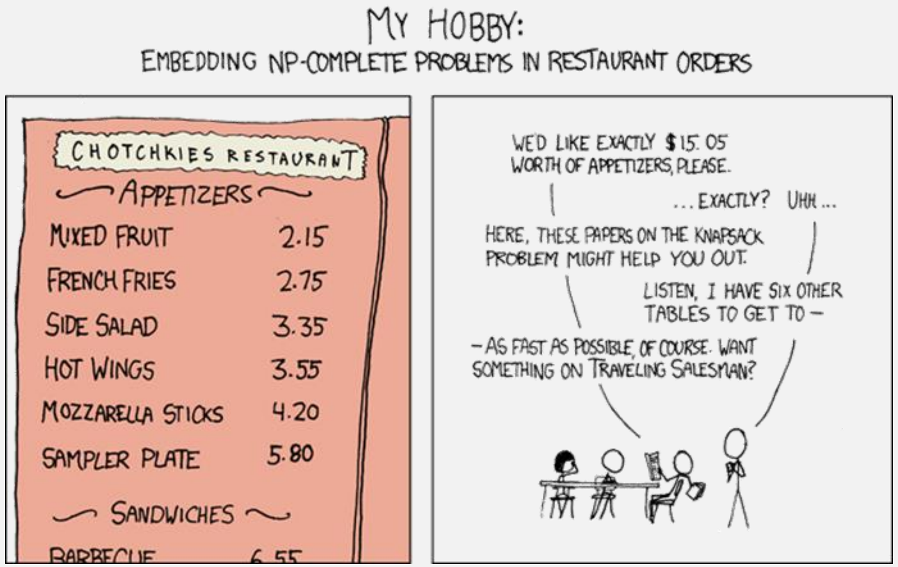
UMB CS 622

NP-Completeness

Monday, November 15, 2021



Announcements

- HW8 due Wed 11:59pm
- Good HW discussions on Piazza

Last Time: Verifiers, Formally

$PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$

An alternate way to define a decidable language

extra argument:
can be any string that helps to find a result in poly time
(is often just a result itself)

A ***verifier*** for a language A is an algorithm V, where

$$A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}.$$

certificate, or ***proof***

We measure the time of a verifier only in terms of the length of w, so a ***polynomial time verifier*** runs in polynomial time in the length of w. A language A is ***polynomially verifiable*** if it has a polynomial time verifier.

- Cert c has length at most n^k, where n = length of w

Last Time: The class **NP**

DEFINITION

NP is the class of languages that have polynomial time verifiers.

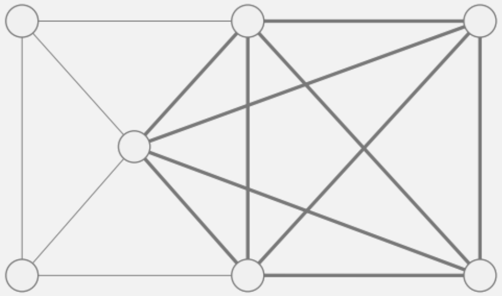
2 ways to show that a language is in **NP**

THEOREM

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.

Last Time: **NP** Problems

- $CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$
  - A clique is a subgraph where every two nodes are connected
  - A k-clique contains k nodes



- $SUBSET\text{-}SUM = \{\langle \overset{\text{set}}{S}, \overset{\text{sum}}{t}\rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \underbrace{\{y_1, \ldots, y_l\}}_{\text{subset}} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = \underbrace{t}_{\text{sum}}\}$
  - Some subset of a set of numbers S must sum to a total t
  - e.g., $\langle\{\boxed{4}, 11, 16, \boxed{21}, 27\}, 25\rangle \in SUBSET\text{-}SUM$

Theorem: *SUBSET-SUM* is in NP

$$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}$$

PROOF IDEA The subset is the certificate.

To prove a lang is in **NP**, create <u>either</u>:
- **Deterministic** poly time **verifier**
- **Nondeterministic** poly time **decider**

PROOF The following is a verifier V for *SUBSET-SUM*.

V = "On input $\langle\langle S, t\rangle, c\rangle$:
1. Test whether c is a collection of numbers that sum to t.
2. Test whether S contains all the numbers in c.
3. If both pass, *accept*; otherwise, *reject*."

Does this run in poly time?

Proof 2: *SUBSET-SUM* is in NP

$$\textit{SUBSET-SUM} = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t\}$$

To prove a lang is in **NP**, create either:
- **Deterministic** poly time **verifier**
- **Nondeterministic** poly time **decider**

ALTERNATIVE PROOF We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for *SUBSET-SUM* as follows.

N = "On input $\langle S, t\rangle$:
1. Nondeterministically select a subset c of the numbers in S.
2. Test whether c is a collection of numbers that sum to t.
3. If the test passes, *accept*; otherwise, *reject*."

Nondeterministically runs the verifier many times in parallel

Does this run in poly time?

