A programming language characterizing quantum polynomial time

Emmanuel Hainry, Romain Péchoux and Mário Silva

LORIA, Université de Lorraine

EQIP 2022

Explicit characterizations

A function $f \in \{0,1\}^\star \to \{0,1\}^\star$ is in FBQP iff...

Explicit characterizations

A function $f \in \{0,1\}^\star \to \{0,1\}^\star$ is in FBQP iff...

Time-bounded quantum Turing machine

... there exists a QTM computing f in polynomial time with probability at least 2/3.

Explicit characterizations

A function $f \in \{0,1\}^\star \to \{0,1\}^\star$ is in FBQP iff...

Time-bounded quantum Turing machine

... there exists a QTM computing f in polynomial time with probability at least 2/3.

Size-bounded quantum circuits (Yao, 1995)

... there exists a uniform family of circuits with polynomial size computing f with probability at least 2/3.

Explicit characterizations

A function $f \in \{0,1\}^\star \to \{0,1\}^\star$ is in FBQP iff...

Time-bounded quantum Turing machine

... there exists a QTM computing f in polynomial time with probability at least $2/3$.

Size-bounded quantum circuits (Yao, 1995)

... there exists a uniform family of circuits with polynomial size computing f with probability at least $2/3$.

These constitute *explicit* characterizations: time or space bounds.

Class $\widehat{\square_1^{\mathrm{QP}}}$ of functions introduced in (Yamakami, 2020):

- Basic functions: $\{Id,\ Ph_\theta,\ Rot_\theta,\ NOT,\ SWAP\}$;
- Composition $g \circ f(|\psi\rangle)$;
- Quantum branching $|0\rangle \otimes f(\langle 0|\psi\rangle) + |1\rangle \otimes g(\langle 1|\psi\rangle)$;
- Multiqubit quantum recursion

$$kQRec_t(|\psi\rangle) = \begin{cases} f(|\psi\rangle) & \text{if size}(|\psi\rangle) \leq t \\ g\Big(\sum_{w\in\{0,1\}^k} |w\rangle \otimes f_w\big(\langle w|\, h(|\psi\rangle)\big)\Big) & \text{else} \end{cases}$$

where $f_w \in \{Id, kQRec_t\}$.

note: $\langle 0|\big(\alpha\,|011\rangle + \beta\,|101\rangle\big) = \alpha\,|11\rangle$

Implicit characterizations

Yamakami (2020)

A function f is in FBQP iff there exists a $g \in \widehat{\square_1^{\mathrm{QP}}}$ that approximates f with probability $2/3$.

Implicit characterizations

Yamakami (2020)

A function f is in FBQP iff there exists a $g \in \widehat{\square_1^{\mathrm{QP}}}$ that approximates f with probability $2/3$.

and also

Dal Lago et al. (2009)

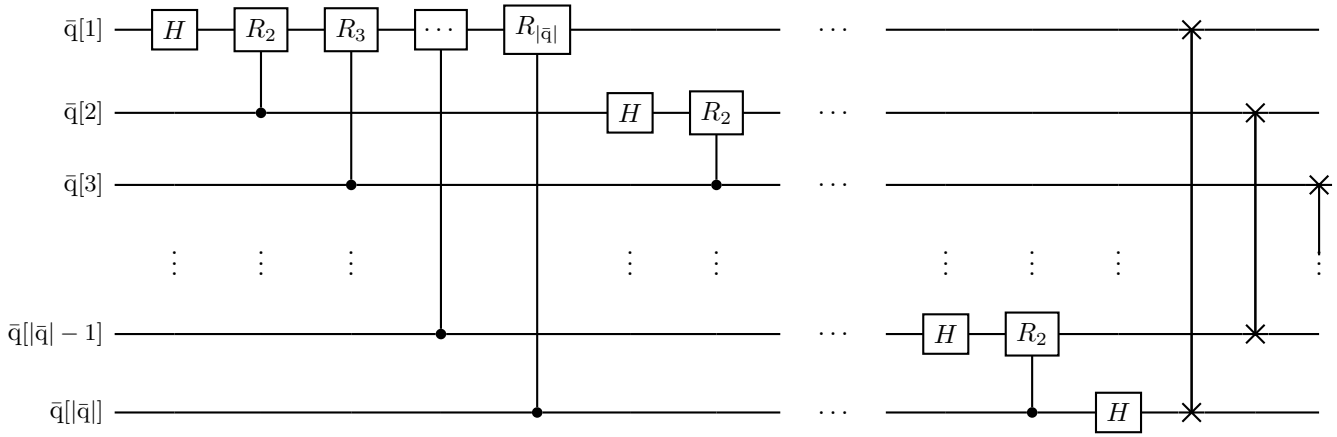
A language L is in BQP iff there exists a language L' with elements decided in the lambda calculus SQ with at least $2/3$ success probability.

What is missing

Possible improvements:

- An implicit description should be as expressive and intuitive as possible, therefore simpler programming languages are preferable to function algebras or lambda calculi;
- In general, given a function in the class, it is very hard to find the corresponding poly-sized circuits (current approach: QTM simulation, then use Yao's equivalency).

Syntax of FOQ: an example




```
decl rec(p̄){
  if |p̄| > 0 then
    p̄[1] *= H;
    call rot[2](p̄);
    call rec(p̄ ⊖ [1]);
  else skip; },
```

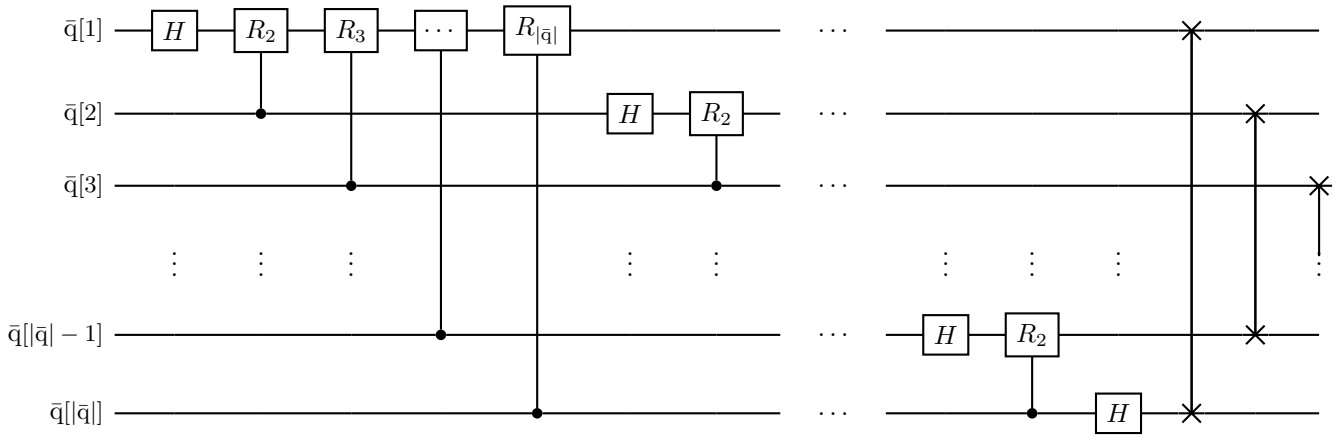
```
decl rot[x](p̄){
  if |p̄| > 1 then
    qcase p̄[2] of {
      0 → skip;
      1 →
p̄[1] *= Ph^{λx.π/2^{x−1}}(x);
    }
    call rot[x + 1](p̄ ⊖ [2]);
  else skip; },
```

```
decl inv(p̄){
  if |p̄| > 1 then
    SWAP(p̄[1], p̄[|p̄|]);
    call inv(p̄ ⊖ [1, |p̄|]);
  else skip; } ::
```

call rec(q̄); **call** inv(q̄);

