct n*) **by** *auto*
qed

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* ϱ $\Longrightarrow$ *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 (*partial-density-operator* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n*))
proof –
have *trace* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n*) $\leq$ *trace* ϱ
using *denote-while-n-sum-trace assms* **by** *auto*
then have *trace* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n*) $\leq$ *1*
using *pdo* **unfolding** *partial-density-operator-def* **by** *auto*
moreover have *positive* (*matrix-sum d* (λn. *denote-while-n M0 M1 DS n* ϱ) *n*)
using *assms DS denote-while-n-sum-positive measurement-lowner-le-one1*[*OF dim0 dim1 id*] **by** *auto*
ultimately show *?thesis* **unfolding** *partial-density-operator-def* **by** *auto*
qed

lemma *denote-while-n-sum-lowner-le*:

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* $\varrho \wedge$ *DS* $\varrho \in$ *carrier-mat d d*
and *pdo*: *partial-density-operator* ϱ **and** *dimr*: $\varrho \in$ *carrier-mat d d*
shows (*matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $\leq_L$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) (*Suc n*))
proof *auto*
have *whilenc*: *denote-while-n M0 M1 DS n* $\varrho \in$ *carrier-mat d d* **using** *denote-while-n-dim assms* **by** *auto*
have *sumc*: *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $\in$ *carrier-mat d d*
using *denote-while-n-sum-dim assms* **by** *auto*
have *denote-while-n M0 M1 DS n* ϱ $+$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $-$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n*
$=$ *denote-while-n M0 M1 DS n* ϱ $+$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $+$ ($-$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n*)
using *minus-add-uminus-mat*[*of matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n d d matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n*] **by** *auto*
also have ... $=$ *denote-while-n M0 M1 DS n* ϱ $+ 0_m$ *d d*
by (*smt* (*verit*) *assoc-add-mat minus-add-uminus-mat minus-r-inv-mat sumc uminus-carrier-mat whilenc*)
also have ... $=$ *denote-while-n M0 M1 DS n* ϱ **using** *whilenc* **by** *auto*
finally have *simp*: *denote-while-n M0 M1 DS n* ϱ $+$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $-$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $=$
denote-while-n M0 M1 DS n ϱ **by** *auto*
have *positive* (*denote-while-n M0 M1 DS n* ϱ) **using** *denote-while-n-positive assms measurement-lowner-le-one1*[*OF dim0 dim1 id*] **by** *auto*
then have *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $\leq_L$ (*denote-while-n M0 M1 DS n* ϱ $+$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n*)
unfolding *lowner-le-def* **using** *simp* **by** *auto*
then show *matrix-sum d* ($\lambda n.$ *M0* $*$ *denote-while-n-iter M0 M1 DS n* ϱ $*$ *adjoint M0*) *n* $\leq_L$
(*M0* $*$ *denote-while-n-iter M0 M1 DS n* ϱ $*$ *adjoint M0* $+$ *matrix-sum d* ($\lambda n.$ *M0* $*$ *denote-while-n-iter M0 M1 DS n* ϱ $*$ *adjoint M0*) *n*) **by** *auto*
qed

lemma *lowner-is-lub-matrix-sum*:
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* $\varrho \wedge$ *DS* $\varrho \in$ *carrier-mat d d*
and *pdo*: *partial-density-operator* ϱ **and** *dimr*: $\varrho \in$ *carrier-mat d d*
shows *matrix-seq.lowner-is-lub* (*matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ)) (*matrix-seq.lowner-lub* (*matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ)))
proof−
have *sumdd*: $\forall n.$ *matrix-sum d* ($\lambda n.$ *denote-while-n M0 M1 DS n* ϱ) *n* $\in$ *carrier-mat d d*

using *denote-while-n-sum-dim assms* **by** *auto*
have *sumtr*: ∀ *n. trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) *n*) ≤ *trace ϱ*
using *denote-while-n-sum-trace assms* **by** *auto*
have *sumpar*: ∀ *n. partial-density-operator* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) *n*)
using *denote-while-n-sum-partial-density assms* **by** *auto*
have *sumle*:∀ *n. matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) *n* $\leq_L$ *matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc n*)
using *denote-while-n-sum-lowner-le assms* **by** *auto*
have *seqd*: *matrix-seq d* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*))
using *matrix-seq-def sumdd sumpar sumle* **by** *auto*
then show *?thesis* **using** *matrix-seq.lowner-lub-prop*[*of d* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*))] **by** *auto*
qed

lemma *denote-while-dim-positive*:
assumes *dim0*: *M0* ∈ *carrier-mat d d* **and** *dim1*: *M1* ∈ *carrier-mat d d* **and** *id*: *adjoint M0* ∗ *M0* + *adjoint M1* ∗ *M1* = 1_m *d*
and *DS*: ⋀*ϱ. ϱ* ∈ *carrier-mat d d* ⟹ *partial-density-operator ϱ* ⟹ *positive* (*DS ϱ*) ∧ *trace* (*DS ϱ*) ≤ *trace ϱ* ∧ *DS ϱ* ∈ *carrier-mat d d*
and *pdo*: *partial-density-operator ϱ* **and** *dimr*: *ϱ* ∈ *carrier-mat d d*
shows
denote-while M0 M1 DS ϱ ∈ *carrier-mat d d* ∧ *positive* (*denote-while M0 M1 DS ϱ*) ∧ *trace* (*denote-while M0 M1 DS ϱ*) ≤ *trace ϱ*
proof −
have *sumdd*: ∀ *n. matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) *n* ∈ *carrier-mat d d*
using *denote-while-n-sum-dim assms* **by** *auto*
have *sumtr*: ∀ *n. trace* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) *n*) ≤ *trace ϱ*
using *denote-while-n-sum-trace assms* **by** *auto*
have *sumpar*: ∀ *n. partial-density-operator* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) *n*)
using *denote-while-n-sum-partial-density assms* **by** *auto*
have *sumle*:∀ *n. matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) *n* $\leq_L$ *matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*) (*Suc n*)
using *denote-while-n-sum-lowner-le assms* **by** *auto*
have *seqd*: *matrix-seq d* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*))
using *matrix-seq-def sumdd sumpar sumle* **by** *auto*
have *matrix-seq.lowner-is-lub* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*)) (*matrix-seq.lowner-lub* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*)))
using *lowner-is-lub-matrix-sum assms* **by** *auto*
then have *matrix-seq.lowner-lub* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*)) ∈ *carrier-mat d d*
∧ *positive* (*matrix-seq.lowner-lub* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*)))
∧ *trace* (*matrix-seq.lowner-lub* (*matrix-sum d* (*λn. denote-while-n M0 M1 DS n ϱ*))) ≤ *trace ϱ*

using *matrix-seq.lowner-is-lub-dim seqd matrix-seq.lowner-lub-is-positive matrix-seq.lowner-lub-trace sumtr* **by** *auto*
then show *?thesis* **unfolding** *denote-while-def matrix-inf-sum-def* **by** *auto*
qed

definition *denote-measure* :: *nat* ⇒ (*nat* ⇒ *complex mat*) ⇒ ((*state* ⇒ *state*) *list*) ⇒ *state* ⇒ *state* **where**
denote-measure n M DS ϱ = *matrix-sum d* (λk. (*DS*!*k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*))) *n*

lemma *denote-measure-dim*:
assumes ϱ ∈ *carrier-mat d d*
measurement d n M
$\bigwedge\varrho'$ *k*. ϱ' ∈ *carrier-mat d d* ⟹ *k* < *n* ⟹ (*DS*!*k*) ϱ' ∈ *carrier-mat d d*
shows
denote-measure n M DS ϱ ∈ *carrier-mat d d*
proof −
have *dMk*: *k* < *n* ⟹ *M k* ∈ *carrier-mat d d* **for** *k* **using** *assms measurement-def* **by** *auto*
have *d*: *k* < *n* ⟹ (*M k*) ∗ ϱ ∗ *adjoint* (*M k*) ∈ *carrier-mat d d* **for** *k*
using *mult-carrier-mat*[*OF mult-carrier-mat*[*OF dMk assms*(*1*)] *adjoint-dim*[*OF dMk*]] **by** *auto*
then have *k* < *n* ⟹ (*DS*!*k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) ∈ *carrier-mat d d* **for** *k* **using** *assms*(*3*) **by** *auto*
then show *?thesis* **unfolding** *denote-measure-def* **using** *matrix-sum-dim*[*of n* λk. (*DS*!*k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*))] **by** *auto*
qed

lemma *measure-well-com*:
assumes *well-com* (*Measure n M S*)
shows $\bigwedge k$. *k* < *n* ⟹ *well-com* (*S* ! *k*)
using *assms* **unfolding** *well-com.simps* **using** *list-all-length* **by** *auto*

Semantics of commands

fun *denote* :: *com* ⇒ *state* ⇒ *state* **where**
denote SKIP ϱ = ϱ
| *denote* (*Utrans U*) ϱ = *U* ∗ ϱ ∗ *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* ≤ *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*) ∗ ϱ ∗ *adjoint* (*M k*))) *m*
unfolding *denote.simps denote-measure-def*
proof −
have *k* < *m* ⟹ *map denote S* ! *k* = *denote* (*S*!*k*) **for** *k* **using** *wc m* **by** *auto*

then have *k* < *m* ⟹ (*map denote S* ! *k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*)) = *denote* (*S*!*k*) ((*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*)) **for** *k* **by** *auto*
then show *matrix-sum d* (λ*k*. (*map denote S* ! *k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*))) *m* = *matrix-sum d* (λ*k*. *denote* (*S* ! *k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*))) *m*
using *matrix-sum-cong*[*of m* λ*k*. (*map denote S* ! *k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*)) λ*k*. *denote* (*S* ! *k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*))] **by** *auto*
qed

lemma *matrix-sum-trace-le*:
fixes *f* :: *nat* ⇒ *complex mat* **and** *g* :: *nat* ⇒ *complex mat*
assumes (⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d d*)
(⋀*k*. *k* < *n* ⟹ *g k* ∈ *carrier-mat d d*)
(⋀*k*. *k* < *n* ⟹ *trace* (*f k*) ≤ *trace* (*g k*))
shows *trace* (*matrix-sum d f n*) ≤ *trace* (*matrix-sum d g n*)
proof −
have *sum* (λ*k*. *trace* (*f k*)) {*0*..<*n*} ≤ *sum* (λ*k*. *trace* (*g k*)) {*0*..<*n*}
using *assms* **by** (*meson atLeastLessThan-iff sum-mono*)
then show *?thesis* **using** *trace-matrix-sum-linear assms* **by** *auto*
qed

lemma *map-denote-positive-trace-dim*:
assumes *well-com* (*Measure x1 x2a x3a*)
x4 ∈ *carrier-mat d d*
partial-density-operator x4
⋀*x3aa ϱ*. *x3aa* ∈ *set x3a* ⟹ *well-com x3aa* ⟹ *ϱ* ∈ *carrier-mat d d* ⟹ *partial-density-operator ϱ*
⟹ *positive* (*denote x3aa ϱ*) ∧ *trace* (*denote x3aa ϱ*) ≤ *trace ϱ* ∧ *denote x3aa ϱ* ∈ *carrier-mat d d*
shows ∀ *k* < *x1*. *positive* ((*map denote x3a* ! *k*) (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*)))
∧ ((*map denote x3a* ! *k*) (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*))) ∈ *carrier-mat d d*
∧ *trace* ((*map denote x3a* ! *k*) (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*))) ≤ *trace* (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*))
proof −
have *x2ak*: ∀ *k* < *x1*. *x2a k* ∈ *carrier-mat d d* **using** *assms*(*1*) *measurement-dim* **by** *auto*
then have *x2aa*:∀ *k* < *x1*. (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*)) ∈ *carrier-mat d d* **using** *assms*(*2*) **by** *fastforce*
have *posct*: *positive* ((*map denote x3a* ! *k*) (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*)))
∧ ((*map denote x3a* ! *k*) (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*))) ∈ *carrier-mat d d*
∧ *trace* ((*map denote x3a* ! *k*) (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*))) ≤ *trace* (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*))
if *k*: *k* < *x1* **for** *k*
proof −
have *lea*: *adjoint* (*x2a k*) ∗ *x2a k* $\leq_L$ 1_m *d* **using** *measurement-le-one-mat assms*(*1*) *k* **by** *auto*
have (*x2a k* ∗ *x4* ∗ *adjoint* (*x2a k*)) ∈ *carrier-mat d d* **using** *k x2aa assms*(*2*) **by** *fastforce*

moreover have *(x3a ! k)* ∈ *set x3a* **using** *k assms(1)* **by** *simp*
moreover have *well-com (x3a ! k)* **using** *k assms(1)* **using** *measure-well-com* **by** *blast*
moreover have *partial-density-operator (x2a k ∗ x4 ∗ adjoint (x2a k))*
using *pdo-close-under-measurement x2ak assms(2,3) lea k* **by** *blast*
ultimately have *positive (denote (x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k)))*
∧ *(denote (x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k)))* ∈ *carrier-mat d d*
∧ *trace (denote (x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k)))* ≤ *trace (x2a k ∗ x4 ∗ adjoint (x2a k))*
using *assms(4)* **by** *auto*
then show *?thesis* **using** *assms(1) k* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed

lemma *denote-measure-positive-trace-dim*:
assumes *well-com (Measure x1 x2a x3a)*
x4 ∈ *carrier-mat d d*
partial-density-operator x4
⋀*x3aa ϱ. x3aa* ∈ *set x3a* ⟹ *well-com x3aa* ⟹ *ϱ* ∈ *carrier-mat d d* ⟹ *partial-density-operator ϱ*
⟹ *positive (denote x3aa ϱ)* ∧ *trace (denote x3aa ϱ)* ≤ *trace ϱ* ∧ *denote x3aa ϱ* ∈ *carrier-mat d d*
shows *positive (denote (Measure x1 x2a x3a) x4)* ∧ *trace (denote (Measure x1 x2a x3a) x4)* ≤ *trace x4*
∧ *(denote (Measure x1 x2a x3a) x4)* ∈ *carrier-mat d d*
proof −
have *x2ak*: ∀ *k < x1. x2a k* ∈ *carrier-mat d d* **using** *assms(1) measurement-dim* **by** *auto*
then have *x2aa*:∀ *k < x1. (x2a k ∗ x4 ∗ adjoint (x2a k))* ∈ *carrier-mat d d* **using** *assms(2)* **by** *fastforce*
have *posct*:∀ *k < x1. positive ((map denote x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k)))*
∧ *((map denote x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k)))* ∈ *carrier-mat d d*
∧ *trace ((map denote x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k)))* ≤ *trace (x2a k ∗ x4 ∗ adjoint (x2a k))*
using *map-denote-positive-trace-dim assms* **by** *auto*

have *trace (matrix-sum d (λk. (map denote x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k))) x1)*
≤ *trace (matrix-sum d (λk. (x2a k ∗ x4 ∗ adjoint (x2a k))) x1)*
using *posct matrix-sum-trace-le[of x1 (λk. (map denote x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k))) (λk. x2a k ∗ x4 ∗ adjoint (x2a k))]*
x2aa **by** *auto*
also have . . . = *trace x4* **using** *trace-measurement[of d x1 x2a x4] assms(1,2)* **by** *auto*
finally have *trace (matrix-sum d (λk. (map denote x3a ! k) (x2a k ∗ x4 ∗ adjoint (x2a k))) x1)* ≤ *trace x4* **by** *auto*

then have *trace* (*denote-measure x1 x2a* (*map denote x3a*) *x4*) ≤ *trace x4*
unfolding *denote-measure-def* **by** *auto*
then have *trace* (*denote* (*Measure x1 x2a x3a*) *x4*) ≤ *trace x4* **by** *auto*
moreover from *posct* **have** *positive* (*denote* (*Measure x1 x2a x3a*) *x4*)
apply *auto*
unfolding *denote-measure-def* **using** *matrix-sum-positive* **by** *auto*
moreover have (*denote* (*Measure x1 x2a x3a*) *x4*) ∈ *carrier-mat d d*
apply *auto*
unfolding *denote-measure-def* **using** *matrix-sum-dim posct*
by (*simp add*: *matrix-sum-dim*)
ultimately show *?thesis* **by** *auto*
qed

lemma *denote-positive-trace-dim*:
well-com S ⟹ *ϱ* ∈ *carrier-mat d d* ⟹ *partial-density-operator ϱ*
⟹ (*positive* (*denote S ϱ*) ∧ *trace* (*denote S ϱ*) ≤ *trace ϱ* ∧ *denote S ϱ* ∈ *carrier-mat d d*)
proof (*induction arbitrary*: *ϱ*)
case *SKIP*
then show *?case* **unfolding** *partial-density-operator-def* **by** *auto*
next
case (*Utrans x*)
then show *?case*
proof −
assume *wc*: *well-com* (*Utrans x*) **and** *r*: *ϱ* ∈ *carrier-mat d d* **and** *pdo*: *partial-density-operator ϱ*
show *positive* (*denote* (*Utrans x*) *ϱ*) ∧ *trace* (*denote* (*Utrans x*) *ϱ*) ≤ *trace ϱ* ∧ *denote* (*Utrans x*) *ϱ* ∈ *carrier-mat d d*
proof −
have *trace* (*x* ∗ *ϱ* ∗ *adjoint x*) = *trace* ((*adjoint x* ∗ *x*) ∗ *ϱ*)
using *r* **apply** (*mat-assoc d*) **using** *wc* **by** *auto*
also have . . . = *trace* (1_m *d* ∗ *ϱ*) **using** *wc inverts-mat-def inverts-mat-symm adjoint-dim* **by** *auto*
also have . . . = *trace ϱ* **using** *r* **by** *auto*
finally have *fst*: *trace* (*x* ∗ *ϱ* ∗ *adjoint x*) = *trace ϱ* **by** *auto*
moreover have *positive* (*x* ∗ *ϱ* ∗ *adjoint x*) **using** *positive-close-under-left-right-mult-adjoint r pdo wc* **unfolding** *partial-density-operator-def* **by** *auto*
moreover have *x* ∗ *ϱ* ∗ *adjoint x* ∈ *carrier-mat d d* **using** *r wc adjoint-dim index-mult-mat* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
qed
next
case (*Seq x1 x2a*)
then show *?case*
proof −
assume *dx1*: (⋀*ϱ*. *well-com x1* ⟹ *ϱ* ∈ *carrier-mat d d* ⟹ *partial-density-operator ϱ* ⟹ *positive* (*denote x1 ϱ*) ∧ *trace* (*denote x1 ϱ*) ≤ *trace ϱ* ∧ *denote x1 ϱ* ∈ *carrier-mat d d*)

and *dx2a*: (⋀ϱ. *well-com x2a* ⟹ ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *positive* (*denote x2a* ϱ) ∧ *trace* (*denote x2a* ϱ) ≤ *trace* ϱ ∧ *denote x2a* ϱ ∈ *carrier-mat d d*)
and *wc*: *well-com* (*Seq x1 x2a*) **and** *r*: ϱ ∈ *carrier-mat d d* **and** *pdo*: *partial-density-operator* ϱ
show *positive* (*denote* (*Seq x1 x2a*) ϱ) ∧ *trace* (*denote* (*Seq x1 x2a*) ϱ) ≤ *trace* ϱ ∧ *denote* (*Seq x1 x2a*) ϱ ∈ *carrier-mat d d*
proof −
have *ptc*: *positive* (*denote x1* ϱ) ∧ *trace* (*denote x1* ϱ) ≤ *trace* ϱ ∧ *denote x1* ϱ ∈ *carrier-mat d d*
using *wc r pdo dx1* **by** *auto*
then have *partial-density-operator* (*denote x1* ϱ) **using** *pdo* **unfolding** *partial-density-operator-def* **by** *auto*
then show *?thesis* **using** *ptc dx2a wc dual-order.trans* **by** *auto*
qed
qed
next
case (*Measure x1 x2a x3a*)
then show *?case* **using** *denote-measure-positive-trace-dim* **by** *auto*
next
case (*While x1 x2a*)
then show *?case*
proof −
have *adjoint* (*x1 0*) ∗ (*x1 0*) + *adjoint* (*x1 1*) ∗ (*x1 1*) = 1_m *d*
using *measurement-id2 While* **by** *auto*
moreover have (⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *positive* (*denote x2a* ϱ) ∧ *trace* (*denote x2a* ϱ) ≤ *trace* ϱ ∧ *denote x2a* ϱ ∈ *carrier-mat d d*)
using *While* **by** *fastforce*
moreover have *x1 0* ∈ *carrier-mat d d* ∧ *x1 1* ∈ *carrier-mat d d*
using *measurement-dim While* **by** *fastforce*
ultimately have *denote-while* (*x1 0*) (*x1 1*) (*denote x2a*) ϱ ∈ *carrier-mat d d* ∧
positive (*denote-while* (*x1 0*) (*x1 1*) (*denote x2a*) ϱ) ∧
trace (*denote-while* (*x1 0*) (*x1 1*) (*denote x2a*) ϱ) ≤ *trace* ϱ
using *denote-while-dim-positive*[*of x1 0 x1 1 denote x2a* ϱ] *While* **by** *fastforce*
then show *?thesis* **by** *auto*
qed
qed

lemma *denote-dim-pdo*:
well-com S ⟹ ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ
⟹ (*denote S* ϱ ∈ *carrier-mat d d*) ∧ (*partial-density-operator* (*denote S* ϱ))
using *denote-positive-trace-dim* **unfolding** *partial-density-operator-def* **by** *fastforce*

lemma *denote-dim*:
well-com S ⟹ ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ
⟹ (*denote S* ϱ ∈ *carrier-mat d d*)

using *denote-positive-trace-dim* **by** *auto*

lemma *denote-trace*:
well-com S $\Longrightarrow$ $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ
$\Longrightarrow$ *trace* (*denote S* ϱ) $\leq$ *trace* ϱ
using *denote-positive-trace-dim* **by** *auto*

lemma *denote-partial-density-operator*:
assumes *well-com S partial-density-operator* ϱ $\varrho \in$ *carrier-mat d d*
shows *partial-density-operator* (*denote S* ϱ)
using *assms denote-positive-trace-dim* **unfolding** *partial-density-operator-def*
using *dual-order.trans* **by** *blast*

lemma *denote-while-n-sum-mat-seq*:
assumes $\varrho \in$ *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 −
let *?A* = *x1 0* **and** *?B* = *x1 1*
have *dx2*:$\bigwedge\varrho$. $\varrho \in$ *carrier-mat d d* $\Longrightarrow$ *partial-density-operator* ϱ $\Longrightarrow$
positive ((*denote x2*) ϱ) $\wedge$ *trace* ((*denote x2*) ϱ) $\leq$ *trace* ϱ $\wedge$ (*denote x2*) $\varrho \in$ *carrier-mat d d*
using *denote-positive-trace-dim wc* **by** *auto*
have *lo1*: *adjoint ?A* $*$ *?A* + *adjoint ?B* $*$ *?B* = 1_m *d* **using** *measurement-id2 assms* **by** *auto*
have $\forall$ *n*. *matrix-sum d* (λn. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n* $\in$ *carrier-mat d d*
using *assms dx2*
by (*metis denote-while-n-dim matrix-sum-dim*)
moreover have ($\forall$ *n*. *partial-density-operator* (*matrix-sum d* (λn. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n*))
using *assms dx2 lo1*
by (*metis denote-while-n-sum-partial-density*)
moreover have ($\forall$ *n*. *matrix-sum d* (λn. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n* $\leq_L$ *matrix-sum d* (λn. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) (*Suc n*))
using *assms dx2 lo1*
by (*metis denote-while-n-sum-lowner-le*)
ultimately show *?thesis*
unfolding *matrix-seq-def* **by** *auto*
qed

lemma *denote-while-n-add*:
assumes *M0*: *x1 0* $\in$ *carrier-mat d d* **and**

M1: *x1 1* ∈ *carrier-mat d d* **and**
wc: *well-com x2* **and** *mea*: *measurement d 2 x1* **and**
DS: (⋀ϱ_1 ϱ_2. ϱ_1 ∈ *carrier-mat d d* ⟹ ϱ_2 ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ_1 ⟹
partial-density-operator ϱ_2 ⟹ *trace* (ϱ_1 + ϱ_2) ≤ *1* ⟹ *denote x2* (ϱ_1 + ϱ_2) = *denote x2* ϱ_1 + *denote x2* ϱ_2)
shows ϱ_1 ∈ *carrier-mat d d* ⟹ ϱ_2 ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ_1 ⟹ *partial-density-operator* ϱ_2 ⟹ *trace* (ϱ_1 + ϱ_2) ≤ *1* ⟹
denote-while-n (*x1 0*) (*x1 1*) (*denote x2*) *k* (ϱ_1 + ϱ_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 (*auto*, *induct k arbitrary*: ϱ_1 ϱ_2)
case *0*
then show *?case*
apply *auto* **using** *M0* **by** (*mat-assoc d*)
next
case (*Suc k*)
then show *?case*
proof −
let *?A* = *x1 0* **and** *?B* = *x1 1*
have *dx2*:(⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *positive* ((*denote x2*) ϱ) ∧ *trace* ((*denote x2*) ϱ) ≤ *trace* ϱ ∧ (*denote x2*) ϱ ∈ *carrier-mat d d*)
using *denote-positive-trace-dim wc* **by** *auto*
have *lo1*: *adjoint ?B* ∗ *?B* $\leq_L$ 1_m *d* **using** *measurement-le-one-mat assms* **by** *auto*
have *dim1*: *x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*) ∈ *carrier-mat d d* **using** *assms Suc*
by (*metis adjoint-dim mult-carrier-mat*)
moreover have *pdo1*: *partial-density-operator* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*))
using *pdo-close-under-measurement assms*(*2*) *Suc*(*2*,*4*) *lo1* **by** *auto*
ultimately have *dimr1*: *denote x2* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) ∈ *carrier-mat d d*
using *dx2* **by** *auto*
have *dim2*: *x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*) ∈ *carrier-mat d d* **using** *assms Suc*
by (*metis adjoint-dim mult-carrier-mat*)
moreover have *pdo2*: *partial-density-operator* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))
using *pdo-close-under-measurement assms*(*2*) *Suc lo1* **by** *auto*
ultimately have *dimr2*: *denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*)) ∈ *carrier-mat d d*
using *dx2* **by** *auto*
have *pdor1*: *partial-density-operator* (*denote x2* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)))
using *denote-partial-density-operator assms dim1 pdo1* **by** *auto*
have *pdor2*: *partial-density-operator* (*denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*)))
using *denote-partial-density-operator assms dim2 pdo2* **by** *auto*
have *trace* (*denote x2* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*))) ≤ *trace* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*))
using *dx2 dim1 pdo1* **by** *auto*
also have *tr1*: . . . ≤ *trace* ϱ_1 **using** *trace-decrease-mul-adj assms Suc lo1* **by** *auto*
finally have *trr1*: *trace* (*denote x2* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*))) ≤ *trace* ϱ_1

by *auto*
  have *trace* (*denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))) ≤ *trace* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))
    using *dx2 dim2 pdo2* **by** *auto*
  also have *tr2*: ... ≤ *trace* ϱ_2 **using** *trace-decrease-mul-adj assms Suc lo1* **by** *auto*
  finally have *trr2*: *trace* (*denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))) ≤ *trace* ϱ_2 **by** *auto*
  from *tr1 tr2 Suc* **have** *trace* ((*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) + (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))) ≤ *trace* (ϱ_1 + ϱ_2)
    using *trace-add-linear trace-add-linear*[*of* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) *d* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))]
      trace-add-linear[*of* ϱ_1 *d* ϱ_2]
    using *dim1 dim2* **by** (*auto simp*: *less-eq-complex-def*)
  then have *trless1*: *trace* ((*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) + (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))) ≤ *1* **using** *Suc* **by** *auto*
  from *trr1 trr2 Suc* **have** *trace* (*denote x2* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) + *denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))) ≤ *trace* (ϱ_1 + ϱ_2)
    using *trace-add-linear*[*of denote x2* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) *d denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))]
      trace-add-linear[*of* ϱ_1 *d* ϱ_2]
    using *dimr1 dimr2* **by** (*auto simp*: *less-eq-complex-def*)
  then have *trless2*: *trace* (*denote x2* (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) + *denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))) ≤ *1*
    using *Suc* **by** *auto*
  have *x1 1* ∗ (ϱ_1 + ϱ_2) ∗ *adjoint* (*x1 1*) = (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) + (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))
    using *M1 Suc* **by** (*mat-assoc d*)
  then have *deadd*: *denote x2* (*x1 1* ∗ (ϱ_1 + ϱ_2) ∗ *adjoint* (*x1 1*)) =
    denote x2 (*x1 1* ∗ ϱ_1 ∗ *adjoint* (*x1 1*)) + *denote x2* (*x1 1* ∗ ϱ_2 ∗ *adjoint* (*x1 1*))
    using *assms*(*5*) *dim1 dim2 pdo1 pdo2 trless1* **by** *auto*
  from *dimr1 dimr2 pdor1 pdor2 trless2 Suc*(*1*) *deadd* **show** *?thesis* **by** *auto*
 qed
qed

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

denote-while (*x1 0*) (*x1 1*) (*denote x2*) (ϱ_1 + ϱ_2) = *denote-while* (*x1 0*) (*x1 1*) (*denote x2*) ϱ_1 + *denote-while* (*x1 0*) (*x1 1*) (*denote x2*) ϱ_2
proof −
let *?A* = *x1 0* **and** *?B* = *x1 1*
have *dx2*:($\bigwedge\varrho$. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *positive* ((*denote x2*) ϱ) ∧ *trace* ((*denote x2*) ϱ) ≤ *trace* ϱ ∧ (*denote x2*) ϱ ∈ *carrier-mat d d*)
using *denote-positive-trace-dim wc* **by** *auto*
have *lo1*: *adjoint ?A* ∗ *?A* + *adjoint ?B* ∗ *?B* = 1_m *d* **using** *measurement-id2 assms* **by** *auto*
have *pdo12*: *partial-density-operator* (ϱ_1 + ϱ_2) **using** *pdo1 pdo2* **unfolding** *partial-density-operator-def* **using** *tr12 positive-add assms* **by** *auto*
have *ms1*: *matrix-seq d* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1))
using *denote-while-n-sum-mat-seq assms* **by** *auto*
have *ms2*: *matrix-seq d* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_2))
using *denote-while-n-sum-mat-seq assms* **by** *auto*
have *dim1*: (∀ *n*. *matrix-sum d* (λ*n*. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ_1) *n* ∈ *carrier-mat d d*)
using *assms dx2*
by (*metis denote-while-n-dim matrix-sum-dim*)
have *dim2*: (∀ *n*. *matrix-sum d* (λ*n*. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ_2) *n* ∈ *carrier-mat d d*)
using *assms dx2*
by (*metis denote-while-n-dim matrix-sum-dim*)
have *trace* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1) *n* +
matrix-sum d (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_2) *n*) ≤ *trace* (ϱ_1 + ϱ_2)
for *n*
proof −
have *trace* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1) *n*) ≤ *trace* ϱ_1
using *denote-while-n-sum-trace dx2 lo1 assms* **by** *auto*
moreover have *trace* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_2) *n*) ≤ *trace* ϱ_2
using *denote-while-n-sum-trace dx2 lo1 assms* **by** *auto*
ultimately show *?thesis*
using *trace-add-linear dim1 dim2*
by (*metis add-mono-thms-linordered-semiring*(*1*) *r1 r2*)
qed
then have ∀ *n*. *trace* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1) *n* + *matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_2) *n*) ≤ *1*
using *assms*(*7*) *dual-order.trans* **by** *blast*
then have *lladd*: *matrix-seq.lowner-lub* (λ*n*. (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1)) *n* + (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_2)) *n*) = *matrix-seq.lowner-lub* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1))
+ *matrix-seq.lowner-lub* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*)

n ϱ_2))
  using *lowner-lub-add ms1 ms2* **by** *auto*

 have *matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* (ϱ_1 + ϱ_2)) *m* =
  matrix-sum d (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1) *m* + *matrix-sum d* (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ_2) *m*
  for *m*
 proof −
  have (⋀*k. k* < *m* ⟹ *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *k* (ϱ_1 + ϱ_2) ∈ *carrier-mat d d*)
   using *denote-while-n-dim dx2 pdo12 assms measurement-dim* **by** *auto*
  moreover have (⋀*k. k* < *m* ⟹ *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *k* ϱ_1 ∈ *carrier-mat d d*)
   using *denote-while-n-dim dx2 assms measurement-dim* **by** *auto*
  moreover have (⋀*k. k* < *m* ⟹ *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *k* ϱ_2 ∈ *carrier-mat d d*)
   using *denote-while-n-dim dx2 assms measurement-dim* **by** *auto*
  moreover have (∀ *k* < *m*.
   denote-while-n (*x1 0*) (*x1 1*) (*denote x2*) *k* (ϱ_1 + ϱ_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)
   using *denote-while-n-add assms* **by** *auto*
  ultimately show *?thesis*
   using *matrix-sum-add*[*of m* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* (ϱ_1 + ϱ_2)) *d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ_1)
    (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ_2)] **by** *auto*
 qed
 then have *matrix-seq.lowner-lub* (*matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* (ϱ_1 + ϱ_2))) =
  matrix-seq.lowner-lub (λ*n.* (*matrix-sum d* (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ_1)) *n* + (*matrix-sum d* (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ_2)) *n*)
  using *lladd* **by** *presburger*
 then show *?thesis* **unfolding** *denote-while-def matrix-inf-sum-def* **using** *lladd* **by** *auto*
qed

lemma *denote-add*:
 well-com S ⟹ ϱ_1 ∈ *carrier-mat d d* ⟹ ϱ_2 ∈ *carrier-mat d d* ⟹
  partial-density-operator ϱ_1 ⟹ *partial-density-operator* ϱ_2 ⟹ *trace* (ϱ_1 + ϱ_2) ≤ *1* ⟹
  denote S (ϱ_1 + ϱ_2) = *denote S* ϱ_1 + *denote S* ϱ_2
proof (*induction arbitrary*: ϱ_1 ϱ_2)
 case *SKIP*
 then show *?case* **by** *auto*
next
 case (*Utrans U*)
 then show *?case* **by** (*clarsimp*, *mat-assoc d*)
next
 case (*Seq x1 x2a*)

then show *?case*
proof −
have *dim1*: *denote x1* ϱ_1 ∈ *carrier-mat d d* **using** *denote-positive-trace-dim Seq* **by** *auto*
have *dim2*: *denote x1* ϱ_2 ∈ *carrier-mat d d* **using** *denote-positive-trace-dim Seq* **by** *auto*
have *trace* (*denote x1* ϱ_1) ≤ *trace* ϱ_1 **using** *denote-positive-trace-dim Seq* **by** *auto*
moreover have *trace* (*denote x1* ϱ_2) ≤ *trace* ϱ_2 **using** *denote-positive-trace-dim Seq* **by** *auto*
ultimately have *tr*: *trace* (*denote x1* ϱ_1 + *denote x1* ϱ_2) ≤ *1* **using** *Seq*(*4*,*5*,*8*) *trace-add-linear dim1 dim2*
by (*smt* (*verit*) *add-mono order-trans*)

have *denote* (*Seq x1 x2a*) (ϱ_1 + ϱ_2) = *denote x2a* (*denote x1* (ϱ_1 + ϱ_2)) **by** *auto*
moreover have *denote x1* (ϱ_1 + ϱ_2) = *denote x1* ϱ_1 + *denote x1* ϱ_2 **using** *Seq* **by** *auto*
moreover have *partial-density-operator* (*denote x1* ϱ_1) **using** *denote-partial-density-operator Seq* **by** *auto*
moreover have *partial-density-operator* (*denote x1* ϱ_2) **using** *denote-partial-density-operator Seq* **by** *auto*
ultimately show *?thesis* **using** *Seq dim1 dim2 tr* **by** *auto*
qed
next
case (*Measure x1 x2a x3a*)
then show *?case*
proof −
have *ptc*: ⋀*x3aa* ϱ. *x3aa* ∈ *set x3a* ⟹ *well-com x3aa* ⟹ ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ
⟹ *positive* (*denote x3aa* ϱ) ∧ *trace* (*denote x3aa* ϱ) ≤ *trace* ϱ ∧ *denote x3aa* ϱ ∈ *carrier-mat d d*
using *denote-positive-trace-dim Measure* **by** *auto*
then have *map*:⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ ∀ *k* < *x1*. *positive* ((*map denote x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*)))
∧ ((*map denote x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*))) ∈ *carrier-mat d d*
∧ *trace* ((*map denote x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*))) ≤ *trace* (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*))
using *Measure map-denote-positive-trace-dim* **by** *auto*

from *map* **have** *mapd1*: ⋀*k*. *k* < *x1* ⟹ (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ_1 ∗ *adjoint* (*x2a k*)) ∈ *carrier-mat d d*
using *Measure* **by** *auto*
from *map* **have** *mapd2*: ⋀*k*. *k* < *x1* ⟹ (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ_2 ∗ *adjoint* (*x2a k*)) ∈ *carrier-mat d d*
using *Measure* **by** *auto*
have *dim1*:⋀*k*. *k* < *x1* ⟹ *x2a k* ∗ ϱ_1 ∗ *adjoint* (*x2a k*) ∈ *carrier-mat d d*
using *well-com.simps*(*5*) *measurement-dim Measure* **by** *fastforce*

have *dim2*: $\bigwedge k.\ k < x1 \Longrightarrow x2a\ k * \varrho_2 * adjoint\ (x2a\ k) \in carrier\text{-}mat\ d\ d$
using *well-com.simps*(*5*) *measurement-dim Measure* **by** *fastforce*
have $\bigwedge k.\ k < x1 \Longrightarrow (x2a\ k * (\varrho_1 + \varrho_2) * adjoint\ (x2a\ k)) \in carrier\text{-}mat\ d\ d$
using *well-com.simps*(*5*) *measurement-dim Measure* **by** *fastforce*
have *lea*: $\bigwedge k.\ k < x1 \Longrightarrow adjoint\ (x2a\ k) * x2a\ k \leq_L 1_m\ d$ **using** *measurement-le-one-mat Measure* **by** *auto*
moreover have *dimx*: $\bigwedge k.\ k < x1 \Longrightarrow x2a\ k \in carrier\text{-}mat\ d\ d$ **using** *Measure measurement-dim* **by** *auto*
ultimately have *pdo12*:$\bigwedge k.\ k < x1 \Longrightarrow partial\text{-}density\text{-}operator\ (x2a\ k * \varrho_1 * adjoint\ (x2a\ k)) \wedge partial\text{-}density\text{-}operator\ (x2a\ k * \varrho_2 * adjoint\ (x2a\ k))$
using *pdo-close-under-measurement Measure measurement-dim* **by** *blast*

have *trless*: $trace\ (x2a\ k * \varrho_1 * adjoint\ (x2a\ k) + x2a\ k * \varrho_2 * adjoint\ (x2a\ k)) \leq 1$
if *k*: $k < x1$ **for** *k*
proof −
have $trace\ (x2a\ k * \varrho_1 * adjoint\ (x2a\ k)) \leq trace\ \varrho_1$ **using** *trace-decrease-mul-adj dimx Measure lea k* **by** *auto*
moreover have $trace\ (x2a\ k * \varrho_2 * adjoint\ (x2a\ k)) \leq trace\ \varrho_2$ **using** *trace-decrease-mul-adj dimx Measure lea k* **by** *auto*
ultimately have $trace\ (x2a\ k * \varrho_1 * adjoint\ (x2a\ k) + x2a\ k * \varrho_2 * adjoint\ (x2a\ k)) \leq trace\ (\varrho_1 + \varrho_2)$
using *trace-add-linear dim1 dim2 Measure k*
by (*metis add-mono-thms-linordered-semiring*(*1*))
then show *?thesis* **using** *Measure*(*7*) **by** *auto*
qed

have *dist*: $(x2a\ k * (\varrho_1 + \varrho_2) * adjoint\ (x2a\ k)) = (x2a\ k * \varrho_1 * adjoint\ (x2a\ k)) + (x2a\ k * \varrho_2 * adjoint\ (x2a\ k))$
if *k*: $k < x1$ **for** *k*
proof −
have $(x2a\ k * (\varrho_1 + \varrho_2) * adjoint\ (x2a\ k)) = ((x2a\ k * \varrho_1 + x2a\ k * \varrho_2) * adjoint\ (x2a\ k))$
using *mult-add-distrib-mat Measure well-com.simps*(*4*) *measurement-dim* **by** (*metis k*)
also have $\ldots = (x2a\ k * \varrho_1 * adjoint\ (x2a\ k)) + (x2a\ k * \varrho_2 * adjoint\ (x2a\ k))$
apply (*mat-assoc d*) **using** *Measure k well-com.simps*(*4*) *measurement-dim* **by** *auto*
finally show *?thesis* **by** *auto*
qed

have *mapadd*: $(map\ denote\ x3a\ !\ k)\ (x2a\ k * (\varrho_1 + \varrho_2) * adjoint\ (x2a\ k)) =$
$(map\ denote\ x3a\ !\ k)\ (x2a\ k * \varrho_1 * adjoint\ (x2a\ k)) + (map\ denote\ x3a\ !\ k)\ (x2a\ k * \varrho_2 * adjoint\ (x2a\ k))$
if *k*: $k < x1$ **for** *k*
proof −
have $(map\ denote\ x3a\ !\ k)\ (x2a\ k * (\varrho_1 + \varrho_2) * adjoint\ (x2a\ k)) = denote\ (x3a\ !\ k)\ (x2a\ k * (\varrho_1 + \varrho_2) * adjoint\ (x2a\ k))$

using *Measure.prems*(*1*) *k* **by** *auto*
then have *mapx*: (*map denote x3a* ! *k*) (*x2a k* ∗ (ϱ_1 + ϱ_2) ∗ *adjoint* (*x2a k*)) = *denote* (*x3a* ! *k*) ((*x2a k* ∗ ϱ_1 ∗ *adjoint* (*x2a k*)) + (*x2a k* ∗ ϱ_2 ∗ *adjoint* (*x2a k*)))
using *dist k* **by** *auto*
have *denote* (*x3a* ! *k*) ((*x2a k* ∗ ϱ_1 ∗ *adjoint* (*x2a k*)) + (*x2a k* ∗ ϱ_2 ∗ *adjoint* (*x2a k*)))
= *denote* (*x3a* ! *k*) (*x2a k* ∗ ϱ_1 ∗ *adjoint* (*x2a k*)) + *denote* (*x3a* ! *k*) (*x2a k* ∗ ϱ_2 ∗ *adjoint* (*x2a k*))
using *Measure*(*1*,*2*) *dim1 dim2 pdo12 trless k*
by (*simp add*: *list-all-length*)
then show *?thesis*
using *Measure.prems*(*1*) *mapx k* **by** *auto*
qed
then have *mapd12*:(⋀*k*. *k* < *x1* ⟹ (*map denote x3a* ! *k*) (*x2a k* ∗ (ϱ_1 + ϱ_2) ∗ *adjoint* (*x2a k*)) ∈ *carrier-mat d d*)
using *mapd1 mapd2* **by** *auto*

have *matrix-sum d* (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ (ϱ_1 + ϱ_2) ∗ *adjoint* (*x2a k*))) *x1* =
matrix-sum d (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ_1 ∗ *adjoint* (*x2a k*))) *x1* +
matrix-sum d (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ_2 ∗ *adjoint* (*x2a k*))) *x1*
using *matrix-sum-add*[*of x1* (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ (ϱ_1 + ϱ_2) ∗ *adjoint* (*x2a k*))) *d* (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ_1 ∗ *adjoint* (*x2a k*))) (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ_2 ∗ *adjoint* (*x2a k*)))]
using *mapd12 mapd1 mapd2 mapadd* **by** *auto*
then show *?thesis* **using** *denote.simps*(*4*) **unfolding** *denote-measure-def* **by** *auto*
qed
next
case (*While x1 x2*)
then show *?case*
apply *auto* **using** *denote-while-add measurement-dim* **by** *auto*
qed

lemma *mulfact*:
fixes *c*:: *real* **and** *a*:: *complex* **and** *b*:: *complex*
assumes *c*≥*0 a* ≤ *b*
shows *c* ∗ *a* ≤ *c* ∗ *b*
using *assms mult-le-cancel-left-pos* **unfolding** *less-eq-complex-def* **by** *force*

lemma *denote-while-n-scale*:
fixes *c*:: *real*
assumes *c*≥*0*
measurement d 2 x1 well-com x2
(⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *trace* (*c* $\cdot_m$ ϱ) ≤ *1* ⟹

denote x2 (*c* $\cdot_m$ ϱ) = *c* $\cdot_m$ *denote x2* ϱ)
shows ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *trace* (*c* $\cdot_m$ ϱ) ≤ *1* ⟹
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 (*auto*, *induct n arbitrary*: ϱ)
case *0*
then show *?case*
apply *auto* **apply** (*mat-assoc d*) **using** *assms measurement-dim* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof −
let *?A* = *x1 0* **and** *?B* = *x1 1*
have *dx2*:(⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *positive* ((*denote x2*) ϱ) ∧ *trace* ((*denote x2*) ϱ) ≤ *trace* ϱ ∧ (*denote x2*) ϱ ∈ *carrier-mat d d*)
using *denote-positive-trace-dim assms* **by** *auto*
have *lo1*: *adjoint ?B* ∗ *?B* $\leq_L$ 1_m *d* **using** *measurement-le-one-mat assms* **by** *auto*
have *dim1*: *x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*) ∈ *carrier-mat d d* **using** *assms*(*2*) *Suc*(*2*) *measurement-dim*
by (*meson adjoint-dim mult-carrier-mat one-less-numeral-iff semiring-norm*(*76*))
moreover have *pdo1*: *partial-density-operator* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))
using *pdo-close-under-measurement assms Suc lo1 measurement-dim*
by (*metis One-nat-def lessI numeral-2-eq-2*)
ultimately have *dimr*: *denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*)) ∈ *carrier-mat d d*
using *dx2* **by** *auto*
have *pdor*: *partial-density-operator* (*denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*)))
using *denote-partial-density-operator assms dim1 pdo1* **by** *auto*
have *trace* (*denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) ≤ *trace* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))
using *dx2 dim1 pdo1* **by** *auto*
also have *trr1*: ... ≤ *trace* ϱ **using** *trace-decrease-mul-adj assms Suc lo1 measurement-dim* **by** *auto*
finally have *trr*: *trace* (*denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) ≤ *trace* ϱ **by** *auto*
moreover have *trace* (*c* $\cdot_m$ *denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) = *c* ∗ *trace* (*denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*)))
using *trace-smult dimr* **by** *auto*
moreover have *trcr*: *trace* (*c* $\cdot_m$ ϱ) = *c* ∗ *trace* ϱ **using** *trace-smult Suc* **by** *auto*
ultimately have *trace* (*c* $\cdot_m$ *denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) ≤ *trace* (*c* $\cdot_m$ ϱ)
using *assms*(*1*) *state-sig.mulfact* **by** *auto*
then have *trrc*: *trace* (*c* $\cdot_m$ *denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) ≤ *1* **using** *Suc* **by** *auto*
have *trace* (*c* $\cdot_m$ (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) = *c* ∗ *trace* (*x1 1* ∗ ϱ ∗ *adjoint*

(*x1 1*))
using *trace-smult dim1* **by** *auto*
then have *trace* ($c \cdot_m$ (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) ≤ *trace* ($c \cdot_m$ ϱ) **using** *trcr trr1 assms*(*1*)
using *state-sig.mulfact* **by** *auto*
then have *trrle*: *trace* ($c \cdot_m$ (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) ≤ *1* **using** *Suc* **by** *auto*
have *x1 1* ∗ (*complex-of-real c* $\cdot_m$ ϱ) ∗ *adjoint* (*x1 1*) = *complex-of-real c* $\cdot_m$ (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))
apply (*mat-assoc d*) **using** *Suc.prems*(*1*) *assms measurement-dim* **by** *auto*
then have *denote x2* (*x1 1* ∗ (*complex-of-real c* $\cdot_m$ ϱ) ∗ *adjoint* (*x1 1*)) = (*denote x2* ($c \cdot_m$ (*x1 1* ∗ (ϱ) ∗ *adjoint* (*x1 1*))))
by *auto*
moreover have *denote x2* ($c \cdot_m$ (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))) = $c \cdot_m$ *denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))
using *assms*(*4*) *dim1 pdo1 trrle* **by** *auto*
ultimately have *denote x2* (*x1 1* ∗ (*complex-of-real c* $\cdot_m$ ϱ) ∗ *adjoint* (*x1 1*)) = $c \cdot_m$ *denote x2* (*x1 1* ∗ ϱ ∗ *adjoint* (*x1 1*))
using *assms* **by** *auto*
then show *?thesis* **using** *Suc dimr pdor trrc* **by** *auto*
qed
qed

lemma *denote-while-scale*:
fixes *c*:: *real*
assumes ϱ ∈ *carrier-mat d d*
partial-density-operator ϱ
trace ($c \cdot_m$ ϱ) ≤ *1* *c* ≥ *0*
measurement d 2 x1 well-com x2
(⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *trace* ($c \cdot_m$ ϱ) ≤ *1* ⟹
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 −
let *?A* = *x1 0* **and** *?B* = *x1 1*
have *dx2*:(⋀ϱ. ϱ ∈ *carrier-mat d d* ⟹ *partial-density-operator* ϱ ⟹ *positive* ((*denote x2*) ϱ) ∧ *trace* ((*denote x2*) ϱ) ≤ *trace* ϱ ∧ (*denote x2*) ϱ ∈ *carrier-mat d d*)
using *denote-positive-trace-dim assms* **by** *auto*
have *lo1*: *adjoint ?A* ∗ *?A* + *adjoint ?B* ∗ *?B* = 1_m *d* **using** *measurement-id2 assms* **by** *auto*
have *ms*: *matrix-seq d* (*matrix-sum d* (λ*n*. *denote-while-n ?A ?B* (*denote x2*) *n* ϱ))
using *denote-while-n-sum-mat-seq assms measurement-dim* **by** *auto*

have *trcless*: *trace* ($c \cdot_m$ *matrix-sum d* (λ*n*. *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n*) ≤ *1* **for** *n*
proof −

have *dimr*: *matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n* ∈ *carrier-mat d d*
using *assms dx2 denote-while-n-dim matrix-sum-dim*
using *matrix-seq.dim ms* **by** *auto*
have *trace* (*matrix-sum d* (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ) *n*) ≤ *trace* ϱ
using *denote-while-n-sum-trace dx2 lo1 assms measurement-dim* **by** *auto*
moreover have *trace* ($c \cdot_m$ *matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n*) = *c* ∗ *trace* (*matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n*)
using *trace-smult dimr* **by** *auto*
moreover have *trace* ($c \cdot_m$ ϱ) = *c* ∗ *trace* ϱ **using** *trace-smult assms* **by** *auto*

ultimately have *trace* ($c \cdot_m$ *matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ) *n*) ≤ *trace* ($c \cdot_m$ ϱ)
using *assms*(*4*) **by** (*simp add*: *ordered-comm-semiring-class.comm-mult-left-mono less-eq-complex-def*)
then show *?thesis*
using *assms* **by** *auto*
qed

have *llscale*: *matrix-seq.lowner-lub* (λ*n.* $c \cdot_m$ (*matrix-sum d* (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ)) *n*)
= $c \cdot_m$ *matrix-seq.lowner-lub* (*matrix-sum d* (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ))
using *lowner-lub-scale*[*of d* (*matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ)) *c*] *ms trcless assms*(*4*) **by** *auto*
have *matrix-sum d* (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* (*complex-of-real* $c \cdot_m$ ϱ)) *m*
= $c \cdot_m$ (*matrix-sum d* (λ*n. denote-while-n ?A ?B* (*denote x2*) *n* ϱ)) *m*
for *m*
proof −
have *dim*:(⋀*k. k* < *m* ⟹ *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *k* ϱ ∈ *carrier-mat d d*)
using *denote-while-n-dim dx2 assms measurement-dim* **by** *auto*
then have *dimr*: (⋀*k. k* < *m* ⟹ $c \cdot_m$ *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *k* ϱ ∈ *carrier-mat d d*)
using *smult-carrier-mat* **by** *auto*
have ∀ *n*<*m. 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)
using *denote-while-n-scale assms* **by** *auto*
then have (*matrix-sum d* (λ*n.* $c \cdot_m$ *denote-while-n ?A ?B* (*denote x2*) *n* ϱ)) *m* =
matrix-sum d (λ*n. denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* (*complex-of-real* $c \cdot_m$ ϱ)) *m*
using *matrix-sum-cong*[*of m* λ*n. complex-of-real* $c \cdot_m$ *denote-while-n* (*x1 0*) (*x1 1*) (*denote x2*) *n* ϱ] *dimr*
by *fastforce*
moreover have (*matrix-sum d* (λ*n.* $c \cdot_m$ *denote-while-n ?A ?B* (*denote x2*) *n*

ϱ)) m = c ·m (matrix-sum d (λn. denote-while-n ?A ?B (denote x2) n ϱ)) m
using *matrix-sum-smult[of m (λn. denote-while-n (x1 0) (x1 1) (denote x2) n ϱ) d c] dim* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
then have *matrix-seq.lowner-lub (matrix-sum d (λn. denote-while-n (x1 0) (x1 1) (denote x2) n (complex-of-real c ·m ϱ))) =*
matrix-seq.lowner-lub (λn. c ·m (matrix-sum d (λn. denote-while-n ?A ?B (denote x2) n ϱ)) n)
by *meson*
then show *?thesis*
unfolding *denote-while-def matrix-inf-sum-def* **using** *llscale* **by** *auto*
qed

lemma *denote-scale*:
fixes *c :: real*
assumes *c≥0*
shows *well-com S ⟹ ϱ ∈ carrier-mat d d ⟹ partial-density-operator ϱ ⟹ trace (c ·m ϱ) ≤ 1 ⟹ denote S (c ·m ϱ) = c ·m denote S ϱ*
proof *(induction arbitrary: ϱ)*
case *SKIP*
then show *?case* **by** *auto*
next
case *(Utrans x)*
then show *?case*
unfolding *denote.simps* **apply** *(mat-assoc d)* **using** *Utrans* **by** *auto*
next
case *(Seq x1 x2a)*
then show *?case*
proof −
have *cd: denote x1 (c ·m ϱ) = c ·m denote x1 ϱ* **using** *Seq* **by** *auto*
have *x1: denote x1 ϱ ∈ carrier-mat d d ∧ partial-density-operator (denote x1 ϱ) ∧ trace (denote x1 ϱ) ≤ trace ϱ*
using *denote-positive-trace-dim Seq denote-partial-density-operator* **by** *auto*
have *trace (c ·m denote x1 ϱ) = c ∗ trace (denote x1 ϱ)* **using** *trace-smult x1* **by** *auto*
also have ... *≤ c ∗ trace ϱ* **using** *x1 assms*
by *(metis Seq.prems cd denote-positive-trace-dim partial-density-operator-def positive-scale smult-carrier-mat trace-smult well-com.simps(3))*
also have ... *≤ 1* **using** *Seq(6) trace-smult Seq(4)*
by *(simp add: trace-smult)*
finally have *trace (c ·m denote x1 ϱ) ≤1* **by** *auto*
then have *denote x2a (c ·m denote x1 ϱ) = c ·m denote x2a (denote x1 ϱ)* **using** *x1 Seq* **by** *auto*
then show *?thesis* **using** *denote.simps(4) cd* **by** *auto*
qed
next
case *(Measure x1 x2a x3a)*
then show *?case*

proof −
have *ptc*: $\bigwedge$*x3aa* ϱ. *x3aa* $\in$ *set x3a* $\Longrightarrow$ *well-com x3aa* $\Longrightarrow$ ϱ $\in$ *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*
using *denote-positive-trace-dim Measure* **by** *auto*
have *cad*: *x2a k* $*$ (*c* $\cdot_m$ ϱ) $*$ *adjoint* (*x2a k*) = *c* $\cdot_m$ (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*))
if *k*: *k* < *x1* **for** *k*
apply (*mat-assoc d*) **using** *well-com.simps Measure measurement-dim k* **by** *auto*
have $\forall$ *k*<*x1*. *x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*) $\in$ *carrier-mat d d*
using *Measure*(*2*) *measurement-dim Measure*(*3*) **by** *fastforce*
have *lea*: $\forall$ *k*<*x1*. *adjoint* (*x2a k*) $*$ *x2a k* $\leq_L$ 1_m *d* **using** *measurement-le-one-mat Measure*(*2*) **by** *auto*
then have *pdox*: $\forall$ *k*<*x1*. *partial-density-operator* (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*))
using *pdo-close-under-measurement Measure*(*2*,*3*,*4*) *measurement-dim*
by (*meson state-sig.well-com.simps*(*4*))
have *x2aa*:$\forall$ *k* < *x1*. (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*)) $\in$ *carrier-mat d d* **using** *Measure*(*2*,*3*) *measurement-dim* **by** *fastforce*
have *dimm*: ($\bigwedge$*k*. *k* < *x1* $\Longrightarrow$ (*map denote x3a* ! *k*) (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*)) $\in$ *carrier-mat d d*)
using *map-denote-positive-trace-dim Measure*(*2*,*3*,*4*) *ptc* **by** *auto*
then have *dimcm*: ($\bigwedge$*k*. *k* < *x1* $\Longrightarrow$ *c* $\cdot_m$ (*map denote x3a* ! *k*) (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*)) $\in$ *carrier-mat d d*)
using *smult-carrier-mat* **by** *auto*

have *tra*: $\forall$ *k* < *x1*. *trace* ((*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*))) $\leq$ *trace* ϱ
using *trace-decrease-mul-adj Measure lea measurement-dim* **by** *auto*

have *tra1*: *trace* (*c* $\cdot_m$ (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*))) $\leq$ *1* **if** *k*: *k* < *x1* **for** *k*
proof −
have *trle*: *trace* (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*)) $\leq$ *trace* ϱ **using** *tra k* **by** *auto*
have *trace* (*c* $\cdot_m$ (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*))) = *c* $*$ *trace* ((*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*)))
using *trace-smult x2aa k* **by** *auto*
also have ... $\leq$ *c* $*$ *trace* ϱ
using *trle assms mulfact* **by** *auto*
also have ... $\leq$ *1* **using** *Measure*(*3*,*5*) *trace-smult* **by** *metis*
finally show *?thesis* **by** *auto*
qed

have (*map denote x3a* ! *k*) (*x2a k* $*$ (*c* $\cdot_m$ ϱ) $*$ *adjoint* (*x2a k*))
= *c* $\cdot_m$ (*map denote x3a* ! *k*) (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*)) **if** *k*: *k* < *x1* **for** *k*
proof −
have *denote* (*x3a* ! *k*) (*x2a k* $*$ (*c* $\cdot_m$ ϱ) $*$ *adjoint* (*x2a k*)) = *denote* (*x3a* ! *k*) (*c* $\cdot_m$ (*x2a k* $*$ ϱ $*$ *adjoint* (*x2a k*)))
using *cad k* **by** *auto*

also have . . . = *c* ·$_m$ *denote* (*x3a* ! *k*) ((*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*)))
using *Measure*(*1*,*2*) *pdox x2aa tra1 k* **using** *measure-well-com* **by** *auto*
finally have *denote* (*x3a* ! *k*) (*x2a k* ∗ (*complex-of-real c* ·$_m$ ϱ) ∗ *adjoint* (*x2a k*)) = *complex-of-real c* ·$_m$ *denote* (*x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*))
by *auto*
then show *?thesis* **using** *Measure.prems*(*1*) *k* **by** *auto*
qed

then have *matrix-sum d* (λ*k*. *c* ·$_m$ (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*))) *x1* =
matrix-sum d (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ (*c* ·$_m$ ϱ) ∗ *adjoint* (*x2a k*))) *x1*
using *matrix-sum-cong*[*of x1* (λ*k*. *complex-of-real c* ·$_m$ (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*)))
(λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ (*complex-of-real c* ·$_m$ ϱ) ∗ *adjoint* (*x2a k*)))] *dimcm* **by** *auto*
then have *matrix-sum d* (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ (*c* ·$_m$ ϱ) ∗ *adjoint* (*x2a k*))) *x1* =
c ·$_m$ *matrix-sum d* (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*))) *x1*
using *matrix-sum-smult*[*of x1* (λ*k*. (*map denote x3a* ! *k*) (*x2a k* ∗ ϱ ∗ *adjoint* (*x2a k*))) *d c*] *dimm* **by** *auto*
then have *denote* (*Measure x1 x2a x3a*) (*c* ·$_m$ ϱ) = *c* ·$_m$ *denote* (*Measure x1 x2a x3a*) ϱ
using *denote.simps*(*4*)[*of x1 x2a x3a c* ·$_m$ ϱ]
using *denote.simps*(*4*)[*of x1 x2a x3a* ϱ] **unfolding** *denote-measure-def* **by** *auto*
then show *?thesis* **by** *auto*
qed
next
case (*While x1 x2a*)
then show *?case*
apply *auto*
using *denote-while-scale assms* **by** *auto*
qed

lemma *limit-mat-denote-while-n*:
assumes *wc*: *well-com* (*While M S*) **and** *dr*: ϱ ∈ *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 −
have *m*: *measurement d 2 M* **using** *wc* **by** *auto*
then have *dM0*: *M 0* ∈ *carrier-mat d d* **and** *dM1*: *M 1* ∈ *carrier-mat d d* **and** *id*: *adjoint* (*M 0*) ∗ (*M 0*) + *adjoint* (*M 1*) ∗ (*M 1*) = 1_m *d*
using *measurement-id2 m measurement-def* **by** *auto*
have *wcs*: *well-com S* **using** *wc* **by** *auto*
have *DS*: *positive* (*denote S* ϱ) ∧ *trace* (*denote S* ϱ) ≤ *trace* ϱ ∧ *denote S* ϱ ∈ *carrier-mat d d*
if ϱ ∈ *carrier-mat d d* **and** *partial-density-operator* ϱ **for** ϱ

using *wcs that denote-positive-trace-dim* **by** *auto*

have *sumdd*: (∀ *n. matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*) *n* ∈ *carrier-mat d d*)
if *ϱ* ∈ *carrier-mat d d* **and** *partial-density-operator ϱ* **for** *ϱ*
using *denote-while-n-sum-dim dM0 dM1 DS that* **by** *auto*
have *sumtr*: ∀ *n. trace* (*matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*) *n*) ≤ *trace ϱ*
if *ϱ* ∈ *carrier-mat d d* **and** *partial-density-operator ϱ* **for** *ϱ*
using *denote-while-n-sum-trace*[*OF dM0 dM1 id DS*] *that* **by** *auto*
have *sumpar*: (∀ *n. partial-density-operator* (*matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*) *n*))
if *ϱ* ∈ *carrier-mat d d* **and** *partial-density-operator ϱ* **for** *ϱ*
using *denote-while-n-sum-partial-density*[*OF dM0 dM1 id DS*] *that* **by** *auto*
have *sumle*:(∀ *n. matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*) *n* $\leq_L$ *matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*) (*Suc n*))
if *ϱ* ∈ *carrier-mat d d* **and** *partial-density-operator ϱ* **for** *ϱ*
using *denote-while-n-sum-lowner-le*[*OF dM0 dM1 id DS*] *that* **by** *auto*
have *seqd*: *matrix-seq d* (*matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*))
if *ϱ* ∈ *carrier-mat d d* **and** *partial-density-operator ϱ* **for** *ϱ*
using *matrix-seq-def sumdd sumpar sumle that* **by** *auto*

have *matrix-seq.lowner-is-lub* (*matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*))
(*matrix-seq.lowner-lub* (*matrix-sum d* (*λn. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *n ϱ*)))
using *DS lowner-is-lub-matrix-sum dM0 dM1 id pdor dr* **by** *auto*
then show *limit-mat* (*matrix-sum d* (*λk. denote-while-n* (*M 0*) (*M 1*) (*denote S*) *k ϱ*)) (*denote* (*While M S*) *ϱ*) *d*
unfolding *denote.simps denote-while-def matrix-inf-sum-def* **using** *matrix-seq.lowner-lub-is-limit*[*OF seqd*[*OF dr pdor*]] **by** *auto*
qed

end

end

4 Partial state

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

lemma *nths-intersection-eq*:
assumes {*0..*<*length xs*} ⊆ *A*
shows *nths xs B* = *nths xs* (*A* ∩ *B*)
proof −
have ⋀*x. x* ∈ *set* (*zip xs* [*0..*<*length xs*]) ⟹ *snd x* < *length xs*

by (*metis atLeastLessThan-iff atLeastLessThan-upt in-set-zip nth-mem*)
then have $\bigwedge x.\ x \in set\ (zip\ xs\ [0..<length\ xs]) \Longrightarrow snd\ x \in A$ **using** *assms* **by** *auto*
then have *eqp*: $\bigwedge x.\ x \in set\ (zip\ xs\ [0..<length\ xs]) \Longrightarrow snd\ x \in B = (snd\ x \in (A \cap B))$ **by** *simp*
then have $filter\ (\lambda p.\ snd\ p \in B)\ (zip\ xs\ [0..<length\ xs]) = filter\ (\lambda p.\ snd\ p \in (A \cap B))\ (zip\ xs\ [0..<length\ xs])$
using $filter\text{-}cong[of\ (zip\ xs\ [0..<length\ xs])\ (zip\ xs\ [0..<length\ xs]),\ OF\ \text{-}\ eqp]$ **by** *auto*
then show $nths\ xs\ B = nths\ xs\ (A \cap B)$ **unfolding** *nths-def* **by** *auto*
qed

lemma *nths-minus-eq*:
assumes $\{0..<length\ xs\} \subseteq A$
shows $nths\ xs\ (-B) = nths\ xs\ (A - B)$
proof −
have $\bigwedge x.\ x \in set\ (zip\ xs\ [0..<length\ xs]) \Longrightarrow snd\ x < length\ xs$
by (*metis atLeastLessThan-iff atLeastLessThan-upt in-set-zip nth-mem*)
then have $\bigwedge x.\ x \in set\ (zip\ xs\ [0..<length\ xs]) \Longrightarrow snd\ x \in A$ **using** *assms* **by** *auto*
then have *eqp*: $\bigwedge x.\ x \in set\ (zip\ xs\ [0..<length\ xs]) \Longrightarrow snd\ x \in (-B) = (snd\ x \in (A - B))$ **by** *simp*
then have $filter\ (\lambda p.\ snd\ p \in (-B))\ (zip\ xs\ [0..<length\ xs]) = filter\ (\lambda p.\ snd\ p \in (A-B))\ (zip\ xs\ [0..<length\ xs])$
using $filter\text{-}cong[of\ (zip\ xs\ [0..<length\ xs])\ (zip\ xs\ [0..<length\ xs]),\ OF\ \text{-}\ eqp]$ **by** *auto*
then show $nths\ xs\ (-B) = nths\ xs\ (A - B)$ **unfolding** *nths-def* **by** *auto*
qed

