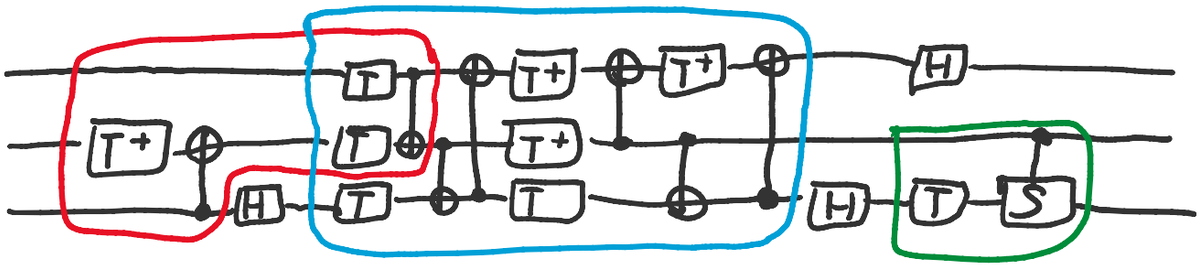
Optimizing Clifford+T circuits

While CNOT-dihedral re-synthesis only applies to the (non-universal) $\{CNOT, X, R_m\}$ gate set, we can apply it to optimize CNOT-dihedral **sub-circuits** of general Clifford+T circuits.

Even with **optimal** re-synthesis, different partitionings of a circuit may result in different costs.

E.g.

Consider the below circuit. There are **3 maximal** CNOT-dihedral sub-circuits: **1**, **2**, & **3**



Since **1** and **2** overlap, the order of re-synthesis matters — re-synthesizing **1** first may remove the gates in the intersection, but associating those gates with **2** instead could have led to better optimization.

One option is to mark gates in the intersection and try to find a **global optimum** over a set of synthesis problems. We're still just **sweeping the floors on the Titanic** though because the optimization fails to take into account the **whole circuit**. As in classical compilers, the best optimizations are **whole program optimizations**, typically using **static analysis** to make it tractable.

Clifford+T representations

To represent a **whole program** over $\{H, CNOT, T\}$ using phase polynomials, we can note that

$$H : |x\rangle \longmapsto \frac{1}{\sqrt{2}} \sum_{x' \in \mathbb{Z}_2} (-1)^{xx'} |x'\rangle$$

In particular, $\frac{1}{\sqrt{2}} \sum_{x' \in \mathbb{Z}_2} (-1)^{xx'} |x'\rangle = \frac{|0\rangle + (-1)^x |1\rangle}{\sqrt{2}}$

$= |+\rangle$ if $x = 0$, $|-\rangle$ if $x = 1$

Noting that $(-1)^{xx'} = i^{x + x' - (x \oplus x')}$, we almost have a **phase polynomial representation**, but with one extra variable (x') which is summed over (+ norm. factor)

Prop

Any circuit C over $\{H, CNOT, R_m\}$ can be written as

$$\text{①} \quad [\![C]\!] : |\vec{x}\rangle \longmapsto \frac{1}{\sqrt{2}^k} \sum_{\vec{x}' \in \mathbb{Z}_2^k} e^{2\pi i \frac{1}{m} P(\vec{x}, \vec{x}')} |A(\vec{x}, \vec{x}') + \vec{b}\rangle$$

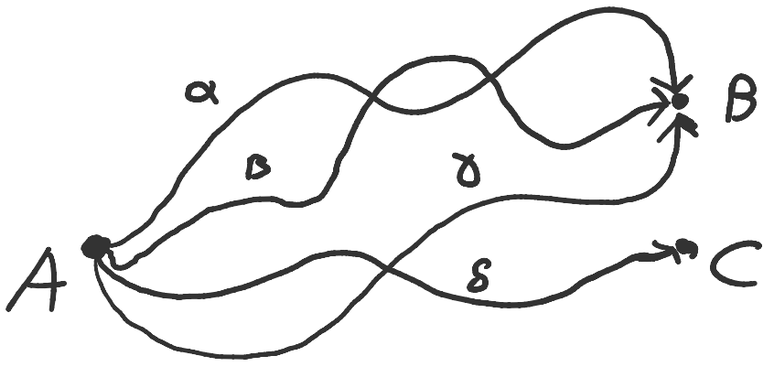
for some phase polynomial P and $A \in M_{n \times n+k}(\mathbb{Z}_2)$

