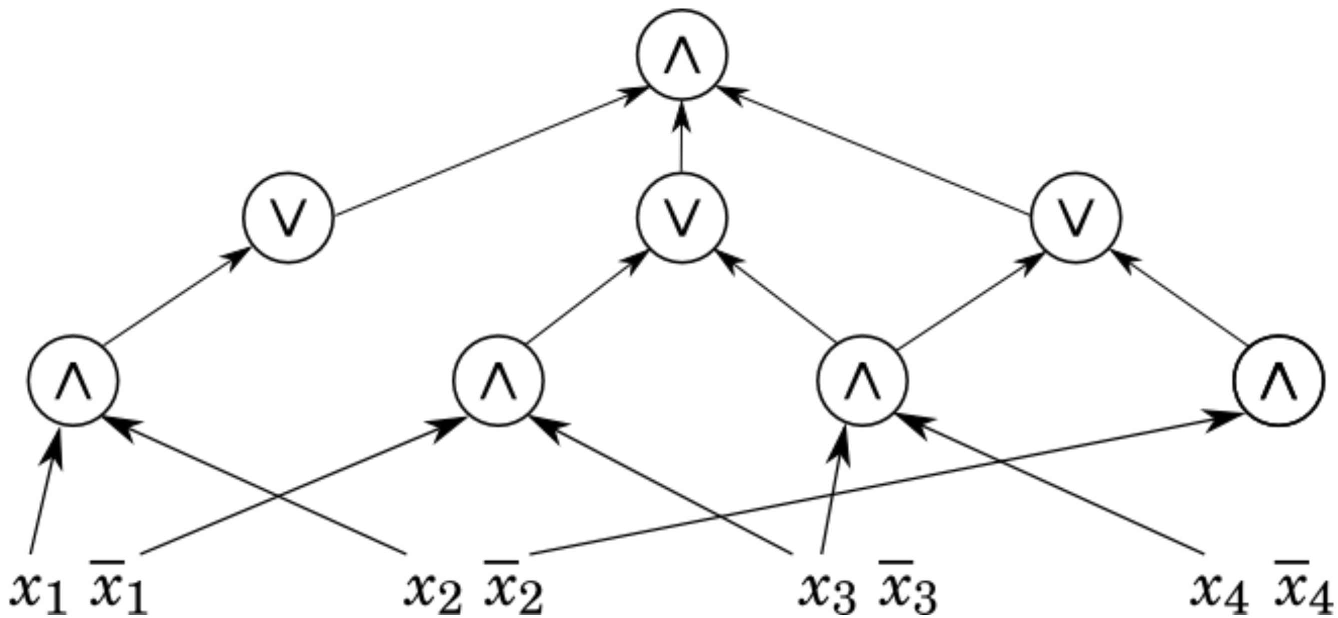
15-251
Great Theoretical Ideas in Computer Science

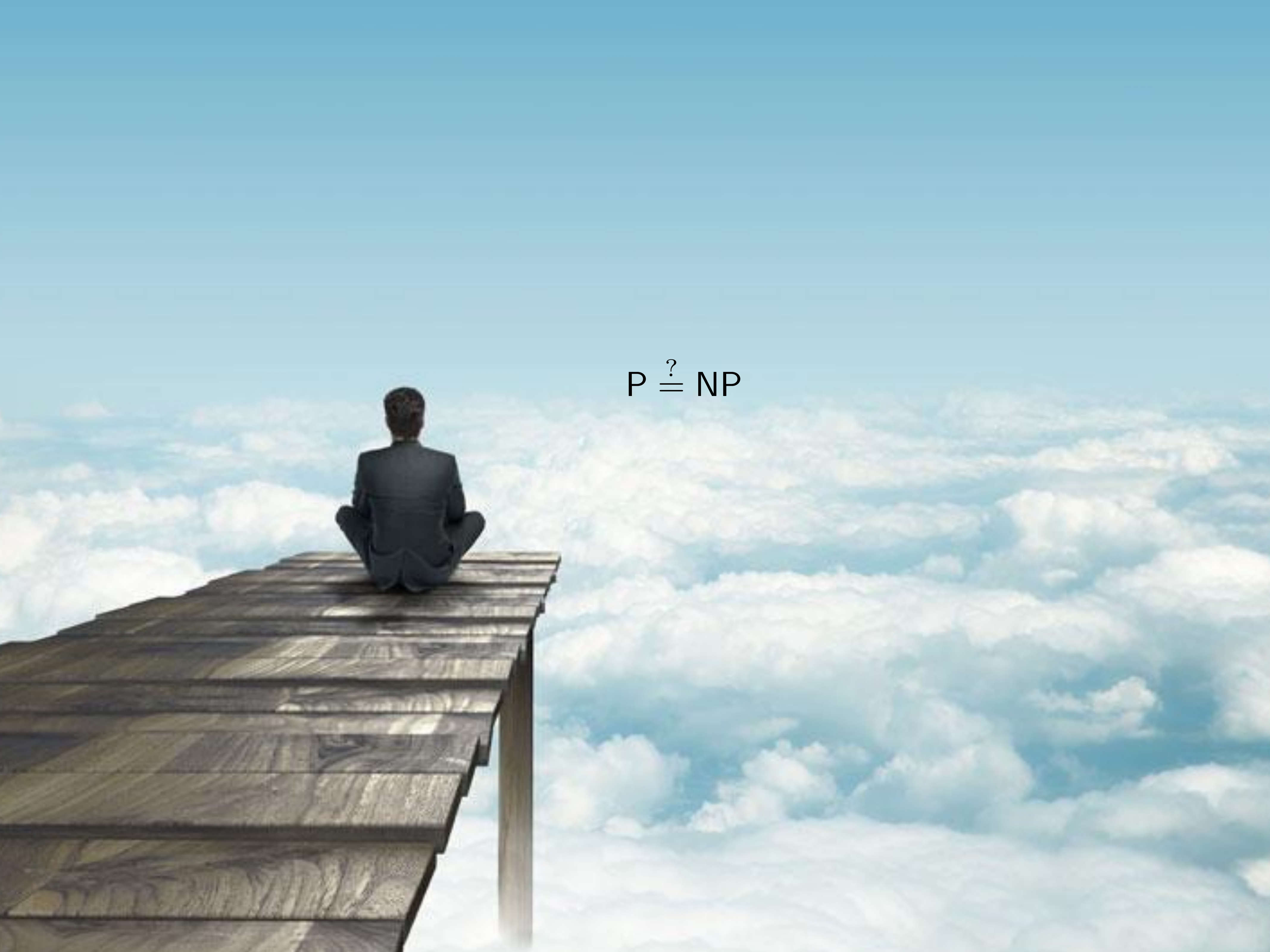
Lecture 12:
Boolean Circuits



October 6th, 2015

Where we are, where we are going

Monday	Tuesday	Wednesday	Thursday	Friday
Aug 29	Aug 30 ***Introduction***	Aug 31 ***On proofs***	Sep 1 ***Combinatorial Games***	Sep 2
Sep 5	Sep 6 ***Finite Automata***	Sep 7 hw1 w.s.	Sep 8 ***Turing Machines***	Sep 9
Sep 12	Sep 13 ***Uncountability***	Sep 14 hw2 w.s.	Sep 15 ***Undecidability***	Sep 16
Sep 19	Sep 20 ***Intro to Complexity 1***	Sep 21 hw3 w.s.	Sep 22 ***Intro to Complexity 2***	Sep 23
Sep 26	Sep 27 ***Graphs 1***	Sep 28 hw4 w.s.	Sep 29 ***Graphs 2***	Sep 30
Oct 3	Oct 4 ***Graphs 3***	Oct 5 Midterm 1	Oct 6 ***Boolean Circuits***	Oct 7
Oct 10	Oct 11 ***NP-completeness 1***	Oct 12 hw5 w.s.	Oct 13 ***NP-completeness 2***	Oct 14



$$\mathsf{P} \stackrel{?}{=} \mathsf{NP}$$

Computational complexity of a **problem**

How to show an upper bound on the intrinsic complexity?

> Give an algorithm that solves the problem.

How to show a lower bound on the intrinsic complexity?

> Argue against **<u>all</u>** possible algorithms that solve the problem.

<u>The dream</u>: Get a matching upper and lower bound.

What is P ?

P

The set of languages that can be decided in $O(n^k)$ steps for some constant k.

The theoretical divide between efficient and inefficient:

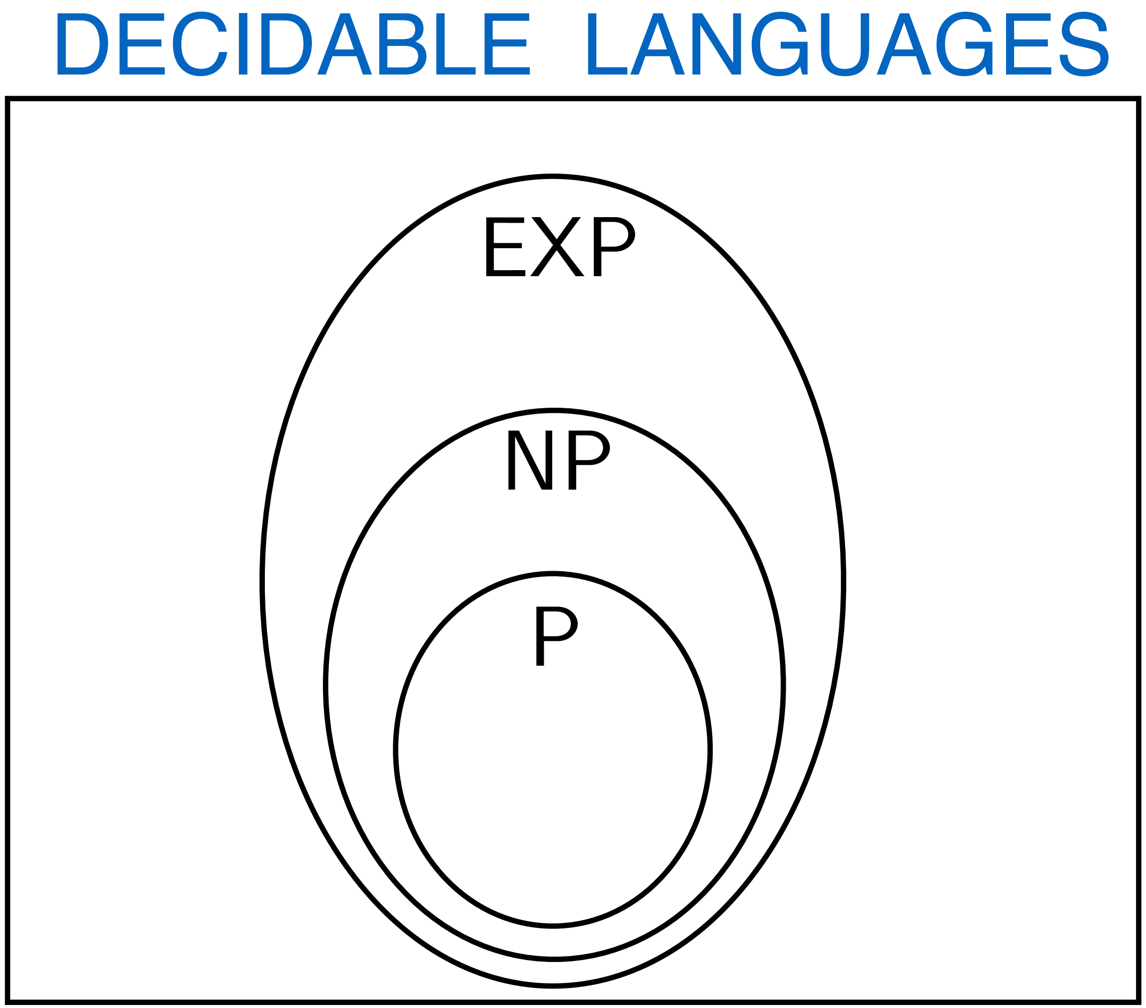
$L \in$ P $\longrightarrow$ efficiently solvable.

$L \notin$ P $\longrightarrow$ not efficiently solvable.

What is NP ?

EXP

The set of languages that can be decided in $O(2^{n^k})$ steps for some constant $k > 0$.

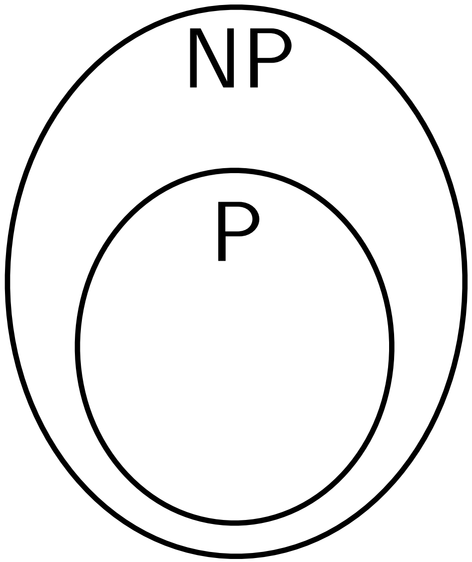


NP:

A class between

P and EXP.

What is NP ?



$$\mathrm{P} \stackrel{?}{=} \mathrm{NP}$$

asks whether these two sets are equal.

How would you show $\mathrm{P} = \mathrm{NP}$?

> Show that every problem in NP can be solved in poly-time.

How would you show $\mathrm{P} \neq \mathrm{NP}$?

> Show that there is a problem in NP which cannot be solved in poly-time.

You have to argue against all possible poly-time TMs.

Boolean Circuits

Some preliminary questions

What is a Boolean circuit?

- It is a computational model for computing decision problems (or computational problems).

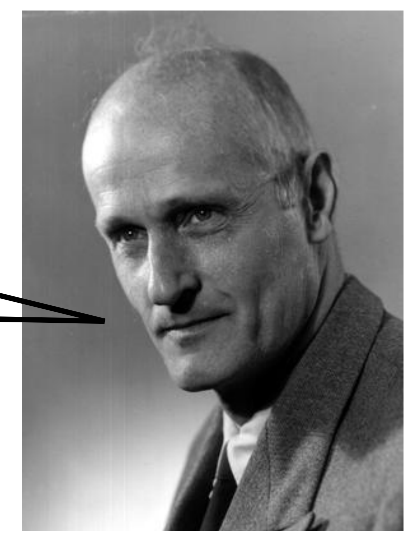
We already have TMs. Why Boolean circuits?

- The definition is simpler.
- Easier to understand, usually easier to reason about.
- Boolean circuits can efficiently simulate TMs.
  (efficient decider TM $\Longrightarrow$ efficient/small circuits.)
- Circuits are good models to study *parallel computation*.
- Real computers are built with digital circuits.

Sounds AWESOME!
So why didn't we just learn about circuits first?

There is a small catch.

An algorithm is a finite answer
to infinite number of questions.



Stephen Kleene
(1909 - 1994)

Sounds AWESOME!
So why didn't we just learn about circuits first?

There is a small catch.

Circuits are an infinite answer to infinite number of questions.

Anil Ada
(???? - 2077)

Dividing a problem according to length of input

$$\Sigma = \{0, 1\}$$

$$L \subseteq \{0, 1\}^*$$

$$f : \{0, 1\}^* \to \{0, 1\}$$

$\{0, 1\}^n$ = all strings of length *n*

$$L_n = \{w \in L : |w| = n\}$$

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

for $x \in \{0, 1\}^n$,

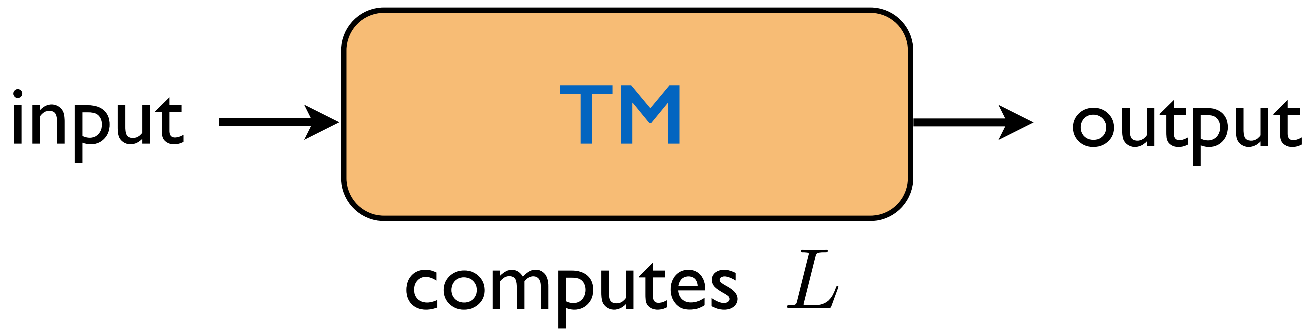
$$f^n(x) = f(x)$$

$$L = L_0 \cup L_1 \cup L_2 \cup \cdots$$

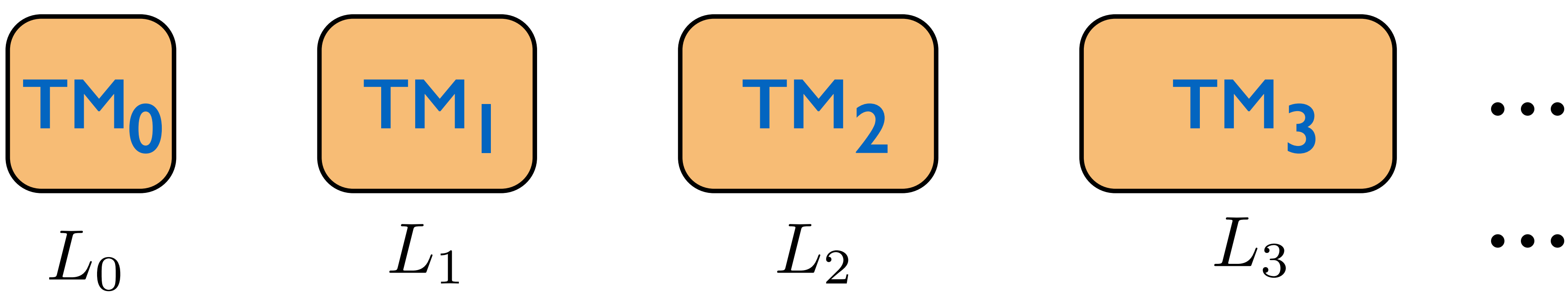
$$f = (f^0, f^1, f^2, \ldots)$$

Dividing a problem according to length of input

A TM is a finite object (finite number of states)
but can handle any input length.



