The CHSH inequality: Tsirelson's upper-bound and other results

Mnacho Echenim and Mehdi Mhalla and Coraline Mori

May 26, 2024

Abstract

The CHSH inequality, named after Clauser, Horne, Shimony and Holt, was used by Alain Aspect to prove experimentally that Einstein's hypothesis stating that quantum mechanics could be defined using local hidden variables was incorrect. The CHSH inequality is based on a setting in which an experiment consisting of two separate parties performing joint measurements is run several times, and a score is derived from these runs. If the local hidden variable hypothesis had been correct, this score would have been bounded by 2, but a suitable choice of observables in a quantum setting permits to violate this inequality when measuring the Bell state; this is the result that Aspect obtained experimentally. Tsirelson answered the question of how large this violation could be by proving that in the quantum setting, $2\sqrt{2}$ is the highest score that can be obtained when running this experiment. Along with elementary results on density matrices which represent quantum states in the finite dimensional setting, we formalize Tsirelson's result and summarize the main results on the CHSH score:

1. Under the local hidden variable hypothesis, this score admits 2 as an upper-bound.
2. When the density matrix under consideration is separable, the upper-bound cannot be violated.
3. When one of the parties in the experiment performs measures using commuting observables, this upper-bound remains valid.
4. Otherwise, the upper-bound of this score is $2\sqrt{2}$, regardless of the observables that are used and the quantum state that is measured, and
5. This upper-bound is reached for a suitable choice of observables when measuring the Bell state.

Contents

1 Basic algebraic results **2**

2 Results in linear algebra **10**

3 Results on tensor products **30**

4 Preliminary results **61**
4.1 Commutator and anticommutator 61

5 Maximum modulus in a spectrum **63**
5.1 Definition and basic properties for Hermitian matrices 63
5.2 Eigenvector for the element with maximum modulus 74

6 The $\mathcal{L}_2$ operator norm **76**
6.1 Definition and preliminary results 76
6.2 The $\mathcal{L}_2$ operator norm is equal to the maximum singular value 80
6.3 Consequences for the $\mathcal{L}_2$ operator norm 86

7 On density matrices **91**
7.1 Density matrix characterization 91
7.2 Separable density matrices 95
7.3 Characterization of pure states 99

8 Quantum expectation values and traces **103**

9 CHSH inequalities **107**
9.1 Some intermediate results for particular observables 107
9.2 The CHSH operator and expectation 109
9.3 CHSH inequality for separable density matrices 124
9.4 CHSH inequality for commuting observables 129
9.5 Result summary on the CHSH inequalities 131

theory *Tensor-Mat-Compl-Properties*
imports
Commuting-Hermitian.Spectral-Theory-Complements
Projective-Measurements.Projective-Measurements
begin

1 Basic algebraic results

lemma *pos-sum-gt-0*:
assumes *finite I*
and $\bigwedge i.\ i \in I \Longrightarrow (0 :: 'a :: linordered\text{-}field) \leq f\ i$
and $0 < sum\ f\ I$
shows $\exists j \in I.\ 0 < f\ j$
proof (*rule ccontr*)
assume $\neg\ (\exists j \in I.\ 0 < f\ j)$

```
  hence ∀ j ∈ I. f j ≤ 0 by auto
  hence ∀ j ∈ I. f j = 0 using assms by fastforce
  hence sum f I = 0 by simp
  thus False using assms by simp
qed

lemma pos-square-1-elem:
  assumes finite I
and ⋀i. i ∈ I ⟹ (0::real) ≤ f i
and sum f I = 1
and sum (λx. f x * f x) I = 1
shows ∃ j ∈ I. f j = 1
proof (rule ccontr)
  assume ¬ (∃ j∈I. f j = 1)
  hence ne: ∀ j∈ I. f j ≠ 1 by simp
  have ∃ j ∈ I. 0 < f j using pos-sum-gt-0[of I f] assms by simp
  from this obtain j where j∈I and 0 < f j by auto
  hence f j ≠ 1 using ne by simp
  moreover have f j ≤ 1 using ‹j∈ I› assms pos-sum-le-comp by force
  ultimately have f j < 1 by auto
  have sum (λx. f x * f x) I = f j * f j + sum (λx. f x * f x) (I−{j})
    by (meson ‹j ∈ I› assms(1) sum.remove)
  also have ... < f j + sum (λx. f x * f x) (I−{j})
    by (simp add: ‹0 < f j› ‹f j < 1›)
  also have ... ≤ f j + sum f (I−{j})
    using square-pos-mult-le[of I −  {j}]
    by (smt (verit, ccfv-SIG) DiffD1 assms
        mult-left-le sum-mono sum-nonneg-leq-bound)
  also have ... = sum f I
    by (metis ‹j ∈ I› assms(1) sum.remove)
  also have ... = 1 using assms by simp
  finally have sum (λx. f x * f x) I < 1 .
  thus False using assms by simp
qed

lemma cpx-pos-square-1-elem:
  assumes finite I
and ⋀i. i ∈ I ⟹ (0::complex) ≤ f i
and sum f I = 1
and sum (λx. f x * f x) I = 1
shows ∃ j ∈ I. f j = 1
proof −
  have ∀ i∈ I. Im( f i) = 0 using assms complex-is-Real-iff
    by (meson  nonnegative-complex-is-real
        positive-unitary-diag-pos real-diag-decompD(1))
  hence al: ∀ i ∈ I. Re(f i) = f i
    by (simp add: assms complex.expand)
  have ∃  j ∈ I. Re (f j) = 1
  proof (rule pos-square-1-elem)
```

```
    show finite I using assms by simp
    show ⋀i. i ∈ I ⟹ 0 ≤ Re (f i) using assms al
      by (simp add: less-eq-complex-def)
    show (∑ j∈I. Re (f j)) = 1 using al
      by (metis Re-sum assms(3) one-complex.simps(1))
    show (∑ x∈I. Re (f x) * Re (f x)) = 1 using al
      by (smt (z3) assms(4) of-real-hom.hom-1 of-real-hom.hom-mult
          of-real-hom.hom-sum sum.cong)
  qed
  thus ?thesis using al by force
qed

lemma sum-eq-elmt:
  assumes finite I
  and ⋀i. i ∈ I ⟹ (0::'a :: linordered-field) ≤ f i
  and sum f I = c
  and j∈I
  and f j = c
shows ∀ k∈(I−{j}). f k = 0
proof −
  have sum f I − f j= sum f (I − {j}) using assms sum-diff1[of I f j] by auto
  also have sum f I − f j = 0 using assms
    using ‹f j = c› by linarith
  hence sum f (I − {j}) = 0 using assms
    using calculation by linarith
  finally show ∀ k∈(I−{j}). f k = 0
     by (meson DiffD1 ‹sum f (I − {j}) = 0› assms(1) assms(2) finite-Diff
sum-nonneg-eq-0-iff)
qed

lemma cpx-sum-eq-elmt:
  assumes finite I
  and ⋀i. i ∈ I ⟹ (0::complex) ≤ f i
  and sum f I = c
  and j∈I
  and f j = c
shows ∀ k∈(I−{j}). f k = 0
proof −
  have sum f I − f j= sum f (I − {j}) using assms sum-diff1[of I f j] by auto
  also have sum f I − f j = 0 using assms
    using ‹f j = c› by simp
  hence sum f (I − {j}) = 0 using assms
    using calculation by simp
  finally show ∀ k∈(I−{j}). f k = 0
    by (meson DiffD1 ‹sum f (I − {j}) = 0› assms(1) assms(2)
        finite-Diff sum-nonneg-eq-0-iff)
qed

lemma sum-nat-div-mod:
```

```
  shows sum (λi. sum (λj. f i * g j) {..< (m::nat)}) {..< (n::nat)} =
    sum (λk. f (k div m) * g (k mod m)) {..< n*m}
proof (induct n)
  case 0
  then show ?case by simp
next
  case (Suc n)
  have (∑ i<Suc n. ∑ j<m. f i * g j) = (∑ i< n. ∑ j<m. f i * g j) +
    (∑ j<m. f n * g j)
    by simp
  also have ... = (∑ k<n * m. f (k div m) * g (k mod m)) +
    (∑ j<m. f n * g j)
    using Suc by simp
  also have ... = (∑ k<n * m. f (k div m) * g (k mod m)) +
    sum (λk. f (k div m) * g (k mod m)) {n*m ..< (Suc n) * m}
  proof −
    have (∑ j<m. f n * g j) =
      sum (λk. f (k div m) * g (k mod m)) {n*m ..< (Suc n) * m}
    proof (rule sum.reindex-cong)
      show inj-on (λj. j mod m) {n * m..<Suc n * m}
      proof
        fix x y
        assume x ∈ {n * m..<Suc n * m} and y ∈ {n * m..<Suc n * m}
          and x mod m = y mod m
        thus x = y
          by (metis atLeastLessThan-iff div-nat-eqI mod-div-decomp
              mult.commute)
      qed
      show {..<m} = (λj. j mod m) ` {n * m..<Suc n * m}
      proof
        show {..<m} ⊆ (λj. j mod m) ` {n * m..<Suc n * m}
        proof
          fix x
          assume x∈ {..< m}
          hence n * m + x∈ {n * m..<Suc n * m} by simp
          moreover have x = (n * m + x) mod m using ‹x ∈ {..<m}› by auto
          ultimately show x ∈ (λj. j mod m) ` {n * m..<Suc n * m}
            using ‹x ∈ {..<m}› by blast
        qed
      qed auto
      fix x
      assume x ∈ {n * m..<Suc n * m}
      thus f n * g (x mod m) = f (x div m) * g (x mod m) by auto
    qed
    thus ?thesis by simp
  qed
  also have ... = sum (λk. f (k div m) * g (k mod m))
    ({..<n*m} ∪ {n * m..<Suc n * m})
    by (rule sum.union-disjoint[symmetric], auto)
```

```
  also have ... = (∑k<(Suc n) * m. f (k div m) * g (k mod m))
  proof −
    have {..<n*m} ∪ {n * m..<Suc n * m} = {..< Suc n * m}
      by (simp add: ivl-disj-un-one(2))
    thus ?thesis by simp
  qed
  finally show ?case .
qed

lemma abs-cmod-eq:
  fixes z::complex
  shows |z| = cmod z
  by (simp add: abs-complex-def)

lemma real-cpx-abs-leq:
  fixes A::complex
  assumes A∈ Reals
  and B∈ Reals
  and |A * B| ≤ 1
shows |Re A * Re B| ≤ 1
proof −
  have |Re A * Re B| = |A * B|  using assms
     by (metis Reals-mult abs-cmod-eq in-Reals-norm real-mult-re)
  also have ... ≤ 1 using assms by simp
  finally show |Re A * Re B| ≤ 1
    by (metis Re-complex-of-real less-eq-complex-def one-complex.sel(1))
qed

lemma cpx-real-abs-eq:
  fixes z::complex and r::real
  assumes z∈ Reals
  and z = r
shows |z| = |r|
proof −
  have Re z = r using assms by simp
  have Im z = 0  using assms complex-is-Real-iff by auto
  have |z| = cmod z by (simp add: abs-complex-def)
  hence |z| = |Re z| using ‹Im z = 0› assms by simp
  thus ?thesis using ‹Re z = r› by simp
qed

lemma cpx-real-abs-leq:
  fixes z::complex and r::real
  assumes z∈ Reals
  and z = r
  and |r| ≤ k
shows |z| ≤ (k::real)
proof −
  have Re z = r using assms by simp
```

hence $|Re\ z| \leq k$ **using** *assms* **by** *simp*
have $Im\ z = 0$ **using** *assms complex-is-Real-iff* **by** *auto*
have $|z| = cmod\ z$ **by** (*simp add*: *abs-complex-def*)
hence $|z| = |Re\ z|$ **using** ‹$Im\ z = 0$› *assms* **by** *simp*
thus *?thesis* **using** ‹$|Re\ z| \leq k$› **by** (*simp add*: *less-eq-complex-def*)
qed

lemma *cpx-abs-mult-le-1*:
fixes z::*complex*
assumes $|z| \leq 1$
and $|z'| \leq 1$
shows $|z*z'| \leq 1$
proof −
have *a*: $cmod\ z \leq 1$
by (*metis Reals-1 abs-1 abs-cmod-eq assms*(*1*)
cpx-real-abs-leq dual-order.antisym linorder-le-cases
of-real-eq-1-iff)
have *b*: $cmod\ z' \leq 1$
by (*metis Reals-1 abs-1 abs-cmod-eq assms*(*2*)
cpx-real-abs-leq dual-order.antisym linorder-le-cases
of-real-eq-1-iff)
have $|z*z'| = |z|*|z'|$
by (*simp add*: *abs-mult*)
also have ... $= cmod\ z * (cmod\ z')$
using *abs-cmod-eq* **by** *auto*
also have ... ≤ 1 **using** *a b*
by (*simp add*: *less-eq-complex-def mult-le-one*)
finally show *?thesis* **.**
qed

lemma *sum-abs-cpx*:
shows $|sum\ K\ I| \leq sum\ (\lambda x.\ |(K\ x)::complex|)\ I$
proof −
have $|sum\ K\ I| = cmod\ (sum\ K\ I)$
using *abs-cmod-eq* **by** *blast*
also have ... $\leq sum\ (\lambda x.\ cmod\ (K\ x))\ I$ **using** *norm-sum*
by (*metis Im-complex-of-real Re-complex-of-real less-eq-complex-def*)
also have ... $= sum\ (\lambda x.\ |(K\ x)::complex|)\ I$
using *abs-cmod-eq* **by** *fastforce*
finally show *?thesis* **.**
qed

lemma *abs-mult-cpx*:
fixes z::*complex*
assumes $0 \leq (a::real)$
shows $|a*z| = a * |z|$
proof −
have $|a*z| = cmod\ (a*z)$ **using** *abs-cmod-eq* **by** *blast*
also have ... $= a * cmod\ z$ **using** *assms*

```
    by (simp add: norm-mult)
  also have ... = a * |z| by (simp add: abs-cmod-eq)
  finally show ?thesis .
qed

lemma cpx-ge-0-real:
  fixes c::complex
  assumes 0 ≤ c
  and c∈ Reals
shows 0 ≤ Re c
proof −
  have Re c = c using assms by simp
  hence 0 ≤ complex-of-real (Re (c::complex)) using assms by simp
  thus ?thesis using less-eq-complex-def by auto
qed

lemma cpx-of-real-ge-0:
  assumes 0 ≤ complex-of-real a
  shows 0 ≤ a
proof −
  have 0 ≤ Re (complex-of-real a)
    using Reals-of-real assms cpx-ge-0-real by blast
  also have ... = a by simp
  finally show ?thesis .
qed


lemma set-cst-list:
  shows (⋀i. i < length l ⟹ l!i = x) ⟹ 0 < length l ⟹ set l = {x}
proof (induct l)
  case Nil
  then show ?case by simp
next
  case (Cons a l)
  then show ?case
    by (metis in-set-conv-nth insert-absorb is-singletonI'
        is-singleton-def singleton-iff)
qed

lemma pos-mult-Max:
  assumes finite F
and F ≠ {}
and 0 ≤ x
and ∀ a∈ F. 0 ≤ (a::real)
shows Max.F {x * a|a. a ∈ F} = x * Max.F F
proof −
  define M where M = Max.F F
  have finite {x * a|a. a ∈ F} using assms by auto
  have M∈ F using assms unfolding M-def by simp
```

hence $x{*}M \in \{x * a|a.\ a \in F\}$ **by** *auto*
moreover have $\forall\, c{\in}\{x * a|a.\ a \in F\}.\ c \leq x{*}M$
using *M-def assms eq-Max-iff*
ordered-comm-semiring-class.comm-mult-left-mono **by** *fastforce*
ultimately show *?thesis* **using** *assms Max-eqI M-def* ‹*finite* $\{x * a\ |a.\ a \in F\}$›

by *blast*
qed

lemma *square-Max*:
assumes *finite A*
and $A{\neq}\ \{\}$
and $\forall\, a{\in}\ A.\ 0 \leq ((f\ a){::}real)$
and $b = Max.F\ \{f\ a\ |a.\ a{\in}\ A\}$
shows $Max.F\ \{f\ a{*}\ f\ a|a.\ a{\in}\ A\} = b * b$
proof −
define B **where** $B = \{f\ a{*}\ f\ a|a.\ a{\in}\ A\}$
have *finite B* **using** *finite-image-set* **unfolding** *B-def* **by** (*simp add*: *assms*)
have *finite* $\{f\ a\ |a.\ a{\in}\ A\}$ **using** *assms* **by** *auto*
hence $b{\in}\ \{f\ a\ |a.\ a{\in}\ A\}$ **using** *assms*
by (*metis* (*mono-tags*, *lifting*) *Collect-empty-eq-bot Max-eq-iff all-not-in-conv bot-empty-eq*)
hence $b{*}b \in B$ **unfolding** *B-def* **by** *auto*
moreover have $\forall\, c{\in}B.\ c \leq b{*}b$
proof
fix *c*
assume $c{\in}\ B$
hence $\exists\, d{\in}\ A.\ c = f\ d{*}\ f\ d$ **unfolding** *B-def* **by** *auto*
from *this* **obtain** *d* **where** $d{\in}\ A$ **and** $c = f\ d * f\ d$ **by** *auto*
note *dprop* = *this*
hence $f\ d \in \{f\ a\ |a.\ a{\in}\ A\}$ **by** *auto*
hence $f\ d \leq b$ **using** *assms* **by** *auto*
thus $c \leq b{*}b$ **using** *assms* **by** (*simp add*: *dprop mult-mono′*)
qed
ultimately show *?thesis* **using** *assms Max-eqI*[*of B b*∗*b*] ‹*finite B*›
by (*metis B-def*)
qed

lemma *ereal-Sup-switch*:
assumes $\forall\ \ m{\in}\ P.\ (b{::}real) \leq f\ m$
and $\forall\, m \in P.\ f\ m \leq (c{::}real)$
and $P \neq \{\}$
shows $ereal\ (Sup\ (f\ `\ P)) = (\bigsqcup m{\in}\ P.\ ereal\ (f\ m))$
proof (*rule ereal-SUP*)
have b: $\forall\, m \in P.\ b \leq (ereal\ (f\ m))$ **using** *assms* **by** *auto*
hence $b \leq (\bigsqcup\ m{\in}\ P.\ ereal\ (f\ m))$ **using** *assms*
by (*meson Sup-upper2 ex-in-conv image-eqI*)
have m: $\forall\, m \in P.\ (ereal\ (f\ m)) \leq c$ **using** *assms* **by** *auto*
hence c: $Sup\ (ereal\ `\ (f`\ P)) \leq c$

by (*simp add*: *assms*(*3*) *cSUP-least image-image*)
show |⨆ *m*∈ *P*. *ereal* (*f m*)| ≠ ∞ **using** *b c MInfty-neq-ereal*(*2*)
by (*metis PInfty-neq-ereal*(*1*) *m b*
assms(*3*) *ereal-SUP-not-infty*)
qed

lemma *Sup-ge-real*:
assumes *a*∈ (*A*::*real set*)
and ∀ *a* ∈ *A*. *a* ≤ *c*
and ∀ *a* ∈ *A*. *b* ≤ *a*
shows *a* ≤ *Sup A*
proof −
define *B* **where** *B* = {*ereal a*|*a*. *a*∈ *A*}
have *ereal a* ∈ *B* **using** *assms* **unfolding** *B-def* **by** *simp*
hence *ereal a* ≤ *Sup B* **by** (*simp add*: *Sup-upper*)
also have ... = *ereal* (*Sup A*)
using *ereal-Sup-switch*[*symmetric*, *of A b* λ*x*. *x c*] *assms* **unfolding** *B-def*
by (*metis B-def Collect-mem-eq empty-iff image-Collect image-ident*)
finally have *ereal a* ≤ *ereal* (*Sup A*) **.**
thus *?thesis* **by** *simp*
qed

lemma *Sup-real-le*:
assumes ∀ *a* ∈ (*A*::*real set*). *a* ≤ *c*
and ∀ *a* ∈ *A*. *b* ≤ *a*
and *A*≠ {}
shows *Sup A* ≤ *c*
proof −
define *B* **where** *B* = {*ereal a*|*a*. *a*∈ *A*}
have *Sup B* ≤ *ereal c* **unfolding** *B-def* **using** *SUP-least*[*of A* λ*x*. *x c*] *assms*
by (*simp add*: *Setcompr-eq-image*)
moreover have *Sup B* = *ereal* (*Sup A*) **unfolding** *B-def*
using *ereal-Sup-switch*[*symmetric*, *of A b* λ*x*. *x c*] *assms*
by (*metis B-def Collect-mem-eq image-Collect image-ident*)
ultimately show *?thesis* **by** *simp*
qed

2 Results in linear algebra

lemma *mat-add-eq-0-if*:
fixes *A*::′*a* ::*group-add Matrix.mat*
assumes *A*∈ *carrier-mat n m*
and *B*∈ *carrier-mat n m*
and *A*+*B* = 0_m *n m*
shows *B* = −*A*
proof (*rule eq-matI*)
show *dim-row B* = *dim-row* (−*A*) **using** *assms* **by** *simp*
show *dim-col B* = *dim-col* (−*A*) **using** *assms* **by** *simp*
fix *i j*

assume $i < \textit{dim-row}\ (-A)$ **and** $j < \textit{dim-col}\ (-A)$ **note** *ij= this*
hence $i < \textit{dim-row}\ B\ j < \textit{dim-col}\ B$
using ‹*dim-row B = dim-row* $(-A)$› ‹*dim-col B = dim-col* $(-A)$› **by** *auto*
hence *A $$ (i,j) + B $$ (i,j) = (A+B)$$(i,j)* **using** *ij* **by** *simp*
also have ... = *0*
by (*metis* ‹*dim-col B = dim-col* $(-A)$› ‹*dim-row B = dim-row* $(-A)$›
assms(*2*) *assms*(*3*) *carrier-matD*(*1*) *ij*(*1*) *ij*(*2*) *index-add-mat*(*3*)
index-zero-mat(*1*) *index-zero-mat*(*3*))
finally have *A $$ (i,j) + B $$ (i,j) = 0* **.**
thus *B $$ (i, j) = (− A) $$ (i, j)*
by (*metis* ‹*dim-col B = dim-col* $(-A)$› ‹*dim-row B = dim-row* $(-A)$›
‹*i < dim-row B*› ‹*j < dim-col B*› *add-eq-0-iff index-uminus-mat*(*1*)
index-uminus-mat(*2*) *index-uminus-mat*(*3*))
qed

lemma *trace-rank-1-proj*:
shows *Complex-Matrix.trace* (*rank-1-proj v*) = $\|v\|^2$
proof −
have *Complex-Matrix.trace* (*rank-1-proj v*) = *inner-prod v v*
using *trace-outer-prod carrier-vecI*
unfolding *rank-1-proj-def* **by** *blast*
also have ... = (*vec-norm v*)^2
unfolding *vec-norm-def* **using** *power2-csqrt* **by** *presburger*
also have ... = $\|v\|^2$ **using** *vec-norm-sq-cpx-vec-length-sq* **by** *simp*
finally show *?thesis* **.**
qed

lemma *trace-ch-expand*:
fixes *A*::*'a*::{*minus*,*comm-ring*} *Matrix.mat*
assumes *A*∈ *carrier-mat n n*
and *B*∈ *carrier-mat n n*
and *C*∈ *carrier-mat n n*
and *D*∈ *carrier-mat n n*
shows *Complex-Matrix.trace* (*A − B + C + D*) =
Complex-Matrix.trace A − Complex-Matrix.trace B +
Complex-Matrix.trace C + Complex-Matrix.trace D
proof −
have *Complex-Matrix.trace* (*A − B + C + D*) =
Complex-Matrix.trace (*A − B + C*) + *Complex-Matrix.trace D*
using *trace-add-linear*[*of - n D*] *assms* **by** *simp*
also have ... = *Complex-Matrix.trace* (*A − B*) + *Complex-Matrix.trace C +*
Complex-Matrix.trace D **using** *assms trace-add-linear*[*of - n C*]
by (*metis minus-carrier-mat'*)
finally show *?thesis* **using** *assms trace-minus-linear* **by** *auto*
qed

lemma *squared-A-trace*:
assumes *A*∈ *carrier-mat n n*
and *unitarily-equiv A B U*

shows *Complex-Matrix.trace* $(A*A)$ = *Complex-Matrix.trace* $(B*B)$
proof (*rule unitarily-equiv-trace*)
  show $A*A \in$ *carrier-mat n n* **using** *assms* **by** *simp*
  show *unitarily-equiv* $(A * A)$ $(B * B)$ U
    using *assms unitarily-equiv-square*[*of A n*] **by** *simp*
qed

lemma *squared-A-trace'*:
assumes $A\in$ *carrier-mat n n*
  and *unitary-diag A B U*
shows *Complex-Matrix.trace* $(A*A) = (\sum\ i \in \{0\ ..<\ n\}.\ (B\ \$\$\ (i,i) * B\ \$\$\ (i,i)))$
proof −
  have *Complex-Matrix.trace* $(A*A)$ = *Complex-Matrix.trace* $(B*B)$
    using *assms squared-A-trace*[*of A*]
    by (*meson unitary-diag-imp-unitarily-equiv*)
  also have ... $= (\sum\ i \in \{0\ ..<\ n\}.\ (B * B)\ \$\$\ (i,i))$ **using** *assms*
    unfolding *Complex-Matrix.trace-def*
    by (*metis* (*mono-tags*, *lifting*) *carrier-matD*(*1*) *index-mult-mat*(*2*)
      unitary-diag-carrier(*1*))
  also have ... $= (\sum\ i \in \{0\ ..<\ n\}.\ (B\ \$\$\ (i,i) * B\ \$\$\ (i,i)))$
  proof (*rule sum.cong*)
    fix i
    assume $i \in \{0..<n\}$
    hence $i < n$ **by** *simp*
    thus $(B*B)\ \$\$\ (i,i) = B\ \$\$(i,i) * B\$\$(i,i)$ **using** *diagonal-mat-sq-index*
      by (*metis assms*(*1*) *assms*(*2*) *unitary-diag-carrier*(*1*)
        unitary-diag-diagonal)
  qed *simp*
  finally show *?thesis* **.**
qed

lemma *positive-square-trace*:
  assumes $A \in$ *carrier-mat n n*
  and *Complex-Matrix.trace A* = (*1*::*real*)
  and *Complex-Matrix.trace* $(A*A)$ = *1*
  and *real-diag-decomp A B U*
  and *Complex-Matrix.positive A*
  and $0 < n$
shows $\exists j<n.\ B\ \$\$\ (j,j) = 1 \wedge (\forall\, i<n.\ i\neq j \longrightarrow B\ \$\$\ (i,i) = 0)$
proof −
  have *b*: $\forall\, i<n.\ 0 \leq B\ \$\$\ (i,\ i)$ **using** *assms positive-unitary-diag-pos*
    by (*meson* ‹*real-diag-decomp A B U*› *real-diag-decompD*(*1*))
  also have *t*: *Complex-Matrix.trace B* = (*1*::*real*)
    using *assms*
    by (*metis* ‹*real-diag-decomp A B U*› *of-real-1 real-diag-decompD*(*1*)
      unitarily-equiv-trace unitary-diag-imp-unitarily-equiv)
  have *t-sq*: $(\sum i\in\{0..<n\}.\ (B\ \$\$\ (i,i) * B\ \$\$\ (i,i))) = 1$
    using *assms unitary-diag-carrier squared-A-trace'*

```
    by (smt (verit, ccfv-SIG) ‹real-diag-decomp A B U› real-diag-decompD(1)
sum.cong)
  have dim-n: dim-row B = n using assms
    by (meson ‹real-diag-decomp A B U› carrier-matD(1)
        real-diag-decompD(1) unitary-diag-carrier(1))
  have ex-j: ∃j∈{0..<n}. (B $$ (j, j)) = 1
  proof (rule cpx-pos-square-1-elem)
    show finite {0..<n} by simp
    show ⋀i. i ∈ {0..<n} ⟹ 0 ≤ B $$ (i, i) using b by simp
    show (∑j ∈ {0..<n}. B $$ (j, j)) = 1 using t
      unfolding Complex-Matrix.trace-def
      by (metis ‹dim-row B = n› of-real-hom.hom-one)
    show (∑x = 0..<n. B $$ (x, x) * B $$ (x, x)) = 1 using t-sq
      by blast
  qed
  from this obtain j where jn: j∈{0..<n} and bj: B $$ (j, j) = 1 by auto
  have ∀k ∈ ({0..<n}−{j}). B $$ (k, k) = 0
  proof (rule cpx-sum-eq-elmt)
    show finite {0..<n} by simp
    show ⋀i. i ∈ {0..<n} ⟹ 0 ≤ B $$ (i, i) using b by simp
    show (∑k = 0..<n. B $$ (k, k)) = 1 using t
      unfolding Complex-Matrix.trace-def
      by (simp add: dim-n)
    show j ∈ {0..<n} using jn by simp
    show B $$ (j, j) = 1 using bj by simp
  qed
  hence ∀i<n. i ≠ j ⟶ B $$ (i, i) = 0
    using atLeastLessThan-iff by blast
  thus ?thesis
    by (metis atLeastLessThan-iff bj jn)
qed

lemma idty-square:
  shows ((1_m n):: 'a :: semiring-1 Matrix.mat) * (1_m n) = 1_m n
  using right-mult-one-mat by simp

lemma pos-hermitian-trace-reals:
  fixes A::complex Matrix.mat
  assumes A∈ carrier-mat n n
  and B∈ carrier-mat n n
  and 0 < n
  and Complex-Matrix.positive A
  and hermitian B
  shows Complex-Matrix.trace (B*A) ∈ Reals
proof −
  define fc::complex Matrix.mat set where fc = carrier-mat n n
  interpret cpx-sq-mat n n fc
  proof
    show 0 < n using assms by simp
```

```
  qed (auto simp add: fc-def)
  have Complex-Matrix.trace (B*A) = Complex-Matrix.trace (A*B) using assms
    by (metis trace-comm)
  also have ... = Re (Complex-Matrix.trace (A * B))
  proof (rule trace-hermitian-pos-real[of B A])
    show hermitian B using assms by simp
    show A∈ fc using assms unfolding fc-def by simp
    show B∈ fc using assms unfolding fc-def by simp
    show Complex-Matrix.positive A using assms by simp
  qed
  finally have Complex-Matrix.trace (B*A) =
    Re (Complex-Matrix.trace (A * B)) .
  thus ?thesis by (metis Reals-of-real)
qed

lemma pos-hermitian-trace-reals':
  fixes A::complex Matrix.mat
  assumes A∈ carrier-mat n n
  and B∈ carrier-mat n n
  and 0 < n
  and Complex-Matrix.positive A
  and hermitian B
  shows Complex-Matrix.trace (A*B) ∈ Reals
  by (metis assms pos-hermitian-trace-reals trace-comm)

lemma hermitian-commute:
  assumes hermitian A
  and hermitian B
  and A*B = B*A
shows hermitian (A*B)
  by (metis adjoint-mult assms hermitian-def
      hermitian-square index-mult-mat(2))


lemma idty-unitary-diag:
  assumes unitary-diag (1_m n) B U
  shows B = 1_m n
proof −
  have l: (Complex-Matrix.adjoint U) * U = 1_m n
    using assms one-carrier-mat similar-mat-witD2(2) unitary-diagD(1) by blast
  have r: (Complex-Matrix.adjoint U) * U = 1_m n
    by (simp add: l)
  hence B = ((Complex-Matrix.adjoint U) * U) * B *
    ((Complex-Matrix.adjoint U) * U) using l r
    by (metis assms index-one-mat(2) left-mult-one-mat' right-mult-one-mat
        similar-mat-witD(5) similar-mat-wit-dim-row unitary-diagD(1))
  also have ... = (Complex-Matrix.adjoint U) *
    (U * B * (Complex-Matrix.adjoint U)) * U
    by (metis assms calculation similar-mat-witD(3) similar-mat-wit-sym
```

```
      unitary-diagD(1))
  also have ... = (Complex-Matrix.adjoint U) * (1_m n) * U
    by (metis assms one-carrier-mat similar-mat-witD2(3) unitary-diagD(1))
  also have ... = 1_m n
    by (metis assms index-one-mat(2) l right-mult-one-mat similar-mat-witD(7)
      unitary-diagD(1))
  finally show ?thesis .
qed

lemma diag-mat-idty:
  assumes 0 < n
  shows set (diag-mat ((1_m n)::'a::{one,zero} Matrix.mat)) = {1}
    (is ?L = ?R)
proof
  show ?L ⊆ ?R
  proof
    fix x::'a
    assume x ∈ set (diag-mat (1_m n))
    hence ∃ i < length (diag-mat (1_m n)). nth (diag-mat (1_m n))  i = x
      using in-set-conv-nth[of x diag-mat (1_m n)] assms by simp
    from this obtain i where i < length (diag-mat (1_m n))
      and nth (diag-mat (1_m n))  i = x
      by auto note iprop = this
    hence i < dim-row (1_m n) unfolding diag-mat-def by simp
    hence i < n using assms by simp
    have x = (1_m n)$$(i,i) using iprop unfolding diag-mat-def by simp
    thus x ∈ ?R using ‹i < n› by simp
  qed
next
  show ?R ⊆ ?L
  proof
    fix x
    assume x∈ ?R
    hence x = 1 by simp
    also have ... = (1_m n)$$(0,0) using assms by simp
    also have ... ∈ ?L using assms unfolding diag-mat-def by simp
    finally show x∈ ?L .
  qed
qed

lemma idty-spectrum:
assumes 0 < n
shows spectrum ((1_m n)::complex Matrix.mat) = {1}
proof −
  have spectrum ((1_m n)::complex Matrix.mat) = set (diag-mat (1_m n))
    using similar-spectrum-eq
    by (meson one-carrier-mat similar-mat-refl upper-triangular-one)
  also have ... = {1} using diag-mat-idty assms by simp
  finally show ?thesis .
```

qed

lemma *spectrum-ne*:
fixes *A*::*complex Matrix.mat*
assumes *A* ∈ *carrier-mat n n*
and *0* < *n*
shows *spectrum A* ≠ {} **unfolding** *spectrum-def*
using *eigvals-poly-length*[*of A*] *assms* **by** *auto*

lemma *unitary-diag-square-spectrum*:
fixes *A*::*complex Matrix.mat*
assumes *hermitian A*
and *A*∈ *carrier-mat n n*
and *unitary-diag A B U*
shows *spectrum* (*A*∗*A*) = *set* (*diag-mat* (*B*∗*B*))
proof −
have *sa*: *similar-mat* (*A*∗*A*) (*B*∗*B*)
using *assms hermitian-square-similar-mat-wit*[*of A n*]
unfolding *similar-mat-def* **by** *auto*
have *diagonal-mat* (*B*∗*B*) **using** *diagonal-mat-sq-diag*[*of B*] *assms*
by (*meson unitary-diag-carrier*(*1*) *unitary-diag-diagonal*)
have (∏ *a*←*eigvals* (*A*∗*A*). [:− *a*, *1*:]) = *char-poly* (*A*∗*A*) **using** *assms*
by (*metis eigvals-poly-length mult-carrier-mat*)
also have ... = *char-poly* (*B*∗*B*) **using** *char-poly-similar*[*OF sa*] **by** *simp*
also have ... = (∏ *a*←*diag-mat* (*B*∗*B*). [:− *a*, *1*:]) **using**
‹*diagonal-mat* (*B*∗*B*)›
by (*metis assms*(*2*) *assms*(*3*) *char-poly-upper-triangular*
diagonal-imp-upper-triangular mult-carrier-mat
unitary-diag-carrier(*1*))
finally have (∏ *a*←*eigvals* (*A*∗*A*). [:− *a*, *1*:]) =
(∏ *a*←*diag-mat* (*B*∗*B*). [:− *a*, *1*:]) **.**
hence *set* (*eigvals* (*A*∗*A*)) = *set* (*diag-mat* (*B*∗*B*))
using *poly-root-set-eq*[*of eigvals* (*A*∗*A*)] **by** *simp*
thus *?thesis* **unfolding** *spectrum-def* **by** *simp*
qed

lemma *diag-mat-square-eq*:
fixes *B*::*'a*::{*ring*} *Matrix.mat*
assumes *diagonal-mat B*
and *B* ∈ *carrier-mat n n*
shows *set* (*diag-mat* (*B*∗*B*)) = {*b*∗*b*|*b*. *b*∈ *set* (*diag-mat B*)}
proof
show *set* (*diag-mat* (*B* ∗ *B*)) ⊆ {*b*∗*b* |*b*. *b* ∈ *set* (*diag-mat B*)}
proof
fix *x*
assume *x* ∈ *set* (*diag-mat* (*B* ∗ *B*))
hence ∃ *i* < *length* (*diag-mat* (*B* ∗ *B*)). *nth* (*diag-mat* (*B* ∗ *B*)) *i* = *x*
using *in-set-conv-nth*[*of x*] **by** *simp*
from *this* **obtain** *i* **where** *i* < *length* (*diag-mat* (*B* ∗ *B*))

```
    and nth (diag-mat (B * B))  i = x
    by auto note iprop = this
  hence i < n using assms unfolding diag-mat-def by simp
  have (B*B) $$ (i,i) = x using iprop
    unfolding diag-mat-def by simp
  hence B $$ (i,i)* B $$ (i,i) = x
    using diagonal-mat-sq-index[of B n i i] assms iprop ‹i < n›
    by simp
  moreover have B $$ (i,i) ∈ set (diag-mat B)
    using ‹i < n› assms in-set-conv-nth[of x]
    unfolding diag-mat-def by auto
  ultimately show x ∈ {b*b |b. b ∈ set (diag-mat B)} by auto
 qed
next
 show {b * b |b. b ∈ set (diag-mat B)} ⊆ set (diag-mat (B * B))
 proof
  fix x
  assume x ∈ {b * b |b. b ∈ set (diag-mat B)}
  hence ∃ b∈ set (diag-mat B). x = b * b by auto
  from this obtain b where b∈ set (diag-mat B) and x = b * b by auto
  hence ∃ i < length (diag-mat B). (diag-mat B)!i = b
    using in-set-conv-nth[of b] by simp
  from this obtain i where i < length (diag-mat B)
    and (diag-mat B) ! i = b by auto
  note iprop = this
  hence B $$ (i,i) = b unfolding diag-mat-def by simp
  moreover have i < n using assms iprop unfolding diag-mat-def by simp
  ultimately have (B*B) $$ (i,i) = x
    using ‹x = b*b› diagonal-mat-sq-index[of B n i i] assms iprop by simp
  hence x = (diag-mat (B*B)) ! i using ‹i < n› assms
    unfolding diag-mat-def by fastforce
  moreover have i<length (diag-mat (B * B))
    using ‹i < n› assms unfolding diag-mat-def by auto
  ultimately show x ∈ set (diag-mat (B * B))
    using in-set-conv-nth[of x diag-mat (B*B)]
    by simp
 qed
qed

lemma hermitian-square-spectrum-eq:
 fixes A::complex Matrix.mat
 assumes hermitian A
and A∈ carrier-mat n n
and 0 < n
shows spectrum (A*A) = {a*a | a. a∈ spectrum A}
proof −
 obtain B U where herm: real-diag-decomp A B U
   using hermitian-real-diag-decomp[of A] assms by auto
 hence spectrum (A*A) = set (diag-mat (B*B))
```

```
    using unitary-diag-square-spectrum assms real-diag-decompD(1) by blast
  also have ... = {a*a|a. a∈ set (diag-mat B)}
    using diag-mat-square-eq[of B] assms herm
    by (meson real-diag-decompD(1) unitary-diagD(2) unitary-diag-carrier(1))
  also have ... = {a*a | a. a∈ spectrum A}
    using assms herm real-diag-decompD(1) spectrum-def unitary-diag-spectrum-eq

    by blast
  finally show ?thesis .
qed

lemma adjoint-uminus:
  shows Complex-Matrix.adjoint (−A) = − (Complex-Matrix.adjoint A)
proof (rule eq-matI)
  fix i j
  assume i < dim-row (− Complex-Matrix.adjoint A) and
    j < dim-col (− Complex-Matrix.adjoint A)
  thus Complex-Matrix.adjoint (− A) $$ (i, j) =
    (− Complex-Matrix.adjoint A) $$ (i, j)
    by (simp add: adjoint-eval conjugate-neg)
qed auto

lemma (in fixed-carrier-mat) sum-mat-zero:
  assumes finite I
  and ⋀i. i ∈ I ⟹ A i ∈ fc-mats
  and ⋀i. i∈ I ⟹ f i = 0
shows sum-mat (λ i. (f i) ·m (A i)) I = 0m dimR dimC using assms
proof (induct rule: finite-induct)
  case empty
  then show ?case using sum-mat-empty by simp
next
  case (insert j F)
  hence sum-mat (λi. f i ·m A i) (insert j F) = f j ·m A j +
    sum-mat (λi. f i ·m A i) F
    using sum-mat-insert
    by (smt (verit, best) Set.basic-monos(7) image-subsetI insertI1
        smult-mem subset-insertI)
  also have ... = 0m dimR dimC + sum-mat (λi. f i ·m A i) F
    using insert smult-zero[of A j] fc-mats-carrier by force
  also have ... = 0m dimR dimC + 0m dimR dimC using insert by simp
  finally show ?case by simp
qed

lemma (in fixed-carrier-mat) sum-mat-zero':
  fixes A::'b ⇒ 'a Matrix.mat
  assumes finite I
  and ⋀i. i ∈ I ⟹ A i = 0m dimR dimC
shows sum-mat A I = 0m dimR dimC using assms
proof (induct rule: finite-induct)
```

case *empty*
then show *?case* **using** *sum-mat-empty* **by** *simp*
next
case (*insert j F*)
have *sum-mat A* (*insert j F*) = *A j* + *sum-mat A F* **using** *sum-mat-insert*
by (*metis Set.basic-monos*(*7*) *image-subsetI insertI1 insert*(*1*)
insert(*2*) *insert*(*4*) *subset-insertI zero-mem*)
also have ... = 0_m *dimR dimC* + *sum-mat A F*
using *insert* **by** *simp*
also have ... = 0_m *dimR dimC* + 0_m *dimR dimC* **using** *insert* **by** *simp*
finally show *?case* **by** *simp*
qed

lemma (**in** *fixed-carrier-mat*) *sum-mat-remove*:
assumes *A* ‘ *I* ⊆ *fc-mats*
and *A*: *finite I* **and** *x*: *x* ∈ *I*
shows *sum-mat A I* = *A x* + *sum-mat A* (*I*−{*x*}) **unfolding** *sum-mat-def*
using *assms sum-with-insert*[*of A x I*−{*x*}] *insert-Diff* **by** *fastforce*

lemma (**in** *fixed-carrier-mat*) *sum-mat-singleton*:
fixes *A*::*'b* ⇒ *'a Matrix.mat*
assumes *finite I*
and *A* ‘ *I* ⊆ *fc-mats*
and *j* ∈ *I*
and ∀ *i*∈*I*. *i* ≠ *j* ⟶ *f i* = *0*
shows *sum-mat* (λ *i*. (*f i*) $\cdot_m$ (*A i*)) *I* = *f j* $\cdot_m$ (*A j*)
proof −
have *sum-mat* (λ *i*. (*f i*) $\cdot_m$ (*A i*)) *I* = *f j* $\cdot_m$ (*A j*) +
sum-mat (λ *i*. (*f i*) $\cdot_m$ (*A i*)) (*I*−{*j*}) **using** *sum-mat-remove*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*)
image-subset-iff smult-mem)
moreover have *sum-mat* (λ *i*. (*f i*) $\cdot_m$ (*A i*)) (*I*−{*j*}) = 0_m *dimR dimC*
proof (*rule sum-mat-zero*)
show ⋀*i*. *i* ∈ *I* − {*j*} ⟹ *A i* ∈ *fc-mats* **using** *assms* **by** *auto*
qed (*auto simp add*: *assms*)
ultimately show *sum-mat* (λ *i*. (*f i*) $\cdot_m$ (*A i*)) *I* = *f j* $\cdot_m$ (*A j*)
by (*metis Matrix.right-add-zero-mat assms*(*2*) *assms*(*3*) *fc-mats-carrier*
image-subset-iff smult-mem)
qed

