Tracking Quantum Circuits By Polynomials

ACSS 2016, Kolkata

Kenneth W. Regan[1]
University at Buffalo (SUNY)

13 August, 2016

[1]Includes joint work with Amlan Chakrabarti, U. Calcutta, and prospectively students...

Quantum Computers

Quantum Computers

"A Particular Wave of the Future. . . "

Quantum Computers

"A Particular Wave of the Future. . . "

- Apparently capable of out-performing current computers vastly at certain tasks. . .

Quantum Computers

"A Particular Wave of the Future..."

- Apparently capable of out-performing current computers vastly at certain tasks...
- ...most notably Factoring, Discete Logarithm, and simulating quantum systems.

Quantum Computers

“A Particular Wave of the Future. . . ”

- Apparently capable of out-performing current computers vastly at certain tasks. . .
- . . . most notably Factoring, Discete Logarithm, and simulating quantum systems.
- If, that is, we can build them *scalably*,

Quantum Computers

“A Particular Wave of the Future. . . ”

- Apparently capable of out-performing current computers vastly at certain tasks. . .
- . . . most notably Factoring, Discete Logarithm, and simulating quantum systems.
- If, that is, we can build them *scalably*, and if those tasks are not classically easy.

Quantum Computers

"A Particular Wave of the Future..."

- Apparently capable of out-performing current computers vastly at certain tasks...
- ...most notably Factoring, Discete Logarithm, and simulating quantum systems.
- If, that is, we can build them *scalably*, and if those tasks are not classically easy.
- Central notion: **Quantum Circuits**.

Quantum Computers

"A Particular Wave of the Future. . . "

- Apparently capable of out-performing current computers vastly at certain tasks. . .
- . . . most notably Factoring, Discete Logarithm, and simulating quantum systems.
- If, that is, we can build them *scalably*, and if those tasks are not classically easy.
- Central notion: **Quantum Circuits**.
- Hardly the only player—quantum adiabatic machines are closer to built and quantum communication systems are already deployed.

Boolean Circuits Have. . .

Boolean Circuits Have. . .

- n inputs $x_1, \ldots, x_n \in \{0,1\}^n$

Boolean Circuits Have...

- n inputs $x_1, \ldots, x_n \in \{0,1\}^n$
- $r \geq 1$ outputs $z_1, \ldots, z_r$

Boolean Circuits Have...

- n inputs $x_1, \ldots, x_n \in \{0,1\}^n$
- $r \geq 1$ outputs $z_1, \ldots, z_r$
- Maybe h-many nondeterministic inputs $y_1, \ldots, y_h$?

Boolean Circuits Have...

- n inputs $x_1, \ldots, x_n \in \{0,1\}^n$
- $r \geq 1$ outputs $z_1, \ldots, z_r$
- Maybe h-many nondeterministic inputs $y_1, \ldots, y_h$?
- m gates $g_1, \ldots, g_m$ (wlog. all NAND)

Boolean Circuits Have...

- n inputs $x_1, \ldots, x_n \in \{0,1\}^n$
- $r \geq 1$ outputs $z_1, \ldots, z_r$
- Maybe h-many nondeterministic inputs $y_1, \ldots, y_h$?
- m gates $g_1, \ldots, g_m$ (wlog. all NAND)
- Up to $2m + r$ wires (if fan-in ≤ 2)

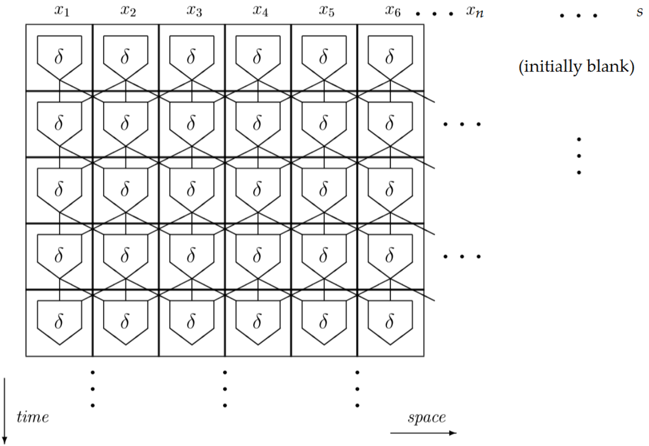
Boolean Circuits Have...

- n inputs $x_1, \ldots, x_n \in \{0,1\}^n$
- $r \geq 1$ outputs $z_1, \ldots, z_r$
- Maybe h-many nondeterministic inputs $y_1, \ldots, y_h$?
- m gates $g_1, \ldots, g_m$ (wlog. all NAND)
- Up to $2m + r$ wires (if fan-in ≤ 2)
- Each wire has a definite 0-1 value.

Boolean Circuits Have...

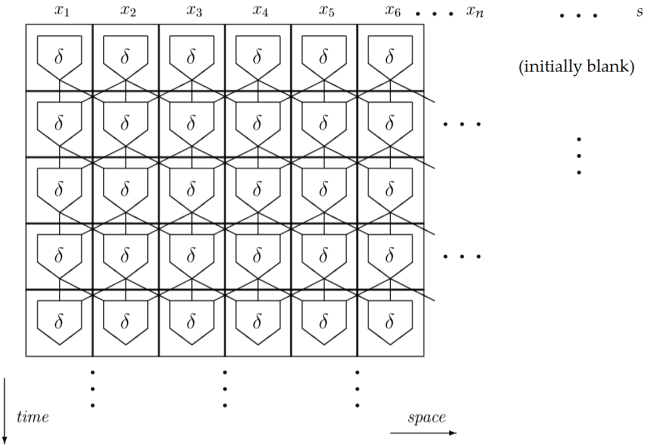
- n inputs $x_1, \ldots, x_n \in \{0,1\}^n$
- $r \geq 1$ outputs $z_1, \ldots, z_r$
- Maybe h-many nondeterministic inputs $y_1, \ldots, y_h$?
- m gates $g_1, \ldots, g_m$ (wlog. all NAND)
- Up to $2m + r$ wires (if fan-in ≤ 2)
- Each wire has a definite 0-1 value.
- Bits have no common identity across wires, but we can create a universal layout for Boolean circuits in which they *do* retain identity...

Turing "Cue Bits"



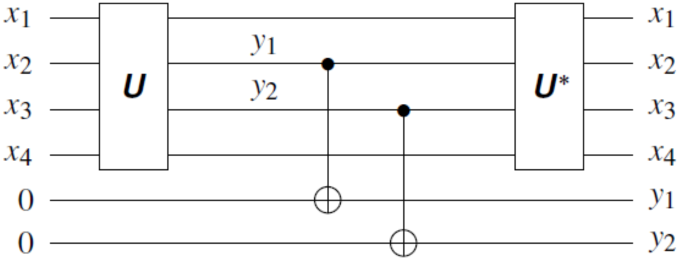
Space s, so $n - s$ "ancillary" cells.

Turing "Cue Bits"

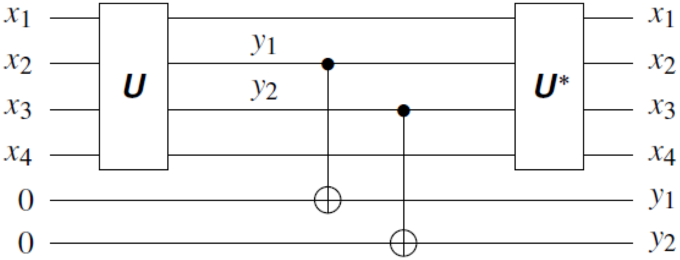


Space s, so $n - s$ "ancillary" cells. Can also be made *reversible*.

Quantum Circuits: similar picture, 90° flipped.



Quantum Circuits: similar picture, 90° flipped.



Must be reversible. In certain circumstances values $y_1, y_2, \ldots$ can be copied to extra lines as "$f(x)$." Then reversing the gates re-creates the input x so the whole mapping is invertible ("copy-uncompute trick").

Quantum Circuits Have. . .

Quantum Circuits Have...

- n input **qubits** $x_1, \ldots, x_n \in \{0, 1\}^n$

Quantum Circuits Have...

- n input **qubits** $x_1, \ldots, x_n \in \{0,1\}^n$ **(What's a qubit?)**

Quantum Circuits Have...

