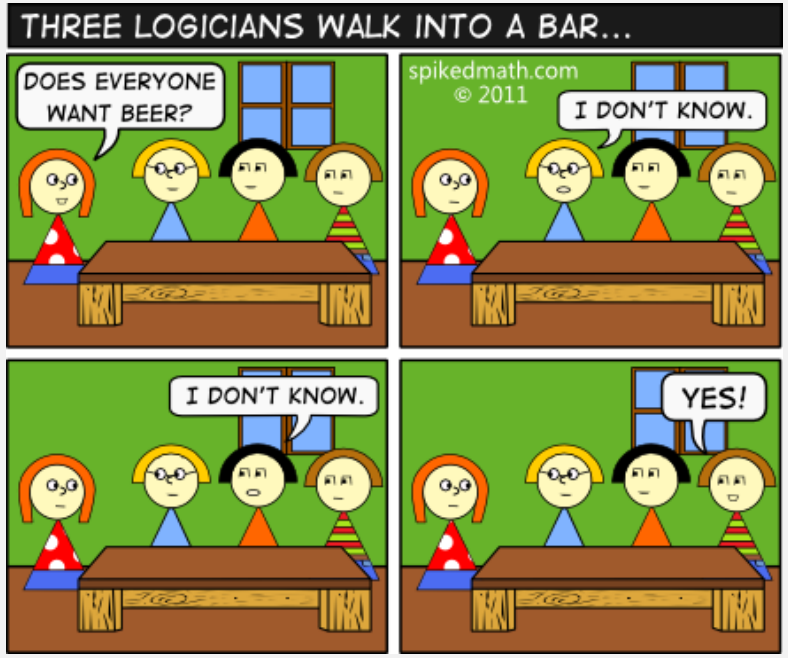
# The Cook-Levin Theorem

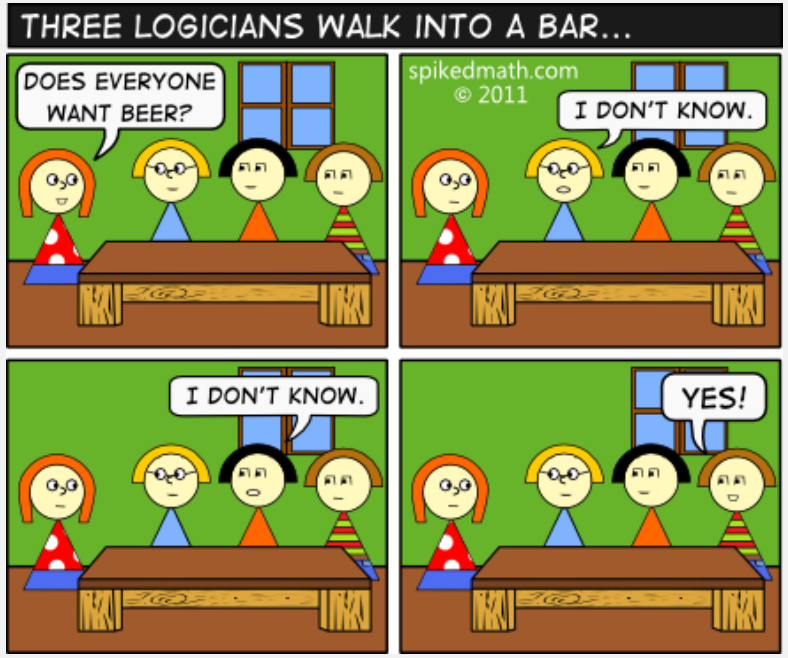
## (i.e., the first NP-Complete Problem)

Monday, May 3, 2021



# Announcements

- HW11 due Tues 5/4 11:59pm EST
  - Note the update to Problem 5, Part 2
- HW12 out soon
  - Due Tues 5/11 11:59pm EST
  - Last HW
- No Final Exam in this course



# Today: The Cook-Levin Theorem

**THEOREM 7.37**

*SAT* is NP-complete

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

*КРАТКИЕ СООБЩЕНИЯ*

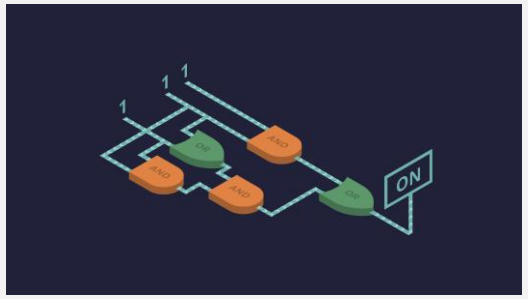
УДК 519.14

**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 7.34**

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 $B$.