context *fixed-carrier-mat*
begin
lemma *sum-mat-disj-union*:
assumes *finite J*
and *finite I*
and *I* ∩ *J* = {}
and ∀ *i* ∈ *I* ∪ *J*. *A i* ∈ *fc-mats*
shows *sum-mat A* (*I* ∪ *J*) = *sum-mat A I* + *sum-mat A J* **using** *assms*
proof (*induct rule*: *finite-induct*)

```
  case empty
  then show ?case
    by (simp add: sum-mat-carrier)
next
  case (insert x F)
  have sum-mat A (I ∪ (insert x F)) = sum-mat A (insert x (I ∪ F)) by simp
  also have ... = A x + sum-mat A (I ∪ F)
  proof (rule sum-mat-insert)
    show A x ∈ fc-mats by (simp add: local.insert(6))
    show A ‘ (I ∪ F) ⊆ fc-mats using local.insert(6) by force
    show finite (I∪ F) using insert by simp
    show x ∉ I ∪ F using insert by auto
  qed
  also have ... = A x + sum-mat A I + sum-mat A F using insert
    by (simp add: add-assoc fc-mats-carrier sum-mat-carrier)
  also have ... = sum-mat A I + sum-mat A (insert x F)
  proof −
    have A x + sum-mat A F = sum-mat A (insert x F)
      by (simp add: insert.prems(3) local.insert(1) local.insert(2)
          subset-eq sum-mat-insert)
    thus ?thesis
      by (metis Un-iff add-assoc add-commute fc-mats-carrier
          insertCI local.insert(6) sum-mat-carrier)
  qed
  finally show ?case .
qed

lemma sum-with-reindex-cong′:
  fixes g :: 'c ⇒ 'a Matrix.mat
  assumes ∀ x. g x ∈ fc-mats
  and ∀ x. h x ∈ fc-mats
  and inj-on l B
  and ⋀x. x ∈ B ⟹ g (l x) = h x
  shows sum-with (+) (0_m dimR dimC) g (l ‘ B) =
  sum-with (+) (0_m dimR dimC) h B
  by (rule sum-with-reindex-cong, (simp add: assms)+)

lemma sum-mat-cong′:
  shows finite I ⟹ (⋀i. i∈ I ⟹ A i = B i) ⟹
    (⋀i. i∈ I ⟹ A i ∈ fc-mats) ⟹
    (⋀i. i∈ I ⟹ B i ∈ fc-mats) ⟹ I = J ⟹ sum-mat A I = sum-mat B J
proof (induct arbitrary: J rule: finite-induct)
  case empty
  then show ?case by simp
next
  case (insert x F)
  have sum-mat A (insert x F) = A x + sum-mat A F
    using insert sum-mat-insert[of A]
    by (meson image-subsetI insert-iff)
```

```
  also have ... = B x + sum-mat B F using insert by force
  also have ... = sum-mat B (insert x F) using insert sum-mat-insert[of B]
    by (metis image-subsetI insert-iff)
  also have ... = sum-mat B J using insert by simp
  finally show ?case .
qed

lemma sum-mat-reindex-cong:
  assumes finite B
  and ⋀x. x ∈ l‘ B ⟹ g x ∈ fc-mats
  and ⋀x. x ∈ B ⟹ h x ∈ fc-mats
  and inj-on l B
  and ⋀x. x ∈ B ⟹ g (l x) = h x
  shows sum-mat g (l ‘ B) = sum-mat h B
proof −
  define gp where gp = (λi. if i∈ l‘B then g i else (0_m dimR dimC))
  define hp where hp = (λi. if i ∈ B then h i else (0_m dimR dimC))
  have sum-mat g (l‘B) = sum-mat gp (l‘B)
  proof (rule sum-mat-cong′)
    show ⋀i. i ∈ l ‘ B ⟹ g i = gp i unfolding gp-def by auto
    show ⋀i. i ∈ l ‘ B ⟹ g i ∈ fc-mats using assms by simp
    show ⋀i. i ∈ l ‘ B ⟹ gp i ∈ fc-mats unfolding gp-def using assms by auto
  qed (simp add: assms)+
  also have ... = sum-mat hp B unfolding sum-mat-def
  proof (rule sum-with-reindex-cong′)
    show ∀ x. gp x ∈ fc-mats unfolding gp-def using assms
      by (simp add: zero-mem)
    show ∀ x. hp x ∈ fc-mats unfolding hp-def using assms
      by (simp add: zero-mem)
    show ⋀x. x ∈ B ⟹ gp (l x) = hp x
      by (simp add: assms(5) gp-def hp-def)
  qed (simp add: assms)
  also have ... = sum-mat h B
  proof (rule sum-mat-cong′)
    show ⋀i. i ∈ B ⟹ hp i = h i unfolding hp-def by auto
    show ⋀i. i ∈ B ⟹ hp i ∈ fc-mats unfolding hp-def using assms by auto
    show ⋀i. i ∈ B ⟹ h i ∈ fc-mats using assms by simp
  qed (simp add: assms)+
  finally show ?thesis .
qed

lemma sum-mat-mod-eq:
  fixes A :: nat ⇒ 'a Matrix.mat
  assumes ⋀x. x ∈ {..<m} ⟹ A x ∈ fc-mats
shows sum-mat (λi. A (i mod m)) ((λi. n * m+i)‘{..< m}) = sum-mat A {..<m}

proof (rule sum-mat-reindex-cong)
  show ⋀x. x ∈ {..<m} ⟹ A ((n * m + x) mod m) = A x by simp
  show inj-on ((+) (n * m)) {..<m} by simp
```

show $\bigwedge x.\ x \in (+)\ (n * m)\ `\ \{..<m\} \Longrightarrow A\ (x\ mod\ m) \in fc\text{-}mats$
using *assms* **by** *force*
qed (*simp add*: *assms*)+

lemma *sum-mat-singleton'*:
assumes $A\ i \in fc\text{-}mats$
shows *sum-mat* $A\ \{i\} = A\ i$
by (*metis add-zero assms comm-add-mat empty-iff fc-mats-carrier finite.intros(1) image-is-empty subsetI sum-mat-empty sum-mat-insert zero-mem*)

end

context *cpx-sq-mat*
begin

lemma *sum-mat-mod-div-ne-0*:
assumes $\bigwedge k.\ k < (nC::nat) \Longrightarrow A\ k \in carrier\text{-}mat\ n\ n$
and $\bigwedge j.\ j < (nD::nat) \Longrightarrow B\ j \in carrier\text{-}mat\ m\ m$
and $0 < n$
and $0 < m$
and $dimR = n * m$
and $nD \neq 0$
shows *sum-mat* $(\lambda i.\ sum\text{-}mat\ (\lambda j.\ f\ i * g\ j \cdot_m ((A\ i) \bigotimes (B\ j)))\ \{..< nD\})$
$\{..< nC\} =$
sum-mat $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))\ \{..< nC*nD\}$
proof −
define D **where** $D = (\lambda i.\ sum\text{-}mat\ (\lambda j.\ f\ i*g\ j \cdot_m ((A\ i) \bigotimes (B\ j)))\ \{..< nD\})$
have *fc*: $fc\text{-}mats = carrier\text{-}mat\ (n*m)\ (n*m)$
using *assms fc-mats-carrier dim-eq*
by *simp*
show *?thesis* **using** *assms*
proof (*induct nC*)
case *0*
define C **where** $C = sum\text{-}mat\ D\ \{..< (0::nat)\}$
have $C = 0_m\ (n*m)\ (n*m)$ **unfolding** *C-def*
using *sum-mat-empty assms dim-eq*
by (*simp add*: *fixed-carrier-mat-def*)
moreover have *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))\ \{..< 0*nD\} = 0_m\ (n*m)\ (n*m)$
using *sum-mat-empty assms dim-eq*
by (*simp add*: *fixed-carrier-mat-def*)
ultimately show *?case* **unfolding** *C-def* **by** *simp*
next
case (*Suc nC*)
define C **where** $C = sum\text{-}mat\ (\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))\ \{..< nC*nD\}$
have *dm*: $\bigwedge i.\ i \in \{..<Suc\ nC\} \Longrightarrow D\ i \in fc\text{-}mats$

proof −
fix i
assume $i \in \{..<Suc\ nC\}$
hence $A\ i \in carrier\text{-}mat\ n\ n$ **using** Suc **by** $simp$
hence $\bigwedge j.\ j \in \{..<nD\} \Longrightarrow B\ j \in carrier\text{-}mat\ m\ m$ **using** Suc
by $simp$
hence $\bigwedge j.\ j \in \{..<nD\} \Longrightarrow A\ i \otimes B\ j \in fc\text{-}mats$
using fc ‹$A\ i \in carrier\text{-}mat\ n\ n$› $tensor\text{-}mat\text{-}carrier$
by ($metis\ carrier\text{-}matD(1)\ carrier\text{-}matD(2)$)
thus $D\ i \in fc\text{-}mats$ **unfolding** $D\text{-}def$
by ($metis$ ($mono\text{-}tags$, $lifting$) $cpx\text{-}sq\text{-}mat\text{-}smult\ fc\text{-}mats\text{-}carrier$
$sum\text{-}mat\text{-}carrier$)
qed
have $sum\text{-}mat\ D\ \{..<Suc\ nC\} = sum\text{-}mat\ D\ (\{..<nC\} \cup \{nC..<Suc\ nC\})$
proof −
have $\{..<Suc\ nC\} = \{..<nC\} \cup \{nC..<Suc\ nC\}$ **by** $auto$
thus $?thesis$ **by** $simp$
qed
also have $... = sum\text{-}mat\ D\ \{..<nC\} + sum\text{-}mat\ D\ \{nC..<Suc\ nC\}$
proof ($rule\ sum\text{-}mat\text{-}disj\text{-}union$)
show $\forall i \in \{..<nC\} \cup \{nC..<Suc\ nC\}.\ D\ i \in fc\text{-}mats$ **using** dm **by** $auto$
qed $auto$
also have $... = C + sum\text{-}mat\ D\ \{nC..<Suc\ nC\}$
using Suc **unfolding** $C\text{-}def\ D\text{-}def$ **by** $simp$
also have $... = C + (sum\text{-}mat\ (\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD))\cdot_m$
$((A\ (i\ div\ nD)) \otimes (B\ (i\ mod\ nD))))\ \{nC*nD..<Suc\ nC*nD\})$
proof −
have $sum\text{-}mat\ D\ \{nC..<Suc\ nC\} = sum\text{-}mat\ D\ \{nC\}$ **by** $simp$
also have $... = D\ nC$ **using** dm
by ($simp\ add$: $sum\text{-}mat\text{-}singleton'$)
also have $... = (sum\text{-}mat\ (\lambda i.\ (f\ nC * g\ (i\ mod\ nD))\cdot_m$
$((A\ nC) \otimes (B\ (i\ mod\ nD))))\ ((+)\ (nC * nD)\ `\ \{..<nD\}))$
unfolding $D\text{-}def$
proof ($rule\ sum\text{-}mat\text{-}mod\text{-}eq[symmetric]$)
show $\bigwedge x.\ x \in \{..<nD\} \Longrightarrow f\ nC * g\ x \cdot_m (A\ nC \otimes B\ x) \in fc\text{-}mats$
proof −
fix x
assume $x \in \{..<nD\}$
hence $B\ x \in carrier\text{-}mat\ m\ m$ **using** Suc **by** $simp$
have $A\ nC \in carrier\text{-}mat\ n\ n$ **using** Suc **by** $simp$
hence $A\ nC \otimes B\ x \in fc\text{-}mats$
using $fc\ tensor\text{-}mat\text{-}carrier$ ‹$B\ x \in carrier\text{-}mat\ m\ m$› **by** $blast$
thus $f\ nC * g\ x \cdot_m (A\ nC \otimes B\ x) \in fc\text{-}mats$
by ($simp\ add$: $cpx\text{-}sq\text{-}mat\text{-}smult$)
qed
qed
also have $... = sum\text{-}mat\ (\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD))\cdot_m$
$((A\ (i\ div\ nD)) \otimes (B\ (i\ mod\ nD))))\ \{nC*nD..<Suc\ nC*nD\}$
proof ($rule\ sum\text{-}mat\text{-}cong'$)

```
    show (+) (nC * nD) ` {..<nD} = {nC * nD..<Suc nC * nD}
      by (simp add: lessThan-atLeast0)
    show ⋀i. i ∈ (+) (nC * nD) ` {..<nD} ⟹
      f nC * g (i mod nD) ·m (A nC ⨂ B (i mod nD)) ∈ fc-mats
    proof −
      fix i
      assume i ∈ (+) (nC * nD) ` {..<nD}
      hence i mod nD < nD using assms mod-less-divisor by blast
      hence B (i mod nD) ∈ carrier-mat m m using Suc by simp
      moreover have A nC ∈ carrier-mat n n using Suc by simp
      ultimately have A nC ⨂ B (i mod nD) ∈ fc-mats
        using fc tensor-mat-carrier by blast
      thus f nC * g (i mod nD) ·m (A nC ⨂ B (i mod nD)) ∈ fc-mats
        by (simp add: cpx-sq-mat-smult)
    qed
    show ⋀i. i ∈ (+) (nC * nD) ` {..<nD} ⟹
      f (i div nD) * g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)) ∈
      fc-mats
    proof −
      fix i
      assume i ∈ (+) (nC * nD) ` {..<nD}
      hence i div nD = nC using Suc(2) mod-less-divisor
        by (metis ‹(+) (nC * nD) ` {..<nD} = {nC * nD..<Suc nC * nD}›
            index-div-eq semiring-norm(174))
      have i mod nD < nD using ‹i ∈ (+) (nC * nD) ` {..<nD}›
          mod-less-divisor assms by blast
      hence B (i mod nD) ∈ carrier-mat m m using Suc by simp
      moreover have A (i div nD) ∈ carrier-mat n n
        using ‹i div nD = nC› Suc by simp
      ultimately have A (i div nD) ⨂ B (i mod nD) ∈ fc-mats
        using fc tensor-mat-carrier by blast
      thus f (i div nD) * g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)) ∈
        fc-mats
        by (simp add: cpx-sq-mat-smult)
    qed
  qed auto
  finally have sum-mat D {nC..< Suc nC} =
    sum-mat (λi. (f (i div nD) * g (i mod nD))·m
    ((A (i div nD)) ⨂ (B (i mod nD)))) {nC*nD..< Suc nC*nD} .
  thus ?thesis by simp
qed
also have ... =
  sum-mat (λi. f (i div nD)*g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)))
  ({..< nC * nD} ∪ {nC * nD..<Suc nC * nD}) unfolding C-def
proof (rule sum-mat-disj-union[symmetric])
  show ∀ i∈{..<nC * nD} ∪ {nC * nD..<Suc nC * nD}.
   f (i div nD) * g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)) ∈ fc-mats
  proof
    fix i
```

```
      assume i ∈ {..<nC * nD} ∪ {nC * nD..<Suc nC * nD}
      hence i ∈ {..< Suc nC * nD} by auto
      hence i div nD < Suc nC using Suc(2) mod-less-divisor
        by (simp add: less-mult-imp-div-less)
      have i mod nD < nD using ‹i ∈ {..<nC * nD} ∪ {nC * nD..<Suc nC *
nD}›
        Suc(2) mod-less-divisor assms by blast
      hence B (i mod nD) ∈ carrier-mat m m using Suc by simp
      moreover have A (i div nD) ∈ carrier-mat n n
        using ‹i div nD < Suc nC› Suc by simp
      ultimately have A (i div nD) ⨂ B (i mod nD) ∈ fc-mats
        using fc tensor-mat-carrier by blast
      thus f (i div nD) * g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)) ∈
        fc-mats
        by (simp add: cpx-sq-mat-smult)
    qed
  qed auto
  also have ... =
    sum-mat (λi. f (i div nD)*g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)))
    {..< Suc nC * nD}
  proof −
    have {..< nC * nD} ∪ {nC * nD..<Suc nC * nD} = {..< Suc nC *  nD}
      by auto
    thus ?thesis by simp
  qed
  finally show ?case unfolding D-def .
 qed
qed

lemma sum-mat-mod-div-eq-0:
 assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
 and 0 < n
 and nD = 0
 and dimR = n *m
shows sum-mat (λi. sum-mat (λj. f i * g j·m ((A i) ⨂ (B j))) {..< nD})
 {..< nC} =
 sum-mat (λi. (f (i div nD) * g (i mod nD))·m
 ((A (i div nD)) ⨂ (B (i mod nD)))) {..< nC*nD}
proof−
 have {..< nC*nD} = {} using assms by simp
 hence sum-mat (λi. f (i div nD) * g (i mod nD) ·m
   (A (i div nD) ⨂ B (i mod nD))) {..<nC * nD} = 0m (n*m) (n*m)
   using sum-mat-empty assms dim-eq
   by (simp add: fixed-carrier-mat-def)
 moreover have sum-mat (λi. sum-mat (λj. f i * g j ·m (A i ⨂ B j)) {..<nD})

   {..<nC} = 0m dimR dimC
 proof (rule sum-mat-zero′)
   fix i
```

```
    assume i ∈ {..< nC}
    show sum-mat (λj. f i * g j ·_m (A i ⊗ B j)) {..<nD} = 0_m dimR dimC
      using assms sum-mat-empty by simp
  qed simp
  ultimately show ?thesis using assms dim-eq by simp
qed

lemma sum-mat-mod-div:
  assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
  and ⋀j. j < (nD::nat) ⟹ B j ∈ carrier-mat m m
  and 0 < n
  and 0 < m
  and dimR = n *m
shows sum-mat (λi. sum-mat (λj. f i * g j·_m ((A i) ⊗ (B j))) {..< nD})
  {..< nC} =
  sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
  ((A (i div nD)) ⊗ (B (i mod nD)))) {..< nC*nD}
proof (cases nD = 0)
  case True
  then show ?thesis using sum-mat-mod-div-eq-0 assms by simp
next
  case False
  then show ?thesis using sum-mat-mod-div-ne-0 assms by simp
qed

lemma sum-sum-mat-expand-ne-0:
  assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
  and ⋀j. j < (nD::nat) ⟹ B j ∈ carrier-mat m m
  and R∈ carrier-mat (n*m) (n*m)
  and 0 < n
  and 0 < m
  and nD ≠ 0
  and dimR = n *m
shows sum-mat (λi. sum-mat (λj. f i * g j·_m ((A i) ⊗ (B j))*R) {..< nD})
  {..< nC} =
  sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
  ((A (i div nD)) ⊗ (B (i mod nD))) * R) {..< nC*nD}
proof −
  define D where D = (λi. sum-mat (λj. f i*g j·_m ((A i) ⊗ (B j)) * R)
    {..< nD})
  have fc: fc-mats = carrier-mat (n*m) (n*m)
    using assms fc-mats-carrier dim-eq
    by simp
  show ?thesis using assms
  proof (induct nC)
    case 0
    define C where C = sum-mat D {..< (0::nat)}
    have C = 0_m (n*m) (n*m) unfolding C-def
      using sum-mat-empty assms dim-eq
```

```
    by (simp add: fixed-carrier-mat-def)
  moreover have sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R) {..< 0*nD} = 0_m (n*m) (n*m)
    using sum-mat-empty assms dim-eq
    by (simp add: fixed-carrier-mat-def)
  ultimately show ?case unfolding C-def by simp
next
  case (Suc nC)
  define C where C = sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⨂ (B (i mod nD)))* R) {..< nC*nD}
  have R∈ fc-mats using fc-mats-carrier Suc dim-eq by simp
  have dm: ⋀i. i ∈ {..<Suc nC} ⟹ D i ∈ fc-mats
  proof −
    fix i
    assume i ∈ {..<Suc nC}
    hence A i ∈ carrier-mat n n using Suc by simp
    hence ⋀j. j∈ {..< nD} ⟹ B j ∈ carrier-mat m m using Suc
      by simp
    hence ⋀j. j∈ {..< nD} ⟹ A i ⨂ B j ∈ fc-mats
      using fc ‹A i ∈ carrier-mat n n› tensor-mat-carrier
      by (metis carrier-matD(1) carrier-matD(2))
    hence ⋀j. j ∈ {..< nD} ⟹ (A i ⨂ B j) * R ∈ fc-mats using Suc fc
      using cpx-sq-mat-mult by blast
    thus D i ∈ fc-mats unfolding D-def
      by (metis (mono-tags, lifting) ‹R ∈ fc-mats›
          ‹⋀j. j ∈ {..<nD} ⟹ A i ⨂ B j ∈ fc-mats› cpx-sq-mat-mult
          cpx-sq-mat-smult fc-mats-carrier sum-mat-carrier)
  qed
  have sum-mat D {..< Suc nC} = sum-mat D ({..< nC} ∪ {nC..< Suc nC})
  proof −
    have {..< Suc nC} = {..< nC} ∪ {nC..< Suc nC} by auto
    thus ?thesis by simp
  qed
  also have ... = sum-mat D {..< nC} + sum-mat D {nC..< Suc nC}
  proof (rule sum-mat-disj-union)
    show ∀ i∈{..<nC} ∪ {nC..<Suc nC}. D i ∈ fc-mats using dm by auto
  qed auto
  also have ... = C + sum-mat D {nC..< Suc nC}
    using Suc unfolding C-def D-def by simp
  also have ... = C + (sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R) {nC*nD..< Suc nC*nD})
  proof −
    have sum-mat D {nC..< Suc nC} = sum-mat D {nC} by simp
    also have ... = D nC  using  dm
      by (simp add: sum-mat-singleton')
    also have ... = (sum-mat (λi. (f nC * g (i mod nD))·_m
      ((A nC) ⨂ (B (i mod nD))) * R) ((+) (nC * nD) ` {..<nD}))
      unfolding D-def
    proof (rule sum-mat-mod-eq[symmetric])
```

show $\bigwedge x.\ x \in \{..<nD\} \Longrightarrow f\ nC * g\ x \cdot_m (A\ nC \bigotimes B\ x) * R \in$ *fc-mats*
proof −
fix x
assume $x \in \{..< nD\}$
hence $B\ x \in$ *carrier-mat m m* **using** *Suc* **by** *simp*
have $A\ nC \in$ *carrier-mat n n* **using** *Suc* **by** *simp*
hence $A\ nC \bigotimes B\ x \in$ *fc-mats*
using *fc tensor-mat-carrier* ‹$B\ x \in$ *carrier-mat m m*› **by** *blast*
thus $f\ nC * g\ x \cdot_m (A\ nC \bigotimes B\ x) * R \in$ *fc-mats*
by (*simp add*: ‹$R \in$ *fc-mats*› *cpx-sq-mat-mult cpx-sq-mat-smult*)
qed
qed
also have ... = *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))) * R)\ \{nC*nD..< Suc\ nC*nD\}$
proof (*rule sum-mat-cong′*)
show $(+)\ (nC * nD)\ `\ \{..<nD\} = \{nC * nD..<Suc\ nC * nD\}$
by (*simp add*: *lessThan-atLeast0*)
show $\bigwedge i.\ i \in (+)\ (nC * nD)\ `\ \{..<nD\} \Longrightarrow$
$f\ nC * g\ (i\ mod\ nD) \cdot_m (A\ nC \bigotimes B\ (i\ mod\ nD)) * R \in$ *fc-mats*
proof −
fix i
assume $i \in (+)\ (nC * nD)\ `\ \{..<nD\}$
hence $i\ mod\ nD < nD$ **using** *Suc mod-less-divisor* **by** *blast*
hence $B\ (i\ mod\ nD) \in$ *carrier-mat m m* **using** *Suc* **by** *simp*
moreover have $A\ nC \in$ *carrier-mat n n* **using** *Suc* **by** *simp*
ultimately have $A\ nC \bigotimes B\ (i\ mod\ nD) \in$ *fc-mats*
using *fc tensor-mat-carrier* **by** *blast*
thus $f\ nC * g\ (i\ mod\ nD) \cdot_m (A\ nC \bigotimes B\ (i\ mod\ nD)) * R \in$ *fc-mats*
by (*simp add*: ‹$R \in$ *fc-mats*› *cpx-sq-mat-mult cpx-sq-mat-smult*)
qed
show $\bigwedge i.\ i \in (+)\ (nC * nD)\ `\ \{..<nD\} \Longrightarrow$
$f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \bigotimes B\ (i\ mod\ nD)) * R \in$
fc-mats
proof −
fix i
assume $i \in (+)\ (nC * nD)\ `\ \{..<nD\}$
hence $i\ div\ nD = nC$ **using** *Suc*(*2*) *mod-less-divisor*
by (*metis* ‹$(+)\ (nC * nD)\ `\ \{..<nD\} = \{nC * nD..<Suc\ nC * nD\}$›
index-div-eq semiring-norm(*174*))
have $i\ mod\ nD < nD$ **using** ‹$i \in (+)\ (nC * nD)\ `\ \{..<nD\}$› *Suc*
mod-less-divisor **by** *blast*
hence $B\ (i\ mod\ nD) \in$ *carrier-mat m m* **using** *Suc* **by** *simp*
moreover have $A\ (i\ div\ nD) \in$ *carrier-mat n n*
using ‹$i\ div\ nD = nC$› *Suc* **by** *simp*
ultimately have $A\ (i\ div\ nD) \bigotimes B\ (i\ mod\ nD) \in$ *fc-mats*
using *fc tensor-mat-carrier* **by** *blast*
thus $f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \bigotimes B\ (i\ mod\ nD)) * R \in$
fc-mats
by (*simp add*: ‹$R \in$ *fc-mats*› *cpx-sq-mat-mult cpx-sq-mat-smult*)

```
      qed
    qed auto
    finally have sum-mat D {nC..< Suc nC} =
      sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
      ((A (i div nD)) ⨂ (B (i mod nD)))* R) {nC*nD..< Suc nC*nD} .
    thus ?thesis by simp
  qed
  also have ... =
   sum-mat (λi. f (i div nD)*g (i mod nD)·_m(A (i div nD) ⨂ B (i mod nD))*R)
    ({..< nC * nD} ∪ {nC * nD..<Suc nC * nD}) unfolding C-def
  proof (rule sum-mat-disj-union[symmetric])
    show ∀ i∈{..<nC * nD} ∪ {nC * nD..<Suc nC * nD}.
    f (i div nD) *g (i mod nD) ·_m (A (i div nD) ⨂ B (i mod nD))*R ∈ fc-mats
    proof
      fix i
      assume i ∈ {..<nC * nD} ∪ {nC * nD..<Suc nC * nD}
      hence i ∈ {..< Suc nC * nD} by auto
      hence i div nD < Suc nC using Suc(2) mod-less-divisor
        by (simp add: less-mult-imp-div-less)
       have i mod nD < nD using ‹i ∈ {..<nC * nD} ∪ {nC * nD..<Suc nC *
nD}›
        Suc mod-less-divisor by blast
      hence B (i mod nD) ∈ carrier-mat m m using Suc by simp
      moreover have A (i div nD) ∈ carrier-mat n n
        using ‹i div nD < Suc nC› Suc by simp
      ultimately have A (i div nD) ⨂ B (i mod nD) ∈ fc-mats
        using fc tensor-mat-carrier by blast
      thus f (i div nD) * g (i mod nD) ·_m (A (i div nD) ⨂ B (i mod nD))*R ∈
        fc-mats
        by (simp add: ‹R ∈ fc-mats› cpx-sq-mat-mult cpx-sq-mat-smult)
    qed
  qed auto
  also have ... =
   sum-mat (λi. f (i div nD)*g (i mod nD)·_m(A (i div nD) ⨂ B (i mod nD))*R)
    {..< Suc nC * nD}
  proof −
    have {..< nC * nD} ∪ {nC * nD..<Suc nC * nD} = {..< Suc nC *  nD}
      by auto
    thus ?thesis by simp
  qed
  finally show ?case unfolding D-def .
 qed
qed

lemma sum-sum-mat-expand-eq-0:
 assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
 and R∈ carrier-mat (n*m) (n*m)
 and 0 < n
 and 0 < m
```

and $nD = 0$
and $dimR = n * m$
shows *sum-mat* $(\lambda i.$ *sum-mat* $(\lambda j.\ f\ i * g\ j \cdot_m ((A\ i) \bigotimes (B\ j)) * R)\ \{..< nD\})$
$\{..< nC\} =$
sum-mat $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))) * R)\ \{..< nC * nD\}$
proof −
have $\{..< nC * nD\} = \{\}$ **using** *assms* **by** *simp*
hence *sum-mat* $(\lambda i.\ f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m$
$(A\ (i\ div\ nD) \bigotimes B\ (i\ mod\ nD)) * R)\ \{..< nC * nD\} = 0_m\ (n * m)\ (n * m)$
using *sum-mat-empty assms dim-eq*
by (*simp add*: *fixed-carrier-mat-def*)
moreover have *sum-mat* $(\lambda i.$ *sum-mat* $(\lambda j.\ f\ i * g\ j \cdot_m (A\ i \bigotimes B\ j) * R)$
$\{..< nD\})$
$\{..< nC\} = 0_m\ dimR\ dimC$
proof (*rule sum-mat-zero'*)
fix i
assume $i \in \{..< nC\}$
show *sum-mat* $(\lambda j.\ f\ i * g\ j \cdot_m (A\ i \bigotimes B\ j) * R)\ \{..< nD\} =$
$0_m\ dimR\ dimC$
using *assms sum-mat-empty* **by** *simp*
qed *simp*
ultimately show *?thesis* **using** *assms dim-eq* **by** *simp*
qed

lemma *sum-sum-mat-expand*:
assumes $\bigwedge k.\ k < (nC::nat) \Longrightarrow A\ k \in carrier\text{-}mat\ n\ n$
and $\bigwedge j.\ j < (nD::nat) \Longrightarrow B\ j \in carrier\text{-}mat\ m\ m$
and $R \in carrier\text{-}mat\ (n * m)\ (n * m)$
and $0 < n$
and $0 < m$
and $dimR = n * m$
shows *sum-mat* $(\lambda i.$ *sum-mat* $(\lambda j.\ f\ i * g\ j \cdot_m ((A\ i) \bigotimes (B\ j)) * R)\ \{..< nD\})$
$\{..< nC\} =$
sum-mat $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))) * R)\ \{..< nC * nD\}$
proof (*cases* $nD = 0$)
case *True*
then show *?thesis* **using** *assms sum-sum-mat-expand-eq-0* **by** *simp*
next
case *False*
then show *?thesis* **using** *assms sum-sum-mat-expand-ne-0* **by** *simp*
qed

end

3 Results on tensor products

lemma *tensor-mat-trace*:

assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat m m*
and $0 < n$
and $0 < m$
shows *Complex-Matrix.trace* $(A \otimes B) =$ *Complex-Matrix.trace A* $*$ *Complex-Matrix.trace B*
proof $-$
have $\{0 ..< n*m\} = \{..< n*m\}$ **by** *auto*
have *n*: $\{0 ..< n\} = \{..< n\}$ **by** *auto*
have *m*: $\{0 ..< m\} = \{..< m\}$ **by** *auto*
have *Complex-Matrix.trace* $(A \otimes B) = (\sum i \in \{0 ..< n*m\}. (A \otimes B)$ $$ $(i,i))$
unfolding *Complex-Matrix.trace-def* **using** *tensor-mat-carrier assms* **by** *simp*
also have $... = (\sum i \in \{..< n*m\}.$
A $$ $(i \text{ div } m, i \text{ div } m) * B$ $$ $(i \text{ mod } m, i \text{ mod } m))$
using *index-tensor-mat′ assms* $\langle\{0 ..< n*m\} = \{..< n*m\}\rangle$ **by** *simp*
also have $... =$*sum* $(\lambda i.$ *sum* $(\lambda j.$ A $$ $(i, i) * B$ $$ $(j,j))$ $\{..< m\})$ $\{..< n\}$
by (*rule sum-nat-div-mod*[*symmetric*])
also have $...$ = *sum* $(\lambda i.$ A $$$(i,i))$ $\{..< n\}*($*sum* $(\lambda j.$ B $$ $(j,j))$ $\{..< m\})$
by (*rule sum-product*[*symmetric*])
also have $...$ = *Complex-Matrix.trace A* $*$ (*Complex-Matrix.trace B*)
using *n m assms* **unfolding** *Complex-Matrix.trace-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *tensor-vec-inner-prod*:
assumes $u \in$ *carrier-vec n*
and $v \in$ *carrier-vec n*
and $a \in$ *carrier-vec n*
and $b \in$ *carrier-vec n*
and $0 < n$
shows *Complex-Matrix.inner-prod* (*tensor-vec u v*) (*tensor-vec a b*) =
Complex-Matrix.inner-prod u a $*$ *Complex-Matrix.inner-prod v b*
proof $-$
have $\{0 ..< n * n\} = \{..< n*n\}$ **by** *auto*
have $\{0 ..< n\} = \{..< n\}$ **by** *auto*
have *Complex-Matrix.inner-prod* (*tensor-vec u v*) (*tensor-vec a b*) =
$(\sum i \in \{0 ..< n * n\}.$ (*vec-index* (*tensor-vec a b*) *i*) $*$
vec-index (*conjugate* (*tensor-vec u v*)) *i*)
unfolding *scalar-prod-def* **using** *assms* **by** *simp*
also have $... = (\sum i \in \{0 ..< n * n\}.$ *vec-index a* $(i \text{ div } n) *$
vec-index b $(i \text{ mod } n) *$ (*vec-index* (*conjugate* (*tensor-vec u v*)) *i*))
proof $-$
have $\forall i < n * n.$ *vec-index* (*tensor-vec a b*) *i* = *vec-index a* $(i \text{ div } n) *$
vec-index b $(i \text{ mod } n)$ **using** *assms* **by** *simp*
thus *?thesis* **by** *auto*
qed
also have $... = (\sum i \in \{0 ..< n * n\}.$ *vec-index a* $(i \text{ div } n) *$
vec-index b $(i \text{ mod } n) *$ (*conjugate* (*vec-index* (*tensor-vec u v*) *i*)))

```
    using assms by simp
  also have ... = (∑ i ∈ {0 ..< n * n}. vec-index a (i div n) *
    vec-index b (i mod n) * (conjugate (vec-index u (i div n) *
    vec-index v (i mod n))))
  proof −
    have ∀ i < n * n. vec-index (tensor-vec u v) i = vec-index u (i div n) *
    vec-index v (i mod n)
      using assms by simp
    thus ?thesis by auto
  qed
  also have ... = (∑ i ∈ {0 ..< n * n}. vec-index a (i div n) *
    vec-index b (i mod n) * (conjugate (vec-index u (i div n)) *
    (conjugate (vec-index v (i mod n)))))
    by simp
  also have ... = (∑ i ∈ {0 ..< n * n}. vec-index a (i div n) *
    (conjugate (vec-index u (i div n)) * (vec-index b (i mod n) *
    (conjugate (vec-index v (i mod n))))))
    by (simp add: ab-semigroup-mult-class.mult-ac(1)
        vector-space-over-itself.scale-left-commute)
  also have ... = (∑ i ∈ {..< n * n}. (vec-index a (i div n) *
    (conjugate (vec-index u (i div n))) * (vec-index b (i mod n) *
    (conjugate (vec-index v (i mod n))))))
    using ‹{0 ..< n * n} = {..< n*n}›
    by (metis (no-types, lifting) sum.cong vector-space-over-itself.scale-scale)
  also have ... =sum (λi. sum (λj. vec-index a i * conjugate (vec-index u i) *
    (vec-index b j * (conjugate (vec-index v j)))) {..< n}) {..< n}
    by (rule sum-nat-div-mod[symmetric])
  also have ... = sum (λi. vec-index a i * conjugate (vec-index u i)) {..< n}*
    (sum (λj. vec-index b j * (conjugate (vec-index v j))) {..< n})
    by (rule sum-product[symmetric])
  also have ... = Complex-Matrix.inner-prod u a * Complex-Matrix.inner-prod v
b
  proof −
    have dim-vec (conjugate u) = n using assms by simp
    moreover have dim-vec (conjugate v) = n using assms by simp
    ultimately show ?thesis using ‹{0 ..< n} = {..< n}›
      unfolding Matrix.scalar-prod-def by simp
  qed
  finally show ?thesis .
qed

lemma tensor-mat-positive:
  assumes A∈ carrier-mat n n
  and B∈ carrier-mat m m
  and 0 < n
  and 0 < m
  and Complex-Matrix.positive A
  and Complex-Matrix.positive B
shows Complex-Matrix.positive (A ⨂ B)
```

proof (*rule positive-if-decomp*)
show $A \bigotimes B \in$ *carrier-mat* $(n*m)$ $(n*m)$ **using** *assms* **by** *auto*
have $\exists P \in$ *carrier-mat n n*. $P *$ *Complex-Matrix.adjoint* $P = A$
using *assms positive-only-if-decomp* **by** *simp*
from *this* **obtain** P **where** $P \in$ *carrier-mat n n*
and $P *$ *Complex-Matrix.adjoint* $P = A$ **by** *auto* **note** *ppr* = *this*
have $\exists Q \in$ *carrier-mat m m*. $Q *$ *Complex-Matrix.adjoint* $Q = B$
using *assms positive-only-if-decomp* **by** *simp*
from *this* **obtain** Q **where** $Q \in$ *carrier-mat m m*
and $Q *$ *Complex-Matrix.adjoint* $Q = B$ **by** *auto* **note** *qpr* = *this*
define M **where** $M = P \bigotimes Q$
have *Complex-Matrix.adjoint M* =
Complex-Matrix.adjoint P $\bigotimes$ (*Complex-Matrix.adjoint Q*) **unfolding** *M-def*
using *tensor-mat-adjoint ppr qpr assms*
by *blast*
hence $M *$ *Complex-Matrix.adjoint M* =
$(P *$ *Complex-Matrix.adjoint P*$) \bigotimes (Q *$ *Complex-Matrix.adjoint Q*$)$
using *mult-distr-tensor M-def ppr qpr assms* **by** *fastforce*
also have ... $= A \bigotimes B$ **using** *ppr qpr* **by** *simp*
finally have $M *$ *Complex-Matrix.adjoint* $M = A \bigotimes B$ **.**
thus $\exists M.\ M *$ *Complex-Matrix.adjoint* $M = A \bigotimes B$ **by** *auto*
qed

lemma *tensor-mat-square-idty*:
assumes $A * A = 1_m\ n$
and $B * B = 1_m\ m$
and $0 < n$
and $0 < m$
shows $(A \bigotimes B) * (A \bigotimes B) = 1_m\ (n*m)$
proof −
have $(A \bigotimes B) * (A \bigotimes B) = A*A \bigotimes (B*B)$
proof (*rule mult-distr-tensor*[*symmetric*])
show *a*: *dim-col A* = *dim-row A*
by (*metis assms*(*1*) *index-mult-mat*(*2*) *index-mult-mat*(*3*) *index-one-mat*(*2*)
index-one-mat(*3*))
show *b*: *dim-col B* = *dim-row B*
by (*metis assms*(*2*) *index-mult-mat*(*2*) *index-mult-mat*(*3*) *index-one-mat*(*2*)
index-one-mat(*3*))
show *0* < *dim-col A*
by (*metis a assms*(*1*) *assms*(*3*) *index-mult-mat*(*2*) *index-one-mat*(*2*))
thus *0* < *dim-col A* **.**
show *0* < *dim-col B*
by (*metis b assms*(*2*) *assms*(*4*) *index-mult-mat*(*2*) *index-one-mat*(*2*))
thus *0* < *dim-col B* **.**
qed
also have ... $= 1_m\ n \bigotimes 1_m\ m$ **using** *assms* **by** *simp*
also have ... $= 1_m\ (n*m)$ **using** *tensor-mat-id assms* **by** *simp*
finally show *?thesis* **.**

qed

lemma *tensor-mat-commute*:
assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat m m*
and $C \in$ *carrier-mat n n*
and $D \in$ *carrier-mat m m*
and $0 < n$
and $0 < m$
and $A * C = C * A$
and $B * D = D * B$
shows $(A \bigotimes B) * (C \bigotimes D) = (C \bigotimes D) * (A \bigotimes B)$
proof −
have $(A \bigotimes B) * (C \bigotimes D) = (A*C) \bigotimes (B*D)$ **using** *mult-distr-tensor assms*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
also have ... $= (C*A) \bigotimes (D*B)$ **using** *assms* **by** *simp*
also have ... $= (C \bigotimes D) * (A \bigotimes B)$ **using** *mult-distr-tensor assms*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
finally show *?thesis* **.**
qed

lemma *tensor-mat-mult-id*:
assumes $A\in$ *carrier-mat n n*
and $B\in$ *carrier-mat m m*
and $0 < n$
and $0 < m$
shows $(A \bigotimes 1_m\ m) * (1_m\ n \bigotimes B) = A \bigotimes B$
proof −
have $(A \bigotimes 1_m\ m) * (1_m\ n \bigotimes B) = (A * 1_m\ n) \bigotimes (1_m\ m * B)$
using *mult-distr-tensor*
by (*metis assms carrier-matD*(*1*) *carrier-matD*(*2*)
index-one-mat(*2*) *index-one-mat*(*3*))
also have ... $= A \bigotimes B$
by (*metis assms*(*1*) *assms*(*2*) *left-mult-one-mat right-mult-one-mat*)
finally show *?thesis* **.**
qed

lemma *tensor-mat-trace-mult-distr*:
assumes $A\in$ *carrier-mat n n*
and $B\in$ *carrier-mat m m*
and $C\in$ *carrier-mat n n*
and $D\in$ *carrier-mat m m*
and $0 < n$
and $0 < m$
shows *Complex-Matrix.trace* $((A \bigotimes B) * (C \bigotimes D)) =$
Complex-Matrix.trace $(A * C) *$ (*Complex-Matrix.trace* $(B * D))$
proof −
have $(A \bigotimes B) * (C \bigotimes D) = (A*C) \bigotimes (B*D)$ **using** *assms mult-distr-tensor*
by *auto*

hence *Complex-Matrix.trace* $((A \bigotimes B) * (C \bigotimes D)) =$
Complex-Matrix.trace $((A*C) \bigotimes (B*D))$ **by** *simp*
also have ... = *Complex-Matrix.trace* $(A * C) *$ (*Complex-Matrix.trace* $(B * D)$)
by (*meson assms mult-carrier-mat tensor-mat-trace*)
finally show *?thesis* **.**
qed

lemma *tensor-mat-diagonal*:
assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat m m*
and *diagonal-mat A*
and *diagonal-mat B*
shows *diagonal-mat* $(A \bigotimes B)$ **unfolding** *diagonal-mat-def*
proof (*intro allI impI*)
fix *i j*
assume $i <$ *dim-row* $(A \bigotimes B)$
and $j <$ *dim-col* $(A \bigotimes B)$
and $i \neq j$
have $A \bigotimes B \in$ *carrier-mat* $(n*m)$ $(n*m)$
using *assms tensor-mat-carrier* **by** *blast*
hence $i < n * m$
by (*metis* ‹$i <$ *dim-row* $(A \bigotimes B)$› *carrier-matD(1)*)
have $j < n* m$
using ‹$A \bigotimes B \in$ *carrier-mat* $(n * m)$ $(n * m)$› ‹$j <$ *dim-col* $(A \bigotimes B)$› **by** *auto*
have $(A \bigotimes B)$ \$\$ $(i, j) = A$ \$\$ $(i$ *div* (*dim-row B*), j *div* (*dim-col B*)) *
B \$\$ (i *mod* (*dim-row B*), j *mod* (*dim-col B*)) **using** *index-tensor-mat'*
by (*metis* ‹$i <$ *dim-row* $(A \bigotimes B)$› ‹$j <$ *dim-col* $(A \bigotimes B)$› *dim-col-tensor-mat*
dim-row-tensor-mat less-nat-zero-code neq0-conv semiring-norm(63)
semiring-norm(64))
also have ... = *0*
proof (*cases i div* (*dim-row B*) = *j div* (*dim-col B*))
case *True*
have *i div* (*dim-row B*) < *n* **using** *assms* ‹$i < n * m$›
by (*metis carrier-matD(1) less-mult-imp-div-less*)
moreover have *j div* (*dim-row B*) < *n* **using** *assms* ‹$j < n * m$›
by (*metis carrier-matD(1) less-mult-imp-div-less*)
ultimately have (*i mod* (*dim-row B*) ≠ *j mod* (*dim-col B*)) **using** ‹$i \neq j$›
by (*metis True assms(2) carrier-matD(1) carrier-matD(2) mod-div-decomp*)
then show *?thesis* **using** *assms* **unfolding** *diagonal-mat-def*
by (*metis* ‹$i < n * m$› *carrier-matD(1) carrier-matD(2) gr-zeroI*
mod-less-divisor mult.commute semiring-norm(63) zero-order(3))
next
case *False*
have *i div* (*dim-row B*) < *n* **using** *assms* ‹$i < n * m$›
by (*metis carrier-matD(1) less-mult-imp-div-less*)
moreover have *j div* (*dim-row B*) < *n* **using** *assms* ‹$j < n * m$›
by (*metis carrier-matD(1) less-mult-imp-div-less*)

ultimately show *?thesis* **using** *assms* **unfolding** *diagonal-mat-def*
by (*metis False carrier-matD*(*1*) *carrier-matD*(*2*) *semiring-norm*(*63*))
qed
finally show $(A \bigotimes B)$ $\$\$ (i, j) = 0$ **.**
qed

lemma *tensor-mat-add-right*:
assumes $A \in$ *carrier-mat n m*
and $B \in$ *carrier-mat i j*
and $C \in$ *carrier-mat i j*
and $0 < m$
and $0 < j$
shows $A \bigotimes (B + C) = (A \bigotimes B) + (A \bigotimes C)$
proof (*rule eq-matI*)
have $B + C \in$ *carrier-mat i j* **using** *assms* **by** *simp*
hence *bc*: $A \bigotimes (B + C) \in$ *carrier-mat* $(n * i)$ $(m * j)$
using *assms tensor-mat-carrier*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
have $A \bigotimes B \in$ *carrier-mat* $(n * i)$ $(m * j)$
using *assms tensor-mat-carrier*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
moreover have $A \bigotimes C \in$ *carrier-mat* $(n * i)$ $(m * j)$
using *assms tensor-mat-carrier*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
ultimately have *a*: $(A \bigotimes B) + (A \bigotimes C) \in$ *carrier-mat* $(n * i)$ $(m * j)$
by *simp*
thus *dr*: *dim-row* $(A \bigotimes (B + C)) =$ *dim-row* $((A \bigotimes B) + (A \bigotimes C))$
using *bc* **by** *simp*
show *dc*: *dim-col* $(A \bigotimes B + C) =$ *dim-col* $((A \bigotimes B) + (A \bigotimes C))$
using *a bc* **by** *simp*
fix *k l*
assume $k <$ *dim-row* $((A \bigotimes B) + (A \bigotimes C))$
and $l <$ *dim-col* $((A \bigotimes B) + (A \bigotimes C))$
hence $(A \bigotimes B + C)$ $\$\$ (k, l) =$
A $\$\$ (*k div dim-row* $(B + C)$, *l div dim-col* $(B + C)) *$
$(B + C)$ $\$\$ (*k mod dim-row* $(B + C)$, *l mod dim-col* $(B + C))$
using *index-tensor-mat′*
by (*metis* ‹$B + C \in$ *carrier-mat i j*› *dc dr assms*(*1*) *assms*(*4*) *assms*(*5*) *bc*
carrier-matD(*1*) *carrier-matD*(*2*))
also have ... $= A$ $\$\$ (*k div dim-row* $(B + C)$, *l div dim-col* $(B + C)) *$
$(B$ $\$\$ (*k mod dim-row* $(B + C)$, *l mod dim-col* $(B + C)) +$
C $\$\$ (*k mod dim-row* $(B + C)$, *l mod dim-col* $(B + C)))$
by (*metis div-eq-0-iff* ‹$B + C \in$ *carrier-mat i j*›
‹$k <$ *dim-row* $((A \bigotimes B) + (A \bigotimes C))$› *assms*(*3*) *assms*(*5*) *bc carrier-matD*(*1*)
carrier-matD(*2*) *dr index-add-mat*(*1*) *less-nat-zero-code mod-div-trivial*
mult-not-zero)
also have ... $= A$ $\$\$ (*k div dim-row* $(B + C)$, *l div dim-col* $(B + C)) *$

