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

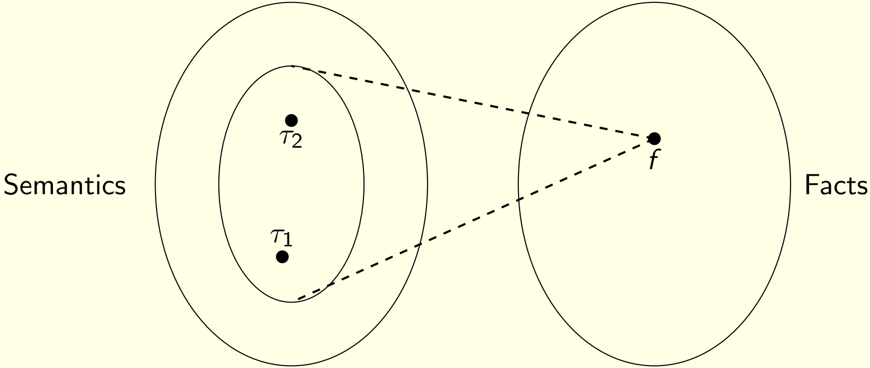
- **Proving** some facts about code
- Classical: Constant propagation, common subexpression elimination, dead code elimination, etc.
- Quantum: ???

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: (semantics + facts) $\times$ soundness relation
  - e.g., In every execution, the read of variable x at location ℓ 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

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

if (x > 0) {
        ...
}
```

Example

Constant propagation

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

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

if (x > 0) {            // x = 0 or x = 3 reach
        /* error */
}
```

Example

Constant propagation

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

```
x = 1;
y = 2;
x = 0;

if (x > 0) {                // x = 0 reaches
        ...
}
```

Semantics of quantum computing

The linear-algebraic view

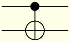
A state of n qubits is a unit vector in $\mathbb{C}^{2^n}$

$$|\psi\rangle = \sum_{\mathbf{x} \in \mathbb{Z}_2^n} \alpha_{\mathbf{x}} |\mathbf{x}\rangle, \qquad \mathbf{x} \in \{0,1\}^n = \mathbb{Z}_2^n$$

Computations change the state by applying unitary matrices to states

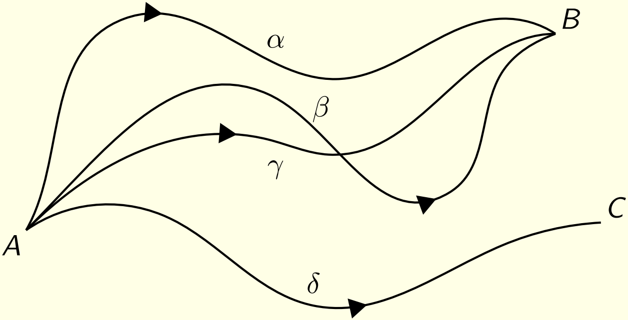
$$X = -\boxed{X}- = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad H = -\boxed{H}- = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

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

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

The path integral view

A quantum process is a collection of (classical) **paths**



The amplitude of a state is the **sum of the amplitudes of all paths** leading to it

$$A \mapsto (\alpha + \beta + \gamma)B + \delta C$$

Formal path integrals

As an object, we can represent a path integral by

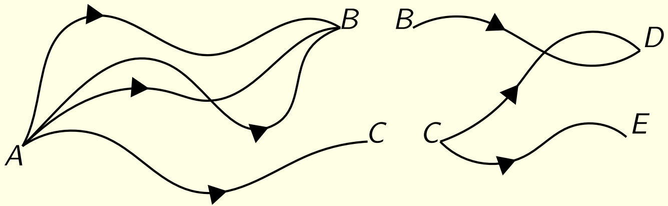
- a collection Π of **paths** $\pi : \mathbf{x} \to \mathbf{x}'$ between basis states, and
- an **amplitude** function $\Phi : \Pi \to \mathbb{C}$

The **action** is the mapping

$$|\mathbf{x}\rangle \mapsto \sum_{\pi:\mathbf{x}\to\mathbf{x}'\in\Pi_{\mathbf{x}}} \Phi(\pi)|\mathbf{x}'\rangle$$