Last Time: **NP** vs **P**

P

The class of languages that have a **deterministic** poly time **decider**

I.e., the class of languages that can be solved "quickly"

- We want search problems to be in here … but they often are not

NP

The class of languages that have a **deterministic** poly time **verifier**

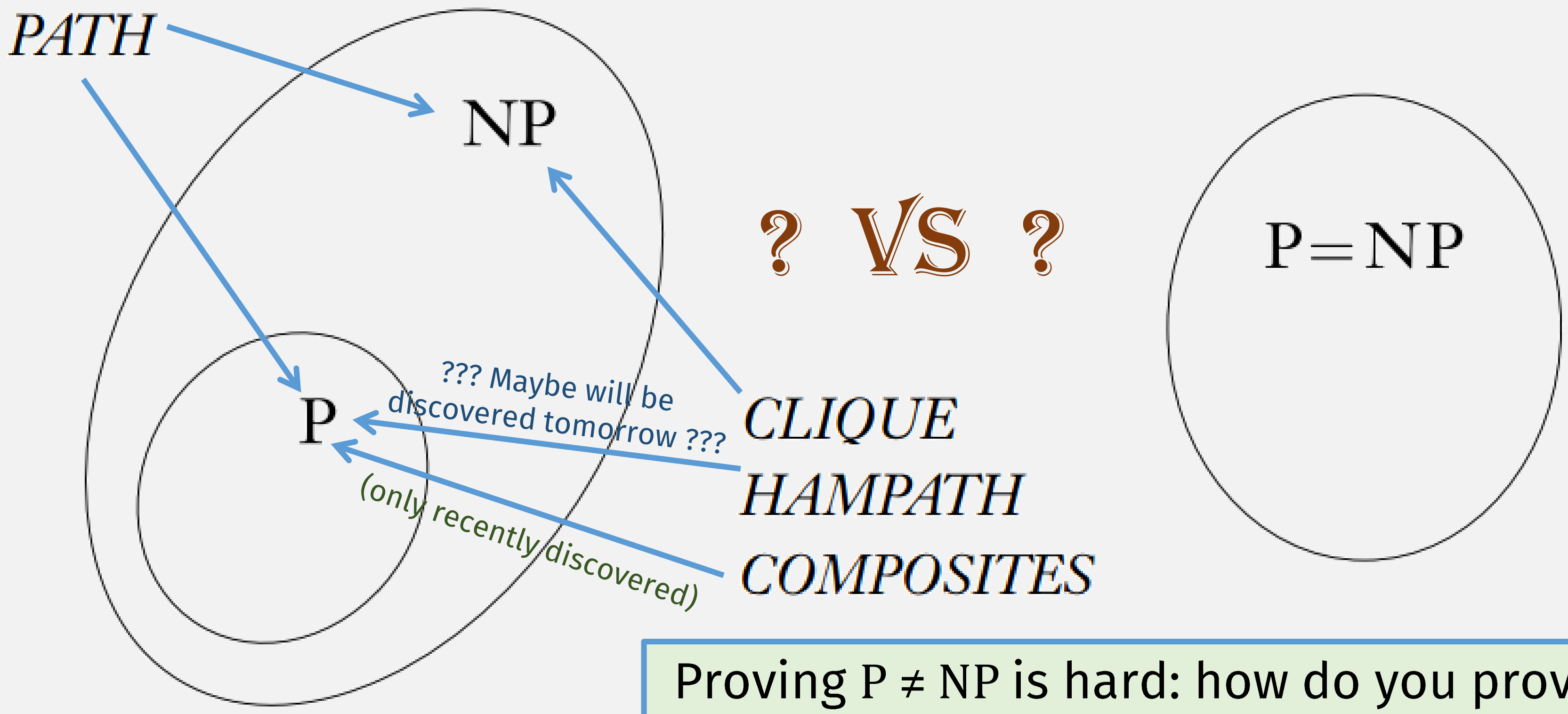
Also, the class of languages that have a **nondeterministic** poly time **decider**

I.e., the class of language that can be verified "quickly"

- Search problems, even those not in **P**, are often in here

One of the Greatest unsolved

~~HW~~ Question: Does P = NP?

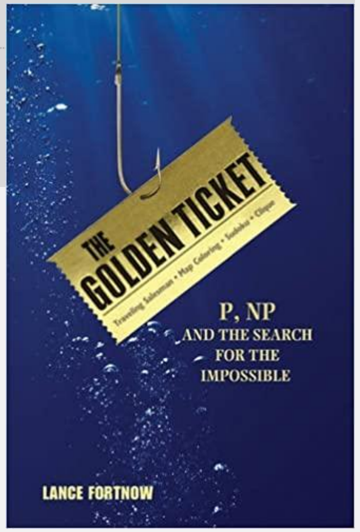


Proving P ≠ NP is hard: how do you prove that an algorithm won't ever have a poly time solution?
(in general, it's hard to prove that something doesn't exist)

Not Much Progress on whether $\mathbf{P} = \mathbf{NP}$?

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:
  - $\mathbf{NP}$-Completeness

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

What's this?

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

THEOREM

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

Flashback: Mapping Reducibility

Language A is ***mapping reducible*** to language B, written $\boxed{A \leq_{\mathrm{m}} B}$, if there is a $\boxed{\text{computable function } f\colon \Sigma^* \longrightarrow \Sigma^*}$, where for every w,

$$\boxed{w \in A \Longleftrightarrow f(w) \in B.}$$

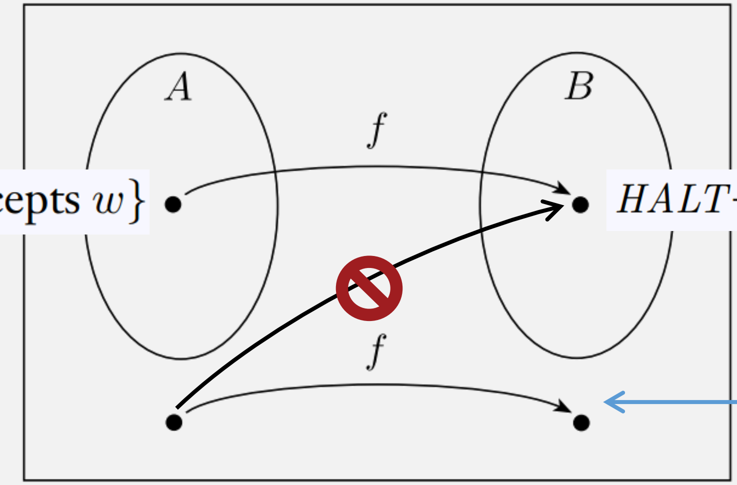
IMPORTANT: "if and only if" …

The function f is called the ***reduction*** from A to B.

