Formal verification of quantum compilers

Hanru Jiang
Peng Cheng Laboratory
2019/09/12

Goal

Contract-based verification of a realistic quantum compiler

Yunong Shi*
The University of Chicago
yunong@uchicago.edu

Xupeng Li*
Columbia University
xupeng.li@columbia.edu

Runzhou Tao
Columbia University
runzhou.tao@columbia.edu

Ali Javadi-Abhari
IBM T.J. Watson Research center
Ali.Javadi@ibm.com

Andrew W. Cross
IBM T.J. Watson Research center
awcross@us.ibm.com

Frederic T. Chong
The University of Chicago
chong@cs.uchicago.edu

Ronghui Gu
Columbia University
ronghui.gu@columbia.edu

MICRO 2019 poster

Goal

Contract-based verification of a realistic quantum compiler

Yunong Shi*
The University of Chicago
yunong@uchicago.edu

Xupeng Li*
Columbia University
xupeng.li@columbia.edu

Runzhou Tao
Columbia University
runzhou.tao@columbia.edu

Ali Javadi-Abhari
IBM T.J. Watson Research center
Ali.Javadi@ibm.com

Andrew W. Cross
IBM T.J. Watson Research center
awcross@us.ibm.com

Frederic T. Chong
The University of Chicago
chong@cs.uchicago.edu

Ronghui Gu
Columbia University
ronghui.gu@columbia.edu

MICRO 2019 poster

- Building **quantum compilers**…

Goal

Contract-based verification of a realistic quantum compiler

Yunong Shi*
The University of Chicago
yunong@uchicago.edu

Xupeng Li*
Columbia University
xupeng.li@columbia.edu

Runzhou Tao
Columbia University
runzhou.tao@columbia.edu

Ali Javadi-Abhari
IBM T.J. Watson Research center
Ali.Javadi@ibm.com

Andrew W. Cross
IBM T.J. Watson Research center
awcross@us.ibm.com

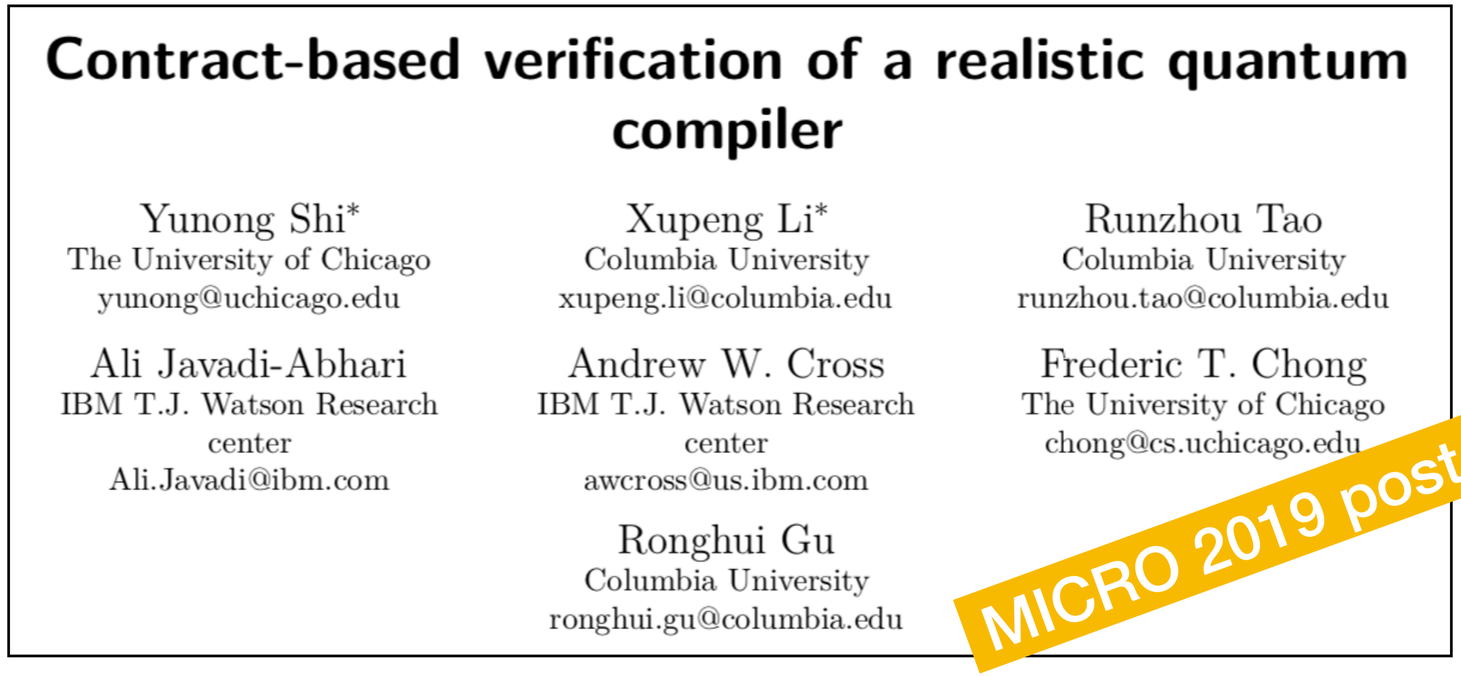
Frederic T. Chong
The University of Chicago
chong@cs.uchicago.edu

