CSCI 1010
Models of Computation

Lecture 11
Proving Languages NP-Complete
Overview

- P-time reductions
- Composition of P-time reductions
- Reduction from CIRCUIT SAT to SAT
- SAT is NP-complete.
- 3-SAT is NP-complete.
- NAESAT is NP-complete.
- Reduction from 3-SAT to INDEPENDENT SET
- Reduction from NAESAT to 3-COLORING
Polynomial-Time Reduction

• Definition: A polynomial-time reduction (P-time) from language $L_1 \subseteq \Gamma^*$ to language $L_2 \subseteq \Sigma^*$ is a reduction $f : \Gamma^* \rightarrow \Sigma^*$ ($x \in L_1$ iff $f(x) \in L_2$) computable by a DTM in time polynomial in the length of its input. ($f$ P-time translates $L_1$ to $L_2$)

The Recognizer for $L_1$ invokes the recognizer $R$ for $L_2$. 

© John E. Savage
NP-Complete Language

• L ⊆ Γ* is **NP-complete** if L ∈ NP and L is NP-hard.
• L is NP-hard if for every L₀ ⊆ Γ* in NP, we can decide membership in L₀ by invoking a P-time reduction f₀ from L₀ to L.
• If L is NP-complete and L in P, then P = NP.

Recognizer for L₀ in NP invokes recognizer R for L.
Cook and Karp Reductions

• A Karp reduction makes one invocation of a P-time reduction from one language to another.
  – Such reductions are called translations.

• A Cook reduction uses a P-time oracle Turing machine to decide membership in a language.

• An oracle TM runs, writes a string on the oracle tape, and enters the oracle state. The oracle writes an answer on the oracle tape in one step and the TM resumes. This can be repeated multiple times.
Cook Versus Karp Reductions

• The Cook reduction may be more general than the Karp reduction although every reduction so far is Karp reduction.
Composition of P-Time Translations

• **Definition:** Let $f_1 : \Gamma^* \rightarrow \Sigma^*$ and $f_2 : \Gamma^* \rightarrow \Sigma^*$ be translations. The composition of $f_2$ with $f_1$ computes $f_2(f_1(x))$.

• **Theorem** The composition of two P-time translations is a P-time translation.

• **Proof** On input $x$, DTM computing $f_1$ generates output $f_1(x)$ whose length, $|f_1(x)|$, is polynomial in $|x|$. This is the input to $f_2$ so its running time is polynomial in $|f_1(x)|$ which is polynomial in $|x|$, the input to the composition.
Proving New Languages \textbf{NP}-Complete

- **Theorem** Let $L_a$ be \textbf{NP}-complete. Let $g$ be a P-time translation of $L_a$ to $L_b$ and let $L_b$ be in \textbf{NP}. Then, $L_b$ is \textbf{NP}-complete.

- **Proof** Because $L_a$ is \textbf{NP}-complete, for each $L_0$ in \textbf{NP} there exists an $f_0$ translating $L_0$ to $L_a$ in P-time. The composition of $g$ with $f_0$ is a P-time translation. Thus, for each $L_0$ in \textbf{NP} there exists a P-time $f(x) = g(f_0(x))$ translating $L_0$ to $L_b$. ($L_b$ is \textbf{NP}-hard.) Since $L_b$ is also in \textbf{NP}, it is \textbf{NP}-complete.
Circuit Satisfiability

• A circuit is **satisfiable** if its nondeterministic inputs can be chosen so that the output is 1.

• **CIRCUIT SAT** is the set of satisfiable circuits.

• **CIRCUIT SAT** has been shown **NP**-complete.

• We reduce it in P-time to other problems to show that they are also **NP**-hard.
Satisfiability (SAT)

• **Instance:** A set of literals $X = \{x_1, \bar{x}_1, ..., x_n, \bar{x}_n \}$ and clauses $C = (c_1, ..., c_n)$, each $c_i$ is a subset of $X$.

• **“Yes” Instance:** There exists an assignment to variables over $\{0,1\}$ such that each clause has a literal with value 1.

• **“No” Instance:** There is no assignment to variables over $\{0,1\}$ such that each clause has a literal with value 1.

• We show that **SAT** is **NP**-complete.
Translating CIRCUIT SAT to SAT

• Consider circuit in SLP form:
  • (1 READ x) \( g_1 := x; \)
  • (2 READ y) \( g_2 := y; \)
  • (3 NOT 1) \( g_3 := \text{NOT}(g_1); \)
  • (4 NOT 2) \( g_4 := \text{NOT}(g_2); \)
  • (5 AND 1 4) \( g_5 := g_1 \text{ AND } g_4; \)
  • (6 AND 3 2) \( g_6 := g_3 \text{ AND } g_2; \)
  • (7 OR 5 6) \( g_7 := g_5 \text{ OR } g_6; \) (\( g_7 \) is the output)
Translating CIRCUIT SAT to SAT

- These clauses simulate gates. \( \neg \) denotes NOT. That is, the clauses are all satisfied only when the function on the left is computed.

