QIRs for Formal Verification

Kesha Hietala
University of Maryland, College Park

QCE21 Workshop on Quantum Intermediate Representations
October 22nd, 2021

About me

- 6th year PhD student at the University of Maryland, College Park
  - On the job market!
- Interested broadly in **formal verification**, **compilers**, and **static analysis**
- For my PhD I've been applying formal verification to the quantum software toolchain
- Spent last summer interning for Microsoft (remotely) thinking about how to apply formal verification to Q#

This talk

Motivation

SQIR — a QIR designed for verification

VOQC — a verified compiler

IRs for oracles

Concluding thoughts

Formal verification

- *Formal verification* is the process of proving that a program matches a specification (e.g., in a *proof assistant*)
  - More expensive than testing, but provides stronger correctness guarantees

- When should you use formal verification?
  - Code has an impact on human well-being (avionics, crypto)
  - Code is “trusted” (compilers, operating systems)
  - Code is hard to test (compilers, *quantum*)
  - Running incorrect code wastes significant resources (*quantum*)

Formal verification *for quantum*

- Quantum computing is an interesting application area for formal verification
  - Simulation is expensive
  - Hardware is noisy
  - Can’t inspect (i.e., measure & print) intermediate state
  - Not intuitive (entanglement may lead to unintended state updates)
  - Formal verification provides the possibility for software assurance, *without having to run the software*

hard to test!

- Increasingly popular topic in the academic community: Quantum Hoare Logic (TOPLAS 2012), QWIRE (POPL 2017), Quantum Relational Hoare Logic (POPL 2019), VOQC (POPL 2021), SQIR (ITP 2021), QBRICKS (ESOP 2021)

our work

This talk

Motivation

SQIR — a QIR designed for verification

VOQC — a verified compiler

IRs for oracles

Concluding thoughts

SQIR

- SQIR is a **S**imple **Q**uantum **I**ntermediate **R**epresentation for expressing quantum circuits + libraries for reasoning about quantum programs in the *Coq Proof Assistant*
- Presented as the intermediate representation of a verified compiler (à la CompCert) at POPL 2021 (arxiv:1912.02250)
- Presented as a *source* language for verified quantum programming at ITP 2021 (arxiv:2010.01240)
- Code available at github.com/inQWIRE/SQIR

Unitary SQIR

- Semantics parameterized by *gate set G* and *dimension d of a global register*

$$U \;::=\; U_1;\ U_2 \mid G\ q \mid G\ q_1\ q_2$$

E.g. $apply_1(X, q, d) = I_{2^q} \otimes \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes I_{2^{(d-q-1)}}$

- The denotation (semantics) of *U* is a $2^d \times 2^d$ unitary matrix

$$[\![U_1;\ U_2]\!]_d = [\![U_2]\!]_d \times [\![U_1]\!]_d$$

$$[\![G_1\ q]\!]_d = \begin{cases} apply_1(G_1,\ q,\ d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

$q < d$

$$[\![G_2\ q_1\ q_2]\!]_d = \begin{cases} apply_2(G_2,\ q_1,\ q_2,\ d) & \text{well-typed} \\ 0_{2^d} & \text{otherwise} \end{cases}$$

$q_1 < d \ \wedge \ q_2 < d \ \wedge \ q_1 \neq q_2$

Non-unitary SQIR

- Semantics parameterized by *gate set G* and *dimension d of a global register*

$$P \;::=\; \texttt{skip} \mid P_1;\; P_2 \mid U \mid \texttt{meas}\; q\; P_1\; P_2$$

- The denotation of *P* is a function over $2^d \times 2^d$ density matrices

$$
\begin{aligned}
\{\!|\mathsf{skip}|\!\}_d(\rho) &= \rho \\
\{\!|P_1;\; P_2|\!\}_d(\rho) &= (\{\!|P_2|\!\}_d \circ \{\!|P_1|\!\}_d)(\rho) \\
\{\!|U|\!\}_d(\rho) &= [\![U]\!]_d \times \rho \times [\![U]\!]_d^{\dagger} \\
\{\!|\mathsf{meas}\; q\; P_1\; P_2|\!\}_d(\rho) &= \{\!|P_2|\!\}_d(|0\rangle_q\langle 0| \times \rho \times |0\rangle_q\langle 0|) \\
&\quad + \{\!|P_1|\!\}_d(|1\rangle_q\langle 1| \times \rho \times |1\rangle_q\langle 1|)
\end{aligned}
$$

Standard semantics;
also used in QHL[1]
and QWIRE[2]

