NP-completeness, Cook-Levin, and additional reductions

CS 580

Recall the languages SAT and 3SAT:

3SAT = { ϕ | ϕ is a satisfiable 3CNF formula}.

Exercise: Clique is polynomial time reducible to 3SAT.

Recall the languages SAT and 3SAT:

3SAT = { ϕ | ϕ is a satisfiable 3CNF formula}.

Exercise: Clique is polynomial time reducible to 3SAT.

What we need to show: Given as input a Clique instance, i.e. an undirected graph G = (V,E) and a number k, we can construct in polynomial time a 3SAT formula ϕ such that the graph G has a k-clique if and only if ϕ has a satisfying assignment.

Recall the languages SAT and 3SAT:

3SAT = { ϕ | ϕ is a satisfiable 3CNF formula}.

Exercise: Clique is polynomial time reducible to 3SAT.

Proof: Given a graph G = (V, E) and a number k, create variables $x_{i,v}$ for every 1 ≤ i ≤ k and every v ∈ V , with the interpretation that $x_{i,v} = True$ if node v is the i-th vertex in the clique. We encode the conditions:

Recall the languages SAT and 3SAT:

3SAT = { ϕ | ϕ is a satisfiable 3CNF formula}.

Exercise: Clique is polynomial time reducible to 3SAT.

Proof: Given a graph G = (V, E) and a number k, create variables $x_{i,v}$ for every 1 ≤ i ≤ k and every v ∈ V , with the interpretation that $x_{i,v} = True$ if node v is the i-th vertex in the clique. We encode the conditions:

1. For each i, there is an i-th vertex in the clique: $\vee_{v \in V} x_{i,v}$.

Recall the languages SAT and 3SAT:

3SAT = { ϕ | ϕ is a satisfiable 3CNF formula}.

Exercise: Clique is polynomial time reducible to 3SAT.

Proof: Given a graph G = (V, E) and a number k, create variables $x_{i,v}$ for every $1 \le i \le k$ and every $v \in V$, with the interpretation that $x_{i,v} = True$ if node v is the i-th vertex in the clique. We encode the conditions:

1. For each i, there is an i-th vertex in the clique: $\bigvee_{v \in V} x_{i,v}$.
2. For all i, j, the i-th vertex is different from the j-th vertex: for each $v \in V$, add $\bar{x}_{i,v} \vee \bar{x}_{j,v}$.

Recall the languages SAT and 3SAT:

3SAT = { ϕ | ϕ is a satisfiable 3CNF formula}.

Exercise: Clique is polynomial time reducible to 3SAT.

Proof: Given a graph G = (V, E) and a number k, create variables $x_{i,v}$ for every $1 \leq i \leq k$ and every $v \in V$, with the interpretation that $x_{i,v} = True$ if node v is the i-th vertex in the clique. We encode the conditions:

1. For each i, there is an i-th vertex in the clique: $\vee_{v \in V} x_{i,v}$.
2. For all i, j, the i-th vertex is different from the j-th vertex: for each $v \in V$, add $\bar{x}_{i,v} \vee \bar{x}_{j,v}$.
3. For each non-edge $(u, v) \notin E$, u and v cannot both belong to the clique: for each i, j, we have $\bar{x}_{i,u} \vee \bar{x}_{j,v}$.

NP-completeness

Definition (NP-complete): A language L is NP-complete if:

- $L \in NP$
- Every language $A \in NP$ is poly-time reducible to L.

Recall: A language $A \subseteq \Sigma^*$ is polynomial time (mapping) reducible to a language $B \subseteq \Sigma^*$ if there exists a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that for every $x \in \Sigma^*$, we have:

$$x \in A \text{ if and only if } f(x) \in B.$$

The function f is the **reduction**.

NP-completeness

Definition (NP-complete): A language L is NP-complete if:

- $L \in NP$
- Every language $A \in NP$ is poly-time reducible to L.

Theorem: If a language A is NP-complete and $A \in P$, then $P = NP$.

Proof: Exercise.

NP-completeness

Definition (NP-complete): A language L is NP-complete if:

- $L \in NP$
- Every language $A \in NP$ is poly-time reducible to L.

Theorem: If a language A is NP-complete and $A \in P$, then $P = NP$.

Proof: Let $L \in NP$ be a language.

Since A is NP-complete, there exists a poly-time reduction f such that for every $x \in \Sigma^*$, we have: $x \in L$ if and only if $f(x) \in A$. Since $A \in P$, there is a poly-time TM M to decide A.

Then we can get a poly-time algorithm for L. On each input $x \in \Sigma^*$:

- Compute $f(x) = y$
- Run the TM M on y and output the same answer as M.

Definition (NP-complete): A language L is NP-complete if:

- L $\in$ NP
- Every language A $\in$ NP is poly-time reducible to L.

Theorem: If a language A is NP-complete and A $\leq_p$ B [i.e. A is poly-time reducible to B] for B $\in$ NP, then B is NP-complete.

Proof: Exercise.

Definition (NP-complete): A language L is NP-complete if:

- $L \in NP$
- Every language $A \in NP$ is poly-time reducible to L.

Theorem: If a language A is NP-complete and $A \leq_p B$ [i.e. A is poly-time reducible to B] for $B \in NP$, then B is NP-complete.

Proof: Since A is NP-complete, every language $L \in NP$ reduces to A in polynomial time. Given such a language L, let f be the poly-time computable function that reduces L to A. Suppose g is the polynomial time computable function that reduces A to B. Then the composition gives a poly-time reduction from L to B.

Definition (NP-complete): A language L is NP-complete if:

- L ∈ NP
- Every language A ∈ NP is poly-time reducible to L.

