

# UMB CS 420

# NP-Completeness

Wednesday, April 27, 2022

MY HOBBY:  
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS

CHOTCHKIES RESTAURANT	
~~ APPETIZERS ~~	
MIXED FRUIT	2.15
FRENCH FRIES	2.75
SIDE SALAD	3.35
HOT WINGS	3.55
MOZZARELLA STICKS	4.20
SAMPLER PLATE	5.80
~~ SANDWICHES ~~	
BARBECUE	6.55



## *Announcements*

- HW 10 in
  - Due Tues 4/26 11:59pm EST
- HW 11 out
  - Due Tues 5/3 11:59pm EST

# Last Time: Verifiers, Formally

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

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

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

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

**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  $n = \text{length of } w$

# 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

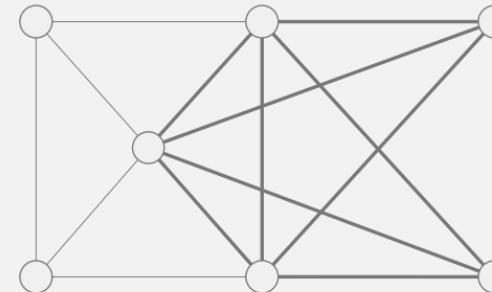
## THEOREM

---

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



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

# Theorem: *SUBSET-SUM* is in NP

*SUBSET-SUM* = { $\langle S, t \rangle$  |  $S = \{x_1, \dots, x_k\}$ , and for some  $\{y_1, \dots, y_l\} \subseteq \{x_1, \dots, x_k\}$ , we have  $\sum y_i = t$ }

**PROOF IDEA** The subset is the certificate.

To prove a lang is in NP, create either:

- 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

*SUBSET-SUM* = { $\langle S, t \rangle$  |  $S = \{x_1, \dots, x_k\}$ , and for some  $\{y_1, \dots, y_l\} \subseteq \{x_1, \dots, x_k\}$ , we have  $\sum y_i = t$ }

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.

Nondeterministically runs  
the verifier on each  
possible subset in parallel

$N$  = “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*.”

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”

