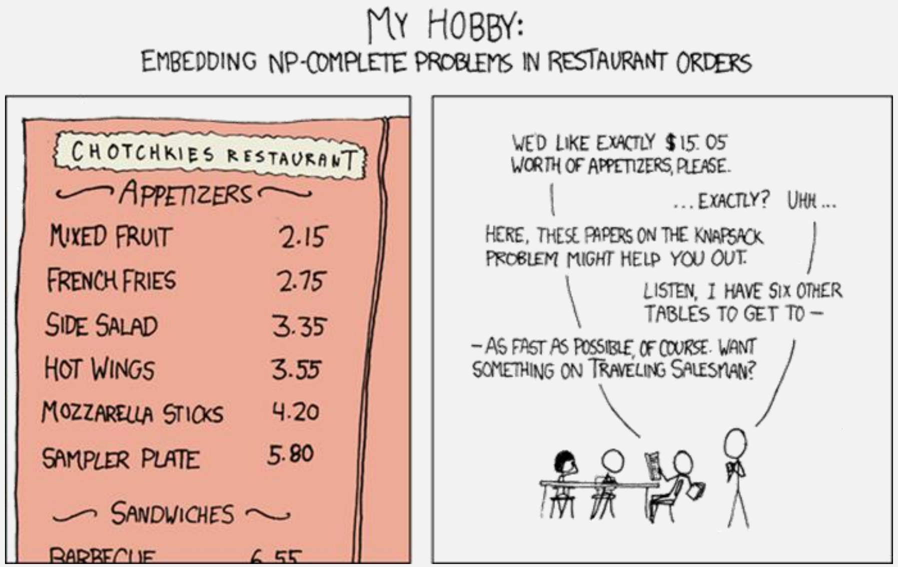
UMB CS 420

NP-Completeness

Wednesday, April 27, 2022



Announcements

- HW 10 in
  - ~~Due Tues 4/26 11:59pm EST~~
- HW 11 out
  - Due Tues 5/3 11:59pm EST
- 5 lectures left!
- No final exam

Last Time: Verifiers, Formally

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

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

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

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.

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

certificate, or ***proof***

- 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.

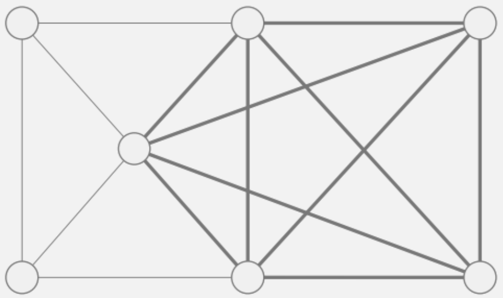
THEOREM

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

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

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



set sum

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

  subset sum

  - Some subset of a set of numbers S must sum to a total t
  - e.g., $\langle\{4, 11, 16, 21, 27\}, 25\rangle \in SUBSET\text{-}SUM$

<u>Theorem</u>: *SUBSET-SUM* is in NP

$SUBSET\text{-}SUM = \{\langle S, t\rangle \mid S = \{x_1, \ldots, x_k\}$, and for some $\{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}$, 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*."

Don't forget to compute run time!
Does this run in poly time?

Proof 2: *SUBSET-SUM* is in NP

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

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

Don't forget to compute run time!
Does this run in poly time?

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 on each possible subset in parallel

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 <u>solved</u> "quickly"