- n input **qubits** $x_1, \ldots, x_n \in \{0,1\}^n$ **(What's a qubit?)**
- $r \geq 1$ output qubits $z_1, \ldots, z_r$ (think $r = 1$ or $r = n$)

Quantum Circuits Have...

- n input **qubits** $x_1, \ldots, x_n \in \{0,1\}^n$ **(What's a qubit?)**
- $r \geq 1$ output qubits $z_1, \ldots, z_r$ (think $r = 1$ or $r = n$)
- $s - n$ **ancilla** qubits

Quantum Circuits Have. . .

- n input **qubits** $x_1, \ldots, x_n \in \{0, 1\}^n$ **(What's a qubit?)**
- $r \geq 1$ output qubits $z_1, \ldots, z_r$ (think $r = 1$ or $r = n$)
- $s - n$ **ancilla** qubits
- m-many **quantum gates** (arities can be 1,2,3)

Quantum Circuits Have. . .

- n input **qubits** $x_1, \ldots, x_n \in \{0,1\}^n$ **(What's a qubit?)**
- $r \geq 1$ output qubits $z_1, \ldots, z_r$ (think $r = 1$ or $r = n$)
- $s - n$ **ancilla** qubits
- m-many **quantum gates** (arities can be 1,2,3)
- Maybe h of them are **Hadamard gates**, which supply nondeterminism.

Quantum Circuits Have...

- n input **qubits** $x_1, \ldots, x_n \in \{0,1\}^n$ **(What's a qubit?)**
- $r \geq 1$ output qubits $z_1, \ldots, z_r$ (think $r = 1$ or $r = n$)
- $s - n$ **ancilla** qubits
- m-many **quantum gates** (arities can be 1,2,3)
- Maybe h of them are **Hadamard gates**, which supply nondeterminism.
- Qubits retain identity as wires transit gates.

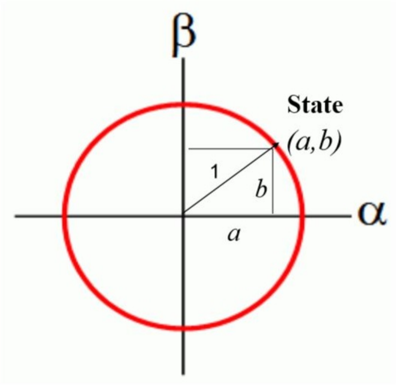
Quantum Circuits Have. . .

- n input **qubits** $x_1, \ldots, x_n \in \{0,1\}^n$ **(What's a qubit?)**
- $r \geq 1$ output qubits $z_1, \ldots, z_r$ (think $r = 1$ or $r = n$)
- $s - n$ **ancilla** qubits
- m-many **quantum gates** (arities can be 1,2,3)
- Maybe h of them are **Hadamard gates**, which supply nondeterminism.
- Qubits retain identity as wires transit gates.
- Each wire need **not** have a definite 0-1 value, owing to **entanglement**.

Quantum Circuits Have. . .

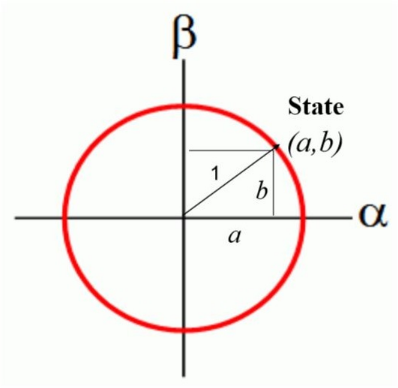
- n input **qubits** $x_1, \ldots, x_n \in \{0,1\}^n$ **(What's a qubit?)**
- $r \geq 1$ output qubits $z_1, \ldots, z_r$ (think $r = 1$ or $r = n$)
- $s - n$ **ancilla** qubits
- m-many **quantum gates** (arities can be 1,2,3)
- Maybe h of them are **Hadamard gates**, which supply nondeterminism.
- Qubits retain identity as wires transit gates.
- Each wire need **not** have a definite 0-1 value, owing to **entanglement**.
- Under the hood are $S = 2^s$ complex entries of a unit **state vector**.

A Qubit



Probability of observing Alpha is a-squared, Beta is b-squared. By Pythagoras, these add to 1.

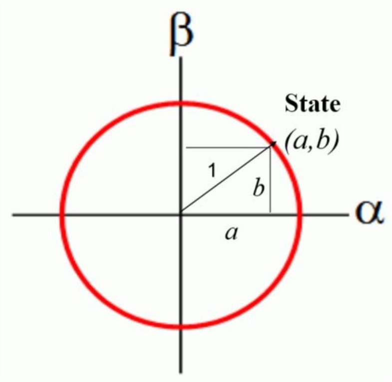
A Qubit



Probability of observing Alpha is a-squared, Beta is b-squared. By Pythagoras, these add to 1.

- A *qubit* is a physical entity that combines as described by a 2-vector $(a, b) = ae_0 + be_1$ over $\mathbb{C}$ and yields two *observations*: e_0 with probability $|a|^2$ or e_1 with probability $|b|^2$.

A Qubit



Probability of observing Alpha is a-squared, Beta is b-squared. By Pythagoras, these add to 1.

- A *qubit* is a physical entity that combines as described by a 2-vector $(a, b) = ae_0 + be_1$ over $\mathbb{C}$ and yields two *observations*: e_0 with probability $|a|^2$ or e_1 with probability $|b|^2$.
- A *qutrit* yields 3 results and is (a, b, c) where $|a|^2 + |b|^2 + |c|^2 = 1$.

Quantum Gates

Quantum Gates

- A k-ary gate can be represented by a $K \times K$ **unitary** matrix, $K = 2^k$.

Quantum Gates

- A k-ary gate can be represented by a $K \times K$ **unitary** matrix, $K = 2^k$.
- Common gates for $k = 1$, $K = 2$:

Quantum Gates

- A k-ary gate can be represented by a $K \times K$ **unitary** matrix, $K = 2^k$.
- Common gates for $k = 1$, $K = 2$:
- $\mathsf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ identity

Quantum Gates

- A k-ary gate can be represented by a $K \times K$ **unitary** matrix, $K = 2^k$.
- Common gates for $k = 1$, $K = 2$:
- $\mathsf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ identity
- $\mathsf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ negation, aka. NOT

Quantum Gates

- A k-ary gate can be represented by a $K \times K$ **unitary** matrix, $K = 2^k$.
- Common gates for $k = 1$, $K = 2$:
- $\mathsf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ identity
- $\mathsf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ negation, aka. NOT
- $\mathsf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ Hadamard gate.

Quantum Gates

- A k-ary gate can be represented by a $K \times K$ **unitary** matrix, $K = 2^k$.
- Common gates for $k = 1$, $K = 2$:
- $\mathsf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ identity
- $\mathsf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ negation, aka. NOT
- $\mathsf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ Hadamard gate.
- Only non-permutation gate needed for universality.

Quantum Gates

- A k-ary gate can be represented by a $K \times K$ **unitary** matrix, $K = 2^k$.
- Common gates for $k = 1$, $K = 2$:
- $\mathsf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ identity
- $\mathsf{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ negation, aka. NOT
- $\mathsf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ Hadamard gate.
- Only non-permutation gate needed for universality.
- But also common: $\mathsf{Y} = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$, $\mathsf{Z} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$, $\mathsf{S} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$.

Binary Gates

With $k = 2$ qubits, $K = 4$. "Controlled Not" showing quantum coordinates:

$$\mathsf{CNOT} = \begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Binary Gates

With $k = 2$ qubits, $K = 4$. "Controlled Not" showing quantum coordinates:

$$\mathsf{CNOT} = \begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Permutation $(1\ 2\ 4\ 3)$,

Binary Gates

With $k = 2$ qubits, $K = 4$. "Controlled Not" showing quantum coordinates:

$$\mathsf{CNOT} = \begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Permutation $(1\ 2\ 4\ 3)$, swap $(3\ 4)$.

Binary Gates

With $k = 2$ qubits, $K = 4$. "Controlled Not" showing quantum coordinates:

$$\mathsf{CNOT} = \begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Permutation (1 2 4 3), swap (3 4). Also called CX.

