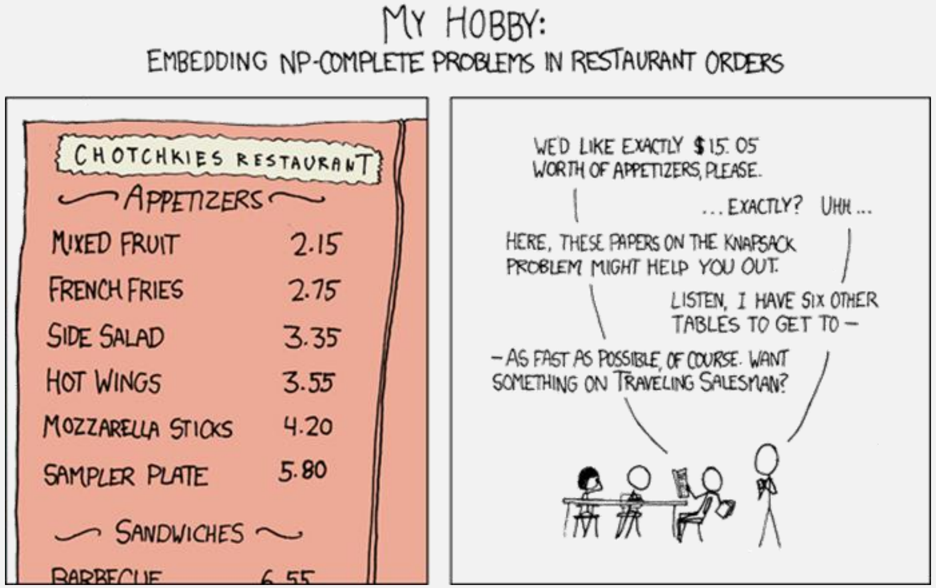
UMB CS 420

NP-Completeness

Monday, May 8, 2023

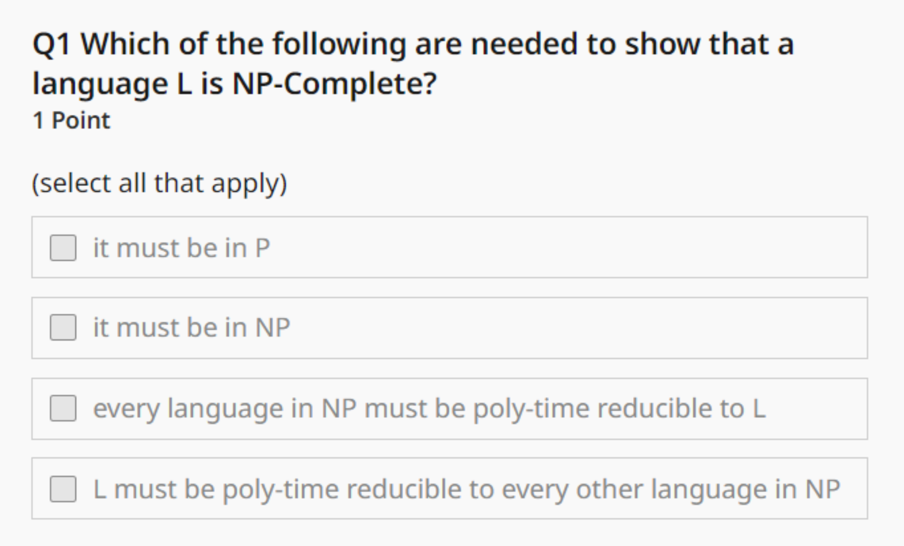


Announcements

- **HW 12 out** (last hw)
  - Due Sunday 5/14 11:59pm

- **Fill out course evaluations!** (sent in email)

Quiz Preview



Q1 Which of the following are needed to show that a language L is NP-Complete?
1 Point

(select all that apply)

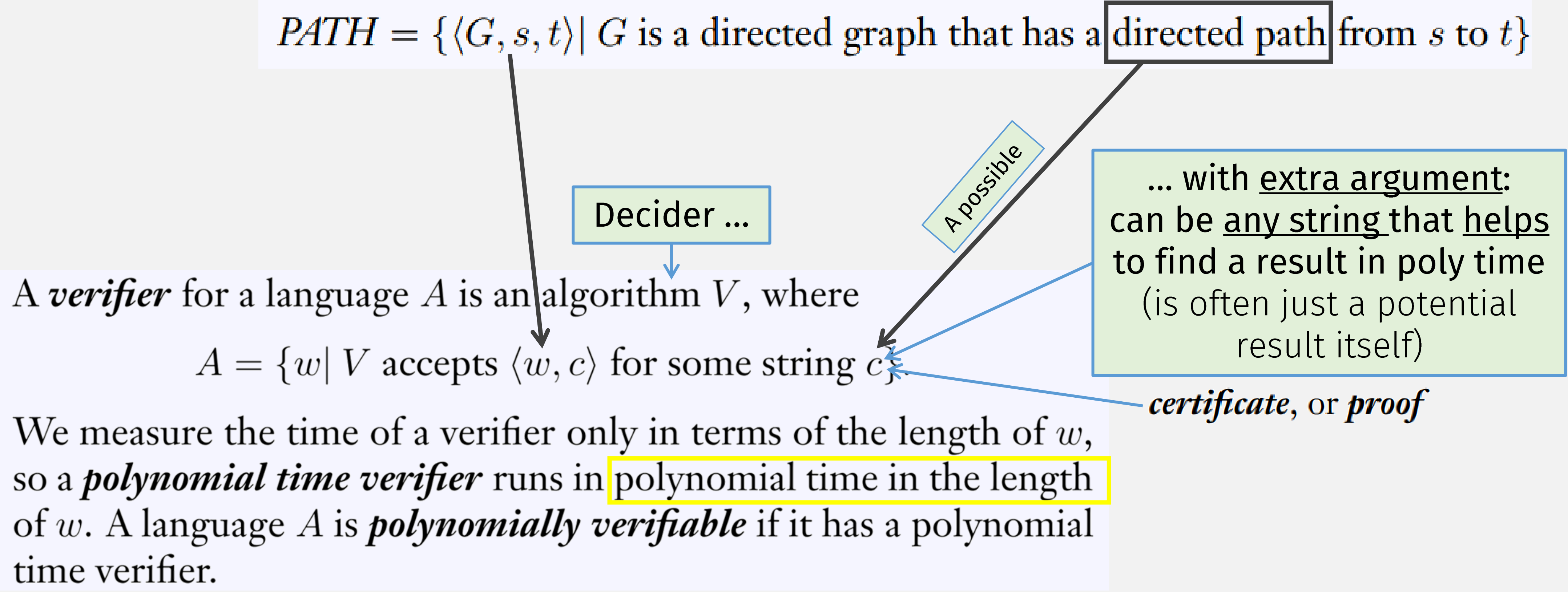
☐ it must be in P

☐ it must be in NP

☐ every language in NP must be poly-time reducible to L

☐ L must be poly-time reducible to every other language in NP

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 possible

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

Decider …

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

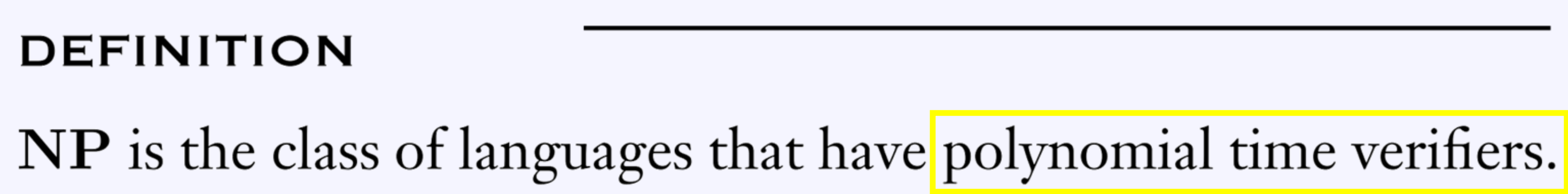
$$A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}.$$

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.

