Hoare logic for verification of quantum programs

Yuan Feng

Centre for Quantum Software and Information
University of Technology Sydney

Outline

CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Outline

CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Motivation

- Programming is error-prone.
- Things will definitely be even worse when programs become larger.
- So, it is indispensable to develop techniques of verifying and debugging programs.

Testing for classical software

- Given the executable software, test if the software satisfies the specification.
- Run the software, and observe the outputs.
- However, the program must be executable, and the correctness is not guaranteed.

Formal verification

- Formal program verification is about proving properties of programs using logic systems.
- It examines source codes.
- Hoare logic for imperative programs.
    - First proposed by C. A. R. Hoare (Turing Award, 1980).
    - Original idea seeded by R. Floyd (Turing Award, 1978).

Outline

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

A simple **while**-language: Syntax

- Arithmetic expressions:

$$e ::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid \dots$$

- Boolean expressions:

$$b ::= \mathbf{true} \mid \mathbf{false} \mid e_1 \bowtie e_2 \mid \neg b \mid b_1 \wedge b_2 \mid \dots$$

- Programs:

$$\begin{aligned} S ::= \ & x := e \mid S_0; S_1 \mid \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_0\ \mathbf{fi} \\ & \mid \mathbf{while}\ b\ \mathbf{do}\ S\ \mathbf{od}. \end{aligned}$$

- Core of other practical languages.

Program states

- Program state σ: function from program variables to values (think of memory states in a computer).

$$\sigma : x \mapsto 1;\;\; y \mapsto -5;\;\; x_3 \mapsto 2000;\;\; \ldots$$

- It can be lifted to arithmetic expressions:

$$e ::= n \mid x \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid \ldots \qquad \Longrightarrow$$

$$\sigma(e) \equiv n \mid \sigma(x) \mid \sigma(e_1) + \sigma(e_2) \mid \sigma(e_1) - \sigma(e_2) \mid \sigma(e_1) * \sigma(e_2) \mid \ldots$$

and Boolean expressions similarly.

- Write $\sigma \models b$ if $\sigma(b) = \mathbf{true}$.

Semantics

- The operational semantics:

$$\langle x := e, \sigma \rangle \to \langle E, \sigma[x \mapsto \sigma(e)] \rangle \qquad \frac{\langle S_0, \sigma \rangle \to \langle S', \sigma' \rangle}{\langle S_0; S_1, \sigma \rangle \to \langle S'; S_1, \sigma' \rangle} \text{ where } E; S_1 \equiv S_1$$

$$\frac{\sigma \models \neg b}{\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ end}, \sigma \rangle \to \langle S_0, \sigma \rangle} \qquad \frac{\sigma \models b}{\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ end}, \sigma \rangle \to \langle S_1, \sigma \rangle}$$

$$\frac{\sigma \models \neg b}{\langle \textbf{while } b \textbf{ do } S \textbf{ end}, \sigma \rangle \to \langle E, \sigma \rangle} \qquad \frac{\sigma \models b}{\langle \textbf{while } b \textbf{ do } S \textbf{ end}, \sigma \rangle \to \langle S; \textbf{while } b \textbf{ do } S \textbf{ end}, \sigma \rangle}$$

- The induced denotational semantics:

$$[\![S]\!](\sigma) = \begin{cases} \sigma' & \text{if } \langle S, \sigma \rangle \to^* \langle E, \sigma' \rangle \\ \perp & \text{otherwise.} \quad \text{non-termination} \end{cases}$$

A simple example

What does the following program do?

$$
\begin{array}{l}
exp := 1; \\
i := 0; \\
\textbf{while } (i \neq n) \textbf{ do} \\
\quad exp := exp * 2; \\
\quad i := i + 1 \\
\textbf{end}
\end{array}
$$

A simple example

The program:

$$
\begin{array}{lcl}
S & \equiv & exp := 1; \\
S_1 & \equiv & i := 0; \\
S_2 & \equiv & \textbf{while } (i \neq n) \textbf{ do} \\
S_3 & \equiv & \quad exp := exp * 2 \\
S_4 & \equiv & \quad i := i + 1 \\
 & & \textbf{end}
\end{array}
$$

