CS281 Spring 2010: Homework 3

Due Wednesday, April 28th – SOLUTION

1. (10 points)
Informally but clearly describe a Nondeterministic Turing machine – multitape if you like – that accepts the following language [Try to take advantage of nondeterminism to avoid iteration and save time in the nondeterministic sense. That is, prefer to have your NTM branch a lot, while each branch is short.] :

   The language of all strings of the form $w_1\#w_2\#\ldots\#w_n$, for any n, such that each w_i is a string of 0's and 1's, and for some j, w_j is the integer j in binary.

   Sol'n: It is easy to define this machine using two tapes - one the input tape, the other a working tape. Define two subroutines:

   - subroutine s_a that adds one to a binary number on the working tape (started scanning the first bit; assume it ends scanning it as well), and advances its head on the input tape to the first bit after the next non-binary bit, accepting if that was a B or a # and rejecting otherwise. [Note: by convention B may only occur at the end of the input string.]
   - subroutine s_c that decides if the bit string on the working tape (from currently scanned bit until just before B) matches the string on the input tape (from currently scanned bit until just before next # or B) – and ends scanning the last compared bit of the input tape, and the first bit of the working tape.

   Then define a NTM M that first writes 1 on the working tape without moving and then enters a state s_{ND} where it nondeterminisitically chooses between a state in which the subroutine s_a will get called and another state in which the subroutine s_c will get called. If s_a rejects then M rejects, otherwise it returns to state s_{ND}. If s_c accepts then M halts and accepts.

2. (10 points)
Consider the nondeterministic Turing machine

$$M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \{q_f\})$$

   Informally but clearly describe the language $L(M)$ if δ consists of the following sets of rules: $\delta(q_0,0) = \{(q_0,1,R),(q_1,1,R)\}; \delta(q_1,1) = \{(q_2,0,L)\}$; $\delta(q_2,1) = \{(q_0,1,R)\}; \delta(q_1,B) = \{(q_f,B,R)\}$.

Sol'n: The language $L(M)$ consists of all binary strings which begin with the bit 0.

To see this, note that if a string x begins with any symbol other than 0 then M cannot leave the initial state q_0 on input x, so $x \notin L(M)$. On the other hand, if a binary x begins with one or more 0's then by using the non-deterministic choice $(q_0, 1, R) \in \delta(q_0, 0)$ repeatedly the machine can convert all 0's at the start to 1's, and then arrive via $(q_1, 1, R)$ to be scanning the first nonzero symbol in state q_1. In fact, this also works whenever M scans the start of a sequence of 0's regardless of whether they are initial or not. If the next non-0 symbol is B then M accepts. If not, then it is 1 and M writes 0 there and by a sequence of moves (see †) returns to this 0 in state q_0 without making any other changes to the string. Thus M may advance through the entire string until it reaches B is state q_1, at which point it will transition to accepting.

The sequence † of moves is: M writes a 0 in scanned cell (via $(q_2, 0, L) \in \delta(q_1, 1)$) and steps to the left, where it finds a 1 by assumption. M is now in state q_2 so this 1 will be retained and M will move to the right (using $(q_0, 1, R) \in \delta(q_2, 1)$).

3. (10 points)
A k-head Turing Machine has k heads reading cells of one tape. A move of this TM depends on the state and on the symbol scanned by each head. In one move, the TM can change state, write a new symbol on the cell scanned by each head. and can move each head left, right or keep it stationary. Since several heads may be scanning the same cell, we assume the heads are numbered 1 through k, and the symbol written by the highest numbered head scanning a given cell is the one that actually gets written there. Prove that the languages accepted by k-head Turing Machines are the same as those accepted by ordinary TM's.

Sol'n: Since any TM may be simulated on a k-head TM using only the first tape, it only remains to show that any k-head TM M_k may be simulated by a standard TM. For this we may instead show M_k may be simulated by a 3-tape TM M since we know the languages accepted by these are the same as those accepted by ordinary TM's. [Note from TA: I assume here the definition given in Hopcroft, also implicit in the above description, whereby M_k has only one state at each moment and each value of the transition function specifies a new state for M and a k-tuple of pairs, $((s_1, d_1), \ldots, (s_k, d_k))$, where s_i is the symbol to be written and d_i the direction to be moved by the i'th head.]

Let M have an input tape containing the input to M_k, and 2 working tapes, the first of which begins blank. Let the symbols used by M on this working tape be the same as those used by M_k and let the set of symbols used on the second work tape be $Pow([k])$. When a symbol for a subset S of $[k] = \{1, \ldots, k\}$ appears in the i'th cell of the second work tape we will

interpret this to mean the set S of heads are scanning the corresponding cell of the input tape. So, let the second work tape start with the symbol for $\{1, \ldots, k\}$ on its scanned cell and let all the rest of its cells be blank. Let the set of states of M be the cartesian product: the set states of M_k $\times$ the set of additional states required as control states to execute the following algorithm (so M always knows both its M_k-state and its control state):

1. Apply the transition function of M_k to the current M_k-state and the vector of symbols scanned by the k simulated heads. Suppose this transition gives $((s_1, d_1), \ldots, (s_k, d_k))$.
2. Go through the $((s_1, d_1), \ldots, (s_k, d_k))$ in order (starting at $i = 1$ and going to $i = k$). Using the symbols written on all the cells of the second work tape as a guide (since they encode the location of the k simulated heads) write each symbol s_i on the first working tape in the current position of the i'th head.
3. Update the entries in the second work tape so that they reflect the new positions of the k heads (as determined by the d_i).
4. Now transfer each of the non-blank symbols on the first work tape to the input tape (in corresponding cell) and replace with blanks on the first work tape.
5. go to 1.

If at any point during the execution of the algorithm M_k accepts then let M accept. Given M_k with input x and M as described (with corresponding input) it is easily verified that the input tape of M will always exactly match that of M_k after each time a new transition is called in both machines (and 1.-5. are executed by M). Since M accepts if and only if M_k does, this shows they accept the same language.

4. (10+10 = 20 points)
   State (with justification) whether the recursive languages and the RE languages are closed under the following operations. You may give informal but clear constructions to show closure.
   a) Concatenation
   b) Kleene Star[1] operation.

Sol'n: Both recursive and r.e. languages are closed under the above operations.

For (a), given TM's M_1 and M_2 which decide (resp. accept) the languages L_1 and L_2, define a NTM which upon input x, arbitrarily chooses a dividing point $i \in \{0, \ldots, |x|\}$ and runs M_1 on the first i bits of $|x|$ and M_2 on the remaining

[1]Given a language L, the Kleene star of L is the language $L^* = \bigcup_{n \in \mathbb{N}} L^n$ where $\forall n > 0$, L^n is the language consisting of concatenations of n elements of L and $L^0 = \{\Lambda\}$, Λ being the empty string.

bits, halting in accepting state if both accepted and going to a halting rejecting state otherwise. Clearly if $x \in L_1 L_2$ then some i produces accepting states for both these machines and so x is accepted, while if $x \notin L_1 L_2$ no i will do so and hence x is not accepted by the nondeterministic machine. Moreover, in the recursive case where the M_i were assumed always halting, the nondeterministic machine just described will have all branches halting and hence is a decider for the language it accepts.

For (b), use analogous nondeterministic machines, except choose from among all possible partitions of x into multiple substrings, rather than just all possible partitions into two substrings. The argument that these machines work is as before.