CS154: Introduction to Automata and Complexity Theory **Fall 1997**

Final Exam Solutions

PROBLEM 1 **Prove that $L_{\text{SIZE}} = \{\langle M_1\rangle\langle M_2\rangle : |L(M_1)| > |L(M_2)|\}$ is not R.E.**

We reduce from $L_\emptyset$, which is not R.E. Remember, we're given a machine M that we want to know if its language has size 0, and we're given an oracle which accepts L_{SIZE}. So we construct a machine $M_{\text{SINGLETON}}$ that accepts only one word (say "a"). We then pass $\langle M_{\text{SINGLETON}}\rangle\langle M\rangle$ to the oracle. Note that the oracle will accept M iff $L(M) = \emptyset$, since only the empty language has size less than 1. Thus, an acceptor for L_{SIZE} would be an acceptor for $L_\emptyset$, and since the latter is not R.E. the former can't be either.

PROBLEM 2 **(a) Prove that if $P = NP$ then *every* language $L \in NP$ is NP-complete, except $L = \emptyset$ and $L = \Sigma^*$. (b) Let $G = (V, E)$ be a graph. Then a K-MOON-CLIQUE is a subset S of V such that S contains a $k-1$ clique and a vertex not connected to any vertex of the $k-1$ clique. Prove that if $P \neq NP$ then the following language is not in P: $L_{MC} = \{\langle G, K\rangle : G$ is a graph with a K-Moon-Clique$\}$.**

(a) Let L be in NP. To prove that it's NP-complete, we must show that every problem in NP reduces to it in polynomial time. We'll need a string $w_{yes} \in L$ and another string $w_{no} \notin L$. These both exist because $L \neq \emptyset$ or Σ^*.

Remember how reducing L' to L works: you give a string w', and I convert it to another string w such that $w' \in L'$ iff $w \in L$. Usually this requires some cleverness, but if $P = NP$ it's simple: just determine whether $w' \in L'$. You can do this in polynomial time because $L' \in NP = P$. Now, if $w' \in L$, map it to w_{YES}. If $w' \notin L$, map it to w_{NO}. Voila! This is exactly the reduction demanded. Now you can use a fast solver for L to figure out if $w' \in L'$. Note that this is silly, since you can easily figure out if $w' \in L'$ without going through L, but hey, we never said the definition was *useful*. In fact, if $P = NP$ it clearly isn't.

(b) We reduce from CLIQUE. Given a graph G and a min-clique-size k, we transform G to G' merely by adding a new vertex not connected to any vertex in G. We then ask if G' has a $k+1$-moon-clique. If G has a clique of size k, G' clearly has a moon-clique of size $k+1$: the clique in G, plus the added vertex. Furthermore, if G' has a $k+1$ moon clique, then the "sun" part of it is also a k clique in G (since clearly the new vertex can't be part of the sun). Finally, the reduction takes constant time. Thus we have a legal reduction, proving our problem is NP-hard.

Next we show it's NP-complete: you give me the sun and the moon and I verify the sun is really a clique and the moon is not connected to the sun. All this takes time $|V|^2|E|$ at most, so we have a polynomial-time verifier.

Finally we note that since the problem is NP-complete, if it's in P then all of NP is in P. Thus, if $NP \neq P$, K-MOON-CLIQUE is not in P.

PROBLEM 3 **True/False. Give a counterexample or a proof sketch.**

1. All $NP-$complete languages are recursive.
   TRUE. NP-complete languages are in NP, and NP is the set of languages that can be decided by an NTM in polynomial time. Thus these languages are decidable.

2. Suppose $P = NP$. Then any R.E. language is also in P.
FALSE. Undecidable languages like L_{NOTEMPTY} can never be in P regardless of whether $P = NP$, but they can be in R.E.

3. If $L \in P$ then $L \in NTIME(n^k)$ for some k.
TRUE. By definition, if $L \in P$ then $L \in \sum_{i=0}^{\infty} DTIME(n^k)$, which means $L \in DTIME(n^k)$ for some k. But $DTIME(n^k) \in NTIME(n^k)$ (all DTMs are also NTMs), so $L \in NTIME(n^k)$ too.

4. If $f(n) = O(n)$ and $f(n) = \Omega(n)$, then $f(n) = \theta(n)$.
TRUE. $f(n) = O(n)$ means that for all $n > N_0$, $\frac{f(n)}{n} \geq c_1$. $f(n) = \Omega(n)$ means that for all $n > N_1$, $\frac{f(n)}{n} \leq c_2$. Thus, for all $n > \max\{N_0, N_1\}$, $c_2 \leq \frac{f(n)}{n} \leq c_1$, which is the condition for $f(n) = \theta(n)$.

5. If $L_1, L_2 \in NP$ then so is $L_1 \cap L_2$.
TRUE. Let M_1 be the NTM deciding L_1, and M_2 the NTM deciding L_2. Then change the accept state on M_1 so that it replaces the original input and then runs M_2. Clearly the new machine accepts iff there is an accepting computation for both M_1 and M_2, that is, if $L \in L_1 \cap L_2$.

6. If a language L_1 is mapping reducible to L_2, i.e. $L_1 \leq_m L_2$, and L_2 is undecidable then so is L_1.
FALSE. It's backwards. For instance, consider L_{EVEN}, which we will so shortly prove is decidable. We can reduce it to L_{NOTEMPTY} by mapping all even strings to a machine accepting "a" (say) and all odd strings to a machine accepting nothing. Thus we've mapped all strings in L_{EVEN} to strings in L_{NOTEMPTY}, and *vice versa*, making this a valid mapping. But even though L_{NOTEMPTY} is not decidable, L_{EVEN} is.