Binary Gates

With $k = 2$ qubits, $K = 4$. "Controlled Not" showing quantum coordinates:

$$\mathsf{CNOT} = \begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Permutation (1 2 4 3), swap (3 4). Also called CX.

$$\mathsf{CNOT} \circ (\mathsf{H} \otimes \mathsf{I}) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}$$

Binary Gates

With $k = 2$ qubits, $K = 4$. "Controlled Not" showing quantum coordinates:

$$\mathsf{CNOT} = \begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Permutation (1 2 4 3), swap (3 4). Also called CX.

$$\mathsf{CNOT} \circ (\mathsf{H} \otimes \mathsf{I}) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}$$

Applied to $e_{00} = (1, 0, 0, 0)^T$ gives $\frac{1}{\sqrt{2}}(e_{00} + e_{11})$.

Binary Gates

With $k = 2$ qubits, $K = 4$. "Controlled Not" showing quantum coordinates:

$$\mathsf{CNOT} = \begin{array}{c|cccc|} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 0 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 1 & 0 \\ \hline \end{array}$$

Permutation $(1\ 2\ 4\ 3)$, swap $(3\ 4)$. Also called CX.

$$\mathsf{CNOT} \circ (\mathsf{H} \otimes \mathsf{I}) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \end{bmatrix}$$

Applied to $e_{00} = (1, 0, 0, 0)^T$ gives $\frac{1}{\sqrt{2}}(e_{00} + e_{11})$. **EPR Entanglement.**

Ternary Toffoli Gate: $K = 8$

Ternary Toffoli Gate: $K = 8$

- $\mathsf{TOF} = diag(1,1,1,1,1,1)$, then $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

Ternary Toffoli Gate: $K = 8$

- $\mathsf{TOF} = diag(1,1,1,1,1,1)$, then $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.
- Fixes $000, \ldots, 101$; swaps $110 \leftrightarrow 111$.

Ternary Toffoli Gate: $K = 8$

- $\mathsf{TOF} = diag(1,1,1,1,1,1)$, then $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.
- Fixes $000, \ldots, 101$; swaps $110 \leftrightarrow 111$.
- Control-Control-NOT, hence also called CCX.

Ternary Toffoli Gate: $K = 8$

- $\mathsf{TOF} = diag(1,1,1,1,1,1)$, then $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.
- Fixes $000, \ldots, 101$; swaps $110 \leftrightarrow 111$.
- Control-Control-NOT, hence also called CCX.
- $\mathsf{TOF}(a, b, 1) = (-, -, a \text{ NAND } b)$, Thus TOF is classically universal.

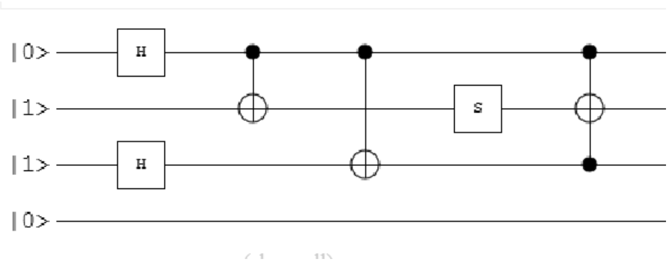
Ternary Toffoli Gate: $K = 8$

- $\mathsf{TOF} = diag(1,1,1,1,1,1)$, then $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.
- Fixes $000, \ldots, 101$; swaps $110 \leftrightarrow 111$.
- Control-Control-NOT, hence also called CCX.
- $\mathsf{TOF}(a,b,1) = (-,-,a \text{ NAND } b)$, Thus TOF is classically universal.
- $\mathsf{H} + \mathsf{TOF}$ is quantum universal.

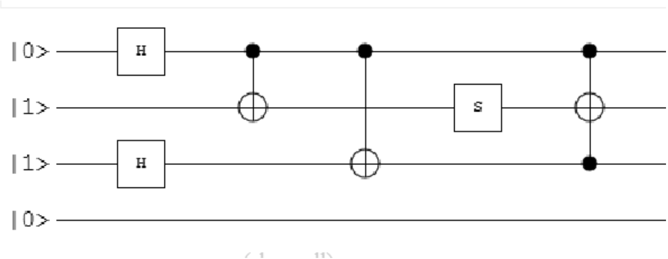
Ternary Toffoli Gate: $K = 8$

- $\mathsf{TOF} = diag(1,1,1,1,1,1)$, then $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.
- Fixes $000, \ldots, 101$; swaps $110 \leftrightarrow 111$.
- Control-Control-NOT, hence also called CCX.
- $\mathsf{TOF}(a, b, 1) = (-, -, a \text{ NAND } b)$, Thus TOF is classically universal.
- $\mathsf{H} + \mathsf{TOF}$ is quantum universal.
- $\mathsf{H} + \mathsf{CNOT}$ is not quantum universal; it recognizes a proper subclass of P.

Example Quantum Circuit

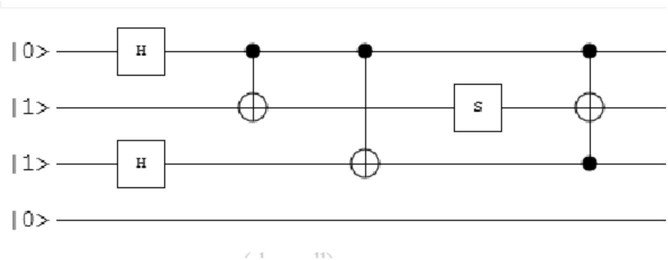


Example Quantum Circuit



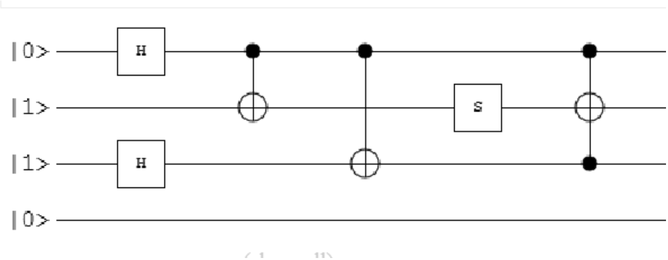
1. $\mathsf{H} \otimes \mathsf{I} \otimes \mathsf{H} \otimes \mathsf{I}^{\otimes(s-3)}$.

Example Quantum Circuit



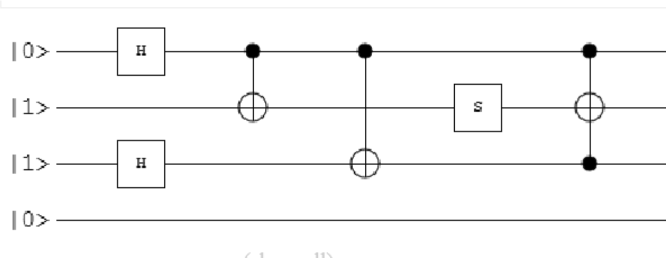
1. $\mathsf{H} \otimes \mathsf{I} \otimes \mathsf{H} \otimes \mathsf{I}^{\otimes(s-3)}$.
2. $\mathsf{CNOT} \otimes \mathsf{I}^{\otimes(s-2)}$. First three lines have "CXI."

Example Quantum Circuit



1. $\mathsf{H} \otimes \mathsf{I} \otimes \mathsf{H} \otimes \mathsf{I}^{\otimes(s-3)}$.
2. $\mathsf{CNOT} \otimes \mathsf{I}^{\otimes(s-2)}$. First three lines have "CXI."
3. "CIX"—semantically but not syntactically $\otimes$ of I and CNOT.

Example Quantum Circuit



1. $\mathsf{H} \otimes \mathsf{I} \otimes \mathsf{H} \otimes \mathsf{I}^{\otimes(s-3)}$.
2. $\mathsf{CNOT} \otimes \mathsf{I}^{\otimes(s-2)}$. First three lines have "CXI."
3. "CIX"—semantically but not syntactically $\otimes$ of I and CNOT.
4. After the S in stage 4, a TOF with controls on 1,3 and target on 2. The whole C computes a unitary U_C.

Input-Output and Measurement

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}}$

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}} = e_{x_1} \otimes e_{x_2} \otimes \cdots \otimes e_{x_n} \otimes e_0^{\otimes(s-n)}$.

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}} = e_{x_1} \otimes e_{x_2} \otimes \cdots \otimes e_{x_n} \otimes e_0^{\otimes(s-n)}$.
- Output: A state vector $(z_0, \ldots, z_{S-1})$, $S = 2^s$.

