

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

Anthony Bordg, Hanna Lachnitt and Yijun He

March 17, 2025

Contents

1 Basic Results	2
1.1 Basic Set-Theoretic Results	3
1.2 Basic Arithmetic Results	3
1.3 Basic Results on Matrices	5
1.4 Basic Results on Sums	5
1.5 Basic Results Involving the Exponential Function.	5
1.6 Basic Results with Trigonometric Functions.	7
1.6.1 Basic Inequalities	7
1.6.2 Basic Equalities	7
2 Binary Representation of Natural Numbers	7
3 Qubits and Quantum Gates	13
3.1 Qubits	13
3.2 The Hermitian Conjugation	15
3.3 Unitary Matrices and Quantum Gates	16
3.4 Relations Between Complex Conjugation, Hermitian Conjugation, Transposition and Unitarity	17
3.5 The Inner Product	21
3.6 Unitary Matrices and Length-Preservation	25
3.6.1 Unitary Matrices are Length-Preserving	25
3.6.2 Length-Preserving Matrices are Unitary	28
3.7 A Few Well-known Quantum Gates	34
3.8 The Bell States	38
3.9 The Bitwise Inner Product	39
4 Complex Vectors	41
4.1 The Vector Space of Complex Vectors of Dimension n	41

5 Tensor Products	47
5.1 The Kronecker Product of Complex Vectors	47
5.2 The Tensor Product of Complex Matrices	47
6 Further Results on Tensor Products	58
7 Measurement	70
7.1 Measurements with Bell States	82
8 Quantum Entanglement	96
8.1 The Product States and Entangled States of a 2-qubits System	97
9 Quantum Teleportation	99
10 The Deutsch Algorithm	116
11 The Deutsch-Jozsa Algorithm	126
12 The No-Cloning Theorem	165
12.1 The Cauchy-Schwarz Inequality	165
12.2 The No-Cloning Theorem	168
13 Quantum Prisoner's Dilemma	171
13.1 The Set-Up	171
13.2 The Separable Case	182
13.3 The Maximally Entangled Case	184
13.4 The Unfair Strategy Case	189
14 Acknowledgements	191

Abstract

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

1 Basic Results

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

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

1.1 Basic Set-Theoretic Results

```

lemma set-2-atLeast0 [simp]: {0..<2::nat} = {0,1} by auto

lemma set-2: {..<2::nat} = {0,1} by auto

lemma set-4-atLeast0 [simp]:{0..<4::nat} = {0,1,2,3} by auto

lemma set-4: {..<4::nat} = {0,1,2,3} by auto

lemma set-4-disj [simp]:
  fixes i:: nat
  assumes i < 4
  shows i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3
  using assms by auto

lemma set-8-atLeast0 [simp]: {0..<8::nat} = {0,1,2,3,4,5,6,7} by auto

lemma index-is-2 [simp]: ∀ i::nat. i ≠ Suc 0 → i ≠ 3 → 0 < i → i < 4 →
i = 2 by simp

lemma index-sl-four [simp]: ∀ i::nat. i < 4 → i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3
by auto

```

1.2 Basic Arithmetic Results

```

lemma index-div-eq [simp]:
  fixes i::nat
  shows i ∈ {a*b..<(a+1)*b} ==> i div b = a
proof-
  fix i::nat
  assume a:i ∈ {a*b..<(a+1)*b}
  then have i div b ≥ a
  by (metis Suc-eq-plus1 atLeastLessThan-iff le-refl semiring-normalization-rules(7)
split-div')
  moreover have i div b < a+1
  using a by (simp add: less-mult-imp-div-less)
  ultimately show i div b = a by simp
qed

lemma index-mod-eq [simp]:
  fixes i::nat
  shows i ∈ {a*b..<(a+1)*b} ==> i mod b = i - a*b
  by (simp add: modulo-nat-def)

```

```

lemma sqr-of-cmod-of-prod:
  shows (cmod (z1 * z2))2 = (cmod z1)2 * (cmod z2)2
  by (simp add: norm-mult power-mult-distrib)

lemma less-power-add-imp-div-less [simp]:
  fixes i m n:: nat
  assumes i < 2^(m+n)
  shows i div 2^n < 2^m
  using assms by (simp add: less-mult-imp-div-less power-add)

lemma div-mult-mod-eq-minus:
  fixes i j:: nat
  shows (i div 2^n) * 2^n + i mod 2^n - (j div 2^n) * 2^n - j mod 2^n = i - j
  by (simp add: div-mult-mod-eq algebra-simps)

lemma neq-imp-neq-div-or-mod:
  fixes i j:: nat
  assumes i ≠ j
  shows i div 2^n ≠ j div 2^n ∨ i mod 2^n ≠ j mod 2^n
  using assms div-mult-mod-eq-minus
  by (metis add.right-neutral cancel-div-mod-rules(2))

lemma index-one-mat-div-mod:
  assumes i < 2^(m+n) and j < 2^(m+n)
  shows ((1_m(2^m) $$ (i div 2^n, j div 2^n))::complex) * 1_m(2^n) $$ (i mod 2^n, j mod 2^n) = 1_m(2^(m+n)) $$ (i, j)
  proof (cases i = j)
    case True
    then show ?thesis by(simp add: assms)
  next
    case c1:False
    have i div 2^n ≠ j div 2^n ∨ i mod 2^n ≠ j mod 2^n
    using c1 neq-imp-neq-div-or-mod by simp
    then have 1_m (2^m) $$ (i div 2^n, j div 2^n) = 0 ∨ 1_m (2^n) $$ (i mod 2^n, j mod 2^n) = 0
    using assms by simp
    then show ?thesis
    using assms by (simp add: c1)
  qed

lemma sqr-of-sqrt-2 [simp]:
  fixes z:: complex
  shows z * 2 / (complex-of-real (sqrt 2) * complex-of-real (sqrt 2)) = z
  by(metis nonzero-mult-div-cancel-right norm-numeral of-real-numeral of-real-power power2-eq-square
real-norm-def real-sqrt-abs real-sqrt-power zero-neq-numeral)

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

```

```

apply(auto simp add: real-sqrt-divide algebra-simps)
by (metis divide-eq-0-iff nonzero-mult-div-cancel-left sqr-of-sqrt-2)

lemma two-div-sqr-of-cmd-sqrt-two [simp]:
  shows 2 * (cmod (1 / complex-of-real (sqrt 2)))2 = 1
  using cmod-def by (simp add: power-divide)

lemma two-div-two [simp]:
  shows 2 div Suc (Suc 0) = 1 by simp

lemma two-mod-two [simp]:
  shows 2 mod Suc (Suc 0) = 0 by (simp add: numeral-2_eq_2)

lemma three-div-two [simp]:
  shows 3 div Suc (Suc 0) = 1 by (simp add: numeral-3_eq_3)

lemma three-mod-two [simp]:
  shows 3 mod Suc (Suc 0) = 1 by (simp add: mod-Suc numeral-3_eq_3)

```

1.3 Basic Results on Matrices

```

lemma index-matrix-prod [simp]:
  assumes i < dim-row A and j < dim-col B and dim-col A = dim-row B
  shows (A * B) $(i,j) = (\sum k < dim-row B. (A $(i,k)) * (B $(k,j)))
  using assms
  apply(simp add: scalar-prod-def atLeast0LessThan).

```

1.4 Basic Results on Sums

```

lemma sum-insert [simp]:
  assumes x $\notin$ F and finite F
  shows ($\sum y \in insert x F. P y) = ($\sum y \in F. P y) + P x
  using assms insert-def by(simp add: add.commute)

lemma sum-of-index-diff [simp]:
  fixes f :: nat $\Rightarrow$ 'a :: comm-monoid-add
  shows ($\sum i \in \{.. < a+b\}. f(i-a)) = ($\sum i \in \{.. < b\}. f(i))
  proof (induction b)
    case 0
    then show ?case by simp
  next
    case (Suc b)
    then show ?case by simp
  qed

```

1.5 Basic Results Involving the Exponential Function.

```

lemma exp-of-real-cnj:
  fixes x :: real
  shows cnj (exp (i * x)) = exp (- (i * x))

```

```

proof
  show  $Re(\operatorname{cnj}(\exp(i * x))) = Re(\exp(-(i * x)))$ 
    using Re-exp by simp
  show  $Im(\operatorname{cnj}(\exp(i * x))) = Im(\exp(-(i * x)))$ 
    using Im-exp by simp
qed

lemma exp-of-real-cnj2:
  fixes  $x :: \text{real}$ 
  shows  $\operatorname{cnj}(\exp(-(i * x))) = \exp(i * x)$ 
proof
  show  $Re(\operatorname{cnj}(\exp(-(i * x)))) = Re(\exp(i * x))$ 
    using Re-exp by simp
  show  $Im(\operatorname{cnj}(\exp(-(i * x)))) = Im(\exp(i * x))$ 
    using Im-exp by simp
qed

lemma exp-of-half-pi:
  fixes  $x :: \text{real}$ 
  assumes  $x = pi/2$ 
  shows  $\exp(i * \operatorname{complex-of-real} x) = i$ 
  using assms cis-conv-exp cis-pi-half by fastforce

lemma exp-of-minus-half-pi:
  fixes  $x :: \text{real}$ 
  assumes  $x = pi/2$ 
  shows  $\exp(-(i * \operatorname{complex-of-real} x)) = -i$ 
  using assms cis-conv-exp cis-minus-pi-half by fastforce

lemma exp-of-real:
  fixes  $x :: \text{real}$ 
  shows  $\exp(i * x) = \cos x + i * (\sin x)$ 
proof
  show  $Re(\exp(i * x)) = Re((\cos x) + i * (\sin x))$ 
    using Re-exp by simp
  show  $Im(\exp(i * x)) = Im((\cos x) + i * (\sin x))$ 
    using Im-exp by simp
qed

lemma exp-of-real-inv:
  fixes  $x :: \text{real}$ 
  shows  $\exp(-(i * x)) = \cos x - i * (\sin x)$ 
proof
  show  $Re(\exp(-(i * x))) = Re((\cos x) - i * (\sin x))$ 
    using Re-exp by simp
  show  $Im(\exp(-(i * x))) = Im((\cos x) - i * (\sin x))$ 
    using Im-exp by simp
qed

```

1.6 Basic Results with Trigonometric Functions.

1.6.1 Basic Inequalities

```
lemma sin-squared-le-one:
  fixes x:: real
  shows (sin x)2 ≤ 1
  using abs-sin-le-one abs-square-le-1 by blast

lemma cos-squared-le-one:
  fixes x:: real
  shows (cos x)2 ≤ 1
  using abs-cos-le-one abs-square-le-1 by blast
```

1.6.2 Basic Equalities

```
lemma sin-of-quarter-pi:
  fixes x:: real
  assumes x = pi/2
  shows sin (x/2) = (sqrt 2)/2
  by (auto simp add: assms sin-45)
```

```
lemma cos-of-quarter-pi:
  fixes x:: real
  assumes x = pi/2
  shows cos (x/2) = (sqrt 2)/2
  by (auto simp add: assms cos-45)
```

```
end
```

2 Binary Representation of Natural Numbers

```
theory Binary-Nat
```

```
imports
```

```
HOL.Nat
```

```
HOL.List
```

```
Basics
```

```
begin
```

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

```
lemma length-of-bin-rep-aux:
  fixes n m:: nat
  assumes m < 2^n
  shows length (bin-rep-aux n m) = n+1
  using assms
proof(induction n arbitrary: m)
```

```

case 0
then show length (bin-rep-aux 0 m) = 0 + 1 by simp
next
  case (Suc n)
    assume a0: $\bigwedge m. m < 2^n \implies$  length (bin-rep-aux n m) = n + 1 and m <
 $2^{\lceil n \rceil}$ 
    then show length (bin-rep-aux (Suc n) m) = Suc n + 1
      using a0 by simp
qed

lemma bin-rep-aux-neq-nil:
  fixes n m:: nat
  shows bin-rep-aux n m  $\neq []$ 
  using bin-rep-aux.simps by (metis list.distinct(1) old.nat.exhaust)

lemma last-of-bin-rep-aux:
  fixes n m:: nat
  assumes m <  $2^n$  and m  $\geq 0$ 
  shows last (bin-rep-aux n m) = 0
  using assms
proof(induction n arbitrary: m)
  case 0
  assume m <  $2^0$  and m  $\geq 0$ 
  then show last (bin-rep-aux 0 m) = 0 by simp
next
  case (Suc n)
    assume a0: $\bigwedge m. m < 2^n \implies m \geq 0 \implies$  last (bin-rep-aux n m) = 0 and m
 $< 2^{\lceil n \rceil}$ 
    and m  $\geq 0$ 
    then show last (bin-rep-aux (Suc n) m) = 0
      using bin-rep-aux-neq-nil by simp
qed

lemma mod-mod-power-cancel:
  fixes m n p:: nat
  assumes m  $\leq n$ 
  shows p mod  $2^n$  mod  $2^m$  = p mod  $2^m$ 
  using assms by (simp add: dvd-power-le mod-mod-cancel)

lemma bin-rep-aux-index:
  fixes n m i:: nat
  assumes n  $\geq 1$  and m <  $2^n$  and m  $\geq 0$  and i  $\leq n$ 
  shows bin-rep-aux n m ! i = (m mod  $2^{\lceil n-i \rceil}$ ) div  $2^{\lceil (n-1)-i \rceil}$ 
  using assms
proof(induction n arbitrary: m i rule: nat-induct-at-least)
  case base
  then have m <  $2^n$  and i  $\in \{0, 1\}$ 
    by auto
  then show ?case

```

```

by auto
next
  case (Suc n)
    assume a0: $\bigwedge m i. m < 2^n \implies m \geq 0 \implies i \leq n \implies \text{bin\_rep\_aux } n m ! i = m$ 
    mod  $2^{\lceil n-i \rceil}$  div  $2^{\lceil n-1-i \rceil}$ 
    and a1: $m < 2^{\lceil \text{Suc } n \rceil}$  and a2: $i \leq \text{Suc } n$  and a3: $m \geq 0$ 
    then show bin-rep-aux (Suc n) m ! i = m mod  $2^{\lceil \text{Suc } n - i \rceil}$  div  $2^{\lceil \text{Suc } n - 1 - i \rceil}$ 
  proof-
    have bin-rep-aux (Suc n) m = m div  $2^{\lceil n \rceil} \# \text{bin\_rep\_aux } n (m \text{ mod } 2^{\lceil n \rceil})$  by
    simp
    then have f0:bin-rep-aux (Suc n) m ! i = (m div  $2^{\lceil n \rceil} \# \text{bin\_rep\_aux } n (m \text{ mod } 2^{\lceil n \rceil})$ ) ! i by simp
    then have bin-rep-aux (Suc n) m ! i = m div  $2^{\lceil n \rceil}$  if i = 0 using that by simp
    then have f1:bin-rep-aux (Suc n) m ! i = m mod  $2^{\lceil \text{Suc } n - i \rceil}$  div  $2^{\lceil \text{Suc } n - 1 - i \rceil}$  if i = 0
  proof-
    have m mod  $2^{\lceil \text{Suc } n - i \rceil} = m$ 
      using that a1 by (simp add: Suc.preds(2))
    then have m mod  $2^{\lceil \text{Suc } n - i \rceil}$  div  $2^{\lceil \text{Suc } n - 1 - i \rceil} = m \text{ div } 2^{\lceil n \rceil}$ 
      using that by simp
    thus ?thesis by (simp add: that)
  qed
  then have bin-rep-aux (Suc n) m ! i = bin-rep-aux n (m mod  $2^{\lceil n \rceil}$ ) ! (i-1) if
  i ≥ 1
    using that f0 by simp
    then have f2:bin-rep-aux (Suc n) m ! i = ((m mod  $2^{\lceil n \rceil}$ ) mod  $2^{\lceil n - (i-1) \rceil}$ ) div  $2^{\lceil n - 1 - (i-1) \rceil}$  if i ≥ 1
      using that a0 a1 a2 a3 Suc.preds(2) by simp
    then have f3:bin-rep-aux (Suc n) m ! i = ((m mod  $2^{\lceil n \rceil}$ ) mod  $2^{\lceil \text{Suc } n - i \rceil}$ )
    div  $2^{\lceil \text{Suc } n - 1 - i \rceil}$  if i ≥ 1
      using that by simp
    then have bin-rep-aux (Suc n) m ! i = m mod  $2^{\lceil \text{Suc } n - i \rceil}$  div  $2^{\lceil \text{Suc } n - 1 - i \rceil}$  if i ≥ 1
  proof-
    have Suc n - i ≤ n using that by simp
    then have m mod  $2^{\lceil n \rceil}$  mod  $2^{\lceil \text{Suc } n - i \rceil} = m \text{ mod } 2^{\lceil \text{Suc } n - i \rceil}$ 
      using mod-mod-power-cancel[Suc n - i n m] by simp
    thus ?thesis
      using that f3 by simp
  qed
  thus ?thesis using f1 f2
    using linorder-not-less by blast
  qed
qed
lemma bin-rep-aux-coeff:
  fixes n m i:: nat
  assumes m <  $2^{\lceil n \rceil}$  and i ≤ n and m ≥ 0

```

```

shows bin-rep-aux n m ! i = 0 ∨ bin-rep-aux n m ! i = 1
using assms
proof(induction n arbitrary: m i)
  case 0
    assume m < 2^0 and i ≤ 0 and m ≥ 0
    then show bin-rep-aux 0 m ! i = 0 ∨ bin-rep-aux 0 m ! i = 1 by simp
  next
    case (Suc n)
      assume a0: ∀ m i. m < 2^n ⇒ i ≤ n ⇒ m ≥ 0 ⇒ bin-rep-aux n m ! i =
      0 ∨ bin-rep-aux n m ! i = 1
      and a1: m < 2^Suc n and a2: i ≤ Suc n and a3: m ≥ 0
      then show bin-rep-aux (Suc n) m ! i = 0 ∨ bin-rep-aux (Suc n) m ! i = 1
      proof-
        have bin-rep-aux (Suc n) m ! i = (m div 2^n) # bin-rep-aux n (m mod 2^n) !
        i by simp
        moreover have ... = bin-rep-aux n (m mod 2^n) ! (i - 1) if i ≥ 1
          using that by simp
        moreover have m mod 2^n < 2^n by simp
        ultimately have bin-rep-aux (Suc n) m ! i = 0 ∨ bin-rep-aux (Suc n) m ! i
        = 1 if i ≥ 1
          using that a0[of m mod 2^n i-1] a2 by simp
        moreover have m div 2^n = 0 ∨ m div 2^n = 1
          using a1 a3 less-mult-imp-div-less by(simp add: less-2-cases)
        ultimately show ?thesis by (simp add: nth-Cons')
      qed
    qed
  qed

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

lemma length-of-bin-rep:
  fixes n m:: nat
  assumes m < 2^n
  shows length (bin-rep n m) = n
  using assms length-of-bin-rep-aux bin-rep-def by simp

lemma bin-rep-coeff:
  fixes n m i:: nat
  assumes m < 2^n and i < n and m ≥ 0
  shows bin-rep n m ! i = 0 ∨ bin-rep n m ! i = 1
  using assms bin-rep-def bin-rep-aux-coeff length-of-bin-rep by(simp add: nth-butlast)

lemma bin-rep-index:
  fixes n m i:: nat
  assumes n ≥ 1 and m < 2^n and i < n and m ≥ 0
  shows bin-rep n m ! i = (m mod 2^(n-i)) div 2^(n-1-i)
  proof-
    have bin-rep n m ! i = bin-rep-aux n m ! i
    using bin-rep-def length-of-bin-rep nth-butlast assms(3)
  qed

```

```

by (simp add: nth-butlast assms(2))
thus ?thesis
  using assms bin-rep-aux-index by simp
qed

lemma bin-rep-eq:
  fixes n m:: nat
  assumes n ≥ 1 and m ≥ 0 and m < 2^n and m ≥ 0
  shows m = (∑ i<n. bin-rep n m ! i * 2^(n-1-i))

proof-
{
  fix i:: nat
  assume i < n
  then have bin-rep n m ! i * 2^(n-1-i) = (m mod 2^(n-i)) div 2^(n-1-i)
  * 2^(n-1-i)
    using assms bin-rep-index by simp
  moreover have ... = m mod 2^(n-i) - m mod 2^(n-i) mod 2^(n-1-i)
    by (simp add: minus-mod-eq-div-mult)
  moreover have ... = int(m mod 2^(n-i)) - m mod 2^(n-i) mod 2^(n-1-i)

    using mod-less-eq-dividend of-nat-diff by blast
  moreover have ... = int(m mod 2^(n-i)) - m mod 2^(n-1-i)
    using mod-mod-power-cancel[of n-1-i n-i] by (simp add: dvd-power-le
mod-mod-cancel)
  ultimately have bin-rep n m ! i * 2^(n-1-i) = int (m mod 2^(n-i)) - m
mod 2^(n-1-i)
    by presburger
}
then have f0:(∑ i<n. bin-rep n m ! i * 2^(n-1-i)) = (∑ i<n. int (m mod
2^(n-i)) - m mod 2^(n-1-i))
  by auto
thus ?thesis
proof-
  have (∑ i<n. int ((m::nat) mod 2^(n - i)) - (m mod 2^(n - 1 - i))) =
    (∑ i<n. (m mod 2^(n - i))) - (∑ i<n. int (m mod 2^(n - 1 - i)))
    using sum-subtractf[of (λi. (m mod 2^(n-i)))::nat⇒nat (λi. (m mod 2^(n-1-i)))::nat⇒nat
{..<(n::nat)}]
    by auto
  moreover have ... = m mod 2^n + (∑ i∈{1... (m mod 2^(n-i)))} - (∑ i<n-1. int (m mod 2^(n-1-i))) - m mod 2^0
    using sum.atLeast-Suc-atMost sum.lessThan-Suc assms(1)
    by (smt (verit) One-nat-def Suc-le-eq diff-self-eq-0 le-add-diff-inverse lessThan-atLeast0
minus-nat.diff-0
      plus-1-eq-Suc sum.atLeast-Suc-lessThan)
  moreover have ... = m mod 2^n + (∑ i<n-1. m mod 2^(n-i-1)) - (∑ i<n-1. int (m mod 2^(n-1-i))) - m mod 2^0
    apply (auto simp add: sum-of-index-diff[of λi. m mod 2^(n - 1 - i) 1
n-1])
    by (smt (verit) One-nat-def assms(1) le-add-diff-inverse lessThan-atLeast0

```

```

plus-1-eq-Suc sum.cong sum.shift-bounds-Suc-ivl)
  moreover have ... = m mod 2^n - m mod 2^0 by simp
  moreover have ... = m using assms by auto
  ultimately show m = ( $\sum_{i < n. \text{bin\_rep } n m ! i * 2^{n-1-i}}$ )
    using assms f0 by linarith
qed
qed

lemma bin-rep-index-0:
  fixes n m:: nat
  assumes m < 2^n and k > n
  shows (bin-rep k m) ! 0 = 0
proof-
  have m < 2^(k-1)
    using assms by (smt (verit) Suc-diff-1 Suc-leI gr0I le-trans less-or-eq-imp-le
linorder-neqE-nat not-less
one-less-numeral-iff power-strict-increasing semiring-norm(76))
  then have f:m div 2^(k-1) = 0
    by auto
  have k ≥ 1
    using assms(2) by simp
  moreover have bin-rep-aux k m = (m div 2^(k-1)) # (bin-rep-aux (k-1) (m
mod 2^(k-1)))
    using bin-rep-aux.simps(2) by (metis Suc-diff-1 assms(2) diff-0-eq-0 neq0-conv
zero-less-diff)
  moreover have bin-rep k m = butlast ((m div 2^(k-1)) # (bin-rep-aux (k-1)
(m mod 2^(k-1))))
    using bin-rep-def by (simp add: calculation(2))
  moreover have ... = butlast (0 # (bin-rep-aux (k-1) (m mod 2^(k-1))))
    using f by simp
  moreover have ... = 0 # butlast (bin-rep-aux (k-1) (m mod 2^(k-1)))
    by (simp add: bin-rep-aux-neq-nil)
  ultimately show ?thesis
    by simp
qed

lemma bin-rep-index-0-geq:
  fixes n m:: nat
  assumes m ≥ 2^n and m < 2^(n+1)
  shows bin-rep (n+1) m ! 0 = 1
proof-
  have bin-rep (Suc n) m = butlast (bin-rep-aux (Suc n) m)
    using bin-rep-def by simp
  moreover have ... = butlast (1 # (bin-rep-aux n (m mod 2^n)))
    using assms bin-rep-aux-def by simp
  moreover have ... = 1 # butlast (bin-rep-aux n (m mod 2^n))
    by (simp add: bin-rep-aux-neq-nil)
  ultimately show ?thesis
    by (simp add: bin-rep-aux-neq-nil)

```

```
qed
```

```
end
```

3 Qubits and Quantum Gates

```
theory Quantum
```

```
imports
```

```
  Jordan-Normal-Form.Matrix
```

```
  HOL-Library.Nonpos-Ints
```

```
  Basics
```

```
  Binary-Nat
```

```
begin
```

3.1 Qubits

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

```
definition cpx-vec-length :: complex vec  $\Rightarrow$  real ( $\langle \parallel - \parallel \rangle$ ) where  
cpx-vec-length v  $\equiv$  sqrt( $\sum i < \dim\text{-vec } v. (cmod(v \$ i))^2$ )
```

```
lemma cpx-length-of-vec-of-list [simp]:
```

```
   $\| \text{vec-of-list } l \| = \sqrt{\sum i < \text{length } l. (cmod(l ! i))^2}$ 
```

```
  by (auto simp: cpx-vec-length-def vec-of-list-def vec-of-list-index)
```

```
    (metis (no-types, lifting) dim-vec-of-list sum.cong vec-of-list.abs-eq vec-of-list-index)
```

```
lemma norm-vec-index-unit-vec-is-0 [simp]:
```

```
  assumes j < n and j ≠ i
```

```
  shows cmod((unit-vec n i) $ j) = 0
```

```
  using assms by (simp add: unit-vec-def)
```

```
lemma norm-vec-index-unit-vec-is-1 [simp]:
```

```
  assumes j < n and j = i
```

```
  shows cmod((unit-vec n i) $ j) = 1
```

```
proof –
```

```
  have f:(unit-vec n i) $ j = 1
```

```
    using assms by simp
```

```
  thus ?thesis
```

```
    by (simp add: f cmod-def)
```

```
qed
```

```
lemma unit-cpx-vec-length [simp]:
```

```
  assumes i < n
```

```
  shows  $\| \text{unit-vec } n i \| = 1$ 
```

```
proof –
```

```
  have  $(\sum j < n. (cmod((\text{unit-vec } n i) \$ j))^2) = (\sum j < n. \text{if } j = i \text{ then } 1 \text{ else } 0)$ 
```

```
  using norm-vec-index-unit-vec-is-0 norm-vec-index-unit-vec-is-1
```

```
  by (smt (verit) lessThan-iff one-power2 sum.cong zero-power2)
```

```
  also have ... = 1
```

```

using assms by simp
finally have sqrt (∑ j < n. (cmod((unit-vec n i) $ j))^2) = 1
  by simp
thus ?thesis
  using cpx-vec-length-def by simp
qed

lemma smult-vec-length [simp]:
assumes x ≥ 0
shows ‖complex-of-real(x) ·v v‖ = x * ‖v‖
proof –
  have (λi::nat. (cmod (complex-of-real x * v $ i))^2) = (λi::nat. (cmod (v $ i))^2 * x2)
    by (auto simp: norm-mult power-mult-distrib)
  then have (∑ i < dim-vec v. (cmod (complex-of-real x * v $ i))^2) =
    (∑ i < dim-vec v. (cmod (v $ i))^2 * x2) by meson
  moreover have (∑ i < dim-vec v. (cmod (v $ i))^2 * x2) = x2 * (∑ i < dim-vec v.
  (cmod (v $ i))^2)
    by (metis (no-types) mult.commute sum-distrib-right)
  moreover have sqrt(x2 * (∑ i < dim-vec v. (cmod (v $ i))^2)) =
    sqrt(x2) * sqrt (∑ i < dim-vec v. (cmod (v $ i))^2)
  using real-sqrt-mult by blast
  ultimately show ?thesis
  by(simp add: cpx-vec-length-def assms)
qed

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

```

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

```

lemma unit-vec-of-right-length-is-state [simp]:
assumes i < 2n
shows unit-vec (2n) i ∈ {v | n v::complex vec. dim-vec v = 2n ∧ ‖v‖ = 1}
proof –
  have dim-vec (unit-vec (2n) i) = 2n
    by simp
  moreover have ‖unit-vec (2n) i‖ = 1
    using assms by simp
  ultimately show ?thesis
    by simp
qed

```

```

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

```

```

lemma (in state) state-to-state-qbit [simp]:
  shows col v 0 ∈ state-qbit n
  using state-def state-qbit-def by simp

```

3.2 The Hermitian Conjugation

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

```

definition dagger :: complex mat ⇒ complex mat ( $\langle \cdot^\dagger \rangle$ ) where
   $M^\dagger \equiv \text{mat} (\text{dim-col } M) (\text{dim-row } M) (\lambda(i,j). \text{cnj}(M \$\$ (j,i)))$ 

```

We introduce the type of complex square matrices.

```

typedef cpx-sqr-mat = {M | M::complex mat. square-mat M}
proof-
  have square-mat (1m n) for n
    using one-mat-def by simp
    thus ?thesis by blast
  qed

```

```

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

```

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

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

```

lemma dim-row-of-dagger [simp]:
  dim-row (M†) = dim-col M
  using dagger-def by simp

lemma dim-col-of-dagger [simp]:
  dim-col (M†) = dim-row M
  using dagger-def by simp

lemma col-of-dagger [simp]:
  assumes j < dim-row M
  shows col (M†) j = vec (dim-col M) (λi. cnj (M \$\$ (j,i)))
  using assms col-def dagger-def by simp

lemma row-of-dagger [simp]:
  assumes i < dim-col M
  shows row (M†) i = vec (dim-row M) (λj. cnj (M \$\$ (j,i)))
  using assms row-def dagger-def by simp

lemma dagger-of-dagger-is-id:
  fixes M :: complex Matrix.mat
  shows (M†)† = M
proof

```

```

show dim-row (( $M^\dagger$ ) $^\dagger$ ) = dim-row  $M$  by simp
show dim-col (( $M^\dagger$ ) $^\dagger$ ) = dim-col  $M$  by simp
fix  $i j$  assume  $a0:i < \text{dim-row } M$  and  $a1:j < \text{dim-col } M$ 
then show ( $M^\dagger$ ) $^\dagger$  $(i,j) = M $(i,j)
proof-
  show ?thesis
  using dagger-def a0 a1 by auto
qed
qed

lemma dagger-of-sqr-is-sqr [simp]:
  square-mat (( $M::\text{cpx-sqr-mat}$ ) $^\dagger$ )
proof-
  have square-mat  $M$ 
  using cpx-sqr-mat-to-cpx-mat-def Rep-cpx-sqr-mat by simp
  then have dim-row  $M$  = dim-col  $M$  by simp
  then have dim-col ( $M^\dagger$ ) = dim-row ( $M^\dagger$ ) by simp
  thus square-mat ( $M^\dagger$ ) by simp
qed

lemma dagger-of-id-is-id [simp]:
  ( $1_m n$ ) $^\dagger$  =  $1_m n$ 
  using dagger-def one-mat-def by auto

```

3.3 Unitary Matrices and Quantum Gates

```

definition unitary :: complex mat  $\Rightarrow$  bool where
  unitary  $M \equiv (M^\dagger) * M = 1_m (\text{dim-col } M) \wedge M * (M^\dagger) = 1_m (\text{dim-row } M)$ 

lemma id-is-unitary [simp]:
  unitary ( $1_m n$ )
  by (simp add: unitary-def)

locale gate =
  fixes  $n::\text{nat}$  and  $A::\text{complex mat}$ 
  assumes dim-row [simp]: dim-row  $A = 2^n$ 
  and square-mat [simp]: square-mat  $A$ 
  and unitary [simp]: unitary  $A$ 

```

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

```

lemma mat-unitary-mat [intro]:
  assumes unitary  $M$ 
  shows inverts-mat  $M$  ( $M^\dagger$ )
  using assms by (simp add: unitary-def inverts-mat-def)

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

```

```
using assms by (simp add: unitary-def inverts-mat-def)
```

```
lemma (in gate) gate-is-inv:
  invertible-mat A
  using square-mat unitary invertible-mat-def by blast
```

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

```
notation transpose-mat ( $\langle \cdot^t \rangle$ )
```

```
lemma col-tranpose [simp]:
  assumes dim-row M = n and i < n
  shows col (Mt) i = row M i
proof
  show dim-vec (col (Mt) i) = dim-vec (row M i)
    by (simp add: row-def col-def transpose-mat-def)
next
  show  $\bigwedge j. j < \text{dim-vec} (\text{row } M i) \implies \text{col } M^t i \$ j = \text{row } M i \$ j$ 
    using assms by (simp add: transpose-mat-def)
qed
```

```
lemma row-transpose [simp]:
  assumes dim-col M = n and i < n
  shows row (Mt) i = col M i
  using assms by simp
```

```
definition cpx-mat-cnj :: complex mat  $\Rightarrow$  complex mat ( $\langle \cdot^*$ ) where
cpx-mat-cnj M  $\equiv$  mat (dim-row M) (dim-col M) ( $\lambda(i,j). \text{cnj} (M \$\$ (i,j))$ )
```

```
lemma cpx-mat-cnj-id [simp]:
  ( $1_m n$ )* =  $1_m n$ 
  by (auto simp: cpx-mat-cnj-def)
```

```
lemma cpx-mat-cnj-cnj [simp]:
  ( $M^*$ )* = M
  by (auto simp: cpx-mat-cnj-def)
```

```
lemma dim-row-of-cjn-prod [simp]:
  dim-row ((M*) * (N*)) = dim-row M
  by (simp add: cpx-mat-cnj-def)
```

```
lemma dim-col-of-cjn-prod [simp]:
  dim-col ((M*) * (N*)) = dim-col N
  by (simp add: cpx-mat-cnj-def)
```

```
lemma cpx-mat-cnj-prod:
  assumes dim-col M = dim-row N
  shows (M * N)* = (M*) * (N*)
```

```

proof
  show dim-row  $(M * N)^*$  = dim-row  $((M^*) * (N^*))$ 
    by (simp add: cpx-mat-cnj-def)
next
  show dim-col  $((M * N)^*)$  = dim-col  $((M^*) * (N^*))$ 
    by (simp add: cpx-mat-cnj-def)
next
  fix i j::nat
  assume a1:  $i < \text{dim-row } ((M^*) * (N^*))$  and a2:  $j < \text{dim-col } ((M^*) * (N^*))$ 
  then have  $(M * N)^* \$(i,j) = \text{cnj}(\sum k < (\text{dim-row } N). M \$\$(i,k) * N \$\$(k,j))$ 
    using assms cpx-mat-cnj-def index-mat times-mat-def scalar-prod-def row-def
    col-def
  dim-row-of-cjn-prod dim-col-of-cjn-prod
    by simp
  also have ... =  $(\sum k < (\text{dim-row } N). \text{cnj}(M \$\$(i,k)) * \text{cnj}(N \$\$(k,j)))$  by simp
  also have  $((M^*) * (N^*)) \$(i,j) =$ 
     $(\sum k < (\text{dim-row } N). \text{cnj}(M \$\$(i,k)) * \text{cnj}(N \$\$(k,j)))$ 
    using assms a1 a2 cpx-mat-cnj-def index-mat times-mat-def scalar-prod-def
    row-def col-def
    by simp
  finally show  $(M * N)^* \$(i,j) = ((M^*) * (N^*)) \$(i,j)$  by simp
qed

lemma transpose-of-prod:
  fixes M N::complex Matrix.mat
  assumes dim-col M = dim-row N
  shows  $(M * N)^t = N^t * (M^t)$ 
proof
  fix i j::nat
  assume a0:  $i < \text{dim-row } (N^t * (M^t))$  and a1:  $j < \text{dim-col } (N^t * (M^t))$ 
  then have  $(M * N)^t \$(i,j) = (M * N) \$(j,i)$  by auto
  also have ... =  $(\sum k < \text{dim-row } M^t. M \$\$(j,k) * N \$\$(k,i))$ 
    using assms a0 a1 by auto
  also have ... =  $(\sum k < \text{dim-row } M^t. N \$\$(k,i) * M \$\$(j,k))$ 
    by (simp add: semiring-normalization-rules(7))
  also have ... =  $(\sum k < \text{dim-row } M^t. ((N^t) \$\$(i,k)) * (M^t) \$\$(k,j))$ 
    using assms a0 a1 by auto
  finally show  $((M * N)^t) \$(i,j) = (N^t * (M^t)) \$(i,j)$ 
    using assms a0 a1 by auto
next
  show dim-row  $((M * N)^t) = \text{dim-row } (N^t * (M^t))$  by auto
next
  show dim-col  $((M * N)^t) = \text{dim-col } (N^t * (M^t))$  by auto
qed

lemma transpose-cnj-is-dagger [simp]:
   $(M^t)^* = (M^\dagger)$ 
proof
  show f1: dim-row  $((M^t)^*) = \text{dim-row } (M^\dagger)$ 
```

```

    by (simp add: cpx-mat-cnj-def transpose-mat-def dagger-def)
next
  show f2:dim-col ((Mt)*) = dim-col (M†)
    by (simp add: cpx-mat-cnj-def transpose-mat-def dagger-def)
next
  fix i j::nat
  assume i < dim-row M† and j < dim-col M†
  then show Mt* $$ (i, j) = M† $$ (i, j)
    by (simp add: cpx-mat-cnj-def transpose-mat-def dagger-def)
qed

lemma cnj-transpose-is-dagger [simp]:
  (M*)t = (M†)
proof
  show dim-row ((M*)t) = dim-row (M†)
    by (simp add: transpose-mat-def cpx-mat-cnj-def dagger-def)
next
  show dim-col ((M*)t) = dim-col (M†)
    by (simp add: transpose-mat-def cpx-mat-cnj-def dagger-def)
next
  fix i j::nat
  assume i < dim-row M† and j < dim-col M†
  then show M*t $$ (i, j) = M† $$ (i, j)
    by (simp add: transpose-mat-def cpx-mat-cnj-def dagger-def)
qed

lemma dagger-of transpose-is-cnj [simp]:
  (Mt)† = (M*)
  by (metis transpose-transpose transpose-cnj-is-dagger)

lemma dagger-of prod:
  fixes M N::complex Matrix.mat
  assumes dim-col M = dim-row N
  shows (M * N)† = N† * (M†)
proof-
  have (M * N)† = ((M * N)*)t by auto
  also have ... = ((M*) * (N*))t using assms cpx-mat-cnj-prod by auto
  also have ... = (N*)t * ((M*)t) using assms transpose-of-prod
    by (metis cnj-transpose-is-dagger dim-col-of-dagger dim-row-of-dagger index-transpose-mat(2)
      index-transpose-mat(3))
  finally show (M * N)† = N† * (M†) by auto
qed

```

The product of two quantum gates is a quantum gate.

```

lemma prod-of-gate-is-gate:
  assumes gate n G1 and gate n G2
  shows gate n (G1 * G2)
proof
  show dim-row (G1 * G2) = 2^n using assms by (simp add: gate-def)

```

```

next
  show square-mat ( $G_1 * G_2$ )
    using assms gate.dim-row gate.square-mat by simp
next
  show unitary ( $G_1 * G_2$ )
  proof-
    have  $((G_1 * G_2)^\dagger) * (G_1 * G_2) = 1_m (\dim\text{-col} (G_1 * G_2))$ 
  proof-
    have  $f0: G_1 \in \text{carrier-mat} (2^n) (2^n) \wedge G_2 \in \text{carrier-mat} (2^n) (2^n)$ 
       $\wedge G_1^\dagger \in \text{carrier-mat} (2^n) (2^n) \wedge G_2^\dagger \in \text{carrier-mat} (2^n) (2^n)$ 
       $\wedge G_1 * G_2 \in \text{carrier-mat} (2^n) (2^n)$ 
    using assms gate.dim-row gate.square-mat by auto
    have  $((G_1 * G_2)^\dagger) * (G_1 * G_2) = ((G_2^\dagger) * (G_1^\dagger)) * (G_1 * G_2)$ 
      using assms dagger-of-prod gate.dim-row gate.square-mat by simp
    also have ...  $= (G_2^\dagger) * ((G_1^\dagger) * (G_1 * G_2))$ 
      using assms f0 by auto
    also have ...  $= (G_2^\dagger) * (((G_1^\dagger) * G_1) * G_2)$ 
      using assms f0 f0 by auto
    also have ...  $= (G_2^\dagger) * ((1_m (\dim\text{-col} G_1)) * G_2)$ 
      using gate.unitary[of n G1] assms unitary-def[of G1] by simp
    also have ...  $= (G_2^\dagger) * ((1_m (\dim\text{-col} G_2)) * G_2)$ 
      using assms f0 by (metis carrier-matD(2))
    also have ...  $= (G_2^\dagger) * G_2$ 
      using f0 by (metis carrier-matD(2) left-mult-one-mat)
    finally show  $((G_1 * G_2)^\dagger) * (G_1 * G_2) = 1_m (\dim\text{-col} (G_1 * G_2))$ 
      using assms gate.unitary unitary-def by simp
  qed
  moreover have  $(G_1 * G_2) * ((G_1 * G_2)^\dagger) = 1_m (\dim\text{-row} (G_1 * G_2))$ 
    using assms calculation
    by (smt (verit) carrier-matI dim-col-of-dagger dim-row-of-dagger gate.dim-row
      gate.square-mat index-mult-mat(2) index-mult-mat(3)
      mat-mult-left-right-inverse square-mat.elims(2))
    ultimately show ?thesis using unitary-def by simp
  qed
qed

lemma left-inv-of-unitary-transpose [simp]:
  assumes unitary U
  shows  $(U^t)^\dagger * (U^t) = 1_m (\dim\text{-row} U)$ 
  proof -
    have dim-col U = dim-row (( $U^t$ )*) by simp
    then have  $(U * ((U^t)^\dagger))^\star = (U^\star) * (U^t)$ 
      using cpx-mat-cnj-prod cpx-mat-cnj-cnj by presburger
    also have ...  $= (U^t)^\dagger * (U^t)$  by simp
    finally show ?thesis
      using assms by (metis transpose-cnj-is-dagger cpx-mat-cnj-id unitary-def)
  qed

lemma right-inv-of-unitary-transpose [simp]:

```

```

assumes unitary U
shows  $U^t * ((U^t)^\dagger) = 1_m(\dim\text{-}col U)$ 
proof -
have  $\dim\text{-}col ((U^t)^*) = \dim\text{-}row U$  by simp
then have  $U^t * ((U^t)^\dagger) = (((U^t)^* * U)^*)$ 
using cpx-mat-cnj-cnj cpx-mat-cnj-prod dagger-of-transpose-is-cnj by presburger
also have ... =  $(U^\dagger * U)^*$  by simp
finally show ?thesis
using assms by (metis cpx-mat-cnj-id unitary-def)
qed

lemma transpose-of-unitary-is-unitary [simp]:
assumes unitary U
shows unitary  $(U^t)$ 
using unitary-def assms left-inv-of-unitary-transpose right-inv-of-unitary-transpose
by simp

```

3.5 The Inner Product

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

definition ket-vec :: complex vec \Rightarrow complex mat $(\langle \cdot | \cdot \rangle)$ **where**
 $|v\rangle \equiv \text{mat} (\dim\text{-}vec v) 1 (\lambda(i,j). v \$ i)$

lemma ket-vec-index [simp]:
assumes $i < \dim\text{-}vec v$
shows $|v\rangle \$\$ (i,0) = v \$ i$
using assms ket-vec-def by simp

lemma ket-vec-col [simp]:
col $|v\rangle 0 = v$
by (auto simp: col-def ket-vec-def)

lemma smult-ket-vec [simp]:
 $|x \cdot_v v\rangle = x \cdot_m |v\rangle$
by (auto simp: ket-vec-def)

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

declare [[coercion ket-vec]]

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

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

```

lemma row-bra-vec [simp]:
  row (bra-vec v) 0 = vec (dim-vec v) ( $\lambda i. cnj(v \$ i)$ )
  by (auto simp: row-def bra-vec-def cpx-mat-cnj-def row-vec-def)

```

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

```

definition bra :: complex mat  $\Rightarrow$  complex mat ( $\langle\langle - | \rangle\rangle$ ) where
   $\langle v | \equiv mat 1 (dim-row v) (\lambda(i,j). cnj(v \$\$ (j,i)))$ 

```

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

```

lemma bra-bra-vec [simp]:
  bra (ket-vec v) = bra-vec v
  by (auto simp: bra-def ket-vec-def bra-vec-def cpx-mat-cnj-def row-vec-def)

```

```

lemma row-bra [simp]:
  fixes v::complex vec
  shows row ⟨v| 0 = vec (dim-vec v) ( $\lambda i. cnj(v \$ i)$ ) by simp

```

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

```

definition inner-prod :: complex vec  $\Rightarrow$  complex vec  $\Rightarrow$  complex ( $\langle\langle - | - \rangle\rangle$ ) where
  inner-prod u v  $\equiv \sum i \in \{0..< dim-vec v\}. cnj(u \$ i) * (v \$ i)$ 

```

```

lemma inner-prod-with-row-bra-vec [simp]:
  assumes dim-vec u = dim-vec v
  shows ⟨u|v⟩ = row (bra-vec u) 0  $\cdot$  v
  using assms inner-prod-def scalar-prod-def row-bra-vec index-vec
  by (smt (verit) lessThan-atLeast0 lessThan-iff sum.cong)

```

```

lemma inner-prod-with-row-bra-vec-col-ket-vec [simp]:
  assumes dim-vec u = dim-vec v
  shows ⟨u|v⟩ = (row ⟨u| 0)  $\cdot$  (col |v⟩ 0)
  using assms by (simp add: inner-prod-def scalar-prod-def)

```

```

lemma inner-prod-with-times-mat [simp]:
  assumes dim-vec u = dim-vec v
  shows ⟨u|v⟩ = ((⟨u| * |v⟩) \$\$ (0,0))
  using assms inner-prod-with-row-bra-vec-col-ket-vec
  by (simp add: inner-prod-def times-mat-def ket-vec-def bra-def)

```

```

lemma orthogonal-unit-vec [simp]:
  assumes i < n and j < n and i  $\neq$  j
  shows ⟨unit-vec n i|unit-vec n j⟩ = 0
  proof-
    have ⟨unit-vec n i|unit-vec n j⟩ = unit-vec n i  $\cdot$  unit-vec n j
    using assms unit-vec-def inner-prod-def scalar-prod-def
    by (smt (verit) complex-cnj-zero index-unit-vec(3) index-vec inner-prod-with-row-bra-vec
      row-bra-vec
      scalar-prod-right-unit)
    thus ?thesis

```

```

using assms scalar-prod-def unit-vec-def by simp
qed

```

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

```

lemma vec-index-is-linear [simp]:
assumes dim-vec u = dim-vec v and j < dim-vec u
shows (k ·v u + l ·v v) \$ j = k * (u \$ j) + l * (v \$ j)
using assms vec-index-def smult-vec-def plus-vec-def by simp

```

```

lemma inner-prod-is-linear [simp]:
fixes u::complex vec and v::nat  $\Rightarrow$  complex vec and l::nat  $\Rightarrow$  complex
assumes  $\forall i \in \{0, 1\}. \dim\text{-vec } u = \dim\text{-vec } (v i)$ 
shows  $\langle u | l 0 \cdot_v v 0 + l 1 \cdot_v v 1 \rangle = (\sum_{i \leq 1} l i * \langle u | v i \rangle)$ 
proof –
  have f1:dim-vec (l 0 ·v v 0 + l 1 ·v v 1) = dim-vec u
  using assms by simp
  then have  $\langle u | l 0 \cdot_v v 0 + l 1 \cdot_v v 1 \rangle = (\sum_{i \in \{0 .. < \dim\text{-vec } u\}} \text{cnj}(u \$ i) * ((l 0 \cdot_v v 0 + l 1 \cdot_v v 1) \$ i))$ 
  by (simp add: inner-prod-def)
  also have ... =  $(\sum_{i \in \{0 .. < \dim\text{-vec } u\}} \text{cnj}(u \$ i) * (l 0 * v 0 \$ i + l 1 * v 1 \$ i))$ 
  using assms by simp
  also have ... =  $l 0 * (\sum_{i \in \{0 .. < \dim\text{-vec } u\}} \text{cnj}(u \$ i) * (v 0 \$ i)) + l 1 * (\sum_{i \in \{0 .. < \dim\text{-vec } u\}} \text{cnj}(u \$ i) * (v 1 \$ i))$ 
  by (auto simp: algebra-simps)
  (simp add: sum.distrib sum-distrib-left)
  also have ... =  $l 0 * \langle u | v 0 \rangle + l 1 * \langle u | v 1 \rangle$ 
  using assms inner-prod-def by auto
  finally show ?thesis by simp
qed

```

```

lemma inner-prod-cnj:
assumes dim-vec u = dim-vec v
shows  $\langle v | u \rangle = \text{cnj}(\langle u | v \rangle)$ 
by (simp add: assms inner-prod-def algebra-simps)

```

```

lemma inner-prod-with-itself-Im [simp]:
   $\text{Im}(\langle u | u \rangle) = 0$ 
  using inner-prod-cnj by (metis Reals-cnj-iff complex-is-Real-iff)

```

```

lemma inner-prod-with-itself-real [simp]:
   $\langle u | u \rangle \in \mathbb{R}$ 
  using inner-prod-with-itself-Im by (simp add: complex-is-Real-iff)

```

```

lemma inner-prod-with-itself-eq0 [simp]:
assumes u = 0v (dim-vec u)
shows  $\langle u | u \rangle = 0$ 
using assms inner-prod-def zero-vec-def
by (smt (verit) atLeastLessThan-iff complex-cnj-zero index-zero-vec(1) mult-zero-left)

```

sum.neutral)

lemma *inner-prod-with-itself-Re*:

Re ($\langle u|u \rangle$) ≥ 0

proof –

have *Re* ($\langle u|u \rangle$) = ($\sum i < \text{dim-vec } u. \text{Re} (\text{cnj}(u \$ i) * (u \$ i))$)

by (*simp add: inner-prod-def lessThan-atLeast0*)

moreover have ... = ($\sum i < \text{dim-vec } u. (\text{Re} (u \$ i))^2 + (\text{Im} (u \$ i))^2$)

using *complex-mult-cnj*

by (*metis (no-types, lifting) Re-complex-of-real semiring-normalization-rules(7)*)

ultimately show *Re* ($\langle u|u \rangle$) ≥ 0 **by** (*simp add: sum-nonneg*)

qed

lemma *inner-prod-with-itself-nonneg-reals*:

fixes $u::\text{complex vec}$

shows $\langle u|u \rangle \in \text{nonneg-Reals}$

using *inner-prod-with-itself-real inner-prod-with-itself-Re complex-nonneg-Reals-iff*

inner-prod-with-itself-Im **by** *auto*

lemma *inner-prod-with-itself-Re-non0*:

assumes $u \neq 0_v$ (*dim-vec u*)

shows *Re* ($\langle u|u \rangle$) > 0

proof –

obtain i **where** $a1: i < \text{dim-vec } u$ **and** $u \$ i \neq 0$

using *assms zero-vec-def* **by** (*metis dim-vec eq-vecI index-zero-vec(1)*)

then have $f1:\text{Re} (\text{cnj} (u \$ i) * (u \$ i)) > 0$

by (*metis Re-complex-of-real complex-mult-cnj complex-neq-0 mult.commute*)

moreover have $f2:\text{Re} (\langle u|u \rangle) = (\sum i < \text{dim-vec } u. \text{Re} (\text{cnj}(u \$ i) * (u \$ i)))$

using *inner-prod-def* **by** (*simp add: lessThan-atLeast0*)

moreover have $f3:\forall i < \text{dim-vec } u. \text{Re} (\text{cnj}(u \$ i) * (u \$ i)) \geq 0$

using *complex-mult-cnj* **by** *simp*

ultimately show ?*thesis*

using $a1$ *inner-prod-def lessThan-iff*

by (*metis (no-types, lifting) finite-lessThan sum-pos2*)

qed

lemma *inner-prod-with-itself-nonneg-reals-non0*:

assumes $u \neq 0_v$ (*dim-vec u*)

shows $\langle u|u \rangle \neq 0$

using *assms inner-prod-with-itself-Re-non0* **by** *fastforce*

lemma *cpx-vec-length-inner-prod [simp]*:

$\|v\|^2 = \langle v|v \rangle$

proof –

have $\|v\|^2 = (\sum i < \text{dim-vec } v. (\text{cmod} (v \$ i))^2)$

using *cpx-vec-length-def complex-of-real-def*

by (*metis (no-types, lifting) real-sqrt-power real-sqrt-unique sum-nonneg zero-le-power2*)

also have ... = ($\sum i < \text{dim-vec } v. \text{cnj} (v \$ i) * (v \$ i)$)

```

using complex-norm-square mult.commute by (smt (verit) of-real-sum sum.cong)
finally show ?thesis
  using inner-prod-def by (simp add: lessThan-atLeast0)
qed

lemma inner-prod-csqrt [simp]:
  csqrt ⟨v|v⟩ = ‖v‖
  using inner-prod-with-itself-Re inner-prod-with-itself-Im csqrt-of-real-nonneg cpx-vec-length-def
  by (metis (no-types, lifting) Re-complex-of-real cpx-vec-length-inner-prod real-sqrt-ge-0-iff
       real-sqrt-unique sum-nonneg zero-le-power2)

```

3.6 Unitary Matrices and Length-Preservation

3.6.1 Unitary Matrices are Length-Preserving

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

```

lemma dagger-of-ket-is-bra:
  fixes v:: complex vec
  shows (|v⟩)^\dagger = ⟨v|
  by (simp add: bra-def dagger-def ket-vec-def)

lemma bra-mat-on-vec:
  fixes v::complex vec and A::complex mat
  assumes dim-col A = dim-vec v
  shows ⟨A * v| = ⟨v| * (A^\dagger)
proof
  show dim-row ⟨A * v| = dim-row (⟨v| * (A^\dagger))
    by (simp add: bra-def times-mat-def)
next
  show dim-col ⟨A * v| = dim-col (⟨v| * (A^\dagger))
    by (simp add: bra-def times-mat-def)
next
  fix i j::nat
  assume a1:i < dim-row (⟨v| * (A^\dagger)) and a2:j < dim-col (⟨v| * (A^\dagger))
  then have cnj((A * v) $$ (j,0)) = cnj (row A j · v)
    using bra-def times-mat-def ket-vec-col ket-vec-def by simp
  also have f7:... = (∑ i∈{0 ..< dim-vec v}. cnj(v $ i) * cnj(A $$ (j,i)))
    by (simp add: row-def scalar-prod-def mult.commute) (use assms in auto)
  moreover have f8:(row ⟨v| 0) · (col (A^\dagger) j) =
    vec (dim-vec v) (λi. cnj (v $ i)) · vec (dim-col A) (λi. cnj (A $$ (j,i)))
    using a2 by simp
  ultimately have cnj((A * v) $$ (j,0)) = (row ⟨v| 0) · (col (A^\dagger) j)
    using assms scalar-prod-def
    by (smt (verit) dim-vec index-vec lessThan-atLeast0 lessThan-iff sum.cong)
  then have ⟨A * v| $$ (0,j) = (⟨v| * (A^\dagger)) $$ (0,j)
    using bra-def times-mat-def a2 by simp
  thus ⟨A * |v⟩| $$ (i, j) = (⟨v| * (A^\dagger)) $$ (i, j)
    using a1 by (simp add: times-mat-def bra-def)

```

qed

lemma *mat-on-ket*:

fixes *v*:: complex vec **and** *A*:: complex mat
 assumes dim-col *A* = dim-vec *v*
 shows *A* * |*v*⟩ = |col (*A* * *v*) 0⟩
 using assms ket-vec-def **by** auto

lemma *dagger-of-mat-on-ket*:

fixes *v*:: complex vec **and** *A* :: complex mat
 assumes dim-col *A* = dim-vec *v*
 shows (*A* * |*v*⟩)† = ⟨*v*| * (*A*†)
 using assms **by** (metis bra-mat-on-vec dagger-of-ket-is-bra mat-on-ket)

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

lemma *col-fst-is-col* [simp]:

col-fst M = col *M* 0
 by (simp add: col-def col-fst-def)

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

declare

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

lemma *unit-vec-to-col*:

assumes dim-col *A* = *n* **and** *i* < *n*
 shows col *A* *i* = *A* * |unit-vec *n* *i*⟩

proof

show dim-vec (col *A* *i*) = dim-vec (*A* * |unit-vec *n* *i*⟩)
 using col-def times-mat-def **by** simp

next

fix *j*::nat

assume *j* < dim-vec (col-fst (*A* * |unit-vec *n* *i*⟩))

then show col *A* *i* \$ *j* = (*A* * |unit-vec *n* *i*⟩) \$ *j*

using assms times-mat-def ket-vec-def

by (smt (verit) col-fst-is-col dim-col dim-col-mat(1) index-col index-mult-mat(1)
index-mult-mat(2))

 index-row(1) ket-vec-col less-numeral-extra(1) scalar-prod-right-unit)

qed

lemma *mult-ket-vec-is-ket-vec-of-mult*:

fixes *A*::complex mat **and** *v*::complex vec
 assumes dim-col *A* = dim-vec *v*
 shows |*A* * |*v*⟩⟩ = *A* * |*v*⟩
 using assms ket-vec-def
 by (metis One-nat-def col-fst-is-col dim-col dim-col-mat(1) index-mult-mat(3))

```

ket-vec-col less-Suc0
mat-col-eqI)

lemma unitary-is-sq-length-preserving [simp]:
  assumes unitary U and dim-vec v = dim-col U
  shows  $\|U * |v\rangle\|^2 = \|v\|^2$ 
proof -
  have  $\langle U * |v\rangle |U * |v\rangle \rangle = (\langle |v\rangle | * (U^\dagger) * |U * |v\rangle \rangle) \quad \text{by simp}$ 
  using assms(2) bra-mat-on-vec
  by (metis inner-prod-with-times-mat mult-ket-vec-is-ket-vec-of-mult)
  then have  $\langle U * |v\rangle |U * |v\rangle \rangle = (\langle |v\rangle | * (U^\dagger) * (U * |v\rangle)) \quad \text{by simp}$ 
  using assms(2) mult-ket-vec-is-ket-vec-of-mult by simp
  moreover have f1:dim-col  $\langle |v\rangle | = \text{dim-vec } v$ 
  using ket-vec-def bra-def by simp
  moreover have dim-row  $(U^\dagger) = \text{dim-vec } v$ 
  using assms(2) by simp
  ultimately have  $\langle U * |v\rangle |U * |v\rangle \rangle = (\langle |v\rangle | * ((U^\dagger) * U) * |v\rangle) \quad \text{by simp}$ 
  using assoc-mult-mat
  by(smt (verit, ccfv-threshold) carrier-mat-triv dim-row-mat(1) dagger-def ket-vec-def
  mat-carrier times-mat-def)
  then have  $\langle U * |v\rangle |U * |v\rangle \rangle = (\langle |v\rangle | * |v\rangle) \quad \text{by simp}$ 
  using assms f1 unitary-def by simp
  thus ?thesis
  using cpx-vec-length-inner-prod by(metis Re-complex-of-real inner-prod-with-times-mat)
qed

lemma col-ket-vec [simp]:
  assumes dim-col M = 1
  shows  $|col M 0\rangle = M$ 
  using eq-matI assms ket-vec-def by auto

lemma state-col-ket-vec:
  assumes state 1 v
  shows state 1  $|col v 0\rangle$ 
  using assms by (simp add: state-def)

lemma col-ket-vec-index [simp]:
  assumes i < dim-row v
  shows  $|col v 0\rangle \quad (i,0) = v \quad (i,0)$ 
  using assms ket-vec-def by (simp add: col-def)

lemma col-index-of-mat-col [simp]:
  assumes dim-col v = 1 and i < dim-row v
  shows  $col v 0 \quad i = v \quad (i,0)$ 
  using assms by simp

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

```

```

proof –
  have dim-vec (col v 0) = dim-col U
    using assms(2) by simp
  then have ‖col-fst (U * |col v 0⟩)‖2 = ‖col v 0‖2
    using unitary-is-sq-length-preserving[of U col v 0] assms(1) by simp
  thus ?thesis
    using assms(3) by simp
qed

```

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

```

lemma unitary-is-length-preserving-bis [simp]:
  fixes U::complex mat and v::complex mat
  assumes unitary U and dim-row v = dim-col U and dim-col v = 1
  shows ‖col (U * v) 0‖ = ‖col v 0‖
  using assms unitary-is-sq-length-preserving-bis
  by (metis cpx-vec-length-inner-prod inner-prod-csqrt of-real-hom.injectivity)

```

```

lemma unitary-is-length-preserving [simp]:
  fixes U:: complex mat and v:: complex vec
  assumes unitary U and dim-vec v = dim-col U
  shows ‖U * |v⟩‖ = ‖v‖
  using assms unitary-is-sq-length-preserving
  by (metis cpx-vec-length-inner-prod inner-prod-csqrt of-real-hom.injectivity)

```

3.6.2 Length-Preserving Matrices are Unitary

```

lemma inverts-mat-sym:
  fixes A B:: complex mat
  assumes inverts-mat A B and dim-row B = dim-col A and square-mat B
  shows inverts-mat B A
proof–
  define n where d0:n = dim-row B
  have A * B = 1m (dim-row A) using assms(1) inverts-mat-def by auto
  moreover have dim-col B = dim-col (A * B) using times-mat-def by simp
  ultimately have dim-col B = dim-row A by simp
  then have c0:A ∈ carrier-mat n n using assms(2,3) d0 by auto
  have c1:B ∈ carrier-mat n n using assms(3) d0 by auto
  have f0:A * B = 1m n using inverts-mat-def c0 c1 assms(1) by auto
  have f1:det B ≠ 0
  proof
    assume det B = 0
    then have ∃ v. v ∈ carrier-vec n ∧ v ≠ 0v n ∧ B *v v = 0v n
      using det-0-iff-vec-prod-zero assms(3) c1 by blast
    then obtain v where d1:v ∈ carrier-vec n ∧ v ≠ 0v n ∧ B *v v = 0v n by
    auto
    then have d2:dim-vec v = n by simp
    have B * |v⟩ = |0v n⟩
    proof

```

```

show dim-row ( $B * |v\rangle$ ) = dim-row  $|0_v n\rangle$  using ket-vec-def d0 by simp
next
show dim-col ( $B * |v\rangle$ ) = dim-col  $|0_v n\rangle$  using ket-vec-def d0 by simp
next
fix i j assume i < dim-row  $|0_v n\rangle$  and j < dim-col  $|0_v n\rangle$ 
then have f2:i < n ∧ j = 0 using ket-vec-def by simp
moreover have vec (dim-row B) (( $\$$ ) v) = v using d0 d1 by auto
moreover have ( $B *_v v$ )  $\$$  i = ( $\sum ia = 0..<\text{dim-row } B.$  row B i  $\$$  ia * v  $\$$ 
ia)
using d0 d2 f2 by (auto simp add: scalar-prod-def)
ultimately show ( $B * |v\rangle$ )  $\$$  (i, j) =  $|0_v n\rangle$   $\$$  (i, j)
using ket-vec-def d0 d1 times-mat-def mult-mat-vec-def by (auto simp add:
scalar-prod-def)
qed
moreover have  $|v\rangle \in \text{carrier-mat } n 1$  using d2 ket-vec-def by simp
ultimately have ( $A * B$ ) *  $|v\rangle$  = A *  $|0_v n\rangle$  using c0 c1 by simp
then have f3: $|v\rangle$  = A *  $|0_v n\rangle$  using d2 f0 ket-vec-def by auto
have v =  $0_v n$ 
proof
show dim-vec v = dim-vec ( $0_v n$ ) using d2 by simp
next
fix i assume f4:i < dim-vec ( $0_v n$ )
then have  $|v\rangle$   $\$$  (i,0) = v  $\$$  i using d2 ket-vec-def by simp
moreover have (A *  $|0_v n\rangle$ )  $\$$  (i, 0) = 0
using ket-vec-def times-mat-def scalar-prod-def f4 c0 by auto
ultimately show v  $\$$  i =  $0_v n$   $\$$  i using f3 f4 by simp
qed
then show False using d1 by simp
qed
have f5:adj-mat B ∈ carrier-mat n n ∧ B * adj-mat B = det B ·_m 1_m n using
c1 adj-mat by auto
then have c2:((1/det B) ·_m adj-mat B) ∈ carrier-mat n n by simp
have f6:B * ((1/det B) ·_m adj-mat B) = 1_m n using c1 f1 f5 mult-smult-distrib[of
B] by auto
then have A = (A * B) * ((1/det B) ·_m adj-mat B) using c0 c1 c2 by simp
then have A = (1/det B) ·_m adj-mat B using f0 c2 by auto
then show ?thesis using c0 c1 f6 inverts-mat-def by auto
qed

lemma sum-of-unit-vec-length:
fixes i j n:: nat and c:: complex
assumes i < n and j < n and i ≠ j
shows  $\| \text{unit-vec } n i + c \cdot_v \text{unit-vec } n j \| ^2 = 1 + \text{cnj}(c) * c$ 
proof-
define v where d0:v = unit-vec n i + c ·_v unit-vec n j
have  $\forall k < n. v \$ k = (\text{if } k = i \text{ then } 1 \text{ else } (\text{if } k = j \text{ then } c \text{ else } 0))$ 
using d0 assms(1,2,3) by auto
then have  $\forall k < n. \text{cnj } (v \$ k) * v \$ k = (\text{if } k = i \text{ then } 1 \text{ else } 0) + (\text{if } k = j \text{ then } \text{cnj}(c) * c \text{ else } 0)$ 

```

```

using assms(3) by auto
moreover have  $\|v\|^2 = (\sum k = 0..<n. cnj(v \$ k) * v \$ k)$ 
  using d0 assms cpx-vec-length-inner-prod inner-prod-def by simp
  ultimately show ?thesis
    using d0 assms by (auto simp add: sum.distrib)
qed

lemma sum-of-unit-vec-to-col:
assumes dim-col A = n and i < n and j < n
shows col A i + c ·_v col A j = A * |unit-vec n i + c ·_v unit-vec n j)
proof
  show dim-vec (col A i + c ·_v col A j) = dim-vec (col-fst (A * |unit-vec n i + c
·_v unit-vec n j)))
    using assms(1) by auto
next
  fix k assume k < dim-vec (col-fst (A * |unit-vec n i + c ·_v unit-vec n j)))
  then have f0:k < dim-row A using assms(1) by auto
  have (col A i + c ·_v col A j) \$ k = A $$ (k, i) + c * A $$ (k, j)
    using f0 assms(1-3) by auto
  moreover have ( $\sum x < n. A $$ (k, x) * ((if x = i then 1 else 0) + c * (if x = j$ 
then 1 else 0))) =
    ( $\sum x < n. A $$ (k, x) * (if x = i then 1 else 0)) +$ 
    ( $\sum x < n. A $$ (k, x) * c * (if x = j then 1 else 0))$ 
    by (auto simp add: sum.distrib algebra-simps)
  moreover have  $\forall x < n. A $$ (k, x) * (if x = i then 1 else 0) = (if x = i then A$ 
$$ (k, x) else 0)
    by simp
  moreover have  $\forall x < n. A $$ (k, x) * c * (if x = j then 1 else 0) = (if x = j$ 
then A $$ (k, x) * c else 0)
    by simp
  ultimately show (col A i + c ·_v col A j) \$ k = col-fst (A * |unit-vec n i + c ·_v
unit-vec n j)) \$ k
    using f0 assms(1-3) times-mat-def scalar-prod-def ket-vec-def by auto
qed

lemma inner-prod-is-sesquilinear:
fixes u1 u2 v1 v2:: complex vec and c1 c2 c3 c4:: complex and n:: nat
assumes dim-vec u1 = n and dim-vec u2 = n and dim-vec v1 = n and dim-vec
v2 = n
shows  $\langle c1 \cdot_v u1 + c2 \cdot_v u2 | c3 \cdot_v v1 + c4 \cdot_v v2 \rangle = cnj(c1) * c3 * \langle u1 | v1 \rangle +$ 
 $cnj(c2) * c3 * \langle u2 | v1 \rangle +$ 
 $cnj(c1) * c4 * \langle u1 | v2 \rangle + cnj(c2) * c4 * \langle u2 | v2 \rangle$ 
proof-
  have  $\langle c1 \cdot_v u1 + c2 \cdot_v u2 | c3 \cdot_v v1 + c4 \cdot_v v2 \rangle = c3 * \langle c1 \cdot_v u1 + c2 \cdot_v u2 | v1 \rangle$ 
+  $c4 * \langle c1 \cdot_v u1 + c2 \cdot_v u2 | v2 \rangle$ 
  using inner-prod-is-linear[of c1 ·_v u1 + c2 ·_v u2 λi. if i = 0 then v1 else v2
λi. if i = 0 then c3 else c4] assms
  by simp

