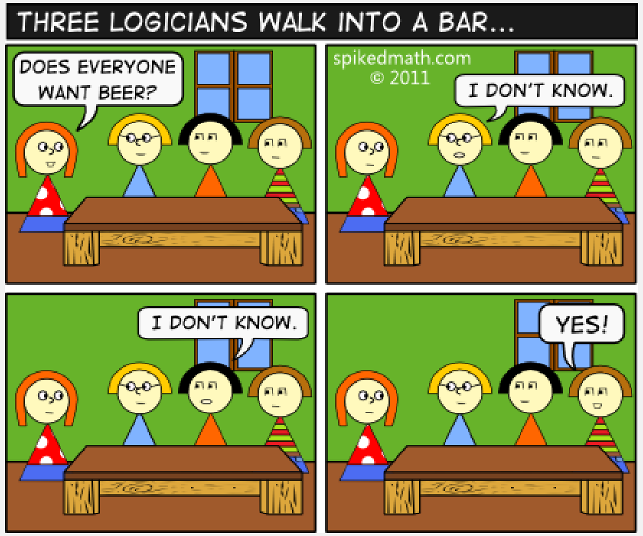
The Cook-Levin Theorem

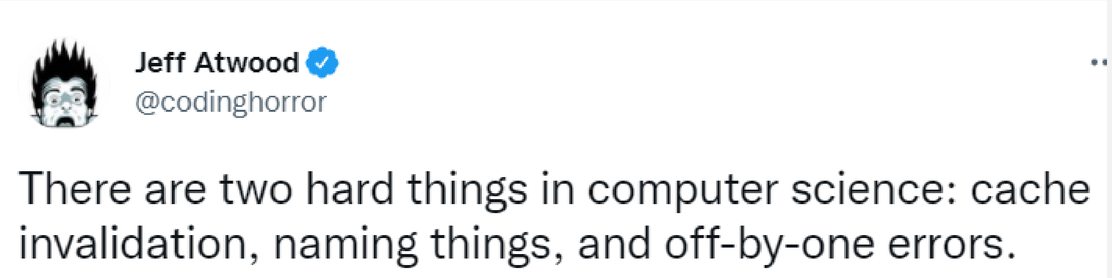
(the 1st NP-Complete Problem)

Monday, May 2, 2022



Announcements

- HW 11 out
  - Due Tues 5/3 11:59pm EST
- 4 lectures left!
- Course evals coming



Last Time: **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.

easy

hard????

Must prove for <u>all</u> langs, not just a single language

It's very hard to prove the first **NP**-Complete problem!

(Just like figuring out the first undecidable problem was hard!)

But after we find one, then we use that problem to prove other problems **NP**-Complete!

THEOREM

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

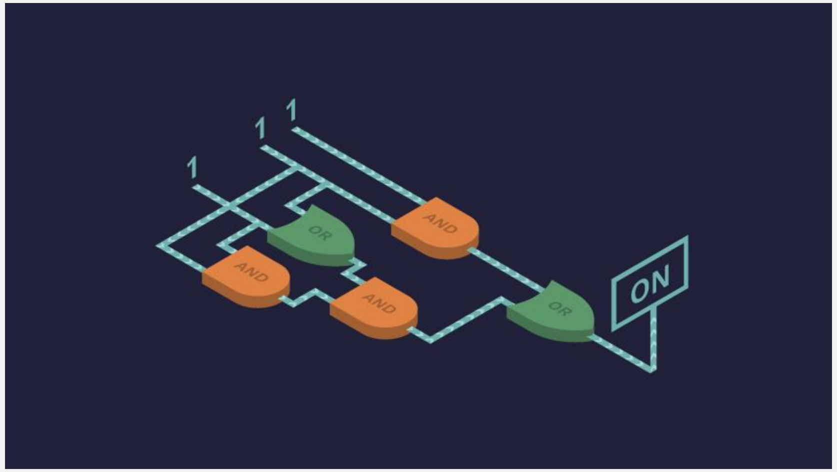
Today: The Cook-Levin Theorem

The first **NP**-Complete problem

THEOREM

SAT is NP-complete.