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 −
have $nths\ xs\ (-B) = nths\ xs\ (A \cup B - B)$ **using** *nths-minus-eq assms* **by** *auto*
moreover have $A \cup B - B = A$ **using** *assms* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

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 −
have $x \notin \{y.\ y \in A \wedge y < x\}$ **by** *auto*
then have $\{y.\ y \in A \wedge y < x\} \subseteq A - \{x\}$ **by** *auto*
then have $card\ \{y.\ y \in A \wedge y < x\} \leq card\ (A - \{x\})$
using *card-mono finite-Diff*[*OF assms*(*1*)] **by** *auto*
also have $\ldots < card\ A$ **using** *card-Diff1-less*[*OF assms*] **by** *auto*

finally show *?thesis* **by** *auto*
qed

definition *ind-in-set* **where**
ind-in-set A x = *card* {*i*. *i* ∈ *A* ∧ *i* < *x*}

lemma *bij-ind-in-set-bound*:
fixes *M* :: *nat* **and** *v0* :: *nat set*
assumes ⋀*x*. *f x* = *card* {*y*. *y* ∈ *v0* ∧ *y* < *x*}
shows *bij-betw f* ({*0*..<*M*} ∩ *v0*) {*0*..<*card* ({*0*..<*M*} ∩ *v0*)}
unfolding *bij-betw-def*
proof
let *?dom* = {*0*..<*M*} ∩ *v0*
let *?ran* = {*0*..<*card* ({*0*..<*M*} ∩ *v0*)}
{
fix *x1 x2* :: *nat* **assume** *x1*: *x1* ∈ *?dom* **and** *x2*: *x2* ∈ *?dom* **and** *f x1* = *f x2*
then have *card* {*y*. *y* ∈ *v0* ∧ *y* < *x1*} = *card* {*y*. *y* ∈ *v0* ∧ *y* < *x2*} **using** *assms* **by** *auto*
then have *pick v0* (*card* {*y*. *y* ∈ *v0* ∧ *y* < *x1*}) = *pick v0* (*card* {*y*. *y* ∈ *v0* ∧ *y* < *x2*}) **by** *auto*
moreover have *pick v0* (*card* {*y*. *y* ∈ *v0* ∧ *y* < *x1*}) = *x1* **using** *pick-card-in-set x1* **by** *auto*
moreover have *pick v0* (*card* {*y*. *y* ∈ *v0* ∧ *y* < *x2*}) = *x2* **using** *pick-card-in-set x2* **by** *auto*
ultimately have *x1* = *x2* **by** *auto*
}
then show *inj-on f ?dom* **unfolding** *inj-on-def* **by** *auto*
{
fix *x* **assume** *x*: *x* ∈ *?dom*
then have (*y* ∈ *v0* ∧ *y* < *x*) = (*y* ∈ *?dom* ∧ *y* < *x*) **for** *y* **using** *x* **by** *auto*
then have *card* {*y*. *y* ∈ *v0* ∧ *y* < *x*} = *card* {*y*. *y* ∈ *?dom* ∧ *y* < *x*} **by** *auto*
moreover have *card* {*y*. *y* ∈ *?dom* ∧ *y* < *x*} < *card ?dom* **using** *x lt-set-card-lt*[*of ?dom*] **by** *auto*
ultimately have *f x* ∈ *?ran* **using** *assms* **by** *auto*
}
then have *sub*: (*f* ‘ *?dom*) ⊆ *?ran* **by** *auto*
{
fix *y* **assume** *y*: *y* ∈ *?ran*
then have *yle*: *y* < *card ?dom* **by** *auto*
then have *pyindom*: *pick ?dom y* ∈ *?dom* **using** *pick-in-set-le*[*of y ?dom*] **by** *auto*
then have *pick ?dom y* < *M* **by** *auto*
then have ⋀*z*. (*z* < *pick ?dom y* ⟹ *z* ∈ *v0* = (*z* ∈ *?dom*)) **by** *auto*
then have {*z*. *z* ∈ *v0* ∧ *z* < *pick ?dom y*} = {*z*. *z* ∈ *?dom* ∧ *z* < *pick ?dom y*} **by** *auto*
then have *card* {*z*. *z* ∈ *v0* ∧ *z* < *pick ?dom y*} = *card* {*z*. *z* ∈ *?dom* ∧ *z* < *pick ?dom y*} **by** *auto*
then have *f* (*pick ?dom y*) = *y* **using** *card-pick-le*[*OF yle*] *assms* **by** *auto*

with *pyindom* **have** $\exists x \in ?dom.\ f\ x = y$ **by** *auto*
}
then have *sup*: $?ran \subseteq (f\ `\ ?dom)$ **by** *fastforce*
show $(f\ `\ ?dom) = ?ran$ **using** *sub sup* **by** *auto*
qed

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 −
have $\{0..<M\} \cap A \subseteq \{i.\ i \in A \wedge i < N\}$ **using** *assms* **by** *auto*
then have *card* $(\{0..<M\} \cap A) \leq$ *card* $\{i.\ i \in A \wedge i < N\}$
using *card-mono*[*of* $\{i.\ i \in A \wedge i < N\}$] **by** *auto*
moreover have *ind-in-set* $A\ N =$ *card* $\{i.\ i \in A \wedge i < N\}$ **unfolding** *ind-in-set-def* **by** *auto*
ultimately have *ind-in-set* $A\ N \geq$ *card* $(\{0..<M\} \cap A)$ **by** *auto*

moreover have $y \in$ *ind-in-set* $A\ `\ (A \cap \{0..<M\}) \Longrightarrow y <$ *card* $(\{0..<M\} \cap A)$ **for** y
proof −
let $?dom = \{0..<M\} \cap A$
assume $y \in$ *ind-in-set* $A\ `\ (A \cap \{0..<M\})$
then obtain x **where** x: $x \in ?dom$ **and** y: *ind-in-set* $A\ x = y$ **by** *auto*
then have $(y \in A \wedge y < x) = (y \in ?dom \wedge y < x)$ **for** y **using** x **by** *auto*
then have *card* $\{y.\ y \in A \wedge y < x\} =$ *card* $\{y.\ y \in ?dom \wedge y < x\}$ **by** *auto*
moreover have *card* $\{y.\ y \in ?dom \wedge y < x\} <$ *card* $?dom$ **using** x *lt-set-card-lt*[*of* $?dom$] **by** *auto*
ultimately show $y <$ *card* $(\{0..<M\} \cap A)$ **using** y **unfolding** *ind-in-set-def* **by** *auto*
qed
ultimately show *?thesis* **by** *fastforce*
qed

lemma *bij-minus-subset*:
bij-betw $f\ A\ B \Longrightarrow C \subseteq A \Longrightarrow (f\ `\ A) - (f\ `\ C) = f\ `\ (A - C)$
by (*metis Diff-subset bij-betw-imp-inj-on bij-betw-imp-surj-on inj-on-image-set-diff*)

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* $A\ `\ (\{0..<M\} \cap A)) \cap ($*ind-in-set* $A\ `\ (A - B))$
proof −
let $?dom = \{0..<M\} \cap A$
let $?ran = \{0..<$*card* $(\{0..<M\} \cap A)\}$
let $?f =$ *ind-in-set* A
let $?C = A - B$
have *bij*: *bij-betw* $?f\ ?dom\ ?ran$

using *bij-ind-in-set-bound*[*of ?f A M*] **unfolding** *ind-in-set-def* **by** *auto*
then have *eq*: *?f ‘ ?dom = ?ran* **using** *bij-betw-imp-surj-on* **by** *fastforce*

have $(?f\ `\ B) = (?f\ `\ (\{0..<M\} \cap B)) \cup (?f\ `\ (\{n.\ n \geq M\} \cap B))$
by *fastforce*
then have $(?f\ `\ ?dom) - (?f\ `\ B)$
$= (?f\ `\ ?dom) - (?f\ `\ (\{n.\ n \geq M\} \cap B)) - (?f\ `\ (\{0..<M\} \cap B))$
by *fastforce*
moreover have $(?f\ `\ (\{n.\ n \geq M\} \cap B)) \cap (?f\ `\ ?dom) = \{\}$ **using** *ind-in-set-bound*[*of M - A*] **by** *auto*
ultimately have *eq1*: $(?f\ `\ ?dom) - (?f\ `\ B) = (?f\ `\ ?dom) - (?f\ `\ (\{0..<M\} \cap B))$ **by** *auto*
have $\{0..<M\} \cap B \subseteq ?dom$ **using** *assms* **by** *auto*
then have $(?f\ `\ ?dom) - (?f\ `\ (\{0..<M\} \cap B)) = ?f\ `\ (?dom - (\{0..<M\} \cap B))$
using *bij bij-minus-subset*[*of ?f*] **by** *auto*
also have $\ldots = ?f\ `\ (\{0..<M\} \cap ?C)$ **by** *auto*
finally have *eq2*: $(?f\ `\ ?dom) - (?f\ `\ B) = ?f\ `\ (\{0..<M\} \cap ?C)$ **using** *eq1* **by** *auto*

have $(?f\ `\ ?C) = (?f\ `\ (\{0..<M\} \cap ?C)) \cup (?f\ `\ (\{n.\ n \geq M\} \cap ?C))$ **by** *fastforce*
moreover have $(?f\ `\ (\{n.\ n \geq M\} \cap ?C)) \cap (?f\ `\ ?dom) = \{\}$ **using** *ind-in-set-bound*[*of M - A*] **by** *auto*
ultimately have *eq3*:$(ind\text{-}in\text{-}set\ A\ `\ ?dom) \cap (?f\ `\ ?C) = (ind\text{-}in\text{-}set\ A\ `\ ?dom) \cap (?f\ `\ (\{0..<M\} \cap ?C))$ **by** *auto*

have $\{0..<M\} \cap ?C \subseteq ?dom$ **using** *assms* **by** *auto*
then have $(ind\text{-}in\text{-}set\ A\ `\ ?dom) \cap (?f\ `\ (\{0..<M\} \cap ?C)) = (?f\ `\ (\{0..<M\} \cap ?C))$ **using** *bij* **by** *fastforce*
then show *?thesis* **using** *eq2 eq3* **by** *auto*
qed

lemma *ind-in-set-iff*:
fixes *A B* :: *nat set*
assumes $x \in A$ **and** $B \subseteq A$
shows $ind\text{-}in\text{-}set\ A\ x \in (ind\text{-}in\text{-}set\ A\ `\ B) = (x \in B)$
proof
have *lemm*: $card\ \{i.\ i \in A \wedge i < (x::nat)\ \} = card\ \{i.\ i \in A \wedge i < (y::nat)\ \} \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x = y$ **for** *A x y*
by (*metis* (*full-types*) *pick-card-in-set*)
{
assume $ind\text{-}in\text{-}set\ A\ x \in (ind\text{-}in\text{-}set\ A\ `\ B)$
then have $\exists y \in B.\ card\ \{i \in A.\ i < x\} = card\ \{i \in A.\ i < y\}$ **unfolding** *ind-in-set-def* **by** *auto*
then obtain *y* **where** *y1*: $y \in B$ **and** *ceq*: $card\ \{i \in A.\ i < x\} = card\ \{i \in A.\ i < y\}$ **by** *auto*
with *y1 assms* **have** *y0*: $y \in A$ **by** *auto*
then have $x = y$ **using** *lemm*[*OF ceq*] *y0 assms* **by** *auto*
then show $x \in B$ **using** *y1* **by** *auto*

}
qed (*simp add*: *ind-in-set-def*)

lemma *nths-reencode-eq*:
assumes *B* ⊆ *A*
shows *nths* (*nths xs A*) (*ind-in-set A* ‘ *B*) = *nths xs B*
proof (*induction xs rule*: *rev-induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*snoc x xs*)
have *seteq*: {*i*. *i* < *length xs* ∧ *i* ∈ *A*} = {*i*. *i* ∈ *A* ∧ *i* < *length xs*} **by** *auto*

show *?case*
proof (*cases length xs* ∈ *B*)
case *True*
have *nths* (*xs* @ [*x*]) *B* = *nths xs B* @ *nths* [*x*] {*l*. *l* + *length xs* ∈ *B*} **using** *nths-append*[*of xs*] **by** *auto*
moreover have *nths* [*x*] {*l*. *l* + *length xs* ∈ *B*} = [*x*] **using** *nths-singleton True* **by** *auto*
ultimately have *eqT1*: *nths* (*xs* @ [*x*]) *B* = *nths xs B* @ [*x*] **by** *auto*

then have *length xs* ∈ *A* **using** *True assms* **by** *auto*
then have *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} = [*x*] **using** *nths-singleton* **by** *auto*
moreover have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} **using** *nths-append*[*of xs*] **by** *auto*
ultimately have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ [*x*] **by** *auto*
then have *eqT2*: *nths* (*nths* (*xs* @ [*x*]) *A*) (*ind-in-set A* ‘ *B*) = *nths* (*nths xs A* @ [*x*]) (*ind-in-set A* ‘ *B*) **by** *auto*
have *eqT3*: *nths* (*nths xs A* @ [*x*]) (*ind-in-set A* ‘ *B*)
= *nths xs B* @ (*nths* [*x*] {*l*. *l* + *length* (*nths xs A*) ∈ (*ind-in-set A* ‘ *B*)})
using *nths-append*[*of nths xs A*] *snoc* **by** *auto*

have *ind-in-set A* (*length xs*) = *card* {*i*. *i* < *length xs* ∧ *i* ∈ *A*} **using** *ind-in-set-def seteq* **by** *auto*
moreover have *length* (*nths xs A*) = *card* {*i*. *i* < *length xs* ∧ *i* ∈ *A*} **using** *length-nths* **by** *auto*
ultimately have *length* (*nths xs A*) = *ind-in-set A* (*length xs*) **by** *auto*
moreover have *ind-in-set A* (*length xs*) ∈ *ind-in-set A* ‘ *B* **using** *True* **by** *auto*
ultimately have *length* (*nths xs A*) ∈ *ind-in-set A* ‘ *B* **by** *auto*
then have (*nths* [*x*] {*l*. *l* + *length* (*nths xs A*) ∈ (*ind-in-set A* ‘ *B*)}) = [*x*] **using** *nths-singleton* **by** *auto*
then have *nths* (*nths xs A* @ [*x*]) (*ind-in-set A* ‘ *B*) = *nths xs B* @ [*x*] **using** *eqT3* **by** *auto*
then show *?thesis* **using** *eqT1 eqT2* **by** *auto*
next
case *False*
have *nths* (*xs* @ [*x*]) *B* = *nths xs B* @ *nths* [*x*] {*l*. *l* + *length xs* ∈ *B*} **using**

nths-append[*of xs*] **by** *auto*
moreover have *nths* [*x*] {*l*. *l* + *length xs* ∈ *B*} = [] **using** *nths-singleton False* **by** *auto*
ultimately have *eqT1*: *nths* (*xs* @ [*x*]) *B* = *nths xs B* **by** *auto*

have *nths* (*nths* (*xs* @ [*x*]) *A*) (*ind-in-set A* ' *B*) = *nths xs B*
proof (*cases length xs* ∈ *A*)
case *True*
then have *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} = [*x*] **using** *nths-singleton* **by** *auto*
moreover have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} **using** *nths-append*[*of xs*] **by** *auto*
ultimately have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ [*x*] **by** *auto*
then have *nths* (*nths* (*xs* @ [*x*]) *A*) (*ind-in-set A* ' *B*) = *nths* (*nths xs A* @ [*x*]) (*ind-in-set A* ' *B*) **by** *auto*
then have *eqT2*: *nths* (*nths* (*xs* @ [*x*]) *A*) (*ind-in-set A* ' *B*)
= *nths xs B* @ (*nths* [*x*] {*l*. *l* + *length* (*nths xs A*) ∈ (*ind-in-set A* ' *B*)})
using *nths-append*[*of nths xs A*] *snoc* **by** *auto*

have *length* (*nths xs A*) ∈ (*ind-in-set A* ' *B*) ⟹ *length xs* ∈ *B*
proof −
assume *length* (*nths xs A*) ∈ (*ind-in-set A* ' *B*)
moreover have *length* (*nths xs A*) = *card* {*i*. *i* ∈ *A* ∧ *i* < *length xs*}
using *length-nths*[*of xs*] *seteq* **by** *auto*
ultimately have *card* {*i*. *i* ∈ *A* ∧ *i* < *length xs*} ∈ (*ind-in-set A* ' *B*) **unfolding** *ind-in-set-def* **by** *auto*
then show *length xs* ∈ *B* **using** *ind-in-set-iff True assms* **unfolding** *ind-in-set-def* **by** *auto*
qed
then have *length* (*nths xs A*) ∉ (*ind-in-set A* ' *B*) **using** *False* **by** *auto*
then have *nths* [*x*] {*l*. *l* + *length* (*nths xs A*) ∈ (*ind-in-set A* ' *B*)} = [] **using** *nths-singleton* **by** *auto*
then show *nths* (*nths* (*xs* @ [*x*]) *A*) (*ind-in-set A* ' *B*) = *nths xs B* **using** *eqT2* **by** *auto*
next
case *False*
then have *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} = [] **using** *nths-singleton* **by** *auto*
moreover have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} **using** *nths-append*[*of xs*] **by** *auto*
ultimately have *nths* (*xs* @ [*x*]) *A* = *nths xs A* **by** *auto*
then show *?thesis* **using** *snoc* **by** *auto*
qed
with *eqT1* **show** *?thesis* **by** *auto*
qed
qed

lemma *nths-reencode-eq-comp*:
assumes *B* ⊆ *A*
shows *nths* (*nths xs A*) (− *ind-in-set A* ' *B*) = *nths xs* (*A* − *B*)
proof (*induction xs rule*: *rev-induct*)

case *Nil*
then show *?case* **by** *auto*
next
case (*snoc x xs*)
have *sub20*: $A - B \subseteq A$ **using** *assms* **by** *auto*
have *seteq*: $\{i.\ i < length\ xs \wedge i \in A\} = \{i.\ i \in A \wedge i < length\ xs\}$ **by** *auto*
show *?case*
proof (*cases length xs* $\in (A - B)$)
case *True*
have *nths* (*xs* @ [*x*]) $(A - B)$ = *nths xs* $(A - B)$ @ *nths* [*x*] $\{l.\ l + length\ xs \in (A - B)\}$ **using** *nths-append*[*of xs*] **by** *auto*
moreover have *nths* [*x*] $\{l.\ l + length\ xs \in (A - B)\}$ = [*x*] **using** *nths-singleton True* **by** *auto*
ultimately have *eqT1*: *nths* (*xs* @ [*x*]) $(A - B)$ = *nths xs* $(A - B)$ @ [*x*] **by** *auto*

then have *length xs* $\in A$ **using** *True sub20* **by** *auto*
then have *nths* [*x*] $\{l.\ l + length\ xs \in A\}$ = [*x*] **using** *nths-singleton* **by** *auto*
moreover have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ *nths* [*x*] $\{l.\ l + length\ xs \in A\}$ **using** *nths-append*[*of xs*] **by** *auto*
ultimately have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ [*x*] **by** *auto*
then have *nths* (*nths* (*xs* @ [*x*]) *A*) (− (*ind-in-set A*) ‘ *B*) = *nths* (*nths xs A* @ [*x*]) (− (*ind-in-set A*) ‘ *B*) **by** *auto*
then have *eqT2*: *nths* (*nths* (*xs* @ [*x*]) *A*) (− (*ind-in-set A*) ‘ *B*)
= *nths xs* $(A - B)$ @ (*nths* [*x*] $\{l.\ l + length$ (*nths xs A*) $\in$ (− (*ind-in-set A*) ‘ *B*)$\}$)
using *nths-append*[*of nths xs A*] *snoc* **by** *auto*

have *length* (*nths xs A*) $\in$ ((*ind-in-set A*) ‘ *B*) $\Longrightarrow$ *length xs* $\in B$
proof −
assume *length* (*nths xs A*) $\in$ ((*ind-in-set A*) ‘ *B*)
moreover have *length* (*nths xs A*) = *card* $\{i.\ i \in A \wedge i < length\ xs\}$
using *length-nths*[*of xs*] *seteq* **by** *auto*
ultimately have *ind-in-set A* (*length xs*) $\in$ (*ind-in-set A* ‘ *B*) **unfolding** *ind-in-set-def* **by** *auto*
then show *length xs* $\in B$ **using** *ind-in-set-iff True assms* **by** *auto*
qed
moreover have *length xs* $\notin B$ **using** *True* **by** *auto*
ultimately have *length* (*nths xs A*) $\in$ (− (*ind-in-set A*) ‘ *B*) **by** *auto*
then have *nths* [*x*] $\{l.\ l + length$ (*nths xs A*) $\in$ (− (*ind-in-set A*) ‘ *B*)$\}$ = [*x*]
using *nths-singleton* **by** *auto*
then have *nths* (*nths* (*xs* @ [*x*]) *A*) (− (*ind-in-set A*) ‘ *B*) = *nths xs* $(A - B)$ @ [*x*] **using** *eqT2* **by** *auto*
then show *?thesis* **using** *eqT1* **by** *auto*
next
case *False*
have *nths* (*xs* @ [*x*]) $(A - B)$ = *nths xs* $(A - B)$ @ *nths* [*x*] $\{l.\ l + length\ xs \in (A - B)\}$ **using** *nths-append*[*of xs*] **by** *auto*
moreover have *nths* [*x*] $\{l.\ l + length\ xs \in (A - B)\}$ = [] **using** *nths-singleton*

False **by** *auto*
  ultimately have *eqT1*: *nths* (*xs* @ [*x*]) (*A* − *B*) = *nths xs* (*A* − *B*) **by** *auto*

  have *nths* (*nths* (*xs* @ [*x*]) *A*) (− (*ind-in-set A*) ‘ *B*) = *nths xs* (*A* − *B*)
  proof (*cases length xs* ∈ *A*)
    case *True*
    then have *True1*: *length xs* ∈ *B* **using** *False* **by** *auto*
    then have *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} = [*x*] **using** *nths-singleton True* **by** *auto*
    moreover have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} **using** *nths-append*[*of xs*] **by** *auto*
    ultimately have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ [*x*] **by** *auto*
    then have *nths* (*nths* (*xs* @ [*x*]) *A*) (− (*ind-in-set A*) ‘ *B*) = *nths* (*nths xs A* @ [*x*]) (− (*ind-in-set A*) ‘ *B*) **by** *auto*
    then have *eqT2*: *nths* (*nths* (*xs* @ [*x*]) *A*) (− (*ind-in-set A*) ‘ *B*)
      = *nths xs* (*A* − *B*) @ (*nths* [*x*] {*l*. *l* + *length* (*nths xs A*) ∈ (− (*ind-in-set A*) ‘ *B*)})
      using *nths-append*[*of nths xs A*] *snoc* **by** *auto*

    have *length* (*nths xs A*) ∈ ((*ind-in-set A*) ‘ *B*)
    proof −
      have *length* (*nths xs A*) = *card* {*i*. *i* ∈ *A* ∧ *i* < *length xs*}
        using *length-nths*[*of xs*] *seteq* **by** *auto*
      moreover have *card* {*i*. *i* ∈ *A* ∧ *i* < *length xs*} ∈ (*ind-in-set A* ‘ *B*)
        unfolding *ind-in-set-def* **using** *True ind-in-set-iff*[*of length xs*] *True1* **by** *auto*
      ultimately show *length* (*nths xs A*) ∈ (*ind-in-set A*) ‘ *B* **by** *auto*
    qed
    then have *nths* [*x*] {*l*. *l* + *length* (*nths xs A*) ∈ (− (*ind-in-set A*) ‘ *B*)} = [] **using** *nths-singleton* **by** *auto*
    then show *nths* (*nths* (*xs* @ [*x*]) *A*) (− (*ind-in-set A*) ‘ *B*) = *nths xs* (*A* − *B*) **using** *eqT2* **by** *auto*
  next
    case *False*
    then have *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} = [] **using** *nths-singleton* **by** *auto*
    moreover have *nths* (*xs* @ [*x*]) *A* = *nths xs A* @ *nths* [*x*] {*l*. *l* + *length xs* ∈ *A*} **using** *nths-append*[*of xs*] **by** *auto*
    ultimately have *nths* (*xs* @ [*x*]) *A* = *nths xs A* **by** *auto*
    then show *?thesis* **using** *snoc* **by** *auto*
  qed
  with *eqT1* **show** *?thesis* **by** *auto*
 qed
qed

lemma *nths-prod-list-split*:
 fixes *A* :: *nat set* **and** *xs* :: *nat list*
 assumes *B* ⊆ *A*
 shows *prod-list* (*nths xs A*) = (*prod-list* (*nths xs B*)) ∗ (*prod-list* (*nths xs* (*A* − *B*)))

```
proof (induction xs rule: rev-induct)
  case Nil
  then show ?case by auto
next
  let ?C = A − B
  case (snoc x xs)
  have SA: nths (xs @ [x]) A = nths xs A @ nths [x] {j. j + length xs ∈ A} using
nths-append[of xs] by auto
  have SB: nths (xs @ [x]) B = nths xs B @ nths [x] {j. j + length xs ∈ B} using
nths-append[of xs] by auto
  have SC: nths (xs @ [x]) ?C = nths xs ?C @ nths [x] {j. j + length xs ∈ ?C}
using nths-append[of xs] by auto
  show ?case
  proof (cases length xs ∈ A)
    case True
    then have nths (xs @ [x]) A = nths xs A @ [x] using SA by auto
    then have eqA: prod-list (nths (xs @ [x]) A) = prod-list (nths xs A) ∗ x by
auto
    show ?thesis
    proof (cases length xs ∈ B)
      case True
      then have nths (xs @ [x]) B = nths xs B @ [x] using SB by auto
      then have eqB: prod-list (nths (xs @ [x]) B) = prod-list (nths xs B) ∗ x by
auto
      have length xs ∉ ?C using True assms by auto
      then have nths (xs @ [x]) ?C = nths xs ?C using SC by auto
      then have eqC: prod-list (nths (xs @ [x]) ?C) = prod-list (nths xs ?C) by
auto
      then show ?thesis using snoc eqA eqB eqC by auto
    next
      case False
      then have nths (xs @ [x]) B = nths xs B using SB by auto
      then have eqB: prod-list (nths (xs @ [x]) B) = prod-list (nths xs B) by auto

      then have length xs ∈ ?C using True False assms by auto
      then have nths (xs @ [x]) ?C = nths xs ?C @ [x] using SC by auto
      then have eqC: prod-list (nths (xs @ [x]) ?C) = prod-list (nths xs ?C) ∗ x
by auto
      then show ?thesis using snoc eqA eqB eqC by auto
    qed
  next
    case False
    then have ninB: length xs ∉ B and ninC: length xs ∉ ?C using assms by
auto

    have nths (xs @ [x]) A = nths xs A using SA False nths-singleton by auto
    then have eqA: prod-list (nths (xs @ [x]) A) = prod-list (nths xs A) by auto
    have nths (xs @ [x]) B = nths xs B using SB ninB nths-singleton by auto
    then have eqB: prod-list (nths (xs @ [x]) B) = prod-list (nths xs B) by auto
```

have *nths* (*xs* @ [*x*]) *?C* = *nths xs ?C* **using** *SC ninC nths-singleton* **by** *auto*
then have *eqC*: *prod-list* (*nths* (*xs* @ [*x*]) *?C*) = *prod-list* (*nths xs ?C*) **by** *auto*
then show *?thesis* **using** *eqA eqB eqC snoc* **by** *auto*
qed
qed

4.1 Encodings

lemma *digit-encode-take*:
take n (*digit-encode ds a*) = *digit-encode* (*take n ds*) *a*
proof (*induct n arbitrary*: *ds a*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof (*cases ds*)
case *Nil*
then show *?thesis* **by** *auto*
next
case (*Cons d ds'*)
then show *?thesis* **by** (*auto simp add*: *Suc*)
qed
qed

lemma *nths-minus-upt-eq-drop*:
nths l (−{..<*n*}) = *drop n l*
apply (*induct l rule*: *rev-induct*)
by (*auto simp add*: *nths-append*)

lemma *digit-encode-drop*:
drop n (*digit-encode ds a*) = *digit-encode* (*drop n ds*) (*a div* (*prod-list* (*take n ds*)))
proof (*induct n arbitrary*: *ds a*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case*
proof (*cases ds*)
case *Nil*
then show *?thesis* **by** *auto*
next
case (*Cons d ds'*)
then show *?thesis* **by** (*auto simp add*: *Suc div-mult2-eq*)
qed
qed

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* ⊆ *A*
shows *dims1* = *nths dims* (*A* ∩ *vars*)
using *nths-intersection-eq assms* **unfolding** *dims1-def* **by** *auto*

lemma *dims2-alter*:
assumes *avars* ⊆ *A*
shows *dims2* = *nths dims* (*A*−*vars*)
using *nths-minus-eq assms* **unfolding** *dims2-def* **by** *auto*

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*))
using *length-digit-encode*[*of dims i*] *nths-intersection-eq*[*of digit-encode dims i A vars*] *assms* **unfolding** *encode1-def*
by (*subgoal-tac nths* (*digit-encode dims i*) (*vars*) = *nths* (*digit-encode dims i*) (*A* ∩ *vars*), *auto*)

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*))
using *length-digit-encode*[*of dims i*] *nths-minus-eq*[*of digit-encode dims i A*] *assms*
unfolding *encode2-def*
by (*subgoal-tac nths* (*digit-encode dims i*) (− *vars*) = *nths* (*digit-encode dims i*) (*A* − *vars*), *auto*)

lemma *encode1-lt* [*simp*]:
assumes *i* < *d*
shows *encode1 i* < *d1*
unfolding *d1-def encode1-def* **apply** (*rule digit-decode-lt*)
using *dims1-def assms d-def digit-encode-valid-index valid-index-nths* **by** *auto*

lemma *encode2-lt* [*simp*]:
assumes *i* < *d*
shows *encode2 i* < *d2*
unfolding *d2-def encode2-def* **apply** (*rule digit-decode-lt*)
using *dims2-def assms d-def digit-encode-valid-index valid-index-nths* **by** *auto*

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*
unfolding *encode12.simps encode1-def encode2-def*
using *assms d-def digit-encode-valid-index dims1-def dims2-def valid-index-nths*
by *auto*

lemma *encode12-inv1*:
assumes *i* < *d1 j* < *d2*
shows *encode1* (*encode12* (*i*, *j*)) = *i*
unfolding *encode12.simps encode1-def*
using *assms* **unfolding** *d1-def d2-def dims1-def dims2-def*
by (*metis digit-decode-encode-lt digit-encode-decode digit-encode-valid-index valid-index-weave*(*1*,*2*))

lemma *encode12-inv2*:
assumes *i* < *d1 j* < *d2*
shows *encode2* (*encode12* (*i*, *j*)) = *j*
unfolding *encode12.simps encode2-def*
using *assms* **unfolding** *d1-def d2-def dims1-def dims2-def*
by (*metis digit-decode-encode-lt digit-encode-decode digit-encode-valid-index valid-index-weave*(*1*,*3*))

lemma *encode12-lt*:
assumes *i* < *d1 j* < *d2*

shows *encode12* (*i*, *j*) < *d*
using *assms* **unfolding** *encode12.simps d-def d1-def d2-def dims1-def dims2-def*
by (*simp add*: *digit-decode-lt digit-encode-valid-index valid-index-weave*(*1*))

lemma *sum-encode*: ($\sum$ *i* = *0..<d1*. $\sum$ *j* = *0..<d2*. *f i j*) = *sum* (*λk*. *f* (*encode1 k*) (*encode2 k*)) {*0..<d*}
apply (*subst sum.cartesian-product*)
apply (*rule sum.reindex-bij-witness*[**where** *i*=*λk*. (*encode1 k*, *encode2 k*) **and** *j*=*encode12*])
by (*auto simp*: *encode12-inv1 encode12-inv2 encode12-inv encode12-lt*)

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* (*λi*. *v1* $ *encode1 i* ∗ *v2* $ *encode2 i*)

lemma *tensor-vec-dim* [*simp*]:
dim-vec (*tensor-vec v1 v2*) = *d*
unfolding *tensor-vec-def* **by** *auto*

lemma *tensor-vec-carrier*:
tensor-vec v1 v2 ∈ *carrier-vec d*
unfolding *tensor-vec-def* **by** *auto*

lemma *tensor-vec-eval*:
assumes *i* < *d*
shows *tensor-vec v1 v2* $ *i* = *v1* $ *encode1 i* ∗ *v2* $ *encode2 i*
unfolding *tensor-vec-def* **using** *assms* **by** *simp*

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*
apply (*rule eq-vecI*, *auto*)
unfolding *tensor-vec-eval*
using *assms*(*2*) *comm-semiring-class.distrib* **by** *force*

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*
apply (*rule eq-vecI*, *auto*)
unfolding *tensor-vec-eval*

using *assms*(*3*) *semiring-class.distrib-left* **by** *force*

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*
unfolding *tensor-mat-def* **by** *auto*

lemma *tensor-mat-dim-col* [*simp*]:
dim-col (*tensor-mat m1 m2*) = *d*
unfolding *tensor-mat-def* **by** *auto*

lemma *tensor-mat-carrier*:
tensor-mat m1 m2 ∈ *carrier-mat d d*
unfolding *tensor-mat-def* **by** *auto*

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*)
unfolding *tensor-mat-def* **using** *assms* **by** *simp*

lemma *tensor-mat-zero1*:
shows *tensor-mat* (0_m *d1 d1*) *m1* = 0_m *d d*
apply (*rule eq-matI*)
by (*auto simp add*: *tensor-mat-eval*)

lemma *tensor-mat-zero2*:
shows *tensor-mat m1* (0_m *d2 d2*) = 0_m *d d*
apply (*rule eq-matI*)
by (*auto simp add*: *tensor-mat-eval*)

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*
apply (*rule eq-matI*, *auto*)
unfolding *tensor-mat-eval*
using *assms*(*2*) *comm-semiring-class.distrib* **by** *force*

lemma *tensor-mat-add2*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
and *m3* ∈ *carrier-mat d2 d2*
shows *tensor-mat m1* (*m2* + *m3*) = *tensor-mat m1 m2* + *tensor-mat m1 m3*

apply (*rule eq-matI*, *auto*)
unfolding *tensor-mat-eval*
using *assms*(*3*) *semiring-class.distrib-left* **by** *force*

lemma *tensor-mat-minus1*:
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*
apply (*rule eq-matI*, *auto*)
unfolding *tensor-mat-eval*
apply (*subst index-minus-mat*)
subgoal using *assms* **by** *auto*
subgoal using *assms* **by** *auto*
using *assms*(*2*) *ring-class.left-diff-distrib* **by** *force*

lemma *tensor-mat-matrix-sum2*:
assumes *m1* ∈ *carrier-mat d1 d1*
shows (⋀*k*. *k* < *n* ⟹ *f k* ∈ *carrier-mat d2 d2*)
⟹ *matrix-sum d* (λ*k*. *tensor-mat m1* (*f k*)) *n* = *tensor-mat m1* (*matrix-sum d2 f n*)
proof (*induct n*)
case *0*
then show *?case* **apply** *simp* **using** *tensor-mat-zero2*[*of m1*] **by** *auto*
next
case (*Suc n*)
then have *k* < *n* ⟹ *f k* ∈ *carrier-mat d2 d2* **for** *k* **by** *auto*
then have *ds*: *matrix-sum d2 f n* ∈ *carrier-mat d2 d2* **using** *matrix-sum-dim* **by** *auto*
have *dn*: *f n* ∈ *carrier-mat d2 d2* **using** *Suc* **by** *auto*
have *matrix-sum d2 f* (*Suc n*) = *f n* + *matrix-sum d2 f n* **by** *simp*
then have *eq*: *tensor-mat m1* (*matrix-sum d2 f* (*Suc n*))
= *tensor-mat m1* (*f n*) + *tensor-mat m1* (*matrix-sum d2 f n*)
using *tensor-mat-add2 dn ds assms* **by** *auto*

have *matrix-sum d* (λ*k*. *tensor-mat m1* (*f k*)) (*Suc n*)
= *tensor-mat m1* (*f n*) + *matrix-sum d* (λ*k*. *tensor-mat m1* (*f k*)) *n* **by** *simp*
also have ... = *tensor-mat m1* (*f n*) + *tensor-mat m1* (*matrix-sum d2 f n*)
using *Suc* **by** *auto*
finally show *?case* **using** *eq* **by** *auto*
qed

lemma *tensor-mat-scale1*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d2 d2*
shows *tensor-mat* (*a* $\cdot_m$ *m1*) *m2* = *a* $\cdot_m$ *tensor-mat m1 m2*
apply (*rule eq-matI*, *auto*)
unfolding *tensor-mat-eval*
using *assms comm-semiring-class.distrib* **by** *force*

lemma *tensor-mat-scale2*:
  assumes *m1* ∈ *carrier-mat d1 d1*
    and *m2* ∈ *carrier-mat d2 d2*
  shows *tensor-mat m1* (*a* $\cdot_m$ *m2*) = *a* $\cdot_m$ *tensor-mat m1 m2*
  apply (*rule eq-matI*, *auto*)
  unfolding *tensor-mat-eval*
  using *assms comm-semiring-class.distrib* **by** *force*

lemma *tensor-mat-trace*:
  assumes *m1* ∈ *carrier-mat d1 d1*
    and *m2* ∈ *carrier-mat d2 d2*
  shows *trace* (*tensor-mat m1 m2*) = *trace m1* ∗ *trace m2*
  apply (*auto simp add*: *tensor-mat-carrier trace-def tensor-mat-eval*)
  apply (*subst Groups-Big.sum-product*)
  apply (*subst sum-encode*[*symmetric*])
  using *assms* **by** *auto*

lemma *tensor-mat-id*:
  tensor-mat (1_m *d1*) (1_m *d2*) = 1_m *d*
proof (*rule eq-matI*, *auto*)
  show *tensor-mat* (1_m *d1*) (1_m *d2*) $$ (*i*, *i*) = *1* **if** *i* < *d* **for** *i*
    using *that* **by** (*simp add*: *tensor-mat-eval*)
next
  show *tensor-mat* (1_m *d1*) (1_m *d2*) $$ (*i*, *j*) = *0* **if** *i* < *d j* < *d i* ≠ *j* **for** *i j*
    using *that* **apply** (*simp add*: *tensor-mat-eval*)
    by (*metis encode12-inv*)
qed

lemma *tensor-mat-mult-vec*:
  assumes *m1* ∈ *carrier-mat d1 d1*
    and *m2* ∈ *carrier-mat d2 d2*
    and *v1* ∈ *carrier-vec d1*
    and *v2* ∈ *carrier-vec d2*
  shows *tensor-vec* (*m1* $*_v$ *v1*) (*m2* $*_v$ *v2*) = *tensor-mat m1 m2* $*_v$ *tensor-vec v1 v2*
proof (*rule eq-vecI*, *auto*)
  fix *i j* :: *nat*
  assume *i*: *i* < *d*
  let *?i1* = *encode1 i* **and** *?i2* = *encode2 i*
  have *tensor-vec* (*m1* $*_v$ *v1*) (*m2* $*_v$ *v2*) $ *i* = (*m1* $*_v$ *v1*) $ *?i1* ∗ (*m2* $*_v$ *v2*) $ *?i2*
    using *i* **by** (*simp add*: *tensor-vec-eval*)
  also have ... = (*row m1 ?i1* · *v1*) ∗ (*row m2 ?i2* · *v2*)
    using *assms i* **by** *auto*
  also have ... = (∑ *i* = *0*..<*d1*. *m1* $$ (*?i1*, *i*) ∗ *v1* $ *i*) ∗ (∑ *j* = *0*..<*d2*. *m2* $$ (*?i2*, *j*) ∗ *v2* $ *j*)
    using *assms i* **by** (*simp add*: *scalar-prod-def*)
  also have ... = (∑ *i* = *0*..<*d1*. ∑ *j* = *0*..<*d2*. (*m1* $$ (*?i1*, *i*) ∗ *v1* $ *i*) ∗ (*m2*

$$ (?i2, j) * v2 $ j))
    by (*rule Groups-Big.sum-product*)
  also have ... = ($\sum$ *i* = *0..<d*. (*m1* $$ (*?i1*, *encode1 i*) * *v1* $ (*encode1 i*)) * (*m2* $$ (*?i2*, *encode2 i*) * *v2* $ (*encode2 i*)))
    by (*rule sum-encode*)
  also have ... = *row* (*tensor-mat m1 m2*) *i* · *tensor-vec v1 v2*
    apply (*simp add*: *scalar-prod-def tensor-mat-eval tensor-vec-eval i*)
    by (*rule sum.cong*, *auto*)
  finally show *tensor-vec* (*m1* $*_v$ *v1*) (*m2* $*_v$ *v2*) $ *i* = *row* (*tensor-mat m1 m2*) *i* · *tensor-vec v1 v2* **by** *auto*
qed

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 (*rule eq-matI*, *auto*)
  fix *i j* :: *nat*
  assume *i*: *i* < *d* **and** *j*: *j* < *d*
  let *?i1* = *encode1 i* **and** *?i2* = *encode2 i* **and** *?j1* = *encode1 j* **and** *?j2* = *encode2 j*
  have *tensor-mat* (*m1* * *m2*) (*m3* * *m4*) $$ (*i*, *j*) = (*m1* * *m2*) $$ (*?i1*, *?j1*) * (*m3* * *m4*) $$ (*?i2*, *?j2*)
    using *i j* **by** (*simp add*: *tensor-mat-eval*)
  also have ... = (*row m1 ?i1* · *col m2 ?j1*) * (*row m3 ?i2* · *col m4 ?j2*)
    using *assms i j* **by** *auto*
  also have ... = ($\sum$ *i* = *0..<d1*. *m1* $$ (*?i1*, *i*) * *m2* $$ (*i*, *?j1*)) * ($\sum$ *j* = *0..<d2*. *m3* $$ (*?i2*, *j*) * *m4* $$ (*j*, *?j2*))
    using *assms i j* **by** (*simp add*: *scalar-prod-def*)
  also have ... = ($\sum$ *i* = *0..<d1*. $\sum$ *j* = *0..<d2*. (*m1* $$ (*?i1*, *i*) * *m2* $$ (*i*, *?j1*)) * (*m3* $$ (*?i2*, *j*) * *m4* $$ (*j*, *?j2*)))
    by (*rule Groups-Big.sum-product*)
  also have ... = ($\sum$ *i* = *0..<d*. (*m1* $$ (*?i1*, *encode1 i*) * *m2* $$ (*encode1 i*, *?j1*)) * (*m3* $$ (*?i2*, *encode2 i*) * *m4* $$ (*encode2 i*, *?j2*)))
    by (*rule sum-encode*)
  also have ... = *row* (*tensor-mat m1 m3*) *i* · *col* (*tensor-mat m2 m4*) *j*
    apply (*simp add*: *scalar-prod-def tensor-mat-eval i j*)
    by (*rule sum.cong*, *auto*)
  finally show *tensor-mat* (*m1* * *m2*) (*m3* * *m4*) $$ (*i*, *j*) = *row* (*tensor-mat m1 m3*) *i* · *col* (*tensor-mat m2 m4*) *j* **.**
qed

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*)

apply (*rule eq-matI*, *auto*)
unfolding *tensor-mat-def adjoint-def*
using *assms* **by** (*simp add*: *conjugate-dist-mul*)

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*)
using *assms* **by** (*metis hermitian-def tensor-mat-adjoint*)

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*)
using *assms* **apply** (*auto simp add*: *unitary-def tensor-mat-adjoint*)
using *assms* **unfolding** *inverts-mat-def*
apply (*subst tensor-mat-mult*[*symmetric*], *auto*)
by (*rule tensor-mat-id*)

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 −
obtain *M1* **where** *M1*: *m1* = *M1* ∗ *adjoint M1* **and** *dM1*:*M1* ∈ *carrier-mat d1 d1* **using** *positive-only-if-decomp assms* **by** *auto*
obtain *M2* **where** *M2*: *m2* = *M2* ∗ *adjoint M2* **and** *dM2*:*M2* ∈ *carrier-mat d2 d2* **using** *positive-only-if-decomp assms* **by** *auto*
have (*adjoint* (*tensor-mat M1 M2*)) = *tensor-mat* (*adjoint M1*) (*adjoint M2*) **using** *tensor-mat-adjoint dM1 dM2* **by** *auto*
then have *tensor-mat M1 M2* ∗ (*adjoint* (*tensor-mat M1 M2*)) = *tensor-mat* (*M1* ∗ *adjoint M1*) (*M2* ∗ *adjoint M2*)
using *dM1 dM2 adjoint-dim*[*OF dM1*] *adjoint-dim*[*OF dM2*] **by** (*auto simp add*: *tensor-mat-mult*)
also have . . . = *tensor-mat m1 m2* **using** *M1 M2* **by** *auto*
finally have *tensor-mat m1 m2* = *tensor-mat M1 M2* ∗ (*adjoint* (*tensor-mat M1 M2*))**..**
then have ∃ *M*. *M* ∗ *adjoint M* = *tensor-mat m1 m2* **by** *auto*
moreover have *tensor-mat m1 m2* ∈ *carrier-mat d d* **using** *tensor-mat-carrier* **by** *auto*
ultimately show *?thesis* **using** *positive-if-decomp*[*of tensor-mat m1 m2*] **by** *auto*
qed

lemma *tensor-mat-positive-le*:
assumes $m1 \in carrier\text{-}mat\ d1\ d1$
and $m2 \in carrier\text{-}mat\ d2\ d2$
and *positive m1*
and *positive m2*
and $m1 \leq_L A$
and $m2 \leq_L B$
shows $tensor\text{-}mat\ m1\ m2 \leq_L tensor\text{-}mat\ A\ B$
proof −
have *dA*: $A \in carrier\text{-}mat\ d1\ d1$ **using** *assms lowner-le-def* **by** *auto*
have *pA*: *positive A* **using** *assms dA lowner-le-trans-positiveI*[*of m1*] **by** *auto*
have *dB*: $B \in carrier\text{-}mat\ d2\ d2$ **using** *assms lowner-le-def* **by** *auto*
have *pB*: *positive B* **using** *assms dB lowner-le-trans-positiveI*[*of m2*] **by** *auto*
have $A - m1 = A + (- m1)$ **using** *assms* **by** (*auto simp add*: *minus-add-uminus-mat*)
then have *positive* $(A + (- m1))$ **using** *assms* **unfolding** *lowner-le-def* **by** *auto*
then have *p1*: *positive* (*tensor-mat* $(A + (- m1))$ *m2*) **using** *assms tensor-mat-positive* **by** *auto*
moreover have *tensor-mat* $(- m1)$ $m2 = -$ *tensor-mat m1 m2* **using** *assms*
apply (*subgoal-tac* $- m1 = -1 \cdot_m m1$)
by (*auto simp add*: *tensor-mat-scale1*)
moreover have *tensor-mat* $(A + (- m1))$ $m2 =$ *tensor-mat A m2* + (*tensor-mat* $(- m1)$ *m2*) **using**
assms **by** (*auto simp add*: *tensor-mat-add1 dA*)
ultimately have *tensor-mat* $(A + (- m1))$ $m2 =$ *tensor-mat A m2* − (*tensor-mat m1 m2*) **by** *auto*
with *p1* **have** *le1*: $tensor\text{-}mat\ m1\ m2 \leq_L tensor\text{-}mat\ A\ m2$ **unfolding** *lowner-le-def* **by** *auto*

have $B - m2 = B + (- m2)$ **using** *assms* **by** (*auto simp add*: *minus-add-uminus-mat*)
then have *positive* $(B + (- m2))$ **using** *assms* **unfolding** *lowner-le-def* **by** *auto*
then have *p2*: *positive* (*tensor-mat A* $(B + (- m2))$)
using *assms tensor-mat-positive positive-one dA dB pA* **by** *auto*
moreover have *tensor-mat A* $(-m2) = -$ *tensor-mat A m2*
using *assms* **apply** (*subgoal-tac* $- m2 = -1 \cdot_m m2$)
by (*auto simp add*: *tensor-mat-scale2 dA*)
moreover have *tensor-mat A* $(B + (- m2)) =$ *tensor-mat A B* + *tensor-mat A* $(- m2)$
using *assms* **by** (*auto simp add*: *tensor-mat-add2 dA dB*)
ultimately have *tensor-mat A* $(B + (- m2)) =$ *tensor-mat A B* − *tensor-mat A m2* **by** *auto*
with *p2* **have** *le20*: $tensor\text{-}mat\ A\ m2 \leq_L tensor\text{-}mat\ A\ B$ **unfolding** *lowner-le-def* **by** *auto*

show *?thesis* **apply** (*subst lowner-le-trans*[*of - d tensor-mat* (*A*) *m2*])
subgoal using *tensor-mat-carrier* **by** *auto*
subgoal using *tensor-mat-carrier* **by** *auto*
using *le1 le20* **by** *auto*
qed

lemma *tensor-mat-le-one*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-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-mat m1 m2* $\leq_L$ *1* $_m$ *d*
proof −
have *1* $_m$ *d1* − *m1* = *1* $_m$ *d1* + (− *m1*) **using** *assms* **by** (*auto simp add*: *minus-add-uminus-mat*)
then have *positive* (*1* $_m$ *d1* + (− *m1*)) **using** *assms* **unfolding** *lowner-le-def* **by** *auto*
then have *p1*: *positive* (*tensor-mat* (*1* $_m$ *d1* + (− *m1*)) *m2*) **using** *assms tensor-mat-positive* **by** *auto*
moreover have *tensor-mat* (− *m1*) *m2* = − *tensor-mat m1 m2* **using** *assms* **apply** (*subgoal-tac* − *m1* = −*1* $\cdot_m$ *m1*)
by (*auto simp add*: *tensor-mat-scale1*)
moreover have *tensor-mat* (*1* $_m$ *d1* + (− *m1*)) *m2* = *tensor-mat* (*1* $_m$ *d1*) *m2* + (*tensor-mat* (− *m1*) *m2*) **using**
assms **by** (*auto simp add*: *tensor-mat-add1*)
ultimately have *tensor-mat* (*1* $_m$ *d1* + (− *m1*)) *m2* = *tensor-mat* (*1* $_m$ *d1*) *m2* − (*tensor-mat m1 m2*) **by** *auto*
with *p1* **have** *le1*: (*tensor-mat m1 m2*) $\leq_L$ *tensor-mat* (*1* $_m$ *d1*) *m2* **unfolding** *lowner-le-def* **by** *auto*

have *1* $_m$ *d2* − *m2* = *1* $_m$ *d2* + (− *m2*) **using** *assms* **by** (*auto simp add*: *minus-add-uminus-mat*)
then have *positive* (*1* $_m$ *d2* + (− *m2*)) **using** *assms* **unfolding** *lowner-le-def* **by** *auto*
then have *p2*: *positive* (*tensor-mat* (*1* $_m$ *d1*) (*1* $_m$ *d2* + (− *m2*))) **using** *assms tensor-mat-positive positive-one* **by** *auto*
moreover have *tensor-mat* (*1* $_m$ *d1*) (−*m2*) = − *tensor-mat* (*1* $_m$ *d1*) *m2* **using** *assms* **apply** (*subgoal-tac* − *m2* = −*1* $\cdot_m$ *m2*)
by (*auto simp add*: *tensor-mat-scale2*)
moreover have *tensor-mat* (*1* $_m$ *d1*) (*1* $_m$ *d2* + (− *m2*)) = *tensor-mat* (*1* $_m$ *d1*) (*1* $_m$ *d2*) + (*tensor-mat* (*1* $_m$ *d1*) (− *m2*)) **using**
assms **by** (*auto simp add*: *tensor-mat-add2*)
ultimately have *tensor-mat* (*1* $_m$ *d1*) (*1* $_m$ *d2* + (− *m2*)) = *tensor-mat* (*1* $_m$ *d1*) (*1* $_m$ *d2*) − (*tensor-mat* (*1* $_m$ *d1*) *m2*) **by** *auto*
with *p2* **have** *le20*: *tensor-mat* (*1* $_m$ *d1*) *m2* $\leq_L$ *tensor-mat* (*1* $_m$ *d1*) (*1* $_m$ *d2*) **unfolding** *lowner-le-def* **by** *auto*
then have *le2*: *tensor-mat* (*1* $_m$ *d1*) *m2* $\leq_L$ *1* $_m$ *d* **apply** (*subst tensor-mat-id*[*symmetric*]) **by** *auto*
have *tensor-mat* (*1* $_m$ *d1*) (*1* $_m$ *d2*) = *1* $_m$ *d* **using** *tensor-mat-id* **by** *auto*

show *?thesis* **apply** (*subst lowner-le-trans*[*of* - *d tensor-mat* (*1* $_m$ *d1*) *m2*])
subgoal using *tensor-mat-carrier* **by** *auto*
subgoal using *tensor-mat-carrier* **by** *auto*

using *le1 le2* **by** *auto*
qed

4.3 Extension of matrices

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

lemma *mat-extension-carrier*:
mat-extension m ∈ *carrier-mat d d*
by (*simp add*: *mat-extension-def tensor-mat-carrier*)

lemma *mat-extension-add*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
shows *mat-extension* (*m1* + *m2*) = *mat-extension m1* + *mat-extension m2*
using *assms* **by** (*simp add*: *mat-extension-def tensor-mat-add1*)

lemma *mat-extension-trace*:
assumes *m* ∈ *carrier-mat d1 d1*
shows *trace* (*mat-extension m*) = *d2* ∗ *trace m*
unfolding *mat-extension-def*
using *assms* **by** (*simp add*: *tensor-mat-trace*)

lemma *mat-extension-id*:
mat-extension (1_m *d1*) = 1_m *d*
unfolding *mat-extension-def* **by** (*rule tensor-mat-id*)

lemma *mat-extension-mult*:
assumes *m1* ∈ *carrier-mat d1 d1*
and *m2* ∈ *carrier-mat d1 d1*
shows *mat-extension* (*m1* ∗ *m2*) = *mat-extension m1* ∗ *mat-extension m2*
using *assms* **by** (*simp add*: *mat-extension-def tensor-mat-mult*[*symmetric*])

lemma *mat-extension-hermitian*:
assumes *m* ∈ *carrier-mat d1 d1*
and *hermitian m*
shows *hermitian* (*mat-extension m*)
using *assms* **by** (*simp add*: *hermitian-one mat-extension-def tensor-mat-hermitian*)

lemma *mat-extension-unitary*:
assumes *m* ∈ *carrier-mat d1 d1*
and *unitary m*
shows *unitary* (*mat-extension m*)
using *assms* **by** (*simp add*: *mat-extension-def tensor-mat-unitary unitary-one*)