Theorem: If a language A is NP-complete and $A \leq_p B$ [i.e. A is poly-time reducible to B] for B ∈ NP, then B is NP-complete.

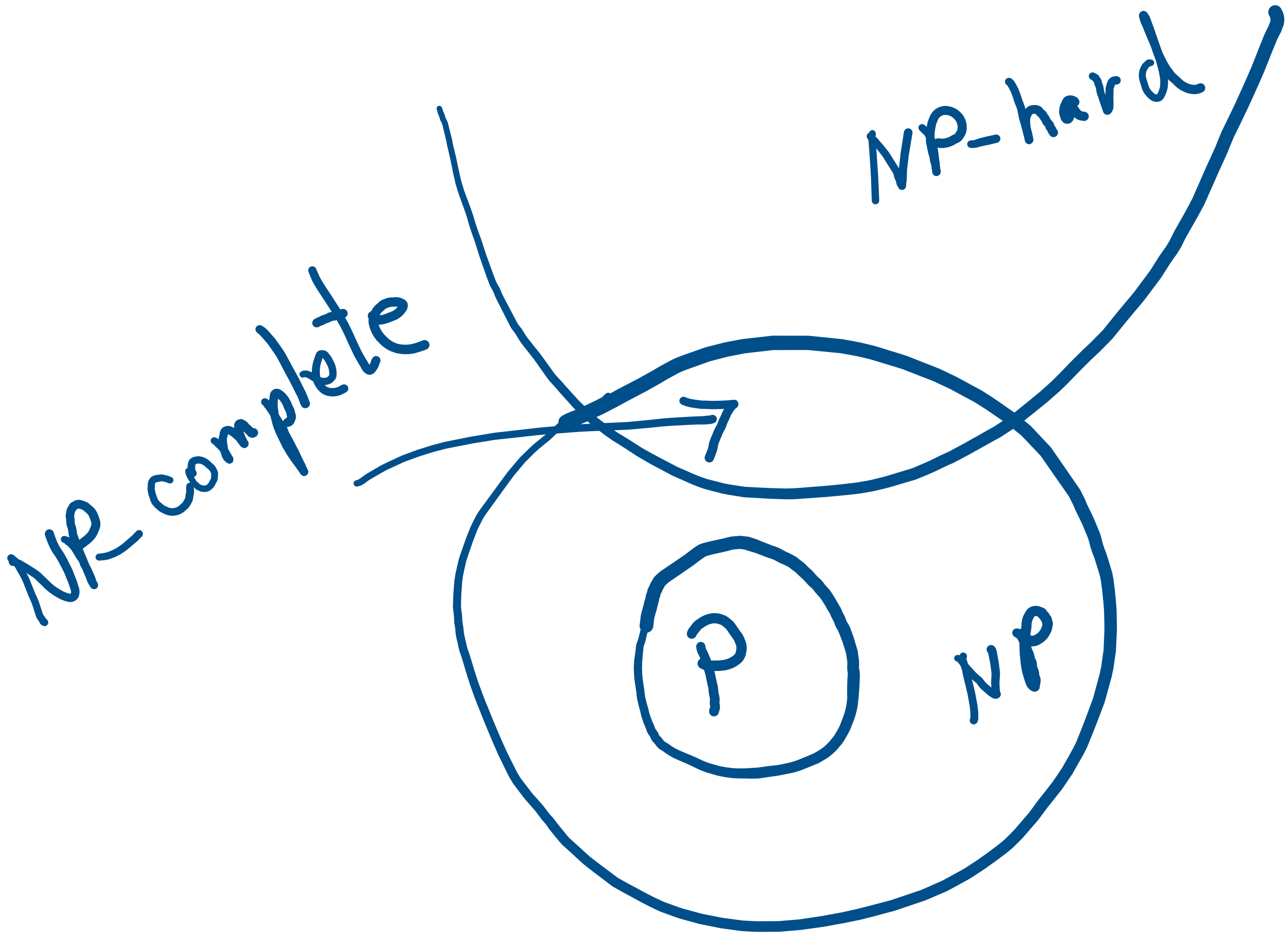
Proof: Since A is NP-complete, every language L ∈ NP reduces to A in polynomial time. Given such a language L, let f be the poly-time computable function that reduces L to A. Suppose g is the polynomial time computable function that reduces A to B. Then the composition gives a poly-time reduction from L to B.

$$\begin{tikzcd} L \arrow[r, "g"] \arrow[rr, bend right] & A \arrow[r, "h"] & B \end{tikzcd}$$

Let f be given by $f(x) = h(g(x))$

Definition (NP-hard): A language L is NP-hard if:

- Every language $A \in NP$ is poly-time reducible to L.