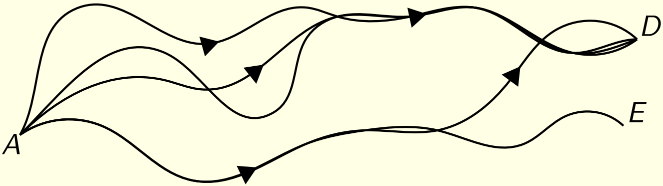
Composition

Composition of computations or circuits is path composition



Composition

Composition of computations or circuits is path composition



Syntactically, corresponds to **relational composition**

$$\Pi' \circ \Pi = \{\pi\pi' : \mathbf{x} \to \mathbf{x}' \mid \pi : \mathbf{x} \to \mathbf{x}'' \in \Pi \wedge \pi' : \mathbf{x}'' \to \mathbf{x}' \in \Pi'\}$$

$$(\Phi' \circ \Phi)(\pi' \circ \pi) = \Phi(\pi)\Phi'(\pi')$$

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

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

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

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

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:

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

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

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

Can be viewed as **delayed matrix multiplication**

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

$\implies$ **BQP $\subseteq$ PSPACE, BQP $\subseteq$ PP**

Symbolic path integrals

For d-dimensional systems we can

- label each path π by a length $k \geq n$ bit string $\mathbf{x} \in \mathbb{Z}_d^k$
- write the end point as a function $f : \mathbb{Z}_d^k \to \mathbb{Z}_d^n$
- write the amplitude $\phi : \mathbb{Z}_d^k \to \mathbb{C}$ as a function of this bit string

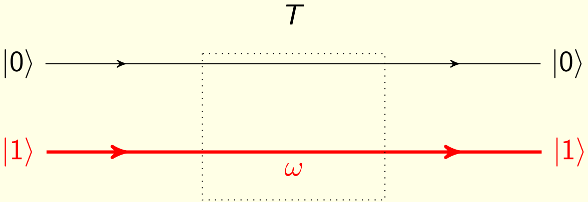
Example

Phase gates

Phase gates, e.g.

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

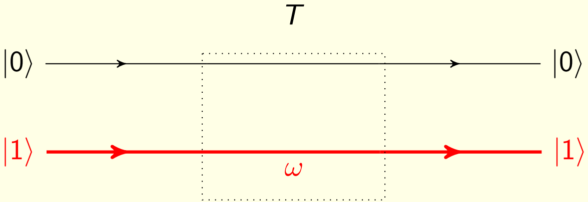
apply a **phase** conditional on certain paths



Phase gates, e.g.

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

apply a **phase** conditional on certain paths



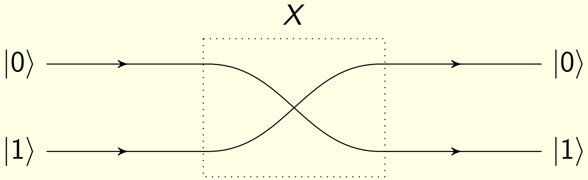
We can write this symbolically as

$$T : |x\rangle \mapsto \omega^x |x\rangle \qquad \text{for any } x \in \mathbb{Z}_2$$

Classical gates like

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

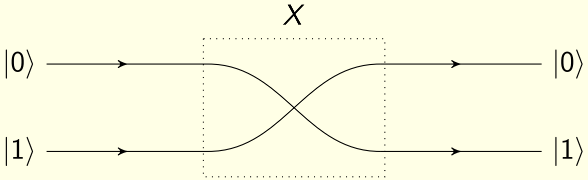
permute the states



Classical gates like

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

permute the states



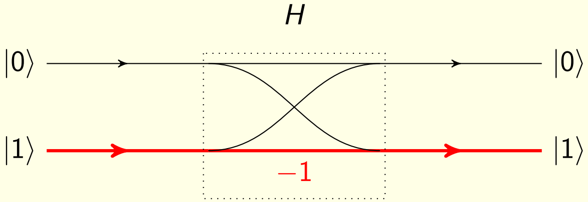
Symbolically,

$$X : |x\rangle \mapsto |1 \oplus x\rangle \qquad \text{for any } x \in \mathbb{Z}_2$$

