Foundations of Quantum Programming
(Lecture 5)

Mingsheng Ying

Centre for Quantum Software and Information
University of Technology Sydney

Outline

Outline

Elegance in programming

- Programming research pioneers (Dijkstra, Hoare, ...) had persuaded a high level of elegance in programming by introducing effective program constructs and programming schemes.

Elegance in programming

- Programming research pioneers (Dijkstra, Hoare, ...) had persuaded a high level of elegance in programming by introducing effective program constructs and programming schemes.

Example

Euclid's algorithm for computing gcd in Dijkstra's GCL:

$$\begin{aligned}&\textbf{do } x > y \to x := x - y\\ &\Box\ \ x < y \to y := y - x\\ &\textbf{od}\end{aligned}$$

Elegance in programming

- Programming research pioneers (Dijkstra, Hoare, ...) had persuaded a high level of elegance in programming by introducing effective program constructs and programming schemes.

Example

Euclid's algorithm for computing gcd in Dijkstra's GCL:

$$\begin{aligned}&\textbf{do } x > y \to x := x - y\\ &\square\ x < y \to y := y - x\\ &\textbf{od}\end{aligned}$$

- How can achieve the same level of elegance in quantum programming?

Recursive programming

- Procedure and recursion constructs are widely used in classical programming for modularity.

Recursive programming

- Procedure and recursion constructs are widely used in classical programming for modularity.
- In particular, an invocation of a (recursive) procedure enables a (unbounded) number of uses of a uniquely defined function or program section.

Recursive programming

- Procedure and recursion constructs are widely used in classical programming for modularity.
- In particular, an invocation of a (recursive) procedure enables a (unbounded) number of uses of a uniquely defined function or program section.

Example

Hoare's Quicksort — divide-and-conquer.

Recursive programming

- Procedure and recursion constructs are widely used in classical programming for modularity.
- In particular, an invocation of a (recursive) procedure enables a (unbounded) number of uses of a uniquely defined function or program section.

Example

Hoare's Quicksort — divide-and-conquer.

- How to support the same level of abstraction in quantum programming?

Recursion in quantum programming

- Recursive quantum programs with classical control

Recursion in quantum programming

- Recursive quantum programs with classical control
- Quantum recursive programs with quantum control

Motivating Example — Quantum Circuit of QFT

$$
\begin{aligned}
&QFT^{(n)}[q_1, ..., q_n] \\
&\quad ::= H[q_1]; C(R_2)[q_2, q_1]; C(R_3)[q_3, q_1]; ...; C(R_{n-1})[q_{n-1}, q_1]; C(R_n)[q_n, q_1]; \\
&\qquad H[q_2]; C(R_2)[q_3, q_2]; C(R_3)[q_4, q_2]; ...; C(R_{n-1})[q_n, q_2]; \\
&\qquad\qquad\qquad\qquad\qquad\qquad ..... \\
&\qquad H[q_{n-1}]; C(R_2)[q_n, q_{n-1}]; \\
&\qquad H[q_n]; \\
&\qquad Reverse[q_1, ..., q_n]
\end{aligned}
$$

Motivating Example — Recursive Definition of QFT

$$QFT(m,n) \Leftarrow \textbf{if } m = n \textbf{ then } S_{0,0}[q[m]];$$
$$\textbf{else qif}[q[m+1:n]]\ (\square|k\rangle \rightarrow S_{n-m,k}[q[m]])\ \textbf{fiq};$$
$$QFT(m+1,n); Shift(m,n)$$
$$\textbf{fi},$$

$$Shift(m,n) \Leftarrow \textbf{if } m < n \textbf{ then}$$
$$Swap[q[m],q[n]]; Shift(m+1,n)$$
$$\textbf{fi}.$$

Outline

Quantum Types

- A basic quantum type $\mathcal{H}$ denotes an intended Hilbert space.

