Verified Optimization in a Quantum Intermediate Representation

Kesha Hietala Robert Rand Shih-Han Hung
Xiaodi Wu Michael Hicks
University of Maryland, College Park, USA
{kesha, rrand, shung, xwu, mwh}@cs.umd.edu

We present sQIRe, a low-level language for expressing and formally verifying quantum programs. sQIRe uses a global register of quantum bits. Doing so allows easy compilation to and from existing "quantum assembly" languages and simplifies the verification process. We demonstrate the power of sQIRe as a compiler intermediate representation of quantum programs by verifying a number of useful optimizations, and we demonstrate sQIRe's use as a tool for general verification by proving several quantum programs correct.

The full paper appears as https://arxiv.org/abs/1904.06319.
Our code is available at https://github.com/inQWIRE/SQIRE.

Programming quantum computers, at least in the near term, will be challenging. Qubits will be scarce, and the risk of decoherence means that gate pipelines will need to be short. These limitations encourage the development of sophisticated algorithms and clever optimizations that are likely to have mistakes. For example, Nam et al. discovered mistakes in both the theory and implementation of optimizing circuit transformations they developed [3], and found that the optimization library they compared against sometimes produced incorrect results [11]. Unfortunately, we cannot apply standard software assurance techniques to address these challenges: Unit testing and debugging are infeasible due to both the indeterminacy of quantum algorithms and the substantial expense involved in executing or simulating them.

To address these challenges, we can apply rigorous *formal methods* to the development of quantum programs and programming languages. These methods aim to ensure mathematically-proved correctness of code by construction. A notable success of formal methods for classical computing is CompCert [9], a *certified compiler* for C programs. CompCert is written and *proved correct* using the Coq proof assistant [4]. CompCert includes sophisticated optimizations whose proofs of correctness are verified to be valid by Coq's type checker. An experimental evaluation of CompCert's reliability provided strong evidence of the validity of Coq's proof checking: While bug finding tools found hundreds of defects in the GCC and LLVM C compilers, no bugs were found in CompCert's verified core [19].

In this paper, we introduce a simple quantum language we call sQIRe (pronounced "squire") that can be used as an intermediate representation in a certified compiler for quantum programs. sQIRe is implemented in Coq, on top of the libraries developed for the $\mathcal{Q}$WIRE circuit language [12, 14, 16]. sQIRe makes a number of simplifying assumptions compared to $\mathcal{Q}$WIRE that make verification of sQIRe programs and program transformations easier. However, sQIRe's simple design sacrifices some desirable features of high-level languages, such as variable binding to support easy compositionality. This is an acceptable tradeoff because sQIRe is intended to be used as an intermediate representation, produced as the output of compiling a higher-level language. Furthermore, low-level languages (like OpenQASM [5] and QUIL [17], both similar to sQIRe) are more practical for programming near-term devices, which must be aware of both the number of qubits available and their connectivity [13].

DRAFT April 16, 2019

```
Definition a : ℕ := 0.
Definition b : ℕ := 1.

Definition bell00 : ucom := H a; CNOT a b.

Definition encode (b1 b2 : 𝔹): ucom :=
    (if b2 then X a else uskip);
    (if b1 then Z a else uskip).

Definition decode : ucom := CNOT a b; H a.

Definition superdense (b1 b2 : 𝔹) := bell00 ; encode b1 b2; decode.
```

Figure 1: sQIRe program for the unitary portion of the superdense coding algorithm.

As an example of a sQIRe program, consider the definition of superdense coding given in Figure 1. Superdense coding is a protocol that allows a sender to transmit two classical bits, b_1 and b_2, to a receiver using a single quantum bit. In the sQIRe version of superdense coding, `encode` is defined as a Coq function that takes two Boolean values and returns a circuit. This shows that although sQIRe's design is simple, we can still express quantum programs that use features like classical control by using help from the host language, Coq.

