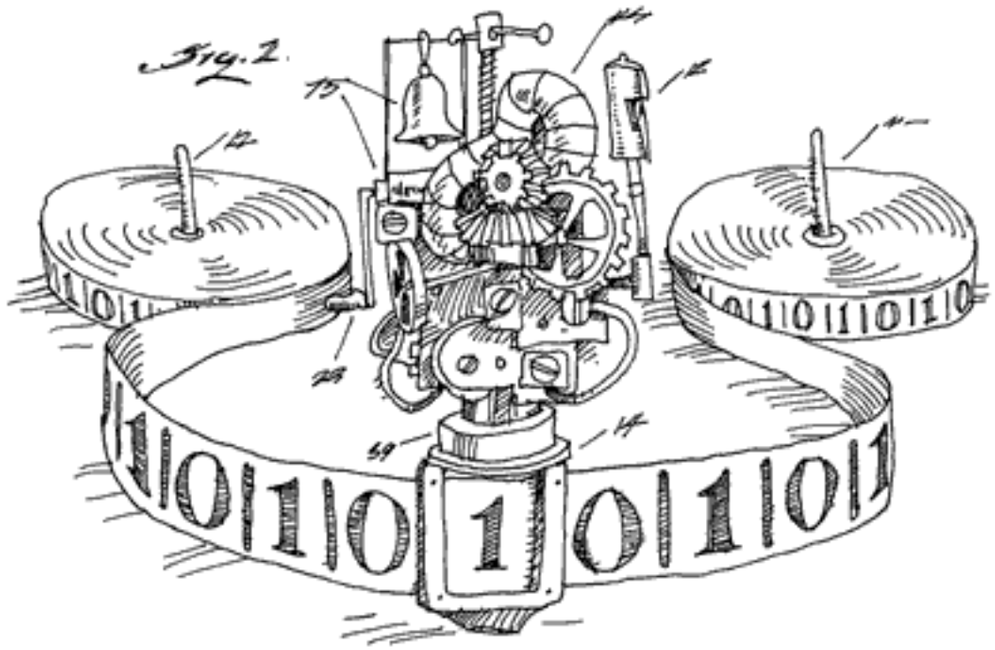
CSC4010: Introduction to Theoretical Computer Science



Lecture 13

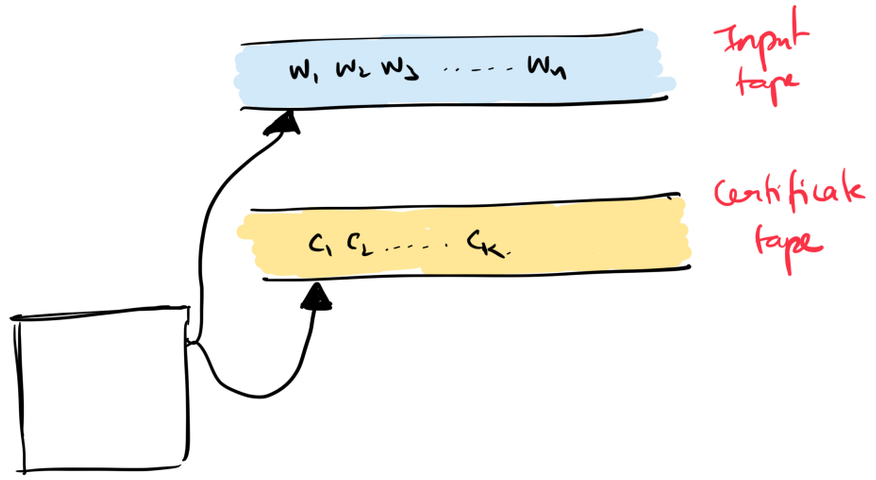
Recap

- A Verifier is a Turing machine with a read-only certificate tape, in addition to an input tape. The input to the verifier is a pair (x, c), with x, the input string written on the input tape, and c, the certificate written on the certificate tape.

- **Definition:** The language $L(V)$ recognised by the verifier V is

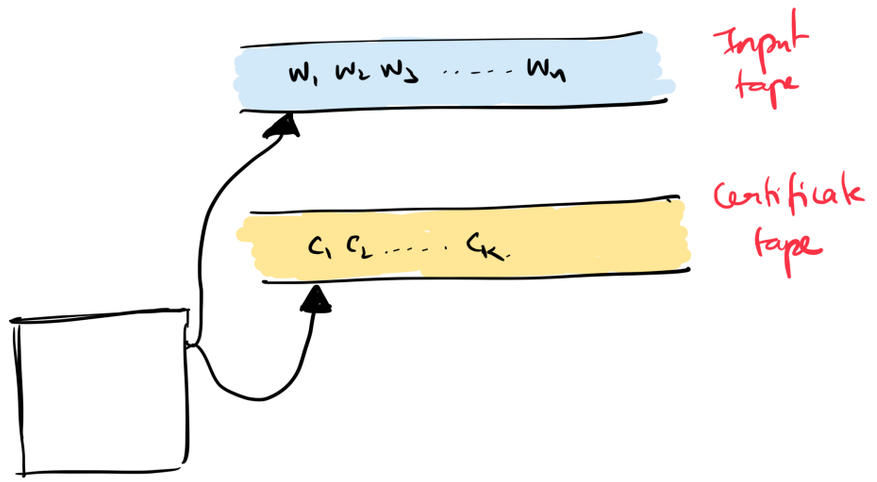
$$L(V) = \{x \in \{0,1\}^* : \exists c \in \{0,1\}^* \text{ s.t. } V \text{ accepts } (x, c)\}$$

The verifier decides $L(V)$ if it halts on all inputs (x, c).



Verifier: As a Turing Machine

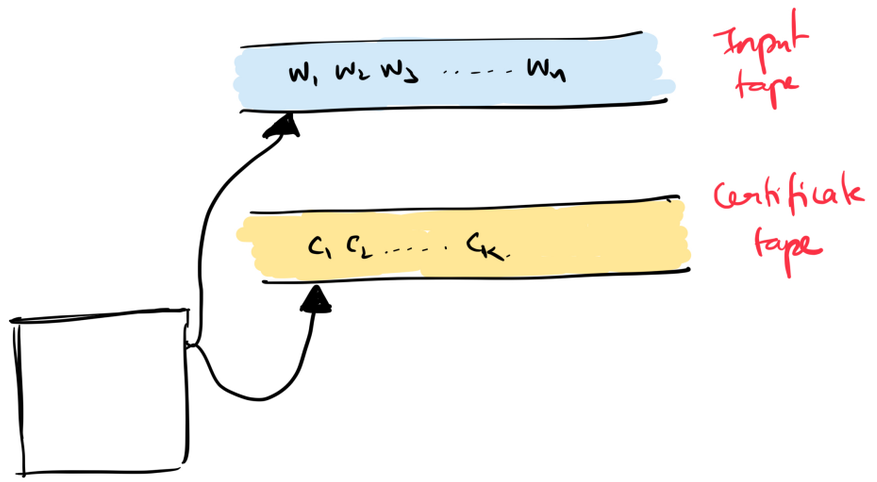
Definition: A polynomial-time verifier is a verifier that decides its language and has time cost $O(n^c)$ for some constant $c \geq 0$.



