Expanding the VOQC Toolkit

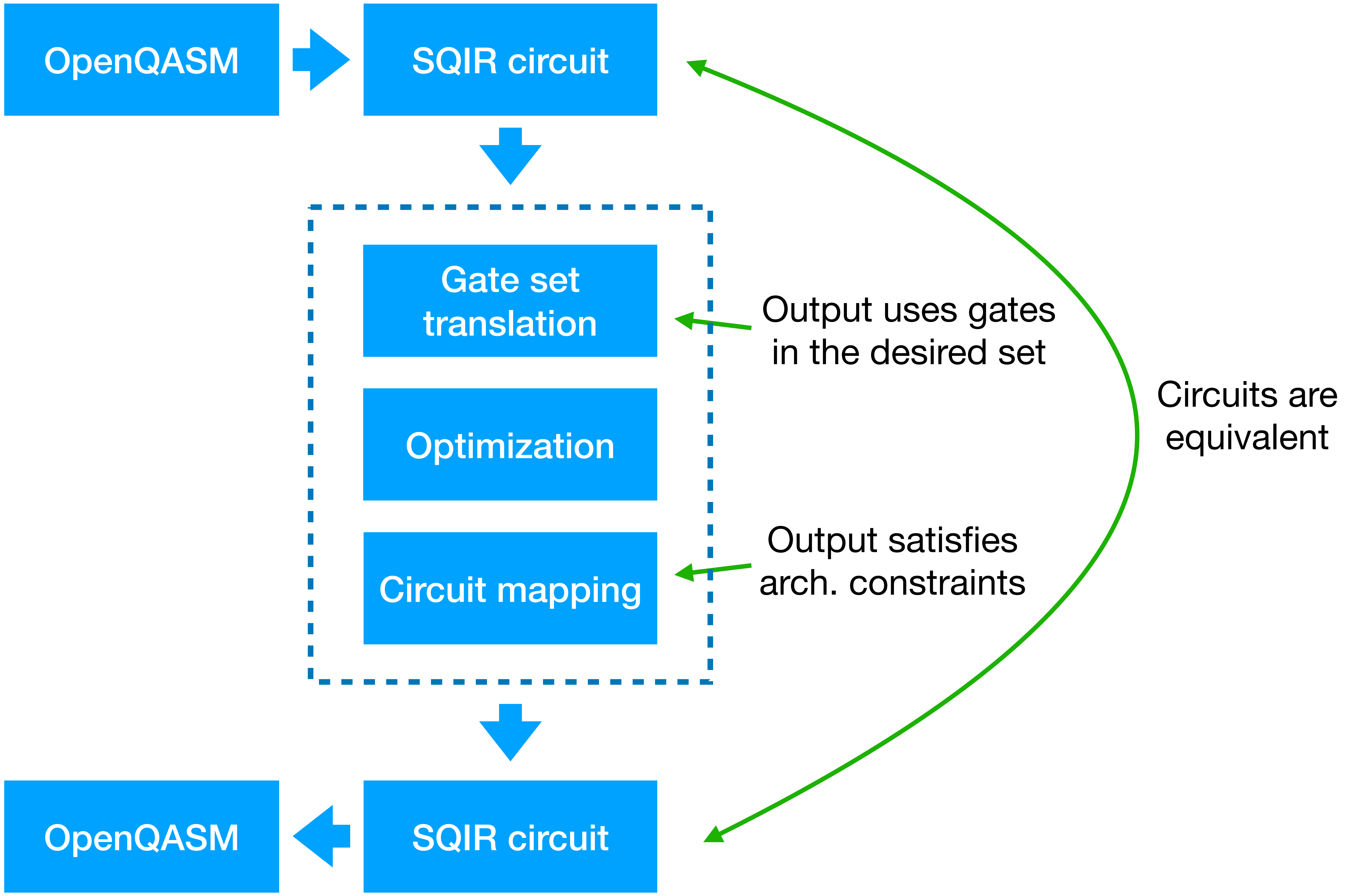
Kesha Hietala[1] Liyi Li[1] Akshaj Gaur[2] Aaron Green[1]
Robert Rand[3] Xiaodi Wu[1] Michael Hicks[1]

