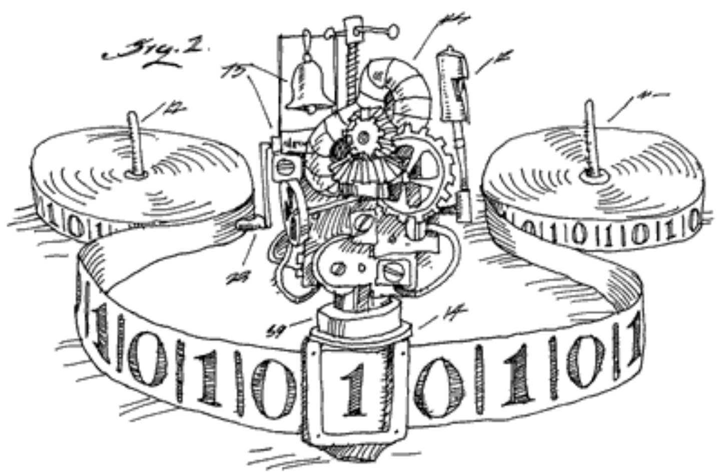
# EECS 376: Foundations of Computer Science

## Lecture 15 - Cook-Levin Theorem and Satisfiability



# Plan in this part of the course

Lecture 1:

- Define **P** and **NP**

Lecture 2: (today)

- Define **polynomial-time mapping reduction**
- Define **NP-hard** and **NP-complete.**
- Show the first NP-complete problem: SAT

Lectures 3 – 4:

- Show many NP-complete problems via reductions

# The Complexity Class **P**

Definition:

**P** is the set of all decision problems that can be decided in polynomial time.

Formally:

- For any problem **L,** an efficient decider **Decide-L** for **L** is s.t.
  - **x** is a "yes" instance ⇔ **Decide-L**(**x**) accepts
  - **x** is a "no" instance ⇔ **Decide-L**(**x**) rejects (follows from above)
  - **Decide-L**(**x**) runs in poly(|**x**|) time

- **P** is the set of all decision problems that have efficient deciders

Example: is gcd(x,y) ≤ b?

# The Complexity Class **NP**

Definition:

**NP** is the set of all decision problems whose yes-instances can be verified in polynomial time.

Formally:

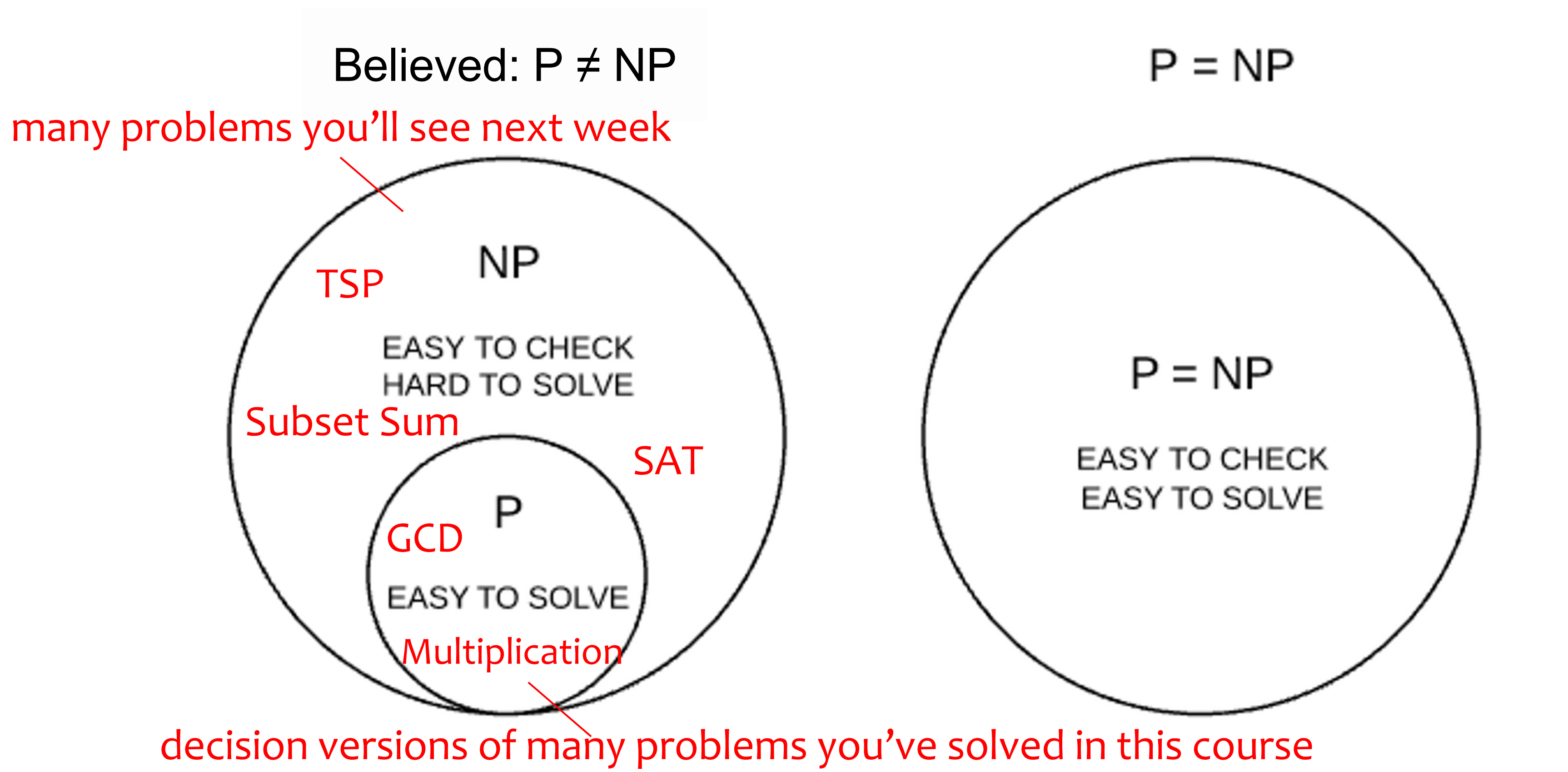
- For any problem **L**, an efficient verifier **Verify-L** for **L** is s.t.
  - **x** is a "yes" instance ⇔ **∃C** **Verify-L**(**x, C**) accepts
  - **x** is a "no" instance ⇔ **∀C** **Verify-L**(**x, C**) rejects (follows from above)
  - **Verify-L**(**x, C**) runs in poly(|**x**|) time

- **NP** is the set of all decision problems that have efficient verifiers
- If **Verify-L**(**x, C**) accepts, then **C** is called a certificate.

Example: Subset Sum, TSP, SAT

**Quiz:** Explain why **P** ⊆ **NP**.

# Two Possible Worlds



**Polynomial-time mapping reduction**
(also called **Karp reduction**)

Note: a different type of reduction from **Turing reduction**

## ***Polynomial-time mapping reduction*** from A to B (denoted $A \le_p B$)

**Defn:** $A \le_p B$ if there is a poly-time-computable function **f** where

**x** is a yes-instance of A $\Leftrightarrow$ **f(x)** is a yes-instance of **B**.

**In words,** given any instance of A, **in polynomial time** we can construct an instance of B whose **yes/no answer is the same.**

Instance **x** of A →  → Instance **f(x)** of B with same answer

fast conversion machine

"Problem B is at least as hard as Problem A"

# *Polynomial-time mapping reduction* from A to B (denoted A $\leq_p$ B)

**Defn:** A $\leq_p$ B if there is a poly-time-computable function **f** where

**x** is a yes-instance of A $\Leftrightarrow$ **f(x)** is a yes-instance of B.

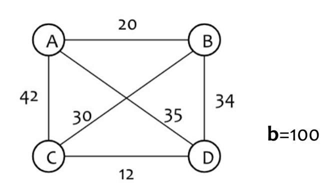
**In words,** given any instance of A, **in polynomial time** we can construct an instance of B whose **yes/no answer is the same.**