Syntax of FOQ: an example

Procedure declarations

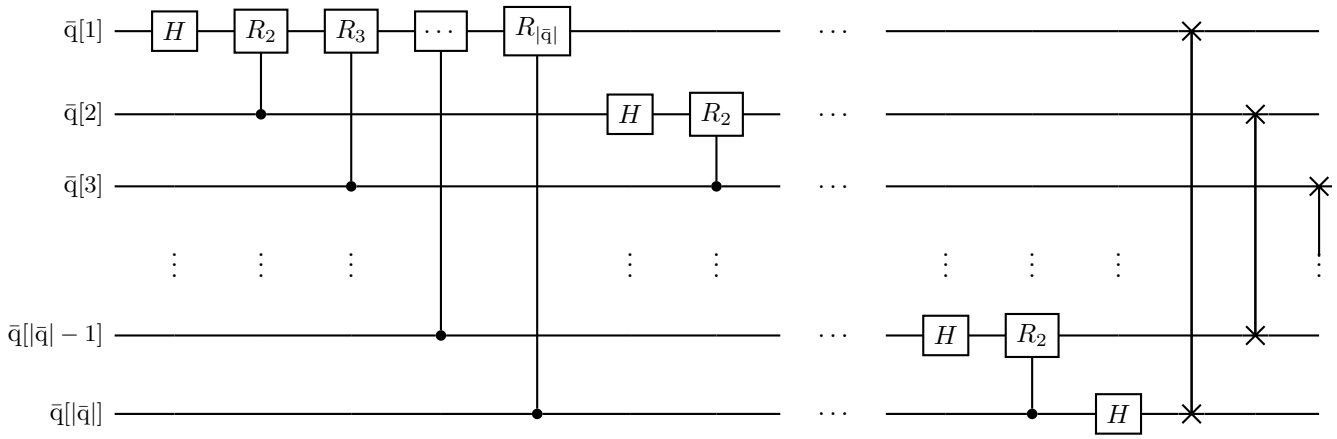



```
decl rec(p̄){
  if |p̄| > 0 then
    p̄[1] *= H;
    call rot[2](p̄);
    call rec(p̄ ⊖ [1]);
  else skip; } ,

decl rot[x](p̄){
  if |p̄| > 1 then
    qcase p̄[2] of {
      0 → skip;
      1 → p̄[1] *= Ph^{λx.π/2^{x−1}}(x);
    }
    call rot[x + 1](p̄ ⊖ [2]);
  else skip; },

decl inv(p̄){
  if |p̄| > 1 then
    SWAP(p̄[1], p̄[|p̄|]);
    call inv(p̄ ⊖ [1, |p̄|]);
  else skip; } ::

call rec(q̄); call inv(q̄);
```

Syntax of FOQ: an example

Procedure declarations

Program statement



decl $\mathrm{rec}(\bar{p})\{$
  if $|\bar{p}| > 0$ **then**
    $\bar{p}[1]$ *= H;
    call $\mathrm{rot}[2](\bar{p})$;
    call $\mathrm{rec}(\bar{p} \ominus [1])$;
  else skip; $\}$,

decl $\mathrm{rot}[x](\bar{p})\{$
  if $|\bar{p}| > 1$ **then**
    qcase $\bar{p}[2]$ **of** $\{$
      $0 \rightarrow$ **skip**;
      $1 \rightarrow$
$\bar{p}[1]$ *= $\mathrm{Ph}^{\lambda x.\pi/2^{x-1}}(x)$;
    $\}$
    call $\mathrm{rot}[x+1](\bar{p} \ominus [2])$;
  else skip; $\}$,

decl $\mathrm{inv}(\bar{p})\{$
  if $|\bar{p}| > 1$ **then**
    $\mathrm{SWAP}(\bar{p}[1], \bar{p}[|\bar{p}|])$;
    call $\mathrm{inv}(\bar{p} \ominus [1, |\bar{p}|])$;
  else skip; $\}$ ::

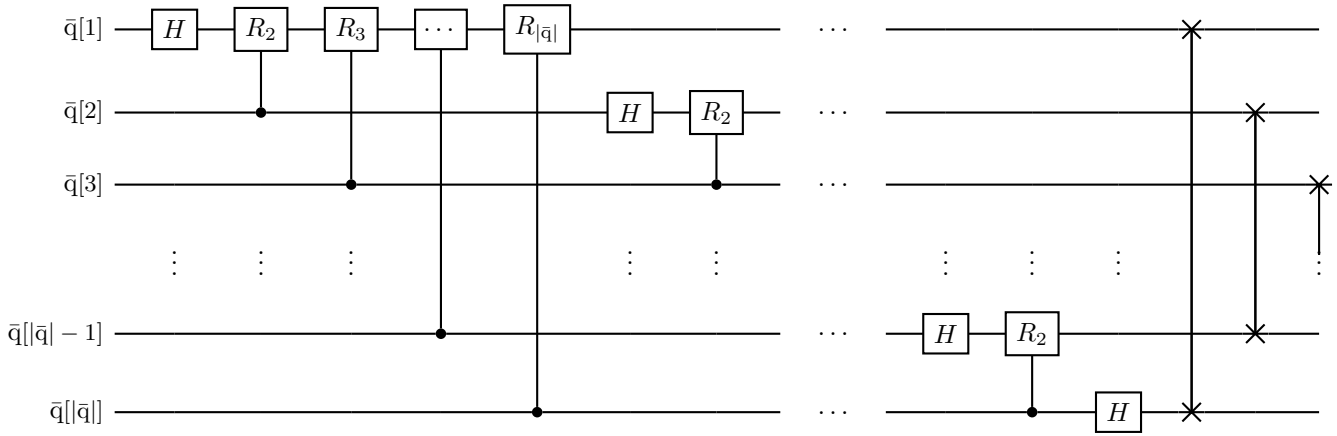
call $\mathrm{rec}(\bar{q})$; **call** $\mathrm{inv}(\bar{q})$;

Syntax of FOQ: an example

Procedure declarations

Program statement

Sorted subsets of qubits



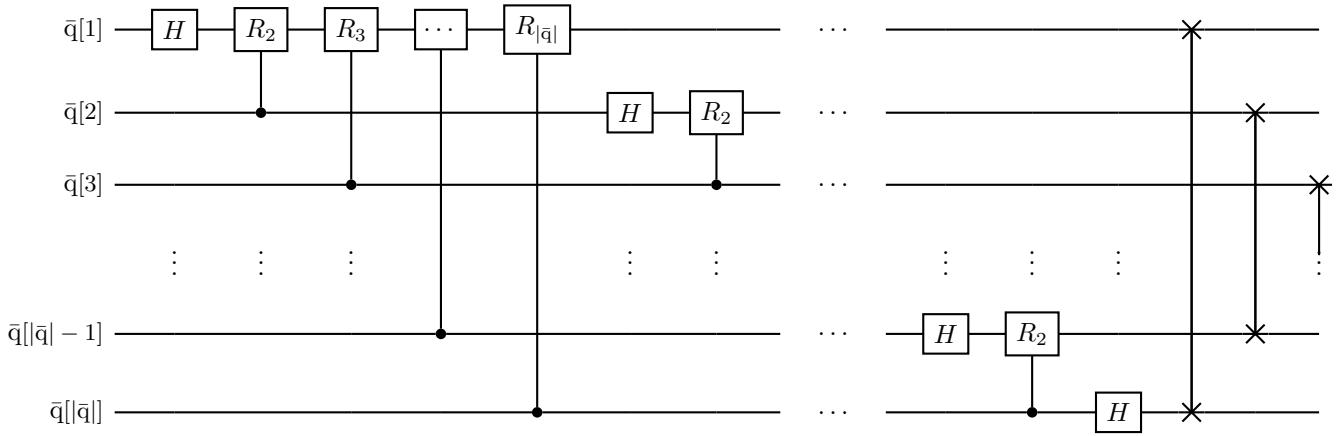

```
decl rec( p̄ ){
  if |p̄| > 0 then
    p̄[1] *= H;
    call rot[2]( p̄ );
    call rec( p̄ ⊖ [1] );
  else skip; },
```

```
decl rot[x]( p̄ ){
  if |p̄| > 1 then
    qcase p̄[2] of {
      0 → skip;
      1 → p̄[1] *= Ph^{λx.π/2^{x−1}}(x);
    }
    call rot[x + 1]( p̄ ⊖ [2] );
  else skip; },
```

```
decl inv( p̄ ){
  if |p̄| > 1 then
    SWAP( p̄[1] , p̄[|p̄|] );
    call inv( p̄ ⊖ [1, |p̄|] );
  else skip; } ::
```

```
call rec( q̄ ); call inv( q̄ );
```

Syntax of FOQ: an example



Procedure declarations

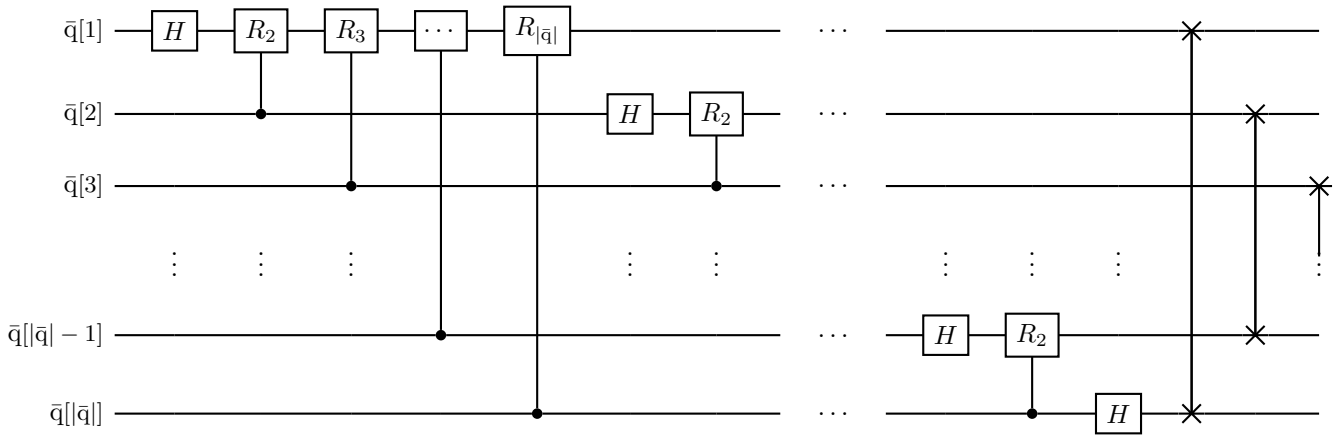
Program statement

Sorted subsets of qubits

Integers

```
decl rec(p̄){
  if |p̄| > 0 then
    p̄[1] *= H;
    call rot[2](p̄);
    call rec(p̄ ⊖ [1]);
  else skip; },

decl rot[x](p̄){
  if |p̄| > 1 then
    qcase p̄[2] of {
      0 → skip;
      1 →
p̄[1] *= Ph^{λx.π/2^{x−1}}(x);
    }
    call rot[x+1](p̄⊖[2]);
  else skip; },

decl inv(p̄){
  if |p̄| > 1 then
    SWAP(p̄[1], p̄[|p̄|]);
    call inv(p̄⊖[1, |p̄|]);
  else skip; } ::

call rec(q̄); call inv(q̄);
```