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

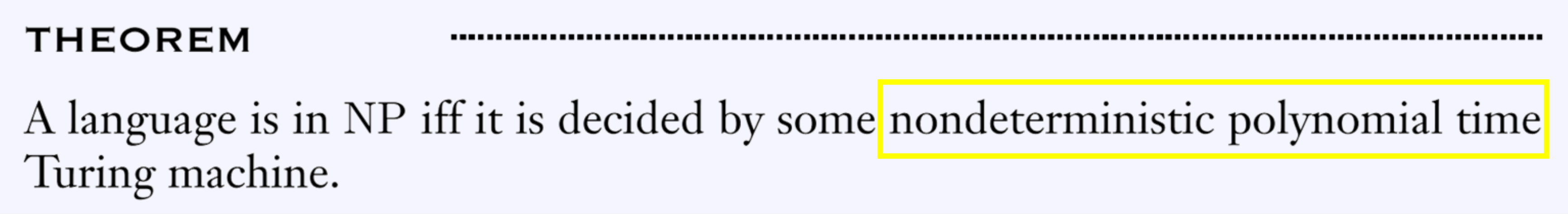
Last Time: The class **NP**



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** 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**

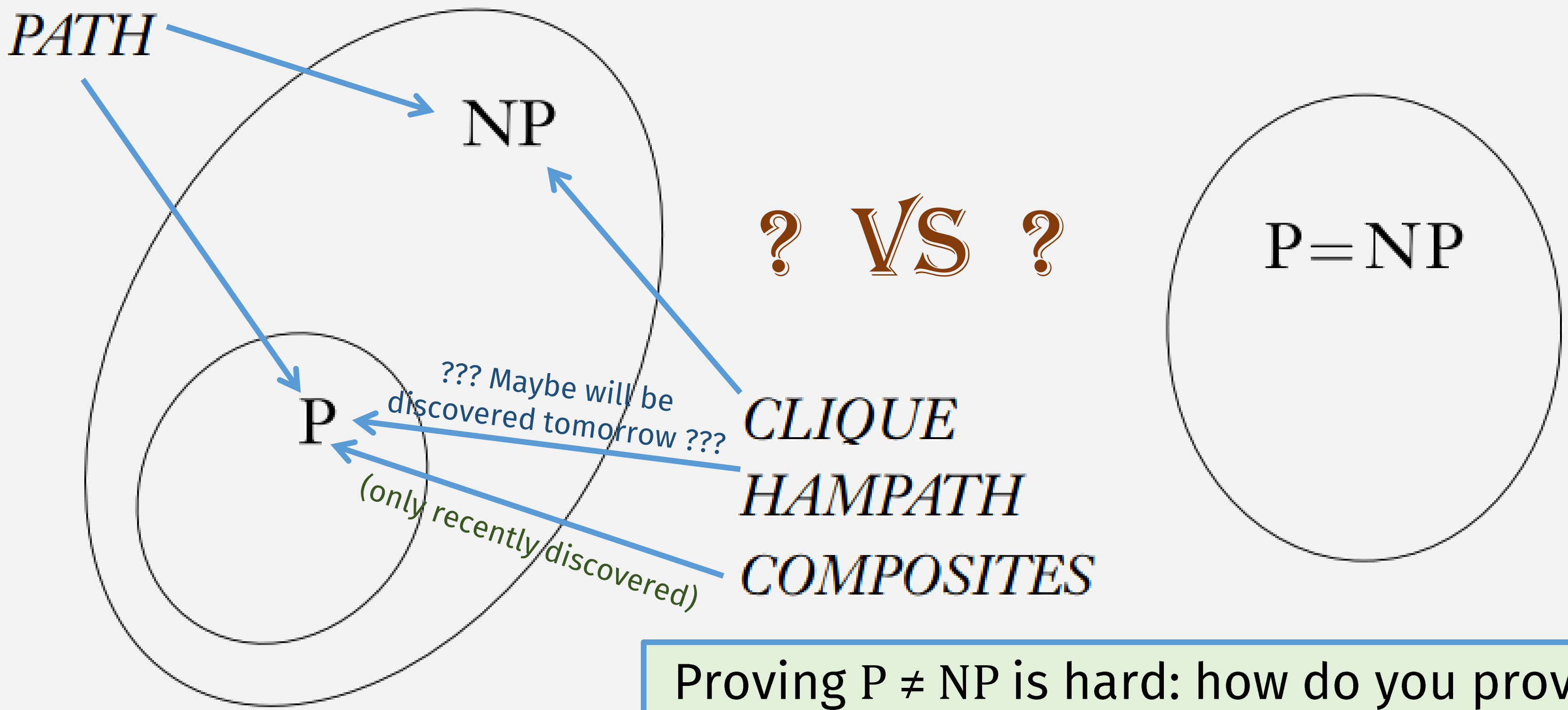
Also, the class of languages that have a **nondeterministic** poly time **decider**

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?



PATH

NP

P

??? Maybe will be discovered tomorrow ???

(only recently discovered)

CLIQUE

HAMPATH

COMPOSITES

? VS ?

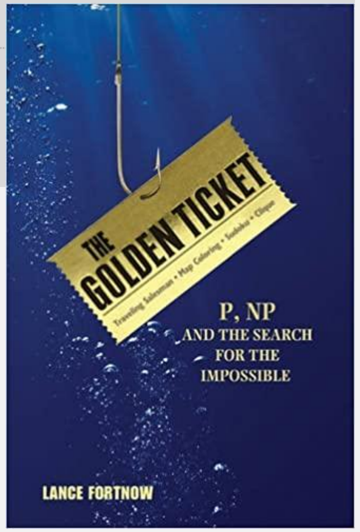
P = NP

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 **P = NP** ?