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 −
  {
  fix *i*
 have *nths dims* (− *vars2*) = *nths dims vars1* **using** *nths-split-complement-eq*[*symmetric*] *assms* **by** *auto*
  then have *eq2211*: *partial-state.dims2 dims vars2* = *partial-state.dims1 dims vars1*
    unfolding *partial-state.dims2-def partial-state.dims1-def* **by** *auto*
 have *nths dims* (− *vars1*) = *nths dims vars2* **using** *nths-split-complement-eq*[*symmetric*, *of vars2*] *assms* **by** *auto*
  then have *eq2112*: *partial-state.dims2 dims vars1* = *partial-state.dims1 dims vars2*
    unfolding *partial-state.dims2-def partial-state.dims1-def* **by** *auto*

  have *vars1* ∪ *vars2* − *vars2* = *vars1* **using** *assms* **by** *auto*
  then have *e1*:*partial-state.encode2 dims vars2 i* = *partial-state.encode1 dims* (*vars1*) *i*
    using *assms*(*2*) *partial-state.encode2-alter*[*of dims vars1* ∪ *vars2 vars2*]
   unfolding *partial-state.encode2-def partial-state.encode1-def* **apply** (*subst eq2211*[*symmetric*]) **by** *auto*

  have *vars1* ∪ *vars2* − *vars1* = *vars2* **using** *assms* **by** *auto*
  then have *e2*:*partial-state.encode2 dims vars1 i* = *partial-state.encode1 dims* (*vars2*) *i*
    using *assms*(*2*) *partial-state.encode2-alter*[*of dims vars1* ∪ *vars2 vars1*]
   unfolding *partial-state.encode2-def partial-state.encode1-def* **apply** (*subst eq2112*[*symmetric*]) **by** *auto*

  note *e1 e2*
  }
  note *e* = *this*
  show *?thesis*
  unfolding *partial-state.tensor-mat-def* **apply** (*rule cong-mat*, *simp-all*)
  unfolding *partial-state.dims1-def partial-state.dims2-def*
  using *e* **by** *auto*
qed

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*
  unfolding *d0-def d1-def d2-def dims0-def dims1-def dims2-def vars0-def*
  using *disjoint nths-prod-list-split*[*of vars1 vars1* ∪ *vars2 dims*]
  apply (*subgoal-tac vars1* ∪ *vars2* − *vars1* = *vars2*)
  by *auto*

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'*
unfolding *vars1'-def ind-in-set-def* **by** *auto*

lemma *vars1'D*:
i ∈ *vars1'* ⟹ ∃ *x*∈*vars1*. *card* {*y*∈*vars0*. *y* < *x*} = *i*
unfolding *vars1'-def ind-in-set-def* **by** *auto*

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*)}
using *bij-ind-in-set-bound* **unfolding** *ind-in-set-def* **by** *auto*

lemma *length-dims0*:
length dims0 = *card* ({*0*..<*length dims*} ∩ *vars0*)
unfolding *dims0-def* **using** *length-nths*[*of dims vars0*]
apply (*subgoal-tac* {*i*. *i* < *length dims* ∧ *i* ∈ *vars0*}= {*0*..<*length dims*} ∩ *vars0*)
by *auto*

lemma *length-dims0-minus-vars2'-is-vars1'*:
{*0*..<*length dims0*} − *vars2'* = {*0*..<*length dims0*} ∩ *vars1'*
proof −
have *sub20*: *vars2* ⊆ *vars0* **unfolding** *vars0-def* **by** *auto*
have *sub1*: *vars1* = *vars0* − *vars2* **unfolding** *vars0-def* **using** *disjoint* **by** *auto*
have *eq*: {*0*..<*length dims0*} = *ind-in-set vars0* ‘ ({*0*..<*length dims*} ∩ *vars0*)
using *ind-in-set-bij length-dims0 bij-betw-imp-surj-on*[*of ind-in-set vars0*] **by** *auto*
show *?thesis* **unfolding** *vars2'-def vars1'-def eq* **using** *ind-in-set-minus-subset-bound*[*OF sub20*] *sub1* **by** *auto*
qed

lemma *length-dims0-minus-vars1'-is-vars2'*:
{*0*..<*length dims0*} − *vars1'* = {*0*..<*length dims0*} ∩ *vars2'*
proof −
have *sub10*: *vars1* ⊆ *vars0* **unfolding** *vars0-def* **by** *auto*
have *sub2*: *vars2* = *vars0* − *vars1* **unfolding** *vars0-def* **using** *disjoint* **by** *auto*
have *eq*: {*0*..<*length dims0*} = *ind-in-set vars0* ‘ ({*0*..<*length dims*} ∩ *vars0*)
using *ind-in-set-bij length-dims0 bij-betw-imp-surj-on*[*of ind-in-set vars0*] **by** *auto*
show *?thesis* **unfolding** *vars2'-def vars1'-def eq* **using** *ind-in-set-minus-subset-bound*[*OF sub10*] *sub2* **by** *auto*
qed

lemma *nths-vars1'*:
nths dims0 vars1' = *dims1*
using *nths-reencode-eq*[*of vars1 vars0 dims*]
using *nths-reencode-eq-comp*[*of vars1 vars0 dims*]
unfolding *vars0-def ind-in-set-def vars1'-def dims1-def dims0-def* **by** *auto*

lemma *nths-vars1′-comp*:
nths dims0 (−*vars2′*) = *dims1*
using *nths-reencode-eq-comp*[*of vars2 vars0 dims*] *disjoint*
unfolding *vars0-def ind-in-set-def vars2′-def dims1-def dims0-def*
apply (*subgoal-tac* (*vars1* ∪ *vars2* − *vars2*) = *vars1*) **by** *auto*

lemma *nths-vars2′*:
nths dims0 (−*vars1′*) = *dims2*
using *nths-reencode-eq-comp*[*of vars1 vars0 dims*] *disjoint*
unfolding *vars0-def ind-in-set-def vars1′-def dims2-def dims0-def*
apply (*subgoal-tac* (*vars1* ∪ *vars2* − *vars1*) = *vars2*) **by** *auto*

lemma *nths-vars2′-comp*:
nths dims0 (*vars2′*) = *dims2*
using *nths-reencode-eq*[*of vars2 vars0 dims*]
unfolding *vars0-def ind-in-set-def vars2′-def dims2-def dims0-def*
by *auto*

lemma *ptensor-encode1-encode2*:
partial-state.encode1 dims0 vars1′ = *partial-state.encode2 dims0 vars2′*
proof −
have *partial-state.encode1 dims0 vars1′ i*
= *digit-decode* (*partial-state.dims1 dims0 vars1′*) (*nths* (*digit-encode dims0 i*) ({*0..<length dims0*} ∩ *vars1′*)) **for** *i*
using *partial-state.encode1-alter* **by** *auto*
moreover have *partial-state.encode2 dims0 vars2′ i*
= *digit-decode* (*partial-state.dims2 dims0 vars2′*) (*nths* (*digit-encode dims0 i*) ({*0..<length dims0*} − *vars2′*)) **for** *i*
using *partial-state.encode2-alter* **by** *auto*
moreover have *partial-state.dims1 dims0 vars1′* = *partial-state.dims2 dims0 vars2′*
unfolding *partial-state.dims1-def partial-state.dims2-def* **using** *nths-vars1′ nths-vars1′-comp* **by** *auto*
ultimately show *?thesis* **using** *length-dims0-minus-vars2′-is-vars1′* **by** *auto*
qed

lemma *ptensor-encode2-encode1*:
partial-state.encode1 dims0 vars2′ = *partial-state.encode2 dims0 vars1′*
proof −
have *partial-state.encode1 dims0 vars2′ i*
= *digit-decode* (*partial-state.dims1 dims0 vars2′*) (*nths* (*digit-encode dims0 i*) ({*0..<length dims0*} ∩ *vars2′*)) **for** *i*
using *partial-state.encode1-alter* **by** *auto*
moreover have *partial-state.encode2 dims0 vars1′ i*
= *digit-decode* (*partial-state.dims2 dims0 vars1′*) (*nths* (*digit-encode dims0 i*) ({*0..<length dims0*} − *vars1′*)) **for** *i*
using *partial-state.encode2-alter* **by** *auto*
moreover have *partial-state.dims1 dims0 vars2′* = *partial-state.dims2 dims0*

vars1'
  unfolding *partial-state.dims1-def partial-state.dims2-def* **using** *nths-vars2' nths-vars2'-comp* **by** *auto*
ultimately show *?thesis* **using** *length-dims0-minus-vars1'-is-vars2'* **by** *auto*
qed

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*
by (*simp add*: *ptensor-vec-def partial-state.tensor-vec-dim state-sig.d-def d0-def*)

lemma *ptensor-vec-carrier*:
ptensor-vec v1 v2 ∈ *carrier-vec d0*
by (*simp add*: *carrier-dim-vec*)

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*
unfolding *ptensor-vec-def*
apply (*rule partial-state.tensor-vec-add1*)
unfolding *partial-state.d1-def partial-state.d2-def*
partial-state.dims1-def partial-state.dims2-def nths-vars1' nths-vars2'
using *assms* **unfolding** *d1-def d2-def* **by** *auto*

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*
by (*simp add*: *ptensor-mat-def partial-state.tensor-mat-dim-row d0-def state-sig.d-def*)

lemma *ptensor-mat-dim-col* [*simp*]:
dim-col (*ptensor-mat m1 m2*) = *d0*
by (*simp add*: *ptensor-mat-def partial-state.tensor-mat-dim-col d0-def state-sig.d-def*)

lemma *ptensor-mat-carrier*:
ptensor-mat m1 m2 ∈ *carrier-mat d0 d0*
by (*simp add*: *carrier-matI*)

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*
unfolding *ptensor-mat-def*
apply (*rule partial-state.tensor-mat-add1*)
unfolding *partial-state.d1-def partial-state.d2-def partial-state.dims1-def partial-state.dims2-def nths-vars1′ nths-vars2′*
using *assms* **unfolding** *d1-def d2-def* **by** *auto*

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*
unfolding *ptensor-mat-def*
apply (*rule partial-state.tensor-mat-trace*)
unfolding *partial-state.d1-def partial-state.d2-def partial-state.dims1-def partial-state.dims2-def nths-vars1′ nths-vars2′*
using *assms* **unfolding** *d1-def d2-def* **by** *auto*

lemma *ptensor-mat-id*:
ptensor-mat (1_m *d1*) (1_m *d2*) = 1_m *d0*
unfolding *ptensor-mat-def*
by (*metis d0-def d1-def d2-def nths-vars1′ nths-vars2′ partial-state.d1-def partial-state.d2-def partial-state.dims1-def partial-state.dims2-def partial-state.tensor-mat-id state-sig.d-def*)

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 −
interpret *st*: *partial-state dims0 vars1′*.
have *st.d1* = *d1* **unfolding** *st.d1-def st.dims1-def d1-def nths-vars1′* **by** *auto*
moreover have *st.d2* = *d2* **unfolding** *st.d2-def st.dims2-def d2-def nths-vars2′* **by** *auto*
ultimately show *?thesis* **unfolding** *ptensor-mat-def*
using *st.tensor-mat-mult assms* **by** *auto*
qed

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 −
  interpret *st*: *partial-state dims0 vars1′*.
  have *st.d1* = *d1* **unfolding** *st.d1-def st.dims1-def d1-def nths-vars1′* **by** *auto*
  moreover have *st.d2* = *d2* **unfolding** *st.d2-def st.dims2-def d2-def nths-vars2′*
by *auto*
  ultimately show *?thesis* **unfolding** *ptensor-mat-def ptensor-vec-def*
    using *st.tensor-mat-mult-vec assms* **by** *auto*
qed

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*
  by (*simp add*: *pmat-extension-def ptensor-mat-carrier*)

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*
  using *assms* **by** (*simp add*: *pmat-extension-def ptensor-mat-add*)

lemma *pmat-extension-trace*:
  assumes *m* ∈ *carrier-mat d1 d1*
  shows *trace* (*pmat-extension m*) = *d2* ∗ *trace m*
  using *assms* **by** (*simp add*: *pmat-extension-def ptensor-mat-trace*)

lemma *pmat-extension-id*:
  pmat-extension (1_m *d1*) = 1_m *d0*
  by (*simp add*: *pmat-extension-def ptensor-mat-id*)

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*
  using *assms* **by** (*simp add*: *pmat-extension-def ptensor-mat-mult*[*symmetric*])

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 −
interpret *st1*: *partial-state2 dims vars1 vars2*
apply *unfold-locales* **using** *assms* **by** *auto*
interpret *st2*: *partial-state2 dims vars2 vars1*
apply *unfold-locales* **using** *assms* **by** *auto*

have *eq1*: *partial-state.encode1 st1.dims0 st1.vars1′* = *partial-state.encode2 st2.dims0 st2.vars1′*
apply (*subst st1.ptensor-encode1-encode2*)
unfolding *st1.dims0-def st1.vars0-def st1.vars2′-def st2.dims0-def st2.vars0-def st2.vars1′-def*
by (*subgoal-tac vars1* ∪ *vars2* = *vars2* ∪ *vars1*, *auto*)
have *eq2*: *partial-state.encode2 st1.dims0 st1.vars1′* = *partial-state.encode1 st2.dims0 st2.vars1′*
apply (*subst st1.ptensor-encode2-encode1*[*symmetric*])
unfolding *st1.dims0-def st1.vars0-def st1.vars2′-def st2.dims0-def st2.vars0-def st2.vars1′-def*
by (*subgoal-tac vars1* ∪ *vars2* = *vars2* ∪ *vars1*, *auto*)

show *?thesis* **unfolding** *st1.ptensor-mat-def st2.ptensor-mat-def partial-state.tensor-mat-def*
apply (*rule cong-mat*, *auto*)
subgoal unfolding *st1.dims0-def st1.vars0-def st2.dims0-def st2.vars0-def* **by** (*subgoal-tac vars1* ∪ *vars2* = *vars2* ∪ *vars1*, *auto*)
subgoal unfolding *st1.dims0-def st1.vars0-def st2.dims0-def st2.vars0-def* **by** (*subgoal-tac vars1* ∪ *vars2* = *vars2* ∪ *vars1*, *auto*)
using *eq1 eq2* **by** *auto*
qed

Key property: associativity of tensor product

lemma *ind-in-set-mono*:
fixes *a b* :: *nat* **and** *A* :: *nat set*
assumes $a \in A$ $b \in A$ $a < b$
shows *ind-in-set A a* < *ind-in-set A b*
unfolding *ind-in-set-def*
apply (*rule psubset-card-mono*)
subgoal by *auto*
proof −
have $x \in \{i \in A.\ i < b\}$ **if** $x \in \{i \in A.\ i < a\}$ **for** *x*
using *assms that* **by** *auto*
moreover have $a \in \{i \in A.\ i < b\}$ **using** *assms* **by** *auto*
moreover have $b \notin \{i \in A.\ i < b\}$ **by** *auto*
ultimately show $\{i \in A.\ i < a\} \subset \{i \in A.\ i < b\}$ **by** *blast*
qed

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 −
have *ind-in-set A a* < *ind-in-set A b* **if** $a < b$
by (*rule ind-in-set-mono*[*OF assms*(*1*) *assms*(*2*) *that*])
moreover have *ind-in-set A b* < *ind-in-set A a* **if** $b < a$
by (*rule ind-in-set-mono*[*OF assms*(*2*) *assms*(*1*) *that*])
ultimately show *?thesis* **using** *assms*(*3*) **by** *arith*
qed

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 $a < b$
using *ind-in-set-mono ind-in-set-inj*
by (*metis assms not-less-iff-gr-or-eq*)

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* ∈ *B*. *i* < *c*} {*i* ∈ *ind-in-set A* ‘ *B*. *i* < *ind-in-set A c*}
unfolding *bij-betw-def* **apply** *auto*
proof −
show *inj-on* (*ind-in-set A*) {*i* ∈ *B*. *i* < *c*}
unfolding *inj-on-def* **apply** *auto*
using *assms*(*1*) *ind-in-set-inj* **by** *blast*
show *ind-in-set A x* < *ind-in-set A c* **if** $x \in B$ $x < c$ **for** *x*
by (*meson assms that ind-in-set-mono subsetCE*)
show *ind-in-set A x* ∈ *ind-in-set A* ‘ {*i* ∈ *B*. *i* < *c*} **if** *ind-in-set A x* < *ind-in-set A c* $x \in B$ **for** *x*
using *that ind-in-set-mono2 assms* **by** *blast*
qed

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 −
have *x* ∈ *ind-in-set* (*ind-in-set A* ‘ *B*) ‘ (*ind-in-set A* ‘ *C*) **if** *x*: *x* ∈ *ind-in-set B* ‘ *C* **for** *x*
proof −
obtain *c* **where** *c*: $c \in C$ **and** *x-eq*: *x* = *card* {*i* ∈ *B*. *i* < *c*}
using *x* **by** (*auto simp add*: *ind-in-set-def*)
have *card* {*i* ∈ *B*. *i* < *c*} = *card* {*i* ∈ *ind-in-set A* ‘ *B*. *i* < *ind-in-set A c*}
apply (*rule bij-betw-same-card*)
using *c assms* **by** (*auto intro*: *ind-in-set-bij-betw*)

then have *ind-in-set* (*ind-in-set A ‘ B*) (*ind-in-set A c*) = *x*
apply (*subst ind-in-set-def*) **using** *x-eq* **by** *auto*
then show *?thesis*
using ‹*c* ∈ *C*› **by** *blast*
qed
moreover have *x* ∈ *ind-in-set B ‘ C* **if** *x*: *x* ∈ *ind-in-set* (*ind-in-set A ‘ B*) ‘ (*ind-in-set A ‘ C*) **for** *x*
proof −
obtain *c* **where** *c*: *c* ∈ *C* **and** *x-eq*: *x* = *card* {*i* ∈ *ind-in-set A ‘ B*. *i* < *ind-in-set A c*}
using *x* **by** (*auto simp add*: *ind-in-set-def*)
have *card* {*i* ∈ *B*. *i* < *c*} = *card* {*i* ∈ *ind-in-set A ‘ B*. *i* < *ind-in-set A c*}
apply (*rule bij-betw-same-card*)
using *c assms* **by** (*auto intro*: *ind-in-set-bij-betw*)
then have *ind-in-set B c* = *x*
apply (*subst ind-in-set-def*) **using** *x-eq* **by** *auto*
then show *?thesis*
using ‹*c* ∈ *C*› **by** *blast*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *nths-reencode-eq3*:
fixes *A B C* :: *nat set*
assumes *C* ⊆ *B B* ⊆ *A*
shows *nths* (*nths xs* (*ind-in-set A ‘ B*)) (*ind-in-set B ‘ C*) = *nths xs* (*ind-in-set A ‘ C*)
apply (*subst ind-in-set-assoc*[*OF assms*, *symmetric*])
apply (*rule nths-reencode-eq*)
using *assms* **by** *blast*

lemma *nths-assoc-three-A*:
fixes *A B C* :: *nat set*
assumes *A* ∩ *B* = {}
and (*A* ∪ *B*) ∩ *C* = {}
shows *nths* (*nths xs* (*ind-in-set* (*A* ∪ *B* ∪ *C*) ‘ (*A* ∪ *B*))) (*ind-in-set* (*A* ∪ *B*) ‘ *A*)
= *nths xs* (*ind-in-set* (*A* ∪ *B* ∪ *C*) ‘ *A*)
apply (*rule nths-reencode-eq3*) **by** *auto*

lemma *nths-assoc-three-B*:
fixes *A B C* :: *nat set*
assumes *A* ∩ *B* = {}
and (*A* ∪ *B*) ∩ *C* = {}
shows *nths* (*nths xs* (*ind-in-set* (*A* ∪ *B* ∪ *C*) ‘ (*A* ∪ *B*))) (*ind-in-set* (*A* ∪ *B*) ‘ *B*)
= *nths* (*nths xs* (*ind-in-set* (*A* ∪ *B* ∪ *C*) ‘ (*B* ∪ *C*))) (*ind-in-set* (*B* ∪ *C*) ‘ *B*)
proof −

have *nths* (*nths xs* (*ind-in-set* ($A \cup B \cup C$) ‘ ($A \cup B$))) (*ind-in-set* ($A \cup B$) ‘ B) = *nths xs* (*ind-in-set* ($A \cup B \cup C$) ‘ B)
using *nths-assoc-three-A*[*of B A C xs*] *assms* **by** (*simp add*: *inf-commute sup-commute*)
moreover have *nths* (*nths xs* (*ind-in-set* ($A \cup B \cup C$) ‘ ($B \cup C$))) (*ind-in-set* ($B \cup C$) ‘ B) = *nths xs* (*ind-in-set* ($A \cup B \cup C$) ‘ B)
using *nths-assoc-three-A*[*of B C A xs*] *assms* **by** (*smt* (*verit*) *Un-empty inf-commute inf-sup-distrib2 sup-assoc sup-commute*)
ultimately show *?thesis* **by** *auto*
qed

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)
using *nths-assoc-three-A*[*of C B A xs*] *assms*
by (*smt* (*verit*) *Un-empty inf-commute inf-sup-distrib2 sup-assoc sup-commute*)

lemma *valid-index-ind-in-set*:
assumes *is* $\lhd$ *nths dims A B* $\subseteq A$
shows *nths is* (*ind-in-set A* ‘ B) $\lhd$ *nths dims B*
apply (*subst nths-reencode-eq*[*OF assms*(*2*), *symmetric*])
apply (*rule valid-index-nths*)
by (*rule assms*(*1*))

lemma *ind-in-set-id*:
fixes A :: *nat set*
assumes *finite A*
shows *ind-in-set A* ‘ $A = \{0..<$ *card A*$\}$
unfolding *ind-in-set-def* **apply** *auto*
subgoal using *assms lt-set-card-lt* **by** *auto*
proof −
fix x **assume** x: $x <$ *card A*
have ∗: *card* $\{i \in A.\ i <$ *pick A x*$\} = x$
apply (*rule card-pick-le*) **by** (*rule x*)
show $x \in (\lambda x.$ *card* $\{i \in A.\ i < x\})$ ‘ A
apply (*subst* ∗[*symmetric*])
apply (*rule imageI*)
apply (*rule pick-in-set-le*) **by** (*rule x*)
qed

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)

apply (*rule nths-split-complement-eq*[*symmetric*])
subgoal apply *auto* **using** *assms*(*1*) *ind-in-set-inj*
by (*metis disjoint-iff-not-equal subsetCE sup-ge1 sup-ge2*)
proof −
have ∗: *ind-in-set* (*A* ∪ *B*) ' *B* ∪ *ind-in-set* (*A* ∪ *B*) ' *A* = *ind-in-set* (*A* ∪ *B*) ' (*A* ∪ *B*)
by *auto*
show {*0*..<*length xs*} ⊆ *ind-in-set* (*A* ∪ *B*) ' *B* ∪ *ind-in-set* (*A* ∪ *B*) ' *A*
apply (*auto simp add*: ∗ *assms*(*2*))
using *ind-in-set-id*
by (*metis assms*(*2*) *atLeastLessThan-iff card.infinite not-le-imp-less not-less-zero*)
qed

lemma *ind-in-set-inj′*:
fixes *A B* :: *nat set*
assumes *B* ⊆ *A*
shows *inj-on* (*ind-in-set A*) *B*
proof (*rule inj-onI*)
fix *x y* **assume** *x*: *x* ∈ *B* **and** *y*: *y* ∈ *B* **and** *eq*: *ind-in-set A x* = *ind-in-set A y*
have *x′*: *x* ∈ *A* **using** *x assms* **by** *auto*
have *y′*: *y* ∈ *A* **using** *y assms* **by** *auto*
show *x* = *y* **by** (*rule ind-in-set-inj*[*OF x′ y′ eq*])
qed

lemma *ind-in-set-less*:
fixes *x* :: *nat* **and** *A* :: *nat set*
assumes *finite A x* ∈ *A*
shows *ind-in-set A x* < *card A*
unfolding *ind-in-set-def*
apply (*rule psubset-card-mono*) **using** *assms* **by** *auto*

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 −
interpret *a*: *partial-state2 dims vars1 vars2*
by (*unfold-locales*, *rule assms*(*1*))
interpret *b*: *partial-state2 dims vars1* ∪ *vars2 vars3*
by (*unfold-locales*, *rule assms*(*2*))
interpret *c*: *partial-state2 dims vars2 vars3*
apply *unfold-locales* **using** *assms*(*2*) **by** *auto*
interpret *d*: *partial-state2 dims vars1 vars2* ∪ *vars3*
apply *unfold-locales* **using** *assms* **by** *auto*

have *uassoc*: *vars1* ∪ (*vars2* ∪ *vars3*) = *vars1* ∪ *vars2* ∪ *vars3*
by *auto*

have ∗∗: {*i*. *i* < *length dims* ∧ (*i* ∈ *vars1* ∨ *i* ∈ *vars2* ∨ *i* ∈ *vars3*)} = *vars1* ∪ *vars2* ∪ *vars3*
using *assms*(*3*) **by** *auto*

have *finite-union*: *finite* (*vars1* ∪ *vars2* ∪ *vars3*)
using *assms*(*3*)
using *subset-eq-atLeast0-lessThan-finite* **by** *blast*

have *m1eq*: *digit-encode a.dims0* (*digit-decode b.dims1* (*nths* (*digit-encode b.dims0* *i*) *b.vars1′*))
= *nths* (*digit-encode b.dims0 i*) *b.vars1′* **if** *i* < *state-sig.d b.dims0* **for** *i*
unfolding *a.dims0-def a.vars0-def b.dims1-def b.dims0-def b.vars0-def b.vars1′-def*
apply (*subst digit-encode-decode*)
apply (*rule valid-index-ind-in-set*)
apply (*rule digit-encode-valid-index*)
using *that* **unfolding** *state-sig.d-def b.dims0-def b.vars0-def* **by** *auto*
have *m1index*: *partial-state.encode1 a.dims0 a.vars1′* (*partial-state.encode1 b.dims0* *b.vars1′ i*)
= *partial-state.encode1 d.dims0 d.vars1′ i* **if** *i* < *state-sig.d b.dims0* **for** *i*
unfolding *partial-state.encode1-def partial-state.dims1-def a.nths-vars1′ d.nths-vars1′* *b.nths-vars1′*
apply (*rule arg-cong*[**where** *f*=*digit-decode d.dims1*])
apply (*subst m1eq*[*OF that*])
unfolding *a.vars0-def a.vars1′-def b.dims0-def b.vars0-def b.vars1′-def d.dims0-def* *d.vars0-def d.vars1′-def*
using *nths-assoc-three-A*[*OF assms*(*1*−*2*)] **using** *uassoc* **by** *auto*

have *m2eq1*: *digit-encode a.dims0* (*digit-decode* (*nths b.dims0 b.vars1′*) (*nths* (*digit-encode b.dims0 i*) *b.vars1′*))
= *nths* (*digit-encode b.dims0 i*) *b.vars1′*
if *i* < *state-sig.d b.dims0* **for** *i*
unfolding *a.dims0-def a.vars0-def b.nths-vars1′ b.dims1-def*
apply (*subst digit-encode-decode*)
unfolding *b.vars1′-def*
apply (*rule valid-index-ind-in-set*)
unfolding *b.dims0-def*
apply (*rule digit-encode-valid-index*)
using *that* **unfolding** *state-sig.d-def b.dims0-def b.vars0-def* **by** *auto*

have *m2eq2*: *digit-encode c.dims0* (*digit-decode* (*nths d.dims0* (− *d.vars1′*)) (*nths* (*digit-encode d.dims0 i*) (− *d.vars1′*)))
= *nths* (*digit-encode d.dims0 i*) (− *d.vars1′*)
if *i* < *state-sig.d b.dims0* **for** *i*
unfolding *c.dims0-def c.vars0-def d.nths-vars2′ d.dims2-def*
apply (*subst digit-encode-decode*)
unfolding *d.vars1′-def d.vars0-def*

apply (*subst nths-complement-ind-in-set*)
subgoal using *assms* **by** *auto*
subgoal apply (*auto simp only*: *length-digit-encode d.dims0-def d.vars0-def length-nths*)
by (*auto simp add*: ∗∗ *uassoc*)
apply (*rule valid-index-ind-in-set*)
unfolding *d.dims0-def d.vars0-def*
apply (*rule digit-encode-valid-index*)
using *that* **unfolding** *state-sig.d-def b.dims0-def b.vars0-def* **using** *uassoc* **by** *auto*

have *m2index*: *partial-state.encode2 a.dims0 a.vars1′* (*partial-state.encode1 b.dims0 b.vars1′ i*) =
partial-state.encode1 c.dims0 c.vars1′ (*partial-state.encode2 d.dims0 d.vars1′ i*)
if *i* < *state-sig.d b.dims0* **for** *i*
unfolding *partial-state.encode2-def partial-state.encode1-def*
partial-state.dims2-def a.nths-vars2′ partial-state.dims1-def c.nths-vars1′
a.dims2-def c.dims1-def
apply (*rule arg-cong*[**where** *f*=*digit-decode* (*nths dims vars2*)])
apply (*subst m2eq1*[*OF that*])
apply (*subst m2eq2*[*OF that*])
unfolding *b.dims0-def b.vars0-def b.vars1′-def a.vars1′-def a.vars0-def*
d.dims0-def d.vars0-def d.vars1′-def c.vars1′-def c.vars0-def
apply (*subst nths-complement-ind-in-set*)
subgoal using *assms* **by** *auto*
subgoal apply (*auto simp only*: *length-nths length-digit-encode*)
apply (*rule bij-betw-same-card*[**where** *f*=*ind-in-set* (*vars1* ∪ *vars2* ∪ *vars3*)])
unfolding *bij-betw-def* **apply** (*rule conjI*)
subgoal apply (*rule ind-in-set-inj′*) **by** *auto*
apply *auto* **using** *finite-union* **by** (*auto simp add*: ∗∗ *intro*: *ind-in-set-less*)
apply (*subst nths-complement-ind-in-set*)
subgoal using *assms* **by** *auto*
subgoal apply (*auto simp only*: *length-digit-encode length-nths*)
by (*auto simp add*: ∗∗ *uassoc*)
using *nths-assoc-three-B*[*OF assms*(*1*−*2*)] *uassoc* **by** *auto*

have *m3eq*: *digit-encode c.dims0* (*digit-decode d.dims2* (*nths* (*digit-encode d.dims0 i*) (− *d.vars1′*)))
= *nths* (*digit-encode d.dims0 i*) (− *d.vars1′*) **if** *i* < *state-sig.d b.dims0* **for** *i*
unfolding *c.dims0-def c.vars0-def d.dims2-def d.dims0-def d.vars1′-def d.vars0-def*
apply (*subst digit-encode-decode*)
apply (*subst nths-complement-ind-in-set*)
subgoal using *assms* **by** *auto*
subgoal apply (*auto simp only*: *length-digit-encode length-nths*)
by (*auto simp add*: ∗∗ *uassoc*)
apply (*rule valid-index-ind-in-set*)
apply (*rule digit-encode-valid-index*)
using *that* **unfolding** *state-sig.d-def b.dims0-def b.vars0-def* **using** *uassoc* **by** *auto*

have *m3index*: *partial-state.encode2 c.dims0 c.vars1'* (*partial-state.encode2 d.dims0 d.vars1' i*) =
partial-state.encode2 b.dims0 b.vars1' i
if *i* < *state-sig.d b.dims0* **for** *i*
unfolding *partial-state.encode2-def partial-state.dims2-def c.nths-vars2' d.nths-vars2' b.nths-vars2'*
apply (*rule arg-cong*[**where** *f*=*digit-decode c.dims2*])
apply (*subst m3eq*[*OF that*])
unfolding *d.dims0-def d.vars0-def d.vars1'-def c.vars1'-def b.dims0-def b.vars1'-def b.vars0-def c.vars0-def*
apply (*subst nths-complement-ind-in-set*)
subgoal using *assms* **by** *auto*
subgoal apply (*auto simp only*: *length-digit-encode length-nths*)
by (*auto simp add*: ∗∗ *uassoc*)
apply (*subst nths-complement-ind-in-set*)
subgoal using *assms* **by** *auto*
subgoal apply (*auto simp only*: *length-nths length-digit-encode*)
apply (*rule bij-betw-same-card*[**where** *f*=*ind-in-set* (*vars1* ∪ *vars2* ∪ *vars3*)])
unfolding *bij-betw-def* **apply** (*rule conjI*)
subgoal apply (*rule ind-in-set-inj'*) **by** *auto*
apply (*auto simp add*: *uassoc*) **using** *finite-union*
by (*auto simp add*: ∗∗ *intro*: *ind-in-set-less*)
apply (*subst nths-complement-ind-in-set*)
subgoal using *assms* **by** *auto*
subgoal apply (*auto simp only*: *length-nths length-digit-encode*)
by (*auto simp add*: ∗∗ *uassoc*)
using *nths-assoc-three-C*[*OF assms*(*1*−*2*)] *uassoc* **by** *auto*

show *?thesis*
unfolding *a.ptensor-mat-def b.ptensor-mat-def c.ptensor-mat-def d.ptensor-mat-def partial-state.tensor-mat-def*
apply (*rule cong-mat*)
subgoal unfolding *b.dims0-def d.dims0-def b.vars0-def d.vars0-def*
apply (*subgoal-tac vars1* ∪ *vars2* ∪ *vars3* = *vars1* ∪ (*vars2* ∪ *vars3*)) **by** *auto*
subgoal unfolding *b.dims0-def d.dims0-def b.vars0-def d.vars0-def*
apply (*subgoal-tac vars1* ∪ *vars2* ∪ *vars3* = *vars1* ∪ (*vars2* ∪ *vars3*)) **by** *auto*
subgoal for *i j*
proof −
assume *lti*: *i* < *state-sig.d b.dims0* **and** *ltj*: *j* < *state-sig.d b.dims0*
have *lti'*: *i* < *state-sig.d d.dims0* **using** ‹*state-sig.d b.dims0* = *state-sig.d d.dims0*› *lti* **by** *auto*
have *ltj'*: *j* < *state-sig.d d.dims0* **using** ‹*state-sig.d b.dims0* = *state-sig.d d.dims0*› *ltj* **by** *auto*
have *eq1*: *partial-state.d2 d.dims0 d.vars1'* = *state-sig.d c.dims0*
unfolding *partial-state.d2-def partial-state.dims2-def d.nths-vars2'*
d.dims2-def state-sig.d-def c.dims0-def c.vars0-def **by** *auto*

have *eq2*: *partial-state.d1 b.dims0 b.vars1' = state-sig.d a.dims0*
  unfolding *partial-state.d1-def partial-state.dims1-def b.nths-vars1'*
    b.dims1-def state-sig.d-def a.dims0-def a.vars0-def **by** *auto*
have *lt1*: *partial-state.encode2 d.dims0 d.vars1' i < state-sig.d c.dims0*
  using *partial-state.encode2-lt*[*OF lti'*, **where** *vars=d.vars1'*] *eq1* **by** *auto*
have *lt2*: *partial-state.encode2 d.dims0 d.vars1' j < state-sig.d c.dims0*
  using *partial-state.encode2-lt*[*OF ltj'*, **where** *vars=d.vars1'*] *eq1* **by** *auto*
have *lt3*: *partial-state.encode1 b.dims0 b.vars1' i < state-sig.d a.dims0*
  using *partial-state.encode1-lt*[*OF lti*, **where** *vars=b.vars1'*] *eq2* **by** *auto*
have *lt4*: *partial-state.encode1 b.dims0 b.vars1' j < state-sig.d a.dims0*
  using *partial-state.encode1-lt*[*OF ltj*, **where** *vars=b.vars1'*] *eq2* **by** *auto*
show *?thesis*
  apply (*auto simp add*: *lt1 lt2 lt3 lt4*)
  apply (*simp only*: *m1index*[*OF lti*] *m1index*[*OF ltj*] *m2index*[*OF lti*] *m2index*[*OF ltj*] *m3index*[*OF lti*] *m3index*[*OF ltj*])
  by *auto*
qed
done
qed

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 −
  interpret *a*: *partial-state2 dims vars1 vars2*
    by (*unfold-locales*, *rule assms*(*1*))
  interpret *b*: *partial-state2 dims vars1* ∪ *vars2 vars3*
    by (*unfold-locales*, *rule assms*(*2*))
  interpret *c*: *partial-state2 dims vars2 vars3*
    apply *unfold-locales* **using** *assms*(*2*) **by** *auto*
  interpret *d*: *partial-state2 dims vars1 vars2* ∪ *vars3*
    apply *unfold-locales* **using** *assms* **by** *auto*
  have *a.d2* = *c.d1*
    by (*simp add*: *c.d1-def a.d2-def c.dims1-def a.dims2-def*)
  have *c.d0* = *d.d2*
    by (*simp add*: *c.d0-def d.d2-def c.dims0-def d.dims2-def c.vars0-def*)
  show *?thesis*
    unfolding *a.pmat-extension-def b.pmat-extension-def d.pmat-extension-def*
    apply (*simp add*: *ptensor-mat-assoc*[*OF assms*])
    apply (*simp add*: ‹*a.d2* = *c.d1*› *c.ptensor-mat-id*)
    by (*simp add*: ‹*c.d0* = *d.d2*›)
qed

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 −
have *1*: *mat-extension dims vars U* ∈ *carrier-mat d d*
by (*rule partial-state.mat-extension-carrier*)
have *2*: *unitary* (*mat-extension dims vars U*)
apply (*rule partial-state.mat-extension-unitary*)
unfolding *partial-state.d1-def partial-state.dims1-def* **using** *assms* **by** *auto*
show *well-com* (*Utrans-P vars U*)
using *1 2 Utrans-P-def* **by** *auto*
qed

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*
unfolding *hadamard-def mat-of-rows-list-def* **by** *auto*

lemma *hermitian-hadamard*:
hermitian hadamard
unfolding *hermitian-def hadamard-def*
apply (*rule eq-matI*) **by** (*auto simp add*: *adjoint-eval adjoint-dim*)

lemma *csqrt-2-sq*:
complex-of-real (*sqrt 2*) ∗ *complex-of-real* (*sqrt 2*) = *2*
by (*smt* (*verit*) *of-real-add of-real-hom.hom-one of-real-power one-add-one power2-eq-square real-sqrt-pow2*)

lemma *sum-le-2*:
⋀(*f*::*nat*⇒*complex*). *sum f* {*0*..<*2*} = *f 0* + *f 1*
by (*simp add*: *numeral-2-eq-2*)

lemma *unitary-hadamard*:
unitary hadamard
unfolding *unitary-def* **apply** (*rule*)
subgoal using *carrier-matD*[*OF hadamard-dim*] *hadamard-def* **by** *auto*
apply (*subst hermitian-hadamard*[*unfolded hermitian-def*])
unfolding *inverts-mat-def*
apply (*rule eq-matI*) **unfolding** *hadamard-def*
apply (*auto simp add*: *carrier-matD*[*OF hadamard-dim*] *scalar-prod-def*)
by (*auto simp add*: *sum-le-2 csqrt-2-sq*)

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*
unfolding *mat-incr-def* **by** *auto*

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*))

apply (*rule eq-matI*) **unfolding** *mat-incr-def*
by (*auto simp add*: *adjoint-eval*)

lemma *mat-incr-mult-adjoint-mat-incr*:
shows *mat-incr n* ∗ (*adjoint* (*mat-incr n*)) = 1_m *n*
apply (*rule eq-matI*, *simp*)
apply (*auto simp add*: *carrier-matD*[*OF mat-incr-dim*] *scalar-prod-def*)
unfolding *adjoint-mat-incr* **unfolding** *mat-incr-def*
apply (*simp-all*)
apply (*case-tac j* = *0*)
subgoal for *j* **by** (*simp add*: *sum-only-one-neq-0*[*of - n* − *Suc 0*])
subgoal for *j* **by** (*simp add*: *sum-only-one-neq-0*[*of - j* − *1*])
done

lemma *unitary-mat-incr*:
unitary (*mat-incr n*)
unfolding *unitary-def inverts-mat-def*
using *carrier-matD*[*OF mat-incr-dim*] *mat-incr-mult-adjoint-mat-incr* **by** *auto*

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*
by (*simp add*: *denote-dim-pdo density-states-def*)

definition *is-quantum-predicate* :: *complex mat* ⇒ *bool* **where**
is-quantum-predicate P ⟷ *P* ∈ *carrier-mat d d* ∧ *positive P* ∧ $P \le_L 1_m$ *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 −
from *m* **have** *dM0*: *M 0* ∈ *carrier-mat n n* **and** *dM1*: *M 1* ∈ *carrier-mat n n*
and *id*: *adjoint* (*M 0*) ∗ (*M 0*) + *adjoint* (*M 1*) ∗ (*M 1*) = 1_m *n*
using *measurement-def measurement-id2* **by** *auto*
have *trace* (*M 1* ∗ *A* ∗ *adjoint* (*M 1*)) + *trace* (*M 0* ∗ *A* ∗ *adjoint* (*M 0*))

= *trace* ((*adjoint* (*M 0*) ∗ *M 0* + *adjoint* (*M 1*) ∗ *M 1*) ∗ *A*)
using *dM0 dM1 dA* **by** (*mat-assoc n*)
also have . . . = *trace* (1_m *n* ∗ *A*) **using** *id* **by** *auto*
also have . . . = *trace A* **using** *dA* **by** *auto*
finally show *?thesis*
using *dA dM0 dM1 local.id state-sig.trace-measure2-id* **by** *blast*
qed

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*)
unfolding *is-quantum-predicate-def*
proof (*auto*)
have *dP*: *P* ∈ *carrier-mat d d* **using** *qp is-quantum-predicate-def* **by** *auto*
show *adjoint U* ∗ *P* ∗ *U* ∈ *carrier-mat d d* **using** *dU dP* **by** *fastforce*
have *positive P* **using** *qp is-quantum-predicate-def* **by** *auto*
then show *positive* (*adjoint U* ∗ *P* ∗ *U*)
using *positive-close-under-left-right-mult-adjoint*[*OF adjoint-dim*[*OF dU*] *dP*, *simplified adjoint-adjoint*] **by** *fastforce*
have *adjoint U* ∗ *U* = 1_m *d* **apply** (*subgoal-tac inverts-mat* (*adjoint U*) *U*)
subgoal unfolding *inverts-mat-def* **using** *dU* **by** *auto*
using *u* **unfolding** *unitary-def* **using** *inverts-mat-symm*[*OF dU adjoint-dim*[*OF dU*]] **by** *auto*
then have *u'*: *adjoint U* ∗ 1_m *d* ∗ *U* = 1_m *d* **using** *dU* **by** *auto*
have *le*: *P* $\leq_L$ 1_m *d* **using** *qp is-quantum-predicate-def* **by** *auto*
show *adjoint U* ∗ *P* ∗ *U* $\leq_L$ 1_m *d*
using *lowner-le-keep-under-measurement*[*OF dU dP one-carrier-mat le*] *u'* **by** *auto*
qed

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*)
unfolding *is-quantum-predicate-def*
proof (*auto*)
have *dMk*: *k* < *n* ⟹ *M k* ∈ *carrier-mat d d* **for** *k* **using** *m measurement-def* **by** *auto*
moreover have *dPk*: *k* < *n* ⟹ *P k* ∈ *carrier-mat d d* **for** *k* **using** *qpk is-quantum-predicate-def* **by** *auto*
ultimately have *dk*: *k* < *n* ⟹ *adjoint* (*M k*) ∗ *P k* ∗ *M k* ∈ *carrier-mat d d* **for** *k* **by** *fastforce*
then show *d*: *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n* ∈ *carrier-mat d d*
using *matrix-sum-dim*[*of n* λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*] **by** *auto*
have *k* < *n* ⟹ *positive* (*P k*) **for** *k* **using** *qpk is-quantum-predicate-def* **by** *auto*

then have *k* < *n* ⟹ *positive* (*adjoint* (*M k*) ∗ *P k* ∗ *M k*) **for** *k*
using *positive-close-under-left-right-mult-adjoint*[*OF adjoint-dim*[*OF dMk*] *dPk*, *simplified adjoint-adjoint*] **by** *fastforce*
then show *positive* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n*) **using** *matrix-sum-positive dk* **by** *auto*
have *k* < *n* ⟹ *P k* $\leq_L$ *1*$_m$ *d* **for** *k* **using** *qpk is-quantum-predicate-def* **by** *auto*
then have *k* < *n* ⟹ *positive* (*1*$_m$ *d* − *P k*) **for** *k* **using** *lowner-le-def* **by** *auto*
then have *p*: *k* < *n* ⟹ *positive* (*adjoint* (*M k*) ∗ (*1*$_m$ *d* − *P k*) ∗ *M k*) **for** *k*
using *positive-close-under-left-right-mult-adjoint*[*OF adjoint-dim*[*OF dMk*], *simplified adjoint-adjoint*, *of - 1*$_m$ *d* − *P k*] *dPk* **by** *fastforce*
{
fix *k* **assume** *k*: *k* < *n*
have *adjoint* (*M k*) ∗ (*1*$_m$ *d* − *P k*) ∗ *M k* = *adjoint* (*M k*) ∗ *M k* − *adjoint* (*M k*) ∗ *P k* ∗ *M k*
apply (*mat-assoc d*) **using** *dMk dPk k* **by** *auto*
}
note *split* = *this*
have *dk'*: *k* < *n* ⟹ *adjoint* (*M k*) ∗ *M k* − *adjoint* (*M k*) ∗ *P k* ∗ *M k* ∈ *carrier-mat d d* **for** *k* **using** *dMk dPk* **by** *fastforce*
have *k* < *n* ⟹ *positive* (*adjoint* (*M k*) ∗ *M k* − *adjoint* (*M k*) ∗ *P k* ∗ *M k*) **for** *k* **using** *p split* **by** *auto*
then have *p'*: *positive* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *M k* − *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n*)
using *matrix-sum-positive*[*OF dk'*, *of n id*, *simplified*] **by** *auto*
have *daMMk*: *k* < *n* ⟹ *adjoint* (*M k*) ∗ *M k* ∈ *carrier-mat d d* **for** *k* **using** *dMk* **by** *fastforce*
have *daMPMk*: *k* < *n* ⟹ *adjoint* (*M k*) ∗ *P k* ∗ *M k* ∈ *carrier-mat d d* **for** *k* **using** *dMk dPk* **by** *fastforce*
have *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *M k* − *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n*
= *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *M k*) *n* − *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n*
using *matrix-sum-minus-distrib*[*OF daMMk daMPMk*] **by** *auto*
also have ... = *1*$_m$ *d* − *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n* **using** *m measurement-def* **by** *auto*
finally have *positive* (*1*$_m$ *d* − *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n*) **using** *p'* **by** *auto*
then show *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n* $\leq_L$ *1*$_m$ *d* **using** *lowner-le-def d* **by** *auto*
qed

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 −
  have *trace* (*P* ∗ ϱ) ≤ *trace* (*Q* ∗ *denote S* ϱ) + (*trace* ϱ − *trace* (*denote S* ϱ)) **if** ϱ: ϱ ∈ *density-states* **for** ϱ
  proof −
    have *trace* (*P* ∗ ϱ) ≤ *trace* (*Q* ∗ *denote S* ϱ)
      using *total hoare-total-correct-def* ϱ **by** *auto*
    moreover have *trace* (*denote S* ϱ) ≤ *trace* ϱ
      using *denote-trace*[*OF S*] ϱ *density-states-def* **by** *auto*
    ultimately show *?thesis* **by** (*auto simp*: *less-eq-complex-def*)
  qed
  then show *?thesis*
    using *hoare-partial-correct-def* **by** *auto*
qed

lemma *predicate-prob-positive*:
  assumes 0_m *d d* $\leq_L$ *P*
    and ϱ ∈ *density-states*
  shows *0* ≤ *trace* (*P* ∗ ϱ)
proof −
  have *trace* (0_m *d d* ∗ ϱ) ≤ *trace* (*P* ∗ ϱ)
    apply (*rule lowner-le-traceD*)
    using *assms* **unfolding** *lowner-le-def density-states-def* **by** *auto*
  then show *?thesis*
    using *assms*(*2*) *density-states-def* **by** *auto*
qed

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 −
  have *trace* (0_m *d d* ∗ ϱ) ≤ *trace* (*Q* ∗ *denote S* ϱ) **if** ϱ: ϱ ∈ *density-states* **for** ϱ
  proof −
    have *1*: *trace* (0_m *d d* ∗ ϱ) = *0*
      using ϱ **unfolding** *density-states-def* **by** *auto*
    show *?thesis*
      apply (*subst 1*)
      apply (*rule predicate-prob-positive*)
      subgoal apply (*simp add*: *lowner-le-def*, *subgoal-tac Q* − 0_m *d d* = *Q*) **using** *Q is-quantum-predicate-def*[*of Q*] **by** *auto*
      subgoal using *denote-density-states* ϱ *S* **by** *auto*
      done
  qed

then show *?thesis*
using *hoare-total-correct-def* **by** *auto*
qed

lemma *partial-post-identity*:
assumes *S*: *well-com S*
and *P*: *is-quantum-predicate P*
shows $\models_p$ $\{P\}$ *S* $\{1_m\ d\}$
proof −
have *trace* $(P * \varrho) \leq$ *trace* $(1_m\ d\ *$ *denote S* $\varrho)$ + (*trace* ϱ − *trace* (*denote S* ϱ)) **if** ϱ: $\varrho \in$ *density-states* **for** ϱ
proof −
have *denote S* $\varrho \in$ *carrier-mat d d*
using *S denote-dim* ϱ *density-states-def* **by** *auto*
then have *trace* $(1_m\ d\ *$ *denote S* $\varrho)$ = *trace* (*denote S* ϱ)
by *auto*
moreover have *trace* $(P * \varrho) \leq$ *trace* $(1_m\ d * \varrho)$
apply (*rule lowner-le-traceD*)
using ϱ **unfolding** *density-states-def* **apply** *auto*
using *P is-quantum-predicate-def* **by** *auto*
ultimately show *?thesis*
using *density-states-def that* **by** *auto*
qed
then show *?thesis*
using *hoare-partial-correct-def* **by** *auto*
qed

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 −
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
have *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d* **using** *m M0-def M1-def measurement-def* **by** *auto*

have *adjoint M1* $*$ *Q* $*$ *M1* $\leq_L$ *adjoint M1* $*$ 1_m *d* $*$ *M1*
using *lowner-le-keep-under-measurement*[*OF dM1 dQ - leQ*] **by** *auto*
then have *le1*: *adjoint M1* $*$ *Q* $*$ *M1* $\leq_L$ *adjoint M1* $*$ *M1* **using** *dM1 dQ* **by** *fastforce*
have *adjoint M0* $*$ *P* $*$ *M0* $\leq_L$ *adjoint M0* $*$ 1_m *d* $*$ *M0*
using *lowner-le-keep-under-measurement*[*OF dM0 dP - leP*] **by** *auto*
then have *le0*: *adjoint M0* $*$ *P* $*$ *M0* $\leq_L$ *adjoint M0* $*$ *M0*
using *dM0 dP* **by** *fastforce*
have *adjoint M0* $*$ *P* $*$ *M0* $+$ *adjoint M1* $*$ *Q* $*$ *M1* $\leq_L$ *adjoint M0* $*$ *M0* $+$ *adjoint M1* $*$ *M1*
apply (*rule lowner-le-add*[*of adjoint M0* $*$ *P* $*$ *M0 d adjoint M0* $*$ *M0 adjoint M1* $*$ *Q* $*$ *M1 adjoint M1* $*$ *M1*])
using *dM0 dP dM1 dQ le0 le1* **by** *auto*
also have ... = 1_m *d* **using** *m M0-def M1-def measurement-id2* **by** *auto*
finally show *adjoint M0* $*$ *P* $*$ *M0* $+$ *adjoint M1* $*$ *Q* $*$ *M1* $\leq_L 1_m\ d$**.**
qed

lemma *wlp-while-n-close*:
assumes *close*: $\bigwedge P$. *is-quantum-predicate P* $\Longrightarrow$ *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 (*induct k*)
case *0*
then show *?case*
unfolding *wlp-while-n.simps is-quantum-predicate-def* **using** *positive-one*[*of d*] *lowner-le-refl*[*of* 1_m *d*] **by** *fastforce*
next
case (*Suc k*)
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
define *W* **where** *W k* = *wlp-while-n M0 M1 WS k P* **for** *k*
show *?case* **unfolding** *wlp-while-n.simps is-quantum-predicate-def*
proof (*fold M0-def M1-def*, *fold W-def*, *auto*)
have *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d* **using** *m M0-def M1-def measurement-def* **by** *auto*
have *dP*: *P* ∈ *carrier-mat d d* **using** *qpP is-quantum-predicate-def* **by** *auto*
have *qpWk*: *is-quantum-predicate* (*W k*) **using** *Suc M0-def M1-def W-def* **by** *auto*
then have *qpWWk*: *is-quantum-predicate* (*WS* (*W k*)) **using** *close* **by** *auto*
from *qpWk* **have** *dWk*: *W k* ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*

from *qpWWk* **have** *dWWk*: *WS* (*W k*) ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
show *adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *WS* (*W k*) ∗ *M1* ∈ *carrier-mat d d* **using** *dM0 dP dM1 dWWk* **by** *auto*

have *pP*: *positive P* **using** *qpP is-quantum-predicate-def* **by** *auto*
then have *pM0P*: *positive* (*adjoint M0* ∗ *P* ∗ *M0*)
using *positive-close-under-left-right-mult-adjoint*[*OF adjoint-dim*[*OF dM0*]] *dM0 dP adjoint-adjoint*[*of M0*] **by** *auto*
have *pWWk*: *positive* (*WS* (*W k*)) **using** *qpWWk is-quantum-predicate-def* **by** *auto*
then have *pM1WWk*: *positive* (*adjoint M1* ∗ *WS* (*W k*) ∗ *M1*)
using *positive-close-under-left-right-mult-adjoint*[*OF adjoint-dim*[*OF dM1*]] *dM1 dWWk adjoint-adjoint*[*of M1*] **by** *auto*
then show *positive* (*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *WS* (*W k*) ∗ *M1*)
using *positive-add*[*OF pM0P pM1WWk*] *dM0 dP dM1 dWWk* **by** *fastforce*

have *leWWk*: *WS* (*W k*) $\leq_L$ 1_m *d* **using** *qpWWk is-quantum-predicate-def* **by** *auto*
have *leP*: *P* $\leq_L$ 1_m *d* **using** *qpP is-quantum-predicate-def* **by** *auto*
show (*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *WS* (*W k*) ∗ *M1*) $\leq_L$ 1_m *d*
using *measurement2-leq-one-mat*[*OF dP dWWk leP leWWk m*] *M0-def M1-def* **by** *auto*
qed
qed

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 (*induct k*)
case *0*
then show *?case* **unfolding** *wlp-while-n.simps* **using** *lowner-le-refl*[*of* 1_m *d*] **by** *fastforce*
next
case (*Suc k*)
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
have *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d* **using** *assms M0-def M1-def measurement-def* **by** *auto*
define *W* **where** *W P k* = *wlp-while-n M0 M1 WS k P* **for** *k P*

have *dP*: *P* ∈ *carrier-mat d d* **and** *dQ*: *Q* ∈ *carrier-mat d d* **using** *assms* *is-quantum-predicate-def* **by** *auto*

have *eq1*: *W P* (*Suc k*) = *adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ (*WS* (*W P k*)) ∗ *M1* **unfolding** *W-def wlp-while-n.simps* **by** *auto*
have *eq2*: *W Q* (*Suc k*) = *adjoint M0* ∗ *Q* ∗ *M0* + *adjoint M1* ∗ (*WS* (*W Q k*)) ∗ *M1* **unfolding** *W-def wlp-while-n.simps* **by** *auto*
have *le1*: *adjoint M0* ∗ *P* ∗ *M0* $\leq_L$ *adjoint M0* ∗ *Q* ∗ *M0* **using** *lowner-le-keep-under-measurement dM0 dP dQ assms* **by** *auto*
have *leWk*: (*W P k*) $\leq_L$ (*W Q k*) **unfolding** *W-def M0-def M1-def* **using** *Suc* **by** *auto*
have *qpWPk*: *is-quantum-predicate* (*W P k*) **unfolding** *W-def M0-def M1-def* **using** *wlp-while-n-close assms* **by** *auto*
then have *is-quantum-predicate* (*WS* (*W P k*)) **unfolding** *W-def M0-def M1-def* **using** *assms* **by** *auto*
then have *dWWPk*: (*WS* (*W P k*)) ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
have *qpWQk*: *is-quantum-predicate* (*W Q k*) **unfolding** *W-def M0-def M1-def* **using** *wlp-while-n-close assms* **by** *auto*
then have *is-quantum-predicate* (*WS* (*W Q k*)) **unfolding** *W-def M0-def M1-def* **using** *assms* **by** *auto*
then have *dWWQk*: (*WS* (*W Q k*)) ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*

have (*WS* (*W P k*)) $\leq_L$ (*WS* (*W Q k*)) **using** *qpWPk qpWQk leWk assms* **by** *auto*
then have *le2*: *adjoint M1* ∗ (*WS* (*W P k*)) ∗ *M1* $\leq_L$ *adjoint M1* ∗ (*WS* (*W Q k*)) ∗ *M1*
using *lowner-le-keep-under-measurement dM1 dWWPk dWWQk* **by** *auto*

have (*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ (*WS* (*W P k*)) ∗ *M1*) $\leq_L$ (*adjoint M0* ∗ *Q* ∗ *M0* + *adjoint M1* ∗ (*WS* (*W Q k*)) ∗ *M1*)
using *lowner-le-add*[*OF - - - - le1 le2*] *dM0 dP dM1 dQ dWWPk dWWQk le1 le2* **by** *fastforce*

then have *W P* (*Suc k*) $\leq_L$ *W Q* (*Suc k*) **using** *eq1 eq2* **by** *auto*
then show *?case* **unfolding** *W-def M0-def M1-def* **by** *auto*
qed