Instance **x** of A

fast conversion machine

Instance **f(x)** of B with same answer

$$(x_1 \vee \overline{x}_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_{42})$$



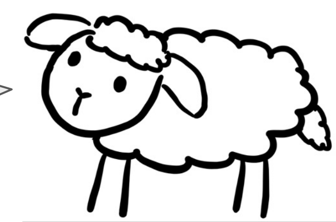

E.g. SAT Instance TSP Instance

# ***Polynomial-time mapping reduction*** from A to B (denoted A $\leq_p$ B)

**Defn:** A $\leq_p$ B if there is a poly-time-computable function **f** where

**x** is a yes-instance of A ⇔ **f(x)** is a yes-instance of B**.**

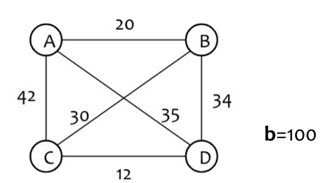
**Difference from Turing's reduction**
1. Polynomial time
2. No "flipping" the answer
3. To solve A, the reduction makes only **one** call to B



Instance **x** of A →  → Instance **f(x)** of B with same answer

fast conversion machine

$$(x_1 \vee \overline{x}_2 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_{42})$$




E.g. SAT Instance TSP Instance

# NP-hardness
and
# NP-completeness

# **NP**-Hardness and **NP**-Completeness

Informal definition: A problem **L** is **NP**-hard if it is at least as hard as EVERY problem in **NP**.

Formal definition: A problem **L** is **NP**-hard if: for EVERY problem **X** in NP, $\mathbf{X} \leq_p \mathbf{L}$.

Showing that some problem is NP-hard seems very strong.

But we will do it soon!