Ronghui Gu
Columbia University
ronghui.gu@columbia.edu

MICRO 2019 poster

- Building **quantum compilers**…
- that are guaranteed to be **bug-free**…

Goal



Contract-based verification of a realistic quantum compiler

Yunong Shi*
The University of Chicago
yunong@uchicago.edu

Xupeng Li*
Columbia University
xupeng.li@columbia.edu

Runzhou Tao
Columbia University
runzhou.tao@columbia.edu

Ali Javadi-Abhari
IBM T.J. Watson Research center
Ali.Javadi@ibm.com

Andrew W. Cross
IBM T.J. Watson Research center
awcross@us.ibm.com

Frederic T. Chong
The University of Chicago
chong@cs.uchicago.edu

Ronghui Gu
Columbia University
ronghui.gu@columbia.edu

MICRO 2019 poster

- Building **quantum compilers**…
- that are guaranteed to be **bug-free**…
- with **low verification burden**.

Result

Result

- **CertiQ** — a mostly-automated verification framework…

Result

- **CertiQ** — a mostly-automated verification framework…
- for the Qiskit-Terra quantum transpilers.

Result

- **CertiQ** — a mostly-automated verification framework…
- for the Qiskit-Terra quantum transpilers.

translate between PLs at approximately the same level of abstraction

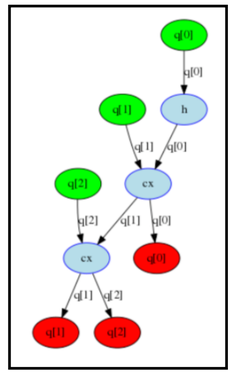
Result

- **CertiQ** — a mostly-automated verification framework…
- for the Qiskit-Terra quantum transpilers.

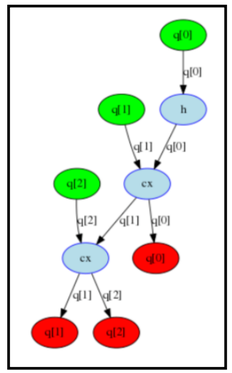
translate between PLs at approximately the same level of abstraction

```
from qiskit import QuantumCircuit, Aer, execute

qc = QuantumCircuit(2, 2)

qc.h(0)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])

backend = Aer.get_backend('qasm_simulator')
job_sim = execute(qc, backend)
sim_result = job_sim.result()

print(sim_result.get_counts(qc))
```

Converter



Transpilers



Assembler

qobj

...

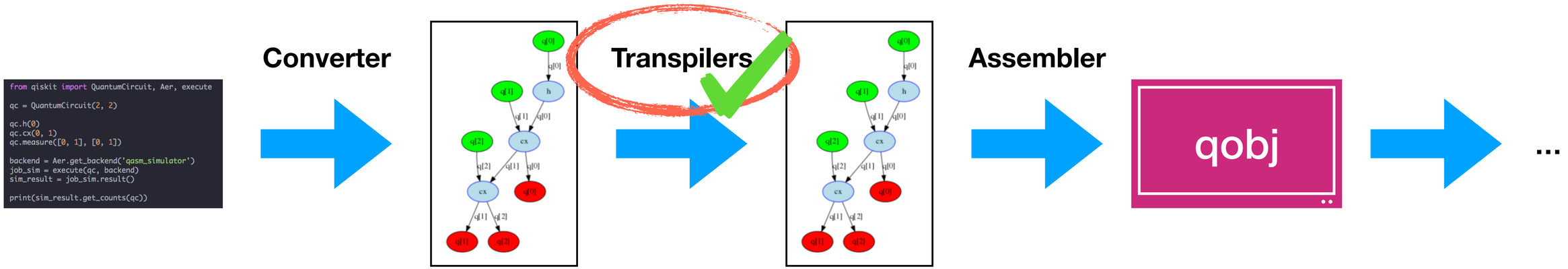
Qiskit Circuit Description | DAG-representation | DAG-representation | Runnable qobj

Execution of Qiskit-Terra circuits: https://github.com/Qiskit/qiskit-terra/blob/master/qiskit/execute.py

Result

- **CertiQ** — a mostly-automated verification framework…
- for the Qiskit-Terra quantum transpilers.

translate between PLs at approximately the same level of abstraction




```
from qiskit import QuantumCircuit, Aer, execute

qc = QuantumCircuit(2, 2)

qc.h(0)
qc.cx(0, 1)
qc.measure([0, 1], [0, 1])

backend = Aer.get_backend('qasm_simulator')
job_sim = execute(qc, backend)
sim_result = job_sim.result()

print(sim_result.get_counts(qc))
```

Execution of Qiskit-Terra circuits: https://github.com/Qiskit/qiskit-terra/blob/master/qiskit/execute.py

