An imperative programming language characterizing FBQP
Emmanuel Hainry, Romain Pechoux, Mário Silva

To cite this version:
hal-03895106

HAL Id: hal-03895106
https://inria.hal.science/hal-03895106
Submitted on 12 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 capture 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.

Rule for termination (R1)

Let \( \text{proc}_i \sim \text{proc}_j \) mean that \( \text{proc}_i \) and \( \text{proc}_j \) are mutually recursive procedures. The following condition ensures that a program with procedure declarations \( \mathcal{D} \) will always terminate:

\[ \forall \text{Proc} \, \text{proc}_i(\bar{p}) \{ S_i \} \in \mathcal{D}, \quad \forall \text{call} \, \text{proc}_j(\sigma) \in S_i, \quad \text{proc}_i \sim \text{proc}_j \Rightarrow \sigma \text{ is a proper subset of } \bar{p}, \]

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 \( \text{RCalls}(\cdot) \) represent the maximum number of recursive calls for any branch, then

\[ \forall \text{Proc} \, \text{proc}_i(\bar{p}) \{ S_i \} \in \mathcal{D}, \quad \text{RCalls}(\text{proc}_i) \leq 1, \]

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

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.

Building poly-sized circuits

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

\[
\text{Proc } f(\bar{p}) \{ \\
\text{ if } |\bar{p}| > 1 : \\
\text{ Case } C(\bar{p}[*]) = 0 \Rightarrow \\
\text{ call } f(\text{remove}(\bar{p}, 1)) \\
\text{ else Case } C(\bar{p}[*]) = 1 \Rightarrow \\
\text{ call } f(\text{remove}(\bar{p}, \{1, 2\})) \\
\text{ else } \bar{p}[*] = 0 \Rightarrow \\
\text{ call } f(\bar{q}) \\
\}
\]

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 \( q \) of size \( n = 5 \).

References
