Automata Theory - Final (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. Let $\Sigma = \{a, b\}$. Write a CFG for the language $L = \{w \mid w = a^i b^{2i},\ i > 0\}$.
   Solution: The following CFG represents L:

$$\begin{array}{rcl} S & \rightarrow & abb \\ S & \rightarrow & aSbb \end{array}$$

   Note that $\epsilon \notin L$, so $S \rightarrow \epsilon$ should not be a production of this CFG. $\Box$

2. (a) Argue that the following CFG is ambiguous.
   (2 points)

$$\begin{array}{rcl} E & \rightarrow & E - E \\ E & \rightarrow & 0 \mid 1 \end{array} \quad (1)$$

   (b) Write an unambiguous CFG for the language represented by the CFG of System (1). (3 points)

   Solution:

   (a) Consider the string $w = 1 - 0 - 1 \in L(E)$.

$$\begin{array}{rcl} E & \Rightarrow & E - E \\ & \Rightarrow & E - E - E \\ & \Rightarrow & 1 - E - E \\ & \Rightarrow & 1 - 0 - E \\ & \Rightarrow & 1 - 0 - 1 \end{array}$$

   The above leftmost derivation represents a parse tree that will be evaluated as $(1-0)-1$ in a bottom-up evaluation.

$$\begin{array}{rcl} E & \Rightarrow & E - E \\ & \Rightarrow & 1 - E \\ & \Rightarrow & 1 - E - E \\ & \Rightarrow & 1 - 0 - E \\ & \Rightarrow & 1 - 0 - 1 \end{array}$$

   The above leftmost derivation represents a parse tree that will be evaluated as $1-(0-1)$ in a bottom-up evaluation. Since there are two distinct leftmost derivations for the same string w, the CFG in System (2) is ambiguous.

(b) The following CFG represents the same language as System (2) and accounts for the ambiguity.

$$\begin{array}{rcl} E & \to & E - T \mid T \\ T & \to & 0 \mid 1 \end{array}$$

$\square$

3. Let $\Sigma = \{0, 1\}$. A string $w \in \Sigma^*$ is said to be *balanced*, if it contains an equal number of $0'$s and $1'$s. Consider the CFG represented by System (2)

$$\begin{array}{rcl} S & \to & \epsilon \\ S & \to & SS \\ S & \to & 0S1 \mid 1S0 \end{array} \tag{2}$$

Argue that this CFG represents the language of *all* and *only* balanced strings over the alphabet Σ.
Hint: Recall the Prefix Theorem proved in class.

Solution: The solution consists of two parts.

We first show that the language of the grammar represnts only balanced strings, i.e., every string w which is derived from S is balanced.

Theorem 1.1 *If $S \Rightarrow^* w$, then w is balanced.*

Proof: We use induction on the number of steps used in the *shortest*, leftmost derivation of w from S.

BASIS: Let w be derived from S in exactly one step. From the production rules, it is clear that w must be ϵ and hence w is indeed balanced.

INDUCTIVE STEP: Assume that the theorem is true for all strings w, whose shortest leftomost derivation from S, takes at most n steps.

Now consider the case in which the shortest leftmost derivation of w from S take $n + 1$ steps, where $n \geq 1$. The first step of the derivation must be one of $S \Rightarrow SS$, $S \Rightarrow 0S1$ or $S \Rightarrow 1S0$.

Assume that the first step of the derivation is $S \Rightarrow 0S1$. It follows that $w = 0x1$, where x is a string in Σ^*. Since $S \Rightarrow^* w$, we must have $S \Rightarrow^* x$; however, the shortest leftmost derivation of x from S can take at most n steps. By the inductive hypothesis, it follows that x is balanced. Consequently, $w = 0x1$ is also balanced.

An identical argument can be used for the case, in which the first step of the derivation is $S \Rightarrow 1S0$.

Finally, consider the case in which the first step of the derivation is $S \Rightarrow SS$. It follows that w can be broken up into w_1w_2, such that $S \Rightarrow w_1$ and $S \Rightarrow w_2$. We cannot immediately apply the inductive hypothesis, since either w_1 or w_2 could be ϵ and therefore the length of w is not altered. However, observe that we are focussing on the *shortest* leftmost derivation of w from S. If either w_1 or w_2 is ϵ, then we have needlessly used an extra step in the derivation and hence our derivation could not have been the shortest one. It therefore follows that neither w_1 nor w_2 is ϵ. Now, the shortest leftmost derivations of w_1 and w_2 from S take strictly less than $n + 1$ steps; as per the inductive hypothesis, w_1 and w_2 are balanced. It therefore follows that $w = w_1 \cdot w_2$ is also balanced. $\square$

We now show that if w is balanced then there is a derivation of w from S.

