Honors Theory, Spring 2002, Yap
Homework 5 (with SOLUTIONS)

This is due on Wed April 3.

Please try to rewrite your solutions to make them more concise (without being more obscure). Always introduce your paragraphs with an introduction (e.g., what method of proof will be used, etc).

1. (10 Points) Question 3 of the midterm is a basic type of argument that you need to master in computability theory.

   Fix any r.e. set A. You were asked to construct a total recursive function $t : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \mathbb{N}$, $x \in A$ iff $t(x) \in$ HALT. Now, you should realize that $t(x)$ is the index of a suitable STM (isn't that all that HALT knows about?). In our published solution, we described this STM and called it M_x. Then we simply said "the function $t(x)$ which gives the index of M_x is clearly total computable". Usually, you can simply assert this without further elaboration. But for this question, we specifically ask you to elaborate on it.

   ANSWER: Let N be a STM that accepts A. The STM M_x, on any input y, ignores y and simulates N on x, and accepts iff N does. We may assume $\Sigma = \{0, 1\}$ is the input alphabet for N and for M_x. However, the state set for M_x must depend on $|x|$. Let all the states we need be identified with with elements of $Q\{0, 1\}^*$ ($Q, 0, 1$ are special symbols). In particular, let $Q, Q0$ be the start state q_0 and accept state $q_1 = q_a$, respectively. We assume that the code for a STM is a sequence of instructions of the form $I = \langle q, a \to q', b, D\rangle$, meansing that if you are in state q reading a then you go into state q', replace a with b and move the head in direction $D \in \{-1, 0, +1\}$. Thus I can be encoded as a string over a fixed alphabet $\Gamma = \{0, 1, Q, \ldots\}$.

   Fix x of length n. The instructions of M_x can be divided into the following four groups:
   (1) First, we have instructions to "blanking out the input" while staying in the start state q_0. We long as the currently scanned symbol is non-blank, we replace it by a blank and move right. For all $a \in \Sigma$, we have the instruction
   $$I_a = \langle q_0, a, q_0, \sqcup, +1\rangle.$$
   When we are done blanking, we move into state q_1:
   $$I_\sqcup = \langle q_0, \sqcup, q_1, \sqcup, 0\rangle.$$

   (2) Next, we write the word $x = x_1, \ldots, x_n$ into the tape of M_x. We introduce the states $q_1, \ldots, q_{n+1}$. For $i = 1, \ldots, n-1$, we have
   $$I_i = \langle q_i, \sqcup, q_{i+1}, x_i, +1\rangle$$
   and also
   $$I_n = \langle q_n, \sqcup, q_{n+1}, x_n, 0\rangle.$$

   (3) Now, we move the head position to the beginning of the word x: for each $a \in \Sigma$, let
   $$I'_a = \langle q_{n+1}, a, q_{n+1}, -1\rangle.$$
   Finally, when we reach the first blank to the left of x, we move into state q_{n+2}:
   $$I'_\sqcup = \langle q_{n+1}, \sqcup, q_{n+2}, +1\rangle.$$

   (4) We now begin simulating N on x. Simply insert all the instructions of N into M_x (but with q_{n+2} in place of the start state of N).

   Thus, $t(x) \in \Gamma^*$ is just a linear sequence of instructions corresponding to the above four groups, (1)–(4). Note that only group 3 depends on x. Using any $|\Gamma|$-adic encoding, we can view $t(x) \in \mathbb{N}$.

   REMARK: all these are tedious, but straightforward. That is why, generally, we do not ask you to provide such details. But you should know how to do it when "challenged" to be explicit.

2. (10 Points) Construct a non-recursive bijection $f : \mathbb{N} \to A$ where A is an infinite r.e. set. NOTE: This possibility seems to have been overlooked by several students that I decided you should try to construct one for yourself.

   HINT: Since A is infinite r.e., there recursive denumeration of A, $\alpha : \mathbb{N} \to A$. Define $\alpha_0(i) = \alpha(2i)$ and $\alpha_1(i) = \alpha(2i+1)$. Thus $\alpha_j : \mathbb{N} \to A_j$ (for $j = 0, 1$) is a denumeration. Try to define $f : \mathbb{N} \to A$ such that $f(i) \in A_0$ iff $i \in \text{HALT}$.