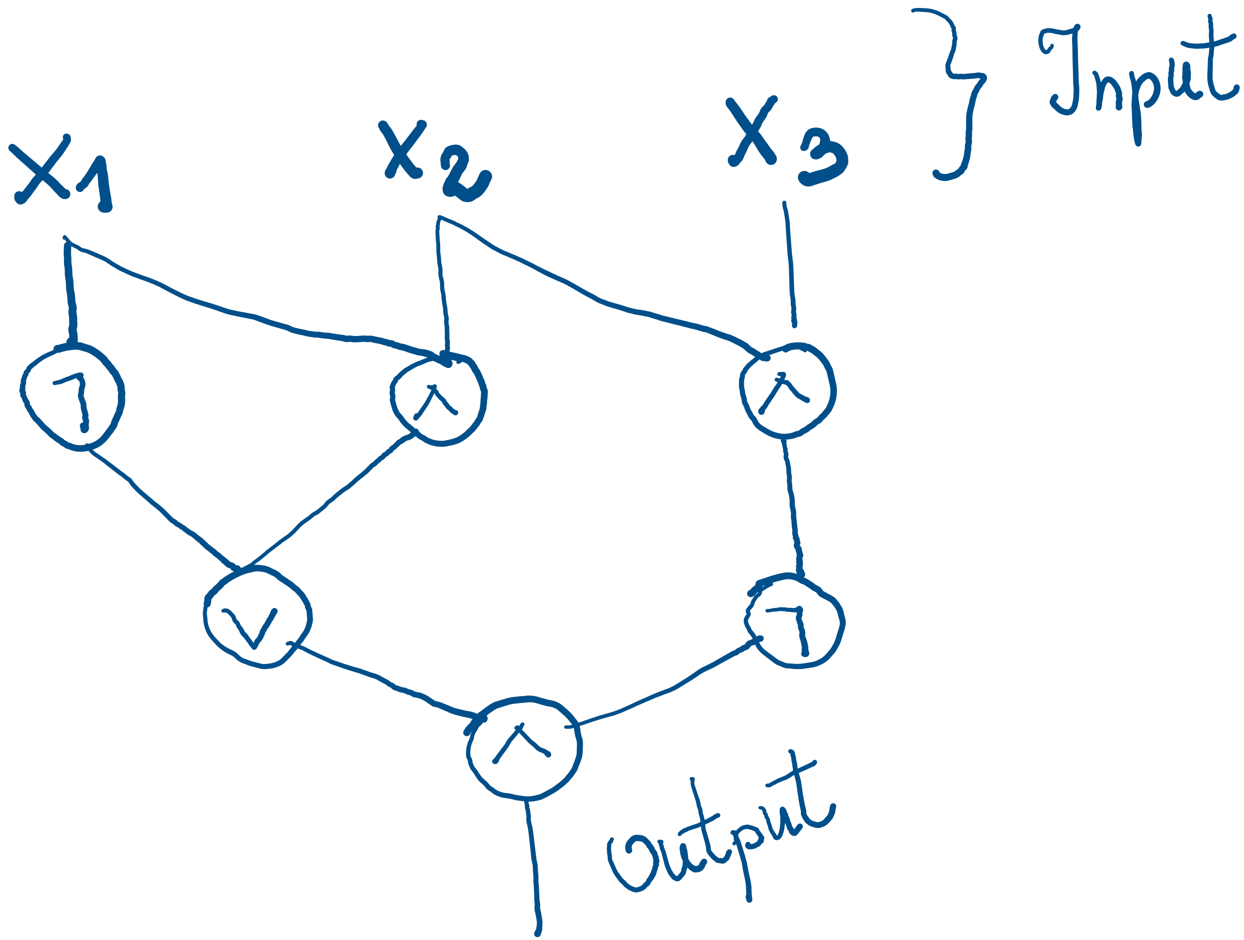
Definition (Boolean circuits): A Boolean circuit is a collection of gates and inputs connected by wires (no cycles). Gates do AND, OR, NOT operations.

Note: All other bit operations can be obtained by combining AND, OR, NOT.

Definition (Boolean circuits): A Boolean circuit is a collection of gates and inputs connected by wires (no cycles). Gates do AND, OR, NOT operations.

Note: All other bit operations can be obtained by combining AND, OR, NOT.

Example:



Definition (Boolean circuits): A Boolean circuit is a collection of gates and inputs connected by wires (no cycles). Gates do AND, OR, NOT operations.

Note: All other bit operations can be obtained by combining AND, OR, NOT.

Function computed by the circuit: To each circuit C we can associate a function $f_C : \{0,1\}^n \to \{0,1\}$, where if C outputs b on input $(x_1, \dots, x_n)$, then $f_C(x_1, \dots, x_n) = b$.

Definition (Boolean circuits): A Boolean circuit is a collection of gates and inputs connected by wires (no cycles). Gates do AND, OR, NOT operations.

Note: All other bit operations can be obtained by combining AND, OR, NOT.

Function computed by the circuit: To each circuit C we can associate a function $f_C : \{0,1\}^n \to \{0,1\}$, where if C outputs b on input $(x_1, \dots, x_n)$, then $f_C(x_1, \dots, x_n) = b$.

Theorem: Let $t : N \to N$ be a function, where $t(n) \geq n$ [i.e. the TM reads the input]. If A ∈ $DTIME\big(t(n)\big)$, then for each $\mathrm{n} \in N$ there is a circuit $\mathrm{C_n}$ of size $\mathrm{O}\big(t(n)^2\big)$ such that for every $x = x_1 \dots x_n \in \{0,1\}^n$, we have $x \in A$ if and only if $C_n(x) = 1$.

Proof: We modify every TM considered so that when it's about to accept, it cleans the tape.

Proof: We modify every TM considered so that when it's about to accept, it cleans the tape.

Notation: write uqv, to mean that the TM is in state q, the tape content to the left of the tape head is u, the head is on the first letter of v, and the tape content to the right of the tape head (including it) is v.

Proof: We modify every TM considered so that when it's about to accept, it cleans the tape.

Notation: write uqv, to mean that the TM is in state q, the tape content to the left of the tape head is u, the head is on the first letter of v, and the tape content to the right of the tape head (including it) is v.

