An imperative programming language characterizing FBQP

Emmanuel Hainry, Romain Péchoux, Mário Silva

An imperative programming language characterizing FBQP

Emmanuel Hainry, Romain Péchoux, Mário Silva

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

Introduction

Quantum programming languages are a useful tool, not only for writing complex algorithms, but also for abstracting and reasoning about their properties and to learn more about what can be done efficiently by quantum computers [1].

However, merely being able to describe a quantum program does not inform us on its complexity, meaning that it may not ultimately be efficiently run on a quantum computer. Consequently, there is a need for static analysis tools and techniques for reasoning about and certifying the complexity of these quantum algorithms.

We introduce an imperative programming language called QPT (Quantum Poly-Time) with recursion rules that captures the complexity class FBQP (Functions Bounded-error Quantum Polytime), the class of functions computable in polynomial time by a quantum Turing machine with at most 1/3 probability of error, commonly accepted as the class of feasible problems for quantum computers. Our result takes advantage of a function algebra proposed by Yamakami [2] characterizing FBQP.

Syntax

$$
\begin{aligned}
P &\triangleq D :: S \\
i, j &\triangleq n \in \mathbb{Z} \mid |\bar{q}| \\
b &\triangleq i < j \mid i = j \\
D &\triangleq \varepsilon \mid \mathbf{Proc}\ proc(m, \bar{p})\{S[m]\},\ D \\
\sigma &\triangleq \emptyset \mid \bar{q} \mid \sigma[i] \mid \sigma_1 \oplus \sigma_2 \mid \mathrm{remove}(\sigma, i) \\
U &\triangleq NOT \mid ROT_\theta \mid PHASE_\theta \\
S &\triangleq \mathbf{skip} \\
&\mid \mathbf{if}\ b\ \mathbf{then}\ S \\
&\mid \sigma[i] {*}= U \\
&\mid S_1; S_2 \\
&\mid \mathbf{Case}\ C(\sigma[i]) = m\ \mathbf{then}\ S_m \\
&\mid \mathbf{call}\ proc(i, \sigma)
\end{aligned}
$$

Sets σ are sorted sets of qubits together with basic operations. The **Case** structure implements a quantum choice: controlled on the state of qubit $\sigma[i]$, we apply S_0 or S_1 to the remainder of the qubits.

Rule for termination (R1)

Let $proc_i \sim proc_j$ mean that $proc_i$ and $proc_j$ are mutually recursive procedures. The following condition ensures that a program with procedure declarations D will always terminate:

$$
\begin{aligned}
&\forall\, \mathbf{Proc}\ proc_i(\bar{p})\{S_i\} \in D, \\
&\quad \forall\ \mathbf{call}\ proc_j(\sigma) \in S_i, \\
&\qquad proc_i \sim proc_j \Rightarrow \sigma \text{ is a proper subset of } \bar{p},
\end{aligned}
$$

i.e., any call to a mutually recursive function must strictly decrease the amount of qubits available.

Rule for poly-time (R2)

We prevent an exponential number of procedure calls by limiting the number of recursive calls in each branch of a **Case**. Let RCalls($\cdot$) represent the maximum number of recursive calls for any branch, then

$$\forall\, \mathbf{Proc}\ proc_i(\bar{p})\{S_i\} \in D,\ \mathrm{RCalls}(proc_i) \le 1,$$

i.e., multiple recursive calls can be done only on different branches of the **Case** structure.

QPT ~ FBQP

Soundness: For any program P in QPT following rules **R1** and **R2**, there exists a poly-sized uniform family of circuits $(C_n)_{n\in\mathbb{N}}$ for each input size n that simulates P.

Completeness: For any function f in FBQP with size-bounding polynomial p, and any constant $\varepsilon \in [0, 1/2)$, there exists a program P in QPT following rules **R1** and **R2** such that, running P on polynomially extended input state ρ_x^p, for $x \in \{0,1\}^n$, a measurement of the first $|f(x)|$ of the output qubits will result in $f(x)$ with probability at least $1 - \varepsilon$.

