Problem Set 7

Spring 10

Due: Thursday Mar 25 in class before the lecture.

Please follow the homework format guidelines posted on the class web page:

`http://www.cs.uiuc.edu/class/sp10/cs373/`

1. The way to Tbilisi [**Category**: Puzzle, **Points**: 10]

   On your way to Tbilisi, you come across a fork in the road, one going left and one going right, and you wonder which one goes to Tbilisi. Fortunately, a woman from the neighboring village of Truthli sits on a boulder at the fork, eating peanuts, who surely knows the way. The people of Truthli are peculiar— a Truthli citizen either speaks always the truth or always lies (how this choice is made at birth is still mysterious). Of course, you do not know this woman's orientation to truth.

   Come up with one question you can ask the woman from Truthli such that from her answer you can infer which road leads to Tbilisi.

2. Reductions [**Category**: Proof, **Points**: 20]

   (a) **Reductions for decidability:**
   C is a restricted class of Turing machines for which `cs373.com` provides a very useful Turing machine M_C that takes in a pair $\langle M, D\rangle$, where M is a Turing machine in class C, and D is a DFA, and *accepts* iff $L(M) \cap L(D) \neq \emptyset$. In other words, M_C can decide whether the language of a Turing machine in class C has a common word with the language of a DFA D.

   More precisely,
   $L(M_C) = \{\langle M, D\rangle \mid M \text{ is a TM in } C \text{ and } D \text{ is a DFA and } L(M) \cap L(D) \neq \emptyset\}$.

   We would like to now solve the membership problem for Turing machines in class C: given a TM M in C and a word w, we want to decide whether $w \in L(M)$. More precisely, we want to show that
   $L_{memb} = \{\langle M, w\rangle \mid M \text{ is a TM in } C \text{ and } w \in L(M)\}$ is decidable.
   Prove that L_{memb} is decidable by using a reduction.

   (b) **Reductions for undecidability:**
   A Turing machine *rejects* a word w if, when started with input w, it halts and reaches the reject state. (Note: If a TM does not accept w, it does not mean it rejects w, as it may not halt on w).
   Let $L_{rej} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ rejects } w\}$.
   Show that L_{rej} is undecidable, using a reduction, and by using the fact that the universal language L_u (or A_{TM} in Sipser) is undecidable.
   Recall that $L_u = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$.

Solution:

(a) We will show this by reducing L_{memb} to $L(M_c)$.

The TM deciding L_{memb} will decide whether M accepts w by constructing a DFA D accepting the language $\{w\}$ and by using the TM M_C to figure out if $L(M) \cap L(D)$ is empty.

The TM deciding L_{memb} will work as follows.

1. Input: $\langle M, w \rangle$
2. Check if M is a valid Turing machine.
3. Construct a DFA D that accepts w and only w. I.e. $L(D) = \{w\}$.
4. Feed $\langle M, D \rangle$ to M_C.
5. Accept iff M_C accepts.

Since the DFA D accepts $\{w\}$, $L(M) \cap L(D) \neq \emptyset$ iff $w \in L(M)$. Hence the above TM accepts the language $L_{memb} = \{\langle M, w \rangle \mid M \textit{ is a TM in } C \textit{ and } w \in L(M)\}$.

Note that the above TM for L_{memb} is a decider, as Steps 2 and 3 always halts, and Step 4 halts because M_C is assumed to be a decider.

Hence if M_c is a decidable, then L_{memb} is also decidable.

(b) We will show this by reducing L_u to L_{rej}. Since L_u is undecidable, it follows that L_{rej} is undecidable.

Assume there is a TM M_{rej} that decides L_{rej}. The Turing machine M_u deciding L_u will, on input $\langle M, w \rangle$, build a TM M' such that M' rejects w iff M accepts w. It will feed $\langle M', w \rangle$ to M_{rej} to figure out whether M' rejects w, from which it will deduce whether M accepts w.