Quantum Types

- A basic quantum type $\mathcal{H}$ denotes an intended Hilbert space.
- A higher quantum type is defined of the form:

$$T_1 \times ... \times T_n \to \mathcal{H}$$

where $T_1, ..., T_n$ are basic classical types, and $\mathcal{H}$ is a basic quantum type.

$$\mathcal{H}^{\otimes(T_1 \times ... \times T_n)} = \bigotimes_{t_1 \in T_1, ..., t_n \in T_n} \mathcal{H}$$

Quantum Types

- A basic quantum type $\mathcal{H}$ denotes an intended Hilbert space.
- A higher quantum type is defined of the form:

$$T_1 \times ... \times T_n \to \mathcal{H}$$

where $T_1, ..., T_n$ are basic classical types, and $\mathcal{H}$ is a basic quantum type.

$$\mathcal{H}^{\otimes(T_1 \times ... \times T_n)} = \bigotimes_{t_1 \in T_1, ..., t_n \in T_n} \mathcal{H}$$

- Example: A qubit array q of type **integer** $\to \mathcal{H}_2$
  Section $q[k : l]$ stands for the restriction of q to the interval $[k : l] = \{\text{integer } i \mid k \leq i \leq l\}$.

Quantum Variables

- Simple quantum variables, of a basic quantum type, say $\mathcal{H}$;

Quantum Variables

- Simple quantum variables, of a basic quantum type, say $\mathcal{H}$;
- Array quantum variables, of a higher quantum type, say $T_1 \times ... \times T_n \to \mathcal{H}$.

Quantum Variables

- Simple quantum variables, of a basic quantum type, say $\mathcal{H}$;
- Array quantum variables, of a higher quantum type, say $T_1 \times ... \times T_n \to \mathcal{H}$.
- Let q be an array quantum variable of the type $T_1 \times ... \times T_n \to \mathcal{H}$, and for each $1 \leq i \leq n$, let s_i be a classical expression of type T_i. Then

  $$q[s_1, ..., s_n]$$

  is a subscripted quantum variable of type $\mathcal{H}$.

Quantum Variables

- Simple quantum variables, of a basic quantum type, say $\mathcal{H}$;
- Array quantum variables, of a higher quantum type, say $T_1 \times ... \times T_n \to \mathcal{H}$.
- Let q be an array quantum variable of the type $T_1 \times ... \times T_n \to \mathcal{H}$, and for each $1 \leq i \leq n$, let s_i be a classical expression of type T_i. Then

  $$q[s_1, ..., s_n]$$

  is a subscripted quantum variable of type $\mathcal{H}$.
- Example: If q : **integer** $\times$ **integer** $\to \mathcal{H}_2$ is a two-dimensional array of qubits, and x, y are integer variables, then

  $$q[2x - y, 3x + 4y - 75]$$

  is a subscripted qubit variable.

Outline

Alphabet of **QC**

- A set QV of simple or subscripted quantum variables;

Alphabet of **QC**

- A set QV of simple or subscripted quantum variables;
- A set $\mathcal{U}$ of unitary matrix constants.
  e.g. $\mathcal{U} = U(2)$ of all single qubit gates, or $\mathcal{U} = \{H, T\}$.

Syntax of **QC**

$$C ::= U[\bar{q}] \mid C_1; C_2 \mid \mathbf{qif}[q]\,(|0\rangle \to C_0)\,\square\,(|1\rangle \to C_1)\,\mathbf{fiq}$$

Alphabet of **QC**

- A set QV of simple or subscripted quantum variables;
- A set $\mathcal{U}$ of unitary matrix constants.
  e.g. $\mathcal{U} = U(2)$ of all single qubit gates, or $\mathcal{U} = \{H, T\}$.

Syntax of **QC**

$$C ::= U[\bar{q}] \mid C_1; C_2 \mid \mathbf{qif}[q]\,(|0\rangle \to C_0) \square (|1\rangle \to C_1)\,\mathbf{fiq}$$

- Quantum if-statement: For $q \notin qv(C_0) \cup qv(C_1)$,

$$C \equiv \mathbf{qif}[q]\,(|0\rangle \to C_0) \square (|1\rangle \to C_1)\,\mathbf{fiq}$$

If q is in state $|0\rangle$, then C behaves like C_0; if q is in state $|1\rangle$, then C behaves like C_1. The difference between quantum and classical if-statements: quantum coin q can be in a superposition of $|0\rangle$ and $|1\rangle$, and thus it defines a quantum control flows.