A problem **L** is NP-complete if

- **L ∈ NP**
- **L** is **NP**-hard

# Exercise

**Recall:** **A** $\leq_p$ **B** if

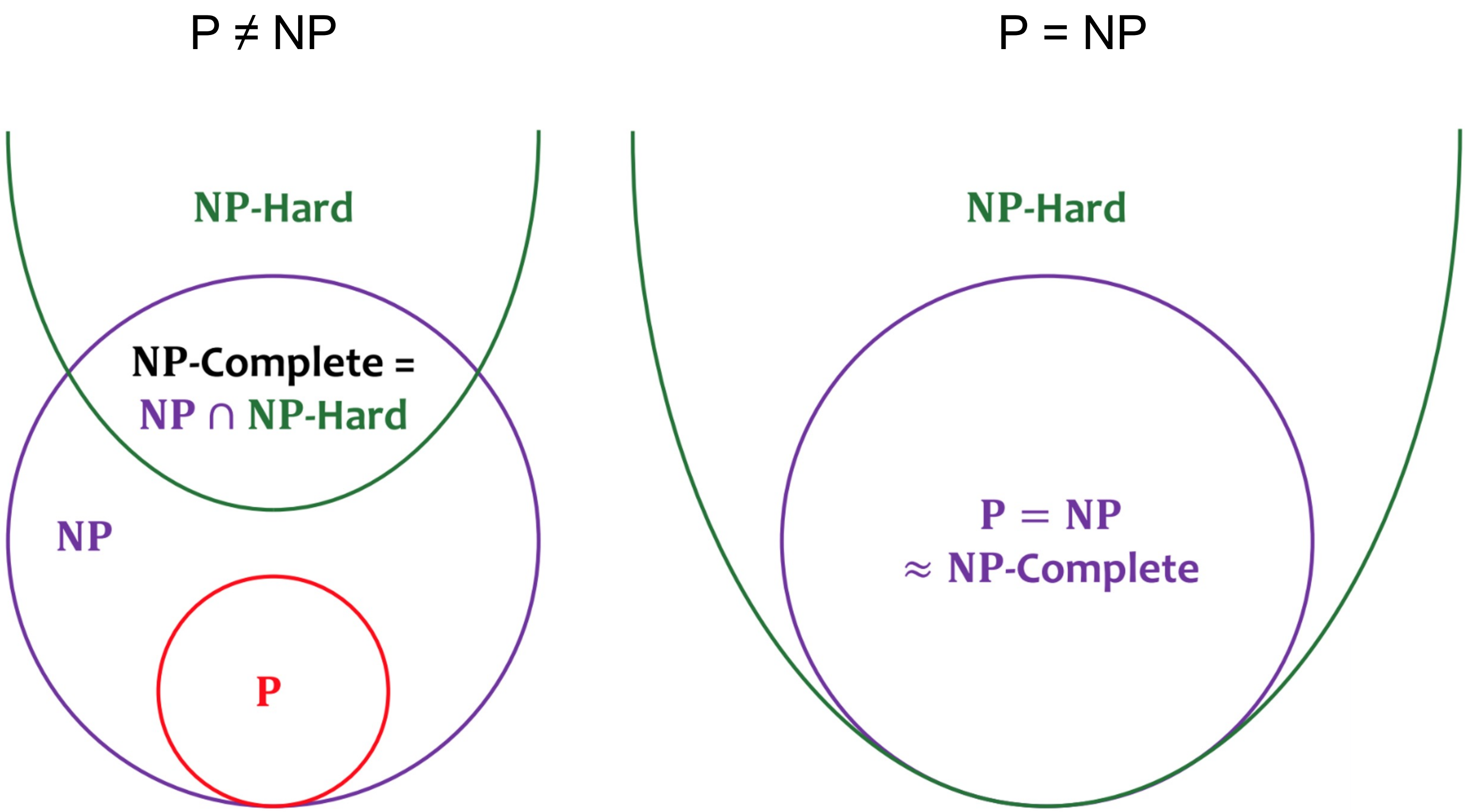
- there is a poly-time-computable function f where
- x in **A** $\Leftrightarrow$ f(x) in **B**.

A problem **L** is NP-hard if for EVERY problem **X** in NP, **X** $\leq_p$ **L.**

**Exercise:** Suppose **A** $\leq_p$**B**.

1. If **B** ∈ **P**, then **A** ∈ **P**.
   - Given an instance **x** for **A,** compute an instance **f(x)** for **B** in **poly time.**
   - As **B** ∈ **P**, we can decide **f(x) in poly time.** Then, return the same answer for **A.**
2. If **A** is NP-hard, then **B** is NP-hard.
   - **Fact:** if **X** $\leq_p$ **A** and **A** $\leq_p$**B**, then **X** $\leq_p$ **B.** (see HW.)
   - For any X ∈ **NP**, **X** $\leq_p$ **A.** (**A** is NP-hard). So, for any X ∈ **NP**, **X** $\leq_p$ **B.** (**B** is NP-hard)
3. Suppose L is NP-complete. Then, L ∈ **P** iff **P** = **NP**
   - Suppose L ∉ **P**. As L ∈ **NP**, then **P** ≠ **NP**.
   - Suppose L ∈ **P**. As L is **NP-hard**, every NP-problem **X** is in **P**. So, **NP** ⊆ **P**.

# Two Possible Worlds





The Cook-Levin Theorem (1971):

**SAT** is **NP-complete**

# Terminology on Formulas

A **Boolean formula** Φ is made up of:

- "literals": variables and their negations (e.g. $x, y, z, \neg x, \neg y, \neg z$)
- OR: $\vee$
- AND: $\wedge$

Example:

$$\Phi 1 = (x \vee y) \wedge (\neg y \vee x \vee \neg z) \wedge (\neg x \vee (y \wedge \neg z))$$

Φ is ***satisfiable*** if

- ∃ a true/false assignment **A** to the variables so that Φ(**A**) = true
- For example, Φ1 is satisfiable.
  - Assign $x = F$, $y = T$, $z = F$

# Satisfiability (SAT)

**Input:** A Boolean formula Φ

**Output:** Is Φ *satisfiable*?

Last time, we showed that SAT is in **NP**.