[1] University of Maryland [2] Poolesville High School [3] University of Chicago

VOQC: Verified Optimizer for Quantum Circuits

- An optimizer for quantum circuits *formally verified* in the Coq proof assistant
  - Optimizations are proved to be *semantics preserving*, i.e., they do not change the "meaning" of the input circuit
- Circuits expressed in SQIR, a Simple Quantum Intermediate Representation
- VOQC and SQIR were presented at POPL 2021
- Followup paper to appear at ITP 2021 shows how to use SQIR as a source language for verifying quantum algorithms (e.g. Grover's, QPE)

VOQC: Verified Optimizer for Quantum Circuits



In This Talk

- New gate sets and optimizations
- Better support for circuit mapping
- Python bindings

“IBM” Gate Set

- Consists of the gates {U1, U2, U3, CX}

- U1, U2, U3 are parameterized by real rotation angles

$$U_1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}, \quad U_2(\phi, \lambda) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda)} \end{pmatrix}, \quad U_3(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda} \sin(\theta/2) \\ e^{i\phi} \sin(\theta/2) & e^{i(\phi+\lambda)} \cos(\theta/2) \end{pmatrix}$$

  - In Coq, we reason about the axiomatized `Real` type; in the extracted OCaml code we use `floats`

Qiskit's Optimize1qGates

- Finds adjacent single-qubit gates (U1, U2, U3) and combines them
  - E.g. merging U1, U2

$$U_1(\lambda_1); U_1(\lambda_2) \rightarrow U_1(\lambda_1 + \lambda_2)$$

$$U_1(\lambda_1); U_2(\phi, \lambda_2) \rightarrow U_2(\lambda_2, \lambda_1 + \phi)$$

  - More complicated: merging U2, U3

$$\begin{aligned}
U_3(\theta_1, \phi_1, \lambda_1); U_3(\theta_2, \phi_2, \lambda_2) \quad &= R_z(\phi_2) \cdot R_y(\theta_2) \cdot R_z(\lambda_2) \cdot R_z(\phi_1) \cdot Ry(\theta_1) \cdot R_z(\lambda_1) \\
&= R_z(\phi_2) \cdot [R_y(\theta_2) \cdot R_z(\lambda_2 + \phi_1) \cdot R_y(\theta_1)] \cdot R_z(\lambda_1) \\
&= R_z(\phi_2) \cdot [R_z(\gamma) \cdot R_y(\beta) \cdot R_z(\alpha)] \cdot R_z(\lambda_1) \\
&= R_z(\phi_2 + \gamma) \cdot R_y(\beta) \cdot R_z(\alpha + \lambda_1) \\
&= U_3(\beta, \phi_2 + \gamma, \alpha + \lambda_1)
\end{aligned}$$

Qiskit's Optimize1qGates

- Finds adjacent single-qubit gates (U1, U2, U3) and combines them
  - E.g. merging U1, U2

$$U_1(\lambda_1); U_1(\lambda_2) \rightarrow U_1(\lambda_1 + \lambda_2)$$

$$U_1(\lambda_1); U_2(\phi, \lambda_2) \rightarrow U_2(\lambda_2, \lambda_1 + \phi)$$

