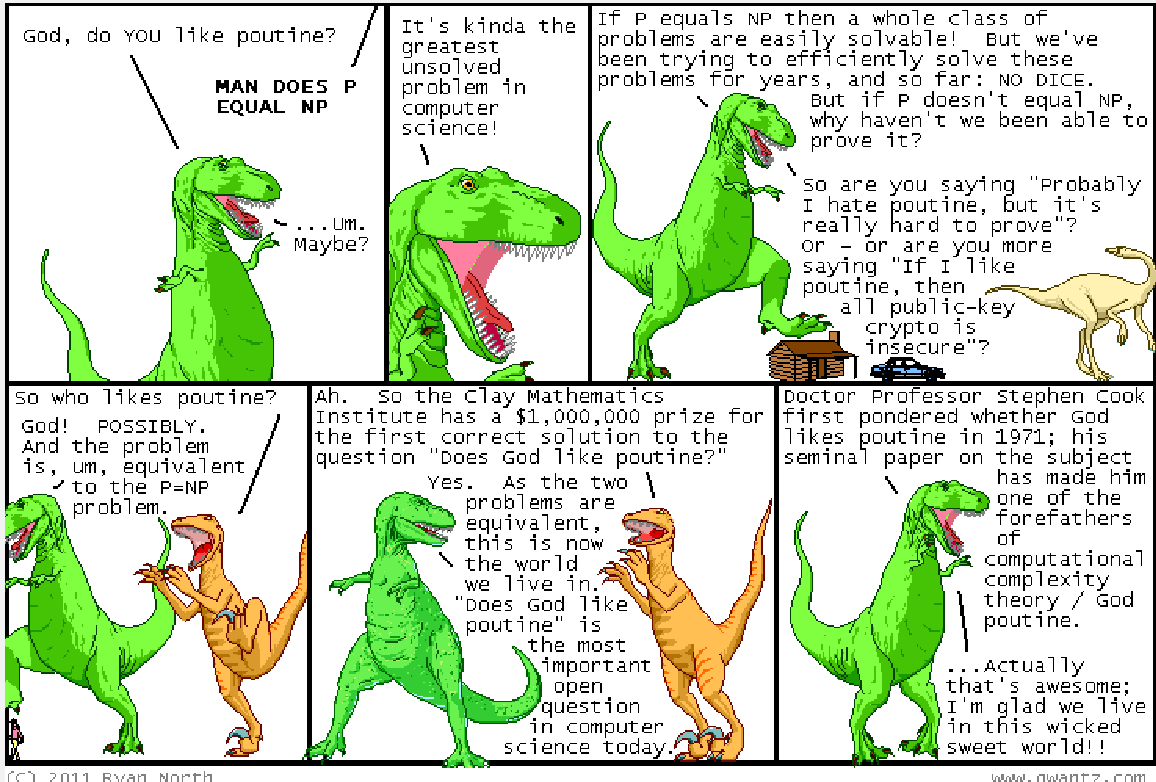
CS 420 / CS 620

The Cook-Levin Theorem

Wednesday, December 10, 2025

UMass Boston Computer Science

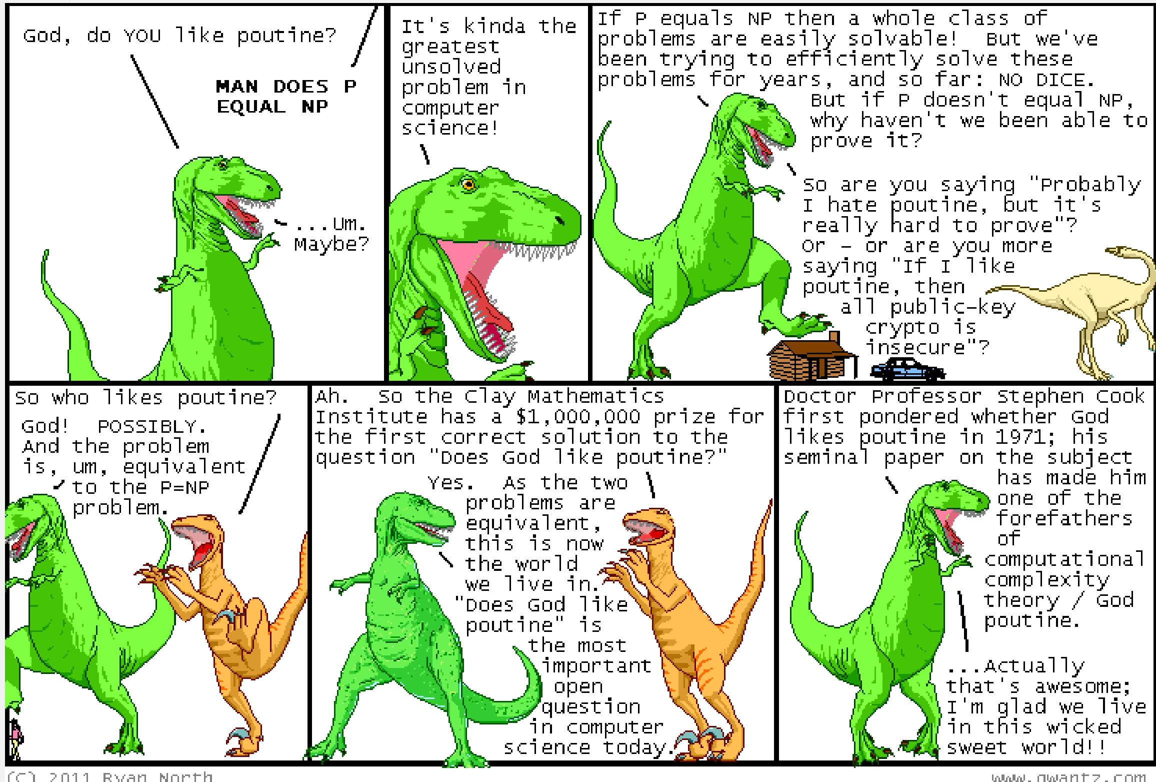
Last lecture!



