Verified Optimization in a Quantum Intermediate Representation

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

University of Maryland, College Park

Verified compiler stack

- End goal: *verified compiler stack* for quantum programs

High-level Language
E.g. QWIRE, Quipper, Q#

Means that we've *formally verified* that the transformation is semantics-preserving.

General Purpose IR
E.g. OpenQASM, Quil

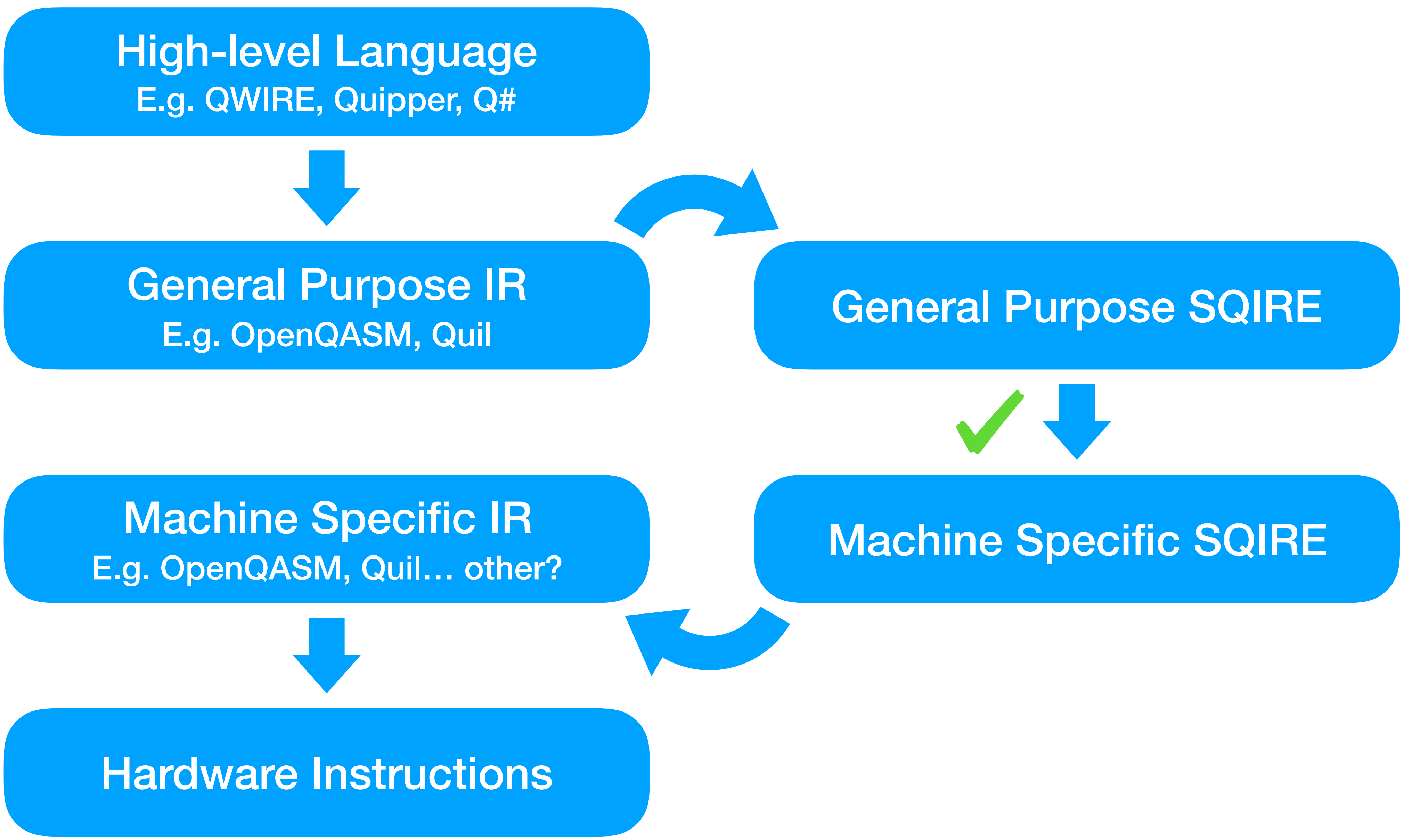
Optimization
Circuit synthesis
Circuit mapping
…

Machine Specific IR
E.g. OpenQASM, Quil… other?