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 (*auto*)
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
have *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d* **using** *assms M0-def M1-def measurement-def* **by** *auto*
define *W* **where** *W k* = *wlp-while-n M0 M1 WS k P* **for** *k*
have *leP*: *P* $\leq_L$ 1_m *d* **and** *dP*: *P* ∈ *carrier-mat d d* **and** *pP*: *positive P* **using** *qpP is-quantum-predicate-def* **by** *auto*
have *pM0P*: *positive* (*adjoint M0* ∗ *P* ∗ *M0*)
using *positive-close-under-left-right-mult-adjoint*[*OF adjoint-dim*[*OF dM0*]] *adjoint-adjoint*[*of M0*] *dP pP* **by** *auto*

have *le-qp*: *W* (*Suc k*) $\leq_L$ *W k* ∧ *is-quantum-predicate* (*W k*) **for** *k*
proof (*induct k*)
case *0*
have *is-quantum-predicate* (1_m *d*)
unfolding *is-quantum-predicate-def* **using** *positive-one lowner-le-refl*[*of* 1_m *d*] **by** *fastforce*
then have *is-quantum-predicate* (*WS* (1_m *d*)) **using** *assms* **by** *auto*
then have (*WS* (1_m *d*)) ∈ *carrier-mat d d* **and** (*WS* (1_m *d*)) $\leq_L$ 1_m *d* **using** *is-quantum-predicate-def* **by** *auto*
then have *W 1* $\leq_L$ *W 0* **unfolding** *W-def wlp-while-n.simps M0-def M1-def*
using *measurement2-leq-one-mat*[*OF dP - leP - m*] **by** *auto*
moreover have *is-quantum-predicate* (*W 0*) **unfolding** *W-def wlp-while-n.simps is-quantum-predicate-def*
using *positive-one lowner-le-refl*[*of* 1_m *d*] **by** *fastforce*
ultimately show *?case* **by** *auto*
next
case (*Suc k*)
then have *leWSk*: *W* (*Suc k*) $\leq_L$ *W k* **and** *qpWk*: *is-quantum-predicate* (*W k*) **by** *auto*
then have *is-quantum-predicate* (*WS* (*W k*)) **using** *assms* **by** *auto*
then have *dWWk*: *WS* (*W k*) ∈ *carrier-mat d d* **and** *leWWk1*: (*WS* (*W k*)) $\leq_L$ 1_m *d* **and** *pWWk*: *positive* (*WS* (*W k*))
using *is-quantum-predicate-def* **by** *auto*
then have *leWSk1*: *W* (*Suc k*) $\leq_L$ 1_m *d* **using** *measurement2-leq-one-mat*[*OF dP dWWk leP leWWk1 m*]
unfolding *W-def wlp-while-n.simps M0-def M1-def* **by** *auto*
then have *dWSk*: *W* (*Suc k*) ∈ *carrier-mat d d* **using** *lowner-le-def* **by** *fastforce*
have *pM1WWk*: *positive* (*adjoint M1* ∗ (*WS* (*W k*)) ∗ *M1*)
using *positive-close-under-left-right-mult-adjoint*[*OF adjoint-dim*[*OF dM1*] *dWWk pWWk*] *adjoint-adjoint*[*of M1*] **by** *auto*

have *pWSk*: *positive* (*W* (*Suc k*)) **unfolding** *W-def wlp-while-n.simps* **apply** (*fold W-def*)
using *positive-add*[*OF pM0P pM1WWk*] *dM0 dP dM1* **by** *fastforce*
have *qpWSk*:*is-quantum-predicate* (*W* (*Suc k*)) **unfolding** *is-quantum-predicate-def* **using** *dWSk pWSk leWSk1* **by** *auto*
then have *qpWWSk*: *is-quantum-predicate* (*WS* (*W* (*Suc k*))) **using** *assms*(*1*) **by** *auto*
then have *dWWSk*: (*WS* (*W* (*Suc k*))) ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*

have *WS* (*W* (*Suc k*)) $\leq_L$ *WS* (*W k*) **using** *assms*(*2*)[*OF qpWSk qpWk*] *leWSk* **by** *auto*
then have *adjoint M1* ∗ *WS* (*W* (*Suc k*)) ∗ *M1* $\leq_L$ *adjoint M1* ∗ *WS* (*W k*) ∗ *M1*
using *lowner-le-keep-under-measurement*[*OF dM1 dWWSk dWWk*] **by** *auto*
then have (*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *WS* (*W* (*Suc k*)) ∗ *M1*)
$\leq_L$ (*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *WS* (*W k*) ∗ *M1*)
using *lowner-le-add*[*of - d - adjoint M1* ∗ *WS* (*W* (*Suc k*)) ∗ *M1 adjoint M1* ∗ *WS* (*W k*) ∗ *M1*,
OF - - - - lowner-le-refl[*of adjoint M0* ∗ *P* ∗ *M0*]] *dM0 dM1 dP dWWSk dWWk* **by** *fastforce*
then have *W* (*Suc* (*Suc k*)) $\leq_L$ *W* (*Suc k*) **unfolding** *W-def wlp-while-n.simps* **by** *auto*
with *qpWSk* **show** *?case* **by** *auto*
qed
then have *dWk*: *W k* ∈ *carrier-mat d d* **for** *k* **using** *is-quantum-predicate-def* **by** *auto*
then have *dmWk*: − *W k* ∈ *carrier-mat d d* **for** *k* **by** *auto*
have *incmWk*: − (*W k*) $\leq_L$ − (*W* (*Suc k*)) **for** *k* **using** *lowner-le-swap*[*of W* (*Suc k*) *d W k*] *dWk le-qp* **by** *auto*
have *pWk*: *positive* (*W k*) **for** *k* **using** *le-qp is-quantum-predicate-def* **by** *auto*
have *ubmWk*: − *W k* $\leq_L$ 0_m *d d* **for** *k*
proof −
have 0_m *d d* $\leq_L$ *W k* **for** *k* **using** *zero-lowner-le-positiveI dWk pWk* **by** *auto*
then have − *W k* $\leq_L$ − 0_m *d d* **for** *k* **using** *lowner-le-swap*[*of* 0_m *d d d W k*] *dWk* **by** *auto*
moreover have (− 0_m *d d* :: *complex mat*) = (0_m *d d*) **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

have ∃ *B*. *lowner-is-lub* (λ*k*. − *W k*) *B* ∧ *limit-mat* (λ*k*. − *W k*) *B d*
using *mat-inc-seq-lub*[*of* λ*k*. − *W k d* 0_m *d d*] *dmWk incmWk ubmWk* **by** *auto*
then obtain *B* **where** *lubB*: *lowner-is-lub* (λ*k*. − *W k*) *B* **and** *limB*: *limit-mat* (λ*k*. − *W k*) *B d* **by** *auto*
then have *dB*: *B* ∈ *carrier-mat d d* **using** *limit-mat-def* **by** *auto*
define *A* **where** *A* = − *B*
then have *dA*: *A* ∈ *carrier-mat d d* **using** *dB* **by** *auto*
have *limit-mat* (λ*k*. (−*1*) $\cdot_m$ (− *W k*)) (−*1* $\cdot_m$ *B*) *d* **using** *limit-mat-scale*[*OF limB*] **by** *auto*

moreover have *W k* = −*1* $\cdot_m$ (− *W k*) **for** *k* **using** *dWk* **by** *auto*
moreover have −*1* $\cdot_m$ *B* = − *B* **by** *auto*
ultimately have *limA*: *limit-mat W A d* **using** *A-def* **by** *auto*
moreover have (*limit-mat W A′ d* ⟹ *A′* = *A*) **for** *A′* **using** *limit-mat-unique*[*of W A d*] *limA* **by** *auto*
ultimately have *eqA*: (*wlp-while* (*M 0*) (*M 1*) *WS P*) = *A* **unfolding** *wlp-while-def W-def M0-def M1-def*
using *the-equality*[*of* λ*X*. *limit-mat* (λ*n*. *wlp-while-n* (*M 0*) (*M 1*) *WS n P*) *X d A*] **by** *fastforce*

show *limit-mat* (λ*n*. *wlp-while-n* (*M 0*) (*M* (*Suc 0*)) *WS n P*) (*wlp-while* (*M 0*) (*M* (*Suc 0*)) *WS P*) *d*
using *limA eqA* **unfolding** *W-def M0-def M1-def* **by** *auto*

have − *W k* $\le_L$ *B* **for** *k* **using** *lubB lowner-is-lub-def* **by** *auto*
then have *glbA*: *A* $\le_L$ *W k* **for** *k* **unfolding** *A-def* **using** *lowner-le-swap*[*of* − *W k d*] *dB dWk* **by** *fastforce*
then show ⋀*n*. *wlp-while* (*M 0*) (*M* (*Suc 0*)) *WS P* $\le_L$ *wlp-while-n* (*M 0*) (*M* (*Suc 0*)) *WS n P* **using** *eqA* **unfolding** *W-def M0-def M1-def* **by** *auto*

have *W k* $\le_L$ 1_m *d* **for** *k* **using** *le-qp* **unfolding** *is-quantum-predicate-def* **by** *auto*
then have *positive* (1_m *d* − *W k*) **for** *k* **using** *lowner-le-def* **by** *auto*
moreover have *limit-mat* (λ*k*. 1_m *d* − *W k*) (1_m *d* − *A*) *d* **using** *mat-minus-limit limA* **by** *auto*
ultimately have *positive* (1_m *d* − *A*) **using** *pos-mat-lim-is-pos* **by** *auto*
then have *leA1*: *A* $\le_L$ 1_m *d* **using** *dA lowner-le-def* **by** *auto*

have *pA*: *positive A* **using** *pos-mat-lim-is-pos limA pWk* **by** *auto*

show *is-quantum-predicate* (*wlp-while* (*M 0*) (*M* (*Suc 0*)) *WS P*) **unfolding** *is-quantum-predicate-def* **using** *pA dA leA1 eqA* **by** *auto*

{
fix *W′* **assume** *asmW′*: ∀ *k*. *W′* $\le_L$ *W k*
then have *dW′*: *W′* ∈ *carrier-mat d d* **unfolding** *lowner-le-def* **using** *carrier-matD*[*OF dWk*] **by** *auto*
then have − *W k* $\le_L$ − *W′* **for** *k* **using** *lowner-le-swap dWk asmW′* **by** *auto*
then have *B* $\le_L$ − *W′* **using** *lubB* **unfolding** *lowner-is-lub-def* **by** *auto*
then have *W′* $\le_L$ *A* **unfolding** *A-def*
using *lowner-le-swap*[*of B d* − *W′*] *dB dW′* **by** *auto*
then have *W′* $\le_L$ *wlp-while* (*M 0*) (*M 1*) *WS P* **using** *eqA* **by** *auto*
}
then show ⋀*W′*. ∀ *n*. *W′* $\le_L$ *wlp-while-n* (*M 0*) (*M* (*Suc 0*)) *WS n P* ⟹ *W′* $\le_L$ *wlp-while* (*M 0*) (*M* (*Suc 0*)) *WS P*
unfolding *W-def M0-def M1-def* **by** *auto*
qed

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 −
define *M0* **where** *M0 = M 0*
define *M1* **where** *M1 = M 1*
have *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d* **using** *assms M0-def M1-def measurement-def* **by** *auto*
define *Wn* **where** *Wn P k = wlp-while-n M0 M1 WS k P* **for** *P k*
define *W* **where** *W P = wlp-while M0 M1 WS P* **for** *P*
have *lePQk*: *Wn P k* $\leq_L$ *Wn Q k* **for** *k* **unfolding** *Wn-def M0-def M1-def*
using *wlp-while-n-mono assms* **by** *auto*
have *is-quantum-predicate* (*Wn P k*) **for** *k* **unfolding** *Wn-def M0-def M1-def* **using** *wlp-while-n-close assms* **by** *auto*
then have *dWPk*: *Wn P k* ∈ *carrier-mat d d* **for** *k* **using** *is-quantum-predicate-def* **by** *auto*
have *is-quantum-predicate* (*Wn Q k*) **for** *k* **unfolding** *Wn-def M0-def M1-def* **using** *wlp-while-n-close assms* **by** *auto*
then have *dWQk*:*Wn Q k* ∈ *carrier-mat d d* **for** *k* **using** *is-quantum-predicate-def* **by** *auto*
have *is-quantum-predicate* (*W P*) **and** *lePk*: (*W P*) $\leq_L$ (*Wn P k*) **and** *limit-mat* (*Wn P*) (*W P*) *d* **for** *k*
unfolding *W-def Wn-def M0-def M1-def* **using** *wlp-while-exists assms* **by** *auto*
then have *dWP*: *W P* ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
have *is-quantum-predicate* (*W Q*) **and** (*W Q*) $\leq_L$ (*Wn Q k*)
and *glb*:(∀ *k*. *W′* $\leq_L$ (*Wn Q k*)) ⟶ *W′* $\leq_L$ (*W Q*) **and** *limit-mat* (*Wn Q*) (*W Q*) *d* **for** *k W′*
unfolding *W-def Wn-def M0-def M1-def* **using** *wlp-while-exists assms* **by** *auto*

have *W P* $\leq_L$ *Wn Q k* **for** *k* **using** *lowner-le-trans*[*of W P d Wn P k Wn Q k*] *lePk lePQk dWPk dWQk dWP* **by** *auto*
then show *W P* $\leq_L$ *W Q* **using** *glb* **by** *auto*
qed

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* ≤ *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*
unfolding *wlp.simps wlp-measure-def*
proof −
have *k* < *m* ⟹ *map wlp S* ! *k* = *wlp* (*S*!*k*) **for** *k* **using** *wc m* **by** *auto*
then have *k* < *m* ⟹ (*map wlp S* ! *k*) *P* = *wlp* (*S*!*k*) *P* **for** *k* **by** *auto*
then show *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ ((*map wlp S* ! *k*) *P*) ∗ (*M k*)) *m*
= *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ (*M k*)) *m*
using *matrix-sum-cong*[*of m* λ*k*. *adjoint* (*M k*) ∗ ((*map wlp S* ! *k*) *P*) ∗ (*M k*) λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ (*M k*)] **by** *auto*
qed

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*
using *wlp-measure-expand-m*[*OF Nat.le-refl*[*of n*]] *wc* **by** *auto*

lemma *wlp-mono-and-close*:
shows *well-com S* ⟹ *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ $P \le_L Q$
⟹ *is-quantum-predicate* (*wlp S P*) ∧ *wlp S P* $\le_L$ *wlp S Q*
proof (*induct S arbitrary*: *P Q*)
case *SKIP*
then show *?case* **by** *auto*
next
case (*Utrans U*)
then have *dU*: *U* ∈ *carrier-mat d d* **and** *u*: *unitary U* **and** *qp*: *is-quantum-predicate P* **and** *le*: $P \le_L Q$
and *dP*: *P* ∈ *carrier-mat d d* **and** *dQ*: *Q* ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
then have *qp'*: *is-quantum-predicate* (*wlp* (*Utrans U*) *P*) **using** *qp-close-under-unitary-operator* **by** *auto*
moreover have *adjoint U* ∗ *P* ∗ *U* $\le_L$ *adjoint U* ∗ *Q* ∗ *U* **using** *lowner-le-keep-under-measurement*[*OF dU dP dQ le*] **by** *auto*
ultimately show *?case* **by** *auto*
next
case (*Seq S1 S2*)
then have *qpP*: *is-quantum-predicate P* **and** *qpQ*: *is-quantum-predicate Q* **and** *wc1*: *well-com S1* **and** *wc2*: *well-com S2*
and *dP*: *P* ∈ *carrier-mat d d* **and** *dQ*: *Q* ∈ *carrier-mat d d* **and** *le*: $P \le_L Q$**using** *is-quantum-predicate-def* **by** *auto*
have *qpP2*: *is-quantum-predicate* (*wlp S2 P*) **using** *Seq qpP wc2* **by** *auto*
have *qpQ2*: *is-quantum-predicate* (*wlp S2 Q*) **using** *Seq*(*2*)[*OF wc2 qpQ qpQ*] *lowner-le-refl dQ* **by** *blast*
have *qpP1*: *is-quantum-predicate* (*wlp S1* (*wlp S2 P*))
using *Seq*(*1*)[*OF wc1 qpP2 qpP2*] *qpP2 is-quantum-predicate-def*[*of wlp S2 P*] *lowner-le-refl* **by** *auto*
have *wlp S2 P* $\le_L$ *wlp S2 Q* **using** *Seq*(*2*) *wc2 qpP qpQ le* **by** *auto*

then have *wlp S1* (*wlp S2 P*) $\leq_L$ *wlp S1* (*wlp S2 Q*) **using** *Seq*(*1*) *wc1 qpP2 qpQ2* **by** *auto*
then show *?case* **using** *qpP1* **by** *auto*
next
case (*Measure n M S*)
then have *wc*: *well-com* (*Measure n M S*) **and** *wck*: $k < n \Longrightarrow$ *well-com* (*S*!*k*) **and** *l*: *length S* = *n*
and *m*: *measurement d n M* **and** *le*: $P \leq_L Q$
and *qpP*: *is-quantum-predicate P* **and** *dP*: $P \in$ *carrier-mat d d*
and *qpQ*: *is-quantum-predicate Q* **and** *dQ*: $Q \in$ *carrier-mat d d*
for *k* **using** *measure-well-com is-quantum-predicate-def* **by** *auto*
have *dMk*: $k < n \Longrightarrow$ *M k* $\in$ *carrier-mat d d* **for** *k* **using** *m measurement-def* **by** *auto*
have *set*: $k < n \Longrightarrow$ *S*!*k* $\in$ *set S* **for** *k* **using** *l* **by** *auto*
have *qpk*: $k < n \Longrightarrow$ *is-quantum-predicate* (*wlp* (*S*!*k*) *P*) **for** *k*
using *Measure*(*1*)[*OF set wck qpP qpP*] *lowner-le-refl*[*of P*] *dP* **by** *auto*
then have *dWkP*: $k < n \Longrightarrow$ *wlp* (*S*!*k*) $P \in$ *carrier-mat d d* **for** *k* **using** *is-quantum-predicate-def* **by** *auto*
then have *dMkP*: $k < n \Longrightarrow$ *adjoint* (*M k*) $*$ (*wlp* (*S*!*k*) *P*) $*$ (*M k*) $\in$ *carrier-mat d d* **for** *k* **using** *dMk* **by** *fastforce*
have $k < n \Longrightarrow$ *is-quantum-predicate* (*wlp* (*S*!*k*) *Q*) **for** *k*
using *Measure*(*1*)[*OF set wck qpQ qpQ*] *lowner-le-refl*[*of Q*] *dQ* **by** *auto*
then have *dWkQ*: $k < n \Longrightarrow$ *wlp* (*S*!*k*) $Q \in$ *carrier-mat d d* **for** *k* **using** *is-quantum-predicate-def* **by** *auto*
then have *dMkQ*: $k < n \Longrightarrow$ *adjoint* (*M k*) $*$ (*wlp* (*S*!*k*) *Q*) $*$ (*M k*) $\in$ *carrier-mat d d* **for** *k* **using** *dMk* **by** *fastforce*
have $k < n \Longrightarrow$ *wlp* (*S*!*k*) $P \leq_L$ *wlp* (*S*!*k*) *Q* **for** *k*
using *Measure*(*1*)[*OF set wck qpP qpQ le*] **by** *auto*
then have $k < n \Longrightarrow$ *adjoint* (*M k*) $*$ (*wlp* (*S*!*k*) *P*) $*$ (*M k*) $\leq_L$ *adjoint* (*M k*) $*$ (*wlp* (*S*!*k*) *Q*) $*$ (*M k*) **for** *k*
using *lowner-le-keep-under-measurement*[*OF dMk dWkP dWkQ*] **by** *auto*
then have *le'*: *wlp* (*Measure n M S*) $P \leq_L$ *wlp* (*Measure n M S*) *Q* **unfolding** *wlp-measure-expand*[*OF wc*]
using *lowner-le-matrix-sum dMkP dMkQ* **by** *auto*
have *qp'*: *is-quantum-predicate* (*wlp* (*Measure n M S*) *P*) **unfolding** *wlp-measure-expand*[*OF wc*]
using *qps-after-measure-is-qp*[*OF m*] *qpk* **by** *auto*
show *?case* **using** *le' qp'* **by** *auto*
next
case (*While M S*)
then have *m*: *measurement d 2 M* **and** *wcs*: *well-com S*
and *qpP*: *is-quantum-predicate P*
by *auto*
have *closeWS*: *is-quantum-predicate P* $\Longrightarrow$ *is-quantum-predicate* (*wlp S P*) **for** *P*
proof −
assume *asm*: *is-quantum-predicate P*
then have *dP*: $P \in$ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
then show *is-quantum-predicate* (*wlp S P*) **using** *While*(*1*)[*OF wcs asm asm*

lowner-le-refl] *dP* **by** *auto*
qed
have *monoWS*: *is-quantum-predicate P* $\Longrightarrow$ *is-quantum-predicate Q* $\Longrightarrow$ $P \leq_L Q$ $\Longrightarrow$ *wlp S P* $\leq_L$ *wlp S Q* **for** *P Q*
using *While*(*1*)[*OF wcs*] **by** *auto*
have *is-quantum-predicate* (*wlp* (*While M S*) *P*)
using *wlp-while-exists*[*of wlp S M P*] *closeWS monoWS m qpP* **by** *auto*
moreover have *wlp* (*While M S*) *P* $\leq_L$ *wlp* (*While M S*) *Q*
using *wlp-while-mono*[*of wlp S M P Q*] *closeWS monoWS m While* **by** *auto*
ultimately show *?case* **by** *auto*
qed

lemma *wlp-close*:
assumes *wc*: *well-com S* **and** *qp*: *is-quantum-predicate P*
shows *is-quantum-predicate* (*wlp S P*)
using *wlp-mono-and-close*[*OF wc qp qp*] *is-quantum-predicate-def*[*of P*] *qp lowner-le-refl* **by** *auto*

lemma *wlp-soundness*:
well-com S $\Longrightarrow$
($\bigwedge$*P*. (*is-quantum-predicate P* $\Longrightarrow$
($\forall \varrho \in$ *density-states*. *trace* (*wlp S P* $*$ ϱ) = *trace* (*P* $*$ (*denote S* ϱ)) + *trace* ϱ $-$ *trace* (*denote S* ϱ))))
proof (*induct S*)
case *SKIP*
then show *?case* **by** *auto*
next
case (*Utrans U*)
then have *dU*: *U* $\in$ *carrier-mat d d* **and** *u*: *unitary U* **and** *wc*: *well-com* (*Utrans U*)
and *qp*: *is-quantum-predicate P* **and** *dP*: *P* $\in$ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
have *qp'*: *is-quantum-predicate* (*wlp* (*Utrans U*) *P*) **using** *wlp-close*[*OF wc qp*] **by** *auto*
have *eq1*: *trace* (*adjoint U* $*$ *P* $*$ *U* $*$ ϱ) = *trace* (*P* $*$ (*U* $*$ ϱ $*$ *adjoint U*)) **if** *dr*: ϱ $\in$ *carrier-mat d d* **for** ϱ
using *dr dP* **apply** (*mat-assoc d*) **using** *wc* **by** *auto*
have *eq2*: *trace* (*U* $*$ ϱ $*$ *adjoint U*) = *trace* ϱ **if** *dr*: ϱ $\in$ *carrier-mat d d* **for** ϱ
using *unitary-operator-keep-trace*[*OF adjoint-dim*[*OF dU*] *dr unitary-adjoint*[*OF dU u*]] *adjoint-adjoint*[*of U*] **by** *auto*
show *?case* **using** *qp' eq1 eq2 density-states-def* **by** *auto*
next
case (*Seq S1 S2*)
then have *qp*: *is-quantum-predicate P* **and** *wc1*: *well-com S1* **and** *wc2*: *well-com S2* **by** *auto*
then have *qp2*: *is-quantum-predicate* (*wlp S2 P*) **using** *wlp-close* **by** *auto*
then have *qp1*: *is-quantum-predicate* (*wlp S1* (*wlp S2 P*)) **using** *wlp-close wc1* **by** *auto*
have *eq1*: *trace* (*wlp S2 P* $*$ ϱ) = *trace* (*P* $*$ *denote S2* ϱ) + *trace* ϱ $-$ *trace*

(*denote S2 ϱ*)
 if *ds*: *ϱ* ∈ *density-states* **for** *ϱ* **using** *Seq*(*2*) *wc2 qp ds* **by** *auto*
 have *eq2*: *trace* (*wlp S1* (*wlp S2 P*) ∗ *ϱ*) = *trace* ((*wlp S2 P*) ∗ *denote S1 ϱ*) + *trace ϱ* − *trace* (*denote S1 ϱ*)
 if *ds*: *ϱ* ∈ *density-states* **for** *ϱ* **using** *Seq*(*1*) *wc1 qp2 ds* **by** *auto*
 have *eq3*: *trace* (*wlp S1* (*wlp S2 P*) ∗ *ϱ*) = *trace* (*P* ∗ (*denote S2* (*denote S1 ϱ*))) + *trace ϱ* − *trace* (*denote S2* (*denote S1 ϱ*))
 if *ds*: *ϱ* ∈ *density-states* **for** *ϱ*
 proof −
 have *denote S1 ϱ* ∈ *density-states*
 using *ds denote-density-states wc1* **by** *auto*
 then have *trace* ((*wlp S2 P*) ∗ *denote S1 ϱ*) = *trace* (*P* ∗ *denote S2* (*denote S1 ϱ*)) + *trace* (*denote S1 ϱ*) − *trace* (*denote S2* (*denote S1 ϱ*))
 using *eq1* **by** *auto*
 then show *trace* (*wlp S1* (*wlp S2 P*) ∗ *ϱ*) = *trace* (*P* ∗ (*denote S2* (*denote S1 ϱ*))) + *trace ϱ* − *trace* (*denote S2* (*denote S1 ϱ*))
 using *eq2 ds* **by** *auto*
 qed
 then show *?case* **using** *qp1* **by** *auto*
next
 case (*Measure n M S*)
 then have *wc*: *well-com* (*Measure n M S*)
 and *wck*: *k* < *n* ⟹ *well-com* (*S*!*k*)
 and *qpP*: *is-quantum-predicate P*
 and *dP*: *P* ∈ *carrier-mat d d*
 and *qpWk*: *k* < *n* ⟹ *is-quantum-predicate* (*wlp* (*S*!*k*) *P*)
 and *dWk*: *k* < *n* ⟹ (*wlp* (*S*!*k*) *P*) ∈ *carrier-mat d d*
 and *c*: *k* < *n* ⟹ *ϱ* ∈ *density-states* ⟹ *trace* (*wlp* (*S*!*k*) *P* ∗ *ϱ*) = *trace* (*P* ∗ *denote* (*S*!*k*) *ϱ*) + *trace ϱ* − *trace* (*denote* (*S*!*k*) *ϱ*)
 and *m*: *measurement d n M*
 and *aMMkleq*: *k* < *n* ⟹ *adjoint* (*M k*) ∗ *M k* $\leq_L$ 1_m *d*
 and *dMk*: *k* < *n* ⟹ *M k* ∈ *carrier-mat d d*
 for *k ϱ* **using** *is-quantum-predicate-def measurement-def measure-well-com measurement-le-one-mat wlp-close* **by** *auto*
 {
 fix *ϱ* **assume** *ϱ*: *ϱ* ∈ *density-states*
 then have *dr*: *ϱ* ∈ *carrier-mat d d* **and** *pdor*: *partial-density-operator ϱ* **using** *density-states-def* **by** *auto*
 have *dsr*: *k* < *n* ⟹ (*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*) ∈ *density-states* **for** *k* **unfolding** *density-states-def*
 using *dMk pdo-close-under-measurement*[*OF dMk dr pdor aMMkleq*] *dr* **by** *fastforce*
 then have *leqk*: *k* < *n* ⟹ *trace* (*wlp* (*S*!*k*) *P* ∗ ((*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*))) =
 trace (*P* ∗ (*denote* (*S*!*k*) ((*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*)))) +
 (*trace* ((*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*)) − *trace* (*denote* (*S* ! *k*) ((*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*)))) **for** *k*
 using *c* **by** *auto*
 have *k* < *n* ⟹ *M k* ∗ *ϱ* ∗ *adjoint* (*M k*) ∈ *carrier-mat d d* **for** *k* **using** *dMk*

dr **by** *fastforce*
then have *dsMrk*: $k < n \Longrightarrow$ *matrix-sum d* (λ*k*. (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) *k* ∈ *carrier-mat d d* **for** *k*
using *matrix-sum-dim*[*of k* λ*k*. (*M k* ∗ ϱ ∗ *adjoint* (*M k*)) *d*] **by** *fastforce*
have $k < n \Longrightarrow$ *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k* ∈ *carrier-mat d d* **for** *k* **using** *dMk* **by** *fastforce*
then have *dsMW*: $k < n \Longrightarrow$ *matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k*) *k* ∈ *carrier-mat d d* **for** *k*
using *matrix-sum-dim*[*of k* λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k d*] **by** *fastforce*
have *dSMrk*: $k < n \Longrightarrow$ *denote* (*S* ! *k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*)) ∈ *carrier-mat d d* **for** *k*
using *denote-dim*[*OF wck*, *of k M k* ∗ ϱ ∗ *adjoint* (*M k*)] *dsr density-states-def* **by** *fastforce*
have *dsSMrk*: $k < n \Longrightarrow$ *matrix-sum d* (λ*k*. *denote* (*S*!*k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) *k* ∈ *carrier-mat d d* **for** *k*
using *matrix-sum-dim*[*of k* λ*k*. *denote* (*S* ! *k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*)) *d*, *OF dSMrk*] **by** *fastforce*
have $k \leq n \Longrightarrow$
trace (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k*) *k* ∗ ϱ)
= *trace* (*P* ∗ (*denote* (*Measure k M S*) ϱ)) + (*trace* (*matrix-sum d* (λ*k*. (*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) *k*) − *trace* (*denote* (*Measure k M S*) ϱ)) **for** *k*
unfolding *denote-measure-expand*[*OF* - *wc*]
proof (*induct k*)
case *0*
then show *?case* **unfolding** *matrix-sum.simps* **using** *dP dr* **by** *auto*
next
case (*Suc k*)
then have *k*: $k < n$ **by** *auto*
have *eq1*: *trace* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k*) (*Suc k*) ∗ ϱ)
= *trace* (*adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k* ∗ ϱ) + *trace* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k*) *k* ∗ ϱ)
unfolding *matrix-sum.simps*
using *dMk*[*OF k*] *dWk*[*OF k*] *dr dsMW*[*OF k*] **by** (*mat-assoc d*)

have *trace* (*adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k* ∗ ϱ) = *trace* ((*wlp* (*S*!*k*) *P*) ∗ (*M k* ∗ ϱ ∗ *adjoint* (*M k*)))
using *dMk*[*OF k*] *dWk*[*OF k*] *dr* **by** (*mat-assoc d*)
also have ... = *trace* (*P* ∗ (*denote* (*S*!*k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))) +
(*trace* ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) − *trace* (*denote* (*S* ! *k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))) **using** *leqk k* **by** *auto*
finally have *eq2*: *trace* (*adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k* ∗ ϱ) = *trace* (*P* ∗ (*denote* (*S*!*k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))) +
(*trace* ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) − *trace* (*denote* (*S* ! *k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))) **.**

have *eq3*: *trace* (*P* ∗ *matrix-sum d* (λ*k*. *denote* (*S*!*k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) (*Suc k*))

= *trace* (*P* ∗ (*denote* (*S*!*k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*)))) + *trace* (*P* ∗ *matrix-sum d* (*λk*. *denote* (*S*!*k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*))) *k*)
unfolding *matrix-sum.simps*
using *dP dSMrk*[*OF k*] *dsSMrk*[*OF k*] **by** (*mat-assoc d*)

have *eq4*: *trace* (*denote* (*S* ! *k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*)) + *matrix-sum d* (*λk*. *denote* (*S*!*k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*))) *k*)
= *trace* (*denote* (*S* ! *k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*))) + *trace* (*matrix-sum d* (*λk*. *denote* (*S*!*k*) (*M k* ∗ *ϱ* ∗ *adjoint* (*M k*))) *k*)
using *dSMrk*[*OF k*] *dsSMrk*[*OF k*] **by** (*mat-assoc d*)

show *?case*
apply (*subst eq1*) **apply** (*subst eq3*)
apply (*simp del*: *less-eq-complex-def*)
apply (*subst trace-add-linear*[*of M k* ∗ *ϱ* ∗ *adjoint* (*M k*) *d matrix-sum d* (*λk*. *M k* ∗ *ϱ* ∗ *adjoint* (*M k*)) *k*])
apply (*simp add*: *dMk adjoint-dim*[*OF dMk*] *dr mult-carrier-mat*[*of - d d - d*] *k*)
apply (*simp add*: *dsMrk k*)
apply (*subst eq4*)
apply (*insert eq2 Suc*(*1*) *k*, *fastforce*)
done
qed
then have *leq*: *trace* (*matrix-sum d* (*λk*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k*) *n* ∗ *ϱ*)
= *trace* (*P* ∗ *denote* (*Measure n M S*) *ϱ*) +
(*trace* (*matrix-sum d* (*λk*. (*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*)) *n*) − *trace* (*denote* (*Measure n M S*) *ϱ*))
by *auto*
have *trace* (*matrix-sum d* (*λk*. (*M k*) ∗ *ϱ* ∗ *adjoint* (*M k*)) *n*) = *trace ϱ* **using** *trace-measurement m dr* **by** *auto*

with *leq* **have** *trace* (*matrix-sum d* (*λk*. *adjoint* (*M k*) ∗ (*wlp* (*S*!*k*) *P*) ∗ *M k*) *n* ∗ *ϱ*)
= *trace* (*P* ∗ *denote* (*Measure n M S*) *ϱ*) + (*trace ϱ* − *trace* (*denote* (*Measure n M S*) *ϱ*))
unfolding *denote-measure-def* **by** *auto*
}
then show *?case* **unfolding** *wlp-measure-expand*[*OF wc*] **by** *auto*
next
case (*While M S*)
then have *qpP*: *is-quantum-predicate P* **and** *dP*: *P* ∈ *carrier-mat d d*
and *wcS*: *well-com S* **and** *m*: *measurement d 2 M* **and** *wc*: *well-com* (*While M S*)
using *is-quantum-predicate-def* **by** *auto*
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
have *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d* **using** *m measurement-def M0-def M1-def* **by** *auto*

have *leM1*: *adjoint M1* ∗ *M1* $\leq_L$ *1*$_m$ *d* **using** *measurement-le-one-mat m M1-def* **by** *auto*
define *W* **where** *W k* = *wlp-while-n M0 M1* (*wlp S*) *k P* **for** *k*
define *DS* **where** *DS* = *denote S*
define *D0* **where** *D0* = *denote-while-n M0 M1 DS*
define *D1* **where** *D1* = *denote-while-n-comp M0 M1 DS*
define *D* **where** *D* = *denote-while-n-iter M0 M1 DS*

have *eqk*: *ϱ* ∈ *density-states* ⟹ *trace* ((*W k*) ∗ *ϱ*) = (∑ *k=0..<k. trace* (*P* ∗ (*D0 k ϱ*))) + *trace ϱ* − (∑ *k=0..<k. trace* (*D0 k ϱ*)) **for** *k ϱ*
proof (*induct k arbitrary*: *ϱ*)
case *0*
then have *dsr*: *ϱ* ∈ *density-states* **by** *auto*
show *?case* **unfolding** *W-def wlp-while-n.simps* **using** *dsr density-states-def* **by** *auto*
next
case (*Suc k*)
then have *dsr*: *ϱ* ∈ *density-states* **and** *dr*: *ϱ* ∈ *carrier-mat d d* **and** *pdor*: *partial-density-operator ϱ* **using** *density-states-def* **by** *auto*
then have *dsM1r*: *M1* ∗ *ϱ* ∗ *adjoint M1* ∈ *density-states* **unfolding** *density-states-def* **using** *pdo-close-under-measurement*[*OF dM1 dr pdor leM1*] *dr dM1* **by** *auto*
then have *dsDSM1r*: (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)) ∈ *density-states* **unfolding** *density-states-def DS-def*
using *denote-dim*[*OF wcS*] *denote-partial-density-operator*[*OF wcS*] *dsM1r* **by** *auto*
have *qpWk*: *is-quantum-predicate* (*W k*)
using *wlp-while-n-close*[*OF - m qpP*, *folded M0-def M1-def*, *of wlp S*, *folded W-def*] *wlp-close*[*OF wcS*] **by** *auto*
then have *is-quantum-predicate* (*wlp S* (*W k*)) **using** *wlp-close*[*OF wcS*] **by** *auto*
then have *dWWk*: *wlp S* (*W k*) ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*

have *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*)) + (∑ *k=0..<k. trace* (*P* ∗ (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)))))
= *trace* (*P* ∗ (*D0 0 ϱ*)) + (∑ *k=0..<k. trace* (*P* ∗ (*D0* (*Suc k*) *ϱ*)))
unfolding *D0-def* **by** *auto*
also have … = *trace* (*P* ∗ (*D0 0 ϱ*)) + (∑ *k=1..<*(*Suc k*). *trace* (*P* ∗ (*D0 k ϱ*)))
using *sum.shift-bounds-Suc-ivl*[*symmetric*, *of* λ*k. trace* (*P* ∗ (*D0 k ϱ*))] **by** *auto*
also have … = (∑ *k=0..<*(*Suc k*). *trace* (*P* ∗ (*D0 k ϱ*))) **using** *sum.atLeast-Suc-lessThan*[*of 0 Suc k* λ*k. trace* (*P* ∗ (*D0 k ϱ*))] **by** *auto*
finally have *eq1*: *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*)) + (∑ *k=0..<k. trace* (*P* ∗ (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)))))
= (∑ *k=0..<*(*Suc k*). *trace* (*P* ∗ (*D0 k ϱ*))).

have *eq2*: *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) = *trace ϱ* − *trace* (*M0* ∗ *ϱ* ∗ *adjoint*

M0)
unfolding *M0-def M1-def* **using** *m trace-measurement2*[*OF m dr*] *dr* **by** (*simp add*: *algebra-simps*)

have *trace* (*M0* ∗ *ϱ* ∗ *adjoint M0*) + (∑ *k*=*0*..<*k*. *trace* (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))))
= *trace* (*D0 0 ϱ*) + (∑ *k*=*0*..<*k*. *trace* (*D0* (*Suc k*) *ϱ*)) **unfolding** *D0-def* **by** *auto*
also have ... = *trace* (*D0 0 ϱ*) + (∑ *k*=*1*..<(*Suc k*). *trace* (*D0 k ϱ*))
using *sum.shift-bounds-Suc-ivl*[*symmetric*, *of λk. trace* (*D0 k ϱ*)] **by** *auto*
also have ... = (∑ *k*=*0*..<(*Suc k*). *trace* (*D0 k ϱ*))
using *sum.atLeast-Suc-lessThan*[*of 0 Suc k λk. trace* (*D0 k ϱ*)] **by** *auto*
finally have *eq3*: *trace* (*M0* ∗ *ϱ* ∗ *adjoint M0*) + (∑ *k*=*0*..<*k*. *trace* (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)))) = (∑ *k*=*0*..<(*Suc k*). *trace* (*D0 k ϱ*)).

then have *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) − (∑ *k*=*0*..<*k*. *trace* (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))))
= *trace ϱ* − (*trace* (*M0* ∗ *ϱ* ∗ *adjoint M0*) + (∑ *k*=*0*..<*k*. *trace* (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)))))
by (*simp add*: *algebra-simps eq2*)
then have *eq4*: *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) − (∑ *k*=*0*..<*k*. *trace* (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)))) = *trace ϱ* − (∑ *k*=*0*..<(*Suc k*). *trace* (*D0 k ϱ*))
by (*simp add*: *eq3*)

have *trace* ((*W* (*Suc k*)) ∗ *ϱ*) = *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*)) + *trace* ((*wlp S* (*W k*)) ∗ (*M1* ∗ *ϱ* ∗ *adjoint M1*))
unfolding *W-def wlp-while-n.simps*
apply (*fold W-def*) **using** *dM0 dP dM1 dWWk dr* **by** (*mat-assoc d*)
also have ... = *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*)) + *trace* ((*W k*) ∗ (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))) + *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) − *trace* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))
using *While*(*1*)[*OF wcS*, *of W k*] *qpWk dsM1r DS-def* **by** *auto*
also have ... = *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*))
+ (∑ *k*=*0*..<*k*. *trace* (*P* ∗ (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))))) + *trace* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)) − (∑ *k*=*0*..<*k*. *trace* (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))))
+ *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) − *trace* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))
using *Suc*(*1*)[*OF dsDSM1r*] **by** *auto*
also have ... = *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*)) + (∑ *k*=*0*..<*k*. *trace* (*P* ∗ (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*)))))
+ *trace* (*M1* ∗ *ϱ* ∗ *adjoint M1*) − (∑ *k*=*0*..<*k*. *trace* (*D0 k* (*DS* (*M1* ∗ *ϱ* ∗ *adjoint M1*))))
by *auto*
also have ... = (∑ *k*=*0*..<(*Suc k*). *trace* (*P* ∗ (*D0 k ϱ*))) + *trace ϱ* − (∑ *k*=*0*..<(*Suc k*). *trace* (*D0 k ϱ*))
by (*simp add*: *eq1 eq4*)
finally show *?case*.
qed

{
fix ϱ **assume** *dsr*: ϱ ∈ *density-states*
then have *dr*: ϱ ∈ *carrier-mat d d* **and** *pdor*: *partial-density-operator* ϱ **using** *density-states-def* **by** *auto*
have *limDW*: *limit-mat* (λ*n*. *matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*)) (*denote* (*While M S*) ϱ) *d*
using *limit-mat-denote-while-n*[*OF wc dr pdor*] **unfolding** *D0-def M0-def M1-def DS-def* **by** *auto*
then have *limit-mat* (λ*n*. (*P* ∗ (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*)))) (*P* ∗ (*denote* (*While M S*) ϱ)) *d*
using *mat-mult-limit*[*OF dP*] **unfolding** *mat-mult-seq-def* **by** *auto*
then have *limtrPm*: (λ*n*. *trace* (*P* ∗ (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*)))) ⟶ *trace* (*P* ∗ (*denote* (*While M S*) ϱ))
using *mat-trace-limit* **by** *auto*

with *limDW* **have** *limtrDW*:(λ*n*. *trace* (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*))) ⟶ *trace* (*denote* (*While M S*) ϱ)
using *mat-trace-limit* **by** *auto*

have *limm*: (λ*n*. *trace* (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*))) ⟶ *trace* (*denote* (*While M S*) ϱ)
using *mat-trace-limit limDW* **by** *auto*

have *closeWS*: *is-quantum-predicate P* ⟹ *is-quantum-predicate* (*wlp S P*) **for** *P*
proof −
assume *asm*: *is-quantum-predicate P*
then have *dP*: *P* ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
then show *is-quantum-predicate* (*wlp S P*) **using** *wlp-mono-and-close*[*OF wcS asm asm*] *lowner-le-refl* **by** *auto*
qed
have *monoWS*: *is-quantum-predicate P* ⟹ *is-quantum-predicate Q* ⟹ $P \leq_L Q$ ⟹ *wlp S P* $\leq_L$ *wlp S Q* **for** *P Q*
using *wlp-mono-and-close*[*OF wcS*] **by** *auto*

have *is-quantum-predicate* (*wlp* (*While M S*) *P*)
using *wlp-while-exists*[*of wlp S M P*] *closeWS monoWS m qpP* **by** *auto*

have *limit-mat W* (*wlp-while M0 M1* (*wlp S*) *P*) *d* **unfolding** *W-def M0-def M1-def*
using *wlp-while-exists*[*of wlp S M P*] *closeWS monoWS m qpP* **by** *auto*
then have *limit-mat* (λ*k*. (*W k*) ∗ ϱ) ((*wlp-while M0 M1* (*wlp S*) *P*) ∗ ϱ) *d* **using** *mult-mat-limit dr* **by** *auto*
then have *lim1*: (λ*k*. *trace* ((*W k*) ∗ ϱ)) ⟶ *trace* ((*wlp-while M0 M1* (*wlp S*) *P*) ∗ ϱ)
using *mat-trace-limit* **by** *auto*

have *dD0kr*: *D0 k* ϱ ∈ *carrier-mat d d* **for** *k* **unfolding** *D0-def*
using *denote-while-n-dim*[*OF dr dM0 dM1 pdor*] *denote-positive-trace-dim*[*OF*

wcS, *folded DS-def*] **by** *auto*
then have (*P* ∗ (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*))) = *matrix-sum d* (λ*k*. *P* ∗ (*D0 k* ϱ)) *n* **for** *n*
using *matrix-sum-distrib-left*[*OF dP*] **by** *auto*
moreover have *trace* (*matrix-sum d* (λ*k*. *P* ∗ (*D0 k* ϱ)) *n*) = (∑ *k*=*0*..<*n*. *trace* (*P* ∗ (*D0 k* ϱ))) **for** *n*
using *trace-matrix-sum-linear dD0kr dP* **by** *auto*
ultimately have *eqPsD0kr*: *trace* (*P* ∗ (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*))) = (∑ *k*=*0*..<*n*. *trace* (*P* ∗ (*D0 k* ϱ))) **for** *n* **by** *auto*
with *limtrPm* **have** *lim2*: (λ*k*. (∑ *k*=*0*..<*k*. *trace* (*P* ∗ (*D0 k* ϱ)))) ⟶ *trace* (*P* ∗ (*denote* (*While M S*) ϱ)) **by** *auto*

have *trace* (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*)) = (∑ *k*=*0*..<*n*. *trace* (*D0 k* ϱ)) **for** *n*
using *trace-matrix-sum-linear dD0kr* **by** *auto*
with *limtrDW* **have** *lim3*: (λ*k*. (∑ *k*=*0*..<*k*. *trace* (*D0 k* ϱ))) ⟶ *trace* (*denote* (*While M S*) ϱ) **by** *auto*

have (λ*k*. (∑ *k*=*0*..<*k*. *trace* (*P* ∗ (*D0 k* ϱ))) + *trace* ϱ) ⟶ *trace* (*P* ∗ (*denote* (*While M S*) ϱ)) + *trace* ϱ
using *tendsto-add*[*of* λ*k*. (∑ *k*=*0*..<*k*. *trace* (*P* ∗ (*D0 k* ϱ)))] *lim2* **by** *auto*
then have (λ*k*. (∑ *k*=*0*..<*k*. *trace* (*P* ∗ (*D0 k* ϱ))) + *trace* ϱ − (∑ *k*=*0*..<*k*. *trace* (*D0 k* ϱ)))
⟶ *trace* (*P* ∗ (*denote* (*While M S*) ϱ)) + *trace* ϱ − *trace* (*denote* (*While M S*) ϱ)
using *tendsto-diff*[*of* - - - λ*k*. (∑ *k*=*0*..<*k*. *trace* (*D0 k* ϱ))] *lim3* **by** *auto*
then have *lim4*: (λ*k*. *trace* ((*W k*) ∗ ϱ)) ⟶ *trace* (*P* ∗ (*denote* (*While M S*) ϱ)) + *trace* ϱ − *trace* (*denote* (*While M S*) ϱ)
using *eqk*[*OF dsr*] **by** *auto*

then have *trace* ((*wlp-while M0 M1* (*wlp S*) *P*) ∗ ϱ) = *trace* (*P* ∗ (*denote* (*While M S*) ϱ)) + *trace* ϱ − *trace* (*denote* (*While M S*) ϱ)
using *eqk*[*OF dsr*] *tendsto-unique*[*OF* - *lim1 lim4*] **by** *auto*
}
then show *?case* **unfolding** *M0-def M1-def DS-def wlp.simps* **by** *auto*
qed

lemma *denote-while-split*:
assumes *wc*: *well-com* (*While M S*) **and** *dsr*: ϱ ∈ *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 −
have *m*: *measurement d 2 M* **using** *wc* **by** *auto*
have *wcs*: *well-com S* **using** *wc* **by** *auto*
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
have *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d* **using** *m measurement-def M0-def M1-def* **by** *auto*
have *M1leq*: *adjoint M1* ∗ *M1* $\leq_L$ 1_m *d* **using** *measurement-le-one-mat m M1-def*

by *auto*
define *DS* **where** *DS* = *denote S*
define *D0* **where** *D0* = *denote-while-n M0 M1 DS*
define *D1* **where** *D1* = *denote-while-n-comp M0 M1 DS*
define *D* **where** *D* = *denote-while-n-iter M0 M1 DS*
define *DW* **where** *DW* ϱ = *denote* (*While M S*) ϱ **for** ϱ

{
fix ϱ **assume** *dsr*: $\varrho \in$ *density-states*
then have *dr*: $\varrho \in$ *carrier-mat d d* **and** *pdor*: *partial-density-operator* ϱ **using** *density-states-def* **by** *auto*
have *pdoDkr*: $\bigwedge k.$ *partial-density-operator* (*D k* ϱ) **unfolding** *D-def*
using *pdo-denote-while-n-iter*[*OF dr pdor dM1 M1leq*]
denote-partial-density-operator[*OF wcs*] *denote-dim*[*OF wcs*, *folded DS-def*]
apply (*fold DS-def*) **by** *auto*
then have *pDkr*: $\bigwedge k.$ *positive* (*D k* ϱ) **unfolding** *partial-density-operator-def* **by** *auto*
have *dDkr*: $\bigwedge k.$ *D k* $\varrho \in$ *carrier-mat d d*
using *denote-while-n-iter-dim*[*OF dr pdor dM1 M1leq denote-dim-pdo*[*OF wcs*, *folded DS-def*], *of id M0*, *simplified*, *folded D-def*] **by** *auto*
then have *dD0kr*: $\bigwedge k.$ *D0 k* $\varrho \in$ *carrier-mat d d* **unfolding** *D0-def denote-while-n.simps* **apply** (*fold D-def*) **using** *dM0* **by** *auto*
}
note *dD0k* = *this*
have *matrix-sum d* ($\lambda k.$ *D0 k* ϱ) *k* $\in$ *carrier-mat d d* **if** *dsr*: $\varrho \in$ *density-states* **for** ϱ *k*
using *matrix-sum-dim*[*OF dD0k*, *of -* $\lambda k.$ ϱ *id*, *OF dsr*] *dsr* **by** *auto*
{
fix *k*
have *matrix-sum d* ($\lambda k.$ *D0 k* ϱ) (*Suc k*) = (*D0 0* ϱ) + *matrix-sum d* ($\lambda k.$ *D0* (*Suc k*) ϱ) *k*
using *matrix-sum-shift-Suc*[*of -* $\lambda k.$ *D0 k* ϱ] *dD0k*[*OF dsr*] **by** *fastforce*
also have ... = *M0* $*$ ϱ $*$ *adjoint M0* + *matrix-sum d* ($\lambda k.$ *D0 k* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*))) *k*
unfolding *D0-def* **by** *auto*
finally have *matrix-sum d* ($\lambda k.$ *D0 k* ϱ) (*Suc k*) = *M0* $*$ ϱ $*$ *adjoint M0* + *matrix-sum d* ($\lambda k.$ *D0 k* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*))) *k*.
}
note *eqk* = *this*

have *dr*: $\varrho \in$ *carrier-mat d d* **and** *pdor*: *partial-density-operator* ϱ **using** *density-states-def dsr* **by** *auto*
then have *M1* $*$ ϱ $*$ *adjoint M1* $\in$ *carrier-mat d d* **and** *partial-density-operator* (*M1* $*$ ϱ $*$ *adjoint M1*)
using *dM1 dr pdo-close-under-measurement*[*OF dM1 dr pdor M1leq*] **by** *auto*
then have *dSM1r*: (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*)) $\in$ *carrier-mat d d* **and** *pdoSM1r*: *partial-density-operator* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*))
unfolding *DS-def* **using** *denote-dim-pdo*[*OF wcs*] **by** *auto*

have *limit-mat* (*matrix-sum d* (λk. *D0 k* ϱ)) (*DW* ϱ) *d* **unfolding** *M0-def M1-def D0-def DS-def DW-def*
using *limit-mat-denote-while-n*[*OF wc dr pdor*] **by** *auto*
then have *liml*: *limit-mat* (λk. *matrix-sum d* (λk. *D0 k* ϱ) (*Suc k*)) (*DW* ϱ) *d*
using *limit-mat-ignore-initial-segment*[*of matrix-sum d* (λk. *D0 k* ϱ) *DW* ϱ *d 1*] **by** *auto*

have *dM0r*: *M0* $*$ ϱ $*$ *adjoint M0* $\in$ *carrier-mat d d* **using** *dM0 dr* **by** *fastforce*
have *limit-mat* (*matrix-sum d* (λk. *D0 k* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*)))) (*DW* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*))) *d*
using *limit-mat-denote-while-n*[*OF wc dSM1r pdoSM1r*] **unfolding** *M0-def M1-def D0-def DS-def DW-def* **by** *auto*
then have
limr: *limit-mat*
(*mat-add-seq* (*M0* $*$ ϱ $*$ *adjoint M0*) (*matrix-sum d* (λk. *D0 k* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*)))))
(*M0* $*$ ϱ $*$ *adjoint M0* + (*DW* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*))))
d
using *mat-add-limit*[*OF dM0r*] **by** *auto*
moreover have
(λk. *matrix-sum d* (λk. *D0 k* ϱ) (*Suc k*))
= (*mat-add-seq* (*M0* $*$ ϱ $*$ *adjoint M0*) (*matrix-sum d* (λk. *D0 k* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*)))))
using *eqk mat-add-seq-def* **by** *auto*
ultimately have
limit-mat
(λk. *matrix-sum d* (λk. *D0 k* ϱ) (*Suc k*))
(*M0* $*$ ϱ $*$ *adjoint M0* + (*DW* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*))))
d **by** *auto*
with *liml limit-mat-unique* **have**
DW ϱ = (*M0* $*$ ϱ $*$ *adjoint M0* + (*DW* (*DS* (*M1* $*$ ϱ $*$ *adjoint M1*)))) **by** *auto*
then show *?thesis* **unfolding** *DW-def M0-def M1-def DS-def* **by** *auto*
qed

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 −
have *m*: *measurement d 2 M* **using** *wc* **by** *auto*
have *wcs*: *well-com S* **using** *wc* **by** *auto*
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
have *dM0*: *M0* $\in$ *carrier-mat d d* **and** *dM1*: *M1* $\in$ *carrier-mat d d* **using** *m measurement-def M0-def M1-def* **by** *auto*
have *M1leq*: *adjoint M1* $*$ *M1* $\leq_L$ 1_m *d* **using** *measurement-le-one-mat m M1-def* **by** *auto*
define *DS* **where** *DS* = *denote S*
define *D0* **where** *D0* = *denote-while-n M0 M1 DS*

define *D1* **where** *D1* = *denote-while-n-comp M0 M1 DS*
define *D* **where** *D* = *denote-while-n-iter M0 M1 DS*
define *DW* **where** *DW* ϱ = *denote* (*While M S*) ϱ **for** ϱ

have *dP*: *P* ∈ *carrier-mat d d* **using** *qpP is-quantum-predicate-def* **by** *auto*
have *qpWP*: *is-quantum-predicate* (*wlp* (*While M S*) *P*) **using** *qpP wc wlp-close*[*OF wc qpP*] **by** *auto*
then have *is-quantum-predicate* (*wlp S* (*wlp* (*While M S*) *P*)) **using** *wc wlp-close*[*OF wcs*] **by** *auto*
then have *dWWP*: (*wlp S* (*wlp* (*While M S*) *P*)) ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
have *dWP*: (*wlp* (*While M S*) *P*) ∈ *carrier-mat d d* **using** *qpWP is-quantum-predicate-def* **by** *auto*
{
fix ϱ **assume** *dsr*: ϱ ∈ *density-states*
then have *dr*: ϱ ∈ *carrier-mat d d* **and** *pdor*: *partial-density-operator* ϱ **using** *density-states-def* **by** *auto*
have *dsM1r*: *M1* ∗ ϱ ∗ *adjoint M1* ∈ *density-states* **unfolding** *density-states-def*
using *pdo-close-under-measurement*[*OF dM1 dr pdor*] *M1leq dM1 dr* **by** *fastforce*
then have *dsDSM1r*: *DS* (*M1* ∗ ϱ ∗ *adjoint M1*) ∈ *density-states* **unfolding** *density-states-def DS-def*
using *denote-dim-pdo*[*OF wcs*] **by** *auto*
have *dM0r*: *M0* ∗ ϱ ∗ *adjoint M0* ∈ *carrier-mat d d* **using** *dM0 dr* **by** *fastforce*
have *dDWDSM1r*: *DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)) ∈ *carrier-mat d d*
unfolding *DW-def* **using** *denote-dim*[*OF wc*] *dsDSM1r density-states-def* **by** *auto*

have *eq2*: *trace* ((*wlp* (*While M S*) *P*) ∗ *DS* (*M1* ∗ ϱ ∗ *adjoint M1*))
= *trace* (*P* ∗ (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))) + *trace* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)) − *trace* (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))
unfolding *DW-def* **using** *wlp-soundness*[*OF wc qpP*] *dsDSM1r* **by** *auto*
have *eq3*: *trace* (*M1* ∗ ϱ ∗ *adjoint M1*) = *trace* ϱ − *trace* (*M0* ∗ ϱ ∗ *adjoint M0*)
unfolding *M0-def M1-def* **using** *m trace-measurement2*[*OF m dr*] *dr* **by** (*simp add*: *algebra-simps*)

have *trace* (*adjoint M1* ∗ (*wlp S* (*wlp* (*While M S*) *P*)) ∗ *M1* ∗ ϱ)
= *trace* ((*wlp S* (*wlp* (*While M S*) *P*)) ∗ (*M1* ∗ ϱ ∗ *adjoint M1*)) **using** *dWWP dM1 dr* **by** (*mat-assoc d*)
also have ... = *trace* ((*wlp* (*While M S*) *P*) ∗ *DS* (*M1* ∗ ϱ ∗ *adjoint M1*))
+ *trace* (*M1* ∗ ϱ ∗ *adjoint M1*) − *trace* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*))
unfolding *DS-def* **using** *wlp-soundness*[*OF wcs qpWP*] *dsM1r* **by** *auto*
also have ... = *trace* (*P* ∗ (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*))))
+ *trace* (*M1* ∗ ϱ ∗ *adjoint M1*) − *trace* (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))
using *eq2* **by** *auto*
also have ... = *trace* (*P* ∗ (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))) + *trace* ϱ − (*trace* (*M0* ∗ ϱ ∗ *adjoint M0*) + *trace* (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*))))

using *eq3* **by** *auto*
finally have *eq4*: *trace* (*adjoint M1* ∗ (*wlp S* (*wlp* (*While M S*) *P*)) ∗ *M1* ∗ ϱ)

= *trace* (*P* ∗ (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))) + *trace* ϱ − (*trace* (*M0* ∗ ϱ ∗ *adjoint M0*) + *trace* (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))).

have *trace* (*adjoint M0* ∗ *P* ∗ *M0* ∗ ϱ) + *trace* (*P* ∗ (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*))))
= *trace* (*P* ∗ ((*M0* ∗ ϱ ∗ *adjoint M0*) + (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))))
using *dP dr dM0 dDWDSM1r* **by** (*mat-assoc d*)
also have ... = *trace* (*P* ∗ (*DW* ϱ)) **unfolding** *DW-def M0-def M1-def DS-def* **using** *denote-while-split*[*OF wc dsr*] **by** *auto*
finally have *eq5*: *trace* (*adjoint M0* ∗ *P* ∗ *M0* ∗ ϱ) + *trace* (*P* ∗ (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))) = *trace* (*P* ∗ (*DW* ϱ)).

have *trace* (*M0* ∗ ϱ ∗ *adjoint M0*) + *trace* (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*)))
= *trace* (*M0* ∗ ϱ ∗ *adjoint M0* + (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*))))
using *dr dM0 dDWDSM1r* **by** (*mat-assoc d*)
also have ... = *trace* (*DW* ϱ)
unfolding *DW-def DS-def M0-def M1-def denote-while-split*[*OF wc dsr*] **by** *auto*
finally have *eq6*: *trace* (*M0* ∗ ϱ ∗ *adjoint M0*) + *trace* (*DW* (*DS* (*M1* ∗ ϱ ∗ *adjoint M1*))) = *trace* (*DW* ϱ).

from *eq5 eq4 eq6* **have**
eq7: *trace* (*adjoint M0* ∗ *P* ∗ *M0* ∗ ϱ) + *trace* (*adjoint M1* ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M1* ∗ ϱ)
= *trace* (*P* ∗ *DW* ϱ) + *trace* ϱ − *trace* (*DW* ϱ) **by** *auto*
have *eq8*: *trace* (*adjoint M0* ∗ *P* ∗ *M0* ∗ ϱ) + *trace* (*adjoint M1* ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M1* ∗ ϱ)
= *trace* ((*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M1*) ∗ ϱ)
using *dM0 dM1 dr dP dWWP* **by** (*mat-assoc d*)
from *eq7 eq8* **have**
eq9: *trace* ((*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M1*) ∗ ϱ) = *trace* (*P* ∗ *DW* ϱ) + *trace* ϱ − *trace* (*DW* ϱ) **by** *auto*
have *eq10*: *trace* ((*wlp* (*While M S*) *P*) ∗ ϱ) = *trace* (*P* ∗ *DW* ϱ) + *trace* ϱ − *trace* (*DW* ϱ)
unfolding *DW-def* **using** *wlp-soundness*[*OF wc qpP*] *dsr* **by** *auto*
with *eq9* **have** *trace* ((*wlp* (*While M S*) *P*) ∗ ϱ) = *trace* ((*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M1*) ∗ ϱ) **by** *auto*
}
then have (*wlp* (*While M S*) *P*) = (*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M1*)
using *trace-pdo-eq-imp-eq*[*OF dWP*, *of adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M1*]
dM0 dP dM1 dWWP density-states-def **by** *fastforce*
then show *?thesis* **using** *M0-def M1-def* **by** *auto*
qed

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*
unfolding *is-weakest-liberal-precondition-def*
proof (*auto*)
show *qpWP*: *is-quantum-predicate* (*wlp S P*) **using** *wlp-close assms* **by** *auto*
have *eq*: *trace* (*wlp S P* $*$ ϱ) = *trace* (*P* $*$ (*denote S* ϱ)) + *trace* ϱ − *trace* (*denote S* ϱ) **if** *dsr*: $\varrho \in$ *density-states* **for** ϱ
using *wlp-soundness assms dsr* **by** *auto*
then show $\models_p$ {*wlp S P*} *S* {*P*} **unfolding** *hoare-partial-correct-def* **by** *auto*
fix *Q* **assume** *qpQ*: *is-quantum-predicate Q* **and** *p*: $\models_p$ {*Q*} *S* {*P*}
{
fix ϱ **assume** *dsr*: $\varrho \in$ *density-states*
then have *trace* (*Q* $*$ ϱ) $\leq$ *trace* (*P* $*$ (*denote S* ϱ)) + *trace* ϱ − *trace* (*denote S* ϱ)
using *hoare-partial-correct-def p* **by** (*auto simp*: *less-eq-complex-def*)
then have *trace* (*Q* $*$ ϱ) $\leq$ *trace* (*wlp S P* $*$ ϱ) **using** *eq*[*symmetric*] *dsr* **by** *auto*
}
then show $Q \leq_L$ *wlp S P* **using** *lowner-le-trace density-states-def qpQ qpWP is-quantum-predicate-def* **by** *auto*
qed