Imagine a model where we allow the TM to grow with input length.



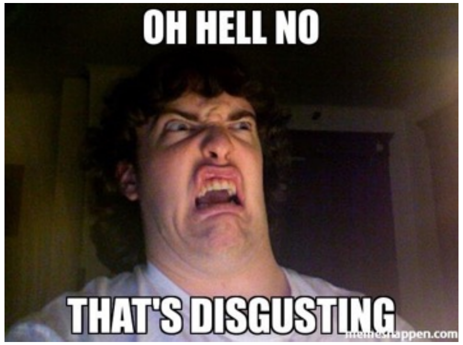
Dividing a problem according to length of input

So one machine does not compute L.

You use a family of machines:

$$(M_0, M_1, M_2, \ldots)$$

(Imagine having a different Python function for each input length.)

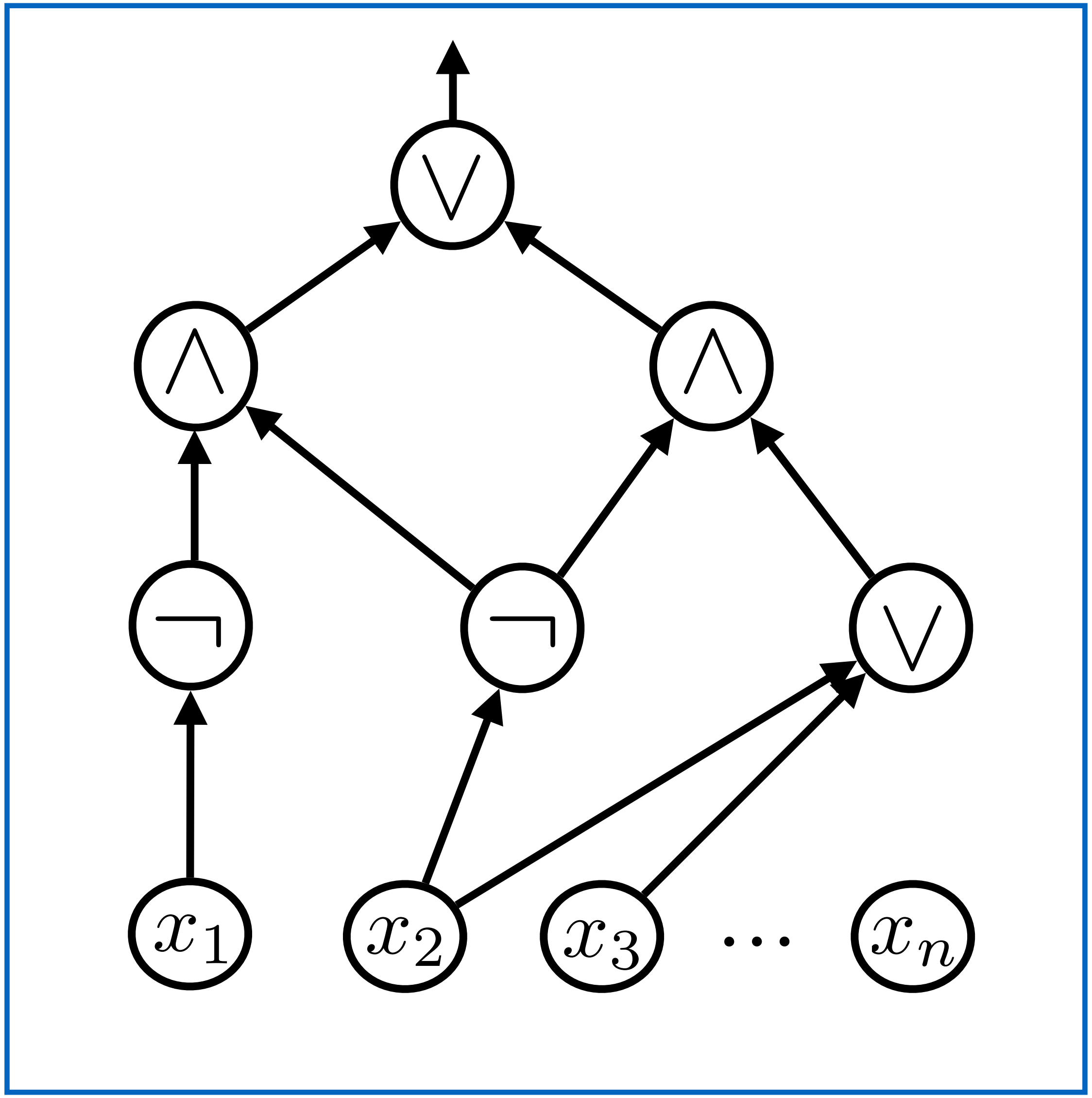


Is this a reasonable/realistic model of computation?!?

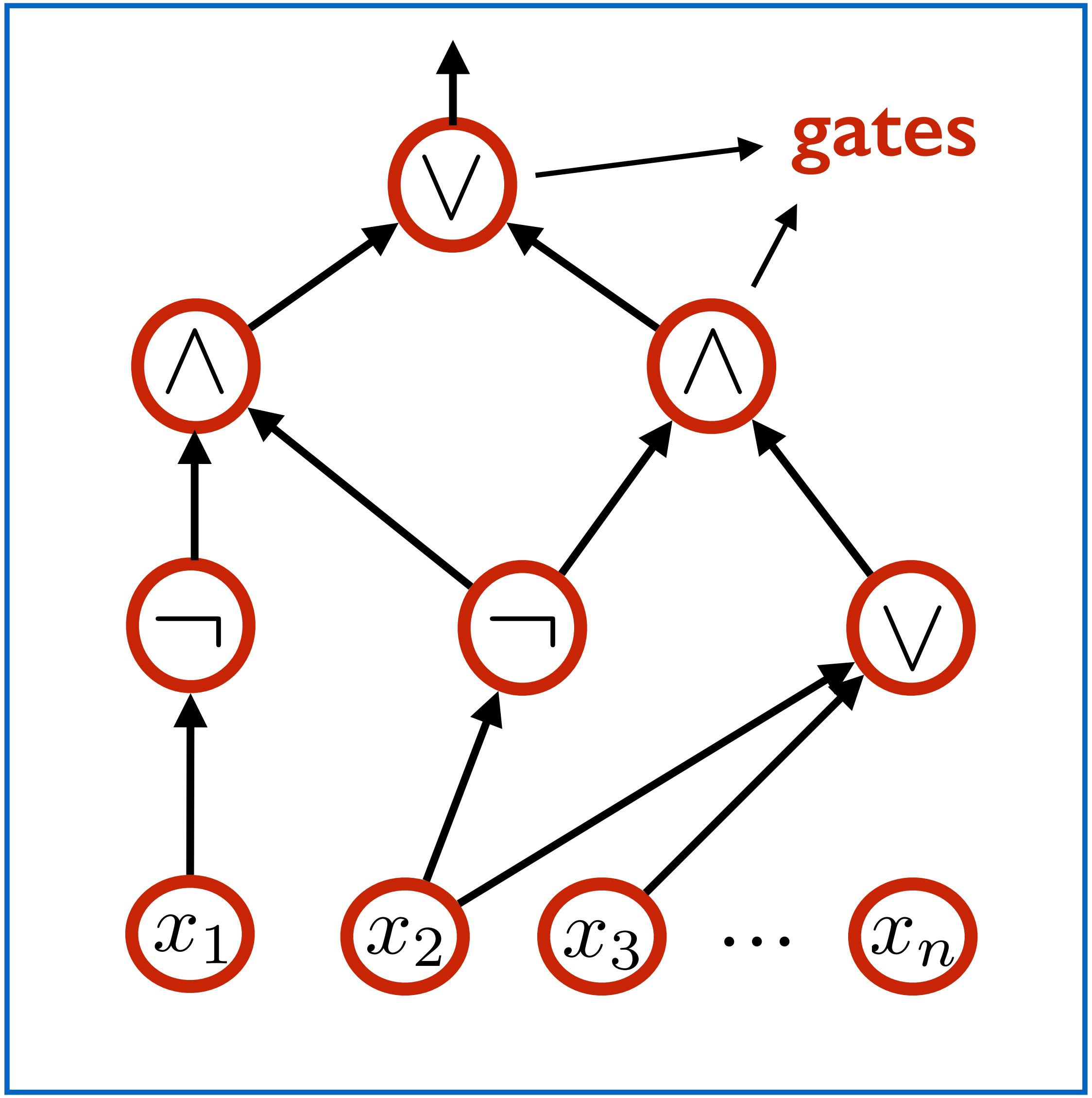
Boolean circuits work this way.
Need a separate circuit for each input length.