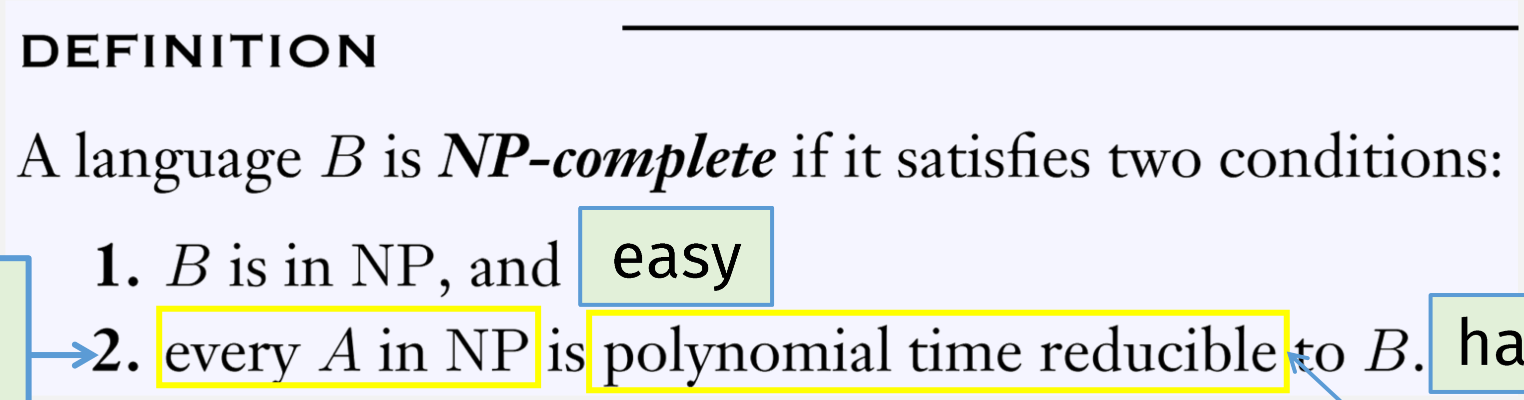
The Status of the P Versus NP Problem

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 all langs, not just a single language

What's this?

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,

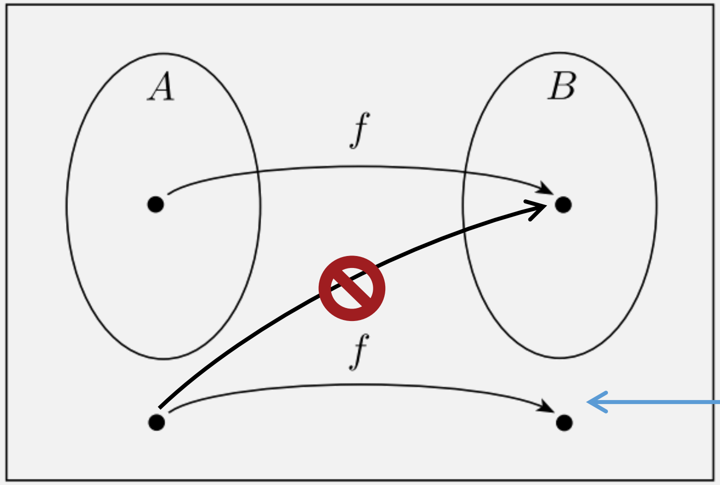
$$w \in A \iff f(w) \in B.$$

IMPORTANT: “if and only if” …

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

To show mapping reducibility:

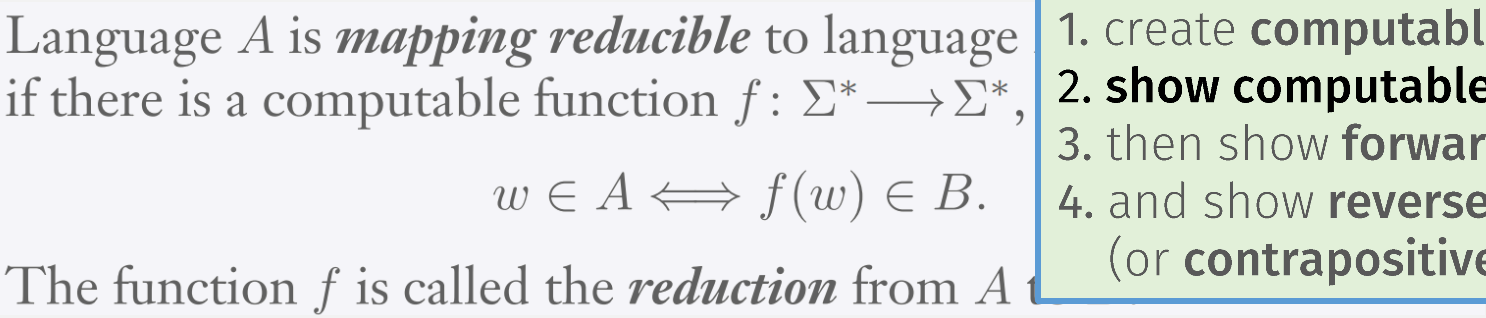
1. create **computable fn**
2. and then show **forward direction**
3. and **reverse direction**
   (or **contrapositive** of **reverse direction**)



… 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



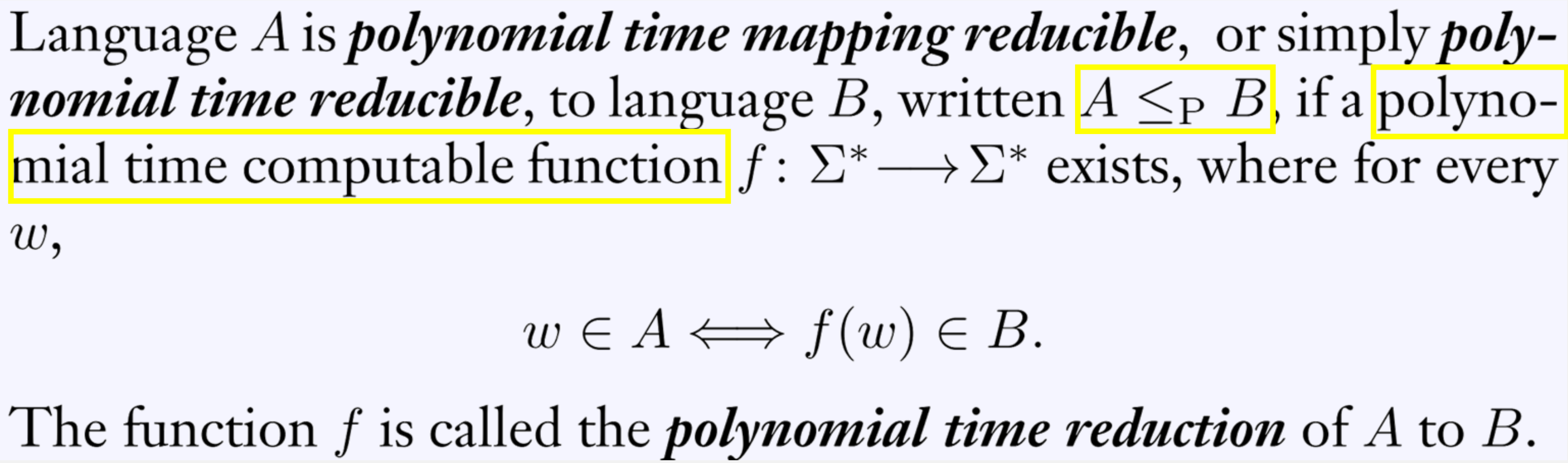
Language A is ***mapping reducible*** to language
if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$,

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

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

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 **reverse 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,

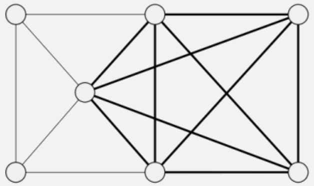
$$w \in A \Longleftrightarrow f(w) \in B.$$

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.

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

Last Time: *CLIQUE* is in NP



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

PROOF IDEA The clique is the certificate.