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′* ⟹
$P \leq_L P'$ ⟹ $\vdash_p$ {*P′*} *S* {*Q′*} ⟹ $Q' \leq_L Q$ ⟹ $\vdash_p$ {*P*} *S* {*Q*}

theorem *hoare-partial-sound*:
$\vdash_p$ {*P*} *S* {*Q*} ⟹ *well-com S* ⟹ $\models_p$ {*P*} *S* {*Q*}
proof (*induction rule*: *hoare-partial.induct*)
case (*1 P*)
then show *?case*

unfolding *hoare-partial-correct-def* **by** *auto*
next
case (*2 P U*)
then have *dU*: $U \in$ *carrier-mat d d* **and** *unitary U* **and** *dP*: $P \in$ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
then have *uU*: *adjoint* $U * U = 1_m$ *d* **using** *unitary-def* **by** *auto*
show *?case*
unfolding *hoare-partial-correct-def denote.simps*(*2*)
proof
fix ϱ **assume** $\varrho \in$ *density-states*
then have *dr*: $\varrho \in$ *carrier-mat d d* **using** *density-states-def* **by** *auto*
have *e1*: *trace* ($U * \varrho *$ *adjoint U*) = *trace* ((*adjoint* $U * U$) $* \varrho$)
using *dr dU* **by** (*mat-assoc d*)
also have ... = *trace* ϱ
using *uU dr* **by** *auto*
finally have *e1*: *trace* ($U * \varrho *$ *adjoint U*) = *trace* ϱ **.**
have *e2*: *trace* ($P *$ ($U * \varrho *$ *adjoint U*)) = *trace* (*adjoint* $U * P * U * \varrho$)
using *dU dP dr* **by** (*mat-assoc d*)
with *e1* **have** *trace* ($P *$ ($U * \varrho *$ *adjoint U*)) + (*trace* ϱ − *trace* ($U * \varrho *$ *adjoint U*)) = *trace* (*adjoint* $U * P * U * \varrho$)
using *e1* **by** *auto*
then show *trace* (*adjoint* $U * P * U * \varrho$) ≤ *trace* ($P *$ ($U * \varrho *$ *adjoint U*)) + (*trace* ϱ − *trace* ($U * \varrho *$ *adjoint U*)) **by** *auto*
qed
next
case (*3 P Q R S1 S2*)
then have *wc1*: $\models_p$ {*P*} *S1* {*Q*} **and** *wc2*: $\models_p$ {*Q*} *S2* {*R*} **by** *auto*
show *?case*
unfolding *hoare-partial-correct-def denote.simps*(*3*)
proof *clarify*
fix ϱ **assume** ϱ: $\varrho \in$ *density-states*
have *1*: *trace* ($P * \varrho$) ≤ *trace* ($Q *$ *denote S1* ϱ) + (*trace* ϱ − *trace* (*denote S1* ϱ))
using *wc1 hoare-partial-correct-def* ϱ **by** *auto*
have ϱ': *denote S1* $\varrho \in$ *density-states*
using *3*(*8*) *denote-density-states* ϱ **by** *auto*
have *2*: *trace* ($Q *$ *denote S1* ϱ) ≤ *trace* ($R *$ *denote S2* (*denote S1* ϱ)) + (*trace* (*denote S1* ϱ) − *trace* (*denote S2* (*denote S1* ϱ)))
using *wc2 hoare-partial-correct-def* ϱ' **by** *auto*
show *trace* ($P * \varrho$) ≤ *trace* ($R *$ *denote S2* (*denote S1* ϱ)) + (*trace* ϱ − *trace* (*denote S2* (*denote S1* ϱ)))
using *1 2* **by** (*auto simp*: *less-eq-complex-def*)
qed
next
case (*4 n P Q S M*)
then have *wc*: $k < n \Longrightarrow$ *well-com* (*S*!*k*)
and *c*: $k < n \Longrightarrow \models_p$ {*P k*} (*S*!*k*) {*Q*} **and** *m*: *measurement d n M*
and *dMk*: $k < n \Longrightarrow M\ k \in$ *carrier-mat d d*
and *aMMkleq*: $k < n \Longrightarrow$ *adjoint* (*M k*) $* M\ k \leq_L 1_m$ *d*

and *dPk*: $k < n \Longrightarrow P\ k \in carrier\text{-}mat\ d\ d$
and *dQ*: $Q \in carrier\text{-}mat\ d\ d$
for *k* **using** *is-quantum-predicate-def measurement-def measure-well-com measurement-le-one-mat* **by** *auto*

{
fix ϱ **assume** ϱ: $\varrho \in density\text{-}states$
then have *dr*: $\varrho \in carrier\text{-}mat\ d\ d$ **and** *pdor*: *partial-density-operator* ϱ **using** *density-states-def* **by** *auto*
have *dsr*: $k < n \Longrightarrow (M\ k) * \varrho * adjoint\ (M\ k) \in density\text{-}states$ **for** *k* **unfolding** *density-states-def*
using *dMk pdo-close-under-measurement*[*OF dMk dr pdor aMMkleq*] *dr* **by** *fastforce*
then have *leqk*: $k < n \Longrightarrow trace\ ((P\ k) * ((M\ k) * \varrho * adjoint\ (M\ k))) \leq$
$trace\ (Q * (denote\ (S!k)\ ((M\ k) * \varrho * adjoint\ (M\ k)))) +$
$(trace\ ((M\ k) * \varrho * adjoint\ (M\ k)) - trace\ (denote\ (S\ !\ k)\ ((M\ k) * \varrho * adjoint\ (M\ k))))$ **for** *k*
using *c* **unfolding** *hoare-partial-correct-def* **by** *auto*
have $k < n \Longrightarrow M\ k * \varrho * adjoint\ (M\ k) \in carrier\text{-}mat\ d\ d$ **for** *k* **using** *dMk dr* **by** *fastforce*
then have *dsMrk*: $k < n \Longrightarrow matrix\text{-}sum\ d\ (\lambda k.\ (M\ k * \varrho * adjoint\ (M\ k)))\ k \in carrier\text{-}mat\ d\ d$ **for** *k*
using *matrix-sum-dim*[*of k* $\lambda k.\ (M\ k * \varrho * adjoint\ (M\ k))$ *d*] **by** *fastforce*
have $k < n \Longrightarrow adjoint\ (M\ k) * P\ k * M\ k \in carrier\text{-}mat\ d\ d$ **for** *k* **using** *dMk dPk* **by** *fastforce*
then have *dsMP*: $k < n \Longrightarrow matrix\text{-}sum\ d\ (\lambda k.\ adjoint\ (M\ k) * P\ k * M\ k)\ k \in carrier\text{-}mat\ d\ d$ **for** *k*
using *matrix-sum-dim*[*of k* $\lambda k.\ adjoint\ (M\ k) * P\ k * M\ k$ *d*] **by** *fastforce*
have *dSMrk*: $k < n \Longrightarrow denote\ (S\ !\ k)\ (M\ k * \varrho * adjoint\ (M\ k)) \in carrier\text{-}mat\ d\ d$ **for** *k*
using *denote-dim*[*OF wc*, *of k* $M\ k * \varrho * adjoint\ (M\ k)$] *dsr density-states-def* **by** *fastforce*
have *dsSMrk*: $k < n \Longrightarrow matrix\text{-}sum\ d\ (\lambda k.\ denote\ (S!k)\ (M\ k * \varrho * adjoint\ (M\ k)))\ k \in carrier\text{-}mat\ d\ d$ **for** *k*
using *matrix-sum-dim*[*of k* $\lambda k.\ denote\ (S\ !\ k)\ (M\ k * \varrho * adjoint\ (M\ k))$ *d*, *OF dSMrk*] **by** *fastforce*
have $k \leq n \Longrightarrow$
$trace\ (matrix\text{-}sum\ d\ (\lambda k.\ adjoint\ (M\ k) * P\ k * M\ k)\ k * \varrho)$
$\leq trace\ (Q * (denote\ (Measure\ k\ M\ S)\ \varrho)) + (trace\ (matrix\text{-}sum\ d\ (\lambda k.\ (M\ k) * \varrho * adjoint\ (M\ k))\ k) - trace\ (denote\ (Measure\ k\ M\ S)\ \varrho))$ **for** *k*
unfolding *denote-measure-expand*[*OF - 4(5)*]
proof (*induct k*)
case *0*
then show *?case* **using** *dQ dr pdor partial-density-operator-def positive-trace* **by** *auto*
next
case (*Suc k*)
then have *k*: $k < n$ **by** *auto*
have *eq1*: $trace\ (matrix\text{-}sum\ d\ (\lambda k.\ adjoint\ (M\ k) * P\ k * M\ k)\ (Suc\ k) *$

ϱ)
= *trace* (*adjoint* (*M k*) ∗ *P k* ∗ *M k* ∗ ϱ) + *trace* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *k* ∗ ϱ)
unfolding *matrix-sum.simps*
using *dMk*[*OF k*] *dPk*[*OF k*] *dr dsMP*[*OF k*] **by** (*mat-assoc d*)

have *trace* (*adjoint* (*M k*) ∗ *P k* ∗ *M k* ∗ ϱ) = *trace* (*P k* ∗ (*M k* ∗ ϱ ∗ *adjoint* (*M k*)))
using *dMk*[*OF k*] *dPk*[*OF k*] *dr* **by** (*mat-assoc d*)
also have ... ≤ *trace* (*Q* ∗ (*denote* (*S*!*k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))) +
(*trace* ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) − *trace* (*denote* (*S* ! *k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))) **using** *leqk k* **by** *auto*
finally have *eq2*: *trace* (*adjoint* (*M k*) ∗ *P k* ∗ *M k* ∗ ϱ) ≤ *trace* (*Q* ∗ (*denote* (*S*!*k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))) +
(*trace* ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) − *trace* (*denote* (*S* ! *k*) ((*M k*) ∗ ϱ ∗ *adjoint* (*M k*)))).

have *eq3*: *trace* (*Q* ∗ *matrix-sum d* (λ*k*. *denote* (*S*!*k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) (*Suc k*))
= *trace* (*Q* ∗ (*denote* (*S*!*k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*)))) + *trace* (*Q* ∗ *matrix-sum d* (λ*k*. *denote* (*S*!*k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) *k*)
unfolding *matrix-sum.simps*
using *dQ dSMrk*[*OF k*] *dsSMrk*[*OF k*] **by** (*mat-assoc d*)

have *eq4*: *trace* (*denote* (*S* ! *k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*)) + *matrix-sum d* (λ*k*. *denote* (*S*!*k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) *k*)
= *trace* (*denote* (*S* ! *k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) + *trace* (*matrix-sum d* (λ*k*. *denote* (*S*!*k*) (*M k* ∗ ϱ ∗ *adjoint* (*M k*))) *k*)
using *dSMrk*[*OF k*] *dsSMrk*[*OF k*] **by** (*mat-assoc d*)

show *?case*
apply (*subst eq1*) **apply** (*subst eq3*)
apply (*simp del*: *less-eq-complex-def*)
apply (*subst trace-add-linear*[*of M k* ∗ ϱ ∗ *adjoint* (*M k*) *d matrix-sum d* (λ*k*. *M k* ∗ ϱ ∗ *adjoint* (*M k*)) *k*])
apply (*simp add*: *dMk adjoint-dim*[*OF dMk*] *dr mult-carrier-mat*[*of - d d - d*] *k*)
apply (*simp add*: *dsMrk k*)
apply (*subst eq4*)
apply (*insert eq2 Suc*(*1*) *k*, *fastforce simp*: *less-eq-complex-def*)
done
qed
then have *leq*: *trace* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n* ∗ ϱ)
≤ *trace* (*Q* ∗ *denote* (*Measure n M S*) ϱ) +
(*trace* (*matrix-sum d* (λ*k*. (*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) *n*) − *trace* (*denote* (*Measure n M S*) ϱ))
by *auto*
have *trace* (*matrix-sum d* (λ*k*. (*M k*) ∗ ϱ ∗ *adjoint* (*M k*)) *n*) = *trace* ϱ **using** *trace-measurement m dr* **by** *auto*

with *leq* **have** *trace* (*matrix-sum d* (λ*k*. *adjoint* (*M k*) ∗ *P k* ∗ *M k*) *n* ∗ ϱ)
≤ *trace* (*Q* ∗ *denote* (*Measure n M S*) ϱ) + (*trace* ϱ − *trace* (*denote* (*Measure n M S*) ϱ))
unfolding *denote-measure-def* **by** *auto*
}
then show *?case* **unfolding** *hoare-partial-correct-def* **by** *auto*
next
case (*5 P Q S M*)
define *M0* **where** *M0* = *M 0*
define *M1* **where** *M1* = *M 1*
from *5* **have** *wcs*: *well-com S* **and** *c*: $\models_p$ {*Q*} *S* {*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *Q* ∗ *M1*}
and *m*: *measurement d 2 M*
and *dM0*: *M0* ∈ *carrier-mat d d* **and** *dM1*: *M1* ∈ *carrier-mat d d*
and *dP*: *P* ∈ *carrier-mat d d* **and** *dQ*: *Q* ∈ *carrier-mat d d*
and *qpQ*: *is-quantum-predicate Q*
and *wc*: *well-com* (*While M S*)
using *measurement-def is-quantum-predicate-def M0-def M1-def* **by** *auto*
then have *M0leq*: *adjoint M0* ∗ *M0* $\leq_L$ 1_m *d* **and** *M1leq*: *adjoint M1* ∗ *M1* $\leq_L$ 1_m *d* **using** *measurement-le-one-mat*[*OF m*] *M0-def M1-def* **by** *auto*
define *DS* **where** *DS* = *denote S*

have ∀ ϱ ∈ *density-states*. *trace* (*Q* ∗ ϱ) ≤ *trace* ((*adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *Q* ∗ *M1*) ∗ *DS* ϱ) + *trace* ϱ − *trace* (*DS* ϱ)
using *hoare-partial-correct-def*[*of Q S adjoint M0* ∗ *P* ∗ *M0* + *adjoint M1* ∗ *Q* ∗ *M1*] *c DS-def*
by (*auto simp*: *less-eq-complex-def*)
define *D0* **where** *D0* = *denote-while-n M0 M1 DS*
define *D1* **where** *D1* = *denote-while-n-comp M0 M1 DS*
define *D* **where** *D* = *denote-while-n-iter M0 M1 DS*
{
fix ϱ **assume** *dsr*: ϱ ∈ *density-states*
then have *dr*: ϱ ∈ *carrier-mat d d* **and** *pr*: *positive* ϱ **and** *pdor*: *partial-density-operator* ϱ
using *density-states-def partial-density-operator-def* **by** *auto*
have *pdoDkr*: ⋀*k*. *partial-density-operator* (*D k* ϱ) **unfolding** *D-def*
using *pdo-denote-while-n-iter*[*OF dr pdor dM1 M1leq*]
denote-partial-density-operator[*OF wcs*] *denote-dim*[*OF wcs*, *folded DS-def*]
apply (*fold DS-def*) **by** *auto*
then have *pDkr*: ⋀*k*. *positive* (*D k* ϱ) **unfolding** *partial-density-operator-def* **by** *auto*
have *dDkr*: ⋀*k*. *D k* ϱ ∈ *carrier-mat d d*
using *denote-while-n-iter-dim*[*OF dr pdor dM1 M1leq denote-dim-pdo*[*OF wcs*, *folded DS-def*], *of id M0*, *simplified*, *folded D-def*] **by** *auto*
then have *dD0kr*: ⋀*k*. *D0 k* ϱ ∈ *carrier-mat d d* **unfolding** *D0-def denote-while-n.simps* **apply** (*fold D-def*) **using** *dM0* **by** *auto*
then have *dPD0kr*: ⋀*k*. *P* ∗ (*D0 k* ϱ) ∈ *carrier-mat d d* **using** *dP* **by** *auto*
have ⋀*k*. *positive* (*D0 k* ϱ) **unfolding** *D0-def denote-while-n.simps*

by (*fold D-def*, *rule positive-close-under-left-right-mult-adjoint*[*OF dM0 dDkr pDkr*])
then have *trge0*: $\bigwedge k$. *trace* (*D0 k* ϱ) $\geq$ *0* **using** *positive-trace dD0kr* **by** *blast*
have *DSr*: $\varrho \in$ *density-states* $\Longrightarrow$ *DS* $\varrho \in$ *density-states* **for** ϱ **unfolding** *DS-def density-states-def*
using *denote-partial-density-operator denote-dim wcs* **by** *auto*
have *dsD1nr*: *D1 n* $\varrho \in$ *density-states* **for** *n* **unfolding** *D1-def denote-while-n-comp.simps*

apply (*fold D-def*) **unfolding** *density-states-def*
apply (*auto*)
apply (*insert dDkr dM1 adjoint-dim*[*OF dM1*], *auto*)
apply (*rule pdo-close-under-measurement*[*OF dM1 spec*[*OF allI*[*OF dDkr*], *of* λx. *n*] *spec*[*OF allI*[*OF pdoDkr*], *of* λx. *n*] *M1leq*])
done

have *leQn*: *trace* (*Q* $*$ *D1 n* ϱ)
$\leq$ *trace* (*P* $*$ *D0* (*Suc n*) ϱ) + *trace* (*Q* $*$ *D1* (*Suc n*) ϱ)
+ *trace* (*D1 n* ϱ) $-$ *trace* (*D* (*Suc n*) ϱ) **for** *n*
proof $-$
have ($\forall$ $\varrho \in$*density-states*. *trace* (*Q* $*$ ϱ) $\leq$ *trace* ((*adjoint M0* $*$ *P* $*$ *M0* + *adjoint M1* $*$ *Q* $*$ *M1*) $*$ *denote S* ϱ) + (*trace* ϱ $-$ *trace* (*denote S* ϱ)))
using *c hoare-partial-correct-def* **by** *auto*
then have *leQn'*: *trace* (*Q* $*$ (*D1 n* ϱ))
$\leq$ *trace* ((*adjoint M0* $*$ *P* $*$ *M0* + *adjoint M1* $*$ *Q* $*$ *M1*) $*$ (*DS* (*D1 n* ϱ)))
+ (*trace* (*D1 n* ϱ) $-$ *trace* (*DS* (*D1 n* ϱ)))
using *dsD1nr*[*of n*] *DS-def* **by** *auto*
have (*DS* (*D1 n* ϱ)) $\in$ *carrier-mat d d* **unfolding** *DS-def* **using** *denote-dim*[*OF wcs*] *dsD1nr density-states-def* **by** *auto*
then have *trace* ((*adjoint M0* $*$ *P* $*$ *M0* + *adjoint M1* $*$ *Q* $*$ *M1*) $*$ (*DS* (*D1 n* ϱ)))
= *trace* (*P* $*$ (*M0* $*$ (*DS* (*D1 n* ϱ)) $*$ *adjoint M0*))
+ *trace* (*Q* $*$ (*M1* $*$ (*DS* (*D1 n* ϱ)) $*$ *adjoint M1*)) **using** *dP dQ dM0 dM1* **by** (*mat-assoc d*)
moreover have *trace* (*P* $*$ (*M0* $*$ (*DS* (*D1 n* ϱ)) $*$ *adjoint M0*)) = *trace* (*P* $*$ *D0* (*Suc n*) ϱ)
unfolding *D0-def denote-while-n.simps*
apply (*subst denote-while-n-iter-assoc*)
by (*fold denote-while-n-comp.simps D1-def*, *auto*)
moreover have *trace* (*Q* $*$ (*M1* $*$ (*DS* (*D1 n* ϱ)) $*$ *adjoint M1*)) = *trace* (*Q* $*$ *D1* (*Suc n*) ϱ)
apply (*subst* (*2*) *D1-def*) **unfolding** *denote-while-n-comp.simps*
apply (*subst denote-while-n-iter-assoc*)
by (*fold denote-while-n-comp.simps D1-def*, *auto*)
ultimately have *trace* ((*adjoint M0* $*$ *P* $*$ *M0* + *adjoint M1* $*$ *Q* $*$ *M1*) $*$ (*DS* (*D1 n* ϱ)))
= *trace* (*P* $*$ *D0* (*Suc n*) ϱ) + *trace* (*Q* $*$ *D1* (*Suc n*) ϱ) **by** *auto*
moreover have *trace* (*DS* (*D1 n* ϱ)) = *trace* (*D* (*Suc n*) ϱ)
unfolding *D-def*

apply (*subst denote-while-n-iter-assoc*)
by (*fold denote-while-n-comp.simps D1-def*, *auto*)
ultimately show *?thesis* **using** *leQn′* **by** (*auto simp*: *less-eq-complex-def*)
qed

have *12*: *trace* ($P * (M0 * \varrho * adjoint\ M0)$) + *trace* ($Q * (M1 * \varrho * adjoint\ M1)$)
$\leq (\sum k=0..<(n+2).\ trace\ (P * (D0\ k\ \varrho))) + trace\ (Q * (D1\ (n+1)\ \varrho))$
$+ (\sum k=0..<(n+1).\ trace\ (D1\ k\ \varrho) - trace\ (D\ (k+1)\ \varrho))$ **for** *n*
proof (*induct n*)
case *0*
show *?case* **apply** (*simp del*: *less-eq-complex-def*)
unfolding *D0-def D1-def D-def denote-while-n-comp.simps denote-while-n.simps denote-while-n-iter.simps*
using *leQn*[*of 0*] **unfolding** *D1-def D0-def D-def denote-while-n.simps denote-while-n-comp.simps denote-while-n-iter.simps*
by (*auto simp*: *less-eq-complex-def*)
next
case (*Suc n*)
have *trace* ($Q * D1\ (n + 1)\ \varrho$)
$\leq trace\ (P * D0\ (Suc\ (Suc\ n))\ \varrho) + trace\ (Q * D1\ (Suc\ (Suc\ n))\ \varrho)$
$+ trace\ (D1\ (Suc\ n)\ \varrho) - trace\ (D\ (Suc\ (Suc\ n))\ \varrho)$ **using** *leQn*[*of n + 1*] **by** *auto*
with *Suc* **show** *?case* **apply** (*simp del*: *less-eq-complex-def*) **by** (*auto simp*: *less-eq-complex-def*)
qed

have *tr-measurement*: $\varrho \in carrier\text{-}mat\ d\ d$
$\Longrightarrow trace\ (M0 * \varrho * adjoint\ M0) + trace\ (M1 * \varrho * adjoint\ M1) = trace\ \varrho$ **for** ϱ
using *trace-measurement2*[*OF m*, *folded M0-def M1-def*] **by** *auto*

have *14*: $(\sum k=0..<(n+1).\ trace\ (D1\ k\ \varrho) - trace\ (D\ (k+1)\ \varrho)) = trace\ \varrho - trace\ (D\ (n+1)\ \varrho) - (\sum k=0..<(n+1).\ trace\ (D0\ k\ \varrho))$ **for** *n*
proof (*induct n*)
case *0*
show *?case* **apply** (*simp*) **unfolding** *D1-def D0-def denote-while-n-comp.simps denote-while-n.simps denote-while-n-iter.simps*
using *tr-measurement*[*OF dr*] **by** (*auto simp add*: *algebra-simps*)
next
case (*Suc n*)
have *trace* ($D0\ (Suc\ n)\ \varrho$) + *trace* ($D1\ (Suc\ n)\ \varrho$) = *trace* ($D\ (Suc\ n)\ \varrho$)
unfolding *D0-def D1-def denote-while-n.simps denote-while-n-comp.simps* **apply** (*fold D-def*)
using *tr-measurement dDkr* **by** *auto*
then have *trace* ($D1\ (Suc\ n)\ \varrho$) = *trace* ($D\ (Suc\ n)\ \varrho$) − *trace* ($D0\ (Suc\ n)\ \varrho$)
by (*auto simp add*: *algebra-simps*)
then show *?case* **using** *Suc* **by** *simp*

qed

have *15*: *trace* (*Q* ∗ (*D1 n* ϱ)) ≤ *trace* (*D n* ϱ) − *trace* (*D0 n* ϱ) **for** *n*
proof −
have *QleId*: *Q* ≤$_L$ *1*$_m$ *d* **using** *is-quantum-predicate-def qpQ* **by** *auto*
then have *trace* (*Q* ∗ (*D1 n* ϱ)) ≤ *trace* (*1*$_m$ *d* ∗ (*D1 n* ϱ)) **using**
dsD1nr[*of n*] **unfolding** *density-states-def lowner-le-trace*[*OF dQ one-carrier-mat*]
by *auto*
also have . . . = *trace* (*D1 n* ϱ) **using** *dsD1nr*[*of n*] **unfolding** *density-states-def*
by *auto*
also have . . . = *trace* (*M1* ∗ (*D n* ϱ) ∗ *adjoint M1*) **unfolding** *D1-def*
denote-while-n-comp.simps D-def **by** *auto*
also have . . . = *trace* (*D n* ϱ) − *trace* (*M0* ∗ (*D n* ϱ) ∗ *adjoint M0*)
using *tr-measurement*[*OF dDkr*[*of n*]] **by** (*simp add*: *algebra-simps*)
also have . . . = *trace* (*D n* ϱ) − *trace* (*D0 n* ϱ) **unfolding** *D0-def denote-while-n.simps* **by** (*fold D-def*, *auto*)
finally show *?thesis*.
qed

have *tmp*: ⋀*a b c*. *0* ≤ *a* ⟹ *b* ≤ *c* − *a* ⟹ *b* ≤ (*c*::*complex*)
by (*simp add*: *less-eq-complex-def*)
then have *151*: ⋀*n*. *trace* (*Q* ∗ (*D1 n* ϱ)) ≤ *trace* (*D n* ϱ)
by (*auto simp add*: *tmp*[*OF trge0 15*] *simp del*: *less-eq-complex-def*)

have *main-leq*: ⋀*n*. *trace* (*P* ∗ (*M0* ∗ ϱ ∗ *adjoint M0*)) + *trace* (*Q* ∗ (*M1* ∗ ϱ ∗ *adjoint M1*))
≤ *trace* (*P* ∗ (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*+*2*))) + *trace* ϱ − *trace* (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*+*2*))
proof −
fix *n*
have (∑*k*=*0*..<(*n*+*2*). *trace* (*P* ∗ (*D0 k* ϱ))) + *trace* (*Q* ∗ (*D1* (*n*+*1*) ϱ))
+ (∑*k*=*0*..<(*n*+*1*). *trace* (*D1 k* ϱ) − *trace* (*D* (*k*+*1*) ϱ))
≤ (∑*k*=*0*..<(*n*+*2*). *trace* (*P* ∗ (*D0 k* ϱ))) + *trace* (*Q* ∗ (*D1* (*n*+*1*) ϱ))
+ *trace* ϱ − *trace* (*D* (*n*+*1*) ϱ) − (∑*k*=*0*..<(*n*+*1*). *trace* (*D0 k* ϱ))
by (*subst 14*, *auto*)
also have
. . . ≤ (∑*k*=*0*..<(*n*+*2*). *trace* (*P* ∗ (*D0 k* ϱ))) + *trace* (*D* (*n*+*1*) ϱ) − *trace* (*D0* (*n*+*1*) ϱ)
+ *trace* ϱ − *trace* (*D* (*n*+*1*) ϱ) − (∑*k*=*0*..<(*n*+*1*). *trace* (*D0 k* ϱ))
using *15*[*of n*+*1*] **by** (*auto simp*: *less-eq-complex-def*)
also have . . . = (∑*k*=*0*..<(*n*+*2*). *trace* (*P* ∗ (*D0 k* ϱ))) + *trace* ϱ − (∑*k*=*0*..<(*n*+*2*). *trace* (*D0 k* ϱ)) **by** *auto*
also have . . . = *trace* (*matrix-sum d* (λ*k*. (*P* ∗ (*D0 k* ϱ))) (*n*+*2*)) + *trace* ϱ − (∑*k*=*0*..<(*n*+*2*). *trace* (*D0 k* ϱ))
using *trace-matrix-sum-linear*[*of n*+*2* λ*k*. (*P* ∗ (*D0 k* ϱ)) *d*, *symmetric*] *dPD0kr* **by** *auto*
also have . . . = *trace* (*matrix-sum d* (λ*k*. (*P* ∗ (*D0 k* ϱ))) (*n*+*2*)) + *trace* ϱ − *trace* (*matrix-sum d* (λ*k*. *D0 k* ϱ) (*n*+*2*))
using *trace-matrix-sum-linear*[*of n*+*2* λ*k*. *D0 k* ϱ *d*, *symmetric*] *dD0kr* **by**

auto
also have ... = *trace* (*P* ∗ (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*))) + *trace ϱ* − *trace* (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*))
using *matrix-sum-distrib-left*[*OF dP dD0kr*, *of id n+2*] **by** *auto*
finally have
(∑ *k=0..<*(*n+2*). *trace* (*P* ∗ (*D0 k ϱ*))) + *trace* (*Q* ∗ (*D1* (*n+1*) *ϱ*))
+ (∑ *k=0..<*(*n+1*). *trace* (*D1 k ϱ*) − *trace* (*D* (*k+1*) *ϱ*))
≤ *trace* (*P* ∗ (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*))) + *trace ϱ* − *trace* (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*)) **.**
then show *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*)) + *trace* (*Q* ∗ (*M1* ∗ *ϱ* ∗ *adjoint M1*))
≤ *trace* (*P* ∗ (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*))) + *trace ϱ* − *trace* (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*)) **using** *12*[*of n*] **by** *auto*
qed

have *limit-mat* (*λn. matrix-sum d* (*λk. D0 k ϱ*) (*n*)) (*denote* (*While M S*) *ϱ*) *d*
using *limit-mat-denote-while-n*[*OF wc dr pdor*] **unfolding** *D0-def M0-def M1-def DS-def* **by** *auto*
then have *limp2*: *limit-mat* (*λn. matrix-sum d* (*λk. D0 k ϱ*) (*n* + *2*)) (*denote* (*While M S*) *ϱ*) *d*
using *limit-mat-ignore-initial-segment*[*of λn. matrix-sum d* (*λk. D0 k ϱ*) (*n*) (*denote* (*While M S*) *ϱ*) *d 2*] **by** *auto*
then have *limit-mat* (*λn.* (*P* ∗ (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*)))) (*P* ∗ (*denote* (*While M S*) *ϱ*)) *d*
using *mat-mult-limit*[*OF dP*] **unfolding** *mat-mult-seq-def* **by** *auto*
then have *limPm*: (*λn. trace* (*P* ∗ (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*)))) ⟶ *trace* (*P* ∗ (*denote* (*While M S*) *ϱ*))
using *mat-trace-limit* **by** *auto*

have *limm*: (*λn. trace* (*matrix-sum d* (*λk. D0 k ϱ*) (*n+2*))) ⟶ *trace* (*denote* (*While M S*) *ϱ*)
using *mat-trace-limit limp2* **by** *auto*

have *leq-lim*: *trace* (*P* ∗ (*M0* ∗ *ϱ* ∗ *adjoint M0*)) + *trace* (*Q* ∗ (*M1* ∗ *ϱ* ∗ *adjoint M1*))
≤ *trace* (*P* ∗ (*denote* (*While M S*) *ϱ*)) + *trace ϱ* − *trace* (*denote* (*While M S*) *ϱ*) (**is** *?lhs* ≤ *?rhs*)
using *main-leq*
proof −
define *seq* **where** *seq n* = *trace* (*P* ∗ *matrix-sum d* (*λk. D0 k ϱ*) (*n* + *2*)) − *trace* (*matrix-sum d* (*λk. D0 k ϱ*) (*n* + *2*)) **for** *n*
define *seqlim* **where** *seqlim* = *trace* (*P* ∗ (*denote* (*While M S*) *ϱ*)) − *trace* (*denote* (*While M S*) *ϱ*)
have *main-leq'*: *?lhs* ≤ *trace ϱ* + *seq n* **for** *n*
unfolding *seq-def* **using** *main-leq* **by** (*simp add*: *algebra-simps*)
have *limseq*: *seq* ⟶ *seqlim*
unfolding *seq-def seqlim-def* **using** *tendsto-diff*[*OF limPm limm*] **by** *auto*
have *limrs*: (*λn. trace ϱ* + *seq n*) ⟶ (*trace ϱ* + *seqlim*) **using** *tendsto-add*[*OF - limseq*] **by** *auto*

have *limrsRe*: ($\lambda n.$ *Re* (*trace* ϱ + *seq n*)) $\longrightarrow$ *Re* (*trace* ϱ + *seqlim*) **using** *tendsto-Re*[*OF limrs*] **by** *auto*
have *main-leq-Re*: *Re ?lhs* $\leq$ *Re* (*trace* ϱ + *seq n*) **for** *n* **using** *main-leq′*
by (*auto simp*: *less-eq-complex-def*)
have *Re*: *Re ?lhs* $\leq$ *Re* (*trace* ϱ + *seqlim*)
using *Lim-bounded2*[*OF limrsRe*] *main-leq-Re* **by** (*auto simp*: *less-eq-complex-def*)

have *limrsIm*: ($\lambda n.$ *Im* (*trace* ϱ + *seq n*)) $\longrightarrow$ *Im* (*trace* ϱ + *seqlim*) **using** *tendsto-Im*[*OF limrs*] **by** *auto*
have *main-leq-Im*: *Im ?lhs* = *Im* (*trace* ϱ + *seq n*) **for** *n* **using** *main-leq′* **unfolding** *less-eq-complex-def* **by** *auto*
then have *limIm*: ($\lambda n.$ *Im* (*trace* ϱ + *seq n*)) $\longrightarrow$ *Im ?lhs* **using** *tendsto-intros*(*1*) **by** *auto*
have *Im*: *Im ?lhs* = *Im* (*trace* ϱ + *seqlim*)
using *tendsto-unique*[*OF - limIm limrsIm*] **by** *auto*

have *?lhs* $\leq$ *trace* ϱ + *seqlim* **using** *Re Im* **by** (*auto simp*: *less-eq-complex-def*)
then show *?lhs* $\leq$ *?rhs* **unfolding** *seqlim-def* **by** (*auto simp*: *less-eq-complex-def*)
qed

have *trace* ((*adjoint M0* $*$ *P* $*$ *M0* + *adjoint M1* $*$ *Q* $*$ *M1*) $*$ ϱ) =
trace (*P* $*$ (*M0* $*$ ϱ $*$ *adjoint M0*)) + *trace* (*Q* $*$ (*M1* $*$ ϱ $*$ *adjoint M1*))
using *dr dM0 dM1 dP dQ* **by** (*mat-assoc d*)
then have *trace* ((*adjoint M0* $*$ *P* $*$ *M0* + *adjoint M1* $*$ *Q* $*$ *M1*) $*$ ϱ) $\leq$
trace (*P* $*$ (*denote* (*While M S*) ϱ)) + (*trace* ϱ $-$ *trace* (*denote* (*While M S*) ϱ))
using *leq-lim* **by** (*auto simp*: *less-eq-complex-def*)
}
then show *?case* **unfolding** *hoare-partial-correct-def denote.simps*(*5*)
apply (*fold M0-def M1-def DS-def D0-def D1-def*) **by** *auto*
next
case (*6 P Q P′ Q′ S*)
then have *wcs*: *well-com S* **and** *c*: $\models_p$ {*P′*} *S* {*Q′*}
and *dP*: *P* $\in$ *carrier-mat d d* **and** *dQ*: *Q* $\in$ *carrier-mat d d*
and *dP′*: *P′* $\in$ *carrier-mat d d* **and** *dQ′*: *Q′* $\in$ *carrier-mat d d*
using *is-quantum-predicate-def* **by** *auto*
show *?case* **unfolding** *hoare-partial-correct-def*
proof
fix ϱ **assume** *pds*: $\varrho \in$ *density-states*
then have *pdor*: *partial-density-operator* ϱ **and** *dr*: $\varrho \in$ *carrier-mat d d*
using *density-states-def* **by** *auto*
have *pdoSr*: *partial-density-operator* (*denote S* ϱ)
using *denote-partial-density-operator pdor dr wcs* **by** *auto*
have *dSr*: *denote S* ϱ $\in$ *carrier-mat d d*
using *denote-dim pdor dr wcs* **by** *auto*
have *trace* (*P* $*$ ϱ) $\leq$ *trace* (*P′* $*$ ϱ) **using** *lowner-le-trace*[*OF dP dP′*] *6 dr pdor* **by** *auto*
also have ... $\leq$ *trace* (*Q′* $*$ *denote S* ϱ) + (*trace* ϱ $-$ *trace* (*denote S* ϱ))

using *c* **unfolding** *hoare-partial-correct-def* **using** *pds* **by** *auto*
also have ... $\leq$ *trace* $(Q * denote\ S\ \varrho) + (trace\ \varrho - trace\ (denote\ S\ \varrho))$ **using** *lowner-le-trace*[*OF dQ' dQ*] *6 dSr pdoSr* **by** *auto*
finally show *trace* $(P * \varrho) \leq trace\ (Q * denote\ S\ \varrho) + (trace\ \varrho - trace\ (denote\ S\ \varrho))$ **.**
qed
qed

lemma *wlp-complete*:
well-com S $\Longrightarrow$ *is-quantum-predicate P* $\Longrightarrow \vdash_p \{wlp\ S\ P\}\ S\ \{P\}$
proof (*induct S arbitrary*: *P*)
case *SKIP*
then show *?case* **unfolding** *wlp.simps* **using** *hoare-partial.intros* **by** *auto*
next
case (*Utrans U*)
then show *?case* **unfolding** *wlp.simps* **using** *hoare-partial.intros* **by** *auto*
next
case (*Seq S1 S2*)
then have *wc1*: *well-com S1* **and** *wc2*: *well-com S2* **and** *qpP*: *is-quantum-predicate P*
and *p2*: $\vdash_p \{wlp\ S2\ P\}\ S2\ \{P\}$ **by** *auto*
have *qpW2P*: *is-quantum-predicate* (*wlp S2 P*) **using** *wlp-close*[*OF wc2 qpP*] **by** *auto*
then have *p1*: $\vdash_p \{wlp\ S1\ (wlp\ S2\ P)\}\ S1\ \{wlp\ S2\ P\}$ **using** *Seq* **by** *auto*
have *qpW1W2P*: *is-quantum-predicate* (*wlp S1* (*wlp S2 P*)) **using** *wlp-close*[*OF wc1 qpW2P*] **by** *auto*
then show *?case* **unfolding** *wlp.simps* **using** *hoare-partial.intros qpW1W2P qpW2P qpP p1 p2* **by** *auto*
next
case (*Measure n M S*)
then have *wc*: *well-com* (*Measure n M S*) **and** *qpP*: *is-quantum-predicate P* **by** *auto*
have *set*: $k < n \Longrightarrow (S!k) \in set\ S$ **for** *k* **using** *wc* **by** *auto*
have *wck*: $k < n \Longrightarrow$ *well-com* $(S!k)$ **for** *k* **using** *wc measure-well-com* **by** *auto*
then have *qpWkP*: $k < n \Longrightarrow$ *is-quantum-predicate* $(wlp\ (S!k)\ P)$ **for** *k* **using** *wlp-close qpP* **by** *auto*
have *pk*: $k < n \Longrightarrow \vdash_p \{(wlp\ (S!k)\ P)\}\ (S!k)\ \{P\}$ **for** *k* **using** *Measure*(*1*) *set wck qpP* **by** *auto*
show *?case* **unfolding** *wlp-measure-expand*[*OF wc*] **using** *hoare-partial.intros qpWkP qpP pk* **by** *auto*
next
case (*While M S*)
then have *wc*: *well-com* (*While M S*) **and** *wcS*: *well-com S* **and** *qpP*: *is-quantum-predicate P* **by** *auto*
have *qpWP*: *is-quantum-predicate* (*wlp* (*While M S*) *P*) **using** *wlp-close*[*OF wc qpP*] **by** *auto*
then have *qpWWP*: *is-quantum-predicate* (*wlp S* (*wlp* (*While M S*) *P*)) **using** *wlp-close wcS* **by** *auto*
have $\vdash_p \{wlp\ S\ (wlp\ (While\ M\ S)\ P)\}\ S\ \{wlp\ (While\ M\ S)\ P\}$ **using** *While*(*1*)

wcS qpWP **by** *auto*
moreover have *eq*: *wlp* (*While M S*) *P* = *adjoint* (*M 0*) ∗ *P* ∗ *M 0* + *adjoint* (*M 1*) ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M 1*
using *wlp-while-split wc qpP* **by** *auto*
ultimately have *p*: $\vdash_p$ {*wlp S* (*wlp* (*While M S*) *P*)} *S* {*adjoint* (*M 0*) ∗ *P* ∗ *M 0* + *adjoint* (*M 1*) ∗ *wlp S* (*wlp* (*While M S*) *P*) ∗ *M 1*} **by** *auto*
then show *?case* **using** *hoare-partial.intros*(*5*)[*OF qpP qpWWP p*] *eq* **by** *auto*
qed

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 −
assume *p*: $\models_p$ {*P*} *S* {*Q*} **and** *wc*: *well-com S* **and** *qpP*: *is-quantum-predicate P* **and** *qpQ*: *is-quantum-predicate Q*
then have *dQ*: *Q* ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
have *qpWP*: *is-quantum-predicate* (*wlp S Q*) **using** *wlp-close wc qpQ* **by** *auto*
then have *dWP*: *wlp S Q* ∈ *carrier-mat d d* **using** *is-quantum-predicate-def* **by** *auto*
have *eq*: *trace* (*wlp S Q* ∗ *ϱ*) = *trace* (*Q* ∗ (*denote S ϱ*)) + *trace ϱ* − *trace* (*denote S ϱ*) **if** *dsr*: *ϱ* ∈ *density-states* **for** *ϱ*
using *wlp-soundness wc qpQ dsr* **by** *auto*
then have $\models_p$ {*wlp S Q*} *S* {*Q*} **unfolding** *hoare-partial-correct-def* **by** *auto*
{
fix *ϱ* **assume** *dsr*: *ϱ* ∈ *density-states*
then have *trace* (*P* ∗ *ϱ*) ≤ *trace* (*Q* ∗ (*denote S ϱ*)) + *trace ϱ* − *trace* (*denote S ϱ*)
using *hoare-partial-correct-def p* **by** (*auto simp*: *less-eq-complex-def*)
then have *trace* (*P* ∗ *ϱ*) ≤ *trace* (*wlp S Q* ∗ *ϱ*) **using** *eq*[*symmetric*] *dsr* **by** *auto*
}
then have *le*: *P* $\leq_L$ *wlp S Q* **using** *lowner-le-trace density-states-def qpP qpWP is-quantum-predicate-def* **by** *auto*
moreover have *wlp*: $\vdash_p$ {*wlp S Q*} *S* {*Q*} **using** *wlp-complete wc qpQ* **by** *auto*
ultimately show $\vdash_p$ {*P*} *S* {*Q*} **using** *hoare-partial.intros*(*6*)[*OF qpP qpQ qpWP qpQ*] *lowner-le-refl*[*OF dQ*] **by** *auto*
qed

6.3 Consequences of completeness

lemma *hoare-patial-seq-assoc-sem*:
shows $\models_p$ {*A*} (*S1* ;; *S2*) ;; *S3* {*B*} ⟷ $\models_p$ {*A*} *S1* ;; (*S2* ;; *S3*) {*B*}
unfolding *hoare-partial-correct-def denote.simps* **by** *auto*

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*} ⟷ $\vdash_p$ {*A*} *S1* ;; (*S2* ;; *S3*) {*B*}
proof

assume $\vdash_p$ $\{A\}$ *S1*;; *S2*;; *S3* $\{B\}$
then have $\models_p$ $\{A\}$ (*S1* ;; *S2*) ;; *S3* $\{B\}$ **using** *hoare-partial-sound assms* **by** *auto*
then have $\models_p$ $\{A\}$ *S1* ;; (*S2* ;; *S3*) $\{B\}$ **using** *hoare-patial-seq-assoc-sem* **by** *auto*
then show $\vdash_p$ $\{A\}$ *S1* ;; (*S2* ;; *S3*) $\{B\}$ **using** *hoare-partial-complete assms* **by** *auto*
next
assume $\vdash_p$ $\{A\}$ *S1*;; (*S2*;; *S3*) $\{B\}$
then have $\models_p$ $\{A\}$ *S1*;; (*S2*;; *S3*) $\{B\}$ **using** *hoare-partial-sound assms* **by** *auto*
then have $\models_p$ $\{A\}$ *S1*;; *S2*;; *S3* $\{B\}$ **using** *hoare-patial-seq-assoc-sem* **by** *auto*
then show $\vdash_p$ $\{A\}$ *S1*;; *S2*;; *S3* $\{B\}$ **using** *hoare-partial-complete assms* **by** *auto*
qed

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* $\{i.\ i < (2::nat)\ \hat{}\ n \wedge f\ i\} > 0$
card $\{i.\ i < (2::nat)\ \hat{}\ n \wedge f\ i\} < (2::nat)\ \hat{}\ n$
begin

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

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

lemma *N-ge-0* [*simp*]: $0 < N$ **by** (*simp add*: *N-def*)

lemma *M-ge-0* [*simp*]: $0 < M$ **by** (*simp add*: *M-def dimM N-def*)

lemma *M-neq-0* [*simp*]: $M \neq 0$ **by** *simp*

lemma *M-le-N* [*simp*]: $M < N$ **by** (*simp add*: *M-def dimM N-def*)

lemma *M-not-ge-N* [*simp*]: ¬ *M* ≥ *N* **using** *M-le-N* **by** *arith*

definition *ψ* :: *complex vec* **where**
ψ = *Matrix.vec N* (*λi. 1* / *sqrt N*)

lemma *ψ-dim* [*simp*]:
ψ ∈ *carrier-vec N*
dim-vec ψ = *N*
by (*simp add*: *ψ-def*)+

lemma *ψ-eval*:
i < *N* ⟹ *ψ* $ *i* = *1* / *sqrt N*
by (*simp add*: *ψ-def*)

lemma *ψ-inner*:
inner-prod ψ ψ = *1*
apply (*simp add*: *ψ-eval scalar-prod-def*)
by (*smt* (*verit*) *of-nat-less-0-iff of-real-mult of-real-of-nat-eq real-sqrt-mult-self*)

lemma *ψ-norm*:
vec-norm ψ = *1*
by (*simp add*: *ψ-eval vec-norm-def scalar-prod-def*)

definition *α* :: *complex vec* **where**
α = *Matrix.vec N* (*λi. if f i then 0 else 1* / *sqrt* (*N* − *M*))

lemma *α-dim* [*simp*]:
α ∈ *carrier-vec N*
dim-vec α = *N*
by (*simp add*: *α-def*)+

lemma *α-eval*:
i < *N* ⟹ *α* $ *i* = (*if f i then 0 else 1* / *sqrt* (*N* − *M*))
by (*simp add*: *α-def*)

lemma *α-inner*:
inner-prod α α = *1*
apply (*simp add*: *scalar-prod-def α-eval*)
apply (*subst sum.mono-neutral-cong-right*[*of* {*0..*<*N*} {*0..*<*N*}−{*i. i* < *N* ∧ *f i*}])
apply *auto*
apply (*subgoal-tac card* ({*0..*<*N*} − {*i. i* < *N* ∧ *f i*}) = *N* − *M*)
subgoal by (*metis of-nat-0-le-iff of-real-of-nat-eq of-real-power power2-eq-square real-sqrt-pow2*)
unfolding *N-def M-def*
by (*metis* (*no-types*, *lifting*) *atLeastLessThan-iff card.infinite card-Diff-subset card-atLeastLessThan diff-zero dimM*(*1*) *mem-Collect-eq neq0-conv subsetI zero-order*(*1*))

definition *β* :: *complex vec* **where**

```
  β = Matrix.vec N (λi. if f i then 1 / sqrt M else 0)

lemma β-dim [simp]:
  β ∈ carrier-vec N
  dim-vec β = N
  by (simp add: β-def)+

lemma β-eval:
  i < N ⟹ β $ i = (if f i then 1 / sqrt M else 0)
  by (simp add: β-def)

lemma β-inner:
  inner-prod β β = 1
  apply (simp add: scalar-prod-def β-eval)
  apply (subst sum.mono-neutral-cong-right[of {0..<N} {i. i < N ∧ f i}])
   apply auto
  apply (fold M-def)
 by (metis of-nat-0-le-iff of-real-of-nat-eq of-real-power power2-eq-square real-sqrt-pow2)

lemma alpha-beta-orth:
  inner-prod α β = 0
  unfolding α-def β-def by (simp add: scalar-prod-def)

lemma beta-alpha-orth:
  inner-prod β α = 0
  unfolding α-def β-def by (simp add: scalar-prod-def)

definition ϑ :: real where
  ϑ = 2 ∗ arccos (sqrt ((N − M) / N))

lemma cos-theta-div-2:
  cos (ϑ / 2) = sqrt ((N − M) / N)
proof −
  have ϑ / 2 = arccos (sqrt ((N − M) / N)) using ϑ-def by simp
  then show cos (ϑ / 2) = sqrt ((N − M) / N)
    by (simp add: cos-arccos-abs)
qed

lemma sin-theta-div-2:
  sin (ϑ / 2) = sqrt (M / N)
proof −
  have a: ϑ / 2 = arccos (sqrt ((N − M) / N)) using ϑ-def by simp
  have N: N > 0 using N-def by auto
  have M: M < N using M-def dimM N-def by auto
  then show sin (ϑ / 2) = sqrt (M / N)
    unfolding a
    apply (simp add: sin-arccos-abs)
  proof −
    have eq: real (N − M) = real N − real M using N M
```

```
      using M-not-ge-N nat-le-linear of-nat-diff by blast
    have 1 − real (N − M) / real N = (real N − (real N − real M)) / real N
      unfolding eq using N
      by (metis diff-divide-distrib divide-self-if eq gr-implies-not0 of-nat-0-eq-iff)
    then show 1 − real (N − M) / real N = real M / real N by auto
  qed
qed

lemma ϑ-neq-0:
  ϑ ≠ 0
proof −
  {
  assume ϑ = 0
  then have ϑ / 2 = 0 by auto
  then have sin (ϑ / 2) = 0 by auto
  }
  note z = this
  have sin (ϑ / 2) = sqrt (M / N) using sin-theta-div-2 by auto
  moreover have M > 0 unfolding M-def N-def using dimM by auto
  ultimately have sin (ϑ / 2) > 0 by auto
  with z show ?thesis by auto
qed

abbreviation ccos where ccos φ ≡ complex-of-real (cos φ)
abbreviation csin where csin φ ≡ complex-of-real (sin φ)

lemma ψ-eq:
  ψ = ccos (ϑ / 2) ·v α + csin (ϑ / 2) ·v β
  apply (simp add: cos-theta-div-2 sin-theta-div-2)
  apply (rule eq-vecI)
  by (auto simp add: α-def β-def ψ-def real-sqrt-divide)

lemma psi-inner-alpha:
  inner-prod ψ α = ccos (ϑ / 2)
  unfolding ψ-eq
proof −
  have inner-prod (ccos (ϑ / 2) ·v α) α = ccos (ϑ / 2)
    apply (subst inner-prod-smult-right[of - N])
    using α-dim α-inner by auto
  moreover have inner-prod (csin (ϑ / 2) ·v β) α = 0
    apply (subst inner-prod-smult-right[of - N])
    using α-dim β-dim beta-alpha-orth by auto
  ultimately show inner-prod (ccos (ϑ / 2) ·v α + csin (ϑ / 2) ·v β) α = ccos
(ϑ / 2)
    apply (subst inner-prod-distrib-left[of - N])
    using α-dim β-dim by auto
qed

lemma psi-inner-beta:
```

inner-prod ψ β = csin (ϑ / 2)
unfolding *ψ-eq*
proof −
have *inner-prod (ccos (ϑ / 2) $\cdot_v$ α) β = 0*
apply (*subst inner-prod-smult-right*[*of - N*])
using *α-dim β-dim alpha-beta-orth* **by** *auto*
moreover have *inner-prod (csin (ϑ / 2) $\cdot_v$ β) β = csin (ϑ / 2)*
apply (*subst inner-prod-smult-right*[*of - N*])
using *β-dim β-inner* **by** *auto*
ultimately show *inner-prod (ccos (ϑ / 2) $\cdot_v$ α + csin (ϑ / 2) $\cdot_v$ β) β = csin (ϑ / 2)*
apply (*subst inner-prod-distrib-left*[*of - N*])
using *α-dim β-dim* **by** *auto*
qed

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

lemma *alpha-l-real*:
alpha-l l ∈ Reals
unfolding *alpha-l-def* **by** *auto*

lemma *cnj-alpha-l*:
conjugate (alpha-l l) = alpha-l l
using *alpha-l-real Reals-cnj-iff* **by** *auto*

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

lemma *beta-l-real*:
beta-l l ∈ Reals
unfolding *beta-l-def* **by** *auto*

lemma *cnj-beta-l*:
conjugate (beta-l l) = beta-l l
using *beta-l-real Reals-cnj-iff* **by** *auto*

lemma *csin-ccos-squared-add*:
ccos (a::real) ∗ ccos a + csin a ∗ csin a = 1
by (*smt* (*verit*) *cos-diff cos-zero of-real-add of-real-hom.hom-one of-real-mult*)

lemma *alpha-l-beta-l-add-norm*:
alpha-l l ∗ alpha-l l + beta-l l ∗ beta-l l = 1
using *alpha-l-def beta-l-def csin-ccos-squared-add* **by** *auto*

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*
unfolding *psi-l-def α-def β-def* **by** *auto*

lemma *inner-psi-l*:
inner-prod (*psi-l l*) (*psi-l l*) = *1*
proof −
have *eq0*: *inner-prod* (*psi-l l*) (*psi-l l*)
= *inner-prod* ((*alpha-l l*) $\cdot_v$ α) (*psi-l l*) + *inner-prod* ((*beta-l l*) $\cdot_v$ β) (*psi-l l*)
unfolding *psi-l-def*
apply (*subst inner-prod-distrib-left*)
using *α-def β-def* **by** *auto*
have *inner-prod* ((*alpha-l l*) $\cdot_v$ α) (*psi-l l*)
= *inner-prod* ((*alpha-l l*) $\cdot_v$ α) ((*alpha-l l*) $\cdot_v$ α) + *inner-prod* ((*alpha-l l*) $\cdot_v$ α)
((*beta-l l*) $\cdot_v$ β)
unfolding *psi-l-def*
apply (*subst inner-prod-distrib-right*)
using *α-def β-def* **by** *auto*
also have . . . = (*conjugate* (*alpha-l l*)) ∗ (*alpha-l l*) ∗ *inner-prod α α*
+ (*conjugate* (*alpha-l l*)) ∗ (*beta-l l*) ∗ *inner-prod α β*
apply (*subst* (*1 2*) *inner-prod-smult-left-right*) **using** *α-def β-def* **by** *auto*
also have . . . = *conjugate* (*alpha-l l*) ∗ (*alpha-l l*)
by (*simp add*: *alpha-beta-orth α-inner*)
also have . . . = (*alpha-l l*) ∗ (*alpha-l l*) **using** *cnj-alpha-l* **by** *simp*
finally have *eq1*: *inner-prod* (*alpha-l l* $\cdot_v$ α) (*psi-l l*) = *alpha-l l* ∗ *alpha-l l*.

have *inner-prod* ((*beta-l l*) $\cdot_v$ β) (*psi-l l*)
= *inner-prod* ((*beta-l l*) $\cdot_v$ β) ((*alpha-l l*) $\cdot_v$ α) + *inner-prod* ((*beta-l l*) $\cdot_v$ β)
((*beta-l l*) $\cdot_v$ β)
unfolding *psi-l-def*
apply (*subst inner-prod-distrib-right*)
using *α-def β-def* **by** *auto*
also have . . . = (*conjugate* (*beta-l l*)) ∗ (*alpha-l l*) ∗ *inner-prod β α*
+ (*conjugate* (*beta-l l*)) ∗ (*beta-l l*) ∗ *inner-prod β β*
apply (*subst* (*1 2*) *inner-prod-smult-left-right*) **using** *α-def β-def* **by** *auto*
also have . . . = (*conjugate* (*beta-l l*)) ∗ (*beta-l l*) **using** *β-inner beta-alpha-orth*
by *auto*
also have . . . = (*beta-l l*) ∗ (*beta-l l*) **using** *cnj-beta-l* **by** *auto*
finally have *eq2*: *inner-prod* (*beta-l l* $\cdot_v$ β) (*psi-l l*) = *beta-l l* ∗ *beta-l l*.

show *?thesis* **unfolding** *eq0 eq1 eq2* **using** *alpha-l-beta-l-add-norm* **by** *auto*
qed

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*
unfolding *psi′-l-def α-def β-def* **by** *auto*

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*
unfolding *proj-psi′-l-def* **using** *psi′-l-dim* **by** *auto*

lemma *psi-inner-psi′-l*:
inner-prod ψ (*psi′-l l*) = (*alpha-l l* ∗ *ccos* (*ϑ* / *2*) − *beta-l l* ∗ *csin* (*ϑ* / *2*))
proof −
have *inner-prod ψ* (*psi′-l l*) = *inner-prod ψ* (*alpha-l l* $\cdot_v$ *α*) − *inner-prod ψ* (*beta-l l* $\cdot_v$ *β*)
unfolding *psi′-l-def* **apply** (*subst inner-prod-minus-distrib-right*[*of* - *N*]) **by** *auto*
also have . . . = *alpha-l l* ∗ (*inner-prod ψ α*) − *beta-l l* ∗ (*inner-prod ψ β*)
using *ψ-dim α-dim β-dim* **by** *auto*
also have . . . = *alpha-l l* ∗ (*ccos* (*ϑ* / *2*)) − *beta-l l* ∗ (*csin* (*ϑ* / *2*))
using *psi-inner-alpha psi-inner-beta* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *double-ccos-square*:
2 ∗ *ccos* (*a*::*real*) ∗ *ccos a* = *ccos* (*2* ∗ *a*) + *1*
proof −
have *eq*: *ccos* (*2* ∗ *a*) = *ccos a* ∗ *ccos a* − *csin a* ∗ *csin a*
using *cos-add*[*of a a*] **by** *auto*
have *csin a* ∗ *csin a* = *1* − *ccos a* ∗ *ccos a*
using *csin-ccos-squared-add*[*of a*]
by (*metis add-diff-cancel-left′*)
then have *ccos a* ∗ *ccos a* − *csin a* ∗ *csin a* = *2* ∗ *ccos a* ∗ *ccos a* − *1*
by *simp*
with *eq* **show** *?thesis* **by** *simp*
qed

lemma *double-csin-square*:
2 ∗ *csin* (*a*::*real*) ∗ *csin a* = *1* − *ccos* (*2* ∗ *a*)
proof −
have *eq*: *ccos* (*2* ∗ *a*) = *ccos a* ∗ *ccos a* − *csin a* ∗ *csin a*
using *cos-add*[*of a a*] **by** *auto*
have *ccos a* ∗ *ccos a* = *1* − *csin a* ∗ *csin a*
using *csin-ccos-squared-add*[*of a*]
by (*auto intro*: *add-implies-diff*)
then have *ccos a* ∗ *ccos a* − *csin a* ∗ *csin a* = *1* − *2* ∗ *csin* (*a*::*real*) ∗ *csin a*
by *simp*
with *eq* **show** *?thesis* **by** *simp*
qed

lemma *csin-double*:
2 ∗ *csin* (*a*::*real*) ∗ *ccos a* = *csin*(*2* ∗ *a*)
using *sin-add*[*of a a*] **by** *simp*

lemma *ccos-add*:
ccos (*x* + *y*) = *ccos x* ∗ *ccos y* − *csin x* ∗ *csin y*
using *cos-add*[*of x y*] **by** *simp*

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 −
have *2* ∗ ((*alpha-l l*) ∗ *ccos* (*ϑ* / *2*) − (*beta-l l*) ∗ *csin* (*ϑ* / *2*)) ∗ *ccos* (*ϑ* / *2*)
= (*alpha-l l*) ∗ (*2* ∗ *ccos* (*ϑ* / *2*)∗ *ccos* (*ϑ* / *2*)) − (*beta-l l*) ∗ (*2* ∗ *csin* (*ϑ* / *2*) ∗ *ccos* (*ϑ* / *2*))
by (*simp add*: *left-diff-distrib*)

also have . . . = (*alpha-l l*) ∗ (*ccos* (*ϑ*) + *1*) − (*beta-l l*) ∗ *csin ϑ*
using *double-ccos-square csin-double* **by** *auto*
finally have *2* ∗ ((*alpha-l l*) ∗ *ccos* (*ϑ* / *2*) − (*beta-l l*) ∗ *csin* (*ϑ* / *2*)) ∗ *ccos* (*ϑ* / *2*)
= (*alpha-l l*) ∗ (*ccos* (*ϑ*) + *1*) − (*beta-l l*) ∗ *csin ϑ*.
then have *?lhs* = (*alpha-l l*) ∗ *ccos* (*ϑ*) − (*beta-l l*) ∗ *csin ϑ* **by** (*simp add*: *algebra-simps*)
also have . . . = (*alpha-l* (*l* + *1*))
unfolding *alpha-l-def beta-l-def*
apply (*subst ccos-add*[*of* (*real l* + *1* / *2*) ∗ *ϑ ϑ*, *symmetric*])
by (*simp add*: *algebra-simps*)
finally show *?thesis* **by** *auto*
qed

lemma *csin-add*:
csin (*x* + *y*) = *ccos x* ∗ *csin y* + *csin x* ∗ *ccos y*
using *sin-add*[*of x y*] **by** *simp*