To show mapping reducibility:

1. create **computable fn**
2. and then show **forward direction**
3. and **reverse direction** (or **contrapositive of forward direction**)

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$



$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

… means $\boxed{\overline{A} \leq_{\mathrm{m}} \overline{B}}$

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Polynomial Time Mapping Reducibility

Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

The function f is called the ***reduction*** from A to B.

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

$$\boxed{w \in A \iff f(w) \in B.}$$

Don't forget: "if and only if" ...

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

poly time

poly time

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a ***computable function*** if some Turing machine M, on every input w, halts with just $f(w)$ on its tape.

Flashback: If $A \leq_m B$ and B is decidable, then A is decidable.

Has a decider

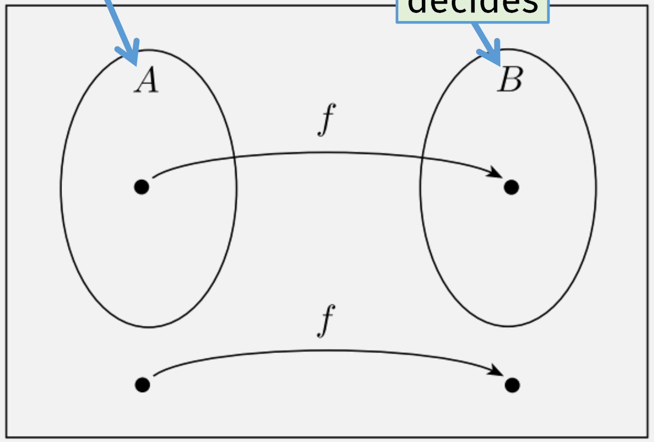
PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

decides

decides



This proof only works because of the if-and-only-if requirement

Language A is ***mapping reducible*** to language B, written $A \leq_m B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$\boxed{w \in A \iff f(w) \in B.}$$

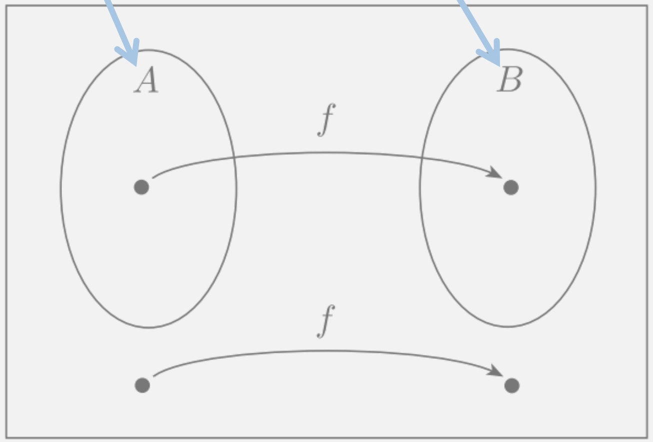
The function f is called the ***reduction*** from A to B.

Thm: If $A \leq_{\cancel{\mathrm{m}}\,\mathbf{P}} B$ and B ~~is decidable,~~ $\in \mathrm{P}$ then A ~~is decidable.~~ $\in \mathrm{P}$

PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

N = "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."



Language A is ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

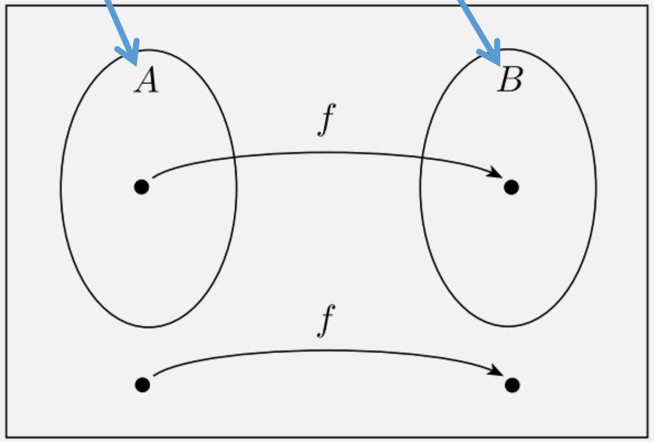
The function f is called the ***reduction*** from A to B.

Thm: If $A \leq_{\cancel{\mathrm{m}}\,\mathbf{P}} B$ and B ~~is decidable,~~ $\in$ P then A ~~is decidable.~~ $\in$ P

PROOF We let M be the poly time decider for B and f be the poly time reduction from A to B. We describe a poly time decider N for A as follows.

$N =$ "On input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."



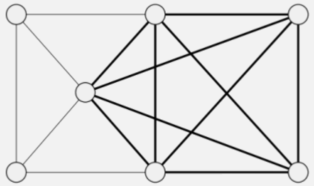
Language A is poly time ***mapping reducible*** to language B, written $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

The function f is called the ***reduction*** from A to B.

<u>Theorem</u>: *3SAT* is polynomial time reducible to *CLIQUE*.

Last Class: $CLIQUE$ is in NP



$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

PROOF IDEA The clique is the certificate.

PROOF The following is a $\boxed{\text{verifier } V}$ for *CLIQUE*.

$V =$ "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, *accept*; otherwise, *reject*."

Theorem: *3SAT* is polynomial time reducible to *CLIQUE*.

??

Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE

Boolean Formulas

A Boolean _____	Is …	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean **value**	x, y, z

Boolean Formulas

A Boolean _____	Is …	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean **value**	x, y, z
Operation	Combines Boolean **variables**	AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)

Boolean Formulas

A Boolean _____	Is …	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean **value**	x, y, z
Operation	Combines Boolean **variables**	AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)
Formula ϕ	Combines **vars** and **operations**	$(\overline{x} \wedge y) \vee (x \wedge \overline{z})$

Boolean Satisfiability

- A Boolean formula is <u>satisfiable</u> 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
  - $x = \mathrm{FALSE}$,
    $y = \mathrm{TRUE}$,
    $z = \mathrm{FALSE}$

The Boolean Satisfiability Problem

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

Theorem: SAT is in $\mathbf{NP}$:

- 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: *3SAT* is polynomial time reducible to *CLIQUE*.

??

More Boolean Formulas

A Boolean _____	Is …	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)
Formula ϕ	Combines vars and operations	$(\overline{x} \wedge y) \vee (x \wedge \overline{z})$

More Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)
Formula ϕ	Combines vars and operations	$(\overline{x} \wedge y) \vee (x \wedge \overline{z})$
Literal	A var or a negated var	x or $\overline{x}$.

More Boolean Formulas

A Boolean _____	Is …	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)
Formula ϕ	Combines vars and operations	$(\overline{x} \wedge y) \vee (x \wedge \overline{z})$
Literal	A var or a negated var	x or $\overline{x}$.
Clause	**Literals** ORed together	$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$

More Boolean Formulas

A Boolean _____	Is …	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)
Formula ϕ	Combines vars and operations	$(\overline{x} \wedge y) \vee (x \wedge \overline{z})$
Literal	A var or a negated var	x or $\overline{x}$.
Clause	**Literals** ORed together	$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$
Conjunctive Normal Form (**CNF**)	**Clauses** ANDed together	$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6)$

∧ = AND = “Conjunction”
∨ = OR = “Disjunction”
¬ = NOT = “Negation”

More Boolean Formulas

A Boolean _____	Is …	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)
Formula ϕ	Combines vars and operations	$(\overline{x} \wedge y) \vee (x \wedge \overline{z})$
Literal	A var or a negated var	x or $\overline{x}$.
Clause	**Literals** ORed together	$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$
Conjunctive Normal Form (**CNF**)	**Clauses** ANDed together	$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6)$
3CNF Formula	Three **literals** in each **clause**	$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

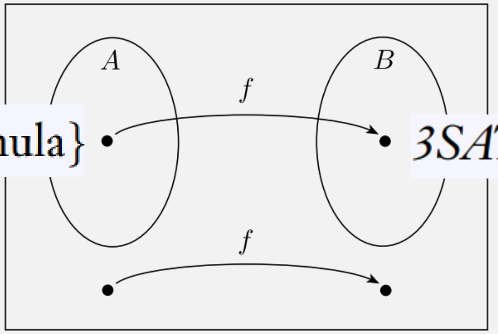
$\wedge$ = AND = "Conjunction"
$\vee$ = OR = "Disjunction"
$\neg$ = NOT = "Negation"

The *3SAT* Problem

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

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

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



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

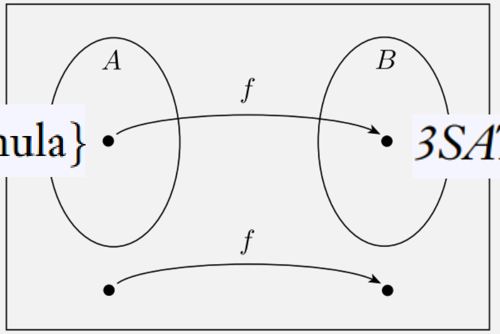
To show poly time mapping reducibility:

1. create **computable fn** f,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
   - $\Rightarrow$ if $\phi \in SAT$, then $f(\phi) \in 3SAT$
4. and **reverse direction**
   - $\Leftarrow$ if $f(\phi) \in 3SAT$, then $\phi \in SAT$

   (or **contrapositive** of **forward direction**)
   - $\Leftarrow$ (alternative) if $\phi \notin SAT$, then $f(\phi) \notin 3SAT$

Theorem: *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

      $\neg(P \vee Q) \boxed{\iff} (\neg P) \wedge (\neg Q)$ $\quad$ $\neg(P \wedge Q) \boxed{\iff} (\neg P) \vee (\neg Q)$ $\boxed{O(\boldsymbol{n})}$

   b) Distribute ORs to get ANDs outside of parens

      $(P \vee (Q \wedge R)) \boxed{\Leftrightarrow} ((P \vee Q) \wedge (P \vee R))$ $\boxed{O(\boldsymbol{n})}$

2. Convert to 3CNF by adding new variables

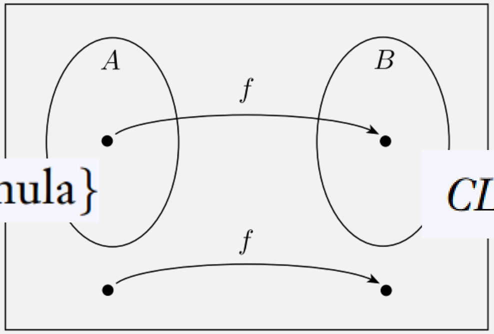
   $(a_1 \vee a_2 \vee a_3 \vee a_4) \boxed{\Leftrightarrow} (a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$ $\boxed{O(\boldsymbol{n})}$

Remaining step: show iff relation holds …

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

Theorem: *3SAT* is polynomial time reducible to *CLIQUE*.