  - More complicated: merging U2, U3

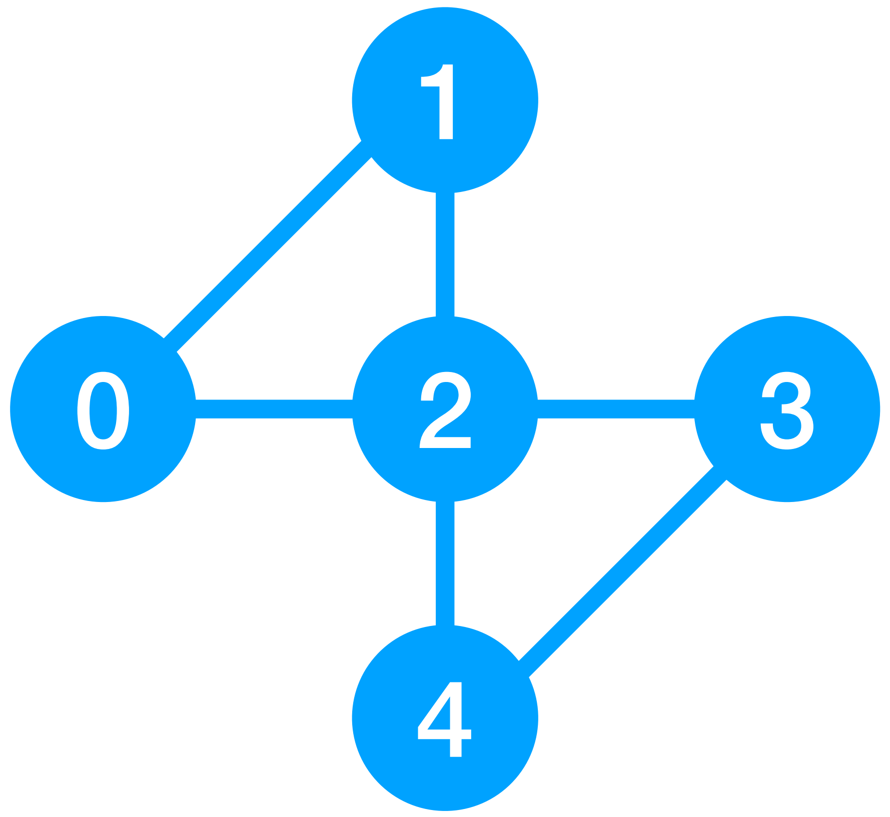
$$\begin{aligned} U_3(\theta_1, \phi_1, \lambda_1); U_3(\theta_2, \phi_2, \lambda_2) \quad &= R_z(\phi_2) \cdot R_y(\theta_2) \cdot R_z(\lambda_2) \cdot R_z(\phi_1) \cdot Ry(\theta_1) \cdot R_z(\lambda_1) \\ &= R_z(\phi_2) \cdot [R_y(\theta_2) \cdot R_z(\lambda_2 + \phi_1) \cdot R_y(\theta_1)] \cdot R_z(\lambda_1) \\ &= R_z(\phi_2) \cdot [R_z(\gamma) \cdot R_y(\beta) \cdot R_z(\alpha)] \cdot R_z(\lambda_1) \\ &= R_z(\phi_2 + \gamma) \cdot R_y(\beta) \cdot R_z(\alpha + \lambda_1) \\ &= U_3(\beta, \phi_2 + \gamma, \alpha + \lambda_1) \end{aligned}$$

The hard part is $yzy \rightarrow zyz$ conversion

Summary of Features

- Gate sets
  - “RzQ” {X, H, Rz, CX}
  - “IBM” {U1, U2, U3, CX}
  - “Standard” {I, X, Y, Z, …, CX, CZ, SWAP, CCX, CCZ}
- Optimizations
  - Five passes from Nam et al. [2018] (evaluated in our POPL paper)
  - Optimize1qGates and CXCancellation from Qiskit
- Simple circuit mapping

Circuit Mapping




```
CNOT 0 2
CNOT 1 3
```

```
CNOT 0 2
SWAP 1 2
CNOT 1 3
```

(with the positions of qubits
1 and 2 swapped)

- We want this transformation to…
  - Be *semantics-preserving* (the two programs should be denoted by the same matrix, up to a permutation of qubits)
  - Produce an output that satisfies the architecture’s constraints