Syntax of FOQ: an example



Procedure declarations

Program statement

Sorted subsets of qubits

Integers

Unitary operations

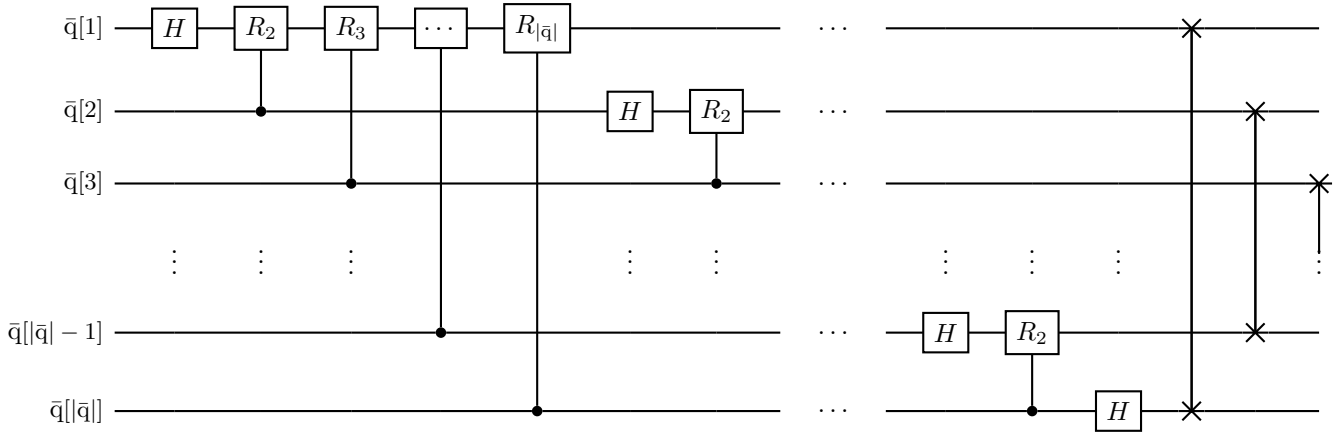
```
decl rec(p̄){
  if |p̄| > 0 then
    p̄[1] *= H;
    call rot[2](p̄);
    call rec(p̄ ⊖ [1]);
  else skip; },
```

```
decl rot[x](p̄){
  if |p̄| > 1 then
    qcase p̄[2] of {
      0 → skip;
      1 →
    p̄[1] *= Ph^{λx.π/2^{x−1}}(x);
    }
    call rot[x + 1](p̄ ⊖ [2]);
  else skip; },
```

```
decl inv(p̄){
  if |p̄| > 1 then
    SWAP(p̄[1], p̄[|p̄|]);
    call inv(p̄ ⊖ [1, |p̄|]);
  else skip; } ::
```

```
call rec(q̄); call inv(q̄);
```

Syntax of FOQ: an example



Procedure declarations

Program statement

Sorted subsets of qubits

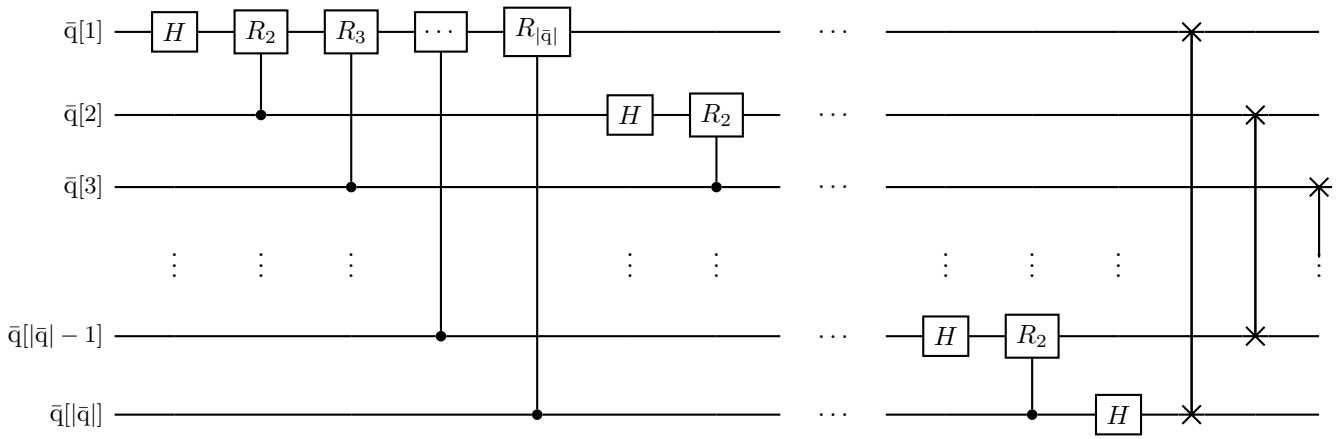
Integers

Unitary operations

Procedure calls

```
decl rec(p̄){
  if |p̄| > 0 then
    p̄[1] *= H;
    call rot[2](p̄);
    call rec(p̄ ⊖ [1]);
  else skip; },

decl rot[x](p̄){
  if |p̄| > 1 then
    qcase p̄[2] of {
      0 → skip;
      1 → p̄[1] *= Ph^{λx.π/2^{x−1}}(x);
    }
    call rot[x + 1](p̄ ⊖ [2]);
  else skip; },

decl inv(p̄){
  if |p̄| > 1 then
    SWAP(p̄[1], p̄[|p̄|]);
    call inv(p̄ ⊖ [1, |p̄|]);
  else skip; } ::

call rec(q̄); call inv(q̄);
```

Syntax of FOQ: an example



Procedure declarations

Program statement

Sorted subsets of qubits

Integers

Unitary operations

Procedure calls

Classical conditions

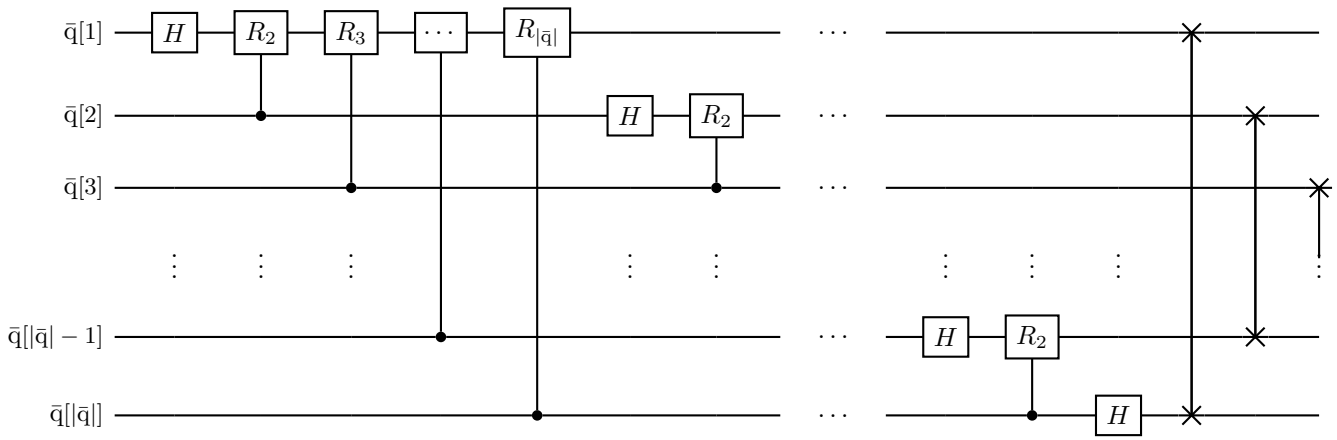
decl $\mathrm{rec}(\bar{p})\{$
  if $|\bar{p}| > 0$ **then**
    $\bar{p}[1]$ *= H;
    call $\mathrm{rot}[2](\bar{p})$;
    call $\mathrm{rec}(\bar{p} \ominus [1])$;
  else **skip**; $\}$,

decl $\mathrm{rot}[x](\bar{p})\{$
  if $|\bar{p}| > 1$ **then**
    qcase $\bar{p}[2]$ **of** $\{$
      $0 \rightarrow$ **skip**;
      $1 \rightarrow$
$\bar{p}[1]$ *= $\mathrm{Ph}^{\lambda x.\pi/2^{x-1}}(x)$;
    $\}$
    call $\mathrm{rot}[x+1](\bar{p} \ominus [2])$;
  else **skip**; $\}$,

decl $\mathrm{inv}(\bar{p})\{$
  if $|\bar{p}| > 1$ **then**
    $\mathrm{SWAP}(\bar{p}[1], \bar{p}[|\bar{p}|])$;
    call $\mathrm{inv}(\bar{p} \ominus [1, |\bar{p}|])$;
  else **skip**; $\}$ ::

call $\mathrm{rec}(\bar{q})$; **call** $\mathrm{inv}(\bar{q})$;

