Formal methods of quantum program analysis

Matthew Amy

School of Computing Science, Simon Fraser University

APS March Meeting
Chicago, March 16th, 2022

What is this talk about?

Formal methods of quantum program analysis

↓ ↓ ↓

Formal methods	quantum	program analysis
precise, mathematical methods of reasoning about hardware & software	quantum	determine the behaviour of a program, typically for the purpose of optimization or verification

What is this talk about?

Formal methods of quantum program analysis

↓ ↓ ↓

Formal methods	quantum	program analysis
precise, mathematical methods of reasoning about hardware & software	quantum	determine the behaviour of a program, typically for the purpose of optimization or verification

This talk: the design of analysis-based quantum program optimizations

Why optimize quantum programs?

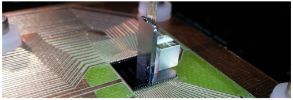
Expectation

QUANTUM COMPUTING KILLS ENCRYPTION

by: Elliot Williams

78 Comments

September 29, 2015



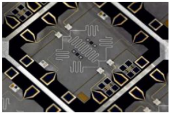
Imagine a world where the most widely-used cryptographic methods turn out to be broken: quantum computers allow encrypted Internet data transactions to become readable by anyone who happened to be listening. No more HTTPS, no more PGP. It sounds a little bit sci-fi, but that's exactly the scenario that cryptographers interested in post-quantum crypto are working to save us from. And although the (potential) threat of quantum computing to cryptography is already well-known, this summer has seen a flurry of activity in the field, so we felt it was time for a recap.

HOW BAD IS IT?

Reality

Physicists demonstrate that 15=3x5 about half of the time

19 August 2012



The device in the photomicrograph was used to run the first solid-state demonstration of Shor's algorithm. It is made up of four phase qubits and five superconducting resonators, for a total of nine engineered quantum elements. The quantum processor measures one-quarter inch square. Credit: UCSB

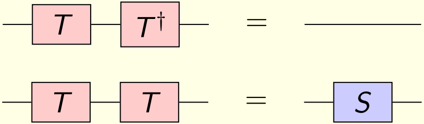
never been done before," said Erik Lucero, the paper's lead author. Now a postdoctoral researcher in experimental quantum computing at IBM, Lucero was a doctoral student in physics at UCSB when the research was conducted and the paper was written.

"What is important is that the concepts used in factoring this small number remain the same when factoring much larger numbers," said Andrew Cleland, a professor of physics at UCSB and a collaborator on the experiment. "We just need to scale up the size of this processor to something much larger. This won't be easy, but the path forward is clear."

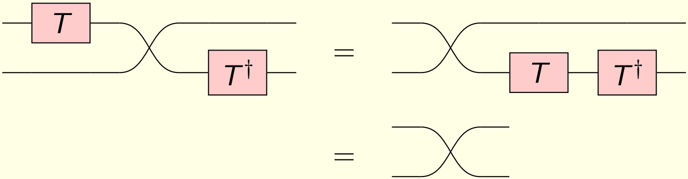
Practical applications motivated the research, according to Lucero, who explained that factoring very large numbers is at the heart of cybersecurity protocols, such as the most common form of

The bread-and-butter of quantum program optimization

Everyone's first circuit optimization: merge adjacent gates



Next level: commute gates first



Gate cancellation

Then

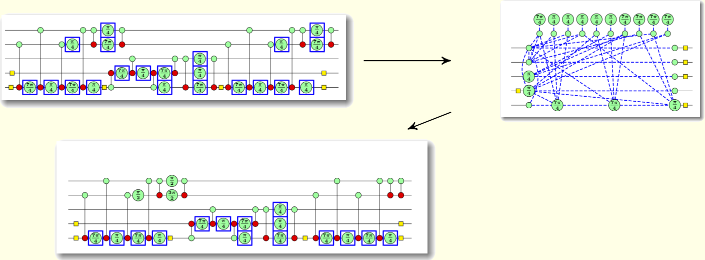
Programs:

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[5];

ccx q[0],q[1],q[4];
ccx q[2],q[4],q[3];
ccx q[0],q[1],q[4];
```

Optimizations: semantics-based



Kissinger & van de Wetering, Reducing T-count with the ZX-calculus. Phys. Rev. A (2020).

Gate cancellation

Now

Programs:

