P and NP

Lecture 13

P=NP?

- We talked about machines that accept sets of strings
  - Best to think of a language in terms of a YES/NO question
  - P = YES/NO questions that can be answered in polynomial time in input size (algorithm)

e.g. is this array sorted? O(n) time

is N prime? (log N bits input)

P=NP?

- NP= Non-deterministic Polynomial Time
- Something to do with Non-Deterministic TM
- YES/No problems where YES instance can be verified in polynomial time

e.g. is this array sorted? verify by running O(n) algorithm

or something not so clear how to find from scratch: does this graph have Hamiltonian cycle?

We do not know if this is in P.

Can check short proof = Non-deterministic choices.

P=NP?

- Asymmetry: how can I convince you that there is no Hamiltonian cycle?

we don't know! check all n vertex cycles.

NP only requires that if the answer is YES I can convince you in polynomial time.

If answer is NO : ummm…

If problem is in P?

P=NP?

- Million Dollar question: P=NP?
  - of course not!
- Clay math institute: 7 most important problems.
  - P = NP is number 1. ($1M)
- it would imply: If there is a short proof, there is an easy way to discover it… Trivialize math.

In a *search* problem, given an input $x \in \{0,1\}^*$ we want to compute some answer $y \in \{0,1\}^*$ that is in some relation to x, if such a y exists. Thus, a search problem is specified by a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, where $(x, y) \in R$ if and only if y is an admissible answer given x.

We denote by **P** the class of decision problems that are solvable in polynomial time.

We say that a search problem defined by a relation R is a **NP** search problem if the relation is efficiently computable and such that solutions, if they exist, are short. Formally, R is an **NP** search problem if there is a polynomial time algorithm that, given x and y, decides whether $(x, y) \in R$, and if there is a polynomial p such that if $(x, y) \in R$ then $|y| \leq p(|x|)$.

We say that a decision problem L is an **NP** decision problem if there is some **NP** relation R such that $x \in L$ if and only if there is a y such that $(x, y) \in R$. Equivalently, a decision problem L is an **NP** decision problem if there is a polynomial time algorithm $V(\cdot,\cdot)$ and a polynomial p such that $x \in L$ if and only if there is a y, $|y| \leq p(|x|)$ such that $V(x, y)$ accepts.

We denote by **NP** the class of **NP** decision problems. The class **NP** has the following alternative characterization, from which it takes its name. (**NP** stands for *N*ondeterministic *P*olynomial time.)

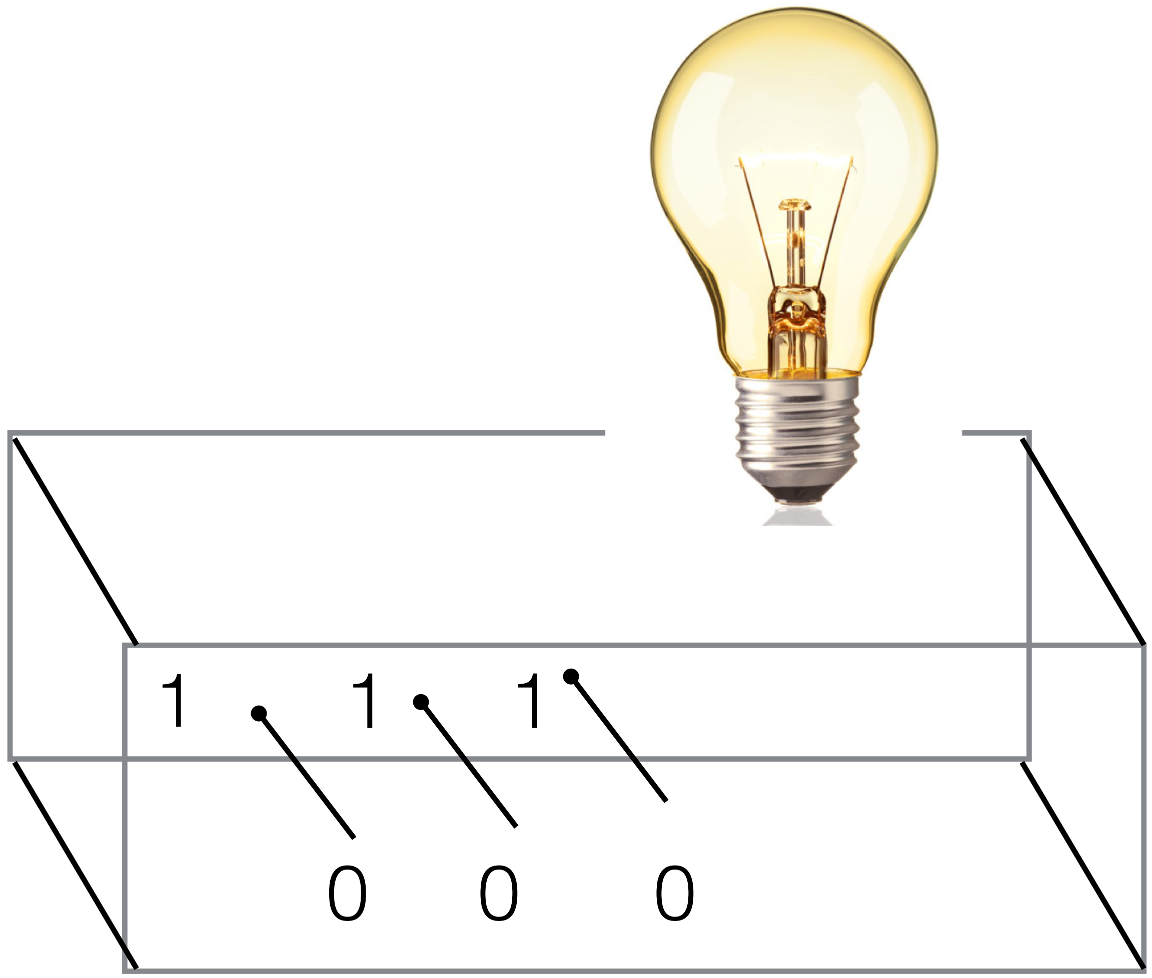
Theorem 1 **NP** *is the set of decision problems that are solvable in polynomial time by a non-deterministic Turing machine.*

Exercise

Show that NP is the set of decision problems solvable in poly time by a non deterministic TM.