Hardware Instructions

Verified compiler stack

- End goal: *verified compiler stack* for quantum programs



Outline

- **SQIRE**
- Verified compilation
- General verification
- Ongoing work

SQIRE

- A **S**mall **Q**uantum **I**ntermediate **RE**presentation
- Design goals:
  - As simple as possible
  - Expressive enough to describe interesting algorithms
  - Similar to existing quantum IRs (e.g. OpenQASM, Quil)
  - *Easy to use in proofs*

Unitary programs

- Syntax

$$
\begin{aligned}
P \;&\rightarrow skip \\
&\mid P_1;\, P_2 \\
&\mid U\ q_1\ \ldots\ q_n
\end{aligned}
$$

$$U \;\rightarrow H \mid X \mid Y \mid Z \mid R_\phi \mid CNOT$$

  - Qubits are referred to by indices into a *global register*

- Semantics

$$
\begin{aligned}
[\![skip]\!]^{dim} &= I_{2^{dim}} \\
[\![P_1;\, P_2]\!]^{dim} &= [\![P_2]\!]^{dim} \times [\![P_1]\!]^{dim} \\
[\![U\ q_1\ \ldots\ q_n]\!]^{dim} &= \begin{cases} ueval(U,\ q_1 \ldots q_n) & \text{well-typed} \\ 0_{2^{dim}} & \text{otherwise} \end{cases}
\end{aligned}
$$

Example

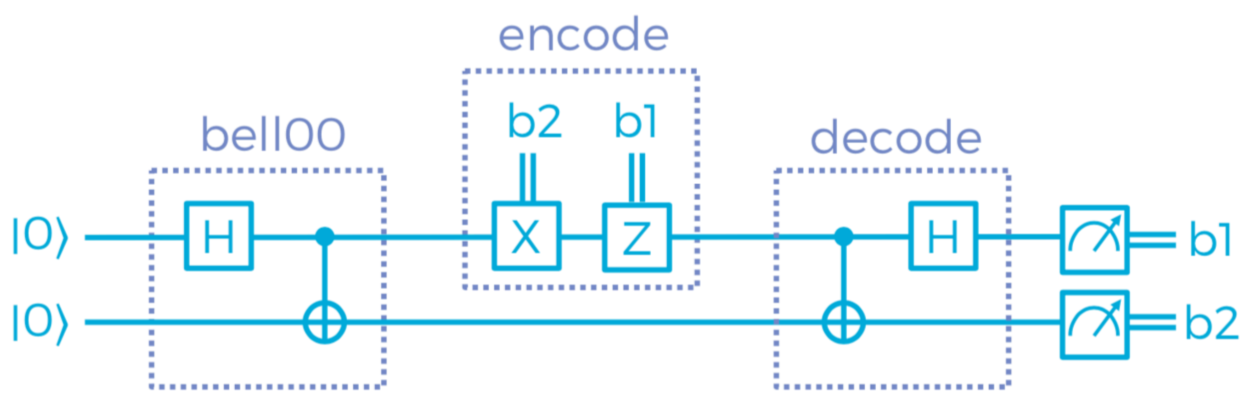
- Superdense coding protocol in SQIRE

```
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.
```



- Correctness:

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

Full SQIRE language

- Includes initialization and measurement
- Two different semantics:
    - Density matrix semantics (used by QWIRE, QHL)

Full SQIRE language

- Includes initialization and measurement
- Tw
  - I

$$[\![skip]\!]^{dim}(\rho) = \rho$$

$$[\![P_1;\ P_2]\!]^{dim}(\rho) = ([\![P_2]\!]^{dim} \circ [\![P_1]\!]^{dim})(\rho)$$

$$[\![U\ q_1\ \ldots\ q_n]\!]^{dim}(\rho) = \begin{cases} ueval(U) \times \rho \times ueval(U)^\dagger & \text{well-typed} \\ 0_{2^{dim}} & \text{otherwise} \end{cases}$$

$$[\![meas\ q]\!]^{dim}(\rho) = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0| + |1\rangle_q\langle 1|\rho|1\rangle_q\langle 1|$$

$$[\![reset\ q]\!]^{dim}(\rho) = |0\rangle_q\langle 0|\rho|0\rangle_q\langle 0| + |0\rangle_q\langle 1|\rho|1\rangle_q\langle 0|$$