- Certificate: a true/false assignment **C** to variables where Φ(**C**) = true
- Verifier: **Verify**(Φ, **C**): Check that Φ(**C**) = true

To prove that SAT is **NP**-complete,
remain to prove that SAT is **NP**-hard

A problem **L** is **NP-complete** if
- **L ∈ NP**
- **L** is **NP**-hard

# Proving SAT is NP-hard

**Goal:** for every problem **L** in NP, **L** $\leq_p$ **SAT**.

Fix a problem **L** in NP.
Let $x$ be an instance of **L** of size $|x| = n$.

There is an efficient verifier $V$ s.t.

- $x$ is a "yes" instance $\Leftrightarrow \exists \boldsymbol{C}\ V(x, \boldsymbol{C})$ accepts
- $V(x, \boldsymbol{C})$ runs in $n^k$ time ($k$ is a constant)

**Important:**
$\varphi_{V,x}$ depends on $V$ and $x$.
But not on $C$

**Will show:** in poly$(n)$ time, can construct a formula $\varphi_{V,x}$ s.t.

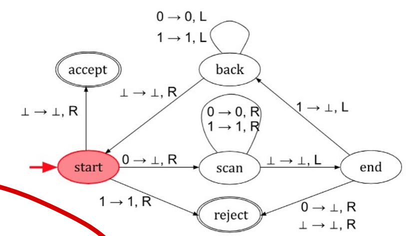
- $\exists \boldsymbol{C}\ V(x, \boldsymbol{C})$ accepts $\Leftrightarrow \varphi_{V,x}$ is satisfiable
- So, **L** $\leq_p$ **SAT**. Done.

**Recall defn: A** $\leq_p$ **B** if
there is a poly-time-computable function **f** where
**x** is a yes-instance of **A** $\Leftrightarrow$ **f(x)** is a yes-instance of **B**.

# Overview of Formula Construction

$x$: instance of size $n$.

$V(x, \boldsymbol{C})$: a TM that runs in $n^k$ steps




**Goal:** construct formula $\varphi_{V,x}$ s.t.

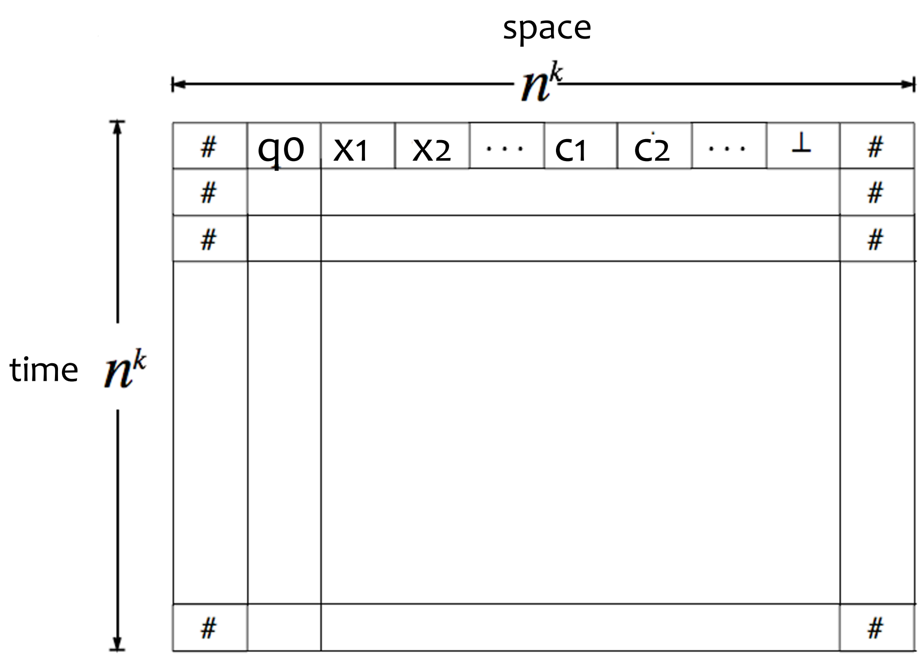
$$\exists \boldsymbol{C}\; V(x, \boldsymbol{C}) \text{ accepts} \Leftrightarrow$$

$$\exists \text{ assignment } \boldsymbol{A}\; \varphi_{V,x}(\boldsymbol{A}) = \text{true}$$

