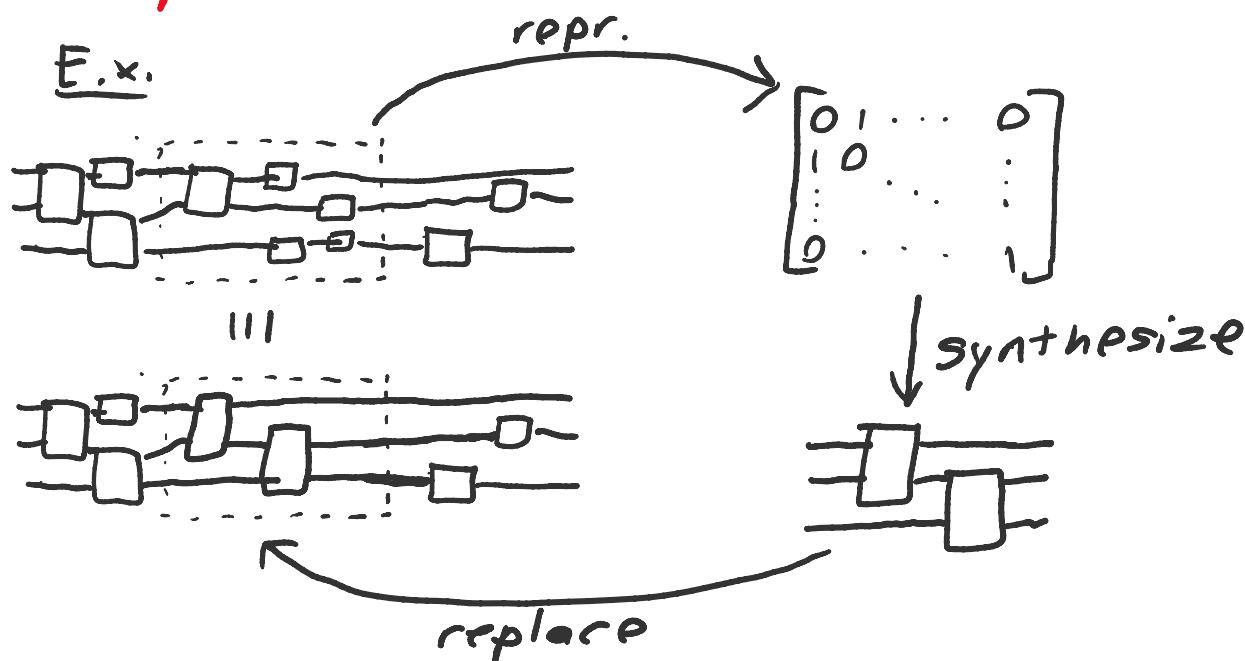


Re-Synthesis

Re-synthesis based optimizations take part of a (or a whole) circuit, and synthesize a new one from some representation of it.



In general, don't want to use **matrix** representations:

- ① Exponential in #qubits,
- ② Synthesis of arbitrary matrices is **highly non-optimal**

Effective re-synthesis methods usually involve some **efficiently computable** representation of a class of circuits, together with optimal or heuristic synthesis for that representation.

Ex.

The CNOT gate implements a **linear function** on \mathbb{Z}_2^2 .

$$\begin{aligned} \text{CNOT } 10 > 10 > &= 10 > 10 > &\text{matrix on } \mathbb{Z}_2^2 \\ \text{CNOT } 10 > 11 > &= 10 > 11 > \\ \text{CNOT } 11 > 10 > &= 11 > 11 > \\ \text{CNOT } 11 > 11 > &= 11 > 10 > \end{aligned}$$

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

Representation of CNOT circuits

Recall that the **general linear group** $GL(n, F)$ consists of the $n \times n$ invertible matrices over F

Prop.

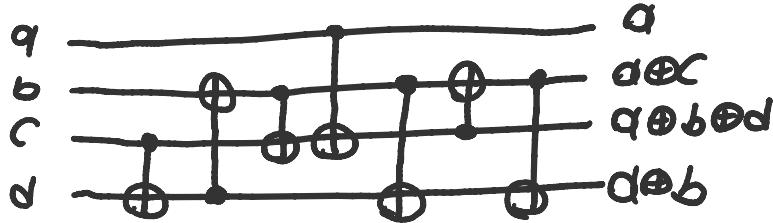
Let C be a circuit over $\{\text{CNOT}\}$. Then

$$[C] | \vec{x} \rangle = | A \vec{x} \rangle \quad \forall \vec{x} \in \mathbb{Z}_2^n$$

where $A \in GL(n, \mathbb{Z}_2)$

Ex.

Consider the CNOT circuit



The representation as a 4×4 invertible matrix A is

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

Note that

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} a \\ a \oplus c \\ a \oplus b \oplus d \\ a \oplus b \end{bmatrix}$$

(CNOT row operations)

Over \mathbb{Z}_2 , the only non-trivial row operations are

$$R_j \rightarrow R_j + R_i \quad \& \quad R_i \leftrightarrow R_j \quad (\text{add \& swap})$$

These correspond to

$$\text{CNOT}_{ij} = E_{ij} \quad \& \quad \text{SWAP}_{ij} = \text{CNOT}_{ij} \text{CNOT}_{ji} \quad \text{CNOT}_{ii} = T_{ii}$$

Synthesis of CNOT circuits

Prop.

