Theory of Computation Assignment no. 9

Goktug Saatcioglu

(1) (i) The CFG for A is given by

$$S \to aSa \mid bSb \mid aSb \mid bSa \mid a.$$

(iii) The CFG for C is given by

$$\begin{aligned} S &\to T \mid U \mid V \\ T &\to aTb \mid aT \mid a \\ U &\to aUb \mid Ub \mid b \\ V &\to WbWaW \\ W &\to aW \mid bW \mid \lambda. \end{aligned}$$

(iv) The CFG for D is given by

$$S \to aSa \mid bSa \mid a \mid b \mid \lambda.$$

(vii) Assuming the word w is also an initial string of w, the CFG for H is given by

$$S \to aS \mid aSbS \mid \lambda.$$

(2) Consider the Turing machine given by

$$M = (\Sigma, \Gamma, Q, q_0, \delta, F)$$

where the input alphabet $\Sigma = \{a, b\}$, the tape alphabet $\Gamma = \Sigma \cup \llcorner$, the set of states $Q = \{q_0, q_a, q_b\} \cup F$ and the set of final states $F = \{q_{yes}, q_{no}\}$. Thus, we need to define the transition function such that we can move an input string $w \in \Sigma^*$ one cell to the right and stop on q_{yes}. Consider the following definition of the transition function

$$\begin{aligned} \forall s \in \Sigma \quad \delta(q_0, s) &= (q_s, \llcorner, R) \\ \forall s, t \in \Sigma \quad \delta(q_s, t) &= (q_t, s, R) \\ \forall s \in \Sigma \quad \delta(q_s, \llcorner) &= (q_{yes}, s, R) \\ \delta(q_0, \llcorner) &= (q_{yes}, \llcorner, R). \end{aligned}$$

With this transition function M will remember the letter it just read by transitioining into a state q_s marked by the letter s. If we are just beginning reading then we transition into the state marked by the letter we read and replace what was read by the space character. From this point, if we read more letters then we transition to the state represented by that letter and write to the tape the letter of the current state. Once we reach the end we can write the letter of the current state to the tape and transition to q_{yes}. We also add the last transition rule to accept empty inputs as this is also a valid input and the processing is to simply transition into q_{yes} and write the space character to tape. Thus, M moves the whole string one cell to the right and then stops on q_{yes}. Note that we never reach q_{no} since there is no string where we should reject and this is represented as an unreachable node if we were to write M as a graph. Finally, it is easy to see that we can extend our implementation to any alphabet by simply replacing the current Q with $Q' = q_0 \cup F \cup \{q_s \mid s \in \Sigma\}$.

(3) Let A, B be two languages such that $A, B \in RE$. Then we know that there exist a Turing machine M_A that identifies A and a Turing machine M_B that identifies B. We can create a Turing machine $M_{A\circ B}$ that identifes $A \circ B$ (the concatenation of A and B) by using M_A and M_B. Observe that a string w that has n letters can be written as a concatenation of two strings $w = xy$ in $n+1$ ways (or w has $n+1$ ways to partition into a concatenation). We could either solve this problem by running the machines in parallel for each partition or take an iterative approach. We do the latter and begin by splitting the tape into 6 portions. The top portion will hold the word w itself, the second portion will hold the word x (first part of the concatenation), the third portion will hold t (the current number of steps needed for simulation), the fourth portion will hold a counter to determine the number of steps for simulation, the fifth portion will hold the length of the string and the final portion will hold a counter for the possible partitions of w. We then describe the following algorithm:

1. Split the tape as described above
2. Count the length of the input word and store in fifth portion
3. For $t = 1, 2, 3, \ldots \infty$ do
   i. For each $n+1$ partitions of w do
      a. Simulate M_A on x for $\leq t$ computation steps
      b. Simulate M_B on y for $\leq t$ computation steps
      c. If both M_A and M_B accept, then $ACCEPT$

Here the loop at step 3 is managed by the tape portions 3 and 4 while the counting of the length can be done using portion 5. The loop at step i can be done by keeping another counter in portion 6 and as it is incremented we copy that many letters of portion 1 into 2 and the rest will be just spaces. This means portion 2 holds the word x and then it is easy to infer y by just checking for the first $\sqcup$ that occurs in the second portion of the tape. This algorithm identifies $A \circ B$ since if $w \in A \circ B$ then there exits some partition of $w = xy$ such that $x \in A$ and $y \in B$. If either (or both) M_A has an infinite loop or rejects then $x \notin M_A$ or M_B has an infinite loop or rejects then $y \notin M_B$ which then implies $w = xy \notin M_{A\circ B}$. This is what happens in our algorithm for this case since we only accept if both accept and are in an infinite loop otherwise. Conversely, if $x \in M_A$ and $y \in M_B$ then $w = xy \in M_{A\circ B}$ which is again what we do and only accept if both accept. Furthermore, if M_A and M_B accept they will accept in k computation steps meaning if such a partition exists we are gauranteed to accept at some point since we check for all possible partitions for t computation steps. Let t_1 be the computation steps required for M_A to accept some partition of w and let t_2 be the computation steps required for M_B to accept the same partition of w. We see that if our algorithm accepts, then it will accept a correct partition of word $w = xy$ in $t = \max(t_1, t_2)$ steps. We conclude that we have created a Turing machine that identifies $M_{A\circ B}$ which in turn implies that if $A, B \in RE$, then $A \circ B \in RE$ (i.e. RE is closed under concatenation).