# To Show Poly Time Mapping Reducibility …

**DEFINITION 7.29**

Language $A$ is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language $B$, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ exists, where for every $w$,

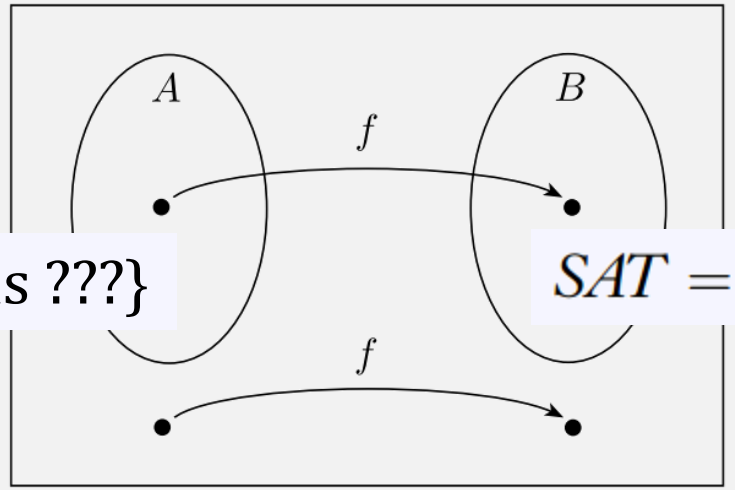
$$w \in A \iff f(w) \in B.$$

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

1. Create a computable fn $f$ converting a string in lang $A$ to one in $B$
2. Show that it runs in polynomial time
3. Show that the "if and only if" relation holds:
   => if $w$ in $A$, then $f(w)$ in $B$
   <= if $f(w)$ in $B$, then $w$ in $A$
   <= (alternative), show contrapositive: if $w$ not in $A$, then $f(w)$ not in $B$

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

Some **NP** lang = {w | w is ???}




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

How can we come up with reduction of some *w* to a Boolean formula if we don't know *w*???

# Proving theorems about an entire class of langs?

- We still know some general things about the languages

**THEOREM 1.45**

- E.g., The class of regular languages is closed under the union operation.
  - **PROOF** uses the theorem that every reg lang has an NFA accepting it

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize $A_1$, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize $A_2$.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

Proof is a algorithm for constructing a union-recognizing NFA from any two NFAs

**THEOREM 4.7**

- E.g., $A_{\text{CFG}}$ is a decidable language. $A_{\text{CFG}} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates string } w\}$

Proof uses the theorem that every CFG has a **Chomsky Normal Form**

# What do we know about strings in **NP** langs?

- They are:
  - Verified by a deterministic poly time verifier (NP definition)
  - Decided by a nondeterministic poly time decider (NTM) (Thm 7.20)

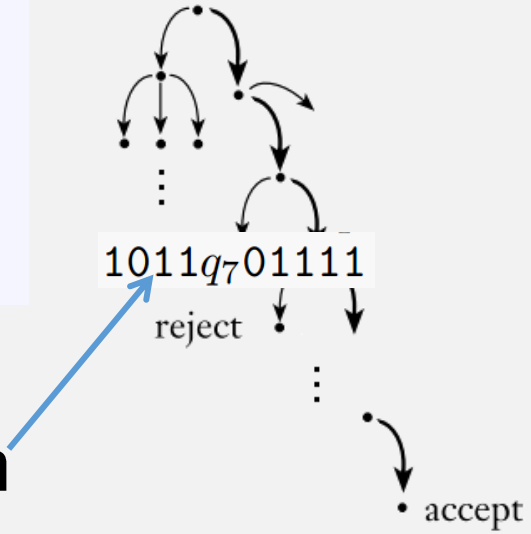
Let's use this one

# Review: Non-deterministic TMs

- Formally defined with states, transitions, alphabet …

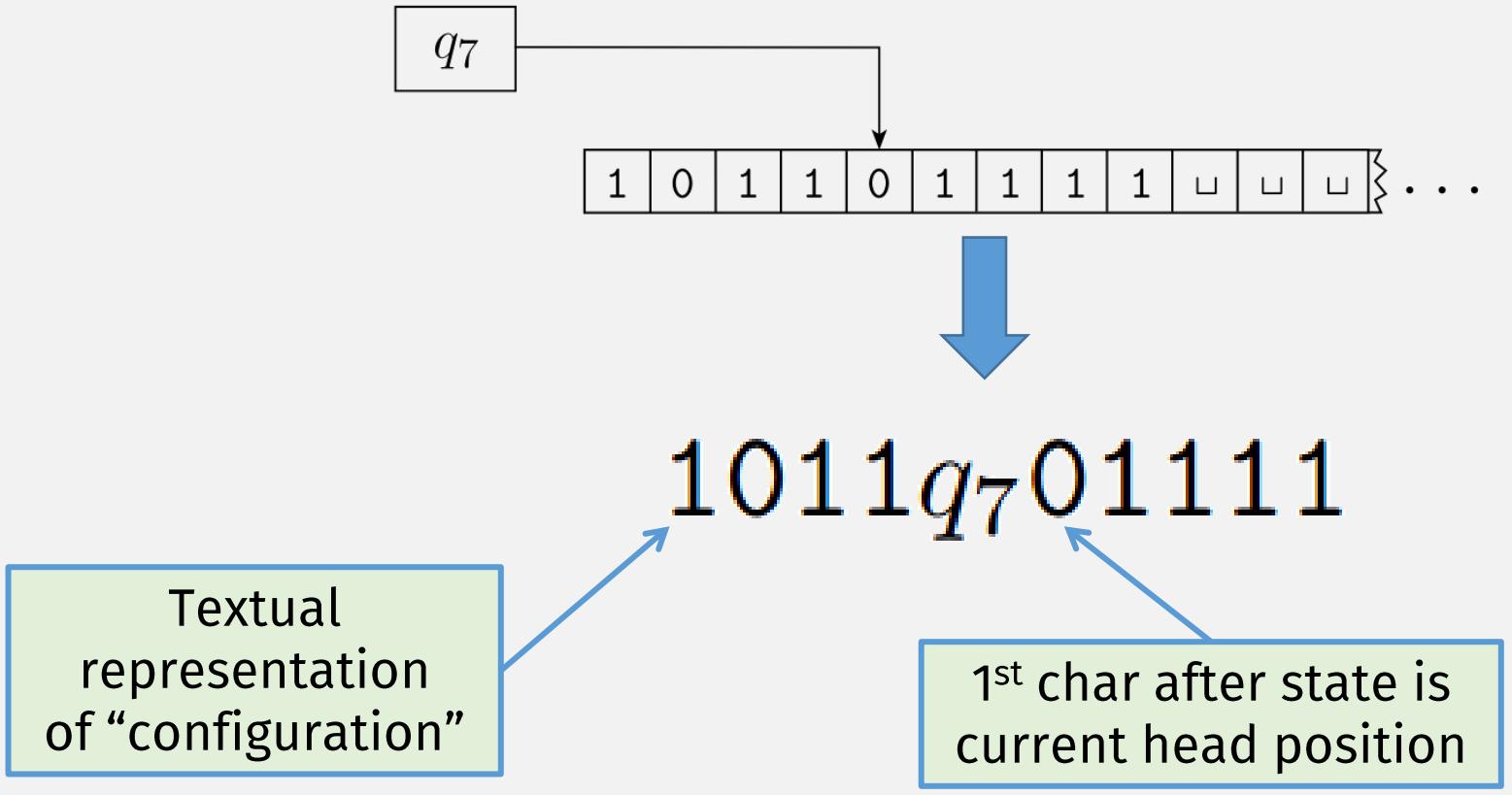
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $Q$, $\Sigma$, $\Gamma$ are all finite sets and