PROOF The following is a verifier V for *CLIQUE*.

```
V = "On input ⟨⟨G, k⟩, c⟩:
        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

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

Theorem: SAT is in **NP**:

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

Verifier:

On input $<\phi, c>$, 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 $<\phi>$, 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 ($\land$, $\lor$, and $\neg$)
Formula ϕ	Combines vars and operations	$(\overline{x} \land y) \lor (x \land \overline{z})$
Literal	A var or a negated var	x or $\overline{x}$.
Clause	**Literals** ORed together	$(x_1 \lor \overline{x_2} \lor \overline{x_3} \lor x_4)$

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 (∧, ∨, and ¬)
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	$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge (x_3 \vee \overline{x_5} \vee x_6)$

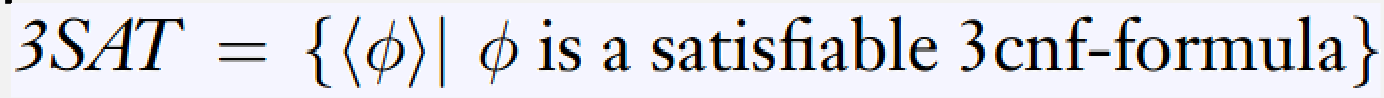
∧ = AND = “Conjunction”
∨ = OR = “Disjunction”
¬ = NOT = “Negation”

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**	$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4)$

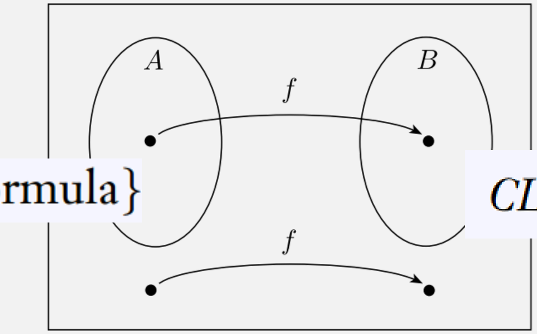
$\wedge$ = AND = “Conjunction”
$\vee$ = OR = “Disjunction”
$\neg$ = NOT = “Negation”

The *3SAT* Problem



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

Theorem: $\mathit{3SAT}$ is polynomial time reducible to CLIQUE.



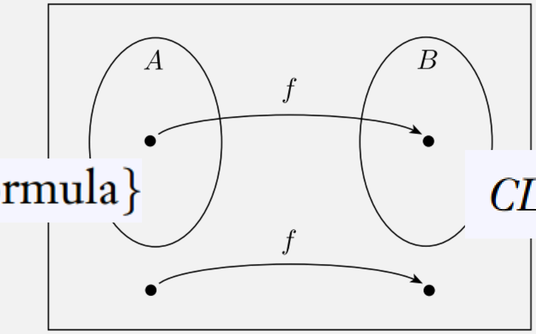
$\mathit{3SAT} = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3cnf-formula}\}$

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

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 **reverse direction**)

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



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

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

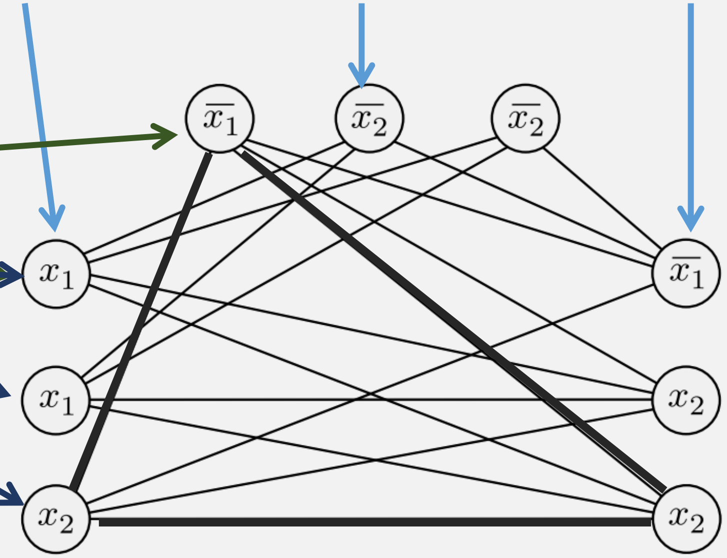
Need: poly time computable fn converting a 3cnf-formula …

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

Don't forget iff

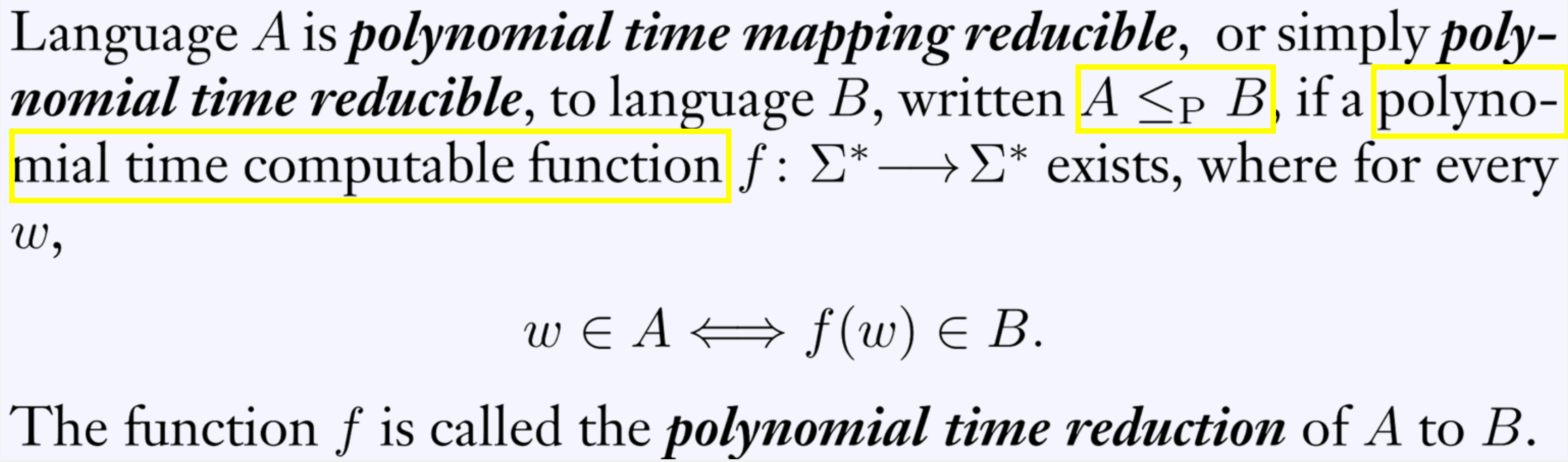


Runs in **poly time:**
- # literals = # nodes $O(\boldsymbol{n})$
- # edges poly in # nodes $O(\boldsymbol{n^2})$

⇒ If $\phi \in 3SAT$
- Then each clause has a TRUE literal
  - Those are nodes in the 3-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

Polynomial Time Mapping Reducibility



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

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

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

What is this used for?

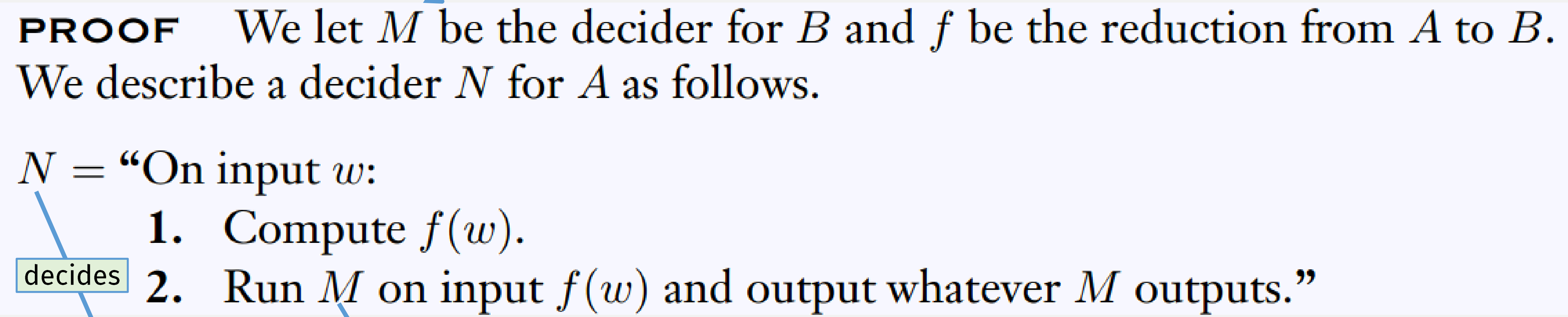