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 −
have *2* ∗ ((*alpha-l l*) ∗ *ccos* (*ϑ* / *2*) − (*beta-l l*) ∗ *csin* (*ϑ* / *2*)) ∗ *csin* (*ϑ* / *2*)
= (*alpha-l l*) ∗ (*2* ∗ *csin* (*ϑ* / *2*)∗ *ccos* (*ϑ* / *2*)) − (*beta-l l*) ∗ (*2* ∗ *csin* (*ϑ* / *2*) ∗ *csin* (*ϑ* / *2*))
by (*simp add*: *left-diff-distrib*)
also have . . . = (*alpha-l l*) ∗ (*csin ϑ*) − (*beta-l l*) ∗ (*1* − *ccos* (*ϑ*))
using *double-csin-square csin-double* **by** *auto*
finally have *2* ∗ ((*alpha-l l*) ∗ *ccos* (*ϑ* / *2*) − (*beta-l l*) ∗ *csin* (*ϑ* / *2*)) ∗ *csin*

$(\vartheta\ /\ 2)$
  $= (alpha\text{-}l\ l) * (csin\ \vartheta) - (beta\text{-}l\ l) * (1 - ccos\ (\vartheta))$.
 then have *?lhs* $= (alpha\text{-}l\ l) * (csin\ \vartheta) + (beta\text{-}l\ l) * ccos\ \vartheta$ **by** (*simp add*: *algebra-simps*)
 also have ... $= (beta\text{-}l\ (l + 1))$
  unfolding *alpha-l-def beta-l-def*
  apply (*subst csin-add*[*of* (*real l* + *1* / *2*) ∗ ϑ ϑ, *symmetric*])
  by (*simp add*: *algebra-simps*)
 finally show *?thesis* **by** *auto*
qed

lemma *psi-l-Suc-l-derive*:
 $2 * (alpha\text{-}l\ l * ccos\ (\vartheta\ /\ 2) - beta\text{-}l\ l * csin\ (\vartheta\ /\ 2)) \cdot_v \psi - psi'\text{-}l\ l = psi\text{-}l\ (l + 1)$
 (**is** *?lhs* = *?rhs*)
proof −
 let $?l = 2 * ((alpha\text{-}l\ l) * ccos\ (\vartheta\ /\ 2) - (beta\text{-}l\ l) * csin\ (\vartheta\ /\ 2))$
 have $?l \cdot_v \psi = ?l \cdot_v (ccos\ (\vartheta\ /\ 2) \cdot_v \alpha + csin\ (\vartheta\ /\ 2) \cdot_v \beta)$ **unfolding** *ψ-eq* **by** *auto*
 also have ... $= ?l \cdot_v (ccos\ (\vartheta\ /\ 2) \cdot_v \alpha) + ?l \cdot_v (csin\ (\vartheta\ /\ 2) \cdot_v \beta)$
  apply (*subst smult-add-distrib-vec*[*of* - *N*]) **using** *α-dim β-dim* **by** *auto*
 also have ... $= (?l * ccos\ (\vartheta\ /\ 2)) \cdot_v \alpha + (?l * csin\ (\vartheta\ /\ 2)) \cdot_v \beta$ **by** *auto*
 finally have $?l \cdot_v \psi = (?l * ccos\ (\vartheta\ /\ 2)) \cdot_v \alpha + (?l * csin\ (\vartheta\ /\ 2)) \cdot_v \beta$.
 then have $?l \cdot_v \psi - (psi'\text{-}l\ l) = ((?l * ccos\ (\vartheta\ /\ 2)) \cdot_v \alpha - (alpha\text{-}l\ l) \cdot_v \alpha) + ((?l * csin\ (\vartheta\ /\ 2)) \cdot_v \beta + (beta\text{-}l\ l) \cdot_v \beta)$
  unfolding *psi′-l-def* **by** *auto*
 also have ... $= (?l * ccos\ (\vartheta\ /\ 2) - alpha\text{-}l\ l) \cdot_v \alpha + (?l * csin\ (\vartheta\ /\ 2) + beta\text{-}l\ l) \cdot_v \beta$
  apply (*subst minus-smult-vec-distrib*) **apply** (*subst add-smult-distrib-vec*) **by** *auto*
 also have ... $= (alpha\text{-}l\ (l + 1)) \cdot_v \alpha + (beta\text{-}l\ (l + 1)) \cdot_v \beta$
  using *alpha-l-Suc-l-derive beta-l-Suc-l-derive* **by** *auto*
 finally have $?l \cdot_v \psi - (psi'\text{-}l\ l) = (alpha\text{-}l\ (l + 1)) \cdot_v \alpha + (beta\text{-}l\ (l + 1)) \cdot_v \beta$.
 then show *?thesis* **unfolding** *psi-l-def* **by** *auto*
qed

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-O ∈ *carrier-mat N N*
 unfolding *proj-O-def* **by** *auto*

lemma *proj-O-mult-alpha*:
 $proj\text{-}O *_v \alpha = zero\text{-}vec\ N$
 by (*auto simp add*: *proj-O-def α-def scalar-prod-def*)

lemma *proj-O-mult-beta*:
  proj-O $*_v$ β = β
  by (*auto simp add*: *proj-O-def* β*-def scalar-prod-def sum-only-one-neq-0*)

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

lemma *mat-O-dim*:
  mat-O ∈ *carrier-mat N N*
  unfolding *mat-O-def* **by** *auto*

lemma *mat-O-mult-alpha*:
  mat-O $*_v$ α = α
  by (*auto simp add*: *mat-O-def* α*-def scalar-prod-def sum-only-one-neq-0*)

lemma *mat-O-mult-beta*:
  mat-O $*_v$ β = − β
  by (*auto simp add*: *mat-O-def* β*-def scalar-prod-def sum-only-one-neq-0*)

lemma *hermitian-mat-O*:
  hermitian mat-O
  by (*auto simp add*: *hermitian-def mat-O-def adjoint-eval*)

lemma *unitary-mat-O*:
  unitary mat-O
proof −
  have *mat-O* ∈ *carrier-mat N N* **unfolding** *mat-O-def* **by** *auto*
  moreover have *mat-O* ∗ *adjoint mat-O* = *mat-O* ∗ *mat-O* **using** *hermitian-mat-O* **unfolding** *hermitian-def* **by** *auto*
  moreover have *mat-O* ∗ *mat-O* = 1_m *N*
    apply (*rule eq-matI*)
    unfolding *mat-O-def*
      apply (*simp add*: *scalar-prod-def*)
    subgoal for *i j* **apply** (*rule*)
      subgoal apply (*subst sum-only-one-neq-0*[*of* {*0*..<*N*} *j*]) **by** *auto*
        apply (*subst sum-only-one-neq-0*[*of* {*0*..<*N*} *j*]) **by** *auto*
    by *auto*
  ultimately show *?thesis* **unfolding** *unitary-def inverts-mat-def* **by** *auto*
qed

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

lemma *hermitian-mat-Ph*:
  hermitian mat-Ph
  unfolding *hermitian-def mat-Ph-def*
  apply (*rule eq-matI*)
  by (*auto simp add*: *adjoint-eval*)

lemma *unitary-mat-Ph*:
  unitary mat-Ph
proof −
  have *mat-Ph* ∈ *carrier-mat N N* **unfolding** *mat-Ph-def* **by** *auto*
  moreover have *mat-Ph* ∗ *adjoint mat-Ph* = *mat-Ph* ∗ *mat-Ph* **using** *hermitian-mat-Ph* **unfolding** *hermitian-def* **by** *auto*
  moreover have *mat-Ph* ∗ *mat-Ph* = 1_m *N*
    apply (*rule eq-matI*)
    unfolding *mat-Ph-def*
      apply (*simp add*: *scalar-prod-def*)
    subgoal for *i j* **apply** (*rule*)
      subgoal apply (*subst sum-only-one-neq-0*[*of* {*0..<N*} *0*]) **by** *auto*
        apply (*subst sum-only-one-neq-0*[*of* {*0..<N*} *j*]) **by** *auto*
    by *auto*
  ultimately show *?thesis* **unfolding** *unitary-def inverts-mat-def* **by** *auto*
qed

definition *mat-G′* :: *complex mat* **where**
  mat-G′ = *mat N N* (λ(*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*
  using *K* **by** *auto*

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*
unfolding *dims-def* **by** *auto*

lemma *dims-nth-lt-n*:
l < *n* ⟹ *nth dims l* = *2*
unfolding *dims-def* **by** (*simp add*: *nth-append*)

lemma *nths-Suc-n-dims*:
nths dims {*0*..<(*Suc n*)} = *dims*
using *length-dims nths-upt-eq-take*
by (*metis add-Suc-right add-Suc-shift lessThan-atLeast0 less-add-eq-less less-numeral-extra*(*4*) *not-less plus-1-eq-Suc take-all*)

interpretation *ps2-P*: *partial-state2 dims vars1 vars2*
apply *unfold-locales* **unfolding** *vars1-def vars2-def* **by** *auto*

interpretation *ps-P*: *partial-state ps2-P.dims0 ps2-P.vars1'* **.**

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 −
have *ps2-P.d0* = *prod-list* (*nths dims* ({*0*..<*n*} ∪ {*n*})) **unfolding** *ps2-P.d0-def ps2-P.dims0-def ps2-P.vars0-def*
by (*simp add*: *vars1-def vars2-def*)
also have ... = *prod-list* (*nths dims* ({*0*..<*Suc n*}))
apply (*subgoal-tac* {*0*..<*n*} ∪ {*n*} = {*0*..<(*Suc n*)}) **by** *auto*
also have ... = *prod-list dims* **using** *nths-Suc-n-dims* **by** *auto*
also have ... = *d* **unfolding** *d-def* **by** *auto*
finally show *?thesis* **using** *ps2-P.ptensor-mat-carrier* **by** *auto*
qed

lemma *dims-nths-le-n*:
assumes *l* ≤ *n*
shows *nths dims* {*0*..<*l*} = *replicate l 2*
proof (*rule nth-equalityI*, *auto*)
have *l* ≤ *n* ⟹ (*i* < *Suc n* ∧ *i* < *l*) = (*i* < *l*) **for** *i*
using *less-trans* **by** *fastforce*
then show *l*: *length* (*nths dims* {*0*..<*l*}) = *l* **using** *assms*
by (*auto simp add*: *length-nths length-dims*)

have *llt*: *l* < *length dims* **using** *length-dims assms* **by** *auto*
have *v1*: ⋀*i*. *i* < *l* ⟹ {*a*. *a* < *i* ∧ *a* ∈ {*0*..<*l*}} = {*0*..<*i*} **unfolding** *vars1-def*

by *auto*
then have ⋀*i*. *i* < *l* ⟹ *card* {*j*. *j* < *i* ∧ *j* ∈ {*0*..<*l*}} = *i* **by** *auto*
then have *nths dims* {*0*..<*l*} ! *i* = *dims* ! *i* **if** *i* < *l* **for** *i*
using *nth-nths-card*[*of i dims* {*0*..<*l*}] *that llt* **by** *auto*
moreover have *dims* ! *i* = *replicate n 2* ! *i* **if** *i* < *n* **for** *i* **unfolding** *dims-def*
by (*auto simp add*: *nth-append that*)
moreover have *replicate n 2* ! *i* = *replicate l 2* ! *i* **if** *i* < *l* **for** *i* **using** *assms that* **by** *auto*
ultimately show *nths dims* {*0*..<*l*} ! *i* = *replicate l 2* ! *i* **if** *i* < *length* (*nths dims* {*0*..<*l*}) **for** *i*
using *l that assms* **by** *auto*
qed

lemma *dims-nths-one-lt-n*:
assumes *l* < *n*
shows *nths dims* {*l*} = [*2*]
proof −
have {*i*. *i* < *length dims* ∧ *i* ∈ {*l*}} = {*l*} **using** *assms length-dims* **by** *auto*
then have *nths dims* {*l*} = [*dims* ! *l*] **using** *nths-only-one*[*of dims* {*l*} *l*] **by** *auto*
moreover have *dims* ! *l* = *2* **unfolding** *dims-def* **using** *assms* **by** (*simp add*: *nth-append*)
ultimately show *?thesis* **by** *auto*
qed

lemma *dims-vars1*:
nths dims vars1 = *replicate n 2*
proof (*rule nth-equalityI*, *auto*)
show *l*: *length* (*nths dims vars1*) = *n*
apply (*auto simp add*: *length-nths vars1-def length-dims*)
by (*metis* (*no-types*, *lifting*) *Collect-cong Suc-lessD card-Collect-less-nat not-less-eq*)

have *v1*: ⋀*i*. *i* < *n* ⟹ {*a*. *a* < *i* ∧ *a* ∈ *vars1*} = {*0*..<*i*} **unfolding** *vars1-def* **by** *auto*
then have ⋀*i*. *i* < *n* ⟹ *card* {*j*. *j* < *i* ∧ *j* ∈ *vars1*} = *i* **by** *auto*
then have *nths dims vars1* ! *i* = *dims* ! *i* **if** *i* < *n* **for** *i*
using *nth-nths-card*[*of i dims vars1*] *that length-dims vars1-def* **by** *auto*
moreover have *dims* ! *i* = *replicate n 2* ! *i* **if** *i* < *n* **for** *i* **unfolding** *dims-def*
by (*simp add*: *nth-append that*)
ultimately show *nths dims vars1* ! *i* = *replicate n 2* ! *i* **if** *i* < *length* (*nths dims vars1*) **for** *i*
using *l that* **by** *auto*
qed

lemma *nths-rep-2-n*:
nths (*replicate n 2*) {*n*} = []
by (*metis* (*no-types*, *lifting*) *Collect-empty-eq card.empty length-0-conv length-replicate less-Suc-eq not-less-eq nths-replicate singletonD*)

lemma *dims-vars2*:

nths dims vars2 = [*K*]
unfolding *dims-def vars2-def*
apply (*subst nths-append*)
apply (*subst nths-rep-2-n*)
by *simp*

lemma *d-vars1*:
prod-list (*nths dims vars1*) = *N*
proof −
have *eq*: {*0..<n*} = {*..<n*} **by** *auto*
have *nths* (*replicate n 2* @ [*K*]) {*0..<n*} = (*replicate n 2*)
apply (*subst eq*)
using *nths-upt-eq-take* **by** *simp*
then show *?thesis* **unfolding** *dims-def vars1-def N-def* **by** *auto*
qed

lemma *ps2-P-dims0*:
ps2-P.dims0 = *dims*
proof −
have *vars1* ∪ *vars2* = {*0..<Suc n*} **unfolding** *vars1-def vars2-def* **by** *auto*
then have *dims*: *nths dims* (*vars1* ∪ *vars2*) = *dims* **unfolding** *vars1-def vars2-def* **using** *nths-Suc-n-dims* **by** *auto*
then show *?thesis* **unfolding** *ps2-P.dims0-def ps2-P.vars0-def* **apply** (*subst dims*) **by** *auto*
qed

lemma *ps2-P-vars1'*:
ps2-P.vars1' = *vars1*
unfolding *ps2-P.vars1'-def ps2-P.vars0-def*
proof −
have *eq*: *vars1* ∪ *vars2* = {*0..<*(*Suc n*)} **unfolding** *vars1-def vars2-def* **by** *auto*
have *x* < *Suc n* ⟹ {*i* ∈ {*0..<Suc n*}. *i* < *x*} = {*i*. *i* < *x*} **for** *x* **by** *auto*
then have *x* < *Suc n* ⟹ *ind-in-set* {*0..<*(*Suc n*)} *x* = *x* **for** *x* **unfolding** *ind-in-set-def* **by** *auto*
then have *x* ∈ *vars1* ⟹ *ind-in-set* {*0..<*(*Suc n*)} *x* = *x* **for** *x* **unfolding** *vars1-def* **by** *auto*
then have *ind-in-set* {*0..<*(*Suc n*)} ‘ *vars1* = *vars1* **by** *force*
with *eq* **show** *ind-in-set* (*vars1* ∪ *vars2*) ‘ *vars1* = *vars1* **by** *auto*
qed

lemma *ps2-P-d0*:
ps2-P.d0 = *d*
unfolding *ps2-P.d0-def* **using** *ps2-P-dims0 d-def* **by** *auto*

lemma *ps2-P-d1*:
ps2-P.d1 = *N*
unfolding *ps2-P.d1-def ps2-P.dims1-def* **by** (*simp add*: *dims-vars1 N-def*)

lemma *ps2-P-d2*:

ps2-P.d2 = *K*
unfolding *ps2-P.d2-def ps2-P.dims2-def* **by** (*simp add*: *dims-vars2*)

lemma *ps-P-d*:
ps-P.d = *d*
unfolding *ps-P.d-def ps2-P-dims0* **by** *auto*

lemma *ps-P-d1*:
ps-P.d1 = *N*
unfolding *ps-P.d1-def ps-P.dims1-def ps2-P.nths-vars1'* **using** *ps2-P-d1* **unfolding** *ps2-P.d1-def* **by** *auto*

lemma *ps-P-d2*:
ps-P.d2 = *K*
unfolding *ps-P.d2-def ps-P.dims2-def ps2-P.nths-vars2'* **using** *ps2-P-d2* **unfolding** *ps2-P.d2-def* **by** *auto*

lemma *nths-uminus-vars1*:
nths dims (− *vars1*) = *nths dims vars2*
using *ps2-P.nths-vars2'* **unfolding** *ps2-P-dims0 ps2-P-vars1' ps2-P.dims2-def* **by** *auto*

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 −
have *eq*:{*0..<n*} = {*..<n*} **by** *auto*
have (*nths dims vars1*) = *replicate n 2*
unfolding *dims-def vars1-def* **apply** (*subst eq*)
by (*simp add*: *nths-upt-eq-take*[*of* (*replicate n 2* @ [*K*]) *n*])

have *ps2-P.d1* = *2^n* **unfolding** *ps2-P.d1-def ps2-P.dims1-def* **using** *d-vars1 N-def* **by** *auto*
moreover have *ps2-P.d2* = *K* **unfolding** *ps2-P.d2-def ps2-P.dims2-def* **using** *dims-vars2* **by** *auto*

ultimately show *?thesis* **apply** (*subst ps2-P.ptensor-mat-mult*) **using** *assms* **by** *auto*
qed

lemma *mat-ext-vars1*:
shows *mat-extension dims vars1 A* = *tensor-P A* (1_m *K*)
unfolding *Utrans-P-def ps2-P.ptensor-mat-def partial-state.mat-extension-def partial-state.d2-def partial-state.dims2-def ps2-P.nths-vars2'*[*simplified ps2-P-dims0 ps2-P-vars1'*]

using *ps2-P-d2* **unfolding** *ps2-P.d2-def* **using** *ps2-P-dims0 ps2-P-vars1'* **by** *auto*

lemma *Utrans-P-is-tensor-P1*:
Utrans-P vars1 A = Utrans (tensor-P A (1_m K))
unfolding *Utrans-P-def ps2-P.ptensor-mat-def partial-state.mat-extension-def partial-state.d2-def partial-state.dims2-def ps2-P.nths-vars2'[simplified ps2-P-dims0 ps2-P-vars1']*
using *ps2-P-d2* **unfolding** *ps2-P.d2-def* **using** *ps2-P-dims0 ps2-P-vars1'* **by** *auto*

lemma *nths-dims-uminus-vars2*:
nths dims (−vars2) = nths dims vars1
proof −
have *nths dims (−vars2) = nths dims ({0..<length dims} − vars2)*
using *nths-minus-eq* **by** *auto*
also have ... = *nths dims vars1* **unfolding** *vars1-def vars2-def length-dims*
apply (*subgoal-tac* {*0..<n + 1*} − {*n*} = {*0..<n*}) **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *mat-ext-vars2*:
assumes *A ∈ carrier-mat K K*
shows *mat-extension dims vars2 A = tensor-P (1_m N) A*
proof −
have *mat-extension dims vars2 A = tensor-mat dims vars2 A (1_m N)*
unfolding *Utrans-P-def partial-state.mat-extension-def partial-state.d2-def partial-state.dims2-def nths-dims-uminus-vars2 dims-vars1 N-def* **by** *auto*
also have ... = *tensor-mat dims vars1 (1_m N) A*
apply (*subst tensor-mat-comm[of vars1 vars2]*)
subgoal unfolding *vars1-def vars2-def* **by** *auto*
subgoal unfolding *length-dims vars1-def vars2-def* **by** *auto*
subgoal unfolding *dims-vars1 N-def* **by** *auto*
unfolding *dims-vars2* **using** *assms* **by** *auto*
finally show *mat-extension dims vars2 A = tensor-P (1_m N) A*
unfolding *ps2-P.ptensor-mat-def ps2-P-dims0 ps2-P-vars1'* **by** *auto*
qed

lemma *Utrans-P-is-tensor-P2*:
assumes *A ∈ carrier-mat K K*
shows *Utrans-P vars2 A = Utrans (tensor-P (1_m N) A)*
unfolding *Utrans-P-def* **using** *mat-ext-vars2 assms* **by** *auto*

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))
  unfolding ps2-P.ptensor-mat-def
  apply (subst ps-P.tensor-mat-unitary)
  subgoal using ps-P-d1 mat-O-def by auto
  subgoal using ps-P-d2 by auto
  subgoal using unitary-mat-O by auto
  using unitary-one by auto

lemma unitary-ex-mat-Ph:
  unitary (tensor-P mat-Ph (1_m K))
  unfolding ps2-P.ptensor-mat-def
  apply (subst ps-P.tensor-mat-unitary)
  subgoal using ps-P-d1 mat-Ph-def by auto
  subgoal using ps-P-d2 by auto
  subgoal using unitary-mat-Ph by auto
  using unitary-one by auto

lemma unitary-hadamard-on-i:
  assumes k < n
  shows unitary (hadamard-on-i k)
proof -
  interpret st2: partial-state2 dims {k} vars1 - {k}
    apply unfold-locales by auto
 show ?thesis unfolding hadamard-on-i-def st2.pmat-extension-def st2.ptensor-mat-def
    apply (rule partial-state.tensor-mat-unitary)
     subgoal unfolding partial-state.d1-def partial-state.dims1-def st2.nths-vars1'
st2.dims1-def
      using dims-nths-one-lt-n assms hadamard-dim by auto
   subgoal unfolding st2.d2-def st2.dims2-def partial-state.d2-def partial-state.dims2-def
st2.nths-vars2' st2.dims1-def
      by auto
    subgoal using unitary-hadamard by auto
    subgoal using unitary-one by auto
```

```
    done
qed

lemma unitary-exhadamard-on-i:
  assumes k < n
  shows unitary (tensor-P (hadamard-on-i k) (1_m K))
proof -
  interpret st2: partial-state2 dims {k} vars1 - {k}
    apply unfold-locales by auto
  have d1: st2.d0 = partial-state.d1 ps2-P.dims0 ps2-P.vars1'
   unfolding partial-state.d1-def partial-state.dims1-def ps2-P.nths-vars1' ps2-P.dims1-def
      st2.d0-def st2.dims0-def st2.vars0-def using assms
    apply (subgoal-tac {k} ∪ (vars1 - {k}) = vars1) apply simp
    unfolding vars1-def by auto
  show ?thesis
  unfolding ps2-P.ptensor-mat-def
  apply (rule partial-state.tensor-mat-unitary)
  subgoal unfolding hadamard-on-i-def st2.pmat-extension-def
    using st2.ptensor-mat-carrier[of hadamard 1_m st2.d2]
    using d1 by auto
  subgoal unfolding partial-state.d2-def partial-state.dims2-def ps2-P.nths-vars2'
ps2-P.dims2-def dims-vars2 by auto
  using unitary-hadamard-on-i unitary-one assms by auto
qed

lemma hadamard-on-i-dim:
  assumes k < n
  shows hadamard-on-i k ∈ carrier-mat N N
proof -
  interpret st: partial-state2 dims {k} (vars1 - {k})
    apply unfold-locales by auto
  have vars1: {k} ∪ (vars1 - {k}) = vars1 unfolding vars1-def using assms by
auto
 show ?thesis unfolding hadamard-on-i-def N-def using st.pmat-extension-carrier
unfolding st.d0-def st.dims0-def st.vars0-def
    using vars1 dims-vars1 by auto
qed

lemma well-com-hadamard-k:
  k ≤ n ⟹ well-com (hadamard-n k)
proof (induct k)
  case 0
  then show ?case by auto
next
  case (Suc n)
  then have well-com (hadamard-n n) by auto
 then show ?case unfolding hadamard-n.simps well-com.simps using tensor-P-dim
unitary-exhadamard-on-i Suc by auto
qed
```

lemma *well-com-hadamard-n*:
well-com (*hadamard-n n*)
using *well-com-hadamard-k* **by** *auto*

lemma *well-com-mat-O*:
well-com (*Utrans-P vars1 mat-O*)
apply (*subst Utrans-P-is-tensor-P1*)
apply *simp* **using** *tensor-P-dim unitary-ex-mat-O* **by** *auto*

lemma *well-com-mat-Ph*:
well-com (*Utrans-P vars1 mat-Ph*)
apply (*subst Utrans-P-is-tensor-P1*)
apply *simp* **using** *tensor-P-dim unitary-ex-mat-Ph* **by** *auto*

lemma *unitary-exmat-incr*:
unitary (*tensor-P* (1_m *N*) (*mat-incr K*))
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-unitary*)
using *unitary-mat-incr K unitary-one* **by** (*auto simp add*: *ps-P-d1 ps-P-d2 mat-incr-def*)

lemma *well-com-mat-incr*:
well-com (*Utrans-P vars2* (*mat-incr K*))
apply (*subst Utrans-P-is-tensor-P2*)
apply (*simp add*: *mat-incr-def*) **using** *tensor-P-dim unitary-exmat-incr* **by** *auto*

lemma *well-com-D*: *well-com D*
unfolding *D-def* **apply** *auto*
using *well-com-hadamard-n well-com-mat-incr well-com-mat-O well-com-mat-Ph*

by *auto*

Test at while loop

definition *M0* :: *complex mat* **where**
M0 = *mat K K* (λ(*i*,*j*). *if* $i = j \wedge i \geq R$ *then 1 else 0*)

lemma *hermitian-M0*:
hermitian M0
by (*auto simp add*: *hermitian-def M0-def adjoint-eval*)

lemma *M0-dim*:
M0 ∈ *carrier-mat K K*
unfolding *M0-def* **by** *auto*

lemma *M0-mult-M0*:
M0 ∗ *M0* = *M0*
by (*auto simp add*: *M0-def scalar-prod-def sum-only-one-neq-0*)

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*
unfolding *M1-def* **by** *auto*

lemma *hermitian-M1*:
hermitian M1
by (*auto simp add*: *hermitian-def M1-def adjoint-eval*)

lemma *M1-mult-M1*:
M1 ∗ *M1* = *M1*
by (*auto simp add*: *M1-def scalar-prod-def sum-only-one-neq-0*)

lemma *M1-add-M0*:
M1 + *M0* = 1_m *K*
unfolding *M0-def M1-def* **by** *auto*

Test at the end

definition *testN* :: *nat* ⇒ *complex mat* **where**
testN k = *mat N N* (λ(*i*,*j*). *if i* = *k* ∧ *j* = *k then 1 else 0*)

lemma *hermitian-testN*:
hermitian (*testN k*)
unfolding *hermitian-def testN-def*
by (*auto simp add*: *scalar-prod-def adjoint-eval*)

lemma *testN-mult-testN*:
testN k ∗ *testN k* = *testN k*
unfolding *testN-def*
by (*auto simp add*: *scalar-prod-def sum-only-one-neq-0*)

lemma *testN-dim*:
testN k ∈ *carrier-mat N N*
unfolding *testN-def* **by** *auto*

definition *test-fst-k* :: *nat* ⇒ *complex mat* **where**
test-fst-k k = *mat N N* (λ(*i*, *j*). *if* (*i* = *j* ∧ *i* < *k*) *then 1 else 0*)

lemma *sum-test-k*:
assumes *m* ≤ *N*
shows *matrix-sum N* (λ*k*. *testN k*) *m* = *test-fst-k m*
proof −
have *m* ≤ *N* ⟹ *matrix-sum N* (λ*k*. *testN k*) *m* = *mat N N* (λ(*i*, *j*). *if* (*i* = *j* ∧ *i* < *m*) *then 1 else 0*) **for** *m*
proof (*induct m*)
case *0*
then show *?case* **apply** *simp* **apply** (*rule eq-matI*) **by** *auto*

next
case (*Suc m*)
then have *m*: $m < N$ **by** *auto*
then have *m'*: $m \leq N$ **by** *auto*
have *matrix-sum N testN* (*Suc m*) = *testN m* + *matrix-sum N testN m* **by** *simp*
also have ... = *mat N N* (λ(*i*, *j*). *if* ($i = j \wedge i <$ (*Suc m*)) *then 1 else 0*)
unfolding *testN-def Suc*(*1*)[*OF m'*] **apply** (*rule eq-matI*) **by** *auto*
finally show *?case* **by** *auto*
qed
then show *?thesis* **unfolding** *test-fst-k-def* **using** *assms* **by** *auto*
qed

lemma *test-fst-kN*:
test-fst-k N = 1_m *N*
apply (*rule eq-matI*)
unfolding *test-fst-k-def* **by** *auto*

lemma *matrix-sum-tensor-P1*:
($\bigwedge k$. $k < m \Longrightarrow g\ k \in$ *carrier-mat N N*) $\Longrightarrow$ ($A \in$ *carrier-mat K K*) $\Longrightarrow$
matrix-sum d (λk. *tensor-P* (*g k*) *A*) *m* = *tensor-P* (*matrix-sum N g m*) *A*
proof (*induct m*)
case *0*
show *?case* **apply** (*simp*) **unfolding** *ps2-P.ptensor-mat-def*
using *ps-P.tensor-mat-zero1*[*simplified ps-P-d ps-P-d1*, *of A*] **by** *auto*
next
case (*Suc m*)
then have *ind*: *matrix-sum d* (λk. *tensor-P* (*g k*) *A*) *m* = *tensor-P* (*matrix-sum N g m*) *A*
and *dk*: $\bigwedge k$. $k < m \Longrightarrow g\ k \in$ *carrier-mat N N* **and** $A \in$ *carrier-mat K K* **by** *auto*
have *ds*: *matrix-sum N g m* $\in$ *carrier-mat N N* **apply** (*subst matrix-sum-dim*)
using *dk* **by** *auto*
show *?case* **apply** *simp*
apply (*subst ind*)
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-add1*)
unfolding *ps-P-d1 ps-P-d2* **using** *Suc ds* **by** *auto*
qed

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*))
unfolding *Measure-P-def* **apply** *auto*
proof −

have *eq0*: $\bigwedge$*n. mat-extension dims vars1* (*testN n*) = *tensor-P* (*testN n*) (1_m *K*)
unfolding *mat-ext-vars1* **by** *auto*
have *eq1*: *adjoint* (*tensor-P* (*testN j*) (1_m *K*)) ∗ *tensor-P* (*testN j*) (1_m *K*) = *tensor-P* (*testN j*) (1_m *K*) **for** *j*
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-adjoint*)
apply (*auto simp add*: *ps-P-d1 ps-P-d2 testN-dim hermitian-testN*[*unfolded hermitian-def*] *hermitian-one*[*unfolded hermitian-def*])
apply (*subst ps-P.tensor-mat-mult*[*symmetric*])
by (*auto simp add*: *ps-P-d1 ps-P-d2 testN-dim testN-mult-testN*)
have *measurement d N* (λ*n. tensor-P* (*testN n*) (1_m *K*))
unfolding *measurement-def*
apply (*simp add*: *tensor-P-dim*)
apply (*subst eq1*)
apply (*subst matrix-sum-tensor-P1*)
apply (*auto simp add*: *testN-dim*)
apply (*subst sum-test-k*, *simp*)
apply (*subst test-fst-kN*)
unfolding *ps2-P.ptensor-mat-def*
using *ps-P.tensor-mat-id ps-P-d ps-P-d1 ps-P-d2* **by** *auto*
then show *measurement d N* (λ*n. mat-extension dims vars1* (*testN n*)) **using** *eq0* **by** *auto*

show *list-all well-com* (*replicate N SKIP*)
apply (*subst list-all-length*) **by** *simp*
qed

lemma *well-com-while*:
well-com (*While-P vars2 M0 M1 D*)
unfolding *While-P-def* **apply** *auto*
apply (*subst* (*1 2*) *mat-ext-vars2*)
apply (*auto simp add*: *M1-dim M0-dim*)
proof −
have *2*: *2* = *Suc* (*Suc 0*) **by** *auto*
have *ad0*: *adjoint* (*tensor-P* (1_m *N*) *M0*) = (*tensor-P* (1_m *N*) *M0*)
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-adjoint*)
unfolding *ps-P-d1 ps-P-d2* **by** (*auto simp add*: *M0-dim adjoint-one hermitian-M0*[*unfolded hermitian-def*])
have *ad1*: *adjoint* (*tensor-P* (1_m *N*) *M1*) = (*tensor-P* (1_m *N*) *M1*)
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-adjoint*)
unfolding *ps-P-d1 ps-P-d2* **by** (*auto simp add*: *M1-dim adjoint-one hermitian-M1*[*unfolded hermitian-def*])
have *m0*: *tensor-P* (1_m *N*) *M0* ∗ *tensor-P* (1_m *N*) *M0* = *tensor-P* (1_m *N*) *M0*
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-mult*[*symmetric*])
unfolding *ps-P-d1 ps-P-d2* **using** *M0-dim M0-mult-M0* **by** *auto*
have *m1*: *tensor-P* (1_m *N*) *M1* ∗ *tensor-P* (1_m *N*) *M1* = *tensor-P* (1_m *N*) *M1*
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-mult*[*symmetric*])
unfolding *ps-P-d1 ps-P-d2* **using** *M1-dim M1-mult-M1* **by** *auto*

have *s*: *tensor-P* (1_m *N*) *M1* + *tensor-P* (1_m *N*) *M0* = 1_m *d*
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-add2*[*symmetric*])
unfolding *ps-P-d1 ps-P-d2*
by (*auto simp add*: *M1-dim M0-dim M1-add-M0 ps-P.tensor-mat-id*[*simplified ps-P-d1 ps-P-d2 ps-P-d*])
show *measurement d 2* (λ*n. if n* = *0 then tensor-P* (1_m *N*) *M0 else if n* = *1 then tensor-P* (1_m *N*) *M1 else undefined*)
unfolding *measurement-def* **apply** (*auto simp add*: *tensor-P-dim*) **apply** (*subst 2*)
apply (*simp add*: *ad0 ad1 m0 m1*)
apply (*subst assoc-add-mat*[*symmetric*, *of - d d*]) **using** *tensor-P-dim s* **by** *auto*
show *well-com D* **using** *well-com-D* **by** *auto*
qed

lemma *well-com-Grover*:
well-com Grover
unfolding *Grover-def* **apply** *auto*
using *well-com-hadamard-n well-com-if well-com-while* **by** *auto*

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* **using** *ket-pre-def* **by** *auto*

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

lemma *pre-dim*:
pre ∈ *carrier-mat N N*
using *pre-def ket-pre-def* **by** *auto*

lemma *norm-pre*:
inner-prod ket-pre ket-pre = *1*
unfolding *ket-pre-def scalar-prod-def*
using *sum-only-one-neq-0*[*of* {*0..*<*N*} *0* λ*i.* (*if i* = *0 then 1 else 0*) ∗ *cnj* (*if i* = *0 then 1 else 0*)] **by** *auto*

lemma *pre-trace*:
trace pre = *1*
unfolding *pre-def*
apply (*subst trace-outer-prod*[*of - N*])
subgoal unfolding *ket-pre-def* **by** *auto* **using** *norm-pre* **by** *auto*

lemma *positive-pre*:
positive pre

using *positive-same-outer-prod* **unfolding** *pre-def ket-pre-def* **by** *auto*

lemma *pre-le-one*:
pre $\leq_L$ *1*$_m$ *N*
unfolding *pre-def* **using** *outer-prod-le-one norm-pre ket-pre-def* **by** *auto*

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* $\wedge$ *f i*) *then 1 else 0*)

lemma *post-dim*:
post $\in$ *carrier-mat N N*
unfolding *post-def* **by** *auto*

lemma *hermitian-post*:
hermitian post
unfolding *hermitian-def post-def*
by (*auto simp add*: *adjoint-eval*)

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 $\in$ *carrier-vec 2* **unfolding** *ket-zero-def* **by** *auto*

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 $\in$ *carrier-vec 2* **unfolding** *ket-plus-def* **by** *auto*

lemma *ket-plus-eval* [*simp*]:
i < *2* $\Longrightarrow$ *ket-plus* $ *i* = *1* / *csqrt 2*
apply (*simp only*: *ket-plus-def*)
using *index-vec less-2-cases* **by** *force*

lemma *csqrt-2-sq* [*simp*]:
complex-of-real (*sqrt 2*) * *complex-of-real* (*sqrt 2*) = *2*
by (*smt* (*verit*) *of-real-add of-real-hom.hom-one of-real-power one-add-one power2-eq-square real-sqrt-pow2*)

```
lemma ket-plus-tensor-n:
  partial-state.tensor-vec [2, 2] {0} ket-plus ket-plus = Matrix.vec 4 (λk. 1 / 2)
  unfolding partial-state.tensor-vec-def state-sig.d-def
proof (rule eq-vecI, auto)
  fix i :: nat assume i: i < 4
  interpret st: partial-state [2, 2] {0} .
  have d1-eq: st.d1 = 2
    by (simp add: st.d1-def st.dims1-def nths-def)
  have st.encode1 i < st.d1
    by (simp add: st.d-def i)
  then have i1-lt: st.encode1 i < 2
    using d1-eq by auto
  have d2-eq: st.d2 = 2
    by (simp add: st.d2-def st.dims2-def nths-def)
  have st.encode2 i < st.d2
    by (simp add: st.d-def i)
  then have i2-lt: st.encode2 i < 2
    using d2-eq by auto
  show ket-plus $ st.encode1 i * ket-plus $ st.encode2 i * 2 = 1
    by (auto simp add: i1-lt i2-lt)
qed

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

lemma hadamard-on-zero:
  hadamard *v ket-zero = ket-plus
  unfolding hadamard-def ket-zero-def ket-plus-def mat-of-rows-list-def
  apply (rule eq-vecI, auto simp add: scalar-prod-def)
  subgoal for i
    apply (drule less-2-cases)
    apply (drule disjE, auto)
    by (subst sum-le-2, auto)+.