7. If a language L_1 is polynomial time reducible to L_2, i.e. $L_1 \leq_P L_2$, and L_1 is undecidable then so is L_2.
TRUE. Otherwise we could decide L_1 by reducing it to L_2 and deciding L_2. (btw, the fact the reduction can be done quickly is unimportant in this context.)

8. If a language L_1 is mapping reducible to $L_2 \in NP$, i.e. $L_1 \leq_m L_2$, and L_1 is $NP-$complete then so is L_2. For your proof or counterexample you may assume $P \neq NP$.
FALSE. The reduction might take exponential time. Remember the mapping should take strings in L_1 to strings to strings in L_2, and strings not in L_1 to strings not in L_2. But it can take as long as it wants. Thus, we can reduce L_{SAT} to L_{even} by taking a formula and figuring out, in exponential time, if it's satisfiable. If so, map it to an even number. If not, map it to a hard number. L_{SAT} is NP-complete but L_{EVEN} is not.

PROBLEM 4 **Consider $L_{\text{EVEN}} = \{w : w$ is the binary representation of an even positive integer$\}$. (a) Prove that L_{EVEN} is decidable by describing a TM for deciding it. (b) Prove that the language $L = \{\langle M \rangle : L(M) = L_{\text{EVEN}}\}$ is undecidable.**

(a) $M = (\{q_0, q_1, q_a, q_r\}, \{0,1\}, \{0,1,\sqcup\}, \delta, q_1, q_a, q_r)$, where we have transitions:

$$\begin{aligned}
\delta(q_0, 0) &= (q_0, 0, R) \\
\delta(q_1, 0) &= (q_0, 0, R) \\
\delta(q_0, 1) &= (q_1, 0, R)
\end{aligned}$$

$$\begin{aligned}\delta(q_1, 1) &= (q_1, 0, R)\\ \delta(q_0, \sqcup) &= (q_a, 0, R)\\ \delta(q_1, \sqcup) &= (q_r, 0, R)\end{aligned}$$

The idea is that q_0 means "last digit seen is 0, and q_1 means "last digit seen is 1." When we get to the blank at the end of the word, we accept iff the last digit seen was a 0. By starting in q_1, we don't accept the empty string. Note that this machine treats 0 as a positive integer, which we'll pretend it is. (If that bugs you, add a new start state q_s and make $\delta(q_s, \sigma) = q_1, 0, R)$ for all $\sigma \in \Sigma$. Thus we never accept a string of length 0 or 1.)

(b) Reduce from the halting problem. Given input $\langle M \rangle, w$ to the halting problem, create a new machine M' which first, ignores its own input and instead simulates M on input w. When M would reach q_a or q_r, however, M' runs the TM from part **(a)** on the actual input. Clearly M' accepts L_{EVEN} if it ever gets to looking at its input, which is iff M halts on w. If M does not halt on w, then M' accepts no word, because it never gets around to looking at its input. Thus, an oracle deciding if $L(M') = L_{\text{EVEN}}$ would also decide the halting problem, an impossibility.

PROBLEM 5 **Define $L_{\text{COMPOSITES}} = \{\langle N \rangle : N$ is a composite integer$\}$, where $\langle N \rangle$ is N in binary. (a) Prove that $L_{\text{COMPOSITES}} \in NP$. (b) Consider the following algorithm for $L_{\text{COMPOSITES}}$: try dividing N by all integers less than N. Accept iff one of these integers divides N. This algorithm runs in polynomial time in N. Does it prove that $L_{\text{COMPOSITES}}$ is in P?**

(a) We come up with a polynomial time verifier. The certificate is a proper divisor of N. We check that the certificate actually divides N and is not 1 or N. Division takes polynomial time in the size of the input, so the verifier is a valid NP verifier.

(b) To be in P, the algorithm must run in polynomial time in the size of the *input*. In this case, the input has size $\log_2 N$ since it's in binary. Thus the algorithm runs in time exponential in the size of the input, not proving that the language is in P.

PROBLEM 6 HITTING SET: **Let $S = \{1, \ldots, n\}$ be a finite set. Let $C_1, \ldots, C_m \subseteq S$ be subsets of S and $K \leq n$ an integer. A subset $S' \subseteq S$ is a K-HITTING SET if $|S'| \leq K$ and S' contains at least one element from each subset C_i.**

Prove that $L = \{\langle S, C_1, \ldots, C_m, K \rangle : S$ contains a K-Hitting Set$\}$ is $NP-$Complete.

We reduce from VERTEX COVER. Let $G = (V, E)$ be a graph whose vertex cover we want to find. We make an instance of vertex cover with $S = V$ and $C_i = (a, b)$ if edge i is between vertices a and b. Thus there are $|E|$ $C_i' s$. By definition, a hitting set for this problem intersects every C_i, which if we translate this back into the language of graphs means that the hitting set includes at least one endpoint of each edge. That is, the hitting set corresponds to a vertex cover of G.

Since each item in the hitting set is also in the vertex cover, if we have a K-hitting set we get a vertex cover of size k. Thus the reduction preserves size. It takes time $|V| + |E|$, so it's polynomial time. Finally, HITTING SET is in NP: the certificate is the solution S', and I verify that each C_i actually includes an element from S' in time $\sum_i |C_i||S|$, which is polynomial. Thus HITTING SET is NP-complete.