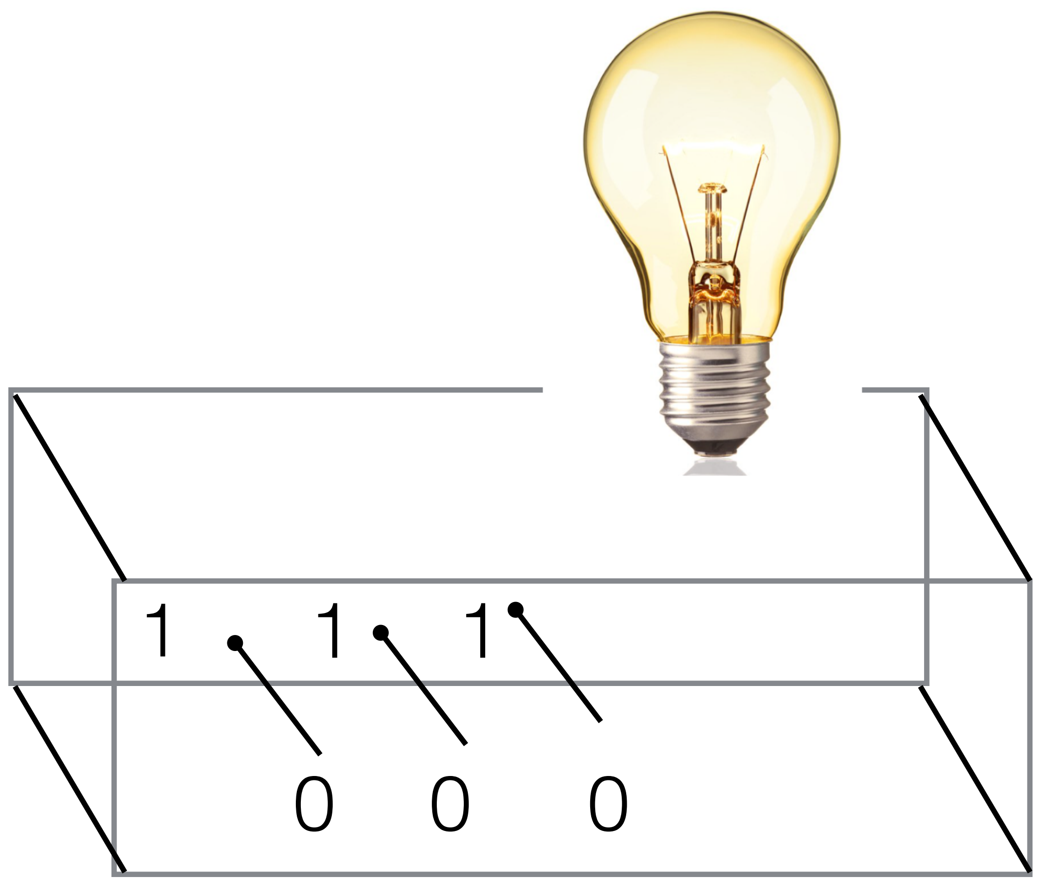
Problem in NP

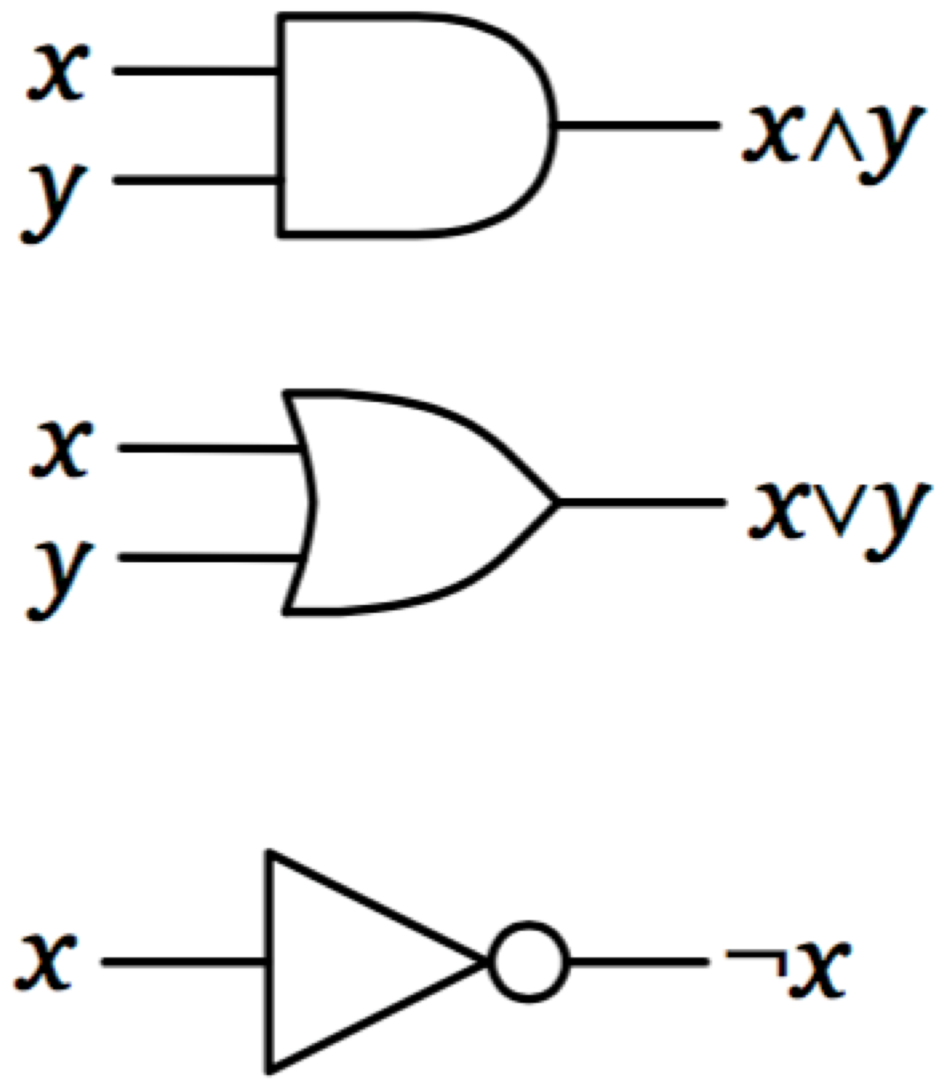
- Problem in NP. It is the “worst problem” in NP



Problem in NP

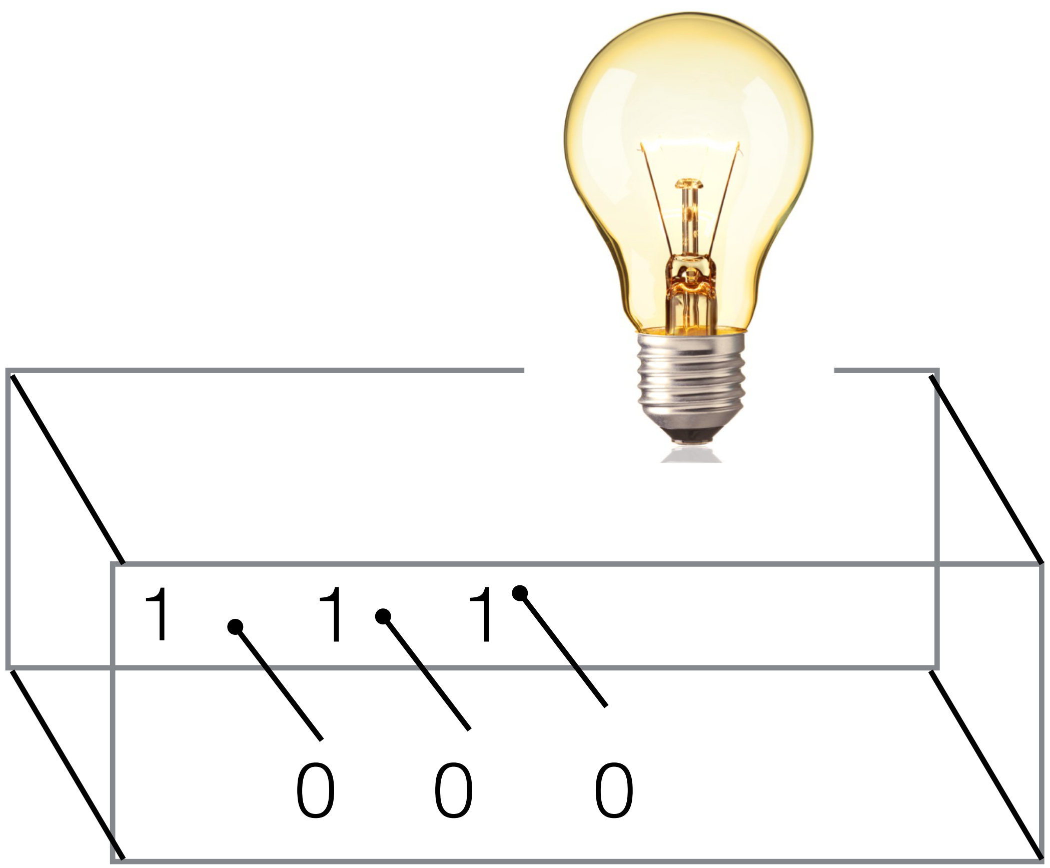
- Problem in NP. It is the “worst problem” in NP

