Symbolic analysis of quantum programs

Matthew Amy

Department of Mathematics & Statistics, Dalhousie University

IBM, November 1st, 2021
Compiler optimizations

Peephole optimizations

- **Re-writing** some small segment of code
- Classical: re-write rules on assembly code
- Quantum: templates, peephole re-synthesis
Compiler optimizations

Peephole optimizations
▶ **Re-writing** some small segment of code
▶ Classical: re-write rules on assembly code
▶ Quantum: templates, peephole re-synthesis

Code generation
▶ **Compiling** efficient code
▶ Classical: Register allocation, instruction scheduling, etc.
▶ Quantum: Oracle synthesis, gate synthesis, routing, etc.
Compiler optimizations

Peephole optimizations
▶ **Re-writing** some small segment of code
▶ Classical: re-write rules on assembly code
▶ Quantum: templates, peephole re-synthesis

Code generation
▶ **Compiling** efficient code
▶ Classical: Register allocation, instruction scheduling, etc.
▶ Quantum: Oracle synthesis, gate synthesis, routing, etc.

Analysis-based optimizations
▶ **Proving** some facts about code
▶ Classical: Constant propagation, common subexpression elimination, dead code elimination, etc.
▶ Quantum: ???
Compiler optimizations

Peephole optimizations
- **Re-writing** some small segment of code
- Classical: re-write rules on assembly code
- Quantum: templates, peephole re-synthesis

Code generation
- **Compiling** efficient code
- Classical: Register allocation, instruction scheduling, etc.
- Quantum: Oracle synthesis, gate synthesis, routing, etc.

Analysis-based optimizations
- **Proving** some facts about code
- Classical: Constant propagation, common subexpression elimination, dead code elimination, etc.
- Quantum: ???

In this talk:

*Tools for writing analysis-based optimizations*
Analysis-based optimizations

- Uses (some form of) abstract interpretation
  - e.g., data-flow analysis, symbolic execution, etc.
- Basic recipe: \((\text{semantics} + \text{facts}) \times \text{soundness relation}\)
  - e.g., In every execution, the read of variable \(x\) at location \(\ell\) may read the definitions of \(x\) at locations in \(\mathcal{M}\)
- Often uses set-based collecting semantics with an abstraction function and/or abstract transformers
Example
Constant propagation

At each location in the program, we want to know which definitions to variables can reach that point

```plaintext
1  x = 1;
2  y = 2;
3  if (x <= y) {
4      x = 0;
5  } else {
6      x = 3;
7  }
8
9  if (x > 0) {
10     ...
11  }
```
At each location in the program, we want to know which definitions to variables can **reach** that point

```
1  x = 1;
2  y = 2;
3  if (x <= y) {
      // x = 1, y = 2 reach
4      x = 0;
5  } else {
6      x = 3;
7  }
8
9  if (x > 0) {
   // x = 0 or x = 3 reach
10     /* error */
11  }
```
Example
Constant propagation

At each location in the program, we want to know which definitions to variables can **reach** that point

1 \( x = 1; \)
2 \( y = 2; \)
3 \( x = 0; \)
4
5 \textbf{if} (x > 0) \{ \quad \text{// } x = 0 \text{ reaches } \\
6 \quad \ldots \\
7 \}
Semantics of quantum computing
The linear-algebraic view

A state of $n$ qubits is a unit vector in $\mathbb{C}^{2^n}$

$$|\psi\rangle = \sum_{x \in \mathbb{Z}_2^n} \alpha_x |x\rangle, \quad x \in \{0, 1\}^n = \mathbb{Z}_2^n$$

Computations change the state by applying unitary matrices to states

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad H = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$S = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \omega = e^{i\frac{\pi}{4}} \end{bmatrix}$$

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$
A quantum process is a collection of (classical) paths.

