HW 5

SJSU Students

December 11, 2006

1 A language is finitely simple if it's the union of simple languages. Show that if a finitely simple language is NP-complete, then P=NP

A language is simple if all strings in it come from {w}*, where w $\in \{0,1\}^*$. Let L be finitely simple. So the strings in L come from $(w_1)^* \cup \ldots (w_m)^*$, which is a regular expression. Since L is NP-complete, there is some reduction R so that $x \in SAT$ iff $R(x) \in L$. We can assume WLOG that if $R(x)$ is in $L((w_1)^* \cup \ldots (w_n)^*) - L$. Given a instance ϕ of SAT we use the same algorithm as in the unary case from class to solve it in P. This time though our hash function $H(t) = R(\phi(t))$ for the R above. We notice that all values of $H(t)$ are at most the length of the time of the reduction so of length bounded by some polynomial $p(n)$. Since an output value must be of the from $(w_i)^*$ for some i from 1 to m. The total number of distinct values H could output is bounded by a polynomial. The remainder of the argument is the same as for the algorithm we did in class.

2 Exhibit an oracle set A with respect to which UP is different from P, and hence, with respect to which there exist one way functions.

Consider the language

$$L = \{0^n | \textit{There is an } x \textit{ in } B \textit{ with } |x| = n\}$$

and let B be the oracle we used in class to show NP^B is different from P^B. The construction of B from class adds at most one string at any particular length. So if we want to check if 0^n is the L a NTM machine M could guess a string x of length n and check if it is in B. It should accept iff x is in B. As we just said, B has at most one of length n, so at most one path in M will accept, so this will be a UP^B algorithm.

3 Exhibit an oracle A under which NP $\bigcap$ coNP has complete problems, yet $\mathbf{NP}^A \neq \mathbf{P}^A$.

Note that if $NP^A = coNP^A$, then $NP^A \bigcap coNP^A = NP^A$. So SAT will be $NP \bigcap coNP$-complete. So we will construct an oracle such that $NP^A = coNP^A$ and $NP^A \neq P^A$.
This can be done by constructing oracle A in even and odd stages such that $A = \bigcup_i A_i$ Our construction has the property that A_i contains all strings in A of length $\leq$ i. We will also have an exception set X, which is also constructed in stages X_i so that $X := \cup_i X_i$. At stage i, we will ensure that it has size at most $\sum_{j=1}^{i} j^{logj} < 2^{i/2}$.

For stages of the form $2i$ we encode validity for strings of length i. We do not change X. i.e., $X_{2i} = X_{2i-1}$. For each y of length n, if y is a valid formula whose code is of length n, then we pick an x of length n so that the concatenated string yx is not in X_{2i}. This is possible since $|X_{2i}| < 2^{2i/2} = 2^i$.

We add the yx to A_{2i}. Notice given the problem of determining if y is valid an NP^A machine could guess x and check if yx was in A, so this step does ensure $NP^A = coNP^A$.

The stages of the form $2i + 1$ are used to force that $NP^A \neq P^A$. These stages we add elements to the set in exactly the same way we did for the oracle in class.