Quantum-Fourier-Transform

pablo

November 5, 2024

Contents

theory *QFT*

imports
Isabelle-Marries-Dirac.Deutsch
begin

1 Some useful lemmas

lemma *gate-carrier-mat*[*simp*]:
assumes *gate n U*
shows $U \in$ *carrier-mat* $(2\hat{\ }n)$ $(2\hat{\ }n)$
proof

show *dim-row U* = *2* ^ *n* **using** *gate-def assms* **by** *auto*
next
show *dim-col U* = *2* ^ *n* **using** *gate-def assms* **by** *auto*
qed

lemma *state-carrier-mat*[*simp*]:
assumes *state n* ψ
shows $\psi \in$ *carrier-mat* (*2* ^ *n*) *1*
proof
show *dim-row* ψ = *2* ^ *n* **using** *state-def assms* **by** *auto*
next
show *dim-col* ψ = *1* **using** *state-def assms* **by** *auto*
qed

lemma *state-basis-carrier-mat*[*simp*]:
$|$*state-basis n j*$\rangle \in$ *carrier-mat* (*2* ^ *n*) *1*
by (*simp add*: *ket-vec-def state-basis-def*)

lemma *left-tensor-id*[*simp*]:
assumes $A \in$ *carrier-mat nr nc*
shows (1_m *1*) $\bigotimes$ $A = A$
by *auto*

lemma *right-tensor-id*[*simp*]:
assumes $A \in$ *carrier-mat nr nc*
shows $A \bigotimes$ (1_m *1*) $= A$
by *auto*

lemma *tensor-carrier-mat*[*simp*]:
assumes $A \in$ *carrier-mat ra ca*
and $B \in$ *carrier-mat rb cb*
shows $A \bigotimes B \in$ *carrier-mat* (*ra*∗*rb*) (*ca*∗*cb*)
proof
show *dim-row* ($A \bigotimes B$) = *ra* ∗ *rb* **using** *dim-row-tensor-mat assms* **by** *auto*
show *dim-col* ($A \bigotimes B$) = *ca* ∗ *cb* **using** *dim-col-tensor-mat assms* **by** *auto*
qed

lemma *smult-tensor*[*simp*]:
assumes *dim-col A* > *0* **and** *dim-col B* > *0*
shows ($a \cdot_m A$) $\bigotimes$ ($b \cdot_m B$) = (a∗b) $\cdot_m$ ($A \bigotimes B$)
proof
fix *i j*::*nat*
assume *ai*:*i* < *dim-row* (a ∗ $b \cdot_m$ ($A \bigotimes B$)) **and** *aj*:*j* < *dim-col* (a ∗ $b \cdot_m$ (A $\bigotimes$ B))
show ($a \cdot_m A \bigotimes b \cdot_m B$) $$ (*i*, *j*) = ((a ∗ b) $\cdot_m$ ($A \bigotimes B$)) $$ (*i*, *j*)
proof −
define *rA cA rB cB* **where** *rA* = *dim-row A* **and** *cA* = *dim-col A* **and** *rB* = *dim-row B*
and *cB* = *dim-col B*

have $(a \cdot_m A \bigotimes b \cdot_m B)\$\$(i, j) = (a \cdot_m A)\$\$(i\ div\ rB, j\ div\ cB) * (b \cdot_m B)\$\$(i\ mod\ rB, j\ mod\ cB)$
proof (*rule index-tensor-mat*)
show *dim-row* $(a \cdot_m A) = rA$ **using** *rA-def* **by** *simp*
show *dim-col* $(a \cdot_m A) = cA$ **using** *cA-def* **by** *simp*
show *dim-row* $(b \cdot_m B) = rB$ **using** *rB-def* **by** *simp*
show *dim-col* $(b \cdot_m B) = cB$ **using** *cB-def* **by** *simp*
show $i < rA * rB$ **using** *ai rA-def rB-def smult-carrier-mat tensor-carrier-mat* **by** *auto*
show $j < cA * cB$ **using** *aj cA-def cB-def smult-carrier-mat tensor-carrier-mat* **by** *auto*
show $0 < cA$ **using** *cA-def assms(1)* **by** *simp*
show $0 < cB$ **using** *cB-def assms(2)* **by** *simp*
qed
also have $\ldots = a*A\$\$(i\ div\ rB, j\ div\ cB)*b*B\$\$(i\ mod\ rB, j\ mod\ cB)$
using *index-smult-mat* **by** (*smt* (*verit*) *Euclidean-Rings.div-eq-0-iff ab-semigroup-mult-class.mult-ac(1) ai aj cB-def dim-col-tensor-mat dim-row-tensor-mat*

less-mult-imp-div-less mod-less-divisor mult-0-right not-gr0 rB-def)
also have $\ldots = (a*b)*(A\$\$(i\ div\ rB, j\ div\ cB)*B\$\$(i\ mod\ rB, j\ mod\ cB))$ **by** *auto*
also have $\ldots = (a*b)*((A \bigotimes B)\ \$\$\ (i,j))$
proof −
have $(A \bigotimes B)\ \$\$\ (i,j) = A\$\$(i\ div\ rB, j\ div\ cB)*B\$\$(i\ mod\ rB, j\ mod\ cB)$
using *index-tensor-mat rA-def cA-def rB-def cB-def ai aj smult-carrier-mat tensor-carrier-mat assms* **by** *auto*
thus *?thesis* **by** *simp*
qed
also have $\ldots = ((a*b) \cdot_m (A \bigotimes B))\ \$\$\ (i,j)$ **using** *index-smult-mat(1)*
by (*metis ai aj index-smult-mat(2) index-smult-mat(3)*)
finally show *?thesis* **by** *this*
qed
next
show *dim-row* $(a \cdot_m A \bigotimes b \cdot_m B) =$ *dim-row* $(a * b \cdot_m (A \bigotimes B))$ **by** *simp*
next
show *dim-col* $(a \cdot_m A \bigotimes b \cdot_m B) =$ *dim-col* $(a * b \cdot_m (A \bigotimes B))$ **by** *simp*
qed

lemma *smult-tensor1* [*simp*]:
assumes *dim-col* $A > 0$ **and** *dim-col* $B > 0$
shows $a \cdot_m (A \bigotimes B) = (a \cdot_m A) \bigotimes B$
proof −
have $a \cdot_m (A \bigotimes B) = (a*1) \cdot_m (A \bigotimes B)$ **by** *auto*
also have $\ldots = (a \cdot_m A) \bigotimes (1 \cdot_m B)$ **using** *assms smult-tensor* **by** *simp*
also have $\ldots = (a \cdot_m A) \bigotimes B$
by (*metis eq-matI index-smult-mat(1) index-smult-mat(2) index-smult-mat(3) mult-cancel-right1*)
finally show *?thesis* **by** *this*
qed

lemma *set-list*:
set [*m..<n*] = {*m..<n*}
by *auto*

lemma *sumof2*:
($\sum k<(2::nat).\ f\ k) = f\ 0 + f\ 1$
by (*metis One-nat-def Suc-1 add.left-neutral lessThan-0 sum.empty sum.lessThan-Suc*)

lemma *sumof4*:
($\sum k<(4::nat).\ f\ k) = f\ 0 + f\ 1 + f\ 2 + f\ 3$
proof −
have ($\sum k<(4::nat).\ f\ k) = sum\ f\ (set\ [0..<4])$ **using** *set-list atLeast-upt* **by** *presburger*
also have ... $= f\ 0 + (f\ (Suc\ 0) + (f\ 2 + f\ 3))$ **by** *simp*
also have ... $= f\ 0 + f\ 1 + f\ 2 + f\ 3$ **by** (*simp add*: *add.commute add.left-commute*)
finally show *?thesis* **by** *this*
qed

2 The operator R_k

definition *R*:: *nat* ⇒ *complex Matrix.mat* **where**
R k = *mat-of-cols-list 2* [[*1*, *0*],
[*0*, *exp*(*2*∗*pi*∗i/*2*^*k*)]]

3 The SWAP gate:

definition *SWAP*:: *complex Matrix.mat* **where**
SWAP ≡ *Matrix.mat 4 4* (λ(*i*,*j*). *if i=0* ∧ *j=0 then 1 else*
if i=1 ∧ *j=2 then 1 else*
if i=2 ∧ *j=1 then 1 else*
if i=3 ∧ *j=3 then 1 else 0*)

lemma *SWAP-index*:
SWAP $$ (*0*,*0*) = *1* ∧
SWAP $$ (*0*,*1*) = *0* ∧
SWAP $$ (*0*,*2*) = *0* ∧
SWAP $$ (*0*,*3*) = *0* ∧
SWAP $$ (*1*,*0*) = *0* ∧
SWAP $$ (*1*,*1*) = *0* ∧
SWAP $$ (*1*,*2*) = *1* ∧
SWAP $$ (*1*,*3*) = *0* ∧
SWAP $$ (*2*,*0*) = *0* ∧
SWAP $$ (*2*,*1*) = *1* ∧
SWAP $$ (*2*,*2*) = *0* ∧
SWAP $$ (*2*,*3*) = *0* ∧
SWAP $$ (*3*,*0*) = *0* ∧
SWAP $$ (*3*,*1*) = *0* ∧

$SWAP$ $$ $(3,2) = 0 \wedge$
$SWAP$ $$ $(3,3) = 1$
by (*simp add*: *SWAP-def*)

lemma *SWAP-nrows*:
dim-row SWAP $= 4$
by (*simp add*: *SWAP-def*)

lemma *SWAP-ncols*:
dim-col SWAP $= 4$
by (*simp add*: *SWAP-def*)

lemma *SWAP-carrier-mat*[*simp*]:
SWAP $\in$ *carrier-mat 4 4*
using *SWAP-nrows SWAP-ncols* **by** *auto*

The SWAP gate indeed swaps the states of two qubits (it is not necessary to assume unitarity)

lemma *SWAP-tensor*:
assumes $u \in$ *carrier-mat 2 1*
and $v \in$ *carrier-mat 2 1*
shows $SWAP * (u \bigotimes v) = v \bigotimes u$
proof
show *dim-row* $(SWAP * (u \bigotimes v)) =$ *dim-row* $(v \bigotimes u)$
using *SWAP-nrows assms*(*1*) *assms*(*2*) **by** *auto*
next
show *dim-col* $(SWAP * (u \bigotimes v)) =$ *dim-col* $(v \bigotimes u)$
using *SWAP-ncols assms* **by** *auto*
next
fix *i j*::*nat* **assume** $i <$ *dim-row* $(v \bigotimes u)$ **and** $j <$ *dim-col* $(v \bigotimes u)$
hence *a3*:$i < 4$ **and** *a4*:$j = 0$ **using** *assms* **by** *auto*
thus $(SWAP * (u \bigotimes v))$ $$ $(i, j) = (v \bigotimes u)$ $$ (i, j)
proof −
define *u0* **where** $u0 = u$ $$ $(0,0)$
define *u1* **where** $u1 = u$ $$ $(1,0)$
define *v0* **where** $v0 = v$ $$ $(0,0)$
define *v1* **where** $v1 = v$ $$ $(1,0)$
have *vu0*:$(v \bigotimes u)$ $$ $(0,0) = v0*u0$ **using** *index-tensor-mat assms u0-def v0-def* **by** *auto*
have *vu1*:$(v \bigotimes u)$ $$ $(1,0) = v0*u1$ **using** *index-tensor-mat assms u1-def v0-def* **by** *auto*
have *vu2*:$(v \bigotimes u)$ $$ $(2,0) = v1*u0$ **using** *index-tensor-mat assms u0-def v1-def* **by** *auto*
have *vu3*:$(v \bigotimes u)$ $$ $(3,0) = v1*u1$ **using** *index-tensor-mat assms u1-def v1-def* **by** *auto*
have *uv0*:$(u \bigotimes v)$ $$ $(0,0) = u0*v0$ **using** *index-tensor-mat assms u0-def v0-def* **by** *auto*
have *uv1*:$(u \bigotimes v)$ $$ $(1,0) = u0*v1$ **using** *index-tensor-mat assms u0-def v1-def* **by** *auto*

have *uv2*:$(u \bigotimes v)$ $\$\$$ $(2,0) = u1*v0$ **using** *index-tensor-mat assms u1-def v0-def* **by** *auto*
have *uv3*:$(u \bigotimes v)$ $\$\$$ $(3,0) = u1*v1$ **using** *index-tensor-mat assms u1-def v1-def* **by** *auto*

have *uvi*:$Matrix.vec\ 4\ (\lambda\ i.\ (u \bigotimes v)\ \$\$\ (i,0))\ \$\ i = (u \bigotimes v)\ \$\$\ (i,0)$
using *a3 index-vec* **by** *blast*
have *sw*:$\forall\,k<4.\ Matrix.vec\ 4\ (\lambda\ j.\ SWAP\ \$\$\ (i,j))\ \$\ k = SWAP\ \$\$\ (i,k)$
using *a3 index-vec* **by** *auto*

have *s0*:$(SWAP * (u \bigotimes v))\ \$\$\ (i,0) = Matrix.vec\ (dim\text{-}col\ SWAP)\ (\lambda\ j.\ SWAP\ \$\$\ (i,j)) \cdot$
$Matrix.vec\ (dim\text{-}row\ (u \bigotimes v))\ (\lambda\ i.\ (u \bigotimes v)\ \$\$\ (i,0))$
by (*metis Matrix.col-def Matrix.row-def SWAP-nrows* ‹$i < 4$› ‹$j < dim\text{-}col\ (v \bigotimes u)$› ‹$j = 0$›
dim-col-tensor-mat index-mult-mat(*1*) *mult.commute*)
also have ... $= Matrix.vec\ 4\ (\lambda\ j.\ SWAP\ \$\$\ (i,j)) \cdot Matrix.vec\ 4\ (\lambda\ i.\ (u \bigotimes v)\ \$\$\ (i,0))$
using *SWAP-ncols assms*(*1*) *assms*(*2*) **by** *fastforce*
also have ... $= (\sum k<4.\ ((Matrix.vec\ 4\ (\lambda\ j.\ SWAP\ \$\$\ (i,j)))\ \$\ k) *$
$((Matrix.vec\ 4\ (\lambda\ i.\ (u \bigotimes v)\ \$\$\ (i,0)))\ \$\ k))$
using *scalar-prod-def* **by** (*metis calculation dim-vec lessThan-atLeast0*)
also have ... $= SWAP\ \$\$\ (i,0) * (u \bigotimes v)\ \$\$\ (0,0) +$
$SWAP\ \$\$\ (i,1) * (u \bigotimes v)\ \$\$\ (1,0) +$
$SWAP\ \$\$\ (i,2) * (u \bigotimes v)\ \$\$\ (2,0) +$
$SWAP\ \$\$\ (i,3) * (u \bigotimes v)\ \$\$\ (3,0)$
using *sumof4* **by** *auto*
also have ... $= SWAP\ \$\$\ (i,0) * u0 * v0 +$
$SWAP\ \$\$\ (i,1) * u0 * v1 +$
$SWAP\ \$\$\ (i,2) * u1 * v0 +$
$SWAP\ \$\$\ (i,3) * u1 * v1$
using *uv0 uv1 uv2 uv3* **by** *simp*
also have ... $= (v \bigotimes u)\ \$\$\ (i,j)$
proof (*rule disjE*)
show $i=0 \vee i=1 \vee i=2 \vee i=3$ **using** *a3* **by** *auto*
next
assume *i0*:$i=0$
hence $SWAP\ \$\$\ (i,0) * u0 * v0 +$
$SWAP\ \$\$\ (i,1) * u0 * v1 +$
$SWAP\ \$\$\ (i,2) * u1 * v0 +$
$SWAP\ \$\$\ (i,3) * u1 * v1 =$
$SWAP\ \$\$\ (0,0) * u0 * v0 +$
$SWAP\ \$\$\ (0,1) * u0 * v1 +$
$SWAP\ \$\$\ (0,2) * u1 * v0 +$
$SWAP\ \$\$\ (0,3) * u1 * v1$ **by** *simp*
also have ... $= (v \bigotimes u)\ \$\$\ (i,\ j)$ **using** *i0 vu0 SWAP-index a4* **by** *simp*
finally show *?thesis* **by** *this*
next
assume *disj3*:$i = 1 \vee i = 2 \vee i = 3$

```
      show ?thesis
      proof (rule disjE)
        show i = 1 ∨ i = 2 ∨ i = 3 using disj3 by this
      next
        assume i1:i=1
        hence SWAP $$ (i,0) ∗ u0 ∗ v0 +
              SWAP $$ (i,1) ∗ u0 ∗ v1 +
              SWAP $$ (i,2) ∗ u1 ∗ v0 +
              SWAP $$ (i,3) ∗ u1 ∗ v1 =
              SWAP $$ (1,0) ∗ u0 ∗ v0 +
              SWAP $$ (1,1) ∗ u0 ∗ v1 +
              SWAP $$ (1,2) ∗ u1 ∗ v0 +
              SWAP $$ (1,3) ∗ u1 ∗ v1 by simp
        also have ... = (v ⨂ u) $$ (i, j) using i1 vu1 SWAP-index a4 by simp
        finally show ?thesis by this
      next
        assume disj2:i = 2 ∨ i = 3
        show ?thesis
        proof (rule disjE)
          show i = 2 ∨ i = 3 using disj2 by this
        next
          assume i2:i=2
          hence SWAP $$ (i,0) ∗ u0 ∗ v0 +
                SWAP $$ (i,1) ∗ u0 ∗ v1 +
                SWAP $$ (i,2) ∗ u1 ∗ v0 +
                SWAP $$ (i,3) ∗ u1 ∗ v1 =
                SWAP $$ (2,0) ∗ u0 ∗ v0 +
                SWAP $$ (2,1) ∗ u0 ∗ v1 +
                SWAP $$ (2,2) ∗ u1 ∗ v0 +
                SWAP $$ (2,3) ∗ u1 ∗ v1 by simp
        also have ... = (v ⨂ u) $$ (i, j) using i2 vu2 SWAP-index a4 by simp
        finally show ?thesis by this
      next
        assume i3:i=3
        hence SWAP $$ (i,0) ∗ u0 ∗ v0 +
              SWAP $$ (i,1) ∗ u0 ∗ v1 +
              SWAP $$ (i,2) ∗ u1 ∗ v0 +
              SWAP $$ (i,3) ∗ u1 ∗ v1 =
              SWAP $$ (3,0) ∗ u0 ∗ v0 +
              SWAP $$ (3,1) ∗ u0 ∗ v1 +
              SWAP $$ (3,2) ∗ u1 ∗ v0 +
              SWAP $$ (3,3) ∗ u1 ∗ v1 by simp
        also have ... = (v ⨂ u) $$ (i, j) using i3 vu3 SWAP-index a4 by simp
        finally show ?thesis by this
      qed
    qed
  qed
  finally show ?thesis using a4 by simp
qed
```

qed

3.1 Downwards SWAP cascade

fun *SWAP-down*:: *nat* ⇒ *complex Matrix.mat* **where**
SWAP-down 0 = 1_m *1*
| *SWAP-down* (*Suc 0*) = 1_m *2*
| *SWAP-down* (*Suc* (*Suc 0*)) = *SWAP*
| *SWAP-down* (*Suc* (*Suc n*)) = ((1_m (*2*^*n*)) ⨂ *SWAP*) ∗ ((*SWAP-down* (*Suc n*)) ⨂ (1_m *2*))

lemma *SWAP-down-carrier-mat*[*simp*]:
shows *SWAP-down n* ∈ *carrier-mat* (*2*^*n*) (*2*^*n*) (**is** *?P n*)
proof (*induct n rule*: *SWAP-down.induct*)
show *?P 0* **by** *auto*
next
show *?P* (*Suc 0*) **by** *auto*
next
show *?P* (*Suc* (*Suc 0*)) **using** *SWAP-carrier-mat* **by** *auto*
next
fix *n*::*nat*
define *k*::*nat* **where** *k* = *Suc n*
assume *HI*:*SWAP-down* (*Suc k*) ∈ *carrier-mat* (*2*^(*Suc k*)) (*2*^(*Suc k*))
show *?P* (*Suc* (*Suc k*))
proof
have *dim-row* (*SWAP-down* (*Suc* (*Suc k*))) =
dim-row (((1_m (*2*^*k*)) ⨂ *SWAP*) ∗ ((*SWAP-down* (*Suc k*)) ⨂ (1_m *2*)))
using *SWAP-down.simps*(*4*) *k-def* **by** *simp*
also have ... = *dim-row* (((1_m (*2*^*k*)) ⨂ *SWAP*)) **by** *simp*
also have ... = (*dim-row* ((1_m (*2*^*k*)))) ∗ (*dim-row SWAP*) **by** *simp*
thus *dim-row* (*SWAP-down* (*Suc* (*Suc k*))) = *2* ^ *Suc* (*Suc k*) **using** *SWAP-nrows index-one-mat*
by (*simp add*: *calculation*)
next
have *dim-col* (*SWAP-down* (*Suc* (*Suc k*))) =
dim-col (((1_m (*2*^*k*)) ⨂ *SWAP*) ∗ ((*SWAP-down* (*Suc k*)) ⨂ (1_m *2*)))
using *SWAP-down.simps*(*4*) *k-def* **by** *simp*
also have ... = *dim-col* ((*SWAP-down* (*Suc k*)) ⨂ (1_m *2*)) **by** *simp*
also have ... = *dim-col* (*SWAP-down* (*Suc k*)) ∗ *dim-col* (1_m *2*) **by** *simp*
thus *dim-col* (*SWAP-down* (*Suc* (*Suc k*))) = *2* ^ *Suc* (*Suc k*)
using *SWAP-ncols index-one-mat calculation HI* **by** *simp*
qed
qed

3.2 Upwards SWAP cascade

fun *SWAP-up*:: *nat* ⇒ *complex Matrix.mat* **where**
SWAP-up 0 = 1_m *1*
| *SWAP-up* (*Suc 0*) = 1_m *2*
| *SWAP-up* (*Suc* (*Suc 0*)) = *SWAP*

| *SWAP-up* (*Suc* (*Suc* *n*)) = (*SWAP* $\bigotimes$ (1_m ($2\hat{}n$))) $*$ ((1_m 2) $\bigotimes$ (*SWAP-up* (*Suc* *n*)))

lemma *SWAP-up-carrier-mat*[*simp*]:
shows *SWAP-up n* $\in$ *carrier-mat* ($2\hat{}n$) ($2\hat{}n$) (**is** *?P n*)
proof (*induct n rule*: *SWAP-up.induct*)
case *1*
then show *?case* **by** *auto*
next
case *2*
then show *?case* **by** *auto*
next
case *3*
then show *?case* **by** *auto*
next
case (*4 v*)
then show *?case* **using** *SWAP-nrows* **by** *fastforce*
qed

4 Reversing qubits

In order to reverse the order of n qubits, we iteratively swap opposite qubits (swap 0th and (n-1)th qubits, 1st and (n-2)th qubits, and so on).

fun *reverse-qubits*:: *nat* $\Rightarrow$ *complex Matrix.mat* **where**
reverse-qubits 0 = 1_m *1*
| *reverse-qubits* (*Suc 0*) = (1_m *2*)
| *reverse-qubits* (*Suc* (*Suc 0*)) = *SWAP*
| *reverse-qubits* (*Suc n*) = ((*reverse-qubits n*) $\bigotimes$ (1_m *2*)) $*$ (*SWAP-down* (*Suc n*))

lemma *reverse-qubits-carrier-mat*[*simp*]:
(*reverse-qubits n*) $\in$ *carrier-mat* ($2\hat{}n$) ($2\hat{}n$)
proof (*induct n rule*: *reverse-qubits.induct*)
case *1*
then show *?case* **by** *auto*
next
case *2*
then show *?case* **by** *auto*
next
case *3*
then show *?case* **by** *auto*
next
case (*4 va*)
then show *?case*
by (*metis SWAP-down-carrier-mat carrier-matD*(*1*) *carrier-matD*(*2*) *carrier-matI dim-row-tensor-mat*
index-mult-mat(*2*) *index-mult-mat*(*3*) *index-one-mat*(*2*) *power-Suc2 reverse-qubits.simps*(*4*))
qed

5 Controlled operations

The two-qubit gate control2 performs a controlled U operation on the first qubit with the second qubit as control

definition *control2*:: *complex Matrix.mat* ⇒ *complex Matrix.mat* **where**
control2 U ≡ *mat-of-cols-list 4* [[*1*, *0*, *0*, *0*],
[*0*, *U*$$(*0*,*0*), *0*, *U*$$(*1*,*0*)],
[*0*, *0*, *1*, *0*],
[*0*, *U*$$(*0*,*1*), *0*, *U*$$(*1*,*1*)]]

lemma *control2-carrier-mat*[*simp*]:
shows *control2 U* ∈ *carrier-mat 4 4*
by (*simp add*: *Tensor.mat-of-cols-list-def control2-def numeral-Bit0*)

lemma *control2-zero*:
assumes *dim-row v* = *2* **and** *dim-col v* = *1*
shows *control2 U* ∗ (*v* ⨂ |*zero*⟩) = *v* ⨂ |*zero*⟩
proof
fix *i j*::*nat*
assume *i* < *dim-row* (*v* ⨂ |*zero*⟩)
hence *i4*:*i* < *4* **using** *assms tensor-carrier-mat ket-vec-def* **by** *auto*
assume *j* < *dim-col* (*v* ⨂ |*zero*⟩)
hence *j0*:*j* = *0* **using** *assms tensor-carrier-mat ket-vec-def* **by** *auto*
show (*control2 U* ∗ (*v* ⨂ |*zero*⟩)) $$ (*i*,*j*) = (*v* ⨂ |*zero*⟩) $$ (*i*,*j*)
proof −
have (*control2 U* ∗ (*v* ⨂ |*zero*⟩)) $$ (*i*,*j*) =
(∑ *k*<*dim-row* (*v* ⨂ |*zero*⟩). *control2 U* $$ (*i*, *k*) ∗ (*v* ⨂ |*zero*⟩) $$ (*k*, *j*))
using *assms index-matrix-prod*
by (*smt* (*z3*) *One-nat-def Suc-1 Tensor.mat-of-cols-list-def* ‹*i* < *dim-row* (*v* ⨂ |*Deutsch.zero*⟩)›
‹*j* < *dim-col* (*v* ⨂ |*Deutsch.zero*⟩)› *add.commute add-Suc-right control2-def dim-col-mat*(*1*)
dim-row-mat(*1*) *dim-row-tensor-mat ket-zero-to-mat-of-cols-list list.size*(*3*) *list.size*(*4*)
mult-2 numeral-Bit0 plus-1-eq-Suc sum.cong)
also have ... = (∑ *k*<*4*. *control2 U* $$ (*i*, *k*) ∗ (*v* ⨂ |*zero*⟩) $$ (*k*, *j*))
using *assms tensor-carrier-mat ket-vec-def* **by** *auto*
also have ... = *control2 U* $$ (*i*, *0*) ∗ (*v* ⨂ |*zero*⟩) $$ (*0*, *0*) +
control2 U $$ (*i*, *1*) ∗ (*v* ⨂ |*zero*⟩) $$ (*1*, *0*) +
control2 U $$ (*i*, *2*) ∗ (*v* ⨂ |*zero*⟩) $$ (*2*, *0*) +
control2 U $$ (*i*, *3*) ∗ (*v* ⨂ |*zero*⟩) $$ (*3*, *0*)
using *sumof4 j0* **by** *blast*
also have ... = (*v* ⨂ |*zero*⟩) $$ (*i*,*0*)
proof (*rule disjE*)
show *i* = *0* ∨ *i* = *1* ∨ *i* = *2* ∨ *i* = *3* **using** *i4* **by** *auto*
next
assume *i0*:*i* = *0*

```
    have c00:control2 U $$ (0,0) = 1
      by (simp add: control2-def one-complex.code)
    have c01:control2 U $$ (0,1) = 0
      by (simp add: control2-def zero-complex.code)
    have c02:control2 U $$ (0,2) = 0
      by (simp add: control2-def zero-complex.code)
    have c03:control2 U $$ (0,3) = 0
      by (simp add: control2-def zero-complex.code)
    have control2 U $$ (0, 0) ∗ (v ⨂ |zero⟩) $$ (0, 0) +
          control2 U $$ (0, 1) ∗ (v ⨂ |zero⟩) $$ (1, 0) +
          control2 U $$ (0, 2) ∗ (v ⨂ |zero⟩) $$ (2, 0) +
          control2 U $$ (0, 3) ∗ (v ⨂ |zero⟩) $$ (3, 0) =
          1 ∗ (v ⨂ |zero⟩) $$ (0, 0) +
          0 ∗ (v ⨂ |zero⟩) $$ (1, 0) +
          0 ∗ (v ⨂ |zero⟩) $$ (2, 0) +
          0 ∗ (v ⨂ |zero⟩) $$ (3, 0)
      using c00 c01 c02 c03 by simp
    also have ... = (v ⨂ |zero⟩) $$ (0, 0) by auto
    finally show control2 U $$ (i, 0) ∗ (v ⨂ |Deutsch.zero⟩) $$ (0, 0) +
                control2 U $$ (i, 1) ∗ (v ⨂ |Deutsch.zero⟩) $$ (1, 0) +
                control2 U $$ (i, 2) ∗ (v ⨂ |Deutsch.zero⟩) $$ (2, 0) +
                control2 U $$ (i, 3) ∗ (v ⨂ |Deutsch.zero⟩) $$ (3, 0) =
                (v ⨂ |Deutsch.zero⟩) $$ (i, 0)
      using i0 by simp
  next
    assume id:i = 1 ∨ i = 2 ∨ i = 3
    show control2 U $$ (i, 0) ∗ (v ⨂ |Deutsch.zero⟩) $$ (0, 0) +
          control2 U $$ (i, 1) ∗ (v ⨂ |Deutsch.zero⟩) $$ (1, 0) +
          control2 U $$ (i, 2) ∗ (v ⨂ |Deutsch.zero⟩) $$ (2, 0) +
          control2 U $$ (i, 3) ∗ (v ⨂ |Deutsch.zero⟩) $$ (3, 0) =
          (v ⨂ |Deutsch.zero⟩) $$ (i, 0)
    proof (rule disjE)
      show i = 1 ∨ i = 2 ∨ i = 3 using id by this
    next
      assume i1:i = 1
      have c10:control2 U $$ (1,0) = 0
        by (simp add: control2-def zero-complex.code)
      have t10:(v ⨂ |zero⟩) $$ (1,0) = 0
        using index-tensor-mat ket-vec-def Tensor.mat-of-cols-list-def
         ‹i < dim-row (v ⨂ |Deutsch.zero⟩)› ‹j < dim-col (v ⨂ |Deutsch.zero⟩)›
i1
        by fastforce
      have c12:control2 U $$ (1,2) = 0
        by (simp add: control2-def zero-complex.code)
      have t30:(v ⨂ |zero⟩) $$ (3,0) = 0
      proof −
        have (v ⨂ |zero⟩) $$ (3,0) = v $$ (1,0) ∗ |zero⟩ $$ (1,0)
          using index-tensor-mat
          by (smt (verit) Euclidean-Rings.div-eq-0-iff H-on-ket-zero-is-state
```

```
              H-without-scalar-prod One-nat-def Suc-1 ‹j < dim-col (v ⊗
|Deutsch.zero⟩)›
              add.commute assms(1) dim-col-tensor-mat dim-row-mat(1) in-
dex-mult-mat(2) j0
              ket-zero-is-state mod-less mod-less-divisor mod-mult2-eq mult-2
nat-0-less-mult-iff
              numeral-3-eq-3 plus-1-eq-Suc pos2 state.dim-row three-div-two
three-mod-two)
      also have ... = 0 by auto
      finally show ?thesis by this
    qed
    show control2 U $$ (i, 0) * (v ⊗ |Deutsch.zero⟩) $$ (0, 0) +
        control2 U $$ (i, 1) * (v ⊗ |Deutsch.zero⟩) $$ (1, 0) +
        control2 U $$ (i, 2) * (v ⊗ |Deutsch.zero⟩) $$ (2, 0) +
        control2 U $$ (i, 3) * (v ⊗ |Deutsch.zero⟩) $$ (3, 0) =
        (v ⊗ |Deutsch.zero⟩) $$ (i, 0)
      using i1 c10 t10 c12 t30 by auto
  next
    assume id2:i = 2 ∨ i = 3
    show control2 U $$ (i, 0) * (v ⊗ |Deutsch.zero⟩) $$ (0, 0) +
        control2 U $$ (i, 1) * (v ⊗ |Deutsch.zero⟩) $$ (1, 0) +
        control2 U $$ (i, 2) * (v ⊗ |Deutsch.zero⟩) $$ (2, 0) +
        control2 U $$ (i, 3) * (v ⊗ |Deutsch.zero⟩) $$ (3, 0) =
        (v ⊗ |Deutsch.zero⟩) $$ (i, 0)
    proof (rule disjE)
      show i = 2 ∨ i = 3
        using id2 by this
    next
      assume i2:i = 2
      have c20:control2 U $$ (2,0) = 0
        by (simp add: control2-def zero-complex.code)
      have c21:control2 U $$ (2,1) = 0
        by (simp add: control2-def zero-complex.code)
      have c22:control2 U $$ (2,2) = 1
        by (simp add: control2-def one-complex.code)
      have c23:control2 U $$ (2,3) = 0
        by (simp add: control2-def zero-complex.code)
      show control2 U $$ (i, 0) * (v ⊗ |Deutsch.zero⟩) $$ (0, 0) +
          control2 U $$ (i, 1) * (v ⊗ |Deutsch.zero⟩) $$ (1, 0) +
          control2 U $$ (i, 2) * (v ⊗ |Deutsch.zero⟩) $$ (2, 0) +
          control2 U $$ (i, 3) * (v ⊗ |Deutsch.zero⟩) $$ (3, 0) =
          (v ⊗ |Deutsch.zero⟩) $$ (i, 0)
        using i2 c20 c21 c22 c23 by auto
    next
      assume i3:i = 3
      have c30:control2 U $$ (3,0) = 0
        by (simp add: control2-def zero-complex.code)
      have t10:(v ⊗ |zero⟩) $$ (1,0) = 0
        using index-tensor-mat ket-vec-def Tensor.mat-of-cols-list-def
```

```
        ‹i < dim-row (v ⨂ |Deutsch.zero⟩)› ‹j < dim-col (v ⨂ |Deutsch.zero⟩)›
i3
          by fastforce
        have c32:control2 U $$ (3,2) = 0
          by (simp add: control2-def zero-complex.code)
        have t30:(v ⨂ |zero⟩) $$ (3,0) = 0
        proof −
          have (v ⨂ |zero⟩) $$ (3,0) = v $$ (1,0) ∗ |zero⟩ $$ (1,0)
            using index-tensor-mat
            by (smt (verit) Euclidean-Rings.div-eq-0-iff H-on-ket-zero-is-state
                  H-without-scalar-prod One-nat-def Suc-1 ‹j < dim-col (v ⨂
|Deutsch.zero⟩)›
                  add.commute assms(1) dim-col-tensor-mat dim-row-mat(1) in-
dex-mult-mat(2) j0
                  ket-zero-is-state mod-less mod-less-divisor mod-mult2-eq mult-2
nat-0-less-mult-iff
                  numeral-3-eq-3 plus-1-eq-Suc pos2 state.dim-row three-div-two
three-mod-two)
          also have ... = 0 by auto
          finally show ?thesis by this
        qed
        show control2 U $$ (i, 0) ∗ (v ⨂ |Deutsch.zero⟩) $$ (0, 0) +
             control2 U $$ (i, 1) ∗ (v ⨂ |Deutsch.zero⟩) $$ (1, 0) +
             control2 U $$ (i, 2) ∗ (v ⨂ |Deutsch.zero⟩) $$ (2, 0) +
             control2 U $$ (i, 3) ∗ (v ⨂ |Deutsch.zero⟩) $$ (3, 0) =
             (v ⨂ |Deutsch.zero⟩) $$ (i, 0)
          using i3 c30 t10 c32 t30 by auto
      qed
    qed
  qed
  finally show ?thesis using j0 by simp
 qed
next
 show dim-row (control2 U ∗ (v ⨂ |Deutsch.zero⟩)) = dim-row (v ⨂ |Deutsch.zero⟩)
  by (metis assms(1) carrier-matD(1) control2-carrier-mat dim-row-mat(1) dim-row-tensor-mat

    index-mult-mat(2) index-unit-vec(3) ket-vec-def num-double numeral-times-numeral)
next
 show dim-col (control2 U ∗ (v ⨂ |Deutsch.zero⟩)) = dim-col (v ⨂ |Deutsch.zero⟩)
   using index-mult-mat(3) by blast
qed


lemma vtensorone-index[simp]:
 assumes dim-row v = 2 and dim-col v = 1
 shows (v ⨂ |one⟩) $$ (0,0) = 0 ∧
       (v ⨂ |one⟩) $$ (1,0) = v $$ (0,0) ∧
       (v ⨂ |one⟩) $$ (2,0) = 0 ∧
       (v ⨂ |one⟩) $$ (3,0) = v $$ (1,0)
```

```
  by (simp add: assms(1) assms(2) ket-vec-def)

lemma control2-one:
  assumes dim-row v = 2 and dim-col v = 1 and dim-row U = 2 and dim-col
U = 2
  shows control2 U ∗ (v ⨂ |one⟩) = (U∗v) ⨂ |one⟩
proof
  fix i j::nat
  assume i < dim-row ((U∗v) ⨂ |one⟩)
  hence il4:i < 4 by (simp add: assms(3) ket-vec-def)
  assume j < dim-col ((U∗v) ⨂ |one⟩)
  hence j0:j = 0 using assms ket-vec-def by simp
  show (control2 U ∗ (v ⨂ |Deutsch.one⟩)) $$ (i, j) = (U ∗ v ⨂ |Deutsch.one⟩)
$$ (i, j)
  proof −
    have (control2 U ∗ (v ⨂ |one⟩)) $$ (i,j) =
        (∑ k<dim-row (v ⨂ |one⟩). (control2 U) $$ (i, k) ∗ (v ⨂ |one⟩) $$ (k,
j))
      using assms index-matrix-prod tensor-carrier-mat
    proof −
      have ⋀m. dim-col (v ⨂ m) = dim-col m
        by (simp add: assms(2))
      then have i < dim-row (control2 U) ∧ 0 < dim-col (v ⨂ Matrix.mat 2 1
(λ(n, n). Deutsch.one $ n)) ∧ dim-row (v ⨂ Matrix.mat 2 1 (λ(n, n). Deutsch.one
$ n)) = dim-col (control2 U)
      by (smt (z3) assms(1) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-mat(1) dim-row-mat(1) dim-row-tensor-mat il4 mult-2 numeral-Bit0 zero-less-one-class.zero-less-one)
      then show ?thesis
        by (simp add: j0 ket-vec-def)
    qed
    also have . . . = (∑ k<4. control2 U $$ (i, k) ∗ (v ⨂ |one⟩) $$ (k, j))
      using assms tensor-carrier-mat ket-vec-def by auto
    also have . . . = control2 U $$ (i, 0) ∗ (v ⨂ |one⟩) $$ (0, 0) +
                 control2 U $$ (i, 1) ∗ (v ⨂ |one⟩) $$ (1, 0) +
                 control2 U $$ (i, 2) ∗ (v ⨂ |one⟩) $$ (2, 0) +
                 control2 U $$ (i, 3) ∗ (v ⨂ |one⟩) $$ (3, 0)
      using sumof4 j0 by blast
    also have . . . = ((U∗v) ⨂ |one⟩) $$ (i,0)
    proof (rule disjE)
      show i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3 using il4 by auto
    next
      assume i0:i = 0
      thus control2 U $$ (i, 0) ∗ (v ⨂ |Deutsch.one⟩) $$ (0, 0) +
          control2 U $$ (i, 1) ∗ (v ⨂ |Deutsch.one⟩) $$ (1, 0) +
          control2 U $$ (i, 2) ∗ (v ⨂ |Deutsch.one⟩) $$ (2, 0) +
          control2 U $$ (i, 3) ∗ (v ⨂ |Deutsch.one⟩) $$ (3, 0) =
          (U ∗ v ⨂ |Deutsch.one⟩) $$ (i, 0)
        using j0 control2-def zero-complex.code one-complex.code vtensorone-index
assms by auto
```

next
assume *id3*: $i = 1 \vee i = 2 \vee i = 3$
show *control2 U* $$ (i, 0) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (0, 0) +
control2 U $$ (i, 1) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (1, 0) +
control2 U $$ (i, 2) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (2, 0) +
control2 U $$ (i, 3) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (3, 0) =
($U * v \bigotimes |Deutsch.one\rangle$) $$ (i, 0)
proof (*rule disjE*)
show $i = 1 \vee i = 2 \vee i = 3$ **using** *id3* **by** *this*
next
assume *i1*: $i = 1$
thus *control2 U* $$ (i, 0) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (0, 0) +
control2 U $$ (i, 1) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (1, 0) +
control2 U $$ (i, 2) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (2, 0) +
control2 U $$ (i, 3) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (3, 0) =
($U * v \bigotimes |Deutsch.one\rangle$) $$ (i, 0)
using *j0 control2-def zero-complex.code one-complex.code vtensorone-index assms*
by (*simp add*: *sumof2*)
next
assume *il2*: $i = 2 \vee i = 3$
show *control2 U* $$ (i, 0) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (0, 0) +
control2 U $$ (i, 1) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (1, 0) +
control2 U $$ (i, 2) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (2, 0) +
control2 U $$ (i, 3) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (3, 0) =
($U * v \bigotimes |Deutsch.one\rangle$) $$ (i, 0)
proof (*rule disjE*)
show $i = 2 \vee i = 3$ **using** *il2* **by** *this*
next
assume *i2*: $i = 2$
thus *control2 U* $$ (i, 0) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (0, 0) +
control2 U $$ (i, 1) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (1, 0) +
control2 U $$ (i, 2) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (2, 0) +
control2 U $$ (i, 3) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (3, 0) =
($U * v \bigotimes |Deutsch.one\rangle$) $$ (i, 0)
using *j0 control2-def zero-complex.code one-complex.code vtensorone-index assms* **by** *auto*
next
assume *i3*: $i = 3$
thus *control2 U* $$ (i, 0) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (0, 0) +
control2 U $$ (i, 1) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (1, 0) +
control2 U $$ (i, 2) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (2, 0) +
control2 U $$ (i, 3) $*$ ($v \bigotimes |Deutsch.one\rangle$) $$ (3, 0) =
($U * v \bigotimes |Deutsch.one\rangle$) $$ (i, 0)
using *j0 control2-def zero-complex.code one-complex.code vtensorone-index assms*
by (*simp add*: *sumof2*)
qed
qed

qed
finally show *?thesis* **using** *j0* **by** *simp*
qed
next
show *dim-row* (*control2 U* ∗ (*v* ⨂ |*Deutsch.one*⟩)) = *dim-row* (*U* ∗ *v* ⨂ |*Deutsch.one*⟩)
by (*metis assms*(*3*) *carrier-matD*(*1*) *control2-carrier-mat dim-row-mat*(*1*) *dim-row-tensor-mat*

index-mult-mat(*2*) *index-unit-vec*(*3*) *ket-vec-def mult-2-right numeral-Bit0*)
next
show *dim-col* (*control2 U* ∗ (*v* ⨂ |*Deutsch.one*⟩)) = *dim-col* (*U* ∗ *v* ⨂ |*Deutsch.one*⟩)
by *simp*
qed