Thus, for any polynomial-time verifier V and $x \in L(V)$, there must be a certificate of length polynomial in $|x|$ that causes V to accept.

Verifier: As a Turing Machine

Definition: A polynomial-time verifier is a verifier that decides its language and has time cost $O(n^c)$ for some constant $c \geq 0$.



Thus, for any polynomial-time verifier V and $x \in L(V)$, there must be a certificate of length polynomial in $|x|$ that causes V to accept.

Definition: NP is the is the set of all languages that can be decided by polynomial-time verifiers.
NP stands for non-deterministic polynomial time.

Polynomial-time Reducibility

Definition: A language $A \subseteq \{0,1\}^*$ is polynomial-time reducible to the language B, denoted $A \leq_P B$ if and only if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$ such that:

- For every $x \in \{0,1\}^*$, we have $x \in A \iff f(x) \in B$,
- There is a polynomial time Turing machine F, that on any input $x \in \{0,1\}^*$, replaces it with $f(x)$ when it halts.

Polynomial-time Reducibility

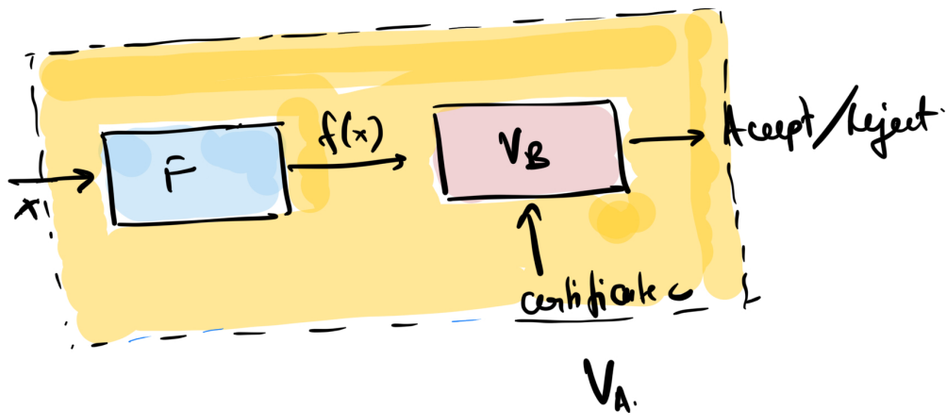
Theorem: For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_P B$:

- If $B \in P$, then $A \in P$ as well, and
- If $B \in NP$, then $A \in NP$ as well.

Polynomial-time Reducibility

Theorem: For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_P B$:

- If $B \in P$, then $A \in P$ as well, and
- If $B \in NP$, then $A \in NP$ as well.



V_A on i/p x certificate c :

(1) Compute f(x)

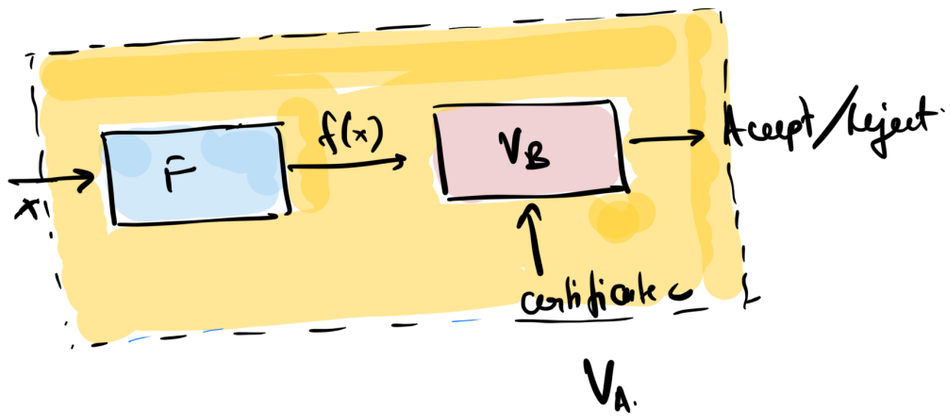
(2) Run $V_B(f(x), c)$

(3) Accept iff $V_B(f(x), c)$ accepts.

Polynomial-time Reducibility

Theorem: For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_P B$:

- If $B \in P$, then $A \in P$ as well, and
- If $B \in NP$, then $A \in NP$ as well.



V_A on i/p x certificate c :

(1) Compute $f(x)$

(2) Run $V_B(f(x), c)$

(3) Accept iff $V_B(f(x), c)$ accepts.

$x \in A \Leftrightarrow f(x) \in B$

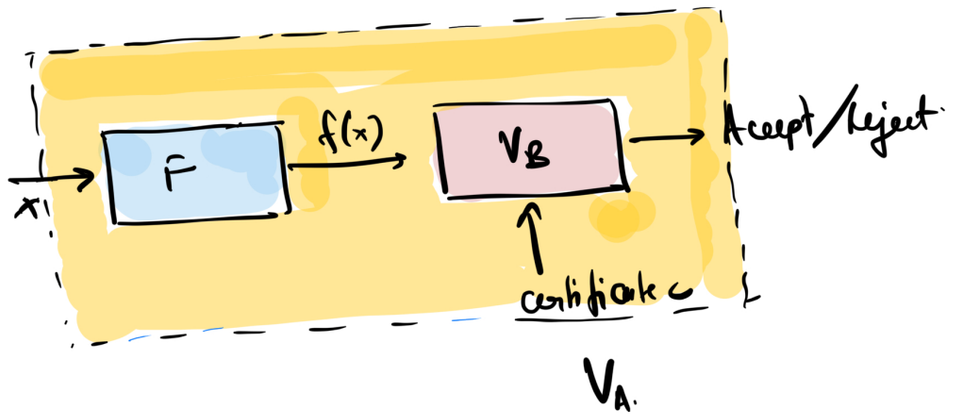
$\Leftrightarrow \exists c$ s.t. $V_B(f(x), c)$ accepts

$\Leftrightarrow \exists c$ s.t. $V_A(x, c)$ accepts

(Completeness & Soundness)

Polynomial-time Reducibility

Theorem: For every two languages $A, B \subseteq \{0,1\}^*$ that satisfy $A \leq_P B$:
- If $B \in P$, then $A \in P$ as well, and
- If $B \in NP$, then $A \in NP$ as well.



