NP-completeness III

For any language L_0, if L_0 is *NP*-complete and $L_0 \in P$, then $P = NP$. For example, since SKD is *NP*-complete, if SKD is in P, then $P = NP$.

Lemma 1 (*NP*-completeness) If L_0 is *NP*-complete, and $L_1 \in NP$, and $L_0 \leq_p L_1$, then L_1 is *NP*-complete.

Thus, to show that a problem is *NP*-complete, we need to show that a previously known *NP*-complete problem is reducible to it. *Important*: Make sure that you get the reduction in the right direction!

Lemma 2 (Transitivity) If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$, then $L_1 \leq_p L_3$.

Proof of the Transitivity Lemma: Suppose $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, $L_3 \subseteq \Sigma_3^*$. Assume that $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$. Then:

$x \in L_1$ iff $f(x) \in L_2$
$y \in L_2$ iff $g(y) \in L_3$

$\forall x \in \Sigma_1^*$, $\forall y \in \Sigma_2^*$, where f and g are polytime functions. We want to show $L_1 \leq_p L_3$, i.e. we need to find a polytime function $h(x)$ such that $x \in L_1$ iff $h(x) \in L_3$ $\forall x \in \Sigma_1^*$. This property holds if we define $h(x) = g(f(x))$. We need to show that h (defined this way) can be computed in polytime.

Lemma 3 (Composition) Suppose $f : \Sigma_1^* \rightarrow \Sigma_2^*$ and $g : \Sigma_2^* \rightarrow \Sigma_3^*$ are two polytime functions and $h(x) = g(f(x))$. Then $h(x)$ is also polytime.

Exercise: Prove the Composition Lemma.

Proof of Lemma 1. Assume that $L_0 \leq_p L_1$ and L_0 is *NP*-complete, and $L_1 \in NP$. To show that L_1 is *NP*-complete, we must show that given any language $L \in NP$, $L \leq_p L_1$. Since L_0 is *NP*-complete, for any language $L \in NP$, $L \leq_p L_0$, and since $L_0 \leq_p L_1$, the result follows by transitivity.

[1] These notes were taken originally by Antonina Kolokolova in Steven Cook's csc364 lecture, Spring 99, and they have been edited for this term.

SAT

SAT is the decision problem for CNF Boolean formula satisfiability, defined as follows:

Start with a set of *Boolean variables* $P, Q, R, \ldots$.

A *literal* is either a variable P or a negated variable $\bar{P}$.

A *clause* is a finite disjunction $L_1 \vee L_2 \vee \ldots \vee L_k$ of distinct literals, $k \geq 0$. If $k = 0$, then the clause is assumed to be empty.

A *truth assignment* to a set of clauses is a map $t : U \rightarrow \{T, F\}$, where U is the set of variables appearing in the set of clauses, and $T =$ True and $F =$ False.

$$t(\bar{P}) = \begin{cases} T & \text{if } t(P) = F \\ F & \text{if } t(P) = T \end{cases}$$

If C is a clause, $C = L_1 \vee L_2 \vee \ldots \vee L_k$, t *satisfies* C if it makes it true, i.e. $t(L_i) = T$ for some $1 \leq i \leq k$.

Let $S = \{C_1, C_2, \ldots, C_m\}$ be set of clauses. Then t satisfies S iff t satisfies every clause in S, i.e. t *satisfies* $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ (CNF formula).

Decision problem SAT:

Input: finite set $\{C_1, \ldots, C_m\}$ of clauses.

Question: Is S satisfiable? That is, is there a truth assignment t which satisfies S?

For example, $S = \{P \vee \bar{Q} \vee R, \bar{P}, \bar{Q} \vee \bar{R}, Q \vee R\}$. S is satisfiable and a satisfying assignment is $t(P) = F$, $t(Q) = F$, $t(R) = T$.

Theorem 1 (Cook, Levin 1971) SAT is *NP*-complete.

It is easy to show that SAT is in *NP*:

Given $S = \{C_1, \ldots, C_m\}$, let U be the set of variables in S. Guess a truth assignment t. Check that t satisfies S. It is possible to decide in polynomial time whether t satisfies S, given both S and t.