(4) Suppose for the sake of contradiction that there exists a Turing machine D that decides $HALT$ (i.e. $D \in R$). We then construct a Turing machine E that runs as follows:

$E(\langle T \rangle)$:
  $DECISION \leftarrow D(\langle T \rangle \langle T \rangle)$
  if $DECISION$ is YES then infinite loop
  else E will accept (and $DECISION$ is NO)
  endif
end

or in words, E will check if the Turing machine T stops on $\langle T \rangle$ by running D on $\langle T \rangle \langle T \rangle$ and then if T stops on $\langle T \rangle$ it will go into an infinite loop and otherwise E will accept. (Note: The copying of $\langle T \rangle$ for $D(\langle T \rangle \langle T \rangle)$ can be done by using another Turing machine before running D). Now we run E on its

own description or in other words we do $E(\langle E \rangle)$. Observe the following contradiction:

$$\begin{aligned} E(\langle E \rangle) \text{ accepts} &\implies D(\langle E \rangle \langle E \rangle) \text{ rejects} \\ &\implies E(\langle E \rangle) \text{ goes into an infinite loop} \implies D(\langle E \rangle \langle E \rangle) \text{ accepts} \\ &\implies E(\langle E \rangle) \text{ accepts} \implies \dots \end{aligned}$$

or in words, if $E(\langle E \rangle)$ halts then $D(\langle E \rangle \langle E \rangle)$ does not halt and if $E(\langle E \rangle)$ does not halt then $D(\langle E \rangle \langle E \rangle)$ does halt. This is a contradiction which means our assumption is wrong and we conclude that there is no Turing machine that decides $HALT$ (i.e. $HALT \notin R$).

(5) a. To prove that $ACCEPT \in RE$ we can construct a universal Turing machine, say U. We use the encoding strategy as defined in class where the description of the Turing machine T is $\langle T \rangle$ given over the alphabet $\{0, 1, \#\}$. We can then similarly encode the input string w which is given as the string $\langle w \rangle$ over the alphabet $\{0, 1, \#\}$. We then "split" the tape into three sections where the very top portion holds $\langle M \rangle$ and $\langle w \rangle$, the second portion holds $\langle w \rangle$ so that we can simulate M and the third portion holds the encoding information of M's states where it initially holds $\langle q_0 \rangle$ (the encoding of the starting state). Then, to simulate M we search for a transition given the current state (which is stored in the third portion of the tape) using the first portion and the current letter which is found in the second portion of the tape. Upon a single step, U updates the third portion with the new state and then updates the necessary letter in the second portion. Next, we move the tape head of M to the appropriate position after applying the transition. This can be done by either using a fourth portion of the tape that will record the position of the head or using a special character such as a dot above the letter the tape of M is at. Finally, if we reach an accepting state of M as denoted in portion three of the tape then U accepts, if we reach a rejecting state of M then U rejects and if neither happens we continue with the simulation algorithm described above. From this construction we see that

1. if T accepts w, U must accept $\langle T \rangle$, $\langle w \rangle$, and
2. if T rejects w, U must reject $\langle T \rangle$, $\langle w \rangle$, and
3. if T never stops on w, then on input $\langle T \rangle$, $\langle w \rangle$ then U will also never stop.

Thus, we see that $L(U) = ACCEPT$ which in turn implies that $ACCEPT \in RE$.

b. Suppose for the sake of contradiction that there exists a Turing machine D that decides $ACCEPT$ (i.e. D in R). We then construct a Turing machine E that runs as follows:

$E(\langle T \rangle)$:
  $DECISION \leftarrow D(\langle T \rangle \langle T \rangle)$
  if $DECISION$ is YES then E will reject
  else if $DECISION$ is NO then E will accept (and $DECISION$ is NO)
  endif
end

or in words, E will check if the Turing machine T accepts $\langle T \rangle$ by running D on $\langle T \rangle \langle T \rangle$ and if T accepts $\langle T \rangle$ then E will reject. Otherwise, when D rejects then E will accept. (Note: The copying of $\langle T \rangle$ for $D(\langle T \rangle \langle T \rangle)$ can be done by using another Turing machine before running D). Now we run E on its own description or in other words we do $E(\langle E \rangle)$. Observe the following contradiction:

$$\begin{aligned} E(\langle E \rangle) \text{ accepts} &\implies D(\langle E \rangle \langle E \rangle) \text{ rejects} \\ &\implies E(\langle E \rangle) \text{ accepts} \implies D(\langle E \rangle \langle E \rangle) \text{ rejects} \\ &\implies E(\langle E \rangle) \text{ accepts} \implies \dots \end{aligned}$$

No matter what E does, it is forced to do the opposite and this is a contradiction to our initial claim. Thus, we conclude that there is no Turing machine that decides $ACCEPT$ (i.e. $ACCEPT \notin R$).