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




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

# Review: TM Config = State + Head + Tape



# Review: Non-deterministic TMs

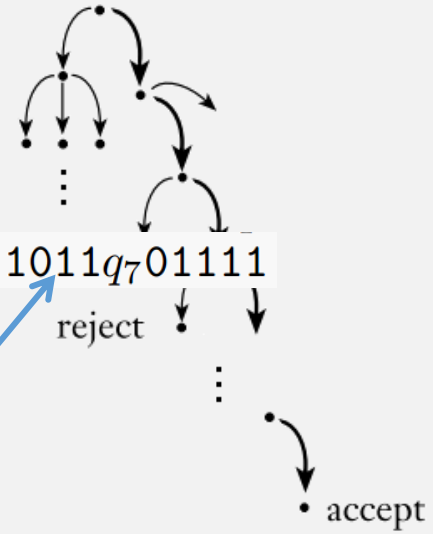
- Formally defined with states, transitions, alphabet …

Idea: **We don't know the specific language or strings in the language, but …**

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

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

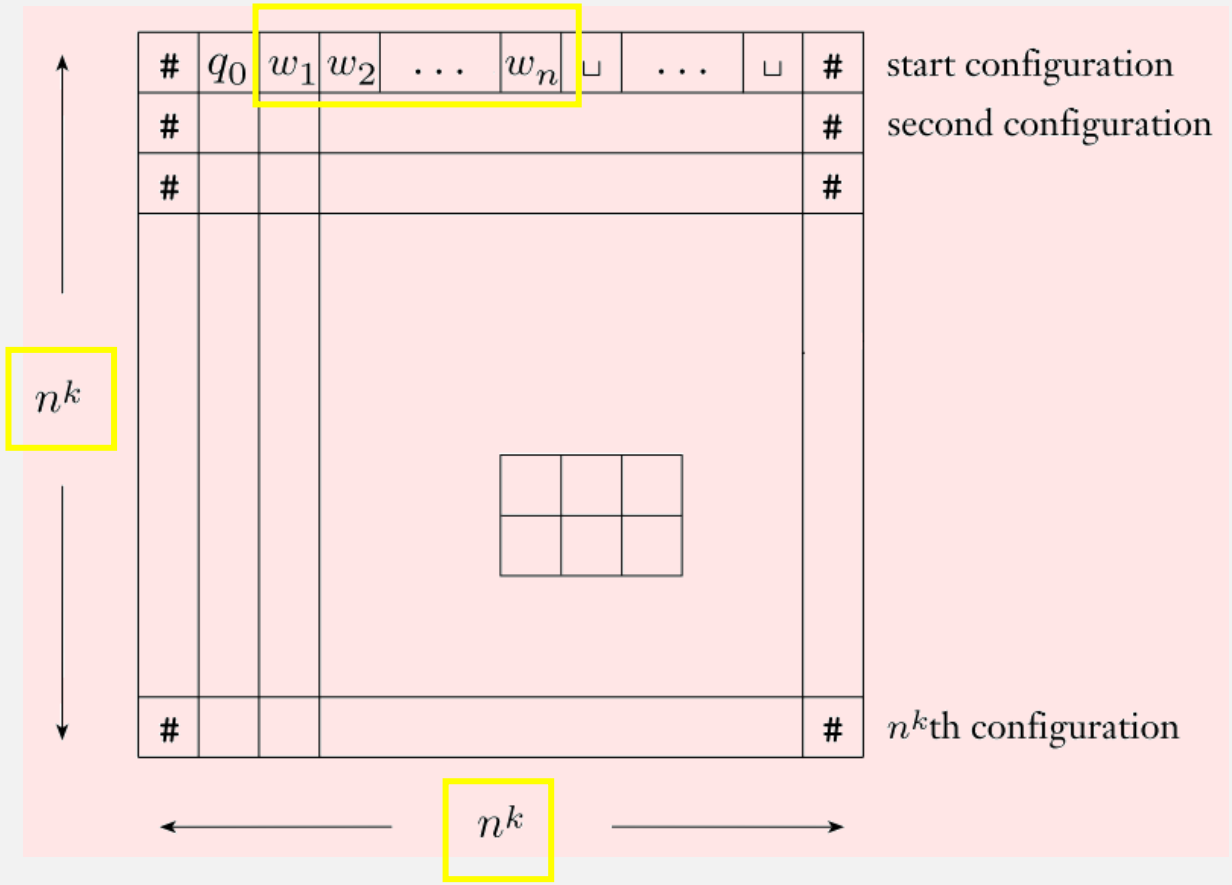
1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet not containing the *blank symbol* $\sqcup$,
3. $\Gamma$ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\})$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