```
qs.circ @ENT(%qb_0, %r_0, %n) {
  %qb_1 = qs.H %qb_0
  %qb_2, %r_1 = affine.for %i = 0 to %n ↩
        iter_args(%qb_i_0 = %qb_0, %r_i_0 = %r_0) {
    %qt_0, %rem = qs.extract %r_i_0[%i]
    %qb_i_1, %qt_1 = qs.CX %qb_i_0, %qt_0
    %r_i_1 = qs.combine %rem[%i], %qt_1
    affine.yield %qb_i_1, %r_i_1
  }
  qs.return %qb_2, %r_1
}
```

loop-carried values

next iteration values

final value of each quantum argument

Optimizations: back to rewrite-based

```
%a1, %b1 = qs.CX %a0, %b0
%a2, %b2 = qs.CX %a1, %b1
%a3 = qs.H %a2
```

⟶

```


%a3 = qs.H %a0
```

Ittah et al., *Enabling Dataflow Optimization for Quantum Programs. ACM Trans. Quant. Comput. (2022).*

Programs:

```
qs.circ @ENT(%qb_0, %r_0, %n) {
  %qb_1 = qs.H %qb_0
  %qb_2, %r_1 = affine.for %i = 0 to %n ↩
        iter_args(%qb_i_0 = %qb_0, %r_i_0 = %r_0) {
    %qt_0, %rem = qs.extract %r_i_0[%i]
    %qb_i_1, %qt_1 = qs.CX %qb_i_0, %qt_0
    %r_i_1 = qs.combine %rem[%i], %qt_1
    affine.yield %qb_i_1, %r_i_1
  }
  qs.return %qb_2, %r_1
}
```

loop-carried values

next iteration values

final value of each quantum argument

Optimizations: back to rewrite-based

```
%a1, %b1 = qs.CX %a0, %b0
%a2, %b2 = qs.CX %a1, %b1
%a3 = qs.H %a2
```

⟶

```


%a3 = qs.H %a0
```

Need formal methods for hybrid programs!

Ittah et al., Enabling Dataflow Optimization for Quantum Programs. ACM Trans. Quant. Comput. (2022).

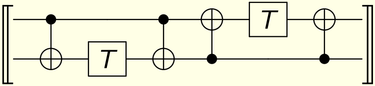
Program analysis has entered the chat

Static program analysis

Goal is to prove properties about the program instead of computing the full semantics

Semantics:

- A description of what a program does in some formal mathematical language
  - E.g., the matrix (semantics) of a circuit (program)



$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & i & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- A language can have many different semantics which expose different aspects of its behaviour
  - E.g. control-flow paths of a parallel program
- Usually, too hard (or impossible) to compute precise semantics

Abstraction

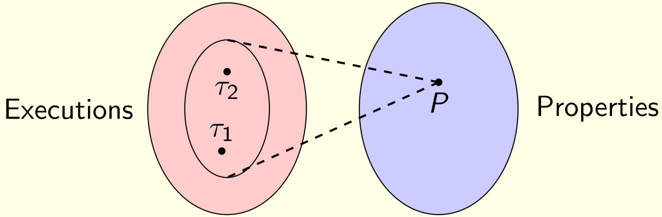
Static analysis uses sound abstraction and approximation to make the problem of proving properties tractable

- Abstraction: only retain relevant properties

  E.g. value of x is negative

- Approximation: occurs when we can't prove the desired property

  (-) + (+) = ???



Trick is to compute a property which is precise but sound

Example: Constant propagation

Want to know when variables are constant

Analysis-based approach (reaching definitions analysis):

- Abstraction: which definitions may reach a location
- Only one definition reaches $\implies$ variable is constant

```
x := 1;
y := 2;
if (x <= y) {
            x := 0;
} else {
            x := 3;
}

z := x*foo(y);
```

Example: Constant propagation

Want to know when variables are constant

Analysis-based approach (reaching definitions analysis):

- Abstraction: which definitions may reach a location
- Only one definition reaches $\implies$ variable is constant

```
x := 1;
y := 2;
if (x <= y) {              // x := 1, y := 2 reach
        x := 0;
} else {
        x := 3;
}
z := x*foo(y);             // x := 0, x := 3 reach
```

Example: Constant propagation

Want to know when variables are constant

Analysis-based approach (reaching definitions analysis):

- Abstraction: which definitions may reach a location
- Only one definition reaches $\implies$ variable is constant

