1 Polynomial time reductions and NP-completeness

Complete problems capture a whole class inside a single problem. They are as hard as the hardest problem in the class (up to reduction).

**Definition 1 (complete problems).** Let $\mathcal{C}$ be a class of problems, and $\leq$ be a type of reduction. A language $L$ is $\mathcal{C}$-complete (under $\leq$) if it satisfies the two conditions:

1. $L \in \mathcal{C}$
2. $\forall L' \in \mathcal{C}, L' \leq L$.

**Example.** $\text{Halt}$ is $\text{R.E.}$-complete under $\leq$.

**Example.** $\text{BTBUP}$ (Binary time-bounded universal problem) is $\text{EXP}$-complete for $\leq_{\text{pm}}$. (We implicitly gave a polynomial-time mapping reduction in previous lectures.)

To preserve complexity of polynomial time, the running time of the reduction algorithm, and the size of instance mapped, should be polynomial to the input size. (For $\text{EXP}$, we allow exponential time reductions, but should be careful not to blow up the instance size exponentially.)

**Definition 2.** A polynomial-time mapping reduction (aka. Karp reduction) from language $L$ to language $L'$ is a polynomial-time computable function $f$ that maps each instance $x'$ of $L'$ to an instance $x = f(x')$ of $L$, so that $x' \in L' \iff x \in L$.

**Lemma 3.** (Properties of polynomial-time reductions)

1. (Transitivity) If $L_1 \leq_{pm} L_2$ and $L_2 \leq_{pm} L_3$, then $L_1 \leq_{pm} L_3$.
2. If $L_1 \leq_{pm} L_2$ and $L_2 \in \text{P}$, then $L_1 \in \text{P}$.
3. If $L_1 \leq_{pm} L_2$ and $L_2 \in \text{NP}$, then $L_1 \in \text{NP}$.

**Corollary 4.** Say $L_1$ is NP-complete, $L_2 \in \text{NP}$, and $L_1 \leq_{pm} L_2$, then $L_2$ is NP-complete.

Proof Sketch: $\forall L' \in \text{NP}, L' \leq_{pm} L_1, L_1 \leq_{pm} L_2$, thus $\forall L' \in \text{NP}, L' \leq_{pm} L_2$. 

1
2 How to prove of \( \text{NP} \)-completeness

To show a reduction from \( L' \) to \( L \), we’d better consider a function that not only maps instances to instances, but also maps solutions to solutions. i.e.

\[
\text{(instance } x', \text{solution } y', R'(x', y')) \xrightarrow{f} \text{(instance } x, \text{solution } y, R(x, y)).
\]

Thus, we need to show: \((x' \in L' \iff \exists y' R'(x', y')) \iff (x \in L \iff \exists y R(x, y))\)

How to prove this? If we find a \( y \), from mapping \( g \) we get a solution \( y' \). And in the other direction, from a solution \( y' \), a mapping \( h(y') \) gives \( y \). (the latter direction is important in the proof, but often missed by people).

Here is an outline of what an \( \text{NP} \)-completeness solution should look like.

### Proof Template

- \( L_1 \) is a known \( \text{NP} \)-complete problem.
- Objective: prove \( L_2 \) is \( \text{NP} \)-complete.
  
  1. Show \( L_2 \) is in \( \text{NP} \).
     - Say what a solution or witness \( y \) is.
     - Show that the length of witness in bounded by \( \text{poly}(n) \).
     - Show that verifying witness is in polynomial time.
     
     (Most of the time, the three steps are trivial (e.g. for graph 3-coloring, the witness is the coloring.), but they are imperative in a \( \text{NP} \)-completeness proof.)
  
  2. Give the actual function for construct \( x_2 \) from \( x_1 \).
  
  3. Define a map \( g \) from solution \( y_2 \) for \( x_2 \) (under \( R_2 \)) to solution \( y_1 \) for \( x_1 \).
     - Assume \( R_2(x_2, y_2) \), prove \( R_1(x_1, y_1) \).
  
  4. Define a map \( h \) from solution \( y_1 \) for \( x_1 \) (under \( R_1 \)) to solution \( y_2 \) for \( x_2 \).
     - Assume \( R_1(x_1, y_1) \), prove \( R_2(x_2, y_2) \).

3 Satisfiability is \( \text{NP} \)-complete

**Definition 5.** A *Boolean circuit* consists of

- Input gates \( g_1 = x_1, \ldots, g_n = x_n \)
- For \( i \in \{n_1, \ldots, m\} \), \((i \text{ is the size of the circuit.})\) gate \( g_i = \text{op}_i(g_j, g_k), 1 \leq j, k < i \).
- Operation \( \text{op}_i \in \{0, 1, \neg g_j, g_j \land g_k, g_j \lor g_k \} \)
Example.

\[ g_1 = x_1 \]
\[ g_2 = x_2 \]
\[ g_3 = x_3 \]
\[ g_4 = g_1 \lor g_2 \]
\[ g_5 = \neg g_3 \]
\[ g_6 = g_4 \land g_5 \]

The circuit defined above, with \( g_6 \) as an output gate, computes \((x_1 \lor x_2) \land \neg x_3\).

