Verified Optimization in a Quantum Intermediate Representation

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks

University of Maryland, College Park

DEPARTMENT OF COMPUTER SCIENCE

JOINT CENTER FOR QUANTUM INFORMATION AND COMPUTER SCIENCE

Abstract

We present a simple quantum language we call SQIRE (pronounced "squire") that can be used as an **intermediate representation** (IR) in a **certified compiler** for quantum programs. SQIRE is implemented in Coq [2], on top of libraries developed for the QWIRE language [9]. This allows us to **formally verify** properties of SQIRE programs and program transformations. We demonstrate the power of SQIRE as a compiler intermediate representation by verifying a number of useful program transformations. For example, we verify soundness of an optimization that removes unnecessary X gates from a unitary program. We also consider a transformation that turns general SQIRE programs into SQIRE programs that can run on a linear nearest neighbor architecture.

The full paper [5] and code [3] are available online.

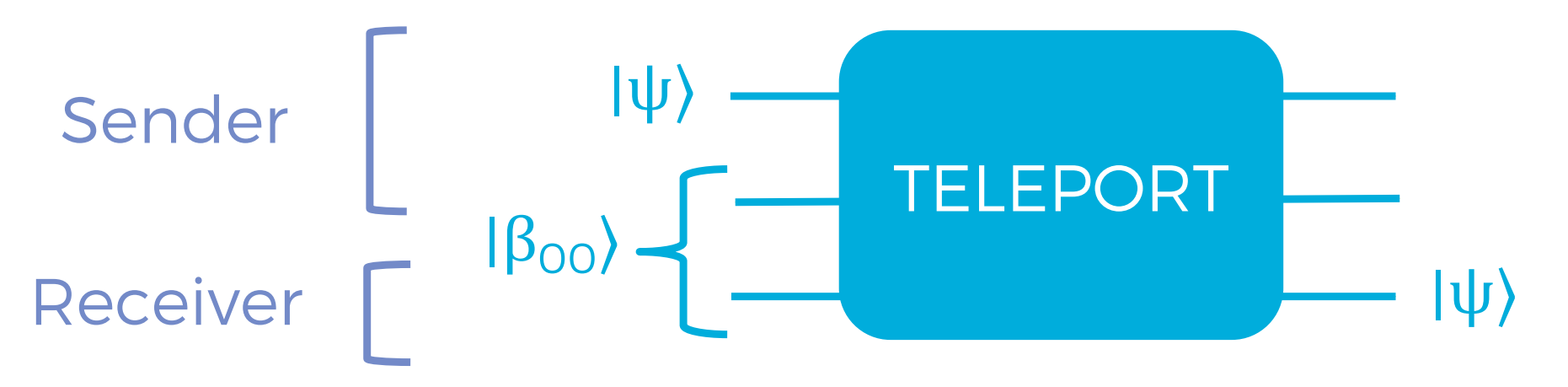
Formal Verification

The process of mathematically proving the correctness of a piece of software is known as *formal verification*. Formal verification is particularly useful in the field of quantum computing, where standard software assurance techniques such as unit testing and runtime debugging are infeasible.

Examples of formal verification that have been applied in the field of quantum computing include:
- Model checking
- Equivalence checking
- Program logics
- Direct proofs about semantics
- Diagrammatic reasoning

For the most part, these techniques are used to prove that a quantum program satisfies some specification. For example, given a program describing the quantum teleportation protocol, the goal may be to prove that the program correctly "teleports" any input qubit to the receiver.

Example. Prove that the following holds for any state $|\psi\rangle$.



Another useful application of formal verification, which has received relatively little attention from the quantum formal verification community, is proving the correctness of program *transformations*. Verifying program transformations allows the construction of *certified compilers*, which are compilers that guarantee that the executable code they output behaves as specified by the input source program. A notable example of a certified compiler (for classical programs) is CompCert [1], an optimizing compiler for C proved correct using the Coq proof assistant.

SQIRE: A Small Quantum IR

SQIRE supports five quantum programming constructs: skip, sequencing, unitary application, measurement of a single qubit, and resetting a single qubit to a fixed basis state.