```
x := 1;
y := 2;
if (1 <= 2) {
          x := 0;
} else {
          x := 3;
}

z := x*foo(y);          // x := 0 reaches
```

Example: Constant propagation

Want to know when variables are constant

Analysis-based approach (reaching definitions analysis):

- Abstraction: which definitions may reach a location
- Only one definition reaches $\Longrightarrow$ variable is constant

x := 1;
y := 2;
~~if (1 <= 2) {~~
        x := 0;
~~} else {~~
~~   x := 3;~~
~~}~~

z := 0~~*foo(y)~~;

Applying analysis to *quantum* program optimization

The phase folding optimization

Merge phase gates by proving their arguments are equal

- Analogous to classical dataflow optimizations
  - must-analysis
  - flow- and context-sensitive, path-insensitive
- Applies to hybrid quantum-classical programs
- Poly-time, strictly gate-count decreasing
  - And fast in practice!
  - 100's of qubits & $1,000,000$'s of gates in seconds
- Provably sound

M. Amy, Formal Methods in Quantum Circuit Design. PhD thesis (2019).

Phase-folding in simple circuits

Definition (Sum-over-paths)

Any circuit over Clifford+R_Z can be represented as

$$|\mathbf{x}\rangle \mapsto \frac{1}{\sqrt{2}^k} \sum_{\mathbf{y}\in\mathbb{Z}_2^k} e^{iP(\mathbf{x},\mathbf{y})}|f(\mathbf{x},\mathbf{y})\rangle$$

where f is affine and $P : \mathbb{Z}_2^{n+k} \to \mathbb{R}/2\pi$ is a phase polynomial

$$P(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{z}\in\mathbb{Z}^n} a_{\mathbf{z}}\chi_{\mathbf{z}}(\mathbf{x},\mathbf{y}), \qquad \chi_{\mathbf{z}}(\mathbf{x}) = x_1z_1 \oplus \cdots \oplus x_nz_n$$

- R_Z gates contribute to exactly one term of P
- Can be implemented with one R_Z gate per term
- Optimize by replacing gates contributing to the same term with a single gate

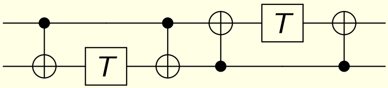
M. Amy, D. Maslov, M. Mosca, Polynomial-time T-depth Optimization via Matroid Partitioning, TCAD (2014).

Example

As sums-over-paths,

$$\begin{aligned} S :& |x\rangle \mapsto i^x|x\rangle \\ T :& |x\rangle \mapsto \omega^x|x\rangle, \qquad\qquad \omega = e^{\frac{\pi i}{4}} \\ \text{CNOT} :& |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle \end{aligned}$$

We can step through the circuit to track the effect of phase gates

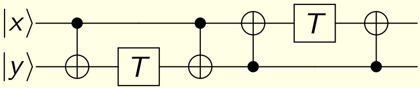


Example

As sums-over-paths,

$$
\begin{aligned}
S :& |x\rangle \mapsto i^x |x\rangle \\
T :& |x\rangle \mapsto \omega^x |x\rangle, \qquad\qquad \omega = e^{\frac{\pi i}{4}} \\
\text{CNOT} :& |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle
\end{aligned}
$$

We can step through the circuit to track the effect of phase gates



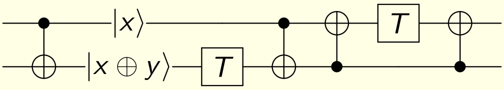
State: $|x\rangle|y\rangle$

Example

As sums-over-paths,

$$
\begin{aligned}
S :& |x\rangle \mapsto i^x |x\rangle \\
T :& |x\rangle \mapsto \omega^x |x\rangle, \qquad\qquad \omega = e^{\frac{\pi i}{4}} \\
\text{CNOT} :& |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle
\end{aligned}
$$

We can step through the circuit to track the effect of phase gates



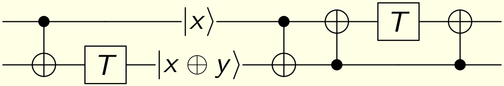
State: $|x\rangle|x \oplus y\rangle$

Example

As sums-over-paths,

$$S : |x\rangle \mapsto i^x |x\rangle$$
$$T : |x\rangle \mapsto \omega^x |x\rangle, \qquad \omega = e^{\frac{\pi i}{4}}$$
$$\text{CNOT} : |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle$$

We can step through the circuit to track the effect of phase gates



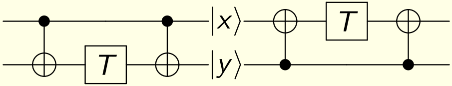
State: $\omega^{x \oplus y}|x\rangle|x \oplus y\rangle$

Example

As sums-over-paths,