We build the TM M_u that decides L_u as follows:

1. Input: $\langle M, w \rangle$
2. Check if M is a valid Turing machine. If not, reject. Let q_{acc} and q_{rej} be the accepting state and rejecting state of M, respectively.
3. Construct the TM M' from M, by making q_{acc} the rejecting state and q_{rej} the accepting state.
4. Feed $\langle M', w \rangle$ to M_{rej}.
5. Accept iff M_C rejects.

When M runs on w, if M halts and accepts, then the corresponding machine M' constructed by the decider above, when running on w, will halt and reject. If M does not accept w, then it could be because M does not halt or halts and rejects. In either case, M' will not reject w. Hence M' rejects w iff M accepts w.

Hence the Turing machine M_u above accepts L_u.

Note that the TM M_u above for L_u halts on all inputs because Step 3 simply constructs the *code* of M' from M, and this code transformation (which involves just flipping the accept and reject states of M) can be done in a finite number of steps. Step 2 is also a check that takes only a finite number of steps. Step 4 halts because M_{rej} is assumed to be a decider, and hence halts on all inputs. Hence the TM M_u is a decider for L_u.

3. TM to NFA [**Category**: Proof., **Points**: 20]

   An Right-move Turing Machine (RTM) is Turing that can only move its head to the right. It can not move it to the left or leave in the same position. More formally, a RTM is a tuple $M = (Q, \Sigma, T, \delta, q_0, q_a, q_r)$ where Q is a set of states, Σ is the input alphabet, T is the tape alphabet, q_0, q_a and q_r are the initial, accept and reject states respectively and $\delta : Q \times T \to Q \times T \times \{R\}$ is a transition function. Prove that the languages accepted by RTMs are regular by giving a construction of a DFA/NFA from RTM.

Solution:

Intuitively, we take a TM and make an NFA that consists of two phrases(transitions between Q and P, respectively). The first phrase emulates the TM on symbols from the input alphabet. Note that since the TM will never be able to read any symbol that it writes (since the head always moves right), the NFA can ignore the symbols written by the TM. The second phrase emulates the TM on the "blank" symbol using epsilon-transitions. We have epsilon transitions from the first part to the second from the states that had a transition on "blank" symbol. Moreover, we make sure that the NFA stops simulating the TM when the TM reaches an accepting or rejecting state.

Let $M = (Q, \Sigma, T, \delta, q_0, q_{acc}, q_{rej})$ be a RTM. We will construct an NFA $A = (Q', \Sigma', \delta', q_0', F')$ that will accept the same language as M. The set of states is $Q' = Q \times \{0, 1\}$ (i.e. we have two copies of states in Q). $\Sigma' = \Sigma$, $q_0' = (q_0, 0)$, $F' = \{(q_{acc}, 0), (q_{acc}, 1)\}$ where $q_a = q_t$ for some t.

The δ'-function is:

$$\begin{cases} \delta'((q,0),a) & = \{(q',0)\} & \text{if } \delta(q,a) = (q',c,R), \text{for some } c \in T, \forall a \in \Sigma, q \notin \{q_{acc}, q_{rej}\} \\ \delta'((q,0),\epsilon) & = \{(q',1)\} & \text{if } \delta(q,\textvisiblespace) = (q',c,R), \text{for some } c \in T, q \notin \{q_{acc}, q_{rej}\} \\ \delta'((q,0),a) & = \emptyset & \forall a \in \Sigma \cup \{\epsilon\}, q \in \{q_{acc}, q_{rej}\} \\ & & \\ \delta'((q,1),\epsilon) & = \{(q',1)\} & \text{if } \delta(q,\textvisiblespace) = (q',c,R) \text{ for some } c \in T, q \notin \{q_{acc}, q_{rej}\} \\ \delta'((q,1),a) & = \emptyset & \forall a \in \Sigma, \forall q \in Q. \end{cases}$$

A gist of a proof as to why the above construction works. If the TM, given an input w, runs on w moving right, and say reads n blank characters beyond the word w to reach a state q, then the NFA simulate the TM by reading w, then will jump from the 0'th copy to the 1'st copy of the state, and simulate the TM moving right using ϵ transitions and will be able to reach the same state q (i.e. reach $(q, 1)$). Also, it is clear that the NFA does not do anything other than the above simulation. Hence it accepts precisely those words that the TM accepts.

