Projective-Measurements

Mnacho

March 17, 2025

Contents

theory *Linear-Algebra-Complements* **imports**
Isabelle-Marries-Dirac.Tensor
Isabelle-Marries-Dirac.More-Tensor
QHLProver.Gates
HOL−Types-To-Sets.Group-On-With
HOL−Probability.Probability

begin
hide-const(**open**) *S*

1 Preliminaries

1.1 Misc

lemma *mult-real-cpx*:
fixes *a*::*complex*
fixes *b*::*complex*
assumes *a*∈ *Reals*
shows *a*∗ (*Re b*) = *Re* (*a* ∗ *b*) ⟨*proof*⟩

lemma *fct-bound*:
fixes *f*::*complex*⇒ *real*
assumes *f*(−*1*) + *f 1* = *1*
and *0* ≤ *f 1*
and *0* ≤ *f* (−*1*)
shows −*1* ≤ *f 1* − *f*(−*1*) ∧ *f 1* − *f*(−*1*) ≤ *1*
⟨*proof*⟩

lemma *fct-bound′*:
fixes *f*::*complex*⇒ *real*
assumes *f*(−*1*) + *f 1* = *1*
and *0* ≤ *f 1*
and *0* ≤ *f* (−*1*)
shows |*f 1* − *f*(−*1*)| ≤ *1* ⟨*proof*⟩

lemma *pos-sum-1-le*:
assumes *finite I*
and ∀ *i* ∈ *I*. (*0*::*real*) ≤ *f i*
and (∑ *i*∈ *I*. *f i*) = *1*
and *j*∈ *I*
shows *f j* ≤ *1*
⟨*proof*⟩

lemma *last-subset*:
assumes *A* ⊆ {*a*,*b*}
and *a*≠ *b*
and *A* ≠ {*a*, *b*}

and $A \neq \{\}$
and $A \neq \{a\}$
shows $A = \{b\}$ ⟨*proof*⟩

lemma *disjoint-Un*:
assumes *disjoint-family-on A* (*insert x F*)
and $x \notin F$
shows $(A\ x) \cap (\bigcup\ a \in F.\ A\ a) = \{\}$
⟨*proof*⟩

lemma *sum-but-one*:
assumes $\forall j < (n::nat).\ j \neq i \longrightarrow f\ j = (0::'a::ring)$
and $i < n$
shows $(\sum\ j \in \{0\ ..<\ n\}.\ f\ j * g\ j) = f\ i * g\ i$
⟨*proof*⟩

lemma *sum-2-elems*:
assumes $I = \{a,b\}$
and $a \neq b$
shows $(\sum a \in I.\ f\ a) = f\ a + f\ b$
⟨*proof*⟩

lemma *sum-4-elems*:
shows $(\sum i < (4::nat).\ f\ i) = f\ 0 + f\ 1 + f\ 2 + f\ 3$
⟨*proof*⟩

lemma *disj-family-sum*:
shows *finite I* $\Longrightarrow$ *disjoint-family-on A I* $\Longrightarrow$ $(\bigwedge i.\ i \in I \Longrightarrow$ *finite* $(A\ i)) \Longrightarrow$
$(\sum\ i \in (\bigcup n \in I.\ A\ n).\ f\ i) = (\sum\ n \in I.\ (\sum\ i \in A\ n.\ f\ i))$
⟨*proof*⟩

lemma *integrable-real-mult-right*:
fixes *c*::*real*
assumes *integrable M f*
shows *integrable M* $(\lambda w.\ c * f\ w)$
⟨*proof*⟩

1.2 Unifying notions between Isabelle Marries Dirac and QHLProver

lemma *mult-conj-cmod-square*:
fixes *z*::*complex*
shows $z * conjugate\ z = (cmod\ z)^2$
⟨*proof*⟩

lemma *vec-norm-sq-cpx-vec-length-sq*:
shows $(vec\text{-}norm\ v)^2 = (cpx\text{-}vec\text{-}length\ v)^2$
⟨*proof*⟩

lemma *vec-norm-eq-cpx-vec-length*:
shows *vec-norm v = cpx-vec-length v* ⟨*proof*⟩

lemma *cpx-vec-length-square*:
shows $\|v\|^2 = (\sum i = 0..<dim\text{-}vec\ v.\ (cmod\ (Matrix.vec\text{-}index\ v\ i))^2)$ ⟨*proof*⟩

lemma *state-qbit-norm-sq*:
assumes *v*∈ *state-qbit n*
shows $(cpx\text{-}vec\text{-}length\ v)^2 = 1$
⟨*proof*⟩

lemma *dagger-adjoint*:
shows *dagger M = Complex-Matrix.adjoint M* ⟨*proof*⟩

1.3 Types to sets lemmata transfers

context *ab-group-add-on-with* **begin**

context includes *lifting-syntax* **assumes** *ltd*: ∃(*Rep*::'*s* ⇒ '*a*) (*Abs*::'*a* ⇒ '*s*).
type-definition Rep Abs S **begin**
interpretation *local-typedef-ab-group-add-on-with pls z mns um S TYPE*('*s*) ⟨*proof*⟩

lemmas *lt-sum-union-disjoint = sum.union-disjoint*
[*var-simplified explicit-ab-group-add*,
unoverload-type '*c*,
OF type.comm-monoid-add-axioms,
untransferred]

lemmas *lt-disj-family-sum = disj-family-sum*
[*var-simplified explicit-ab-group-add*,
unoverload-type '*d*,
OF type.comm-monoid-add-axioms,
untransferred]

lemmas *lt-sum-reindex-cong = sum.reindex-cong*
[*var-simplified explicit-ab-group-add*,
unoverload-type '*d*,
OF type.comm-monoid-add-axioms,
untransferred]
end

lemmas *sum-with-union-disjoint =*
lt-sum-union-disjoint
[*cancel-type-definition*,
OF carrier-ne,
simplified pred-fun-def, *simplified*]

lemmas *disj-family-sum-with =*
lt-disj-family-sum

[*cancel-type-definition*,
OF carrier-ne,
simplified pred-fun-def, *simplified*]

lemmas *sum-with-reindex-cong* =
lt-sum-reindex-cong
[*cancel-type-definition*,
OF carrier-ne,
simplified pred-fun-def, *simplified*]

end

lemma (**in** *comm-monoid-add-on-with*) *sum-with-cong'*:
shows *finite I* $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow A\ i = B\ i$) $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow A\ i \in S$) $\Longrightarrow$
($\bigwedge i.\ i \in I \Longrightarrow B\ i \in S$) $\Longrightarrow$ *sum-with pls z A I* = *sum-with pls z B I*
⟨*proof*⟩

2 Linear algebra complements

2.1 Additional properties of matrices

lemma *smult-one*:
shows (*1*::'*a*::*monoid-mult*) $\cdot_m$ *A* = *A* ⟨*proof*⟩

lemma *times-diag-index*:
fixes *A*::'*a*::*comm-ring Matrix.mat*
assumes *A* ∈ *carrier-mat n n*
and *B*∈ *carrier-mat n n*
and *diagonal-mat B*
and *j* < *n*
and *i* < *n*
shows *Matrix.vec-index* (*Matrix.row* (*A*∗*B*) *j*) *i* = *diag-mat B* ! *i* ∗*A* $$ (*j*, *i*)
⟨*proof*⟩

lemma *inner-prod-adjoint-comp*:
assumes (*U*::'*a*::*conjugatable-field Matrix.mat*) ∈ *carrier-mat n n*
and (*V*::'*a*::*conjugatable-field Matrix.mat*) ∈ *carrier-mat n n*
and *i* < *n*
and *j* < *n*
shows *Complex-Matrix.inner-prod* (*Matrix.col V i*) (*Matrix.col U j*) =
((*Complex-Matrix.adjoint V*) ∗ *U*) $$ (*i*, *j*)
⟨*proof*⟩

lemma *mat-unit-vec-col*:
assumes (*A*::'*a*::*conjugatable-field Matrix.mat*) ∈ *carrier-mat n n*
and *i* < *n*
shows *A* ∗$_v$ (*unit-vec n i*) = *Matrix.col A i*
⟨*proof*⟩

lemma *mat-prod-unit-vec-cong*:
assumes $(A::'a::conjugatable\text{-}field\ Matrix.mat) \in carrier\text{-}mat\ n\ n$
and $B \in carrier\text{-}mat\ n\ n$
and $\bigwedge i.\ i < n \Longrightarrow A *_v (unit\text{-}vec\ n\ i) = B *_v (unit\text{-}vec\ n\ i)$
shows $A = B$
⟨*proof*⟩

lemma *smult-smult-times*:
fixes $a::'a::semigroup\text{-}mult$
shows $a \cdot_m (k \cdot_m A) = (a * k) \cdot_m A$
⟨*proof*⟩

lemma *mat-minus-minus*:
fixes $A :: 'a :: ab\text{-}group\text{-}add\ Matrix.mat$
assumes $A \in carrier\text{-}mat\ n\ m$
and $B \in carrier\text{-}mat\ n\ m$
and $C \in carrier\text{-}mat\ n\ m$
shows $A - (B - C) = A - B + C$
⟨*proof*⟩

2.2 Complements on complex matrices

lemma *hermitian-square*:
assumes *hermitian M*
shows $M \in carrier\text{-}mat\ (dim\text{-}row\ M)\ (dim\text{-}row\ M)$
⟨*proof*⟩

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

lemma *hermitian-minus*:
assumes $A \in carrier\text{-}mat\ n\ n$
and $B \in carrier\text{-}mat\ n\ n$
and *hermitian A*
and *hermitian B*
shows *hermitian* $(A - B)$ ⟨*proof*⟩

lemma *hermitian-smult*:
fixes $a::real$
fixes $A::complex\ Matrix.mat$
assumes $A \in carrier\text{-}mat\ n\ n$
and *hermitian A*
shows *hermitian* $(a \cdot_m A)$
⟨*proof*⟩

lemma *unitary-eigenvalues-norm-square*:
fixes U::*complex Matrix.mat*
assumes *unitary U*
and $U \in$ *carrier-mat n n*
and *eigenvalue U k*
shows *conjugate k* $*$ *k* $=$ *1*
⟨*proof*⟩

lemma *outer-prod-smult-left*:
fixes v::*complex Matrix.vec*
shows *outer-prod* $(a \cdot_v v)$ w $=$ $a \cdot_m$ *outer-prod v w*
⟨*proof*⟩

lemma *outer-prod-smult-right*:
fixes v::*complex Matrix.vec*
shows *outer-prod v* $(a \cdot_v w)$ $=$ $(conjugate\ a) \cdot_m$ *outer-prod v w*
⟨*proof*⟩

lemma *outer-prod-add-left*:
fixes v::*complex Matrix.vec*
assumes *dim-vec v* $=$ *dim-vec x*
shows *outer-prod* $(v + x)$ w $=$ *outer-prod v w* $+$ $(outer\text{-}prod\ x\ w)$
⟨*proof*⟩

lemma *outer-prod-add-right*:
fixes v::*complex Matrix.vec*
assumes *dim-vec w* $=$ *dim-vec x*
shows *outer-prod v* $(w + x)$ $=$ *outer-prod v w* $+$ $(outer\text{-}prod\ v\ x)$
⟨*proof*⟩