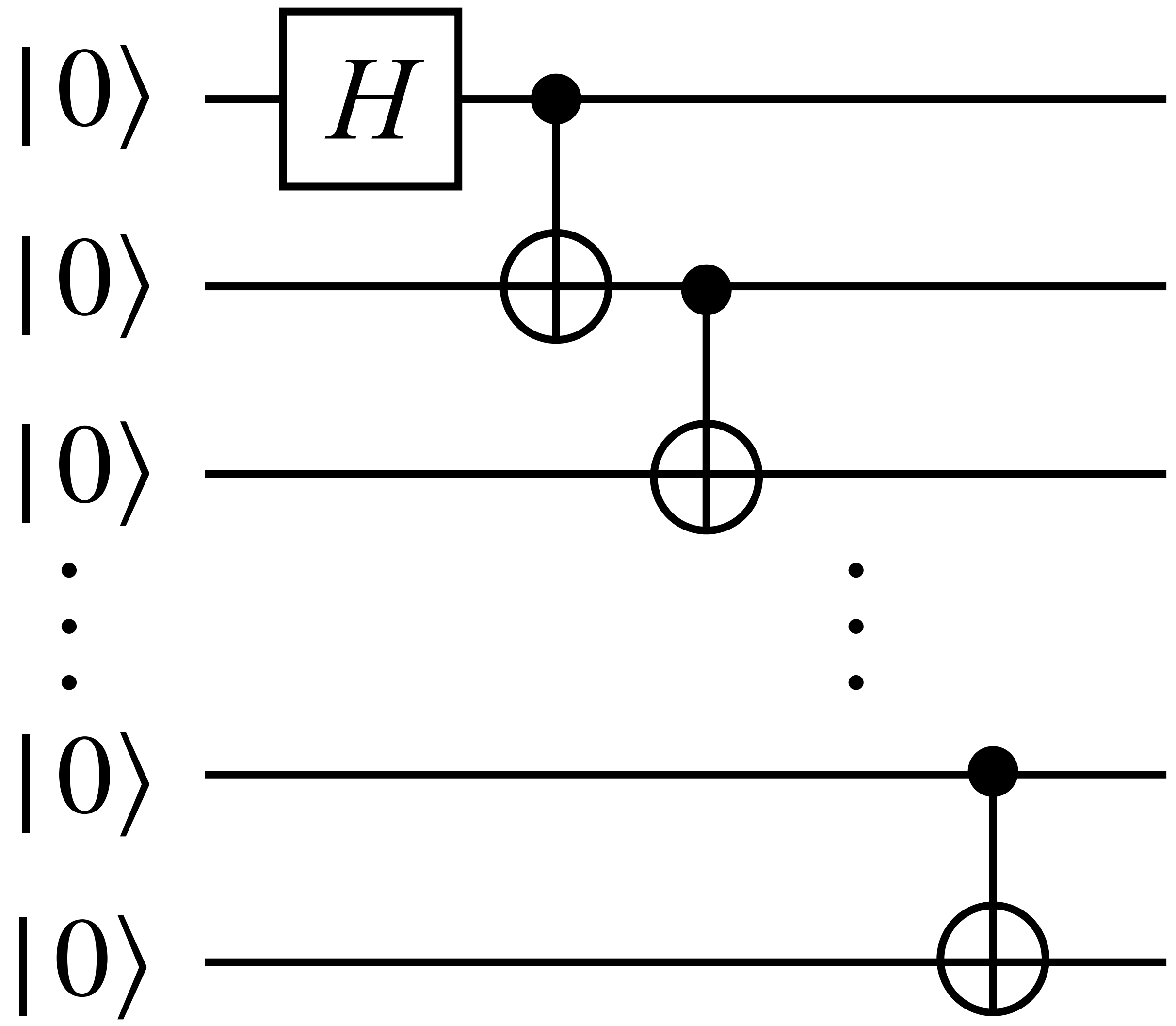
---

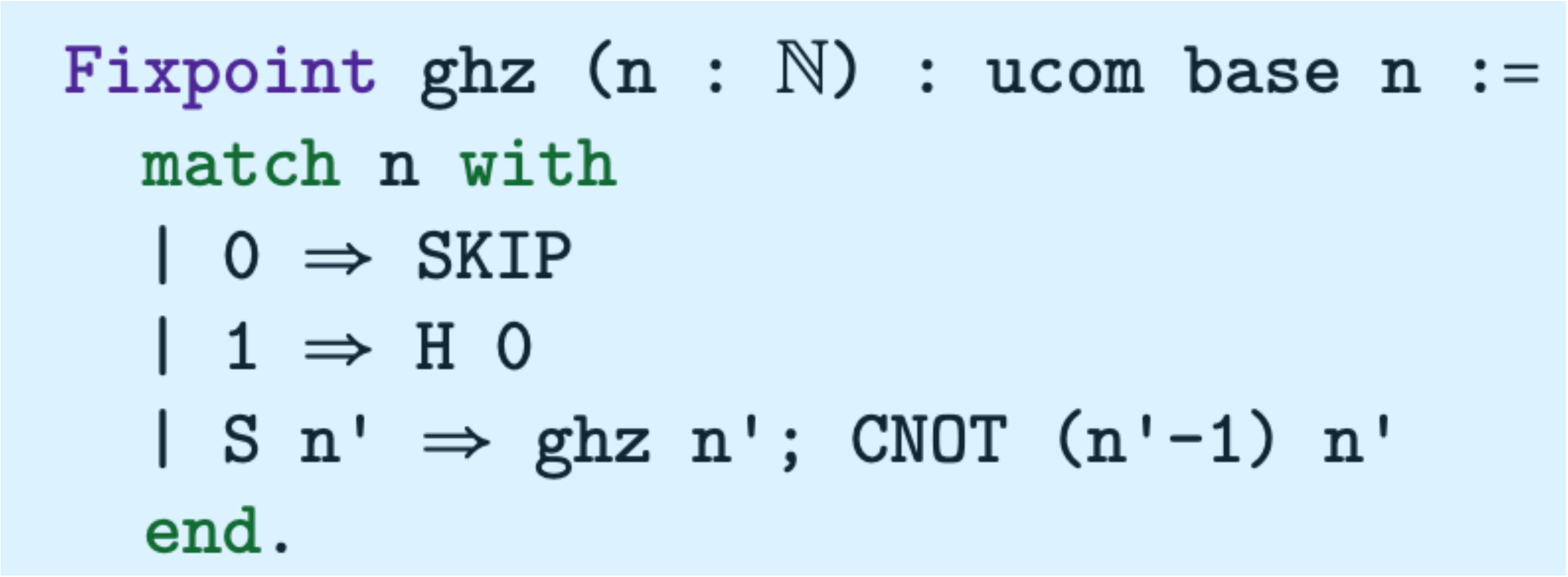
[1] Ying. *Floyd-Hoare logic for quantum programs*. TOPLAS 2012.