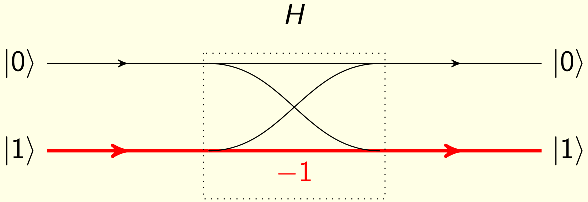
Example

Branching gates

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



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



Symbolically,

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

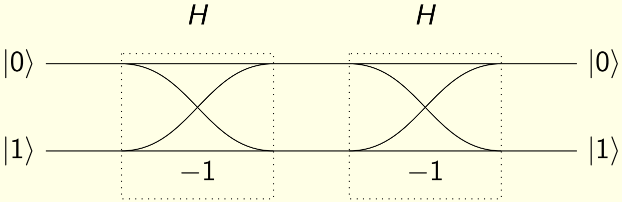
y represents the path taken

Interference

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

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

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

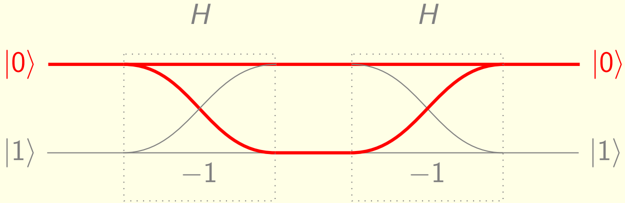


Interference

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

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

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



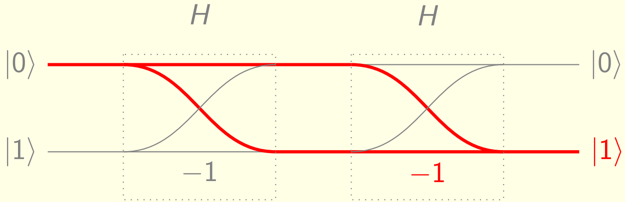
$\frac{1}{2} + \frac{1}{2} = 1$
Amplitudes add

Interference

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

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

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



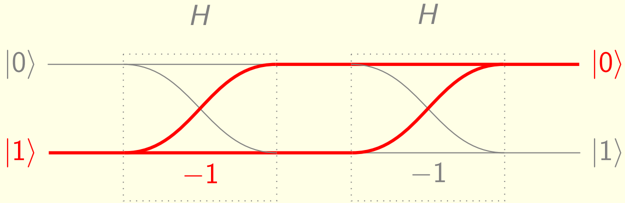
$\frac{1}{2} - \frac{1}{2} = 0$
Amplitudes cancel

Interference

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

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

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



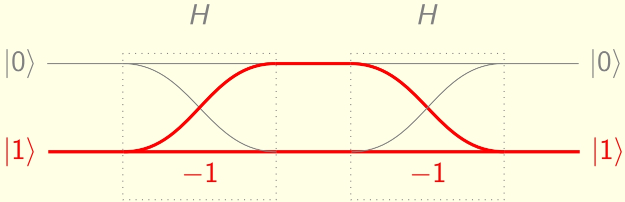
$\frac{1}{2} - \frac{1}{2} = 0$
Amplitudes cancel

Interference

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

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

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



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

Amplitudes add

Symbolic path integrals

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

Theorem

Any circuit over Clifford+R_Z can be represented symbolically as*

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

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

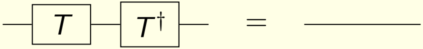
$$P(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{z} \in \mathbb{Z}^n} a_{\mathbf{z}} \chi_{\mathbf{z}}(\mathbf{x}, \mathbf{y}), \qquad \chi_{\mathbf{z}}(\mathbf{x}) = x_1 z_1 \oplus \cdots \oplus x_n z_n$$

Moreover, this representation is poly-time and -space computable.

