Christopher Kwan ckwan@bu.edu

CS 535 | Homer | Fall 2010 due: 11/23/2010

Homework 5

Discussed with Mike Breslav and Dan Schatzberg

1. Page 136, #6.11.
   Prove: co-NP = NP <-> some NP-complete set (X) has its complement (XBar) in NP.

   (->): co-NP = NP -> some NP-complete set (X) has its complement (XBar) in NP.
   - Since co-NP = NP, the complements of all languages in NP are also in NP.
   - Since X is NP-complete, it is in NP.
   - Thus, since the complements of all languages in NP are also in NP and since X is in NP, the complement of X, XBar, is also in NP.

   (<-): XBar in NP -> co-NP = NP.
   - A co-NP-complete problem is the complement of an NP-complete problem (Page 136, Homework 6.12 part 1). Thus, since X is NP-complete, its complement, XBar, is co-NP-complete.
   - Since XBar is co-NP-complete, all languages in co-NP can be reduced to XBar.
   - Since XBar is in NP and all co-NP languages can be reduced to it, all co-NP languages are in NP.
   - Since all co-NP languages are in NP, co-NP is a subset of NP.

   - Since XBar is in NP, its complement XBarBar is in co-NP by definition of complements.
   - The complement of the complement of X (XBarBar) = X, thus X is in co-NP.
   - Since X is NP-complete, all NP languages can be reduced to it.
   - Since X is in co-NP and all NP languages can be reduced to X, all NP languages are in co-NP.
   - Since all NP languages are in co-NP, NP is a subset of co-NP.

   - Since, co-NP is a subset of NP and NP is a subset of co-NP, co-NP = NP.

10/10

2. Page 136, #6.12, part 2.
   A set A in co-NP is $\leq_m^p$-complete for co-NP if for all L in co-NP, L $\leq_m^p$ A.

   Show the problem (X) of determining whether a formula (F) of propositional logic is a tautology is $\leq_m^p$-complete for co-NP.
   - A co-NP-complete problem is the complement of an NP-complete problem (Page 136, Homework 6.12 part 1).

Homework 5

Rick Skowyra
CS535

Problem 2

Claim: TAUTOLOGY = {x | x is a formula of propositional logic which is valid for all assignments} is co-NP-Complete.
Proof: To show that TAUTOLOGY is co-NP-Complete, we must show that $TAUTOLOGY \in$ co-NP and $\forall A \in$ co-NP, $A \leq_M^P TAUTOLOGY$. To show $TAUTOLOGY \in$ co-NP, we can construct a polynomial time verifier of the 'reject' case. Define a TM M which, given a Boolean formula and an assignment of its variables, decides if the assignment does not satisfy the formula. Let M be an Turing machine with a read-only input tape and a work tape. Construct M as follows:

1. Copy the Boolean formula on to the work tape. This takes linear time.
2. Replace each variable with its value. This takes linear time.

3. Recursively replace every pair-wise operation with its value (i.e. $(0 \wedge 1)$ is replaced with 0). Consider that the first replacement phase reduces the size of the formula from $|x|$ to $\log(|x|)$ in $O(|x|)$ steps. Each replacement phase continues to halve the size of the formula to be evaluated in the next phase. This step therefore takes polynomial time.

4. If the final replacement phase yields a 0, halt and reject.

5. If the final replacement phase yields a 1, halt and accept.

Clearly, $M \in DTIME(n^k)$. Next, we must show that $\forall A \in$ co-NP, $A \leq_M^P TAUTOLOGY$. To do so we can show $\overline{SAT} \leq_M^P TAUTOLOGY$. Given a NTM N which decides TAUTOLOGY in polynomial time, construct a NTM M which decides $\overline{SAT}$ in polynomial time as follows:

1. On input x, create $x' = \neg x$. This take linear time.
2. Run $N(x')$. This takes polynomial time.
3. If N accepts, halt and accept.
4. Otherwise halt and reject.

M clearly runs in polynomial time. Since $\overline{SAT}$ is co-NP-Complete, $TAUTOLOGY \in$co-NP, and $\overline{SAT} \leq_M^P TAUTOLOGY$, TAUTOLOGY is co-NP-Complete.

Problem 3

Claim: CLIQUE $\leq_M^P$ VERTEX COVER
Proof: Given a NTM N which decides VERTEX COVER in polynomial time, we can construct a NTM M which decides CLIQUE in polynomial time. Define M as follows:

1. Ensure that the input is of the form G=(V,E) and an integer k.
2. Create $G^c = (V, E^c)$. (E^c is the set of all edges not present in E). This takes polynomial time. (TIME(n^2) if using an adjacency matrix.)
3. Run N(G^c, $|V| - k$). This takes polynomial time.
4. If N accepts, halt and accept.
5. Otherwise halt and reject.

Clearly, M halts in polynomial time. The above reduction relies on the fact that in the complement to a graph, an independent set (the remaining vertices after having found the vertex cover) becomes a clique, and vice versa. So a vertex cover of size $|V| - k$ in the complement to a graph indicates a clique of size k in the original graph.

10/10

Problem 4

Define L(N) to be the set of all input strings x that are accepted with probability at least one half by a probabilistic polynomial-time NTM N.