Operational Semantics of **QC**

$$\text{(GA)}\quad (U[\bar{q}], |\psi\rangle) \to \left(\downarrow, (U \otimes I_{var(|\psi\rangle)\setminus\bar{q}})|\psi\rangle\right)$$

$$\text{(SC)}\quad \frac{(C_1, |\psi\rangle) \to (C_1', |\psi'\rangle)}{(C_1; C_2, |\psi\rangle) \to (C_1'; C_2, |\psi'\rangle)}$$

$$\text{(QIF)}\quad \frac{|\psi\rangle = \alpha_0|0\rangle_q|\theta_0\rangle + \alpha_1|1\rangle_q|\theta_1\rangle \qquad (C_i, |\theta_i\rangle) \to^* (\downarrow, |\theta_i'\rangle)\ (i = 0, 1)}{(\mathbf{qif}[q]\,(|0\rangle \to C_0) \square (|1\rangle \to C_1)\,\mathbf{fiq}, |\psi\rangle) \to (\downarrow, \alpha_0|0\rangle_q|\theta_0'\rangle + \alpha_1|1\rangle_q|\theta_1'\rangle)}$$

Operational Semantics of **QC**

$$\text{(GA)}\quad (U[\bar{q}], |\psi\rangle) \to \left(\downarrow, (U \otimes I_{var(|\psi\rangle)\setminus\bar{q}})|\psi\rangle\right)$$

$$\text{(SC)}\quad \frac{(C_1, |\psi\rangle) \to (C_1', |\psi'\rangle)}{(C_1; C_2, |\psi\rangle) \to (C_1'; C_2, |\psi'\rangle)}$$

$$\text{(QIF)}\quad \frac{|\psi\rangle = \alpha_0|0\rangle_q|\theta_0\rangle + \alpha_1|1\rangle_q|\theta_1\rangle \qquad (C_i, |\theta_i\rangle) \to^* (\downarrow, |\theta_i'\rangle)\ (i = 0, 1)}{(\mathbf{qif}[q]\,(|0\rangle \to C_0) \square (|1\rangle \to C_1)\,\mathbf{fiq}, |\psi\rangle) \to (\downarrow, \alpha_0|0\rangle_q|\theta_0'\rangle + \alpha_1|1\rangle_q|\theta_1'\rangle)}$$

Denotational Semantics of **QC**

$$[\![\mathbf{qif}[q]\,(|0\rangle \to C_0) \square (|1\rangle \to C_1)\,\mathbf{fiq}]\!] = |0\rangle\langle 0| \otimes [\![C_0]\!] + |1\rangle\langle 1| \otimes [\![C_1]\!]$$

quantum multiplexor

$$= \begin{pmatrix} [\![C_0]\!] & 0 \\ 0 & [\![C_1]\!] \end{pmatrix}.$$

Outline

Syntax of $\mathbf{QC}^+$

$\mathbf{QC}^+ =$ embed $\mathbf{QC}$ into a classical programming language

$$
\begin{aligned}
C ::= \ & \mathbf{skip} \mid \overline{x} := \overline{t} \mid U[\overline{q}] \mid C_1; C_2 \\
& \mid \mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \mathbf{fi} \\
& \mid \mathbf{qif}[q]\ (|0\rangle \to C_0) \square (|1\rangle \to C_1)\ \mathbf{fiq}
\end{aligned}
$$

Operational Semantics of of $\mathbf{QC}^+$

(SK) $(\mathbf{skip}, \sigma, |\psi\rangle) \to (\downarrow, \sigma, |\psi\rangle)$

(AS) $(\overline{x} := \overline{t}, \sigma, |\psi\rangle) \to (\downarrow, \sigma[\overline{x} := \sigma(\overline{t})], |\psi\rangle)$

(GA) $(U[\overline{q}], \sigma, |\psi\rangle) \to (\downarrow, \sigma, (U \otimes I_{var(|\psi\rangle)\setminus\overline{q}})|\psi\rangle)$

(SC) $$\frac{(C_1, \sigma, |\psi\rangle) \to (C_1', \sigma', |\psi'\rangle)}{(C_1; C_2, \sigma, |\psi\rangle) \to (C_1'; C_2, \sigma', |\psi'\rangle)}$$

(IF) $$\frac{\sigma \models b}{(\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \mathbf{fi}, \sigma, |\varphi\rangle) \to (C_1, \sigma, |\varphi\rangle)}$$

$$\frac{\sigma \models \neg b}{(\mathbf{if}\ b\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \mathbf{fi}, \sigma, |\varphi\rangle) \to (C_2, \sigma, |\varphi\rangle)}$$