- Want search problems to be in here ... but they often are not

**NP**

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

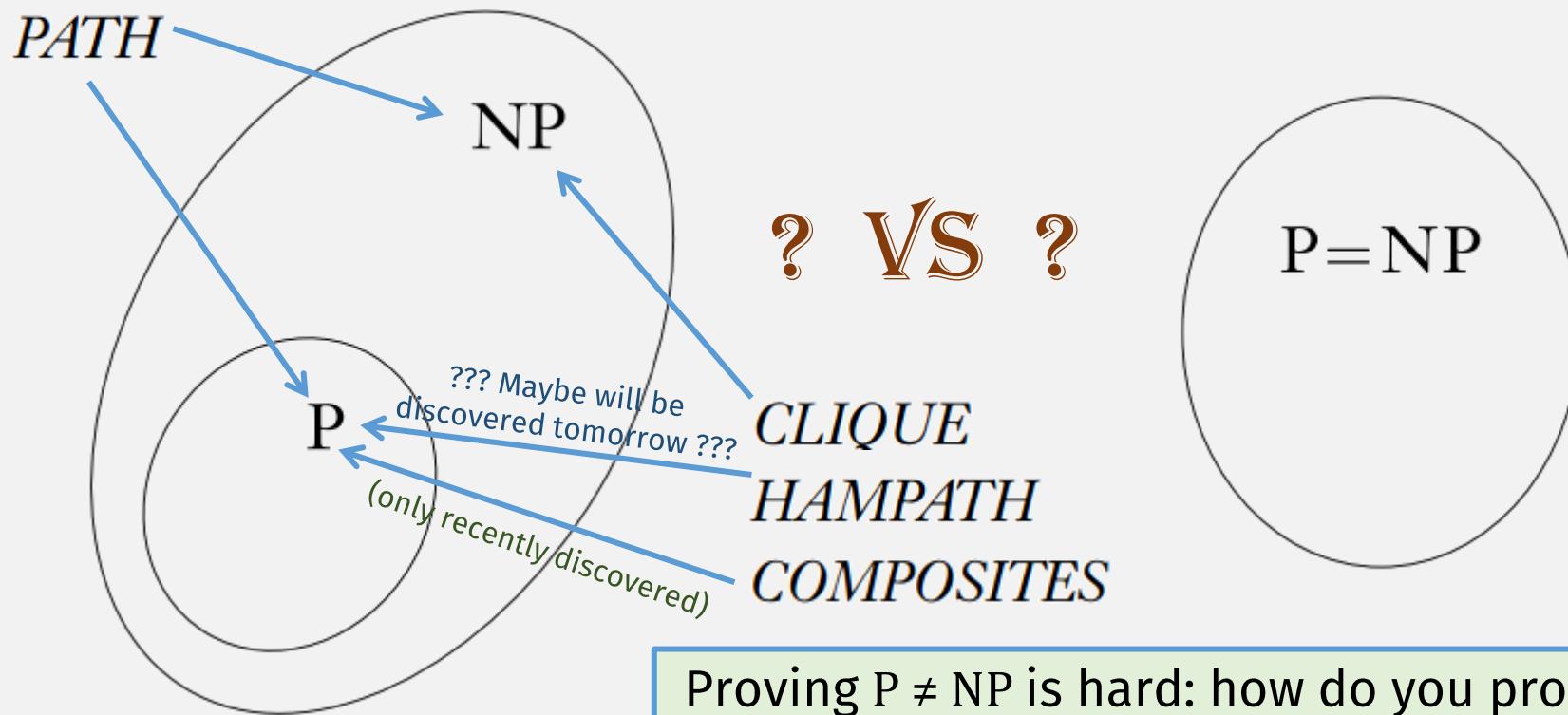
Also, the class of languages that have a **nondeterministic** poly time **decider**

i.e., the class of language that can be verified “quickly”

- Actual search problems (even those not in **P**) are often in here

**One of the Greatest unsolved**

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



Proving  $P \neq NP$  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)

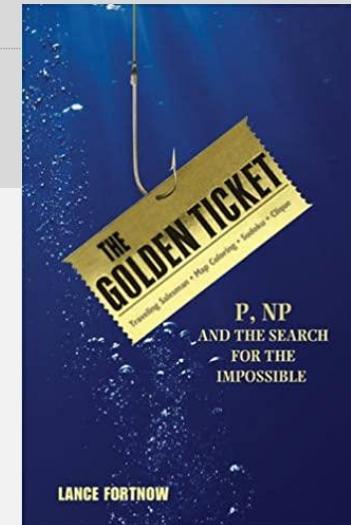
# Not Much Progress on whether $P = NP$ ?

## 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:

Must prove for all  
langs, not just a  
single language

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

- How does this help the  $P = NP$  problem? **What's this?**

## THEOREM

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

# Flashback: Mapping Reducibility

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

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

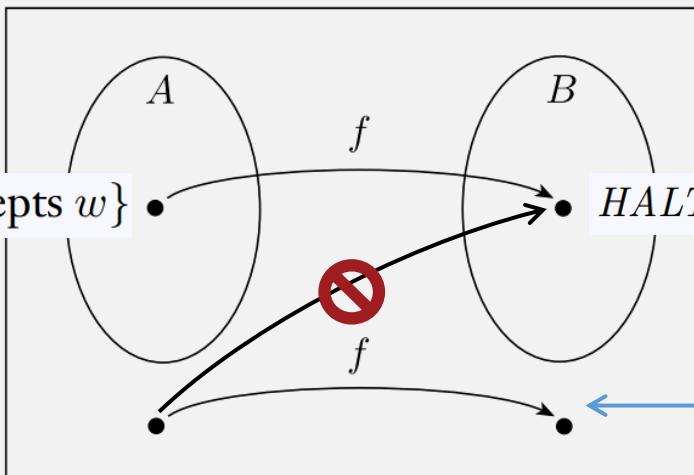
**IMPORTANT:** “if and only if” ...