Building poly-sized circuits

Dealing with procedures that include recursive branching, a straightforward approach to building the circuit will reasily require an exponential number of gates, e.g. in the following procedure:

$$
\begin{aligned}
&\mathbf{Proc}\ f(\bar{p})\{ \\
&\quad \mathbf{if}\ |\bar{p}| > 1 : \\
&\qquad \mathbf{Case}\ C(\bar{p}[1]) = 0\ \mathbf{then} \\
&\qquad\quad \mathbf{call}\ f(\mathrm{remove}(\bar{p}, 1)) \\
&\qquad \mathbf{else\ Case}\ C(\bar{p}[2]) = 1\ \mathbf{then} \\
&\qquad\quad \mathbf{call}\ f(\mathrm{remove}(\bar{p}, \{1, 2\})) \\
&\quad \mathbf{else}\ \bar{p}[1] {*}= U\} :: \\
&\mathbf{call}\ f(\bar{q})
\end{aligned}
$$

As a proof of **Soundness**, we provide an algorithm to build all programs following rule **R2** with a polynomially large set of gates and wires, such as in the example of Figure 1 for the above function applied on an input $\bar{q}$ of size $n = 5$.

For function f the number of gates and wires needed is $O(n^2)$.

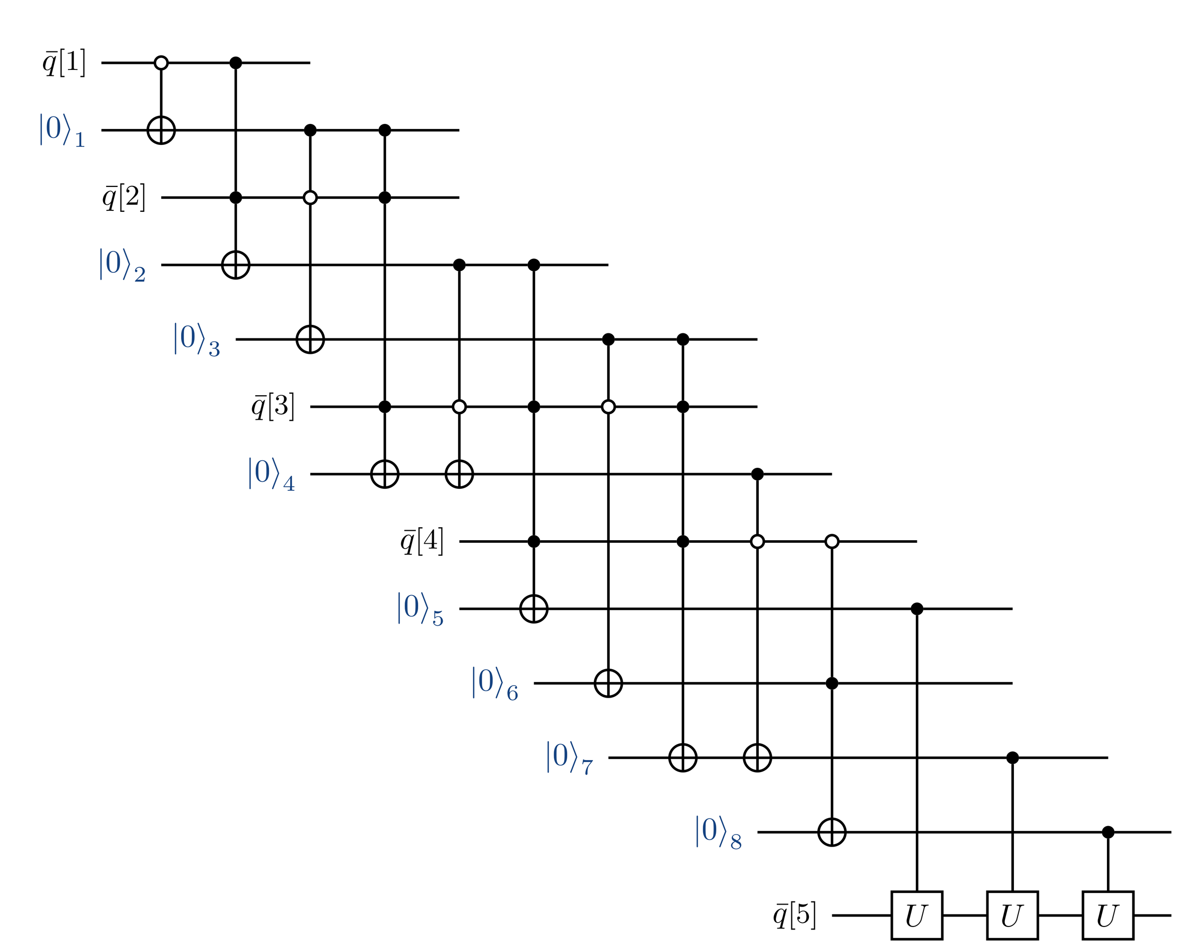

Figure 1: Example circuit of a recursive quantum function f applied on a set of 5 qubits.