   ANSWER: We know that there exists denumerations

$$h : \mathbb{N} \to \text{HALT}, \qquad \overline{h} : \mathbb{N} \to \text{co-HALT}.$$

   With α's as in the HINT, define

$$f(i) = \begin{cases} \alpha_0(h^{-1}(i)) & \text{if } \ i \in \text{HALT} \\ \alpha_1(\overline{h}^{-1}(i)) & \text{else.} \end{cases}$$

   Then f is a recursive denumeration of A. It remains to show that if f is recursive, then HALT is recursive. This is because $i \in \text{HALT}$ iff $f(i) \in A_0$. To check if $f(i) \in A_0$, we simply dovetail the computations

$$\{\alpha_0(j), \alpha_1(j) : j \in \mathbb{N}\}$$

   until we see $f(i)$ being output. Then we know whether $f(i) \in A_0$ or not.

3. (10 Points) Show that $NP = \text{co-}NP$ iff there is an NP-complete language that is in co-NP. There is an implicit notion of reducibility (which I will simply denote as "$\leq$") in these terminology. You have to tell us what property is needed of $\leq$ to make this result true.

   ANSWER: The direction from left to right is trivial. Take any language L that is NP-complete, for example SAT. Since $L \in NP$ and $NP = \text{co-}NP$ then $L \in \text{co-}NP$.

   For the other direction, given a NP-complete language L_0 in co-NP, we show that $NP = \text{co-}NP$. For any language $L \in NP$, we have $L \leq L_0$. As $L_0 \in \text{co-}NP$, and co-NP is closed under reductions, $L \in \text{co-}NP$. Therefore, $NP \subseteq \text{co-}NP$. We can now also conclude $\text{co-}NP \subseteq NP$: if $L \in \text{co-}NP$ then $\text{co-}L \in NP \subseteq \text{co-}NP$, and so $L \in NP$.

   What is the property needed of our reductions? In the preceding argument, we need the unproved claim, that

$$\text{co-}NP \text{ is closed under reductions.} \tag{1}$$

   We can prove this directly, just the same way we prove that NP is closed under reduction.

   Alternatively, we can use a different property, that our $\leq$-reduction has the property:

$$L \leq L' \Leftrightarrow \text{co-}L \leq \text{co-}L'. \tag{2}$$

   For instance, (2) would be true if $\leq$ is a many-one reducibility (polynomial time or not). From (2), it is quite easy to prove (1).

4. (15 Points) Show that if A is $PSPACE$-complete under Karp-reducibility, then $P^A = NP^A$. NOTE: P^A means the class of language accepted in polynomial time by a deterministic oracle machine with A as oracle. The class NP^A is similarly defined, but in the nondeterministic mode.

   ANSWER: If A is a $PSPACE$-complete languge, we will show

$$PSPACE \subseteq P^A \subseteq NP^A \subseteq PSPACE.$$

   The first inclusion holds because A is $PSPACE$-complete, and thus any language $L \in PSPACE$ can be decided by a polynomial-time deterministic Turing machine M that performs the reduction from L to A. More precisely, if the output of M on input x is $t(x)$, we simply ask the oracle A if $t(x) \in A$. If the oracle says YES, we accept, else reject.

The second inclusion is trivial, as any polynomial-time deterministic Turing machine is also a polynomial-time nondeterministic machine.

For the third inclusion, any nondeterministic polynomial-time Turing machine M with oracle A can be simulated by a nondeterministic polynomial space-bounded Turing machine M', which resolves the queries to A by running a third polynomial space machine that accepts A. Although this M' is nondeterministic, Savitch's theorem tells us that there is a deterministic polynomial space machine that simulates M'.

[You can also refer to Papadimitriou's book p.340.]

