# Formal verification of quantum algorithms using quantum Hoare logic

Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying and Naijun Zhan

State Key Lab. of Computer Science, Institute of Software, CAS

July 18, 2019

# Why verify quantum algorithms?

- Quantum programming languages have been developing rapidly in recent years:

  $\mathcal{Q}$wire, LIQUi$|\rangle$, Q$\#$, OpenQASM, Cirq, ProjectQ, etc.

# Why verify quantum algorithms?

- Quantum programming languages have been developing rapidly in recent years:

  $\mathcal{Q}$wire, LIQUi$|\rangle$, Q$\#$, OpenQASM, Cirq, ProjectQ, etc.

- The quantum world is very unintuitive. There is a lot of potential for mistakes.

# Why verify quantum algorithms?

- Quantum programming languages have been developing rapidly in recent years:

  $\mathcal{Q}$wire, LIQUi$|\rangle$, Q$\#$, OpenQASM, Cirq, ProjectQ, etc.

- The quantum world is very unintuitive. There is a lot of potential for mistakes.
- Quantum programs are difficult to test or model-check on classical computers. On the best (classical) computers today:
  - Simulation is limited to 50-60 qubits.
  - Model-checking algorithms are limited to 25-30 qubits.

# Why verify quantum algorithms?

- Quantum programming languages have been developing rapidly in recent years:

  $\mathcal{Q}$wire, LIQUi$|\rangle$, Q#, OpenQASM, Cirq, ProjectQ, etc.

- The quantum world is very unintuitive. There is a lot of potential for mistakes.
- Quantum programs are difficult to test or model-check on classical computers. On the best (classical) computers today:
  - Simulation is limited to 50-60 qubits.
  - Model-checking algorithms are limited to 25-30 qubits.

Deductive verification can help: no need for simulation or traversing the state space.

# Classical vs. Quantum programming

| Classical | Quantum |
| --- | --- |
| Variable (bit) | Qubit |
| State (function) | Density matrix |
| Assignment | Unitary transformation |
| Conditional | Measurement |
| Assertion / predicate | Quantum predicate |
| Entailment of predicates | Löwner order |

# Classical vs. Quantum programming

| Classical | Quantum |
| --- | --- |
| Variable (bit) | Qubit |
| State (function) | Density matrix |
| Assignment | Unitary transformation |
| Conditional | Measurement |
| Assertion / predicate | Quantum predicate |
| Entailment of predicates | Löwner order |

# Quantum variables and states

- The state of each *qubit* is a vector in the *2-dimensional complex vector space*, spanned by

$$|0\rangle \text{ and } |1\rangle$$

## Quantum variables and states

- The state of each *qubit* is a vector in the *2-dimensional complex vector space*, spanned by

$$|0\rangle \text{ and } |1\rangle$$

- The state space of *two qubits* is the tensor product of the two vector spaces, spanned by

$$|00\rangle, |01\rangle, |10\rangle, \text{ and } |11\rangle$$

# Quantum variables and states

- The state of each *qubit* is a vector in the *2-dimensional complex vector space*, spanned by

$$|0\rangle \text{ and } |1\rangle$$

- The state space of *two qubits* is the tensor product of the two vector spaces, spanned by

$$|00\rangle, |01\rangle, |10\rangle, \text{ and } |11\rangle$$

- The state space for $n$ qubits has dimension $2^n$.

# Quantum variables and states

- The state of each *qubit* is a vector in the *2-dimensional complex vector space*, spanned by

$$|0\rangle \text{ and } |1\rangle$$

- The state space of *two qubits* is the tensor product of the two vector spaces, spanned by

$$|00\rangle, |01\rangle, |10\rangle, \text{ and } |11\rangle$$

- The state space for $n$ qubits has dimension $2^n$.
- A *mixed state* on $n$ qubits is given by a *density matrix* with dimension $2^n \times 2^n$.