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



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 tautologyhood 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

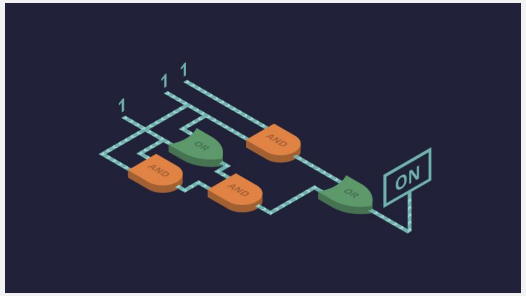
УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

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



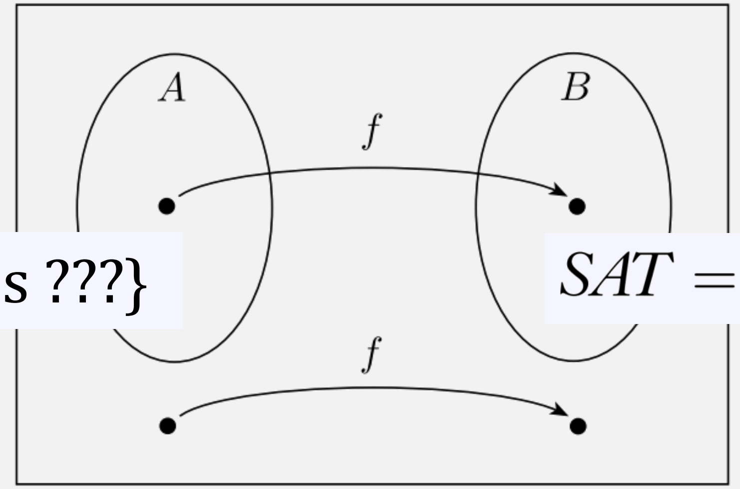
Hard part

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 | w is ???}

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

How can we reduce some 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 **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 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 deterministic poly time verifier
2. Decided by a nondeterministic poly time decider (NTM)

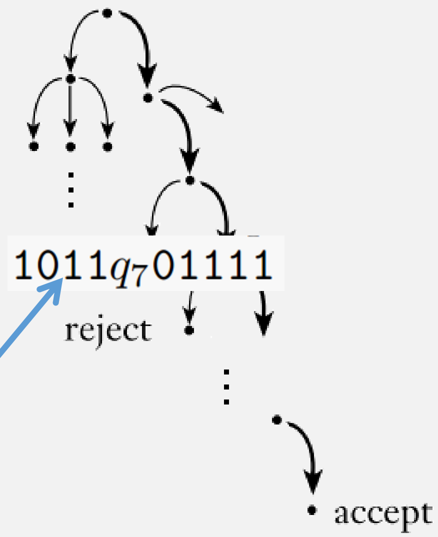
Let's use this one

Flashback: Non-deterministic TMs

- Formally defined with states, transitions, alphabet …

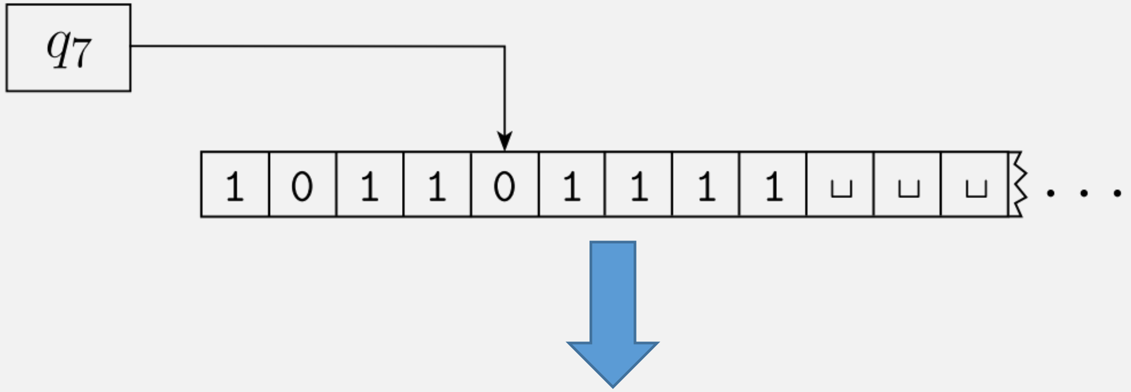
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

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. $\delta \colon Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ 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

Flashback: Non-deterministic TMs

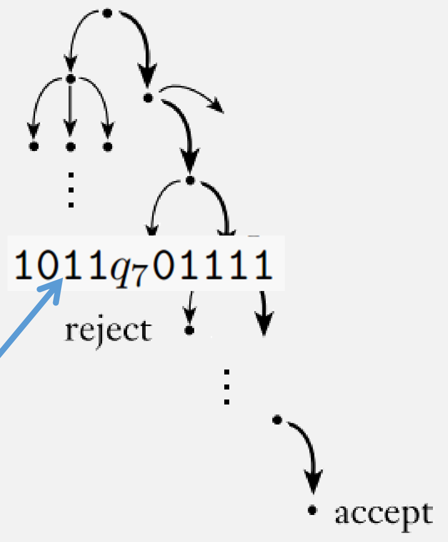
- Formally defined with states, transitions, alphabet …