Given a single qubit gate U, control n U creates a quantum n-qubit gate that performs a controlled-U operation on the first qubit using the last qubit as control.

fun *control*:: *nat* ⇒ *complex Matrix.mat* ⇒ *complex Matrix.mat* **where**
control 0 U = 1_m *1*
| *control* (*Suc 0*) *U* = 1_m *2*
| *control* (*Suc* (*Suc 0*)) *U* = *control2 U*
| *control* (*Suc* (*Suc n*)) *U* =
(((1_m *2*) ⨂ *SWAP-down* (*Suc n*)) ∗ (*control2 U* ⨂ (1_m (*2^n*))) ∗ ((1_m *2*) ⨂ *SWAP-up* (*Suc n*))

lemma *control-carrier-mat*[*simp*]:
shows *control n U* ∈ *carrier-mat* (*2^n*) (*2^n*)
proof (*cases n*)
case *0*
then show *?thesis* **by** *auto*
next
case (*Suc nat*)
then show *?thesis*
by (*smt* (*verit*, *best*) *One-nat-def SWAP-down-carrier-mat SWAP-up.simps*(*2*) *SWAP-up.simps*(*4*)
SWAP-up-carrier-mat Suc-1 Zero-not-Suc carrier-matD(*1*) *carrier-matD*(*2*) *carrier-matI*
control.elims control2-carrier-mat dim-col-tensor-mat dim-row-tensor-mat index-mult-mat(*2*)
index-mult-mat(*3*) *mult-2 numeral-Bit0 power2-eq-square*)
qed

6 Quantum Fourier Transform Circuit

6.1 QFT definition

The function kron is the generalization of the Kronecker product to a finite number of qubits

fun *kron*:: (*nat* ⇒ *complex Matrix.mat*) ⇒ *nat list* ⇒ *complex Matrix.mat* **where**
kron f [] = 1_m *1*
| *kron f* (*x#xs*) = (*f x*) ⨂ (*kron f xs*)

lemma *kron-carrier-mat*[*simp*]:
assumes ∀ *m*. *dim-row* (*f m*) = *2* ∧ *dim-col* (*f m*) = *1*
shows *kron f xs* ∈ *carrier-mat* (*2*^(*length xs*)) *1*
proof (*induct xs*)
case *Nil*
show *?case*
proof
have *dim-row* (*kron f* []) = *dim-row* (1_m *1*) **using** *kron.simps*(*1*) **by** *simp*
then show *dim-row* (*kron f* []) = *2* ^ *length* [] **by** *simp*
next
have *dim-col* (*kron f* []) = *dim-col* (1_m *1*) **using** *kron.simps*(*1*) **by** *simp*
then show *dim-col* (*kron f* []) = *1* **by** *simp*
qed
next
case (*Cons x xs*)
assume *HI*:*kron f xs* ∈ *carrier-mat* (*2* ^ *length xs*) *1*
have *f x* ∈ *carrier-mat 2 1* **using** *assms* **by** *auto*
moreover have (*f x*) ⨂ (*kron f xs*) ∈ *carrier-mat* ((*2* ^ *length xs*) ∗ *2*) *1*
using *tensor-carrier-mat HI calculation* **by** *auto*
moreover have *kron f* (*x#xs*) ∈ *carrier-mat* (*2* ^ (*length* (*x#xs*))) *1*
using *kron.simps*(*2*) *length-Cons* **by** (*metis calculation*(*2*) *power-Suc2*)
thus *?case* **by** *this*
qed

lemma *kron-cons-right*:
shows *kron f* (*xs*@[*x*]) = *kron f xs* ⨂ *f x*
proof (*induct xs*)
case *Nil*
have *kron f* ([]@[*x*]) = *kron f* [*x*] **by** *simp*
also have . . . = *f x* **using** *kron.simps* **by** *auto*
also have . . . = *kron f* [] ⨂ *f x* **by** *auto*
finally show *?case* **by** *this*
next
case (*Cons a xs*)
assume *IH*:*kron f* (*xs*@[*x*]) = *kron f xs* ⨂ *f x*
have *kron f* ((*a#xs*)@[*x*]) = *f a* ⨂ (*kron f* (*xs*@[*x*])) **using** *kron.simps* **by** *auto*
also have . . . = *f a* ⨂ (*kron f xs* ⨂ *f x*) **using** *IH* **by** *simp*
also have . . . = *kron f* (*a#xs*) ⨂ *f x* **using** *kron.simps tensor-mat-is-assoc* **by** *auto*
finally show *?case* **by** *this*
qed

We define the QFT product representation

definition *QFT-product-representation*:: *nat* ⇒ *nat* ⇒ *complex Matrix.mat* **where**

QFT-product-representation j n ≡ *1*/(*sqrt* (*2*^*n*)) $\cdot_m$
(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$
|*one*⟩)
(*map nat* [*1*..*n*]))

We also define the reverse version of the QFT product representation, which is the output state of the QFT circuit alone

definition *reverse-QFT-product-representation*:: *nat* ⇒ *nat* ⇒ *complex Matrix.mat*
where
reverse-QFT-product-representation j n ≡ *1*/(*sqrt* (*2*^*n*)) $\cdot_m$
(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*))
$\cdot_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*])))

6.2 QFT circuit

The recursive function controlled_rotations computes the controlled-R_k gates subcircuit of the QFT circuit at each stage (i.e. for each qubit).

fun *controlled-rotations*:: *nat* ⇒ *complex Matrix.mat* **where**
controlled-rotations 0 = 1_m *1*
| *controlled-rotations* (*Suc 0*) = 1_m *2*
| *controlled-rotations* (*Suc n*) = (*control* (*Suc n*) (*R* (*Suc n*))) ∗
((*controlled-rotations n*) ⨂ (1_m *2*))

lemma *controlled-rotations-carrier-mat*[*simp*]:
controlled-rotations n ∈ *carrier-mat* (*2*^*n*) (*2*^*n*)
proof (*induct n rule*: *controlled-rotations.induct*)
case *1*
then show *?case* **by** *auto*
next
case *2*
then show *?case* **by** *auto*
next
case *3*
then show *?case*
by (*smt* (*verit*, *del-insts*) *carrier-matD*(*1*) *carrier-matD*(*2*) *carrier-mat-triv*
control-carrier-mat
controlled-rotations.simps(*3*) *dim-col-tensor-mat index-mult-mat*(*2*) *in-dex-mult-mat*(*3*)
index-one-mat(*3*) *mult.commute power-Suc*)
qed

The recursive function QFT computes the Quantum Fourier Transform circuit.

fun *QFT*:: *nat* ⇒ *complex Matrix.mat* **where**
QFT 0 = 1_m *1*
| *QFT* (*Suc 0*) = *H*

| QFT (Suc n) = ((*1*$_m$ *2*) $\bigotimes$ (QFT n)) $*$ (*controlled-rotations* (Suc n)) $*$ (H $\bigotimes$ ((*1*$_m$ (*2*^n))))

lemma *QFT-carrier-mat*[*simp*]:
  QFT n $\in$ *carrier-mat* (*2*^*n*) (*2*^*n*)
proof (*induct n rule*: *QFT.induct*)
  case *1*
  then show *?case* **by** *auto*
next
  case *2*
  then show *?case*
    using *H-is-gate One-nat-def QFT.simps*(*2*) *gate-carrier-mat* **by** *presburger*
next
  case *3*
  then show *?case*
   by (*metis H-inv QFT.simps*(*3*) *carrier-matD*(*1*) *carrier-mat-triv dim-col-tensor-mat*
      dim-row-tensor-mat index-mult-mat(*2*) *index-mult-mat*(*3*) *index-one-mat*(*2*)
index-one-mat(*3*)
       power.simps(*2*))
qed

ordered_QFT reverses the order of the qubits at the end of the QFT circuit

definition *ordered-QFT*:: *nat* $\Rightarrow$ *complex Matrix.mat* **where**
  ordered-QFT n $\equiv$ (*reverse-qubits n*) $*$ (*QFT n*)

7 QFT circuit correctness

Some useful lemmas:

lemma *state-basis-dec*:
  assumes *j* < *2* ^ *Suc n*
  shows $|$*state-basis 1* (*j div 2*^*n*)$\rangle$ $\bigotimes$ $|$*state-basis n* (*j mod 2*^*n*)$\rangle$ = $|$*state-basis* (*Suc n*) *j*$\rangle$
proof −
  define *jd jm* **where** *jd* = *j div 2*^*n* **and** *jm* = *j mod 2*^*n*
  hence *jml*:*jm* < *2*^*n* **by** *auto*
  have *j-dec*:*j* = *jd*$*$(*2*^*n*) + *jm* **using** *jd-def jm-def* **by** *presburger*
  show *?thesis*
  proof (*rule disjE*)
    show *jd* = *0* $\vee$ *jd* = *1* **using** *jd-def assms*
      by (*metis One-nat-def less-2-cases less-power-add-imp-div-less plus-1-eq-Suc power-one-right*)
  next
    assume *jd0*:*jd* = *0*
    hence *jjm*:*j* = *jm* **using** *j-dec* **by** *auto*
    show $|$*state-basis 1* (*j div 2*^*n*)$\rangle$ $\bigotimes$ $|$*state-basis n* (*j mod 2*^*n*)$\rangle$ = $|$*state-basis* (*Suc n*) *j*$\rangle$
    proof

fix *i ja*
assume $i < \textit{dim-row}\ (\ |\textit{state-basis}\ (\textit{Suc}\ n)\ j\rangle)$
and *ja-dim*:$ja < \textit{dim-col}\ (\ |\textit{state-basis}\ (\textit{Suc}\ n)\ j\rangle)$
hence *il*:$i < 2\hat{}\textit{Suc}\ n$ **using** *state-basis-carrier-mat ket-vec-def state-basis-def*
by *simp*
have *jal*:$ja < 1$ **using** *ja-dim state-basis-carrier-mat state-basis-def ket-vec-def*
by *simp*
hence *ja0*:$ja = 0$ **by** *auto*
show $(\ |\textit{state-basis}\ 1\ (j\ \textit{div}\ 2\ \hat{}\ n)\rangle \bigotimes |\textit{state-basis}\ n\ (j\ \textit{mod}\ 2\ \hat{}\ n)\rangle)$ \$\$ $(i,$
$ja) =$
$|\textit{state-basis}\ (\textit{Suc}\ n)\ j\rangle$ \$\$ $(i,\ ja)$
proof −
have $(\ |\textit{state-basis}\ 1\ (j\ \textit{div}\ 2\ \hat{}\ n)\rangle \bigotimes |\textit{state-basis}\ n\ (j\ \textit{mod}\ 2\ \hat{}\ n)\rangle)$ \$\$ $(i,$
$ja) =$
$(\ |\textit{state-basis}\ 1\ 0\rangle \bigotimes |\textit{state-basis}\ n\ jm\rangle)$ \$\$ $(i,0)$
using *jm-def jd0 ja0 jd-def* **by** *auto*
also have $\ldots = |\textit{state-basis}\ 1\ 0\rangle$ \$\$
$(i\ \textit{div}\ (\textit{dim-row}\ |\textit{state-basis}\ n\ jm\rangle),\ 0\ \textit{div}\ (\textit{dim-col}\ |\textit{state-basis}\ n$
$jm\rangle)) *$
$|\textit{state-basis}\ n\ jm\rangle$ \$\$
$(i\ \textit{mod}\ (\textit{dim-row}\ |\textit{state-basis}\ n\ jm\rangle),\ 0\ \textit{mod}\ (\textit{dim-col}\ |\textit{state-basis}$
$n\ jm\rangle))$
proof (*rule index-tensor-mat*)
show $\textit{dim-row}\ |\textit{state-basis}\ 1\ 0\rangle = 2$
using *state-basis-carrier-mat state-basis-def ket-vec-def* **by** *simp*
show $\textit{dim-col}\ |\textit{state-basis}\ 1\ 0\rangle = 1$
using *state-basis-carrier-mat state-basis-def ket-vec-def* **by** *simp*
show $\textit{dim-row}\ |\textit{state-basis}\ n\ jm\rangle = \textit{dim-row}\ |\textit{state-basis}\ n\ jm\rangle$ **by** *auto*
show $\textit{dim-col}\ |\textit{state-basis}\ n\ jm\rangle = \textit{dim-col}\ |\textit{state-basis}\ n\ jm\rangle$ **by** *auto*
show $i < 2 * \textit{dim-row}\ |\textit{state-basis}\ n\ jm\rangle$
using *il state-basis-def state-basis-carrier-mat ket-vec-def* **by** *simp*
show $0 < 1 * \textit{dim-col}\ |\textit{state-basis}\ n\ jm\rangle$
using *state-basis-def state-basis-carrier-mat ket-vec-def* **by** *simp*
show $0 < (1{::}\textit{nat})$ **using** *zero-less-Suc One-nat-def* **by** *blast*
show $0 < \textit{dim-col}\ |\textit{state-basis}\ n\ jm\rangle$
using *state-basis-def state-basis-carrier-mat ket-vec-def* **by** *simp*
qed
also have $\ldots = |\textit{state-basis}\ 1\ 0\rangle$ \$\$ $(i\ \textit{div}\ 2\hat{}n,\ 0) * |\textit{state-basis}\ n\ jm\rangle$ \$\$ $(i$
$\textit{mod}\ 2\hat{}n,\ 0)$
using *state-basis-def state-basis-carrier-mat ket-vec-def* **by** *auto*
also have $\ldots = (\textit{mat-of-cols-list}\ 2\ [[1,0]])$ \$\$ $(i\ \textit{div}\ 2\hat{}n,\ 0) *$
$|\textit{state-basis}\ n\ jm\rangle$ \$\$ $(i\ \textit{mod}\ 2\hat{}n,\ 0)$
using *state-basis-def unit-vec-def* **by** *auto*
also have $\ldots = |\textit{state-basis}\ (\textit{Suc}\ n)\ j\rangle$ \$\$ $(i,0)$
proof −
define *id im* **where** $id = i\ \textit{div}\ 2\hat{}n$ **and** $im = i\ \textit{mod}\ 2\hat{}n$
have *i-dec*:$i = id*(2\hat{}n) + im$ **using** *id-def im-def* **by** *presburger*
show *?thesis*
proof (*rule disjE*)

show *id* = *0* ∨ *id* = *1* **using** *id-def* **by** (*metis One-nat-def il less-2-cases less-power-add-imp-div-less plus-1-eq-Suc power-one-right*)
next
assume *id0*:*id* = *0*
hence *iim*:*i* = *im* **using** *i-dec* **by** *presburger*
have *mat-of-cols-list 2* [[*1*,*0*]] $$ (*i div 2^n*,*0*) ∗ |*state-basis n jm*⟩ $$ (*i mod 2^n*, *0*)
= *mat-of-cols-list 2* [[*1*,*0*]] $$ (*0*,*0*) ∗ |*state-basis n jm*⟩ $$ (*im*,*0*)
using *id-def id0 im-def* **by** *simp*
also have . . . = *1* ∗ |*state-basis n jm*⟩ $$ (*im*,*0*) **using** *mat-of-cols-list-def* **by** *auto*
also have . . . = |*state-basis* (*Suc n*) *jm*⟩ $$ (*im*,*0*) **using** *iim jjm state-basis-def*
by (*smt* (*verit*, *best*) *il im-def index-unit-vec*(*3*) *index-vec ket-vec-index lambda-one mod-less-divisor pos2 unit-vec-def zero-less-power*)
also have . . . = |*state-basis* (*Suc n*) *j*⟩ $$ (*i*,*0*) **using** *iim jjm* **by** *simp*
finally show *?thesis* **by** *this*
next
assume *id1*:*id* = *1*
hence *iid*:*i* = *2^n* + *im* **using** *i-dec* **by** *simp*
have *jma*:*jm* ≠ *2^n* + *im* **using** *jml iid* **by** *auto*
have *mat-of-cols-list 2* [[*1*,*0*]] $$ (*i div 2^n*,*0*) ∗ |*state-basis n jm*⟩ $$ (*i mod 2^n*,*0*)
= *mat-of-cols-list 2* [[*1*,*0*]] $$ (*1*,*0*) ∗ |*state-basis n jm*⟩ $$ (*im*,*0*)
using *id1 id-def im-def* **by** *simp*
also have . . . = *0* **using** *mat-of-cols-list-def* **by** *auto*
also have . . . = |*state-basis* (*Suc n*) *jm*⟩ $$ (*2^n* + *im*,*0*)
proof −
have |*state-basis* (*Suc n*) *jm*⟩ $$ (*2^n* + *im*,*0*) =
|*unit-vec* (*2^*(*Suc n*)) *jm*⟩ $$ (*2^n*+*im*,*0*)
using *state-basis-def* **by** *simp*
also have . . . = *Matrix.mat* (*2^*(*Suc n*)) *1* (λ(*i*, *j*). (*unit-vec* (*2^*(*Suc n*)) *jm*) $ *i*)
$$ (*2^n*+*im*,*0*)
using *ket-vec-def* **by** *simp*
also have . . . = *Matrix.mat* (*2^*(*Suc n*)) *1* (λ(*i*,*j*). *Matrix.vec* (*2^*(*Suc n*))
(λ*j′*. *if j′*=*jm then 1 else 0*) $ *i*) $$ (*2^n*+*im*,*0*)
using *unit-vec-def* **by** *metis*
also have . . . = *0* **using** *iid il jma* **by** *fastforce*
finally show *?thesis* **by** *auto*
qed
also have . . . = |*state-basis* (*Suc n*) *j*⟩ $$ (*i*,*0*) **using** *jjm iid* **by** *simp*
finally show *?thesis* **by** *this*
qed
qed
finally show *?thesis* **using** *ja0* **by** *auto*

qed
next
show *dim-row* (|*state-basis 1* (*j div 2* ^ *n*)⟩ ⨂ |*state-basis n* (*j mod 2* ^ *n*)⟩)
=
dim-row |*state-basis* (*Suc n*) *j*⟩
using *state-basis-def state-basis-carrier-mat ket-vec-def* **by** *auto*
next
show *dim-col* (|*state-basis 1* (*j div 2* ^ *n*)⟩ ⨂ |*state-basis n* (*j mod 2* ^ *n*)⟩)
=
dim-col |*state-basis* (*Suc n*) *j*⟩
using *state-basis-def state-basis-carrier-mat ket-vec-def* **by** *auto*
qed
next
assume *jd1*:*jd* = *1*
hence *j-dec2*:*j* = *2*^*n* + *jm* **using** *j-dec* **by** *auto*
show |*state-basis 1* (*j div 2* ^ *n*)⟩ ⨂ |*state-basis n* (*j mod 2* ^ *n*)⟩ = |*state-basis* (*Suc n*) *j*⟩
proof
fix *i ja*
assume *i* < *dim-row* |*state-basis* (*Suc n*) *j*⟩
hence *il*:*i* < *2*^(*Suc n*) **using** *state-basis-def state-basis-carrier-mat ket-vec-def* **by** *simp*
assume *ja* < *dim-col* |*state-basis* (*Suc n*) *j*⟩
hence *jal*:*ja* < *1* **using** *state-basis-def state-basis-carrier-mat ket-vec-def* **by** *simp*
hence *ja0*:*ja* = *0* **by** *auto*
show (|*state-basis 1* (*j div 2* ^ *n*)⟩ ⨂ |*state-basis n* (*j mod 2* ^ *n*)⟩) $$ (*i*, *ja*) =
|*state-basis* (*Suc n*) *j*⟩ $$ (*i*, *ja*)
proof −
have (|*state-basis 1 jd*⟩ ⨂ |*state-basis n jm*⟩) $$ (*i*, *0*) =
(|*state-basis 1 1*⟩ ⨂ |*state-basis n jm*⟩) $$ (*i*, *0*)
using *jd1* **by** *simp*
also have ... = |*state-basis 1 1*⟩ $$
(*i div* (*dim-row* |*state-basis n jm*⟩), *0 div* (*dim-col* |*state-basis n jm*⟩)) ∗
|*state-basis n jm*⟩ $$
(*i mod* (*dim-row* |*state-basis n jm*⟩), *0 mod* (*dim-col* |*state-basis n jm*⟩))
proof (*rule index-tensor-mat*)
show *dim-row* |*state-basis 1 1*⟩ = *2*
using *state-basis-carrier-mat state-basis-def ket-vec-def* **by** *simp*
show *dim-col* |*state-basis 1 1*⟩ = *1*
using *state-basis-carrier-mat state-basis-def ket-vec-def* **by** *simp*
show *dim-row* |*state-basis n jm*⟩ = *dim-row* |*state-basis n jm*⟩ **by** *auto*
show *dim-col* |*state-basis n jm*⟩ = *dim-col* |*state-basis n jm*⟩ **by** *auto*
show *i* < *2* ∗ *dim-row* |*state-basis n jm*⟩
using *state-basis-carrier-mat state-basis-def ket-vec-def il* **by** *auto*
show *0* < *1* ∗ *dim-col* |*state-basis n jm*⟩

```
        using state-basis-carrier-mat state-basis-def ket-vec-def by auto
      show 0 < (1::nat) by simp
      show 0 < dim-col |state-basis n jm⟩
        using state-basis-carrier-mat state-basis-def ket-vec-def by auto
    qed
    also have ... = (mat-of-cols-list 2 [[0,1]]) $$ (i div 2^n,0) *
                |state-basis n jm⟩ $$ (i mod 2^n,0)
     using state-basis-carrier-mat state-basis-def ket-vec-def mat-of-cols-list-def
        ket-one-to-mat-of-cols-list
      by auto
    also have ... = |state-basis (Suc n) j⟩ $$ (i,0)
    proof −
      define id im where id = i div 2^n and im = i mod 2^n
      have i-dec:i = id*(2^n) + im using id-def im-def by presburger
      show ?thesis
      proof (rule disjE)
        show id = 0 ∨ id = 1 using id-def il
      by (metis One-nat-def less-2-cases less-power-add-imp-div-less plus-1-eq-Suc

            power-one-right)
      next
        assume id0:id = 0
        hence iim:i = im using i-dec by presburger
        have mat-of-cols-list 2 [[0,1]] $$ (i div 2^n,0) * |state-basis n jm⟩ $$ (i
mod 2^n,0)
            = mat-of-cols-list 2 [[0,1]] $$ (0,0) * |state-basis n jm⟩ $$ (im,0)
          using id0 id-def im-def by simp
        also have ... = 0 using mat-of-cols-list-def by auto
        also have ... = |state-basis (Suc n) j⟩ $$ (im,0)
          using state-basis-def ket-vec-def j-dec2 assms id0 iim il local.id-def by
force
        also have ... = |state-basis (Suc n) j⟩ $$ (i,0) using iim by simp
        finally show ?thesis by this
      next
        assume id1:id = 1
        hence i2m:i = 2^n + im using i-dec by presburger
        have mat-of-cols-list 2 [[0,1]] $$ (i div 2^n,0) * |state-basis n jm⟩ $$ (i
mod 2^n,0)
            = mat-of-cols-list 2 [[0,1]] $$ (1,0) * |state-basis n jm⟩ $$ (im,0)
          using id1 id-def im-def by simp
         also have ... = |state-basis n jm⟩ $$ (im,0) using mat-of-cols-list-def
by auto
        also have ... = |state-basis (Suc n) j⟩ $$ (i,0)
          using i2m j-dec2 il assms state-basis-def by auto
        finally show ?thesis by this
      qed
    qed
   finally show ( |state-basis 1 (j div 2 ^ n)⟩ ⨂ |state-basis n (j mod 2 ^ n)⟩)
$$ (i, ja) =
```

```
                    |state-basis (Suc n) j⟩ $$ (i, ja)
          using ja0 jd-def jm-def by auto
      qed
    next
      show dim-row ( |state-basis 1 (j div 2 ^ n)⟩ ⊗ |state-basis n (j mod 2 ^ n)⟩)
=
            dim-row |state-basis (Suc n) j⟩
        using state-basis-def state-basis-carrier-mat ket-vec-def by simp
    next
      show dim-col ( |state-basis 1 (j div 2 ^ n)⟩ ⊗ |state-basis n (j mod 2 ^ n)⟩)
=
            dim-col |state-basis (Suc n) j⟩
        using state-basis-def state-basis-carrier-mat ket-vec-def by simp
    qed
  qed
qed

lemma state-basis-dec':
  ∀ j. j < 2 ^ Suc n ⟶
    |state-basis n (j div 2)⟩ ⊗ |state-basis 1 (j mod 2)⟩ = |state-basis (Suc n) j⟩
proof (induct n)
  case 0
  show ?case
  proof
    fix j::nat
    show j < 2 ^ Suc 0 ⟶
         |state-basis 0 (j div 2)⟩ ⊗ |state-basis 1 (j mod 2)⟩ = |state-basis (Suc 0)
j⟩
    proof
      assume j < 2 ^ Suc 0
      hence j2:j < 2 by auto
      hence jd0:j div 2 = 0 by auto
      have jmj:j mod 2 = j using j2 by auto
      have |state-basis 0 (j div 2)⟩ ⊗ |state-basis 1 (j mod 2)⟩ =
            |state-basis 0 0⟩ ⊗ |state-basis 1 j⟩
        using jmj jd0 by simp
      also have ... = (1_m 1) ⊗ |state-basis 1 j⟩
        using state-basis-def unit-vec-def ket-vec-def by auto
      also have ... = |state-basis 1 j⟩ using left-tensor-id by blast
     finally show |state-basis 0 (j div 2)⟩ ⊗ |state-basis 1 (j mod 2)⟩ = |state-basis
(Suc 0) j⟩
        by auto
    qed
  qed
next
  case (Suc n)
  assume HI:∀ j<2 ^ Suc n. |state-basis n (j div 2)⟩ ⊗ |state-basis 1 (j mod 2)⟩
=
                          |state-basis (Suc n) j⟩
```

define *m* **where** $m = Suc\ n$
show *?case*
proof
fix *j*::*nat*
show $j < 2 \ \hat{}\ Suc\ (Suc\ n) \longrightarrow$
$|state\text{-}basis\ (Suc\ n)\ (j\ div\ 2)\rangle \bigotimes |state\text{-}basis\ 1\ (j\ mod\ 2)\rangle = |state\text{-}basis\ (Suc\ (Suc\ n))\ j\rangle$
proof
assume $jleq{:}j < 2 \ \hat{}\ Suc\ (Suc\ n)$
define *jd2* **where** $jd2 = j\ div\ 2$
define *jm2* **where** $jm2 = j\ mod\ 2$
define *jd2m* **where** $jd2m = j\ div\ 2\hat{}m$
define *jm2m* **where** $jm2m = j\ mod\ 2\hat{}m$
define *jmm* **where** $jmm = jd2\ mod\ 2\hat{}n$
have $|state\text{-}basis\ m\ jd2\rangle \bigotimes |state\text{-}basis\ 1\ jm2\rangle =$
$(\ |state\text{-}basis\ 1\ jd2m\rangle \bigotimes |state\text{-}basis\ n\ jmm\rangle) \bigotimes |state\text{-}basis\ 1\ jm2\rangle$
using *jleq state-basis-dec m-def jd2-def jm2-def jd2m-def jmm-def jm2-def*
by (*metis Suc-eq-plus1 div-exp-eq less-power-add-imp-div-less plus-1-eq-Suc power-one-right*)
also have $\ldots = |state\text{-}basis\ 1\ jd2m\rangle \bigotimes (\ |state\text{-}basis\ n\ jmm\rangle \bigotimes |state\text{-}basis\ 1\ jm2\rangle)$
using *tensor-mat-is-assoc* **by** *presburger*
also have $\ldots = |state\text{-}basis\ 1\ jd2m\rangle \bigotimes |state\text{-}basis\ m\ jm2m\rangle$
using *HI jm2m-def jmm-def jm2-def*
by (*metis Suc-eq-plus1 div-exp-mod-exp-eq jd2-def le-simps*(2) *less-add-same-cancel2 m-def*
mod-less-divisor mod-mod-power-cancel plus-1-eq-Suc pos2 power-one-right zero-less-Suc
zero-less-power)
also have $\ldots = |state\text{-}basis\ (Suc\ m)\ j\rangle$
using *state-basis-dec m-def jleq jd2m-def jm2m-def* **by** *presburger*
finally show $|state\text{-}basis\ (Suc\ n)\ (j\ div\ 2)\rangle \bigotimes |state\text{-}basis\ 1\ (j\ mod\ 2)\rangle =$
$|state\text{-}basis\ (Suc\ (Suc\ n))\ j\rangle$
using *jd2-def jm2-def m-def* **by** *simp*
qed
qed
qed

Action of the H gate in the circuit

lemma *H-on-first-qubit*:
assumes $j < 2 \ \hat{}\ Suc\ n$
shows $((H \bigotimes ((1_m\ (2\hat{}n))))) * |state\text{-}basis\ (Suc\ n)\ j\rangle =$
$1/sqrt\ 2 \cdot_m (\ |zero\rangle + exp(2{*}i{*}pi{*}(complex\text{-}of\text{-}nat\ (j\ div\ 2\hat{}n))/2) \cdot_m |one\rangle) \bigotimes$
$|state\text{-}basis\ n\ (j\ mod\ 2\hat{}n)\rangle$
proof −
define *jd jm* **where** $jd = j\ div\ 2\hat{}n$ **and** $jm = j\ mod\ 2\hat{}n$
have $((H \bigotimes ((1_m\ (2\hat{}n))))) * |state\text{-}basis\ (Suc\ n)\ j\rangle =$
$((H \bigotimes ((1_m\ (2\hat{}n))))) * (\ |state\text{-}basis\ 1\ jd\rangle \bigotimes |state\text{-}basis\ n\ jm\rangle)$

using *jd-def jm-def state-basis-dec assms* **by** *simp*
also have ... = $(H * |state\text{-}basis\ 1\ jd\rangle) \bigotimes ((1_m\ (2\hat{}n)) * |state\text{-}basis\ n\ jm\rangle)$
using *H-def state-basis-carrier-mat state-basis-def ket-vec-def mult-distr-tensor*
by (*metis* (*no-types*, *lifting*) *H-without-scalar-prod carrier-matD*(*1*) *dim-col-mat*(*1*)

index-one-mat(*3*) *pos2 power-one-right zero-less-one-class.zero-less-one zero-less-power*)
also have ... = $1/sqrt\ 2 \cdot_m (|zero\rangle + exp(2*\mathrm{i}*pi*(complex\text{-}of\text{-}nat\ jd)/2) \cdot_m |one\rangle) \bigotimes$
$|state\text{-}basis\ n\ jm\rangle$
proof −
have $0{:}1_m\ (2\ \hat{}\ n) * |state\text{-}basis\ n\ jm\rangle = |state\text{-}basis\ n\ jm\rangle$
using *left-mult-one-mat state-basis-carrier-mat* **by** *metis*
have $H * |state\text{-}basis\ 1\ jd\rangle =$
$1/sqrt\ 2 \cdot_m (|zero\rangle + exp(2*\mathrm{i}*pi*(complex\text{-}of\text{-}nat\ jd)/2) \cdot_m |one\rangle)$
proof (*rule disjE*)
show $jd = 0 \vee jd = 1$ **using** *jd-def assms* **by** (*metis One-nat-def less-2-cases*

less-power-add-imp-div-less plus-1-eq-Suc power-one-right)
next
assume $jd0{:}jd = 0$
have $H * |state\text{-}basis\ 1\ 0\rangle =$
mat-of-cols-list 2 (*map* (*map complex-of-real*) [[$1\ /\ sqrt\ 2$, $1\ /\ sqrt\ 2$]])
using *H-on-ket-zero state-basis-def* **by** *auto*
also have ... = $1/sqrt\ 2 \cdot_m (|zero\rangle + exp(2*\mathrm{i}*pi*(complex\text{-}of\text{-}nat\ 0)/2) \cdot_m |one\rangle)$
proof
fix *i j*
assume $ai{:}i < dim\text{-}row\ ((1/sqrt\ 2) \cdot_m (|zero\rangle + exp\ (2*\mathrm{i}*pi*complex\text{-}of\text{-}nat\ 0/2) \cdot_m |one\rangle))$
hence $i < 2$ **using** *mat-of-cols-list-def smult-carrier-mat ket-vec-def* **by** *simp*
hence $i2{:}i \in \{0,1\}$ **by** *auto*
assume $aj{:}j < dim\text{-}col\ ((1/sqrt\ 2) \cdot_m (|zero\rangle + exp\ (2*\mathrm{i}*pi*complex\text{-}of\text{-}nat\ 0/2) \cdot_m |one\rangle))$
hence $j0{:}j = 0$ **using** *mat-of-cols-list-def smult-carrier-mat ket-vec-def* **by** *simp*
have (*mat-of-cols-list 2* (*map* (*map complex-of-real*) [[$1\ /\ sqrt\ 2$, $1\ /\ sqrt\ 2$]])) $$ $(i,0)$ =
(*mat-of-cols-list 2* [[$1/sqrt\ 2$, $1/sqrt\ 2$]]) $$ $(i,0)$
using *map-def* **by** *simp*
also have ... = $1/sqrt\ 2$ **using** *i2 index-mat-of-cols-list* **by** *auto*
also have ... = ($1/sqrt\ 2 \cdot_m$ (*mat-of-cols-list 2* [[$1,1$]])) $$ $(i,0)$
using *smult-mat-def mat-of-cols-list-def index-mat-of-cols-list*
by (*smt* (*verit*, *best*) *Suc-1* ‹$i < 2$› *dim-col-mat*(*1*) *dim-row-mat*(*1*) *index-smult-mat*(*1*)
ket-one-is-state ket-one-to-mat-of-cols-list less-Suc-eq-0-disj less-one list.size(*4*)
mult.right-neutral nth-Cons-0 nth-Cons-Suc state-def)

```
also have ... = (1/sqrt 2 ·m ( |zero⟩ + |one⟩)) $$ (i,0)
proof −
  have mat-of-cols-list 2 [[1,1]] = |zero⟩ + |one⟩
  proof
    fix i j::nat
     define s1 s2 where s1 = mat-of-cols-list 2 [[1,1]] and s2 = |zero⟩ +
|one⟩
    assume i < dim-row s2 and j < dim-col s2
    hence i ∈ {0,1} ∧ j = 0 using index-add-mat
      by (simp add: ket-vec-def less-Suc-eq numerals(2) s2-def)
    thus s1 $$ (i,j) = s2 $$ (i,j) using s1-def s2-def mat-of-cols-list-def
          ‹i < dim-row s2› ket-one-to-mat-of-cols-list by force
  next
     define s1 s2 where s1 = mat-of-cols-list 2 [[1,1]] and s2 = |zero⟩ +
|one⟩
    thus dim-row s1 = dim-row s2 using mat-of-cols-list-def by (simp add:
ket-vec-def)
  next
     define s1 s2 where s1 = mat-of-cols-list 2 [[1,1]] and s2 = |zero⟩ +
|one⟩
     thus dim-col s1 = dim-col s2 using mat-of-cols-list-def by (simp add:
ket-vec-def)
  qed
  thus ?thesis by simp
qed
also have ... = (1/sqrt 2 ·m ( |zero⟩ + 1 ·m |one⟩)) $$ (i,0)
  using smult-mat-def ‹i < 2› ket-one-is-state state-def by force
also have ... = (1/sqrt 2 ·m ( |zero⟩ + exp (2∗i∗pi∗(complex-of-nat 0)/2)
·m |one⟩)) $$ (i,0)
  by auto
finally show Tensor.mat-of-cols-list 2 (map (map complex-of-real)
              [[1 / sqrt 2, 1 / sqrt 2]]) $$ (i, j) =
              (complex-of-real (1 / sqrt 2) ·m ( |Deutsch.zero⟩ +
                   exp (2 ∗ i ∗ complex-of-real pi ∗ complex-of-nat 0 / 2) ·m
|Deutsch.one⟩)) $$
               (i, j)
  using j0 i2 ai aj by auto
next
  show dim-row (Tensor.mat-of-cols-list 2 (map (map complex-of-real)
        [[1 / sqrt 2, 1 / sqrt 2]])) = dim-row (complex-of-real (1 / sqrt 2) ·m
         ( |Deutsch.zero⟩ + exp (2 ∗ i ∗ complex-of-real pi ∗ complex-of-nat 0
/2) ·m
          |Deutsch.one⟩))
  using mat-of-cols-list-def index-mat-of-cols-list smult-carrier-mat ket-vec-def
by auto
next
  show dim-col (Tensor.mat-of-cols-list 2 (map (map complex-of-real)
        [[1 / sqrt 2, 1 / sqrt 2]])) = dim-col (complex-of-real (1 / sqrt 2) ·m
         ( |Deutsch.zero⟩ + exp (2 ∗ i ∗ complex-of-real pi ∗ complex-of-nat 0
```