We will make some composite symbols to capture the state and symbol under the tape head together:

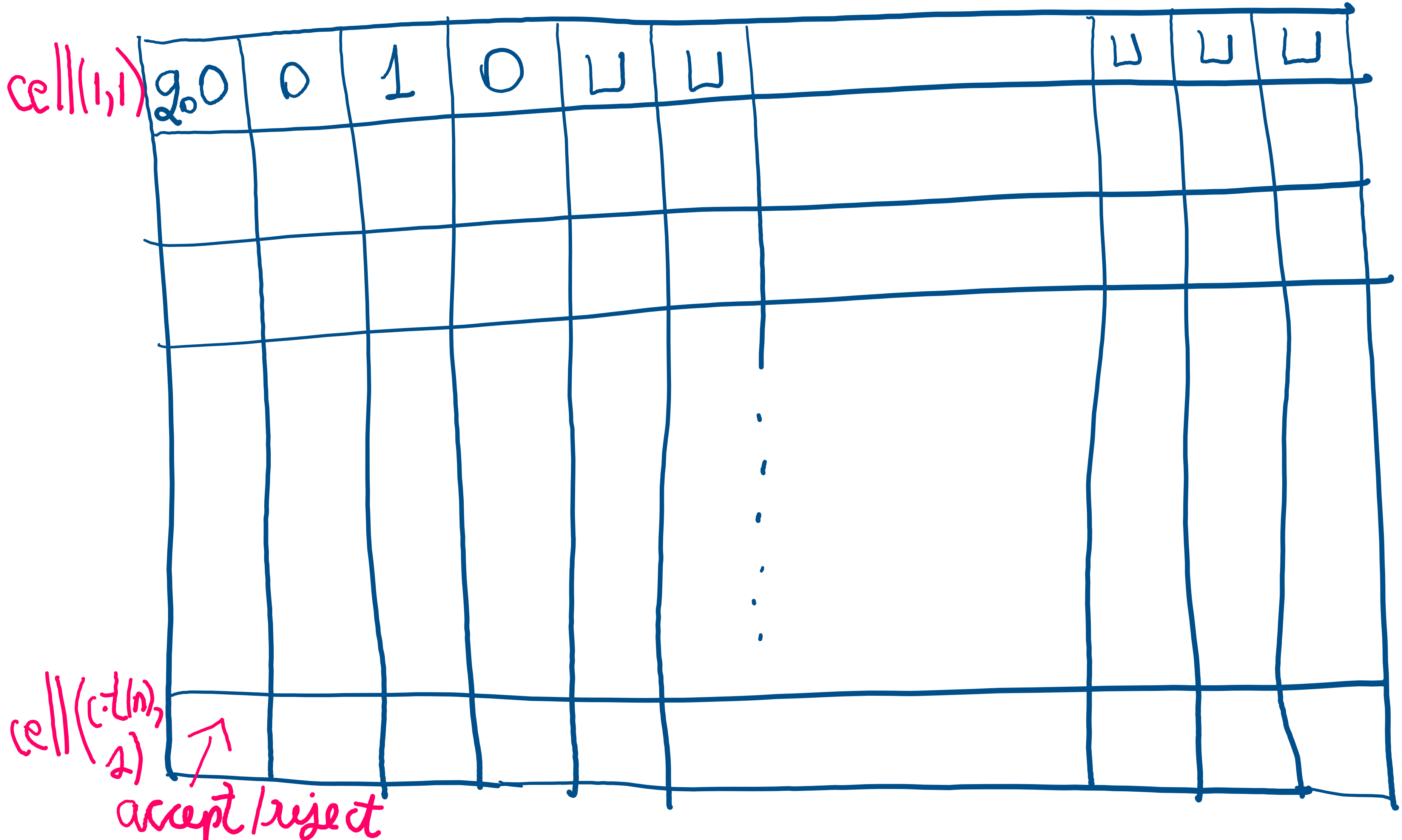
$$\underbrace{0010}_{\text{tape contents to the left of the head.}}\boxed{q\,1}\,110101$$

composite symbol

Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

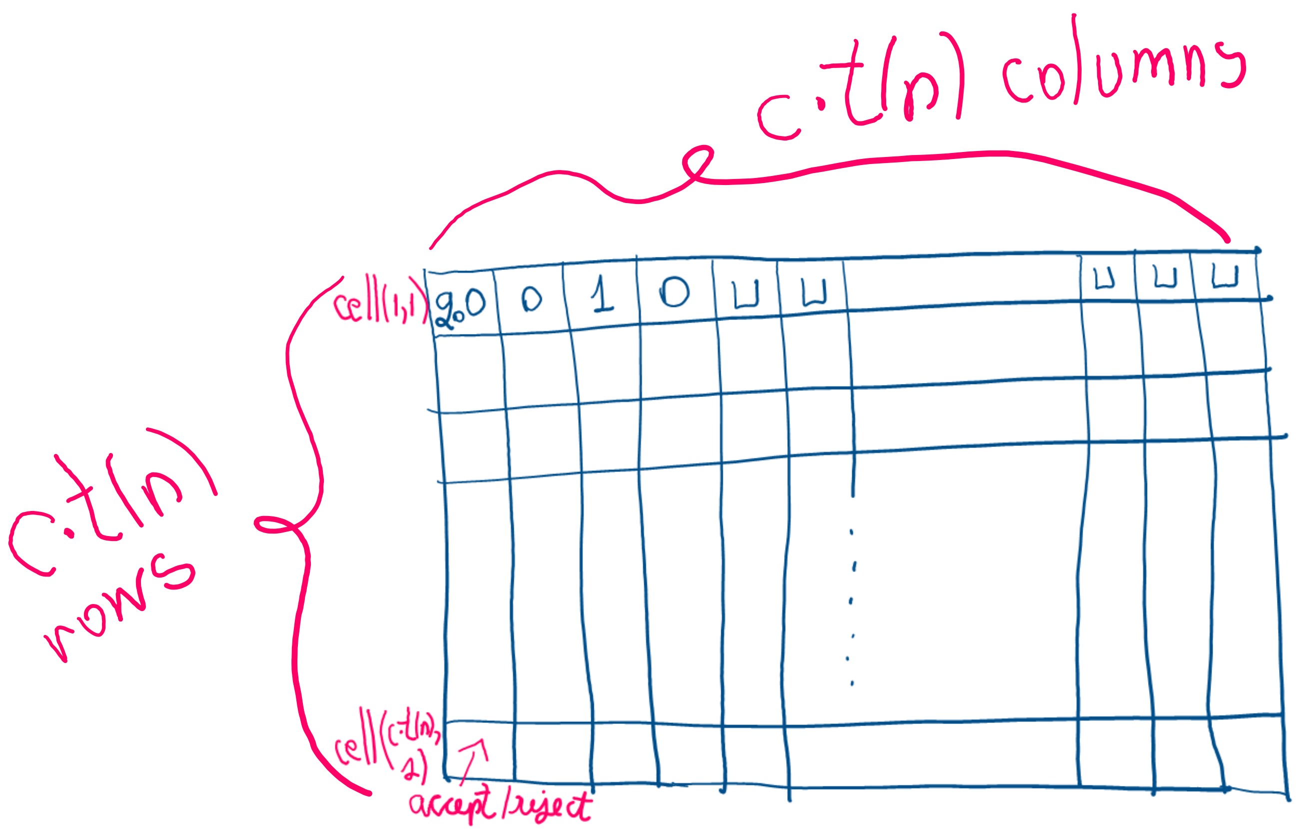
We draw a table to visualize the execution of the TM M.



Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We draw a table to visualize the execution of the TM M.

Each row represents a ***configuration*** of the TM.

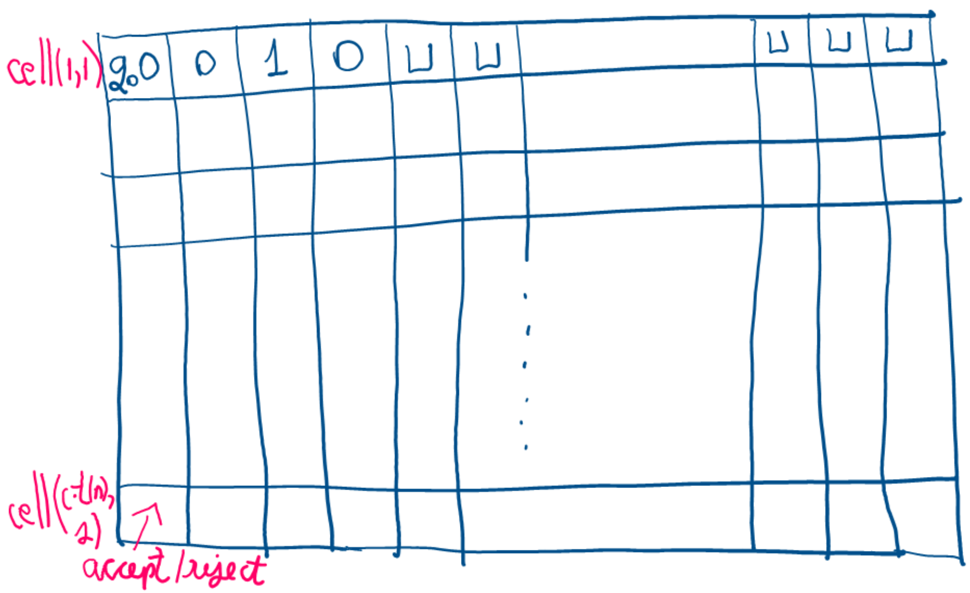


Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We draw a table to visualize the execution of the TM M.

Each row represents a ***configuration*** of the TM.

Question: How is the content of each cell determined?



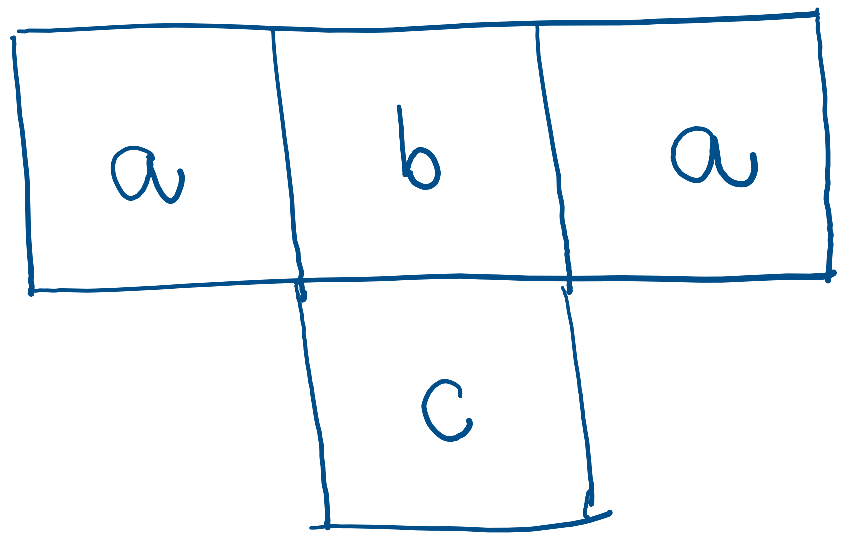
Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We draw a table to visualize the execution of the TM M.

Each row represents a *configuration* of the TM.

Question: How is the content of each cell determined?

Observation: if we know cell(i-1,j-1), cell(i-1,j), and cell (i-1,j+1), we also know cell(i,j) by the way the transition function δ of the machine works.



Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We can begin constructing the circuit C_n.

Let k = number of elements in $\Gamma \cup (\mathrm{Q} \times \Gamma)$ [i.e. one for each tape symbol, and also one for each combined symbol (state, symbol under tape head)].

Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We can begin constructing the circuit C_n.