<table>
<thead>
<tr>
<th>Gate Type</th>
<th>Corresponding Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g = x ):</td>
<td>((\neg g \lor x))</td>
</tr>
<tr>
<td>( g = \text{NOT of } h ):</td>
<td>((\neg g \lor \neg h))</td>
</tr>
<tr>
<td>( g = \text{OR of } h \text{ and } k ):</td>
<td>((g \lor \neg h))</td>
</tr>
<tr>
<td>( g = \text{AND of } h \text{ and } k ):</td>
<td>((\neg g \lor h))</td>
</tr>
<tr>
<td>( g = \text{output} ):</td>
<td>((g))</td>
</tr>
</tbody>
</table>
Translating **CIRCUIT SAT to SAT**

- Tables showing why these mappings work.

<table>
<thead>
<tr>
<th>g</th>
<th>x</th>
<th>Sat?</th>
<th>$\neg g \lor x$</th>
<th>$g \lor \neg x$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>No</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>No</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Both clauses satisfied when $g = x$.

<table>
<thead>
<tr>
<th>g</th>
<th>h</th>
<th>k</th>
<th>Sat?</th>
<th>$\neg g \lor h$</th>
<th>$\neg g \lor k$</th>
<th>$g \lor \neg h \lor \neg k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>No</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>No</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>No</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

All three clauses satisfied when $g = h \land k$

Note that not all literals in the third clause are True when all clauses satisfied.
Translating \textbf{CIRCUIT SAT} to \textbf{SAT}

\begin{itemize}
\item $g_1 := x$; \hspace{1cm} ($\bar{g}_1 \lor x$) ($g_1 \lor \bar{x}$)
\item $g_2 := y$; \hspace{1cm} ($\bar{g}_2 \lor y$) ($g_2 \lor \bar{y}$)
\item $g_3 := \text{NOT}(g_1)$; \hspace{1cm} ($\bar{g}_3 \lor \bar{g}_1$) ($g_3 \lor g_1$)
\item $g_4 := \text{NOT}(g_2)$; \hspace{1cm} ($\bar{g}_4 \lor \bar{g}_2$) ($g_4 \lor g_2$)
\item $g_5 := g_1 \text{ AND } g_4$; \hspace{1cm} ($\bar{g}_5 \lor g_1$) ($\bar{g}_5 \lor g_4$) ($g_5 \lor \bar{g}_1 \lor \bar{g}_4$)
\item $g_6 := g_3 \text{ AND } g_2$; \hspace{1cm} ($\bar{g}_6 \lor g_3$) ($\bar{g}_6 \lor g_2$) ($g_6 \lor \bar{g}_3 \lor \bar{g}_2$)
\item $g_7 := g_5 \text{ OR } g_6$; \hspace{1cm} ($g_7 \lor \bar{g}_5$) ($g_7 \lor \bar{g}_6$) ($\bar{g}_7 \lor g_5 \lor g_6$)
\item $(g_7 \text{ is the output}) (g_7)$ \hspace{1cm} Form AND of all the clauses
\end{itemize}
SAT is NP-complete

• Theorem SAT is NP-complete.
• Proof The above reduction from CIRCUIT SAT to SAT maps an instance of CIRCUIT SAT to a instance of SAT. The former is satisfied if and only if the latter is satisfied. Why?
• The reduction takes time polynomial in the length of the input. Thus, SAT is NP-hard.
• Why is SAT is in NP?
  It follows that SAT is NP-complete.
3-SAT

- **Instance**: Set of literals $X = \{x_1, \bar{x}_1, ..., x_n, \bar{x}_n\}$ & clauses $C = (c_1, ..., c_n)$, each $c_i$ is a subset of $X$, $|c_i| \leq 3$.

- “Yes” **Instance**: There exists an assignment to variables over $\{0,1\}$ such that each clause has a literal with value 1.

- “No” **Instance**: There is no assignment to variables over $\{0,1\}$ such that each clause has a literal with value 1.

- The same proof shows that **3-SAT** is **NP**-complete because each clause has at most three literals.
Not All Equal SAT (NAESAT)

• NAESAT

• Instance: An instance of 3-SAT

• “Yes” Instance: Each clause is satisfied with not all literals being equal (one has value 1, another has value 0).

• “No” Instance: Cannot satisfy each clause with not all literals being equal.
On Reductions

• To show that NAESAT is NP-complete, we make the following observation:

• Let $f$ be a translation of problem A to problem B, that is, that it maps a “Yes” instance of A to one of B and a “No” instance of A to one of B.

• Instead of showing that $f$ translates a “No” instance of A to a “No instance” of B, it suffices to show that the pre-image of $f$ from a “Yes” instance of B is a “Yes” instance of A. Then, a “No” instance of A does not map to a “Yes” of B. See the following.
Proving Reductions

• If we show that every element mapping to $\text{Yes}_B$ is in $\text{Yes}_A$, then every element in $\text{No}_A$ must map to an instance in $\text{No}_B$. We use this idea to show several problems NP-complete.
NAESAT is NP-Complete

• Theorem NAESAT is NP-complete.
• Proof The 3-literal clauses in the reduction to 3-SAT have the requisite property (check this).
• Add a new literal $y$ to the other clauses.
• We claim that this reduction is satisfied with not all equal literals iff CIRCUIT SAT is satisfied.
  a) If CIRCUIT SAT is satisfied, at least one literal is 1 in each clause. Set $y = 0$!
NAESAT is NP-Complete

• Proof (cont.)
  b) If the instance of NAESAT produced by the reduction is a Yes instance, then each clause is satisfied with the NAE property. Thus, if we complement all variables, the same property holds. In one of these cases \( y = 0 \). Choose this case. Then, the values of variables is such as to cause the original instance of CIRCUIT SAT to be satisfied.