Computation:

$$
\begin{array}{ll}
\langle S, \sigma \rangle & \to \langle S_1, \sigma[exp \mapsto 1] \rangle \\
 & \to \langle S_2, \sigma[exp \mapsto 1, i \mapsto 0] \rangle \\
 & \to \langle S_3, \sigma[exp \mapsto 1, i \mapsto 0] \rangle \\
 & \to \langle S_4, \sigma[exp \mapsto 2, i \mapsto 0] \rangle \\
 & \to \langle S_2, \sigma[exp \mapsto 2, i \mapsto 1] \rangle \\
 & \to \ldots \\
 & \to \langle S_2, \sigma[exp \mapsto 2^n, i \mapsto n] \rangle \\
 & \to \langle E, \sigma[exp \mapsto 2^n, i \mapsto n] \rangle
\end{array}
$$

Thus

$$
[\![S]\!](\sigma) = \sigma[exp \mapsto 2^n, i \mapsto n]
$$

Assertions

- Assertion/predicate p: first-order logic formula:

$$x > 1; \quad y^2 = 2; \quad exp = 2^i$$

- Given a state σ and an assertion p,

$$\sigma \models p \quad \texttt{iff} \quad \sigma(p) = \textbf{true}$$

- Examples:

$$\sigma[x \mapsto 3] \models x > 1$$

$$\sigma \models exp = 2^i, \ \texttt{if} \ \sigma(exp) = 8 \wedge \sigma(i) = 3$$

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Program Correctness

A correctness formula or Hoare triple $\{p\}\ S\ \{q\}$ is true in the sense of

- `partial correctness`, written $\models_{par} \{p\}\ S\ \{q\}$, if for any state σ with $\sigma \models p$,

$$[\![S]\!](\sigma) \neq \bot \quad \Rightarrow \quad [\![S]\!](\sigma) \models q$$

- `total correctness`, written $\models_{tot} \{p\}\ S\ \{q\}$, if for any state σ with $\sigma \models p$,

$$[\![S]\!](\sigma) \neq \bot \quad \wedge \quad [\![S]\!](\sigma) \models q$$

Program Verification

- The task of program verification (for partial correctness) is to check if

$$\models_{par} \{p\}\ S\ \{q\}.$$

- We can do it directly from the semantics of S.
- Hoare logic does it in a syntax-oriented way by giving an axiom or inference rule for each program construct (assignment, composition, conditional branch, while loop).

Axiom for assignment

Axiom

$$\{p[e/x]\}\ \ x := e\ \ \{p\}$$

Example:

$$\{1 = 1\}\ x := 1\ \{x = 1\}$$

$$\{exp = 2^{i+1}\}\ i := i + 1\ \{exp = 2^{i}\}$$

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Rule for sequential composition

Inference Rule

$$\frac{\{p\}\ S_0\ \{p'\},\quad \{p'\}\ S_1\ \{q\}}{\{p\}\ S_0; S_1\ \{q\}}$$

Example:

$$\frac{\{exp * 2 = 2^{i+1}\}\ exp := exp * 2\ \{exp = 2^{i+1}\}, \{exp = 2^{i+1}\}\ i := i + 1\ \{exp = 2^i\}}{\{exp * 2 = 2^{i+1}\}\ exp := exp * 2;\ i := i + 1\ \{exp = 2^i\}}$$

UTS:QSI CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Rule of consequence

Inference Rule

$$\frac{p \to p', \quad \{p'\}\ S\ \{q'\}, \quad q' \to q}{\{p\}\ S\ \{q\}}$$

where both $p \to p'$ and $q' \to q$ are implications in the underlying (FO) logic. Example:

$$\frac{exp = 2^i \to exp * 2 = 2^{i+1}, \quad \{exp * 2 = 2^{i+1}\}\ exp := exp * 2;\ \ i := i + 1\ \{exp = 2^i\}}{\{exp = 2^i\}\ exp := exp * 2;\ \ i := i + 1\ \{exp = 2^i\}}$$

We call $exp = 2^i$ an invariant of $exp := exp * 2;\ i := i + 1$.