[2] Paykin et al. *QWIRE: A core language for quantum circuits*. POPL 2017.

SQIR metaprogramming

- SQIR programs just express circuits. We can express parameterized circuit families using Coq as a meta programming language



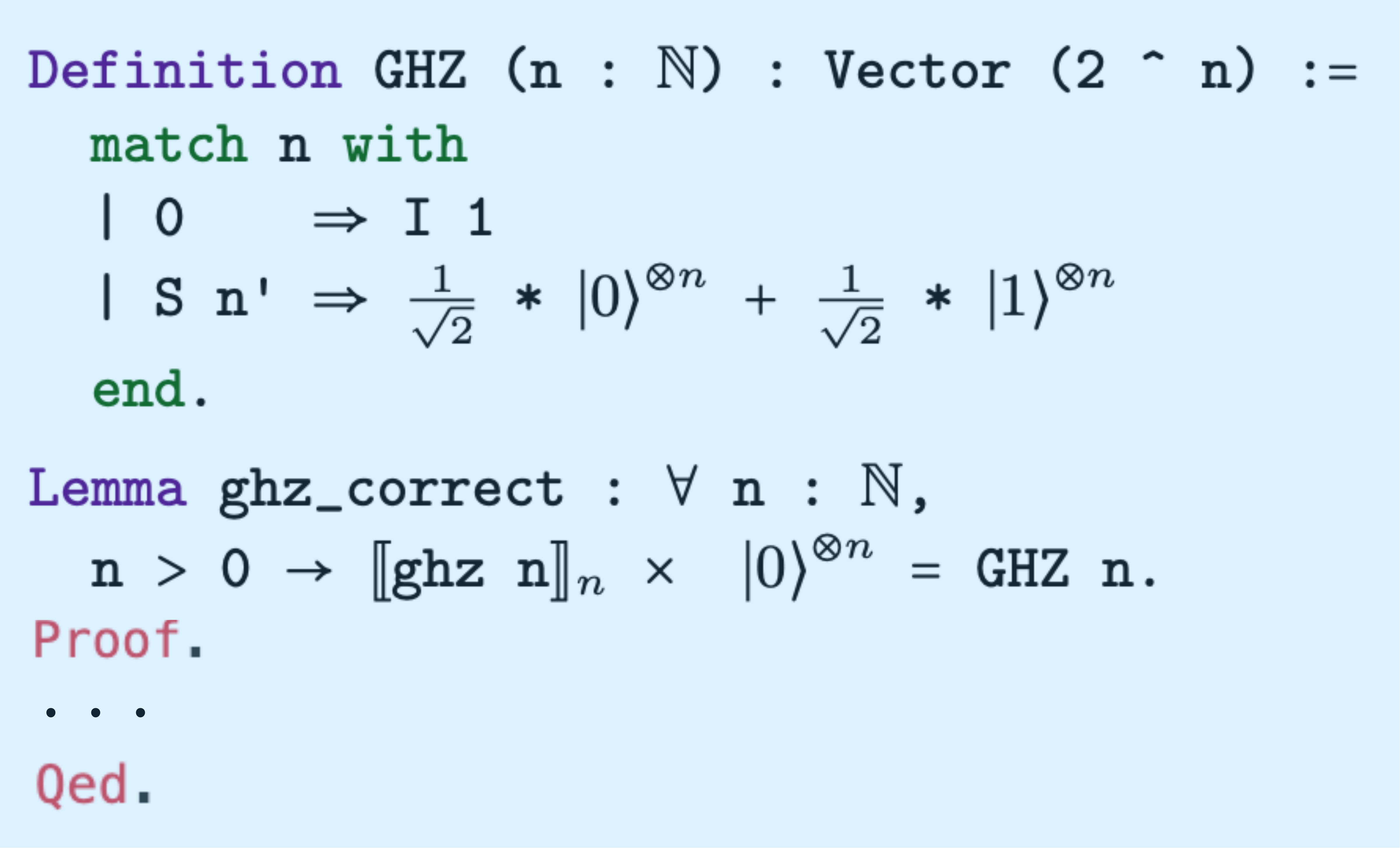


```
Fixpoint ghz (n : ℕ) : ucom base n :=
  match n with
  | 0 ⇒ SKIP
  | 1 ⇒ H 0
  | S n' ⇒ ghz n'; CNOT (n'-1) n'
  end.
```

- The `ghz` Coq function returns a SQIR program (of type `ucom base n`) whose semantics is the n-qubit GHZ state