<u>Idea</u>: We don't know the specific language or strings in the language, but …

… we know those strings must have an **accepting sequence of configurations!**

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

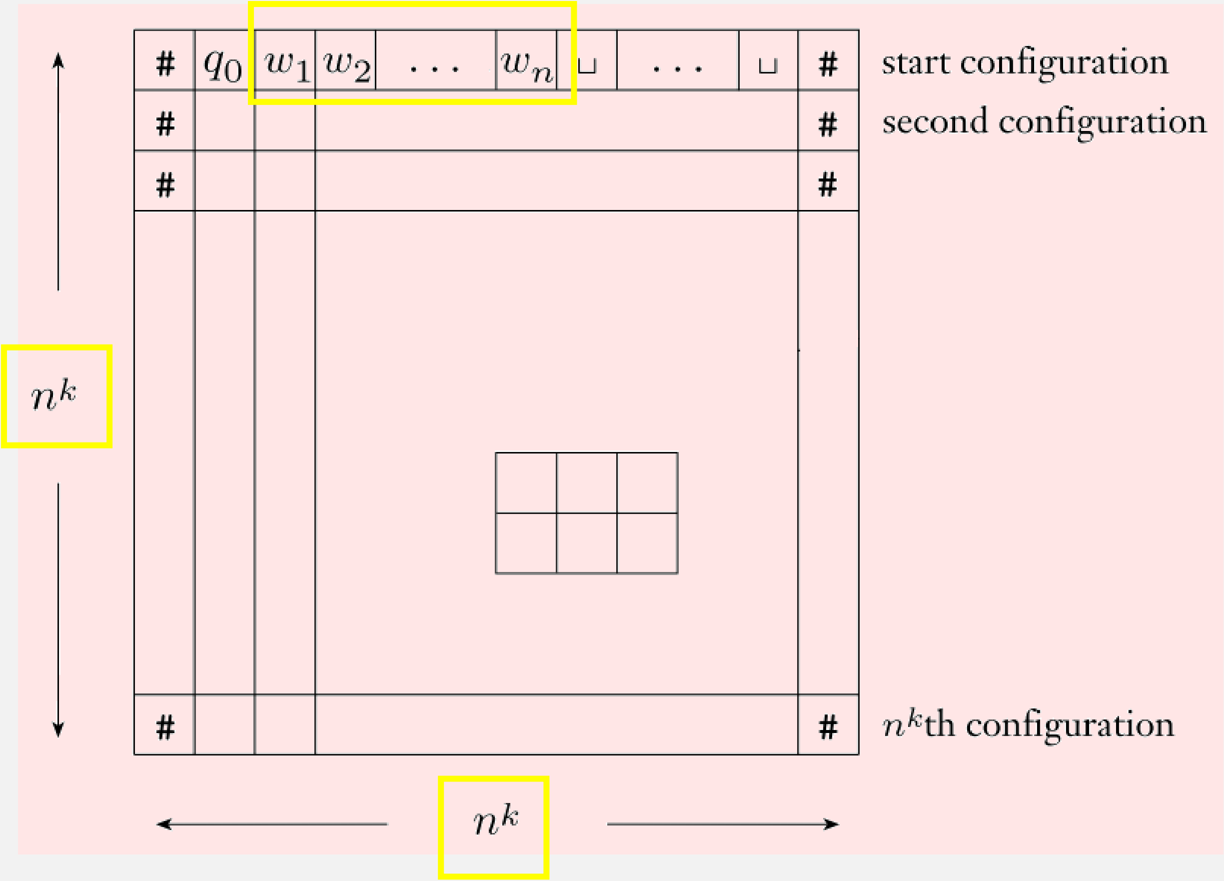
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ transition function,
5. $q_0 \in Q$ 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
- Transitions specify valid configuration <u>sequences</u>

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

Accepting config sequence = "Tableau"



- input $w = w_1 \ldots w_n$
- Assume configs start/end with #
- Must have an accepting config
- At most n^k configs
  - (why?)
- Each config has length n^k
  - (why?)