Syntax of FOQ: an example



Procedure declarations

Program statement

Sorted subsets of qubits

Integers

Unitary operations

Procedure calls

Classical conditions

Quantum branching

decl $\mathrm{rec}(\bar{p})\{$
  if $|\bar{p}| > 0$ **then**
    $\bar{p}[1]$ *= H;
    call $\mathrm{rot}[2](\bar{p})$;
    call $\mathrm{rec}(\bar{p} \ominus [1])$;
  else skip; $\}$,

decl $\mathrm{rot}[x](\bar{p})\{$
  if $|\bar{p}| > 1$ **then**
    qcase $\bar{p}[2]$ **of** $\{$
      $0 \rightarrow$ **skip**;
      $1 \rightarrow$
$\bar{p}[1]$ *= $\mathrm{Ph}^{\lambda x.\pi/2^{x-1}}(x)$;
    $\}$
    call $\mathrm{rot}[x+1](\bar{p} \ominus [2])$;
  else skip; $\}$,

decl $\mathrm{inv}(\bar{p})\{$
  if $|\bar{p}| > 1$ **then**
    $\mathrm{SWAP}(\bar{p}[1], \bar{p}[|\bar{p}|])$;
    call $\mathrm{inv}(\bar{p} \ominus [1, |\bar{p}|])$;
  else skip; $\}$ ::

call $\mathrm{rec}(\bar{q})$; **call** $\mathrm{inv}(\bar{q})$;

Restrictions on recursion

Ensuring termination: $\mathrm{P} \in \mathrm{WF}$

$\forall \mathrm{proc} \in \mathrm{P},$

$\quad \forall$ **call** $\mathrm{proc}'[\mathrm{i}](s); \in \mathrm{S}^{\mathrm{proc}},$

$\quad\quad \mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}' \Rightarrow s = \bar{\mathrm{p}} \ominus [\mathrm{i}_1, \ldots, \mathrm{i}_k]$

(all mutually recursive calls decrease the number of qubits)

Width of a procedure

$$\mathsf{width}_{\mathrm{P}}(\mathrm{proc}) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}^{\mathrm{proc}}),$$

$$w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{skip};) \triangleq 0,$$

$$w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{q} \mathbin{*=} \mathrm{U}^{f}(\mathrm{i});) \triangleq 0,$$

$$w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1\ \mathrm{S}_2) \triangleq w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1) + w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_2)$$

$$w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{if}\ \mathrm{b}\ \textbf{then}\ \mathrm{S}_{\textbf{true}}\ \textbf{else}\ \mathrm{S}_{\textbf{false}}) \triangleq \max(w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_{\textbf{true}}), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_{\textbf{false}})),$$

$$w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{qcase}\ \mathrm{q}\ \textbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}) \triangleq \max(w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_0), w_{\mathrm{P}}^{\mathrm{proc}}(\mathrm{S}_1)),$$

$$w_{\mathrm{P}}^{\mathrm{proc}}(\textbf{call}\ \mathrm{proc}'[\mathrm{i}](\mathrm{s});) \triangleq \begin{cases} 1 & \text{if } \mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}', \\ 0 & \text{otherwise.} \end{cases}$$

Polynomial time programs: $\mathrm{P} \in \mathrm{PFOQ}$

$\mathrm{P} \in \mathrm{WF}$ and $\forall \mathrm{proc} \in \mathrm{P},\ \mathrm{width}_{\mathrm{P}}(\mathrm{proc}) \leq 1.$

(at most one recursive call per branch and reduction of input size)

Example. We have that $\mathrm{width}_{\mathrm{QFT}}(\mathrm{rec}) = 1$.

decl $\mathrm{rec}(\bar{\mathrm{p}})\{$
$\bar{\mathrm{p}}[1]$ *= H;
call $\mathrm{rot}[2](\bar{\mathrm{p}})$;
call $\mathrm{rec}(\bar{\mathrm{p}} \ominus [1])$; $\}$

Reversibility

Terminating FOQ programs are reversible.

Proof.
We show this by induction on the structure of the program. Let $\mathrm{P} = \mathrm{D} :: \mathrm{S}$ be a program that terminates. Define $\mathrm{P}^{-1} \triangleq (\mathrm{D} :: \mathrm{S})^{-1}$ inductively by

$$
\begin{array}{rcl}
(\mathrm{D} :: \mathrm{S})^{-1} & \triangleq & \mathrm{D}^{-1} :: \mathrm{S}^{-1} \\
(\textbf{decl}\ \mathrm{proc}[\mathrm{x}](\bar{\mathrm{p}})\{\mathrm{S}\}, \mathrm{D})^{-1} & \triangleq & \textbf{decl}\ \mathrm{proc}[\mathrm{x}](\bar{\mathrm{p}})\{\mathrm{S}^{-1}\}, \mathrm{D}^{-1} \\
\varepsilon^{-1} & \triangleq & \varepsilon \\
\textbf{skip;}^{-1} & \triangleq & \textbf{skip;} \\
(\mathrm{q} \mathbin{*\!=} \mathrm{U}^f(\mathrm{i});)^{-1} & \triangleq & \mathrm{q} \mathbin{*\!=} (\mathrm{U}^f(\mathrm{i}))^\dagger; \\
(\mathrm{S}_1\ \mathrm{S}_2)^{-1} & \triangleq & \mathrm{S}_2^{-1}\ \mathrm{S}_1^{-1} \\
(\textbf{if}\ \mathrm{b}\ \textbf{then}\ \mathrm{S}_{\textbf{true}}\ \textbf{else}\ \mathrm{S}_{\textbf{false}})^{-1} & \triangleq & \textbf{if}\ \mathrm{b}\ \textbf{then}\ \mathrm{S}_{\textbf{true}}^{-1}\ \textbf{else}\ \mathrm{S}_{\textbf{false}}^{-1} \\
(\textbf{qcase}\ \mathrm{q}\ \textbf{of}\ \{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\})^{-1} & \triangleq & \textbf{qcase}\ \mathrm{q}\ \textbf{of}\ \{0 \to \mathrm{S}_0^{-1}, 1 \to \mathrm{S}_1^{-1}\} \\
(\textbf{call}\ \mathrm{proc}[\mathrm{i}](\mathrm{s});)^{-1} & \triangleq & \textbf{call}\ \mathrm{proc}[\mathrm{i}](\mathrm{s});,
\end{array}
$$

with $\mathrm{NOT}^\dagger \triangleq \mathrm{NOT}$, $\mathrm{R}_Y^f(\mathrm{i})^\dagger \triangleq \mathrm{R}_Y^{-f}(\mathrm{i})$ and $\mathrm{Ph}^f(\mathrm{i})^\dagger \triangleq \mathrm{Ph}^{-f}(\mathrm{i})$. P^{-1} terminates and is such that $\forall\, |\psi\rangle,\ [\![\mathrm{P}^{-1}]\!]([\![\mathrm{P}]\!](|\psi\rangle)) = |\psi\rangle$. □

Completeness

Given $f \in \mathrm{FBQP}$ with polynomial bound $Q \in \mathbb{N}[X]$, there exists a program $\mathrm{P} \in \mathrm{PFOQ}$ such that $[\![P]\!] \circ \phi_Q$ computes f with probability at least $2/3$.

Proof. We can simulate Yamakami's class.