The path integral view

A → (α + β + γ)B + δC
Formal path integrals

As an object, we can represent a path integral by:

- a collection $\Pi$ of paths $\pi : x \rightarrow x'$ between basis states, and
- an amplitude function $\Phi : \Pi \rightarrow \mathbb{C}$

The **action** is the mapping

$$|x\rangle \mapsto \sum_{\pi : x \rightarrow x' \in \Pi_x} \Phi(\pi)|x'\rangle$$
Composition of computations or circuits is path composition

\[
(\Phi' \circ \Phi)(\pi' \circ \pi) = \Phi(\pi') \Phi'(\pi)
\]
Composition of computations or circuits is path composition

\[
\Pi' \circ \Pi = \{ \pi \pi' : x \rightarrow x' \mid \pi : x \rightarrow x'' \in \Pi \land \pi' : x'' \rightarrow x' \in \Pi' \}
\]

\[
(\Phi' \circ \Phi)(\pi' \circ \pi) = \Phi(\pi)\Phi'(\pi')
\]
Recovering the linear algebraic view

We can encode a unitary $U : |i\rangle \mapsto \sum_{j \in \mathbb{Z}_2^n} U_{ij} |j\rangle$ as a path integral:

$$\Pi_U = \{\pi_{ij} | i, j \in \mathbb{Z}_2^n\}$$

$$\Phi_U(\pi_{ij}) = U_{ij}$$
Recovering the linear algebraic view

We can encode a unitary $U : |i\rangle \mapsto \sum_{j \in \mathbb{Z}^n} U_{ij} |j\rangle$ as a path integral:

$$\Pi_U = \{\pi_{ij} | i, j \in \mathbb{Z}_2^n\}$$

$$\Phi_U(\pi_{ij}) = U_{ij}$$

We can also recover a matrix by summing over all paths for each beginning and end point:

$$U_{ij} = \sum_{\pi : i \rightarrow j \in \Pi} \phi(\pi)$$
Recovering the linear algebraic view

We can encode a unitary $U : |i\rangle \mapsto \sum_{j \in \mathbb{Z}_2} U_{ij} |j\rangle$ as a path integral:

$$\Pi_U = \{ \pi_{ij} \mid i, j \in \mathbb{Z}_2^n \}$$

$$\Phi_U(\pi_{ij}) = U_{ij}$$

We can also recover a matrix by summing over all paths for each beginning and end point:

$$U_{ij} = \sum_{\pi : i \to j \in \Pi} \phi(\pi)$$

Can be viewed as **delayed matrix multiplication**

$$(VU)_{ij} = \sum_{\pi : i \to j \in \Pi_U, \pi' : j \to k \in \Pi_V} \phi_U(\pi) \phi_V(\pi')$$

$\Rightarrow \text{BQP} \subseteq \text{PSPACE}, \text{BQP} \subseteq \text{PP}$
Symbolic path integrals

For $d$-dimensional systems we can

- label each path $\pi$ by a length $k \geq n$ bit string $x \in \mathbb{Z}_d^k$
- write the end point as a function $f : \mathbb{Z}_d^k \rightarrow \mathbb{Z}_d^n$
- write the amplitude $\phi : \mathbb{Z}_d^k \rightarrow \mathbb{C}$ as a function of this bit string
Phase gates, e.g.

\[ S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \omega \end{bmatrix}, \quad R_Z(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \]

apply a phase conditional on certain paths
**Example**

Phase gates

Phase gates, e.g.

\[
S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & \omega \end{bmatrix}, \quad R_{Z}(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}
\]

apply a phase conditional on certain paths

We can write this symbolically as

\[
T : |x\rangle \mapsto \omega^{x} |x\rangle \quad \text{for any } x \in \mathbb{Z}_{2}
\]
Classical gates like

\[ X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \]

permute the states
**Example**

Classical gates

Classical gates like

\[
X = \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}
\]

permute the states

Symbolically,

\[
X : |x\rangle \leftrightarrow |1 \oplus x\rangle \quad \text{for any } x \in \mathbb{Z}_2
\]
Example
Branching gates

The hadamard gate $H$ **branches** on a classical value in superposition with equal weight $\frac{1}{\sqrt{2}}$ and varying phase.

\[ H: |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{Z}_2} (-1)^{xy} |y\rangle \]

$y$ represents the path taken.
The hadamard gate $H$ **branches** on a classical value in superposition with equal weight $\frac{1}{\sqrt{2}}$ and varying phase.

