CS3231
Tutorial 9

1. (a) Suppose L is RE and is accepted by TM M. Let w_i denote the string with code i.

   Now, there exists a Turing Machine M' defined as follows:

   M':

   For $t = 0$ to ∞ do:

   For $i = 0$ to t do:

   If $M(w_i)$ accepted within t steps, then write "w_i," at the end of the currently written part of the output tape.

   EndFor

   EndFor

   Thus, M' enumerates the elements of L.

   Now suppose M enumerates L as in the question.

   Then define M' as follows:

   $M'(x)$:

   For $t = 0$ to ∞ do:

   Run M for t steps. If within these t steps, in the output tape it writes "x," where to the left of x is either another comma or a beginning of the output tape, then accept.

   EndFor

   It is easy to verify that M' accepts the language enumerated by M.

   (b) If the listing is finite, then clearly L is recursive (see Q5(a)). If the listing is infinite, then suppose the enumeration $x_1, x_2, \ldots$ as in the question is given. Then, consider the following decision procedure for L:

   On input y, search for least n such that $x_n > y$. Then, $y \in L$ if and only if $y \in \{x_1, x_2, \ldots, x_n\}$.

2. (sketch): A one tape Turing machine can accept the language by, first ensuring that the string has exactly one c, and then matching the characters to the left of c with the characters to the right of c, one by one. This takes $O(n^2)$ time, where n is the length of the input.

   A two tape Turing Machine can first check if the string has exactly one c, and then copying the string to the second tape. Then, it can check whether the string is same as its reverse by comparing the strings on the two tapes, starting at left end on one tape, and right end of another tape. This takes linear time.

3. (a) Suppose M_1 accepts L_1 and M_2 accepts L_2. Then, define the following machine M':

   $M'(x)$

   For $t = 0$ to ∞ do

   If $M_1(x)$ accepts within t steps or $M_2(x)$ accepts within t steps then accept.

   End For

   It is easy to verify that the above machine M' accepts $L_1 \cup L_2$.

   Now define the following machine M''

   $M''(x)$

   For $t = 0$ to ∞ do

   If $M_1(x)$ accepts within t steps and $M_2(x)$ accepts within t steps then accept.

   End For

   It is easy to verify that the above machine M' accepts $L_1 \cap L_2$.

   (b) If L_1, L_2 are recursive, then $L_1, L_2, \overline{L_1}, \overline{L_2}$ are all RE, by result done in class.

   Thus, $L_1 \cap L_2$ and $\overline{L_1 \cap L_2} = \overline{L_1} \cup \overline{L_2}$ are both RE by part (a). Thus, $L_1 \cap L_2$ is recursive by result done in class.

   Similarly, $L_1 \cup L_2$ and $\overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2}$ are both RE by part (a). Thus, $L_1 \cup L_2$ is recursive by result done in class.

4. As L_1 is recursive, $\overline{L_1}$ is also recursive and RE.

   Now $L_2 - L_1 = L_2 \cap \overline{L_1}$. As L_2 and $\overline{L_1}$ are both RE. Thus, by Q3, so is $L_2 \cap \overline{L_1} = L_2 - L_1$.

   In general, $L_1 - L_2$ may not even be RE. For example, L_1 may be Σ^* and L_2 may be L_u. However, the complement of $L_1 - L_2$, that is $\overline{L_1 - L_2}$, is RE (as this equals: $\overline{L_1} \cup L_2$, which is union of two RE sets).

   Ofcourse, in special cases, $L_1 - L_2$ may be recursive, for example when L_2 itself is recursive.

5. (a) Every finite language is regular, and thus context free and recursive (recall the algorithm for CFL we did in class).

   (b) Follows from part (a) and the theorem that complement of a recursive language is recursive.

   (c) Suppose L is recursive and D is a finite language. Then, L, D, $\overline{D}$, $\overline{L}$ are all recursive (and hence RE).

   Thus, $(L \cap \overline{D}) \cup (D \cap \overline{L})$, as well as $(L \cap D) \cup (\overline{L} \cap \overline{D})$ are also RE. This, implies that $L \mathbf{\Delta} D = (L \cap \overline{D}) \cup (D \cap \overline{L})$ is recursive.

   (d) Suppose by way of contradiction that M halts on all but finitely many inputs. Suppose D is the set of finitely many inputs on which M does not halt.

Define $M'(x)$ as follows:

$M'(x)$:

If $x \in D$, then halt and reject.

If $x \notin D$, then simulate $M(x)$. Accept if and only if $M(x)$ accepts.

End

Now, M' halts on all the inputs and accepts exactly the language L. However, as L is not recusive, no such M' can exist.

Thus, M does not halt on infinitely many inputs.

6. (a): Suppose M accepts L and halts on all the inputs.

Define M' as follows.

$M'(x)$:

Simulate $M(xx)$. If $M(xx)$ accepts, then accept. Otherwise reject.

End

It is easy to verify that M' accepts $\{x|xx \in L\}$ and halts on all the inputs.

(b): Suppose M accepts L and halts on all the inputs.

Define M' as follows.

$M'(x)$:

For each prefix y of x do:

Siumalte $M(y)$. If $M(y)$ accepts, then accept.

End for

If none of above accepts, then reject.

End

It is easy to verify that M' accepts $\{x|$ some prefix of x is in $L\}$ and halts on all the inputs.

7: Informally done in class. In the UTM construction, use extra tape to keep t. While simulating Turing Machine M_i on input w_j, after every simulated step, decrease t by 1. If M_i halts and accepts w_j before t becomes 0, then accept. Else (i.e., if either M_i rejects within t steps, or does not halt in t steps) reject.

8: Suppose by way of contradiction that halting problem is decidable. Let F be a recursive function which on input i and j decides if Turing Machine M_i halts on input w_j. Then, the following algorithm accepts L_d, a contradiction to result done in class.

$L_d(w)$:

1. First compute i such that $w = w_i$.
2. If $F(i, i)$ returns "No", then accept.

   Otherwise, run $M_i(w_i)$. If $M_i(w_i)$ accepts, then reject. Else accept. (Note that $M_i(w_i)$ must halt if we reach this step).

   End $L_d(w)$:

   Now, consider any input $w = w_i$ to L_d above. If $M_i(w_i)$ does not halt, then $F(i, i)$ would return "No" and $w_i \notin L(M_i)$ and thus in L_d. Thus by step 2, If case, the above algorithm accepts correctly.

   If $M_i(w_i)$ halts, then $F(i, i)$ would return "Yes", and thus the above algorithm will go into the Otherwise clause of step 2. Here $M_i(w_i)$ will halt, and thus the algorithm correctly accepts if and only if $M_i(w_i)$ does not accept. Thus the above algorithm correctly accepts L_d. A contradiction to the result done in class.