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}} = e_{x_1} \otimes e_{x_2} \otimes \cdots \otimes e_{x_n} \otimes e_0^{\otimes(s-n)}$.
- Output: A state vector $(z_0, \ldots, z_{S-1})$, $S = 2^s$.
- Measure all lines: For any outcome $b \in \{0,1\}^s$,
  $\Pr[C(x) \to b] = |z_b|^2 = |\langle E_x U_C e_b \rangle|^2$.

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}} = e_{x_1} \otimes e_{x_2} \otimes \cdots \otimes e_{x_n} \otimes e_0^{\otimes(s-n)}$.
- Output: A state vector $(z_0, \ldots, z_{S-1})$, $S = 2^s$.
- Measure all lines: For any outcome $b \in \{0,1\}^s$, $\Pr[C(x) \to b] = |z_b|^2 = |\langle E_x U_C e_b \rangle|^2$.
- (Show how DavyW applet does this.)

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}} = e_{x_1} \otimes e_{x_2} \otimes \cdots \otimes e_{x_n} \otimes e_0^{\otimes(s-n)}$.
- Output: A state vector $(z_0, \ldots, z_{S-1})$, $S = 2^s$.
- Measure all lines: For any outcome $b \in \{0,1\}^s$, $\Pr[C(x) \to b] = |z_b|^2 = |\langle E_x U_C e_b \rangle|^2$.
- (Show how DavyW applet does this.)
- For outcome $d \in \{0,1\}^r$ on r-many designated qubit lines, $\Pr[C(x) \to d] = \sum_{b \sqsupseteq d} |z_b|^2$.

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}} = e_{x_1} \otimes e_{x_2} \otimes \cdots \otimes e_{x_n} \otimes e_0^{\otimes(s-n)}$.
- Output: A state vector $(z_0, \ldots, z_{S-1})$, $S = 2^s$.
- Measure all lines: For any outcome $b \in \{0,1\}^s$, $\Pr[C(x) \to b] = |z_b|^2 = |\langle E_x U_C e_b \rangle|^2$.
- (Show how DavyW applet does this.)
- For outcome $d \in \{0,1\}^r$ on r-many designated qubit lines, $\Pr[C(x) \to d] = \sum_{b \sqsupseteq d} |z_b|^2$.
- Can project as amplitudes: $C(x) \mapsto (z'_0, \ldots, z'_{2^r-1})$ where $|z_d|^2$ is the probability of outcome $d \in \{0,1\}^r$.

Input-Output and Measurement

- Input: $E_x = e_{x0^{s-n}} = e_{x_1} \otimes e_{x_2} \otimes \cdots \otimes e_{x_n} \otimes e_0^{\otimes(s-n)}$.
- Output: A state vector $(z_0, \ldots, z_{S-1})$, $S = 2^s$.
- Measure all lines: For any outcome $b \in \{0,1\}^s$, $\Pr[C(x) \to b] = |z_b|^2 = |\langle E_x U_C e_b \rangle|^2$.
- (Show how DavyW applet does this.)
- For outcome $d \in \{0,1\}^r$ on r-many designated qubit lines, $\Pr[C(x) \to d] = \sum_{b \sqsupseteq d} |z_b|^2$.
- Can project as amplitudes: $C(x) \mapsto (z'_0, \ldots, z'_{2^r-1})$ where $|z_d|^2$ is the probability of outcome $d \in \{0,1\}^r$.
- Call this amplitude z_d as $A[C(x) \mapsto d]$.

BQP

Definition

A language L belongs to BQP if there are poly-time uniform quantum circuits C_n for each n such that forall n and inputs $x \in \{0,1\}^n$, designating qubit 1 for yes/no output:

$$\begin{aligned} x \in L &\quad\Longrightarrow\quad \Pr[C_n(x) \mapsto 1] > \frac{3}{4}, \\ x \notin L &\quad\Longrightarrow\quad \Pr[C_n(x) \mapsto 1] < \frac{1}{4}, \end{aligned}$$

BQP

Definition

A language L belongs to BQP if there are poly-time uniform quantum circuits C_n for each n such that forall n and inputs $x \in \{0,1\}^n$, designating qubit 1 for yes/no output:

$$\begin{aligned} x \in L &\implies \Pr[C_n(x) \mapsto 1] > \frac{3}{4}, \\ x \notin L &\implies \Pr[C_n(x) \mapsto 1] < \frac{1}{4}, \end{aligned}$$

- The language $L = \{(x, w) : w$ is an initial part of the unique prime factorization of $x\}$ captures the *task* of factoring x.

BQP

Definition

A language L belongs to BQP if there are poly-time uniform quantum circuits C_n for each n such that forall n and inputs $x \in \{0,1\}^n$, designating qubit 1 for yes/no output:

$$\begin{aligned} x \in L &\implies \Pr[C_n(x) \mapsto 1] > \frac{3}{4}, \\ x \notin L &\implies \Pr[C_n(x) \mapsto 1] < \frac{1}{4}, \end{aligned}$$

- The language $L = \{(x, w) : w$ is an initial part of the unique prime factorization of $x\}$ captures the *task* of factoring x.
- So the fact that $L \in$ BQP says that QCs can efficiently factor.

BQP

Definition

A language L belongs to BQP if there are poly-time uniform quantum circuits C_n for each n such that forall n and inputs $x \in \{0,1\}^n$, designating qubit 1 for yes/no output:

$$x \in L \implies \Pr[C_n(x) \mapsto 1] > \frac{3}{4},$$
$$x \notin L \implies \Pr[C_n(x) \mapsto 1] < \frac{1}{4},$$

- The language $L = \{(x, w) : w$ is an initial part of the unique prime factorization of $x\}$ captures the *task* of factoring x.
- So the fact that $L \in$ BQP says that QCs can efficiently factor.
- Goal: calculate the *amplitude* of $C_n(x) \mapsto 1$,

BQP

Definition

A language L belongs to BQP if there are poly-time uniform quantum circuits C_n for each n such that forall n and inputs $x \in \{0,1\}^n$, designating qubit 1 for yes/no output:

$$
\begin{aligned}
x \in L \quad &\implies \quad \Pr[C_n(x) \mapsto 1] > \frac{3}{4}, \\
x \notin L \quad &\implies \quad \Pr[C_n(x) \mapsto 1] < \frac{1}{4},
\end{aligned}
$$

- The language $L = \{(x, w) : w$ is an initial part of the unique prime factorization of $x\}$ captures the *task* of factoring x.
- So the fact that $L \in$ BQP says that QCs can efficiently factor.
- Goal: calculate the *amplitude* of $C_n(x) \mapsto 1$, explicitly or implicitly.

BQP

Definition

A language L belongs to BQP if there are poly-time uniform quantum circuits C_n for each n such that forall n and inputs $x \in \{0,1\}^n$, designating qubit 1 for yes/no output:

$$\begin{aligned} x \in L &\implies \Pr[C_n(x) \mapsto 1] > \frac{3}{4}, \\ x \notin L &\implies \Pr[C_n(x) \mapsto 1] < \frac{1}{4}, \end{aligned}$$

- The language $L = \{(x, w) : w$ is an initial part of the unique prime factorization of $x\}$ captures the *task* of factoring x.
- So the fact that $L \in$ BQP says that QCs can efficiently factor.
- Goal: calculate the *amplitude* of $C_n(x) \mapsto 1$, explicitly or implicitly.
- But no classical way known without paying $\approx 2^n$ time overhead as for factoring.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.
- Call that number e.g. $N_{P,x,z}[1]$.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.
- Call that number e.g. $N_{P,x,z}[1]$. Opposite: $N_{P,x,z}[0]$

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.
- Call that number e.g. $N_{P,x,z}[1]$. Opposite: $N_{P,x,z}[0]$
- For general P this number is NP-hard to compute. "No Free Lunch."

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.
- Call that number e.g. $N_{P,x,z}[1]$. Opposite: $N_{P,x,z}[0]$
- For general P this number is NP-hard to compute. "No Free Lunch." It is complete for the solution-counting class **#P**.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.
- Call that number e.g. $N_{P,x,z}[1]$. Opposite: $N_{P,x,z}[0]$
- For general P this number is NP-hard to compute. "No Free Lunch." It is complete for the solution-counting class **#P**.
- But in many cases, **#SAT solvers** may be effective.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.
- Call that number e.g. $N_{P,x,z}[1]$. Opposite: $N_{P,x,z}[0]$
- For general P this number is NP-hard to compute. "No Free Lunch." It is complete for the solution-counting class **#P**.
- But in many cases, **#SAT solvers** may be effective.
- Computation by solver more "offline" than managing 2^n-sized arrays.

