A formalization of the Quipper quantum programming language

Neil J. Ross

Dalhousie University, Halifax, Canada
neil.jr.ross@dal.ca

Quipper is a programming language for quantum computation ([1], [2]). At the moment, Quipper is implemented as an embedded language, whose host language is Haskell. This means that Quipper can be seen as a collection of predefined Haskell functions and data types, together with a preferred style of writing embedded programs, called an idiom.

Developing a programming language as an embedded language has many advantages. In particular, it allows for the rapid implementation of a large-scale system. This is one of the reasons why Quipper is already a rich language. However, there are also disadvantages to Quipper being an embedded language. One of these is that the Haskell type system, while providing many type-safety properties, is not in general strong enough to ensure full type-safety of the quantum programs. In the current Quipper implementation, it is therefore the programmer's responsibility to ensure that quantum components are plugged together in physically meaningful ways. This means that certain types of programming errors will not be prevented by the compiler. In the worst case, this may lead to ill-formed output or run-time errors.

In this talk, we present a quantum programming language which we call *Proto-Quipper*. This language formalizes a fragment of Quipper and is defined as a typed lambda calculus equipped with a reduction strategy. It is a type-safe language in the sense that it enjoys the usual Subject Reduction and Progress properties. Proto-Quipper provides a foundation for the development of a stand-alone (i.e., non-embedded) version of Quipper. It is designed to "enforce the physics", in the sense that it would detect, at compile-time, programming errors that could lead to ill-formed or undefined circuits.

In designing the Proto-Quipper language, our approach was to start with a limited, but still expressive, fragment of the Quipper language and make it completely type-safe. This fragment will serve as a robust basis for future language extensions. The idea is to eventually close the gap between Proto-Quipper and Quipper by extending Proto-Quipper with one feature at a time while retaining type-safety.

Our main inspiration for the design of Proto-Quipper is the *quantum lambda calculus* of [3]. One of the main differences between Quipper and the quantum lambda calculus is that Quipper acts as a circuit description language. It provides the ability to treat circuits as data, and to manipulate them as a whole. For example, Quipper has operators for reversing circuits, decomposing them into gate sets, etc. By contrast, the quantum lambda calculus only manipulates qubits and all quantum operations are immediately carried out on a quantum device, not stored for symbolic manipulation.

We therefore extend the quantum lambda calculus with the minimal set of features that makes it Quipper-like. The current version of Proto-Quipper is designed to:

- incorporate Quipper's ability to generate and act on quantum circuits, and
- provide a linear type system to guarantee that the produced circuits are physically meaningful (in particular, properties like no-cloning are respected).

To achieve these goals, we extend the types of the quantum lambda calculus with a type $Circ(T, U)$ of circuits, and add constant terms to capture some of Quipper's circuit-level operations, like reversing. The execution of Proto-Quipper programs is formalized as a reduction relation on pairs $[C, t]$ of a term t of the language and a so-called circuit state C. The state C represents the circuit currently being built. The reduction is defined as a rewrite procedure on such pairs, with the state being affected by redexes involving quantum constants.

This is joint work with Henri Chataing and Peter Selinger.

References

[1] A.S. Green, P. Lefanu Lumsdaine, N.J. Ross, P. Selinger, and B. Valiron. An introduction to quantum programming in quipper. In *Proceedings of the 5th International Conference on Reversible Computation (RC 2013), Victoria, BC, Canada*, volume 7948, pages 110–124. Springer Lecture Notes in Computer Science, 2013.

[2] A.S. Green, P. Lefanu Lumsdaine, N.J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. In *Proceedings of the 34th Annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2013), Seattle*, volume 48(6), pages 333–342, 2013.

[3] P. Selinger and B. Valiron. Quantum lambda calculus. In S. Gay and I. Mackie, editors, *Semantic Techniques in Quantum Computation*, pages 135–172. Cambridge University Press, 2009.