B $$ (k *mod dim-row* (B + C), l *mod dim-col* (B + C)) +
A $$ (k *div dim-row* (B + C), l *div dim-col* (B + C)) ∗
C $$ (k *mod dim-row* (B + C), l *mod dim-col* (B + C))
using *distrib-left* **by** *blast*
also have ... = (A ⨂ B) $$ (k,l) + (A ⨂ C) $$ (k,l)
using ‹k < *dim-row* ((A ⨂ B) + (A ⨂ C))› ‹l < *dim-col* ((A ⨂ B) + (A ⨂ C))›
assms **by** *force*
also have ... = ((A ⨂ B) + (A ⨂ C)) $$ (k,l)
using ‹k < *dim-row* ((A ⨂ B) + (A ⨂ C))› ‹l < *dim-col* ((A ⨂ B) + (A ⨂ C))›
by *force*
finally show (A ⨂ B + C) $$ (k, l) = ((A ⨂ B) + (A ⨂ C)) $$ (k,l) .
qed

lemma *tensor-mat-zero*:
assumes B ∈ *carrier-mat i j*
and 0 < j
and 0 < m
shows 0_m n m ⨂ B = 0_m (n ∗ i) (m ∗ j)
proof (*rule eq-matI*)
show *dim-row* (0_m n m ⨂ B) = *dim-row* (0_m (n ∗ i) (m ∗ j))
using *assms* **by** *simp*
show *dim-col* (0_m n m ⨂ B) = *dim-col* (0_m (n ∗ i) (m ∗ j))
using *assms* **by** *simp*
fix k l
assume k < *dim-row* (0_m (n ∗ i) (m ∗ j))
and l < *dim-col* (0_m (n ∗ i) (m ∗ j))
thus (0_m n m ⨂ B) $$ (k, l) = 0_m (n ∗ i) (m ∗ j) $$ (k,l)
using *index-tensor-mat assms less-mult-imp-div-less* **by** *force*
qed

lemma *tensor-mat-zero′*:
assumes B ∈ *carrier-mat i j*
and 0 < j
and 0 < m
shows B ⨂ 0_m n m = 0_m (i ∗ n) (j∗m)
proof (*rule eq-matI*)
show *dim-row* (B ⨂ 0_m n m) = *dim-row* (0_m (i ∗ n) (j ∗ m))
using *assms* **by** *simp*
show *dim-col* (B ⨂ 0_m n m) = *dim-col* (0_m (i ∗ n) (j ∗ m))
using *assms* **by** *simp*
fix k l
assume k < *dim-row* (0_m (i ∗ n) (j ∗ m))
and l < *dim-col* (0_m (i ∗ n) (j ∗ m))
thus (B ⨂ 0_m n m) $$ (k, l) = 0_m (i ∗ n) (j ∗ m) $$ (k,l)
using *index-tensor-mat assms less-mult-imp-div-less*
by (*metis* (*no-types*, *lifting*) *carrier-matD*(*1*) *carrier-matD*(*2*)
index-zero-mat(*1*) *index-zero-mat*(*2*) *index-zero-mat*(*3*)

less-nat-zero-code linorder-neqE-nat mod-less-divisor mult-eq-0-iff)
qed

lemma *tensor-mat-sum-right*:
fixes *A*::*complex Matrix.mat*
assumes *finite I*
and *A*∈ *carrier-mat n m*
and ⋀*k*. *k*∈ *I* ⟹ ((*B k*)::*complex Matrix.mat*) ∈ *carrier-mat i j*
and *0* < *m*
and *0* < *j*
and *dimR* = *n* ∗*i*
and *dimC* = *m*∗*j*
shows *A* ⨂ (*fixed-carrier-mat.sum-mat i j B I*) =
fixed-carrier-mat.sum-mat (*n*∗*i*) (*m*∗*j*) (λ*i*. *A* ⨂ (*B i*)) *I*
using *assms*
proof (*induct rule*: *finite-induct*)
case *empty*
hence *A* ⨂ (*fixed-carrier-mat.sum-mat i j B* {}) = 0_m (*n* ∗*i*) (*m*∗*j*)
using *tensor-mat-zero*′
by (*simp add*: *fixed-carrier-mat.sum-mat-empty fixed-carrier-mat-def*)
also have ... = *fixed-carrier-mat.sum-mat* (*n*∗*i*) (*m*∗*j*) (λ*i*. *A* ⨂ (*B i*)) {}
by (*metis fixed-carrier-mat.intro fixed-carrier-mat.sum-mat-empty*)
finally show *?case* **.**
next
case (*insert x F*)
hence *A* ⨂ (*fixed-carrier-mat.sum-mat i j B* (*insert x F*)) =
A ⨂ (*B x* + (*fixed-carrier-mat.sum-mat i j B F*))
proof −
have *fixed-carrier-mat.sum-mat i j B* (*insert x F*) =
B x + (*fixed-carrier-mat.sum-mat i j B F*)
using *fixed-carrier-mat.sum-mat-insert*
by (*metis fixed-carrier-mat.intro image-subsetI insertCI*
insert(*1*) *insert*(*2*) *insert*(*5*))
thus *?thesis* **by** *simp*
qed
also have ... = (*A* ⨂ (*B x*)) + (*A* ⨂ (*fixed-carrier-mat.sum-mat i j B F*))
proof (*rule tensor-mat-add-right*)
show *0* < *m* **using** *assms* **by** *simp*
show *0* < *j* **using** *assms* **by** *simp*
show *A* ∈ *carrier-mat n m* **using** *insert* **by** *simp*
show *B x* ∈ *carrier-mat i j* **using** *insert* **by** *simp*
show *fixed-carrier-mat.sum-mat i j B F* ∈ *carrier-mat i j*
proof (*rule fixed-carrier-mat.sum-mat-carrier*)
show ⋀*k*. *k* ∈ *F* ⟹ *B k* ∈ *carrier-mat i j* **using** *insert* **by** *simp*
show *fixed-carrier-mat* (*carrier-mat i j*) *i j*
by (*simp add*: *fixed-carrier-mat.intro*)
qed
qed
also have ... = (*A* ⨂ (*B x*)) +

```
    fixed-carrier-mat.sum-mat (n∗i) (m∗j) (λi. A ⨂ (B i)) F
    using insert by simp
  also have ... = fixed-carrier-mat.sum-mat (n∗i) (m∗j) (λi. A ⨂ (B i))
    (insert x F)
  proof (rule fixed-carrier-mat.sum-mat-insert[symmetric])
    show finite F using insert by simp
    show x∉ F using insert by simp
    show A ⨂ B x ∈ carrier-mat (n∗i) (m∗j)
      using tensor-mat-carrier insert
      by (metis carrier-matD(1) carrier-matD(2) insertI1)
    show (λi. A ⨂ B i) ‘ F ⊆ carrier-mat (n∗i) (m∗j)
    proof −
      {
        fix k
        assume k∈ F
        hence A ⨂ (B k) ∈ carrier-mat (n∗i) (m∗j)
          using tensor-mat-carrier insert by blast
      }
      thus ?thesis by auto
    qed
    show fixed-carrier-mat (carrier-mat (n ∗ i) (m ∗ j)) (n ∗ i) (m ∗ j)
      by (simp add: fixed-carrier-mat.intro)
  qed
  finally show A ⨂ (fixed-carrier-mat.sum-mat i j B (insert x F)) =
    fixed-carrier-mat.sum-mat (n∗i) (m∗j) (λi. A ⨂ (B i)) (insert x F) .
qed

lemma tensor-mat-add-left:
  assumes A∈ carrier-mat n m
  and B∈ carrier-mat n m
  and C∈ carrier-mat i j
  and 0 < m
  and 0 < j
shows (A + B) ⨂ C = (A ⨂ C) + (B ⨂ C)
proof (rule eq-matI)
  have A + B ∈ carrier-mat n m using assms by simp
  hence bc: (A+B) ⨂ C ∈ carrier-mat (n ∗ i) (m ∗ j)
    using assms tensor-mat-carrier
    by (metis carrier-matD(1) carrier-matD(2))
  have A ⨂ C ∈ carrier-mat (n ∗ i) (m ∗ j)
    using assms tensor-mat-carrier
    by (metis carrier-matD(1) carrier-matD(2))
  moreover have B ⨂ C ∈ carrier-mat (n ∗ i) (m ∗ j)
    using assms tensor-mat-carrier
    by (metis carrier-matD(1) carrier-matD(2))
  ultimately have a: (A ⨂ C) + (B ⨂ C) ∈ carrier-mat (n ∗ i) (m ∗ j)
    by simp
  thus dr: dim-row ((A+B) ⨂ C) = dim-row ((A ⨂ C) + (B ⨂ C))
    using bc by simp
```

show *dc*: *dim-col* ((*A*+*B*) ⨂ *C*) = *dim-col* ((*A* ⨂ *C*) + (*B* ⨂ *C*))
using *a bc* **by** *simp*
fix *k l*
assume *k* < *dim-row* ((*A* ⨂ *C*) + (*B* ⨂ *C*))
and *l* < *dim-col* ((*A* ⨂ *C*) + (*B* ⨂ *C*))
hence ((*A*+*B*) ⨂ *C*) $$ (*k*, *l*) =
(*A*+*B*) $$ (*k div dim-row C*, *l div dim-col C*) ∗
C $$ (*k mod dim-row C*, *l mod dim-col C*)
using *index-tensor-mat′*
by (*metis* ‹*A* + *B* ∈ *carrier-mat n m*› *assms*(*3*) *assms*(*4*) *assms*(*5*) *bc*
carrier-matD(*1*) *carrier-matD*(*2*) *dc dr*)
also have ... = (*A* $$ (*k div dim-row C*, *l div dim-col C*) +
B $$ (*k div dim-row C*, *l div dim-col C*)) ∗
C $$ (*k mod dim-row C*, *l mod dim-col C*)
using ‹*k* < *dim-row* ((*A* ⨂ *C*) + (*B* ⨂ *C*))› ‹*l* < *dim-col* ((*A* ⨂ *C*) + (*B* ⨂ *C*))›
less-mult-imp-div-less **by** *force*
also have ... = *A* $$ (*k div dim-row C*, *l div dim-col C*) ∗
C $$ (*k mod dim-row C*, *l mod dim-col C*) +
B $$ (*k div dim-row C*, *l div dim-col C*) ∗
C $$ (*k mod dim-row C*, *l mod dim-col C*)
using *distrib-right* **by** *blast*
also have ... = (*A* ⨂ *C*) $$ (*k*,*l*) + (*B* ⨂ *C*) $$ (*k*,*l*)
using ‹*k* < *dim-row* ((*A* ⨂ *C*) + (*B* ⨂ *C*))› ‹*l* < *dim-col* ((*A* ⨂ *C*) + (*B* ⨂ *C*))›
assms **by** *fastforce*
also have ... = ((*A* ⨂ *C*) + (*B* ⨂ *C*)) $$ (*k*,*l*)
using ‹*k* < *dim-row* ((*A* ⨂ *C*) + (*B* ⨂ *C*))› ‹*l* < *dim-col* ((*A* ⨂ *C*) + (*B* ⨂ *C*))›
by *force*
finally show ((*A*+*B*) ⨂ *C*) $$ (*k*, *l*) = ((*A* ⨂ *C*) + (*B* ⨂ *C*)) $$ (*k*,*l*) .
qed

lemma *tensor-mat-smult-left*:
assumes *A*∈ *carrier-mat n m*
and *B*∈ *carrier-mat i j*
and *0* < *m*
and *0* < *j*
shows *x* ·$_m$ *A* ⨂ *B* = *x* ·$_m$ (*A* ⨂ *B*)
proof (*rule eq-matI*)
have *x* ·$_m$ *A* ∈ *carrier-mat n m* **using** *assms* **by** *simp*
hence *x* ·$_m$ *A* ⨂ *B* ∈ *carrier-mat* (*n* ∗ *i*) (*m* ∗ *j*)
using *assms tensor-mat-carrier*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
moreover have *A* ⨂ *B* ∈ *carrier-mat* (*n* ∗ *i*) (*m* ∗ *j*)
using *assms tensor-mat-carrier*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
ultimately show
dim-row (*x* ·$_m$ *A* ⨂ *B*) = *dim-row* (*x* ·$_m$ (*A* ⨂ *B*))

dim-col (x $\cdot_m$ A $\bigotimes$ B) = *dim-col* (x $\cdot_m$ (A $\bigotimes$ B)) **by** *auto*
fix k l
assume k: k < *dim-row* (x $\cdot_m$ (A $\bigotimes$ B))
and l: l < *dim-col* (x $\cdot_m$ (A $\bigotimes$ B))
hence (x $\cdot_m$ A $\bigotimes$ B) $$ (k, l) =
(x $\cdot_m$ A) $$ (k *div dim-row* B, l *div dim-col* B) $*$
B $$ (k *mod dim-row* B, l *mod dim-col* B)
using *index-tensor-mat′ assms* **by** *force*
also have ... = x $*$ (A $$ (k *div dim-row* B, l *div dim-col* B)) $*$
B $$ (k *mod dim-row* B, l *mod dim-col* B)
using k l *less-mult-imp-div-less* **by** *fastforce*
also have ... = x $*$ (A $$ (k *div dim-row* B, l *div dim-col* B) $*$
B $$ (k *mod dim-row* B, l *mod dim-col* B)) **by** *simp*
also have ... = x $*$ (A $\bigotimes$ B) $$ (k,l)
using *assms* k l **by** *force*
also have ... = (x $\cdot_m$ (A $\bigotimes$ B)) $$ (k,l) **using** *assms* k l **by** *auto*
finally show (x $\cdot_m$ A $\bigotimes$ B) $$ (k, l) = (x $\cdot_m$ (A $\bigotimes$ B)) $$ (k,l) **.**
qed

lemma *tensor-mat-smult-right*:
assumes A∈ *carrier-mat* n m
and B∈ *carrier-mat* i j
and 0 < m
and 0 < j
shows A $\bigotimes$ (x $\cdot_m$ B) = x $\cdot_m$ (A $\bigotimes$ B)
proof (*rule eq-matI*)
have x $\cdot_m$ B ∈ *carrier-mat* i j **using** *assms* **by** *simp*
hence A $\bigotimes$ (x $\cdot_m$ B) ∈ *carrier-mat* (n $*$ i) (m $*$ j)
using *assms tensor-mat-carrier*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
moreover have A $\bigotimes$ B ∈ *carrier-mat* (n $*$ i) (m $*$ j)
using *assms tensor-mat-carrier*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
ultimately show
dim-row (A $\bigotimes$ x $\cdot_m$ B) = *dim-row* (x $\cdot_m$ (A $\bigotimes$ B))
dim-col (A $\bigotimes$ x $\cdot_m$ B) = *dim-col* (x $\cdot_m$ (A $\bigotimes$ B)) **by** *auto*
fix k l
assume k: k < *dim-row* (x $\cdot_m$ (A $\bigotimes$ B))
and l: l < *dim-col* (x $\cdot_m$ (A $\bigotimes$ B))
hence (A $\bigotimes$ (x $\cdot_m$ B)) $$ (k, l) =
A $$ (k *div dim-row* (x $\cdot_m$ B), l *div dim-col* (x $\cdot_m$ B)) $*$
(x $\cdot_m$ B) $$ (k *mod dim-row* (x $\cdot_m$ B), l *mod dim-col* (x $\cdot_m$ B))
using *index-tensor-mat′ assms* **by** *force*
also have ... = A $$ (k *div dim-row* (x $\cdot_m$ B), l *div dim-col* (x $\cdot_m$ B)) $*$
(x $*$ B $$ (k *mod dim-row* (x $\cdot_m$ B), l *mod dim-col* (x $\cdot_m$ B)))
using k l
by (*metis* (*no-types*, *opaque-lifting*) *add-lessD1 dim-col-tensor-mat*
dim-row-tensor-mat index-smult-mat(*1*) *index-smult-mat*(*2*)
index-smult-mat(*3*) *mod-less-divisor nat-0-less-mult-iff*

plus-nat.simps(1))
also have ... = x *(A$$ (k div dim-row (x $\cdot_m$ B), l div dim-col (x $\cdot_m$ B))*
B $$ (k mod dim-row (x $\cdot_m$ B), l mod dim-col (x $\cdot_m$ B))) **by** *simp*
also have ... = $x * (A \bigotimes B)$ $$ (k,l)
using *assms k l* **by** *force*
also have ... = $(x \cdot_m (A \bigotimes B))$ $$ (k,l) **using** *assms k l* **by** *auto*
finally show $(A \bigotimes (x \cdot_m B))$ $$ (k, l) = $(x \cdot_m (A \bigotimes B))$ $$ (k,l) **.**
qed

lemma *tensor-mat-smult*:
assumes $A \in$ *carrier-mat n m*
and $B \in$ *carrier-mat i j*
and $0 < m$
and $0 < j$
shows $x \cdot_m A \bigotimes (y \cdot_m B) = x * y \cdot_m (A \bigotimes B)$
by (*metis* (*no-types*, *opaque-lifting*) *assms smult-carrier-mat*
smult-smult-times tensor-mat-smult-left tensor-mat-smult-right)

lemma *tensor-mat-singleton-right*:
assumes $0 <$ *dim-col A*
and $B \in$ *carrier-mat 1 1*
shows $A \bigotimes B = B$ $$(0,0) $\cdot_m$ A
proof (*rule eq-matI*)
show *dim-row* $(A \bigotimes B)$ = *dim-row* $(B$ $$ $(0, 0) \cdot_m A)$ **using** *assms* **by** *auto*
show *dim-col* $(A \bigotimes B)$ = *dim-col* $(B$ $$ $(0, 0) \cdot_m A)$ **using** *assms* **by** *auto*
fix *i j*
assume $i <$ *dim-row* $(B$ $$ $(0, 0) \cdot_m A)$
and $j <$ *dim-col* $(B$ $$ $(0, 0) \cdot_m A)$
have $(A \bigotimes B)$ $$ $(i, j) = A$ $$ *(i div dim-row B,j div dim-col B)* $*$
B $$(i mod dim-row B, j mod dim-col B) **using** *index-tensor-mat*
‹$i <$ *dim-row* $(B$ $$ $(0, 0) \cdot_m A)$› ‹$j <$ *dim-col* $(B$ $$ $(0, 0) \cdot_m A)$› *assms*
by *fastforce*
also have ... = A $$$(i,j) * B$$$(0,0)$ **using** *assms* **by** *auto*
also have ... = $(B$ $$ $(0, 0) \cdot_m A)$ $$ (i, j)
using ‹$i <$ *dim-row* $(B$ $$ $(0, 0) \cdot_m A)$› ‹$j <$ *dim-col* $(B$ $$ $(0, 0) \cdot_m A)$›
by *force*
finally show $(A \bigotimes B)$ $$ $(i, j) = (B$ $$ $(0, 0) \cdot_m A)$ $$ (i, j) **.**
qed

lemma *tensor-mat-singleton-left*:
assumes $0 <$ *dim-col A*
and $B \in$ *carrier-mat 1 1*
shows $B \bigotimes A = B$ $$(0,0) $\cdot_m$ A
proof (*rule eq-matI*)
show *dim-row* $(B \bigotimes A)$ = *dim-row* $(B$ $$ $(0, 0) \cdot_m A)$ **using** *assms* **by** *auto*
show *dim-col* $(B \bigotimes A)$ = *dim-col* $(B$ $$ $(0, 0) \cdot_m A)$ **using** *assms* **by** *auto*
fix *i j*
assume $i <$ *dim-row* $(B$ $$ $(0, 0) \cdot_m A)$
and $j <$ *dim-col* $(B$ $$ $(0, 0) \cdot_m A)$

have $(B \bigotimes A)$ $\$\$ (i, j) = A \$\$ (i \mathit{ div dim\text{-}row}\ B, j \mathit{ div dim\text{-}col}\ B) *$
$B \$\$(i \mathit{ mod dim\text{-}row}\ B, j \mathit{ mod dim\text{-}col}\ B)$ **using** *index-tensor-mat*
$\langle i < \mathit{dim\text{-}row}\ (B \$\$ (0, 0) \cdot_m A)\rangle$ $\langle j < \mathit{dim\text{-}col}\ (B \$\$ (0, 0) \cdot_m A)\rangle$ *assms*
by *fastforce*
also have ... $= A \$\$(i,j) * B\$\$(0,0)$ **using** *assms* **by** *auto*
also have ... $= (B \$\$ (0, 0) \cdot_m A) \$\$ (i, j)$
using $\langle i < \mathit{dim\text{-}row}\ (B \$\$ (0, 0) \cdot_m A)\rangle$ $\langle j < \mathit{dim\text{-}col}\ (B \$\$ (0, 0) \cdot_m A)\rangle$
by *force*
finally show $(B \bigotimes A) \$\$ (i, j) = (B \$\$ (0, 0) \cdot_m A) \$\$ (i, j)$ **.**
qed

lemma *tensor-mat-sum-left*:
assumes *finite I*
and $B \in$ *carrier-mat i j*
and $\bigwedge k.\ k \in I \Longrightarrow A\ k \in$ *carrier-mat n m*
and $0 < m$
and $0 < j$
and $\mathit{dimR} = n * i$
and $\mathit{dimC} = m * j$
shows $(\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ I) \bigotimes B =$
$\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ (n*i)\ (m*j)\ (\lambda i.\ (A\ i) \bigotimes B)\ I$
using *assms*
proof (*induct rule*: *finite-induct*)
case *empty*
hence $(\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ \{\}) \bigotimes B = 0_m\ (n * i)\ (m*j)$
using *tensor-mat-zero*
by (*simp add*: *fixed-carrier-mat.sum-mat-empty fixed-carrier-mat-def*)
also have ... $= \mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ (n*i)\ (m*j)\ (\lambda i.\ (A\ i) \bigotimes B)\ \{\}$
by (*metis fixed-carrier-mat.intro fixed-carrier-mat.sum-mat-empty*)
finally show *?case* **.**
next
case (*insert x F*)
hence $(\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ (\mathit{insert}\ x\ F)) \bigotimes B =$
$(A\ x + (\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ F)) \bigotimes B$
proof −
have $\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ (\mathit{insert}\ x\ F) =$
$A\ x + (\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ F)$
using *fixed-carrier-mat.sum-mat-insert*
by (*metis fixed-carrier-mat.intro image-subsetI insertCI*
insert(*1*) *insert*(*2*) *insert*(*5*))
thus *?thesis* **by** *simp*
qed
also have ... $= (A\ x \bigotimes B) + (\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ F \bigotimes B)$
proof (*rule tensor-mat-add-left*)
show $0 < m$ **using** *assms* **by** *simp*
show $0 < j$ **using** *assms* **by** *simp*
show $A\ x \in$ *carrier-mat n m* **using** *insert* **by** *simp*
show $B \in$ *carrier-mat i j* **using** *insert* **by** *simp*
show $\mathit{fixed\text{-}carrier\text{-}mat.sum\text{-}mat}\ n\ m\ A\ F \in$ *carrier-mat n m*

```
    proof (rule fixed-carrier-mat.sum-mat-carrier)
      show ⋀k. k ∈ F ⟹ A k ∈ carrier-mat n m using insert by simp
      show fixed-carrier-mat (carrier-mat n m) n m
        by (simp add: fixed-carrier-mat.intro)
    qed
  qed
  also have ... = (A x ⊗ B) +
    fixed-carrier-mat.sum-mat (n∗i) (m∗j) (λi. A i ⊗ B) F
    using insert by simp
  also have ... = fixed-carrier-mat.sum-mat (n∗i) (m∗j) (λi. A i ⊗ B)
    (insert x F)
  proof (rule fixed-carrier-mat.sum-mat-insert[symmetric])
    show finite F using insert by simp
    show x∉ F using insert by simp
    show A x ⊗ B ∈ carrier-mat (n∗i) (m∗j)
      using tensor-mat-carrier insert by blast
    show (λi. A i ⊗ B) ‘ F ⊆ carrier-mat (n∗i) (m∗j)
    proof −
      {
        fix k
        assume k∈ F
        hence A k ⊗ B ∈ carrier-mat (n∗i) (m∗j)
          using tensor-mat-carrier insert by blast
      }
      thus ?thesis by auto
    qed
    show fixed-carrier-mat (carrier-mat (n ∗ i) (m ∗ j)) (n ∗ i) (m ∗ j)
      by (simp add: fixed-carrier-mat.intro)
  qed
  finally show fixed-carrier-mat.sum-mat n m A (insert x F) ⊗ B =
    fixed-carrier-mat.sum-mat (n∗i) (m∗j) (λi. A i ⊗ B) (insert x F) .
qed

lemma tensor-mat-diag-elem:
  assumes A∈ carrier-mat n n
  and B∈ carrier-mat m m
  and i < n ∗ m
  and 0 < n∗m
shows (A ⊗ B) $$ (i, i) = A $$ (i div m, i div m) ∗
    B $$ (i mod m, i mod m)
proof −
  have i < dim-row (A ⊗ B) using assms by auto
  have (A ⊗ B) $$ (i, i) = A $$ (i div (dim-row B), i div (dim-col B)) ∗
    B $$ (i mod (dim-row B), i mod (dim-col B)) using index-tensor-mat′
    by (metis ‹i < dim-row (A ⊗ B)› assms carrier-matD(2) dim-row-tensor-mat

        nat-0-less-mult-iff)
  also have ... = A $$ (i div m, i div m) ∗ B $$ (i mod m, i mod m)
    using assms by auto
```

finally show *?thesis* **.**
qed

context *cpx-sq-mat*
begin

lemma *tensor-mat-sum-mat-right*:
assumes *finite I*
and $A \in$ *carrier-mat n n*
and $\bigwedge k.\ k \in I \Longrightarrow B\ k \in$ *carrier-mat i i*
and $0 < n$
and $0 < i$
and $dimR = n * i$
shows $A \bigotimes$ (*fixed-carrier-mat.sum-mat i i B I*) = *sum-mat* $(\lambda i.\ A \bigotimes (B\ i))\ I$
using *assms dim-eq tensor-mat-sum-right* **by** *blast*

lemma *tensor-mat-sum-mat-left*:
assumes *finite I*
and $B \in$ *carrier-mat i i*
and $\bigwedge k.\ k \in I \Longrightarrow A\ k \in$ *carrier-mat n n*
and $0 < n$
and $0 < i$
and $dimR = n * i$
shows (*fixed-carrier-mat.sum-mat n n A I*) $\bigotimes$ B = *sum-mat* $(\lambda i.\ (A\ i) \bigotimes B)\ I$
using *assms dim-eq tensor-mat-sum-left* **by** *blast*

lemma *tensor-mat-sum-nat-mod-div-ne-0*:
assumes $\bigwedge k.\ k < (nC::nat) \Longrightarrow A\ k \in$ *carrier-mat n n*
and $\bigwedge j.\ j < (nD::nat) \Longrightarrow B\ j \in$ *carrier-mat m m*
and *fixed-carrier-mat.sum-mat n n* $(\lambda i.\ f\ i \cdot_m (A\ i))\ \{..< nC\} = C$
and *fixed-carrier-mat.sum-mat m m* $(\lambda j.\ g\ j \cdot_m (B\ j))\ \{..< nD\} = D$
and $0 < n$
and $0 < m$
and $nD \neq 0$
and $dimR = n * m$
shows *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))$
$\{..< nC*nD\} = C \bigotimes D$ **using** *assms*
proof (*induct nC arbitrary*: *C*)
case *0*
hence $C =$ *fixed-carrier-mat.sum-mat n n* $(\lambda i.\ f\ i \cdot_m (A\ i))\ \{\}$ **by** *simp*
also have $... = 0_m\ n\ n$
using *fixed-carrier-mat.sum-mat-empty*[*of - n n* $\lambda i.\ f\ i \cdot_m (A\ i)$]
by (*simp add*: *fixed-carrier-mat-def*)
finally have $C = 0_m\ n\ n$ **.**
moreover have $D \in$ *carrier-mat m m* **using** *0*
fixed-carrier-mat.sum-mat-carrier[*of - m m* $\{..< nD\}$ $\lambda j.\ g\ j \cdot_m (B\ j)$]
by (*simp add*: *fixed-carrier-mat-def*)
ultimately have $C \bigotimes D = 0_m\ (n*m)\ (n*m)$ **using** *tensor-mat-zero*

```
    by (simp add: 0(5) 0(6))
  have sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⊗ (B (i mod nD))))
    {..< 0*nD} = sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⊗ (B (i mod nD)))) {} by simp
  also have ... = 0_m (n*m) (n*m) using sum-mat-empty
    using 0 dim-eq by blast
  also have ... = C⊗ D using ‹C⊗ D = 0_m (n*m) (n*m)› by simp
  finally show ?case .
next
  case (Suc nC)
  define Cp where
    Cp = fixed-carrier-mat.sum-mat n n (λi. f i ·_m (A i)) {..< nC}
  have fc: ∀ i∈{..<nC * nD} ∪ {nC * nD..<Suc nC * nD}.
    (A (i div nD) ⊗ B (i mod nD)) ∈ fc-mats
  proof
    fix i
    assume i ∈ {..<nC * nD} ∪ {nC * nD..<Suc nC * nD}
    hence i: i ∈ {..< Suc nC * nD} by auto
    hence i div nD < Suc nC
      by (simp add: less-mult-imp-div-less)
    hence A (i div nD) ∈ carrier-mat n n using Suc by simp
    have i mod nD < nD using Suc by simp
    hence B (i mod nD) ∈ carrier-mat m m using Suc by simp
    hence A (i div nD) ⊗ B (i mod nD) ∈ carrier-mat (n*m) (n*m)
      using tensor-mat-carrier
      by (metis ‹A (i div nD) ∈ carrier-mat n n›
          carrier-matD(1) carrier-matD(2))
    thus (A (i div nD) ⊗ B (i mod nD)) ∈ fc-mats
      using Suc dim-eq fc-mats-carrier by blast
  qed
  have sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⊗ (B (i mod nD))))
    {..< (Suc nC)*nD} = sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⊗ (B (i mod nD)))) {..< nC*nD} +
    sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
    ((A (i div nD)) ⊗ (B (i mod nD)))) {nC*nD..< (Suc nC)*nD}
  proof −
    have {..< (Suc nC)*nD} = {..< nC*nD} ∪ {nC*nD..< (Suc nC)*nD} by
auto
    moreover have sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
      ((A (i div nD)) ⊗ (B (i mod nD))))
      ({..< nC*nD} ∪ {nC*nD..< (Suc nC)*nD}) =
      sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
      ((A (i div nD)) ⊗ (B (i mod nD)))) {..< nC*nD} +
      sum-mat (λi. (f (i div nD) * g (i mod nD))·_m
      ((A (i div nD)) ⊗ (B (i mod nD)))) {nC*nD..< (Suc nC)*nD}
    proof (rule sum-mat-disj-union)
      show {..<nC * nD} ∩ {nC * nD..<Suc nC * nD} = {}
```

by (*simp add*: *ivl-disj-int*(2))
show $\forall i \in \{..<nC * nD\} \cup \{nC * nD..<Suc\ nC * nD\}$.
$f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \bigotimes B\ (i\ mod\ nD)) \in$ *fc-mats*
using *fc smult-mem* **by** *blast*
qed *simp*+
ultimately show *?thesis* **by** *simp*
qed
also have ... =
$(Cp \bigotimes D) +$
sum-mat $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD)))) \{nC*nD..<(Suc\ nC)*nD\}$
proof −
have *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD)))) \{..<nC*nD\} = Cp \bigotimes D$
unfolding *Cp-def* **using** *Suc* **by** *simp*
thus *?thesis* **by** *simp*
qed
also have ... =
$(Cp \bigotimes D) +$
sum-mat $(\lambda i.\ (f\ nC * g\ (i\ mod\ nD)) \cdot_m$
$((A\ nC) \bigotimes (B\ (i\ mod\ nD)))) \{nC*nD..<(Suc\ nC)*nD\}$
proof −
have *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD)))) \{nC*nD..<(Suc\ nC)*nD\} =$
sum-mat $(\lambda i.\ (f\ nC * g\ (i\ mod\ nD)) \cdot_m$
$((A\ nC) \bigotimes (B\ (i\ mod\ nD)))) \{nC*nD..<(Suc\ nC)*nD\}$
proof (*rule sum-mat-cong*)
show $\bigwedge i.\ i \in \{nC * nD..<Suc\ nC * nD\} \Longrightarrow$
$f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \bigotimes B\ (i\ mod\ nD)) \in$
fc-mats **using** *fc* **by** (*metis UnI2 smult-mem*)
show $\bigwedge i.\ i \in \{nC * nD..<Suc\ nC * nD\} \Longrightarrow$
$f\ nC * g\ (i\ mod\ nD) \cdot_m (A\ nC \bigotimes B\ (i\ mod\ nD)) \in$ *fc-mats*
proof
fix *i*
assume $i \in \{nC * nD..<Suc\ nC * nD\}$
hence $i\ mod\ nD < nD$ **using** *Suc mod-less-divisor* **by** *blast*
hence $B\ (i\ mod\ nD) \in carrier\text{-}mat\ m\ m$ **using** *Suc* **by** *simp*
moreover have $A\ nC \in carrier\text{-}mat\ n\ n$ **using** *Suc* **by** *simp*
ultimately have $A\ nC \bigotimes B\ (i\ mod\ nD) \in carrier\text{-}mat\ (n*m)\ (n*m)$
using *tensor-mat-carrier* **by** (*metis carrier-matD*(1) *carrier-matD*(2))
hence $(A\ nC \bigotimes B\ (i\ mod\ nD)) \in$ *fc-mats*
using *Suc dim-eq fc-mats-carrier* **by** *blast*
thus $f\ nC * g\ (i\ mod\ nD) \cdot_m (A\ nC \bigotimes B\ (i\ mod\ nD)) \in$ *fc-mats*
using *smult-mem* **by** *blast*
qed *simp*
show $\bigwedge i.\ i \in \{nC * nD..<Suc\ nC * nD\} \Longrightarrow$
$f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \bigotimes B\ (i\ mod\ nD)) =$
$f\ nC * g\ (i\ mod\ nD) \cdot_m (A\ nC \bigotimes B\ (i\ mod\ nD))$
proof −

```
      fix i
      assume i ∈ {nC * nD..<Suc nC * nD}
      hence i div nD = nC
        by (metis atLeastLessThan-iff div-nat-eqI mult.commute)
      thus f (i div nD) * g (i mod nD) ·_m (A (i div nD) ⨂ B (i mod nD)) =
        f nC * g (i mod nD) ·_m (A nC ⨂ B (i mod nD)) by simp
    qed
  qed simp
  thus ?thesis by simp
qed
also have ... =
  (Cp ⨂ D) +
  sum-mat (λi. (f nC ·_m (A nC)) ⨂ (g (i mod nD)·_m (B (i mod nD))))
  {nC*nD..< (Suc nC)*nD}
proof −
  have sum-mat (λi. f nC * g (i mod nD) ·_m (A nC ⨂ B (i mod nD)))
    {nC * nD..<Suc nC * nD} =
    sum-mat (λi. f nC ·_m A nC ⨂ g (i mod nD) ·_m B (i mod nD))
    {nC * nD..<Suc nC * nD}
  proof (rule sum-mat-cong)
    show ⋀i. i ∈ {nC * nD..<Suc nC * nD} ⟹
      f nC * g (i mod nD) ·_m (A nC ⨂ B (i mod nD)) ∈ fc-mats
    proof −
      fix i
      assume i ∈ {nC * nD..<Suc nC * nD}
      have i mod nD < nD using Suc mod-less-divisor by blast
      hence B (i mod nD) ∈ carrier-mat m m using Suc by simp
      moreover have A nC ∈ carrier-mat n n by (simp add: Suc(2))
      ultimately have A nC ⨂ (B (i mod nD)) ∈ carrier-mat (n*m) (n*m)
        using tensor-mat-carrier
        by (metis carrier-matD(1) carrier-matD(2))
      hence A nC ⨂ B (i mod nD) ∈ fc-mats using fc-mats-carrier
        Suc dim-eq by blast
      thus f nC * g (i mod nD) ·_m (A nC ⨂ B (i mod nD)) ∈ fc-mats
        using cpx-sq-mat-smult by blast
    qed
    show ⋀i. i ∈ {nC * nD..<Suc nC * nD} ⟹
      f nC ·_m A nC ⨂ g (i mod nD) ·_m B (i mod nD) ∈ fc-mats
    proof −
      fix i
      assume i ∈ {nC * nD..<Suc nC * nD}
      have i mod nD < nD using Suc mod-less-divisor by blast
      hence g (i mod nD) ·_m B (i mod nD) ∈ carrier-mat m m using Suc
        by simp
      moreover have f nC ·_m A nC ∈ carrier-mat n n by (simp add: Suc(2))
      ultimately have f nC ·_m A nC ⨂ g (i mod nD) ·_m B (i mod nD) ∈
        carrier-mat (n*m) (n*m)
        using tensor-mat-carrier
        by (metis carrier-matD(1) carrier-matD(2))
```

```
    thus f nC ·m A nC ⨂ g (i mod nD) ·m B (i mod nD) ∈ fc-mats
      using fc-mats-carrier Suc dim-eq by blast
  qed
  show ⋀i. i ∈ {nC * nD..<Suc nC * nD} ⟹
    f nC * g (i mod nD) ·m (A nC ⨂ B (i mod nD)) =
    f nC ·m A nC ⨂ g (i mod nD) ·m B (i mod nD)
  proof −
    fix i
    assume i ∈ {nC * nD..<Suc nC * nD}
    show f nC * g (i mod nD) ·m (A nC ⨂ B (i mod nD)) =
    f nC ·m A nC ⨂ g (i mod nD) ·m B (i mod nD) using tensor-mat-smult
      by (metis div-eq-0-iff Suc(3) Suc(8)
          Suc.prems(1) assms(5) assms(6) lessI mod-div-trivial)
  qed
 qed simp
 thus ?thesis by simp
qed
also have ... =
 (Cp ⨂ D) +
 ((f nC ·m (A nC)) ⨂ (fixed-carrier-mat.sum-mat m m
   (λi. g (i mod nD)·m (B (i mod nD)))
 {nC*nD..< (Suc nC)*nD}))
proof −
 have sum-mat (λi. f nC ·m A nC ⨂ g (i mod nD) ·m B (i mod nD))
   {nC * nD..<Suc nC * nD} =
   f nC ·m (A nC) ⨂ (fixed-carrier-mat.sum-mat m m
     (λi. g (i mod nD)·m (B (i mod nD)))
   {nC*nD..< (Suc nC)*nD})
 proof (rule tensor-mat-sum-mat-right[symmetric])
   show 0 < n 0 < m dimR = n*m using Suc by auto
   show f nC ·m A nC ∈ carrier-mat n n by (simp add: Suc(2))
   fix i
   assume i ∈ {nC * nD..<Suc nC * nD}
   have i mod nD < nD using Suc mod-less-divisor by blast
   hence B (i mod nD) ∈ carrier-mat m m using Suc by simp
   thus g (i mod nD) ·m B (i mod nD) ∈ carrier-mat m m by simp
 qed simp
 thus ?thesis by simp
qed
also have ... =
 (Cp ⨂ D) +
 ((f nC ·m (A nC)) ⨂ (fixed-carrier-mat.sum-mat m m
 (λj. g j ·m (B j)) {..< nD}))
proof −
 have fixed-carrier-mat.sum-mat m m (λi. g (i mod nD) ·m B (i mod nD))
   {nC * nD..<Suc nC * nD} =
   fixed-carrier-mat.sum-mat m m (λi. g (i mod nD) ·m B (i mod nD))
   ((+) (nC * nD) ` {..<nD})
 proof (rule fixed-carrier-mat.sum-mat-cong′)
```

show $\{nC * nD..<Suc\ nC * nD\} = (+)\ (nC * nD)\ `\ \{..<nD\}$
by (*simp add*: *lessThan-atLeast0*)
show *fixed-carrier-mat* (*carrier-mat m m*) *m m*
by (*simp add*: *fixed-carrier-mat.intro*)
show $\bigwedge i.\ i \in \{nC * nD..<Suc\ nC * nD\} \Longrightarrow$
$g\ (i\ mod\ nD) \cdot_m B\ (i\ mod\ nD) \in carrier\text{-}mat\ m\ m$
proof −
fix *i*
assume $i \in \{nC * nD..<Suc\ nC * nD\}$
hence $i\ mod\ nD < nD$
using *Suc mod-less-divisor* **by** *blast*
thus $g\ (i\ mod\ nD) \cdot_m B\ (i\ mod\ nD) \in carrier\text{-}mat\ m\ m$
using *Suc*(*3*) *smult-carrier-mat* **by** *blast*
qed
thus $\bigwedge i.\ i \in \{nC * nD..<Suc\ nC * nD\} \Longrightarrow$
$g\ (i\ mod\ nD) \cdot_m B\ (i\ mod\ nD) \in carrier\text{-}mat\ m\ m$ **.**
qed *simp*+
also have ... =
fixed-carrier-mat.sum-mat m m $(\lambda j.\ g\ j \cdot_m B\ j)\ \{..<nD\}$
proof (*rule fixed-carrier-mat.sum-mat-mod-eq*)
show *fixed-carrier-mat* (*carrier-mat m m*) *m m*
by (*simp add*: *fixed-carrier-mat.intro*)
show $\bigwedge x.\ x \in \{..<nD\} \Longrightarrow g\ x \cdot_m B\ x \in carrier\text{-}mat\ m\ m$
by (*simp add*: *Suc*(*3*))
qed
finally have *fixed-carrier-mat.sum-mat m m*
$(\lambda i.\ g\ (i\ mod\ nD) \cdot_m B\ (i\ mod\ nD))$
$\{nC * nD..<Suc\ nC * nD\} =$
fixed-carrier-mat.sum-mat m m $(\lambda j.\ g\ j \cdot_m B\ j)\ \{..<nD\}$ **.**
thus *?thesis* **by** *simp*
qed
also have ... = $(Cp \bigotimes D) + ((f\ nC \cdot_m (A\ nC)) \bigotimes D)$ **using** *Suc* **by** *simp*
also have ... = $Cp + (f\ nC \cdot_m (A\ nC)) \bigotimes D$
proof (*rule tensor-mat-add-left*[*symmetric*])
show $Cp \in carrier\text{-}mat\ n\ n$ **unfolding** *Cp-def*
proof (*rule fixed-carrier-mat.sum-mat-carrier*)
show $\bigwedge i.\ i \in \{..<nC\} \Longrightarrow f\ i \cdot_m A\ i \in carrier\text{-}mat\ n\ n$
by (*simp add*: *Suc*(*2*))
show *fixed-carrier-mat* (*carrier-mat n n*) *n n*
by (*simp add*: *fixed-carrier-mat.intro*)
qed
have *fixed-carrier-mat.sum-mat m m* $(\lambda j.\ g\ j \cdot_m B\ j)\ \{..<nD\} \in$
carrier-mat m m
proof (*rule fixed-carrier-mat.sum-mat-carrier*)
show $\bigwedge i.\ i \in \{..<nD\} \Longrightarrow g\ i \cdot_m B\ i \in carrier\text{-}mat\ m\ m$
by (*simp add*: *Suc*)
show *fixed-carrier-mat* (*carrier-mat m m*) *m m*
by (*simp add*: *fixed-carrier-mat.intro*)
qed

thus $D \in$ *carrier-mat m m* **using** *Suc* **by** *simp*
show *f nC* $\cdot_m$ *A nC* $\in$ *carrier-mat n n*
by (*simp add*: *Suc*(*2*))
qed (*auto simp add*: *Suc*)
also have ... =
(*fixed-carrier-mat.sum-mat n n* ($\lambda i.$ *f i* $\cdot_m$ (*A i*)) {..< *Suc nC*}) $\bigotimes$ *D*
proof −
have *Cp* + *f nC* $\cdot_m$ *A nC* = *f nC* $\cdot_m$ *A nC* + *Cp*
proof (*rule comm-add-mat*)
show *f nC* $\cdot_m$ *A nC* $\in$ *carrier-mat n n* **by** (*simp add*: *Suc*(*2*))
show *Cp* $\in$ *carrier-mat n n* **unfolding** *Cp-def*
proof (*rule fixed-carrier-mat.sum-mat-carrier*)
show $\bigwedge i.\ i \in \{..< nC\} \Longrightarrow$ *f i* $\cdot_m$ *A i* $\in$ *carrier-mat n n*
by (*simp add*: *Suc*(*2*))
show *fixed-carrier-mat* (*carrier-mat n n*) *n n*
by (*simp add*: *fixed-carrier-mat.intro*)
qed
qed
also have ... = *fixed-carrier-mat.sum-mat n n*
($\lambda i.$ *f i* $\cdot_m$ (*A i*)) (*insert nC* {..< *nC*}) **unfolding** *Cp-def*
proof (*rule fixed-carrier-mat.sum-mat-insert*[*symmetric*])
show *f nC* $\cdot_m$ *A nC* $\in$ *carrier-mat n n*
by (*simp add*: *Suc*(*2*))
show *fixed-carrier-mat* (*carrier-mat n n*) *n n*
by (*simp add*: *fixed-carrier-mat.intro*)
show ($\lambda i.$ *f i* $\cdot_m$ *A i*) ‘ {..<*nC*} $\subseteq$ *carrier-mat n n*
proof
fix *x*
assume *x* $\in$ ($\lambda i.$ *f i* $\cdot_m$ *A i*) ‘ {..<*nC*}
hence $\exists i \in \{..<nC\}.\ x =$ *f i* $\cdot_m$ *A i* **by** *auto*
from *this* **obtain** *i* **where** *i*$\in$ {..<*nC*} **and** *x* = *f i* $\cdot_m$ *A i* **by** *auto*
have *f i* $\cdot_m$ *A i* $\in$ *carrier-mat n n*
using *Suc.prems*(*1*) ‹*i* $\in$ {..<*nC*}› **by** *auto*
thus *x* $\in$ *carrier-mat n n* **using** ‹*x* = *f i* $\cdot_m$ *A i*› **by** *simp*
qed
qed *auto*
also have ... = *fixed-carrier-mat.sum-mat n n* ($\lambda i.$ *f i* $\cdot_m$ (*A i*))
{..< *Suc nC*}
proof (*rule fixed-carrier-mat.sum-mat-cong*′)
show *fixed-carrier-mat* (*carrier-mat n n*) *n n*
by (*simp add*: *fixed-carrier-mat.intro*)
show *insert nC* {..<*nC*} = {..<*Suc nC*}
by (*simp add*: *lessThan-Suc*)
show $\bigwedge i.\ i \in$ *insert nC* {..<*nC*} $\Longrightarrow$ *f i* $\cdot_m$ *A i* $\in$ *carrier-mat n n*
by (*simp add*: *Suc*(*2*) ‹*insert nC* {..<*nC*} = {..<*Suc nC*}›)
thus $\bigwedge i.\ i \in$ *insert nC* {..<*nC*} $\Longrightarrow$ *f i* $\cdot_m$ *A i* $\in$ *carrier-mat n n* .
qed *auto*
finally have *Cp* + *f nC* $\cdot_m$ *A nC* = *fixed-carrier-mat.sum-mat n n*
($\lambda i.$ *f i* $\cdot_m$ (*A i*)) {..< *Suc nC*} .

