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

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

A verifier for a language $A$ is an algorithm $V$, where $A = \{ w | V \text{ accepts } \langle w, c \rangle \text{ for some string } c \}$.

We measure the time of a verifier only in terms of the length of $w$, so a \textbf{polynomial time verifier} runs in polynomial time in the length of $w$. A language $A$ is \textit{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.

**THEOREM**

A language is in NPiff 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, \ldots, x_k\}, \text{ and for some } y_1, \ldots, y_l \subseteq \{x_1, \ldots, 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., $\langle \{4, 11, 16, 21, 27\}, 25 \rangle \in \text{ SUBSET-SUM}$
Theorem: $SUBSET-SUM$ is in NP

$SUBSET-SUM = \{ (S, t) | S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_t\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t \}$

**Proof Idea**: The subset is the certificate.

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

**Proof**: The following is a verifier $V$ for $SUBSET-SUM$.

$V = \text{"On input } \langle S, t, 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: \textit{SUBSET-SUM} is in NP

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

To prove a lang is in \textit{NP}, create either:
- Deterministic poly time verifier
- Nondeterministic poly time decider

\textbf{ALTERNATIVE PROOF}: We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for \textit{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, \textit{accept}; otherwise, \textit{reject}.”

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

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 **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?

One of the Greatest unsolved

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
10.1145/1592164.1592186

• 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 \( B \).

**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 $w$,

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

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

**IMPORTANT:** “if and only if” ...

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

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

$$B_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ halts on input } w \}$$

... 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 $B$.

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 $w$,

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

The function $f$ is called the **polynomial time reduction** of $A$ 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)

**Don’t forget:** “if and only if” ...

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.

**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 $f: \Sigma^* \to \Sigma^*$, where for every $w$, $w \in A \iff f(w) \in B$.

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

This proof only works because of the if-and-only-if requirement.
Thm: \( A \leq_{m} P \) if \( A \leq_{m} B \) and \( B \in \mathcal{P} \) is decidable, then \( A \in \mathcal{P} \) 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.

\[
N = \text{"On input } w: \\
1. \text{ Compute } f(w). \\
2. \text{ Run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs."}
\]

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 $A \in \mathbb{P}$.

**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.”
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 (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 $A \leq_P B$ and $B \in P$, then $A \in P$

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**

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

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

**PROOF** The following is a verifier \( V \) for **CLIQUE**.

\[ V = \text{“On input } \langle \langle G, k \rangle, c \rangle \text{:} \]

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 ______</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>
</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 <strong>variables</strong></td>
<td>AND, OR, NOT ((\wedge, \lor, \text{and} \neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines <strong>vars</strong> and <strong>operations</strong></td>
<td>((\overline{x} \land y) \lor (x \land \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 \((\overline{x} \land y) \lor (x \land \overline{z})\) 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 \( \langle \phi, c \rangle \), 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 \( \langle \phi \rangle \), 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

<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, \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 \text{ 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>$(\overline{x} \land y) \lor (x \land \overline{z})$</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><strong>Literals</strong> ORed together</td>
<td>$(x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4)$</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><strong>Literals</strong> 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><strong>Clauses</strong> 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><strong>A Boolean ______</strong></th>
<th><strong>Is ...</strong></th>
<th><strong>Example:</strong></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><strong>Literals</strong> 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><strong>Clauses</strong> 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 <strong>literals</strong> 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>

\(\land = \text{AND} = \text{“Conjunction”}\)  
\(\lor = \text{OR} = \text{“Disjunction”}\)  
\(\neg = \text{NOT} = \text{“Negation”}\)
The $3SAT$ Problem

$$3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula} \}$$
Theorem: \( SAT \) is Poly Time Reducible to \( 3SAT \)

\[
SAT = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \}
\]

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

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*

\[ SAT = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \} \]

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

**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
      \[ \neg (P \lor Q) \iff (\neg P) \land (\neg Q) \]
      \[ \neg (P \land Q) \iff (\neg P) \lor (\neg Q) \quad O(n) \]
   b) Distribute ORs to get ANDs outside of parens
      \[ (P \lor (Q \land R)) \iff ((P \lor Q) \land (P \lor R)) \quad O(n) \]

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

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 $CLIQUE$.

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

$CLIQUE = \{\langle G, k \rangle | G$ is an undirected graph with a $k$-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 forward direction)
**Theorem:** \(3SAT\) is polynomial time reducible to \(CLIQUE\).

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

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

**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

**\(\Rightarrow\) 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 \(\phi \notin 3SAT\)**
- 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 \(CLIQUE\).

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

\(CLIQUE = \{(G, k) | G\) is an undirected graph with a \(k\)-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 $B$.

It's very hard to prove the first NP-Complete problem!

(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