Full SQIRE language

- Includes initialization and measurement
- Two different semantics:
  - Density matrix semantics (used by QWIRE, QHL)
  - Non-deterministic semantics (allows states to be represented as vectors)

Full SQIRE language

- Includes initialization and measurement

```
Inductive nd_eval {dim : ℕ} : com → Vector (2^dim) → Vector (2^dim) → ℙ :=
  | nd_app : ∀ n (u : Unitary n) (l : list ℕ) (ψ : Vector (2^dim)),
      app u l / ψ ⇓ ((ueval dim u l) × ψ)
  | nd_meas0 : ∀ n (ψ : Vector (2^dim)),
      let ψ' := pad n dim |0⟩⟨0| × ψ in
      norm ψ' ≠ 0 →
      meas n / ψ ⇓ ψ'
  ...

where "c '/' ψ '⇓' ψ'" := (nd_eval c ψ ψ').
```

Full SQIRE language

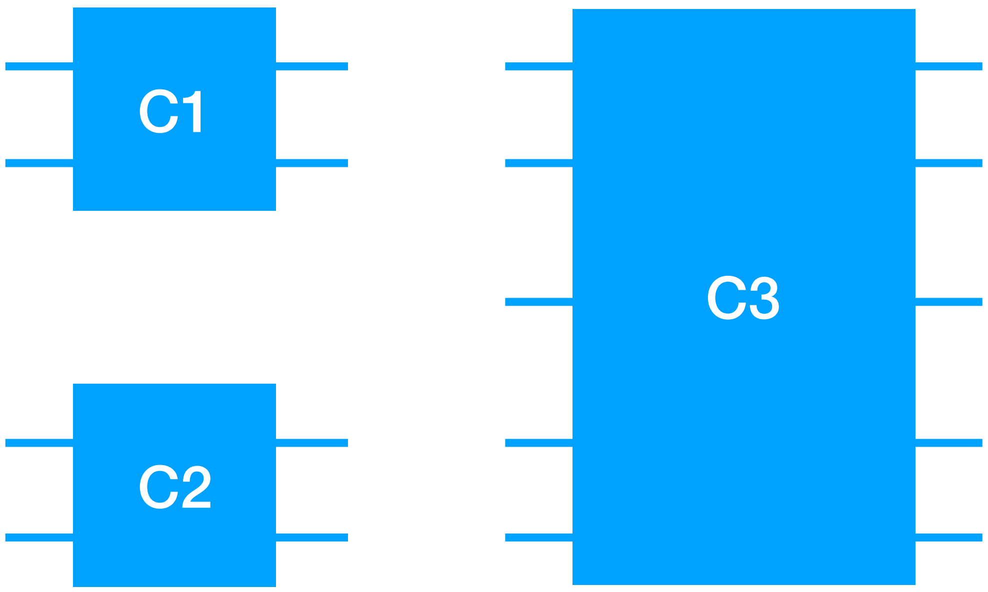
- Includes initialization and measurement
- Two different semantics:
  - Density matrix semantics (used by QWIRE, QHL)
  - Non-deterministic semantics (allows states to be represented as vectors)
  - Example proof of teleport protocol using both semantics in our paper

SQIRE vs. QWIRE

- SQIRE is built on the Coq libraries developed for QWIRE
- Why a new language?
  - Verification of QWIRE program transformations is complicated by the use of higher-order abstract syntax
  - Referencing qubits by natural numbers indexing into a global register makes denotation function simpler
  - Other simplifying assumptions (only multi-qubit gate is CNOT, no dynamic lifting)
- However, by using a global register we sacrifice compositionality