- Want <u>search</u> problems to be in here … but they often are not

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

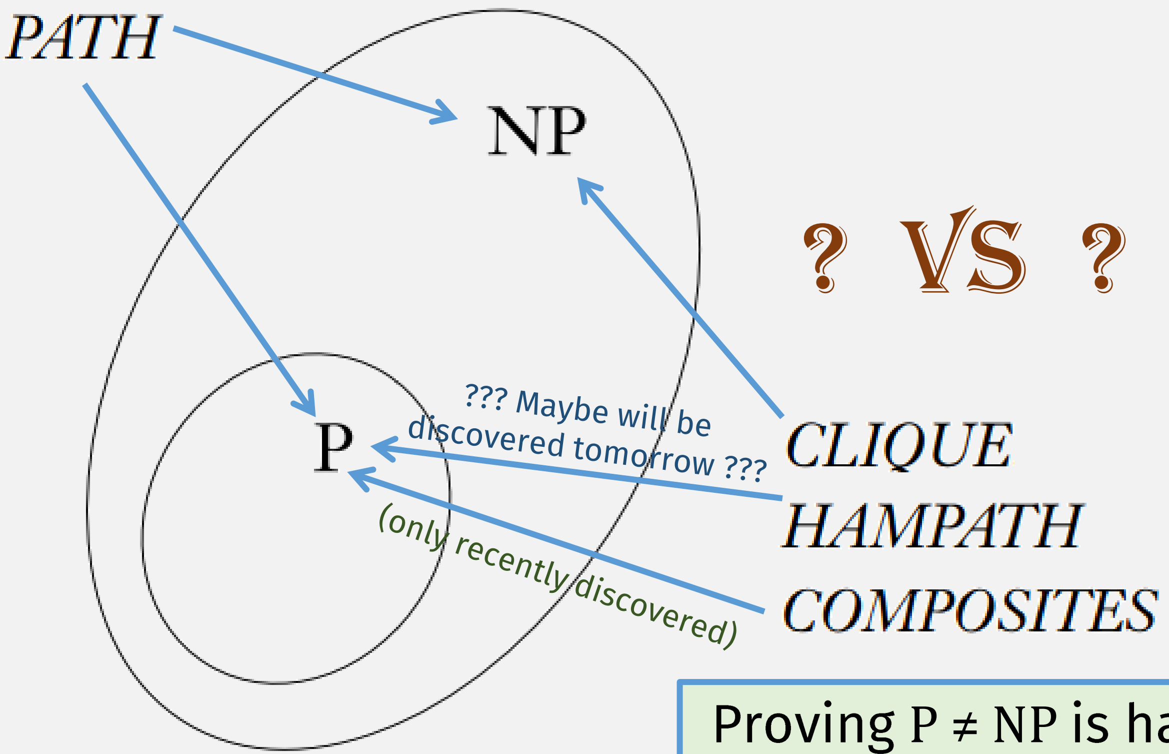
Also, the class of languages that have a **nondeterministic** poly time **decider**

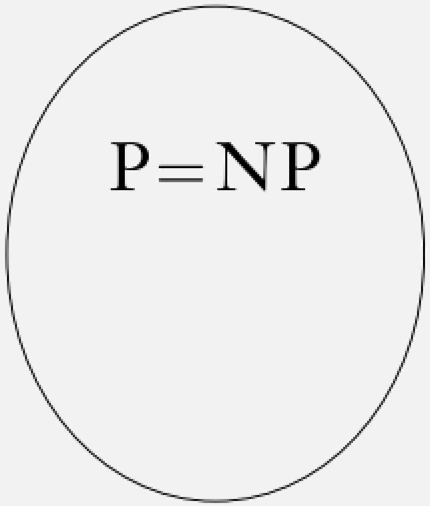
I.e., the class of language that can be <u>verified</u> "quickly"

- Actual <u>search</u> problems (even those not in **P**) are often in here

One of the Greatest unsolved

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



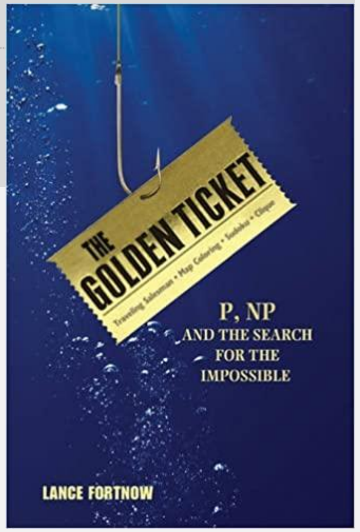


Proving P ≠ NP is hard: how do you prove that an algorithm <u>won't ever</u> have a poly time solution?
(in general, it's hard to prove that something <u>doesn't</u> exist)

Not Much Progress on whether **P = 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:
  - **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 $A \leq_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