4. Enumerate lexicographically [**Category**: Proof., **Points**: 20]

   Recall the definition of an enumerator(Sipser p. 152). Prove the following theorem which extends Theorem 3.21 in the Sipser book:

A language is *Turing-decidable* if and only if some enumerator enumerates it in *lexicographic* order.

Solution:

We show the proof in both directions.

($\Rightarrow$) If TM M decides a language A, we can construct the following enumerator E for A. Let $s_1, s_2, s_3, \ldots$ be the list of all possible strings in Σ^* in lexicographic order. Let us fix a computable enumeration, i.e. there is a TM L that outputs s_i when given input i.

Then $E =$ "

1 Ignore the input.

2 Repeat the following for $i = 1, 2, 3, \ldots$

3 Compute s_i using L, run M on s_i. If M accepts, print out s_i."

Note that in Step 3, since M is a decider, it always halts with accept or reject. Then E enumerates all strings in lexicographical order, and prints out a string w iff the string is in A. Thus E prints all strings of A in lexicographical order.

($\Leftarrow$) Using an enumerator E that enumerates a language A in lexicographic order, we will construct a TM M that decides A. We consider two cases: if A is finite, it is Turing-decidable because all finite languages are Turing-decidable; if A is infinite, the TM M works in the following way: On input w,

1 Run E until a string greater than w in lexicographic order appears.

2 Every time that E outputs a string, compare it with w. If the string is the same as w, *accept*.

3 When E has output a string that is lexicographically greater than w, *reject*.

Clearly, if M accepts w, then w is enumerated by E.

Note that the run of M eventually stops (though E never stops), since A is infinite and hence E must at some point output w or a string lexicographically greater than w. If a string greater than w is output, and w has not appeared yet, it will never appear, because E enumerates strings in A in lexicographical order; hence M can safely reject w. This shows that if E enumerates w, then M accepts w.

Thus M indeed decides A.

5. (ExtraCredit)Two-Phase TM [**Category**: Comprehension, **Points**: 20]

   We define a *Two-Phase TM*, $M = (Q, \Sigma, \Gamma, \delta_0, \delta_1, q_0, q_1, F_0, F_1)$ as follows:
   Q is the set of states of this machine (a finite set).
   Σ and Γ are input and tape alphabet respectively ($\textvisiblespace \in \Gamma$ and $\Sigma \subseteq \Gamma - \{\textvisiblespace\}$).
   $q_0, q_1 \in Q$ and $F_0, F_1 \subseteq Q$.
   $\delta_0 : Q \times \Gamma \to Q \times \Gamma - \{\textvisiblespace\}$ and $\delta_1 : Q \times \Gamma \to Q$.

M has two tapes:  $_1$ and $_2$. At the beginning of the computation, the input string is written at the start of $_1$ and the two other tapes are blank. The head of each tape is initially located at the beginning of that tape. The computation has two phases (starting with phase 1 and in state q_0). The following recipe tells you how M operates in each phase:

Phase 1: Assume that M is in state q and the head of $_1$ is on top of symbol a. Assume $\delta_0(q, a) = (p, b)$. If $p \notin F_0$, then the head of $_1$ just goes one step to the right, the head of $_2$ first writes b on $_2$ and then goes one step to the right, and finally state of M changes to p and M continues to operate in Phase 1. If $p \in F_0$, then the head of $_2$ writes b on the tape and immediately jumps to the beginning of the tape, state of M changes to q_1 and Phase 1 ends (which means that Phase 2 starts).

Phase 2: Assume that M is in state q and the head of $_2$ is on top of symbol a. If $a = \textvisiblespace$, then the computation stops, M accepts iff $q \in F_1$. Otherwise assume $\delta_1(q, a) = p$. Then the head of $_2$ progresses one step to the right and M switches to state p. M will continue to operate in Phase 2.

As usual, we define $L(M)$ to be the set of all those strings in Σ^* that M accepts. Prove that $L(M)$ is regular.