V_A on i/p x certificate c :
(1) Compute f(x)
(2) Run $V_B(f(x), c)$
(3) Accept iff $V_B(f(x), c)$ accepts.

Claim: (1) Running time of V_A is polynomial.
(2) $|c| \leq p(|x|)$ for some polynomial p.

Proof: (1) F runs in polynomial time: $O(|x|^k)$
(2) $|f(x)| \leq O(|x|^k)$
(3) $|c| \leq |f(x)|^\ell \leq |x|^{k\ell}$
(4) Running time of V_B: $|f(x)|^m$

NP-Hardness

Definition: The language L is NP-hard if every language $A \in NP$ satisfies $A \leq_P L$. The language L is NP-complete if $L \in NP$ and L is NP-hard.

Proposition: If any NP-hard language L is in P, then $P = NP$.
Equivalently, if $P \neq NP$ and L is NP-hard, then $L \notin P$.

A First NP Complete Language

Theorem: There exists an NP-complete language.

A First NP Complete Language

Theorem: There exists an NP-complete language.

Proof: $A_{TM}^{\text{cert}} = \{\langle M \rangle x 0^m 1^t \mid \exists c \text{ with } |c| \leq m \text{ s.t. } M \text{ accepts } (x, c) \text{ in } t \text{ steps}\}$

A First NP Complete Language

Theorem: There exists an NP-complete language.

Proof: $A_{TM}^{\text{cert}} = \{\langle M \rangle x 0^m 1^t \mid \exists c \text{ with } |c| \leq m \text{ s.t. } M \text{ accepts } (x, c) \text{ in } t \text{ steps}\}$

$A_{TM}^{\text{cert}} \in NP$:

(1) Consider $\langle M \rangle x 0^m 1^t \in A_{TM}^{\text{cert}}$. There exists c of length m such that $M(x, c) = 1$ in t steps.

(2) The verifier V for A_{TM}^{cert} simulates M on (x, c) for t steps.

A First NP Complete Language

Theorem: There exists an NP-complete language.

Proof: $A_{TM}^{\text{cert}} = \{\langle M \rangle x 0^m 1^t \mid \exists c \text{ with } |c| \leq m \text{ s.t. } M \text{ accepts } (x, c) \text{ in } t \text{ steps}\}$

$A_{TM}^{\text{cert}} \in NP$:

(1) Consider $\langle M \rangle x 0^m 1^t \in A_{TM}^{\text{cert}}$. There exists c of length m such that $M(x, c) = 1$ in t steps.

(2) The verifier V for A_{TM}^{cert} simulates M on (x, c) for t steps.

Completeness: For every $\langle M \rangle x 0^m 1^t \in A_{TM}^{\text{cert}}$ there exists such a certificate c.

Soundness: For every $\langle M \rangle x 0^m 1^t \notin A_{TM}^{\text{cert}}$ there is no such certificate c .

Polynomial Time: $|c| \leq m$, each step of M can be be simulated in polynomial time.

A First NP Complete Language

Theorem: There exists an NP-complete language.

Proof: $A_{TM}^{\text{cert}} = \{\langle M \rangle x 0^m 1^t \mid \exists c \text{ with } |c| \leq m \text{ s.t. } M \text{ accepts } (x, c) \text{ in } t \text{ steps}\}$

A_{TM}^{cert} is NP-hard: Consider any $L \in NP$. There exists polynomials p, q and a verifier V such that for every $x \in \{0,1\}^n$:

(1) If $x \in L$, there exists $u \in \{0,1\}^{p(n)}$ that causes $V(x, u)$ to accept.

(2) If $x \notin L$, every possible $u \in \{0,1\}^{p(n)}$ causes $V(x, u)$ to reject.

(3) V has time cost $q(n)$.

A First NP Complete Language

Theorem: There exists an NP-complete language.

Proof: $A_{TM}^{\text{cert}} = \{\langle M \rangle x 0^m 1^t \mid \exists c \text{ with } |c| \leq m \text{ s.t. } M \text{ accepts } (x, c) \text{ in } t \text{ steps}\}$

A_{TM}^{cert} is NP-hard: Consider any $L \in NP$. There exists polynomials p, q and a verifier V such that for every $x \in \{0,1\}^n$:

(1) If $x \in L$, there exists $u \in \{0,1\}^{p(n)}$ that causes $V(x, u)$ to accept.

(2) If $x \notin L$, every possible $u \in \{0,1\}^{p(n)}$ causes $V(x, u)$ to reject.

(3) V has time cost $q(n)$.

$$f(x) = \langle V \rangle x 0^{p(|x|)} 1^{q(|x|)}$$

Boolean Circuits and Formulas

Boolean Circuits and Formulas

- We have seen several models of computation.
- Next: Computation based on networks of logic gates.
  - Closely related to practical electronics.
  - Extremely important in theory, too!

Binary Logical Operations

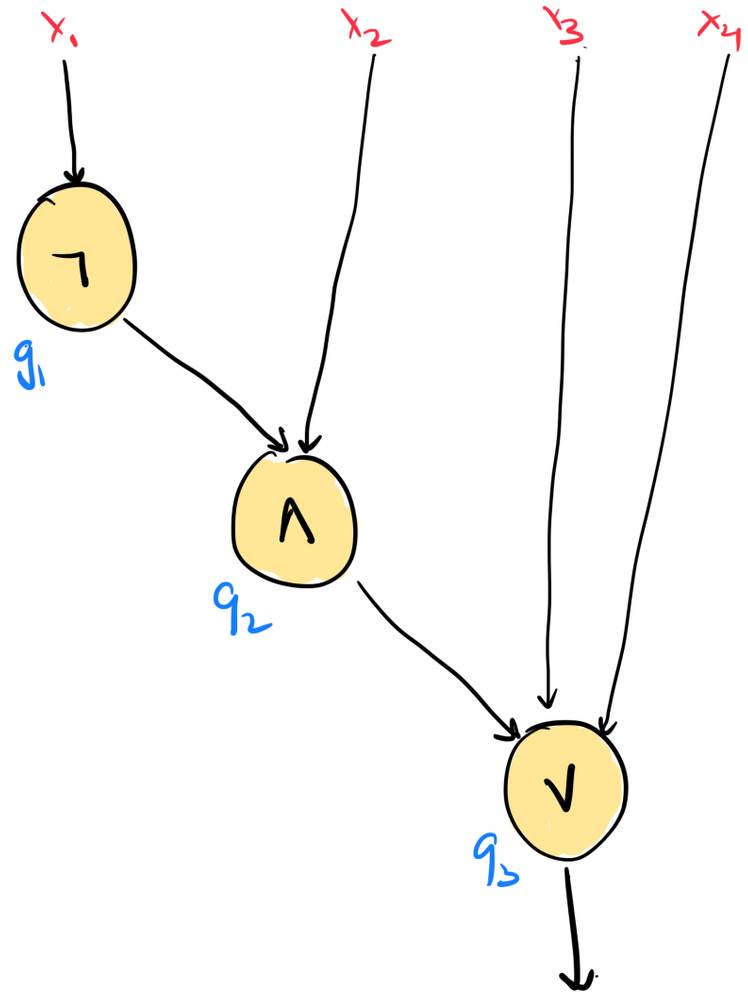
- AND: $a \wedge b$
- OR: $a \vee b$
- XOR: $a \oplus b$
- Equality: $a == b$
- NOT: $\bar{a}$ or $\neg a$

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.