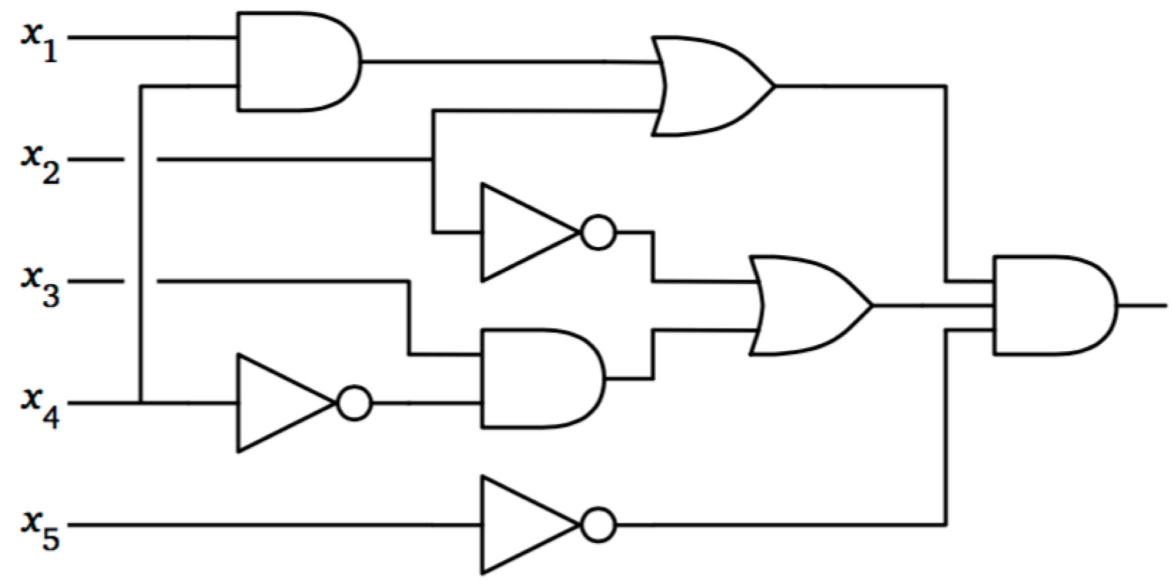




Problem in NP

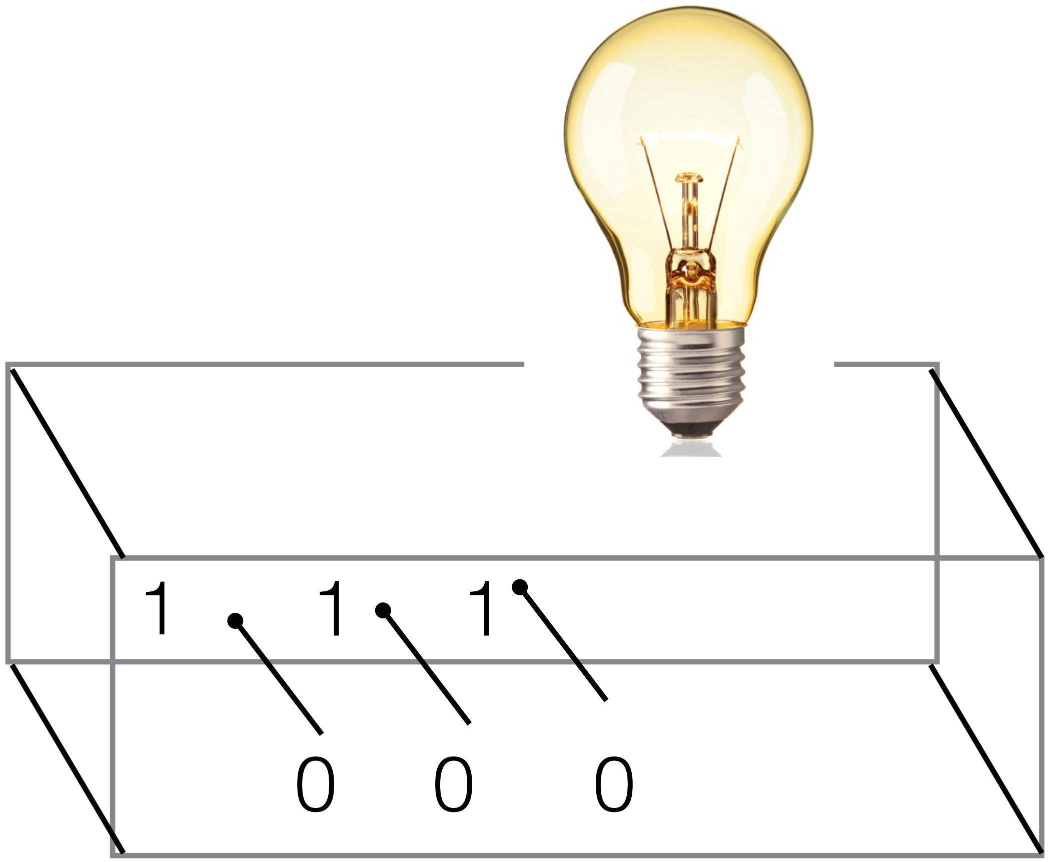
- Problem in NP. It is the “worst problem” in NP

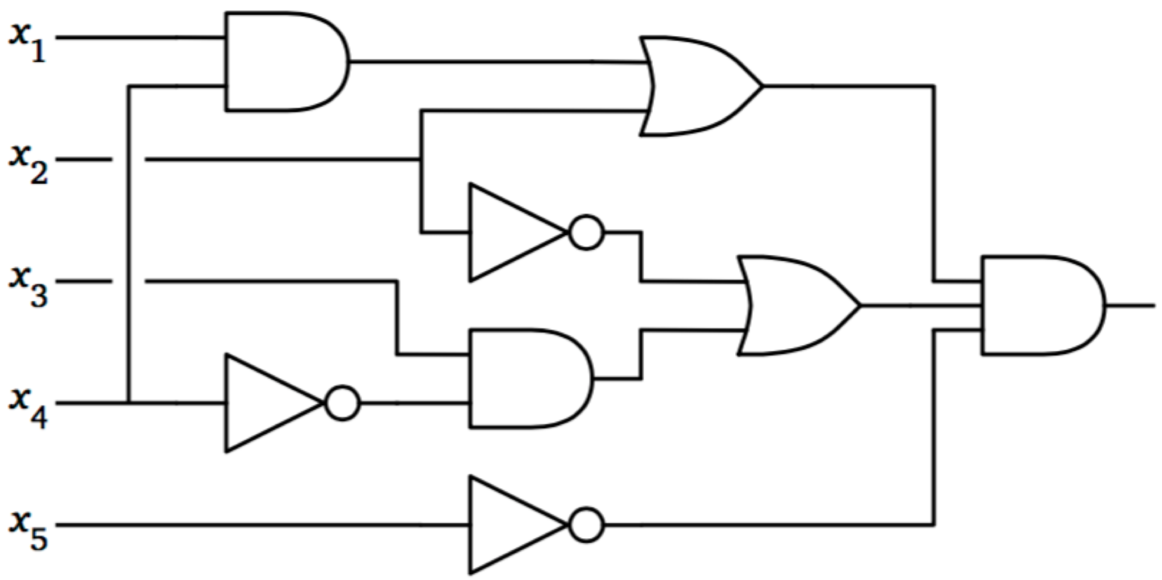




Problem in NP

- Problem in NP. It is the “worst problem” in NP





Circuit Satisfiability:

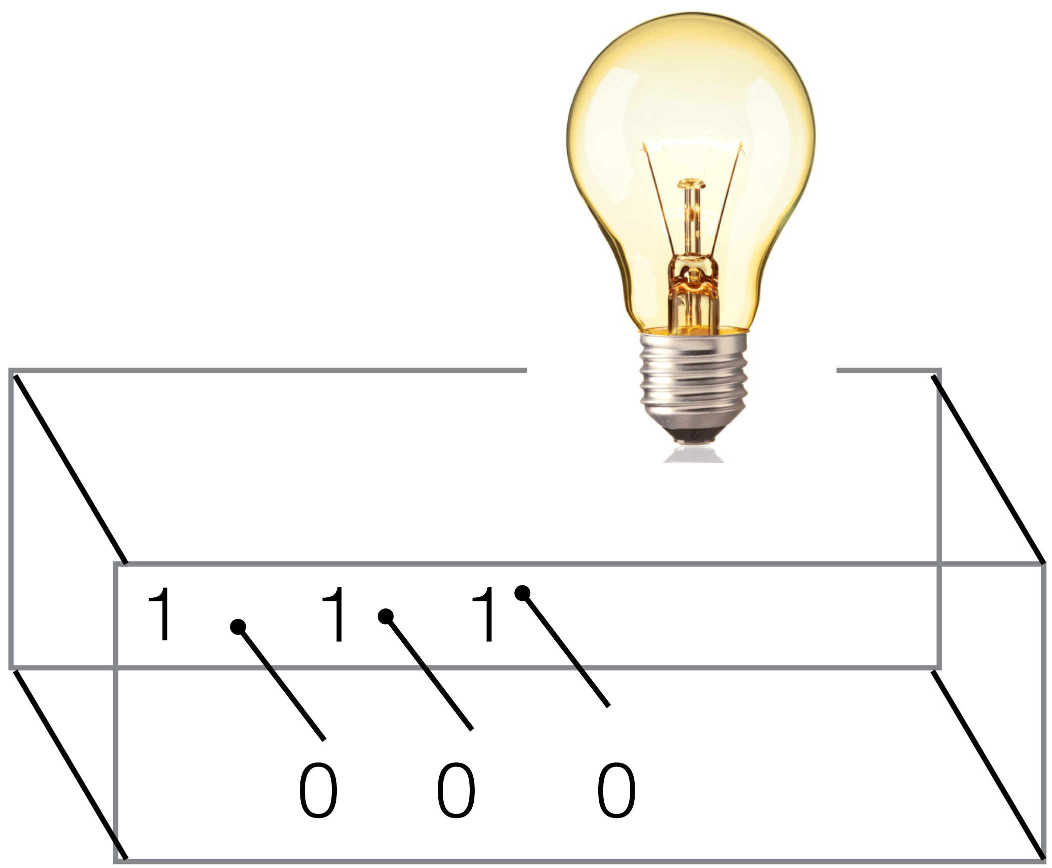
Given a boolean circuit are there inputs the produce output 1?