Proofs of correctness in Coq

- We might like to prove that evaluating `ghz n` on $|0\rangle^{\otimes n}$ produces $|GHZ^n\rangle$
  - where $|GHZ^n\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n})$



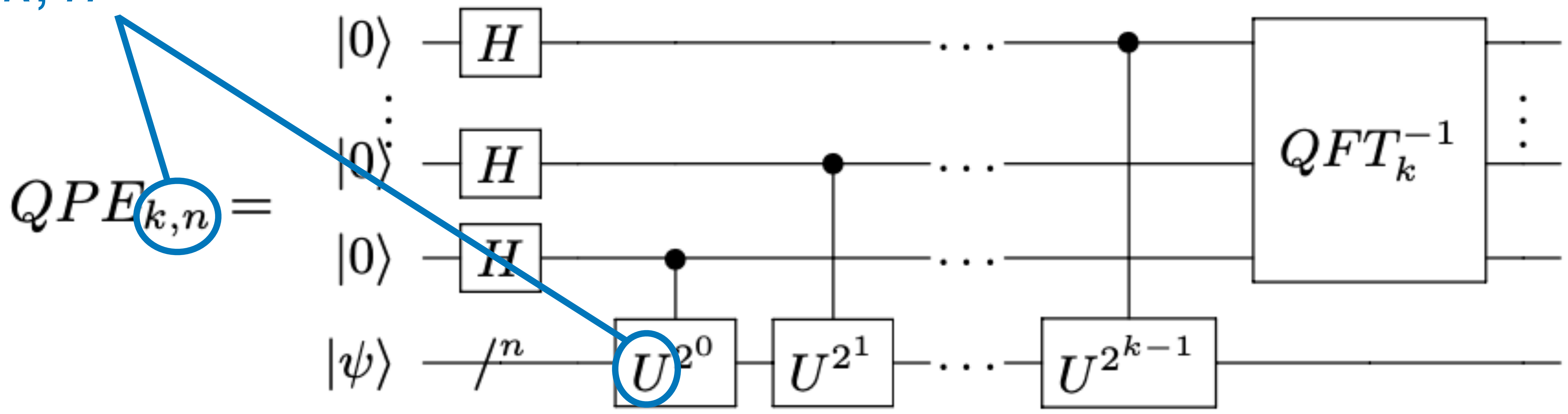
```
Definition GHZ (n : ℕ) : Vector (2 ^ n) :=
  match n with
  | 0    ⇒ I 1
  | S n' ⇒ 1/√2 * |0⟩^⊗n + 1/√2 * |1⟩^⊗n
  end.

Lemma ghz_correct : ∀ n : ℕ,
  n > 0 → ⟦ghz n⟧_n × |0⟩^⊗n = GHZ n.
Proof.
. . .
Qed.
```

Proofs so far

- To date, we have formally verified:
  - Quantum teleportation / superdense coding
  - GHZ state preparation
  - Deutsch-Jozsa algorithm
  - Simon’s algorithm
  - Grover’s search algorithm
  - Quantum phase estimation (key part of Shor’s algorithm)
- These proofs as well as the basic support of SQIR (lemmas, tactics, etc.) constitute about 3500 lines of Coq code

Example: QPE

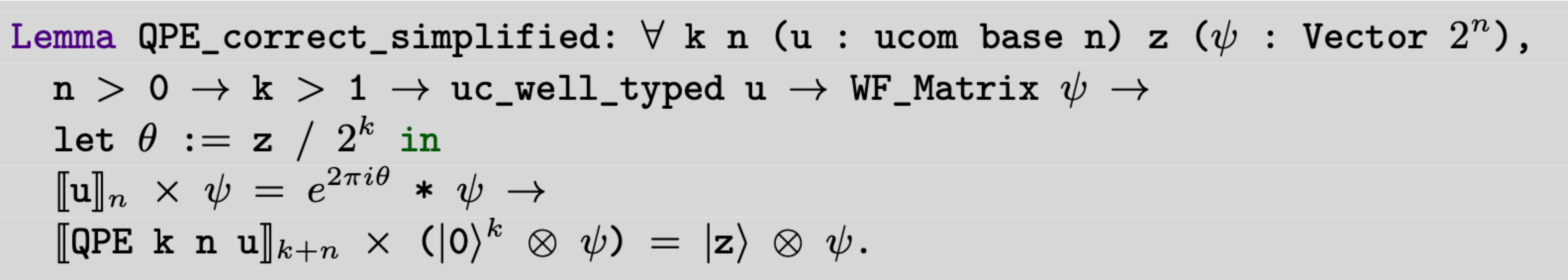


parameterized by *U*, *k*, *n*

- **Q**uantum **P**hase **E**stimation: given a circuit implementing some unitary U and a state $|\psi\rangle$ such that $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, find θ
  - The key “quantum” part of Shor’s factoring algorithm
  - The most sophisticated quantum algorithm verified by any current tool
- The SQIR implementation is 40 lines and the proof is 1000 lines
  - Proof completed in two person-weeks

Example: QPE

- Correctness property in the case where θ can be represented using exactly *k* bits (call this representation *z*):