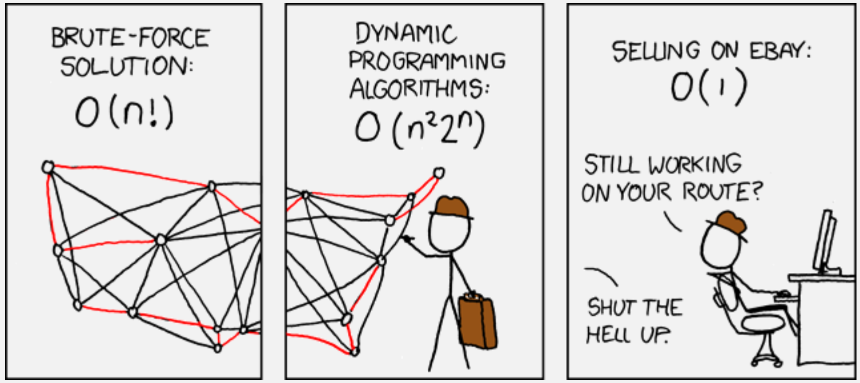
lecture28

Announcements

- HW 13 Last HW
  - ~~Out: **Fri 12/5 12pm (noon)**~~
  - Due: **Fri 12/12 12pm (noon)** (classes end)
  - Late due: **Mon 12/15 12pm (noon)** (exams start)
    - **Nothing accepted after this** (please don't ask)



<u>Last Time</u>: **P** vs **NP**



- **P = class of languages** that can be **decided "quickly"**
  - i.e., **"solvable"** with a <u>**deterministic**</u> **TM**

Want <u>**search**</u> **problems to be in P** … but they mostly are not

- **NP = class of languages** that can be **verified "quickly"**
  - or, **"solvable"** with a <u>**nondeterministic**</u> **TM**

Most <u>**search**</u> **problems are in NP** …

- Does **P = NP ?** (brute force becomes solvable!)
  - **Problem first posed by John Nash**

- It's a **difficult problem** because **how do you prove: "we'll never find a poly time algorithm for X"?**

Progress on whether **P = NP** ?

- Some, but still not close

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

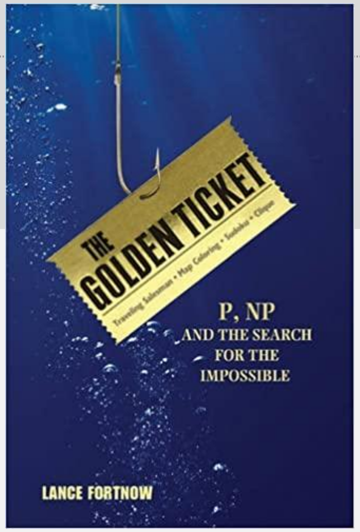
Scott Aaronson*

The Status of the P Versus NP Problem

By Lance Fortnow

Communications of the ACM, September 2009, Vol. 52 No. 9, Pages 78-86

10.1145/1562164.1562186



- One important concept discovered:
  - **NP**-Completeness

NP-Completeness

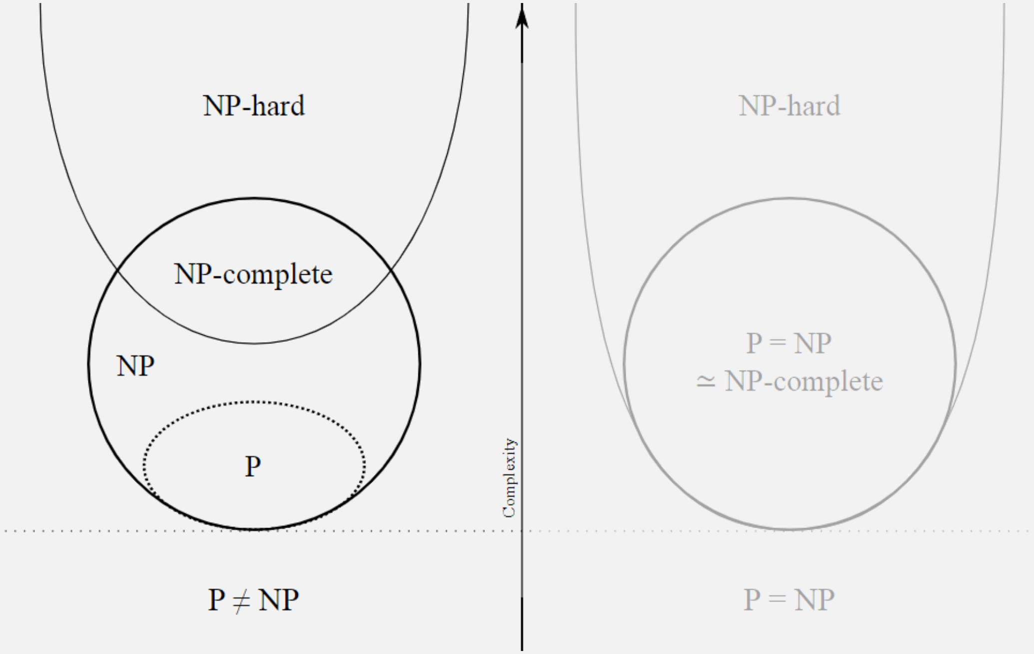
Must prove for all **langs**, not just a single lang

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and easy
2. every A in NP is polynomial time reducible to B. **"NP-hard"**

hard????



NP-Completeness

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

- How does this **help** the **P = NP problem?**

THEOREM

If B is NP-complete and $B \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$.

So to prove **P = NP**, we only need to **find a poly-time algorithm** for one **NP-Complete problem**!

An **NP**-Complete Language?

$$SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

So to prove **P** = **NP**, we only need to **find a poly-time algorithm** for <u>one</u> **NP-Complete problem**!

Boolean Satisfiability

- A **Boolean formula** is **satisfiable** if …
- … there is some **assignment** of TRUE or FALSE (1 or 0) to its **variables** that makes the entire formula TRUE
- Is $(\overline{x} \wedge y) \vee (x \wedge \overline{z})$ satisfiable?
  - Yes
  - x = FALSE,
y = TRUE,
z = FALSE

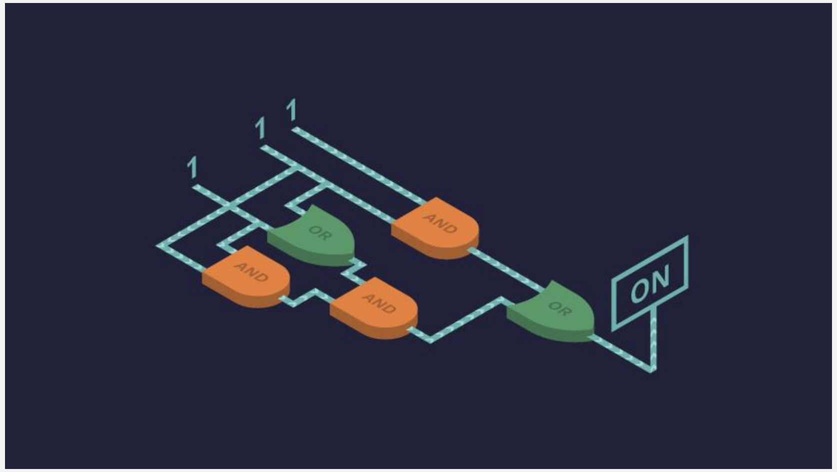
The Boolean Satisfiability Problem

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

Theorem: *SAT* is **NP**-complete

The first **NP**-Complete problem

It sort of makes sense that every problem can be reduced to it …



PROOF: The Cook-Levin Theorem

(Then it'll be much easier to find other **NP**-Complete problems!)

THEOREM

If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

The Cook-Levin Theorem

THEOREM

SAT is NP-complete.