/2) $\cdot_m$
|*Deutsch.one*⟩))
using *mat-of-cols-list-def index-mat-of-cols-list smult-carrier-mat ket-vec-def* **by** *auto*
qed
finally show *?thesis* **using** *jd0* **by** *simp*
next
assume *jd1*:*jd* = *1*
have *H* ∗ |*state-basis 1 1*⟩ =
mat-of-cols-list 2 (*map* (*map complex-of-real*) [[*1* / *sqrt 2*, − *1* / *sqrt 2*]])
using *H-on-ket-one map-def* **by** (*simp add*: *state-basis-def*)
also have . . . = (*1* / *sqrt 2*) $\cdot_m$ (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat 1* / *2*) $\cdot_m$ |*one*⟩)
proof
fix *i j*
assume *ai*:*i* < *dim-row* (*complex-of-real* (*1* / *sqrt 2*) $\cdot_m$ (|*zero*⟩ +
exp (*2*∗i∗*complex-of-real pi* ∗*complex-of-nat 1* /*2*) $\cdot_m$ |*one*⟩))
hence *i* < *2* **using** *mat-of-cols-list-def smult-carrier-mat ket-vec-def* **by** *simp*
hence *i2*:*i* ∈ {*0*,*1*} **by** *auto*
assume *aj*:*j* < *dim-col* (*complex-of-real* (*1* / *sqrt 2*) $\cdot_m$ (|*zero*⟩ +
exp (*2*∗i∗*complex-of-real pi* ∗*complex-of-nat 1* /*2*) $\cdot_m$ |*one*⟩))
hence *j0*:*j* = *0* **using** *mat-of-cols-list-def smult-carrier-mat ket-vec-def* **by** *simp*
have (*mat-of-cols-list 2* (*map* (*map complex-of-real*) [[*1* / *sqrt 2*,−*1* / *sqrt 2*]])) $$ (*i*,*0*) =
(*mat-of-cols-list 2* [[*1*/*sqrt 2*,− *1*/*sqrt 2*]]) $$ (*i*,*0*)
using *map-def* **by** *simp*
also have . . . = ((*1*/*sqrt 2*) $\cdot_m$ (*mat-of-cols-list 2* [[*1*,−*1*]])) $$ (*i*,*0*)
using *i2 smult-mat-def index-mat-of-cols-list mat-of-cols-list-def Suc-1* ‹*i* < *2*›
dim-col-mat(*1*) *dim-row-mat*(*1*) *index-smult-mat*(*1*) *nth-Cons-0 nth-Cons-Suc*
ket-one-is-state ket-one-to-mat-of-cols-list
by (*smt* (*z3*) *One-nat-def* ψ_0-*to*-ψ_1 *bot-nat-0*.*not-eq-extremum dim-col-tensor-mat*

less-2-cases-iff list.*map*(*2*) *list*.*size*(*4*) *mult-0-right mult-1 of-real-1*
of-real-divide of-real-minus state-def times-divide-eq-left)
also have . . . = (*1*/*sqrt 2* $\cdot_m$ (|*zero*⟩ − |*one*⟩)) $$ (*i*,*0*)
proof −
define *r1 r2* **where** *r1* = *mat-of-cols-list 2* [[*1*,−*1*]] **and** *r2* = |*zero*⟩ − |*one*⟩
have *r1* $$ (*0*,*0*) = *r2* $$ (*0*,*0*) **using** *r1-def r2-def mat-of-cols-list-def*
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def add*.*commute diff-zero dim-row-mat*(*1*)
index-mat(*1*) *index-mat-of-cols-list ket-one-is-state ket-one-to-mat-of-cols-list*

ket-zero-to-mat-of-cols-list list.*size*(*3*) *list*.*size*(*4*) *minus-mat-def nth-Cons-0*
plus-1-eq-Suc pos2 state-def zero-less-one-class.*zero-less-one*)

moreover have *r1* $$ (*1*,*0*) = *r2* $$ (*1*,*0*)
using *r1-def r2-def mat-of-cols-list-def ket-vec-def* **by** *simp*
ultimately show *?thesis* **using** *r1-def r2-def i2*
by (*smt* (*verit*) *One-nat-def Tensor.mat-of-cols-list-def* ‹*i* < *2*› *add.commute*

dim-col-mat(*1*) *dim-row-mat*(*1*) *empty-iff index-smult-mat*(*1*) *index-unit-vec*(*3*)
insert-iff ket-vec-def list.size(*3*) *list.size*(*4*) *minus-mat-def plus-1-eq-Suc*

zero-less-one-class.zero-less-one)
qed
also have ... = (*1*/*sqrt 2* $\cdot_m$ (|*zero*⟩ + (−*1*) $\cdot_m$ |*one*⟩)) $$ (*i*,*0*)
using *smult-mat-def* ‹*i* < *2*› *ket-one-is-state state-def* **by** *force*
also have ... = (*1*/*sqrt 2* $\cdot_m$ (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat 1* / *2*) $\cdot_m$ |*one*⟩)) $$ (*i*,*0*)
using *exp-pi-i′* **by** *auto*
finally show *mat-of-cols-list 2* (*map* (*map complex-of-real*) [[*1*/*sqrt 2*,−*1*/*sqrt 2*]]) $$ (*i*,*j*)
= (*complex-of-real* (*1* / *sqrt 2*) $\cdot_m$ (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat 1* /*2*) $\cdot_m$
|*one*⟩)) $$ (*i*, *j*) **using** *i2 ai aj j0* **by** *auto*
next
show *dim-row* (*Tensor.mat-of-cols-list 2* (*map* (*map complex-of-real*)
[[*1* / *sqrt 2*,− *1* / *sqrt 2*]])) = *dim-row* (*complex-of-real* (*1* / *sqrt 2*) $\cdot_m$
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat 1* /*2*) $\cdot_m$
|*Deutsch.one*⟩))
using *mat-of-cols-list-def index-mat-of-cols-list smult-carrier-mat ket-vec-def* **by** *auto*
next
show *dim-col* (*Tensor.mat-of-cols-list 2* (*map* (*map complex-of-real*)
[[*1* / *sqrt 2*,− *1* / *sqrt 2*]])) = *dim-col* (*complex-of-real* (*1* / *sqrt 2*) $\cdot_m$
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat 1* /*2*) $\cdot_m$
|*Deutsch.one*⟩))
using *mat-of-cols-list-def index-mat-of-cols-list smult-carrier-mat ket-vec-def* **by** *auto*
qed
finally show *?thesis* **using** *jd1* **by** *simp*
qed
hence (*H* ∗ |*state-basis 1 jd*⟩) ⨂ |*state-basis n jm*⟩ =
(*1*/*sqrt 2* $\cdot_m$ ((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗(*complex-of-nat jd*)/*2*) $\cdot_m$ |*one*⟩))) ⨂ |*state-basis n jm*⟩
by *simp*
thus *?thesis* **using** *0* **by** *presburger*
qed
finally show *?thesis* **using** *jm-def jd-def* **by** *auto*
qed

Action of the R gate in the circuit

lemma *R-action*:
assumes $j < 2$ ^ *Suc n* **and** *j mod 2 = 1*
shows (*R* (*Suc n*)) ∗ (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) $\cdot_m$ |*one*⟩) =
|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^(*Suc n*)) $\cdot_m$ |*one*⟩
proof
fix *i ja*::*nat*
assume *i* < *dim-row* (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^(*Suc n*)) $\cdot_m$ |*one*⟩)
hence *il2*:*i* < *2* **by** (*simp add*: *ket-vec-def*)
assume *ja* < *dim-col* (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^(*Suc n*)) $\cdot_m$ |*one*⟩)
hence *ja0*:*ja* = *0* **by** (*simp add*: *ket-vec-def*)
have (*R* (*Suc n*)) ∗ (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) $\cdot_m$ |*one*⟩) =
(*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp*(*2*∗*pi*∗i/*2*^(*Suc n*))]]) ∗
(|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) $\cdot_m$ |*one*⟩)
using *R-def* **by** *simp*
also have . . . = (*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp*(*2*∗*pi*∗i/*2*^(*Suc n*))]]) ∗
(*mat-of-cols-list 2* [[*1*,*0*]] +
exp (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) $\cdot_m$ *mat-of-cols-list 2* [[*0*,*1*]])
using *ket-one-to-mat-of-cols-list ket-zero-to-mat-of-cols-list* **by** *presburger*
also have . . . = (*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp*(*2*∗*pi*∗i/*2*^(*Suc n*))]]) ∗
(*mat-of-cols-list 2* [[*1*,*0*]] +
mat-of-cols-list 2 [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]])
proof −
have *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) $\cdot_m$ *mat-of-cols-list 2* [[*0*,*1*]] =
mat-of-cols-list 2 [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]]
proof
fix *a b*::*nat*
assume *a* < *dim-row* (*mat-of-cols-list 2* [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]])
hence *a2*:*a* < *2* **by** (*simp add*: *Tensor.mat-of-cols-list-def*)
assume *b* < *dim-col* (*mat-of-cols-list 2* [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]])
hence *b0*:*b* = *0*
by (*metis One-nat-def Suc-eq-plus1 Tensor.mat-of-cols-list-def dim-col-mat*(*1*) *less-Suc0*
list.size(*3*) *list.size*(*4*))
have (*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) $\cdot_m$ *mat-of-cols-list 2* [[*0*,*1*]]) $$ (*a*,*0*) =
exp (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) ∗ (*mat-of-cols-list 2* [[*0*,*1*]] $$ (*a*,*0*))
using *index-smult-mat a2 ket-one-is-state ket-one-to-mat-of-cols-list state-def* **by** *force*
also have . . . = (*mat-of-cols-list 2* [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]]) $$ (*a*,*0*)

proof (*rule disjE*)
show *a* = *0* ∨ *a* = *1* **using** *a2* **by** *auto*
next
assume *a0*:*a* = *0*
have *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) ∗ (*mat-of-cols-list 2* [[*0*,*1*]] $$ (*0*,*0*)) =
exp (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) ∗ *0*
using *index-mat-of-cols-list* **by** *auto*
thus *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) ∗ (*mat-of-cols-list 2* [[*0*,*1*]] $$ (*a*,*0*)) =
(*mat-of-cols-list 2* [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]]) $$ (*a*,*0*)
using *a0* **by** *auto*
next
assume *a1*:*a* = *1*
have *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) ∗ (*mat-of-cols-list 2* [[*0*,*1*]] $$ (*1*,*0*)) =
exp (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) ∗ *1*
using *index-mat-of-cols-list* **by** *auto*
thus *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) ∗ (*mat-of-cols-list 2* [[*0*,*1*]] $$ (*a*,*0*)) =
(*mat-of-cols-list 2* [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]]) $$ (*a*,*0*)
using *a1* **by** *auto*
qed
finally show (*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*) $\cdot_m$ *mat-of-cols-list 2* [[*0*,*1*]])
$$ (*a*,*b*) = (*mat-of-cols-list 2* [[*0*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]]) $$ (*a*,*b*)
using *b0* **by** *simp*
next
show *dim-row* (*exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *n*) $\cdot_m$
Tensor.*mat-of-cols-list 2* [[*0*, *1*]]) =
dim-row (*Tensor*.*mat-of-cols-list 2* [[*0*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]])
by (*simp add*: *Tensor*.*mat-of-cols-list-def*)
next
show *dim-col* (*exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *n*) $\cdot_m$
Tensor.*mat-of-cols-list 2* [[*0*, *1*]]) =
dim-col (*Tensor*.*mat-of-cols-list 2* [[*0*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]])
by (*simp add*: *mat-of-cols-list-def*)
qed
thus *?thesis* **by** *auto*
qed
also have ... = (*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp*(*2*∗*pi*∗i/*2*^(*Suc n*))]]) ∗
(*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^*n*)]])

```
  proof -
    have mat-of-cols-list 2 [[1,0]] +
          mat-of-cols-list 2 [[0,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]] =
          mat-of-cols-list 2 [[1,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]
    proof
      fix a b::nat
      assume a < dim-row (mat-of-cols-list 2 [[1,exp (2*i*pi*complex-of-nat (j div
2) / 2^n)]])
      hence a2:a < 2 using mat-of-cols-list-def by simp
      assume b < dim-col (mat-of-cols-list 2 [[1,exp (2*i*pi*complex-of-nat (j div
2) / 2^n)]])
      hence b0:b = 0 using mat-of-cols-list-def by auto
      show (mat-of-cols-list 2 [[1,0]] +
            mat-of-cols-list 2 [[0,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(a,b) =
           (mat-of-cols-list 2 [[1,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(a,b)
      proof (rule disjE)
        show a = 0 ∨ a = 1 using a2 by auto
      next
        assume a0:a = 0
        have (mat-of-cols-list 2 [[1,0]] +
              mat-of-cols-list 2 [[0,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(0,0) =
             (mat-of-cols-list 2 [[1,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(0,0)
          using index-mat-of-cols-list by (simp add: Tensor.mat-of-cols-list-def)
        thus (mat-of-cols-list 2 [[1,0]] +
              mat-of-cols-list 2 [[0,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(a,b) =
             (mat-of-cols-list 2 [[1,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(a,b)
          using a0 b0 by simp
      next
        assume a1:a = 1
        show (mat-of-cols-list 2 [[1,0]] +
              mat-of-cols-list 2 [[0,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(a,b) =
             (mat-of-cols-list 2 [[1,exp (2*i*pi*complex-of-nat (j div 2) / 2^n)]]) $$
(a,b)
          using a1 b0 index-mat-of-cols-list mat-of-cols-list-def by simp
      qed
    next
      show dim-row (Tensor.mat-of-cols-list 2 [[1, 0]] + Tensor.mat-of-cols-list 2
           [[0, exp (2 * i * complex-of-real pi * complex-of-nat (j div 2) / 2 ^ n)]])
=
          dim-row (Tensor.mat-of-cols-list 2 [[1, exp (2 * i * complex-of-real pi *
                  complex-of-nat (j div 2) / 2 ^ n)]])
        by (simp add: Tensor.mat-of-cols-list-def)
```

```
    next
      show dim-col (Tensor.mat-of-cols-list 2 [[1, 0]] + Tensor.mat-of-cols-list 2
            [[0, exp (2 ∗ i ∗ complex-of-real pi ∗ complex-of-nat (j div 2) / 2 ^ n)]])
=
            dim-col (Tensor.mat-of-cols-list 2 [[1, exp (2 ∗ i ∗ complex-of-real pi ∗
                  complex-of-nat (j div 2) / 2 ^ n)]])
        by (simp add: mat-of-cols-list-def)
    qed
    thus ?thesis by simp
  qed
  finally have 1:R (Suc n) ∗ ( |Deutsch.zero⟩ + exp (2 ∗ i ∗ complex-of-real pi ∗
                complex-of-nat (j div 2) / 2 ^ n) ·m |Deutsch.one⟩) =
                Tensor.mat-of-cols-list 2 [[1, 0], [0, exp (complex-of-real (2 ∗ pi) ∗
i /
              2 ^ Suc n)]] ∗ Tensor.mat-of-cols-list 2 [[1, exp (2 ∗ i ∗ complex-of-real
pi ∗
                complex-of-nat (j div 2) / 2 ^ n)]]
    by this
  show (R (Suc n) ∗ ( |Deutsch.zero⟩ + exp (2 ∗ i ∗ pi ∗  complex-of-nat (j div 2)
/  2 ^ n) ·m
        |Deutsch.one⟩)) $$ (i, ja) =
        ( |Deutsch.zero⟩ + exp (2 ∗ i ∗ complex-of-real pi ∗ complex-of-nat j / 2 ^
Suc n) ·m
        |Deutsch.one⟩) $$ (i, ja)
  proof −
    have ((R (Suc n) ∗ ( |Deutsch.zero⟩ + exp (2 ∗ i ∗ pi ∗  complex-of-nat (j div
2) /  2 ^ n) ·m
          |Deutsch.one⟩))) $$ (i, ja) =
          (Tensor.mat-of-cols-list 2 [[1, 0], [0, exp (complex-of-real (2 ∗ pi) ∗ i /
              2 ^ Suc n)]] ∗ Tensor.mat-of-cols-list 2 [[1, exp (2 ∗ i ∗ complex-of-real
pi ∗
                complex-of-nat (j div 2) / 2 ^ n)]]) $$ (i,ja)
      using 1 by simp
    also have ... = mat-of-cols-list 2 [[1,exp (2∗i∗pi∗complex-of-nat j / 2^Suc n)]]
$$ (i,ja)
    proof (rule disjE)
      show i = 0 ∨ i = 1 using il2 by auto
    next
      assume i0:i = 0
      have (Tensor.mat-of-cols-list 2 [[1, 0],[0, exp (complex-of-real (2 ∗ pi) ∗ i /
2 ^ Suc n)]]
            ∗ Tensor.mat-of-cols-list 2 [[1, exp (2 ∗ i ∗ complex-of-real pi ∗
            complex-of-nat (j div 2) / 2 ^ n)]]) $$ (0, 0) =
          (∑ k<2. (mat-of-cols-list 2 [[1, 0],[0, exp (complex-of-real (2 ∗ pi) ∗ i /
2 ^ Suc n)]])
            $$ (0,k) ∗ (mat-of-cols-list 2 [[1, exp (2 ∗ i ∗ complex-of-real pi ∗
            complex-of-nat (j div 2) / 2 ^ n)]]) $$ (k,0))
        using index-mult-mat mat-of-cols-list-def by auto
      also have ... = (mat-of-cols-list 2 [[1, 0],[0, exp (complex-of-real (2 ∗ pi) ∗
```

i / *2* ^ *Suc n*)]])
$$ (*0*,*0*) ∗ (*mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*0*,*0*) +
(*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i / *2* ^ *Suc n*)]])
$$ (*0*,*1*) ∗ (*mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*1*,*0*)
by (*simp only*:*sumof2*)
also have . . . = *1* **by** *auto*
also have . . . = *mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]] $$ (*0*,*0*)
using *index-mat-of-cols-list* **by** *simp*
finally show (*Tensor*.*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i /
2 ^ *Suc n*)]] ∗ *Tensor*.*mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*i*, *ja*) =
(*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]]) $$ (*i*,*ja*)
using *i0 ja0* **by** *simp*
next
assume *i1*:*i* = *1*
have (*Tensor*.*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i / *2* ^ *Suc n*)]]
∗ *Tensor*.*mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*1*, *0*) =
(∑ *k*<*2*. (*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i / *2* ^ *Suc n*)]])
$$ (*1*,*k*) ∗ (*mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*k*,*0*))
using *index-mult-mat mat-of-cols-list-def* **by** *auto*
also have . . . = (*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i / *2* ^ *Suc n*)]])
$$ (*1*,*0*) ∗ (*mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*0*,*0*) +
(*mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i / *2* ^ *Suc n*)]])
$$ (*1*,*1*) ∗ (*mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*1*,*0*)
by (*simp only*: *sumof2*)
also have . . . = *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i / *2* ^ *Suc n*) ∗
exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *n*)
using *index-mat-of-cols-list* **by** *auto*
also have . . . = *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i / *2* ^ *Suc n* +
2 ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *n*)

using *mult-exp-exp* **by** *simp*
also have ... = *exp* (*2* ∗ i ∗ *pi* / *2* ^ *Suc n* +
2 ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *n*)
by (*simp add*: *mult.commute*)
also have ... = *exp* (*2*∗i∗*pi*∗(*1*/*2*^*Suc n* + *complex-of-nat* (*j div 2*)/*2*^*n*))
by (*simp add*: *distrib-left*)
also have ... = *exp* (*2*∗i∗*pi*∗((*1* + *2*∗(*j div 2*))/*2*^*Suc n*))
by (*simp add*: *add-divide-distrib*)
also have ... = *exp* (*2*∗i∗*pi*∗(*j*)/*2*^*Suc n*)
using *assms*
by (*smt* (*verit*, *ccfv-threshold*) *Suc-eq-plus1 div-mult-mod-eq mult.commute of-real-1*
of-real-add of-real-divide of-real-of-nat-eq of-real-power one-add-one plus-1-eq-Suc
times-divide-eq-right)
also have ... = (*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]]) $$ (*1*,*0*)
using *index-mat-of-cols-list* **by** *simp*
finally show (*Tensor.mat-of-cols-list 2* [[*1*, *0*],[*0*, *exp* (*complex-of-real* (*2* ∗ *pi*) ∗ i /
2 ^ *Suc n*)]] ∗ *Tensor.mat-of-cols-list 2* [[*1*, *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat (*j div 2*) / *2* ^ *n*)]]) $$ (*i*, *ja*) =
(*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]]) $$ (*i*,*ja*)
using *i1 ja0* **by** *simp*
qed
also have ... = (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) $$ (*i*,*ja*)
proof (*rule disjE*)
show *i* = *0* ∨ *i* = *1* **using** *il2* **by** *auto*
next
assume *i0*:*i* = *0*
have (*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]]) $$ (*0*,*0*) = *1*
by *auto*
also have ... = (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) $$ (*0*,*0*)
proof −
have |*zero*⟩ $$ (*0*,*0*) = *1* **by** *auto*
moreover have (*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) $$ (*0*,*0*) = *0*
proof −
have (*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) $$ (*0*,*0*) =
exp (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) ∗ |*one*⟩ $$ (*0*,*0*)
using *index-smult-mat* **using** *ket-one-is-state state-def* **by** *auto*
also have ... = *0* **by** *auto*
finally show *?thesis* **by** *this*
qed

ultimately show *?thesis* **by** (*simp add*: *ket-vec-def*)
qed
finally show (*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]]) \$\$ (*i*,*ja*) =
(|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) \$\$ (*i*,*ja*)
using *i0 ja0* **by** *simp*
next
assume *i1*:*i* = *1*
have (*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]]) \$\$ (*1*,*0*) =
exp (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) **by** *auto*
also have ... = (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) \$\$ (*1*,*0*)
proof −
have |*zero*⟩ \$\$ (*1*,*0*) = *0* **by** *auto*
moreover have (*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) \$\$ (*1*,*0*) =
exp (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)
proof −
have (*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) \$\$ (*1*,*0*) =
exp (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) ∗ |*one*⟩ \$\$ (*1*,*0*)
using *index-smult-mat ket-one-is-state state-def* **by** *auto*
also have ... = *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) **by** *auto*
finally show *?thesis* **by** *this*
qed
ultimately show *?thesis* **by** (*simp add*: *ket-vec-def*)
qed
finally show (*mat-of-cols-list 2* [[*1*,*exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*)]]) \$\$ (*i*,*ja*) =
(|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^*Suc n*) $\cdot_m$ |*one*⟩) \$\$ (*i*,*ja*)
using *i1 ja0* **by** *simp*
qed
finally show *?thesis* **by** *this*
qed
next
show *dim-row* (*R* (*Suc n*) ∗ (|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *n*) $\cdot_m$ |*Deutsch.one*⟩)) =
dim-row (|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat j* / *2* ^ *Suc n*) $\cdot_m$
|*Deutsch.one*⟩)
by (*simp add*: *R-def Tensor.mat-of-cols-list-def ket-vec-def*)
next
show *dim-col* (*R* (*Suc n*) ∗ (|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *n*) $\cdot_m$ |*Deutsch.one*⟩)) =
dim-col (|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat j* / *2* ^ *Suc n*) $\cdot_m$
|*Deutsch.one*⟩)

by (*simp add*: *R-def Tensor.mat-of-cols-list-def ket-vec-def*)
qed

Action of the SWAP cascades in the circuit

lemma *SWAP-up-action*:
  ∀ *j*. *j* < *2* ^(*Suc* (*Suc* *n*)) ⟶
    SWAP-up (*Suc* (*Suc* *n*)) ∗ (|*state-basis* (*Suc* *n*) (*j* *div* *2*)⟩ ⨂ |*state-basis* *1* (*j* *mod* *2*)⟩) =
    |*state-basis* *1* (*j* *mod* *2*)⟩ ⨂ |*state-basis* (*Suc* *n*) (*j* *div* *2*)⟩
proof (*induct* *n*)
  case *0*
  show *?case*
  proof
    fix *j*
    show *j* < *2* ^ *Suc* (*Suc* *0*) ⟶ *SWAP-up* (*Suc* (*Suc* *0*)) ∗ (|*state-basis* (*Suc* *0*) (*j* *div* *2*)⟩ ⨂
        |*state-basis* *1* (*j* *mod* *2*)⟩) =
        |*state-basis* *1* (*j* *mod* *2*)⟩ ⨂ |*state-basis* (*Suc* *0*) (*j* *div* *2*)⟩
    proof
      assume *j* < *2*^ *Suc* (*Suc* *0*)
      show *SWAP-up* (*Suc* (*Suc* *0*)) ∗ (|*state-basis* (*Suc* *0*) (*j* *div* *2*)⟩ ⨂ |*state-basis* *1* (*j* *mod* *2*)⟩)
          = |*state-basis* *1* (*j* *mod* *2*)⟩ ⨂ |*state-basis* (*Suc* *0*) (*j* *div* *2*)⟩
      proof −
        have *SWAP-up* (*Suc* (*Suc* *0*))∗(|*state-basis* (*Suc* *0*) (*j* *div* *2*)⟩ ⨂ |*state-basis* *1* (*j* *mod* *2*)⟩)
            = *SWAP* ∗ (|*state-basis* (*Suc* *0*) (*j* *div* *2*)⟩ ⨂ |*state-basis* *1* (*j* *mod* *2*)⟩)
          using *SWAP-up.simps* **by** *simp*
        also have . . . = |*state-basis* *1* (*j* *mod* *2*)⟩ ⨂ |*state-basis* (*Suc* *0*) (*j* *div* *2*)⟩
          using *SWAP-tensor*
          by (*metis* *One-nat-def* *power-one-right* *state-basis-carrier-mat*)
        finally show *?thesis* **by** *this*
      qed
    qed
  qed
next
  case (*Suc* *n*)
  assume *HI*:∀ *j*<*2* ^ *Suc* (*Suc* *n*).
        SWAP-up (*Suc* (*Suc* *n*)) ∗ (|*state-basis* (*Suc* *n*) (*j* *div* *2*)⟩ ⨂ |*state-basis* *1* (*j* *mod* *2*)⟩)
        = |*state-basis* *1* (*j* *mod* *2*)⟩ ⨂ |*state-basis* (*Suc* *n*) (*j* *div* *2*)⟩
  show ∀ *j*<*2* ^ *Suc* (*Suc* (*Suc* *n*)).
        SWAP-up (*Suc* (*Suc* (*Suc* *n*))) ∗ (|*state-basis* (*Suc* (*Suc* *n*)) (*j* *div* *2*)⟩ ⨂
        |*state-basis* *1* (*j* *mod* *2*)⟩) =
        |*state-basis* *1* (*j* *mod* *2*)⟩ ⨂ |*state-basis* (*Suc* (*Suc* *n*)) (*j* *div* *2*)⟩
  proof
    fix *j*::*nat*
    show *j* < *2* ^ *Suc* (*Suc* (*Suc* *n*)) ⟶
        SWAP-up (*Suc* (*Suc* (*Suc* *n*))) ∗ (|*state-basis* (*Suc* (*Suc* *n*)) (*j* *div* *2*)⟩ ⨂

|state-basis 1 (j mod 2)⟩) =
|state-basis 1 (j mod 2)⟩ $\bigotimes$ *|state-basis (Suc (Suc n)) (j div 2)⟩*
proof
assume *jl:j < 2 ^ Suc (Suc (Suc n))*
show *SWAP-up (Suc (Suc (Suc n))) ∗ (|state-basis (Suc (Suc n)) (j div 2)⟩*
$\bigotimes$
|state-basis 1 (j mod 2)⟩) =
|state-basis 1 (j mod 2)⟩ $\bigotimes$ *|state-basis (Suc (Suc n)) (j div 2)⟩*
proof −
have *SWAP-up (Suc (Suc (Suc n))) ∗ (|state-basis (Suc (Suc n)) (j div 2)⟩*
$\bigotimes$
|state-basis 1 (j mod 2)⟩) =
((SWAP $\bigotimes$ *(1_m (2^(Suc n)))) ∗ ((1_m 2)* $\bigotimes$ *(SWAP-up (Suc (Suc n))))) ∗*
(|state-basis (Suc (Suc n)) (j div 2)⟩ $\bigotimes$ *|state-basis 1 (j mod 2)⟩)*
using *SWAP-up.simps* **by** *simp*
also have *. . . = (SWAP* $\bigotimes$ *(1_m (2^(Suc n)))) ∗ (((1_m 2)* $\bigotimes$ *(SWAP-up (Suc (Suc n)))) ∗*
(|state-basis (Suc (Suc n)) (j div 2)⟩ $\bigotimes$ *|state-basis 1 (j mod 2)⟩))*
using *assoc-mult-mat*
by *(smt (verit, ccfv-threshold) Groups.mult-ac(2) Groups.mult-ac(3) One-nat-def*
SWAP-up.simps(3) SWAP-up-carrier-mat carrier-matD(2) carrier-matI dim-col-tensor-mat
dim-row-mat(1) dim-row-tensor-mat index-mult-mat(2) index-one-mat(3)

index-unit-vec(3) ket-vec-def left-mult-one-mat power-Suc2 power-one-right

state-basis-def)
also have *. . . = (SWAP* $\bigotimes$ *(1_m (2^(Suc n)))) ∗ (((1_m 2)* $\bigotimes$ *(SWAP-up (Suc (Suc n)))) ∗*
((|state-basis 1 ((j div 2) div 2^Suc n)⟩ $\bigotimes$
|state-basis (Suc n) ((j div 2) mod 2^Suc n)⟩)
$\bigotimes$ *|state-basis 1 (j mod 2)⟩))*
using *state-basis-dec*
by *(metis jl less-mult-imp-div-less power-Suc2)*
also have *. . . = (SWAP* $\bigotimes$ *(1_m (2^(Suc n)))) ∗ (((1_m 2)* $\bigotimes$ *(SWAP-up (Suc (Suc n)))) ∗*
(|state-basis 1 ((j div 2) div 2^Suc n)⟩ $\bigotimes$
(|state-basis (Suc n) ((j div 2) mod 2^Suc n)⟩
$\bigotimes$ *|state-basis 1 (j mod 2)⟩)))*
using *tensor-mat-is-assoc state-basis-carrier-mat* **by** *auto*
also have *. . . = (SWAP* $\bigotimes$ *(1_m (2^(Suc n)))) ∗ (((1_m 2)* $\bigotimes$ *(SWAP-up (Suc (Suc n)))) ∗*
(|state-basis 1 ((j div 2) div 2^Suc n)⟩ $\bigotimes$
(|state-basis (Suc n) ((j mod 2^Suc (Suc n)) div 2)⟩
$\bigotimes$ *|state-basis 1 ((j mod 2^Suc (Suc n)) mod 2)⟩)))*
using *jl power-Suc power-add power-one-right*

by (*smt* (*z3*) *Suc-1 add-0 div-Suc div-exp-mod-exp-eq lessI mod-less mod-mod-cancel*
mod-mult-self2 n-not-Suc-n odd-Suc-div-two plus-1-eq-Suc)
also have ... = (*SWAP* ⨂ (*1*$_m$ (*2*^(*Suc n*)))) ∗
(((*1*$_m$ *2*) ∗ |*state-basis 1* ((*j div 2*) *div 2*^*Suc n*)⟩) ⨂
((*SWAP-up* (*Suc* (*Suc n*)))) ∗
(|*state-basis* (*Suc n*) ((*j mod 2*^*Suc* (*Suc n*)) *div 2*)⟩
⨂ |*state-basis 1* ((*j mod 2*^*Suc* (*Suc n*)) *mod 2*)⟩)))
using *mult-distr-tensor*
by (*metis SWAP-up-carrier-mat carrier-matD*(*1*) *carrier-matD*(*2*) *index-one-mat*(*3*)
less-numeral-extra(*1*) *mod-less-divisor pos2 power-one-right state-basis-carrier-mat*

state-basis-dec′ zero-less-power)
also have ... = (*SWAP* ⨂ (*1*$_m$ (*2*^(*Suc n*)))) ∗
(|*state-basis 1* ((*j div 2*) *div 2*^*Suc n*)⟩ ⨂
(|*state-basis 1* ((*j mod 2*^*Suc* (*Suc n*)) *mod 2*)⟩ ⨂
|*state-basis* (*Suc n*) ((*j mod 2*^*Suc* (*Suc n*)) *div 2*)⟩))
using *HI*
by (*metis left-mult-one-mat mod-less-divisor pos2 power-one-right state-basis-carrier-mat*
zero-less-power)
also have ... = (*SWAP* ⨂ (*1*$_m$ (*2*^(*Suc n*)))) ∗
((|*state-basis 1* ((*j div 2*) *div 2*^*Suc n*)⟩ ⨂
|*state-basis 1* ((*j mod 2*^*Suc* (*Suc n*)) *mod 2*)⟩) ⨂
|*state-basis* (*Suc n*) ((*j mod 2*^*Suc* (*Suc n*)) *div 2*)⟩)
using *tensor-mat-is-assoc* **by** *simp*
also have ... = (*SWAP* ∗ (|*state-basis 1* ((*j div 2*) *div 2*^*Suc n*)⟩ ⨂
|*state-basis 1* ((*j mod 2*^*Suc* (*Suc n*)) *mod 2*)⟩)) ⨂
((*1*$_m$ (*2*^(*Suc n*))) ∗ |*state-basis* (*Suc n*) ((*j mod 2*^*Suc* (*Suc*
n)) *div 2*)⟩)
using *mult-distr-tensor*
by (*smt* (*verit, del-insts*) *One-nat-def SWAP-ncols SWAP-nrows SWAP-tensor carrier-matD*(*2*)
dim-col-tensor-mat dim-row-mat(*1*) *dim-row-tensor-mat index-mult-mat*(*2*)

index-one-mat(*3*) *index-unit-vec*(*3*) *ket-vec-def lessI one-power2 pos2 power-Suc2*
power-one-right state-basis-carrier-mat state-basis-def zero-less-power)
also have ... = (|*state-basis 1* ((*j mod 2*^*Suc* (*Suc n*)) *mod 2*)⟩ ⨂
|*state-basis 1* ((*j div 2*) *div 2*^*Suc n*)⟩) ⨂
|*state-basis* (*Suc n*) ((*j mod 2*^*Suc* (*Suc n*)) *div 2*)⟩
using *SWAP-tensor*
by (*metis left-mult-one-mat power-one-right state-basis-carrier-mat*)
also have ... = |*state-basis 1* ((*j mod 2*^*Suc* (*Suc n*)) *mod 2*)⟩ ⨂
(|*state-basis 1* ((*j div 2*) *div 2*^*Suc n*)⟩ ⨂
|*state-basis* (*Suc n*) ((*j mod 2*^*Suc* (*Suc n*)) *div 2*)⟩)
using *tensor-mat-is-assoc* **by** *simp*
also have ... = |*state-basis 1* (*j mod 2*)⟩ ⨂
(|*state-basis 1* ((*j div 2*) *div 2*^*Suc n*)⟩ ⨂

```
                    |state-basis (Suc n) ((j div 2) mod 2^Suc n)⟩)
    proof –
      have f1: ∀ n na. (n::nat) ^ (1 + na) = n ^ Suc na
        by simp
      have ∀ n na. (n::nat) dvd n ^ Suc na
        by simp
      then show ?thesis
      using f1 by (smt (z3) div-exp-mod-exp-eq mod-mod-cancel power-one-right)
    qed
    also have ... = |state-basis 1 (j mod 2)⟩ ⨂ |state-basis (Suc (Suc n)) (j
div 2)⟩
      using state-basis-dec jl
      by (metis less-mult-imp-div-less power-Suc2)
    finally show ?thesis by this
   qed
  qed
 qed
qed


lemma SWAP-down-action:
  ∀ j. j < 2 ^Suc (Suc n) ⟶
    SWAP-down (Suc (Suc n)) * ( |state-basis 1 (j mod 2)⟩ ⨂ |state-basis (Suc n)
(j div 2)⟩) =
    |state-basis (Suc n) (j div 2)⟩ ⨂ |state-basis 1 (j mod 2)⟩
proof (induct n)
  case 0
  show ?case
  proof
    fix j::nat
    show j < 2 ^ Suc (Suc 0) ⟶
        SWAP-down (Suc (Suc 0)) * ( |state-basis 1 (j mod 2)⟩ ⨂ |state-basis (Suc
0) (j div 2)⟩) =
         |state-basis (Suc 0) (j div 2)⟩ ⨂ |state-basis 1 (j mod 2)⟩
    proof
      assume j < 2 ^ Suc (Suc 0)
      show SWAP-down (Suc (Suc 0))*( |state-basis 1 (j mod 2)⟩ ⨂ |state-basis
(Suc 0) (j div 2)⟩)
         = |state-basis (Suc 0) (j div 2)⟩ ⨂ |state-basis 1 (j mod 2)⟩
      proof –
        have SWAP-down (Suc (Suc 0))*( |state-basis 1 (j mod 2)⟩⨂|state-basis
(Suc 0) (j div 2)⟩)
          = SWAP * ( |state-basis 1 (j mod 2)⟩ ⨂ |state-basis (Suc 0) (j div 2)⟩)
          using SWAP-down.simps by simp
        also have ... = |state-basis (Suc 0) (j div 2)⟩ ⨂ |state-basis 1 (j mod 2)⟩
          using SWAP-tensor state-basis-carrier-mat
          by (metis One-nat-def power-one-right)
        finally show ?thesis by this
```

qed
qed
qed
next
case (*Suc n*)
assume *HI*:$\forall j<2$ ^ *Suc* (*Suc n*).
SWAP-down (*Suc* (*Suc n*))∗(|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$ |*state-basis* (*Suc n*) (*j div 2*)⟩)
= |*state-basis* (*Suc n*) (*j div 2*)⟩ $\bigotimes$ |*state-basis 1* (*j mod 2*)⟩
show $\forall j<2$ ^ *Suc* (*Suc* (*Suc n*)).
SWAP-down (*Suc* (*Suc* (*Suc n*)))∗(|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$
|*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩)
= |*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩ $\bigotimes$ |*state-basis 1* (*j mod 2*)⟩
proof
fix *j*::*nat*
show *j* < *2* ^ *Suc* (*Suc* (*Suc n*)) ⟶
SWAP-down (*Suc* (*Suc* (*Suc n*))) ∗ (|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$ |*state-basis* (*Suc* (*Suc n*))
(*j div 2*)⟩) =
|*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩ $\bigotimes$ |*state-basis 1* (*j mod 2*)⟩
proof
assume *jl*:*j* < *2* ^ *Suc* (*Suc* (*Suc n*))
show *SWAP-down* (*Suc* (*Suc* (*Suc n*))) ∗ (|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$
|*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩) =
|*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩ $\bigotimes$ |*state-basis 1* (*j mod 2*)⟩
proof −
define *x* **where** *x* = *2*∗((*j div 2*) *div 2*) + (*j mod 2*)
have *xl*:*x* < *2*^*Suc* (*Suc n*)
proof −
have *j mod 2* < *2* **by** *auto*
moreover have *0*:(*j div 2*) *div 2* < *2*^*Suc n* **using** *jl* **by** *auto*
moreover have *2*∗((*j div 2*) *div 2*) < *2*^*Suc* (*Suc n*) **using** *0* **by** *auto*
ultimately show *?thesis* **using** *x-def*
by (*metis* (*no-types*, *lifting*) *Suc-double-not-eq-double add.right-neutral add-Suc-right*
less-2-cases-iff linorder-neqE-nat not-less-eq power-Suc)
qed
have *xm*:*x mod 2* = *j mod 2* **using** *x-def* **by** *auto*
have *xd*:*x div 2* = *j div 2 div 2* **using** *x-def* **by** *auto*
have *SWAP-down* (*Suc* (*Suc* (*Suc n*))) ∗ (|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$
|*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩) =
(((*1*$_m$ (*2*^(*Suc n*))) $\bigotimes$ *SWAP*) ∗ ((*SWAP-down* (*Suc* (*Suc n*))) $\bigotimes$
(*1*$_m$ *2*))) ∗
(|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$ |*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩)
using *SWAP-down.simps* **by** *simp*
also have . . . = ((*1*$_m$ (*2*^(*Suc n*))) $\bigotimes$ *SWAP*) ∗ (((*SWAP-down* (*Suc* (*Suc n*))) $\bigotimes$ (*1*$_m$ *2*)) ∗
(|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$ |*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩))

proof (*rule assoc-mult-mat*)
show 1_m (2 ^ Suc n) ⨂ SWAP ∈ carrier-mat (2^Suc (Suc (Suc n))) (2^Suc (Suc (Suc n)))
by (*simp add*: *SWAP-ncols SWAP-nrows carrier-matI*)
show *SWAP-down* (*Suc* (*Suc n*)) ⨂ 1_m 2
∈ *carrier-mat* (2 ^ *Suc* (*Suc* (*Suc n*))) (2 ^ *Suc* (*Suc* (*Suc n*)))
by (*metis One-nat-def SWAP-down.simps*(2) *SWAP-down-carrier-mat power-Suc2*
power-one-right tensor-carrier-mat)
show |*state-basis* 1 (*j mod* 2)⟩ ⨂ |*state-basis* (*Suc* (*Suc n*)) (*j div* 2)⟩
∈ *carrier-mat* (2 ^ *Suc* (*Suc* (*Suc n*))) 1
by (*metis Suc-1 one-power2 power-Suc power-one-right state-basis-carrier-mat*
tensor-carrier-mat)
qed
also have ... = ((1_m (2^(*Suc n*))) ⨂ *SWAP*) ∗ (((*SWAP-down* (*Suc* (*Suc n*))) ⨂ (1_m 2)) ∗
(|*state-basis* 1 (*j mod* 2)⟩ ⨂
(|*state-basis* (*Suc n*) ((*j div* 2) *div* 2)⟩ ⨂
|*state-basis* 1 ((*j div* 2) *mod* 2)⟩)))
using *state-basis-dec′ jl*
by (*metis less-mult-imp-div-less power-Suc2*)
also have ... = ((1_m (2^(*Suc n*))) ⨂ *SWAP*) ∗ (((*SWAP-down* (*Suc* (*Suc n*))) ⨂ (1_m 2)) ∗
((|*state-basis* 1 (*j mod* 2)⟩ ⨂
|*state-basis* (*Suc n*) ((*j div* 2) *div* 2)⟩) ⨂
|*state-basis* 1 ((*j div* 2) *mod* 2)⟩))
using *tensor-mat-is-assoc* **by** *simp*
also have ... = ((1_m (2^(*Suc n*))) ⨂ *SWAP*) ∗
(((*SWAP-down* (*Suc* (*Suc n*))) ∗ (|*state-basis* 1 (*j mod* 2)⟩ ⨂
|*state-basis* (*Suc n*) ((*j div* 2) *div* 2)⟩)) ⨂
((1_m 2) ∗ |*state-basis* 1 ((*j div* 2) *mod* 2)⟩))
using *mult-distr-tensor*
by (*smt* (*verit*, *ccfv-threshold*) *SWAP-down-carrier-mat carrier-matD*(1) *carrier-matD*(2)
dim-col-tensor-mat dim-row-tensor-mat index-one-mat(3) *mult.right-neutral*
nat-zero-less-power-iff pos2 power-Suc2 power-commutes power-one-right
state-basis-carrier-mat zero-less-one-class.zero-less-one)
also have ... = ((1_m (2^(*Suc n*))) ⨂ *SWAP*) ∗
(((*SWAP-down* (*Suc* (*Suc n*))) ∗ (|*state-basis* 1 (*x mod* 2)⟩ ⨂
|*state-basis* (*Suc n*) (*x div* 2)⟩)) ⨂
((1_m 2) ∗ |*state-basis* 1 ((*j div* 2) *mod* 2)⟩))
using *xm xd* **by** *simp*
also have ... = ((1_m (2^(*Suc n*))) ⨂ *SWAP*) ∗
((|*state-basis* (*Suc n*) (*x div* 2)⟩ ⨂ |*state-basis* 1 (*x mod* 2)⟩) ⨂
|*state-basis* 1 ((*j div* 2) *mod* 2)⟩))

using *HI*
by (*metis dim-row-mat*(*1*) *index-unit-vec*(*3*) *ket-vec-def left-mult-one-mat′ power-one-right*
state-basis-def xl)
also have ... = ((1_m (*2*^(*Suc n*))) ⨂ *SWAP*) ∗
(|*state-basis* (*Suc n*) (*x div 2*)⟩ ⨂ (|*state-basis 1* (*x mod 2*)⟩ ⨂
|*state-basis 1* ((*j div 2*) *mod 2*)⟩))
using *tensor-mat-is-assoc* **by** *force*
also have ... = ((1_m (*2*^(*Suc n*))) ∗ |*state-basis* (*Suc n*) (*x div 2*)⟩) ⨂
(*SWAP* ∗ (|*state-basis 1* (*x mod 2*)⟩ ⨂ |*state-basis 1* ((*j div 2*) *mod 2*)⟩))
using *mult-distr-tensor state-basis-carrier-mat SWAP-carrier-mat*
by (*smt* (*verit*, *del-insts*) *SWAP-tensor carrier-matD*(*1*) *carrier-matD*(*2*) *dim-col-tensor-mat*
index-mult-mat(*2*) *index-one-mat*(*3*) *nat-0-less-mult-iff power-one-right*
tensor-mat-is-assoc zero-less-numeral zero-less-one-class.zero-less-one zero-less-power)
also have ... = |*state-basis* (*Suc n*) (*x div 2*)⟩ ⨂
(|*state-basis 1* ((*j div 2*) *mod 2*)⟩ ⨂ |*state-basis 1* (*x mod 2*)⟩)
using *SWAP-tensor*
by (*metis left-mult-one-mat power-one-right state-basis-carrier-mat*)
also have ... = (|*state-basis* (*Suc n*) (*x div 2*)⟩ ⨂ |*state-basis 1* ((*j div 2*) *mod 2*)⟩) ⨂
|*state-basis 1* (*x mod 2*)⟩
using *assoc-mult-mat tensor-mat-is-assoc* **by** *presburger*
also have ... = |*state-basis* (*Suc* (*Suc n*)) (*j div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩
using *state-basis-dec′ xd xm*
by (*metis jl less-mult-imp-div-less power-Suc2*)
finally show *?thesis* **by** *this*
qed
qed
qed
qed