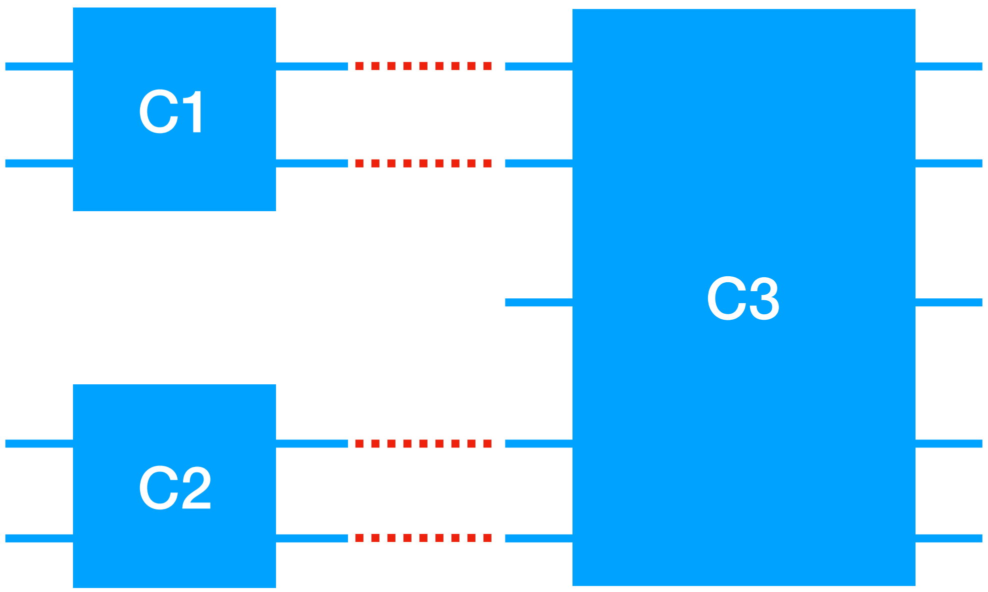
Composition in SQIRE

- Composition in SQIRE requires renaming qubits



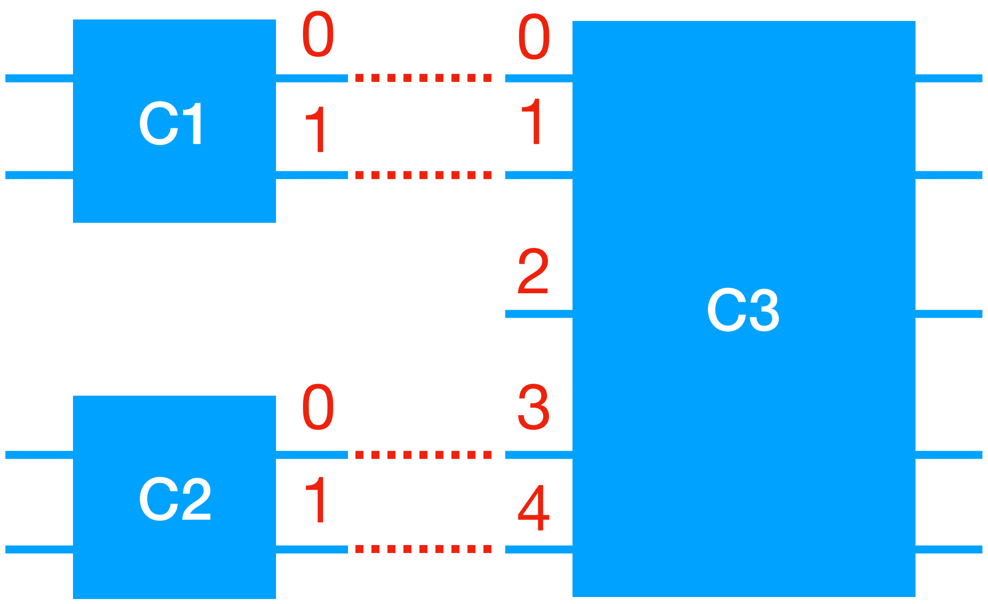
Composition in SQIRE

- Composition in SQIRE requires renaming qubits



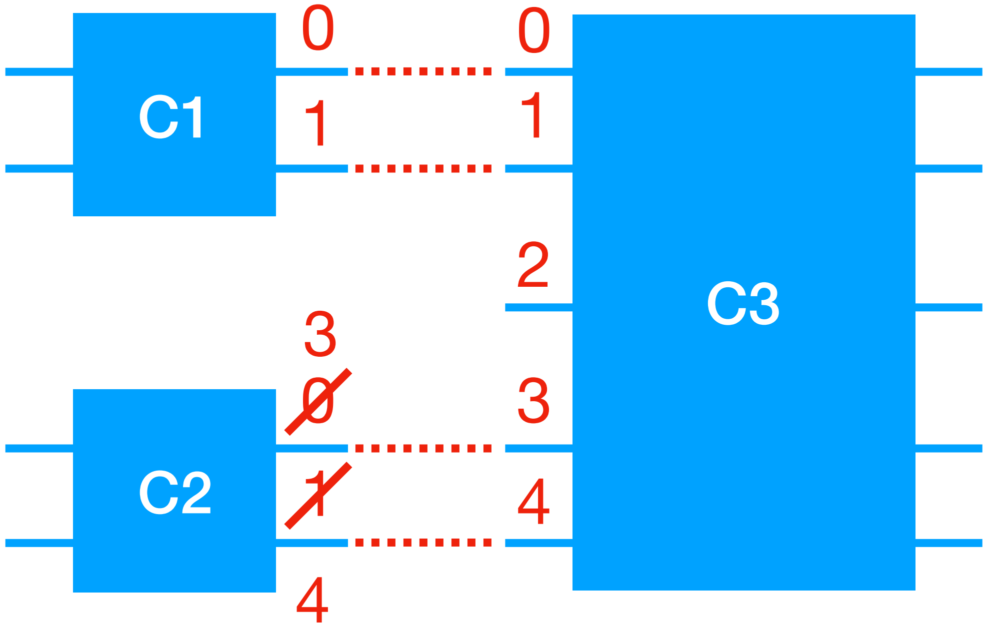
Composition in SQIRE