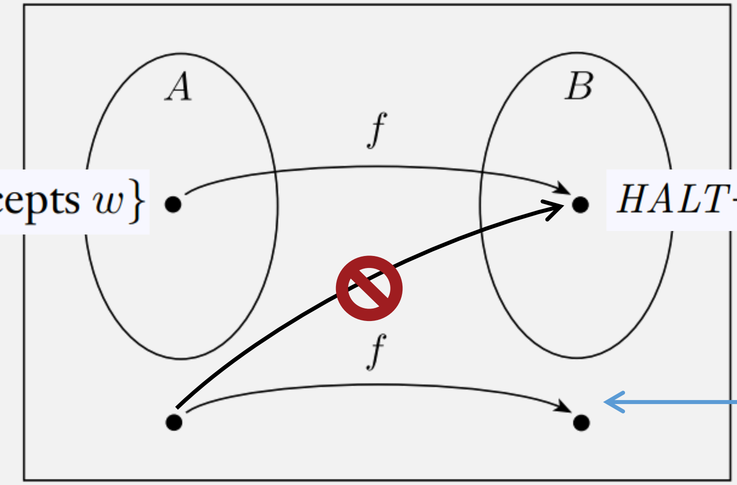
IMPORTANT: "if and only if" …

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 $\overline{A} \leq_{\mathrm{m}} \overline{B}$

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 ... if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, ...

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

The function f is called the ***reduction*** from A t...

To show poly time mapping reducibility:

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

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,

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

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

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

A function $f\colon \Sigma^* \longrightarrow \Sigma^*$ is a *poly time* ***computable function*** if some *poly time* 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

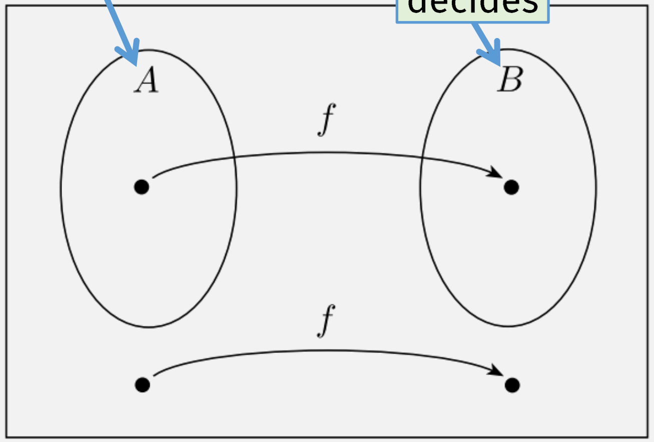
PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

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,

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

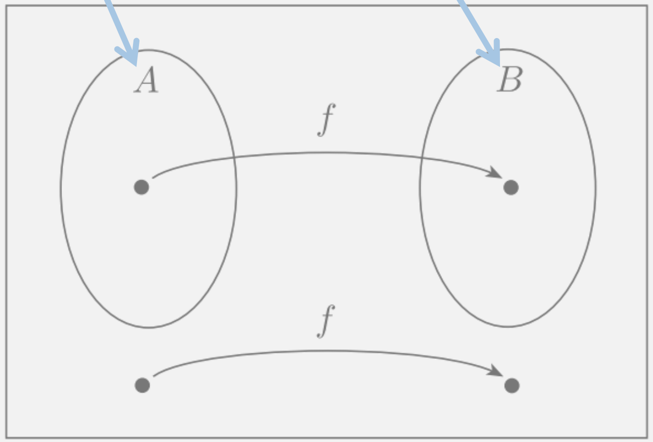
The function f is called the ***reduction*** from A to B.

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

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,

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

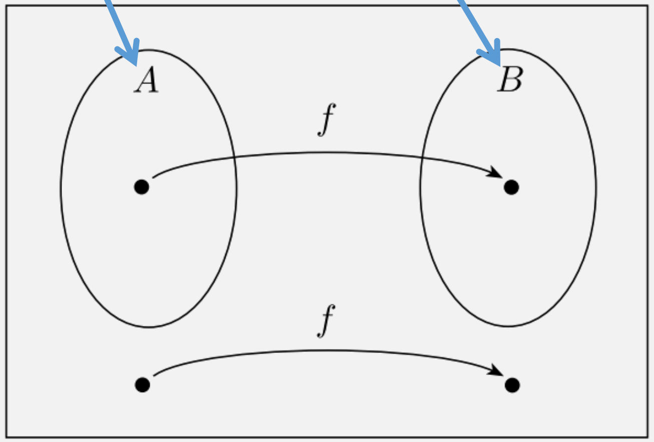
The function f is called the ***reduction*** from A to B.