$$\begin{aligned} S :& |x\rangle \mapsto i^x |x\rangle \\ T :& |x\rangle \mapsto \omega^x |x\rangle, \qquad\qquad \omega = e^{\frac{\pi i}{4}} \\ \text{CNOT} :& |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle \end{aligned}$$

We can step through the circuit to track the effect of phase gates



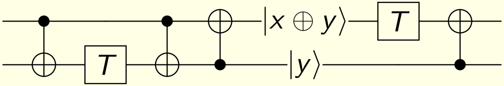
State: $\omega^{x \oplus y}|x\rangle|y\rangle$

Example

As sums-over-paths,

$$S : |x\rangle \mapsto i^x |x\rangle$$
$$T : |x\rangle \mapsto \omega^x |x\rangle, \qquad \omega = e^{\frac{\pi i}{4}}$$
$$\text{CNOT} : |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle$$

We can step through the circuit to track the effect of phase gates



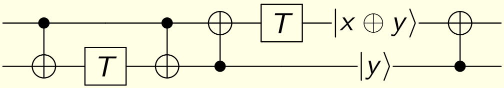
State: $\omega^{x \oplus y} |x \oplus y\rangle |y\rangle$

Example

As sums-over-paths,

$$S : |x\rangle \mapsto i^x |x\rangle$$
$$T : |x\rangle \mapsto \omega^x |x\rangle, \qquad \omega = e^{\frac{\pi i}{4}}$$
$$\text{CNOT} : |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle$$

We can step through the circuit to track the effect of phase gates



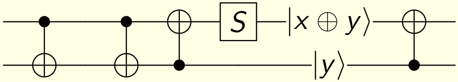
State: $\omega^{x\oplus y}\omega^{x\oplus y}|x \oplus y\rangle|y\rangle = i^{x\oplus y}|x \oplus y\rangle|y\rangle$

Example

As sums-over-paths,

$$S : |x\rangle \mapsto i^x |x\rangle$$
$$T : |x\rangle \mapsto \omega^x |x\rangle, \qquad \omega = e^{\frac{\pi i}{4}}$$
$$\text{CNOT} : |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle$$

We can step through the circuit to track the effect of phase gates



State: $\omega^{x \oplus y} \omega^{x \oplus y} |x \oplus y\rangle|y\rangle = i^{x \oplus y} |x \oplus y\rangle|y\rangle$

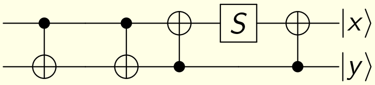
Both T gates can be replaced with a single S gate

Example

As sums-over-paths,

$$S : |x\rangle \mapsto i^x |x\rangle$$
$$T : |x\rangle \mapsto \omega^x |x\rangle, \qquad \omega = e^{\frac{\pi i}{4}}$$
$$\text{CNOT} : |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle$$

We can step through the circuit to track the effect of phase gates



State: $i^{x \oplus y} |x\rangle |y\rangle$

What about hybrid programs?

Consider a simple imperative hybrid quantum-classical language

$$
\begin{aligned}
S ::= &\ U\ q \\
\mid &\ c \leftarrow meas\ q \\
\mid &\ S_1;\ S_2 \\
\mid &\ if\ E\ then\ S_1\ else\ S_2 \\
\mid &\ while\ E\ do\ S
\end{aligned}
$$

Problem: can no longer explicitly represent as a sum-over-paths!

What about hybrid programs?

Consider a simple imperative hybrid quantum-classical language

$$\begin{aligned} S ::= {} & U\ q \\ \mid {} & c \leftarrow meas\ q \\ \mid {} & S_1;\ S_2 \\ \mid {} & if\ E\ then\ S_1\ else\ S_2 \\ \mid {} & while\ E\ do\ S \end{aligned}$$

Problem: can no longer explicitly represent as a sum-over-paths!

As a program analysis:

Rather than compute a single sum-over-paths, prove two gates can be merged in every execution

Designing an analysis

To prove two gates can be merged, need to know when their arguments are the same across every control flow path

Key idea: track the relations between program locations

- E.g. the equation $CNOT|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle$ can be written as a relation between the input and output values

$$CNOT|x\rangle|y\rangle = |x'\rangle|y'\rangle \text{ where } x' = x, y' = x \oplus y$$

- Explicitly, keep a set of relations which must hold

$$\{x' = x, y' = x \oplus y\}$$

Designing an analysis

To prove two gates can be merged, need to know when their arguments are the same across every control flow path

Key idea: track the relations between program locations

- E.g. the equation $CNOT|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle$ can be written as a relation between the input and output values