(QIF) $$\frac{|\psi\rangle = \alpha_0|0\rangle_q|\theta_0\rangle + \alpha_1|1\rangle_q|\theta_1\rangle \quad (C_i, \sigma, |\theta_i\rangle) \to^* (\downarrow, \sigma', |\theta_i'\rangle)\ (i = 0, 1)}{(\mathbf{qif}[q]\ (|0\rangle \to C_0) \square (|1\rangle \to C_1)\ \mathbf{fiq}, \sigma, |\psi\rangle) \to \left(\downarrow, \sigma', |\psi\rangle = \alpha_0|0\rangle_q|\theta_0'\rangle + \alpha_1|1\rangle_q|\theta_1'\rangle\right)}$$

Remarks

- In a configuration $(C, \sigma, |\psi\rangle)$, classical state σ and quantum $|\psi\rangle$ are related to each other: $|\psi\rangle$ may contain certain subscripted quantum variables of the form $q[s_1, ..., s_n]$, where $s_1, ..., s_n$ are classical expressions.

Remarks

- In a configuration $(C, \sigma, |\psi\rangle)$, classical state σ and quantum $|\psi\rangle$ are related to each other: $|\psi\rangle$ may contain certain subscripted quantum variables of the form $q[s_1, ..., s_n]$, where $s_1, ..., s_n$ are classical expressions.
- In the premise of rule (QIF), the executions of two branches $(C_i, \sigma, |\theta_i\rangle)$ $(i = 0, 1)$ are required to terminate in the same classical state σ'. This requirement seems hard to meet in practical applications. Indeed, it can be easily achieved using local variables.

Local variables

- Block statement:

$$C ::= \textbf{begin local } \overline{x} := \overline{t}; C \textbf{ end}$$

Local variables

- Block statement:

$$C ::= \textbf{begin local } \overline{x} := \overline{t}; C \textbf{ end}$$

- Operational semantics:

$$(\text{BS})\ \ (\textbf{begin local } \overline{x} := \overline{t}; C \textbf{ end}, \sigma, |\psi\rangle) \to (\overline{x} := \overline{t}; C; \overline{x} := \sigma(\overline{x}), \sigma, |\psi\rangle)$$

Outline

Syntax of $\mathbf{RQC}^+$

- Alphabet of $\mathbf{RQC}^+$ = Alphabet of $\mathbf{QC}^+$ + procedure identifiers $P, P_1, P_2, \ldots$

Syntax of $\mathbf{RQC}^+$

- Alphabet of $\mathbf{RQC}^+$ = Alphabet of $\mathbf{QC}^+$ + procedure identifiers $P, P_1, P_2, \ldots$
- Syntax:

$$C ::= \cdots\cdots \mid P.$$

Semantics of $\mathbf{RQC}^+$

Syntax of $\mathbf{RQC}^+$

- Alphabet of $\mathbf{RQC}^+$ = Alphabet of $\mathbf{QC}^+$ + procedure identifiers $P, P_1, P_2, \ldots$
- Syntax:

$$C ::= \cdots\cdots \mid P.$$

- Declarations: each procedure identifier P is defined by a *declaration*

$$P \Leftarrow C$$

where $C \in \mathbf{RQC}^+$ is called the procedure body.
We assume a fixed set $\mathcal{D}$ of procedure declarations.

Semantics of $\mathbf{RQC}^+$

Syntax of $\mathbf{RQC}^+$

- Alphabet of $\mathbf{RQC}^+$ = Alphabet of $\mathbf{QC}^+$ + procedure identifiers $P, P_1, P_2, \ldots$
- Syntax:

$$C ::= \cdots\cdots \mid P.$$

- Declarations: each procedure identifier P is defined by a *declaration*

$$P \Leftarrow C$$

where $C \in \mathbf{RQC}^+$ is called the procedure body.
We assume a fixed set $\mathcal{D}$ of procedure declarations.