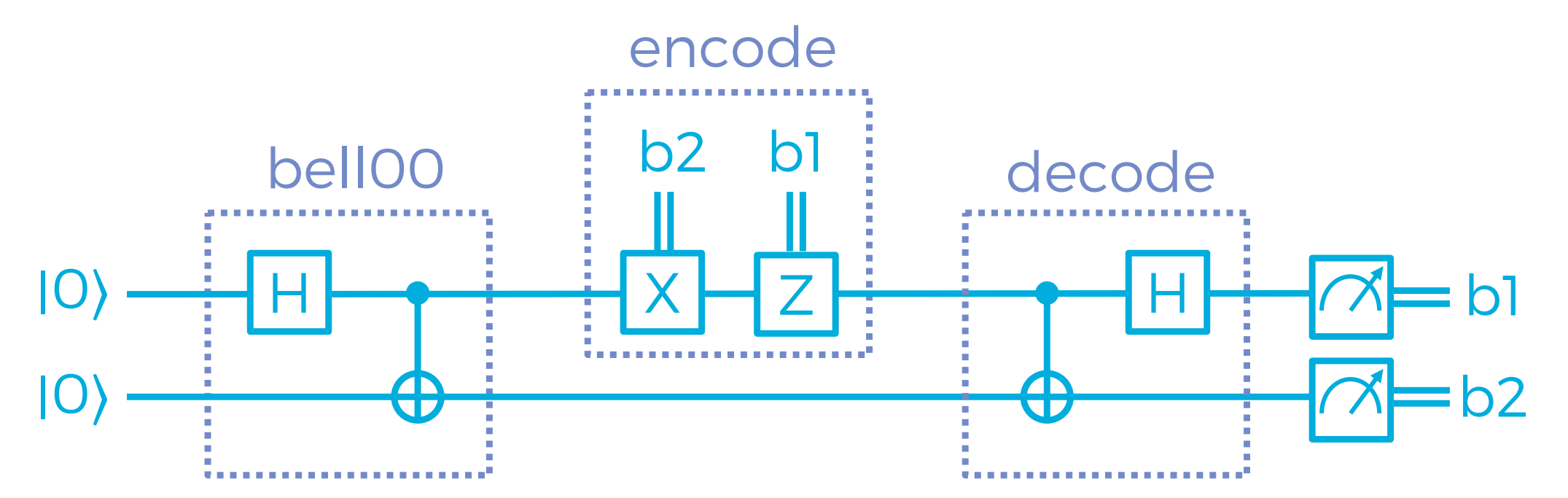
$$
\begin{aligned}
P &::= \mathit{skip} \\
&\mid P_1; P_2 \\
&\mid U\ q_1 \ldots q_n \\
&\mid \mathit{meas}\ q \\
&\mid \mathit{reset}\ q \\
U &::= H \mid X \mid Y \mid Z \mid R_\phi \mid \mathit{CNOT}
\end{aligned}
$$

For simplicity, we support a fixed set of gates. This set can be extended in our implementation, or new gates can be defined in terms of built-in gates. For example, we define the swap operation as follows.

```
Definition SWAP a b := CNOT a b; CNOT b a; CNOT a b.
```

We can then state and prove properties about the semantics of the defined operations. For example, we can prove that the SWAP program swaps its arguments, as intended.

Example. Superdense coding is a protocol that allows a sender to transmit two classical bits, b1 and b2, to a receiver using a single quantum bit. The circuit and SQIRE program for superdense coding are shown below.



```
Definition bell00 := H 0; CNOT 0 1.

Definition encode (b1 b2 : 𝔹) :=
    (if b2 then X 0 else skip);
    (if b1 then Z 0 else skip).

Definition decode := CNOT 0 1; H 0.

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

Although SQIRE was designed to be used as an intermediate representation, we can also prove properties about SQIRE programs directly, since these 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 |b1, b2⟩. We write this as follows.

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

In our development, we also verify the correctness of *n*-qubit GHZ state preparation, quantum teleportation, and the *n*-qubit Deutsch-Jozsa algorithm.

Verified Optimization

Because near-term quantum machines will only be able to perform small computations before decoherence takes effect, compilers for quantum programs must apply sophisticated optimizations to reduce resource usage. These optimizations can be complicated to implement and are vulnerable to programmer error. It is thus important to verify that the implementations of these optimizations are correct.

In general, we will be interested in proving that a transformation is *semantics-preserving*, meaning that the transformation does not change the behavior of the program.

Example. The following optimization removes skip operations from a program.