(Sum-over-paths)

The representation ① is known as the (circuit) **sum-over-paths**. The summed variables x_i' are called **path variables** and can be thought of as identifying the branch taken (in superposition) at some **Hadamard** gate. The sum-over-paths corresponds to Feynman's **path integral formulation** of quantum mechanics.

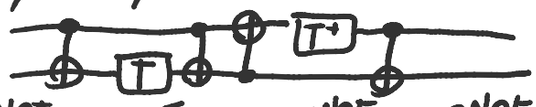
The Feynman path integral

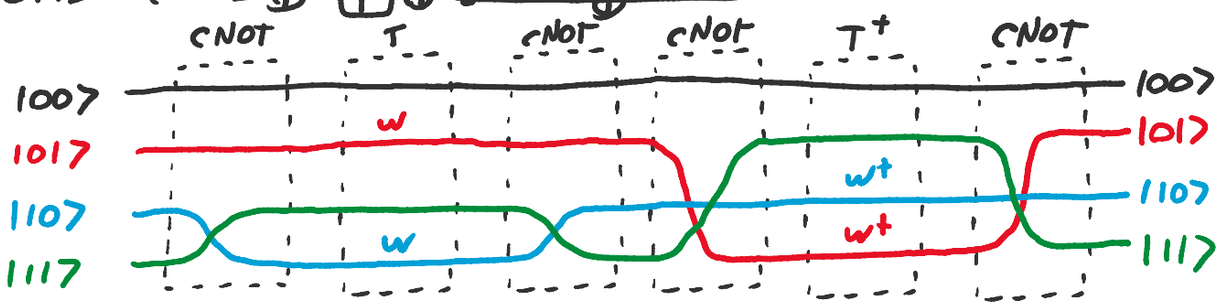
In Feynman's **path integral formulation** of QM, the evolution of a particle or system is viewed as the **sum/integral** over the different **(classical)** paths/evolutions it takes **in superposition.**



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

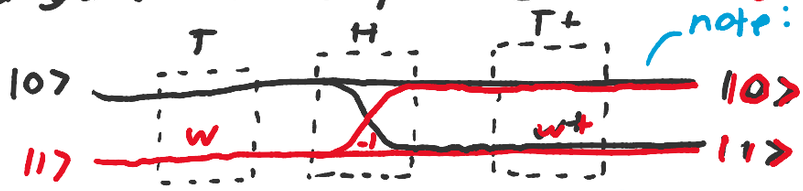
E.g.
We can analyze the evolution of circuits in the path integral style by tracking the evolution of an input basis state

Consider 



Each path ends in the state it started, and the red (01) and blue (10) paths both pick up a phase of $w \cdot w^\dagger = 1$, so the overall effect of the circuit is nothing.

E.g.
Hadamard gates correspond to **branching paths**



note: each path has magnitude $\frac{1}{\sqrt{2}}$

phases don't easily slide/commute!

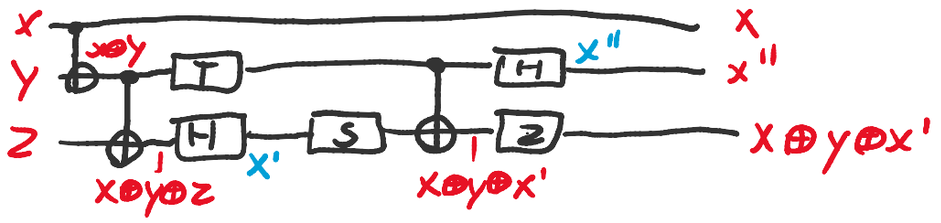
Annotated circuits

As with phase polynomial representations, the sum-over-paths can be determined by annotating

E.g.