Symbolically,

$$H : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{Z}_2} (-1)^{xy} |y\rangle$$

for $x \in \mathbb{Z}_2$

$y$ represents the path taken
Interference

Linear algebraically, $HH = I$, but symbolically,

$$HH|\chi\rangle = \frac{1}{2} \sum_{y,z \in \mathbb{Z}_2} (-1)^{xy+yz} |z\rangle$$

for any $x \in \mathbb{Z}_2$

In particular, the **internal** paths indexed by $y$ **interfere**
Interference

Linear algebraically, $HH = I$, but symbolically,

$$HH\left| x \right\rangle = \frac{1}{2} \sum_{y, z \in \mathbb{Z}_2} (-1)^{xy + yz}\left| z \right\rangle$$

for any $x \in \mathbb{Z}_2$

In particular, the **internal** paths indexed by $y$ **interfere**

$$\frac{1}{2} + \frac{1}{2} = 1$$

Amplitudes add
Interference

Linear algebraically, $HH = I$, but symbolically,

$$HH|x⟩ = \frac{1}{2} \sum_{y,z\in\mathbb{Z}_2} (-1)^{xy+yz}|z⟩$$ for any $x \in \mathbb{Z}_2$

In particular, the **internal** paths indexed by $y$ **interfere**

$$\frac{1}{2} - \frac{1}{2} = 0$$

Amplitudes cancel
Interference

Linear algebraically, $HH = I$, but symbolically,

$$HH|x\rangle = \frac{1}{2}\sum_{y,z\in\mathbb{Z}_2} (-1)^{xy+yz}|z\rangle$$

for any $x \in \mathbb{Z}_2$

In particular, the **internal** paths indexed by $y$ **interfere**

\[ \frac{1}{2} - \frac{1}{2} = 0 \]

Amplitudes cancel
Linear algebraically, $HH = I$, but symbolically,

$$HH|x\rangle = \frac{1}{2} \sum_{y,z \in \mathbb{Z}_2} (-1)^{xy+yz} |z\rangle$$

for any $x \in \mathbb{Z}_2$

In particular, the internal paths indexed by $y$ interfere

\[\frac{1}{2} + \frac{1}{2} = 1\]

Amplitudes add
Symbolic path integrals

\[ R_Z(\theta) : |x\rangle \mapsto e^{i\theta x} |x\rangle, \quad \text{CNOT} : |x\rangle|y\rangle \mapsto |x\rangle|x \oplus y\rangle, \]
\[ H : |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{Z}_2} (-1)^{xy} \]

**Theorem**

*Any circuit*\(^*\) over Clifford\(\!+\!R_Z\) can be represented symbolically as

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

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

\[ P(x, y) = \sum_{z \in \mathbb{Z}^n} a_z \chi_z(x, y), \quad \chi_z(x) = x_1 z_1 \oplus \cdots \oplus x_n z_n \]

Moreover, this representation is poly-time and -space computable.
Optimization
A standard way to remove phase gates is by merging adjacent ones

\[
\begin{array}{c}
T \quad T^\dagger \\
\end{array}
\]

= 

\[
\begin{array}{c}
\text{merge gates}
\end{array}
\]
Merging gates

A standard way to remove phase gates is by merging adjacent ones.

\[ T \quad T^\dagger \quad = \quad \quad \]

\[ T \quad T \quad = \quad S \]
Merging gates

A standard way to remove phase gates is by merging adjacent ones

\[ T \quad T^\dagger \quad = \quad \]
\[ T \quad T \quad = \quad S \]