thus *?thesis* **by** *simp*
qed
also have $... = C \bigotimes D$ **using** *Suc* **by** *simp*
finally show *?case* **.**
qed

lemma *tensor-mat-sum-nat-mod-div-eq-0*:
assumes $\bigwedge k.\ k < (nC::nat) \Longrightarrow A\ k \in carrier\text{-}mat\ n\ n$
and *fixed-carrier-mat.sum-mat* $n\ n\ (\lambda i.\ f\ i \cdot_m (A\ i))\ \{..< nC\} = C$
and *fixed-carrier-mat.sum-mat* $m\ m\ (\lambda j.\ g\ j \cdot_m (B\ j))\ \{..< nD\} = D$
and $0 < n$
and $0 < m$
and $nD = 0$
and $dimR = n * m$
shows *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))$
$\{..< nC*nD\} = C \bigotimes D$
proof −
have $D =$ *fixed-carrier-mat.sum-mat* $m\ m\ (\lambda i.\ g\ i \cdot_m (B\ i))\ \{\}$
using *assms* **by** *auto*
also have $... = 0_m\ m\ m$
using *fixed-carrier-mat.sum-mat-empty*[*of* - $m\ m\ \lambda i.\ g\ i \cdot_m (B\ i)$]
by (*simp add*: *fixed-carrier-mat-def*)
finally have $D = 0_m\ m\ m$ **.**
moreover have $C \in carrier\text{-}mat\ n\ n$ **using** *assms*
fixed-carrier-mat.sum-mat-carrier[*of* - $n\ n\ \{..< nC\}\ \lambda j.\ f\ j \cdot_m (A\ j)$]
by (*simp add*: *fixed-carrier-mat-def*)
ultimately have $C \bigotimes D = 0_m\ (n*m)\ (n*m)$ **using** *tensor-mat-zero′*
by (*simp add*: *assms*)
have *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))$
$\{..< nC*nD\} =$ *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))\ \{\}$ **using** *assms* **by** *simp*
also have $... = 0_m\ (n*m)\ (n*m)$ **using** *sum-mat-empty*
using *assms*(7) *dim-eq* **by** *blast*
also have $... = C \bigotimes D$ **using** ‹$C \bigotimes D = 0_m\ (n*m)\ (n*m)$› **by** *simp*
finally show *?thesis* **.**
qed

lemma *tensor-mat-sum-nat-mod-div*:
assumes $\bigwedge k.\ k < (nC::nat) \Longrightarrow A\ k \in carrier\text{-}mat\ n\ n$
and $\bigwedge j.\ j < (nD::nat) \Longrightarrow B\ j \in carrier\text{-}mat\ m\ m$
and *fixed-carrier-mat.sum-mat* $n\ n\ (\lambda i.\ f\ i \cdot_m (A\ i))\ \{..< nC\} = C$
and *fixed-carrier-mat.sum-mat* $m\ m\ (\lambda j.\ g\ j \cdot_m (B\ j))\ \{..< nD\} = D$
and $0 < n$
and $0 < m$
and $dimR = n * m$
shows *sum-mat* $(\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \bigotimes (B\ (i\ mod\ nD))))$

```
  {..< nC*nD} = C ⨂ D
proof (cases nD = 0)
  case True
  then show ?thesis using assms
      tensor-mat-sum-nat-mod-div-eq-0[OF assms(1) assms(3)] by simp
next
  case False
  then show ?thesis using assms tensor-mat-sum-nat-mod-div-ne-0 by simp
qed

end

lemma tensor-mat-sum-mult-trace-expand-ne-0:
  assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
  and ⋀j. j < (nD::nat) ⟹ B j ∈ carrier-mat m m
  and R ∈ carrier-mat (n*m) (n*m)
  and fixed-carrier-mat.sum-mat n n (λi. f i ·m (A i)) {..< nC} = C
  and fixed-carrier-mat.sum-mat m m (λj. g j ·m (B j)) {..< nD} = D
  and 0 < n
  and 0 < m
  and nD ≠ 0
  shows sum (λi. Complex-Matrix.trace ((f (i div nD) * g (i mod nD))·m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R)) {..< nC * nD} =
    Complex-Matrix.trace ((C ⨂ D) * R)
proof −
  define fc::complex Matrix.mat set where fc = carrier-mat (n*m) (n*m)
  interpret cpx-sq-mat n*m n*m fc
  proof
    show 0 < n*m using assms by simp
  qed (auto simp add: fc-def)
  have fc: ∀ i∈{..<nC * nD}.
    f (i div nD) * g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)) ∈ fc
  proof
    fix i
    assume i ∈ {..<nC * nD}
    hence i div nD <  nC
      by (simp add: less-mult-imp-div-less)
    hence A (i div nD) ∈ carrier-mat n n using assms by simp
    have i mod nD < nD using assms by simp
    hence B (i mod nD) ∈ carrier-mat m m using assms by simp
    hence A (i div nD) ⨂ B (i mod nD) ∈ carrier-mat (n*m) (n*m)
      using tensor-mat-carrier
      by (metis ‹A (i div nD) ∈ carrier-mat n n›
          carrier-matD(1) carrier-matD(2))
    hence (A (i div nD) ⨂ B (i mod nD)) ∈ fc
      using assms dim-eq fc-mats-carrier by blast
    thus f (i div nD) * g (i mod nD) ·m(A (i div nD) ⨂ B (i mod nD)) ∈ fc
      using smult-mem by blast
  qed
```

```
  have sum-mat (λi. ((f (i div nD) * g (i mod nD))·m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R)) {..< nC * nD} =
    (sum-mat (λi. f (i div nD) * g (i mod nD)·m
    ((A (i div nD)) ⨂ (B (i mod nD)))) {..< nC * nD}) * R
  proof (rule sum-mat-distrib-right)
    show R∈ fc using assms unfolding fc-def by simp
  qed (auto simp add: fc assms)
  also have ... = (C ⨂ D) * R
  proof −
    have sum-mat (λi. f (i div nD) * g (i mod nD)·m
      ((A (i div nD)) ⨂ (B (i mod nD)))) {..< nC * nD} = C ⨂ D
      using tensor-mat-sum-nat-mod-div assms by simp
    thus ?thesis by simp
  qed
  finally have sr: sum-mat (λi. ((f (i div nD) * g (i mod nD))·m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R)) {..< nC * nD} = (C ⨂ D) * R .
  have sum (λi. Complex-Matrix.trace ((f (i div nD) * g (i mod nD))·m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R)) {..< nC * nD} =
    Complex-Matrix.trace (sum-mat (λi. ((f (i div nD) * g (i mod nD))·m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R)) {..< nC * nD})
  proof (rule trace-sum-mat[symmetric])
    show ⋀i. i ∈ {..<nC * nD} ⟹
      f (i div nD) * g (i mod nD) ·m (A (i div nD) ⨂ B (i mod nD)) * R ∈ fc
      using fc assms cpx-sq-mat-mult fc-def by blast
  qed simp
  also have ... = Complex-Matrix.trace ((C ⨂ D) * R) using sr by simp
  finally show ?thesis .
qed

lemma tensor-mat-sum-mult-trace-expand-eq-0:
  assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
  and R ∈ carrier-mat (n*m) (n*m)
  and fixed-carrier-mat.sum-mat n n (λi. f i ·m (A i)) {..< nC} = C
  and fixed-carrier-mat.sum-mat m m (λj. g j ·m (B j)) {..< nD} = D
  and 0 < n
  and 0 < m
  and nD = 0
  shows sum (λi. Complex-Matrix.trace ((f (i div nD) * g (i mod nD))·m
    ((A (i div nD)) ⨂ (B (i mod nD))) * R)) {..< nC * nD} =
    Complex-Matrix.trace ((C ⨂ D) * R)
proof −
  have D = 0m m m using assms fixed-carrier-mat.sum-mat-empty
    fixed-carrier-mat.intro by fastforce
  hence C ⨂ D = C ⨂ (0m m m) by simp
  also have ... = 0m (n*m) (n*m)
  proof (rule tensor-mat-zero′)
    have fixed-carrier-mat.sum-mat n n (λi. f i ·m A i) {..<nC} ∈
      carrier-mat n n
    proof (rule fixed-carrier-mat.sum-mat-carrier)
```

```
    show fixed-carrier-mat (carrier-mat n n) n n
      by (simp add: fixed-carrier-mat.intro)
    show ⋀i. i ∈ {..<nC} ⟹ f i ·m A i ∈ carrier-mat n n using assms
      by simp
  qed
  thus C∈ carrier-mat n n using assms by simp
 qed (simp add: assms)+
 finally have C ⨂ D = 0m (n∗m) (n∗m) .
 hence (C ⨂ D) ∗ R = 0m (n∗m) (n∗m)
  by (simp add: assms left-mult-zero-mat)
 hence Complex-Matrix.trace ((C ⨂ D) ∗ R) = 0 by simp
 moreover have sum (λi. Complex-Matrix.trace ((f (i div nD)∗g (i mod nD))·m

  ((A (i div nD)) ⨂ (B (i mod nD))) ∗ R)) {..< nC ∗ nD} = 0
  using assms by simp
 ultimately show ?thesis by simp
qed

lemma tensor-mat-sum-mult-trace-expand:
 assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
 and ⋀j. j < (nD::nat) ⟹ B j ∈ carrier-mat m m
 and R ∈ carrier-mat (n∗m) (n∗m)
 and fixed-carrier-mat.sum-mat n n (λi. f i ·m (A i)) {..< nC} = C
 and fixed-carrier-mat.sum-mat m m (λj. g j ·m (B j)) {..< nD} = D
 and 0 < n
 and 0 < m
 shows sum (λi. Complex-Matrix.trace ((f (i div nD) ∗ g (i mod nD))·m
  ((A (i div nD)) ⨂ (B (i mod nD))) ∗ R)) {..< nC ∗ nD} =
  Complex-Matrix.trace ((C ⨂ D) ∗ R)
proof (cases nD = 0)
 case True
 then show ?thesis
  using assms tensor-mat-sum-mult-trace-expand-eq-0[OF assms(1)] by simp
next
 case False
 then show ?thesis
  using assms tensor-mat-sum-mult-trace-expand-ne-0[OF assms(1) assms(2)]
  by simp
qed

lemma tensor-mat-sum-mult-trace-ne-0:
 assumes ⋀k. k < (nC::nat) ⟹ A k ∈ carrier-mat n n
 and ⋀j. j < (nD::nat) ⟹ B j ∈ carrier-mat m m
 and R ∈ carrier-mat (n∗m) (n∗m)
 and fixed-carrier-mat.sum-mat n n (λi. f i ·m (A i)) {..< nC} = C
 and fixed-carrier-mat.sum-mat m m (λj. g j ·m (B j)) {..< nD} = D
 and 0 < n
 and 0 < m
 and 0 ≠ nD
```

shows *sum* (λi. (*sum* (λj. *Complex-Matrix.trace* ((*f i* ∗ *g j*)$\cdot_m$
((*A i*) ⨂ (*B j*)) ∗ *R*)) {..< *nD*})) {..< *nC*} =
Complex-Matrix.trace ((*C* ⨂ *D*) ∗ *R*)
proof −
define *fc*::*complex Matrix.mat set* **where** *fc* = *carrier-mat* (*n*∗*m*) (*n*∗*m*)
interpret *cpx-sq-mat n*∗*m n*∗*m fc*
proof
show *0* < *n*∗*m* **using** *assms* **by** *simp*
qed (*auto simp add*: *fc-def*)
have *sum* (λi. (*sum* (λj. *Complex-Matrix.trace* ((*f i* ∗ *g j*)$\cdot_m$
((*A i*) ⨂ (*B j*)) ∗ *R*)) {..< *nD*})) {..< *nC*} =
sum (λi. *Complex-Matrix.trace* (*sum-mat* (λj. (*f i* ∗ *g j*)$\cdot_m$
((*A i*) ⨂ (*B j*)) ∗ *R*) {..< *nD*})) {..< *nC*}
proof (*rule sum.cong*)
fix *x*
assume *x* ∈ {..< *nC*}
hence *A x* ∈ *carrier-mat n n* **using** *assms* **by** *simp*
show (∑ *j*∈ {..< *nD*}. *Complex-Matrix.trace* (*f x* ∗ *g j* ·$_m$ (*A x* ⨂ *B j*) ∗ *R*))=
Complex-Matrix.trace (*sum-mat* (λj. *f x* ∗ *g j* ·$_m$ (*A x* ⨂ *B j*) ∗ *R*)
{..<*nD*})
proof (*rule trace-sum-mat*[*symmetric*])
fix *j*
assume *j* ∈ {..< *nD*}
hence *B j* ∈ *carrier-mat m m* **using** *assms* **by** *simp*
hence *A x* ⨂ *B j* ∈ *carrier-mat* (*n*∗*m*) (*n*∗*m*)
using *tensor-mat-carrier*
by (*metis* ‹*A x* ∈ *carrier-mat n n*› *carrier-matD*(*1*) *carrier-matD*(*2*))
hence *A x* ⨂ *B j* ∈ *fc*
using *assms dim-eq fc-mats-carrier* **by** *blast*
thus *f x* ∗ *g j* ·$_m$(*A x* ⨂ *B j*)∗*R* ∈ *fc*
using *smult-mem assms*(*3*) *cpx-sq-mat-mult fc-def* **by** *blast*
qed *simp*
qed *simp*
also have ... = *Complex-Matrix.trace* (*sum-mat* (λi.
(*sum-mat* (λj. (*f i* ∗ *g j*)$\cdot_m$ ((*A i*) ⨂ (*B j*)) ∗ *R*) {..< *nD*})) {..< *nC*})
proof (*rule trace-sum-mat*[*symmetric*])
fix *x*
assume *x*∈{..< *nC*}
hence *A x*∈ *carrier-mat n n* **using** *assms* **by** *simp*
show *sum-mat* (λj. *f x* ∗ *g j* ·$_m$ (*A x* ⨂ *B j*) ∗ *R*) {..<*nD*} ∈ *fc*
unfolding *fc-def*
proof (*rule sum-mat-carrier*)
fix *j*
assume *j*∈ {..< *nD*}
hence *B j* ∈ *carrier-mat m m* **using** *assms* **by** *simp*
hence *A x* ⨂ *B j* ∈ *carrier-mat* (*n*∗*m*) (*n*∗*m*)
using *tensor-mat-carrier*
by (*metis* ‹*A x* ∈ *carrier-mat n n*› *carrier-matD*(*1*) *carrier-matD*(*2*))
hence *A x* ⨂ *B j* ∈ *fc*

using *assms dim-eq fc-mats-carrier* **by** *blast*
thus $f\ x * g\ j \cdot_m (A\ x \otimes B\ j) * R \in fc$
using *smult-mem assms(3) cpx-sq-mat-mult fc-def* **by** *blast*
qed
qed *simp*
also have ... = *Complex-Matrix.trace*
$(sum\text{-}mat\ (\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \otimes (B\ (i\ mod\ nD))) * R)\ \{..< nC * nD\})$
proof −
have $sum\text{-}mat\ (\lambda i.\ sum\text{-}mat\ (\lambda j.\ f\ i * g\ j \cdot_m (A\ i \otimes B\ j) * R)\ \{..<nD\})$
$\{..<nC\} = sum\text{-}mat\ (\lambda i.\ (f\ (i\ div\ nD) * g\ (i\ mod\ nD)) \cdot_m$
$((A\ (i\ div\ nD)) \otimes (B\ (i\ mod\ nD))) * R)\ \{..< nC * nD\}$
by (*rule sum-sum-mat-expand*, (*auto simp add*: *assms*))
thus *?thesis* **by** *simp*
qed
also have ... = $(\sum i{<}nC * nD.$
Complex-Matrix.trace
$(f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \otimes B\ (i\ mod\ nD)) * R))$
proof (*rule trace-sum-mat*)
fix *i*
assume $i \in \{..<nC * nD\}$
hence $i\ div\ nD < nC$
by (*simp add*: *less-mult-imp-div-less*)
hence $A\ (i\ div\ nD) \in carrier\text{-}mat\ n\ n$ **using** *assms* **by** *simp*
have $i\ mod\ nD < nD$ **using** *assms* **by** *simp*
hence $B\ (i\ mod\ nD) \in carrier\text{-}mat\ m\ m$ **using** *assms* **by** *simp*
hence $A\ (i\ div\ nD) \otimes B\ (i\ mod\ nD) \in carrier\text{-}mat\ (n{*}m)\ (n{*}m)$
using *tensor-mat-carrier*
by (*metis* ‹$A\ (i\ div\ nD) \in carrier\text{-}mat\ n\ n$›
carrier-matD(1) carrier-matD(2))
hence $(A\ (i\ div\ nD) \otimes B\ (i\ mod\ nD)) \in fc$
using *assms dim-eq fc-mats-carrier* **by** *blast*
hence $f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \otimes B\ (i\ mod\ nD)) \in fc$
using *smult-mem* **by** *blast*
thus $f\ (i\ div\ nD) * g\ (i\ mod\ nD) \cdot_m (A\ (i\ div\ nD) \otimes B\ (i\ mod\ nD)) * R \in fc$
using *assms(3) cpx-sq-mat-mult fc-mats-carrier* **by** *blast*
qed *simp*
also have ... = *Complex-Matrix.trace* $((C \otimes D) * R)$
proof (*rule tensor-mat-sum-mult-trace-expand*)
show $\bigwedge k.\ k < nC \Longrightarrow A\ k \in carrier\text{-}mat\ n\ n$ **using** *assms* **by** *simp*
show $\bigwedge j.\ j < nD \Longrightarrow B\ j \in carrier\text{-}mat\ m\ m$ **using** *assms* **by** *simp*
qed (*auto simp add*: *assms*)
finally show *?thesis* **.**
qed

lemma *tensor-mat-sum-mult-trace-eq-0*:
assumes $\bigwedge k.\ k < (nC{::}nat) \Longrightarrow A\ k \in carrier\text{-}mat\ n\ n$
and $R \in carrier\text{-}mat\ (n{*}m)\ (n{*}m)$
and $fixed\text{-}carrier\text{-}mat.sum\text{-}mat\ n\ n\ (\lambda i.\ f\ i \cdot_m (A\ i))\ \{..< nC\} = C$

and *fixed-carrier-mat.sum-mat m m* ($\lambda j.\ g\ j \cdot_m (B\ j)$) $\{..< nD\} = D$
and $0 < n$
and $0 < m$
and $0 = (nD::nat)$
shows *sum* ($\lambda i.$ (*sum* ($\lambda j.$ *Complex-Matrix.trace* (($f\ i * g\ j$)$\cdot_m$
(($A\ i$) $\bigotimes$ ($B\ j$)) $* R$)) $\{..< nD\}$)) $\{..< nC\}$ =
Complex-Matrix.trace (($C \bigotimes D$) $* R$)
proof −
define *fc*::*complex Matrix.mat set* **where** *fc* = *carrier-mat* ($n*m$) ($n*m$)
interpret *cpx-sq-mat* $n*m$ $n*m$ *fc*
proof
show $0 < n*m$ **using** *assms* **by** *simp*
qed (*auto simp add*: *fc-def*)
have *fixed-carrier-mat.sum-mat m m* ($\lambda j.\ g\ j \cdot_m (B\ j)$) $\{\} = 0_m\ m\ m$
using *assms fixed-carrier-mat.sum-mat-empty*[*of* - *m m*]
fixed-carrier-mat.intro **by** *fastforce*
hence $D = 0_m\ m\ m$ **using** *assms* **by** *simp*
hence $C \bigotimes D = C \bigotimes (0_m\ m\ m)$ **by** *simp*
also have ... = 0_m ($n*m$) ($n*m$)
proof (*rule tensor-mat-zero'*)
have *fixed-carrier-mat.sum-mat n n* ($\lambda i.\ f\ i \cdot_m A\ i$) $\{..<nC\} \in$
carrier-mat n n
proof (*rule fixed-carrier-mat.sum-mat-carrier*)
show *fixed-carrier-mat* (*carrier-mat n n*) *n n*
by (*simp add*: *fixed-carrier-mat.intro*)
show $\bigwedge i.\ i \in \{..<nC\} \Longrightarrow f\ i \cdot_m A\ i \in$ *carrier-mat n n* **using** *assms*
by *simp*
qed
thus $C \in$ *carrier-mat n n* **using** *assms* **by** *simp*
show $0<n$ $0<m$ **using** *assms* **by** *auto*
qed
finally have $C \bigotimes D = 0_m$ ($n*m$) ($n*m$) **.**
hence ($C \bigotimes D$) $* R = 0_m$ ($n*m$) ($n*m$)
by (*simp add*: *assms left-mult-zero-mat*)
hence *1*: *Complex-Matrix.trace* (($C \bigotimes D$) $* R$) = *0* **by** *simp*
have $\bigwedge i.\ i \in \{..< nC\} \Longrightarrow$ *sum* ($\lambda j.$ *Complex-Matrix.trace* (($f\ i * g\ j$)$\cdot_m$
(($A\ i$) $\bigotimes$ ($B\ j$)) $* R$)) $\{..< nD\}$ = *0*
proof −
fix *i*
assume $i \in \{..< nC\}$
show *sum* ($\lambda j.$ *Complex-Matrix.trace* (($f\ i * g\ j$)$\cdot_m$
(($A\ i$) $\bigotimes$ ($B\ j$)) $* R$)) $\{..< nD\}$ = *0* **using** *assms* **by** *simp*
qed
hence *sum* ($\lambda i.$ (*sum* ($\lambda j.$ *Complex-Matrix.trace* (($f\ i * g\ j$)$\cdot_m$
(($A\ i$) $\bigotimes$ ($B\ j$)) $* R$)) $\{..< nD\}$)) $\{..< nC\}$ = *0* **by** *simp*
thus *?thesis* **using** *1* **by** *simp*
qed

lemma *tensor-mat-sum-mult-trace*:

assumes $\bigwedge k.\ k < (nC::nat) \Longrightarrow A\ k \in carrier\text{-}mat\ n\ n$
and $\bigwedge j.\ j < (nD::nat) \Longrightarrow B\ j \in carrier\text{-}mat\ m\ m$
and $R \in carrier\text{-}mat\ (n*m)\ (n*m)$
and $fixed\text{-}carrier\text{-}mat.sum\text{-}mat\ n\ n\ (\lambda i.\ f\ i\ \cdot_m\ (A\ i))\ \{..< nC\} = C$
and $fixed\text{-}carrier\text{-}mat.sum\text{-}mat\ m\ m\ (\lambda j.\ g\ j\ \cdot_m\ (B\ j))\ \{..< nD\} = D$
and $0 < n$
and $0 < m$
shows $sum\ (\lambda i.\ (sum\ (\lambda j.\ Complex\text{-}Matrix.trace\ ((f\ i * g\ j)\cdot_m$
$((A\ i) \bigotimes (B\ j)) * R))\ \{..< nD\}))\ \{..< nC\} =$
$Complex\text{-}Matrix.trace\ ((C \bigotimes D) * R)$
proof $(cases\ nD = 0)$
case *True*
then show *?thesis* **using** *assms tensor-mat-sum-mult-trace-eq-0*$[OF\ assms(1)]$
by *simp*
next
case *False*
then show *?thesis*
using *assms tensor-mat-sum-mult-trace-ne-0*$[OF\ assms(1)\ assms(2)]$ **by** *simp*
qed

lemma *tensor-mat-make-pm-mult-trace*:
assumes $A \in carrier\text{-}mat\ n\ n$
and *hermitian A*
and $B \in carrier\text{-}mat\ m\ m$
and *hermitian B*
and $R \in carrier\text{-}mat\ (n*m)\ (n*m)$
and $(nA,\ M) = cpx\text{-}sq\text{-}mat.make\text{-}pm\ n\ n\ A$
and $(nB,\ N) = cpx\text{-}sq\text{-}mat.make\text{-}pm\ m\ m\ B$
and $0 < n$
and $0 < m$
shows $sum\ (\lambda i.\ (sum\ (\lambda j.\ Complex\text{-}Matrix.trace$
$((complex\text{-}of\text{-}real\ (meas\text{-}outcome\text{-}val\ (M\ i)) *$
$complex\text{-}of\text{-}real\ (meas\text{-}outcome\text{-}val\ (N\ j)))\cdot_m$
$((meas\text{-}outcome\text{-}prj\ (M\ i)) \bigotimes (meas\text{-}outcome\text{-}prj\ (N\ j))) * R))\ \{..< nB\}))$
$\{..< nA\} =$
$Complex\text{-}Matrix.trace\ ((A \bigotimes B) * R)$
proof $(rule\ tensor\text{-}mat\text{-}sum\text{-}mult\text{-}trace)$
have *A*: $cpx\text{-}sq\text{-}mat.proj\text{-}measurement\ n\ n\ (carrier\text{-}mat\ n\ n)\ nA\ M$
proof $(rule\ cpx\text{-}sq\text{-}mat.make\text{-}pm\text{-}proj\text{-}measurement)$
show $A \in carrier\text{-}mat\ n\ n$ **using** *assms* **by** *simp*
show $cpx\text{-}sq\text{-}mat\ n\ n\ (carrier\text{-}mat\ n\ n)$
by $(simp\ add{:}\ assms\ cpx\text{-}sq\text{-}mat.intro\ cpx\text{-}sq\text{-}mat\text{-}axioms.intro$
$fixed\text{-}carrier\text{-}mat\text{-}def)$
qed $(auto\ simp\ add{:}\ assms)$
have *B*: $cpx\text{-}sq\text{-}mat.proj\text{-}measurement\ m\ m\ (carrier\text{-}mat\ m\ m)\ nB\ N$
proof $(rule\ cpx\text{-}sq\text{-}mat.make\text{-}pm\text{-}proj\text{-}measurement)$
show $B \in carrier\text{-}mat\ m\ m$ **using** *assms* **by** *simp*
show $cpx\text{-}sq\text{-}mat\ m\ m\ (carrier\text{-}mat\ m\ m)$
by $(simp\ add{:}\ assms\ cpx\text{-}sq\text{-}mat.intro\ cpx\text{-}sq\text{-}mat\text{-}axioms.intro$

fixed-carrier-mat-def)
qed (*auto simp add*: *assms*)
show $\bigwedge k.\ k < nA \Longrightarrow$ *meas-outcome-prj* (*M k*) $\in$ *carrier-mat n n*
proof −
fix *k*
assume $k < nA$
show *meas-outcome-prj* (*M k*) $\in$ *carrier-mat n n*
using *cpx-sq-mat.proj-measurement-carrier*
by (*meson A* ‹$k < nA$› *assms*(*8*) *cpx-sq-mat-axioms.intro cpx-sq-mat-def*
fixed-carrier-mat.intro)
qed
show $\bigwedge k.\ k < nB \Longrightarrow$ *meas-outcome-prj* (*N k*) $\in$ *carrier-mat m m*
proof −
fix *k*
assume $k < nB$
show *meas-outcome-prj* (*N k*) $\in$ *carrier-mat m m*
using *cpx-sq-mat.proj-measurement-carrier*
by (*meson B* ‹$k < nB$› *assms*(*9*) *cpx-sq-mat-axioms.intro cpx-sq-mat-def*
fixed-carrier-mat.intro)
qed
show *fixed-carrier-mat.sum-mat n n*
(λi. *complex-of-real* (*meas-outcome-val* (*M i*)) $\cdot_m$ *meas-outcome-prj* (*M i*))
{..<*nA*} = *A*
proof (*rule cpx-sq-mat.make-pm-sum*)
show *cpx-sq-mat n n* (*carrier-mat n n*)
by (*simp add*: *assms cpx-sq-mat.intro cpx-sq-mat-axioms.intro*
fixed-carrier-mat-def)
qed (*auto simp add*: *assms*)
show *fixed-carrier-mat.sum-mat m m*
(λi. *complex-of-real* (*meas-outcome-val* (*N i*)) $\cdot_m$ *meas-outcome-prj* (*N i*))
{..<*nB*} = *B*
proof (*rule cpx-sq-mat.make-pm-sum*)
show *cpx-sq-mat m m* (*carrier-mat m m*)
by (*simp add*: *assms cpx-sq-mat.intro cpx-sq-mat-axioms.intro*
fixed-carrier-mat-def)
qed (*auto simp add*: *assms*)
qed (*auto simp add*: *assms*)

lemma *tensor-mat-mat-conj*:
assumes *A*∈ *carrier-mat n n*
and *B* ∈ *carrier-mat n n*
and *U* ∈ *carrier-mat n n*
and *C*∈ *carrier-mat m m*
and *D* ∈ *carrier-mat m m*
and *V* ∈ *carrier-mat m m*
and *0* < *n*
and *0* < *m*
and *A* = *mat-conj U B*
and *C* = *mat-conj V D*

shows $A \bigotimes C = mat\text{-}conj\ (U \bigotimes V)\ (B \bigotimes D)$
proof −
have $A \bigotimes C = (U * B * Complex\text{-}Matrix.adjoint\ U) \bigotimes$
$(V * D * Complex\text{-}Matrix.adjoint\ V)$ **using** *assms* **unfolding** *mat-conj-def*
by *simp*
also have ... $= (U{*}B \bigotimes (V{*}D)) *$
$(Complex\text{-}Matrix.adjoint\ U \bigotimes Complex\text{-}Matrix.adjoint\ V)$
using *mult-distr-tensor assms* **by** *simp*
also have ... $= (U \bigotimes V) * (B \bigotimes D) * Complex\text{-}Matrix.adjoint\ (U \bigotimes V)$
using *mult-distr-tensor assms*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *tensor-mat-adjoint*)
finally show *?thesis* **unfolding** *mat-conj-def* **by** *simp*
qed

lemma *unitarily-equiv-mat-conj*[*simp*]:
assumes *unitarily-equiv A B U*
shows $A = mat\text{-}conj\ U\ B$ **unfolding** *mat-conj-def*
by (*simp add*: *assms unitarily-equiv-eq*)

lemma *hermitian-tensor-mat-decomp*:
assumes $A{\in}$ *carrier-mat n n*
and $C{\in}$ *carrier-mat m m*
and *unitary-diag A B U*
and *unitary-diag C D V*
and $0 < n$
and $0 < m$
shows *unitary-diag* $(A \bigotimes C)\ (B \bigotimes D)\ (U \bigotimes V)$
proof (*rule unitary-diagI'*)
show $A \bigotimes C \in carrier\text{-}mat\ (n * m)\ (n{*}m)$ **using** *assms*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *tensor-mat-carrier*)
show $B \bigotimes D \in carrier\text{-}mat\ (n * m)\ (n * m)$ **using** *assms*
by (*metis* (*no-types*, *opaque-lifting*) *carrier-matD*(*1*)
carrier-matD(*2*) *carrier-mat-triv dim-col-tensor-mat*
dim-row-tensor-mat unitary-diag-carrier(*1*))
show *Complex-Matrix.unitary* $(U \bigotimes V)$
by (*metis Complex-Matrix.unitary-def assms*(*3*) *assms*(*4*)
carrier-matD(*2*) *carrier-mat-triv dim-col-tensor-mat dim-row-tensor-mat*
nat-0-less-mult-iff tensor-mat-unitary unitary-diagD(*3*) *unitary-zero*
zero-order(*5*))
show *diagonal-mat* $(B \bigotimes D)$ **using** *tensor-mat-diagonal*
by (*meson assms*(*3*) *assms*(*4*) *unitarily-equiv-carrier'*(*2*) *unitary-diagD*(*2*)
unitary-diag-imp-unitarily-equiv)
show $A \bigotimes C = mat\text{-}conj\ (U \bigotimes V)\ (B \bigotimes D)$
proof (*rule tensor-mat-mat-conj*[*of - n - - - m*])
show $B{\in}$ *carrier-mat n n*
using *assms*(*1*) *assms*(*3*) *unitary-diag-carrier*(*1*) **by** *auto*
show $D \in carrier\text{-}mat\ m\ m$
using *assms unitary-diag-carrier*(*1*) **by** *auto*
show $U \in carrier\text{-}mat\ n\ n$

using *assms(1) assms(3) unitary-diag-carrier(2)* **by** *blast*
show $V \in$ *carrier-mat m m*
using *assms unitary-diag-carrier(2)* **by** *blast*
qed (*auto simp add*: *assms*)
qed

end

theory *Matrix-L2-Operator-Norm*
imports
Tensor-Mat-Compl-Properties
begin

We formalize the $\mathcal{L}_2$ operator norm on matrices on nonempty vector spaces. This norm can be defined on a matrix A by $\|A\|_2 = \sup\{\|A \cdot v\|_2 \mid \|v\|_2 = 1\}$, and it is equal to the maximum singular value of A.

4 Preliminary results

4.1 Commutator and anticommutator

We define the notions of commutator and anticommutator of two matrices. When these matrices commute, their commutator is the zero matrix.

definition *commutator* :: *complex Matrix.mat* ⇒ *complex Matrix.mat* ⇒
complex Matrix.mat **where**
commutator A B = *A* ∗ *B* − *B* ∗ *A*

definition *anticommutator* **where**
anticommutator A B = *A* ∗ *B* + *B* ∗ *A*

lemma *commutator-dim*:
assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n n*
shows *commutator A B* ∈ *carrier-mat n n* **using** *assms* **unfolding** *commutator-def*
by (*metis minus-carrier-mat mult-carrier-mat*)

lemma *anticommutator-dim*:
assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n n*
shows *anticommutator A B* ∈ *carrier-mat n n* **using** *assms*
unfolding *anticommutator-def*
by (*metis add-carrier-mat mult-carrier-mat*)

lemma *commutator-zero-iff*:
assumes *A*∈ *carrier-mat n n*
and *B*∈ *carrier-mat n n*

shows *commutator A B* = 0_m *n n* $\longleftrightarrow$ $A*B = B * A$
proof −
have $A*B \in$ *carrier-mat n n* **using** *assms* **by** *simp*
moreover have $B*A\in$ *carrier-mat n n* **using** *assms* **by** *simp*
ultimately show *?thesis* **unfolding** *commutator-def*
by (*metis left-add-zero-mat mat-minus-minus minus-r-inv-mat*)
qed

lemma *anticommutator-zero-iff*:
fixes *A*::'*a* ::*ring Matrix.mat*
assumes *A*∈ *carrier-mat n n*
and *B*∈ *carrier-mat n n*
shows *anticommutator A B* = 0_m *n n* $\longleftrightarrow$ $B*A = -(A*B)$
proof −
have *ab*: $A*B \in$ *carrier-mat n n* **using** *assms* **by** *simp*
have *ba*: $B*A\in$ *carrier-mat n n* **using** *assms* **by** *simp*
show *?thesis* **unfolding** *anticommutator-def*
proof
assume $A * B + B * A = 0_m$ *n n*
thus $B*A = -(A*B)$ **using** *ab ba mat-add-eq-0-if* **by** *auto*
next
show $B * A = -(A * B) \Longrightarrow A * B + B * A = 0_m$ *n n* **using** *ab ba*
by (*metis uminus-l-inv-mat uminus-uminus-mat*)
qed
qed

lemma *commutator-mult-expand*:
assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n n*
and $C \in$ *carrier-mat n n*
and $D \in$ *carrier-mat n n*
shows *commutator A B* ∗ *commutator C D* =
$A * B * (C * D) - A * B * (D * C) - B * A * (C * D) + B * A * (D * C)$
proof −
have *commutator A B* ∗ *commutator C D* = *A* ∗ *B* ∗ *commutator C D* −
B ∗ *A* ∗ *commutator C D*
using *assms commutator-def*
minus-mult-distrib-mat[*of A* ∗ *B n n B* ∗ *A commutator C D*]
commutator-dim[*of C n D*] **by** *simp*
also have ... = *A* ∗ *B* ∗ (*C* ∗ *D*) − *A* ∗ *B* ∗ (*D* ∗ *C*) − *B* ∗ *A* ∗ *commutator C*
D
using *assms commutator-def*
mult-minus-distrib-mat[*of A* ∗ *B n n C* ∗ *D n D* ∗ *C*]
by *simp*
also have ... = *A* ∗ *B* ∗ (*C* ∗ *D*) − *A* ∗ *B* ∗ (*D* ∗ *C*) − *B* ∗ *A* ∗ (*C* ∗ *D*) +
B ∗ *A* ∗ (*D* ∗ *C*)
using *assms commutator-def*
mult-minus-distrib-mat[*of B* ∗ *A n n C* ∗ *D n D* ∗ *C*]
by (*auto simp add*: *algebra-simps*)

finally show *?thesis* **.**
qed

5 Maximum modulus in a spectrum

We prove some basic results on the maximum modulus of elements in a matrix A, and focus on the case where A is a Hermitian matrix.

5.1 Definition and basic properties for Hermitian matrices

definition *spmax*:: *complex Matrix.mat* ⇒ *real* **where**
spmax A = *Max.F* {*cmod a*|*a*. *a*∈ *spectrum A*}

lemma *spmax-mem*:
assumes *A* ∈ *carrier-mat n n*
and *0* < *n*
shows *spmax A* ∈ {*cmod a*|*a*. *a* ∈ *spectrum A*}
proof −
define *del* **where** *del* = {*cmod a*|*a*. *a* ∈ *spectrum A*}
define *M* **where** *M* = *Max.F del*
have *del* ≠ {} **using** *spectrum-ne assms* **unfolding** *del-def* **by** *auto*
moreover have ⋀*x*. *x* ∈ *del* ⟹ *0* ≤ *x* **unfolding** *del-def* **by** *force*
have *finite del* **using** *del-def* **by** (*simp add*: *spectrum-finite*)
hence *M* ∈ {*cmod a*|*a*. *a* ∈ *spectrum A*}
using *Max-in*[*of del*] ‹*del* ≠ {}› *M-def del-def* **by** *simp*
thus *?thesis* **unfolding** *spmax-def M-def del-def* **.**
qed

lemma *spmax-geq-0*:
assumes *A* ∈ *carrier-mat n n*
and *0* < *n*
shows *0* ≤ *spmax A*
proof −
define *del* **where** *del* = {*cmod a*|*a*. *a* ∈ *spectrum A*}
define *M* **where** *M* = *Max.F del*
have *del* ≠ {} **using** *spectrum-ne assms* **unfolding** *del-def* **by** *auto*
moreover have ⋀*x*. *x* ∈ *del* ⟹ *0* ≤ *x* **unfolding** *del-def* **by** *force*
have *finite del* **using** *del-def* **by** (*simp add*: *spectrum-finite*)
hence *M* ∈ *del* **using** *Max-in*[*of del*] ‹*del* ≠ {}› *M-def* **by** *simp*
hence *0* ≤ *M* **using** ‹⋀*x*. *x* ∈ *del* ⟹ *0* ≤ *x*› **by** *simp*
thus *?thesis* **unfolding** *spmax-def del-def M-def* **.**
qed

lemma *Re-inner-mult-diag-le*:
fixes *B*::*complex Matrix.mat*
assumes *diagonal-mat B*
and *B* ∈ *carrier-mat n n*
and *0* < *n*

```
  and M = Max.F {Re (conjugate a)|a. a∈ diag-elems B}
shows ∀ v ∈ carrier-vec n. Re (inner-prod (B *v v) v) ≤
  M * Re ((inner-prod v v))
proof −
  define del where del = {Re (conjugate a)|a. a∈ diag-elems B}
  have finite del using del-def by simp
  moreover have del ≠ {} using diag-elems-ne[of B] assms del-def by simp
  ultimately have M ∈ del using  Max-in[of del] del-def assms
    unfolding spmax-def by simp
  have ∀ v ∈ carrier-vec n. Re (inner-prod (B *v v) v) ≤
    M * Re ((inner-prod v v))
  proof
    fix v::complex Matrix.vec
    assume v ∈ carrier-vec n
    hence  Re (inner-prod (B *v v) v) =
      Re (∑ i ∈ {0 ..< n}. (conjugate (B $$ (i,i))) *
      (vec-index v i *  (conjugate (vec-index v i))))
      using assms inner-mult-diag-expand by simp
    also have ... = (∑ i ∈ {0 ..< n}. Re ((conjugate (B $$ (i,i))) *
      (vec-index v i *  (conjugate (vec-index v i))))) by simp
    also have ... ≤ (∑ i ∈ {0 ..< n}. M *
      Re (vec-index v i *  (conjugate (vec-index v i))))
    proof (rule sum-mono)
      show ⋀i. i ∈ {0..<n} ⟹  Re ((conjugate (B $$ (i,i))) *
      (vec-index v i *  (conjugate (vec-index v i)))) ≤
      M * Re  (vec-index v i *  (conjugate (vec-index v i)))
      proof −
        fix i
        assume i ∈ {0 ..< n}
        hence Re ((conjugate (B $$ (i,i))) *
          (vec-index v i *  (conjugate (vec-index v i)))) =
          Re (conjugate (B $$ (i,i))) * Re (vec-index v i *
          (conjugate (vec-index v i)))
          using real-mult-re mult-conj-real assms by auto
        also have ... ≤ M *  Re (vec-index v i * (conjugate (vec-index v i)))
        proof −
          have Re (conjugate (B $$ (i, i))) ∈ del using assms ‹i ∈ {0 ..< n}›
            unfolding del-def diag-elems-def by auto
          hence rel: Re (conjugate (B $$ (i, i))) ≤ M
            using assms ‹finite del› del-def by auto
          have 0 ≤ vec-index v i * conjugate (vec-index v i)
            using less-eq-complex-def by simp
          moreover have vec-index v i * conjugate (vec-index v i) =
            Re (vec-index v i * conjugate (vec-index v i))
            using mult-conj-real complex-is-Real-iff
            by (metis of-real-Re)
          ultimately have 0 ≤ Re (vec-index v i * conjugate (vec-index v i))
            by simp
          thus Re (conjugate (B $$ (i, i))) * Re (vec-index v i *
```

```
        conjugate (vec-index v i)) ≤
        M * Re (vec-index v i * conjugate (vec-index v i))
        using rel mult-right-mono by blast
    qed
    finally show
      Re ((conjugate (B $$ (i,i))) * (vec-index v i *
      (conjugate (vec-index v i)))) ≤
      M * Re (vec-index v i * conjugate (vec-index v i)) .
  qed
 qed
 also have ... = Re (∑ i ∈ {0 ..< n}. M *
  (vec-index v i *  (conjugate (vec-index v i))))
  by simp
 also have ... = Re (M * (inner-prod v v))
 proof −
  have (∑ i ∈ {0 ..< n}. M*(vec-index v i * (conjugate (vec-index v i))))=
    M * (∑ i ∈ {0 ..< n}. (vec-index v i *  (conjugate (vec-index v i))))
    by (simp add: sum-distrib-left)
  also have ... = M * (inner-prod v v) unfolding Matrix.scalar-prod-def
    using assms ‹v ∈ carrier-vec n› by force
  finally show ?thesis by simp
 qed
 also have ... = M * Re (inner-prod v v) using assms by simp
 finally show Re (Complex-Matrix.inner-prod (B *v v) v) ≤
  M * Re (Complex-Matrix.inner-prod v v) .
qed
thus ?thesis using assms by auto
qed

lemma Re-inner-mult-diag-le':
 fixes B::complex Matrix.mat
 assumes diagonal-mat B
 and B ∈ carrier-mat n n
 and 0 < n
 and (M::real) = Max.F {cmod a|a. a∈ diag-elems B}
 and v∈ carrier-vec n
shows cmod (inner-prod v (B *v v)) ≤ M * inner-prod v v
proof −
 define del where del = {cmod a|a. a∈ diag-elems B}
 have finite del using del-def by simp
 moreover have del ≠ {} using diag-elems-ne[of B] assms del-def by simp
 ultimately have M ∈ del using  Max-in[of del] del-def assms
  unfolding  spmax-def by simp
 have cmod (inner-prod v (B *v v)) = cmod (∑ i ∈ {0 ..< n}. B $$ (i,i) *
    (vec-index v i *  (conjugate (vec-index v i))))
  using assms inner-mult-diag-expand' by simp
 also have ... ≤ (∑ i ∈ {0 ..< n}. cmod (B $$ (i,i) *
  (vec-index v i *  (conjugate (vec-index v i)))))
  by (simp add: sum-norm-le)
```

```
  also have ... ≤ (∑ i ∈ {0 ..< n}. M *
    cmod (vec-index v i *  (conjugate (vec-index v i))))
  proof (rule sum-mono)
    show ⋀i. i ∈ {0..<n} ⟹  cmod (B $$ (i,i) *
    (vec-index v i *  (conjugate (vec-index v i)))) ≤
    M * cmod  (vec-index v i *  (conjugate (vec-index v i)))
    proof −
      fix i
      assume i ∈ {0 ..< n}
      hence cmod (B $$ (i,i) * (vec-index v i *(conjugate (vec-index v i)))) =
        cmod (B $$ (i,i)) * cmod (vec-index v i *  (conjugate (vec-index v i)))
        by (simp add: norm-mult)
      also have ... ≤ M *  cmod (vec-index v i *  (conjugate (vec-index v i)))
      proof −
        have cmod (B $$ (i, i)) ∈ del using assms ‹i ∈ {0 ..< n}›
          unfolding del-def diag-elems-def by auto
        hence  cmod (B $$ (i, i)) ≤ M using assms ‹finite del› del-def
          by auto
        thus ?thesis using mult-right-mono norm-ge-zero by blast
      qed
      finally show
        cmod (B $$ (i,i) * (vec-index v i *  (conjugate (vec-index v i)))) ≤
        M * cmod (vec-index v i * conjugate (vec-index v i)) .
    qed
  qed
  also have ... = (∑ i ∈ {0 ..< n}. M *
    (vec-index v i *  (conjugate (vec-index v i))))
    using cmod-conjugate-square-eq by auto
  also have ... = M * (∑ i ∈ {0 ..< n}.
    (vec-index v i *  (conjugate (vec-index v i))))
    by (simp add: sum-distrib-left)
  also have ... = M * (inner-prod v v) unfolding Matrix.scalar-prod-def
    using assms ‹v ∈ carrier-vec n› by force
  finally show ?thesis using less-eq-complex-def by simp
qed

lemma hermitian-mult-inner-prod-le:
  fixes A::complex Matrix.mat
  assumes A ∈ carrier-mat n n
  and 0 < n
  and hermitian A
  and v ∈ carrier-vec n
  shows cmod (inner-prod v (A *v v)) ≤ (spmax A) * (inner-prod v v)
proof −
  obtain B U where bu: real-diag-decomp A B U
    using assms hermitian-real-diag-decomp[of A] by auto
  define M where M = Max.F {cmod a|a. a∈ diag-elems B}
  have meq: M = spmax A unfolding spmax-def M-def
    using unitary-diag-spectrum-eq'[of A] bu
```

