Qurry: A quantum programming language

Lucas Saldyt, *Arizona State University*

***Abstract*—Quantum programming languages are largely still in early development. However, many lack true *abstraction* and are simply proxies to circuit languages. There is reason for this, primarily because the desired semantics of a quantum programming language are not yet completely crystalized. This paper focuses on the creation of "lightweight abstractions," which allow human-level understanding without sacrificing low level control. Additionally, this paper describes a testbed, which is meant to catalyze the development of quantum programming languages.**

***Index Terms*—Quantum Computing, Programming Languages**

I. Introduction

INNOVATION in near-term quantum programming requires the use of lightweight abstractions. Because the field is (comparatively) nascent, transparency is crucial, but so is usability. Of course, there are already plenty of abstractions in quantum programming. Theorists will be familiar with a linear algebra matrix-operator model, which is an abstract scaffold over true hardware implementations, such as superconducting implementations using microwave pulses. In general, quantum algorithms can look over certain implementation details, such as wether the substrate is superconducting or ion trap based. However, there are other implementation details which are far more important, such as the entanglement topology (See google's talk).

Some abstract programming interfaces do exist, such as IBM's qiskit for universal gate set computing and a prolog interface for DWave's adiabatic model. However, an important aspect that these lack is true generality. Instead, there is a general tendency to create a list of algorithmic classes, and simply provide frameworks for each class of algorithm. It may very much be possible that specialized interfaces like this are an efficient method of quantum programming, but creation of a general language is also important

Given these facts, current term quantum programming languages are analogous to classical assembly languages, and the next stage of development is to create the equivalent of C for quantum programming. When C was invented for classical programming, it revolutionized software (and is still used for essentially all modern operating systems, games, and other significant software). C++, despite its age, actually has one of the most innovative communities of any programming language. In particular, the term "lightweight abstraction" originates for Bjarne Stroustrup, the creator of C++. This term is used to describe what Stroustrup considers the fundamental ideaology behind C++.

Since quantum computers are simply special probabilistic computers, Qurry also attempts to create a statistical library for high-level modeling. This is particularly useful in the same way that a classical probabilistic programming language is, namely for modeling anything statistical, and especially for bayesian machine learning. For instance, the R. Tucci and H. group have shown uses for this through their software, Bayesforge. Creating a statistical library will complement these approaches, allowing richer modeling.

TODO

Lastly, the matrix operator model of quantum computing actually lends itself to functional programming paradigms quite nicely. Since each unitary linear operator is essentially a function, constructs such as mapping or composition can be used to create higher order functions (and, in the most abstract terms, a quantum program in total is simply one of these higher order functions). Additionally, the general inability to query state mid-operations also lends itself to a pure functional paradigm. Thus, Qurry draws a large influence from Haskell, as well as LISP (and Clojure in particular,which, like C++, has an incredible innovative community).

Additionally, it is necessary to rapidly test new ideas in quantum programming from the bottom up and simply collect data on them, as opposed to architecting a top-down "perfect" language. As can be seen in the natural language attempt to create Esperanto as a universal language, this is unlikely to work. In general, evolution will create a much more robust system. Thus, Qurry offers a framework in which programmers can easily create new syntax features. This framework works similarly to macros in LISP, but has a simplified interface as well.

II. Conclusion

In the creation of a Qurry and its corresponding framework, it is hoped that this will aid the development of quantum algorithms, as algorithm designers will have a new, richer, more abstract vocabulary with which to express themselves. To recap, this goal is approached in the following N ways. By introduction of lightweight abstractions from the C++ school of thought, efficient and transparent programming interfaces are created. Through specialized libraries, Qurry can claim to be a generalized library, while still offering powerful sub-frameworks for specific tasks. With functional programming paradigms, Qurry can move towards higher levels of abstraction as the semantics of quantum programming become better understood. Lastly, by creating a rapid prototyping framework, new language features can be developed in a bottom-up style, which will allow Qurry to be created naturally, instead of artificially.

Acknowledgment

The author would like to thank Dr. Will Zeng of Rigetti computing, an organizer of the Unitary Fund, Dr. Ajay Bansal of Arizona State University, PLoS and other donators to the unitary fund, and ASU's FURI program.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

Lucas Saldyt is currently a student researcher at Arizona State University, and has previously worked for Sandia National Laboratories and Los Alamos National Laboratories as a student intern in the quantum computing department.