# Classical vs. Quantum programming

| Classical | Quantum |
|---|---|
| Variable (bit) | Qubit |
| State (function) | Density matrix |
| Assignment | Unitary transformation |
| Conditional | Measurement |
| Assertion / predicate | Quantum predicate |
| Entailment of predicates | Löwner order |

# Assignment

Assignment

$$\Updownarrow$$

Multiplying the state by a unitary matrix $U$
(satisfying $U^\dagger U = \mathbb{I}$).

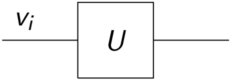
# Assignment

Assignment

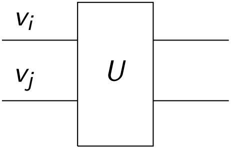
$$\Updownarrow$$

Multiplying the state by a unitary matrix $U$
(satisfying $U^\dagger U = \mathbb{I}$).

Unitary transformations may act on one or more variables:




$U$ acts on $V_i$




$U$ acts on $V_i \otimes V_j$

# Conditional

Conditional

$$\Updownarrow$$

Measurement using hermitian matrices $M_1, \ldots, M_n$ satisfying $\sum_i M_i^\dagger M_i = I_N$.

# Conditional

Conditional

$$\Updownarrow$$

Measurement using hermitian matrices $M_1, \ldots, M_n$ satisfying $\sum_i M_i^\dagger M_i = I_N$.

Measurement returns a result between 1 and $n$, and can modify the state!

# Classical vs. Quantum programming

| Classical | Quantum |
|---|---|
| Variable (bit) | Qubit |
| State (function) | Density matrix |
| Assignment | Unitary transformation |
| Conditional | Measurement |
| Assertion (predicate) | Quantum predicate |
| Entailment of predicates | Löwner order |

# Assertions

- Assertions and entailments are described using positive (semi-definite) matrices ($v^\dagger A v \geq 0$ for any $v$).

# Assertions

- Assertions and entailments are described using positive (semi-definite) matrices ($v^\dagger A v \geq 0$ for any $v$).
- The Löwner (partial) order on positive matrices is defined as:

$$A \leq_L B \iff B - A \text{ is positive.}$$

# Assertions

- Assertions and entailments are described using positive (semi-definite) matrices ($v^\dagger A v \geq 0$ for any $v$).
- The Löwner (partial) order on positive matrices is defined as:

$$A \leq_L B \iff B - A \text{ is positive.}$$

- An assertion is a positive matrix $P$ satisfying $P \leq_L \mathbb{I}_N$.

# Assertions

- Assertions and entailments are described using positive (semi-definite) matrices ($v^\dagger A v \geq 0$ for any $v$).
- The Löwner (partial) order on positive matrices is defined as:

$$A \leq_L B \iff B - A \text{ is positive.}$$

- An assertion is a positive matrix $P$ satisfying $P \leq_L \mathbb{I}_N$.
- Evalution of $P$ on density state $\rho$ is given by $\mathrm{tr}(P\rho)$. (intuition: probability that $\rho$ satisfies $P$).

# Assertions

- Assertions and entailments are described using positive (semi-definite) matrices ($v^\dagger A v \geq 0$ for any $v$).
- The Löwner (partial) order on positive matrices is defined as:

$$A \leq_L B \iff B - A \text{ is positive.}$$

- An assertion is a positive matrix $P$ satisfying $P \leq_L \mathbb{I}_N$.
- Evalution of $P$ on density state $\rho$ is given by $\mathrm{tr}(P\rho)$. (intuition: probability that $\rho$ satisfies $P$).
- Assertion $P$ entails assertion $Q$ if $P \leq_L Q$.