UTS:QSI CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Rule for conditional branch

Inference Rule

$$\frac{\{b \wedge p\}\ S_1\ \{q\},\quad \{\neg b \wedge p\}\ S_0\ \{q\}}{\{p\}\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_0\ \textbf{end}\ \{q\}}$$

Example:

$$\frac{\{x < y\}\ z := y\ \{z = \max\{x, y\}\},\quad \{\neg(x < y)\}\ z := x\ \{z = \max\{x, y\}\}}{\{\textbf{true}\}\ \textbf{if}\ x < y\ \textbf{then}\ z := y\ \textbf{else}\ z := x\ \textbf{end}\ \{z = \max\{x, y\}\}}$$

Rule for loops

Inference Rule

$$\frac{\{b \wedge p\} \;\; S \;\; \{p\}}{\{p\} \;\; \textbf{while} \;\; b \;\; \textbf{do} \;\; S \;\; \textbf{end} \;\; \{\neg b \wedge p\}}$$

- Here p is called loop invariant, the key to prove correctness of loop programs.
- Non-trivial task for most cases; human intervention often required. Thus semi-automatic.

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Example for loop rule

$$\frac{\{b \wedge p\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ b\ \textbf{do}\ S\ \textbf{end}\ \{\neg b \wedge p\}}$$

- We already know

$$\{exp = 2^i\}\ \ exp := exp * 2;\ \ i := i + 1\ \ \{exp = 2^i\}.$$

- Note that

$$i \neq n \wedge exp = 2^i \rightarrow exp = 2^i$$
$$\neg(i \neq n) \wedge exp = 2^i \rightarrow exp = 2^n$$

- Thus we have (with the help of consequence rule)

$$\{exp = 2^i\}\ \ \textbf{while}\ \ (i \neq n)\ \ \textbf{do}\ \ exp := exp * 2;\ \ i := i + 1;\ \ \textbf{end}\ \ \{exp = 2^n\}$$

Running example revisited

- We want to prove $\vdash_{par} \{\textbf{true}\}\ S\ \{exp = 2^n\}$.
- Starting from the postcondition:

$$
\begin{array}{l}
\{1 = 2^0\} \\
exp := 1; \\
\{exp = 2^0\} \\
i := 0; \\
\{exp = 2^i\} \\
\textbf{while}\ \ (i \neq n)\ \ \textbf{do}\ \ exp := exp * 2;\ \ i := i + 1;\ \ \textbf{end} \\
\{exp = 2^n\}
\end{array}
$$

UTS:QSI CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Soundness and Completeness

Theorem (Soundness)

For any program S and assertions p and q,

$$\vdash_{par} \{p\}S\{q\} \text{ implies } \models_{par} \{p\}S\{q\}.$$

Theorem (Completeness)

For any program S and assertions p and q,

$$\models_{par} \{p\}S\{q\} \text{ implies } \vdash_{par} \{p\}S\{q\}$$

given an oracle to decide if a formula $p \to p'$ is true in FO logic (used in Consequence rule).

Outline

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Quantum **while**-Language

- Based on [M Ying. Floyd-Hoare logic for quantum programs. ACM TOPLAS 33: 1-49, 2012].
- Recall the syntax of the classical while-language:

$$\begin{aligned} S ::= \quad & x := e \mid S_0; S_1 \mid \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ fi} \\ & \mid \textbf{while } b \textbf{ do } S \textbf{ end}. \end{aligned}$$

- Our (purely, no classical variables) quantum while-language:

$$\begin{aligned} S ::= \quad & q := |0\rangle \mid \overline{q} \mathrel{*}= U \mid S_0; S_1 \mid \textbf{if } q \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ fi} \\ & \mid \textbf{while } q \textbf{ do } S \textbf{ end} \end{aligned}$$

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Program states

- Recall: a classical state σ is a function from program variables to values.

$$\sigma : x \mapsto 1; \;\; y \mapsto -5; \;\; x_3 \mapsto 2000; \;\; \ldots$$

- A quantum state ρ of S is a partial density operator on $\mathcal{H}_{qv(S)}$ where $\mathrm{tr}(\rho) \leq 1$.

