CS281 Spring 2010: Homework 2

Due Wednesday, April 21st – SOLUTIONS

1. **Exercise 9.1.3:** Here are two definitions of languages that are similar to the definition of L_d, yet different from that language. For each, show that the language is not accepted by a Turing machine, using a diagonalization-type argument. Note that you cannot develop an argument based on the diagonal itself, but must find another infinite sequence of points in the matrix suggested by Fig. 9.1.

   a) The set A of all w_i such that w_i is not accepted by M_{2i}.

   b) The set B of all w_i such that w_{2i} is not accepted by M_i.

   Essentially we wish to use lines of slope -2 and $-1/2$ in the matrix respectively for (a) and (b) (in place of the diagonal, which has a slope of -1). However we must be careful in (b) since the string w_i we are considering for our set is *not* the same as the string whose acceptance we test.

   a) Suppose A is accepted by some machine. We may assume there is at least one transition defined for this machine, because otherwise we could add in a trivial one (from an extra state that never gets reached, to anywhere). Thus, using the enumeration scheme defined in Hopcroft we have an index ending in 0 for this machine; i.e., our machine is M_{2j} for some positive integer j. If we now consider the string w_j we find: $w_j \in A$ iff M_{2j} accepts w_j iff $w_j \notin A$, a contradiction. So in fact there can be no machine accepting A.

   b) Let B' be the set of all w_{2i} such that w_{2i} is not accepted by M_i. Clearly B is not accepted by any Turing machine (this is even more straightforward than (a), as any index j for such a machine allows us to look at w_{2j} and get a contradiction). On the other hand, if B were accepted by some machine M then we could construct a machine M' that accepts B' as follows: given a string of odd index, reject; given a string of even index, run M on the string with half that index. This implies there is no TM that accepts B.

2. **Exercise 9.2.4:** Let $L_1, L_2, \ldots, L_k$ be a collection of languages over the alphabet Σ such that:

   1. For all $i \neq j$, $L_i \cap L_j = \emptyset$; i.e. no string is in two of the lnaguages.
   2. $L_1 \cup L_2 \cup \ldots \cup L_k = \Sigma^*$; i.e., every string is in one of the languages.
   3. Each of the languages L_i, for $i = 1, 2, \ldots, k$ is recursively enumerable.

   Prove that each of the languages is therefore recursive. Note that for each i, $\bar{L}_i = \bigcup_{j \neq i} L_j$ and so is $\bar{L}_i$ is r.e. by question 3 (using induction to extend that result to finite unions). Thus L_i is recursive (since it and its complement are both r.e. - result proved in class).

3. **Exercise 9.2.6 (a) (b):** We have not discussed closure properties of the recursive langages or the RE languages, other than our discussion of complementation in Section 9.2.2. Tell whether the recursive languages and / or the RE languages are closed under the following operations. You may give informal, but clear, constructions to show closure.

   a) Union.

   b) Intersection.

   a) *The union of two r.e. sets is r.e. and the union of two recursive sets is recursive.* Suppose M_1 and M_2 are machines accepting (resp. deciding) the r.e. (resp. recursive) languages L_1 and L_2. Then define a new machine M which runs M_1 and M_2 and accepts if either of the M_i accepts. Then clearly M acccepts exactly those strings from $L_1 \cup L_2$. Moreover, in the case of recursive L_i, we may force M to halt if both of the M_i have halted. This does not change acceptance so we still have $L(M) = L_1 \cup L_2$ but now our machine is guaranteed to halt.

   b) *The intersection of two r.e. sets is r.e. and the intersection of two recursive sets is recursive.* Suppose M_1 and M_2 are machines accepting (resp. deciding) the r.e. (resp. recursive) languages L_1 and L_2. Then define a new machine M which runs M_1 and M_2 and accepts if both of the M_i accept. Then clearly M acccepts exactly those strings from $L_1 \cap L_2$. Moreover, in the case of recursive L_i, we may force M to halt if both of the M_i have halted. This does not change acceptance so we still have $L(M) = L_1 \cap L_2$ but now our machine is guaranteed to halt.

4. **Exercise 9.3.7:** Show that the following problems are not recursively enumerable:

   a) The set S of pairs (M, w) such that TM M, started with input w, does not halt.

   b) The set T of pairs (M_1, M_2) such that $L(M_1) \cap L(M_2) = \emptyset$.

   c) The set V of triples (M_1, M_2, M_3) such that $L(M_1) = L(M_2)L(M_3)$; i.e. the language of the first is the concatenation of the languages of the other two TM's.

   a) It is mentioned in Hopcroft (p. 390) that the Halting Problem

   $$\{(M, w) \mid \text{M halts on input } w\}$$

   is r.e. but not recursive. Note that S is the complement of this problem, so if S were r.e. then both the Halting Problem and its complement would be r.e., hence by the result shown in class, recursive, a contradiction.

   b) It is shown in Hopcroft (Theorem 9.6) that the complement of the set $L_u = \{(M, w) \mid w \in L(M)\}$ is not r.e. (specifically Hopcroft reduces the diagonalization language L_d to $\bar{L}_u$). Thus there is no TM which accepts $\bar{L}_u$. For each string w,

let $N(w)$ denote a particular machine which accepts $\{w\}$ (it need only compare its input character-by-character with a stored copy of w). Suppose there were a machine M accepting T. Then we could define a new machine as follows: on input (M, w), run M on the pair $(M, N(w))$. This machine would accept iff $L(M) \cap L(N(w)) = \emptyset$, i.e. iff $w \notin L(M)$ because $L(N(w)) = \{w\}$. In other words this machine would accept $\bar{L}_u$, a contradiction.

c) Let N be a machine deciding the empty set (for example a machine with no transitions). So $L(N) = \emptyset$. Now let $L_e = \{M \mid L(M) = \emptyset\}$ as in Hopcroft. This is non r.e. by Theorem 9.10. Suppose there were a machine M accepting V then we could define a new machine as follows: on input M_1, run M on the triple (M_1, N, N). This machine would accept M_1 iff $L(M_1) = L(N)L(N) = \emptyset$, a contradiction to Theorem 9.10.