lemma *outer-prod-minus-left*:
fixes v::*complex Matrix.vec*
assumes *dim-vec v* $=$ *dim-vec x*
shows *outer-prod* $(v - x)$ w $=$ *outer-prod v w* $-$ $(outer\text{-}prod\ x\ w)$
⟨*proof*⟩

lemma *outer-prod-minus-right*:
fixes v::*complex Matrix.vec*
assumes *dim-vec w* $=$ *dim-vec x*
shows *outer-prod v* $(w - x)$ $=$ *outer-prod v w* $-$ $(outer\text{-}prod\ v\ x)$
⟨*proof*⟩

lemma *outer-minus-minus*:
fixes a::*complex Matrix.vec*
assumes *dim-vec a* $=$ *dim-vec b*
and *dim-vec u* $=$ *dim-vec v*
shows *outer-prod* $(a - b)$ $(u - v)$ $=$ *outer-prod a u* $-$ *outer-prod a v* $-$
outer-prod b u $+$ *outer-prod b v*
⟨*proof*⟩

lemma *outer-trace-inner*:
assumes $A \in$ *carrier-mat n n*
and *dim-vec u* $=$ *n*
and *dim-vec v* $=$ *n*
shows *Complex-Matrix.trace* (*outer-prod u v* $*$ *A*) $=$ *Complex-Matrix.inner-prod v* (*A* $*_v$ *u*)
⟨*proof*⟩

lemma *zero-hermitian*:
shows *hermitian* (*0*$_m$ *n n*) ⟨*proof*⟩

lemma *trace-1*:
shows *Complex-Matrix.trace* ((*1*$_m$ *n*)::*complex Matrix.mat*) $=$(*n*::*complex*) ⟨*proof*⟩

lemma *trace-add*:
assumes *square-mat A*
and *square-mat B*
and *dim-row A* $=$ *dim-row B*
shows *Complex-Matrix.trace* (*A* $+$ *B*) $=$ *Complex-Matrix.trace A* $+$ *Complex-Matrix.trace B*
⟨*proof*⟩

lemma *bra-vec-carrier*:
shows *bra-vec v* $\in$ *carrier-mat 1* (*dim-vec v*)
⟨*proof*⟩

lemma *mat-mult-ket-carrier*:
assumes *A*$\in$ *carrier-mat n m*
shows *A* $*$ $|v\rangle \in$ *carrier-mat n 1* ⟨*proof*⟩

lemma *mat-mult-ket*:
assumes *A* $\in$ *carrier-mat n m*
and *dim-vec v* $=$ *m*
shows *A* $*$ $|v\rangle = |A *_v v\rangle$
⟨*proof*⟩

lemma *unitary-density*:
assumes *density-operator R*
and *unitary U*
and *R*$\in$ *carrier-mat n n*
and *U*$\in$ *carrier-mat n n*
shows *density-operator* (*U* $*$ *R* $*$ (*Complex-Matrix.adjoint U*)) ⟨*proof*⟩

2.3 Tensor product complements

lemma *tensor-vec-dim*[*simp*]:
shows *dim-vec* (*tensor-vec u v*) $=$ *dim-vec u* $*$ (*dim-vec v*)
⟨*proof*⟩

lemma *index-tensor-vec*[*simp*]:
assumes *0 < dim-vec v*
and *i < dim-vec u * dim-vec v*
shows *vec-index (tensor-vec u v) i =*
*vec-index u (i div (dim-vec v)) * vec-index v (i mod dim-vec v)*
⟨*proof*⟩

lemma *outer-prod-tensor-comm*:
fixes *a*::*complex Matrix.vec*
fixes *u*::*complex Matrix.vec*
assumes *0 < dim-vec a*
and *0 < dim-vec b*
shows *outer-prod (tensor-vec u v) (tensor-vec a b) = tensor-mat (outer-prod u a) (outer-prod v b)*
⟨*proof*⟩

lemma *tensor-mat-adjoint*:
assumes *m1 ∈ carrier-mat r1 c1*
and *m2 ∈ carrier-mat r2 c2*
and *0 < c1*
and *0 < c2*
and *0 < r1*
and *0 < r2*
shows *Complex-Matrix.adjoint (tensor-mat m1 m2) =*
tensor-mat (Complex-Matrix.adjoint m1) (Complex-Matrix.adjoint m2)
⟨*proof*⟩

lemma *index-tensor-mat'*:
assumes *0 < dim-col A*
and *0 < dim-col B*
and *i < dim-row A * dim-row B*
and *j < dim-col A * dim-col B*
shows *(A ⨂ 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))*
⟨*proof*⟩

lemma *tensor-mat-carrier*:
shows *tensor-mat U V ∈ carrier-mat (dim-row U * dim-row V) (dim-col U * dim-col V)* ⟨*proof*⟩

lemma *tensor-mat-id*:
assumes *0 < d1*
and *0 < d2*
shows *tensor-mat (1$_m$ d1) (1$_m$ d2) = 1$_m$ (d1 * d2)*
⟨*proof*⟩

lemma *tensor-mat-hermitian*:

assumes $A \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n' n'*
and $0 < n$
and $0 < n'$
and *hermitian A*
and *hermitian B*
shows *hermitian* (*tensor-mat A B*) ⟨*proof*⟩

lemma *tensor-mat-unitary*:
assumes *Complex-Matrix.unitary U*
and *Complex-Matrix.unitary V*
and $0 <$ *dim-row U*
and $0 <$ *dim-row V*
shows *Complex-Matrix.unitary* (*tensor-mat U V*)
⟨*proof*⟩

2.4 Fixed carrier matrices locale

We define a locale for matrices with a fixed number of rows and columns, and define a finite sum operation on this locale. The `Type_To_Sets` transfer tools then permits to obtain lemmata on the locale for free.

locale *fixed-carrier-mat* =
fixes *fc-mats*::'*a*::*field Matrix.mat set*
fixes *dimR dimC*
assumes *fc-mats-carrier*: *fc-mats* = *carrier-mat dimR dimC*
begin

sublocale *semigroup-add-on-with fc-mats* (+)
⟨*proof*⟩

sublocale *ab-semigroup-add-on-with fc-mats* (+)
⟨*proof*⟩

sublocale *comm-monoid-add-on-with fc-mats* (+) 0_m *dimR dimC*
⟨*proof*⟩

sublocale *ab-group-add-on-with fc-mats* (+) 0_m *dimR dimC* (−) *uminus*
⟨*proof*⟩
end

lemma (**in** *fixed-carrier-mat*) *smult-mem*:
assumes $A \in$ *fc-mats*
shows $a \cdot_m A \in$ *fc-mats* ⟨*proof*⟩

definition (**in** *fixed-carrier-mat*) *sum-mat* **where**
sum-mat A I = *sum-with* (+) (0_m *dimR dimC*) *A I*

lemma (**in** *fixed-carrier-mat*) *sum-mat-empty*[*simp*]:
shows *sum-mat A* {} = 0_m *dimR dimC* ⟨*proof*⟩

lemma (**in** *fixed-carrier-mat*) *sum-mat-carrier*:
shows $(\bigwedge i.\ i \in I \Longrightarrow (A\ i) \in \textit{fc-mats}) \Longrightarrow \textit{sum-mat}\ A\ I \in \textit{carrier-mat}\ \textit{dimR}\ \textit{dimC}$
⟨*proof*⟩

lemma (**in** *fixed-carrier-mat*) *sum-mat-insert*:
assumes $A\ x \in \textit{fc-mats}\ A\ `\ I \subseteq \textit{fc-mats}$
and A: *finite* I **and** x: $x \notin I$
shows $\textit{sum-mat}\ A\ (\textit{insert}\ x\ I) = A\ x + \textit{sum-mat}\ A\ I$ ⟨*proof*⟩

2.5 A locale for square matrices

locale *cpx-sq-mat* = *fixed-carrier-mat fc-mats*::*complex Matrix.mat set* **for** *fc-mats* +
assumes *dim-eq*: $\textit{dimR} = \textit{dimC}$
and *npos*: $0 < \textit{dimR}$

lemma (**in** *cpx-sq-mat*) *one-mem*:
shows $1_m\ \textit{dimR} \in \textit{fc-mats}$ ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *square-mats*:
assumes $A \in \textit{fc-mats}$
shows *square-mat* A ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *cpx-sq-mat-mult*:
assumes $A \in \textit{fc-mats}$
and $B \in \textit{fc-mats}$
shows $A * B \in \textit{fc-mats}$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-distrib-left*:
shows $\textit{finite}\ I \Longrightarrow R \in \textit{fc-mats} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow (A\ i) \in \textit{fc-mats}) \Longrightarrow$
$\textit{sum-mat}\ (\lambda i.\ R * (A\ i))\ I = R * (\textit{sum-mat}\ A\ I)$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-distrib-right*:
shows $\textit{finite}\ I \Longrightarrow R \in \textit{fc-mats} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow (A\ i) \in \textit{fc-mats}) \Longrightarrow$
$\textit{sum-mat}\ (\lambda i.\ (A\ i) * R)\ I = (\textit{sum-mat}\ A\ I) * R$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *trace-sum-mat*:
fixes A::$'b \Rightarrow$ *complex Matrix.mat*
shows $\textit{finite}\ I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow (A\ i) \in \textit{fc-mats}) \Longrightarrow$
$\textit{Complex-Matrix.trace}\ (\textit{sum-mat}\ A\ I) = (\sum\ i \in I.\ \textit{Complex-Matrix.trace}\ (A\ i))$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *cpx-sq-mat-smult*:
assumes $A \in \textit{fc-mats}$

shows $x \cdot_m A \in$ *fc-mats*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *mult-add-distrib-right*:
assumes $A \in$ *fc-mats* $B \in$ *fc-mats* $C \in$ *fc-mats*
shows $A * (B + C) = A * B + A * C$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *mult-add-distrib-left*:
assumes $A \in$ *fc-mats* $B \in$ *fc-mats* $C \in$ *fc-mats*
shows $(B + C) * A = B * A + C * A$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *mult-sum-mat-distrib-left*:
shows *finite* $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow (A\ i) \in$ *fc-mats*$) \Longrightarrow B \in$ *fc-mats* $\Longrightarrow$
$($*sum-mat* $(\lambda i.\ B * (A\ i))\ I) = B * ($*sum-mat* $A\ I)$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *mult-sum-mat-distrib-right*:
shows *finite* $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow (A\ i) \in$ *fc-mats*$) \Longrightarrow B \in$ *fc-mats* $\Longrightarrow$
$($*sum-mat* $(\lambda i.\ (A\ i) * B)\ I) = ($*sum-mat* $A\ I) * B$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *trace-sum-mat-mat-distrib*:
assumes *finite* I
and $\bigwedge i.\ i \in I \Longrightarrow B\ i \in$ *fc-mats*
and $A \in$ *fc-mats*
and $C \in$ *fc-mats*
shows $(\sum\ i \in I.$ *Complex-Matrix.trace*$(A * (B\ i) * C)) =$
Complex-Matrix.trace $(A * ($*sum-mat* $B\ I) * C)$
⟨*proof*⟩

definition (**in** *cpx-sq-mat*) *zero-col* **where**
zero-col $U = (\lambda i.$ *if* $i <$ *dimR then Matrix.col* $U\ i$ *else* 0_v *dimR*$)$