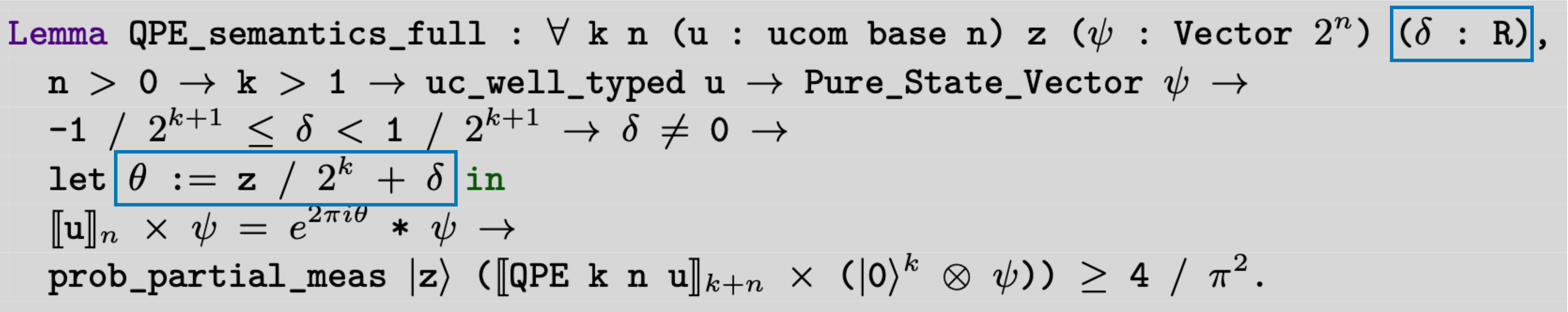


```
Lemma QPE_correct_simplified: ∀ k n (u : ucom base n) z (ψ : Vector 2^n),
  n > 0 → k > 1 → uc_well_typed u → WF_Matrix ψ →
  let θ := z / 2^k in
  ⟦u⟧_n × ψ = e^(2πiθ) * ψ →
  ⟦QPE k n u⟧_(k+n) × (|0⟩^k ⊗ ψ) = |z⟩ ⊗ ψ.
```

- Conclusion says that the running `QPE` on the input $|00...0\rangle \otimes |\psi\rangle$ produces *z* in the first *k* bits

Example: QPE

- If θ cannot be exactly expressed using *k* bits, we get an approximation within $\frac{1}{2^{k+1}}$ of the true value with probability at least $\frac{4}{\pi^2} \approx 0.41$



δ is the error in representing θ

$$
\begin{array}{l}
\texttt{Lemma QPE\_semantics\_full : } \forall\ \texttt{k n (u : ucom base n) z } (\psi \texttt{ : Vector } 2^n)\ \boxed{(\delta \texttt{ : R})},\\
\quad \texttt{n} > 0 \to \texttt{k} > 1 \to \texttt{uc\_well\_typed u} \to \texttt{Pure\_State\_Vector } \psi \to \\
\quad \texttt{-1} / 2^{k+1} \le \delta < 1 / 2^{k+1} \to \delta \neq 0 \to \\
\quad \texttt{let } \boxed{\theta := \texttt{z} / 2^k + \delta}\ \texttt{in} \\
\quad [\![\texttt{u}]\!]_n \times \psi = e^{2\pi i\theta} * \psi \to \\
\quad \texttt{prob\_partial\_meas } |\texttt{z}\rangle\ ([\![\texttt{QPE k n u}]\!]_{k+n} \times (|0\rangle^k \otimes \psi)) \ge 4 / \pi^2.
\end{array}
$$