# Quantum Hoare logic (TOPLAS, 2011)

Syntax:

$$S ::= \textbf{skip} \mid \overline{q} = U[\overline{q}] \mid S_1; S_2 \mid \textbf{measure } M[\overline{q}] : \overline{S}$$
$$\mid \textbf{while } M[\overline{q}] = 1 \textbf{ do } S$$

# Quantum Hoare logic (TOPLAS, 2011)

Syntax:

$$S ::= \textbf{skip} \mid \overline{q} = U[\overline{q}] \mid S_1; S_2 \mid \textbf{measure}\ M[\overline{q}] : \overline{S}$$
$$\mid \textbf{while}\ M[\overline{q}] = 1\ \textbf{do}\ S$$

Semantics (mapping of density matrices):

$[\![\textbf{skip}]\!](\rho) = \rho.$
$[\![\overline{q} = U[\overline{q}]]\!](\rho) = U\rho U^\dagger.$
$[\![S_1; S_2]\!](\rho) = [\![S_2]\!]([\![S_1]\!](\rho)).$
$[\![\textbf{measure}\ M[\overline{q}] : \overline{S}]\!](\rho) = \sum_m [\![S_m]\!](M_m \rho M_m^\dagger).$
$[\![\textbf{while}\ M[\overline{q}] = 1\ \textbf{do}\ S]\!](\rho) = \sum_{k=0}^{\infty} \mathcal{E}_0 \circ ([\![S]\!] \circ \mathcal{E}_1)^k(\rho),$
where $\mathcal{E}_i(\rho) = M_i \rho M_i^\dagger$ for $i = 0, 1$.

# Quantum Hoare logic: semantic correctness

The correctness formula $\{P\}S\{Q\}$ is true in the sense of partial correctness, written

$$\models_p \{P\}S\{Q\}$$

if we have

$$\mathrm{tr}(P\rho) \leq \mathrm{tr}(Q[\![S]\!](\rho)) + [\mathrm{tr}(\rho) - \mathrm{tr}([\![S]\!](\rho))]$$

for all density operator $\rho$ in the state space of $S$.

# Quantum Hoare logic: reasoning rules

**(Skip)** $\{P\}\ \mathbf{skip}\ \{P\}$

**(UT)** $\{U^\dagger P U\}\ \overline{q} := U\overline{q}\ \{P\}$

**(Seq)** $$\frac{\{P\}\ S_1\ \{Q\} \qquad \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1; S_2\ \{R\}}$$

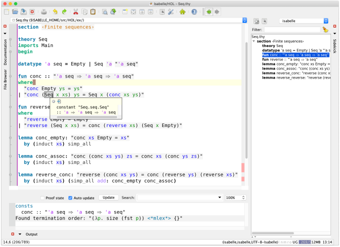
**(Mea)** $$\frac{\{P_m\}\ S_m\ \{Q\}\ \text{for all}\ m}{\{\sum_m {M_m}^\dagger P_m M_m\}\ \mathbf{measure}\ M[\overline{q}] : \overline{S}\ \{Q\}}$$

**(Loop)** $$\frac{\{Q\}\ S\ \{M_0^\dagger P M_0 + M_1^\dagger Q M_1\}}{\{M_0^\dagger P M_0 + M_1^\dagger Q M_1\}\ \mathbf{while}\ M[\overline{q}] = 1\ \mathbf{do}\ S\ \{P\}}$$

**(Order)** $$\frac{P \leq_L P' \quad \{P'\}\ S\ \{Q'\} \quad Q' \leq_L Q}{\{P\}\ S\ \{Q\}}$$

# Isabelle/HOL

- Proof assistant based on higher-order logic.
- Extensive library for analysis and linear algebra, including some material on complex matrices.



# Formalization of Quantum Hoare logic

Our work:

- Continued development of Isabelle/HOL's library in linear algebra, adding properties of positivity, hermitian and unitary matrices.

# Formalization of Quantum Hoare logic

Our work:

- Continued development of Isabelle/HOL's library in linear algebra, adding properties of positivity, hermitian and unitary matrices.
- Results about limits of matrices.

# Formalization of Quantum Hoare logic

Our work:

- Continued development of Isabelle/HOL's library in linear algebra, adding properties of positivity, hermitian and unitary matrices.
- Results about limits of matrices.
- Formally verified soundness and completeness of the deduction system (for partial correctness).

# Formalization of Quantum Hoare logic

Our work:

- Continued development of Isabelle/HOL's library in linear algebra, adding properties of positivity, hermitian and unitary matrices.
- Results about limits of matrices.
- Formally verified soundness and completeness of the deduction system (for partial correctness).
- Library for working with tensor products of vectors and matrices (for reasoning about operations on a subset of variables).