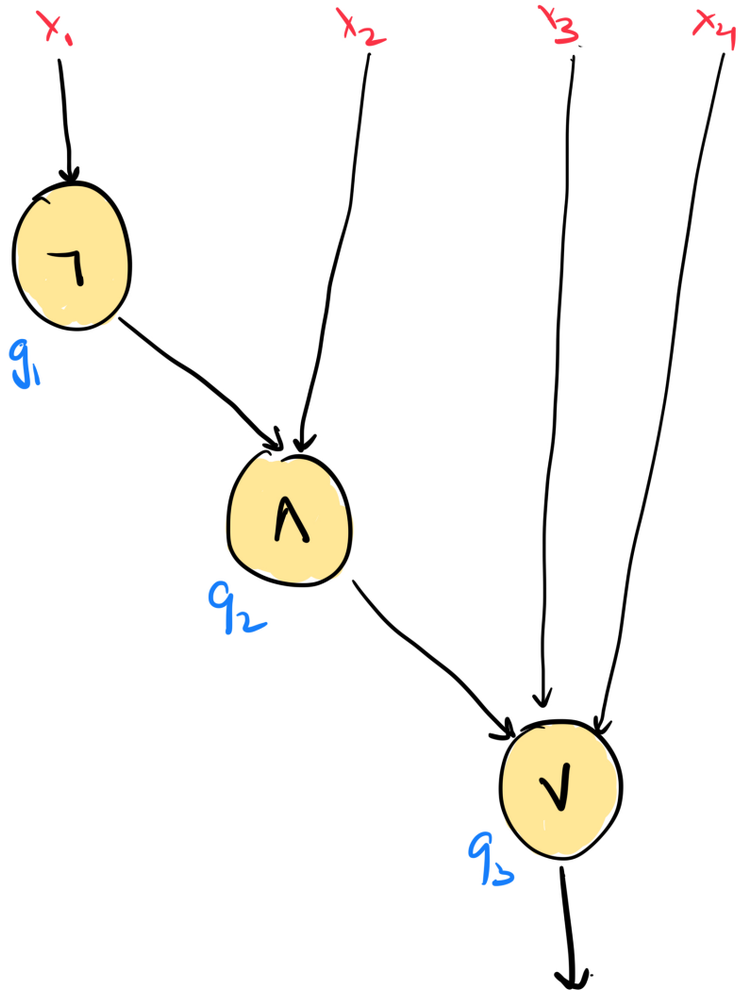


Boolean Circuit as a DAG

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

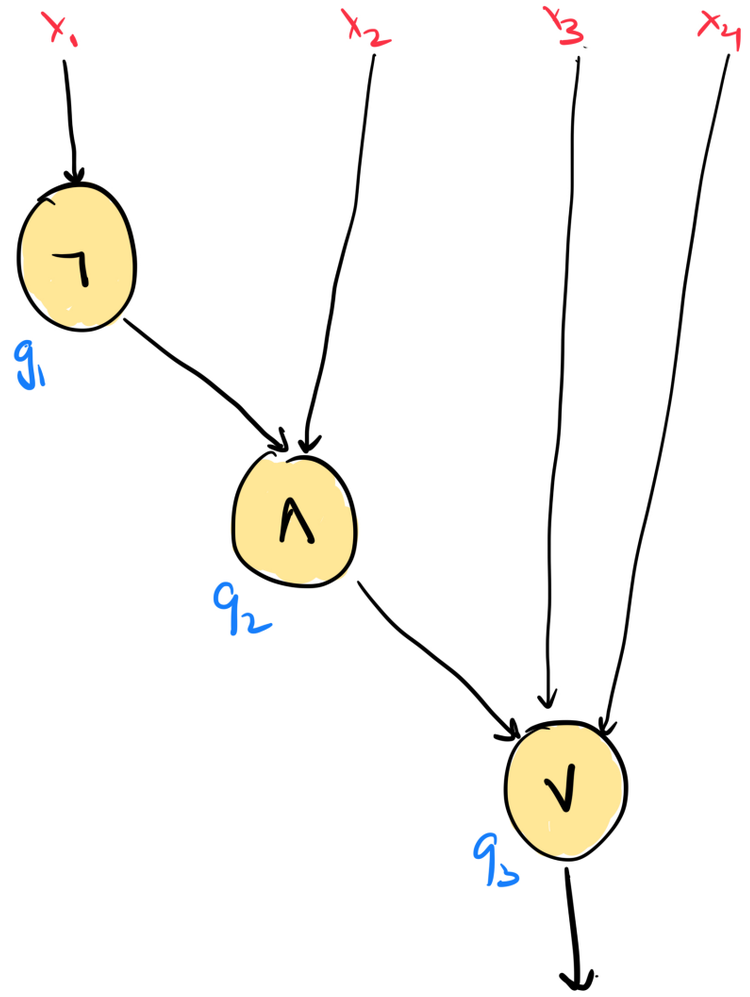
Languages and Boolean Circuits

- 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
  $L = \cup_{i \geq 1} f^{-1}(1)$.

- 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.