CertiQ

CertiQ

- **Input**: transpiler pass + **contract** (specification, e.g. circuit equivalence)

CertiQ

- **Input**: transpiler pass + **contract** (specification, e.g. circuit equivalence)

```
# @swap
class BasicSwap(TranformationPass):
  def __init__(self, layout=None, coupling_map):
     self.layout = layout
    self.coupling_map = coupling_map
```

CertiQ

- **Input**: transpiler pass + **contract** (specification, e.g. circuit equivalence)
- **Output**:
  - **CORRECT!** & Z3 **proof** w.r.t. the contract
  - **BUGGY!** & **counter example**
  - **UNKNOWN!** & **verified validator** if applicable

```
# @swap
class BasicSwap(TranformationPass):
  def __init__(self, layout=None, coupling_map):
     self.layout = layout
    self.coupling_map = coupling_map
```

CertiQ

- **Input**: transpiler pass + **contract** (specification, e.g. circuit equivalence)
- **Output**:
  - **CORRECT!** & Z3 **proof** w.r.t. the contract
  - **BUGGY!** & **counter example**
  - **UNKNOWN!** & **verified validator** if applicable
- **Underlying technique**: **SMT solver** & **validator**

```
# @swap
class BasicSwap(TranformationPass):
  def __init__(self, layout=None, coupling_map):
     self.layout = layout
    self.coupling_map = coupling_map
```

CertiQ

- **Input**: transpiler pass + **contract** (specification, e.g. circuit equivalence)
- **Output**:
  - **CORRECT!** & Z3 **proof** w.r.t. the contract
  - **BUGGY!** & **counter example**
  - **UNKNOWN!** & **verified validator** if applicable
- **Underlying technique**: **SMT solver** & **validator**

```
# @swap
class BasicSwap(TranformationPass):
  def __init__(self, layout=None, coupling_map):
     self.layout = layout
    self.coupling_map = coupling_map
```

We will come back to it later

Challenges

- Checking circuit equivalence is **QMA** (quantum version of NP).
- Code base grows rapidly, need **modular** proof technique.
- Hard to **automate** the proof.

Contributions

For proving transpiler correctness **efficiently**

- Small step operational semantics for **equivalent circuit transformations**.
- Contract-based specification for **modular verification**.
- Combining SMT solver, symbolic execution, etc. for proof automation.
- Verified 7 transpilers, detected 3 bugs.

Contributions

For proving transpiler correctness **efficiently**

- Small step operational semantics for **equivalent circuit transformations**.
- Contract-based specification for **modular verification**.
- Combining SMT solver, symbolic execution, etc. for proof automation.
- Verified 7 transpilers, detected 3 bugs.

What is transpiler correctness?

- Denote circuit equivalence as $\vDash [\![C]\!] = [\![C']\!]$
- Correct(transpiler) iff:
  - $\forall C, C' . transpiler(C) = C' \implies \vDash [\![C]\!] = [\![C']\!]$
- CertiQ contracts are actually more expressive
  - Transpiler must terminate
  - Resulting circuit conform to coupling-map of quits

Equivalent Circuit Transformation

Equivalent Circuit Transformation

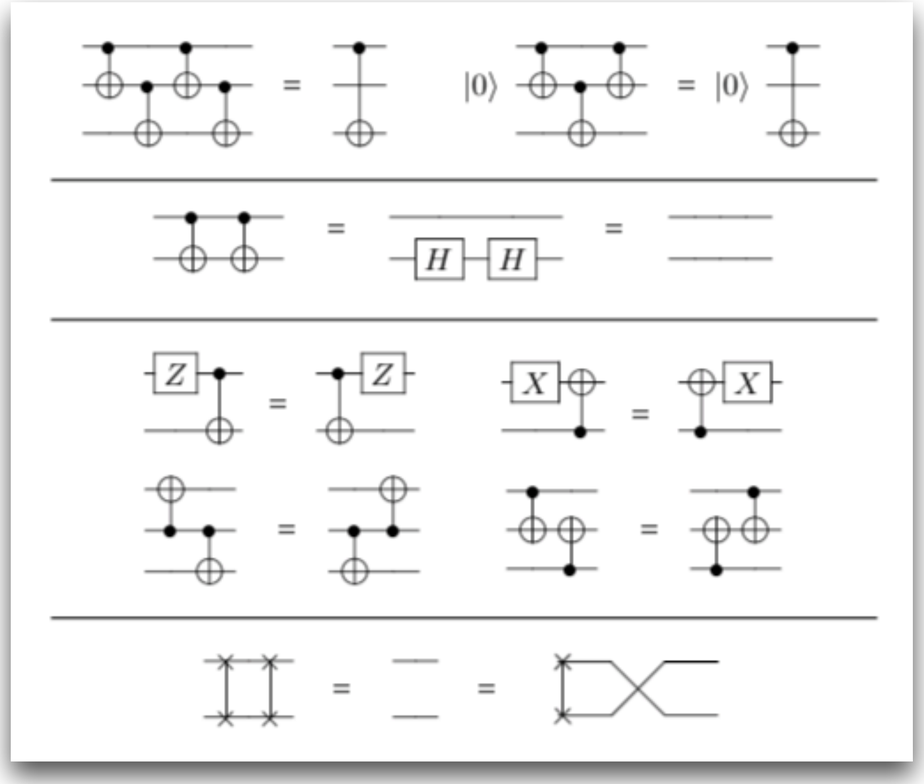
- How to check/prove equivalence **efficiently**?