Points of the Polynomial Simulation

- Existing general simulations pay $2^n m$ or $2^s m$ overhead right away.
- **Idea:** Paying only $O(s \cdot m)$ we can build a polynomial $P = P_C$ that encodes all info of C algebraically.
- The amplitude(s) we seek are calculable from the numbers of solutions to certain equations $P(y\,;\,\hat{x}) = \hat{z}$.
- Call that number e.g. $N_{P,x,z}[1]$. Opposite: $N_{P,x,z}[0]$
- For general P this number is NP-hard to compute. "No Free Lunch." It is complete for the solution-counting class **#P**.
- But in many cases, **#SAT solvers** may be effective.
- Computation by solver more "offline" than managing 2^n-sized arrays.
- P_C may yield other algebraic and physical info about C, including about entanglement.

The Basic Theorem

Theorem (Dawson et al.. 2004, implicitly before?)

Given C built from TOF *gates and h-many* H *gates, we can efficiently compute a polynomial $P_C(x_1, \ldots, x_n, y_1, \ldots, y_h, z_1, \ldots, z_r)$ and a constant R (here, $R = \sqrt{2^h}$) such that for all $x \in \{0,1\}^n$ and $z \in \{0,1\}^r$,*

$$A[C(x) \mapsto z] = \frac{1}{R}(N_{P,x,z}[1] - N_{P,x,z}[0]).$$

The Basic Theorem

Theorem (Dawson et al.. 2004, implicitly before?)

Given C built from TOF *gates and h-many* H *gates, we can efficiently compute a polynomial $P_C(x_1, \dots, x_n, y_1, \dots, y_h, z_1, \dots, z_r)$ and a constant R (here, $R = \sqrt{2^h}$) such that for all $x \in \{0,1\}^n$ and $z \in \{0,1\}^r$,*

$$A[C(x) \mapsto z] = \frac{1}{R}(N_{P,x,z}[1] - N_{P,x,z}[0]).$$

- Thus BQP reduces to the difference between two #P functions.

The Basic Theorem

Theorem (Dawson et al.. 2004, implicitly before?)

Given C built from TOF *gates and h-many* H *gates, we can efficiently compute a polynomial $P_C(x_1, \dots, x_n, y_1, \dots, y_h, z_1, \dots, z_r)$ and a constant R (here, $R = \sqrt{2^h}$) such that for all $x \in \{0,1\}^n$ and $z \in \{0,1\}^r$,*

$$A[C(x) \mapsto z] = \frac{1}{R}(N_{P,x,z}[1] - N_{P,x,z}[0]).$$

- Thus BQP reduces to the difference between two #P functions.
- Note heavy promise: $0 \leq N[1] - N[0] \leq R = \sqrt{2^h}$.

The Basic Theorem

Theorem (Dawson et al.. 2004, implicitly before?)

Given C built from TOF *gates and h-many* H *gates, we can efficiently compute a polynomial $P_C(x_1, \ldots, x_n, y_1, \ldots, y_h, z_1, \ldots, z_r)$ and a constant R (here, $R = \sqrt{2^h}$) such that for all $x \in \{0,1\}^n$ and $z \in \{0,1\}^r$,*

$$A[C(x) \mapsto z] = \frac{1}{R}(N_{P,x,z}[1] - N_{P,x,z}[0]).$$

- Thus BQP reduces to the difference between two #P functions.
- Note heavy promise: $0 \leq N[1] - N[0] \leq R = \sqrt{2^h}$.
- Means all but a trace of pairs $y, y' \in \{0,1\}^h$ cancel.

The Basic Theorem

Theorem (Dawson et al.. 2004, implicitly before?)

Given C built from TOF *gates and h-many* H *gates, we can efficiently compute a polynomial $P_C(x_1, \ldots, x_n, y_1, \ldots, y_h, z_1, \ldots, z_r)$ and a constant R (here, $R = \sqrt{2^h}$) such that for all $x \in \{0,1\}^n$ and $z \in \{0,1\}^r$,*

$$A[C(x) \mapsto z] = \frac{1}{R}(N_{P,x,z}[1] - N_{P,x,z}[0]).$$

- Thus BQP reduces to the difference between two #P functions.
- Note heavy promise: $0 \leq N[1] - N[0] \leq R = \sqrt{2^h}$.
- Means all but a trace of pairs $y, y' \in \{0,1\}^h$ cancel.
- Hence cannot approximate the difference by approximating each term. Need exact solution counting.

My Extensions

My Extensions

- Say a gate is *balanced* if all nonzero entries $re^{i\theta}$ of its matrix have equal magnitude $|r|$.

My Extensions

- Say a gate is *balanced* if all nonzero entries $re^{i\theta}$ of its matrix have equal magnitude $|r|$.
- A circuit C is balanced if every gate in C is balanced.

My Extensions

- Say a gate is *balanced* if all nonzero entries $re^{i\theta}$ of its matrix have equal magnitude $|r|$.
- A circuit C is balanced if every gate in C is balanced.
- $K(C) =$ the least K such that all θ in entries of gates in C are multiples of $2\pi/K$. "Min-Phase"

My Extensions

- Say a gate is *balanced* if all nonzero entries $re^{i\theta}$ of its matrix have equal magnitude $|r|$.
- A circuit C is balanced if every gate in C is balanced.
- $K(C)$ = the least K such that all θ in entries of gates in C are multiples of $2\pi/K$. "Min-Phase"
- Let G be a field or ring such that G^* embeds the K-th roots of unity ω^j by a multiplicative homomorphism $e(\omega^j)$.

Theorem

Can arrange $P_C = \prod_{gates\ g} P_g$ *such that for all* x *and* z*,*

$$A[C(x) \mapsto z] = \frac{1}{R} \sum_{j=0}^{K-1} \omega^j N_{P_C,x,z}[e(\omega^j)]$$

My Extensions

- Say a gate is *balanced* if all nonzero entries $re^{i\theta}$ of its matrix have equal magnitude $|r|$.
- A circuit C is balanced if every gate in C is balanced.
- $K(C) =$ the least K such that all θ in entries of gates in C are multiples of $2\pi/K$. "Min-Phase"
- Let G be a field or ring such that G^* embeds the K-th roots of unity ω^j by a multiplicative homomorphism $e(\omega^j)$.

Theorem

Can arrange $P_C = \prod_{gates\ g} P_g$ *such that for all* x *and* z,

$$A[C(x) \mapsto z] = \frac{1}{R} \sum_{j=0}^{K-1} \omega^j N_{P_C,x,z}[e(\omega^j)] = \frac{1}{R} \sum_{y} \omega^{P_C(x,y,z)}.$$

How it Works: The "Quantum Grille"

Consider a general 2×2 matrix A. Assign an indicator variable u to its input and y to its output:

	$(1-y)$	y
$(1-u)$	a_{11}	a_{12}
u	a_{21}	a_{22}

How it Works: The "Quantum Grille"

Consider a general 2×2 matrix A. Assign an indicator variable u to its input and y to its output:

	$(1-y)$	y
$(1-u)$	a_{11}	a_{12}
u	a_{21}	a_{22}

$$P_A = a_{11} + (a_{21} - a_{11})u + (a_{12} - a_{11})y + (a_{11} - a_{12} - a_{21} + a_{22})uy.$$

How it Works: The "Quantum Grille"

Consider a general 2×2 matrix A. Assign an indicator variable u to its input and y to its output:

$$\begin{array}{r|c|c|} & (1-y) & y \\ \hline (1-u) & a_{11} & a_{12} \\ u & a_{21} & a_{22} \end{array}$$

$$P_A = a_{11} + (a_{21} - a_{11})u + (a_{12} - a_{11})y + (a_{11} - a_{12} - a_{21} + a_{22})uy.$$

- Given $u, y \in \{0, 1\}$, only one path is allowed—others zeroed out.