by (*metis assms*(1) *real-diag-decompD*(1))
have *uc*: *Complex-Matrix.adjoint U* ∈ *carrier-mat n n* **using** *bu*
by (*meson adjoint-dim′ assms*(1) *real-diag-decompD*(1)
unitary-diag-carrier(2))
hence *mv*: *U* ∗ *B* ∗ (*Complex-Matrix.adjoint U*) ∗$_v$ *v* =
U ∗$_v$ (*B* ∗$_v$ ((*Complex-Matrix.adjoint U*) ∗$_v$ *v*))
using *assoc-mat-mult-vec′*[*of U*] *assms bu real-diag-decompD*(1)
unitary-diag-carrier(1)
by (*metis Complex-Matrix.adjoint-adjoint adjoint-dim*)
have *inner-prod v* (*A* ∗$_v$ *v*) = *inner-prod v* (*U* ∗$_v$ (*B* ∗$_v$
((*Complex-Matrix.adjoint U*) ∗$_v$ *v*)))
using *unitarily-equiv-eq bu mv real-diag-decompD*(1)
by (*metis unitary-diag-imp-unitarily-equiv*)
also have ... = *inner-prod* ((*Complex-Matrix.adjoint U*) ∗$_v$ *v*)
(*B* ∗$_v$ ((*Complex-Matrix.adjoint U*) ∗$_v$ *v*))
proof (*rule adjoint-def-alter*)
show *v*∈ *carrier-vec n* **using** ‹*v*∈ *carrier-vec n*› **.**
show *U* ∈ *carrier-mat n n* **using** *uc*
by (*metis Complex-Matrix.adjoint-adjoint adjoint-dim′*)
have (*Complex-Matrix.adjoint U* ∗$_v$ *v*) ∈ *carrier-vec n*
using ‹*Complex-Matrix.adjoint U* ∈ *carrier-mat n n*› ‹*v*∈ *carrier-vec n*›
by *simp*
thus *B* ∗$_v$ (*Complex-Matrix.adjoint U* ∗$_v$ *v*) ∈ *carrier-vec n*
using *assms bu real-diag-decompD*(1) *unitary-diag-carrier*(1)
by (*metis mult-mat-vec-carrier*)
qed
finally have *inner-prod v* (*A* ∗$_v$ *v*) =
inner-prod ((*Complex-Matrix.adjoint U*) ∗$_v$ *v*)
(*B* ∗$_v$ ((*Complex-Matrix.adjoint U*) ∗$_v$ *v*)) **.**
hence *cmod* (*inner-prod v* (*A* ∗$_v$ *v*)) =
cmod (*inner-prod* ((*Complex-Matrix.adjoint U*) ∗$_v$ *v*)
(*B* ∗$_v$ ((*Complex-Matrix.adjoint U*) ∗$_v$ *v*))) **by** *simp*
also have ... ≤ *M* ∗ *inner-prod* (*Complex-Matrix.adjoint U* ∗$_v$ *v*)
(*Complex-Matrix.adjoint U* ∗$_v$ *v*)
proof (*rule Re-inner-mult-diag-le′*)
show *bc*: *B*∈ *carrier-mat n n*
using *assms bu real-diag-decompD*(1) *unitary-diag-carrier*(1) **by** *metis*
show *M* = *Max.F* {*cmod a* |*a*. *a* ∈ *diag-elems B*}
using *M-def* **by** *simp*
show (*Complex-Matrix.adjoint U* ∗$_v$ *v*) ∈ *carrier-vec n*
using ‹*Complex-Matrix.adjoint U* ∈ *carrier-mat n n*› ‹*v*∈ *carrier-vec n*›
by *simp*
show *diagonal-mat B* **using** *assms bu real-diag-decompD*(1) *unitary-diagD*(2)
by *metis*
show 0 < *n* **using** *assms* **by** *simp*
qed
also have ... = *M* ∗ *inner-prod v v*
proof −
have *unitary U* **using** *bu unitarily-equivD*(1)

```
    using real-diag-decompD(1) unitary-diagD(3) by blast
  thus ?thesis using  assms bu uc unitary-inner-prod
    by (metis Complex-Matrix.adjoint-adjoint adjoint-dim' unitary-adjoint
        unitary-inner-prod)
 qed
 finally show ?thesis using meq by simp
qed

lemma  hermitian-trace-rank-le:
assumes A ∈ carrier-mat n n
 and hermitian A
 and v ∈ carrier-vec n
 and 0 < n
shows cmod (Complex-Matrix.trace (A * (rank-1-proj v))) ≤
 (spmax A) * (inner-prod v v)
 using assms  hermitian-mult-inner-prod-le
 by (metis rank-1-proj-trace-inner)

lemma hermitian-pos-decomp-cmod-le:
 assumes A∈ carrier-mat n n
 and C∈ carrier-mat n n
 and 0 < n
and hermitian C
and Complex-Matrix.positive A
shows cmod (Complex-Matrix.trace (C * A)) ≤
 Re (Complex-Matrix.trace A) * (spmax C)
proof −
 have a: A∈ carrier-mat n n using assms by simp
 have b: C∈ carrier-mat n n using assms by simp
 have 0 < n using assms by simp
 obtain B U where bu: real-diag-decomp A B U
   using hermitian-real-diag-decomp[of A] positive-is-hermitian assms by auto
 have ud: unitary-diag A B U using bu by simp
 have Complex-Matrix.positive A using assms by simp
 {
   fix i
   assume i <n
   have cmod (Complex-Matrix.trace (C * rank-1-proj (Matrix.col U i))) ≤
     spmax C * inner-prod (Matrix.col U i) (Matrix.col U i)
   proof (rule hermitian-trace-rank-le)
     show C∈ carrier-mat n n hermitian C using assms by simp+
     show Matrix.col U i ∈ carrier-vec n using
         unitary-diag-carrier(2) assms ud
       by (metis carrier-dim-vec carrier-matD(1) dim-col)
   qed (simp add: assms)
   also have ... = spmax C
   proof −
     have inner-prod (Matrix.col U i) (Matrix.col U i) = ‖Matrix.col U i‖²
       using vec-norm-sq-cpx-vec-length-sq inner-prod-vec-norm-pow2 by auto
```

```
    also have ... = 1 using unitary-col-norm-square assms unitary-diagD(3)
      unitary-diag-carrier(2) ud
      by (metis ‹i < n› of-real-eq-1-iff)
    finally show ?thesis by simp
  qed
  finally have
    cmod (Complex-Matrix.trace (C * rank-1-proj (Matrix.col U i))) ≤ spmax C
    using less-eq-complex-def by simp
 } note mprop = this
 thus cmod (Complex-Matrix.trace (C * A)) ≤
   Re (Complex-Matrix.trace A) * spmax C
   using a b ud positive-decomp-cmod-le  assms by simp
qed

lemma hermitian-density-cmod-le:
 fixes R::complex Matrix.mat
 assumes R∈ carrier-mat n n
 and A∈ carrier-mat n n
 and 0 < n
and hermitian A
and density-operator R
shows cmod (Complex-Matrix.trace (A * R)) ≤ (spmax A)
proof −
 have cmod (Complex-Matrix.trace (A * R)) ≤
   Re (Complex-Matrix.trace R) * (spmax A)
   using hermitian-pos-decomp-cmod-le assms unfolding density-operator-def
   by blast
 also have ... = spmax A using assms unfolding density-operator-def by simp
 finally show ?thesis .
qed

lemma tensor-mat-hermitian-positive-le:
 assumes A∈ carrier-mat n n
 and B∈ carrier-mat m m
 and C∈ carrier-mat n n
 and D∈ carrier-mat m m
 and 0 < n
 and 0 < m
 and hermitian A
 and hermitian B
 and Complex-Matrix.positive C
 and Complex-Matrix.positive D
shows cmod (Complex-Matrix.trace ((A⨂ B)*(C⨂ D))) ≤
 Re (Complex-Matrix.trace C) * Re (Complex-Matrix.trace D) *
 spmax A * spmax B
proof −
 have Complex-Matrix.trace ((A⨂ B)*(C⨂ D)) =
   Complex-Matrix.trace ((A*C) ⨂ (B*D))
   using mult-distr-tensor assms
```

```
    by (metis carrier-matD(2) positive-dim-eq)
  also have ... =
    Complex-Matrix.trace (A * C) * (Complex-Matrix.trace (B * D))
    using assms tensor-mat-trace by (meson mult-carrier-mat)
  finally have Complex-Matrix.trace ((A⨂B)*(C⨂D)) =
    Complex-Matrix.trace (A * C) * (Complex-Matrix.trace (B * D)) .
  hence cmod (Complex-Matrix.trace ((A⨂B)*(C⨂D))) =
    cmod (Complex-Matrix.trace (A * C)) *
    cmod (Complex-Matrix.trace (B * D))
    by (simp add: norm-mult)
  also have ... ≤ Re (Complex-Matrix.trace C) * spmax A *
    cmod (Complex-Matrix.trace (B * D))
    by (meson assms(1) assms(3) assms(5) assms(7) assms(9)
        hermitian-pos-decomp-cmod-le mult-right-mono norm-ge-zero)
  also have ... ≤ Re (Complex-Matrix.trace C) * spmax A *
    Re (Complex-Matrix.trace D) * spmax B
  proof −
    have 0 ≤ Re (Complex-Matrix.trace C) * spmax A
      using assms positive-trace spmax-geq-0
      by (simp add: cpx-ge-0-real nonnegative-complex-is-real)
    moreover have cmod (Complex-Matrix.trace (B * D)) ≤
      Re (Complex-Matrix.trace D) * spmax B
      using assms hermitian-pos-decomp-cmod-le by auto
    ultimately show ?thesis
      by (metis Groups.mult-ac(2) Groups.mult-ac(3) mult-left-mono)
  qed
  also have ... = Re (Complex-Matrix.trace C) *
    Re (Complex-Matrix.trace D) * spmax A * spmax B by simp
  finally show ?thesis .
qed

lemma tensor-mat-hermitian-density-le:
  assumes A∈ carrier-mat n n
  and B∈ carrier-mat m m
  and C∈ carrier-mat n n
  and D∈ carrier-mat m m
  and 0 < n
  and 0 < m
  and hermitian A
  and hermitian B
  and density-operator C
  and density-operator D
shows cmod (Complex-Matrix.trace ((A⨂B)*(C⨂D))) ≤
  spmax A * spmax B
proof −
  have cmod (Complex-Matrix.trace ((A⨂B)*(C⨂D))) ≤
  Re (Complex-Matrix.trace C) * Re (Complex-Matrix.trace D) *
  spmax A * spmax B
    by (meson assms density-operator-def tensor-mat-hermitian-positive-le)
```

moreover have *Complex-Matrix.trace C = 1*
using *assms* **unfolding** *density-operator-def* **by** *simp*
moreover have *Complex-Matrix.trace D = 1*
using *assms* **unfolding** *density-operator-def* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed

lemma *idty-spmax*:
assumes *0 < n*
shows *spmax* $(1_m\ n) = 1$ **using** *idty-spectrum assms* **unfolding** *spmax-def* **by** *simp*

lemma *spmax-uminus*:
fixes *A*::*complex Matrix.mat*
assumes *hermitian A*
and *A*∈ *carrier-mat n n*
and *0 < n*
shows *spmax* $(-A) =$ *spmax A*
proof −
have {*cmod a* |*a*. *a* ∈ *spectrum* $(-A)$} = {*cmod* $(-a)$ |*a*. *a* ∈ *spectrum A*}
using *assms spectrum-uminus*[*of A n*]
by (*smt* (*verit*) *Collect-cong mem-Collect-eq*)
also have ... = {*cmod a* |*a*. *a* ∈ *spectrum A*} **by** *simp*
finally have {*cmod a* |*a*. *a* ∈ *spectrum* $(-A)$} =
{*cmod a* |*a*. *a* ∈ *spectrum A*} **.**
thus *?thesis* **unfolding** *spmax-def* **by** *simp*
qed

lemma *spmax-smult*:
fixes *A*::*complex Matrix.mat*
assumes *hermitian A*
and *A*∈ *carrier-mat n n*
and *0 < n*
shows *spmax* $(x \cdot_m A) =$ *cmod x* ∗ *spmax A*
proof −
have {*cmod a* |*a*. *a* ∈ *spectrum* $(x \cdot_m A)$} = {*cmod* $(x{*}a)$ |*a*. *a* ∈ *spectrum A*}
using *assms spectrum-smult*[*of A n*] **by** *auto*
also have ... = {*cmod x* ∗ *cmod a* |*a*. *a* ∈ *spectrum A*}
by (*simp add*: *norm-mult*)
finally have *eq*: {*cmod a* |*a*. *a* ∈ *spectrum* $(x \cdot_m A)$} =
{*cmod x* ∗ *cmod a* |*a*. *a* ∈ *spectrum A*} **.**
have ∀ *b* ∈ {*cmod a* |*a*. *a* ∈ *spectrum A*}. *0* ≤ *b* **by** *auto*
moreover have *finite* {*cmod a* |*a*. *a* ∈ *spectrum A*}
by (*simp add*: *spectrum-finite*)
moreover have {*cmod a* |*a*. *a* ∈ *spectrum A*} ≠ {}
using *assms spectrum-ne* **by** *fastforce*

```
  moreover have {cmod x * a |a. a ∈ {cmod a |a. a ∈ spectrum A}} =
    {cmod x * cmod a |a. a ∈ spectrum A} by auto
  ultimately have Max.F {cmod x * cmod a |a. a ∈ spectrum A} =
    cmod x * Max.F {cmod a |a. a ∈ spectrum A}
    using pos-mult-Max[of {cmod a |a. a ∈ spectrum A}]
    by (smt (verit) Collect-cong norm-ge-zero)
  thus ?thesis using eq unfolding spmax-def by auto
qed

lemma spmax-smult-pos:
fixes A::complex Matrix.mat
  assumes hermitian A
  and A∈ carrier-mat n n
  and 0 < n
  and 0 ≤ x
  shows spmax (x ·_m A) = x * spmax A
proof −
  have spmax (x ·_m A) = cmod x * spmax A
    using assms spmax-smult by simp
  also have ... = x * spmax A using assms by simp
  finally show ?thesis .
qed

lemma hermitian-square-spmax:
fixes A::complex Matrix.mat
  assumes hermitian A
and A∈ carrier-mat n n
and 0 < n
shows spmax (A*A) = spmax A * spmax A
proof −
  have spmax (A * A) = Max.F {cmod (a*a) |a. a ∈ spectrum A}
  proof −
    have {cmod a|a. a ∈ spectrum (A*A)} = {cmod (a*a) |a. a ∈ spectrum A}
      using assms hermitian-square-spectrum-eq[of A n]  by auto
    thus ?thesis unfolding spmax-def by simp
  qed
  also have ... = Max.F {cmod a * cmod a|a. a∈ spectrum A}
    by (simp add: norm-mult)
  also have ... = spmax A * spmax A unfolding spmax-def
  proof (rule square-Max)
    show finite (spectrum A) using spectrum-finite[of A] by simp
    show spectrum A ≠ {} using spectrum-ne[of A] assms by simp
  qed auto
  finally show ?thesis .
qed

lemma hermitian-square-idty-spmax:
  assumes 0 < n
  and A∈ carrier-mat n n
```

and *hermitian A*
and $A*A = 1_m\ n$
shows *spmax A = 1*
proof −
have *spmax A ∗ spmax A = 1*
using *hermitian-square-spmax*[*of A*] *assms idty-spmax* **by** *simp*
thus *spmax A = 1*
using *spmax-geq-0 assms*
by (*metis abs-of-nonneg le-numeral-extra*(*1*) *more-arith-simps*(*6*)
real-sqrt-abs2)
qed

lemma *hermitian-mult-density-trace*:
assumes *A*∈ *carrier-mat n n*
and *R*∈ *carrier-mat n n*
and *0 < n*
and *hermitian A*
and $A * A = 1_m\ n$
and *density-operator R*
shows |*Complex-Matrix.trace* (*A*∗*R*)| ≤ *1*
proof −
have *spmax A = 1* **using** *assms hermitian-square-idty-spmax* **by** *simp*
hence *cmod* (*Complex-Matrix.trace* (*A*∗*R*)) ≤ *1*
using *hermitian-density-cmod-le assms* **by** *metis*
thus *?thesis* **using** *abs-cmod-eq*
by (*metis Reals-of-real abs-norm-cancel cpx-real-abs-eq*
cpx-real-abs-leq of-real-1)
qed

lemma *tensor-mat-hermitian-density-spmax-le*:
assumes *A*∈ *carrier-mat n n*
and *B*∈ *carrier-mat m m*
and *C*∈ *carrier-mat n n*
and *D*∈ *carrier-mat m m*
and *0 < n*
and *0 < m*
and *hermitian A*
and *hermitian B*
and $A * A = 1_m\ n$
and $B * B = 1_m\ m$
and *density-operator C*
and *density-operator D*
shows *cmod* (*Complex-Matrix.trace* ((*A*⨂ *B*)∗(*C*⨂ *D*))) ≤ *1*
proof −
have *cmod* (*Complex-Matrix.trace* ((*A*⨂ *B*)∗(*C*⨂ *D*))) ≤
spmax A ∗ spmax B
using *tensor-mat-hermitian-density-le assms* **by** *simp*
moreover have *spmax A = 1*
by (*metis assms*(*1*) *assms*(*5*) *assms*(*7*) *assms*(*9*)

hermitian-square-spmax idty-spmax less-1-mult
linorder-le-less-linear mult-eq-1 semiring-norm(138)
spmax-geq-0)
moreover have *spmax B = 1*
by (*metis assms(10) assms(2) assms(6) assms(8)*
hermitian-square-spmax idty-spmax less-1-mult
linorder-le-less-linear mult-eq-1 semiring-norm(138)
spmax-geq-0)
ultimately show *?thesis* **by** *simp*
qed

5.2 Eigenvector for the element with maximum modulus

definition *spmax-wit* **where**
spmax-wit A = (*SOME k. eigenvalue A k* ∧ *spmax A* = *cmod k*)

lemma *spmax-wit-eigenvalue*:
assumes *A*∈ *carrier-mat n n*
and *0* < *n*
shows *eigenvalue A* (*spmax-wit A*) ∧ *spmax A* = *cmod* (*spmax-wit A*)
proof −
let *?V* = *SOME k. eigenvalue A k* ∧ *spmax A* = *cmod k*
have *vprop*: *eigenvalue A ?V* ∧ *spmax A* = *cmod ?V* **using**
someI-ex[*of* λ*h. eigenvalue A h* ∧ *spmax A* = *cmod h*]
spmax-def spmax-mem assms spectrum-eigenvalues
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
thus *?thesis* **using** *spmax-wit-def* **by** *simp*
qed

lemma *find-eigen-spmax-neq-0*:
assumes *A*∈ *carrier-mat n n*
and *0* < *n*
shows *find-eigenvector A* (*spmax-wit A*) ≠ 0_v *n* **using**
find-eigenvector assms spmax-wit-eigenvalue **unfolding** *eigenvector-def*
by *blast*

lemma *find-eigen-spmax-dim*:
assumes *A*∈ *carrier-mat n n*
and *0* < *n*
shows *dim-vec* (*vec-normalize* (*find-eigenvector A* (*spmax-wit A*))) = *n*
using *find-eigenvector assms spmax-wit-eigenvalue*
unfolding *eigenvector-def*
by (*metis carrier-dim-vec carrier-matD*(*1*) *carrier-vec-dim-vec*
normalized-vec-dim)

lemma *nrm-spmax-eigenvector-eq*:
assumes *v* = *vec-normalize* (*find-eigenvector A* (*spmax-wit A*))
and *A*∈ *carrier-mat n n*
and *0* < *n*

shows *cmod* (*inner-prod v* (*A* $*_v$ *v*)) = *spmax A*
proof −
define *ve* **where** *ve* = *find-eigenvector A* (*spmax-wit A*)
have *ve* ≠ 0_v *n* **using** *assms find-eigen-spmax-neq-0* **unfolding** *ve-def* **by** *simp*
have *dim-vec ve* = *n* **using** *assms*(*2*) *assms*(*3*) *spmax-wit-eigenvalue*
unfolding *ve-def eigenvector-def*
by (*metis find-eigen-spmax-dim index-smult-vec*(*2*)
vec-eq-norm-smult-normalized)
have *eigenvector A*
(*vec-normalize* (*find-eigenvector A* (*spmax-wit A*))) (*spmax-wit A*)
proof (*rule normalize-keep-eigenvector*)
show *eigenvector A* (*find-eigenvector A* (*spmax-wit A*)) (*spmax-wit A*)
using *assms*
find-eigenvector spmax-wit-eigenvalue
unfolding *eigenvector-def* **by** *blast*
show *A* ∈ *carrier-mat n n* **using** *assms* **by** *simp*
show *find-eigenvector A* (*spmax-wit A*) ∈ *carrier-vec n* **using** *assms*
find-eigenvector spmax-wit-eigenvalue
unfolding *eigenvector-def* **by** *blast*
qed
hence *inner-prod v* (*A* $*_v$ *v*) = *inner-prod v* ((*spmax-wit A*)·$_v$ *v*)
using *assms*
unfolding *eigenvector-def* **by** *force*
also have ... = *spmax-wit A* ∗ *inner-prod v v* **by** *simp*
also have ... = *spmax-wit A* **using** *normalized-vec-norm*[*of ve*] *assms*
‹*dim-vec ve* = *n*› ‹*ve* ≠ 0_v *n*› *carrier-vec-dim-vec mult-cancel-left2 ve-def*
by *blast*
finally have *inner-prod v* (*A* $*_v$ *v*) = *spmax-wit A* **.**
thus *cmod* (*inner-prod v* (*A* $*_v$ *v*)) = *spmax A*
using *assms*(*2*) *assms*(*3*) *spmax-wit-eigenvalue* **by** *presburger*
qed

6 The $\mathcal{L}_2$ operator norm

6.1 Definition and preliminary results

definition *rvec-norm* **where**
rvec-norm v = *Re* (*vec-norm v*)

definition *L2-op-nrm* **where**
L2-op-nrm A =
Sup {*rvec-norm* (*A* $*_v$ *v*) |*v*. *dim-vec v* = *dim-col A* ∧ *rvec-norm v* = *1*}

lemma *mat-mult-inner-prod-le*:
fixes *A*::*complex Matrix.mat*
assumes *0* < *dim-col A*
and *v* ∈ *carrier-vec* (*dim-col A*)
shows *cmod* (*inner-prod* (*A* $*_v$ *v*) (*A* $*_v$ *v*)) ≤

```
  spmax ((Complex-Matrix.adjoint A) * A) * (inner-prod v v)
proof −
  define dimr where dimr = dim-col A
  define fc::complex Matrix.mat set
    where fc = carrier-mat (dim-col A) (dim-col A)
  interpret cpx-sq-mat dim-col A dim-col A fc
  proof
    show 0 < dim-col A using assms by simp
  qed (auto simp add: fc-def)
  define C where C = (Complex-Matrix.adjoint A) * A
  have hermitian C
    using mult-adjoint-hermitian[of A dim-row A dim-col A] C-def by simp
  have C ∈ fc using assms adjoint-dim' fc-def C-def
    by (metis carrier-mat-triv mult-carrier-mat)
  have cmod (inner-prod (A *v v) (A *v v)) = cmod (inner-prod v (C *v v))
    unfolding C-def
    using inner-prod-mult-mat-vec-right ‹v∈ carrier-vec (dim-col A)›
    by (metis carrier-mat-triv)
  also have ... ≤ (spmax C) * inner-prod v v
    using hermitian-mult-inner-prod-le ‹C∈ fc› ‹hermitian C›
    ‹v∈ carrier-vec (dim-col A)›
    by (metis assms(1) fc-mats-carrier)
  finally show cmod (inner-prod (A *v v) (A *v v)) ≤
    spmax ((Complex-Matrix.adjoint A) * A) * (inner-prod v v)
    unfolding C-def by simp
qed

lemma normalized-rvec-norm:
  assumes v ≠ 0v (dim-vec v)
  shows rvec-norm (vec-normalize v) = 1
  using normalized-vec-norm assms carrier-vec-dim-vec csqrt-eq-1
  unfolding vec-norm-def rvec-norm-def
  by (metis one-complex.sel(1))

lemma vec-norm-smult:
  shows vec-norm (c ·v v) = (cmod c) * (vec-norm v)
proof −
  have Complex-Matrix.inner-prod (c ·v v) (c ·v v) =
    conjugate c * c * Complex-Matrix.inner-prod v v
    unfolding vec-norm-def using inner-prod-smult-right
    by (simp add: conjugate-smult-vec)
  also have ... = (cmod c)^2 * Complex-Matrix.inner-prod v v
    by (metis cross3-simps(11) mult-conj-cmod-square)
  finally have eq: Complex-Matrix.inner-prod (c ·v v) (c ·v v) =
    (cmod c)^2 * Complex-Matrix.inner-prod v v .
  have (cmod c)^2 * Complex-Matrix.inner-prod v v ∈ Reals
    using self-inner-prod-real by simp
  hence csqrt ((cmod c)^2 * Complex-Matrix.inner-prod v v) =
    sqrt (Re ((cmod c)^2 * Complex-Matrix.inner-prod v v))
```

```
    using self-cscalar-prod-geq-0 by auto
  also have ... = (sqrt (Re (cmod c)^2) *
    sqrt (Re (Complex-Matrix.inner-prod v v)))
    by (simp add: real-sqrt-mult)
  also have ... = cmod c * sqrt (Re (Complex-Matrix.inner-prod v v))
    by fastforce
  also have ... = cmod c * csqrt (Complex-Matrix.inner-prod v v) by auto
  finally have csqrt ((cmod c)^2 * Complex-Matrix.inner-prod v v) =
    cmod c * csqrt (Complex-Matrix.inner-prod v v) .
  hence csqrt (Complex-Matrix.inner-prod (c ·_v v) (c ·_v v)) =
    cmod c * csqrt (Complex-Matrix.inner-prod v v) using eq by simp
  thus ?thesis unfolding vec-norm-def by simp
qed

lemma rvec-norm-smult:
  shows rvec-norm (c ·_v v) = (cmod c) * (rvec-norm v)
  using vec-norm-smult unfolding rvec-norm-def by simp

lemma mult-mat-zero-vec:
  assumes A ∈ carrier-mat n m
and v = 0_v m
shows A *_v v = 0_v n
proof (intro eq-vecI)
  show dim-vec (A *_v v) = dim-vec (0_v n) using assms by simp
next
  fix i
  assume i < dim-vec (0_v n)
  hence Matrix.vec-index (A *_v v) i = Matrix.scalar-prod (Matrix.row A i) v
    using assms by simp
  also have ... = 0 using assms by auto
  also have ... = Matrix.vec-index (0_v n) i using ‹i < dim-vec (0_v n)›
    by auto
  finally show Matrix.vec-index (A *_v v) i = Matrix.vec-index (0_v n) i .
qed

lemma mat-mult-vec-normalize:
  assumes dim-col A = dim-vec v
  shows A *_v v = vec-norm v ·_v (A *_v (vec-normalize v))
proof−
  have A *_v v = A *_v (vec-norm v ·_v vec-normalize v)
    using vec-eq-norm-smult-normalized by simp
  also have ... = vec-norm v ·_v (A *_v (vec-normalize v))
    using mult-mat-vec[of A - - vec-normalize v vec-norm v] assms
    by (metis carrier-mat-triv carrier-vec-dim-vec normalized-vec-dim)
  finally show ?thesis .
qed

lemma vec-norm-real:
  shows vec-norm v ∈ Reals
```

proof −
have *Im* (*vec-norm v*) = *0* **using** *vec-norm-geq-0 less-eq-complex-def* **by** *force*
thus *?thesis* **using** *complex-is-Real-iff* **by** *auto*
qed

lemma *rvec-norm-geq-0*:
shows *0* ≤ *rvec-norm v* **unfolding** *rvec-norm-def*
using *vec-norm-geq-0 less-eq-complex-def* **by** *auto*

lemma *rvec-norm-triangle*:
assumes *dim-vec u* = *dim-vec v*
shows *rvec-norm* (*u* + *v*) ≤ *rvec-norm u* + *rvec-norm v*
using *vec-norm-triangle*[*OF assms*] *less-eq-complex-def*
unfolding *rvec-norm-def* **by** *simp*

lemma *cmod-vec-norm*:
shows *cmod* (*vec-norm v*) = *vec-norm v*
proof −
have *cmod* (*vec-norm v*) = *sqrt* ((*Re* (*vec-norm v*))^*2*) **using** *vec-norm-real*
by (*simp add*: *in-Reals-norm*)
also have ... = *Re* (*vec-norm v*)
using *vec-norm-real vec-norm-geq-0 cpx-ge-0-real* **by** *simp*
also have ... = *vec-norm v* **using** *vec-norm-real* **by** *simp*
finally show *?thesis* **.**
qed

lemma *cmod-rvec-norm*:
shows *cmod* (*rvec-norm v*) = *rvec-norm v*
unfolding *rvec-norm-def* **using** *cmod-vec-norm*
by (*metis Re-complex-of-real*)

lemma *inner-prod-rvec-norm-pow2*:
shows (*rvec-norm v*)^2 = *v* ·*c v*
using *rvec-norm-def inner-prod-vec-norm-pow2 vec-norm-eq-cpx-vec-length*
by *auto*

lemma *rvec-norm-mat-mult-le*:
assumes *v* ∈*carrier-vec* (*dim-col A*)
and *0* < *dim-col A*
shows *cmod* (*inner-prod* (*A* ∗$_v$ *v*) (*A* ∗$_v$ *v*))
≤ *spmax* (*Complex-Matrix.adjoint A* ∗ *A*) ∗ (*rvec-norm v*)^2
proof −
have *cmod* (*inner-prod* (*A* ∗$_v$ *v*) (*A* ∗$_v$ *v*)) ≤
spmax (*Complex-Matrix.adjoint A* ∗ *A*) ∗ *inner-prod v v*
using *assms mat-mult-inner-prod-le*[*of A v*] **by** *simp*
also have ... = *spmax* (*Complex-Matrix.adjoint A* ∗ *A*) ∗ (*rvec-norm v*)^2
using *cmod-rvec-norm inner-prod-rvec-norm-pow2 norm-power* **by** *simp*
finally show *?thesis* **using** *less-eq-complex-def* **by** *simp*
qed

lemma *square-leq*:
assumes $a^2 \leq b * c^2$
and $0 \leq c$
shows $a \leq (sqrt\ b) * c$
by (*metis assms real-le-rsqrt real-sqrt-mult real-sqrt-unique*)

lemma *rvec-set-ne*:
assumes $0 < dim\text{-}col\ A$
shows $\{rvec\text{-}norm\ (A *_v v) | v.\ dim\text{-}vec\ v = dim\text{-}col\ A \wedge rvec\text{-}norm\ v = 1\} \neq \{\}$
proof −
define *vn*::*complex Matrix.vec* **where** $vn = unit\text{-}vec\ (dim\text{-}col\ A)\ 0$
have $vn \neq 0_v\ (dim\text{-}vec\ vn)$ **unfolding** *vn-def* **using** *assms* **by** *simp*
hence $rvec\text{-}norm\ (vec\text{-}normalize\ vn) = 1$ **using** *normalized-rvec-norm* **by** *simp*
moreover have $dim\text{-}vec\ (vec\text{-}normalize\ vn) = dim\text{-}col\ A$ **unfolding** *vn-def*
by *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *unitary-col-vec-norm*:
assumes $U \in carrier\text{-}mat\ n\ n$
and *unitary U*
and $i < n$
shows $vec\text{-}norm\ (Matrix.col\ U\ i) = 1$ **using** *unitary-col-norm assms*
by (*simp add*: *vec-norm-eq-cpx-vec-length*)

lemma *unitary-col-rvec-norm*:
assumes $U \in carrier\text{-}mat\ n\ n$
and *unitary U*
and $i < n$
shows $rvec\text{-}norm\ (Matrix.col\ U\ i) = 1$ **using** *unitary-col-vec-norm*[*OF assms*]
by (*simp add*: *rvec-norm-def*)

lemma *Cauchy-Schwarz-complex-rvec-norm*:
assumes $dim\text{-}vec\ x = dim\text{-}vec\ y$
shows $cmod\ (inner\text{-}prod\ x\ y) \leq rvec\text{-}norm\ x * rvec\text{-}norm\ y$
proof −
have *x*: $x \in carrier\text{-}vec\ (dim\text{-}vec\ x)$ **by** *simp*
moreover have *y*: $y \in carrier\text{-}vec\ (dim\text{-}vec\ x)$ **using** *assms* **by** *simp*
ultimately have $(cmod\ (inner\text{-}prod\ x\ y))^2 = inner\text{-}prod\ x\ y * inner\text{-}prod\ y\ x$
using *complex-norm-square* **by** (*metis inner-prod-swap mult-conj-cmod-square*)
also have $... \leq inner\text{-}prod\ x\ x * inner\text{-}prod\ y\ y$
using *Cauchy-Schwarz-complex-vec x y* **by** *blast*
finally have $(cmod\ (inner\text{-}prod\ x\ y))^2 \leq inner\text{-}prod\ x\ x * inner\text{-}prod\ y\ y$ **.**
hence $(cmod\ (inner\text{-}prod\ x\ y))^2 \leq Re\ (inner\text{-}prod\ x\ x) * Re\ (inner\text{-}prod\ y\ y)$
using *less-eq-complex-def* **by** *simp*
hence $sqrt\ ((cmod\ (inner\text{-}prod\ x\ y))^2) \leq$
$sqrt\ (Re\ (inner\text{-}prod\ x\ x) * Re\ (inner\text{-}prod\ y\ y))$
using *real-sqrt-le-iff* **by** *blast*

also have ... = *sqrt* (*Re* (*inner-prod x x*)) ∗ *sqrt* (*Re* (*inner-prod y y*))
by (*simp add*: *real-sqrt-mult*)
finally have *sqrt* ((*cmod* (*inner-prod x y*))^2) ≤
sqrt (*Re* (*inner-prod x x*)) ∗ *sqrt* (*Re* (*inner-prod y y*)) **.**
thus *?thesis* **using** *less-eq-complex-def*
by (*metis Re-complex-of-real cmod-power2 cmod-rvec-norm*
inner-prod-rvec-norm-pow2 norm-complex-def)
qed

6.2 The $\mathcal{L}_2$ operator norm is equal to the maximum singular value

definition *max-sgval* **where**
max-sgval A = *sqrt* (*spmax* (*Complex-Matrix.adjoint A* ∗ *A*))

lemma *max-sgval-geq-0*:
assumes *A*∈ *carrier-mat n n*
and *0* < *n*
shows *0* ≤ *max-sgval A*
using *spmax-geq-0*[*of Complex-Matrix.adjoint A* ∗ *A n*]
unfolding *max-sgval-def*
by (*meson adjoint-dim' assms*(*1*) *assms*(*2*) *mult-carrier-mat real-sqrt-ge-zero*)

lemma *max-sgval-uminus*:
shows *max-sgval* (−*A*) = *max-sgval A*
proof −
have *Complex-Matrix.adjoint* (−*A*) = − (*Complex-Matrix.adjoint A*)
using *adjoint-uminus*[*of A*]
by *simp*
hence *Complex-Matrix.adjoint* (−*A*)∗(−*A*) = −(*Complex-Matrix.adjoint A*) ∗
(−*A*)
by *simp*
also have ... = − (*Complex-Matrix.adjoint A* ∗ (−*A*)) **by** *simp*
also have ... = *Complex-Matrix.adjoint A* ∗ *A* **by** *simp*
finally have *Complex-Matrix.adjoint* (−*A*) ∗ (−*A*) =
Complex-Matrix.adjoint A ∗ *A* **.**
thus *?thesis* **unfolding** *max-sgval-def* **by** *simp*
qed

lemma *rvec-leq-sg-spmax*:
assumes *0* < *dim-col A*
and *v*∈ *carrier-vec* (*dim-col A*)
shows *rvec-norm* (*A* ∗$_v$ *v*) ≤ (*max-sgval A*) ∗ *rvec-norm v*
proof −
define *M* **where** *M* = *spmax* (*Complex-Matrix.adjoint A* ∗ *A*)
have *cmod* (*inner-prod* (*A* ∗$_v$ *v*) (*A* ∗$_v$ *v*))
≤ *M* ∗ (*rvec-norm v*)^2 **using** *rvec-norm-mat-mult-le assms*

```
    unfolding M-def by simp
  hence (rvec-norm (A *v v))^2 ≤ M * (rvec-norm v)^2
    using inner-prod-rvec-norm-pow2[of A *v v] assms cmod-rvec-norm
    by (metis inner-prod-rvec-norm-pow2 norm-power of-real-hom.hom-power)
  hence rvec-norm (A *v v) ≤ (sqrt M) * rvec-norm v
    by (rule square-leq, (auto simp add: rvec-norm-geq-0))
  thus ?thesis unfolding max-sgval-def M-def by simp
qed

lemma max-sgval-smult:
  assumes A∈ carrier-mat n n
  and 0 < n
  shows max-sgval (a·mA) = cmod a * max-sgval A
proof −
  have Complex-Matrix.adjoint (a·mA) * (a·mA) =
    conjugate a·m (Complex-Matrix.adjoint A) * (a·mA)
    using adjoint-scale[of a] by simp
  also have ... = conjugate a·m (Complex-Matrix.adjoint A * (a·mA))
    using mult-smult-assoc-mat[of Complex-Matrix.adjoint A] adjoint-dim[of A]
      smult-carrier-mat[of A] assms by (meson mult-smult-assoc-mat)
  also have ... = (conjugate a) ·m (a ·m (Complex-Matrix.adjoint A) * A)
    using mult-smult-distrib[of Complex-Matrix.adjoint A] adjoint-dim[of A]
      smult-carrier-mat[of A] assms by (metis mult-smult-assoc-mat)
  also have ... = (conjugate a * a) ·m (Complex-Matrix.adjoint A) * A
    by (metis adjoint-dim-col carrier-mat-triv index-smult-mat(3)
      mult-smult-assoc-mat smult-smult-times)
  also have ... = (cmod a)^2 ·m(Complex-Matrix.adjoint A * A)
    by (metis adjoint-dim' assms(1) cross3-simps(11) mult-conj-cmod-square
      mult-smult-assoc-mat)
  finally have Complex-Matrix.adjoint (a·mA) * (a·mA) =
    (cmod a)^2 ·m(Complex-Matrix.adjoint A * A) .
  moreover have spmax ((cmod a)^2 ·m(Complex-Matrix.adjoint A * A)) =
    (cmod a)^2 * spmax ((Complex-Matrix.adjoint A * A))
  proof (rule spmax-smult-pos)
    show hermitian (Complex-Matrix.adjoint A * A)
      using assms mult-adjoint-hermitian by auto
    show 0 < dim-col A using assms by simp
  qed (auto simp add: assms)
  ultimately have spmax (Complex-Matrix.adjoint (a·mA) * (a·mA)) =
    (cmod a)^2 * spmax (Complex-Matrix.adjoint A * A) by simp
  thus ?thesis unfolding max-sgval-def
    by (metis abs-norm-cancel real-sqrt-abs real-sqrt-mult)
qed

lemma L2-op-nrm-le-max-sgval:
  assumes 0 < dim-col A
  shows L2-op-nrm A ≤ max-sgval A unfolding L2-op-nrm-def
proof (rule Sup-real-le)
  have vg: ∀ v∈carrier-vec (dim-col A). rvec-norm (A *v v) ≤
```

(*max-sgval A*) ∗ *rvec-norm v*
using *assms rvec-leq-sg-spmax* **by** *simp*
show ∀ *a*∈{*rvec-norm* (*A* ∗$_v$ *v*) |*v*. *dim-vec v* = *dim-col A* ∧ *rvec-norm v* = *1*}.
0 ≤ *a*
using *rvec-norm-geq-0* **by** *auto*
show ∀ *a*∈{*rvec-norm* (*A* ∗$_v$ *v*) |*v*. *dim-vec v* = *dim-col A* ∧ *rvec-norm v* = *1*}.
a ≤*max-sgval A*
using *vg carrier-vec-def* **by** *force*
show {*rvec-norm* (*A* ∗$_v$ *v*) |*v*. *dim-vec v* = *dim-col A* ∧ *rvec-norm v* = *1*} ≠ {}
using *assms rvec-set-ne* **by** *simp*
qed

lemma *max-sgval-eigen*:
assumes *A*∈ *carrier-mat n n*
and *0* < *n*
and *C* = *Complex-Matrix.adjoint A* ∗ *A*
and *v* = *vec-normalize* (*find-eigenvector C* (*spmax-wit C*))
shows *rvec-norm* (*A* ∗$_v$ *v*) = *max-sgval A*
proof −
have *cmod* (*inner-prod* (*A* ∗$_v$ *v*) (*A* ∗$_v$ *v*)) = *cmod* (*inner-prod v* (*C* ∗$_v$ *v*))
using *inner-prod-mult-mat-vec-right nrm-spmax-eigenvector-eq*
by (*metis adjoint-dim′ assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)
carrier-vecI find-eigen-spmax-dim mult-carrier-mat)
also have ... = *spmax C* **using** *assms nrm-spmax-eigenvector-eq*[*of C*]
by (*metis assms*(*3*) *adjoint-dim′ mult-carrier-mat*
nrm-spmax-eigenvector-eq assms(*4*))
finally have *cmod* (*inner-prod* (*A* ∗$_v$ *v*) (*A* ∗$_v$ *v*)) = *spmax C* **.**
hence (*rvec-norm* (*A* ∗$_v$ *v*))^2 = *spmax C*
using *inner-prod-rvec-norm-pow2*[*of A* ∗$_v$ *v*] *assms cmod-rvec-norm*
by (*metis inner-prod-rvec-norm-pow2 norm-power of-real-hom.hom-power*)
thus *rvec-norm* (*A* ∗$_v$ *v*) = *max-sgval A* **using** *assms* **unfolding** *max-sgval-def*
by (*simp add*: *real-sqrt-unique rvec-norm-geq-0*)
qed

lemma *rvec-normalize-leq-L2-op-nrm*:
assumes *rvec-norm v* = *1*
and *dim-col A* = *dim-vec v*
and *0* < *dim-col A*
shows *rvec-norm* (*A* ∗$_v$ *v*) ≤ *L2-op-nrm A*
proof −
have *vg*: ∀ *v*∈*carrier-vec* (*dim-col A*). *rvec-norm* (*A* ∗$_v$ *v*) ≤
(*max-sgval A*) ∗ *rvec-norm v*
using *assms rvec-leq-sg-spmax* **by** *simp*
show *?thesis* **unfolding** *L2-op-nrm-def*
proof (*rule Sup-ge-real*)
show *rvec-norm* (*A* ∗$_v$ *v*) ∈
{*rvec-norm* (*A* ∗$_v$ *v*) |*v*. *dim-vec v* = *dim-col A* ∧ *rvec-norm v* = *1*}
using *assms* **by** *auto*
show ∀ *a*∈{*rvec-norm* (*A* ∗$_v$ *v*) |*v*. *dim-vec v* = *dim-col A* ∧ *rvec-norm v* = *1*}.

```
    a ≤max-sgval A
    using vg carrier-vec-def by force
  show ∀ a∈{rvec-norm (A *v v) |v. dim-vec v = dim-col A ∧ rvec-norm v = 1}.

    0 ≤ a
    using rvec-norm-geq-0 by auto
 qed
qed

lemma max-sgval-le-L2-op-nrm:
 assumes A ∈ carrier-mat n n
 and 0 < n
 shows max-sgval A ≤ L2-op-nrm A
proof −
 define C where C = Complex-Matrix.adjoint A * A
 define v where v = vec-normalize (find-eigenvector C (spmax-wit C))
 have max-sgval A = rvec-norm (A *v v)
  using assms max-sgval-eigen v-def C-def
  by simp
 also have ... ∈ {rvec-norm (A *v v) |v. dim-vec v = dim-col A ∧
  rvec-norm v = 1}
 proof −
  have dim-vec v = dim-col A using assms find-eigen-spmax-dim
   unfolding v-def
   by (metis C-def adjoint-dim' carrier-matD(2) mult-carrier-mat)
  moreover have rvec-norm v = 1
   using normalized-vec-norm find-eigen-spmax-neq-0 assms
   by (metis C-def adjoint-dim' calculation carrier-matD(2)
       carrier-vec-dim-vec mult-carrier-mat normalize-zero
       normalized-rvec-norm v-def)
  ultimately show ?thesis by auto
 qed
 finally have max-sgval A ∈
  {rvec-norm (A *v v) |v. dim-vec v = dim-col A ∧ rvec-norm v = 1} .
 thus max-sgval A ≤ L2-op-nrm A unfolding L2-op-nrm-def
  using assms L2-op-nrm-def rvec-normalize-leq-L2-op-nrm by auto
qed


lemma vec-norm-leq-L2-op-nrm:
 assumes A∈ carrier-mat n n
 and v∈ carrier-vec n
 and 0 < n
 and vec-norm v = 1
shows vec-norm (A *v v) ≤ L2-op-nrm A
proof −
 have rvec-norm v = 1 using assms
  by (simp add: rvec-norm-def)
```

hence *rvec-norm* $(A *_v v) \leq$ *L2-op-nrm* A **unfolding** *L2-op-nrm-def*
by (*metis assms*(1) *assms*(2) *assms*(3) *carrier-dim-vec carrier-matD*(2) *L2-op-nrm-def rvec-normalize-leq-L2-op-nrm*)
have *vec-norm* $(A *_v v) =$ *rvec-norm* $(A *_v v)$
by (*metis Re-complex-of-real cmod-vec-norm rvec-norm-def*)
also have ... $\leq$ *L2-op-nrm* A **using** ‹*rvec-norm* $(A *_v v) \leq$ *L2-op-nrm* A›
by (*simp add*: *less-eq-complex-def*)
finally show *?thesis* **.**
qed

lemma *rvec-norm-leq-L2-op-nrm*:
assumes $A \in$ *carrier-mat n n*
and $v \in$ *carrier-vec n*
and $0 < n$
and *rvec-norm* $v = 1$
shows *rvec-norm* $(A *_v v) \leq$ *L2-op-nrm* A **unfolding** *L2-op-nrm-def*
by (*metis assms carrier-dim-vec carrier-matD*(2) *L2-op-nrm-def rvec-normalize-leq-L2-op-nrm*)

lemma *cmod-trace-rank-le-L2-op-nrm*:
assumes $A \in$ *carrier-mat n n*
and $v \in$ *carrier-vec n*
and $0 < n$
and *rvec-norm* $v = 1$
shows *cmod* (*Complex-Matrix.trace* $(A *$ *rank-1-proj* $v)) \leq$ *L2-op-nrm* A
proof −
have *cmod* (*Complex-Matrix.trace* $(A *$ *rank-1-proj* $v)) =$
cmod (*Complex-Matrix.inner-prod* v $(A *_v v))$
using *rank-1-proj-trace-inner*[*of* A] *assms* **by** *simp*
also have ... $\leq$ *rvec-norm* $v *$ *rvec-norm* $(A *_v v)$
using *Cauchy-Schwarz-complex-rvec-norm assms*(1) *assms*(2) **by** *auto*
also have ... = *rvec-norm* $(A *_v v)$ **using** *assms* **by** *simp*
also have ... $\leq$ *L2-op-nrm* A **using** *rvec-norm-leq-L2-op-nrm assms* **by** *simp*
finally show *?thesis* **.**
qed

lemma *expect-val-L2-op-nrm*:
fixes A::*complex Matrix.mat*
assumes $A \in$ *carrier-mat n n*
and $R \in$ *carrier-mat n n*
and $0 < n$
and *density-operator* R
shows *cmod* (*Complex-Matrix.trace* $(A * R)) \leq$ *L2-op-nrm* A
proof −
have *hermitian* R **using** *assms* **unfolding** *density-operator-def*
by (*simp add*: *positive-is-hermitian*)

```
  from this obtain B U where rd: real-diag-decomp R B U
    using hermitian-real-diag-decomp[of R] assms by auto
  hence unitary U
    using real-diag-decompD(1) unitary-diagD(3) by blast
  have U∈ carrier-mat n n using rd unitary-diag-carrier(2) assms
    by (metis real-diag-decompD(1))
  have cmod (Complex-Matrix.trace (A * R)) ≤
    Re (Complex-Matrix.trace R) * L2-op-nrm A
  proof (rule positive-decomp-cmod-le[of R n])
    show Complex-Matrix.positive R using assms
      unfolding density-operator-def by simp
    show unitary-diag R B U using rd by simp
    show ⋀i. i < n ⟹
      cmod (Complex-Matrix.trace (A * rank-1-proj (Matrix.col U i))) ≤
      L2-op-nrm A
    proof−
      fix i
      assume i < n
      show cmod (Complex-Matrix.trace (A * rank-1-proj (Matrix.col U i))) ≤
        L2-op-nrm A
      proof (rule cmod-trace-rank-le-L2-op-nrm[of A n])
        show Matrix.col U i ∈ carrier-vec n
          by (metis ‹unitary-diag R B U› assms(2) carrier-matD(1)
              col-dim unitary-diag-carrier(2))
        show rvec-norm (Matrix.col U i) = 1
          using unitary-col-rvec-norm ‹unitary U› ‹i < n› ‹U∈ carrier-mat n n›
          by simp
      qed (auto simp add: assms)
    qed
  qed (auto simp add: assms)
  also have ... = L2-op-nrm A using assms
    unfolding density-operator-def by simp
  finally show ?thesis .
qed
```