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**)

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



... means  $\overline{A} \leq_m \overline{B}$

A function  $f: \Sigma^* \rightarrow \Sigma^*$  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  $B$  if there is a computable function  $f: \Sigma^* \rightarrow \Sigma^*$ ,

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

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

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  $A \leq_P B$ , if a polynomial time computable function  $f: \Sigma^* \rightarrow \Sigma^*$  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  $f: \Sigma^* \rightarrow \Sigma^*$  is a **computable function** if some Turing machine  $M$ , on every input  $w$ , halts with just  $f(w)$  on its tape.

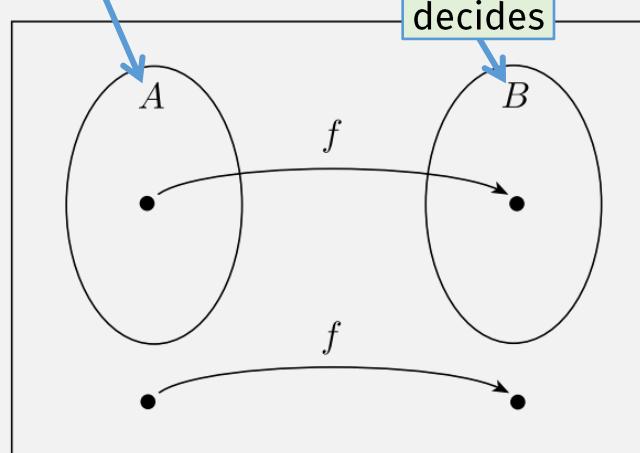
Flashback: If  $A \leq_m B$  and  $B$  is decidable, then  $A$  is decidable.

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.”



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  $f: \Sigma^* \rightarrow \Sigma^*$ , where for every  $w$ ,

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

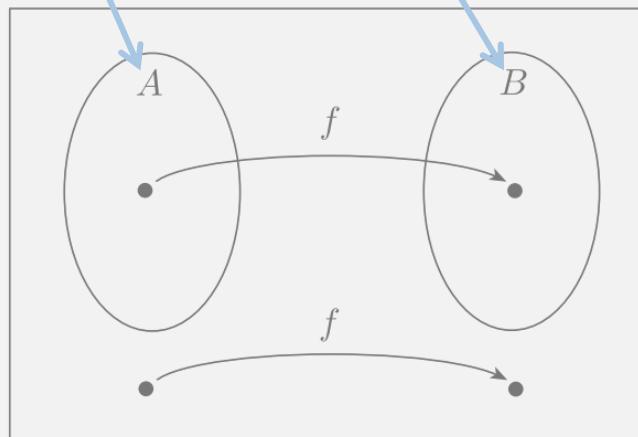
The function  $f$  is called the **reduction** from  $A$  to  $B$ .

Thm: If  $A \leq_m^P B$  and  $B$  is decidable, then  $A \in P$ .

**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  $A \leq_m B$ , if there is a computable function  $f: \Sigma^* \rightarrow \Sigma^*$ , where for every  $w$ ,

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

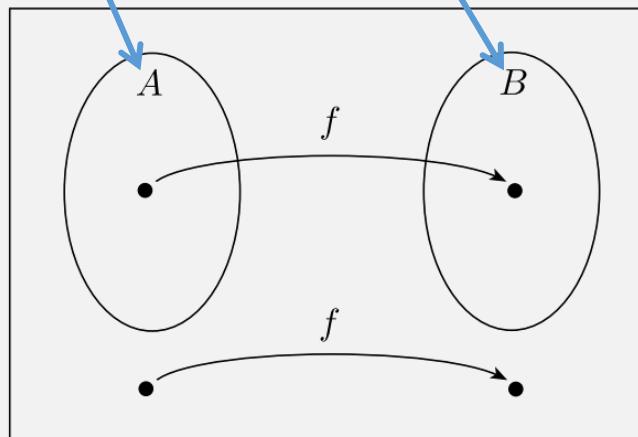
The function  $f$  is called the *reduction* from  $A$  to  $B$ .