Annotating ① basis state transformations, and
② branches,

the sum-over-paths representation of the following circuit can be read out by collecting the phase terms and summing over all branches:



SOP form: $|xyz\rangle \mapsto \frac{1}{2} \sum_{x',x''} \omega^{P(x,y,z,x',x'')} |x\, x''\, (x \oplus y \oplus x')\rangle$

$$P(x,y,z,x',x'') = x \oplus y + 4x'(x \oplus y \oplus z) + 2x' + 4(x \oplus y \oplus x') + 4|y''(x \oplus y)$$

$$= x \oplus y + 2x' + 2(x \oplus y \oplus z) + 6(x' \oplus x \oplus y \oplus z) + 2x'$$

$$4x + 4y + 4x' + 2x'' + 2(x \oplus y) + 6(x'' \oplus x \oplus y)$$

$$= x \oplus y + 2(x \oplus y \oplus z) + 6(x' \oplus x \oplus y \oplus z) + 4x + 4y + 2x'' + 2(x \oplus y) + 6(x'' \oplus x \oplus y)$$

This is tedious! Computers are much better

Prop.

The sum-over-paths representation of a Clifford+T circuit C can be computed in poly(|C|) time & space

Sum-over-paths re-synthesis

Synthesizing a circuit from a sum-over-paths is generically **hard** (at least **coNP-hard**)

Ex.

Consider a sum-over-paths with k distinct variables $\{x_i\}$ and phases **only of the form** $(-1)^{x_i x_j} = i^{x_i + x_j - (x_i \oplus x_j)}$. We can draw the phase polynomial as a **graph** where a phase of $(-1)^{x_i x_j}$ means an edge between x_i & x_j

$\longrightarrow (-1)^{x_1x_2 + x_1x_3 + x_1x_4 + x_2x_4}$

we can implement phases by either

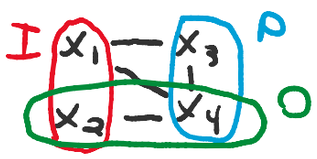
$CZ : |x_i\rangle|x_j\rangle \longmapsto (-1)^{x_i x_j}|x_i\rangle|x_j\rangle$, or

$H : |x_i\rangle \longmapsto \frac{1}{\sqrt{2}} \sum_{x_j} (-1)^{x_i x_j}|x_j\rangle$

Since H "**consumes**" an x_i, given subsets (I, P, O) of $\{x_i\}$ corr. to inputs, path variables, and outputs, synthesizing a circuit over $\{H, CZ\} \simeq \{H, CX, S\}$ amounts to finding an assignment $V(x_j) = x_i$ for all $x_j \in P$ s.t.

1. $V(x_j) = x_i \Rightarrow (x_i, x_j) \in E$
2. $V(x_j) = x_i = V(x_k) \Rightarrow x_j = x_k$
3. $V(\{x_i\}) \cap O = \emptyset$
4. $V(x_j) = x_i \Rightarrow x_i < x_j$ is a partial order

These constraints are unsatisfiable for the graph



But it can be satisfied by applying a **linear transform first**

$x_2 \to x_1 \oplus x_2$, i.e. $x_1 \to x_1$, $x_2 \to x_1 \oplus x_2$

Where do we go from here?

Classical program optimization also runs into inherent problems of intractability (or even uncomputability...). To solve this problem, we use static program analysis to prove properties which allow a compiler to safely perform an optimization.

Our final (lecture) topic is the phase folding optimization which uses the sum-over-paths model to prove two phase gates can be merged into one.

We first start by showing that the phase polynomial can be used to optimize circuits without re-synthesis by identifying which gates add to the same term of P and hence can be replaced with a single gate. Then we will look at proving (via an algorithm) that two gates add to the same term without explicitly computing the sum-over-paths.