1971

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the

certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautology could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles

1973

КРАТКИЕ СООБЩЕНИЯ

"Universal Search Problems"

УДК 519.14

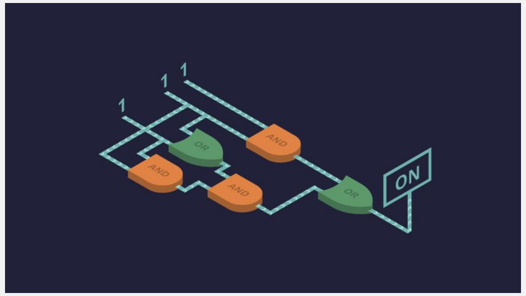
УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

Leonid Levin

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем тождества элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения других задач не снимает для них аналогичного вопроса из-за фантастически большого объема работы, предписываемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизации булевых функций, поиска доказательств ограниченной длины, выяснения изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что



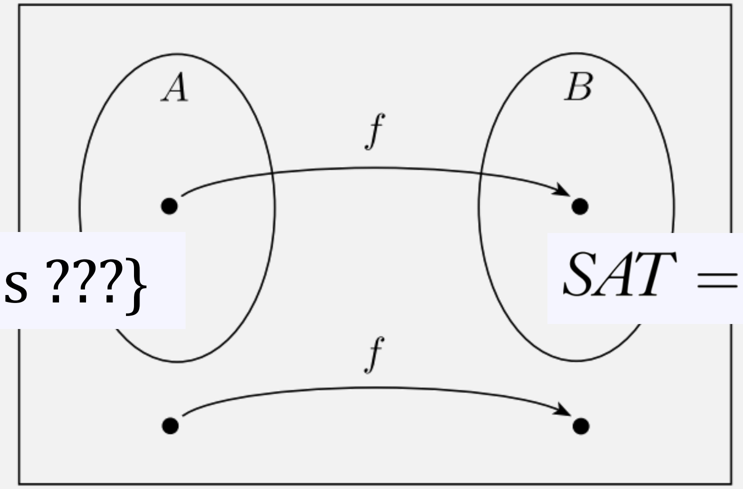
Hard part

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Reducing every **NP** language to **SAT**



Some **NP lang** = $\{w \mid w \text{ is } ???\}$

$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

How can we **convert a string** w to
a **Boolean formula** if we **don't know** w???

Proving theorems about an entire class of langs?

We can still use **general facts about the languages!**

E.g., **"Prove** that **every regular language** is **in $\mathbf{P}$"**

- **Even though** we don't know **what the language is …**
- … we do know **that every regular lang has an DFA accepting it**

E.g., **"Prove** that **every CFL** is **decidable"**

- **Even though** we don't know **what the language is …**
- … we do know **that every CFL has a CFG representation …**
- … and **every CFG has a Chomsky Normal Form**

What do we know about **NP** languages?

They are:

1. **Verified** by a <u>deterministic</u> poly time <u>verifier</u>

2. **Decided** by a <u>nondeterministic</u> poly time <u>decider</u> (NTM)

Let's use this one

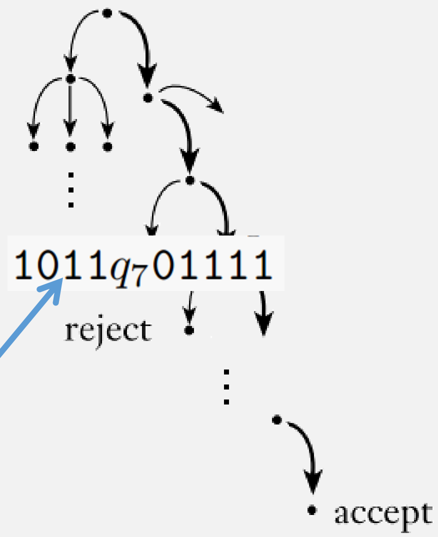
Nondeterministic TMs

- **Formally defined** with **states, transitions, alphabet …**

Nondeterministic

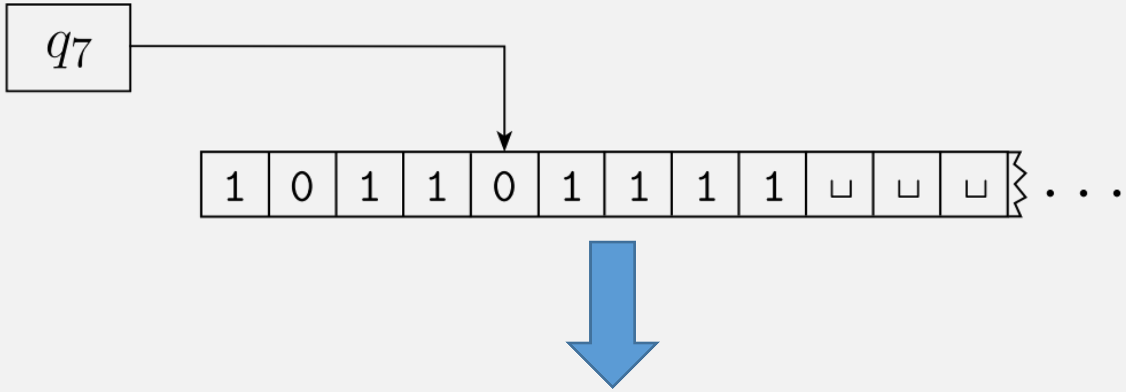
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow \boxed{\mathcal{P}(Q \times \Gamma \times \{L, R\})}$ transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{reject} \neq q_{accept}$.



- **Computation can branch**
- **Each node in the tree represents a TM configuration**

Flashback: TM Config = State + Head + Tape



$$1011q_701111$$

Textual representation of "configuration"

1st char after state is current head position

Nondeterministic TMs

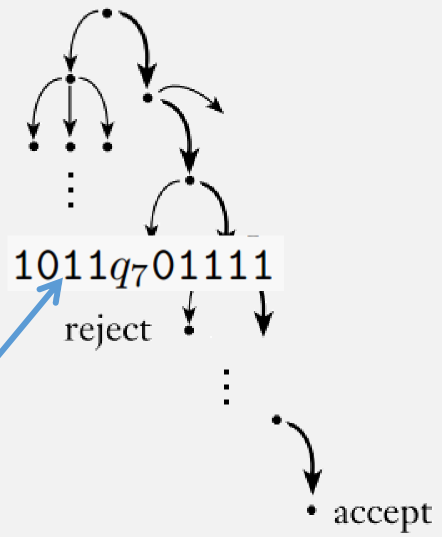
- **Formally defined** with **states, transitions, alphabet …**

Nondeterministic

A *Turing machine* is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\boxed{\delta \colon Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\})}$ transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