Although we primarily designed sQIRe to be a target for compilation, we can also prove properties about sQIRe programs directly, since programs and their semantics are embedded in Coq. For example, we can prove that the result of evaluating the program `superdense b1 b2` on an input state consisting of two qubits initialized to zero is the state $|b_1, b_2\rangle$. In our development, we write this as follows:

```
Lemma superdense_correct : ∀ b1 b2,
    ⟦superdense b1 b2⟧ × | 0,0 ⟩ = | b1,b2 ⟩.
```

Here, ⟦`superdense b1 b2`⟧ is the denotation of the `superdense b1 b2` circuit (which is a unitary matrix).

In our full paper [8] we give additional examples of verifying correctness of sQIRe programs. In particular, we prove that the sQIRe program to prepare the GHZ state indeed produces the correct state, and show the correctness of the Deutsch-Jozsa algorithm and quantum teleportation. To be precise, the GHZ state and Deutsch-Jozsa proofs verify properties of *families* of sQIRe programs, parameterized by the number of qubits in the circuit.

However, our primary goal is not to prove that sQIRe programs are correct, but that transformations of sQIRe programs are *semantics-preserving*, meaning that the transformation does not change the denotation of the program. When a transformation is semantics-preserving, we say that it is *sound*.

As an example, consider the "skip removal" function shown in Figure 2. To show that this function is sound, we prove the following lemma, where equivalence (≡) relates the denotations of two programs:

```
Lemma rm_uskips_sound : ∀ c, c ≡ (rm_uskips c).
```

The proof is straightforward and relies on the identities `uskip; c ≡ c` and `c; uskip ≡ c` (which are easily proven in our development). We can also prove other useful properties about `rm_uskips`. For example, we can prove that the output of `rm_uskips` is either a single skip operation, or contains no skip operations. We can also prove that the output of `rm_uskips` contains no more basic operations than the input program.

In our full paper [8], we verify soundness of a more realistic optimization, inspired by a recent optimizer [11], which removes unnecessary X gates from a unitary program. We also consider a transfor-

```
Fixpoint rm_uskips (c : ucom) : ucom :=
  match c with
  | c1 ; c2 ⇒ match rm_uskips c1, rm_uskips c2 with
              | uskip, c2' ⇒ c2'
              | c1', uskip ⇒ c1'
              | c1', c2'   ⇒ c1'; c2'
              end
  | c'      ⇒ c'
  end.
```

Figure 2: Skip removal optimization.

mation that turns general sQIRe programs into sQIRe programs that can run on a linear nearest neighbor architecture. We verify that this transformation is sound, and that it produces a program that satisfies the architecture's constraints.

The problem of proving quantum program optimizations correct has previously been considered in the context of the ZX calculus [6], but, as far as we are aware, our sQIRe-based transformations are the first certified-correct optimizations applied to a realistic quantum circuit language. Amy et al. [2] developed a proved-correct optimizing compiler from source Boolean expressions to reversible circuits, but did not handle general quantum programs. Rand et al. [15] developed a similar compiler for quantum circuits but without optimizations. Prior low-level quantum languages [5, 17] have not been developed with verification in mind, and prior circuit-level optimizations [1, 7, 11] have not been formally verified. Some recent efforts have examined using formal methods to prove properties of high-level quantum programs, for instance using Hoare logics [20, 10, 18]. These lines of work are largely complementary to ours—a property proved of a source program is provably preserved by a certified compiler. At the same time, sQIRe can also be used to directly prove properties about quantum programs by reasoning about their semantics.

Our work on sQIRe constitutes a step toward developing a full-scale verified compiler toolchain. Next steps include developing certified transformations from high-level quantum languages to sQIRe, implementing more powerful program transformations, and verifying more sophisticated quantum algorithms.

References

[1] Matthew Amy, Dmitri Maslov & Michele Mosca (2013): *Polynomial-Time T-Depth Optimization of Clifford+T Circuits Via Matroid Partitioning*. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, doi:10.1109/TCAD.2014.2341953.