```

```

also have ... =  $c3 * \text{cnj}(\langle v1 | c1 \cdot_v u1 + c2 \cdot_v u2 \rangle) + c4 * \text{cnj}(\langle v2 | c1 \cdot_v u1 + c2 \cdot_v u2 \rangle)$ 
  using assms inner-prod-cnj[of  $v1 c1 \cdot_v u1 + c2 \cdot_v u2$ ] inner-prod-cnj[of  $v2 c1 \cdot_v u1 + c2 \cdot_v u2$ ]
    by simp
also have ... =  $c3 * \text{cnj}(c1 * \langle v1 | u1 \rangle + c2 * \langle v1 | u2 \rangle) + c4 * \text{cnj}(c1 * \langle v2 | u1 \rangle + c2 * \langle v2 | u2 \rangle)$ 
  using inner-prod-is-linear[of  $v1 \lambda i. \text{if } i = 0 \text{ then } u1 \text{ else } u2$ ] inner-prod-is-linear[of  $v2 \lambda i. \text{if } i = 0 \text{ then } c1 \text{ else } c2$ ]
    inner-prod-is-linear[of  $v2 \lambda i. \text{if } i = 0 \text{ then } u1 \text{ else } u2 \lambda i. \text{if } i = 0 \text{ then } c1 \text{ else } c2$ ] assms
    by simp
also have ... =  $c3 * (\text{cnj}(c1) * \langle u1 | v1 \rangle + \text{cnj}(c2) * \langle u2 | v1 \rangle) + c4 * (\text{cnj}(c1) * \langle u1 | v2 \rangle + \text{cnj}(c2) * \langle u2 | v2 \rangle)$ 
  using inner-prod-cnj[of  $v1 u1$ ] inner-prod-cnj[of  $v1 u2$ ] inner-prod-cnj[of  $v2 u1$ ] inner-prod-cnj[of  $v2 u2$ ] assms
    by simp
finally show ?thesis
  by (auto simp add: algebra-simps)
qed

```

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

```

lemma length-preserving-is-unitary:
  fixes  $U :: \text{complex mat}$ 
  assumes square-mat  $U$  and  $\forall v :: \text{complex vec}. \dim\text{-vec } v = \dim\text{-col } U \longrightarrow \|U * |v\rangle\| = \|v\|$ 
  shows unitary  $U$ 
proof-
  define  $n$  where  $n = \dim\text{-col } U$ 
  then have  $c0: U \in \text{carrier-mat } n n$  using assms(1) by auto
  then have  $c1: U^\dagger \in \text{carrier-mat } n n$  using assms(1) dagger-def by auto
  have  $f0: (U^\dagger) * U = 1_m (\dim\text{-col } U)$ 
  proof
    show  $\dim\text{-row } (U^\dagger * U) = \dim\text{-row } (1_m (\dim\text{-col } U))$  using c0 by simp
  next
    show  $\dim\text{-col } (U^\dagger * U) = \dim\text{-col } (1_m (\dim\text{-col } U))$  using c0 by simp
  next
    fix  $i j$  assume  $i < \dim\text{-row } (1_m (\dim\text{-col } U))$  and  $j < \dim\text{-col } (1_m (\dim\text{-col } U))$ 
    then have  $a0: i < n \wedge j < n$  using c0 by simp
    have  $f1: \bigwedge l. l < n \longrightarrow (\sum k < n. \text{cnj } (U \$\$ (k, l)) * U \$\$ (k, l)) = 1$ 
    proof
      fix  $l$  assume  $a1: l < n$ 
      define  $v :: \text{complex vec}$  where  $d1: v = \text{unit-vec } n l$ 
      have  $\| \text{col } U l \|^2 = (\sum k < n. \text{cnj } (U \$\$ (k, l)) * U \$\$ (k, l))$ 
        using c0 a1 cpx-vec-length-inner-prod inner-prod-def lessThanAtLeast0 by simp
      moreover have  $\| \text{col } U l \|^2 = \|v\|^2$  using c0 d1 a1 assms(2) unit-vec-to-col
    
```

```

by simp
  moreover have  $\|v\|^2 = 1$  using d1 a1 cpx-vec-length-inner-prod by simp
  ultimately show  $(\sum_{k < n. cnj (U \$\$ (k, l)) * U \$\$ (k, l)}) = 1$  by simp
qed
moreover have  $i \neq j \rightarrow (\sum_{k < n. cnj (U \$\$ (k, i)) * U \$\$ (k, j)}) = 0$ 
proof
  assume a2:i  $\neq j$ 
  define v1::complex vec where d1:v1 = unit-vec n i + 1 ·v unit-vec n j
  define v2::complex vec where d2:v2 = unit-vec n i + i ·v unit-vec n j
  have  $\|v1\|^2 = 1 + cnj 1 * 1$  using d1 a0 a2 sum-of-unit-vec-length by blast
  then have  $\|v1\|^2 = 2$ 
  by (metis complex-cnj-one cpx-vec-length-inner-prod mult.left-neutral of-real-eq-iff
      of-real-numeral one-add-one)
  then have  $\|U * |v1\rangle\|^2 = 2$  using c0 d1 assms(2) unit-vec-to-col by simp
  moreover have col U i + 1 ·v col U j = U * |v1⟩
    using c0 d1 a0 sum-of-unit-vec-to-col by blast
  moreover have col U i + 1 ·v col U j = col U i + col U j by simp
  ultimately have  $\langle col U i + col U j | col U i + col U j \rangle = 2$ 
    using cpx-vec-length-inner-prod by (metis of-real-numeral)
  moreover have  $\langle col U i + col U j | col U i + col U j \rangle =$ 
     $\langle col U i | col U i \rangle + \langle col U j | col U i \rangle + \langle col U i | col U j \rangle + \langle col U j | col U j \rangle$ 
    using inner-prod-is-sesquilinear[of col U i dim-row U col U j col U i col U j
      1 1 1 1]
    by simp
  ultimately have f2:(col U j | col U i) + (col U i | col U j) = 0
    using c0 a0 f1 inner-prod-def lessThanAtLeast0 by simp

  have  $\|v2\|^2 = 1 + cnj i * i$  using a0 a2 d2 sum-of-unit-vec-length by simp
  then have  $\|v2\|^2 = 2$ 
    by (metis Re-complex-of-real complex-norm-square mult.commute norm-ii
        numeral-Bit0
        numeral-One numeral-eq-one-iff of-real-numeral one-power2)
  moreover have  $\|U * |v2\rangle\|^2 = \|v2\|^2$  using c0 d2 assms(2) unit-vec-to-col
  by simp
  moreover have  $\langle col U i + i ·_v col U j | col U i + i ·_v col U j \rangle = \|U * |v2\rangle\|^2$ 
    using c0 a0 d2 sum-of-unit-vec-to-col cpx-vec-length-inner-prod by auto
  moreover have  $\langle col U i + i ·_v col U j | col U i + i ·_v col U j \rangle =$ 
     $\langle col U i | col U i \rangle + (-i) * \langle col U j | col U i \rangle + i * \langle col U i | col U j \rangle$ 
    +  $\langle col U j | col U j \rangle$ 
    using inner-prod-is-sesquilinear[of col U i dim-row U col U j col U i col U j
      1 i 1 i]
    by simp
  ultimately have  $\langle col U j | col U i \rangle - \langle col U i | col U j \rangle = 0$ 
    using c0 a0 f1 inner-prod-def lessThanAtLeast0 by auto
  then show  $(\sum_{k < n. cnj (U \$\$ (k, i)) * U \$\$ (k, j)}) = 0$ 
    using c0 a0 f2 lessThanAtLeast0 inner-prod-def by auto
qed

```

```

ultimately show  $(U^\dagger * U) \$(i, j) = 1_m$  (dim-col  $U$ )  $\$(i, j)$ 
  using  $c0$  assms(1) a0 one-mat-def dagger-def by auto
qed
then have  $(U^\dagger) * U = 1_m$  n using  $c0$  by simp
then have  $\text{inverts-mat } (U^\dagger) U$  using  $c1$   $\text{inverts-mat-def}$  by auto
then have  $\text{inverts-mat } U (U^\dagger)$  using  $c0 c1$   $\text{inverts-mat-sym}$  by simp
then have  $U * (U^\dagger) = 1_m$  (dim-row  $U$ ) using  $c0$   $\text{inverts-mat-def}$  by auto
then show ?thesis using  $f0$   $\text{unitary-def}$  by simp
qed

```

```

lemma inner-prod-with-unitary-mat [simp]:
  assumes unitary  $U$  and dim-vec  $u = \text{dim-col } U$  and dim-vec  $v = \text{dim-col } U$ 
  shows  $\langle U * |u\rangle |U * |v\rangle \rangle = \langle u | v \rangle$ 
proof -
have  $f1: \langle U * |u\rangle |U * |v\rangle \rangle = (\langle |u\rangle | * (U^\dagger) * U * |v\rangle) \$(0,0)$ 
  using assms(2-3) bra-mat-on-vec mult-ket-vec-is-ket-vec-of-mult
  by (smt (verit, ccfv-threshold) assoc-mult-mat carrier-mat-triv col-fst-def dim-vec
    dim-col-of-dagger index-mult-mat(2)
    index-mult-mat(3) inner-prod-with-times-mat ket-vec-def mat-carrier)
moreover have  $f2: \langle |u\rangle | \in \text{carrier-mat } 1$  (dim-vec  $v$ )
  using bra-def ket-vec-def assms(2-3) by simp
moreover have  $f3: U^\dagger \in \text{carrier-mat } (\text{dim-col } U)$  (dim-row  $U$ )
  using dagger-def by simp
ultimately have  $\langle U * |u\rangle |U * |v\rangle \rangle = (\langle |u\rangle | * (U^\dagger * U) * |v\rangle) \$(0,0)$ 
  using assms(3) assoc-mult-mat by (metis carrier-mat-triv)
also have ... =  $(\langle |u\rangle | * |v\rangle) \$(0,0)$ 
  using assms(1) unitary-def
  by (simp add: assms(2) bra-def ket-vec-def)
finally show ?thesis
  using assms(2-3) inner-prod-with-times-mat by presburger
qed

```

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

```

lemma unitary-unit-col [simp]:
  assumes unitary  $U$  and dim-col  $U = n$  and  $i < n$ 
  shows  $\|\text{col } U i\| = 1$ 
  using assms unit-vec-to-col unitary-is-length-preserving by simp

lemma unitary-unit-row [simp]:
  assumes unitary  $U$  and dim-row  $U = n$  and  $i < n$ 
  shows  $\|\text{row } U i\| = 1$ 
proof -
have  $\text{row } U i = \text{col } (U^t) i$ 
  using assms(2-3) by simp
thus ?thesis
  using assms transpose-of-unitary-is-unitary unitary-unit-col
  by (metis index-transpose-mat(3))
qed

```

```

lemma orthogonal-col-of-unitary [simp]:
  assumes unitary U and dim-col U = n and i < n and j < n and i ≠ j
  shows ⟨col U i|col U j⟩ = 0
proof -
  have ⟨col U i|col U j⟩ = ⟨U * |unit-vec n i⟩| U * |unit-vec n j⟩⟩
    using assms(2–4) unit-vec-to-col by simp
  also have ... = ⟨unit-vec n i |unit-vec n j⟩
    using assms(1–2) inner-prod-with-unitary-mat index-unit-vec(3) by simp
  finally show ?thesis
    using assms(3–5) by simp
qed

lemma orthogonal-row-of-unitary [simp]:
  fixes U::complex mat
  assumes unitary U and dim-row U = n and i < n and j < n and i ≠ j
  shows ⟨row U i|row U j⟩ = 0
  using assms orthogonal-col-of-unitary transpose-of-unitary-is-unitary col-transpose
  by (metis index-transpose-mat(3))

```

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

```

lemma gate-on-state-is-state [intro, simp]:
  assumes a1:gate n A and a2:state n v
  shows state n (A * v)
proof
  show dim-row (A * v) = 2^n
    using gate-def state-def a1 by simp
next
  show dim-col (A * v) = 1
    using state-def a2 by simp
next
  have square-mat A
    using a1 gate-def by simp
  then have dim-col A = 2^n
    using a1 gate.dim-row by simp
  then have dim-col A = dim-row v
    using a2 state.dim-row by simp
  then have ‖col (A * v) 0‖ = ‖col v 0‖
    using unitary-is-length-preserving-bis assms gate-def state-def by simp
  thus ‖col (A * v) 0‖ = 1
    using a2 state.is-normal by simp
qed

```

3.7 A Few Well-known Quantum Gates

Any unitary operation on n qubits can be implemented exactly by composing single qubits and CNOT-gates (controlled-NOT gates). However, no straightforward method is known to implement these gates in a fashion

which is resistant to errors. But, the Hadamard gate, the phase gate, the CNOT-gate and the $\pi/8$ gate are also universal for quantum computations, i.e. any quantum circuit on n qubits can be approximated to an arbitrary accuracy by using only these gates, and these gates can be implemented in a fault-tolerant way.

We introduce a coercion from real matrices to complex matrices.

```
definition real-to-cpx-mat:: real mat  $\Rightarrow$  complex mat where
  real-to-cpx-mat  $A \equiv \text{mat} (\dim\text{-row } A) (\dim\text{-col } A) (\lambda(i,j). A \mathbin{\$\$} (i,j))$ 
```

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

```
definition Id :: nat  $\Rightarrow$  complex mat where
  Id  $n \equiv 1_m (2^n)$ 
```

```
lemma id-is-gate [simp]:
  gate  $n$  (Id  $n$ )
proof
  show dim-row (Id  $n$ ) =  $2^n$ 
    using Id-def by simp
next
  show square-mat (Id  $n$ )
    using Id-def by simp
next
  show unitary (Id  $n$ )
    by (simp add: Id-def)
qed
```

More interesting: the Pauli matrices.

```
definition X :: complex mat where
  X  $\equiv \text{mat} 2 2 (\lambda(i,j). \text{if } i=j \text{ then } 0 \text{ else } 1)$ 
```

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

```
lemma dagger-of-X [simp]:
   $X^\dagger = X$ 
  using dagger-def by (simp add: X-def cong-mat)
```

```
lemma X-inv [simp]:
   $X * X = 1_m 2$ 
  apply(simp add: X-def times-mat-def one-mat-def)
  apply(rule cong-mat)
  by(auto simp: scalar-prod-def)
```

```
lemma X-is-gate [simp]:
  gate 1 X
```

```

by (simp add: gate-def unitary-def)
      (simp add: X-def)

definition  $Y :: \text{complex mat}$  where

$$Y \equiv \text{mat } 2 \ 2 (\lambda(i,j). \text{ if } i=j \text{ then } 0 \text{ else } (\text{if } i=0 \text{ then } -i \text{ else } i))$$


lemma dagger-of-Y [simp]:

$$Y^\dagger = Y$$

using dagger-def by (simp add: Y-def cong-mat)

lemma Y-inv [simp]:

$$Y * Y = 1_m \ 2$$

apply(simp add: Y-def times-mat-def one-mat-def)
apply(rule cong-mat)
by(auto simp: scalar-prod-def)

lemma Y-is-gate [simp]:

$$\text{gate } 1 \ Y$$

by (simp add: gate-def unitary-def)
      (simp add: Y-def)

definition  $Z :: \text{complex mat}$  where

$$Z \equiv \text{mat } 2 \ 2 (\lambda(i,j). \text{ if } i \neq j \text{ then } 0 \text{ else } (\text{if } i=0 \text{ then } 1 \text{ else } -1))$$


lemma dagger-of-Z [simp]:

$$Z^\dagger = Z$$

using dagger-def by (simp add: Z-def cong-mat)

lemma Z-inv [simp]:

$$Z * Z = 1_m \ 2$$

apply(simp add: Z-def times-mat-def one-mat-def)
apply(rule cong-mat)
by(auto simp: scalar-prod-def)

lemma Z-is-gate [simp]:

$$\text{gate } 1 \ Z$$

by (simp add: gate-def unitary-def)
      (simp add: Z-def)

The Hadamard gate

definition  $H :: \text{complex mat}$  where

$$H \equiv 1/\sqrt(2) \cdot_m (\text{mat } 2 \ 2 (\lambda(i,j). \text{ if } i \neq j \text{ then } 1 \text{ else } (\text{if } i=0 \text{ then } 1 \text{ else } -1)))$$


lemma H-without-scalar-prod:

$$H = \text{mat } 2 \ 2 (\lambda(i,j). \text{ if } i \neq j \text{ then } 1/\sqrt(2) \text{ else } (\text{if } i=0 \text{ then } 1/\sqrt(2) \text{ else } -(1/\sqrt(2))))$$

using cong-mat by (auto simp: H-def)

lemma dagger-of-H [simp]:

```

```

 $H^\dagger = H$ 
using dagger-def by (auto simp: H-def cong-mat)

lemma H-inv [simp]:
 $H * H = 1_m 2$ 
apply(simp add: H-def times-mat-def one-mat-def)
apply(rule cong-mat)
by(auto simp: scalar-prod-def complex-eqI)

lemma H-is-gate [simp]:
 $\text{gate } 1 H$ 
by (simp add: gate-def unitary-def)
(simp add: H-def)

lemma H-values:
fixes i j:: nat
assumes i < dim-row H and j < dim-col H and i ≠ 1 ∨ j ≠ 1
shows H $$ (i,j) = 1/\sqrt{2}
proof-
have i < 2
using assms(1) by (simp add: H-without-scalar-prod less-2-cases)
moreover have j < 2
using assms(2) by (simp add: H-without-scalar-prod less-2-cases)
ultimately show ?thesis
using assms(3) H-without-scalar-prod by (smt (verit) One-nat-def index-mat(1)
less-2-cases old.prod.case)
qed

lemma H-values-right-bottom:
fixes i j:: nat
assumes i = 1 and j = 1
shows H $$ (i,j) = -1/\sqrt{2}
using assms by (simp add: H-without-scalar-prod)

```

The controlled-NOT gate

```

definition CNOT ::complex mat where
CNOT ≡ mat 4 4
 $(\lambda(i,j). \text{if } i=0 \wedge j=0 \text{ then } 1 \text{ else}$ 
 $(\text{if } i=1 \wedge j=1 \text{ then } 1 \text{ else}$ 
 $(\text{if } i=2 \wedge j=3 \text{ then } 1 \text{ else}$ 
 $(\text{if } i=3 \wedge j=2 \text{ then } 1 \text{ else } 0)))$ 

```

```

lemma dagger-of-CNOT [simp]:
 $CNOT^\dagger = CNOT$ 
using dagger-def by (simp add: CNOT-def cong-mat)

lemma CNOT-inv [simp]:
 $CNOT * CNOT = 1_m 4$ 
apply(simp add: CNOT-def times-mat-def one-mat-def)

```

```

apply(rule cong-mat)
by(auto simp: scalar-prod-def)

lemma CNOT-is-gate [simp]:
  gate 2 CNOT
  by (simp add: gate-def unitary-def)
    (simp add: CNOT-def)

```

The phase gate, also known as the S-gate

```

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

```

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

```

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

```

A few relations between the Hadamard gate and the Pauli matrices

```

lemma HXH-is-Z [simp]:
  H * X * H = Z
  apply(simp add: X-def Z-def H-def times-mat-def)
  apply(rule cong-mat)
  by(auto simp add: scalar-prod-def complex-eqI)

lemma HYH-is-minusY [simp]:
  H * Y * H = - Y
  apply(simp add: Y-def H-def times-mat-def)
  apply(rule eq-matI)
  by(auto simp add: scalar-prod-def complex-eqI)

```

```

lemma HZH-is-X [simp]:
  shows H * Z * H = X
  apply(simp add: X-def Z-def H-def times-mat-def)
  apply(rule cong-mat)
  by(auto simp add: scalar-prod-def complex-eqI)

```

3.8 The Bell States

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

```

definition bell00 ::complex mat (⟨|β₀₀⟩⟩) where
  bell00 ≡ 1/sqrt(2) ·ₘ |vec 4 (λi. if i=0 ∨ i=3 then 1 else 0)⟩

```

```

definition bell01 ::complex mat (⟨|β₀₁⟩⟩) where
  bell01 ≡ 1/sqrt(2) ·ₘ |vec 4 (λi. if i=1 ∨ i=2 then 1 else 0)⟩

```

```

definition bell10 ::complex mat (⟨|β₁₀⟩⟩) where
  bell10 ≡ 1/sqrt(2) ·ₘ |vec 4 (λi. if i=0 then 1 else if i=3 then -1 else 0)⟩

```

```

definition bell11 ::complex mat ( $\langle|\beta_{11}\rangle\rangle$ ) where
  bell11  $\equiv$   $1/\sqrt{2} \cdot_m |\text{vec } 4 (\lambda i. \text{ if } i=1 \text{ then } 1 \text{ else if } i=2 \text{ then } -1 \text{ else } 0)\rangle$ 

lemma
  shows bell00-is-state [simp]:state  $\langle|\beta_{00}\rangle$  and bell01-is-state [simp]:state  $\langle|\beta_{01}\rangle$ 
  and
    bell10-is-state [simp]:state  $\langle|\beta_{10}\rangle$  and bell11-is-state [simp]:state  $\langle|\beta_{11}\rangle$ 
  by (auto simp: state-def bell00-def bell01-def bell10-def bell11-def ket-vec-def)
    (auto simp: cpx-vec-length-def Set-Interval.lessThan-atLeast0 cmod-def power2-eq-square)

lemma bell00-index [simp]:
  shows  $|\beta_{00}\rangle \langle(0,0)| = 1/\sqrt{2}$  and  $|\beta_{00}\rangle \langle(1,0)| = 0$  and  $|\beta_{00}\rangle \langle(2,0)| = 0$  and
   $|\beta_{00}\rangle \langle(3,0)| = 1/\sqrt{2}$ 
  by (auto simp: bell00-def ket-vec-def)

lemma bell01-index [simp]:
  shows  $|\beta_{01}\rangle \langle(0,0)| = 0$  and  $|\beta_{01}\rangle \langle(1,0)| = 1/\sqrt{2}$  and  $|\beta_{01}\rangle \langle(2,0)| = 1/\sqrt{2}$  and
   $|\beta_{01}\rangle \langle(3,0)| = 0$ 
  by (auto simp: bell01-def ket-vec-def)

lemma bell10-index [simp]:
  shows  $|\beta_{10}\rangle \langle(0,0)| = 1/\sqrt{2}$  and  $|\beta_{10}\rangle \langle(1,0)| = 0$  and  $|\beta_{10}\rangle \langle(2,0)| = 0$  and
   $|\beta_{10}\rangle \langle(3,0)| = -1/\sqrt{2}$ 
  by (auto simp: bell10-def ket-vec-def)

lemma bell11-index [simp]:
  shows  $|\beta_{11}\rangle \langle(0,0)| = 0$  and  $|\beta_{11}\rangle \langle(1,0)| = 1/\sqrt{2}$  and  $|\beta_{11}\rangle \langle(2,0)| = -1/\sqrt{2}$  and
   $|\beta_{11}\rangle \langle(3,0)| = 0$ 
  by (auto simp: bell11-def ket-vec-def)

```

3.9 The Bitwise Inner Product

```

definition bitwise-inner-prod:: nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat where
  bitwise-inner-prod n i j  $=$   $(\sum k \in \{0..<n\}. (bin\text{-}rep\ n\ i) !\ k * (bin\text{-}rep\ n\ j) !\ k)$ 

abbreviation bip:: nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat ( $\langle\cdot\ \cdot\ \cdot\ \cdot\rangle$ ) where
  bip i n j  $\equiv$  bitwise-inner-prod n i j

lemma bitwise-inner-prod-fst-el-0:
  assumes  $i < 2^n \vee j < 2^n$ 
  shows  $(i \cdot_{Suc\ n} j) = (i \text{ mod } 2^n) \cdot_n (j \text{ mod } 2^n)$ 
proof-
  have bip i (Suc n) j  $=$   $(\sum k \in \{0..<(Suc\ n)\}. (bin\text{-}rep\ (Suc\ n)\ i) !\ k * (bin\text{-}rep\$ 

```

```

(Suc n) j) ! k)
  using bitwise-inner-prod-def by simp
  also have ... = bin-rep (Suc n) i ! 0 * bin-rep (Suc n) j ! 0 +
    ( $\sum_{k \in \{1..<(Suc n)\}}.$  bin-rep (Suc n) i ! k * bin-rep (Suc n) j ! k)
    by (simp add: sum.atLeast-Suc-lessThan)
  also have ... = ( $\sum_{k \in \{1..<(Suc n)\}}.$  bin-rep (Suc n) i ! k * bin-rep (Suc n) j !
    k)
    using bin-rep-index-0[of i n] bin-rep-index-0[of j n] assms by auto
  also have ... = ( $\sum_{k \in \{0..<n\}}.$  bin-rep (Suc n) i !(k+1) * bin-rep (Suc n) j !
    (k+1))
    using sum.shift-bounds-Suc-ivl[of  $\lambda k.$  bin-rep (Suc n) i ! k * bin-rep (Suc n)
    j ! k 0 n]
    by (metis (no-types, lifting) One-nat-def add.commute plus-1-eq-Suc sum.cong)
  finally have bip i (Suc n) j = ( $\sum_{k \in \{0..<n\}}.$  bin-rep (Suc n) i !(k+1) * bin-rep
  (Suc n) j !(k+1))
    by simp
  moreover have  $k \in \{0..n\} \rightarrow$  bin-rep (Suc n) i !(k+1) = bin-rep n (i mod  $2^n$ )
  ! k for k
    using bin-rep-def by (simp add: bin-rep-aux-neq-nil)
  moreover have  $k \in \{0..n\} \rightarrow$  bin-rep (Suc n) j !(k+1) = bin-rep n (j mod  $2^n$ )
  ! k for k
    using bin-rep-def by (simp add: bin-rep-aux-neq-nil)
  ultimately show ?thesis
  using assms bin-rep-index-0 bitwise-inner-prod-def by simp
qed

lemma bitwise-inner-prod-fst-el-is-1:
  fixes n i j :: nat
  assumes i  $\geq 2^n \wedge j \geq 2^n$  and i  $< 2^{n+1} \wedge j < 2^{n+1}$ 
  shows  $(i \cdot_{(n+1)} j) = 1 + ((i \text{ mod } 2^n) \cdot_n (j \text{ mod } 2^n))$ 

proof-
  have bip i (Suc n) j = ( $\sum_{k \in \{0..<(Suc n)\}}.$  bin-rep (Suc n) i ! k * bin-rep (Suc
  n) j ! k)
    using bitwise-inner-prod-def by simp
  also have ... = bin-rep (Suc n) i ! 0 * bin-rep (Suc n) j ! 0 +
    ( $\sum_{k \in \{1..<(Suc n)\}}.$  bin-rep (Suc n) i ! k * bin-rep (Suc n) j ! k)
    by (simp add: sum.atLeast-Suc-lessThan)
  also have ... = 1 + ( $\sum_{k \in \{1..<(Suc n)\}}.$  bin-rep (Suc n) i ! k * bin-rep (Suc
  n) j ! k)
    using bin-rep-index-0-geq[of n i] bin-rep-index-0-geq[of n j] assms by simp
  also have ... = 1 + ( $\sum_{k \in \{0..<n\}}.$  bin-rep (Suc n) i !(k+1) * bin-rep (Suc
  n) j !(k+1))
    using sum.shift-bounds-Suc-ivl[of  $\lambda k.$  (bin-rep (Suc n) i)!k * (bin-rep (Suc n)
  j)!k 0 n]
    by (metis (no-types, lifting) One-nat-def Suc-eq-plus1 sum.cong)
  finally have f0:bip i (Suc n) j = 1 + ( $\sum_{k \in \{0..<n\}}.$  bin-rep (Suc n) i !(k+1)
  * bin-rep (Suc n) j !(k+1))
    by simp
  moreover have  $k \in \{0..n\} \rightarrow$  bin-rep (Suc n) i !(k+1) = bin-rep n (i mod  $2^n$ )

```

```

! k
 $\wedge \text{bin\_rep}(\text{Suc } n) j ! (k+1) = \text{bin\_rep } n (j \bmod 2^n) ! k$  for  $k$ 
using bin-rep-def by(metis Suc-eq-plus1 bin-rep-aux.simps(2) bin-rep-aux-neq-nil
butlast.simps(2) nth-Cons-Suc)
ultimately show ?thesis
using bitwise-inner-prod-def by simp
qed

lemma bitwise-inner-prod-with-zero:
assumes  $m < 2^n$ 
shows  $(0 \cdot_n m) = 0$ 
proof-
have  $(0 \cdot_n m) = (\sum_{j \in \{0..n\}} \text{bin\_rep } n 0 ! j * \text{bin\_rep } n m ! j)$ 
using bitwise-inner-prod-def by simp
moreover have  $(\sum_{j \in \{0..n\}} \text{bin\_rep } n 0 ! j * \text{bin\_rep } n m ! j)$ 
 $= (\sum_{j \in \{0..n\}} 0 * (\text{bin\_rep } n m) ! j)$ 
by (simp add: bin-rep-index)
ultimately show ?thesis
by simp
qed

end

```

4 Complex Vectors

```

theory Complex-Vectors
imports
  Quantum
  VectorSpace.VectorSpace
begin

```

4.1 The Vector Space of Complex Vectors of Dimension n

```

definition module-cpx-vec:: nat  $\Rightarrow$  (complex, complex vec) module where
  module-cpx-vec n  $\equiv$  module-vec TYPE(complex) n

```

```

definition cpx-rng:: complex ring where
  cpx-rng  $\equiv$  (carrier = UNIV, mult = (*), one = 1, zero = 0, add = (+))

```

```

lemma cpx-cring-is-field [simp]:
  field cpx-rng
  apply unfold-locales
  apply (auto intro: right-inverse simp: cpx-rng-def Units-def
  field-simps)
  by (metis add.right-neutral add-diff-cancel-left' add-uminus-conv-diff)

```

```

lemma cpx-abelian-monoid [simp]:
  abelian-monoid cpx-rng

```

```

using cpx-crng-is-field
by (simp add: field-def abelian-group-def cring-def domain-def ring-def)

lemma vecspace-cpx-vec [simp]:
  vectorspace cpx-rng (module-cpx-vec n)
  apply unfold-locales
    apply (auto simp: cpx-rng-def module-cpx-vec-def module-vec-def
  Units-def field-simps)
    apply (auto intro: right-inverse add-inv-exists-vec)
  by (metis add.right-neutral add-diff-cancel-left' add-uminus-conv-diff)

lemma module-cpx-vec [simp]:
  Module.module cpx-rng (module-cpx-vec n)
  using vecspace-cpx-vec by (simp add: vectorspace-def)

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

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

lemma unit-vectors-carrier-vec [simp]:
  unit-vectors n ⊆ carrier-vec n
  using unit-vectors-def by auto

lemma (in Module.module) finsum-over-singleton [simp]:
  assumes f x ∈ carrier M
  shows finsum M f {x} = f x
  using assms by simp

lemma lincomb-over-singleton [simp]:
  assumes x ∈ carrier-vec n and f ∈ {x} → UNIV
  shows module.lincomb (module-cpx-vec n) f {x} = f x ·_v x
  using assms module.lincomb-def module-cpx-vec module-cpx-vec-def module.finsum-over-singleton
  by (smt (verit) module-vec-simps(3) module-vec-simps(4) smult-carrier-vec)

lemma dim-vec-lincomb [simp]:
  assumes finite F and f: F → UNIV and F ⊆ carrier-vec n
  shows dim-vec (module.lincomb (module-cpx-vec n) f F) = n
  using assms
proof(induct F)
  case empty
  show dim-vec (module.lincomb (module-cpx-vec n) f {}) = n
  proof –
    have module.lincomb (module-cpx-vec n) f {} = 0_v n
    using module.lincomb-def abelian-monoid.finsum-empty module-cpx-vec-def
    vectorspace-cpx-vec vectorspace-def
    by (smt (verit) abelian-group-def Module.module-def module-vec-simps(2))
  thus ?thesis by simp

```

```

qed
next
  case (insert x F)
  hence module.lincomb (module-cpx-vec n) f (insert x F) =
    (f x ·v x) ⊕module-cpx-vec n module.lincomb (module-cpx-vec n) f F
    using module-cpx-vec-def module-vec-def module-cpx-vec module.lincomb-insert
    cpx-rng-def insert-subset
    by (smt (verit) Pi-I' UNIV-I Un-insert-right module-vec-simps(4) partial-object.select-convs(1)
      sup-bot.comm-neutral)
    hence dim-vec (module.lincomb (module-cpx-vec n) f (insert x F)) =
      dim-vec (module.lincomb (module-cpx-vec n) f F)
      using index-add-vec by (simp add: module-cpx-vec-def module-vec-simps(1))
    thus dim-vec (module.lincomb (module-cpx-vec n) f (insert x F)) = n
      using insert.hyps(3) insert.preds(2) by simp
qed

lemma lincomb-vec-index [simp]:
  assumes finite F and a2:i < n and F ⊆ carrier-vec n and f: F → UNIV
  shows module.lincomb (module-cpx-vec n) f F $ i = (∑ v∈F. f v * (v $ i))
  using assms
proof(induct F)
  case empty
  then show module.lincomb (module-cpx-vec n) f {} $ i = (∑ v∈{}. f v * v $ i)
    apply auto
    using a2 module.lincomb-def abelian-monoid.finsum-empty module-cpx-vec-def
    by (metis (mono-tags) abelian-group-def index-zero-vec(1) module-cpx-vec Module.module-def module-vec-simps(2))
  next
  case(insert x F)
  have module.lincomb (module-cpx-vec n) f (insert x F) =
    f x ·v x ⊕module-cpx-vec n module.lincomb (module-cpx-vec n) f F
    using module.lincomb-insert module-cpx-vec insert.hyps(1) module-cpx-vec-def
    module-vec-def
    insert.preds(2) insert.hyps(2) insert.preds(3) insert-def
    by (smt (verit) Pi-I' UNIV-I Un-insert-right cpx-rng-def insert-subset module-vec-simps(4)
      partial-object.select-convs(1) sup-bot.comm-neutral)
  then have module.lincomb (module-cpx-vec n) f (insert x F) $ i =
    (f x ·v x) $ i + module.lincomb (module-cpx-vec n) f F $ i
    using index-add-vec(1) a2 dim-vec-lincomb
    by (metis Pi-split-insert-domain insert.hyps(1) insert.preds(2) insert.preds(3)
      insert-subset
      module-cpx-vec-def module-vec-simps(1))
  hence module.lincomb (module-cpx-vec n) f (insert x F) $ i = f x * x $ i +
    (∑ v∈F. f v * v $ i)
    using index-smult-vec a2 insert.preds(2) insert-def insert.hyps(3) by auto
    with insert show module.lincomb (module-cpx-vec n) f (insert x F) $ i =
    (∑ v∈insert x F. f v * v $ i)
    by auto

```

qed

```

lemma unit-vectors-is-lin-indpt [simp]:
  module.lin-indpt cpx-rng (module-cpx-vec n) (unit-vectors n)
proof
  assume module.lin-dep cpx-rng (module-cpx-vec n) (unit-vectors n)
  hence  $\exists A \ a \ v. (\text{finite } A \wedge A \subseteq (\text{unit-vectors } n) \wedge (a \in A \rightarrow \text{UNIV}) \wedge$ 
     $(\text{module.lincomb } (\text{module-cpx-vec } n) \ a \ A = \mathbf{0}_{\text{module-cpx-vec } n}) \wedge (v \in A) \wedge (a$ 
     $v \neq \mathbf{0}_{\text{cpx-rng}})$ 
  using module.lin-dep-def cpx-rng-def module-cpx-vec by (smt (verit) Pi-UNIV
  UNIV-I)
  moreover obtain A and a and v where f1:finite A and f2:A  $\subseteq$  (unit-vectors
  n) and a  $\in A \rightarrow \text{UNIV}$ 
    and f4:module.lincomb (module-cpx-vec n) a A =  $\mathbf{0}_{\text{module-cpx-vec } n}$  and f5:v
     $\in A$  and
      f6:a v  $\neq \mathbf{0}_{\text{cpx-rng}}$ 
      using calculation by blast
      moreover obtain i where f7:v = unit-vec n i and f8:i < n
        using unit-vectors-def calculation by auto
        ultimately have f9:module.lincomb (module-cpx-vec n) a A $ i = ( $\sum_{u \in A} a u$ 
         $* (u \$ i))$ 
        using lincomb-vec-index
        by (smt (verit) carrier-dim-vec index-unit-vec(3) mem-Collect-eq subset-iff
        sum.cong unit-vectors-def)
        moreover have  $\forall u \in A. \forall j < n. u = \text{unit-vec } n j \longrightarrow j \neq i \longrightarrow a u * (u \$ i) = 0$ 
        using unit-vectors-def index-unit-vec by (simp add: f8)
        then have  $(\sum_{u \in A} a u * (u \$ i)) = (\sum_{u \in A. \text{if } u=v \text{ then } a v * v \$ i \text{ else } 0})$ 
        using f2 unit-vectors-def f7 by (smt (verit) mem-Collect-eq subsetCE sum.cong)
        also have ... = a v * (v \$ i)
        using abelian-monoid.finsum-singleton[of cpx-rng v A  $\lambda u \in A. a u * (u \$ i)$ ]
        cpx-abelian-monoid
        f5 f1 cpx-rng-def by simp
        also have ... = a v
        using f7 index-unit-vec f8 by simp
        also have ...  $\neq 0$ 
        using f6 by (simp add: cpx-rng-def)
        finally show False
        using f4 module-cpx-vec-def module-vec-def index-zero-vec f8 f9 by (simp add:
        module-vec-simps(2))
qed

```

```

lemma unit-vectors-is-genset [simp]:
  module.gen-set cpx-rng (module-cpx-vec n) (unit-vectors n)
proof
  show module.span cpx-rng (module-cpx-vec n) (unit-vectors n)  $\subseteq$  carrier (module-cpx-vec
  n)
  using module.span-def dim-vec-lincomb carrier-vec-def cpx-rng-def
  by (smt (verit) Collect-mono index-unit-vec(3) module.span-is-subset2 mod-
  ule-cpx-vec module-cpx-vec-def

```

```

module-vec-simps(3) unit-vectors-def
next
  show carrier (module-cpx-vec n) ⊆ module.span cpx-rng (module-cpx-vec n)
  (unit-vectors n)
  proof
    fix v
    assume a1:v ∈ carrier (module-cpx-vec n)
    define A a lc where A = {unit-vec n i ::complex vec| i::nat. i < n ∧ v $ i ≠
    0} and
      a = (λu∈A. u · v) and lc = module.lincomb (module-cpx-vec n) a A
    then have f1:finite A by simp
    have f2:A ⊆ carrier-vec n
      using carrier-vec-def A-def by auto
    have f3:a ∈ A → UNIV
      using a-def by simp
    then have f4:dim-vec v = dim-vec lc
      using f1 f2 f3 a1 module-cpx-vec-def dim-vec-lincomb lc-def by (simp add:
      module-vec-simps(3))
    then have f5:i < n ⇒ lc $ i = (∑ u∈A. u · v * u $ i) for i
      using lincomb-vec-index lc-def a-def f1 f2 f3 by simp
    then have i < n ⇒ j < n ⇒ j ≠ i ⇒ unit-vec n j · v * unit-vec n j $ i
    = 0 for i j by simp
    then have i < n ⇒ lc $ i = (∑ u∈A. if u = unit-vec n i then v $ i else 0)
    for i
      using a1 A-def f5 scalar-prod-left-unit
      by (smt (verit) f4 carrier-vecI dim-vec-lincomb f1 f2 f3 index-unit-vec(2) lc-def

      mem-Collect-eq mult.right-neutral sum.cong)
    then have i < n ⇒ lc $ i = v $ i for i
      using abelian-monoid.finsum-singleton[of cpx-rng i] A-def cpx-rng-def by
      simp
    then have f6:v = lc
      using eq-vecI f4 dim-vec-lincomb f1 f2 lc-def by auto
    have A ⊆ unit-vectors n
      using A-def unit-vectors-def by auto
    thus v ∈ module.span cpx-rng (module-cpx-vec n) (unit-vectors n)
      using f6 module.span-def[of cpx-rng module-cpx-vec n] lc-def f1 f2 cpx-rng-def
      module-cpx-vec
      by (smt (verit) Pi-I' UNIV-I mem-Collect-eq partial-object.select-convs(1))
    qed
  qed

lemma unit-vectors-is-basis [simp]:
  vectorspace.basis cpx-rng (module-cpx-vec n) (unit-vectors n)
proof –
  fix n
  have unit-vectors n ⊆ carrier (module-cpx-vec n)
    using unit-vectors-def module-cpx-vec-def module-vec-simps(3) by fastforce
  then show ?thesis

```

```

using vectorspace.basis-def unit-vectors-is-lin-indpt unit-vectors-is-genset vec-
space-cpx-vec
by (smt (verit) carrier-dim-vec index-unit-vec(3) mem-Collect-eq module-cpx-vec-def
module-vec-simps(3)
subsetI unit-vectors-def)
qed

lemma state-qbit-is-lincomb [simp]:
state-qbit n =
{module.lincomb (module-cpx-vec (2^n)) a A | a A.
 finite A  $\wedge$  A  $\subseteq$  (unit-vectors (2^n))  $\wedge$  a  $\in$  A  $\rightarrow$  UNIV  $\wedge$  \|module.lincomb
(module-cpx-vec (2^n)) a A\| = 1}
proof
show state-qbit n
 $\subseteq$  {module.lincomb (module-cpx-vec (2^n)) a A | a A.
 finite A  $\wedge$  A  $\subseteq$  unit-vectors (2^n)  $\wedge$  a  $\in$  A  $\rightarrow$  UNIV  $\wedge$  \|module.lincomb
(module-cpx-vec (2^n)) a A\| = 1}
proof
fix v
assume a1:v  $\in$  state-qbit n
then show v  $\in$  {module.lincomb (module-cpx-vec (2^n)) a A | a A.
 finite A  $\wedge$  A  $\subseteq$  unit-vectors (2^n)  $\wedge$  a  $\in$  A  $\rightarrow$  UNIV  $\wedge$  \|module.lincomb
(module-cpx-vec (2^n)) a A\| = 1}
proof -
obtain a and A where finite A and a  $\in$  A  $\rightarrow$  UNIV and A  $\subseteq$  unit-vectors
(2^n) and
v = module.lincomb (module-cpx-vec (2^n)) a A
using a1 state-qbit-def unit-vectors-is-basis vectorspace.basis-def module.span-def

vecspace-cpx-vec module-cpx-vec module-cpx-vec-def module-vec-def car-
rier-vec-def
by (smt (verit) Pi-UNIV UNIV-I mem-Collect-eq module-vec-simps(3))
thus ?thesis
using a1 state-qbit-def by auto
qed
qed
show {module.lincomb (module-cpx-vec (2^n)) a A | a A.
 finite A  $\wedge$  A  $\subseteq$  unit-vectors (2^n)  $\wedge$  a  $\in$  A  $\rightarrow$  UNIV  $\wedge$  \|module.lincomb
(module-cpx-vec (2^n)) a A\| = 1}
 $\subseteq$  state-qbit n
proof
fix v
assume v  $\in$  {module.lincomb (module-cpx-vec (2^n)) a A | a A.
 finite A  $\wedge$  A  $\subseteq$  unit-vectors (2^n)  $\wedge$  a  $\in$  A  $\rightarrow$  UNIV  $\wedge$  \|module.lincomb
(module-cpx-vec (2^n)) a A\| = 1}
then show v  $\in$  state-qbit n
using state-qbit-def dim-vec-lincomb unit-vectors-carrier-vec by (smt (verit)
mem-Collect-eq order-trans)
qed

```

```
qed
```

```
end
```

5 Tensor Products

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

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

5.1 The Kronecker Product of Complex Vectors

```
definition tensor-vec:: complex Matrix.vec ⇒ complex Matrix.vec ⇒ complex Matrix.vec (infixl ⟨⊗⟩ 63)
where tensor-vec u v ≡ vec-of-list (mult.vec-vec-Tensor (*) (list-of-vec u) (list-of-vec v))
```

5.2 The Tensor Product of Complex Matrices

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

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

```
lemma length-mat-to-cols-list [simp]:
  length (mat-to-cols-list A) = dim-col A
  by (simp add: mat-to-cols-list-def)
```

```
lemma length-cols-mat-to-cols-list [simp]:
  assumes j < dim-col A
  shows length [A $$ (i,j) . i <- [0..< dim-row A]] = dim-row A
  using assms by simp
```

```

lemma length-row-mat-to-cols-list [simp]:
  assumes i < dim-row A
  shows length (row (mat-to-cols-list A) i) = dim-col A
  using assms by (simp add: row-def)

lemma length-col-mat-to-cols-list [simp]:
  assumes j < dim-col A
  shows length (col (mat-to-cols-list A) j) = dim-row A
  using assms by (simp add: col-def mat-to-cols-list-def)

lemma mat-to-cols-list-is-not-Nil [simp]:
  assumes dim-col A > 0
  shows mat-to-cols-list A ≠ []
  using assms by (simp add: mat-to-cols-list-def)

```

Link between Matrix.Tensor.row_length and Matrix.dim_row

```

lemma row-length-mat-to-cols-list [simp]:
  assumes dim-col A > 0
  shows mult.row-length (mat-to-cols-list A) = dim-row A
proof -
  have mat-to-cols-list A ≠ [] by (simp add: assms)
  then have mult.row-length (mat-to-cols-list A) = length (hd (mat-to-cols-list A))
    using mult.row-length-def[of 1 (*)]
    by (simp add: ‹⋀xs. Matrix.Tensor.mult 1 (*) ⟹ mult.row-length xs ≡ if xs
      = [] then 0 else length (hd xs)› mult.intro)
  thus ?thesis by (simp add: assms mat-to-cols-list-def upt-conv-Cons)
qed

```

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

```

lemma mat-to-cols-list-is-mat [simp]:
  assumes dim-col A > 0
  shows mat (mult.row-length (mat-to-cols-list A)) (length (mat-to-cols-list A))
  (mat-to-cols-list A)
proof -
  have Ball (set (mat-to-cols-list A)) (Matrix-Legacy.vec (mult.row-length (mat-to-cols-list A)))
    using assms row-length-mat-to-cols-list mat-to-cols-list-def Ball-def set-def vec-def
    by fastforce
  thus ?thesis by (auto simp: mat-def)
qed

```

```

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

```

```

lemma index-mat-of-cols-list [simp]:
  assumes i < nr and j < length cs
  shows mat-of-cols-list nr cs $$ (i,j) = cs ! j ! i
  by (simp add: assms mat-of-cols-list-def)

```

```

lemma mat-to-cols-list-to-mat [simp]:
  mat-of-cols-list (dim-row A) (mat-to-cols-list A) = A
proof
  show f1:dim-row (mat-of-cols-list (dim-row A) (mat-to-cols-list A)) = dim-row
  A
    by (simp add: mat-of-cols-list-def)
next
  show f2:dim-col (mat-of-cols-list (dim-row A) (mat-to-cols-list A)) = dim-col A
    by (simp add: Tensor.mat-of-cols-list-def)
next
  show  $\bigwedge i j. i < \text{dim-row } A \implies j < \text{dim-col } A \implies$ 
    (mat-of-cols-list (dim-row A) (mat-to-cols-list A))  $\$ \$ (i, j) = A \$ \$ (i, j)$ 
    by (simp add: mat-of-cols-list-def mat-to-cols-list-def)
qed

lemma plus-mult-cpx [simp]:
  plus-mult 1 (*) 0 (+) (a-inv cpx-rng)
  apply unfold-locales
  apply (auto intro: cpx-cring-is-field simp: field-simps)
proof -
  show  $\bigwedge x. x + \ominus_{\text{cpx-rng}} x = 0$ 
    using group.r-inv[of cpx-rng] cpx-cring-is-field field-def domain-def cpx-rng-def
    by (metis UNIV-I cring.cring-simprules(17) ordered-semiring-record-simps(1)
         ordered-semiring-record-simps(11) ordered-semiring-record-simps(12))
  show  $\bigwedge x. x + \ominus_{\text{cpx-rng}} x = 0$ 
    using group.r-inv[of cpx-rng] cpx-cring-is-field field-def domain-def cpx-rng-def
    by (metis UNIV-I cring.cring-simprules(17) ordered-semiring-record-simps(1)
         ordered-semiring-record-simps(11) ordered-semiring-record-simps(12))
qed

lemma list-to-mat-to-cols-list [simp]:
  fixes l::complex list list
  assumes mat nr nc l
  shows mat-to-cols-list (mat-of-cols-list nr l) = l
proof -
  have length (mat-to-cols-list (mat-of-cols-list nr l)) = length l
    by (simp add: mat-of-cols-list-def)
  moreover have f1: $\forall j < \text{length } l. \text{length}(l ! j) = \text{mult.row-length } l$ 
    using assms plus-mult.row-length-constant plus-mult-cpx by fastforce
  moreover have  $\bigwedge j. j < \text{length } l \longrightarrow \text{mat-to-cols-list} (\text{mat-of-cols-list nr } l) ! j = l$ 
  ! j
proof
  fix j
  assume a:j < length l
  then have f2:length (mat-to-cols-list (mat-of-cols-list nr l) ! j) = length (l ! j)
    by (metis col-def mat-def vec-def mat-of-cols-list-def assms dim-col-mat(1)
          dim-row-mat(1)
          length-col-mat-to-cols-list nth-mem)

```

```

then have  $\forall i < \text{mult.row-length } l. \text{mat-to-cols-list} (\text{mat-of-cols-list} \text{ nr } l) ! j ! i$ 
 $= l ! j ! i$ 
  using a mat-to-cols-list-def mat-of-cols-list-def f1 by simp
  thus mat-to-cols-list (Tensor.mat-of-cols-list nr l) ! j = l ! j
    using f2 by(simp add: nth-equalityI a f1)
qed
ultimately show ?thesis using nth-equalityI by metis
qed

lemma col-mat-of-cols-list [simp]:
  assumes  $j < \text{length } l$ 
  shows Matrix.col (mat-of-cols-list (length (l ! j)) l) j = vec-of-list (l ! j)
proof -
  define u where u = Matrix.col (mat-of-cols-list (length (l ! j)) l) j
  then have dim-vec u = dim-vec (vec-of-list (l ! j))
    by (auto simp: u-def mat-of-cols-list-def Matrix.col-def vec-of-list-def)
      (metis dim-vec-of-list vec-of-list.abs-eq)
  moreover have  $\forall i < \text{length}(l ! j). u \$ i = \text{vec-of-list} (l ! j) \$ i$ 
    by (simp add: u-def vec-of-list-index mat-of-cols-list-def assms)
  ultimately show ?thesis by(simp add: vec-eq-iff u-def)
qed

definition tensor-mat:: [complex Matrix.mat, complex Matrix.mat]  $\Rightarrow$  complex Matrix.mat (infixl  $\otimes$  63) where
tensor-mat A B  $\equiv$ 
mat-of-cols-list (dim-row A * dim-row B) (mult.Tensor (*) (mat-to-cols-list A)
(mat-to-cols-list B))

lemma dim-row-tensor-mat [simp]:
dim-row (A  $\otimes$  B) = dim-row A * dim-row B
by (simp add: mat-of-cols-list-def tensor-mat-def)

lemma dim-col-tensor-mat [simp]:
dim-col (A  $\otimes$  B) = dim-col A * dim-col B
using tensor-mat-def mat-of-cols-list-def mult.length-Tensor[of 1 (*)]
by(simp add:  $\langle \bigwedge M2 M1. \text{Matrix-Tensor.mult } 1 (*) \Rightarrow \text{length} (\text{mult.Tensor} (*) M1 M2) = \text{length } M1 * \text{length } M2 \rangle \text{ mult.intro}$ )

lemma mat-to-cols-list-nth-nth-eq-index-mat:
 $\langle \text{mat-to-cols-list } A ! j ! i = A \$\$ (i, j) \rangle \text{ if } \langle i < \text{dim-row } A \rangle \langle j < \text{dim-col } A \rangle$ 
using that by (simp add: mat-to-cols-list-def)

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

```

```

from a5 have ⟨rB > 0⟩
  by (auto intro: gr0I)
have (A ⊗ B) $$ (i,j) = (mult.Tensor (*) (mat-to-cols-list A) (mat-to-cols-list
B)) ! j ! i
  using assms tensor-mat-def mat-of-cols-list-def dim-col-tensor-mat by simp
moreover have f:i < mult.row-length (mat-to-cols-list A) * mult.row-length
(mat-to-cols-list B)
  by (simp add: a1 a2 a3 a4 a5 a7 a8)
moreover have j < length (mat-to-cols-list A) * length (mat-to-cols-list B)
  by (simp add: a2 a4 a6)
moreover have mat (mult.row-length (mat-to-cols-list A)) (length (mat-to-cols-list
A)) (mat-to-cols-list A)
  using a2 a7 mat-to-cols-list-is-mat by blast
moreover have mat (mult.row-length (mat-to-cols-list B)) (length (mat-to-cols-list
B)) (mat-to-cols-list B)
  using a4 a8 mat-to-cols-list-is-mat by blast
ultimately have (A ⊗ B) $$ (i,j) =
  (mat-to-cols-list A) ! (j div length (mat-to-cols-list B)) ! (i div mult.row-length
(mat-to-cols-list B))
  * (mat-to-cols-list B) ! (j mod length (mat-to-cols-list B)) ! (i mod mult.row-length
(mat-to-cols-list B))
  using mult.matrix-Tensor-elements[of 1 (*)]
  by(simp add: ∧ M2 M1. mult 1 (*) ⟹ ∀ i j. (i < mult.row-length M1 *
mult.row-length M2
  ∧ j < length M1 * length M2) ∧ mat (mult.row-length M1) (length M1) M1 ∧
mat (mult.row-length M2) (length M2) M2 ⟶
  mult.Tensor (*) M1 M2 ! j ! i = M1 ! (j div length M2) ! (i div mult.row-length
M2) * M2 ! (j mod length M2) ! (i mod mult.row-length M2) mult.intro)
moreover have ⟨i div rB < rA⟩ ⟨j div cB < cA⟩ ⟨i mod rB < rB⟩
  using a5 a6 ⟨rB > 0⟩ by (simp-all add: less-mult-imp-div-less)
ultimately show ?thesis
  using a1 a2 a3 a4 a8
  by (simp add: mat-to-cols-list-nth-nth-eq-index-mat)
qed

```

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

```

lemma Matrix-row-is-Legacy-row:
  assumes i < dim-row A
  shows Matrix.row A i = vec-of-list (row (mat-to-cols-list A) i)
proof
  show dim-vec (Matrix.row A i) = dim-vec (vec-of-list (row (mat-to-cols-list A)
i))
    using length-mat-to-cols-list Matrix.dim-vec-of-list by (metis row-def index-row(2)
length-map)
next
  show ∧ j. j < dim-vec (vec-of-list (row (mat-to-cols-list A) i)) ⟹
    Matrix.row A i $ j = vec-of-list (row (mat-to-cols-list A) i) $ j
    using Matrix.row-def vec-of-list-def mat-to-cols-list-def
    by (smt (verit) row-def assms dim-vec-of-list index-mat-of-cols-list index-row(1))

```

length-mat-to-cols-list length-row-mat-to-cols-list mat-to-cols-list-to-mat nth-map vec-of-list-index)
qed

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

```
lemma Legacy-row-is-Matrix-row:
  assumes i < mult.row-length A
  shows row A i = list-of-vec (Matrix.row (mat-of-cols-list (mult.row-length A) A)
i)
  proof (rule nth-equalityI)
    show length (row A i) = length (list-of-vec (Matrix.row (mat-of-cols-list (mult.row-length
A) A) i))
    using row-def length-list-of-vec by(metis mat-of-cols-list-def dim-col-mat(1)
index-row(2) length-map)
  next
    fix j:: nat
    assume j < length (row A i)
    then show row A i ! j = list-of-vec (Matrix.row (mat-of-cols-list (mult.row-length
A) A) i) ! j
    using assms index-mat-of-cols-list
    by(metis row-def mat-of-cols-list-def dim-col-mat(1) dim-row-mat(1) index-row(1)
length-map list-of-vec-index nth-map)
  qed
```

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

```
lemma Matrix-col-is-Legacy-col:
  assumes j < dim-col A
  shows Matrix.col A j = vec-of-list (col (mat-to-cols-list A) j)
proof
  show dim-vec (Matrix.col A j) = dim-vec (vec-of-list (col (mat-to-cols-list A) j))
  by (simp add: col-def assms mat-to-cols-list-def)
next
  show  $\bigwedge i. i < \text{dim-vec}(\text{vec-of-list}(\text{col}(\text{mat-to-cols-list } A) j)) \implies$ 
    Matrix.col A j $ i = vec-of-list (col (mat-to-cols-list A) j) $ i
  using mat-to-cols-list-def
  by (metis col-def assms col-mat-of-cols-list length-col-mat-to-cols-list length-mat-to-cols-list
mat-to-cols-list-to-mat)
qed
```

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

```
lemma Legacy-col-is-Matrix-col:
  assumes a1:j < length A and a2:length (A ! j) = mult.row-length A
  shows col A j = list-of-vec (Matrix.col (mat-of-cols-list (mult.row-length A) A)
j)
proof (rule nth-equalityI)
  have length (list-of-vec (Matrix.col (mat-of-cols-list (mult.row-length A) A) j))
  =
  dim-vec (Matrix.col (mat-of-cols-list (mult.row-length A) A) j)
```

```

using length-list-of-vec by blast
also have ... = dim-row (mat-of-cols-list (mult.row-length A) A)
  using Matrix.col-def by simp
also have f1:... = mult.row-length A
  by (simp add: mat-of-cols-list-def)
finally show f2:length (col A j) = length (list-of-vec (Matrix.col (mat-of-cols-list
(mult.row-length A) A) j))
  using a2 by (simp add: col-def)
next
fix i:: nat
assume i < length (col A j)
then show (col A j) ! i = (list-of-vec (Matrix.col (mat-of-cols-list (mult.row-length
A) A) j)) ! i
  by (metis col-def a1 a2 col-mat-of-cols-list list-vec)
qed

```

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

```

lemma scalar-prod-is-Matrix-scalar-prod [simp]:
fixes u::complex list and v::complex list
assumes length u = length v
shows plus-mult.scalar-product (*) 0 (+) u v = (vec-of-list u) · (vec-of-list v)
proof -
have f:(vec-of-list u) · (vec-of-list v) = (∑ i=0..<length v. u ! i * v ! i)
  using assms scalar-prod-def[of vec-of-list u vec-of-list v] Matrix.dim-vec-of-list[of
v] index-vec-of-list
  by (metis (no-types, lifting) atLeastLessThanIff sum.cong)
thus ?thesis
proof -
have plus-mult.scalar-product (*) 0 (+) u v = semiring-0-class.scalar-prod u v
  using plus-mult.scalar-product-def[of 1 (*) 0 (+) a-inv cpx-rng u v] by simp
also have ... = sum-list (map (λ(x,y). x * y) (zip u v))
  by (simp add: scalar-prod)
moreover have ∀ i < length v. (zip u v) ! i = (u ! i, v ! i)
  using assms zip-def by simp
then have ∀ i < length v. (map (λ(x,y). x * y) (zip u v)) ! i = u ! i * v ! i
  by (simp add: assms)
ultimately have plus-mult.scalar-product (*) 0 (+) u v = (∑ i=0..<length v.
u ! i * v ! i)
  by (metis (no-types, lifting) assms atLeastLessThanIff length-map map-fst-zip
sum.cong sum-list-sum-nth)
thus ?thesis by (simp add: f)
qed
qed

```

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

```

lemma matrix-mult-to-times-mat:
assumes dim-col A > 0 and dim-col B > 0 and dim-col (A::complex Matrix.mat) = dim-row B

```

```

shows  $A * B = \text{mat-of-cols-list}(\dim\text{-row } A) (\text{plus-mult.matrix-mult } (*) 0 (+) (\text{mat-to-cols-list } A) (\text{mat-to-cols-list } B))$ 
proof
  define  $M$  where  $M = \text{mat-of-cols-list}(\dim\text{-row } A) (\text{plus-mult.matrix-mult } (*) 0 (+) (\text{mat-to-cols-list } A) (\text{mat-to-cols-list } B))$ 
  then show  $f1:\dim\text{-row}(A * B) = \dim\text{-row } M$ 
    by (simp add: mat-of-cols-list-def times-mat-def)
  have  $\text{length}(\text{plus-mult.matrix-mult } (*) 0 (+) (\text{mat-to-cols-list } A) (\text{mat-to-cols-list } B)) = \dim\text{-col } B$ 
    by (simp add: mat-multI-def)
  then show  $f2:\dim\text{-col}(A * B) = \dim\text{-col } M$ 
    by (simp add: M-def times-mat-def mat-of-cols-list-def)
  show  $\bigwedge i j. i < \dim\text{-row } M \implies j < \dim\text{-col } M \implies (A * B) \$\$ (i, j) = M \$\$ (i, j)$ 
proof -
  fix  $i j$ 
  assume  $a1:i < \dim\text{-row } M$  and  $a2:j < \dim\text{-col } M$ 
  then have  $(A * B) \$\$ (i, j) = \text{Matrix.row } A i \cdot \text{Matrix.col } B j$ 
    using  $f1 f2$  by simp
  also have ... =  $\text{vec-of-list}(\text{row}(\text{mat-to-cols-list } A) i) \cdot \text{vec-of-list}(\text{col}(\text{mat-to-cols-list } B) j)$ 
    using  $f1 f2 a1 a2$  by (simp add: Matrix-row-is-Legacy-row Matrix-col-is-Legacy-col)
  also have ... =  $\text{plus-mult.scalar-product } (*) 0 (+) (\text{row}(\text{mat-to-cols-list } A) i) (\text{col}(\text{mat-to-cols-list } B) j)$ 
    using  $a1 a2 \text{assms}(3)$   $f1 f2$  by simp
  also have  $M \$\$ (i, j) = \text{plus-mult.scalar-product } (*) 0 (+) (\text{row}(\text{mat-to-cols-list } A) i) (\text{col}(\text{mat-to-cols-list } B) j)$ 
proof -
  have  $M \$\$ (i, j) = (\text{plus-mult.matrix-mult } (*) 0 (+) (\text{mat-to-cols-list } A) (\text{mat-to-cols-list } B)) ! j ! i$ 
    using  $M\text{-def } f1 f2$ 
  have  $\langle \text{length}(\text{mat-mult}(\text{mult.row-length } (\text{mat-to-cols-list } A))) (\text{mat-to-cols-list } A) (\text{mat-to-cols-list } B) \rangle = \dim\text{-col } B$ 
    a1 a2 by simp
    moreover have  $\text{mat}(\text{mult.row-length } (\text{mat-to-cols-list } A)) (\dim\text{-col } A)$ 
      (mat-to-cols-list A)
      using mat-to-cols-list-is-mat assms(1) by simp
    moreover have  $\text{mat}(\dim\text{-col } A) (\dim\text{-col } B)$  (mat-to-cols-list B)
      using assms(2) assms(3) mat-to-cols-list-is-mat by simp
    ultimately show ?thesis
      using assms(1) a1 a2 row-length-mat-to-cols-list plus-mult.matrix-index[of
      1 (*) 0 (+)] plus-mult-cpx
        by (smt (verit) f1 f2 index-mult-mat(2) index-mult-mat(3))
      qed
      finally show  $(A * B) \$\$ (i, j) = M \$\$ (i, j)$  by simp
    qed
  qed
lemma mat-to-cols-list-times-mat [simp]:
  assumes  $\dim\text{-col } A = \dim\text{-row } B$  and  $\dim\text{-col } A > 0$ 

```

```

shows mat-to-cols-list (A * B) = plus-mult.matrix-mult (*) 0 (+) (mat-to-cols-list
A) (mat-to-cols-list B)
proof (rule nth-equalityI)
  define M where M = plus-mult.matrix-mult (*) 0 (+) (mat-to-cols-list A)
  (mat-to-cols-list B)
  then show f0:length (mat-to-cols-list (A * B)) = length M by (simp add:
  mat-multI-def)
  moreover have f1: $\bigwedge j. j < \text{length} (\text{mat-to-cols-list} (A * B)) \longrightarrow \text{mat-to-cols-list}$ 
  (A * B) ! j = M ! j
  proof
    fix j:: nat
    assume a0: $j < \text{length} (\text{mat-to-cols-list} (A * B))$ 
    then have length (mat-to-cols-list (A * B) ! j) = dim-row A
      by (simp add: mat-to-cols-list-def)
    then also have f2:length (M ! j) = dim-row A
      using a0 M-def mat-multI-def[of 0 (+) (*) dim-row A mat-to-cols-list A
      mat-to-cols-list B]
      row-length-mat-to-cols-list assms(2)
      by (metis assms(1) f0 length-greater-0-conv length-map length-mat-to-cols-list

list-to-mat-to-cols-list mat-mult mat-to-cols-list-is-mat matrix-mult-to-times-mat)
ultimately have length (mat-to-cols-list (A * B) ! j) = length (M ! j) by simp
moreover have  $\bigwedge i. i < \text{dim-row} A \longrightarrow \text{mat-to-cols-list} (A * B) ! j ! i = M ! j$ 
! i
proof
  fix i
  assume a1: $i < \text{dim-row} A$ 
  have mat (mult.row-length (mat-to-cols-list A)) (dim-col A) (mat-to-cols-list
  A)
    using mat-to-cols-list-is-mat assms(2) by simp
  moreover have mat (dim-col A) (dim-col B) (mat-to-cols-list B)
    using mat-to-cols-list-is-mat assms(1) a0 by simp
  ultimately have M ! j ! i = plus-mult.scalar-product (*) 0 (+) (row
  (mat-to-cols-list A) i) (col (mat-to-cols-list B) j)
    using plus-mult.matrix-index a0 a1 row-length-mat-to-cols-list assms(2)
    plus-mult-cpx M-def
    by (metis index-mult-mat(3) length-mat-to-cols-list)
    also have ... = vec-of-list (row (mat-to-cols-list A) i) · vec-of-list (col
    (mat-to-cols-list B) j)
      using a0 a1 assms(1) by simp
    finally show mat-to-cols-list (A * B) ! j ! i = M ! j ! i
      using mat-to-cols-list-def index-mult-mat(1) a0 a1
      by (simp add: Matrix-row-is-Legacy-row Matrix-col-is-Legacy-col)
    qed
  ultimately show mat-to-cols-list (A * B) ! j = M ! j by (simp add: nth-equalityI
f2)
  qed
  fix i:: nat
  assume i < length (mat-to-cols-list (A * B))

```

```

thus mat-to-cols-list (A * B) ! i = M ! i by (simp add: f1)
qed

```

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

```

lemma mult-distr-tensor:
assumes a1:dim-col A = dim-row B and a2:dim-col C = dim-row D and
a3:dim-col A > 0 and
a4:dim-col B > 0 and a5:dim-col C > 0 and a6:dim-col D > 0
shows (A * B)  $\otimes$  (C * D) = (A  $\otimes$  C) * (B  $\otimes$  D)
proof -
define A' B' C' D' M N where A' = mat-to-cols-list A and B' = mat-to-cols-list
B and
C' = mat-to-cols-list C and D' = mat-to-cols-list D and
M = mat-of-cols-list (dim-row A * dim-row C) (mult.Tensor (*)) (mat-to-cols-list
A) (mat-to-cols-list C)) and
N = mat-of-cols-list (dim-row B * dim-row D) (mult.Tensor (*)) (mat-to-cols-list
B) (mat-to-cols-list D))
then have (A  $\otimes$  C) * (B  $\otimes$  D) = M * N
by (simp add: tensor-mat-def)
also have ... = mat-of-cols-list (dim-row A * dim-row C) (plus-mult.matrix-mult
(*) 0 (+)
(mat-to-cols-list M) (mat-to-cols-list N))
using assms M-def N-def dim-col-tensor-mat dim-row-tensor-mat tensor-mat-def
by (simp add: matrix-mult-to-times-mat)
also have f1:... = mat-of-cols-list (dim-row A * dim-row C) (plus-mult.matrix-mult
(*) 0 (+)
(mult.Tensor (*)) A' C') (mult.Tensor (*)) B' D')
proof -
define M' N' where M' = mult.Tensor (*) (mat-to-cols-list A) (mat-to-cols-list
C) and
N' = mult.Tensor (*) (mat-to-cols-list B) (mat-to-cols-list D)
then have mat (mult.row-length M') (length M') M'
using M'-def mult.effective-well-defined-Tensor[of 1 (*)] mat-to-cols-list-is-mat
a3 a5
by (smt (verit) mult.length-Tensor mult.row-length-mat plus-mult-cpx plus-mult-def)
moreover have mat (mult.row-length N') (length N') N'
using N'-def mult.effective-well-defined-Tensor[of 1 (*)] mat-to-cols-list-is-mat
a4 a6
by (smt (verit) mult.length-Tensor mult.row-length-mat plus-mult-cpx plus-mult-def)
ultimately show ?thesis
using list-to-mat-to-cols-list M-def N-def mult.row-length-mat row-length-mat-to-cols-list
assms(3) a4 a5 a6 A'-def B'-def C'-def D'-def by (metis M'-def N'-def
plus-mult-cpx plus-mult-def)
qed
also have ... = mat-of-cols-list (dim-row A * dim-row C) (mult.Tensor (*))
(plus-mult.matrix-mult (*)) 0 (+) A' B'