Optimization

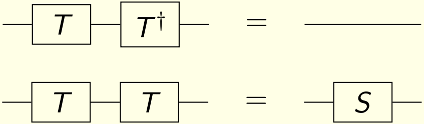
Merging gates

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



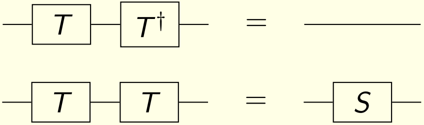
Merging gates

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

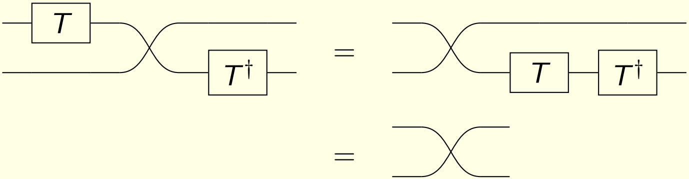


Merging gates

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

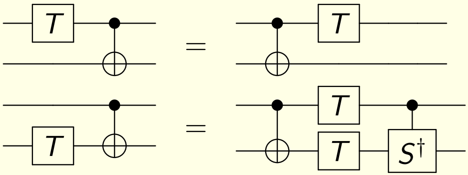


In some cases we have to **commute** gates to merge them

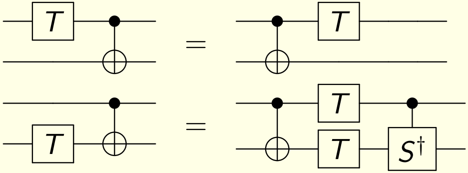


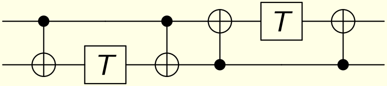
Commutations

What about more complicated cases?



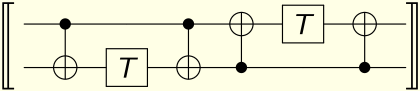
What about more complicated cases?





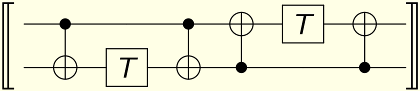
= ???

Paths & phases



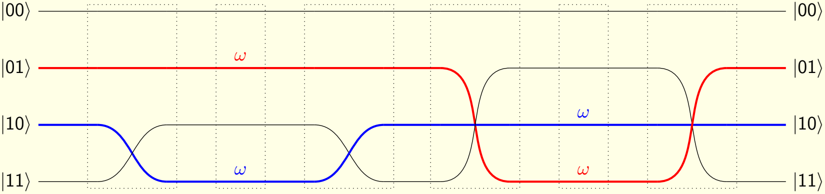
$$= |x\rangle|y\rangle \mapsto i^{x\oplus y}|x\rangle|y\rangle$$

Paths & phases

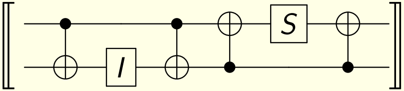


$= |x\rangle|y\rangle \mapsto i^{x \oplus y}|x\rangle|y\rangle$

As a collection of paths:

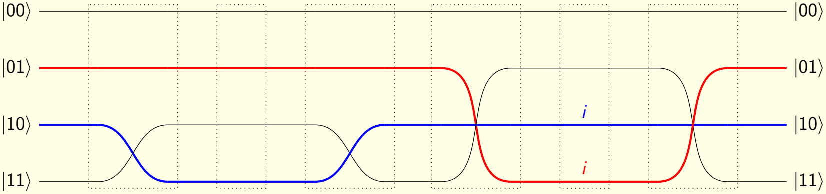


Paths & phases



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

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!**

[1] M. Amy, D. Maslov, M. Mosca, Polynomial-Time T-Depth Optimization of Clifford+T Circuits via Matroid Partitioning. IEEE Tr. CAD (2014).

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

[1] M. Amy, D. Maslov, M. Mosca, Polynomial-Time T-Depth Optimization of Clifford+T Circuits via Matroid Partitioning. IEEE Tr. CAD (2014).

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

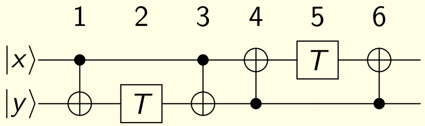
Need path integrals to (easily) prove correctness!