(i.e. $\varphi_{V,x}$ is satisfiable)

Variables of $\varphi_{V,x}$ is based on the **<u>execution tableau</u>** of $V$

# **Execution Tableau:** Visualizing the execution of TM



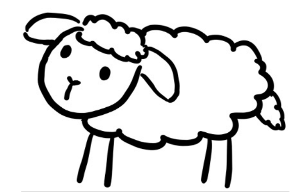

Initial configuration of TM

After 1 step

⋮

Verifier $V$ halts after at most $n^k$ steps

If we know the top row, we can fill in the entire table



If a row of the tableau says "#011$q_5$0001#" it means:

- The tape says: "0110001"
- We are in state $q_5$
- The head is pointing to the symbol right after the state, i.e. 011**0**001

Symbols in tableau consists of $S = \{0,1\} \cup \{\#, \$, \perp\} \cup Q$

$Q$: set of the states of $V$

# Overview of Formula Construction

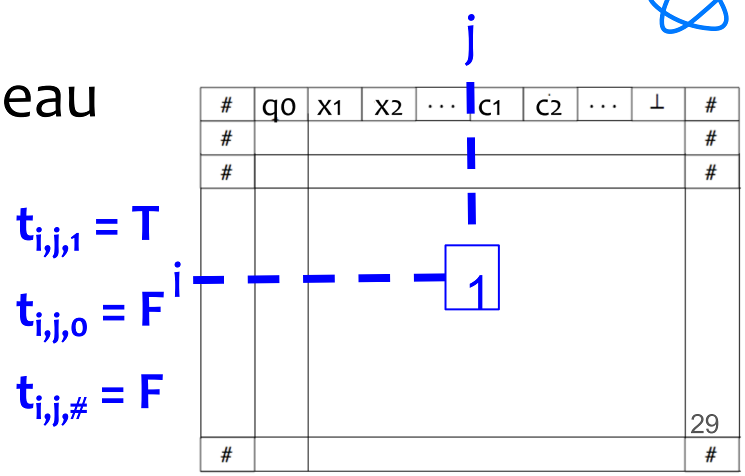
**Goal:** construct formula $\varphi_{V,x}$ s.t.

$$\exists \boldsymbol{C}\ V(x, \boldsymbol{C}) \text{ accepts} \Leftrightarrow$$

$$\exists \text{ assignment } \boldsymbol{A}\ \varphi_{V,x}(\boldsymbol{A}) = \mathrm{T}$$

- **Variables** of $\varphi_{V,x}$ are $\boldsymbol{t_{i,j,s}}$ for all $i, j \le n^k$ and each symbol $s$
- **Intention:** $\boldsymbol{t_{i,j,s}} = \mathbf{T}$ iff *symbol in cell* $(i, j)$ *of the tableau is "s"*

- Assignment of $\varphi_{V,x} \Leftrightarrow$ Values in tableau



# Overview of Formula Construction

Assignment of $\varphi_{V,x}$ $\Leftrightarrow$ Values in tableau

$$\varphi_{V,x} = \varphi_{start} \wedge \varphi_{cell} \wedge \varphi_{accept} \wedge \varphi_{step}$$

1. $\varphi_{start}$ fixes **the value of $x$** in the **first row** of the tableau
2. $\varphi_{accept}$ checks if the tableau contains the **accepting state $q_{accept}$**
3. $\varphi_{cell}$ checks that there is exactly **one symbol per cell**
4. $\varphi_{step}$ checks that the tableau is valid according to **transition rules of $V$**