```

```

(plus-mult.matrix-mult (*) 0 (+) C' D'))
proof –
  have f2:mat (mult.row-length A') (length A') A'
    using A'-def a3 mat-to-cols-list-is-mat by simp
  moreover have mat (mult.row-length B') (length B') B'
    using B'-def a4 mat-to-cols-list-is-mat by simp
  moreover have mat (mult.row-length C') (length C') C'
    using C'-def a5 mat-to-cols-list-is-mat by simp
  moreover have mat (mult.row-length D') (length D') D'
    using D'-def a6 mat-to-cols-list-is-mat by simp
  moreover have length A' = mult.row-length B'
    using A'-def B'-def a1 a4 by simp
  moreover have length C' = mult.row-length D'
    using C'-def D'-def a2 a6 by simp
  moreover have A' ≠ [] ∧ B' ≠ [] ∧ C' ≠ [] ∧ D' ≠ []
    using A'-def B'-def C'-def D'-def a3 a4 a5 a6 by simp
  ultimately have plus-mult.matrix-match A' B' C' D'
    using plus-mult.matrix-match-def[of 1 (*) 0 (+) a-inv cpx-rng] by simp
  thus ?thesis
    using f1 plus-mult.distributivity plus-mult-cpx by fastforce
  qed
  also have ... = mat-of-cols-list (dim-row A * dim-row C) (mult.Tensor (*)
    (mat-to-cols-list (A * B)) (mat-to-cols-list (C * D)))
    using A'-def B'-def C'-def D'-def a1 a2 a3 a5 by simp
  finally show ?thesis by(simp add: tensor-mat-def)
  qed

lemma tensor-mat-is-assoc:
  fixes A B C:: complex Matrix.mat
  shows A ⊗ (B ⊗ C) = (A ⊗ B) ⊗ C
proof –
  define M where d:M = mat-of-cols-list (dim-row B * dim-row C) (mult.Tensor (*)
    (mat-to-cols-list B) (mat-to-cols-list C))
  then have *: B ⊗ C = M
    using tensor-mat-def by simp
  then have **: A ⊗ (B ⊗ C) = mat-of-cols-list (dim-row A * (dim-row B *
    dim-row C))
    (mult.Tensor (*)) (mat-to-cols-list A) (mat-to-cols-list M)
    using tensor-mat-def d dim-row-tensor-mat by simp
  then have ***: mat-to-cols-list M = mult.Tensor (*) (mat-to-cols-list B) (mat-to-cols-list
    C)
    unfolding d using list-to-mat-to-cols-list
    by (smt (verit) Tensor.mat-of-cols-list-def bot-nat-0.not-eq-extremum * d dim-col-mat(1)
      dim-col-tensor-mat length-greater-0-conv length-mat-to-cols-list mat-to-cols-list-is-mat
      mult.effective-well-defined-Tensor mult-is-0 plus-mult-cpx plus-mult-def row-length-mat-to-cols-list)
  with ** have A ⊗ (B ⊗ C) = mat-of-cols-list (dim-row A * (dim-row B *
    dim-row C))
    (mult.Tensor (*)) (mat-to-cols-list A) (mult.Tensor (*)) (mat-to-cols-list B) (mat-to-cols-list
    C))) by simp

```

```

moreover have ... = mat-of-cols-list ((dim-row A * dim-row B) * dim-row C)
(mult.Tensor (*) (mult.Tensor (*) (mat-to-cols-list A) (mat-to-cols-list B)) (mat-to-cols-list C))
  using Matrix-Tensor.mult.associativity
  by (smt (verit) ab-semigroup-mult-class.mult-ac(1) length-greater-0-conv length-mat-to-cols-list
mat-to-cols-list-is-mat mult.Tensor.simps(1) mult.Tensor-null plus-mult-cpx plus-mult-def)
  ultimately show ?thesis
  using tensor-mat-def
  by (smt (verit) Tensor.mat-of-cols-list-def dim-col-mat(1) dim-col-tensor-mat
dim-row-tensor-mat length-0-conv
list-to-mat-to-cols-list mat-to-cols-list-is-mat mult.well-defined-Tensor mult-is-0 neq0-conv
plus-mult-cpx plus-mult-def row-length-mat-to-cols-list)
qed

end

```

6 Further Results on Tensor Products

```

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

lemma tensor-prod-2 [simp]:
mult.vec-vec-Tensor (*) [x1::complex,x2] [x3, x4] = [x1 * x3, x1 * x4, x2 * x3,
x2 * x4]
proof -
  have Matrix-Tensor.mult (1::complex) (*)
    by (simp add: Matrix-Tensor.mult-def)
  thus mult.vec-vec-Tensor (*) [x1::complex,x2] [x3,x4] = [x1*x3,x1*x4,x2*x3,x2*x4]
    using mult.vec-vec-Tensor-def[of (1::complex) (*)] mult.times-def[of (1::complex) (*)]
  qed

lemma list-vec [simp]:
  assumes v ∈ state-qbit 1
  shows list-of-vec v = [v $ 0, v $ 1]
proof -
  have Rep-vec v = (fst(Rep-vec v), snd(Rep-vec v)) by simp
  also have ... = (2, vec-index v)
  by (metis (mono-tags, lifting) assms dim-vec.rep-eq mem-Collect-eq power-one-right
state-qbit-def vec-index.rep-eq)
  moreover have [0..<2::nat] = [0,1]
  by(simp add: upt-rec)

```

```

ultimately show ?thesis
  by(simp add: list-of-vec-def)
qed

lemma vec-tensor-prod-2 [simp]:
assumes v ∈ state-qbit 1 and w ∈ state-qbit 1
shows v ⊗ w = vec-of-list [v $ 0 * w $ 0, v $ 0 * w $ 1, v $ 1 * w $ 0, v $ 1
* w $ 1]
proof -
  have list-of-vec v = [v $ 0, v $ 1]
    using assms by simp
  moreover have list-of-vec w = [w $ 0, w $ 1]
    using assms by simp
  ultimately show v ⊗ w = vec-of-list [v $ 0 * w $ 0, v $ 0 * w $ 1, v $ 1 * w
$ 0, v $ 1 * w $ 1]
    by(simp add: tensor-vec-def)
qed

lemma vec-dim-of-vec-of-list [simp]:
assumes length l = n
shows dim-vec (vec-of-list l) = n
using assms vec-of-list-def by simp

lemma vec-tensor-prod-2-bis [simp]:
assumes v ∈ state-qbit 1 and w ∈ state-qbit 1
shows v ⊗ w = Matrix.vec 4 (λi. if i = 0 then v $ 0 * w $ 0 else
if i = 3 then v $ 1 * w $ 1 else
if i = 1 then v $ 0 * w $ 1 else v $ 1 * w $ 0)
proof
  define u where u = Matrix.vec 4 (λi. if i = 0 then v $ 0 * w $ 0 else
if i = 3 then v $ 1 * w $ 1 else
if i = 1 then v $ 0 * w $ 1 else v $ 1 * w $ 0)
  then show f2:dim-vec (v ⊗ w) = dim-vec u
    using assms by simp
  show ∀i. i < dim-vec u ⟹ (v ⊗ w) $ i = u $ i
    apply (auto simp: u-def)
    using assms apply auto[3]
    apply (simp add: numeral-3-eq-3)
    using u-def vec-of-list-index vec-tensor-prod-2 index-is-2
    by (metis (no-types, lifting) One-nat-def assms nth-Cons-0 nth-Cons-Suc nu-
meral-2-eq-2)
qed

lemma index-col-mat-of-cols-list [simp]:
assumes i < n and j < length l
shows Matrix.col (mat-of-cols-list n l) j $ i = l ! j ! i
apply (auto simp: Matrix.col-def mat-of-cols-list-def)
using assms less-le-trans by fastforce

```

```

lemma multTensor2 [simp]:
  assumes a1:A = Matrix.mat 2 1 ( $\lambda(i,j)$ . if  $i = 0$  then a0 else a1) and
           a2:B = Matrix.mat 2 1 ( $\lambda(i,j)$ . if  $i = 0$  then b0 else b1)
  shows mult.Tensor (*) (mat-to-cols-list A) (mat-to-cols-list B) = [[a0*b0, a0*b1,
           a1*b0, a1*b1]]
  proof -
    have mat-to-cols-list A = [[a0, a1]]
    by (auto simp: a1 mat-to-cols-list-def) (simp add: numeral-2-eq-2)
    moreover have f2:mat-to-cols-list B = [[b0, b1]]
    by (auto simp: a2 mat-to-cols-list-def) (simp add: numeral-2-eq-2)
    ultimately have mult.Tensor (*) (mat-to-cols-list A) (mat-to-cols-list B) =
      mult.Tensor (*) [[a0,a1]] [[b0,b1]] by simp
  thus ?thesis
    using mult.Tensor-def[of (1::complex) (*)] mult.times-def[of (1::complex) (*)]
    by (metis (mono-tags, lifting) append-self-conv list.simps(6) mult.Tensor.simps(2)
      mult.vec-mat-Tensor.simps(1)
      mult.vec-mat-Tensor.simps(2) plus-mult-cpx plus-mult-def tensor-prod-2)
  qed

lemma multTensor2-bis [simp]:
  assumes a1:dim-row A = 2 and a2:dim-col A = 1 and a3:dim-row B = 2 and
           a4:dim-col B = 1
  shows mult.Tensor (*) (mat-to-cols-list A) (mat-to-cols-list B) =
    [[A $$ (0,0) * B $$ (0,0), A $$ (0,0) * B $$ (1,0), A $$ (1,0) * B $$ (0,0), A
    $$ (1,0) * B $$ (1,0)]]
  proof -
    have mat-to-cols-list A = [[A $$ (0,0), A $$ (1,0)]]
    by (auto simp: a1 a2 mat-to-cols-list-def) (simp add: numeral-2-eq-2)
    moreover have f2:mat-to-cols-list B = [[B $$ (0,0), B $$ (1,0)]]
    by (auto simp: a3 a4 mat-to-cols-list-def) (simp add: numeral-2-eq-2)
    ultimately have mult.Tensor (*) (mat-to-cols-list A) (mat-to-cols-list B) =
      mult.Tensor (*) [[A $$ (0,0), A $$ (1,0)]] [[B $$ (0,0), B $$ (1,0)]]
  by simp
  thus ?thesis
    using mult.Tensor-def[of (1::complex) (*)] mult.times-def[of (1::complex) (*)]
    by (smt (verit) append-self-conv list.simps(6) mult.Tensor.simps(2) mult.vec-mat-Tensor.simps(1)
      mult.vec-mat-Tensor.simps(2) plus-mult-cpx plus-mult-def tensor-prod-2)
  qed

lemma mat-tensor-prod-2-prelim [simp]:
  assumes state 1 v and state 1 w
  shows v  $\otimes$  w = mat-of-cols-list 4
    [[v $$ (0,0) * w $$ (0,0), v $$ (0,0) * w $$ (1,0), v $$ (1,0) * w $$ (0,0), v
    $$ (1,0) * w $$ (1,0)]]
  proof
    define u where u = mat-of-cols-list 4
    [[v $$ (0,0) * w $$ (0,0), v $$ (0,0) * w $$ (1,0), v $$ (1,0) * w $$ (0,0), v
    $$ (1,0) * w $$ (1,0)]]

```

```

then show f1:dim-row (v  $\otimes$  w) = dim-row u
  using assms state-def mat-of-cols-list-def tensor-mat-def by simp
show f2:dim-col (v  $\otimes$  w) = dim-col u
  using assms state-def mat-of-cols-list-def tensor-mat-def u-def by simp
show  $\bigwedge i j. i < \text{dim-row } u \implies j < \text{dim-col } u \implies (v \otimes w) \$\$ (i, j) = u \$\$ (i, j)$ 
  using u-def tensor-mat-def assms state-def by simp
qed

lemma mat-tensor-prod-2-col [simp]:
  assumes state 1 v and state 1 w
  shows Matrix.col (v  $\otimes$  w) 0 = Matrix.col v 0  $\otimes$  Matrix.col w 0
proof
  show f1:dim-vec (Matrix.col (v  $\otimes$  w) 0) = dim-vec (Matrix.col v 0  $\otimes$  Matrix.col w 0)
  using assms vec-tensor-prod-2-bis
  by (smt (verit) Tensor.mat-of-cols-list-def dim-col dim-row-mat(1) dim-vec mat-tensor-prod-2-prelim state.state-to-state-qbit)
next
  show  $\bigwedge i. i < \text{dim-vec} (\text{Matrix.col } v 0 \otimes \text{Matrix.col } w 0) \implies \text{Matrix.col} (v \otimes w) 0 \$ i = (\text{Matrix.col } v 0 \otimes \text{Matrix.col } w 0) \$ i$ 
  proof –
    have dim-vec (Matrix.col v 0  $\otimes$  Matrix.col w 0) = 4
    by (metis (no-types, lifting) assms(1) assms(2) dim-vec state.state-to-state-qbit vec-tensor-prod-2-bis)
    moreover have (Matrix.col v 0  $\otimes$  Matrix.col w 0) \$ 0 = v \$\$ (0,0) * w \$\$ (0,0)
    using assms vec-tensor-prod-2 state.state-to-state-qbit col-index-of-mat-col
    by (smt (verit) nth-Cons-0 power-one-right state-def vec-of-list-index zero-less-numeral)
    moreover have ... = Matrix.col (v  $\otimes$  w) 0 \$ 0
    using assms by simp
    moreover have (Matrix.col v 0  $\otimes$  Matrix.col w 0) \$ 1 = v \$\$ (0,0) * w \$\$ (1,0)
    using assms vec-tensor-prod-2 state.state-to-state-qbit col-index-of-mat-col
    by (smt (verit) One-nat-def Suc-1 lessI nth-Cons-0 power-one-right state-def vec-index-vCons-Suc
    vec-of-list-Cons vec-of-list-index zero-less-numeral)
    moreover have ... = Matrix.col (v  $\otimes$  w) 0 \$ 1
    using assms by simp
    moreover have (Matrix.col v 0  $\otimes$  Matrix.col w 0) \$ 2 = v \$\$ (1,0) * w \$\$ (0,0)
    using assms vec-tensor-prod-2 state.state-to-state-qbit col-index-of-mat-col
    by (smt (verit) One-nat-def Suc-1 lessI nth-Cons-0 power-one-right state-def vec-index-vCons-Suc
    vec-of-list-Cons vec-of-list-index zero-less-numeral)
    moreover have ... = Matrix.col (v  $\otimes$  w) 0 \$ 2
    using assms by simp
    moreover have (Matrix.col v 0  $\otimes$  Matrix.col w 0) \$ 3 = v \$\$ (1,0) * w \$\$ (1,0)
  
```

```

using assms vec-tensor-prod-2 state.state-to-state-qbit col-index-of-mat-col
numeral-3-eq-3
by (smt (verit) One-nat-def Suc-1 lessI nth-Cons-0 power-one-right state-def
vec-index-vCons-Suc
vec-of-list-Cons vec-of-list-index zero-less-numeral)
moreover have ... = Matrix.col (v  $\otimes$  w) 0 \$ 3
using assms by simp
ultimately show  $\bigwedge i. i < \dim\text{-vec} (\text{Matrix.col } v 0 \otimes \text{Matrix.col } w 0) \implies \text{Matrix.col } (v \otimes w) 0 \$ i = (\text{Matrix.col } v 0 \otimes \text{Matrix.col } w 0) \$ i$ 
using index-sl-four by auto
qed
qed

lemma mat-tensor-prod-2 [simp]:
assumes state 1 v and state 1 w
shows v  $\otimes$  w = Matrix.mat 4 1 ( $\lambda(i,j)$ . if  $i = 0$  then  $v \$\$ (0,0) * w \$\$ (0,0)$ 
else
if  $i = 3$  then  $v \$\$ (1,0) * w \$\$ (1,0)$  else
if  $i = 1$  then  $v \$\$ (0,0) * w \$\$ (1,0)$  else
 $v \$\$ (1,0) * w \$\$ (0,0)$ )
proof
define u where u = Matrix.mat 4 1 ( $\lambda(i,j)$ . if  $i = 0$  then  $v \$\$ (0,0) * w \$\$ (0,0)$ 
else
if  $i = 3$  then  $v \$\$ (1,0) * w \$\$ (1,0)$  else
if  $i = 1$  then  $v \$\$ (0,0) * w \$\$ (1,0)$  else
 $v \$\$ (1,0) * w \$\$ (0,0)$ )
then show dim-row (v  $\otimes$  w) = dim-row u
using assms tensor-mat-def state-def by (simp add: Tensor.mat-of-cols-list-def)
also have ... = 4 by (simp add: u-def)
show dim-col (v  $\otimes$  w) = dim-col u
using u-def assms tensor-mat-def state-def Tensor.mat-of-cols-list-def by simp
moreover have ... = 1 by (simp add: u-def)
ultimately show  $\bigwedge i j. i < \dim\text{-row } u \implies j < \dim\text{-col } u \implies (v \otimes w) \$\$ (i, j) = u \$\$ (i, j)$ 
proof –
fix i j::nat
assume a1:i < dim-row u and a2:j < dim-col u
then have (v  $\otimes$  w) \$\$ (i, j) = Matrix.col (v  $\otimes$  w) 0 \$ i
using Matrix.col-def u-def assms by simp
then have f1:(v  $\otimes$  w) \$\$ (i, j) = (Matrix.col v 0  $\otimes$  Matrix.col w 0) \$ i
using assms mat-tensor-prod-2-col by simp
have (Matrix.col v 0  $\otimes$  Matrix.col w 0) \$ i =
Matrix.vec 4 ( $\lambda i.$  if  $i = 0$  then Matrix.col v 0 \$ 0 * Matrix.col w 0 \$ 0
else
if  $i = 3$  then Matrix.col v 0 \$ 1 * Matrix.col w 0 \$ 0
1 else
if  $i = 1$  then Matrix.col v 0 \$ 0 * Matrix.col w 0 \$ 0
\$ 1 else
 $Matrix.col v 0 \$ 1 * Matrix.col w 0 \$ 0) \$ i$ 
using vec-tensor-prod-2-bis assms state.state-to-state-qbit by presburger

```

```

thus  $(v \otimes w) \$\$ (i, j) = u \$\$ (i, j)$ 
  using u-def a1 a2 assms state-def by simp
qed
qed

lemma mat-tensor-prod-2-bis:
assumes state 1 v and state 1 w
shows  $v \otimes w = |\text{Matrix.vec } 4 (\lambda i. \text{if } i = 0 \text{ then } v \$\$ (0,0) * w \$\$ (0,0) \text{ else}$ 
       $\quad \text{if } i = 3 \text{ then } v \$\$ (1,0) * w \$\$ (1,0) \text{ else}$ 
       $\quad \text{if } i = 1 \text{ then } v \$\$ (0,0) * w \$\$ (1,0) \text{ else}$ 
       $\quad v \$\$ (1,0) * w \$\$ (0,0))\rangle$ 
using assms ket-vec-def mat-tensor-prod-2 by(simp add: mat-eq-iff)

lemma eq-ket-vec:
fixes u v:: complex Matrix.vec
assumes u = v
shows  $|u\rangle = |v\rangle$ 
using assms by simp

lemma mat-tensor-ket-vec:
assumes state 1 v and state 1 w
shows  $v \otimes w = |(\text{Matrix.col } v 0) \otimes (\text{Matrix.col } w 0)\rangle$ 
proof -
have  $v \otimes w = |\text{Matrix.col } v 0\rangle \otimes |\text{Matrix.col } w 0\rangle$ 
  using assms state-def by simp
also have ... =
 $|\text{Matrix.vec } 4 (\lambda i. \text{if } i = 0 \text{ then } |\text{Matrix.col } v 0\rangle \$\$ (0,0) * |\text{Matrix.col } w 0\rangle \$\$$ 
 $(0,0) \text{ else}$ 
 $\quad \text{if } i = 3 \text{ then } |\text{Matrix.col } v 0\rangle \$\$ (1,0) * |\text{Matrix.col }$ 
 $w 0\rangle \$\$ (1,0) \text{ else}$ 
 $\quad \text{if } i = 1 \text{ then } |\text{Matrix.col } v 0\rangle \$\$ (0,0) *$ 
 $|\text{Matrix.col } w 0\rangle \$\$ (1,0) \text{ else}$ 
 $\quad |\text{Matrix.col } v 0\rangle \$\$ (1,0) * |\text{Matrix.col } w 0\rangle$ 
 $\quad \$\$ (0,0))\rangle$ 
using assms mat-tensor-prod-2-bis state-col-ket-vec by simp
also have ... =
 $|\text{Matrix.vec } 4 (\lambda i. \text{if } i = 0 \text{ then } v \$\$ (0,0) * w \$\$ (0,0) \text{ else}$ 
 $\quad \text{if } i = 3 \text{ then } v \$\$ (1,0) * w \$\$ (1,0) \text{ else}$ 
 $\quad \text{if } i = 1 \text{ then } v \$\$ (0,0) * w \$\$ (1,0) \text{ else}$ 
 $\quad v \$\$ (1,0) * w \$\$ (0,0))\rangle$ 
using assms mat-tensor-prod-2-bis calculation by auto
also have ... =
 $|\text{Matrix.vec } 4 (\lambda i. \text{if } i = 0 \text{ then } \text{Matrix.col } v 0 \$ 0 * \text{Matrix.col } w 0 \$ 0 \text{ else}$ 
 $\quad \text{if } i = 3 \text{ then } \text{Matrix.col } v 0 \$ 1 * \text{Matrix.col } w 0 \$$ 
 $1 \text{ else}$ 
 $\quad \text{if } i = 1 \text{ then } \text{Matrix.col } v 0 \$ 0 * \text{Matrix.col } w 0 \$$ 
 $\$ 1 \text{ else}$ 
 $\quad \text{Matrix.col } v 0 \$ 1 * \text{Matrix.col } w 0 \$ 0)\rangle$ 

```

```

apply(rule eq-ket-vec)
apply (rule eq-vecI)
using col-index-of-mat-col assms state-def by auto
finally show ?thesis
  using vec-tensor-prod-2-bis assms state.state-to-state-qbit by presburger
qed

```

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

```

lemma tensor-state2 [simp]:
  assumes state 1 u and state 1 v
  shows state 2 (u  $\otimes$  v)
proof
  show dim-col (u  $\otimes$  v) = 1
    using assms dim-col-tensor-mat state.is-column by presburger
  show dim-row (u  $\otimes$  v) = 22
    using assms dim-row-tensor-mat state.dim-row
    by (metis (mono-tags, lifting) power2-eq-square power-one-right)
  show \|Matrix.col (u  $\otimes$  v) 0\| = 1
  proof -
    define l where d0:l = [u $$ (0,0) * v $$ (0,0), u $$ (0,0) * v $$ (1,0), u $$ (1,0) * v $$ (0,0), u $$ (1,0) * v $$ (1,0)]
    then have f4:length l = 4 by simp
    also have u  $\otimes$  v = mat-of-cols-list 4
      [[u $$ (0,0) * v $$ (0,0), u $$ (0,0) * v $$ (1,0), u $$ (1,0) * v $$ (0,0), u $$ (1,0) * v $$ (1,0)]]
      using assms by simp
    then have Matrix.col (u  $\otimes$  v) 0 = vec-of-list [u $$ (0,0) * v $$ (0,0), u $$ (0,0) * v $$ (1,0),
      u $$ (1,0) * v $$ (0,0), u $$ (1,0) * v $$ (1,0)]
      by (metis One-nat-def Suc-eq-plus1 add-Suc col-mat-of-cols-list list.size(3)
        list.size(4)
        nth-Cons-0 numeral-2-eq-2 numeral-Bit0 plus-1-eq-Suc vec-of-list-Cons zero-less-one-class.zero-less-one)
      nth-Cons-0 numeral-2-eq-2 numeral-Bit0 plus-1-eq-Suc vec-of-list-Cons zero-less-one-class.zero-less-one)

    then have f5: \|Matrix.col (u  $\otimes$  v) 0\| = sqrt( $\sum i < 4. (cmod(l ! i))^2$ )
      by (metis d0 f4 One-nat-def cpx-length-of-vec-of-list d0 vec-of-list-Cons)
    also have ... = sqrt ((cmod(u $$ (0,0) * v $$ (0,0)))2 + (cmod(u $$ (0,0) * v $$ (1,0)))2
      + (cmod(u $$ (1,0) * v $$ (0,0)))2 + (cmod(u $$ (1,0) * v $$ (1,0)))2)
    proof -
      have ( $\sum i < 4. (cmod(l ! i))^2$ ) = (cmod(l ! 0))2 + (cmod(l ! 1))2 + (cmod(l ! 2))2
      + (cmod(l ! 3))2
        by (simp add: numeral-eq-Suc)
      also have ... = (cmod(u $$ (0,0) * v $$ (0,0)))2 + (cmod(u $$ (0,0) * v $$ (1,0)))2
        + (cmod(u $$ (1,0) * v $$ (0,0)))2 + (cmod(u $$ (1,0) * v $$ (1,0)))2
        using d0 by simp
      finally show ?thesis by (simp add: f5)
    qed
  qed

```

```

moreover have ... =
   $\sqrt{((\text{cmod}(u \$(0,0))^2 * (\text{cmod}(v \$(0,0))^2 + (\text{cmod}(u \$(0,0))^2 * (\text{cmod}(v \$(1,0))^2 + (\text{cmod}(u \$(1,0))^2 * (\text{cmod}(v \$(0,0))^2 + (\text{cmod}(u \$(1,0))^2 * (\text{cmod}(v \$(1,0))^2)))^2)) * (((\text{cmod}(v \$(0,0))^2 + (\text{cmod}(v \$(1,0))^2)))^2)}$ 
    by (simp add: norm-mult power-mult-distrib)
  moreover have ... =  $\sqrt{(((\text{cmod}(u \$(0,0))^2 + (\text{cmod}(u \$(1,0))^2)) * (((\text{cmod}(v \$(0,0))^2 + (\text{cmod}(v \$(1,0))^2)))^2)) * (((\text{cmod}(v \$(0,0))^2 + (\text{cmod}(v \$(1,0))^2)))^2)}$ 
    by (simp add: distrib-left mult.commute)
  ultimately have  $f6: \|\text{Matrix.col}(u \otimes v) 0\|^2 = (((\text{cmod}(u \$(0,0))^2 + (\text{cmod}(u \$(1,0))^2)) * (((\text{cmod}(v \$(0,0))^2 + (\text{cmod}(v \$(1,0))^2)))^2))$ 
    by (simp add: f4)
  also have  $f7:\dots = (\sum i < 2. (\text{cmod}(u \$(i,0))^2) * (\sum i < 2. (\text{cmod}(v \$(i,0))^2))$ 
    by (simp add: numeral-2-eq-2)
  also have  $f8:\dots = (\sum i < 2. (\text{cmod}(\text{Matrix.col } u 0 \$ i))^2) * (\sum i < 2. (\text{cmod}(\text{Matrix.col } v 0 \$ i))^2)$ 
    using assms index-col state-def by simp
  finally show ?thesis
  proof -
    have  $f1:(\sum i < 2. (\text{cmod}(\text{Matrix.col } u 0 \$ i))^2) = 1$ 
      using assms(1) state-def cpx-vec-length-def by auto
    have  $f2:(\sum i < 2. (\text{cmod}(\text{Matrix.col } v 0 \$ i))^2) = 1$ 
      using assms(2) state-def cpx-vec-length-def by auto
    thus ?thesis
      using f1 f8 f5 f6 f7
      by (simp add: sqrt (sum i < 4. (cmod(l ! i))^2) =  $\sqrt{((\text{cmod}(u \$(0,0) * v \$(0,0))^2 + (\text{cmod}(u \$(0,0) * v \$(1,0))^2 + (\text{cmod}(u \$(1,0) * v \$(0,0))^2 + (\text{cmod}(u \$(1,0) * v \$(1,0))^2))})$ )
    qed
    qed
  qed

lemma sum-prod:
  fixes  $f::nat \Rightarrow complex$  and  $g::nat \Rightarrow complex$ 
  shows  $(\sum i < a*b. f(i \text{ div } b) * g(i \text{ mod } b)) = (\sum i < a. f(i)) * (\sum j < b. g(j))$ 
  proof (induction a)
    case 0
    then show ?case by simp
  next
    case (Suc a)
    have  $(\sum i < (a+1)*b. f(i \text{ div } b) * g(i \text{ mod } b)) = (\sum i < a*b. f(i \text{ div } b) * g(i \text{ mod } b)) + (\sum i \in \{a*b..<(a+1)*b\}. f(i \text{ div } b) * g(i \text{ mod } b))$ 
    apply (auto simp: algebra-simps)
    by (smt (verit) add.commute mult-Suc sum.lessThan-Suc sum.nat-group)
    also have ... =  $(\sum i < a. f(i)) * (\sum j < b. g(j)) + (\sum i \in \{a*b..<(a+1)*b\}. f(i \text{ div } b) * g(i \text{ mod } b))$ 

```

```


lemma tensor-state [simp]:  

assumes state m u and state n v  

shows state (m + n) (u  $\otimes$  v)



proof



```

show c1:dim-col (u \otimes v) = 1
 using assms dim-col-tensor-mat state.is-column by presburger
show c2:dim-row (u \otimes v) = 2m+n
 using assms dim-row-tensor-mat state.dim-row by (metis power-add)
have (∑ i<2m+n. (cmod (u $$ (i div 2n, 0) * v $$ (i mod 2n, 0)))2)
= (∑ i<2m+n. (cmod (u $$ (i div 2n, 0)))2 * (cmod (v $$ (i mod 2n, 0)))2)
 by (simp add: sqr-of-cmod-of-prod)
also have ... = (∑ i<2m. (cmod (u $$ (i, 0)))2) * (∑ i<2n. (cmod (v $$ (i, 0)))2)
 proof-
 have ... = (∑ i<2m+n. complex-of-real((cmod (u $$ (i div 2n, 0)))2) *
complex-of-real((cmod (v $$ (i mod 2n, 0)))2))
 by simp
 moreover have (∑ i<2m. (cmod (u $$ (i, 0)))2) = (∑ i<2m. complex-of-real
((cmod (u $$ (i, 0)))2)) by simp
 moreover have (∑ i<2n. (cmod (v $$ (i, 0)))2) = (∑ i<2n. complex-of-real
((cmod (v $$ (i, 0)))2)) by simp
 ultimately show ?thesis
 using sum-prod[of λi. (cmod (u $$ (i, 0)))2 2n λi. (cmod (v $$ (i, 0)))2
2m] by (smt (verit) of-real-eq-iff of-real-mult power-add)
qed
also have ... = 1
 proof-
 have (∑ i<2m. (cmod (u $$ (i, 0)))2) = 1
 using assms(1) state-def cpx-vec-length-def by auto
 moreover have (∑ i<2n. (cmod (v $$ (i, 0)))2) = 1
 using assms(2) state-def cpx-vec-length-def by auto
 ultimately show ?thesis by simp
qed
ultimately show ∥Matrix.col (u \otimes v) 0∥ = 1
```



66


```

```

using c1 c2 assms state-def by (auto simp add: cpx-vec-length-def)
qed

lemma dim-row-of-tensor-gate:
assumes gate m G1 and gate n G2
shows dim-row (G1  $\otimes$  G2) =  $2^{\lceil(m+n)}$ 
using assms dim-row-tensor-mat gate.dim-row by (simp add: power-add)

lemma tensor-gate-sqr-mat:
assumes gate m G1 and gate n G2
shows square-mat (G1  $\otimes$  G2)
using assms gate.square-mat by simp

lemma dim-row-of-one-mat-less-pow:
assumes gate m G1 and gate n G2 and i < dim-row (1m(dim-col G1 * dim-col G2))
shows i <  $2^{\lceil(m+n)}$ 
using assms gate-def by (simp add: power-add)

lemma dim-col-of-one-mat-less-pow:
assumes gate m G1 and gate n G2 and j < dim-col (1m(dim-col G1 * dim-col G2))
shows j <  $2^{\lceil(m+n)}$ 
using assms gate-def by (simp add: power-add)

lemma index-tensor-gate-unitary1:
assumes gate m G1 and gate n G2 and i < dim-row (1m(dim-col G1 * dim-col G2)) and
j < dim-col (1m(dim-col G1 * dim-col G2))
shows ((G1  $\otimes$  G2)† * (G1  $\otimes$  G2)) $$ (i, j) = 1m(dim-col G1 * dim-col G2)
$$ (i, j)
proof-
have  $\bigwedge k. k < 2^{\lceil(m+n)} \implies \text{cnj}((G1 \otimes G2) $$ (k, i)) =$ 
 $\text{cnj}(G1 $$ (k \text{ div } 2^{\lceil n}), i \text{ div } 2^{\lceil n})) * \text{cnj}(G2 $$ (k \text{ mod } 2^{\lceil n}), i \text{ mod } 2^{\lceil n}))$ 
using assms(1–3) by (simp add: gate-def power-add)
moreover have  $\bigwedge k. k < 2^{\lceil(m+n)} \implies (G1 \otimes G2) $$ (k, j) =$ 
 $G1 $$ (k \text{ div } 2^{\lceil n}, j \text{ div } 2^{\lceil n}) * G2 $$ (k \text{ mod } 2^{\lceil n}, j \text{ mod } 2^{\lceil n})$ 
using assms(1,2,4) by (simp add: gate-def power-add)
ultimately have  $\bigwedge k. k < 2^{\lceil(m+n)} \implies \text{cnj}((G1 \otimes G2) $$ (k, i)) * ((G1 \otimes G2) $$ (k, j)) =$ 
 $\text{cnj}(G1 $$ (k \text{ div } 2^{\lceil n}, i \text{ div } 2^{\lceil n})) * G1 $$ (k \text{ div } 2^{\lceil n}, j \text{ div } 2^{\lceil n}) *$ 
 $\text{cnj}(G2 $$ (k \text{ mod } 2^{\lceil n}, i \text{ mod } 2^{\lceil n})) * G2 $$ (k \text{ mod } 2^{\lceil n}, j \text{ mod } 2^{\lceil n})$  by simp
then have  $(\sum k < 2^{\lceil(m+n)}. \text{cnj}((G1 \otimes G2) $$ (k, i)) * ((G1 \otimes G2) $$ (k, j))) =$ 
 $= (\sum k < 2^{\lceil(m+n)}. \text{cnj}(G1 $$ (k \text{ div } 2^{\lceil n}, i \text{ div } 2^{\lceil n})) * G1 $$ (k \text{ div } 2^{\lceil n}, j \text{ div } 2^{\lceil n}) *$ 
 $\text{cnj}(G2 $$ (k \text{ mod } 2^{\lceil n}, i \text{ mod } 2^{\lceil n})) * G2 $$ (k \text{ mod } 2^{\lceil n}, j \text{ mod } 2^{\lceil n}))$  by simp

```

```

also have ... =
  ( $\sum k < 2^m. cnj(G1 \$\$ (k, i \text{ div } 2^n)) * G1 \$\$ (k, j \text{ div } 2^n)$ ) *
  ( $\sum k < 2^n. cnj(G2 \$\$ (k, i \text{ mod } 2^n)) * G2 \$\$ (k, j \text{ mod } 2^n)$ )
  using sum-prod[of  $\lambda x. cnj(G1 \$\$ (x, i \text{ div } 2^n)) * G1 \$\$ (x, j \text{ div } 2^n) 2^n$ 
 $\lambda x. cnj(G2 \$\$ (x, i \text{ mod } 2^n)) * G2 \$\$ (x, j \text{ mod } 2^n) 2^m]$ 
  by (metis (no-types, lifting) power-add semigroup-mult-class.mult.assoc sum.cong)
  also have  $((G1 \otimes G2)^\dagger * (G1 \otimes G2)) \$\$ (i, j) = (\sum k < 2^{(m+n)}. cnj((G1 \otimes G2) \$\$ (k,i)) * ((G1 \otimes G2) \$\$ (k,j)))$ 
  using assms index-matrix-prod[of  $i (G1 \otimes G2)^\dagger j (G1 \otimes G2)$ ] dagger-def
  dim-row-of-tensor-gate tensor-gate-sqr-mat by simp
  ultimately have  $((G1 \otimes G2)^\dagger * (G1 \otimes G2)) \$\$ (i,j) =$ 
     $(\sum k1 < 2^m. cnj(G1 \$\$ (k1, i \text{ div } 2^n)) * G1 \$\$ (k1, j \text{ div } 2^n)) *$ 
     $(\sum k2 < 2^n. cnj(G2 \$\$ (k2, i \text{ mod } 2^n)) * G2 \$\$ (k2, j \text{ mod } 2^n))$  by
  simp
  moreover have  $(\sum k < 2^m. cnj(G1 \$\$ (k, i \text{ div } 2^n)) * G1 \$\$ (k, j \text{ div } 2^n))$ 
=  $(G1^\dagger * G1) \$\$ (i \text{ div } 2^n, j \text{ div } 2^n)$ 
  using assms gate-def dagger-def index-matrix-prod[of  $i \text{ div } 2^n G1^\dagger j \text{ div } 2^n G1$ ]
  by (simp add: less-mult-imp-div-less power-add)
  moreover have ... =  $1_m(2^m) \$\$ (i \text{ div } 2^n, j \text{ div } 2^n)$ 
  using assms(1,2) gate-def unitary-def by simp
  moreover have  $(\sum k < 2^n. cnj(G2 \$\$ (k, i \text{ mod } 2^n)) * G2 \$\$ (k, j \text{ mod } 2^n))$ 
=  $(G2^\dagger * G2) \$\$ (i \text{ mod } 2^n, j \text{ mod } 2^n)$ 
  using assms(1,2) gate-def dagger-def index-matrix-prod[of  $i \text{ mod } 2^n G2^\dagger j \text{ mod } 2^n G2$ ] by simp
  moreover have ... =  $1_m(2^n) \$\$ (i \text{ mod } 2^n, j \text{ mod } 2^n)$ 
  using assms(1,2) gate-def unitary-def by simp
  ultimately have  $((G1 \otimes G2)^\dagger * (G1 \otimes G2)) \$\$ (i,j) = 1_m(2^m) \$\$ (i \text{ div } 2^n, j \text{ div } 2^n) * 1_m(2^n) \$\$ (i \text{ mod } 2^n, j \text{ mod } 2^n)$ 
  by simp
  thus ?thesis
  using assms assms(3,4) gate-def index-one-mat-div-mod[of  $i m n j$ ] by (simp
add: power-add)
qed

```

```

lemma tensor-gate-unitary1 [simp]:
assumes gate m G1 and gate n G2
shows  $(G1 \otimes G2)^\dagger * (G1 \otimes G2) = 1_m(\text{dim-col } G1 * \text{dim-col } G2)$ 
proof
  show dim-row  $((G1 \otimes G2)^\dagger * (G1 \otimes G2)) = \text{dim-row } (1_m(\text{dim-col } G1 * \text{dim-col } G2))$  by simp
  show dim-col  $((G1 \otimes G2)^\dagger * (G1 \otimes G2)) = \text{dim-col } (1_m(\text{dim-col } G1 * \text{dim-col } G2))$  by simp
  fix i j assume i < dim-row  $(1_m(\text{dim-col } G1 * \text{dim-col } G2))$  and j < dim-col  $(1_m(\text{dim-col } G1 * \text{dim-col } G2))$ 
  thus  $((G1 \otimes G2)^\dagger * (G1 \otimes G2)) \$\$ (i, j) = 1_m(\text{dim-col } G1 * \text{dim-col } G2) \$\$ (i, j)$ 
  using assms index-tensor-gate-unitary1 by simp
qed

```

lemma *index-tensor-gate-unitary2* [*simp*]:
assumes gate m $G1$ **and** gate n $G2$ **and** $i < \text{dim-row} (1_m (\text{dim-col } G1 * \text{dim-col } G2))$ **and**
 $j < \text{dim-col} (1_m (\text{dim-col } G1 * \text{dim-col } G2))$
shows $((G1 \otimes G2) * ((G1 \otimes G2)^\dagger)) \$\$ (i, j) = 1_m (\text{dim-col } G1 * \text{dim-col } G2) \$\$ (i, j)$

proof–

have $\bigwedge k. k < 2^{\lceil m+n \rceil} \implies (G1 \otimes G2) \$\$ (i, k) =$
 $G1 \$\$ (i \text{ div } 2^{\lceil n \rceil}, k \text{ div } 2^{\lceil n \rceil}) * G2 \$\$ (i \bmod 2^{\lceil n \rceil}, k \bmod 2^{\lceil n \rceil})$
using *assms(1–3)* **by** (*simp add: gate-def power-add*)
moreover have $\bigwedge k. k < 2^{\lceil m+n \rceil} \implies \text{cnj}((G1 \otimes G2) \$\$ (j, k)) =$
 $\text{cnj}(G1 \$\$ (j \text{ div } 2^{\lceil n \rceil}, k \text{ div } 2^{\lceil n \rceil}) * \text{cnj}(G2 \$\$ (j \bmod 2^{\lceil n \rceil}, k \bmod 2^{\lceil n \rceil}))$
using *assms(1,2,4)* **by** (*simp add: gate-def power-add*)
ultimately have $\bigwedge k. k \in \{.. < 2^{\lceil m+n \rceil}\} \implies (G1 \otimes G2) \$\$ (i, k) * \text{cnj}((G1 \otimes G2) \$\$ (j, k)) =$
 $G1 \$\$ (i \text{ div } 2^{\lceil n \rceil}, k \text{ div } 2^{\lceil n \rceil}) * \text{cnj}(G1 \$\$ (j \text{ div } 2^{\lceil n \rceil}, k \text{ div } 2^{\lceil n \rceil})) *$
 $G2 \$\$ (i \bmod 2^{\lceil n \rceil}, k \bmod 2^{\lceil n \rceil}) * \text{cnj}(G2 \$\$ (j \bmod 2^{\lceil n \rceil}, k \bmod 2^{\lceil n \rceil}))$ **by simp**
then have $(\sum k < 2^{\lceil m+n \rceil}. (G1 \otimes G2) \$\$ (i, k) * \text{cnj}((G1 \otimes G2) \$\$ (j, k))) =$
 $= (\sum k < 2^{\lceil m+n \rceil}. G1 \$\$ (i \text{ div } 2^{\lceil n \rceil}, k \text{ div } 2^{\lceil n \rceil}) * \text{cnj}(G1 \$\$ (j \text{ div } 2^{\lceil n \rceil}, k \text{ div } 2^{\lceil n \rceil})) *$
 $G2 \$\$ (i \bmod 2^{\lceil n \rceil}, k \bmod 2^{\lceil n \rceil}) * \text{cnj}(G2 \$\$ (j \bmod 2^{\lceil n \rceil}, k \bmod 2^{\lceil n \rceil}))$ **by simp**
also have $\dots =$
 $(\sum k < 2^m. G1 \$\$ (i \text{ div } 2^{\lceil n \rceil}, k) * \text{cnj}(G1 \$\$ (j \text{ div } 2^{\lceil n \rceil}, k))) *$
 $(\sum k < 2^{\lceil n \rceil}. G2 \$\$ (i \bmod 2^{\lceil n \rceil}, k) * \text{cnj}(G2 \$\$ (j \bmod 2^{\lceil n \rceil}, k)))$
using *sum-prod*[of $\lambda k. G1 \$\$ (i \text{ div } 2^{\lceil n \rceil}, k) * \text{cnj}(G1 \$\$ (j \text{ div } 2^{\lceil n \rceil}, k)) 2^{\lceil n \rceil}$
 $\lambda k. G2 \$\$ (i \bmod 2^{\lceil n \rceil}, k) * \text{cnj}(G2 \$\$ (j \bmod 2^{\lceil n \rceil}, k)) 2^{\lceil n \rceil}]$
by (*metis (no-types, lifting) power-add semigroup-mult-class.mult.assoc sum.cong*)
also have $((G1 \otimes G2) * ((G1 \otimes G2)^\dagger)) \$\$ (i, j) = (\sum k < 2^{\lceil m+n \rceil}. (G1 \otimes G2) \$\$ (i, k) * \text{cnj}((G1 \otimes G2) \$\$ (j, k)))$
using *assms index-matrix-prod*[of $i (G1 \otimes G2) j (G1 \otimes G2)^\dagger$] **dagger-def**
dim-row-of-tensor-gate tensor-gate-sqr-mat **by simp**
ultimately have $((G1 \otimes G2) * ((G1 \otimes G2)^\dagger)) \$\$ (i, j) =$
 $(\sum k < 2^m. G1 \$\$ (i \text{ div } 2^{\lceil n \rceil}, k) * \text{cnj}(G1 \$\$ (j \text{ div } 2^{\lceil n \rceil}, k))) *$
 $(\sum k < 2^{\lceil n \rceil}. G2 \$\$ (i \bmod 2^{\lceil n \rceil}, k) * \text{cnj}(G2 \$\$ (j \bmod 2^{\lceil n \rceil}, k)))$ **by simp**
moreover have $(\sum k < 2^m. G1 \$\$ (i \text{ div } 2^{\lceil n \rceil}, k) * \text{cnj}(G1 \$\$ (j \text{ div } 2^{\lceil n \rceil}, k)))$
 $= (G1 * (G1^\dagger)) \$\$ (i \text{ div } 2^{\lceil n \rceil}, j \text{ div } 2^{\lceil n \rceil})$
using *assms gate-def dagger-def index-matrix-prod*[of $i \text{ div } 2^{\lceil n \rceil} G1 j \text{ div } 2^{\lceil n \rceil} G1^\dagger$]
by (*simp add: less-mult-imp-div-less power-add*)
moreover have $\dots = 1_m (2^{\lceil m+n \rceil}) \$\$ (i \text{ div } 2^{\lceil n \rceil}, j \text{ div } 2^{\lceil n \rceil})$
using *assms(1,2)* **gate-def unitary-def** **by simp**
moreover have $(\sum k < 2^{\lceil n \rceil}. G2 \$\$ (i \bmod 2^{\lceil n \rceil}, k) * \text{cnj}(G2 \$\$ (j \bmod 2^{\lceil n \rceil}, k)))$
 $= (G2 * (G2^\dagger)) \$\$ (i \bmod 2^{\lceil n \rceil}, j \bmod 2^{\lceil n \rceil})$
using *assms(1,2)* **gate-def dagger-def index-matrix-prod**[of $i \bmod 2^{\lceil n \rceil} G2 j \bmod G2^\dagger$]

```

 $2^{\wedge}n \ G2^{\dagger}]$  by simp
moreover have ... =  $1_m(2^{\wedge}n) \ \$\$ (i \ mod \ 2^{\wedge}n, j \ mod \ 2^{\wedge}n)$ 
  using assms(1,2) gate-def unitary-def by simp
ultimately have  $((G1 \otimes \ G2) * ((G1 \otimes \ G2)^{\dagger})) \ \$\$ (i,j) = 1_m(2^{\wedge}m) \ \$\$ (i \ div$ 
 $2^{\wedge}n, j \ div \ 2^{\wedge}n) * 1_m(2^{\wedge}n) \ \$\$ (i \ mod \ 2^{\wedge}n, j \ mod \ 2^{\wedge}n)$ 
  by simp
thus ?thesis
using assms gate-def index-one-mat-div-mod[of i m n j] by(simp add: power-add)
qed

lemma tensor-gate-unitary2 [simp]:
assumes gate m G1 and gate n G2
shows  $(G1 \otimes \ G2) * ((G1 \otimes \ G2)^{\dagger}) = 1_m(dim\text{-}col \ G1 * dim\text{-}col \ G2)$ 
proof
show dim-row  $((G1 \otimes \ G2) * ((G1 \otimes \ G2)^{\dagger})) = dim\text{-}row(1_m(dim\text{-}col \ G1 * dim\text{-}col \ G2))$ 
  using assms gate-def by simp
show dim-col  $((G1 \otimes \ G2) * ((G1 \otimes \ G2)^{\dagger})) = dim\text{-}col(1_m(dim\text{-}col \ G1 * dim\text{-}col \ G2))$ 
  using assms gate-def by simp
fix i j assume i < dim-row  $(1_m(dim\text{-}col \ G1 * dim\text{-}col \ G2))$  and j < dim-col
 $(1_m(dim\text{-}col \ G1 * dim\text{-}col \ G2))$ 
thus  $((G1 \otimes \ G2) * ((G1 \otimes \ G2)^{\dagger})) \ \$\$ (i, j) = 1_m(dim\text{-}col \ G1 * dim\text{-}col \ G2)$ 
\$ (i, j)
using assms index-tensor-gate-unitary2 by simp
qed

lemma tensor-gate [simp]:
assumes gate m G1 and gate n G2
shows gate  $(m + n) (G1 \otimes \ G2)$ 
proof
show dim-row  $(G1 \otimes \ G2) = 2^{\wedge}(m+n)$ 
  using assms dim-row-tensor-mat gate.dim-row by (simp add: power-add)
show square-mat  $(G1 \otimes \ G2)$ 
  using assms gate.square-mat by simp
thus unitary  $(G1 \otimes \ G2)$ 
  using assms unitary-def by simp
qed

end

```

7 Measurement

```

theory Measurement
imports
  Quantum
begin

```

Given an element v such that *state n v*, its components $v \$ i$ (when v is seen

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

definition *select-index* :: $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$ **where**

select-index $n\ i\ j \equiv (i \leq n - 1) \wedge (j \leq 2^{\lceil n - 1 \rceil}) \wedge (j \bmod 2^{\lceil n - i \rceil} \geq 2^{\lceil n - 1 - i \rceil})$

lemma *select-index-union*:

$\{k \mid k : \text{nat}. \text{ select-index } n\ i\ k\} \cup \{k \mid k : \text{nat}. (k < 2^{\lceil n \rceil}) \wedge \neg \text{ select-index } n\ i\ k\} = \{0..<2^{\lceil n \rceil} : \text{nat}\}$

proof

have $\{k \mid k. \text{ select-index } n\ i\ k\} \subseteq \{0..<2^{\lceil n \rceil}\}$

proof

fix $x : \text{nat}$ **assume** $x \in \{k \mid k. \text{ select-index } n\ i\ k\}$

then show $x \in \{0..<2^{\lceil n \rceil}\}$

using *select-index-def*

by (*metis (no-types, lifting)* *atLeastLessThan-iff* *diff-diff-cancel* *diff-is-0-eq'*)

```

diff-le-mono2
le-less-linear le-numeral-extra(2) mem-Collect-eq one-le-numeral one-le-power se-
lect-index-def zero-order(1))
qed
moreover have {k | k. k < 2^n ∧ ¬ select-index n i k} ⊆ {0..<2^n} by auto
ultimately show {k | k. select-index n i k} ∪ {k | k. k < 2^n ∧ ¬ select-index n
i k} ⊆ {0..<2^n} by simp
next
show {0..<2^n} ⊆ {k | k. select-index n i k} ∪ {k | k. k < 2^n ∧ ¬ select-index
n i k} by auto
qed

lemma select-index-inter:
{k | k::nat. select-index n i k} ∩ {k | k::nat. (k < 2^n) ∧ ¬ select-index n i k} = {}
by auto

lemma outcomes-sum [simp]:
fixes f :: nat ⇒ real
shows
(∑ j ∈ {k | k::nat. select-index n i k}. (f j)) +
(∑ j ∈ {k | k::nat. (k < 2^n) ∧ ¬ select-index n i k}. (f j)) =
(∑ j ∈ {0..<2^n::nat}. (f j))
proof –
have {k | k::nat. select-index n i k} ⊆ {0..<2^n::nat}
using select-index-union by blast
then have finite {k | k::nat. select-index n i k}
using rev-finite-subset by blast
moreover have {k | k::nat. (k < 2^n) ∧ ¬ select-index n i k} ⊆ {0..<2^n::nat}
using select-index-union by blast
then have finite {k | k::nat. (k < 2^n) ∧ ¬ select-index n i k}
using rev-finite-subset by blast
ultimately have (∑ j ∈ {k | k::nat. select-index n i k} ∪ {k | k::nat. (k < 2^n) ∧
¬ select-index n i k}. (f j)) =
(∑ j ∈ {k | k::nat. select-index n i k}. (f j)) +
(∑ j ∈ {k | k::nat. (k < 2^n) ∧ ¬ select-index n i k}. (f j)) −
(∑ j ∈ {k | k::nat. select-index n i k} ∩ {k | k::nat. (k < 2^n) ∧ ¬ select-index
n i k}. (f j))
using sum-Un by blast
then have (∑ j ∈ {0..<2^n::nat}. (f j)) =
(∑ j ∈ {k | k::nat. select-index n i k}. (f j)) +
(∑ j ∈ {k | k::nat. (k < 2^n) ∧ ¬ select-index n i k}. (f j)) −
(∑ j ∈ {}. (f j))
using select-index-union select-index-inter by simp
thus ?thesis by simp
qed

```

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

```
definition prob1 ::nat ⇒ complex mat ⇒ nat ⇒ real where
```

```

prob1 n v i ≡ ∑ j∈{k| k::nat. select-index n i k}. (cmod(v $$ (j,0)))2

definition prob0 ::nat ⇒ complex mat ⇒ nat ⇒ real where
prob0 n v i ≡ ∑ j∈{k| k::nat. (k < 2^n) ∧ ¬ select-index n i k}. (cmod(v $$ (j,0)))2

lemma
  shows prob1-geq-zero:prob1 n v i ≥ 0 and prob0-geq-zero:prob0 n v i ≥ 0
proof –
  have (∑ j∈{k| k::nat. select-index n i k}. (cmod(v $$ (j,0)))2) ≥
    (∑ j∈{k| k::nat. select-index n i k}. (0::real))
  by (simp add: sum-nonneg)
  then have (∑ j∈{k| k::nat. select-index n i k}. (cmod(v $$ (j,0)))2) ≥ 0 by
  simp
  thus prob1 n v i ≥ 0
  using prob1-def by simp
next
  have (∑ j∈{k| k::nat. (k < 2^n) ∧ ¬ select-index n i k}. (cmod(v $$ (j,0)))2)
  ≥
    (∑ j∈{k| k::nat. (k < 2^n) ∧ ¬ select-index n i k}. (0::real))
  by (simp add: sum-nonneg)
  then have (∑ j∈{k| k::nat. (k < 2^n) ∧ ¬ select-index n i k}. (cmod(v $$ (j,0)))2) ≥ 0 by simp
  thus prob0 n v i ≥ 0
  using prob0-def by simp
qed

lemma prob-sum-is-one [simp]:
  assumes state n v
  shows prob1 n v i + prob0 n v i = 1
proof –
  have prob1 n v i + prob0 n v i = (∑ j∈{0..<2^n:nat}. (cmod(v $$ (j,0)))2)
  using prob1-def prob0-def outcomes-sum by simp
  also have ... = ‖col v 0‖2
  using cpx-vec-length-def assms state-def atLeast0LessThan by fastforce
  finally show ?thesis
  using assms state-def by simp
qed

lemma
  assumes state n v
  shows prob1-leq-one:prob1 n v i ≤ 1 and prob0-leq-one:prob0 n v i ≤ 1
  apply(metis assms le-add-same-cancel1 prob0-geq-zero prob-sum-is-one)
  apply(metis assms le-add-same-cancel2 prob1-geq-zero prob-sum-is-one)
  done

lemma prob0-is-prob:
  assumes state n v
  shows prob0 n v i ≥ 0 ∧ prob0 n v i ≤ 1
  by (simp add: assms prob0-geq-zero prob0-leq-one)

```

```

lemma prob1-is-prob:
  assumes state n v
  shows prob1 n v i  $\geq 0 \wedge$  prob1 n v i  $\leq 1$ 
  by (simp add: assms prob1-geq-zero prob1-leq-one)

```

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

```

definition post-meas0 ::nat  $\Rightarrow$  complex mat  $\Rightarrow$  nat  $\Rightarrow$  complex mat where
post-meas0 n v i  $\equiv$ 
  of-real(1/sqrt(prob0 n v i))  $\cdot_m$  |vec (2^n) ( $\lambda j$ . if  $\neg$  select-index n i j then v $$ (j,0) \text{ else } 0)

```

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

```

lemma post-meas0-is-state [simp]:
  assumes state n v and prob0 n v i  $\neq 0$ 
  shows state n (post-meas0 n v i)
proof -
  have  $(\sum_{j \in \{0..<2^n\}} (\text{cmod} (\text{if } \neg \text{select-index } n \ i \ j \text{ then } v \ $$ (j,0) \text{ else } 0))^2) =$ 
     $(\sum_{j \in \{k \mid k \in \text{nat}\}} \text{select-index } n \ i \ k) \cdot (\text{cmod} (\text{if } \neg \text{select-index } n \ i \ j \text{ then } v \ $$ (j,0) \text{ else } 0))^2 +$ 
     $(\sum_{j \in \{k \mid k \in \text{nat}\} \wedge \neg \text{select-index } n \ i \ k} (\text{cmod} (\text{if } \neg \text{select-index } n \ i \ j \text{ then } v \ $$ (j,0) \text{ else } 0))^2)$ 
  using outcomes-sum[of  $\lambda j$ . (cmod (if  $\neg$  select-index n i j then v $$ (j,0) \text{ else } 0))^2 n i] by simp
  moreover have  $(\sum_{j \in \{k \mid k \in \text{nat}\} \wedge \neg \text{select-index } n \ i \ k} (\text{cmod} (\text{if } \neg \text{select-index } n \ i \ j \text{ then } v \ $$ (j,0) \text{ else } 0))^2) =$ 
    prob0 n v i
  by (simp add: prob0-def)
  ultimately have  $\|\text{vec} (2^n) (\lambda j. \text{if } \neg \text{select-index } n \ i \ j \text{ then } v \ $$ (j,0) \text{ else } 0)\| =$ 
   $= \sqrt(\text{prob0 } n \ v \ i)$ 
  using lessThan-atLeast0 by (simp add: cpx-vec-length-def)
  moreover have  $\|\text{col} (\text{complex-of-real} (1/\sqrt(\text{prob0 } n \ v \ i)) \cdot_m \text{vec} (2^n) (\lambda j. \text{if } \neg \text{select-index } n \ i \ j \text{ then } v \ $$ (j,0) \text{ else } 0))\| =$ 
   $= (1/\sqrt(\text{prob0 } n \ v \ i)) * \|\text{vec} (2^n) (\lambda j. \text{if } \neg \text{select-index } n \ i \ j \text{ then } v \ $$ (j,0) \text{ else } 0)\|$ 
  using prob0-geq-zero smult-vec-length-bis by (metis (no-types, lifting) real-sqrt-ge-0-iff zero-le-divide-1-iff)
  ultimately show ?thesis
  using state-def post-meas0-def by (simp add: ket-vec-def post-meas0-def assms(2))
qed

```

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

```

definition post-meas1 ::nat  $\Rightarrow$  complex mat  $\Rightarrow$  nat  $\Rightarrow$  complex mat where

```

```

post-meas1 n v i ≡
  of-real(1/sqrt(prob1 n v i)) ·m |vec (2n) (λj. if select-index n i j then v $$ (j,0)
else 0)⟩

```

Note that a division by 0 never occurs. Indeed, if $\sqrt{\text{prob1 } n \text{ } v \text{ } i}$ would be 0 then $\text{prob1 } n \text{ } v \text{ } i$ would be 0 and it would mean that the measurement of the i th qubit gave 0.

```

lemma post-meas1-is-state [simp]:
  assumes state n v and prob1 n v i ≠ 0
  shows state n (post-meas1 n v i)
proof –
  have (∑ j ∈ {0..<2n:nat}. (cmod (if select-index n i j then v $$ (j,0) else 0))2)
  =
  (∑ j ∈ {k| k:nat. select-index n i k}. (cmod (if select-index n i j then v $$ (j,0) else 0))2) +
  (∑ j ∈ {k| k:nat. (k<2n) ∧ ¬ select-index n i k}. (cmod (if select-index n i j then v $$ (j,0) else 0))2)
  using outcomes-sum[of λj. (cmod (if select-index n i j then v $$ (j,0) else 0))2
n i] by simp
  then have ||vec (2n) (λj. if select-index n i j then v $$ (j,0) else 0)|| = sqrt(prob1
n v i)
  using lessThan-atLeast0 by (simp add: cpx-vec-length-def prob1-def)
  moreover have ||col(complex-of-real (1/sqrt (prob1 n v i)) ·m |vec (2n) (λj. if
select-index n i j then v $$ (j,0) else 0))) 0|| =
  (1/sqrt(prob1 n v i)) * ||vec (2n) (λj. if select-index n i j then v $$ (j,0)
else 0) ||
  using prob1-geq-zero smult-vec-length-bis
  by (metis (no-types, lifting) real-sqrt-ge-0-iff zero-le-divide-1-iff)
  ultimately have ||col(complex-of-real (1/sqrt (prob1 n v i)) ·m |vec (2n) (λj.
if select-index n i j then v $$ (j,0) else 0))) 0||
  = (1/sqrt(prob1 n v i)) * sqrt(prob1 n v i) by simp
  thus ?thesis
  using state-def post-meas1-def by (simp add: ket-vec-def post-meas1-def assms(2))
qed

```

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

```

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

```

We want to determine the probability that the first n qubits of an $n+1$ qubit system are 0. For this we need to find the indices of the states of the

computational basis whose labels do not have a 1 at spot $i = 0, \dots, n$.

```

definition prob0-fst-qubits:: nat  $\Rightarrow$  complex Matrix.mat  $\Rightarrow$  real where
prob0-fst-qubits n v  $\equiv$ 
 $\sum_{j \in \{k \mid k::nat\}} (k < 2^{\lceil(n+1\rceil)}) \wedge (\forall i \in \{0..n\}. \neg select\text{-index} (n+1) i k) \cdot (cmod(v \text{``\$\$''} (j, 0)))^2$ 

lemma select-index-div-2:
  fixes n i j::nat
  assumes i  $< 2^{\lceil(n+1\rceil)}$  and j  $< n$ 
  shows select-index n j (i div 2) = select-index (n+1) j i
proof-
  have  $2^{\lceil(n-Suc j)\rceil} \leq i \text{ div } 2 \text{ mod } 2^{\lceil(n-j)\rceil} \implies 2^{\lceil(n-j)\rceil} \leq i \text{ mod } 2^{\lceil(n+1-j)\rceil}$ 
  proof-
    define a::nat where a0:a = i div 2 mod 2^{\lceil(n-j)\rceil}
    assume 2^{\lceil(n-Suc j)\rceil}  $\leq a$ 
    then have 2*a + i mod 2  $\geq 2^{\lceil(n-(Suc j)+1)\rceil}$  by simp
    then have f0:2*a + i mod 2  $\geq 2^{\lceil(n-j)\rceil}$ 
      by (metis Suc-diff-Suc Suc-eq-plus1 assms(2))
    have a  $< 2^{\lceil(n-j)\rceil}$  using a0 by simp
    then have 2*a + i mod 2  $< 2*2^{\lceil(n-j)\rceil}$  by linarith
    then have 2*a + i mod 2  $< 2^{\lceil(n-j+1)\rceil}$  by simp
    then have f1:2*a + i mod 2  $< 2^{\lceil(n+1-j)\rceil}$ 
      by (metis Nat.add-diff-assoc2 Suc-leD Suc-leI assms(2))
    have i = 2*(a + 2^{\lceil(n-j)\rceil}*(i div 2 div 2^{\lceil(n-j)\rceil})) + i mod 2 using a0 by simp
    then have i = 2*a + i mod 2 + 2^{\lceil(n-j+1)\rceil}*(i div 2 div 2^{\lceil(n-j)\rceil}) by simp
    then have i = 2*a + i mod 2 + 2^{\lceil(n+1-j)\rceil}*(i div 2 div 2^{\lceil(n-j)\rceil})
      by (metis Nat.add-diff-assoc2 Suc-leD Suc-leI assms(2))
    then have i mod 2^{\lceil(n+1-j)\rceil} = 2*a + i mod 2
      using f1 by (metis mod-if mod-mult-self2)
    then show 2^{\lceil(n-j)\rceil}  $\leq i \text{ mod } 2^{\lceil(n+1-j)\rceil}$ 
      using f0 by simp
  qed
  moreover have 2^{\lceil(n-j)\rceil}  $\leq i \text{ mod } 2^{\lceil(n+1-j)\rceil} \implies 2^{\lceil(n-Suc j)\rceil} \leq i \text{ div } 2 \text{ mod } 2^{\lceil(n-j)\rceil}$ 
  proof-
    define a::nat where a0:a = i div 2 mod 2^{\lceil(n-j)\rceil}
    assume a1:2^{\lceil(n-j)\rceil}  $\leq i \text{ mod } 2^{\lceil(n+1-j)\rceil}$ 
    have f0:2^{\lceil(n-j)\rceil} = 2^{\lceil(n-Suc j+1)\rceil}
      by (metis Suc-diff-Suc Suc-eq-plus1 assms(2))
    have a  $< 2^{\lceil(n-j)\rceil}$  using a0 by simp
    then have 2*a + i mod 2  $< 2*2^{\lceil(n-j)\rceil}$  by linarith
    then have 2*a + i mod 2  $< 2^{\lceil(n-j+1)\rceil}$  by simp
    then have f1:2*a + i mod 2  $< 2^{\lceil(n+1-j)\rceil}$ 
      by (metis Nat.add-diff-assoc2 Suc-leD Suc-leI assms(2))
    have i = 2*(a + 2^{\lceil(n-j)\rceil}*(i div 2 div 2^{\lceil(n-j)\rceil})) + i mod 2 using a0 by simp
    then have i = 2*a + i mod 2 + 2^{\lceil(n-j+1)\rceil}*(i div 2 div 2^{\lceil(n-j)\rceil}) by simp
    then have i = 2*a + i mod 2 + 2^{\lceil(n+1-j)\rceil}*(i div 2 div 2^{\lceil(n-j)\rceil})
      by (metis Nat.add-diff-assoc2 Suc-leD Suc-leI assms(2))
    then have i mod 2^{\lceil(n+1-j)\rceil} = 2*a + i mod 2
  
```

```

using f1 by (metis mod-if mod-mult-self2)
then have  $2*a + i \text{ mod } 2 \geq 2^{\lceil n-j \rceil}$ 
  using a1 by simp
then have  $(2*a + i \text{ mod } 2) \text{ div } 2 \geq (2^{\lceil n-j \rceil}) \text{ div } 2$ 
  using div-le-mono by blast
then show  $2^{\lceil n-Suc j \rceil} \leq a$  by (simp add: f0)
qed
ultimately show ?thesis
  using select-index-def assms by auto
qed

lemma select-index-suc-even:
fixes n k i:: nat
assumes k <  $2^{\lceil n \rceil}$  and select-index n i k
shows select-index (Suc n) i ( $2*k$ )
proof-
have select-index n i k = select-index n i ( $2*k \text{ div } 2$ ) by simp
moreover have ... = select-index (Suc n) i ( $2*k$ )
proof-
have i < n using assms(2) select-index-def
  by (metis (no-types, opaque-lifting) Suc-eq-plus1 assms(1) calculation diff-diff-left
diff-le-self
diff-self-eq-0 div-by-1 le-0-eq le-eq-less-or-eq less-imp-diff-less mod-div-trivial mult.left-neutral
mult-eq-0-iff mult-le-mono1 not-less plus-1-eq-Suc power-0 semiring-normalization-rules(7))
thus ?thesis
  using select-index-div-2 assms(1) select-index-def by (metis Suc-1 Suc-eq-plus1
Suc-mult-less-cancel1 power-Suc)
qed
ultimately show select-index (Suc n) i ( $2*k$ )
  using assms(2) by simp
qed

lemma select-index-suc-odd:
fixes n k i:: nat
assumes k ≤  $2^{\lceil n \rceil} - 1$  and select-index n i k
shows select-index (Suc n) i ( $2*k+1$ )
proof-
have  $((2*k+1) \text{ mod } 2^{\lceil n-i \rceil} \geq 2^{\lceil n-i \rceil}) =$ 
 $((2*k+1) \text{ div } 2) \text{ mod } 2^{\lceil n-i \rceil} \geq 2^{\lceil n-1-i \rceil})$ 
proof-
have  $2*k+1 < 2^{\lceil n+1 \rceil}$ 
  using assms(1)
  by (smt (verit) Suc-1 Suc-eq-plus1 Suc-le-lessD Suc-le-mono add-Suc-right dis-
trib-left-numeral le-add-diff-inverse mult-le-mono2 nat-mult-1-right one-le-numeral
one-le-power plus-1-eq-Suc power-add power-one-right)
moreover have i < n
  using assms(2) select-index-def
  by (metis (no-types, opaque-lifting) add-cancel-left-left add-diff-inverse-nat
diff-le-self div-by-1 le-antisym less-le-trans less-one mod-div-trivial not-le power-0)