Thm: If  $A \leq_m^P B$  and  $B$  is decidable, then  $A \in P$ .

**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.”



**poly time**  
Language  $A$  is *mapping reducible* to language  $B$ , written  $A \leq_m B$ , if there is a computable function  $f: \Sigma^* \rightarrow \Sigma^*$ , 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  $B \in P$ , then  $P = NP$ .

To prove  $P = NP$ , must show:

1. every language in  $P$  is in  $NP$

- Trivially true

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  $A \leq_P B$  and  $B \in P$ , then  $A \in P$

## 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  $B$ .

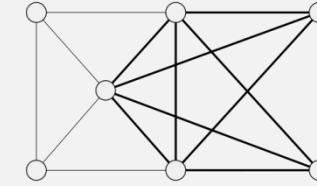
Next: How to do poly time mapping reducibility

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



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

**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:
<b>Value</b>	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
<b>Variable</b>	Represents a Boolean <b>value</b>	x, y, z

# Boolean Formulas

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

# Boolean Formulas

A Boolean _____	Is ...	Example:
<b>Value</b>	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
<b>Variable</b>	Represents a Boolean <b>value</b>	x, y, z
<b>Operation</b>	Combines Boolean <b>variables</b>	AND, OR, NOT ( $\wedge$ , $\vee$ , and $\neg$ )
<b>Formula <math>\phi</math></b>	Combines <b>vars</b> and <b>operations</b>	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$

# 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  $(\bar{x} \wedge y) \vee (x \wedge \bar{z})$  satisfiable?
  - Yes
  - $x = \text{FALSE}$ ,
  - $y = \text{TRUE}$ ,
  - $z = \text{FALSE}$

# The Boolean Satisfiability Problem

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

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

- Let  $n$  = the number of variables in the formula

Verifier:

On input  $\langle \phi, c \rangle$ , where  $c$  is a possible assignment of variables in  $\phi$  to values:

- Accept if  $c$  satisfies  $\phi$

Running Time:  $O(n)$

| Non-deterministic Decider:

| On input  $\langle \phi \rangle$ , where  $\phi$  is a boolean formula:

- Non-deterministically try all possible assignments in parallel
- Accept if any satisfy  $\phi$

| Running Time: Checking each assignment takes time  $O(n)$

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 $\phi$	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$

# 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 $\phi$	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	$x$ or $\bar{x}$

# 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 $\phi$	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	$x$ or $\bar{x}$
Clause	Literals ORed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4)$

# 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 $\phi$	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	$x$ or $\bar{x}$ .
Clause	Literals ORed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4)$
Conjunctive Normal Form (CNF)	Clauses ANDed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_5 \vee x_6)$

$\wedge$  = AND = “Conjunction”  
 $\vee$  = OR = “Disjunction”  
 $\neg$  = 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 $\phi$	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	$x$ or $\bar{x}$ .
Clause	Literals ORed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4)$
Conjunctive Normal Form (CNF)	Clauses ANDed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_5 \vee x_6)$
3CNF Formula	Three literals in each clause	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_5 \vee x_6) \wedge (x_3 \vee \bar{x}_6 \vee x_4)$