Action of the controlled-R gates in the circuit

lemma *controlR-action*:
assumes *j* < *2* ^ *Suc* (*Suc n*)
shows (*control* (*Suc* (*Suc n*)) (*R* (*Suc* (*Suc n*)))) ∗
((|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat* (*j div 2*) / *2*^(*Suc n*)) $\cdot_m$ |*one*⟩) ⨂
|*state-basis n* ((*j mod 2*^(*Suc n*)) *div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) =
(|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^(*Suc* (*Suc n*))) $\cdot_m$ |*one*⟩) ⨂
|*state-basis n* ((*j mod 2*^(*Suc n*)) *div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩
proof (*cases n*)
case *0*
then show *?thesis*
proof −

assume *n0*:*n* = *0*
show *control* (*Suc* (*Suc n*)) (*R* (*Suc* (*Suc n*))) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*)
$\cdot_m$ |*Deutsch.one*⟩ ⨂ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) =
|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat j* / *2* ^ *Suc* (*Suc n*)) $\cdot_m$
|*Deutsch.one*⟩ ⨂ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩
proof −
have *control* (*Suc* (*Suc 0*)) (*R* (*Suc* (*Suc 0*))) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc 0*)
$\cdot_m$ |*Deutsch.one*⟩ ⨂ |*state-basis 0* (*j mod 2* ^ *Suc 0 div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) =
control2 (*R 2*) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc 0*)
$\cdot_m$ |*Deutsch.one*⟩ ⨂ |*state-basis 0* (*j mod 2* ^ *Suc 0 div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩)
using *control.simps* **by** (*metis One-nat-def Suc-1*)
also have … = *control2* (*R 2*) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc 0*)
$\cdot_m$ |*Deutsch.one*⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩)
using *state-basis-def unit-vec-def ket-vec-def*
by (*smt* (*verit*, *del-insts*) *H-inv H-is-gate One-nat-def gate-def index-mult-mat*(*2*)

index-one-mat(*2*) *mod-less-divisor mod-mod-trivial pos2 state-basis-dec′*
tensor-mat-is-assoc)
also have … = (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^(*Suc* (*Suc 0*))) $\cdot_m$ |*one*⟩) ⨂
|*state-basis 1* (*j mod 2*)⟩
proof (*rule disjE*)
show *j mod 2* = *0* ∨ *j mod 2* = *1* **by** *auto*
next
assume *jm0*:*j mod 2* = *0*
hence *jdj*:*j div 2* = *j*/*2* **by** *auto*
have *control2* (*R 2*) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc 0*)
$\cdot_m$ |*Deutsch.one*⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) =
control2 (*R 2*) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc 0*)
$\cdot_m$ |*Deutsch.one*⟩ ⨂ |*zero*⟩)
using *jm0 state-basis-def mat-of-cols-list-def* **by** *fastforce*
also have … = |*Deutsch.zero*⟩ + *exp* (*2*∗i∗*pi*∗ *complex-of-nat* (*j div 2*) / *2*

^ *Suc 0*)
·$_m$ |*Deutsch.one*⟩ ⨂ |*zero*⟩
using *control2-zero* **by** (*simp add*: *ket-vec-def*)
also have . . . = |*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗
complex-of-nat j / *2* ^ *Suc* (*Suc 0*)) ·$_m$ |*Deutsch.one*⟩ ⨂
|*state-basis 1* (*j mod 2*)⟩
using *jm0 state-basis-def mat-of-cols-list-def jdj*
by (*smt* (*verit*, *best*) *Euclidean-Rings.div-eq-0-iff One-nat-def Suc-1 assms*
divide-divide-eq-left divide-eq-0-iff less-2-cases-iff less-power-add-imp-div-less
n0
neq-imp-neq-div-or-mod of-nat-0 of-nat-1 of-nat-Suc of-nat-numeral
of-real-1
of-real-divide of-real-numeral power-Suc power-one-right times-divide-eq-right

two-div-two two-mod-two)
finally show *?thesis* **by** *this*
next
assume *jm1*:*j mod 2* = *1*
have *control2* (*R 2*) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div*
2) / *2* ^ *Suc 0*)
·$_m$ |*Deutsch.one*⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) =
control2 (*R 2*) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div*
2) / *2* ^ *Suc 0*)
·$_m$ |*Deutsch.one*⟩ ⨂ |*one*⟩)
using *jm1* **by** (*simp add*: *state-basis-def*)
also have . . . = ((*R 2*) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div*
2) / *2* ^ *Suc 0*)
·$_m$ |*Deutsch.one*⟩)) ⨂ |*one*⟩
using *control2-one ket-vec-def R-def mat-of-cols-list-def* **by** *simp*
also have . . . = (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j*/*2*^*Suc* (*Suc 0*)) ·$_m$
|*one*⟩) ⨂ |*one*⟩
using *R-action jm1 assms* **by** (*metis One-nat-def Suc-1 n0*)
finally show *?thesis* **by** (*metis jm1 power-one-right state-basis-def*)
qed
finally show *?thesis*
by (*smt* (*verit*, *best*) *Euclidean-Rings.div-eq-0-iff Suc-1 mod-less-divisor n0*
not-mod2-eq-Suc-0-eq-0 one-mod-two-eq-one pos2 power-0 power-one-right
state-basis-dec′
tensor-mat-is-assoc)
qed
qed
next
case (*Suc nat*)
then show *?thesis*
proof −
assume *n* = *Suc nat*

define *jd2* **where** *jd2 = j div 2*
define *jm2* **where** *jm2 = j mod 2*
define *jm2sn* **where** *jm2sn = j mod 2^Suc n*
have *jeq:jm2sn mod 2 = j mod 2* **using** *jm2sn-def*
by (*metis One-nat-def Suc-le-mono mod-mod-power-cancel power-one-right zero-order*(*1*))
have (*control* (*Suc* (*Suc n*)) (*R* (*Suc* (*Suc n*)))) ∗ (|*Deutsch.zero*⟩ +
exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*Deutsch.one*⟩ ⨂
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) =
(((*1*$_m$ *2*) ⨂ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ⨂ (*1*$_m$ (*2^n*))) ∗
((*1*$_m$ *2*) ⨂ *SWAP-up* (*Suc n*))) ∗ (|*Deutsch.zero*⟩ +
exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*Deutsch.one*⟩ ⨂
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩)
using *control.simps Suc* **by** *presburger*
also have ... = (((*1*$_m$ *2*) ⨂ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ⨂ (*1*$_m$ (*2^n*)))) ∗
(((*1*$_m$ *2*) ⨂ *SWAP-up* (*Suc n*)) ∗ (|*Deutsch.zero*⟩ +
exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*Deutsch.one*⟩ ⨂
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩))
proof (*rule assoc-mult-mat*)
show (*1*$_m$ *2* ⨂ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ⨂ *1*$_m$ (*2* ^ *n*))
∈ *carrier-mat* (*2^Suc* (*Suc n*)) (*2^Suc* (*Suc n*))
using *SWAP-down-carrier-mat SWAP-up-carrier-mat control2-carrier-mat*
by (*smt* (*verit*) *Suc carrier-matD*(*1*) *carrier-matD*(*2*) *carrier-matI control.simps*(*4*)
control-carrier-mat dim-col-tensor-mat index-mult-mat(*2*) *index-mult-mat*(*3*)

index-one-mat(*3*) *mult-numeral-left-semiring-numeral num-double power-Suc*)
show *1*$_m$ *2* ⨂ *SWAP-up* (*Suc n*) ∈ *carrier-mat* (*2* ^ *Suc* (*Suc n*)) (*2* ^ *Suc* (*Suc n*))
using *SWAP-up-carrier-mat*
by (*metis One-nat-def SWAP-up.simps*(*2*) *power-Suc power-one-right tensor-carrier-mat*)
show |*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) /
2 ^ *Suc n*) $\cdot_m$ |*Deutsch.one*⟩ ⨂ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂
|*state-basis 1* (*j mod 2*)⟩ ∈ *carrier-mat* (*2* ^ *Suc* (*Suc n*)) *1*
using *ket-vec-def state-basis-carrier-mat*
by (*simp add*: *carrier-matI index-unit-vec*(*3*) *state-basis-def*)
qed
also have ... = (((*1*$_m$ *2*) ⨂ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ⨂ (*1*$_m$ (*2^n*)))) ∗
(((*1*$_m$ *2*) ⨂ *SWAP-up* (*Suc n*)) ∗ (|*Deutsch.zero*⟩ +

exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*Deutsch.one*⟩ ⊗
(|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⊗ |*state-basis 1* (*j mod 2*)⟩)))
using *tensor-mat-is-assoc* **by** *presburger*
also have ... = (((*1*$_m$ *2*) ⊗ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ⊗ (*1*$_m$ (*2*^*n*)))) ∗
(((*1*$_m$ *2*) ∗ (|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$
|*Deutsch.one*⟩)) ⊗ ((*SWAP-up* (*Suc n*)) ∗ (|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⊗
|*state-basis 1* (*j mod 2*)⟩))))
using *mult-distr-tensor*
by (*smt* (*verit*, *del-insts*) *SWAP-up-carrier-mat carrier-matD*(*2*) *dim-col-mat*(*1*)

dim-col-tensor-mat dim-row-mat(*1*) *dim-row-tensor-mat index-add-mat*(*2*) *index-add-mat*(*3*)
index-one-mat(*3*) *index-smult-mat*(*2*) *index-smult-mat*(*3*) *index-unit-vec*(*3*) *ket-vec-def*
one-power2 pos2 power-Suc2 power-one-right state-basis-def
zero-less-one-class.zero-less-one zero-less-power)
also have ... = (((*1*$_m$ *2*) ⊗ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ⊗ (*1*$_m$ (*2*^*n*)))) ∗
((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$
|*one*⟩) ⊗ (|*state-basis 1* (*j mod 2*)⟩ ⊗ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩)))
using *SWAP-up-action jeq*
by (*smt* (*verit*, *best*) *Suc index-add-mat*(*2*) *index-smult-mat*(*2*) *jm2sn-def ket-one-is-state*
left-mult-one-mat' *mod-less-divisor pos2 power-one-right state.dim-row zero-less-power*)
also have ... = (((*1*$_m$ *2*) ⊗ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ⊗ (*1*$_m$ (*2*^*n*)))) ∗
(((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$
|*one*⟩) ⊗ |*state-basis 1* (*j mod 2*)⟩) ⊗ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩))
using *tensor-mat-is-assoc* **by** *presburger*
also have ... = ((*1*$_m$ *2*) ⊗ *SWAP-down* (*Suc n*)) ∗ (((*control2* (*R* (*Suc* (*Suc n*))) ⊗ (*1*$_m$ (*2*^*n*)))) ∗
(((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$
|*one*⟩) ⊗ |*state-basis 1* (*j mod 2*)⟩) ⊗ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩)))
proof (*rule assoc-mult-mat*)
show *1*$_m$ *2* ⊗ *SWAP-down* (*Suc n*) ∈ *carrier-mat* (*2*^*Suc* (*Suc n*)) (*2*^*Suc* (*Suc n*))
using *SWAP-down-carrier-mat*
by (*metis One-nat-def SWAP-down.simps*(*2*) *power-Suc power-one-right*

tensor-carrier-mat)
show *control2* (*R* (*Suc* (*Suc* *n*))) $\bigotimes$ 1_m (*2* ^ *n*) ∈ *carrier-mat* (*2*^*Suc* (*Suc* *n*)) (*2*^*Suc* (*Suc* *n*))
using *control2-carrier-mat* **by** *simp*
show |*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*)
$\cdot_m$ |*Deutsch.one*⟩ $\bigotimes$ |*state-basis 1* (*j mod 2*)⟩ $\bigotimes$ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩
∈ *carrier-mat* (*2* ^ *Suc* (*Suc n*)) *1*
using *state-basis-carrier-mat ket-vec-def*
by (*simp add*: *carrier-matI state-basis-def*)
qed
also have ... = ((*1*$_m$ *2*) $\bigotimes$ *SWAP-down* (*Suc n*)) ∗ (((*control2* (*R* (*Suc* (*Suc n*)))) ∗
((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*one*⟩)
$\bigotimes$ |*state-basis 1* (*j mod 2*)⟩)) $\bigotimes$ ((*1*$_m$ (*2*^*n*)) ∗ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩))
using *mult-distr-tensor*
by (*smt* (*verit*, *del-insts*) *SWAP-nrows SWAP-tensor carrier-matD*(*1*) *carrier-matD*(*2*)
carrier-matI control2-carrier-mat dim-col-tensor-mat index-add-mat(*2*) *index-add-mat*(*3*)
index-mult-mat(*2*) *index-one-mat*(*3*) *index-smult-mat*(*2*) *index-smult-mat*(*3*) *ket-one-is-state*
less-numeral-extra(*1*) *one-power2 power-Suc2 power-one-right state-basis-carrier-mat*
state-def zero-less-numeral zero-less-power)
also have ... = ((*1*$_m$ *2*) $\bigotimes$ *SWAP-down* (*Suc n*)) ∗
((|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat j* / *2* ^ *Suc* (*Suc n*)) $\cdot_m$ |*one*⟩) $\bigotimes$
|*state-basis 1* (*j mod 2*)⟩ $\bigotimes$ ((*1*$_m$ (*2*^*n*)) ∗ |*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩))
proof (*rule disjE*)
show *j mod 2* = *0* ∨ *j mod 2* = *1* **by** *auto*
next
assume *jm0*:*j mod 2* = *0*
hence *jid*:*j* / *2* = *j div 2* **by** *auto*
have (*control2* (*R* (*Suc* (*Suc n*)))) ∗
((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*one*⟩)
$\bigotimes$ |*state-basis 1* (*j mod 2*)⟩) =
(*control2* (*R* (*Suc* (*Suc n*)))) ∗
((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*one*⟩)
$\bigotimes$ |*zero*⟩)
using *state-basis-def jm0* **by** *fastforce*
also have ... = ((|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$ |*one*⟩)
$\bigotimes$ |*zero*⟩)

using *control2-zero* **by** (*simp add*: *ket-vec-def*)
also have ... = (|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat j* / *2* ^ *Suc* (*Suc* *n*)) ·$_m$ |*one*⟩) ⨂
|*zero*⟩
using *jid*
by (*smt* (*verit*, *del-insts*) *dbl-simps*(*3*) *dbl-simps*(*5*) *divide-divide-eq-left* *numerals*(*1*)
of-nat-1 of-nat-numeral of-real-divide of-real-of-nat-eq power-Suc
times-divide-eq-right)
finally show (*1*$_m$ *2* ⨂ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ∗ (|*Deutsch.zero*⟩ +
exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc* *n*) ·$_m$
|*Deutsch.one*⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) ⨂ *1*$_m$ (*2* ^ *n*) ∗
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩) = (*1*$_m$ *2* ⨂ *SWAP-down* (*Suc n*)) ∗
(|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat j* /
2 ^ *Suc* (*Suc n*)) ·$_m$ |*Deutsch.one*⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩ ⨂ *1*$_m$ (*2* ^ *n*) ∗
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩)
by (*metis jm0 power-one-right state-basis-def*)
next
assume *jm1*:*j mod 2* = *1*
have (*control2* (*R* (*Suc* (*Suc n*)))) ∗
((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc* *n*) ·$_m$ |*one*⟩)
⨂ |*state-basis 1* (*j mod 2*)⟩) =
(*control2* (*R* (*Suc* (*Suc n*)))) ∗
((|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc* *n*) ·$_m$ |*one*⟩)
⨂ |*one*⟩)
using *jm1 state-basis-def* **by** *fastforce*
also have ... = ((*R* (*Suc* (*Suc n*))) ∗
(|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) ·$_m$ |*one*⟩))
⨂ |*one*⟩
using *control2-one* **by** (*simp add*: *ket-vec-def R-def mat-of-cols-list-def*)
also have ... = (|*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat j* / *2*^(*Suc* (*Suc n*))) ·$_m$ |*one*⟩) ⨂ |*one*⟩
using *R-action*
by (*metis assms jm1*)
finally show (*1*$_m$ *2* ⨂ *SWAP-down* (*Suc n*)) ∗ (*control2* (*R* (*Suc* (*Suc n*))) ∗ (|*Deutsch.zero*⟩ +
exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc* *n*) ·$_m$
|*Deutsch.one*⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩) ⨂ *1*$_m$ (*2* ^ *n*) ∗
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩) =
(*1*$_m$ *2* ⨂ *SWAP-down* (*Suc n*)) ∗ (|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗

complex-of-real pi ∗ *complex-of-nat j* / *2* ^ *Suc* (*Suc n*)) $\cdot_m$
|*Deutsch.one*⟩ ⨂
|*state-basis 1* (*j mod 2*)⟩ ⨂ 1_m (*2* ^ *n*) ∗ |*state-basis n* (*j mod 2* ^
Suc n div 2)⟩)
by (*metis jm1 power-one-right state-basis-def*)
qed
also have ... = ((*1*$_m$ *2*) ⨂ *SWAP-down* (*Suc n*)) ∗
((|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat j* / *2* ^ *Suc* (*Suc n*))
$\cdot_m$ |*one*⟩) ⨂
(|*state-basis 1* (*j mod 2*)⟩ ⨂ ((*1*$_m$ (*2*^*n*)) ∗
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩)))
using *tensor-mat-is-assoc ket-vec-def* **by** *auto*
also have ... = (|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat j* / *2* ^ *Suc* (*Suc*
n)) $\cdot_m$ |*one*⟩) ⨂
((*SWAP-down* (*Suc n*)) ∗ (|*state-basis 1* (*j mod 2*)⟩ ⨂ ((*1*$_m$
(*2*^*n*)) ∗
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩)))
using *mult-distr-tensor*
by (*smt* (*verit*, *del-insts*) *SWAP-down-carrier-mat carrier-matD*(*1*) *car-
rier-matD*(*2*)
dim-col-tensor-mat dim-row-tensor-mat index-add-mat(*2*) *index-add-mat*(*3*)
index-one-mat(*3*)
index-smult-mat(*2*) *index-smult-mat*(*3*) *ket-one-is-state left-mult-one-mat′*
one-power2 pos2
power.simps(*2*) *power-one-right state-basis-carrier-mat state-def*
zero-less-one-class.zero-less-one zero-less-power)
also have ... = (|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat j* / *2* ^ *Suc* (*Suc*
n)) $\cdot_m$ |*one*⟩) ⨂
(|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂ |*state-basis 1* (*j mod*
2)⟩)
using *SWAP-down-action jeq*
by (*metis Suc dim-row-mat*(*1*) *index-unit-vec*(*3*) *jm2sn-def ket-vec-def left-mult-one-mat′*

mod-less-divisor pos2 state-basis-def zero-less-power)
finally show *control* (*Suc* (*Suc n*)) (*R* (*Suc* (*Suc n*))) ∗ (|*Deutsch.zero*⟩ + *exp*
(*2* ∗ i ∗
complex-of-real pi ∗ *complex-of-nat* (*j div 2*) / *2* ^ *Suc n*) $\cdot_m$
|*Deutsch.one*⟩ ⨂
|*state-basis n* (*j mod 2* ^ *Suc n div 2*)⟩ ⨂ |*state-basis 1* (*j mod 2*)⟩)
=
|*Deutsch.zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat j* /
2 ^ *Suc* (*Suc n*)) $\cdot_m$ |*Deutsch.one*⟩ ⨂ |*state-basis n* (*j mod 2* ^ *Suc*
n div 2)⟩ ⨂
|*state-basis 1* (*j mod 2*)⟩
using *tensor-mat-is-assoc ket-vec-def* **by** *auto*
qed
qed

Action of the controlled rotations subcircuit

lemma *controlled-rotations-ind*:
  $\forall j.\ j < 2 \,\hat{}\, Suc\ n \longrightarrow$
  controlled-rotations (*Suc n*) $*$
  (($|zero\rangle$ + *exp*($2*$i$*pi*$(*complex-of-nat* (*j div* $2\hat{}n$))/2) $\cdot_m$ $|one\rangle$) $\bigotimes$ $|state\text{-}basis$
n (*j mod* $2\hat{}n$)$\rangle$) =
  ($|zero\rangle$ + *exp*($2*$i$*pi*j/(2\hat{}(Suc\ n))$) $\cdot_m$ $|one\rangle$) $\bigotimes$ $|state\text{-}basis\ n\ (j\ mod\ 2\hat{}n)\rangle$
proof (*induct n*)
  case *0*
  then show *?case*
  proof (*rule allI*)
    fix *j*::*nat*
    show $j < 2 \,\hat{}\, Suc\ 0 \longrightarrow$
          controlled-rotations (*Suc 0*) $*$ ($|zero\rangle$ +
          exp (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat* (*j div* $2 \,\hat{}\, 0$) / 2) $\cdot_m$ $|one\rangle$
$\bigotimes$
          $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$)) =
        $|zero\rangle$ + *exp* (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat j* / $2 \,\hat{}\, Suc\ 0$) $\cdot_m$
$|one\rangle$ $\bigotimes$
        $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$
    proof
      assume $j < 2 \,\hat{}\, Suc\ 0$
      hence *j2*:$j < 2$ **by** *auto*
      have *controlled-rotations* (*Suc 0*) $*$ ($|zero\rangle$ +
            exp (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat* (*j div* $2 \,\hat{}\, 0$) / 2) $\cdot_m$
$|one\rangle$ $\bigotimes$
           $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$)) =
           (1_m 2) $*$ ($|zero\rangle$ +
            exp (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat* (*j div* $2 \,\hat{}\, 0$) / 2) $\cdot_m$
$|one\rangle$ $\bigotimes$
           $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$))
        using *controlled-rotations.simps* **by** *simp*
      also have ... = $|zero\rangle$ +
                 exp (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat* (*j div* $2 \,\hat{}\, 0$) /
2) $\cdot_m$ $|one\rangle$ $\bigotimes$
                 $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$
        using *left-mult-one-mat* **by** (*simp add*: *ket-vec-def state-basis-def*)
      also have ... = $|zero\rangle$ +
                 exp (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat j* / $2\hat{}Suc\ 0$) $\cdot_m$
$|one\rangle$ $\bigotimes$
                 $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$
        by *auto*
      finally show *controlled-rotations* (*Suc 0*) $*$ ($|zero\rangle$ +
               exp (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat* (*j div* $2 \,\hat{}\, 0$) / 2)
$\cdot_m$ $|one\rangle$ $\bigotimes$
               $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$)) =
                $|zero\rangle$ + *exp* (2 $*$ i $*$ *complex-of-real pi* $*$ *complex-of-nat j* / $2 \,\hat{}$
Suc 0) $\cdot_m$ $|one\rangle$
               $\bigotimes$ $|state\text{-}basis\ 0\ (j\ mod\ 2 \,\hat{}\, 0)\rangle$
        by *this*

qed
qed
next
case (*Suc n'*)
define *n* **where** $n = Suc\ n'$
assume *HI*: $\forall j<2\ \hat{}\ Suc\ n'.\ controlled\text{-}rotations\ (Suc\ n') * (\ |zero\rangle +$
$exp\ (2 * i * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ (j\ div\ 2\ \hat{}\ n')\ /\ 2) \cdot_m$
$|one\rangle \bigotimes$
$|state\text{-}basis\ n'\ (j\ mod\ 2\ \hat{}\ n')\rangle) =$
$|Deutsch.zero\rangle + exp\ (2 * i * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ j\ /\ 2\ \hat{}$
$Suc\ n') \cdot_m$
$|Deutsch.one\rangle \bigotimes |state\text{-}basis\ n'\ (j\ mod\ 2\ \hat{}\ n')\rangle$
show $\forall j<2\ \hat{}\ Suc\ (Suc\ n').$
$controlled\text{-}rotations\ (Suc\ (Suc\ n')) *$
$(\ |Deutsch.zero\rangle + exp\ (2 * i * complex\text{-}of\text{-}real\ pi *$
$complex\text{-}of\text{-}nat\ (j\ div\ 2\ \hat{}\ Suc\ n')\ /\ 2) \cdot_m |Deutsch.one\rangle \bigotimes$
$|state\text{-}basis\ (Suc\ n')\ (j\ mod\ 2\ \hat{}\ Suc\ n')\rangle) =$
$|Deutsch.zero\rangle + exp\ (2 * i * complex\text{-}of\text{-}real\ pi *$
$complex\text{-}of\text{-}nat\ j\ /\ 2\ \hat{}\ Suc\ (Suc\ n')) \cdot_m |Deutsch.one\rangle \bigotimes$
$|state\text{-}basis\ (Suc\ n')\ (j\ mod\ 2\ \hat{}\ Suc\ n')\rangle$
proof (*rule allI*)
fix *j*::*nat*
show $j < 2\ \hat{}\ Suc\ (Suc\ n') \longrightarrow$
$controlled\text{-}rotations\ (Suc\ (Suc\ n')) * (\ |Deutsch.zero\rangle +$
$exp\ (2 * i * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ (j\ div\ 2\ \hat{}\ Suc\ n')\ /\ 2) \cdot_m$
$|Deutsch.one\rangle \bigotimes |state\text{-}basis\ (Suc\ n')\ (j\ mod\ 2\ \hat{}\ Suc\ n')\rangle) =$
$|Deutsch.zero\rangle + exp\ (2 * i * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ j\ /\ 2\ \hat{}$
$Suc\ (Suc\ n')) \cdot_m$
$|Deutsch.one\rangle \bigotimes |state\text{-}basis\ (Suc\ n')\ (j\ mod\ 2\ \hat{}\ Suc\ n')\rangle$
proof
assume *jass*:$j < 2\ \hat{}\ Suc\ (Suc\ n')$
show $controlled\text{-}rotations\ (Suc\ (Suc\ n')) * (\ |Deutsch.zero\rangle +$
$exp\ (2 * i * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ (j\ div\ 2\ \hat{}\ Suc\ n')\ /\ 2) \cdot_m$
$|Deutsch.one\rangle \bigotimes |state\text{-}basis\ (Suc\ n')\ (j\ mod\ 2\ \hat{}\ Suc\ n')\rangle) =$
$|Deutsch.zero\rangle + exp\ (2 * i * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ j\ /\ 2\ \hat{}$
$Suc\ (Suc\ n')) \cdot_m$
$|Deutsch.one\rangle \bigotimes |state\text{-}basis\ (Suc\ n')\ (j\ mod\ 2\ \hat{}\ Suc\ n')\rangle$
proof −
define *jd2n jm2n* **where** $jd2n = j\ div\ 2\hat{}n$ **and** $jm2n = j\ mod\ 2\hat{}n$
define *jlast* **where** $jlast = jm2n\ mod\ 2$
define *jmm* **where** $jmm = jm2n\ div\ 2$
define *jd2* **where** $jd2 = j\ div\ 2$
have *jlastj*:$jlast = j\ mod\ 2$ **using** *jlast-def jm2n-def*
by (*metis less-Suc-eq-0-disj less-Suc-eq-le mod-mod-power-cancel n-def*
power-Suc0-right)
have $controlled\text{-}rotations\ (Suc\ n) * (\ |Deutsch.zero\rangle +$
$exp\ (2 * i * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ jd2n\ /\ 2) \cdot_m$
$|Deutsch.one\rangle \bigotimes |state\text{-}basis\ n\ jm2n\rangle) =$
$((control\ (Suc\ n)\ (R\ (Suc\ n))) * ((controlled\text{-}rotations\ n) \bigotimes (1_m\ 2))) *$

(|*zero*⟩ +
exp (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat jd2n* / *2*) $\cdot_m$
|*Deutsch.one*⟩ ⨂ |*state-basis n jm2n*⟩)
using *controlled-rotations.simps n-def* **by** *simp*
also have . . . = ((*control* (*Suc n*) (*R* (*Suc n*))) ∗ ((*controlled-rotations n*)
⨂ (*1*$_m$ *2*))) ∗
(|*zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat jd2n* / *2*) $\cdot_m$
|*one*⟩ ⨂
(|*state-basis n′ jmm*⟩ ⨂ |*state-basis 1 jlast*⟩))
using *state-basis-dec′ jass n-def jlast-def jmm-def jm2n-def mod-less-divisor pos2*
by *presburger*
also have . . . = (*control* (*Suc n*) (*R* (*Suc n*))) ∗ ((((*controlled-rotations n*)
⨂ (*1*$_m$ *2*))) ∗
(|*zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat jd2n* / *2*) $\cdot_m$
|*one*⟩ ⨂
(|*state-basis n′ jmm*⟩ ⨂ |*state-basis 1 jlast*⟩)))
proof (*rule assoc-mult-mat*)
show *control* (*Suc n*) (*R* (*Suc n*)) ∈ *carrier-mat* (*2*^(*Suc n*)) (*2*^(*Suc n*))
using *control-carrier-mat n-def* **by** *blast*
show *controlled-rotations n* ⨂ *1*$_m$ *2* ∈ *carrier-mat* (*2* ^ *Suc n*) (*2* ^ *Suc n*)
using *controlled-rotations-carrier-mat n-def*
by (*metis One-nat-def controlled-rotations.simps*(*2*) *power-Suc2 power-one-right*
tensor-carrier-mat)
show |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*complex-of-nat jd2n* /*2*) $\cdot_m$ |*one*⟩ ⨂ (|*state-basis n′ jmm*⟩ ⨂
|*state-basis 1 jlast*⟩) ∈ *carrier-mat* (*2* ^ *Suc n*) *1*
using *state-basis-carrier-mat ket-vec-def*
by (*simp add*: *carrier-matI n-def state-basis-def*)
qed
also have . . . = (*control* (*Suc n*) (*R* (*Suc n*))) ∗ ((((*controlled-rotations n*)
⨂ (*1*$_m$ *2*))) ∗
((|*zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat jd2n* / *2*) $\cdot_m$
|*one*⟩ ⨂
|*state-basis n′ jmm*⟩) ⨂ |*state-basis 1 jlast*⟩))
using *tensor-mat-is-assoc control-carrier-mat n-def controlled-rotations-carrier-mat*
state-basis-carrier-mat ket-vec-def **by** *simp*
also have . . . = (*control* (*Suc n*) (*R* (*Suc n*))) ∗ (((*controlled-rotations n*) ∗
((|*zero*⟩ + *exp* (*2* ∗ i ∗ *pi* ∗ *complex-of-nat jd2n* / *2*) $\cdot_m$ |*one*⟩)
⨂
|*state-basis n′ jmm*⟩)) ⨂ ((*1*$_m$ *2*) ∗ |*state-basis 1 jlast*⟩))
using *mult-distr-tensor control-carrier-mat n-def controlled-rotations-carrier-mat*
state-basis-carrier-mat ket-vec-def
by (*smt* (*verit*) *carrier-matD*(*1*) *carrier-matD*(*2*) *dim-col-tensor-mat*
dim-row-tensor-mat
index-add-mat(*2*) *index-add-mat*(*3*) *index-one-mat*(*3*) *index-smult-mat*(*2*)

index-smult-mat(3) ket-one-is-state one-power2 pos2 power-Suc power-one-right
state-def zero-less-one-class.zero-less-one zero-less-power)
also have ... = (*control* (*Suc n*) (*R* (*Suc n*))) $*$
((|*zero*⟩ + *exp* (2$*$i$*$*pi*$*$*complex-of-nat jd2* / 2^*n*) $\cdot_m$
|*one*⟩ $\bigotimes$ |*state-basis n′* (*jd2 mod* 2 ^ *n′*)⟩) $\bigotimes$
((1$_m$ 2) $*$ |*state-basis 1 jlast*⟩))
using *HI jd2-def n-def*
by (*smt* (*verit, del-insts*) *Suc-eq-plus1 div-exp-eq div-exp-mod-exp-eq jass jd2n-def*
jm2n-def jmm-def less-power-add-imp-div-less plus-1-eq-Suc power-one-right)
also have ... = (*control* (*Suc n*) (*R* (*Suc n*))) $*$
((|*zero*⟩ + *exp* (2$*$i$*$*pi*$*$*complex-of-nat jd2* / 2^*n*) $\cdot_m$
|*one*⟩ $\bigotimes$ |*state-basis n′ jmm*⟩) $\bigotimes$
|*state-basis 1 jlast*⟩)
using *jmm-def jd2-def*
by (*metis div-exp-mod-exp-eq jm2n-def left-mult-one-mat n-def plus-1-eq-Suc*
power-one-right state-basis-carrier-mat)
also have ... = (|*zero*⟩ + *exp* (2$*$i$*$*pi*$*$*complex-of-nat j* / 2^*Suc n*) $\cdot_m$ |*one*⟩) $\bigotimes$
|*state-basis n′ jmm*⟩ $\bigotimes$ |*state-basis 1 jlast*⟩
using *controlR-action jmm-def jlast-def jd2-def n-def jm2n-def jass jlastj*
by *presburger*
also have ... = (|*zero*⟩ + *exp* (2$*$i$*$*pi*$*$*complex-of-nat j* / 2^*Suc n*) $\cdot_m$ |*one*⟩) $\bigotimes$
|*state-basis n jm2n*⟩
using *state-basis-dec′ jm2n-def jmm-def jlast-def*
by (*metis mod-less-divisor n-def pos2 tensor-mat-is-assoc zero-less-power*)
finally show *?thesis* **using** *jm2n-def n-def jd2n-def* **by** *meson*
qed
qed
qed
qed

lemma *controlled-rotations-on-first-qubit*:
assumes *j* < 2 ^ *Suc n*
shows *controlled-rotations* (*Suc n*) $*$
(1/*sqrt* 2 $\cdot_m$ (|*zero*⟩ + *exp*(2$*$i$*$*pi*$*$(*complex-of-nat* (*j div* 2^*n*))/2) $\cdot_m$ |*one*⟩) $\bigotimes$
|*state-basis n* (*j mod* 2^*n*)⟩) =
(1/*sqrt* 2 $\cdot_m$ ((|*zero*⟩ + *exp*(2$*$i$*$*pi*$*$*j*/(2^(*Suc n*))) $\cdot_m$ |*one*⟩)) $\bigotimes$ |*state-basis n* (*j mod* 2^*n*)⟩)
proof −
have *controlled-rotations* (*Suc n*) $*$
(1/*sqrt* 2 $\cdot_m$ (|*zero*⟩ + *exp*(2$*$i$*$*pi*$*$(*complex-of-nat* (*j div* 2^*n*))/2) $\cdot_m$ |*one*⟩) $\bigotimes$
|*state-basis n* (*j mod* 2^*n*)⟩) =
controlled-rotations (*Suc n*) $*$

$(1/sqrt\ 2\ \cdot_m\ ((\ |zero\rangle\ +\ exp(2*\mathrm{i}*pi*(complex\text{-}of\text{-}nat\ (j\ div\ 2\hat{\ }n))/2)\ \cdot_m$
$|one\rangle)\ \bigotimes$
$|state\text{-}basis\ n\ (j\ mod\ 2\hat{\ }n)\rangle))$
using *smult-mat-def tensor-mat-def*
by (*smt* (*verit*) *One-nat-def carrier-matD*(*2*) *index-add-mat*(*3*) *index-smult-mat*(*3*) *lessI power-one-right smult-tensor1 state-basis-carrier-mat state-basis-def*)
also have $\ldots = 1/sqrt\ 2\ \cdot_m\ (controlled\text{-}rotations\ (Suc\ n)\ *$
$((\ |zero\rangle\ +\ exp(2*\mathrm{i}*pi*(complex\text{-}of\text{-}nat\ (j\ div\ 2\hat{\ }n))/2)\ \cdot_m\ |one\rangle)\ \bigotimes$
$|state\text{-}basis\ n\ (j\ mod\ 2\hat{\ }n)\rangle))$
using *mult-smult-distrib controlled-rotations-carrier-mat state-basis-carrier-mat*
by (*smt* (*verit*) *carrier-matI dim-row-mat*(*1*) *dim-row-tensor-mat index-add-mat*(*2*)

index-smult-mat(*2*) *index-unit-vec*(*3*) *ket-vec-def power-Suc state-basis-def*)
also have $\ldots = (1/sqrt\ 2\ \cdot_m$
$((\ |zero\rangle\ +\ exp(2*\mathrm{i}*pi*j/(2\hat{\ }(Suc\ n)))\ \cdot_m\ |one\rangle))\ \bigotimes\ |state\text{-}basis\ n$
$(j\ mod\ 2\hat{\ }n)\rangle)$
using *assms controlled-rotations-ind ket-vec-def* **by** *simp*
finally show *?thesis* **by** *this*
qed