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

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

# Accepting config sequence = “Tableau”

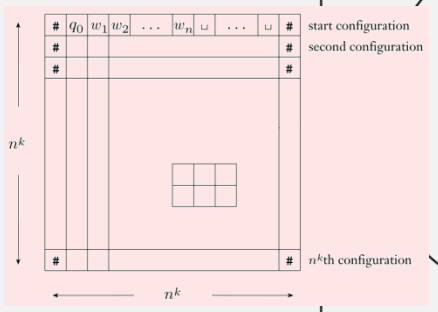


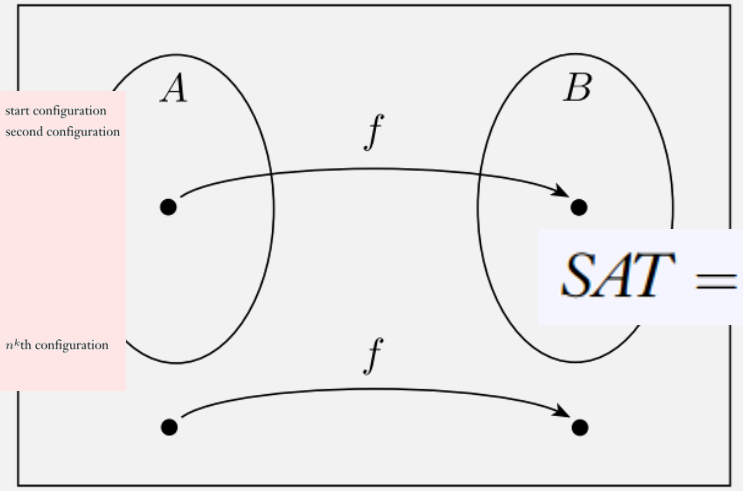

- input $w = w_1 \ldots w_n$
- To simplify proof, assume configs start/end with #
- Some config must be accepting config
- At most $n^k$ configs
  - (why?)
- Each config has length $n^k$
  - (why?)

# 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:

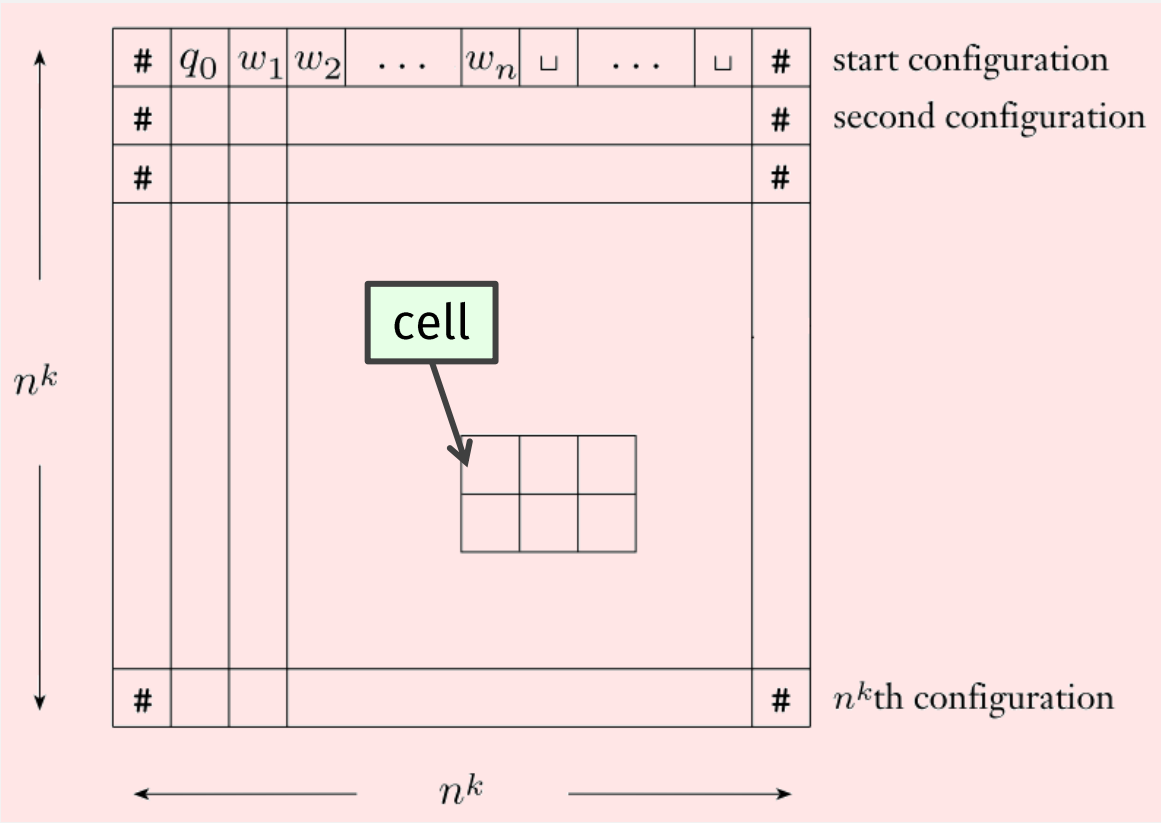
$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$