Equivalent Circuit Transformation

- How to check/prove equivalence **efficiently**?
  - $C \rightarrow_e C'$ if and only if $(C, C') \in$

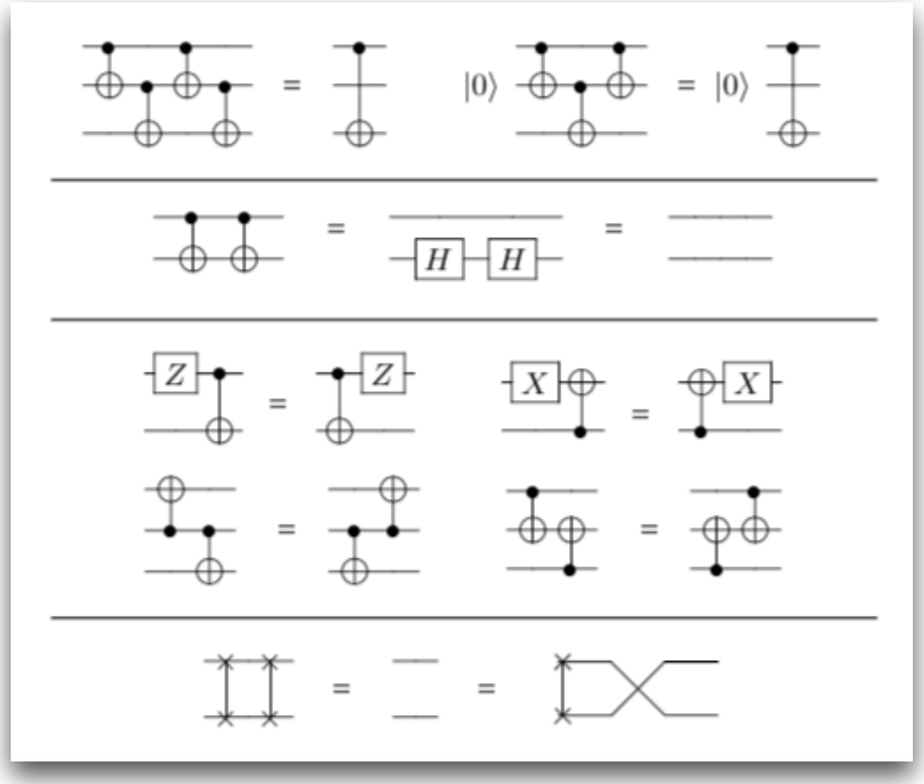
Equivalent Circuit Transformation

- How to check/prove equivalence **efficiently**?
  - $C \rightarrow_e C'$ if and only if $(C, C') \in$



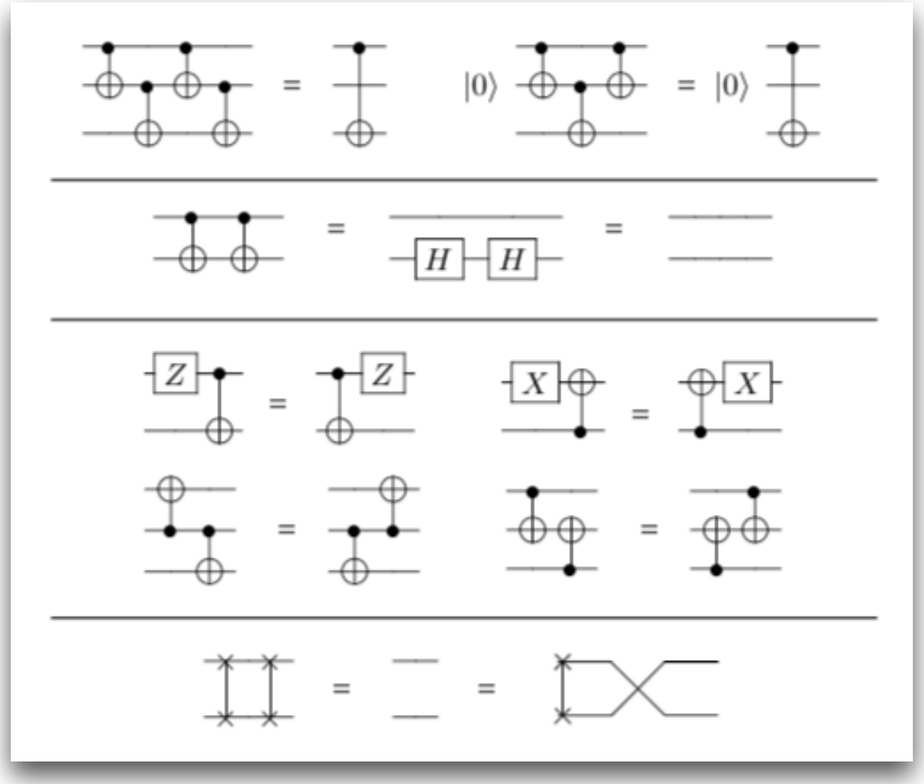
Equivalent Circuit Transformation

- How to check/prove equivalence **efficiently**?
  - $C \to_e C'$ if and only if $(C, C') \in$
  - $$\frac{C \to_e^* C'}{\vdash [\![C]\!] = [\![C']\!]}$$