Soundness

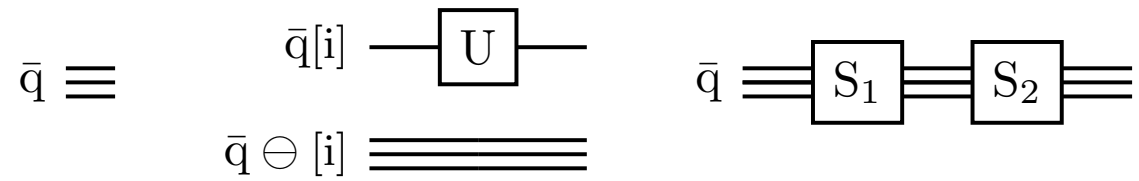
Given a PFOQ program P and a function $f : \{0,1\}^\star \to \{0,1\}^\star$, if f is computed by $[\![P]\!]$ with probability at least $2/3$ then $f \in \mathrm{FBQP}$.

Proof. Simulation of $[\![P]\!]$ by polytime QTM or, equivalently, polytime compilation to circuits.

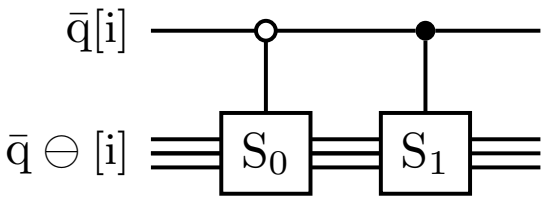
Building a polysized circuit from a statement

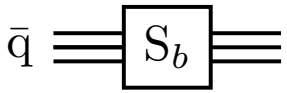
Given a program $\mathrm{P} = \mathrm{D} :: \mathrm{S}$, the circuit interpretation of S is easy to find for non-recursive cases:

skip	$\bar{\mathrm{q}}[\mathrm{i}]$ *= U	$\mathrm{S}_1\ \mathrm{S}_2$



qcase $\bar{\mathrm{q}}[\mathrm{i}]$ **of** $\{0 \to \mathrm{S}_0, 1 \to \mathrm{S}_1\}$	**if** b **then** $\mathrm{S}_{\mathbf{true}}$ **else** $\mathrm{S}_{\mathbf{false}}$

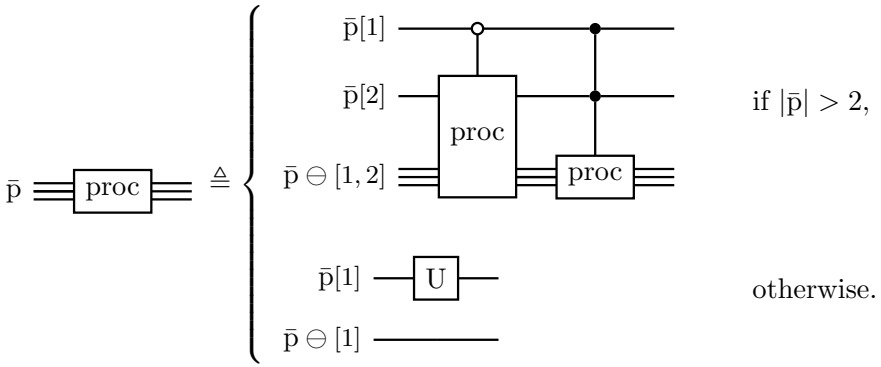




Handling the recursive case

Consider the following program in PFOQ:

$$
\begin{array}{l}
\mathrm{P}(\bar{\mathrm{q}}) \triangleq \textbf{decl}\ \mathrm{proc}(\bar{\mathrm{p}})\{ \\
\quad \textbf{if}\ |\bar{\mathrm{p}}| > 2: \\
\quad\quad \textbf{qcase}\ \bar{\mathrm{p}}[1]\ \textbf{of}\ \{0 \to \textbf{call}\ \mathrm{proc}(\bar{\mathrm{p}} \ominus [1]);, \\
\quad\quad\quad\quad\quad\quad 1 \to \textbf{qcase}\ \bar{\mathrm{p}}[2]\ \textbf{of}\ \{0 \to \textbf{skip};, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad 1 \to \textbf{call}\ \mathrm{proc}(\bar{\mathrm{p}} \ominus [1,2]); \\
\quad\quad\quad\quad\quad\quad \} \\
\quad\quad \} \\
\quad \textbf{else}\ \bar{\mathrm{p}}[1] *= \mathrm{U}; \\
\quad \} \\
\quad :: \textbf{call}\ \mathrm{proc}(\bar{\mathrm{q}});
\end{array}
$$

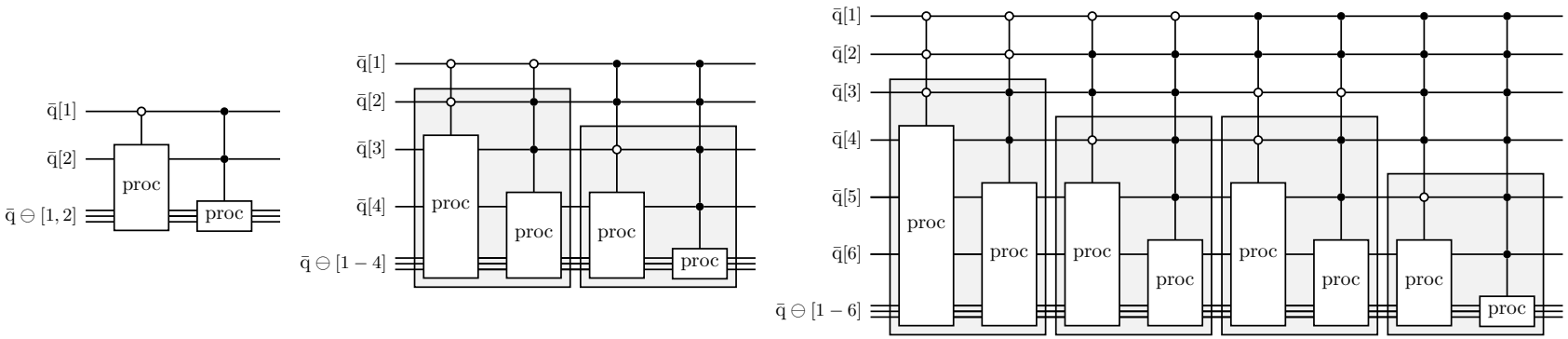


Handling the recursive case

For a large input length, creating the circuit by applying this definition would deal the following progression.

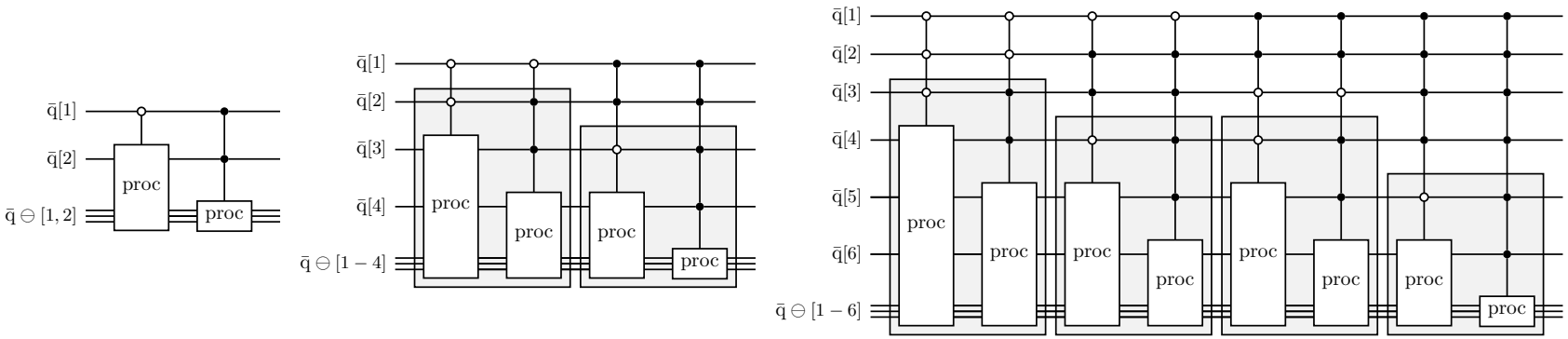
Handling the recursive case

For a large input length, creating the circuit by applying this definition would deal the following progression.



Handling the recursive case

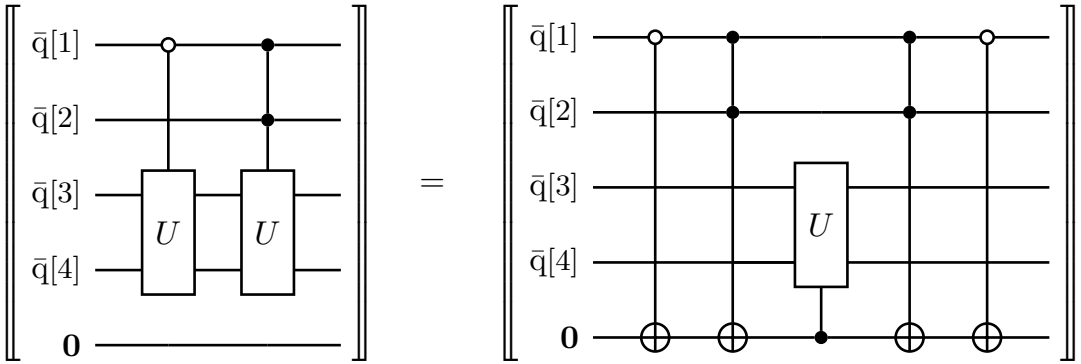
For a large input length, creating the circuit by applying this definition would deal the following progression.