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

# Tableau Terminology

- A tableau <u>cell</u> has coordinate *i,j*

- A cell has <u>symbol</u>:
  $s \in C = Q \cup \Gamma \cup \{\#\}$

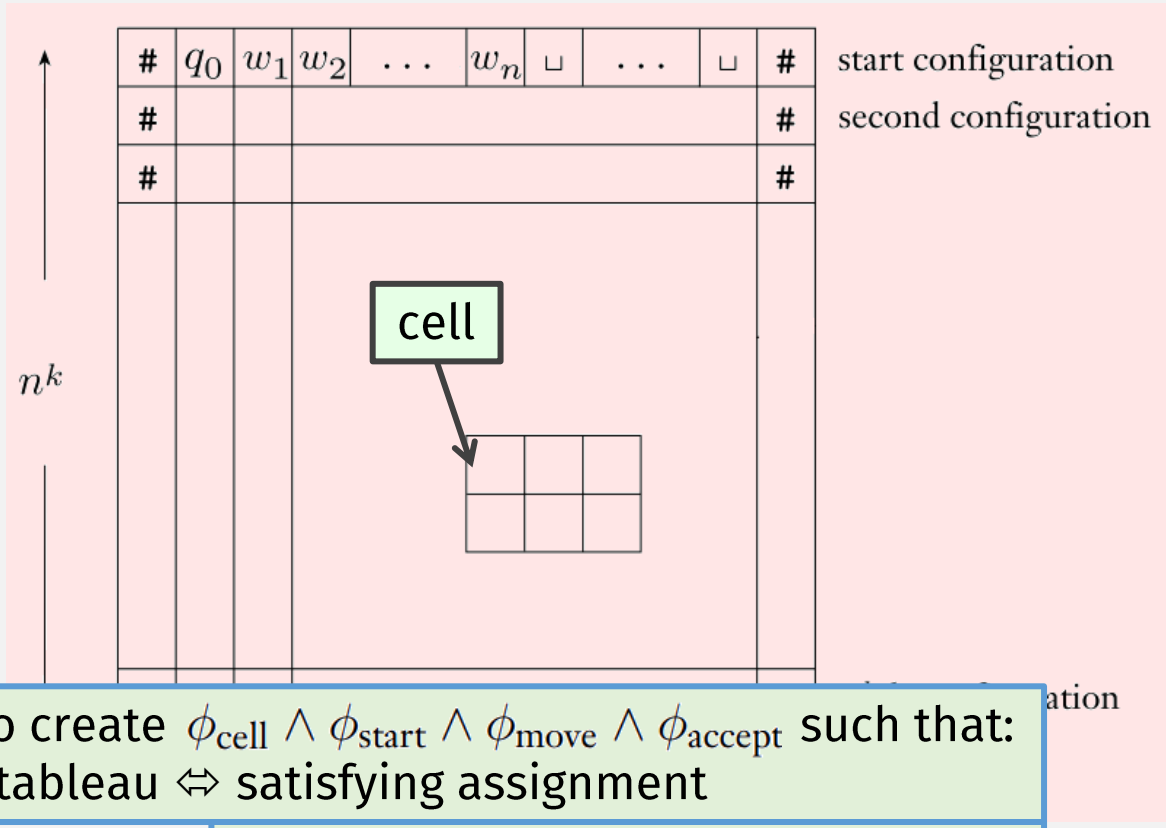



A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $Q$, $\Sigma$, $\Gamma$ are all finite sets and

1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. $\Gamma$ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\})$ is the 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 has symbol:
  $s \in C = Q \cup \Gamma \cup \{\#\}$
- For every $i,j,s$ create variable $x_{i,j,s}$
  - i.e., one var for every possible symbol/cell combination
- Total variables =
  - # cells * # symbols =
  - $n^k * n^k * |C| = O(\boldsymbol{n^{2k}})$

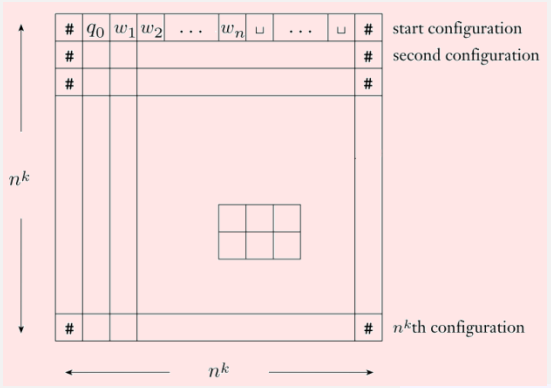


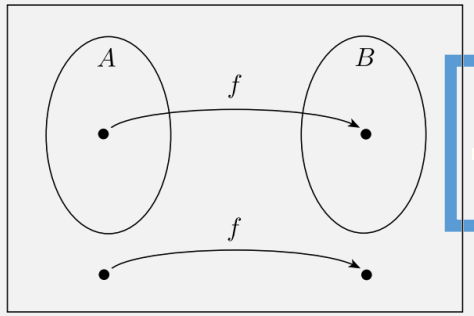

Use these variables to create $\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$ such that:
accepting tableau ⇔ satisfying assignment