Composing VOQC Transformations

- Coq program to map a circuit to a 10-qubit LNN architecture and then perform optimization (OCaml syntax is similar)

```
Definition optimize_then_map c :=
  let gr := make_lnn 10 in          (* 10-qubit LNN architecture *)
  let la := trivial_layout 10 in    (* trivial layout on 10 qubits *)
  if check_well_typed c 10          (* check that c is well-typed & uses ≤10 qubits *)
  then
    let c' := optimize_nam c in     (* optimization #1 *)
    let c'' := optimize_ibm c' in   (* optimization #2 *)
    simple_map c'' la gr            (* map *)
  else None.
```

Composing VOQC Transformations

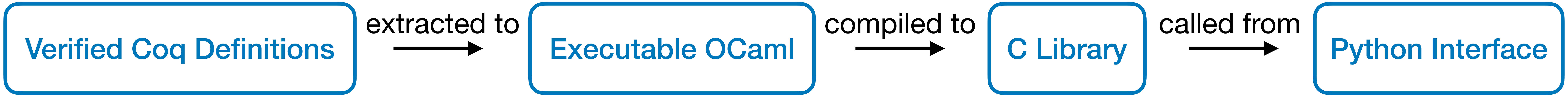
- Coq program to optimize a circuit and then map it to a 10-qubit LNN architecture (OCaml syntax is similar)

```
Definition map_then_optimize c :=
  let gr := make_lnn 10 in                (* 10-qubit LNN architecture *)
  let la := trivial_layout 10 in          (* trivial layout on 10 qubits *)
  if check_well_typed c 10                (* check that c is well-typed & uses ≤10 qubits *)
  then
    match simple_map c la gr with         (* map *)
    | Some (c', la') ⇒
        let c'' := optimize_nam c' in     (* optimization #1 *)
        let c''' := optimize_ibm c'' in   (* optimization #2 *)
        (c''', la')
    | None ⇒ None
  else None.
```

- To support optimization after mapping, we prove that all optimizations are *mapping preserving*

