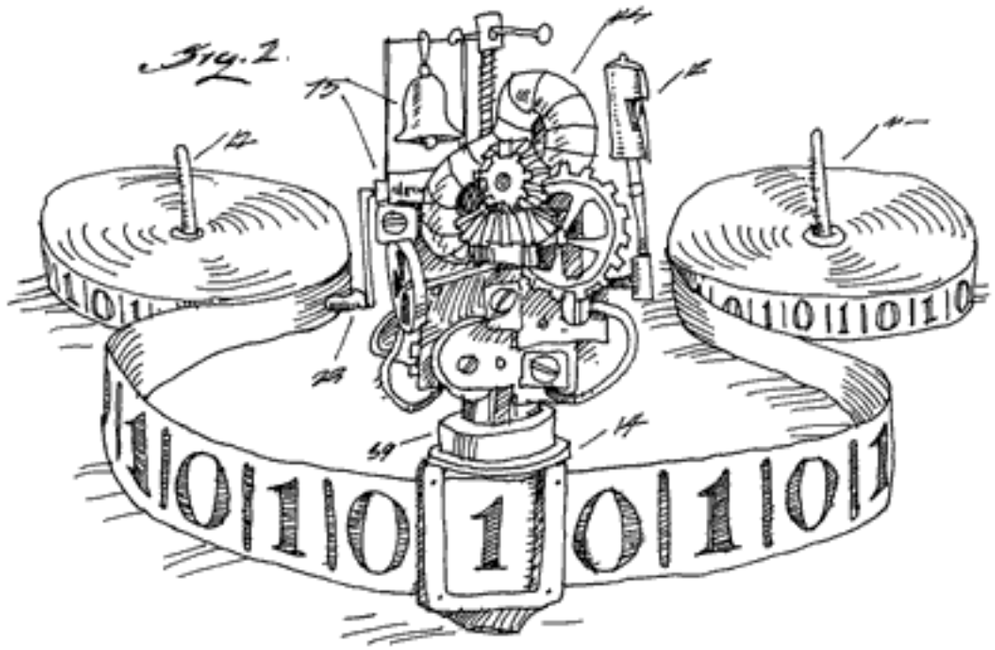
CSC4010: Introduction to Theoretical Computer Science



Lecture 14

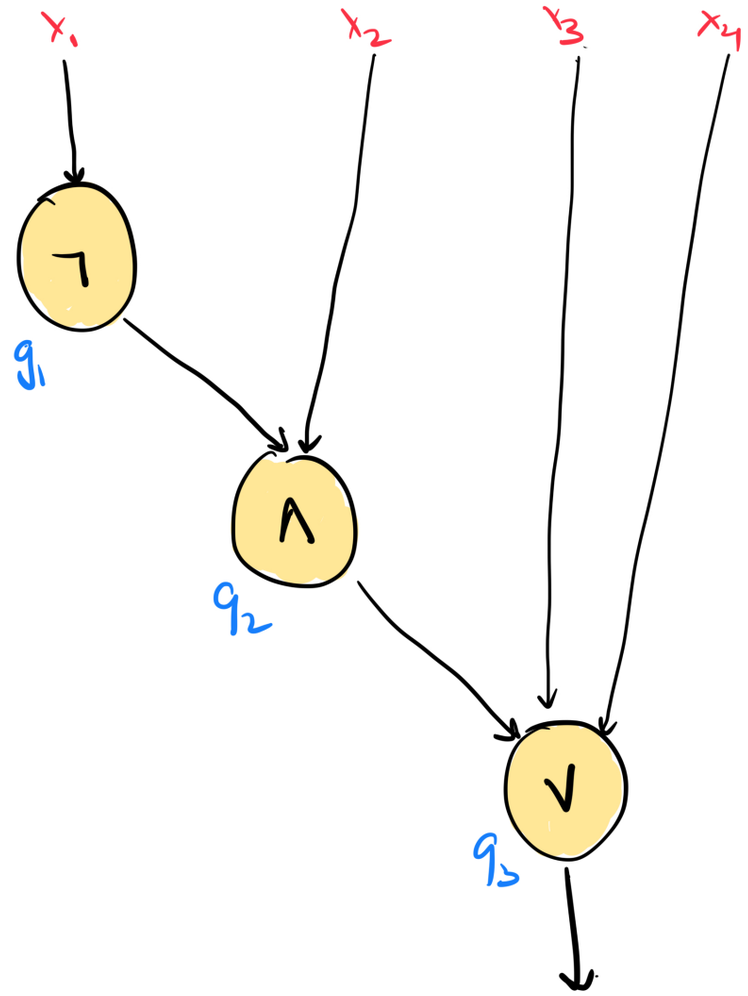
Boolean Circuits and Formulas

Boolean Circuits

A boolean circuit consists of n input bits labelled $x_1, x_2, \cdots, x_n$ and a sequence of gates $g_1, g_2, \cdots, g_m$. There are three types of gates:

- NOT gates
- AND gates
- OR gates

The input to the gate g_k are either one of the input bits $x_1, \cdots, x_n$ or one of the gates $g_1, \cdots, g_{k-1}$. The output of the circuit is the output of gate g_m.