Semantics of $\mathbf{RQC}^+$

- Copy Rule: a procedure call is dynamically replaced by the procedure body

$$(\mathrm{CR}) \qquad \frac{P \Leftarrow C \in \mathcal{D}}{(P, \sigma, |\psi\rangle) \to (C, \sigma, |\psi\rangle)}$$

Example

$$C^{(n)}(U)|i_1,...,i_n\rangle|\psi\rangle = \begin{cases} |i_1,...,i_n\rangle U|\psi\rangle & \text{if } i_1 = ... = i_n = 1; \\ |i_1,...,i_n\rangle|\psi\rangle & \text{otherwise} \end{cases}$$

Let q be a qubit array of type **integer** $\to \mathcal{H}_2$. Then the controlled-U gate on the section $q[first : last]$ with the first $(last - first)$ qubits as its control qubits and the last one as its target qubit:

$C^{(*)}(U) \Leftarrow$
if $first = last$
  then $U[q[last]]$
  else qif$[q[first]]|0\rangle \to$ **skip**
    $\square\ |1\rangle \to$ **begin local** $first := first + 1;$
      $C^{(*)}(U)$ **end**
  fiq
fi

Outline

Syntax of $\mathbf{RQC}^{++}$

- Alphabet of $\mathbf{RQC}^{++}$ = Alphabet of $\mathbf{QC}^{+}$ + procedure identifiers $P, P_1, P_2, \ldots$ (each procedure identifier P is given an arity $ar(P)$)

Syntax of $\mathbf{RQC}^{++}$

- Alphabet of $\mathbf{RQC}^{++}$ = Alphabet of $\mathbf{QC}^{+}$ + procedure identifiers $P, P_1, P_2, \ldots$ (each procedure identifier P is given an arity $ar(P)$)
- Syntax:

$$C ::= \cdots\cdots \mid P(\bar{t}).$$

Syntax of $\mathbf{RQC}^{++}$

- Alphabet of $\mathbf{RQC}^{++}$ = Alphabet of $\mathbf{QC}^{+}$ + procedure identifiers $P, P_1, P_2, \ldots$ (each procedure identifier P is given an arity $ar(P)$)
- Syntax:

$$C ::= \cdots\cdots \mid P(\bar{t}).$$

- Declarations: each procedure identifier P is defined by a *declaration*

$$P(\bar{u}) \Leftarrow C$$

where $C \in \mathbf{RQC}^{+}$ is called the procedure body.
We assume a fixed set $\mathcal{D}$ of procedure declarations.

Semantics of $\mathbf{RQC}^{++}$

- Copy rule + Formal parameters $\overline{u}$ are initialised by actual parameters $\overline{t}$ in a block statement:

$$(\mathrm{RC})\quad \frac{P(\overline{u}) \Leftarrow C \in \mathcal{D}}{(P(\overline{t}), \sigma, |\psi\rangle) \to (\textbf{begin local } \overline{u} := \overline{t}; C \textbf{ end}, \sigma, |\psi\rangle)}$$

Example - Controlled Gates

- The controlled-U gate on the section $q[m:n]$ with the first $(n-m)$ qubits as its control qubits and the last one as its target qubit:

$$
\begin{aligned}
C^{(*)}(U)(m,n) \Leftarrow\ & \textbf{if } m = n \\
& \quad \textbf{then } U[q[n]] \\
& \quad \textbf{else qif}[q[m]]|0\rangle \rightarrow \textbf{skip} \\
& \qquad\qquad \square \quad |1\rangle \rightarrow C^{(*)}(U)[m+1,n] \\
& \quad \textbf{fiq} \\
& \textbf{fi}
\end{aligned}
$$

Example - Controlled Gates

- The controlled-U gate on the section $q[m:n]$ with the first $(n-m)$ qubits as its control qubits and the last one as its target qubit:

$$
\begin{aligned}
C^{(*)}(U)(m,n) \Leftarrow & \textbf{ if } m = n \\
& \quad \textbf{then } U[q[n]] \\
& \quad \textbf{else qif} [q[m]]|0\rangle \rightarrow \textbf{skip} \\
& \qquad\qquad \square \;\; |1\rangle \rightarrow C^{(*)}(U)[m+1,n] \\
& \quad \textbf{fiq} \\
& \textbf{fi}
\end{aligned}
$$

- Decrease and Conquer!

Example - QRAMs

- A simple QRAM performs the transformation: for any data set $D[0:N]$ with $N = 2^n - 1$, and for any address $0 \leq j \leq N$,

$$|j\rangle|D[0:N]\rangle \mapsto |j\rangle|D[j]\rangle|D[0:j-1]\rangle|D[0]\rangle|D[j+1:N]\rangle.$$

Example - QRAMs

- A simple QRAM performs the transformation: for any data set $D[0:N]$ with $N = 2^n - 1$, and for any address $0 \leq j \leq N$,

$$|j\rangle|D[0:N]\rangle \mapsto |j\rangle|D[j]\rangle|D[0:j-1]\rangle|D[0]\rangle|D[j+1:N]\rangle.$$

- Intuition: given an address j, the desired data element $D[j]$ is swapped out.