# Syntax

```
datatype com =
  SKIP
| Utrans "complex mat"
| Seq com com  ("_;;/ _" [60, 61] 60)
| Measure nat "nat ⇒ complex mat" "com list"
| While "nat ⇒ complex mat" com
```

# Semantics

```
fun denote :: "com ⇒ state ⇒ state" where
   "denote SKIP ϱ = ϱ"
 | "denote (Utrans U) ϱ = U * ϱ * adjoint U"
 | "denote (Seq S1 S2) ϱ = denote S2 (denote S1 ϱ)"
 | "denote (Measure n M S) ϱ =
      denote_measure n M (map denote S) ϱ"
 | "denote (While M S) ϱ =
       denote_while (M 0) (M 1) (denote S) ϱ"
```

# Reasoning rules

```
inductive hoare_partial :: "complex mat ⇒ com ⇒ complex mat ⇒ bool"
  ("⊢p ({(1_)}/ (_)/ {(1_)})" 50) where
  "is_quantum_predicate P ⟹ ⊢p {P} SKIP {P}"
| "is_quantum_predicate P ⟹ ⊢p {adjoint U * P * U} Utrans U {P}"
| "is_quantum_predicate P ⟹ is_quantum_predicate Q ⟹ is_quantum_predicate R ⟹
   ⊢p {P} S1 {Q} ⟹ ⊢p {Q} S2 {R} ⟹
   ⊢p {P} Seq S1 S2 {R}"
| "(⋀k. k < n ⟹ is_quantum_predicate (P k)) ⟹ is_quantum_predicate Q ⟹
   (⋀k. k < n ⟹ ⊢p {P k} S ! k {Q}) ⟹
   ⊢p {matrix_sum d (λk. adjoint (M k) * P k  * M k) n} Measure n M S {Q}"
| "is_quantum_predicate P ⟹ is_quantum_predicate Q ⟹
   ⊢p {Q} S {adjoint (M 0) * P * M 0 + adjoint (M 1) * Q * M 1} ⟹
   ⊢p {adjoint (M 0) * P * M 0 + adjoint (M 1) * Q * M 1} While M S {P}"
| "is_quantum_predicate P ⟹ is_quantum_predicate Q ⟹ is_quantum_predicate P' ⟹
   is_quantum_predicate Q' ⟹ P ≤L P' ⟹ ⊢p {P'} S {Q'} ⟹ Q' ≤L Q ⟹ ⊢p {P} S {Q}"
```

# Soundness and completeness

```
theorem hoare_partial_sound:
  "⊢p {P} S {Q} ⟹ well_com S ⟹
   ⊨p {P} S {Q}"

theorem hoare_partial_complete:
  "⊨p {P} S {Q} ⟹ well_com S ⟹
   is_quantum_predicate P ⟹
   is_quantum_predicate Q ⟹
   ⊢p {P} S {Q}"
```

# Application: Grover's algorithm

- Given a set of $N$ elements, $M$ of which satisfy $f(x) = 1$ for some boolean function $f$ (think $M \ll N$). Find one such element.

## Application: Grover's algorithm

- Given a set of $N$ elements, $M$ of which satisfy $f(x) = 1$ for some boolean function $f$ (think $M \ll N$). Find one such element.
- $f$ is given by an oracle.

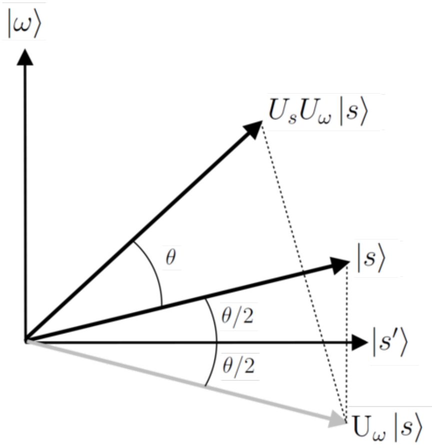
# Application: Grover's algorithm

- Given a set of $N$ elements, $M$ of which satisfy $f(x) = 1$ for some boolean function $f$ (think $M \ll N$). Find one such element.
- $f$ is given by an oracle.
- Classically, takes $N/M$ calls to the oracle on average.