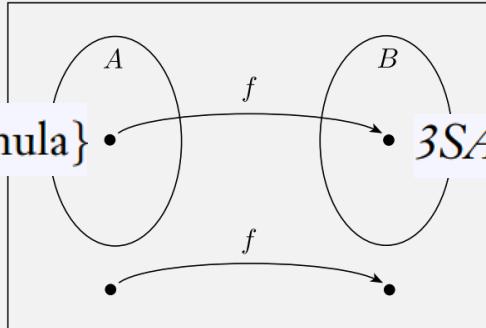
$\wedge$  = AND = “Conjunction”  
 $\vee$  = OR = “Disjunction”  
 $\neg$  = NOT = “Negation”

# The $3SAT$ Problem

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

# Theorem: $SAT$ is Poly Time Reducible to $3SAT$

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



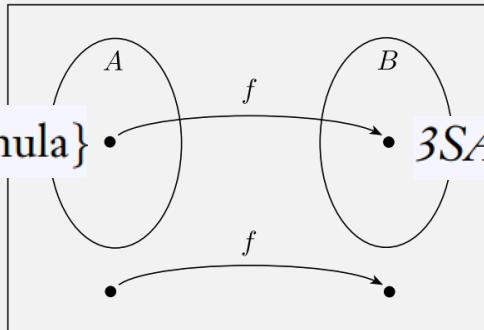
$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

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 $3SAT$

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



Need: poly time computable fn converting a Boolean formula  $\phi$  to 3CNF:

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

$$\neg(P \vee Q) \Leftrightarrow (\neg P) \wedge (\neg Q) \quad \neg(P \wedge Q) \Leftrightarrow (\neg P) \vee (\neg Q)$$

Remaining step: show  
iff relation holds ...

$O(n)$

- b) Distribute ORs to get ANDs outside of parens

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

$O(n)$

... easy for formula conversion: each step is already a known “law”

2. Convert to 3CNF by adding new variables

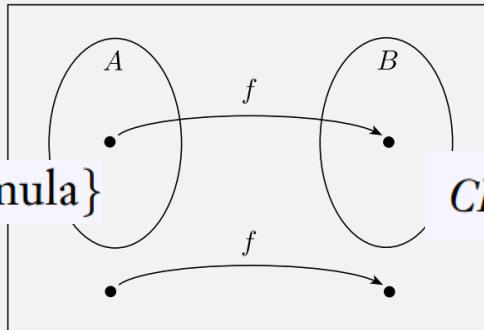
$$(a_1 \vee a_2 \vee a_3 \vee a_4) \Leftrightarrow (a_1 \vee a_2 \vee z) \wedge (\bar{z} \vee a_3 \vee a_4)$$

$O(n)$

# Theorem: $3SAT$ is polynomial time reducible to $CLIQUE$ .

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

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



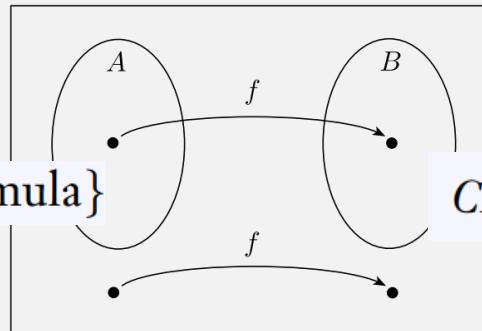
To show poly time mapping reducibility:

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:

$$\phi = (x_1 \vee x_1 \vee \boxed{x_2}) \wedge (\boxed{\bar{x}_1} \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee \boxed{x_2})$$

- ... to a graph containing a clique:

- Each clause maps to a group of 3 nodes
- Connect all nodes except:
- Contradictory nodes