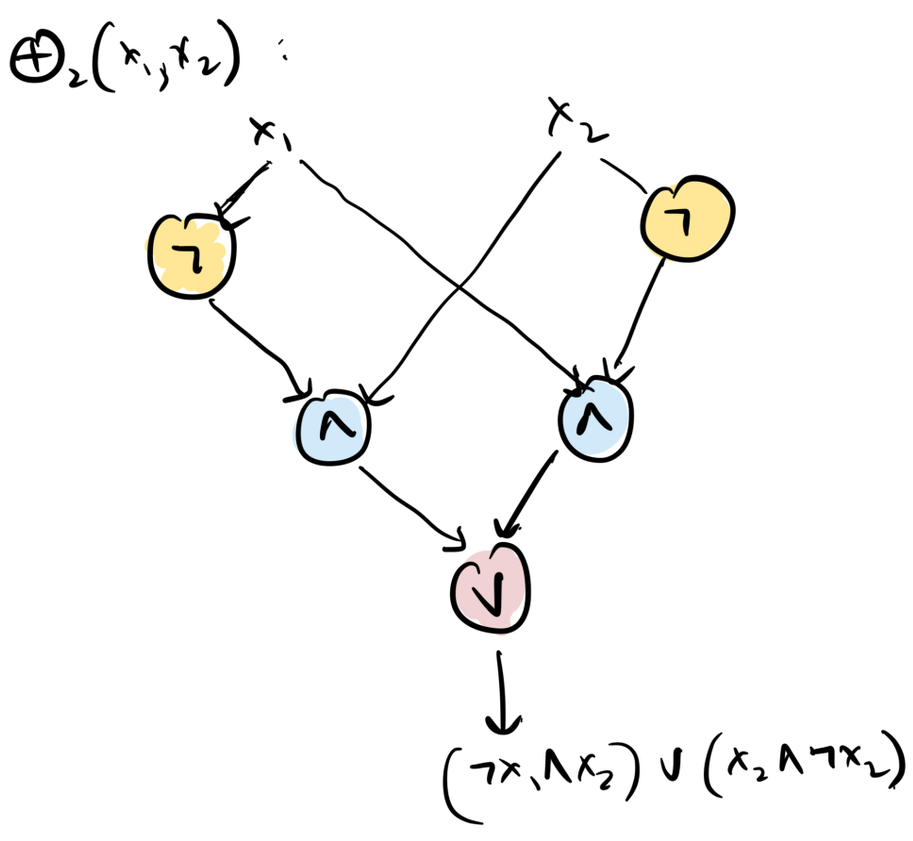
Boolean Circuit as a DAG

Examples: PARITY

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

Examples: PARITY

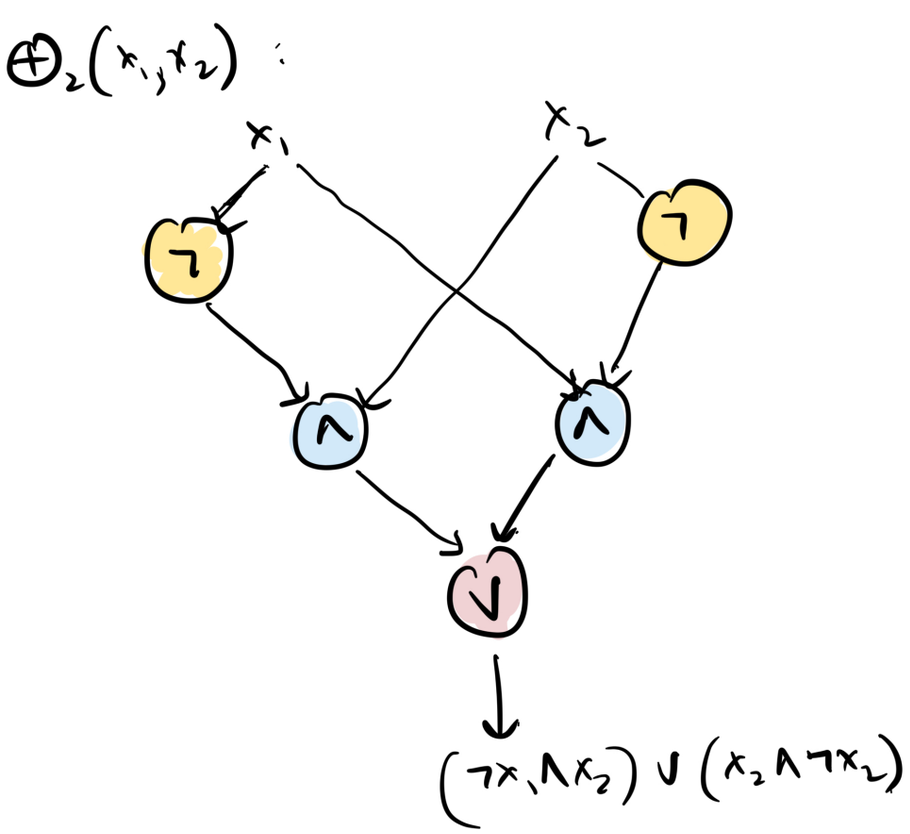
- The parity of a set of n bits is a function $\oplus_n : \{0,1\}^n \to \{0,1\}$ defined as $\oplus_n (x_1, \cdots, x_n) = \sum_{i=1}^{n} x_i \ (\mod 2)$.

