UMB CS 420

NP-Completeness

Monday, May 8, 2023

---

### Chotchkie's Restaurant

**Appetizers**

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mixed Fruit</td>
<td>2.15</td>
</tr>
<tr>
<td>French Fries</td>
<td>2.75</td>
</tr>
<tr>
<td>Side Salad</td>
<td>3.35</td>
</tr>
<tr>
<td>Hot Wings</td>
<td>3.55</td>
</tr>
<tr>
<td>Mozzarella Sticks</td>
<td>4.20</td>
</tr>
<tr>
<td>Sampler Plate</td>
<td>5.80</td>
</tr>
</tbody>
</table>

**Sandwiches**

- Barbecue: 6.55

---

**MY HOBBY:**

Embedding NP-Complete Problems in Restaurant Orders

---

We'd like exactly $15's worth of appetizers please.

...EXACTLY? Um...

Here's the paper on the knapsack problem, might help you out.

Listen, I have six other tables to get to -

As fast as possible of course. Want something on traveling salesmen?
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

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.

- A certificate $c$ 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.

**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 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
**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 $P = NP$?

- 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 $B$.

*Must prove for all langs, not just a single language*

*What’s this?*

*hard????*

*easy*
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 reverse direction**)

... 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 \textit{mapping reducible} to language $B$ if there is a computable function $f : \Sigma^* \to \Sigma^*$, $w \in A \iff f(w) \in B$.

The function $f$ is called the \textit{reduction} from $A$ to $B$.

Language $A$ is \textit{polynomial time mapping reducible}, or simply \textit{polynomial time reducible}, to language $B$, written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every $w$,

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

The function $f$ is called the \textit{polynomial time reduction} of $A$ to $B$.

To show \textit{poly time mapping reducibility}:

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

A function $f : \Sigma^* \to \Sigma^*$ is a \textit{computable function} if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
Theorem: 3SAT is polynomial time reducible to CLIQUE.
Last Time: CLIQUE is in NP

\[
CLIQUE = \{ (G, k) : G \text{ is an undirected graph with a } k\text{-clique} \}
\]

**Proof Idea**  The clique is the certificate.

**Proof**  The following is a verifier $V$ for CLIQUE.

$V$ = "On input $(G, k, c)$:

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

<table>
<thead>
<tr>
<th>A Boolean Value</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
</tbody>
</table>
### Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
</tbody>
</table>
## Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ($\wedge$, $\vee$, and $\neg$)</td>
</tr>
</tbody>
</table>
## Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT (∧, ∨, and ¬)</td>
</tr>
<tr>
<td>Formula $\phi$</td>
<td>Combines vars and operations</td>
<td>$(x \wedge y) \lor (x \wedge \overline{z})$</td>
</tr>
</tbody>
</table>
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 \((\bar{x} \land y) \lor (x \land \bar{z})\) satisfiable?
  • Yes
  • \(x = \text{FALSE},\)
    \(y = \text{TRUE},\)
    \(z = \text{FALSE}\)
The Boolean Satisfiability Problem

**Theorem:** \( SAT \) is in \( \textbf{NP} \):

- Let \( n \) = the number of variables in the formula

**Verifier:**

On input \( <\phi, c> \), where \( c \) is a possible assignment of variables in \( \phi \) to values:

- Plug values from \( c \) into \( \phi \), **Accept** if result is TRUE

**Running Time:** \( O(n) \)

**Non-deterministic Decider:**

On input \( <\phi> \), 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

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT $\land, \lor, \text{ and } \neg$</td>
</tr>
<tr>
<td>Formula $\phi$</td>
<td>Combines vars and operations</td>
<td>$(\overline{x} \land y) \lor (x \land \overline{z})$</td>
</tr>
</tbody>
</table>
## More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>$x, y, z$</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ($\land, \lor$, and $\neg$)</td>
</tr>
<tr>
<td>Formula $\phi$</td>
<td>Combines vars and operations</td>
<td>$(\overline{x} \land y) \lor (x \land \overline{z})$</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>$x$ or $\overline{x}$</td>
</tr>
</tbody>
</table>
# More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor, \text{and} \neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines vars and operations</td>
<td>((\bar{x} \land y) \lor (x \land \bar{z}))</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>(x) or (\bar{x}),</td>
</tr>
<tr>
<td>Clause</td>
<td><strong>Literals</strong> ORed together</td>
<td>((x_1 \lor \bar{x}_2 \lor \bar{x}_3 \lor x_4))</td>
</tr>
</tbody>
</table>
# More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean _____</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor, \text{ and } \neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines vars and operations</td>
<td>((\overline{x} \land y) \lor (x \land \overline{z}))</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>(x) or (\overline{x})</td>
</tr>
<tr>
<td>Clause</td>
<td>Literals ORed together</td>
<td>((x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4))</td>
</tr>
<tr>
<td>Conjunctive Normal Form (CNF)</td>
<td>Clauses ANDed together</td>
<td>((x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4) \land (x_3 \lor \overline{x}_5 \lor x_6))</td>
</tr>
</tbody>
</table>

\(\land = \text{AND} = \text{“Conjunction”}\)

\(\lor = \text{OR} = \text{“Disjunction”}\)

\(\neg = \text{NOT} = \text{“Negation”}\)
# More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean _______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ($\land$, $\lor$, and $\neg$)</td>
</tr>
<tr>
<td>Formula $\phi$</td>
<td>Combines vars and operations</td>
<td>$(\overline{x} \land y) \lor (x \land z)$</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>$x$ or $\overline{x}$</td>
</tr>
<tr>
<td>Clause</td>
<td>Literals ORed together</td>
<td>$(x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4)$</td>
</tr>
<tr>
<td>Conjunctive Normal Form (CNF)</td>
<td>Clauses ANDed together</td>
<td>$(x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4) \land (x_3 \lor \overline{x}_5 \lor x_6)$</td>
</tr>
<tr>
<td>3CNF Formula</td>
<td>Three literals in each clause</td>
<td>$(x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_3 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6 \lor x_4)$</td>
</tr>
</tbody>
</table>

$\land$ = AND = “Conjunction”  
$\lor$ = OR = “Disjunction”  
$\neg$ = NOT = “Negation”
The $3SAT$ Problem

$$3SAT = \{ \langle \phi \rangle | \phi \text{ is a satisfiable 3cnf-formula} \}$$
Theorem: \(3SAT\) is polynomial time reducible to \(CLIQUE\).

\[3SAT = \{\langle\phi\rangle|\ \text{\phi is a satisfiable 3cnf-formula}\}\]

\[CLIQUE = \{\langle G, k\rangle| G is an undirected graph with a \text{k-clique}\}\]

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 ...

- ... 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

\[ \phi = (x_1 \lor x_1 \lor \overline{x}_2) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_2) \land (\overline{x}_1 \lor x_2 \lor \overline{x}_2) \]

**Example:**

**Runs in poly time:**
- # literals = # nodes
- # edges poly in # nodes

**Don't forget:**

\[ \Rightarrow \text{If } \phi \in 3SAT \]

- Then each clause has a TRUE literal
  - Those are nodes in the 3-clique!
  - E.g., \( x_1 = 0, x_2 = 1 \)

\[ \Leftarrow \text{If } \phi \notin 3SAT \]

- 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
Language $A$ is \textit{polynomial time mapping reducible}, or simply \textit{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 $w$,

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

The function $f$ is called the \textit{polynomial time reduction} of $A$ to $B$.

What is this used for?

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a \textit{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.

**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 = \text{“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 $A \leq_m B$ and $B \in \mathbb{P}$ is decidable, then $A \in \mathbb{P}$ 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.

$N = \text{“On input } w:\n\quad 1. \text{ Compute } f(w).$
\quad 2. \text{ Run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs.”}
Thm: If \( A \leq_m B \) and \( B \in \mathbb{P} \) is decidable, then \( A \in \mathbb{P} \).

**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.

\[ N = \text{"On input } w:\]
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 \( B \).
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 $B \in P$, then $P = NP$. 
Proof:

**THEOREM**

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

**DEFINITION**

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

1. \( B \) is in NP, and \( A \leq_P B \)
2. every \( A \) in NP is polynomial time reducible to \( B \).

