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

Tuesday, December 13, 2022
Announcements

• Last lecture!

• HW 11 in
  • Due Monday 12/12 11:59pm

• HW 12 out (last HW!)
  • Due Monday 12/19 11:59pm
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 $B$.

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

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

*But if we have one, then (poly time) **mapping reducibility** can help prove other NP-Complete problems!*

**THEOREM**

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

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 difficult 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.

DEFINITION

A language $B$ is $\mathbf{NP}$-complete if it satisfies two conditions:

1. $B$ is in $\mathbf{NP}$, and
2. every $A$ in $\mathbf{NP}$ is polynomial time reducible to $B$.\footnote{This definition is slightly different from the one in the original paper.}
Reducing every **NP** language to **SAT**

Some **NP** lang = \{w \mid w \text{ is } ???\}

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

How can we convert a string \(w\) 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 $\textbf{NP}$ languages?

They are:

1. **Verified** by a deterministic poly time verifier

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

- Computation can branch
- Each node in the tree represents a TM configuration
- Transitions specify valid configuration sequences

---

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 \(q_{\text{reject}} \neq q_{\text{accept}}\).
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

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

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

• A tableau cell has coordinate $i,j$

• A cell contains: state, tape char, or #
  $s \in C = Q \cup \Gamma \cup \{\#\}$

---

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. $\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$ 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 contains: state, tape symbol \(s \in C = Q \cup \Gamma \cup \{\#\}\).

- For every \(i,j,s\) create variable \(x_{ij,s}\), i.e., one var for every possible cell coordinate/content combination.

- Total variables =
  - \(\#\) cells \(\times \#\) 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 \(\Leftrightarrow\) satisfying assignment

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

⇒ 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 = \text{FALSE}

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

⇒ accepting tableau: \textbf{all four} must be \text{TRUE}
⇐ nonaccepting tableau: \textbf{one} must be \text{FALSE}
The variables in the start config, ANDed together

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

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

\( \Rightarrow \) 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

\( \Leftarrow \) Does a non-accepting tableau correspond to an unsatisfiable formula?
  * Not necessarily
$\phi_{\text{accept}} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{\text{accept}}}$

The state $q_{\text{accept}}$ must appear in some cell

i.e., tableau has valid accept config

⇒ 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?
  • Yes, because it won’t have $q_{\text{accept}}$
• Ensures that every configuration is legal according to the previous configuration and the TM’s $\delta$ transitions

• Only need to verify every $2 \times 3$ “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?
\[ \phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \left( \text{the } (i, j)\text{-window is legal} \right) \]

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

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

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

is a legal window

⇒ 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 = $O(n^{2k})$
Time complexity of the reduction

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

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

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

The variables in the start config, ANDed together
Time complexity of the reduction

• Number of cells = $O(n^{2k})$

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

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

$$
\phi_{accept} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{accept}} \quad \text{The state } q_{accept} \text{ must appear in some cell} \quad O(n^{2k})
$$
Time complexity of the reduction

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

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

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

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

$$
\phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \text{(the } (i,j)-\text{window is legal)} \quad O(n^{2k})
$$
Time complexity of the reduction

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

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

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

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

\[
\phi_{\text{move}} = \bigwedge_{1 \leq i < n^k, 1 < j < n^k} \text{(the (i, j)-window is legal)} \quad O(n^{2k})
\]
To Show Poly Time Mapping Reducibility ...

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 $B$.

\begin{itemize}
  \item[\checkmark] To show poly time mapping reducibility:
  \begin{itemize}
    \item[\checkmark] 1. create \textit{computable fn},
    \item[\checkmark] 2. show that it \textit{runs in poly time},
    \item[\checkmark] 3. then show \textbf{forward direction} of mapping red.,
    \item[\checkmark] 4. and \textbf{reverse direction}
    \item[\checkmark] (or \textit{contrapositive of forward direction})
  \end{itemize}
\end{itemize}
**QED:** \( SAT \) is NP-complete

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

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

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

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

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

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 $B$. If you’re not Stephen Cook or Leonid Levin, use this theorem to prove a language is NP-complete.

To use this theorem, $C$ must be in NP.
Theorem: 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: 3SAT 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:
- Accept if $c$ satisfies $\phi$

Running Time: $O(n)$

Non-deterministic Decider:
On input $\langle \phi \rangle$, where $\phi$ is a boolean formula:
- Non-deterministically try all possible assignments in parallel
- Accept if any satisfy $\phi$

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

**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
- [x] 2. Choose $B$, the NP-complete problem to reduce from: SAT
- [x] 3. Show a poly time mapping reduction from SAT to 3SAT
Flashback: SAT is Poly Time Reducible to 3SAT

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

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
      \[ \neg (P \lor Q) \iff (\neg P) \land (\neg Q) \quad \neg (P \land Q) \iff (\neg P) \lor (\neg Q) \quad O(n) \]
   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
   \[ (a_1 \lor a_2 \lor a_3 \lor a_4) \iff (a_1 \lor a_2 \lor z) \land (\neg z \lor a_3 \lor a_4) \quad O(n) \]

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
**THEOREM**

Using: If \( B \) is \( \text{NP} \)-complete and \( B \leq \text{P} \) \( C \) for \( C \) in \( \text{NP} \), then \( C \) is \( \text{NP} \)-complete.

3 steps to prove a language is \( \text{NP} \)-complete:

1. Show \( C \) is in \( \text{NP} \)
2. Choose \( B \), the \( \text{NP} \)-complete problem to reduce from
3. Show a poly time mapping reduction from \( B \) to \( C \)

**Example:**

Let \( C = 3\text{SAT-CLIQUE} \), to prove \( 3\text{SAT-CLIQUE} \) is \( \text{NP} \)-Complete:

1. Show \( 3\text{SAT-CLIQUE} \) is in \( \text{NP} \)
2. Choose \( B \), the \( \text{NP} \)-complete problem to reduce from: \( \text{SAT-3SAT} \)
3. Show a poly time mapping reduction from \( 3\text{SAT} \) to \( 3\text{SAT-CLIQUE} \)
Flashback: 

**CLIQUE is in NP**

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

**Proof Idea**

The clique is the certificate.

**Proof**

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

\[ V = \text{"On input } \langle \langle G, k \rangle, c \rangle \text{:
\begin{enumerate}
\item Test whether } c \text{ is a subgraph with } k \text{ nodes in } G.
\item Test whether } G \text{ contains all edges connecting nodes in } c.
\item If both pass, accept; otherwise, reject."
\end{enumerate}} \]

- Let \( n = \# \text{ nodes in } G \)
- \( c \) is at most \( n \)
- For each node in \( c \), check whether it’s in \( G \): \( O(n^2) \)
- 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:

- ✔ 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

• $SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}$ (Cook-Levin Theorem)

• $3SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable 3cnf-formula} \}$ (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!
Check-in Quiz 12/13

On gradescope

Thank you for a great semester!