6.3 Consequences for the $\mathcal{L}_2$ operator norm

```
lemma L2-op-nrm-geq-0:
assumes A ∈ carrier-mat n n
  and 0 < n
shows 0 ≤ L2-op-nrm A
  using assms max-sgval-le-L2-op-nrm[of A n] max-sgval-geq-0[of A n]
  by simp

lemma L2-op-nrm-max-sgval-eq:
  assumes A ∈ carrier-mat n n
  and 0 < n
  shows L2-op-nrm A = max-sgval A
proof −
```

```
  have L2-op-nrm A ≤ max-sgval A using assms L2-op-nrm-le-max-sgval by simp
  moreover have max-sgval A ≤ L2-op-nrm A
    using assms max-sgval-le-L2-op-nrm by simp
  ultimately show ?thesis by simp
qed

lemma rvec-leq-L2-op-nrm:
  assumes A ∈ carrier-mat n n
  and 0 < n
  and v∈ carrier-vec n
shows rvec-norm (A *v v) ≤ (L2-op-nrm A) * rvec-norm v
  using assms L2-op-nrm-max-sgval-eq rvec-leq-sg-spmax by simp

lemma L2-op-nrm-mult-le:
  assumes A ∈ carrier-mat n n
  and B∈ carrier-mat n n
  and 0 < n
shows L2-op-nrm (A*B) ≤ L2-op-nrm A * L2-op-nrm B
proof −
  have Sup {rvec-norm (A * B *v v) |v. dim-vec v = n ∧ rvec-norm v = 1} ≤
    L2-op-nrm A * L2-op-nrm B
  proof (rule Sup-real-le)
    show {rvec-norm (A * B *v v) |v. dim-vec v = n ∧ rvec-norm v = 1} ≠ {}
      using rvec-set-ne assms by auto
    show ∀ a∈{rvec-norm (A * B *v v) |v. dim-vec v = n ∧ rvec-norm v = 1}.
      0 ≤ a
      using rvec-norm-geq-0 by auto
    show ∀ a∈{rvec-norm (A * B *v v) |v. dim-vec v = n ∧ rvec-norm v = 1}.
      a ≤ L2-op-nrm A * L2-op-nrm B
    proof
      fix x
      assume x ∈ {rvec-norm (A * B *v v)|v. dim-vec v = n ∧ rvec-norm v = 1}
      hence ∃ v.(dim-vec v = n ∧rvec-norm v = 1 ∧ x = rvec-norm (A * B *v v))
        by auto
      from this obtain v where dim-vec v = n and rvec-norm v = 1
        and x = rvec-norm (A * B *v v) by auto note vprop = this
      have A * B *v v = A *v (B *v v) using assms vprop by auto
      hence x = rvec-norm (A *v (B *v v)) using vprop by simp
      also have ... ≤ L2-op-nrm A * rvec-norm (B *v v)
        using assms rvec-leq-L2-op-nrm[of A n B *v v] carrier-vecI
          mult-mat-vec-carrier vprop(1)
        by blast
      also have ... ≤ L2-op-nrm A * (L2-op-nrm B)
        using vprop rvec-normalize-leq-L2-op-nrm[of v] assms
       by (metis carrier-matD(2) more-arith-simps(6) mult-mono' mult-zero-right
           rvec-norm-geq-0 verit-comp-simplify1(2))
      finally show x ≤ L2-op-nrm A * L2-op-nrm B .
    qed
  qed
```

thus *?thesis* **using** *assms L2-op-nrm-def* **by** *simp*
qed

lemma *L2-op-nrm-smult*:
assumes $A\in$ *carrier-mat n n*
and $0 < n$
shows *L2-op-nrm* $(c \cdot_m A)$ = *cmod c* $*$ *L2-op-nrm A*
by (*metis L2-op-nrm-max-sgval-eq assms*(*1*) *assms*(*2*) *max-sgval-smult smult-carrier-mat*)

lemma *L2-op-nrm-uminus*:
assumes $A\in$ *carrier-mat n n*
and $0 < n$
shows *L2-op-nrm* $(-A)$ = *L2-op-nrm A*
using *L2-op-nrm-max-sgval-eq max-sgval-uminus*[*of A*] *assms* **by** *simp*

lemma *L2-op-nrm-triangle*:
assumes $A \in$ *carrier-mat n n*
and $B\in$ *carrier-mat n n*
and $0 < n$
shows *L2-op-nrm* $(A+B) \leq$ *L2-op-nrm A* + *L2-op-nrm B*
proof −
define *C* **where** *C* = *Complex-Matrix.adjoint* $(A+B) * (A+B)$
have *C* $\in$ *carrier-mat n n* **using** *C-def*
by (*metis add-carrier-mat adjoint-dim′ assms*(*2*) *mult-carrier-mat*)
define *v* **where** *v* = *vec-normalize* (*find-eigenvector C* (*spmax-wit C*))
have *v*$\in$ *carrier-vec n* **using** *v-def*
by (*metis* ‹*C* $\in$ *carrier-mat n n*› *assms*(*3*) *carrier-dim-vec find-eigen-spmax-dim*)
have *rvec-norm* $(A *_v v) \leq$ *L2-op-nrm A*
proof (*rule rvec-normalize-leq-L2-op-nrm*)
show *0* < *dim-col A* **using** *assms* **by** *simp*
show *dim-col A* = *dim-vec v* **using** ‹*v*$\in$ *carrier-vec n*› *assms* **by** *simp*
show *rvec-norm v* = *1* **using** *v-def*
by (*metis* ‹*C* $\in$ *carrier-mat n n*› ‹*v* $\in$ *carrier-vec n*› *assms*(*3*) *carrier-vecD find-eigen-spmax-neq-0 normalize-zero normalized-rvec-norm zero-carrier-vec*)
qed
have *rvec-norm* $(B *_v v) \leq$ *L2-op-nrm B*
proof (*rule rvec-normalize-leq-L2-op-nrm*)
show *0* < *dim-col B* **using** *assms* **by** *simp*
show *dim-col B* = *dim-vec v* **using** ‹*v*$\in$ *carrier-vec n*› *assms* **by** *simp*
show *rvec-norm v* = *1* **using** *v-def*
by (*metis* ‹*C* $\in$ *carrier-mat n n*› ‹*v* $\in$ *carrier-vec n*› *assms*(*3*) *carrier-vecD find-eigen-spmax-neq-0 normalize-zero normalized-rvec-norm zero-carrier-vec*)
qed
have $(A+B)*_v v = A *_v v + B *_v v$
proof (*rule add-mult-distrib-mat-vec*)

show $A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n* **using** *assms* **by** *auto*
show $v \in$ *carrier-vec n* **using** ‹$v \in$ *carrier-vec n*› **.**
qed
hence *L2-op-nrm* $(A+B)$ = *rvec-norm* $(A *_v v + B *_v v)$
using *L2-op-nrm-max-sgval-eq*[*of A+B n*] *max-sgval-eigen*[*of A+B n*]
C-def v-def **by** (*simp add*: *assms*(2) *assms*(3))
also have ... $\leq$ *rvec-norm* $(A *_v v)$ + *rvec-norm* $(B *_v v)$
using *rvec-norm-triangle assms* ‹$v \in$ *carrier-vec n*› **by** *simp*
also have ... $\leq$ *L2-op-nrm A* + *L2-op-nrm B*
using ‹*rvec-norm* $(B *_v v) \leq$ *L2-op-nrm B*› ‹*rvec-norm* $(A *_v v) \leq$ *L2-op-nrm A*›
by *simp*
finally show *?thesis* **.**
qed

lemma *L2-op-nrm-triangle′*:
assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n n*
and $0 < n$
shows *L2-op-nrm* $(A-B) \leq$ *L2-op-nrm A* + *L2-op-nrm B*
proof −
have *L2-op-nrm* $(A-B)$ = *L2-op-nrm* $(A + (-B))$
using *assms add-uminus-minus-mat*[*of A*] **by** *simp*
also have ... $\leq$ *L2-op-nrm A* + *L2-op-nrm* $(-B)$
using *L2-op-nrm-triangle assms* **by** *simp*
also have ... = *L2-op-nrm A* + *L2-op-nrm B*
using *L2-op-nrm-uminus assms* **by** *simp*
finally show *?thesis* **.**
qed

lemma *hermitian-max-sgval-eq*:
fixes *A*::*complex Matrix.mat*
assumes *hermitian A*
and $0 <$ *dim-row A*
shows *max-sgval A* = *spmax A*
proof −
define *n* **where** $n =$ *dim-row A*
have $A \in$ *carrier-mat n n*
using *assms n-def hermitian-square* **by** (*simp add*: *hermitian-square*)
have *max-sgval A* = *sqrt* (*spmax* $(A * A)$)
using *assms* **unfolding** *max-sgval-def hermitian-def* **by** *simp*
also have ... = *spmax A*
using *assms hermitian-square-spmax spmax-geq-0* ‹$A \in$ *carrier-mat n n*›
by *simp*
finally show *?thesis* **.**
qed

lemma *hermitian-L2-op-nrm-spmax-eq*:
fixes *A*::*complex Matrix.mat*

assumes *hermitian A*
and *0 < dim-row A*
shows *L2-op-nrm A = spmax A*
proof −
define *n* **where** *n = dim-row A*
have *A ∈ carrier-mat n n*
using *n-def* **by** (*metis assms*(*1*) *hermitian-square*)
thus *?thesis*
using *assms hermitian-max-sgval-eq*[*of A*] *L2-op-nrm-max-sgval-eq*[*of A n*] *n-def*
by *metis*
qed

lemma *hermitian-L2-op-nrm-sqrt*:
fixes *A*::*complex Matrix.mat*
assumes *hermitian A*
and *0 < dim-row A*
shows *L2-op-nrm A = sqrt* (*L2-op-nrm* (*A∗A*))
by (*metis assms hermitian-L2-op-nrm-spmax-eq hermitian-commute hermitian-def*

hermitian-max-sgval-eq index-mult-mat(*2*) *max-sgval-def*)

lemma *idty-L2-op-nrm*:
assumes *0 < n*
shows *L2-op-nrm* (*1*$_m$ *n*) = *1*
using *assms idty-spmax*[*of n*] *hermitian-L2-op-nrm-spmax-eq*
by (*simp add*: *hermitian-one*)

lemma *commutator-L2-op-nrm-le*:
assumes *A∈ carrier-mat n n*
and *B∈ carrier-mat n n*
and *0 < n*
shows *L2-op-nrm* (*commutator A B*) ≤ *2 ∗ L2-op-nrm A ∗ L2-op-nrm B*
proof −
have *L2-op-nrm* (*commutator A B*) ≤ *L2-op-nrm* (*A∗B*) + *L2-op-nrm* (*B ∗ A*)
unfolding *commutator-def*
using *L2-op-nrm-triangle*′[*of A∗B n*] *assms* **by** *simp*
also have ... ≤ *L2-op-nrm A ∗ L2-op-nrm B* + *L2-op-nrm* (*B∗A*)
using *L2-op-nrm-mult-le assms* **by** *simp*
also have ... ≤ *L2-op-nrm A ∗ L2-op-nrm B* + *L2-op-nrm B ∗ L2-op-nrm A*
using *L2-op-nrm-mult-le*[*of B n A*] *assms*
by *linarith*
also have ... = *L2-op-nrm A ∗ L2-op-nrm B* + *L2-op-nrm A ∗ L2-op-nrm B* **by** *simp*
also have ... = *2 ∗ L2-op-nrm A ∗ L2-op-nrm B* **by** *simp*
finally show *?thesis* **.**
qed

lemma *herm-sq-id-L2-op-nrm*:
assumes *0 < n*

and $A \in$ *carrier-mat n n*
and *hermitian A*
and $A * A = 1_m\ n$
shows *L2-op-nrm A = 1*
proof −
have *spmax A = 1* **using** *assms hermitian-square-idty-spmax* **by** *simp*
thus *?thesis* **using** *hermitian-L2-op-nrm-spmax-eq assms* **by** *simp*
qed

lemma *comm-L2-op-nrm-le*:
assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n n*
and $0 < n$
and $A * A = 1_m\ n$
and $B * B = 1_m\ n$
and *hermitian A*
and *hermitian B*
shows *L2-op-nrm* (*commutator A B*) ≤ 2
proof −
have *L2-op-nrm* (*commutator A B*) $\leq 2 *$ *L2-op-nrm A* $*$ *L2-op-nrm B*
using *assms commutator-L2-op-nrm-le* **by** *simp*
also have ... $= 2 *$ *L2-op-nrm A*
using *herm-sq-id-L2-op-nrm assms* **by** *simp*
also have ... $= 2$
using *herm-sq-id-L2-op-nrm assms* **by** *simp*
finally show *?thesis* **.**
qed

lemma *idty-smult-nat-L2-op-nrm*:
assumes $0 < n$
shows *L2-op-nrm* $((m::nat) \cdot_m (1_m\ n)) = m$
proof −
have *L2-op-nrm* $((m::nat) \cdot_m (1_m\ n))$ = *spmax* $((m::nat) \cdot_m (1_m\ n))$
using *hermitian-L2-op-nrm-spmax-eq*[*of m* $\cdot_m$ 1_m *n*] *hermitian-one*
assms hermitian-smult
by (*metis index-one-mat*(2) *index-smult-mat*(2) *of-real-of-nat-eq*
one-carrier-mat)
also have ... $= m *$ *L2-op-nrm* $(1_m\ n)$
using *spmax-smult-pos* [*of* 1_m *n n m*] *assms hermitian-one*
hermitian-L2-op-nrm-spmax-eq[*of* 1_m *n*]
by (*simp add*: ‹$\bigwedge$*n. hermitian* $(1_m\ n)$›)
also have ... $= m$ **using** *idty-L2-op-nrm*[*of n*] *assms* **by** *simp*
finally show *?thesis* **.**
qed
end

theory *Density-Matrix-Basics*
imports

Matrix-L2-Operator-Norm
begin

7 On density matrices

7.1 Density matrix characterization

Density matrices are defined as positive operators with trace 1, we prove in this section that they are exactly the convex combinations of pure states.

lemma (**in** *cpx-sq-mat*) *mixed-state-density-operator*:
assumes $\bigwedge i.\ i \in \{..< (n::nat)\} \Longrightarrow 0 \leq p\ i$
and *sum p* $\{..< n\} = 1$
and $\bigwedge i.\ i \in \{..< n\} \Longrightarrow$ *dim-vec* $(v\ i) =$ *dimR*
and $\bigwedge i.\ i \in \{..< n\} \Longrightarrow \|v\ i\| = 1$
shows *density-operator* (*sum-mat* $(\lambda\ i.\ (p\ i)\ \cdot_m$ (*rank-1-proj* $(v\ i))) \{..< n\}$)
unfolding *density-operator-def*
proof
have *car*: $\bigwedge i.\ i \in \{..< n\} \Longrightarrow$ *rank-1-proj* $(v\ i) \in$ *fc-mats*
using *assms rank-1-proj-carrier fc-mats-carrier dim-eq*
by *metis*
show *Complex-Matrix.positive* (*sum-mat* $(\lambda i.\ p\ i \cdot_m$ *rank-1-proj* $(v\ i))$
$\{..< n\}$)
proof (*rule sum-mat-positive*)
show *finite* $\{..< n\}$ **by** *simp*
show $\bigwedge i.\ i \in \{..< n\} \Longrightarrow p\ i \cdot_m$ *rank-1-proj* $(v\ i) \in$ *fc-mats* **using** *car*
by (*simp add*: *cpx-sq-mat-smult*)
show $\bigwedge i.\ i \in \{..< n\} \Longrightarrow$ *Complex-Matrix.positive* $(p\ i \cdot_m$ *rank-1-proj* $(v\ i))$
proof −
fix *i*
assume $i \in \{..< n\}$
show *Complex-Matrix.positive* $(p\ i \cdot_m$ *rank-1-proj* $(v\ i))$
proof (*rule positive-smult*)
show *Complex-Matrix.positive* (*rank-1-proj* $(v\ i))$ **using** $\langle i \in \{..< n\}\rangle$
by (*simp add*: *assms rank-1-proj-positive*)
show $0 \leq p\ i$ **using** *assms* $\langle i \in \{..< n\}\rangle$ **by** *simp*
show *rank-1-proj* $(v\ i) \in$ *carrier-mat dimR dimR*
using $\langle i \in \{..< n\}\rangle$ *car fc-mats-carrier dim-eq* **by** *simp*
qed
qed
qed
have *Complex-Matrix.trace* (*sum-mat* $(\lambda i.\ p\ i \cdot_m$ *rank-1-proj* $(v\ i))$ $\{..< n\}$)=
sum $(\lambda i.$ *Complex-Matrix.trace* $(p\ i \cdot_m$ *rank-1-proj* $(v\ i)))$ $\{..< n\}$
proof (*rule trace-sum-mat*)
show *finite* $\{..< n\}$ **by** *simp*
show $\bigwedge i.\ i \in \{..< n\} \Longrightarrow p\ i \cdot_m$ *rank-1-proj* $(v\ i) \in$ *fc-mats* **using** *car*
by (*simp add*: *cpx-sq-mat-smult*)
qed
also have ... = *sum* $(\lambda i.\ p\ i *$ *Complex-Matrix.trace* (*rank-1-proj* $(v\ i)))$

```
    {..< n}
  proof (rule sum.cong)
    fix i
    assume i∈{..< n}
    show Complex-Matrix.trace (p i ·_m rank-1-proj (v i)) =
      p i * Complex-Matrix.trace (rank-1-proj (v i))
    proof (rule trace-smult)
      show rank-1-proj (v i) ∈ carrier-mat dimR dimR
        using ‹i∈ {..< n}› car fc-mats-carrier dim-eq by simp
    qed
  qed simp
  also have ... = sum (λi. p i) {..< n}
  proof (rule sum.cong)
    fix i
    assume i∈{..< n}
    thus p i * Complex-Matrix.trace (rank-1-proj (v i)) = p i
      using assms rank-1-proj-trace by simp
  qed simp
  also have ... = 1 using assms by simp
  finally show Complex-Matrix.trace
    (sum-mat (λi. p i ·_m rank-1-proj (v i)) {..< n}) = 1 .
qed

lemma (in cpx-sq-mat) density-operator-mixed-state:
  assumes R∈ fc-mats
  and density-operator R
shows ∃ p v (n::nat). (∀ i∈{..< n}. 0 ≤ p i) ∧
  (∀ i ∈ {..< n}. dim-vec (v i) = dimR) ∧
  (∀ i ∈ {..< n}. ‖v i‖ = 1) ∧ (sum p {..< n} = 1) ∧
  (R = sum-mat (λ i. (p i) ·_m (rank-1-proj (v i))) {..< n})
proof −
  have R∈ carrier-mat dimR dimR using assms fc-mats-carrier dim-eq by simp
  have 0 < dimR using npos .
  moreover have hermitian R using assms positive-is-hermitian
    unfolding density-operator-def by simp
  moreover have R∈ carrier-mat dimR dimR using assms fc-mats-carrier dim-eq

    by simp
  ultimately obtain B U where rdd: real-diag-decomp R B U
    using hermitian-real-diag-decomp by blast
  hence unitary-diag R B U by simp
  hence dim-row B = dimR
    using assms dim-eq fc-mats-carrier unitary-diag-carrier(1) by blast
  define p where p = (λi. diag-mat B!i)
  define v where v = (λi. Matrix.col U i)
  have ∀ i∈{..< dimR}. 0 ≤ p i
  proof
    fix i
    assume i ∈ {..< dimR}
```

```
  have 0 ≤ B$$(i,i)
  proof (rule positive-unitary-diag-pos)
    show R∈ carrier-mat dimR dimR using ‹R ∈ carrier-mat dimR dimR› .
    show Complex-Matrix.positive R
      using  assms unfolding density-operator-def by simp
    show unitary-diag R B U using rdd by simp
    show i < dimR using ‹i∈ {..< dimR}› by simp
  qed
  also have ... = p i
    using ‹dim-row B = dimR› ‹i∈ {..< dimR}›
    unfolding p-def diag-mat-def by simp
  finally show 0 ≤ p i .
qed
moreover have ∀ i ∈ {..< dimR}. dim-vec (v i) = dimR
  using ‹unitary-diag R B U› assms(1) dim-col dim-eq fc-mats-carrier
    unitary-diag-carrier(2) v-def by blast
moreover have ∀ i ∈ {..< dimR}. ‖v i‖ = 1
proof
  fix i
  assume i ∈ {..< dimR}
  show ‖v i‖ = 1 unfolding v-def
  proof (rule unitary-col-norm)
    show i < dimR using ‹i∈ {..< dimR}› by simp
    show Complex-Matrix.unitary U
      using rdd ‹unitary-diag R B U› unitary-diagD(3) by blast
    show U ∈ carrier-mat dimR dimR
      using ‹R ∈ carrier-mat dimR dimR› ‹unitary-diag R B U›
        unitary-diag-carrier(2) by auto
  qed
qed
moreover have sum p {..< dimR} = 1 using unitarily-equiv-trace'
proof−
  have sum p {..< dimR} = (∑ i = 0..<dim-row R. B $$ (i, i))
  proof (rule sum.cong)
    show {..<dimR} = {0..<dim-row R}
      using ‹R ∈ carrier-mat dimR dimR› by auto
    show ⋀x. x ∈ {0..<dim-row R} ⟹ p x = B $$ (x, x)
      using ‹dim-row B = dimR› ‹R ∈ carrier-mat dimR dimR›
      unfolding p-def diag-mat-def by auto
  qed
  also have ... = Complex-Matrix.trace R
    using unitarily-equiv-trace' ‹R ∈ carrier-mat dimR dimR›
    by (metis ‹unitary-diag R B U› unitary-diag-imp-unitarily-equiv)
  also have ... = 1 using assms unfolding density-operator-def by simp
  finally show ?thesis .
qed
moreover have R = sum-mat (λ i. (p i)  ·m (rank-1-proj (v i))) {..< dimR}
  unfolding p-def v-def
proof  (rule sum-decomp-cols[symmetric])
```

show *R∈ fc-mats* **using** *assms* **by** *simp*
show *unitary-diag R B U* **using** ‹*unitary-diag R B U*› **.**
show *hermitian R* **using** *assms positive-is-hermitian*
unfolding *density-operator-def* **by** *simp*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma (**in** *cpx-sq-mat*) *density-operator-iff-mixed-state*:
assumes *R∈ fc-mats*
shows *density-operator R ⟷*
(∃ *p v* (*n*::*nat*). (∀ *i*∈{..< *n*}. *0* ≤ *p i*) ∧
(∀ *i* ∈ {..< *n*}. *dim-vec* (*v i*) = *dimR*) ∧
(∀ *i* ∈ {..< *n*}. ‖*v i*‖ = *1*) ∧ (*sum p* {..< *n*} = *1*) ∧
(*R* = *sum-mat* (λ *i*. (*p i*) ·$_m$ (*rank-1-proj* (*v i*))) {..< *n*})) (**is** *?L ⟷ ?R*)
proof
show *?L ⟹ ?R* **using** *density-operator-mixed-state*[*OF assms*] **by** *simp*
next
show *?R ⟹ ?L*
proof −
assume ∃ *p v* (*n*::*nat*). (∀ *i*∈{..<*n*}. *0* ≤ *p i*) ∧
(∀ *i*∈{..<*n*}. *dim-vec* (*v i*) = *dimR*) ∧ (∀ *i*∈{..<*n*}. ‖*v i*‖ = *1*) ∧
sum p {..<*n*} = *1* ∧ *R* = *sum-mat* (λ*i*. *p i* ·$_m$ *rank-1-proj* (*v i*)) {..<*n*}
from *this* **obtain** *n p v* **where** ⋀*i*. *i*∈{..<(*n*::*nat*)} ⟹ *0* ≤ *p i* **and**
∀ *i*∈{..<*n*}. *dim-vec* (*v i*) = *dimR* **and** ∀ *i*∈{..<*n*}. ‖*v i*‖ = *1* **and**
sum p {..<*n*} = *1* **and**
R = *sum-mat* (λ*i*. *p i* ·$_m$ *rank-1-proj* (*v i*)) {..<*n*} **by** *auto* **note** *npv* = *this*
thus *density-operator R* **using** *npv mixed-state-density-operator* **by** *auto*
qed
qed

7.2 Separable density matrices

We define the notion of a separable density matrix: this is a matrix of the form $\sum_{i=1}^{n} p_i \rho_A^i \otimes \rho_B^i$, where the p_is are positive and sum up to 1.

definition *separately-decomposes* **where**
separately-decomposes R (*n*::*nat*) *nA nB K F S* ≡
(∀ *a*< *n*. (*0*::*complex*) ≤ (*complex-of-real* (*K a*)) ∧
F a∈ *carrier-mat nA nA* ∧ *S a* ∈ *carrier-mat nB nB* ∧
density-operator (*F a*) ∧ *density-operator* (*S a*)) ∧ *0* < *nA* ∗ *nB* ∧
sum K {..< *n*} = *1* ∧ *R* = *fixed-carrier-mat.sum-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*)
(λ*a*. *K a* ·$_m$ ((*F a*) ⨂ (*S a*))) {..< *n*}

definition *separable-density* **where**
separable-density nA nB R ≡
∃ (*n*::*nat*) *K F S*. *separately-decomposes R n nA nB K F S*

lemma *separately-decomposes-carrier*:
assumes *separately-decomposes R* (*n*::*nat*) *nA nB K F S*

and $0 < nA$
and $0 < nB$
shows $R \in$ *carrier-mat* $(nA*nB)$ $(nA*nB)$
proof −
define *fc*::*complex Matrix.mat set*
where $fc =$ *carrier-mat* $(nA * nB)$ $(nA * nB)$
interpret *cpx-sq-mat* $nA * nB$ $nA * nB$ *fc*
proof
show $fc =$ *carrier-mat* $(nA * nB)$ $(nA * nB)$ **using** *fc-def* **by** *simp*
show $0 < nA * nB$ **using** *assms* **unfolding** *separately-decomposes-def*
by *simp*
qed *simp*
have *car*: $\bigwedge a.\ a \in \{..<n\} \Longrightarrow F\ a \bigotimes S\ a \in fc$
proof −
fix *a*
assume $a \in \{..< n\}$
hence $F\ a \in$ *carrier-mat* nA nA $S\ a \in$ *carrier-mat* nB nB
using *assms* **unfolding** *separately-decomposes-def* **by** *auto*
thus $F\ a \bigotimes S\ a \in fc$ **using** *tensor-mat-carrier* **unfolding** *fc-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
qed
have $R =$ *sum-mat* $(\lambda a.\ K\ a \cdot_m ((F\ a) \bigotimes (S\ a)))\ \{..< n\}$
using *assms* **unfolding** *separately-decomposes-def* **by** *simp*
also have ...$\in$ *carrier-mat* $(nA*nB)$ $(nA*nB)$
proof (*rule sum-mat-carrier*)
show $\bigwedge i.\ i \in \{..<n\} \Longrightarrow K\ i \cdot_m (F\ i \bigotimes S\ i) \in fc$ **using** *car*
by (*simp add*: *smult-mem*)
qed
finally show *?thesis* **.**
qed

lemma *separately-decomposes-carrier-pos*:
assumes *separately-decomposes R n nA nB K F S*
shows $0 < nA$ $0 < nB$
using *assms* **unfolding** *separately-decomposes-def* **by** *auto*

lemma *separable-density-carrier*:
assumes *separable-density nA nB R*
and $0 < nA$
and $0 < nB$
shows $R \in$ *carrier-mat* $(nA*nB)$ $(nA*nB)$
proof −
have $\exists n\ K\ F\ S.$ *separately-decomposes R n nA nB K F S*
using *assms* **unfolding** *separable-density-def* **by** *simp*
from *this* **obtain** *n K F S* **where**
separately-decomposes R n nA nB K F S **by** *auto*
note *props* = *this*
thus *?thesis* **using** *separately-decomposes-carrier assms* **by** *simp*
qed

lemma *separately-decomposes-trace*:
assumes *separately-decomposes R n nA nB K F S*
shows *Complex-Matrix.trace R = 1*
proof −
define *fc*::*complex Matrix.mat set*
where *fc = carrier-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*)
interpret *cpx-sq-mat nA* ∗ *nB nA* ∗ *nB fc*
proof
show *fc = carrier-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*) **using** *fc-def* **by** *simp*
show *0 < nA* ∗ *nB* **using** *assms* **unfolding** *separately-decomposes-def*
by *simp*
qed *simp*
have *car*: ⋀*a*. *a* ∈ {..<*n*} ⟹ *F a* ⨂ *S a* ∈ *fc*
proof −
fix *a*
assume *a*∈ {..< *n*}
hence *F a* ∈ *carrier-mat nA nA S a*∈ *carrier-mat nB nB*
using *assms* **unfolding** *separately-decomposes-def* **by** *auto*
thus *F a* ⨂ *S a* ∈ *fc* **using** *tensor-mat-carrier* **unfolding** *fc-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
qed
have *adev*: ∀ *a* < *n*. *Complex-Matrix.trace* (*K a* ·$_m$ ((*F a*) ⨂ (*S a*))) =
K a ∗ (*Complex-Matrix.trace* (*F a*) ∗ *Complex-Matrix.trace* (*S a*))
proof (*intro allI impI*)
fix *a*
assume *a* < *n*
have *Complex-Matrix.trace* (*K a* ·$_m$ ((*F a*) ⨂ (*S a*))) =
K a ∗ *Complex-Matrix.trace* ((*F a*) ⨂ (*S a*))
proof (*rule trace-smult*)
show *F a* ⨂ *S a* ∈ *carrier-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*) **using** *car* ‹*a* < *n*›
by (*simp add*: *fc-def*)
qed
also have ... = *K a* ∗ (*Complex-Matrix.trace* (*F a*) ∗
Complex-Matrix.trace (*S a*))
proof −
have *Complex-Matrix.trace* ((*F a*) ⨂ (*S a*)) =
Complex-Matrix.trace (*F a*) ∗ *Complex-Matrix.trace* (*S a*)
using *tensor-mat-trace assms* **unfolding** *separately-decomposes-def*
by (*meson* ‹*a* < *n*› *nat-0-less-mult-iff*)
thus *?thesis* **by** *simp*
qed
finally show *Complex-Matrix.trace* (*K a* ·$_m$ ((*F a*) ⨂ (*S a*))) =
K a ∗ (*Complex-Matrix.trace* (*F a*) ∗ *Complex-Matrix.trace* (*S a*)) .
qed
have *Complex-Matrix.trace R* =
Complex-Matrix.trace (*sum-mat* (λ*a*. *K a* ·$_m$ ((*F a*) ⨂ (*S a*))) {..< *n*})
using *assms* **unfolding** *separately-decomposes-def* **by** *simp*
also have ... =

```
    sum (λa. Complex-Matrix.trace (K a ·m ((F a) ⊗ (S a)))) {..< n}
  proof (rule trace-sum-mat)
    show ⋀a. a ∈ {..<n} ⟹ K a ·m (F a ⊗ S a) ∈ fc
      using car cpx-sq-mat-smult by auto
  qed simp
  also have ... =
    sum (λa. K a ∗ (Complex-Matrix.trace (F a)∗ Complex-Matrix.trace (S a)))
      {..< n} using adev by simp
  also have ... = sum (λa. K a) {..< n}
  proof −
    have ∀ a < n. Complex-Matrix.trace (F a)∗ Complex-Matrix.trace (S a) = 1
    proof (intro allI impI)
      fix a
      assume a < n
      thus Complex-Matrix.trace (F a) ∗ Complex-Matrix.trace (S a) = 1
        using assms unfolding separately-decomposes-def
        by (metis density-operator-def lambda-one)
    qed
    thus ?thesis by simp
  qed
  also have ... = 1 using assms unfolding separately-decomposes-def
    by simp
  finally show ?thesis .
qed

lemma separately-decomposes-positive:
  assumes separately-decomposes R  n nA nB K F S
  and 0 < nA
  and 0 < nB
  shows Complex-Matrix.positive R
proof −
  define fc::complex Matrix.mat set
    where fc = carrier-mat (nA ∗ nB) (nA ∗ nB)
  interpret cpx-sq-mat nA ∗ nB nA ∗ nB fc
  proof
    show fc = carrier-mat (nA ∗ nB) (nA ∗ nB) using fc-def by simp
    show 0 < nA ∗ nB using assms unfolding separately-decomposes-def
      by simp
  qed simp
  have ac: ∀ a∈{..<n}.(F a ⊗ S a) ∈ fc
  proof
    fix a
    assume a∈ {..< n}
    hence F a ∈ carrier-mat nA nA S a∈ carrier-mat nB nB
      using assms unfolding separately-decomposes-def by auto
    thus F a ⊗ S a ∈ fc using tensor-mat-carrier unfolding fc-def
      by (metis carrier-matD(1) carrier-matD(2))
  qed
  have Complex-Matrix.positive (sum-mat (λa. K a·m(F a ⊗ (S a))) {..< n})
```

```
  proof (rule sum-mat-positive)
    show ⋀a. a∈{..<n} ⟹ K a ·m (F a ⨂ S a) ∈ fc
      using ac by (simp add: cpx-sq-mat-smult)
    show ⋀i. i∈{..<n} ⟹ Complex-Matrix.positive (K i ·m (F i ⨂ S i))
    proof −
      fix i
      assume i ∈ {..< n}
      show Complex-Matrix.positive (K i ·m (F i ⨂ S i))
      proof (rule positive-smult)
        show F i ⨂ S i ∈ carrier-mat (nA∗nB) (nA∗nB)
          using ‹i ∈ {..< n}› ac fc-def by simp
        show 0 ≤ complex-of-real (K i) using ‹i ∈ {..< n}› assms
          unfolding separately-decomposes-def by simp
        show Complex-Matrix.positive (F i ⨂ S i)
        proof (rule tensor-mat-positive)
          show 0 < nA using assms by simp
          show 0 < nB using assms by simp
          show F i ∈ carrier-mat nA nA using ‹i ∈ {..< n}› assms
            unfolding separately-decomposes-def by simp
          show S i ∈ carrier-mat nB nB using ‹i ∈ {..< n}› assms
            unfolding separately-decomposes-def by simp
          show Complex-Matrix.positive (F i) using ‹i ∈ {..< n}› assms
            unfolding separately-decomposes-def density-operator-def by simp
          show Complex-Matrix.positive (S i) using ‹i ∈ {..< n}› assms
            unfolding separately-decomposes-def density-operator-def by simp
        qed
      qed
    qed
  qed simp
  thus ?thesis using assms unfolding separately-decomposes-def by simp
qed
```

A separable density matrix is indeed a density matrix:

```
lemma separable-density-operator:
  assumes separable-density nA nB R
  and 0 < nA
  and 0 < nB
  shows density-operator R unfolding density-operator-def
proof
  have ∃n K F S. separately-decomposes R n nA nB K F S
    using assms unfolding separable-density-def by simp
  from this obtain n K F S where
    separately-decomposes R n nA nB K F S by auto
  note props = this
  show Complex-Matrix.positive R
    using assms props separately-decomposes-positive
    by metis
  show Complex-Matrix.trace R = 1 using props separately-decomposes-trace
    by metis
```

qed

7.3 Characterization of pure states

A density matrix represents a pure state if it is the rank 1 projection of a single vector. These can be characterized either as the density matrices with a square of trace 1, or as the density matrices that are projectors.

definition *pure-density-operator* **where**
pure-density-operator R ≡ (∃ *v*. *R* = *rank-1-proj v*)

lemma *density-pure-single-diag*:
assumes *A* ∈ *carrier-mat n n*
and *Complex-Matrix.trace A* = (*1*::*real*)
and *Complex-Matrix.trace* (*A*∗*A*) = (*1*::*real*)
and *unitary-diag A B U*
and *I* = {*0* ..< *n*}
and ∀ *i*∈*I*. *A* $$ (*i*,*i*) ≥ *0*
and ∀ *i*∈*I*. *B* $$ (*i*,*i*) ≥ *0*
shows ∃ *j*∈*I*. *B* $$ (*j*,*j*) = *1* ∧ (∀ *i* ∈ *I*−{*j*}. *B* $$ (*i*,*i*) = *0*)
proof −
have (∑ *i*∈*I*. *B* $$ (*i*,*i*)) = *1*
using *assms* **by** (*smt* (*verit*, *best*) *carrier-matD*(*1*)
sum.cong unitarily-equiv-trace′ *unitary-diag-imp-unitarily-equiv*)
also have (∑ *i*∈*I*. (*B* $$ (*i*,*i*) ∗ *B* $$ (*i*,*i*))) = *1*
using *assms squared-A-trace*′[*of A*] **by** *simp*
hence ∃ *j*∈*I*. *B* $$ (*j*,*j*) = *1* **using** *assms pos-square-1-elem*[*of I* λ*x*.(*B* $$ (*x*, *x*))]
using *calculation* **by** *blast*
from *this* **obtain** *j* **where** *j*∈*I* **and** *B* $$ (*j*,*j*) = *1* **by** *auto*
hence ∀ *i* ∈ (*I*−{*j*}). *B* $$ (*i*,*i*) = *0*
using *assms sum-eq-elmt*[*of I* λ*x*.(*B* $$ (*x*, *x*)) *1 j*]
using *calculation* **by** *blast*
thus ∃ *j*∈*I*. *B* $$ (*j*,*j*) = *1* ∧ (∀ *i* ∈ *I*−{*j*}. *B* $$ (*i*,*i*) = *0*)
using ‹*B* $$ (*j*, *j*) = *1*› ‹*j* ∈ *I*› **by** *blast*
qed

lemma *rank-1-proj-square-trace*:
fixes *v*::*complex Matrix.vec*
assumes *A* = *rank-1-proj v*
shows *Complex-Matrix.trace* (*A*∗*A*) = ‖*v*‖^2 ∗ *Complex-Matrix.trace A*
proof −
have *Complex-Matrix.trace* (*A*∗*A*) =
Complex-Matrix.trace ((*rank-1-proj v*) ∗ *rank-1-proj v*)
using *assms* **by** *simp*
also have ... = *Complex-Matrix.trace* ((*inner-prod v v*) ·$_m$ (*outer-prod v v*))
using *outer-prod-mult-outer-prod*
unfolding *rank-1-proj-def*
by (*metis carrier-vec-dim-vec*)
also have ... = (*inner-prod v v*) ∗ *Complex-Matrix.trace* (*outer-prod v v*)

by (*metis rank-1-proj-carrier rank-1-proj-def trace-smult*)
also have ... = $\|v\|^2$ * *Complex-Matrix.trace* (*outer-prod v v*)
using *cmod-rvec-norm inner-prod-rvec-norm-pow2*
inner-prod-vec-norm-pow2 vec-norm-sq-cpx-vec-length-sq **by** *presburger*
also have ... = $\|v\|^2$ * *Complex-Matrix.trace A*
using *assms* **unfolding** *rank-1-proj-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *rank-1-proj-trace'*:
assumes *Complex-Matrix.trace* (*rank-1-proj v*) = *1*
shows $\|v\|$ = *1*
proof −
have *Complex-Matrix.trace* (*rank-1-proj v*) = *inner-prod v v* **using** *trace-outer-prod*

unfolding *rank-1-proj-def* **using** *carrier-vecI* **by** *blast*
also have ... = (*vec-norm v*)^2 **unfolding** *vec-norm-def* **using** *power2-csqrt* **by** *presburger*
also have ... = $\|v\|^2$ **using** *vec-norm-sq-cpx-vec-length-sq* **by** *simp*
finally have ... = *1* **using** *assms* **by** *simp*
thus $\|v\|$ = *1*
by (*metis cmod-vec-norm norm-neg-numeral numeral-One of-real-hom.hom-1-iff*

of-real-hom.hom-uminus one-neq-neg-one power2-eq-1-iff
vec-norm-eq-cpx-vec-length)
qed

lemma *density-square-pure*:
assumes *A* ∈ *carrier-mat n n*
and *0* < *n*
and *density-operator A*
and *Complex-Matrix.trace* (*A*∗*A*) = *1*
shows *pure-density-operator A*
proof −
define *fc*::*complex Matrix.mat set* **where** *fc* = *carrier-mat n n*
interpret *cpx-sq-mat n n fc*
proof
show *fc* = *carrier-mat n n* **unfolding** *fc-def* **by** *simp*
show *0* < *n* **using** *assms* **by** *simp*
qed *simp*
have *her*:*hermitian A* **using** *assms hermitian-def positive-is-hermitian*
by (*simp add*: *density-operator-def*)
from *this* **obtain** *B U* **where** *uni*:*real-diag-decomp A B U*
using *assms hermitian-real-diag-decomp*[*of A*]
by (*smt* (*verit*, *best*) *hermitian-decomp-decomp' hermitian-schur-decomp*)
have *exj*:∃ *j*<*dim-row A*. *B* $$ (*j*,*j*) = *1* ∧ (∀ *i*<*dim-row A*. *i*≠*j* ⟶ *B* $$ (*i*,*i*) = *0*)
proof (*rule positive-square-trace*)
show *A* ∈ *carrier-mat* (*dim-row A*) (*dim-row A*)

```
    by (simp add: ‹hermitian A› hermitian-square)
  show Complex-Matrix.trace A = complex-of-real 1
    using assms density-operator-def by simp
  show Complex-Matrix.trace (A * A) = 1
    using assms by simp
  show real-diag-decomp A B U
    by (simp add: ‹real-diag-decomp A B U›)
  show Complex-Matrix.positive A
    using assms density-operator-def by simp
  show 0 < dim-row A using assms npos
    by (metis carrier-matD(1))
qed
from this obtain j where jdim:j<dim-row A and j1:B $$ (j,j) = 1
  and ji0:(∀ i<dim-row A. i≠j ⟶ B $$ (i,i) = 0) by auto
have dim-row B = dim-row A using ‹real-diag-decomp A B U›
      unitarily-equivD real-diag-decomp-def similar-mat-wit-dim-row
        unitary-diag-imp-unitarily-equiv by blast
hence diag-mat B ! j = 1 using j1 jdim
  unfolding diag-mat-def
  by simp
have insj:{..< dim-row A} = insert j ({..< dim-row A}−{j})
  using jdim by blast
have A = sum-mat (λi. (diag-mat B ! i) ·_m rank-1-proj (Matrix.col U i))
    {..< dim-row A}
  using assms sum-decomp-cols ‹hermitian A› real-diag-decompD(1)
  by (simp add: ‹real-diag-decomp A B U› fc-mats-carrier)
also have ... = (diag-mat B ! j) ·_m rank-1-proj (Matrix.col U j)
proof (rule sum-mat-singleton)
  have ⋀i. i < dim-row A ⟹ rank-1-proj (Matrix.col U i) ∈ fc
  proof −
    fix i
    assume i<dim-row A
    have dim-vec (Matrix.col U i) = n using ‹real-diag-decomp A B U› assms
      by (metis carrier-matD(1) dim-col fc-mats-carrier
          real-diag-decompD(1) unitary-diag-carrier(2))
    thus rank-1-proj (Matrix.col U i) ∈ fc using rank-1-proj-carrier
       fc-mats-carrier dim-eq
      by blast
  qed
  thus (λi. rank-1-proj (Matrix.col U i)) ‘ {..<dim-row A} ⊆ fc by auto
  show ∀ i∈{..<dim-row A}. i ≠ j ⟶ diag-mat B ! i = 0
  proof (intro ballI impI)
    fix i
    assumei∈ {..<dim-row A}
    and i ≠ j
    have diag-mat B ! i = B $$ (i,i) using ‹i∈ {..<dim-row A}›
       ‹dim-row B = dim-row A›
      unfolding diag-mat-def by simp
    thus diag-mat B ! i = 0 using ‹i≠j› ji0
```

```
      using ‹i ∈ {..<dim-row A}› by simp
    qed
  qed (auto simp add: jdim)
  also have ... = rank-1-proj (Matrix.col U j)
    using ‹diag-mat B ! j = 1› by auto
  finally have A = rank-1-proj (Matrix.col U j) .
  thus pure-density-operator A
    unfolding pure-density-operator-def by auto
qed

lemma density-square-pure':
  assumes density-operator A
  and A = rank-1-proj v
shows Complex-Matrix.trace (A*A) = 1
proof −
  have Complex-Matrix.trace (A*A) = ‖v‖² * Complex-Matrix.trace A
    using assms by (simp add: rank-1-proj-square-trace)
  also have ... = Complex-Matrix.trace A
    using rank-1-proj-trace' assms unfolding density-operator-def
    by simp
  also have ... = 1 using assms unfolding density-operator-def
    by simp
  finally show ?thesis by auto
qed

lemma
  assumes A∈ carrier-mat n n
  and 0 < n
  and density-operator A
shows pure-density-charact:
  (pure-density-operator A) ⟷ (Complex-Matrix.trace (A*A) = 1)
and pure-density-charact':
  (pure-density-operator A) ⟷ (A*A = A)
proof −
  show (pure-density-operator A) ⟷ (Complex-Matrix.trace (A*A) = 1)
  using assms density-square-pure density-square-pure'
    pure-density-operator-def[of A] by auto
next
  show (pure-density-operator A) ⟷ (A*A = A)
  proof
    assume pure-density-operator A
    hence ∃ v. A = rank-1-proj v unfolding pure-density-operator-def by simp
    from this obtain v where A = rank-1-proj v by auto
    have 1 = Complex-Matrix.trace A
      using assms unfolding density-operator-def by simp
    also have ... = ‖v‖² using trace-rank-1-proj ‹A = rank-1-proj v› by simp
    finally have ‖v‖ = 1
      by (simp add: ‹1 = Complex-Matrix.trace A› ‹A = rank-1-proj v›
          rank-1-proj-trace')
```

thus $A*A = A$ **using** *rank-1-proj-projector* ‹A = *rank-1-proj* v›
unfolding *projector-def* **by** *simp*
next
assume $A*A = A$
hence *Complex-Matrix.trace* $(A*A)$ = *Complex-Matrix.trace* A **by** *simp*
also have ... = *1* **using** *assms* **unfolding** *density-operator-def* **by** *simp*
finally have *Complex-Matrix.trace* $(A*A) = 1$ **.**
thus *pure-density-operator* A **using** *assms density-square-pure* **by** *simp*
qed
qed