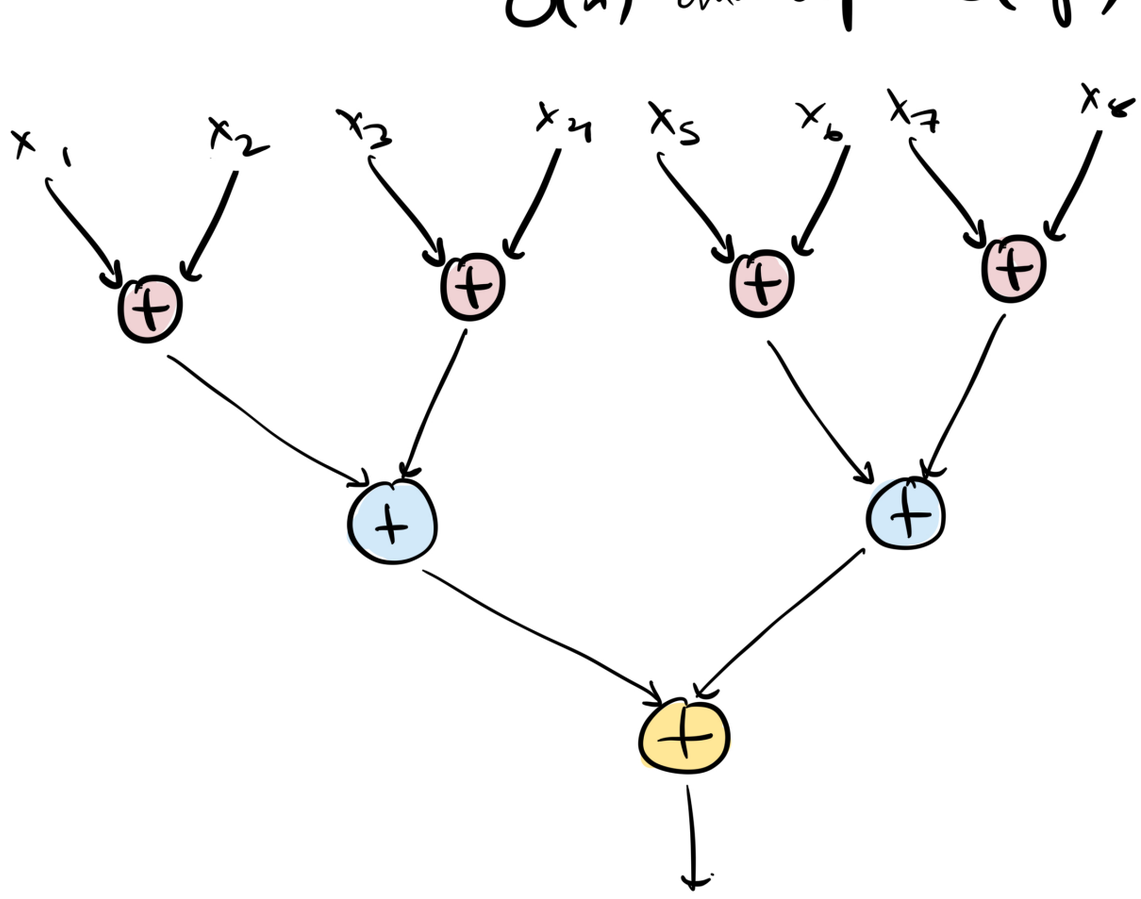


Examples: PARITY

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

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



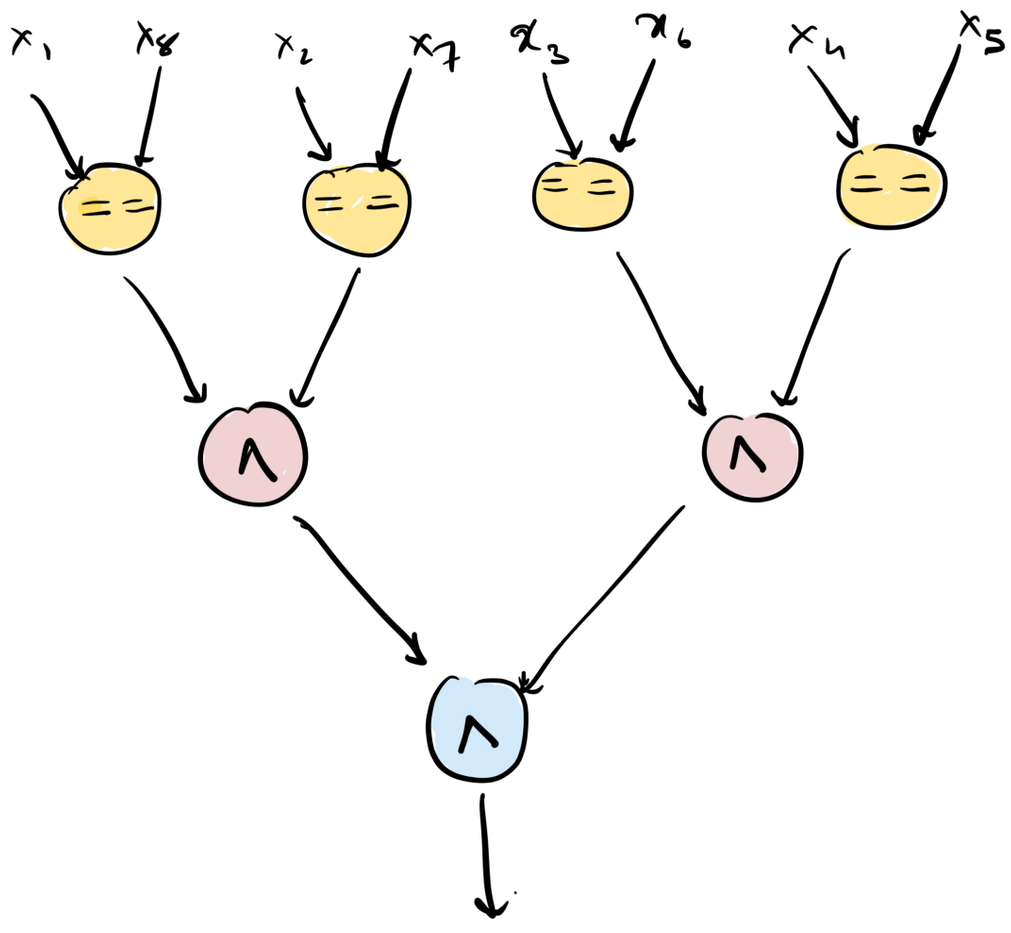


Examples: PALINDROME

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

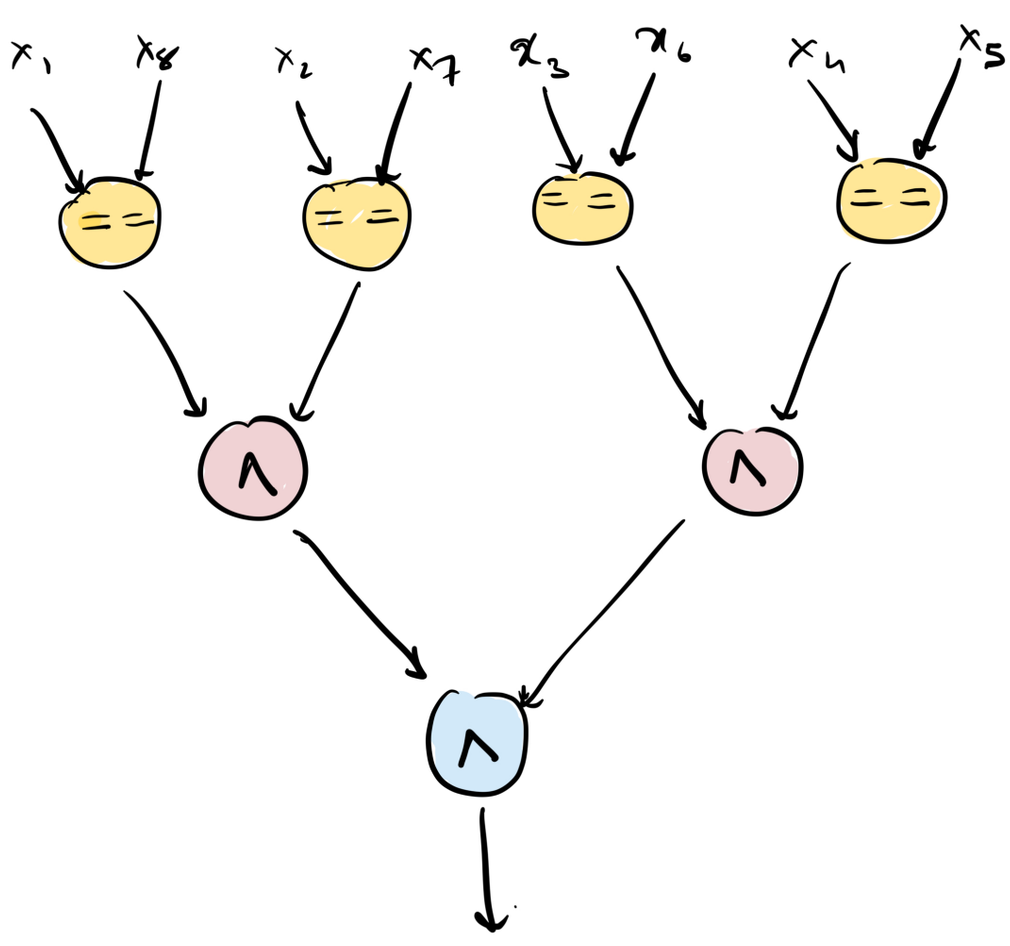
Examples: PALINDROME

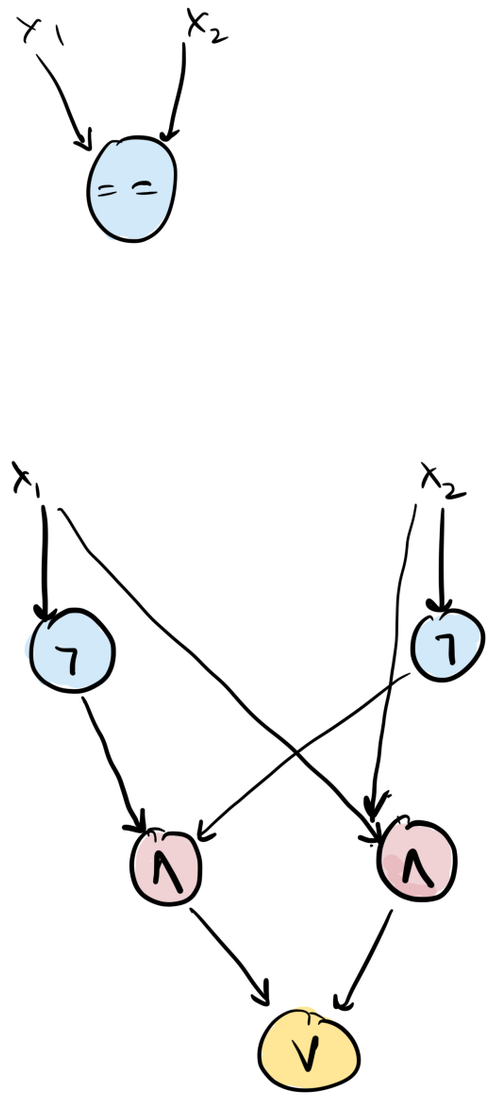
- Define $f : \{0,1\}^n \to \{0,1\}$ by $f(x) = 1 \iff x$ is a palindrome.



Examples: PALINDROME

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