- Composition in SQIRE requires renaming qubits



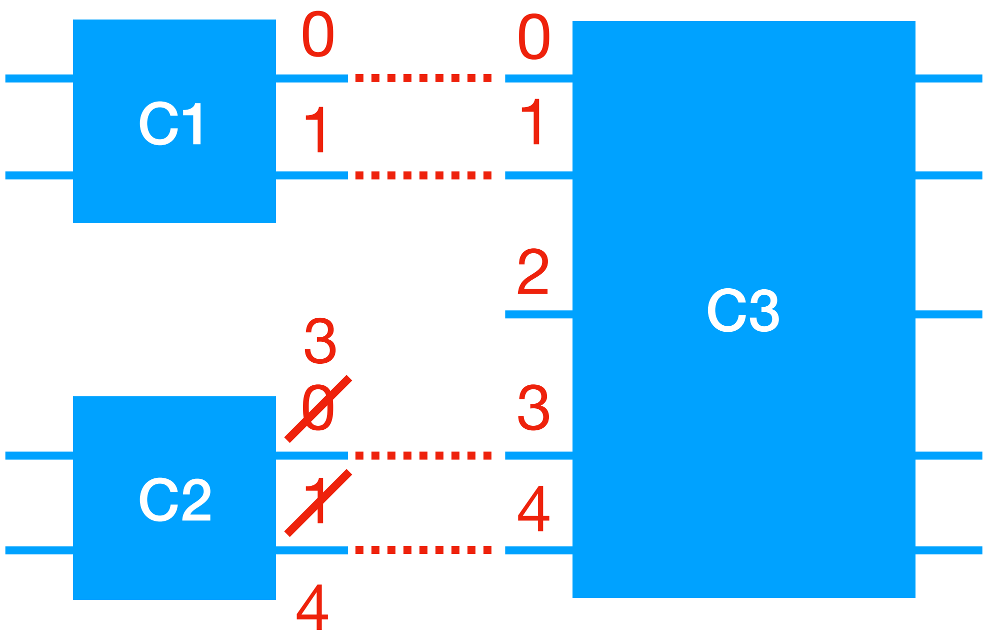
Composition in SQIRE

- Composition in SQIRE requires renaming qubits



Composition in SQIRE

- Composition in SQIRE requires renaming qubits



  - This is tedious to do by hand, but we don't expect to write SQIRE programs manually