More useful lemmas:

lemma *exp-j*:
assumes $l < Suc\ n$
shows $exp\ (2*\mathrm{i}*pi*j/(2\hat{\ }l)) = exp\ (2*\mathrm{i}*pi*(j\ mod\ 2\hat{\ }n)/(2\hat{\ }l))$
proof −
define *jd jm* **where** $jd = j\ div\ 2\hat{\ }n$ **and** $jm = j\ mod\ 2\hat{\ }n$
have $0{:}real\ (2\hat{\ }n)/(2\hat{\ }l) = (2\hat{\ }(n-l))$
proof −
have $1{:}(2{::}nat) \neq 0$ **by** *simp*
have $2{:}l \leq n$ **using** *assms* **by** *simp*
show *?thesis*
using *1 2 power-diff*
by (*metis numeral-power-eq-of-nat-cancel-iff zero-neq-numeral*)
qed
have $j = jd*(2\hat{\ }n) + jm$ **using** *jd-def jm-def* **by** *presburger*
hence $exp\ (2*\mathrm{i}*pi*j/(2\hat{\ }l)) = exp\ (2*pi*\mathrm{i}*(jd*(2\hat{\ }n) + jm)/(2\hat{\ }l))$
by (*simp add*: *mult.commute mult.left-commute*)
also have $\ldots = exp\ (2*pi*\mathrm{i}*(jd*(2\hat{\ }n))/(2\hat{\ }l) + 2*\mathrm{i}*pi*jm/(2\hat{\ }l))$
by (*simp add*: *add-divide-distrib distrib-left mult.left-commute semigroup-mult-class.mult.assoc*)
also have $\ldots = exp\ (2*pi*\mathrm{i}*(jd*(2\hat{\ }n))/(2\hat{\ }l)) * exp\ (2*\mathrm{i}*pi*jm/(2\hat{\ }l))$ **using** *exp-add* **by** *blast*
also have $\ldots = exp\ (2*pi*\mathrm{i}*jd*((2\hat{\ }n)/(2\hat{\ }l))) * exp\ (2*\mathrm{i}*pi*jm/(2\hat{\ }l))$
by (*simp add*: *semigroup-mult-class.mult.assoc*)
also have $\ldots = exp\ (2*pi*\mathrm{i}*jd*((2\hat{\ }(n-l)))) * exp\ (2*\mathrm{i}*pi*jm/(2\hat{\ }l))$
using *0* **by** (*smt* (*verit*) *dbl-simps*(*3*) *dbl-simps*(*5*) *numerals*(*1*) *of-nat-1 of-nat-numeral*

of-nat-power of-real-divide of-real-of-nat-eq)
also have $\ldots = exp\ ((2*pi*\mathrm{i}*jd)*(of\text{-}nat\ (2\hat{\ }(n-l)))) * exp\ (2*\mathrm{i}*pi*jm/(2\hat{\ }l))$
by *auto*

also have ... = $(exp\ (2{*}pi{*}\mathrm{i}))\hat{}(2\hat{}(n{-}l)) * exp\ (2{*}\mathrm{i}{*}pi{*}jm/(2\hat{}l))$
using *exp-of-nat2-mult* **by** (*smt* (*verit*, *best*) *cis-2pi cis-conv-exp exp-power-int exp-zero*
mult.commute mult-zero-right)
also have ... = $1\hat{}(2\hat{}(n{-}l)) * exp\ (2{*}\mathrm{i}{*}pi{*}jm/(2\hat{}l))$ **using** *exp-two-pi-i* **by** *auto*
also have ... = $exp\ (2{*}\mathrm{i}{*}pi{*}jm/(2\hat{}l))$ **by** *auto*
finally show *?thesis* **using** *jd-def jm-def* **by** *simp*
qed

lemma *kron-list-fun*[*simp*]:
$\forall x.$ *List.member xs x* $\longrightarrow$ *f x = g x* $\Longrightarrow$ *kron f xs = kron g xs*
proof (*induct xs*)
case *Nil*
show *kron f* [] = *kron g* [] **by** *simp*
next
fix *a xs*
assume *HI*:($\forall x.$ *List.member xs x* $\longrightarrow$ *f x = g x* $\Longrightarrow$ *kron f xs = kron g xs*)
show $\forall x.$ *List.member* (*a* # *xs*) *x* $\longrightarrow$ *f x = g x* $\Longrightarrow$ *kron f* (*a* # *xs*) = *kron g* (*a* # *xs*)
proof –
assume *1*:$\forall x.$ *List.member* (*a* # *xs*) *x* $\longrightarrow$ *f x = g x*
show *kron f* (*a* # *xs*) = *kron g* (*a* # *xs*)
proof –
from *1* **have** *List.member* (*a* # *xs*) *a* $\longrightarrow$ *f a = g a* **by** *auto*
moreover have *List.member* (*a* # *xs*) *a* **by** (*simp add*: *List.member-rec*(*1*))
ultimately have *2*:*f a = g a* **by** *auto*
have *kron f* (*a*#*xs*) = *f a* $\bigotimes$ *kron f xs* **by** *simp*
also have ... = *g a* $\bigotimes$ *kron f xs* **using** *2* **by** *simp*
also have ... = *g a* $\bigotimes$ *kron g xs* **using** *HI 1* **by** (*simp add*: *member-rec*(*1*))
also have ... = *kron g* (*a*#*xs*) **using** *kron.simps*(*2*) **by** *simp*
finally show *?thesis* **by** *this*
qed
qed
qed

lemma *member-rev*:
shows *List.member* (*rev xs*) *x* = *List.member xs x*
proof (*induct xs*)
show *List.member* (*rev* []) *x* = *List.member* [] *x* **by** *simp*
next
case (*Cons a xs*)
assume *HI*:*List.member* (*rev xs*) *x* = *List.member xs x*
have *List.member* (*rev* (*a*#*xs*)) *x* = *List.member* ((*rev xs*)@[*a*]) *x* **using** *rev-append* **by** *auto*
also have ... = ($x \in$ *set* ((*rev xs*) @ [*a*])) **using** *List.member-def* **by** *metis*

also have ... = ($x \in$ *set* (*rev xs*) $\cup$ *set* [*a*]) **using** *set-append* **by** *metis*
also have ... = ($x \in$ *set* [*a*] $\vee$ $x \in$ *set* (*rev xs*)) **by** *blast*
also have ... = ($x = a \vee$ *List.member* (*rev xs*) *x*) **using** *List.member-def* **by** *fastforce*
also have ... = ($x = a \vee$ *List.member xs x*) **using** *HI* **by** *metis*
also have ... = *List.member* (*a#xs*) *x* **using** *List.member-rec*(*1*) **by** *metis*
finally show *List.member* (*rev* (*a#xs*)) *x* = *List.member* (*a#xs*) *x* **by** *this*
qed

lemma *kron-j*:
shows *kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩) (*map nat* (*rev* [*1*..*n*])) =
kron (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗(*complex-of-nat* (*j mod 2*^*n*))/(*2*^*l*)) $\cdot_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*]))
proof −
define *fj fjm* **where** *fj* = (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩)
and *fjm* = (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗(*complex-of-nat* (*j mod 2*^*n*))/(*2*^*l*)) $\cdot_m$ |*one*⟩)
have $\forall$ *x*. ((*List.member* (*map nat* (*rev* [*1*..*n*])) *x*) ⟶ (*x* < *Suc n*))
proof (*rule allI*)
fix *x*
show *List.member* (*map nat* (*rev* [*1*..*int n*])) *x* ⟶ *x* < *Suc n*
proof
assume *List.member* (*map nat* (*rev* [*1*..*int n*])) *x*
hence *List.member* (*rev* (*map nat* [*1*..*int n*])) *x* **using** *rev-map* **by** *metis*
hence *List.member* (*map nat* [*1*..*int n*]) *x* **using** *member-rev* **by** *metis*
hence $x \in$ *set* (*map nat* [*1*..*int n*]) **using** *List.member-def* **by** *metis*
hence $x \in$ {*1*..*n*} **by** *auto*
thus *x* < *Suc n* **by** *auto*
qed
qed
hence $\forall$ *x*. ((*List.member* (*map nat* (*rev* [*1*..*n*])) *x*) ⟶
(*exp* (*2*∗i∗*pi*∗*j*/(*2*^*x*)) = *exp* (*2*∗i∗*pi*∗(*j mod 2*^*n*)/(*2*^*x*))))
using *exp-j*
by (*metis* (*mono-tags*, *lifting*) *of-int-of-nat-eq of-nat-numeral of-nat-power zmod-int*)
hence $\forall$ *x*. ((*List.member* (*map nat* (*rev* [*1*..*n*])) *x*) ⟶ (*fj x* = *fjm x*))
using *fj-def fjm-def* **by** *presburger*
hence *kron fj* (*map nat* (*rev* [*1*..*n*])) = *kron fjm* (*map nat* (*rev* [*1*..*n*]))
by *simp*
thus *?thesis* **using** *fj-def fjm-def* **by** *auto*
qed

We proof that the QFT circuit is correct:

theorem *QFT-is-correct*:
shows $\forall$ *j*. *j* < *2*^*n* ⟶ (*QFT n*) ∗ |*state-basis n j*⟩ = *reverse-QFT-product-representation j n*

```
proof (induct n rule: QFT.induct)
  case 1
  thus ?case
  proof (rule allI)
    fix j::nat
    show j < 2 ^ 0 ⟶ QFT 0 ∗ |state-basis 0 j⟩ = reverse-QFT-product-representation
j 0
    proof
      assume j < 2 ^ 0
      hence j0:j = 0 by auto
      have QFT 0 ∗ |state-basis 0 j⟩ = (1_m 1) ∗ |state-basis 0 j⟩ using QFT.simps
by simp
      also have ... = |unit-vec 1 j⟩ using state-basis-def
        by (metis left-mult-one-mat power-0 state-basis-carrier-mat)
      also have ... = (1_m 1) using unit-vec-def unit-vec-carrier ket-vec-def j0 by
auto
      also have ... = reverse-QFT-product-representation j 0
        using reverse-QFT-product-representation-def by auto
      finally show QFT 0 ∗ |state-basis 0 j⟩ = reverse-QFT-product-representation
j 0 by this
    qed
  qed
next
  case 2
  thus ?case
  proof (rule allI)
    fix j::nat
    show j < 2 ^ Suc 0 ⟶
          QFT (Suc 0) ∗
          |state-basis (Suc 0) j⟩ =
          reverse-QFT-product-representation j
           (Suc 0)
    proof
      assume a1:j < 2^Suc 0
      then show QFT (Suc 0) ∗ |state-basis (Suc 0) j⟩ =
                reverse-QFT-product-representation j (Suc 0)
      proof −
        have QFT (Suc 0) ∗ |state-basis (Suc 0) j⟩ = H ∗ |unit-vec (2^(Suc 0)) j⟩
          using QFT.simps(2) state-basis-def by auto
        also have ... = reverse-QFT-product-representation j (Suc 0)
        proof (rule disjE)
          show j=0 ∨ j=1 using a1 by auto
        next
          assume j0:j=0
          hence H ∗ |unit-vec (2^(Suc 0)) j⟩ = H ∗ |unit-vec (2^(Suc 0)) 0⟩ by
simp
          also have ... = H ∗ |zero⟩ by auto
          also have ... = mat-of-cols-list 2 [[1/sqrt(2),1/sqrt(2)]]
            using H-on-ket-zero by simp
```

```
        also have ... = 1/sqrt(2) ·m (mat-of-cols-list 2 [[1,1]])
        proof
          fix i j::nat
            define ψ1 ψ2 where ψ1 = mat-of-cols-list 2 [[1/sqrt(2),1/sqrt(2)]]
and
                        ψ2 = 1/sqrt(2) ·m (mat-of-cols-list 2 [[1,1]])
          assume i < dim-row ψ2 and j < dim-col ψ2
          hence a2:i ∈ {0,1} ∧ j=0
               by (simp add: Tensor.mat-of-cols-list-def ψ2-def less-Suc-eq-0-disj
numerals(2))
          have ψ1 $$ (0,0) = 1/sqrt 2 using mat-of-cols-list-def ψ1-def by simp
        moreover have ψ1 $$ (1,0) = 1/sqrt 2 using mat-of-cols-list-def ψ1-def
by simp
          moreover have ψ2 $$ (0,0) = 1/sqrt 2
            using smult-mat-def mat-of-cols-list-def ψ2-def by simp
          moreover have ψ2 $$ (1,0) = 1/sqrt 2
            using smult-mat-def mat-of-cols-list-def ψ2-def by simp
          ultimately show ψ1 $$ (i,j) = ψ2 $$ (i,j) using a2 by auto
        next
            define ψ1 ψ2 where ψ1 = mat-of-cols-list 2 [[1/sqrt(2),1/sqrt(2)]]
and
                        ψ2 = 1/sqrt(2) ·m (mat-of-cols-list 2 [[1,1]])
       show dim-row ψ1 = dim-row ψ2 using ψ1-def ψ2-def Tensor.mat-of-cols-list-def
by simp
        next
            define ψ1 ψ2 where ψ1 = mat-of-cols-list 2 [[1/sqrt(2),1/sqrt(2)]]
and
                        ψ2 = 1/sqrt(2) ·m (mat-of-cols-list 2 [[1,1]])
       show dim-col ψ1 = dim-col ψ2 using ψ1-def ψ2-def Tensor.mat-of-cols-list-def
by simp
        qed
        also have ... = 1/sqrt 2 ·m ( |zero⟩ + |one⟩)
        proof −
          have mat-of-cols-list 2 [[1,1]] = |zero⟩ + |one⟩
          proof
            fix i j::nat
            define s1 s2 where s1 = mat-of-cols-list 2 [[1,1]] and s2 = |zero⟩ +
|one⟩
            assume i < dim-row s2 and j < dim-col s2
            hence i ∈ {0,1} ∧ j = 0 using index-add-mat
              by (simp add: ket-vec-def less-Suc-eq numerals(2) s2-def)
            thus s1 $$ (i,j) = s2 $$ (i,j) using s1-def s2-def mat-of-cols-list-def
                 ‹i < dim-row s2› ket-one-to-mat-of-cols-list by force
          next
            define s1 s2 where s1 = mat-of-cols-list 2 [[1,1]] and s2 = |zero⟩ +
|one⟩
              thus dim-row s1 = dim-row s2 using mat-of-cols-list-def by (simp
add: ket-vec-def)
          next
```

define *s1 s2* **where** *s1* = *mat-of-cols-list 2* [[*1*,*1*]] **and** *s2* = |*zero*⟩ + |*one*⟩
thus *dim-col s1* = *dim-col s2* **using** *mat-of-cols-list-def* **by** (*simp add*: *ket-vec-def*)
qed
thus *?thesis* **by** *simp*
qed
also have ... = *1*/*sqrt 2* $\cdot_m$ (*kron* (λ *l*. |*zero*⟩ + |*one*⟩) [*1*]) **using** *kron.simps* **by** *auto*
also have ... = *1*/*sqrt 2* $\cdot_m$ (*kron* (λ *l*. |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*0*/(*2*^*l*)) $\cdot_m$ |*one*⟩) [*1*])
using *exp-zero smult-mat-def* **by** *auto*
also have ... = *reverse-QFT-product-representation 0* (*Suc 0*)
using *reverse-QFT-product-representation-def rev-def map-def* **by** *auto*
finally show *H* ∗ |*unit-vec* (*2* ^ *Suc 0*) *j*⟩ = *reverse-QFT-product-representation j* (*Suc 0*)
using *j0* **by** *simp*
next
assume *j1*:*j* = *1*
hence *H* ∗ |*unit-vec* (*2* ^ *Suc 0*) *j*⟩ = *H* ∗ |*one*⟩ **by** *simp*
also have ... = *mat-of-cols-list 2* [[*1*/*sqrt*(*2*), −*1*/*sqrt*(*2*)]] **using** *H-on-ket-one* **by** *simp*
also have ... = *1*/*sqrt 2* $\cdot_m$ (*mat-of-cols-list 2* [[*1*,−*1*]])
proof
fix *i j*::*nat*
define *φ1 φ2* **where** *φ1* = *mat-of-cols-list 2* [[*1*/*sqrt*(*2*), −*1*/*sqrt*(*2*)]] **and**
φ2 = *1*/*sqrt 2* $\cdot_m$ (*mat-of-cols-list 2* [[*1*,−*1*]])
assume *i* < *dim-row φ2* **and** *j* < *dim-col φ2*
hence *a3*:*i* ∈ {*0*,*1*} ∧ *j* = *0*
using *φ2-def mat-of-cols-list-def numerals*(*2*) *less-2-cases* **by** *simp*
have *φ1* $$ (*0*,*0*) = *φ2* $$ (*0*,*0*)
using *φ1-def φ2-def smult-def mat-of-cols-list-def* **by** *simp*
moreover have *φ1* $$ (*1*,*0*) = *φ2* $$ (*1*,*0*)
using *φ1-def φ2-def smult-def mat-of-cols-list-def* **by** *simp*
ultimately show *φ1* $$ (*i*,*j*) = *φ2* $$ (*i*,*j*) **using** *a3* **by** *auto*
next
define *φ1 φ2* **where** *φ1* = *mat-of-cols-list 2* [[*1*/*sqrt*(*2*), −*1*/*sqrt*(*2*)]] **and**
φ2 = *1*/*sqrt 2* $\cdot_m$ (*mat-of-cols-list 2* [[*1*,−*1*]])
then show *dim-row φ1* = *dim-row φ2* **using** *smult-def mat-of-cols-list-def* **by** *simp*
next
define *φ1 φ2* **where** *φ1* = *mat-of-cols-list 2* [[*1*/*sqrt*(*2*), −*1*/*sqrt*(*2*)]] **and**
φ2 = *1*/*sqrt 2* $\cdot_m$ (*mat-of-cols-list 2* [[*1*,−*1*]])
then show *dim-col φ1* = *dim-col φ2* **using** *smult-def mat-of-cols-list-def* **by** *simp*
qed

also have ... = *1/sqrt 2* $\cdot_m$ (|*zero*⟩ − |*one*⟩)
proof −
have *mat-of-cols-list 2* [[*1*,−*1*]] = |*zero*⟩ − |*one*⟩
proof
fix *i j*::*nat*
define *r1 r2* **where** *r1* = *mat-of-cols-list 2* [[*1*,−*1*]] **and** *r2* = |*zero*⟩ − |*one*⟩
assume *i* < *dim-row r2* **and** *j* < *dim-col r2*
hence *a4*:*i* ∈ {*0*,*1*} ∧ *j*=*0*
using *ket-vec-def index-add-mat* **by** (*simp add*: *less-2-cases r2-def*)
have *r1* $$ (*0*,*0*) = *r2* $$ (*0*,*0*) **using** *r1-def r2-def mat-of-cols-list-def*
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def add.commute diff-zero dim-row-mat*(*1*)
index-mat(*1*) *index-mat-of-cols-list ket-one-is-state ket-one-to-mat-of-cols-list*

ket-zero-to-mat-of-cols-list list.size(*3*) *list.size*(*4*) *minus-mat-def nth-Cons-0*
plus-1-eq-Suc pos2 state-def zero-less-one-class.zero-less-one)
moreover have *r1* $$ (*1*,*0*) = *r2* $$ (*1*,*0*)
using *r1-def r2-def mat-of-cols-list-def ket-vec-def* **by** *simp*
ultimately show *r1* $$ (*i*,*j*) = *r2* $$ (*i*,*j*) **using** *a4* **by** *auto*
next
define *r1 r2* **where** *r1* = *mat-of-cols-list 2* [[*1*,−*1*]] **and** *r2* = |*zero*⟩ − |*one*⟩
thus *dim-row r1* = *dim-row r2* **using** *mat-of-cols-list-def ket-vec-def* **by** *simp*
next
define *r1 r2* **where** *r1* = *mat-of-cols-list 2* [[*1*,−*1*]] **and** *r2* = |*zero*⟩ − |*one*⟩
thus *dim-col r1* = *dim-col r2* **using** *mat-of-cols-list-def ket-vec-def* **by** *simp*
qed
thus *?thesis* **by** *simp*
qed
also have ... = *1/sqrt 2* $\cdot_m$ (*kron* (λ*l*. |*zero*⟩ − |*one*⟩) [*1*])
using *kron.simps* **by** *auto*
also have ... = *1/sqrt 2* $\cdot_m$ (*kron* (λ*l*. |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*1*/(*2*^*l*)) $\cdot_m$ |*one*⟩) [*1*])
proof −
have *exp* (*2*∗i∗*pi*∗*1*/(*2*^*1*)) = −*1* **using** *exp-pi-i* **by** *auto*
hence |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*1*/(*2*^*1*)) $\cdot_m$ |*one*⟩ = |*zero*⟩ + (−*1*) $\cdot_m$ |*one*⟩ **by** *simp*
also have ... = |*zero*⟩ − |*one*⟩ **by** *auto*
thus *?thesis* **by** *auto*
qed
also have ... = *reverse-QFT-product-representation 1* (*Suc 0*)
using *reverse-QFT-product-representation-def* **by** *auto*
finally show *H* ∗ |*unit-vec* (*2* ^ *Suc 0*) *j*⟩ = *reverse-QFT-product-representation j* (*Suc 0*)

using *j1* **by** *simp*
qed
finally show *?thesis* **by** *this*
qed
qed
qed
next
case *3*
fix n'::*nat*
define *n* **where** $n = Suc\ n'$
assume *HI*:$\forall j<2\ \hat{}\ n.$ *QFT n* $*$ $|$*state-basis n j*$\rangle$ = *reverse-QFT-product-representation j n*
show $\forall j<2\hat{}Suc\ n.$
QFT (*Suc n*) $*$ $|$*state-basis* (*Suc n*) *j*$\rangle$ = *reverse-QFT-product-representation j* (*Suc n*)
proof (*rule allI*)
fix *j*::*nat*
show $j < 2\ \hat{}\ Suc\ n \longrightarrow$ *QFT* (*Suc n*) $*$ $|$*state-basis* (*Suc n*) *j*$\rangle$ =
reverse-QFT-product-representation j (*Suc n*)
proof
assume *aj*:$j < 2\ \hat{}\ Suc\ n$
show *QFT* (*Suc n*) $*$
$|$*state-basis* (*Suc n*) *j*$\rangle$ =
reverse-QFT-product-representation j
(*Suc n*)
proof −
define *jd jm* **where** $jd = j\ div\ 2\hat{}n$ **and** $jm = j\ mod\ 2\hat{}n$
hence $jm < 2\hat{}n$ **by** *auto*
hence *HI-jm*:*QFT n* $*$ $|$*state-basis n jm*$\rangle$ = *reverse-QFT-product-representation jm n*
using *HI* **by** *auto*
have (*QFT* (*Suc n*)) $*$ $|$*state-basis* (*Suc n*) *j*$\rangle$ =
(((1_m *2*) $\bigotimes$ (*QFT n*)) $*$ (*controlled-rotations* (*Suc n*)) $*$ (*H* $\bigotimes$ ((1_m ($2\hat{}n$))))) $*$
$|$*state-basis* (*Suc n*) *j*$\rangle$
using *QFT.simps*(*3*) *n-def* **by** *simp*
also have ... = (((1_m *2*) $\bigotimes$ (*QFT n*)) $*$ (*controlled-rotations* (*Suc n*))) $*$
(((*H* $\bigotimes$ ((1_m ($2\hat{}n$))))) $*$ $|$*state-basis* (*Suc n*) *j*$\rangle$)
proof (*rule assoc-mult-mat*)
show (1_m *2* $\bigotimes$ *QFT n*) $*$ *controlled-rotations* (*Suc n*) $\in$ *carrier-mat* ($2\hat{}$(*Suc n*)) ($2\hat{}$(*Suc n*))
proof (*rule mult-carrier-mat*)
show 1_m *2* $\bigotimes$ *QFT n* $\in$ *carrier-mat* (*2* $\hat{}$ *Suc n*) (*2* $\hat{}$ *Suc n*) **by** *simp*
show *controlled-rotations* (*Suc n*) $\in$ *carrier-mat* (*2* $\hat{}$ *Suc n*) (*2* $\hat{}$ *Suc n*)
using *controlled-rotations-carrier-mat* **by** *blast*
qed
next
show *H* $\bigotimes$ 1_m (*2* $\hat{}$ *n*) $\in$ *carrier-mat* (*2* $\hat{}$ *Suc n*) (*2* $\hat{}$ *Suc n*)
using *tensor-carrier-mat*

by (*metis QFT.simps*(*2*) *QFT-carrier-mat one-carrier-mat power-Suc power-Suc0-right*)
next
show $|$*state-basis* (*Suc n*) *j*$\rangle \in$ *carrier-mat* (*2* ^ *Suc n*) *1*
using *state-basis-carrier-mat* **by** *blast*
qed
also have ... = (((*1*$_m$ *2*) $\bigotimes$ (*QFT n*)) $*$ (*controlled-rotations* (*Suc n*))) $*$
((*1*/*sqrt 2* $\cdot_m$ ($|$*zero*$\rangle$ + *exp*(*2*$*$i$*$*pi*$*$*jd*/*2*) $\cdot_m$ $|$*one*$\rangle$))) $\bigotimes$
$|$*state-basis n jm*$\rangle$)
using *aj H-on-first-qubit jd-def jm-def* **by** *simp*
also have ... = ((*1*$_m$ *2*) $\bigotimes$ (*QFT n*)) $*$ (*controlled-rotations* (*Suc n*) $*$
(((*1*/*sqrt 2* $\cdot_m$ ($|$*zero*$\rangle$ + *exp*(*2*$*$i$*$*pi*$*$*jd*/*2*) $\cdot_m$ $|$*one*$\rangle$))) $\bigotimes$
$|$*state-basis n jm*$\rangle$)))
using *assoc-mult-mat tensor-carrier-mat QFT-carrier-mat one-carrier-mat*

state-basis-carrier-mat
by (*smt* (*verit*, *ccfv-threshold*) *H-on-first-qubit QFT.simps*(*2*) *aj*
controlled-rotations-carrier-mat jd-def jm-def mult-carrier-mat power-Suc

power-Suc0-right)
also have ... = ((*1*$_m$ *2*) $\bigotimes$ (*QFT n*)) $*$
(*1*/*sqrt 2* $\cdot_m$ (($|$*zero*$\rangle$ + *exp*(*2*$*$i$*$*pi*$*$*j*/(*2*^(*Suc n*))) $\cdot_m$ $|$*one*$\rangle$))
$\bigotimes$
$|$*state-basis n jm*$\rangle$)
using *controlled-rotations-on-first-qubit aj jd-def jm-def* **by** *simp*
also have ... = ((*1*$_m$ *2*) $*$ (*1*/*sqrt 2* $\cdot_m$ (($|$*zero*$\rangle$ + *exp*(*2*$*$i$*$*pi*$*$*j*/(*2*^(*Suc n*))) $\cdot_m$ $|$*one*$\rangle$)))) $\bigotimes$
((*QFT n*) $*$ $|$*state-basis n jm*$\rangle$)
proof −
have *dim-col* (*1*$_m$ *2*) = *dim-row* (*1*/*sqrt 2* $\cdot_m$ (($|$*zero*$\rangle$ + *exp*(*2*$*$i$*$*pi*$*$*j*/(*2*^(*Suc n*))) $\cdot_m$ $|$*one*$\rangle$)))
proof −
have *dim-col* (*1*$_m$ *2*) = *2* **by** *simp*
moreover have *dim-row* (*1*/*sqrt 2* $\cdot_m$ (($|$*zero*$\rangle$ + *exp*(*2*$*$i$*$*pi*$*$*j*/(*2*^(*Suc n*))) $\cdot_m$ $|$*one*$\rangle$))) = *2*
using *smult-carrier-mat mat-of-cols-list-def add-carrier-mat ket-vec-def* **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
moreover have *dim-col* (*QFT n*) = *dim-row* $|$*state-basis n jm*$\rangle$
using *state-basis-carrier-mat QFT-carrier-mat*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
moreover have *dim-col* (*1*$_m$ *2*) > *0* **by** *simp*
moreover have *dim-col* (*1*/*sqrt 2* $\cdot_m$ (($|$*zero*$\rangle$ + *exp*(*2*$*$i$*$*pi*$*$*j*/(*2*^(*Suc n*))) $\cdot_m$ $|$*one*$\rangle$))) > *0*
using *smult-carrier-mat mat-of-cols-list-def add-carrier-mat ket-vec-def* **by** *simp*
moreover have *dim-col* (*QFT n*) > *0* **using** *QFT-carrier-mat*
by (*metis carrier-matD*(*2*) *pos2 zero-less-power*)

moreover have *dim-col* |*state-basis n jm*⟩ > *0* **using** *state-basis-carrier-mat*
by (*simp add*: *ket-vec-def*)
ultimately show ((*1*$_m$ *2*) ⨂ (*QFT n*)) ∗
(*1*/*sqrt 2* ·$_m$ ((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$ |*one*⟩)) ⨂
|*state-basis n jm*⟩) =
((*1*$_m$ *2*) ∗ (*1*/*sqrt 2* ·$_m$ ((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$
|*one*⟩)))) ⨂
((*QFT n*) ∗ |*state-basis n jm*⟩)
using *mult-distr-tensor* **by** (*smt* (*verit*, *best*))
qed
also have . . . = (*1*/*sqrt 2* ·$_m$ ((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$
|*one*⟩))) ⨂
reverse-QFT-product-representation jm n
using *ket-one-is-state state.dim-row HI-jm* **by** *auto*
also have . . . = *reverse-QFT-product-representation j* (*Suc n*)
proof −
have (*1*/*sqrt 2* ·$_m$ ((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$ |*one*⟩))) ⨂
reverse-QFT-product-representation jm n =
(*1*/*sqrt 2* ·$_m$ ((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$ |*one*⟩))) ⨂
(*1*/*sqrt* (*2*^*n*) ·$_m$ (*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*jm*/(*2*^*l*)) ·$_m$
|*one*⟩)
(*map nat* (*rev* [*1*..*n*]))))
using *reverse-QFT-product-representation-def* **by** *simp*
also have . . . = (*1*/*sqrt 2* ·$_m$ ((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$
|*one*⟩))) ⨂
(*1*/*sqrt* (*2*^*n*) ·$_m$ (*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*))
·$_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*]))))
using *kron-j jm-def* **by** *simp*
also have . . . = ((*1*/*sqrt 2*)∗(*1*/*sqrt* (*2*^*n*))) ·$_m$
(((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$ |*one*⟩)) ⨂
(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) ·$_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*]))))
proof −
have *dim-col* (|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$ |*one*⟩) > *0*
by (*simp add*: *ket-vec-def*)
moreover have *dim-col* (*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*))
·$_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*]))) > *0*
using *kron-carrier-mat ket-vec-def*
by (*metis* (*no-types*, *lifting*) *calculation carrier-matD*(*2*) *dim-col-mat*(*1*)

dim-row-mat(*1*) *index-add-mat*(*2*) *index-add-mat*(*3*) *index-smult-mat*(*2*)

index-smult-mat(*3*) *index-unit-vec*(*3*))
ultimately show *?thesis* **by** *simp*
qed
also have . . . = (*1*/*sqrt* (*2*^(*Suc n*))) ·$_m$
(((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) ·$_m$ |*one*⟩)) ⨂

(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*]))))
by (*simp add*: *real-sqrt-mult*)
also have ... = (*1*/*sqrt* (*2*^(*Suc n*))) $\cdot_m$
(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩)
(*map nat* (*rev* [*1*..(*Suc n*)])))
proof −
define *f* **where** *f* = (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩)
hence |*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) $\cdot_m$ |*one*⟩ = *f* (*Suc n*) **by** *simp*
hence (((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) $\cdot_m$ |*one*⟩)) ⨂
(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*])))) =
(*f* (*Suc n*)) ⨂ (*kron f* (*map nat* (*rev* [*1*..*n*])))
using *f-def* **by** *simp*
also have ... = *kron f* ((*Suc n*)#(*map nat* (*rev* [*1*..*n*])))
using *kron.simps*(*2*) **by** *simp*
also have ... = *kron f* (*map nat* (*rev* [*1*..(*Suc n*)]))
using *map-def rev-append*
by (*smt* (*z3*) *append-Cons append-self-conv2 list.simps*(*9*) *nat-int*
negative-zless
of-nat-Suc rev-eq-Cons-iff rev-is-Nil-conv upto-rec2)
finally have (((|*zero*⟩ + *exp*(*2*∗i∗*pi*∗*j*/(*2*^(*Suc n*))) $\cdot_m$ |*one*⟩)) ⨂
(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩)
(*map nat* (*rev* [*1*..*n*])))) =
(*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩)
(*map nat* (*rev* [*1*..(*Suc n*)])))
using *f-def* **by** *simp*
thus *?thesis* **by** *simp*
qed
also have ... = *reverse-QFT-product-representation j* (*Suc n*)
using *reverse-QFT-product-representation-def* **by** *simp*
finally show *?thesis* **by** *this*
qed
finally show *?thesis* **by** *this*
qed
qed
qed
qed

7.1 QFT with qubits reordering correctness

lemma *SWAP-down-kron*:
assumes ∀ *m*. *dim-row* (*f m*) = *2* ∧ *dim-col* (*f m*) = *1*
shows *SWAP-down* (*length* (*x*#*xs*)) ∗ *kron f* (*x*#*xs*) = *kron f xs* ⨂ *f x*
proof (*induct xs rule*: *rev-induct*)
case *Nil*
have *SWAP-down* (*length* [*x*]) ∗ *kron f* [*x*] = (*1*$_m$ *2*) ∗ *f x* **using** *SWAP-down.simps*(*2*)
kron.simps(*2*)
by (*metis carrier-matI kron.simps*(*1*) *length-0-conv length-Cons right-tensor-id*)

also have ... = $f\ x$ using *left-mult-one-mat′ assms* by *auto*
also have ... = $(1_m\ 1) \bigotimes f\ x$ using *left-tensor-id* by *auto*
also have ... = $kron\ f\ [] \bigotimes f\ x$ using *kron.simps* by *auto*
finally show *?case* by *this*
next
case (*snoc a xs*)
assume *HI*:$SWAP\text{-}down\ (length\ (x\#xs)) * kron\ f\ (x\#xs) = kron\ f\ xs \bigotimes f\ x$
define n::*nat* where $n = length\ xs$
show *?case*
proof (*cases*)
assume *Nil*:$xs = []$
hence $n = 0$ using *n-def* by *auto*
have $SWAP\text{-}down\ (length\ (x\#xs@[a])) * kron\ f\ (x\#xs@[a]) =$
$SWAP\text{-}down\ (Suc\ (Suc\ 0)) * kron\ f\ (x\#[a])$
using *n-def Nil* by *simp*
also have ... = $SWAP * kron\ f\ (x\#[a])$ using *SWAP-down.simps(3)* by *simp*
also have ... = $SWAP * ((f\ x) \bigotimes (f\ a))$ using *kron.simps(2)*
by (*metis carrier-matI kron.simps(1) right-tensor-id*)
also have ... = $(f\ a) \bigotimes (f\ x)$ using *SWAP-tensor assms* by *auto*
also have ... = $kron\ f\ (xs@[a]) \bigotimes (f\ x)$ using *kron.simps Nil*
by (*metis carrier-mat-triv kron-cons-right left-tensor-id*)
finally show *?case* by *this*
next
assume *NNil*:$xs \neq []$
hence $n > 0$ using *n-def* by *auto*
hence *e*:$\exists\, m.\ n = Suc\ m$ by (*simp add: gr0-implies-Suc*)
have $SWAP\text{-}down\ (length\ (x\#xs@[a])) * kron\ f\ (x\#xs@[a]) =$
$SWAP\text{-}down\ (Suc\ (Suc\ n)) * kron\ f\ (x\#xs@[a])$
using *n-def* by *auto*
also have ... = $((1_m\ (2\hat{}n)) \bigotimes SWAP) * ((SWAP\text{-}down\ (Suc\ n)) \bigotimes (1_m\ 2)) * kron\ f\ (x\#xs@[a])$
using *SWAP-down.simps e* by *auto*
also have ... = $((1_m\ (2\hat{}n)) \bigotimes SWAP) * (((SWAP\text{-}down\ (Suc\ n)) \bigotimes (1_m\ 2)) * kron\ f\ (x\#xs@[a]))$
proof (*rule assoc-mult-mat*)
show $((1_m\ (2\hat{}n)) \bigotimes SWAP) \in carrier\text{-}mat\ (2\hat{}(Suc\ (Suc\ n)))\ (2\hat{}(Suc\ (Suc\ n)))$
proof −
have $(1_m\ (2\hat{}n)) \in carrier\text{-}mat\ (2\hat{}n)\ (2\hat{}n)$ by *simp*
moreover have $SWAP \in carrier\text{-}mat\ 4\ 4$ using *SWAP-carrier-mat* by *simp*
ultimately show *?thesis* using *tensor-carrier-mat*
by (*smt (verit, ccfv-threshold) mult-numeral-left-semiring-numeral num-double*

numeral-times-numeral power-Suc power-commuting-commutes)
qed
next
show $SWAP\text{-}down\ (Suc\ n) \bigotimes 1_m\ 2 \in carrier\text{-}mat\ (2\ \hat{}\ Suc\ (Suc\ n))\ (2\ \hat{}\ Suc\ (Suc\ n))$

proof −
have *SWAP-down* (*Suc n*) ∈ *carrier-mat* (*2*^(*Suc n*)) (*2*^(*Suc n*)) **using** *SWAP-down-carrier-mat*
by *blast*
moreover have 1_m *2* ∈ *carrier-mat 2 2* **by** *simp*
ultimately show *?thesis* **using** *tensor-carrier-mat* **by** *auto*
qed
next
show *kron f* (*x* # *xs* @ [*a*]) ∈ *carrier-mat* (*2* ^ *Suc* (*Suc n*)) *1* **using** *kron-carrier-mat*
by (*metis assms length-Cons length-append-singleton n-def*)
qed
also have … = ((1_m (*2*^*n*)) ⨂ *SWAP*) ∗ (((*SWAP-down* (*Suc n*)) ⨂ (1_m *2*)) ∗
(*kron f* (*x*#*xs*) ⨂ *f a*))
using *kron.simps* **by** (*metis append-Cons kron-cons-right*)
also have … = ((1_m (*2*^*n*)) ⨂ *SWAP*) ∗ (((*SWAP-down* (*Suc n*))∗(*kron f* (*x*#*xs*))) ⨂
(1_m *2*) ∗ (*f a*))
proof −
have *c1*:*dim-col* (*SWAP-down* (*Suc n*)) = *2*^(*Suc n*) **using** *SWAP-down-carrier-mat* **by** *blast*
hence *a3*: *dim-col* (*SWAP-down* (*Suc n*)) > *0* **by** *simp*
have *r2*:*dim-row* (*kron f* (*x*#*xs*)) = *2*^(*Suc n*) **using** *kron-carrier-mat assms n-def* **by** *auto*
hence *a4*:*dim-row* (*kron f* (*x*#*xs*)) > *0* **by** *simp*
with *c1 r2* **have** *a1*:*dim-col* (*SWAP-down* (*Suc n*)) = *dim-row* (*kron f* (*x*#*xs*)) **by** *simp*
have *c3*:*dim-col* (1_m *2*) = *2* **by** *simp*
hence *a5*:*dim-col* (1_m *2*) > *0* **by** *simp*
have *r4*:*dim-row* (*f a*) = *2* **using** *assms* **by** *simp*
hence *a6*:*dim-row* (*f a*) > *0* **by** *simp*
with *c3 r4* **have** *a2*:*dim-col* (1_m *2*) = *dim-row* (*f a*) **by** *simp*
have (((*SWAP-down* (*Suc n*)) ⨂ (1_m *2*)) ∗ (*kron f* (*x*#*xs*) ⨂ *f a*)) =
(((*SWAP-down* (*Suc n*))∗(*kron f* (*x*#*xs*))) ⨂ (1_m *2*) ∗ (*f a*))
using *a1 a2 a3 a4 a5 a6*
by (*metis assms carrier-matD*(*2*) *gr0I kron-carrier-mat mult-distr-tensor zero-neq-one*)
thus *?thesis* **by** *simp*
qed
also have … = ((1_m (*2*^*n*)) ⨂ *SWAP*) ∗ (*kron f xs* ⨂ *f x* ⨂ *f a*)
using *HI* **by** (*simp add*: *assms n-def*)
also have … = ((1_m (*2*^*n*)) ⨂ *SWAP*) ∗ (*kron f xs* ⨂ (*f x* ⨂ *f a*))
using *tensor-mat-is-assoc* **by** *auto*
also have … = ((1_m (*2*^*n*)) ∗ (*kron f xs*)) ⨂ (*SWAP* ∗ (*f x* ⨂ *f a*))
using *mult-distr-tensor*
by (*smt* (*verit*, *del-insts*) *SWAP-ncols assms carrier-matD*(*2*) *dim-col-tensor-mat*

dim-row-tensor-mat index-mult-mat(*2*) *index-one-mat*(*2*) *index-one-mat*(*3*)