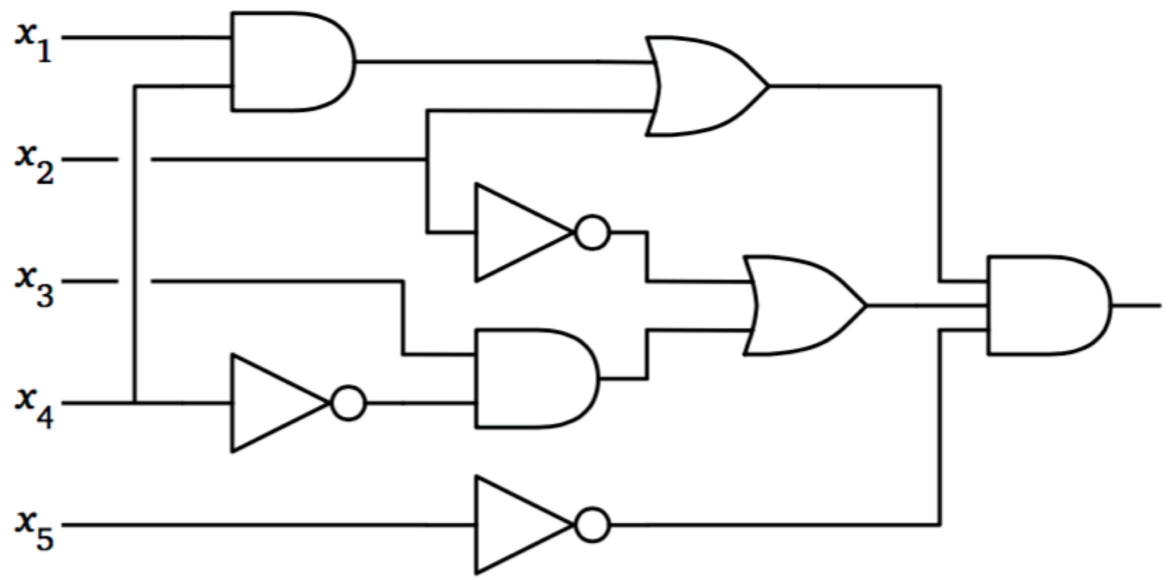
Obvious $O(2^n n)$ algorithm, brute force

Best known algorithm $O(2^n/n)$

Problem in NP

- Problem in NP. It is the “worst problem” in NP





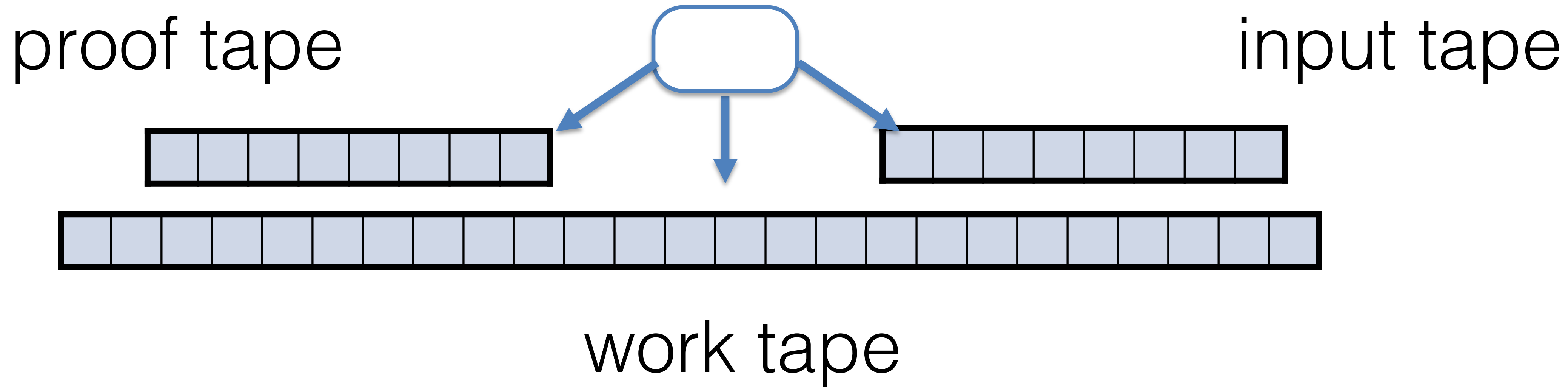
If I can solve CircuitSat in P,
then every other problem in NP has
a polynomial time algorithm!
Levin, Cook

Cook Levin

Cook-Levin Theorem:
CircuitSAT is NP-hard.

Cook-Levin Theorem:
If CircuitSAT in P then P=NP

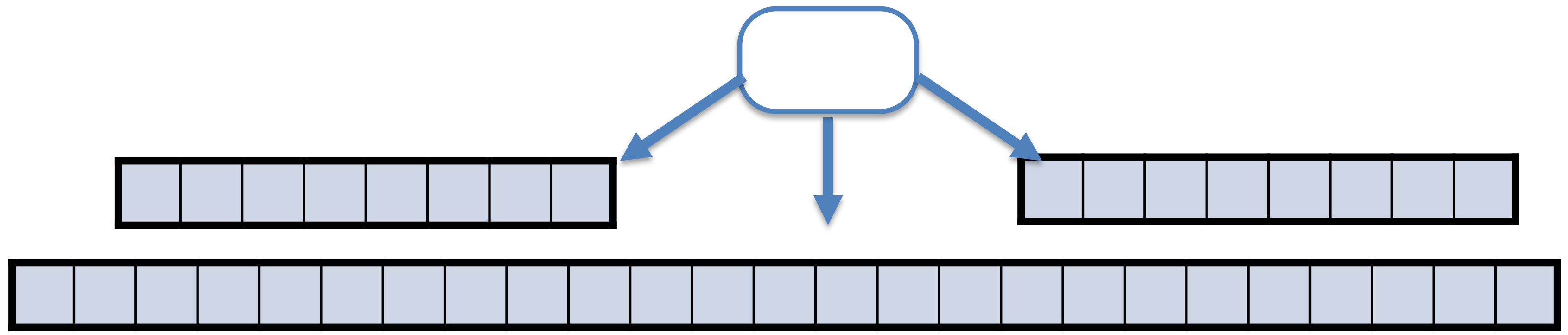
Cook Levin Proof? Non-deterministic TMs

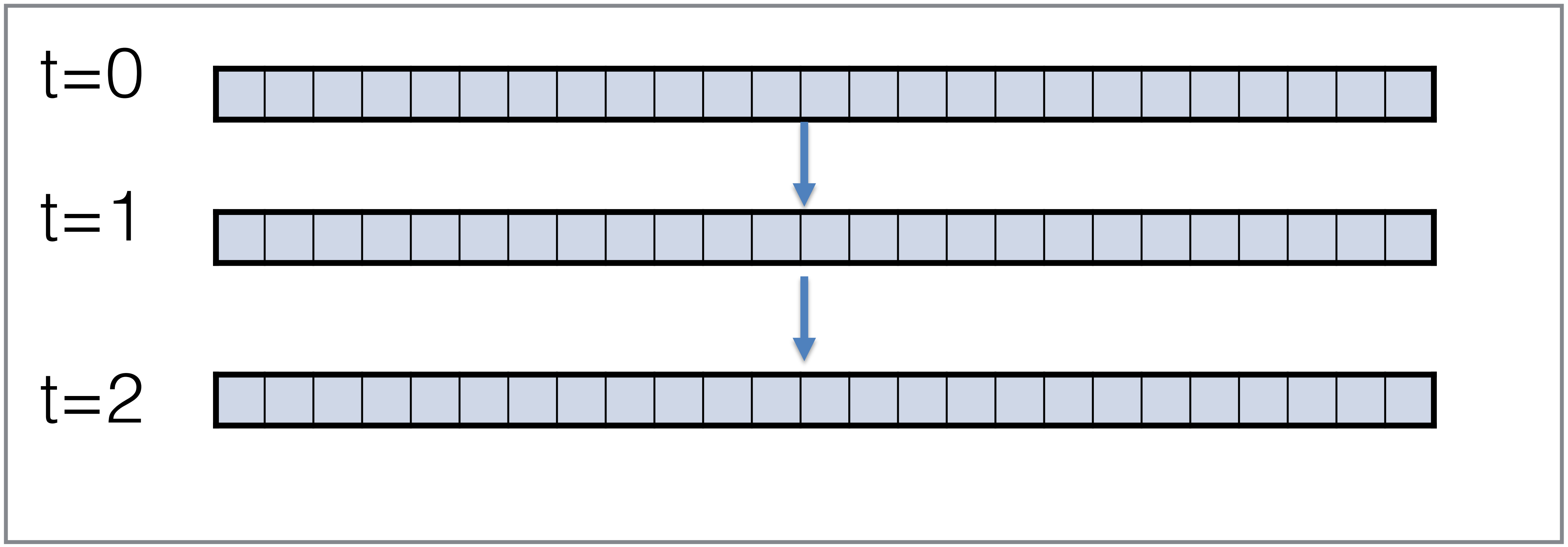


Verify if input is YES in p time =
is there a string to put in the proof tape to make this TM to accept in poly time?

