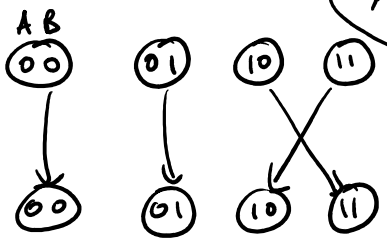
Lecture 5: Reversible (Deterministic) Computation

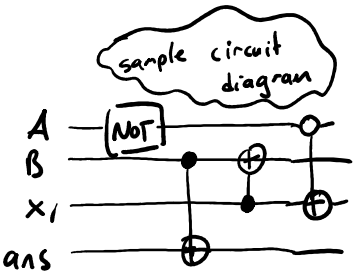
Inverse = Reverse:

- "Hat" → Hadamard (H)
- Deterministic:



"Add A to B"

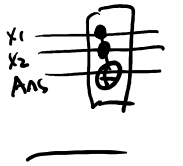
sample circuit diagram



"Add $F(x_1, x_2, x_3 \ldots)$ To Answer"

examples

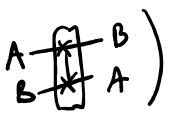
F	Instruction	Real name	Diagram
$F() = 0$	Add 0 to Ans (does nothing)	I	( )
$F() = 1$	Add 1 to Ans	NOT, X	NOT, ⊕
$F(x_1) = x_1$	Add x_1 to Ans	Controlled-NOT, or CNOT	
$F(x_1) = \neg x_1$	Add (NOT x_1) to Ans	C_0NOT	
$F(x_1, x_2) =$ x_1 AND x_2	Add (x_1 AND x_2) to Ans	Controlled-CNOT, CCNOT, Toffoli	
$F(x_1, x_2) =$ x_1 OR x_2	Add (x_1 OR x_2) to Ans		



Other reversible det. instructions:

$$\text{LeftShift}(A,B,C)^{-1} = \text{RightShift}(A,B,C)$$

$$\text{L/R Shift}(A,B) \equiv \text{SWAP}(A,B)$$



"Compute-A-Function Tasks":

$F: \{0,1\}^n \longrightarrow \{0,1\}^m$

input → output

$n=3, m=2$

e.g.: Count: $\{0,1\}^3 \to \{0,1\}^2$

Count $(x_1, x_2, x_3) = y = (y_1, y_2)$

input output

Truth Table of Count

input	output
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

MAJ XOR

$Count(x_1,x_2,x_3) = y = (y_1,y_2)$

binary rep. of # 1's in input

Q0: Can every truth table be computed by an AND/OR/NOT circuit?

*Q1: How efficiently?

A0: Yes.

Obs.: can assume # output bits is $m=1$.

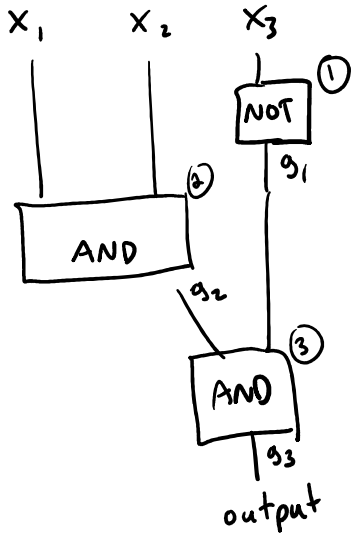
Obs.: Suffices to solve functions of the form

"Is Input 110?" : $\{0,1\}^3 \to \{0,1\}$

in	out
000	0
001	0
⋮	⋮
110	1
111	0

$MAJ(x_1,x_2,x_3) = ((\text{Is 011?}(x) \text{ OR Is 101?}(x)) \text{ OR Is 110?}(x)) \text{ OR Is 111?}(x)$

Is 110? (x_1,x_2,x_3):



reversible form

Given x_1, x_2, x_3:

$g1 := \text{NOT } x_3$

$g2 := x_1 \text{ AND } x_2$

$g3 := g_1 \text{ AND } g_2$

$answer := g_3$

:= → add to ANS

Given x_1, x_2, x_3

Initialize $g1, g2, g3$, ans to 0

Add NOT x_3 to $g1$

Add x_1 AND x_2 to $g2$

Add $g1$ AND $g2$ to $g3$

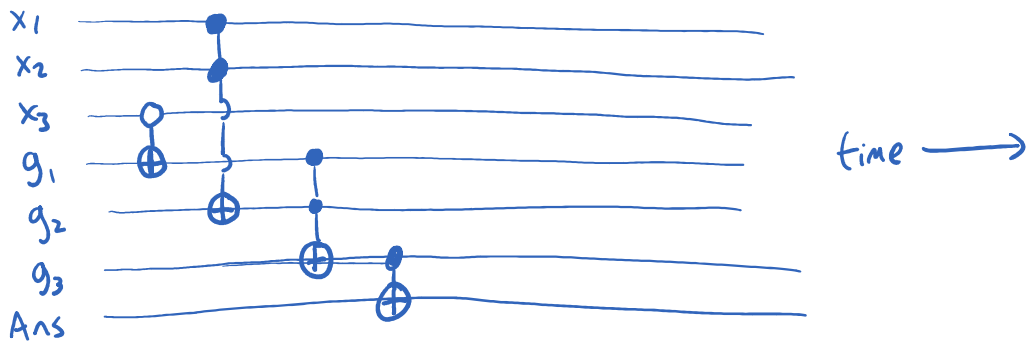
Add g_3 to ans

"Trashy-Add Is110? (x_1, x_2, x_3) To Ans":