[1] M. Amy, D. Maslov, M. Mosca, Polynomial-Time T-Depth Optimization of Clifford+T Circuits via Matroid Partitioning. IEEE Tr. CAD (2014).

Recall:

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

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



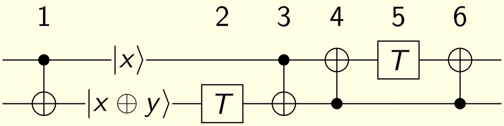
$P = 0$

Symbolic execution

Recall:

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

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



$P = 0$

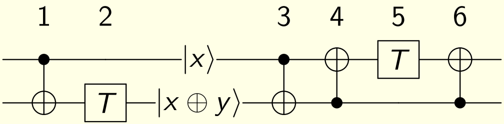
Recall:

$$S|x\rangle = i^x|x\rangle$$

$$T|x\rangle = \omega^x|x\rangle, \qquad\qquad \omega = e^{\frac{\pi i}{4}}$$

$$\mathrm{CNOT}|x\rangle|y\rangle = |x\rangle|x \oplus y\rangle$$

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



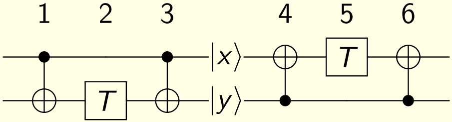
$P = 2\pi i(\frac{\pi}{4})_2(x \oplus y)$

Symbolic execution

Recall:

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

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



$P = 2\pi i(\frac{\pi}{4})_2(x \oplus y)$

Symbolic execution

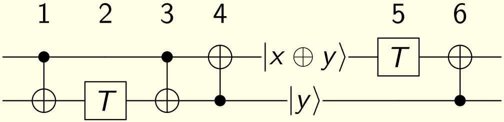
Recall:

$$S|x\rangle = i^x|x\rangle$$

$$T|x\rangle = \omega^x|x\rangle, \qquad \omega = e^{\frac{\pi i}{4}}$$

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

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



$P = 2\pi i(\frac{\pi}{4})_2(x \oplus y)$

Symbolic execution

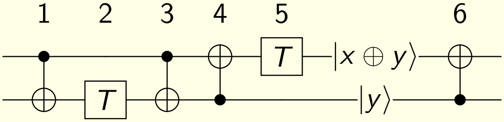
Recall:

$$S|x\rangle = i^x|x\rangle$$

$$T|x\rangle = \omega^x|x\rangle, \qquad \omega = e^{\frac{\pi i}{4}}$$

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

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

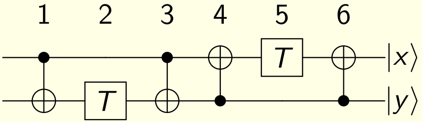


$$P = 2\pi i[(\tfrac{\pi}{4})_2(x \oplus y) + (\tfrac{\pi}{4})_5(x \oplus y)]$$

Recall:

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

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



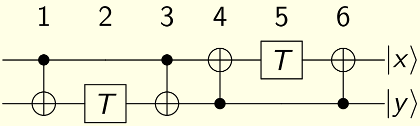
$P = 2\pi i[(\frac{\pi}{4})_2 + (\frac{\pi}{4})_5](x \oplus y)$

Symbolic execution

Recall:

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

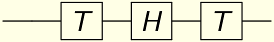
We can **execute** the circuit to identify phases applied to the same set of paths, along with their location in the program



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

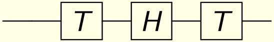
Branching gates

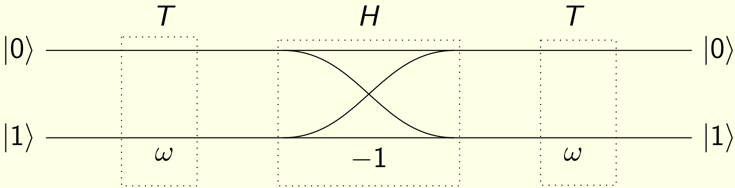
Consider the circuit



Branching gates

Consider the circuit





Symbolically,

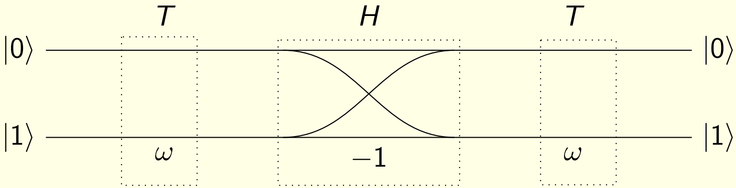
$$|x\rangle \mapsto \frac{1}{\sqrt{2}} \sum_{y \in \mathbb{Z}_2} \omega^{x+4xy+y} |y\rangle$$