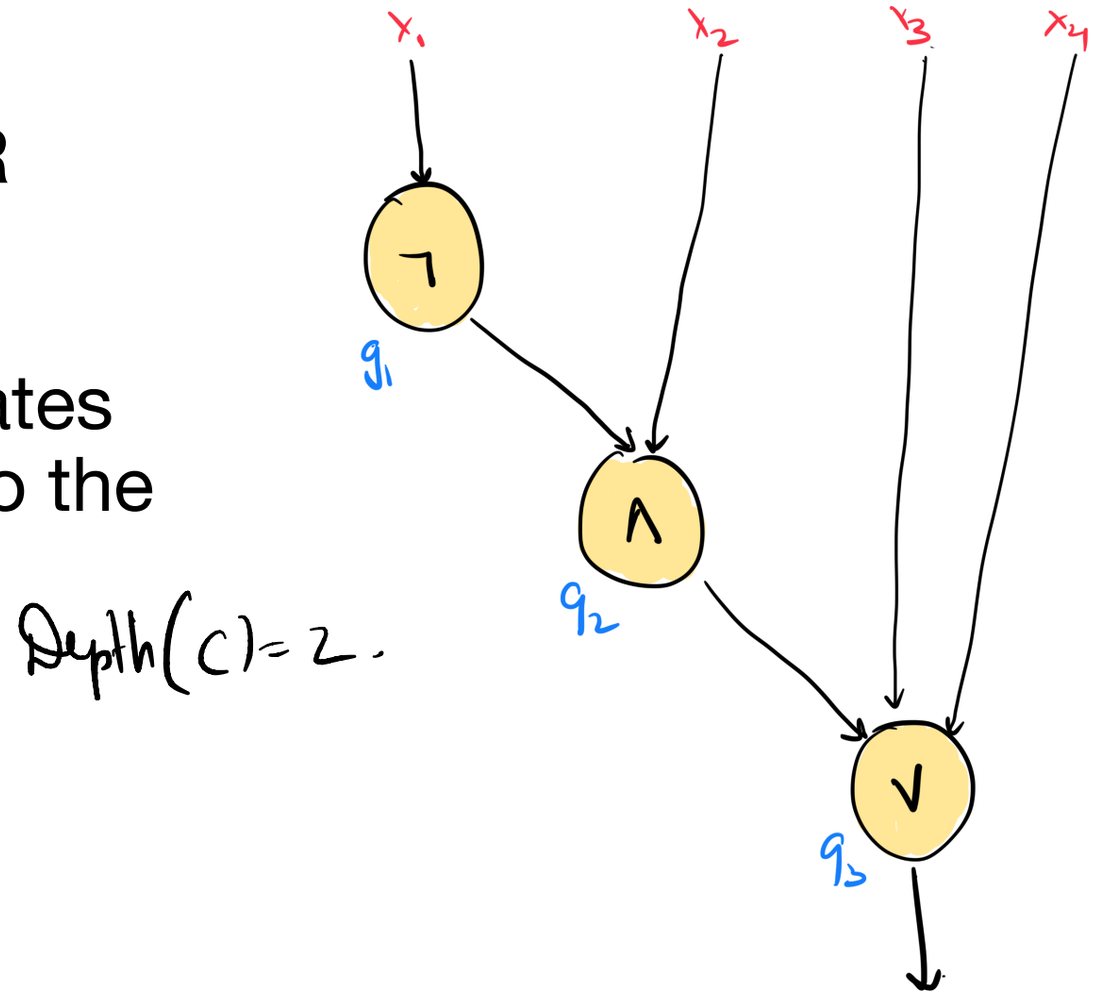


Boolean Circuit as a DAG

Complexity Measure

Size: The total number of AND and OR gates it contains.

Depth: The number of AND and OR gates in the longest path from any input bit to the output of the circuit.



Boolean Circuit as a DAG

Languages and Boolean Circuits

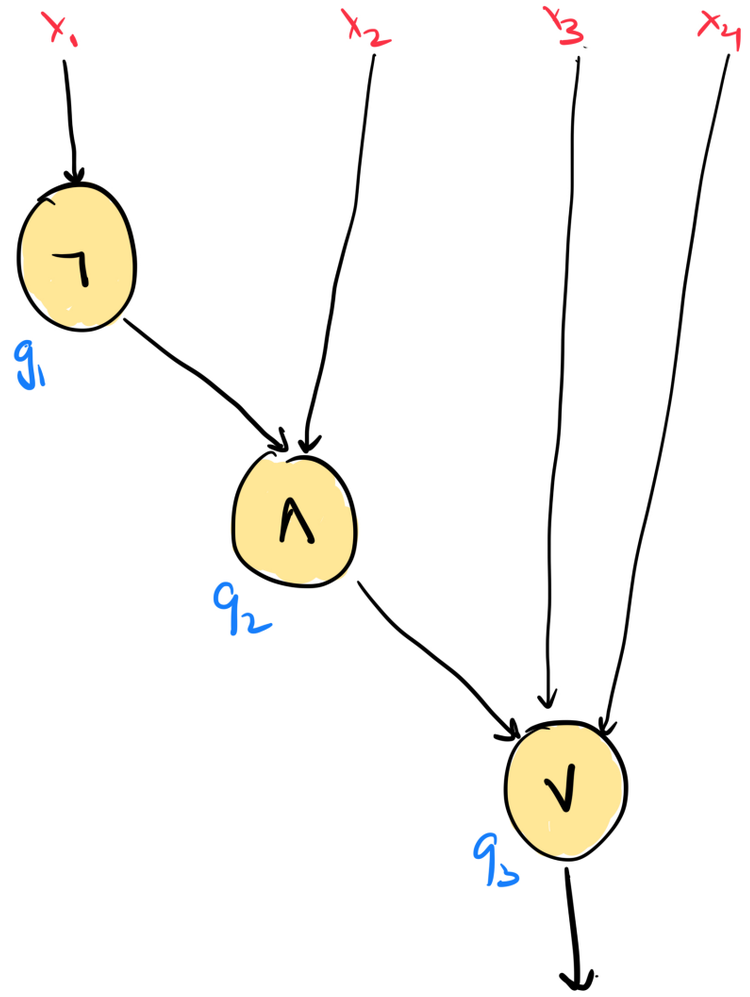
$(\neg x_1 \wedge x_2) \vee x_3 \vee x_4$.

- Number of inputs to a Boolean circuit is fixed. It computes a Boolean function:
  $f : \{0,1\}^n \to \{0,1\}$.

- A language $L \subseteq \{0,1\}^*$ can be expressed as family of Boolean functions:
  $f_n : \{0,1\}^n \to \{0,1\}$, where
  $L = \cup_{i \geq 1} f_i^{-1}(1)$.

- A single circuit does not compute a language, but a family of circuits $\{C_n\}_{n \geq 0}$ does.

- In contrast with a TM. Non-uniform model of computation.