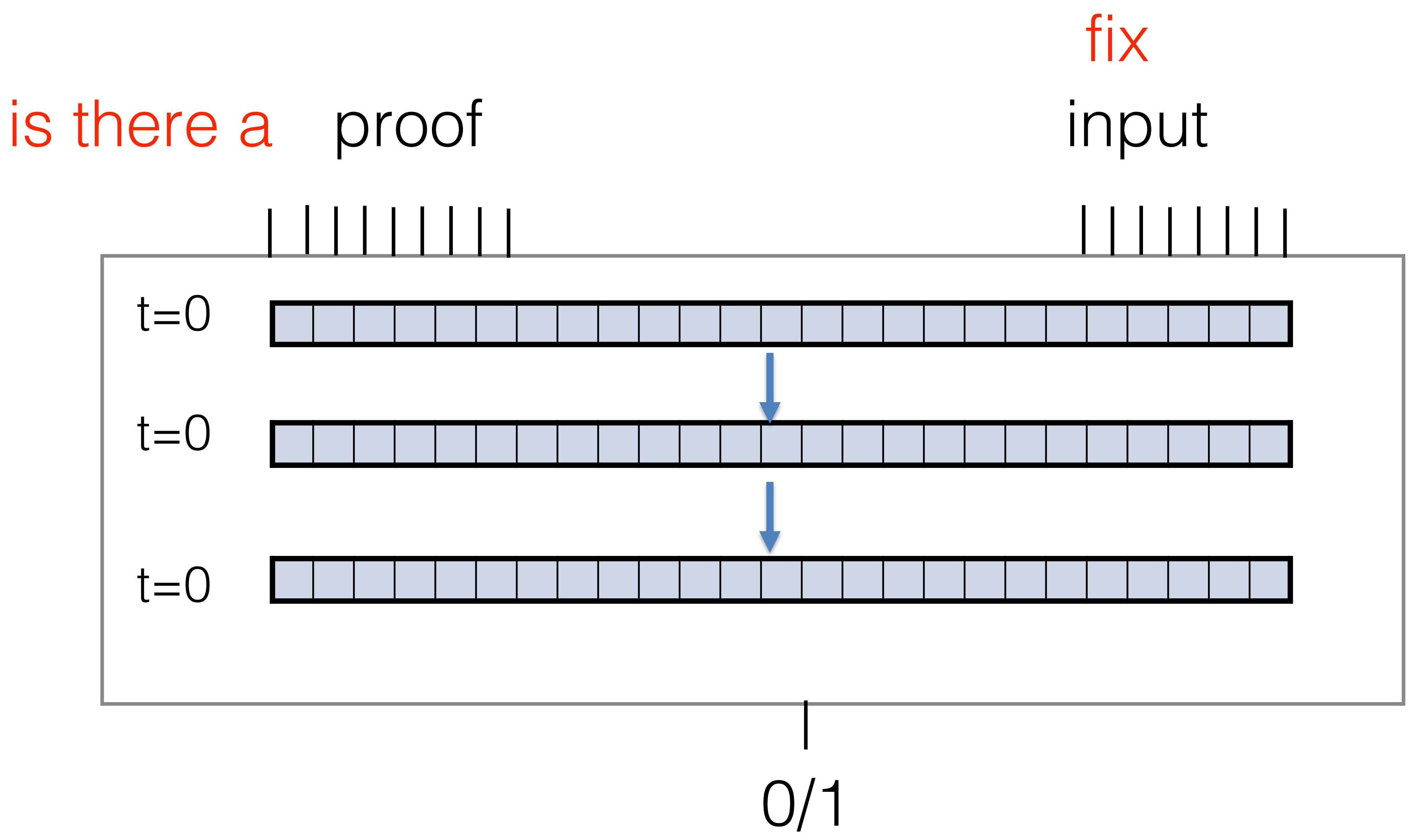
Build a circuit that simulates that TM

Cook Levin Proof? Non-deterministic TMs



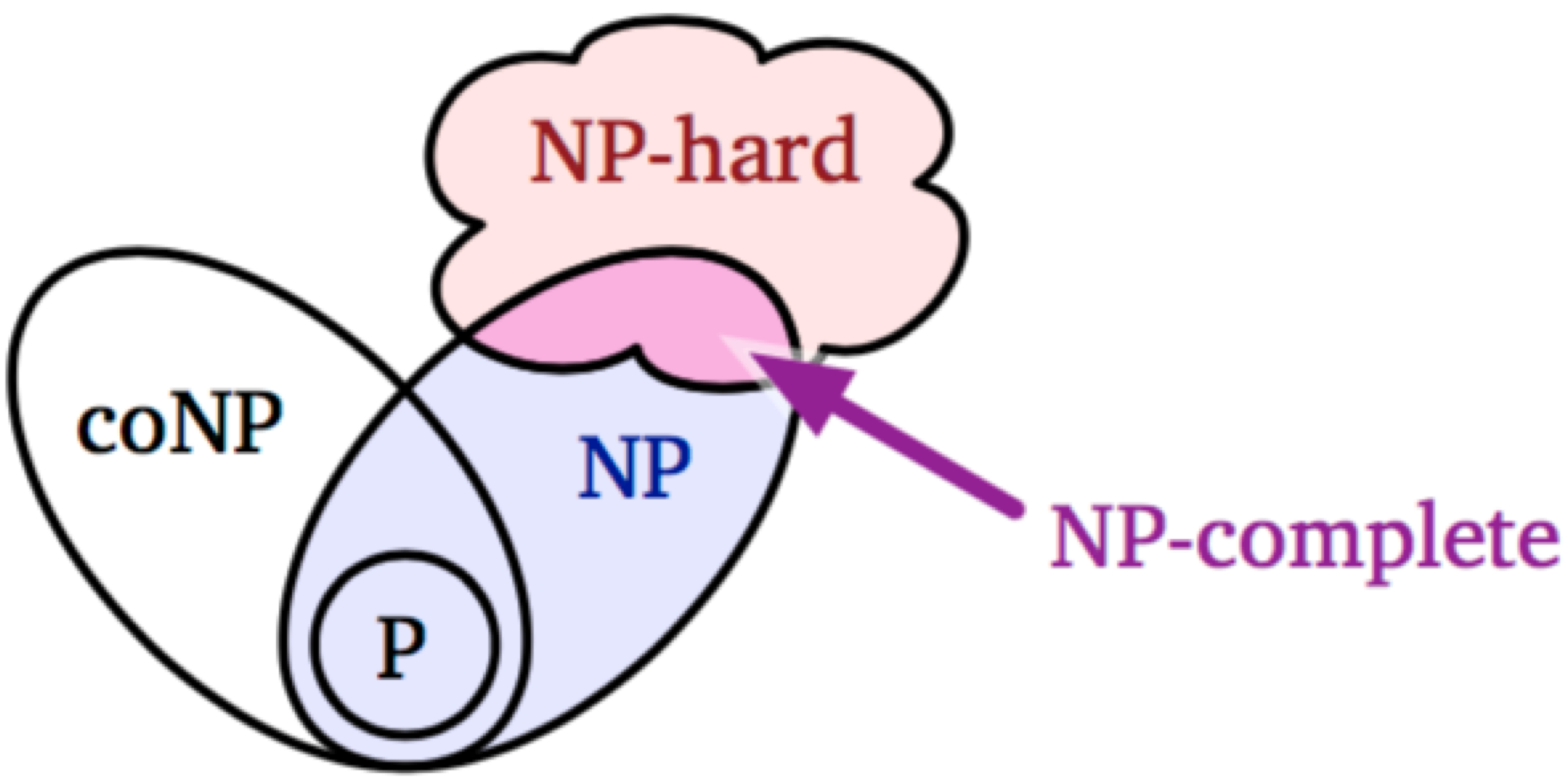


Cook Levin Proof? Non-deterministic TMs



Mickey Mouse Diagram

Problem is NP hard if a poly time algorithm for that problem implies P=NP.