- 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

$$-\boxed{T}-\boxed{H}-\boxed{T}-$$



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)

$\implies$ **it suffices to say $H|x\rangle = |x'\rangle$ 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

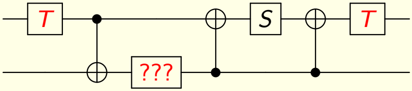
$U|x_1 \cdots x_n\rangle = |x_1' \cdots x_n'\rangle$ *is a sound approximation of any unitary* U *with respect to phase folding.*

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.

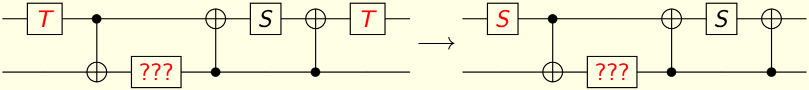


Abstraction

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 $|A\mathbf{x}\rangle$ for $A \in \mathrm{GA}(\mathbb{Z}_2, n)$
- when a variable is quantified out, **re-normalize** parities using a **pseudoinverse** A^g of A
- set any parities without a solution to $\perp$

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 $|A\mathbf{x}\rangle$ for $A \in \mathrm{GA}(\mathbb{Z}_2, n)$
- 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.,

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

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 $|A\mathbf{x}\rangle$ for $A \in \mathrm{GA}(\mathbb{Z}_2, n)$
- 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.,

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

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 $|A\mathbf{x}\rangle$ for $A \in \mathrm{GA}(\mathbb{Z}_2, n)$
- 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.,

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

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

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 $|A\mathbf{x}\rangle$ for $A \in \mathrm{GA}(\mathbb{Z}_2, n)$
- 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.,

$$\begin{aligned} P &= \theta_\ell \cdot (x \oplus y) & \exists y.P &= \theta_\ell \cdot \perp \\ \exists z.P &= \theta_\ell \cdot (x \oplus y) & \exists y.P &= \theta_\ell \cdot z \quad \text{if } z = x \oplus y \end{aligned}$$

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

$$S ::= U\ q \ \mid\ \mathit{meas}\ q \ \mid\ S_1;\ S_2 \ \mid\ \mathit{if}\ E\ \mathit{then}\ S_1\ \mathit{else}\ S_2 \ \mid\ \mathit{while}\ E\ \mathit{do}\ S$$

Definition

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

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

S_a is a sound approximation of $[\![S]\!]_c$ with respect to phase folding if it is a sound approximation of every trace τ

The analysis

(Informal) phase analysis for a quantum WHILE language

$$\begin{aligned}
[\![R_Z^\ell(\theta)]\!]_a &: e^P|x\rangle && \mapsto e^{P+\theta_\ell x}|x\rangle \\
[\![X]\!]_a &: e^P|x\rangle && \mapsto e^P|1 \oplus x\rangle \\
[\![\mathrm{CNOT}]\!]_a &: e^P|x\rangle|y\rangle && \mapsto e^P|x\rangle|x \oplus y\rangle \\
[\![U]\!]_a &: e^P|x_1x_2\ldots x_n\rangle && \mapsto e^{\exists x_1x_2\ldots x_n.P}|x_1'x_2'\ldots x_n'\rangle \\
[\![c(U)]\!]_a &: e^P|x_1\rangle|x_2\ldots x_n\rangle && \mapsto e^{\exists x_2\ldots x_n.P}|x_1\rangle|x_2'\ldots x_n'\rangle \\
[\![meas]\!]_a &: e^P|x_1x_2\ldots x_n\rangle && \mapsto e^{\exists x_1x_2\ldots x_n.P}|x_1'x_2'\ldots x_n'\rangle
\end{aligned}$$