- For accepting tableau:
  - **all four parts** must be TRUE
- For non-accepting tableau
  - **only one part** must be FALSE

A *Turing ma* ... where
$Q, \Sigma, \Gamma$ are a

1. $Q$ is th
2. $\Sigma$ is the ... ⊔,
3. $\Gamma$ is the tape alphabet, where ⊔ ∈ $\Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\})$ 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 $q_{\text{reject}} \neq q_{\text{accept}}$.




accepting tableau: **all four** must be TRUE
non-accepting tableau: **one** must be FALSE

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

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

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

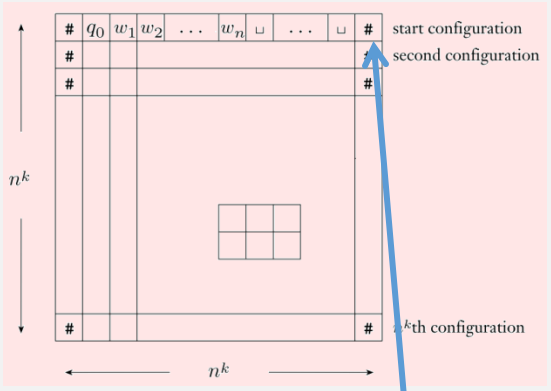
"The following must be TRUE for <u>every</u> cell *i,j*"

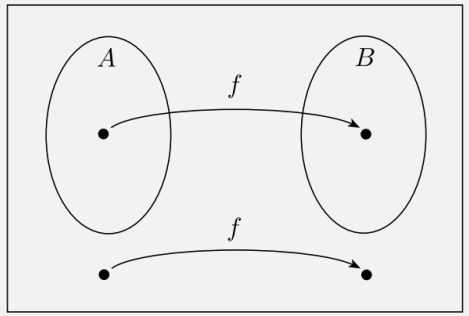
"The variable for <u>one</u> *s* must be TRUE"

<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




accepting tableau: **all four** must be TRUE
non-accepting tableau: **one** must be FALSE

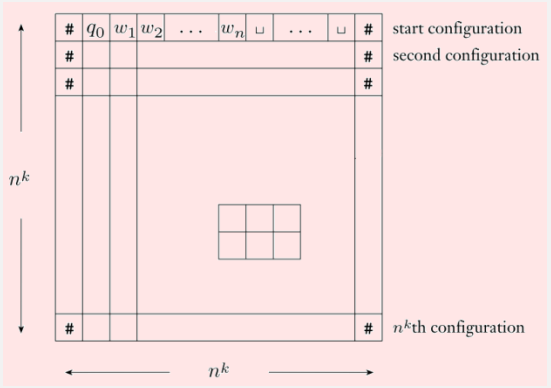
$$\phi_{\text{cell}} \wedge \boxed{\phi_{\text{start}}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

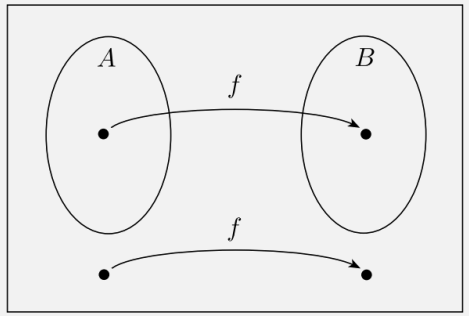
The variables in the start config, ANDed together

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

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
non-accepting tableau: **one** must be FALSE

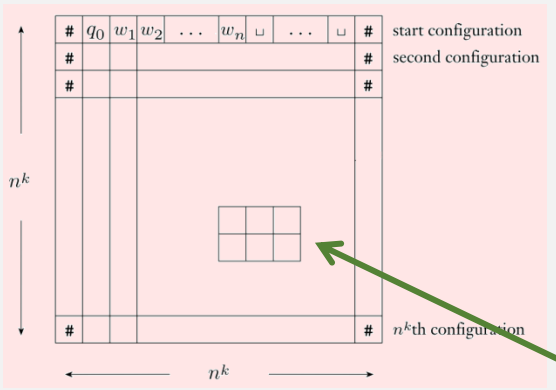
$$\phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{move}} \wedge \phi_{\text{accept}}$$

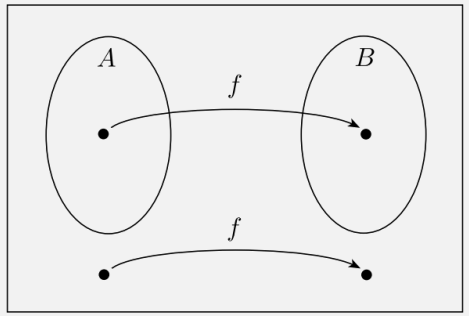
$$\phi_{\text{accept}} = \bigvee_{1 \le i,j \le n^k} x_{i,j,q_{\text{accept}}}$$

