The Cook-Levin Theorem
(the 1st NP-Complete Problem)

Monday, May 2, 2022
Announcements

• HW 11 out
  • Due Tues 5/3 11:59pm EST

• 4 lectures left!

• Course evals coming

Jeff Atwood
@codinghorror

There are two hard things in computer science: cache invalidation, naming things, and off-by-one errors.
Last Time: **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 $B \leq_P C$ for $C$ in NP, then $C$ is NP-complete.

---

**Notes**

- Must prove for all langs, not just a single language.
- It’s very hard to prove the first NP-Complete problem! (Just like figuring out the first undecidable problem was hard!)
- But after we find one, then we use that problem to prove other problems NP-Complete!
Today: The Cook-Levin Theorem

The first **NP-Complete** problem

THEOREM .................. 

SAT is NP-complete.

It makes sense that every problem can be reduced to it ...
The Cook-Levin Theorem

SAT is NP-complete.

THEOREM

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 $B$. 

1971

The Complexity of Theorem-Proving Procedures
Stephen A. Cook
University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that [tautologies] is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time.

In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles.

1973

КРАТКИЕ СООБЩЕНИЯ

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Леони

В статье рассматриваются несколько известных массовых задач "переборного типа" и доказывается, что эти задачи можно решить лишь за такое время, за которое можно решить вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем теодолитных элементов групп, гомеоморфности многогранников, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения этих задач не означает для них аналогичного вопроса из-за фантастически большого объема работы, предполагаемого этими алгоритмами. Такова ситуация с так называемыми переборными задачами: минимизация булевых функций, поиска доказательств ограниченной длины, вычисления изоморфности графов и другими. Все эти задачи решаются трансляционными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что
Reducing every **NP** language to **SAT**

Some **NP** lang = \{w | w is ???\}

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

How can we reduce some w to a Boolean formula if we don’t know w???
Proving theorems about an entire class of langs?

We can still use general facts about the languages!

E.g., “Prove that every regular language is in P”
  • Even though we don’t know what the language is ...
  • We do know that every regular lang has an DFA accepting it

E.g., “Prove that every CFL decidable”
  • Even though we don’t know what the language is ...
  • We do know that every CFL has a CFG representation ...
  • And every CFG has a Chomsky Normal Form
What do we know about NP languages?

They are:

1. Verified by a deterministic poly time verifier

2. Decided by a nondeterministic poly time decider (NTM)

Let’s use this one
Flashback: Non-deterministic TMs

- Formally defined with states, transitions, alphabet ...

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet not containing the blank symbol \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) and \(\Sigma \subseteq \Gamma\),
4. \(\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})\) transition function,
5. \(q_0 \in Q\) is the start state,
6. \(q_{\text{accept}} \in Q\) is the accept state, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{reject}} \neq q_{\text{accept}}\).

- Computation can branch
- Each node in the tree represents a TM configuration
Flashback: TM Config = State + Head + Tape
Flashback: Non-deterministic TMs

- Formally defined with states, transitions, alphabet ...

**Idea:** We don’t know the specific language or strings in the language, but ...

... we know those strings must have an accepting sequence of configurations!

A *Turing machine* is a 7-tuple, \( (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \), where \( Q, \Sigma, \Gamma \) are all finite sets and

1. \( Q \) is the set of states,
2. \( \Sigma \) is the input alphabet not containing the blank symbol \( \sqcup \),
3. \( \Gamma \) is the tape alphabet, where \( \sqcup \in \Gamma \) and \( \Sigma \subseteq \Gamma \),
4. \( \delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}) \) is the transition function,
5. \( q_0 \in Q \) is the start state,
6. \( q_{\text{accept}} \in Q \) is the accept state, and
7. \( q_{\text{reject}} \in Q \) is the reject state, where \( q_{\text{reject}} \neq q_{\text{accept}} \).

- Computation can branch
- Each node in the tree represents a TM configuration
- Transitions specify valid configuration sequences
Accepting config sequence = “Tableau”

- input $w = w_1 \ldots w_n$
- Assume configs start/end with $#$
- Must have an accepting config
- At most $n^k$ configs
  - (why?)
- Each config has length $n^k$
  - (why?)
Theorem: $SAT$ is NP-complete

Proof idea:
• Give an algorithm that reduces accepting tableaus to satisfiable formulas
• Thus **every string in the NP lang will be mapped to a sat. formula**
  • and **vice versa**

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

Resulting formulas will have **four components:**
$$\phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{move}} \land \phi_{\text{accept}}$$
Tableau Terminology

• A tableau **cell** has coordinate $i,j$

• A cell has **symbol**: $s \in C = Q \cup \Gamma \cup \{\#\}$

A *Turing machine* is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, where $Q, \Sigma, \Gamma$ are all finite sets and

1. $Q$ is the set of states,
2. $\Sigma$ is the input alphabet not containing the **blank symbol** $\sqcup$,
3. $\Gamma$ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{\text{accept}} \in Q$ is the accept state, and
7. $q_{\text{reject}} \in Q$ is the reject state, where $q_{\text{reject}} \neq q_{\text{accept}}$. 
Formula Variables

- A tableau cell has coordinate $i,j$.