If we can ensure (1) – (4), we have

$$\exists \boldsymbol{C}\ V(x, \boldsymbol{C}) \text{ accepts} \Leftrightarrow \exists \text{ assignment } \boldsymbol{A}\ \varphi_{V,x}(\boldsymbol{A}) = \mathrm{T}$$

# (1) $\varphi_{start}$ fixes **the value of $x$** in the **first row** of the tableau

$\varphi_{start}$ enforces the starting configuration

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **#** | **$q_0$** | **x** | **$** | **c** | $\perp$ | $\perp$ | **#** |

- Initial state $q_0$,
- Input x, |x| = n; certificate C, $|C| = m \leq n^k$
- $ - a special symbol that separates x and c
- **WE DO NOT KNOW c (‼)**, so we leave a "placeholder"

$$\varphi_{start} = t_{1,1,\#} \wedge t_{1,2,q0} \wedge t_{1,3,x1} \wedge t_{1,4,x2} \wedge \ldots \wedge t_{1,n+2,xn} \wedge t_{1,n+3,\$} \wedge$$

This fixes the first n+3 symbols

$$\underbrace{(t_{1,n+4,1} \vee t_{1,n+4,0} \vee t_{1,n+4,\perp})}_{(c_1 \text{ can be either 1 or 0 or } \perp)} \wedge (t_{1,n+5,1} \vee t_{1,n+5,0} \vee t_{1,n+5,\perp}) \wedge \ldots$$

Placeholders for $\approx n^k$ symbols

**Note**: The size of $\varphi_{start}$ is $O(n^k) = \text{poly(n)}$

# (2) $\varphi_{accept}$ checks if the tableau contains the **accepting state $q_{accept}$**

$$\varphi_{accept} = \bigvee_{1 \le i,j \le n^k} t_{i,j,q_{accept}}$$

**Note**: The size of $\varphi_{accept}$ is $O\left(n^{2k}\right) = \text{poly}(\text{n})$

# (3) $\varphi_{cell}$ checks that there is exactly **one symbol per cell**

every cell

contains at least one symbol

cannot contain two different symbols

$$\varphi_{cell} = \bigwedge_{1 \le i,j \le n^k} \left[ \bigvee_{\sigma \in S} t_{i,j,\sigma} \wedge \bigwedge_{\sigma \neq \tau \in S} \overline{\left(t_{i,j,\sigma} \wedge t_{i,j,\tau}\right)} \right]$$

What do these parts mean, in English?

**Note**: The size of $\varphi_{cell}$ is $O\left(n^{2k}\right) = \mathrm{poly(n)}$

# (4) $\varphi_{step}$ checks that the tableau is valid according to **transition rules of $V$**

**Definition:** A 2x3 "window" is *valid* if it could appear in a valid tableau

Example transition rule: $q_4 \xrightarrow{1 \to 0,\ R} q_5$

| | | |
|---|---|---|
| 0 | $q_4$ | 1 |
| 0 | 0 | $q_5$ |

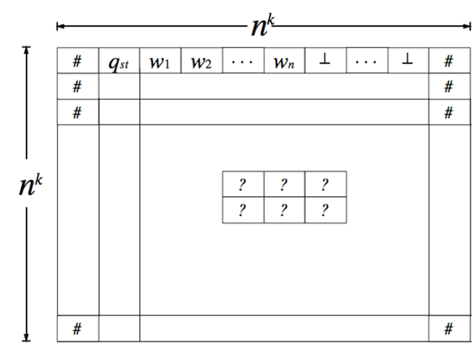
| | | |
|---|---|---|
| 0 | $q_4$ | 1 |
| 0 | 1 | $q_5$ |

**Theorem:**

The whole tableau is valid if and only if **every 2x3 window is valid.**

Proof Idea:

TM can only move 1 left/right step at each time.