Boolean Circuit as a DAG

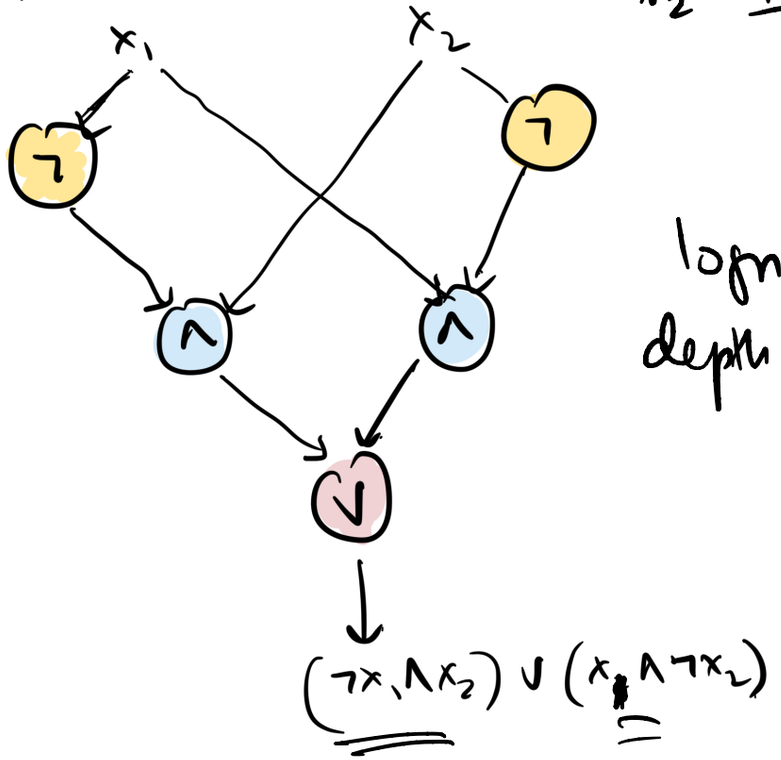
Examples: PARITY

- The parity of a set of n bits is a function $\oplus_n : \{0,1\}^n \to \{0,1\}$ defined as $\oplus_n (x_1, \cdots, x_n) = \sum_{i=1}^{n} x_i \ (\mod 2)$.

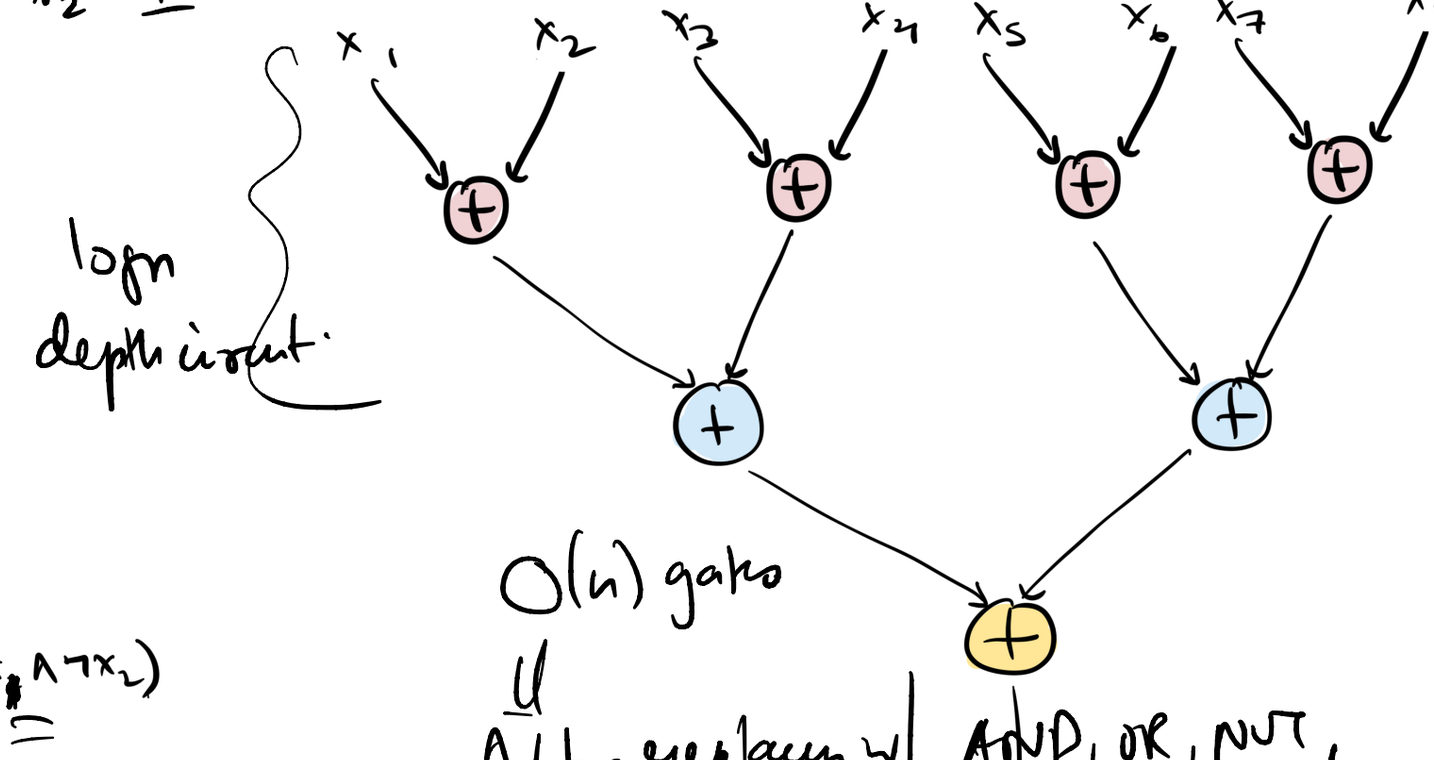
$\oplus_n(x_1, x_2, \cdots, x_n)$ = 0 if even # x_i's are 1; 1 if odd # x_i's are 1.

$\oplus_n(x_1, x_2, \cdots, x_n)$ using a circuit of size $O(n)$ and depth $O(\log n)$

$\oplus_2(x_1, x_2) = 1$ (=) exactly one of x_1 or $x_2 = 1$



log n depth circuit.