- A cell has symbol: $s \in C = Q \cup \Gamma \cup \{\#\}$.

- For every $i,j,s$ create variable $x_{i,j,s}$.
  - i.e., one var for every possible symbol/cell combination.

- Total variables =
  - # cells * # symbols = $n^k \times n^k \times |C| = O(n^{2k})$.

- Resulting formulas will have four components:
  $\phi_{cell} \land \phi_{start} \land \phi_{move} \land \phi_{accept}$.

- Use these variables to create $\phi_{cell} \land \phi_{start} \land \phi_{move} \land \phi_{accept}$ such that:
  accepting tableau $\iff$ satisfying assignment.

- $\Rightarrow$ If input is accepting tableau, then output satisfiable $\phi$.
  - all four parts of $\phi$ must be TRUE

- $\Leftarrow$ If input is non-accepting tableau, then output unsatisfiable $\phi$.
  - only one part of $\phi$ must be FALSE.
\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s, t \in C \setminus \{\#\}} (x_{i,j,s} \lor \overline{x_{i,j,t}}) \right) \right]
\]

“The following must be TRUE for every cell \(i,j\)”

“The variable for one \(s\) must be TRUE”

And only one variable for some \(s\) must be TRUE

i.e., every cell has a valid character

⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?
  • **Yes**, assign \(x_{i,j,s} = \text{TRUE}\) if it’s in the tableau,
  • and assign other vars = \(\text{FALSE}\)

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?
  • Not necessarily
∀ \phi_{cell} \land \phi_{start} \land \phi_{move} \land \phi_{accept}

The variables in the start config, ANDed together

For a string \( w \), start config is always \#q_0w_1 \ldots w_n \ldots \# 

\phi_{start} = x_{1,1,\#} \land x_{1,2,q_0} \land
\quad x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land
\quad x_{1,n+3,\sqcup} \land \ldots \land x_{1,n^k-1,\sqcup} \land x_{1,n^k,\#}

i.e., tableau has valid start config

⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?
  • Yes, assign \( x_{i,j,s} = \) TRUE if it’s in the tableau,
  • and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?
  • Not necessarily

⇒ accepting tableau: all four must be TRUE
⇐ non-accepting tableau: one must be FALSE
⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?
  • Yes, assign \( x_{ij,s} = \text{TRUE} \) if it’s in the tableau,
  • and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?
  • Yes, because it won’t have \( q_{\text{accept}} \)
• Ensures that every configuration is legal according to the previous configuration and the TM’s δ transitions

• Only need to verify every 2×3 “window”
  • Why?
  • Because in one step, only the cell at the head can change

• E.g., if \( \delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\} \)
  • Which are legal?

![Diagram showing configurations and transitions]

- (a) \[
\begin{array}{ccc}
  a & q_1 & b \\
  q_2 & a & c
\end{array}
\]
- (b) \[
\begin{array}{ccc}
  a & q_1 & b \\
  a & a & q_2
\end{array}
\]
- (c) \[
\begin{array}{ccc}
  a & a & q_1 \\
  a & a & b
\end{array}
\]
- (d) \[
\begin{array}{ccc}
  # & b & a \\
  # & b & a
\end{array}
\]
- (e) \[
\begin{array}{ccc}
  a & b & a \\
  a & b & q_2
\end{array}
\]
- (f) \[
\begin{array}{ccc}
  b & b & b \\
  c & b & b
\end{array}
\]
\[ \phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \left( \text{the } (i, j)\text{-window is legal} \right) \]

\[ \bigvee_{a_1, \ldots, a_6} \left( x_{i, j-1}, a_1 \land x_{i, j}, a_2 \land x_{i, j+1}, a_3 \land x_{i+1, j-1}, a_4 \land x_{i+1, j}, a_5 \land x_{i+1, j+1}, a_6 \right) \]

i.e., all transitions are legal, according to \(\delta\) fn

\[ \Rightarrow \text{Does an accepting tableau correspond to a satisfiable (sub)formula?} \]
- **Yes**, assign \(x_{ij,s} = \text{TRUE}\) if it’s in the tableau,
- and assign other vars = FALSE

\[ \Leftarrow \text{Does a non-accepting tableau correspond to an unsatisfiable formula?} \]
- Not necessarily
\[ \phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \left( \text{the (i, j)-window is legal} \right) \]

is a legal window

\[ \bigvee_{a_1, \ldots, a_6} \left( x_{i,j-1}, a_1 \land x_{i,j}, a_2 \land x_{i,j+1}, a_3 \land x_{i+1,j-1}, a_4 \land x_{i+1,j}, a_5 \land x_{i+1,j+1}, a_6 \right) \]

⇒ Does an accepting tableau correspond to a satisfiable (sub)formula?
  * Yes, assign \( x_{i,j,s} = \text{TRUE} \) if it’s in the tableau,
  * and assign other vars = FALSE

⇐ Does a non-accepting tableau correspond to an unsatisfiable formula?
  * Not necessarily
To Show Poly Time Mapping Reducibility ...

