An imperative programming language characterizing FBQP

Emmanuel Hainry, Romain Péchoux, 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.
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 algorithm 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 whether a quantum program is quantum polynomial time (QPT) [1].

We introduce an imperative programming language called QPT (Quantum Poly-Time) with recursion rules that capture the power of QPT [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 example procedure: let

Proc f(p) { 
  if |p| > 1 :
    Case C(p[1]) = 0 then
      call f(remove(p, 1))
    else Case C(p[2]) = 1 then
      call f(remove(p, {1, 2}))
    else p[|p|] = U; 
  Case C(p[|p|]) = m then S_m
  call proc(i, σ)
}

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.

Rule for termination (R1)

Let proc_i, 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:

∀ Proc proc_1(p) ∈ D, ∀ call proc_2(σ) ∈ S, proc_i, proc_j ⇒ σ is a proper subset of p, i.e., any call to a mutually recursive function must strictly decrease the amount of qubits available.

We prevent an exponential number of procedure calls by limiting the number of recursive calls in each branch of a Case. Let Rcalls(·) represent the maximum number of recursive calls for any branch, then

∀ Proc proc_i(p) ∈ D, Rcalls(proc_i) ≤ 1,

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

Rule for poly-time (R2)

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.

Proc QFT(p) { 
  p[|p|] = H;
  call Chain(2, p);
  call QFT(remove(p, 1))
}

Proc Chain(m, p) { 
  Case C(p[|p|]) = 1 then p[|p|] = R_m;
  call Chain(m + 1, remove(p, 2))

  call QFT(q)
}

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

References