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

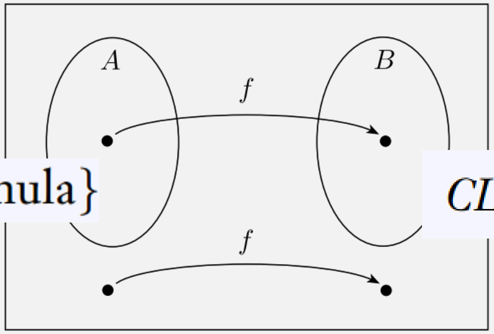


$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

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 **forward direction**)

Theorem: *3SAT* is polynomial time reducible to *CLIQUE*.



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

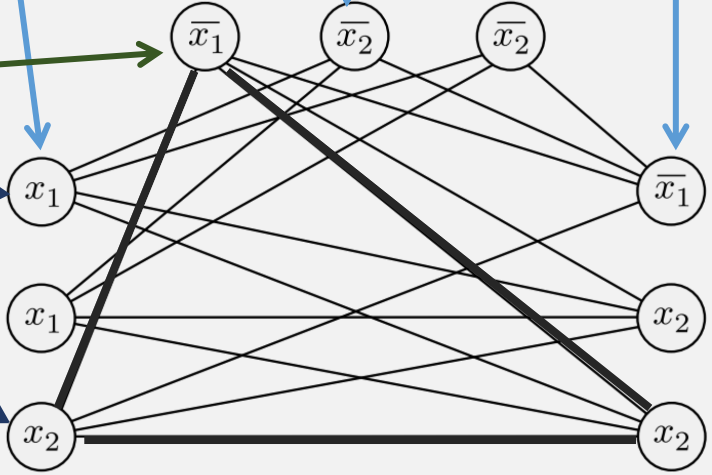
$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

Need: poly time computable fn converting a 3cnf-formula …

Example:

$$\phi = (x_1 \vee x_1 \vee \boxed{x_2}) \wedge (\boxed{\overline{x_1}} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee \boxed{x_2})$$

- … to a graph containing a clique:
  - Each clause maps to a group of 3 nodes
  - Connect all nodes except:
    - Contradictory nodes
    - Nodes in the same group



Runs in **poly time:**
- # literals = # nodes $O(\boldsymbol{n})$
- # edges poly in # nodes $O(\boldsymbol{n^2})$

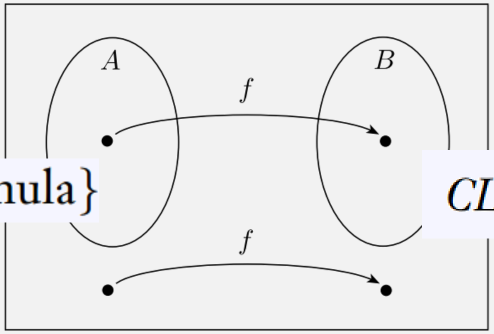
Don't forget iff

$\Rightarrow$ If $\phi \in 3SAT$
- Then each clause has a TRUE literal
  - Those are nodes in the clique!
  - E.g., $x_1 = 0, x_2 = 1$

$\Leftarrow$ If $\phi \notin 3SAT$
- For any assignment, some clause must have a contradiction with another clause
- Then in the graph, some clause's group of nodes won't be connected to another group, preventing the clique

Theorem: *3SAT* is polynomial time reducible to *CLIQUE*.

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



$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

- But this a single language reducing to another single language

NP-Completeness

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.

hard????

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

It's very hard to prove **NP**-Completeness, but only for <u>first</u> problem!

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

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.

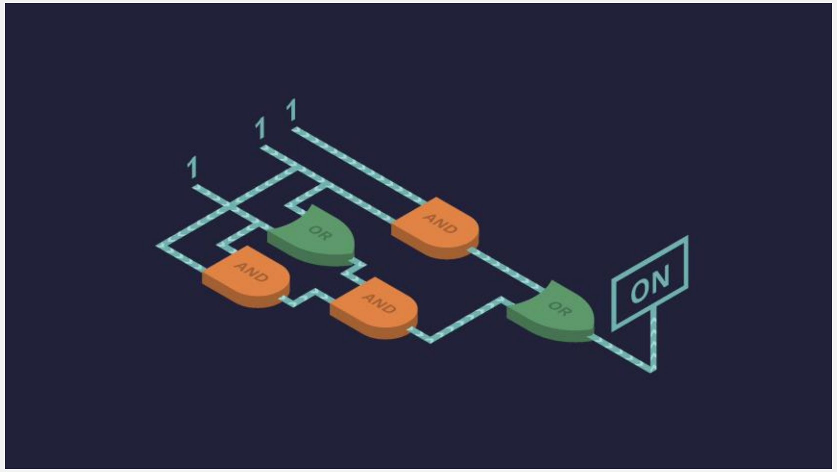
The Cook-Levin Theorem

The first **NP**-Complete problem

THEOREM

SAT is NP-complete.