```
Fixpoint rm_skips c :=
  match c with
  | c1 ; c2 ⇒ match rm_skips c1, rm_skips c2 with
             | skip, c2' ⇒ c2'
             | c1', skip ⇒ c1'
             | c1', c2'  ⇒ c1'; c2'
             end
  | _ ⇒ c
  end.
```

To ensure that this transformation is semantics-preserving, we prove the following lemma.

```
Lemma rm_skips_sound : ∀ c, c ≡ (rm_skips c).
```

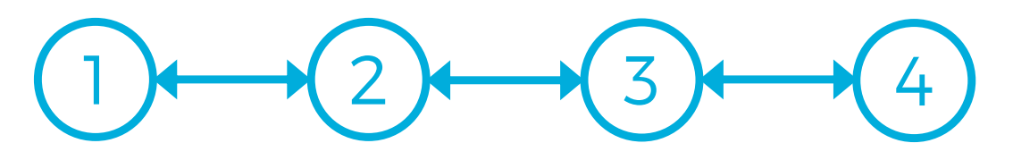
In our development, we also verify soundness of an optimization from a recent circuit optimizer [7] that removes unnecessary X gates from a unitary program.

Verified Circuit Mapping

Similar to how optimization aims to reduce resource usage to make programs more feasible to run on near-term machines, *circuit mapping* aims to address the connectivity constraints of near-term machines. Circuit mapping algorithms take as input an arbitrary program and output a program that respects the connectivity constraints of some underlying architecture.

As a simple illustration of certified circuit mapping, we map circuits to a linear nearest neighbor (LNN) architecture. We map a program to this architecture by adding SWAP operations before and after every CNOT so that the target and control are adjacent when the CNOT is performed, and are returned to their original positions before the next operation.

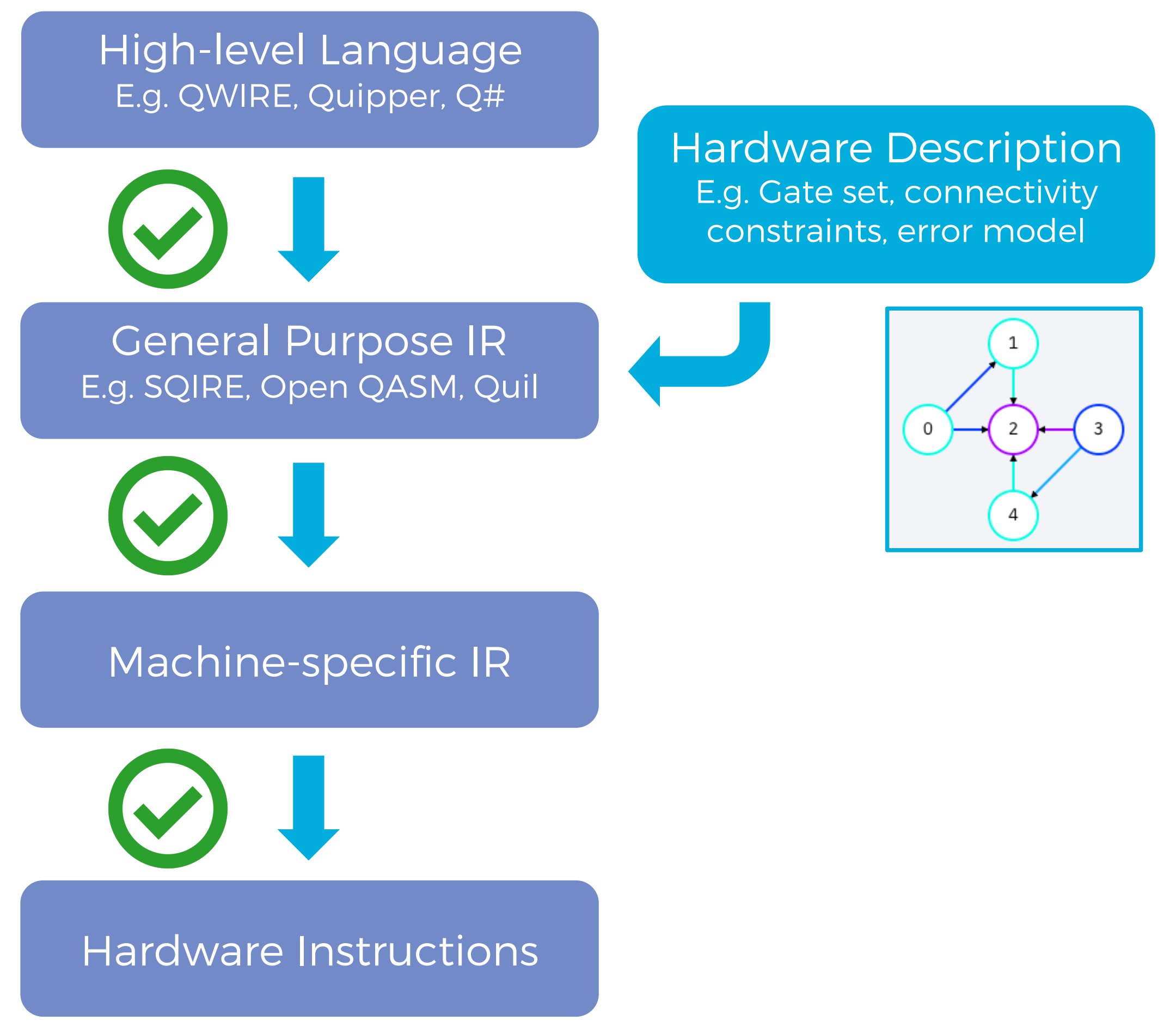
Example. On the following 4-qubit LNN architecture, "CNOT 1 3" becomes "SWAP 1 2; CNOT 2 3; SWAP 1 2".



We have proven that this transformation is sound, and that the output program satisfies the LNN constraint.

Verified Compilation Stack