lemma (**in** *cpx-sq-mat*) *zero-col-dim*:
assumes $U \in$ *fc-mats*
shows *dim-vec* $($*zero-col* $U\ i) =$ *dimR* ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *zero-col-col*:
assumes $i <$ *dimR*
shows *zero-col* $U\ i =$ *Matrix.col* $U\ i$ ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-index*:
shows *finite* $I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow (A\ i) \in$ *fc-mats*$) \Longrightarrow i <$ *dimR* $\Longrightarrow j <$ *dimC*
$\Longrightarrow$
$($*sum-mat* $(\lambda k.\ (A\ k))\ I)$ \$\$ $(i,j) = (\sum\ k \in I.\ (A\ k)$ \$\$ $(i,j))$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-cong*:
shows *finite I* $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow A\ i = B\ i$) $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *fc-mats*) $\Longrightarrow$
($\bigwedge i.\ i \in I \Longrightarrow B\ i \in$ *fc-mats*) $\Longrightarrow$ *sum-mat A I* = *sum-mat B I*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *smult-sum-mat*:
shows *finite I* $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *fc-mats*) $\Longrightarrow$ *a* $\cdot_m$ *sum-mat A I* = *sum-mat* ($\lambda i.\ a\ \cdot_m\ (A\ i)$) *I*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *zero-sum-mat*:
shows *finite I* $\Longrightarrow$ *sum-mat* ($\lambda i.$ ((0_m *dimR dimR*)::*complex Matrix.mat*)) *I* = ((0_m *dimR dimR*)::*complex Matrix.mat*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-adjoint*:
shows *finite I* $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *fc-mats*) $\Longrightarrow$
Complex-Matrix.adjoint (*sum-mat A I*) = *sum-mat* ($\lambda i.$ *Complex-Matrix.adjoint* (*A i*)) *I*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-hermitian*:
assumes *finite I*
and $\forall\, i \in I.$ *hermitian* (*A i*)
and $\forall\, i \in I.$ *A i* ∈ *fc-mats*
shows *hermitian* (*sum-mat A I*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-positive*:
shows *finite I* $\Longrightarrow$ ($\bigwedge i.\ i \in I \Longrightarrow$ *Complex-Matrix.positive* (*A i*)) $\Longrightarrow$
($\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *fc-mats*) $\Longrightarrow$ *Complex-Matrix.positive* (*sum-mat A I*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-left-ortho-zero*:
shows *finite I* $\Longrightarrow$
($\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *fc-mats*) $\Longrightarrow$ (*B* ∈ *fc-mats*) $\Longrightarrow$
($\bigwedge\ i.\ i \in I \Longrightarrow A\ i * B = (0_m$ *dimR dimR*)) $\Longrightarrow$
(*sum-mat A I*) ∗ *B* = 0_m *dimR dimR*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-right-ortho-zero*:
shows *finite I* $\Longrightarrow$
($\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *fc-mats*) $\Longrightarrow$ (*B* ∈ *fc-mats*) $\Longrightarrow$
($\bigwedge\ i.\ i \in I \Longrightarrow B * A\ i = (0_m$ *dimR dimR*)) $\Longrightarrow$
B ∗ (*sum-mat A I*) = 0_m *dimR dimR*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-mat-ortho-square*:

shows *finite I* $\Longrightarrow$ ($\bigwedge i.\ i\in I \Longrightarrow ((A\ i)::complex\ Matrix.mat) * (A\ i) = A\ i$)
$\Longrightarrow$
($\bigwedge i.\ i\in I \Longrightarrow A\ i \in fc\text{-}mats$) $\Longrightarrow$
($\bigwedge i\ j.\ i\in I \Longrightarrow j\in I \Longrightarrow i\neq j \Longrightarrow A\ i * (A\ j) = (0_m\ dimR\ dimR)$) $\Longrightarrow$
$(sum\text{-}mat\ A\ I) * (sum\text{-}mat\ A\ I) = (sum\text{-}mat\ A\ I)$
⟨*proof*⟩

lemma *diagonal-unit-vec*:
assumes $B \in carrier\text{-}mat\ n\ n$
and *diagonal-mat* (*B*::*complex Matrix.mat*)
shows $B *_v (unit\text{-}vec\ n\ i) = B\ \$\$\ (i,i)\ \cdot_v (unit\text{-}vec\ n\ i)$
⟨*proof*⟩

lemma *mat-vec-mult-assoc*:
assumes $A \in carrier\text{-}mat\ n\ p$
and $B\in carrier\text{-}mat\ p\ q$
and $dim\text{-}vec\ v = q$
shows $A *_v (B *_v v) = (A * B) *_v v$ ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *similar-eigenvectors*:
assumes $A\in fc\text{-}mats$
and $B\in fc\text{-}mats$
and $P\in fc\text{-}mats$
and *similar-mat-wit A B P* (*Complex-Matrix.adjoint P*)
and *diagonal-mat B*
and $i < n$
shows $A *_v (P *_v (unit\text{-}vec\ dimR\ i)) = B\ \$\$\ (i,i) \cdot_v (P *_v (unit\text{-}vec\ dimR\ i))$
⟨*proof*⟩

2.6 Projectors

definition *projector* **where**
$projector\ M \longleftrightarrow (hermitian\ M \wedge M * M = M)$

lemma *projector-hermitian*:
assumes *projector M*
shows *hermitian M* ⟨*proof*⟩

lemma *zero-projector*[*simp*]:
shows *projector* $(0_m\ n\ n)$ ⟨*proof*⟩

lemma *projector-square-eq*:
assumes *projector M*
shows $M * M = M$ ⟨*proof*⟩

lemma *projector-positive*:
assumes *projector M*
shows *Complex-Matrix.positive M*
⟨*proof*⟩

lemma *projector-collapse-trace*:
assumes *projector* (*P*::*complex Matrix.mat*)
and $P \in$ *carrier-mat n n*
and *R*∈ *carrier-mat n n*
shows *Complex-Matrix.trace* $(P * R * P)$ = *Complex-Matrix.trace* $(R * P)$
⟨*proof*⟩

lemma *positive-proj-trace*:
assumes *projector* (*P*::*complex Matrix.mat*)
and *Complex-Matrix.positive R*
and $P \in$ *carrier-mat n n*
and *R*∈ *carrier-mat n n*
shows *Complex-Matrix.trace* $(R * P) \geq 0$
⟨*proof*⟩

lemma *trace-proj-pos-real*:
assumes *projector* (*P*::*complex Matrix.mat*)
and *Complex-Matrix.positive R*
and $P \in$ *carrier-mat n n*
and *R*∈ *carrier-mat n n*
shows *Re* (*Complex-Matrix.trace* $(R * P)$) = *Complex-Matrix.trace* $(R * P)$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *trace-sum-mat-proj-pos-real*:
fixes *f*::$'a \Rightarrow$ *real*
assumes *finite I*
and ∀ *i*∈ *I*. *projector* (*P i*)
and *Complex-Matrix.positive R*
and ∀ *i*∈ *I*. *P i* ∈ *fc-mats*
and *R* ∈ *fc-mats*
shows *Complex-Matrix.trace* $(R * (sum\text{-}mat\ (\lambda i.\ f\ i \cdot_m (P\ i))\ I)) =$
Re (*Complex-Matrix.trace* $(R * (sum\text{-}mat\ (\lambda i.\ f\ i \cdot_m (P\ i))\ I)))$
⟨*proof*⟩

2.7 Rank 1 projection

definition *rank-1-proj* **where**
rank-1-proj v = *outer-prod v v*

lemma *rank-1-proj-square-mat*:
shows *square-mat* (*rank-1-proj v*) ⟨*proof*⟩

lemma *rank-1-proj-dim*[*simp*]:
shows *dim-row* (*rank-1-proj v*) = *dim-vec v* ⟨*proof*⟩

lemma *rank-1-proj-carrier*[*simp*]:
shows *rank-1-proj v* ∈ *carrier-mat* (*dim-vec v*) (*dim-vec v*) ⟨*proof*⟩

lemma *rank-1-proj-coord*:
assumes $i < dim\text{-}vec\ v$
and $j < dim\text{-}vec\ v$
shows $(rank\text{-}1\text{-}proj\ v)\ \$\$\ (i,\ j) = Matrix.vec\text{-}index\ v\ i * (cnj\ (Matrix.vec\text{-}index\ v\ j))$ ⟨*proof*⟩

lemma *rank-1-proj-adjoint*:
shows $Complex\text{-}Matrix.adjoint\ (rank\text{-}1\text{-}proj\ (v::complex\ Matrix.vec)) = rank\text{-}1\text{-}proj\ v$
⟨*proof*⟩

lemma *rank-1-proj-hermitian*:
shows $hermitian\ (rank\text{-}1\text{-}proj\ (v::complex\ Matrix.vec))$ ⟨*proof*⟩

lemma *rank-1-proj-trace*:
assumes $\|v\| = 1$
shows $Complex\text{-}Matrix.trace\ (rank\text{-}1\text{-}proj\ v) = 1$
⟨*proof*⟩

lemma *rank-1-proj-mat-col*:
assumes $A \in carrier\text{-}mat\ n\ n$
and $i < n$
and $j < n$
and $k < n$
shows $(rank\text{-}1\text{-}proj\ (Matrix.col\ A\ i))\ \$\$\ (j,\ k) = A\ \$\$\ (j,\ i) * conjugate\ (A\ \$\$\ (k,i))$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *weighted-sum-rank-1-proj-unitary-index*:
assumes $A \in fc\text{-}mats$
and $B \in fc\text{-}mats$
and $diagonal\text{-}mat\ B$
and $Complex\text{-}Matrix.unitary\ A$
and $j < dimR$
and $k < dimR$
shows $(sum\text{-}mat\ (\lambda i.\ (diag\text{-}mat\ B)!i \cdot_m rank\text{-}1\text{-}proj\ (Matrix.col\ A\ i))\ \{..< dimR\})\ \$\$\ (j,k) =$
$(A * B * (Complex\text{-}Matrix.adjoint\ A))\ \$\$\ (j,k)$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *weighted-sum-rank-1-proj-unitary*:
assumes $A \in fc\text{-}mats$
and $B \in fc\text{-}mats$
and $diagonal\text{-}mat\ B$
and $Complex\text{-}Matrix.unitary\ A$
shows $(sum\text{-}mat\ (\lambda i.\ (diag\text{-}mat\ B)!i \cdot_m rank\text{-}1\text{-}proj\ (Matrix.col\ A\ i))\ \{..< dimR\}) =$
$(A * B * (Complex\text{-}Matrix.adjoint\ A))$
⟨*proof*⟩

lemma *rank-1-proj-projector*:
assumes $\|v\| = 1$
shows *projector* (*rank-1-proj v*)
⟨*proof*⟩

lemma *rank-1-proj-positive*:
assumes $\|v\| = 1$
shows *Complex-Matrix.positive* (*rank-1-proj v*)
⟨*proof*⟩