Equivalent Circuit Transformation

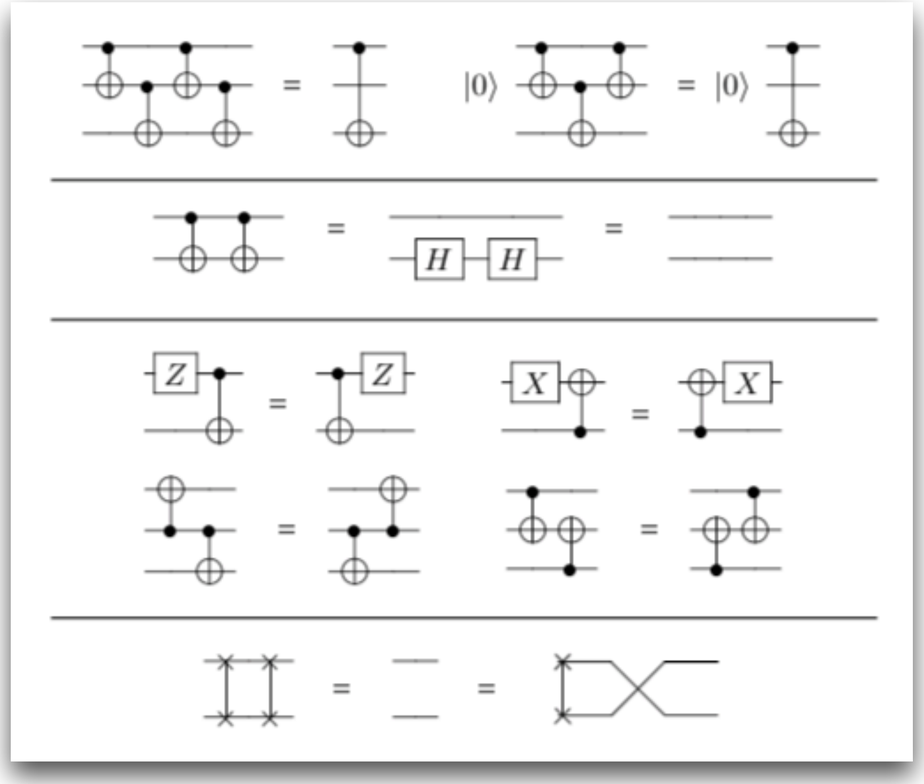
- How to check/prove equivalence **efficiently**?
  - $C \to_e C'$ if and only if $(C, C') \in$
  - $$\frac{C \to_e^* C'}{\vdash [\![C]\!] = [\![C']\!]}$$
  - **Soundness** (not proved):



Equivalent Circuit Transformation

- How to check/prove equivalence **efficiently**?
  - $C \to_e C'$ if and only if $(C, C') \in$
  - $$\frac{C \to_e^* C'}{\vdash [\![C]\!] = [\![C']\!]}$$
  - **Soundness** (not proved):

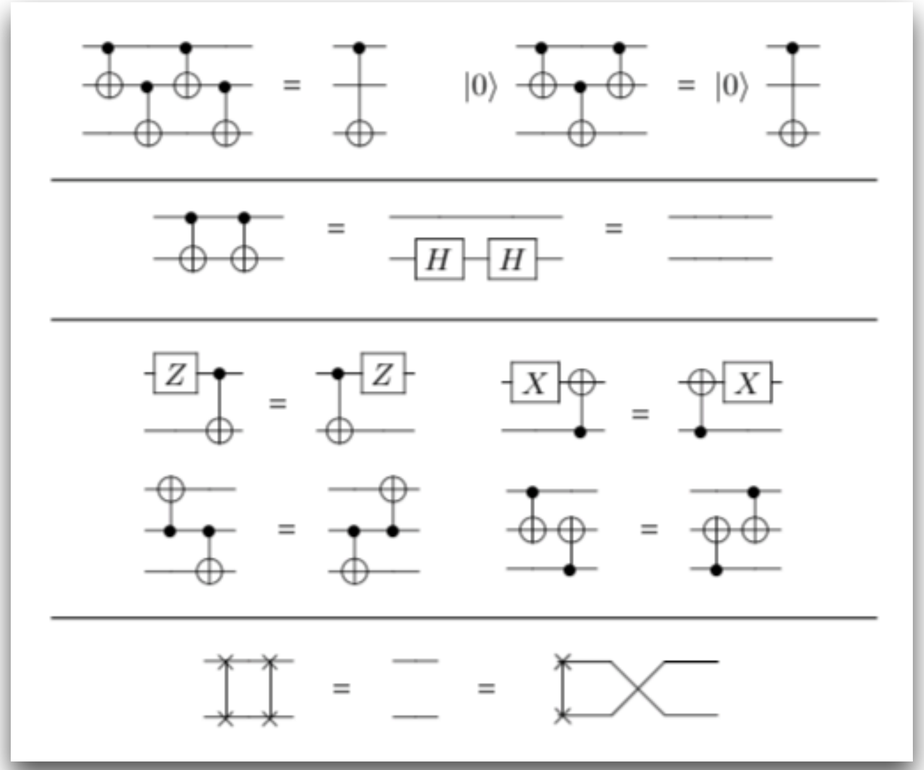
$$\vdash [\![C]\!] = [\![C']\!] \implies \vDash [\![C]\!] = [\![C']\!]$$