Let k = number of elements in $\Gamma \cup (\mathrm{Q} \times \Gamma)$ [i.e. one for each tape symbol, and also one for each combined symbol (state, symbol under tape head)].

We create k “lights” for each cell in the tableau: $k\ t(n)^2\ C^2$ in total.

Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We can begin constructing the circuit C_n.

Let k = number of elements in $\Gamma \cup (\mathrm{Q} \times \Gamma)$ [i.e. one for each tape symbol, and also one for each combined symbol (state, symbol under tape head)].

We create k "lights" for each cell in the tableau: $k\ t(n)^2\ C^2$ in total.

The condition of the lights in a cell indicates the contents:

- E.g., if light[i,j,s] is on, then cell[i,j] contains the symbol s.

Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We can begin constructing the circuit C_n.

Let k = number of elements in $\Gamma \cup (\mathrm{Q} \times \Gamma)$ [i.e. one for each tape symbol, and also one for each combined symbol (state, symbol under tape head).

We create k “lights” for each cell in the tableau: $k\ t(n)^2\ C^2$ in total.

The condition of the lights in a cell indicates the contents:

- E.g., if light[i,j,s] is on, then cell[i,j] contains the symbol s.

Suppose the cells cell[i-1,j-1], cell[i-1,j], and cell[i-1,j+1], contain the symbols a,b, and c, respectively. Moreover, suppose δ prescribes that cell[i,j] should then contain the symbol s.

Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

We can begin constructing the circuit C_n.

Let k = number of elements in $\Gamma \cup (\mathrm{Q} \times \Gamma)$ [i.e. one for each tape symbol, and also one for each combined symbol (state, symbol under tape head).

We create k “lights” for each cell in the tableau: $k\ t(n)^2\ C^2$ in total.

The condition of the lights in a cell indicates the contents:

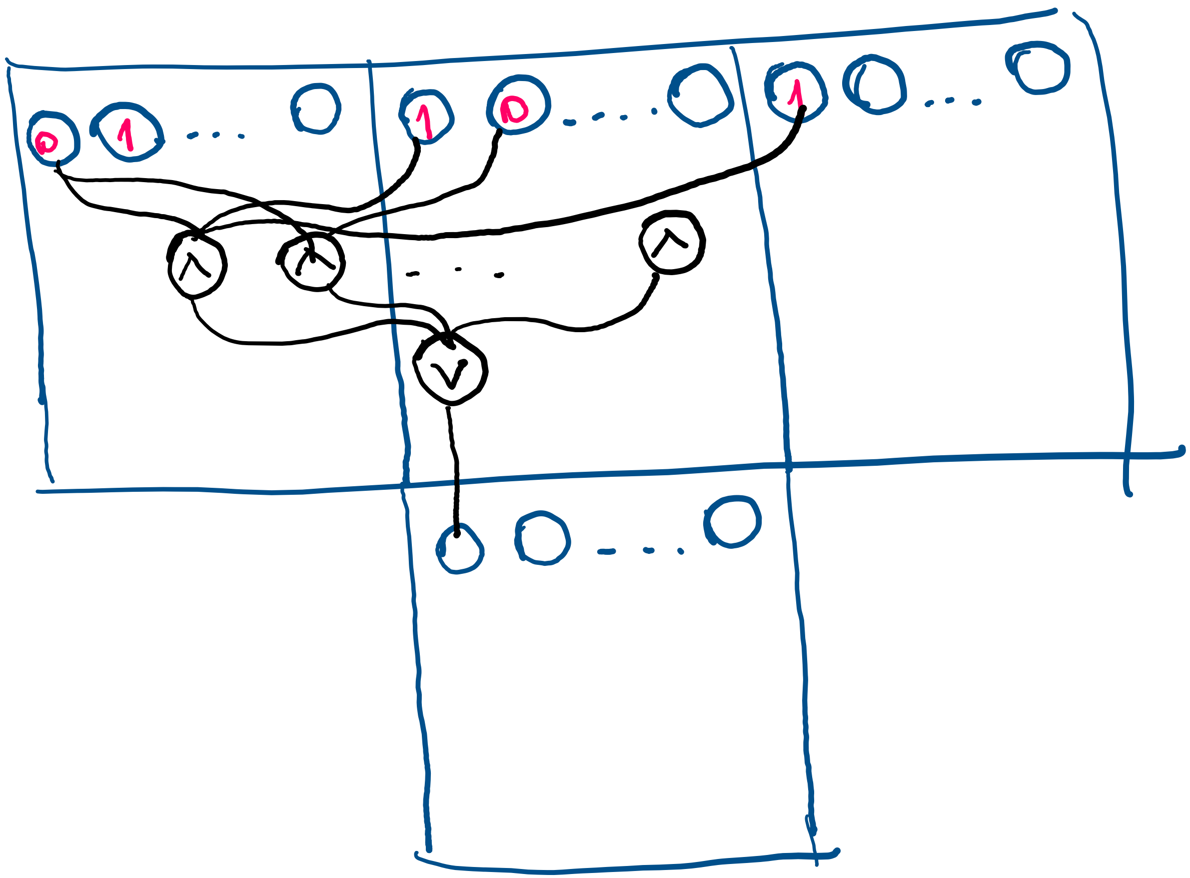
- E.g., if light[i,j,s] is on, then cell[i,j] contains the symbol s.

Suppose the cells cell[i-1,j-1], cell[i-1,j], and cell[i-1,j+1], contain the symbols a,b, and c, respectively. Moreover, suppose δ prescribes that cell[i,j] should then contain the symbol s.

Then we can wire the circuit so that if light[i-1,j-1,a], light[i-1,j,b], and light[i-1,j+1,c] are on, then light[i,j,s] is on.