Operational Semantics

The operational semantics:

$$\langle q := |0\rangle, \rho\rangle \to \langle E, \rho_0^q\rangle \quad \text{where} \quad \rho_0^q = |0\rangle_q\langle 0| \otimes \mathrm{tr}_q(\rho)$$

$$\langle \overline{q} \mathrel{*}= U, \rho\rangle \to \langle E, U_{\overline{q}}\rho U_{\overline{q}}^{\dagger}\rangle \qquad \frac{\langle S_0, \rho\rangle \to \langle S', \rho'\rangle}{\langle S_0; S_1, \rho\rangle \to \langle S'; S_1, \rho'\rangle} \text{ where } E; S_1 \equiv S_1$$

$$\frac{\rho_0^q = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0|}{\langle \textbf{if } q \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ end}, \rho\rangle \to \langle S_0, \rho_0^q\rangle} \qquad \frac{\rho_1^q = |1\rangle_q\langle 1|\rho|1\rangle_q\langle 1|}{\langle \textbf{if } q \textbf{ then } S_1 \textbf{ else } S_0 \textbf{ end}, \rho\rangle \to \langle S_1, \rho_1^q\rangle}$$

$$\frac{\rho_0^q = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0|}{\langle \textbf{while } q \textbf{ do } S \textbf{ end}, \rho\rangle \to \langle E, \rho_0^q\rangle} \qquad \frac{\rho_1^q = |1\rangle_q\langle 1|\rho|1\rangle_q\langle 1|}{\langle \textbf{while } q \textbf{ do } S \textbf{ end}, \rho\rangle \to \langle S; \textbf{while } q \textbf{ do } S \textbf{ end}, \rho_1^q\rangle}$$

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Example: a simple quantum loop

The program:

$$
\begin{array}{lcl}
S & \equiv & q := |0\rangle; \\
S_1 & \equiv & q \mathrel{*}= H; \\
S_w & \equiv & \textbf{while } q \textbf{ do} \\
S_3 & \equiv & \qquad q \mathrel{*}= H \\
 & & \textbf{end}
\end{array}
$$

Example: a simple quantum loop

The program:

$$
\begin{aligned}
S &\equiv q := |0\rangle; \\
S_1 &\equiv q \mathrel{*}= H; \\
S_w &\equiv \textbf{while } q \textbf{ do} \\
S_3 &\equiv \qquad q \mathrel{*}= H \\
&\quad\ \textbf{end}
\end{aligned}
$$

Computations:

$$
\begin{aligned}
\langle S, \rho\rangle \quad &\to \langle S_1, \rho_0\rangle \to \langle S_w, \rho_+\rangle \\
&\to \langle E, \tfrac{1}{2}\rho_0\rangle
\end{aligned}
$$

where $\rho_x = |x\rangle_q\langle x| \otimes \mathrm{tr}_q(\rho), x \in \{0, 1, +, -\}$.

Example: a simple quantum loop

The program:

$$
\begin{array}{lcl}
S & \equiv & q := |0\rangle; \\
S_1 & \equiv & q \mathrel{*}= H; \\
S_w & \equiv & \textbf{while } q \textbf{ do} \\
S_3 & \equiv & \qquad q \mathrel{*}= H \\
 & & \textbf{end}
\end{array}
$$

Computations:

$$
\begin{array}{ll}
\langle S, \rho \rangle & \to \langle S_1, \rho_0 \rangle \to \langle S_w, \rho_+ \rangle \\
 & \to \langle S_3, \tfrac{1}{2}\rho_1 \rangle \to \langle S_w, \tfrac{1}{2}\rho_- \rangle \\
 & \to \langle E, \tfrac{1}{4}\rho_0 \rangle
\end{array}
$$

where $\rho_x = |x\rangle_q\langle x| \otimes \mathrm{tr}_q(\rho), x \in \{0, 1, +, -\}$.

Example: a simple quantum loop

The program:

$$\begin{array}{lcl} S & \equiv & q := |0\rangle; \\ S_1 & \equiv & q \mathrel{*}= H; \\ S_w & \equiv & \textbf{while } q \textbf{ do} \\ S_3 & \equiv & \qquad q \mathrel{*}= H \\ & & \textbf{end} \end{array}$$

Computations:

$$\begin{array}{ll} \langle S, \rho \rangle & \to \langle S_1, \rho_0 \rangle \to \langle S_w, \rho_+ \rangle \\ & \to \langle S_3, \frac{1}{2}\rho_1 \rangle \to \langle S_w, \frac{1}{2}\rho_- \rangle \\ & \ldots \\ & \to \langle E, \frac{1}{2^n}\rho_0 \rangle \end{array}$$

where $\rho_x = |x\rangle_q\langle x| \otimes \mathrm{tr}_q(\rho)$, $x \in \{0, 1, +, -\}$.