Example - QRAMs

- A simple QRAM performs the transformation: for any data set $D[0:N]$ with $N = 2^n - 1$, and for any address $0 \leq j \leq N$,

$$|j\rangle|D[0:N]\rangle \mapsto |j\rangle|D[j]\rangle|D[0:j-1]\rangle|D[0]\rangle|D[j+1:N]\rangle.$$

- Intuition: given an address j, the desired data element $D[j]$ is swapped out.
- A simple (but not very efficient) implementation of QRAM:

Example - QRAMs

- A simple QRAM performs the transformation: for any data set $D[0:N]$ with $N = 2^n - 1$, and for any address $0 \leq j \leq N$,

$$|j\rangle|D[0:N]\rangle \mapsto |j\rangle|D[j]\rangle|D[0:j-1]\rangle|D[0]\rangle|D[j+1:N]\rangle.$$

- Intuition: given an address j, the desired data element $D[j]$ is swapped out.
- A simple (but not very efficient) implementation of QRAM:
  - $q_a[1:n]$ for the address register holding $|j\rangle$;

Example - QRAMs

- A simple QRAM performs the transformation: for any data set $D[0:N]$ with $N = 2^n - 1$, and for any address $0 \leq j \leq N$,

$$|j\rangle|D[0:N]\rangle \mapsto |j\rangle|D[j]\rangle|D[0:j-1]\rangle|D[0]\rangle|D[j+1:N]\rangle.$$

- Intuition: given an address j, the desired data element $D[j]$ is swapped out.
- A simple (but not very efficient) implementation of QRAM:
  - $q_a[1:n]$ for the address register holding $|j\rangle$;
  - $q_D[0:N]$ for the data register holding quantum state $|D[0:N]\rangle$.

Example - QRAMs (Continued)

- Quantum (!!!) Divide and Conquer:

$$
\begin{aligned}
U(l,r,k) \Leftarrow \textbf{if } k \leq n \textbf{ then} \\
&\textbf{begin local } m := \lfloor (l+r)/2 \rfloor; \\
&\quad \textbf{qif}[q_a[k]]\ |0\rangle \rightarrow U(l,m,k+1) \\
&\qquad\qquad \square\ \ |1\rangle \rightarrow U(m+1,r,k+1); \\
&\qquad\qquad\quad SWAP[q_D[l], q_D[m+1]]; \\
&\qquad\qquad\quad U(m+1,r,k+1) \\
&\quad \textbf{fiq} \\
&\textbf{end} \\
&\textbf{fi}
\end{aligned}
$$

Then we call $U(0,N,1)$ for the QRAQM operation.

Outline

Back to QFT

$$QFT(m,n) \Leftarrow \textbf{if } m = n \textbf{ then } S_{0,0}[q[m]];$$
$$\textbf{else qif}[q[m+1:n]]\ (\square|k\rangle \rightarrow S_{n-m,k}[q[m]])\ \textbf{fiq};$$
$$QFT(m+1,n); Shift(m,n)$$
$$\textbf{fi},$$

$$Shift(m,n) \Leftarrow \textbf{if } m < n \textbf{ then}$$
$$Swap[q[m],q[n]]; Shift(m+1,n)$$
$$\textbf{fi}$$

where single-qubit quantum gate:

$$S_{n,k} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 2^{-\frac{2\pi ik}{2^n}} & -2^{-\frac{2\pi ik}{2^n}} \end{pmatrix}$$

Question

- It is not easy to see that this recursive program actually realise QFT!

Question

- It is not easy to see that this recursive program actually realise QFT!
- How to verify its correctness?

Logic System

Question

- It is not easy to see that this recursive program actually realise QFT!
- How to verify its correctness?

Logic System

- We developed a sound and (relatively) complete logic system for verifying $\mathbf{RQC}^{++}$:

Question

- It is not easy to see that this recursive program actually realise QFT!
- How to verify its correctness?