Proof (cont): Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

Then we can wire the circuit so that if light[i-1,j-1,a], light[i-1,j,b], and light[i-1,j+1,c] are on, then light[i,j,s] is on.



Proof (cont): Let A ∈ $DTIME\left(t(n)\right)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

Then we can wire the circuit so that if light[i-1,j-1,a], light[i-1,j,b], and light[i-1,j+1,c] are on, then light[i,j,s] is on.

Proof (cont): Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

Then we can wire the circuit so that if light[i-1,j-1,a], light[i-1,j,b], and light[i-1,j+1,c] are on, then light[i,j,s] is on.

How should we wire the cells in row 1, e.g. light$[1,1,q_0 1]$?

Proof (cont): Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

Then we can wire the circuit so that if light[i-1,j-1,a], light[i-1,j,b], and light[i-1,j+1,c] are on, then light[i,j,s] is on.

The cells in row 1 have no predecessors and are handled in a special way:

- light[$1,1,q_0 1$] is connected to input x_1. Since the start state is q_0 and the head starts at x_1.
- light[$1,1,q_0 0$] ?

Proof (cont): Let A ∈ $DTIME\big(t(n)\big)$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

Then we can wire the circuit so that if light[i-1,j-1,a], light[i-1,j,b], and light[i-1,j+1,c] are on, then light[i,j,s] is on.

The cells in row 1 have no predecessors and are handled in a special way:

- light$[1,1,q_0 1]$ is connected to input x_1. Since the start state is q_0 and the head starts at x_1.
- light$[1,1,q_0 0]$? connected through a NOT gate to input x_1. Since the start state is q_0 and the head starts at x_1.
- Similarly, the next ones.
- In particular light$[1,n+1,$⊔], … , light$[1,C\ * t(n),$⊔] are on.

What is the output gate?

Proof (cont): Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

Then we can wire the circuit so that if light[i-1,j-1,a], light[i-1,j,b], and light[i-1,j+1,c] are on, then light[i,j,s] is on.

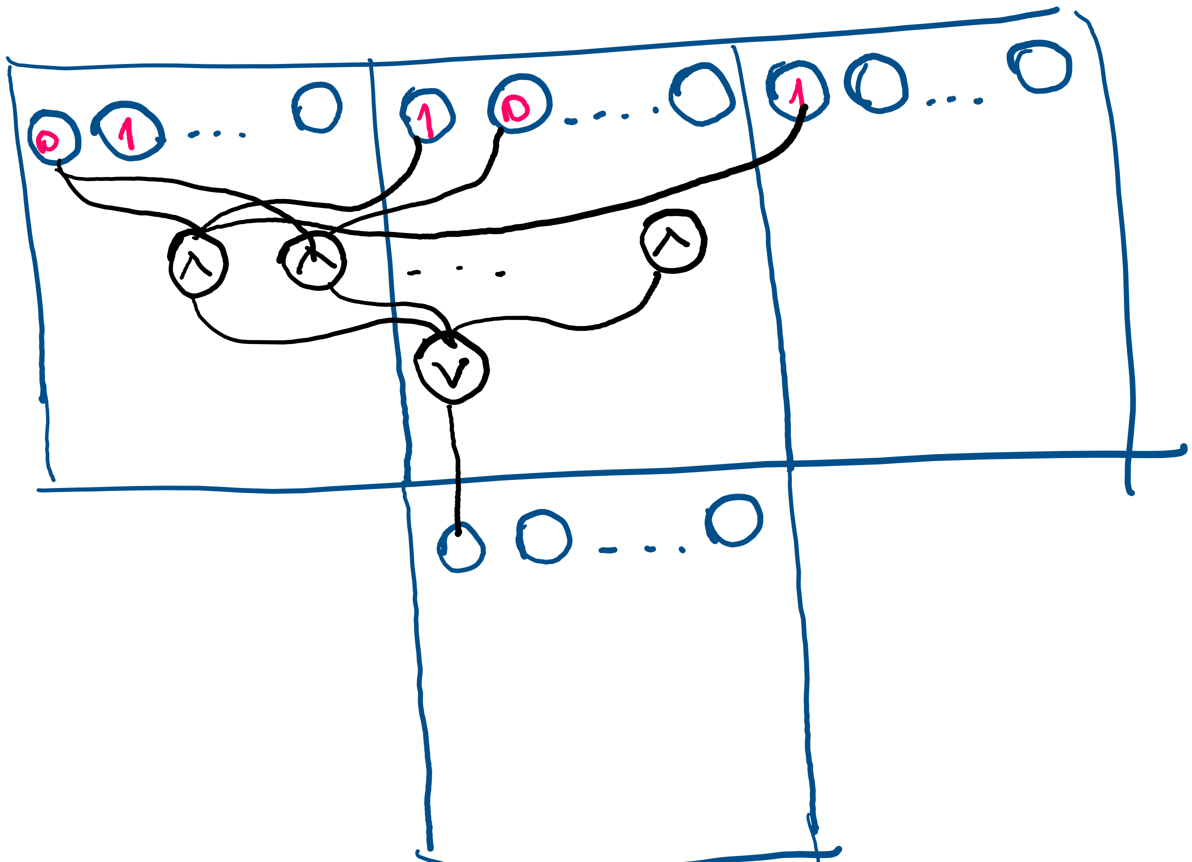
The cells in row 1 have no predecessors and are handled in a special way:

- light$[1,1, q_0 1]$ is connected to input x_1. Since the start state is q_0 and the head starts at x_1.
- light$[1,1, q_0 0]$? connected through a NOT gate to input x_1. Since the start state is q_0 and the head starts at x_1.
- Similarly, the next ones.
- In particular light$[1, n+1, ⊔]$, … , light$[1, C \ * t(n), ⊔]$ are on.