Outline

- SQIRE
- **Verified compilation**
- General verification
- Ongoing work

Verified transformations

- In general, we want to verify that the input to a given transformation is semantically equivalent to the output

- We say that program c_1 is *equivalent* to (≡) c_2 if for all *dim*, $[\![\texttt{c1}]\!]^{dim} = [\![\texttt{c2}]\!]^{dim}$.
  - Some useful equivalences:

```
Lemma useq_assoc : ∀ c1 c2 c3, ((c1 ; c2) ; c3) ≡ (c1 ; (c2 ; c3)).

Lemma useq_congruence : ∀ c1 c1' c2 c2',
    c1 ≡ c1' →
    c2 ≡ c2' →
    c1 ; c2 ≡ c1' ; c2'.

Lemma uskip_id_l : ∀ c, (uskip ; c) ≡ c.

Lemma X_CNOT_comm : ∀ c t, (X t; CNOT c t) ≡ (CNOT c t ; X t).
```

Skip removal

- Simple optimization that removes “skip” constructs

```
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.
```

Skip removal

- Simple optimization that removes “skip” constructs
  - rm_skips preserves semantics

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

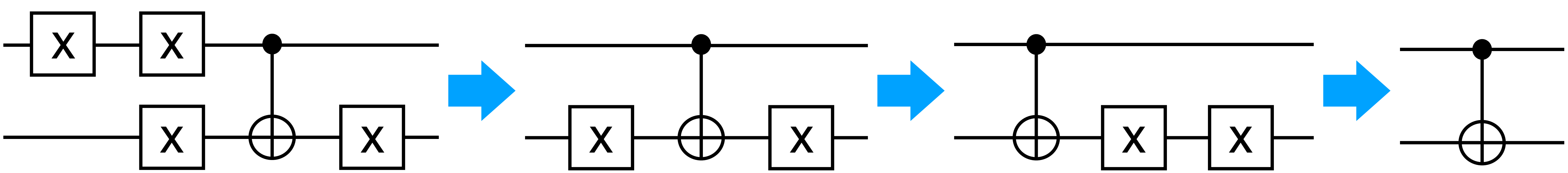
  - rm_skips removes skips

```
Inductive skip_free : ucom → ℙ :=
  | SF_seq : ∀ c1 c2, skip_free c1 → skip_free c2 → skip_free (c1; c2)
  | SF_app : ∀ n l (u : Unitary n), skip_free (uapp u l).

Lemma rm_skips_correct : ∀ c,
    (rm_skips c) = skip ∨ skip_free (rm_skips c).
```

X propagation

- Pre-processing step from a recent circuit optimizer[1]
    - Idea is to cancel redundant X gates

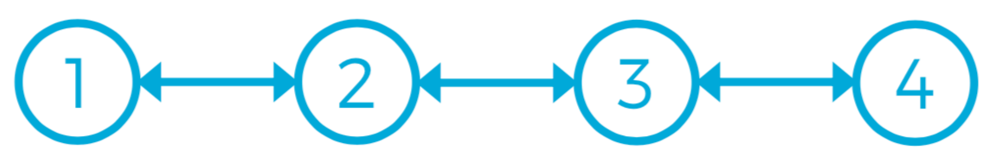


- We have proved that this transformation is sound

[1] Nam et al. "Automated optimization of large quantum circuits with continuous parameters" npj Quantum Information 4. 2018.

LNN mapping

- Convert a SQIRE program into a program that can be run on a linear nearest neighbor (LNN) architecture
    - Consider a naïve mapping strategy that adjusts qubit placement using swaps before & after each CNOT
    - E.g. `CNOT 1 3 → SWAP 1 2; CNOT 2 3; SWAP 1 2`



- The transformation is sound, and the output program satisfies the LNN constraint