poly time

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 $A \le_m B$ and B is decidable, then A is decidable.

Has a decider

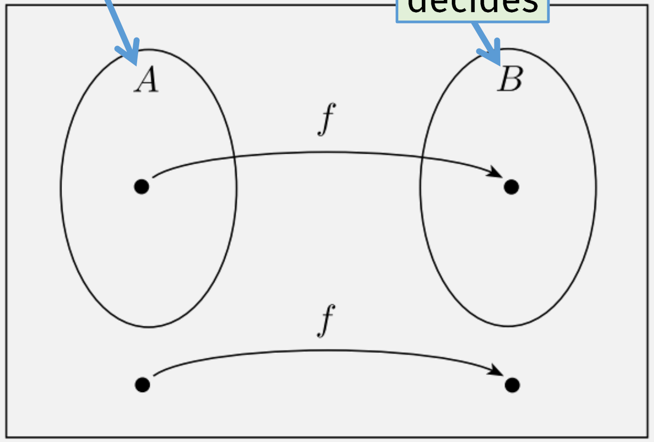


PROOF We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows.

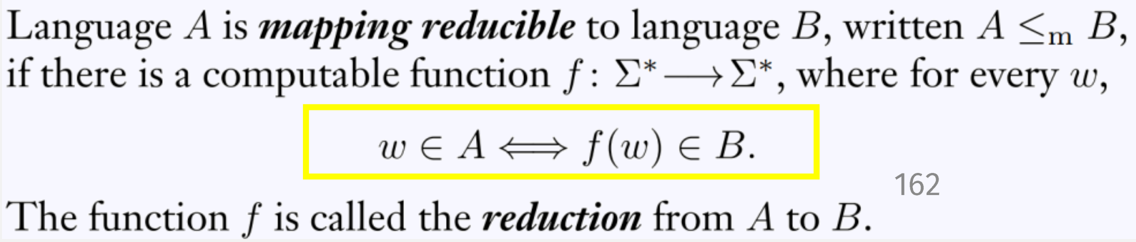
N = "On input w:
1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

decides

decides



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

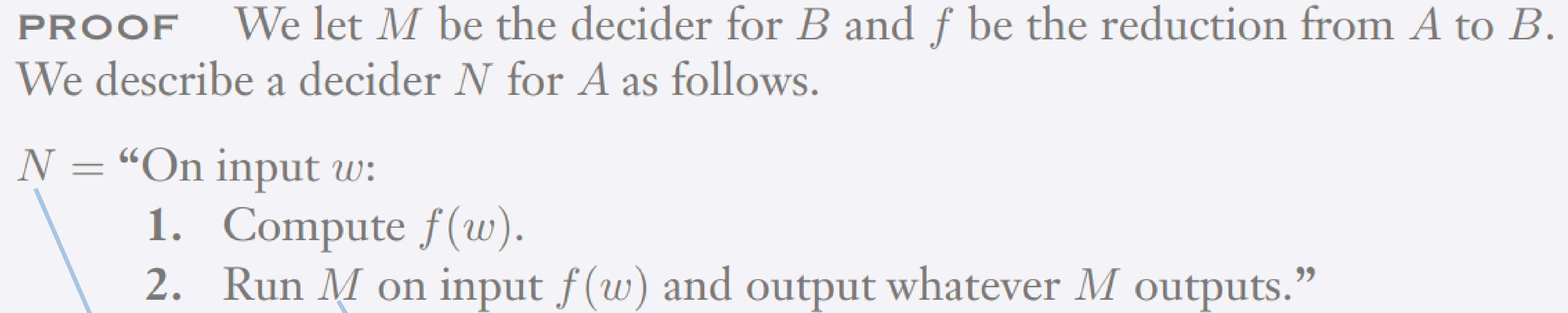


Language A is ***mapping reducible*** to language B, written $A \le_m B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

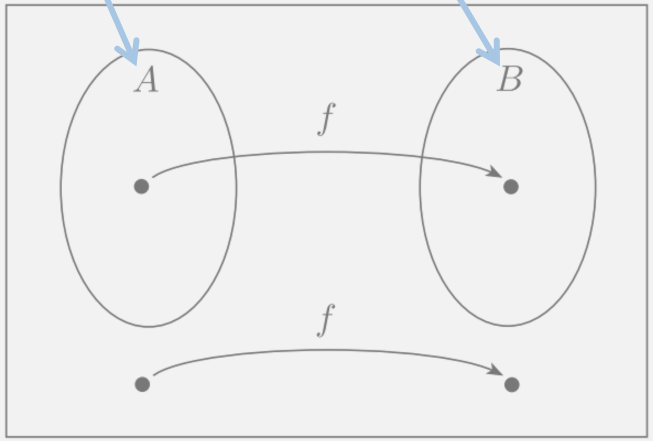
The function f is called the ***reduction*** from A to B.

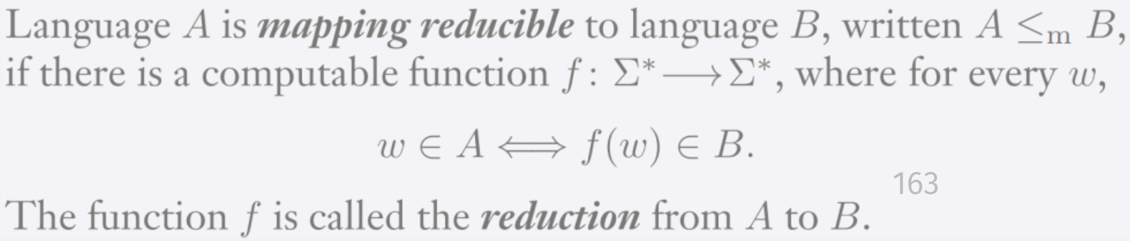
Thm: If $A \leq_{\cancel{\mathrm{m}}\,\mathbf{P}} B$ and B ~~is decidable,~~ $\in \mathrm{P}$ then A ~~is decidable.~~ $\in \mathrm{P}$



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,

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

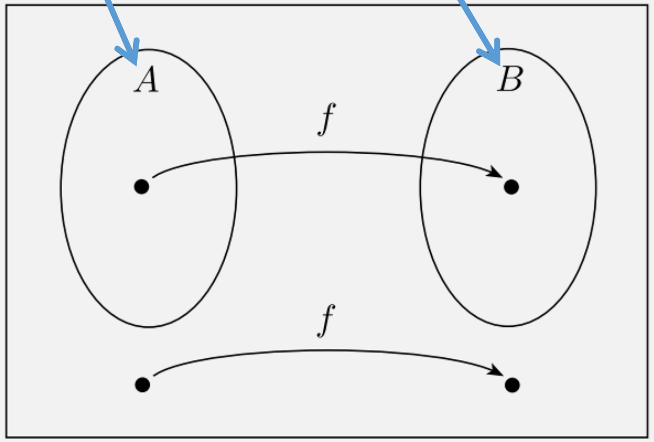
The function f is called the ***reduction*** from A to B.

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

PROOF We let M be the 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."

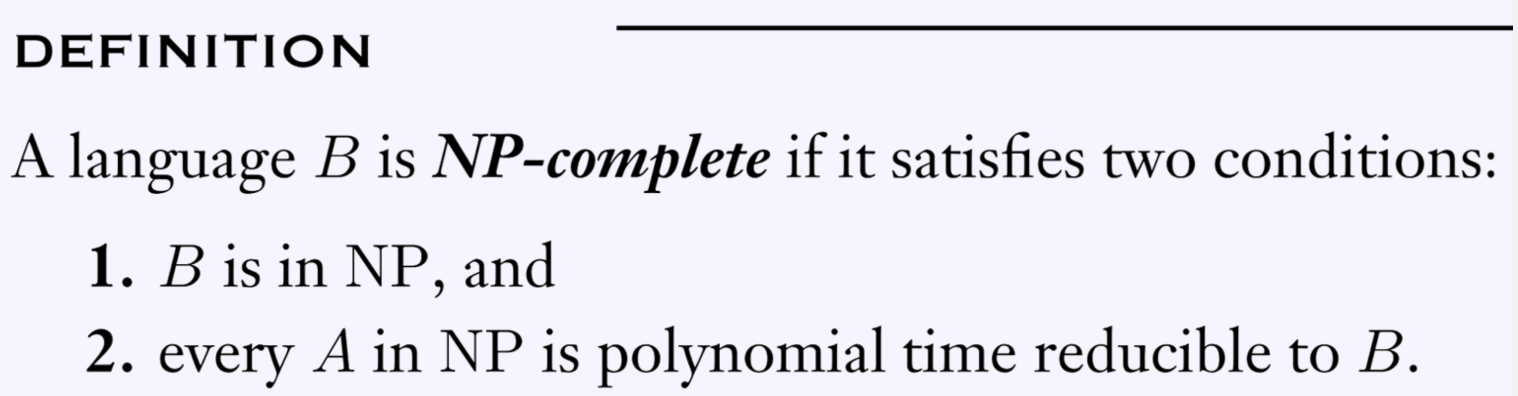


Language A is poly time ***mapping reducible*** to language B, written $A \le_{\mathrm{m}} B$, if there is a computable function $f\colon \Sigma^* \longrightarrow \Sigma^*$, where for every w,

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

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

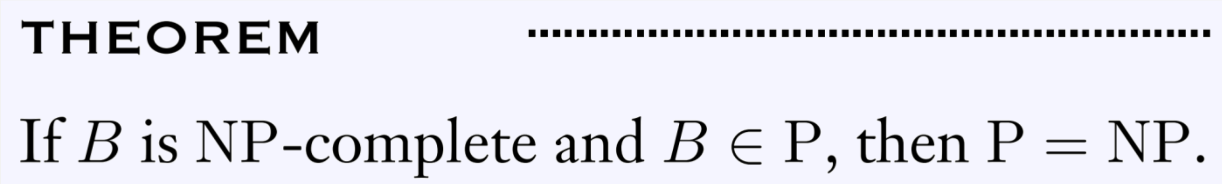
NP-Completeness


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.

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


THEOREM

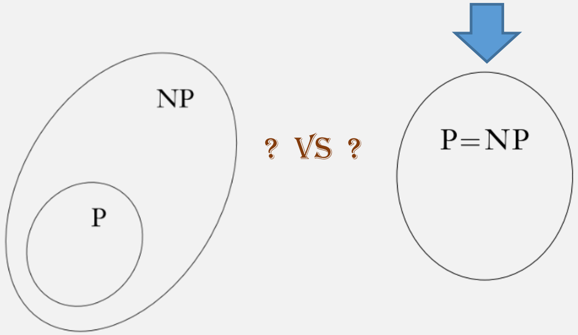
If B is NP-complete and $B \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$.

assume

THEOREM

Proof:

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



DEFINITION

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

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

$A \leq_{\mathrm{P}} B$

P

r A → verifier for A that ignores its certificate

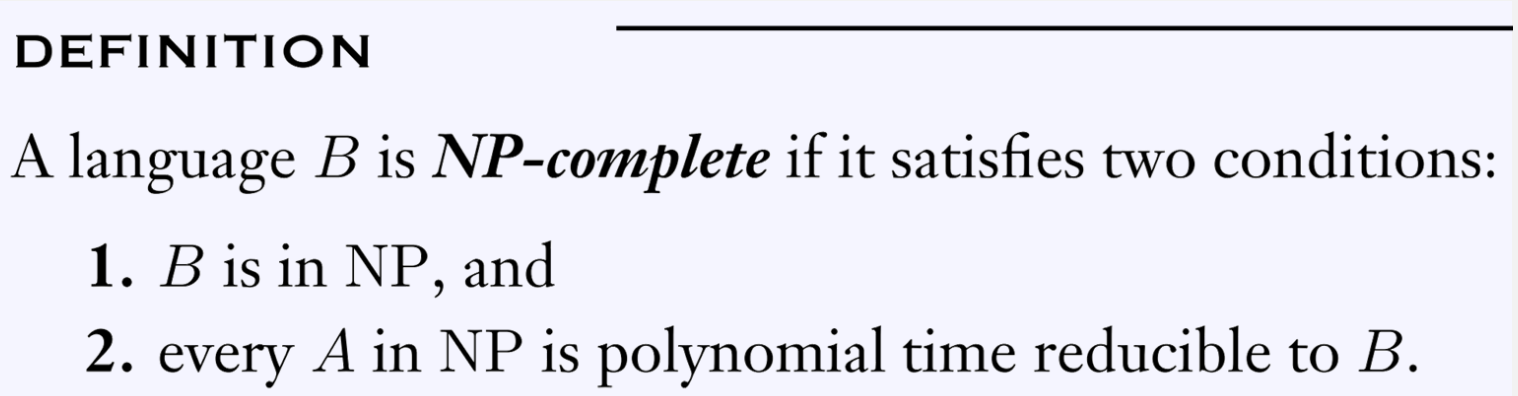
2. If **a language** A ∈ **NP**, then A ∈ **P**
   - Given **a language** A ∈ **NP** …
   - **… can poly time mapping reduce** A **to** B --- why?
     - because B **is NP-Complete** (assumption)
   - Then A **also** ∈ **P** …
     - Because $A \leq_{\mathrm{P}} B$ and $B \in \mathrm{P}$, then $A \in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