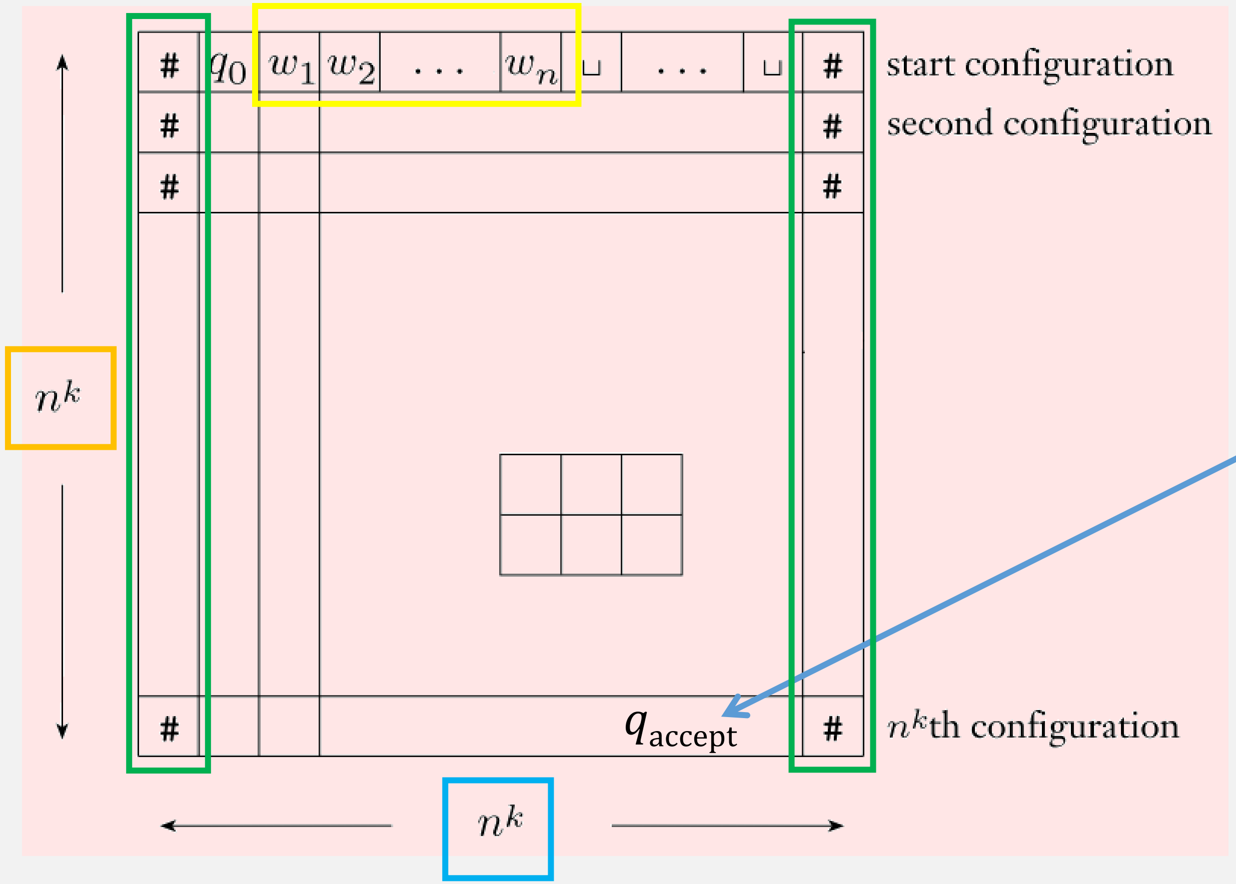
Strings accepted by an NTM must have an **accepting sequence of configurations!**



- Computation can branch
- Each node in the tree represents a TM configuration
- Transitions specify valid configuration <u>sequences</u>

$q_1 0000 \Rightarrow \sqcup q_2 000 \Rightarrow \sqcup \text{x} q_3 00 \Rightarrow \sqcup \text{x} 0 q_4 0 \cdots \boxed{\Rightarrow \sqcup \text{xxx} \sqcup q_{\text{accept}}}$

Accepting config sequence = "Tableau"



- input $w = w_1 \ldots w_n$
- Assume configs start/end with $\boxed{\#}$
- Must have an accepting config
- At most $\boxed{n^k}$ configs
  - (why?) **NP** langs have poly time NTMs
- Each config has length $\boxed{n^k}$
  - (why?) Reading input must be poly time

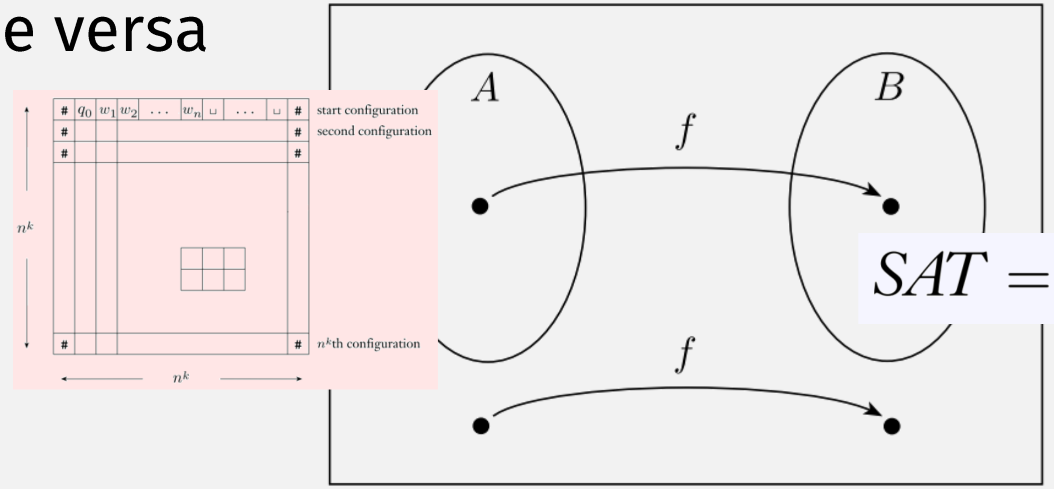
Theorem: SAT is NP-complete

Proof idea:

- **Give an algorithm** reducing **accepting tableaus** to **satisfiable formulas**

- Thus **every string** in the **NP language** (which has an accepting tableau) will be **mapped to a satisfiable formula**
  - and **vice versa**

Resulting formulas will have four components:
$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