Outline

- SQIRE
- Verified compilation
- **General verification**
- Ongoing work

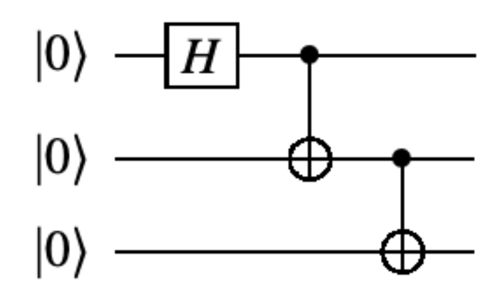
GHZ state preparation

- *n*-qubit GHZ state preparation in SQIRE

```
Definition ghz (n : ℕ) : Vector (2 ^ n) :=
  match n with
  | 0 ⇒ I 1
  | S n' ⇒ 1/√2 .* (nket n |0⟩) .+ 1/√2 .* (nket n |1⟩)
  end.
```

$$|\mathrm{GHZ}\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}).$$

```
Fixpoint GHZ (n : ℕ) : ucom :=
  match n with
  | 0 ⇒ uskip
  | 1 ⇒ H 0
  | S n' ⇒ GHZ n'; CNOT (n'-1) n'
  end.
```

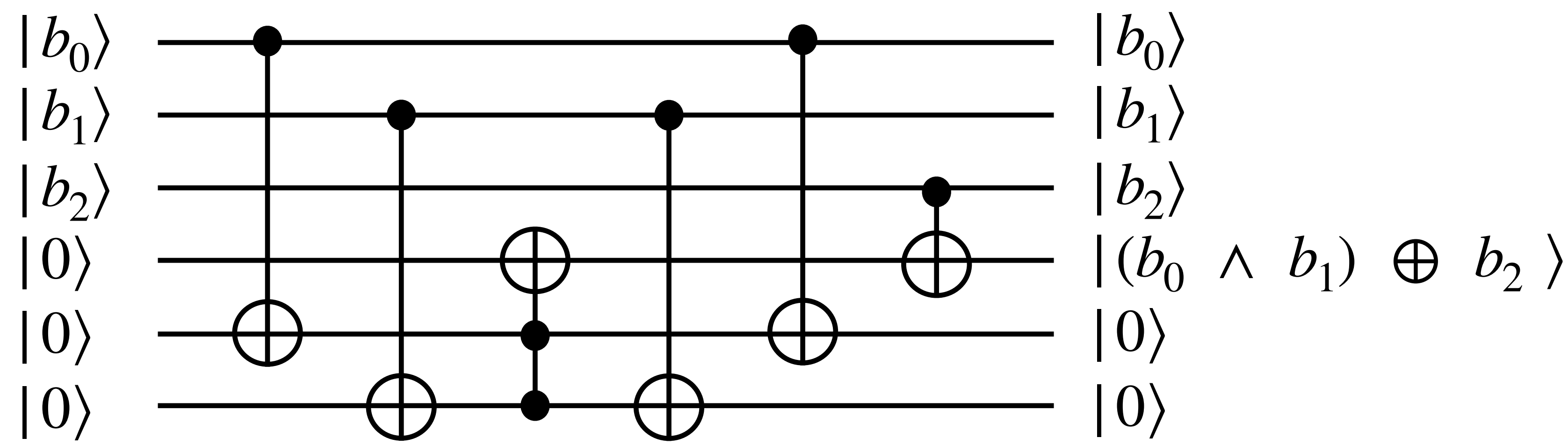


  - Correctness

```
Theorem ghz_correct : ∀ n : ℕ, [[GHZ n]]ⁿ × nket n |0⟩ = ghz n.
```

Boolean oracle compilation

$(b_0 \wedge b_1) \oplus b_2$



- Verify the compilation of boolean formulas into quantum circuits with X, CNOT, and Toffoli gates
  - Previously done by Amy et al.[2] and Rand et al.[3]
  - The output should be logically correct, and all ancilla should be returned to the zero state

[2] Amy et al. "Verified compilation of space-efficient reversible circuits" CAV 2017.
[3] Rand et al. "ReQWIRE: Reasoning about reversible quantum circuits" QPL 2018.

Outline

- SQIRE
- Verified compilation
- General verification
- **Ongoing work**

Ongoing work

- Full-featured optimizer for quantum circuits, in the style of Nam et al., Qiskit, etc.