Thm: If $A \leq_{\text{\sout{m}}\,\mathbf{P}} B$ 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.

THEOREM

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

To prove **P** = **NP**, must show:

1. every language in **P** is in **NP**
   - Trivially true (why?)
2. every language in **NP** is in **P**
   - Given a language $A \in$ **NP** …
   - … can poly time mapping reduce A to B
     - because B is **NP**-Complete
   - Then A also $\in$ **P** …
     - Because $A \leq_{\mathrm{P}} B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$

DEFINITION

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

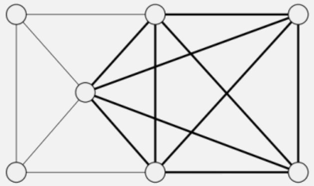
1. B is in NP, and
2. every A in NP is polynomial time reducible to B.

Next: How to do poly time mapping reducibility

Thus, if a language B is **NP**-complete and in **P**, then **P** = **NP**

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

Last Time: # *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 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 **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

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

Theorem: SAT is in **NP**:

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

Verifier:

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

- Plug values from c into ϕ, Accept if result is TRUE

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

Non-deterministic Decider:

On input <ϕ>, 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**	$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

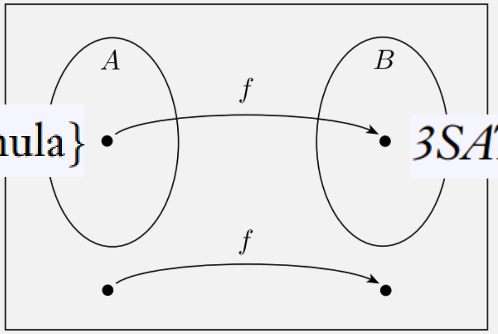
$\wedge$ = AND = “Conjunction”
$\vee$ = OR = “Disjunction”
$\neg$ = NOT = “Negation”

The *3SAT* Problem

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

<u>Theorem</u>: *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.,

    ⇒ if $\phi \in SAT$, then $f(\phi) \in 3SAT$

4. and **reverse direction**

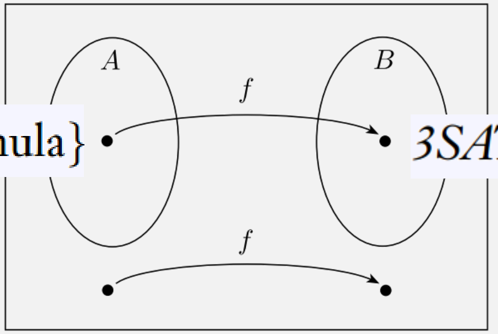
    ⇐ if $f(\phi) \in 3SAT$, then $\phi \in SAT$

    (or **contrapositive** of **forward direction**)

    ⇐ (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

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

   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

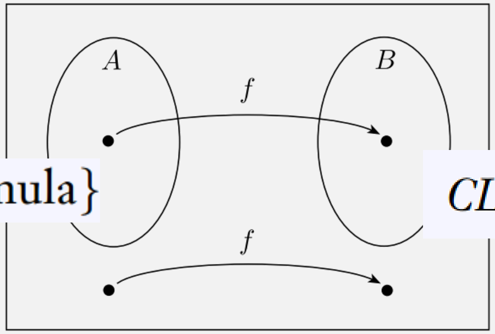
   $(a_1 \vee a_2 \vee a_3 \vee a_4) \Leftrightarrow (a_1 \vee a_2 \vee z) \wedge (\overline{z} \vee a_3 \vee a_4)$ $\quad$ $O(n)$

Remaining step: show iff relation holds ...

