Lecture 10 CPSC 447/547 - Intro to QC

Reversible Logic and Oracles.

Outline

- Quantum query model
- Types of oracles
- Reversible circuits, ancilla & garbage qubits.

How to compute classical Boolean functions on a quantum computer?

Query model

Classical:

$$x \;-\!\boxed{f}\!-\; f(x)$$

input oracle output

E.g. $f: \{0,1\} \to \{0,1\}$

$x \longrightarrow f(x) = \neg x$

"NOT function"

Quantum:

$$|x\rangle \;-\!\boxed{f}\!-\; |f(x)\rangle$$

$\downarrow$ superpositions

$$\underbrace{\sum_x \alpha_x |x\rangle} \;-\!\boxed{f}\!-\; \underbrace{\sum_x \alpha_x |f(x)\rangle}_{\text{state}}$$

E.g. Same NOT function

$\alpha|0\rangle + \beta|1\rangle \longrightarrow \alpha|f(0)\rangle + \beta|f(1)\rangle$

$= \alpha|1\rangle + \beta|0\rangle$

"X gate".

q. state q. circuit q. state

This is like a query to some memory where x is the address, and $f(x)$ is the data.

but the key here is we can:

"access memory"/"evaluate f" in superposition.

To accomplish this, $-\boxed{f}-$ must be a q. circuit. (reversible)

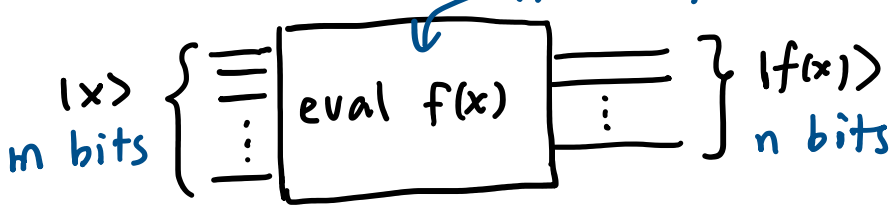
Recall from an earlier lecture:

Thm Any Boolean functions $f: \{0,1\}^m \to \{0,1\}^n$ can be computed by a reversible circuit of k bits input and k bits output, $k \geq m, n$.

Let's see how to do that for general f.

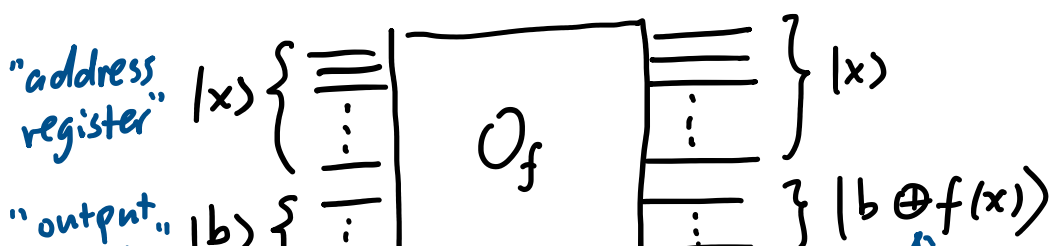
$$f: \{0,1\}^m \to \{0,1\}^n.$$

Problematic? $m \neq n$.



We can fix the mismatch of input/output:

"Bit oracle" (XOR oracle) O_f



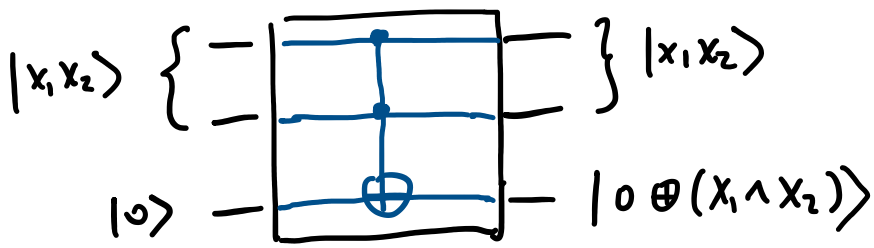
register  bitwise XOR.

Key: Storing $f(x)$ **out-of-place**.