This talk

Motivation

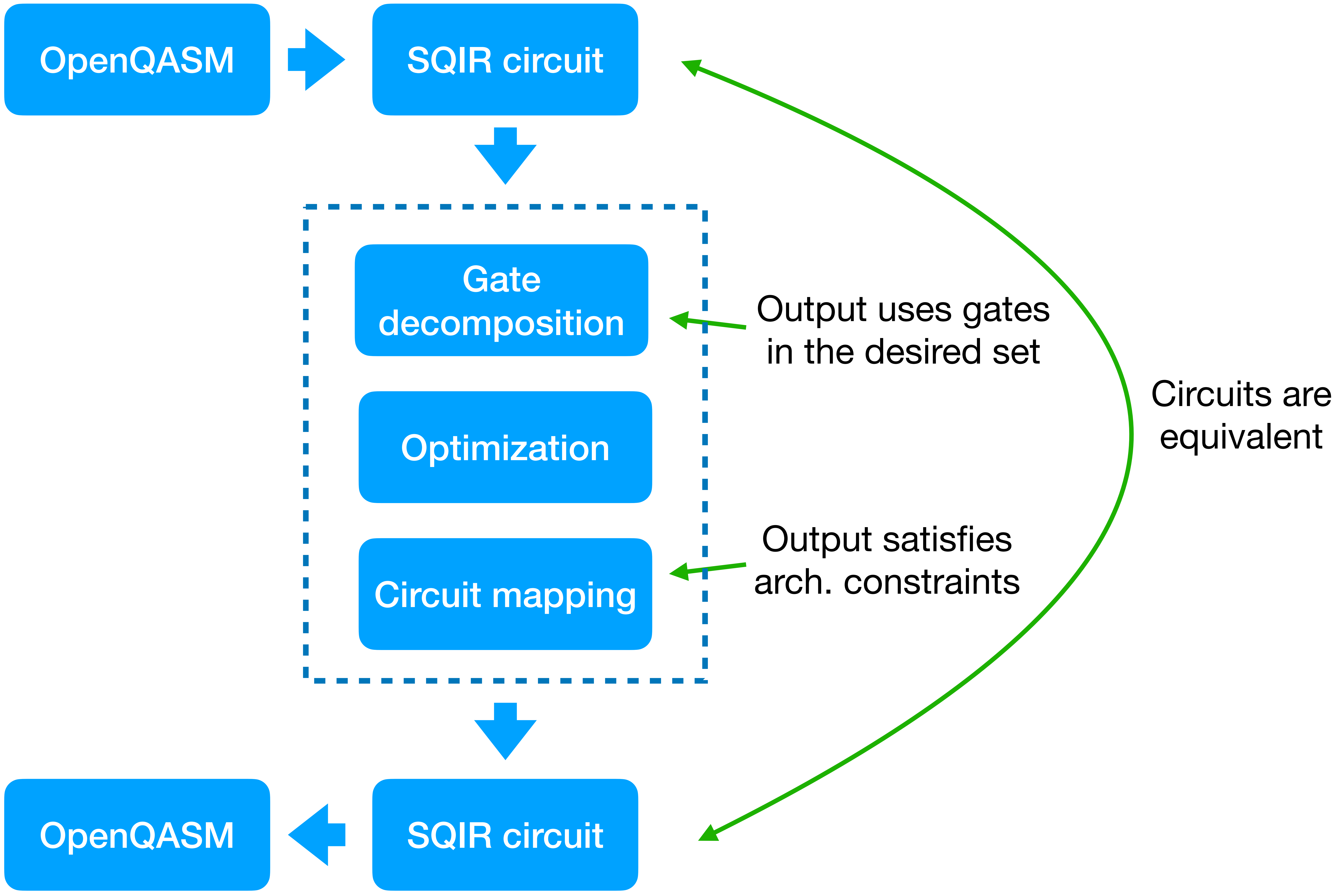
SQIR — a QIR designed for verification

VOQC — a verified compiler

IRs for oracles

Concluding thoughts

VOQC: Verified Optimizer for Quantum Circuits



OpenQASM

SQIR circuit

Gate decomposition

Optimization

Circuit mapping

SQIR circuit

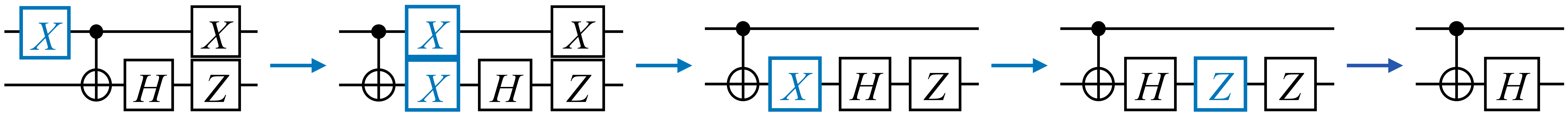
OpenQASM

Output uses gates in the desired set

Output satisfies arch. constraints

Circuits are equivalent

Example: X propagation

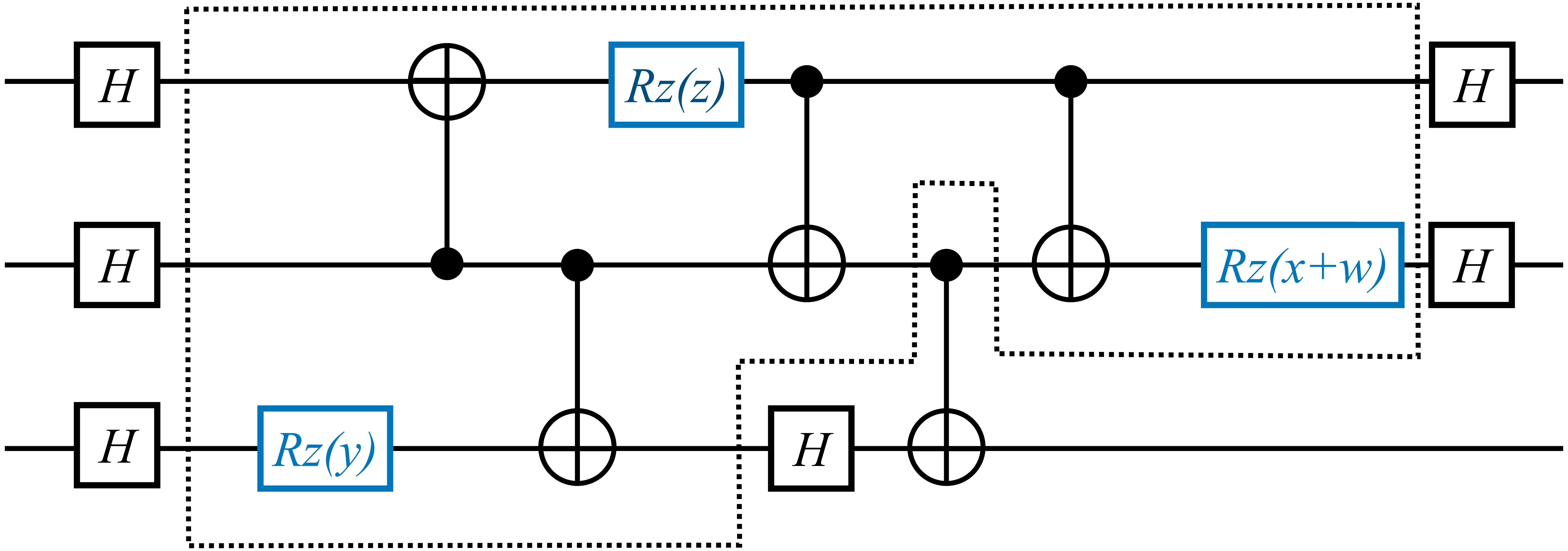


- Based on Nam et al[1] “not propagation”
- We verify **semantics-preservation**
  - At each step, the denotation of the program (i.e. unitary matrix) does not change
- We prove this via induction on the structure of the input program
  - ~30 lines to implement optimization
  - ~270 lines to prove semantics-preservation

---

[1] Nam, Ross, Su, Childs and Maslov. *Automated Optimization of Large Quantum Circuits with Continuous Parameters*. npj 2018.

More interesting: Rotation merging



- Based on Nam et al rotation merging
- Combines *Rz* gates in arbitrary {*Rz*, *CNOT*} sub-circuits
  - ~100 lines to implement optimization
  - ~920 lines to prove semantics-preservation

Evaluation

[1] https://qiskit.org/
[2] https://cqcl.github.io/pytket/build/html/index.html
[3] https://arxiv.org/pdf/1710.07345.pdf