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 ⟹ H-k k ∈ carrier-mat (2^(Suc k)) (2^(Suc k))
proof (induct k)
  case 0
  then show ?case using hadamard-dim by auto
next
```

case (*Suc k*)
interpret *st*: *partial-state2 dims* {*0*..<(*Suc k*)} {*Suc k*}
apply *unfold-locales* **by** *auto*
have *Suc* (*Suc k*) ≤ *n* **using** *Suc* **by** *auto*
then have *nths dims* ({*0*..<*Suc* (*Suc k*)}) = *replicate* (*Suc* (*Suc k*)) *2* **using** *dims-nths-le-n* **by** *auto*
moreover have *prod-list* (*replicate l 2*) = *2*^*l* **for** *l* **by** *simp*
moreover have {*0*..<*Suc k*} ∪ {*Suc k*} = {*0*..<(*Suc* (*Suc k*))} **by** *auto*
ultimately have *plssk*: *prod-list* (*nths dims* ({*0*..<*Suc k*} ∪ {*Suc k*})) = *2*^(*Suc* (*Suc k*)) **by** *auto*
have *dim-col* (*H-k* (*Suc k*)) = *2*^(*Suc* (*Suc k*)) **using** *st.ptensor-mat-dim-col* **unfolding** *st.d0-def st.dims0-def st.vars0-def* **using** *plssk* **by** *auto*
moreover have *dim-row* (*H-k* (*Suc k*)) = *2*^(*Suc* (*Suc k*)) **using** *st.ptensor-mat-dim-row* **unfolding** *st.d0-def st.dims0-def st.vars0-def* **using** *plssk* **by** *auto*
ultimately show *?case* **by** *auto*
qed

lemma *exH-k-eq-H-k*:
k < *n* ⟹ *exH-k k* = *pmat-extension dims* {*0*..<(*Suc k*)} {(*Suc k*)..<*n*} (*H-k k*)
proof(*induct k*)
case *0*
have {(*Suc 0*)..<*n*} = *vars1* − {*0*..<(*Suc 0*)} **using** *vars1-def* **by** *fastforce*
then show *?case* **unfolding** *exH-k.simps* **using** *vars1-def* **by** *auto*
next
case (*Suc k*)
interpret *st*: *partial-state2 dims* {*0*..<*Suc k*} {(*Suc k*)..<*n*}
apply *unfold-locales* **by** *auto*
interpret *st1*: *partial-state2 dims* {*Suc k*} {(*Suc* (*Suc k*))..<*n*}
apply *unfold-locales* **by** *auto*
interpret *st2*: *partial-state2 dims* {*Suc k*} *vars1* − {*Suc k*}
apply *unfold-locales* **by** *auto*
interpret *st3*: *partial-state2 dims* {*0*..<*Suc k*} {*Suc* (*Suc k*)..<*n*}
apply *unfold-locales* **by** *auto*
interpret *st4*: *partial-state2 dims* {*0*..<*Suc* (*Suc k*)} {*Suc* (*Suc k*)..<*n*}
apply *unfold-locales* **by** *auto*

from *Suc* **have** *eq0*: *exH-k* (*Suc k*)
= (*st.pmat-extension* (*H-k k*)) ∗ (*st2.pmat-extension hadamard*) **by** *auto*
have *vars1* − {*0*..<*Suc k*} = {(*Suc k*)..<*n*} **using** *vars1-def* **by** *auto*

then have *eql1*: *st.pmat-extension* (*H-k k*) = *st.ptensor-mat* (*H-k k*) (1_m *st.d2*)
using *st.pmat-extension-def* **by** *auto*

from *dims-nths-one-lt-n*[*OF Suc*(*2*)] **have** *st1d1*: *st1.d1* = *2* **unfolding** *st1.d1-def st1.dims1-def* **by** *fastforce*
have {*Suc k*} ∪ {*Suc* (*Suc k*)..<*n*} = {*Suc k*..<*n*} **using** *Suc* **by** *auto*
then have *st1.d0* = *st.d2* **unfolding** *st1.d0-def st1.dims0-def st1.vars0-def st.d2-def st.dims2-def* **by** *fastforce*
then have *eql2*: *st1.ptensor-mat* (1_m *2*) (1_m *st1.d2*) = 1_m *st.d2*

using st1.ptensor-mat-id st1d1 **by** auto
have eql3: st.ptensor-mat (H-k k) (1_m st.d2) = st.ptensor-mat (H-k k) (st1.ptensor-mat (1_m 2) (1_m st1.d2))
apply (subst eql2[symmetric]) **by** auto

have eqr1: (st2.pmat-extension hadamard) = st2.ptensor-mat hadamard (1_m st2.d2) **using** st2.pmat-extension-def **by** auto
have splitset: {0..<Suc k} ∪ {Suc (Suc k)..<n} = vars1 − {Suc k} **unfolding** vars1-def **using** Suc(2) **by** auto

have Sksplit: {Suc k} ∪ {Suc (Suc k)..<n} = {Suc k..<n} **using** Suc(2) **by** auto
have Sksplit1: {0..<Suc k}∪{Suc k} = {0..<Suc (Suc k)} **by** auto
have st.ptensor-mat (H-k k) (st1.ptensor-mat (1_m 2) (1_m st1.d2))
= ptensor-mat dims ({0..<Suc k}∪{Suc k}) {Suc (Suc k)..<n} (ptensor-mat dims {0..<Suc k} {Suc k} (H-k k) (1_m 2)) (1_m st1.d2)
apply (subst ptensor-mat-assoc[symmetric, of {0..<Suc k} {Suc k} {Suc (Suc k)..<n} H-k k 1_m 2 1_m st1.d2, simplified Sksplit])
using Suc length-dims **by** auto
also have . . . = ptensor-mat dims ({0..<Suc k}∪{Suc k}) {Suc (Suc k)..<n} (ptensor-mat dims {Suc k} {0..<Suc k} (1_m 2) (H-k k)) (1_m st1.d2)
using ptensor-mat-comm[of {0..<Suc k} {Suc k}] **by** auto
also have . . . = ptensor-mat dims {Suc k} ({0..<Suc k} ∪ {Suc (Suc k)..<n}) (1_m 2)
(ptensor-mat dims {0..<Suc k} {Suc (Suc k)..<n} (H-k k) (1_m st1.d2))
apply (subst sup-commute)
apply (subst ptensor-mat-assoc[of {Suc k} {0..<Suc k} {Suc (Suc k)..<n} (1_m 2) H-k k 1_m st1.d2])
using Suc length-dims **by** auto
finally have eql4: st.pmat-extension (H-k k)
= st2.ptensor-mat (1_m 2) (st3.ptensor-mat (H-k k) (1_m st3.d2)) **using** eql1 eql3 splitset **by** auto

have st2.ptensor-mat (1_m 2) (st3.ptensor-mat (H-k k) (1_m st3.d2)) ∗ st2.ptensor-mat hadamard (1_m st2.d2)
= st2.ptensor-mat ((1_m 2)∗hadamard) ((st3.ptensor-mat (H-k k) (1_m st3.d2))∗(1_m st2.d2))
apply (rule st2.ptensor-mat-mult[symmetric, of 1_m 2 hadamard (st3.ptensor-mat (H-k k) (1_m st3.d2)) (1_m st2.d2)])
subgoal unfolding st2.d1-def st2.dims1-def
by (simp add: dims-nths-one-lt-n Suc(2))
subgoal unfolding st2.d1-def st2.dims1-def
apply (simp add: dims-nths-one-lt-n Suc(2)) **using** hadamard-dim **by** auto
subgoal unfolding st2.d2-def[unfolded st2.dims2-def]
using st3.ptensor-mat-dim-col[unfolded st3.d0-def st3.dims0-def st3.vars0-def, simplified splitset]
st3.ptensor-mat-dim-row[unfolded st3.d0-def st3.dims0-def st3.vars0-def, simplified splitset] **by** auto
by auto

also have ... = *st2.ptensor-mat* (*hadamard*) (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*))
unfolding *st2.d2-def*[*unfolded st2.dims2-def*]
using *hadamard-dim st3.ptensor-mat-dim-col*[*unfolded st3.d0-def st3.dims0-def st3.vars0-def*, *simplified splitset*]
st3.ptensor-mat-dim-row[*unfolded st3.d0-def st3.dims0-def st3.vars0-def*, *simplified splitset*] **by** *auto*
also have ... = *ptensor-mat dims* ({*0..<Suc k*}∪{*Suc k*}) {*Suc* (*Suc k*)..<*n*} (*ptensor-mat dims* {*Suc k*} {*0..<Suc k*} *hadamard* (*H-k k*)) (1_m *st3.d2*)
apply (*subst ptensor-mat-assoc*[*symmetric*, *of* {*Suc k*} {*0..<Suc k*} {*Suc* (*Suc k*)..<*n*} *hadamard H-k k* 1_m *st3.d2*, *simplified splitset*])
using *Suc length-dims* **by** *auto*
also have ... = *ptensor-mat dims* ({*0..<Suc k*}∪{*Suc k*}) {*Suc* (*Suc k*)..<*n*} (*H-k* (*Suc k*)) (1_m *st3.d2*)
using *ptensor-mat-comm*[*of* {*Suc k*}] *Sksplit1* **by** *auto*
also have ... = *ptensor-mat dims* ({*0..<Suc* (*Suc k*)}) {*Suc* (*Suc k*)..<*n*} (*H-k* (*Suc k*)) (1_m *st3.d2*) **using** *Sksplit1* **by** *auto*
also have ... = *pmat-extension dims* {*0..<Suc* (*Suc k*)} {*Suc* (*Suc k*)..<*n*} (*H-k* (*Suc k*))
unfolding *st4.pmat-extension-def* **by** *auto*
finally show *?case* **using** *eq0 eql4 eqr1* **by** *auto*
qed

lemma *mult-exH-k-left*:
assumes *Suc k* < *n*
shows *hadamard-on-i* (*Suc k*) ∗ *exH-k k* = *exH-k* (*Suc k*)
proof −
interpret *st*: *partial-state2 dims* {*0..<Suc k*} {(*Suc k*)..<*n*}
apply *unfold-locales* **by** *auto*
interpret *st1*: *partial-state2 dims* {*Suc k*} {(*Suc* (*Suc k*))..<*n*}
apply *unfold-locales* **by** *auto*
interpret *st2*: *partial-state2 dims* {*Suc k*} *vars1* − {*Suc k*}
apply *unfold-locales* **by** *auto*
interpret *st3*: *partial-state2 dims* {*0..<Suc k*} {*Suc* (*Suc k*)..<*n*}
apply *unfold-locales* **by** *auto*
interpret *st4*: *partial-state2 dims* {*0..<Suc* (*Suc k*)} {*Suc* (*Suc k*)..<*n*}
apply *unfold-locales* **by** *auto*

from *exH-k-eq-H-k assms* **have** *eq0*: *exH-k* (*Suc k*)
= (*st.pmat-extension* (*H-k k*)) ∗ (*st2.pmat-extension hadamard*) **by** *auto*
have *vars1* − {*0..<Suc k*} = {(*Suc k*)..<*n*} **using** *vars1-def* **by** *auto*

then have *eql1*: *st.pmat-extension* (*H-k k*) = *st.ptensor-mat* (*H-k k*) (1_m *st.d2*)
using *st.pmat-extension-def* **by** *auto*

from *dims-nths-one-lt-n*[*OF assms*] **have** *st1d1*: *st1.d1* = *2* **unfolding** *st1.d1-def st1.dims1-def* **by** *fastforce*
have {*Suc k*} ∪ {*Suc* (*Suc k*)..<*n*} = {*Suc k..<n*} **using** *assms* **by** *auto*
then have *st1.d0* = *st.d2* **unfolding** *st1.d0-def st1.dims0-def st1.vars0-def*

st.d2-def st.dims2-def **by** *fastforce*
then have *eql2*: *st1.ptensor-mat* (1_m *2*) (1_m *st1.d2*) = 1_m *st.d2*
using *st1.ptensor-mat-id st1d1* **by** *auto*
have *eql3*: *st.ptensor-mat* (*H-k k*) (1_m *st.d2*) = *st.ptensor-mat* (*H-k k*) (*st1.ptensor-mat* (1_m *2*) (1_m *st1.d2*))
apply (*subst eql2*[*symmetric*]) **by** *auto*

have *eqr1*: (*st2.pmat-extension hadamard*) = *st2.ptensor-mat hadamard* (1_m *st2.d2*) **using** *st2.pmat-extension-def* **by** *auto*
have *splitset*: {*0..<Suc k*} ∪ {*Suc* (*Suc k*)..<*n*} = *vars1* − {*Suc k*} **unfolding** *vars1-def* **using** *assms* **by** *auto*

have *Sksplit*: {*Suc k*} ∪ {*Suc* (*Suc k*)..<*n*} = {*Suc k*..<*n*} **using** *assms* **by** *auto*
have *Sksplit1*: {*0..<Suc k*}∪{*Suc k*} = {*0..<Suc* (*Suc k*)} **by** *auto*
have *st.ptensor-mat* (*H-k k*) (*st1.ptensor-mat* (1_m *2*) (1_m *st1.d2*))
= *ptensor-mat dims* ({*0..<Suc k*}∪{*Suc k*}) {*Suc* (*Suc k*)..<*n*} (*ptensor-mat dims* {*0..<Suc k*} {*Suc k*} (*H-k k*) (1_m *2*)) (1_m *st1.d2*)
apply (*subst ptensor-mat-assoc*[*symmetric*, *of* {*0..<Suc k*} {*Suc k*} {*Suc* (*Suc k*)..<*n*} *H-k k* 1_m *2* 1_m *st1.d2*, *simplified Sksplit*])
using *assms length-dims* **by** *auto*
also have ... = *ptensor-mat dims* ({*0..<Suc k*}∪{*Suc k*}) {*Suc* (*Suc k*)..<*n*} (*ptensor-mat dims* {*Suc k*} {*0..<Suc k*} (1_m *2*) (*H-k k*)) (1_m *st1.d2*)
using *ptensor-mat-comm*[*of* {*0..<Suc k*} {*Suc k*}] **by** *auto*
also have ... = *ptensor-mat dims* {*Suc k*} ({*0..<Suc k*} ∪ {*Suc* (*Suc k*)..<*n*})
(1_m *2*)
(*ptensor-mat dims* {*0..<Suc k*} {*Suc* (*Suc k*)..<*n*} (*H-k k*) (1_m *st1.d2*))
apply (*subst sup-commute*)
apply (*subst ptensor-mat-assoc*[*of* {*Suc k*} {*0..<Suc k*} {*Suc* (*Suc k*)..<*n*} (1_m *2*) *H-k k* 1_m *st1.d2*]) **using** *assms length-dims* **by** *auto*
finally have *st.pmat-extension* (*H-k k*)
= *st2.ptensor-mat* (1_m *2*) (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*)) **using** *eql1 eql3 splitset* **by** *auto*
moreover have *st.pmat-extension* (*H-k k*) = *exH-k k* **using** *exH-k-eq-H-k assms* **by** *auto*
ultimately have *eql4*: *exH-k k* = *st2.ptensor-mat* (1_m *2*) (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*)) **by** *auto*

have *st2.ptensor-mat hadamard* (1_m *st2.d2*) ∗ *st2.ptensor-mat* (1_m *2*) (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*))
= *st2.ptensor-mat* (*hadamard*∗(1_m *2*)) ((1_m *st2.d2*)∗ (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*)))
apply (*rule st2.ptensor-mat-mult*[*symmetric*, *of hadamard* 1_m *2* (1_m *st2.d2*) (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*))])
subgoal unfolding *st2.d1-def st2.dims1-def* **apply** (*simp add*: *dims-nths-one-lt-n assms*) **using** *hadamard-dim* **by** *auto*
subgoal unfolding *st2.d1-def st2.dims1-def* **by** (*simp add*: *dims-nths-one-lt-n assms*)
subgoal by *auto*

subgoal unfolding *st2.d2-def*[*unfolded st2.dims2-def*] **using** *st3.ptensor-mat-dim-col*[*unfolded st3.d0-def st3.dims0-def st3.vars0-def*, *simplified splitset*]
st3.ptensor-mat-dim-row[*unfolded st3.d0-def st3.dims0-def st3.vars0-def*, *simplified splitset*] **by** *auto*
done
also have ... = *st2.ptensor-mat* (*hadamard*) (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*))
unfolding *st2.d2-def*[*unfolded st2.dims2-def*]
using *hadamard-dim st3.ptensor-mat-dim-col*[*unfolded st3.d0-def st3.dims0-def st3.vars0-def*, *simplified splitset*]
st3.ptensor-mat-dim-row[*unfolded st3.d0-def st3.dims0-def st3.vars0-def*, *simplified splitset*] **by** *auto*
also have ... = *ptensor-mat dims* ({*0..<Suc k*}∪{*Suc k*}) {*Suc* (*Suc k*)..<*n*} (*ptensor-mat dims* {*Suc k*} {*0..<Suc k*} *hadamard* (*H-k k*)) (1_m *st3.d2*)
apply (*subst ptensor-mat-assoc*[*symmetric*, *of* {*Suc k*} {*0..<Suc k*} {*Suc* (*Suc k*)..<*n*} *hadamard H-k k* 1_m *st3.d2*, *simplified splitset*])
using *assms length-dims* **by** *auto*
also have ... = *ptensor-mat dims* ({*0..<Suc k*}∪{*Suc k*}) {*Suc* (*Suc k*)..<*n*} (*H-k* (*Suc k*)) (1_m *st3.d2*)
using *ptensor-mat-comm*[*of* {*Suc k*}] *Sksplit1* **by** *auto*
also have ... = *ptensor-mat dims* ({*0..<Suc* (*Suc k*)}) {*Suc* (*Suc k*)..<*n*} (*H-k* (*Suc k*)) (1_m *st3.d2*) **using** *Sksplit1* **by** *auto*
also have ... = *pmat-extension dims* {*0..<Suc* (*Suc k*)} {*Suc* (*Suc k*)..<*n*} (*H-k* (*Suc k*))
unfolding *st4.pmat-extension-def* **by** *auto*
also have ... = *exH-k* (*Suc k*) **using** *exH-k-eq-H-k*[*of Suc k*] *assms* **by** *auto*
finally have *st2.ptensor-mat hadamard* (1_m *st2.d2*) ∗ *st2.ptensor-mat* (1_m *2*) (*st3.ptensor-mat* (*H-k k*) (1_m *st3.d2*))
=*exH-k* (*Suc k*).
then show *?thesis* **unfolding** *hadamard-on-i-def*
using *eql4 eqr1* **by** *auto*
qed

lemma *exH-eq-H*:
exH-k (*n* − *1*) = *H-k* (*n* − *1*)
proof −
have ∃ *m*. *n* = *Suc* (*Suc m*) **using** *n* **by** *presburger*
then obtain *m* **where** *m*: *n* = *Suc* (*Suc m*) **using** *n* **by** *auto*
then have *exH-k m* = *pmat-extension dims* {*0..<*(*Suc m*)} {(*Suc m*)..<*n*} (*H-k m*) **using** *exH-k-eq-H-k* **by** *auto*
then have *exH-k* (*Suc m*) = *pmat-extension dims* {*0..<*(*Suc m*)} {(*Suc m*)..<*n*} (*H-k m*)
∗ (*pmat-extension dims* {*Suc m*} (*vars1* − {*Suc m*}) *hadamard*) **by** *auto*
moreover have {(*Suc m*)..<*n*} = {*Suc m*} **using** *m* **by** *auto*
moreover have *vars1* − {*Suc m*} = {*0..<Suc m*} **unfolding** *vars1-def* **using** *m* **by** *auto*
ultimately have *eqSm*: *exH-k* (*Suc m*) = *pmat-extension dims* {*0..<*(*Suc m*)} {*Suc m*} (*H-k m*)

∗ (*pmat-extension dims* {*Suc m*} {*0..<Suc m*} *hadamard*)
by *auto*

interpret *stm1*: *partial-state2 dims* {*Suc m*} {*0..<Suc m*}
apply *unfold-locales* **by** *auto*
interpret *stm2*: *partial-state2 dims* {*0..<Suc m*} {*Suc m*}
apply *unfold-locales* **by** *auto*
have *nths dims* {*0..<Suc m*} = *replicate* (*Suc m*) *2* **using** *dims-nths-le-n m* **by** *auto*
then have *stm2d1*: *stm2.d1* = $2^{(Suc\ m)}$ **unfolding** *stm2.d1-def stm2.dims1-def* **by** *auto*
have *stm2d2*: *stm2.d2* = *2* **unfolding** *stm2.d2-def stm2.dims2-def* **using** *dims-nths-one-lt-n m* **by** *auto*

have *m* < *n* **using** *m* **by** *auto*
then have *H-k m* ∈ *carrier-mat* ($2^{(Suc\ m)}$) ($2^{(Suc\ m)}$) **using** *H-k-dim* **by** *auto*
then have *Hkm1*: (*H-k m*) ∗ (1_m *stm2.d1*) = (*H-k m*) **unfolding** *stm2d1* **by** *auto*

have *eqd12*: *stm1.d2* = *stm2.d1* **unfolding** *stm1.d2-def stm1.dims2-def stm2.d1-def stm2.dims1-def* **by** *auto*
have *pmat-extension dims* {*Suc m*} {*0..<Suc m*} *hadamard* = *stm1.ptensor-mat hadamard* (1_m *stm1.d2*) **using** *stm1.pmat-extension-def* **by** *auto*
also have ... = *stm2.ptensor-mat* (1_m *stm2.d1*) *hadamard* **using** *ptensor-mat-comm eqd12* **by** *auto*
finally have *eqr*: (*pmat-extension dims* {*Suc m*} {*0..<Suc m*} *hadamard*) = *stm2.ptensor-mat* (1_m *stm2.d1*) *hadamard*.
then have *exH-k* (*Suc m*) = *stm2.ptensor-mat* (*H-k m*) (1_m *stm2.d2*) ∗ *stm2.ptensor-mat* (1_m *stm2.d1*) *hadamard*
using *eqSm* **unfolding** *stm2.pmat-extension-def* **by** *auto*
also have ... = *stm2.ptensor-mat* ((*H-k m*) ∗ (1_m *stm2.d1*)) (1_m *stm2.d2* ∗ *hadamard*)
apply (*rule stm2.ptensor-mat-mult*[*symmetric*, *of H-k m* 1_m *stm2.d1* 1_m *stm2.d2 hadamard*])
unfolding *stm2d1 stm2d2* **using** *H-k-dim m hadamard-dim* **by** *auto*
also have ... = *stm2.ptensor-mat* (*H-k m*) (*hadamard*) **using** *H-k-dim hadamard-dim stm2d1 stm2d2 Hkm1* **by** *auto*
also have ... = *H-k* (*Suc m*) **unfolding** *stm2.ptensor-mat-def H-k.simps* **by** *auto*
finally have *exH-k* (*Suc m*) = *H-k* (*Suc m*) **by** *auto*
moreover have *Suc m* = *n* − *1* **using** *m* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

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* $k \in$ *carrier-vec* (*2*^(*Suc k*))
proof (*cases k*)
case *0*
show *?thesis* **using** *ket-zero-dim 0* **by** *auto*
next
case (*Suc k*)
interpret *st*: *partial-state2 dims* {*0*..<(*Suc k*)} {*Suc k*}
apply *unfold-locales* **by** *auto*
have *Suc* (*Suc k*) $\leq n$ **using** *assms Suc* **by** *auto*
then have *nths dims* ({*0*..<*Suc* (*Suc k*)}) = *replicate* (*Suc* (*Suc k*)) *2* **using** *dims-nths-le-n* **by** *auto*
moreover have *prod-list* (*replicate l 2*) = *2*^*l* **for** *l* **by** *simp*
moreover have {*0*..<*Suc k*} ∪ {*Suc k*} = {*0*..<(*Suc* (*Suc k*))} **by** *auto*
ultimately have *plssk*: *prod-list* (*nths dims* ({*0*..<*Suc k*} ∪ {*Suc k*})) = *2*^(*Suc* (*Suc k*)) **by** *auto*
show *?thesis* **apply** (*rule carrier-vecI*) **unfolding** *ket-zero-k.simps Suc*
using *st.ptensor-vec-dim*[*of ket-zero-k k ket-zero*] *plssk* **unfolding** *st.d0-def st.dims0-def st.vars0-def* **by** *auto*
qed

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* $k \in$ *carrier-vec* (*2*^(*Suc k*))
proof (*cases k*)
case *0*
show *?thesis* **using** *ket-plus-dim 0* **by** *auto*
next
case (*Suc k*)
interpret *st*: *partial-state2 dims* {*0*..<(*Suc k*)} {*Suc k*}
apply *unfold-locales* **by** *auto*
have *Suc* (*Suc k*) $\leq n$ **using** *assms Suc* **by** *auto*
then have *nths dims* ({*0*..<*Suc* (*Suc k*)}) = *replicate* (*Suc* (*Suc k*)) *2* **using** *dims-nths-le-n* **by** *auto*
moreover have *prod-list* (*replicate l 2*) = *2*^*l* **for** *l* **by** *simp*
moreover have {*0*..<*Suc k*} ∪ {*Suc k*} = {*0*..<(*Suc* (*Suc k*))} **by** *auto*
ultimately have *plssk*: *prod-list* (*nths dims* ({*0*..<*Suc k*} ∪ {*Suc k*})) = *2*^(*Suc* (*Suc k*)) **by** *auto*
show *?thesis* **apply** (*rule carrier-vecI*) **unfolding** *ket-zero-k.simps Suc*
using *st.ptensor-vec-dim plssk* **unfolding** *st.d0-def st.dims0-def st.vars0-def* **by** *auto*
qed

lemma *H-k-ket-zero-k*:
$k < n \Longrightarrow (H\text{-}k\ k) *_v (ket\text{-}zero\text{-}k\ k) = (ket\text{-}plus\text{-}k\ k)$
proof (*induct k*)
case *0*
show *?case* **using** *hadamard-on-zero* **unfolding** *H-k.simps ket-zero-k.simps ket-plus-k.simps* **by** *auto*
next
case (*Suc k*)
then have *k*: $k < n$ **by** *auto*
interpret *st*: *partial-state2 dims* {*0..<(Suc k)*} {*Suc k*}
apply *unfold-locales* **by** *auto*
have *nths dims* {*0..<Suc k*} = *replicate* (*Suc k*) *2* **using** *dims-nths-le-n Suc* **by** *auto*
then have *std1*: *st.d1* = *2^(Suc k)* **unfolding** *st.d1-def st.dims1-def* **by** *auto*
have *std2*: *st.d2* = *2* **unfolding** *st.d2-def st.dims2-def* **using** *dims-nths-one-lt-n Suc* **by** *auto*
have *H-k* (*Suc k*) $*_v$ *ket-zero-k* (*Suc k*) = *st.ptensor-mat* (*H-k k*) *hadamard* $*_v$ *st.ptensor-vec* (*ket-zero-k k*) *ket-zero* **by** *auto*
also have . . . = *st.ptensor-vec* ((*H-k k*) $*_v$ (*ket-zero-k k*)) (*hadamard* $*_v$ *ket-zero*)

using *st.ptensor-mat-mult-vec*[*unfolded std1 std2*, *OF H-k-dim*[*OF k*] *ket-zero-k-dim*[*OF k*] *hadamard-dim ket-zero-dim*] **by** *auto*
also have . . . = *st.ptensor-vec* (*ket-plus-k k*) *ket-plus* **using** *Suc hadamard-on-zero* **by** *auto*
finally show *?case* **by** *auto*
qed

lemma *encode1-replicate-2*:
partial-state.encode1 (*replicate* (*Suc k*) *2*) {*0..<k*} *i* = *i mod* (*2* ^ *k*)
proof −
have *take-Suc*: *take k* (*replicate* (*Suc k*) *2*) = *replicate k 2*
apply (*subst take-replicate*) **by** *auto*
have *take-encode*: *take k* (*digit-encode* (*replicate* (*Suc k*) *2*) *i*) = *digit-encode* (*replicate k 2*) *i*
apply (*subst digit-encode-take*) **using** *take-Suc* **by** *metis*
show *?thesis*
unfolding *partial-state.encode1-def partial-state.dims1-def*
nths-upt-eq-take[*simplified lessThan-atLeast0*] *take-Suc take-encode*
digit-decode-encode prod-list-replicate **..**
qed

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 −
have *drop-Suc*: *drop k* (*replicate* (*Suc k*) *2*) = [*2*]

apply (*subst drop-replicate*) **by** *auto*
have *drop-encode*: *drop k* (*digit-encode* (*replicate* (*Suc k*) *2*) *i*) = *digit-encode* [*2*] (*i div* (*2* ^ *k*))
unfolding *digit-encode-drop drop-Suc take-replicate prod-list-replicate*
by (*metis lessI min.strict-order-iff*)
have *le2*: *i div 2* ^ *k* < *2*
using *assms* **by** (*auto simp add*: *less-mult-imp-div-less*)
have *prod-list-2*: *prod-list* [*2*] = *2* **by** *simp*
show *?thesis*
unfolding *partial-state.encode2-def partial-state.dims2-def nths-minus-upt-eq-drop*[*simplified lessThan-atLeast0*] *drop-Suc drop-encode digit-decode-encode prod-list-2*
using *le2* **by** *auto*
qed

lemma *ket-zero-k-decode*:
k < *n* ⟹ *ket-zero-k k* = *Matrix.vec* (*2*^(*Suc k*)) (λ*k. if k* = *0 then 1 else 0*)
proof (*induct k*)
case *0*
show *?case* **apply** (*rule eq-vecI*) **by** (*auto simp add*: *ket-zero-def*)
next
case (*Suc k*)
then have *k*: *k* < *n* **by** *auto*
have *kzkk*: *ket-zero-k k* = *Matrix.vec* (*2* ^ *Suc k*) (λ*k. if* (*k* = *0*) *then 1 else 0*) **using** *Suc*(*1*)[*OF k*] **by** *auto*

have *dSk*: *ket-zero-k* (*Suc k*) ∈ *carrier-vec* (*2*^(*Suc* (*Suc k*))) **using** *ket-zero-k-dim*[*OF Suc*(*2*)] **by** *auto*

interpret *st*: *partial-state replicate* (*Suc* (*Suc k*)) *2* {*0*..<*Suc k*}**.**
interpret *st2*: *partial-state2 dims* {*0*..<*Suc k*} {*Suc k*} **by** (*unfold-locales*, *auto*)

have *splitset*: ({*0*..<*Suc k*} ∪ {*Suc k*}) = {*0*..<*Suc* (*Suc k*)} **by** *auto*
then have *st2dims0*: *st2.dims0* = *replicate* (*Suc* (*Suc k*)) *2* **unfolding** *st2.dims0-def st2.vars0-def*
using *dims-nths-le-n*[*of Suc* (*Suc k*)] *Suc* **by** *auto*
have ⋀*x.* (*x* ∈ {*0*..<*Suc k*} ⟹ {*y* ∈ {*0*..<*Suc* (*Suc k*)}. *y* < *x*} = {*0*..<*x*}) **by** *auto*
then have *cardeq*: ⋀*x.* (*x* ∈ {*0*..<*Suc k*} ⟹ *card* {*y* ∈ {*0*..<*Suc* (*Suc k*)}. *y* < *x*} = *card* {*0*..<*x*}) **by** *auto*
have *setcong*: ⋀*g h I.* (⋀*x.* (*x* ∈ *I* ⟹ *g x* = *h x*)) ⟹ {*g x* | *x. x* ∈ *I*} = {*h x* | *x. x* ∈ *I*} **by** *metis*
have {*card* {*y* ∈ {*0*..<*Suc* (*Suc k*)}. *y* < *x*} |*x. x* ∈ {*0*..<*Suc k*}} = {*card* {*0*..<*x*} |*x. x* ∈ {*0*..<*Suc k*}}
using *setcong*[*OF cardeq*, *of* {*0*..<*Suc k*}] **by** *auto*
also have ... = {*0*..<*Suc k*} **by** *auto*
finally have *st2vars1'*: *st2.vars1'* = {*0*..<*Suc k*} **unfolding** *st2.vars1'-def st2.vars0-def splitset ind-in-set-def* **by** *fastforce*
have *st2pvsttv*: *st2.ptensor-vec* = *st.tensor-vec* **unfolding** *st2.ptensor-vec-def*

using *st2dims0 st2vars1'* **by** *auto*
have *st.encode1 0 = 0* **using** *encode1-replicate-2*[*of Suc k 0*] **by** *auto*
moreover have *st.encode2 0 = 0* **using** *encode2-replicate-2*[*of 0 Suc k*] **by** *auto*
moreover have *std*: *st.d = 2^(Suc (Suc k))* **unfolding** *st.d-def* **by** *auto*
ultimately have *kzkk0*: *ket-zero-k (Suc k) $ 0 = 1*
unfolding *ket-zero-k.simps st2pvsttv st.tensor-vec-def ket-zero-def* **using** *kzkk*
by *auto*

have *kzkki*: *ket-zero-k (Suc k) $ i = 0* **if** *ine0*: *i ≠ 0* **and** *ile*: *i < 2^(Suc (Suc k))* **for** *i*
proof (*cases i mod (2 ^ Suc k) ≠ 0*)
case *True*
then have *ket-zero-k k $ st.encode1 i = 0* **unfolding** *kzkk* **using** *encode1-replicate-2*[*of Suc k i*] *ile* **by** *auto*
then show *?thesis* **unfolding** *ket-zero-k.simps st2pvsttv st.tensor-vec-def ket-zero-def std* **using** *ile* **by** *auto*
next
case *False*
have *i div (2 ^ Suc k) ≠ 0 ∨ i mod (2 ^ Suc k) ≠ 0* **using** *ine0* **by** *fastforce*
then have *i div (2 ^ Suc k) ≠ 0* **using** *False* **by** *auto*
moreover have *i div (2 ^ Suc k) < 2* **using** *ile less-mult-imp-div-less* **by** *auto*
ultimately have *i div (2 ^ Suc k) = 1* **by** *auto*
then have *st.encode2 i = 1* **using** *encode2-replicate-2*[*of i Suc k*] *ile* **by** *auto*
then have *Matrix.vec 2 (λk. if k = 0 then 1 else 0) $ st.encode2 i = 0*
unfolding *kzkk* **by** *fastforce*
then show *?thesis* **unfolding** *ket-zero-k.simps st2pvsttv st.tensor-vec-def ket-zero-def std* **using** *ile* **by** *auto*
qed

show *?case* **apply** (*rule eq-vecI*)
subgoal for *i* **using** *kzkk0 kzkki* **by** *auto*
using *carrier-vecD*[*OF dSk*] **by** *auto*
qed

lemma *ket-plus-k-decode*:
k < n ⟹ ket-plus-k k = Matrix.vec (2^(Suc k)) (λl. 1 / csqrt (2^(Suc k)))
proof (*induct k*)
case *0*
then show *?case* **unfolding** *ket-plus-k.simps ket-plus-def* **by** *auto*
next
case (*Suc k*)
then have *kpkk*: *ket-plus-k k = Matrix.vec (2 ^ Suc k) (λl. 1 / csqrt (2 ^ Suc k))* **by** *auto*

have *dSk*: *ket-plus-k (Suc k) ∈ carrier-vec (2^(Suc (Suc k)))* **using** *ket-plus-k-dim*[*OF Suc(2)*] **by** *auto*

interpret *st*: *partial-state replicate (Suc (Suc k)) 2 {0..<Suc k}*.
interpret *st2*: *partial-state2 dims {0..<Suc k} {Suc k}* **by** (*unfold-locales*, *auto*)

have *splitset*: ({*0..<Suc k*} ∪ {*Suc k*}) = {*0..<Suc* (*Suc k*)} **by** *auto*
then have *st2dims0*: *st2.dims0* = *replicate* (*Suc* (*Suc k*)) *2* **unfolding** *st2.dims0-def st2.vars0-def*
using *dims-nths-le-n*[*of Suc* (*Suc k*)] *Suc* **by** *auto*
have ⋀*x*. (*x* ∈ {*0..<Suc k*} ⟹ {*y* ∈ {*0..<Suc* (*Suc k*)}. *y* < *x*} = {*0..<x*}) **by** *auto*
then have *cardeq*: ⋀*x*. (*x* ∈ {*0..<Suc k*} ⟹ *card* {*y* ∈ {*0..<Suc* (*Suc k*)}. *y* < *x*} = *card* {*0..<x*}) **by** *auto*
have *setcong*: ⋀*g h I*. (⋀*x*. (*x* ∈ *I* ⟹ *g x* = *h x*)) ⟹ {*g x* | *x*. *x* ∈ *I*} = {*h x* | *x*. *x* ∈ *I*} **by** *metis*
have {*card* {*y* ∈ {*0..<Suc* (*Suc k*)}. *y* < *x*} |*x*. *x* ∈ {*0..<Suc k*}} = {*card* {*0..<x*} |*x*. *x* ∈ {*0..<Suc k*}}
using *setcong*[*OF cardeq*, *of* {*0..<Suc k*}] **by** *auto*
also have ... = {*0..<Suc k*} **by** *auto*
finally have *st2vars1′*: *st2.vars1′* = {*0..<Suc k*} **unfolding** *st2.vars1′-def st2.vars0-def splitset ind-in-set-def* **by** *blast*
have *st2pvsttv*: *st2.ptensor-vec* = *st.tensor-vec* **unfolding** *st2.ptensor-vec-def* **using** *st2dims0 st2vars1′* **by** *auto*

have *csqrt* (*2* ^ (*Suc k*)) = *complex-of-real* (*sqrt* (*2* ^ (*Suc k*))) **by** *simp*
moreover have *complex-of-real* (*sqrt* (*2* ^ (*Suc k*))) ∗ *complex-of-real* (*sqrt 2*) = *complex-of-real* (*sqrt* (*2* ^ (*Suc* (*Suc k*))))
by (*metis of-real-mult power-Suc power-commutes real-sqrt-power*)
ultimately have *csqrt* (*2* ^ (*Suc k*)) ∗ *csqrt 2* = *csqrt* (*2* ^ (*Suc* (*Suc k*))) **by** *auto*
moreover have *1* / *csqrt* (*2* ^ *Suc k*) ∗ *1* / *csqrt 2* = *1* / (*csqrt* (*2* ^ (*Suc k*)) ∗ *csqrt 2*) **by** *simp*
ultimately have *csqrt2p* :*1* / *csqrt* (*2* ^ *Suc k*) ∗ *1* / *csqrt 2* = *1* / (*csqrt* (*2* ^ (*Suc* (*Suc k*)))) **by** *simp*

have *std*: *st.d* = *2*^(*Suc* (*Suc k*)) **unfolding** *st.d-def* **by** *auto*

have *nthsSSk2*: *nths* (*replicate* (*Suc* (*Suc k*)) *2*) {*0..<Suc k*} = *replicate* (*Suc k*) *2*
unfolding *nths-replicate*[*of Suc* (*Suc k*) *2* {*0..<Suc k*}]
by (*smt* (*verit*) *Collect-cong* ‹{*card* {*0..<x*} |*x*. *x* ∈ {*0..<Suc k*}} = {*0..<Suc k*}› *atLeastLessThan-iff card-atLeastLessThan diff-zero less-SucI*)
then have *std1*: *st.d1* = *2*^(*Suc k*) **unfolding** *st.d1-def st.dims1-def nthsSSk2* **by** *auto*
have {*i*. *i* < *Suc* (*Suc k*) ∧ *i* ∈ {*Suc k..*}} = {*Suc k*} **by** *auto*
then have *nths* (*replicate* (*Suc* (*Suc k*)) *2*) ({*Suc k..*}) = *replicate 1 2* **unfolding** *nths-replicate* **by** *auto*
moreover have (− {*0..<Suc k*}) = {*Suc k..*} **by** *auto*
ultimately have *nthsSSk2c*: *nths* (*replicate* (*Suc* (*Suc k*)) *2*) (− {*0..<Suc k*}) = *replicate 1 2* **by** *auto*
have *std2*: *st.d2* = *2* **unfolding** *st.d2-def st.dims2-def* **apply** (*subst nthsSSk2c*) **by** *auto*

have *st.encode1 i* < *st.d1* **if** *i* < *st.d* **for** *i* **using** *that st.encode1-lt*[*OF that*] **by** *auto*
then have *kpkki*: *ket-plus-k k* $ *st.encode1 i* = *1* / *csqrt* (*2*^(*Suc k*)) **if** *i* < *st.d* **for** *i* **unfolding** *kpkk std1* **using** *that* **by** *auto*
have *st.encode2 i* < *st.d2* **if** *i* < *st.d* **for** *i* **using** *that st.encode2-lt*[*OF that*] **by** *auto*
then have *kpi*: *ket-plus* $ *st.encode2 i* = *1* / *csqrt 2* **if** *i* < *st.d* **for** *i* **unfolding** *ket-plus-def std2* **using** *that* **by** *auto*
have *kzkki*: *ket-plus-k* (*Suc k*) $ *i* = *1* / (*csqrt* (*2* ^ (*Suc* (*Suc k*)))) **if** *i* < *st.d* **for** *i*
unfolding *ket-plus-k.simps st2pvsttv st.tensor-vec-def* **using** *csqrt2p kpkki kpi that* **by** *auto*
show *?case* **apply** (*rule eq-vecI*)
subgoal for *i* **using** *kzkki* **unfolding** *std* **by** *auto*
using *carrier-vecD*[*OF dSk*] **by** *auto*
qed

lemma *exH-k-mult-pre-is-psi*:
exH-k (*n* − *1*) $*_v$ *ket-pre* = *ψ*
proof −
have *exH-k* (*n* − *1*) = *H-k* (*n* − *1*) **using** *exH-eq-H* **by** *auto*
moreover have *ket-zero-k* (*n* − *1*) = *ket-pre* **using** *ket-zero-k-decode*[*of n* − *1*] *ket-pre-def N-def n* **by** *auto*
moreover have *ket-plus-k* (*n* − *1*) = *ψ* **using** *ket-plus-k-decode*[*of n* − *1*] *ψ-def N-def n* **by** *auto*
moreover have *H-k* (*n* − *1*) $*_v$ *ket-zero-k* (*n* − *1*) = *ket-plus-k* (*n* − *1*) **using** *H-k-ket-zero-k n* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

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

lemma *ket-k-dim*:
ket-k k ∈ *carrier-vec K*
unfolding *ket-k-def* **by** *auto*

lemma *mat-incr-mult-ket-k*:
k < *K* ⟹ (*mat-incr K*) $*_v$ (*ket-k k*) = (*ket-k* ((*k* + *1*) *mod K*))
apply (*rule eq-vecI*)
unfolding *mat-incr-def ket-k-def*
apply (*simp add*: *scalar-prod-def*)
apply (*case-tac k* = *K* − *1*)
subgoal for *i* **apply** *auto* **by** (*simp add*: *sum-only-one-neq-0*[*of - K* − *1*])
subgoal for *i* **apply** *auto* **by** (*simp add*: *sum-only-one-neq-0*[*of - i* − *1*])
by *auto*

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

lemma *proj-k-dim*:
proj-k k ∈ *carrier-mat K K*
unfolding *proj-k-def* **using** *ket-k-dim* **by** *auto*

lemma *norm-ket-k-lt-K*:
$k < K \Longrightarrow$ *inner-prod* (*ket-k k*) (*ket-k k*) = *1*
unfolding *ket-k-def* **apply** (*simp add*: *scalar-prod-def*)
using *sum-only-one-neq-0*[*of* {*0..<K*} *k* λ*i*. (*if i* = *k then 1 else 0*) ∗ *cnj* (*if i* = *k then 1 else 0*)] **by** *auto*

lemma *norm-ket-k-ge-K*:
$k \geq K \Longrightarrow$ *inner-prod* (*ket-k k*) (*ket-k k*) = *0*
unfolding *ket-k-def* **by** (*simp add*: *scalar-prod-def*)

lemma *norm-ket-k*:
inner-prod (*ket-k k*) (*ket-k k*) ≤ *1*
apply (*case-tac k* < *K*)
using *norm-ket-k-lt-K norm-ket-k-ge-K* **by** (*auto simp*: *less-eq-complex-def*)

lemma *proj-k-mat*:
assumes *k* < *K*
shows *proj-k k* = *mat K K* (λ(*i*, *j*). *if* (*i* = *j* ∧ *i* = *k*) *then 1 else 0*)
apply (*rule eq-matI*)
apply (*simp add*: *proj-k-def ket-k-def index-outer-prod*)
using *proj-k-dim* **by** *auto*

lemma *positive-proj-k*:
positive (*proj-k k*)
using *positive-same-outer-prod* **unfolding** *proj-k-def ket-k-def* **by** *auto*

lemma *proj-k-le-one*:
(*proj-k k*) $\leq_L$ 1_m *K*
unfolding *proj-k-def* **using** *outer-prod-le-one norm-ket-k ket-k-def* **by** *auto*

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

lemma *proj-psi-dim*:
proj-psi ∈ *carrier-mat N N*
unfolding *proj-psi-def* ψ*-def* **by** *auto*

lemma *norm-psi*:
inner-prod ψ ψ = *1*
apply (*simp add*: ψ*-eval scalar-prod-def*)
by (*metis norm-of-nat norm-of-real of-real-mult of-real-of-nat-eq real-sqrt-mult-self*)

lemma *proj-psi-mat*:
proj-psi = *mat N N* (λ*k*. *1* / *N*)

unfolding *proj-psi-def*
apply (*rule eq-matI*, *simp-all*)
apply (*simp add*: ψ-*def index-outer-prod*)
apply (*smt* (*verit*) *of-nat-less-0-iff of-real-of-nat-eq of-real-power power2-eq-square real-sqrt-pow2*)
by (*auto simp add*: *carrier-matD*[*OF outer-prod-dim*[*OF* ψ-*dim*(*1*) ψ-*dim*(*1*)]])

lemma *hermitian-proj-psi*:
hermitian proj-psi
unfolding *hermitian-def proj-psi-mat* **apply** (*rule eq-matI*)
by (*auto simp add*: *adjoint-eval*)

lemma *hermitian-exproj-psi*:
hermitian (*tensor-P proj-psi* (1_m *K*))
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-hermitian*)
using *proj-psi-dim ps-P-d1 ps-P-d2 hermitian-proj-psi hermitian-one* **by** *auto*

lemma *proj-psi-is-projection*:
proj-psi ∗ *proj-psi* = *proj-psi*
proof −
have *proj-psi* ∗ *proj-psi* = *inner-prod* ψ ψ $\cdot_m$ *proj-psi*
unfolding *proj-psi-def*
apply (*subst outer-prod-mult-outer-prod*) **using** ψ-*def* **by** *auto*
also have . . . = *proj-psi*
using ψ-*inner* **by** *auto*
finally show *?thesis***.**
qed

lemma *proj-psi-trace*:
trace (*proj-psi*) = *1*
unfolding *proj-psi-def*
apply (*subst trace-outer-prod*[*of - N*])
subgoal unfolding ψ-*def* **by** *auto* **using** *norm-psi* **by** *auto*

lemma *positive-proj-psi*:
positive (*proj-psi*)
using *positive-same-outer-prod* **unfolding** *proj-psi-def* ψ-*def* **by** *auto*

lemma *proj-psi-le-one*:
(*proj-psi*) $\leq_L$ 1_m *N*
unfolding *proj-psi-def* **using** *outer-prod-le-one norm-psi* ψ-*def* **by** *auto*

lemma *hermitian-hadamard-on-k*:
assumes $k < n$
shows *hermitian* (*hadamard-on-i k*)
proof −
interpret *st2*: *partial-state2 dims* {*k*} (*vars1* − {*k*})
apply *unfold-locales* **by** *auto*

have *st2d1*: *st2.dims1* = [*2*] **unfolding** *st2.dims1-def dims-def*
using *assms dims-nths-one-lt-n local.dims-def st2.dims1-def* **by** *auto*
show *hermitian* (*hadamard-on-i k*) **unfolding** *hadamard-on-i-def st2.pmat-extension-def st2.ptensor-mat-def*
apply (*rule partial-state.tensor-mat-hermitian*)
subgoal unfolding *partial-state.d1-def partial-state.dims1-def st2.nths-vars1′ hadamard-def* **by** (*simp add*: *st2d1*)
subgoal unfolding *partial-state.d2-def partial-state.dims2-def st2.nths-vars2′ st2.d2-def* **by** *auto*
subgoal unfolding *hermitian-def hadamard-def* **apply** (*rule eq-matI*) **by** (*auto simp add*: *adjoint-dim adjoint-eval*)
using *hermitian-one* **by** *auto*
qed

lemma *hermitian-H-k*:
$k < n \Longrightarrow$ *hermitian* (*H-k k*)
proof (*induct k*)
case *0*
show *?case* **unfolding** *H-k.simps hermitian-def hadamard-def* **apply** (*rule eq-matI*) **by** (*auto simp add*: *adjoint-dim adjoint-eval*)
next
case (*Suc k*)
interpret *st2*: *partial-state2 dims* {*0..<Suc k*} {*Suc k*}
apply *unfold-locales* **by** *auto*
have *st2d1*: *prod-list st2.dims1* = (*2*^(*Suc k*)) **unfolding** *st2.dims1-def dims-def* **using** *Suc*(*2*)
using *dims-nths-le-n local.dims-def st2.dims1-def* **by** *auto*
have *st2d2*: *st2.dims2* = [*2*] **unfolding** *st2.dims2-def dims-def* **using** *Suc*(*2*)
using *dims-nths-one-lt-n local.dims-def st2.dims2-def* **by** *auto*
show *?case* **unfolding** *H-k.simps st2.ptensor-mat-def*
apply (*rule partial-state.tensor-mat-hermitian*)
subgoal unfolding *partial-state.d1-def partial-state.dims1-def st2.nths-vars1′* **using** *st2d1 H-k-dim Suc* **by** *auto*
subgoal unfolding *partial-state.d2-def partial-state.dims2-def st2.nths-vars2′ st2.d2-def* **using** *st2d2* **by** (*simp add*: *hadamard-def*)
subgoal using *Suc* **by** *auto*
using *hermitian-hadamard* **by** *auto*
qed

lemma *unitary-H-k*:
$k < n \Longrightarrow$ *unitary* (*H-k k*)
proof (*induct k*)
case *0*
show *?case* **using** *unitary-hadamard* **by** *auto*
next
case (*Suc k*)
then have *k*: $k < n$ **by** *auto*
interpret *st2*: *partial-state2 dims* {*0..<Suc k*} {*Suc k*} **by** (*unfold-locales*, *auto*)

```
  have st2d1: prod-list st2.dims1 = (2^(Suc k)) unfolding st2.dims1-def dims-def
using Suc(2)
    using dims-nths-le-n local.dims-def st2.dims1-def by auto
  have st2d2: st2.dims2 = [2] unfolding st2.dims2-def dims-def using Suc(2)
    using dims-nths-one-lt-n local.dims-def st2.dims2-def by auto
  show ?case unfolding H-k.simps st2.ptensor-mat-def
   apply (rule partial-state.tensor-mat-unitary[of H-k k st2.dims0 st2.vars1' hadamard]
)
   unfolding partial-state.d1-def partial-state.dims1-def st2.nths-vars1' partial-state.d2-def
partial-state.dims2-def
      st2.nths-vars2'
       apply (auto simp add: st2d1 st2d2 )
    subgoal using H-k-dim[OF k] by auto
    subgoal using hadamard-dim by auto
    subgoal using Suc by auto
    using unitary-hadamard by auto
qed

lemma exH-k-dim:
  shows k < n ⟹ exH-k k ∈ carrier-mat N N
  apply (induct k)
  using hadamard-on-i-dim by auto

lemma exH-n-dim:
  shows exH-k (n − 1) ∈ carrier-mat N N
  using exH-k-dim n by auto

lemma unitary-exH-k:
  shows k < n ⟹ unitary (exH-k k)
proof (induct k)
  case 0
   then show ?case unfolding exH-k.simps using unitary-hadamard-on-i 0 by
auto
next
  case (Suc k)
  show ?case unfolding exH-k.simps apply (subst unitary-times-unitary[of - N])
    subgoal using exH-k-dim Suc by auto
    subgoal using hadamard-on-i-dim Suc by auto
    subgoal using Suc by auto
    using unitary-hadamard-on-i Suc by auto
qed

lemma hermitian-exH-n:
  hermitian (exH-k (n − 1))
  using hermitian-H-k exH-eq-H n by auto

lemma exH-k-mult-psi-is-pre:
  exH-k (n − 1) *v ψ = ket-pre
proof −
```

```
  let ?H = exH-k (n − 1)
  have hermitian ?H using hermitian-H-k exH-eq-H n by auto
  then have eqad: adjoint ?H = ?H unfolding hermitian-def by auto
  have d: ?H ∈ carrier-mat N N using exH-k-dim n by auto
  have unitary ?H using unitary-exH-k n by auto
  then have id: ?H ∗ ?H = 1_m N
    unfolding unitary-def inverts-mat-def
    using d apply (subst (2) eqad[symmetric]) by auto
  have ?H ∗_v ψ = ?H ∗_v (?H ∗_v ket-pre)
    using exH-k-mult-pre-is-psi by auto
  also have ... = (?H ∗ ?H) ∗_v ket-pre
    using d ket-pre-def by auto
  also have ... = ket-pre
    using id ket-pre-def by auto
  finally show ?thesis by auto
qed

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

lemma unitary-exexH-k:
  k < n ⟹ unitary (exexH-k k)
  unfolding exexH-k.simps ps2-P.ptensor-mat-def
  apply (subst partial-state.tensor-mat-unitary)
  subgoal using exH-k-dim unfolding partial-state.d1-def partial-state.dims1-def
ps2-P.nths-vars1' ps2-P.dims1-def dims-vars1 N-def by auto
  subgoal unfolding partial-state.d2-def partial-state.dims2-def ps2-P.nths-vars2'
ps2-P.dims2-def dims-vars2 by auto
  using unitary-exH-k unitary-one by auto

lemma exexH-k-dim:
  k < n ⟹ exexH-k k ∈ carrier-mat d d
  unfolding exexH-k.simps using ps2-P.ptensor-mat-carrier ps2-P-d0 by auto

lemma hoare-seq-utrans:
  fixes P :: complex mat
  assumes unitary U1 and unitary U2 and is-quantum-predicate P
    and dU1: U1 ∈ carrier-mat d d and dU2: U2 ∈ carrier-mat d d
  shows
   ⊢_p
   {adjoint (U2 ∗ U1) ∗ P ∗ (U2 ∗ U1)}
   Utrans U1;; Utrans U2
   {P}
proof −
  have hp0: ⊢_p {adjoint (U2) ∗ P ∗ (U2)} Utrans U2 {P}
    using assms hoare-partial.intros by auto
  have qp: is-quantum-predicate (adjoint (U2) ∗ P ∗ (U2))
    using qp-close-under-unitary-operator assms by auto
  then have hp1: ⊢_p {adjoint U1 ∗ (adjoint (U2) ∗ P ∗ (U2)) ∗ U1} Utrans U1
```

{*adjoint* (*U2*) ∗ *P* ∗ (*U2*)}
using *hoare-partial.intros* **by** *auto*
have *dP*: *P* ∈ *carrier-mat d d* **using** *assms is-quantum-predicate-def* **by** *auto*
have *eq*: *adjoint U1* ∗ (*adjoint U2* ∗ *P* ∗ *U2*) ∗ *U1* = *adjoint* (*U2* ∗ *U1*) ∗ *P* ∗ (*U2* ∗ *U1*)
using *dU1 dU2 dP* **by** (*mat-assoc d*)
with *hp1* **have** *hp2*: $\vdash_p$ {*adjoint* (*U2* ∗ *U1*) ∗ *P* ∗ (*U2* ∗ *U1*)} *Utrans U1* {*adjoint* (*U2*) ∗ *P* ∗ (*U2*)} **by** *auto*

have *is-quantum-predicate* (*adjoint U1* ∗ (*adjoint U2* ∗ *P* ∗ *U2*) ∗ *U1*) **using** *qp qp-close-under-unitary-operator assms* **by** *auto*
then have *is-quantum-predicate* (*adjoint* (*U2* ∗ *U1*) ∗ *P* ∗ (*U2* ∗ *U1*)) **using** *eq* **by** *auto*
then show *?thesis* **using** *hoare-partial.intros*(*3*)[*OF* - *qp assms*(*3*)] *hp0 hp2* **by** *auto*
qed

lemma *qp-close-after-exexH-k*:
fixes *P* :: *complex mat*
assumes *is-quantum-predicate P*
shows *k* < *n* ⟹ *is-quantum-predicate* (*adjoint* (*exexH-k k*) ∗ *P* ∗ *exexH-k k*)
apply (*subst qp-close-under-unitary-operator*)
subgoal using *exexH-k-dim* **by** *auto*
subgoal using *unitary-exexH-k* **by** *auto*
using *assms* **by** *auto*

lemma *hoare-hadamard-n*:
fixes *P* :: *complex mat*
shows *is-quantum-predicate P* ⟹ *k* < *n* ⟹
$\vdash_p$
{*adjoint* (*exexH-k k*) ∗ *P* ∗ *exexH-k k*}
hadamard-n (*Suc k*)
{*P*}
proof (*induct k arbitrary*: *P*)
case *0*
have *qp*: *is-quantum-predicate* (*adjoint* (*exexH-k 0*) ∗ *P* ∗ *exexH-k 0*)
using *qp-close-under-unitary-operator*[*OF* - *unitary-exhadamard-on-i*[*of 0*]] *tensor-P-dim 0* **by** *auto*
then have $\vdash_p$ {*adjoint* (*exexH-k 0*) ∗ *P* ∗ *exexH-k 0*} *SKIP* {*adjoint* (*exexH-k 0*) ∗ *P* ∗ *exexH-k 0*}
using *hoare-partial.intros*(*1*) **by** *auto*
moreover have $\vdash_p$ {*adjoint* (*exexH-k 0*) ∗ *P* ∗ *exexH-k 0*} *Utrans* (*tensor-P* (*hadamard-on-i 0*) (1_m *K*)) {*P*}
using *hoare-partial.intros*(*2*) *0* **by** *auto*
ultimately have $\vdash_p$ {*adjoint* (*exexH-k 0*) ∗ *P* ∗ *exexH-k 0*} *SKIP*;; *Utrans* (*tensor-P* (*hadamard-on-i 0*) (1_m *K*)) {*P*}
using *hoare-partial.intros*(*3*) *qp 0* **by** *auto*
then show *?case* **using** *qp* **by** *auto*
next

case (*Suc k*)
have *h1*: $\vdash_p$
{*adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *P* ∗ (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*))}
Utrans (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*))
{*P*}
using *hoare-partial.intros Suc* **by** *auto*
have *qp*: *is-quantum-predicate* (*adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *P* ∗ (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)))
apply (*subst qp-close-under-unitary-operator*)
subgoal using *ps2-P.ptensor-mat-carrier ps2-P-d0* **by** *auto*
subgoal unfolding *ps2-P.ptensor-mat-def* **apply** (*subst partial-state.tensor-mat-unitary*)
subgoal unfolding *partial-state.d1-def partial-state.dims1-def ps2-P.nths-vars1′ ps2-P.dims1-def d-vars1* **using** *hadamard-on-i-dim Suc* **by** *auto*
subgoal unfolding *partial-state.d2-def partial-state.dims2-def ps2-P.nths-vars2′ ps2-P.dims2-def* **using** *dims-vars2* **by** *auto*
using *unitary-hadamard-on-i unitary-one Suc* **by** *auto*
using *Suc* **by** *auto*
then have *h2*: $\vdash_p$
{*adjoint* (*exexH-k k*) ∗ (*adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *P* ∗ (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*))) ∗ *exexH-k k*}
hadamard-n (*Suc k*)
{*adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *P* ∗ (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*))}
using *Suc* **by** *auto*
have (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *exexH-k k*
= (*tensor-P* (*hadamard-on-i* (*Suc k*) ∗ (*exH-k k*)) (1_m *K* ∗ (1_m *K*)))
apply (*subst ps2-P.ptensor-mat-mult*)
subgoal using *hadamard-on-i-dim ps2-P-d1 Suc* **by** *auto*
subgoal using *exH-k-dim ps2-P-d1 Suc* **by** *auto*
using *ps2-P-d2* **by** *auto*
also have ... = *exexH-k* (*Suc k*) **using** *mult-exH-k-left Suc* **by** *auto*
finally have *eq1*: (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *exexH-k k* = *exexH-k* (*Suc k*)**.**
then have *eq2*: *adjoint* (*exexH-k k*) ∗ *adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) = *adjoint* (*exexH-k* (*Suc k*))
apply (*subst adjoint-mult*[*symmetric*, *of - d d - d*])
subgoal using *tensor-P-dim* **by** *auto*
using *exexH-k-dim Suc* **by** *auto*
have *dP*: *P* ∈ *carrier-mat d d* **using** *is-quantum-predicate-def Suc* **by** *auto*
moreover have *dH*: *exexH-k k* ∈ *carrier-mat d d* **using** *exexH-k-dim Suc* **by** *auto*
moreover have *dHi*: *tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*) ∈ *carrier-mat d d* **using** *tensor-P-dim* **by** *auto*
ultimately have *eq3*: *adjoint* (*exexH-k k*) ∗ (*adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *P* ∗ *tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *exexH-k k*
= (*adjoint* (*exexH-k k*) ∗ *adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*))) ∗ *P* ∗ (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*) ∗ *exexH-k k*)

by (*mat-assoc d*)
show *?case* **apply** (*subst hadamard-n.simps*)
apply (*subst hoare-partial.intros*(*3*)[*of - adjoint* (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*)) ∗ *P* ∗ (*tensor-P* (*hadamard-on-i* (*Suc k*)) (1_m *K*))]])
subgoal using *qp-close-after-exexH-k*[*of P Suc k*] *Suc* **by** *auto*
subgoal using *qp* **by** *auto*
subgoal using *Suc* **by** *auto*
subgoal using *h2*[*simplified eq3 eq1 eq2*] **by** *auto*
using *h1* **by** *auto*
qed

lemma *qp-pre*:
is-quantum-predicate (*tensor-P pre* (*proj-k 0*))
unfolding *is-quantum-predicate-def*
proof (*intro conjI*)
show *tensor-P pre* (*proj-k 0*) ∈ *carrier-mat d d* **using** *tensor-P-dim* **by** *auto*
interpret *st*: *partial-state dims vars1* **.**
have *d1*: *st.d1* = *N* **unfolding** *st.d1-def st.dims1-def* **using** *d-vars1* **by** *auto*
have *d2*: *st.d2* = *K* **unfolding** *st.d2-def st.dims2-def nths-uminus-vars1 dims-vars2* **by** *auto*
show *positive* (*tensor-P pre* (*proj-k 0*))
unfolding *ps2-P.ptensor-mat-def ps2-P-dims0 ps2-P-vars1'*
apply (*subst st.tensor-mat-positive*)
subgoal unfolding *pre-def* **using** *outer-prod-dim ket-pre-def d1* **by** *auto*
subgoal unfolding *proj-k-def* **using** *outer-prod-dim ket-k-def d2* **by** *auto*
subgoal using *positive-pre* **by** *auto*
using *positive-proj-k*[*of 0*] *K-gt-0* **by** *auto*
show *tensor-P pre* (*proj-k 0*) $\leq_L$ 1_m *d*
unfolding *ps2-P.ptensor-mat-def ps2-P-dims0 ps2-P-vars1'*
apply (*subst st.tensor-mat-le-one*)
subgoal using *pre-def ket-pre-def outer-prod-dim d1* **by** *auto*
subgoal using *proj-k-def K-gt-0 ket-k-def outer-prod-dim d2* **by** *auto*
using *d1 d2 K-gt-0 outer-prod-dim positive-pre positive-proj-k pre-le-one proj-k-le-one* **by** *auto*
qed

lemma *qp-init-post*:
is-quantum-predicate (*tensor-P proj-psi* (*proj-k 0*))
unfolding *is-quantum-predicate-def*
proof (*intro conjI*)
show *tensor-P proj-psi* (*proj-k 0*) ∈ *carrier-mat d d* **using** *tensor-P-dim* **by** *auto*
interpret *st*: *partial-state dims vars1* **.**
have *d1*: *st.d1* = *N* **unfolding** *st.d1-def st.dims1-def* **using** *d-vars1* **by** *auto*
have *d2*: *st.d2* = *K* **unfolding** *st.d2-def st.dims2-def nths-uminus-vars1 dims-vars2* **by** *auto*
show *positive* (*tensor-P proj-psi* (*proj-k 0*))
unfolding *ps2-P.ptensor-mat-def ps2-P-dims0 ps2-P-vars1'*
apply (*subst st.tensor-mat-positive*)
subgoal unfolding *proj-psi-def* **using** *outer-prod-dim ψ-def d1* **by** *auto*

subgoal unfolding *proj-k-def* **using** *outer-prod-dim ket-k-def d2* **by** *auto*
subgoal using *positive-proj-psi* **by** *auto*
using *positive-proj-k*[*of 0*] *K-gt-0* **by** *auto*
show *tensor-P proj-psi* (*proj-k 0*) $\leq_L$ *1*$_m$ *d*
unfolding *ps2-P.ptensor-mat-def ps2-P-dims0 ps2-P-vars1'*
apply (*subst st.tensor-mat-le-one*)
subgoal using *proj-psi-def outer-prod-dim d1* **by** *auto*
subgoal using *proj-k-def K-gt-0 ket-k-def outer-prod-dim d2* **by** *auto*
using *d1 d2 K-gt-0 outer-prod-dim positive-proj-psi positive-proj-k proj-psi-le-one proj-k-le-one* **by** *auto*
qed

lemma *tensor-P-adjoint-left-right*:
assumes *m1* ∈ *carrier-mat N N* **and** *m2* ∈ *carrier-mat K K* **and** *m3* ∈ *carrier-mat N N* **and** *m4* ∈ *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 −
have *eq1*: *adjoint* (*tensor-P m1 m2*) = *tensor-P* (*adjoint m1*) (*adjoint m2*)
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-adjoint*)
using *ps-P-d1 ps-P-d2 assms* **by** *auto*
have *eq2*: *adjoint* (*tensor-P m1 m2*) ∗ *tensor-P m3 m4* = *tensor-P* (*adjoint m1* ∗ *m3*) (*adjoint m2* ∗ *m4*)
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-mult*)
using *ps-P-d1 ps-P-d2 assms eq1* **unfolding** *ps2-P.ptensor-mat-def* **by** (*auto simp add*: *adjoint-dim*)
have *eq3*: *tensor-P* (*adjoint m1* ∗ *m3*) (*adjoint m2* ∗ *m4*) ∗ (*tensor-P m1 m2*) = *tensor-P* (*adjoint m1* ∗ *m3* ∗ *m1*) (*adjoint m2* ∗ *m4* ∗ *m2*)
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-mult*[*of adjoint m1* ∗ *m3*])
using *ps-P-d1 ps-P-d2 assms* **by** (*auto simp add*: *adjoint-dim*)
show *?thesis* **using** *eq1 eq2 eq3* **by** *auto*
qed

abbreviation *exH-n* **where**
exH-n ≡ *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 −
have *h*: $\vdash_p$
{*adjoint* (*exexH-k* (*n* − *1*)) ∗ (*tensor-P proj-psi* (*proj-k 0*)) ∗ (*exexH-k* (*n* − *1*))}
hadamard-n n

{*tensor-P proj-psi* (*proj-k 0*)}
using *hoare-hadamard-n*[*OF qp-init-post*, *of n* − *1*] *qp-init-post n* **by** *auto*
have *adjoint* (*exexH-k* (*n* − *1*)) ∗ *tensor-P proj-psi* (*proj-k 0*) ∗ *exexH-k* (*n* − *1*) =
tensor-P (*adjoint exH-n* ∗ *proj-psi* ∗ *exH-n*) (*adjoint* (1_m *K*) ∗ *proj-k 0* ∗ 1_m *K*)
unfolding *exexH-k.simps*
apply (*subst tensor-P-adjoint-left-right*)
using *exH-k-dim proj-psi-def ψ-def proj-k-def ket-k-def n* **by** (*auto*)
moreover have *adjoint exH-n* ∗ *proj-psi* ∗ *exH-n* = *pre*
unfolding *proj-psi-def pre-def*
apply (*subst outer-prod-left-right-mat*[*of - N - N - N - N*])
subgoal using *ψ-def* **by** *auto*
subgoal using *exH-k-dim n* **by** (*simp add*: *adjoint-dim*)
subgoal using *exH-k-dim n* **by** *simp*
apply (*subst* (*1 2*) *hermitian-exH-n*[*simplified hermitian-def*])
apply (*subst* (*1 2*) *exH-k-mult-psi-is-pre*)
by *auto*
moreover have *adjoint* (1_m *K*) ∗ (*proj-k 0*) ∗ (1_m *K*) = *proj-k 0*
apply (*subst adjoint-one*) **using** *proj-k-dim*[*of 0*] *K-gt-0* **by** *auto*
ultimately have *adjoint* (*exexH-k* (*n* − *1*)) ∗ *tensor-P proj-psi* (*proj-k 0*) ∗ *exexH-k* (*n* − *1*) = *tensor-P pre* (*proj-k 0*)
by *auto*
with *h* **show** *?thesis* **by** *auto*
qed

Hoare triples of while loop

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

lemma *positive-psi-l*:
k < *K* ⟹ *positive* (*proj-psi-l k*)
unfolding *proj-psi-l-def*
apply (*subst positive-same-outer-prod*)
using *psi-l-dim* **by** *auto*

lemma *hermitian-proj-psi-l*:
k < *K* ⟹ *hermitian* (*proj-psi-l k*)
using *positive-psi-l positive-is-hermitian* **by** *auto*

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*
unfolding *proj-psi-l-def* **using** *psi-l-def* **by** *auto*

definition *Q* :: *complex mat* **where**
Q = *matrix-sum d* (*λl. 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 −
have *inner-prod* (*psi-l l*) (*psi-l l*) = *1*
using *inner-psi-l* **by** *auto*
then show *?thesis* **using** *outer-prod-le-one psi-l-def proj-psi-l-def* **by** *auto*
qed

lemma *positive-proj-psi-l*:
shows *positive* (*proj-psi-l l*)
using *positive-same-outer-prod proj-psi-l-def psi-l-dim* **by** *auto*

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

lemma *hermitian-proj-fst-k*:
adjoint (*proj-fst-k k*) = *proj-fst-k k*
by (*auto simp add*: *proj-fst-k-def adjoint-eval*)

lemma *proj-fst-k-is-projection*:
proj-fst-k k ∗ *proj-fst-k k* = *proj-fst-k k*
by (*auto simp add*: *proj-fst-k-def scalar-prod-def sum-only-one-neq-0*)

lemma *positive-proj-fst-k*:
positive (*proj-fst-k k*)
proof −
have (*proj-fst-k k*) ∗ *adjoint* (*proj-fst-k k*) = (*proj-fst-k k*)
using *hermitian-proj-fst-k proj-fst-k-is-projection* **by** *auto*
then have ∃ *M*. *M* ∗ *adjoint M* = (*proj-fst-k k*) **by** *auto*
then show *?thesis* **apply** (*subst positive-if-decomp*) **using** *proj-fst-k-def* **by** *auto*
qed

lemma *proj-fst-k-le-one*:
proj-fst-k k $\leq_L$ *1*$_m$ *K*
proof −
define *M* **where** *M l* = *mat K K* (λ(*i*, *j*). *if* (*i* = *j* ∧ *i* ≥ *l*) *then* (*1*::*complex*) *else 0*) **for** *l*
have *eq*: *1*$_m$ *K* − *proj-fst-k k* = *M k* **unfolding** *M-def proj-fst-k-def*
apply (*rule eq-matI*) **by** *auto*
have *M k* ∗ *M k* = *M k* **unfolding** *M-def*
apply (*rule eq-matI*) **apply** (*simp add*: *scalar-prod-def*)
apply (*subst sum-only-one-neq-0*[*of - j*]) **by** *auto*
moreover have *adjoint* (*M k*) = *M k* **unfolding** *M-def*
apply (*rule eq-matI*) **by** (*auto simp add*: *adjoint-eval*)
ultimately have *M k* ∗ *adjoint* (*M k*) = *M k* **by** *auto*
then have ∃ *M*. *M* ∗ *adjoint M* = *1*$_m$ *K* − *proj-fst-k k* **using** *eq* **by** *auto*
then have *positive* (*1*$_m$ *K* − *proj-fst-k k*)
apply (*subst positive-if-decomp*) **using** *proj-fst-k-def* **by** *auto*

then show *?thesis* **unfolding** *lowner-le-def* **using** *proj-fst-k-def* **by** *auto*
qed

lemma *sum-proj-k*:
assumes $m \leq K$
shows *matrix-sum K* (λk. *proj-k k*) m = *proj-fst-k m*
proof −
have $m \leq K \Longrightarrow$ *matrix-sum K* (λk. *proj-k k*) m = *mat K K* ($\lambda(i, j)$. *if* ($i = j \wedge i < m$) *then 1 else 0*) **for** m
proof (*induct m*)
case *0*
then show *?case* **apply** *simp* **apply** (*rule eq-matI*) **by** *auto*
next
case (*Suc m*)
then have m: $m < K$ **by** *auto*
then have m': $m \leq K$ **by** *auto*
have *matrix-sum K proj-k* (*Suc m*) = *proj-k m* + *matrix-sum K proj-k m* **by** *simp*
also have ... = *mat K K* ($\lambda(i, j)$. *if* ($i = j \wedge i <$ (*Suc m*)) *then 1 else 0*)
unfolding *proj-k-mat*[*OF m*] *Suc*(*1*)[*OF m'*] **apply** (*rule eq-matI*) **by** *auto*
finally show *?case* **by** *auto*
qed
then show *?thesis* **unfolding** *proj-fst-k-def* **using** *assms* **by** *auto*
qed

lemma *proj-psi-proj-k-le-exproj-k*:
shows *tensor-P* (*proj-psi-l k*) (*proj-k l*) $\leq_L$ *tensor-P* (1_m *N*) (*proj-k l*)
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive-le*)
subgoal using *proj-psi-l-def psi-l-dim ps-P-d1* **by** *auto*
subgoal using *proj-k-def ket-k-def ps-P-d2* **by** *auto*
subgoal using *positive-proj-psi-l* **by** *auto*
subgoal using *positive-same-outer-prod proj-k-def ket-k-def* **by** *auto*
subgoal using *psi-l-le-id* **by** *auto*
apply (*subst lowner-le-refl*[*of - K*]) **by** (*auto simp add*: *proj-k-def ket-k-def*)

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

lemma *tensor-P-left-right-partial1*:
assumes *m1* $\in$ *carrier-mat N N* **and** *m2* $\in$ *carrier-mat N N* **and** *m3* $\in$ *carrier-mat K K* **and** *m4* $\in$ *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 −
have *tensor-P m1* (1_m *K*) $*$ *tensor-P m2 m3* = *tensor-P* (*m1* $*$ *m2*) *m3*
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-mult*[*symmetric*])
using *assms ps-P-d1 ps-P-d2* **by** *auto*

moreover have *tensor-P* $(m1 * m2)$ $m3 * $ *tensor-P* $m4$ $(1_m\ K)$ = *tensor-P* $(m1 * m2 * m4)$ $m3$
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-mult*[*symmetric*])
using *assms ps-P-d1 ps-P-d2* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *tensor-P-left-right-partial2*:
assumes $m1 \in$ *carrier-mat K K* **and** $m2 \in$ *carrier-mat K K* **and** $m3 \in$ *carrier-mat N N* **and** $m4 \in$ *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 −
have *tensor-P* $(1_m\ N)$ $m1 *$ *tensor-P* $m3\ m2$ = *tensor-P* $m3$ $(m1 * m2)$
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-mult*[*symmetric*])
using *assms ps-P-d1 ps-P-d2* **by** *auto*
moreover have *tensor-P* $m3$ $(m1 * m2)$ $*$ *tensor-P* $(1_m\ N)$ $m4$ = *tensor-P* $m3$ $(m1 * m2 * m4)$
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-mult*[*symmetric*])
using *assms ps-P-d1 ps-P-d2* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

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*: $A \in$ *carrier-mat m m* **and** *dB*: $B \in$ *carrier-mat m m*
shows *matrix-sum m* $(\lambda k.\ A * g\ k * B)$ $l = A *$ *matrix-sum m g l* $* B$
proof −
have *eq*: $A *$ *matrix-sum m g l* = *matrix-sum m* $(\lambda k.\ A * g\ k)$ l
using *matrix-sum-distrib-left assms* **by** *auto*
have $A *$ *matrix-sum m g l* $* B$ = *matrix-sum m* $(\lambda k.\ A * g\ k * B)$ l
apply (*subst eq*)
using *matrix-sum-mult-right*[*of l* $\lambda k.\ A * g\ k$] *assms* **by** *auto*
then show *?thesis* **by** *auto*
qed

lemma *mat-O-split*:
mat-O = $1_m\ N - 2 \cdot_m$ *proj-O*
apply (*rule eq-matI*)
unfolding *mat-O-def proj-O-def* **by** *auto*

lemma *mat-O-mult-psi′-l*:
mat-O $*_v$ (*psi′-l l*) = *psi-l l*
proof −
have *mat-O* $*_v$ (*psi′-l l*) = *mat-O* $*_v$ ((*alpha-l l*) $\cdot_v\ \alpha$) − *mat-O* $*_v$ ((*beta-l l*) $\cdot_v$

β)
unfolding *psi'-l-def* **apply** (*subst mult-minus-distrib-mat-vec*)
using *mat-O-dim* α*-dim* β*-dim* **by** *auto*
also have ... = (*alpha-l l*) $\cdot_v$ (*mat-O* $*_v$ α) − (*beta-l l*) $\cdot_v$ (*mat-O* $*_v$ β)
using *mult-mat-vec-smult-vec-assoc*[*of mat-O N N*] *mat-O-dim* α*-dim* β*-dim* **by** *auto*
also have ... = (*alpha-l l*) $\cdot_v$ α − (*beta-l l*) $\cdot_v$ (− β)
using *mat-O-mult-alpha mat-O-mult-beta* **by** *auto*
also have ... = (*alpha-l l*) $\cdot_v$ α + (*beta-l l*) $\cdot_v$ β
by *auto*
finally show *?thesis* **unfolding** *psi-l-def* **by** *auto*
qed

lemma *mat-O-times-Q1*:
adjoint (*tensor-P mat-O* (1_m *K*)) ∗ *Q1* ∗ (*tensor-P mat-O* (1_m *K*)) = *Q*
proof −
let *?m1* = *tensor-P mat-O* (1_m *K*)
have *eq*:*adjoint ?m1* = *?m1*
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-adjoint*)
apply (*auto simp add*: *mat-O-dim ps-P-d1 ps-P-d2*)
by (*simp add*: *hermitian-mat-O*[*unfolded hermitian-def*] *hermitian-one*[*unfolded hermitian-def*])
{
fix *l*
let *?m2* = *tensor-P* (*proj-psi'-l l*) (*proj-k l*)
have *?m1* ∗ *?m2* ∗ *?m1* = *tensor-P* (*mat-O* ∗ (*proj-psi'-l l*) ∗ *mat-O*) (*proj-k l*)
apply (*subst tensor-P-left-right-partial1*)
using *mat-O-dim proj-psi'-dim proj-k-dim* **by** *auto*
moreover have *mat-O* ∗ (*proj-psi'-l l*) ∗ *mat-O* = *outer-prod* (*psi-l l*) (*psi-l l*)
unfolding *proj-psi'-l-def* **apply** (*subst outer-prod-left-right-mat*[*of - N - N - N - N*])
using *psi'-l-dim mat-O-dim mat-O-mult-psi'-l hermitian-mat-O*[*unfolded hermitian-def*] **by** *auto*
ultimately have *?m1* ∗ *?m2* ∗ *?m1* = *tensor-P* (*proj-psi-l l*) (*proj-k l*) **unfolding** *proj-psi-l-def* **by** *auto*
}
note *p1* = *this*
have *adjoint* (*tensor-P mat-O* (1_m *K*)) ∗ *Q1* ∗ (*tensor-P mat-O* (1_m *K*)) = *?m1* ∗ *Q1* ∗ *?m1*
using *eq* **by** *auto*
also have ... = *matrix-sum d* (λ*l. ?m1* ∗ (*tensor-P* (*proj-psi'-l l*) (*proj-k l*)) ∗ *?m1*) *R*
unfolding *Q1-def*
apply (*subst matrix-sum-mult-left-right*) **using** *tensor-P-dim* **by** *auto*
also have ... = *Q*
unfolding *Q-def* **using** *p1* **by** *auto*
finally show *?thesis* **by** *auto*

qed

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

lemma *Q2-dim*:
Q2 ∈ *carrier-mat d d*
unfolding *Q2-def* **apply** (*subst matrix-sum-dim*) **using** *tensor-P-dim* **by** *auto*

lemma *Q2-le-one*:
Q2 $\leq_L$ 1_m *d*
proof −
have *leq*: *Q2* $\leq_L$ *matrix-sum d* (λ*k*. *tensor-P* (1_m *N*) (*proj-k k*)) *R*
unfolding *Q2-def*
apply (*subst lowner-le-matrix-sum*)
subgoal using *tensor-P-dim* **by** *auto*
subgoal using *tensor-P-dim* **by** *auto*
using *proj-psi-proj-k-le-exproj-k* **by** *auto*
have *matrix-sum d* (λ*k*. *tensor-P* (1_m *N*) (*proj-k k*)) *R*
= *tensor-P* (1_m *N*) (*matrix-sum K proj-k R*)
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-matrix-sum2*[*simplified ps-P-d ps-P-d2*])
subgoal using *ps-P-d1* **by** *auto*
using *proj-k-dim* **by** *auto*
also have ... = *tensor-P* (1_m *N*) (*proj-fst-k R*) **using** *sum-proj-k K* **by** *auto*
also have ... $\leq_L$ *tensor-P* (1_m *N*) (1_m *K*) **unfolding** *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive-le*)
subgoal using *ps-P-d1* **by** *auto*
subgoal using *ps-P-d2 proj-fst-k-def* **by** *auto*
subgoal using *positive-one* **by** *auto*
subgoal using *positive-proj-fst-k* **by** *auto*
subgoal using *lowner-le-refl*[*of* 1_m *N N*] **by** *auto*
using *proj-fst-k-le-one* **by** *auto*
also have ... = 1_m *d* **unfolding** *ps2-P.ptensor-mat-def*
using *ps-P.tensor-mat-id ps-P-d1 ps-P-d2 ps-P-d* **by** *auto*
finally have *leq2*: *matrix-sum d* (λ*k*. *tensor-P* (1_m *N*) (*proj-k k*)) *R* $\leq_L$ 1_m *d*
by *auto*
have *ds*: *matrix-sum d* (λ*k*. *tensor-P* (1_m *N*) (*proj-k k*)) *R* ∈ *carrier-mat d d*
apply (*subst matrix-sum-dim*) **using** *tensor-P-dim* **by** *auto*
then show *?thesis* **using** *leq leq2 lowner-le-trans*[*OF Q2-dim ds*, *of* 1_m *d*] **by**
auto
qed

lemma *qp-Q2*:
is-quantum-predicate Q2
unfolding *is-quantum-predicate-def*
proof (*intro conjI*)
show *Q2* ∈ *carrier-mat d d* **unfolding** *Q2-def*
apply (*subst matrix-sum-dim*) **using** *tensor-P-dim* **by** *auto*

next
  show *positive Q2* **unfolding** *Q2-def*
    apply (*subst matrix-sum-positive*)
    subgoal using *tensor-P-dim* **by** *auto*
    subgoal for *k* **unfolding** *ps2-P.ptensor-mat-def*
      apply (*subst ps-P.tensor-mat-positive*)
      subgoal using *proj-psi-l-def psi-l-dim ps-P-d1* **by** *auto*
      subgoal using *proj-k-dim ps-P-d2 K* **by** *auto*
      subgoal using *positive-proj-psi-l* **by** *auto*
       using *positive-proj-k K* **by** *auto*
    by *auto*
next
  show $Q2 \leq_L 1_m\ d$ **using** *Q2-le-one* **by** *auto*
qed

lemma *pre-mat*:
  $pre = mat\ N\ N\ (\lambda(i, j).\ if\ i = j \wedge i = 0\ then\ 1\ else\ 0)$
  apply (*rule eq-matI*)
 subgoal for *i j* **unfolding** *pre-def* **apply** (*subst index-outer-prod*[*OF ket-pre-dim ket-pre-dim*])
      apply *simp-all*
    unfolding *ket-pre-def* **by** *auto*
  using *outer-prod-dim*[*OF ket-pre-dim ket-pre-dim*, *folded pre-def*] **by** *auto*

lemma *mat-Ph-split*:
  $mat\text{-}Ph = 2 \cdot_m pre - 1_m\ N$
  unfolding *mat-Ph-def pre-mat*
  apply (*rule eq-matI*) **by** *auto*

lemma *H-Ph-H*:
  $exexH\text{-}k\ (n{-}1) * tensor\text{-}P\ mat\text{-}Ph\ (1_m\ K) * exexH\text{-}k\ (n - 1) = 2 \cdot_m tensor\text{-}P\ proj\text{-}psi\ (1_m\ K) - 1_m\ d$
  unfolding *mat-Ph-split exexH-k.simps*
  apply (*subst tensor-P-left-right-partial1*)
  subgoal using *exH-k-dim*[*of n − 1*] *n* **by** *auto*
  subgoal using *pre-dim* **by** *auto*
  subgoal by *auto*
proof −
  have *eq1*: $exH\text{-}n * exH\text{-}n = 1_m\ N$
    using *unitary-exH-k*[*of n − 1*]
    unfolding *unitary-def inverts-mat-def*
    using *n hermitian-exH-n*[*simplified hermitian-def*] *exH-n-dim* **by** *auto*
  have *eq2*: $exH\text{-}n * pre * exH\text{-}n = proj\text{-}psi$
    unfolding *pre-def proj-psi-def*
    apply (*subst outer-prod-left-right-mat*[*of - N - N - N - N*])
    subgoal using *ket-pre-dim* **by** *auto*
    subgoal using *exH-n-dim* **by** *auto*
    apply (*subst hermitian-exH-n*[*simplified hermitian-def*])
    using *exH-k-mult-pre-is-psi* **by** *auto*

have *eq3*: *exH-n* ∗ (*2* $\cdot_m$ *pre*) ∗ *exH-n* = *2* $\cdot_m$ (*exH-n* ∗ *pre* ∗ *exH-n*)
using *pre-dim exH-n-dim* **by** (*mat-assoc N*)
have *exH-n* ∗ (*2* $\cdot_m$ *pre* − 1_m *N*) ∗ *exH-n* = *exH-n* ∗ (*2* $\cdot_m$ *pre*) ∗ *exH-n* − *exH-n* ∗ *exH-n*
using *pre-dim exH-n-dim* **apply** (*mat-assoc N*) **by** *auto*
also have ... = *2* $\cdot_m$ (*exH-n* ∗ *pre* ∗ *exH-n*) − 1_m *N*
using *eq1 eq3* **by** *auto*
finally have *eq4*: *exH-n* ∗ (*2* $\cdot_m$ *pre* − 1_m *N*) ∗ *exH-n* = *2* $\cdot_m$ *proj-psi* − 1_m *N* **using** *eq2* **by** *auto*
show *tensor-P* (*exH-n* ∗ (*2* $\cdot_m$ *pre* − 1_m *N*) ∗ *exH-n*) (1_m *K*) = *2* $\cdot_m$ *tensor-P proj-psi* (1_m *K*) − 1_m *d*
unfolding *eq4* **unfolding** *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-minus1*)
unfolding *ps-P-d1 ps-P-d2* **apply** (*auto simp add*: *proj-psi-dim*)
apply (*subst ps-P.tensor-mat-scale1*)
unfolding *ps-P-d1 ps-P-d2* **apply** (*auto simp add*: *proj-psi-dim*)
apply (*subst ps-P.tensor-mat-id*[*simplified ps-P-d1 ps-P-d2 ps-P-d*]) **by** *auto*
qed

lemma *hermitian-proj-psi-minus-1*:
hermitian (*2* $\cdot_m$ *proj-psi* − 1_m *N*)
unfolding *hermitian-def*
apply (*subst adjoint-minus*[*of - N N*])
apply (*auto simp add*: *proj-psi-dim*)
apply (*subst adjoint-scale*)
using *hermitian-proj-psi*[*simplified hermitian-def*] *hermitian-def adjoint-one* **by** *auto*

lemma *unitary-proj-psi-minus-1*:
unitary (*2* $\cdot_m$ *proj-psi* − 1_m *N*)
proof −
have *a*: *adjoint* (*2* $\cdot_m$ *proj-psi*) = *2* $\cdot_m$ *proj-psi*
apply (*subst adjoint-scale*) **using** *hermitian-proj-psi*[*simplified hermitian-def*] **by** *simp*
have *eq*: *adjoint* (*2* $\cdot_m$ *proj-psi* − 1_m *N*) = *2* $\cdot_m$ *proj-psi* − 1_m *N*
apply (*subst adjoint-minus*) **using** *proj-psi-dim a adjoint-one* **by** *auto*
have (*2* $\cdot_m$ *proj-psi*) ∗ (*2* $\cdot_m$ *proj-psi*) = *4* $\cdot_m$ (*proj-psi* ∗ *proj-psi*)
using *proj-psi-dim* **by** *auto*
also have ... = *4* $\cdot_m$ *proj-psi* **using** *proj-psi-is-projection* **by** *auto*
finally have *sq*: (*2* $\cdot_m$ *proj-psi*) ∗ (*2* $\cdot_m$ *proj-psi*) = *4* $\cdot_m$ *proj-psi*.
have *l*: (*2* $\cdot_m$ *proj-psi*) ∗ (*2* $\cdot_m$ *proj-psi* − 1_m *N*) = *4* $\cdot_m$ *proj-psi* − (*2* $\cdot_m$ *proj-psi*)
apply (*subst mult-minus-distrib-mat*) **using** *proj-psi-dim sq* **by** *auto*

have (*2* $\cdot_m$ *proj-psi* − 1_m *N*) ∗ *adjoint* (*2* $\cdot_m$ *proj-psi* − 1_m *N*)
= (*2* $\cdot_m$ *proj-psi* − 1_m *N*) ∗ (*2* $\cdot_m$ *proj-psi* − 1_m *N*) **using** *eq* **by** *auto*
also have ... = (*2* $\cdot_m$ *proj-psi*) ∗ (*2* $\cdot_m$ *proj-psi* − 1_m *N*) − *2* $\cdot_m$ *proj-psi* + 1_m *N*
apply (*subst minus-mult-distrib-mat*[*of - N N*]) **using** *proj-psi-dim* **by** *auto*

also have . . . = *4* $\cdot_m$ *proj-psi* − (*2* $\cdot_m$ *proj-psi*) − *2* $\cdot_m$ *proj-psi* + *1*$_m$ *N*
using *l* **by** *auto*
also have . . . = *1*$_m$ *N* **using** *proj-psi-dim* **by** *auto*
finally have (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) ∗ *adjoint* (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) = *1*$_m$ *N* **.**
then show *?thesis* **unfolding** *unitary-def inverts-mat-def* **using** *proj-psi-dim* **by** *auto*
qed

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 −
have *eq1*: (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) $*_v$ *psi′-l l* = *2* $\cdot_m$ *proj-psi* $*_v$ *psi′-l l* − *psi′-l l*
apply (*subst minus-mult-distrib-mat-vec*)
using *psi′-l-dim proj-psi′-dim proj-psi-dim* **by** *auto*
have *eq2*: *2* $\cdot_m$ *proj-psi* $*_v$ (*psi′-l l*) = *2* $\cdot_v$ (*proj-psi* $*_v$ (*psi′-l l*))
apply (*subst smult-mat-mult-mat-vec-assoc*)
using *proj-psi-dim psi′-l-dim* **by** *auto*
have *proj-psi* $*_v$ (*psi′-l l*) = *inner-prod* ψ (*psi′-l l*) $\cdot_v$ ψ
unfolding *proj-psi-def*
apply (*subst outer-prod-mult-vec*[*of - N - N*])
using *ψ-dim psi′-l-dim* **by** *auto*
also have . . . = ((*alpha-l l*) ∗ *ccos* (ϑ / *2*) − (*beta-l l*) ∗ *csin* (ϑ / *2*)) $\cdot_v$ ψ
using *psi-inner-psi′-l* **by** *auto*
finally have *proj-psi* $*_v$ (*psi′-l l*) = ((*alpha-l l*) ∗ *ccos* (ϑ / *2*) − (*beta-l l*) ∗ *csin* (ϑ / *2*)) $\cdot_v$ ψ **by** *auto*
then have *eq3*: *2* $\cdot_v$ (*proj-psi* $*_v$ (*psi′-l l*)) = *2* ∗ ((*alpha-l l*) ∗ *ccos* (ϑ / *2*) − (*beta-l l*) ∗ *csin* (ϑ / *2*)) $\cdot_v$ ψ **by** *auto*
then show (*2* $\cdot_m$ *proj-psi* − (*1*$_m$ *N*)) $*_v$ (*psi′-l l*) = *psi-l* (*l* + *1*)
using *eq1 eq2 eq3 psi-l-Suc-l-derive* **by** *simp*
qed

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 −
have *id*: (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) ∗ (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) = *1*$_m$ *N*
using *unitary-proj-psi-minus-1* **unfolding** *unitary-def hermitian-proj-psi-minus-1*[*simplified hermitian-def*]
unfolding *inverts-mat-def* **by** *auto*
have (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) $*_v$ *psi-l* (*l* + *1*) = (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) $*_v$ ((*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) $*_v$ *psi′-l l*)
using *proj-psi-minus-1-mult-psi′-l* **by** *auto*
also have . . . = ((*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) ∗ (*2* $\cdot_m$ *proj-psi* − *1*$_m$ *N*) $*_v$ *psi′-l l*)
apply (*subst assoc-mult-mat-vec*) **using** *proj-psi-dim psi′-l-dim* **by** *auto*
also have . . . = *psi′-l l* **using** *psi′-l-dim id* **by** *auto*
finally show *?thesis* **by** *auto*
qed

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* $- (1_m\ N))$ $(1_m\ K)$
  unfolding *ps2-P.ptensor-mat-def*
  apply (*subst ps-P.tensor-mat-id*[*symmetric*, *simplified ps-P-d*])
  apply (*auto simp add*: *ps-P-d1 ps-P-d2*)
  apply (*subst ps-P.tensor-mat-scale1*[*symmetric*])
    apply (*auto simp add*: *ps-P-d1 ps-P-d2 proj-psi-dim*)
  apply (*subst ps-P.tensor-mat-minus1*)
  by (*auto simp add*: *ps-P-d1 ps-P-d2 proj-psi-dim*)

lemma *unitary-exproj-psi-minus-1*:
  unitary $(2 \cdot_m$ *tensor-P proj-psi* $(1_m\ K) - 1_m\ d)$
  unfolding *exproj-psi-minus-1-tensor*
  unfolding *ps2-P.ptensor-mat-def*
  apply (*subst ps-P.tensor-mat-unitary*)
  using *ps-P-d1 ps-P-d2 unitary-proj-psi-minus-1 unitary-one* **by** *auto*

lemma *proj-psi-minus-1-Q2*:
  adjoint $(2 \cdot_m$ *tensor-P proj-psi* $(1_m\ K) - 1_m\ d) * Q2 * (2 \cdot_m$ *tensor-P proj-psi* $(1_m\ K) - 1_m\ d) = Q1$
proof −
  have *eq1*: *adjoint* $(2 \cdot_m$ *tensor-P proj-psi* $(1_m\ K) - 1_m\ d) = 2 \cdot_m$ *tensor-P proj-psi* $(1_m\ K) - 1_m\ d$
    apply (*subst adjoint-minus*[*of - d d*])
    subgoal using *tensor-P-dim*[*of proj-psi*] **by** *auto*
    subgoal by *auto*
    apply (*subst adjoint-one*) **apply** (*subst adjoint-scale*)
    using *hermitian-exproj-psi*[*simplified hermitian-def*] **by** *auto*
  let *?m1* = *tensor-P* $(2 \cdot_m$ *proj-psi* $- (1_m\ N))$ $(1_m\ K)$
  {
    fix *l*
    let *?m2* = *tensor-P* (*proj-psi-l* $(l + 1)$) (*proj-k l*)
    have *121*: *?m1* ∗ *?m2* ∗ *?m1*
       = *tensor-P* $((2 \cdot_m$ *proj-psi* $- (1_m\ N)) *$ (*proj-psi-l* $(l + 1)) * (2 \cdot_m$ *proj-psi* $- (1_m\ N)))$
          (*proj-k l*)
      apply (*subst tensor-P-left-right-partial1*)
      using *proj-psi-dim proj-psi-l-dim proj-k-dim* **by** *auto*
    have $(2 \cdot_m$ *proj-psi* $- (1_m\ N)) *$ (*proj-psi-l* $(l + 1)) * (2 \cdot_m$ *proj-psi* $- (1_m\ N))$
      = *outer-prod* $((2 \cdot_m$ *proj-psi* $- (1_m\ N)) *_v$ (*psi-l* $(l + 1)$)) $((2 \cdot_m$ *proj-psi* $- (1_m\ N)) *_v$ (*psi-l* $(l + 1)$))
      unfolding *proj-psi-l-def* **apply** (*subst outer-prod-left-right-mat*[*of - N - N - N - N*])
        using *proj-psi-dim psi-l-dim hermitian-proj-psi-minus-1*[*simplified hermitian-def*] **by** *auto*
    also have ... = *outer-prod* (*psi′-l l*) (*psi′-l l*)
      using *proj-psi-minus-1-mult-psi-Suc-l* **by** *auto*

finally have (*2* $\cdot_m$ *proj-psi* − (*1*$_m$ *N*)) ∗ (*proj-psi-l* (*l* + *1*)) ∗ (*2* $\cdot_m$ *proj-psi* − (*1*$_m$ *N*))
= *outer-prod* (*psi′-l l*) (*psi′-l l*)**.**
then have *?m1* ∗ *?m2* ∗ *?m1* = *tensor-P* (*proj-psi′-l l*) (*proj-k l*)
using *121 proj-psi′-l-def* **by** *auto*
}
note *p1* = *this*
have *adjoint* (*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*) ∗ *Q2* ∗ (*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*)
= (*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*) ∗ *Q2* ∗ (*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*)
using *eq1* **by** *auto*
also have . . . = *matrix-sum d*
(λ*l*. (*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*) ∗ *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k l*) ∗ (*2* $\cdot_m$ *tensor-P proj-psi* (*1*$_m$ *K*) − *1*$_m$ *d*))
R **unfolding** *Q2-def* **apply** (*subst matrix-sum-mult-left-right*)
using *tensor-P-dim* **by** *auto*
also have . . . = *matrix-sum d* (λ*l*. *tensor-P* (*proj-psi′-l l*) (*proj-k l*)) *R*
using *p1 exproj-psi-minus-1-tensor* **by** *auto*
also have . . . = *Q1* **unfolding** *Q1-def* **by** *auto*
finally show *?thesis* **using** *eq1* **by** *auto*
qed

lemma *qp-Q1*:
is-quantum-predicate Q1
unfolding *proj-psi-minus-1-Q2*[*symmetric*]
apply (*subst qp-close-under-unitary-operator*)
using *tensor-P-dim unitary-exproj-psi-minus-1 qp-Q2* **by** *auto*

lemma *qp-Q*:
is-quantum-predicate Q
proof −
have *u*: *unitary* (*tensor-P mat-O* (*1*$_m$ *K*))
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-unitary*)
subgoal unfolding *ps-P-d1 mat-O-def* **by** *auto*
subgoal unfolding *ps-P-d2* **by** *auto*
subgoal using *unitary-mat-O* **by** *auto*
using *unitary-one* **by** *auto*
then show *?thesis* **using** *tensor-P-dim qp-Q1*
using *qp-close-under-unitary-operator*[*OF tensor-P-dim u qp-Q1*]
by (*simp add*: *mat-O-times-Q1*)
qed

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

unfolding *Utrans-P-is-tensor-P1*
mat-O-times-Q1[*symmetric*]
apply (*subst hoare-partial.intros*(*2*))
using *qp-Q1* **by** *auto*

lemma *hoare-triple-D2*:
$\vdash_p$
{*Q1*}
hadamard-n n ;;
Utrans-P vars1 mat-Ph ;;
hadamard-n n
{*Q2*}
proof −
let *?H* = *exexH-k* (*n* − *1*)
let *?Ph* = *tensor-P mat-Ph* (1_m *K*)
let *?O* = *tensor-P mat-O* (1_m *K*)
have *h1*: $\vdash_p$
{*adjoint ?H* ∗ *Q2* ∗ *?H*}
hadamard-n n
{*Q2*}
using *hoare-hadamard-n*[*OF qp-Q2*, *of n* − *1*] *n* **by** *auto*
have *qp1*: *is-quantum-predicate* ((*adjoint ?H*) ∗ *Q2* ∗ *?H*)
using *qp-close-under-unitary-operator unitary-exexH-k n exexH-k-dim qp-Q2*
by *auto*
then have *h2*: $\vdash_p$
{*adjoint ?Ph* ∗ (*adjoint ?H* ∗ *Q2* ∗ *?H*) ∗ *?Ph*}
Utrans-P vars1 mat-Ph
{*adjoint ?H* ∗ *Q2* ∗ *?H*}
using *qp1 Utrans-P-is-tensor-P1 hoare-partial.intros* **by** *auto*
have *qp2*: *is-quantum-predicate* (*adjoint ?Ph* ∗ (*adjoint ?H* ∗ *Q2* ∗ *?H*) ∗ *?Ph*)
using *qp-close-under-unitary-operator*[*of tensor-P mat-Ph* (1_m *K*)] *ps2-P.ptensor-mat-carrier*
ps2-P-d0 unitary-ex-mat-Ph qp1 **by** *auto*
then have *h3*: $\vdash_p$
{*adjoint ?H* ∗ (*adjoint ?Ph* ∗ (*adjoint ?H* ∗ *Q2* ∗ *?H*) ∗ *?Ph*) ∗ *?H*}
hadamard-n n
{*adjoint ?Ph* ∗ (*adjoint ?H* ∗ *Q2* ∗ *?H*) ∗ *?Ph*}
using *hoare-hadamard-n*[*OF qp2*, *of n* − *1*] *n* **by** *auto*
have *qp3*: *is-quantum-predicate* (*adjoint ?H* ∗ (*adjoint ?Ph* ∗ (*adjoint ?H* ∗ *Q2*
∗ *?H*) ∗ *?Ph*) ∗ *?H*)
using *qp-close-under-unitary-operator*[*of ?H*] *exexH-k-dim unitary-exexH-k qp2*
n **by** *auto*
have *h4*: $\vdash_p$
{*adjoint ?H* ∗ (*adjoint ?Ph* ∗ (*adjoint ?H* ∗ *Q2* ∗ *?H*) ∗ *?Ph*) ∗ *?H*}
hadamard-n n ;;
Utrans-P vars1 mat-Ph
{*adjoint ?H* ∗ *Q2* ∗ *?H*}
using *h2 h3 qp1 qp2 qp3 hoare-partial.intros* **by** *auto*
then have *h5*: $\vdash_p$
{*adjoint ?H* ∗ (*adjoint ?Ph* ∗ (*adjoint ?H* ∗ *Q2* ∗ *?H*) ∗ *?Ph*) ∗ *?H*}