8 Quantum expectation values and traces

The expectation value of a projective measurement is the average outcome value of the measurement, where each outcome value is weighted by the probability that it occurs. We show that the expectation value of a density matrix ρ for an observable represented by the Hermitian matrix A is $\mathrm{Tr}(A{\cdot}\rho)$.

definition (**in** *cpx-sq-mat*) *expect-value* **where**
expect-value R p M =
sum $(\lambda i.$ *meas-outcome-prob* R M i $*$ (*meas-outcome-val* $(M\ i)))$ $\{..< p\}$

definition (**in** *cpx-sq-mat*) *obs-expect-value* **where**
obs-expect-value R A =
expect-value R (*proj-meas-size* (*make-pm* A)) (*proj-meas-outcomes* (*make-pm* A))

lemma (**in** *cpx-sq-mat*) *expect-value-trace*:
assumes *proj-measurement* p M
and $R\in$ *fc-mats*
shows *expect-value* R p M =
Complex-Matrix.trace (*sum-mat*
$(\lambda i.$ *meas-outcome-val* $(M\ i)\cdot_m$ (*meas-outcome-prj* $(M\ i)))$ $\{..< p\} * R)$
proof −
have *car*: $\bigwedge i.\ i < p \Longrightarrow$ *meas-outcome-prj* $(M\ i) * R \in$ *fc-mats*
using *assms* **unfolding** *proj-measurement-def*
using *cpx-sq-mat-mult* **by** *auto*
have *expect-value* R p M = *sum* $(\lambda i.$ *meas-outcome-val* $(M\ i)$ $*$
(*Complex-Matrix.trace*($R*$ *meas-outcome-prj* $(M\ i))))$ $\{..< p\}$
unfolding *expect-value-def meas-outcome-prob-def*
by (*simp add*: *mult.commute*)
also have ... = *sum* $(\lambda i.$ *meas-outcome-val* $(M\ i)$ $*$
(*Complex-Matrix.trace*(*meas-outcome-prj* $(M\ i) * R))$) $\{..< p\}$
proof −
have $\bigwedge i.\ i < p \Longrightarrow$ *Complex-Matrix.trace* $(R *$ *meas-outcome-prj* $(M\ i)) =$
Complex-Matrix.trace (*meas-outcome-prj* $(M\ i) * R)$
using *assms dim-eq fc-mats-carrier trace-comm*
unfolding *proj-measurement-def* **by** *auto*
thus *?thesis* **by** *simp*
qed

```
  also have ... = sum (λi. (Complex-Matrix.trace
    (meas-outcome-val (M i)·m meas-outcome-prj (M i) ∗ R))) {..< p}
  proof −
    have ⋀i. i < p ⟹ meas-outcome-val (M i) ∗
    (Complex-Matrix.trace(meas-outcome-prj (M i) ∗ R)) =
    Complex-Matrix.trace (meas-outcome-val (M i)·m meas-outcome-prj (M i)∗ R)
    proof −
      fix i
      assume i < p
      hence meas-outcome-val (M i) ∗
        (Complex-Matrix.trace(meas-outcome-prj (M i) ∗ R)) =
       Complex-Matrix.trace (meas-outcome-val (M i)·m (meas-outcome-prj (M i)∗
R))
        using assms car
        by (metis dim-eq fc-mats-carrier trace-smult)
      also have ... =  Complex-Matrix.trace
        (meas-outcome-val (M i)·m meas-outcome-prj (M i)∗ R)
      proof −
        have meas-outcome-val (M i)·m (meas-outcome-prj (M i)∗ R) =
          meas-outcome-val (M i)·m meas-outcome-prj (M i)∗ R
          using car assms unfolding proj-measurement-def
          by (metis ‹i < p› dim-eq fc-mats-carrier mult-smult-assoc-mat)
        thus ?thesis by simp
      qed
      finally show meas-outcome-val (M i) ∗
        (Complex-Matrix.trace(meas-outcome-prj (M i) ∗ R)) =
        Complex-Matrix.trace
        (meas-outcome-val (M i)·m meas-outcome-prj (M i)∗ R) .
    qed
    thus ?thesis by simp
  qed
  also have ... = Complex-Matrix.trace (sum-mat
    (λi. meas-outcome-val (M i)·m (meas-outcome-prj (M i)) ∗ R) {..< p})
  proof (rule trace-sum-mat[symmetric])
    fix i
    assume i ∈ {..< p}
    hence meas-outcome-val (M i) ·m meas-outcome-prj (M i) ∈ fc-mats
      using assms cpx-sq-mat-smult[of  meas-outcome-prj (M i)]
      unfolding proj-measurement-def by simp
    thus meas-outcome-val (M i) ·m meas-outcome-prj (M i) ∗ R ∈ fc-mats
      by (simp add: assms(2) cpx-sq-mat-mult)
  qed simp
  also have ... = Complex-Matrix.trace (sum-mat
    (λi. meas-outcome-val (M i)·m (meas-outcome-prj (M i))) {..< p} ∗ R)
  proof −
    have sum-mat (λi. meas-outcome-val (M i)·m (meas-outcome-prj (M i)) ∗ R)
      {..< p} = sum-mat
      (λi. meas-outcome-val (M i)·m (meas-outcome-prj (M i))) {..< p} ∗ R
    proof (rule sum-mat-distrib-right)
```

```
      show ⋀i. i ∈ {..<p} ⟹ meas-outcome-val (M i) ·m meas-outcome-prj (M
i) ∈
        fc-mats
      proof −
        fix i
        assume i ∈ {..<p}
        thus meas-outcome-val (M i) ·m meas-outcome-prj (M i) ∈ fc-mats
          using assms cpx-sq-mat-smult[of  meas-outcome-prj (M i)]
          unfolding proj-measurement-def by simp
      qed
    qed (auto simp add: assms)
    thus ?thesis by simp
  qed
  finally show ?thesis .
qed

lemma (in cpx-sq-mat) expect-value-hermitian:
  assumes A∈ fc-mats
  and hermitian A
  and make-pm A = (p, M)
  and R∈ fc-mats
shows expect-value R p M = Complex-Matrix.trace (A ∗ R)
proof −
  have expect-value R p M = Complex-Matrix.trace (sum-mat
    (λi. meas-outcome-val (M i)·m (meas-outcome-prj (M i))) {..< p} ∗ R)
    using assms make-pm-proj-measurement expect-value-trace by simp
  also have ... = Complex-Matrix.trace (A ∗ R)
  proof −
    have sum-mat (λi. meas-outcome-val (M i)·m (meas-outcome-prj (M i)))
      {..< p} = A
      using make-pm-sum assms by simp
    thus ?thesis by simp
  qed
  finally show ?thesis .
qed

lemma obs-expect-value:
  assumes A∈ carrier-mat n n
  and hermitian A
  and R∈ carrier-mat n n
  and 0 < n
shows cpx-sq-mat.obs-expect-value n n R A = Complex-Matrix.trace (A ∗ R)
proof −
  define fc::complex Matrix.mat set
  where fc = carrier-mat n n
  interpret cpx-sq-mat n n fc
  proof
    show fc = carrier-mat n n using fc-def by simp
    show 0 < n using assms by simp
```

qed *simp*
show *?thesis* **unfolding** *obs-expect-value-def*
proof (*rule expect-value-hermitian*)
show *make-pm A*=(*proj-meas-size* (*make-pm A*), *proj-meas-outcomes* (*make-pm A*))
using *make-pm-decomp* **by** *simp*
qed (*auto simp add*: *assms fc-def*)
qed

end

theory *Tsirelson*
imports
Projective-Measurements.CHSH-Inequality
Matrix-L2-Operator-Norm Density-Matrix-Basics

begin

This part contains a formalization of the CHSH operator and the CHSH quantum expectation, along with Tsirelson's proof that this quantum expectation cannot be greater than $2 \cdot \sqrt{2}$. The development of this proof permits to extract the additional result that when one of the parties involved in the CHSH experiment makes measurements on commuting observables, the quantum expectation cannot be greater than 2. This is the same upper-bound as in the case where a local hidden variable hypothesis is made.

9 CHSH inequalities

The CHSH operator is used to represent the experiment in which two parties each perform measurements using two observables, respectively A_1, A_2 and B_1, B_2. Given the resource R, in general a density matrix representing an entangled state, the CHSH expectation represents the quantum expectation of performing simultaneous measurements on R. The CHSH setting also assumes that along with being Hermitian matrics, all the squared observables are equal to the identity and commute with the observables of the other party.

9.1 Some intermediate results for particular observables

lemma *chsh-complex*:
fixes *A0*::*complex*
assumes *A0* ∈ *Reals*
and *B0* ∈ *Reals*
and *A1* ∈ *Reals*
and *B1* ∈ *Reals*

```
  and |A0 * B1| ≤ 1
  and |A0 * B0| ≤ 1
  and |A1 * B0| ≤ 1
  and |A1 * B1| ≤ 1
shows |A0 * B1 − A0 * B0 + A1 * B0 + A1*B1| ≤ 2
proof −
  have |A0 * B1 − A0 * B0 + A1 * B0 + A1*B1| =
    |Re A0 * (Re B1) − Re A0 * (Re B0) + Re A1 * (Re B0) + Re A1 * (Re
B1)|
    using assms by (simp add: cpx-real-abs-eq)
  moreover have |Re A0 * (Re B1) − Re A0 * (Re B0) +
    Re A1 * (Re B0) + Re A1 * (Re B1)| ≤ 2
  proof (rule chsh-real)
    show |Re A0 * Re B1| ≤ 1 using assms real-cpx-abs-leq by simp
    show |Re A1 * Re B1| ≤ 1 using assms real-cpx-abs-leq by simp
    show |Re A0 * Re B0| ≤ 1 using assms real-cpx-abs-leq by simp
    show |Re A1 * Re B0| ≤ 1 using assms real-cpx-abs-leq by simp
  qed
  ultimately show ?thesis
    by (simp add: less-eq-complex-def)
qed


lemma (in bin-cpx) Z-XpZ-rho-trace:
  shows Complex-Matrix.trace (Z-I * I-XpZ * rho-psim) = 1/sqrt 2
proof −
  have Complex-Matrix.trace (Z-I * I-XpZ * rho-psim) =
    Complex-Matrix.trace (Z-XpZ * rho-psim)
    by (simp add: Z-I-XpZ-eq)
  also have ... = Complex-Matrix.trace (rho-psim * Z-XpZ)
  proof (rule trace-comm)
    show Z-XpZ ∈ carrier-mat 4 4 using Z-XpZ-carrier .
    show rho-psim ∈ carrier-mat 4 4 using rho-psim-carrier .
  qed
  also have ... = 1/sqrt 2 by simp
  finally show ?thesis .
qed

lemma (in bin-cpx) X-XpZ-rho-trace:
  shows Complex-Matrix.trace (X-I * I-XpZ * rho-psim) = 1/sqrt 2
proof −
  have Complex-Matrix.trace (X-I * I-XpZ * rho-psim) =
    Complex-Matrix.trace (X-XpZ * rho-psim)
    by (simp add: X-I-XpZ-eq)
  also have ... = Complex-Matrix.trace (rho-psim * X-XpZ)
  proof (rule trace-comm)
    show X-XpZ ∈ carrier-mat 4 4 using X-XpZ-carrier .
    show rho-psim ∈ carrier-mat 4 4 using rho-psim-carrier .
  qed
```

also have ... = *1/sqrt 2* **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *bin-cpx*) *X-ZmX-rho-trace*:
shows *Complex-Matrix.trace* (*X-I* ∗ *I-ZmX* ∗ *rho-psim*) = *1/sqrt 2*
proof −
have *Complex-Matrix.trace* (*X-I* ∗ *I-ZmX* ∗ *rho-psim*) =
Complex-Matrix.trace (*X-ZmX* ∗ *rho-psim*)
by (*simp add*: *X-I-ZmX-eq*)
also have ... = *Complex-Matrix.trace* (*rho-psim* ∗ *X-ZmX*)
proof (*rule trace-comm*)
show *X-ZmX* ∈ *carrier-mat 4 4* **using** *X-ZmX-carrier* **.**
show *rho-psim* ∈ *carrier-mat 4 4* **using** *rho-psim-carrier* **.**
qed
also have ... = *1/sqrt 2* **by** *simp*
finally show *?thesis* **.**
qed

lemma (**in** *bin-cpx*) *Z-ZmX-rho-trace*:
shows *Complex-Matrix.trace* (*Z-I* ∗ *I-ZmX* ∗ *rho-psim*) = −*1/sqrt 2*
proof −
have *Complex-Matrix.trace* (*Z-I* ∗ *I-ZmX* ∗ *rho-psim*) =
Complex-Matrix.trace (*Z-ZmX* ∗ *rho-psim*)
by (*simp add*: *Z-I-ZmX-eq*)
also have ... = *Complex-Matrix.trace* (*rho-psim* ∗ *Z-ZmX*)
proof (*rule trace-comm*)
show *Z-ZmX* ∈ *carrier-mat 4 4* **using** *Z-ZmX-carrier* **.**
show *rho-psim* ∈ *carrier-mat 4 4* **using** *rho-psim-carrier* **.**
qed
also have ... = −*1/sqrt 2* **by** *simp*
finally show *?thesis* **.**
qed

9.2 The CHSH operator and expectation

definition *CHSH-op* :: *'a*::*conjugatable-field Matrix.mat* ⇒ *'a Matrix.mat* ⇒
'a Matrix.mat ⇒ *'a Matrix.mat* ⇒ *'a Matrix.mat*
where
CHSH-op A0 A1 B0 B1 = *A0* ∗ *B1* − *A0* ∗ *B0* + *A1* ∗ *B0* + *A1* ∗ *B1*

definition *CHSH-expect* :: *'a*::*conjugatable-field Matrix.mat* ⇒ *'a Matrix.mat* ⇒
'a Matrix.mat ⇒ *'a Matrix.mat* ⇒ *'a Matrix.mat* ⇒ *'a*
where
CHSH-expect A0 A1 B0 B1 R = *Complex-Matrix.trace* ((*CHSH-op A0 A1 B0 B1*)
∗ *R*)

definition *CHSH-cond* :: *nat* ⇒ *'a*::*conjugatable-field Matrix.mat* ⇒
'a::*conjugatable-field Matrix.mat* ⇒

'a::conjugatable-field Matrix.mat ⇒ 'a::conjugatable-field Matrix.mat ⇒ bool
where
CHSH-cond n A0 A1 B0 B1 =
(A0 ∈ carrier-mat n n ∧
A0 ∗ A0 = 1$_m$ n ∧
A1 ∈ carrier-mat n n ∧
A1 ∗ A1 = 1$_m$ n ∧
B0 ∈ carrier-mat n n ∧
B0 ∗ B0 = 1$_m$ n ∧
B1 ∈ carrier-mat n n ∧
B1 ∗ B1 = 1$_m$ n ∧
A0 ∗ B1 = B1 ∗ A0 ∧
A0 ∗ B0 = B0 ∗ A0 ∧
A1 ∗ B0 = B0 ∗ A1 ∧
A1 ∗ B1 = B1 ∗ A1)

definition *CHSH-cond-hermit* **where**
CHSH-cond-hermit n A0 A1 B0 B1 ⟷ CHSH-cond n A0 A1 B0 B1 ∧ hermitian A0 ∧
hermitian A1 ∧ hermitian B0 ∧ hermitian B1

lemma *CHSH-op-dim*:
assumes *A0 ∈ carrier-mat n m*
and *A1 ∈ carrier-mat n m*
and *B0 ∈ carrier-mat m p*
and *B1 ∈ carrier-mat m p*
shows *CHSH-op A0 A1 B0 B1 ∈ carrier-mat n p* **unfolding** *CHSH-op-def*
using *assms* **by** *simp*

lemma *CHSH-op-hermitian*:
assumes *hermitian A0*
and *hermitian B0*
and *hermitian A1*
and *hermitian B1*
and *A0 ∗ B0 = B0 ∗ A0*
and *A1 ∗ B0 = B0 ∗ A1*
and *A0 ∗ B1 = B1 ∗ A0*
and *A1 ∗ B1 = B1 ∗ A1*
shows *hermitian (CHSH-op A0 A1 B0 B1)*
using *assms hermitian-add hermitian-def hermitian-minus hermitian-square*
index-add-mat(2) index-minus-mat(2) index-mult-mat(2)
unfolding *CHSH-op-def*
by *(smt (z3) Linear-Algebra-Complements.hermitian-square adjoint-mult)*

lemma *CHSH-cond-hermit-expect-eq*:
assumes *CHSH-cond-hermit n A0 A1 B0 B1*
and *R∈ carrier-mat n n*
and *0 < n*
shows *CHSH-expect A0 A1 B0 B1 R =*

```
    cpx-sq-mat.obs-expect-value n n  R (CHSH-op A0 A1 B0 B1)
  unfolding CHSH-expect-def
proof (rule obs-expect-value[symmetric])
  show hermitian (CHSH-op A0 A1 B0 B1) using CHSH-op-hermitian assms
    unfolding CHSH-cond-hermit-def CHSH-cond-def by metis
  show CHSH-op A0 A1 B0 B1 ∈ carrier-mat n n
    using assms unfolding CHSH-cond-hermit-def CHSH-cond-def
    by (meson CHSH-op-dim)
qed (auto simp add: assms)

lemma CHSH-op-expand-right:
  fixes A0::'a::conjugatable-field Matrix.mat
  assumes A0 ∈ carrier-mat n m
  and A1 ∈ carrier-mat n m
  and B0 ∈ carrier-mat m p
  and B1 ∈ carrier-mat m p
  and R ∈ carrier-mat p p'
shows (CHSH-op A0 A1 B0 B1) * R =
  A0 * B1 * R - A0 * B0 * R + A1 * B0 * R + A1 * B1 * R
proof -
  have (CHSH-op A0 A1 B0 B1) * R =
   (A0 * B1 - A0 * B0 + A1 * B0) * R + A1 * B1 * R unfolding CHSH-op-def
    by (meson add-carrier-mat add-mult-distrib-mat assms(2) assms(3)
        assms(4) assms(5) mult-carrier-mat)
  also have ... = (A0 * B1 - A0 * B0) * R + A1 * B0 * R + A1 * B1 * R
    by (metis add-mult-distrib-mat assms(1) assms(2) assms(3) assms(5)
        minus-carrier-mat mult-carrier-mat)
  also have ... = A0 * B1 * R - A0 * B0 * R + A1 * B0 * R + A1 * B1 * R
    by (metis assms(1) assms(3) assms(4) assms(5) minus-mult-distrib-mat
        mult-carrier-mat)
  finally show ?thesis .
qed

lemma CHSH-op-expand-left:
  fixes A0::'a::conjugatable-field Matrix.mat
  assumes A0 ∈ carrier-mat n m
  and A1 ∈ carrier-mat n m
  and B0 ∈ carrier-mat m p
  and B1 ∈ carrier-mat m p
  and R ∈ carrier-mat p n
shows R * (CHSH-op A0 A1 B0 B1) =
  R * (A0 * B1) - R * (A0 * B0) + R * (A1 * B0) + R * (A1 * B1)
proof -
  have R * (CHSH-op A0 A1 B0 B1) =
     R * (A0 * B1 - A0 * B0 + A1 * B0) + R * (A1 * B1) unfolding
CHSH-op-def
    using mult-add-distrib-mat[of R p n - p A1 * B1] assms by simp
  also have ... = R * (A0 * B1 - A0 * B0) + R * (A1 * B0) + R * (A1 * B1)
    using mult-add-distrib-mat assms
```

by (*metis minus-carrier-mat mult-carrier-mat*)
also have ... = $R * (A0 * B1) - R * (A0 * B0) + R * (A1 * B0) +$
$R * (A1 * B1)$
using *mult-minus-distrib-mat*[*of R p n A0 * B1 p*] *assms* **by** *simp*
finally show *?thesis* **.**
qed

lemma *CHSH-expect-expand*:
assumes $A0 \in$ *carrier-mat n m*
and $A1 \in$ *carrier-mat n m*
and $B0 \in$ *carrier-mat m p*
and $B1 \in$ *carrier-mat m p*
and $R \in$ *carrier-mat p n*
shows *CHSH-expect A0 A1 B0 B1 R* =
Complex-Matrix.trace $(A0 * B1 * R) -$
Complex-Matrix.trace $(A0 * B0 * R) +$
Complex-Matrix.trace $(A1 * B0 * R) +$
Complex-Matrix.trace $(A1 * B1 * R)$
proof −
have *CHSH-expect A0 A1 B0 B1 R* =
Complex-Matrix.trace $(A0 * B1 * R - A0 * B0 * R + A1 * B0 * R +$
$A1 * B1 * R)$
unfolding *CHSH-expect-def* **using** *CHSH-op-expand-right*[*of A0*] *assms* **by** *simp*
also have ... = *Complex-Matrix.trace* $(A0 * B1 * R) -$
Complex-Matrix.trace $(A0 * B0 * R) +$
Complex-Matrix.trace $(A1 * B0 * R) +$
Complex-Matrix.trace $(A1 * B1 * R)$
by (*meson assms mult-carrier-mat trace-ch-expand*)
finally show *?thesis* **.**
qed

lemma *CHSH-condD*:
assumes *CHSH-cond n A0 A1 B0 B1*
shows $A0 \in$ *carrier-mat n n*
$A0 * A0 = 1_m\ n$
$A1 \in$ *carrier-mat n n*
$A1 * A1 = 1_m\ n$
$B0 \in$ *carrier-mat n n*
$B0 * B0 = 1_m\ n$
$B1 \in$ *carrier-mat n n*
$B1 * B1 = 1_m\ n$
$A0 * B1 = B1 * A0$
$A0 * B0 = B0 * A0$
$A1 * B0 = B0 * A1$
$A1 * B1 = B1 * A1$ **using** *assms* **unfolding** *CHSH-cond-def* **by** *auto*

lemma *CHSH-cond-simps*[*simp*]:
assumes *CHSH-cond n A0 A1 B0 B1*

shows $A1 * B1 * (A0 * B1) = A1{*}A0$
$A1 * B1 * (A1 * B0) = B1 * B0$
$A1 * B1 * (A1 * B1) = 1_m\ n$
$A1 * B1 * (A0 * B0) = A1 * A0 * (B1 * B0)$
$A1 * B0 * (A0 * B1) = A1 * A0 * (B0 * B1)$
$A1 * B0 * (A0 * B0) = A1 * A0$
$A1 * B0 * (A1 * B0) = 1_m\ n$
$A1 * B0 * (A1 * B1) = B0 * B1$
$A0 * B0 * (A0 * B1) = B0 * B1$
$A0 * B0 * (A0 * B0) = 1_m\ n$
$A0 * B0 * (A1 * B0) = A0 * A1$
$A0 * B0 * (A1 * B1) = A0 * A1 * (B0 * B1)$
$A0 * B1 * (A0 * B1) = 1_m\ n$
$A0 * B1 * (A0 * B0) = B1 * B0$
$A0 * B1 * (A1 * B0) = A0 * A1 * (B1 * B0)$
$A0 * B1 * (A1 * B1) = A0 * A1$
proof −
show $A1 * B1 * (A0 * B1) = A1{*}A0$ **using** *assms* **unfolding** *CHSH-cond-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A1 * B1 * (A1 * B0) = B1 * B0$ **using** *assms* **unfolding** *CHSH-cond-def*

by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A1 * B1 * (A1 * B1) = 1_m\ n$ **using** *assms* **unfolding** *CHSH-cond-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A1 * B1 * (A0 * B0) = A1 * A0 * (B1 * B0)$
using *assms* **unfolding** *CHSH-cond-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A1 * B0 * (A0 * B1) = A1 * A0 * (B0 * B1)$
using *assms* **unfolding** *CHSH-cond-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A1 * B0 * (A0 * B0) = A1 * A0$ **using** *assms* **unfolding** *CHSH-cond-def*

by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A1 * B0 * (A1 * B0) = 1_m\ n$ **using** *assms* **unfolding** *CHSH-cond-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A1 * B0 * (A1 * B1) = B0 * B1$ **using** *assms* **unfolding** *CHSH-cond-def*

by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A0 * B0 * (A0 * B1) = B0 * B1$ **using** *assms* **unfolding** *CHSH-cond-def*

by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A0 * B0 * (A0 * B0) = 1_m\ n$ **using** *assms* **unfolding** *CHSH-cond-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A0 * B0 * (A1 * B0) = A0 * A1$ **using** *assms* **unfolding** *CHSH-cond-def*

by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)
show $A0 * B0 * (A1 * B1) = A0 * A1 * (B0 * B1)$
using *assms* **unfolding** *CHSH-cond-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat right-mult-one-mat*)

```
  show  A0 * B1 * (A0 * B1) = 1_m n using assms unfolding CHSH-cond-def
    by (smt (verit) assoc-mult-mat mult-carrier-mat right-mult-one-mat)
  show  A0 * B1 * (A0 * B0) = B1 * B0 using assms unfolding CHSH-cond-def

    by (smt (verit) assoc-mult-mat mult-carrier-mat right-mult-one-mat)
  show  A0 * B1 * (A1 * B0) = A0 * A1 * (B1 * B0)
    using assms unfolding CHSH-cond-def
    by (smt (verit) assoc-mult-mat mult-carrier-mat right-mult-one-mat)
  show  A0 * B1 * (A1 * B1) = A0 * A1 using assms unfolding CHSH-cond-def
    by (smt (verit) assoc-mult-mat mult-carrier-mat right-mult-one-mat)
qed

lemma CHSH-op-square:
  assumes CHSH-cond n A0 A1 B0 B1
shows (CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1) =
  (4::nat) ·_m (1_m n) − (commutator A0 A1) * (commutator B0 B1)
proof −
  have (CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1) =
    A0 * B1 * (CHSH-op A0 A1 B0 B1) − A0 * B0 * (CHSH-op A0 A1 B0 B1)
+
    A1 * B0 * (CHSH-op A0 A1 B0 B1) + A1 * B1 * (CHSH-op A0 A1 B0 B1)

  proof (rule CHSH-op-expand-right)
    show A0 ∈ carrier-mat n n using assms unfolding CHSH-cond-def by simp
    show A1 ∈ carrier-mat n n using assms unfolding CHSH-cond-def by simp
    show B0 ∈ carrier-mat n n using assms unfolding CHSH-cond-def by simp
    show B1 ∈ carrier-mat n n using assms unfolding CHSH-cond-def by simp
    show CHSH-op A0 A1 B0 B1 ∈ carrier-mat n n
      using assms CHSH-op-dim[of A0]
      unfolding CHSH-cond-def by force
  qed
  also have ... = A0 * B1 * (CHSH-op A0 A1 B0 B1) −
    A0 * B0 * (CHSH-op A0 A1 B0 B1) +
    A1 * B0 * (CHSH-op A0 A1 B0 B1) +
    (A1 * B1 * (A0 * B1) − A1 * B1 * (A0 * B0) + A1 * B1 * (A1 * B0) +
    A1 * B1 * (A1 * B1))
    using assms CHSH-op-expand-left[of A0  n n A1 B0 n B1 A1*B1]
      unfolding CHSH-cond-def by auto
  also have ... = A0 * B1 * (CHSH-op A0 A1 B0 B1) −
    A0 * B0 * (CHSH-op A0 A1 B0 B1) +
    A1 * B0 * (CHSH-op A0 A1 B0 B1) +
    (A1*A0 − A1 * A0 * (B1 * B0) + B1 * B0 + 1_m n) using assms by simp
  also have ... = A0 * B1 * (CHSH-op A0 A1 B0 B1) −
    A0 * B0 * (CHSH-op A0 A1 B0 B1) +
    (A1 * B0 * (A0 * B1) − A1 * B0 * (A0 * B0) + A1 * B0 * (A1 * B0) +
    A1 * B0 * (A1 * B1)) +
    (A1*A0 − A1 * A0 * (B1 * B0) + B1 * B0 + 1_m n)
    using assms CHSH-op-expand-left[of A0  n n A1 B0 n B1 A1*B0]
    unfolding CHSH-cond-def by auto
```

also have ... = $A0 * B1 * (CHSH\text{-}op\ A0\ A1\ B0\ B1) -$
$A0 * B0 * (CHSH\text{-}op\ A0\ A1\ B0\ B1) +$
$(A1 * A0 * (B0 * B1) - A1 * A0\ + 1_m\ n + B0 * B1) +$
$(A1 * A0 - A1 * A0 * (B1 * B0) + B1 * B0 + 1_m\ n)$ **using** *assms* **by** *simp*
also have ... = $A0 * B1 * (CHSH\text{-}op\ A0\ A1\ B0\ B1) -$
$(A0 * B0 * (A0 * B1) - A0 * B0 * (A0 * B0) + A0 * B0 * (A1 * B0) +$
$A0 * B0 * (A1 * B1)) +$
$(A1 * A0 * (B0 * B1) - A1 * A0\ + 1_m\ n + B0 * B1) +$
$(A1 * A0 - A1 * A0 * (B1 * B0) + B1 * B0 + 1_m\ n)$
using *assms CHSH-op-expand-left*[*of A0 n n A1 B0 n B1 A0∗B0*]
unfolding *CHSH-cond-def* **by** *auto*
also have ... = $A0 * B1 * (CHSH\text{-}op\ A0\ A1\ B0\ B1) -$
$(B0∗\ B1 - 1_m\ n + A0 * A1 + A0 * A1 * (B0 * B1)) +$
$(A1 * A0 * (B0 * B1) - A1 * A0\ + 1_m\ n + B0 * B1) +$
$(A1 * A0 - A1 * A0 * (B1 * B0) + B1 * B0 + 1_m\ n)$
using *assms* **by** *simp*
also have ... =
$(A0 * B1 * (A0 * B1) - A0 * B1 * (A0 * B0) + A0 * B1 * (A1 * B0) +$
$A0 * B1 * (A1 * B1)) -$
$(B0∗\ B1 - 1_m\ n + A0 * A1 + A0 * A1 * (B0 * B1)) +$
$(A1 * A0 * (B0 * B1) - A1 * A0\ + 1_m\ n + B0 * B1) +$
$(A1 * A0 - A1 * A0 * (B1 * B0) + B1 * B0 + 1_m\ n)$
using *assms CHSH-op-expand-left*[*of A0 n n A1 B0 n B1 A0∗B1*]
unfolding *CHSH-cond-def* **by** *auto*
also have ... = $(1_m\ n - B1 *\ B0 + A0 * A1 * (B1 * B0) + A0 * A1) -$
$(B0 * B1 - 1_m\ n + A0 * A1 + A0 * A1 * (B0 * B1)) +$
$(A1 * A0 * (B0 * B1) - A1 * A0\ + 1_m\ n + B0 * B1) +$
$(A1 * A0 - A1 * A0 * (B1 * B0) + B1 * B0 + 1_m\ n)$
using *assms* **by** *simp*
also have ... = $(1_m\ n\ + A0 * A1 * (B1 * B0)\ + 1_m\ n\ -$
$A0 * A1 * (B0 * B1)) +$
$(A1 * A0 * (B0 * B1)\ + 1_m\ n) - A1 * A0 * (B1 * B0)\ + 1_m\ n$
using *assms* **unfolding** *CHSH-cond-def*
by (*auto simp add*: *algebra-simps*)
also have ... = $(4::nat)\cdot_m\ 1_m\ n -$
$(A0 * A1 * (B0 * B1) - A0 * A1 * (B1 * B0) - A1 * A0 *(B0 * B1) +$
$A1 * A0 * (B1 * B0))$
using *assms* **unfolding** *CHSH-cond-def*
by (*auto simp add*: *algebra-simps*)
also have ... = $(4::nat)\cdot_m\ 1_m\ n - (commutator\ A0\ A1) * (commutator\ B0\ B1)$
using *assms commutator-mult-expand*[*of A0 n A1*]
unfolding *CHSH-cond-def* **by** *simp*
finally show *?thesis* **.**
qed

lemma *CHSH-cond-hermitD*:
assumes *CHSH-cond-hermit n A0 A1 B0 B1*
shows *CHSH-cond n A0 A1 B0 B1*
hermitian A0

hermitian A1
hermitian B0
hermitian B1
using *assms* **unfolding** *CHSH-cond-hermit-def* **by** *auto*

lemma *CHSH-cond-hermit-unitary*:
assumes *CHSH-cond-hermit n A0 A1 B0 B1*
shows *unitary A0 unitary A1 unitary B0 unitary B1*
using *assms* **unfolding** *CHSH-cond-hermit-def CHSH-cond-def*
by (*metis Complex-Matrix.unitary-def carrier-matD*(*1*)
hermitian-def inverts-mat-def)+

lemma *CHSH-expect-add*:
assumes *A0* ∈ *carrier-mat n n*
and *A1* ∈ *carrier-mat n n*
and *B0* ∈ *carrier-mat n n*
and *B1* ∈ *carrier-mat n n*
and *R0* ∈ *carrier-mat n n*
and *R1* ∈ *carrier-mat n n*
shows *CHSH-expect A0 A1 B0 B1* (*R0* + *R1*) =
CHSH-expect A0 A1 B0 B1 R0 +
CHSH-expect A0 A1 B0 B1 R1
proof −
note *chsh* = *CHSH-op-dim*[*OF assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)]
have *CHSH-op A0 A1 B0 B1* ∗ (*R0* + *R1*) =
CHSH-op A0 A1 B0 B1 ∗ *R0* + *CHSH-op A0 A1 B0 B1* ∗ *R1*
using *mult-add-distrib-mat*[*OF chsh*] *assms* **by** *auto*
thus *?thesis*
using *assms trace-add-linear* **unfolding** *CHSH-expect-def*
by (*metis chsh mult-carrier-mat*)
qed

lemma *CHSH-expect-zero*:
assumes *A0* ∈ *carrier-mat n n*
and *A1* ∈ *carrier-mat n n*
and *B0* ∈ *carrier-mat n n*
and *B1* ∈ *carrier-mat n n*
shows *CHSH-expect A0 A1 B0 B1* (0$_m$ *n n*) = *0*
using *CHSH-expect-expand assms*
proof −
have *CHSH-expect A0 A1 B0 B1* (0$_m$ *n n*) =
Complex-Matrix.trace (*A0* ∗ *B1* ∗ 0$_m$ *n n*) −
Complex-Matrix.trace (*A0* ∗ *B0* ∗ 0$_m$ *n n*) +
Complex-Matrix.trace (*A1* ∗ *B0* ∗ 0$_m$ *n n*) +
Complex-Matrix.trace (*A1* ∗ *B1* ∗ 0$_m$ *n n*)
by (*meson CHSH-expect-expand assms zero-carrier-mat*)
then show *?thesis*
using *assms*(*2*) *assms*(*3*) *assms*(*4*) **by** *force*

qed

lemma (**in** *cpx-sq-mat*) *CHSH-expect-sum*:
assumes *finite S*
and *A0* ∈ *fc-mats*
and *A1* ∈ *fc-mats*
and *B0* ∈ *fc-mats*
and *B1* ∈ *fc-mats*
and ⋀*i*. *i* ∈ *S* ⟹ *R i* ∈ *fc-mats*
shows *CHSH-expect A0 A1 B0 B1* (*sum-mat R S*) =
sum (λ*i*. *CHSH-expect A0 A1 B0 B1* (*R i*)) *S* **using** *assms*
proof (*induct rule*: *finite-induct*)
case *empty*
then show *?case* **using** *CHSH-expect-zero*
by (*metis dim-eq fc-mats-carrier sum.empty sum-mat-empty*)
next
case (*insert x F*)
have *CHSH-expect A0 A1 B0 B1* (*sum-mat R* (*insert x F*)) =
CHSH-expect A0 A1 B0 B1 (*R x* + (*sum-mat R F*))
using *insert sum-mat-insert*[*of R*]
by (*simp add*: *image-subsetI*)
also have ... = *CHSH-expect A0 A1 B0 B1* (*R x*) +
CHSH-expect A0 A1 B0 B1 (*sum-mat R F*)
proof (*rule CHSH-expect-add*)
have *fc*: *fc-mats* = *carrier-mat dimR dimR*
using *fc-mats-carrier dim-eq* **by** *simp*
show *A0* ∈ *carrier-mat dimR dimR A1* ∈ *carrier-mat dimR dimR*
B0 ∈ *carrier-mat dimR dimR B1* ∈ *carrier-mat dimR dimR*
R x ∈ *carrier-mat dimR dimR*
using *insert fc* **by** *auto*
show *sum-mat R F* ∈ *carrier-mat dimR dimR*
using *insert fc sum-mat-carrier dim-eq* **by** *blast*
qed
also have ... = *sum* (λ*i*. *CHSH-expect A0 A1 B0 B1* (*R i*)) (*insert x F*)
by (*simp add*: *assms insert*(*1*) *insert*(*2*) *insert*(*3*) *insert*(*8*))
finally show *?case* **.**
qed

lemma *CHSH-expect-smult*:
assumes *A0* ∈ *carrier-mat n n*
and *A1* ∈ *carrier-mat n n*
and *B0* ∈ *carrier-mat n n*
and *B1* ∈ *carrier-mat n n*
and *R0* ∈ *carrier-mat n n*
shows *CHSH-expect A0 A1 B0 B1* (*a* ·$_m$ *R0*) =
a ∗ *CHSH-expect A0 A1 B0 B1 R0*
proof −
note *chsh* = *CHSH-op-dim*[*OF assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*)]
show *?thesis* **using** *chsh*

```
    by (metis (no-types, lifting) CHSH-expect-def assms(5) mult-carrier-mat
        mult-smult-distrib trace-smult)
qed

lemma CHSH-expect-real:
  assumes 0 < n
  and CHSH-cond-hermit n A0 A1 B0 B1
  and R∈ carrier-mat n n
  and Complex-Matrix.positive R
  shows CHSH-expect A0 A1 B0 B1 R ∈ Reals
proof −
  define fc::complex Matrix.mat set where fc = carrier-mat n n
  interpret cpx-sq-mat n n fc
  proof
    show 0 < n using assms by simp
  qed (auto simp add: fc-def)
  have Complex-Matrix.trace (A0 ∗ B1 ∗ R) ∈ Reals
  proof (rule pos-hermitian-trace-reals)
    show A0 ∗ B1 ∈ carrier-mat n n using assms
      unfolding CHSH-cond-hermit-def CHSH-cond-def
      by (metis mult-carrier-mat)
    show hermitian (A0∗B1) using hermitian-commute assms
      unfolding CHSH-cond-hermit-def CHSH-cond-def
      by blast
  qed (auto simp add: assms)
  moreover have Complex-Matrix.trace (A0 ∗ B0 ∗ R) ∈ Reals
  proof (rule pos-hermitian-trace-reals)
    show A0 ∗ B0 ∈ carrier-mat n n using assms
      unfolding CHSH-cond-hermit-def CHSH-cond-def
      by (metis mult-carrier-mat)
    show hermitian (A0∗B0) using hermitian-commute assms
      unfolding CHSH-cond-hermit-def CHSH-cond-def
      by blast
  qed (auto simp add: assms)
  moreover have Complex-Matrix.trace (A1 ∗ B0 ∗ R) ∈ Reals
  proof (rule pos-hermitian-trace-reals)
    show A1 ∗ B0 ∈ carrier-mat n n using assms
      unfolding CHSH-cond-hermit-def CHSH-cond-def
      by (metis mult-carrier-mat)
    show hermitian (A1∗B0) using hermitian-commute assms
      unfolding CHSH-cond-hermit-def CHSH-cond-def
      by blast
  qed (auto simp add: assms)
  moreover have Complex-Matrix.trace (A1 ∗ B1 ∗ R) ∈ Reals
  proof (rule pos-hermitian-trace-reals)
    show A1 ∗ B1 ∈ carrier-mat n n using assms
      unfolding CHSH-cond-hermit-def CHSH-cond-def
      by (metis mult-carrier-mat)
    show hermitian (A1∗B1) using hermitian-commute assms
```

```
    unfolding CHSH-cond-hermit-def CHSH-cond-def
    by blast
  qed (auto simp add: assms)
  moreover have CHSH-expect A0 A1 B0 B1 R =
    Complex-Matrix.trace (A0 * B1 * R) -
    Complex-Matrix.trace (A0 * B0 * R) +
    Complex-Matrix.trace (A1 * B0 * R) +
    Complex-Matrix.trace (A1 * B1 * R)
    using CHSH-expect-expand assms
    unfolding CHSH-cond-hermit-def CHSH-cond-def by meson
  ultimately show ?thesis by simp
qed

lemma CHSH-op-square-L2-op-nrm-le:
  assumes CHSH-cond-hermit n A0 A1 B0 B1
  and 0 < n
  shows L2-op-nrm ((CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1)) ≤ 8
proof -
  have dima: commutator A0 A1 ∈ carrier-mat n n
   using assms commutator-dim unfolding CHSH-cond-hermit-def CHSH-cond-def

    by metis
  moreover have dimb: commutator B0 B1 ∈ carrier-mat n n
   using assms commutator-dim unfolding CHSH-cond-hermit-def CHSH-cond-def

    by metis
  ultimately have
    dim: (commutator A0 A1) * (commutator B0 B1) ∈ carrier-mat n n by simp
  have L2-op-nrm ((CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1)) =
   L2-op-nrm ((4::nat) ·_m (1_m n) - (commutator A0 A1) * (commutator B0 B1))

   using CHSH-op-square[of n] assms unfolding CHSH-cond-hermit-def by simp
  also have ... ≤ L2-op-nrm ((4::nat) ·_m (1_m n)) +
    L2-op-nrm ((commutator A0 A1) * (commutator B0 B1))
    by (rule L2-op-nrm-triangle', (auto simp add: assms dim))
  also have ... = 4 + L2-op-nrm ((commutator A0 A1) * (commutator B0 B1))
    using idty-smult-nat-L2-op-nrm[of n 4] assms by simp
  also have ... ≤ 4 + L2-op-nrm (commutator A0 A1) * L2-op-nrm (commutator
B0 B1)
  proof -
    have L2-op-nrm ((commutator A0 A1) * (commutator B0 B1)) ≤
      L2-op-nrm (commutator A0 A1) * L2-op-nrm (commutator B0 B1)
    proof (rule L2-op-nrm-mult-le)
      show commutator A0 A1 ∈ carrier-mat n n using assms commutator-dim
        unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
      show commutator B0 B1 ∈ carrier-mat n n using assms commutator-dim
        unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
    qed (simp add: assms)
    thus ?thesis by simp
```

```
  qed
  also have ... ≤ 4 + L2-op-nrm (commutator A0 A1) * 2
  proof −
    have L2-op-nrm (commutator B0 B1) ≤ 2
      using comm-L2-op-nrm-le[of B0 n] assms commutator-dim
      unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
    hence L2-op-nrm (commutator A0 A1) * L2-op-nrm (commutator B0 B1) ≤
      L2-op-nrm (commutator A0 A1) * 2
      using L2-op-nrm-geq-0 dima
      by (metis Groups.mult-ac(2) assms(2) linorder-not-less
          mult-le-cancel-right)
    thus ?thesis by simp
  qed
  also have ... ≤ 4 + 4
  proof −
    have L2-op-nrm (commutator A0 A1) ≤ 2
      using comm-L2-op-nrm-le[of A0 n] assms commutator-dim
      unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
    hence L2-op-nrm (commutator A0 A1) * 2 ≤ 2 * 2 by linarith
    thus ?thesis by simp
  qed
 finally show L2-op-nrm ((CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1))
≤ 8
    by simp
qed

lemma CHSH-op-square-spmax-le:
  assumes CHSH-cond-hermit n A0 A1 B0 B1
  and 0 < n
shows spmax ((CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1)) ≤ 8
proof −
  define Op where Op = CHSH-op A0 A1 B0 B1
  have spmax (Op * Op) =  L2-op-nrm (Op * Op)
  proof (rule hermitian-L2-op-nrm-spmax-eq[symmetric])
    show 0 < dim-row (Op * Op)
      using assms CHSH-op-dim[of A0 n n A1 B0 n B1]
      unfolding Op-def CHSH-cond-hermit-def CHSH-cond-def by simp
    show hermitian (Op * Op)
      using hermitian-square-hermitian[of Op] CHSH-op-hermitian[of A0] assms
      unfolding Op-def CHSH-cond-hermit-def CHSH-cond-def by simp
  qed
  also have ... ≤ 8 using CHSH-op-square-L2-op-nrm-le assms
    unfolding Op-def by simp
  finally show ?thesis unfolding Op-def .
qed

lemma CHSH-op-L2-op-nrm-le:
assumes CHSH-cond-hermit n A0 A1 B0 B1
  and 0 < n
```

shows *L2-op-nrm* (*CHSH-op A0 A1 B0 B1*) ≤ *2* ∗ *sqrt 2*
proof −
define *Op* **where** *Op* = *CHSH-op A0 A1 B0 B1*
have *L2-op-nrm Op* = *max-sgval Op*
using *L2-op-nrm-max-sgval-eq*[*of Op n*] *CHSH-op-dim*[*of A0 n n*] *assms*
unfolding *CHSH-cond-hermit-def CHSH-cond-def Op-def* **by** *simp*
also have ... = *sqrt* (*spmax* (*Op* ∗ *Op*))
using *CHSH-op-hermitian*[*of A0*] *assms*
unfolding *max-sgval-def hermitian-def CHSH-cond-hermit-def*
CHSH-cond-def Op-def
by *simp*
also have ... ≤ *sqrt 8*
using *assms CHSH-op-square-spmax-le*[*of n A0 A1 B0 B1*]
unfolding *Op-def*
by *simp*
also have ... = *2* ∗ *sqrt 2*
by (*metis mult-2-right numeral.simps*(*2*) *real-sqrt-four real-sqrt-mult*)
finally show *?thesis* **unfolding** *Op-def* **.**
qed