Example: a simple quantum loop

The program:

$$
\begin{array}{lll}
S & \equiv & q := |0\rangle; \\
S_1 & \equiv & q \mathrel{*}= H; \\
S_w & \equiv & \textbf{while } q \textbf{ do} \\
S_3 & \equiv & \qquad q \mathrel{*}= H \\
 & & \textbf{end}
\end{array}
$$

Computations:

$$
\begin{array}{ll}
\langle S, \rho\rangle & \to \langle S_1, \rho_0\rangle \to \langle S_w, \rho_+\rangle \\
 & \to \langle S_3, \frac{1}{2}\rho_1\rangle \to \langle S_w, \frac{1}{2}\rho_-\rangle \\
 & \dots \\
 & \to \langle E, \frac{1}{2^n}\rho_0\rangle
\end{array}
$$

$$
\begin{array}{ll}
\langle S, \rho\rangle & \to \langle S_1, \rho_0\rangle \to \langle S_w, \rho_+\rangle \\
 & \to \langle S_3, \frac{1}{2}\rho_1\rangle \to \langle S_w, \frac{1}{2}\rho_-\rangle \\
 & \to \langle S_3, \frac{1}{4}\rho_1\rangle \to \langle S_w, \frac{1}{4}\rho_-\rangle \\
 & \to \langle S_3, \frac{1}{8}\rho_1\rangle \to \langle S_w, \frac{1}{8}\rho_-\rangle \\
 & \dots
\end{array}
$$

where $\rho_x = |x\rangle_q\langle x| \otimes \mathrm{tr}_q(\rho)$, $x \in \{0, 1, +, -\}$.

Example: a simple quantum loop

The program:

$$
\begin{array}{lcl}
S & \equiv & q := |0\rangle; \\
S_1 & \equiv & q \mathrel{*}= H; \\
S_w & \equiv & \textbf{while } q \textbf{ do} \\
S_3 & \equiv & \qquad q \mathrel{*}= H \\
 & & \textbf{end}
\end{array}
$$

Computations:

$$
\begin{aligned}
\langle S, \rho\rangle \quad & \to \langle S_1, \rho_0\rangle \to \langle S_w, \rho_+\rangle \\
& \to \langle S_3, \tfrac{1}{2}\rho_1\rangle \to \langle S_w, \tfrac{1}{2}\rho_-\rangle \\
& \ldots \\
& \to \langle E, \tfrac{1}{2^n}\rho_0\rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle S, \rho\rangle \quad & \to \langle S_1, \rho_0\rangle \to \langle S_w, \rho_+\rangle \\
& \to \langle S_3, \tfrac{1}{2}\rho_1\rangle \to \langle S_w, \tfrac{1}{2}\rho_-\rangle \\
& \to \langle S_3, \tfrac{1}{4}\rho_1\rangle \to \langle S_w, \tfrac{1}{4}\rho_-\rangle \\
& \to \langle S_3, \tfrac{1}{8}\rho_1\rangle \to \langle S_w, \tfrac{1}{8}\rho_-\rangle \\
& \cdots \ (\texttt{divergence!})
\end{aligned}
$$

where $\rho_x = |x\rangle_q\langle x| \otimes \mathrm{tr}_q(\rho)$, $x \in \{0, 1, +, -\}$.

Semantic Function

- The denotational semantics of a quantum program S: for all $\rho \in \mathcal{D}(\mathcal{H}_{qv(S)})$,

$$[\![S]\!](\rho) = \sum \{\!|\ \rho' : \langle S, \rho \rangle \rightarrow^* \langle E, \rho' \rangle\ |\!\}.$$

- This is well-defined since it can be shown that

$$\mathrm{tr}([\![S]\!](\rho)) \leq 1.$$

Example: a simple quantum loop (Cont.)