... this thm is special, don't need to separate forward/reverse dir for this thm: bc each step is already a known "law"

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

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

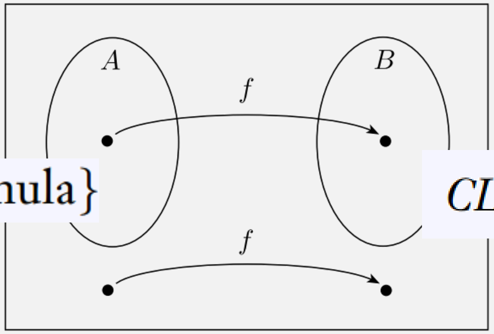


$CLIQUE = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

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*.

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



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

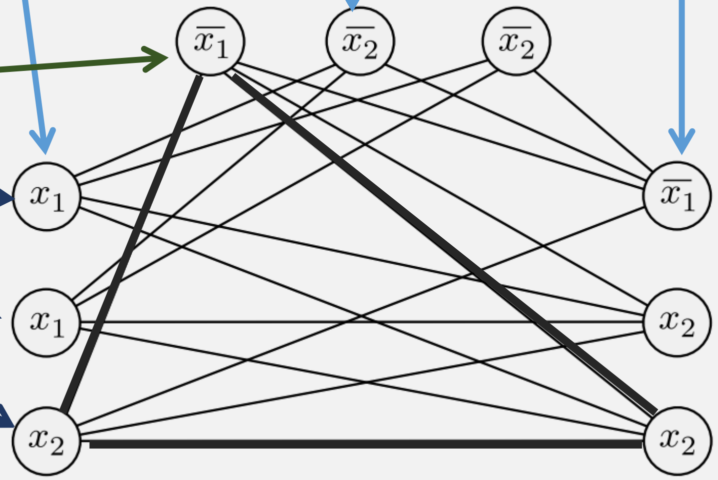
Need: poly time computable fn converting a 3cnf-formula …

Example:

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

- … 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

Don't forget iff



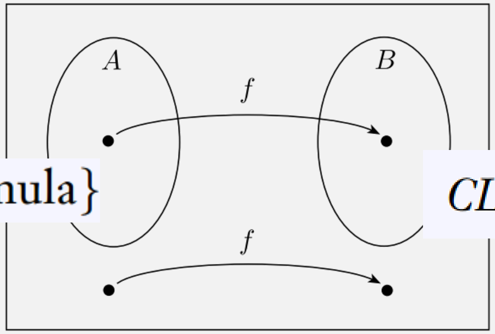
Runs in **poly time:**
- \# literals = # nodes — $O(n)$
- \# edges poly in # nodes — $O(n^2)$

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

⇐ If $\phi \notin 3SAT$
- Then 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

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.

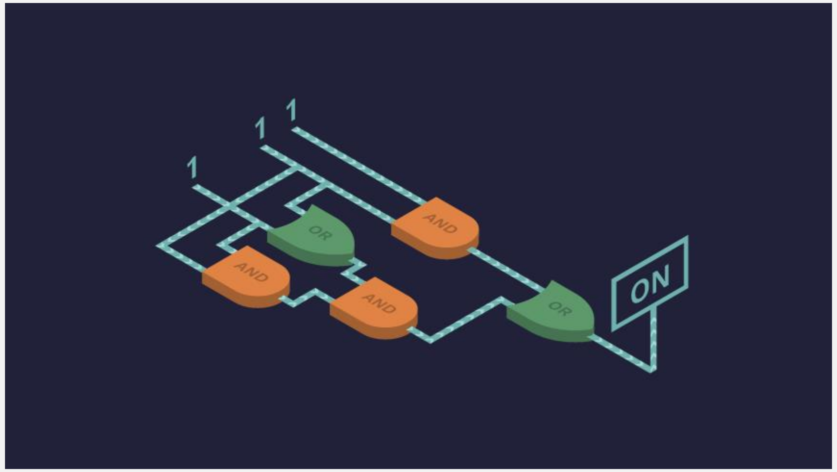
Next Time: The Cook-Levin Theorem

The first **NP**-Complete problem

THEOREM

SAT is NP-complete.

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



Check-in Quiz 4/27

On gradescope