Boolean Circuit Definition

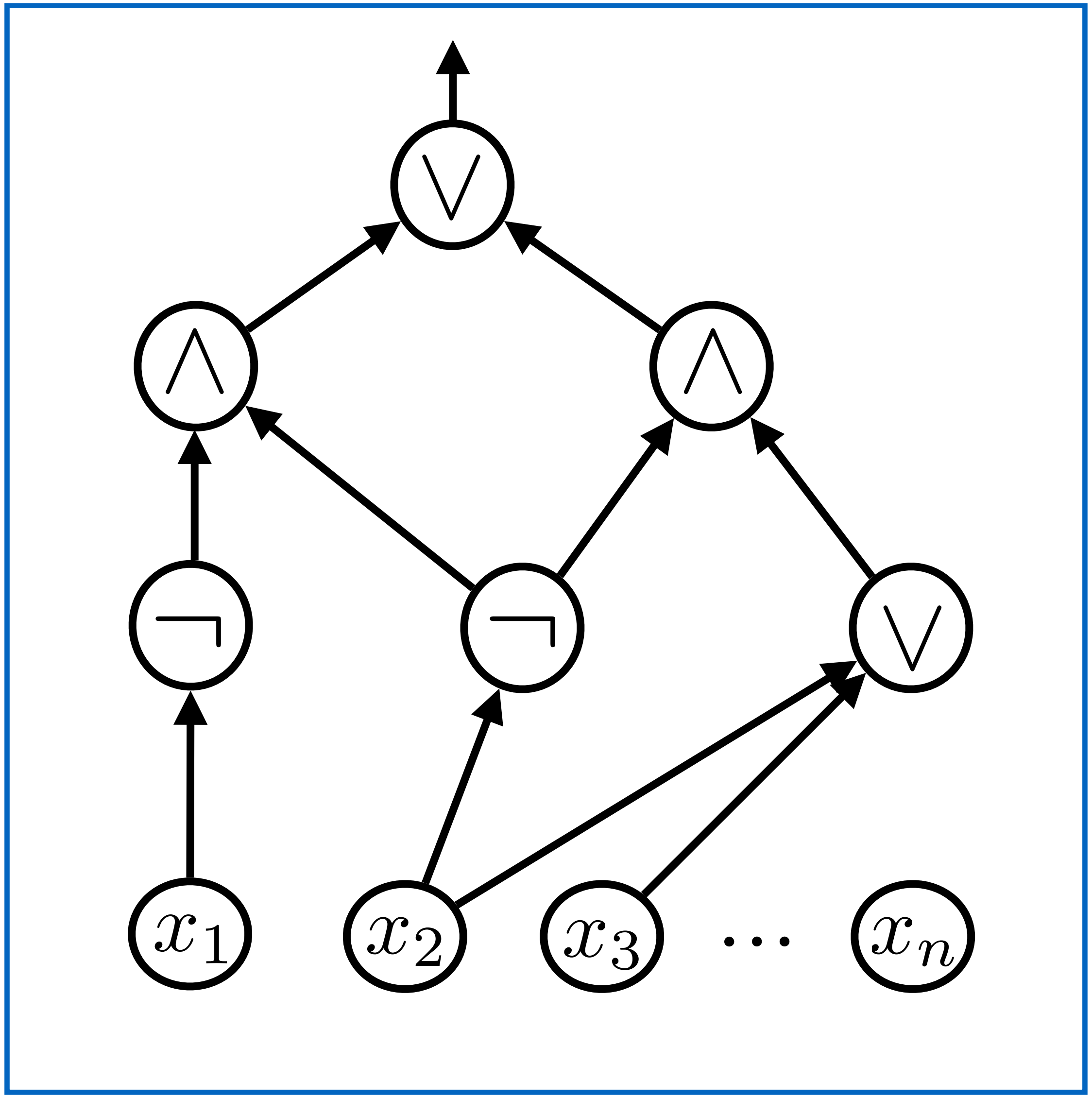
Picture of a circuit



Picture of a circuit



Picture of a circuit



$\vee$ binary OR gate

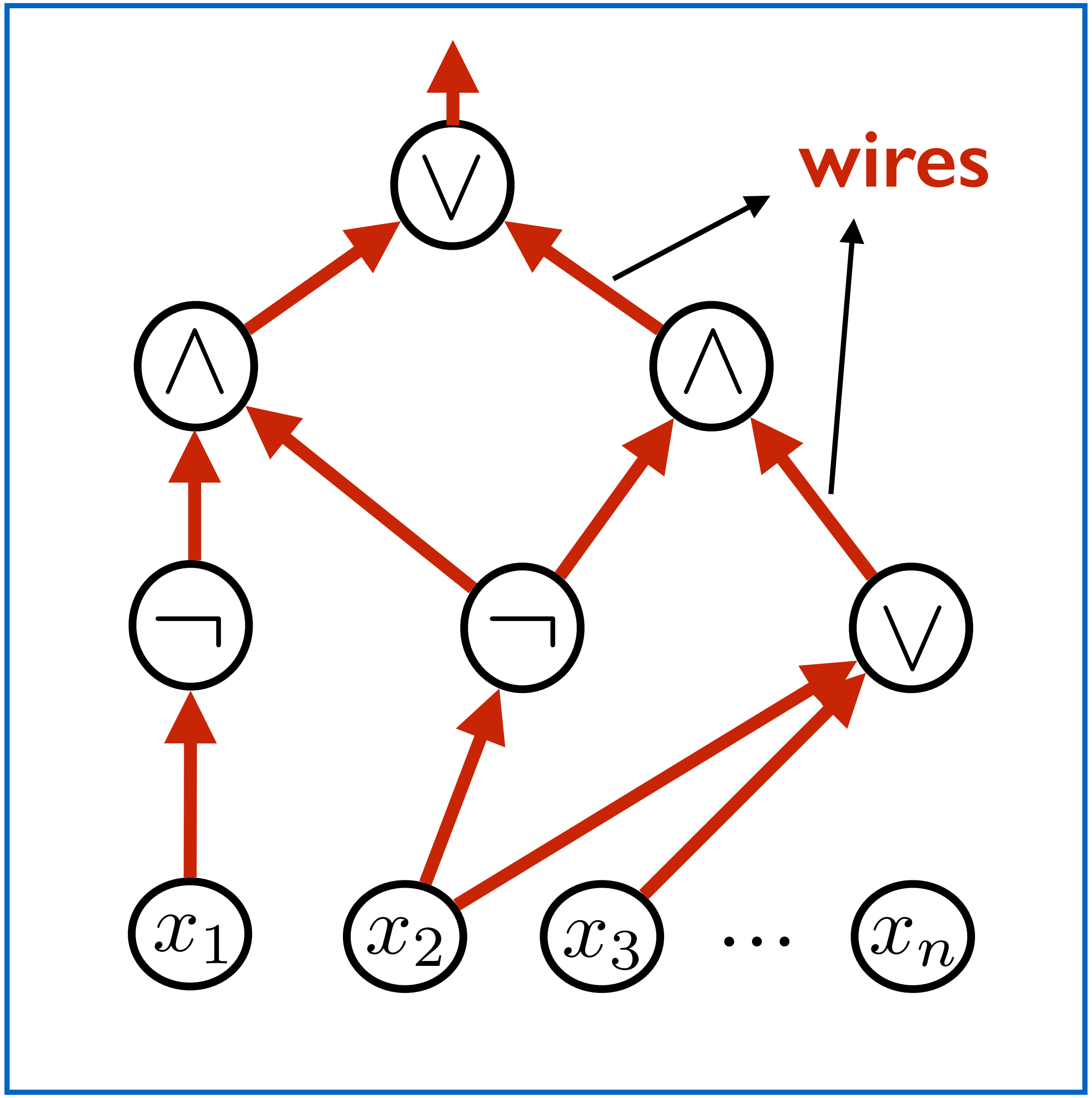
$\wedge$ binary AND gate

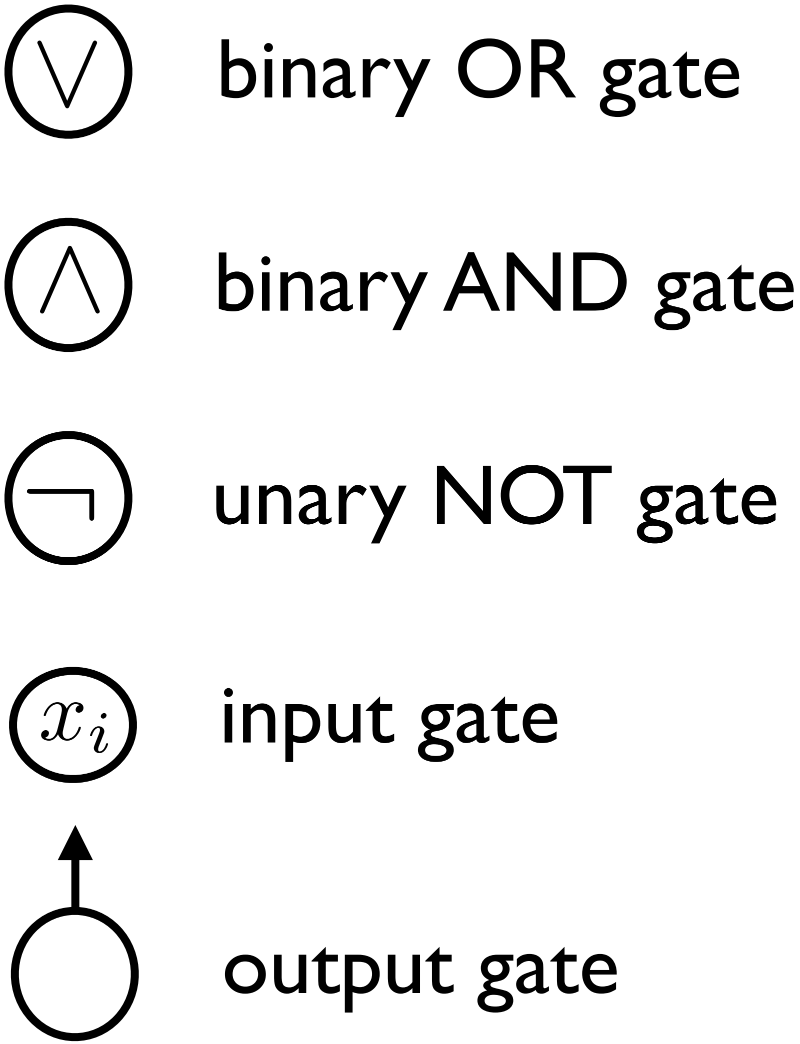
$\neg$ unary NOT gate

x_i input gate

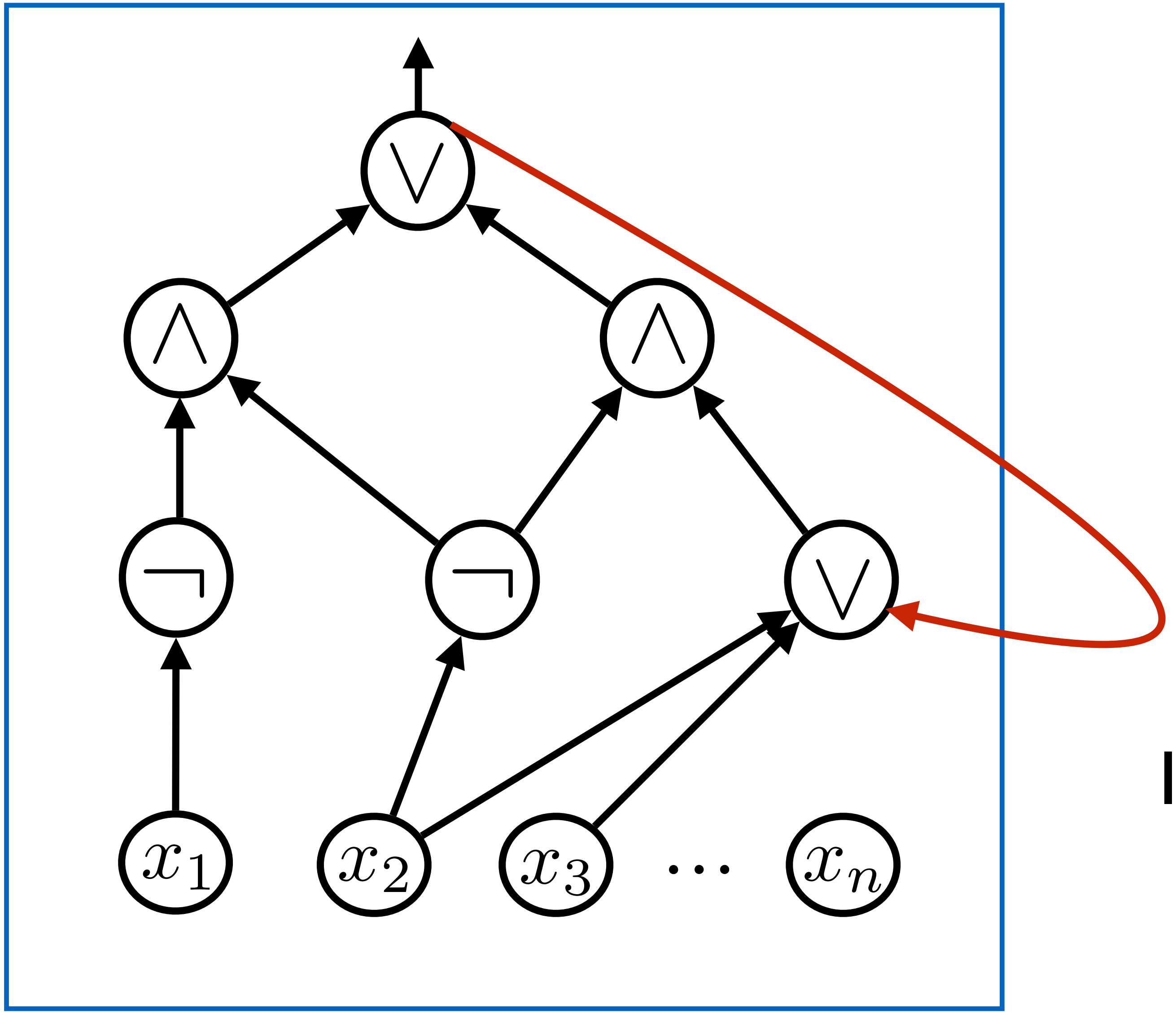
output gate

Picture of a circuit





Picture of a circuit

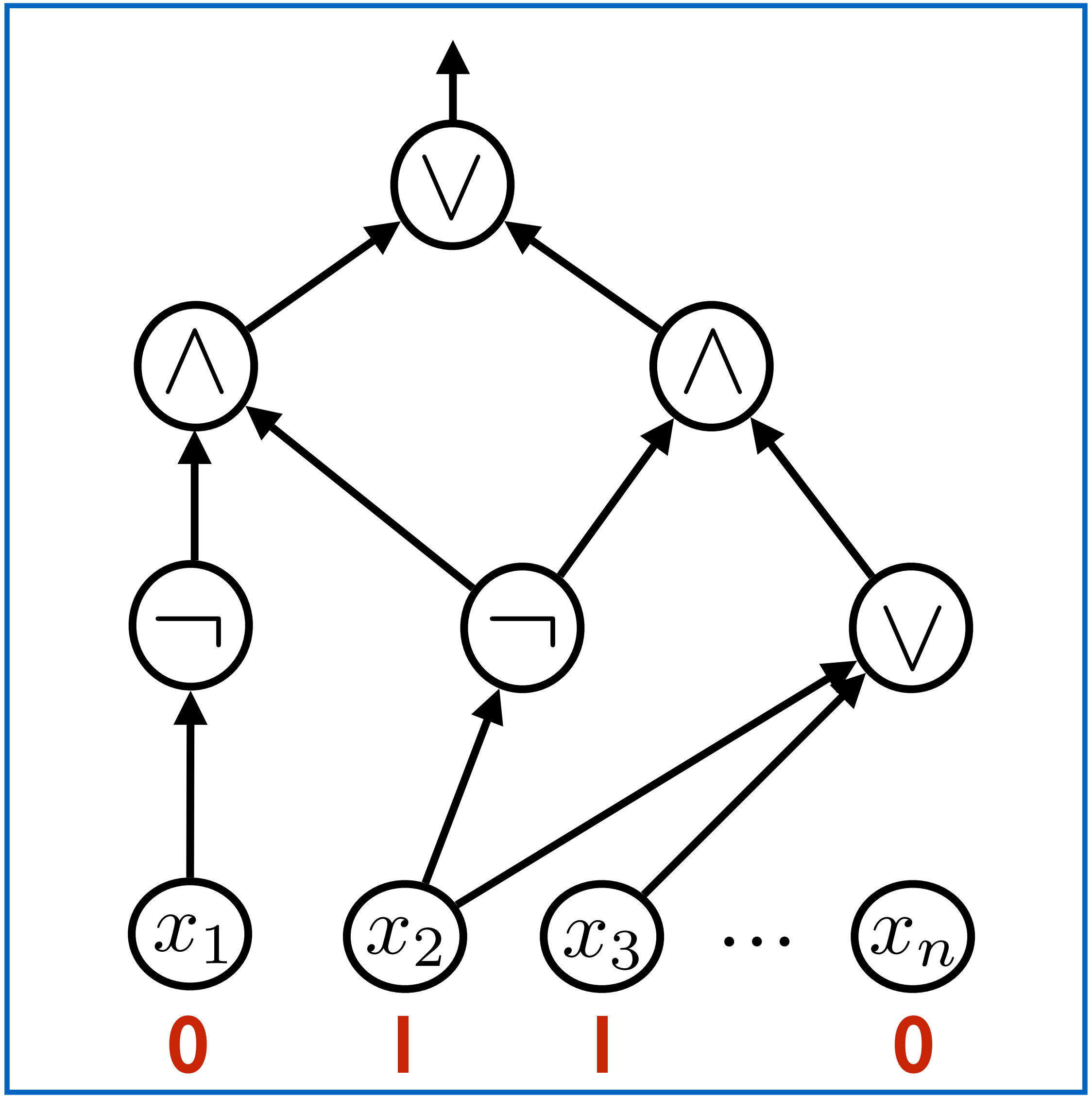


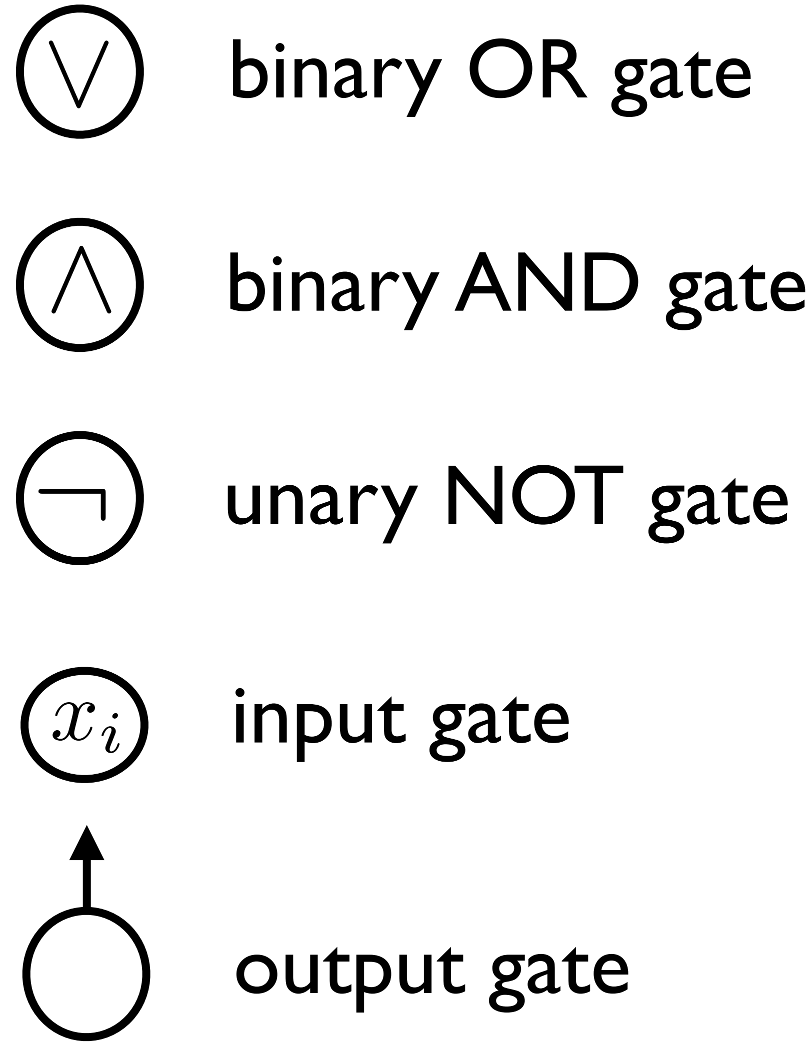
No feedback loops allowed!

It is a directed acyclic graph.

Information flows from input gates to the output gate.