The circuit model is different the computation models we introduced before. For each size of input, there can be a different circuit. Each fixed size of input has its own algorithms rather than a finitely described algorithm for all sizes. This is called a \textit{non-uniform} computation model.

A \textit{circuit family} \((C_0, C_1, \ldots, C_n, \ldots)\) is \(P\)-\textit{uniform} if there is an algorithm that given \(n\), outputs \(C_n\) in time \(\text{poly}(n)\).

**Theorem 6.** \(L \in P\) iff there is a \(P\)-uniform circuit family for \(L\).

<table>
<thead>
<tr>
<th>Problem: Circuit-SAT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> Boolean circuit ('\mathcal{C}') taking (x_1, \ldots, x_n) as input</td>
</tr>
<tr>
<td><strong>Solution:</strong> (y_1, \ldots, y_n \in {0, 1})</td>
</tr>
<tr>
<td><strong>Constraint:</strong> ('\mathcal{C}(y_1, \ldots, y_n)')</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem: CNF-SAT (or SAT)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance:</strong> CNF(conjunctive normal form) ('\mathcal{C}') on variables (x_1, \ldots, x_n). i.e. (\mathcal{C} = \bigwedge_{i=1}^{m} \left( \bigvee_{j} \ell_{i,j} \right)) where literal (\ell_{i,j}) is either (x_k) or (\overline{x_k}), (1 \leq k \leq n).</td>
</tr>
<tr>
<td><strong>Solution:</strong> (y_1, \ldots, y_n \in {0, 1})</td>
</tr>
<tr>
<td><strong>Constraint:</strong> ('\mathcal{C}(y_1, \ldots, y_n)')</td>
</tr>
<tr>
<td><strong>Example:</strong> ((x_1 \lor x_2 \lor \overline{x_3}) \land (x_1 \lor x_4) \land (x_5 \lor \overline{x_1} \lor x_2))</td>
</tr>
</tbody>
</table>

We define problem \(k\)-SAT as SAT problem where each clause has size at most \(k\).

It is known that 2-SAT is polynomial-time decidable, but 3-SAT is \(NP\)-complete.

Here we prove that CNF-SAT is \(NP\)-complete. Because it is a special case of Circuit-SAT, Circuit-SAT is also \(NP\)-complete.

**Proof that CNF-SAT is \(NP\)-complete**

Step 1: CNF-SAT is in \(NP\). The witness of a satisfiable CNF is its satisfying assignment. It is polynomial in length and its checkable in polynomial time.
Step 2: Next we construct a reduction algorithm.

Let \( L \) be an arbitrary language in \( \text{NP} \), and \( N \) be a 1-tape NTM that recognizes \( L \) in time \( T(n) \).

Recall that the tableau of a TM is a matrix of symbols containing the contents of the tape at each time. For an NTM, it has a set of possible tableaus, rather than a single tableau like in DTM.

For a \( \text{NP} \) language \( L \), \( x \in L \) iff there exists a tableau where \( N \) accepts.

Set variable \( x_{i,j,\sigma} = 1 \), if \( i, j \)th cell of tableau is \( \sigma \), and 0 otherwise.

We construct the clauses of CNF as follows.

1. There is exactly one symbol in each cell in the tableau: \( \bigvee_{\sigma \in \bar{\Gamma}} x_{i,j,\sigma} \)
2. The first row is correct.
3. The last line should have a symbol for an accepting state.
4. Every single step is correct. For each \( 2 \times 3 \) rectangle, and any invalid sub-matrix, write the clause "that sub-matrix is not there".

Example: Suppose sub-matrix

\[
\begin{array}{ccc}
  a & b & c \\
  d & e & f \\
\end{array}
\]

is not allowed in the tableau, then we write clause

\[
x_{i,j,a} \lor x_{i,j+1,b} \lor x_{i,j+2,c} \lor x_{i+1,j,d} \lor x_{i+1,j+1,e} \lor x_{i+1,j+2,f}
\]

The CNF is big, but its size is still polynomial to \( n \).

Step 3: If there is a satisfying assignment to this CNF, each \( x_{i,j,\sigma} \) is true for exactly one \( \sigma_{i,j} = \sigma \) and these \( \sigma_{i,j} \) form a tableau on an accepting computation.

Step 4: If \( \sigma_{i,j} \) is a symbol from an accepting tableau for \( N \), we just define \( x_{i,j,\sigma} = 1 \) iff \( \sigma = \sigma_{i,j} \).