$O(n)$ gates

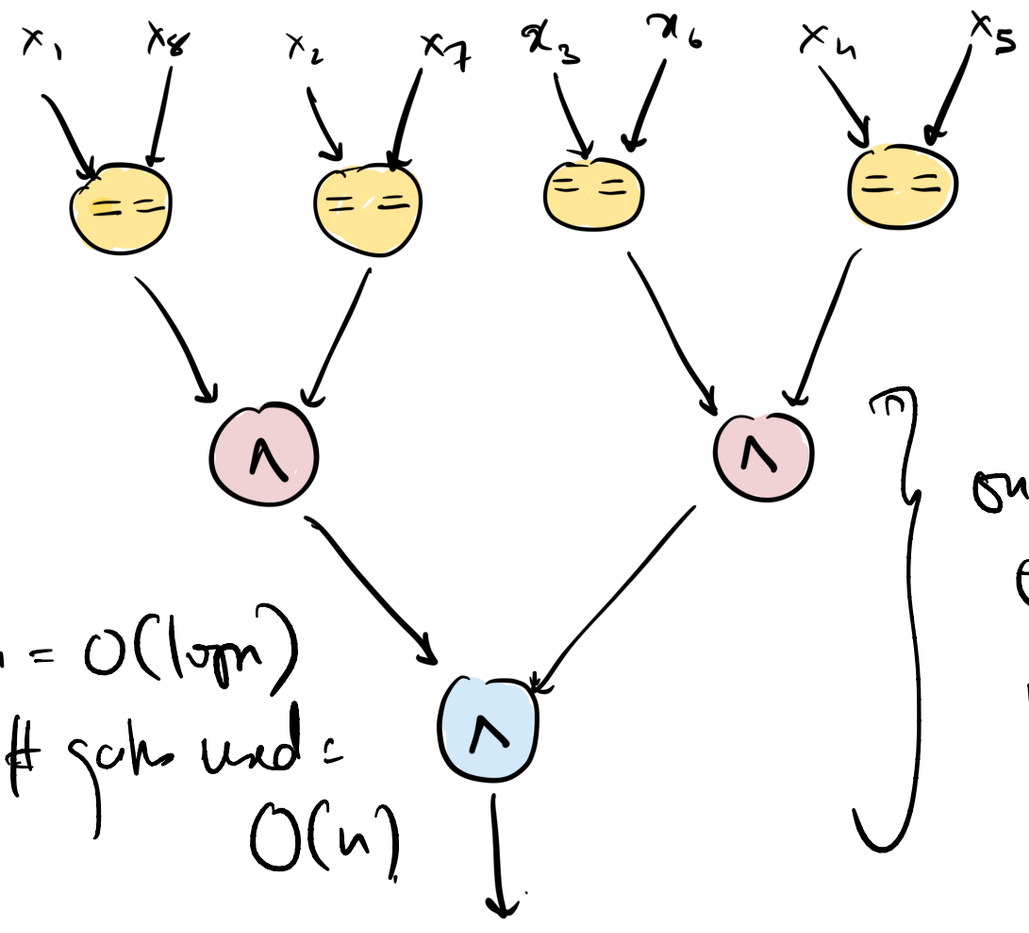
⇓

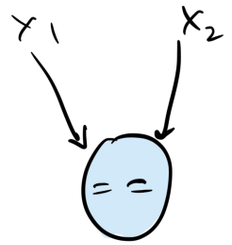
After replacing w/ AND, OR, NOT, total depth = $\Theta(\log n)$ and size = $\Theta(n)$

Examples: PALINDROME

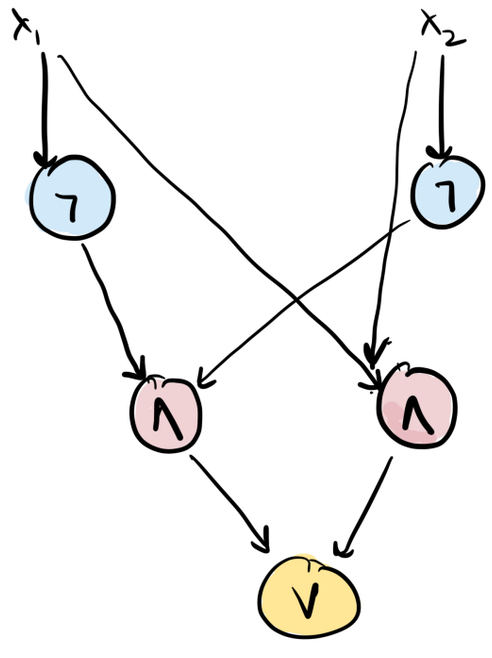
x is an n-bit string it is a palindrome
$\iff x_1 = x_n,\ x_2 = x_{n-1} \cdots$

- Define $f : \{0,1\}^n \to \{0,1\}$ by $f(x) = 1 \iff x$ is a palindrome.





$(x_1 == x_2)$
$= (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$



Shannon's Theorem

Theorem: For every function $f : \{0,1\}^n \to \{0,1\}$, there exists a Boolean circuit C of size at most $2^n + 1$ that computes f.

$\uparrow Maj_n(x_1, x_2, \ldots x_n) = 1 \iff \sum_{i=1}^{n} x_i \geq n/2$

x_1	x_2	x_3	$Maj_3(x)$	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\to 1$
1	0	0	0	
1	0	1	1	$\to 1$
1	1	0	1	$\to 1$
1	1	1	1	$\to 1$