# Application: Grover's algorithm

- Given a set of $N$ elements, $M$ of which satisfy $f(x) = 1$ for some boolean function $f$ (think $M \ll N$). Find one such element.
- $f$ is given by an oracle.
- Classically, takes $N/M$ calls to the oracle on average.
- Grover's algorithm finds a solution in $O(\sqrt{N/M})$ calls to the oracle.

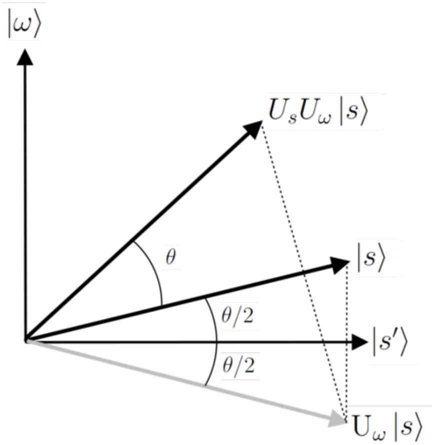
# Grover's algorithm: intuition




- $|s'\rangle$ contains *bad* elements, $|\omega\rangle$ contains *good* elements.

[1] By Danski14 - Own work, CC BY-SA 3.0, Wikipedia Commons

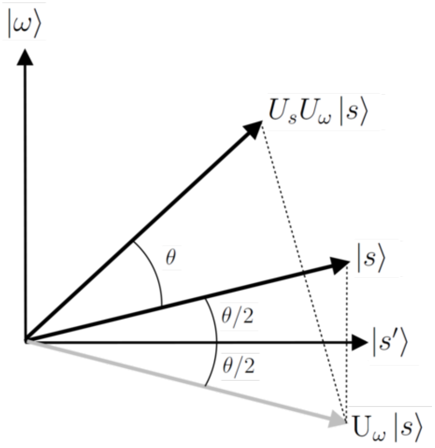
# Grover's algorithm: intuition




- $|s'\rangle$ contains *bad* elements, $|\omega\rangle$ contains *good* elements.
- Start from $|s\rangle$, a linear combination of $|s'\rangle$ and $|\omega\rangle$, closer to $|s'\rangle$.

[1] By Danski14 - Own work, CC BY-SA 3.0, Wikipedia Commons

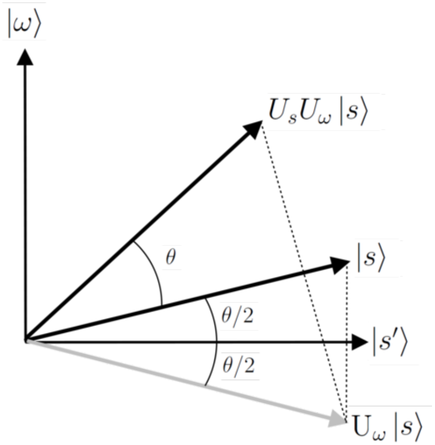
# Grover's algorithm: intuition




- $|s'\rangle$ contains *bad* elements, $|\omega\rangle$ contains *good* elements.
- Start from $|s\rangle$, a linear combination of $|s'\rangle$ and $|\omega\rangle$, closer to $|s'\rangle$.
- Each iteration rotates the state towards $|\omega\rangle$.

[1] By Danski14 - Own work, CC BY-SA 3.0, Wikipedia Commons

# Grover's algorithm: intuition




- $|s'\rangle$ contains *bad* elements, $|\omega\rangle$ contains *good* elements.
- Start from $|s\rangle$, a linear combination of $|s'\rangle$ and $|\omega\rangle$, closer to $|s'\rangle$.
- Each iteration rotates the state towards $|\omega\rangle$.
- The number of rotations is $O(\sqrt{N/M})$.