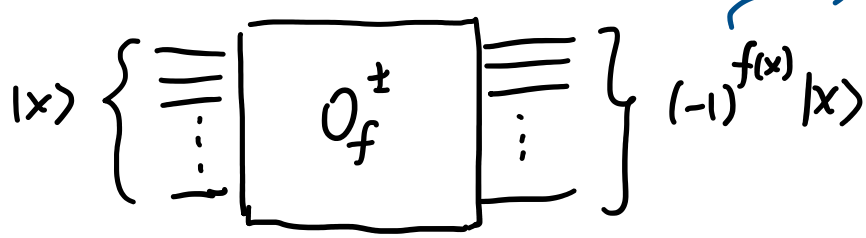
When $|b\rangle = |0\rangle$, then $|x\rangle|0\rangle \xrightarrow{O_f} |x\rangle|f(x)\rangle$

Remark: with superposition: $\sum_x \alpha_x |x\rangle|0\rangle \xrightarrow{O_f} \sum_x \alpha_x |x\rangle|f(x)\rangle$.

E.g. $f = \text{AND} : \{0,1\}^2 \to \{0,1\}$



"Phase oracle" $O_f^{\pm}$ (for $n=1$)

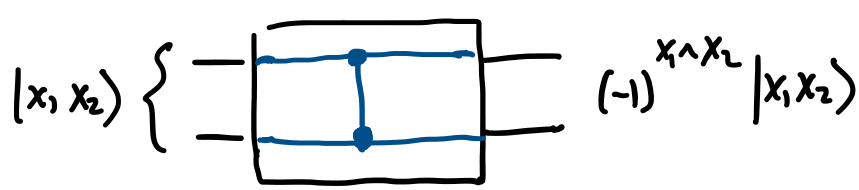


Is this global phase? Not if we put in superposition.

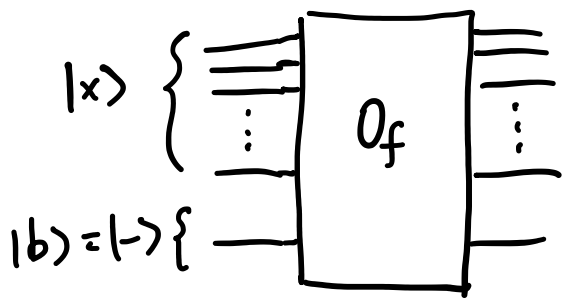
Key: Storing output **in the phase**.

Remark: Superposition: $\sum_x \alpha_x |x\rangle \xrightarrow{O_f^{\pm}} \sum_x \alpha_x (-1)^{f(x)} |x\rangle$.

E.g. Again for **AND**:



Bonus We can **use O_f to implement $O_f^{\pm}$**



Input :

$$|x\rangle|-\rangle = \frac{1}{\sqrt{2}}\left(|x\rangle|0\rangle - |x\rangle|1\rangle\right)$$

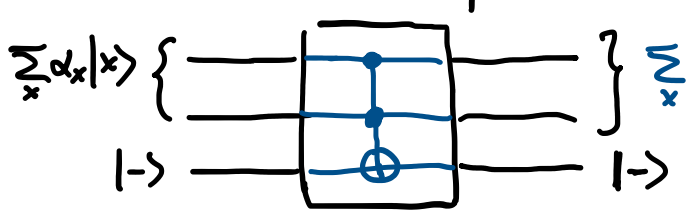
? Output :

$$\frac{1}{\sqrt{2}}\left(|x\rangle|f(x)\rangle - |x\rangle|1 \oplus f(x)\rangle\right)$$

$$\begin{cases} \text{If } f(x)=0: \\ \quad \frac{1}{\sqrt{2}}\left(|x\rangle|0\rangle - |x\rangle|1\rangle\right) = |x\rangle|-\rangle \\ \text{If } f(x)=1: \\ \quad \frac{1}{\sqrt{2}}\left(|x\rangle|1\rangle - |x\rangle|0\rangle\right) = (-1)|x\rangle|-\rangle \end{cases}$$

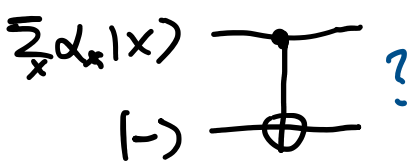
$$\left(\sum_x \alpha_x |x\rangle\right)|-\rangle \xrightarrow{O_f} \left(\sum_x \alpha_x (-1)^{f(x)} |x\rangle\right)|-\rangle$$

So in the AND example :



Seem to change the control qubits !

Phase kickback – a closer look .



$\sum_x \alpha_x (-1)^x |x\rangle$

$|-\rangle$.

$$(\alpha_0|0\rangle + \alpha_1|1\rangle) \otimes \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = \frac{\alpha_0}{\sqrt{2}}|00\rangle - \frac{\alpha_0}{\sqrt{2}}|01\rangle + \frac{\alpha_1}{\sqrt{2}}|10\rangle - \frac{\alpha_1}{\sqrt{2}}|11\rangle$$

$\downarrow$ CNOT

$$\frac{\alpha_0}{\sqrt{2}}|00\rangle - \frac{\alpha_0}{\sqrt{2}}|01\rangle + \frac{\alpha_1}{\sqrt{2}}|11\rangle - \frac{\alpha_1}{\sqrt{2}}|10\rangle$$

$$= \underbrace{(\alpha_0|0\rangle - \alpha_1|1\rangle)} \otimes \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right)$$

Control qubit got a phase!