But 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

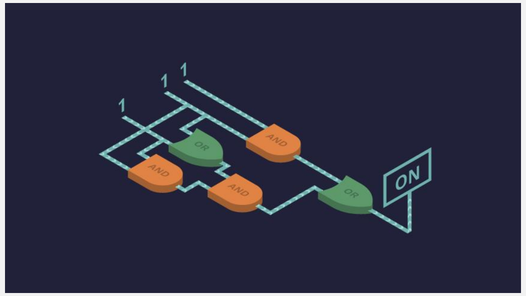
УДК 519.14

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



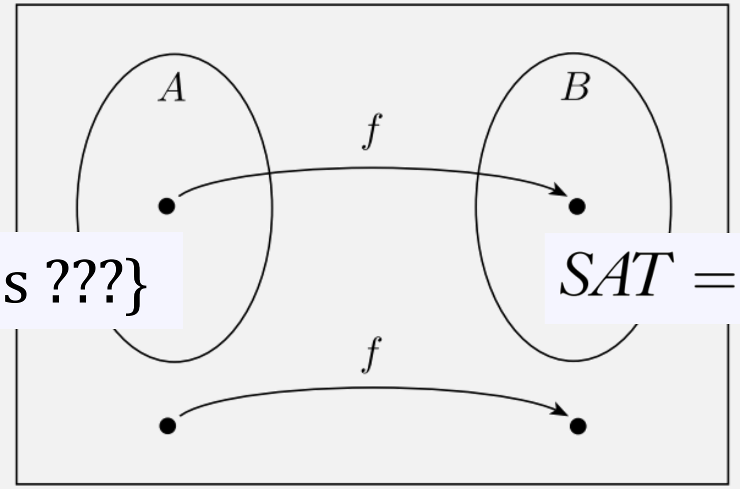
Hard part

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 = $\{w \mid w \text{ is ???}\}$

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

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!

THEOREM

E.g., The class of regular languages is closed under the union operation.

PROOF uses the fact that every regular 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.

Proof constructs a union-recognizing NFA from any two general NFA descriptions

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

THEOREM

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

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

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

They are:

1. Verified by a deterministic poly time verifier

2. Decided by a nondeterministic poly time decider (NTM)

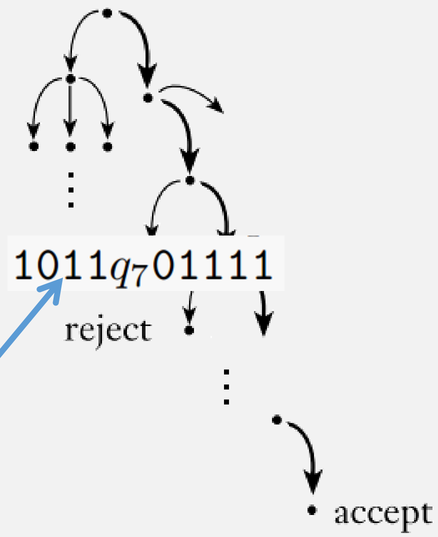
Let's use this one

Flashback: Non-deterministic TMs

- Formally defined with states, transitions, alphabet …

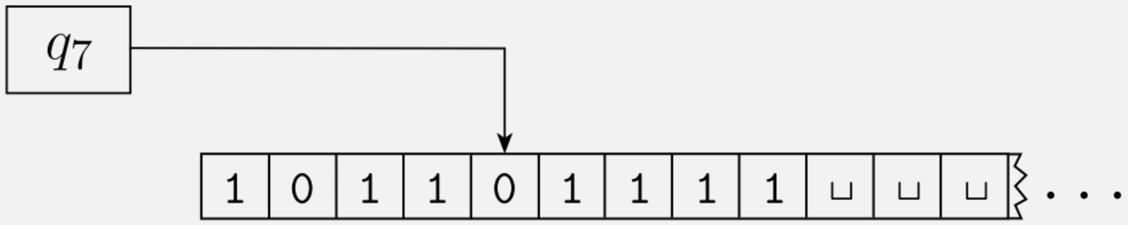
A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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

Flashback: TM Config = State + Head + Tape



$$1011q_7 01111$$

Textual representation of “configuration”

1st char after state is current head position

Flashback: Non-deterministic TMs

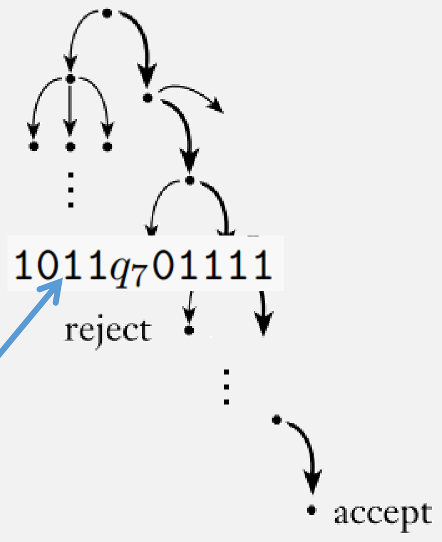
- Formally defined with states, transitions, alphabet …

<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, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