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.

Shannon's Theorem

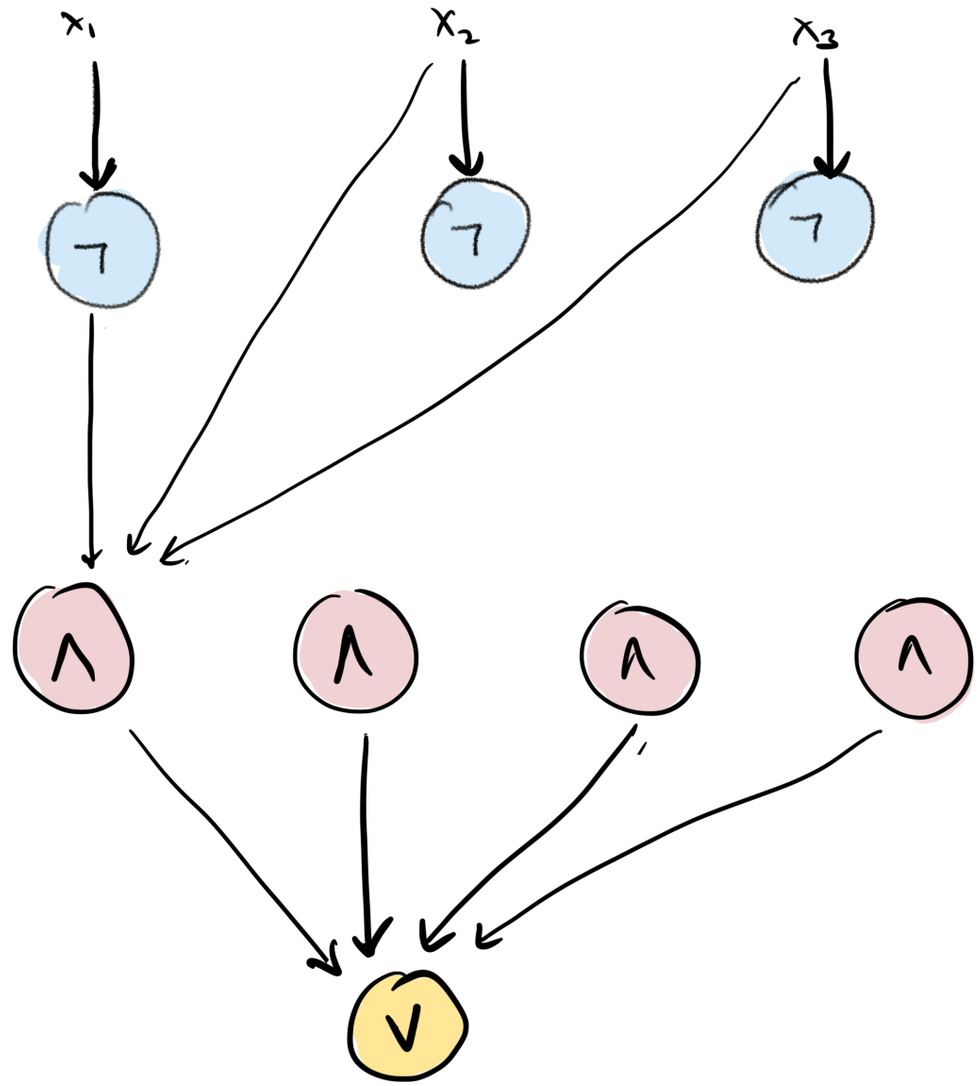
Theorem: For every function $f : \{0,1\}^n \to \{0,1\}$, there exists a Boolean circuit C of size at most $2^n + 1$ that computes f.

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

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.

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

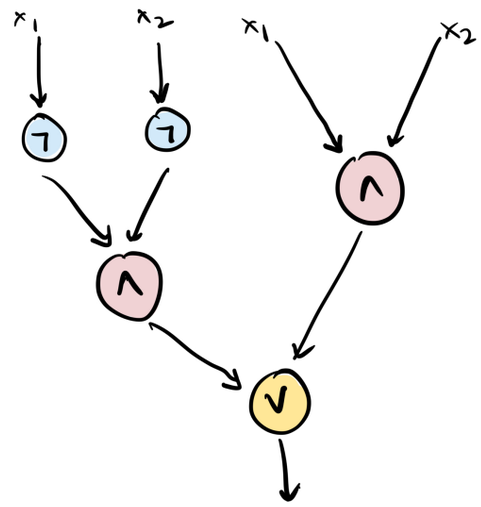
Corollary: For every language $L \subseteq \{0,1\}^*$, there exists a family of circuits $\{C_n\}_{n \geq 0}$ that computes L.