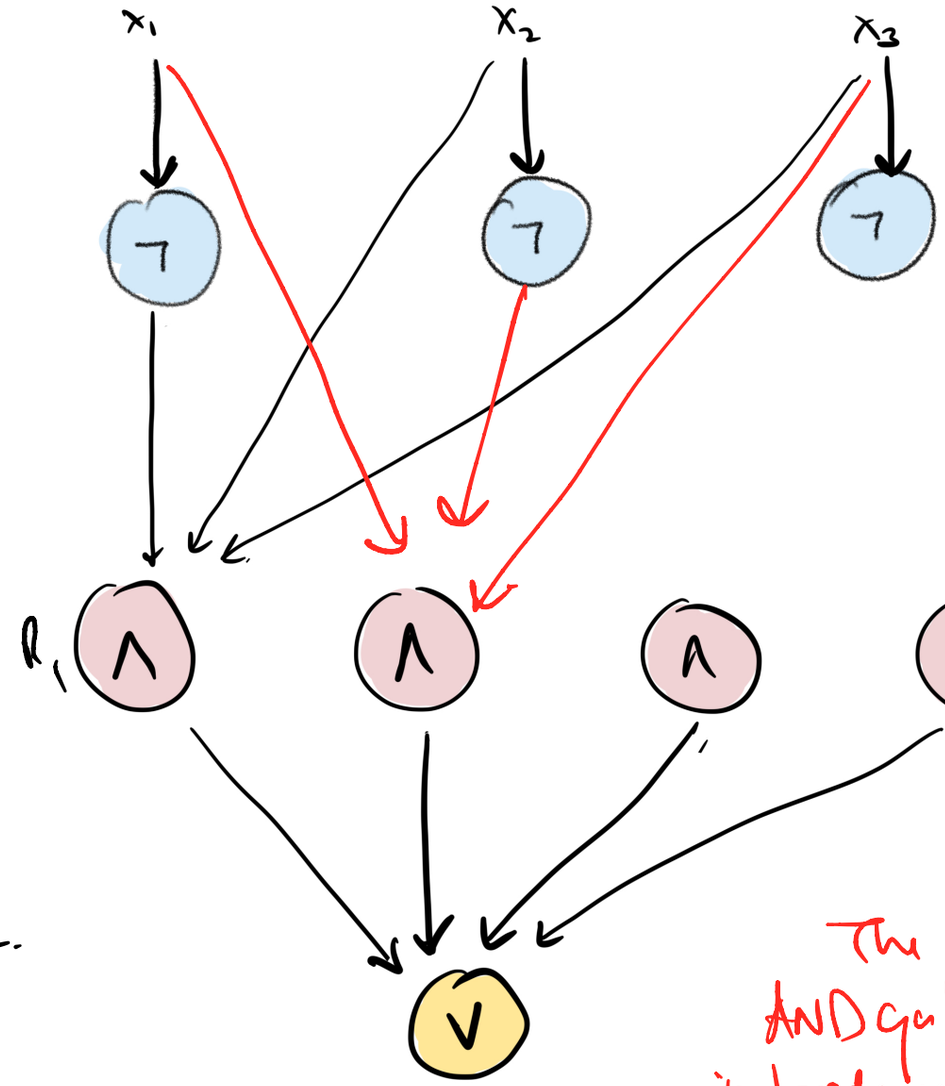
2^n rows

R_1

R_2

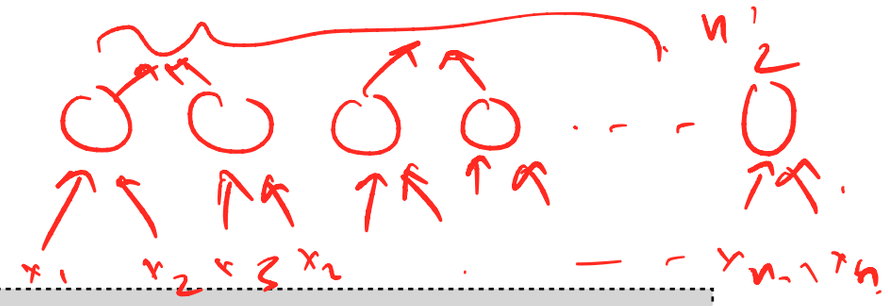
$(\neg x_1 \wedge x_2 \wedge x_3)$

$(x_1 \wedge \neg x_2 \wedge x_3)$



The total # AND gates we introduced $\leq 2^n$.
+ an OR gate.

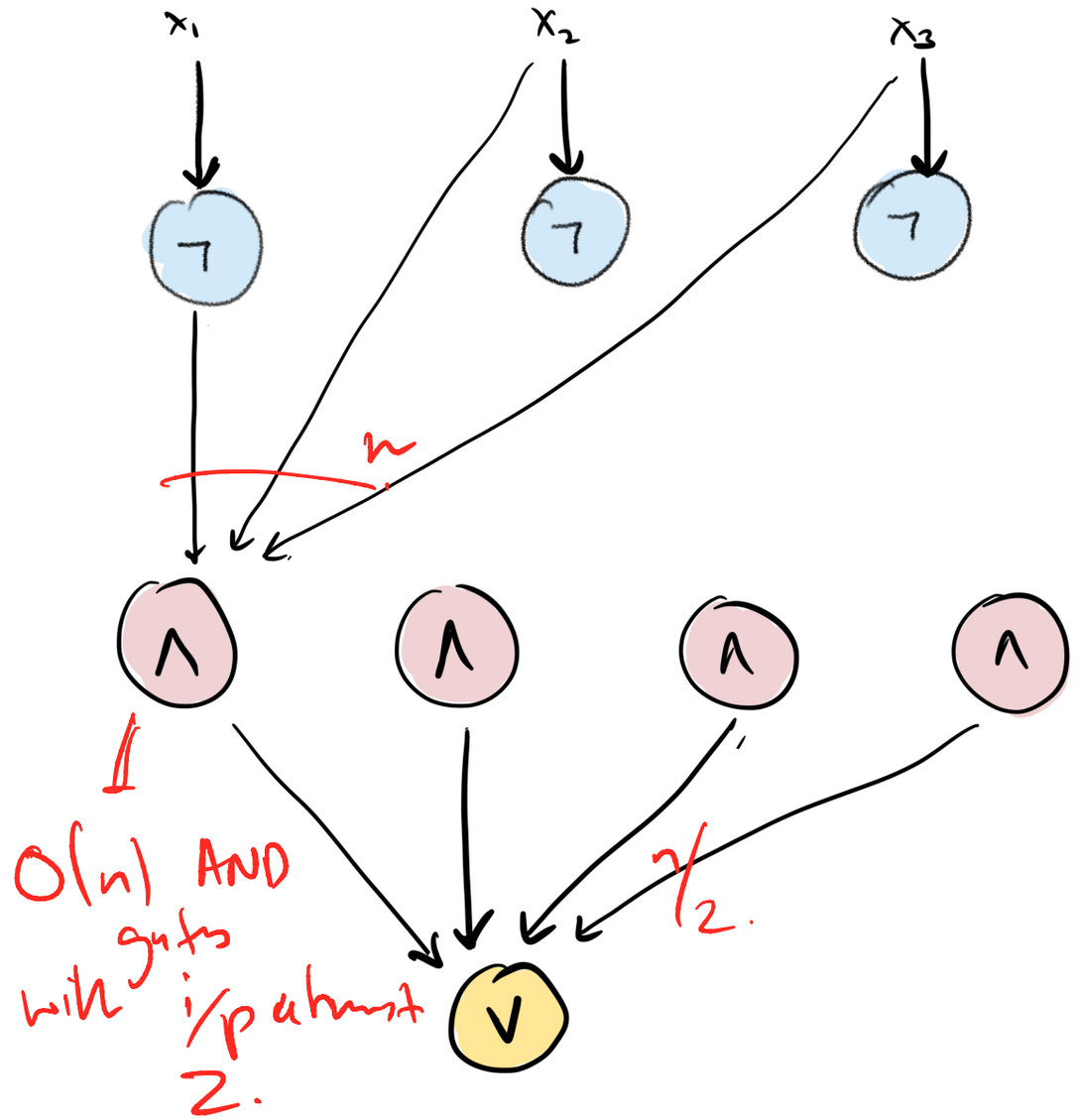
Shannon's Theorem



Theorem: For every function $f : \{0,1\}^n \to \{0,1\}$, there exists a Boolean circuit C of size at most $2^n + 1$ that computes f.