How it Works: The "Quantum Grille"

Consider a general 2×2 matrix A. Assign an indicator variable u to its input and y to its output:

	$(1-y)$	y
$(1-u)$	a_{11}	a_{12}
u	a_{21}	a_{22}

$$P_A = a_{11} + (a_{21} - a_{11})u + (a_{12} - a_{11})y + (a_{11} - a_{12} - a_{21} + a_{22})uy.$$

- Given $u, y \in \{0, 1\}$, only one path is allowed—others zeroed out.
- Carries out Feynman's "Sum Over Paths" construction.

How it Works: The "Quantum Grille"

Consider a general 2×2 matrix A. Assign an indicator variable u to its input and y to its output:

	$(1-y)$	y
$(1-u)$	a_{11}	a_{12}
u	a_{21}	a_{22}

$$P_A = a_{11} + (a_{21} - a_{11})u + (a_{12} - a_{11})y + (a_{11} - a_{12} - a_{21} + a_{22})uy.$$

- Given $u, y \in \{0, 1\}$, only one path is allowed—others zeroed out.
- Carries out Feynman's "Sum Over Paths" construction.
- Works over any field or ring that embeds $0, 1, -1$.

How it Works: The "Quantum Grille"

Consider a general 2×2 matrix A. Assign an indicator variable u to its input and y to its output:

	$(1-y)$	y
$(1-u)$	a_{11}	a_{12}
u	a_{21}	a_{22}

$$P_A = a_{11} + (a_{21} - a_{11})u + (a_{12} - a_{11})y + (a_{11} - a_{12} - a_{21} + a_{22})uy.$$

- Given $u, y \in \{0, 1\}$, only one path is allowed—others zeroed out.
- Carries out Feynman's "Sum Over Paths" construction.
- Works over any field or ring that embeds $0, 1, -1$.
- If gate is deterministic, can *substitute* y by expression in u.

How it Works: The "Quantum Grille"

Consider a general 2×2 matrix A. Assign an indicator variable u to its input and y to its output:

	$(1-y)$	y
$(1-u)$	a_{11}	a_{12}
u	a_{21}	a_{22}

$$P_A = a_{11} + (a_{21} - a_{11})u + (a_{12} - a_{11})y + (a_{11} - a_{12} - a_{21} + a_{22})uy.$$

- Given $u, y \in \{0, 1\}$, only one path is allowed—others zeroed out.
- Carries out Feynman's "Sum Over Paths" construction.
- Works over any field or ring that embeds $0, 1, -1$.
- If gate is deterministic, can *substitute* y by expression in u.
- Every qubit at every stage has a well-defined *local* "algebraic value."

How it Works: Two-Qubit Gate

Now let A be a general 4×4 matrix. Assign indicator variables u_1, u_2 to the two incoming qubits and y_1, y_2 to their outgoing selves:

	$(1-y_1)(1-y_2)$	$(1-y_1)y_2$	$y_1(1-y_2)$	$y_1 y_2$
$(1-u_1)(1-u_2)$	a_{11}	a_{12}	a_{13}	a_{14}
$(1-u_1)u_2$	a_{21}	a_{22}	a_{23}	a_{24}
$u_1(1-u_2)$	a_{31}	a_{32}	a_{33}	a_{34}
$u_1 u_2$	a_{41}	a_{42}	a_{43}	a_{44}

How it Works: Two-Qubit Gate

Now let A be a general 4×4 matrix. Assign indicator variables u_1, u_2 to the two incoming qubits and y_1, y_2 to their outgoing selves:

	$(1-y_1)(1-y_2)$	$(1-y_1)y_2$	$y_1(1-y_2)$	$y_1 y_2$
$(1-u_1)(1-u_2)$	a_{11}	a_{12}	a_{13}	a_{14}
$(1-u_1)u_2$	a_{21}	a_{22}	a_{23}	a_{24}
$u_1(1-u_2)$	a_{31}	a_{32}	a_{33}	a_{34}
$u_1 u_2$	a_{41}	a_{42}	a_{43}	a_{44}

- Similar for 8×8 etc. Initially there are a lot of terms.

How it Works: Two-Qubit Gate

Now let A be a general 4×4 matrix. Assign indicator variables u_1, u_2 to the two incoming qubits and y_1, y_2 to their outgoing selves:

	$(1-y_1)(1-y_2)$	$(1-y_1)y_2$	$y_1(1-y_2)$	$y_1 y_2$
$(1-u_1)(1-u_2)$	a_{11}	a_{12}	a_{13}	a_{14}
$(1-u_1)u_2$	a_{21}	a_{22}	a_{23}	a_{24}
$u_1(1-u_2)$	a_{31}	a_{32}	a_{33}	a_{34}
$u_1 u_2$	a_{41}	a_{42}	a_{43}	a_{44}

- Similar for 8×8 etc. Initially there are a lot of terms.
- However, with substitution for permutation gates, the entire polynomial collapses to the constant 1

How it Works: Two-Qubit Gate

Now let A be a general 4×4 matrix. Assign indicator variables u_1, u_2 to the two incoming qubits and y_1, y_2 to their outgoing selves:

	$(1-y_1)(1-y_2)$	$(1-y_1)y_2$	$y_1(1-y_2)$	$y_1 y_2$
$(1-u_1)(1-u_2)$	a_{11}	a_{12}	a_{13}	a_{14}
$(1-u_1)u_2$	a_{21}	a_{22}	a_{23}	a_{24}
$u_1(1-u_2)$	a_{31}	a_{32}	a_{33}	a_{34}
$u_1 u_2$	a_{41}	a_{42}	a_{43}	a_{44}

- Similar for 8×8 etc. Initially there are a lot of terms.
- However, with substitution for permutation gates, the entire polynomial collapses to the constant 1!

How it Works: Two-Qubit Gate

Now let A be a general 4×4 matrix. Assign indicator variables u_1, u_2 to the two incoming qubits and y_1, y_2 to their outgoing selves:

	$(1-y_1)(1-y_2)$	$(1-y_1)y_2$	$y_1(1-y_2)$	$y_1 y_2$
$(1-u_1)(1-u_2)$	a_{11}	a_{12}	a_{13}	a_{14}
$(1-u_1)u_2$	a_{21}	a_{22}	a_{23}	a_{24}
$u_1(1-u_2)$	a_{31}	a_{32}	a_{33}	a_{34}
$u_1 u_2$	a_{41}	a_{42}	a_{43}	a_{44}

- Similar for 8×8 etc. Initially there are a lot of terms.
- However, with substitution for permutation gates, the entire polynomial collapses to the constant 1!
- The effect on P_C is then how the substituted terms are input to subsequent Hadamard and other kinds of gates.

How it Works: Qutrit Case

Let qutrit values be $0, 1, -1$ indexed by the basis vectors $e_0 = (1, 0, 0)^T$, $e_1 = (0, 1, 0)^T$, and $e_2 = (0, 0, 1)^T$.

How it Works: Qutrit Case

Let qutrit values be $0, 1, -1$ indexed by the basis vectors $e_0 = (1, 0, 0)^T$, $e_1 = (0, 1, 0)^T$, and $e_2 = (0, 0, 1)^T$. We need indicator polynomials in u and y that have the same nonzero value (here, 2) only when u, y have the corresponding values.

How it Works: Qutrit Case

Let qutrit values be $0, 1, -1$ indexed by the basis vectors $e_0 = (1, 0, 0)^T$, $e_1 = (0, 1, 0)^T$, and $e_2 = (0, 0, 1)^T$. We need indicator polynomials in u and y that have the same nonzero value (here, 2) only when u, y have the corresponding values. Given a general 3×3 matrix A, it goes:

	$2 - 2y^2$	$y^2 + y$	$y^2 - y$
$2 - 2u^2$	a_{11}	a_{12}	a_{13}
$u^2 + u$	a_{21}	a_{22}	a_{23}
$u^2 - u$	a_{31}	a_{32}	a_{33}

Now P_A has 36 terms.

