

## Problem Set 3 Solutions

**Problem 1**

(a) Give a context-free grammar over the alphabet  $T = \{x, +, *, (, )\}$  corresponding to all valid polynomial expressions in the symbol  $x$ , i.e., expressions built using symbol  $x$ , binary operations  $+$ ,  $*$  and parenthesis  $(, )$  and obeying the usual syntactical rules of arithmetic. E.g.,  $(x + x) * x + x * x + ((x))$ ,  $(x + x + (x * x) * (x + (x + (x + x))))$  are valid expressions, but  $((x))$ ,  $x(+x)$ , or  $x * +(x)$  are not.

(b) Give a left-most derivation of the string  $(x + x) * (x + x * (x + x))$  using your grammar.

(c) Convert your grammar into an equivalent Push Down Automaton. (You can use generalized PDA if you like. Remember, a generalized PDA is a PDA with transition function of the form  $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \wp(Q \times \Gamma^*)$  that can push arbitrary strings  $\gamma \in \Gamma^*$  onto the stack at each transition.) Give both the formal definition and the state diagram of the PDA.

**Solution:**

(a) The formal definition for the context free grammar  $G = (V, \Sigma, R, S)$  is given below:

$$\begin{aligned} V &= \{E\} \\ \Sigma &= \{x, +, *, (, )\} \\ S &= E, \end{aligned}$$

where the rule  $R$  is specified as follows:

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E * E \\ E &\rightarrow (E) \\ E &\rightarrow x \end{aligned}$$

(b) A derivation of a string  $w$  in a grammar  $G$  is a *left-most derivation* if at every step the left-most remaining variable is the one replaced. The left-most derivation of the string  $(x + x) * (x + x * (x + x))$  is given as follows:  $E \Rightarrow E * E \Rightarrow (E) * E \Rightarrow (E + E) * E \Rightarrow (x + x) * E \Rightarrow (x + x) * (E * E) \Rightarrow (x + x) * (E + E * E) \Rightarrow (x + x) * (x + x * E) \Rightarrow (x + x) * (x + x * (E)) \Rightarrow (x + x) * (x + x * (E + E)) \Rightarrow (x + x) * (x + x * (x + x))$ . Note that some steps (i.e.  $E + E \Rightarrow x + E \Rightarrow x + x$ ) have been combined into one step for brevity.

(c) The state diagram of the generalized PDA  $M$  converted from the grammar  $G$  is shown in Figure 1.