The size of the circuit is then $O(n 2^n)$.

x_1	x_2	x_3	$Maj_3(x)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Shannon's Theorem

Corollary: For every language $L \subseteq \{0,1\}^*$, there exists a family of circuits $\{C_n\}_{n \geq 0}$ that computes L.

$\{f_i\}_{i \geq 0}$ that represent L:

$f_i(x) = 1 \Leftrightarrow x \in L \cap \{0,1\}^i$.

a) Corresponding f_i, $\exists$ a circuit C_i of size $\leq 2^i + 1$

b) So, the family of circuits is $\{C_i\}_{i \geq 0}$.

for any i:

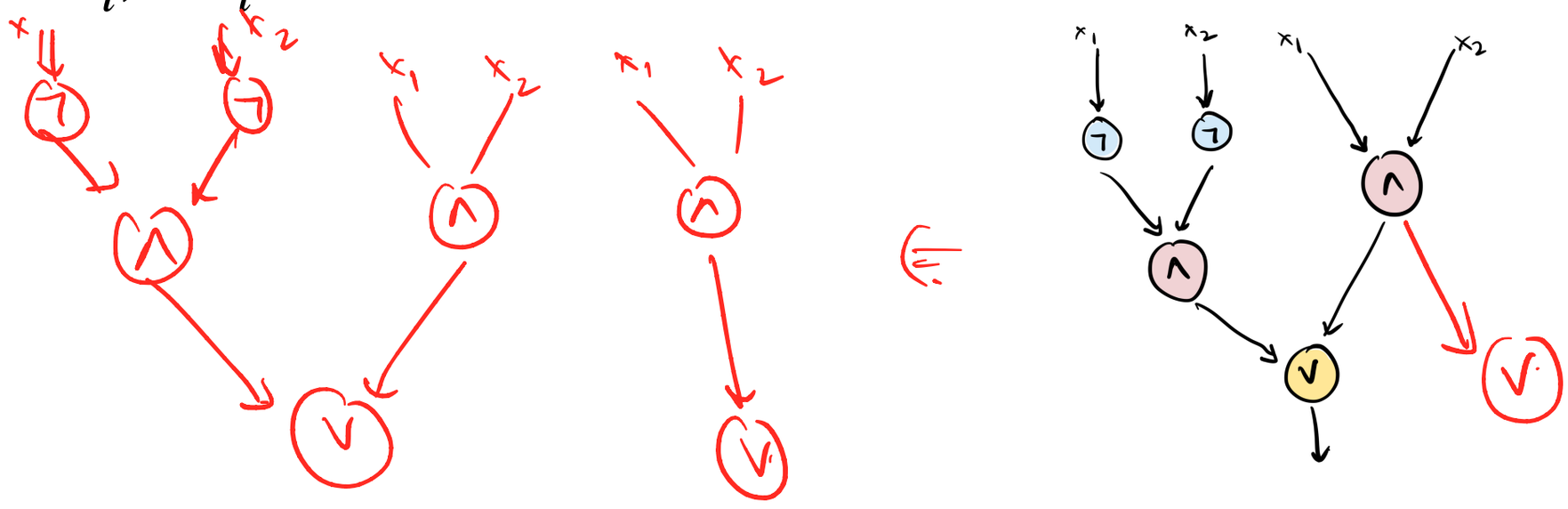
$C_i(x) = 1 \Leftrightarrow f_i(x) = 1 \Leftrightarrow x \in L \cap \{0,1\}^i$.

Boolean Formulas

- A boolean formula is a Boolean circuit which has an additional restriction: all of the gates have only one output wire. Each output can be used as the input of at most one gate. (Bounded fan-out).

- A boolean formula has bounded fan-in when each of $\wedge$, $\vee$ gates have exactly two inputs.

- Represented as a rooted binary tree, with leaves being multiple copies of x_i, $\neg x_i$.