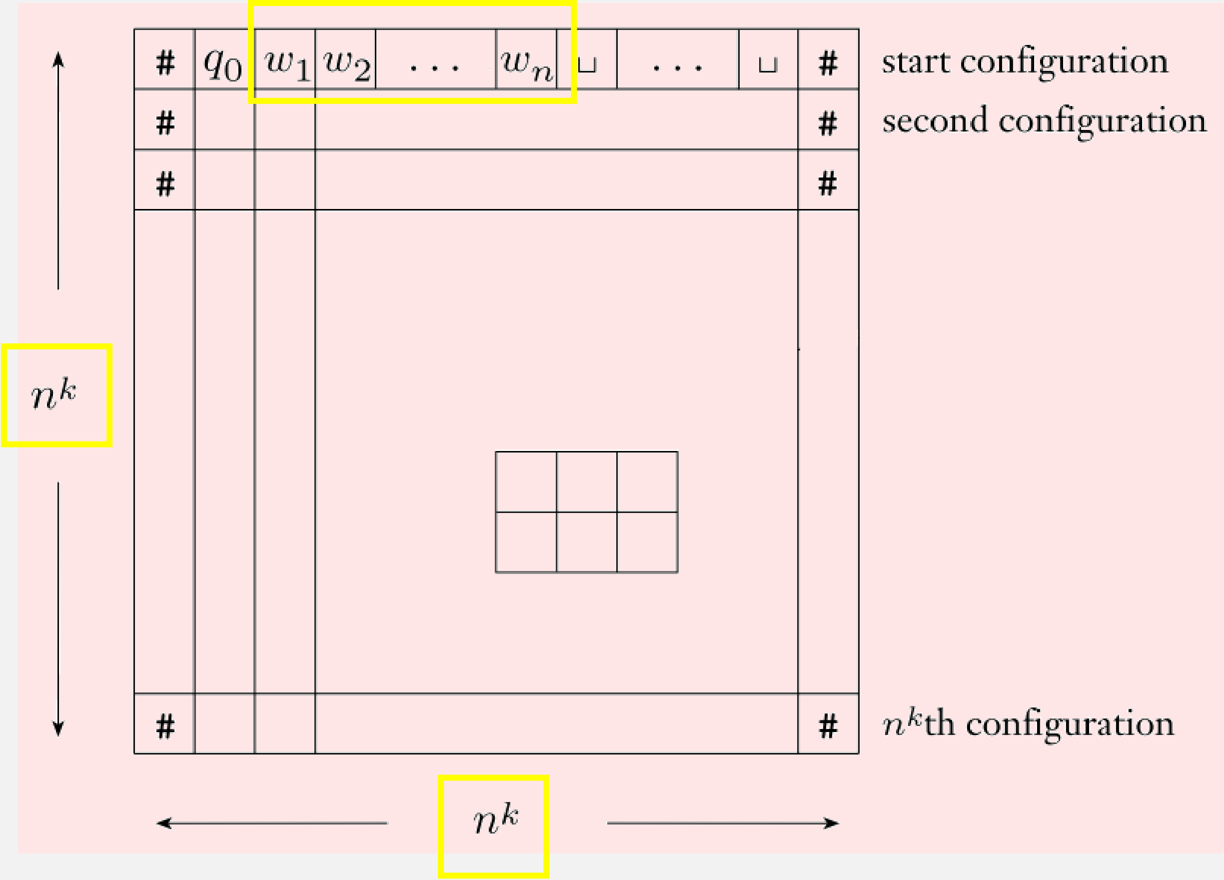
1. Q is the set of states,
2. Σ is the input alphabet not containing the ***blank symbol*** $\sqcup$,
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\boxed{\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\text{L}, \text{R}\})}$ transition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

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

Accepting config sequence = "Tableau"



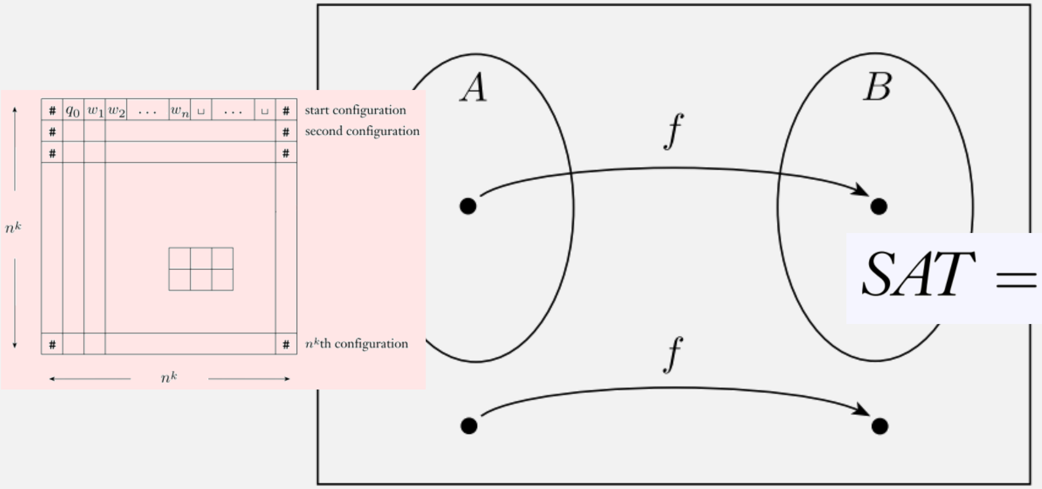
- input $w = w_1 \ldots w_n$
- Assume configs start/end with #
- Must have an accepting config
- At most $\boldsymbol{n^k}$ configs
  - (why?)
- Each config has length $\boldsymbol{n^k}$
  - (why?)

