Quantum computation

Lecture 1

Classical circuit model

Classical computational models

① Turing machine ← we're not going to discuss

② Circuit model: bits, wires, gates

Single-bit gates:

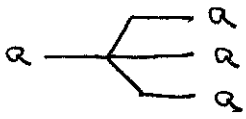
IDENTITY $a \to a$

a	a
0	0
1	1

NOT $a \to \boxed{\text{NOT}} \to \bar{a} = 1 \oplus a$

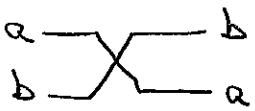
a	$\bar{a}$
0	1
1	0

FANOUT (COPY)

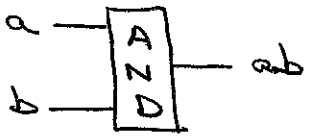


Two-bit gates:

CROSSOVER (SWAP)

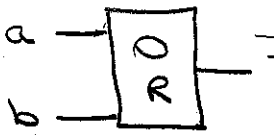


AND



a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

OR

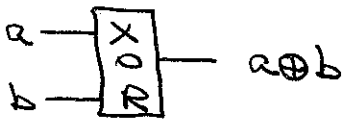


$\overline{ab} = 1 \oplus (1 \oplus a)(1 \oplus b)$

$= a \oplus b \oplus ab$

a	b	$a \oplus b \oplus ab$
0	0	0
0	1	1
1	0	1
1	1	1

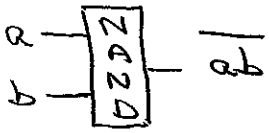
XOR



a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

← No nonlinear piece

NAND



$\overline{ab} = 1 \oplus ab$

a	b	$1 \oplus ab$
0	0	1
0	1	1
1	0	1
1	1	0

2 inputs

NOR

$\bar{a}\bar{b} = 1 \oplus a \oplus b \oplus ab$

a	b	$\bar{a}\bar{b}$
0	0	1
0	1	0
1	0	0
1	1	0

We could go on, but these are enough to do anything. What is it we want to do?

Only 2-bit gates (quadratic and linear polynomials) required.

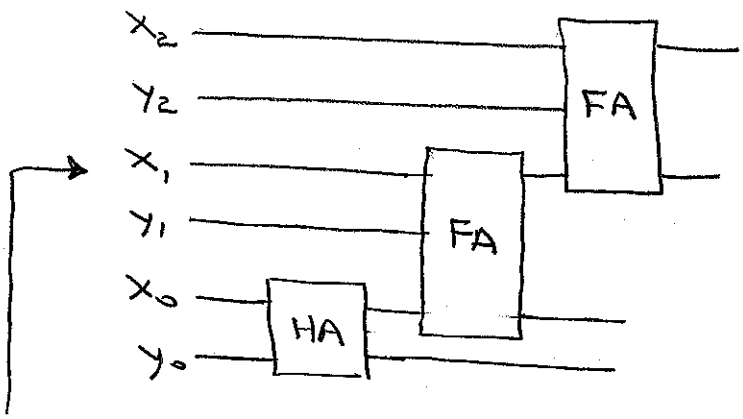
Evaluate functions $f: \{0,1\}^N \longrightarrow \{0,1\}^M$ that map N bits to M bits. But M=1 is good enough: Boolean functions (decision problems).

There are 2^{2^N} Boolean functions on N bits. The single- and two-bit gates are N=1 and N=2 Boolean functions.

Factoring

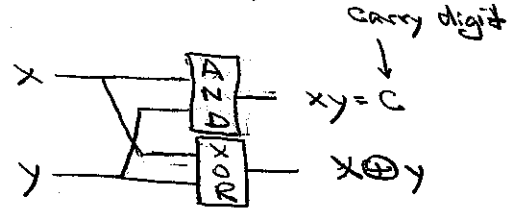
$f(x,y) = \begin{cases} 1 & \text{if } x \text{ has factor} < y \\ 0 & \text{otherwise} \end{cases}$

Example: adding $x = x_2x_1x_0$ and $y = y_2y_1y_0$



x	y	c	$x \oplus y$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

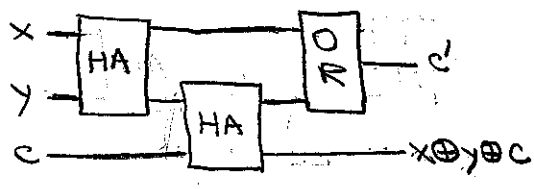
Half-adder (HA)



Binary version of the add-and-carry method taught to school children

x	y	c	c'	$x \oplus y \oplus c$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full-adder (FA)



$c' = xy \oplus cx \oplus cy$

Computing any Boolean function:

$N=1$: 4 functions

IDENTITY

a	a
0	0
1	1

NOT

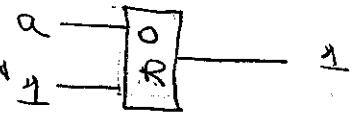
a	$\bar{a}$
0	1
1	0

Need NOT, AND, and OR + ancilla bits

0	0
1	0

ancilla bit

0	1
1	1



Induction: Assume there is a circuit for any Boolean function on N bits

f an $(N+1)$-bit function

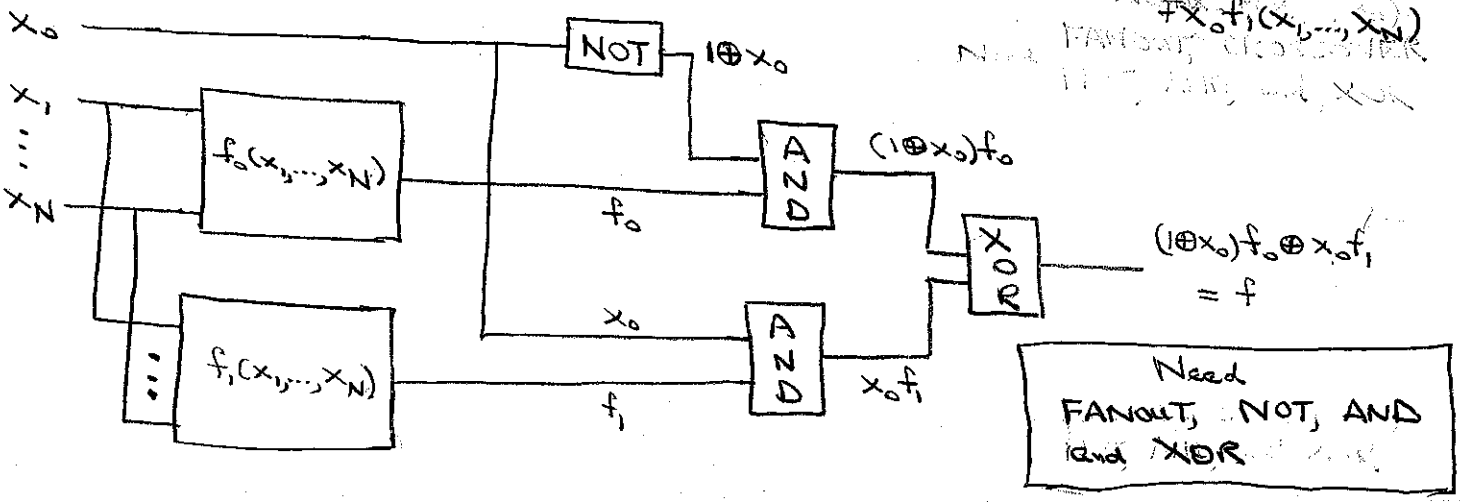
f_0 and f_1 N-bit functions defined by

$$f_0(x_1,\dots,x_N) \equiv f(0,x_1,\dots,x_N)$$
$$f_1(x_1,\dots,x_N) \equiv f(1,x_1,\dots,x_N)$$

$$f_{x_0}(x_1,\dots,x_N) \equiv f(x_0,x_1,\dots,x_N)$$

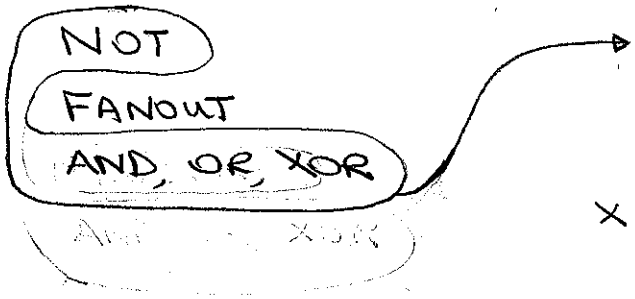
i.e., f_{x_0} OR f_{x_N}

$$f(x_0,x_1,\dots,x_N) = (1\oplus x_0) f_0(x_1,\dots,x_N) + x_0 f_1(x_1,\dots,x_N)$$



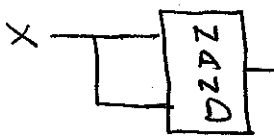
Need FANOUT, NOT, AND and XOR

We can evaluate any Boolean function using wires and ancilla bits



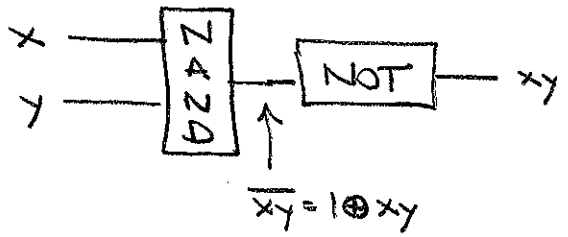
If we can use ancilla bits and FANOUT, we can get all these from NAND