For input $\bar{\mathrm{q}}$ of length n, this construction needs $O(n2^n)$ gates

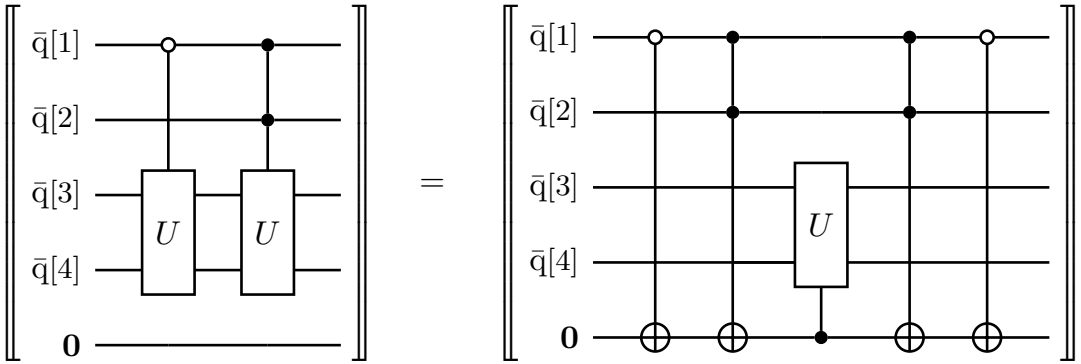
Simplification by merging

Unitary transformation U applied twice on orthogonal control structures.



Simplification by merging

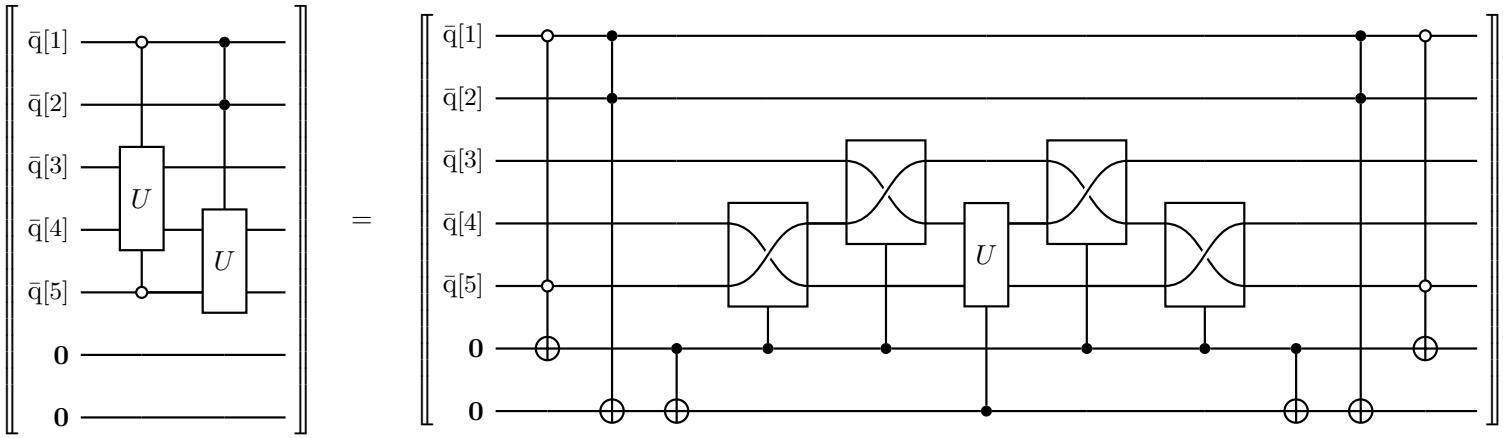
Unitary transformation U applied twice on orthogonal control structures.



For any set of k such calls, we can merge using a single ancilla.

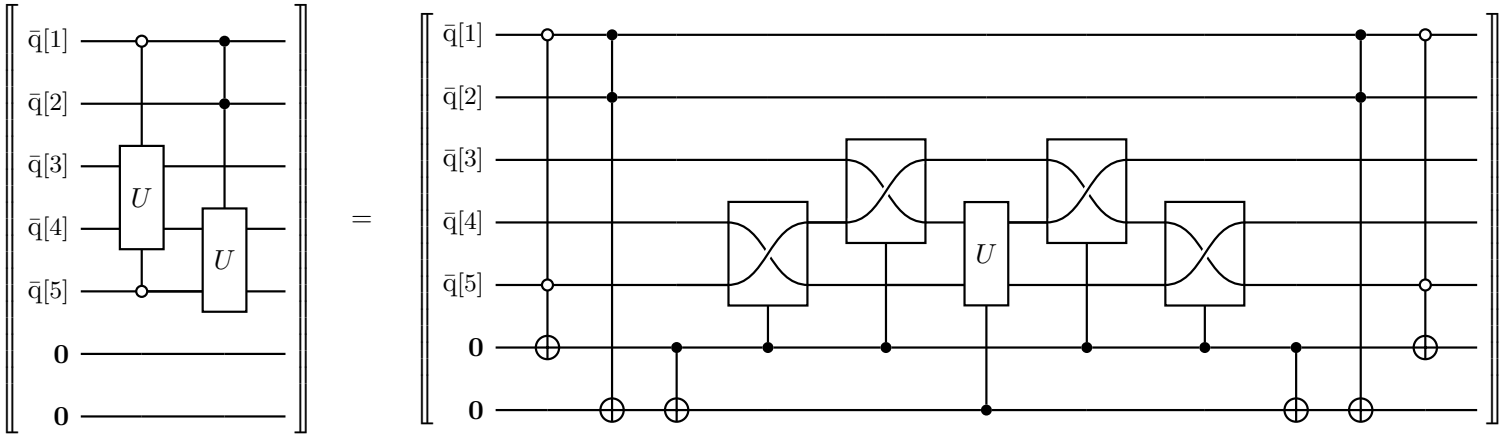
Simplification by merging

General case: gates U applied on different qubit sets and control structures overlap with other cases of U.



Simplification by merging

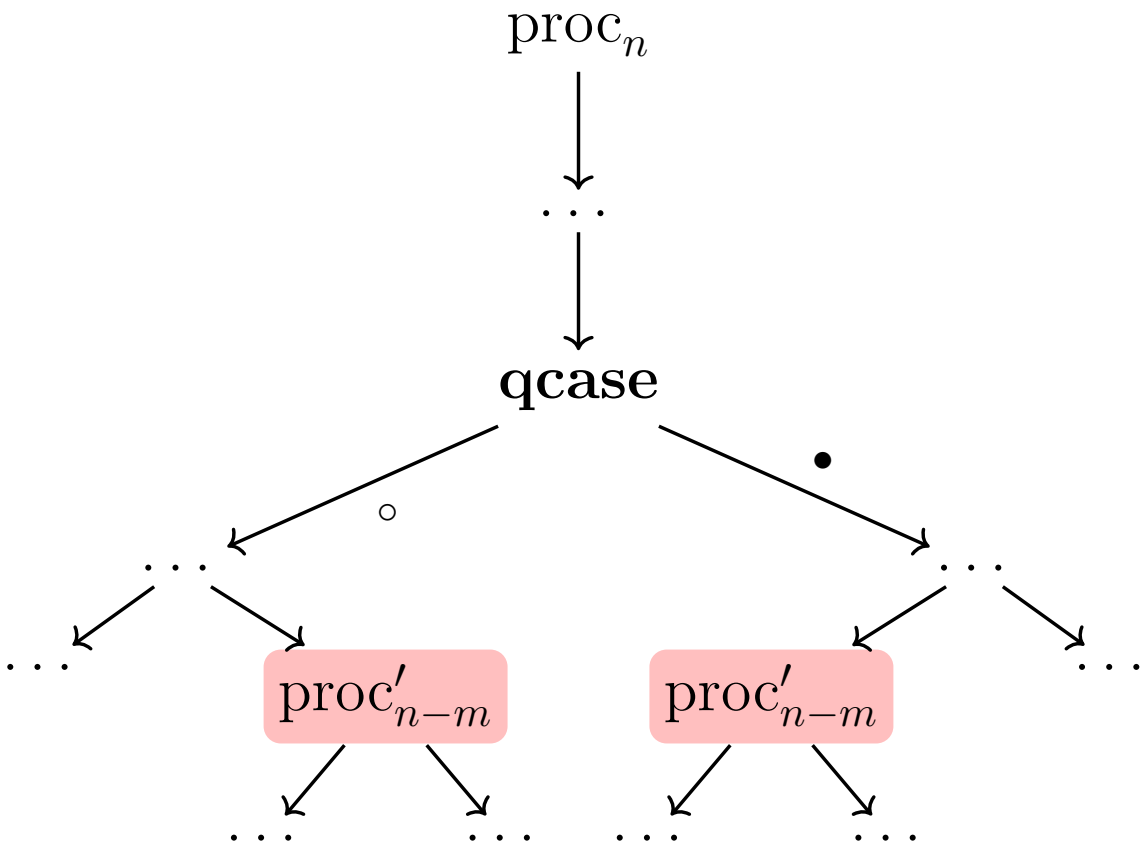
General case: gates U applied on different qubit sets and control structures overlap with other cases of U.