Equivalent Circuit Transformation

- How to check/prove equivalence **efficiently**?
  - $C \to_e C'$ if and only if $(C, C') \in$
  - $$\frac{C \to_e^* C'}{\vdash [\![C]\!] = [\![C']\!]}$$
  - **Soundness** (not proved):

$$\vdash [\![C]\!] = [\![C']\!] \implies \vDash [\![C]\!] = [\![C']\!]$$

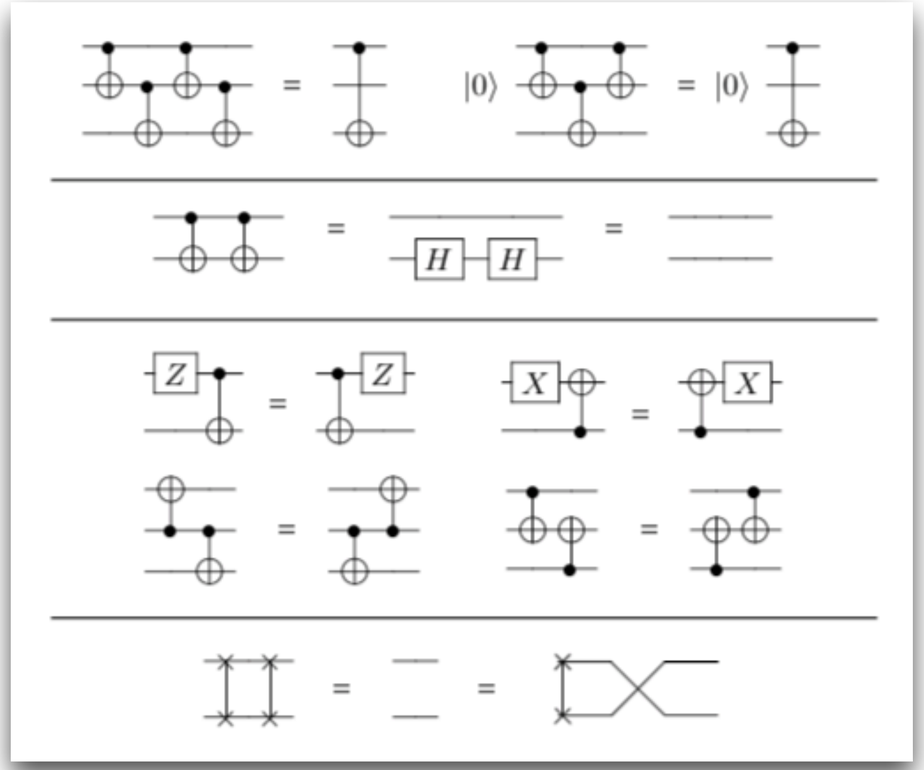


- It suffice to prove $\forall C, C' \,.\, transpiler(C) = C' \implies C \to_e^* C'$

Equivalent Circuit Transformation

- How to check/prove equivalence **efficiently**?
  - $C \to_e C'$ if and only if $(C, C') \in$
  - $$\frac{C \to_e^* C'}{\vdash [\![C]\!] = [\![C']\!]}$$
  - **Soundness** (not proved):