(prev slide)

So to prove P = NP, we only need to **find a poly-time algorithm** for <u>one</u> NP-Complete problem!

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

NP-Completeness

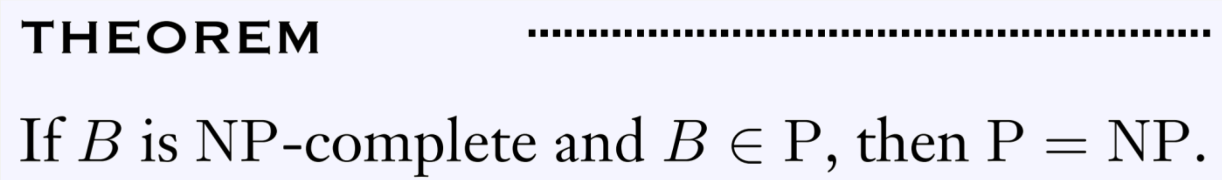


DEFINITION

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.

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



THEOREM

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

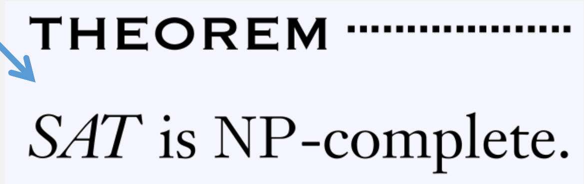
But we still don't know any **NP**-Complete problems!

So to prove **P** = **NP**, we only need to find a poly-time algorithm for one **NP**-Complete problem!

Figuring out the first one is hard!
(just like figuring out the first undecidable problem was hard!)

The Cook-Levin Theorem

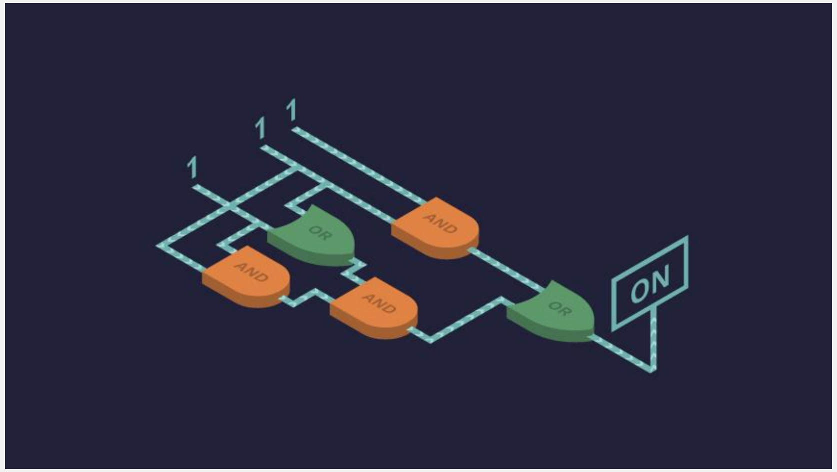
The first **NP**-Complete problem


THEOREM

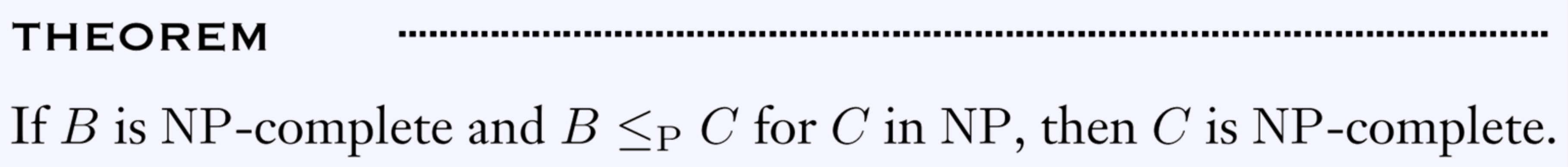
SAT is NP-complete.

(complicated proof --- defer explaining for now)

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



After this, it’ll be much easier to find other **NP**-Complete problems!


THEOREM

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

THEOREM

known

unknown

Key Thm: 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**

DEFINITION

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

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

Proof:

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

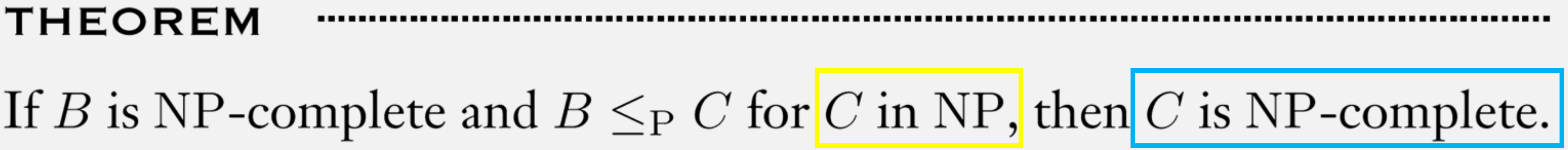
THEOREM

<u>Using</u>: If B is NP-complete and $B \leq_{\mathrm{P}} C$ for C in NP, then C is NP-complete.

<u>**3 steps**</u> to prove a language C is **NP**-complete:

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

To show poly time <u>mapping reducibility</u>:
1. create **computable fn**,
2. show that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
   (or **contrapositive** of **reverse direction**)


THEOREM

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

3 steps to prove a language C is **NP**-complete:

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

Example:

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

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

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

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

Let ***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:

- Accept if *c* satisfies *ϕ*

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

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

3 steps to prove a language is **NP**-complete:

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

