Types and Quantum Programming Languages

Quantum algorithms can provide asymptotic improvement over classical algorithms in key problems like integer factorization. However, they require a very different set of abstractions and tools than classical computing. Quantum algorithms typically reason explicitly about quantum gates and wires. If used or composed incorrectly, these elements will not form circuits that can actually be implemented by any quantum computers. The simple act of forking a wire is invalid in the language of quantum circuits. This has led to a rich vein of programming languages research to use types to statically ensure properties of quantum programs, as well as allow abstraction, code reuse and code verification. This paper introduces three, typed quantum programming languages, the classic Quantum Lambda Calculus, and the modern Qwire and Quipper. To give the reader the proper context to understand these languages, core concepts of quantum computing, like qubits, quantum circuits, and entanglement are explained. This paper will also discuss efforts to apply formal methods to each of these languages. A researcher with a strong background in programming languages theory and is familiar with undergraduate level linear algebra should come away from this paper understanding these three languages and with enough background to read quantum programming languages papers.

1 INTRODUCTION

2 CONTRIBUTIONS

- An extended background section, designed to introduce quantum programming to a computer scientist with no background in quantum mechanics.
- A presentation of three quantum programming languages: the Quantum Lambda Calculus, Qwire, and Quipper. For the Quantum Lambda Calculus and Qwire, we present the formal semantics along with the type safety theorems. For Quipper, we will discuss its implementation as a DSL in Haskell, along with its mathematically rigorous core calculus, Proto-Quipper-M.

3 BACKGROUND

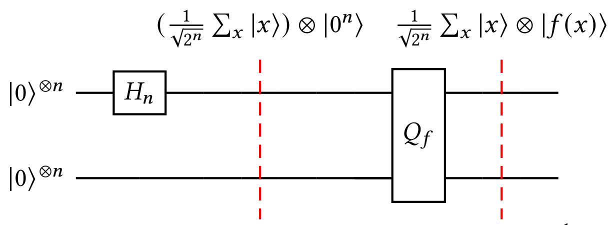
3.1 Qubits and Quantum State Systems

3.2 Quantum Circuits

3.3 QRAM Model

3.4 Running Example: Functional Entanglement

Consider a fixed positive number n, along with a fixed quantum oracle Q_f where $f : 2^n \longrightarrow 2^n$. We can construct the below $2n$ qubit quantum circuit whose inputs are fixed to be the classical state $|0\rangle$ scaled up to n qubits.


$(\frac{1}{\sqrt{2^n}} \sum_x |x\rangle) \otimes |0^n\rangle$
$\frac{1}{\sqrt{2^n}} \sum_x |x\rangle \otimes |f(x)\rangle$
$|0\rangle^{\otimes n}$
H_n
Q_f
$|0\rangle^{\otimes n}$

The circuit diagram marks the intermediate state $(\frac{1}{\sqrt{2^n}} \sum_x |x\rangle) \otimes |0^n\rangle$. The state at this point in the circuit is the tensor product of the state of the upper wire, which is an equal superposition of

Author's address:

2022. XXXX-XXXX/2022/3-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

all classical states, and the state of the lower wire, which is just the classical state of all 0's. We derive this from the behavior from the top row of the extended hadamard gate H_n. The following reasoning shows how this intermediate state transforms into the final state in the diagram.

$$Q_f((\frac{1}{\sqrt{2^n}}\sum_x |x\rangle) \otimes |0^n\rangle) = \tag{1}$$

$$Q_f(\frac{1}{\sqrt{2^n}}\sum_x (|x\rangle \otimes |0^n\rangle)) = \tag{2}$$

$$\frac{1}{\sqrt{2^n}}\sum_x Q_f(|x\rangle) \otimes |0^n\rangle) = \tag{3}$$

$$\frac{1}{\sqrt{2^n}}\sum_x (|x\rangle \otimes |f(x)\rangle) \tag{4}$$

The first two lines are equal because the tensor product is linear in both of its arguments. The next two lines are equal because Q_f is also a linear transformation. And the final two lines are equal by the definition of a quantum oracle.

The final state, $\frac{1}{\sqrt{2^n}}\sum_x(|x\rangle \otimes |f(x)\rangle)$, has the property that that the $x \times y$th coordinate is $\frac{1}{\sqrt{2^n}}$ if $y = f(x)$ and 0 otherwise. Preparing this state is part of several quantum algorithms, including Shor's factorization algorithm. Intuitively, this state is useful because it turns this sequence of qubits into a search space of every input-output pair in f. Even though this system only requires $2n$ qubits to represent, it still encodes all 2^{2n} possible input-output pairs. From this state, we can solve problems by carefully designing quantum circuits to have "good" input-output pairs reinforce each other and "bad" input-output pairs interfere with each other, leading to a final state with a high probability of measuring a "good" input-output pair. This paper will use this circuit as a running example, to see how the different languages each encode this

4 QUANTUM LAMBDA CALCULUS

4.1 Syntax and Type System

4.2 Running Example Implementation

4.3 Operational Semantics And Quantum Closures

5 QWIRE

5.1 Syntax and Type System

5.2 Running Example Implementation

5.3 Operational and Denotational Semantics

5.4 Formal Verification with Qwire

6 QUIPPER

6.1 Haskell Implementation

6.2 Runnning Example Implementation

6.3 Summary of Experimental Analysis

6.4 Proto-Quipper-M