The state $q_{\text{accept}}$ 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 $q_{\text{accept}}$

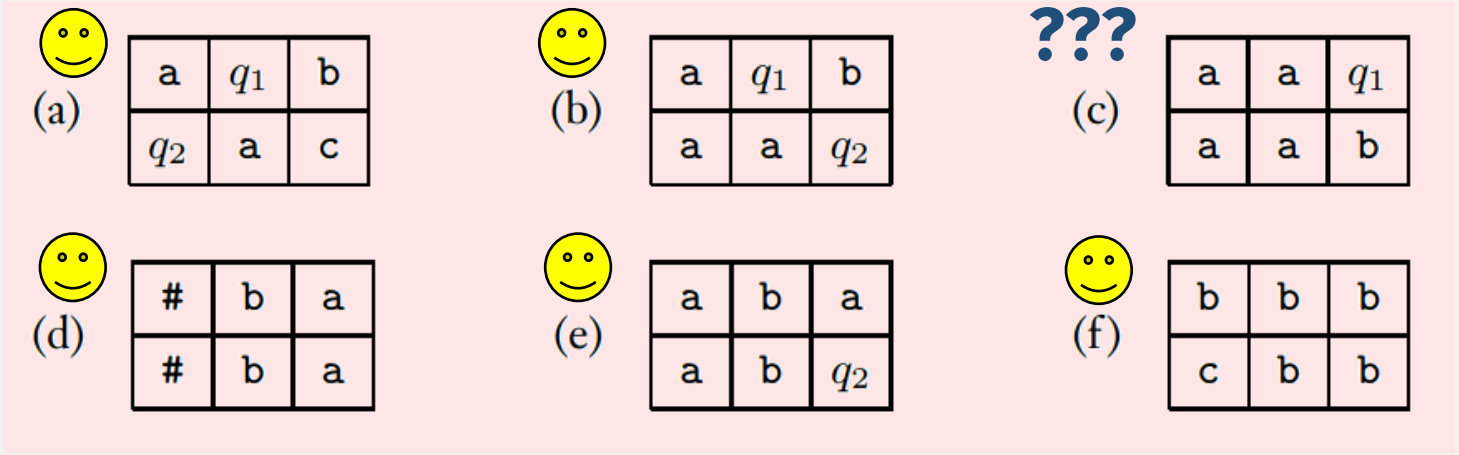


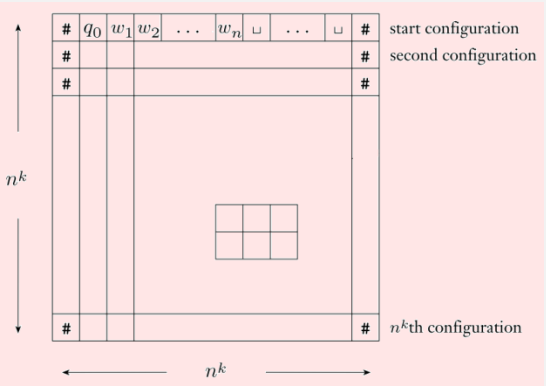



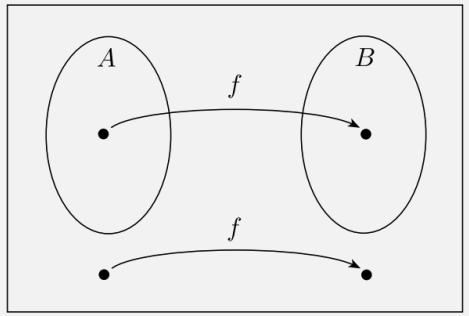

accepting tableau: **all four** must be TRUE
non-accepting tableau: **one** must be FALSE ☑

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

- Ensures that every configuration is legal according to the previous configuration and the TM's $\delta$ transitions
- Only need to verify every 2x3 "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?






accepting tableau: **all four** must be TRUE
non-accepting tableau: **one** must be FALSE ☑

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

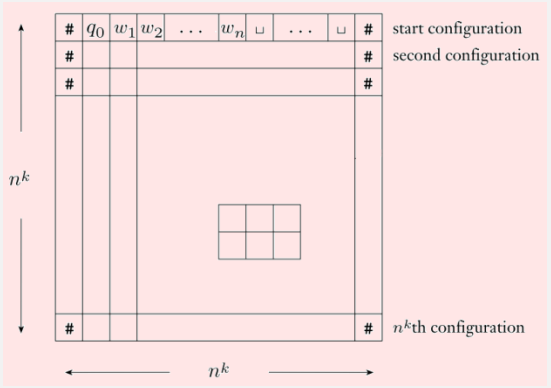
i.e., **all transitions are legal, according to delta fn**

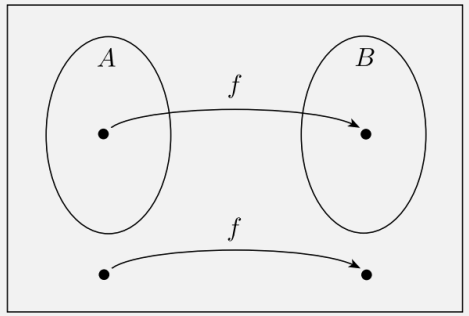
$$\phi_{\text{move}} = \bigwedge_{1 \le i < n^k,\ 1 < j < n^k} (\text{the } (i,j)\text{-window is legal})$$

*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




accepting tableau: **all four** must be TRUE ☑
non-accepting 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

# To Show Poly Time Mapping Reducibility …

**DEFINITION 7.29**

Language $A$ is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language $B$, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ 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 $B$.