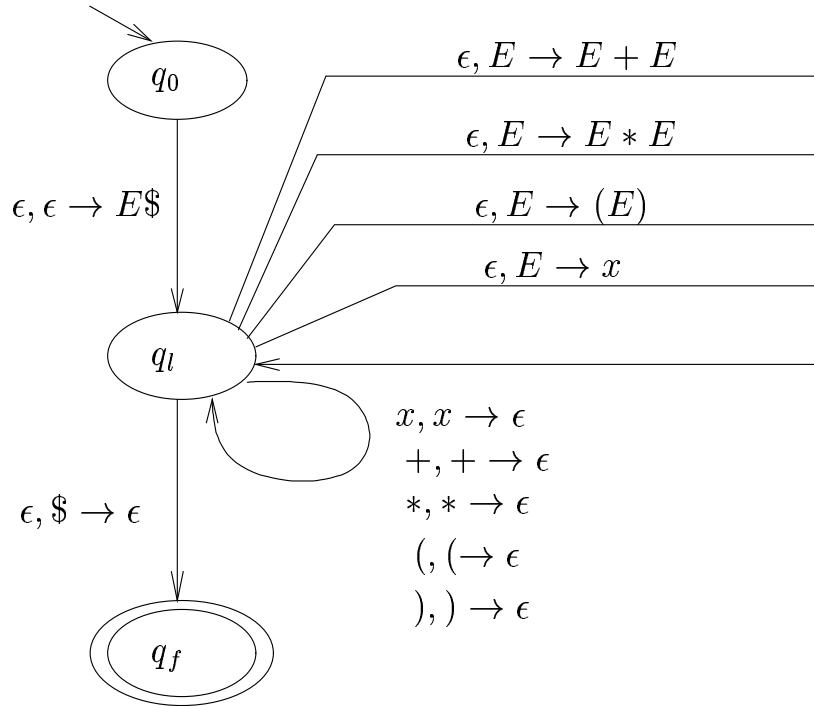


Figure 1: State diagram of the PDA  $M$  converted from the grammar  $G$

We now give the formal definition of the generalized PDA  $M = (Q, \Sigma, \Gamma, \delta, s, F)$  as follows:  $Q = \{q_0, q_l, q_f\}$ ,  $\Sigma = \{x, +, *, (, )\}$ ,  $\Gamma = \{x, +, *, (, ), E, \$\}$ ,  $s = q_0$ ,  $F = \{q_f\}$ , and the transition function  $\delta$  is formally defined as follows:  $\delta(q_0, \epsilon, \epsilon) = \{(q_l, E\$)\}$ ,  $\delta(q_l, \epsilon, E) = \{(q_l, E + E), (q_l, E * E), (q_l, (E)), (q_l, x)\}$ ,  $\delta(q_l, \epsilon, \$) = \{(q_f, \epsilon)\}$ , and for all  $a \in \Sigma$ ,  $\delta(q_l, a, a) = \{(q_l, \epsilon)\}$ .

## Problem 2

(a) Give a Push Down Automaton for the language over the alphabet  $\{a, b\}$  consisting of all words  $w$  that contains an equal number of  $a$ 's and  $b$ 's (in any order.) E.g.,  $aaabbb$  and  $abbaabab$  are in the language, but  $aabbbb$  and  $bbaabbaabb$  are not. (Here it is enough to draw the transition diagram. No formal definition required.)

(b) Give an accepting computation of your automaton on input  $abbaba$ . (Remember, a computation is a sequence of  $w_i \in \Sigma_\epsilon$ ,  $r_i \in Q$  and  $s_i \in \Gamma^*$  satisfying the conditions in definition 2.8 from the book.)

(c) Transform the PDA into an equivalent context-free grammar.

### Solution:

(a) The language we need to consider is

$$L_2 = \{ w \in \{ a, b \}^* : n_a(w) = n_b(w) \}$$

where  $n_x(w)$  is the number of occurrences of symbol  $x$  within string  $w$ .

One possible approach to design a PDA  $P$  that recognizes language  $L_2$  consists of using the stack to keep track of the difference between the number of  $a$ 's and the number of  $b$ 's. Consequently, if the input symbol read is an  $a$  then if the stack is empty or there is an  $a$  on the top, we just push an  $a$  into it. If there is a  $b$  on the top of the stack we just pop it off (we *match* the  $a$  with a  $b$ ). In case we read a symbol  $b$ , the situation is analogous: if the stack is empty or there is a  $b$  on the top, we just push an  $b$ . If the top of the stack is a  $a$  we just pop it off (we now *match* the  $b$  with an  $a$ ).

The state transition diagram for  $P$  is given in Figure 2.

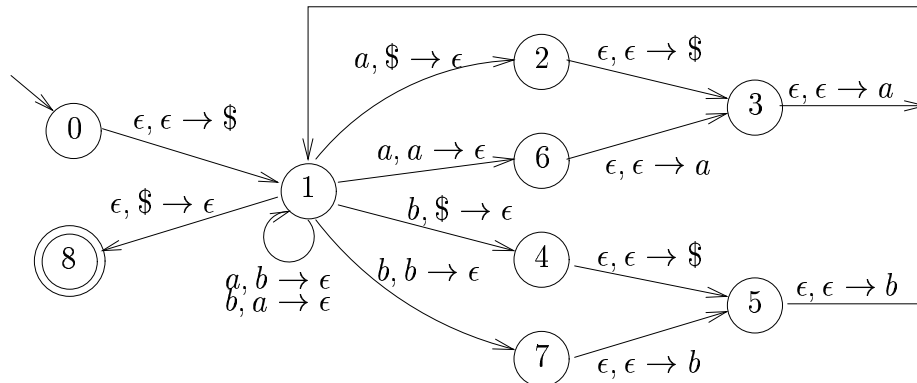


Figure 2: State transition diagram of the PDA  $P$  that recognizes the language  $L$

(b) Given a PDA  $P = (Q, \Sigma, \Gamma, \delta, q_1, F)$  and a string  $w \in \mathcal{L}(P)$ , an accepting computation of  $P$  on input  $w = w_1 \dots w_n$  is a sequence of triples  $(w_{i+1}, r_i, s_i)$ ,  $i = 0, \dots, n$ , where  $w_i \in \Sigma$ ,  $r_i \in Q$  and  $s_i \in \Gamma^*$ , such that the conditions in definition 2.8 from the book hold.