[1] By Danski14 - Own work, CC BY-SA 3.0, Wikipedia Commons

# Grover's algorithm: the verification

$$
\begin{array}{l}
q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; \\
\textbf{while } M[q_n] = 1 \textbf{ do} \\
\quad q_0 q_1 \ldots q_{n-1} := U_f[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_0 q_1 \ldots q_{n-1} := Ph[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_n := Inc[q_n]; \\
\textbf{measure } N[q_0 q_1 \ldots q_{n-1}] : \overline{\textbf{skip}}
\end{array}
$$

# Grover's algorithm: the verification

$$
\begin{array}{l}
q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; \qquad \textit{Initialization} \\
\textbf{while } M[q_n] = 1 \textbf{ do} \\
\quad q_0 q_1 \ldots q_{n-1} := U_f[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_0 q_1 \ldots q_{n-1} := Ph[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; \\
\quad q_n := Inc[q_n]; \\
\textbf{measure } N[q_0 q_1 \ldots q_{n-1}] : \overline{\textbf{skip}}
\end{array}
$$

# Grover's algorithm: the verification

$q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$ *Initialization*

**while** $M[q_n] = 1$ **do** *Compare counter*

$\quad q_0 q_1 \ldots q_{n-1} := U_f[q_0 q_1 \ldots q_{n-1}];$

$\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$

$\quad q_0 q_1 \ldots q_{n-1} := Ph[q_0 q_1 \ldots q_{n-1}];$

$\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$

$\quad q_n := Inc[q_n];$ *Increment counter*

**measure** $N[q_0 q_1 \ldots q_{n-1}] : \overline{\textbf{skip}}$

# Grover's algorithm: the verification

$$
\begin{array}{ll}
q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; & \textit{Initialization} \\
\mathbf{while}\ M[q_n] = 1\ \mathbf{do} & \textit{Compare counter} \\
\quad q_0 q_1 \ldots q_{n-1} := U_f[q_0 q_1 \ldots q_{n-1}]; & \textit{Oracle call} \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; & \\
\quad q_0 q_1 \ldots q_{n-1} := Ph[q_0 q_1 \ldots q_{n-1}]; & \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; & \\
\quad q_n := Inc[q_n]; & \textit{Increment counter} \\
\mathbf{measure}\ N[q_0 q_1 \ldots q_{n-1}] : \overline{\mathbf{skip}} &
\end{array}
$$

## Grover's algorithm: the verification

$$
\begin{array}{ll}
q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; & \textit{Initialization} \\
\textbf{while } M[q_n] = 1 \textbf{ do} & \textit{Compare counter} \\
\quad q_0 q_1 \ldots q_{n-1} := U_f[q_0 q_1 \ldots q_{n-1}]; & \textit{Oracle call} \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; & \\
\quad q_0 q_1 \ldots q_{n-1} := Ph[q_0 q_1 \ldots q_{n-1}]; & \\
\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}]; & \\
\quad q_n := Inc[q_n]; & \textit{Increment counter} \\
\textbf{measure } N[q_0 q_1 \ldots q_{n-1}] : \overline{\textbf{skip}} & \textit{Project to basis}
\end{array}
$$

# Grover's algorithm: the verification

$\{\ |0\rangle_{\overline{q}}\langle 0| \otimes |0\rangle_{q_n}\langle 0|\ \}$

$q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$ *Initialization*

**while** $M[q_n] = 1$ **do** *Compare counter*

$\quad q_0 q_1 \ldots q_{n-1} := U_f[q_0 q_1 \ldots q_{n-1}];$ *Oracle call*

$\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$

$\quad q_0 q_1 \ldots q_{n-1} := Ph[q_0 q_1 \ldots q_{n-1}];$

$\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$

$\quad q_n := Inc[q_n];$ *Increment counter*

**measure** $N[q_0 q_1 \ldots q_{n-1}]$ : $\overline{\textbf{skip}}$ *Project to basis*

$\{\ \sum_{f(x)=1} |x\rangle_{\overline{q}}\langle x| \otimes I_{q_n}\ \}$

## Grover's algorithm: the verification

$\{\ |0\rangle_{\overline{q}}\langle 0| \otimes |0\rangle_{q_n}\langle 0|\ \}$

$q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$ *Initialization*

**while** $M[q_n] = 1$ **do** *Compare counter*

$\quad \{\ \sum_{k=0}^{R} |\psi_k\rangle_{\overline{q}}\langle \psi_k| \otimes |k\rangle_{q_n}\langle k|\ \}$