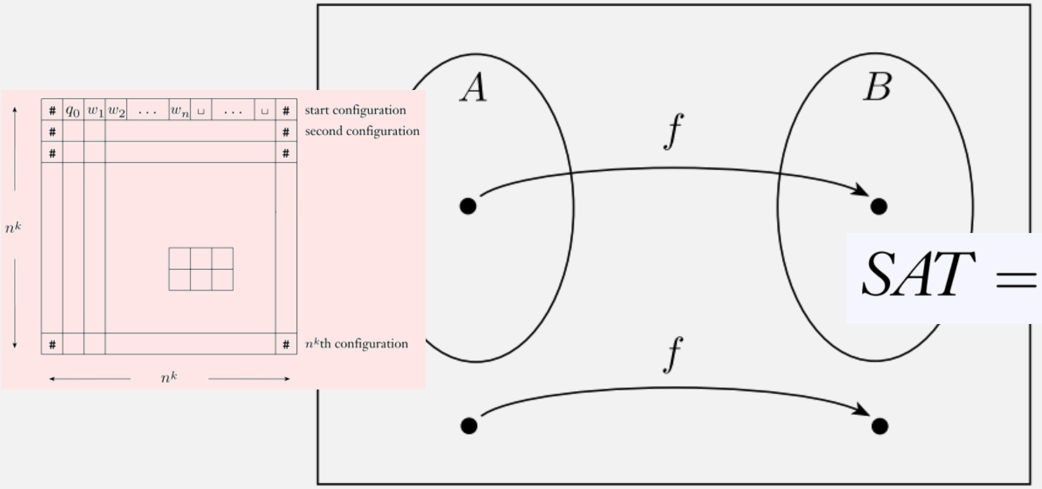
Theorem: SAT is NP-complete

Proof idea:

- Give an algorithm that reduces accepting tableaus to satisfiable formulas

- Thus **every string in the NP lang will be mapped to a sat. formula**
  - and **vice versa**

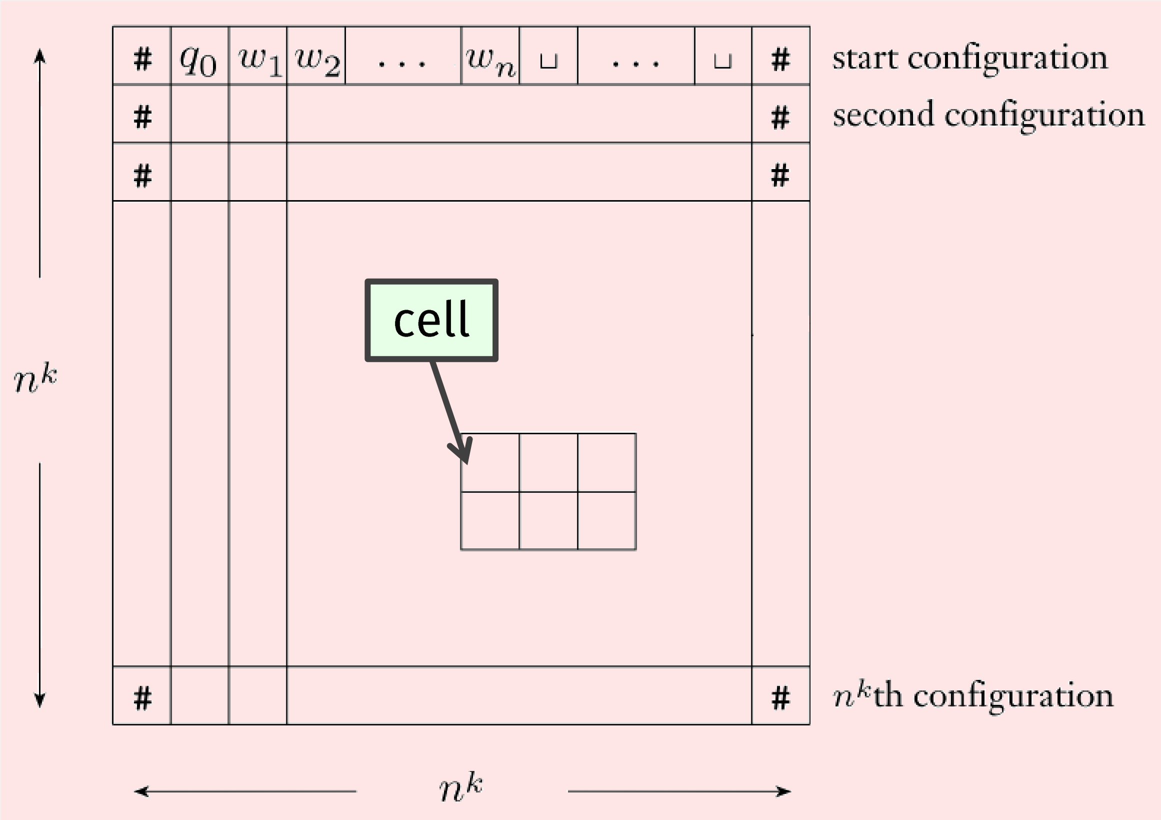
Resulting formulas will have four components:
$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$



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

Tableau Terminology

- A tableau cell has coordinate *i,j*
- A cell has symbol:
  $s \in C = Q \cup \Gamma \cup \{\#\}$