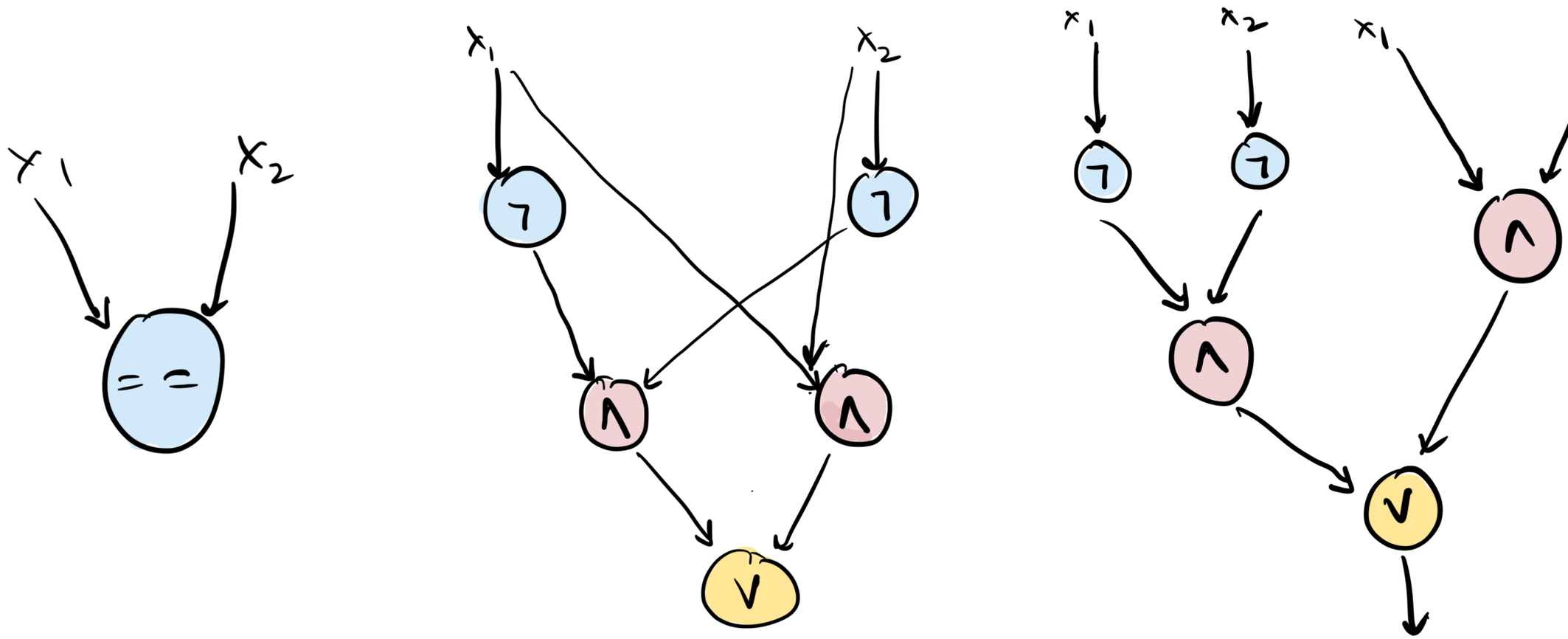


Complexity Measure: Boolean Formula

F = # leaves in the formula.

Size: The number of leaves in the tree representation of the formula.

→ As it is a binary tree, the # of gates in the circuit is ≤ c·F where c is a constant.



Conjunctive Normal Form

Definition: A clause is an OR of literals. Example: $C = \bar{x}_1 \vee x_2 \vee \bar{x}_3$.

Definition: A conjunctive normal form (CNF) formula is a conjunction of clauses (AND of ORs of literals). Example:

$$f(x) = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

$C_1 \qquad \wedge \qquad C_2$

$\Downarrow$

each clause is an OR of upto n literals.

Claim: Any boolean function can be written as CNF.

Every Function has a CNF Formula

↳ DNF form = $C_1 \vee C_2 \vee \cdots$ (Disjunctive normal form)

$C_i = \wedge \ell_j$

Theorem: Let $f : \{0,1\}^n \to \{0,1\}$ be any function, then there is a CNF formula that computes f, with at most 2^n clauses and n literals per clause.

↳ $C_1 \wedge C_2 \wedge C_3 \cdots \wedge C_{2^n}$

↳ $C_i = \ell_1 \vee \ell_2 \vee \ell_3 \cdots \ell_n$.

$K_1 \Leftarrow$

x_1	x_2	x_3	$Maj_3(x)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$f(a_1 a_2 a_3) = 0$ then, $C_{a_1 a_2 a_3}$ is in the formula.

↳ $C_{000} =$ $x_i = 0$ then $\ell_i = x_i$

$x_i = 1$ then $\ell_i = \neg x_i$

↳ $x_1 \vee x_2 \vee x_3$

$C_{000}(x) = 0 \Leftrightarrow x_1 = 0, x_2 = 0, x_3 = 0.$

CNF : $C_{000} \wedge C_{001} \wedge C_{010} \wedge C_{100}$.

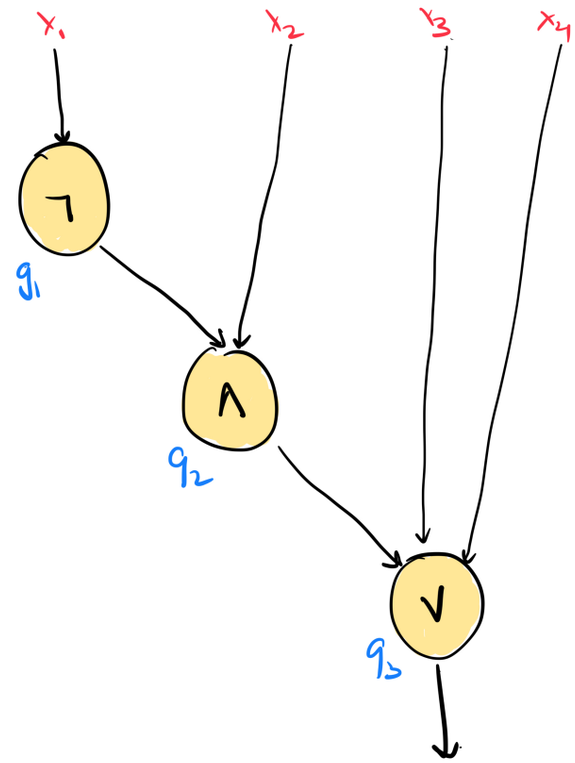
$f(a_1 a_2 a_3) = 1 \Rightarrow C_{a_1 a_2 a_3}$ is not in the formula.

Why are we studying circuits?

- Why are we studying circuits?
  - It will help us show that many interesting problems are NP-Complete.
- Key Idea: Code as Data

→ Eg CLIQUE, SATISFIABILITY
↳ Cook Levin Thm.
a) CIRCUIT-SAT is NP-complete.
(Using circuits as code, and feeding it as data to TMs)

Circuits as Code as Data



Recall: A Turing Machine can be encoded as a string $\langle M \rangle$. $\langle M \rangle$ is an algorithm, but at the same time, $\langle M \rangle$ can be an input to another algorithm.

Similar Idea: A circuit C can be encoded as a string $\langle C \rangle$.

- Assume n variables and m gates.
- Each node in the graph can be of four types: literal, AND, OR, NOT

$1^n 0 1^m 0 \langle v \rangle \langle \text{type } g_j \text{ of } v \rangle \cdots$ ← "x_i or g_j"

$\langle M' \rangle$

$\langle C \rangle = ?$ true $|\langle C \rangle| = \text{poly}(\text{size } g_{15} \text{ of } C)$

$M' =$

	x_1	x_2	x_3	x_4	g_1	g_2	g_3
x_1	0	0	0	0	1	0	0
x_2	0	0	0	0	0	1	0
x_3	0	0	0	0	0	0	1
x_4	0	0	0	0	0	0	1
g_1	0	0	0	0	0	1	0
g_2	0	0	0	0	0	0	1
g_3	0	0	0	0	0	0	0

$M'(x_1, g_1) = 1$ ⇔ x_1 is an input to g_1 ⇒ the directed edge from x_1 to g_1.

Circuit Value Problem

↳ n inputs and one output.