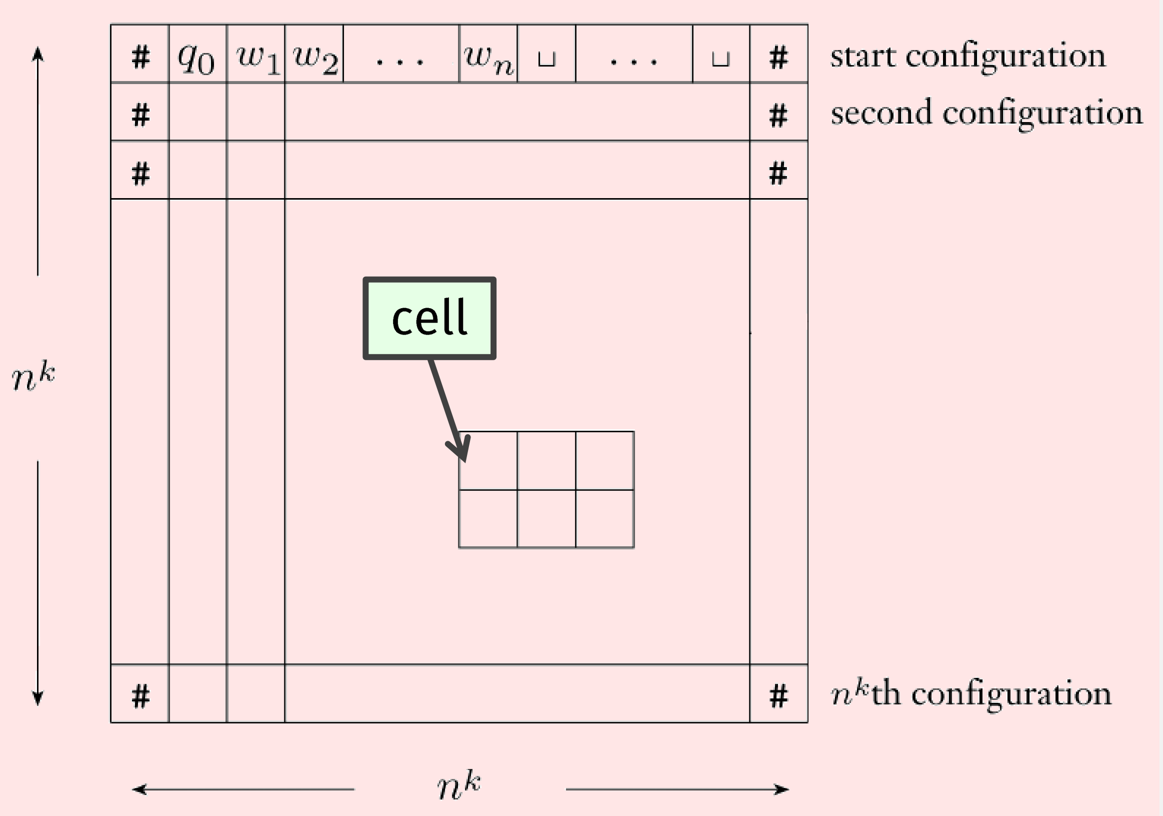


$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

Tableau Terminology

- A tableau <u>cell</u> has coordinate *i,j*
- A cell contains: state, tape char, or #

  $s \in C = Q \cup \Gamma \cup \{\#\}$

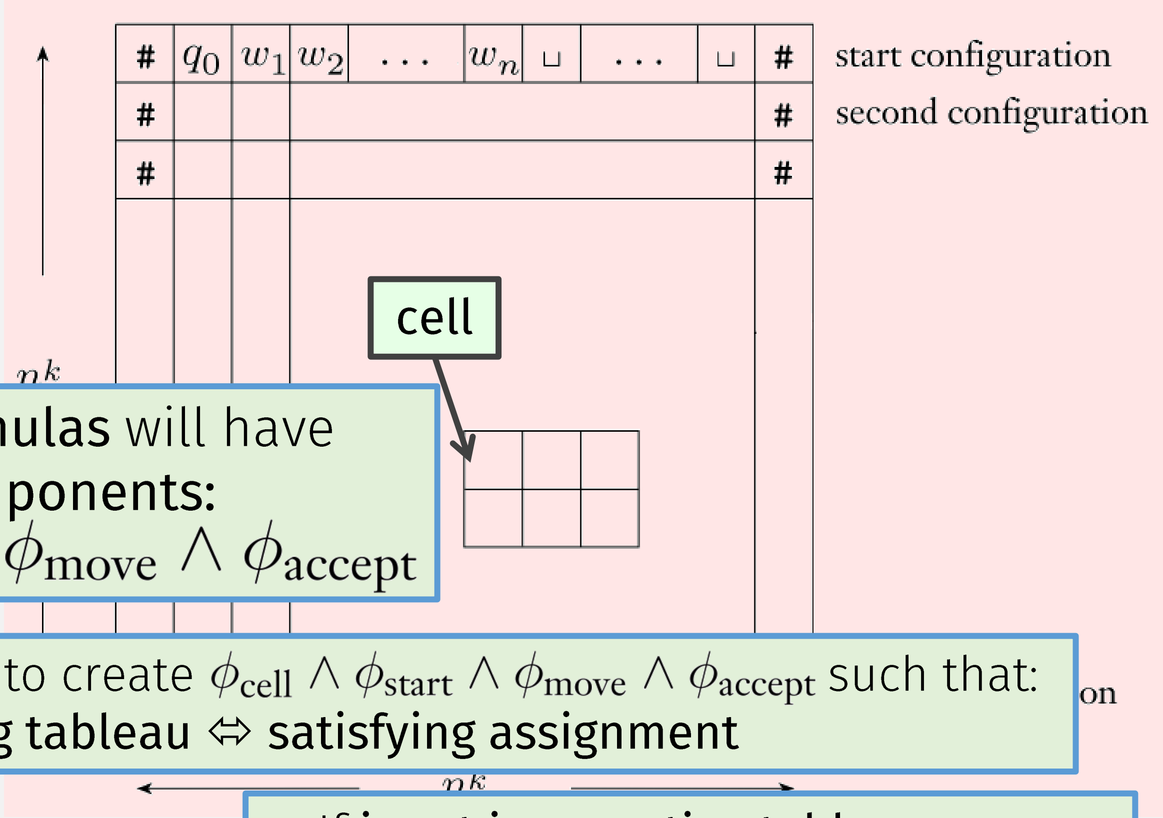


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ⊔,
3. Γ is the tape alphabet, where ⊔ $\in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta\colon Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$ e transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.