[2] Matthew Amy, Martin Roetteler & Krysta M. Svore (2017): *Verified compilation of space-efficient reversible circuits*. In: *Proceedings of the 28th International Conference on Computer Aided Verification (CAV 2017)*, Springer. Available at `https://www.microsoft.com/en-us/research/publication/verified-compilation-of-space-efficient-reversible-circuits/`.

[3] Andrew Childs (2018): Private Communication.

[4] Coq Development Team (2019): *The Coq Proof Assistant Reference Manual, Version 8.9*. Electronic resource, available from `https://coq.inria.fr/refman/`.

[5] Andrew W. Cross, Lev S. Bishop, John A. Smolin & Jay M. Gambetta (2017): *Open Quantum Assembly Language*. *arXiv e-prints*:arXiv:1707.03429.

[6] Andrew Fagan & Ross Duncan (2018): *Optimising Clifford Circuits with Quantomatic*. In: *Proceedings of the 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Nova Scotia, 3-7 June 2018*.

[7] Luke Heyfron & Earl T. Campbell (2017): *An Efficient Quantum Compiler that reduces T count*. *Quantum Science and Technology* 4, doi:10.1088/2058-9565/aad604.

[8] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu & Michael Hicks (2019): *Verified Optimization in a Quantum Intermediate Representation*. *arXiv e-prints*:arXiv:1904.06319.

[9] Xavier Leroy et al. (2004): *The CompCert verified compiler*. *Development available at http://compcert. inria. fr* 2009.

[10] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying & Naijun Zhan (2019): *Quantum Hoare Logic*. *Archive of Formal Proofs*. `http://isa-afp.org/entries/QHLProver.html`, Formal proof development.

[11] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs & Dmitri Maslov (2018): *Automated optimization of large quantum circuits with continuous parameters*. *npj Quantum Information* 4(1), p. 23, doi:10.1038/s41534-018-0072-4. Available at `https://doi.org/10.1038/s41534-018-0072-4`.

[12] Jennifer Paykin, Robert Rand & Steve Zdancewic (2017): *QWIRE: A Core Language for Quantum Circuits*. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, ACM, New York, NY, USA, pp. 846–858, doi:10.1145/3009837.3009894.

[13] John Preskill (2018): *Quantum Computing in the NISQ era and beyond*. *Quantum* 2, p. 79, doi:10.22331/q-2018-08-06-79. Available at `https://doi.org/10.22331/q-2018-08-06-79`.

[14] Robert Rand (2018): *Formally Verified Quantum Programming*. Ph.D. thesis, University of Pennsylvania.

[15] Robert Rand, Jennifer Paykin, Dong-Ho Lee & Steve Zdancewic (2018): *ReQWIRE: Reasoning about Reversible Quantum Circuits*. In: *Proceedings of the 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Nova Scotia, 3-7 June 2018*.

[16] Robert Rand, Jennifer Paykin & Steve Zdancewic (2017): *QWIRE Practice: Formal Verification of Quantum Circuits in Coq*, pp. 119–132. doi:10.4204/EPTCS.266.8. Available at `https://doi.org/10.4204/EPTCS.266.8`.

[17] Robert S. Smith, Michael J. Curtis & William J. Zeng (2016): *A Practical Quantum Instruction Set Architecture*. *arXiv e-prints*:arXiv:1608.03355.

[18] Dominique Unruh (2019): *Quantum Hoare Logic with Ghost Variables*. *arXiv preprint arXiv:1902.00325*.

[19] Xuejun Yang, Yang Chen, Eric Eide & John Regehr (2011): *Finding and Understanding Bugs in C Compilers*. In: *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, ACM, pp. 283–294.

[20] Mingsheng Ying (2011): *Floyd–hoare logic for quantum programs*. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 33(6), p. 19.