Our work is a first step towards a certified compiler for quantum programs [8]. In the long term, we envision a fully-verified compilation stack from high-level quantum languages to hardware instructions, as shown below.



Ongoing work:
- Additional verified optimizations and mapping algorithms taking inspiration from existing compilation infrastructures like Qiskit [4] and ScaffCC [6].
- Verified circuit synthesis.
- Verified compilation of Boolean oracles.
- Verified translation from QWIRE to SQIRE.

Acknowledgments

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research Quantum Testbed Pathfinder Program under Award Number DE-SC0019040.

References

[1] *The CompCert Verified Compiler*. Available at http://compcert.inria.fr/.
[2] *The Coq Proof Assistant*. Available at https://coq.inria.fr/.
[3] *SQIRE Development*. Available at https://github.com/inQWIRE/SQIRE.
[4] *Qiskit*. Available at https://qiskit.org/.
[5] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. *Verified Optimization in a Quantum Intermediate Representation*. arXiv:1904.06319.
[6] Ali Javadi Abhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. *ScaffCC: Scalable Compilation and Analysis of Quantum Programs*. Parallel Computing 45. 2015.
[7] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. *Automated Optimization of Large quantum Circuits with Continuous Parameters*. npj Quantum Information 4(1). 2018.
[8] Robert Rand, Kesha Hietala, and Michael Hicks. *Formal Verification vs. Quantum Uncertainty*. SNAPL 2019.
[9] Robert Rand, Jennifer Paykin, and Steve Zdancewic. *QWIRE Practice: Formal Verification of Quantum Circuits in Coq*. QPL 2017.