lemma *rank-1-proj-density*:
assumes $\|v\| = 1$
shows *density-operator* (*rank-1-proj v*) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-rank-1-proj-unitary-index*:
assumes $A \in$ *fc-mats*
and *Complex-Matrix.unitary A*
and $j <$ *dimR*
and $k <$ *dimR*
shows (*sum-mat* ($\lambda i.$ *rank-1-proj* (*Matrix.col A i*)) {..< *dimR*}) $$ (*j*,*k*) = (1_m *dimR*) $$ (*j*,*k*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *rank-1-proj-sum-density*:
assumes *finite I*
and $\forall i \in I.$ $\|u\ i\| = 1$
and $\forall i \in I.$ *dim-vec* (*u i*) = *dimR*
and $\forall i \in I.$ $0 \leq p\ i$
and $(\sum i \in I.\ p\ i) = 1$
shows *density-operator* (*sum-mat* ($\lambda i.$ $p\ i \cdot_m$ (*rank-1-proj* (*u i*))) *I*) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-rank-1-proj-unitary*:
assumes $A \in$ *fc-mats*
and *Complex-Matrix.unitary A*
shows (*sum-mat* ($\lambda i.$ *rank-1-proj* (*Matrix.col A i*)) {..< *dimR*})= (1_m *dimR*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *rank-1-proj-unitary*:
assumes $A \in$ *fc-mats*
and *Complex-Matrix.unitary A*
and $j <$ *dimR*
and $k <$ *dimR*
shows *rank-1-proj* (*Matrix.col A j*) $*$ (*rank-1-proj* (*Matrix.col A k*)) =
(1_m *dimR*) $$ (*j*,*k*) $\cdot_m$ (*outer-prod* (*Matrix.col A j*) (*Matrix.col A k*))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *rank-1-proj-unitary-ne*:
assumes $A \in$ *fc-mats*
and *Complex-Matrix.unitary A*
and $j <$ *dimR*
and $k <$ *dimR*
and $j \neq k$
shows *rank-1-proj* (*Matrix.col A j*) $*$ (*rank-1-proj* (*Matrix.col A k*)) $=$ (0_m *dimR dimR*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *rank-1-proj-unitary-eq*:
assumes $A \in$ *fc-mats*
and *Complex-Matrix.unitary A*
and $j <$ *dimR*
shows *rank-1-proj* (*Matrix.col A j*) $*$ (*rank-1-proj* (*Matrix.col A j*)) $=$ *rank-1-proj* (*Matrix.col A j*)
⟨*proof*⟩

end

theory *Projective-Measurements* **imports**
Linear-Algebra-Complements

begin

3 Projective measurements

In this part we define projective measurements, also refered to as von Neumann measurements. The latter are characterized by a set of orthogonal projectors, which are used to compute the probabilities of measure outcomes and to represent the state of the system after the measurement.

The state of the system (a density operator in this case) after a measurement is represented by the `density_collapse` function.

3.1 First definitions

We begin by defining a type synonym for couples of measurement values (reals) and the associated projectors (complex matrices).

type-synonym *measure-outcome* $=$ *real* $\times$ *complex Matrix.mat*

The corresponding values and projectors are retrieved thanks to `meas_outcome_val` and `meas_outcome_prj`.

definition *meas-outcome-val*::*measure-outcome* $\Rightarrow$ *real* **where**

meas-outcome-val Mi = fst Mi

definition *meas-outcome-prj*::*measure-outcome* ⇒ *complex Matrix.mat* **where**
meas-outcome-prj Mi = snd Mi

We define a predicate for projective measurements. A projective measurement is characterized by the number p of possible measure outcomes, and a function M mapping outcome i to the corresponding `measure_outcome`.

definition (**in** *cpx-sq-mat*) *proj-measurement*::*nat* ⇒ (*nat* ⇒ *measure-outcome*) ⇒ *bool* **where**
proj-measurement n M ⟷
(*inj-on* (λ*i*. *meas-outcome-val* (*M i*)) {..< *n*}) ∧
(∀ *j* < *n*. *meas-outcome-prj* (*M j*) ∈ *fc-mats* ∧
projector (*meas-outcome-prj* (*M j*))) ∧
(∀ *i* < *n*. ∀ *j* < *n*. *i* ≠ *j* ⟶
meas-outcome-prj (*M i*) ∗ *meas-outcome-prj* (*M j*) = 0_m *dimR dimR*) ∧
sum-mat (λ*j*. *meas-outcome-prj* (*M j*)) {..< *n*} = 1_m *dimR*

lemma (**in** *cpx-sq-mat*) *proj-measurement-inj*:
assumes *proj-measurement p M*
shows *inj-on* (λ*i*. *meas-outcome-val* (*M i*)) {..< *p*} ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *proj-measurement-carrier*:
assumes *proj-measurement p M*
and *i* < *p*
shows *meas-outcome-prj* (*M i*) ∈ *fc-mats* ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *proj-measurement-ortho*:
assumes *proj-measurement p M*
and *i* < *p*
and *j* < *p*
and *i*≠ *j*
shows *meas-outcome-prj* (*M i*) ∗ *meas-outcome-prj* (*M j*) = 0_m *dimR dimR*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *proj-measurement-id*:
assumes *proj-measurement p M*
shows *sum-mat* (λ*j*. *meas-outcome-prj* (*M j*)) {..< *p*} = 1_m *dimR* ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *proj-measurement-square*:
assumes *proj-measurement p M*
and *i* < *p*
shows *meas-outcome-prj* (*M i*) ∈ *fc-mats* ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *proj-measurement-proj*:
assumes *proj-measurement p M*
and *i* < *p*
shows *projector* (*meas-outcome-prj* (*M i*)) ⟨*proof*⟩

We define the probability of obtaining a measurement outcome: this is a positive number and the sum over all the measurement outcomes is 1.

definition (**in** *cpx-sq-mat*) *meas-outcome-prob* :: *complex Matrix.mat* ⇒
(*nat* ⇒ *real* × *complex Matrix.mat*) ⇒ *nat* ⇒ *complex* **where**
meas-outcome-prob R M i = *Complex-Matrix.trace* (*R*∗ (*meas-outcome-prj* (*M i*)))

lemma (**in** *cpx-sq-mat*) *meas-outcome-prob-real*:
assumes *R*∈ *fc-mats*
and *density-operator R*
and *proj-measurement n M*
and *i* < *n*
shows *meas-outcome-prob R M i* ∈ ℝ
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *meas-outcome-prob-pos*:
assumes *R*∈ *fc-mats*
and *density-operator R*
and *proj-measurement n M*
and *i* < *n*
shows *0* ≤ *meas-outcome-prob R M i* ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *meas-outcome-prob-sum*:
assumes *density-operator R*
and *R*∈ *fc-mats*
and *proj-measurement n M*
shows (∑ *j* ∈ {..< *n*}. *meas-outcome-prob R M j*) = *1*
⟨*proof*⟩

We introduce the maximally mixed density operator. Intuitively, this density operator corresponds to a uniform distribution of the states of an orthonormal basis. This operator will be used to define the density operator after a measurement for the measure outcome probabilities equal to zero.

definition *max-mix-density* :: *nat* ⇒ *complex Matrix.mat* **where**
max-mix-density n = ((*1*::*real*)/ *n*) ·$_m$ (*1*$_m$ *n*)

lemma *max-mix-density-carrier*:
shows *max-mix-density n* ∈ *carrier-mat n n* ⟨*proof*⟩

lemma *max-mix-is-density*:
assumes *0* < *n*
shows *density-operator* (*max-mix-density n*) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *max-mix-density-square*:
shows *max-mix-density dimR* ∈ *fc-mats* ⟨*proof*⟩

Given a measurement outcome, `density_collapse` represents the resulting density operator. In practice only the measure outcomes with nonzero probabilities are of interest; we (arbitrarily) collapse the density operator

for zero-probability outcomes to the maximally mixed density operator.

definition *density-collapse* ::*complex Matrix.mat* ⇒ *complex Matrix.mat* ⇒ *complex Matrix.mat* **where**
density-collapse R P = (*if* ((*Complex-Matrix.trace* (*R* ∗ *P*)) = *0*) *then* (*max-mix-density* (*dim-row R*))
else ((*1*::*real*)/ ((*Complex-Matrix.trace* (*R* ∗ *P*)))) ·$_m$ (*P* ∗ *R* ∗ *P*))

lemma *density-collapse-carrier*:
assumes *0* < *dim-row R*
and *P* ∈ *carrier-mat n n*
and *R* ∈ *carrier-mat n n*
shows (*density-collapse R P*) ∈ *carrier-mat n n*
⟨*proof*⟩

lemma *density-collapse-operator*:
assumes *projector P*
and *density-operator R*
and *0* < *dim-row R*
and *P* ∈ *carrier-mat n n*
and *R* ∈ *carrier-mat n n*
shows *density-operator* (*density-collapse R P*)
⟨*proof*⟩

3.2 Measurements with observables

It is standard in quantum mechanics to represent projective measurements with so-called *observables*. These are Hermitian matrices which encode projective measurements as follows: the eigenvalues of an observable represent the possible projective measurement outcomes, and the associated projectors are the projectors onto the corresponding eigenspaces. The results in this part are based on the spectral theorem, which states that any Hermitian matrix admits an orthonormal basis consisting of eigenvectors of the matrix.

3.2.1 On the diagonal elements of a matrix

We begin by introducing definitions that will be used on the diagonalized version of a Hermitian matrix.

definition *diag-elems* **where**
diag-elems B = {*B*$$(*i*,*i*) |*i*. *i* < *dim-row B*}

Relationship between `diag_elems` and the list `diag_mat`

lemma *diag-elems-set-diag-mat*:
shows *diag-elems B* = *set* (*diag-mat B*) ⟨*proof*⟩

lemma *diag-elems-finite*[*simp*]:
shows *finite* (*diag-elems B*) ⟨*proof*⟩

lemma *diag-elems-mem*[*simp*]:
assumes $i < dim\text{-}row\ B$
shows $B\ \$\$(i,i) \in diag\text{-}elems\ B\ \langle proof \rangle$

When x is a diagonal element of B, `diag_elem_indices` returns the set of diagonal indices of B with value x.

definition *diag-elem-indices* **where**
$diag\text{-}elem\text{-}indices\ x\ B = \{i | i.\ i < dim\text{-}row\ B \wedge B\ \$\$\ (i,i) = x\}$

lemma *diag-elem-indices-elem*:
assumes $a \in diag\text{-}elem\text{-}indices\ x\ B$
shows $a < dim\text{-}row\ B \wedge B\$\$(a,a) = x\ \langle proof \rangle$

lemma *diag-elem-indices-itself*:
assumes $i < dim\text{-}row\ B$
shows $i \in diag\text{-}elem\text{-}indices\ (B\ \$\$(i,i))\ B\ \langle proof \rangle$

lemma *diag-elem-indices-finite*:
shows $finite\ (diag\text{-}elem\text{-}indices\ x\ B)\ \langle proof \rangle$

We can therefore partition the diagonal indices of a matrix B depending on the value of the diagonal elements. If B admits p elements on its diagonal, then we define bijections between its set of diagonal elements and the initial segment $[0..p-1]$.