Note: If a set S has u different variables, then there are 2^u different truth assignments.

The difficult part is demonstrating that for every language $L \in NP$, $L \leq_p SAT$. We are not going to prove this — check Garey and Johnson if you are interested.

3SAT:

Input: Set $S = \{C_1, \ldots, C_m\}$ of clauses, such that every clause has exactly 3 literals.

Question: Is S satisfiable?

Theorem 2 3SAT is NPC

Proof: $3SAT \in NP$. Use the same guess and check algorithm as for SAT (and check that input has the right format).

To show 3SAT is NPC, use the NPC lemma and the fact that SAT is NPC. We must show that SAT $\leq_p$ 3SAT.

Given an instance (input) $S = \{C_1, \ldots, C_m\}$ to SAT, we must compute (in polytime) an input $S' = \{C'_1, \ldots, C'_{m'}\}$ to 3SAT, such that S is satisfiable iff S' is satisfiable. We need to "shrink" the existing clauses and introduce more clauses. For that, we introduce new variables.

Suppose $C_1 = L_1 \vee L_2 \vee L_3 \vee \ldots \vee L_k$, where $k \geq 4$. Let P be a new variable (P does not occur in S). Make P equivalent to $L_1 \vee L_2$ by introducing clauses, i.e. introduce 3 new clauses:

$$\begin{cases} D_1 = \bar{P} \vee L_1 \vee L_2 \\ D_2 = P \vee \bar{L_1} \\ D_3 = P \vee \bar{L_2} \end{cases}$$

These three clauses assert that $P \equiv L_1 \vee L_2$. Let $C'_1 = P \vee L_3 \vee \ldots \vee L_k$ (C_1 shrunk by 1).

Claim 1 The new set of clauses $S' = \{D_1, D_2, D_3, C'_1, C_2, \ldots, C_m\}$ is satisfiable iff S is satisfiable.

Proof of claim: If t satisfies S', then t also satisfies S, because $t(P) = t(L_1 \vee L_2)$, since t satisfies D_1, D_2, D_3.

Conversely, suppose t satisfies S. Is S' satisfiable? Let t' be the same as t, except let $t'(P) = t(L_1 \vee L_2)$. Then t' satisfies $C_2, \ldots, C_m$ because P does not occur in these clauses. Also, t' satisfies D_1, D_2 and D_3, and t' satisfies $C'_1 = P \vee L_3 \vee \ldots \vee L_k$, because t satisfies $C_1 = L_1 \vee L_2 \vee L_3 \vee \ldots \vee L_k$.

We can continue doing this $k - 3$ times, until C_1 is reduced to 3 literals. Now proceed to reduce C_2 in the same way.

This can be computed in polytime. The resulting S' will be satisfiable iff the original S is satisfiable.

Given $L_1 \vee L_2$, expand it to $\{L_1 \vee L_2 \vee P, L_1 \vee L_2 \vee \bar{P}\}$, where P is any variable, not necessarily new. Then t satisfies $L_1 \vee L_2$ iff t satisfies both ($L_1 \vee L_2 \vee P$ and $L_1 \vee L_2 \vee \bar{P}$).

Exercise: What do you do in the case that $C = L_1$, i.e. C is a clause consisting of a single literal?

Let BOOL be:
Input: Boolean formula B over $\wedge, \vee, \neg$, e.g. $(P \wedge \neg(Q \vee R) \wedge \neg(S \vee \neg T)) \vee \neg P \wedge Q$.
Question: Is B satisfiable?

It easy to see that SAT $\leq_p$ BOOL, since in fact SAT is essentially a special case of BOOL.

Exercise: Show: BOOL $\leq_p$ SAT (*Hint*: there is an easy way to do this using the fact that SAT is *NP*-complete, and there is a hard way of doing this by reduction. You should do it both ways. Note that in the reduction, we cannot use a direct conversion of B to a CNF formula, since the size of a CNF formula can be exponential in the size of the original formula. Instead, you should introduce new variables corresponding to each sub-formula of B.)