Phase folding (first pass)

Basic insight

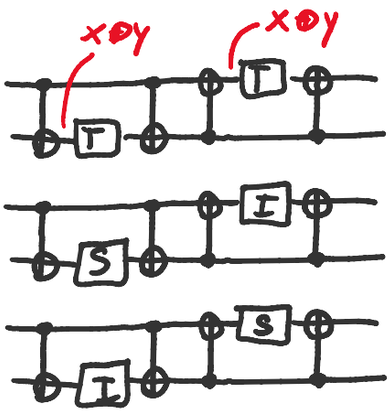
Given the path sum of a Clifford+T circuit

$$|\vec{x}\rangle \longmapsto \frac{1}{N}\sum_{\vec{y}} w^{P(\vec{x},\vec{y})}\,|f(\vec{x},\vec{y})\rangle$$

if two T gates (say T_1 & T_2) contribute to the **same term** of P, e.g. $w^{x_{\vec{z}}(\vec{x},\vec{y})}$ & $w^{x_{\vec{z}'}(\vec{x},\vec{y})}$, $\vec{z}=\vec{z}'$,

Then the circuit with T_1 replaced with S and T_2 replaced with I **has the same path integral**.

Ex.



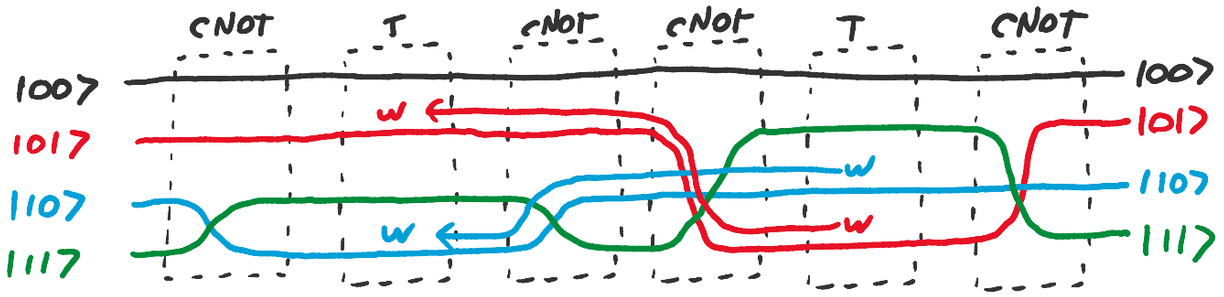
$|xy\rangle \longmapsto w^{x\oplus y} w^{xy}\,|xy\rangle$

$|xy\rangle \longmapsto w^{2(x\oplus y)}\,|xy\rangle$

$|xy\rangle \longmapsto w^{2(xy)}\,|xy\rangle$

} the same

Seen another way, if the phase gates contribute to the same term, the **affected paths are the same** so we may **slide** the phases along the paths to match up

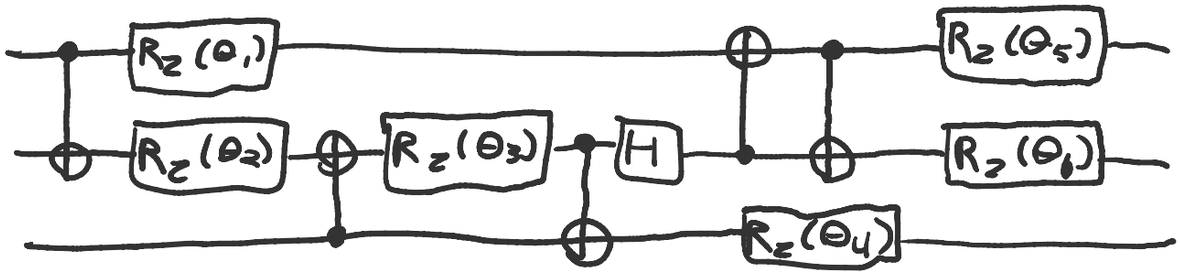


(Phase folding over Clifford + $R_z(\theta)$)