Theorem 1.2 *Let w be a balanced binary string. Then, $S \Rightarrow^* w$.*

Proof: We use induction on the length of w.

BASIS: Let $|w| = 0$; it follows that $w = \epsilon$. Since $S \to \epsilon$ is a production of the grammar, we have $S \Rightarrow w$ and the base case is proven.

INDUCTIVE STEP: Assume that the theorem is true, whenever $|w| \leq n$. Now consider the case in which $|w| = n+1$. The following cases need to be examined:

(i) $w = 0x1$ - Observe that x must be a balanced string and since it has length at most n, we must have $S \Rightarrow^* x$. Now, w can be derived from S as follows: $S \Rightarrow 0S1 \Rightarrow^* 0x1 \Rightarrow w$.

(ii) $w = 1x0$ - This case is symmetric to the above case.

(iii) $w = 0x0$ - In class (Prefix Theorem), we showed that if a balanced binary string is of the form $0x0$, then there is some non-trivial prefix of this string that is also balanced. In other words, w can be broken up as $w_1 \cdot w_2$, where $|w_1| \leq n$ and w_1 is balanced. However, this immediately implies that w_2 is balanced as well. Thus, w can be derived from S as follows: $S \Rightarrow SS \Rightarrow^* w_1 S \Rightarrow^* w_1 w_2 \Rightarrow w$.

(iv) $w = 1x1$ - This case is symmetric to the above case.

□

□

4. In class, we proved that if Σ is a finite alphabet, then Σ^* is a countable set. What can you say about the set 2^{Σ^*}, as regards countability?

Solution: The set $P = 2^{\Sigma^*}$ is uncountable. Assume the contrary and let P be countable. It is therefore denumerable and I can enumerate the elements of P. Let $O_1 = \{P_0, P_1, \ldots\}$ denote one such enumeration.

We know that Σ^* is a countable set and hence there exists an enumeration $O_2 = \{w_0, w_1, \ldots\}$ of the elements of Σ^*.

Accordingly, we can construct the table $T = P \times \Sigma^*$, where the rows represent elements of P, as per O_1, and the columns represent the elements of Σ^*, as per O_2. Note that $T[i, j]$ represents the intersection of set P_i with string w_j. We set $T[i, j]$ to 1, if $w_i \in P_j$ and to 0, otherwise. Now consider the set P' which is constructed as follows: $w_i \in P'$, if and only if $w_i \notin P_i$. Clearly, P' is a valid subset of Σ^* and hence $P' \in P$. Accordingly, there is some number k, such that $P' = P_k$. Let us try to assign a value to $T[k, k]$. If $T[k, k] = 0$, it means that $w_k \notin P_k$. But as per the definition of P', this means that $w_k \in P'$; however, $P' = P_k$ and we have a contradiction. On the other hand, if $T[k, k] = 1$, it means that $w_k \in P_k$; therefore, w_k cannot belong to P' and hence P_k! This contradiction has arisen from the assumption that P is a countable set; it follows that P is uncountable.

□

5. Let $\Sigma = \{(,)\}$. A string $w \in \Sigma^*$ is said to be *parenthetically balanced*, if the following two conditions are met:

(a) The total number of left parentheses and right parentheses in w are equal, and

(b) In each prefix of w, the number of left parentheses is at least as large as the number of right parentheses.

For instance, "$((((()))))$" is parenthetically balanced, whereas "$)($" and "$(()$" are not. Design a Pushdown Automaton that accepts the language of parenthetically balanced strings over the alphabet Σ.

Solution: We design a PDA $P = (Q, \Sigma, T, \delta, q_0, Z_0, F)$ that accepts by final state, the language L_{bal} of parenthetically balanced strings.

Curiously enough, we do not need a PDA for this language; as will be seen, a DPDA suffices.

The idea behind the DPDA design is simple. Each time you see a left parenthesis, you push it onto the stack; likewise each time you see a right parenthesis, you pop a left parenthesis from the stack. If this process succeeds in consuming the entire input string, then the input string must be parenthetically balanced.

We thus have,

(i) $Q = \{q_0, q_1\}$,

(ii) $\Sigma = \{(,)\}$,

(iii) $T = \{(,), Z_0\}$,

(iv) $F = \{q_1\}$

The transition function δ is defined as follows:

(a) $\delta(q_0, \epsilon, Z_0) = (q_1, Z_0)$ - If the input is consumed and the marker on the stack is revealed, move to state q_1 and accept.