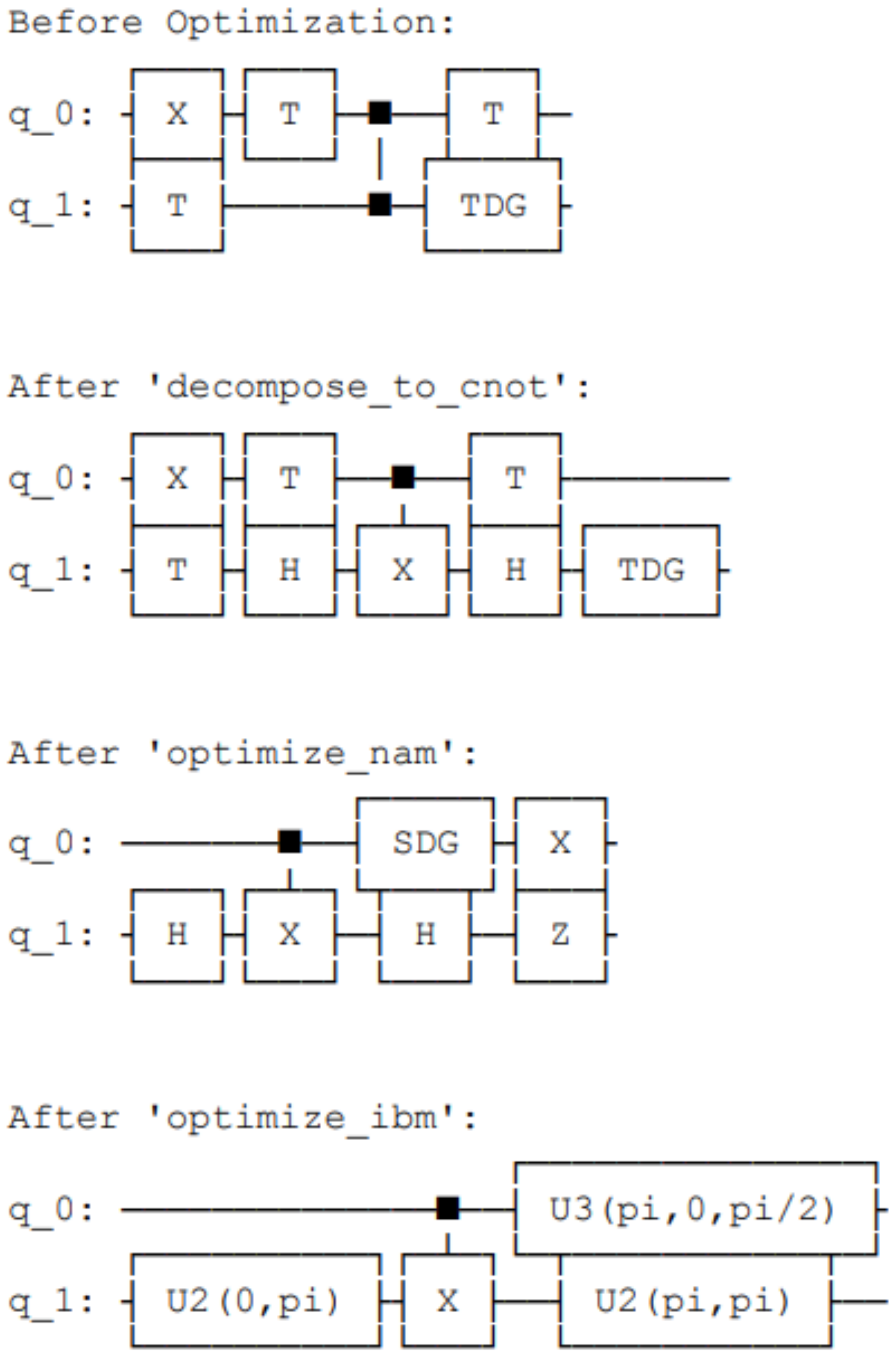
pyvoqc

- We want to make VOQC a drop-in replacement for circuit optimizers used in frameworks like Qiskit, pytket, pyQuil, Cirq (all written in Python)
- So we added Python bindings for VOQC optimizations



pyvoqc

```python
from qiskit import QuantumCircuit
from pyvoqc.qiskit.voqc_pass import QiskitVOQC
from qiskit.transpiler import PassManager

# create a circuit using Qiskit's interface
circ = QuantumCircuit(2)
circ.x(0)
circ.t(0)
circ.t(1)
circ.cz(0, 1)
circ.t(0)
circ.tdg(1)
print("Before Optimization:")
print(circ)

# create a Qiskit PassManager
pm = PassManager()

# decompose CZ gate
pm.append(QiskitVOQC(["decompose_to_cnot"]))
new_circ = pm.run(circ)
print("\n\nAfter 'decompose_to_cnot':")
print(new_circ)

# run optimizations from Nam et al.
pm.append(QiskitVOQC(["optimize_nam", "replace_rzq"]))
new_circ = pm.run(circ)
print("\n\nAfter 'optimize_nam':")
print(new_circ)

# run IBM gate merging
pm.append(QiskitVOQC(["optimize_ibm"]))
new_circ = pm.run(circ)
print("\n\nAfter 'optimize_ibm':")
print(new_circ)
```



Ongoing Work

- More thorough evaluation of VOQC (e.g. using Arline's benchmarks)
- New optimizations, especially approximate
- More sophisticated mapping & mapping-aware optimizations
- *In progress*: Compilation from classical (reversible) programs to SQIR circuits

Resources

- Our Coq definitions and proofs are available at https://github.com/inQWIRE/SQIR.
- Our OCaml library is available at https://github.com/inQWIRE/mlvoqc and can be installed with “opam install voqc”.
    - Documentation on the OCaml library interface is available at https://inqwire.github.io/mlvoqc/voqc/Voqc/index.html.
- Our Python bindings and a tutorial are available at https://github.com/inQWIRE/pyvoqc.
- ***We welcome contributions!*** Feel free to file issues or submit pull requests.