kron-carrier-mat
left-mult-one-mat n-def numeral-One numeral-times-numeral semiring-norm(*11*)

semiring-norm(*13*) *zero-less-numeral zero-less-power*)
also have ... = *kron f xs* $\bigotimes$ *f a* $\bigotimes$ *f x* **using** *SWAP-tensor*
by (*metis assms carrier-matI kron-carrier-mat left-mult-one-mat n-def tensor-mat-is-assoc*)
also have ... = *kron f* (*xs*@[*a*]) $\bigotimes$ *f x* **using** *kron.simps kron-cons-right* **by** *presburger*
finally show *?thesis* **by** *this*
qed
qed

lemma *SWAP-down-kron-map-rev*:
assumes ∀ *m. dim-row* (*f m*) = *2* ∧ *dim-col* (*f m*) = *1*
shows (*SWAP-down* (*Suc k*)) ∗
kron f (*map nat* (*rev* [*1*..*int* (*Suc k*)])) =
(*kron f* (*map nat* (*rev* [*1*..*int k*])) $\bigotimes$ (*f* (*Suc k*)))
proof −
have *rev* [*1*..*int* (*Suc k*)] = *int* (*Suc k*) # *rev* [*1*..*int k*] **using** *rev-append upto-rec2* **by** *simp*
hence *1*:*map nat* (*rev* [*1*..*int* (*Suc k*)]) = *Suc k* # (*map nat* (*rev* [*1*.. *int k*]))
using *list.map*(*2*) **by** *simp*
define *x xs* **where** *x* = *Suc k* **and** *xs* = (*map nat* (*rev* [*1*.. *int k*]))
have *length xs* = *k* **using** *xs-def* **by** *simp*
hence *2*:*length* (*x*#*xs*) = *Suc k* **by** *simp*
with *1 2 x-def xs-def* **have** (*SWAP-down* (*Suc k*)) ∗ *kron f* (*map nat* (*rev* [*1*..*int* (*Suc k*)])) =
(*SWAP-down* (*length* (*x*#*xs*))) ∗ *kron f* (*x*#*xs*) **by** *auto*
also have ... = *kron f xs* $\bigotimes$ *f x* **using** *SWAP-down-kron x-def xs-def assms* **by** *blast*
finally show *?thesis* **using** *x-def xs-def* **by** *simp*
qed

lemma *reverse-qubits-kron*:
assumes ∀ *m. dim-row* (*f m*) = *2* ∧ *dim-col* (*f m*) = *1*
shows (*reverse-qubits n*) ∗ (*kron f* (*map nat* (*rev* [*1*..*n*]))) = *kron f* (*map nat* [*1*..*n*])
proof (*induct n rule*: *reverse-qubits.induct*)
case *1*
then show *?case* **by** *auto*
next
case *2*
then show *?case*
proof −
have *1*:*rev* [*1*] = [*1*] **using** *rev-def* **by** *auto*
have *2*:*reverse-qubits* (*Suc 0*) = 1_m *2* **by** *simp*

```
    have 3:(f 1) ∈ carrier-mat 2 1 using assms carrier-mat-def by auto
    have 4:kron f [1] = (f 1) using kron.simps(2) by auto
    show ?case using 1 2 3 4 by auto
  qed
next
  case 3
  have reverse-qubits (Suc (Suc 0)) * kron f (map nat (rev [1..int (Suc (Suc 0))])) =
      SWAP * kron f [2,1]
    using reverse-qubits.simps(3) upto-rec1 by auto
  also have ... = SWAP * ((f 2) ⨂ (f 1))
    using right-tensor-id by (metis carrier-mat-triv kron.simps(1) kron.simps(2))
  also have ... = (f 1) ⨂ (f 2) using SWAP-tensor assms by auto
  also have ... = kron f [1,2] using upto-rec1 assms by auto
  also have ... = kron f (map nat [1..int (Suc (Suc 0))]) using right-tensor-id assms
    by (auto simp add: upto-rec1)
  finally show reverse-qubits (Suc (Suc 0)) * kron f (map nat (rev [1..int (Suc (Suc 0))])) =
              kron f (map nat [1..int (Suc (Suc 0))]) by this
next
  case 4
  fix n::nat
  define k::nat where k = Suc (Suc n)
  assume HI:reverse-qubits (Suc (Suc n)) * kron f (map nat (rev [1..int (Suc (Suc n))])) =
            kron f (map nat [1..int (Suc (Suc n))])
  have sk:(SWAP-down (Suc k)) * kron f (map nat (rev [1..int (Suc k)])) =
      (kron f (map nat (rev [1..int k])) ⨂ (f (Suc k)))
    using SWAP-down-kron-map-rev assms by this
  have reverse-qubits (Suc k) * kron f (map nat (rev [1..int (Suc k)])) =
      ((reverse-qubits k) ⨂ (1_m 2)) * (SWAP-down (Suc k)) *
      kron f (map nat (rev [1..int (Suc k)]))
    using reverse-qubits.simps(4) k-def by simp
  also have ... = ((reverse-qubits k) ⨂ (1_m 2)) * ((SWAP-down (Suc k)) *
         kron f (map nat (rev [1..int (Suc k)])))
  proof (rule assoc-mult-mat)
    show (reverse-qubits k) ⨂ (1_m 2) ∈ carrier-mat (2^(k+1)) (2^(k+1))
    proof -
      have reverse-qubits k ∈ carrier-mat (2^k) (2^k) by simp
      moreover have 1_m 2 ∈ carrier-mat 2 2 by simp
     ultimately show ?thesis using tensor-carrier-mat by (smt (verit) power-add power-one-right)
    qed
  next
    show (SWAP-down (Suc k)) ∈ carrier-mat (2^(k+1)) (2^(k+1))
      using Suc-eq-plus1 SWAP-down-carrier-mat by presburger
  next
    show kron f (map nat (rev [1..int (Suc k)])) ∈ carrier-mat (2 ^ (k + 1)) 1
```

proof −
define *xs* **where** *xs* = (*map nat* (*rev* [*1*..*int* (*Suc k*)]))
then have *k1*:*length xs* = *k* + *1* **by** *auto*
then have *kron f xs* ∈ *carrier-mat* (*2* ^(*k* + *1*)) *1*
using *kron-carrier-mat assms k1* **by** *metis*
thus *?thesis* **using** *xs-def* **by** *simp*
qed
qed
also have ... = ((*reverse-qubits k*) ⨂ (*1$_m$ 2*)) ∗ (*kron f* (*map nat* (*rev* [*1*..*int* *k*])) ⨂ (*f* (*Suc k*)))
using *sk* **by** *simp*
also have ... = ((*reverse-qubits k*) ∗ (*kron f* (*map nat* (*rev* [*1*..*int k*])))) ⨂ ((*1$_m$* *2*) ∗ (*f* (*Suc k*)))
proof −
have *c1*:*dim-col* (*reverse-qubits k*) = *2*^*k* **using** *reverse-qubits-carrier-mat* **by** *blast*
have *r2*:*dim-row* (*kron f* (*map nat* (*rev* [*1*..*int k*]))) = *2*^*k*
using *kron-carrier-mat* **by** (*metis HI assms carrier-matD*(*1*) *index-mult-mat*(*2*) *k-def length-map*
length-rev reverse-qubits-carrier-mat)
with *c1 r2* **have** *a1*:*dim-col* (*reverse-qubits k*) = *dim-row* (*kron f* (*map nat* (*rev* [*1*..*int k*])))
by *auto*
have *c3*:*dim-col* (*1$_m$ 2*) = *2* **by** *simp*
have *r4*:*dim-row* (*f* (*Suc k*)) = *2* **using** *assms* **by** *simp*
with *c3 r4* **have** *a2*:*dim-col* (*1$_m$ 2*) = *dim-row* (*f* (*Suc k*)) **by** *simp*
have *a3*:*dim-col* (*reverse-qubits k*) > *0* **using** *c1* **by** *auto*
have *a4*:*dim-row* (*kron f* (*map nat* (*rev* [*1*..*int k*]))) > *0* **using** *r2* **by** *auto*
have *a5*:*dim-col* (*1$_m$ 2*) > *0* **using** *c3* **by** *auto*
have *a6*:*dim-row* (*f* (*Suc k*)) > *0* **using** *r4* **by** *auto*
show *?thesis* **using** *a1 a2 a3 a4 a5 a6 mult-distr-tensor*
by (*metis assms carrier-matD*(*2*) *kron-carrier-mat zero-less-one-class.zero-less-one*)
qed
also have ... = *kron f* (*map nat* [*1*..*int k*]) ⨂ (*f* (*Suc k*))
using *HI k-def assms* **by** *auto*
also have ... = *kron f* (*map nat* [*1*..*int* (*Suc k*)]) **using** *kron-cons-right*
by (*smt* (*verit*, *ccfv-threshold*) *list.simps*(*8*) *list.simps*(*9*) *map-append nat-int negative-zless*
of-nat-Suc upto-rec2)
finally show *reverse-qubits* (*Suc* (*Suc* (*Suc n*))) ∗
kron f (*map nat* (*rev* [*1*..*int* (*Suc* (*Suc* (*Suc n*)))])) =
kron f (*map nat* [*1*..*int* (*Suc* (*Suc* (*Suc n*)))]) **using** *k-def* **by** *simp*
qed

lemma *prod-rep-fun*:
assumes *f* = (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) ·$_m$ |*one*⟩)
shows ∀ *m*. *dim-row* (*f m*) = *2* ∧ *dim-col* (*f m*) = *1*
apply (*rule allI*)

apply (*rule conjI*)
 apply (*simp add*: *assms ket-vec-def cpx-vec-length-def*)+
done

lemma *rev-upto*:
 assumes $n1 \leq n2$
 shows *rev* $[n1..n2] = n2 \# rev\ [n1..(n2-1)]$
 apply (*simp*)
 apply (*rule upto-rec2*)
 apply (*simp add*:*assms*)
 done

lemma *dim-row-kron*:
 shows *dim-row* (*kron f xs*) = ($\prod x \leftarrow xs.$ *dim-row* (*f x*))
proof (*induct xs*)
 case *Nil*
 show *?case* **using** *kron.simps*(*1*) *prod-list-def* **by** *auto*
next
 case (*Cons a xs*)
 assume *HI*:*dim-row* (*kron f xs*) = ($\prod x \leftarrow xs.$ *dim-row* (*f x*))
 have *dim-row* (*kron f* (*a#xs*)) = *dim-row* ((*f a*) $\bigotimes$ (*kron f xs*)) **using** *kron.simps*(*2*)
by *auto*
 hence ... = (*dim-row* (*f a*)) $*$ (*dim-row* (*kron f xs*)) **by** *auto*
 hence ... = (*dim-row* (*f a*)) $*$ ($\prod x \leftarrow xs.$ *dim-row* (*f x*)) **using** *HI* **by** *auto*
 hence ... = ($\prod x \leftarrow a \# xs.$ *dim-row* (*f x*)) **by** *auto*
 thus *?case* **using** *HI* **by** *auto*
qed

lemma *dim-col-kron*:
 shows *dim-col* (*kron f xs*) = ($\prod x \leftarrow xs.$ *dim-col* (*f x*))
proof (*induct xs*)
 case *Nil*
 show *?case* **using** *kron.simps*(*1*) *prod-list-def* **by** *auto*
next
 case (*Cons a xs*)
 assume *HI*:*dim-col* (*kron f xs*) = ($\prod x \leftarrow xs.$ *dim-col* (*f x*))
 have *dim-col* (*kron f* (*a#xs*)) = *dim-col* ((*f a*) $\bigotimes$ (*kron f xs*)) **using** *kron.simps*(*2*)
by *auto*
 hence ... = (*dim-col* (*f a*)) $*$ (*dim-col* (*kron f xs*)) **by** *auto*
 hence ... = (*dim-col* (*f a*)) $*$ ($\prod x \leftarrow xs.$ *dim-col* (*f x*)) **using** *HI* **by** *auto*
 hence ... = ($\prod x \leftarrow a \# xs.$ *dim-col* (*f x*)) **by** *auto*
 thus *?case* **using** *HI* **by** *auto*
qed

lemma *prod-2-n*:
 ($\prod x \leftarrow$*map nat* (*rev* [*1..int n*]). *2*) = *2* ^ *n*
 apply (*induct n*)
 apply (*simp add*: *rev-upto*)+
 done

lemma *prod-2-n-b*:
$(\prod x \leftarrow map\ nat\ [1..int\ n].\ 2) = 2\ \hat{}\ n$
apply (*induct n*)
apply *simp*
apply (*simp add*: *upto-rec2 power-commutes*)
done

lemma *prod-1-n*:
$(\prod x \leftarrow map\ nat\ (rev\ [1..int\ n]).\ 1) = 1$
apply (*induct n*)
apply (*simp add*: *rev-upto*)+
done

lemma *prod-1-n-b*:
$(\prod x \leftarrow map\ nat\ [1..int\ n].\ Suc\ 0) = Suc\ 0$
apply (*induct n*)
apply *simp*
apply (*simp add*: *upto-rec2*)
done

lemma *reverse-qubits-product-representation*:
reverse-qubits n ∗ *reverse-QFT-product-representation j n* = *QFT-product-representation j n*
proof −
have (*reverse-qubits n*) ∗ *reverse-QFT-product-representation j n* = (*reverse-qubits n*) ∗
$((1/sqrt(2\hat{}n)) \cdot_m kron\ (\lambda l.\ |zero\rangle + exp\ (2*\mathrm{i}*pi*j/2\hat{}l) \cdot_m |one\rangle)\ (map\ nat\ (rev\ [1..int\ n])))$
using *reverse-QFT-product-representation-def* **by** *simp*
also have ... $= (1/sqrt(2\hat{}n)) \cdot_m ((reverse\text{-}qubits\ n) *$
$kron\ (\lambda l.\ |zero\rangle + exp\ (2*\mathrm{i}*pi*j/2\hat{}l) \cdot_m |one\rangle)\ (map\ nat\ (rev\ [1..int\ n])))$
proof (*rule mult-smult-distrib*)
show *reverse-qubits n* ∈ *carrier-mat* (2^n) (2^n) **by** *simp*
next
show $kron\ (\lambda l.\ |zero\rangle + exp\ (2*\mathrm{i}*pi*j/2\hat{}l) \cdot_m |one\rangle)\ (map\ nat\ (rev\ [1..int\ n]))$

∈ *carrier-mat* (2^n) 1
proof
show $dim\text{-}row\ (kron\ (\lambda(l::nat).\ |zero\rangle + exp\ (2*\mathrm{i}*pi*j/(2\hat{}l)) \cdot_m |one\rangle)\ (map\ nat\ (rev\ [1..n])))$
$= 2\ \hat{}\ n$
proof −
have *a1*:$dim\text{-}row\ (kron\ (\lambda l.\ |zero\rangle + exp\ (2 * \mathrm{i} * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ j\ /\ 2\ \hat{}\ l) \cdot_m |one\rangle)\ (map\ nat\ (rev\ [1..int\ n])))$
$= (\prod x \leftarrow (map\ nat\ (rev\ [1..int\ n])).\ (dim\text{-}row\ ((\lambda l.\ |zero\rangle + exp\ (2 * \mathrm{i} * complex\text{-}of\text{-}real\ pi * complex\text{-}of\text{-}nat\ j\ /\ 2\ \hat{}\ l) \cdot_m |one\rangle)\ x)))$
using *dim-row-kron* **by** *simp*

hence *b1*:... = ($\prod$ *x*←(*map nat* (*rev* [*1*..*int n*])). *2*) **using** *prod-rep-fun* **by** *auto*
hence ... = *2* ^ *n* **using** *prod-2-n* **by** *simp*
thus *?thesis* **using** *a1 b1* **by** *auto*
qed
next
show *dim-col* (*kron* (λ(*l*::*nat*). |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/(*2*^*l*)) $\cdot_m$ |*one*⟩) (*map nat* (*rev* [*1*..*n*])))
= *1*
proof −
have *a2*:*dim-col* (*kron* (λ*l*. |*zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat j* / *2* ^ *l*) $\cdot_m$ |*one*⟩) (*map nat* (*rev* [*1*..*int n*])))
= ($\prod$ *x*←(*map nat* (*rev* [*1*..*int n*])). (*dim-col* ((λ*l*. |*zero*⟩ + *exp* (*2* ∗ i ∗ *complex-of-real pi* ∗ *complex-of-nat j* / *2* ^ *l*) $\cdot_m$ |*one*⟩) *x*)))
using *dim-col-kron* **by** *simp*
also have ... = ($\prod$ *x*←(*map nat* (*rev* [*1*..*int n*])). *1*) **using** *prod-rep-fun* **by** *auto*
also have ... = *1* **using** *prod-1-n* **by** *metis*
finally show *?thesis* **using** *a2* **by** *auto*
qed
qed
qed
also have ... = (*1* / *sqrt* (*2*^*n*)) $\cdot_m$ *kron* (λ*l*. |*zero*⟩ + *exp* (*2*∗i∗*pi*∗*j*/*2*^*l*) $\cdot_m$ |*one*⟩) (*map nat* [*1*..*int n*])
using *reverse-qubits-kron prod-rep-fun* **by** *presburger*
also have ... = *QFT-product-representation j n* **using** *QFT-product-representation-def* **by** *simp*
finally show *?thesis* **by** *this*
qed

Finally, we proof the correctness of the algorithm

theorem *ordered-QFT-is-correct*:
assumes *j* < *2*^*n*
shows (*ordered-QFT n*) ∗ |*state-basis n j*⟩ = *QFT-product-representation j n*
proof −
have (*ordered-QFT n*) ∗ |*state-basis n j*⟩ = (*reverse-qubits n*) ∗ (*QFT n*) ∗ |*state-basis n j*⟩
using *ordered-QFT-def* **by** *simp*
also have ... = (*reverse-qubits n*) ∗ ((*QFT n*) ∗ |*state-basis n j*⟩)
proof (*rule assoc-mult-mat*)
show *reverse-qubits n* ∈ *carrier-mat* (*2*^*n*) (*2*^*n*) **by** *simp*
next
show *QFT n* ∈ *carrier-mat* (*2*^*n*) (*2*^*n*) **by** *simp*
next
show |*state-basis n j*⟩ ∈ *carrier-mat* (*2* ^ *n*) *1* **using** *state-basis-carrier-mat* **by** *simp*
qed
also have ... = (*reverse-qubits n*) ∗ *reverse-QFT-product-representation j n*
using *QFT-is-correct assms* **by** *simp*

also have ... = *QFT-product-representation j n*
using *reverse-qubits-product-representation* **by** *simp*
finally show *?thesis* **by** *this*
qed

8 Unitarity

Although unitarity is not required to proof QFT's correctness, in this section we will prove it, i.e., QFT and ordered_QFT functions create quantum gates and QFT product representation is a quantum state.

lemma *state-basis-is-state*:
assumes $j < n$
shows *state n* $|$*state-basis n j*$\rangle$
proof
show *dim-col* $|$*state-basis n j*$\rangle$ = *1* **by** (*simp add*: *ket-vec-def*)
show *dim-row* $|$*state-basis n j*$\rangle$ = *2^n* **by** (*simp add*: *ket-vec-def state-basis-def*)
show $\|$*Matrix.col* $|$*state-basis n j*$\rangle$ *0*$\|$ = *1*
by (*metis assms ket-vec-col less-exp order-less-trans state-basis-def unit-cpx-vec-length*)
qed

lemma *R-dagger-mat*:
shows $(R\ k)^\dagger$ = *Matrix.mat 2 2* (λ(*i*,*j*). *if i≠j then 0 else* (*if i=0 then 1 else exp*(−*2*∗*pi*∗i/*2^k*)))
proof
define *m* **where** *m* = *Matrix.mat 2 2*
(λ(*i*,*j*). *if i≠j then 0 else* (*if i=0 then 1 else exp*(−*2*∗*pi*∗i/*2^k*)))
thus ⋀*i j*. *i* < *dim-row m* ⟹ *j* < *dim-col m* ⟹ *R k*† \$\$ (*i*, *j*) = *m* \$\$ (*i*, *j*)
proof −
fix *i j*
assume *i* < *dim-row m*
hence *i2*:*i* < *2* **using** *m-def* **by** *auto*
assume *j* < *dim-col m*
hence *j2*:*j* < *2* **using** *m-def* **by** *auto*
show *R k*† \$\$ (*i*, *j*) = *m* \$\$ (*i*, *j*)
proof (*rule disjE*)
show *i* = *0* ∨ *i* = *1* **using** *i2* **by** *auto*
next
assume *i0*:*i* = *0*
show *R k*† \$\$ (*i*, *j*) = *m* \$\$ (*i*, *j*)
proof (*rule disjE*)
show *j* = *0* ∨ *j* = *1* **using** *j2* **by** *auto*
next
assume *j0*:*j* = *0*
show *R k*† \$\$ (*i*, *j*) = *m* \$\$ (*i*, *j*)
proof −
have *R k*† \$\$ (*0*,*0*) = *cnj* (*R k* \$\$ (*0*,*0*))
using *dagger-def*
by (*metis* (*no-types*, *lifting*) *One-nat-def R-def Suc-1 Suc-eq-plus1*

Tensor.mat-of-cols-list-def dim-col-mat(*1*) *dim-row-mat*(*1*) *index-mat*(*1*) *list.size*(*3*)
list.size(*4*) *old.prod.case power-eq-0-iff power-zero-numeral*)
also have ... = *1*
using *R-def mat-of-cols-list-def*
by (*metis One-nat-def Suc-1 Suc-eq-plus1 complex-cnj-one-iff index-mat-of-cols-list*
list.size(*3*) *list.size*(*4*) *nth-Cons-0 pos2*)
also have ... = *m* $$ (*0*,*0*) **using** *m-def* **by** *simp*
finally show *?thesis* **using** *i0 j0* **by** *auto*
qed
next
assume *j1*:*j* = *1*
show *R k*† $$ (*i*, *j*) = *m* $$ (*i*, *j*)
proof −
have *R k*† $$ (*0*,*1*) = *cnj* (*R k* $$ (*1*,*0*))
using *dagger-def*
by (*metis* (*no-types*, *lifting*) *One-nat-def R-def Suc-1 Suc-eq-plus1*
Tensor.mat-of-cols-list-def ‹*j* < *dim-col m*› *dim-col-mat*(*1*) *dim-row-mat*(*1*)

index-mat(*1*) *j1 list.size*(*3*) *list.size*(*4*) *m-def old.prod.case pos2*)
also have ... = *0*
using *R-def mat-of-cols-list-def*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-eq-plus1* ‹*j* < *dim-col m*›
complex-cnj-zero-iff dim-col-mat(*1*) *index-mat-of-cols-list j1 list.size*(*3*)

list.size(*4*) *m-def nth-Cons-0 nth-Cons-Suc pos2*)
also have ... = *m* $$ (*0*,*1*) **using** *m-def* **by** *auto*
finally show *?thesis* **using** *i0 j1* **by** *auto*
qed
qed
next
assume *i1*:*i* = *1*
show *R k*† $$ (*i*, *j*) = *m* $$ (*i*, *j*)
proof (*rule disjE*)
show *j* = *0* ∨ *j* = *1* **using** *j2* **by** *auto*
next
assume *j0*:*j* = *0*
show *R k*† $$ (*i*, *j*) = *m* $$ (*i*, *j*)
proof −
have *R k*† $$ (*1*,*0*) = *cnj* (*R k* $$ (*0*,*1*))
using *dagger-def*
by (*metis* (*no-types*, *lifting*) *One-nat-def R-def Suc-1 Suc-eq-plus1*
Tensor.mat-of-cols-list-def dim-col-mat(*1*) *dim-row-mat*(*1*) *index-mat*(*1*)

less-Suc-numeral list.size(*3*) *list.size*(*4*) *old.prod.case power-eq-0-iff*
power-zero-numeral pred-numeral-simps(*2*))
also have ... = *0*

```
          using R-def mat-of-cols-list-def
        by (metis One-nat-def Suc-eq-plus1 complex-cnj-zero-iff index-mat-of-cols-list
              less-Suc-eq-0-disj list.size(4) nth-Cons-0 nth-Cons-Suc pos2)
        also have ... = m $$ (1,0) using m-def by simp
        finally show ?thesis using i1 j0 by simp
      qed
    next
      assume j1:j = 1
      show R k† $$ (i, j) = m $$ (i, j)
      proof –
        have R k† $$ (1,1) = cnj (R k $$ (1,1))
          using dagger-def
          by (metis (no-types, lifting) One-nat-def R-def Suc-1 Suc-eq-plus1
          Tensor.mat-of-cols-list-def dim-col-mat(1) dim-row-mat(1) index-mat(1)

              less-Suc-numeral list.size(3) list.size(4) old.prod.case power-eq-0-iff
              power-zero-numeral pred-numeral-simps(2))
        also have ... = cnj (exp(2∗pi∗i/2^k))
          using R-def mat-of-cols-list-def
            by (metis One-nat-def Suc-1 Suc-eq-plus1 index-mat-of-cols-list lessI
list.size(3)
              list.size(4) nth-Cons-0 nth-Cons-Suc)
        also have ... = exp (−2∗pi∗i/2^k)
        by (smt (verit, ccfv-threshold) exp-of-real-cnj mult.commute mult.left-commute

                      mult-1s-ring-1(1) of-real-divide of-real-minus of-real-numeral
of-real-power
              times-divide-eq-right)
        also have ... = m $$ (1,1) using m-def by simp
        finally have R k† $$ (i, j) = m $$ (i, j) using i1 j1 by simp
        thus ?thesis by this
      qed
    qed
  qed
qed
next
  define m where m = Matrix.mat 2 2
  (λ(i,j). if i≠j then 0 else (if i=0 then 1 else exp(−2∗pi∗i/2^k)))
  thus dim-row R k† = dim-row m
   by (metis (no-types, lifting) One-nat-def R-def Suc-1 Suc-eq-plus1 Tensor.mat-of-cols-list-def
       dim-col-mat(1) dim-row-mat(1) dim-row-of-dagger list.size(3) list.size(4))
next
  define m where m = Matrix.mat 2 2
  (λ(i,j). if i≠j then 0 else (if i=0 then 1 else exp(−2∗pi∗i/2^k)))
  thus dim-col R k† = dim-col m
    by (simp add: R-def Tensor.mat-of-cols-list-def)
qed

lemma R-is-gate:
```

shows *gate 1* (*R n*)
proof
show *dim-row* (*R n*) = *2*^*1* **using** *R-def* **by** (*simp add*: *Tensor.mat-of-cols-list-def*)
show *square-mat* (*R n*) **using** *R-def* **by** (*simp add*: *Tensor.mat-of-cols-list-def*)
show *unitary* (*R n*)
proof −
have $((R\ n)^{\dagger}) * (R\ n) = 1_m\ 2 \wedge (R\ n) * ((R\ n)^{\dagger}) = 1_m\ 2$
proof
show $R\ n^{\dagger} * R\ n = 1_m\ 2$
proof
show $\bigwedge i\ j.\ i < \textit{dim-row}\ (1_m\ 2) \Longrightarrow j < \textit{dim-col}\ (1_m\ 2) \Longrightarrow$
$(R\ n^{\dagger} * R\ n)\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof −
fix *i j*
assume $i < \textit{dim-row}\ (1_m\ 2)$
hence *i2*:$i < 2$ **by** *auto*
assume $j < \textit{dim-col}\ (1_m\ 2)$
hence *j2*:$j < 2$ **by** *auto*
show $(R\ n^{\dagger} * R\ n)\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof (*rule disjE*)
show $i = 0 \vee i = 1$ **using** *i2* **by** *auto*
next
assume *i0*:$i = 0$
show $(R\ n^{\dagger} * R\ n)\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof (*rule disjE*)
show $j = 0 \vee j = 1$ **using** *j2* **by** *auto*
next
assume *j0*:$j = 0$
show $(R\ n^{\dagger} * R\ n)\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof −
have $(R\ n^{\dagger} * R\ n)\ \$\$\ (0,0) = ((R\ n)^{\dagger}\ \$\$\ (0,0)) * ((R\ n)\ \$\$\ (0,0)) +$
$((R\ n)^{\dagger}\ \$\$\ (0,1)) * ((R\ n)\ \$\$\ (1,0))$
using ‹*dim-row* (*R n*) = *2* ^ *1*› ‹*square-mat* (*R n*)› *sumof2* **by**
fastforce
also have ... = *1* **using** *R-dagger-mat R-def index-mat-of-cols-list*
by (*smt* (*verit*, *del-insts*) *Suc-1 Suc-eq-plus1 add.commute add-0*
index-mat(*1*)
lessI list.size(*3*) *list.size*(*4*) *mult-1 mult-zero-left nth-Cons-0*
nth-Cons-Suc old.prod.case pos2)
also have ... = $1_m\ 2\ \$\$\ (0,0)$ **by** *simp*
finally show *?thesis* **using** *i0 j0* **by** *simp*
qed
next
assume *j1*:$j = 1$
show $(R\ n^{\dagger} * R\ n)\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof −
have $(R\ n^{\dagger} * R\ n)\ \$\$\ (0,1) = ((R\ n)^{\dagger}\ \$\$\ (0,0)) * ((R\ n)\ \$\$\ (0,1)) +$
$((R\ n)^{\dagger}\ \$\$\ (0,1)) * ((R\ n)\ \$\$\ (1,1))$
using ‹*dim-row* (*R n*) = *2* ^ *1*› ‹*square-mat* (*R n*)› *sumof2* **by**

```
fastforce
        also have ... = 0 using R-dagger-mat R-def index-mat-of-cols-list
         by (smt (verit) Suc-1 Suc-eq-plus1 add-cancel-left-left index-mat(1)
lessI
             list.size(3) list.size(4) mult-eq-0-iff nth-Cons-0 nth-Cons-Suc
old.prod.case
           pos2)
        also have ... = 1m 2 $$ (0,1) by simp
        finally show ?thesis using i0 j1 by simp
       qed
      qed
     next
      assume i1:i = 1
      show ((R n†) * R n) $$ (i, j) = 1m 2 $$ (i, j)
      proof (rule disjE)
       show j = 0 ∨ j = 1 using j2 by auto
      next
       assume j0:j = 0
       show (R n† * R n) $$ (i, j) = 1m 2 $$ (i, j)
       proof -
        have (R n† * R n) $$ (1,0) = ((R n)† $$ (1,0)) * ((R n) $$ (0,0)) +
           ((R n)† $$ (1,1)) * ((R n) $$ (1,0))
          using ‹dim-row (R n) = 2 ^ 1› ‹square-mat (R n)› sumof2 by
fastforce
        also have ... = 0 using R-dagger-mat R-def index-mat-of-cols-list
        by (smt (verit) Suc-1 Suc-eq-plus1 add-cancel-right-right index-mat(1)
lessI
             list.size(3) list.size(4) mult-eq-0-iff nth-Cons-0 nth-Cons-Suc
old.prod.case
           plus-1-eq-Suc pos2)
        also have ... = 1m 2 $$ (1,0) by simp
        finally show ?thesis using i1 j0 by simp
       qed
      next
       assume j1:j = 1
       show (R n† * R n) $$ (i, j) = 1m 2 $$ (i, j)
       proof -
        have (R n† * R n) $$ (1,1) = ((R n)† $$ (1,0)) * ((R n) $$ (0,1)) +
           ((R n)† $$ (1,1)) * ((R n) $$ (1,1))
          using ‹dim-row (R n) = 2 ^ 1› ‹square-mat (R n)› sumof2 by
fastforce
        also have ... = exp(−2*pi*i/2^n) * exp(2*pi*i/2^n)
         using R-dagger-mat R-def index-mat-of-cols-list by auto
        also have ... = 1
         by (metis (no-types, lifting) exp-minus-inverse minus-divide-divide
           minus-divide-right mult-minus-left of-real-minus)
        also have ... = 1m 2 $$ (1,1) by simp
        finally show ?thesis using i1 j1 by simp
       qed
```

qed
qed
qed
next
show *dim-row* $(R\ n^\dagger * R\ n) = $ *dim-row* $(1_m\ 2)$
using ‹*dim-row* $(R\ n) = 2\ \hat{}\ 1$› ‹*square-mat* $(R\ n)$› **by** *auto*
next
show *dim-col* $(R\ n^\dagger * R\ n) = $ *dim-col* $(1_m\ 2)$
using ‹*dim-row* $(R\ n) = 2\ \hat{}\ 1$› ‹*square-mat* $(R\ n)$› **by** *auto*
qed
next
show $R\ n * ((R\ n)^\dagger) = 1_m\ 2$
proof
show $\bigwedge i\ j.\ i <$ *dim-row* $(1_m\ 2) \Longrightarrow j <$ *dim-col* $(1_m\ 2) \Longrightarrow$
$(R\ n * (R\ n^\dagger))\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof −
fix $i\ j$
assume $i <$ *dim-row* $(1_m\ 2)$
hence $i2{:}i < 2$ **by** *auto*
assume $j <$ *dim-col* $(1_m\ 2)$
hence $j2{:}j < 2$ **by** *auto*
show $(R\ n * (R\ n^\dagger))\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof (*rule disjE*)
show $i = 0 \lor i = 1$ **using** *i2* **by** *auto*
next
assume $i0{:}i = 0$
show $(R\ n * (R\ n^\dagger))\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof (*rule disjE*)
show $j = 0 \lor j = 1$ **using** *j2* **by** *auto*
next
assume $j0{:}j = 0$
show $(R\ n * (R\ n^\dagger))\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof −
have $(R\ n * (R\ n^\dagger))\ \$\$\ (0,0) = ((R\ n)\ \$\$\ (0,0)) * ((R\ n)^\dagger\ \$\$\ (0,0))$
$+$
$((R\ n)\ \$\$\ (0,1)) * ((R\ n)^\dagger\ \$\$\ (1,0))$
using ‹*dim-row* $(R\ n) = 2\ \hat{}\ 1$› ‹*square-mat* $(R\ n)$› *sumof2* **by**
fastforce
also have $\ldots = 1$ **using** *R-dagger-mat R-def index-mat-of-cols-list*
by *simp*
also have $\ldots = 1_m\ 2\ \$\$\ (0,0)$ **by** *simp*
finally show *?thesis* **using** *i0 j0* **by** *simp*
qed
next
assume $j1{:}j = 1$
show $(R\ n * (R\ n^\dagger))\ \$\$\ (i, j) = 1_m\ 2\ \$\$\ (i, j)$
proof −
have $(R\ n * (R\ n^\dagger))\ \$\$\ (0,1) = ((R\ n)\ \$\$\ (0,0)) * ((R\ n)^\dagger\ \$\$\ (0,1))$
$+$

```
              ((R n) $$ (0,1)) * ((R n)† $$ (1,1))
             using ‹dim-row (R n) = 2 ^ 1› ‹square-mat (R n)› sumof2 by
fastforce
           also have ... = 0 using R-dagger-mat R-def index-mat-of-cols-list
by simp
          also have ... = 1m 2 $$ (0,1) by simp
          finally show ?thesis using i0 j1 by simp
        qed
      qed
    next
      assume i1:i = 1
      show (R n * (R n†)) $$ (i, j) = 1m 2 $$ (i, j)
      proof (rule disjE)
        show j = 0 ∨ j = 1 using j2 by auto
      next
        assume j0:j = 0
        show (R n * (R n†)) $$ (i, j) = 1m 2 $$ (i, j)
        proof -
         have (R n * (R n†)) $$ (1,0) = ((R n) $$ (1,0)) * ((R n)† $$ (0,0))
+
              ((R n) $$ (1,1)) * ((R n)† $$ (1,0))
             using ‹dim-row (R n) = 2 ^ 1› ‹square-mat (R n)› sumof2 by
fastforce
          also have ... = 1m 2 $$ (1,0)
            using R-dagger-mat R-def index-mat-of-cols-list by simp
          finally show ?thesis using i1 j0 by simp
        qed
      next
        assume j1:j = 1
        show (R n * (R n†)) $$ (i, j) = 1m 2 $$ (i, j)
        proof -
         have (R n * (R n†)) $$ (1,1) = ((R n) $$ (1,0)) * ((R n)† $$ (0,1))
+
              ((R n) $$ (1,1)) * ((R n)† $$ (1,1))
             using ‹dim-row (R n) = 2 ^ 1› ‹square-mat (R n)› sumof2 by
fastforce
          also have ... = exp(2*pi*i/2^n) * exp(-2*pi*i/2^n)
            using R-dagger-mat R-def index-mat-of-cols-list by simp
          also have ... = 1
            by (simp add: exp-minus-inverse)
          also have ... = 1m 2 $$ (1,1) by simp
          finally show ?thesis using i1 j1 by simp
        qed
      qed
    qed
  qed
next
  show dim-row (R n * (R n†)) = dim-row (1m 2)
    by (simp add: ‹dim-row (R n) = 2 ^ 1›)
```

next
show *dim-col* $(R\ n * (R\ n^\dagger)) = $ *dim-col* $(1_m\ 2)$
by (*simp add*: ‹*dim-row* $(R\ n) = 2\ \hat{}\ 1$›)
qed
qed
thus *?thesis* **using** *unitary-def R-def mat-of-cols-list-def* **by** *auto*
qed
qed

lemma *SWAP-dagger-mat*:
shows $SWAP^\dagger = SWAP$
proof −
have $SWAP^\dagger = $ *Matrix.mat 4 4* (λ(*i*,*j*). *cnj* (*SWAP* $$ (*j*,*i*)))
using *dagger-def SWAP-carrier-mat*
by (*metis SWAP-ncols carrier-matD*(*1*))
also have ... = *Matrix.mat 4 4* (λ(*i*,*j*). *cnj* (*SWAP* $$ (*i*,*j*)))
using *SWAP-def SWAP-index*
proof −
obtain *nn* :: (*nat* × *nat* ⇒ *complex*) ⇒ (*nat* × *nat* ⇒ *complex*) ⇒ *nat* ⇒ *nat* ⇒ *nat* **and** *nna* :: (*nat* × *nat* ⇒ *complex*) ⇒ (*nat* × *nat* ⇒ *complex*) ⇒ *nat* ⇒ *nat* ⇒ *nat* **where**
∀ *x0 x1 x3 x5*. (∃ *v6 v7*. (*v6* < *x5* ∧ *v7* < *x3*) ∧ *x1* (*v6*, *v7*) ≠ *x0* (*v6*, *v7*)) = ((*nn x0 x1 x3 x5* < *x5* ∧ *nna x0 x1 x3 x5* < *x3*) ∧ *x1* (*nn x0 x1 x3 x5*, *nna x0 x1 x3 x5*) ≠ *x0* (*nn x0 x1 x3 x5*, *nna x0 x1 x3 x5*))
by *moura*
then have ∀ *n na nb nc f fa*. (*n* ≠ *na* ∨ *nb* ≠ *nc* ∨ (*nn fa f nb n* < *n* ∧ *nna fa f nb n* < *nb*) ∧ *f* (*nn fa f nb n*, *nna fa f nb n*) ≠ *fa* (*nn fa f nb n*, *nna fa f nb n*)) ∨ *Matrix.mat n nb f* = *Matrix.mat na nc fa*
by (*meson cong-mat*)
moreover
{ **assume** *nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* ≠ *3* ∨ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* ≠ *3*
then have (*if nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* ≠ *2* ∨ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* ≠ *1 then if nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* ≠ *3* ∨ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* ≠ *3 then* (*if nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *0* ∧ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *0 then 1*::*complex else if nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *1* ∧ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *2 then 1 else if nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *2* ∧ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *1 then 1 else if nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *3* ∧ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *3 then 1 else 0*) = *0 else* (*if nn* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*, *n*))) *4 4* = *0* ∧ *nna* (λ(*na*, *n*). *cnj* (*SWAP* $$ (*n*, *na*))) (λ(*na*, *n*). *cnj* (*SWAP* $$ (*na*,

n))) 4 4 = 0 then 1::complex else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1 else (if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1) ⟶ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∨ (if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 0

by *presburger* **}**

moreover

{ assume *nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3*

then have *(if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 ≠ 2 ∨ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 ≠ 1 then if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 ≠ 3 ∨ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 ≠ 3 then (if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 0 else (if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0*

∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1 else (if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1) ⟶ (if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1

by *presburger* **}**

moreover

{ assume (if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 0 ∧ (if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 0

moreover

{ **assume** ((if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 then 1::complex else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 then 1 else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 then 1 else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 then 1 else 0) = 0 ∧ (if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 then 1::complex else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 then 1 else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 then 1 else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 then 1 else 0) = 0) ∧ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ cnj (SWAP $$ (n, na))) ≠ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ cnj (SWAP $$ (na, n)))

then have Matrix.mat 4 4 (λ(n, na). if n = 0 ∧ na = 0 then 1::complex else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) $$ (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) = (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) ⟶ ((if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 then 1::complex else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 then 1 else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 then 1 else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 then 1 else 0) = 0 ∧ (if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 then 1::complex else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 then 1 else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 then 1 else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj

(SWAP $$ (n, na))) 4 4 = 3 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 then 1 else 0) = 0) ∧ SWAP $$ (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) ≠ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0)

by (smt (z3) SWAP-def old.prod.case)

then have Matrix.mat 4 4 (λ(n, na). if n = 0 ∧ na = 0 then 1::complex else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) $$ (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) ≠ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) ∨ SWAP $$ (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) ≠ (case (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0)

by fastforce **}**

ultimately have SWAP $$ (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) = (case (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) ∧ Matrix.mat 4 4 (λ(n, na). if n = 0 ∧ na = 0 then 1::complex else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) $$ (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) = (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) ⟶ ¬ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 < 4 ∨ ¬ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 < 4 ∨ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ cnj (SWAP $$ (n, na))) = (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ cnj (SWAP $$ (na, n)))

by blast **}**

moreover

{ assume (if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$

(na, n))) 4 4 = 0 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1 ∧ (if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1

moreover

{ assume ((if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 then 1::complex else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 then 1 else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 then 1 else if nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 ∧ nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 then 1 else 0) = 1 ∧ (if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 0 then 1::complex else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 then 1 else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 2 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 1 then 1 else if nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 ∧ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4 = 3 then 1 else 0) = 1) ∧ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ cnj (SWAP $$ (n, na))) ≠ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ cnj (SWAP $$ (na, n)))

then have ((if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1 ∧ (if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 0 then 1::complex else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 2 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 1 then 1 else if nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 ∧ nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4 = 3 then 1 else 0) = 1) ∧ SWAP $$ (nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4, nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4) ≠ SWAP $$ (nn (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4, nna (λ(na, n). cnj (SWAP $$ (n, na))) (λ(na, n). cnj (SWAP $$ (na, n))) 4 4)

by (*smt* (*z3*) *old.prod.case*)

then have *Matrix.mat 4 4* (λ(n, na). *if* n = 0 ∧ na = 0 *then* 1::complex *else if* n = 1 ∧ na = 2 *then* 1 *else if* n = 2 ∧ na = 1 *then* 1 *else if* n = 3 ∧ na = 3 *then* 1 *else* 0) $$ (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) ≠ (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) ∨ SWAP $$ (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) ≠ (case (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0)

using *SWAP-def* **by** *auto* **}**

ultimately have SWAP $$ (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) = (case (nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) ∧ Matrix.mat 4 4 (λ(n, na). if n = 0 ∧ na = 0 then 1::complex else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) $$ (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) = (case (nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4, nna (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na))) 4 4) of (n, na) ⇒ if n = 0 ∧ na = 0 then 1 else if n = 1 ∧ na = 2 then 1 else if n = 2 ∧ na = 1 then 1 else if n = 3 ∧ na = 3 then 1 else 0) ⟶ ¬ nn (λ(n, na). cnj (SWAP $$ (na, n))) (λ(n, na). cnj (SWAP $$ (n, na)))

4 4 < *4* ∨ ¬ *nna* (λ(*n*, *na*). *cnj* (*SWAP* $$ (*na*, *n*))) (λ(*n*, *na*). *cnj* (*SWAP* $$ (*n*, *na*))) *4 4* < *4* ∨ (*case* (*nn* (λ(*n*, *na*). *cnj* (*SWAP* $$ (*na*, *n*))) (λ(*n*, *na*). *cnj* (*SWAP* $$ (*n*, *na*))) *4 4*, *nna* (λ(*n*, *na*). *cnj* (*SWAP* $$ (*na*, *n*))) (λ(*n*, *na*). *cnj* (*SWAP* $$ (*n*, *na*))) *4 4*) *of* (*n*, *na*) ⇒ *cnj* (*SWAP* $$ (*n*, *na*))) = (*case* (*nn* (λ(*n*, *na*). *cnj* (*SWAP* $$ (*na*, *n*))) (λ(*n*, *na*). *cnj* (*SWAP* $$ (*n*, *na*))) *4 4*, *nna* (λ(*n*, *na*). *cnj* (*SWAP* $$ (*na*, *n*))) (λ(*n*, *na*). *cnj* (*SWAP* $$ (*n*, *na*))) *4 4*) *of* (*n*, *na*) ⇒ *cnj* (*SWAP* $$ (*na*, *n*)))
by *linarith* **}**
ultimately show *?thesis*
by (*smt* (*z3*) *SWAP-def index-mat*(*1*))
qed
also have . . . = *SWAP* **using** *SWAP-def SWAP-index*
by (*smt* (*verit*, *ccfv-SIG*) *case-prod-conv complex-cnj-one complex-cnj-zero cong-mat index-mat*(*1*))
finally show *?thesis* **by** *this*
qed

lemma *SWAP-inv*:
shows *SWAP* ∗ (*SWAP*†) = *1*$_m$ *4*
apply (*simp add*: *SWAP-def times-mat-def one-mat-def*)
apply (*rule cong-mat*)
by (*auto simp*: *scalar-prod-def complex-eqI*)

lemma *SWAP-inv′*:
shows (*SWAP*†) ∗ *SWAP* = *1*$_m$ *4*
apply (*simp add*: *SWAP-def times-mat-def one-mat-def*)
apply (*rule cong-mat*)
by (*auto simp*: *scalar-prod-def complex-eqI*)

lemma *SWAP-is-gate*:
shows *gate 2 SWAP*
proof
show *dim-row SWAP* = 2^2 **using** *SWAP-carrier-mat* **by** (*simp add*: *numeral-Bit0*)
next
show *square-mat SWAP* **using** *SWAP-carrier-mat* **by** (*simp add*: *numeral-Bit0*)
next
show *unitary SWAP*
using *unitary-def SWAP-inv SWAP-inv′ SWAP-ncols SWAP-nrows* **by** *presburger*
qed

lemma *control2-inv*:
assumes *gate 1 U*
shows (*control2 U*) ∗ ((*control2 U*)†) = *1*$_m$ *4*
proof
show ⋀*i j*. *i* < *dim-row* (*1*$_m$ *4*) ⟹ *j* < *dim-col* (*1*$_m$ *4*) ⟹
(*control2 U* ∗ ((*control2 U*)†)) $$ (*i*, *j*) = *1*$_m$ *4* $$ (*i*, *j*)
proof −

```
  fix i j
  assume i < dim-row (1m 4)
  hence i4:i < 4 by auto
  assume j < dim-col (1m 4)
  hence j4:j < 4 by auto
  show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
  proof (rule disjE)
    show i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3 using i4 by auto
  next
    assume i0:i = 0
    show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
    proof (rule disjE)
      show j = 0 ∨ j = 1 ∨ j = 2 ∨ j = 3 using j4 by auto
    next
      assume j0:j = 0
      show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
      proof −
        have (control2 U * ((control2 U)†)) $$ (0,0) =
              (control2 U) $$ (0,0) * ((control2 U)†) $$ (0,0) +
              (control2 U) $$ (0,1) * ((control2 U)†) $$ (1,0) +
              (control2 U) $$ (0,2) * ((control2 U)†) $$ (2,0) +
              (control2 U) $$ (0,3) * ((control2 U)†) $$ (3,0)
          using times-mat-def sumof4
            by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dagger-def
              dim-col-of-dagger dim-row-mat(1) i0 i4 index-matrix-prod)
        also have ... = ((control2 U)†) $$ (0,0)
          using control2-def index-mat-of-cols-list by force
        also have ... = cnj ((control2 U) $$ (0,0))
          using dagger-def
           by (metis carrier-matD(1) carrier-matD(2) control2-carrier-mat i0 i4
index-mat(1)
              old.prod.case)
        also have ... = 1 using control2-def index-mat-of-cols-list by auto
        also have ... = 1m 4 $$ (0,0) by simp
        finally show ?thesis using i0 j0 by simp
      qed
    next
      assume jl3:j = 1 ∨ j = 2 ∨ j = 3
      show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
      proof (rule disjE)
        show j = 1 ∨ j = 2 ∨ j = 3 using jl3 by this
      next
        assume j1:j = 1
        show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
        proof −
          have (control2 U * ((control2 U)†)) $$ (0,1) =
                (control2 U) $$ (0,0) * ((control2 U)†) $$ (0,1) +
                (control2 U) $$ (0,1) * ((control2 U)†) $$ (1,1) +
```

```
        (control2 U) $$ (0,2) * ((control2 U)†) $$ (2,1) +
        (control2 U) $$ (0,3) * ((control2 U)†) $$ (3,1)
      using times-mat-def sumof4
       by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
          dim-row-of-dagger i0 i4 index-matrix-prod j1 j4)
    also have ... = ((control2 U)†) $$ (0,1)
      using control2-def index-mat-of-cols-list by force
    also have ... = cnj ((control2 U) $$ (1,0))
      using dagger-def
        by (metis (mono-tags, lifting) One-nat-def Suc-1 add-Suc-right car-
rier-matD(1)
        carrier-matD(2) control2-carrier-mat index-mat(1) less-Suc-eq-0-disj
numeral-Bit0
         prod.simps(2))
    also have ... = 0 using control2-def index-mat-of-cols-list by auto
    also have ... = 1_m 4 $$ (0,1) by simp
    finally show ?thesis using i0 j1 by simp
  qed
next
  assume jl2:j = 2 ∨ j = 3
  show (control2 U * ((control2 U)†)) $$ (i, j) = 1_m 4 $$ (i, j)
  proof (rule disjE)
    show j = 2 ∨ j = 3 using jl2 by this
  next
    assume j2:j = 2
    show (control2 U * ((control2 U)†)) $$ (i, j) = 1_m 4 $$ (i, j)
    proof -
      have (control2 U * ((control2 U)†)) $$ (0,2) =
          (control2 U) $$ (0,0) * ((control2 U)†) $$ (0,2) +
          (control2 U) $$ (0,1) * ((control2 U)†) $$ (1,2) +
          (control2 U) $$ (0,2) * ((control2 U)†) $$ (2,2) +
          (control2 U) $$ (0,3) * ((control2 U)†) $$ (3,2)
        using times-mat-def sumof4
         by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
            dim-row-of-dagger i0 i4 index-matrix-prod j2 j4)
      also have ... = ((control2 U)†) $$ (0,2)
        using control2-def index-mat-of-cols-list by force
      also have ... = cnj ((control2 U) $$ (2,0))
        using dagger-def
          by (smt (verit, del-insts) carrier-matD(1) carrier-matD(2) con-
trol2-carrier-mat
            index-mat(1) less-add-same-cancel2 numeral-Bit0 prod.simps(2)
zero-less-numeral)
      also have ... = 0 using control2-def index-mat-of-cols-list by auto
      also have ... = 1_m 4 $$ (0,2) by simp
      finally show ?thesis using i0 j2 by simp
    qed
```

next
assume *j3*:$j = 3$
show $(control2\ U * ((control2\ U)^\dagger))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof −
have $(control2\ U * ((control2\ U)^\dagger))\ \$\$\ (0,3) =$
$(control2\ U)\ \$\$\ (0,0) * ((control2\ U)^\dagger)\ \$\$\ (0,3) +$
$(control2\ U)\ \$\$\ (0,1) * ((control2\ U)^\dagger)\ \$\$\ (1,3) +$
$(control2\ U)\ \$\$\ (0,2) * ((control2\ U)^\dagger)\ \$\$\ (2,3) +$
$(control2\ U)\ \$\$\ (0,3) * ((control2\ U)^\dagger)\ \$\$\ (3,3)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger
dim-row-of-dagger i0 i4 index-matrix-prod j3 j4)
also have ... $= ((control2\ U)^\dagger)\ \$\$\ (0,3)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have ... $= cnj\ ((control2\ U)\ \$\$\ (3,0))$
using *dagger-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
index-mat(*1*) *j3 j4*
prod.simps(*2*) *zero-less-numeral*)
also have ... $= 0$ **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have ... $= 1_m\ 4\ \$\$\ (0,3)$ **by** *simp*
finally show *?thesis* **using** *i0 j3* **by** *simp*
qed
qed
qed
qed
next
assume *il3*:$i = 1 \vee i = 2 \vee i = 3$
show $(control2\ U * ((control2\ U)^\dagger))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof (*rule disjE*)
show $i = 1 \vee i = 2 \vee i = 3$ **using** *il3* **by** *this*
next
assume *i1*:$i = 1$
show $(control2\ U * ((control2\ U)^\dagger))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof (*rule disjE*)
show *jl4*:$j = 0 \vee j = 1 \vee j = 2 \vee j = 3$ **using** *j4* **by** *auto*
next
assume *j0*:$j = 0$
show $(control2\ U * ((control2\ U)^\dagger))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof −
have $(control2\ U * ((control2\ U)^\dagger))\ \$\$\ (1,0) =$
$(control2\ U)\ \$\$\ (1,0) * ((control2\ U)^\dagger)\ \$\$\ (0,0) +$
$(control2\ U)\ \$\$\ (1,1) * ((control2\ U)^\dagger)\ \$\$\ (1,0) +$
$(control2\ U)\ \$\$\ (1,2) * ((control2\ U)^\dagger)\ \$\$\ (2,0) +$
$(control2\ U)\ \$\$\ (1,3) * ((control2\ U)^\dagger)\ \$\$\ (3,0)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger

```
                dim-row-of-dagger i1 i4 index-matrix-prod j0 j4)
          also have ... = (control2 U) $$ (1,1) * ((control2 U)†) $$ (1,0) +
                          (control2 U) $$ (1,3) * ((control2 U)†) $$ (3,0)
            using control2-def index-mat-of-cols-list by force
          also have ... = (control2 U) $$ (1,1) * (cnj ((control2 U) $$ (0,1))) +
                          (control2 U) $$ (1,3) * (cnj ((control2 U) $$ (0,3)))
            using dagger-def
              by (smt (verit, ccfv-threshold) One-nat-def Suc-1 add.commute
add-Suc-right
                  carrier-matD(1) carrier-matD(2) control2-carrier-mat i1 i4
index-mat(1) j0 j4
                lessI numeral-3-eq-3 numeral-Bit0 plus-1-eq-Suc prod.simps(2))
          also have ... = (control2 U) $$ (1,1) * (cnj 0) +
                          (control2 U) $$ (1,3) * (cnj 0)
            using control2-def index-mat-of-cols-list by simp
          also have ... = 0 by auto
          also have ... = 1m 4 $$ (1,0) by simp
          finally show ?thesis using i1 j0 by simp
        qed
      next
        assume jl3:j = 1 ∨ j = 2 ∨ j = 3
        show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
        proof (rule disjE)
          show j = 1 ∨ j = 2 ∨ j = 3 using jl3 by this
        next
          assume j1:j = 1
          show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
          proof -
            have (control2 U * ((control2 U)†)) $$ (1,1) =
                  (control2 U) $$ (1,0) * ((control2 U)†) $$ (0,1) +
                  (control2 U) $$ (1,1) * ((control2 U)†) $$ (1,1) +
                  (control2 U) $$ (1,2) * ((control2 U)†) $$ (2,1) +
                  (control2 U) $$ (1,3) * ((control2 U)†) $$ (3,1)
              using times-mat-def sumof4
              by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
                  dim-row-of-dagger i1 i4 index-matrix-prod j1 j4)
            also have ... = (control2 U) $$ (1,1) * ((control2 U)†) $$ (1,1) +
                            (control2 U) $$ (1,3) * ((control2 U)†) $$ (3,1)
              using control2-def index-mat-of-cols-list by force
            also have ... = (control2 U) $$ (1,1) * (cnj ((control2 U) $$ (1,1)))
+
                            (control2 U) $$ (1,3) * (cnj ((control2 U) $$ (1,3)))
              using dagger-def
              by (smt (verit, best) One-nat-def Suc-1 add.commute add-Suc-right
carrier-matD(1)
                  carrier-matD(2) control2-carrier-mat i1 i4 index-mat(1) lessI
numeral-3-eq-3
                numeral-Bit0 plus-1-eq-Suc prod.simps(2))
```

```
        also have ... = U $$ (0,0) * (cnj (U $$ (0,0))) +
                U $$ (0,1) * (cnj (U $$ (0,1)))
          using control2-def index-mat-of-cols-list by simp
        also have ... = (U $$ (0,0)) * ((U†) $$ (0,0)) +
                (U $$ (0,1)) * ((U†) $$ (1,0))
          using dagger-def assms(1) gate-def by force
        also have ... = (U * (U†)) $$ (0,0)
          using times-mat-def assms(1) gate-carrier-mat sumof2
      by (smt (z3) carrier-matD(2) dagger-def dim-col-mat(1) dim-row-of-dagger

            gate.dim-row index-matrix-prod pos2 power-one-right)
        also have ... = (1m 2) $$ (0,0) using assms(1) gate-def unitary-def
by auto
        also have ... = 1 by auto
        also have ... = 1m 4 $$ (1,1) by simp
        finally show ?thesis using i1 j1 by simp
      qed
    next
      assume jl2:j = 2 ∨ j = 3
      show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
      proof (rule disjE)
        show j = 2 ∨ j = 3 using jl2 by this
      next
        assume j2:j = 2
        show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
        proof −
          have (control2 U * ((control2 U)†)) $$ (1,2) =
              (control2 U) $$ (1,0) * ((control2 U)†) $$ (0,2) +
              (control2 U) $$ (1,1) * ((control2 U)†) $$ (1,2) +
              (control2 U) $$ (1,2) * ((control2 U)†) $$ (2,2) +
              (control2 U) $$ (1,3) * ((control2 U)†) $$ (3,2)
          using times-mat-def sumof4
           by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
               dim-row-of-dagger i1 i4 index-matrix-prod j2 j4)
        also have ... = (control2 U) $$ (1,1) * ((control2 U)†) $$ (1,2) +
                (control2 U) $$ (1,3) * ((control2 U)†) $$ (3,2)
            using control2-def index-mat-of-cols-list by force
        also have ... = (control2 U) $$ (1,1) * (cnj ((control2 U) $$ (2,1)))
+
                (control2 U) $$ (1,3) * (cnj ((control2 U) $$ (2,3)))
            using dagger-def
              by (smt (verit, ccfv-threshold) One-nat-def Suc-1 add.commute
add-Suc-right
                carrier-matD(1) carrier-matD(2) control2-carrier-mat i1 i4
index-mat(1) j2 j4
              lessI numeral-3-eq-3 numeral-Bit0 plus-1-eq-Suc prod.simps(2))
        also have ... = (control2 U) $$ (1,1) * (cnj 0) +
                (control2 U) $$ (1,3) * (cnj 0)
```

```
          using control2-def index-mat-of-cols-list by simp
        also have ... = 0 by auto
        also have ... = 1m 4 $$ (1,2) by simp
        finally show ?thesis using i1 j2 by simp
      qed
    next
      assume j3:j = 3
      show (control2 U * ((control2 U)†)) $$ (i, j) = 1m 4 $$ (i, j)
      proof -
        have (control2 U * ((control2 U)†)) $$ (1,3) =
              (control2 U) $$ (1,0) * ((control2 U)†) $$ (0,3) +
              (control2 U) $$ (1,1) * ((control2 U)†) $$ (1,3) +
              (control2 U) $$ (1,2) * ((control2 U)†) $$ (2,3) +
              (control2 U) $$ (1,3) * ((control2 U)†) $$ (3,3)
          using times-mat-def sumof4
          by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
                dim-row-of-dagger i1 i4 index-matrix-prod j3 j4)
        also have ... = (control2 U) $$ (1,1) * ((control2 U)†) $$ (1,3) +
                    (control2 U) $$ (1,3) * ((control2 U)†) $$ (3,3)
          using control2-def index-mat-of-cols-list by force
        also have ... = (control2 U) $$ (1,1) * (cnj ((control2 U) $$ (3,1)))
+
                    (control2 U) $$ (1,3) * (cnj ((control2 U) $$ (3,3)))
          using dagger-def
          by (smt (verit, best) One-nat-def Suc-1 add.commute add-Suc-right
carrier-matD(1)
                carrier-matD(2) control2-carrier-mat i1 i4 index-mat(1) lessI
numeral-3-eq-3
              numeral-Bit0 plus-1-eq-Suc prod.simps(2))
        also have ... = U $$ (0,0) * (cnj (U $$ (1,0))) +
                    U $$ (0,1) * (cnj (U $$ (1,1)))
          using control2-def index-mat-of-cols-list by simp
        also have ... = (U $$ (0,0)) * ((U†) $$ (0,1)) +
                    (U $$ (0,1)) * ((U†) $$ (1,1))
          using dagger-def assms(1) gate-def by force
        also have ... = (U * (U†)) $$ (0,1)
          using times-mat-def assms(1) gate-carrier-mat sumof2
            by (smt (z3) Suc-1 carrier-matD(2) dagger-def dim-col-mat(1)
dim-row-of-dagger
              gate.dim-row index-matrix-prod lessI pos2 power-one-right)
        also have ... = (1m 2) $$ (0,1) using assms(1) gate-def unitary-def
by auto
        also have ... = 0 by auto
        also have ... = 1m 4 $$ (1,3) by simp
        finally show ?thesis using i1 j3 by simp
      qed
    qed
  qed
```

qed
next
assume *il2*:$i = 2 \vee i = 3$
show $(control2\ U * ((control2\ U)^{\dagger}))$ $$ $(i, j) = 1_m\ 4$ $$ (i, j)
proof (*rule disjE*)
show $i = 2 \vee i = 3$ **using** *il2* **by** *this*
next
assume *i2*:$i = 2$
show $(control2\ U * ((control2\ U)^{\dagger}))$ $$ $(i, j) = 1_m\ 4$ $$ (i, j)
proof (*rule disjE*)
show $j = 0 \vee j = 1 \vee j = 2 \vee j = 3$ **using** *j4* **by** *auto*
next
assume *j0*:$j = 0$
show $(control2\ U * ((control2\ U)^{\dagger}))$ $$ $(i, j) = 1_m\ 4$ $$ (i, j)
proof −
have $(control2\ U * ((control2\ U)^{\dagger}))$ $$ $(2,0) =$
$(control2\ U)$ $$ $(2,0) * ((control2\ U)^{\dagger})$ $$ $(0,0) +$
$(control2\ U)$ $$ $(2,1) * ((control2\ U)^{\dagger})$ $$ $(1,0) +$
$(control2\ U)$ $$ $(2,2) * ((control2\ U)^{\dagger})$ $$ $(2,0) +$
$(control2\ U)$ $$ $(2,3) * ((control2\ U)^{\dagger})$ $$ $(3,0)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat dim-col-of-dagger*
dim-row-of-dagger i2 i4 index-matrix-prod j0 j4)
also have $\ldots = ((control2\ U)^{\dagger})$ $$ $(2,0)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have $\ldots = cnj\ ((control2\ U)$ $$ $(0,2))$
using *dagger-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat i2 i4 index-mat*(*1*)
j0 j4 prod.simps(*2*))
also have $\ldots = 0$ **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have $\ldots = 1_m\ 4$ $$ $(2,0)$ **by** *simp*
finally show *?thesis* **using** *i2 j0* **by** *simp*
qed
next
assume *jl3*:$j = 1 \vee j = 2 \vee j = 3$
show $(control2\ U * ((control2\ U)^{\dagger}))$ $$ $(i, j) = 1_m\ 4$ $$ (i, j)
proof (*rule disjE*)
show $j = 1 \vee j = 2 \vee j = 3$ **using** *jl3* **by** *this*
next
assume *j1*:$j = 1$
show $(control2\ U * ((control2\ U)^{\dagger}))$ $$ $(i, j) = 1_m\ 4$ $$ (i, j)
proof −
have $(control2\ U * ((control2\ U)^{\dagger}))$ $$ $(2,1) =$
$(control2\ U)$ $$ $(2,0) * ((control2\ U)^{\dagger})$ $$ $(0,1) +$
$(control2\ U)$ $$ $(2,1) * ((control2\ U)^{\dagger})$ $$ $(1,1) +$
$(control2\ U)$ $$ $(2,2) * ((control2\ U)^{\dagger})$ $$ $(2,1) +$
$(control2\ U)$ $$ $(2,3) * ((control2\ U)^{\dagger})$ $$ $(3,1)$

using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat dim-col-of-dagger*
dim-row-of-dagger i2 i4 index-matrix-prod j1 j4)
also have ... = ((*control2 U*)†) $$ (*2*,*1*)
using *control2-def index-mat-of-cols-list* **by** *force*
also have ... = *cnj* ((*control2 U*) $$ (*1*,*2*))
using *dagger-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat i2 i4 index-mat*(*1*)
j1 j4 prod.simps(*2*))
also have ... = *0* **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have ... = 1_m *4* $$ (*2*,*1*) **by** *simp*
finally show *?thesis* **using** *i2 j1* **by** *simp*
qed
next
assume *jl2*:*j* = *2* ∨ *j* = *3*
show (*control2 U* ∗ ((*control2 U*)†)) $$ (*i*, *j*) = 1_m *4* $$ (*i*, *j*)
proof (*rule disjE*)
show *j* = *2* ∨ *j* = *3* **using** *jl2* **by** *this*
next
assume *j2*:*j* = *2*
show (*control2 U* ∗ ((*control2 U*)†)) $$ (*i*, *j*) = 1_m *4* $$ (*i*, *j*)
proof −
have (*control2 U* ∗ ((*control2 U*)†)) $$ (*2*,*2*) =
(*control2 U*) $$ (*2*,*0*) ∗ ((*control2 U*)†) $$ (*0*,*2*) +
(*control2 U*) $$ (*2*,*1*) ∗ ((*control2 U*)†) $$ (*1*,*2*) +
(*control2 U*) $$ (*2*,*2*) ∗ ((*control2 U*)†) $$ (*2*,*2*) +
(*control2 U*) $$ (*2*,*3*) ∗ ((*control2 U*)†) $$ (*3*,*2*)
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat dim-col-of-dagger*
dim-row-of-dagger i2 i4 index-matrix-prod j2 j4)
also have ... = ((*control2 U*)†) $$ (*2*,*2*)
using *control2-def index-mat-of-cols-list* **by** *force*
also have ... = *cnj* ((*control2 U*) $$ (*2*,*2*))
using *dagger-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat i2 index-mat*(*1*)
j2 j4 prod.simps(*2*))
also have ... = *1* **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have ... = 1_m *4* $$ (*2*,*2*) **by** *simp*
finally show *?thesis* **using** *i2 j2* **by** *simp*
qed
next
assume *j3*:*j* = *3*
show (*control2 U* ∗ ((*control2 U*)†)) $$ (*i*, *j*) = 1_m *4* $$ (*i*, *j*)
proof −
have (*control2 U* ∗ ((*control2 U*)†)) $$ (*2*,*3*) =

$(control2\ U)$ \$\$ $(2,0) * ((control2\ U)^\dagger)$ \$\$ $(0,3)$ +
$(control2\ U)$ \$\$ $(2,1) * ((control2\ U)^\dagger)$ \$\$ $(1,3)$ +
$(control2\ U)$ \$\$ $(2,2) * ((control2\ U)^\dagger)$ \$\$ $(2,3)$ +
$(control2\ U)$ \$\$ $(2,3) * ((control2\ U)^\dagger)$ \$\$ $(3,3)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger
dim-row-of-dagger i2 i4 index-matrix-prod j3 j4)
also have ... = $((control2\ U)^\dagger)$ \$\$ $(2,3)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have ... = *cnj* ((*control2 U*) \$\$ (*3*,*2*))
using *dagger-def*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat i2*
i4 index-mat(*1*)
j3 j4 prod.simps(*2*))
also have ... = *0* **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have ... = $1_m\ 4$ \$\$ $(2,3)$ **by** *simp*
finally show *?thesis* **using** *i2 j3* **by** *simp*
qed
qed
qed
qed
next
assume *i3*:*i* = *3*
show $(control2\ U * ((control2\ U)^\dagger))$ \$\$ $(i, j) = 1_m\ 4$ \$\$ (i, j)
proof (*rule disjE*)
show $j = 0 \lor j = 1 \lor j = 2 \lor j = 3$ **using** *j4* **by** *auto*
next
assume *j0*:*j* = *0*
show $(control2\ U * ((control2\ U)^\dagger))$ \$\$ $(i, j) = 1_m\ 4$ \$\$ (i, j)
proof −
have $(control2\ U * ((control2\ U)^\dagger))$ \$\$ $(3,0)$ =
$(control2\ U)$ \$\$ $(3,0) * ((control2\ U)^\dagger)$ \$\$ $(0,0)$ +
$(control2\ U)$ \$\$ $(3,1) * ((control2\ U)^\dagger)$ \$\$ $(1,0)$ +
$(control2\ U)$ \$\$ $(3,2) * ((control2\ U)^\dagger)$ \$\$ $(2,0)$ +
$(control2\ U)$ \$\$ $(3,3) * ((control2\ U)^\dagger)$ \$\$ $(3,0)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger
dim-row-of-dagger i3 i4 index-matrix-prod j0 j4)
also have ... = $(control2\ U)$ \$\$ $(3,1) * ((control2\ U)^\dagger)$ \$\$ $(1,0)$ +
$(control2\ U)$ \$\$ $(3,3) * ((control2\ U)^\dagger)$ \$\$ $(3,0)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have ... = (*control2 U*) \$\$ (*3*,*1*) ∗ (*cnj* ((*control2 U*) \$\$ (*0*,*1*))) +
(*control2 U*) \$\$ (*3*,*3*) ∗ (*cnj* ((*control2 U*) \$\$ (*0*,*3*)))
using *dagger-def Tensor.mat-of-cols-list-def control2-def* **by** *auto*
also have ... = (*control2 U*) \$\$ (*3*,*1*) ∗ (*cnj 0*) +
(*control2 U*) \$\$ (*3*,*3*) ∗ (*cnj 0*)
using *control2-def index-mat-of-cols-list* **by** *simp*

also have ... = *0* **by** *auto*
also have ... = $1_m\ 4\ \$\$\ (3,0)$ **by** *simp*
finally show *?thesis* **using** *i3 j0* **by** *simp*
qed
next
assume *jl3*:$j = 1 \lor j = 2 \lor j = 3$
show $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof (*rule disjE*)
show $j = 1 \lor j = 2 \lor j = 3$ **using** *jl3* **by** *this*
next
assume *j1*:$j = 1$
show $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof −
have $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (3,1) =$
$(control2\ U)\ \$\$\ (3,0) * ((control2\ U)^{\dagger})\ \$\$\ (0,1) +$
$(control2\ U)\ \$\$\ (3,1) * ((control2\ U)^{\dagger})\ \$\$\ (1,1) +$
$(control2\ U)\ \$\$\ (3,2) * ((control2\ U)^{\dagger})\ \$\$\ (2,1) +$
$(control2\ U)\ \$\$\ (3,3) * ((control2\ U)^{\dagger})\ \$\$\ (3,1)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger
dim-row-of-dagger i3 i4 index-matrix-prod j1 j4)
also have ... = $(control2\ U)\ \$\$\ (3,1) * ((control2\ U)^{\dagger})\ \$\$\ (1,1) +$
$(control2\ U)\ \$\$\ (3,3) * ((control2\ U)^{\dagger})\ \$\$\ (3,1)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have ... = $(control2\ U)\ \$\$\ (3,1) * (cnj\ ((control2\ U)\ \$\$\ (1,1))) +$
$(control2\ U)\ \$\$\ (3,3) * (cnj\ ((control2\ U)\ \$\$\ (1,3)))$
using *dagger-def Tensor.mat-of-cols-list-def control2-def* **by** *auto*
also have ... = $U\ \$\$\ (1,0) * (cnj\ (U\ \$\$\ (0,0))) +$
$U\ \$\$\ (1,1) * (cnj\ (U\ \$\$\ (0,1)))$
using *control2-def index-mat-of-cols-list* **by** *simp*
also have ... = $(U\ \$\$\ (1,0)) * ((U^{\dagger})\ \$\$\ (0,0)) +$
$(U\ \$\$\ (1,1)) * ((U^{\dagger})\ \$\$\ (1,0))$
using *dagger-def assms*(*1*) *gate-def* **by** *force*
also have ... = $(U * (U^{\dagger}))\ \$\$\ (1,0)$
using *times-mat-def assms*(*1*) *gate-carrier-mat sumof2*
by (*smt* (*z3*) *Suc-1 carrier-matD*(*2*) *dagger-def dim-col-mat*(*1*)
dim-row-of-dagger
gate.dim-row index-matrix-prod lessI pos2 power-one-right)
also have ... = $(1_m\ 2)\ \$\$\ (1,0)$ **using** *assms*(*1*) *gate-def unitary-def*
by *auto*
also have ... = *0* **by** *auto*
also have ... = $1_m\ 4\ \$\$\ (3,1)$ **by** *simp*
finally show *?thesis* **using** *i3 j1* **by** *simp*
qed
next
assume *jl2*:$j = 2 \lor j = 3$
show $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof (*rule disjE*)

show $j = 2 \vee j = 3$ **using** *jl2* **by** *this*
next
assume $j2{:}j = 2$
show $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof $-$
have $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (3,2) =$
$(control2\ U)\ \$\$\ (3,0) * ((control2\ U)^{\dagger})\ \$\$\ (0,2)\ +$
$(control2\ U)\ \$\$\ (3,1) * ((control2\ U)^{\dagger})\ \$\$\ (1,2)\ +$
$(control2\ U)\ \$\$\ (3,2) * ((control2\ U)^{\dagger})\ \$\$\ (2,2)\ +$
$(control2\ U)\ \$\$\ (3,3) * ((control2\ U)^{\dagger})\ \$\$\ (3,2)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger
dim-row-of-dagger i3 i4 index-matrix-prod j2 j4)
also have $\ldots = (control2\ U)\ \$\$\ (3,1) * ((control2\ U)^{\dagger})\ \$\$\ (1,2)\ +$
$(control2\ U)\ \$\$\ (3,3) * ((control2\ U)^{\dagger})\ \$\$\ (3,2)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have $\ldots = (control2\ U)\ \$\$\ (3,1) * (cnj\ ((control2\ U)\ \$\$\ (2,1)))$
$+$
$(control2\ U)\ \$\$\ (3,3) * (cnj\ ((control2\ U)\ \$\$\ (2,3)))$
using *dagger-def Tensor.mat-of-cols-list-def control2-def* **by** *auto*
also have $\ldots = (control2\ U)\ \$\$\ (3,1) * (cnj\ 0)\ +$
$(control2\ U)\ \$\$\ (3,3) * (cnj\ 0)$
using *control2-def index-mat-of-cols-list* **by** *simp*
also have $\ldots = 0$ **by** *auto*
also have $\ldots = 1_m\ 4\ \$\$\ (3,2)$ **by** *simp*
finally show *?thesis* **using** *i3 j2* **by** *simp*
qed
next
assume $j3{:}j = 3$
show $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof $-$
have $(control2\ U * ((control2\ U)^{\dagger}))\ \$\$\ (3,3) =$
$(control2\ U)\ \$\$\ (3,0) * ((control2\ U)^{\dagger})\ \$\$\ (0,3)\ +$
$(control2\ U)\ \$\$\ (3,1) * ((control2\ U)^{\dagger})\ \$\$\ (1,3)\ +$
$(control2\ U)\ \$\$\ (3,2) * ((control2\ U)^{\dagger})\ \$\$\ (2,3)\ +$
$(control2\ U)\ \$\$\ (3,3) * ((control2\ U)^{\dagger})\ \$\$\ (3,3)$
using *times-mat-def sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger
dim-row-of-dagger i3 i4 index-matrix-prod j3 j4)
also have $\ldots = (control2\ U)\ \$\$\ (3,1) * ((control2\ U)^{\dagger})\ \$\$\ (1,3)\ +$
$(control2\ U)\ \$\$\ (3,3) * ((control2\ U)^{\dagger})\ \$\$\ (3,3)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have $\ldots = (control2\ U)\ \$\$\ (3,1) * (cnj\ ((control2\ U)\ \$\$\ (3,1)))$
$+$
$(control2\ U)\ \$\$\ (3,3) * (cnj\ ((control2\ U)\ \$\$\ (3,3)))$
using *dagger-def Tensor.mat-of-cols-list-def control2-def* **by** *auto*
also have $\ldots = U\ \$\$\ (1,0) * (cnj\ (U\ \$\$\ (1,0)))\ +$

```
                  U $$ (1,1) ∗ (cnj (U $$ (1,1)))
          using control2-def index-mat-of-cols-list by simp
        also have ... = (U $$ (1,0)) ∗ ((U†) $$ (0,1)) +
                  (U $$ (1,1)) ∗ ((U†) $$ (1,1))
          using dagger-def assms(1) gate-def by force
        also have ... = (U ∗ (U†)) $$ (1,1)
          using times-mat-def assms(1) gate-carrier-mat sumof2
             by (smt (z3) Suc-1 carrier-matD(2) dagger-def dim-col-mat(1)
dim-row-of-dagger
             gate.dim-row index-matrix-prod lessI pos2 power-one-right)
        also have ... = (1m 2) $$ (1,1) using assms(1) gate-def unitary-def
by auto
        also have ... = 1 by auto
        also have ... = 1m 4 $$ (3,3) by simp
        finally show ?thesis using i3 j3 by simp
       qed
      qed
     qed
    qed
   qed
  qed
qed
qed
next
  show dim-row (control2 U ∗ ((control2 U)†)) = dim-row (1m 4)
   by (metis carrier-matD(1) control2-carrier-mat index-mult-mat(2) index-one-mat(2))
next
  show dim-col (control2 U ∗ ((control2 U)†)) = dim-col (1m 4)
   by (metis carrier-matD(1) control2-carrier-mat dim-col-of-dagger index-mult-mat(3)

    index-one-mat(3))
qed

lemma control2-inv':
  assumes gate 1 U
  shows (control2 U)† ∗ (control2 U) = 1m 4
proof
  show ⋀i j. i < dim-row (1m 4) ⟹ j < dim-col (1m 4) ⟹
        ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
  proof −
    fix i j
    assume i < dim-row (1m 4)
    hence i4:i < 4 by auto
    assume j < dim-col (1m 4)
    hence j4:j < 4 by auto
    show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
    proof (rule disjE)
      show i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3 using i4 by auto
    next
```