```

```

ultimately show ?thesis
  using select-index-div-2[of 2*k+1 n i] select-index-def
  by (metis Nat.le-diff-conv2 Suc-eq-plus1 Suc-leI assms(2) diff-Suc-1 less-imp-le
less-power-add-imp-div-less one-le-numeral one-le-power power-one-right)
qed
moreover have ... = (k mod 2^(n - i) ≥ 2^(n-1-i)) by simp
ultimately show ?thesis
proof-
  have i ≤ Suc n - 1 using assms(2) select-index-def by auto
  moreover have 2*k+1 ≤ 2^(Suc n)-1
    using assms(1) by (smt (verit) Suc-diff-1 Suc-eq-plus1 add-diff-cancel-right'
diff-Suc-diff-eq2 diff-diff-left diff-is-0-eq diff-mult-distrib2 le-add2 mult-2 mult-Suc-right
plus-1-eq-Suc pos2 power-Suc zero-less-power)
  ultimately show ?thesis
  using select-index-def
  by (metis <(2^(n - 1 - i) ≤ (2 * k + 1) div 2 mod 2^(n - i)) = (2^(n
- 1 - i) ≤ k mod 2^(n - i))> <(2^(n - i) ≤ (2 * k + 1) mod 2^(Suc n - i))
= (2^(n - 1 - i) ≤ (2 * k + 1) div 2 mod 2^(n - i))> assms(2) diff-Suc-1)
qed
qed

lemma aux-range:
fixes k:: nat
assumes k < 2^(Suc n + 1) and k ≥ 2
shows k = 2 ∨ k = 3 ∨ (∃ l. l ≥ 2 ∧ l ≤ 2^(n+1)-1 ∧ (k = 2*l ∨ k = 2*l + 1))
proof(rule disjCI)
assume ¬(k = 3 ∨ (∃ l ≥ 2. l ≤ 2^(n + 1) - 1 ∧ (k = 2 * l ∨ k = 2 * l + 1)))
have k > 3 → (∃ l ≥ 2. l ≤ 2^(n + 1) - 1 ∧ (k = 2 * l ∨ k = 2 * l + 1))
proof
assume asm:k > 3
have even k ∨ odd k by simp
then obtain l where k = 2*l ∨ k = 2*l+1 by (meson evenE oddE)
moreover have l ≥ 2
  using asm calculation by linarith
moreover have l ≤ 2^(n+1) - 1
  using assms(1) by (metis Suc-diff-1 Suc-eq-plus1 calculation(1) dvd-triv-left
even-Suc-div-two less-Suc-eq-le less-power-add-imp-div-less nonzero-mult-div-cancel-left
pos2 power-one-right zero-less-power zero-neq-numeral)
ultimately show ∃ l ≥ 2. l ≤ 2^(n + 1) - 1 ∧ (k = 2 * l ∨ k = 2 * l + 1)
by auto
qed
then have k ≤ 2
using <¬(k = 3 ∨ (∃ l ≥ 2. l ≤ 2^(n + 1) - 1 ∧ (k = 2 * l ∨ k = 2 * l +
1)))> less-Suc-eq-le by auto
thus k = 2
  using assms(2) by simp
qed

lemma select-index-with-1:

```

```

fixes n:: nat
assumes n ≥ 1
shows ∀ k. k < 2^(n+1) → k ≥ 2 → (∃ i<n. select-index (n+1) i k)
using assms
proof(rule nat-induct-at-least)
show ∀ k< 2^(1+1). 2 ≤ k → (∃ i<1. select-index (1+1) i k)
proof-
have select-index 2 0 2 = True
using select-index-def by simp
moreover have select-index 2 0 3
using select-index-def by simp
ultimately show ?thesis
by (metis Suc-leI add-Suc-shift le-eq-less-or-eq mult-2 not-less one-add-one
one-plus-numeral
plus-1-eq-Suc power.simps(2) power-one-right semiring-norm(3) zero-less-one-class.zero-less-one)
qed
next
show ∧n. 1 ≤ n ==>
  ∀ k < 2^(n+1). 2 ≤ k → (∃ i<n. select-index (n+1) i k) ==>
  ∀ k < 2^(Suc n + 1). 2 ≤ k → (∃ i<Suc n. select-index (Suc n + 1) i k)
proof-
fix n:: nat
assume asm:n ≥ 1 and IH:∀ k < 2^(n+1). 2 ≤ k → (∃ i<n. select-index
(n+1) i k)
have select-index (Suc n + 1) n 2
proof-
have select-index (Suc n) n 1
using select-index-def by (smt (verit) Suc-1 Suc-diff-Suc Suc-lessI add-diff-cancel-right'
diff-Suc-1
diff-commute diff-zero le-eq-less-or-eq less-Suc-eq-le nat.simps(3) nat-power-eq-Suc-0-iff
one-mod-two-eq-one plus-1-eq-Suc power-one-right zero-less-power)
thus ?thesis
using select-index-suc-even by (metis Suc-eq-plus1 less-numeral-extra(4)
mult-2 not-less-less-Suc-eq one-add-one one-less-power zero-less-Suc)
qed
moreover have select-index (Suc n + 1) n 3
proof-
have select-index (Suc n) n 1
using select-index-def by (smt (verit) Suc-1 Suc-diff-Suc Suc-lessI add-diff-cancel-right'
diff-Suc-1
diff-commute diff-zero le-eq-less-or-eq less-Suc-eq-le nat.simps(3) nat-power-eq-Suc-0-iff
one-mod-two-eq-one plus-1-eq-Suc power-one-right zero-less-power)
thus ?thesis
using select-index-suc-odd by (metis One-nat-def Suc-eq-plus1 mult-2 nu-
meral-3-eq-3 select-index-def)
qed
moreover have ∃ i<Suc n. select-index (Suc n + 1) i (2*k) if k ≥ 2 and k ≤

```

```

 $2^{\lceil n+1 \rceil} - 1$  for  $k :: nat$ 
proof-
  obtain  $i$  where  $i < n$  and  $\text{select-index } (n+1) i k$ 
    using  $IH$  by(metis One-nat-def Suc-diff-Suc  $\langle 2 \leq k \rangle \langle k \leq 2^{\lceil n+1 \rceil} - 1 \rangle$   $\text{diff-zero le-imp-less-Suc}$   $\text{pos2 zero-less-power}$ )
    then have  $\text{select-index } (\text{Suc } n+1) i (2*k)$ 
      using  $\text{select-index-suc-even}$ 
        by (metis One-nat-def Suc-diff-Suc add.commute diff-zero le-imp-less-Suc  $\text{plus-1-eq-Suc}$   $\text{pos2 that}(2)$   $\text{zero-less-power}$ )
    thus  $?thesis$ 
      using  $\langle i < n \rangle \text{ less-SucI}$  by blast
  qed
  moreover have  $\exists i < \text{Suc } n. \text{select-index } (\text{Suc } n+1) i (2*k+1)$  if  $k \geq 2$  and
 $k \leq 2^{\lceil n+1 \rceil} - 1$  for  $k :: nat$ 
proof-
  obtain  $i$  where  $i < n$  and  $\text{select-index } (n+1) i k$ 
    using  $IH$  by(metis One-nat-def Suc-diff-Suc  $\langle 2 \leq k \rangle \langle k \leq 2^{\lceil n+1 \rceil} - 1 \rangle$   $\text{diff-zero le-imp-less-Suc}$   $\text{pos2 zero-less-power}$ )
    then have  $\text{select-index } (\text{Suc } n+1) i (2*k+1)$ 
      using  $\text{select-index-suc-odd}$   $\text{that}(2)$  by simp
    thus  $?thesis$ 
      using  $\langle i < n \rangle \text{ less-SucI}$  by blast
  qed
  ultimately show  $\forall k < 2^{\lceil n+1 \rceil}. 2 \leq k \longrightarrow (\exists i < \text{Suc } n. \text{select-index } (\text{Suc } n+1) i k)$ 
    using  $\text{aux-range}$  by (metis lessI)
  qed
qed

lemma prob0-fst-qubits-index:
  fixes  $n :: nat$  and  $v :: \text{complex Matrix.mat}$ 
  shows  $\{k | k :: nat. (k < 2^{\lceil n+1 \rceil}) \wedge (\forall i \in \{0..<n\}. \neg \text{select-index } (n+1) i k)\} = \{0, 1\}$ 
proof(induct n)
  case 0
  show  $\{k | k. k < 2^{\lceil 0+1 \rceil} \wedge (\forall i \in \{0..<0\}. \neg \text{select-index } (0+1) i k)\} = \{0, 1\}$ 
by auto
next
  case  $(\text{Suc } n)$ 
  show  $\bigwedge n. \{k | k. k < 2^{\lceil n+1 \rceil} \wedge (\forall i \in \{0..<n\}. \neg \text{select-index } (n+1) i k)\} = \{0, 1\} \implies \{k | k. k < 2^{\lceil \text{Suc } n+1 \rceil} \wedge (\forall i \in \{0..<\text{Suc } n\}. \neg \text{select-index } (\text{Suc } n+1) i k)\} = \{0, 1\}$ 
proof-
  fix  $n$ 
  assume  $IH: \{k | k. k < 2^{\lceil n+1 \rceil} \wedge (\forall i \in \{0..<n\}. \neg \text{select-index } (n+1) i k)\} = \{0, 1\}$ 
  then have  $\{0, 1\} \subseteq \{k | k. k < 2^{\lceil \text{Suc } n+1 \rceil} \wedge (\forall i \in \{0..<\text{Suc } n\}. \neg \text{select-index }$ 

```

```

(Suc n + 1) i k)
proof-
  have  $k < 2^{\lceil n+1 \rceil} \rightarrow k < 2^{\lceil \text{Suc } n + 1 \rceil}$  for  $k::nat$  by simp
  moreover have  $(\forall i \in \{0..<n\}. \neg \text{select-index } (\text{Suc } n + 1) i 0) \wedge (\forall i \in \{0..<n\}. \neg \text{select-index } (\text{Suc } n + 1) i 1)$ 
    using IH by auto
    then have  $(\forall i \in \{0..<n\}. \neg \text{select-index } (\text{Suc } n + 1) i 0) \wedge (\forall i \in \{0..<n\}. \neg \text{select-index } (\text{Suc } n + 1) i 1)$ 
      using select-index-suc-odd[of 0 n+1] Suc-eq-plus1
      by (smt (verit) One-nat-def Suc-1 add-Suc-shift add-diff-cancel-right' atLeastLessThan-iff diff-diff-cancel
le-eq-less-or-eq less-Suc-eq linorder-not-le mod-less nat-power-eq-Suc-0-iff select-index-def zero-less-power)
  moreover have select-index (Suc n + 1) n 0 = False using select-index-def
by simp
  moreover have select-index (Suc n + 1) n 1 = False using select-index-def
by simp
  ultimately show ?thesis
  by (smt (verit) One-nat-def Suc-1 Suc-eq-plus1 Suc-lessI atLeast0-lessThan-Suc empty-iff insertE
mem-Collect-eq nat.simps(1) nat-power-eq-Suc-0-iff pos2 subsetI zero-less-power)
qed
  moreover have  $\{k \mid k. k < 2^{\lceil \text{Suc } n + 1 \rceil} \wedge (\forall i \in \{0..<\text{Suc } n\}. \neg \text{select-index } (\text{Suc } n + 1) i k)\} \subseteq \{0,1\}$ 
  proof-
    have  $\forall k < 2^{\lceil \text{Suc } n + 1 \rceil}. k \geq 2 \rightarrow (\exists i < \text{Suc } n. \neg \text{select-index } (\text{Suc } n + 1) i k = \text{False})$ 
      using select-index-with-1[of Suc n] by (metis Suc-eq-plus1 add.commute le-add1)
      thus ?thesis by auto
    qed
    ultimately show  $\{k \mid k. k < 2^{\lceil \text{Suc } n + 1 \rceil} \wedge (\forall i \in \{0..<\text{Suc } n\}. \neg \text{select-index } (\text{Suc } n + 1) i k)\} = \{0,1\}$  by auto
    qed
  qed

lemma prob0-fst-qubits-eq:
  fixes  $n::nat$ 
  shows prob0-fst-qubits n v = (cmod(v $$ (0,0)))^2 + (cmod(v $$ (1,0)))^2
proof-
  have prob0-fst-qubits n v = (∑ j ∈ {k | k::nat. (k < 2^(n+1)) ∧ (∀ i ∈ {0..<n}. ¬ select-index (n+1) i k)}. (cmod(v $$ (j,0)))^2)
    using prob0-fst-qubits-def by simp
  moreover have ...  $= (\sum j \in \{0,1\}. (cmod(v $$ (j,0)))^2)$ 
    using prob0-fst-qubits-index by simp
  finally show ?thesis by simp
qed

```

```

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

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

```

7.1 Measurements with Bell States

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

```

lemma prob0-bell-fst [simp]:
  assumes v =  $|\beta_{00}\rangle$   $\vee$  v =  $|\beta_{01}\rangle$   $\vee$  v =  $|\beta_{10}\rangle$   $\vee$  v =  $|\beta_{11}\rangle$ 
  shows prob0 2 v 0 = 1/2
proof -
  have set-0 [simp]:{k| k::nat. (k<4)  $\wedge$   $\neg$  select-index 2 0 k} = {0,1}
  using select-index-def by auto
  have v =  $|\beta_{00}\rangle \implies$  prob0 2 v 0 = 1/2
  proof -
    fix v assume asm:v =  $|\beta_{00}\rangle$ 
    show prob0 2 v 0 = 1/2
    proof -
      have prob0 2 v 0 = ( $\sum_{j \in \{k| k::nat. (k<4) \wedge \neg select-index 2 0 k\}}$ . (cmod(bell00 $$ (j,0)))2)
      by (auto simp: prob0-def asm)
      also have ... = ( $\sum_{j \in \{0,1\}}$ . (cmod(bell00 $$ (j,0)))2)
      using set-0 by simp
      also have ... = (cmod(1/sqrt(2)))2 + (cmod(0))2
      by (auto simp: bell00-def ket-vec-def)
      finally show ?thesis by(simp add: cmod-def power-divide)
    qed
  qed
  moreover have v =  $|\beta_{01}\rangle \implies$  prob0 2 v 0 = 1/2
  proof -
    fix v assume asm:v =  $|\beta_{01}\rangle$ 
    show prob0 2 v 0 = 1/2
    proof -
      have prob0 2 v 0 = ( $\sum_{j \in \{k| k::nat. (k<4) \wedge \neg select-index 2 0 k\}}$ . (cmod(bell01 $$ (j,0)))2)

```

```

    by (auto simp: prob0-def asm)
  also have ... = ( $\sum_{j \in \{0,1\}} (cmod(bell01 \$\$ (j,0)))^2$ )
    using set-0 by simp
  also have ... = ( $cmod(0))^2 + (cmod(1/\sqrt{2}))^2$ 
    by (auto simp: bell01-def ket-vec-def)
  finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
moreover have  $v = |\beta_{10}\rangle \implies prob0 2 v 0 = 1/2$ 
proof -
fix  $v$  assume  $asm:v = |\beta_{10}\rangle$ 
show  $prob0 2 v 0 = 1/2$ 
proof -
have  $prob0 2 v 0 = (\sum_{j \in \{k \mid k::nat. (k < 4) \wedge \neg select-index 2 0 k\}} (cmod(bell10 \$\$ (j,0)))^2)$ 
  by (auto simp: prob0-def asm)
also have ... = ( $\sum_{j \in \{0,1\}} (cmod(bell10 \$\$ (j,0)))^2$ )
  using set-0 by simp
also have ... = ( $cmod(1/\sqrt{2}))^2 + (cmod(0))^2$ 
  by (auto simp: bell10-def ket-vec-def)
finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
moreover have  $v = |\beta_{11}\rangle \implies prob0 2 v 0 = 1/2$ 
proof -
fix  $v$  assume  $asm:v = |\beta_{11}\rangle$ 
show  $prob0 2 v 0 = 1/2$ 
proof -
have  $prob0 2 v 0 = (\sum_{j \in \{k \mid k::nat. (k < 4) \wedge \neg select-index 2 0 k\}} (cmod(bell11 \$\$ (j,0)))^2)$ 
  by (auto simp: prob0-def asm)
also have ... = ( $\sum_{j \in \{0,1\}} (cmod(bell11 \$\$ (j,0)))^2$ )
  using set-0 by simp
also have ... = ( $cmod(0))^2 + (cmod(1/\sqrt{2}))^2$ 
  by (auto simp: bell11-def ket-vec-def)
finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
ultimately show ?thesis using assms by auto
qed

lemma prob-1-bell-fst [simp]:
assumes  $v = |\beta_{00}\rangle \vee v = |\beta_{01}\rangle \vee v = |\beta_{10}\rangle \vee v = |\beta_{11}\rangle$ 
shows  $prob1 2 v 0 = 1/2$ 
proof -
have set-0 [simp]: $\{k \mid k::nat. select-index 2 0 k\} = \{2,3\}$ 
  using select-index-def by auto
have  $v = |\beta_{00}\rangle \implies prob1 2 v 0 = 1/2$ 
proof -

```

```

fix v assume asm:v =  $|\beta_{00}\rangle$ 
show prob1 2 v 0 = 1/2
proof -
  have prob1 2 v 0 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 0 \ k\}} (cmod(bell00 \$\$ (j,0)))^2$ )
    by (auto simp: prob1-def asm)
  also have ... = ( $\sum_{j \in \{2,3\}} (cmod(bell00 \$\$ (j,0)))^2$ )
    using set-0 by simp
  also have ... = ( $cmod(1/\sqrt{2}))^2 + (cmod(0))^2$ 
    by (auto simp: bell00-def ket-vec-def)
  finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
moreover have v =  $|\beta_{01}\rangle \implies \text{prob1 } 2 \ v \ 0 = 1/2$ 
proof -
  fix v assume asm:v =  $|\beta_{01}\rangle$ 
  show prob1 2 v 0 = 1/2
  proof -
    have prob1 2 v 0 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 0 \ k\}} (cmod(bell01 \$\$ (j,0)))^2$ )
      by (auto simp: prob1-def asm)
    also have ... = ( $\sum_{j \in \{2,3\}} (cmod(bell01 \$\$ (j,0)))^2$ )
      using set-0 by simp
    also have ... = ( $cmod(0))^2 + (cmod(1/\sqrt{2}))^2$ 
      by (auto simp: bell01-def ket-vec-def)
    finally show ?thesis by(simp add: cmod-def power-divide)
  qed
qed
moreover have v =  $|\beta_{10}\rangle \implies \text{prob1 } 2 \ v \ 0 = 1/2$ 
proof -
  fix v assume asm:v =  $|\beta_{10}\rangle$ 
  show prob1 2 v 0 = 1/2
  proof -
    have prob1 2 v 0 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 0 \ k\}} (cmod(bell10 \$\$ (j,0)))^2$ )
      by (auto simp: prob1-def asm)
    also have ... = ( $\sum_{j \in \{2,3\}} (cmod(bell10 \$\$ (j,0)))^2$ )
      using set-0 by simp
    also have ... = ( $cmod(1/\sqrt{2}))^2 + (cmod(0))^2$ 
      by (auto simp: bell10-def ket-vec-def)
    finally show ?thesis by(simp add: cmod-def power-divide)
  qed
qed
moreover have v =  $|\beta_{11}\rangle \implies \text{prob1 } 2 \ v \ 0 = 1/2$ 
proof -
  fix v assume asm:v =  $|\beta_{11}\rangle$ 
  show prob1 2 v 0 = 1/2
  proof -
    have prob1 2 v 0 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 0 \ k\}} (cmod(bell11 \$\$ (j,0)))^2$ )
      by (auto simp: prob1-def asm)
  
```

```

(j,0)))^2)
  by (auto simp: prob1-def asm)
  also have ... = (∑ j∈{2,3}. (cmod(bell11 $$ (j,0)))^2)
    using set-0 by simp
  also have ... = (cmod(0))^2 + (cmod(1/sqrt(2)))^2
    by (auto simp: bell11-def ket-vec-def)
  finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
ultimately show ?thesis using assms by auto
qed

lemma prob0-bell-snd [simp]:
assumes v = |β00⟩ ∨ v = |β01⟩ ∨ v = |β10⟩ ∨ v = |β11⟩
shows prob0 2 v 1 = 1/2
proof -
  have set-0 [simp]: {k| k::nat. (k<4) ∧ ¬ select-index 2 1 k} = {0,2}
    by (auto simp: select-index-def)
      (metis Suc-le-mono add-Suc add-Suc-right le-numeral-extra(3) less-antisym
      mod-Suc-eq mod-less
      neq0-conv not-mod2-eq-Suc-0-eq-0 numeral-2-eq-2 numeral-Bit0 one-add-one
      one-mod-two-eq-one one-neq-zero)
  have v = |β00⟩ ⟹ prob0 2 v 1 = 1/2
  proof -
    fix v assume asm:v = |β00⟩
    show prob0 2 v 1 = 1/2
    proof -
      have prob0 2 v 1 = (∑ j∈{k| k::nat. (k<4) ∧ ¬ select-index 2 1 k}.
      (cmod(bell00 $$ (j,0)))^2)
        by (auto simp: prob0-def asm)
      also have ... = (∑ j∈{0,2}. (cmod(bell00 $$ (j,0)))^2)
        using set-0 by simp
      also have ... = (cmod(1/sqrt(2)))^2 + (cmod(0))^2
        by (auto simp: bell00-def ket-vec-def)
      finally show ?thesis by(simp add: cmod-def power-divide)
    qed
  qed
  moreover have v = |β01⟩ ⟹ prob0 2 v 1 = 1/2
  proof -
    fix v assume asm:v = |β01⟩
    show prob0 2 v 1 = 1/2
    proof -
      have prob0 2 v 1 = (∑ j∈{k| k::nat. (k<4) ∧ ¬ select-index 2 1 k}.
      (cmod(bell01 $$ (j,0)))^2)
        by (auto simp: prob0-def asm)
      also have ... = (∑ j∈{0,2}. (cmod(bell01 $$ (j,0)))^2)
        using set-0 by simp
      also have ... = (cmod(0))^2 + (cmod(1/sqrt(2)))^2
        by (auto simp: bell01-def ket-vec-def)
    qed
  qed

```

```

  finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
moreover have  $v = |\beta_{10}\rangle \implies \text{prob0 } 2 v 1 = 1/2$ 
proof -
fix  $v$  assume  $\text{asm}:v = |\beta_{10}\rangle$ 
show  $\text{prob0 } 2 v 1 = 1/2$ 
proof -
have  $\text{prob0 } 2 v 1 = (\sum_{j \in \{k \mid k::\text{nat}. (k < 4) \wedge \neg \text{select-index } 2 1 k\}} (cmod(\text{bell10 } \$\$ (j, 0)))^2)$ 
by (auto simp: prob0-def asm)
also have ... =  $(\sum_{j \in \{0, 2\}} (cmod(\text{bell10 } \$\$ (j, 0)))^2)$ 
using set-0 by simp
also have ... =  $(cmod(1/\sqrt{2}))^2 + (cmod(0))^2$ 
by (auto simp: bell10-def ket-vec-def)
finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
moreover have  $v = |\beta_{11}\rangle \implies \text{prob0 } 2 v 1 = 1/2$ 
proof -
fix  $v$  assume  $\text{asm}:v = |\beta_{11}\rangle$ 
show  $\text{prob0 } 2 v 1 = 1/2$ 
proof -
have  $\text{prob0 } 2 v 1 = (\sum_{j \in \{k \mid k::\text{nat}. (k < 4) \wedge \neg \text{select-index } 2 1 k\}} (cmod(\text{bell11 } \$\$ (j, 0)))^2)$ 
by (auto simp: prob0-def asm)
also have ... =  $(\sum_{j \in \{0, 2\}} (cmod(\text{bell11 } \$\$ (j, 0)))^2)$ 
using set-0 by simp
also have ... =  $(cmod(0))^2 + (cmod(1/\sqrt{2}))^2$ 
by (auto simp: bell11-def ket-vec-def)
finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
ultimately show ?thesis using assms by auto
qed

lemma prob-1-bell-snd [simp]:
assumes  $v = |\beta_{00}\rangle \vee v = |\beta_{01}\rangle \vee v = |\beta_{10}\rangle \vee v = |\beta_{11}\rangle$ 
shows  $\text{prob1 } 2 v 1 = 1/2$ 
proof -
have set-0:{ $k \mid k::\text{nat}. \text{select-index } 2 1 k\} = \{1, 3\}$ 
by (auto simp: select-index-def)
(metis Suc-le-lessD le-SucE le-less mod2-gr-0 mod-less mod-self numeral-2-eq-2
numeral-3-eq-3)
have  $v = |\beta_{00}\rangle \implies \text{prob1 } 2 v 1 = 1/2$ 
proof -
fix  $v$  assume  $\text{asm}:v = |\beta_{00}\rangle$ 
show  $\text{prob1 } 2 v 1 = 1/2$ 
proof -

```

```

have prob1 2 v 1 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 1 \ k\}} (cmod(bell00 \$\$ (j,0)))^2$ )
  by (auto simp: prob1-def asm)
also have ... = ( $\sum_{j \in \{1,3\}} (cmod(bell00 \$\$ (j,0)))^2$ )
  using set-0 by simp
also have ... = ( $cmod(1/\sqrt{2}))^2 + (cmod(0))^2$ 
  by (auto simp: bell00-def ket-vec-def)
finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
moreover have v =  $|\beta_{01}\rangle \implies \text{prob1 } 2 \ v \ 1 = 1/2$ 
proof -
  fix v assume asm:v =  $|\beta_{01}\rangle$ 
  show prob1 2 v 1 = 1/2
  proof -
    have prob1 2 v 1 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 1 \ k\}} (cmod(bell01 \$\$ (j,0)))^2$ )
      by (auto simp: prob1-def asm)
    also have ... = ( $\sum_{j \in \{1,3\}} (cmod(bell01 \$\$ (j,0)))^2$ )
      using set-0 by simp
    also have ... = ( $cmod(0))^2 + (cmod(1/\sqrt{2}))^2$ 
      by (auto simp: bell01-def ket-vec-def)
    finally show ?thesis by(simp add: cmod-def power-divide)
  qed
  qed
moreover have v =  $|\beta_{10}\rangle \implies \text{prob1 } 2 \ v \ 1 = 1/2$ 
proof -
  fix v assume asm:v =  $|\beta_{10}\rangle$ 
  show prob1 2 v 1 = 1/2
  proof -
    have prob1 2 v 1 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 1 \ k\}} (cmod(bell10 \$\$ (j,0)))^2$ )
      by (auto simp: prob1-def asm)
    also have ... = ( $\sum_{j \in \{1,3\}} (cmod(bell10 \$\$ (j,0)))^2$ )
      using set-0 by simp
    also have ... = ( $cmod(1/\sqrt{2}))^2 + (cmod(0))^2$ 
      by (auto simp: bell10-def ket-vec-def)
    finally show ?thesis by(simp add: cmod-def power-divide)
  qed
  qed
moreover have v =  $|\beta_{11}\rangle \implies \text{prob1 } 2 \ v \ 1 = 1/2$ 
proof -
  fix v assume asm:v =  $|\beta_{11}\rangle$ 
  show prob1 2 v 1 = 1/2
  proof -
    have prob1 2 v 1 = ( $\sum_{j \in \{k \mid k:\text{nat. select-index } 2 \ 1 \ k\}} (cmod(bell11 \$\$ (j,0)))^2$ )
      by (auto simp: prob1-def asm)
    also have ... = ( $\sum_{j \in \{1,3\}} (cmod(bell11 \$\$ (j,0)))^2$ )

```

```

using set-0 by simp
also have ... = (cmod(0))^2 + (cmod(1/sqrt(2)))^2
  by (auto simp: bell11-def ket-vec-def)
  finally show ?thesis by(simp add: cmod-def power-divide)
qed
qed
ultimately show ?thesis using assms by auto
qed

lemma post-meas0-bell00-fst [simp]:
  post-meas0 2 |β₀₀⟩ 0 = |unit-vec 4 0⟩
proof
  fix i j::nat assume i < dim-row |unit-vec 4 0⟩ and j < dim-col |unit-vec 4 0⟩
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas0 2 |β₀₀⟩ 0 $$ (0,0) = |unit-vec 4 0⟩ $$ (0,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def ket-vec-def real-sqrt-divide)
  moreover have post-meas0 2 |β₀₀⟩ 0 $$ (1,0) = |unit-vec 4 0⟩ $$ (1,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell00-def ket-vec-def real-sqrt-divide)
  moreover have post-meas0 2 |β₀₀⟩ 0 $$ (2,0) = |unit-vec 4 0⟩ $$ (2,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell00-def ket-vec-def real-sqrt-divide)
  moreover have post-meas0 2 |β₀₀⟩ 0 $$ (3,0) = |unit-vec 4 0⟩ $$ (3,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell00-def ket-vec-def real-sqrt-divide)
  ultimately show post-meas0 2 |β₀₀⟩ 0 $$ (i,j) = |unit-vec 4 0⟩ $$ (i,j) by auto
next
  show dim-row (post-meas0 2 |β₀₀⟩ 0) = dim-row |unit-vec 4 0⟩
    by(auto simp add: post-meas0-def ket-vec-def)
  show dim-col (post-meas0 2 |β₀₀⟩ 0) = dim-col |unit-vec 4 0⟩
    by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas0-bell00-snd [simp]:
  post-meas0 2 |β₀₀⟩ 1 = |unit-vec 4 0⟩
proof
  fix i j::nat assume i < dim-row |unit-vec 4 0⟩ and j < dim-col |unit-vec 4 0⟩
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas0 2 |β₀₀⟩ 1 $$ (0,0) = |unit-vec 4 0⟩ $$ (0,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell00-def ket-vec-def real-sqrt-divide del:One-nat-def)
  moreover have post-meas0 2 |β₀₀⟩ 1 $$ (1,0) = |unit-vec 4 0⟩ $$ (1,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell00-def ket-vec-def real-sqrt-divide)
  moreover have post-meas0 2 |β₀₀⟩ 1 $$ (2,0) = |unit-vec 4 0⟩ $$ (2,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell00-def ket-vec-def real-sqrt-divide)

```

```

moreover have post-meas0 2  $|\beta_{00}\rangle_1$  $$ (3,0) = |unit-vec 4 0\rangle $$ (3,0)
  by(simp add: post-meas0-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas0 2  $|\beta_{00}\rangle_1$  $$ (i,j) = |unit-vec 4 0\rangle $$ (i,j) by auto
next
  show dim-row (post-meas0 2  $|\beta_{00}\rangle_1$ ) = dim-row |unit-vec 4 0\rangle
    by(auto simp add: post-meas0-def ket-vec-def)
  show dim-col (post-meas0 2  $|\beta_{00}\rangle_1$ ) = dim-col |unit-vec 4 0\rangle
    by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas0-bell01-fst [simp]:
  post-meas0 2  $|\beta_{01}\rangle_0$  = |unit-vec 4 1\rangle
proof
  fix i j::nat assume i < dim-row |unit-vec 4 1\rangle and j < dim-col |unit-vec 4 1\rangle
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas0 2  $|\beta_{01}\rangle_0$  $$ (0,0) = |unit-vec 4 1\rangle $$ (0,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{01}\rangle_0$  $$ (1,0) = |unit-vec 4 1\rangle $$ (1,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{01}\rangle_0$  $$ (2,0) = |unit-vec 4 1\rangle $$ (2,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{01}\rangle_0$  $$ (3,0) = |unit-vec 4 1\rangle $$ (3,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas0 2  $|\beta_{01}\rangle_0$  $$ (i,j) = |unit-vec 4 1\rangle $$ (i,j) by auto
next
  show dim-row (post-meas0 2  $|\beta_{01}\rangle_0$ ) = dim-row |unit-vec 4 1\rangle
    by(auto simp add: post-meas0-def ket-vec-def)
  show dim-col (post-meas0 2  $|\beta_{01}\rangle_0$ ) = dim-col |unit-vec 4 1\rangle
    by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas0-bell01-snd [simp]:
  post-meas0 2  $|\beta_{01}\rangle_1$  = |unit-vec 4 2\rangle
proof
  fix i j::nat assume i < dim-row |unit-vec 4 2\rangle and j < dim-col |unit-vec 4 2\rangle
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas0 2  $|\beta_{01}\rangle_1$  $$ (0,0) = |unit-vec 4 2\rangle $$ (0,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{01}\rangle_1$  $$ (1,0) = |unit-vec 4 2\rangle $$ (1,0)
    by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)

```

```

moreover have post-meas0 2  $|\beta_{01}\rangle_1 \quad \$\$ (2,0) = |unit\text{-}vec\ 4\ 2\rangle\ \$\$ (2,0)$ 
  by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide del:One-nat-def)
moreover have post-meas0 2  $|\beta_{01}\rangle_1 \quad \$\$ (3,0) = |unit\text{-}vec\ 4\ 2\rangle\ \$\$ (3,0)$ 
  by(simp add: post-meas0-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
ultimately show post-meas0 2  $|\beta_{01}\rangle_1 \quad \$\$ (i,j) = |unit\text{-}vec\ 4\ 2\rangle\ \$\$ (i,j)$  by auto
next
  show dim-row (post-meas0 2  $|\beta_{01}\rangle_1) = dim\text{-}row |unit\text{-}vec\ 4\ 2\rangle$ 
    by(auto simp add: post-meas0-def ket-vec-def)
  show dim-col (post-meas0 2  $|\beta_{01}\rangle_1) = dim\text{-}col |unit\text{-}vec\ 4\ 2\rangle$ 
    by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas0-bell10-fst [simp]:
  post-meas0 2  $|\beta_{10}\rangle_0 = |unit\text{-}vec\ 4\ 0\rangle$ 
proof
  fix i j::nat assume i < dim-row |unit-vec 4 0⟩ and j < dim-col |unit-vec 4 0⟩
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas0 2  $|\beta_{10}\rangle_0 \quad \$\$ (0,0) = |unit\text{-}vec\ 4\ 0\rangle\ \$\$ (0,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{10}\rangle_0 \quad \$\$ (1,0) = |unit\text{-}vec\ 4\ 0\rangle\ \$\$ (1,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{10}\rangle_0 \quad \$\$ (2,0) = |unit\text{-}vec\ 4\ 0\rangle\ \$\$ (2,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{10}\rangle_0 \quad \$\$ (3,0) = |unit\text{-}vec\ 4\ 0\rangle\ \$\$ (3,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas0 2  $|\beta_{10}\rangle_0 \quad \$\$ (i,j) = |unit\text{-}vec\ 4\ 0\rangle\ \$\$ (i,j)$  by auto
next
  show dim-row (post-meas0 2  $|\beta_{10}\rangle_0) = dim\text{-}row |unit\text{-}vec\ 4\ 0\rangle$ 
    by(auto simp add: post-meas0-def ket-vec-def)
  show dim-col (post-meas0 2  $|\beta_{10}\rangle_0) = dim\text{-}col |unit\text{-}vec\ 4\ 0\rangle$ 
    by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas0-bell10-snd [simp]:
  post-meas0 2  $|\beta_{10}\rangle_1 = |unit\text{-}vec\ 4\ 0\rangle$ 
proof
  fix i j::nat assume i < dim-row |unit-vec 4 0⟩ and j < dim-col |unit-vec 4 0⟩
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas0 2  $|\beta_{10}\rangle_1 \quad \$\$ (0,0) = |unit\text{-}vec\ 4\ 0\rangle\ \$\$ (0,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide del:One-nat-def)

```

```

moreover have post-meas0 2  $|\beta_{10}\rangle_1$   $\$ \$ (1,0) = |unit\text{-}vec 4 0\rangle \$ \$ (1,0)$ 
  by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
moreover have post-meas0 2  $|\beta_{10}\rangle_1$   $\$ \$ (2,0) = |unit\text{-}vec 4 0\rangle \$ \$ (2,0)$ 
  by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
moreover have post-meas0 2  $|\beta_{10}\rangle_1$   $\$ \$ (3,0) = |unit\text{-}vec 4 0\rangle \$ \$ (3,0)$ 
  by(simp add: post-meas0-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
ultimately show post-meas0 2  $|\beta_{10}\rangle_1$   $\$ \$ (i,j) = |unit\text{-}vec 4 0\rangle \$ \$ (i,j)$  by auto
next
show dim-row (post-meas0 2  $|\beta_{10}\rangle_1$ ) = dim-row  $|unit\text{-}vec 4 0\rangle$ 
  by(auto simp add: post-meas0-def ket-vec-def)
show dim-col (post-meas0 2  $|\beta_{10}\rangle_1$ ) = dim-col  $|unit\text{-}vec 4 0\rangle$ 
  by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas0-bell11-fst [simp]:
  post-meas0 2  $|\beta_{11}\rangle_0 = |unit\text{-}vec 4 1\rangle$ 
proof
  fix i j::nat assume i < dim-row  $|unit\text{-}vec 4 1\rangle$  and j < dim-col  $|unit\text{-}vec 4 1\rangle$ 
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas0 2  $|\beta_{11}\rangle_0$   $\$ \$ (0,0) = |unit\text{-}vec 4 1\rangle \$ \$ (0,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{11}\rangle_0$   $\$ \$ (1,0) = |unit\text{-}vec 4 1\rangle \$ \$ (1,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{11}\rangle_0$   $\$ \$ (2,0) = |unit\text{-}vec 4 1\rangle \$ \$ (2,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas0 2  $|\beta_{11}\rangle_0$   $\$ \$ (3,0) = |unit\text{-}vec 4 1\rangle \$ \$ (3,0)$ 
    by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas0 2  $|\beta_{11}\rangle_0$   $\$ \$ (i,j) = |unit\text{-}vec 4 1\rangle \$ \$ (i,j)$  by auto
next
show dim-row (post-meas0 2  $|\beta_{11}\rangle_0$ ) = dim-row  $|unit\text{-}vec 4 1\rangle$ 
  by(auto simp add: post-meas0-def ket-vec-def)
show dim-col (post-meas0 2  $|\beta_{11}\rangle_0$ ) = dim-col  $|unit\text{-}vec 4 1\rangle$ 
  by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas0-bell11-snd [simp]:
  post-meas0 2  $|\beta_{11}\rangle_1 = - |unit\text{-}vec 4 2\rangle$ 
proof
  fix i j::nat assume i < dim-row ( $- |unit\text{-}vec 4 2\rangle$ ) and j < dim-col ( $- |unit\text{-}vec 4 2\rangle$ )
  then have i ∈ {0,1,2,3} and j = 0

```

```

by(auto simp add: ket-vec-def)
moreover have post-meas0 2  $|\beta_{11}\rangle_1$  $$ (0,0) = (- |unit-vec 4 2\rangle) $$ (0,0)
  by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
moreover have post-meas0 2  $|\beta_{11}\rangle_1$  $$ (1,0) = (- |unit-vec 4 2\rangle) $$ (1,0)
  by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
moreover have post-meas0 2  $|\beta_{11}\rangle_1$  $$ (2,0) = (- |unit-vec 4 2\rangle) $$ (2,0)
  by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide del:One-nat-def)
moreover have post-meas0 2  $|\beta_{11}\rangle_1$  $$ (3,0) = (- |unit-vec 4 2\rangle) $$ (3,0)
  by(simp add: post-meas0-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
ultimately show post-meas0 2  $|\beta_{11}\rangle_1$  $$ (i,j) = (- |unit-vec 4 2\rangle) $$ (i,j) by
auto
next
  show dim-row (post-meas0 2  $|\beta_{11}\rangle_1$ ) = dim-row (- |unit-vec 4 2\rangle)
    by(auto simp add: post-meas0-def ket-vec-def)
  show dim-col (post-meas0 2  $|\beta_{11}\rangle_1$ ) = dim-col (- |unit-vec 4 2\rangle)
    by(auto simp add: post-meas0-def ket-vec-def)
qed

lemma post-meas1-bell00-fst [simp]:
  post-meas1 2  $|\beta_{00}\rangle_0$  = |unit-vec 4 3\rangle
proof
  fix i j::nat assume i < dim-row |unit-vec 4 3\rangle and j < dim-col |unit-vec 4 3\rangle
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2  $|\beta_{00}\rangle_0$  $$ (0,0) = |unit-vec 4 3\rangle $$ (0,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{00}\rangle_0$  $$ (1,0) = |unit-vec 4 3\rangle $$ (1,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{00}\rangle_0$  $$ (2,0) = |unit-vec 4 3\rangle $$ (2,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{00}\rangle_0$  $$ (3,0) = |unit-vec 4 3\rangle $$ (3,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas1 2  $|\beta_{00}\rangle_0$  $$ (i,j) = |unit-vec 4 3\rangle $$ (i,j) by auto
next
  show dim-row (post-meas1 2  $|\beta_{00}\rangle_0$ ) = dim-row |unit-vec 4 3\rangle
    by(auto simp add: post-meas1-def ket-vec-def)
  show dim-col (post-meas1 2  $|\beta_{00}\rangle_0$ ) = dim-col |unit-vec 4 3\rangle
    by(auto simp add: post-meas1-def ket-vec-def)
qed

lemma post-meas1-bell00-snd [simp]:

```

```

post-meas1 2  $|\beta_{00}\rangle$  1 =  $|unit\text{-}vec\ 4\ 3\rangle$ 
proof
  fix i j::nat assume i < dim-row  $|unit\text{-}vec\ 4\ 3\rangle$  and j < dim-col  $|unit\text{-}vec\ 4\ 3\rangle$ 
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2  $|\beta_{00}\rangle$  1 $$ (0,0) =  $|unit\text{-}vec\ 4\ 3\rangle$  $$ (0,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{00}\rangle$  1 $$ (1,0) =  $|unit\text{-}vec\ 4\ 3\rangle$  $$ (1,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{00}\rangle$  1 $$ (2,0) =  $|unit\text{-}vec\ 4\ 3\rangle$  $$ (2,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{00}\rangle$  1 $$ (3,0) =  $|unit\text{-}vec\ 4\ 3\rangle$  $$ (3,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell00-def ket-vec-def
real-sqrt-divide del: One-nat-def)
  ultimately show post-meas1 2  $|\beta_{00}\rangle$  1 $$ (i,j) =  $|unit\text{-}vec\ 4\ 3\rangle$  $$ (i,j) by auto
next
  show dim-row (post-meas1 2  $|\beta_{00}\rangle$  1) = dim-row  $|unit\text{-}vec\ 4\ 3\rangle$ 
    by(auto simp add: post-meas1-def ket-vec-def)
  show dim-col (post-meas1 2  $|\beta_{00}\rangle$  1) = dim-col  $|unit\text{-}vec\ 4\ 3\rangle$ 
    by(auto simp add: post-meas1-def ket-vec-def)
qed

lemma post-meas1-bell01-fst [simp]:
  post-meas1 2  $|\beta_{01}\rangle$  0 =  $|unit\text{-}vec\ 4\ 2\rangle$ 
proof
  fix i j::nat assume i < dim-row  $|unit\text{-}vec\ 4\ 2\rangle$  and j < dim-col  $|unit\text{-}vec\ 4\ 2\rangle$ 
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2  $|\beta_{01}\rangle$  0 $$ (0,0) =  $|unit\text{-}vec\ 4\ 2\rangle$  $$ (0,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{01}\rangle$  0 $$ (1,0) =  $|unit\text{-}vec\ 4\ 2\rangle$  $$ (1,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{01}\rangle$  0 $$ (2,0) =  $|unit\text{-}vec\ 4\ 2\rangle$  $$ (2,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{01}\rangle$  0 $$ (3,0) =  $|unit\text{-}vec\ 4\ 2\rangle$  $$ (3,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas1 2  $|\beta_{01}\rangle$  0 $$ (i,j) =  $|unit\text{-}vec\ 4\ 2\rangle$  $$ (i,j) by auto
next
  show dim-row (post-meas1 2  $|\beta_{01}\rangle$  0) = dim-row  $|unit\text{-}vec\ 4\ 2\rangle$ 
    by(auto simp add: post-meas1-def ket-vec-def)
  show dim-col (post-meas1 2  $|\beta_{01}\rangle$  0) = dim-col  $|unit\text{-}vec\ 4\ 2\rangle$ 
    by(auto simp add: post-meas1-def ket-vec-def)

```

qed

```
lemma post-meas1-bell01-snd [simp]:
  post-meas1 2 |β01⟩ 1 = |unit-vec 4 1⟩
proof
  fix i j::nat assume i < dim-row |unit-vec 4 1⟩ and j < dim-col |unit-vec 4 1⟩
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2 |β01⟩ 1 $$ (0,0) = |unit-vec 4 1⟩ $$ (0,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2 |β01⟩ 1 $$ (1,0) = |unit-vec 4 1⟩ $$ (1,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide del: One-nat-def)
  moreover have post-meas1 2 |β01⟩ 1 $$ (2,0) = |unit-vec 4 1⟩ $$ (2,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2 |β01⟩ 1 $$ (3,0) = |unit-vec 4 1⟩ $$ (3,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell01-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas1 2 |β01⟩ 1 $$ (i,j) = |unit-vec 4 1⟩ $$ (i,j) by auto
next
  show dim-row (post-meas1 2 |β01⟩ 1) = dim-row |unit-vec 4 1⟩
    by(auto simp add: post-meas1-def ket-vec-def)
  show dim-col (post-meas1 2 |β01⟩ 1) = dim-col |unit-vec 4 1⟩
    by(auto simp add: post-meas1-def ket-vec-def)
qed
```

```
lemma post-meas1-bell10-fst [simp]:
  post-meas1 2 |β10⟩ 0 = - |unit-vec 4 3⟩
proof
  fix i j::nat assume i < dim-row (- |unit-vec 4 3⟩) and j < dim-col (- |unit-vec
4 3⟩)
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2 |β10⟩ 0 $$ (0,0) = (- |unit-vec 4 3⟩) $$ (0,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2 |β10⟩ 0 $$ (1,0) = (- |unit-vec 4 3⟩) $$ (1,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2 |β10⟩ 0 $$ (2,0) = (- |unit-vec 4 3⟩) $$ (2,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2 |β10⟩ 0 $$ (3,0) = (- |unit-vec 4 3⟩) $$ (3,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas1 2 |β10⟩ 0 $$ (i,j) = (- |unit-vec 4 3⟩) $$ (i,j) by auto
```

```

next
  show dim-row (post-meas1 2  $|\beta_{10}\rangle 0$ ) = dim-row ( $-|unit\text{-}vec 4 3\rangle$ )
    by(auto simp add: post-meas1-def ket-vec-def)
  show dim-col (post-meas1 2  $|\beta_{10}\rangle 0$ ) = dim-col ( $-|unit\text{-}vec 4 3\rangle$ )
    by(auto simp add: post-meas1-def ket-vec-def)
qed

lemma post-meas1-bell10-snd [simp]:
  post-meas1 2  $|\beta_{10}\rangle 1 = -|unit\text{-}vec 4 3\rangle$ 
proof
  fix i j::nat assume i < dim-row ( $-|unit\text{-}vec 4 3\rangle$ ) and j < dim-col ( $-|unit\text{-}vec 4 3\rangle$ )
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2  $|\beta_{10}\rangle 1 \$\$ (0,0) = (-|unit\text{-}vec 4 3\rangle) \$\$ (0,0)$ 
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{10}\rangle 1 \$\$ (1,0) = (-|unit\text{-}vec 4 3\rangle) \$\$ (1,0)$ 
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{10}\rangle 1 \$\$ (2,0) = (-|unit\text{-}vec 4 3\rangle) \$\$ (2,0)$ 
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{10}\rangle 1 \$\$ (3,0) = (-|unit\text{-}vec 4 3\rangle) \$\$ (3,0)$ 
    by(simp add: post-meas1-def unit-vec-def select-index-def bell10-def ket-vec-def
real-sqrt-divide del: One-nat-def)
  ultimately show post-meas1 2  $|\beta_{10}\rangle 1 \$\$ (i,j) = (-|unit\text{-}vec 4 3\rangle) \$\$ (i,j)$  by
auto
next
  show dim-row (post-meas1 2  $|\beta_{10}\rangle 1) = dim-row (-|unit\text{-}vec 4 3\rangle)$ 
    by(auto simp add: post-meas1-def ket-vec-def)
  show dim-col (post-meas1 2  $|\beta_{10}\rangle 1) = dim-col (-|unit\text{-}vec 4 3\rangle)$ 
    by(auto simp add: post-meas1-def ket-vec-def)
qed

lemma post-meas1-bell11-fst [simp]:
  post-meas1 2  $|\beta_{11}\rangle 0 = -|unit\text{-}vec 4 2\rangle$ 
proof
  fix i j::nat assume i < dim-row ( $-|unit\text{-}vec 4 2\rangle$ ) and j < dim-col ( $-|unit\text{-}vec 4 2\rangle$ )
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2  $|\beta_{11}\rangle 0 \$\$ (0,0) = (-|unit\text{-}vec 4 2\rangle) \$\$ (0,0)$ 
    by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{11}\rangle 0 \$\$ (1,0) = (-|unit\text{-}vec 4 2\rangle) \$\$ (1,0)$ 
    by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{11}\rangle 0 \$\$ (2,0) = (-|unit\text{-}vec 4 2\rangle) \$\$ (2,0)$ 

```

```

by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
moreover have post-meas1 2  $|\beta_{11}\rangle_0$  $$ (3,0) = (- |unit-vec 4 2\rangle) $$ (3,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
ultimately show post-meas1 2  $|\beta_{11}\rangle_0$  $$ (i,j) = (- |unit-vec 4 2\rangle) $$ (i,j) by
auto
next
    show dim-row (post-meas1 2  $|\beta_{11}\rangle_0$ ) = dim-row (- |unit-vec 4 2\rangle)
        by(auto simp add: post-meas1-def ket-vec-def)
    show dim-col (post-meas1 2  $|\beta_{11}\rangle_0$ ) = dim-col (- |unit-vec 4 2\rangle)
        by(auto simp add: post-meas1-def ket-vec-def)
qed

lemma post-meas1-bell11-snd [simp]:
  post-meas1 2  $|\beta_{11}\rangle_1$  = |unit-vec 4 1\rangle
proof
  fix i j::nat assume i < dim-row |unit-vec 4 1\rangle and j < dim-col |unit-vec 4 1\rangle
  then have i ∈ {0,1,2,3} and j = 0
    by(auto simp add: ket-vec-def)
  moreover have post-meas1 2  $|\beta_{11}\rangle_1$  $$ (0,0) = |unit-vec 4 1\rangle $$ (0,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{11}\rangle_1$  $$ (1,0) = |unit-vec 4 1\rangle $$ (1,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide del: One-nat-def)
  moreover have post-meas1 2  $|\beta_{11}\rangle_1$  $$ (2,0) = |unit-vec 4 1\rangle $$ (2,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  moreover have post-meas1 2  $|\beta_{11}\rangle_1$  $$ (3,0) = |unit-vec 4 1\rangle $$ (3,0)
    by(simp add: post-meas1-def unit-vec-def select-index-def bell11-def ket-vec-def
real-sqrt-divide)
  ultimately show post-meas1 2  $|\beta_{11}\rangle_1$  $$ (i,j) = |unit-vec 4 1\rangle $$ (i,j) by auto
next
    show dim-row (post-meas1 2  $|\beta_{11}\rangle_1$ ) = dim-row |unit-vec 4 1\rangle
        by(auto simp add: post-meas1-def ket-vec-def)
    show dim-col (post-meas1 2  $|\beta_{11}\rangle_1$ ) = dim-col |unit-vec 4 1\rangle
        by(auto simp add: post-meas1-def ket-vec-def)
qed

end

```

8 Quantum Entanglement

```

theory Entanglement
imports
  Quantum
  More-Tensor
begin

```

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

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

```
definition prod-state2:: complex Matrix.mat  $\Rightarrow$  bool where
prod-state2  $u \equiv$  if state 2  $u$  then  $\exists v w. \text{state 1 } v \wedge \text{state 1 } w \wedge u = v \otimes w$  else undefined
```

```
definition entangled2:: complex Matrix.mat  $\Rightarrow$  bool where
entangled2  $u \equiv \neg \text{prod-state2 } u$ 
```

The Bell states are entangled states.

```
lemma bell00-is-entangled2 [simp]:
entangled2  $|\beta_{00}\rangle$ 
proof –
have  $\forall v w. \text{state 1 } v \rightarrow \text{state 1 } w \rightarrow |\beta_{00}\rangle \neq v \otimes w$ 
proof((rule allI)+,(rule impI)+, rule notI)
fix  $v w$ 
assume  $a0:\text{state 1 } v \text{ and } a1:\text{state 1 } w \text{ and } a2:|\beta_{00}\rangle = v \otimes w$ 
have  $(v \$\$ (0,0) * w \$\$ (0,0)) * (v \$\$ (1,0) * w \$\$ (1,0)) =$ 
 $(v \$\$ (0,0) * w \$\$ (1,0)) * (v \$\$ (1,0) * w \$\$ (0,0)) \text{ by simp}$ 
then have  $(v \otimes w) \$\$ (0,0) * (v \otimes w) \$\$ (3,0) = (v \otimes w) \$\$ (1,0) * (v$ 
 $\otimes w) \$\$ (2,0)$ 
using  $a0 a1 \text{ by simp}$ 
then have  $|\beta_{00}\rangle \$\$ (0,0) * |\beta_{00}\rangle \$\$ (3,0) = |\beta_{00}\rangle \$\$ (1,0) * |\beta_{00}\rangle \$\$ (2,0)$ 
using  $a2 \text{ by simp}$ 
then have  $1 / \sqrt{2} * 1 / \sqrt{2} = 0 \text{ by simp}$ 
thus False by simp
qed
thus ?thesis by(simp add: entangled2-def prod-state2-def)
qed
```

```
lemma bell01-is-entangled2 [simp]:
entangled2  $|\beta_{01}\rangle$ 
proof –
have  $\forall v w. \text{state 1 } v \rightarrow \text{state 1 } w \rightarrow |\beta_{01}\rangle \neq v \otimes w$ 
proof((rule allI)+,(rule impI)+, rule notI)
fix  $v w$ 
assume  $a0:\text{state 1 } v \text{ and } a1:\text{state 1 } w \text{ and } a2:|\beta_{01}\rangle = v \otimes w$ 
have  $(v \$\$ (0,0) * w \$\$ (1,0)) * (v \$\$ (1,0) * w \$\$ (0,0)) =$ 
 $(v \$\$ (0,0) * w \$\$ (0,0)) * (v \$\$ (1,0) * w \$\$ (1,0)) \text{ by simp}$ 
then have  $(v \otimes w) \$\$ (1,0) * (v \otimes w) \$\$ (2,0) = (v \otimes w) \$\$ (0,0) * (v$ 
 $\otimes w) \$\$ (3,0)$ 
using  $a0 a1 \text{ by simp}$ 
then have  $|\beta_{01}\rangle \$\$ (1,0) * |\beta_{01}\rangle \$\$ (2,0) = |\beta_{01}\rangle \$\$ (0,0) * |\beta_{01}\rangle \$\$ (3,0)$ 
using  $a2 \text{ by simp}$ 
then have  $1 / \sqrt{2} * 1 / \sqrt{2} = 0$ 
```

```

using bell01-index by simp
thus False by simp
qed
thus ?thesis by(simp add: entangled2-def prod-state2-def)
qed

lemma bell10-is-entangled2 [simp]:
  entangled2  $|\beta_{10}\rangle$ 
proof -
  have  $\forall v w. \text{state } 1 v \rightarrow \text{state } 1 w \rightarrow |\beta_{10}\rangle \neq v \otimes w$ 
  proof((rule allI)+,(rule impI)+, rule notI)
    fix v w
    assume a0:state 1 v and a1:state 1 w and a2: $|\beta_{10}\rangle = v \otimes w$ 
    have  $(v \$\$ (0,0) * w \$\$ (0,0)) * (v \$\$ (1,0) * w \$\$ (1,0)) =$ 
          $(v \$\$ (0,0) * w \$\$ (1,0)) * (v \$\$ (1,0) * w \$\$ (0,0))$  by simp
    then have  $(v \otimes w) \$\$ (0,0) * (v \otimes w) \$\$ (3,0) = (v \otimes w) \$\$ (1,0) * (v$ 
               $\otimes w) \$\$ (2,0)$ 
      using a0 a1 by simp
    then have  $|\beta_{10}\rangle \$\$ (1,0) * |\beta_{10}\rangle \$\$ (2,0) = |\beta_{10}\rangle \$\$ (0,0) * |\beta_{10}\rangle \$\$ (3,0)$ 
      using a2 by simp
    then have  $1/\sqrt{2} * 1/\sqrt{2} = 0$  by simp
    thus False by simp
  qed
  thus ?thesis by(simp add: entangled2-def prod-state2-def)
qed

lemma bell11-is-entangled2 [simp]:
  entangled2  $|\beta_{11}\rangle$ 
proof -
  have  $\forall v w. \text{state } 1 v \rightarrow \text{state } 1 w \rightarrow |\beta_{11}\rangle \neq v \otimes w$ 
  proof((rule allI)+,(rule impI)+, rule notI)
    fix v w
    assume a0:state 1 v and a1:state 1 w and a2: $|\beta_{11}\rangle = v \otimes w$ 
    have  $(v \$\$ (0,0) * w \$\$ (1,0)) * (v \$\$ (1,0) * w \$\$ (0,0)) =$ 
          $(v \$\$ (0,0) * w \$\$ (0,0)) * (v \$\$ (1,0) * w \$\$ (1,0))$  by simp
    then have  $(v \otimes w) \$\$ (1,0) * (v \otimes w) \$\$ (2,0) = (v \otimes w) \$\$ (0,0) * (v$ 
               $\otimes w) \$\$ (3,0)$ 
      using a0 a1 by simp
    then have  $|\beta_{11}\rangle \$\$ (1,0) * |\beta_{11}\rangle \$\$ (2,0) = |\beta_{11}\rangle \$\$ (0,0) * |\beta_{11}\rangle \$\$ (3,0)$ 
      using a2 by simp
    then have  $1/\sqrt{2} * 1/\sqrt{2} = 0$ 
      using bell-11-index by simp
    thus False by simp
  qed
  thus ?thesis by(simp add: entangled2-def prod-state2-def)
qed

```

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

```

definition prod-state:: nat  $\Rightarrow$  complex Matrix.mat  $\Rightarrow$  bool where
prod-state  $m\ u \equiv$  if state  $m\ u$  then  $\exists n\ p::nat.\exists v\ w.$  state  $n\ v \wedge$  state  $p\ w \wedge$ 
 $n < m \wedge p < m \wedge u = v \otimes w$  else undefined

definition entangled:: nat  $\Rightarrow$  complex Matrix.mat  $\Rightarrow$  bool where
entangled  $n\ v \equiv \neg (\text{prod-state } n\ v)$ 

```

```

lemma sanity-check:
 $\neg(\text{entangled } 2\ (\text{mat-of-cols-list } 2\ [[1/\sqrt{2}], 1/\sqrt{2}]] \otimes \text{mat-of-cols-list } 2\ [[1/\sqrt{2}], 1/\sqrt{2}]))$ 
proof -
  define  $u$  where  $u = \text{mat-of-cols-list } 2\ [[1/\sqrt{2}], 1/\sqrt{2}]]$ 
  then have state 1  $u$ 
  proof -
    have dim-col  $u = 1$ 
    using  $u\text{-def mat-of-cols-list-def by simp}$ 
    moreover have  $f:\text{dim-row } u = 2$ 
    using  $u\text{-def mat-of-cols-list-def by simp}$ 
    moreover have  $\|\text{Matrix.col } u\ 0\| = 1$ 
    proof -
      have  $(\sum i < 2. (\text{cmod } (u \$\$ (i, 0)))^2) = (1/\sqrt{2})^2 + (1/\sqrt{2})^2$ 
      by(simp add:  $u\text{-def cmod-def numeral-2-eq-2}$ )
      then have  $\|\text{Matrix.col } u\ 0\| = \sqrt{(1/\sqrt{2})^2 + (1/\sqrt{2})^2}$ 
      using  $f$  by(auto simp: Matrix.col-def  $u\text{-def cpx-vec-length-def}$ )
      thus ?thesis by(simp add: power-divide)
    qed
    ultimately show ?thesis by(simp add: state-def)
  qed
  then have state 2  $(u \otimes u)$ 
  using tensor-state by(metis one-add-one)
  thus ?thesis
    using entangled-def prod-state-def by(metis `state 1 u` one-less-numeral-iff
semiring-norm(76)  $u\text{-def}$ )
  qed
end

```

9 Quantum Teleportation

```

theory Quantum-Teleportation
imports
  More-Tensor
  Basics
  Measurement
begin

```

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

```
abbreviation M1:: complex Matrix.mat where
M1  $\equiv$  mat-of-cols-list 8 [[1, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0]]
```

```
lemma tensor-prod-of-cnot-id-1:
shows ( $CNOT \otimes Id 1$ ) = M1
proof
show dim-col ( $CNOT \otimes Id 1$ ) = dim-col M1
by(simp add: CNOT-def Id-def mat-of-cols-list-def)
show dim-row ( $CNOT \otimes Id 1$ ) = dim-row M1
by(simp add: CNOT-def Id-def mat-of-cols-list-def)
fix i j::nat assume i < dim-row M1 and j < dim-col M1
then have i  $\in \{0..8\}$   $\wedge$  j  $\in \{0..8\}$ 
by (auto simp add: mat-of-cols-list-def)
then show ( $CNOT \otimes Id 1$ ) $(i, j) = M1 $(i, j)
by (auto simp add: Id-def CNOT-def mat-of-cols-list-def)
qed
```

```
abbreviation M2:: complex Matrix.mat where
M2  $\equiv$  mat-of-cols-list 8 [[1/sqrt(2), 0, 0, 0, 1/sqrt(2), 0, 0, 0],
[0, 1/sqrt(2), 0, 0, 0, 1/sqrt(2), 0, 0],
[0, 0, 1/sqrt(2), 0, 0, 0, 1/sqrt(2), 0],
[0, 0, 0, 1/sqrt(2), 0, 0, 0, 1/sqrt(2)],
[1/sqrt(2), 0, 0, 0, -1/sqrt(2), 0, 0, 0],
[0, 1/sqrt(2), 0, 0, 0, -1/sqrt(2), 0, 0],
[0, 0, 1/sqrt(2), 0, 0, 0, -1/sqrt(2), 0],
[0, 0, 0, 1/sqrt(2), 0, 0, 0, -1/sqrt(2)]]
```

```
lemma tensor-prod-of-h-id-2:
shows ( $H \otimes Id 2$ ) = M2
proof
show dim-col ( $H \otimes Id 2$ ) = dim-col M2
by(simp add: H-def Id-def mat-of-cols-list-def)
show dim-row ( $H \otimes Id 2$ ) = dim-row M2
by(simp add: H-def Id-def mat-of-cols-list-def)
fix i j::nat assume i < dim-row M2 and j < dim-col M2
then have i  $\in \{0..8\}$   $\wedge$  j  $\in \{0..8\}$ 
by (auto simp add: mat-of-cols-list-def)
then show ( $H \otimes Id 2$ ) $(i, j) = M2 $(i, j)
by (auto simp add: Id-def H-def mat-of-cols-list-def)
```

qed

lemma alice-step-1-state [simp]:

assumes state 1 φ

shows state 3 $(\varphi \otimes |\beta_{00}\rangle)$

using assms bell00-is-state tensor-state **by** (metis One-nat-def Suc-1 numeral-3-eq-3 plus-1-eq-Suc)

lemma alice-step-2-state:

assumes state 1 φ

shows state 3 $((CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle))$

proof-

have gate 3 $(CNOT \otimes Id 1)$

using CNOT-is-gate id-is-gate tensor-gate **by** (metis numeral-plus-one semiring-norm(5))

then show state 3 $((CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle))$ **using** assms **by** simp

qed

lemma alice-state [simp]:

assumes state 1 φ

shows state 3 $(alice \varphi)$

proof-

have gate 3 $(H \otimes Id 2)$

using tensor-gate id-is-gate H-is-gate **by** (metis eval-nat-numeral(3) plus-1-eq-Suc)

then show ?thesis

using assms alice-step-2-state **by** (simp add: alice-def)

qed

lemma alice-step-1:

assumes state 1 φ **and** $\alpha = \varphi \$\$ (0,0)$ **and** $\beta = \varphi \$\$ (1,0)$

shows $(\varphi \otimes |\beta_{00}\rangle) = mat-of-cols-list 8 [[\alpha/\sqrt{2}, 0, 0, \alpha/\sqrt{2}, \beta/\sqrt{2}, 0, 0, \beta/\sqrt{2}]]$

proof

define v **where** $asm:v = mat-of-cols-list 8 [[\alpha/\sqrt{2}, 0, 0, \alpha/\sqrt{2}, \beta/\sqrt{2}, 0, 0, \beta/\sqrt{2}]]$

then show dim-row $(\varphi \otimes |\beta_{00}\rangle) = dim-row v$

using assms(1) alice-step-1-state.state.dim-row mat-of-cols-list-def **by** fastforce

show dim-col $(\varphi \otimes |\beta_{00}\rangle) = dim-col v$

using assms(1) alice-step-1-state.state.is-column asm mat-of-cols-list-def **by**

fastforce

show $\bigwedge i. i < dim-row v \implies j < dim-col v \implies (\varphi \otimes |\beta_{00}\rangle) \$\$ (i,j) = v \$\$ (i,j)$

(i,j)

proof-

fix i j **assume** $i < dim-row v$ **and** $j < dim-col v$

then have $i \in \{0..<8\} \wedge j = 0$

using asm **by** (auto simp add: mat-of-cols-list-def)

moreover have dim-row $|\beta_{00}\rangle = 4$

using bell00-is-state state-def **by** simp

moreover have dim-col $|\beta_{00}\rangle = 1$

using bell00-is-state state-def **by** simp

ultimately show $(\varphi \otimes |\beta_{00}\rangle) \$\$ (i,j) = v \$\$ (i,j)$

```

    using state-def assms asm by (auto simp add: bell00-def)
qed
qed

lemma alice-step-2:
assumes state 1  $\varphi$  and  $\alpha = \varphi \$\$ (0,0)$  and  $\beta = \varphi \$\$ (1,0)$ 
shows  $(CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle) = mat\text{-}of\text{-}cols\text{-}list 8 [[\alpha/\sqrt{2}, 0, 0, \alpha/\sqrt{2}, 0, \beta/\sqrt{2}, \beta/\sqrt{2}, 0]]$ 
proof
have  $f0:(\varphi \otimes |\beta_{00}\rangle) = mat\text{-}of\text{-}cols\text{-}list 8 [[\alpha/\sqrt{2}, 0, 0, \alpha/\sqrt{2}, \beta/\sqrt{2}, 0, 0, \beta/\sqrt{2}]]$ 
using assms alice-step-1 by simp
define v where  $asm:v = mat\text{-}of\text{-}cols\text{-}list 8 [[\alpha/\sqrt{2}, 0, 0, \alpha/\sqrt{2}, 0, \beta/\sqrt{2}, \beta/\sqrt{2}, 0]]$ 
then show dim-row  $((CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle)) = dim\text{-}row v$ 
using assms(1) alice-step-2-state state.dim-row mat-of-cols-list-def by fastforce
show dim-col  $((CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle)) = dim\text{-}col v$ 
using assms(1) alice-step-2-state state.is-column asm mat-of-cols-list-def by
fastforce
show  $\bigwedge i j. i < dim\text{-}row v \implies j < dim\text{-}col v \implies ((CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle)) \$\$ (i,j) = v \$\$ (i,j)$ 
proof-
fix i j assume  $i < dim\text{-}row v$  and  $j < dim\text{-}col v$ 
then have  $i \in \{0..<8::nat\} \wedge j = 0$ 
using asm by (auto simp add: mat-of-cols-list-def)
then have  $(M1 * (\varphi \otimes |\beta_{00}\rangle)) \$\$ (i,j) = v \$\$ (i,j)$ 
by (auto simp add: f0 asm mat-of-cols-list-def times-mat-def scalar-prod-def)
then show  $((CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle)) \$\$ (i,j) = v \$\$ (i,j)$ 
using tensor-prod-of-cnot-id-1 by simp
qed
qed

lemma alice-result:
assumes state 1  $\varphi$  and  $\alpha = \varphi \$\$ (0,0)$  and  $\beta = \varphi \$\$ (1,0)$ 
shows alice  $\varphi = mat\text{-}of\text{-}cols\text{-}list 8 [[\alpha/2, \beta/2, \beta/2, \alpha/2, \alpha/2, -\beta/2, -\beta/2, \alpha/2]]$ 
proof
define v where  $a0:v = mat\text{-}of\text{-}cols\text{-}list 8 [[\alpha/2, \beta/2, \beta/2, \alpha/2, \alpha/2, -\beta/2, -\beta/2, \alpha/2]]$ 
define w where  $a1:w = (CNOT \otimes Id 1) * (\varphi \otimes |\beta_{00}\rangle)$ 
then have  $f0:w = mat\text{-}of\text{-}cols\text{-}list 8 [[\alpha/\sqrt{2}, 0, 0, \alpha/\sqrt{2}, 0, \beta/\sqrt{2}, \beta/\sqrt{2}, 0]]$ 
using assms alice-step-2 by simp
show dim-row (alice  $\varphi$ ) = dim-row v
using assms(1) alice-state state-def a0 by (simp add: mat-of-cols-list-def)
show dim-col (alice  $\varphi$ ) = dim-col v
using assms(1) alice-state state-def a0 by (simp add: mat-of-cols-list-def)
show  $\bigwedge i j. i < dim\text{-}row v \implies j < dim\text{-}col v \implies alice \varphi \$\$ (i,j) = v \$\$ (i,j)$ 
proof-
fix i j assume  $i < dim\text{-}row v$  and  $j < dim\text{-}col v$ 
then have  $i \in \{0..<8\} \wedge j = 0$ 
using a0 by (auto simp add: Tensor.mat-of-cols-list-def)