In some cases we have to commute gates to merge them

\[ T \quad T^\dagger \quad = \quad \]
\[ T \quad T^\dagger \quad = \quad \]
What about more complicated cases?

\[ T \bullet \quad \quad = \quad \quad T \bullet \]
\[ T \quad \quad = \quad \quad T \quad S^\dagger \]
What about more complicated cases?

\[ T \begin{array}{c} \bullet \\ \circ \end{array} = \begin{array}{c} \bullet \\ \circ \end{array} T \]

\[ T \begin{array}{c} \bullet \\ \circ \end{array} = \begin{array}{c} \bullet \\ \circ \end{array} T \]

\[ T \begin{array}{c} \bullet \\ \circ \end{array} = \begin{array}{c} \bullet \\ \circ \end{array} T S^\dagger \]

\[ \begin{array}{c} \bullet \\ \circ \end{array} T \begin{array}{c} \bullet \\ \circ \end{array} \begin{array}{c} \bullet \\ \circ \end{array} = ??? \]
Paths & phases

As a collection of paths:

$$
\begin{vmatrix}
|00\rangle & |01\rangle & |10\rangle & |11\rangle \\
\end{vmatrix}

= |x\rangle|y\rangle \mapsto i^{x\oplus y} |x\rangle|y\rangle$$
Paths & phases

\[
\begin{bmatrix}
T \\
 & & & \\
 & & & \\
 & & & \\
T \\
\end{bmatrix} = |x\rangle|y\rangle \mapsto i^{x \oplus y} |x\rangle|y\rangle
\]

As a collection of paths:

|00\rangle -- |00\rangle

|01\rangle -- \omega -- |01\rangle

|10\rangle -- \omega -- \omega -- |10\rangle

|11\rangle -- \omega -- \omega -- |11\rangle
Paths & phases

As a collection of paths:
Quantum Phase folding

We could re-synthesize $^1 |x\rangle|y\rangle \mapsto i^{x\oplus y}|x\rangle|y\rangle$

- Each $R_Z$ gate contributes to exactly one term
- Synthesis produces one $R_Z$ gate per term
- **Profit!**

---

Quantum Phase folding

We could re-synthesize\(^1\) \(|x\rangle|y\rangle \mapsto i^{x \oplus y}|x\rangle|y\rangle\)

- Each \(R_Z\) gate contributes to exactly one term
- Synthesis produces one \(R_Z\) gate per term
- **Profit!**

Alternatively, only need to know which gates would be merged

- want to **prove** that two \(R_Z\) gates “rotate” the same paths
- replacing them with a single aggregate \(R_Z\) gate will then leave the semantics unchanged
- do this by **executing** the circuit symbolically to see which phase gates add to the same term of \(P\)

Quantum Phase folding

We could re-synthesize\(^1\) \( |x\rangle |y\rangle \mapsto i^{x \oplus y} |x\rangle |y\rangle \)

- Each \( R_Z \) gate contributes to exactly one term
- Synthesis produces one \( R_Z \) gate per term
- **Profit!**

Alternatively, only need to know which gates **would be merged**

- want to **prove** that two \( R_Z \) gates “rotate” the same paths
- replacing them with a single aggregate \( R_Z \) gate will then leave the semantics unchanged
- do this by **executing** the circuit symbolically to see which phase gates add to the same term of \( P \)

**Need path integrals to (easily) prove correctness!**

Symbolic execution

Recall:

\[ S|x\rangle = i^x |x\rangle \]
\[ T|x\rangle = \omega^x |x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]
\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can **execute** the circuit to identify phases applied to the same set of paths, along with their location in the program.

\[ P = 0 \]
Symbolic execution

Recall:

\[ S|x\rangle = i^x|x\rangle \]

\[ T|x\rangle = \omega^x|x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]