Operational semantics:

for any $\rho \in \mathcal{D}(\mathcal{H}_{qv(S)})$ and any $n \geq 1$,

$$\begin{aligned}\langle S, \rho\rangle \quad & \rightarrow \langle S_1, \rho_0\rangle \rightarrow \langle S_w, \rho_+\rangle \\ & \rightarrow \langle S_3, \tfrac{1}{2}\rho_1\rangle \rightarrow \langle S_w, \tfrac{1}{2}\rho_-\rangle \\ & \rightarrow \langle S_3, \tfrac{1}{4}\rho_1\rangle \rightarrow \langle S_w, \tfrac{1}{4}\rho_-\rangle \\ & \rightarrow \cdots \\ & \rightarrow \langle E, \tfrac{1}{2^n}\rho_0\rangle\end{aligned}$$

Denotational semantics:

for any $\rho \in \mathcal{D}(\mathcal{H}_{qv(S)})$,

$$\begin{aligned}[\![S]\!](\rho) \quad & = \sum_{n=1}^{\infty} \frac{1}{2^n}\rho_0 \\ & = \rho_0.\end{aligned}$$

CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Assertions

- A quantum assertion M of S is a Hermitian operator on $\mathcal{H}_{qv(S)}$ with all its eigenvalues lying within the unit interval $[0,1]$.
- Recall that for a classical state σ and assertion p,

$$\sigma \models p \quad \texttt{iff} \quad p(\sigma) = \mathbf{true}$$

- Given a quantum state ρ and a quantum assertion M,

$$\mathrm{Exp}(\rho \models M) = \mathrm{tr}(M\rho)$$

denotes the degree to which ρ satisfies M.

Total Correctness

- The correctness formula $\{P\}S\{Q\}$ is true in the sense of total correctness, written $\models_{tot} \{P\}S\{Q\}$, if for all $\rho \in \mathcal{D}(\mathcal{H}_{qv(S)})$:

$$\mathrm{Exp}(\rho \models P) \;\leq\; \mathrm{Exp}([\![S]\!](\rho) \models Q)$$

- Compare: in classical case,

$$\sigma \models p \quad \Rightarrow \quad [\![S]\!](\sigma) \models q$$

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Partial Correctness

- The correctness formula $\{P\}S\{Q\}$ is true in the sense of partial correctness, written $\models_{par} \{P\}S\{Q\}$, if for all $\rho \in \mathcal{D}(\mathcal{H}_{qv(S)})$:

$$\mathrm{Exp}(\rho \models P) \quad \leq \quad \mathrm{Exp}(\llbracket S \rrbracket(\rho) \models Q) + [\mathrm{tr}(\rho) - \mathrm{tr}(\llbracket S \rrbracket(\rho))].$$

- Compare: in classical case,

$$\sigma \models p \quad \Rightarrow \quad \llbracket S \rrbracket(\sigma) \models q \;\vee\; \llbracket S \rrbracket(\sigma) = \bot$$

Axiom for initialisation

Axiom

$$\{I_q \otimes \langle 0|P|0\rangle_q\}\ q := |0\rangle\ \{P\}$$

Example:

$$\{I_q\}\ q := |0\rangle\ \{|0\rangle_q\langle 0|\}$$
$$\{0\}\ q := |0\rangle\ \{|1\rangle_q\langle 1|\}$$
$$\{|\alpha_0|^2\}\ q := |0\rangle\ \{[\alpha_0|0\rangle + \alpha_1|1\rangle]_q\}$$

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Axiom for unitary application

Axiom

$$\{U_{\bar{q}}^{\dagger} P U_{\bar{q}}\}\ \bar{q} *= U\ \{P\}$$

Example:

$$\{|+\rangle_q\langle+|\}\ q *= H\ \{|0\rangle_q\langle 0|\}$$
$$\{I\}\ q *= H\ \{I\}$$

Rule for sequential composition

Inference Rule

$$\frac{\{P\}\ \ S_0\ \ \{P'\},\ \ \{P'\}\ \ S_1\ \ \{Q\}}{\{P\}\ \ S_0; S_1\ \ \{Q\}}$$

Example:

$$\frac{\{I\}\ q := |0\rangle\ \{I\},\ \{I\}\ q \mathrel{*}= H\ \{I\}}{\{I\}\ q := |0\rangle;\ q \mathrel{*}= H\ \{I\}}$$

Rule of consequence

Inference Rule

$$\frac{P \sqsubseteq P', \ \ \{P'\} \ \ S \ \ \{Q'\}, \ \ Q' \sqsubseteq Q}{\{P\} \ \ S \ \ \{Q\}}$$