```

```

then have ( $M2 * w$ ) $$ (i,j) = v $$ (i,j)
  by (auto simp add: f0 a0 mat-of-cols-list-def times-mat-def scalar-prod-def)
then show alice  $\varphi$  $$ (i,j) = v $$ (i,j)
  by (simp add: tensor-prod-of-h-id-2 alice-def a1)
qed
qed

```

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

```

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

```



```

definition alice-pos:: complex Matrix.mat  $\Rightarrow$  complex Matrix.mat  $\Rightarrow$  bool where
  alice-pos  $\varphi$  q  $\equiv$  q = mat-of-cols-list 8 [[ $\varphi$  $$ (0,0),  $\varphi$  $$ (1,0), 0, 0, 0, 0, 0, 0]]  $\vee$ 
    q = mat-of-cols-list 8 [[0, 0,  $\varphi$  $$ (1,0),  $\varphi$  $$ (0,0), 0, 0, 0, 0]]  $\vee$ 
    q = mat-of-cols-list 8 [[0, 0, 0, 0,  $\varphi$  $$ (0,0), - $\varphi$  $$ (1,0), 0, 0]]  $\vee$ 
    q = mat-of-cols-list 8 [[0, 0, 0, 0, 0, 0, - $\varphi$  $$ (1,0),  $\varphi$  $$ (0,0)]]

```

lemma phi-vec-length:

```

assumes state 1  $\varphi$ 
shows cmod( $\varphi$  $$ (0,0)) \geq 2 + cmod(\varphi$$ $$(Suc 0,0)) \geq 2 = 1
using set-2 assms state-def Matrix.col-def cpx-vec-length-def by(auto simp add:
atLeast0LessThan)

```

lemma select-index-3-subsets [simp]:

```

shows {j::nat. select-index 3 0 j} = {4,5,6,7}  $\wedge$ 
  {j::nat. j < 8  $\wedge$   $\neg$  select-index 3 0 j} = {0,1,2,3}  $\wedge$ 
  {j::nat. select-index 3 1 j} = {2,3,6,7}  $\wedge$ 
  {j::nat. j < 8  $\wedge$   $\neg$  select-index 3 1 j} = {0,1,4,5}

```

proof-

```

have {j::nat. select-index 3 0 j} = {4,5,6,7} by (auto simp add: select-index-def)
moreover have {j::nat. j < 8  $\wedge$   $\neg$  select-index 3 0 j} = {0,1,2,3} by(auto simp
add: select-index-def)

```

```

moreover have f1:{j::nat. select-index 3 1 j} = {2,3,6,7}

```

proof

```

show {j. select-index 3 1 j}  $\subseteq$  {2,3,6,7}

```

proof

```

fix j::nat assume j  $\in$  {j. select-index 3 1 j}

```

```

then have  $j \in \{0..8\} \wedge j \bmod 4 \in \{2,3\}$  by (auto simp add: select-index-def)
  then show  $j \in \{2,3,6,7\}$  by auto
qed
show  $\{2,3,6,7\} \subseteq \{j. \text{select-index } 3 \ 1 \ j\}$  by (auto simp add: select-index-def)
qed
moreover have  $\{j::nat. j < 8 \wedge \neg \text{select-index } 3 \ 1 \ j\} = \{0,1,4,5\}$ 
proof-
  have  $\{j::nat. j < 8 \wedge j \notin \{2,3,6,7\}\} = \{0,1,4,5\}$  by auto
  then show ?thesis using f1 by blast
qed
ultimately show ?thesis by simp
qed

lemma prob-index-0-alice:
assumes state 1  $\varphi$ 
shows prob0 3 (alice  $\varphi$ ) 0 = 1/2  $\wedge$  prob1 3 (alice  $\varphi$ ) 0 = 1/2
proof
  show prob0 3 (alice  $\varphi$ ) 0 = 1/2
    using alice-result assms prob0-def alice-state
    apply auto by (metis (no-types, opaque-lifting) One-nat-def phi-vec-length
four-x-squared mult.commute
nonzero-mult-div-cancel-right times-divide-eq-right zero-neq-numeral)
  then show prob1 3 (alice  $\varphi$ ) 0 = 1/2
    using prob-sum-is-one[of 3 alice  $\varphi$  0] alice-state[of  $\varphi$ ] assms by linarith
qed

lemma post-meas0-index-0-alice:
assumes state 1  $\varphi$  and  $\alpha = \varphi \$\$ (0,0)$  and  $\beta = \varphi \$\$ (1,0)$ 
shows post-meas0 3 (alice  $\varphi$ ) 0 =
mat-of-cols-list 8 [[ $\alpha/\sqrt{2}$ ,  $\beta/\sqrt{2}$ ,  $\beta/\sqrt{2}$ ,  $\alpha/\sqrt{2}$ , 0, 0, 0, 0]]
proof
  define v where asm:v = mat-of-cols-list 8 [[ $\alpha/\sqrt{2}$ ,  $\beta/\sqrt{2}$ ,  $\beta/\sqrt{2}$ ,  $\alpha/\sqrt{2}$ , 0, 0, 0, 0]]
  then show dim-row (post-meas0 3 (alice  $\varphi$ ) 0) = dim-row v
    using mat-of-cols-list-def post-meas0-def assms(1) alice-state ket-vec-def by
simp
  show dim-col (post-meas0 3 (alice  $\varphi$ ) 0) = dim-col v
    using mat-of-cols-list-def post-meas0-def assms(1) alice-state ket-vec-def asm
by simp
  show  $\bigwedge i. i < \text{dim-row } v \implies j < \text{dim-col } v \implies \text{post-meas0 } 3 \ (\text{alice } \varphi) \ 0 \$\$ (i,j)$ 
= v  $\$ (i,j)$ 
proof-
  fix i j assume i < dim-row v and j < dim-col v
  then have i  $\in \{0..8\} \wedge j = 0$ 
    using asm set-8-atLeast0 mat-of-cols-list-def by auto
  then show post-meas0 3 (alice  $\varphi$ ) 0  $\$ (i, j) = v \$ (i, j)$ 
    using post-meas0-def assms asm mat-of-cols-list-def ket-vec-def
    apply (auto simp add: prob-index-0-alice)
    using assms(1) alice-result select-index-def by auto
qed

```

qed

```
lemma post-meas1-index-0-alice:
  assumes state 1  $\varphi$  and  $\alpha = \varphi \$\$ (0,0)$  and  $\beta = \varphi \$\$ (1,0)$ 
  shows post-meas1 3 (alice  $\varphi$ ) 0 = mat-of-cols-list 8 [[0,0,0,0, $\alpha/\sqrt{2}$ ], $-\beta/\sqrt{2}$ , $-\beta/\sqrt{2}$ , $\alpha/\sqrt{2}$ ]]
proof
  define v where  $asm:v = mat-of-cols-list 8 [[0,0,0,0, $\alpha/\sqrt{2}$ ], $-\beta/\sqrt{2}$ , $-\beta/\sqrt{2}$ , $\alpha/\sqrt{2}$ ]]$ 
  then show dim-row (post-meas1 3 (alice  $\varphi$ ) 0) = dim-row v
    using mat-of-cols-list-def post-meas1-def assms(1) alice-state ket-vec-def by
  simp
  show dim-col (post-meas1 3 (alice  $\varphi$ ) 0) = dim-col v
    using mat-of-cols-list-def post-meas1-def assms(1) alice-state ket-vec-def asm
  by simp
  show  $\bigwedge i j. i < dim\text{-}row v \implies j < dim\text{-}col v \implies post\text{-}meas1 3 (alice \varphi) 0 \$\$ (i,j)$ 
  = v \$\$ (i,j)
  proof-
    fix i j assume i < dim-row v and j < dim-col v
    then have i ∈ {0..<8} ∧ j = 0
      using asm set-8-atLeast0 mat-of-cols-list-def by auto
    then show post-meas1 3 (alice  $\varphi$ ) 0 \$\$ (i,j) = v \$\$ (i,j)
      using post-meas1-def assms asm mat-of-cols-list-def ket-vec-def
      apply (auto simp add: prob-index-0-alice)
      using assms(1) alice-result select-index-def by auto
  qed
qed
```

lemma post-meas0-index-0-alice-state [simp]:

```
assumes state 1  $\varphi$ 
shows state 3 (post-meas0 3 (alice  $\varphi$ ) 0)
using assms by (simp add: prob-index-0-alice)
```

lemma post-meas1-index-0-alice-state [simp]:

```
assumes state 1  $\varphi$ 
shows state 3 (post-meas1 3 (alice  $\varphi$ ) 0)
using assms by (simp add: prob-index-0-alice)
```

lemma Alice-case [simp]:

```
assumes state 1  $\varphi$  and state 3 q and List.member (alice-meas  $\varphi$ ) (p, q)
shows alice-pos  $\varphi$  q
proof-
  define  $\alpha$   $\beta$  where  $a0:\alpha = \varphi \$\$ (0,0)$  and  $a1:\beta = \varphi \$\$ (1,0)$ 
  have f0:prob0 3 (Matrix.mat 8 (Suc 0) ( $\lambda(i,j). [[\varphi \$\$ (0,0)/\sqrt{2}, \varphi \$\$ (Suc 0)/\sqrt{2}], [\varphi \$\$ (0,0)/\sqrt{2}, \varphi \$\$ (0,0)/\sqrt{2}], [\varphi \$\$ (0,0)/\sqrt{2}, \varphi \$\$ (0,0)/\sqrt{2}]] !j!i))$  (Suc 0)
  = 1/2
  using post-meas0-index-0-alice prob0-def mat-of-cols-list-def post-meas0-index-0-alice-state
  assms(1) a0 a1 select-index-3-subsets by (auto simp add: norm-divide power-divide
  phi-vec-length)
```

```

have f1:prob1 3 (Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ ). [[ $\varphi$  $$ (0,0)/\sqrt{2}, \varphi $$ (Suc 0,0)/sqrt 2,  

 $\varphi$  $$ (Suc 0,0)/sqrt 2,  $\varphi$  $$ (0,0)/sqrt 2, 0, 0, 0, 0]] ! j ! i)) (Suc 0) = 1/2
using post-meas0-index-0-alice prob1-def mat-of-cols-list-def post-meas0-index-0-alice-state

assms(1) a0 a1 select-index-3-subsets by(auto simp add: norm-divide power-divide
phi-vec-length algebra-simps)
have f2:prob0 3 (Matrix.mat 8 (Suc 0)
( $\lambda(i,j)$ . [[0,0,0,0, $\varphi$  $$ (0,0)/\text{complex-of-real}(\sqrt{2}), -(\varphi$$ $(Suc 0,0)/\text{complex-of-real}(\sqrt{2})),  

 $-(\varphi$  $$ (Suc 0,0)/\text{complex-of-real}(\sqrt{2})),  $\varphi$  $$ (0,0)/\text{complex-of-real}(\sqrt{2})]] ! j ! i)) (Suc 0) = 1/2
using post-meas1-index-0-alice prob0-def mat-of-cols-list-def post-meas1-index-0-alice-state

assms(1) a0 a1 select-index-3-subsets by(auto simp add: norm-divide power-divide
phi-vec-length)
have f3:prob1 3 (Matrix.mat 8 (Suc 0)
( $\lambda(i,j)$ . [[0,0,0,0, $\varphi$  $$ (0,0)/\text{complex-of-real}(\sqrt{2}), -(\varphi$$ $(Suc 0,0)/\text{complex-of-real}(\sqrt{2})),  

 $-(\varphi$  $$ (Suc 0,0)/\text{complex-of-real}(\sqrt{2})),  $\varphi$  $$ (0,0)/\text{complex-of-real}(\sqrt{2})]] ! j ! i)) (Suc 0) = 1/2
using post-meas1-index-0-alice prob1-def mat-of-cols-list-def post-meas1-index-0-alice-state

assms(1) a0 a1 select-index-3-subsets by(auto simp add: norm-divide power-divide
phi-vec-length algebra-simps)
have (p, q) = ((prob0 3 (alice  $\varphi$ ) 0) * (prob0 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1),  

post-meas0 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1)  $\vee$   

(p, q) = ((prob0 3 (alice  $\varphi$ ) 0) * (prob1 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1),  

post-meas1 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1)  $\vee$   

(p, q) = ((prob1 3 (alice  $\varphi$ ) 0) * (prob0 3 (post-meas1 3 (alice  $\varphi$ ) 0) 1),  

post-meas0 3 (post-meas1 3 (alice  $\varphi$ ) 0) 1)  $\vee$   

(p, q) = ((prob1 3 (alice  $\varphi$ ) 0) * (prob1 3 (post-meas1 3 (alice  $\varphi$ ) 0) 1),  

post-meas1 3 (post-meas1 3 (alice  $\varphi$ ) 0) 1)
using assms(3) alice-meas-def List.member-def by(smt (verit) list.distinct(1)
list.exhaust list.inject member-rec(1) member-rec(2))
then have q = post-meas0 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1  $\vee$  q = post-meas1 3  

(post-meas0 3 (alice  $\varphi$ ) 0) 1  $\vee$   

q = post-meas0 3 (post-meas1 3 (alice  $\varphi$ ) 0) 1  $\vee$  q = post-meas1 3  

(post-meas1 3 (alice  $\varphi$ ) 0) 1
by auto
moreover have post-meas0 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1 = mat-of-cols-list 8  

[[ $\alpha, \beta, 0, 0, 0, 0, 0, 0$ ]]
proof
define v where asm:v = mat-of-cols-list 8 [[ $\alpha, \beta, 0, 0, 0, 0, 0, 0$ ]]
then show dim-row (post-meas0 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1) = dim-row v
using mat-of-cols-list-def post-meas0-def ket-vec-def by simp
show dim-col (post-meas0 3 (post-meas0 3 (alice  $\varphi$ ) 0) 1) = dim-col v
using mat-of-cols-list-def post-meas0-def ket-vec-def asm by simp

```

```

show  $\bigwedge i j. i < \text{dim-row } v \implies j < \text{dim-col } v \implies \text{post-meas0 } 3 (\text{post-meas0 } 3 (\text{alice } \varphi) 0) 1 \$\$ (i,j) = v \$\$ (i,j)$ 
proof-
fix  $i j$  assume  $i < \text{dim-row } v$  and  $j < \text{dim-col } v$ 
then have  $i \in \{0..<8\} \wedge j = 0$ 
  using asm mat-of-cols-list-def by auto
then show  $\text{post-meas0 } 3 (\text{post-meas0 } 3 (\text{alice } \varphi) 0) 1 \$\$ (i,j) = v \$\$ (i,j)$ 
  using post-meas0-index-0-alice assms(1) a0 a1
  apply (auto)
  using post-meas0-def asm mat-of-cols-list-def ket-vec-def select-index-def
  by (auto simp add: f0 real-sqrt-divide)
qed
qed
moreover have  $\text{post-meas1 } 3 (\text{post-meas0 } 3 (\text{alice } \varphi) 0) 1 = \text{mat-of-cols-list } 8 [[0,0,\beta,\alpha,0,0,0,0]]$ 
proof
define  $v$  where asm:v = mat-of-cols-list 8 [[0,0,\beta,\alpha,0,0,0,0]]
then show  $\text{dim-row } (\text{post-meas1 } 3 (\text{post-meas0 } 3 (\text{alice } \varphi) 0) 1) = \text{dim-row } v$ 
  using mat-of-cols-list-def post-meas1-def ket-vec-def asm by auto
show  $\text{dim-col } (\text{post-meas1 } 3 (\text{post-meas0 } 3 (\text{alice } \varphi) 0) 1) = \text{dim-col } v$ 
  using mat-of-cols-list-def post-meas1-def ket-vec-def asm by auto
show  $\bigwedge i j. i < \text{dim-row } v \implies j < \text{dim-col } v \implies \text{post-meas1 } 3 (\text{post-meas0 } 3 (\text{alice } \varphi) 0) 1 \$\$ (i,j) = v \$\$ (i,j)$ 
proof-
fix  $i j$  assume  $i < \text{dim-row } v$  and  $j < \text{dim-col } v$ 
then have  $i \in \{0..<8\} \wedge j = 0$ 
  using asm mat-of-cols-list-def by auto
then show  $\text{post-meas1 } 3 (\text{post-meas0 } 3 (\text{alice } \varphi) 0) 1 \$\$ (i,j) = v \$\$ (i,j)$ 
  using post-meas0-index-0-alice assms(1) a0 a1
  apply (auto)
  using post-meas1-def asm mat-of-cols-list-def ket-vec-def select-index-def
  by (auto simp add: f1 real-sqrt-divide)
qed
qed
moreover have  $\text{post-meas0 } 3 (\text{post-meas1 } 3 (\text{alice } \varphi) 0) 1 = \text{mat-of-cols-list } 8 [[0,0,0,0,\alpha,-\beta,0,0]]$ 
proof
define  $v$  where asm:v = mat-of-cols-list 8 [[0,0,0,0,\alpha,-\beta,0,0]]
then show  $\text{dim-row } (\text{post-meas0 } 3 (\text{post-meas1 } 3 (\text{alice } \varphi) 0) 1) = \text{dim-row } v$ 
  using mat-of-cols-list-def post-meas0-def ket-vec-def by simp
show  $\text{dim-col } (\text{post-meas0 } 3 (\text{post-meas1 } 3 (\text{alice } \varphi) 0) 1) = \text{dim-col } v$ 
  using mat-of-cols-list-def post-meas0-def ket-vec-def asm by simp
show  $\bigwedge i j. i < \text{dim-row } v \implies j < \text{dim-col } v \implies \text{post-meas0 } 3 (\text{post-meas1 } 3 (\text{alice } \varphi) 0) 1 \$\$ (i,j) = v \$\$ (i,j)$ 
proof-
fix  $i j$  assume  $i < \text{dim-row } v$  and  $j < \text{dim-col } v$ 
then have  $i \in \{0..<8\} \wedge j = 0$ 
  using asm mat-of-cols-list-def by auto
then show  $\text{post-meas0 } 3 (\text{post-meas1 } 3 (\text{alice } \varphi) 0) 1 \$\$ (i,j) = v \$\$ (i,j)$ 

```

```

using post-meas1-index-0-alice assms(1) a0 a1
apply (auto)
using post-meas0-def asm mat-of-cols-list-def ket-vec-def select-index-def
by (auto simp add: f2 real-sqrt-divide)
qed
qed
moreover have post-meas1 3 (post-meas1 3 (alice φ) 0) 1 = mat-of-cols-list 8
[[0,0,0,0,0,0,-β,α]]
proof
define v where asm:v = mat-of-cols-list 8 [[0,0,0,0,0,0,-β,α]]
then show dim-row (post-meas1 3 (post-meas1 3 (alice φ) 0) 1) = dim-row v
using mat-of-cols-list-def post-meas1-def ket-vec-def by simp
show dim-col (post-meas1 3 (post-meas1 3 (alice φ) 0) 1) = dim-col v
using mat-of-cols-list-def post-meas1-def ket-vec-def asm by simp
show ⋀ i j. i < dim-row v ⟹ j < dim-col v ⟹ post-meas1 3 (post-meas1 3 (alice
φ) 0) 1 $$ (i,j) = v $$ (i,j)
proof-
fix i j assume i < dim-row v and j < dim-col v
then have i ∈ {0..<8} ∧ j = 0
using asm mat-of-cols-list-def by auto
then show post-meas1 3 (post-meas1 3 (alice φ) 0) 1 $$ (i,j) = v $$ (i,j)
using post-meas1-index-0-alice assms(1) a0 a1
apply (auto)
using post-meas1-def asm mat-of-cols-list-def ket-vec-def select-index-def
by (auto simp add: f3 real-sqrt-divide)
qed
qed
ultimately show ?thesis using alice-pos-def a0 a1 by simp
qed

```

```

datatype bit = zero | one

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

definition bob:: complex Matrix.mat => bit × bit ⇒ complex Matrix.mat where
bob q b ≡
if (fst b, snd b) = (zero, zero) then q else

```

```

if (fst b, snd b) = (zero, one) then (Id 2  $\otimes$  X) * q else
  if (fst b, snd b) = (one, zero) then (Id 2  $\otimes$  Z) * q else
    if (fst b, snd b) = (one, one) then (Id 2  $\otimes$  Z * X) * q else
      undefined

```

lemma alice-out-unique [simp]:

```

assumes state 1  $\varphi$ 
shows Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0,  $\varphi$  $$ (Suc 0, 0),  $\varphi$  $$ (0, 0), 0, 0, 0]])!j!i)  $\neq$ 
  Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[ $\varphi$  $$ (0, 0),  $\varphi$  $$ (Suc 0, 0), 0, 0, 0, 0]])!j!i)  $\wedge$ 
    Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0, 0, 0,  $\varphi$  $$ (0, 0),  $\neg\varphi$  $$ (Suc 0, 0), 0, 0]])!j!i)  $\neq$ 
      Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[ $\varphi$  $$ (0, 0),  $\varphi$  $$ (Suc 0, 0), 0, 0, 0, 0, 0]])!j!i)  $\wedge$ 
        Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0, 0, 0, 0, 0,  $\varphi$  $$ (Suc 0, 0),  $\varphi$  $$ (0, 0)]])!j!i)  $\neq$ 
          Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[ $\varphi$  $$ (0, 0),  $\varphi$  $$ (Suc 0, 0), 0, 0, 0, 0, 0]])!j!i)  $\wedge$ 
            Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0, 0, 0, 0, 0,  $\varphi$  $$ (0, 0),  $\neg\varphi$  $$ (Suc 0, 0), 0, 0]])!j!i)  $\neq$ 
              Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[ $\varphi$  $$ (0, 0),  $\varphi$  $$ (Suc 0, 0), 0, 0, 0, 0, 0]])!j!i)  $\wedge$ 
                Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0, 0, 0, 0, 0,  $\varphi$  $$ (0, 0),  $\neg\varphi$  $$ (Suc 0, 0), 0, 0]])!j!i)  $\neq$ 
                  Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[ $\varphi$  $$ (0, 0),  $\varphi$  $$ (Suc 0, 0), 0, 0, 0, 0, 0]])!j!i)  $\wedge$ 
                    Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0, 0, 0, 0, 0,  $\varphi$  $$ (0, 0),  $\varphi$  $$ (0, 0), 0, 0, 0]])!j!i)  $\neq$ 
                      Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[ $\varphi$  $$ (0, 0),  $\varphi$  $$ (Suc 0, 0), 0, 0, 0, 0, 0, 0]])!j!i)  $\wedge$ 
                        Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0, 0, 0, 0, 0,  $\varphi$  $$ (0, 0),  $\neg\varphi$  $$ (Suc 0, 0), 0, 0]])!j!i)  $\neq$ 
                          Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[ $\varphi$  $$ (0, 0),  $\varphi$  $$ (Suc 0, 0), 0, 0, 0, 0, 0, 0]])!j!i)  $\wedge$ 
                            Matrix.mat 8 (Suc 0) ( $\lambda(i,j)$ . [[0, 0, 0, 0, 0, 0,  $\varphi$  $$ (0, 0),  $\neg\varphi$  $$ (Suc 0, 0), 0, 0]])!j!i)

```

proof-

have f0: φ \$\$ (0,0) \neq 0 \vee φ \$\$ (1,0) \neq 0

using assms state-def Matrix.col-def cpx-vec-length-def set-2 **by**(auto simp add: atLeast0LessThan)

define v1 v2 v3 v4 **where** d0:v1 = Matrix.mat 8 1 ($\lambda(i,j)$. [[φ \$\$ (0,0), φ \$\$ (1,0), 0, 0, 0, 0, 0, 0]])!j!i)

and d1:v2 = Matrix.mat 8 1 ($\lambda(i,j)$. [[0, 0, φ \$\$ (1,0), φ \$\$ (0,0), 0, 0, 0, 0]])!j!i)

and d2:v3 = Matrix.mat 8 1 ($\lambda(i,j)$. [[0, 0, 0, 0, φ \$\$ (0,0), $\neg\varphi$ \$\$ (1,0), 0, 0]])!j!i)

and d3:v4 = Matrix.mat 8 1 ($\lambda(i,j)$. [[0, 0, 0, 0, 0, 0, φ \$\$ (0,0), $\neg\varphi$ \$\$ (1,0), φ \$\$ (0,0)]])!j!i)

then have (v1 \$\$ (0,0) \neq v2 \$\$ (0,0) \vee v1 \$\$ (1,0) \neq v2 \$\$ (1,0)) \wedge

(v1 \$\$ (0,0) \neq v3 \$\$ (0,0) \vee v1 \$\$ (1,0) \neq v3 \$\$ (1,0)) \wedge

(v1 \$\$ (0,0) \neq v4 \$\$ (0,0) \vee v1 \$\$ (1,0) \neq v4 \$\$ (1,0)) \wedge

(v2 \$\$ (2,0) \neq v3 \$\$ (2,0) \vee v2 \$\$ (3,0) \neq v3 \$\$ (3,0)) \wedge

(v2 \$\$ (2,0) \neq v4 \$\$ (2,0) \vee v2 \$\$ (3,0) \neq v4 \$\$ (3,0)) \wedge

(v3 \$\$ (4,0) \neq v4 \$\$ (4,0) \vee v3 \$\$ (5,0) \neq v4 \$\$ (5,0)) **using** f0 **by**

```

auto
  then have v1 ≠ v2 ∧ v1 ≠ v3 ∧ v1 ≠ v4 ∧ v2 ≠ v3 ∧ v2 ≠ v4 ∧ v3 ≠ v4 by
auto
  thus ?thesis by (auto simp add: d0 d1 d2 d3)
qed

```

```

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

```

```

lemma tensor-prod-of-id-2-x:
  shows (Id 2 ⊗ X) = M3
proof
  have f0:gate 3 (Id 2 ⊗ X)
    using X-is-gate tensor-gate[of 2 Id 2 1 X] by simp
  then show dim-row (Id 2 ⊗ X) = dim-row M3
    using gate-def by (simp add: mat-of-cols-list-def)
  show dim-col (Id 2 ⊗ X) = dim-col M3
    using f0 gate-def by (simp add: mat-of-cols-list-def)
  show ∀i j. i < dim-row M3 ⇒ j < dim-col M3 ⇒ (Id 2 ⊗ X) $$ (i,j) = M3 $$ (i,j)
  proof-
    fix i j assume i < dim-row M3 and j < dim-col M3
    then have i ∈ {0..<8} ∧ j ∈ {0..<8} by (auto simp add: mat-of-cols-list-def)
    then show (Id 2 ⊗ X) $$ (i,j) = M3 $$ (i,j)
      using Id-def X-def index-tensor-mat[of Id 2 4 4 X 2 2 i j] gate-def X-is-gate
      id-is-gate Id-def by (auto simp add: mat-of-cols-list-def X-def)
  qed
qed

```

```

abbreviation M4:: complex Matrix.mat where
M4 ≡ mat-of-cols-list 8 [[0, -1, 0, 0, 0, 0, 0, 0],
  [1, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, -1, 0, 0, 0, 0],
  [0, 0, 1, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, -1, 0, 0],
  [0, 0, 0, 0, 1, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, -1],
  [0, 0, 0, 0, 0, 0, 1, 0]]

```

```

abbreviation ZX:: complex Matrix.mat where
ZX ≡ mat-of-cols-list 2 [[0, -1], [1, 0]]

```

```

lemma l-inv-of-ZX:
  shows ZX† * ZX = 1m 2
proof
  show dim-row (ZX† * ZX) = dim-row (1m 2) using dagger-def mat-of-cols-list-def
  by simp
  show dim-col (ZX† * ZX) = dim-col (1m 2) using dagger-def mat-of-cols-list-def
  by simp
  show  $\bigwedge i j. i < \text{dim-row} (1_m 2) \implies j < \text{dim-col} (1_m 2) \implies (ZX^\dagger * ZX) \$\$ (i, j) = 1_m 2 \$\$ (i, j)$ 
proof-
  fix i j assume i < dim-row (1m 2) and j < dim-col (1m 2)
  then have i ∈ {0..<2} ∧ j ∈ {0..<2} by auto
  then show (ZX† * ZX) \$\$ (i, j) = 1m 2 \$\$ (i, j)
    using mat-of-cols-list-def dagger-def by (auto simp add: set-2)
qed
qed

lemma r-inv-of-ZX:
  shows ZX * (ZX†) = 1m 2
proof
  show dim-row (ZX * (ZX†)) = dim-row (1m 2) using dagger-def mat-of-cols-list-def
  by simp
  show dim-col (ZX * (ZX†)) = dim-col (1m 2) using dagger-def mat-of-cols-list-def
  by simp
  show  $\bigwedge i j. i < \text{dim-row} (1_m 2) \implies j < \text{dim-col} (1_m 2) \implies (ZX * (ZX^\dagger)) \$\$ (i, j) = 1_m 2 \$\$ (i, j)$ 
proof-
  fix i j assume i < dim-row (1m 2) and j < dim-col (1m 2)
  then have i ∈ {0..<2} ∧ j ∈ {0..<2} by auto
  then show (ZX * (ZX†)) \$\$ (i, j) = 1m 2 \$\$ (i, j)
    using mat-of-cols-list-def dagger-def by (auto simp add: set-2)
qed
qed

lemma ZX-is-gate [simp]:
  shows gate 1 ZX
proof
  show dim-row ZX = 2 ^ 1 using mat-of-cols-list-def by simp
  show square-mat ZX using mat-of-cols-list-def by simp
  show unitary ZX using unitary-def l-inv-of-ZX r-inv-of-ZX mat-of-cols-list-def
  by auto
qed

lemma prod-of-ZX:
  shows Z * X = ZX
proof
  show dim-row (Z * X) = dim-row ZX
    using Z-is-gate mat-of-cols-list-def gate-def by auto
  show dim-col (Z * X) = dim-col ZX

```

```

using X-is-gate mat-of-cols-list-def gate-def by auto
show  $\bigwedge i j. i < \text{dim-row } ZX \implies j < \text{dim-col } ZX \implies (Z * X) \$\$ (i, j) = ZX \$\$ (i, j)$ 
proof-
  fix i j assume i < dim-row ZX and j < dim-col ZX
  then have  $i \in \{0..<2\} \wedge j \in \{0..<2\}$  by (auto simp add: mat-of-cols-list-def)
    then show  $(Z * X) \$\$ (i, j) = ZX \$\$ (i, j)$  by (auto simp add: set-2 Z-def X-def)
  qed
qed

lemma tensor-prod-of-id-2-y:
shows  $(Id 2 \otimes Z * X) = M4$ 
proof
  have f0:gate 3  $(Id 2 \otimes Z * X)$ 
  using prod-of-ZX ZX-is-gate[of 2 Id 2 1 Z * X] by simp
  then show dim-row  $(Id 2 \otimes Z * X) = \text{dim-row } M4$ 
    using gate-def by (simp add: mat-of-cols-list-def)
  show dim-col  $(Id 2 \otimes Z * X) = \text{dim-col } M4$ 
    using f0 gate-def by (simp add: mat-of-cols-list-def)
  show  $\bigwedge i j. i < \text{dim-row } M4 \implies j < \text{dim-col } M4 \implies (Id 2 \otimes Z * X) \$\$ (i, j) = M4 \$\$ (i, j)$ 
proof-
  fix i j assume i < dim-row M4 and j < dim-col M4
  then have  $i \in \{0..<8\} \wedge j \in \{0..<8\}$  by (auto simp add: mat-of-cols-list-def)
  then have  $(Id 2 \otimes ZX) \$\$ (i, j) = M4 \$\$ (i, j)$ 
    using Id-def mat-of-cols-list-def index-tensor-mat[of Id 2 4 4 ZX 2 2 i j]
      gate-def ZX-is-gate id-is-gate
        by (auto simp add: mat-of-cols-list-def)
  then show  $(Id 2 \otimes Z * X) \$\$ (i, j) = M4 \$\$ (i, j)$ 
    using prod-of-ZX by simp
  qed
qed

abbreviation M5:: complex Matrix.mat where
M5 ≡ mat-of-cols-list 8 [[1, 0, 0, 0, 0, 0, 0, 0],
                           [0, -1, 0, 0, 0, 0, 0, 0],
                           [0, 0, 1, 0, 0, 0, 0, 0],
                           [0, 0, 0, -1, 0, 0, 0, 0],
                           [0, 0, 0, 0, 1, 0, 0, 0],
                           [0, 0, 0, 0, 0, -1, 0, 0],
                           [0, 0, 0, 0, 0, 0, 1, 0],
                           [0, 0, 0, 0, 0, 0, 0, -1]]

```

lemma tensor-prod-of-id-2-z:
shows $(Id 2 \otimes Z) = M5$

proof

have f0:gate 3 $(Id 2 \otimes Z)$

using Z-is-gate tensor-gate[of 2 Id 2 1 Z] **by** simp

```

then show dim-row ( $Id \otimes Z$ ) = dim-row  $M5$ 
  using gate-def by (simp add: mat-of-cols-list-def)
show dim-col ( $Id \otimes Z$ ) = dim-col  $M5$ 
  using f0 gate-def by (simp add: mat-of-cols-list-def)
show  $\bigwedge i j. i < \text{dim-row } M5 \implies j < \text{dim-col } M5 \implies (\mathit{Id} \otimes Z) \mathbin{\$\$} (i,j) = M5 \mathbin{\$\$} (i,j)$ 
proof-
  fix  $i j$  assume  $i < \text{dim-row } M5$  and  $j < \text{dim-col } M5$ 
  then have  $i \in \{0..8\} \wedge j \in \{0..8\}$  by (auto simp add: mat-of-cols-list-def)
  then show ( $\mathit{Id} \otimes Z$ )  $\mathbin{\$\$} (i, j) = M5 \mathbin{\$\$} (i,j)$ 
    using Id-def Z-def index-tensor-mat[of  $\mathit{Id} \otimes Z$ ] gate-def Z-is-gate
    id-is-gate Id-def by (auto simp add: mat-of-cols-list-def Z-def)
  qed
qed

```

lemma teleportation:

```

assumes state 1  $\varphi$  and state 3  $q$  and List.member (alice-meas  $\varphi$ ) ( $p, q$ )
shows  $\exists r. \text{state } 2 r \wedge \text{bob } q (\text{alice-out } \varphi q) = r \otimes \varphi$ 
proof-
  define  $\alpha \beta$  where  $a0:\alpha = \varphi \mathbin{\$\$} (0,0)$  and  $a1:\beta = \varphi \mathbin{\$\$} (1,0)$ 
  then have  $q = \text{mat-of-cols-list } 8 [[\alpha, \beta, 0, 0, 0, 0, 0, 0]] \vee$ 
     $q = \text{mat-of-cols-list } 8 [[0, 0, \beta, \alpha, 0, 0, 0, 0]] \vee$ 
     $q = \text{mat-of-cols-list } 8 [[0, 0, 0, 0, \alpha, -\beta, 0, 0]] \vee$ 
     $q = \text{mat-of-cols-list } 8 [[0, 0, 0, 0, 0, 0, 0, -\beta, \alpha]]$ 
  using assms Alice-case alice-pos-def by simp
  moreover have  $q = \text{mat-of-cols-list } 8 [[\alpha, \beta, 0, 0, 0, 0, 0, 0]] \implies \text{bob } q (\text{alice-out } \varphi q) =$ 
 $\text{mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi$ 
proof
  assume  $asm:q = \text{Tensor.mat-of-cols-list } 8 [[\alpha, \beta, 0, 0, 0, 0, 0, 0]]$ 
  show dim-row ( $\text{bob } q (\text{alice-out } \varphi q)$ ) = dim-row ( $\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi$ )
    using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm
    by simp
  show dim-col ( $\text{bob } q (\text{alice-out } \varphi q)$ ) = dim-col ( $\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi$ )
    using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm
    by simp
  show  $\bigwedge i j. i < \text{dim-row } (\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi) \implies$ 
     $j < \text{dim-col } (\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi) \implies$ 
     $\text{bob } q (\text{alice-out } \varphi q) \mathbin{\$\$} (i, j) = (\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi) \mathbin{\$\$} (i,j)$ 
proof-
  fix  $i j$  assume  $i < \text{dim-row } (\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi)$  and
     $j < \text{dim-col } (\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi)$ 
  then have  $i \in \{0..8\} \wedge j = 0$ 
    using asm mat-of-cols-list-def assms state-def by auto
  then show  $\text{bob } q (\text{alice-out } \varphi q) \mathbin{\$\$} (i,j) = (\text{Tensor.mat-of-cols-list } 4 [[1, 0, 0, 0]] \otimes \varphi) \mathbin{\$\$} (i,j)$ 

```

```

    using bob-def alice-out-def asm mat-of-cols-list-def a0 a1 assms state-def by
auto
qed
qed
moreover have  $q = \text{mat-of-cols-list } 8 [[0,0,\beta,\alpha,0,0,0,0]] \implies \text{bob } q (\text{alice-out } \varphi$ 
 $q) =$ 
 $\text{mat-of-cols-list } 4 [[0,1,0,0]] \otimes \varphi$ 
proof
assume  $\text{asm}:q = \text{Tensor.mat-of-cols-list } 8 [[0,0,\beta,\alpha,0,0,0,0]]$ 
show  $\text{dim-row}(\text{bob } q (\text{alice-out } \varphi \ q)) = \text{dim-row}(\text{Tensor.mat-of-cols-list } 4$ 
 $[[0,1,0,0]] \otimes \varphi)$ 
using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm
tensor-prod-of-id-2-x by simp
show  $\text{dim-col}(\text{bob } q (\text{alice-out } \varphi \ q)) = \text{dim-col}(\text{Tensor.mat-of-cols-list } 4$ 
 $[[0,1,0,0]] \otimes \varphi)$ 
using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm
by simp
show  $\bigwedge_{i,j} i < \text{dim-row}(\text{Tensor.mat-of-cols-list } 4 [[0,1,0,0]] \otimes \varphi) \implies$ 
 $j < \text{dim-col}(\text{Tensor.mat-of-cols-list } 4 [[0,1,0,0]] \otimes \varphi) \implies$ 
 $\text{bob } q (\text{alice-out } \varphi \ q) \$\$ (i,j) = (\text{Tensor.mat-of-cols-list } 4 [[0,1,0,0]] \otimes$ 
 $\varphi) \$\$ (i,j)$ 
proof-
fix i j assume  $i < \text{dim-row}(\text{Tensor.mat-of-cols-list } 4 [[0,1,0,0]] \otimes \varphi)$  and
 $j < \text{dim-col}(\text{Tensor.mat-of-cols-list } 4 [[0,1,0,0]] \otimes \varphi)$ 
then have c1:i ∈ {0..<8} ∧ j = 0
using asm mat-of-cols-list-def assms(1) state-def by auto
then have  $(M3 * (\text{Matrix.mat } 8 1 (\lambda(i,j). [[0,0,\varphi \$\$ (1,0),\varphi \$\$ (0,0),0,0,0,0]]!j!i)))$ 
 $\$\$ (i,j) =$ 
 $(\text{Tensor.mat-of-cols-list } 4 [[0,1,0,0]] \otimes \varphi) \$\$ (i,j)$ 
using state-def assms(1) by(auto simp add: a0 a1 mat-of-cols-list-def
times-mat-def scalar-prod-def)
then show  $\text{bob } q (\text{alice-out } \varphi \ q) \$\$ (i,j) = (\text{Tensor.mat-of-cols-list } 4 [[0,1,0,0]]$ 
 $\otimes \varphi) \$\$ (i,j)$ 
using bob-def alice-out-def asm c1 a0 a1 mat-of-cols-list-def tensor-prod-of-id-2-x
assms(1) by simp
qed
qed
moreover have  $q = \text{mat-of-cols-list } 8 [[0,0,0,0,\alpha,-\beta,0,0]] \implies \text{bob } q (\text{alice-out }$ 
 $\varphi \ q) =$ 
 $\text{mat-of-cols-list } 4 [[0,0,1,0]] \otimes \varphi$ 
proof
assume  $\text{asm}:q = \text{Tensor.mat-of-cols-list } 8 [[0,0,0,0,\alpha,-\beta,0,0]]$ 
show  $\text{dim-row}(\text{bob } q (\text{alice-out } \varphi \ q)) = \text{dim-row}(\text{Tensor.mat-of-cols-list } 4$ 
 $[[0,0,1,0]] \otimes \varphi)$ 
using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm
tensor-prod-of-id-2-z by simp
show  $\text{dim-col}(\text{bob } q (\text{alice-out } \varphi \ q)) = \text{dim-col}(\text{Tensor.mat-of-cols-list } 4$ 
 $[[0,0,1,0]] \otimes \varphi)$ 
using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm

```

```

by simp
show  $\bigwedge i j. i < \text{dim-row} (\text{Tensor.mat-of-cols-list } 4 [[0,0,1,0]] \otimes \varphi) \implies$ 
       $j < \text{dim-col} (\text{Tensor.mat-of-cols-list } 4 [[0,0,1,0]] \otimes \varphi) \implies$ 
       $\text{bob } q (\text{alice-out } \varphi q) \$\$ (i,j) = (\text{Tensor.mat-of-cols-list } 4 [[0,0,1,0]] \otimes$ 
 $\varphi) \$\$ (i,j)$ 
proof-
fix i j assume i < dim-row (Tensor.mat-of-cols-list 4 [[0,0,1,0]]  $\otimes \varphi$ ) and
      j < dim-col (Tensor.mat-of-cols-list 4 [[0,0,1,0]]  $\otimes \varphi$ )
then have c1:i ∈ {0..<8} ∧ j = 0
using asm mat-of-cols-list-def assms state-def by auto
then have (M5 * (Matrix.mat 8 (Suc 0) (λ(i,j). [[0,0,0,0,φ]] [[0,0,0,0,-φ]])) \$\$ (i,j) =
(Tensor.mat-of-cols-list 4 [[0,0,1,0]]  $\otimes \varphi) \$\$ (i,j)
using state-def assms(1) by(auto simp add: a0 a1 mat-of-cols-list-def times-mat-def scalar-prod-def)
then show bob q (alice-out φ q) \$\$ (i,j) = (Tensor.mat-of-cols-list 4 [[0,0,1,0]]  $\otimes \varphi) \$\$ (i,j)
using bob-def alice-out-def asm c1 a0 a1 mat-of-cols-list-def tensor-prod-of-id-2-z
assms(1) by simp
qed
qed
moreover have q = mat-of-cols-list 8 [[0, 0, 0, 0, 0, 0, 0, -β, α]]  $\implies$  bob q
(alice-out φ q) =
mat-of-cols-list 4 [[0, 0, 0, 1]]  $\otimes \varphi$ 
proof
assume asm:q = Tensor.mat-of-cols-list 8 [[0, 0, 0, 0, 0, 0, 0, -β, α]]
show dim-row (bob q (alice-out φ q)) = dim-row (Tensor.mat-of-cols-list 4 [[0,0,0,1]]  $\otimes \varphi$ )
using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm
tensor-prod-of-id-2-y by simp
show dim-col (bob q (alice-out φ q)) = dim-col (Tensor.mat-of-cols-list 4 [[0,0,0,1]]  $\otimes \varphi$ )
using mat-of-cols-list-def a0 a1 assms(1) state-def bob-def alice-out-def asm
by simp
show  $\bigwedge i j. i < \text{dim-row} (\text{Tensor.mat-of-cols-list } 4 [[0,0,0,1]] \otimes \varphi) \implies$ 
       $j < \text{dim-col} (\text{Tensor.mat-of-cols-list } 4 [[0,0,0,1]] \otimes \varphi) \implies$ 
       $\text{bob } q (\text{alice-out } \varphi q) \$\$ (i,j) = (\text{Tensor.mat-of-cols-list } 4 [[0,0,0,1]] \otimes$ 
 $\varphi) \$\$ (i,j)$ 
proof-
fix i j assume i < dim-row (Tensor.mat-of-cols-list 4 [[0,0,0,1]]  $\otimes \varphi$ ) and
      j < dim-col (Tensor.mat-of-cols-list 4 [[0,0,0,1]]  $\otimes \varphi$ )
then have c1:i ∈ {0..<8} ∧ j = 0
using asm mat-of-cols-list-def assms state-def by auto
then have (M4 * (Matrix.mat 8 (Suc 0) (λ(i,j). [[0,0,0,0,0,0,-φ]] [[0,0,0,0,0,0,φ]])) \$\$ (i,j) =
(Tensor.mat-of-cols-list 4 [[0,0,0,1]]  $\otimes \varphi) \$\$ (i,j)
using state-def assms(1) by(auto simp add: a0 a1 mat-of-cols-list-def times-mat-def scalar-prod-def)
then show bob q (alice-out φ q) \$\$ (i,j) = (Tensor.mat-of-cols-list 4 [[0,0,0,1]])$$$ 
```

```

⊗ φ) $$ (i,j)
  using bob-def alice-out-def asm c1 a0 a1 mat-of-cols-list-def tensor-prod-of-id-2-y
assms(1) by simp
qed
qed
moreover have state 2 (mat-of-cols-list 4 [[1, 0, 0, 0]])
  using state-def mat-of-cols-list-def cpx-vec-length-def lessThan-atLeast0 by simp
moreover have state 2 (mat-of-cols-list 4 [[0, 1, 0, 0]])
  using state-def mat-of-cols-list-def cpx-vec-length-def lessThan-atLeast0 by simp
moreover have state 2 (mat-of-cols-list 4 [[0, 0, 1, 0]])
  using state-def mat-of-cols-list-def cpx-vec-length-def lessThan-atLeast0 by simp
moreover have state 2 (mat-of-cols-list 4 [[0, 0, 0, 1]])
  using state-def mat-of-cols-list-def cpx-vec-length-def lessThan-atLeast0 by simp
ultimately show ?thesis by auto
qed

```

end

10 The Deutsch Algorithm

```

theory Deutsch
imports
  More-Tensor
  Measurement
begin

```

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

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

```

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

context deutsch
begin

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

lemma is-swap-values:
  assumes is-swap
  shows f 0 = 1 and f 1 = 0

```

```

using assms is-swap-def by auto

lemma is-swap-sum-mod-2:
  assumes is-swap
  shows ( $f 0 + f 1$ ) mod 2 = 1
  using assms is-swap-def by simp

definition const:: nat  $\Rightarrow$  bool where
  const  $n = (\forall x \in \{0,1\}. (f x = n))$ 

definition is-const:: bool where
  is-const  $\equiv$  const 0  $\vee$  const 1

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

lemma f-values: ( $f 0 = 0 \vee f 0 = 1$ )  $\wedge$  ( $f 1 = 0 \vee f 1 = 1$ )
  using dom by auto

lemma f-cases:
  shows is-const  $\vee$  is-balanced
  using dom is-balanced-def const-def is-const-def is-swap-def f-values by auto

lemma const-0-sum-mod-2:
  assumes const 0
  shows ( $f 0 + f 1$ ) mod 2 = 0
  using assms const-def by simp

lemma const-1-sum-mod-2:
  assumes const 1
  shows ( $f 0 + f 1$ ) mod 2 = 0
  using assms const-def by simp

lemma is-const-sum-mod-2:
  assumes is-const
  shows ( $f 0 + f 1$ ) mod 2 = 0
  using assms is-const-def const-0-sum-mod-2 const-1-sum-mod-2 by auto

lemma id-sum-mod-2:
  assumes  $f = id$ 
  shows ( $f 0 + f 1$ ) mod 2 = 1
  using assms by simp

lemma is-balanced-sum-mod-2:
  assumes is-balanced
  shows ( $f 0 + f 1$ ) mod 2 = 1
  using assms is-balanced-def id-sum-mod-2 is-swap-sum-mod-2 by auto

lemma f-ge-0:  $\forall x. (f x \geq 0)$  by simp

```

end

The Deutsch's Transform U_f .

definition (in deutsch) deutsch-transform:: complex Matrix.mat ($\langle U_f \rangle$) where
 $U_f \equiv \text{mat-of-cols-list } 4 [[1 - f(0), f(0), 0, 0],$
 $[f(0), 1 - f(0), 0, 0],$
 $[0, 0, 1 - f(1), f(1)],$
 $[0, 0, f(1), 1 - f(1)]]$

lemma (in deutsch) deutsch-transform-dim [simp]:
shows dim-row $U_f = 4$ and dim-col $U_f = 4$
by (auto simp add: deutsch-transform-def mat-of-cols-list-def)

lemma (in deutsch) deutsch-transform-coeff-is-zero [simp]:
shows $U_f \$\$ (0,2) = 0$ and $U_f \$\$ (0,3) = 0$
and $U_f \$\$ (1,2) = 0$ and $U_f \$\$ (1,3) = 0$
and $U_f \$\$ (2,0) = 0$ and $U_f \$\$ (2,1) = 0$
and $U_f \$\$ (3,0) = 0$ and $U_f \$\$ (3,1) = 0$
using deutsch-transform-def by auto

lemma (in deutsch) deutsch-transform-coeff [simp]:
shows $U_f \$\$ (0,1) = f(0)$ and $U_f \$\$ (1,0) = f(0)$
and $U_f \$\$ (2,3) = f(1)$ and $U_f \$\$ (3,2) = f(1)$
and $U_f \$\$ (0,0) = 1 - f(0)$ and $U_f \$\$ (1,1) = 1 - f(0)$
and $U_f \$\$ (2,2) = 1 - f(1)$ and $U_f \$\$ (3,3) = 1 - f(1)$
using deutsch-transform-def by auto

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

lemma (in deutsch) deutsch-transform-alt-rep-coeff-is-zero [simp]:
shows $V_f \$\$ (0,2) = 0$ and $V_f \$\$ (0,3) = 0$
and $V_f \$\$ (1,2) = 0$ and $V_f \$\$ (1,3) = 0$
and $V_f \$\$ (2,0) = 0$ and $V_f \$\$ (2,1) = 0$
and $V_f \$\$ (3,0) = 0$ and $V_f \$\$ (3,1) = 0$
by auto

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

and $V_f \$(0,0) = 1 - f(0)$ **and** $V_f \$(1,1) = 1 - f(0)$
and $V_f \$(2,2) = 1 - f(1)$ **and** $V_f \$(3,3) = 1 - f(1)$
by auto

```

lemma (in deutsch) deutsch-transform-alt-rep:
  shows  $U_f = V_f$ 
proof
  show  $c0:\dim\text{-row } U_f = \dim\text{-row } V_f$  by simp
  show  $c1:\dim\text{-col } U_f = \dim\text{-col } V_f$  by simp
  fix  $i j :: nat$ 
  assume  $i < \dim\text{-row } V_f$  and  $j < \dim\text{-col } V_f$ 
  then have  $i < 4$  and  $j < 4$  by auto
  thus  $U_f \$(i,j) = V_f \$(i,j)$ 
  by (smt (verit) deutsch-transform-alt-rep-coeff deutsch-transform-alt-rep-coeff-is-zero
    deutsch-transform-coeff
    deutsch-transform-coeff-is-zero set-4-disj)
qed

```

U_f is a gate.

```

lemma (in deutsch) transpose-of-deutsch-transform:
  shows  $(U_f)^t = U_f$ 
proof
  show  $\dim\text{-row } (U_f^t) = \dim\text{-row } U_f$  by simp
  show  $\dim\text{-col } (U_f^t) = \dim\text{-col } U_f$  by simp
  fix  $i j :: nat$ 
  assume  $i < \dim\text{-row } U_f$  and  $j < \dim\text{-col } U_f$ 
  thus  $U_f^t \$(i,j) = U_f \$(i,j)$ 
  by (auto simp add: transpose-mat-def)
  (metis deutsch-transform-coeff(1-4) deutsch-transform-coeff-is-zero set-4-disj)
qed

```

```

lemma (in deutsch) adjoint-of-deutsch-transform:
  shows  $(U_f)^\dagger = U_f$ 
proof
  show  $\dim\text{-row } (U_f^\dagger) = \dim\text{-row } U_f$  by simp
  show  $\dim\text{-col } (U_f^\dagger) = \dim\text{-col } U_f$  by simp
  fix  $i j :: nat$ 
  assume  $i < \dim\text{-row } U_f$  and  $j < \dim\text{-col } U_f$ 
  thus  $U_f^\dagger \$(i,j) = U_f \$(i,j)$ 
  by (auto simp add: dagger-def)
  (metis complex-cnj-of-nat complex-cnj-zero deutsch-transform-coeff
    deutsch-transform-coeff-is-zero set-4-disj)
qed

```

```

lemma (in deutsch) deutsch-transform-is-gate:
  shows gate 2  $U_f$ 
proof
  show  $\dim\text{-row } U_f = 2^2$  by simp
  show square-mat  $U_f$  by simp

```

```

show unitary  $U_f$ 
proof-
  have  $U_f * U_f = 1_m$  (dim-col  $U_f$ )
  proof
    show dim-row ( $U_f * U_f$ ) = dim-row ( $1_m$  (dim-col  $U_f$ )) by simp
  next
    show dim-col ( $U_f * U_f$ ) = dim-col ( $1_m$  (dim-col  $U_f$ )) by simp
  next
    fix  $i j :: nat$ 
    assume  $i < \text{dim-row} (1_m (\text{dim-col } U_f))$  and  $j < \text{dim-col} (1_m (\text{dim-col } U_f))$ 
    then show ( $U_f * U_f$ )  $\$\$ (i, j) = 1_m (\text{dim-col } U_f) \$\$ (i, j)$ 
    apply (auto simp add: deutsch-transform-alt-rep one-mat-def times-mat-def)
    apply (auto simp: scalar-prod-def)
    using f-values by auto
  qed
  thus ?thesis by (simp add: adjoint-of-deutsch-transform unitary-def)
  qed
  qed

```

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

```

abbreviation zero where zero  $\equiv$  unit-vec 2 0
abbreviation one where one  $\equiv$  unit-vec 2 1

```

```

lemma ket-zero-is-state:
  shows state 1  $|zero\rangle$ 
  by (simp add: state-def ket-vec-def cpx-vec-length-def numerals(2))

```

```

lemma ket-one-is-state:
  shows state 1  $|one\rangle$ 
  by (simp add: state-def ket-vec-def cpx-vec-length-def numerals(2))

```

```

lemma ket-zero-to-mat-of-cols-list [simp]:  $|zero\rangle = \text{mat-of-cols-list } 2 [[1, 0]]$ 
  by (auto simp add: ket-vec-def mat-of-cols-list-def)

```

```

lemma ket-one-to-mat-of-cols-list [simp]:  $|one\rangle = \text{mat-of-cols-list } 2 [[0, 1]]$ 
  apply (auto simp add: ket-vec-def unit-vec-def mat-of-cols-list-def)
  using less-2-cases by fastforce

```

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

```

abbreviation  $\psi_{00} :: \text{complex Matrix.mat}$  where
 $\psi_{00} \equiv \text{mat-of-cols-list } 2 [[1/\sqrt{2}, 1/\sqrt{2}]]$ 

```

```

lemma H-on-ket-zero:
  shows ( $H * |zero\rangle$ ) =  $\psi_{00}$ 
proof
  fix  $i j :: nat$ 

```

```

assume  $i < \text{dim-row } \psi_{00}$  and  $j < \text{dim-col } \psi_{00}$ 
then have  $i \in \{0,1\} \wedge j = 0$  by (simp add: mat-of-cols-list-def less-2-cases)
then show  $(H * |\text{zero}\rangle) \$\$ (i,j) = \psi_{00} \$\$ (i,j)$ 
    by (auto simp add: mat-of-cols-list-def times-mat-def scalar-prod-def H-def)
next
    show  $\text{dim-row } (H * |\text{zero}\rangle) = \text{dim-row } \psi_{00}$  by (simp add: H-def mat-of-cols-list-def)
    show  $\text{dim-col } (H * |\text{zero}\rangle) = \text{dim-col } \psi_{00}$  by (simp add: H-def mat-of-cols-list-def)
qed

lemma H-on-ket-zero-is-state:
shows state 1  $(H * |\text{zero}\rangle)$ 
proof
    show gate 1  $H$ 
        using H-is-gate by simp
    show state 1  $|\text{zero}\rangle$ 
        using ket-zero-is-state by simp
qed

```

Applying the Hadamard gate to the state $|0\rangle$ results in the new state $\psi_{01} = \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$.

```

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

```

```

lemma H-on-ket-one:
shows  $(H * |\text{one}\rangle) = \psi_{01}$ 
proof
    fix  $i j :: \text{nat}$ 
    assume  $i < \text{dim-row } \psi_{01}$  and  $j < \text{dim-col } \psi_{01}$ 
    then have  $i \in \{0,1\} \wedge j = 0$  by (simp add: mat-of-cols-list-def less-2-cases)
    then show  $(H * |\text{one}\rangle) \$\$ (i,j) = \psi_{01} \$\$ (i,j)$ 
        by (auto simp add: mat-of-cols-list-def times-mat-def scalar-prod-def H-def ket-vec-def)
next
    show  $\text{dim-row } (H * |\text{one}\rangle) = \text{dim-row } \psi_{01}$  by (simp add: H-def mat-of-cols-list-def)
    show  $\text{dim-col } (H * |\text{one}\rangle) = \text{dim-col } \psi_{01}$  by (simp add: H-def mat-of-cols-list-def ket-vec-def)
qed

```

```

lemma H-on-ket-one-is-state:
shows state 1  $(H * |\text{one}\rangle)$ 
using H-is-gate ket-one-is-state by simp

```

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

```

abbreviation  $\psi_1 :: \text{complex Matrix.mat where}$ 
 $\psi_1 \equiv \text{mat-of-cols-list 4 } [[1/2, -1/2, 1/2, -1/2]]$ 

```

```

lemma  $\psi_0$ -to- $\psi_1$ :
  shows  $(\psi_{00} \otimes \psi_{01}) = \psi_1$ 
proof
  fix  $i j :: nat$ 
  assume  $i < dim\text{-}row \psi_1$  and  $j < dim\text{-}col \psi_1$ 
  then have  $i \in \{0,1,2,3\}$  and  $j = 0$  using mat-of-cols-list-def by auto
  moreover have complex-of-real (sqrt 2) * complex-of-real (sqrt 2) = 2
    by (metis mult-2-right numeral-Bit0 of-real-mult of-real-numeral real-sqrt-four
real-sqrt-mult)
  ultimately show  $(\psi_{00} \otimes \psi_{01}) \$(i,j) = \psi_1 \$(i,j)$  using mat-of-cols-list-def
by auto
next
  show dim-row  $(\psi_{00} \otimes \psi_{01}) = dim\text{-}row \psi_1$  by (simp add: mat-of-cols-list-def)
  show dim-col  $(\psi_{00} \otimes \psi_{01}) = dim\text{-}col \psi_1$  by (simp add: mat-of-cols-list-def)
qed

```

```

lemma  $\psi_1$ -is-state:
  shows state 2  $\psi_1$ 
proof
  show dim-col  $\psi_1 = 1$ 
    by (simp add: Tensor.mat-of-cols-list-def)
  show dim-row  $\psi_1 = 2^2$ 
    by (simp add: Tensor.mat-of-cols-list-def)
  show  $\|Matrix.col \psi_1 0\| = 1$ 
    using H-on-ket-one-is-state H-on-ket-zero-is-state state.is-normal tensor-state2
 $\psi_0$ -to- $\psi_1$ 
  H-on-ket-one H-on-ket-zero by force
qed

```

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

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

```

lemma (in deutsch)  $\psi_1$ -to- $\psi_2$ :
  shows  $U_f * \psi_1 = \psi_2$ 
proof
  fix  $i j :: nat$ 
  assume  $i < dim\text{-}row \psi_2$  and  $j < dim\text{-}col \psi_2$ 
  then have asm:i  $\in \{0,1,2,3\} \wedge j = 0$  by (auto simp add: mat-of-cols-list-def)
  then have  $i < dim\text{-}row U_f \wedge j < dim\text{-}col \psi_1$ 

```

```

using deutsch-transform-def mat-of-cols-list-def by auto
then have ( $U_f * \psi_1$ ) $$ (i, j)
= (\sum k \in \{0 .. < \text{dim-vec } \psi_1\}. (\text{Matrix.row } U_f i) \$ k * (\text{Matrix.col } \psi_1 j)
\$ k)
apply (auto simp add: times-mat-def scalar-prod-def).
thus ( $U_f * \psi_1$ ) $$ (i, j) =  $\psi_2$  $$ (i, j)
using mat-of-cols-list-def deutsch-transform-def asm by auto
next
show dim-row ( $U_f * \psi_1$ ) = dim-row  $\psi_2$  by (simp add: mat-of-cols-list-def)
show dim-col ( $U_f * \psi_1$ ) = dim-col  $\psi_2$  by (simp add: mat-of-cols-list-def)
qed

lemma (in deutsch)  $\psi_2$ -is-state:
shows state 2  $\psi_2$ 
proof
show dim-col  $\psi_2$  = 1
by (simp add: Tensor.mat-of-cols-list-def)
show dim-row  $\psi_2$  =  $2^2$ 
by (simp add: Tensor.mat-of-cols-list-def)
show  $\|\text{Matrix.col } \psi_2 0\| = 1$ 
using gate-on-state-is-state  $\psi_1$ -is-state deutsch-transform-is-gate  $\psi_1$ -to- $\psi_2$  state-def
by (metis (no-types, lifting))
qed

lemma  $H$ -tensor-Id-1:
defines  $d:v \equiv$  mat-of-cols-list 4 [[1/sqrt(2), 0, 1/sqrt(2), 0],
[0, 1/sqrt(2), 0, 1/sqrt(2)],
[1/sqrt(2), 0, -1/sqrt(2), 0],
[0, 1/sqrt(2), 0, -1/sqrt(2)]]
shows ( $H \otimes \text{Id } 1$ ) =  $v$ 
proof
show dim-col ( $H \otimes \text{Id } 1$ ) = dim-col  $v$ 
by (simp add: d H-def Id-def mat-of-cols-list-def)
show dim-row ( $H \otimes \text{Id } 1$ ) = dim-row  $v$ 
by (simp add: d H-def Id-def mat-of-cols-list-def)
fix  $i j :: \text{nat}$  assume  $i < \text{dim-row } v$  and  $j < \text{dim-col } v$ 
then have  $i \in \{0..<4\} \wedge j \in \{0..<4\}$ 
by (auto simp add: d mat-of-cols-list-def)
thus ( $H \otimes \text{Id } 1$ ) $$ (i, j) =  $v$  $$ (i, j)
by (auto simp add: d Id-def H-def mat-of-cols-list-def)
qed

lemma  $H$ -tensor-Id-1-is-gate:
shows gate 2 ( $H \otimes \text{Id } 1$ )
proof
show dim-row ( $H \otimes \text{Quantum.Id } 1$ ) =  $2^2$ 
using H-tensor-Id-1 by (simp add: mat-of-cols-list-def)
show square-mat ( $H \otimes \text{Quantum.Id } 1$ )
using H-is-gate id-is-gate tensor-gate-sqr-mat by blast

```

```

show unitary ( $H \otimes Quantum.Id\ 1$ )
  using H-is-gate gate-def id-is-gate tensor-gate by blast
qed

Applying the Hadamard gate to the first qubit of  $\psi_2$  results in  $\psi_3 = \pm|f(0)\oplus f(1)\rangle \left[ \frac{(|0\rangle - |1\rangle)}{\sqrt{2}} \right]$ 

abbreviation (in deutsch)  $\psi_3$ :: complex Matrix.mat where
 $\psi_3 \equiv mat\text{-}of\text{-}cols\text{-}list\ 4$ 
 $[(1-f(0))/(2*sqrt(2)) - f(0)/(2*sqrt(2)) + (1-f(1))/(2*sqrt(2)) - f(1)/(2*sqrt(2)),$ 
 $f(0)/(2*sqrt(2)) - (1-f(0))/(2*sqrt(2)) + (f(1)/(2*sqrt(2)) - (1-f(1))/(2*sqrt(2))),$ 
 $(1-f(0))/(2*sqrt(2)) - f(0)/(2*sqrt(2)) - (1-f(1))/(2*sqrt(2)) + f(1)/(2*sqrt(2)),$ 
 $f(0)/(2*sqrt(2)) - (1-f(0))/(2*sqrt(2)) - f(1)/(2*sqrt(2)) + (1-f(1))/(2*sqrt(2))]]$ 

lemma (in deutsch)  $\psi_2$ -to- $\psi_3$ :
  shows ( $H \otimes Id\ 1$ ) *  $\psi_2 = \psi_3$ 
proof
  fix i j:: nat
  assume i < dim-row  $\psi_3$  and j < dim-col  $\psi_3$ 
  then have a0:i  $\in \{0,1,2,3\}$   $\wedge$  j = 0 by (auto simp add: mat-of-cols-list-def)
  then have i < dim-row ( $H \otimes Id\ 1$ )  $\wedge$  j < dim-col  $\psi_2$ 
    using mat-of-cols-list-def H-tensor-Id-1 by auto
  then have (( $H \otimes Id\ 1$ )* $\psi_2$ ) $$ (i,j)
    = (\sum k \in \{0 .. < dim\text{-}vec \psi_2\}. (Matrix.row ( $H \otimes Id\ 1$ ) i) \$ k * (Matrix.col
 $\psi_2$  j) \$ k)
    by (auto simp: times-mat-def scalar-prod-def)
  thus (( $H \otimes Id\ 1$ )* $\psi_2$ ) $$ (i, j) =  $\psi_3$  $$ (i, j)
    using mat-of-cols-list-def H-tensor-Id-1 a0 f-ge-0
    by (auto simp: diff-divide-distrib)
next
  show dim-row (( $H \otimes Id\ 1$ )* $\psi_2$ ) = dim-row  $\psi_3$ 
    using H-tensor-Id-1 mat-of-cols-list-def by simp
  show dim-col (( $H \otimes Id\ 1$ )* $\psi_2$ ) = dim-col  $\psi_3$ 
    using H-tensor-Id-1 mat-of-cols-list-def by simp
qed

lemma (in deutsch)  $\psi_3$ -is-state:
  shows state 2  $\psi_3$ 
proof-
  have gate 2 ( $H \otimes Id\ 1$ )
    using H-tensor-Id-1-is-gate by simp
  thus state 2  $\psi_3$ 
    using  $\psi_2$ -is-state  $\psi_2$ -to- $\psi_3$  by (metis gate-on-state-is-state)
qed

```

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

always succeed.

```

definition (in deutsch) deutsch-algo:: complex Matrix.mat where
  deutsch-algo ≡ (H ⊗ Id 1) * (U_f * ((H * |zero⟩) ⊗ (H * |one⟩)))

lemma (in deutsch) deutsch-algo-result [simp]:
  shows deutsch-algo = ψ3
  using deutsch-algo-def H-on-ket-zero H-on-ket-one ψ0-to-ψ1 ψ1-to-ψ2 ψ2-to-ψ3
  by simp

lemma (in deutsch) deutsch-algo-result-is-state:
  shows state 2 deutsch-algo
  using ψ3-is-state by simp

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

lemma (in deutsch) prob0-deutsch-algo-const:
  assumes is-const
  shows prob0 2 deutsch-algo 0 = 1
proof –
  have {k| k::nat. (k < 4) ∧ ¬ select-index 2 0 k} = {0,1}
  using select-index-def by auto
  then have prob0 2 deutsch-algo 0 = (∑ j ∈ {0,1}. (cmod(deutsch-algo $$ (j,0)))2)
  using deutsch-algo-result-is-state prob0-def by simp
  thus prob0 2 deutsch-algo 0 = 1
  using assms is-const-def const-def by auto
qed

lemma (in deutsch) prob1-deutsch-algo-const:
  assumes is-const
  shows prob1 2 deutsch-algo 0 = 0
  using assms prob0-deutsch-algo-const prob-sum-is-one[of 2 deutsch-algo 0]
  deutsch-algo-result-is-state by simp

```

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

```

lemma (in deutsch) prob0-deutsch-algo-balanced:
  assumes is-balanced
  shows prob0 2 deutsch-algo 0 = 0
proof –
  have {k| k::nat. (k < 4) ∧ ¬ select-index 2 0 k} = {0,1}
  using select-index-def by auto
  then have prob0 2 deutsch-algo 0 = (∑ j ∈ {0,1}. (cmod(deutsch-algo $$ (j,0)))2)
  using deutsch-algo-result-is-state prob0-def by simp
  thus prob0 2 deutsch-algo 0 = 0
  using is-swap-values assms is-balanced-def by auto
qed

```

```

lemma (in deutsch) prob1-deutsch-algo-balanced:
  assumes is-balanced
  shows prob1 2 deutsch-algo 0 = 1
  using assms prob0-deutsch-algo-balanced prob-sum-is-one[of 2 deutsch-algo 0]
  deutsch-algo-result-is-state by simp

```

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

```

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

```

```

lemma (in deutsch) sum-mod-2-cases:
  shows (f 0 + f 1) mod 2 = 0 → is-const
  and (f 0 + f 1) mod 2 = 1 → is-balanced
  using f-cases is-balanced-sum-mod-2 is-const-sum-mod-2 by auto

```

```

lemma (in deutsch) deutsch-algo-eval-is-sum-mod-2:
  shows deutsch-algo-eval = (f 0 + f 1) mod 2
  using deutsch-algo-eval-def f-cases is-const-sum-mod-2 is-balanced-sum-mod-2
  prob1-deutsch-algo-const prob1-deutsch-algo-balanced by auto

```

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

```

theorem (in deutsch) deutsch-algo-is-correct:
  shows deutsch-algo-eval = 0 → is-const and deutsch-algo-eval = 1 → is-balanced
  using deutsch-algo-eval-is-sum-mod-2 sum-mod-2-cases by auto

```

end

11 The Deutsch-Jozsa Algorithm

```

theory Deutsch-Jozsa
imports
  Deutsch
  More-Tensor
  Binary-Nat
begin

```

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

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

```

locale bob-fun =
  fixes f:: nat ⇒ nat and n:: nat
  assumes dom:  $f \in (\{(i::nat). i < 2^n\} \rightarrow_E \{0,1\})$ 
  assumes dim:  $n \geq 1$ 

```

```

context bob-fun
begin

definition const:: nat  $\Rightarrow$  bool where
const c = ( $\forall x \in \{i:\text{nat}. i < 2^n\}$ . f x = c)

definition is-const:: bool where
is-const  $\equiv$  const 0  $\vee$  const 1

definition is-balanced:: bool where
is-balanced  $\equiv$   $\exists A B ::\text{nat set}$ . A  $\subseteq \{i:\text{nat}. i < 2^n\}$   $\wedge$  B  $\subseteq \{i:\text{nat}. i < 2^n\}$ 
 $\wedge$  card A =  $2^{n-1}$   $\wedge$  card B =  $2^{n-1}$ 
 $\wedge$  ( $\forall x \in A$ . f x = 0)  $\wedge$  ( $\forall x \in B$ . f x = 1)

lemma is-balanced-inter:
fixes A B:: nat set
assumes  $\forall x \in A$ . f x = 0 and  $\forall x \in B$ . f x = 1
shows A  $\cap$  B = {}
using assms by auto

lemma is-balanced-union:
fixes A B:: nat set
assumes A  $\subseteq \{i:\text{nat}. i < 2^n\}$  and B  $\subseteq \{i:\text{nat}. i < 2^n\}$ 
and card A =  $2^{n-1}$  and card B =  $2^{n-1}$ 
and A  $\cap$  B = {}
shows A  $\cup$  B =  $\{i:\text{nat}. i < 2^n\}$ 
proof–
have finite A and finite B
by (simp add: assms(3) card-ge-0-finite)
(simp add: assms(4) card-ge-0-finite)
then have card(A  $\cup$  B) =  $2 * 2^{n-1}$ 
using assms(3-5) by (simp add: card-Un-disjoint)
then have card(A  $\cup$  B) =  $2^n$ 
by (metis Nat.nat.simps(3) One-nat-def dim le-0-eq power-eq-if)
moreover have ... = card( $\{i:\text{nat}. i < 2^n\}$ ) by simp
moreover have A  $\cup$  B  $\subseteq \{i:\text{nat}. i < 2^n\}$ 
using assms(1,2) by simp
moreover have finite ( $\{i:\text{nat}. i < 2^n\}$ ) by simp
ultimately show ?thesis
using card-subset-eq[of  $\{i:\text{nat}. i < 2^n\}$  A  $\cup$  B] by simp
qed

lemma f-ge-0:  $\forall x$ . f x  $\geq 0$  by simp

lemma f-dom-not-zero:
shows f  $\in (\{i:\text{nat}. n \geq 1 \wedge i < 2^n\} \rightarrow_E \{0,1\})$ 
using dim dom by simp

```

```
lemma f-values:  $\forall x \in \{(i::nat). i < 2^n\} . f x = 0 \vee f x = 1$ 
  using dom by auto
```

```
end
```

The input function has to be constant or balanced.

```
locale jozsa = bob-fun +
assumes const-or-balanced: is-const  $\vee$  is-balanced
```

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

```
lemma disj-four-cases:
```

```
  assumes A  $\vee$  B  $\vee$  C  $\vee$  D and A  $\implies$  P and B  $\implies$  P and C  $\implies$  P and D
 $\implies$  P
  shows P
  using assms by auto
```

The unitary transform U_f .