CircuitSAT NP Hard

Every NDTM that accepts some language, equivalent to a circui

If I can solve CircuitSat in poly time,

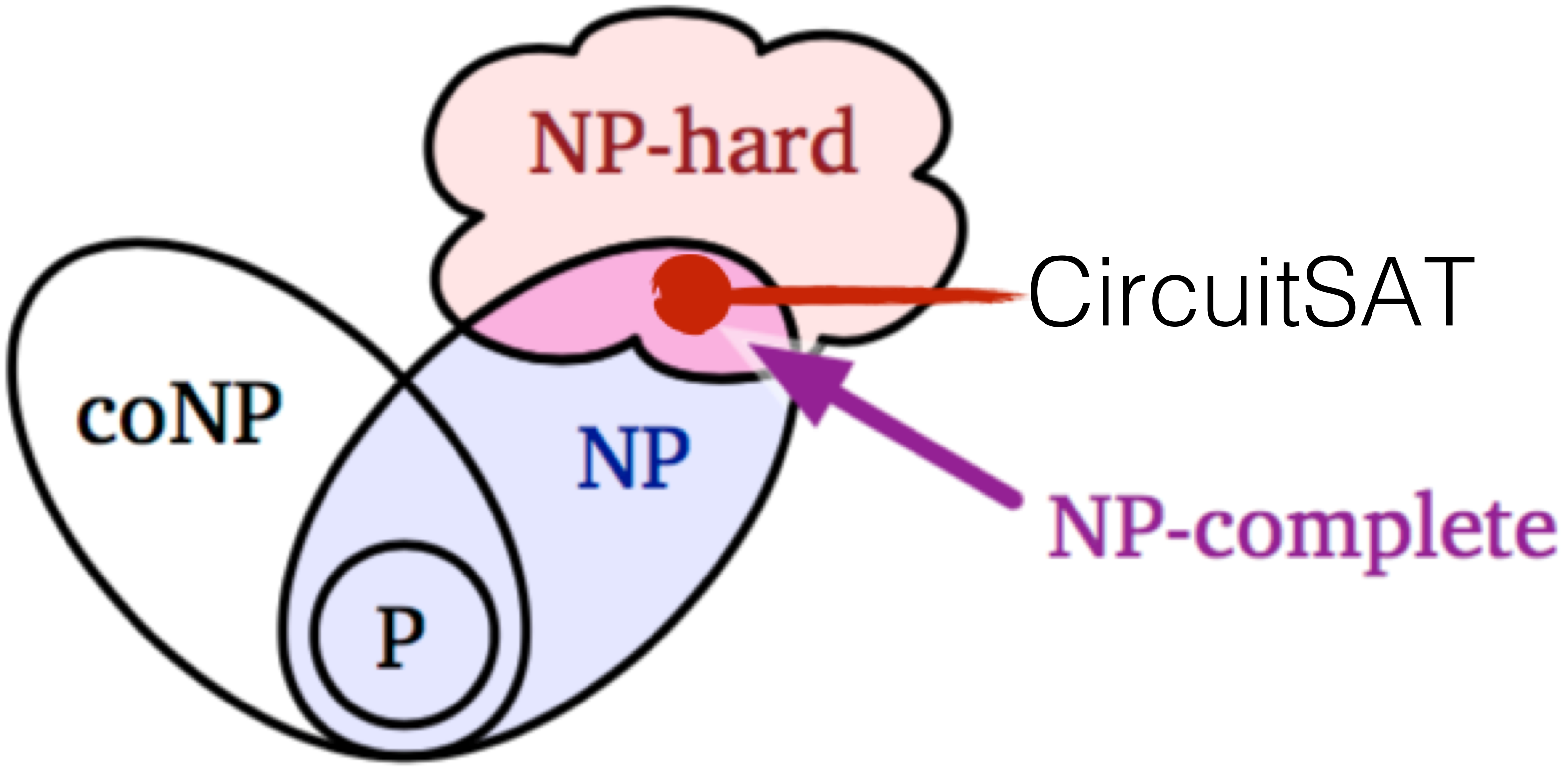
then I can solve any other problem in NP

Step 1: Build giant circuit

Step 2: pass it to the CircuitSAT algorithm

Step 3: profit

Mickey Mouse Diagram



NP hardness

- Assume P≠NP
- Then NP hard means no polynomial time algo!
- Example of reduction: formula SAT
- The only problem we know is NP hard is CircuitSAT, so let's reduce from that.

Formula SAT

Input: boolean formula
Want to decide if there is an assignment to the variables that make it TRUE

$$(a \vee b \vee c \vee \bar{d}) \Leftrightarrow ((b \wedge \bar{c}) \vee \overline{(\bar{a} \Rightarrow d)} \vee (c \neq a \wedge b)),$$

Assume, towards contradiction that SAT can be solved in poly time

NP hardness

- Poly time reduction from CircuitSAT.

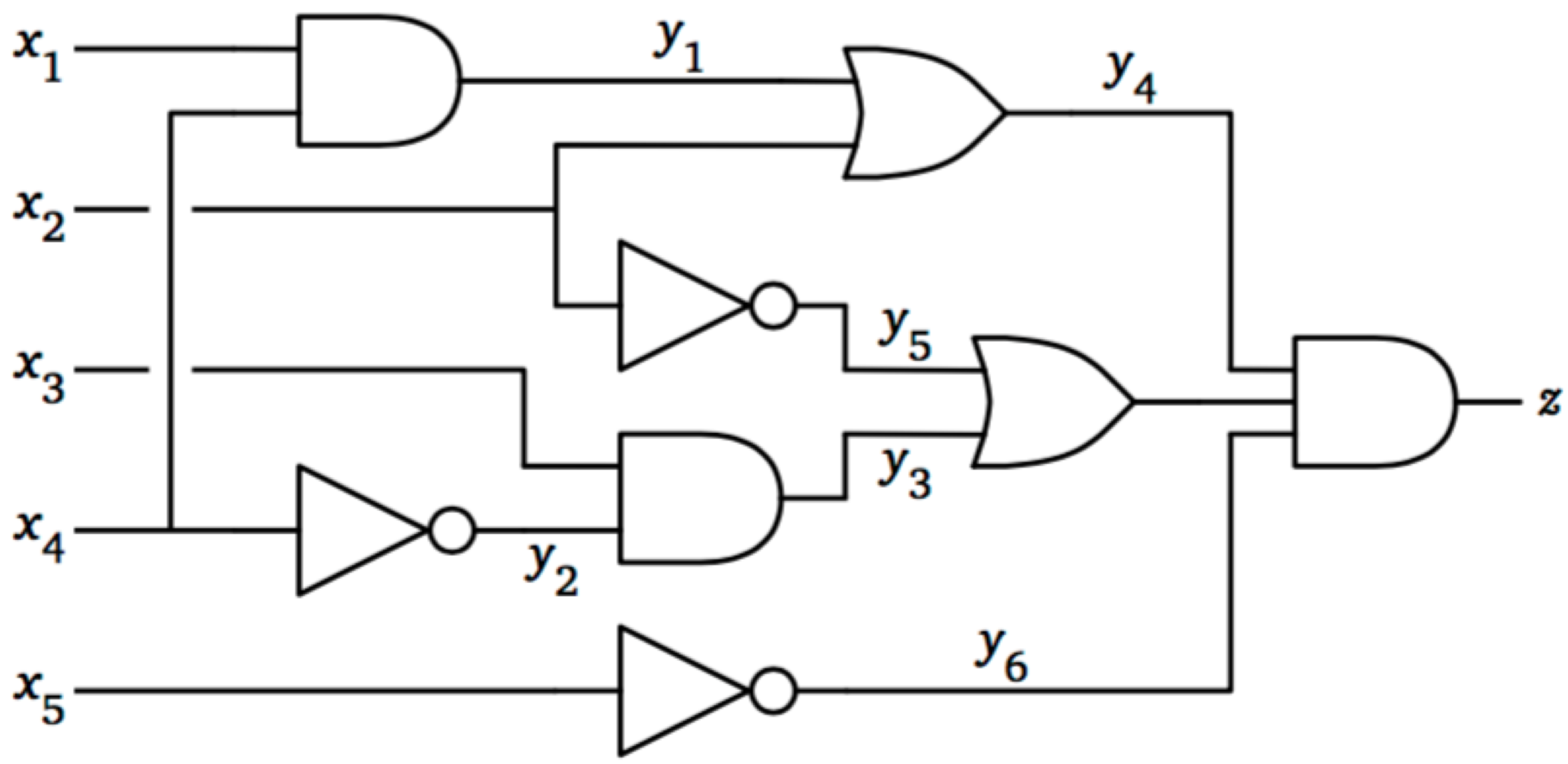
- If there is a poly time algorithm to solve formula SAT, then there is poly time algorithm to solve CircuitSAT

Formula SAT

I am given input a circuit and I want to produce an equivalent formula.

How is the circuit given?

Name the inputs, wires, output.