Logic System

- We developed a sound and (relatively) complete logic system for verifying $\mathbf{RQC}^{++}$:
- Hoare triple for specifying correctness:

$$\{A, |\varphi\rangle\}\ C\ \{B, |\psi\rangle\}$$

Logic System (Continued)

- Proof rules for reasoning about correctness:

$$\text{(QIF)}\quad \frac{\{A, |\varphi_i\rangle\}\ C_i\ \{B, |\psi_i\rangle\}\ (i = 0, 1)}{\begin{array}{l}\{A, \alpha_0|0\rangle_q|\varphi_0\rangle + \alpha_1|1\rangle_q|\varphi_1\rangle\} \\ \qquad \textbf{qif}[q](|0\rangle \to C_0) \square (|1\rangle \to C_1)\ \textbf{fiq} \\ \qquad\qquad \{B, \alpha_0|0\rangle_q|\psi_0\rangle + \alpha_1|1\rangle_q|\psi_1\rangle\}\end{array}}$$

$$\text{(Recursion)}\quad \frac{\begin{array}{l}A \to r \geq 0 \\ \{A \wedge r < z, |\varphi\rangle\}\ P(\bar{t})\ \{B, |\psi\rangle\}\ \vdash\ \{A \wedge r = z, |\varphi\rangle\} \\ \qquad \textbf{begin local}\ \bar{u} := \bar{t}; C\ \textbf{end}\ \{B, |\psi\rangle\}\end{array}}{\{A, |\varphi\rangle\}\ P(\bar{t})\ \{B, |\psi\rangle\}}$$

Logic System (Continued)

- Proof rules for reasoning about correctness:

$$\text{(QIF)}\quad \frac{\{A, |\varphi_i\rangle\}\ C_i\ \{B, |\psi_i\rangle\}\ (i = 0, 1)}{\begin{array}{l}\{A, \alpha_0|0\rangle_q|\varphi_0\rangle + \alpha_1|1\rangle_q|\varphi_1\rangle\} \\ \qquad \textbf{qif}[q](|0\rangle \to C_0) \square (|1\rangle \to C_1)\ \textbf{fiq} \\ \qquad\qquad \{B, \alpha_0|0\rangle_q|\psi_0\rangle + \alpha_1|1\rangle_q|\psi_1\rangle\}\end{array}}$$

$$\text{(Recursion)}\quad \frac{\begin{array}{l}\{A \wedge r < z, |\varphi\rangle\}\ P(\bar{t})\ \{B, |\psi\rangle\} \vdash \{A \wedge r = z, |\varphi\rangle\} \\ \qquad\qquad \textbf{begin local}\ \bar{u} := \bar{t}; C\ \textbf{end}\ \{B, |\psi\rangle\}\end{array} \quad A \to r \geq 0}{\{A, |\varphi\rangle\}\ P(\bar{t})\ \{B, |\psi\rangle\}}$$

- M. S. Ying and Z. C. Zhang, Verification of recursively defined quantum circuits, *PLDI* 2026.

Verification in Lean or Rcoq???

Outline

Compilation

- We defined a notion of quantum register machine.

Compilation

- We defined a notion of quantum register machine.
- A framework for implementation of quantum recursive programs with quantum if-statements, including automatic parallelisation.

Compilation

- We defined a notion of quantum register machine.
- A framework for implementation of quantum recursive programs with quantum if-statements, including automatic parallelisation.
- Z. C. Zhang and M. S. Ying, Quantum register Machine: efficient implementation of quantum recursive programs, *PLDI* 2025

Some key ideas and techniques inspired by:

Compilation

- We defined a notion of quantum register machine.
- A framework for implementation of quantum recursive programs with quantum if-statements, including automatic parallelisation.
- Z. C. Zhang and M. S. Ying, Quantum register Machine: efficient implementation of quantum recursive programs, *PLDI* 2025

Some key ideas and techniques inspired by:

- C. Yuan, A. Villanyi and M. Carbin, Quantum control machine: the limits of control flow in quantum programming, *OOPSLA* 2024

Compilation

- We defined a notion of quantum register machine.
- A framework for implementation of quantum recursive programs with quantum if-statements, including automatic parallelisation.
- Z. C. Zhang and M. S. Ying, Quantum register Machine: efficient implementation of quantum recursive programs, *PLDI* 2025

Some key ideas and techniques inspired by:

- C. Yuan, A. Villanyi and M. Carbin, Quantum control machine: the limits of control flow in quantum programming, *OOPSLA* 2024
- We are developing a compiler for quantum recursive programs.

THANK YOU!