How it Works: Qutrit Case

Let qutrit values be $0, 1, -1$ indexed by the basis vectors $e_0 = (1, 0, 0)^T$, $e_1 = (0, 1, 0)^T$, and $e_2 = (0, 0, 1)^T$. We need indicator polynomials in u and y that have the same nonzero value (here, 2) only when u, y have the corresponding values. Given a general 3×3 matrix A, it goes:

	$2-2y^2$	y^2+y	y^2-y
$2-2u^2$	a_{11}	a_{12}	a_{13}
u^2+u	a_{21}	a_{22}	a_{23}
u^2-u	a_{31}	a_{32}	a_{33}

Now P_A has 36 terms. But it simplifies for certain matrices:

$$A = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 & 1 & 1 \\ 1 & \omega & \omega^2 \\ 1 & \omega^2 & \omega \end{bmatrix}; \qquad P_A = 4 + 2\sqrt{3}iuy + 2u^2y^2.$$

Here $\omega = \frac{1}{2}(-1 + \sqrt{3}i)$, and the "$\mathsf{H}_3$ multiplier" is $2/\sqrt{3}$.

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g,

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi i j/K})$.

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi ij/K})$.
- However, degree is high: $\Theta(s)$.

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi ij/K})$.
- However, degree is high: $\Theta(s)$. We would prefer to *add* terms P_g and work e.g. over $\mathbb{Z}_K$.

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi ij/K})$.
- However, degree is high: $\Theta(s)$. We would prefer to *add* terms P_g and work e.g. over $\mathbb{Z}_K$.
- But where can the zero-values for invalid paths go?

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi ij/K})$.
- However, degree is high: $\Theta(s)$. We would prefer to *add* terms P_g and work e.g. over $\mathbb{Z}_K$.
- But where can the zero-values for invalid paths go?
- **My trick:** Allocate new "validity indicator" variables $w, \ldots$

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi ij/K})$.
- However, degree is high: $\Theta(s)$. We would prefer to *add* terms P_g and work e.g. over $\mathbb{Z}_K$.
- But where can the zero-values for invalid paths go?
- **My trick:** Allocate new "validity indicator" variables $w, \dots$
- Given a **constraint** c with values $0 = \text{OK}$, $1 = \textit{fail}$, add

$$q_c = w_0 c + 2w_1 c + 4w_2 c + \cdots + 2^{k-1} w_{k-1} c.$$

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi ij/K})$.
- However, degree is high: $\Theta(s)$. We would prefer to *add* terms P_g and work e.g. over $\mathbb{Z}_K$.
- But where can the zero-values for invalid paths go?
- **My trick:** Allocate new "validity indicator" variables $w, \dots$
- Given a **constraint** c with values $0 = \text{OK}$, $1 = \textit{fail}$, add

$$q_c = w_0 c + 2w_1 c + 4w_2 c + \cdots + 2^{k-1} w_{k-1} c.$$

- Then $c = 1 \implies$ binary assignments to $w_0, \dots, w_{k-1}$ run through all K values $\implies$ the entire net contribution over $\vec{u}, \vec{y}, \vec{w}$ **cancels**.

Multiplicative and Additive Cases

- P_C is simply the product of P_g over all gates g, possibly after substitutions for input and output variables and in-between.
- For min-phase K, works over any ring that embeds $(0, e^{2\pi ij/K})$.
- However, degree is high: $\Theta(s)$. We would prefer to *add* terms P_g and work e.g. over $\mathbb{Z}_K$.
- But where can the zero-values for invalid paths go?
- **My trick:** Allocate new "validity indicator" variables $w, \dots$
- Given a **constraint** c with values $0 = \text{OK}$, $1 = \textit{fail}$, add

$$q_c = w_0c + 2w_1c + 4w_2c + \cdots + 2^{k-1}w_{k-1}c.$$

- Then $c = 1 \implies$ binary assignments to $w_0, \dots, w_{k-1}$ run through all K values $\implies$ the entire net contribution over $\vec{u}, \vec{y}, \vec{w}$ **cancels**.
- Whereas $c = 0$ zeroes all such terms, so he only effect is to inflate R.

Additive Extension

Theorem

Given any C of minphase K, we can efficiently compute a polynomial $Q_C(x_1, \ldots, x_n, y_1, \ldots, y_h, z_1, \ldots, z_r, w_1, \ldots, w_t)$ over $\mathbb{Z}_K$ and a constant R such that for all $x \in \{0,1\}^n$ and $z \in \{0,1\}^r$,

$$A[C(x) \mapsto z] = \frac{1}{R} \sum_{j=0}^{K-1} \omega^j N_{Q_C,x,z}[j]$$

Additive Extension

Theorem

Given any C of minphase K, we can efficiently compute a polynomial $Q_C(x_1,\ldots,x_n,y_1,\ldots,y_h,z_1,\ldots,z_r,w_1,\ldots,w_t)$ over $\mathbb{Z}_K$ and a constant R such that for all $x \in \{0,1\}^n$ and $z \in \{0,1\}^r$,

$$A[C(x) \mapsto z] = \frac{1}{R}\sum_{j=0}^{K-1} \omega^j N_{Q_C,x,z}[j] = \frac{1}{R}\sum_{y,w} \omega^{Q_C(x,y,z,w)},$$

where $Q_C = \sum_{gates\ g} q_g + \sum_{constraints\ c} q_c$ has bounded degree.

Thus we can do all calculations using Q_C over $\mathbb{Z}_K$.

Computing the Polynomials

- “Annotate” every juncture of qubit i with variable y_j or term u_i.

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.
- u_i—H—: new variable y_j,

$$\begin{aligned} P_C &\;*= & (1 - u_i y_j) \\ Q_C &\;+= & 2^{k-1} u_i y_j. \end{aligned}$$

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.
- u_i—H—: new variable y_j,

$$\begin{aligned} P_C \quad &*= \quad (1 - u_i y_j) \\ Q_C \quad &+= \quad 2^{k-1} u_i y_j. \end{aligned}$$

- CNOT with incoming terms u_i on control, u_j on target: u_i stays, $u_j := 2u_i u_j - u_i - u_j$.

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.
- u_i—H—: new variable y_j,

$$\begin{aligned} P_C &\ *= & (1 - u_i y_j) \\ Q_C &\ += & 2^{k-1} u_i y_j. \end{aligned}$$

- CNOT with incoming terms u_i on control, u_j on target: u_i stays, $u_j := 2u_i u_j - u_i - u_j$.
- No change to P_C or Q_C, as with any permutation gate.

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.
- u_i—H—: new variable y_j,

$$\begin{aligned} P_C &\;*= & (1 - u_i y_j) \\ Q_C &\;+= & 2^{k-1} u_i y_j. \end{aligned}$$

- CNOT with incoming terms u_i on control, u_j on target: u_i stays, $u_j := 2u_i u_j - u_i - u_j$.
- No change to P_C or Q_C, as with any permutation gate.
- In characteristic 2, linearity is preserved.

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.
- u_i—H—: new variable y_j,

$$\begin{aligned} P_C \quad &*= \quad (1 - u_i y_j) \\ Q_C \quad &+= \quad 2^{k-1} u_i y_j. \end{aligned}$$

- CNOT with incoming terms u_i on control, u_j on target: u_i stays, $u_j := 2u_i u_j - u_i - u_j$.
- No change to P_C or Q_C, as with any permutation gate.
- In characteristic 2, linearity is preserved.
- TOF: controls u_i, u_j stay, target u_k changes to $2u_i u_j u_k - u_i u_j - u_k$.

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.
- u_i—H—: new variable y_j,

$$\begin{aligned} P_C &\ *= & (1 - u_i y_j) \\ Q_C &\ += & 2^{k-1} u_i y_j. \end{aligned}$$

- CNOT with incoming terms u_i on control, u_j on target: u_i stays, $u_j := 2u_iu_j - u_i - u_j$.
- No change to P_C or Q_C, as with any permutation gate.
- In characteristic 2, linearity is preserved.
- TOF: controls u_i, u_j stay, target u_k changes to $2u_iu_ju_k - u_iu_j - u_k$.
- Linearity not preserved.

Computing the Polynomials

- "Annotate" every juncture of qubit i with variable y_j or term u_i.
- Initially x_i and 0 terms, $P_C = 1$, $Q_C = 0$.
- u_i—H—: new variable y_j,

$$\begin{aligned} P_C &\ast= (1 - u_i y_j) \\ Q_C &+= 2^{k-1} u_i y_j. \end{aligned}$$

- CNOT with incoming terms u_i on control, u_j on target: u_i stays, $u_j := 2u_iu_j - u_i - u_j$.
- No change to P_C or Q_C, as with any permutation gate.
- In characteristic 2, linearity is preserved.
- TOF: controls u_i, u_j stay, target u_k changes to $2u_iu_ju_k -$ u_iu_j $-u_k$.
- Linearity not preserved. Similar considerations in paper by Bacon-van Dam-Russell, 2008 (unpub., morphed into "least action" talk. . .).