$\quad q_0 q_1 \ldots q_{n-1} := U_f[q_0 q_1 \ldots q_{n-1}];$ *Oracle call*

$\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$

$\quad q_0 q_1 \ldots q_{n-1} := Ph[q_0 q_1 \ldots q_{n-1}];$

$\quad q_0 q_1 \ldots q_{n-1} := H^{\otimes n}[q_0 q_1 \ldots q_{n-1}];$

$\quad q_n := Inc[q_n];$ *Increment counter*

**measure** $N[q_0 q_1 \ldots q_{n-1}] : \overline{\textbf{skip}}$ *Project to basis*

$\{\ \sum_{f(x)=1} |x\rangle_{\overline{q}}\langle x| \otimes I_{q_n}\ \}$

# Grover's algorithm: end of proof

```
theorem grover_partial_correct:
  "⊨p
   {tensor_P pre (proj_k 0)}
    Grover
   {tensor_P post (1m K)}"
  using grover_partial_deduct well_com_Grover qp_pre qp_post
        hoare_partial_sound by auto
```

# Statistics

| Description | Files | Lines of proof |
|---|---|---|
| Preliminaries | *Complex_Matrix, ...* | 4193 |
| Semantics | *Quantum_Program* | 1110 |
| Hoare logic | *Quantum_Hoare* | 1417 |
| Tensor product | *Partial_State* | 1664 |
| Grover's algorithm | *Grover* | 3184 |
| Total | | 11568 |

# Lessons learned

- Deductive reasoning about quantum algorithms is more difficult than for classical algorithms (but quite feasible).

# Lessons learned

- Deductive reasoning about quantum algorithms is more difficult than for classical algorithms (but quite feasible).
- Automation for working with linear algebra is very helpful.

# Lessons learned

- Deductive reasoning about quantum algorithms is more difficult than for classical algorithms (but quite feasible).
- Automation for working with linear algebra is very helpful.
- Currently, can automatically prove:

$$
\begin{aligned}
\mathrm{tr}(MM^\dagger(PP^\dagger)) &= \mathrm{tr}((P^\dagger M)(P^\dagger M)^\dagger) \\
\mathrm{tr}(M_0AM_0^\dagger) + \mathrm{tr}(M_1AM_1^\dagger) &= \mathrm{tr}((M_0^\dagger M_0 + M_1^\dagger M_1)A) \\
H^\dagger(Ph^\dagger(H^\dagger Q_2 H)Ph)H &= (HPhH)^\dagger Q_2(HPhH)
\end{aligned}
$$

# Related work

- Robert Rand's implementation of $\mathcal{Q}$*wire* in Coq.

# Related work

- Robert Rand's implementation of $\mathcal{Q}$*wire* in Coq.
- Models quantum algorithms using circuits.

# Related work

- Robert Rand's implementation of *$\mathcal{Q}$wire* in Coq.
- Models quantum algorithms using circuits.
- Program verification proceeds directly from the semantics.

# Future work

- Verify more complex algorithms, including Shor's algorithm.

# Future work

- Verify more complex algorithms, including Shor's algorithm.
- Improvements to automation, which leads to more efficient verification in general:
  - Verification condition generator.
  - Automatic procedures for dealing with the verification conditions, involving positivity of matrices and tensor products.

# Future work

- Verify more complex algorithms, including Shor's algorithm.
- Improvements to automation, which leads to more efficient verification in general:
  - Verification condition generator.
  - Automatic procedures for dealing with the verification conditions, involving positivity of matrices and tensor products.
- Quantum communication protocols?