// precondition: assumes temp vars g1, g2, g3 set to 0

Add (NOT x_2) to g1	CoNOT $(x_3, g1)$
Add (x_1 AND x_2) to g2	CCNOT $(x_1, x_2, g2)$
Add (g1 AND g2) to g3	CCNOT $(g1, g2, g3)$
Add g3 to Ans	CNOT (g_3, Ans)

// postcondition: g1, g2, g3 are some trash



Summary: For every $F: \{0,1\}^n \to \{0,1\}^m$

If you can compute it with a classical AND/OR/NOT circuit using G gates then you can convert to reversible Add-F-to-Ans code

using:

- $n+m+G$ bits in total
- $\approx G$ total instructions
- assumes the extra G vars are initialized to 0, left in a trash state

Moreover: $G \leq n \cdot m \cdot 2^n$ always

$\leq m \cdot 2^n/n$ [C. Shannon]

Remark: What about using computer code, not circuits?

Theorem: If python/C/TM code computes a

Remark: ... using ... to ... code, not circuits.

Theorem: If python/C/TM code computes a function $F: \{0,1\}^n \to \{0,1\}^m$ in T "steps", it can be compiled into AND/OR/NOT circuits with $G \leq \tilde{O}(T^2)$. ✓

($\leq \tilde{O}(T)$ usually)

Taking Out The Trash

Reason 1: Waste of space!

Would be great to *restore* trash bits to 00...0 by the end.

(Could reuse these temp vars several times.)

Reason 2: "Magic" in q. computing is: amplitudes can cancel out to 0.

0. Make A
1. Had A → q. state is now
   Ampl[0] = $\sqrt{1/2}$, Ampl[1] = $\sqrt{1/2}$
2. Had A → Ampl[0] = 1 Ampl[1] = $\frac{1}{2} - \frac{1}{2} = 0$
3. Print All
   100% chance A=0!

→ say replaced by many instructions but say they create $g_1, \ldots, g_4$ are all 0

Then after line 1: Ampl(0, 0000) = $\sqrt{1/2}$ ($g_1 \ldots g_4$)

Ampl(1, 0000) = $\sqrt{1/2}$

After line 2: ↳ Ampl(0, 0000) = 1

Ampl[1, 0000] = 0

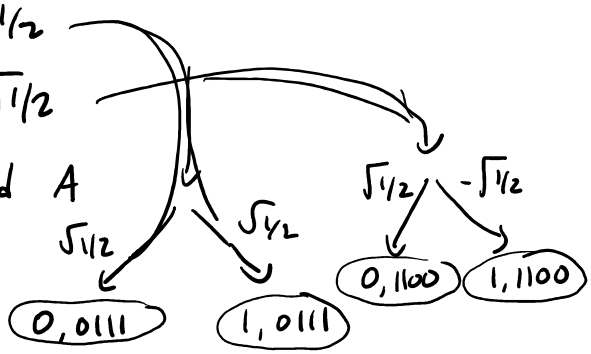
↳ Ampl(0,0000) = 1
Ampl[1,0000] = 0

Print All → 100% A=0 (& g_is are 0)

↳ OTOH, say line 1 replaced by instrs that leave $g_1 \ldots g_4$ in some trashy vals

eg: Ampl($\underset{A}{0}$, 0111) = $\sqrt{1/2}$
Ampl($\underset{A}{1}$, 1100] = $\sqrt{1/2}$

Now if we do Line 2: (Had A



Final: Ampl(00111] = 1/2
Ampl(10111] = 1/2
Ampl(01100] = 1/2
Ampl(11100] = -1/2

→ Print All
⇒ 1/2 prob. A=0
1/2 prob. A=1

---

garbage-free → "~~Trashy~~ Add Is110?(x_1,x_2,x_3) to Ans":

// precondition: g_1, g_2, g_3 initialized to 0

Add (NOT x_3) to g_1
Add (x_1 AND x_2) to g_2
Add (g_1 AND g_2) to g_3
Add g_3 to Ans

// Ans is good, but g_1, g_2, g_3 are trash

reverse!

Add (g_1 AND g_2) to g_3
Add (x_1 AND x_2) to g_2
Add (NOT x_3) to g_1

// postcondition: g_1, g_2, g_3 restored to 0

We can always make deterministic reversible code garbage-free with $\leq 2 \cdot G$ instructions