Let CVP $= \{\langle C, x\rangle : C$ is a circuit and $C(x) = 1\}$

↳ n bit string. The circuit is in the form ↳ encoding we specified.

Lemma: CVP $\in P$

Proof Sketch: Suppose C has m nodes. To compute $C(x)$:

Consider the encoding $\langle C, x\rangle$

(a) Mark all the input nodes with their values.

and, mark all i/p variables by values given by x.

(b) While there is an unmarked node:

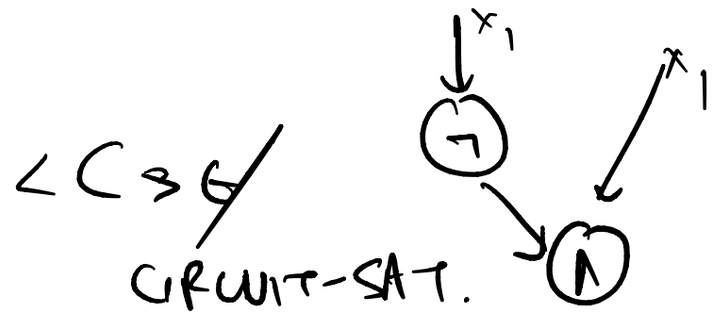
i) For every gate g find all nodes that feed into g. If they are marked with their values, then mark g with its value.

Runtime is $|\langle C, x\rangle|^k$ for a constant $k \geq 0$.

↳ by looking at its type: AND, OR, NOT.

↳ to evaluate g on its inputs.

Circuit Satisfiability



We say that circuit C (on n inputs) is satisfiable if there exists $x \in \{0,1\}^n$ such that $C(x) = 1$

→ Given a circuit ∃ a subset of the i/p so that the output is 1.

$$\text{CIRCUIT-SAT} = \{\langle C \rangle \mid \exists x \text{ s.t. } C(x) = 1\}$$

CIRCUIT-SAT $\in NP$

Next time: CIRCUIT-SAT is NP-complete.

a) Certificate and a verifier that is sound & complete

b) Certificate & Runtime is polynomial.

$\langle C \rangle \in$ CIRCUIT-SAT, $\exists x \in \{0,1\}^n$ so that $C(x) = 1$.

(a) Certificate is x, $|x| = n$, and we can (on x in polynomial time).

(Algorithm for CVP)

(b) If $\langle C \rangle \notin$ CIRCUIT-SAT then there is no such certificate.

Parting Thoughts

Circuit Size

2^{2^n} = # Boolean functions. 2^n

x_1	x_2	x_n		
0	0	0	0/1	$\to 2$
0	0	1	0/1	$\times 2$
⋮	⋮	⋮	⋮	$\times 2$
				!

$(2)^{2^n}$

Every function has a circuit (of size exponential).

Do you really need exponential size?

(Shannon)

Theorem: For every $n \geq 4$, there exists a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by any Boolean circuit of size $2^{n/2}/4$.

Counting argument: (1) Count the # of Boolean functions $f : \{0,1\}^n \to \{0,1\}$

(2) Count # of Boolean circuits of size $2^{n/2}/4$.

- Existence of such a Boolean function

- We don't know an explicit Boolean function for which we need circuits of size $> 2^{n/2}/4$.

(1) > (2)

(open!!)

Circuit Complexity of a Binary Language

Let $Y \subseteq \{0,1\}^*$. Let $Y_n = Y \cap \{0,1\}^n$.

Definition: The circuit complexity of Y is the function $S : \mathbb{N} \to \mathbb{N}$ defined by $S(n)$ = the size of the smallest circuit that computes Y_n.

Let $S : \mathbb{N} \to \mathbb{N}$ be a function

Definition: SIZE$(S) = \{Y \subseteq \{0,1\} * :$ the circuit complexity of Y is $O(S)\}$

Definition: PSIZE $= \{Y \subseteq \{0,1\}^* :$ circuit complexity of Y is poly$(n)\}$

Next Time

PSIZE = P ?

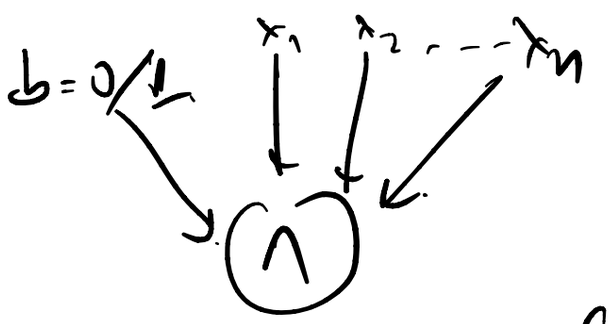
We will show P ⊆ PSIZE. But is the converse true?

Theorem: There exists an undecidable language Y with circuit complexity $O(n)$.

Let $M_1, M_2, \ldots, M_n, \ldots$ be an ordering of Turing machines.

$$L = \{ 1^n : M_n \text{ halts on input } \varepsilon \}$$

a) L is an undecidable language. (If not, we can use L to solve: $\{ \langle M \rangle \mid M \text{ halts on } \varepsilon \}$ which is undecidable.)



$$C_n = \begin{cases} x_1 \wedge x_2 \wedge x_3 \cdots \wedge x_n & \text{if } 1^n \in L. \\ 0 & \text{if } 1^n \notin L \end{cases}$$

$\rightsquigarrow$ O(n) gates, depth ≤ 2.

"non-uniformity"

How to interpret PSIZE?

We will prove $P \subsetneq$ PSIZE

Circuits are more powerful than Turing machines.

Does this mean something is wrong with Turing machines?

No! Something is "wrong" with the circuit model.

PSIZE is not a good model of tractable languages.

"$Y \in$ PSIZE" means there is a family of polynomial-sized circuits that decide Y (one for each input size).

Each circuit performs only a polynomial work. But what about the work needed to construct these circuits?

Given n it is hard to construct C_n (as was evident in the previous slide).

Nonuniformity

PSIZE allows us to use different "algorithms" for different input lengths.

Undecidable language: a) A TM deals w/ all input lengths.
b) An undecidable language has infinite size.

Computing in this "nonuniform" manner is cheating/unrealistic.

But circuits as we will see, are a valuable conceptual tool.

In the previous example, for a particular length n, $|L_n|$ is finite.

And as we are allowing a separate algorithm for each n, we can hardcode $x \in L_n$ in the algorithm.