Equality Constraints

To enforce a desired output value z_i on qubit i with final term u_i:

$$\begin{aligned} P_C &\mathrel{*}= (1 + 2u_i z_i - u_i - z_i) \\ Q_C &\mathrel{+}= w_j(u_i + z_i - 2u_i z_i). \end{aligned}$$

Equality Constraints

To enforce a desired output value z_i on qubit i with final term u_i:

$$
\begin{aligned}
P_C &\ast= (1 + 2u_i z_i - u_i - z_i) \\
Q_C &+= w_j(u_i + z_i - 2u_i z_i).
\end{aligned}
$$

In characteristic 2, Q_C remains quadratic.

Equality Constraints

To enforce a desired output value z_i on qubit i with final term u_i:

$$\begin{aligned} P_C &\ast= (1 + 2u_i z_i - u_i - z_i) \\ Q_C &+= w_j(u_i + z_i - 2u_i z_i). \end{aligned}$$

In characteristic 2, Q_C remains quadratic.

Theorem (Cai-Chen-Lipton-Lu 2010, after Grigoriev-Karpinski (et al.))

For quadratic $p(x_1, \ldots, x_n)$ over $\mathbb{Z}_K$, and all $a < K$, $N_p[a]$ is computable in $\mathsf{poly}(nK)$ time.

Equality Constraints

To enforce a desired output value z_i on qubit i with final term u_i:

$$\begin{aligned} P_C &\mathrel{*}= (1 + 2u_i z_i - u_i - z_i) \\ Q_C &\mathrel{+}= w_j(u_i + z_i - 2u_i z_i). \end{aligned}$$

In characteristic 2, Q_C remains quadratic.

Theorem (Cai-Chen-Lipton-Lu 2010, after Grigoriev-Karpinski (et al.))

For quadratic $p(x_1, \dots, x_n)$ over $\mathbb{Z}_K$, and all $a < K$, $N_p[a]$ is computable in $\mathsf{poly}(nK)$ time.

Open: replace K by $\log K$ in the time? Affirmative for $A[C(x) \mapsto z]$.

Gottesman-Knill: alternative methodology

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.
- H gives $Q_C \mathrel{+}= 2u_i y_j$.

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.
- H gives $Q_C \mathrel{+}= 2u_i y_j$.
- CNOT: Nonlinear term has a 2 which will cancel the 2 from Hadamard.

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.
- H gives $Q_C \mathrel{+}= 2u_i y_j$.
- CNOT: Nonlinear term has a 2 which will cancel the 2 from Hadamard.
- Equality constraint $w_j(u_i + z_i - 2u_i z_i)$: OK with [G-K], [CCLL] because w_j appears only here.

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.
- H gives $Q_C \mathrel{+}= 2u_i y_j$.
- CNOT: Nonlinear term has a 2 which will cancel the 2 from Hadamard.
- Equality constraint $w_j(u_i + z_i - 2u_i z_i)$: OK with [G-K], [CCLL] because w_j appears only here.
- S: u_i left alone but $Q_C \mathrel{+}= u_i^2$.

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.
- H gives $Q_C \mathrel{+}= 2u_i y_j$.
- CNOT: Nonlinear term has a 2 which will cancel the 2 from Hadamard.
- Equality constraint $w_j(u_i + z_i - 2u_i z_i)$: OK with [G-K], [CCLL] because w_j appears only here.
- S: u_i left alone but $Q_C \mathrel{+}= u_i^2$.
- Inductively every term in Q_C has form y_j^2 or $2y_i y_j$.

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.
- H gives $Q_C \mathrel{+}= 2u_i y_j$.
- CNOT: Nonlinear term has a 2 which will cancel the 2 from Hadamard.
- Equality constraint $w_j(u_i + z_i - 2u_i z_i)$: OK with [G-K], [CCLL] because w_j appears only here.
- S: u_i left alone but $Q_C \mathrel{+}= u_i^2$.
- Inductively every term in Q_C has form y_j^2 or $2y_i y_j$.
- These terms are invariant under $0 \leftrightarrow 2$, $1 \leftrightarrow 3$.

Gottesman-Knill: alternative methodology

- To represent u_i—S— we need $K = 4$.
- H gives $Q_C \mathrel{+}= 2u_i y_j$.
- CNOT: Nonlinear term has a 2 which will cancel the 2 from Hadamard.
- Equality constraint $w_j(u_i + z_i - 2u_i z_i)$: OK with [G-K], [CCLL] because w_j appears only here.
- S: u_i left alone but $Q_C \mathrel{+}= u_i^2$.
- Inductively every term in Q_C has form y_j^2 or $2y_i y_j$.
- These terms are invariant under $0 \leftrightarrow 2$, $1 \leftrightarrow 3$.
- Hence [CCLL] gives poly-time simulation by soution counting in $\mathbb{Z}_4$.

Open Questions

Open Questions

- Solution counting is #P-complete for degree 3 over $\mathbb{Z}_K$ in general.

Open Questions

- Solution counting is #P-complete for degree 3 over $\mathbb{Z}_K$ in general.
- Are some "structured" subcases of degree 3 tractable? Can they come from families of QC's?

Open Questions

- Solution counting is #P-complete for degree 3 over $\mathbb{Z}_K$ in general.
- Are some "structured" subcases of degree 3 tractable? Can they come from families of QC's?
- What else is (physically!) meaningful about polynomials in the circuit's partition function?

Open Questions

- Solution counting is #P-complete for degree 3 over $\mathbb{Z}_K$ in general.
- Are some "structured" subcases of degree 3 tractable? Can they come from families of QC's?
- What else is (physically!) meaningful about polynomials in the circuit's partition function?
- Invariants based on Strassen's *geometric degree* $\gamma(f)$ concept, others?

Open Questions

- Solution counting is $\#\mathsf{P}$-complete for degree 3 over $\mathbb{Z}_K$ in general.
- Are some "structured" subcases of degree 3 tractable? Can they come from families of QC's?
- What else is (physically!) meaningful about polynomials in the circuit's partition function?
- Invariants based on Strassen's *geometric degree* $\gamma(f)$ concept, others?
- Baur-Strassen showed that $\log_2 \gamma(f)$ lower-bounds the arithmetical complexity of f, indeed the number of binary multiplication gates.

Open Questions

- Solution counting is #P-complete for degree 3 over $\mathbb{Z}_K$ in general.
- Are some "structured" subcases of degree 3 tractable? Can they come from families of QC's?
- What else is (physically!) meaningful about polynomials in the circuit's partition function?
- Invariants based on Strassen's *geometric degree* $\gamma(f)$ concept, others?
- Baur-Strassen showed that $\log_2 \gamma(f)$ lower-bounds the arithmetical complexity of f, indeed the number of binary multiplication gates.
- Relevance to complexity of quantum circuits $C\ldots$

Open Questions

- Solution counting is #P-complete for degree 3 over $\mathbb{Z}_K$ in general.
- Are some "structured" subcases of degree 3 tractable? Can they come from families of QC's?
- What else is (physically!) meaningful about polynomials in the circuit's partition function?
- Invariants based on Strassen's *geometric degree* $\gamma(f)$ concept, others?
- Baur-Strassen showed that $\log_2 \gamma(f)$ lower-bounds the arithmetical complexity of f, indeed the number of binary multiplication gates.
- Relevance to complexity of quantum circuits C...
- Possibly quantify the "entangling capacity" of C?

Open Questions

- Solution counting is #P-complete for degree 3 over $\mathbb{Z}_K$ in general.
- Are some "structured" subcases of degree 3 tractable? Can they come from families of QC's?
- What else is (physically!) meaningful about polynomials in the circuit's partition function?
- Invariants based on Strassen's *geometric degree* $\gamma(f)$ concept, others?
- Baur-Strassen showed that $\log_2 \gamma(f)$ lower-bounds the arithmetical complexity of f, indeed the number of binary multiplication gates.
- Relevance to complexity of quantum circuits C...
- Possibly quantify the "entangling capacity" of C?
- Delineate algebraic properties of qutrit circuits...