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

```
lemma (in jozsa) jozsa-transform-dim [simp]:
```

```
  shows dim-row  $U_f = 2^{(n+1)}$  and dim-col  $U_f = 2^{(n+1)}$ 
  by (auto simp add: jozsa-transform-def)
```

```
lemma (in jozsa) jozsa-transform-coeff-is-zero [simp]:
```

```
  assumes i < dim-row  $U_f \wedge j < \text{dim-col } U_f$ 
  shows (i  $\neq$  j  $\wedge \neg(i=j+1 \wedge \text{odd } i) \wedge \neg(i=j-1 \wedge \text{even } i \wedge j \geq 1)$ )  $\longrightarrow U_f \$\$ (i,j) = 0$ 
  using jozsa-transform-def assms by auto
```

```
lemma (in jozsa) jozsa-transform-coeff [simp]:
```

```
  assumes i < dim-row  $U_f \wedge j < \text{dim-col } U_f$ 
  shows i = j  $\longrightarrow U_f \$\$ (i,j) = 1 - f(i \text{ div } 2)$ 
    and i = j + 1  $\wedge \text{odd } i \longrightarrow U_f \$\$ (i,j) = f(i \text{ div } 2)$ 
    and j  $\geq 1 \wedge i = j - 1 \wedge \text{even } i \longrightarrow U_f \$\$ (i,j) = f(i \text{ div } 2)$ 
  using jozsa-transform-def assms by auto
```

```
lemma (in jozsa)  $U_f$ -mult-without-empty-summands-sum-even:
```

```
  fixes i j A
```

```
  assumes i < dim-row  $U_f$  and j < dim-col A and even i and dim-col  $U_f = \text{dim-row } A$ 
  shows ( $\sum k \in \{0..< \text{dim-row } A\}. U_f \$\$ (i,k) * A \$\$ (k,j)$ ) = ( $\sum k \in \{i,i+1\}. U_f \$\$ (i,k) * A \$\$ (k,j)$ )
```

```
proof-
```

```
  have ( $\sum k \in \{0..< 2^{(n+1)}\}. U_f \$\$ (i,k) * A \$\$ (k,j)$ ) =
```

$$\begin{aligned}
& (\sum k \in \{0..<i\}. U_f \$\$ (i,k) * A \$\$ (k,j)) + \\
& (\sum k \in \{i,i+1\}. U_f \$\$ (i,k) * A \$\$ (k,j)) + \\
& (\sum k \in \{(i+2)..< 2^{\lceil n+1 \rceil}\}. U_f \$\$ (i,k) * A \$\$ (k,j))
\end{aligned}$$

proof-

have $\{0..< 2^{\lceil n+1 \rceil}\} = \{0..< i\} \cup \{i..< 2^{\lceil n+1 \rceil}\}$
 $\wedge \{i..< 2^{\lceil n+1 \rceil}\} = \{i,i+1\} \cup \{(i+2)..< 2^{\lceil n+1 \rceil}\}$ using `assms(1-3)`

by `auto`

moreover have $\{0..< i\} \cap \{i,i+1\} = \{\}$
 $\wedge \{i,i+1\} \cap \{(i+2)..< 2^{\lceil n+1 \rceil}\} = \{\}$
 $\wedge \{0..< i\} \cap \{(i+2)..< 2^{\lceil n+1 \rceil}\} = \{\}$ using `assms by simp`

ultimately show ?thesis
using `sum.union-disjoint`
by (`metis (no-types, lifting) finite-Un finite-atLeastLessThan is-num-normalize(1)`
`ivl-disj-int-two(3)`)

qed

moreover have $(\sum k \in \{0..<i\}. U_f \$\$ (i,k) * A \$\$ (k,j)) = 0$

proof-

have $k \in \{0..<i\} \rightarrow (i \neq k \wedge \neg(i=k+1 \wedge \text{odd } i) \wedge \neg(i=k-1 \wedge \text{even } i \wedge k \geq 1))$ for k
using `assms by auto`

then have $k \in \{0..<i\} \rightarrow U_f \$\$ (i,k) = 0$ for k
using `assms(1) by auto`

then show ?thesis by `simp`

qed

moreover have $(\sum k \in \{(i+2)..< 2^{\lceil n+1 \rceil}\}. U_f \$\$ (i,k) * A \$\$ (k,j)) = 0$

proof-

have $k \in \{(i+2)..< 2^{\lceil n+1 \rceil}\} \rightarrow (i \neq k \wedge \neg(i=k+1 \wedge \text{odd } i) \wedge \neg(i=k-1 \wedge \text{even } i \wedge k \geq 1))$ for k by `auto`

then have $k \in \{(i+2)..< 2^{\lceil n+1 \rceil}\} \rightarrow U_f \$\$ (i,k) = 0$ for k
using `assms(1) by auto`

then show ?thesis by `simp`

qed

moreover have `dim-row A = 2^{\lceil n+1 \rceil}` using `assms(4) by simp`

ultimately show ?thesis by (`metis (no-types, lifting) add.left-neutral add.right-neutral`)

qed

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

proof-

have $(U_f * A) \$\$ (i,j) = (\sum k \in \{0..< \text{dim-row } A\}. (U_f \$\$ (i,k)) * (A \$\$ (k,j)))$
using `assms(1,2,4) index-matrix-prod by (simp add: atLeast0LessThan)`

then show ?thesis
using `assms U_f -mult-without-empty-summands-sum-even by simp`

qed

lemma (in jozsa) U_f -mult-without-empty-summands-sum-odd:

```

fixes i j A
assumes i < dim-row U_f and j < dim-col A and odd i and dim-col U_f =
dim-row A
shows ( $\sum_{k \in \{0..<\text{dim-row } A\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ ) = ( $\sum_{k \in \{i-1,i\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ )
proof-
have ( $\sum_{k \in \{0..< 2^{\lceil n+1 \rceil}\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ ) =
( $\sum_{k \in \{0..<i-1\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ ) +
( $\sum_{k \in \{i-1,i\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ ) +
( $\sum_{k \in \{i+1..< 2^{\lceil n+1 \rceil}\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ )
proof-
have {0..< 2^{\lceil n+1 \rceil}} = {0..<i-1}  $\cup$  {i-1..< 2^{\lceil n+1 \rceil}}
 $\wedge$  {i-1..< 2^{\lceil n+1 \rceil}} = {i-1,i}  $\cup$  {i+1..< 2^{\lceil n+1 \rceil}} using assms(1-3)
by auto
moreover have {0..<i-1}  $\cap$  {i-1,i} = {}
 $\wedge$  {i-1,i}  $\cap$  {i+1..< 2^{\lceil n+1 \rceil}} = {}
 $\wedge$  {0..<i-1}  $\cap$  {i+1..< 2^{\lceil n+1 \rceil}} = {} using assms by simp
ultimately show ?thesis
using sum.union-disjoint
by(metis (no-types, lifting) finite-Un finite-atLeastLessThan is-num-normalize(1)
ivl-disj-int-two(3))
qed
moreover have ( $\sum_{k \in \{0..<i-1\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ ) = 0
proof-
have k ∈ {0..<i-1}  $\longrightarrow$  (i ≠ k  $\wedge$   $\neg$ (i = k + 1  $\wedge$  odd i)  $\wedge$   $\neg$ (i = k - 1  $\wedge$  even i  $\wedge$ 
k ≥ 1)) for k by auto
then have k ∈ {0..<i-1}  $\longrightarrow$  U_f \$\$ (i,k) = 0 for k
using assms(1) by auto
then show ?thesis by simp
qed
moreover have ( $\sum_{k \in \{i+1..< 2^{\lceil n+1 \rceil}\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ ) = 0
using assms(3) by auto
moreover have dim-row A = 2^{\lceil n+1 \rceil} using assms(4) by simp
ultimately show ?thesis by(metis (no-types, lifting) add.left-neutral add.right-neutral)
qed

```

```

lemma (in jozsa) U_f-mult-without-empty-summands-odd:
fixes i j A
assumes i < dim-row U_f and j < dim-col A and odd i and dim-col U_f =
dim-row A
shows (U_f * A) \$\$ (i,j) = ( $\sum_{k \in \{i-1,i\}} U_f \$\$ (i,k) * A \$\$ (k,j)$ )
proof-
have (U_f * A) \$\$ (i,j) = ( $\sum_{k \in \{0..<\text{dim-row } A\}} (U_f \$\$ (i,k)) * (A \$\$ (k,j))$ )
using assms(1,2,4) index-matrix-prod by (simp add: atLeast0LessThan)
then show ?thesis
using assms U_f-mult-without-empty-summands-sum-odd by auto
qed

```

U_f is a gate.

```

lemma (in jozsa) transpose-of-jozsa-transform:
  shows  $(U_f)^t = U_f$ 
proof
  show dim-row  $(U_f^t) = \text{dim-row } U_f$  by simp
next
  show dim-col  $(U_f^t) = \text{dim-col } U_f$  by simp
next
  fix i j:: nat
  assume a0:  $i < \text{dim-row } U_f$  and a1:  $j < \text{dim-col } U_f$ 
  then show  $U_f^t \$\$ (i, j) = U_f \$\$ (i, j)$ 
  proof (induct rule: disj-four-cases)
    show  $i=j \vee (i=j+1 \wedge \text{odd } i) \vee (i=j-1 \wedge \text{even } i \wedge j \geq 1) \vee (i \neq j \wedge \neg(i=j+1 \wedge \text{odd } i) \wedge \neg(i=j-1 \wedge \text{even } i \wedge j \geq 1))$ 
      by linarith
  next
    assume i = j
    then show  $U_f^t \$\$ (i,j) = U_f \$\$ (i,j)$  using a0 by simp
  next
    assume  $(i=j+1 \wedge \text{odd } i)$ 
    then show  $U_f^t \$\$ (i,j) = U_f \$\$ (i,j)$  using transpose-mat-def a0 a1 by auto
  next
    assume a2:  $(i=j-1 \wedge \text{even } i \wedge j \geq 1)$ 
    then have  $U_f \$\$ (i,j) = f(i \text{ div } 2)$ 
    using a0 a1 jozsa-transform-coeff by auto
    moreover have  $U_f \$\$ (j,i) = f(i \text{ div } 2)$ 
    using a0 a1 a2 jozsa-transform-coeff
    by (metis add-diff-assoc2 diff-add-inverse2 even-plus-one-iff even-succ-div-two
        jozsa-transform-dim)
    ultimately show ?thesis
    using transpose-mat-def a0 a1 by simp
  next
    assume a2:  $(i \neq j \wedge \neg(i=j+1 \wedge \text{odd } i) \wedge \neg(i=j-1 \wedge \text{even } i \wedge j \geq 1))$ 
    then have  $(j \neq i \wedge \neg(j=i+1 \wedge \text{odd } j) \wedge \neg(j=i-1 \wedge \text{even } j \wedge i \geq 1))$ 
    by (metis le-imp-diff-is-add diff-add-inverse even-plus-one-iff le-add1)
    then have  $U_f \$\$ (j,i) = 0$ 
    using jozsa-transform-coeff-is-zero a0 a1 by auto
    moreover have  $U_f \$\$ (i,j) = 0$ 
    using jozsa-transform-coeff-is-zero a0 a1 a2 by auto
    ultimately show  $U_f^t \$\$ (i,j) = U_f \$\$ (i,j)$ 
    using transpose-mat-def a0 a1 by simp
qed
qed

lemma (in jozsa) adjoint-of-jozsa-transform:
  shows  $(U_f)^\dagger = U_f$ 
proof
  show dim-row  $(U_f^\dagger) = \text{dim-row } U_f$  by simp
next
  show dim-col  $(U_f^\dagger) = \text{dim-col } U_f$  by simp

```

```

next
  fix  $i j :: \text{nat}$ 
  assume  $a0: i < \text{dim-row } U_f \text{ and } a1: j < \text{dim-col } U_f$ 
  then show  $U_f^\dagger \$\$ (i,j) = U_f \$\$ (i,j)$ 
  proof (induct rule: disj-four-cases)
    show  $i=j \vee (i=j+1 \wedge \text{odd } i) \vee (i=j-1 \wedge \text{even } i \wedge j \geq 1) \vee (i \neq j \wedge \neg(i=j+1 \wedge \text{odd } i) \wedge \neg(i=j-1 \wedge \text{even } i \wedge j \geq 1))$ 
    by linarith
next
  assume  $i=j$ 
  then show  $U_f^\dagger \$\$ (i,j) = U_f \$\$ (i,j)$  using  $a0 \text{ dagger-def}$  by  $\text{simp}$ 
next
  assume  $(i=j+1 \wedge \text{odd } i)$ 
  then show  $U_f^\dagger \$\$ (i,j) = U_f \$\$ (i,j)$  using  $a0 \text{ dagger-def}$  by  $\text{auto}$ 
next
  assume  $a2:(i=j-1 \wedge \text{even } i \wedge j \geq 1)$ 
  then have  $U_f \$\$ (i,j) = f (i \text{ div } 2)$ 
  using  $a0 \text{ a1 jozsa-transform-coeff}$  by  $\text{auto}$ 
  moreover have  $U_f^\dagger \$\$ (j,i) = f (i \text{ div } 2)$ 
  using  $a1 \text{ a2 jozsa-transform-coeff dagger-def}$  by  $\text{auto}$ 
  ultimately show  $U_f^\dagger \$\$ (i,j) = U_f \$\$ (i,j)$ 
  by (metis a0 a1 cnj transpose-is-dagger dim-row-of-dagger index-transpose-mat dagger-of transpose-is-cnj transpose-of-jozsa-transform)
next
  assume  $a2: (i \neq j \wedge \neg(i=j+1 \wedge \text{odd } i) \wedge \neg(i=j-1 \wedge \text{even } i \wedge j \geq 1))$ 
  then have  $f0:(i \neq j \wedge \neg(j=i+1 \wedge \text{odd } j) \wedge \neg(j=i-1 \wedge \text{even } j \wedge i \geq 1))$ 
  by (metis le-imp-diff-is-add diff-add-inverse even-plus-one-iff le-add1)
  then have  $U_f \$\$ (j,i) = 0 \text{ and } \text{cnj } 0 = 0$ 
  using  $\text{jozsa-transform-coeff-is-zero a0 a1 a2}$  by  $\text{auto}$ 
  then have  $U_f^\dagger \$\$ (i,j) = 0$ 
  using  $a0 \text{ a1 dagger-def}$  by  $\text{simp}$ 
  then show  $U_f^\dagger \$\$ (i,j) = U_f \$\$ (i,j)$ 
  using  $a0 \text{ a1 a2 jozsa-transform-coeff-is-zero}$  by  $\text{auto}$ 
qed
qed

lemma (in jozsa) jozsa-transform-is-unitary-index-even:
  fixes  $i j :: \text{nat}$ 
  assumes  $i < \text{dim-row } U_f \text{ and } j < \text{dim-col } U_f \text{ and } \text{even } i$ 
  shows  $(U_f * U_f) \$\$ (i,j) = 1_m (\text{dim-col } U_f) \$\$ (i,j)$ 
proof-
  have  $(U_f * U_f) \$\$ (i,j) = (\sum k \in \{i,i+1\}. U_f \$\$ (i,k) * U_f \$\$ (k,j))$ 
  using  $U_f\text{-mult-without-empty-summands-even}[of i j U_f]$  assms by  $\text{simp}$ 
  moreover have  $U_f \$\$ (i,i) * U_f \$\$ (i,j) = (1 - f(i \text{ div } 2)) * U_f \$\$ (i,j)$ 
  using  $\text{assms}(1,3)$  by  $\text{simp}$ 
  moreover have  $f0: U_f \$\$ (i,i+1) * U_f \$\$ (i+1,j) = f(i \text{ div } 2) * U_f \$\$ (i+1,j)$ 
  by (metis One-nat-def Suc-leI add.right-neutral add-Suc-right assms(1) assms(3)
diff-add-inverse2
even-add even-mult-iff jozsa-transform-coeff(3) jozsa-transform-dim le-add2 le-eq-less-or-eq)

```

`odd-one`
`one-add-one power.simps(2))`
ultimately have $f1: (U_f * U_f) \$(i,j) = (1-f(i \text{ div } 2)) * U_f \$(i,j) + f(i \text{ div } 2) * U_f \$(i+1,j)$ **by auto**
thus $?thesis$
proof (*induct rule: disj-four-cases*)
show $j=i \vee (j=i+1 \wedge \text{odd } j) \vee (j=i-1 \wedge \text{even } j \wedge i \geq 1) \vee (j \neq i \wedge \neg(j=i+1 \wedge \text{odd } j) \wedge \neg(j=i-1 \wedge \text{even } j \wedge i \geq 1))$
by *linarith*
next
assume $a0:j=i$
then have $U_f \$(i,j) = (1-f(i \text{ div } 2))$
using assms(1,2) a0 by simp
moreover have $U_f \$(i+1,j) = f(i \text{ div } 2)$
using assms(1,3) a0 by auto
ultimately have $(U_f * U_f) \$(i,j) = (1-f(i \text{ div } 2)) * (1-f(i \text{ div } 2)) + f(i \text{ div } 2) * f(i \text{ div } 2)$
using f1 by simp
moreover have $(1-f(i \text{ div } 2)) * (1-f(i \text{ div } 2)) + f(i \text{ div } 2) * f(i \text{ div } 2) = 1$
using f-values assms(1)
by (*metis (no-types, lifting) Nat.minus-nat.diff-0 diff-add-0 diff-add-inverse jozsa-transform-dim(1)*
less-power-add-imp-div-less mem-Collect-eq mult-eq-if one-power2 power2-eq-square power-one-right)
ultimately show $(U_f * U_f) \$(i,j) = 1_m (\text{dim-col } U_f) \(i,j) **by** (*metis assms(2) a0 index-one-mat(1) of-nat-1*)
next
assume $a0: (j=i+1 \wedge \text{odd } j)$
then have $U_f \$(i,j) = f(i \text{ div } 2)$
using assms(1,2) a0 by simp
moreover have $U_f \$(i+1,j) = (1-f(i \text{ div } 2))$
using assms(2,3) a0 by simp
ultimately have $(U_f * U_f) \$(i,j) = (1-f(i \text{ div } 2)) * f(i \text{ div } 2) + f(i \text{ div } 2) * (1-f(i \text{ div } 2))$
using f0 f1 assms by simp
then show $(U_f * U_f) \$(i,j) = 1_m (\text{dim-col } U_f) \(i,j)
using assms(1,2) a0 by auto
next
assume $(j=i-1 \wedge \text{even } j \wedge i \geq 1)$
then show $(U_f * U_f) \$(i,j) = 1_m (\text{dim-col } U_f) \(i,j)
using assms(3) dvd-diffD1 odd-one by blast
next
assume $a0:(j \neq i \wedge \neg(j=i+1 \wedge \text{odd } j) \wedge \neg(j=i-1 \wedge \text{even } j \wedge i \geq 1))$
then have $U_f \$(i,j) = 0$
using assms(1,2) by (*metis index-transpose-mat(1) jozsa-transform-coeff-is-zero jozsa-transform-dim transpose-of-jozsa-transform*)
moreover have $U_f \$(i+1,j) = 0$
using assms a0 by auto
ultimately have $(U_f * U_f) \$(i,j) = (1-f(i \text{ div } 2)) * 0 + f(i \text{ div } 2) * 0$

```

    by (simp add: f1)
  then show (U_f * U_f) $$ (i,j) = 1_m (dim-col U_f) $$ (i,j)
    using a0 assms(1,2) by(metis add.left-neutral index-one-mat(1) jozsa-transform-dim
mult-0-right of-nat-0)
    qed
qed

lemma (in jozsa) jozsa-transform-is-unitary-index-odd:
fixes i j:: nat
assumes i < dim-row U_f and j < dim-col U_f and odd i
shows (U_f * U_f) $$ (i,j) = 1_m (dim-col U_f) $$ (i,j)
proof-
have f0: i ≥ 1
  using linorder-not-less assms(3) by auto
have (U_f * U_f) $$ (i,j) = (∑ k ∈ {i-1,i}. U_f $$ (i,k) * U_f $$ (k,j))
  using U_f-mult-without-empty-summands-odd[of i j U_f] assms by simp
moreover have (∑ k ∈ {i-1,i}. U_f $$ (i,k) * U_f $$ (k,j))
  = U_f $$ (i,i-1) * U_f $$ (i-1,j) + U_f $$ (i,i) * U_f $$ (i,j)
  using f0 by simp
moreover have U_f $$ (i,i) * U_f $$ (i,j) = (1-f(i div 2)) * U_f $$ (i,j)
  using assms(1,2) by simp
moreover have f1: U_f $$ (i,i-1) * U_f $$ (i-1,j) = f(i div 2) * U_f $$ (i-1,j)

  using assms(1) assms(3) by simp
ultimately have f2: (U_f * U_f) $$ (i,j) = f(i div 2) * U_f $$ (i-1,j) + (1-f(i
div 2)) * U_f $$ (i,j) by simp
then show ?thesis
proof (induct rule: disj-four-cases)
show j=i ∨ (j=i+1 ∧ odd j) ∨ (j=i-1 ∧ even j ∧ i≥1) ∨ (j≠i ∧ ¬(j=i+1
∧ odd j) ∧ ¬(j=i-1 ∧ even j ∧ i≥1))
  by linarith
next
assume a0:j=i
then have U_f $$ (i,j) = (1-f(i div 2))
  using assms(1,2) by simp
moreover have U_f $$ (i-1,j) = f(i div 2)
  using a0 assms
  by (metis index transpose-mat(1) jozsa-transform-coeff(2) less-imp-diff-less
odd-two-times-div-two-nat
        odd-two-times-div-two-succ transpose-of-jozsa-transform)
ultimately have (U_f * U_f) $$ (i,j) = f(i div 2) * f(i div 2) + (1-f(i div
2)) * (1-f(i div 2))
  using f2 by simp
moreover have f(i div 2) * f(i div 2) + (1-f(i div 2)) * (1-f(i div 2)) = 1
  using f-values assms(1)
  by (metis (no-types, lifting) Nat.minus-nat.diff-0 diff-add-0 diff-add-inverse
jozsa-transform-dim(1)
        less-power-add-imp-div-less mem-Collect-eq mult-eq-if one-power2 power2-eq-square
power-one-right)

```

```

ultimately show ( $U_f * U_f$ )  $\$ \$ (i,j) = 1_m (\dim\text{-}\col U_f) \$ \$ (i,j)$  by(metis
assms(2) a0 index-one-mat(1) of-nat-1)
next
assume a0:( $j=i+1 \wedge \text{odd } j$ )
then show ( $U_f * U_f$ )  $\$ \$ (i,j) = 1_m (\dim\text{-}\col U_f) \$ \$ (i,j)$ 
using assms(3) dvd-diffD1 odd-one even-plus-one-iff by blast
next
assume a0:( $j=i-1 \wedge \text{even } j \wedge i \geq 1$ )
then have ( $U_f * U_f$ )  $\$ \$ (i,j) = f(i \text{ div } 2) * (1-f(i \text{ div } 2)) + (1-f(i \text{ div } 2))$ 
*  $f(i \text{ div } 2)$ 
using f0 f1 f2 assms
by (metis jozsa-transform-coeff(1) Groups.ab-semigroup-mult-class.mult.commute
even-succ-div-two f2
jozsa-transform-dim odd-two-times-div-two-nat odd-two-times-div-two-succ
of-nat-add of-nat-mult)
then show ( $U_f * U_f$ )  $\$ \$ (i,j) = 1_m (\dim\text{-}\col U_f) \$ \$ (i,j)$ 
using assms(1) a0 by auto
next
assume a0: $j \neq i \wedge \neg(j=i+1 \wedge \text{odd } j) \wedge \neg(j=i-1 \wedge \text{even } j \wedge i \geq 1)$ 
then have  $U_f \$ \$ (i,j) = 0$ 
by (metis assms(1,2) index-transpose-mat(1) jozsa-transform-coeff-is-zero
jозса-transform-dim transpose-of-jозса-transform)
moreover have  $U_f \$ \$ (i-1,j) = 0$ 
using assms a0 f0
by auto (smt (verit) One-nat-def Suc-n-not-le-n add-diff-inverse-nat assms(1)
assms(2) diff-Suc-less even-add
jозса-transform-coeff-is-zero jozsa-axioms less-imp-le less-le-trans less-one odd-one)
ultimately have ( $U_f * U_f$ )  $\$ \$ (i,j) = (1-f(i \text{ div } 2)) * 0 + f(i \text{ div } 2) * 0$ 
using f2 by simp
then show ( $U_f * U_f$ )  $\$ \$ (i,j) = 1_m (\dim\text{-}\col U_f) \$ \$ (i,j)$ 
using a0 assms by (metis add.left-neutral index-one-mat(1) jozsa-transform-dim
mult-0-right of-nat-0)
qed
qed

```

```

lemma (in jozsa) jozsa-transform-is-gate:
shows gate (n+1)  $U_f$ 
proof
show dim-row  $U_f = \mathcal{Z}^{\sim}(n+1)$  by simp
next
show square-mat  $U_f$  by simp
next
show unitary  $U_f$ 
proof-
have  $U_f * U_f = 1_m (\dim\text{-}\col U_f)$ 
proof
show dim-row ( $U_f * U_f$ ) = dim-row ( $1_m (\dim\text{-}\col U_f)$ ) by simp
show dim-col ( $U_f * U_f$ ) = dim-col ( $1_m (\dim\text{-}\col U_f)$ ) by simp
fix i j:: nat

```

```

assume  $i < \text{dim-row} (1_m (\text{dim-col } U_f))$  and  $j < \text{dim-col} (1_m (\text{dim-col } U_f))$ 
then have  $i < \text{dim-row } U_f$  and  $j < \text{dim-col } U_f$  by auto
then show  $(U_f * U_f) \$\$ (i,j) = 1_m (\text{dim-col } U_f) \$\$ (i,j)$ 
using jozsa-transform-is-unitary-index-odd jozsa-transform-is-unitary-index-even
by blast
qed
thus ?thesis by (simp add: adjoint-of-jozsa-transform unitary-def)
qed
qed

```

N-fold application of the tensor product

```

fun iter-tensor:: complex Matrix.mat  $\Rightarrow$  nat  $\Rightarrow$  complex Matrix.mat ( $\langle\cdot \otimes^\cdot\rangle$  75)
where
 $A \otimes^{(\text{Suc } 0)} = A$ 
 $| A \otimes^{(\text{Suc } k)} = A \otimes (A \otimes^k)$ 

lemma one-tensor-is-id [simp]:
fixes A
shows  $A \otimes^1 = A$ 
using one-mat-def by simp

lemma iter-tensor-suc:
fixes n
assumes  $n \geq 1$ 
shows  $A \otimes^{(\text{Suc } n)} = A \otimes (A \otimes^n)$ 
using assms by (metis Deutsch-Jozsa.iter-tensor.simps(2) One-nat-def Suc-le-D)

lemma dim-row-of-iter-tensor [simp]:
fixes A n
assumes  $n \geq 1$ 
shows  $\text{dim-row}(A \otimes^n) = (\text{dim-row } A) \hat{\wedge} n$ 
using assms
proof (rule nat-induct-at-least)
show  $\text{dim-row}(A \otimes^1) = (\text{dim-row } A) \hat{\wedge} 1$ 
using one-tensor-is-id by simp
fix n:: nat
assume  $n \geq 1$  and  $\text{dim-row}(A \otimes^n) = (\text{dim-row } A) \hat{\wedge} n$ 
then show  $\text{dim-row}(A \otimes^{\text{Suc } n}) = (\text{dim-row } A) \hat{\wedge} (\text{Suc } n)$ 
using iter-tensor-suc assms dim-row-tensor-mat by simp
qed

lemma dim-col-of-iter-tensor [simp]:
fixes A n
assumes  $n \geq 1$ 
shows  $\text{dim-col}(A \otimes^n) = (\text{dim-col } A) \hat{\wedge} n$ 
using assms
proof (rule nat-induct-at-least)
show  $\text{dim-col}(A \otimes^1) = (\text{dim-col } A) \hat{\wedge} 1$ 
using one-tensor-is-id by simp

```

```

fix n:: nat
assume n ≥ 1 and dim-col (A ⊗n) = (dim-col A) ^n
then show dim-col (A ⊗Suc n) = (dim-col A) ^Suc n
  using iter-tensor-suc assms dim-col-tensor-mat by simp
qed

lemma iter-tensor-values:
fixes A n i j
assumes n ≥ 1 and i < dim-row (A ⊗ (A ⊗n)) and j < dim-col (A ⊗ (A ⊗n))
shows (A ⊗(Suc n)) $$ (i,j) = (A ⊗ (A ⊗n)) $$ (i,j)
using assms by (metis One-nat-def le-0-eq not0-implies-Suc iter-tensor.simps(2))

lemma iter-tensor-mult-distr:
assumes n ≥ 1 and dim-col A = dim-row B and dim-col A > 0 and dim-col B > 0
shows (A ⊗(Suc n)) * (B ⊗(Suc n)) = (A * B) ⊗ ((A ⊗n) * (B ⊗n))
proof-
have (A ⊗(Suc n)) * (B ⊗(Suc n)) = (A ⊗ (A ⊗n)) * (B ⊗ (B ⊗n))
  using Suc-le-D assms(1) by fastforce
then show ?thesis
  using mult-distr-tensor[of A B (iter-tensor A n) (iter-tensor B n)] assms by
simp
qed

lemma index-tensor-mat-with-vec2-row-cond:
fixes A B:: complex Matrix.mat and i:: nat
assumes i < 2 * (dim-row B) and i ≥ dim-row B and dim-col B > 0
and dim-row A = 2 and dim-col A = 1
shows (A ⊗ B) $$ (i,0) = (A $$ (1,0)) * (B $$ (i - dim-row B, 0))
proof-
have (A ⊗ B) $$ (i,0) = A $$ (i div (dim-row B), 0) * B $$ (i mod (dim-row B), 0)
  using assms index-tensor-mat[of A dim-row A dim-col A B dim-row B dim-col B i 0] by simp
moreover have i div (dim-row B) = 1
  using assms(1,2,4) by simp
then have i mod (dim-row B) = i - (dim-row B)
  by (simp add: modulo-nat-def)
ultimately show (A ⊗ B) $$ (i,0) = (A $$ (1,0)) * (B $$ (i - dim-row B, 0))
  by (simp add: i div dim-row B = 1)
qed

lemma iter-tensor-of-gate-is-gate:
fixes A:: complex Matrix.mat and n m:: nat
assumes gate m A and n ≥ 1
shows gate (m*n) (A ⊗n)
using assms(2)
proof(rule nat-induct-at-least)

```

```

show gate (m * 1) (A  $\otimes^1$ ) using assms(1) by simp
fix n:: nat
assume n  $\geq 1$  and IH:gate (m * n) (A  $\otimes^n$ )
then have A  $\otimes^{(Suc\ n)}$  = A  $\otimes$  (A  $\otimes^n$ )
  by (simp add: iter-tensor-suc)
moreover have gate (m*n + m) (A  $\otimes^{(Suc\ n)}$ )
  using tensor-gate assms(1) by (simp add: IH add.commute calculation(1))
then show gate (m*(Suc n)) (A  $\otimes^{(Suc\ n)}$ )
  by (simp add: add.commute)
qed

lemma iter-tensor-of-state-is-state:
fixes A:: complex Matrix.mat and n m:: nat
assumes state m A and n  $\geq 1$ 
shows state (m*n) (A  $\otimes^n$ )
using assms(2)
proof(rule nat-induct-at-least)
show state (m * 1) (A  $\otimes^1$ )
  using one-tensor-is-id assms(1) by simp
fix n:: nat
assume n  $\geq 1$  and IH:state (m * n) (A  $\otimes^n$ )
then have A  $\otimes^{(Suc\ n)}$  = A  $\otimes$  (A  $\otimes^n$ )
  by (simp add: iter-tensor-suc)
moreover have state (m*n + m) (A  $\otimes^{(Suc\ n)}$ )
  using tensor-gate assms(1) by (simp add: IH add.commute calculation)
then show state (m*(Suc n)) (A  $\otimes^{(Suc\ n)}$ )
  by (simp add: add.commute)
qed

```

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

```

abbreviation  $\psi_{10}$ :: nat  $\Rightarrow$  complex Matrix.mat where
 $\psi_{10}\ n \equiv$  Matrix.mat ( $2^{\wedge}n$ ) 1 ( $\lambda(i,j).$   $1/(\sqrt{2})^{\wedge}n$ )

```

```

lemma  $\psi_{10}$ -values:
fixes i j n
assumes i < dim-row ( $\psi_{10}\ n$ ) and j < dim-col ( $\psi_{10}\ n$ )
shows ( $\psi_{10}\ n$ )  $\$$  (i,j) =  $1/(\sqrt{2})^{\wedge}n$ 
using assms case-prod-conv by simp

```

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

```

lemma H-on-ket-zero:
shows (H * |zero>) =  $\psi_{10}\ 1$ 
proof
fix i j:: nat
assume i < dim-row ( $\psi_{10}\ 1$ ) and j < dim-col ( $\psi_{10}\ 1$ )
then have f1: i  $\in \{0,1\} \wedge j = 0$  by (simp add: less-2-cases)
then show (H * |zero>)  $\$$  (i,j) = ( $\psi_{10}\ 1$ )  $\$$  (i,j)

```

```

    by (auto simp add: times-mat-def scalar-prod-def H-def ket-vec-def)
next
  show dim-row (H * |zero>) = dim-row (ψ₁₀ 1) by (simp add: H-def)
  show dim-col (H * |zero>) = dim-col (ψ₁₀ 1) using H-def
    by (simp add: ket-vec-def)
qed

lemma ψ₁₀-tensor:
  assumes n ≥ 1
  shows (ψ₁₀ 1) ⊗ (ψ₁₀ n) = (ψ₁₀ (Suc n))
proof
  have dim-row (ψ₁₀ 1) * dim-row (ψ₁₀ n) = 2^(Suc n) by simp
  then show dim-row ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) = dim-row (ψ₁₀ (Suc n)) by simp
  have dim-col (ψ₁₀ 1) * dim-col (ψ₁₀ n) = 1 by simp
  then show dim-col ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) = dim-col (ψ₁₀ (Suc n)) by simp
next
  fix i j:: nat
  assume a0: i < dim-row (ψ₁₀ (Suc n)) and a1: j < dim-col (ψ₁₀ (Suc n))
  then have f0: j = 0 and f1: i < 2^(Suc n) by auto
  then have f2:(ψ₁₀ (Suc n)) $$ (i,j) = 1/(sqrt 2)^(Suc n)
    using ψ₁₀-values[of i (Suc n) j] a0 a1 by simp
  show ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) $$ (i,j) = (ψ₁₀ (Suc n)) $$ (i,j)
  proof (rule disjE)
    show i < dim-row (ψ₁₀ n) ∨ i ≥ dim-row (ψ₁₀ n) by linarith
  next
    assume a2: i < dim-row (ψ₁₀ n)
    then have ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) $$ (i,j) = (ψ₁₀ 1) $$ (0,0) * (ψ₁₀ n) $$ (i,0)
      using index-tensor-mat f0 assms by simp
    also have ... = 1/sqrt(2) * 1/(sqrt(2) ^n)
      using ψ₁₀-values a2 assms by simp
    finally show ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) $$ (i,j) = (ψ₁₀ (Suc n)) $$ (i,j)
      using f2 divide-divide-eq-left power-Suc by simp
  next
    assume i ≥ dim-row (ψ₁₀ n)
    then have ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) $$ (i,0) = ((ψ₁₀ 1) $$ (1,0)) * ((ψ₁₀ n) $$ (i - dim-row (ψ₁₀ n), 0))
      using index-tensor-mat-with-vec2-row-cond[of i (ψ₁₀ 1) (ψ₁₀ n)] a0 a1 f0
      by (metis dim-col-mat(1) dim-row-mat(1) index-tensor-mat-with-vec2-row-cond
          power-Suc power-one-right)
    then have ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) $$ (i,0) = 1/sqrt(2) * 1/(sqrt 2) ^n
      using ψ₁₀-values[of i - dim-row (ψ₁₀ n) n j] a0 a1 by simp
    then show ((ψ₁₀ 1) ⊗ (ψ₁₀ n)) $$ (i,j) = (ψ₁₀ (Suc n)) $$ (i,j)
      using f0 f1 divide-divide-eq-left power-Suc by simp
  qed
qed

lemma ψ₁₀-tensor-is-state:
  assumes n ≥ 1
  shows state n ( |zero⟩ ⊗n)

```

using iter-tensor-of-state-is-state ket-zero-is-state assms **by** fastforce

lemma iter-tensor-of-H-is-gate:

assumes $n \geq 1$

shows gate $n (H \otimes^n)$

using iter-tensor-of-gate-is-gate H-is-gate assms **by** fastforce

lemma iter-tensor-of-H-on-zero-tensor:

assumes $n \geq 1$

shows $(H \otimes^n) * (|zero\rangle \otimes^n) = \psi_{10} n$

using assms

proof(rule nat-induct-at-least)

show $(H \otimes^1) * (|zero\rangle \otimes^1) = \psi_{10} 1$

using H-on-ket-zero **by** simp

next

fix $n :: nat$

assume $a0 : n \geq 1$ and IH: $(H \otimes^n) * (|zero\rangle \otimes^n) = \psi_{10} n$

then have $(H \otimes^{(Suc n)}) * (|zero\rangle \otimes^{(Suc n)}) = (H * |zero\rangle) \otimes ((H \otimes^n) * (|zero\rangle \otimes^n))$

using iter-tensor-mult-distr[of n H |zero] a0 ket-vec-def H-def **by**(simp add: H-def)

also have ... = $(H * |zero\rangle) \otimes (\psi_{10} n)$ using IH **by** simp

also have ... = $(\psi_{10} 1) \otimes (\psi_{10} n)$ using H-on-ket-zero **by** simp

also have ... = $(\psi_{10} (Suc n))$ using ψ_{10} -tensor a0 **by** simp

finally show $(H \otimes^{(Suc n)}) * (|zero\rangle \otimes^{(Suc n)}) = (\psi_{10} (Suc n))$ **by** simp

qed

lemma ψ_{10} -is-state:

assumes $n \geq 1$

shows state $n (\psi_{10} n)$

using iter-tensor-of-H-is-gate ψ_{10} -tensor-is-state assms gate-on-state-is-state iter-tensor-of-H-on-zero-tensor assms **by** metis

abbreviation $\psi_{11} :: complex Matrix.mat$ **where**

$\psi_{11} \equiv Matrix.mat 2 1 (\lambda(i,j). if i=0 then 1/sqrt(2) else -1/sqrt(2))$

lemma H-on-ket-one-is- ψ_{11} :

shows $(H * |one\rangle) = \psi_{11}$

proof

fix $i j :: nat$

assume $i < dim\text{-}row \psi_{11}$ and $j < dim\text{-}col \psi_{11}$

then have $i \in \{0,1\} \wedge j = 0$ **by** (simp add: less-2-cases)

then show $(H * |one\rangle) \$\$ (i,j) = \psi_{11} \$\$ (i,j)$

by (auto simp add: times-mat-def scalar-prod-def H-def ket-vec-def)

next

show dim-row $(H * |one\rangle) = dim\text{-}row \psi_{11}$ **by** (simp add: H-def)

next

show dim-col $(H * |one\rangle) = dim\text{-}col \psi_{11}$ **by** (simp add: H-def ket-vec-def)

qed

```

abbreviation  $\psi_1$ :: nat  $\Rightarrow$  complex Matrix.mat where
 $\psi_1\ n \equiv \text{Matrix.mat } (2^{\wedge}(n+1))\ 1\ (\lambda(i,j).\ \text{if even } i \text{ then } 1/(\sqrt{2})^{\wedge}(n+1) \text{ else } -1/(\sqrt{2})^{\wedge}(n+1))$ 

lemma  $\psi_1\text{-values-even}[\text{simp}]\!:$ 
  fixes  $i\ j\ n$ 
  assumes  $i < \text{dim-row } (\psi_1\ n)$  and  $j < \text{dim-col } (\psi_1\ n)$  and even  $i$ 
  shows  $(\psi_1\ n)\ \$\$(i,j) = 1/(\sqrt{2})^{\wedge}(n+1)$ 
  using assms case-prod-conv by simp

lemma  $\psi_1\text{-values-odd}[\text{simp}]\!:$ 
  fixes  $i\ j\ n$ 
  assumes  $i < \text{dim-row } (\psi_1\ n)$  and  $j < \text{dim-col } (\psi_1\ n)$  and odd  $i$ 
  shows  $(\psi_1\ n)\ \$\$(i,j) = -1/(\sqrt{2})^{\wedge}(n+1)$ 
  using assms case-prod-conv by simp

lemma  $\psi_{10}\text{-tensor-}\psi_{11}\text{-is-}\psi_1\!:$ 
  assumes  $n \geq 1$ 
  shows  $(\psi_{10}\ n) \otimes \psi_{11} = \psi_1\ n$ 
proof
  show  $\text{dim-col } ((\psi_{10}\ n) \otimes \psi_{11}) = \text{dim-col } (\psi_1\ n)$  by simp
next
  show  $\text{dim-row } ((\psi_{10}\ n) \otimes \psi_{11}) = \text{dim-row } (\psi_1\ n)$  by simp
next
  fix  $i\ j::\ \text{nat}$ 
  assume  $a0: i < \text{dim-row } (\psi_1\ n)$  and  $a1: j < \text{dim-col } (\psi_1\ n)$ 
  then have  $i < 2^{\wedge}(n+1)$  and  $j = 0$  by auto
  then have  $f0: ((\psi_{10}\ n) \otimes \psi_{11}) \$\$(i,j) = 1/(\sqrt{2})^{\wedge}n * \psi_{11} \$\$(i \bmod 2, j)$ 
    using  $\psi_{10}\text{-values}[\text{of } i \bmod 2\ n\ j \bmod 1]$  a0 a1 by simp
  show  $((\psi_{10}\ n) \otimes \psi_{11}) \$\$(i,j) = (\psi_1\ n) \$\$(i,j)$ 
    using f0  $\psi_1\text{-values-even}$   $\psi_1\text{-values-odd}$  a0 a1 by auto
qed

lemma  $\psi_1\text{-is-state}\!:$ 
  assumes  $n \geq 1$ 
  shows state  $(n+1)$   $(\psi_1\ n)$ 
  using assms  $\psi_{10}\text{-tensor-}\psi_{11}\text{-is-}\psi_1$   $\psi_{10}\text{-is-state}$  H-on-ket-one-is-state H-on-ket-one-is- $\psi_{11}$  tensor-state by metis

abbreviation (in jozsa)  $\psi_2$ :: complex Matrix.mat where
 $\psi_2 \equiv \text{Matrix.mat } (2^{\wedge}(n+1))\ 1\ (\lambda(i,j).\ \text{if even } i \text{ then } (-1)^{\wedge}f(i \bmod 2)/(\sqrt{2})^{\wedge}(n+1)$ 
 $\text{else } (-1)^{\wedge}(f(i \bmod 2)+1)/(\sqrt{2})^{\wedge}(n+1))$ 

lemma (in jozsa)  $\psi_2\text{-values-even}[\text{simp}]\!:$ 
  fixes  $i\ j$ 
  assumes  $i < \text{dim-row } \psi_2$  and  $j < \text{dim-col } \psi_2$  and even  $i$ 
  shows  $\psi_2 \$\$(i,j) = (-1)^{\wedge}f(i \bmod 2)/(\sqrt{2})^{\wedge}(n+1)$ 
```

using assms case-prod-conv by simp

lemma (in jozsa) ψ_2 -values-odd [simp]:
fixes $i j$
assumes $i < \text{dim-row } \psi_2$ and $j < \text{dim-col } \psi_2$ and odd i
shows $\psi_2 \$\$ (i,j) = (-1)^{\lceil f(i \text{ div } 2) + 1 \rceil} / (\sqrt{2})^{\lceil n+1 \rceil}$
using assms case-prod-conv by simp

lemma (in jozsa) ψ_2 -values-odd-hidden [simp]:
assumes $2*k+1 < \text{dim-row } \psi_2$ and $j < \text{dim-col } \psi_2$
shows $\psi_2 \$\$ (2*k+1,j) = ((-1)^{\lceil f((2*k+1) \text{ div } 2) + 1 \rceil}) / (\sqrt{2})^{\lceil n+1 \rceil}$
using assms by simp

lemma (in jozsa) snd-rep-of- ψ_2 :
assumes $i < \text{dim-row } \psi_2$
shows $((1-f(i \text{ div } 2)) + -f(i \text{ div } 2)) * 1 / (\sqrt{2})^{\lceil n+1 \rceil} = (-1)^{\lceil f(i \text{ div } 2) \rceil} / (\sqrt{2})^{\lceil n+1 \rceil}$
and $((-1-f(i \text{ div } 2)) + f(i \text{ div } 2)) * 1 / (\sqrt{2})^{\lceil n+1 \rceil} = (-1)^{\lceil f(i \text{ div } 2) \rceil + 1} / (\sqrt{2})^{\lceil n+1 \rceil}$

proof –

have $i \text{ div } 2 \in \{i. i < 2^{\lceil n \rceil}\}$

using assms by auto

then have real (Suc 0 - f (i div 2)) - real (f (i div 2)) = (-1)^{\lceil f (i div 2) \rceil}

using assms f-values by auto

thus $((1-f(i \text{ div } 2)) + -f(i \text{ div } 2)) * 1 / (\sqrt{2})^{\lceil n+1 \rceil} = (-1)^{\lceil f(i \text{ div } 2) \rceil} / (\sqrt{2})^{\lceil n+1 \rceil}$
by auto

next

have $i \text{ div } 2 \in \{i. i < 2^{\lceil n \rceil}\}$

using assms by simp

then have $(\text{real } (f (i \text{ div } 2)) - \text{real } (\text{Suc 0} - f (i \text{ div } 2))) / (\sqrt{2}^{\lceil n+1 \rceil}) =$
 $-((-1)^{\lceil f (i \text{ div } 2) \rceil} / (\sqrt{2}^{\lceil n+1 \rceil}))$

using assms f-values by fastforce

then show $((-1-f(i \text{ div } 2)) + f(i \text{ div } 2)) * 1 / (\sqrt{2})^{\lceil n+1 \rceil} = (-1)^{\lceil f(i \text{ div } 2) \rceil + 1} / (\sqrt{2})^{\lceil n+1 \rceil}$
by simp

qed

lemma (in jozsa) jozsa-transform-times- ψ_1 -is- ψ_2 :

shows $U_f * (\psi_1 n) = \psi_2$

proof

show dim-row ($U_f * (\psi_1 n)$) = dim-row ψ_2 by simp

next

show dim-col ($U_f * (\psi_1 n)$) = dim-col ψ_2 by simp

next

fix $i j :: nat$

assume a0: $i < \text{dim-row } \psi_2$ and a1: $j < \text{dim-col } \psi_2$

then have f0: $i \in \{0..2^{\lceil n+1 \rceil}\} \wedge j=0$ by simp

then have f1: $i < \text{dim-row } U_f \wedge j < \text{dim-col } U_f$ using a0 by simp

have f2: $i < \text{dim-row } (\psi_1 n) \wedge j < \text{dim-col } (\psi_1 n)$ using a0 a1 by simp

show $(U_f * (\psi_1 n)) \$\$ (i,j) = \psi_2 \$\$ (i,j)$

```

proof (rule disjE)
  show even  $i \vee \text{odd } i$  by auto
next
  assume  $a2: \text{even } i$ 
  then have  $(U_f * (\psi_1 n)) \$\$ (i,j) = (\sum k \in \{i,i+1\}. U_f \$\$ (i,k) * (\psi_1 n) \$\$ (k,j))$ 
    using  $f1 f2 U_f\text{-mult-without-empty-summands-even}[of i j (\psi_1 n)]$  by simp
    moreover have  $U_f \$\$ (i,i) * (\psi_1 n) \$\$ (i,j) = (1-f(i \text{ div } 2))* 1/(\sqrt{2})^{\sim(n+1)}$ 
      using  $f0 f1 a2$  by simp
      moreover have  $U_f \$\$ (i,i+1) * (\psi_1 n) \$\$ (i+1,j) = (-f(i \text{ div } 2))* 1/(\sqrt{2})^{\sim(n+1)}$ 
        using  $f0 f1 a2$  by auto
        ultimately have  $(U_f * (\psi_1 n)) \$\$ (i,j) = (1-f(i \text{ div } 2))* 1/(\sqrt{2})^{\sim(n+1)}$ 
       $+ (-f(i \text{ div } 2))* 1/(\sqrt{2})^{\sim(n+1)}$  by simp
      also have ...  $= ((1-f(i \text{ div } 2))+-f(i \text{ div } 2))* 1/(\sqrt{2})^{\sim(n+1)}$ 
        using add-divide-distrib
        by (metis (no-types, opaque-lifting) mult.right-neutral of-int-add of-int-of-nat-eq)
      also have ...  $= \psi_2 \$\$ (i,j)$ 
        using  $a0 a1 a2 \text{snd-rep-of-}\psi_2$  by simp
      finally show  $(U_f * (\psi_1 n)) \$\$ (i,j) = \psi_2 \$\$ (i,j)$  by simp
next
  assume  $a2: \text{odd } i$ 
  then have  $f6: i \geq 1$ 
  using linorder-not-less by auto
  have  $(U_f * (\psi_1 n)) \$\$ (i,j) = (\sum k \in \{i-1,i\}. U_f \$\$ (i,k) * (\psi_1 n) \$\$ (k,j))$ 
    using  $f1 f2 a2 U_f\text{-mult-without-empty-summands-odd}[of i j (\psi_1 n)]$ 
    by (metis dim-row-mat(1) jozsa-transform-dim(2))
    moreover have  $(\sum k \in \{i-1,i\}. U_f \$\$ (i,k) * (\psi_1 n) \$\$ (k,j))$ 
       $= U_f \$\$ (i,i-1) * (\psi_1 n) \$\$ (i-1,j) + U_f \$\$ (i,i) * (\psi_1 n) \$\$ (i,j)$ 
      using  $a2 f6$  by simp
      moreover have  $U_f \$\$ (i,i) * (\psi_1 n) \$\$ (i,j) = (1-f(i \text{ div } 2))* -1/(\sqrt{2})^{\sim(n+1)}$ 
       $+ (f(i \text{ div } 2))* 1/(\sqrt{2})^{\sim(n+1)}$ 
        using  $f1 f2 a2$  by simp
        moreover have  $U_f \$\$ (i,i-1) * (\psi_1 n) \$\$ (i-1,j) = f(i \text{ div } 2)* 1/(\sqrt{2})^{\sim(n+1)}$ 
        using  $a0 a1 a2$  by simp
        ultimately have  $(U_f * (\psi_1 n)) \$\$ (i,j) = (1-f(i \text{ div } 2))* -1/(\sqrt{2})^{\sim(n+1)}$ 
       $+ (f(i \text{ div } 2))* 1/(\sqrt{2})^{\sim(n+1)}$ 
        using of-real-add by simp
        also have ...  $= ((-1-f(i \text{ div } 2)) + (f(i \text{ div } 2)))* 1/(\sqrt{2})^{\sim(n+1)}$ 
        by (metis (no-types, opaque-lifting) mult.right-neutral add-divide-distrib mult-minus1-right
          of-int-add of-int-of-nat-eq)
      also have ...  $= (-1)^{\sim(f(i \text{ div } 2)+1)} / (\sqrt{2})^{\sim(n+1)}$ 
        using  $a0 a1 a2 \text{snd-rep-of-}\psi_2$  by simp
      finally show  $(U_f * (\psi_1 n)) \$\$ (i,j) = \psi_2 \$\$ (i,j)$ 
        using  $a0 a1 a2$  by simp

```

```

qed
qed

lemma (in jozsa) ψ₂-is-state:
  shows state (n+1) ψ₂
  using jozsa-transform-times-ψ₁-is-ψ₂ jozsa-transform-is-gate ψ₁-is-state dim gate-on-state-is-state
by fastforce

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

abbreviation iter-tensor-of-H-rep:: nat  $\Rightarrow$  complex Matrix.mat ( $\langle H \hat{\otimes} \cdot \rangle$ ) where
iter-tensor-of-H-rep n  $\equiv$  Matrix.mat ( $2^{\hat{n}}$ ) ( $2^{\hat{n}}$ ) ( $\lambda(i,j).(-1)^{(i \cdot_n j)} / (\sqrt{2})^{\hat{n}}$ )

lemma tensor-of-H-values [simp]:
  fixes n i j:: nat
  assumes i < dim-row ( $H \hat{\otimes} n$ ) and j < dim-col ( $H \hat{\otimes} n$ )
  shows ( $H \hat{\otimes} n$ ) $(i,j) = (-1)^{(i \cdot_n j)} / (\sqrt{2})^{\hat{n}}"
  using assms by simp

lemma dim-row-of-iter-tensor-of-H [simp]:
  assumes n  $\geq$  1
  shows 1 < dim-row ( $H \hat{\otimes} n$ )
  using assms by (metis One-nat-def Suc-1 dim-row-mat(1) le-trans lessI linorder-not-less
one-less-power)

lemma iter-tensor-of-H-fst-pos:
  fixes n i j:: nat
  assumes i <  $2^{\hat{n}}$  ∨ j <  $2^{\hat{n}}$  and i <  $2^{\hat{(n+1)}}$  ∧ j <  $2^{\hat{(n+1)}}$ 
  shows ( $H \hat{\otimes} (Suc n)$ ) $(i,j) = 1/\sqrt{2} * ((H \hat{\otimes} n) $(i mod  $2^{\hat{n}}$ , j mod
 $2^{\hat{n}}))$ 
proof-
  have ( $H \hat{\otimes} (Suc n)$ ) $(i,j) = (-1)^{(bip i (Suc n) j)} / (\sqrt{2})^{\hat{(Suc n)}}
  using assms by simp
  moreover have bip i (Suc n) j = bip (i mod  $2^{\hat{n}}$ ) n (j mod  $2^{\hat{n}}$ )
  using bitwise-inner-prod-fst-el-0 assms(1) by simp
  ultimately show ?thesis
  using bitwise-inner-prod-def by simp
qed

lemma iter-tensor-of-H-fst-neg:
  fixes n i j:: nat
  assumes i  $\geq$   $2^{\hat{n}}$  ∧ j  $\geq$   $2^{\hat{n}}$  and i <  $2^{\hat{(n+1)}}$  ∧ j <  $2^{\hat{(n+1)}}$ 
  shows ( $H \hat{\otimes} (Suc n)$ ) $(i,j) = -1/\sqrt{2} * ((H \hat{\otimes} n) $(i mod  $2^{\hat{n}}$ , j mod
 $2^{\hat{n}}))$ 
proof-
  have ( $H \hat{\otimes} (Suc n)$ ) $(i,j) = (-1)^{(bip i (n+1) j)} / (\sqrt{2})^{\hat{(n+1)}}
  using assms(2) by simp
  moreover have bip i (n+1) j = 1 + bip (i mod  $2^{\hat{n}}$ ) n (j mod  $2^{\hat{n}}$ )
  using bitwise-inner-prod-fst-el-is-1 assms by simp
  ultimately show ?thesis by simp

```

qed

```

lemma H-tensor-iter-tensor-of-H:
  fixes n:: nat
  shows (H  $\otimes$  H $\hat{\otimes}$  n) = H $\hat{\otimes}$  (Suc n)
proof
  fix i j:: nat
  assume a0: i < dim-row (H $\hat{\otimes}$  (Suc n)) and a1: j < dim-col (H $\hat{\otimes}$  (Suc n))
  then have f0: i  $\in \{0..<2^{\lceil n+1 \rceil}\} \wedge j \in \{0..<2^{\lceil n+1 \rceil}\}$  by simp
  then have f1: (H  $\otimes$  H $\hat{\otimes}$  n) $$ (i,j) = H $$ (i div (dim-row (H $\hat{\otimes}$  n)),j div
  (dim-col (H $\hat{\otimes}$  n)))
    * (H $\hat{\otimes}$  n) $$ (i \bmod (\dim-row (H $\hat{\otimes}$  n)),j \bmod
  (dim-col (H $\hat{\otimes}$  n)))
  by (simp add: H-without-scalar-prod)
  show (H  $\otimes$  H $\hat{\otimes}$  n) $$ (i,j) = (H\hat{\otimes} (Suc n)) $$ (i,j)
  proof (rule disjE)
    show (i < 2 $\lceil n \rceil \vee j < 2^{\lceil n \rceil}) \vee \neg(i < 2^{\lceil n \rceil} \vee j < 2^{\lceil n \rceil})$  by auto
  next
    assume a2: (i < 2 $\lceil n \rceil \vee j < 2^{\lceil n \rceil})$ 
    then have (H $\hat{\otimes}$  (Suc n)) $$ (i,j) = 1/\sqrt{2} * ((H\hat{\otimes} n) $$ (i mod 2 $\lceil n \rceil, j
  \bmod 2^{\lceil n \rceil}))$ 
    using a0 a1 f0 iter-tensor-of-H-fst-pos by (metis (mono-tags, lifting) atLeast-
  LessThan-iff)
    moreover have H $$ (i div (\dim-row (H $\hat{\otimes}$  n)),j div (\dim-col (H $\hat{\otimes}$  n))) = 1/\sqrt{2}
    using a0 a1 f0 H-without-scalar-prod H-values a2
    by (metis (no-types, lifting) dim-col-mat(1) dim-row-mat(1) div-less le-eq-less-or-eq
      le-numeral-extra(2) less-power-add-imp-div-less plus-1-eq-Suc power-one-right)

    ultimately show (H  $\otimes$  H $\hat{\otimes}$  n) $$ (i,j) = (H\hat{\otimes} (Suc n)) $$ (i,j)
    using f1 by simp
  next
    assume a2: \neg(i < 2 $\lceil n \rceil \vee j < 2^{\lceil n \rceil})$ 
    then have i  $\geq 2^{\lceil n \rceil} \wedge j \geq 2^{\lceil n \rceil}$  by simp
    then have f2: (H $\hat{\otimes}$  (Suc n)) $$ (i,j) = -1/\sqrt{2} * ((H\hat{\otimes} n) $$ (i mod 2 $\lceil n \rceil,
  j \bmod 2^{\lceil n \rceil}))$ 
    using a0 a1 f0 iter-tensor-of-H-fst-neg by simp
    have i div (\dim-row (H $\hat{\otimes}$  n)) = 1 and j div (\dim-row (H $\hat{\otimes}$  n)) = 1
    using a2 a0 a1 by auto
    then have H $$ (i div (\dim-row (H $\hat{\otimes}$  n)),j div (\dim-col (H $\hat{\otimes}$  n))) = -1/\sqrt{2}
    using a0 a1 f0 H-values-right-bottom[of i div (\dim-row (H $\hat{\otimes}$  n)) j div (\dim-col
  (H $\hat{\otimes}$  n))] a2
    by fastforce
    then show (H  $\otimes$  H $\hat{\otimes}$  n) $$ (i,j) = (H\hat{\otimes} (Suc n)) $$ (i,j)
    using f1 f2 by simp
  qed
  next

```

```

show dim-row ( $H \otimes H^\wedge_\otimes n$ ) = dim-row ( $H^\wedge_\otimes (\text{Suc } n)$ )
  by (simp add:  $H$ -without-scalar-prod)
next
  show dim-col ( $H \otimes H^\wedge_\otimes n$ ) = dim-col ( $H^\wedge_\otimes (\text{Suc } n)$ )
    by (simp add:  $H$ -without-scalar-prod)
qed

```

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

```

lemma one-tensor-of-H-is-H:
  shows ( $H^\wedge_\otimes 1$ ) =  $H$ 
proof(rule eq-matI)
  show dim-row ( $H^\wedge_\otimes 1$ ) = dim-row  $H$ 
    by (simp add:  $H$ -without-scalar-prod)
  show dim-col ( $H^\wedge_\otimes 1$ ) = dim-col  $H$ 
    by (simp add:  $H$ -without-scalar-prod)
next
  fix  $i j :: \text{nat}$ 
  assume  $a0:i < \text{dim-row } H$  and  $a1:j < \text{dim-col } H$ 
  then show ( $H^\wedge_\otimes 1$ ) $$ (i,j) = H $$ (i,j)
  proof-
    have ( $H^\wedge_\otimes 1$ ) $$ (0, 0) = 1/\sqrt{2}
    using bitwise-inner-prod-def bin-rep-def by simp
    moreover have ( $H^\wedge_\otimes 1$ ) $$ (0,1) = 1/\sqrt{2}
      using bitwise-inner-prod-def bin-rep-def by simp
    moreover have ( $H^\wedge_\otimes 1$ ) $$ (1,0) = 1/\sqrt{2}
      using bitwise-inner-prod-def bin-rep-def by simp
    moreover have ( $H^\wedge_\otimes 1$ ) $$ (1,1) = -1/\sqrt{2}
      using bitwise-inner-prod-def bin-rep-def by simp
    ultimately show ( $H^\wedge_\otimes 1$ ) $$ (i,j) = H $$ (i,j)
      using a0 a1  $H$ -values  $H$ -values-right-bottom
    by (metis (no-types, lifting)  $H$ -without-scalar-prod One-nat-def dim-col-mat(1)
      dim-row-mat(1)
      divide-minus-left less-2-cases)
  qed
qed

```

```

lemma iter-tensor-of-H-rep-is-correct:
  fixes  $n :: \text{nat}$ 
  assumes  $n \geq 1$ 
  shows ( $H \otimes^n$ ) =  $H^\wedge_\otimes n$ 
  using assms
proof(rule nat-induct-at-least)
  show ( $H \otimes^1$ ) =  $H^\wedge_\otimes 1$ 
    using one-tensor-is-id one-tensor-of-H-is-H by simp
next
  fix  $n :: \text{nat}$ 
  assume  $a0:n \geq 1$  and IH:( $H \otimes^n$ ) =  $H^\wedge_\otimes n$ 

```

```

then have  $(H \otimes^{(Suc n)}) = H \otimes (H \otimes^n)$ 
  using iter-tensor-suc Nat.Suc-eq-plus1 by metis
also have ... =  $H \otimes (H^\wedge \otimes n)$ 
  using IH by simp
also have ... =  $H^\wedge \otimes (Suc n)$ 
  using a0 H-tensor-iter-tensor-of-H by simp
finally show  $(H \otimes^{(Suc n)}) = H^\wedge \otimes (Suc n)$ 
  by simp
qed