```
    hadamard-n n ;;
    Utrans-P vars1 mat-Ph ;;
    hadamard-n n
    {Q2}
     using h1 qp-Q2 qp3 qp1 hoare-partial.intros(3)[OF qp3 qp1 qp-Q2 h4 h1] by
auto

  have adjoint ?H ∗ (adjoint ?Ph ∗ (adjoint ?H ∗ Q2 ∗ ?H) ∗ ?Ph) ∗ ?H =
       adjoint (?H ∗ ?Ph ∗ ?H) ∗ Q2 ∗ (?H ∗ ?Ph ∗ ?H)
    apply (mat-assoc d) using exexH-k-dim n tensor-P-dim Q2-dim by auto
  also have ... = Q1 using H-Ph-H proj-psi-minus-1-Q2 by auto
  finally show ?thesis using h5 by auto
qed

definition exM0 where
  exM0 = tensor-P (1_m N) M0

lemma M0-mult-ket-k-R:
  M0 ∗_v ket-k R = ket-k R
  apply (rule eq-vecI)
  unfolding M0-def ket-k-def
  by (auto simp add: scalar-prod-def sum-only-one-neq-0)

lemma exP0-P':
  adjoint exM0 ∗ P' ∗ exM0 = P'
proof −
  have eq: adjoint exM0 = exM0
    unfolding exM0-def ps2-P.ptensor-mat-def
    apply (subst ps-P.tensor-mat-adjoint)
    unfolding ps-P-d1 ps-P-d2 using M0-dim adjoint-one hermitian-M0[unfolded
hermitian-def] by auto
  have eq2: M0 ∗ (proj-k R) ∗ M0 = (proj-k R)
    unfolding proj-k-def
    apply (subst outer-prod-left-right-mat[of - K - K - K - K])
    unfolding hermitian-M0[unfolded hermitian-def] M0-mult-ket-k-R
    using ket-k-dim M0-dim by auto
  show ?thesis unfolding eq unfolding exM0-def P'-def
    apply (subst tensor-P-left-right-partial2)
    using M0-dim proj-k-dim eq2 proj-psi-l-dim by auto
qed

definition exM1 where
  exM1 = tensor-P (1_m N) M1

lemma M1-mult-ket-k:
  assumes k < R
  shows M1 ∗_v ket-k k = ket-k k
  apply (rule eq-vecI)
  unfolding M1-def ket-k-def
```

by (*auto simp add*: *scalar-prod-def assms R sum-only-one-neq-0*)

lemma *exP1-Q*:
adjoint exM1 ∗ *Q* ∗ *exM1* = *Q*
proof −
have *eq*: *adjoint exM1* = *exM1*
unfolding *exM1-def ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-adjoint*)
unfolding *ps-P-d1 ps-P-d2* **using** *M1-dim adjoint-one hermitian-M1*[*unfolded hermitian-def*] **by** *auto*
{
fix *k* **assume** *k*: *k* < *R*
let *?m* = *tensor-P* (*proj-psi-l k*) (*proj-k k*)
have *exM1* ∗ *?m* ∗ *exM1* = *tensor-P* (*proj-psi-l k*) (*M1* ∗ (*proj-k k*) ∗ *M1*)
unfolding *exM1-def* **apply** (*subst tensor-P-left-right-partial2*)
using *M1-dim proj-k-dim proj-psi-l-dim* **by** *auto*
also have ... = *tensor-P* (*proj-psi-l k*) (*outer-prod* (*M1* $*_v$ *ket-k k*) (*M1* $*_v$ *ket-k k*))
unfolding *proj-k-def* **apply** (*subst outer-prod-left-right-mat*[*of - K - K - K - K*])
unfolding *hermitian-M1*[*unfolded hermitian-def*]
using *ket-k-dim M1-dim* **by** *auto*
finally have *exM1* ∗ *?m* ∗ *exM1* = *?m* **unfolding** *proj-k-def* **using** *k M1-mult-ket-k* **by** *auto*
}
note *p1* = *this*
have *adjoint exM1* ∗ *Q* ∗ *exM1* = *exM1* ∗ *Q* ∗ *exM1* **using** *eq* **by** *auto*
also have ... = *matrix-sum d* (λ*k*. *exM1* ∗ (*tensor-P* (*proj-psi-l k*) (*proj-k k*)) ∗ *exM1*) *R*
unfolding *Q-def*
apply (*subst matrix-sum-mult-left-right*)
using *tensor-P-dim exM1-def* **by** *auto*
also have ... = *matrix-sum d* (λ*k*. *tensor-P* (*proj-psi-l k*) (*proj-k k*)) *R*
apply (*subst matrix-sum-cong*)
using *p1* **by** *auto*
finally show *?thesis* **using** *Q-def* **by** *auto*
qed

lemma *qp-P′*:
is-quantum-predicate P′
unfolding *is-quantum-predicate-def*
proof (*intro conjI*)
show *P′* ∈ *carrier-mat d d* **unfolding** *P′-def* **using** *tensor-P-dim* **by** *auto*
show *positive P′* **unfolding** *P′-def ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive*)
apply (*auto simp add*: *ps-P-d1 ps-P-d2 proj-O-dim proj-k-dim*)
using *proj-psi-l-dim positive-proj-psi-l positive-proj-k K* **by** *auto*
show $P' \le_L 1_m\ d$ **unfolding** *P′-def ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-le-one*[*simplified ps-P-d*])

by (*auto simp add*: *ps-P-d1 ps-P-d2 proj-psi-l-dim K proj-k-dim positive-proj-psi-l positive-proj-k proj-k-le-one psi-l-le-id*)
qed

lemma *P′-add-Q*:
P′ + Q = matrix-sum d (*λl. tensor-P* (*proj-psi-l l*) (*proj-k l*)) (*R + 1*)
apply *simp* **unfolding** *P′-def Q-def* **by** *auto*

lemma *positive-Qk*:
positive (*tensor-P* (*proj-psi-l l*) (*proj-k l*))
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive*)
unfolding *ps-P-d1 ps-P-d2*
using *proj-psi-l-dim proj-k-dim positive-proj-psi-l positive-proj-k* **by** *auto*

lemma *P′-Q-dim*:
P′ + Q ∈ carrier-mat d d
unfolding *P′-add-Q*
apply (*subst matrix-sum-dim*)
using *tensor-P-dim* **by** *auto*

lemma *P′-add-Q-le-one*:
$P' + Q \le_L 1_m\ d$
proof −
have *leq*: *matrix-sum d* (*λl. tensor-P* (*proj-psi-l l*) (*proj-k l*)) (*R + 1*)
$\le_L$ *matrix-sum d* (*λk. tensor-P* (1_m *N*) (*proj-k k*)) (*R + 1*)
unfolding *Q2-def*
apply (*subst lowner-le-matrix-sum*)
subgoal using *tensor-P-dim* **by** *auto*
subgoal using *tensor-P-dim* **by** *auto*
using *proj-psi-proj-k-le-exproj-k* **by** *auto*
have *matrix-sum d* (*λk. tensor-P* (1_m *N*) (*proj-k k*)) (*R + 1*)
= *tensor-P* (1_m *N*) (*matrix-sum K proj-k* (*R + 1*))
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-matrix-sum2*[*simplified ps-P-d ps-P-d2*])
subgoal using *ps-P-d1* **by** *auto*
using *proj-k-dim* **by** *auto*
also have . . . = *tensor-P* (1_m *N*) (*proj-fst-k* (*R + 1*)) **using** *sum-proj-k*[*of R + 1*] *K* **by** *auto*
also have . . . $\le_L$ *tensor-P* (1_m *N*) (1_m *K*) **unfolding** *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive-le*)
subgoal using *ps-P-d1* **by** *auto*
subgoal using *ps-P-d2 proj-fst-k-def* **by** *auto*
subgoal using *positive-one* **by** *auto*
subgoal using *positive-proj-fst-k* **by** *auto*
subgoal using *lowner-le-refl*[*of* 1_m *N N*] **by** *auto*
using *proj-fst-k-le-one* **by** *auto*
also have . . . = 1_m *d* **unfolding** *ps2-P.ptensor-mat-def*
using *ps-P.tensor-mat-id ps-P-d1 ps-P-d2 ps-P-d* **by** *auto*

finally have *leq2*: *matrix-sum d* (*λk. tensor-P* (*1*$_m$ *N*) (*proj-k k*)) (*R* + *1*) $\leq_L$ *1*$_m$ *d* **by** *auto*
have *ds*: *matrix-sum d* (*λk. tensor-P* (*1*$_m$ *N*) (*proj-k k*)) (*R* + *1*) ∈ *carrier-mat d d*
apply (*subst matrix-sum-dim*) **using** *tensor-P-dim* **by** *auto*
then show *?thesis*
using *leq leq2 lowner-le-trans*[*OF P′-Q-dim ds*, *of 1*$_m$ *d*] **unfolding** *P′-add-Q* **by** *auto*
qed

lemma *qp-P′-Q*:
is-quantum-predicate (*P′* + *Q*)
unfolding *is-quantum-predicate-def*
proof (*intro conjI*)
show *P′* + *Q* ∈ *carrier-mat d d*
unfolding *P′-add-Q* **apply** (*subst matrix-sum-dim*)
using *tensor-P-dim* **by** *auto*
show *positive* (*P′* + *Q*) **unfolding** *P′-add-Q*
apply (*subst matrix-sum-positive*)
using *tensor-P-dim positive-Qk* **by** *auto*
show *P′* + *Q* $\leq_L$ *1*$_m$ *d* **using** *P′-add-Q-le-one* **by** *auto*
qed

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 −
have *ad*: *adjoint* (*tensor-P* (*1*$_m$ *N*) (*mat-incr K*)) = *tensor-P* (*1*$_m$ *N*) (*adjoint* (*mat-incr K*))
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-adjoint*)
using *ps-P-d1 ps-P-d2 mat-incr-dim adjoint-one* **by** *auto*
let *?m1* = *tensor-P* (*1*$_m$ *N*) (*mat-incr K*)
let *?m3* = *tensor-P* (*1*$_m$ *N*) (*adjoint* (*mat-incr K*))
{
fix *l* **assume** *l* < *R*
then have *l* < *K* − *1* **using** *K* **by** *auto*
then have *m*: (*mat-incr K*) $*_v$ (*ket-k l*) = (*ket-k* (*l* + *1*))
using *mat-incr-mult-ket-k* **by** *auto*
let *?m2* = *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k l*)
have *eq*: *?m1* ∗ *?m2* ∗ *?m3* = *tensor-P* (*proj-psi-l* (*l* + *1*)) ((*mat-incr K*) ∗ (*proj-k l*) ∗ *adjoint* (*mat-incr K*))
apply (*subst tensor-P-left-right-partial2*)
using *proj-k-dim proj-psi-l-dim mat-incr-dim adjoint-dim*[*OF mat-incr-dim*] **by** *auto*
have (*mat-incr K*) ∗ (*proj-k l*) ∗ *adjoint* (*mat-incr K*) = *outer-prod* ((*mat-incr K*) $*_v$ (*ket-k l*)) ((*mat-incr K*) $*_v$ (*ket-k l*))
unfolding *proj-k-def* **apply** (*subst outer-prod-left-right-mat*[*of - K - K - K - K*])
using *ket-k-dim mat-incr-dim adjoint-dim*[*OF mat-incr-dim*] *adjoint-adjoint*[*of*

mat-incr K] **by** *auto*
also have ... = *proj-k* (*l* + *1*) **unfolding** *proj-k-def* **using** *m* **by** *auto*
finally have *?m1* ∗ *?m2* ∗ *?m3* = *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k* (*l* + *1*)) **using** *eq* **by** *auto*
}
note *p1* = *this*
have *?m1* ∗ *Q2* ∗ *?m3*
= *matrix-sum d* (λ*l*. *?m1* ∗ (*tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k l*)) ∗ *?m3*) *R*
unfolding *Q2-def* **apply**(*subst matrix-sum-mult-left-right*)
using *tensor-P-dim* **by** *auto*
also have ... = *matrix-sum d* (λ*l*. *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k* (*l* + *1*))) *R*
apply (*subst matrix-sum-cong*) **using** *p1* **by** *auto*
finally have *eq1*: *?m1* ∗ *Q2* ∗ *?m3* = *matrix-sum d* (λ*l*. *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k* (*l* + *1*))) *R* (**is** *-=?r*) **.**
have *eq2*: *P′* + *Q* = *tensor-P* (*proj-psi-l 0*) (*proj-k 0*) + *?r*
unfolding *P′-add-Q*
apply (*subst matrix-sum-Suc-remove-head*) **using** *tensor-P-dim* **by** *auto*
have *tensor-P* (*proj-psi-l 0*) (*proj-k 0*) + *?r* $\leq_L$ *P′* + *Q*
unfolding *eq2*[*symmetric*] **apply** (*subst lowner-le-refl*) **using** *P′-Q-dim* **by** *auto*
moreover have *positive* (*tensor-P* (*proj-psi-l 0*) (*proj-k 0*))
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-positive*)
unfolding *ps-P-d1 ps-P-d2* **using** *proj-psi-l-dim proj-k-dim positive-proj-psi-l positive-proj-k* **by** *auto*
moreover have *matrix-sum d* (λ*l*. *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k* (*l* + *1*))) *R* ∈ *carrier-mat d d*
apply (*subst matrix-sum-dim*) **using** *tensor-P-dim* **by** *auto*
ultimately have *?r* $\leq_L$ *P′* + *Q*
apply (*subst add-positive-le-reduce2*[*of ?r d tensor-P* (*proj-psi-l 0*) (*proj-k 0*) *P′* + *Q*])
using *tensor-P-dim P′-Q-dim* **by** *auto*
then show *?thesis* **using** *eq1 ad* **by** *auto*
qed

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 −
let *?m1* = *tensor-P* (1_m *N*) (*mat-incr K*)
let *?m2* = *adjoint* (*tensor-P* (1_m *N*) (*mat-incr K*))
have *?m1* ∗ *?m2* = 1_m *d*
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-adjoint*)
unfolding *ps-P-d1 ps-P-d2* **apply** (*auto simp add*: *mat-incr-dim adjoint-one*)
apply (*subst ps-P.tensor-mat-mult*[*symmetric*])
unfolding *ps-P-d1 ps-P-d2* **apply** (*auto simp add*: *mat-incr-dim adjoint-dim mat-incr-mult-adjoint-mat-incr*)
using *ps-P.tensor-mat-id ps-P-d ps-P-d1 ps-P-d2* **by** *auto*

then have *inv*: *?m2* ∗ *?m1* = *1*$_m$ *d*
using *mat-mult-left-right-inverse*[*of ?m1 d ?m2*]
tensor-P-dim adjoint-dim **by** *auto*
have *d*: *?m1* ∗ *Q2* ∗ *?m2* ∈ *carrier-mat d d* **using** *tensor-P-dim adjoint-dim*[*OF tensor-P-dim*] *Q2-dim* **by** *fastforce*
have *le*: *?m2* ∗ (*?m1* ∗ *Q2* ∗ *?m2*) ∗ *?m1* $\leq_L$ *?m2* ∗ (*P*′ + *Q*) ∗ *?m1* (**is** *lowner-le ?l ?r*)
apply (*subst lowner-le-keep-under-measurement*[*of - d*])
using *Q2-leq-lemma tensor-P-dim P*′*-Q-dim d* **by** *auto*
have *?l* = (*?m2* ∗ *?m1*) ∗ *Q2* ∗ (*?m2* ∗ *?m1*)
apply (*mat-assoc d*) **using** *tensor-P-dim Q2-dim* **by** *auto*
also have . . . = *1*$_m$ *d* ∗ *Q2* ∗ *1*$_m$ *d* **using** *inv* **by** *auto*
also have . . . = *Q2* **using** *Q2-dim* **by** *auto*
finally have *eq*: *?l* = *Q2*.
show *?thesis* **using** *eq le* **by** *auto*
qed

lemma *hoare-triple-D3*:
$\vdash_p$
{*Q2*}
Utrans-P vars2 (*mat-incr K*)
{*adjoint exM0* ∗ *P*′ ∗ *exM0* + *adjoint exM1* ∗ *Q* ∗ *exM1*}
unfolding *exP0-P*′ *exP1-Q*
proof −
let *?m* = *tensor-P* (*1*$_m$ *N*) (*mat-incr K*)
have *h1*: $\vdash_p$
{*adjoint ?m* ∗ (*P*′ + *Q*) ∗ *?m*}
Utrans ?m
{*P*′ + *Q*}
using *qp-P*′*-Q hoare-partial.intros* **by** *auto*
have *qp*: *is-quantum-predicate* (*adjoint ?m* ∗ (*P*′ + *Q*) ∗ *?m*)
using *qp-close-under-unitary-operator tensor-P-dim qp-P*′*-Q unitary-exmat-incr* **by** *auto*
then have $\vdash_p$
{*Q2*}
Utrans ?m
{*P*′ + *Q*}
using *hoare-partial.intros*(*6*)[*OF qp-Q2 qp-P*′*-Q qp qp-P*′*-Q*] *Q2-leq h1 lowner-le-refl*[*OF P*′*-Q-dim*] **by** *auto*
moreover have *Utrans ?m* = *Utrans-P vars2* (*mat-incr K*)
apply (*subst Utrans-P-is-tensor-P2*) **unfolding** *mat-incr-def* **by** *auto*
ultimately show $\vdash_p$ {*Q2*} *Utrans-P vars2* (*mat-incr K*) {*P*′ + *Q*} **by** *auto*
qed

lemma *qp-D3-post*:
is-quantum-predicate (*adjoint exM0* ∗ *P*′ ∗ *exM0* + *adjoint exM1* ∗ *Q* ∗ *exM1*)
unfolding *exP0-P*′ *exP1-Q* **using** *qp-P*′*-Q* **by** *auto*

lemma *hoare-triple-D*:

$\vdash_p$
  {*Q*}
  D
  {*adjoint exM0* ∗ *P'* ∗ *exM0* + *adjoint exM1* ∗ *Q* ∗ *exM1*}
proof −
  have $\vdash_p$ {*Q1*} *hadamard-n n*;; (*Utrans-P vars1 mat-Ph*;; *hadamard-n n*) {*Q2*}
    using *well-com-hadamard-n well-com-mat-Ph hoare-triple-D2 qp-Q1 qp-Q2* **by** (*auto simp add*: *hoare-patial-seq-assoc*)
  then have $\vdash_p$ {*Q*} *Utrans-P vars1 mat-O*;; (*hadamard-n n*;; (*Utrans-P vars1 mat-Ph*;; *hadamard-n n*)) {*Q2*}
    using *hoare-triple-D1 qp-Q qp-Q1 qp-Q2 hoare-partial.intros*(*3*) **by** *auto*
  moreover have *well-com* (*Utrans-P vars1 mat-Ph*;; *hadamard-n n*) **using** *well-com-hadamard-n well-com-mat-Ph* **by** *auto*
  ultimately have $\vdash_p$ {*Q*} (*Utrans-P vars1 mat-O*;; *hadamard-n n*);; (*Utrans-P vars1 mat-Ph*;; *hadamard-n n*) {*Q2*}
    using *well-com-hadamard-n well-com-mat-O qp-Q qp-Q2* **by** (*auto simp add*: *hoare-patial-seq-assoc*)
  moreover have *well-com* (*Utrans-P vars1 mat-O*;; *hadamard-n n*)
    using *well-com-mat-O well-com-hadamard-n* **by** *auto*
  ultimately have $\vdash_p$ {*Q*} *Utrans-P vars1 mat-O*;; *hadamard-n n*;; *Utrans-P vars1 mat-Ph*;; *hadamard-n n* {*Q2*}
    using *well-com-hadamard-n well-com-mat-Ph qp-Q qp-Q2* **by** (*auto simp add*: *hoare-patial-seq-assoc*)
  with *qp-Q qp-Q2 qp-D3-post hoare-triple-D3* **show** $\vdash_p$
  {*Q*}
  D
  {*adjoint exM0* ∗ *P'* ∗ *exM0* + *adjoint exM1* ∗ *Q* ∗ *exM1*}
    unfolding *D-def* **using** *hoare-partial.intros*(*3*) **by** *auto*
qed

lemma *psi-is-psi-l0*:
  ψ = *psi-l 0*
  unfolding ψ*-eq psi-l-def alpha-l-def beta-l-def* **by** *auto*

lemma *proj-psi-is-proj-psi-l0*:
  proj-psi = *proj-psi-l 0*
  unfolding *proj-psi-def psi-is-psi-l0 proj-psi-l-def* **by** *auto*

lemma *lowner-le-Q*:
  tensor-P proj-psi (*proj-k 0*) $\leq_L$ *adjoint exM0* ∗ *P'* ∗ *exM0* + *adjoint exM1* ∗ *Q* ∗ *exM1*
proof −
  let *?r* = *matrix-sum d* (λ*l*. *tensor-P* (*proj-psi-l l*) (*proj-k l*)) (*R* + *1*)
  let *?l* = *tensor-P* (*proj-psi-l 0*) (*proj-k 0*)
  have *eq*: *?r* = *?l* + *matrix-sum d* (λ*l*. *tensor-P* (*proj-psi-l* (*l* + *1*)) (*proj-k* (*l* + *1*))) *R* (**is** - = - + *?s*)
    apply (*subst matrix-sum-Suc-remove-head*)
    using *tensor-P-dim* **by** *auto*
  have *d*: *?s* ∈ *carrier-mat d d*

apply (*subst matrix-sum-dim*) **using** *tensor-P-dim* **by** *auto*
have *pt*: *positive* (*tensor-P* (*proj-psi-l l*) (*proj-k l*)) **for** *l*
unfolding *ps2-P.ptensor-mat-def* **apply** (*subst ps-P.tensor-mat-positive*)
unfolding *ps-P-d1 ps-P-d2* **using** *proj-psi-l-dim proj-k-dim positive-proj-psi-l positive-proj-k* **by** *auto*
have *ps*: *positive ?s*
apply (*subst matrix-sum-positive*)
subgoal using *tensor-P-dim* **by** *auto*
using *pt* **by** *auto*
have *?l* $\leq_L$ *?r*
unfolding *eq*
apply (*subst add-positive-le-reduce1*[*of ?l d ?s*])
subgoal using *tensor-P-dim* **by** *auto*
subgoal using *d* **by** *auto*
subgoal using *tensor-P-dim d* **by** *auto*
subgoal using *ps* **by** *auto*
apply (*subst lowner-le-refl*[*of - d*])
using *tensor-P-dim d* **by** *auto*
then show *?thesis* **unfolding** *exP0-P′ exP1-Q P′-add-Q proj-psi-is-proj-psi-l0* **by** *auto*
qed

lemma *hoare-triple-while*:
$\vdash_p$
{*adjoint exM0* ∗ *P′* ∗ *exM0* + *adjoint exM1* ∗ *Q* ∗ *exM1*}
While-P vars2 M0 M1 D
{*P′*}
proof −
let *?m* = λ(*n*::*nat*). *if n* = *0 then mat-extension dims vars2 M0 else*
if n = *1 then mat-extension dims vars2 M1 else undefined*
have *dM0*: *M0* ∈ *carrier-mat K K* **unfolding** *M0-def* **by** *auto*
have *dM1*: *M1* ∈ *carrier-mat K K* **unfolding** *M1-def* **by** *auto*
have *m0*: *?m 0* = *exM0* **apply** (*simp*) **unfolding** *exM0-def ps2-P.ptensor-mat-def mat-ext-vars2*[*OF dM0*] **by** *auto*
have *m1*: *?m 1* = *exM1* **unfolding** *exM1-def ps2-P.ptensor-mat-def mat-ext-vars2*[*OF dM1*] **by** *auto*
have $\vdash_p$ {*Q*} *D* {*adjoint* (*?m 0*) ∗ *P′* ∗ (*?m 0*) + *adjoint* (*?m 1*) ∗ *Q* ∗ (*?m 1*)}
using *hoare-triple-D m0 m1* **by** *auto*
then show *?thesis* **unfolding** *While-P-def* **using** *qp-D3-post qp-P′ hoare-partial.intros*(*5*)[*OF qp-P′ qp-Q*, *of D ?m*] *m0 m1* **by** *auto*
qed

lemma *R-and-a-half-ϑ*:
(*R* + *1*/*2*) ∗ ϑ = *pi* / *2*
using *R ϑ-neq-0* **by** *auto*

lemma *psi-lR-is-beta*:
psi-l R = β
unfolding *psi-l-def alpha-l-def beta-l-def R-and-a-half-ϑ* **by** *auto*

lemma *post-mult-beta*:
  post $*_v$ β = β
  by (*auto simp add*: *post-def* β*-def scalar-prod-def sum-only-one-neq-0*)

lemma *post-mult-post*:
  post ∗ *post* = *post*
  by (*auto simp add*: *post-def scalar-prod-def sum-only-one-neq-0*)

lemma *post-mult-proj-psi-lR*:
  post ∗ *proj-psi-l R* = *proj-psi-l R*
proof −
  let *?R* = *proj-psi-l R*
  have *post* ∗ *?R* = *post* ∗ *?R* ∗ 1_m *N*
    using *post-dim proj-psi-l-dim*[*of R*] **by** *auto*
  also have . . . = *outer-prod* (*post* $*_v$ *psi-l R*) ((1_m *N*) $*_v$ *psi-l R*)
    unfolding *proj-psi-l-def*
    apply (*subst outer-prod-left-right-mat*[*of - N - N - N - N*])
    by (*auto simp add*: *psi-l-dim post-dim adjoint-one*)
  also have . . . = *?R* **unfolding** *proj-psi-l-def* **unfolding** *psi-lR-is-beta* **unfolding** *post-mult-beta*
    using β*-dim* **by** *auto*
  finally show *post* ∗ *?R* = *?R*.
qed

lemma *proj-psi-lR-mult-post*:
  proj-psi-l R ∗ *post* = *proj-psi-l R*
proof −
  let *?R* = *proj-psi-l R*
  have *?R* ∗ *post* = 1_m *N* ∗ *?R* ∗ *post*
    using *post-dim proj-psi-l-dim*[*of R*] **by** *auto*
  also have . . . = *outer-prod* ((1_m *N*) $*_v$ *psi-l R*) (*post* $*_v$ *psi-l R*)
    unfolding *proj-psi-l-def*
    apply (*subst outer-prod-left-right-mat*[*of - N - N - N - N*])
    by (*auto simp add*: *psi-l-dim post-dim hermitian-post*[*unfolded hermitian-def*])
  also have . . . = *?R* **unfolding** *proj-psi-l-def* **unfolding** *psi-lR-is-beta* **unfolding** *post-mult-beta*
    using β*-dim* **by** *auto*
  finally show *?R* ∗ *post* = *?R*.
qed

lemma *proj-psi-lR-mult-proj-psi-lR*:
  proj-psi-l R ∗ *proj-psi-l R* = *proj-psi-l R*
  unfolding *proj-psi-l-def psi-lR-is-beta*
  apply (*subst outer-prod-mult-outer-prod*[*of - N - N - - N*])
  by (*auto simp add*: β*-inner*)

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

proof −
  let *?R* = *proj-psi-l R*
  let *?s* = *post* − *?R*
  have *eq1*: *post* ∗ (*post* − *?R*) = *post* − *?R*
    apply (*subst mult-minus-distrib-mat*[*of* - *N N* - *N*])
      apply (*auto simp add*: *post-dim proj-psi-l-dim*[*of R*])
    using *post-mult-post post-mult-proj-psi-lR* **by** *auto*
  have *eq2*: *?R* ∗ (*post* − *?R*) = 0_m *N N*
    apply (*subst mult-minus-distrib-mat*[*of* - *N N* - *N*])
      apply (*auto simp add*: *post-dim proj-psi-l-dim*[*of R*])
    unfolding *proj-psi-lR-mult-post proj-psi-lR-mult-proj-psi-lR*
    using *proj-psi-l-dim*[*of R*] **by** *auto*
  have *adjoint ?s* = *?s*
    apply (*subst adjoint-minus*[*of* - *N N*])
     using *post-dim proj-psi-l-dim hermitian-post hermitian-proj-psi-l K* **by** (*auto simp add*: *hermitian-def*)
  then have *?s* ∗ *adjoint ?s* = *?s* ∗ *?s* **by** *auto*
  also have ... = *post* ∗ (*post* − *?R*) − *?R* ∗ (*post* − *?R*)
    using *post-dim proj-psi-l-dim*[*of R*] **by** (*mat-assoc N*)
  also have ... = *post* − *?R*
    unfolding *eq1 eq2* **using** *post-dim proj-psi-l-dim*[*of R*] **by** *auto*
  finally have *?s* ∗ *adjoint ?s* = *?s*.
  then have ∃ *M*. *M* ∗ *adjoint M* = *?s* **by** *auto*
  then have *positive ?s* **apply** (*subst positive-if-decomp*[*of ?s N*]) **using** *post-dim proj-psi-l-dim*[*of R*] **by** *auto*
  then show *?thesis* **unfolding** *lowner-le-def* **using** *post-dim proj-psi-l-dim*[*of R*] **by** *auto*
qed

lemma *P′-le-post-R*:
  $P' \leq_L$ (*tensor-P post* (*proj-k R*))
proof −
  let *?r* = *tensor-P post* (*proj-k R*)
  have *?r* − *P′* = *tensor-P* (*post* − *proj-psi-l R*) (*proj-k R*)
    unfolding *P′-def ps2-P.ptensor-mat-def*
    apply (*subst ps-P.tensor-mat-minus1*)
    unfolding *ps-P-d1 ps-P-d2*
    using *post-dim proj-psi-l-dim proj-k-dim* **by** *auto*
  moreover have *positive* (*tensor-P* (*post* − *proj-psi-l R*) (*proj-k R*))
    unfolding *ps2-P.ptensor-mat-def*
    apply (*subst ps-P.tensor-mat-positive*)
    unfolding *ps-P-d1 ps-P-d2*
    using *proj-psi-lR-le-post*[*unfolded lowner-le-def*]
      post-dim proj-psi-l-dim[*of R*] *proj-k-dim positive-proj-k*
    by *auto*
  ultimately show $P' \leq_L$ *?r*
    unfolding *lowner-le-def P′-def*
    using *tensor-P-dim* **by** *auto*
qed

lemma *positive-post*:
positive post
proof −
have *ad*: *adjoint post* = *post* **using** *hermitian-post*[*unfolded hermitian-def*] **by** *auto*
then have *post* ∗ *adjoint post* = *post*
unfolding *ad post-mult-post* **by** *auto*
then have ∃ *M*. *M* ∗ *adjoint M* = *post* **by** *auto*
then show *?thesis* **using** *positive-if-decomp post-dim* **by** *auto*
qed

lemma *lowner-le-P'*:
$P' \le_L$ *tensor-P post* (1_m *K*)
proof −
let *?r* = *tensor-P post* (1_m *K*)
let *?m* = *tensor-P post* (*proj-k R*)
have *?m* $\le_L$ *?r*
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive-le*)
unfolding *ps-P-d1 ps-P-d2*
using *post-dim proj-k-dim positive-post positive-proj-k*
lowner-le-refl[*of post*] *proj-k-le-one* **by** *auto*
then show $P' \le_L$ *?r*
using *lowner-le-trans*[*of P' d ?m ?r*] *P'-le-post-R*
unfolding *P'-def* **using** *tensor-P-dim* **by** *auto*
qed

lemma *post-mult-testNk*:
assumes *f k*
shows *post* ∗ (*testN k*) = *testN k*
using *assms* **by** (*auto simp add*: *post-def testN-def scalar-prod-def sum-only-one-neq-0*)

lemma *post-mult-testNk-neg*:
assumes ¬ *f k*
shows *post* ∗ *testN k* = 0_m *N N*
using *assms* **by** (*auto simp add*: *post-def testN-def scalar-prod-def sum-only-one-neq-0*)

lemma *testN-post1*:
f k ⟹ *adjoint* (*testN k*) ∗ *post* ∗ *testN k* = *testN k*
apply (*subst assoc-mult-mat*[*of - N N - N - N*])
apply (*auto simp add*: *adjoint-dim testN-dim post-dim*)
apply (*subst post-mult-testNk*, *simp*)
unfolding *hermitian-testN*[*unfolded hermitian-def*]
using *testN-mult-testN* **by** *auto*

lemma *testN-post2*:
¬ *f k* ⟹ *adjoint* (*testN k*) ∗ *post* ∗ *testN k* = 0_m *N N*
apply (*subst assoc-mult-mat*[*of - N N - N - N*])

apply (*auto simp add*: *adjoint-dim testN-dim post-dim*)
apply (*subst post-mult-testNk-neg*, *simp*)
unfolding *hermitian-testN*[*unfolded hermitian-def*]
using *testN-dim*[*of k*] **by** *auto*

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

lemma *post-fst-kN*:
post-fst-k N = *post*
unfolding *post-fst-k-def post-def* **by** *auto*

lemma *post-fst-k-Suc*:
$f\ i \Longrightarrow post\text{-}fst\text{-}k\ (Suc\ i) = testN\ i + post\text{-}fst\text{-}k\ i$
apply (*rule eq-matI*)
unfolding *post-fst-k-def testN-def* **by** *auto*

lemma *post-fst-k-Suc-neg*:
$\neg\ f\ i \Longrightarrow post\text{-}fst\text{-}k\ (Suc\ i) = post\text{-}fst\text{-}k\ i$
apply (*rule eq-matI*)
unfolding *post-fst-k-def*
apply *auto*
using *less-antisym* **by** *fastforce*

lemma *testN-sum*:
$matrix\text{-}sum\ N\ (\lambda k.\ adjoint\ (testN\ k) * post * testN\ k)\ N = post$
proof −
have $m \le N \Longrightarrow matrix\text{-}sum\ N\ (\lambda k.\ adjoint\ (testN\ k) * post * testN\ k)\ m = post\text{-}fst\text{-}k\ m$ **for** *m*
proof (*induct m*)
case *0*
then show *?case* **apply** *simp* **unfolding** *post-fst-k-def* **by** *auto*
next
case (*Suc m*)
then have *m*: $m \le N$ **by** *auto*
show *?case*
proof (*cases f m*)
case *True*
show *?thesis* **apply** *simp*
apply (*subst testN-post1*[*OF True*])
apply (*subst Suc*(*1*)[*OF m*])
using *post-fst-k-Suc True* **by** *auto*
next
case *False*
show *?thesis* **apply** *simp*
apply (*subst testN-post2*[*OF False*])
apply (*subst Suc*(*1*)[*OF m*])
using *post-fst-k-Suc-neg False post-fst-k-def* **by** *auto*
qed

```
  qed
  then show ?thesis using post-fst-kN by auto
qed

lemma tensor-P-testN-sum:
  matrix-sum d (λk. adjoint (tensor-P (testN k) (1_m K)) * tensor-P post (1_m K)
* tensor-P (testN k) (1_m K)) N =
   tensor-P post (1_m K)
proof -
  have eq: adjoint (tensor-P (testN k) (1_m K)) * tensor-P post (1_m K) * tensor-P
(testN k) (1_m K) =
            tensor-P (adjoint (testN k) * post * (testN k)) (1_m K) for k
    apply (subst tensor-P-adjoint-left-right)
    subgoal unfolding testN-def by auto
    subgoal by auto
    subgoal using post-dim by auto
    using adjoint-one by auto
  moreover have matrix-sum N (λk. adjoint (testN k) * post * testN k) N = post
    using testN-sum by auto
  show ?thesis unfolding eq
    apply (subst matrix-sum-tensor-P1)
    subgoal unfolding testN-def by auto
    subgoal by auto
    using testN-sum by auto
qed

lemma post-le-one:
  post ≤_L 1_m N
proof -
  let ?s = 1_m N - post
  have eq1: 1_m N * (1_m N - post) = 1_m N - post
    apply (mat-assoc N) using post-dim by auto
  have eq2: post * (1_m N - post) = 0_m N N
    apply (subst mult-minus-distrib-mat[of - N N])
    using post-dim by (auto simp add: post-mult-post)

  have adjoint ?s = ?s
    apply (subst adjoint-minus)
      apply (auto simp add: post-dim adjoint-dim)
    using adjoint-one hermitian-post[unfolded hermitian-def] by auto
  then have ?s * adjoint ?s = ?s * ?s by auto
  also have ... = 1_m N * (1_m N - post) - post * (1_m N - post)
    apply (mat-assoc N) using post-dim by auto
  also have ... = ?s unfolding eq1 eq2 using post-dim by auto
  finally have ?s * adjoint ?s = ?s.
  then have ∃M. M * adjoint M = ?s by auto
  then have positive ?s apply (subst positive-if-decomp[of ?s N]) using post-dim
by auto
  then show ?thesis unfolding lowner-le-def using post-dim by auto
```

qed

lemma *qp-post*:
is-quantum-predicate (*tensor-P post* (1_m *K*))
unfolding *is-quantum-predicate-def*
proof (*intro conjI*)
show *tensor-P post* (1_m *K*) ∈ *carrier-mat d d*
using *tensor-P-dim* **by** *auto*
show *positive* (*tensor-P post* (1_m *K*))
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive*)
by (*auto simp add*: *ps-P-d1 ps-P-d2 post-dim positive-post positive-one*)
show *tensor-P post* (1_m *K*) $\leq_L$ 1_m *d*
unfolding *ps-P.tensor-mat-id*[*symmetric*, *unfolded ps-P-d ps-P-d1 ps-P-d2*]
unfolding *ps2-P.ptensor-mat-def*
apply (*subst ps-P.tensor-mat-positive-le*)
unfolding *ps-P-d1 ps-P-d2* **using** *post-dim positive-post positive-one post-le-one lowner-le-refl*[*of* 1_m *K K*]
by *auto*
qed

lemma *hoare-triple-if*:
$\vdash_p$
{*tensor-P post* (1_m *K*)}
Measure-P vars1 N testN (*replicate N SKIP*)
{*tensor-P post* (1_m *K*)}
proof −
define *M* **where** *M* = (λ*n*. *mat-extension dims vars1* (*testN n*))
define *Post* **where** *Post* = (λ(*k*::*nat*). *tensor-P post* (1_m *K*))
have *M*: *M* = (λ*n*. *tensor-P* (*testN n*) (1_m *K*))
unfolding *M-def* **using** *mat-ext-vars1* **by** *auto*
have *skip*: ⋀*k*. *k* < *N* ⟹ (*replicate N SKIP*) ! *k* = *SKIP* **by** *simp*
have *h*: ⋀*k*. *k* < *N* ⟹ $\vdash_p$ {*Post k*} *replicate N SKIP* ! *k* {*tensor-P post* (1_m *K*)}
unfolding *Post-def skip* **using** *qp-post hoare-partial.intros* **by** *auto*
moreover have ⋀*k*. *k* < *N* ⟹ *is-quantum-predicate* (*Post k*) **unfolding** *Post-def* **using** *qp-post* **by** *auto*
ultimately show *?thesis*
unfolding *Measure-P-def* **apply** (*fold M-def*)
using *hoare-partial.intros*(*4*)[*of N Post tensor-P post* (1_m *K*) *replicate N SKIP M*]
unfolding *M Post-def* **using** *tensor-P-testN-sum qp-post* **by** *auto*
qed

theorem *grover-partial-deduct*:
$\vdash_p$
{*tensor-P pre* (*proj-k 0*)}
Grover
{*tensor-P post* (1_m *K*)}

unfolding *Grover-def*
proof −
have $\vdash_p$
{*tensor-P pre* (*proj-k 0*)}
hadamard-n n
{*adjoint exM0* ∗ *P′* ∗ *exM0* + *adjoint exM1* ∗ *Q* ∗ *exM1*}
using *hoare-partial.intros*(*6*)[*OF qp-pre qp-D3-post qp-pre qp-init-post*]
hoare-triple-init lowner-le-refl[*OF tensor-P-dim*] *lowner-le-Q* **by** *auto*
then have $\vdash_p$
{*tensor-P pre* (*proj-k 0*)}
hadamard-n n;;
While-P vars2 M0 M1 D
{*P′*}
using *hoare-triple-while hoare-partial.intros*(*3*) *qp-pre qp-D3-post qp-P′* **by** *auto*
then have $\vdash_p$
{*tensor-P pre* (*proj-k 0*)}
hadamard-n n;;
While-P vars2 M0 M1 D
{*tensor-P post* (1_m *K*)}
using *lowner-le-P′ hoare-partial.intros*(*6*)[*OF qp-pre qp-post qp-pre qp-P′*]
lowner-le-P′ lowner-le-refl[*OF tensor-P-dim*] **by** *auto*
then show $\vdash_p$
{*tensor-P pre* (*proj-k 0*)}
hadamard-n n;;
While-P vars2 M0 M1 D;;
Measure-P vars1 N testN (*replicate N SKIP*)
{*tensor-P post* (1_m *K*)}
using *hoare-triple-if qp-pre qp-post hoare-partial.intros*(*3*) **by** *auto*
qed

theorem *grover-partial-correct*:
$\models_p$
{*tensor-P pre* (*proj-k 0*)}
Grover
{*tensor-P post* (1_m *K*)}
using *grover-partial-deduct well-com-Grover qp-pre qp-post hoare-partial-sound*
by *auto*
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.