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).

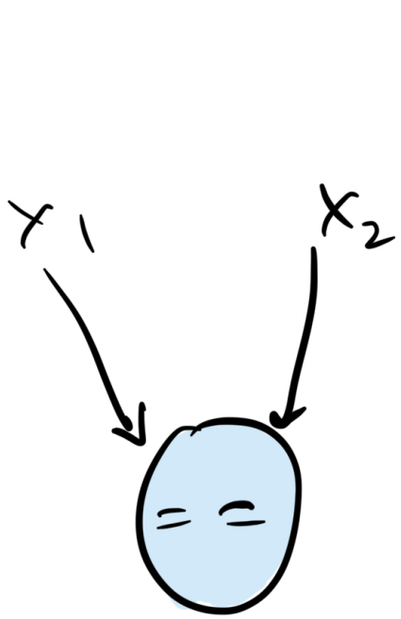
- A boolean formula has bounded fan-in when each of $\wedge$, $\vee$ gates have exactly two inputs.

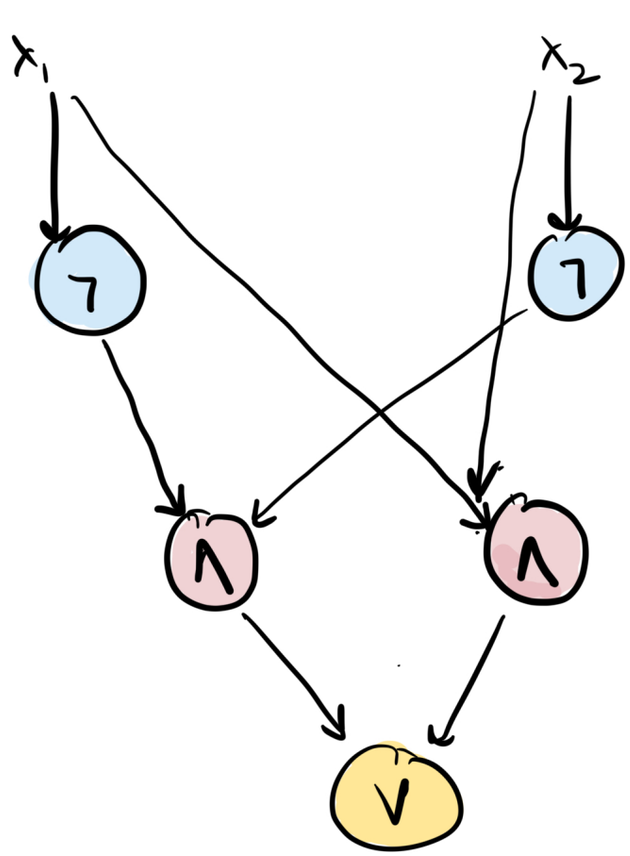
- Represented as a rooted binary tree, with leaves being multiple copies of x_i, $\neg x_i$.

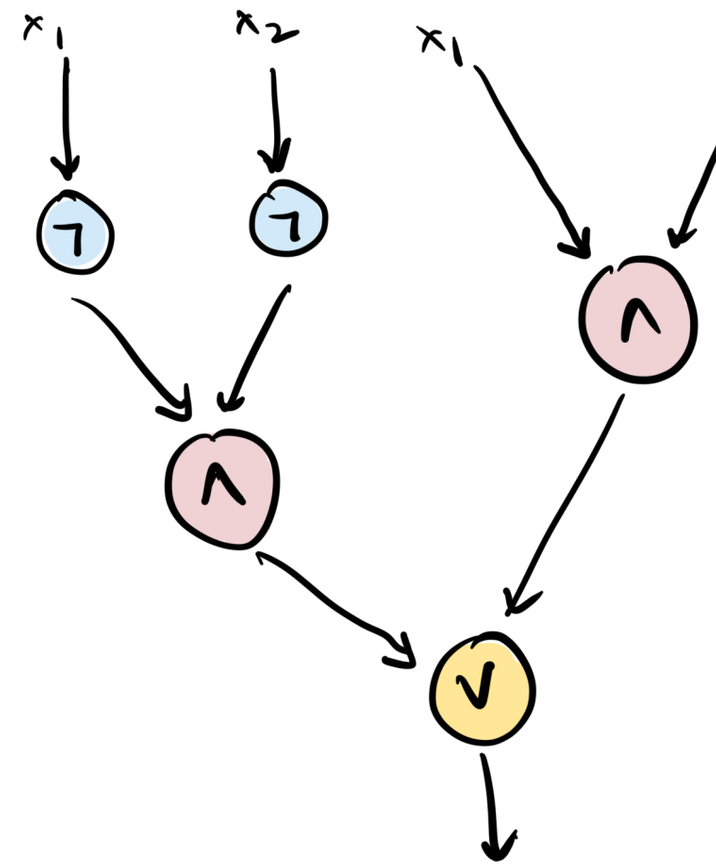


Complexity Measure: Boolean Formula

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







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:

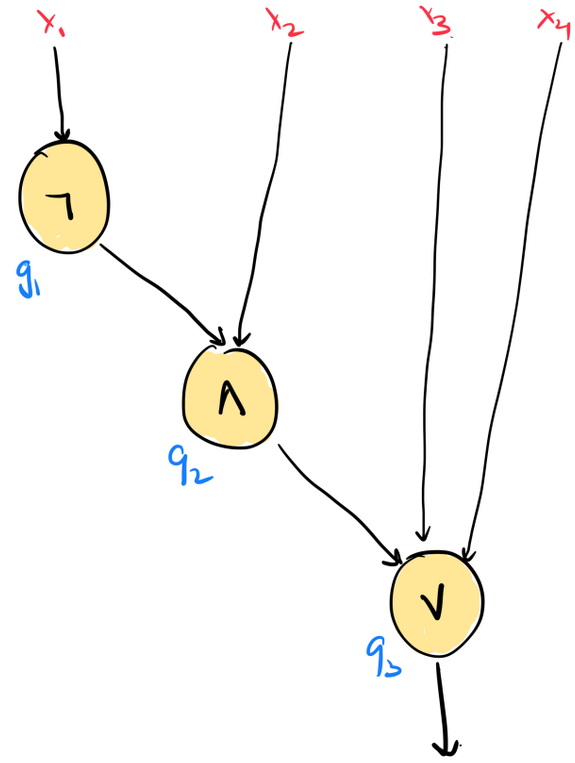
$$f(x) = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$$

Every Function has a CNF Formula

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.

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

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

	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

Circuit Value Problem

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

Lemma: CVP $\in P$

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

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

(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.