Here $P \sqsubseteq P'$ if $P' - P$ is positive. Example:

$$\frac{|0\rangle_q\langle 0| \sqsubseteq I, \ \{I\} \ q := |0\rangle \ \{I\}}{\{|0\rangle_q\langle 0|\} \ q := |0\rangle \ \{I\}}$$

Rule for conditional branch

Inference Rule

$$\frac{\{P_1\}\ S_1\ \{Q\},\ \ \{P_0\}\ S_0\ \{Q\}}{\{R\}\ \textbf{if}\ q\ \textbf{then}\ S_1\ \textbf{else}\ S_0\ \textbf{end}\ \{Q\}}$$

where $R = |0\rangle_q\langle 0|P_0|0\rangle_q\langle 0| + |1\rangle_q\langle 1|P_1|1\rangle_q\langle 1|$.

Compare with the classical rule:

$$\frac{\{b \wedge p\}\ S_1\ \{q\},\ \{\neg b \wedge p\}\ S_0\ \{q\}}{\{p\}\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_0\ \textbf{end}\ \{q\}}$$

CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Rule for while loops

Inference Rule

$$\frac{\{Q\}\ \ S\ \ \{R\}}{\{R\}\ \ \textbf{while}\ \ q\ \ \textbf{do}\ \ S\ \ \textbf{end}\ \ \{P\}}$$

where $R = |0\rangle_q\langle 0|P|0\rangle_q\langle 0| + |1\rangle_q\langle 1|Q|1\rangle_q\langle 1|$.

Here R is called loop invariant. Compare with the classical rule:

$$\frac{\{b \wedge p\}\ S\ \{p\}}{\{p\}\ \textbf{while}\ b\ \textbf{do}\ S\ \textbf{end}\ \{\neg b \wedge p\}}$$

Soundness and Completeness [Ying 2012]

Theorem (Soundness)

*For any quantum **while**-program S and quantum predicates P, Q,*

$$\vdash_{par} \{P\}S\{Q\} \text{ implies } \models_{par} \{P\}S\{Q\}.$$

Theorem (Completeness)

*For any quantum **while**-program S and quantum predicates P, Q,*

$$\models_{par} \{P\}S\{Q\} \text{ implies } \vdash_{par} \{P\}S\{Q\}$$

given an oracle to decide if a formula $P \sqsubseteq P'$ is true in linear algebra (used in Consequence rule).

Refs. on Quantum Hoare Logic

- Purely quantum programs:
  1. E D'Hondt and P Panangaden. Quantum weakest preconditions. MSCS 16:3, 429-451, 2006.
  2. Y Feng, R Duan, Z Ji, M Ying. Proof rules for the correctness of quantum programs, TCS 386:151-166,2007.
  3. M Ying. Floyd-Hoare logic for quantum programs. ACM TOPLAS 33: 1-49, 2012. 1st sound & complete system
  4. L Zhou, N Yu, M Ying. An applied quantum Hoare logic. PLDI'19: 1149-1162.
  5. M Ying, L Zhou, Y Li, Y Feng. A proof system for disjoint parallel quantum programs. TCS 897:164-184, 2022.
  6. Y Feng and Y Xu. Verification of nondeterministic quantum programs. Accepted by ASPLOS'23.

UTS:QSI CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Refs. on Quantum Hoare Logic

- Classical-quantum programs:
  1. Y Feng and M Ying. Quantum Hoare logic with classical variables. ACM TQC 2(4), 16:1-16:43, 2021.
  2. Y Feng, S Li, M Ying. Verification of Distributed Quantum Programs. ACM TCL 23(3), 19:1-19:40, 2022.

UTS:QSI
CENTRE FOR QUANTUM SOFTWARE AND INFORMATION

Refs. on Quantum Hoare Logic

- Quantum relational Hoare logic:
  1. D Unruh. Quantum relational Hoare logic. POPL'19, 1-31.
  2. G Barthe, J Hsu, M Ying, N Yu, L Zhou. Relational proofs for quantum programs. POPL'20: 21:1-21:29.
- Tool implementation:
  1. J Liu, B Zhan, S Wang, S Ying, T Liu, Y Li, M Ying, N Zhan. Formal Verification of Quantum Algorithms Using Quantum Hoare Logic. CAV'19: 187-207.
  2. L Zhou, G Barthe, P Strub, J Liu, M Ying. CoqQ: Foundational Verification of Quantum Programs. POPL'23: 833-865.