\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can execute the circuit to identify phases applied to the same set of paths, along with their location in the program

\[ P = 0 \]
Symbolic execution

Recall:

\[ S|x\rangle = i^x|x\rangle \]
\[ T|x\rangle = \omega^x|x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]
\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can execute the circuit to identify phases applied to the same set of paths, along with their location in the program

\[ P = 2\pi i \left( \frac{\pi}{4} \right)_2 (x \oplus y) \]
Symbolic execution

Recall:

\[ S|x\rangle = i^x|x\rangle \]

\[ T|x\rangle = \omega^x|x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]

\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can \textbf{execute} the circuit to identify phases applied to the same set of paths, along with their location in the program

\[ P = 2\pi i \left( \frac{\pi}{4} \right)_2 (x \oplus y) \]

\[
\begin{array}{ccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\bullet & & & \bullet & & & \\
\bullet & & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \bullet \\
\bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \bigcirc & \\
|y\rangle & & |x\rangle & & & & \\
\end{array}
\]
Symbolic execution

Recall:

\[ S|x\rangle = i^x|x\rangle \]
\[ T|x\rangle = \omega^x|x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]
\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can **execute** the circuit to identify phases applied to the same set of paths, along with their location in the program

\[ P = 2\pi i^2 \frac{\pi}{4} (x \oplus y) \]
Symbolic execution

Recall:

\[ S|x\rangle = i^x|x\rangle \]
\[ T|x\rangle = \omega^x|x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]
\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can \textbf{execute} the circuit to identify phases applied to the same set of paths, along with their location in the program

\[ P = 2\pi i[(\frac{\pi}{4})_2(x \oplus y) + (\frac{\pi}{4})_5(x \oplus y)] \]
Symbolic execution

Recall:

\[ S|x\rangle = i^x|x\rangle \]

\[ T|x\rangle = \omega^x|x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]

\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can **execute** the circuit to identify phases applied to the same set of paths, along with their location in the program.

\[ P = 2\pi i[(\frac{\pi}{4})_2 + (\frac{\pi}{4})_5](x \oplus y) \]
Symbolic execution

Recall:

\[ S|x\rangle = i^x |x\rangle \]
\[ T|x\rangle = \omega^x |x\rangle, \quad \omega = e^{\frac{\pi i}{4}} \]
\[ \text{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle \]

We can **execute** the circuit to identify phases applied to the same set of paths, along with their location in the program:

\[ P = 2\pi i[(\frac{\pi}{4})_2 + (\frac{\pi}{4})_5](x \oplus y) \implies T \text{ gates at locations 2 and 5 can be merged} \]
Branching gates

Consider the circuit

\[ T \quad H \quad T \]

Symbolically, \(|x⟩\mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{Z}^2} ω^x + 4xy |y⟩\)

Phases conditional on the output path can't be commuted.

The phase \(-1\) is tied to the \(H\) gate (i.e. it can't be merged).

So it suffices to say \(H|x⟩ = |x'⟩\) for some \(x'\).
Branching gates

Consider the circuit

Symbolically,

\[ |x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{Z}_2} \omega^{x+4xy+y} |y\rangle \]

- phases conditional on the output path can’t be commuted
- the phase \(-1\) is tied to the \(H\) gate (i.e. it can’t be merged)
Branching gates

Consider the circuit

\[
\begin{array}{c}
|0\rangle \\
\hline
\hline
T & |0\rangle \\
\hline
|1\rangle \\
\hline
ω & H & |1\rangle \\
\hline
\omega & T & |0\rangle
\end{array}
\]

Symbolically,

\[
|x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{Z}_2} \omega^x + 4xy + y |y\rangle
\]

- phases conditional on the output path can’t be commuted
- the phase $-1$ is tied to the $H$ gate (i.e. it can’t be merged)

\[\implies \text{it suffices to say } H|x\rangle = |x'\rangle \text{ for some } x'\]
Abstraction

Idea of program analysis is to **abstract** the concrete semantics, retaining enough information to be able to prove useful facts

**Lemma**

\[ U|x_1 \cdots x_n \rangle = |x'_1 \cdots x'_n \rangle \] is a sound approximation of any unitary \( U \) with respect to phase folding.
Idea of program analysis is to **abstract** the concrete semantics, retaining enough information to be able to prove useful facts.