Any matrix $A \in GL(n, \mathbb{Z}_2)$ can be synthesized ^{efficiently} as a circuit of at most $O(n^2)$ CNOT gates

Pf.

Use Gaussian Elimination to compute a sequence of $O(n^2)$ row ops $R_1 \cdots R_k A = I$. Then $A = R_k \cdots R_1$, where each row operation is 1 or 3 CNOT gates.

Ex.

Synthesize $U|\vec{x}\rangle = |A\vec{x}\rangle$ where $A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

$$\begin{array}{c}
 \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right] \xrightarrow{R_3 \rightarrow R_1 + R_3} \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right] \xrightarrow{R_1 \rightarrow R_1 + R_2} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right] \\
 \downarrow R_3 \rightarrow R_2 + R_3 \\
 \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \xleftarrow{R_4 \rightarrow R_2 + R_4} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]
 \end{array}$$

Hence

$$U = \begin{array}{c} \text{CNOT}(a, b) \\ \text{CNOT}(a, c) \\ \text{CNOT}(a, d) \\ \text{CNOT}(b, c) \\ \text{CNOT}(b, d) \\ \text{CNOT}(c, d) \end{array}$$

Note: a more efficient sequence exists:

$$\begin{array}{c}
 \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right] \xrightarrow{R_1 \rightarrow R_1 + R_2} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right] \xrightarrow{R_3 \rightarrow R_1 + R_3} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right] \xrightarrow{R_4 \rightarrow R_2 + R_4} \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right] \\
 \therefore U = \begin{array}{c} \text{CNOT}(a, b) \\ \text{CNOT}(a, c) \\ \text{CNOT}(a, d) \\ \text{CNOT}(b, c) \end{array}
 \end{array}$$

Optimal* CNOT-Synthesis

(Patel - Markov - Hayes, arXiv:quant-ph/0302002)

Patel, Markov & Hayes in 2003 devised an algorithm which produces circuits with $O(n^2/\log n)$ CNOT gates. Given that a simple counting argument shows that there exist $A \in GL(n, \mathbb{Z}_2)$ which require $\Omega(n^2/\log n)$ gates, this is **asymptotically optimal**. In practice though, performance leaves much to be desired...

PMH algorithm

1. Divide into groups of $k (\sim \log n)$ columns
2. For each column group
3. Add any rows which are identical on those columns
4. Perform regular Gaussian elimination on those columns

Other considerations

Often when re-synthesizing, the input state lies in some subspace $V \subseteq \mathbb{Z}_2^n$ due to ancillas. Knowing V gives **extra freedom** in synthesizing A . For example,

$$\begin{array}{c} q \\ b \\ c \\ d \\ \text{C}\oplus\text{d} \end{array} \xrightarrow{\text{CNOTs}} \begin{array}{c} \text{a}\oplus\text{c}\oplus\text{d} \\ \text{b}\oplus\text{c}\oplus\text{d} \\ \text{c} \\ \text{d} \\ \text{C}\oplus\text{d} \end{array}$$
$$\rightarrow A = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Synthesizing A gives 4 CNOT gates, but knowing that the 5th qubit starts in the $\text{C}\oplus\text{d}$ state gives a solution with 2 CNOT gates. In general, we want

$$A \in GL(n, \mathbb{Z}_2) \text{ s.t. } AA_0 = A,$$

For A_0, A , $n \times m$ matrices giving the **input** and **output**, resp. Using a **pseudoinverse** A_0^\dagger we can synthesize

$$A = A_0 A_0^\dagger \in GL(n, \mathbb{Z}_2)$$

Affine permutations

Another class of **permutations** which can be efficiently represented are **affine permutations**.

(General Affine group)

The general affine group $GA(n, \mathbb{Z}_2) \cong GL(n, \mathbb{Z}_2) \times \mathbb{Z}_2^n$ consists of the invertible affine transformations over \mathbb{Z}_2^n .

That is, for $A \in GL(n, \mathbb{Z}_2)$, $\vec{b} \in \mathbb{Z}_2^n$, then

$$(A, \vec{b}) \in GA(n, \mathbb{Z}_2)$$

$$(A, \vec{b}) \vec{x} = A\vec{x} + \vec{b} \quad \forall \vec{x} \in \mathbb{Z}_2^n$$

Fact

$\{\text{CNOT}, X\}$ generates $GA(n, \mathbb{Z}_2)$

To see why, it suffices to note that if

$$U|\vec{x}\rangle = |A\vec{x} + \vec{b}\rangle, \quad (A, \vec{b}) \in GA(n, \mathbb{Z}_2)$$

Then we can factorize as $U = U_x U_{\text{CNOT}}$ where

$$U_{\text{CNOT}}|\vec{x}\rangle = |A\vec{x}\rangle$$

$$U_x|\vec{x}\rangle = |\vec{x} + \vec{b}\rangle$$

Since $A \in GL(n, \mathbb{Z}_2)$, U_{CNOT} can be implemented using only CNOT gates. Moreover,