```

$HId^\wedge \otimes 1$ is the result of taking the tensor product of the nth tensor of H and $Id 1$

abbreviation tensor-of-H-tensor- Id :: nat \Rightarrow complex Matrix.mat ($\langle HId^\wedge \otimes \cdot \rangle$) **where**
 $tensor-of-H-tensor-Id n \equiv Matrix.mat (2^{\lceil n+1 \rceil}) (2^{\lceil n+1 \rceil}) (\lambda(i,j).$
 $if (i \bmod 2 = j \bmod 2) then (-1)^{\lceil (i \bmod 2) \cdot n (j \bmod 2) \rceil} / (sqrt 2)^{\lceil n \rceil}$ else 0)

lemma mod-2-is-both-even-or-odd:
 $((even i \wedge even j) \vee (odd i \wedge odd j)) \longleftrightarrow (i \bmod 2 = j \bmod 2)$
by (metis even-iff-mod-2-eq-zero odd-iff-mod-2-eq-one)

lemma HId -values [simp]:
assumes $n \geq 1$ **and** $i < dim\text{-}row (HId^\wedge \otimes n)$ **and** $j < dim\text{-}col (HId^\wedge \otimes n)$
shows $even i \wedge even j \longrightarrow (HId^\wedge \otimes n) \$\$ (i,j) = (-1)^{\lceil (i \bmod 2) \cdot n (j \bmod 2) \rceil} / (sqrt 2)^{\lceil n \rceil}$
and $odd i \wedge odd j \longrightarrow (HId^\wedge \otimes n) \$\$ (i,j) = (-1)^{\lceil (i \bmod 2) \cdot n (j \bmod 2) \rceil} / (sqrt 2)^{\lceil n \rceil}$
and $(i \bmod 2 = j \bmod 2) \longrightarrow (HId^\wedge \otimes n) \$\$ (i,j) = (-1)^{\lceil (i \bmod 2) \cdot n (j \bmod 2) \rceil} / (sqrt 2)^{\lceil n \rceil}$
and $\neg(i \bmod 2 = j \bmod 2) \longrightarrow (HId^\wedge \otimes n) \$\$ (i,j) = 0$
using assms mod-2-is-both-even-or-odd **by** auto

lemma iter-tensor-of-H-tensor- Id -is- HId :
shows $(H^\wedge \otimes n) \otimes Id 1 = HId^\wedge \otimes n$
proof
show $dim\text{-}row ((H^\wedge \otimes n) \otimes Id 1) = dim\text{-}row (HId^\wedge \otimes n)$
by (simp add: Quantum.Id-def)
show $dim\text{-}col ((H^\wedge \otimes n) \otimes Id 1) = dim\text{-}col (HId^\wedge \otimes n)$
by (simp add: Quantum.Id-def)
next
fix $i j :: nat$
assume $a0: i < dim\text{-}row (HId^\wedge \otimes n)$ **and** $a1: j < dim\text{-}col (HId^\wedge \otimes n)$
then have $f0: i < (2^{\lceil n+1 \rceil}) \wedge j < (2^{\lceil n+1 \rceil})$ **by** simp
then have $i < dim\text{-}row (H^\wedge \otimes n) * dim\text{-}row (Id 1) \wedge j < dim\text{-}col (H^\wedge \otimes n) * dim\text{-}col (Id 1)$
using Id-def **by** simp
moreover have $dim\text{-}col (H^\wedge \otimes n) \geq 0 \wedge dim\text{-}col (Id 1) \geq 0$
using Id-def **by** simp
ultimately have $f1: ((H^\wedge \otimes n) \otimes (Id 1)) \$\$ (i,j)$
 $= (H^\wedge \otimes n) \$\$ (i \bmod 2 \cdot dim\text{-}row (Id 1), j \bmod 2 \cdot dim\text{-}col (Id 1)) * (H^\wedge \otimes n) \$\$ (i \bmod 2 \cdot dim\text{-}row (Id 1), j \bmod 2 \cdot dim\text{-}col (Id 1))$

```

(Id 1) $$ (i \bmod (\dim-row (Id 1)), j \bmod (\dim-col (Id 1)))
by (simp add: Quantum.Id-def)
show ((H $\widehat{\otimes}$  n)  $\widehat{\otimes}$  Id 1) $$ (i,j) = (HId $\widehat{\otimes}$  n) $$ (i,j)
proof (rule disjE)
  show (i mod 2 = j mod 2) ∨ ¬(i mod 2 = j mod 2) by simp
next
  assume a2:(i mod 2 = j mod 2)
  then have (Id 1) $$ (i \bmod (\dim-row (Id 1)), j \bmod (\dim-col (Id 1))) = 1
    by (simp add: Quantum.Id-def)
  moreover have (H $\widehat{\otimes}$  n) $$ (i \bmod (\dim-row (Id 1)), j \bmod (\dim-col (Id 1)))
    = (-1) $\widehat{\wedge}$ ((i \bmod (\dim-row (Id 1)))  $\cdot$  n (j \bmod (\dim-col (Id 1))))/(sqrt
2) $\widehat{\wedge}$ n
    using tensor-of-H-values Id-def f0 less-mult-imp-div-less by simp
  ultimately show ((H $\widehat{\otimes}$  n)  $\widehat{\otimes}$  Id 1) $$ (i,j) = (HId $\widehat{\otimes}$  n) $$ (i,j)
    using a2 f0 f1 Id-def by simp
next
  assume a2: ¬(i mod 2 = j mod 2)
  then have (Id 1) $$ (i \bmod (\dim-row (Id 1)), j \bmod (\dim-col (Id 1))) = 0
    by (simp add: Quantum.Id-def)
  then show ((H $\widehat{\otimes}$  n)  $\widehat{\otimes}$  Id 1) $$ (i,j) = (HId $\widehat{\otimes}$  n) $$ (i,j)
    using a2 f0 f1 by simp
qed
qed

```

lemma *HId-is-gate*:

```

assumes n ≥ 1
shows gate (n+1) (HId $\widehat{\otimes}$  n)
proof –
  have (HId $\widehat{\otimes}$  n) = (H $\widehat{\otimes}$  n)  $\widehat{\otimes}$  Id 1
    using iter-tensor-of-H-tensor-Id-is-HId by simp
  moreover have gate 1 (Id 1)
    using id-is-gate by simp
  moreover have gate n (H $\widehat{\otimes}$  n)
    using H-is-gate iter-tensor-of-gate-is-gate[of 1 H n] assms by (simp add: iter-tensor-of-H-rep-is-correct)
  ultimately show gate (n+1) (HId $\widehat{\otimes}$  n)
    using tensor-gate by presburger
qed

```

State ψ_3 is obtained by the multiplication of $Matrix.mat[2^{n+1} \times 2^{n+1}]$
 $(\lambda x. complex-of-real(case x of (i, j) \Rightarrow if i \bmod 2 = j \bmod 2 then (-1)^{i \bmod 2} \cdot n^{j \bmod 2} / (\sqrt{2})^n else 0))$ and ψ_2

abbreviation (in *jzcsa*) $\psi_3 :: complex Matrix.mat$ where
 $\psi_3 \equiv Matrix.mat[2^{(n+1)} \times 2^{(n+1)}] (\lambda(i,j).$
if even i
then $(\sum k < 2^n. (-1)^k * ((i \bmod 2) \cdot n^{j \bmod 2})) / ((\sqrt{2})^n * (\sqrt{2})^{(n+1)})$
else $(\sum k < 2^n. (-1)^k * ((i \bmod 2) \cdot n^{j \bmod 2})) / ((\sqrt{2})^n * (\sqrt{2})^{(n+1)}))$

lemma (in *jzcsa*) ψ_3 -values:

```

assumes i < dim-row ψ3
shows odd i → ψ3 $$ (i,0) = (\sum k < 2^n. (-1)^{f(k)} + 1 + ((i \text{ div } 2) \cdot_n k)) / ((\sqrt{2})^n * (\sqrt{2})^{(n+1)}) )
using assms by simp

lemma (in jozsa) ψ3-dim [simp]:
shows 1 < dim-row ψ3
using dim-row-mat(1) nat-neq-iff by fastforce

lemma sum-every-odd-summand-is-zero:
fixes n:: nat
assumes n ≥ 1
shows ∀ f:(nat ⇒ complex). (forall i. i < 2^(n+1) ∧ odd i → f i = 0) →
    (∑ k ∈ {0..< 2^(n+1)}. f k) = (∑ k ∈ {0..< 2^n}. f (2*k))
using assms
proof(rule nat-induct-at-least)
show ∀ f:(nat ⇒ complex). (forall i. i < 2^(1+1) ∧ odd i → f i = 0) →
    (∑ k ∈ {0..< 2^(1+1)}. f k) = (∑ k ∈ {0..< 2^1}. f (2*k))
proof(rule allI,rule impI)
fix f:: (nat ⇒ complex)
assume asm: (forall i. i < 2^(1+1) ∧ odd i → f i = 0)
moreover have (∑ k ∈ {0..< 4}. f k) = f 0 + f 1 + f 2 + f 3
by (simp add: add.commute add.left-commute)
moreover have f 1 = 0
using asm by simp
moreover have f 3 = 0
using asm by simp
moreover have (∑ k ∈ {0..< 2^1}. f (2*k)) = f 0 + f 2
using add.commute add.left-commute by simp
ultimately show (∑ k ∈ {0..< 2^(1+1)}. f k) = (∑ k ∈ {0..< 2^1}. f (2*k))
by simp
qed
next
fix n:: nat
assume n ≥ 1
and IH: ∀ f:(nat ⇒ complex). (forall i. i < 2^(n+1) ∧ odd i → f i = 0) →
    (∑ k ∈ {0..< 2^(n+1)}. f k) = (∑ k ∈ {0..< 2^n}. f (2*k))
show ∀ f:(nat ⇒ complex). (forall i. i < 2^(Suc n + 1) ∧ odd i → f i = 0) →
    (∑ k ∈ {0..< 2^(Suc n + 1)}. f k) = (∑ k ∈ {0..< 2^(Suc n)}. f (2*k))
proof (rule allI,rule impI)
fix f::nat ⇒ complex
assume asm: (forall i. i < 2^(Suc n + 1) ∧ odd i → f i = 0)
have f0: (∑ k ∈ {0..< 2^(n+1)}. f k) = (∑ k ∈ {0..< 2^n}. f (2*k))
using asm IH by simp
have f1: (∑ k ∈ {0..< 2^(n+1)}. (λx. f (x + 2^(n+1))) k) = (∑ k ∈ {0..< 2^n}. (λx. f (x + 2^(n+1))) (2*k))
using asm IH by simp
have (∑ k ∈ {0..< 2^(n+2)}. f k) = (∑ k ∈ {0..< 2^(n+1)}. f k) + (∑ k ∈ {2^(n+1)..< 2^(n+2)}. f k)

```

```

    by (simp add: sum.atLeastLessThan-concat)
  also have ... = ( $\sum_{k \in \{0..<2^n\}} f(2*k)$ ) + ( $\sum_{k \in \{2^{\lceil n+1 \rceil}..<2^{\lceil n+2 \rceil}\}} f(k)$ )
  using f0 by simp
  also have ... = ( $\sum_{k \in \{0..<2^n\}} f(2*k)$ ) + ( $\sum_{k \in \{0..<2^{\lceil n+1 \rceil}\}} f(k+2^{\lceil n+1 \rceil})$ )
  using sum.shift-bounds-nat-ivl[of f 0 2^{\lceil n+1 \rceil} 2^{\lceil n+1 \rceil}] by simp
  also have ... = ( $\sum_{k \in \{0..<2^n\}} f(2*k)$ ) + ( $\sum_{k \in \{0..<2^n\}} (\lambda x. f(x+2^{\lceil n+1 \rceil}))$ )
  using f1 by simp
  also have ... = ( $\sum_{k \in \{0..<2^n\}} f(2*k)$ ) + ( $\sum_{k \in \{2^n..<2^{\lceil n+1 \rceil}\}} f(2*k)$ )
  using sum.shift-bounds-nat-ivl[of  $\lambda x. (f::nat \Rightarrow complex)$  (2*(x-2^n)+2^{\lceil n+1 \rceil})]
  0 2^n 2^n
  by (simp add: mult-2)
  also have ... = ( $\sum_{k \in \{0..<2^{\lceil n+1 \rceil}\}} f(2*k)$ )
  by (metis Suc-eq-plus1 lessI less-imp-le-nat one-le-numeral power-increasing
  sum.atLeastLessThan-concat zero-le)
  finally show ( $\sum_{k \in \{0..<2^{\lceil (Suc n)+1 \rceil}\}} f(k)$ ) = ( $\sum_{k \in \{0..<2^{\lceil Suc n \rceil}\}} f(k)$ )
  by (metis Suc-eq-plus1 add-2-eq-Suc')
qed
qed

lemma sum-every-even-summand-is-zero:
fixes n:: nat
assumes n ≥ 1
shows ∀ f::(nat ⇒ complex). (∀ i. i < 2^{\lceil n+1 \rceil} ∧ even i → f i = 0) →
  ( $\sum_{k \in \{0..<2^{\lceil n+1 \rceil}\}} f(k)$ ) = ( $\sum_{k \in \{0..<2^n\}} f(2*k+1)$ )
using assms
proof(rule nat-induct-at-least)
show ∀ f::(nat ⇒ complex). (∀ i. i < 2^{\lceil 1+1 \rceil} ∧ even i → f i = 0) →
  ( $\sum_{k \in \{0..<2^{\lceil 1+1 \rceil}\}} f(k)$ ) = ( $\sum_{k \in \{0..<2^1\}} f(2*k+1)$ )
proof(rule allI,rule impI)
fix f:: nat ⇒ complex
assume asm: (∀ i. i < 2^{\lceil 1+1 \rceil} ∧ even i → f i = 0)
moreover have ( $\sum_{k \in \{0..<4\}} f(k)$ ) = f 0 + f 1 + f 2 + f 3
  by (simp add: add.commute add.left-commute)
moreover have f 0 = 0 using asm by simp
moreover have f 2 = 0 using asm by simp
moreover have ( $\sum_{k \in \{0..<2^1\}} f(2*k+1)$ ) = f 1 + f 3
  using add.commute add.left-commute by simp
ultimately show ( $\sum_{k \in \{0..<2^{\lceil 1+1 \rceil}\}} f(k)$ ) = ( $\sum_{k \in \{0..<2^1\}} f(2*k+1)$ )
by simp
qed
next
fix n:: nat
assume n ≥ 1
and IH: ∀ f::(nat ⇒ complex). (∀ i. i < 2^{\lceil n+1 \rceil} ∧ even i → f i = 0) →

```

```


$$(\sum k \in \{0..<2^{\lceil(n+1)\rceil}\}. f k) = (\sum k \in \{0..<2^{\lceil n\rceil}\}. f (2*k+1))$$

show  $\forall f:(\text{nat} \Rightarrow \text{complex}). (\forall i. i < 2^{\lceil(Suc n)+1\rceil} \wedge \text{even } i \longrightarrow f i = 0) \longrightarrow$ 

$$(\sum k \in \{0..<2^{\lceil(Suc n)+1\rceil}\}. f k) = (\sum k \in \{0..<2^{\lceil Suc n\rceil}\}. f (2*k+1))$$

proof (rule allI,rule impI)
  fix  $f:\text{nat} \Rightarrow \text{complex}$ 
  assume asm:  $(\forall i. i < 2^{\lceil(Suc n)+1\rceil} \wedge \text{even } i \longrightarrow f i = 0)$ 
  have  $f0: (\sum k \in \{0..<2^{\lceil(n+1)\rceil}\}. f k) = (\sum k \in \{0..<2^{\lceil n\rceil}\}. f (2*k+1))$ 
    using asm IH by simp
  have  $f1: (\sum k \in \{0..<2^{\lceil(n+1)\rceil}\}. (\lambda x. f (x+2^{\lceil(n+1)\rceil})) k)$ 
     $= (\sum k \in \{0..<2^{\lceil n\rceil}\}. (\lambda x. f (x+2^{\lceil(n+1)\rceil})) (2*k+1))$ 
    using asm IH by simp
  have  $(\sum k \in \{0..<2^{\lceil(n+2)\rceil}\}. f k)$ 
     $= (\sum k \in \{0..<2^{\lceil(n+1)\rceil}\}. f k) + (\sum k \in \{2^{\lceil(n+1)\rceil}..<2^{\lceil(n+2)\rceil}\}. f k)$ 
    by (simp add: sum.atLeastLessThan-concat)
  also have ...  $= (\sum k \in \{0..<2^{\lceil n\rceil}\}. f (2*k+1)) + (\sum k \in \{2^{\lceil(n+1)\rceil}..<2^{\lceil(n+2)\rceil}\}. f k)$ 
    using f0 by simp
  also have ...  $= (\sum k \in \{0..<2^{\lceil n\rceil}\}. f (2*k+1)) + (\sum k \in \{0..<2^{\lceil(n+1)\rceil}\}. f (k+(2^{\lceil(n+1)\rceil})))$ 
    using sum.shift-bounds-nat-ivl[of f 0 2^{\lceil(n+1)\rceil} 2^{\lceil(n+1)\rceil}] by simp
  also have ...  $= (\sum k \in \{0..<2^{\lceil n\rceil}\}. f (2*k+1)) + (\sum k \in \{0..<2^{\lceil n\rceil}\}. (\lambda x. f (x+2^{\lceil(n+1)\rceil})) (2*k+1))$ 
    using f1 by simp
  also have ...  $= (\sum k \in \{0..<2^{\lceil n\rceil}\}. f (2*k+1)) + (\sum k \in \{2^{\lceil n\rceil}..<2^{\lceil(n+1)\rceil}\}. f (2*k+1))$ 
    using sum.shift-bounds-nat-ivl[of \lambda x. (f::nat \Rightarrow complex) (2*(x-2^{\lceil n\rceil})+1+2^{\lceil(n+1)\rceil}) 0 2^{\lceil n\rceil} 2^{\lceil n\rceil}] by (simp add: mult-2)
  also have ...  $= (\sum k \in \{0..<2^{\lceil(n+1)\rceil}\}. f (2*k+1))$ 
    by (metis Suc-eq-plus1 lessI less-imp-le-nat one-le-numeral power-increasing sum.atLeastLessThan-concat zero-le)
  finally show  $(\sum k \in \{0..<2^{\lceil(Suc n)+1\rceil}\}. f k) = (\sum k \in \{0..<2^{\lceil Suc n\rceil}\}. f (2*k+1))$ 
    by (metis Suc-eq-plus1 add-2-eq-Suc')
  qed
  qed

```

```

lemma (in jozsa) iter-tensor-of-H-times-ψ₂-is-ψ₃:
  shows  $((H^\wedge_\otimes n) \otimes \text{Id } 1) * \psi_2 = \psi_3$ 
proof
  fix  $i j$ 
  assume  $a0:i < \text{dim-row } \psi_3$  and  $a1:j < \text{dim-col } \psi_3$ 
  then have  $f0: i < (2^{\lceil(n+1)\rceil}) \wedge j = 0$  by simp
  have  $f1: ((H\text{Id}^\wedge_\otimes n) * \psi_2) \$\$ (i,j) = (\sum k < (2^{\lceil(n+1)\rceil}). ((H\text{Id}^\wedge_\otimes n) \$\$ (i,k)) * (\psi_2 \$\$ (k,j)))$ 
    using a1 f0 by (simp add: atLeast0LessThan)
  show  $((H^\wedge_\otimes n) \otimes \text{Id } 1) * \psi_2 \$\$ (i,j) = \psi_3 \$\$ (i,j)$ 
  proof (rule disjE)
    show  $\text{even } i \vee \text{odd } i$  by simp

```

next

assume $a2: even\ i$

have $(\neg(i \ mod\ 2 = k \ mod\ 2) \wedge k < dim_col(HId^\wedge_\otimes n)) \longrightarrow ((HId^\wedge_\otimes n) \$\$ (i,k)) * (\psi_2 \$\$ (k,j)) = 0$ **for** k

using $f0$ **by** *simp*

then have $k < (2^\wedge(n+1)) \wedge odd\ k \longrightarrow ((HId^\wedge_\otimes n) \$\$ (i,k)) * (\psi_2 \$\$ (k,j)) = 0$ **for** k

using $a2\ mod\ 2\ -is\ both\ -even\ -or\ -odd\ f0$ **by** (*metis (no-types, lifting) dim-col-mat(1)*)

then have $(\sum_{k \in \{(0::nat)..<(2^\wedge(n+1))\}} ((HId^\wedge_\otimes n) \$\$ (i,k)) * (\psi_2 \$\$ (k,j))) = (\sum_{k \in \{(0::nat)..<(2^\wedge n)\}} ((HId^\wedge_\otimes n) \$\$ (i,2*k)) * (\psi_2 \$\$ (2*k,j)))$

using *sum-every-odd-summand-is-zero dim by simp*

moreover have $(\sum_{k < 2^\wedge n} ((HId^\wedge_\otimes n) \$\$ (i,2*k)) * (\psi_2 \$\$ (2*k,j))) = (\sum_{k < 2^\wedge n} (-1)^\wedge((i \ div\ 2) \cdot_n k) / (sqrt(2)^\wedge n) * ((-1)^\wedge f(k)) / (sqrt(2)^\wedge(n+1)))$

proof-

have $(even\ k \wedge k < dim_row\ \psi_2) \longrightarrow (\psi_2 \$\$ (k,j)) = ((-1)^\wedge f(k \ div 2)) / (sqrt(2)^\wedge(n+1))$ **for** k

using $a0\ a1$ **by** *simp*

then have $(\sum_{k < 2^\wedge n} ((HId^\wedge_\otimes n) \$\$ (i,2*k)) * (\psi_2 \$\$ (2*k,j))) = (\sum_{k < 2^\wedge n} ((HId^\wedge_\otimes n) \$\$ (i,2*k)) * ((-1)^\wedge f((2*k) \ div 2)) / (sqrt(2)^\wedge(n+1)))$

by *simp*

moreover have $(even\ k \wedge k < dim_col(HId^\wedge_\otimes n)) \longrightarrow ((HId^\wedge_\otimes n) \$\$ (i,k)) = (-1)^\wedge((i \ div\ 2) \cdot_n (k \ div 2)) / (sqrt(2)^\wedge n)$

for k

using $a2\ a0\ a1$ **by** *simp*

ultimately have $(\sum_{k < 2^\wedge n} ((HId^\wedge_\otimes n) \$\$ (i,2*k)) * (\psi_2 \$\$ (2*k,j))) = (\sum_{k < 2^\wedge n} (-1)^\wedge((i \ div\ 2) \cdot_n ((2*k) \ div 2)) / (sqrt(2)^\wedge n) * ((-1)^\wedge f((2*k) \ div 2)) / (sqrt(2)^\wedge(n+1)))$

by *simp*

then show $(\sum_{k < 2^\wedge n} ((HId^\wedge_\otimes n) \$\$ (i,2*k)) * (\psi_2 \$\$ (2*k,j))) = (\sum_{k < 2^\wedge n} (-1)^\wedge((i \ div\ 2) \cdot_n k) / (sqrt(2)^\wedge n) * ((-1)^\wedge f(k)) / (sqrt(2)^\wedge(n+1)))$

by *simp*

qed

ultimately have $((HId^\wedge_\otimes n) * \psi_2) \$\$ (i,j) = (\sum_{k < 2^\wedge n} (-1)^\wedge((i \ div\ 2) \cdot_n k) / (sqrt(2)^\wedge n) * ((-1)^\wedge f(k)) / (sqrt(2)^\wedge(n+1)))$

using $f1$ **by** (*metis atLeast0LessThan*)

also have ... = $(\sum_{k < 2^\wedge n} (-1)^\wedge(f(k) + ((i \ div\ 2) \cdot_n k)) / ((sqrt(2)^\wedge n) * (sqrt(2)^\wedge(n+1))))$

by (*simp add: power-add mult.commute*)

finally have $((HId^\wedge_\otimes n) * \psi_2) \$\$ (i,j) = (\sum_{k < 2^\wedge n} (-1)^\wedge(f(k) + ((i \ div\ 2) \cdot_n k)) / ((sqrt(2)^\wedge n) * (sqrt(2)^\wedge(n+1))))$

by *simp*

moreover have $\psi_3 \$\$ (i,j) = (\sum_{k < 2^\wedge n} (-1)^\wedge(f(k) + ((i \ div\ 2) \cdot_n k)) / (sqrt(2)^\wedge n) * (sqrt(2)^\wedge(n+1)))$

using $a0\ a1\ a2$ **by** *simp*

ultimately show $((H^\wedge_\otimes n) \otimes Id\ 1) * \psi_2) \$\$ (i,j) = \psi_3 \$\$ (i,j)$

```

using iter-tensor-of-H-tensor-Id-is-HId dim by simp
next
  assume a2: odd i
  have (¬(i mod 2 = k mod 2) ∧ k < dim-col (HId $\widehat{\otimes}$  n)) → ((HId $\widehat{\otimes}$  n) $$ (i,k)) * (ψ₂ $$ (k,j)) = 0 for k
    using f0 by simp
    then have k < (2 $\widehat{\wedge}$ (n+1)) ∧ even k → ((HId $\widehat{\otimes}$  n) $$ (i,k)) * (ψ₂ $$ (k,j)) = 0 for k
      using a2 mod-2-is-both-even-or-odd f0 by (metis (no-types, lifting) dim-col-mat(1))
      then have (∑ k ∈ {0..<2 $\widehat{\wedge}$ (n+1)}. ((HId $\widehat{\otimes}$  n) $$ (i,k)) * (ψ₂ $$ (k,j))) =
        = (∑ k ∈ {0..<2 $\widehat{\wedge}$ n}. ((HId $\widehat{\otimes}$  n) $$ (i,2*k+1)) * (ψ₂ $$ (2*k+1,j)))
        using sum-every-even-summand-is-zero dim by simp
      moreover have (∑ k < 2 $\widehat{\wedge}$ n. ((HId $\widehat{\otimes}$  n) $$ (i,2*k+1)) * (ψ₂ $$ (2*k+1,j))) =
        = (∑ k < 2 $\widehat{\wedge}$ n. (-1) $\widehat{\wedge}$ ((i div 2) · n k)/(sqrt(2) $\widehat{\wedge}$ n) * ((-1) $\widehat{\wedge}$ (f(k)+1))/(sqrt(2) $\widehat{\wedge}$ (n+1)))

proof-
  have (odd k ∧ k < dim-row ψ₂) → (ψ₂ $$ (k,j)) = ((-1) $\widehat{\wedge}$ (f(k div 2)+1))/(sqrt(2) $\widehat{\wedge}$ (n+1))
for k
  using a0 a1 a2 by simp
  then have f2:(∑ k < 2 $\widehat{\wedge}$ n. ((HId $\widehat{\otimes}$  n) $$ (i,2*k+1)) * (ψ₂ $$ (2*k+1,j))) =
    = (∑ k < 2 $\widehat{\wedge}$ n. ((HId $\widehat{\otimes}$  n) $$ (i,2*k+1)) * ((-1) $\widehat{\wedge}$ (f((2*k+1) div 2)+1))/(sqrt(2) $\widehat{\wedge}$ (n+1)))
    by simp
  have i < dim-row (HId $\widehat{\otimes}$  n)
    using f0 a2 mod-2-is-both-even-or-odd by simp
    then have ((i mod 2 = k mod 2) ∧ k < dim-col (HId $\widehat{\otimes}$  n))
      → ((HId $\widehat{\otimes}$  n) $$ (i,k)) = (-1) $\widehat{\wedge}$ ((i div 2) · n (k div 2))/(sqrt(2) $\widehat{\wedge}$ n)
for k
  using a2 a0 a1 f0 dim HId-values by simp
  moreover have odd k → (i mod 2 = k mod 2) for k
    using a2 mod-2-is-both-even-or-odd by auto
    ultimately have (odd k ∧ k < dim-col (HId $\widehat{\otimes}$  n))
      → ((HId $\widehat{\otimes}$  n) $$ (i,k)) = (-1) $\widehat{\wedge}$ ((i div 2) · n (k div 2))/(sqrt(2) $\widehat{\wedge}$ n)
for k
  by simp
  then have k < 2 $\widehat{\wedge}$ n → ((HId $\widehat{\otimes}$  n) $$ (i,2*k+1)) = (-1) $\widehat{\wedge}$ ((i div 2) · n ((2*k+1) div 2))/(sqrt(2) $\widehat{\wedge}$ n) for k
    by simp
  then have (∑ k < 2 $\widehat{\wedge}$ n. ((HId $\widehat{\otimes}$  n) $$ (i,2*k+1)) * (ψ₂ $$ (2*k+1,j))) =
    = (∑ k < 2 $\widehat{\wedge}$ n. (-1) $\widehat{\wedge}$ ((i div 2) · n ((2*k+1) div 2))/(sqrt(2) $\widehat{\wedge}$ n) *
      ((-1) $\widehat{\wedge}$ (f((2*k+1) div 2)+1))/(sqrt(2) $\widehat{\wedge}$ (n+1)))
    using f2 by simp
  then show (∑ k < 2 $\widehat{\wedge}$ n. ((HId $\widehat{\otimes}$  n) $$ (i,2*k+1)) * (ψ₂ $$ (2*k+1,j))) =
    = (∑ k < 2 $\widehat{\wedge}$ n. (-1) $\widehat{\wedge}$ ((i div 2) · n k)/(sqrt(2) $\widehat{\wedge}$ n) * ((-1) $\widehat{\wedge}$ (f(k)+1))/(sqrt(2) $\widehat{\wedge}$ (n+1)))

by simp
qed
ultimately have ((HId $\widehat{\otimes}$  n) * ψ₂) $$ (i,j) = (∑ k < 2 $\widehat{\wedge}$ n. (-1) $\widehat{\wedge}$ ((i div 2) · n k)/(sqrt(2) $\widehat{\wedge}$ n))

```

```

        * ((-1)^(f(k)+1))/(sqrt(2)^(n+1)))
    using f1 by (metis atLeast0LessThan)
also have ... = (∑ k<2^n. (-1)^(f(k)+1+((i div 2) · n k))/((sqrt(2)^n)*(sqrt(2)^(n+1))))
    by (simp add: mult.commute power-add)
finally have ((HId^⊗ n)* ψ₂) $$ (i,j)
    = (∑ k<2^n. (-1)^(f(k)+1+((i div 2) · n k))/((sqrt(2)^n)*(sqrt(2)^(n+1))))
    by simp
then show (((H^⊗ n) ⊗ Id 1)* ψ₂) $$ (i,j) = ψ₃ $$ (i,j)
    using iter-tensor-of-H-tensor-Id-is-HId dim a2 a0 a1 by simp
qed
next
show dim-row (((H^⊗ n) ⊗ Id 1) * ψ₂) = dim-row ψ₃
    using iter-tensor-of-H-tensor-Id-is-HId dim by simp
next
show dim-col (((H^⊗ n) ⊗ Id 1)* ψ₂) = dim-col ψ₃
    using iter-tensor-of-H-tensor-Id-is-HId dim by simp
qed

lemma (in jozsa) ψ₃-is-state:
shows state (n+1) ψ₃
proof-
have ((H^⊗ n) ⊗ Id 1) * ψ₂ = ψ₃
    using iter-tensor-of-H-times-ψ₂-is-ψ₃ by simp
moreover have gate (n+1) ((H^⊗ n) ⊗ Id 1)
    using iter-tensor-of-H-tensor-Id-is-HId HId-is-gate dim by simp
moreover have state (n+1) ψ₂
    using ψ₂-is-state by simp
ultimately show state (n+1) ψ₃
    using gate-on-state-is-state dim by (metis (no-types, lifting))
qed

```

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

```

definition (in jozsa) jozsa-algo:: complex Matrix.mat where
jозса-algo ≡ ((H ⊗n) ⊗ Id 1) * (U_f * (((H ⊗n) * (|zero⟩ ⊗n) ⊗ (H * |one⟩)))

```

lemma (in jozsa) jozsa-algo-result [simp]:
shows jozsa-algo = ψ₃
using jozsa-algo-def H-on-ket-one-is-ψ₁₁ iter-tensor-of-H-on-zero-tensor ψ₁₀-tensor-ψ₁₁-is-ψ₁
jозса-transform-times-ψ₁-is-ψ₂ iter-tensor-of-H-times-ψ₂-is-ψ₃ dim iter-tensor-of-H-rep-is-correct

by simp

lemma (in jozsa) jozsa-algo-result-is-state:
shows state (n+1) jozsa-algo
using ψ₃-is-state **by** simp

```

lemma (in jozsa) prob0-fst-qubits-of-jozsa-algo:
  shows (prob0-fst-qubits n jozsa-algo) = ( $\sum_{j \in \{0,1\}} (cmod(jozsa-algo \$\$ (j,0)))^2$ )
  using prob0-fst-qubits-eq by simp

General lemmata required to compute probabilities.

lemma aux-comp-with-sqrt2:
  shows ( $\sqrt{2}$ ) $^n * (\sqrt{2})^n = 2^n$ 
  by (smt (verit) power-mult-distrib real-sqrt-mul-self)

lemma aux-comp-with-sqrt2-bis [simp]:
  shows  $2^n / (\sqrt{2})^n * \sqrt{2}^{(n+1)} = 1/\sqrt{2}$ 
  using aux-comp-with-sqrt2 by (simp add: mult.left-commute)

lemma aux-ineq-with-card:
  fixes g:: nat  $\Rightarrow$  nat and A:: nat set
  assumes finite A
  shows ( $\sum_{k \in A} (-1)^{(g k)}$ )  $\leq card A$  and ( $\sum_{k \in A} (-1)^{(g k)}$ )  $\geq -card A$ 
  apply (smt (verit) assms neg-one-even-power neg-one-odd-power card-eq-sum
of-nat-1 of-nat-sum sum-mono)
  apply (smt (verit) assms neg-one-even-power neg-one-odd-power card-eq-sum
of-nat-1 of-nat-sum sum-mono sum-neg).

lemma aux-comp-with-cmod:
  fixes g:: nat  $\Rightarrow$  nat
  assumes ( $\forall x < 2^n. g x = 0$ )  $\vee$  ( $\forall x < 2^n. g x = 1$ )
  shows ( $cmod(\sum_{k < 2^n} (-1)^{(g k)})^2 = 2^{(2*n)}$ )
  proof(rule disjE)
    show ( $\forall x < 2^n. g x = 0$ )  $\vee$  ( $\forall x < 2^n. g x = 1$ )
    using assms by simp
  next
    assume  $\forall x < 2^n. g x = 0$ 
    then have ( $cmod(\sum_{k < 2^n} (-1)^{(g k)})^2 = (2^n)^2$ )
    by (simp add: norm-power)
    then show ?thesis
    by (simp add: power-even-eq)
  next
    assume  $\forall x < 2^n. g x = 1$ 
    then have ( $cmod(\sum_{k < 2^n} (-1)^{(g k)})^2 = (2^n)^2$ )
    by (simp add: norm-power)
    then show ?thesis
    by (simp add: power-even-eq)
  qed

lemma cmod-less:
  fixes a n:: int
  assumes a < n and a > -n
  shows cmod a < n
  using assms by simp

```

```

lemma square-less:
  fixes a n:: real
  assumes a < n and a > -n
  shows a2 < n2
  using assms by (smt (verit) power2-eq-iff power2-minus power-less-imp-less-base)

lemma cmod-square-real [simp]:
  fixes n:: real
  shows (cmod n)2 = n2
  by simp

lemma aux-comp-sum-divide-cmod:
  fixes n:: nat and g:: nat  $\Rightarrow$  int and a:: real
  shows (cmod(complex-of-real( $\sum k < n. g k / a$ )))2 = (cmod ( $\sum k < n. g k$ ) / a)2
  by (metis cmod-square-real of-int-sum of-real-of-int-eq power-divide sum-divide-distrib)

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

lemma (in jozsa) prob0-jozsa-algo-of-const-0:
  assumes const 0
  shows prob0-fst-qubits n jozsa-algo = 1
proof-
  have prob0-fst-qubits n jozsa-algo = ( $\sum j \in \{0,1\}. (cmod(jozsa-algo \$\$ (j,0)))^2$ )
    using prob0-fst-qubits-of-jozsa-algo by simp
  moreover have (cmod(jozsa-algo \$\$ (0,0)))2 = 1/2
proof-
  have k < 2^n  $\rightarrow$  ((0 div 2) ·n k) = 0 for k::nat
    using bitwise-inner-prod-with-zero by simp
  then have (cmod(jozsa-algo \$\$ (0,0)))2 = (cmod( $\sum k :: nat < 2^n. 1 / (\sqrt(2)^n * \sqrt(2)^{(n+1)})$ ))2
    using jozsa-algo-result const-def assms by simp
  also have ... = (cmod((2::nat)^n / (sqrt(2)^n * sqrt(2)^(n+1))))2 by simp
  also have ... = (cmod(1 / (sqrt(2))))2
    using aux-comp-with-sqrt2-bis by simp
  also have ... = 1/2
    by (simp add: norm-divide power2-eq-square)
  finally show ?thesis by simp
qed
moreover have (cmod(jozsa-algo \$\$ (1,0)))2 = 1/2
proof-
  have k < 2^n  $\rightarrow$  ((1 div 2) ·n k) = 0 for k:: nat
    using bitwise-inner-prod-with-zero by simp
  then have k < 2^n  $\rightarrow$  f k + 1 + ((1 div 2) ·n k) = 1 for k::nat
    using const-def assms by simp
  moreover have (cmod(jozsa-algo \$\$ (1,0)))2
    = (cmod ( $\sum k :: nat < 2^n. (-1)^{f k + 1} + ((1 \text{ div } 2) \cdot_n k) / (\sqrt(2)^n * \sqrt(2)^{(n+1)})$ ))2

```

```

    using  $\psi_3$ -dim by simp
  ultimately have  $(\text{cmod}(\text{jozsa-algo} \$\$ (1,0)))^2 = (\text{cmod}(\sum k::nat < 2^n. -1 / (\sqrt{2})^n * \sqrt{2}^{(n+1)}))^2$ 
    by (smt (verit) lessThan-iff power-one-right sum.cong)
  also have ... =  $(\text{cmod}(-1 / (\sqrt{2})))^2$ 
    using aux-comp-with-sqrt2-bis by simp
  also have ... =  $1/2$ 
    by (simp add: norm-divide power2-eq-square)
  finally show ?thesis by simp
qed
ultimately have prob0-fst-qubits n jozsa-algo =  $1/2 + 1/2$  by simp
then show ?thesis by simp
qed

lemma (in jozsa) prob0-jozsa-algo-of-const-1:
assumes const 1
shows prob0-fst-qubits n jozsa-algo = 1
proof-
  have prob0-fst-qubits n jozsa-algo =  $(\sum j \in \{0,1\}. (\text{cmod}(\text{jozsa-algo} \$\$ (j,0)))^2)$ 
    using prob0-fst-qubits-of-jozsa-algo by simp
  moreover have  $(\text{cmod}(\text{jozsa-algo} \$\$ (0,0)))^2 = 1/2$ 
proof-
  have  $k < 2^n \rightarrow ((0 \text{ div } 2) \cdot_n k) = 0$  for  $k::nat$ 
    using bitwise-inner-prod-with-zero by simp
  then have  $(\text{cmod}(\text{jozsa-algo} \$\$ (0,0)))^2 = (\text{cmod}(\sum k::nat < 2^n. 1 / (\sqrt{2})^n * \sqrt{2}^{(n+1)}))^2$ 
    using jozsa-algo-result const-def assms by simp
  also have ... =  $(\text{cmod}((-1) / (\sqrt{2})))^2$ 
    using aux-comp-with-sqrt2-bis by simp
  also have ... =  $1/2$ 
    by (simp add: norm-divide power2-eq-square)
  finally show ?thesis by simp
qed
moreover have  $(\text{cmod}(\text{jozsa-algo} \$\$ (1,0)))^2 = 1/2$ 
proof-
  have  $k < 2^n \rightarrow ((1 \text{ div } 2) \cdot_n k) = 0$  for  $k::nat$ 
    using bitwise-inner-prod-with-zero by simp
  then have  $(\sum k::nat < 2^n. (-1)^f k + 1 + ((1 \text{ div } 2) \cdot_n k)) / (\sqrt{2})^n * \sqrt{2}^{(n+1)})$ 
    =  $(\sum k::nat < 2^n. 1 / (\sqrt{2})^n * \sqrt{2}^{(n+1)})$ 
    using const-def assms by simp
  moreover have  $(\text{cmod}(\text{jozsa-algo} \$\$ (1,0)))^2$ 
    =  $(\text{cmod}(\sum k::nat < 2^n. (-1)^f k + 1 + ((1 \text{ div } 2) \cdot_n k)) / (\sqrt{2})^n * \sqrt{2}^{(n+1)}))^2$ 
    using  $\psi_3$ -dim by simp
  ultimately have  $(\text{cmod}(\text{jozsa-algo} \$\$ (1,0)))^2 = (\text{cmod}(\sum k::nat < 2^n. 1 / (\sqrt{2})^n * \sqrt{2}^{(n+1)}))^2$  by simp
  also have ... =  $(\text{cmod}(1 / (\sqrt{2})))^2$ 
    using aux-comp-with-sqrt2-bis by simp

```

```

also have ... = 1 / 2
  by (simp add: norm-divide power2-eq-square)
  finally show ?thesis by simp
qed
ultimately have prob0-fst-qubits n jozsa-algo = 1 / 2 + 1 / 2 by simp
then show ?thesis by simp
qed

```

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

```

lemma (in jozsa) max-value-of-not-const-less:
assumes ¬ const 0 and ¬ const 1
shows (cmod (∑ k::nat < 2^n. (-(1::nat))^(f k)))^2 < (2::nat)^^(2*n)
proof-
have cmod (∑ k::nat < 2^n. (-(1::nat))^(f k)) < 2^n
proof-
have (∑ k::nat < 2^n. (-(1::nat))^(f k)) < 2^n
proof-
obtain x where f0:x < 2^n and f1:f x = 1
using assms(1) const-def f-values by auto
then have (∑ k::nat < 2^n. (-(1::nat))^(f k)) < (∑ k ∈ {i | i :: nat. i < 2^n} - {x}. (-(1::nat))^(f k))
proof-
have (-(1::nat))^(f x) = -1 using f1 by simp
moreover have x ∈ {i | i :: nat. i < 2^n} using f0 by simp
moreover have finite {i | i :: nat. i < 2^n} by simp
moreover have (∑ k ∈ {i | i :: nat. i < 2^n}. (-(1::nat))^(f k)) <
(∑ k ∈ {i | i :: nat. i < 2^n} - {x}. (-(1::nat))^(f k))
using calculation(1,2,3) sum-diff1 by (simp add: sum-diff1)
ultimately show ?thesis by (metis Collect-cong Collect-mem-eq lessThan-iff)
qed
moreover have ... ≤ int (2^n - 1)
using aux-ineq-with-card(1)[of {i | i :: nat. i < 2^n} - {x}] f0 by simp
ultimately show ?thesis
  by (meson diff-le-self less-le-trans of-nat-le-numeral-power-cancel-iff)
qed
moreover have (∑ k::nat < 2^n. (-(1::nat))^(f k)) > - (2^n)
proof-
obtain x where f0:x < 2^n and f1:f x = 0
using assms(2) const-def f-values by auto
then have (∑ k::nat < 2^n. (-(1::nat))^(f k)) > (∑ k ∈ {i | i :: nat. i < 2^n} - {x}. (-(1::nat))^(f k))
proof-
have (-(1::nat))^(f x) = 1 using f1 by simp
moreover have x ∈ {i | i :: nat. i < 2^n} using f0 by simp
moreover have finite {i | i :: nat. i < 2^n} by simp
moreover have (∑ k ∈ {i | i :: nat. i < 2^n}. (-(1::nat))^(f k)) >
(∑ k ∈ {i | i :: nat. i < 2^n} - {x}. (-(1::nat))^(f k))
using calculation(1,2,3) sum-diff1 by (simp add: sum-diff1)

```

```

ultimately show ?thesis by (metis Collect-cong Collect-mem-eq lessThan-iff)
qed
  moreover have - int (2^n - 1) ≤ (∑ k∈{i| i:: nat. i < 2^n} - {x}.(-(1::nat))^(f k))
    using aux-ineq-with-card(2)[of {i| i:: nat. i<2^n} - {x}] f0 by simp
  ultimately show ?thesis
    by (smt (verit) diff-le-self of-nat-1 of-nat-add of-nat-power-le-of-nat-cancel-iff
one-add-one)
  qed
  ultimately show ?thesis
    using cmod-less of-int-of-nat-eq of-nat-numeral of-nat-power by (metis (no-types,
lifting))
  qed
  then have (cmod (∑ k::nat<2^n. (-(1::nat))^(f k)))^2 < (2^n)^2
    using square-less norm-ge-zero by (smt (verit))
  thus ?thesis
    by (simp add: power-even-eq)
qed

lemma (in jozsa) max-value-of-not-const-less-bis:
assumes ¬ const 0 and ¬ const 1
shows (cmod (∑ k::nat<2^n. (-(1::nat))^(f k + 1)))^2 < (2::nat)^^(2*n)
proof-
  have cmod (∑ k::nat<2^n. (-(1::nat))^(f k + 1)) < 2^n
  proof-
    have (∑ k::nat<2^n. (-(1::nat))^(f k + 1)) < 2^n
    proof-
      obtain x where f0:x < 2^n and f1:f x = 0
        using assms(2) const-def f-values by auto
      then have (∑ k::nat<2^n. (-(1::nat))^(f k + 1)) < (∑ k∈{i| i:: nat. i <
2^n} - {x}. (-(1::nat))^(f k + 1))
      proof-
        have (-(1::nat))^(f x + 1) = -1 using f1 by simp
        moreover have x∈{i| i:: nat. i<2^n} using f0 by simp
        moreover have finite {i| i:: nat. i<2^n} by simp
        moreover have (∑ k∈{i| i:: nat. i<2^n}. (-(1::nat))^(f k + 1)) <
(∑ k∈{i| i:: nat. i<2^n} - {x}. (-(1::nat))^(f k + 1))
          using calculation(1,2,3) sum-diff1 by (simp add: sum-diff1)
        ultimately show ?thesis by (metis Collect-cong Collect-mem-eq lessThan-iff)
      qed
      moreover have ... ≤ int (2^n - 1)
        using aux-ineq-with-card(1)[of {i| i:: nat. i<2^n} - {x} λk. f k + 1] f0 by
simp
      ultimately show ?thesis
        by (meson diff-le-self less-le-trans of-nat-le-numeral-power-cancel-iff)
    qed
    moreover have (∑ k::nat<2^n. (-(1::nat))^(f k + 1)) > - (2^n)
    proof-
      obtain x where f0:x < 2^n and f1:f x = 1

```

```

using assms(1) const-def f-values by auto
then have  $(\sum k::nat < 2^n. (-(1::nat)) \wedge(f k + 1)) > (\sum k \in \{i | i :: nat. i < 2^n\} - \{x\}. (-(1::nat)) \wedge(f k + 1))$ 
proof-
  have  $(-(1::nat)) \wedge(f x + 1) = 1$  using f1 by simp
  moreover have  $x \in \{i | i :: nat. i < 2^n\}$  using f0 by simp
  moreover have finite  $\{i | i :: nat. i < 2^n\}$  by simp
  moreover have  $(\sum k \in \{i | i :: nat. i < 2^n\}. (-(1::nat)) \wedge(f k + 1)) > (\sum k \in \{i | i :: nat. i < 2^n\} - \{x\}. (-(1::nat)) \wedge(f k + 1))$ 
    using calculation(1,2,3) sum-diff1 by (simp add: sum-diff1)
  ultimately show ?thesis by (metis Collect-cong Collect-mem-eq lessThan-iff)
qed
  moreover have  $- \text{int } (2^n - 1) \leq (\sum k \in \{i | i :: nat. i < 2^n\} - \{x\}. (-(1::nat)) \wedge(f k + 1))$ 
    using aux-ineq-with-card(2)[of  $\{i | i :: nat. i < 2^n\} - \{x\}$   $\lambda k. f k + 1$ ] f0 by simp
  ultimately show ?thesis
  by (smt (verit) diff-le-self of-nat-1 of-nat-add of-nat-power-le-of-nat-cancel-iff one-add-one)
qed
ultimately show ?thesis
using cmod-less of-int-of-nat-eq of-nat-numeral of-nat-power by (metis (no-types, lifting))
qed
then have  $(cmod (\sum k::nat < 2^n. (-(1::nat)) \wedge(f k + 1)))^2 < (2^n)^2$ 
  using square-less norm-ge-zero by (smt (verit))
thus ?thesis
  by (simp add: power-even-eq)
qed

lemma (in jozsa) f-const-has-max-value:
assumes const 0 ∨ const 1
shows  $(cmod (\sum k < (2::nat) \wedge n. (-1) \wedge(f k)))^2 = (2::nat) \wedge(2*n)$ 
and  $(cmod (\sum k < (2::nat) \wedge n. (-1) \wedge(f k + 1)))^2 = (2::nat) \wedge(2*n)$ 
using aux-comp-with-cmod[of n  $\lambda k. f k$ ] aux-comp-with-cmod[of n  $\lambda k. f k + 1$ ] const-def assms by auto

lemma (in jozsa) prob0-fst-qubits-leq:
shows  $(cmod (\sum k < (2::nat) \wedge n. (-1) \wedge(f k)))^2 \leq (2::nat) \wedge(2*n)$ 
and  $(cmod (\sum k < (2::nat) \wedge n. (-1) \wedge(f k + 1)))^2 \leq (2::nat) \wedge(2*n)$ 
proof-
  show  $(cmod (\sum k < (2::nat) \wedge n. (-1) \wedge(f k)))^2 \leq (2::nat) \wedge(2*n)$ 
  proof(rule disjE)
    show  $(const 0 \vee const 1) \vee (\neg const 0 \wedge \neg const 1)$  by auto
  next
    assume const 0 ∨ const 1
    then show  $(cmod (\sum k < (2::nat) \wedge n. (-1) \wedge(f k)))^2 \leq (2::nat) \wedge(2*n)$ 
      using f-const-has-max-value by simp
  next

```

```

assume  $\neg \text{const } 0 \wedge \neg \text{const } 1$ 
then show ( $\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k)})^2 \leq (2:\text{nat})^{\wedge(2*n)}$ )
  using max-value-of-not-const-less by simp
qed
next
  show ( $\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k + 1)})^2 \leq (2:\text{nat})^{\wedge(2*n)}$ )
  proof(rule disjE)
    show ( $\text{const } 0 \vee \text{const } 1 \vee (\neg \text{const } 0 \wedge \neg \text{const } 1)$  by auto
next
  assume  $\text{const } 0 \vee \text{const } 1$ 
  then show ( $\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k + 1)})^2 \leq (2:\text{nat})^{\wedge(2*n)}$ )
    using f-const-has-max-value by simp
next
  assume  $\neg \text{const } 0 \wedge \neg \text{const } 1$ 
  then show ( $\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k + 1)})^2 \leq (2:\text{nat})^{\wedge(2*n)}$ )
    using max-value-of-not-const-less-bis by simp
qed
qed

lemma (in jozsa) prob0-jozsa-algo-1-is-const:
assumes prob0-fst-qubits n jozsa-algo = 1
shows const 0  $\vee$  const 1
proof-
  have f0:  $(\sum j \in \{0,1\}. (\text{cmod(jozsa-algo \$\$ (j,0)))}^2) = 1$ 
  using prob0-fst-qubits-of-jozsa-algo assms by simp
  have k <  $2^{\wedge n} \rightarrow ((0 \text{ div } 2) \cdot_n k) = 0$  for k:nat
  using bitwise-inner-prod-with-zero by simp
  then have f1:  $(\text{cmod(jozsa-algo \$\$ (0,0)))}^2 = (\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k / (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)})))^2$ 
  by simp
  have k <  $2^{\wedge n} \rightarrow ((1 \text{ div } 2) \cdot_n k) = 0$  for k:nat
  using bitwise-inner-prod-with-zero by simp
  moreover have  $(\text{cmod(jozsa-algo \$\$ (1,0)))}^2 = (\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k + 1 + ((1 \text{ div } 2) \cdot_n k)) / (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)}))^2$ 
  using  $\psi_3\text{-dim}$  by simp
  ultimately have f2:  $(\text{cmod(jozsa-algo \$\$ (1,0)))}^2 = (\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k + 1))} / (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)}))^2$  by simp
  have f3:  $1 = (\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k / (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)})))^2 + (\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k + 1))} / (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)}))^2$ 
  using f0 f1 f2 by simp
  also have ... =  $((\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k))}) / (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)})^2 + ((\text{cmod}(\sum k < (2:\text{nat})^{\wedge n}. (-1)^{\wedge(f k + 1))}) / (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)})^2$ 
  using aux-comp-sum-divide-cmod[of  $\lambda k. (-1)^{\wedge(f k)} (\sqrt{2})^{\wedge n} * \sqrt{2})^{\wedge(n+1)}$ ]  $(2:\text{nat})^{\wedge n}$ 

```

```

aux-comp-sum-divide-cmod[of  $\lambda k. (-1)^{\lceil f k + 1 \rceil} (\sqrt{2})^n * \sqrt{2}^{\lceil n+1 \rceil}$ 
(2::nat) $^n$ ]
by simp
also have ... = ((cmod ( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k \rceil}$ ))) $^2$  / (( $\sqrt{2}^n * \sqrt{2}^{\lceil n+1 \rceil}$ )) $^2$ 
+ ((cmod( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k + 1 \rceil}$ ))) $^2$  / (( $\sqrt{2}^n * \sqrt{2}^{\lceil n+1 \rceil}$ )) $^2$ 
by (simp add: power-divide)
also have ... = ((cmod ( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k \rceil}$ )) $^2$  / (2 $^{\lceil 2*n+1 \rceil}$ )
+ ((cmod( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k + 1 \rceil}$ ))) $^2$  / (2 $^{\lceil 2*n+1 \rceil}$ )
by (smt (verit) left-add-twice power2-eq-square power-add power-mult-distrib
real-sqrt-pow2)
also have ... = (((cmod ( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k \rceil}$ ))) $^2$ 
+ ((cmod( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k + 1 \rceil}$ ))) $^2$ ) / (2 $^{\lceil 2*n+1 \rceil}$ )
by (simp add: add-divide-distrib)
finally have ((2::nat) $^{\lceil 2*n+1 \rceil}$ ) = (((cmod ( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k \rceil}$ ))) $^2$ 
+ ((cmod( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k + 1 \rceil}$ ))) $^2$ ) by simp
moreover have ((2::nat) $^{\lceil 2*n+1 \rceil}$ ) = 2 $^{\lceil 2*n \rceil} + 2^{\lceil 2*n \rceil}$  by auto
moreover have (cmod ( $\sum k < (2::nat)^n. (-1)^{\lceil f k \rceil}$ )) $^2$   $\leq 2^{\lceil 2*n \rceil}$ 
using prob0-fst-qubits-leq by simp
moreover have (cmod ( $\sum k < (2::nat)^n. (-1)^{\lceil f k + 1 \rceil}$ )) $^2$   $\leq 2^{\lceil 2*n \rceil}$ 
using prob0-fst-qubits-leq by simp
ultimately have 2 $^{\lceil 2*n \rceil} = ((cmod ( $\sum k::nat < (2::nat)^n. (-1)^{\lceil f k \rceil}$ ))) $^2$  by
simp
then show ?thesis
using max-value-of-not-const-less by auto
qed$ 
```

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

```

lemma sum-union-disjoint-finite-set:
fixes C::nat set and g::nat  $\Rightarrow$  int
assumes finite C
shows  $\forall A B. A \cap B = \{\} \wedge A \cup B = C \longrightarrow (\sum k \in C. g k) = (\sum k \in A. g k) + (\sum k \in B. g k)$ 
using assms sum.union-disjoint by auto

lemma (in jozsa) balanced-pos-and-neg-terms-cancel-out1:
assumes is-balanced
shows ( $\sum k < (2::nat)^n. (-(1::nat))^{\lceil f k \rceil}$ ) = 0
proof-
have  $\bigwedge A B. A \subseteq \{i::nat. i < (2::nat)^n\} \wedge B \subseteq \{i::nat. i < (2::nat)^n\}$ 
 $\wedge card A = ((2::nat)^{\lceil n-1 \rceil}) \wedge card B = ((2::nat)^{\lceil n-1 \rceil})$ 
 $\wedge (\forall (x::nat) \in A. f x = (0::nat)) \wedge (\forall (x::nat) \in B. f x = 1)$ 
 $\longrightarrow (\sum k < (2::nat)^n. (-(1::nat))^{\lceil f k \rceil}) = 0$ 
proof
fix A B::nat set

```

```

assume asm:  $A \subseteq \{i:\text{nat}. i < (2:\text{nat})^n\} \wedge B \subseteq \{i:\text{nat}. i < (2:\text{nat})^n\}$ 
       $\wedge \text{card } A = ((2:\text{nat})^n)$   $\wedge \text{card } B = ((2:\text{nat})^n)$ 
       $\wedge (\forall (x:\text{nat}) \in A. f x = (0:\text{nat})) \wedge (\forall (x:\text{nat}) \in B. f x = 1)$ 
then have  $A \cap B = \{\}$  and  $\{0..<(2:\text{nat})^n\} = A \cup B$ 
    using is-balanced-union is-balanced-inter by auto
then have  $(\sum k \in \{0..<(2:\text{nat})^n\}. (-(1:\text{nat}))^{(f k)}) =$ 
     $(\sum k \in A. (-(1:\text{nat}))^{(f k)})$ 
     $+ (\sum k \in B. (-(1:\text{nat}))^{(f k)})$ 
    by (metis finite-atLeastLessThan sum-union-disjoint-finite-set)
moreover have  $(\sum k \in A. (-(1:\text{nat}))^{(f k)}) = ((2:\text{nat})^n)$ 
    using asm by simp
moreover have  $(\sum k \in B. (-(1:\text{nat}))^{(f k)}) = -((2:\text{nat})^n)$ 
    using asm by simp
ultimately have  $(\sum k \in \{0..<(2:\text{nat})^n\}. (-(1:\text{nat}))^{(f k)}) = 0$  by simp
then show  $(\sum k < (2:\text{nat})^n. (-(1:\text{nat}))^{(f k)}) = 0$ 
    by (simp add: lessThan-atLeast0)
qed
then show ?thesis
    using assms is-balanced-def by auto
qed

lemma (in jozsa) balanced-pos-and-neg-terms-cancel-out2:
assumes is-balanced
shows  $(\sum k < (2:\text{nat})^n. (-(1:\text{nat}))^{(f k + 1)}) = 0$ 
proof –
have  $\bigwedge A B. A \subseteq \{i:\text{nat}. i < (2:\text{nat})^n\} \wedge B \subseteq \{i:\text{nat}. i < (2:\text{nat})^n\}$ 
       $\wedge \text{card } A = ((2:\text{nat})^n)$   $\wedge \text{card } B = ((2:\text{nat})^n)$ 
       $\wedge (\forall (x:\text{nat}) \in A. f x = (0:\text{nat})) \wedge (\forall (x:\text{nat}) \in B. f x = 1)$ 
     $\longrightarrow (\sum k < (2:\text{nat})^n. (-(1:\text{nat}))^{(f k + 1)}) = 0$ 
proof
  fix  $A B : \text{nat set}$ 
  assume asm:  $A \subseteq \{i:\text{nat}. i < (2:\text{nat})^n\} \wedge B \subseteq \{i:\text{nat}. i < (2:\text{nat})^n\}$ 
       $\wedge \text{card } A = ((2:\text{nat})^n)$   $\wedge \text{card } B = ((2:\text{nat})^n)$ 
       $\wedge (\forall (x:\text{nat}) \in A. f x = (0:\text{nat})) \wedge (\forall (x:\text{nat}) \in B. f x = 1)$ 
  have  $A \cap B = \{\}$  and  $\{0..<(2:\text{nat})^n\} = A \cup B$ 
    using is-balanced-union is-balanced-inter asm by auto
  then have  $(\sum k \in \{0..<(2:\text{nat})^n\}. (-(1:\text{nat}))^{(f k + 1)}) =$ 
     $(\sum k \in A. (-(1:\text{nat}))^{(f k + 1)})$ 
     $+ (\sum k \in B. (-(1:\text{nat}))^{(f k + 1)})$ 
    by (metis finite-atLeastLessThan sum-union-disjoint-finite-set)
  moreover have  $(\sum k \in A. (-(1:\text{nat}))^{(f k + 1)}) = -((2:\text{nat})^n)$ 
    using asm by simp
  moreover have  $(\sum k \in B. (-(1:\text{nat}))^{(f k + 1)}) = ((2:\text{nat})^n)$ 
    using asm by simp
  ultimately have  $(\sum k \in \{0..<(2:\text{nat})^n\}. (-(1:\text{nat}))^{(f k + 1)}) = 0$  by simp
  then show  $(\sum k < (2:\text{nat})^n. (-(1:\text{nat}))^{(f k + 1)}) = 0$ 
    by (simp add: lessThan-atLeast0)
qed
then show  $(\sum k < (2:\text{nat})^n. (-(1:\text{nat}))^{(f k + 1)}) = 0$ 

```

```

using assms is-balanced-def by auto
qed

lemma (in jozsa) prob0-jozsa-algo-of-balanced:
assumes is-balanced
shows prob0-fst-qubits n jozsa-algo = 0
proof-
have prob0-fst-qubits n jozsa-algo = ( $\sum_{j \in \{0,1\}} (cmod(jozsa-algo \$\$ (j,0)))^2$ )
using prob0-fst-qubits-of-jozsa-algo by simp
moreover have  $(cmod(jozsa-algo \$\$ (0,0)))^2 = 0$ 
proof-
have  $k < 2^n \rightarrow ((1 \text{ div } 2) \cdot_n k) = 0$  for  $k::nat$ 
using bitwise-inner-prod-with-zero by simp
then have  $(cmod(jozsa-algo \$\$ (0,0)))^2 = (cmod(\sum k < (2::nat)^n. (-1)^{(f k) / (\sqrt(2)^n * \sqrt(2)^(n+1))}))^2$ 
using  $\psi_3$ -values by simp
also have ... =  $(cmod(\sum k < (2::nat)^n. (-(1::nat))^{\lceil f k \rceil}) / (\sqrt(2)^n * \sqrt(2)^(n+1)))^2$ 

using aux-comp-sum-divide-cmod[of  $\lambda k. (-(1::nat))^{\lceil f k \rceil} (\sqrt(2)^n * \sqrt(2)^(n+1))^{2^n}$ ]
by simp
also have ... =  $(cmod((0::int) / (\sqrt(2)^n * \sqrt(2)^(n+1))))^2$ 
using balanced-pos-and-neg-terms-cancel-out1 assms by (simp add: bob-fun-axioms)
also have ... = 0 by simp
finally show ?thesis by simp
qed

moreover have  $(cmod(jozsa-algo \$\$ (1,0)))^2 = 0$ 
proof-
have  $k < 2^n \rightarrow (((1::nat) \text{ div } 2) \cdot_n k) = 0$  for  $k::nat$ 
using bitwise-inner-prod-with-zero by auto
moreover have  $(cmod(jozsa-algo \$\$ (1,0)))^2 = (cmod(\sum k < (2::nat)^n. (-(1::nat))^{\lceil f k + (1::nat) + ((1 \text{ div } 2) \cdot_n k) \rceil} / (\sqrt(2)^n * \sqrt(2)^(n+1))))^2$ 
using  $\psi_3$ -dim by simp
ultimately have  $(cmod(jozsa-algo \$\$ (1,0)))^2 = (cmod(\sum k < (2::nat)^n. (-(1::nat))^{\lceil f k + (1::nat) \rceil} / (\sqrt(2)^n * \sqrt(2)^(n+1))))^2$ 

by simp
also have ... =  $(cmod(\sum k < (2::nat)^n. (-(1::nat))^{\lceil f k + 1 \rceil}) / (\sqrt(2)^n * \sqrt(2)^(n+1)))^2$ 
using aux-comp-sum-divide-cmod[of  $\lambda k. (-(1::nat))^{\lceil f k + 1 \rceil} (\sqrt(2)^n * \sqrt(2)^(n+1))^{2^n}$ ]
by simp
also have ... =  $(cmod((0::int) / (\sqrt(2)^n * \sqrt(2)^(n+1))))^2$ 
using balanced-pos-and-neg-terms-cancel-out2 assms by (simp add: bob-fun-axioms)
also have ... = 0 by simp
finally show ?thesis by simp
qed

ultimately have prob0-fst-qubits n jozsa-algo = 0 + 0 by simp

```

```

then show ?thesis by simp
qed

```

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

```

lemma (in jozsa) balanced-prob0-jozsa-algo:
  assumes prob0-fst-qubits n jozsa-algo = 0
  shows is-balanced
proof-
  have is-const ∨ is-balanced
  using const-or-balanced by simp
  moreover have is-const → prob0-fst-qubits n jozsa-algo = 0
  using is-const-def prob0-jozsa-algo-of-const-0 prob0-jozsa-algo-of-const-1 by
  simp
  ultimately show ?thesis
  using assms by simp
qed

```

We prove the correctness of the algorithm.

```

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

```

```

theorem (in jozsa) jozsa-algo-is-correct:
  shows jozsa-algo-eval = 1 ↔ is-const
  and jozsa-algo-eval = 0 ↔ is-balanced
  using prob0-jozsa-algo-of-const-1 prob0-jozsa-algo-of-const-0 jozsa-algo-eval-def
prob0-jozsa-algo-1-is-const is-const-def balanced-prob0-jozsa-algo prob0-jozsa-algo-of-balanced
  by auto

```

end

12 The No-Cloning Theorem

```

theory No-Cloning
imports
  Quantum
  Tensor
begin

```

12.1 The Cauchy-Schwarz Inequality

```

lemma inner-prod-expand:
  assumes dim-vec a = dim-vec b and dim-vec a = dim-vec c and dim-vec a =
dim-vec d
  shows ⟨a + b | c + d⟩ = ⟨a | c⟩ + ⟨a | d⟩ + ⟨b | c⟩ + ⟨b | d⟩
  apply (simp add: inner-prod-def)
  using assms sum.cong by (simp add: sum.distrib algebra-simps)

```

```

lemma inner-prod-distrib-left:
  assumes dim-vec a = dim-vec b
  shows  $\langle c \cdot_v a | b \rangle = cnj(c) * \langle a | b \rangle$ 
  using assms inner-prod-def by (simp add: algebra-simps mult-hom.hom-sum)

lemma inner-prod-distrib-right:
  assumes dim-vec a = dim-vec b
  shows  $\langle a | c \cdot_v b \rangle = c * \langle a | b \rangle$ 
  using assms by (simp add: algebra-simps mult-hom.hom-sum)

lemma cauchy-schwarz-ineq:
  assumes dim-vec v = dim-vec w
  shows ( $cmod(\langle v | w \rangle)$ )2 ≤ Re ( $\langle v | v \rangle * \langle w | w \rangle$ )
  proof (cases  $\langle v | v \rangle = 0$ )
    case c0:True
      then have  $\bigwedge i. i < dim\text{-}vec v \implies v \$ i = 0$ 
      by (metis index-zero-vec(1) inner-prod-with-itself-nonneg-reals-non0)
      then have ( $cmod(\langle v | w \rangle)$ )2 = 0 by (simp add: assms inner-prod-def)
      moreover have Re ( $\langle v | v \rangle * \langle w | w \rangle$ ) = 0 by (simp add: c0)
      ultimately show ?thesis by simp
    next
      case c1:False
      have dim-vec w = dim-vec ( $-\langle v | w \rangle / \langle v | v \rangle \cdot_v v$ ) by (simp add: assms)
      then have  $\langle w + -\langle v | w \rangle / \langle v | v \rangle \cdot_v v | w + -\langle v | w \rangle / \langle v | v \rangle \cdot_v v \rangle = \langle w | w \rangle + \langle w | -\langle v | w \rangle / \langle v | v \rangle \cdot_v v \rangle +$ 
       $\langle -\langle v | w \rangle / \langle v | v \rangle \cdot_v v | w \rangle + \langle -\langle v | w \rangle / \langle v | v \rangle \cdot_v v | -\langle v | w \rangle / \langle v | v \rangle \cdot_v v \rangle$ 
      using inner-prod-expand[of w -<v|w>/<v|v> ·v v w -<v|w>/<v|v> ·v v] by auto
      moreover have  $\langle w | -\langle v | w \rangle / \langle v | v \rangle \cdot_v v \rangle = -\langle v | w \rangle / \langle v | v \rangle * \langle w | v \rangle$ 
      using assms inner-prod-distrib-right[of w v -<v|w>/<v|v>] by simp
      moreover have  $\langle -\langle v | w \rangle / \langle v | v \rangle \cdot_v v | w \rangle = cnj(-\langle v | w \rangle / \langle v | v \rangle) * \langle v | w \rangle$ 
      using assms inner-prod-distrib-left[of v w -<v|w>/<v|v>] by simp
      moreover have  $\langle -\langle v | w \rangle / \langle v | v \rangle \cdot_v v | -\langle v | w \rangle / \langle v | v \rangle \cdot_v v \rangle = cnj(-\langle v | w \rangle / \langle v | v \rangle) * (-\langle v | w \rangle / \langle v | v \rangle) * \langle v | v \rangle$ 
      using inner-prod-distrib-left[of v -<v|w>/<v|v> ·v v -<v|w>/<v|v>]
      inner-prod-distrib-right[of v v -<v|w>/<v|v>] by simp
      ultimately have  $\langle w + -\langle v | w \rangle / \langle v | v \rangle \cdot_v v | w + -\langle v | w \rangle / \langle v | v \rangle \cdot_v v \rangle = \langle w | w \rangle -$ 
       $cmod(\langle v | w \rangle)^2 / \langle v | v \rangle$ 
      using assms inner-prod-cnj[of w v] inner-prod-cnj[of v v] complex-norm-square
      by simp
      moreover have Re( $\langle w + -\langle v | w \rangle / \langle v | v \rangle \cdot_v v | w + -\langle v | w \rangle / \langle v | v \rangle \cdot_v v \rangle$ ) ≥ 0
      using inner-prod-with-itself-Re by blast
      ultimately have Re( $\langle w | w \rangle$ ) ≥  $cmod(\langle v | w \rangle)^2 / Re(\langle v | v \rangle)$ 
      using inner-prod-with-itself-real by simp
      moreover have c2:Re( $\langle v | v \rangle$ ) > 0
      using inner-prod-with-itself-Re-non0 inner-prod-with-itself-eq0 c1 by auto
      ultimately have Re( $\langle w | w \rangle$ ) * Re( $\langle v | v \rangle$ ) ≥  $cmod(\langle v | w \rangle)^2 / Re(\langle v | v \rangle) * Re(\langle v | v \rangle)$ 
      by (metis less-numeral-extra(3) nonzero-divide-eq-eq pos-divide-le-eq)
      thus ?thesis