**Lemma**

\[ U|x_1 \cdots x_n\rangle = |x'_1 \cdots x'_n\rangle \] is a sound approximation of any unitary U with respect to phase folding.
Abstraction

Idea of program analysis is to **abstract** the concrete semantics, retaining enough information to be able to prove useful facts

**Lemma**

\[ U|x_1 \cdots x_n\rangle = |x'_1 \cdots x'_n\rangle \] is a sound approximation of any unitary \( U \) with respect to phase folding.
Existential quantification

To avoid representing all $O(n|C|)$ variables in $P$, periodically **quantify out** variables which are no longer “in scope”

In practice,

- track the state in the form $|Ax\rangle$ for $A \in GA(\mathbb{Z}_2, n)$
- when a variable is quantified out, **re-normalize** parities using a **pseudoinverse** $A^g$ of $A$
- set any parities without a solution to $\bot$
Existential quantification

To avoid representing all $O(n|C|)$ variables in $P$, periodically quantify out variables which are no longer “in scope”

In practice,
- track the state in the form $|Ax\rangle$ for $A \in GA(\mathbb{Z}_2, n)$
- when a variable is quantified out, re-normalize parities using a pseudoinverse $A^g$ of $A$
- set any parities without a solution to $\bot$

E.g.,

$$P = \theta_\ell \cdot (x \oplus y)$$
Existential quantification

To avoid representing all $O(n|C|)$ variables in $P$, periodically quantify out variables which are no longer “in scope”

In practice,

- track the state in the form $|Ax\rangle$ for $A \in \text{GA}(\mathbb{Z}_2, n)$
- when a variable is quantified out, re-normalize parities using a pseudoinverse $A^g$ of $A$
- set any parities without a solution to $\bot$

E.g.,

$$P = \theta_\ell \cdot (x \oplus y) \quad \exists y. P = \theta_\ell \cdot \bot$$

Lemma

$\exists x. P$ is a sound approximation of $P$ with respect to phase folding.
Existential quantification

To avoid representing all $O(n|C|)$ variables in $P$, periodically quantify out variables which are no longer “in scope”

In practice,
- track the state in the form $|Ax\rangle$ for $A \in GA(\mathbb{Z}_2, n)$
- when a variable is quantified out, re-normalize parities using a pseudoinverse $A^g$ of $A$
- set any parities without a solution to $\perp$

E.g.,

$$P = \theta_\ell \cdot (x \oplus y) \quad \exists y. P = \theta_\ell \cdot \perp \quad \exists z. P = \theta_\ell \cdot (x \oplus y)$$

Lemma

$\exists x. P$ is a sound approximation of $P$ with respect to phase folding.
Existential quantification

To avoid representing all $O(n|C|)$ variables in $P$, periodically quantify out variables which are no longer “in scope”

In practice,

- track the state in the form $|Ax\rangle$ for $A \in GA(\mathbb{Z}_2, n)$
- when a variable is quantified out, re-normalize parities using a pseudoinverse $A^g$ of $A$
- set any parities without a solution to $\bot$

E.g.,

\[
\begin{align*}
P &= \theta_\ell \cdot (x \oplus y) \\
\exists z. P &= \theta_\ell \cdot (x \oplus y) \\
\exists y. P &= \theta_\ell \cdot \bot \\
\exists y. P &= \theta_\ell \cdot z \quad \text{if } z = x \oplus y
\end{align*}
\]

**Lemma**

$\exists x. P$ is a sound approximation of $P$ with respect to phase folding.
Extending to quantum programs

We can go even further to mixed quantum/classical programs using collecting semantics.
Extending to quantum programs

We can go even further to mixed quantum/classical programs using collecting semantics.