Picture of a circuit

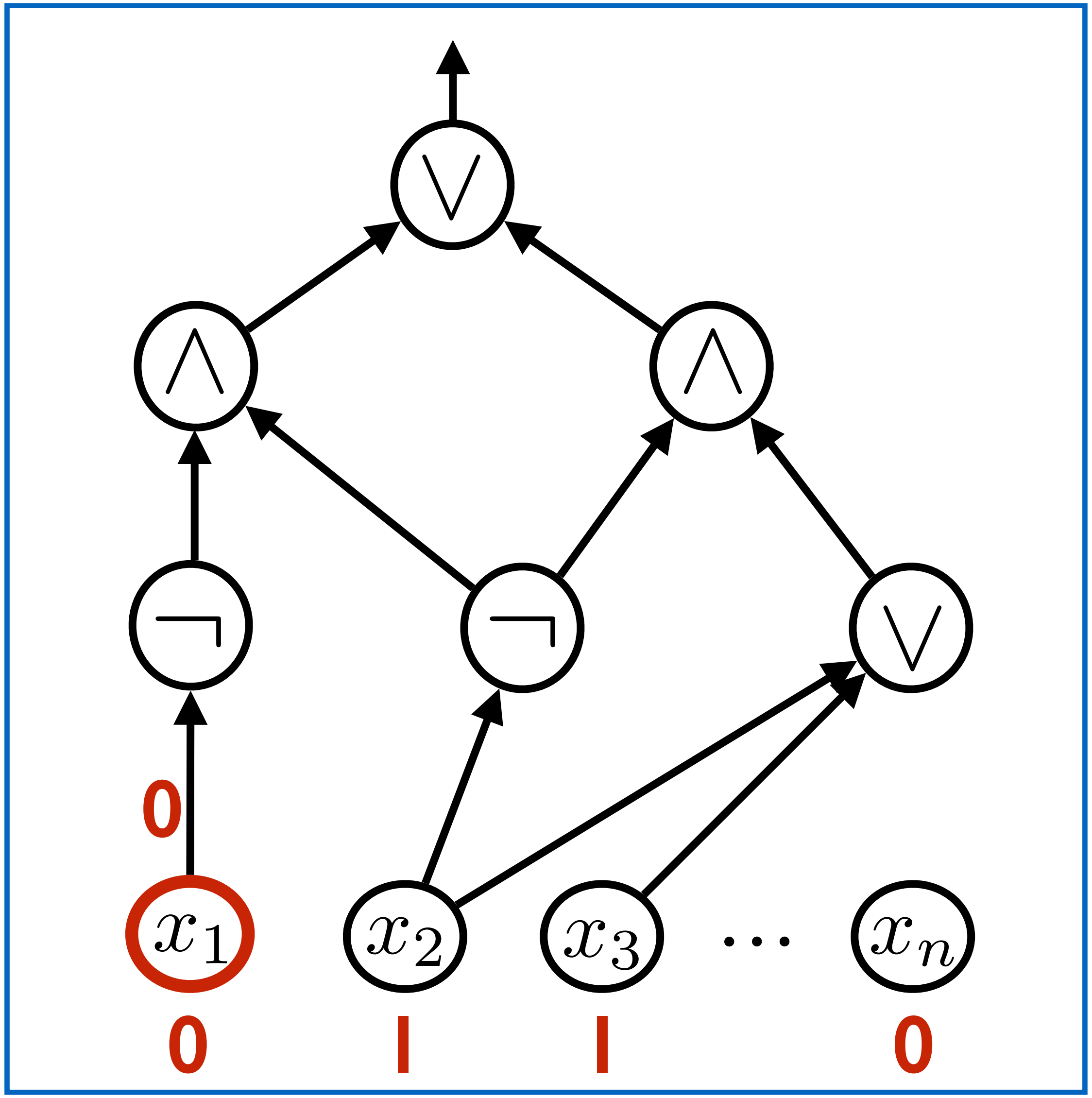


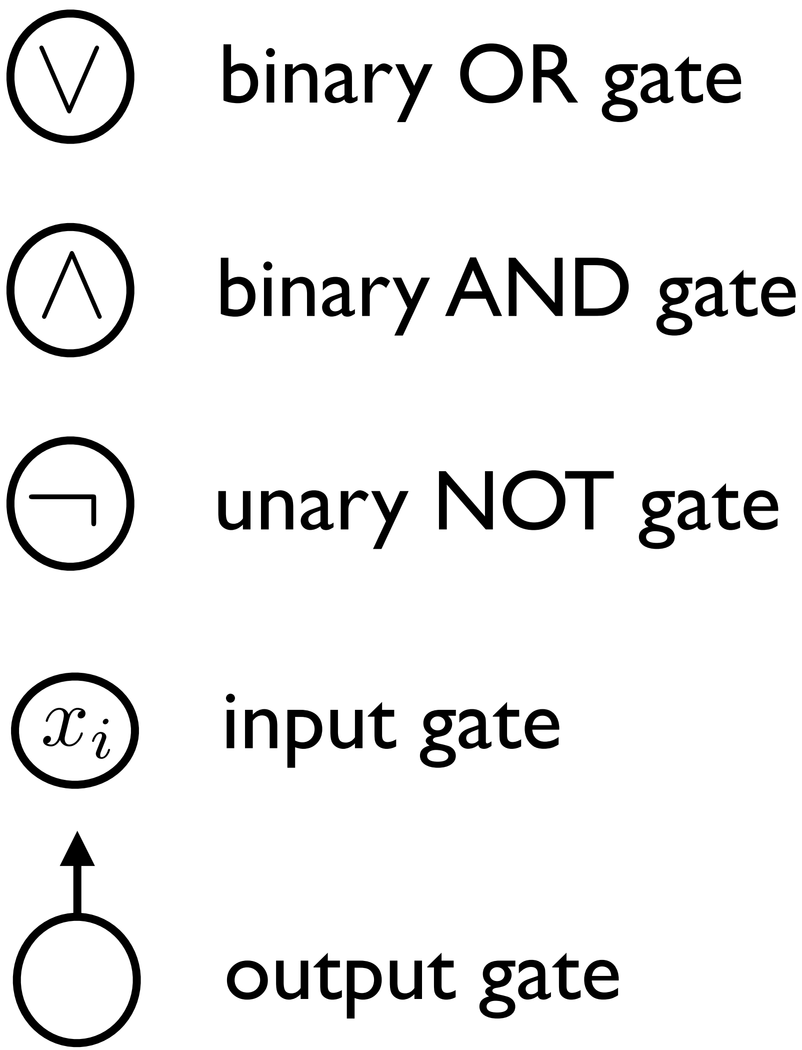


Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit

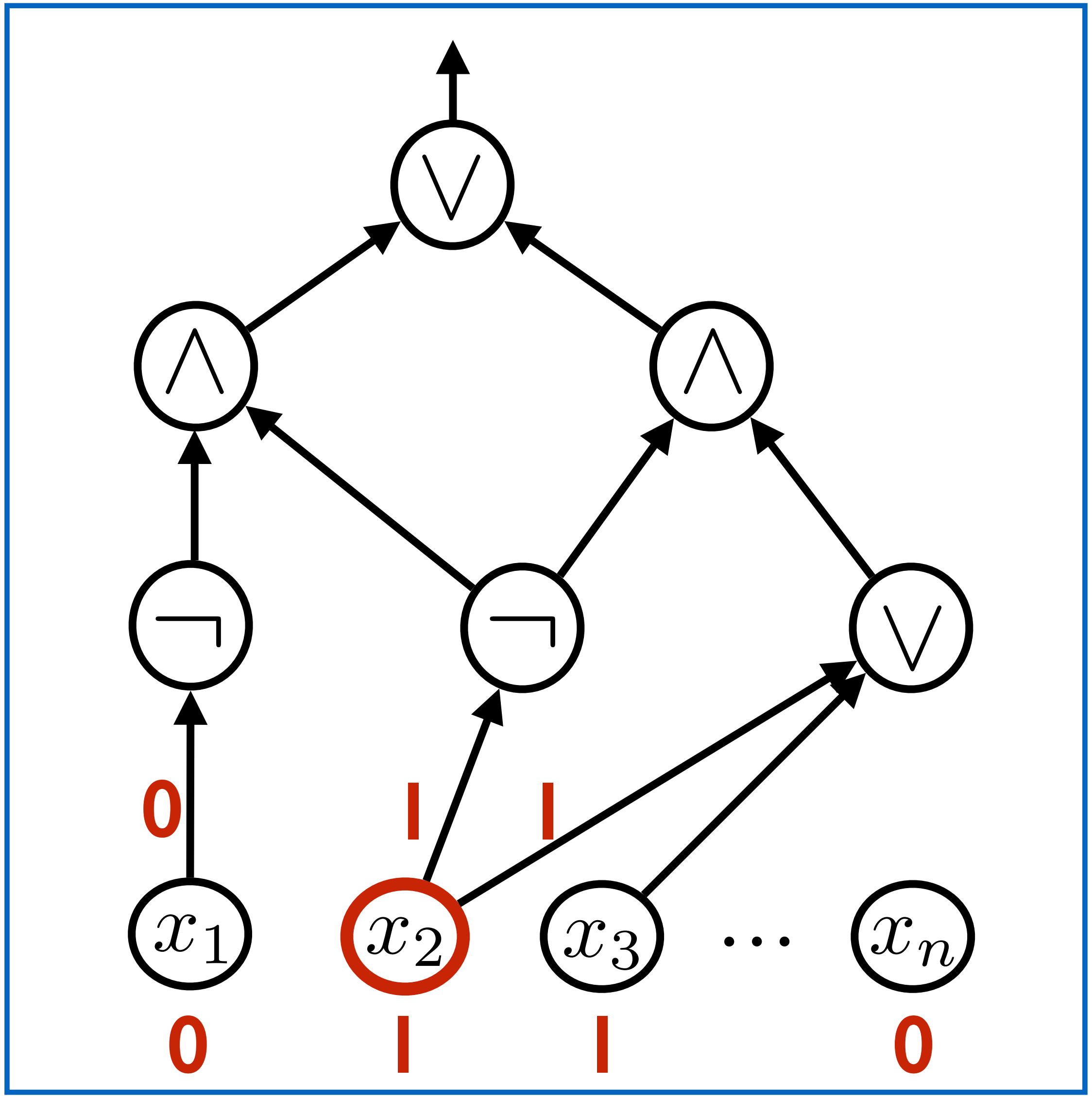


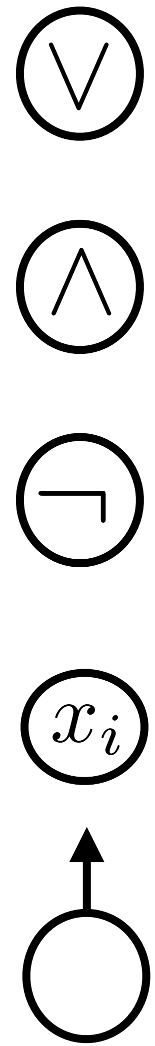


Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit





binary OR gate

binary AND gate

unary NOT gate

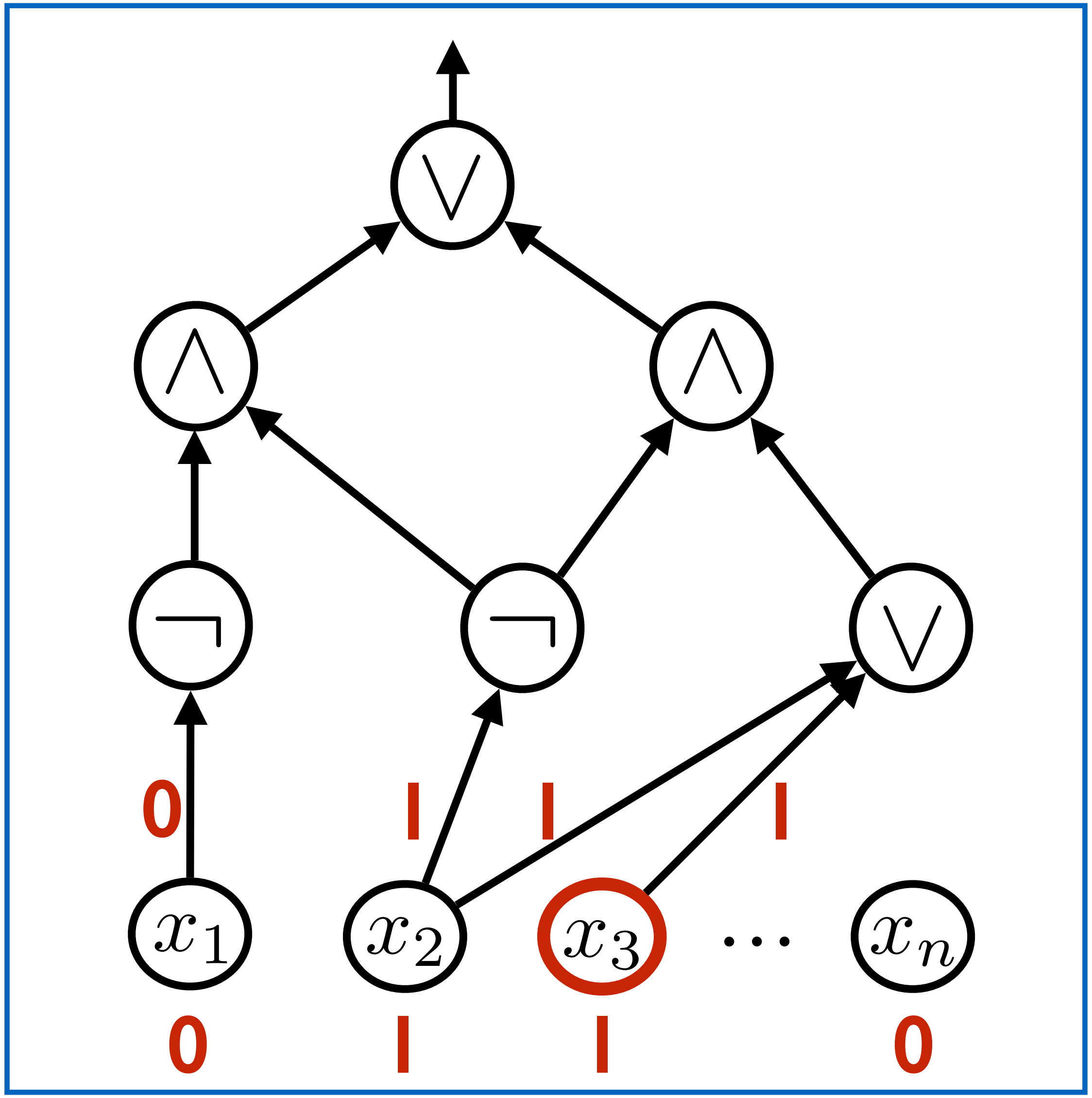
input gate

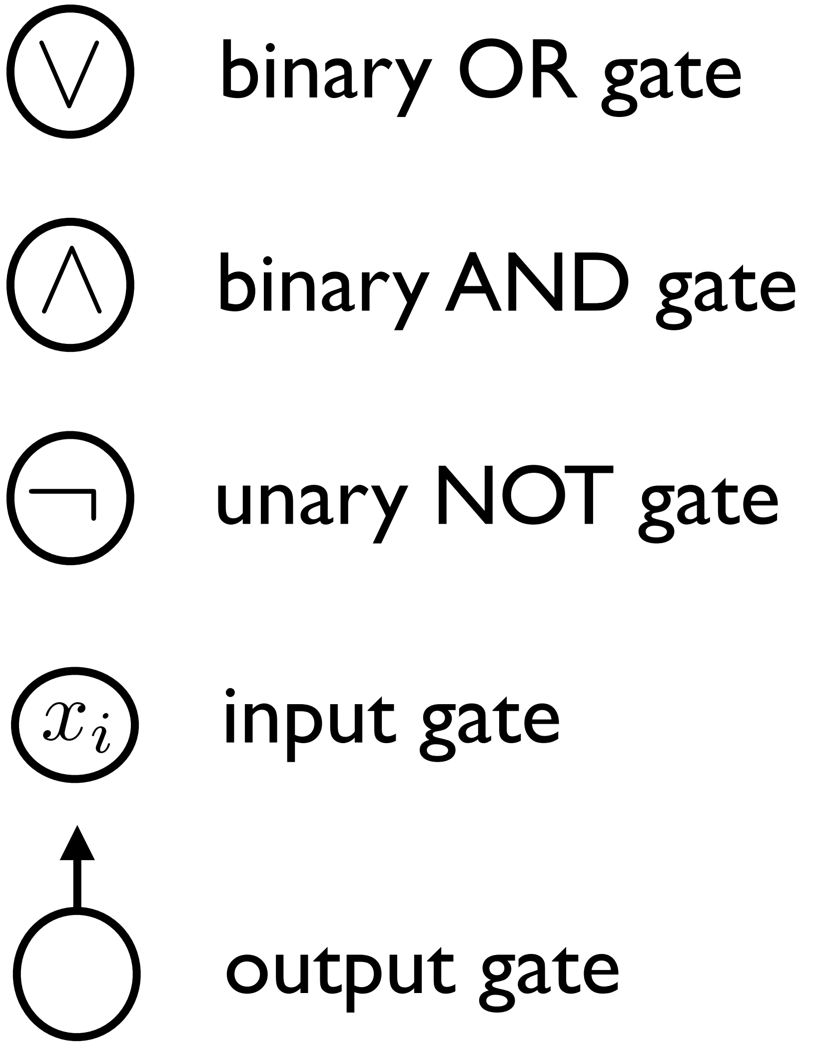
output gate

Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit

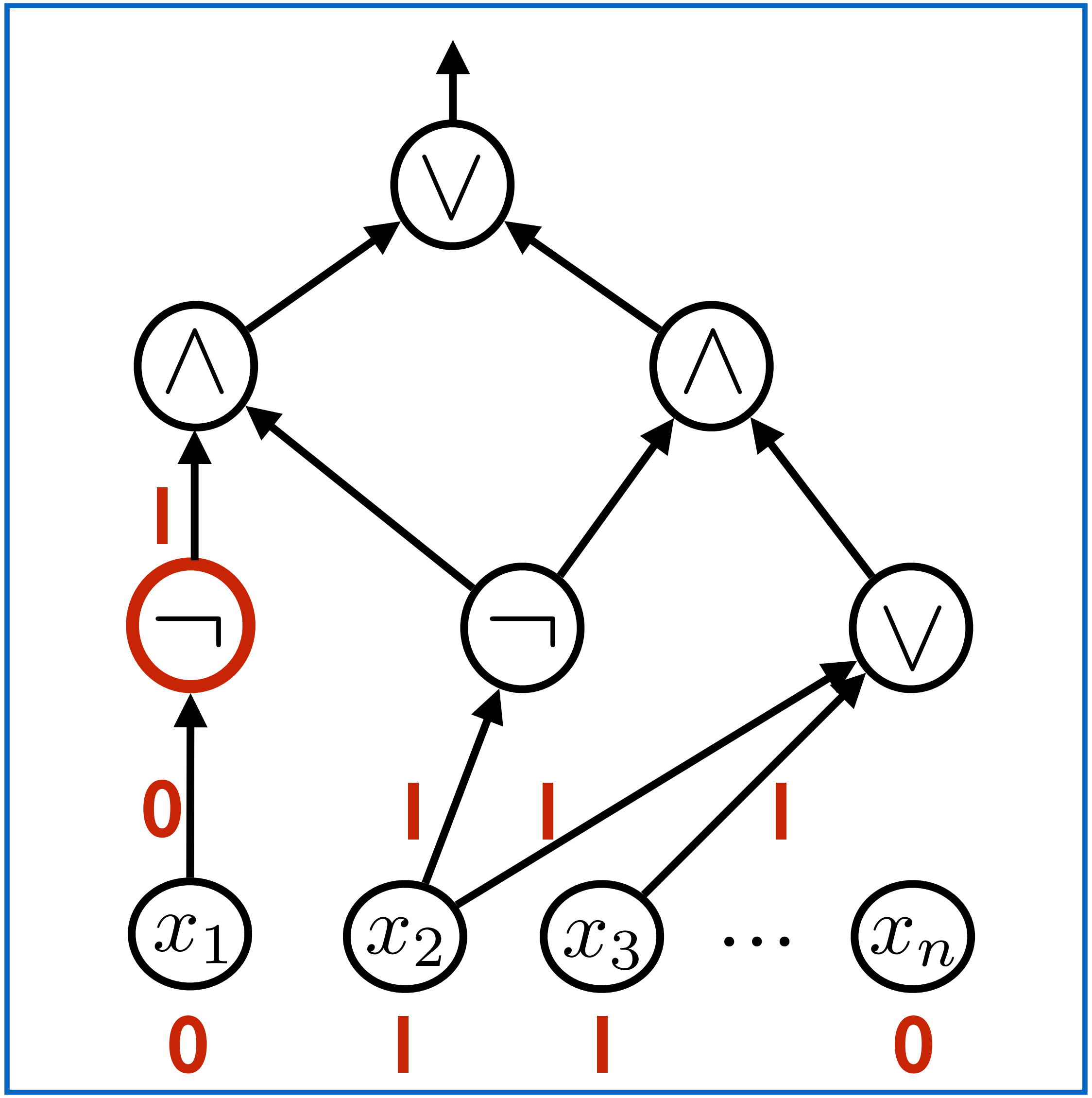


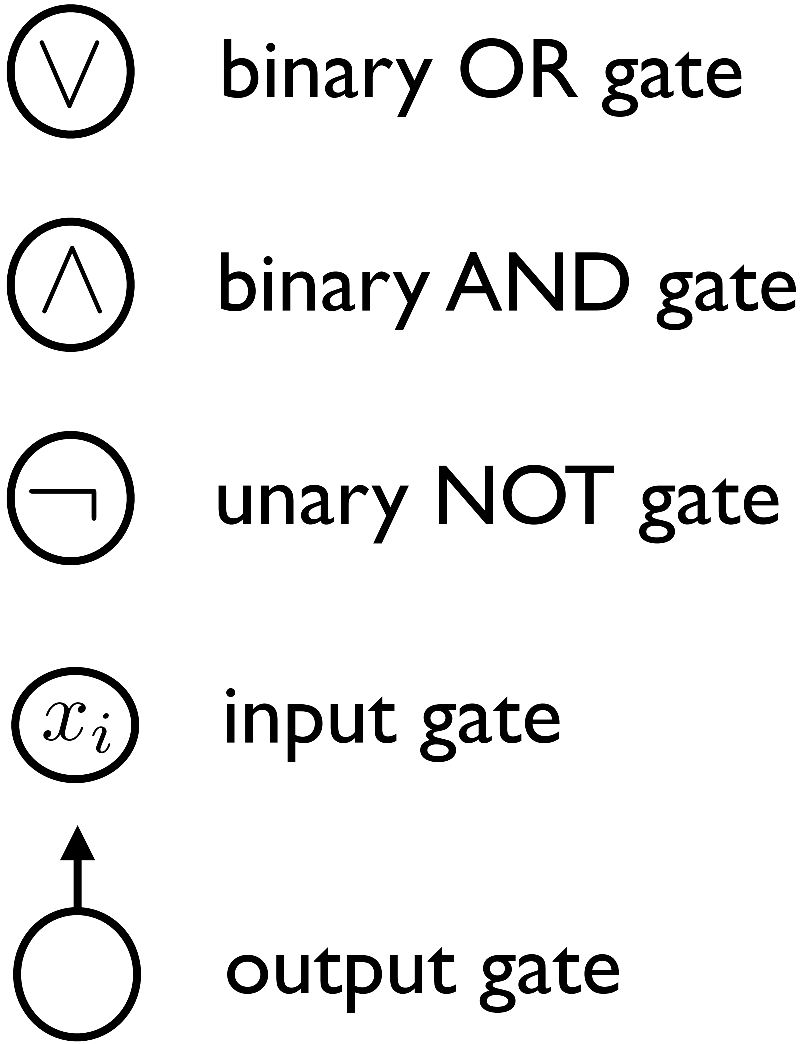


Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit

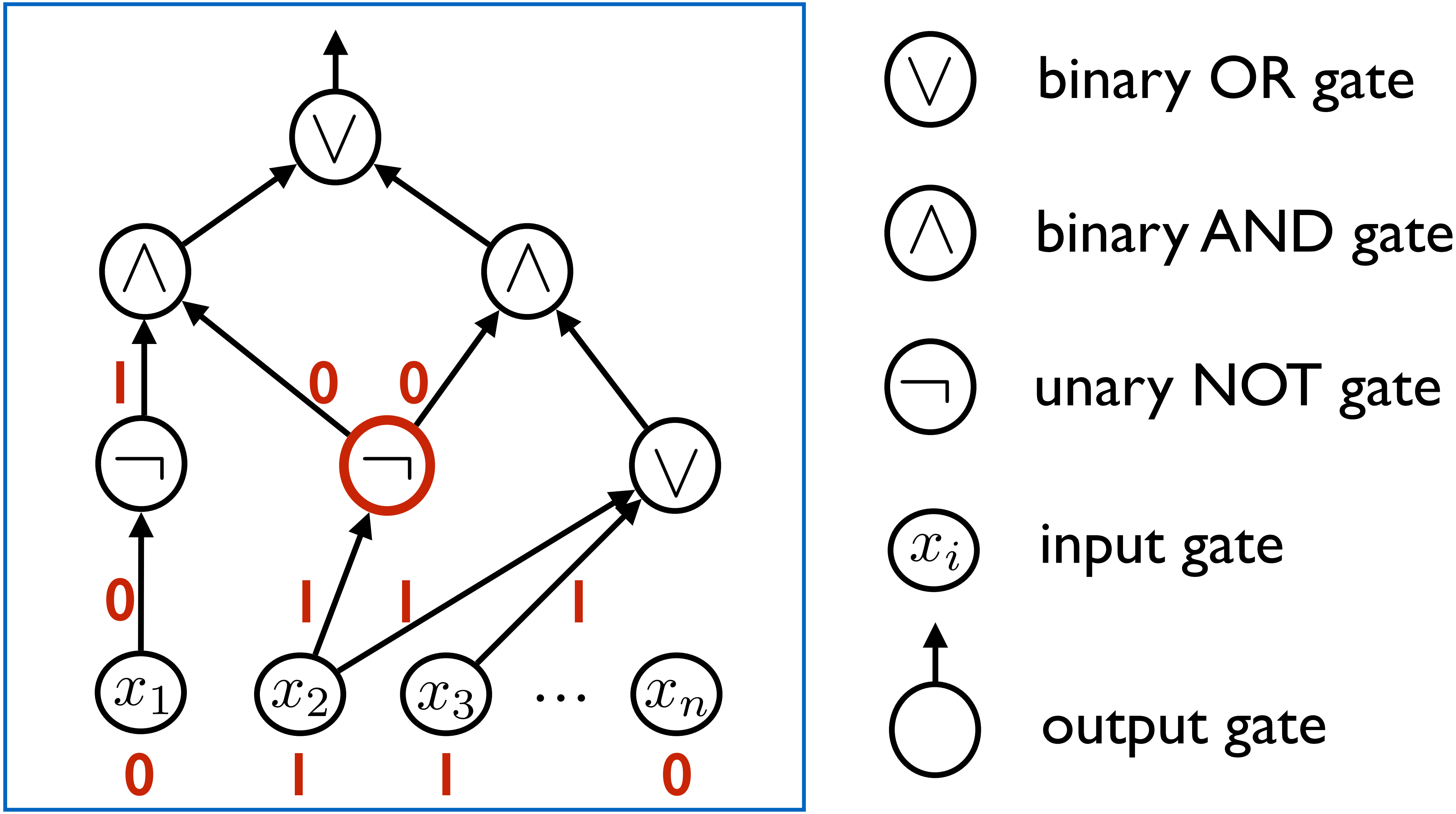




Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

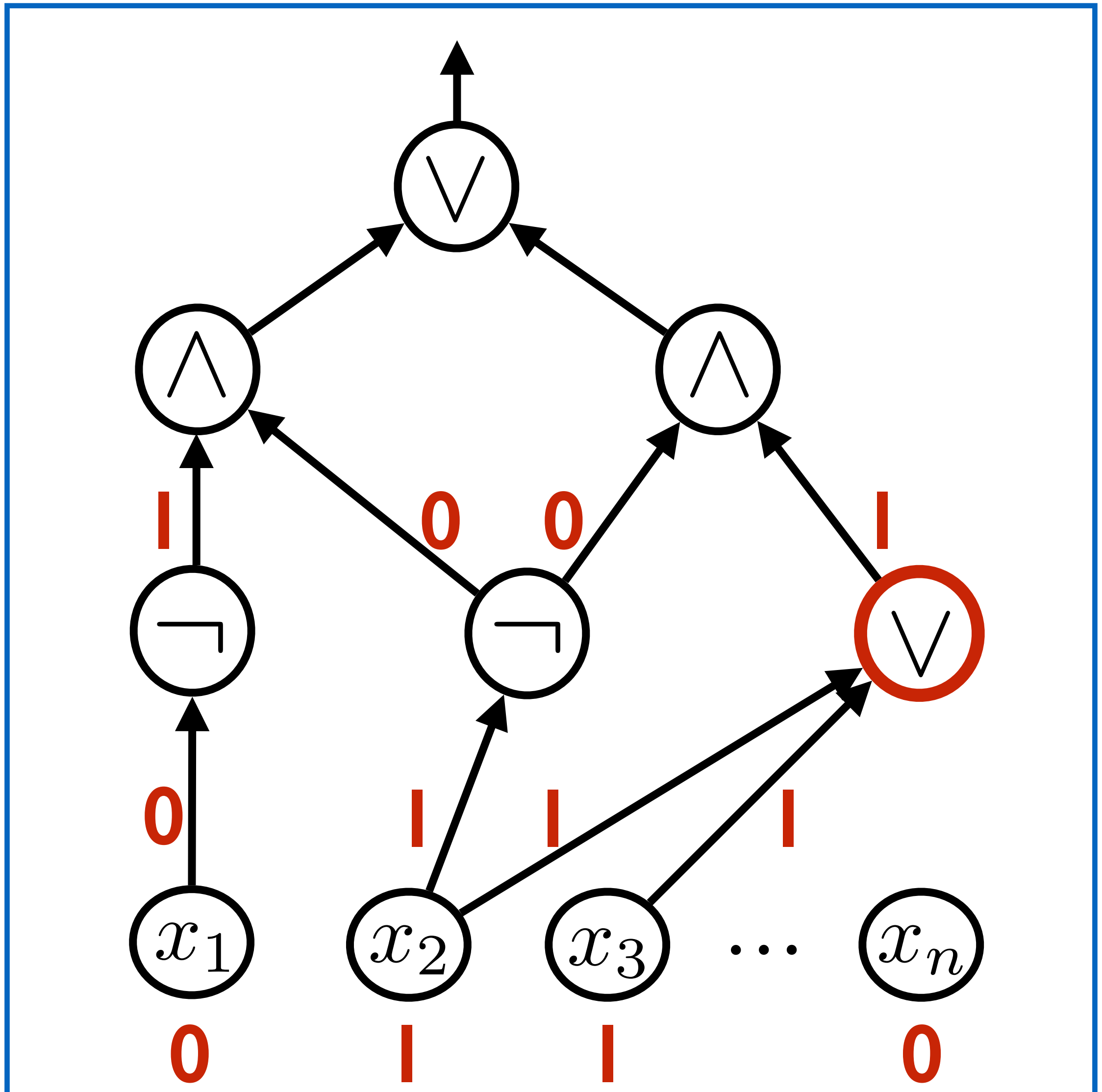
Picture of a circuit

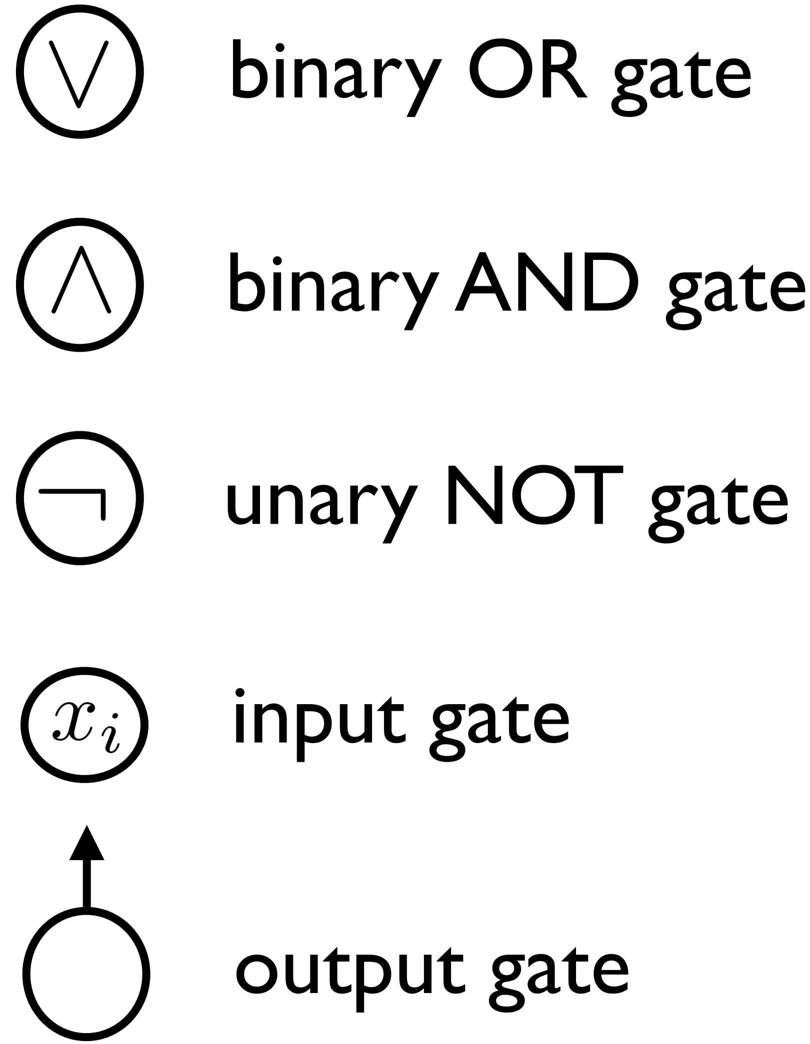


Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit

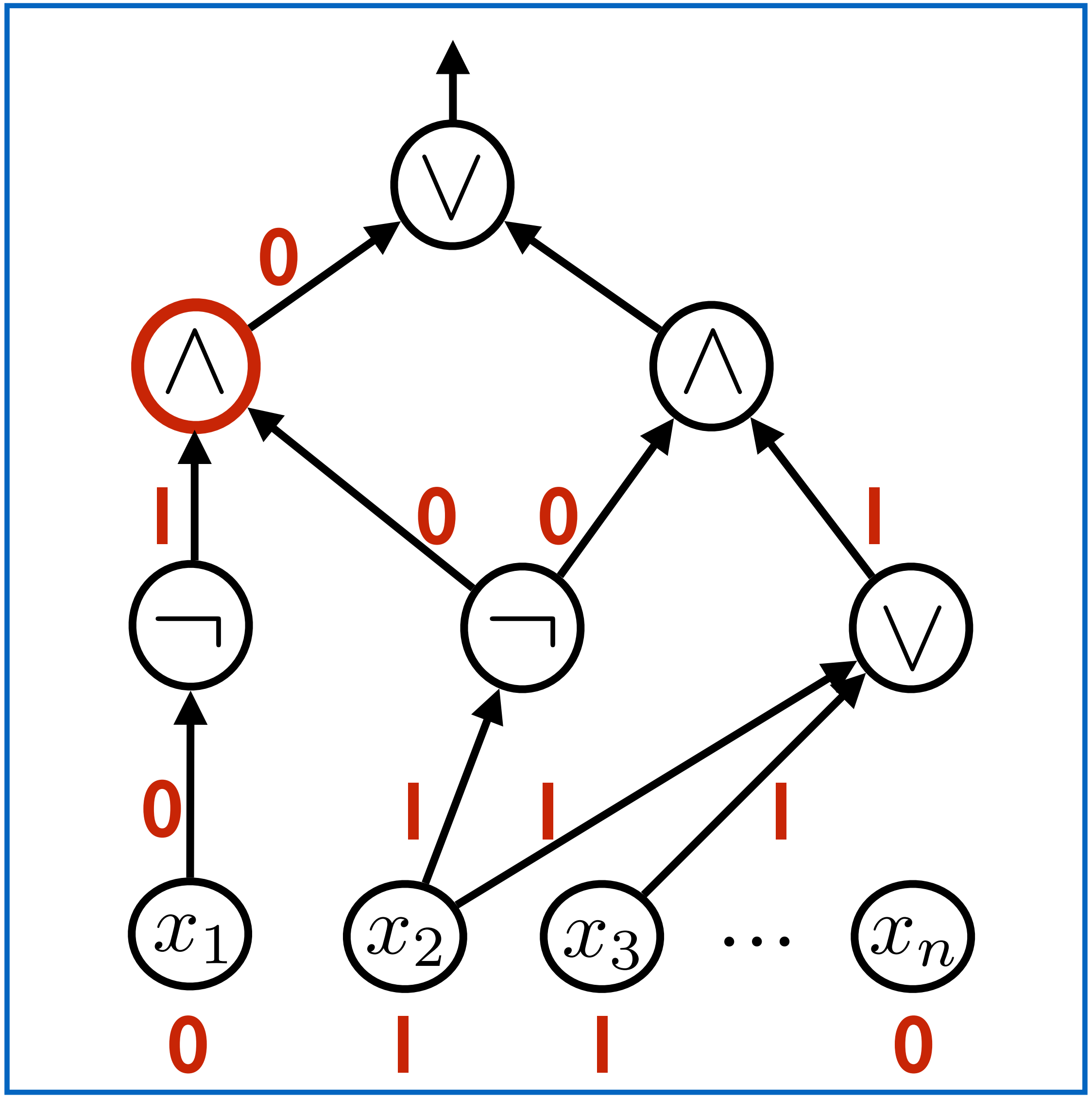


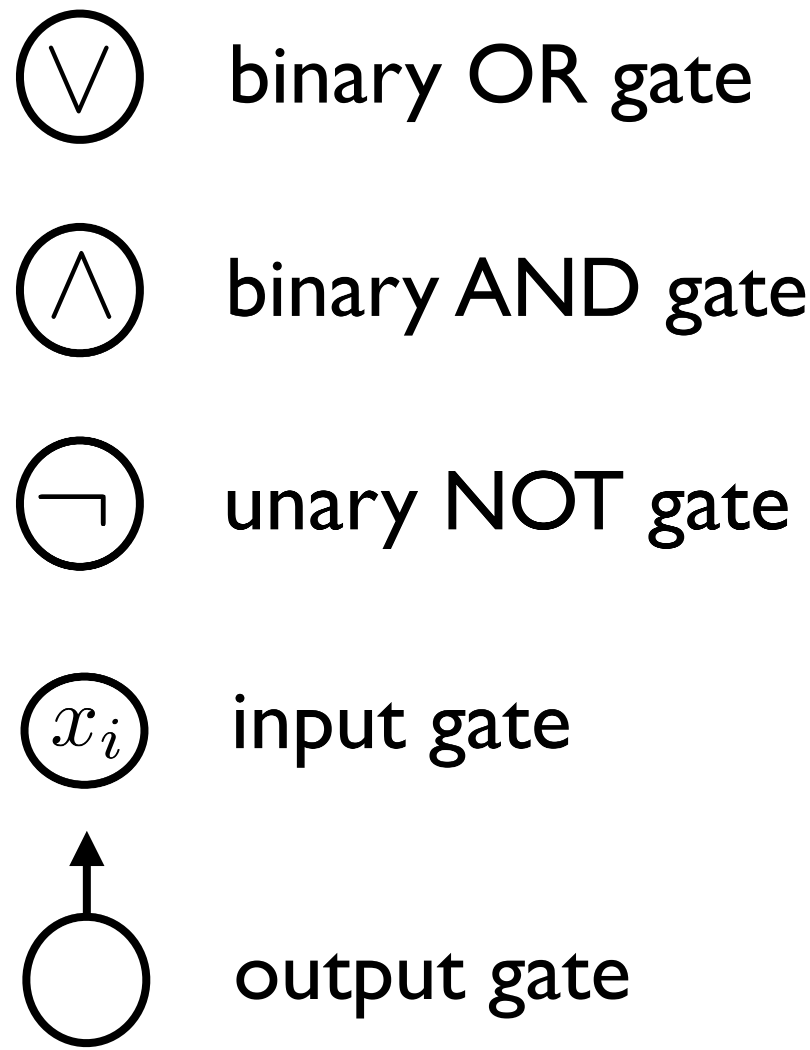


Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit

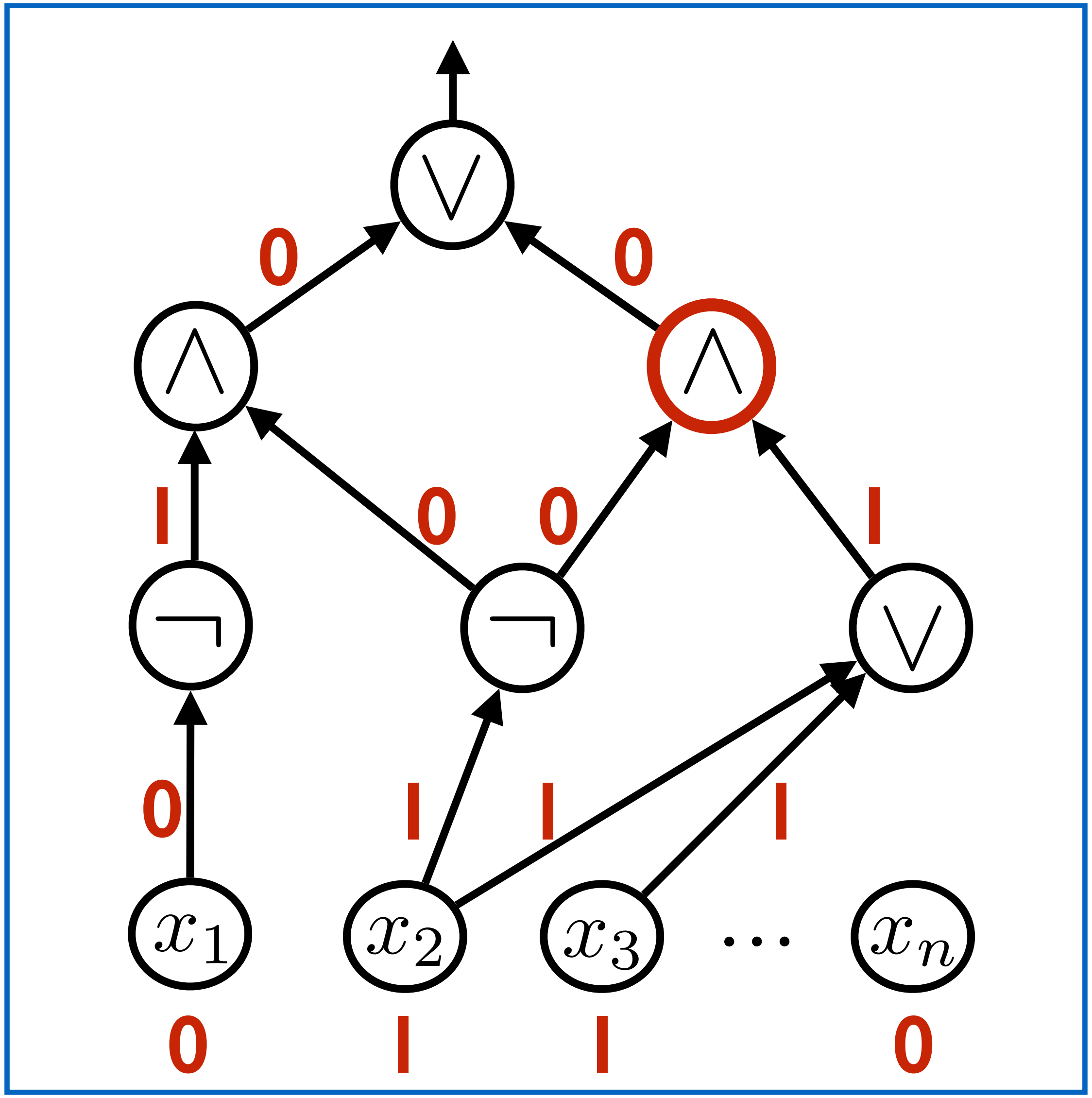


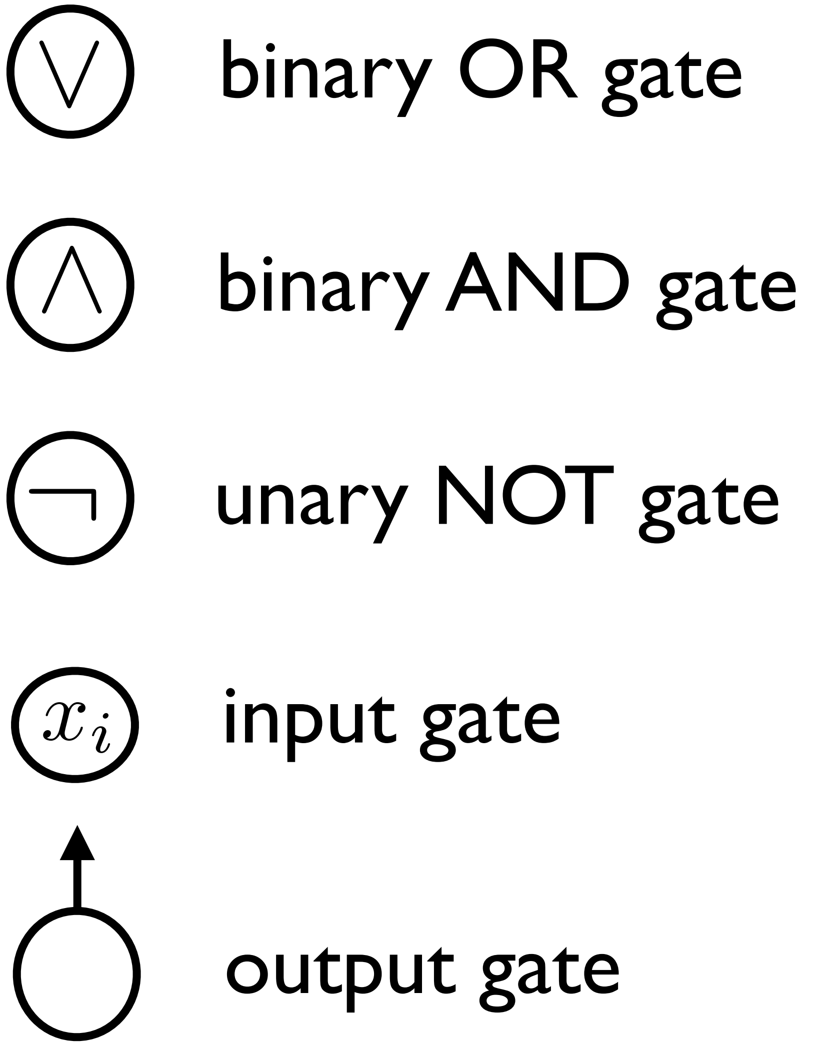


Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit

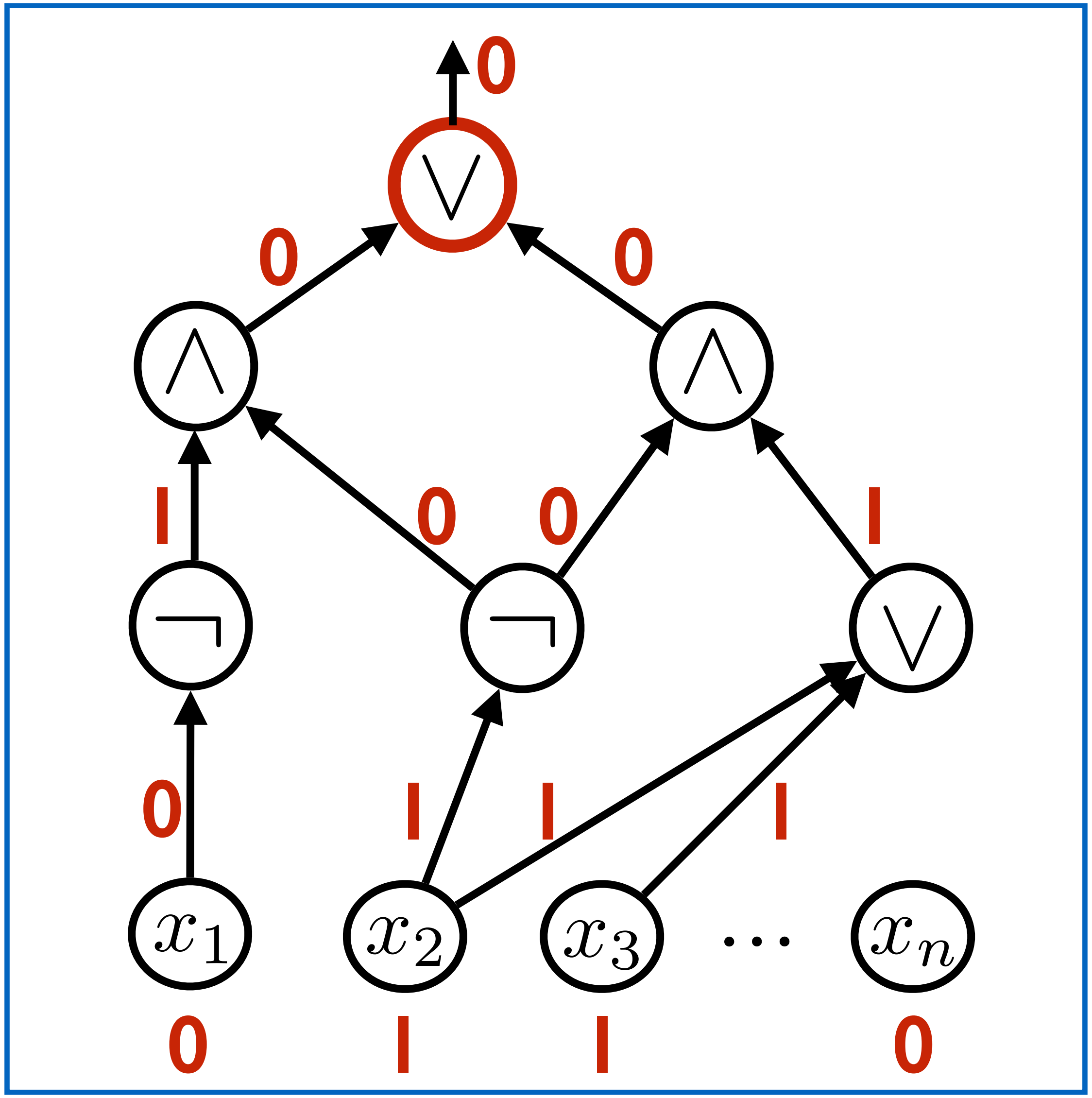


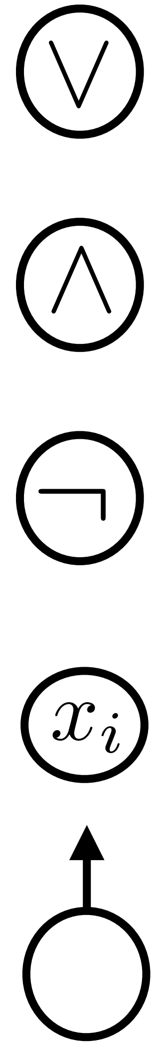


Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \ldots, x_n)$?

Picture of a circuit





binary OR gate

binary AND gate

unary NOT gate

input gate

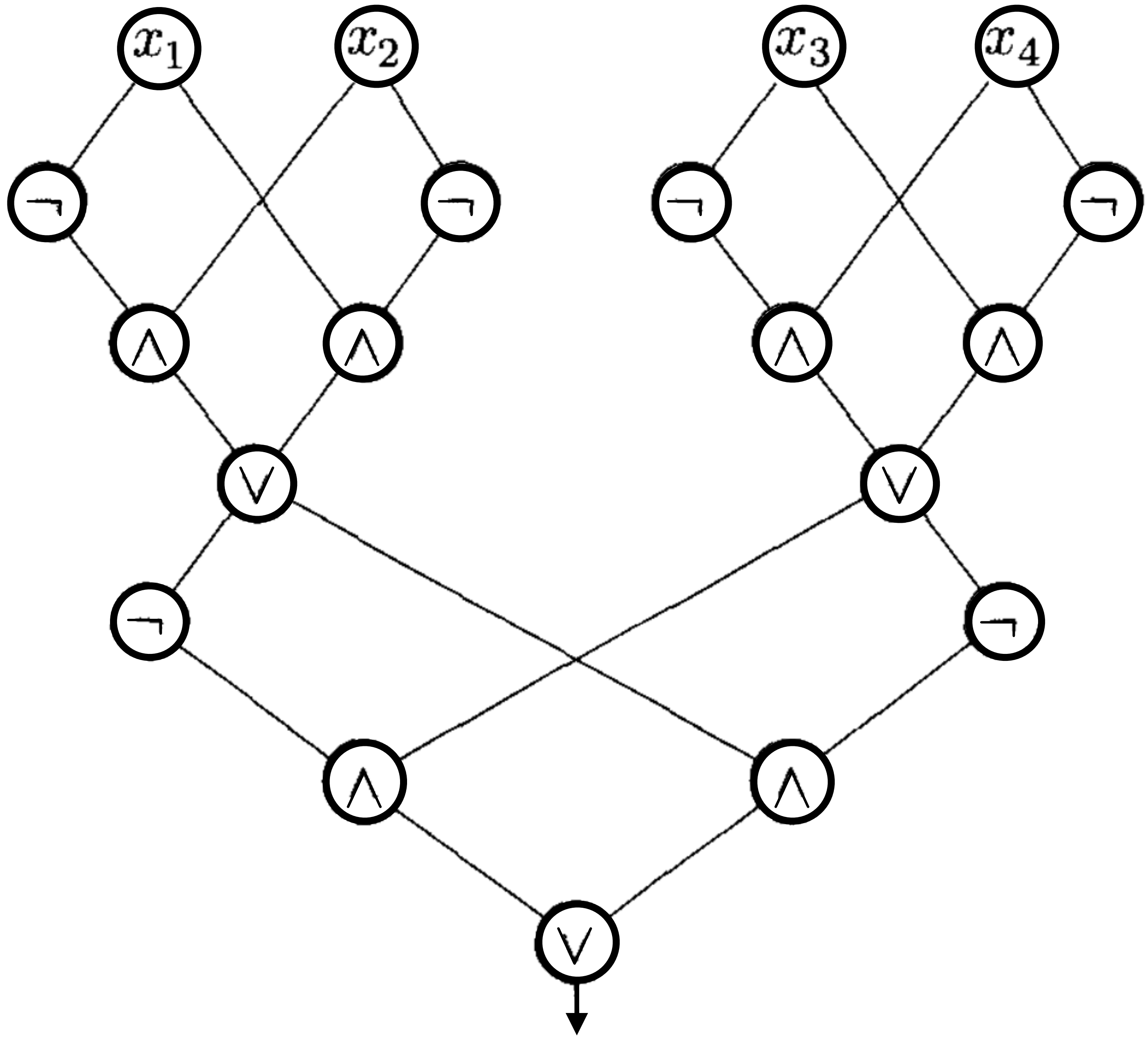
output gate

Computes a function $f : \{0,1\}^n \to \{0,1\}$.