- ☑ 1. Create a computable fn $f$ converting a string in lang $A$ to one in $B$
- 2. Show that it runs in polynomial time
- ☑ 3. Show that the "if and only if" relation holds:
  - ☑ => if $w$ in $A$, then $f(w)$ in $B$
    <= if $f(w)$ in $B$, then $w$ in $A$
  - ☑ <= (alternative), show contrapositive: if $w$ not in $A$, then $f(w)$ not in $B$

# Time complexity of the reduction

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

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

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

"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

# Time complexity of the reduction

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

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

$O(n^{2k})$

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

$O(n^k)$

The variables in the start config, ANDed together

# Time complexity of the reduction

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

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

$O(n^{2k})$

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

$O(n^k)$

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

The state $q_{\text{accept}}$ must appear in some cell

$O(n^{2k})$

# Time complexity of the reduction

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

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

$O(n^{2k})$

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

$O(n^k)$

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

$O(n^{2k})$

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

$O(n^{2k})$

# Time complexity of the reduction

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

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

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

$O(n^{2k})$

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

$O(n^k)$

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

$O(n^{2k})$

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

$O(n^{2k})$

# To Show Poly Time Mapping Reducibility …

**DEFINITION 7.29**

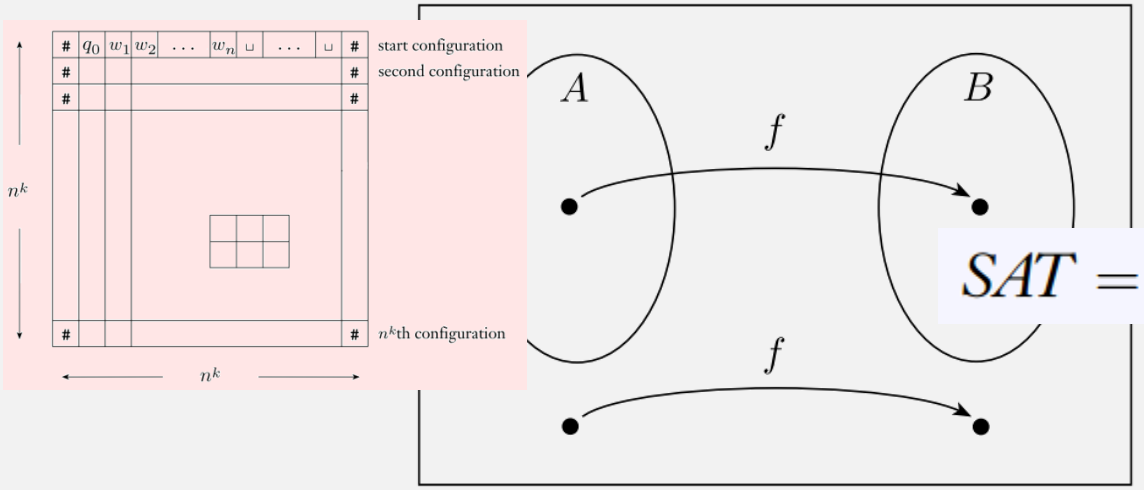
Language $A$ is ***polynomial time mapping reducible***,[1] or simply ***polynomial time reducible***, to language $B$, written $A \leq_{\mathrm{P}} B$, if a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ exists, where for every $w$,

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

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

- ☑ 1. Create a computable fn $f$ converting a string in lang $A$ to one in $B$
- ☑ 2. Show that it runs in polynomial time
- ☑ 3. Show that the "if and only if" relation holds:
  - ☑ => if $w$ in $A$, then $f(w)$ in $B$
    <= if $f(w)$ in $B$, then $w$ in $A$
  - ☑ <= (alternative), show contrapositive: if $w$ not in $A$, then $f(w)$ not in $B$

# QED: *SAT* is NP-complete




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

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

**THEOREM 7.36**

known unknown

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

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

Proof:

- For every language $A$ in **NP**, reduce $A$ to $C$ by:
  - First use the reduction from $A$ to $B$
    - This exists because $B$ is **NP**-Complete
  - Then $B$ to $C$
    - This is given
- This runs in poly time because of the definition of **NP**-completeness and poly time reducibility

# Theorem: *3SAT* is NP-complete.

- Proof: To use thm 7.36, must show poly time reduction from:
  - *SAT* (known to be **NP**-Complete)
  - to *3SAT* (known to be in **NP**)

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

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

- Given an arbitrary *SAT* formula:
  1. First convert to CNF (an AND of OR clauses)
     - Use DeMorgan's Law to push negations onto literals

$$\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q) \qquad \neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$$

     - Distribute ORs to get ANDs outside of parens

$$(P \vee (Q \wedge R)) \Leftrightarrow ((P \vee Q) \wedge (P \vee R))$$

     - Then convert to 3cnf by adding new variables

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

$O(n)$

$O(n)$

$O(n)$

Remaining step: show iff relation holds

**THEOREM 7.36**

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

# Check-in Quiz 5/3

On gradescope