Let's come back to this idea of using scratch space.

We need scratch space by counting argument.

$$\begin{cases} \#\text{ functions on } n\text{-bit input, } n\text{-bit output} = (2^n)^{2^n} \\ \#\text{ reversible functions on } n \text{ bits} = (2^n)! \end{cases}$$

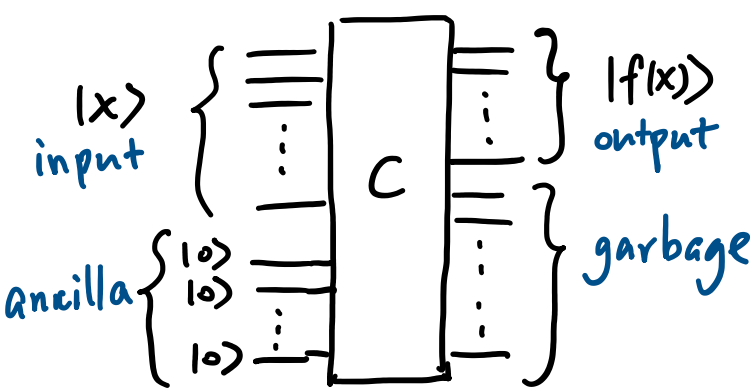
↳ can be viewed as permutations

- Reversible function is a small fraction.

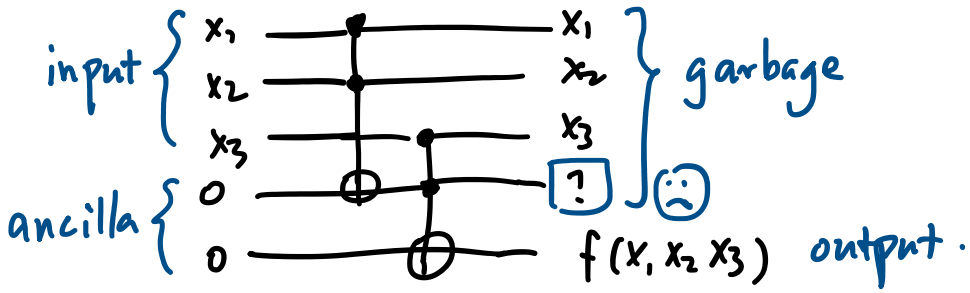
$$\frac{(2^n)!}{(2^n)^{2^n}} \approx \frac{(2^n/e)^{2^n}}{(2^n)^{2^n}} = e^{-2^n}$$

- Reversible function on $(n+1)$ bits $= (2^{n+1})! > (2^n)^{2^n}$

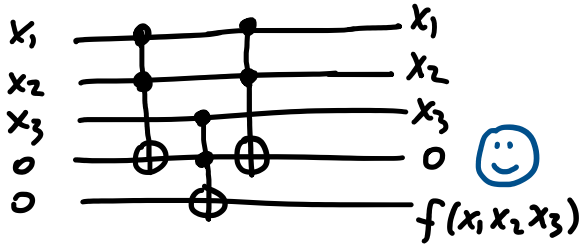
$$\begin{cases} \text{How much scratch space do we need?} \\ \text{How to save qubits by reusing ancilla?} \end{cases}$$



<u>Example</u> for $f = AND(x_1, x_2, x_3)$.