$$\vdash [\![C]\!] = [\![C']\!] \implies \vDash [\![C]\!] = [\![C']\!]$$

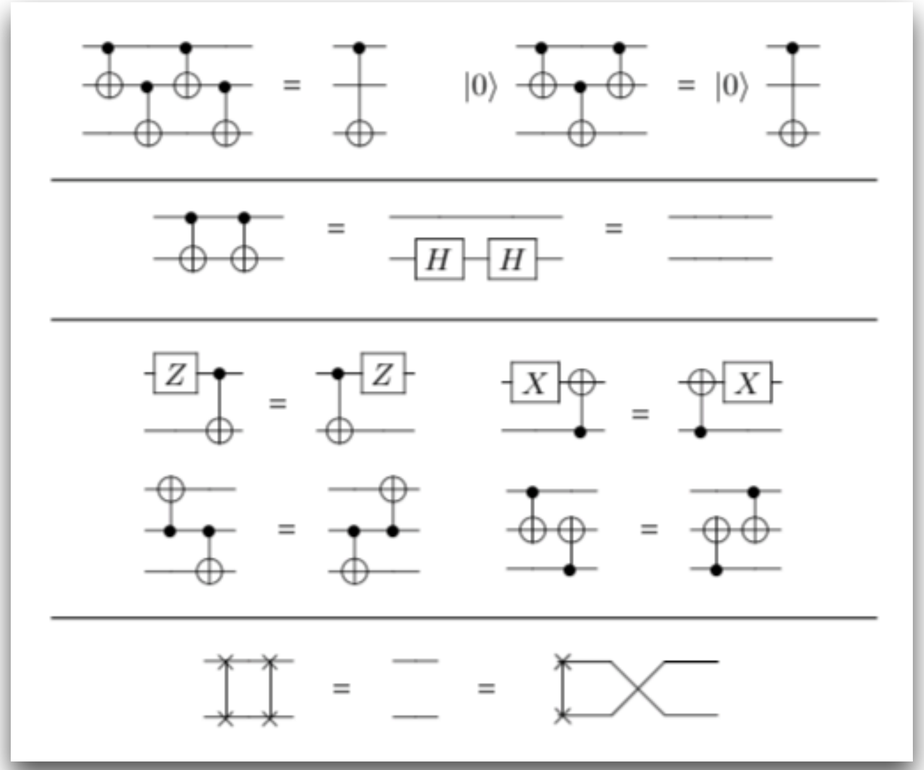


- It suffice to prove $\forall C, C' . transpiler(C) = C' \implies C \to_e^* C'$
- **No matrices/semantics of program involved.**

Equivalent Circuit Transformation

- How to check/prove equivalence **efficiently**?
  - $C \to_e C'$ if and only if $(C, C') \in$
  - $$\frac{C \to_e^* C'}{\vdash [\![C]\!] = [\![C']\!]}$$
  - **Soundness** (not proved):

$$\vdash [\![C]\!] = [\![C']\!] \implies \vDash [\![C]\!] = [\![C']\!]$$



- It suffice to prove $\forall C, C' \,.\, transpiler(C) = C' \implies C \to_e^* C'$
- **No matrices/semantics of program involved.**

Verified Passes (7 of 20 Terra passes)

Verified Passes (7 of 20 Terra passes)

- **Analysis** — do not change DAG
    - collect_2q_block, commutative_analysis

Verified Passes (7 of 20 Terra passes)

- **Analysis** — do not change DAG **Trivial?**
    - collect_2q_block, commutative_analysis

Verified Passes (7 of 20 Terra passes)

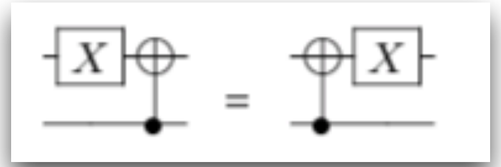
- **Analysis** — do not change DAG **Trivial?**
    - collect_2q_block, commutative_analysis
- **Exchanging/cancelling gates**
    - commutative_cancellation, optimize_1q_gate

Verified Passes (7 of 20 Terra passes)

- **Analysis** — do not change DAG **Trivial?**
    - collect_2q_block, commutative_analysis
- **Exchanging/cancelling gates**
    - commutative_cancellation, optimize_1q_gate

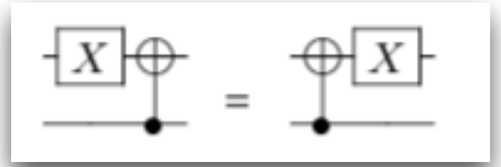
Verified Passes (7 of 20 Terra passes)

- **Analysis** — do not change DAG **Trivial?**
    - collect_2q_block, commutative_analysis
- **Exchanging/cancelling gates**
    - commutative_cancellation, optimize_1q_gate



Verified Passes (7 of 20 Terra passes)

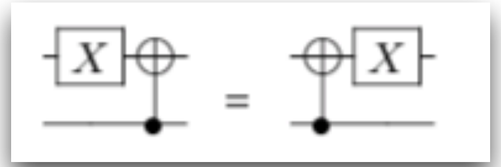
- **Analysis** — do not change DAG **Trivial?**
    - collect_2q_block, commutative_analysis
- **Exchanging/cancelling gates**
    - commutative_cancellation, optimize_1q_gate

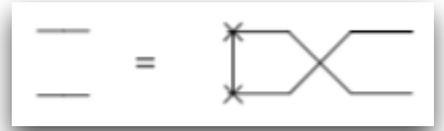


- **Swapping**
    - lookahead_swap, basic_swap, noise_adaptive_swap

Verified Passes (7 of 20 Terra passes)

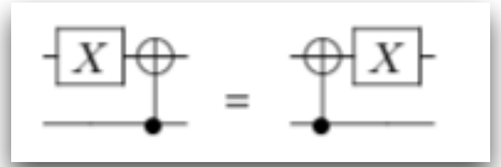
- **Analysis** — do not change DAG **Trivial?**
    - collect_2q_block, commutative_analysis
- **Exchanging/cancelling gates**
    - commutative_cancellation, optimize_1q_gate
- **Swapping**
    - lookahead_swap, basic_swap, noise_adaptive_swap