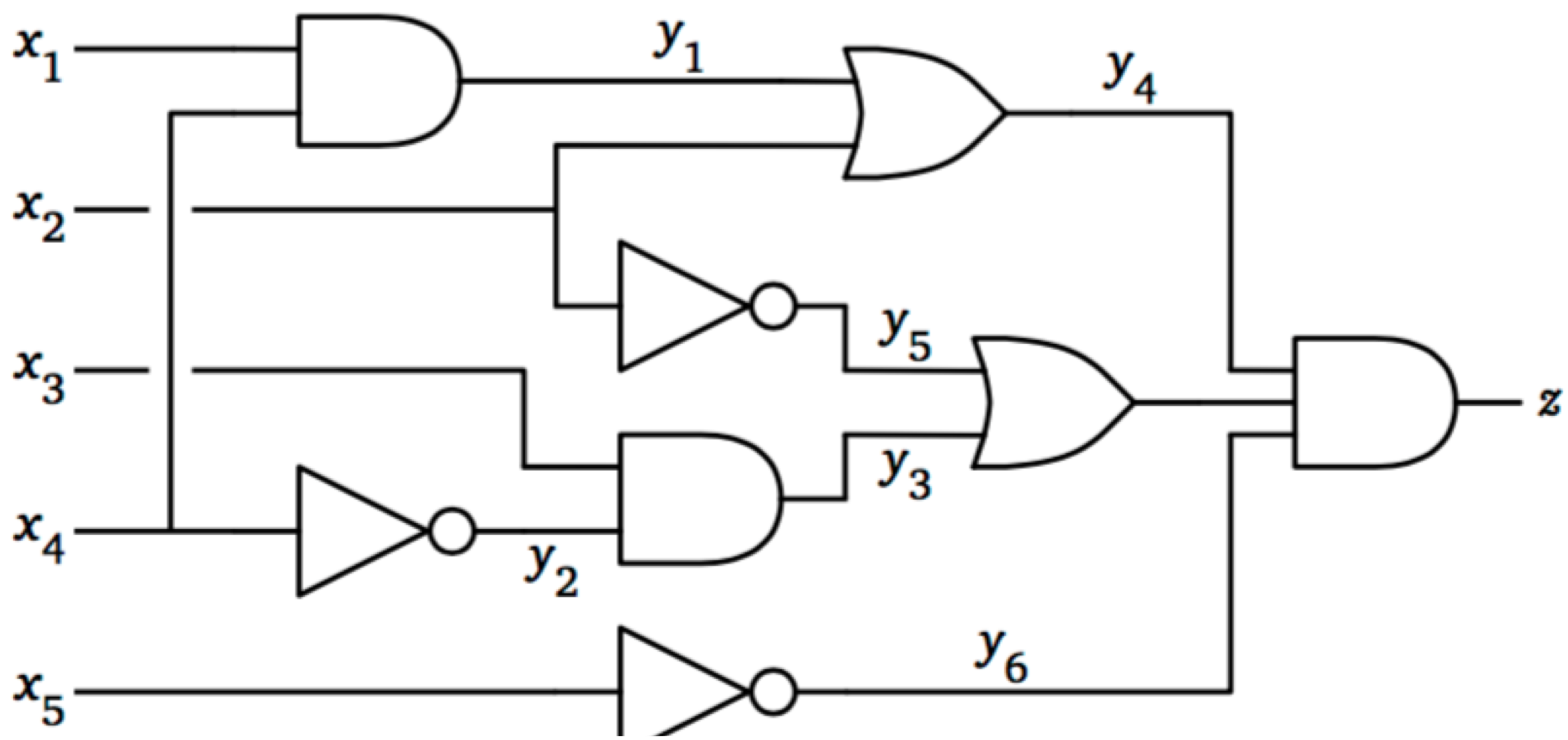


Formula SAT

I am given input a circuit and I want to produce an equivalent formula.

How is the circuit given?

Name the inputs, wires, output.



$$(y_1 = x_1 \land x_4) \land (y_2 = \overline{x_4}) \land (y_3 = x_3 \land y_2) \land (y_4 = y_1 \lor x_2) \land$$
$$(y_5 = \overline{x_2}) \land (y_6 = \overline{x_5}) \land (y_7 = y_3 \lor y_5) \land (z = y_4 \land y_7 \land y_6) \land z$$

NP hardness

- There are inputs to the circuit that force z t be true if and only if there are values to these variables that make the expression true

- I have reduced CircuitSat to formula SAT

- Proof? 2 stages

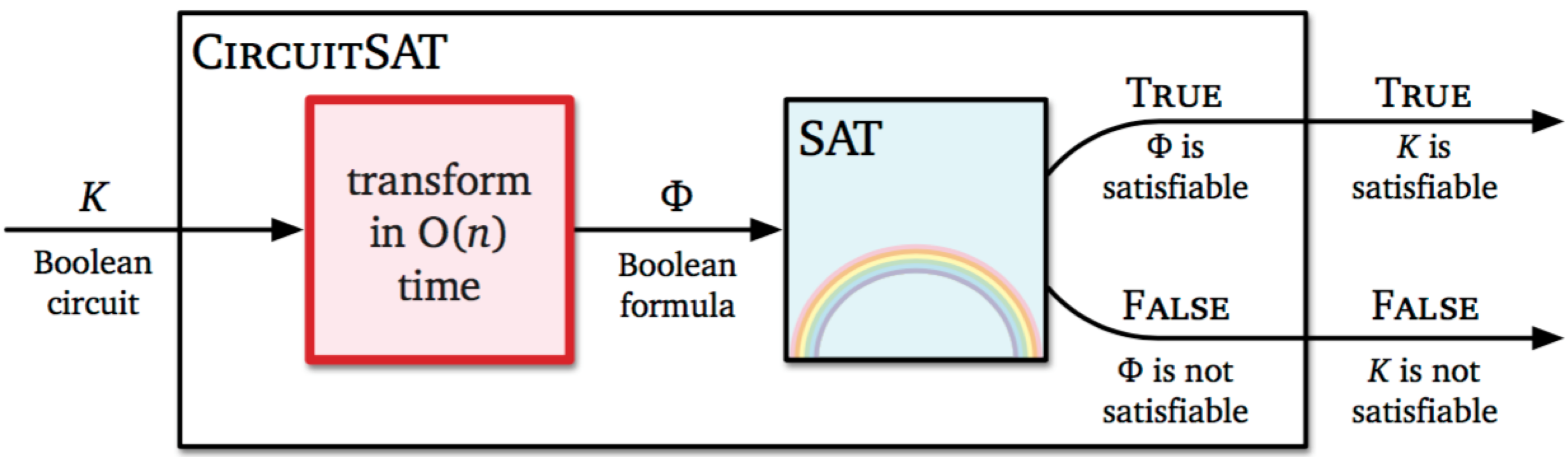
- **Stage 1:** Suppose I can satisfy the circuit, then I can find corresponding values for all the wires, same values satisfy the formula

- **Stage 2:** Suppose I can satisfy the formula, I can pull those values to the wires

NP hardness

- Poly time reduction from CircuitSAT.

- If there is a poly time algorithm to solve formula SAT, then there is poly time algorithm to solve CircuitSAT



NP hardness

- Poly time reduction from CircuitSAT.

- If there is a poly time algorithm to solve formula SAT, then there is poly time algorithm to solve CircuitSAT

```
CIRCUITSAT(K):
    transcribe K into a boolean formula Φ
    return SAT(Φ)            ⟨⟨Magic!!⟩⟩
```

$$T_{\text{CircuitSAT}}(n) \le O(n) + T_{\text{SAT}}(O(n))$$

How to prove NP hardness

- To prove X is NP-hard:

- **Step 1**: Pick a known NP-hard problem Y
- **Step 2:** Assume for the sake of argument, a polynomial time algorithm for X.
- **Step 3**: Derive a polynomial time algorithm for Y, using algorithm for X as subroutine.
- **Step 4**: Contradiction

Reduce Y to X

Reduce FROM the problem
I know about
TO the problem
I am curious about

NP hardness

- Library of NP-hard problems

CircuitSAT

SAT

?

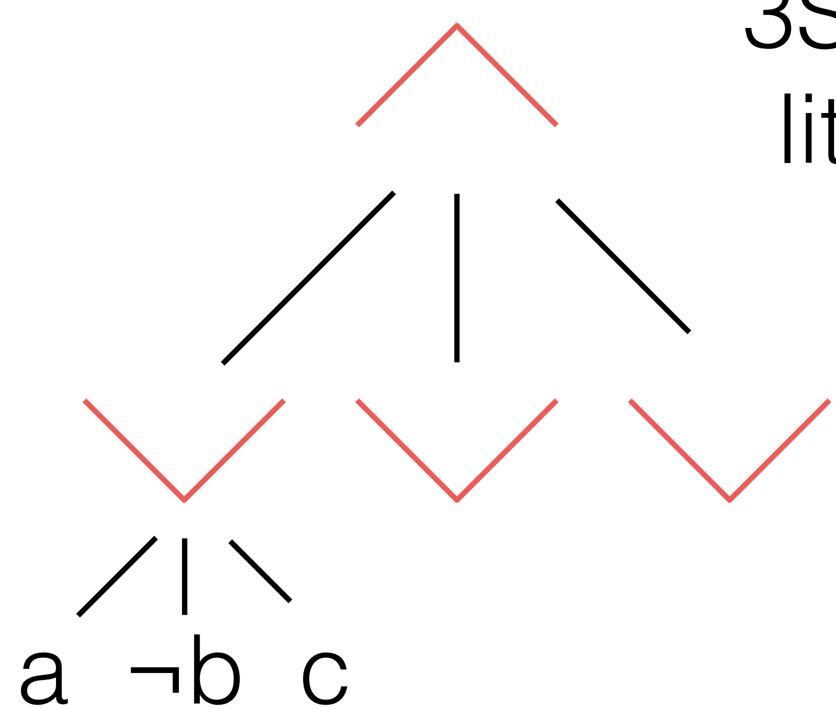
Let's assume the problem is easy
and see what ridiculous consequences follow