Don't forget iff

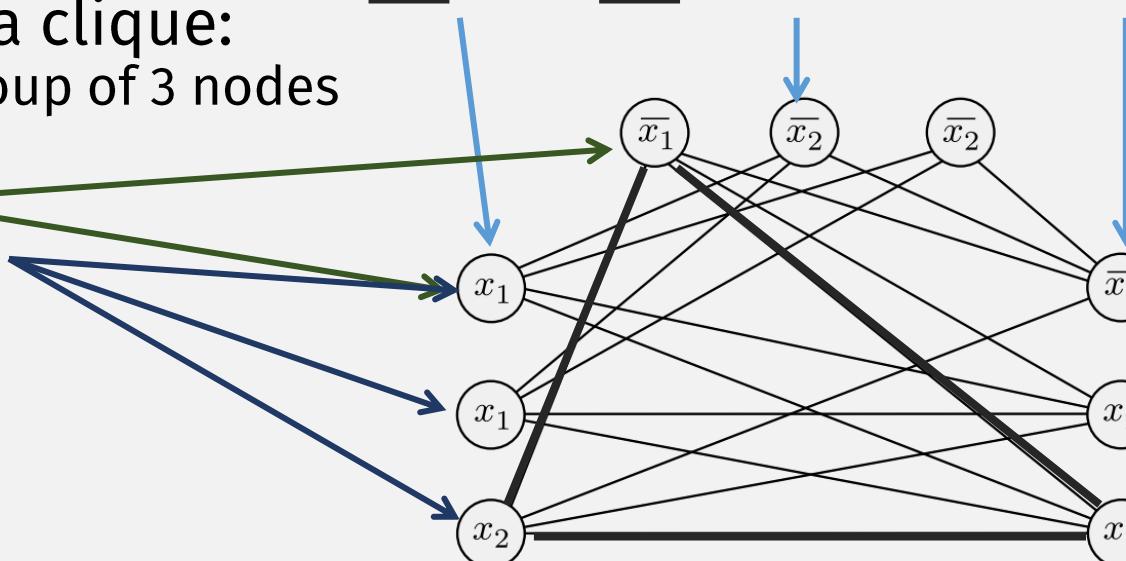
Nodes in the same group

$\Rightarrow$  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



Runs in **poly time**:

- # literals = # nodes
- # edges poly in # nodes

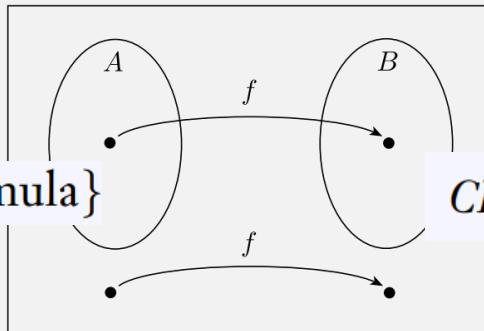
$O(n)$

$O(n^2)$

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:

Must prove for all  
langs, not just a  
single language

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

It's very hard to prove NP-Completeness,  
but only for first problem!

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

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

## THEOREM

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

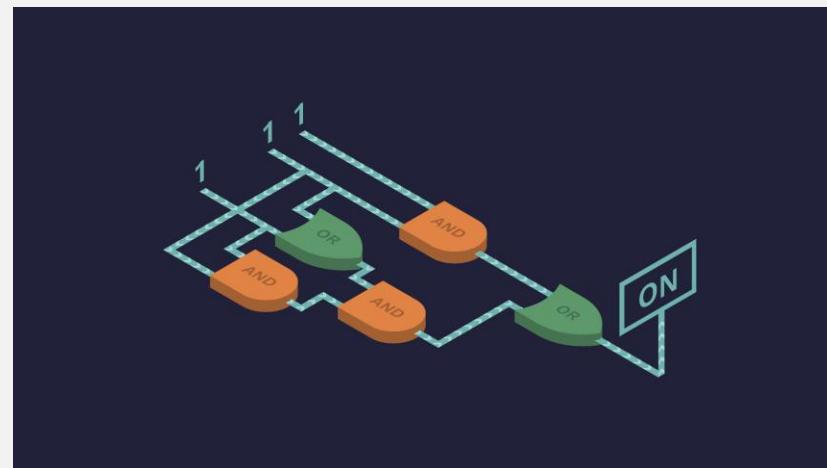
# *Next Time:* The Cook-Levin Theorem

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