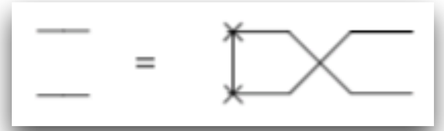




Verified Passes (7 of 20 Terra passes)

- **Analysis** — do not change DAG **Trivial?**
    - collect_2q_block, commutative_analysis
- **Exchanging/cancelling gates**
    - commutative_cancellation, optimize_1q_gate
- **Swapping**
    - lookahead_swap, basic_swap, noise_adaptive_swap

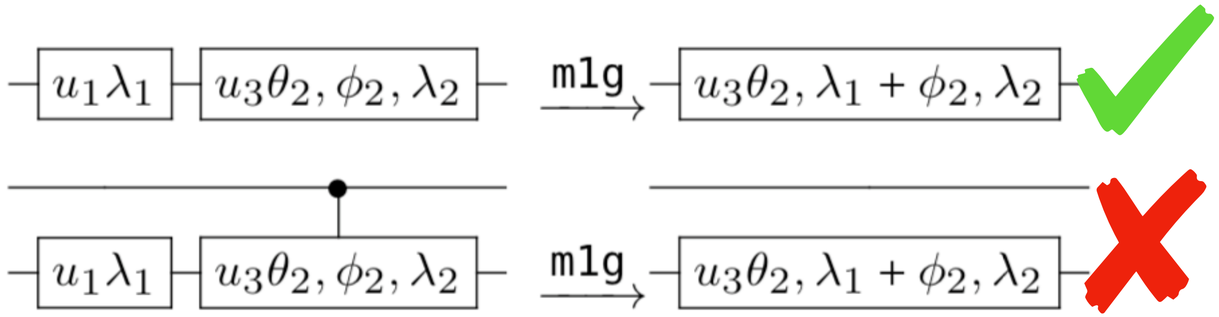




Have to re-implement ALL the passes to enable automated verification

Bugs Detected

- **`lookahead_swap`**: may not terminate
- **`optimize_1q_gate`**



- **`commutation`**
  - $U_1U_2 = U_2U_1 \wedge U_2U_3 = U_3U_2 \implies U_1U_3 = U_3U_1$
  - Does not hold in general, hold for gate set $\{\mathrm{CNOT}, \mathrm{X}, \mathrm{Z}, \mathrm{H}, \mathrm{T}, u_1, u_2, u_3\}$

CertiQ

- **Input**: transpiler pass + **contract** (specification, e.g. circuit equivalence)
- **Output**:
  - **CORRECT!** & Z3 **proof** w.r.t. the contract
  - **BUGGY!** & **counter example**
  - **UNKNOWN!** & **verified validator** if applicable
- **Underlying technique**: **SMT solver** & **validator**

```
class Layout():
  ... # omitted code
  # Contract of the Layout object
  def precondition(self): return True
  def invariant(self): return True
  def postcondition(self):
    i = fresh_int()
    return ForAll([i], self.p2v(self.v2p(i))==i)
```

We will come back to it later

SMT solver

- **SMT** — Satisfiability Modulo Theories
    - decision problem for logical formulas…
    - w.r.t. background theories (e.g. integer, array, linear arithmetic, …)
    - expressed in first-order logic with equality.
- **SMT-solver**:
    - **Input**: (quantifier-free) formula, e.g. $(a = b) \wedge \neg(b - a = 0)$
    - **Output**: sat + model / unsat / out of time
- **Z3**: SMT-solver, available in C/C++, .NET, OCaml, Python, Java, Haskell

SMT solver applications

- Symbolic-execution based analysis and testing:
    - E.g., KLEE (llvm), SAGE (x86-binary),…
- Computer-aided verification
    - Encode pre-conditions/post-conditions/invariants into SMT formulas
    - E.g., VCC (verifier for concurrent C), Why3 (deductive program verification platform),..
- Program synthesis
    - Input: property P
    - Output: program satisfying P

Validator

- Type of validator:

$$\text{QuantumCircuit} \to \text{QuantumCircuit} \to \mathbb{B}$$

- Soundness:

$$\text{validator}(C_{in}, C_{out}) = \text{True} \implies \vdash [\![C_{in}]\!] = [\![C_{out}]\!]$$

$$\text{i.e. } C_{out} \to_e^* C_{in}$$

- Run after compilation.

- Support only swap and CNOT cancellation passes.

Strength

- **Relevant**
    - Verified 7 real transpilers, detected 3 bugs
- **Light-weighted** verification by introducing…
    - Equivalent circuit transformation
    - Z3, static analysis, contract continuation,…
- **Expressive** contract
    - circuit equivalence, conform with coupling-map, termination

Weakness

- Large **trusted computing base**:
    - Qiskit libs, Z3, equivalent circuits, static analyzer…
- Only 7/20? passes verified
- Passes need **re-implement** to enable automated verification
- Validators are **limited** to swap and CNOT cancellation passes
- **Not very clear** written, no contract example for transpiler
- The code is **not publicly available**

THE END