• Thus, the instance CIRCUIT SAT is satisfied iff the instance of NAESAT is satisfied.
Independent Set (IS)

• *Instance*: A graph $G = (V,E)$ and integer $k$.
• “*Yes*” *Instance*: $(G,k)$ such that there is a set of $k$ vertices with no edges between them.

• Let $G$ be the following graph. Is $(G,3)$ in *Independent Set (IS)*? If so, why? Is $(G,4)$ in IS?
Theorem Independent Set is NP-complete.

Proof We reduce 3-SAT to Independent Set. Modify 3-SAT so that each clause contains three literals and no clause identically true. E.g. if \((a+b)\) exists, add \((a+b+z)\) and \((a+b+\overline{z})\), \(z\) is new. \((a+b)\) is satisfied iff \((a+b+z)\) \((a+b+\overline{z})\) is satisfied.

If a clause contains one variable, create four clauses with all four combinations of literals in new variables \(u\) and \(v\). This is a P-time reduction.
Independent Set in NP-Complete

• **Proof (cont.)** Given an instance of **3-SAT**, construct graph G by creating a triangle for each clause. The vertices of a triangle are labeled by the literals in a corresponding clause.

• Add edge between vertices in different clauses if one is the NOT of another. Let $k =$ number of clauses. (G,k) an instance of IS. This reduction is P-time.
Example of Reduction

- Example: \((x_1 + \overline{x}_2 + x_3)(\overline{x}_1 + x_2 + \overline{x}_3)(\overline{x}_1 + \overline{x}_2 + x_3)\) maps to \((G,3)\) below. It is satisfied by \(x_1 = x_2 = x_3 = 1\).
Independent Set in NP-Complete

• Given a Yes instance of 3-SAT, we show that (G,k) is a Yes instance of IS.
  – Pick one True literal per clause. No edges exist between corresponding vertices in (G,k) so they form a Yes instance of IS.
Independent Set in NP-Complete

- Given a Yes instance \((G,k)\) of IS, we construct a Yes instance of 3-SAT.
  - IS of \((G,k)\) must have one vertex from each triangle. Since labels are not complements, assign value 1 to corresponding literals to obtain a Yes instance of 3-SAT.
3-COLORING

• Instance: A graph $G = (V,E)$.
• “Yes” Instance: $G$ such that three colors can be assigned to vertices so that adjacent vertices have different colors.
• Theorem 3-COLORING is NP-complete.
• Proof By reduction from NAESAT. Consider an instance of NAESAT that has three literals in each clause.
3-COLORING

• Construct a graph $G$ from clauses as follows:
  – Create a variable triangle $(v, x_j, \overline{x}_j)$ for each $j$.
  – Create a triangle for each clause. If the $r$th clause has literals $\lambda_1, \lambda_2, \lambda_3$ let $(r, \lambda_1), (r, \lambda_1), (r, \lambda_1)$ be labels for the three vertices in the $r$th triangle.
  – Insert edge between $(r, \lambda_i)$ in a clause triangle and $\lambda_i$ in a variable triangle.
3-COLORING

Variable Triangle

Clause Triangle

© John E. Savage
CSCI 1010 Lect 11
3-COLORING

• Given a Yes instance of 3-COLORING, we show it corresponds to a Yes instance of NAESAT.

• Let colors be \(\{0,1,2\}\). Let the root \(\nu\) have color 2.

• Assign colors in \(\{0,1\}\) to \((r, \lambda_i)\) and \(\lambda_i\). To the third vertex in clause triangle, assign color 2

• If \((r, \lambda_i)\) has color \(c\) in \(\{0,1\}\), assign value \(c\) to \(\lambda_i\).

This means that in each clause not all literals have the same value and we have Yes instance of NAESAT.
**3-COLORING**

- Given a Yes instance of NAESAT, we show it corresponds to a Yes instance of 3-COLORING.
- Let $\nu$ have color 2 and $x_j$ and $\overline{x}_j$ have values that satisfy the clauses in the Yes instance.
- Consider two unequal literals in a clause. If $(r, \tilde{x}_j)$ is one of these, give it a complementary color in $\{0,1\}$ to $\tilde{x}_j$ in the variable triangle. Give the third vertex in clause triangle color 2. This is a 3-COLORING.
Review

• P-time reductions
• Composition of P-time reductions
• Reduction from CIRCUIT SAT to SAT
• SAT is NP-complete.
• 3-SAT is NP-complete.
• NAESAT is NP-complete.
• Reduction from 3-SAT to INDEPENDENT SET
• Reduction from NAESAT to 3-COLORING