So how does it compute $f(x_1, x_2, \dots, x_n)$?

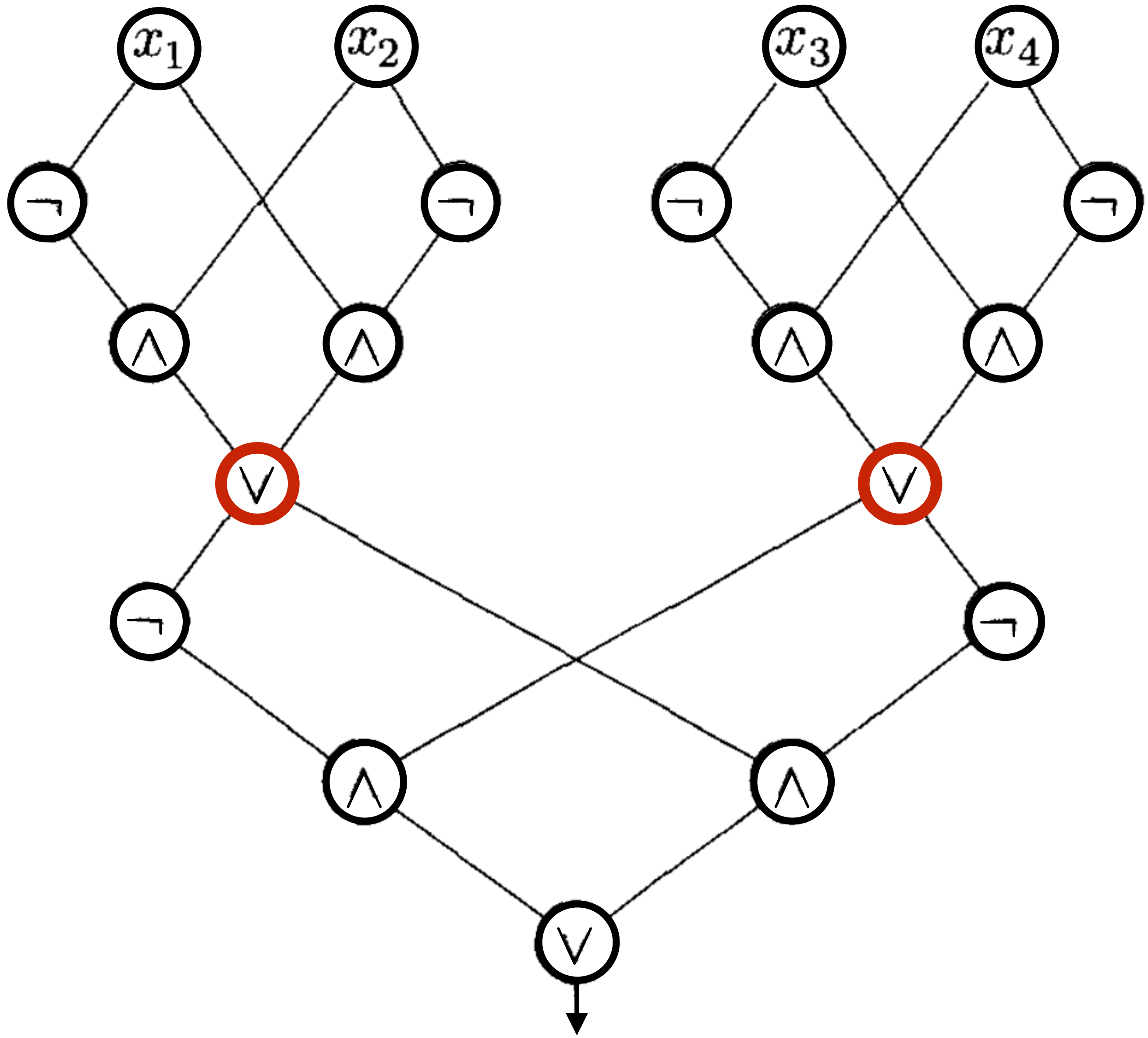
Poll: What does this circuit compute ?

(sometimes circuits are drawn upside down)



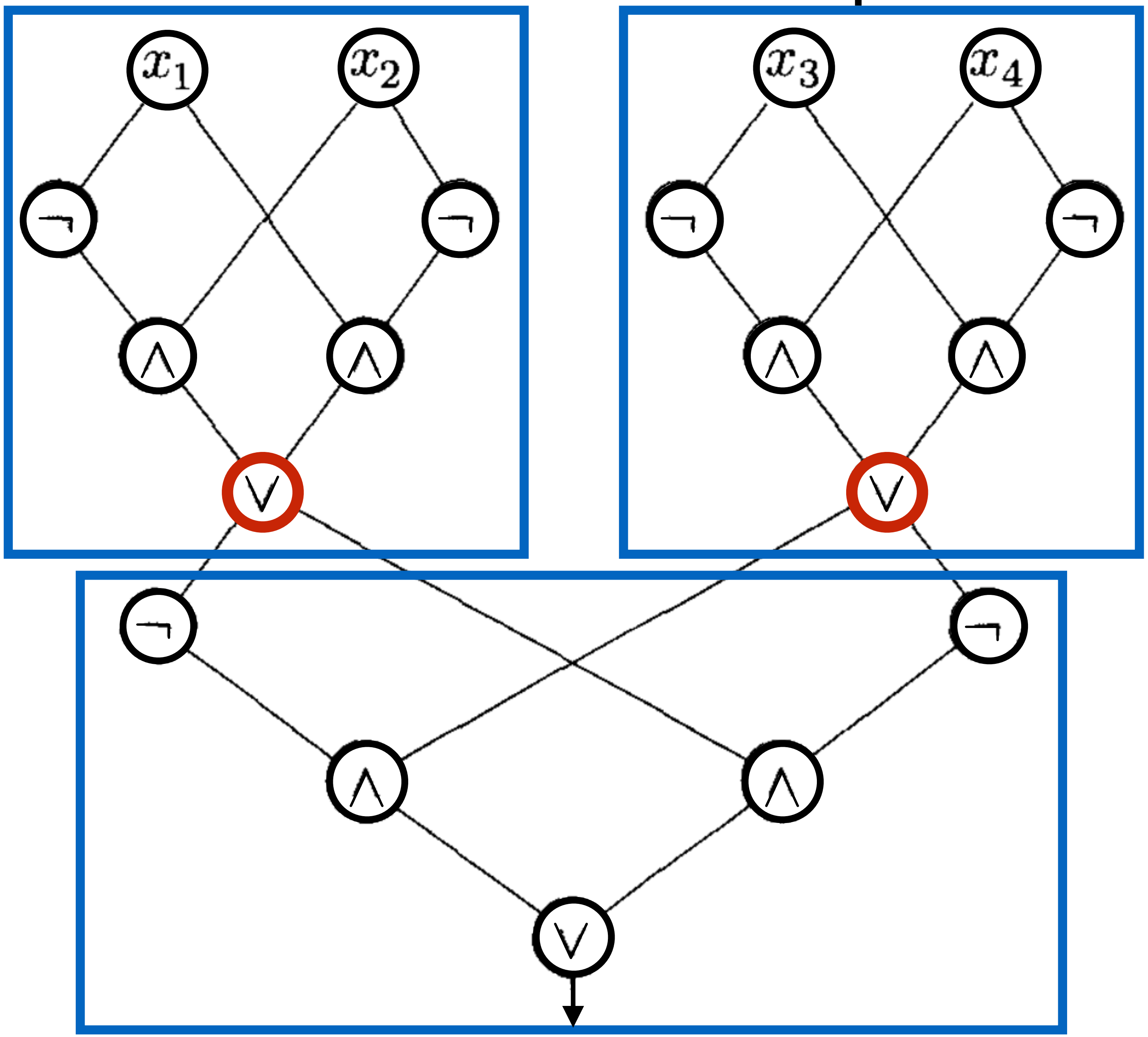
Poll: What does this circuit compute ?

(sometimes circuits are drawn upside down)



Poll: What does this circuit compute ?

(sometimes circuits are drawn upside down)



parity of $x_1 + x_2$

$x_1 \oplus x_2$

parity of $x_3 + x_4$

$x_3 \oplus x_4$

$x_1 \oplus x_2 \oplus x_3 \oplus x_4$

How does a circuit decide/compute a language?

How do we measure the complexity of a circuit?

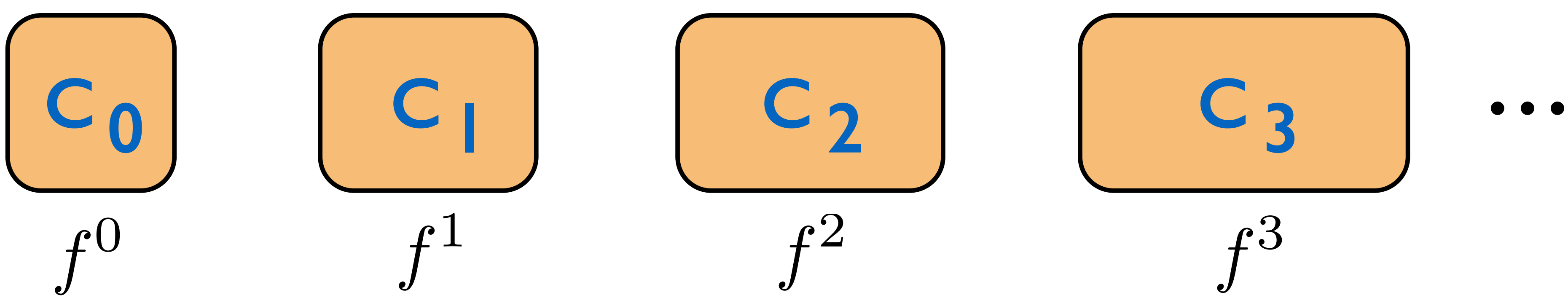
How can a circuit compute a language?

A circuit has a fixed number of inputs.

How can we compute/decide a decision problem $f : \{0,1\}^* \to \{0,1\}$ with circuits?

$$f = (f^0, f^1, f^2, \dots) \quad \text{where} \quad f^n : \{0,1\}^n \to \{0,1\}$$

Construct a circuit for each input length.



A circuit family C is a collection of circuits $(C_0, C_1, C_2, \dots)$

where each C_n takes n input variables.

How can a circuit compute a language?

We say that a circuit family $C = (C_0, C_1, C_2, \ldots)$

decides/computes $f : \{0,1\}^* \to \{0,1\}$

if C_n computes f^n for every n.

Circuit size and complexity

Definition: [size of a circuit]

The size of a circuit is the total number of gates (*counting the input variables as gates too*) in the circuit.

Definition: [size of a circuit family]

The size of a circuit family $C = (C_0, C_1, C_2, \ldots)$ is a function $s(\cdot)$ such that $s(n)$ = size of C_n.

Definition: [circuit complexity]

The circuit complexity of a decision problem is the size of the minimal circuit family that decides it.

(This is the intrinsic complexity with respect to circuit size)

Poll

Let $f : \{0,1\}^* \to \{0,1\}$ be the parity decision problem.

$$f(x) = x_1 + \ldots + x_n \mod 2 \qquad \text{(where } n = |x|\text{)}$$

$$f(x) = x_1 \oplus \cdots \oplus x_n$$

What is the circuit complexity of this function?

Choose the tightest one:

$O(n)$

$O(n^2)$

$O(n^{2.5})$

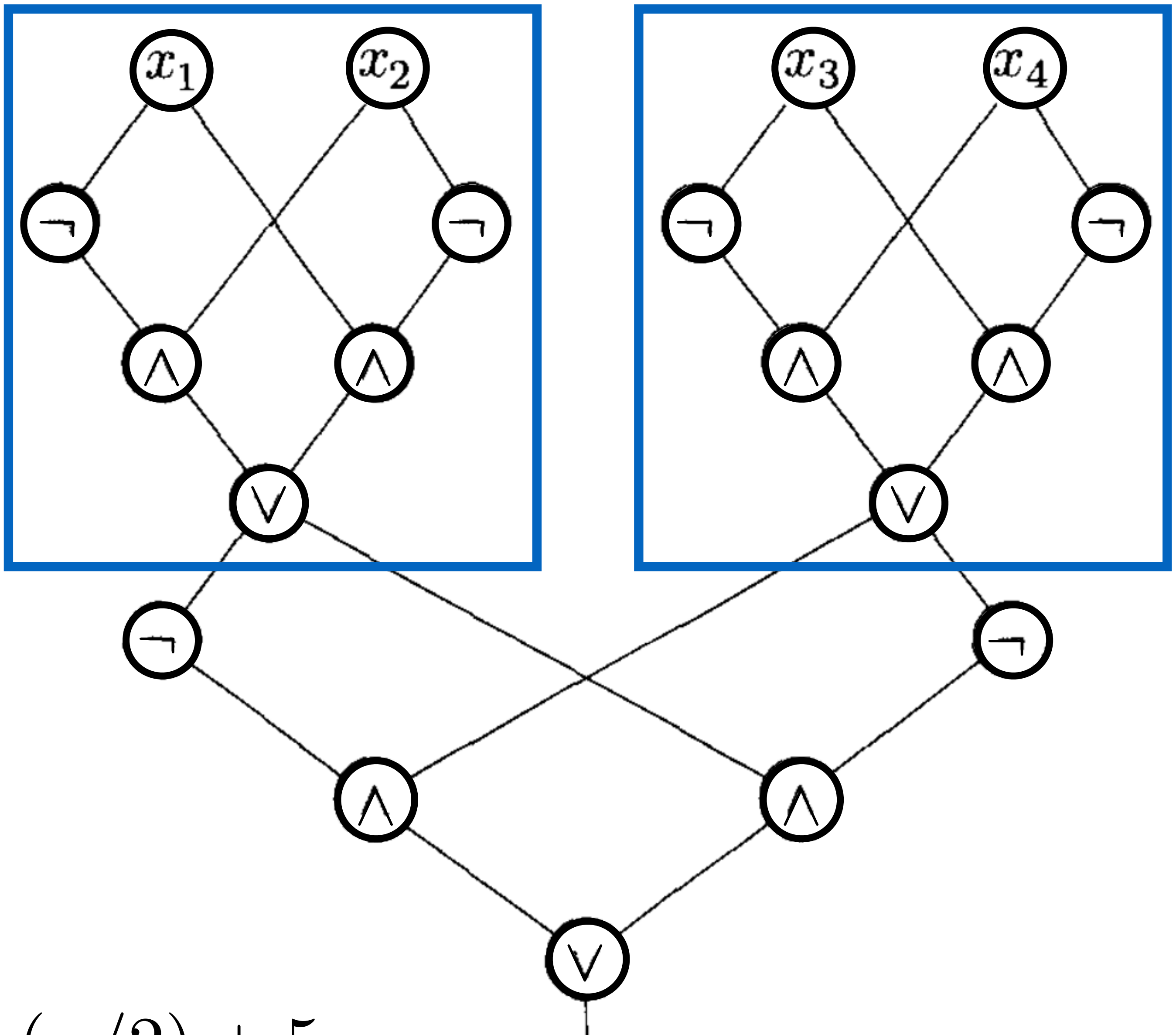
None of the above.

$O(2^n)$

$O(2^{2^n})$

Beats me.

Poll



$$s(n) = 2s(n/2) + 5$$
$$s(1) = 1$$
$$\implies s(n) = O(n).$$

The Big Picture Regarding Boolean Circuits

The big picture

Computability with respect to circuits

Theorem: **Any** decision problem $f : \{0,1\}^* \to \{0,1\}$ can be computed by a circuit family of size $O(2^n)$.

The big picture

Limits of efficient computability with respect to circuits

Theorem: There exists a decision problem such that any circuit family computing it must have size at least $2^n / 4n$.

In fact, **most** decision problems require exponential size.