For k calls, we can merge using k ancillas and extra $O(kn)$ gates.

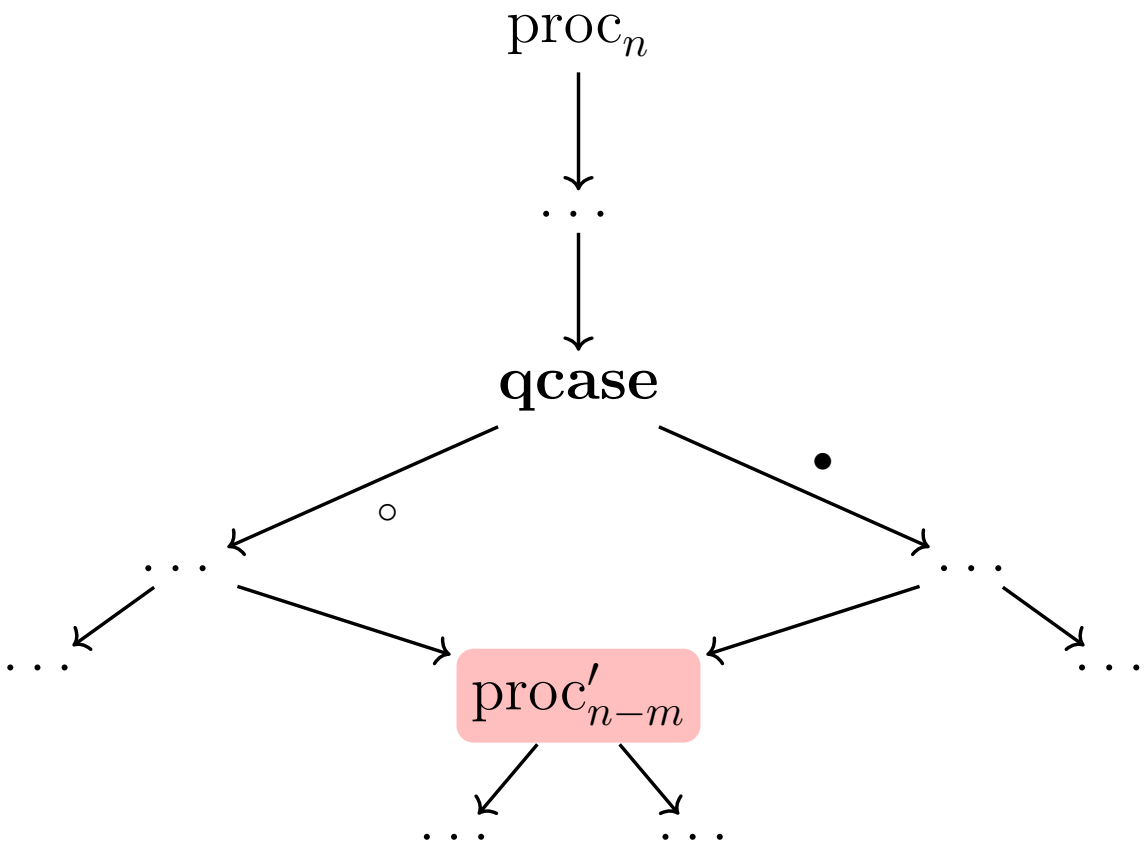
Reduction of complexity

$\mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}'$

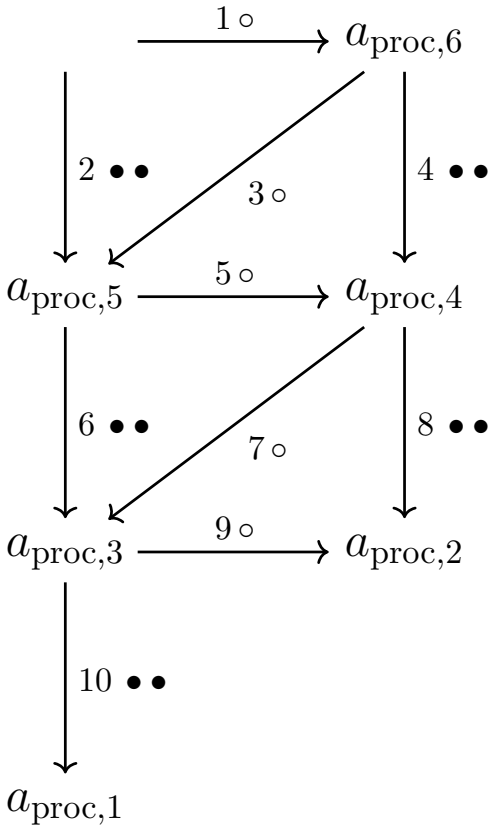


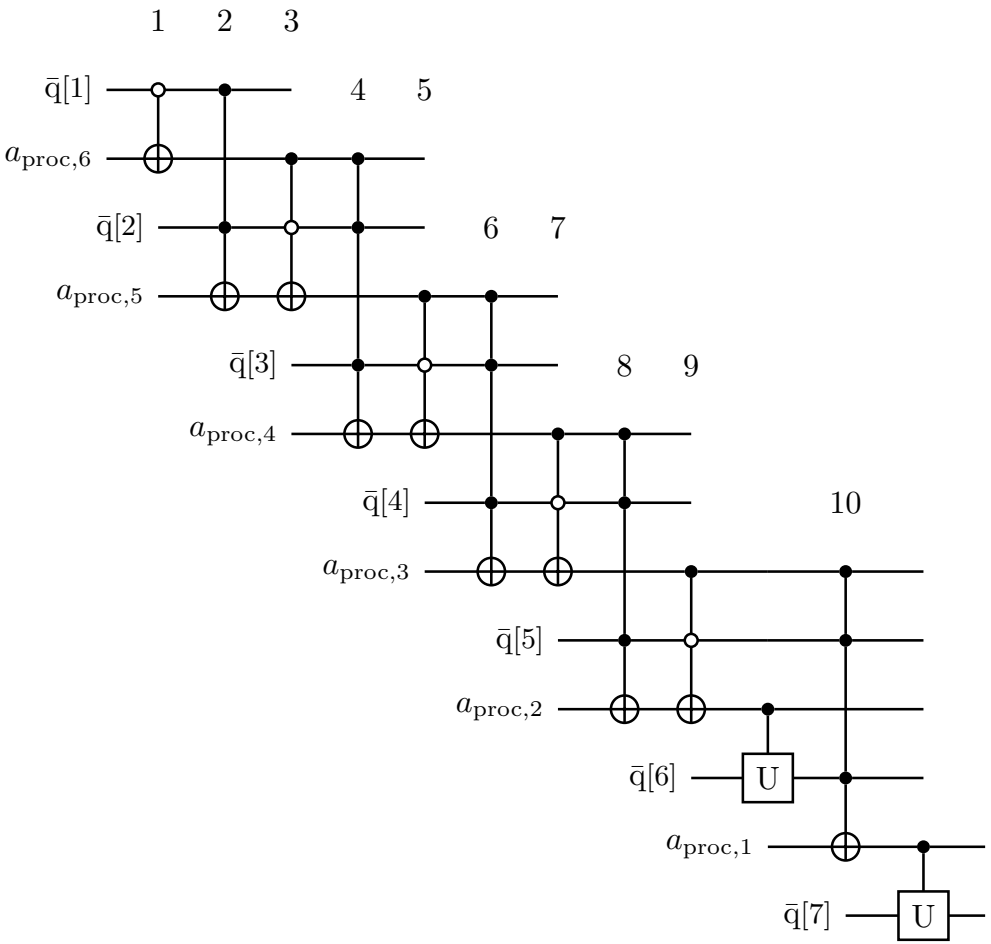
Reduction of complexity

$\mathrm{proc} \sim_{\mathrm{P}} \mathrm{proc}'$



Our approach grows the circuit in $O(n)$ for the previously stated example. Case $n = 7$:





Compilation of polysized circuits

For any $\mathrm{P} \in \mathrm{PFOQ}$ and $n \in \mathbb{N}$, **compile**(P, n) runs in polytime in n and outputs a circuit simulating $[\![\mathrm{P}]\!]$ for input size n.

Proof. Orthogonality and adjacency insurance, then a counting argument on the number of different possible calls. More precisely, the circuit complexity is worst case $O(n^{1+2\#\text{recursive families}})$.