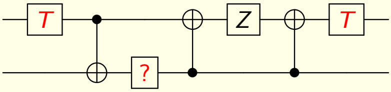
$$CNOT|x\rangle|y\rangle = |x'\rangle|y'\rangle \text{ where } x' = x, y' = x \oplus y$$

- Explicitly, keep a set of relations which must hold

$$\{x' = x, y' = x \oplus y\}$$

Free optimizations: it's always safe to assume no relations hold

- E.g. $U|x_1 \cdots x_n\rangle = |x_1' \cdots x_n'\rangle$



Designing an analysis

To prove two gates can be merged, need to know when their arguments are the same across every control flow path

Key idea: track the relations between program locations

- E.g. the equation $CNOT|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle$ can be written as a relation between the input and output values

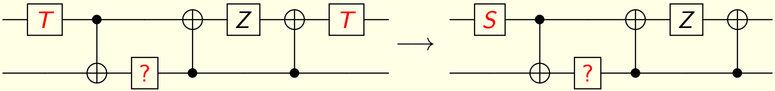
$$CNOT|x\rangle|y\rangle = |x'\rangle|y'\rangle \text{ where } x' = x, y' = x \oplus y$$

- Explicitly, keep a set of relations which must hold

$$\{x' = x, y' = x \oplus y\}$$

Free optimizations: it's always safe to assume no relations hold

- E.g. $U|x_1 \cdots x_n\rangle = |x_1' \cdots x_n'\rangle$



Meet-over-paths

For branching classical control, a relation holds if and only if it holds on all control-flow paths

- $\implies$ take intersection of relations across branches

Example

Consider the program

$$c \leftarrow meas\ q_1;\ if\ (c)\ then\ \{X\ q_2;\ Tq_3\}\ else\ \{Z\ q_2;\ T^\dagger q_3\}$$

The circuits along each control flow path are

$$\{|0\rangle\langle 0| \otimes X \otimes T, |1\rangle\langle 1| \otimes Z \otimes T^\dagger\}$$

For the first circuit the relations $\{x' = 0, y' = 1 \oplus y, z' = z\}$ *hold, while for the second we have* $\{x' = 1, y' = y, z' = z\}$*. The only relation that holds in both paths is* $\{z' = z\}$*. Hence the overall effect is*

$$|xyz\rangle \mapsto |x'y'z'\rangle \text{ such that } z' = z$$

Putting it all together

Phase analysis for the quantum WHILE language, semi-formally

$$
\begin{aligned}
[\![R_Z^\ell(\theta)]\!]_a &= |x\rangle \mapsto e^{\theta_\ell x}|x\rangle \\
[\![X]\!]_a &= |x\rangle \mapsto |1 \oplus x\rangle \\
[\![\mathrm{CNOT}]\!]_a &= |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle \\
[\![meas]\!]_a &= |x\rangle \mapsto |x'\rangle \\
[\![U]\!]_a &= |x_1 x_2 \ldots x_n\rangle \mapsto |x_1' x_2' \ldots x_n'\rangle \\
[\![c(U)]\!]_a &= |x_1\rangle|x_2 \ldots x_n\rangle \mapsto |x_1\rangle|x_2' \ldots x_n'\rangle
\end{aligned}
$$

$$
\begin{aligned}
[\![S_1;\ S_2]\!]_a &= [\![S_2]\!]_a \circ [\![S_1]\!]_a \\
[\![if\ E\ then\ S_1\ else\ S_2\ ]\!]_a &= [\![S_1]\!]_a \cap [\![S_2]\!]_a \\
[\![while\ E\ do\ S\ ]\!]_a &= \cap_{i=0}^{\infty} [\![S_1]\!]^i
\end{aligned}
$$

The optimization

Run phase analysis and normalize P up to the computed relations. Then, for any term $\sum_{\ell \in S} \theta_\ell$ of P

1. Select some $\ell_0 \in S$
2. Set $\theta_{\ell_0} \leftarrow \sum_{\ell \in S} \theta_\ell$
3. Set $\theta_\ell \leftarrow 0$ for all $\ell \in S \setminus \{\ell_0\}$

Theorem (Soundness)

If P contains a term $\sum_{\ell \in S} \theta_\ell$, then the gates at locations $\ell \in S$ can be replaced with a single $R_Z(\sum_{\ell \in S} \theta_\ell)$ gate

Performance

Need hybrid benchmarks to actually test the interesting parts!

Conclusion

Take-aways

- Need new methods of optimizing hybrid quantum programs
- Static program analysis is a powerful tool for tackling this problem
  - Get a lot of things for free
  - Takes a lot of the guesswork away
  - Existing tools & frameworks for proving correctness
- Think about proving properties rather than rewriting!

Thank you!