lemma (**in** *cpx-sq-mat*) *CHSH-cond-hermit-lhv-upper*:
assumes *CHSH-cond-hermit dimR A0 A1 B0 B1*
and *lhv M A0 B1 R U0 V1*
and *lhv M A0 B0 R U0 V0*
and *lhv M A1 B0 R U1 V0*
and *lhv M A1 B1 R U1 V1*
and *0* < *n*
shows |(*LINT w*|*M*. *qt-expect A0 U0 w* ∗ *qt-expect B1 V1 w*) −
(*LINT w*|*M*. *qt-expect A0 U0 w* ∗ *qt-expect B0 V0 w*) +
(*LINT w*|*M*. *qt-expect A1 U1 w* ∗ *qt-expect B0 V0 w*) +
(*LINT w*|*M*. *qt-expect A1 U1 w* ∗ *qt-expect B1 V1 w*)|
≤ *2*
proof −
have |(*LINT w*|*M*. *qt-expect A0 U0 w* ∗ *qt-expect B1 V1 w*) −
(*LINT w*|*M*. *qt-expect A0 U0 w* ∗ *qt-expect B0 V0 w*) +
(*LINT w*|*M*. *qt-expect A1 U1 w* ∗ *qt-expect B0 V0 w*) +
(*LINT w*|*M*. *qt-expect A1 U1 w* ∗ *qt-expect B1 V1 w*)| =
|(*LINT w*|*M*. *qt-expect A1 U1 w* ∗ *qt-expect B0 V0 w*) +
(*LINT w*|*M*. *qt-expect A0 U0 w* ∗ *qt-expect B1 V1 w*) +
(*LINT w*|*M*. *qt-expect A1 U1 w* ∗ *qt-expect B1 V1 w*)−
(*LINT w*|*M*. *qt-expect A0 U0 w* ∗ *qt-expect B0 V0 w*)| **by** *simp*
also have ... ≤ *2*
proof (*rule prob-space.chsh-expect*)
show *prob-space M* **using** *assms* **unfolding** *lhv-def* **by** *simp*
show *AE w in M*. |*qt-expect A0 U0 w*| ≤ *1* **unfolding** *qt-expect-def*
proof (*rule spectrum-abs-1-weighted-suml*)
show *lhv M A0 B1 R U0 V1* **using** *assms* **by** *simp*
show *hermitian A0* **using** *assms* **unfolding** *CHSH-cond-hermit-def* **by** *simp*
show *A0* ∈ *fc-mats* **using** *fc-mats-carrier dim-eq assms*

```
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  thus {Re x |x. x ∈ spectrum A0} ⊆ {- 1, 1}
    using assms CHSH-cond-hermit-unitary(1) unitary-hermitian-Re-spectrum
      ‹hermitian A0› fc-mats-carrier npos dim-eq
    by (metis (no-types, lifting))
  show {Re x |x. x ∈ spectrum A0} ≠ {}
    using ‹A0∈ fc-mats› fc-mats-carrier npos dim-eq
      ‹hermitian A0› spectrum-ne by fastforce
qed
show AE w in M. |qt-expect A1 U1 w| ≤ 1 unfolding qt-expect-def
proof (rule spectrum-abs-1-weighted-suml)
  show lhv M A1 B1 R U1 V1 using assms by simp
 show hermitian A1 using assms unfolding CHSH-cond-hermit-def by simp
  show A1 ∈ fc-mats using fc-mats-carrier dim-eq assms
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  thus {Re x |x. x ∈ spectrum A1} ⊆ {- 1, 1}
    using assms CHSH-cond-hermit-unitary(2) unitary-hermitian-Re-spectrum
      ‹hermitian A1› fc-mats-carrier npos dim-eq
    by (metis (no-types, lifting))
  show {Re x |x. x ∈ spectrum A1} ≠ {}
    using ‹A1∈ fc-mats› fc-mats-carrier npos dim-eq
      ‹hermitian A1› spectrum-ne by fastforce
qed
show AE w in M. |qt-expect B0 V0 w| ≤ 1 unfolding qt-expect-def
proof (rule spectrum-abs-1-weighted-sumr)
  show lhv M A1 B0 R U1 V0 using assms by simp
 show hermitian B0 using assms unfolding CHSH-cond-hermit-def by simp
  show B0 ∈ fc-mats using fc-mats-carrier dim-eq assms
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  thus {Re x |x. x ∈ spectrum B0} ⊆ {- 1, 1}
    using assms CHSH-cond-hermit-unitary(3) unitary-hermitian-Re-spectrum
      ‹hermitian B0› fc-mats-carrier npos dim-eq
    by (metis (no-types, lifting))
  show {Re x |x. x ∈ spectrum B0} ≠ {}
    using ‹B0∈ fc-mats› fc-mats-carrier npos dim-eq
      ‹hermitian B0› spectrum-ne by fastforce
qed
show AE w in M. |qt-expect B1 V1 w| ≤ 1 unfolding qt-expect-def
proof (rule spectrum-abs-1-weighted-sumr)
  show lhv M A1 B1 R U1 V1 using assms by simp
 show hermitian B1 using assms unfolding CHSH-cond-hermit-def by simp
  show B1 ∈ fc-mats using fc-mats-carrier dim-eq assms
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  thus {Re x |x. x ∈ spectrum B1} ⊆ {- 1, 1}
    using assms CHSH-cond-hermit-unitary(4) unitary-hermitian-Re-spectrum
      ‹hermitian B1› fc-mats-carrier npos dim-eq
    by (metis (no-types, lifting))
  show {Re x |x. x ∈ spectrum B1} ≠ {}
    using ‹B1∈ fc-mats› fc-mats-carrier npos dim-eq
```

```
      ‹hermitian B1› spectrum-ne by fastforce
    qed
    show integrable M (λw. qt-expect A0 U0 w * qt-expect B1 V1 w)
      using spectr-sum-integrable[of M] assms by simp
    show integrable M (λw. qt-expect A1 U1 w * qt-expect B1 V1 w)
      using spectr-sum-integrable[of M] assms by simp
    show integrable M (λw. qt-expect A1 U1 w * qt-expect B0 V0 w)
      using spectr-sum-integrable[of M] assms by simp
    show integrable M (λw. qt-expect A0 U0 w * qt-expect B0 V0 w)
      using spectr-sum-integrable[of M] assms by simp
  qed
  finally show ?thesis .
qed

lemma (in cpx-sq-mat) CHSH-expect-lhv-lint-eq:
  assumes R ∈ fc-mats
  and Complex-Matrix.positive R
  and CHSH-cond-hermit dimR A0 A1 B0 B1
  and lhv M A0 B1 R U0 V1
  and lhv M A0 B0 R U0 V0
  and lhv M A1 B0 R U1 V0
  and lhv M A1 B1 R U1 V1
shows (LINT w|M. qt-expect A0 U0 w * qt-expect B1 V1 w) −
       (LINT w|M. qt-expect A0 U0 w * qt-expect B0 V0 w) +
       (LINT w|M. qt-expect A1 U1 w * qt-expect B0 V0 w) +
       (LINT w|M. qt-expect A1 U1 w * qt-expect B1 V1 w) =
       CHSH-expect A0 A1 B0 B1 R (is ?L = ?R)
proof −
  have A0 ∈ fc-mats using assms fc-mats-carrier dim-eq
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  have B0 ∈ fc-mats using assms fc-mats-carrier dim-eq
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  have A1 ∈ fc-mats using assms fc-mats-carrier dim-eq
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  have B1 ∈ fc-mats using assms fc-mats-carrier dim-eq
    unfolding CHSH-cond-hermit-def CHSH-cond-def by simp
  have LINT w|M. qt-expect A0 U0 w * qt-expect B1 V1 w =
    Re (Complex-Matrix.trace (A0 * B1 * R))
  proof (rule sum-qt-expect)
    show hermitian A0 hermitian B1
      using assms unfolding CHSH-cond-hermit-def by auto
  qed (auto simp add: ‹A0 ∈ fc-mats› ‹B1 ∈ fc-mats› assms)
  moreover have LINT w|M. qt-expect A0 U0 w * qt-expect B0 V0 w =
    Re (Complex-Matrix.trace (A0 * B0 * R))
  proof (rule sum-qt-expect)
    show hermitian A0 hermitian B0
      using assms unfolding CHSH-cond-hermit-def by auto
  qed (auto simp add: ‹A0 ∈ fc-mats› ‹B0 ∈ fc-mats› assms)
  moreover have LINT w|M. qt-expect A1 U1 w * qt-expect B0 V0 w =
```

Re (*Complex-Matrix.trace* (*A1* ∗ *B0* ∗ *R*))
proof (*rule sum-qt-expect*)
show *hermitian A1 hermitian B0*
using *assms* **unfolding** *CHSH-cond-hermit-def* **by** *auto*
qed (*auto simp add*: ‹*A1* ∈ *fc-mats*› ‹*B0* ∈ *fc-mats*› *assms*)
moreover have *LINT w|M. qt-expect A1 U1 w* ∗ *qt-expect B1 V1 w* =
Re (*Complex-Matrix.trace* (*A1* ∗ *B1* ∗ *R*))
proof (*rule sum-qt-expect*)
show *hermitian A1 hermitian B1*
using *assms* **unfolding** *CHSH-cond-hermit-def* **by** *auto*
qed (*auto simp add*: ‹*A1* ∈ *fc-mats*› ‹*B1* ∈ *fc-mats*› *assms*)
ultimately have *?L* =
Re (*Complex-Matrix.trace* (*A0* ∗ *B1* ∗ *R*)) −
Re (*Complex-Matrix.trace* (*A0* ∗ *B0* ∗ *R*)) +
Re (*Complex-Matrix.trace* (*A1* ∗ *B0* ∗ *R*)) +
Re (*Complex-Matrix.trace* (*A1* ∗ *B1* ∗ *R*)) **by** *simp*
also have ... = *Re* (*Complex-Matrix.trace* (*A0* ∗ *B1* ∗ *R*) −
Complex-Matrix.trace (*A0* ∗ *B0* ∗ *R*) +
Complex-Matrix.trace (*A1* ∗ *B0* ∗ *R*) +
Complex-Matrix.trace (*A1* ∗ *B1* ∗ *R*)) **by** *simp*
also have ... = *Re* (*CHSH-expect A0 A1 B0 B1 R*)
using *CHSH-expect-expand assms fc-mats-carrier dim-eq*
‹*A0* ∈ *fc-mats*› ‹*B0* ∈ *fc-mats*› ‹*A1* ∈ *fc-mats*› ‹*B1* ∈ *fc-mats*›
by *metis*
also have ... = *CHSH-expect A0 A1 B0 B1 R*
using *CHSH-expect-real assms fc-mats-carrier dim-eq npos*
by *simp*
finally show *?thesis* **.**
qed

9.3 CHSH inequality for separable density matrices

definition *CHSH-cond-local* **where**
CHSH-cond-local n m A0 A1 B0 B1 ≡
A0 ∈ *carrier-mat n n* ∧ *A1* ∈ *carrier-mat n n* ∧
B0 ∈ *carrier-mat m m* ∧ *B1* ∈ *carrier-mat m m* ∧
hermitian A0 ∧ *hermitian A1* ∧ *hermitian B0* ∧ *hermitian B1* ∧
A0 ∗ *A0* = 1_m *n* ∧ *A1* ∗ *A1* = 1_m *n* ∧ *B0* ∗ *B0* = 1_m *m* ∧ *B1* ∗ *B1* = 1_m *m*

lemma *CHSH-cond-local-imp-cond-hermit*:
assumes *CHSH-cond-local n m A0 A1 B0 B1*
and *0* < *n*
and *0* < *m*
shows *CHSH-cond-hermit* (*n*∗*m*) (*A0* ⊗ 1_m *m*) (*A1* ⊗ 1_m *m*)
(1_m *n* ⊗ *B0*) (1_m *n* ⊗ *B1*)
unfolding *CHSH-cond-hermit-def CHSH-cond-def*
proof (*intro conjI*)
show *A0* ⊗ 1_m *m* ∈ *carrier-mat* (*n* ∗ *m*) (*n* ∗ *m*)
A1 ⊗ 1_m *m* ∈ *carrier-mat* (*n* ∗ *m*) (*n* ∗ *m*)

$1_m\ n \otimes B0 \in$ *carrier-mat* $(n * m)\ (n * m)$
$1_m\ n \otimes B1 \in$ *carrier-mat* $(n * m)\ (n * m)$
using *assms* **unfolding** *CHSH-cond-local-def* **by** *auto*
show *hermitian* $(A0 \otimes 1_m\ m)$ *hermitian* $(A1 \otimes 1_m\ m)$
hermitian $(1_m\ n \otimes B0)$ *hermitian* $(1_m\ n \otimes B1)$
using *assms tensor-mat-hermitian* **unfolding** *CHSH-cond-local-def*
by (*metis hermitian-one one-carrier-mat*)+
show $(A0 \otimes 1_m\ m) * (A0 \otimes 1_m\ m) = 1_m\ (n * m)$
using *assms tensor-mat-square-idty idty-square*
unfolding *CHSH-cond-local-def* **by** *auto*
show $(A1 \otimes 1_m\ m) * (A1 \otimes 1_m\ m) = 1_m\ (n * m)$
using *assms tensor-mat-square-idty idty-square*
unfolding *CHSH-cond-local-def* **by** *auto*
show $(1_m\ n \otimes B0) * (1_m\ n \otimes B0) = 1_m\ (n * m)$
using *assms tensor-mat-square-idty idty-square*
unfolding *CHSH-cond-local-def* **by** *auto*
show $(1_m\ n \otimes B1) * (1_m\ n \otimes B1) = 1_m\ (n * m)$
using *assms tensor-mat-square-idty idty-square*
unfolding *CHSH-cond-local-def* **by** *auto*
show $(A0 \otimes 1_m\ m) * (1_m\ n \otimes B1) = (1_m\ n \otimes B1) * (A0 \otimes 1_m\ m)$
using *tensor-mat-commute assms* **unfolding** *CHSH-cond-local-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat*)
show $(A0 \otimes 1_m\ m) * (1_m\ n \otimes B0) = (1_m\ n \otimes B0) * (A0 \otimes 1_m\ m)$
using *tensor-mat-commute assms* **unfolding** *CHSH-cond-local-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat*)
show $(A1 \otimes 1_m\ m) * (1_m\ n \otimes B0) = (1_m\ n \otimes B0) * (A1 \otimes 1_m\ m)$
using *tensor-mat-commute assms* **unfolding** *CHSH-cond-local-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat*)
show $(A1 \otimes 1_m\ m) * (1_m\ n \otimes B1) = (1_m\ n \otimes B1) * (A1 \otimes 1_m\ m)$
using *tensor-mat-commute assms* **unfolding** *CHSH-cond-local-def*
by (*smt* (*verit*) *assoc-mult-mat mult-carrier-mat*)
qed

lemma *limit-CHSH-cond*:
shows *CHSH-cond-hermit 4 Z-I X-I I-ZmX I-XpZ*
proof −
have *CHSH-cond-hermit* $(2 * 2)$ *Z-I X-I I-ZmX I-XpZ*
unfolding *Z-I-def X-I-def I-ZmX-def I-XpZ-def*
proof (*rule CHSH-cond-local-imp-cond-hermit*)
show *CHSH-cond-local 2 2 Z X ZmX XpZ* **unfolding** *CHSH-cond-local-def*
by (*simp add*: *X-carrier X-hermitian XpZ-carrier XpZ-hermitian XpZ-inv*
Z-carrier Z-hermitian ZmX-carrier ZmX-hermitian ZmX-inv)
qed *auto*
thus *?thesis* **by** *simp*
qed

lemma *CHSH-expect-separable-expand*:
assumes *separately-decomposes R n nA nB K F S*
and $A0 \in$ *carrier-mat nA nA*

and *A1* ∈ *carrier-mat nA nA*
and *B0* ∈ *carrier-mat nB nB*
and *B1* ∈ *carrier-mat nB nB*
shows *CHSH-expect* (*A0*⨂ 1_m *nB*) (*A1*⨂ 1_m *nB*) (1_m *nA*⨂ *B0*) (1_m *nA*⨂ *B1*) *R* =
sum (λ*a*. *K a* ∗ *CHSH-expect* (*A0*⨂ 1_m *nB*) (*A1*⨂ 1_m *nB*) (1_m *nA*⨂ *B0*) (1_m *nA*⨂ *B1*)
((*F a*) ⨂ (*S a*))) {..< *n*}
proof −
define *fc*::*complex Matrix.mat set*
where *fc* = *carrier-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*)
interpret *cpx-sq-mat nA* ∗ *nB nA* ∗ *nB fc*
proof
show *fc* = *carrier-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*) **using** *fc-def* **by** *simp*
show *0* < *nA* ∗ *nB* **using** *assms separately-decomposes-carrier-pos*
by *simp*
qed *simp*
have *dec*: ⋀*a*. *a* ∈ {..<*n*} ⟹ (*F a* ⨂ *S a*) ∈ *fc*
proof −
fix *a*
assume *a*∈ {..< *n*}
hence *F a* ∈ *carrier-mat nA nA S a*∈ *carrier-mat nB nB*
using *assms* **unfolding** *separately-decomposes-def* **by** *auto*
thus (*F a* ⨂ *S a*) ∈ *fc*
using *tensor-mat-carrier assms* **unfolding** *fc-def* **by** *auto*
qed
hence *dec′*: ⋀*a*. *a* ∈ {..<*n*} ⟹ *K a* ·$_m$ (*F a* ⨂ *S a*) ∈ *fc*
by (*simp add*: *smult-mem*)
have *car*: *A0* ⨂ 1_m *nB* ∈ *fc A1* ⨂ 1_m *nB* ∈ *fc*
1_m *nA* ⨂ *B0* ∈ *fc* 1_m *nA* ⨂ *B1* ∈ *fc*
using *assms tensor-mat-carrier* **unfolding** *fc-def* **by** *auto*
have *CHSH-expect* (*A0*⨂ 1_m *nB*) (*A1*⨂ 1_m *nB*) (1_m *nA*⨂ *B0*) (1_m *nA*⨂ *B1*) *R* =
CHSH-expect (*A0*⨂ 1_m *nB*) (*A1*⨂ 1_m *nB*) (1_m *nA*⨂ *B0*) (1_m *nA*⨂ *B1*)
(*sum-mat* (λ*a*. *K a* ·$_m$ ((*F a*) ⨂ (*S a*))) {..< *n*})
using *assms* **unfolding** *separately-decomposes-def* **by** *simp*
also have ... =
sum (λ*a*. *CHSH-expect* (*A0*⨂ 1_m *nB*) (*A1*⨂ 1_m *nB*) (1_m *nA*⨂ *B0*) (1_m *nA*⨂ *B1*)
(*K a* ·$_m$ ((*F a*) ⨂ (*S a*)))) {..< *n*}
by (*rule CHSH-expect-sum*, (*auto simp add*: *dec′ car*))
also have ... =
sum (λ*a*. *K a* ∗ *CHSH-expect* (*A0*⨂ 1_m *nB*) (*A1*⨂ 1_m *nB*) (1_m *nA*⨂ *B0*) (1_m *nA*⨂ *B1*)
((*F a*) ⨂ (*S a*))) {..< *n*}
proof (*rule sum.cong*)
fix *x*
assume *x*∈ {..< *n*}
thus *CHSH-expect* (*A0* ⨂ 1_m *nB*) (*A1* ⨂ 1_m *nB*) (1_m *nA* ⨂ *B0*) (1_m *nA*

$\bigotimes$ *B1*)
(*K x* $\cdot_m$ (*F x* $\bigotimes$ *S x*)) =
K x ∗ *CHSH-expect* (*A0* $\bigotimes$ 1_m *nB*) (*A1* $\bigotimes$ 1_m *nB*) (1_m *nA* $\bigotimes$ *B0*)
(1_m *nA* $\bigotimes$ *B1*) (*F x* $\bigotimes$ *S x*)
using *car dec CHSH-expect-smult fc-mats-carrier* **by** *blast*
qed *simp*
finally show *?thesis* **.**
qed

lemma *CHSH-expect-tensor-leq*:
assumes *CHSH-cond-local nA nB A0 A1 B0 B1*
and *RA* ∈ *carrier-mat nA nA*
and *density-operator RA*
and *RB* ∈ *carrier-mat nB nB*
and *density-operator RB*
and *0* < *nA*
and *0* < *nB*
shows |*CHSH-expect* (*A0*$\bigotimes$ 1_m *nB*) (*A1*$\bigotimes$ 1_m *nB*) (1_m *nA*$\bigotimes$ *B0*) (1_m *nA*$\bigotimes$ *B1*)
(*RA*$\bigotimes$ *RB*)| ≤*2*
proof −
have *CHSH-expect* (*A0*$\bigotimes$ 1_m *nB*) (*A1*$\bigotimes$ 1_m *nB*) (1_m *nA*$\bigotimes$ *B0*) (1_m *nA*$\bigotimes$ *B1*)
(*RA*$\bigotimes$ *RB*) =
Complex-Matrix.trace ((*A0* $\bigotimes$ 1_m *nB*) ∗ (1_m *nA* $\bigotimes$ *B1*) ∗ (*RA*$\bigotimes$ *RB*)) −
Complex-Matrix.trace ((*A0* $\bigotimes$ 1_m *nB*) ∗ (1_m *nA* $\bigotimes$ *B0*) ∗ (*RA*$\bigotimes$ *RB*)) +
Complex-Matrix.trace ((*A1* $\bigotimes$ 1_m *nB*) ∗ (1_m *nA* $\bigotimes$ *B0*) ∗ (*RA*$\bigotimes$ *RB*)) +
Complex-Matrix.trace ((*A1* $\bigotimes$ 1_m *nB*) ∗ (1_m *nA* $\bigotimes$ *B1*) ∗ (*RA*$\bigotimes$ *RB*))
proof (*rule CHSH-expect-expand*)
show *A0* $\bigotimes$ 1_m *nB* ∈ *carrier-mat* (*nA*∗*nB*) (*nA*∗*nB*)
using *assms* **unfolding** *CHSH-cond-local-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *index-mult-mat*(*2*)
index-mult-mat(*3*) *tensor-mat-carrier*)
show *A1* $\bigotimes$ 1_m *nB* ∈ *carrier-mat* (*nA*∗*nB*) (*nA*∗*nB*)
using *assms* **unfolding** *CHSH-cond-local-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *index-mult-mat*(*2*)
index-mult-mat(*3*) *tensor-mat-carrier*)
show 1_m *nA* $\bigotimes$ *B0* ∈ *carrier-mat* (*nA* ∗ *nB*) (*nA*∗*nB*)
using *assms* **unfolding** *CHSH-cond-local-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *index-mult-mat*(*2*)
index-mult-mat(*3*) *tensor-mat-carrier*)
show 1_m *nA* $\bigotimes$ *B1* ∈ *carrier-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*)
using *assms* **unfolding** *CHSH-cond-local-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *index-mult-mat*(*2*)
index-mult-mat(*3*) *tensor-mat-carrier*)
show (*RA*$\bigotimes$ *RB*) ∈ *carrier-mat* (*nA* ∗ *nB*) (*nA* ∗ *nB*)
using *tensor-mat-carrier assms* **by** *blast*
qed
also have ... =
Complex-Matrix.trace ((*A0* $\bigotimes$ *B1*) ∗ (*RA*$\bigotimes$ *RB*)) −
Complex-Matrix.trace ((*A0* $\bigotimes$ *B0*) ∗ (*RA*$\bigotimes$ *RB*)) +

Complex-Matrix.trace ((*A1* ⨂ *B0*) ∗ (*RA*⨂ *RB*)) +
Complex-Matrix.trace ((*A1* ⨂ *B1*) ∗ (*RA*⨂ *RB*))
using *assms tensor-mat-mult-id* **unfolding** *CHSH-cond-local-def* **by** *presburger*
also have ... =
Complex-Matrix.trace (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*) −
Complex-Matrix.trace (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*) +
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*) +
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*)
proof −
have *Complex-Matrix.trace* ((*A0* ⨂ *B1*) ∗ (*RA*⨂ *RB*)) =
Complex-Matrix.trace (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*)
using *tensor-mat-trace-mult-distr assms* **unfolding** *CHSH-cond-local-def* **by** *auto*
moreover have *Complex-Matrix.trace* ((*A0* ⨂ *B0*) ∗ (*RA*⨂ *RB*)) =
Complex-Matrix.trace (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*)
using *tensor-mat-trace-mult-distr assms* **unfolding** *CHSH-cond-local-def* **by** *auto*
moreover have *Complex-Matrix.trace* ((*A1* ⨂ *B0*) ∗ (*RA*⨂ *RB*)) =
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*)
using *tensor-mat-trace-mult-distr assms* **unfolding** *CHSH-cond-local-def* **by** *auto*
moreover have *Complex-Matrix.trace* ((*A1* ⨂ *B1*) ∗ (*RA*⨂ *RB*)) =
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*)
using *tensor-mat-trace-mult-distr assms* **unfolding** *CHSH-cond-local-def* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
finally have *exp*: *CHSH-expect* (*A0*⨂ 1_m *nB*) (*A1*⨂ 1_m *nB*) (1_m *nA*⨂ *B0*)
(1_m *nA*⨂ *B1*) (*RA*⨂ *RB*) =
Complex-Matrix.trace (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*) −
Complex-Matrix.trace (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*) +
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*) +
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*) **.**
have |*Complex-Matrix.trace* (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*) −
Complex-Matrix.trace (*A0* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*) +
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B0* ∗ *RB*) +
Complex-Matrix.trace (*A1* ∗ *RA*) ∗ *Complex-Matrix.trace* (*B1* ∗ *RB*)| ≤ *2*
proof (*rule chsh-complex*)
show *Complex-Matrix.trace* (*A0* ∗ *RA*) ∈ ℝ
using *assms* **unfolding** *CHSH-cond-local-def*
by (*simp add*: *density-operator-def pos-hermitian-trace-reals*)
show |*Complex-Matrix.trace* (*A0*∗*RA*) ∗ *Complex-Matrix.trace* (*B1*∗*RB*)| ≤ *1*
proof (*rule cpx-abs-mult-le-1*)
show |*Complex-Matrix.trace* (*A0* ∗ *RA*)| ≤ *1*
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def* **by** *auto*
show |*Complex-Matrix.trace* (*B1* ∗ *RB*)| ≤ *1*
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def* **by** *auto*

qed
show *Complex-Matrix.trace* $(A1 * RA) \in \mathbb{R}$
using *assms* **unfolding** *CHSH-cond-local-def*
by (*simp add*: *density-operator-def pos-hermitian-trace-reals*)
show $|$*Complex-Matrix.trace* $(A1{*}RA) *$ *Complex-Matrix.trace* $(B1{*}RB)| \leq 1$
proof (*rule cpx-abs-mult-le-1*)
show $|$*Complex-Matrix.trace* $(A1 * RA)| \leq 1$
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def*
by *auto*
show $|$*Complex-Matrix.trace* $(B1 * RB)| \leq 1$
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def*
by *auto*
qed
show *Complex-Matrix.trace* $(B0 * RB) \in \mathbb{R}$
using *assms* **unfolding** *CHSH-cond-local-def*
by (*simp add*: *density-operator-def pos-hermitian-trace-reals*)
show $|$*Complex-Matrix.trace* $(A0{*}RA) *$ *Complex-Matrix.trace* $(B0{*}RB)| \leq 1$
proof (*rule cpx-abs-mult-le-1*)
show $|$*Complex-Matrix.trace* $(A0 * RA)| \leq 1$
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def*
by *auto*
show $|$*Complex-Matrix.trace* $(B0 * RB)| \leq 1$
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def*
by *auto*
qed
show *Complex-Matrix.trace* $(B1 * RB) \in \mathbb{R}$
using *assms* **unfolding** *CHSH-cond-local-def*
by (*simp add*: *density-operator-def pos-hermitian-trace-reals*)
show $|$*Complex-Matrix.trace* $(A1{*}RA) *$ *Complex-Matrix.trace* $(B0{*}RB)| \leq 1$
proof (*rule cpx-abs-mult-le-1*)
show $|$*Complex-Matrix.trace* $(A1 * RA)| \leq 1$
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def*
by *auto*
show $|$*Complex-Matrix.trace* $(B0 * RB)| \leq 1$
using *assms hermitian-mult-density-trace* **unfolding** *CHSH-cond-local-def*
by *auto*
qed
qed
thus *?thesis* **using** *exp* **by** *simp*
qed

9.4 CHSH inequality for commuting observables

lemma *CHSH-op-square-commute-L2-op-nrm-eq*:
assumes *CHSH-cond-hermit n A0 A1 B0 B1*
and $0 < n$
and *commutator A0 A1* $= 0_m$ *n n* $\vee$ *commutator B0 B1* $= 0_m$ *n n*
shows *L2-op-nrm* ((*CHSH-op A0 A1 B0 B1*) $*$ (*CHSH-op A0 A1 B0 B1*)) $= 4$
proof −

```
  have dima: commutator A0 A1 ∈ carrier-mat n n
    using assms commutator-dim
    unfolding CHSH-cond-hermit-def CHSH-cond-def by metis
  moreover have dimb: commutator B0 B1 ∈ carrier-mat n n
    using assms commutator-dim
    unfolding CHSH-cond-hermit-def CHSH-cond-def by metis
  ultimately have
    dim: (commutator A0 A1) * (commutator B0 B1) ∈ carrier-mat n n by simp
  have L2-op-nrm ((CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1)) =
   L2-op-nrm ((4::nat) ·m (1m n) − (commutator A0 A1) * (commutator B0 B1))

    using CHSH-op-square[of n] assms unfolding CHSH-cond-hermit-def by simp
  also have ... = L2-op-nrm ((4::nat) ·m (1m n))
  proof (cases commutator A0 A1 = 0m n n)
    case True
    hence (commutator A0 A1) * (commutator B0 B1) = 0m n n
      using dima dimb by simp
    hence (4::nat) ·m (1m n) − (commutator A0 A1) * (commutator B0 B1) =
      (4::nat) ·m (1m n)
      using right-minus-zero-mat
      by (metis index-one-mat(2) index-one-mat(3) index-smult-mat(2)
          index-smult-mat(3))
    then show ?thesis by simp
  next
    case False
    hence commutator B0 B1 = 0m n n using assms by simp
     hence (commutator A0 A1) * (commutator B0 B1) = 0m n n
      using dima dimb by simp
    hence (4::nat) ·m (1m n) − (commutator A0 A1) * (commutator B0 B1) =
      (4::nat) ·m (1m n)
      using right-minus-zero-mat
      by (metis index-one-mat(2) index-one-mat(3) index-smult-mat(2)
          index-smult-mat(3))
    then show ?thesis by simp
  qed
  also have ... = 4 using idty-smult-nat-L2-op-nrm[of n 4] assms by simp
  finally show ?thesis .
qed

lemma CHSH-op-square-commute-spmax-eq:
  assumes CHSH-cond-hermit n A0 A1 B0 B1
  and 0 < n
  and commutator A0 A1 = 0m n n ∨ commutator B0 B1 = 0m n n
shows spmax ((CHSH-op A0 A1 B0 B1) * (CHSH-op A0 A1 B0 B1)) =4
proof −
  define Op where Op = CHSH-op A0 A1 B0 B1
  have spmax (Op * Op) =  L2-op-nrm (Op * Op)
  proof (rule hermitian-L2-op-nrm-spmax-eq[symmetric])
    show 0 < dim-row (Op * Op)
```

```
    using assms CHSH-op-dim[of A0 n n A1 B0 n B1]
    unfolding Op-def CHSH-cond-hermit-def CHSH-cond-def by simp
  show hermitian (Op ∗ Op)
    using hermitian-square-hermitian[of Op] CHSH-op-hermitian[of A0] assms
    unfolding Op-def CHSH-cond-hermit-def CHSH-cond-def by simp
 qed
 also have ... =4 using CHSH-op-square-commute-L2-op-nrm-eq assms
  unfolding Op-def by simp
 finally show ?thesis unfolding Op-def .
qed

lemma CHSH-op-commute-L2-op-nrm-eq:
assumes CHSH-cond-hermit n A0 A1 B0 B1
 and 0 < n
 and commutator A0 A1 = 0_m n n ∨ commutator B0 B1 = 0_m n n
shows L2-op-nrm (CHSH-op A0 A1 B0 B1) = 2
proof −
 define Op where Op = CHSH-op A0 A1 B0 B1
 have L2-op-nrm Op = max-sgval Op
  using L2-op-nrm-max-sgval-eq[of Op n] CHSH-op-dim[of A0 n n] assms
  unfolding CHSH-cond-hermit-def CHSH-cond-def Op-def by simp
 also have ... = sqrt (spmax (Op ∗ Op))
  using CHSH-op-hermitian[of A0] assms
  unfolding max-sgval-def hermitian-def CHSH-cond-hermit-def
    CHSH-cond-def Op-def
  by simp
 also have ... = 2
  using assms CHSH-op-square-commute-spmax-eq[of n A0 A1 B0 B1]
  unfolding Op-def
  by simp
 finally show ?thesis unfolding Op-def .
qed
```

9.5 Result summary on the CHSH inequalities

Under the local hidden variable hypothesis, this value is bounded by 2.

```
lemma CHSH-expect-lhv-leq:
 assumes R ∈ carrier-mat n n
 and 0 < n
 and Complex-Matrix.positive R
 and CHSH-cond-hermit n A0 A1 B0 B1
 and cpx-sq-mat.lhv n n M A0 B1 R U0 V1
 and cpx-sq-mat.lhv n n M A0 B0 R U0 V0
 and cpx-sq-mat.lhv n n M A1 B0 R U1 V0
 and cpx-sq-mat.lhv n n M A1 B1 R U1 V1
shows |CHSH-expect A0 A1 B0 B1 R| ≤ 2
proof −
 define fc::complex Matrix.mat set
  where fc = carrier-mat n n
```

```
  interpret cpx-sq-mat n n fc
  proof
    show fc = carrier-mat n n using fc-def by simp
    show 0 < n using assms
      by simp
  qed simp
  have R∈ fc using assms fc-def by simp
  have |CHSH-expect A0 A1 B0 B1 R| ≤ complex-of-real 2
  proof (rule cpx-real-abs-leq)
    have R∈ carrier-mat n n using assms by simp
    show |(LINT w|M. qt-expect A0 U0 w * qt-expect B1 V1 w) −
          (LINT w|M. qt-expect A0 U0 w * qt-expect B0 V0 w) +
          (LINT w|M. qt-expect A1 U1 w * qt-expect B0 V0 w) +
          (LINT w|M. qt-expect A1 U1 w * qt-expect B1 V1 w)| ≤ 2
      using CHSH-cond-hermit-lhv-upper assms by blast
    show CHSH-expect A0 A1 B0 B1 R =
      (LINT w|M. qt-expect A0 U0 w * qt-expect B1 V1 w) −
          (LINT w|M. qt-expect A0 U0 w * qt-expect B0 V0 w) +
          (LINT w|M. qt-expect A1 U1 w * qt-expect B0 V0 w) +
          (LINT w|M. qt-expect A1 U1 w * qt-expect B1 V1 w)
      using CHSH-expect-lhv-lint-eq[OF ‹R∈ fc› assms(3) assms(4)] assms
      by fastforce
    show CHSH-expect A0 A1 B0 B1 R ∈ ℝ
      using CHSH-expect-real[OF assms(2) assms(4) assms(1) assms(3)]
      by simp
  qed
  thus ?thesis by simp
qed
```

When the considered density operator is separable, this value is still bounded by 2.

```
lemma CHSH-expect-separable-leq:
  assumes CHSH-cond-local nA nB A0 A1 B0 B1
  and separable-density nA nB R
  and A0 ∈ carrier-mat nA nA
  and A1 ∈ carrier-mat nA nA
  and B0 ∈ carrier-mat nB nB
  and B1 ∈ carrier-mat nB nB
shows |CHSH-expect (A0⊗ 1_m nB) (A1⊗ 1_m nB) (1_m nA⊗ B0) (1_m nA⊗ B1)
R|
  ≤ 2
proof −
  have ∃ n K F S. separately-decomposes R n nA nB K F S
    using assms unfolding separable-density-def by simp
  from this obtain n K F S where
    separately-decomposes R n nA nB K F S by auto
  note props = this
  define fc::complex Matrix.mat set
    where fc = carrier-mat (nA * nB) (nA * nB)
```

interpret *cpx-sq-mat nA * nB nA * nB fc*
proof
show *fc = carrier-mat (nA * nB) (nA * nB)* **using** *fc-def* **by** *simp*
show *0 < nA * nB* **using** *assms props separately-decomposes-carrier-pos*
by *simp*
qed *simp*
have *dec*: $\bigwedge a.\ a \in \{..<n\} \Longrightarrow (F\ a \bigotimes S\ a) \in fc$
proof −
fix *a*
assume *a*∈ {..< *n*}
hence *F a ∈ carrier-mat nA nA S a*∈ *carrier-mat nB nB*
using *props* **unfolding** *separately-decomposes-def* **by** *auto*
thus $(F\ a \bigotimes S\ a) \in fc$
using *tensor-mat-carrier assms* **unfolding** *fc-def* **by** *auto*
qed
hence *dec′*: $\bigwedge a.\ a \in \{..<n\} \Longrightarrow K\ a \cdot_m (F\ a \bigotimes S\ a) \in fc$
by (*simp add*: *smult-mem*)
have *car*: $A0 \bigotimes 1_m\ nB \in fc\ A1 \bigotimes 1_m\ nB \in fc$
$1_m\ nA \bigotimes B0 \in fc\ 1_m\ nA \bigotimes B1 \in fc$
using *assms tensor-mat-carrier* **unfolding** *fc-def* **by** *auto*
have *CHSH-expect* $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)$ *R* =
CHSH-expect $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)$
(*sum-mat* $(\lambda a.\ K\ a \cdot_m ((F\ a) \bigotimes (S\ a)))\ \{..< n\}$)
using *props* **unfolding** *separately-decomposes-def* **by** *simp*
also have ... =
sum $(\lambda a.$ *CHSH-expect* $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)$
$(K\ a \cdot_m ((F\ a) \bigotimes (S\ a))))\ \{..< n\}$
by (*rule CHSH-expect-sum*, (*auto simp add*: *dec′ car*))
also have ... =
sum $(\lambda a.\ K\ a *$ *CHSH-expect* $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)$
$(1_m\ nA \bigotimes B1)$
$((F\ a) \bigotimes (S\ a)))\ \{..< n\}$
proof (*rule sum.cong*)
fix *x*
assume *x*∈ {..< *n*}
thus *CHSH-expect* $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)$
$(K\ x \cdot_m (F\ x \bigotimes S\ x)) =$
$K\ x *$ *CHSH-expect* $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)$
$(1_m\ nA \bigotimes B1)\ (F\ x \bigotimes S\ x)$
using *car dec CHSH-expect-smult fc-mats-carrier* **by** *blast*
qed *simp*
finally have *CHSH-expect* $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)\ R =$
sum $(\lambda a.\ K\ a *$ *CHSH-expect* $(A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)$
$(1_m\ nA \bigotimes B1)$
$((F\ a) \bigotimes (S\ a)))\ \{..< n\}$.

hence $|CHSH\text{-}expect\ (A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)\ R| \leq$
$sum\ (\lambda a.\ |K\ a * CHSH\text{-}expect\ (A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)$
$((F\ a) \bigotimes (S\ a))|)\ \{..< n\}$ **using** *sum-abs-cpx* **by** *simp*
also have $... = sum\ (\lambda a.\ K\ a * |CHSH\text{-}expect\ (A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)$
$(1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)\ ((F\ a) \bigotimes (S\ a))|)\ \{..< n\}$
proof (*rule sum.cong*)
fix x
assume $x \in \{..< n\}$
show $|complex\text{-}of\text{-}real\ (K\ x) *$
$CHSH\text{-}expect\ (A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)$
$(F\ x \bigotimes S\ x)| =$
$complex\text{-}of\text{-}real\ (K\ x) * |CHSH\text{-}expect\ (A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)$
$(1_m\ nA \bigotimes B0)\ (1_m\ nA \bigotimes B1)\ (F\ x \bigotimes S\ x)|$
proof (*rule abs-mult-cpx*)
show $0 \leq K\ x$
using ‹$x \in \{..< n\}$› *props cpx-of-real-ge-0*
unfolding *separately-decomposes-def* **by** *simp*
qed
qed *simp*
also have $... \leq sum\ (\lambda a.\ complex\text{-}of\text{-}real\ (K\ a) * 2)\ \{..< n\}$
proof (*rule sum-mono*)
fix a
assume $a \in \{..< n\}$
have $|CHSH\text{-}expect\ (A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)$
$(1_m\ nA \bigotimes B1)\ (F\ a \bigotimes S\ a)| \leq 2$
proof (*rule CHSH-expect-tensor-leq*)
show *CHSH-cond-local nA nB A0 A1 B0 B1* **using** *assms* **by** *simp*
show $F\ a \in carrier\text{-}mat\ nA\ nA$ **using** *props* ‹$a \in \{..< n\}$›
unfolding *separately-decomposes-def* **by** *simp*
show $density\text{-}operator\ (F\ a)$ **using** *props* ‹$a \in \{..< n\}$›
unfolding *separately-decomposes-def* **by** *simp*
show $S\ a \in carrier\text{-}mat\ nB\ nB$ **using** *props* ‹$a \in \{..< n\}$›
unfolding *separately-decomposes-def* **by** *simp*
show $density\text{-}operator\ (S\ a)$ **using** *props* ‹$a \in \{..< n\}$›
unfolding *separately-decomposes-def* **by** *simp*
qed (*auto simp add*: *separately-decomposes-carrier-pos*[*OF props*])
moreover have $0 \leq complex\text{-}of\text{-}real\ (K\ a)$
using *props* ‹$a \in \{..< n\}$› **unfolding** *separately-decomposes-def* **by** *simp*
ultimately show $complex\text{-}of\text{-}real\ (K\ a) *$
$|CHSH\text{-}expect\ (A0 \bigotimes 1_m\ nB)\ (A1 \bigotimes 1_m\ nB)\ (1_m\ nA \bigotimes B0)$
$(1_m\ nA \bigotimes B1)\ (F\ a \bigotimes S\ a)|$
$\leq complex\text{-}of\text{-}real\ (K\ a) * 2$
using *mult-left-mono* **by** *blast*
qed
also have $... = (sum\ (\lambda a.\ complex\text{-}of\text{-}real\ (K\ a))\ \{..< n\}) * 2$
by (*metis sum-distrib-right*)

```
  also have ... = 2
  proof −
    have sum (λa. complex-of-real (K a)) {..< n} = 1
      using props unfolding separately-decomposes-def
      by (metis  of-real-hom.hom-one of-real-hom.hom-sum)
    thus ?thesis by simp
  qed
  finally show ?thesis .
qed
```

When any of the pairs of observables used in the measurements commutes, this value remains bounded by 2.

```
lemma CHSH-expect-commute-leq:
  assumes CHSH-cond-hermit n A0 A1 B0 B1
  and R∈ carrier-mat n n
  and density-operator R
  and 0 < n
  and commutator A0 A1 = 0_m n n ∨ commutator B0 B1 = 0_m n n
shows |CHSH-expect A0 A1 B0 B1 R| ≤  2
proof −
  have cmod (CHSH-expect A0 A1 B0 B1 R) ≤ L2-op-nrm (CHSH-op A0 A1 B0
B1)
    unfolding CHSH-expect-def
  proof (rule expect-val-L2-op-nrm[of - n])
    show CHSH-op A0 A1 B0 B1 ∈ carrier-mat n n using assms CHSH-op-dim
      unfolding CHSH-cond-hermit-def CHSH-cond-def by auto
  qed (auto simp add: assms)
  also have ... = 2 using assms CHSH-op-commute-L2-op-nrm-eq by simp
  finally have cmod (CHSH-expect A0 A1 B0 B1 R) ≤ 2 .
  moreover have |CHSH-expect A0 A1 B0 B1 R| =
    cmod (CHSH-expect A0 A1 B0 B1 R)
    by (simp add: abs-complex-def)
  ultimately show ?thesis
    by (metis Reals-of-real abs-norm-cancel cpx-real-abs-eq
        cpx-real-abs-leq of-real-numeral)
qed
```

In the general case, this value is bounded by $2 \cdot \sqrt{2}$.

```
lemma CHSH-expect-gen-leq:
  assumes CHSH-cond-hermit n A0 A1 B0 B1
  and R∈ carrier-mat n n
  and density-operator R
  and 0 < n
shows |CHSH-expect A0 A1 B0 B1 R| ≤  (2 * sqrt 2)
proof −
  have cmod (CHSH-expect A0 A1 B0 B1 R) ≤ L2-op-nrm (CHSH-op A0 A1 B0
B1)
    unfolding CHSH-expect-def
  proof (rule expect-val-L2-op-nrm[of - n])
```

```
    show CHSH-op A0 A1 B0 B1 ∈ carrier-mat n n using assms CHSH-op-dim
      unfolding CHSH-cond-hermit-def CHSH-cond-def by auto
  qed (auto simp add: assms)
  also have ... ≤ 2 * sqrt 2 using assms CHSH-op-L2-op-nrm-le by simp
  finally have cmod (CHSH-expect A0 A1 B0 B1 R) ≤ 2 * sqrt 2 .
  moreover have |CHSH-expect A0 A1 B0 B1 R| =
    cmod (CHSH-expect A0 A1 B0 B1 R)
    by (simp add: abs-complex-def)
  ultimately show ?thesis
    by (metis Reals-of-real abs-norm-cancel cpx-real-abs-eq cpx-real-abs-leq)
qed
```

The bound $2 \cdot \sqrt{2}$ can be reached by a suitable choice of observables, when the Bell state is measured.

```
lemma CHSH-expect-limit:
shows |CHSH-expect Z-I X-I I-ZmX I-XpZ  rho-psim| = 2 * sqrt 2
proof −
  define fc::complex Matrix.mat set where fc = carrier-mat 4 4
  interpret bin-cpx 4 4 fc
  proof
    show 0 < (4::nat)  by simp
  qed (auto simp add: fc-def)
  have CHSH-expect Z-I X-I I-ZmX I-XpZ rho-psim =
    Complex-Matrix.trace (Z-I * I-XpZ * rho-psim) −
    Complex-Matrix.trace (Z-I * I-ZmX * rho-psim) +
    Complex-Matrix.trace (X-I * I-ZmX * rho-psim) +
    Complex-Matrix.trace (X-I * I-XpZ * rho-psim)
    using CHSH-expect-expand I-XpZ-carrier I-ZmX-carrier X-I-carrier
      Z-I-carrier rho-psim-carrier by blast
  also have ... = complex-of-real (1 / sqrt 2) −
    complex-of-real (− 1 / sqrt 2) +
    complex-of-real (1 / sqrt 2) +
    complex-of-real (1 / sqrt 2)
    using X-XpZ-rho-trace X-ZmX-rho-trace Z-XpZ-rho-trace Z-ZmX-rho-trace
    by presburger
  also have ... = 2 * sqrt 2
    using real-sqrt-divide two-div-sqrt-two by force
  finally have c: CHSH-expect Z-I X-I I-ZmX I-XpZ rho-psim = 2 * sqrt 2 .
  have |CHSH-expect Z-I X-I I-ZmX I-XpZ rho-psim| =
    |Re (CHSH-expect Z-I X-I I-ZmX I-XpZ rho-psim)|
    by (metis Re-complex-of-real Reals-of-real c cpx-real-abs-eq)
  thus ?thesis using c by simp
qed

end
```