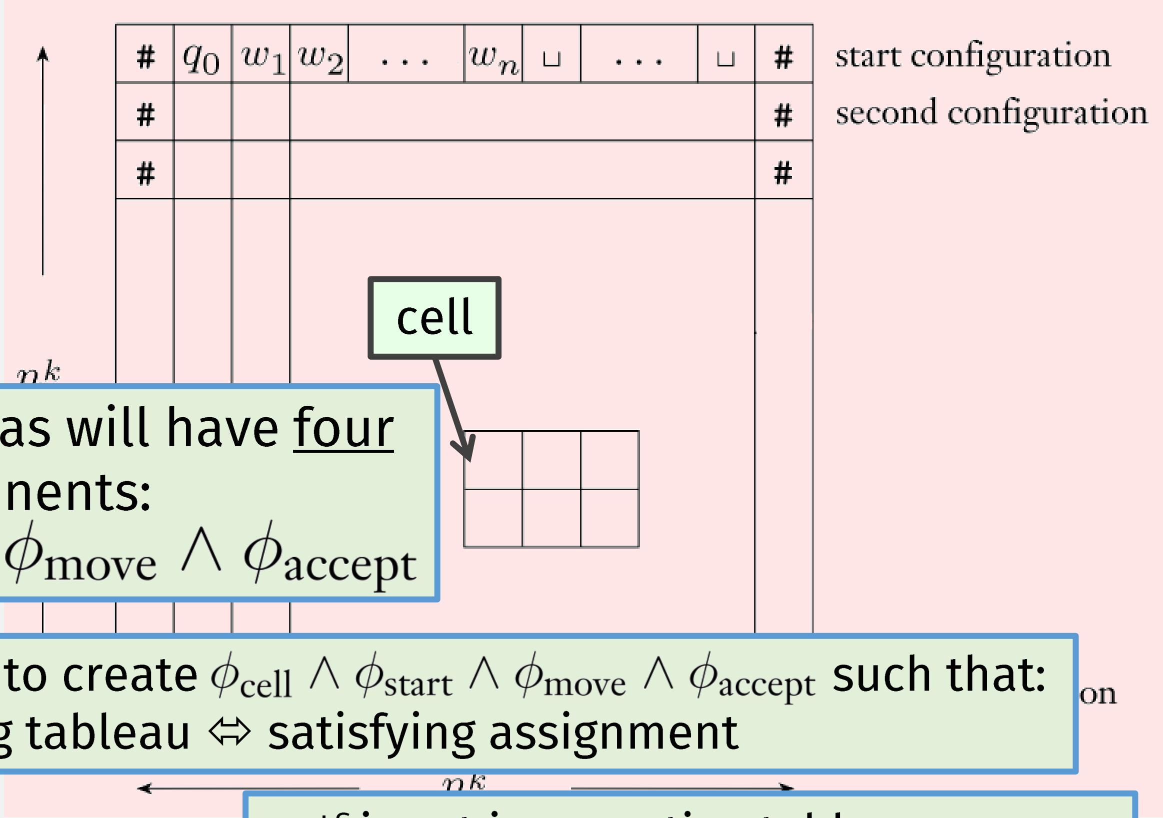


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and

1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** ⊔,
3. Γ is the tape alphabet, where $⊔ \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ e 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 $q_{reject} \neq q_{accept}$.

Formula Variables

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



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

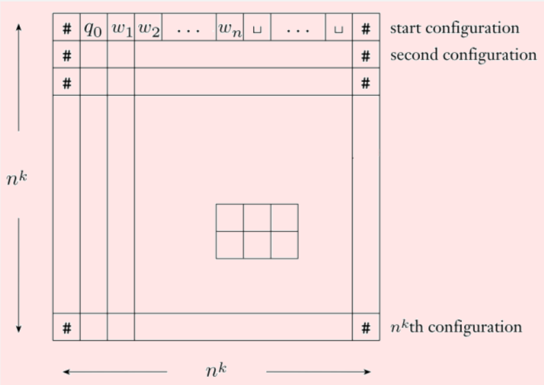
Use these variables to create $\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$ such that:
accepting tableau ⇔ satisfying assignment

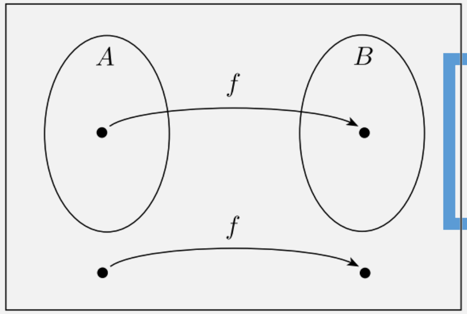
⇒ If input is <u>accepting tableau,</u> then output satisfiable ϕ:
- **all four parts** of ϕ must be TRUE

⇐ If input is <u>non-accepting tableau,</u> 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. $\delta: Q \times \Gamma$ ... 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 $q_{\text{reject}} \neq q_{\text{accept}}$.