assume *i0*:$i = 0$
show $((control2\ U)^\dagger * control2\ U)\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof (*rule disjE*)
show $j = 0 \lor j = 1 \lor j = 2 \lor j = 3$ **using** *j4* **by** *auto*
next
assume *j0*:$j = 0$
show $((control2\ U)^\dagger * control2\ U)\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof −
have $((control2\ U)^\dagger * control2\ U)\ \$\$\ (0,0) =$
$((control2\ U)^\dagger)\ \$\$\ (0,0) * (control2\ U)\ \$\$\ (0,0) +$
$((control2\ U)^\dagger)\ \$\$\ (0,1) * (control2\ U)\ \$\$\ (1,0) +$
$((control2\ U)^\dagger)\ \$\$\ (0,2) * (control2\ U)\ \$\$\ (2,0) +$
$((control2\ U)^\dagger)\ \$\$\ (0,3) * (control2\ U)\ \$\$\ (3,0)$
using *sumof4*
by (*metis* (*no-types*, *lifting*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger dim-row-of-dagger i0 i4 index-matrix-prod)
also have ... $= ((control2\ U)^\dagger)\ \$\$\ (0,0)$
using *control2-def index-mat-of-cols-list* **by** *force*
also have ... $= cnj\ ((control2\ U)\ \$\$\ (0,0))$
using *dagger-def*
by (*simp add*: *Tensor.mat-of-cols-list-def control2-def*)
also have ... $= 1$ **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have ... $= 1_m\ 4\ \$\$\ (0,0)$ **by** *simp*
finally show *?thesis* **using** *i0 j0* **by** *simp*
qed
next
assume *jl3*:$j = 1 \lor j = 2 \lor j = 3$
show $((control2\ U)^\dagger * control2\ U)\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof (*rule disjE*)
show $j = 1 \lor j = 2 \lor j = 3$ **using** *jl3* **by** *this*
next
assume *j1*:$j = 1$
show $((control2\ U)^\dagger * control2\ U)\ \$\$\ (i, j) = 1_m\ 4\ \$\$\ (i, j)$
proof −
have $((control2\ U)^\dagger * control2\ U)\ \$\$\ (0,1) =$
$((control2\ U)^\dagger)\ \$\$\ (0,0) * (control2\ U)\ \$\$\ (0,1) +$
$((control2\ U)^\dagger)\ \$\$\ (0,1) * (control2\ U)\ \$\$\ (1,1) +$
$((control2\ U)^\dagger)\ \$\$\ (0,2) * (control2\ U)\ \$\$\ (2,1) +$
$((control2\ U)^\dagger)\ \$\$\ (0,3) * (control2\ U)\ \$\$\ (3,1)$
using *sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat dim-col-of-dagger*
dim-row-of-dagger index-matrix-prod one-less-numeral-iff semiring-norm(*76*)
zero-less-numeral)
also have ... $= ((control2\ U)^\dagger)\ \$\$\ (0,1) * (control2\ U)\ \$\$\ (1,1) +$
$((control2\ U)^\dagger)\ \$\$\ (0,3) * (control2\ U)\ \$\$\ (3,1)$
using *control2-def index-mat-of-cols-list* **by** *force*

```
      also have ... = cnj ((control2 U) $$ (1,0)) * (control2 U) $$ (1,1) +
                cnj ((control2 U) $$ (3,0)) * (control2 U) $$ (3,1)
        using dagger-def
        by (simp add: Tensor.mat-of-cols-list-def control2-def)
      also have ... = 0 using control2-def index-mat-of-cols-list by auto
      also have ... = 1m 4 $$ (0,1) by simp
      finally show ?thesis using i0 j1 by simp
    qed
  next
    assume jl2:j = 2 ∨ j = 3
    show ((control2 U)† * control2 U) $$ (i, j) = 1m 4 $$ (i, j)
    proof (rule disjE)
      show j = 2 ∨ j = 3 using jl2 by this
    next
      assume j2:j = 2
      show ((control2 U)† * control2 U) $$ (i, j) = 1m 4 $$ (i, j)
      proof –
        have ((control2 U)† * control2 U) $$ (0,2) =
          ((control2 U)†) $$ (0,0) * (control2 U) $$ (0,2) +
          ((control2 U)†) $$ (0,1) * (control2 U) $$ (1,2) +
          ((control2 U)†) $$ (0,2) * (control2 U) $$ (2,2) +
          ((control2 U)†) $$ (0,3) * (control2 U) $$ (3,2)
          using sumof4
           by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
              dim-row-of-dagger index-matrix-prod j2 j4 zero-less-numeral)
        also have ... = ((control2 U)†) $$ (0,2)
          using control2-def index-mat-of-cols-list by force
        also have ... = cnj ((control2 U) $$ (2,0))
          using dagger-def
          by (simp add: Tensor.mat-of-cols-list-def control2-def)
        also have ... = 0 using control2-def index-mat-of-cols-list by auto
        also have ... = 1m 4 $$ (0,2) by simp
        finally show ?thesis using i0 j2 by simp
      qed
    next
      assume j3:j = 3
      show ((control2 U)† * control2 U) $$ (i, j) = 1m 4 $$ (i, j)
      proof –
        have ((control2 U)† * control2 U) $$ (0,3) =
          ((control2 U)†) $$ (0,0) * (control2 U) $$ (0,3) +
          ((control2 U)†) $$ (0,1) * (control2 U) $$ (1,3) +
          ((control2 U)†) $$ (0,2) * (control2 U) $$ (2,3) +
          ((control2 U)†) $$ (0,3) * (control2 U) $$ (3,3)
          using sumof4
           by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
              dim-row-of-dagger index-matrix-prod j3 j4 zero-less-numeral)
        also have ... = ((control2 U)†) $$ (0,1) * (control2 U) $$ (1,3) +
```

```
                ((control2 U)†) $$ (0,3) ∗ (control2 U) $$ (3,3)
          using control2-def index-mat-of-cols-list by force
        also have ... = cnj ((control2 U) $$ (1,0)) ∗ (control2 U) $$ (1,3) +
                cnj ((control2 U) $$ (3,0)) ∗ (control2 U) $$ (3,3)
          using dagger-def
          by (simp add: Tensor.mat-of-cols-list-def control2-def)
        also have ... = 0 using control2-def index-mat-of-cols-list by auto
        also have ... = 1m 4 $$ (0,3) by simp
        finally show ?thesis using i0 j3 by simp
      qed
    qed
  qed
qed
next
  assume il3:i = 1 ∨ i = 2 ∨ i = 3
  show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
  proof (rule disjE)
    show i = 1 ∨ i = 2 ∨ i = 3 using il3 by this
  next
    assume i1:i = 1
    show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
    proof (rule disjE)
      show j = 0 ∨ j = 1 ∨ j = 2 ∨ j = 3 using j4 by auto
    next
      assume j0:j = 0
      show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
      proof −
        have ((control2 U)† ∗ control2 U) $$ (1,0) =
            ((control2 U)†) $$ (1,0) ∗ (control2 U) $$ (0,0) +
            ((control2 U)†) $$ (1,1) ∗ (control2 U) $$ (1,0) +
            ((control2 U)†) $$ (1,2) ∗ (control2 U) $$ (2,0) +
            ((control2 U)†) $$ (1,3) ∗ (control2 U) $$ (3,0)
          using sumof4
          by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
              dim-row-of-dagger index-matrix-prod one-less-numeral-iff semir-
ing-norm(76)
            zero-less-numeral)
        also have ... = ((control2 U)†) $$ (1,0)
          using control2-def index-mat-of-cols-list by force
        also have ... = cnj ((control2 U) $$ (0,1))
          using dagger-def
          by (simp add: Tensor.mat-of-cols-list-def control2-def)
        also have ... = 0 using control2-def index-mat-of-cols-list by auto
        also have ... = 1m 4 $$ (1,0) by simp
        finally show ?thesis using i1 j0 by simp
      qed
    next
      assume jl3:j = 1 ∨ j = 2 ∨ j = 3
```

```
show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
proof (rule disjE)
  show j = 1 ∨ j = 2 ∨ j = 3 using jl3 by this
next
  assume j1:j = 1
  show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
  proof −
    have ((control2 U)† ∗ control2 U) $$ (1,1) =
      ((control2 U)†) $$ (1,0) ∗ (control2 U) $$ (0,1) +
      ((control2 U)†) $$ (1,1) ∗ (control2 U) $$ (1,1) +
      ((control2 U)†) $$ (1,2) ∗ (control2 U) $$ (2,1) +
      ((control2 U)†) $$ (1,3) ∗ (control2 U) $$ (3,1)
      using sumof4
       by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
           dim-row-of-dagger index-matrix-prod one-less-numeral-iff semir-
ing-norm(76)
         zero-less-numeral)
    also have . . . = ((control2 U)†) $$ (1,1) ∗ (control2 U) $$ (1,1) +
                 ((control2 U)†) $$ (1,3) ∗ (control2 U) $$ (3,1)
      using control2-def index-mat-of-cols-list by force
   also have . . . = cnj ((control2 U) $$ (1,1)) ∗ (control2 U) $$ (1,1) +
                 cnj ((control2 U) $$ (3,1)) ∗ (control2 U) $$ (3,1)
      using dagger-def
      by (simp add: Tensor.mat-of-cols-list-def control2-def)
    also have . . . = cnj (U $$ (0,0)) ∗ (U $$ (0,0)) +
                 cnj (U $$ (1,0)) ∗ (U $$ (1,0))
      using control2-def index-mat-of-cols-list by simp
    also have . . . = ((U†) ∗ U) $$ (0,0)
      using times-mat-def sumof2 assms(1) gate-carrier-mat
          by (smt (verit, del-insts) Suc-1 carrier-matD(2) dagger-def
dim-col-mat(1)
           dim-row-of-dagger gate.dim-row index-mat(1) index-matrix-prod
lessI
         old.prod.case pos2 power-one-right)
    also have . . . = (1m 2) $$ (0,0) using assms(1) gate-def unitary-def
by auto
    also have . . . = 1 using control2-def index-mat-of-cols-list by auto
    also have . . . = 1m 4 $$ (1,1) by simp
    finally show ?thesis using i1 j1 by simp
  qed
next
  assume jl2:j = 2 ∨ j = 3
  show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
  proof (rule disjE)
    show j = 2 ∨ j = 3 using jl2 by this
  next
    assume j2:j = 2
    show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
```

proof −
have $((control2\ U)^\dagger * control2\ U)$ \$\$ $(1,2) =$
$((control2\ U)^\dagger)$ \$\$ $(1,0) * (control2\ U)$ \$\$ $(0,2) +$
$((control2\ U)^\dagger)$ \$\$ $(1,1) * (control2\ U)$ \$\$ $(1,2) +$
$((control2\ U)^\dagger)$ \$\$ $(1,2) * (control2\ U)$ \$\$ $(2,2) +$
$((control2\ U)^\dagger)$ \$\$ $(1,3) * (control2\ U)$ \$\$ $(3,2)$
using *sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat* *dim-col-of-dagger* *dim-row-of-dagger* *index-matrix-prod* *j2* *j4* *one-less-numeral-iff* *semiring-norm*(*76*))
also have ... = $((control2\ U)^\dagger)$ \$\$ $(1,2)$
using *control2-def* *index-mat-of-cols-list* **by** *force*
also have ... = *cnj* ((*control2 U*) \$\$ (*2*,*1*))
using *dagger-def*
by (*simp add*: *Tensor.mat-of-cols-list-def* *control2-def*)
also have ... = *0* **using** *control2-def* *index-mat-of-cols-list* **by** *auto*
also have ... = $1_m\ 4$ \$\$ (*1*,*2*) **by** *simp*
finally show *?thesis* **using** *i1* *j2* **by** *simp*
qed
next
assume *j3*:*j* = *3*
show $((control2\ U)^\dagger * control2\ U)$ \$\$ $(i, j) = 1_m\ 4$ \$\$ (i, j)
proof −
have $((control2\ U)^\dagger * control2\ U)$ \$\$ $(1,3) =$
$((control2\ U)^\dagger)$ \$\$ $(1,0) * (control2\ U)$ \$\$ $(0,3) +$
$((control2\ U)^\dagger)$ \$\$ $(1,1) * (control2\ U)$ \$\$ $(1,3) +$
$((control2\ U)^\dagger)$ \$\$ $(1,2) * (control2\ U)$ \$\$ $(2,3) +$
$((control2\ U)^\dagger)$ \$\$ $(1,3) * (control2\ U)$ \$\$ $(3,3)$
using *sumof4*
by (*metis* (*no-types*, *lifting*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat* *dim-col-of-dagger* *dim-row-of-dagger* *i1* *i4* *index-matrix-prod* *j3* *j4*)
also have ... = $((control2\ U)^\dagger)$ \$\$ $(1,1) * (control2\ U)$ \$\$ $(1,3) +$
$((control2\ U)^\dagger)$ \$\$ $(1,3) * (control2\ U)$ \$\$ $(3,3)$
using *control2-def* *index-mat-of-cols-list* **by** *force*
also have ... = *cnj* ((*control2 U*) \$\$ (*1*,*1*)) ∗ (*control2 U*) \$\$ (*1*,*3*)
+
cnj ((*control2 U*) \$\$ (*3*,*1*)) ∗ (*control2 U*) \$\$ (*3*,*3*)
using *dagger-def*
by (*simp add*: *Tensor.mat-of-cols-list-def* *control2-def*)
also have ... = *cnj* (*U* \$\$ (*0*,*0*)) ∗ (*U* \$\$ (*0*,*1*)) +
cnj (*U* \$\$ (*1*,*0*)) ∗ (*U* \$\$ (*1*,*1*))
using *control2-def* *index-mat-of-cols-list* **by** *simp*
also have ... = $((U^\dagger) * U)$ \$\$ $(0,1)$
using *times-mat-def* *sumof2* *assms*(*1*) *gate-carrier-mat*
by (*smt* (*verit*, *del-insts*) *Suc-1* *carrier-matD*(*2*) *dagger-def*
dim-col-mat(*1*)
dim-row-of-dagger *gate.dim-row* *index-mat*(*1*) *index-matrix-prod*
lessI

```
                  old.prod.case pos2 power-one-right)
              also have ... = (1m 2) $$ (0,1) using assms(1) gate-def unitary-def
by auto
              also have ... = 0 using control2-def index-mat-of-cols-list by auto
              also have ... = 1m 4 $$ (1,3) by simp
              finally show ?thesis using i1 j3 by simp
            qed
          qed
        qed
      qed
    next
      assume il2:i = 2 ∨ i = 3
      show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
      proof (rule disjE)
        show i = 2 ∨ i = 3 using il2 by this
      next
        assume i2:i = 2
        show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
        proof (rule disjE)
          show j = 0 ∨ j = 1 ∨ j = 2 ∨ j = 3 using j4 by auto
        next
          assume j0:j = 0
          show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
          proof −
            have ((control2 U)† ∗ control2 U) $$ (2,0) =
              ((control2 U)†) $$ (2,0) ∗ (control2 U) $$ (0,0) +
              ((control2 U)†) $$ (2,1) ∗ (control2 U) $$ (1,0) +
              ((control2 U)†) $$ (2,2) ∗ (control2 U) $$ (2,0) +
              ((control2 U)†) $$ (2,3) ∗ (control2 U) $$ (3,0)
              using sumof4
              by (smt (z3) carrier-matD(1) carrier-matD(2) control2-carrier-mat
dim-col-of-dagger
                  dim-row-of-dagger i2 i4 index-matrix-prod zero-less-numeral)
            also have ... = ((control2 U)†) $$ (2,0)
              using control2-def index-mat-of-cols-list by force
            also have ... = cnj ((control2 U) $$ (0,2))
              using dagger-def
              by (simp add: Tensor.mat-of-cols-list-def control2-def)
            also have ... = 0 using control2-def index-mat-of-cols-list by auto
            also have ... = 1m 4 $$ (2,0) by simp
            finally show ?thesis using i2 j0 by simp
          qed
        next
          assume jl3:j = 1 ∨ j = 2 ∨ j = 3
          show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
          proof (rule disjE)
            show j = 1 ∨ j = 2 ∨ j = 3 using jl3 by this
          next
            assume j1:j = 1
```

show $((control2\ U)^\dagger * control2\ U)$ \$\$ $(i, j) = 1_m\ 4$ \$\$ (i, j)
proof −
have $((control2\ U)^\dagger * control2\ U)$ \$\$ $(2,1) =$
$((control2\ U)^\dagger)$ \$\$ $(2,0) * (control2\ U)$ \$\$ $(0,1)$ +
$((control2\ U)^\dagger)$ \$\$ $(2,1) * (control2\ U)$ \$\$ $(1,1)$ +
$((control2\ U)^\dagger)$ \$\$ $(2,2) * (control2\ U)$ \$\$ $(2,1)$ +
$((control2\ U)^\dagger)$ \$\$ $(2,3) * (control2\ U)$ \$\$ $(3,1)$
using *sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat* *dim-col-of-dagger* *dim-row-of-dagger* *i2* *i4* *index-matrix-prod* *one-less-numeral-iff* *semiring-norm*(*76*))
also have ... = $((control2\ U)^\dagger)$ \$\$ $(2,1) * (control2\ U)$ \$\$ $(1,1)$ +
$((control2\ U)^\dagger)$ \$\$ $(2,3) * (control2\ U)$ \$\$ $(3,1)$
using *control2-def* *index-mat-of-cols-list* **by** *force*
also have ... = *cnj* $((control2\ U)$ \$\$ $(1,2)) * (control2\ U)$ \$\$ $(1,1)$
+
cnj $((control2\ U)$ \$\$ $(3,2)) * (control2\ U)$ \$\$ $(3,1)$
using *dagger-def*
by (*simp add*: *Tensor.mat-of-cols-list-def* *control2-def*)
also have ... = *0* **using** *control2-def* *index-mat-of-cols-list* **by** *auto*
also have ... = $1_m\ 4$ \$\$ $(2,1)$ **by** *simp*
finally show *?thesis* **using** *i2* *j1* **by** *simp*
qed
next
assume *jl2*:$j = 2 \lor j = 3$
show $((control2\ U)^\dagger * control2\ U)$ \$\$ $(i, j) = 1_m\ 4$ \$\$ (i, j)
proof (*rule disjE*)
show $j = 2 \lor j = 3$ **using** *jl2* **by** *this*
next
assume *j2*:$j = 2$
show $((control2\ U)^\dagger * control2\ U)$ \$\$ $(i, j) = 1_m\ 4$ \$\$ (i, j)
proof −
have $((control2\ U)^\dagger * control2\ U)$ \$\$ $(2,2) =$
$((control2\ U)^\dagger)$ \$\$ $(2,0) * (control2\ U)$ \$\$ $(0,2)$ +
$((control2\ U)^\dagger)$ \$\$ $(2,1) * (control2\ U)$ \$\$ $(1,2)$ +
$((control2\ U)^\dagger)$ \$\$ $(2,2) * (control2\ U)$ \$\$ $(2,2)$ +
$((control2\ U)^\dagger)$ \$\$ $(2,3) * (control2\ U)$ \$\$ $(3,2)$
using *sumof4*
by (*smt* (*z3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *control2-carrier-mat*
dim-col-of-dagger
dim-row-of-dagger *i2* *i4* *index-matrix-prod* *zero-less-numeral*)
also have ... = $((control2\ U)^\dagger)$ \$\$ $(2,2)$
using *control2-def* *index-mat-of-cols-list* **by** *force*
also have ... = *cnj* $((control2\ U)$ \$\$ $(2,2))$
using *dagger-def*
by (*simp add*: *Tensor.mat-of-cols-list-def* *control2-def*)
also have ... = *1* **using** *control2-def* *index-mat-of-cols-list* **by** *auto*
also have ... = $1_m\ 4$ \$\$ $(2,2)$ **by** *simp*
finally show *?thesis* **using** *i2* *j2* **by** *simp*

```
          qed
        next
          assume j3:j = 3
          show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
          proof −
            have ((control2 U)† ∗ control2 U) $$ (2,3) =
                  ((control2 U)†) $$ (2,0) ∗ (control2 U) $$ (0,3) +
                  ((control2 U)†) $$ (2,1) ∗ (control2 U) $$ (1,3) +
                  ((control2 U)†) $$ (2,2) ∗ (control2 U) $$ (2,3) +
                  ((control2 U)†) $$ (2,3) ∗ (control2 U) $$ (3,3)
              using sumof4
              by (metis (no-types, lifting) carrier-matD(1) carrier-matD(2)
                 control2-carrier-mat dim-col-of-dagger dim-row-of-dagger i2 i4
                  index-matrix-prod j3 j4)
           also have . . . = ((control2 U)†) $$ (2,1) ∗ (control2 U) $$ (1,3) +
                          ((control2 U)†) $$ (2,3) ∗ (control2 U) $$ (3,3)
              using control2-def index-mat-of-cols-list by force
           also have . . . = cnj ((control2 U) $$ (1,2)) ∗ (control2 U) $$ (1,3)
+
                          cnj ((control2 U) $$ (3,2)) ∗ (control2 U) $$ (3,3)
              using dagger-def
              by (simp add: Tensor.mat-of-cols-list-def control2-def)
           also have . . . = 0 using control2-def index-mat-of-cols-list by auto
            also have . . . = 1m 4 $$ (2,3) by simp
            finally show ?thesis using i2 j3 by simp
          qed
        qed
      qed
    qed
  next
    assume i3:i = 3
    show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
    proof (rule disjE)
      show j = 0 ∨ j = 1 ∨ j = 2 ∨ j = 3 using j4 by auto
    next
      assume j0:j = 0
      show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
      proof −
        have ((control2 U)† ∗ control2 U) $$ (3,0) =
          ((control2 U)†) $$ (3,0) ∗ (control2 U) $$ (0,0) +
          ((control2 U)†) $$ (3,1) ∗ (control2 U) $$ (1,0) +
          ((control2 U)†) $$ (3,2) ∗ (control2 U) $$ (2,0) +
          ((control2 U)†) $$ (3,3) ∗ (control2 U) $$ (3,0)
          using sumof4
             by (metis (no-types, lifting) carrier-matD(1) carrier-matD(2)
control2-carrier-mat
             dim-col-of-dagger dim-row-of-dagger i3 i4 index-matrix-prod j0 j4)
        also have . . . = ((control2 U)†) $$ (3,0)
          using control2-def index-mat-of-cols-list by force
```

also have . . . = *cnj* ((*control2 U*) $$ (*0*,*3*))
using *dagger-def*
by (*simp add*: *Tensor*.*mat-of-cols-list-def control2-def*)
also have . . . = *0* **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have . . . = 1_m *4* $$ (*3*,*0*) **by** *simp*
finally show *?thesis* **using** *i3 j0* **by** *simp*
qed
next
assume *jl3*:*j* = *1* ∨ *j* = *2* ∨ *j* = *3*
show ((*control2 U*)† ∗ *control2 U*) $$ (*i*, *j*) = 1_m *4* $$ (*i*, *j*)
proof (*rule disjE*)
show *j* = *1* ∨ *j* = *2* ∨ *j* = *3* **using** *jl3* **by** *this*
next
assume *j1*:*j* = *1*
show ((*control2 U*)† ∗ *control2 U*) $$ (*i*, *j*) = 1_m *4* $$ (*i*, *j*)
proof −
have ((*control2 U*)† ∗ *control2 U*) $$ (*3*,*1*) =
((*control2 U*)†) $$ (*3*,*0*) ∗ (*control2 U*) $$ (*0*,*1*) +
((*control2 U*)†) $$ (*3*,*1*) ∗ (*control2 U*) $$ (*1*,*1*) +
((*control2 U*)†) $$ (*3*,*2*) ∗ (*control2 U*) $$ (*2*,*1*) +
((*control2 U*)†) $$ (*3*,*3*) ∗ (*control2 U*) $$ (*3*,*1*)
using *sumof4*
by (*metis* (*no-types*, *lifting*) *carrier-matD*(*1*) *carrier-matD*(*2*)
control2-carrier-mat dim-col-of-dagger dim-row-of-dagger i3 i4
index-matrix-prod j1 j4)
also have . . . = ((*control2 U*)†) $$ (*3*,*1*) ∗ (*control2 U*) $$ (*1*,*1*) +
((*control2 U*)†) $$ (*3*,*3*) ∗ (*control2 U*) $$ (*3*,*1*)
using *control2-def index-mat-of-cols-list* **by** *force*
also have . . . = *cnj* ((*control2 U*) $$ (*1*,*3*)) ∗ (*control2 U*) $$ (*1*,*1*)
+
cnj ((*control2 U*) $$ (*3*,*3*)) ∗ (*control2 U*) $$ (*3*,*1*)
using *dagger-def*
by (*simp add*: *Tensor*.*mat-of-cols-list-def control2-def*)
also have . . . = *cnj* (*U* $$ (*0*,*1*)) ∗ (*U* $$ (*0*,*0*)) +
cnj (*U* $$ (*1*,*1*)) ∗ (*U* $$ (*1*,*0*))
using *control2-def index-mat-of-cols-list* **by** *simp*
also have . . . = ((*U*†) ∗ *U*) $$ (*1*,*0*)
using *times-mat-def sumof2 assms*(*1*) *gate-carrier-mat*
by (*smt* (*verit*, *del-insts*) *Suc-1 carrier-matD*(*2*) *dagger-def*
dim-col-mat(*1*)
dim-row-of-dagger gate.*dim-row index-mat*(*1*) *index-matrix-prod*
lessI
old.*prod*.*case pos2 power-one-right*)
also have . . . = (1_m *2*) $$ (*1*,*0*) **using** *assms*(*1*) *gate-def unitary-def*
by *auto*
also have . . . = *0* **using** *control2-def index-mat-of-cols-list* **by** *auto*
also have . . . = 1_m *4* $$ (*3*,*1*) **by** *simp*
finally show *?thesis* **using** *i3 j1* **by** *simp*
qed

```
        next
          assume jl2:j = 2 ∨ j = 3
          show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
          proof (rule disjE)
            show j = 2 ∨ j = 3 using jl2 by this
          next
            assume j2:j = 2
            show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
            proof −
              have ((control2 U)† ∗ control2 U) $$ (3,2) =
                    ((control2 U)†) $$ (3,0) ∗ (control2 U) $$ (0,2) +
                    ((control2 U)†) $$ (3,1) ∗ (control2 U) $$ (1,2) +
                    ((control2 U)†) $$ (3,2) ∗ (control2 U) $$ (2,2) +
                    ((control2 U)†) $$ (3,3) ∗ (control2 U) $$ (3,2)
                using sumof4
                  by (metis (no-types, lifting) carrier-matD(1) carrier-matD(2)
control2-carrier-mat
                    dim-col-of-dagger dim-row-of-dagger i3 i4 index-matrix-prod j2
j4)
              also have . . . = ((control2 U)†) $$ (3,2)
                using control2-def index-mat-of-cols-list by force
              also have . . . = cnj ((control2 U) $$ (2,3))
                using dagger-def
                by (simp add: Tensor.mat-of-cols-list-def control2-def)
              also have . . . = 0 using control2-def index-mat-of-cols-list by auto
              also have . . . = 1m 4 $$ (3,2) by simp
              finally show ?thesis using i3 j2 by simp
            qed
          next
            assume j3:j = 3
            show ((control2 U)† ∗ control2 U) $$ (i, j) = 1m 4 $$ (i, j)
            proof −
              have ((control2 U)† ∗ control2 U) $$ (3,3) =
                    ((control2 U)†) $$ (3,0) ∗ (control2 U) $$ (0,3) +
                    ((control2 U)†) $$ (3,1) ∗ (control2 U) $$ (1,3) +
                    ((control2 U)†) $$ (3,2) ∗ (control2 U) $$ (2,3) +
                    ((control2 U)†) $$ (3,3) ∗ (control2 U) $$ (3,3)
                using sumof4
                by (metis (no-types, lifting) carrier-matD(1) carrier-matD(2)
                    control2-carrier-mat dim-col-of-dagger dim-row-of-dagger i3
                    index-matrix-prod j3 j4)
              also have . . . = ((control2 U)†) $$ (3,1) ∗ (control2 U) $$ (1,3) +
                          ((control2 U)†) $$ (3,3) ∗ (control2 U) $$ (3,3)
                using control2-def index-mat-of-cols-list by force
              also have . . . = cnj ((control2 U) $$ (1,3)) ∗ (control2 U) $$ (1,3)
+
                          cnj ((control2 U) $$ (3,3)) ∗ (control2 U) $$ (3,3)
                using dagger-def
                by (simp add: Tensor.mat-of-cols-list-def control2-def)
```

```
              also have ... = cnj (U $$ (0,1)) * (U $$ (0,1)) +
                      cnj (U $$ (1,1)) * (U $$ (1,1))
               using control2-def index-mat-of-cols-list by simp
              also have ... = ((U†) * U) $$ (1,1)
               using times-mat-def sumof2 assms(1) gate-carrier-mat
                  by (smt (verit, del-insts) Suc-1 carrier-matD(2) dagger-def
dim-col-mat(1)
                dim-row-of-dagger gate.dim-row index-mat(1) index-matrix-prod
lessI
                 old.prod.case pos2 power-one-right)
             also have ... = (1m 2) $$ (1,1) using assms(1) gate-def unitary-def
by auto
              also have ... = 1 using control2-def index-mat-of-cols-list by auto
              also have ... = 1m 4 $$ (3,3) by simp
              finally show ?thesis using i3 j3 by simp
             qed
            qed
           qed
          qed
         qed
        qed
       qed
      qed
next
  show dim-row ((control2 U)† * control2 U) = dim-row (1m 4)
    by (metis carrier-matD(2) control2-carrier-mat dim-row-of-dagger
        index-mult-mat(2) index-one-mat(2))
next
  show dim-col ((control2 U)† * control2 U) = dim-col (1m 4)
    by (metis carrier-matD(2) control2-carrier-mat index-mult-mat(3)
        index-one-mat(3))
qed

lemma control2-is-gate:
  assumes gate 1 U
  shows gate 2 (control2 U)
proof
  show dim-row (control2 U) = 2^2 using control2-carrier-mat
    by (simp add: Tensor.mat-of-cols-list-def control2-def)
next
  show square-mat (control2 U)
   by (metis carrier-matD(1) carrier-matD(2) control2-carrier-mat square-mat.elims(3))
next
  show unitary (control2 U)
    using control2-inv control2-inv' unitary-def
    by (metis assms carrier-matD(1) carrier-matD(2) control2-carrier-mat)
qed

lemma SWAP-down-is-gate:
```

shows *gate n* (*SWAP-down n*)
proof (*induct n rule*: *SWAP-down.induct*)
case *1*
then show *?case*
by (*metis Quantum.Id-def SWAP-down.simps*(*1*) *SWAP-up.simps*(*1*) *SWAP-up-carrier-mat*

carrier-matD(*2*) *id-is-gate index-one-mat*(*3*))
next
case *2*
then show *?case*
by (*metis H-inv H-is-gate One-nat-def SWAP-down.simps*(*2*) *prod-of-gate-is-gate*)
next
case *3*
then show *?case*
by (*metis One-nat-def SWAP-down.simps*(*3*) *SWAP-is-gate Suc-1*)
next
case (*4 v*)
then show *?case*
proof −
assume *HI*:*gate* (*Suc* (*Suc v*)) (*SWAP-down* (*Suc* (*Suc v*)))
show *gate* (*Suc* (*Suc* (*Suc v*))) (*SWAP-down* (*Suc* (*Suc* (*Suc v*))))
proof −
have *gate* (*Suc* (*Suc* (*Suc v*))) (((1_m (*2^Suc v*)) $\bigotimes$ *SWAP*) ∗
((*SWAP-down* (*Suc* (*Suc v*))) $\bigotimes$ (1_m *2*)))
proof (*rule prod-of-gate-is-gate*)
show *gate* (*Suc* (*Suc* (*Suc v*))) (1_m (*2* ^ *Suc v*) $\bigotimes$ *SWAP*)
using *SWAP-is-gate tensor-gate*
by (*metis Quantum.Id-def add-2-eq-Suc′ id-is-gate*)
next
show *gate* (*Suc* (*Suc* (*Suc v*))) (*SWAP-down* (*Suc* (*Suc v*)) $\bigotimes$ 1_m *2*)
using *HI tensor-gate*
by (*metis Suc-eq-plus1 Y-inv Y-is-gate prod-of-gate-is-gate*)
qed
thus *?thesis* **using** *SWAP-down.simps* **by** *auto*
qed
qed
qed

lemma *SWAP-up-is-gate*:
shows *gate n* (*SWAP-up n*)
proof (*induct n rule*: *SWAP-up.induct*)
case *1*
then show *?case* **using** *id-is-gate SWAP-up.simps*
by (*metis SWAP-down.simps*(*1*) *SWAP-down-is-gate*)
next
case *2*
then show *?case*
by (*metis SWAP-down.simps*(*2*) *SWAP-down-is-gate SWAP-up.simps*(*2*))
next

```
  case 3
  then show ?case
    by (metis One-nat-def SWAP-is-gate SWAP-up.simps(3) Suc-1)
next
  case (4 v)
  then show ?case
  proof –
    assume HI:gate (Suc (Suc v)) (SWAP-up (Suc (Suc v)))
    show gate (Suc (Suc (Suc v))) (SWAP-up (Suc (Suc (Suc v))))
    proof –
      have gate (Suc (Suc (Suc v))) ((SWAP ⨂ (1_m (2^(Suc v)))) * ((1_m 2) ⨂
                                   (SWAP-up (Suc (Suc v)))))
      proof (rule prod-of-gate-is-gate)
        show gate (Suc (Suc (Suc v))) (SWAP ⨂ 1_m (2 ^ Suc v))
          using tensor-gate SWAP-is-gate
          by (metis Quantum.Id-def add-2-eq-Suc id-is-gate)
      next
        show gate (Suc (Suc (Suc v))) (1_m 2 ⨂ SWAP-up (Suc (Suc v)))
          using tensor-gate HI
       by (metis One-nat-def SWAP-down.simps(2) SWAP-down-is-gate plus-1-eq-Suc)
      qed
      thus ?thesis using SWAP-up.simps(3) by simp
    qed
  qed
qed

lemma control-is-gate:
  assumes gate 1 U
  shows gate n (control n U)
proof (cases n)
  case 0
  then show ?thesis
    by (metis SWAP-up.simps(1) SWAP-up-is-gate control.simps(1))
next
  case (Suc nat)
  then show ?thesis
  proof –
    assume nnat:n = Suc nat
    show gate n (control n U)
    proof –
      have gate (Suc nat) (control (Suc nat) U)
      proof (cases nat)
        case 0
        then show ?thesis
          by (simp add: gate-def)
      next
        case (Suc nata)
        then show ?thesis
        proof –
```

assume *nnat-*:*nat* = *Suc nata*
show *gate* (*Suc nat*) (*control* (*Suc nat*) *U*)
proof −
have *gate* (*Suc* (*Suc nata*)) (*control* (*Suc* (*Suc nata*)) *U*)
proof (*cases nata*)
case *0*
then show *?thesis*
using *One-nat-def Suc-1 assms control.simps*(*3*) *control2-is-gate* **by** *presburger*
next
case (*Suc natb*)
then show *?thesis*
proof −
assume *nnatb*:*nata* = *Suc natb*
show *gate* (*Suc* (*Suc nata*)) (*control* (*Suc* (*Suc nata*)) *U*)
proof −
have *gate* (*Suc* (*Suc* (*Suc natb*))) (*control* (*Suc* (*Suc* (*Suc natb*))) *U*)
proof −
have *gate* (*Suc* (*Suc* (*Suc natb*))) (((1_m *2*) $\bigotimes$ *SWAP-down* (*Suc* (*Suc natb*))) ∗
(*control2 U* $\bigotimes$ (1_m (*2*^(*Suc natb*)))) ∗ ((1_m *2*) $\bigotimes$ *SWAP-up* (*Suc* (*Suc natb*))))
proof (*rule prod-of-gate-is-gate*)+
show *gate* (*Suc* (*Suc* (*Suc natb*))) (1_m *2* $\bigotimes$ *SWAP-down* (*Suc* (*Suc natb*)))
using *SWAP-down-is-gate id-is-gate tensor-gate*
by (*metis One-nat-def SWAP-up.simps*(*2*) *SWAP-up-is-gate plus-1-eq-Suc*)
next
show *gate* (*Suc* (*Suc* (*Suc natb*))) (*control2 U* $\bigotimes$ 1_m (*2* ^ *Suc natb*))
using *control2-is-gate id-is-gate tensor-gate*
by (*metis Quantum.Id-def add-2-eq-Suc assms*)
next
show *gate* (*Suc* (*Suc* (*Suc natb*))) (1_m *2* $\bigotimes$ *SWAP-up* (*Suc* (*Suc natb*)))
using *SWAP-up-is-gate id-is-gate tensor-gate*
by (*metis Y-inv Y-is-gate plus-1-eq-Suc prod-of-gate-is-gate*)
qed
thus *?thesis* **using** *control.simps* **by** *simp*
qed
thus *?thesis* **using** *nnatb* **by** *simp*
qed
qed
qed
thus *?thesis* **using** *nnat-* **by** *simp*
qed
qed

```
      qed
      thus ?thesis using nnat by simp
    qed
  qed
qed

lemma controlled-rotations-is-gate:
  shows gate n (controlled-rotations n)
proof (induct n rule: controlled-rotations.induct)
  case 1
  then show ?case
   by (metis SWAP-down.simps(1) SWAP-down-is-gate controlled-rotations.simps(1))
next
  case 2
  then show ?case
   by (metis SWAP-down.simps(2) SWAP-down-is-gate controlled-rotations.simps(2))
next
  case (3 v)
  then show ?case
  proof −
    assume HI:gate (Suc v) (controlled-rotations (Suc v))
    show gate (Suc (Suc v)) (controlled-rotations (Suc (Suc v)))
    proof −
      have gate (Suc (Suc v)) ((control (Suc (Suc v)) (R (Suc (Suc v)))) *
                         ((controlled-rotations (Suc v)) ⨂ (1_m 2)))
      proof (rule prod-of-gate-is-gate)
        show gate (Suc (Suc v)) (control (Suc (Suc v)) (R (Suc (Suc v))))
          using control-is-gate R-is-gate by blast
      next
        show gate (Suc (Suc v)) (controlled-rotations (Suc v) ⨂ 1_m 2)
          using tensor-gate HI id-is-gate
         by (metis One-nat-def SWAP-up.simps(2) SWAP-up-is-gate Suc-eq-plus1)
      qed
      thus ?thesis using controlled-rotations.simps by simp
    qed
  qed
qed

theorem QFT-is-gate:
  shows gate n (QFT n)
proof (induction n rule: QFT.induct)
  case 1
  then show ?case
   by (metis QFT.simps(1) controlled-rotations.simps(1) controlled-rotations-is-gate)
next
  case 2
  then show ?case
    using H-is-gate by auto
next
```

case (*3 v*)
then show *?case*
proof −
assume *HI*:*gate* (*Suc v*) (*QFT* (*Suc v*))
show *gate* (*Suc* (*Suc v*)) (*QFT* (*Suc* (*Suc v*)))
proof −
have *gate* (*Suc* (*Suc v*)) (((1_m *2*) $\bigotimes$ (*QFT* (*Suc v*))) ∗
(*controlled-rotations* (*Suc* (*Suc v*))) ∗ (*H* $\bigotimes$ ((1_m (*2^Suc*
v)))))
proof (*rule prod-of-gate-is-gate*)+
show *gate* (*Suc* (*Suc v*)) (1_m *2* $\bigotimes$ *QFT* (*Suc v*))
using *HI tensor-gate id-is-gate*
by (*metis One-nat-def controlled-rotations.simps*(*2*) *controlled-rotations-is-gate*

plus-1-eq-Suc)
show *gate* (*Suc* (*Suc v*)) (*controlled-rotations* (*Suc* (*Suc v*)))
using *controlled-rotations-is-gate* **by** *metis*
show *gate* (*Suc* (*Suc v*)) (*H* $\bigotimes$ 1_m (*2* ^ *Suc v*))
using *H-is-gate id-is-gate tensor-gate*
by (*metis Quantum.Id-def plus-1-eq-Suc*)
qed
thus *?thesis* **using** *QFT.simps* **by** *simp*
qed
qed
qed

corollary *QFT-is-unitary*:
shows *unitary* (*QFT n*)
using *QFT-is-gate gate-def* **by** *simp*

corollary *reverse-product-rep-is-state*:
assumes *j* < *2^n*
shows *state n* (*reverse-QFT-product-representation j n*)
using *QFT-is-gate QFT-is-correct gate-on-state-is-state assms state-basis-is-state*
by (*metis dim-col-mat*(*1*) *dim-row-mat*(*1*) *index-unit-vec*(*3*) *ket-vec-col ket-vec-def*

state-basis-def state-def unit-cpx-vec-length)

lemma *reverse-qubits-is-gate*:
shows *gate n* (*reverse-qubits n*)
proof (*induct n rule*: *reverse-qubits.induct*)
case *1*
then show *?case*
by (*metis QFT.simps*(*1*) *QFT-is-gate reverse-qubits.simps*(*1*))
next
case *2*
then show *?case*
using *Y-is-gate prod-of-gate-is-gate* **by** *fastforce*
next

case *3*
then show *?case*
using *One-nat-def SWAP-is-gate Suc-1 reverse-qubits.simps*(*3*) **by** *presburger*
next
case (*4 va*)
then show *?case*
proof −
assume *HI*:*gate* (*Suc* (*Suc va*)) (*reverse-qubits* (*Suc* (*Suc va*)))
show *gate* (*Suc* (*Suc* (*Suc va*))) (*reverse-qubits* (*Suc* (*Suc* (*Suc va*))))
proof −
have *gate* (*Suc* (*Suc* (*Suc va*))) (((*reverse-qubits* (*Suc* (*Suc va*))) $\bigotimes$ (1_m *2*))
*
(*SWAP-down* (*Suc* (*Suc* (*Suc va*)))))
proof (*rule prod-of-gate-is-gate*)
show *gate* (*Suc* (*Suc* (*Suc va*))) (*reverse-qubits* (*Suc* (*Suc va*)) $\bigotimes$ 1_m *2*)
using *HI id-is-gate tensor-gate*
by (*metis One-nat-def Suc-eq-plus1 controlled-rotations.simps*(*2*)
controlled-rotations-is-gate)
next
show *gate* (*Suc* (*Suc* (*Suc va*))) (*SWAP-down* (*Suc* (*Suc* (*Suc va*))))
using *SWAP-down-is-gate* **by** *metis*
qed
thus *?thesis* **using** *reverse-qubits.simps* **by** *simp*
qed
qed
qed

theorem *ordered-QFT-is-gate*:
shows *gate n* (*ordered-QFT n*)
using *reverse-qubits-is-gate QFT-is-gate ordered-QFT-def prod-of-gate-is-gate*
by *auto*

corollary *ordered-QFT-is-unitary*:
shows *unitary* (*ordered-QFT n*)
using *ordered-QFT-is-gate gate-def* **by** *simp*

corollary *product-rep-is-state*:
assumes $j < 2\hat{~}n$
shows *state n* (*QFT-product-representation j n*)
using *ordered-QFT-is-gate ordered-QFT-is-correct gate-on-state-is-state assms*
state-basis-is-state
by (*metis reverse-product-rep-is-state reverse-qubits-is-gate*
reverse-qubits-product-representation)

end