The output gate is the one attached to light[C $* \, t(n), 1, q_{accept}$].

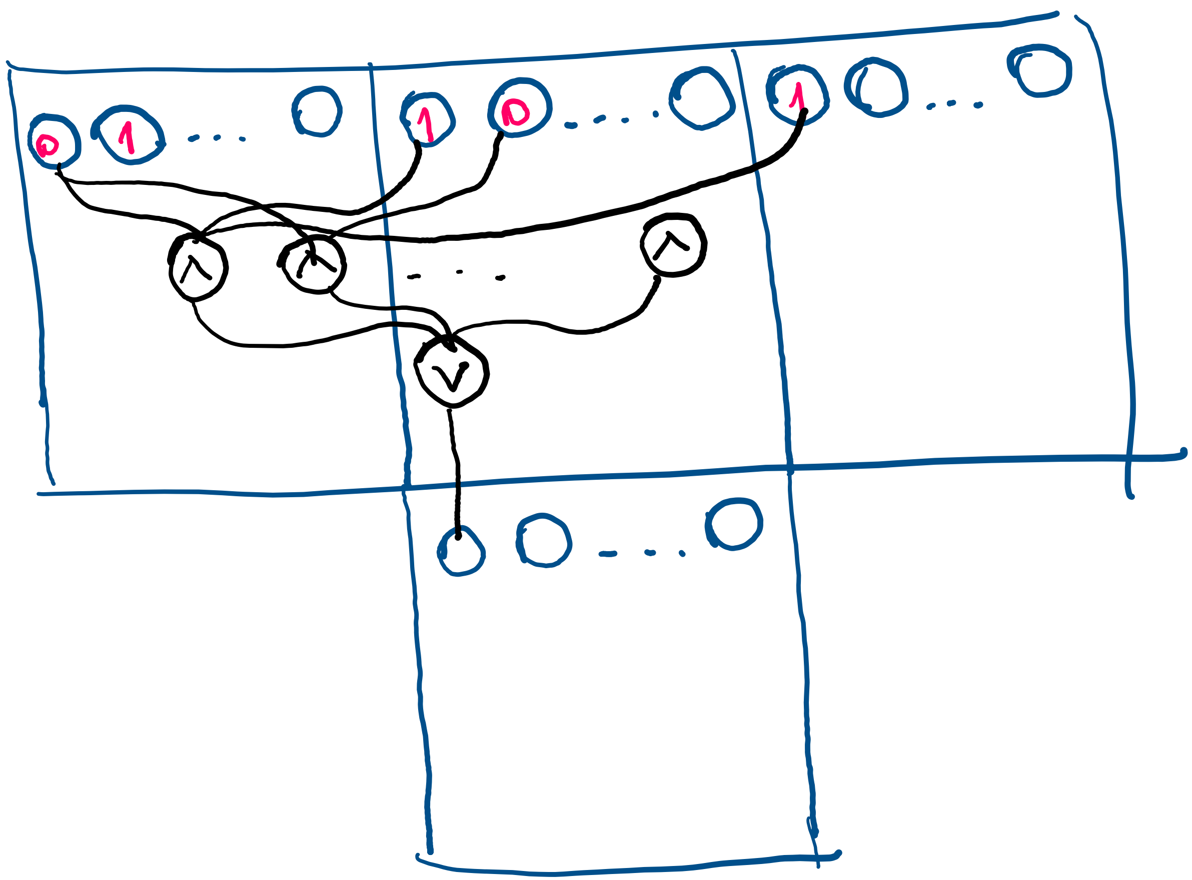
Proof (cont): Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

How many gates do we use for each cell?



Proof (cont): Let A ∈ $DTIME(t(n))$. Then there is a TM M that runs in $O(t(n))$ that decides A; suppose it is C * t(n) for a constant C.

For each cell we use $O(f(k))$ gates to obtain its input from the three cells above. There are $C * t(n)^2$ cells, so overall the circuit has $O(t(n)^2)$ gates.



CircuitSAT = {C | C is a satisfiable Boolean circuit}.

What can we infer about CircuitSAT?

Claim: CircuitSAT $\in$ NP.

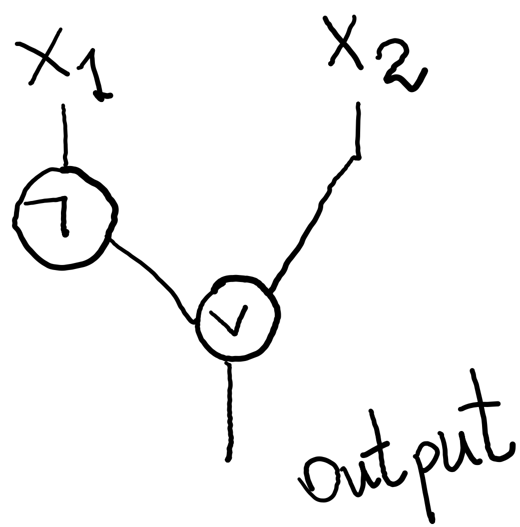
CircuitSAT = {C | C is a satisfiable Boolean circuit}.

What can we infer about CircuitSAT?

Claim: CircuitSAT ∈ NP.

Proof: Design a verifier for CircuitSAT. Given as input (C,x), where C is a circuit and $\mathrm{x} = x_1, \ldots, x_n$ is an input for C, check if C(x) = 1.

E.g: C



$x_1 = 0$
$x_2 = 1$

CircuitSAT = {C | C is a satisfiable Boolean circuit}.

Can we show that CircuitSAT is NP-complete? **Hint:** Use the previous theorem.

Recall: **Definition** (NP-complete): A language L is NP-complete if:

- $L \in NP$
- Every language $A \in NP$ is poly-time reducible to L.