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

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

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

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

"The following must be TRUE for <u>every</u> cell *i,j*"

"The variable for <u>one</u> *s* must be TRUE"

<u>And only one</u> variable for some *s* must be TRUE

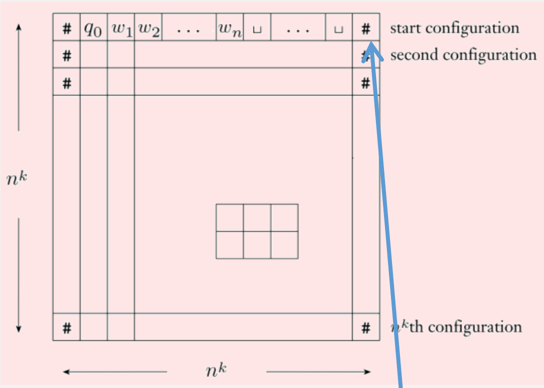
i.e., **every cell has a valid character**

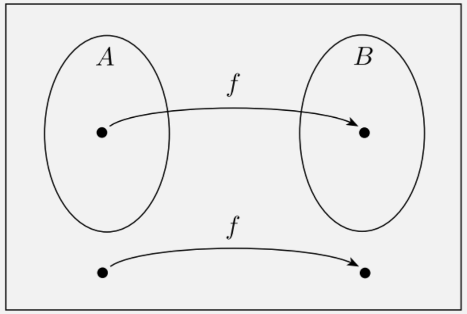
⇒ Does an <u>accepting tableau</u> correspond to a satisfiable (sub)formula?

- **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?

- Not necessarily





ϕ_{cell} ☑ $\wedge$ ϕ_{start} $\wedge$ ϕ_{move} $\wedge$ ϕ_{accept}

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

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

The variables in the start config, ANDed together

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

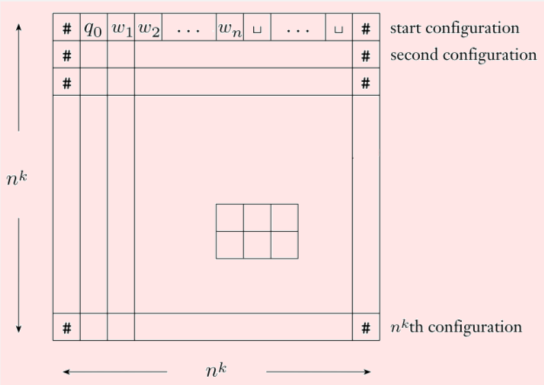
i.e., **tableau has valid start config**

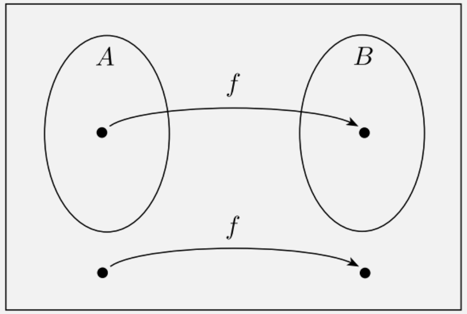
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **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





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

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

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

The state q_{accept} must appear in some cell

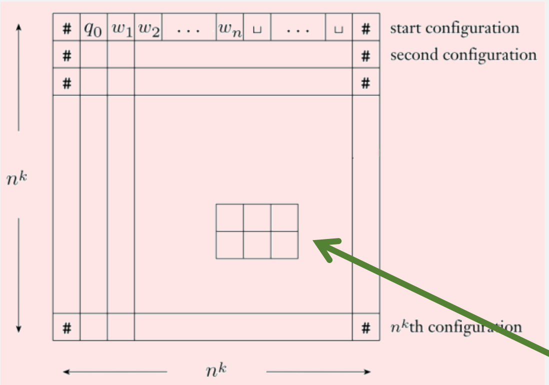
i.e., **tableau has valid accept config**

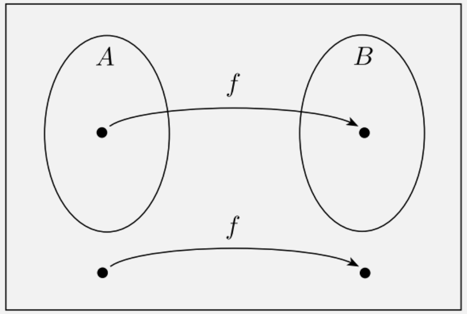
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **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?