[4] https://arxiv.org/pdf/1303.2042.pdf
[5] https://github.com/Quantomatic/pyzx

- Compared our *verified* optimizer against existing *unverified* optimizers on a benchmark by Amy et al.[4]
  - IBM Qiskit Terra v0.15.12[1]
  - Cambridge CQC tket v0.6.0[2]
  - Nam et al.,[3] both L and H levels (used by IonQ)
  - Amy et al.[4]
  - PyZX v0.6.0[5]

Results

[1] https://qiskit.org/
[2] https://cqcl.github.io/pytket/build/html/index.html
[3] https://arxiv.org/pdf/1710.07345.pdf
[4] https://arxiv.org/pdf/1303.2042.pdf
[5] https://github.com/Quantomatic/pyzx

Geo. mean compilation times

Qiskit[1]	tket[2]	Nam[3] (L)	Nam (H)	Amy[4]	PyZX[5]	VOQC ✓
0.812s	0.129s	0.002s	0.018s	0.007s	0.384s	**0.013s**

VOQC is the same ballpark

Geo. mean reduction in gate count

Qiskit	tket	Nam (H)	VOQC ✓
10.1%	10.6%	24.8%	**17.8%**

VOQC only outperformed by Nam

Geo mean. reduction in T gate count

Amy	PyZX	Nam (H)	VOQC ✓
39.7%	42.6%	41.4%	**41.4%**

VOQC only outperformed by PyZX

This talk

Motivation

SQIR — a QIR designed for verification

VOQC — a verified compiler

IRs for oracles

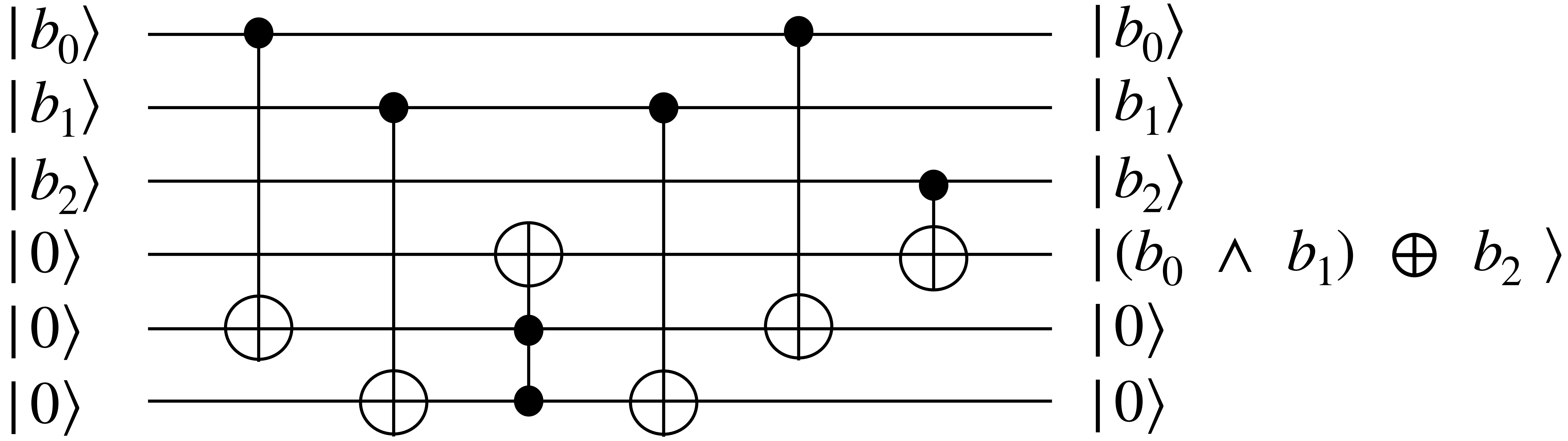
Concluding thoughts

Motivation: Verifying oracles

- Many quantum programs rely on *oracles*, classical functions evaluated on quantum data
  - E.g., Deutsch-Jozsa algorithm, Shor's factoring algorithm