```

```

using inner-prod-with-itself-Im c2 by (simp add: mult.commute)
qed

```

```

lemma cauchy-schwarz-eq [simp]:
assumes  $v = (l \cdot_v w)$ 
shows  $(\text{cmod}(\langle v|w \rangle))^2 = \text{Re} (\langle v|v \rangle * \langle w|w \rangle)$ 
proof-
  have  $\text{cmod}(\langle v|w \rangle) = \text{cmod}(\text{cnj}(l) * \langle w|w \rangle)$ 
    using assms inner-prod-distrib-left[of w w l] by simp
  then have  $\text{cmod}(\langle v|w \rangle)^2 = \text{cmod}(l)^2 * \langle w|w \rangle * \langle w|w \rangle$ 
    using complex-norm-square inner-prod-cnj[of w w] by simp
  moreover have  $\langle v|v \rangle = \text{cmod}(l)^2 * \langle w|w \rangle$ 
    using assms complex-norm-square inner-prod-distrib-left[of w v l]
inner-prod-distrib-right[of w w l] by simp
  ultimately show ?thesis by (metis Re-complex-of-real)
qed

```

```

lemma cauchy-schwarz-col [simp]:
assumes dim-vec  $v = \text{dim-vec } w$  and  $(\text{cmod}(\langle v|w \rangle))^2 = \text{Re} (\langle v|v \rangle * \langle w|w \rangle)$ 
shows  $\exists l. v = (l \cdot_v w) \vee w = (l \cdot_v v)$ 
proof (cases  $\langle v|v \rangle = 0$ )
  case c0:True
    then have  $\bigwedge i. i < \text{dim-vec } v \implies v \$ i = 0$ 
      by(metis index-zero-vec(1) inner-prod-with-itself-nonneg-reals-non0)
    then have  $v = 0 \cdot_v w$  by (auto simp: assms)
    then show ?thesis by auto
  next
    case c1:False
      have  $f0:\text{dim-vec } w = \text{dim-vec } (-\langle v|w \rangle / \langle v|v \rangle \cdot_v v)$  by (simp add: assms(1))
      then have  $\langle w + -\langle v|w \rangle / \langle v|v \rangle \cdot_v v | w + -\langle v|w \rangle / \langle v|v \rangle \cdot_v v \rangle = \langle w|w \rangle + \langle w | -\langle v|w \rangle / \langle v|v \rangle \cdot_v v \rangle +$ 
 $\langle -\langle v|w \rangle / \langle v|v \rangle \cdot_v v | w \rangle + \langle -\langle v|w \rangle / \langle v|v \rangle \cdot_v v | -\langle v|w \rangle / \langle v|v \rangle \cdot_v v \rangle$ 
        using inner-prod-expand[of w -<v|w>/<v|v> · v w -<v|w>/<v|v> · v v] by simp
      moreover have  $\langle w | -\langle v|w \rangle / \langle v|v \rangle \cdot_v v \rangle = -\langle v|w \rangle / \langle v|v \rangle * \langle w|v \rangle$ 
        using assms(1) inner-prod-distrib-right[of w v -<v|w>/<v|v>] by simp
      moreover have  $\langle -\langle v|w \rangle / \langle v|v \rangle \cdot_v v | w \rangle = \text{cnj}(-\langle v|w \rangle / \langle v|v \rangle) * \langle v|w \rangle$ 
        using assms(1) inner-prod-distrib-left[of v w -<v|w>/<v|v>] by simp
      moreover have  $\langle -\langle v|w \rangle / \langle v|v \rangle \cdot_v v | -\langle v|w \rangle / \langle v|v \rangle \cdot_v v \rangle = \text{cnj}(-\langle v|w \rangle / \langle v|v \rangle) * (-\langle v|w \rangle / \langle v|v \rangle) * \langle v|v \rangle$ 
        using inner-prod-distrib-left[of v -<v|w>/<v|v> · v w -<v|w>/<v|v>]
inner-prod-distrib-right[of v v -<v|w>/<v|v>] by simp
      ultimately have  $\langle w + -\langle v|w \rangle / \langle v|v \rangle \cdot_v v | w + -\langle v|w \rangle / \langle v|v \rangle \cdot_v v \rangle = \langle w|w \rangle -$ 
 $\text{cmod}(\langle v|w \rangle)^2 / \langle v|v \rangle$ 
        using inner-prod-cnj[of w v] inner-prod-cnj[of v v] assms(1) complex-norm-square
        by simp
      moreover have  $\langle w|w \rangle = \text{cmod}(\langle v|w \rangle)^2 / \langle v|v \rangle$ 
        using assms(2) inner-prod-with-itself-real by(metis Reals-mult c1 nonzero-mult-div-cancel-left
of-real-Re)
      ultimately have  $\langle w + -\langle v|w \rangle / \langle v|v \rangle \cdot_v v | w + -\langle v|w \rangle / \langle v|v \rangle \cdot_v v \rangle = 0$  by simp

```

```

then have  $\bigwedge i. i < \text{dim-vec } w \implies (w + -\langle v|w \rangle / \langle v|v \rangle \cdot_v v) \$ i = 0$ 
  by (metis f0 index-add-vec(2) index-zero-vec(1) inner-prod-with-itself-nonneg-reals-non0)
then have  $\bigwedge i. i < \text{dim-vec } w \implies w \$ i + -\langle v|w \rangle / \langle v|v \rangle * v \$ i = 0$ 
  by (metis assms(1) f0 index-add-vec(1) index-smult-vec(1))
then have  $\bigwedge i. i < \text{dim-vec } w \implies w \$ i = \langle v|w \rangle / \langle v|v \rangle * v \$ i$  by simp
then have  $w = \langle v|w \rangle / \langle v|v \rangle \cdot_v v$  by (auto simp add: assms(1))
thus ?thesis by auto
qed

```

12.2 The No-Cloning Theorem

```

lemma eq-from-inner-prod [simp]:
  assumes dim-vec  $v = \text{dim-vec } w$  and  $\langle v|w \rangle = 1$  and  $\langle v|v \rangle = 1$  and  $\langle w|w \rangle = 1$ 
  shows  $v = w$ 
proof-
  have  $(\text{cmod}(\langle v|w \rangle))^2 = \text{Re} (\langle v|v \rangle * \langle w|w \rangle)$  by (simp add: assms)
  then have  $f0: \exists l. v = (l \cdot_v w) \vee w = (l \cdot_v v)$  by (simp add: assms(1))
  then show ?thesis
  proof (cases  $\exists l. v = (l \cdot_v w)$ )
    case True
    then have  $\exists l. v = (l \cdot_v w) \wedge \langle v|w \rangle = \text{cnj}(l) * \langle w|w \rangle$ 
      using inner-prod-distrib-left by auto
    then show ?thesis by (simp add: assms(2,4))
  next
    case False
    then have  $\exists l. w = (l \cdot_v v) \wedge \langle v|w \rangle = l * \langle v|v \rangle$ 
      using f0 inner-prod-distrib-right by auto
    then show ?thesis by (simp add: assms(2,3))
  qed
qed

```

```

lemma hermite-cnj-of-tensor:
  shows  $(A \otimes B)^\dagger = (A^\dagger) \otimes (B^\dagger)$ 
proof
  show c0: dim-row  $((A \otimes B)^\dagger) = \text{dim-row } ((A^\dagger) \otimes (B^\dagger))$  by simp
  show c1: dim-col  $((A \otimes B)^\dagger) = \text{dim-col } ((A^\dagger) \otimes (B^\dagger))$  by simp
  show  $\bigwedge i j. i < \text{dim-row } ((A^\dagger) \otimes (B^\dagger)) \implies j < \text{dim-col } ((A^\dagger) \otimes (B^\dagger)) \implies$ 
     $((A \otimes B)^\dagger) \$\$ (i, j) = ((A^\dagger) \otimes (B^\dagger)) \$\$ (i, j)$ 
  proof-
    fix  $i j$  assume a0:  $i < \text{dim-row } ((A^\dagger) \otimes (B^\dagger))$  and a1:  $j < \text{dim-col } ((A^\dagger) \otimes (B^\dagger))$ 
    then have  $(A \otimes B)^\dagger \$\$ (i, j) = \text{cnj}((A \otimes B) \$\$ (j, i))$  by (simp add: dagger-def)
    also have ... =  $\text{cnj}(A \$\$ (j \text{ div } \text{dim-row}(B), i \text{ div } \text{dim-col}(B)) * B \$\$ (j \text{ mod } \text{dim-row}(B), i \text{ mod } \text{dim-col}(B)))$ 
      by (metis (mono-tags, lifting) a0 a1 c1 dim-row-tensor-mat dim-col-of-dagger
        dim-row-of-dagger
        index-tensor-mat less-nat-zero-code mult-not-zero neq0-conv)
    moreover have  $((A^\dagger) \otimes (B^\dagger)) \$\$ (i, j) =$ 

```

```

( $A^\dagger$ ) $$ (i \text{ div } \text{dim-col}(B), j \text{ div } \text{dim-row}(B)) * ( $B^\dagger$ ) $$ (i \text{ mod } \text{dim-col}(B), j \text{ mod } \text{dim-row}(B))
  by (smt (verit) a0 a1 c1 dim-row-tensor-mat dim-col-of-dagger dim-row-of-dagger
  index-tensor-mat
  less-nat-zero-code mult-eq-0-iff neq0-conv)
  moreover have ( $B^\dagger$ ) $$ (i \text{ mod } \text{dim-col}(B), j \text{ mod } \text{dim-row}(B)) = \text{cnj}(B) $$ (j
  mod dim-row(B), i mod dim-col(B))
  proof-
    have i mod dim-col(B) < dim-col(B)
    using a0 gr-implies-not-zero mod-div-trivial by fastforce
    moreover have j mod dim-row(B) < dim-row(B)
    using a1 gr-implies-not-zero mod-div-trivial by fastforce
    ultimately show ?thesis by (simp add: dagger-def)
  qed
  moreover have ( $A^\dagger$ ) $$ (i \text{ div } \text{dim-col}(B), j \text{ div } \text{dim-row}(B)) = \text{cnj}(A) $$ (j
  div dim-row(B), i div dim-col(B))
  proof-
    have i div dim-col(B) < dim-col(A)
    using a0 dagger-def by (simp add: less-mult-imp-div-less)
    moreover have j div dim-row(B) < dim-row(A)
    using a1 dagger-def by (simp add: less-mult-imp-div-less)
    ultimately show ?thesis by (simp add: dagger-def)
  qed
  ultimately show (( $A \otimes B$ ) $^\dagger$ ) $$ (i, j) = ((A^\dagger) \otimes (B^\dagger)) $$ (i, j) by simp
  qed
qed

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

lemma inner-prod-of-unit-vec:
  fixes n i:: nat
  assumes i < n
  shows  $\langle \text{unit-vec } n | \text{unit-vec } n \rangle = 1$ 
  by (auto simp add: inner-prod-def unit-vec-def)
  (simp add: assms sum.cong[of {0.. $n$ } {0.. $n$ }]
   $\lambda j. \text{cnj} (\text{if } j = i \text{ then } 1 \text{ else } 0) * (\text{if } j = i \text{ then } 1 \text{ else } 0) \lambda j. (\text{if } j = i \text{ then } 1 \text{ else } 0)]$ 

theorem (in quantum-machine) no-cloning:
  assumes [simp]: dim-vec v =  $2^n$  and [simp]: dim-vec w =  $2^n$  and
  cloning1:  $\bigwedge s. U * (|v\rangle \otimes |s\rangle) = |v\rangle \otimes |v\rangle$  and
  cloning2:  $\bigwedge s. U * (|w\rangle \otimes |s\rangle) = |w\rangle \otimes |w\rangle$  and
   $\langle v|v\rangle = 1$  and  $\langle w|w\rangle = 1$ 
  shows v = w  $\vee \langle v|w\rangle = 0$ 
  proof-

```

```

define s:: complex Matrix.vec where d0:s = unit-vec (2^n) 0
have f0:(|v⟩| ⊗ ⟨|s⟩|) = ((|v⟩ ⊗ |s⟩)†)
  using hermite-cnj-of-tensor[of |v⟩ |s⟩] bra-def dagger-def ket-vec-def by simp
moreover have f1:(|v⟩ ⊗ |v⟩)† * (|w⟩ ⊗ |w⟩) = (⟨|v⟩| ⊗ ⟨|s⟩|) * (|w⟩ ⊗ |s⟩)
|s⟩)
proof-
  have (U * (|v⟩ ⊗ |s⟩))† = (⟨|v⟩| ⊗ ⟨|s⟩|) * (U†)
    using dagger-of-prod[of U |v⟩ ⊗ |s⟩] f0 d0 by (simp add: ket-vec-def)
    then have (U * (|v⟩ ⊗ |s⟩))† * U * (|w⟩ ⊗ |s⟩) = (⟨|v⟩| ⊗ ⟨|s⟩|) * (U†) *
  U * (|w⟩ ⊗ |s⟩) by simp
    moreover have (U * (|v⟩ ⊗ |s⟩))† * U * (|w⟩ ⊗ |s⟩) = ((|v⟩ ⊗ |v⟩)†) *
  (|w⟩ ⊗ |w⟩)
      using assms(2-4) d0 unit-vec-def by (smt (verit) Matrix.dim-vec assoc-mult-mat
carrier-mat-triv dim-row-mat(1)
dim-row-tensor-mat dim-col-of-dagger index-mult-mat(2) ket-vec-def square square-mat.elims(2))
  moreover have (U†) * U = 1_m (2^n * 2^n)
    using unitary-def dim-col unitary by simp
  moreover have (⟨|v⟩| ⊗ ⟨|s⟩|) * (U†) * U = (⟨|v⟩| ⊗ ⟨|s⟩|) * ((U†) * U)
    using d0 assms(1) unit-vec-def by (smt (verit) Matrix.dim-vec assoc-mult-mat
carrier-mat-triv dim-row-mat(1)
dim-row-tensor-mat f0 dim-col-of-dagger dim-row-of-dagger ket-vec-def local.dim-col)
  moreover have (⟨|v⟩| ⊗ ⟨|s⟩|) * 1_m (2^n * 2^n) = (⟨|v⟩| ⊗ ⟨|s⟩|)
    using f0 ket-vec-def d0 by simp
  ultimately show ?thesis by simp
qed
then have f2:(⟨|v⟩| * |w⟩) ⊗ (⟨|v⟩| * |w⟩) = (⟨|v⟩| * |w⟩) ⊗ (⟨|s⟩| * |s⟩)
proof-
  have ⟨|v⟩| ⊗ ⟨|v⟩| = ((|v⟩ ⊗ |v⟩)†)
    using hermite-cnj-of-tensor[of |v⟩ |v⟩] bra-def dagger-def ket-vec-def by simp
  then show ?thesis
    using f1 d0 by (simp add: bra-def mult-distr-tensor ket-vec-def)
qed
then have ⟨v|w⟩ * ⟨v|w⟩ = ⟨v|w⟩ * ⟨s|s⟩
proof-
  have ((⟨|v⟩| * |w⟩) ⊗ (⟨|v⟩| * |w⟩)) $$ (0,0) = ⟨v|w⟩ * ⟨v|w⟩
    using assms inner-prod-with-times-mat[of v w] by (simp add: bra-def ket-vec-def)
  moreover have ((⟨|v⟩| * |w⟩) ⊗ (⟨|s⟩| * |s⟩)) $$ (0,0) = ⟨v|w⟩ * ⟨s|s⟩
    using inner-prod-with-times-mat[of v w] inner-prod-with-times-mat[of s s]
by(simp add: bra-def ket-vec-def)
  ultimately show ?thesis using f2 by auto
qed
then have ⟨v|w⟩ = 0 ∨ ⟨v|w⟩ = ⟨s|s⟩ by (simp add: mult-left-cancel)
moreover have ⟨s|s⟩ = 1 by (simp add: d0 inner-prod-of-unit-vec)
ultimately show ?thesis using assms(1,2,5,6) by auto
qed
end

```

13 Quantum Prisoner's Dilemma

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

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

13.1 The Set-Up

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

abbreviation (in prisoner)  $J :: complex Matrix.mat$  where

$$J \equiv mat-of-cols-list 4 [[cos(\gamma/2), 0, 0, i*sin(\gamma/2)],$$


$$[0, cos(\gamma/2), -i*sin(\gamma/2), 0],$$


$$[0, -i*sin(\gamma/2), cos(\gamma/2), 0],$$


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


abbreviation (in prisoner)  $\psi_1 :: complex Matrix.mat$  where

$$\psi_1 \equiv mat-of-cols-list 4 [[cos(\gamma/2), 0, 0, i*sin(\gamma/2)]]$$


lemma (in prisoner) psi-one:
  shows  $J * |unit-vec 4 0\rangle = \psi_1$ 
proof
  fix  $i j$  assume  $a0:i < dim-row \psi_1$  and  $a1:j < dim-col \psi_1$ 
  then have  $(J * |unit-vec 4 0\rangle) \$\$ (i,j) = (\sum k < 4. (J \$\$ (i,k)) * (|unit-vec 4 0\rangle \$\$ (k,j)))$ 
  using mat-of-cols-list-def ket-vec-def by auto
  also have ... =  $(\sum k \in \{0,1,2,3\}. (J \$\$ (i,k)) * (|unit-vec 4 0\rangle \$\$ (k,j)))$ 
  using set-4 atLeast0LessThan by simp
  also have ... =  $\psi_1 \$\$ (i,j)$ 
  proof-
    have  $i \in \{0,1,2,3\} \wedge j = 0$ 
    using a0 a1 mat-of-cols-list-def by auto
    thus ?thesis
```

```

    using mat-of-cols-list-def by auto
qed
finally show (J * |unit-vec 4 0>) $$ (i,j) = ψ_1 $$ (i,j) by simp
next
show dim-row (J * |unit-vec 4 0>) = dim-row ψ_1
using mat-of-cols-list-def by simp
next
show dim-col (J * |unit-vec 4 0>) = dim-col ψ_1
using mat-of-cols-list-def by (simp add: ket-vec-def)
qed

locale strategic-space-2p = prisoner +
fixes θ_A:: real
and φ_A:: real
and θ_B:: real
and φ_B:: real
assumes 0 ≤ θ_A ∧ θ_A ≤ pi
and 0 ≤ φ_A ∧ φ_A ≤ pi/2
and 0 ≤ θ_B ∧ θ_B ≤ pi
and 0 ≤ φ_B ∧ φ_B ≤ pi/2

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

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

abbreviation (in strategic-space-2p) ψ_2 :: complex Matrix.mat where
ψ_2 ≡
mat-of-cols-list 4 [[exp(i*φ_A)*cos(θ_A/2) * exp(i*φ_B)*cos(θ_B/2) * cos(γ/2) + sin(θ_A/2)
* sin(θ_B/2) * i*sin(γ/2),
exp(i*φ_A)*cos(θ_A/2) * -sin(θ_B/2) * cos(γ/2) + sin(θ_A/2) *
exp(-i*φ_B)*cos(θ_B/2) * i*sin(γ/2),
-sin(θ_A/2) * exp(i*φ_B)*cos(θ_B/2) * cos(γ/2) + exp(-i*φ_A)*cos(θ_A/2) *
* sin(θ_B/2) * i*sin(γ/2),
sin(θ_A/2) * sin(θ_B/2) * cos(γ/2) + exp(-i*φ_A)*cos(θ_A/2) *
exp(-i*φ_B)*cos(θ_B/2) * i*sin(γ/2)]]]

abbreviation (in strategic-space-2p) U_AB :: complex Matrix.mat where
U_AB ≡
mat-of-cols-list 4 [[exp(i*φ_A)*cos(θ_A/2) * exp(i*φ_B)*cos(θ_B/2), exp(i*φ_A)*cos(θ_A/2) *
* -sin(θ_B/2),
-sin(θ_A/2) * exp(i*φ_B)*cos(θ_B/2), -sin(θ_A/2) *
-sin(θ_B/2)],
[exp(i*φ_A)*cos(θ_A/2) * sin(θ_B/2), exp(i*φ_A)*cos(θ_A/2) *
exp(-i*φ_B)*cos(θ_B/2),
-sin(θ_A/2) * sin(θ_B/2), -sin(θ_A/2) * exp(-i*φ_B)*cos(θ_B/2)],
```

```

[sin(θ_A/2) * exp(i*φ_B)*cos(θ_B/2), -sin(θ_A/2) * sin(θ_B/2),
 exp(-i*φ_A)*cos(θ_A/2) * exp (i*φ_B)*cos(θ_B/2),
exp(-i*φ_A)*cos(θ_A/2) * -sin(θ_B/2)],
[sin(θ_A/2) * sin(θ_B/2), sin(θ_A/2) * exp(-i*φ_B)*cos(θ_B/2),
 exp(-i*φ_A)*cos(θ_A/2) * sin(θ_B/2), exp (-i*φ_A)*cos(θ_A/2)
* exp(-i*φ_B)*cos(θ_B/2)]]

```

lemma (in strategic-space-2p) U_A -tensor- U_B :
shows $(U_A \otimes U_B) = U_{AB}$
proof
fix $i j$ **assume** $a0: i < \text{dim-row } U_{AB}$ **and** $a1: j < \text{dim-col } U_{AB}$
then have $i \in \{0,1,2,3\} \wedge j \in \{0,1,2,3\}$
using mat-of-cols-list-def by auto
then show $(U_A \otimes U_B) \$\$ (i,j) = U_{AB} \$\$ (i,j)$
using mat-of-cols-list-def by auto
next
show $\text{dim-row } (U_A \otimes U_B) = \text{dim-row } U_{AB}$
using mat-of-cols-list-def by simp
next
show $\text{dim-col } (U_A \otimes U_B) = \text{dim-col } U_{AB}$
using mat-of-cols-list-def by simp
qed

lemma (in strategic-space-2p) ψ_1 -two:
shows $(U_A \otimes U_B) * \psi_1 = \psi_2$
proof
fix $i j$
assume $a0: i < \text{dim-row } \psi_2$ **and** $a1: j < \text{dim-col } \psi_2$
then show $((U_A \otimes U_B) * \psi_1) \$\$ (i,j) = \psi_2 \$\$ (i,j)$
proof-
have $i \in \{0,1,2,3\} \wedge j = 0$
using a0 a1 mat-of-cols-list-def by auto
thus ?thesis
using mat-of-cols-list-def U_A -tensor- U_B set-4 by auto
qed
next
show $\text{dim-row } ((U_A \otimes U_B) * \psi_1) = \text{dim-row } \psi_2$
using mat-of-cols-list-def by simp
next
show $\text{dim-col } ((U_A \otimes U_B) * \psi_1) = \text{dim-col } \psi_2$
using mat-of-cols-list-def by simp
qed

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

```

lemma (in prisoner) hermite-cnj-of-J [simp]:
  shows  $J^\dagger = J\text{-cnj}$ 
proof
  fix  $i j$  assume  $a0:i < \text{dim-row } J\text{-cnj}$  and  $a1:j < \text{dim-col } J\text{-cnj}$ 
  then show  $J^\dagger \$\$ (i,j) = J\text{-cnj} \$\$ (i,j)$ 
proof-
  have  $i \in \{0,1,2,3\} \wedge j \in \{0,1,2,3\}$ 
  using  $a0 a1 \text{ mat-of-cols-list-def by auto}$ 
  thus ?thesis
    using  $\text{mat-of-cols-list-def dagger-def by auto}$ 
qed
next
  show  $\text{dim-row } (J^\dagger) = \text{dim-row } J\text{-cnj}$ 
  using  $\text{mat-of-cols-list-def by simp}$ 
next
  show  $\text{dim-col } (J^\dagger) = \text{dim-col } J\text{-cnj}$ 
  using  $\text{mat-of-cols-list-def by simp}$ 
qed

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

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

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

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

lemma (in strategic-space-2p)  $\psi_2$ :
  shows  $(J^\dagger) * \psi_2 = \psi_f$ 
proof
  fix  $i j$  assume  $a0:i < \text{dim-row } \psi_f$  and  $a1:j < \text{dim-col } \psi_f$ 
  then show  $((J^\dagger) * \psi_2) \$\$ (i,j) = \psi_f \$\$ (i,j)$ 
proof-
  have  $i \in \{0,1,2,3\} \wedge j = 0$ 

```

```

using a0 a1 mat-of-cols-list-def by auto
thus ?thesis
  using mat-of-cols-list-def set-4 hermite-cnj-of-J by auto
qed
next
show dim-row ((J†) * ψ2) = dim-row ψf
  using mat-of-cols-list-def by simp
next
show dim-col ((J†) * ψ2) = dim-col ψf
  using mat-of-cols-list-def by simp
qed

lemma (in prisoner) unit-vec-4-0-ket-is-state:
shows state 2 |unit-vec 4 0⟩
  using state-def cpx-vec-length-def ket-vec-def unit-vec-def by (auto simp add:
set-4)

lemma cos-sin-squared-add-cpx:
complex-of-real (cos (γ/2)) * complex-of-real (cos (γ/2)) −
i*complex-of-real (sin (γ/2)) * (i*complex-of-real (sin (γ/2))) = 1
  apply (auto simp add: algebra-simps)
  by (metis of-real-add of-real-hom.hom-one of-real-mult sin-cos-squared-add3)

lemma sin-cos-squared-add-cpx:
i*complex-of-real (sin (γ/2)) * (i*complex-of-real (sin (γ/2))) −
complex-of-real (cos (γ/2)) * complex-of-real (cos (γ/2)) = −1
  apply (auto simp add: algebra-simps)
  by (metis of-real-add of-real-hom.hom-one of-real-mult sin-cos-squared-add3)

lemma (in prisoner) J-cnj-times-J:
shows J† * J = 1m 4
proof
fix i j assume a0:i < dim-row (1m 4) and a1:j < dim-col (1m 4)
then show (J† * J) $(i,j) = 1m 4 $(i,j)
proof-
have i ∈ {0,1,2,3} ∧ j ∈ {0,1,2,3}
  using a0 a1 mat-of-cols-list-def by auto
thus ?thesis
  using mat-of-cols-list-def hermite-cnj-of-J set-4 cos-sin-squared-add-cpx by
auto
qed
next
show dim-row (J† * J) = dim-row (1m 4)
  using mat-of-cols-list-def by simp
next
show dim-col (J† * J) = dim-col (1m 4)
  using mat-of-cols-list-def by simp
qed

```

```

lemma (in prisoner) J-times-J-cnj:
  shows  $J * (J^\dagger) = 1_m 4$ 
proof
  fix  $i j$  assume  $a0:i < \text{dim-row} (1_m 4)$  and  $a1:j < \text{dim-col} (1_m 4)$ 
  then show  $(J * (J^\dagger)) \$\$ (i,j) = 1_m 4 \$\$ (i,j)$ 
proof-
  have  $i \in \{0,1,2,3\} \wedge j \in \{0,1,2,3\}$ 
  using  $a0 a1 \text{mat-of-cols-list-def}$  by auto
  thus  $?thesis$ 
    using mat-of-cols-list-def hermite-cnj-of-J set-4 cos-sin-squared-add-cpx by
    auto
  qed
next
  show  $\text{dim-row} (J * (J^\dagger)) = \text{dim-row} (1_m 4)$ 
  using mat-of-cols-list-def by simp
next
  show  $\text{dim-col} (J * (J^\dagger)) = \text{dim-col} (1_m 4)$ 
  using mat-of-cols-list-def by simp
qed

lemma (in prisoner) J-is-gate:
  shows gate 2 J
proof
  show  $\text{dim-row } J = 2^2$ 
  using mat-of-cols-list-def by simp
  moreover show square-mat J
  using mat-of-cols-list-def by simp
  ultimately show unitary J
  using mat-of-cols-list-def unitary-def J-cnj-times-J J-times-J-cnj by auto
qed

lemma (in strategic-space-2p) psi-one-is-state:
  shows state 2  $\psi_1$ 
proof-
  have state 2 ( $J * |\text{unit-vec } 4 0\rangle$ )
  using unit-vec-4-0-ket-is-state J-is-gate by auto
  then show  $?thesis$ 
  using psi-one by simp
qed

abbreviation (in strategic-space-2p) U_A-cnj :: complex Matrix.mat where

$$U_A\text{-cnj} \equiv \text{mat-of-cols-list } 2 [[(\exp(-i*\varphi_A))*\cos(\vartheta_A/2), \sin(\vartheta_A/2)], [-\sin(\vartheta_A/2), (\exp(i*\varphi_A))*\cos(\vartheta_A/2)]]$$


abbreviation (in strategic-space-2p) U_B-cnj :: complex Matrix.mat where

$$U_B\text{-cnj} \equiv \text{mat-of-cols-list } 2 [[(\exp(-i*\varphi_B))*\cos(\vartheta_B/2), \sin(\vartheta_B/2)], [-\sin(\vartheta_B/2), (\exp(i*\varphi_B))*\cos(\vartheta_B/2)]]$$


lemma (in strategic-space-2p) hermite-cnj-of-U_A:
```

```

shows  $U_A^\dagger = U_A\text{-cnj}$ 
proof
fix  $i j$  assume  $a0:i < \dim\text{-row } U_A\text{-cnj}$  and  $a1:j < \dim\text{-col } U_A\text{-cnj}$ 
then show  $U_A^\dagger \$\$ (i,j) = U_A\text{-cnj} \$\$ (i,j)$ 
proof-
have  $i \in \{0,1\} \wedge j \in \{0,1\}$ 
using  $a0 a1 \text{ mat-of-cols-list-def by auto}$ 
thus ?thesis
using  $\text{mat-of-cols-list-def dagger-def exp-of-real-cnj exp-of-real-cnj2 by auto}$ 
qed
next
show  $\dim\text{-row } (U_A^\dagger) = \dim\text{-row } U_A\text{-cnj}$ 
using  $\text{mat-of-cols-list-def by simp}$ 
next
show  $\dim\text{-col } (U_A^\dagger) = \dim\text{-col } U_A\text{-cnj}$ 
using  $\text{mat-of-cols-list-def by simp}$ 
qed

lemma (in strategic-space-2p) hermite-cnj-of- $U_B$ :
shows  $U_B^\dagger = U_B\text{-cnj}$ 
proof
fix  $i j$  assume  $a0:i < \dim\text{-row } U_B\text{-cnj}$  and  $a1:j < \dim\text{-col } U_B\text{-cnj}$ 
then show  $U_B^\dagger \$\$ (i,j) = U_B\text{-cnj} \$\$ (i,j)$ 
proof-
have  $i \in \{0,1\} \wedge j \in \{0,1\}$ 
using  $a0 a1 \text{ mat-of-cols-list-def by auto}$ 
thus ?thesis
using  $\text{mat-of-cols-list-def dagger-def exp-of-real-cnj exp-of-real-cnj2 by auto}$ 
qed
next
show  $\dim\text{-row } (U_B^\dagger) = \dim\text{-row } U_B\text{-cnj}$ 
using  $\text{mat-of-cols-list-def by simp}$ 
next
show  $\dim\text{-col } (U_B^\dagger) = \dim\text{-col } U_B\text{-cnj}$ 
using  $\text{mat-of-cols-list-def by simp}$ 
qed

lemma exp-sin-cos-squared-add:
fixes  $x y :: \text{real}$ 
shows  $\exp(-(\text{i} * x)) * \cos(y) * (\exp(\text{i} * x) * \cos(y)) + \sin(y) * \sin(y) = 1$ 
proof-
have  $\exp(-(\text{i} * x)) * \cos(y) * (\exp(\text{i} * x) * \cos(y)) = \cos(y) * \cos(y)$ 
using  $\text{exp-minus-inverse by (auto simp add: algebra-simps)}$ 
then show ?thesis
by (metis of-real-add of-real-hom.hom-one sin-cos-squared-add3)
qed

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

```

```

proof
fix i j assume a0:i < dim-row (1m 2) and a1:j < dim-col (1m 2)
then show (UA† * UA) $$ (i,j) = 1m 2 $$ (i,j)
proof-
have i ∈ {0,1} ∧ j ∈ {0,1}
using a0 a1 mat-of-cols-list-def by auto
thus ?thesis
using mat-of-cols-list-def cos-sin-squared-add-cpx hermite-cnj-of-UA exp-sin-cos-squared-add[of
φA θA / 2]
by (auto simp add: set-2 algebra-simps)
qed
next
show dim-row (UA† * UA) = dim-row (1m 2)
using mat-of-cols-list-def by simp
next
show dim-col (UA† * UA) = dim-col (1m 2)
using mat-of-cols-list-def by simp
qed

lemma (in strategic-space-2p) UA-times-UA-cnj:
shows UA * (UA†) = 1m 2
proof
fix i j assume a0:i < dim-row (1m 2) and a1:j < dim-col (1m 2)
then show (UA * (UA†)) $$ (i,j) = 1m 2 $$ (i,j)
proof-
have i ∈ {0,1} ∧ j ∈ {0,1}
using a0 a1 mat-of-cols-list-def by auto
thus ?thesis
using mat-of-cols-list-def cos-sin-squared-add-cpx hermite-cnj-of-UA exp-sin-cos-squared-add[of
φA θA / 2]
by (auto simp add: set-2 algebra-simps)
qed
next
show dim-row (UA * (UA†)) = dim-row (1m 2)
using mat-of-cols-list-def by simp
next
show dim-col (UA * (UA†)) = dim-col (1m 2)
using mat-of-cols-list-def by simp
qed

lemma (in strategic-space-2p) UB-cnj-times-UB:
shows UB† * UB = 1m 2
proof
fix i j assume a0:i < dim-row (1m 2) and a1:j < dim-col (1m 2)
then show (UB† * UB) $$ (i,j) = 1m 2 $$ (i,j)
proof-
have i ∈ {0,1} ∧ j ∈ {0,1}
using a0 a1 mat-of-cols-list-def by auto
thus ?thesis

```

```

using mat-of-cols-list-def cos-sin-squared-add-cpx hermite-cnj-of- $U_B$  exp-sin-cos-squared-add[of
 $\varphi_B \vartheta_B / 2]$ 
  by (auto simp add: set-2 algebra-simps)
qed
next
show dim-row ( $U_B^\dagger * U_B$ ) = dim-row ( $1_m 2$ )
  using mat-of-cols-list-def by simp
next
show dim-col ( $U_B^\dagger * U_B$ ) = dim-col ( $1_m 2$ )
  using mat-of-cols-list-def by simp
qed

lemma (in strategic-space-2p)  $U_B$ -times- $U_B$ -cnj:
  shows  $U_B * (U_B^\dagger) = 1_m 2$ 
proof
  fix  $i j$  assume  $a0:i < \text{dim-row } (1_m 2)$  and  $a1:j < \text{dim-col } (1_m 2)$ 
  then show  $(U_B * (U_B^\dagger)) \$(i,j) = 1_m 2 \$ (i,j)$ 
  proof-
    have  $i \in \{0,1\} \wedge j \in \{0,1\}$ 
      using a0 a1 mat-of-cols-list-def by auto
    thus ?thesis
      using mat-of-cols-list-def cos-sin-squared-add-cpx hermite-cnj-of- $U_B$  exp-sin-cos-squared-add[of
 $\varphi_B \vartheta_B / 2]$ 
        by (auto simp add: set-2 algebra-simps)
    qed
  next
  show dim-row ( $U_B * (U_B^\dagger)$ ) = dim-row ( $1_m 2$ )
    using mat-of-cols-list-def by simp
  next
  show dim-col ( $U_B * (U_B^\dagger)$ ) = dim-col ( $1_m 2$ )
    using mat-of-cols-list-def by simp
  qed

lemma (in strategic-space-2p)  $U_A$ -is-gate:
  shows gate 1  $U_A$ 
proof
  show dim-row  $U_A = 2^{\wedge}1$ 
    using mat-of-cols-list-def by simp
  moreover show square-mat  $U_A$ 
    using mat-of-cols-list-def by simp
  ultimately show unitary  $U_A$ 
    using mat-of-cols-list-def unitary-def  $U_A$ -cnj-times- $U_A$   $U_A$ -times- $U_A$ -cnj by
    auto
  qed

lemma (in strategic-space-2p)  $U_B$ -is-gate:
  shows gate 1  $U_B$ 
proof
  show dim-row  $U_B = 2^{\wedge}1$ 

```

```

using mat-of-cols-list-def by simp
moreover show square-mat  $U_B$ 
  using mat-of-cols-list-def by simp
  ultimately show unitary  $U_B$ 
    using mat-of-cols-list-def unitary-def  $U_B\text{-cnj-times-}U_B$   $U_B\text{-times-}U_B\text{-cnj}$  by
auto
qed

lemma (in strategic-space-2p)  $U_{AB}\text{-is-gate}:$ 
  shows gate 2 ( $U_A \otimes U_B$ )
proof-
  have gate (1+1) ( $U_A \otimes U_B$ )
  using  $U_A\text{-is-gate } U_B\text{-is-gate tensor-gate}[of 1 ]$  by auto
  then show ?thesis
    by (auto simp add: numeral-2_eq_2)
qed

lemma (in strategic-space-2p)  $\psi_2\text{-two-is-state}:$ 
  shows state 2  $\psi_2$ 
proof-
  have state 2 (( $U_A \otimes U_B$ ) *  $\psi_1$ )
  using psi-one-is-state  $U_{AB}\text{-is-gate}$  by auto
  then show ?thesis
  using psi-two by simp
qed

lemma (in strategic-space-2p)  $J\text{-cnj-is-gate}:$ 
  shows gate 2 ( $J^\dagger$ )
proof
  show dim-row ( $J^\dagger$ ) = 22
  using mat-of-cols-list-def by simp
  moreover show square-mat ( $J^\dagger$ )
    using mat-of-cols-list-def by simp
  moreover have ( $J^\dagger$ )† =  $J$ 
    using dagger-of-dagger-is-id by auto
  ultimately show unitary ( $J^\dagger$ )
  using mat-of-cols-list-def unitary-def  $J\text{-cnj-times-}J$   $J\text{-times-}J\text{-cnj}$  by auto
qed

lemma (in strategic-space-2p)  $\psi_f\text{-f-is-state}:$ 
  shows state 2  $\psi_f$ 
proof-
  have state 2 (( $J^\dagger$ ) *  $\psi_2$ )
  using psi-two-is-state  $J\text{-cnj-is-gate}$  by auto
  then show ?thesis
  using psi-f by simp
qed

```

lemma (in strategic-space-2p) equation-one:
shows $(J^\dagger) * ((U_A \otimes U_B) * (J * |\text{unit-vec } 4\ 0\rangle)) = \psi_f$
using $\psi\text{-one}$ $\psi\text{-two}$ $\psi\text{-f}$ **by** auto

abbreviation (in strategic-space-2p) prob00 :: complex Matrix.mat \Rightarrow **real where**
 $\text{prob00 } v \equiv \text{if state } 2\ v \text{ then } (\text{cmod } (v \$\$ (0,0)))^2 \text{ else undefined}$

abbreviation (in strategic-space-2p) prob01 :: complex Matrix.mat \Rightarrow **real where**
 $\text{prob01 } v \equiv \text{if state } 2\ v \text{ then } (\text{cmod } (v \$\$ (1,0)))^2 \text{ else undefined}$

abbreviation (in strategic-space-2p) prob10 :: complex Matrix.mat \Rightarrow **real where**
 $\text{prob10 } v \equiv \text{if state } 2\ v \text{ then } (\text{cmod } (v \$\$ (2,0)))^2 \text{ else undefined}$

abbreviation (in strategic-space-2p) prob11 :: complex Matrix.mat \Rightarrow **real where**
 $\text{prob11 } v \equiv \text{if state } 2\ v \text{ then } (\text{cmod } (v \$\$ (3,0)))^2 \text{ else undefined}$

definition (in strategic-space-2p) alice-payoff :: real where
 $\text{alice-payoff} \equiv 3 * (\text{prob00 } \psi_f) + 1 * (\text{prob11 } \psi_f) + 0 * (\text{prob01 } \psi_f) + 5 * (\text{prob10 } \psi_f)$

definition (in strategic-space-2p) bob-payoff :: real where
 $\text{bob-payoff} \equiv 3 * (\text{prob00 } \psi_f) + 1 * (\text{prob11 } \psi_f) + 5 * (\text{prob01 } \psi_f) + 0 * (\text{prob10 } \psi_f)$

definition (in strategic-space-2p) is-nash-eq :: bool where
 $\text{is-nash-eq} \equiv$
 $(\forall tA\ pA. \text{strategic-space-2p} \gamma tA\ pA\ \vartheta_B\ \varphi_B \longrightarrow$
 $\quad \text{alice-payoff} \geq \text{strategic-space-2p.alice-payoff} \gamma tA\ pA\ \vartheta_B\ \varphi_B)$
 \wedge
 $(\forall tB\ pB. \text{strategic-space-2p} \gamma \vartheta_A\ \varphi_A\ tB\ pB \longrightarrow$
 $\quad \text{bob-payoff} \geq \text{strategic-space-2p.bob-payoff} \gamma \vartheta_A\ \varphi_A\ tB\ pB)$

definition (in strategic-space-2p) is-pareto-opt :: bool where
 $\text{is-pareto-opt} \equiv \forall tA\ pA\ tB\ pB. \text{strategic-space-2p} \gamma tA\ pA\ tB\ pB \longrightarrow$
 $((\text{strategic-space-2p.alice-payoff} \gamma tA\ pA\ tB\ pB > \text{alice-payoff} \longrightarrow$
 $\quad \text{strategic-space-2p.bob-payoff} \gamma tA\ pA\ tB\ pB < \text{bob-payoff}) \wedge$
 $\quad (\text{strategic-space-2p.bob-payoff} \gamma tA\ pA\ tB\ pB > \text{bob-payoff} \longrightarrow$
 $\quad \text{strategic-space-2p.alice-payoff} \gamma tA\ pA\ tB\ pB < \text{alice-payoff}))$

lemma (in strategic-space-2p) sum-of-prob:
fixes $v :: \text{complex Matrix.mat}$
assumes $\text{state } 2\ v$
shows $(\text{prob00 } v) + (\text{prob11 } v) + (\text{prob01 } v) + (\text{prob10 } v) = 1$
proof-
have $(\text{prob00 } v) + (\text{prob11 } v) + (\text{prob01 } v) + (\text{prob10 } v) =$
 $(\text{cmod } (v \$\$ (0,0)))^2 + (\text{cmod } (v \$\$ (1,0)))^2 + (\text{cmod } (v \$\$ (2,0)))^2 +$
 $(\text{cmod } (v \$\$ (3,0)))^2$
using assms **by** auto
then show $?thesis$

```

using state-def assms cpx-vec-length-def by (auto simp add: set-4)
qed

lemma (in strategic-space-2p) sum-payoff-le-6:
  fixes tA pA tB pB :: real
  shows alice-payoff + bob-payoff ≤ 6
proof-
  have alice-payoff + bob-payoff = 6 * (prob00 ψ_f) + 2 * (prob11 ψ_f) + 5 *
  (prob01 ψ_f) + 5 * (prob10 ψ_f)
    using alice-payoff-def bob-payoff-def psi-f-is-state by auto
  then have alice-payoff + bob-payoff ≤ 6 * ((prob00 ψ_f) + (prob11 ψ_f) +
  (prob01 ψ_f) + (prob10 ψ_f))
    using psi-f-is-state by (auto simp add: algebra-simps)
  moreover have (prob00 ψ_f) + (prob11 ψ_f) + (prob01 ψ_f) + (prob10 ψ_f) = 1
    using sum-of-prob[of ψ_f] psi-f-is-state by auto
  ultimately show ?thesis
    by auto
qed

lemma (in strategic-space-2p) coop-is-pareto-opt:
  assumes alice-payoff = 3 ∧ bob-payoff = 3
  shows is-pareto-opt
  using is-pareto-opt-def strategic-space-2p.sum-payoff-le-6 assms by fastforce

```

13.2 The Separable Case

```

lemma (in strategic-space-2p) separable-case-CC:
  assumes γ = 0
  shows φ_A = 0 ∧ θ_A = 0 ∧ φ_B = 0 ∧ θ_B = 0 → alice-payoff = 3 ∧ bob-payoff
= 3
  using alice-payoff-def bob-payoff-def cos-sin-squared-add-cpx psi-f-is-state by auto

lemma (in strategic-space-2p) separable-case-DD:
  assumes γ = 0
  shows φ_A = 0 ∧ θ_A = pi ∧ φ_B = 0 ∧ θ_B = pi → alice-payoff = 1 ∧ bob-payoff
= 1
  using alice-payoff-def bob-payoff-def cos-sin-squared-add-cpx psi-f-is-state by auto

lemma (in strategic-space-2p) separable-case-DC:
  assumes γ = 0
  shows φ_A = 0 ∧ θ_A = pi ∧ φ_B = 0 ∧ θ_B = 0 → alice-payoff = 5 ∧ bob-payoff
= 0
  using alice-payoff-def bob-payoff-def sin-cos-squared-add-cpx psi-f-is-state by auto

lemma (in strategic-space-2p) separable-alice-payoff-D_B:
  assumes γ = 0 and φ_B = 0 ∧ θ_B = pi
  shows alice-payoff ≤ 1
  using alice-payoff-def assms sin-squared-le-one psi-f-is-state by auto

```

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

assumes $\gamma = 0$ and $\varphi_A = 0 \wedge \vartheta_A = pi$
shows $bob\text{-payoff} \leq 1$
using $bob\text{-payoff}\text{-def assms sin-squared-le-one psi-f-is-state}$ by auto

lemma (in strategic-space-2p) separable-case-DD-alice-opt:

assumes $\gamma = 0$ and $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi$
shows $\bigwedge tA pA. strategic\text{-space-2p } \gamma tA pA \vartheta_B \varphi_B \longrightarrow strategic\text{-space-2p}.alice\text{-payoff}$
 $\gamma tA pA \vartheta_B \varphi_B \leq alice\text{-payoff}$

proof

fix $tA pA$ assume $strategic\text{-space-2p } \gamma tA pA \vartheta_B \varphi_B$
then show $strategic\text{-space-2p}.alice\text{-payoff } \gamma tA pA \vartheta_B \varphi_B \leq alice\text{-payoff}$
using $separable\text{-case-DD strategic-space-2p}.separable\text{-alice-payoff-}D_B$ assms by
auto

qed

lemma (in strategic-space-2p) separable-case-DD-bob-opt:

assumes $\gamma = 0$ and $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi$
shows $\bigwedge tB pB. strategic\text{-space-2p } \gamma \vartheta_A \varphi_A tB pB \longrightarrow strategic\text{-space-2p}.bob\text{-payoff}$
 $\gamma \vartheta_A \varphi_A tB pB \leq bob\text{-payoff}$

proof

fix $tB pB$ assume $strategic\text{-space-2p } \gamma \vartheta_A \varphi_A tB pB$
then show $strategic\text{-space-2p}.bob\text{-payoff } \gamma \vartheta_A \varphi_A tB pB \leq bob\text{-payoff}$
using $separable\text{-case-DD strategic-space-2p}.separable\text{-bob-payoff-}D_A$ assms by
auto

qed

lemma (in strategic-space-2p) separable-case-DD-is-nash-eq:

assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = pi \wedge \varphi_B = 0 \wedge \vartheta_B = pi \longrightarrow is\text{-nash-eq}$
using $is\text{-nash-eq}\text{-def separable-case-DD-alice-opt separable-case-DD-bob-opt assms}$
by auto

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

assumes $\gamma = 0$
shows $\varphi_A = 0 \wedge \vartheta_A = 0 \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow \neg is\text{-nash-eq}$

proof

assume $asm:\varphi_A = 0 \wedge \vartheta_A = 0 \wedge \varphi_B = 0 \wedge \vartheta_B = 0$
then have $f0:strategic\text{-space-2p } \gamma pi 0 \vartheta_B \varphi_B$
using $strategic\text{-space-2p}\text{-def strategic-space-2p-axioms}\text{-def prisoner}\text{-def asm}$ by
(simp add: assms)

then have $strategic\text{-space-2p}.alice\text{-payoff } \gamma pi 0 \vartheta_B \varphi_B = 5$
using $strategic\text{-space-2p}.separable\text{-case-DC}$ assms asm by blast

moreover have $alice\text{-payoff} = 3$
using $separable\text{-case-CC}$ assms asm by blast

ultimately have $strategic\text{-space-2p } \gamma pi 0 \vartheta_B \varphi_B \wedge strategic\text{-space-2p}.alice\text{-payoff}$
 $\gamma pi 0 \vartheta_B \varphi_B > alice\text{-payoff}$

```

using f0 by simp
then show  $\neg$ is-nash-eq
  using is-nash-eq-def by fastforce
qed

lemma (in strategic-space-2p) separable-case-CC-is-pareto-optimal:
  assumes  $\gamma = 0$ 
  shows  $\varphi_A = 0 \wedge \vartheta_A = 0 \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow$  is-pareto-opt
  using coop-is-pareto-opt separable-case-CC assms by auto

```

13.3 The Maximally Entangled Case

```

lemma exp-to-sin:
  fixes x:: real
  shows  $\exp(i * x) - \exp(-(i * x)) = 2 * i * (\sin x)$ 
  using exp-of-real exp-of-real-inv by simp

lemma exp-to-cos:
  fixes x:: real
  shows  $\exp(i * x) + \exp(-(i * x)) = 2 * (\cos x)$ 
  using exp-of-real exp-of-real-inv by simp

lemma cmod-real-prod-squared:
  fixes x y:: real
  shows  $(\text{cmod}(\text{complex-of-real } x * \text{complex-of-real } y))^2 = x^2 * y^2$ 
  by (simp add: norm-mult power-mult-distrib)

lemma quantum-payoff-simp:
  fixes x y:: real
  shows  $3 * (\text{cmod}(\text{complex-of-real } (\sin x) * \text{complex-of-real } (\cos y)))^2 +$ 
     $(\text{cmod}(\text{complex-of-real } (\cos x) * \text{complex-of-real } (\cos y)))^2 =$ 
     $2 * (\sin x)^2 * (\cos y)^2 + (\cos y)^2$ 
proof-
  have  $3 * (\sin x)^2 * (\cos y)^2 + (\cos x)^2 * (\cos y)^2 =$ 
     $(2 * (\sin x)^2 * (\cos y)^2) + ((\sin x)^2 + (\cos x)^2) * (\cos y)^2$ 
  by (auto simp add: algebra_simps simp del: sin-cos-squared-add2)
  then show ?thesis
  by (simp add: cmod-real-prod-squared power-mult-distrib)
qed

lemma quantum-payoff-le-3:
  fixes x y:: real
  shows  $2 * (\sin x)^2 * (\cos y)^2 + (\cos y)^2 \leq 3$ 
proof-
  have  $(\sin x)^2 * (\cos y)^2 \leq 1$  by (simp add: sin-squared-le-one cos-squared-le-one mult-le-one)
  then have  $2 * (\sin x)^2 * (\cos y)^2 \leq 2$  by simp
  moreover have  $(\cos y)^2 \leq 1$ 
  using cos-squared-le-one[of y] by simp

```

```

ultimately show ?thesis by simp
qed

lemma sqrt-two-squared-cpx: complex-of-real (sqrt 2) * complex-of-real (sqrt 2) =
2
  by (metis mult-2-right numeral-Bit0 of-real-mult of-real-numeral real-sqrt-four
real-sqrt-mult)

lemma hidden-sqrt-two-squared-cpx: complex-of-real (sqrt 2) * (complex-of-real
(sqrt 2) * x) / 4 = x/2
  using sqrt-two-squared-cpx by (auto simp add: algebra-simps)

lemma (in strategic-space-2p) max-entangled-DD:
  assumes γ = pi/2
  shows φA = 0 ∧ θA = pi ∧ φB = 0 ∧ θB = pi → alice-payoff = 1 ∧ bob-payoff
= 1
  using alice-payoff-def bob-payoff-def cos-sin-squared-add-cpx psi-f-is-state
  by auto

lemma (in strategic-space-2p) max-entangled-QQ:
  assumes γ = pi/2
  shows φA = pi/2 ∧ θA = 0 ∧ φB = pi/2 ∧ θB = 0 → alice-payoff = 3 ∧
bob-payoff = 3
  using alice-payoff-def bob-payoff-def sin-cos-squared-add-cpx exp-of-half-pi exp-of-minus-half-pi
psi-f-is-state
  by auto

lemma (in strategic-space-2p) max-entangled-QD:
  assumes γ = pi/2
  shows φA = pi/2 ∧ θA = 0 ∧ φB = 0 ∧ θB = pi → alice-payoff = 5 ∧
bob-payoff = 0
  using alice-payoff-def bob-payoff-def cos-sin-squared-add-cpx exp-of-half-pi exp-of-minus-half-pi
psi-f-is-state sqrt-two-squared-cpx
  by (auto simp add: assms algebra-simps sin-45 cos-45)

lemma (in strategic-space-2p) max-entangled-alice-payoff-QB:
  assumes γ = pi/2
  shows φB = pi/2 ∧ θB = 0 → alice-payoff ≤ 3
  proof
    assume asm:φB = pi/2 ∧ θB = 0
    have ψf $$ (0,0) = -(sin φA) * (cos (θA/2))
    proof-
      have ψf $$ (0,0) = i * (sqrt 2/2) * (sqrt 2/2) * (cos (θA/2)) * exp (i * φA)
    +

```

```

          i * (sqrt 2/2) * (sqrt 2/2) * (cos (vartheta_A/2)) * -exp(-(i * varphi_A))
  using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi
  by (auto simp add: sin-of-quarter-pi[of gamma] cos-of-quarter-pi[of gamma] algebra-simps)
  then have psi_f $$ (0,0) = i * (sqrt 2/2) * (sqrt 2/2) * (cos (vartheta_A/2)) * (exp (i * varphi_A) - exp(-(i * varphi_A)))
    by (auto simp add: algebra-simps)
  then have psi_f $$ (0,0) = i * (cos (vartheta_A/2)) * (1/2) * (exp (i * varphi_A) - exp(-(i * varphi_A)))
    using sqrt-two-squared-cpx by (auto simp add: algebra-simps)
    then show ?thesis
      using exp-to-sin by (simp add: algebra-simps)
qed
moreover have psi_f $$ (1,0) = sin (vartheta_A/2)
using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi sqrt-two-squared-cpx
  by (auto simp add: sin-of-quarter-pi[of gamma] cos-of-quarter-pi[of gamma])
moreover have psi_f $$ (2,0) = 0
using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi sqrt-two-squared-cpx
  by (auto simp add: sin-of-quarter-pi[of gamma] cos-of-quarter-pi[of gamma] algebra-simps)
moreover have psi_f $$ (3,0) = (cos varphi_A) * (cos (vartheta_A/2))
proof-
  have psi_f $$ (3,0) = exp (i * varphi_A) * (cos (vartheta_A/2)) * (sqrt 2/2) * (sqrt 2/2) +
    exp(-(i * varphi_A)) * (cos (vartheta_A/2)) * (sqrt 2/2) * (sqrt 2/2)
    using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi
    by (auto simp add: sin-of-quarter-pi[of gamma] cos-of-quarter-pi[of gamma] algebra-simps)
    then have psi_f $$ (3,0) = (exp (i * varphi_A) + exp(-(i * varphi_A))) * (cos (vartheta_A/2)) *
      (sqrt 2/2) * (sqrt 2/2)
      by (auto simp add: algebra-simps)
    then have psi_f $$ (3,0) = (exp (i * varphi_A) + exp(-(i * varphi_A))) * (cos (vartheta_A/2)) *
      (1/2)
      using sqrt-two-squared-cpx hidden-sqrt-two-squared-cpx by (auto simp add:
        algebra-simps)
      then show ?thesis
        using exp-to-cos by (simp add: algebra-simps)
qed
ultimately show alice-payoff ≤ 3
using alice-payoff-def psi-f-is-state quantum-payoff-simp quantum-payoff-le-3
by auto
qed

```

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

```

assumes gamma = pi/2
shows varphi_A = pi/2 ∧ vartheta_A = 0 → bob-payoff ≤ 3
proof
  assume asm:varphi_A = pi/2 ∧ vartheta_A = 0
  have psi_f $$ (0,0) = -(sin varphi_B) * (cos (vartheta_B/2))
  proof-
    have psi_f $$ (0,0) = i * (sqrt 2/2) * (sqrt 2/2) * (cos (vartheta_B/2)) * exp (i * varphi_B)
  +

```

```

          i * (sqrt 2/2) * (sqrt 2/2) * (cos (θ_B/2)) * -exp(-(i * φ_B))
  using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi
  by (auto simp add: sin-of-quarter-pi[of γ] cos-of-quarter-pi[of γ] algebra-simps)
  then have ψ_f $$ (0,0) = i * (sqrt 2/2) * (sqrt 2/2) * (cos (θ_B/2)) * (exp (i
* φ_B) - exp(-(i * φ_B)))
    by (auto simp add: algebra-simps)
  then have ψ_f $$ (0,0) = i * (cos (θ_B/2)) * (1/2) * (exp (i * φ_B) - exp (-(i
* φ_B)))
    using sqrt-two-squared-cpx by (auto simp add: algebra-simps)
    then show ?thesis
      using exp-to-sin by (simp add: algebra-simps)
qed
moreover have ψ_f $$ (1,0) = 0
using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi sqrt-two-squared-cpx
by (auto simp add: sin-of-quarter-pi[of γ] cos-of-quarter-pi[of γ] algebra-simps)
moreover have ψ_f $$ (2,0) = sin (θ_B/2)
using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi sqrt-two-squared-cpx
by (auto simp add: sin-of-quarter-pi[of γ] cos-of-quarter-pi[of γ] algebra-simps)
moreover have ψ_f $$ (3,0) = (cos φ_B) * (cos (θ_B/2))
proof-
  have ψ_f $$ (3,0) = exp (i * φ_B) * (cos (θ_B/2)) * (sqrt 2/2) * (sqrt 2/2) +
    exp (-(i * φ_B)) * (cos (θ_B/2)) * (sqrt 2/2) * (sqrt 2/2)
    using mat-of-cols-list-def asm assms exp-of-half-pi exp-of-minus-half-pi
    by (auto simp add: sin-of-quarter-pi[of γ] cos-of-quarter-pi[of γ] algebra-simps)
    then have ψ_f $$ (3,0) = (exp (i * φ_B) + exp (-(i * φ_B))) * (cos (θ_B/2)) *
  (sqrt 2/2) * (sqrt 2/2)
    by (auto simp add: algebra-simps)
    then have ψ_f $$ (3,0) = (exp (i * φ_B) + exp (-(i * φ_B))) * (cos (θ_B/2)) *
  (1/2)
    using sqrt-two-squared-cpx hidden-sqrt-two-squared-cpx by (auto simp add:
algebra-simps)
    then show ?thesis
      using exp-to-cos by (simp add: algebra-simps)
qed
ultimately show bob-payoff ≤ 3
using bob-payoff-def psi-f-is-state quantum-payoff-simp quantum-payoff-le-3
by auto
qed

lemma (in strategic-space-2p) max-entangled-DD-is-not-nash-eq:
assumes γ = pi/2
shows φ_A = 0 ∧ θ_A = pi ∧ φ_B = 0 ∧ θ_B = pi → ¬is-nash-eq
proof
assume asm:φ_A = 0 ∧ θ_A = pi ∧ φ_B = 0 ∧ θ_B = pi
then have f0:strategic-space-2p γ 0 (pi/2) θ_B φ_B
  using strategic-space-2p-def strategic-space-2p-axioms-def prisoner-def asm by
(simp add: assms)
then have strategic-space-2p.alice-payoff γ 0 (pi/2) θ_B φ_B = 5
  using strategic-space-2p.max-entangled-QD assms asm by blast

```

```

moreover have alice-payoff = 1
  using max-entangled-DD assms asm by blast
ultimately have strategic-space-2p γ 0 (pi/2) θ_B φ_B ∧ strategic-space-2p.alice-payoff
γ 0 (pi/2) θ_B φ_B > alice-payoff
  using f0 by simp
then show ¬is-nash-eq
  using is-nash-eq-def by fastforce
qed

lemma (in strategic-space-2p) max-entangled-alice-opt:
  assumes γ = pi/2 and φ_A = pi/2 ∧ θ_A = 0 ∧ φ_B = pi/2 ∧ θ_B = 0
  shows ∨ tA pA. strategic-space-2p γ tA pA θ_B φ_B → strategic-space-2p.alice-payoff
γ tA pA θ_B φ_B ≤ alice-payoff
proof
fix tA pA assume strategic-space-2p γ tA pA θ_B φ_B
then have strategic-space-2p.alice-payoff γ tA pA θ_B φ_B ≤ 3
  using strategic-space-2p.max-entangled-alice-payoff-Q_B assms by blast
moreover have alice-payoff = 3
  using max-entangled-QQ assms by blast
ultimately show strategic-space-2p.alice-payoff γ tA pA θ_B φ_B ≤ alice-payoff
  by simp
qed

lemma (in strategic-space-2p) max-entangled-bob-opt:
  assumes γ = pi/2 and φ_A = pi/2 ∧ θ_A = 0 ∧ φ_B = pi/2 ∧ θ_B = 0
  shows ∨ tB pB. strategic-space-2p γ θ_A φ_A tB pB → strategic-space-2p.bob-payoff
γ θ_A φ_A tB pB ≤ bob-payoff
proof
fix tB pB assume strategic-space-2p γ θ_A φ_A tB pB
then have strategic-space-2p.bob-payoff γ θ_A φ_A tB pB ≤ 3
  using strategic-space-2p.max-entangled-bob-payoff-Q_A assms by blast
moreover have bob-payoff = 3
  using max-entangled-QQ assms by blast
ultimately show strategic-space-2p.bob-payoff γ θ_A φ_A tB pB ≤ bob-payoff
  by simp
qed

lemma (in strategic-space-2p) max-entangled-QQ-is-nash-eq:
  assumes γ = pi/2
  shows φ_A = pi/2 ∧ θ_A = 0 ∧ φ_B = pi/2 ∧ θ_B = 0 → is-nash-eq
  using max-entangled-alice-opt max-entangled-bob-opt is-nash-eq-def assms by
blast

lemma (in strategic-space-2p) max-entangled-QQ-is-pareto-optimal:
  assumes γ = pi/2
  shows φ_A = pi/2 ∧ θ_A = 0 ∧ φ_B = pi/2 ∧ θ_B = 0 → is-pareto-opt
  using coop-is-pareto-opt max-entangled-QQ assms by blast

```

13.4 The Unfair Strategy Case

```

lemma half-sqrt-two-squared:  $2 * (\sqrt{2} / 2)^2 = 1$ 
  by (auto simp add: power2-eq-square)

lemma (in strategic-space-2p) max-entangled-MD:
  assumes  $\gamma = \pi/2$ 
  shows  $\varphi_A = \pi/2 \wedge \vartheta_A = \pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = \pi \longrightarrow \text{alice-payoff} = 3 \wedge \text{bob-payoff} = 1/2$ 
  proof
    assume asm: $\varphi_A = \pi/2 \wedge \vartheta_A = \pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = \pi$ 
    show  $\text{alice-payoff} = 3 \wedge \text{bob-payoff} = 1/2$ 
    using alice-payoff-def bob-payoff-def sqrt-two-squared-cpx half-sqrt-two-squared
      exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2] psi-f-is-state
    by (auto simp add: asm assms sin-45 cos-45 algebra-simps)
  qed

lemma (in strategic-space-2p) max-entangled-MC:
  assumes  $\gamma = \pi/2$ 
  shows  $\varphi_A = \pi/2 \wedge \vartheta_A = \pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = 0 \longrightarrow \text{alice-payoff} = 3 \wedge \text{bob-payoff} = 1/2$ 
  proof
    assume asm: $\varphi_A = \pi/2 \wedge \vartheta_A = \pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = 0$ 
    show  $\text{alice-payoff} = 3 \wedge \text{bob-payoff} = 1/2$ 
    using alice-payoff-def bob-payoff-def sqrt-two-squared-cpx half-sqrt-two-squared
      exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2] psi-f-is-state
    by (auto simp add: asm assms sin-45 cos-45 algebra-simps)
  qed

lemma (in strategic-space-2p) max-entangled-MH:
  assumes  $\gamma = \pi/2$ 
  shows  $\varphi_A = \pi/2 \wedge \vartheta_A = \pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = \pi/2 \longrightarrow \text{alice-payoff} = 1 \wedge \text{bob-payoff} = 1$ 
  proof
    assume asm: $\varphi_A = \pi/2 \wedge \vartheta_A = \pi/2 \wedge \varphi_B = 0 \wedge \vartheta_B = \pi/2$ 
    show  $\text{alice-payoff} = 1 \wedge \text{bob-payoff} = 1$ 
    using alice-payoff-def bob-payoff-def sqrt-two-squared-cpx half-sqrt-two-squared
      exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2] psi-f-is-state
    by (auto simp add: asm assms sin-45 cos-45 algebra-simps)
  qed

```

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

lemma (in strategic-space-2p) M-correct:

```

assumes  $\varphi_A = pi/2 \wedge \vartheta_A = pi/2$ 
shows  $U_A = M$ 
proof
  show dim-row  $U_A = \text{dim-row } M$  using mat-of-cols-list-def by simp
  show dim-col  $U_A = \text{dim-col } M$  using mat-of-cols-list-def by simp
  fix  $i j$  assume  $a0:i < \text{dim-row } M$  and  $a1:j < \text{dim-col } M$ 
  then show  $U_A \$\$ (i,j) = M \$\$ (i,j)$ 
proof-
  have  $i \in \{0,1\} \wedge j \in \{0,1\}$ 
    using a0 a1 mat-of-cols-list-def by auto
  thus ?thesis
    using mat-of-cols-list-def exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2]
    by (auto simp add: assms sin-45 cos-45)
qed
qed

lemma hidden-sqrt-two-squared-cpx2:
fixes  $x y :: \text{complex}$ 
shows  $(\sqrt{2}) * ((\sqrt{2}) * (x * y)) / 2 = x * y$ 
using sqrt-two-squared-cpx by auto

lemma (in strategic-space-2p) unfair-strategy-no-benefit:
assumes  $\gamma = pi/2$ 
shows  $\varphi_A = pi/2 \wedge \varphi_B = 0 \wedge \vartheta_A = \vartheta_B \rightarrow \text{alice-payoff} = 1 \wedge \text{bob-payoff} = 1$ 
proof
  assume asm: $\varphi_A = pi/2 \wedge \varphi_B = 0 \wedge \vartheta_A = \vartheta_B$ 
  have  $\psi_f \$\$ (0,0) = 0$ 
    using exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2]
    by (auto simp add: asm assms sin-45 cos-45 algebra-simps)
  moreover have  $\psi_f \$\$ (1,0) = 0$ 
    using exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2]
    by (auto simp add: asm assms sin-45 cos-45 algebra-simps)
  moreover have  $\psi_f \$\$ (2,0) = 0$ 
    using exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2]
    by (auto simp add: asm assms sin-45 cos-45 hidden-sqrt-two-squared-cpx2 algebra-simps)
  moreover have  $\psi_f \$\$ (3,0) = 1$ 
    using exp-of-half-pi[of pi/2] exp-of-minus-half-pi[of pi/2] cos-sin-squared-add-cpx
    by (auto simp add: asm assms sin-45 cos-45 hidden-sqrt-two-squared-cpx2 algebra-simps)
  ultimately show alice-payoff = 1  $\wedge$  bob-payoff = 1
    using alice-payoff-def bob-payoff-def psi-f-is-state
    by auto
qed

end

```

14 Acknowledgements

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

References

- [1] J. Boender, F. Kammüller, and R. Nagarajan. Formalization of quantum protocols using coq. In *QPL*, 2015.
- [2] J. Eisert, M. Wilkens, and M. Lewenstein. Quantum Games and Quan-
tum Strategies. *Physical Review Letters*, 83:3077–3080, Oct. 1999.
- [3] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.