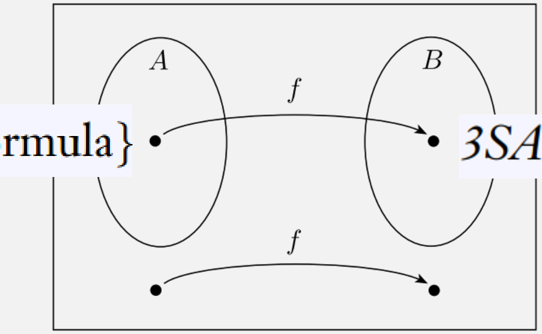
Example:

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

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

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

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



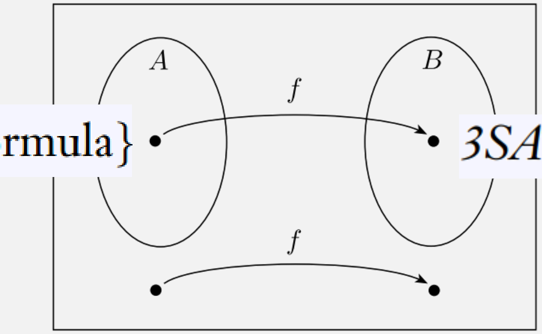
$3SAT \;=\; \{\langle\phi\rangle \mid\; \phi \text{ is a satisfiable 3cnf-formula}\}$

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.,
    - ⇒ **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 **reverse direction**)
    - ⇐ (alternative) **if $\phi \notin SAT$, then $f(\phi) \notin 3SAT$**

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

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



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

Want: 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

      $\neg(P \lor Q) \boxed{\iff} (\neg P) \land (\neg Q)$ $\neg(P \land Q) \boxed{\iff} (\neg P) \lor (\neg Q)$ $\boxed{O(\boldsymbol{n})}$

   b) Distribute ORs to get ANDs outside of parens

      $(P \lor (Q \land R)) \boxed{\Leftrightarrow} ((P \lor Q) \land (P \lor R))$ $\boxed{O(\boldsymbol{n})}$

2. Convert to 3CNF by adding new variables

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

Remaining step: show iff relation holds …

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

THEOREM

<u>Using</u>: If B is NP-complete and $B \leq_P C$ for C in NP, then C is NP-complete.

<u>3 steps</u> to prove a language is **NP**-complete:

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

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

Example:

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

☑1. Show $3SAT$ is in **NP**

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

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

THEOREM

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

3 steps to prove a language is **NP**-complete:

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

Example:

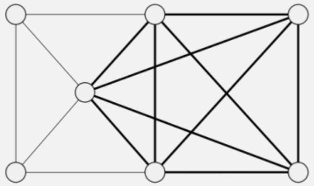
Let C = ~~*3SAT*~~ ***CLIQUE***, to prove ~~*3SAT*~~ ***CLIQUE*** is **NP**-Complete:

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

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

? 3. Show a poly time mapping reduction from ***3SAT*** to ~~*3SAT*~~ ***CLIQUE***

Flashback: *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.

Let $\boldsymbol{n}$ = # nodes in G

c is at most $\boldsymbol{n}$

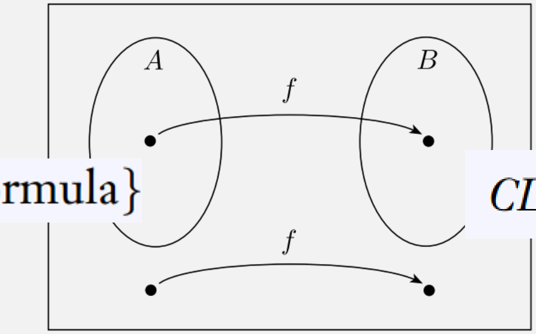
PROOF The following is a verifier V for *CLIQUE*.