3SAT

- Look at boolean formulas in CNF

$$\overbrace{(a \vee b \vee c \vee d)}^{\text{clause}} \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{a} \vee c \vee d) \wedge (a \vee \bar{b})$$

Parse tree:

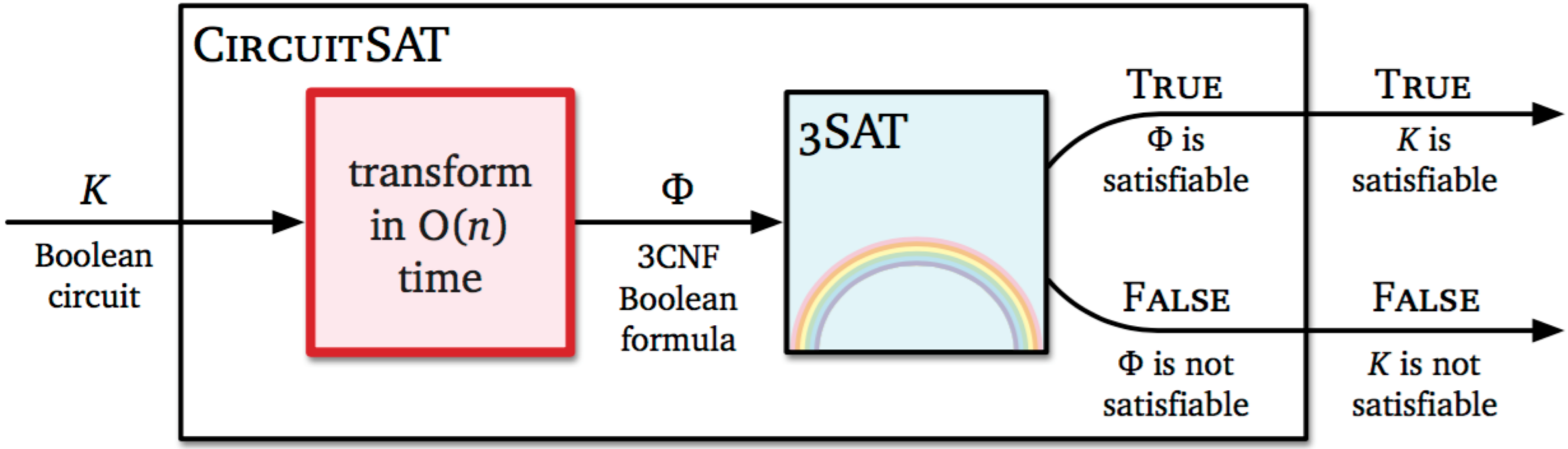


3SAT: exactly three literals per clause!

NP hardness

- Poly time reduction from CircuitSAT.

- If there is a poly time algorithm to solve formula 3SAT, then there is poly time algorithm to solve CircuitSAT



1. *Make sure every* AND *and* OR *gate in K has exactly two inputs.* If any gate has $k > 2$ inputs, replace it with a binary tree of $k-1$ two-input gates. Call the resulting circuit K'.

2. *Transcribe K' into a boolean formula Φ_1 with one clause per gate,* exactly as in our previous reduction to SAT.

3. *Replace each clause in Φ_1 with a CNF formula.* There are only three types of clauses in Φ_1, one for each type of gate in K':

$$\begin{aligned} a = b \wedge c &\longmapsto (a \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee b) \wedge (\bar{a} \vee c) \\ a = b \vee c &\longmapsto (\bar{a} \vee b \vee c) \wedge (a \vee \bar{b}) \wedge (a \vee \bar{c}) \\ a = \bar{b} &\longmapsto (a \vee b) \wedge (\bar{a} \vee \bar{b}) \end{aligned}$$

   Call the resulting CNF formula Φ_2.

4. *Replace each clause in Φ_2 with a 3CNF formula.* Every clause in Φ_2 has at most three literals. We can keep the three-literal clauses as-is. We expand each two-literal clause into two three-literal clauses by introducing a new variable. Finally, we expand any one-literal clause into four three-literal clauses by introducing two new variables.

$$
\begin{aligned}
a \vee b &\longmapsto (a \vee b \vee x) \wedge (a \vee b \vee \bar{x}) \\
a &\longmapsto (a \vee x \vee y) \wedge (a \vee \bar{x} \vee y) \wedge (a \vee x \vee \bar{y}) \wedge (a \vee \bar{x} \vee \bar{y})
\end{aligned}
$$

Call the final 3CNF formula Φ_3.

For example, if we start with the same example circuit we used earlier, we obtain the following 3CNF formula Φ_3.

Call the final 3CNF formula Φ_3.

For example, if we start with the same example circuit we used earlier, we obtain the following 3CNF formula Φ_3.

$$
\begin{aligned}
&(y_1 \vee \overline{x_1} \vee \overline{x_4}) \wedge (\overline{y_1} \vee x_1 \vee z_1) \wedge (\overline{y_1} \vee x_1 \vee \overline{z_1}) \wedge (\overline{y_1} \vee x_4 \vee z_2) \wedge (\overline{y_1} \vee x_4 \vee \overline{z_2}) \\
\wedge\, &(y_2 \vee x_4 \vee z_3) \wedge (y_2 \vee x_4 \vee \overline{z_3}) \wedge (\overline{y_2} \vee \overline{x_4} \vee z_4) \wedge (\overline{y_2} \vee \overline{x_4} \vee \overline{z_4}) \\
\wedge\, &(y_3 \vee \overline{x_3} \vee \overline{y_2}) \wedge (\overline{y_3} \vee x_3 \vee z_5) \wedge (\overline{y_3} \vee x_3 \vee \overline{z_5}) \wedge (\overline{y_3} \vee y_2 \vee z_6) \wedge (\overline{y_3} \vee y_2 \vee \overline{z_6}) \\
\wedge\, &(\overline{y_4} \vee y_1 \vee x_2) \wedge (y_4 \vee \overline{x_2} \vee z_7) \wedge (y_4 \vee \overline{x_2} \vee \overline{z_7}) \wedge (y_4 \vee \overline{y_1} \vee z_8) \wedge (y_4 \vee \overline{y_1} \vee \overline{z_8}) \\
\wedge\, &(y_5 \vee x_2 \vee z_9) \wedge (y_5 \vee x_2 \vee \overline{z_9}) \wedge (\overline{y_5} \vee \overline{x_2} \vee z_{10}) \wedge (\overline{y_5} \vee \overline{x_2} \vee \overline{z_{10}}) \\
\wedge\, &(y_6 \vee x_5 \vee z_{11}) \wedge (y_6 \vee x_5 \vee \overline{z_{11}}) \wedge (\overline{y_6} \vee \overline{x_5} \vee z_{12}) \wedge (\overline{y_6} \vee \overline{x_5} \vee \overline{z_{12}}) \\
\wedge\, &(\overline{y_7} \vee y_3 \vee y_5) \wedge (y_7 \vee \overline{y_3} \vee z_{13}) \wedge (y_7 \vee \overline{y_3} \vee \overline{z_{13}}) \wedge (y_7 \vee \overline{y_5} \vee z_{14}) \wedge (y_7 \vee \overline{y_5} \vee \overline{z_{14}}) \\
\wedge\, &(y_8 \vee \overline{y_4} \vee \overline{y_7}) \wedge (\overline{y_8} \vee y_4 \vee z_{15}) \wedge (\overline{y_8} \vee y_4 \vee \overline{z_{15}}) \wedge (\overline{y_8} \vee y_7 \vee z_{16}) \wedge (\overline{y_8} \vee y_7 \vee \overline{z_{16}}) \\
\wedge\, &(y_9 \vee \overline{y_8} \vee \overline{y_6}) \wedge (\overline{y_9} \vee y_8 \vee z_{17}) \wedge (\overline{y_9} \vee y_8 \vee \overline{z_{17}}) \wedge (\overline{y_9} \vee y_6 \vee z_{18}) \wedge (\overline{y_9} \vee y_6 \vee \overline{z_{18}}) \\
\wedge\, &(y_9 \vee z_{19} \vee z_{20}) \wedge (y_9 \vee \overline{z_{19}} \vee z_{20}) \wedge (y_9 \vee z_{19} \vee \overline{z_{20}}) \wedge (y_9 \vee \overline{z_{19}} \vee \overline{z_{20}})
\end{aligned}
$$

Although this formula may look a lot more ugly and complicated than the original circuit at first glance, it's actually only a constant factor larger—every binary gate in the original