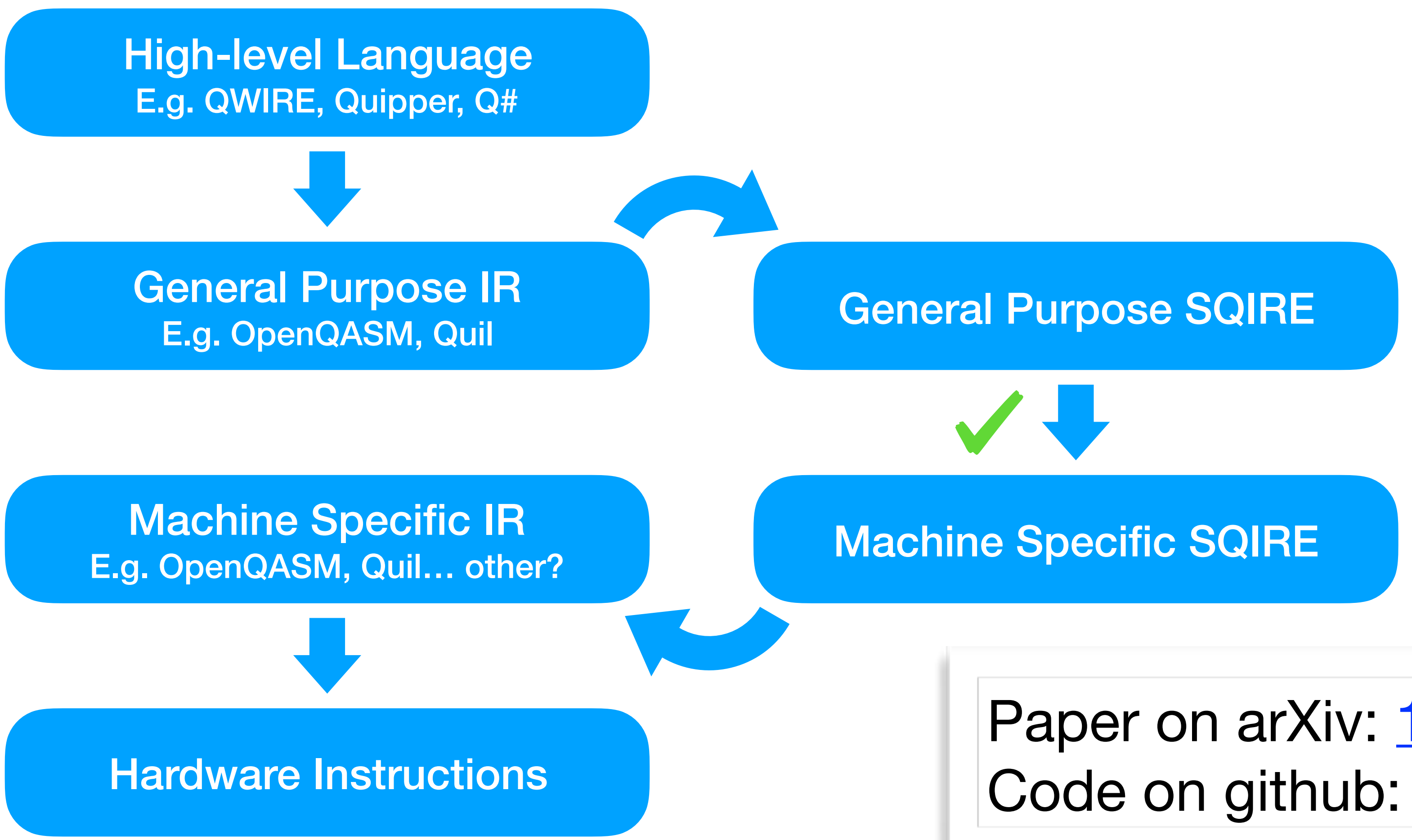
http://www.cs.umd.edu/~rrand/vqc/

- Support for mapping to arbitrary architectures

- SQIRE for teaching: see [Verified Quantum Computing](http://www.cs.umd.edu/~rrand/vqc/)
    - This has led to several proof engineering questions:
        - What program representation should we use?
        - What mathematical representation (e.g. of tensor product) should we use?
        - What automation do we need?

Conclusions

- Initial progress on a *verified compiler stack* for quantum programs
  - Presented an IR for quantum programs, embedded in Coq
  - Verified simple optimization and mapping algorithms



Paper on arXiv: [1904.06319]
Code on github: [inQWIRE/SQIRE]