definition *dist-el-card* **where**
$dist\text{-}el\text{-}card\ B = card\ (diag\text{-}elems\ B)$

definition *diag-idx-to-el* **where**
$diag\text{-}idx\text{-}to\text{-}el\ B = (SOME\ h.\ bij\text{-}betw\ h\ \{..<\ dist\text{-}el\text{-}card\ B\}\ (diag\text{-}elems\ B))$

definition *diag-el-to-idx* **where**
$diag\text{-}el\text{-}to\text{-}idx\ B = inv\text{-}into\ \{..<\ dist\text{-}el\text{-}card\ B\}\ (diag\text{-}idx\text{-}to\text{-}el\ B)$

lemma *diag-idx-to-el-bij*:
shows $bij\text{-}betw\ (diag\text{-}idx\text{-}to\text{-}el\ B)\ \{..<\ dist\text{-}el\text{-}card\ B\}\ (diag\text{-}elems\ B)$
$\langle proof \rangle$

lemma *diag-el-to-idx-bij*:
shows $bij\text{-}betw\ (diag\text{-}el\text{-}to\text{-}idx\ B)\ (diag\text{-}elems\ B)\ \{..<\ dist\text{-}el\text{-}card\ B\}$
$\langle proof \rangle$

lemma *diag-idx-to-el-less-inj*:
assumes $i < dist\text{-}el\text{-}card\ B$
and $j < dist\text{-}el\text{-}card\ B$
and $diag\text{-}idx\text{-}to\text{-}el\ B\ i = diag\text{-}idx\text{-}to\text{-}el\ B\ j$
shows $i = j$
$\langle proof \rangle$

lemma *diag-idx-to-el-less-surj*:
  assumes $x \in$ *diag-elems B*
shows $\exists\, k \in \{..<$ *dist-el-card B*$\}$. $x =$ *diag-idx-to-el B k*
⟨*proof*⟩

lemma *diag-idx-to-el-img*:
  assumes $k <$ *dist-el-card B*
shows *diag-idx-to-el B k* $\in$ *diag-elems B*
⟨*proof*⟩

lemma *diag-elems-real*:
  fixes *B*::*complex Matrix.mat*
  assumes $\forall\, i <$ *dim-row B*. *B*$$(*i*,*i*) $\in$ *Reals*
  shows *diag-elems B* $\subseteq$ *Reals*
⟨*proof*⟩

lemma *diag-elems-Re*:
  fixes *B*::*complex Matrix.mat*
  assumes $\forall\, i <$ (*dim-row B*). *B*$$(*i*,*i*) $\in$ *Reals*
  shows $\{$*Re x*$|$*x*. *x*$\in$ *diag-elems B*$\}$ = *diag-elems B*
⟨*proof*⟩

lemma *diag-idx-to-el-real*:
  fixes *B*::*complex Matrix.mat*
  assumes $\forall\, i <$ *dim-row B*. *B*$$(*i*,*i*) $\in$ *Reals*
and $i <$ *dist-el-card B*
shows *Re* (*diag-idx-to-el B i*) = *diag-idx-to-el B i*
⟨*proof*⟩

lemma *diag-elem-indices-empty*:
  assumes *B* $\in$ *carrier-mat dimR dimC*
  and $i <$ (*dist-el-card B*)
and $j <$ (*dist-el-card B*)
and $i \neq j$
shows *diag-elem-indices* (*diag-idx-to-el B i*) *B* $\cap$
  (*diag-elem-indices* (*diag-idx-to-el B j*) *B*) = {}
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *diag-elem-indices-disjoint*:
  assumes *B*$\in$ *carrier-mat dimR dimC*
  shows *disjoint-family-on* (λn. *diag-elem-indices* (*diag-idx-to-el B n*) *B*)
    {..<*dist-el-card B*} ⟨*proof*⟩

lemma *diag-elem-indices-union*:
  assumes *B*$\in$ *carrier-mat dimR dimC*
  shows ($\bigcup$ *i* $\in$ {..< *dist-el-card B*}. *diag-elem-indices* (*diag-idx-to-el B i*) *B*) =
    {..< *dimR*}
⟨*proof*⟩

3.2.2 Construction of measurement outcomes

The construction of a projective measurement for a hermitian matrix A is based on the Schur decomposition $A = U * B * U^\dagger$, where B is diagonal and U is unitary. The columns of U are normalized and pairwise orthogonal; they are used to construct the projectors associated with a measurement value

definition (**in** *cpx-sq-mat*) *project-vecs* **where**
project-vecs (*f*::*nat* ⇒ *complex Matrix.vec*) *N* = *sum-mat* (λ*i*. *rank-1-proj* (*f i*)) *N*

lemma (**in** *cpx-sq-mat*) *project-vecs-dim*:
assumes ∀ *i* ∈ *N*. *dim-vec* (*f i*) = *dimR*
shows *project-vecs f N* ∈ *fc-mats*
⟨*proof*⟩

definition (**in** *cpx-sq-mat*) *mk-meas-outcome* **where**
mk-meas-outcome B U = (λ*i*. (*Re* (*diag-idx-to-el B i*),
project-vecs (λ*i*. *zero-col U i*) (*diag-elem-indices* (*diag-idx-to-el B i*) *B*)))

lemma (**in** *cpx-sq-mat*) *mk-meas-outcome-carrier*:
assumes *Complex-Matrix.unitary U*
and *U* ∈ *fc-mats*
and *B*∈ *fc-mats*
shows *meas-outcome-prj* ((*mk-meas-outcome B U*) *j*) ∈ *fc-mats*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *mk-meas-outcome-sum-id*:
assumes *Complex-Matrix.unitary U*
and *U* ∈ *fc-mats*
and *B*∈ *fc-mats*
shows *sum-mat* (λ*j*. *meas-outcome-prj* ((*mk-meas-outcome B U*) *j*))
{..<(*dist-el-card B*)} = 1_m *dimR*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-meas-outcome-prj-ortho*:
assumes *Complex-Matrix.unitary U*
and *U* ∈ *fc-mats*
and *B*∈ *fc-mats*
and *i* < *dist-el-card B*
and *j* < *dist-el-card B*
and *i* ≠ *j*
shows *meas-outcome-prj* ((*mk-meas-outcome B U*) *i*) ∗
meas-outcome-prj ((*mk-meas-outcome B U*) *j*) = 0_m *dimR dimR*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-meas-outcome-prjectors*:
assumes *Complex-Matrix.unitary U*
and *U* ∈ *fc-mats*

and $B \in$ *fc-mats*
and $j <$ *dist-el-card B*
shows *projector* (*meas-outcome-prj* ((*mk-meas-outcome B U*) *j*)) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *mk-meas-outcome-fst-inj*:
assumes $\forall\, i <$ (*dim-row B*). *B*$$(*i*,*i*) ∈ *Reals*
shows *inj-on* (λ*i*. *meas-outcome-val* ((*mk-meas-outcome B U*) *i*)) {..<*dist-el-card B*}
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *mk-meas-outcome-fst-bij*:
assumes $\forall\, i <$ (*dim-row B*). *B*$$(*i*,*i*) ∈ *Reals*
shows *bij-betw* (λ*i*. *meas-outcome-val* ((*mk-meas-outcome B U*) *i*)) {..< *dist-el-card B*}
{*Re x*|*x*. *x* ∈ *diag-elems B*}
⟨*proof*⟩

3.2.3 Projective measurement associated with an observable

definition *eigvals* **where**
eigvals M = (*SOME as*. *char-poly M* = (∏ *a*←*as*. [:− *a*, *1*:]) ∧ *length as* = *dim-row M*)

lemma *eigvals-poly-length*:
assumes (*M*::*complex Matrix.mat*) ∈ *carrier-mat n n*
shows *char-poly M* = (∏ *a*←(*eigvals M*). [:− *a*, *1*:]) ∧ *length* (*eigvals M*) = *dim-row M*
⟨*proof*⟩

We define the spectrum of a matrix A: this is its set of eigenvalues; its elements are roots of the characteristic polynomial of A.

definition *spectrum* **where**
spectrum M = *set* (*eigvals M*)

lemma *spectrum-finite*:
shows *finite* (*spectrum M*) ⟨*proof*⟩

lemma *spectrum-char-poly-root*:
fixes *A*::*complex Matrix.mat*
assumes *A* ∈ *carrier-mat n n*
and *k* ∈ *spectrum A*
shows *poly* (*char-poly A*) *k* = *0* ⟨*proof*⟩

lemma *spectrum-eigenvalues*:
fixes *A*::*complex Matrix.mat*
assumes *A* ∈ *carrier-mat n n*
and *k* ∈ *spectrum A*
shows *eigenvalue A k* ⟨*proof*⟩

The main result that is used to construct a projective measurement for

a Hermitian matrix is that it is always possible to decompose it as $A = U * B * U^{\dagger}$, where B is diagonal and only contains real elements, and U is unitary.

definition *hermitian-decomp* **where**
hermitian-decomp A B U ≡ *similar-mat-wit A B U* (*Complex-Matrix.adjoint U*)
∧ *diagonal-mat B* ∧
diag-mat B = (*eigvals A*) ∧ *unitary U* ∧ (∀ *i* < *dim-row B*. *B*$$(*i*, *i*) ∈ *Reals*)

lemma *hermitian-decomp-sim*:
assumes *hermitian-decomp A B U*
shows *similar-mat-wit A B U* (*Complex-Matrix.adjoint U*) ⟨*proof*⟩

lemma *hermitian-decomp-diag-mat*:
assumes *hermitian-decomp A B U*
shows *diagonal-mat B* ⟨*proof*⟩

lemma *hermitian-decomp-eigenvalues*:
assumes *hermitian-decomp A B U*
shows *diag-mat B* = (*eigvals A*) ⟨*proof*⟩

lemma *hermitian-decomp-unitary*:
assumes *hermitian-decomp A B U*
shows *unitary U* ⟨*proof*⟩

lemma *hermitian-decomp-real-eigvals*:
assumes *hermitian-decomp A B U*
shows ∀ *i* < *dim-row B*. *B*$$(*i*, *i*) ∈ *Reals* ⟨*proof*⟩

lemma *hermitian-decomp-dim-carrier*:
assumes *hermitian-decomp A B U*
shows *B* ∈ *carrier-mat* (*dim-row A*) (*dim-col A*) ⟨*proof*⟩

lemma *similar-mat-wit-dim-row*:
assumes *similar-mat-wit A B Q R*
shows *dim-row B* = *dim-row A* ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *hermitian-schur-decomp*:
assumes *hermitian A*
and *A*∈ *fc-mats*
obtains *B U* **where** *hermitian-decomp A B U*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *hermitian-spectrum-real*:
assumes *A* ∈ *fc-mats*
and *hermitian A*
and *a* ∈ *spectrum A*
shows *a* ∈ *Reals*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-ne*:
assumes $A \in$ *fc-mats*
and *hermitian A*
shows *spectrum A* $\neq \{\}$
⟨*proof*⟩

lemma *unitary-hermitian-eigenvalues*:
fixes *U*::*complex Matrix.mat*
assumes *unitary U*
and *hermitian U*
and $U \in$ *carrier-mat n n*
and $0 < n$
and $k \in$ *spectrum U*
shows $k \in \{-1, 1\}$
⟨*proof*⟩

lemma *unitary-hermitian-Re-spectrum*:
fixes *U*::*complex Matrix.mat*
assumes *unitary U*
and *hermitian U*
and $U \in$ *carrier-mat n n*
and $0 < n$
shows $\{Re\ x | x.\ x \in$ *spectrum U*$\} \subseteq \{-1,1\}$
⟨*proof*⟩

The projective measurement associated with matrix M is obtained from its Schur decomposition, by considering the number of distinct elements on the resulting diagonal matrix and constructing the projectors associated with each of them.