Example: QFT

The quantum Fourier transform (QFT) is in FBQP, appearing as a subroutine of Shor's algorithm. It contains two recursive patterns, following rules **R1** and **R2**, highlighted in the circuit of Figure 2.

$$
\begin{aligned}
&\mathbf{Proc}\ QFT(\bar{p})\{ \\
&\quad \bar{p}[1] {*}= H; \\
&\quad \mathbf{call}\ Chain(2, \bar{p}); \\
&\quad \mathbf{call}\ QTF(\mathrm{remove}(\bar{p}, 1))\}, \\
\\
&\mathbf{Proc}\ Chain(m, \bar{p})\{ \\
&\quad \mathbf{Case}\ C(\bar{p}[2]) = 1\ \mathbf{then}\ \bar{p}[1] {*}= R_m; \\
&\quad \mathbf{call}\ Chain(m+1, \mathrm{remove}(\bar{p}, 2))\} :: \\
\\
&\mathbf{call}\ QFT(\bar{q})
\end{aligned}
$$

The circuit can be implemented with size $O(n^2)$ gates, for an input with n qubits.

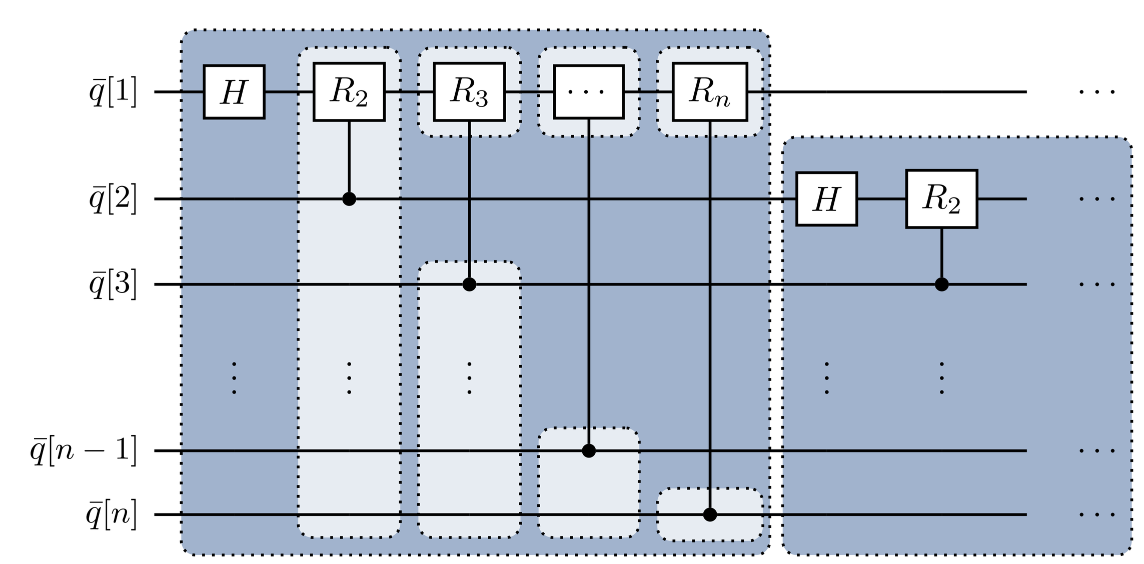

Figure 2: Representation of two recursive patterns with decreasing qubit set in the QFT.

References

[1] Peter Selinger.
Towards a quantum programming language.
Mathematical Structures in Computer Science, 14(4):527–586, 2004.

[2] Tomoyuki Yamakami.
A schematic definition of quantum polynomial time computability.
J. Symb. Log., 85(4):1546–1587, 2020.