Formula Variables

- A tableau cell has coordinate *i,j*
- A cell contains: state, ta
  $s \in C = Q \cup \Gamma \cup \{\#\}$
- For every *i,j,s* **create** variable $x_{i,j,s}$
  - i.e., **one var** for every **possible cell coordinate/content combination**
- Total variables =
  - # cells × # symbols =
  - $n^k \times n^k \times |C| = O(\boldsymbol{n^{2k}})$



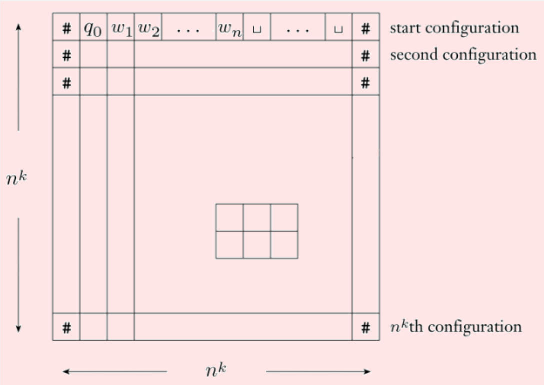
Resulting formulas will have four components:
$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

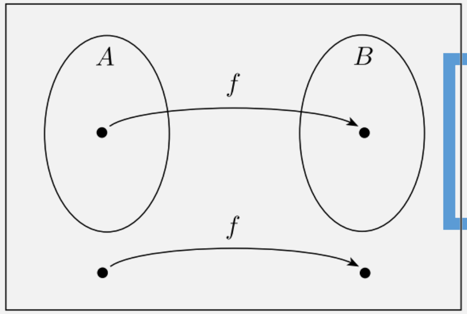
Use these variables to create $\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$ such that:
accepting tableau ⇔ satisfying assignment

⇒ If input is accepting tableau, then output satisfiable ϕ:
- **all four parts** of ϕ must be TRUE

⇐ If input is non-accepting tableau, then output unsatisfiable ϕ:
- **only one part** of ϕ must be FALSE

A *Turing mac*
Q, Σ, Γ are all
1. Q is the s
2. Σ is the i
3. Γ is the t
4. $\delta : Q \times \Gamma$ -
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$.





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE

$$\boxed{\phi_{\text{cell}}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \ne t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

$C = Q \cup \Gamma \cup \{\#\}$

"The following must be TRUE for every cell *i,j*"

"The variable for one s ($\in C$) must be TRUE"

And only one variable for some s must be TRUE

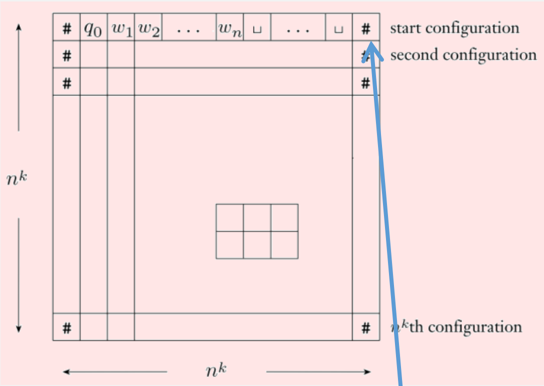
i.e., **every cell has a valid character**

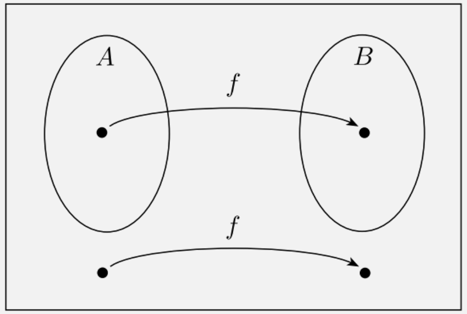
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- Not necessarily (non-accepting sequence of configs can have all valid chars)





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE

$$\phi_{\text{cell}} \wedge \boxed{\phi_{\text{start}}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

For a string w, **start config** is always $\#q_0w_1 \ldots w_n \ldots \#$

The variables in the start config, ANDed together

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

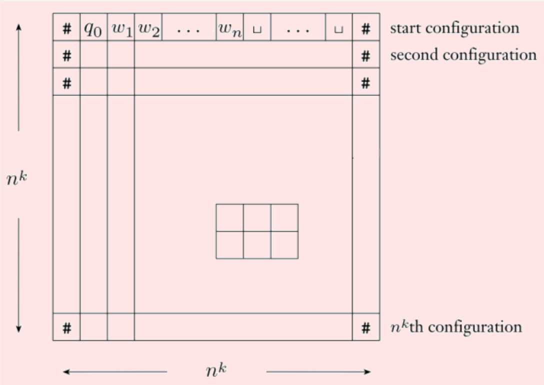
i.e., **tableau has valid start config**

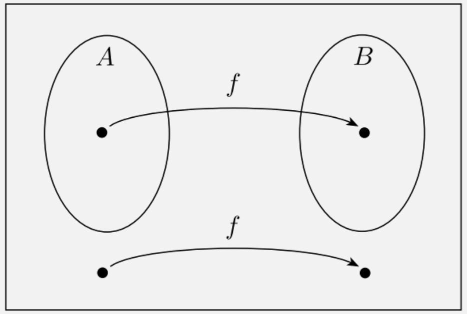
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- **Not necessarily** (non-accepting sequence of configs can have valid start config)





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \boxed{\phi_{\text{accept}}}$$

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}$$

The state q_{accept} must appear in some cell

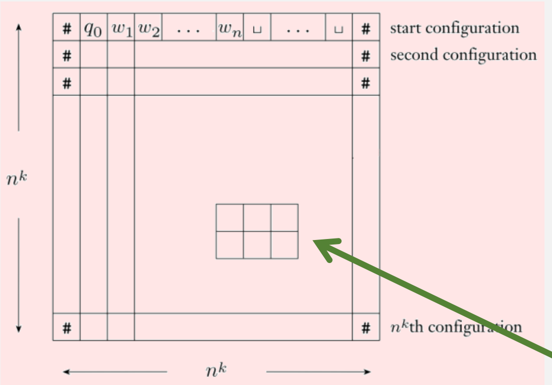
i.e., **tableau has valid accept config**

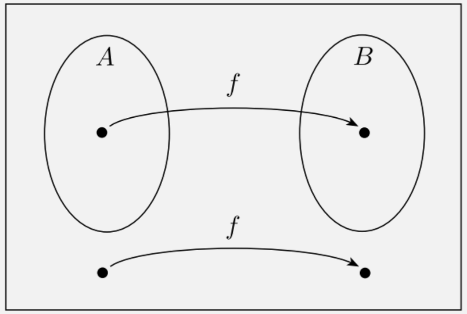
⇒ Does an <u>accepting tableau</u> correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a <u>non-accepting tableau</u> correspond to an unsatisfiable formula?