NOT

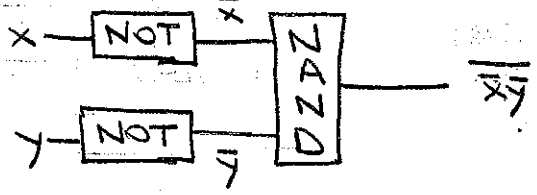


$1 \oplus x^2 = 1 \oplus x = \bar{x}$

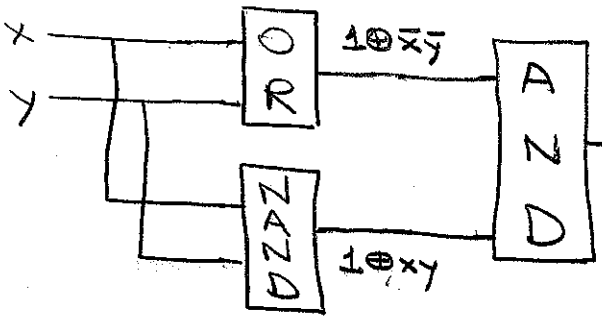
AND



OR



XOR



$(1 \oplus \bar{x}\bar{y})(1 \oplus xy) = 1 \oplus \bar{x}\bar{y} \oplus xy$

$= 1 \oplus (1 \oplus x)(1 \oplus y) + xy$

$= x \oplus y$

So with ancilla bits, and FANOUT, NAND is a <u>universal gate</u>.

NAND/NOT  can be used in place of NAND and FANOUT.

Computational complexity

Reversible classical computation.

Why?
Energy dissipation
Quantum computation is inherently reversible.

Any irreversible function $f: \{0,1\}^N \to \{0,1\}^M$ can be imbedded in a function $\tilde{f}: \{0,1\}^{N+M} \to \{0,1\}^{N+M}$ such that

$$\tilde{f}(x, 0^{(M)}) = (x, f(x)).$$

(N over x, M over $0^{(M)}$)

$\tilde{f}$ can be extended to a 1-1 function, e.g.,

$$\tilde{f}(x,y) = (x, y \oplus f(x))$$

← not a unique extension, but we will generally use it as the reversible function.

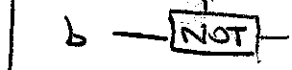
But can we do this with reversible gates?

$2^{N 2^N}$ N-bit gates (N in, N out) → $N=1$: 4; $N=2$: 256

$(2^N)!$ N-bit reversible gates → $N=1$: 2; $N=2$: 24

(reversible = permutations of the 2^N input strings)

CNOT is an example: XOR placed in 2nd bit, with first bit unchanged.



Can get CROSSOVER from 3 CNOT's.

↔ The 2-bit reversible gates are all linear, with invertible M; they do not have the irreversible, nonlinear xy piece.

$$\begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x' \\ y' \end{pmatrix} = M \begin{pmatrix} x \\ y \end{pmatrix} \oplus \begin{pmatrix} a \\ b \end{pmatrix}$$

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

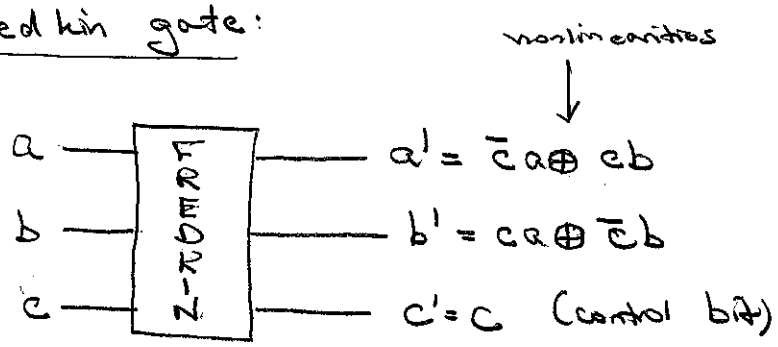
$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

24 reversible 2-bit gates

General 2-bit gate: $\begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a \oplus \sum_{i=1}^{2} M_{1i} x \oplus \sum_{i=1}^{2} M_{1i} y \oplus xy \\ b \oplus \sum_{i=1}^{2} M_{2i} x \oplus \sum_{i=1}^{2} M_{2i} y \oplus xy \end{pmatrix}$

We need 3-bit reversible gates to get a universal set.