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

So to prove \( P = NP \), we only need to find a poly-time algorithm for one 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 $B \in P$, then $P = NP$.

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

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

So to prove $P = NP$, we only need to find a poly-time algorithm for one NP-Complete problem!
The Cook-Levin Theorem

The first **NP-Complete** problem

**THEOREM**

\( SAT \) is NP-complete.

(complicated proof --- defer explaining for now)

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

**THEOREM**

If \( B \) is NP-complete and \( B \leq_P C \) for \( C \) in NP, then \( C \) is NP-complete.
Key Thm: If $B$ is NP-complete and $B \leq_P C$ for $C$ in NP, then $C$ is NP-complete.

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 $A \rightarrow C$ by:
  - First reduce $A \rightarrow B$ in poly time
    - Can do this because $B$ is NP-Complete
  - Then reduce $B \rightarrow C$ in poly time
    - This is given
- **Total run time**: Poly time + poly time = poly time
**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 $C$ 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 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**

**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 $C$ 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:** \(3\text{SAT} \text{ is in } \mathbf{NP}\)

Let \(n\) = the number of variables in the formula

Verifier:
On input \(<\phi, c>\), 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 \(<\phi>\), 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

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 $C = 3SAT$, 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 \text{computable function} \( f \),
2. show that it \text{runs in poly time},
3. then show \text{forward direction} of mapping reduc.,
   \[ \Rightarrow \text{if } \phi \in SAT, \text{ then } f(\phi) \in 3SAT \]
4. and \text{reverse direction}
   \[ \Leftarrow \text{if } f(\phi) \in 3SAT, \text{ then } \phi \in SAT \]
   (or \text{contrapositive of reverse direction})
   \[ \Leftarrow (\text{alternative}) \text{ if } \phi \notin SAT, \text{ then } f(\phi) \notin 3SAT \]
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 \( \phi \) to 3CNF:

1. Convert \( \phi \) to CNF (an AND of OR clauses)
   a) Use DeMorgan's Law to push negations onto literals
      \[
      \neg (P \lor Q) \iff (\neg P) \land (\neg Q)
      \]
      \[
      \neg (P \land Q) \iff (\neg P) \lor (\neg Q)
      \]
      \( O(n) \)
   b) Distribute ORs to get ANDs outside of parens
      \[
      (P \lor (Q \land R)) \iff ((P \lor Q) \land (P \lor R))
      \]
      \( O(n) \)

2. Convert to 3CNF by adding new variables
   \[
   (a_1 \lor a_2 \lor a_3 \lor a_4) \iff (a_1 \lor a_2 \lor z) \land (\neg z \lor a_3 \lor a_4)
   \]
   \( O(n) \)

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

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 $C = 3SAT$, 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

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

**Proof Idea** The clique is the certificate.

**Proof** The following is a verifier $V$ for CLIQUE.

$V = \text{"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."

Let $n = \#$ nodes in $G$

$c$ is at most $n$

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

For each pair of nodes in $c$, check whether there's an edge in $G$: $O(n^2)$
Flashback: \(3SAT\) is polynomial time reducible to \(CLIQUE\).

\[ 3SAT = \{ (\phi) | \phi \text{ is a satisfiable 3cnf-formula} \} \]

\[ CLIQUE = \{ (G, k) | G \text{ is an undirected graph with a } k \text{-clique} \} \]

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

... 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

⇒ 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

Example:
\[ \phi = (x_1 \lor x_1 \lor \overline{x_2}) \land (x_1 \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor \overline{x_2}) \]

Runs in poly time:
- # literals = # nodes
- # edges poly in # nodes

\[ O(n) \]
\[ O(n^2) \]
**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:

- $\square$ 1. Show $3SAT\_CLIQUE$ is in NP
- $\square$ 2. Choose $B$, the NP-complete problem to reduce from: SAT $\rightarrow$ 3SAT
- $\square$ 3. Show a poly time mapping reduction from 3SAT to 3SAT-CLIQUE
**NP-Complete problems, so far**

- $SAT = \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}$ (haven't 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: 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 \( B \leq_P C \) for \( C \) in NP, then \( C \) is NP-complete.
Quiz 5/8
On gradescope