**Exercise after class:** see why **2x2 windows** do not work

## (4) $\varphi_{step}$ checks that the tableau is valid according to **transition rules of $V$**

$$\varphi_{step} = \bigwedge_{1 \le i,j \le n^k} \varphi_{window,i,j}$$

$$\varphi_{window,i,j} = (t_{i,j,0} \wedge ti_{,j+1,q4} \wedge t_{i,j+2,1} \wedge t_{i+1,j,0} \wedge t_{i+1,j+1,0} \wedge t_{i+1,j+2,q5)} \vee (\ldots) \vee \ldots)$$

Example of 2x3 valid window

| | | |
|---|---|---|
| 0 | $q_4$ | 1 |
| 0 | 0 | $q_5$ |

More valid windows:

| | | |
|---|---|---|
| 0 | 1 | 1 |
| 0 | 1 | 1 |

nothing changes if head isn't around

| | | |
|---|---|---|
| 0 | 1 | 1 |
| $q_3$ | 1 | 1 |

head could enter from the side

# (4) $\varphi_{step}$ checks that the tableau is valid according to **transition rules of $V$**

$$\varphi_{step} = \bigwedge_{1 \le i,j \le n^k} \varphi_{window,i,j}$$

$$\varphi_{\mathrm{window},i,j} = \bigvee_{\substack{\binom{s_1,s_2,s_3}{s_4,s_5,s_6} \\ \text{valid 2x3 window}}} \left( \begin{array}{c} t_{i,j,s_1} \wedge t_{i,j+1,s_2} \wedge t_{i,j+2,s_3} \wedge \\ t_{i+1,j,s_4} \wedge t_{i+1,j+1,s_5} \wedge t_{i+1,j+2,s_6} \end{array} \right)$$

**Note:** the size of $\varphi_{window,i,j}$ is $O(|S|^6) = O(1)$.
So, the size of $\varphi_{step}$ is $O\left(n^{2k}\right)$

# Conclusion: Formula Construction

Assignment of $\varphi_{V,x}$ ⇔ Values in tableau

$$\varphi_{V,x} = \varphi_{start} \wedge \varphi_{cell} \wedge \varphi_{accept} \wedge \varphi_{step}$$

1. $\varphi_{start}$ fixes **the value of $\boldsymbol{x}$** in the **first row** of the tableau
2. $\varphi_{accept}$ checks if the tableau contains the **accepting state $\mathbf{q_{accept}}$**
3. $\varphi_{cell}$ checks that there is exactly **one symbol per cell**
4. $\varphi_{step}$ checks that the tableau is valid according to **transition rules of $\boldsymbol{V}$**

If we can ensure (1) – (4), we have

$$\exists \boldsymbol{C}\ V(x, \boldsymbol{C}) \text{ accepts} \Leftrightarrow \exists \text{ assignment } \boldsymbol{A}\ \varphi_{V,x}(\boldsymbol{A}) = \mathrm{T}$$

# Proving SAT is NP-hard

**Goal:** for every problem **L** in NP, **L** $\leq_p$ **SAT**.

Fix a problem **L** in NP.
Let $x$ be an instance of **L** of size $|x| = n$.

There is an efficient verifier $V$ s.t.

- $x$ is a "yes" instance $\Leftrightarrow \exists \boldsymbol{C}\ V(x, \boldsymbol{C})$ accepts
- $V(x, \boldsymbol{C})$ runs in $n^k$ time ($k$ is a constant)

**Will show:** in $\text{poly}(n)$ time, can construct a formula $\varphi_{V,x}$ s.t.

- $\exists \boldsymbol{C}\ V(x, \boldsymbol{C})$ accepts $\Leftrightarrow \varphi_{V,x}$ is satisfiable
- So, **L** $\leq_p$ **SAT**. Done.

# Wrap Up

- Define **polynomial-time mapping reduction**
- Define **NP-hard** and **NP-complete.**
- Show the first NP-complete problem: SAT

- SAT $\in$ **P** iff **P** = **NP**
  - Assuming **P** $\neq$ **NP,** no efficient algorithm for **SAT.**

- **Next week:**
  - Assuming **P** $\neq$ **NP,** no efficient algorithm for **many other problems**