Consider a simple quantum WHILE language:

\[ S ::= Uq | \text{meas } q | S_1; S_2 | \text{if } E \text{ then } S_1 \text{ else } S_2 | \text{while } E \text{ do } S \]

**Definition**

The (circuit) collecting semantics \( [S]_c \) can be defined as

\[ [S]_c = \{ \tau | \tau \text{ is the sequence of gates (\& proj.) in a trace of } S \} \]

\( S_a \) is a sound approximation of \( [S]_c \) with respect to phase folding if it is a sound approximation of every trace \( \tau \).
The analysis

(Informal) phase analysis for a quantum WHILE language

\[
\begin{align*}
[R_\theta^\ell]_a : & \ e^P |x\rangle \quad \mapsto \ e^{P + \theta \ell x} |x\rangle \\
[X]_a : & \ e^P |x\rangle \quad \mapsto \ e^P |1 \oplus x\rangle \\
[CNOT]_a : & \ e^P |x\rangle |y\rangle \quad \mapsto \ e^P |x\rangle |x \oplus y\rangle \\
[U]_a : & \ e^P |x_1 x_2 \ldots x_n\rangle \quad \mapsto \ e^{\exists x_1 x_2 \ldots x_n. P} |x_1' x_2' \ldots x_n'\rangle \\
[c(U)]_a : & \ e^P |x_1\rangle |x_2 \ldots x_n\rangle \quad \mapsto \ e^{\exists x_2 \ldots x_n. P} |x_1\rangle |x_2' \ldots x_n'\rangle \\
[meas]_a : & \ e^P |x_1 x_2 \ldots x_n\rangle \quad \mapsto \ e^{\exists x_1 x_2 \ldots x_n. P} |x_1' x_2' \ldots x_n'\rangle
\end{align*}
\]

\[
[U_1]_a : \ |x\rangle \mapsto e^{P_1} |x'\rangle \quad [U_2]_a : \ |x\rangle \mapsto e^{P_2} |x''\rangle
\]

\[
[\text{if } E \text{ then } U_1 \text{ else } U_2]_a : \ |x\rangle e^P \mapsto e^{\exists x. P + \exists x. P_1 + \exists x. P_2} |x' \sqcap x''\rangle
\]

\[
[U]_a : \ |x\rangle \mapsto e^{P'} |x'\rangle
\]

\[
[\text{while } E \text{ do } U]_a : \ |x\rangle e^P \mapsto e^{\exists x. P + \exists x. P'} |x'\rangle
\]
Phase folding optimization

Compute $P$ with a phase analysis. 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

Proof idea:

▶ establish a soundness relation between abstract states of the analysis and (sets of) path integrals
▶ soundness relation encodes the fact that the path integrals are invariant under the distribution of $\sum_{\ell \in S} \theta_\ell$
▶ show that execution preserves this relation
Phase folding optimization

Compute $P$ with a phase analysis. 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

Proof idea:

- establish a soundness relation between abstract states of the analysis and (sets of) path integrals
- soundness relation encodes the fact that the path integrals are invariant under the distribution of $\sum_{\ell \in S} \theta_\ell$
- show that execution preserves this relation
Where to go from here?
Phase polynomial re-synthesis

We can go further by **re-synthesizing** parts of the phase polynomial $P$ corresponding to CNOT-dihedral circuits.

Can power up with a **range analysis** to get a sequence of overlapping synthesis problems for extra flexibility.

Phase polynomial synthesis algorithms for $T$-depth$^1$, $T$-count$^2$, $CNOT$-count$^3$, Routing$^4$, etc. work here.

---

Can we take the phase analysis further?