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





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

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

- Ensures that every configuration is <u>legal</u> 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 <u>legal</u>?

(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)

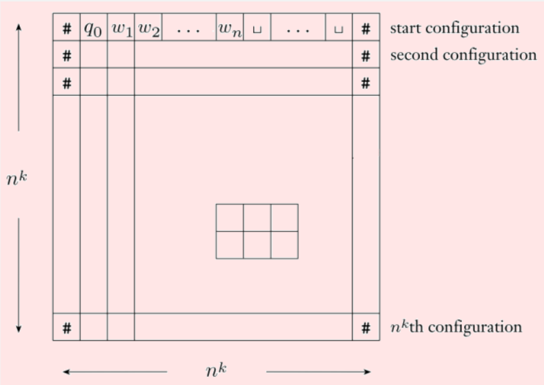
#	b	a
#	b	a

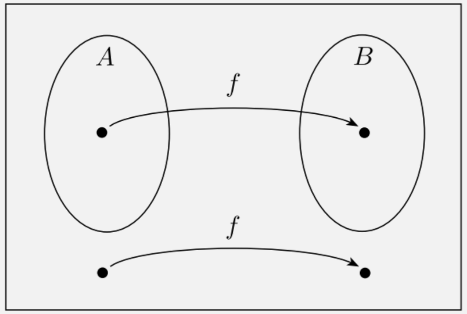
(e)

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 ☑

ϕ_{cell} ☑ $\wedge$ ϕ_{start} ☑ $\wedge$ ϕ_{move} $\wedge$ ϕ_{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

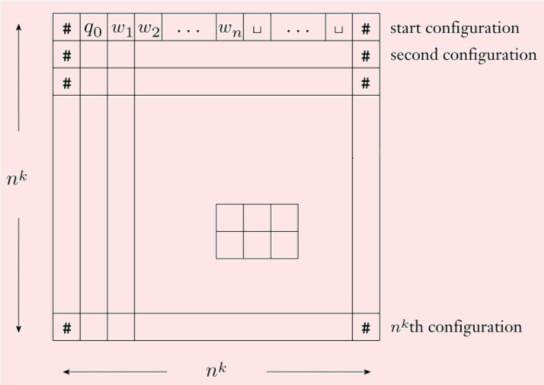
$$\bigvee_{\substack{a_1,\ldots,a_6 \\ \text{is a legal window}}} \left(x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}\right)$$

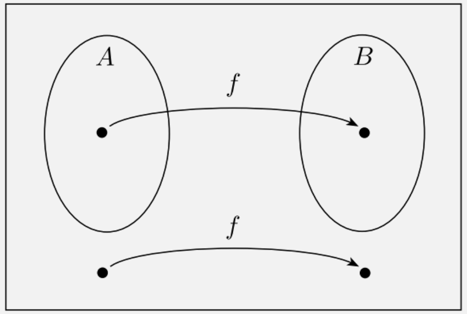
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?

- **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





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

ϕ_{cell} ✅ $\wedge$ ϕ_{start} ✅ $\wedge$ ϕ_{move} ✅ $\wedge$ ϕ_{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

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

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

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

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

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

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

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

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

The variables in the start config, ANDed together

Time complexity of the reduction

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

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

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

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

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

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

The state q_{accept} must appear in some cell

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

Time complexity of the reduction

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

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

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

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

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

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

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

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

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

Time complexity of the reduction

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

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

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

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

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

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

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

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

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

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

To Show Poly Time Mapping Reducibility …

Language A is ***polynomial time mapping reducible***, or simply ***polynomial time reducible***, to language B, written $A \leq_P B$, 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>:

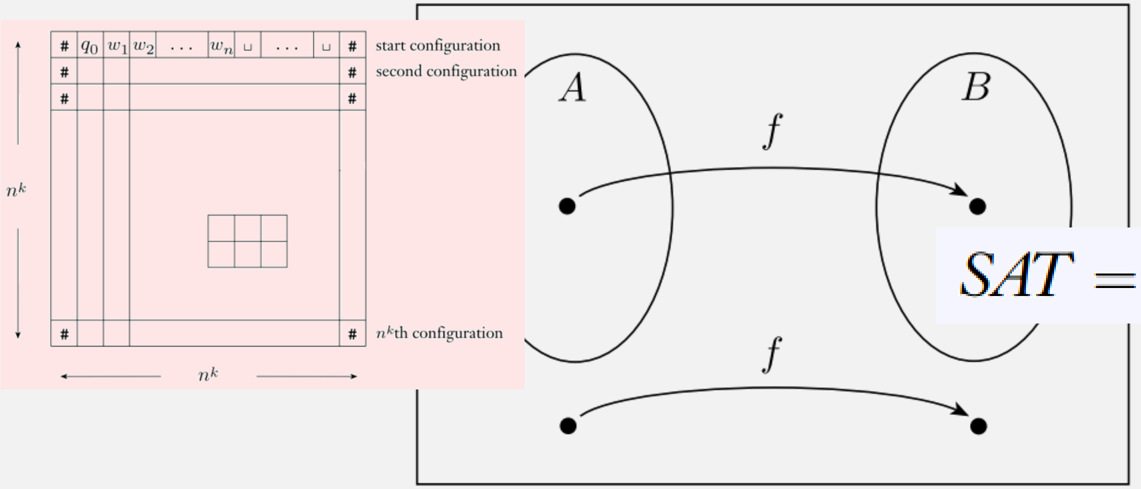
☑ 1. create **computable fn**,
☑ 2. show that it **runs in poly time**,
☑ 3. then show **forward direction** of mapping red.,
4. and **reverse direction**
☑ (or **contrapositive** of **forward direction**)