Results:

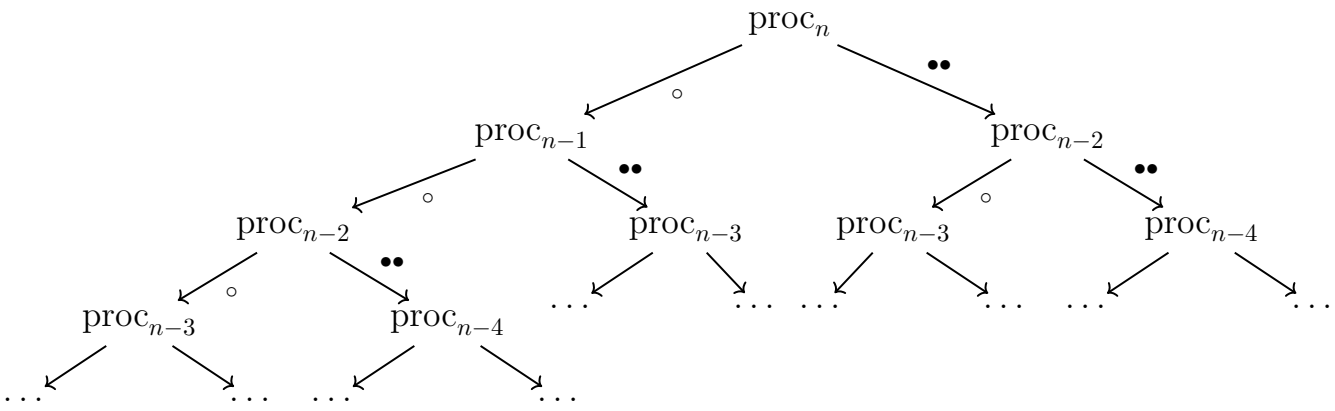
- Quantum programming language with first-order recursive procedures;
- Syntactical restrictions ensuring termination and polynomial time termination;
- Restrictions rendering the class equivalent to polytime quantum procedures, in particular to FBQP;
- Technique for direct circuit compilation, closing the gap between the implicit characterization and the feasibility of application.

Future work:

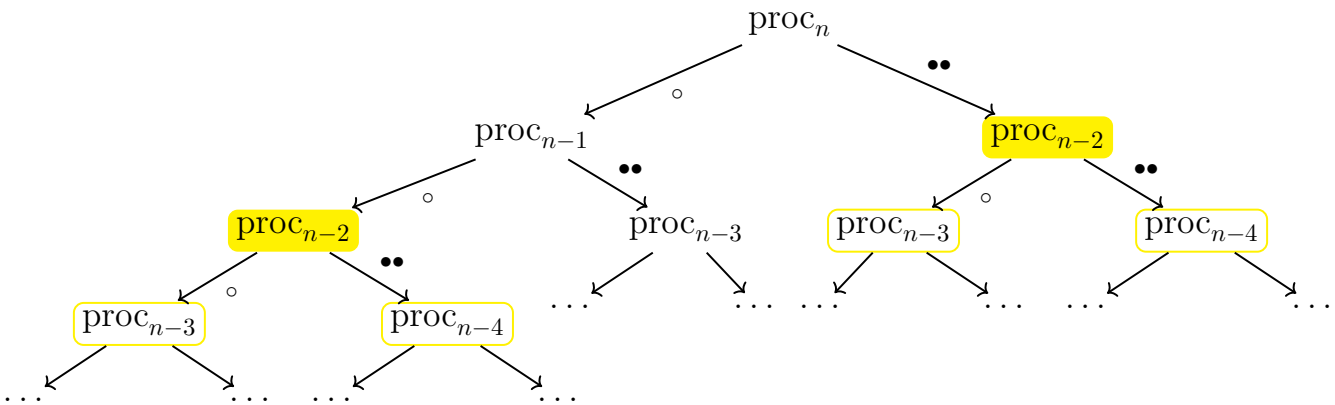
- Expanding the syntax of the language to other constructions (e.g. while, divide and conquer, etc.)
- Applying restrictions to established languages (eg. Quipper, Proto-Quipper)

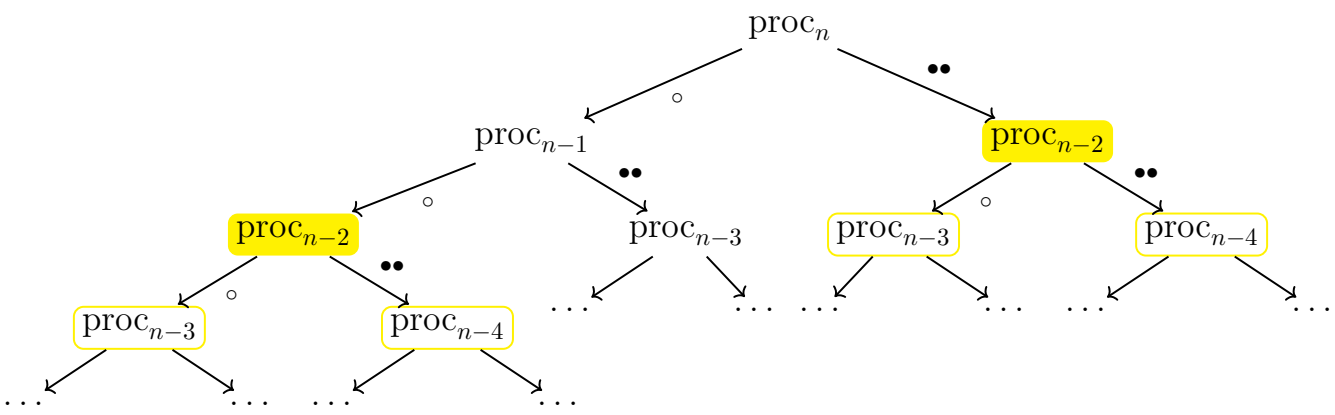
Thank you!

Consider the tree of calls in our example

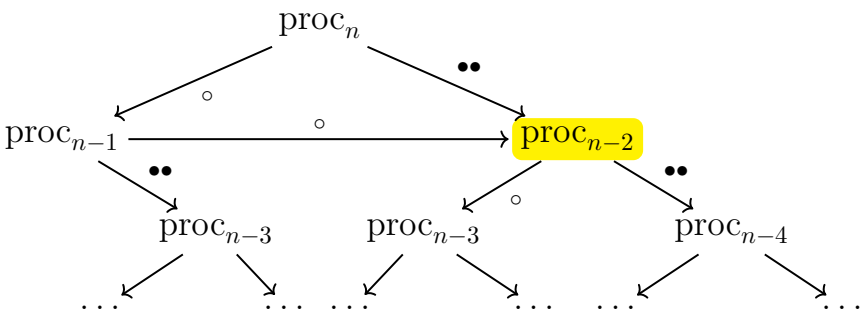


Consider the tree of calls in our example

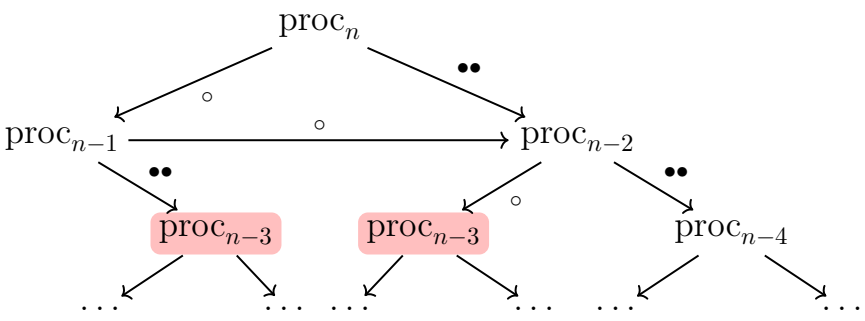




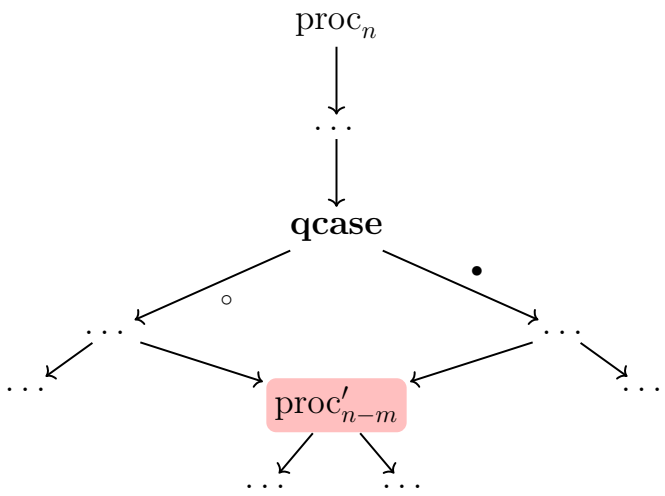
With at most $O(n)$ cost:



Again, we can transform



into



How much does this reduce the complexity?