1. $L(N) \in$ PSPACE. To show this, consider the binary tree of computational paths formed by N. For a path p in N(x)'s computational tree, $Pr[p] = \frac{1}{2^t}$, where t is the number of nondeterministic choices made in the path. In order to determine the total probability of acceptance for an input word x, the sum over all accepting paths for x must be computed. We can do this in polynomial space with the following algorithm:

   (a) On input x, deterministically perform a depth-first search over N's computational tree.

   (b) For each path p explored, compute $Pr[p]$.

   (c) If that path accepts, record $Pr[p]$.

   (d) Otherwise, discard the sum and clear all state related to that path.

   (e) Explore the next path, re-using the tape cells that were used for the previous path.

   (f) If at any point the total probability of accepting paths exceeds $\frac{1}{2}$, halt and accept.

   (g) If all paths have been explored and the total probability of accepting paths does not exceed $\frac{1}{2}$, halt and reject.

   You should also mention that the space to store the probability is in PSPACE.

   8/10

   The above algorithm uses a polynomial amount of space. Consider that a N must halt in $|x|^k$ steps for some fixed k. This puts a polynomial bound on the computational path length of N. Since a TM cannot use more than one tape cell per time step, each path must use only a polynomial amount of space. Since we re-use the tape cells for each path, the number of paths that must be explored is independent of the amount of space used. Therefore $L(N) \in PSPACE$. ✓

2. If $Pr(x) = 0 \; \forall x \notin L(N)$, $L(N) \in NP$. This follows directly. Since $Pr(x) = 0$ $\forall x \notin L(N)$, N will always halt in a non-accepting state if $x \notin L(n)$. Furthermore, if $x \in L(n)$, there always exists at least one accepting path since $Pr(x) \geq \frac{1}{2}$. Therefore $N \in NTIME(n^k)$, N has at least one accepting path if $x \in L(N)$, and N has no accepting path if $x \notin L(N)$. Therefore $L(N) \in NP$. ✓

Problem 5

1. $A \leq_T^P B$ and $B \in P$ implies $A \in P$. Recall that if $A \leq_T^P B$, there exists a polynomial time bounded oracle TM O s.t. $x \in A \leftrightarrow O(x, B)$ accepts. Then , we can construct a polynomial time TM M to decide A:

   (a) On input x, run O(x).

(b) If O accepts, halt and accept.

(c) Otherwise halt and reject.

By definition, $O \in P$, so it runs in at most $TIME(|x|^k)$. Each time O queries the oracle B, it writes an input word y on the oracle tape. Since $TIME(n^k) \subseteq SPACE(n^k)$, $|y| \leq |x|^k$ for some fixed k. We know that $B \in P$, so on input y, it runs in $TIME(|y|^l)$. Since $|y| \leq |x|^k$, $|y|^l \leq |x|^{kl}$, which is still polynomial. Finally, recall that O may query B $|x|^k$ times, since $O \in P$ and an oracle query takes one time step. Therefore the total running time of M is $TIME(|x|^k * |x|^{kl}) = TIME(x^{k+kl})$, so $A \in P$.

2. $\overline{A} \leq_T^P A$. This can be easily shown via construction of polynomial time OTM O:

   (a) Write x on to the oracle tape. This takes $|x|$ steps.

   (b) Query the oracle. This takes one step.

   (c) If the oracle accepts, halt and reject. This takes one step.

   (d) If the oracle rejects, halt and accept. This takes one step.

The total running time of O is $TIME(O(n))$, so $O \in P$.

Problem 6

The following function is one-way: it is polynomial-time computable, but cannot be inverted in polynomial time. Define $f : 0^* \rightarrow \{0,1\}^*$ s.t. on unary input x, $f(x)$ is its binary representation. The following algorithm computes $f(x)$ given x, and runs in polynomial time on an offline Turing machine:

1. If there is a B on the input tape, halt and reject.

2. Read one character of the input word.

3. Write a 1 on the work tape.

4. Read the next character.

5. If it is B, halt and accept.

6. If not, start scanning the work tape head left. This takes at most TIME(

7. If a 0 is read:

   (a) Write a 1 and start scanning right.

   (b) Until a B is encountered, write a 0 in each cell.

8. If a 0 is not read:

(a) Write a 1 to the first tape cell.
(b) Scan right, writing a 0 in each tape cell until a B is encountered.
(c) Replace B with 0.

9. Go to step 4

As an upper bound on the time complexity of the above algorithm, note that reading one character of the input causes at most $2 * y + 1$ operations on the work tape, where y is the size of the string currently on the work tape. Recall that the length of a binary number is logarithmically smaller than its equivalent in unary. Therefore above algorithm runs in $TIME(O(2 * log(n)^2))$. Clearly, $f \in PF$. Trying to invert the function, however, causes an exponential increase in the space needed to represent the output (by definition of a unary number). Since it requires one time step to write to a tape cell, an exponential amount of time relative to the input is required. Therefore $f^{-1} \notin PF$.

10/10

Problem 7

Let L be a PSPACE-Complete language.
Claim: If $L \in NP$, then $NP = PSPACE$.

Proof:
This proof follows naturally from the definition of completeness. Recall that if L is PSPACE-Complete, then $\forall A \in PSPACE$, $A \leq_M^P L$. Note that if $A \leq_M^P L$ and $L \in NP$, $A \in NP$. We know that $NP \subseteq PSPACE$. If $L \in NP$, then all elements of PSPACE are also in NP, and $PSPACE \subseteq NP$. By the definition of set equality, $NP = PSPACE$.

10/10