Consider the circuit

\[ \begin{array}{cccc}
T & H & H & T^\dagger \\
\end{array} \]

Symbolically,

\[ \text{THHT} : |x\rangle \mapsto \frac{1}{2} \sum_{y,z \in \mathbb{Z}_2} \omega^{x+4xy+4yz+7z} |z\rangle \]
Can we take the phase analysis further?

Consider the circuit

\[ T \quad H \quad H \quad T^\dagger \]

Symbolically,

\[ THHT : |x\rangle \mapsto \frac{1}{2} \sum_{y,z \in \mathbb{Z}_2} \omega^{x+4xy+4yz+7z} |z\rangle \]

We could simplify the circuit first, but what about
Interference patterns

We know

\[ HH : |x\rangle \mapsto \frac{1}{2} \sum_{y,z \in \mathbb{Z}_2} (-1)^{xy+yz} |z\rangle \]

is the identity
Interference patterns

We know

\[ HH : |x\rangle \mapsto \frac{1}{2} \sum_{y,z \in \mathbb{Z}_2} (-1)^{xy+yz} |z\rangle \]

is the identity

To analyze the interference, we can expand it out:

\[ |x\rangle \xrightarrow{y=0} \frac{1}{2} \sum_{z \in \mathbb{Z}_2} |z\rangle \]
\[ |x\rangle \xrightarrow{y=1} (-1)^x \frac{1}{2} \sum_{z \in \mathbb{Z}_2} (-1)^z |z\rangle \]
Interference patterns

We know

\[ HH : |x\rangle \mapsto \frac{1}{2} \sum_{y,z \in \mathbb{Z}_2} (-1)^{xy+yz} |z\rangle \]

is the identity

To analyze the interference, we can expand it out:
Interference patterns

If we sum over $z \in \{x, \neg x\} = \mathbb{Z}_2$ instead,

$$|x\rangle \mapsto \frac{1}{2} \sum_{y \in \mathbb{Z}_2, z \in \{x, \neg x\}} (-1)^{xy+yz} |z\rangle$$

we get a simple pattern
If we sum over $z \in \{x, \neg x\} = \mathbb{Z}_2$ instead,

$$|x\rangle \mapsto \frac{1}{2} \sum_{y \in \mathbb{Z}_2, z \in \{x, \neg x\}} (-1)^{xy+yz} |z\rangle$$

we get a simple pattern

\[ y = 0 \]
\[ y = 1 \]
If we sum over \( z \in \{x, \neg x\} = \mathbb{Z}_2 \) instead,

\[
|x\rangle \mapsto \frac{1}{2} \sum_{y \in \mathbb{Z}_2, z \in \{x, \neg x\}} (-1)^{xy + yz} |z\rangle
\]

we get a simple pattern
Lemma

For any Boolean-valued expression $P$

$$\sum_{y,z} (-1)^{z_y+y_P} |\psi(z)\rangle = 2 |\psi(P)\rangle$$

In particular only the paths where $z = P$ survive.
A slightly more precise analysis

Basic idea:

- Apply phase analysis to get $P$
- Compute the circuit's path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize $P$ with respect to the list of equalities
A slightly more precise analysis

Basic idea:

- Apply phase analysis to get $P$
- Compute the circuit’s path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize $P$ with respect to the list of equalities
A slightly more precise analysis

Basic idea:

- Apply phase analysis to get $P$
- Compute the circuit’s path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize $P$ with respect to the list of equalities
A slightly more precise analysis

Basic idea:

- Apply phase analysis to get $P$
- Compute the circuit’s path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize $P$ with respect to the list of equalities
A slightly more precise analysis

Basic idea:

- Apply phase analysis to get $P$
- Compute the circuit’s path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize $P$ with respect to the list of equalities
A slightly more precise analysis

Basic idea:

- Apply phase analysis to get $P$
- Compute the circuit’s path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize $P$ with respect to the list of equalities
Take-away:

*Classical program analysis tools can be applied in the quantum domain by taking a more operational view of quantum computation*
Thank you!