(b) $\delta(q_0, '(', Z_0) = (q_0, '(' Z_0)$ - Push the left parenthesis onto the stack.

(c) $\delta(q_0, '(', '(') = (q_0, '(' '(')$ - Continue to push the left parenthesis onto the stack.

(d) $\delta(q_0, ')', '(') = (q_0, \epsilon)$ - Pop a left parenthesis, whenever you see a right parenthesis.

It is important to note that:

(i) The right parenthesis $')'$ never enters the stack.

(ii) If a right parenthesis is seen with Z_0 on the stack, then the PDA simply stops, since no such transition is defined and the string is rejected.

(iii) If the input is consumed and there still exist left parentheses on the stack, then the PDA stops in state q_0 and hence the input is rejected.

Based on the above observations, it is not hard (and not necessary!) to prove that

Theorem 1.3 $w \in L_{bal} \Leftrightarrow (q_0, w, Z_0) \vdash^* (q_1, \epsilon, Z_0)$.

□

6. Design a Deterministic Turing Machine that *decides* the language $L = \{0^n \cdot 1^n, \ n \geq 0\}$.

Solution: We design a 2-tape Turing Machine to employ the following strategy:

(a) Assume that the input is given as $\triangleright x$ on the first tape.

(b) Copy x in reverse onto the second tape, so that the second tape contains the string $\triangleright x^R$.

(c) Make appropriate transitions so that both tape heads are at the leftmost ends of their respective tapes.

(d) In this configuration, sweep from left to right on both strings till a blank $\sqcup$ is reached on both strings or it is discovered that x is not of the form $0^n \cdot 1^n$, $n \geq 0$.

The details of our Turing Machine $M = (Q, \Sigma, \delta, s)$ are as follows:

(i) $Q = \{s, q_0, q_1, q_2, q_3, \text{“}yes'', \text{“}no''\}$.

(ii) $\Sigma = \{0, 1, \sqcup, \triangleright\}$.

(iii) The transition function $\delta()$ is defined by the table below:

It is important to note that there should be a total of $6 \times 4 \times 4 = 96$ entries in the transition table. We impose the condition that for all the missing entries, the Turing Machine enters state “no'' and halts.

□

$q \in Q$	$\sigma_1 \in \Sigma$	$\sigma_2 \in \Sigma$	$\delta(q, \sigma_1, \sigma_2)$
s	$\triangleright$	$\sqcup$	$(s, \triangleright, \rightarrow, \sqcup, \rightarrow)$
s	0	$\sqcup$	$(s, 0, \rightarrow, \sqcup, \rightarrow)$
s	1	$\sqcup$	$(s, 1, \rightarrow, \sqcup, \rightarrow)$
s	$\sqcup$	$\sqcup$	$(q_0, \sqcup, \leftarrow, \sqcup, \leftarrow)$
q_0	0	$\sqcup$	$(q_0, 0, \leftarrow, \sqcup, -)$
q_0	1	$\sqcup$	$(q_0, 1, \leftarrow, \sqcup, -)$
q_0	$\triangleright$	$\sqcup$	$(q_1, \triangleright, \rightarrow, \sqcup, -)$
q_1	0	$\sqcup$	$(q_1, 0, \rightarrow, 0, \leftarrow)$
q_1	1	$\sqcup$	$(q_1, 1, \rightarrow, 1, \leftarrow)$
q_1	$\sqcup$	$\triangleright$	$(q_2, \sqcup, \leftarrow, \triangleright, -)$
q_2	0	$\triangleright$	$(q_2, 0, \leftarrow, \triangleright, -)$
q_2	1	$\triangleright$	$(q_2, 1, \leftarrow, \triangleright, -)$
q_2	$\triangleright$	$\triangleright$	$(q_3, \triangleright, \rightarrow, \triangleright, \rightarrow)$
q_3	$\sqcup$	$\sqcup$	$(\text{“}yes'', \sqcup, -, \sqcup, -)$
q_3	0	1	$(q_3, 0, \rightarrow, 1, \rightarrow)$
q_3	1	0	$(q_4, 1, \rightarrow, 0, \rightarrow)$
q_4	$\sqcup$	$\sqcup$	$(\text{“}yes'', \sqcup, -, \sqcup, -)$
q_4	1	0	$(q_4, 1, \rightarrow, 0, \rightarrow)$

Table 1: A 2-tape DTM to accept $0^n \cdot 1^n,\ n \geq 0$.