- Rather than verifying the oracle circuit directly, it's easier to verify the oracle in a special-purpose IR first and then used a verified compiler

$$(b_0 \wedge b_1) \oplus b_2$$



RCIR: Reversible Circuit IR

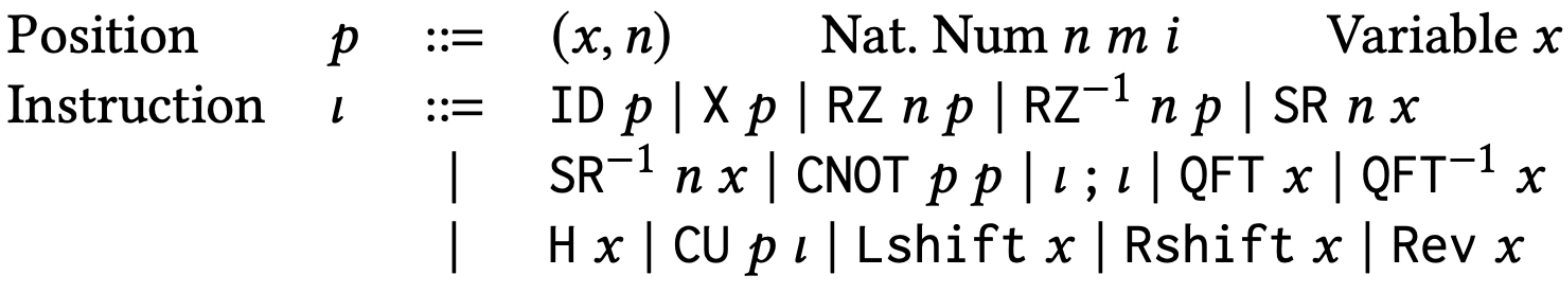
- We developed RCIR, a language for describing Boolean functions with a proved-correct compiler to SQIR

$$R := \texttt{skip} \mid \texttt{X}\, n \mid \texttt{ctrl}\, n\, R \mid \texttt{swap}\, m\, n \mid R_1; R_2$$

- We use RCIR to define the modular multiplication oracle in our full implementation of Shor's algorithm
  - Project lead by Yuxiang Peng (UMD), draft in preparation

PQASM: “phase-space” QASM

- We are also working on a new IR that allows some non-classical operations (e.g., Hadamard transform, QFT) while still being efficiently simulatable



$$
\begin{array}{llll}
\text{Position} & p & ::= & (x, n) \qquad \text{Nat. Num } n\ m\ i \qquad \text{Variable } x \\
\text{Instruction} & \iota & ::= & \mathtt{ID}\ p \mid \mathtt{X}\ p \mid \mathtt{RZ}\ n\ p \mid \mathtt{RZ}^{-1}\ n\ p \mid \mathtt{SR}\ n\ x \\
 & & \mid & \mathtt{SR}^{-1}\ n\ x \mid \mathtt{CNOT}\ p\ p \mid \iota\ ;\ \iota \mid \mathtt{QFT}\ x \mid \mathtt{QFT}^{-1}\ x \\
 & & \mid & \mathtt{H}\ x \mid \mathtt{CU}\ p\ \iota \mid \mathtt{Lshift}\ x \mid \mathtt{Rshift}\ x \mid \mathtt{Rev}\ x
\end{array}
$$

- We prove properties about PQASM programs first, and then use a verified compiler from PQASM to SQIR
  - Project lead by Liyi Li (UMD), draft available upon request

This talk

Motivation

SQIR — a QIR designed for verification

VOQC — a verified compiler

IRs for oracles

Concluding thoughts

Lessons learned

- Formal verification requires a well-defined semantics so it is naturally easier to verify *small, domain-specific (sub-)languages* like SQIR, RCIR, and PQASM
  - Restricts language features & interoperability with other compilers
  - Larger languages may be ok with *comprehensive documentation*

- A matrix-based semantics requires a mapping from program “variables” to matrix/vector indices. This requires forsaking variables (SQIR) or reasoning about the allocation of variables to indices (PQASM)
  - Restricts IR design
  - An indication that matrices are not the right approach?

Moving forward

- In order to scale up to industry-grade IRs like QIR, we may be able to reuse existing verified IR frameworks
  - E.g., the Vellvm project out of UPenn provides a semantics for LLVM
- Alternatively, we might choose to verify properties simpler than full semantic correctness. E.g.,
  - Qubits are used linearly
  - Qubits are unentangled when they are discarded
- During my internship with Microsoft, we wrote a plugin for the Q# compiler to automatically check some of these simpler properties

Get involved

- Our code is available online: github.com/inQWIRE/SQIR
  - Pull requests & issues welcome!
- ITP 2021 paper on verifying SQIR programs: arxiv:2010.01240
- POPL 2021 paper on optimizing SQIR programs with VOQC: arxiv:1912.02250

- Collaborators:
  - Mike Hicks (UMD)
  - Shih-Han Hung (UT Austin)
  - Liyi Li (UMD)
  - Sarah Marshall (Microsoft)
  - Yuxiang Peng (UMD)
  - Robert Rand (U Chicago)
  - Kartik Singhal (U Chicago)
  - Finn Voichick (UMD)
  - Xiaodi Wu (UMD)