Fredkin gate:



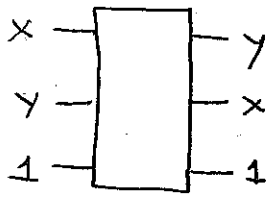
a	b	c	a'	b'	c'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	1	1	1

FREDKIN is its own inverse.

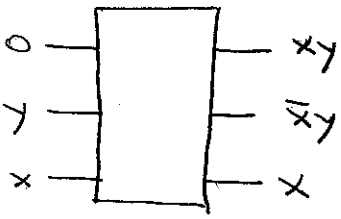
$c=0: (a,b) \to (a,b)$

$c=1: (a,b) \to (b,a)$

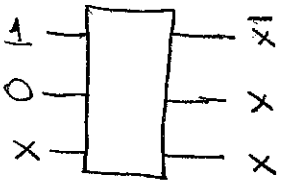
controlled swap (crossover)



CROSSOVER



AND

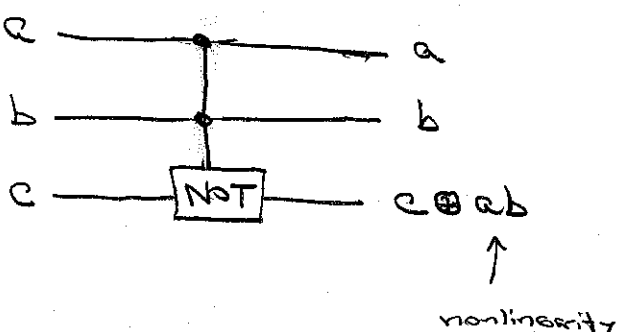


NOT

FANOUT

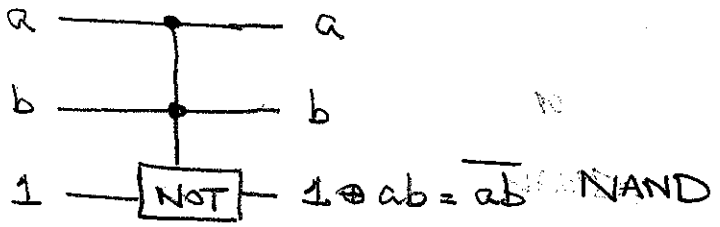
NOT and AND are NAND, so FREDKIN is a universal gate

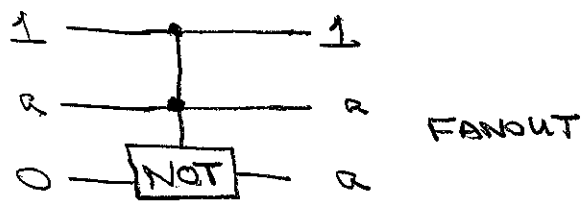
Toffoli gate: CCNOT



a	b	c	a	b	c ⊕ ab
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

TOFFOLI is its own inverse.





TOFFOLI is a universal gate.

Computing a function reversibly:

N M N M

$(x, 0, 0, y) \longrightarrow (x, f(x), g(x), y)$

garbage bits, however many are necessary

reversible gate implementation of f

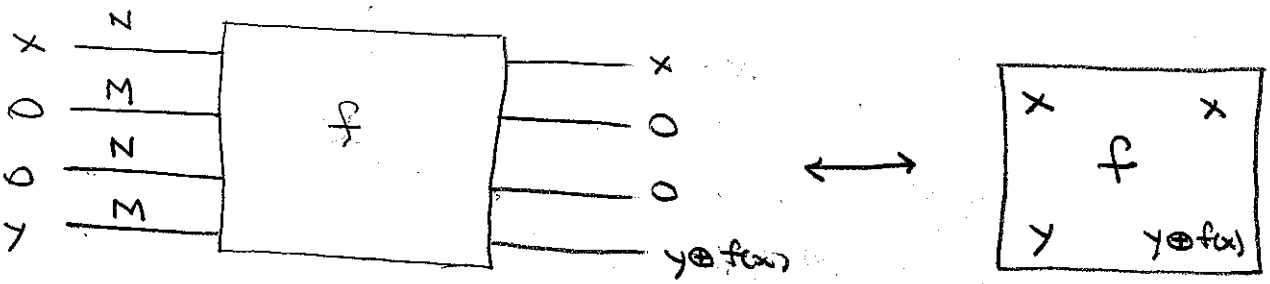
$\longrightarrow (x, f(x), g(x), y \oplus f(x))$

copy f(x) into last M bits; this can be done reversibly with CNOTs.

Uncompute f reversibly

$\longrightarrow (x, 0, 0, y \oplus f(x))$

Now we have our reversible computation of f w/ the garbage bits erased



Complexity?