$$\frac{[\![U_1]\!]_a : |\mathbf{x}\rangle \mapsto e^{P_1}|\mathbf{x}'\rangle \qquad [\![U_2]\!]_a : |\mathbf{x}\rangle \mapsto e^{P_2}|\mathbf{x}''\rangle}{[\![\mathit{if}\ E\ \mathit{then}\ U_1\ \mathit{else}\ U_2\ ]\!]_a : |\mathbf{x}\rangle e^P \mapsto e^{\exists \mathbf{x}.P+\exists \mathbf{x}.P_1+\exists \mathbf{x}.P_2}|\mathbf{x}' \sqcap \mathbf{x}''\rangle}$$

$$\frac{[\![U]\!]_a : |\mathbf{x}\rangle \mapsto e^{P'}|\mathbf{x}'\rangle}{[\![\mathit{while}\ E\ \mathit{do}\ U\ ]\!]_a : |\mathbf{x}\rangle e^P \mapsto e^{\exists \mathbf{x}.P+\exists \mathbf{x}.P'}|\mathbf{x}'\rangle}$$

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 $\ell \in S \setminus \{\ell_0\}$

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

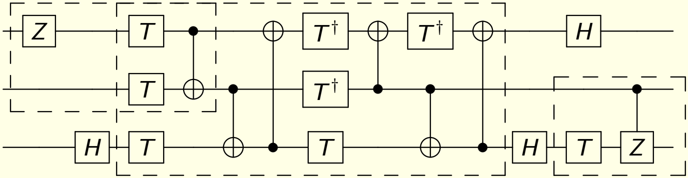
Proof idea:

- 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

[1] Amy, Maslov, Mosca, IEEE Tr. CAD (2014).
[2] Heyfron, Campbell, Quantum Science & Technology (2018).
[3] Amy, Azimzadeh, Mosca, Quantum Science & Technology (2018).
[4] Meijer-van de Griend, Duncan, QPL (2020).

Can we take the phase analysis further?

Consider the circuit

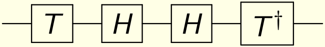
$$
\text{---}\boxed{T}\text{---}\boxed{H}\text{---}\boxed{H}\text{---}\boxed{T^\dagger}\text{---}
$$

Symbolically,

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

Can we take the phase analysis further?

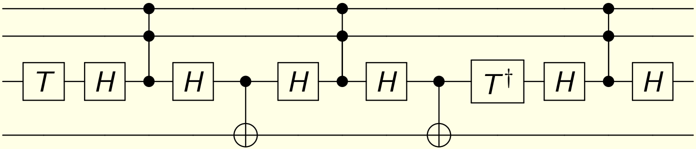
Consider the circuit



Symbolically,

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

We could simplify the circuit first, but what about



Interference patterns

We know

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

is the identity

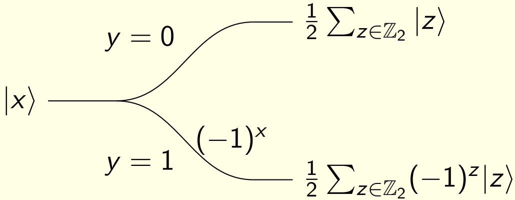
Interference patterns

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:



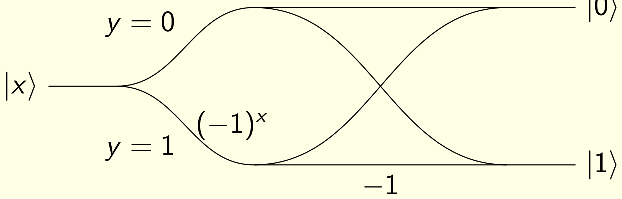
Interference patterns

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:

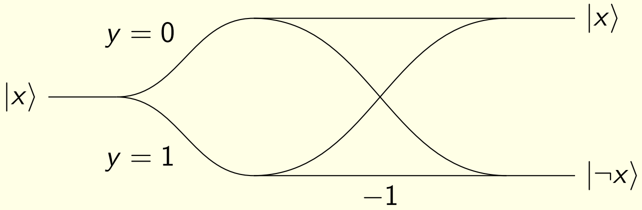


Interference patterns

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

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

we get a simple pattern

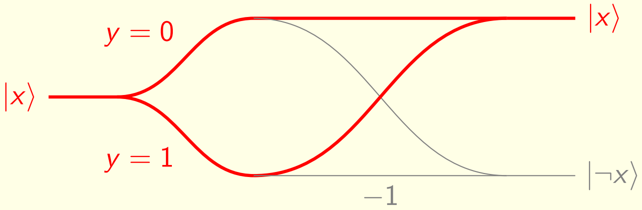


Interference patterns

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

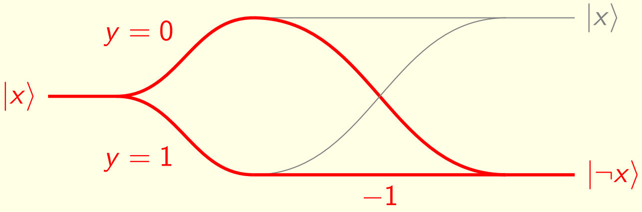


Interference patterns

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



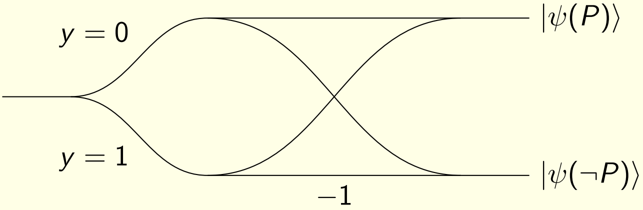
Generalization

Lemma

For any Boolean-valued expression P

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

In particular **only the paths where $z = P$ survive**



A slightly more precise analysis

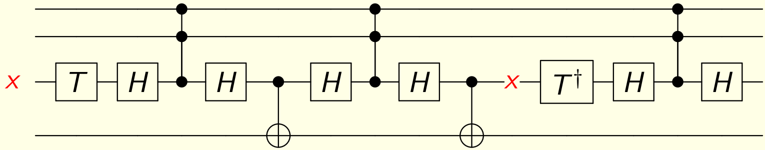
Basic idea:

- 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:

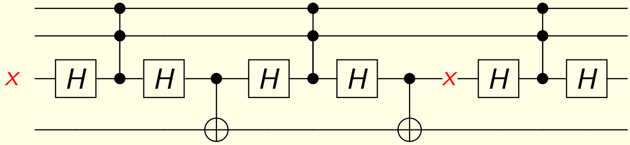
- Apply phase analysis to get P
- Compute the circuit's path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize P with respect to the list of equalities



A slightly more precise analysis

Basic idea:

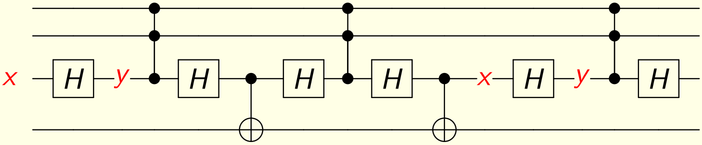
- Apply phase analysis to get P
- Compute the circuit's path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize P with respect to the list of equalities



A slightly more precise analysis

Basic idea:

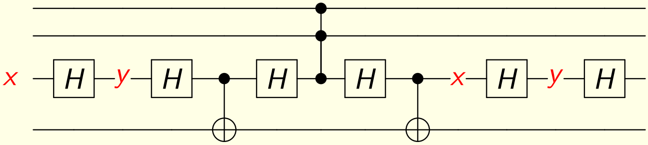
- Apply phase analysis to get P
- Compute the circuit's path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize P with respect to the list of equalities



A slightly more precise analysis

Basic idea:

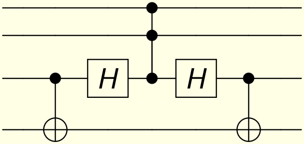
- Apply phase analysis to get P
- Compute the circuit's path integral
- Apply interference reductions to get a list of equalities $z_i = P_i$
- Normalize P with respect to the list of equalities



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



Conclusion

Take-away:

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

Thank you!