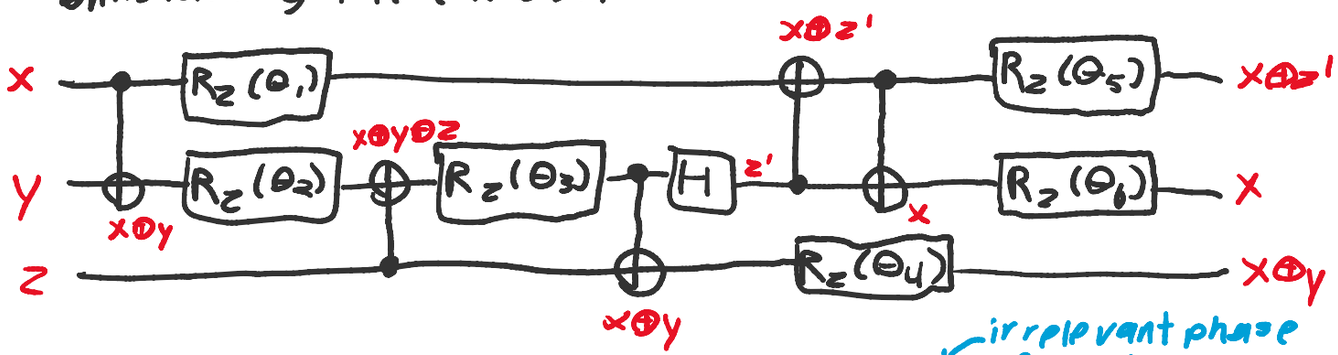
1. Compute the path sum, taking note of which R_z gates contribute to each term of P.
2. For each term $(\theta_1+\cdots+\theta_k)\,x_{\vec{y}}(\vec{x})$ of P,
3. Replace $R_z(\theta_1)$ with $R_z(\theta_1+\cdots+\theta_k)$
4. Replace each $R_z(\theta_j)$, $j>1$ with I

Example

To apply the phase folding optimization to the following circuit,



we first calculate the sum-over-paths by annotating the circuit

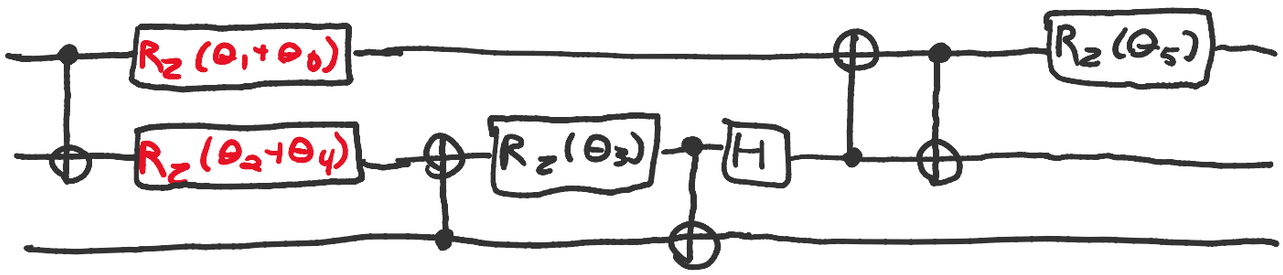


$$\therefore \quad [\![C]\!] : |xyz\rangle \mapsto \frac{1}{\sqrt{2}^{z'}} \omega^{2\pi i P(x,y,z,z')} (-1)^{Q} |(x \oplus z') \times (x \oplus y)\rangle$$

← irrelevant phase from H

$$P(x,y,z,z') = (\theta_1 + \theta_6)x + (\theta_2 + \theta_4)x \oplus y + \theta_3 x \oplus y \oplus z + \theta_5 x \oplus z'$$

Next we merge any of the rotations which add to the same term of P, notably θ_1 & θ_6 and θ_2 & θ_4, by simply changing the angles. The optimized circuit is shown below