```
V = "On input ⟨⟨G, k⟩, c⟩:
        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."
```

For each node in c, check whether it's in G: $O(\boldsymbol{n})$

For each pair of nodes in c, check whether there's an edge in G: $O(\boldsymbol{n}^2)$

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



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

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

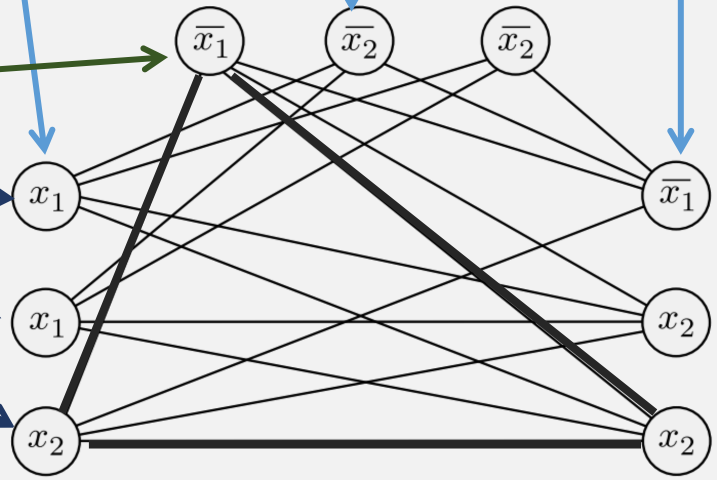
Need: poly time computable fn converting a 3cnf-formula …

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

Don't forget iff



Runs in **poly time:**

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

⇒ 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 $\phi \notin 3SAT$

- 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

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

3 steps to prove a language is **NP**-complete:

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

Example:

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

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

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

☑ 3. Show a poly time mapping reduction from ***3SAT*** to ~~*3SAT*~~ ***CLIQUE***

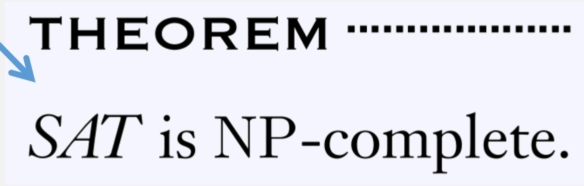
NP-Complete problems, so far

- $SAT = \{\langle \phi \rangle | \ \phi \text{ is a satisfiable Boolean formula}\}$ (havent proven yet)
- $3SAT \ = \ \{\langle \phi \rangle | \ \phi \text{ is a satisfiable 3cnf-formula}\}$ (reduced *SAT* to *3SAT*)
- $CLIQUE = \{\langle G, k \rangle | \ G \text{ is an undirected graph with a } k\text{-clique}\}$ (reduced *3SAT* to *CLIQUE*)

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

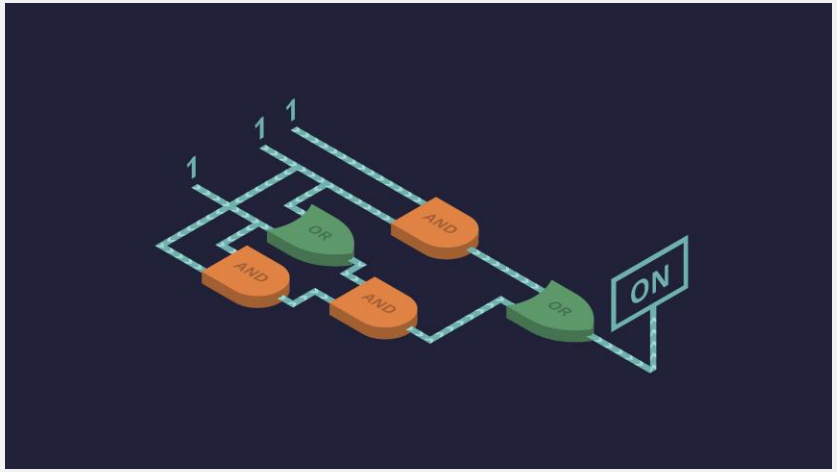
Next Time: The Cook-Levin Theorem

The first **NP**-Complete problem

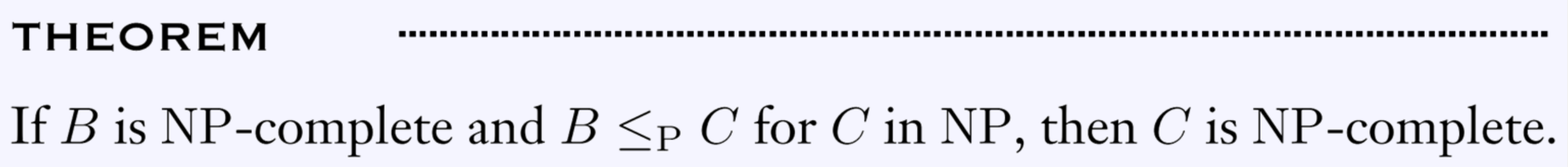

THEOREM

SAT is NP-complete.

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



After this, it'll be much easier to find other **NP**-Complete problems!


THEOREM

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

Quiz 5/8

On gradescope