The big picture

Circuits can efficiently "simulate" TMs

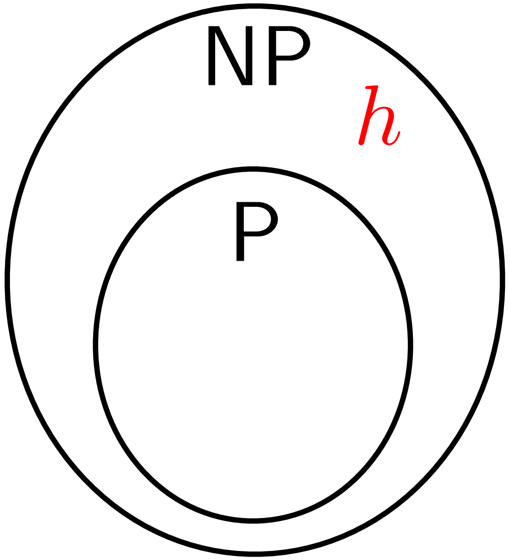
Theorem: Let $f : \{0,1\}^* \rightarrow \{0,1\}$ be a decision problem which can be decided in time $O(T(n))$.
Then it can be computed by a circuit family of size $O(T(n)^2)$.

poly-time TM $\Longrightarrow$ poly-size circuits

no poly-size circuits $\Longrightarrow$ no poly-time TM

The big picture

Circuits can efficiently "simulate" TMs



To show $\mathrm{P} \neq \mathrm{NP}$:

Find h in NP whose circuit complexity is more than poly(n).

The big picture

So we can just work with circuits instead

<u>This is awesome in 2 ways:</u>

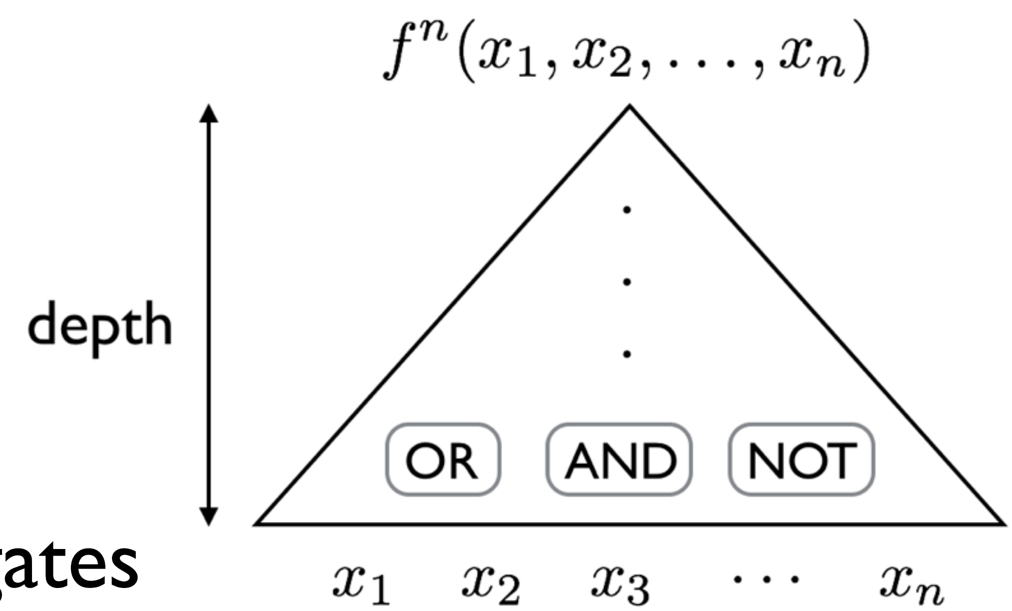
1. Circuits: clean and simple definition of computation.
"Just" a composition of AND, OR, NOT gates.

2. Restrict the circuit.

Make it less powerful.

e.g. (i) restrict *depth*

(ii) restrict types of gates



The big picture

So we can just work with circuits instead

Exciting progress was made in the 1980s.

People thought $\mathrm{P} \neq \mathrm{NP}$ would be proved soon.

Alas…

After 60 years of research,
best lower bound on circuit size for an explicit function:

$$5n - \text{peanuts}$$

The big picture

Theorem: Any decision problem $f : \{0,1\}^* \to \{0,1\}$ can be computed by a circuit family of size $O(2^n)$.

Theorem: There exists a decision problem such that any circuit family computing it must have size at least $2^n/4n$.

Theorem: Let $f : \{0,1\}^* \to \{0,1\}$ be a decision problem which can be decided in time $O(T(n))$.
Then it can be computed by a circuit family of size $O(T(n)^2)$.

Theorem 1: Max circuit size of a function

Theorem: Any decision problem $f : \{0,1\}^* \to \{0,1\}$ can be computed by a circuit family of size $O(2^n)$.

Proof:

Goal:
construct a circuit of size $O(2^n)$ for $f^n : \{0,1\}^n \to \{0,1\}$.

Observation:

$$
\begin{aligned}
f^n(x_1, x_2, \ldots, x_n) \;=\; & (x_1 \wedge f^n(1, x_2, \ldots, x_n)) \;\vee \\
& (\neg x_1 \wedge f^n(0, x_2, \ldots, x_n))
\end{aligned}
$$

Theorem 1: Max circuit size of a function

Theorem: Any decision problem $f : \{0,1\}^* \to \{0,1\}$ can be computed by a circuit family of size $O(2^n)$.

Proof:

Goal:
construct a circuit of size $O(2^n)$ for $f^n : \{0,1\}^n \to \{0,1\}$.

Observation:

$$f^n(x_1, x_2, \ldots, x_n) = (\overset{1}{\cancel{x_1}} \wedge \boxed{f^n(1, x_2, \ldots, x_n)}) \vee$$

if $x_1 = 1$

$$\underset{0}{\cancel{(\neg \underset{0}{\cancel{x_1}} \wedge f^n(0, x_2, \ldots, x_n))}}$$

Theorem 1: Max circuit size of a function

Theorem: Any decision problem $f : \{0,1\}^* \to \{0,1\}$ can be computed by a circuit family of size $O(2^n)$.

Proof:

Goal:
construct a circuit of size $O(2^n)$ for $f^n : \{0,1\}^n \to \{0,1\}$.

Observation:

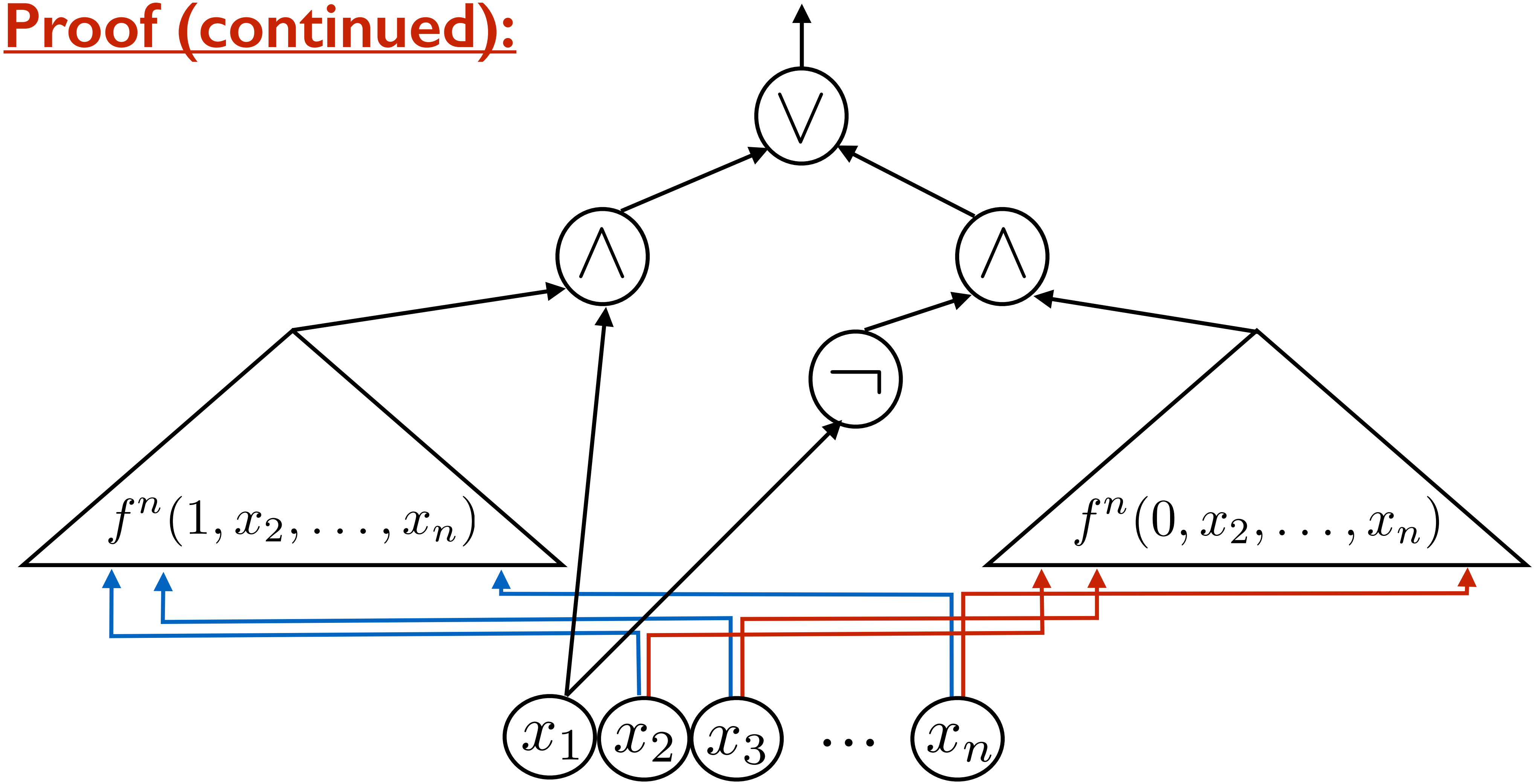
$$f^n(x_1, x_2, \ldots, x_n) = \cancel{(\overset{0}{\cancel{x_1}} \wedge f^n(1, x_2, \ldots, \overset{0}{x_n}))} \vee$$

if $x_1 = 0$

$$(\neg \underset{1}{\cancel{x_1}} \wedge \boxed{f^n(0, x_2, \ldots, x_n)})$$

Theorem 1: Max circuit size of a function

Proof (continued):



$s(n) =$ max size of a circuit computing n-variable function

$$s(n) \leq \; 2s(n-1) + 5\,, \quad s(1) \leq 3 \implies s(n) = O(2^n) \quad \square$$

Poll

How many different functions $f : \{0,1\}^n \to \{0,1\}$ are there?

- n
- $2n$
- n^2
- 2^n
- 2^{2^n}
- none of the above
- beats me

Theorem 2: Some functions are hard

Theorem: There exists a decision problem such that any circuit family computing it must have size at least $2^n/5n$.

Proof:

Want to show: there is a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $< 2^n/5n$.

Observation: # possible functions is 2^{2^n}.

Claim1: # circuits of size at most s is $\leq 2^{5s \log s}$.

Claim2: For $s \leq 2^n/5n$, $2^{5s \log s} < 2^{2^n}$.

$\#\ \text{circuits} < \#\ \text{functions}$

Theorem 2: Some functions are hard

Theorem: There exists a decision problem such that any circuit family computing it must have size at least $2^n/5n$.

Proof:

Want to show: there is a function $f : \{0,1\}^n \to \{0,1\}$ that cannot be computed by a circuit of size $< 2^n/5n$.

Observation: # possible functions is 2^{2^n}.

Claim1: # circuits of size at most s is $\leq 2^{5s \log s}$.

Claim2: For $s \leq 2^n/5n$, $2^{5s \log s} < 2^{2^n}$.

We are done once we prove Claim 1. (Claim 2 is easy/uninteresting.)

Theorem 2: Some functions are hard

Proof (continued):

Claim1: # circuits of size at most s is $\leq 2^{5s \log s}$.

Proof of claim:

Recall $|A| \leq |B|$ iff $B \twoheadrightarrow A$.

Let $A = \{\text{circuits of size at most } s\}$

$$B = \{0,1\}^{5s \log s} \qquad |B| = 2^{5s \log s}$$

To show $B \twoheadrightarrow A$:

encode a circuit with a binary string of length $5s \log s$.

(just like the CS method)

Theorem 2: Some functions are hard

Proof (continued):

Claim1: # circuits of size at most s is $\leq 2^{5s \log s}$.

Proof of claim (continued):

Encoding a circuit with a binary string of length $5s \log s$:

Number the gates: 1, 2, 3, 4, …, s

For each gate in the circuit, write down:

- type of the gate (3 bits)
- from which gates the inputs are coming from (2 log s bits)

Total: s(3 + 2 log s) bits

(3s + 2s log s) bits ≤ (5s log s) bits $\square$

Theorem 2: Some functions are hard

Claude Shannon
(1916 - 2001)

That was due to Claude Shannon (1949).

Father of *Information Theory*.

A non-constructive argument.

In fact, it is easy to show that **most** functions require exponential size circuits.

Theorem 3: Circuits can simulate TMs

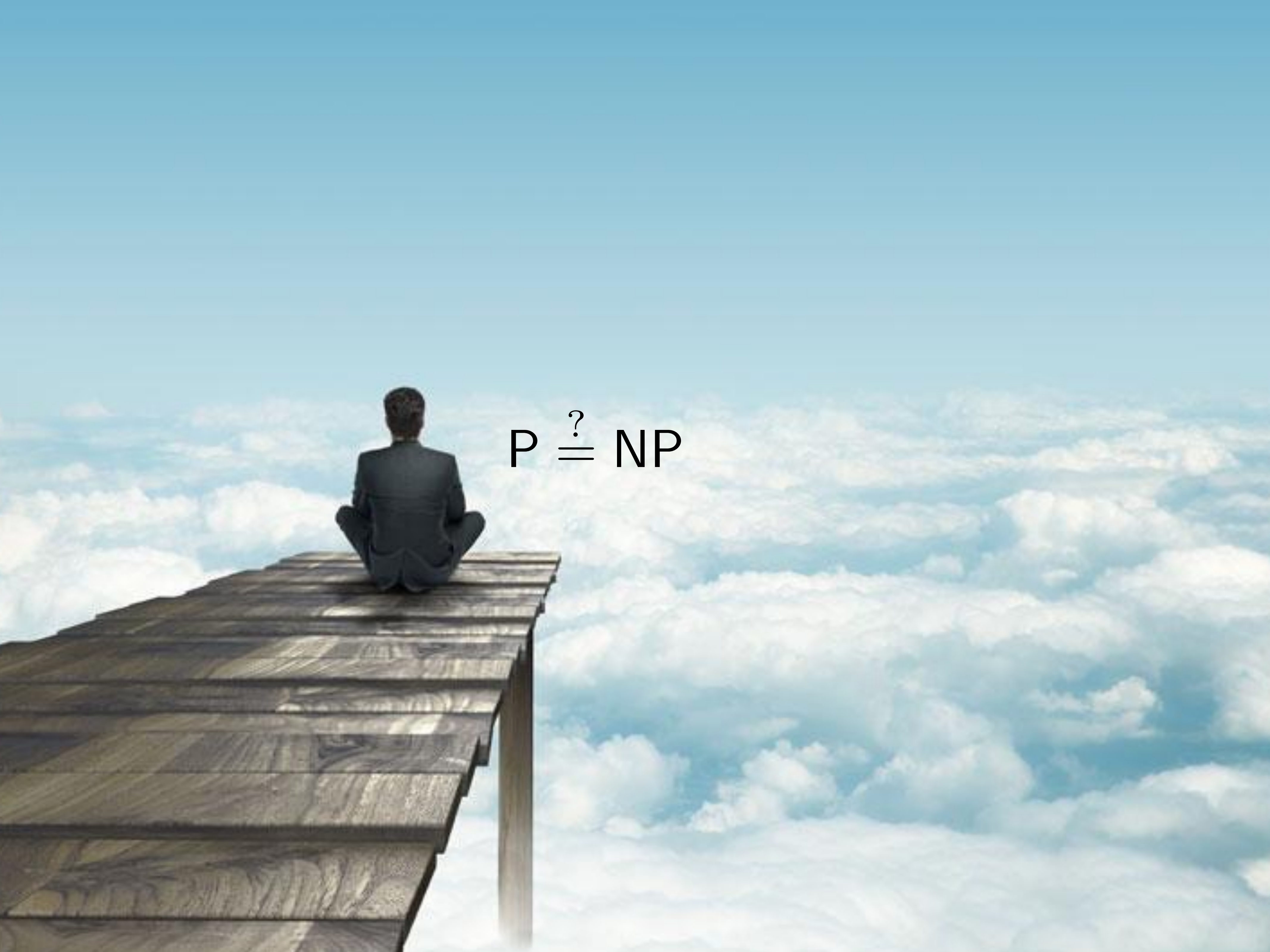
Theorem: Let $f : \{0,1\}^* \to \{0,1\}$ be a decision problem which can be decided in time $O(T(n))$.
Then it can be computed by a circuit family of size $O(T(n)^2)$.

How can you prove such a theorem?

If you like a challenge, try to prove it yourself.

If you don't like a challenge, but still curious,
see the course notes for a sketch of the proof.

If you don't like a challenge, and are not curious,
😞 you can ignore the proof.


$\mathsf{P} \stackrel{?}{=} \mathsf{NP}$