5. (15 Points) Let s be a space-constructible function (see Chapter 2 for definition). Assume $s(n) < \infty$ for all n. Prove a $(DSPACE(s), RE)$ separation result.

   ANSWER: From space hierarchy theorem, given a space-constructable function s, if we take a space-constructable function s' such that $s'(n) = \omega(s(n))$, then

   $DSPACE(s') - DSPACE(s) \neq \emptyset$.

   Simply choose $s'(n) = s(n)n$. Since $DSPACE(s') \subseteq RE$, $RE \setminus DSPACE(s) \neq \emptyset$.

6. (20 Points) A **Simple Alternating Machine** is like an alternating machine, except that it has 1 worktape but no input tapes. The input is placed on the sole worktape. Show that such a machine can accept the palindrome language in linear time, using only a constant number of alternations. HINT: guess the positions (in binary) of about log n equally spaced input symbols, writing them directly under the corresponding symbols. Call these the "marker positions". The positions of the other symbols can be determined relative to these marker positions. Remember that if you guess something, you need to verify it.

   ANSWER:

   Given an input $w = w_0, \ldots, w_{n-1}$ where $w_i \in \{0, 1\}$, we want to check if $w[i] = w_i$ is equal to $w[n-i]$ for all $i = 0, \ldots, n-1$. Here is the algorithm (it is basically what we outline in class):

   (a) Universally guess $i = 0, \ldots, n-1$ by placing a marker "X" under $w[i]$. Assume w is on track 1, and "X' is on track 2.

   (b) Existentially guess $n - i$ by placing another marker "Y" under some $w[j]$. If $w[i] \neq w[j]$, answer NO. Otherwise, we must check that $i + j = n$. Note that we allow $i = j = n/2$.

   (c) On track 3, existentially guess a sequence of markers denoted "Z" under w. You must put a marker under $w[1]$ and under $w[n]$. Let there be $k \geq 2$ such Z-markers.

   (d) On track 4, existentially guess a binary string. This binary string must extend beyond $x[n]$. The information on tracks 3 and 4 can be viewed as a sequence of dyadic numbers,

   $$m_1, m_2, \ldots, m_k$$

   where the most significant bit of m_i is the binary bit under the ith "Z" marker. The guess is "correct" if $x[m_i]$ is the position of the ith Z-marker. For instance, we want $m_1 = 0$ and $m_k = n$. [Actually $m_1 = 0$ is a special case, as its dyadic representation is the empty string, so we should not write 0 or 1 under $x[0]$.] You also see why we insist that the binary string on track 4 must be longer than n, otherwise m_k is probably wrong.

   (e) On track 5, we existentially guess another binary string. Again, this string combined with the Z-markers on track 3 gives us a sequence of dyadic numbers

   $$x_1, x_2, \ldots, x_k$$

   This sequence is "correct" if $x_1 = x_2, \ldots, = x_k = i$. In otherwords, x_i indicates the position of the X-marker.

(f) Similarly, on track 6, we existentially guess a binary string to represent a sequence of dyadic numbers

$$y_1, y_2, \ldots, y_k$$

This sequence is "correct" if $y_1 = y_2, \ldots, = y_k = j$. In otherwords, y_i indicates the position of the Y-marker.

(g) Universally choose to verify that each m_i is correct. We simply make sure that $m_{i+1} - m_i$ is equal to the distance between the $i+1$st Z-marker and the ith Z-marker. This is done by computing the "counter" $c_i = m_{i+1} - m_i$, and then decrementing c_i to 0. As you decrement, you check the number of spaces between the two Z-markers.

(h) Universally choose to verify that $x_i = x_{i+1}$ and $y_i = y_{i+1}$ for all $i = 1, \ldots, k$.

(i) Finally, verify that the position of the X-marker is equal to the x_i value that is closest to it. Similarly, the position of the Y-marker is equal to the y_i-value that closest.

(j)) The final verification is that $x_k + y_k = m_k$.

This completes the description of the simple alternating machine. It is easy to argue that (1) if the input w is a palindrome, this machine will accept in $O(n)$ time, and (2) if w is not a palindrom, this machine never accepts.