- **Yes**, because it wont have q_{accept}





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE ☑

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \boxed{\phi_{\text{move}}} \wedge \phi_{\text{accept}}$$

☑ ☑ ☑

- Ensures that **every configuration** is **legal** according to the **previous configuration** and the **TM's δ transitions**
- **Only need to verify every** 2×3 "**window**"
  - **Why?**
  - Because **in one step,** only the **cell at the head can change**
- E.g., **if** $\delta(q_1, \texttt{b}) = \{(q_2,\texttt{c},\text{L}), (q_2,\texttt{a},\text{R})\}$
  - **Which are legal?**

(a)

a	q_1	b
q_2	a	c

(b)

a	q_1	b
a	a	q_2

???
(c)

a	a	q_1
a	a	b

(d)

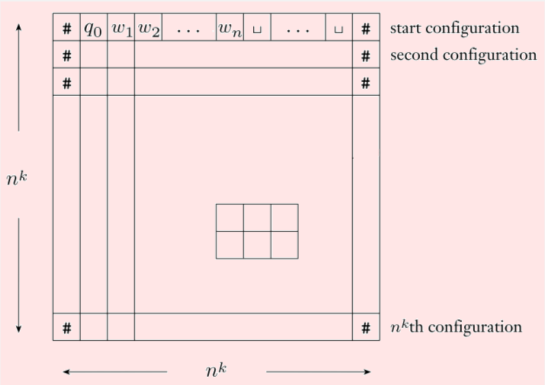
#	b	a
#	b	a

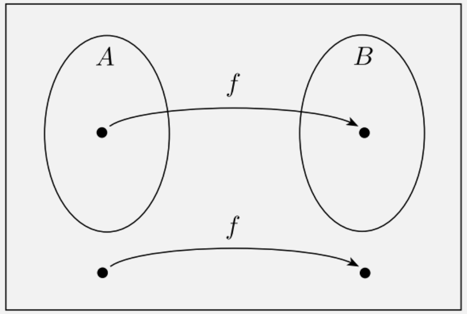
(e)

a	b	a
a	b	q_2

(f)

b	b	b
c	b	b





⇒ accepting tableau: **all four** must be TRUE
⇐ nonaccepting tableau: **one** must be FALSE ☑

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \boxed{\phi_{\text{move}}} \wedge \phi_{\text{accept}}$$

☑ ☑ ☑

i.e., **all transitions are legal, according to δ fn**

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \left(\text{the } (i,j)\text{-window is legal}\right)$$

i,j = upper center cell

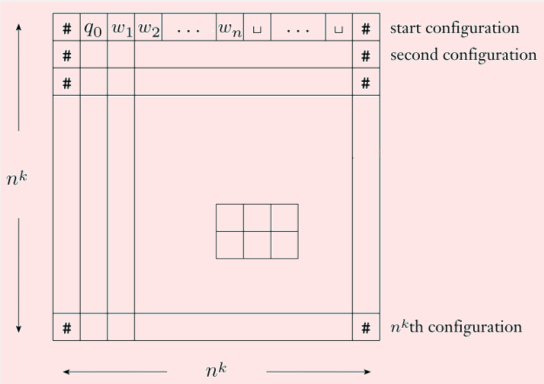
$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

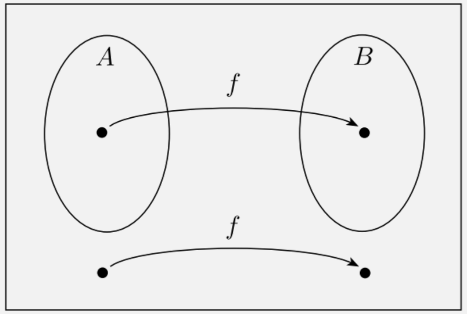
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's **in the tableau**,
- and **assign other vars** = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- **Not necessarily** (non-accepting sequence of configs can have all valid transitions)





⇒ accepting tableau: **all four** must be TRUE ✅
⇐ nonaccepting tableau: **one** must be FALSE ✅

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} (\text{the } (i,j)\text{-window is legal})$$

i,j = upper center cell

$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6} \right)$$

⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **Yes**, assign $x_{i,j,s}$ = TRUE if it's in the tableau,
- and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?

- Not necessarily (non-accepting sequence of configs can have all valid transitions)

To Show Poly Time Mapping Reducibility …

Language A is ***polynomial time mapping reducible***, or simply ***polynomial time reducible***, to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

To show poly time mapping reducibility:

☑ 1. create **computable fn**,

2. show that it **runs in poly time**,

☑ 3. then show **forward direction** of mapping red.,

☑ 4. and **reverse direction**
   (or **contrapositive** of **reverse direction**)

Time complexity of the reduction

- Number of cells = $O(\boldsymbol{n^{2k}})$

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \ne t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right] \qquad \boxed{O(\boldsymbol{n^{2k}})}$$

"The following must be TRUE for every cell i,j"

"The variable for one s must be TRUE"

And only one variable for some s must be TRUE

Time complexity of the reduction

- Number of cells = $O(\boldsymbol{n}^{2k})$

$$\phi_{\text{cell}} = \bigwedge_{1\le i,j\le n^k} \left[\left(\bigvee_{s\in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t\in C \\ s\neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]$$

$\boxed{O(\boldsymbol{n}^{2k})}$

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

$\boxed{O(\boldsymbol{n}^{k})}$

The **variables** in the **start config**, **ANDed together**

Time complexity of the reduction

- Number of cells = $O(\boldsymbol{n^{2k}})$

$$\phi_{\text{cell}} = \bigwedge_{1\le i,j\le n^k} \left[\left(\bigvee_{s\in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t\in C \\ s\neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right] \quad \boxed{O(\boldsymbol{n^{2k}})}$$

$$\begin{aligned}\phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \quad \boxed{O(\boldsymbol{n^{k}})} \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}\end{aligned}$$

$$\phi_{\text{accept}} = \bigvee_{1\le i,j\le n^k} x_{i,j,q_{\text{accept}}}$$

The state q_{accept} must appear in some cell

$\boxed{O(\boldsymbol{n^{2k}})}$

Time complexity of the reduction

- Number of cells = $O(\boldsymbol{n}^{2k})$

$$\phi_{\text{cell}} = \bigwedge_{1\le i,j\le n^k} \left[\left(\bigvee_{s\in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t\in C \\ s\neq t}} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right] \quad \boxed{O(\boldsymbol{n}^{2k})}$$

$$\begin{aligned} \phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \quad \boxed{O(\boldsymbol{n}^{k})} \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \end{aligned}$$

$$\phi_{\text{accept}} = \bigvee_{1\le i,j\le n^k} x_{i,j,q_{\text{accept}}} \quad \boxed{O(\boldsymbol{n}^{2k})}$$

$$\phi_{\text{move}} = \bigwedge_{1\le i<n^k,\ 1<j<n^k} \big(\text{the } (i,j)\text{-window is legal}\big) \quad \boxed{O(\boldsymbol{n}^{2k})}$$

Time complexity of the reduction

Total: $O(\boldsymbol{n^{2k}})$

- Number of cells = $O(\boldsymbol{n^{2k}})$

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \ne t}} \left(\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right) \right]$$

$O(\boldsymbol{n^{2k}})$

$$\begin{aligned} \phi_{\text{start}} = {} & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \end{aligned}$$

$O(\boldsymbol{n^{k}})$

$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}$$

$O(\boldsymbol{n^{2k}})$

$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} \left(\text{the } (i,j)\text{-window is legal} \right)$$

$O(\boldsymbol{n^{2k}})$

To Show Poly Time Mapping Reducibility …

Language A is ***polynomial time mapping reducible***, or simply ***polynomial time reducible***, to language B, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f \colon \Sigma^* \longrightarrow \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the ***polynomial time reduction*** of A to B.