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 that it **runs in poly time**,
3. then show **forward direction** of mapping red.,
4. and **reverse direction**
5. (or contrapositive of reverse direction)
Time complexity of the reduction

• Number of cells = $O(n^{2k})$

$$
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s,t \in C, s \neq t} \left( \overline{x_{i,j,s}} \lor x_{i,j,t} \right) \right) \right] \quad O(n^{2k})
$$

“The following must be TRUE for every cell $i,j$”

“The variable for one $s$ must be TRUE”

And only one variable for some $s$ must be TRUE
Time complexity of the reduction

• Number of cells = \( O(n^{2k}) \)

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s, t \in C \atop s \neq t} (x_{i,j,s} \lor x_{i,j,t}) \right) \right]
\]

\[
\phi_{\text{start}} = x_{1,1,#} \land x_{1,2,q_0} \land x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land x_{1,n+3,\sqcup} \land \ldots \land x_{1,n^k-1,\sqcup} \land x_{1,n^k,#}
\]

\[O(n^k)\]
Time complexity of the reduction

- Number of cells = \(O(n^{2k})\)

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s, t \in C} \left( x_{i,j,s} \lor x_{i,j,t} \right) \right) \right] \quad O(n^{2k})
\]

\[
\phi_{\text{start}} = x_{1,1,#} \land x_{1,2,q_0} \land
x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land
x_{1,n+3,\sqcup} \land \ldots \land x_{1,n^k-1,\sqcup} \land x_{1,n^k,#} \quad O(n^k)
\]

\[
\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}} \quad O(n^{2k})
\]

The state \(q_{\text{accept}}\) must appear in some cell.
Time complexity of the reduction

- Number of cells = $O(n^{2k})$

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s, t \in C \atop s \neq t} (x_{i,j,s} \lor x_{i,j,t}) \right)
\]  

\[O(n^{2k})\]

\[
\phi_{\text{start}} = x_{1,1,\#} \land x_{1,2,q_0} \land
x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land
x_{1,n+3,\sqcup} \land \ldots \land x_{1,n^k-1,\sqcup} \land x_{1,n^k,\#}
\]

\[O(n^k)\]

\[
\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}}
\]

\[O(n^{2k})\]

\[
\phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, \ 1 < j < n^k} \text{(the} \ (i, j)\text{-window is legal)}
\]

\[O(n^{2k})\]
Time complexity of the reduction

• Number of cells = $O(n^{2k})$

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s,t \in C} (x_{i,j,s} \lor x_{i,j,t}) \right) \right] \quad O(n^{2k})
\]

\[
\phi_{\text{start}} = x_{1,1,\#} \land x_{1,2,q_0} \land \\
x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land \\
x_{1,n+3,\sqcup} \land \ldots \land x_{1,n^k-1,\sqcup} \land x_{1,n^k,\#} \quad O(n^k)
\]

\[
\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}} \quad O(n^{2k})
\]

\[
\phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \left( \text{the (i, j)-window is legal} \right) \quad O(n^{2k})
\]

Total: $O(n^{2k})$
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 that it runs in poly time,
- ✓ 3. then show forward direction of mapping red.,
- ✓ 4. and reverse direction
- ✓ (or contrapositive of forward direction)
QED: \textit{SAT} is NP-complete

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

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

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

\[
\phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{move}} \land \phi_{\text{accept}}
\]

Now it will be much easier to prove that other languages are NP-complete!
**THEOREM**

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 \( \text{NP} \) (given), and
  - every lang \( A \) in \( \text{NP} \) reduces to \( C \) in poly time (must show)

- **For every language** \( A \) in \( \text{NP} \), reduce \( A \to C \) by:
  - First reduce \( A \to B \) in poly time
    - Can do this because \( B \) is NP-Complete
  - Then reduce \( B \to C \) 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)
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: $3\text{SAT}$ is in $\textbf{NP}$

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

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 $3SAT$
**Flashback:** \( SAT \) is Poly Time Reducible to \( 3SAT \)

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

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

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

Remaining step: show iff relation holds ...

... easy for formula conversion: each step is already a known “law”
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!
Next Time: 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 $B$ to $C$
Check-in Quiz 5/2

On gradescope