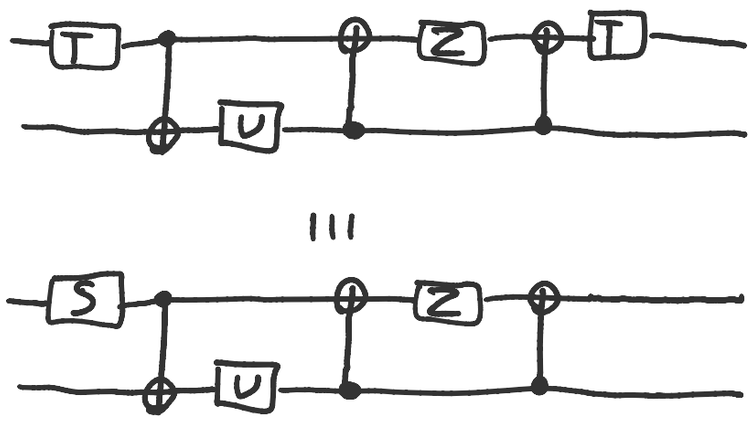
Note that the angles θ_i need not be real numbers and in particular can be circuit parameters.

Beyond exact representation

What if it's too expensive (or impossible) to compute an exact representation as a path sum?

Ex.

Suppose U is some gate for which we don't know its path sum. How could we find this optimization?



Note that if $U=H$, then the path sum would be

$$|xy\rangle \mapsto \frac{1}{\sqrt{2}} \sum_z w^{x + 4(x \cdot y)z + 4(x \cdot z) + x} |xz\rangle$$

where we can readily observe that the T gates, corresponding to w^x phases can be merged.

The key idea is that even if we know nothing about what U does, we can still prove that the inputs to the T gates are the same by treating U as if it were an H gate. This is an approximation of U.

To understand approximation, we need to understand how classical static analysis works (roughly)

Analysis-based optimization

Static analysis is used in most classical compilers to perform optimization. Roughly, the goal is to prove properties of a program that hold on any execution.

E.g. in C,

```
        ⋮
i       x = f(y);
i+1     x = 0;
        ⋮
```

Property: in any execution of the program, no read of x will ever read the write on line i.

Optimization: remove line i

Since we can't run a classical program on every input and check whether the property holds, in analysis-based optimization a (tractable) set of properties which are sufficient to prove the property in question are used to describe the current state. Commands are executed by updating the properties of the current state correspondingly

E.g.

Say we want to know whether a variable is positive or negative at the end of the computation. We might try to prove this by keeping track of whether each variable is positive or negative.

```
1   x = 2      ⟶  {x ≥ 0}
2   y = -1     ⟶  {x ≥ 0, y < 0}
3   x = x + 1  ⟶  {x ≥ 0, y < 0}
4   z = x + y  ⟶  {x ≥ 0, y < 0, z ?}
```

If we only know that $x \geq 0, y < 0$, we don't know whether $z = x + y \geq 0$ or < 0

Abstraction & approximation

The representation of a state (or set of states) S by a property (or set of properties) P is an abstraction. If P is a property which is true of some larger set of states $S' \supset S$, then P over-approximates S. If we have sensible (monotone) maps

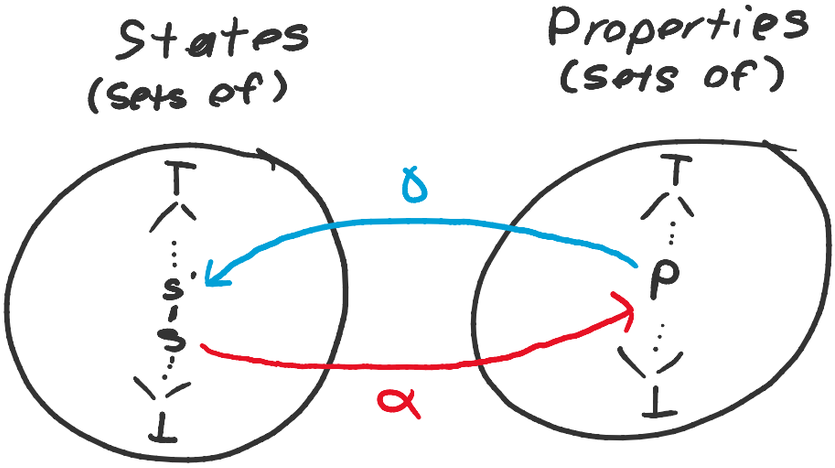
α : States $\longrightarrow$ Properties (abstraction)

