1 Honors Theory: Solution for HW3

1.1 QUESTION 1 (5 pts)

Unusual definition of NP-completeness: define L to be NP-complete if (a) it is in NP and (b) if $L \in P$ then $NP = P$. Say why this is unusual.

This definition is purely logical. This is very unusual: ALL our other definitions of NP-completeness refers to some "efficient reducibility" or machines concepts.

There are two logical possibilities:
(a) if $NP = P$, then this definition says every language in NP is NP-complete.
(b) if $NP \neq P$, then this definition says every language in $NP \setminus P$ is NP-complete.

Suppose $L \in NP$. How do we find out if L if NP-complete under this definition? This definition tells us to either show $L \in NP \setminus P$ (this implies showing $NP \neq P$) or to show that $NP = P$. In this sense, it appears to be useless for understanding the NP versus P question. In contrast, the usual definitions (by way of the Basic Lemma) are extremely useful in understanding the P versus NP question.

It is not hard to see that SAT is NP-complete under this funny definition. But we know this by way of other "useful" definitions of NP-completeness.

The definition, in contrast to the usual definitions, does not give us a polynomial analogue of Post's theorem.

1.2 QUESTION 2 (10 pts)

Show: If $P \neq DLOG$ then $DLOG$ is not closed under $\leq_m^P$.

Suppose A be any language in $DP \setminus DLOG$. Let B is the language comprising of just one string "yes". Clearly $B \in DLOG$. The result we want follows if we show that $A \leq_m^P B$. This is easy: let T be the transducer that on any input x, simulates any deterministic polynomial time acceptor M that accepts A. If M accepts, then T outputs "yes". If M rejects, then T outputs "no". Thus $A \leq_m^P B$ via T.

1.3 QUESTION 3 (10 pts)

Show: $NP = \text{co-}NP$ iff there is an NP-complete language in co-NP. State the needed properties for the underlying reducibilities.

SOLUTION: Let $\leq$ denote the reducibility under consideration. Some of you simply assume $\leq$ is Karp reducibility. This is cheating – we really want you to keep the reducibility abstract, but pinpoint the special property you need for the statement to be true. The property you need is this: co-NP is $\leq$-closed.

($\Rightarrow$) This direction is immediate.

($\Leftarrow$) Suppose L_0 is NP-complete and $L_0 \in \text{co-}NP$. We first show that $\text{co-}NP \subseteq NP$. Let $L \in \text{co-}NP$.

So co-$L \in NP$. But co-$L \leq L_0$ (since L_0 is NP-complete). Hence co-$L \in$ co-NP (since co-NP is $\leq$-closed). Hence $L \in NP$. This shows co-$NP \subseteq NP$. Hence co-$NP \subseteq NP$ implies co-$NP = NP$.

The last statement follows from the following fact: for any class K, co-$K \subseteq K$ iff $K \subseteq$ co-K iff $K =$ co-K. [Show this!]

Question: can you assume NP (instead of co-NP) is $\leq$-closed?

1.4 QUESTION 4 (10 pts)

Show: A is $PSPACE$-complete then $P^A = NP^A$.

SOLUTION: Since $P^A \subseteq NP^A$ is trivial, we only need to show $NP^A \subseteq P^A$. The desired result follows from the following two claims.

CLAIM 1: $NP^A \subseteq$. Proof: Since $A \in PSPACE$, suppose M accepts A in deterministic polynomial space. For any language $L \in NP^A$, let $Q^{(A)}$ be a non-deterministic polynomial time query machine that accepts L. Then we can define a non-deterministic acceptor N for L as follows: it simulates $Q^{(A)}$ in each step except that any oracle query is answered by simulating M on the query word. It is easy to see that N accepts in polynomial time. By Savitch's theorem, $L \in PSPACE$.

CLAIM 2: $PSPACE \subseteq P^A$. Proof: This is because every language $L \in PSPACE$ is $\leq_m^P$-reducible to A via some polynomial time transducer T. We then have a query machine Q that, on any input x, simulates T on x. If T outputs y, then Q queries its oracle, accepting iff the oracle answers "yes".

1.5 QUESTION 5 (20 pts)

For any language $A \subseteq \Sigma^*$, let

$$len(A) := \{w \in \Sigma^* : (\exists x \in A)|x| = |w|\}.$$

We need to construct a language A such that $len(A) \notin P^A$.

If $|\Sigma| = 1$ then $len(A) = A$ and hence $A \in P^A$. Thus $|\Sigma| \geq 2$ for this result to hold. Fix $\Sigma = \{0, 1\}$.

As hinted, let $Q = \{Q_0, Q_1, \ldots\}$ be an efficient universal oracle machine that runs in deterministic polynomial time. Condition C_i says that

$$C_i : len(A) \neq L(Q_i^{(A)}).$$

We proceed in stages. In stage i, we try to satisfy condition C_i. Inductively, assume that some finite partial set A_i of Σ^* has been formed. [RECALL the definition of "partial sets" in Chapter 4.6.] For simplicity, assume that the domain of A_i comprise all strings of length $\leq n_i$ for some n_i. Initially, let $n_0 = 0$ and we decide that the empty string is NOT in A_0.

We construct A_{i+1} to extend A_i as follows. Pick m_i larger than n_i and so that the running time of Q_i on any input of length m_i is strictly less than 2^{m_i}. Pick any input x of length m_i. Hence $Q_i^{(A)}$ on x could not possibly all strings of length m_i. For any language A and any natural number m, let $A|_m$ denote the restriction of A to those words of length m in A. Our goal is to satisfy C_i via

$$C'_i : len(A)|_{m_i} \neq L(Q_i^{(A)})|_{m_i}.$$

We consider the computation path C of $Q_i^{(A)}$ on input 0^{m_i}. Our rule for answering queries will be this: whenever a query word w has been decided to be in or out of A, we answer accordingly. But if a word w has not yet been decided, we will answer with a NO. This means w is henceforth decided to be out of A.

There exists at least one word $x_i \in \Sigma^{m_i}$ that is not queried by C. This is the critical decision: we declare x_i to be in A if and only if $Q_i^{(A)}$ rejects 0^{m_i}. We have thus satisfied the condition C_i'.

Notice that the computation of $Q_i^{(A)}$ on input 0^{m_i} may query words of length greater than m_i. Finally, let n_{i+1} be the length of the longest word that has been decided (to be in or out of A). Declare all strings of length $\leq n_i$ which are not yet decided to be OUT of A. This completes the construction. Notice that our construction defines two sequences of numbers, $n_0 < m_0 < n_1 < m_1 < \cdots < n_i < m_i < \cdots$. Correctness of this construction is immediate.