type-synonym *proj-meas-rep* = *nat* × (*nat* ⇒ *measure-outcome*)

definition *proj-meas-size*::*proj-meas-rep* ⇒ *nat* **where**
proj-meas-size P = *fst P*

definition *proj-meas-outcomes*::*proj-meas-rep* ⇒ (*nat* ⇒ *measure-outcome*) **where**
proj-meas-outcomes P = *snd P*

definition (**in** *cpx-sq-mat*) *make-pm*::*complex Matrix.mat* ⇒ *proj-meas-rep* **where**
make-pm A = (*let* (*B*,*U*,-) = *unitary-schur-decomposition A* (*eigvals A*) *in*
(*dist-el-card B*, *mk-meas-outcome B U*))

lemma (**in** *cpx-sq-mat*) *make-pm-decomp*:
shows *make-pm A* = (*proj-meas-size* (*make-pm A*), *proj-meas-outcomes* (*make-pm A*))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-pm-proj-measurement*:
assumes $A \in$ *fc-mats*
and *hermitian A*

and *make-pm A* = (*n*, *M*)
shows *proj-measurement n M*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-pm-proj-measurement′*:
assumes *A*∈ *fc-mats*
and *hermitian A*
shows *proj-measurement* (*proj-meas-size* (*make-pm A*)) (*proj-meas-outcomes* (*make-pm A*))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-pm-projectors*:
assumes *A*∈ *fc-mats*
and *hermitian A*
and *i* < *proj-meas-size* (*make-pm A*)
shows *projector* (*meas-outcome-prj* (*proj-meas-outcomes* (*make-pm A*) *i*))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-pm-square*:
assumes *A*∈ *fc-mats*
and *hermitian A*
and *i* < *proj-meas-size* (*make-pm A*)
shows *meas-outcome-prj* (*proj-meas-outcomes* (*make-pm A*) *i*) ∈ *fc-mats*
⟨*proof*⟩

3.2.4 Properties on the spectrum of a Hermitian matrix

lemma (**in** *cpx-sq-mat*) *hermitian-schur-decomp′*:
assumes *hermitian A*
and *A*∈ *fc-mats*
obtains *B U* **where** *hermitian-decomp A B U* ∧
make-pm A = (*dist-el-card B*, *mk-meas-outcome B U*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-meas-outcome-val-eq*:
assumes *hermitian A*
and *A* ∈ *fc-mats*
and *make-pm A* = (*p*, *M*)
shows *spectrum A* = (λ*i*. *meas-outcome-val* (*M i*)) ‘{..< *p*}
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-meas-outcome-val-eq′*:
assumes *hermitian A*
and *A* ∈ *fc-mats*
and *make-pm A* = (*p*, *M*)
shows {*Re x*|*x*. *x*∈ *spectrum A*} = (λ*i*. *meas-outcome-val* (*M i*)) ‘{..< *p*}
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-pm-eigenvalues*:

assumes $A \in$ *fc-mats*
and *hermitian A*
and $i <$ *proj-meas-size* (*make-pm A*)
shows *meas-outcome-val* (*proj-meas-outcomes* (*make-pm A*) *i*) $\in$ *spectrum A*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-pm-spectrum*:
assumes $A \in$ *fc-mats*
and *hermitian A*
and *make-pm A* = (*p*,*M*)
shows (λi. *meas-outcome-val* (*proj-meas-outcomes* (*make-pm A*) *i*)) ' {..< *p*} = *spectrum A*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-size*:
assumes *hermitian A*
and $A \in$ *fc-mats*
and *make-pm A* = (*p*, *M*)
shows *p* = *card* (*spectrum A*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-size'*:
assumes *hermitian A*
and $A \in$ *fc-mats*
shows *proj-meas-size* (*make-pm A*) = *card* (*spectrum A*) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *make-pm-projectors'*:
assumes *hermitian A*
and $A \in$ *fc-mats*
and *a*<*card* (*spectrum A*)
shows *projector* (*meas-outcome-prj* (*proj-meas-outcomes* (*make-pm A*) *a*))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *meas-outcome-prj-carrier*:
assumes *hermitian A*
and $A \in$ *fc-mats*
and *a*<*card* (*spectrum A*)
shows *meas-outcome-prj* (*proj-meas-outcomes* (*make-pm A*) *a*) $\in$ *fc-mats*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *meas-outcome-prj-trace-real*:
assumes *hermitian A*
and *density-operator R*
and $R \in$ *fc-mats*
and $A \in$ *fc-mats*
and *a*<*card* (*spectrum A*)
shows *Re* (*Complex-Matrix.trace* (*R* ∗ *meas-outcome-prj* (*proj-meas-outcomes* (*make-pm A*) *a*))) =
Complex-Matrix.trace (*R* ∗ *meas-outcome-prj* (*proj-meas-outcomes* (*make-pm A*)

a))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-over-spectrum*:
assumes $A \in$ *fc-mats*
and *hermitian A*
and *make-pm A* = (*p*, *M*)
shows ($\sum$ $y \in$ *spectrum A*. *f y*) = ($\sum$ $i < p$. *f* (*meas-outcome-val* (*M i*)))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-over-spectrum′*:
assumes $A \in$ *fc-mats*
and *hermitian A*
and *make-pm A* = (*p*, *M*)
shows ($\sum$ $y \in \{Re\ x | x.\ x \in$ *spectrum A*$\}$. *f y*) = ($\sum$ $i < p$. *f* (*meas-outcome-val* (*M i*)))
⟨*proof*⟩

When a matrix A is decomposed into a projective measurement $\{(\lambda_a, \pi_a)\}$, it can be recovered by the formula $A = \sum \lambda_a \pi_a$.

lemma (**in** *cpx-sq-mat*) *make-pm-sum*:
assumes $A \in$ *fc-mats*
and *hermitian A*
and *make-pm A* = (*p*, *M*)
shows *sum-mat* (λi. (*meas-outcome-val* (*M i*)) $\cdot_m$ *meas-outcome-prj* (*M i*)) {..< *p*} = *A*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *trace-hermitian-pos-real*:
fixes $f :: 'a \Rightarrow$ *real*
assumes *hermitian A*
and *Complex-Matrix.positive R*
and $A \in$ *fc-mats*
and $R \in$ *fc-mats*
shows *Complex-Matrix.trace* (*R* ∗ *A*) =
Re (*Complex-Matrix.trace* (*R* ∗ *A*))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *hermitian-Re-spectrum*:
assumes *hermitian A*
and $A \in$ *fc-mats*
and $\{Re\ x\ |x.\ x \in$ *spectrum A*$\}$ = {*a*,*b*}
shows *spectrum A* = {*complex-of-real a*, *complex-of-real b*}
⟨*proof*⟩

3.2.5 Similar properties for eigenvalues rather than spectrum indices

In this part we go the other way round: given an eigenvalue of A, `spectrum_to_pm_idx` permits to retrieve its index in the associated projective measurement

definition (**in** *cpx-sq-mat*) *spectrum-to-pm-idx*
  where *spectrum-to-pm-idx A x* = (*let* (*B*,*U*,-) = *unitary-schur-decomposition A*
(*eigvals A*) *in*
    diag-el-to-idx B x)

lemma (**in** *cpx-sq-mat*) *spectrum-to-pm-idx-bij*:
assumes *hermitian A*
  and *A*∈ *fc-mats*
shows *bij-betw* (*spectrum-to-pm-idx A*) (*spectrum A*) {..< *card* (*spectrum A*)}
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-to-pm-idx-lt-card*:
assumes *A*∈ *fc-mats*
  and *hermitian A*
and *a*∈ *spectrum A*
shows *spectrum-to-pm-idx A a* < *card* (*spectrum A*) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-to-pm-idx-inj*:
assumes *hermitian A*
  and *A*∈ *fc-mats*
shows *inj-on* (*spectrum-to-pm-idx A*) (*spectrum A*) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-meas-outcome-val-inv*:
assumes *A*∈ *fc-mats*
  and *hermitian A*
and *make-pm A* = (*p*,*M*)
and *i* < *p*
shows *spectrum-to-pm-idx A* (*meas-outcome-val* (*M i*)) = *i*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *meas-outcome-val-spectrum-inv*:
  assumes *A*∈ *fc-mats*
  and *hermitian A*
and *x*∈ *spectrum A*
and *make-pm A* = (*p*,*M*)
shows *meas-outcome-val* (*M* (*spectrum-to-pm-idx A x*)) = *x*
⟨*proof*⟩

definition (**in** *cpx-sq-mat*) *eigen-projector* **where**
eigen-projector A a =
  meas-outcome-prj ((*proj-meas-outcomes* (*make-pm A*)) (*spectrum-to-pm-idx A a*))

lemma (**in** *cpx-sq-mat*) *eigen-projector-carrier*:
  assumes *A*∈ *fc-mats*

and $a \in$ *spectrum A*
and *hermitian A*
shows *eigen-projector A a* $\in$ *fc-mats* ⟨*proof*⟩

We obtain the following result, which is similar to `make_pm_sum` but with a sum on the elements of the spectrum of Hermitian matrix A, which is a more standard statement of the spectral decomposition theorem.

lemma (**in** *cpx-sq-mat*) *make-pm-sum'*:
assumes $A \in$ *fc-mats*
and *hermitian A*
shows *sum-mat* ($\lambda a.\ \ a \cdot_m$ (*eigen-projector A a*)) (*spectrum A*) = *A*
⟨*proof*⟩

end

theory *CHSH-Inequality* **imports**
Projective-Measurements

begin

4 The CHSH inequality

The local hidden variable assumption for quantum mechanics was developed to make the case that quantum theory was incomplete. In this part we formalize the CHSH inequality, which provides an upper-bound of a quantity involving expectations in a probability space, and exploit this inequality to show that the local hidden variable assumption does not hold.

4.1 Inequality statement

lemma *chsh-real*:
fixes *A0*::*real*
assumes $|A0 * B1| \leq 1$
and $|A0 * B0| \leq 1$
and $|A1 * B0| \leq 1$
and $|A1 * B1| \leq 1$
shows $|A0 * B1 - A0 * B0 + A1 * B0 + A1 * B1| \leq 2$
⟨*proof*⟩

lemma (**in** *prob-space*) *chsh-expect*:
fixes *A0*::$'a \Rightarrow$ *real*
assumes *AE w in M*. $|A0\ w| \leq 1$
and *AE w in M*. $|A1\ w| \leq 1$

and *AE w in M*. $|B0\ w| \leq 1$
and *AE w in M*. $|B1\ w| \leq 1$
and *integrable M* $(\lambda w.\ A0\ w * B1\ w)$
and *integrable M* $(\lambda w.\ A1\ w * B1\ w)$
and *integrable M* $(\lambda w.\ A1\ w * B0\ w)$
and *integrable M* $(\lambda w.\ A0\ w * B0\ w)$
shows $|$*expectation* $(\lambda w.\ A1\ w * B0\ w)$ + *expectation* $(\lambda w.\ A0\ w * B1\ w)$ +
expectation $(\lambda w.\ A1\ w * B1\ w)$ − *expectation* $(\lambda w.\ A0\ w * B0\ w)| \leq 2$
⟨*proof*⟩

The local hidden variable assumption states that separated quantum measurements are independent. It is standard for this assumption to be stated in a context where the hidden variable admits a density; it is stated here in a more general contest involving expectations, with no assumption on the existence of a density.