γ : Properties $\longrightarrow$ States (concretization)

then α (soundly) over-approximates α means that α & γ are in a Galois connection:

$$S \subseteq \gamma \cdot \alpha(S) \quad \forall S$$

We visualize this as



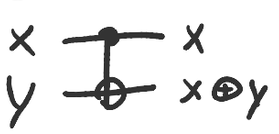
We won't use any of this directly, but it's helpful to understand the set up & intuition!

(For the category theorists - note that α & γ are adjoint functors :))

Annotations as Properties

Our circuit annotations are really asserting a property, namely that a relationship holds between the state (in the comp. basis, by linearity) at two locations in the circuit.

E.g.

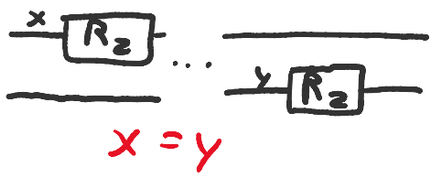


means that after applying CNOT to $|xy\rangle$, the result projected onto the computational basis is $|x'y'\rangle$ where $x' = x$, $y' = x \oplus y$

means that after applying X to $|x\rangle$, the result $|x'\rangle$ satisfies $x' = 1 \oplus x$

means that after applying H to $|x\rangle$, the result is $|x'\rangle$ such that no relation holds between x & x' - e.g. if $x = 0$, then x' can be 0 or 1.

The phase folding algorithm only merges gates if their inputs are (provably) the same, i.e.



Intuitively, if our goal is to prove that the state (i.e. the path) is the same at two locations, the least precise way we can model an arbitrary gate is to assume that no relationship holds between its inputs and outputs.

Prop

For any gate U, $U|\vec{x}\rangle = |\vec{x}'\rangle$ is a sound over-approximation of U with respect to phase folding

Phase folding (second version)

The analysis-style phase folding optimization can be specified by giving an abstract semantics

$$[\![X]\!]_q : e^{iP}|x\rangle \longmapsto e^{iP}|1 \oplus x\rangle$$

$$[\![CNOT]\!]_q : e^{iP}|xy\rangle \longmapsto e^{iP}|x(x \oplus y)\rangle$$

$$[\![R_z(\theta)]\!]_q : e^{iP}|x\rangle \longmapsto e^{i(P+\theta x)}|x\rangle$$

$$[\![U]\!]_q : e^{iP}|\vec{x}\rangle \longmapsto e^{iP}|\vec{x}'\rangle$$ where U is any unitary on n bits and $\vec{x}'$ is a vector of fresh variables

(Quantum phase folding)

Given a circuit C over any set of gate symbols,

1. Compute $[\![C]\!]_q|\vec{x}\rangle = e^{iP}|\vec{x}'\rangle$ sequentially
   (i.e. $[\![U_2 \cdot U_1]\!]_q|\vec{x}\rangle = [\![U_2]\!]_q([\![U_1]\!]_q|\vec{x}\rangle)$)
2. Merge any gates contributing to the same term of P

Prop (soundness)

Let C' be the circuit obtained by phase folding C.
Then $[\![C']\!] = [\![C]\!]$ (i.e. $C' = C$ as matrices)

Pf (sketch)

The idea is to show that P is a sound over-approx. of the real phase polynomial P'. That is, if P has a term with coefficient $\theta_1 + \theta_2$, then so does P'.