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

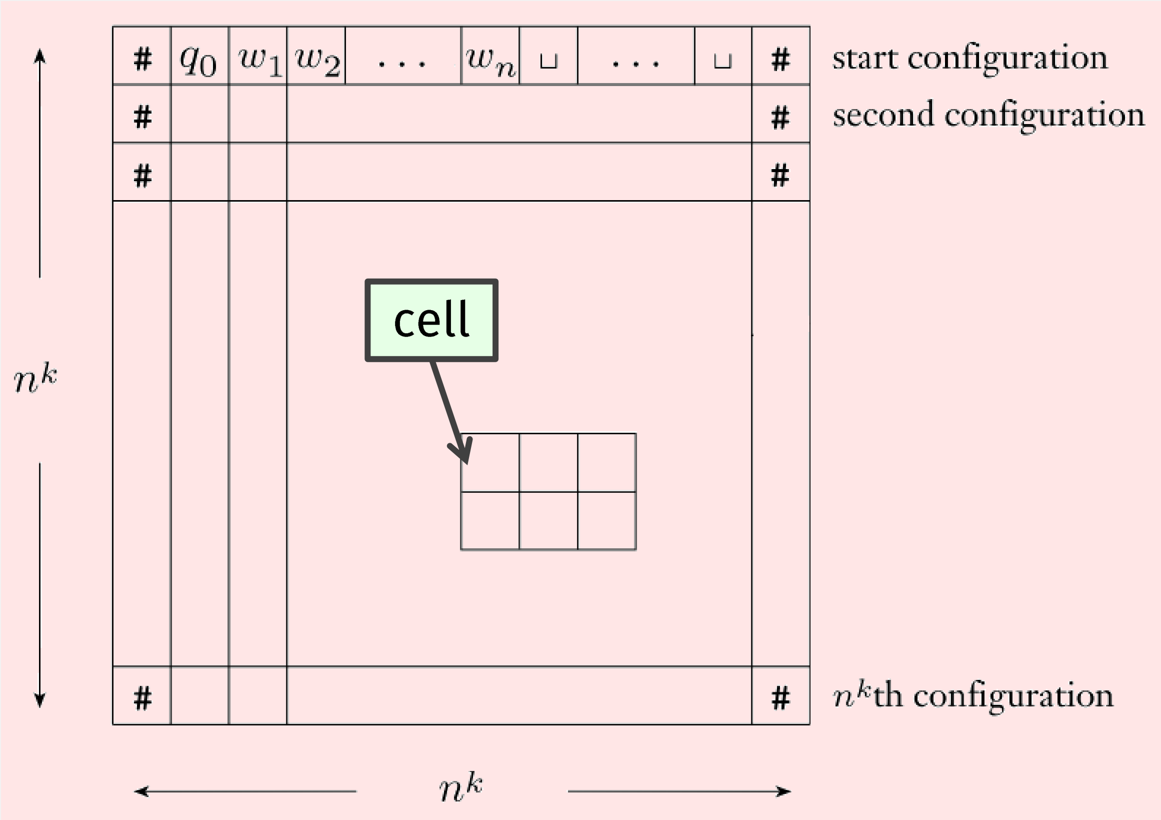


$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$

Tableau Terminology

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

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

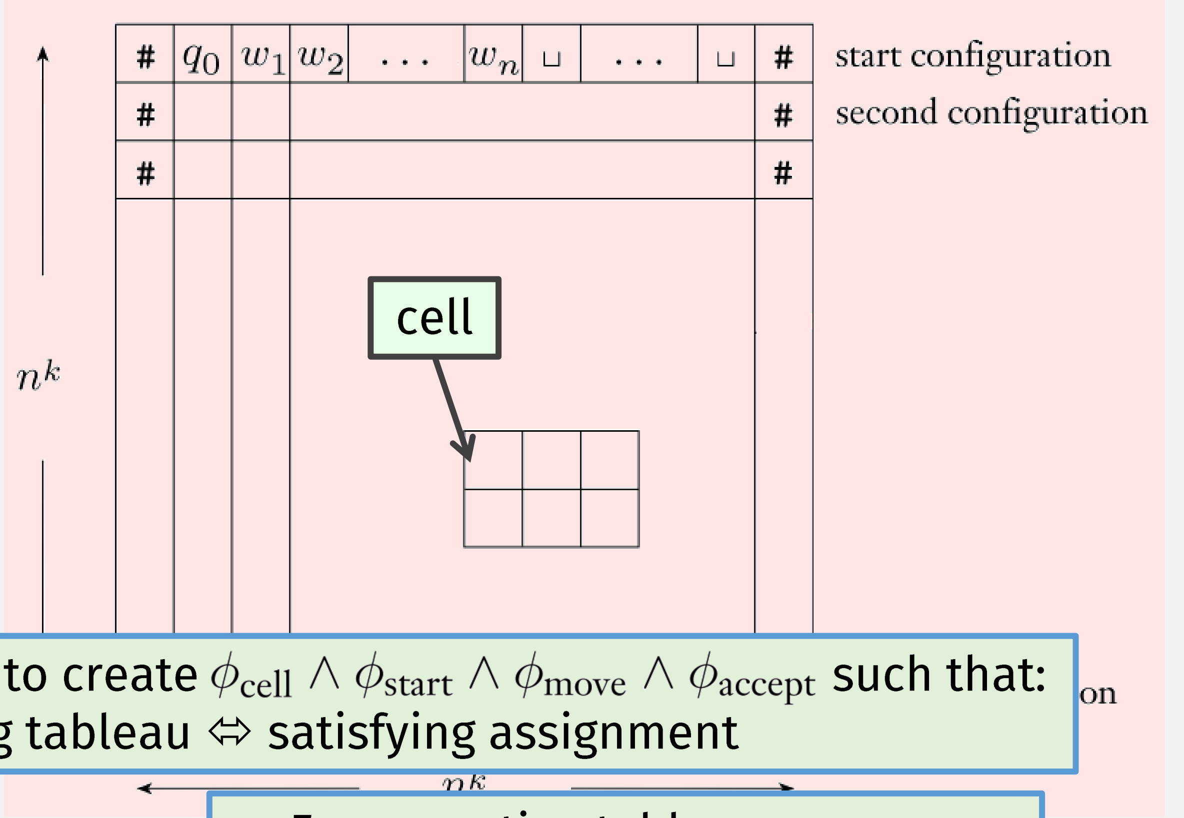


A ***Turing machine*** is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where Q, Σ, Γ are all finite sets and

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. $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$ 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 <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
- Total variables =
  - # cells * # symbols =
  - $n^k * n^k * |C| = O(\mathbf{n^{2k}})$



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

A *Turing m*... where Q, Σ, Γ are a...

1. Q is the...
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. $\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$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 $q_{reject} \neq q_{accept}$.

Check-in Quiz 11/15

On gradescope