definition *pos-rv*:: *'a measure* ⇒ (*'a* ⇒ *real*) ⇒ *bool* **where**
pos-rv M Xr ≡ *Xr* ∈ *borel-measurable M* ∧ (*AE w in M*. (*0*::*real*) ≤ *Xr w*)

definition *prv-sum*:: *'a measure* ⇒ *complex Matrix.mat* ⇒ (*complex* ⇒ *'a* ⇒ *real*)
⇒ *bool* **where**
prv-sum M A Xr ≡ (*AE w in M*. ($\sum$ *a*∈ *spectrum A*. *Xr a w*) = *1*)

definition (**in** *cpx-sq-mat*) *lhv* **where**
lhv M A B R XA XB ≡
prob-space M ∧
(∀ *a* ∈*spectrum A*. *pos-rv M* (*XA a*)) ∧
(*prv-sum M A XA*) ∧
(∀ *b* ∈*spectrum B*. *pos-rv M* (*XB b*)) ∧
(*prv-sum M B XB*) ∧
(∀ *a* ∈*spectrum A* . ∀ *b* ∈ *spectrum B*.
(*integrable M* (*λw. XA a w* ∗ *XB b w*)) ∧
$integral^L$ *M* (*λw. XA a w* ∗ *XB b w*) =
Re (*Complex-Matrix.trace*(*eigen-projector A a* ∗ (*eigen-projector B b*) ∗ *R*)))

lemma (**in** *cpx-sq-mat*) *lhv-posl*:
assumes *lhv M A B R XA XB*
shows *AE w in M*. (∀ *a* ∈ *spectrum A*. *0* ≤ *XA a w*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-lt1-l*:
assumes *lhv M A B R XA XB*
shows *AE w in M*. (∀ *a* ∈ *spectrum A*. *XA a w* ≤ *1*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-posr*:
assumes *lhv M A B R XA XB*
shows *AE w in M*. (∀ *b* ∈ *spectrum B*. *0* ≤ *XB b w*)

⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-lt1-r*:
assumes *lhv M A B R XA XB*
shows *AE w in M*. (∀ *a* ∈ *spectrum B*. *XB a w* ≤ *1*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-AE-propl*:
assumes *lhv M A B R XA XB*
shows *AE w in M*. (∀ *a* ∈ *spectrum A*. *0* ≤ *XA a w* ∧ *XA a w* ≤ *1*) ∧ (∑ *a*∈ *spectrum A*. *XA a w*) = *1*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-AE-propr*:
assumes *lhv M A B R XA XB*
shows *AE w in M*. (∀ *a* ∈ *spectrum B*. *0* ≤ *XB a w* ∧ *XB a w* ≤ *1*) ∧ (∑ *a*∈ *spectrum B*. *XB a w*) = *1*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-integral-eq*:
fixes *c*::*real*
assumes *lhv M A B R XA XB*
and *a*∈ *spectrum A*
and *b*∈ *spectrum B*
shows *integral*^{L} *M* (λ*w*. *XA a w* ∗ *XB b w*) =
Re (*Complex-Matrix.trace*(*eigen-projector A a* ∗ (*eigen-projector B b*) ∗ *R*))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-integrable*:
fixes *c*::*real*
assumes *lhv M A B R XA XB*
and *a*∈ *spectrum A*
and *b*∈ *spectrum B*
shows *integrable M* (λ*w*. *XA a w* ∗ *XB b w*) ⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-scal-integrable*:
fixes *c*::*real*
assumes *lhv M A B R XA XB*
and *a*∈ *spectrum A*
and *b*∈ *spectrum B*
shows *integrable M* (λ*w*. *c* ∗*XA a w* ∗ *d* ∗ *XB b w*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-lsum-scal-integrable*:
assumes *lhv M A B R XA XB*
and *a*∈ *spectrum A*
shows *integrable M* (λ*x*. ∑ *b*∈*spectrum B*. *c* ∗ *XA a x* ∗ (*f b*) ∗ *XB b x*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-sum-integrable*:
assumes *lhv M A B R XA XB*
shows *integrable M* (λ*w*. (∑ *a* ∈ *spectrum A*. (∑ *b* ∈ *spectrum B*. *f a* ∗ *XA a w* ∗ *g b* ∗ *XB b w*)))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-abs-1-weighted-suml*:
assumes *lhv M A B R Va Vb*
and {*Re x* |*x*. *x* ∈ *spectrum A*} ≠ {}
and {*Re x* |*x*. *x* ∈ *spectrum A*} ⊆ {− *1*, *1*}
and *hermitian A*
and *A*∈ *fc-mats*
shows *AE w in M*. |(∑ *a*∈*spectrum A*. *Re a* ∗ *Va a w*)| ≤ *1*
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *spectrum-abs-1-weighted-sumr*:
assumes *lhv M B A R Vb Va*
and {*Re x* |*x*. *x* ∈ *spectrum A*} ≠ {}
and {*Re x* |*x*. *x* ∈ *spectrum A*} ⊆ {− *1*, *1*}
and *hermitian A*
and *A*∈ *fc-mats*
shows *AE w in M*. |(∑ *a*∈*spectrum A*. *Re a* ∗ *Va a w*)| ≤ *1*
⟨*proof*⟩

definition *qt-expect* **where**
qt-expect A Va = (λ*w*. (∑ *a*∈*spectrum A*. *Re a* ∗ *Va a w*))

lemma (**in** *cpx-sq-mat*) *spectr-sum-integrable*:
assumes *lhv M A B R Va Vb*
shows *integrable M* (λ*w*. *qt-expect A Va w* ∗ *qt-expect B Vb w*)
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *lhv-sum-integral′*:
assumes *lhv M A B R XA XB*
shows *integral*^{L} *M* (λ*w*. (∑ *a* ∈ *spectrum A*. *f a* ∗ *XA a w*) ∗ (∑ *b* ∈ *spectrum B*. *g b* ∗ *XB b w*)) =
(∑ *a* ∈ *spectrum A*. *f a* ∗ (∑ *b* ∈ *spectrum B*. *g b* ∗ *integral*^{L} *M* (λ*w*. *XA a w* ∗ *XB b w*)))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-qt-expect-trace*:
assumes *lhv M A B R XA XB*
shows (∑ *a* ∈ *spectrum A*. *f a* ∗ (∑ *b* ∈ *spectrum B*. *g b* ∗ *integral*^{L} *M* (λ*w*. *XA a w* ∗ *XB b w*))) =
(∑ *a* ∈ *spectrum A*. *f a* ∗ (∑ *b* ∈ *spectrum B*. *g b* ∗
Re (*Complex-Matrix.trace*(*eigen-projector A a* ∗ (*eigen-projector B b*) ∗ *R*))))
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-eigen-projector-trace-dist*:

assumes *hermitian B*
and *A*∈ *fc-mats*
and *B*∈ *fc-mats*
and *R*∈ *fc-mats*
shows $(\sum\ b \in spectrum\ B.\ (b *$
Complex-Matrix.trace$(A * (eigen\text{-}projector\ B\ b) * R))) =$ *Complex-Matrix.trace*$(A * B * R)$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-eigen-projector-trace-right*:
assumes *hermitian A*
and *A*∈ *fc-mats*
and *B*∈ *fc-mats*
shows $(\sum\ a \in spectrum\ A.$ *Complex-Matrix.trace* $(a \cdot_m eigen\text{-}projector\ A\ a * B))$
=
Complex-Matrix.trace $(A * B)$
⟨*proof*⟩

lemma (**in** *cpx-sq-mat*) *sum-eigen-projector-trace*:
assumes *hermitian A*
and *hermitian B*
and *A*∈ *fc-mats*
and *B*∈ *fc-mats*
and *R*∈ *fc-mats*
shows $(\sum\ a \in spectrum\ A.\ \ a *\ (\sum\ b \in spectrum\ B.\ (b *$
Complex-Matrix.trace$(eigen\text{-}projector\ A\ a * (eigen\text{-}projector\ B\ b) * R)))) =$
Complex-Matrix.trace$(A * B * R)$
⟨*proof*⟩

We obtain the main result of this part, which relates the quantum expectation value of a joint measurement with an expectation.

lemma (**in** *cpx-sq-mat*) *sum-qt-expect*:
assumes *lhv M A B R XA XB*
and *A*∈ *fc-mats*
and *B*∈ *fc-mats*
and *R*∈ *fc-mats*
and *hermitian A*
and *hermitian B*
shows $integral^L\ M\ (\lambda w.\ (qt\text{-}expect\ A\ XA\ w) * (qt\text{-}expect\ B\ XB\ w)) =$
Re (*Complex-Matrix.trace*$(A * B * R)$)
⟨*proof*⟩

4.2 Properties of specific observables

In this part we consider a specific density operator and specific observables corresponding to joint bipartite measurements. We will compute the quantum expectation value of this system and prove that it violates the CHSH inequality, thus proving that the local hidden variable assumption cannot

hold.

4.2.1 Ket 0, Ket 1 and the corresponding projectors

definition *ket-0*::*complex Matrix.vec* **where**
ket-0 = unit-vec 2 0

lemma *ket-0-dim*:
shows *dim-vec ket-0 = 2* ⟨*proof*⟩

lemma *ket-0-norm*:
shows ‖*ket-0*‖ *= 1* ⟨*proof*⟩

lemma *ket-0-mat*:
shows *ket-vec ket-0 = Matrix.mat-of-cols-list 2* [[*1*, *0*]]
⟨*proof*⟩

definition *ket-1*::*complex Matrix.vec* **where**
ket-1 = unit-vec 2 1

lemma *ket-1-dim*:
shows *dim-vec ket-1 = 2* ⟨*proof*⟩

lemma *ket-1-norm*:
shows ‖*ket-1*‖ *= 1* ⟨*proof*⟩

definition *ket-01*
where *ket-01 = tensor-vec ket-0 ket-1*

lemma *ket-01-dim*:
shows *dim-vec ket-01 = 4* ⟨*proof*⟩

definition *ket-10*
where *ket-10 = tensor-vec ket-1 ket-0*

lemma *ket-10-dim*:
shows *dim-vec ket-10 = 4* ⟨*proof*⟩

We define `ket_psim`, one of the Bell states (or EPR pair).

definition *ket-psim* **where**
ket-psim = 1/(*sqrt 2*) ·$_v$ (*ket-01 − ket-10*)

lemma *ket-psim-dim*:
shows *dim-vec ket-psim = 4* ⟨*proof*⟩

lemma *ket-psim-norm*:
shows ‖*ket-psim*‖ *= 1*
⟨*proof*⟩

`rho_psim` represents the density operator associated with the quantum state `ket_psim`.

definition *rho-psim* **where**
rho-psim = *rank-1-proj ket-psim*

lemma *rho-psim-carrier*:
shows *rho-psim* ∈ *carrier-mat 4 4* ⟨*proof*⟩

lemma *rho-psim-dim-row*:
shows *dim-row rho-psim* = *4* ⟨*proof*⟩

lemma *rho-psim-density*:
shows *density-operator rho-psim* ⟨*proof*⟩

4.2.2 The X and Z matrices and two of their combinations

In this part we prove properties of two standard matrices in quantum theory, X and Z, as well as two of their combinations: $\frac{X+Z}{\sqrt{2}}$ and $\frac{Z-X}{\sqrt{2}}$. Note that all of these matrices are observables, they will be used to violate the CHSH inequality.