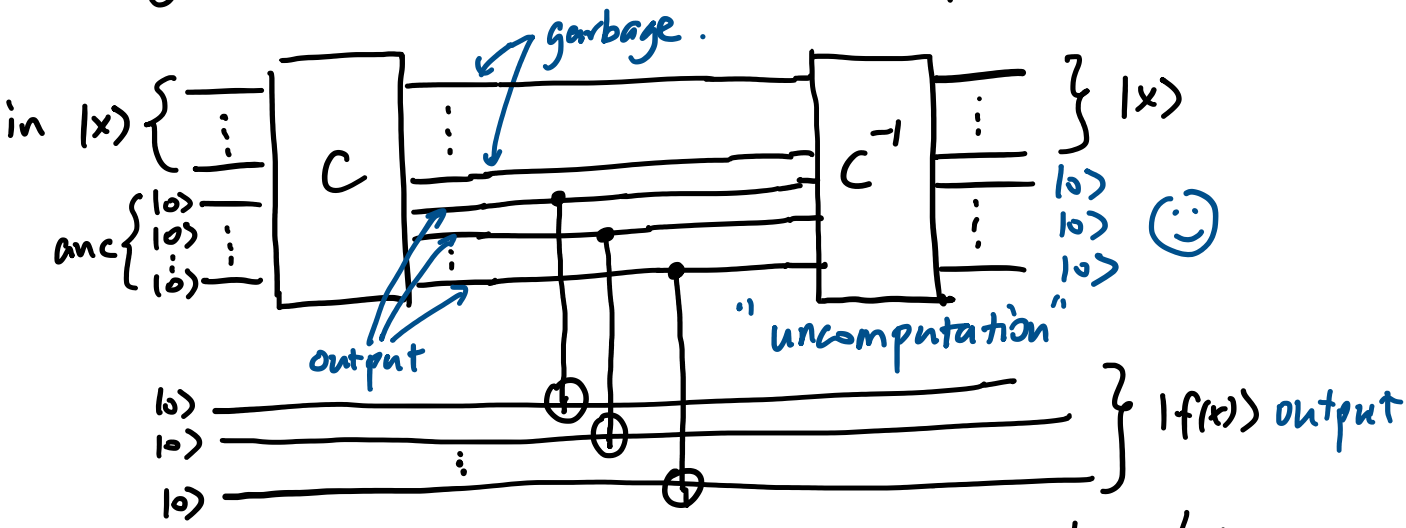


Fortunately, some of the garbage bits can be **cleaned up**.



We saw this trick in last lecture too.

In general, we can perform **uncomputation strategies**.



This seems to have **large overhead** though!

We need to run computation C forward and backward.

which means roughly <u>twice</u> as long circuit!

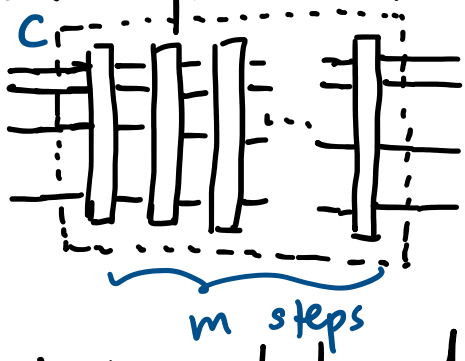
We can do much better!

<u>Divide-and-Conquer</u> (reversible pebble game) <u>Optional</u>

Idea: Clean up garbage frequently by uncomputing parts of C.

Outline

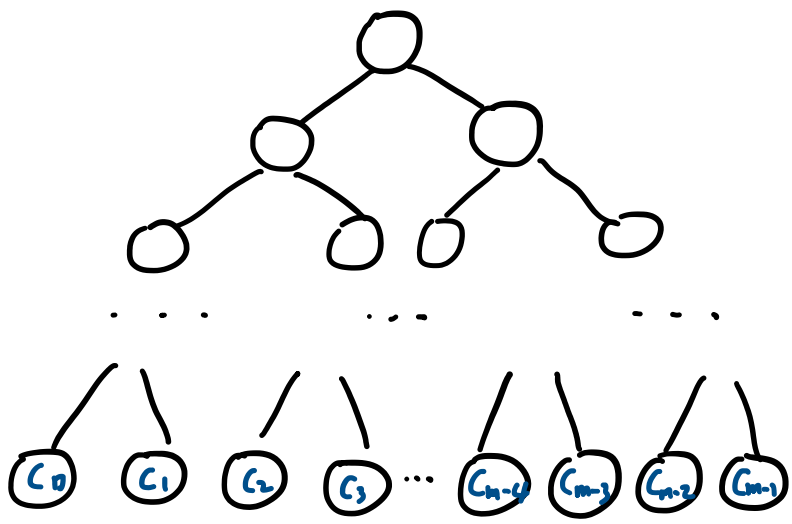
1. Divide computation into m steps. (assume $m = 2^d$)



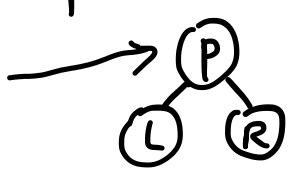
$C_0, C_1, \cdots, C_{m-1}$

(Each accumulate 1 unit of garbage)

2. Construct a balanced binary tree of d layers.



Leaves : computation steps C_i

Branches : 

$f_P = f_L f_R f_L^{-1}$

3. Run computation according to f_{root}.

- How much garbage accumulated ?
- How long does f_{root} take ?

} D&C §6.4.3.