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.



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

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

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

THEOREM

known — unknown

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

To use this theorem, C must be in **NP**

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.

Proof:

- Need to show: C is **NP**-complete:
  - it's in **NP** (given), and
  - every lang A in **NP** reduces to C in poly time (must show)
- For every language A in **NP**, reduce $A \rightarrow C$ by:
  - First reduce $A \rightarrow B$ in poly time
    - Can do this because B is **NP**-Complete
  - Then reduce $B \rightarrow C$ in poly time
    - This is given
- Total run time: Poly time + poly time = poly time

If you're not Stephen Cook or Leonid Levin, **use this theorem to prove a language is 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**)

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

Example:

Let $C = 3SAT$, to prove *3SAT* is **NP**-Complete:

1. Show *3SAT* is in **NP**

Flashback: 3*SAT* is in **NP**

$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

Let $\boldsymbol{n}$ = the number of variables in the formula

Verifier:

On input $<\phi, c>$, where c is a possible assignment of variables in ϕ to values:

- Accept if c satisfies ϕ

Running Time: $O(\boldsymbol{n})$

Non-deterministic Decider:

On input $<\phi>$, where ϕ is a boolean formula:

- Non-deterministically try all possible assignments in parallel
- Accept if any satisfy ϕ

Running Time: Checking each assignment takes time $O(\boldsymbol{n})$

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 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

Example:

Let $C = 3SAT$, to prove $3SAT$ is **NP**-Complete:

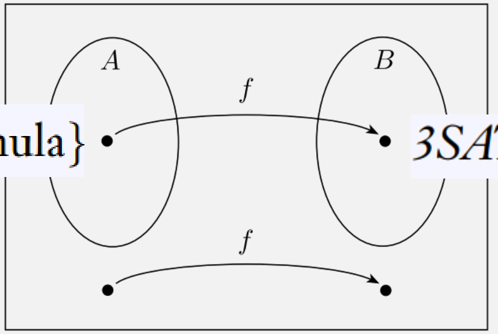
☑ 1. Show $3SAT$ is in **NP**

☑ 2. Choose B, the **NP**-complete problem to reduce from: SAT

3. Show a poly time mapping reduction from SAT to $3SAT$

Flashback: *SAT* is Poly Time Reducible to *3SAT*

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



$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

Need: poly time computable fn converting a Boolean formula ϕ to 3CNF:

1. Convert ϕ to CNF (an AND of OR clauses)
   a) Use DeMorgan's Law to push negations onto literals

   $\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$ $\quad$ $\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$ $\quad$ $O(n)$

   b) Distribute ORs to get ANDs outside of parens

   $(P \vee (Q \wedge R)) \Leftrightarrow ((P \vee Q) \wedge (P \vee R))$ $\quad$ $O(n)$

2. Convert to 3CNF by adding new variables

   $(a_1 \vee a_2 \vee a_3 \vee a_4) \Leftrightarrow (a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$ $\quad$ $O(n)$

Remaining step: show iff relation holds …

… easy for formula conversion: each step is already a known "law"

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 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

Example:

Let $C = 3SAT$, to prove $3SAT$ is **NP**-Complete:

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

Each NP-complete problem we prove makes it easier to prove the next one!

THEOREM

Next Time: 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 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

Example:

Let C = ~~*3SAT*~~ ***CLIQUE***, to prove ~~*3SAT*~~ ***CLIQUE*** is **NP**-Complete:

? 1. Show ~~*3SAT*~~ ***CLIQUE*** is in **NP**

? 2. Choose B, the **NP**-complete problem to reduce from: ~~*SAT*~~ ***3SAT***

? 3. Show a poly time mapping reduction from B to C

Check-in Quiz 5/2

On gradescope