To show poly time <u>mapping reducibility</u>:

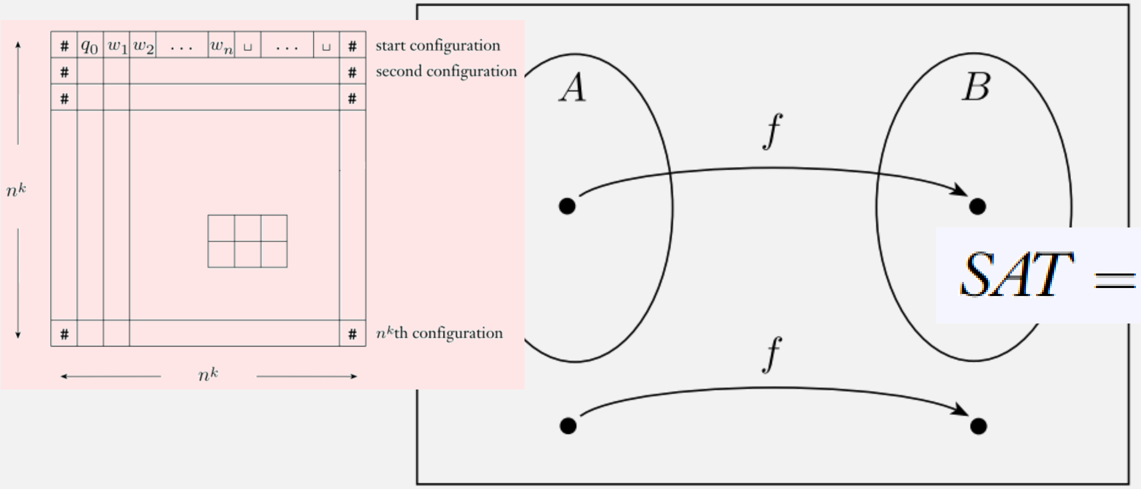
☑ 1. create **computable fn**,
☑ 2. show that it **runs in poly time**,
☑ 3. then show **forward direction** of mapping red.,
4. and **reverse direction**
☑ (or **contrapositive** of **forward direction**)

QED: SAT is NP-complete

DEFINITION

A language B is ***NP-complete*** if it satisfies two conditions:

☑ **1.** B is in NP, and
☑ **2.** every A in NP is polynomial time reducible to B.



$$\mathit{SAT} = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

Now it will be **much easier to prove** that **other languages are NP-complete!**

THEOREM

Using: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

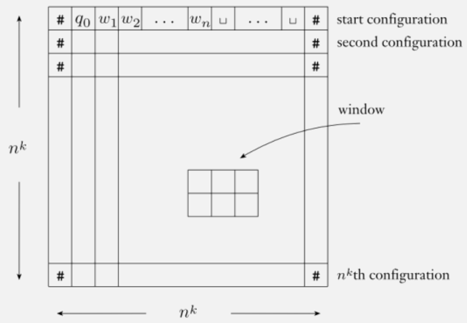
3 steps to prove a language C is **NP**-complete:

1. Show C is in **NP**
2. Choose B, the **NP**-complete problem to reduce from
3. Show a poly time mapping reduction from B to C

To show poly time mapping reducibility:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
   (or **contrapositive** of **reverse direction**)

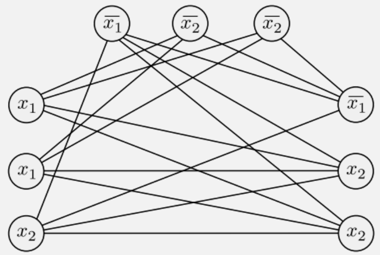
If you are not **Stephen Cook** or **Leonid Levin**, **use this theorem** to **prove** a language is **NP**-complete

NP-Complete problems

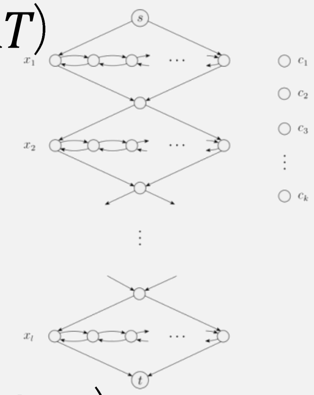


- $SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$ (Cook-Levin Theorem)

- $3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$ (reduce from *SAT*)



- $CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$ (reduce from 3*SAT*)



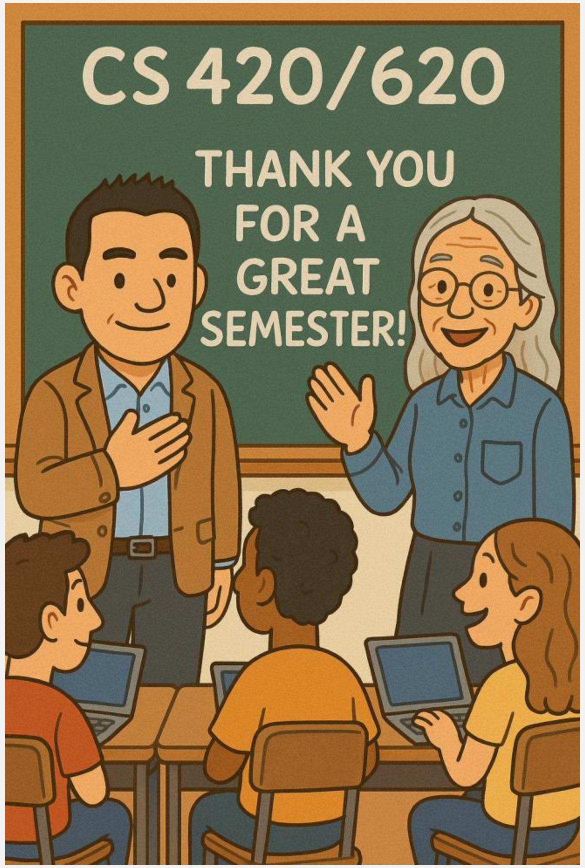
- $HAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t\}$ (reduce from 3*SAT*)

- $UHAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a undirected graph with a Hamiltonian path from } s \text{ to } t\}$ (reduce from *HAMPATH*)

More **NP**-Complete problems

- $$\begin{aligned} \mathit{SUBSET\text{-}SUM} = \{\langle S, t\rangle \mid\ & S = \{x_1, \ldots, x_k\}, \text{ and for some} \\ & \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\} \end{aligned}$$
  - (reduce from *3SAT*)

- $$\begin{aligned} \mathit{VERTEX\text{-}COVER} = \{\langle G, k\rangle \mid\ & G \text{ is an undirected graph that} \\ & \text{has a } k\text{-node vertex cover}\} \end{aligned}$$
  - (reduce from *3SAT*)



Thank you for a great semester!

create a picture for the last lecture of CS 420 / 620