lemma *Z-carrier*: **shows** *Z* ∈ *carrier-mat 2 2* ⟨*proof*⟩

lemma *Z-hermitian*:
shows *hermitian Z* ⟨*proof*⟩

lemma *unitary-Z*:
shows *Complex-Matrix.unitary Z*
⟨*proof*⟩

lemma *X-carrier*: **shows** *X* ∈ *carrier-mat 2 2* ⟨*proof*⟩

lemma *X-hermitian*:
shows *hermitian X* ⟨*proof*⟩

lemma *unitary-X*:
shows *Complex-Matrix.unitary X*
⟨*proof*⟩

definition *XpZ*
where *XpZ* = *−1/sqrt*(*2*) ·$_m$ (*X* + *Z*)

lemma *XpZ-carrier*:
shows *XpZ* ∈ *carrier-mat 2 2* ⟨*proof*⟩

lemma *XpZ-hermitian*:
shows *hermitian XpZ*
⟨*proof*⟩

lemma *XpZ-inv*:
XpZ ∗ *XpZ* = *1*$_m$ *2* ⟨*proof*⟩

lemma *unitary-XpZ*:
shows *Complex-Matrix.unitary XpZ*
⟨*proof*⟩

definition *ZmX*
where *ZmX* = *1*/*sqrt*(*2*) ·$_m$ (*Z* − *X*)

lemma *ZmX-carrier*:
shows *ZmX* ∈ *carrier-mat 2 2* ⟨*proof*⟩

lemma *ZmX-hermitian*:
shows *hermitian ZmX*
⟨*proof*⟩

lemma *ZmX-inv*:
ZmX ∗ *ZmX* = *1*$_m$ *2* ⟨*proof*⟩

lemma *unitary-ZmX*:
shows *Complex-Matrix.unitary ZmX*
⟨*proof*⟩

definition *Z-XpZ*
where *Z-XpZ* = *tensor-mat Z XpZ*

lemma *Z-XpZ-carrier*:
shows *Z-XpZ* ∈ *carrier-mat 4 4* ⟨*proof*⟩

definition *X-XpZ*
where *X-XpZ* = *tensor-mat X XpZ*

lemma *X-XpZ-carrier*:
shows *X-XpZ* ∈ *carrier-mat 4 4* ⟨*proof*⟩

definition *Z-ZmX*
where *Z-ZmX* = *tensor-mat Z ZmX*

lemma *Z-ZmX-carrier*:
shows *Z-ZmX* ∈ *carrier-mat 4 4* ⟨*proof*⟩

definition *X-ZmX*
where *X-ZmX* = *tensor-mat X ZmX*

lemma *X-ZmX-carrier*:
shows *X-ZmX* ∈ *carrier-mat 4 4* ⟨*proof*⟩

lemma *X-ZmX-rho-psim*[*simp*]:

shows *Complex-Matrix.trace* (*rho-psim* ∗ *X-ZmX*) = *1*/ (*sqrt 2*)
⟨*proof*⟩

lemma *Z-ZmX-rho-psim*[*simp*]:
shows *Complex-Matrix.trace* (*rho-psim* ∗ *Z-ZmX*) = −*1*/ (*sqrt 2*)
⟨*proof*⟩

lemma *X-XpZ-rho-psim*[*simp*]:
shows *Complex-Matrix.trace* (*rho-psim* ∗ *X-XpZ*) =*1*/ (*sqrt 2*)
⟨*proof*⟩

lemma *Z-XpZ-rho-psim*[*simp*]:
shows *Complex-Matrix.trace* (*rho-psim* ∗ *Z-XpZ*) =*1*/ (*sqrt 2*)
⟨*proof*⟩

definition *Z-I* **where**
Z-I = *tensor-mat Z* (*1*$_m$ *2*)

lemma *Z-I-carrier*:
shows *Z-I* ∈ *carrier-mat 4 4* ⟨*proof*⟩

lemma *Z-I-hermitian*:
shows *hermitian Z-I* ⟨*proof*⟩

lemma *Z-I-unitary*:
shows *unitary Z-I* ⟨*proof*⟩

lemma *Z-I-spectrum*:
shows {*Re x* |*x*. *x* ∈ *spectrum Z-I*} ⊆ {− *1*, *1*} ⟨*proof*⟩

definition *X-I* **where**
X-I = *tensor-mat X* (*1*$_m$ *2*)

lemma *X-I-carrier*:
shows *X-I* ∈ *carrier-mat 4 4* ⟨*proof*⟩

lemma *X-I-hermitian*:
shows *hermitian X-I* ⟨*proof*⟩

lemma *X-I-unitary*:
shows *unitary X-I* ⟨*proof*⟩

lemma *X-I-spectrum*:
shows {*Re x* |*x*. *x* ∈ *spectrum X-I*} ⊆ {− *1*, *1*} ⟨*proof*⟩

definition *I-XpZ* **where**
I-XpZ = *tensor-mat* (*1*$_m$ *2*) *XpZ*

lemma *I-XpZ-carrier*:

shows *I-XpZ* ∈ *carrier-mat 4 4* ⟨*proof*⟩

lemma *I-XpZ-hermitian*:
shows *hermitian I-XpZ* ⟨*proof*⟩

lemma *I-XpZ-unitary*:
shows *unitary I-XpZ* ⟨*proof*⟩

lemma *I-XpZ-spectrum*:
shows {*Re x* |*x*. *x* ∈ *spectrum I-XpZ*} ⊆ {− *1*, *1*} ⟨*proof*⟩

definition *I-ZmX* **where**
I-ZmX = *tensor-mat* (*1*$_m$ *2*) *ZmX*

lemma *I-ZmX-carrier*:
shows *I-ZmX* ∈ *carrier-mat 4 4* ⟨*proof*⟩

lemma *I-ZmX-hermitian*:
shows *hermitian I-ZmX* ⟨*proof*⟩

lemma *I-ZmX-unitary*:
shows *unitary I-ZmX* ⟨*proof*⟩

lemma *I-ZmX-spectrum*:
shows {*Re x* |*x*. *x* ∈ *spectrum I-ZmX*} ⊆ {− *1*, *1*} ⟨*proof*⟩

lemma *X-I-ZmX-eq*:
shows *X-I* ∗ *I-ZmX* = *X-ZmX* ⟨*proof*⟩

lemma *X-I-XpZ-eq*:
shows *X-I* ∗ *I-XpZ* = *X-XpZ* ⟨*proof*⟩

lemma *Z-I-XpZ-eq*:
shows *Z-I* ∗ *I-XpZ* = *Z-XpZ* ⟨*proof*⟩

lemma *Z-I-ZmX-eq*:
shows *Z-I* ∗ *I-ZmX* = *Z-ZmX* ⟨*proof*⟩

4.2.3 No local hidden variable

We show that the local hidden variable hypothesis cannot hold by exhibiting a quantum expectation value that is greater than the upper-bound givven by the CHSH inequality.

locale *bin-cpx* = *cpx-sq-mat* +
assumes *dim4*: *dimR* = *4*

lemma (**in** *bin-cpx*) *X-I-XpZ-trace*:
assumes *lhv M X-I I-XpZ R Vx Vp*
and *R*∈ *fc-mats*

shows *LINT w|M. (qt-expect X-I Vx w) * (qt-expect I-XpZ Vp w) = Re (Complex-Matrix.trace (R * X-XpZ))*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *X-I-XpZ-chsh*:
assumes *lhv M X-I I-XpZ rho-psim Vx Vp*
shows *LINT w|M. (qt-expect X-I Vx w) * (qt-expect I-XpZ Vp w) = 1/sqrt 2*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *Z-I-XpZ-trace*:
assumes *lhv M Z-I I-XpZ R Vz Vp*
and *R∈ fc-mats*
shows *LINT w|M. (qt-expect Z-I Vz w) * (qt-expect I-XpZ Vp w) = Re (Complex-Matrix.trace (R * Z-XpZ))*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *Z-I-XpZ-chsh*:
assumes *lhv M Z-I I-XpZ rho-psim Vz Vp*
shows *LINT w|M. (qt-expect Z-I Vz w) * (qt-expect I-XpZ Vp w) = 1/sqrt 2*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *X-I-ZmX-trace*:
assumes *lhv M X-I I-ZmX R Vx Vp*
and *R∈ fc-mats*
shows *LINT w|M. (qt-expect X-I Vx w) * (qt-expect I-ZmX Vp w) = Re (Complex-Matrix.trace (R * X-ZmX))*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *X-I-ZmX-chsh*:
assumes *lhv M X-I I-ZmX rho-psim Vx Vp*
shows *LINT w|M. (qt-expect X-I Vx w) * (qt-expect I-ZmX Vp w) = 1/sqrt 2*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *Z-I-ZmX-trace*:
assumes *lhv M Z-I I-ZmX R Vz Vp*
and *R∈ fc-mats*
shows *LINT w|M. (qt-expect Z-I Vz w) * (qt-expect I-ZmX Vp w) = Re (Complex-Matrix.trace (R * Z-ZmX))*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *Z-I-ZmX-chsh*:
assumes *lhv M Z-I I-ZmX rho-psim Vz Vp*
shows *LINT w|M. (qt-expect Z-I Vz w) * (qt-expect I-ZmX Vp w) = −1/sqrt 2*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *chsh-upper-bound*:
assumes *prob-space M*
and *lhv M X-I I-XpZ rho-psim Vx Vp*
and *lhv M Z-I I-XpZ rho-psim Vz Vp*
and *lhv M X-I I-ZmX rho-psim Vx Vm*
and *lhv M Z-I I-ZmX rho-psim Vz Vm*
shows $|$(*LINT w*|*M*. *qt-expect X-I Vx w* ∗ *qt-expect I-ZmX Vm w*) +
(*LINT w*|*M*. *qt-expect Z-I Vz w* ∗ *qt-expect I-XpZ Vp w*) +
(*LINT w*|*M*. *qt-expect X-I Vx w* ∗ *qt-expect I-XpZ Vp w*) −
(*LINT w*|*M*. *qt-expect Z-I Vz w* ∗ *qt-expect I-ZmX Vm w*)$|$
≤ *2*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *quantum-value*:
assumes *lhv M X-I I-XpZ rho-psim Vx Vp*
and *lhv M Z-I I-XpZ rho-psim Vz Vp*
and *lhv M X-I I-ZmX rho-psim Vx Vm*
and *lhv M Z-I I-ZmX rho-psim Vz Vm*
shows $|$(*LINT w*|*M*. *qt-expect X-I Vx w* ∗ *qt-expect I-ZmX Vm w*) +
(*LINT w*|*M*. *qt-expect Z-I Vz w* ∗ *qt-expect I-XpZ Vp w*) +
(*LINT w*|*M*. *qt-expect X-I Vx w* ∗ *qt-expect I-XpZ Vp w*) −
(*LINT w*|*M*. *qt-expect Z-I Vz w* ∗ *qt-expect I-ZmX Vm w*)$|$
= *2*∗ *sqrt 2*
⟨*proof*⟩

lemma (**in** *bin-cpx*) *no-lhv*:
assumes *lhv M X-I I-XpZ rho-psim Vx Vp*
and *lhv M Z-I I-XpZ rho-psim Vz Vp*
and *lhv M X-I I-ZmX rho-psim Vx Vm*
and *lhv M Z-I I-ZmX rho-psim Vz Vm*
shows *False*
⟨*proof*⟩

end