The following is an accepting computation of  $P$  on input  $w$ :

$$(\epsilon, 0, \epsilon), (\epsilon, 1, \$), (a, 2, \epsilon), (\epsilon, 3, \$), (\epsilon, 1, a\$), (b, 1, \$), (b, 4, \epsilon), (\epsilon, 5, \$), (\epsilon, 1, b\$), \\ (a, 1, \$), (b, 4, \epsilon), (\epsilon, 5, \$), (\epsilon, 1, b\$), (a, 1, \$), (\epsilon, 8, \epsilon).$$

(c) In order to transform the PDA  $P$  shown above into a CFG  $G = (V, \Sigma, R, S)$  the PDA  $P$  must meet three conditions: (1)  $P$  must have a single accept state, (2)  $P$  must empty its stack before accepting, and (3) each transition  $P$  either does a push move or a pop move, but does not do both at the same time. It is easy to see that  $P$  already meets these three conditions.

Therefore, the grammar  $G = (V, \Sigma, R, S)$  is as follows:

$$V = \{ A_{ij} : i, j = 0, \dots, 8 \} \\ \Sigma = \{ a, b \} \\ S = A_{08}$$

We separate the rules  $R$  in four sets. The first set is obtained by applying the steps (2) and (3) in the procedure described in the proof of Lemma 2.15 from the book.

$$A_{ii} \longrightarrow \epsilon \quad i = 0, \dots, 8 \\ A_{ij} \longrightarrow A_{ik}A_{kj} \quad i, j, k = 0, \dots, 8$$

The next set of rules is obtained by applying step (1) on transitions that push/pop symbol \$ into/from the stack.

$$A_{08} \longrightarrow A_{11} \\ A_{02} \longrightarrow A_{11}a \\ A_{04} \longrightarrow A_{11}b \\ A_{22} \longrightarrow A_{31}a \\ A_{24} \longrightarrow A_{31}b \\ A_{28} \longrightarrow A_{31} \\ A_{42} \longrightarrow A_{51}a \\ A_{44} \longrightarrow A_{51}b \\ A_{48} \longrightarrow A_{51}$$

**Remark:** Rules for  $A_{24}$  and  $A_{42}$  were missing from the first solutions. However, they were considered while grading.

The third set of rules is obtained by applying step (1) on transitions that push/pop symbol  $a$  into/from the stack:

$$\begin{aligned}
A_{36} &\longrightarrow A_{11}a \\
A_{31} &\longrightarrow A_{11}b \\
A_{66} &\longrightarrow A_{31}a \\
A_{61} &\longrightarrow A_{31}b
\end{aligned}$$

And, the last set of rules is obtained by applying step (1) on transitions that push/pop symbol  $b$  into/from the stack:

$$\begin{aligned}
A_{51} &\longrightarrow A_{11}a \\
A_{57} &\longrightarrow A_{11}b \\
A_{71} &\longrightarrow A_{51}a \\
A_{77} &\longrightarrow A_{51}b
\end{aligned}$$

The four sets combined form the set of rules  $R$  of grammar  $G$ .

---

### Problem 3

In class we saw that the intersection of context-free languages is not always context-free, i.e., context-free languages are not closed under intersection. Prove that the intersection of a context-free language and a regular language is context-free.

[*Hint:* given a DFA  $M$  and a PDA  $N$ , show how  $M$  and  $N$  can be combined into a single PDA  $P$  such that the language accepted by  $P$  is the intersection of the languages accepted by  $M$  and  $N$ . *Additional Hint:* use construction similar to the one in Theorem 1.12 in the book for the intersection of regular languages. You will need to augment the construction with a stack.]

**Solution:** We prove that the intersection of a context-free language and a regular language is context-free by constructing a PDA that recognizes the intersection.

Let  $M = (Q_1, \Sigma, \delta_1, s_1, F_1)$  be a DFA that recognizes a regular language  $L_1$  and let  $N = (Q_2, \Sigma, \Gamma, \delta_2, s_2, F_2)$  be a PDA that recognizes a context-free language  $L_2$ . We construct a PDA  $P = (Q, \Sigma, \Gamma, \delta, s, F)$  that recognizes the intersection of the two languages,  $L_1 \cap L_2$ , in a similar manner as the one shown for the intersection of regular languages. The formal definition is given below:

1.  $Q = \{(r_1, r_2) : r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$ , the set of states, is the Cartesian product,  $Q_1 \times Q_2$ , of sets  $Q_1$  and  $Q_2$ .
2.  $\Sigma$ , the input alphabet, is the same as in  $M$  and  $N$ .
3.  $\Gamma$ , the stack alphabet, is the same as in the given PDA  $N$ .

4.  $\delta$ , the transition function, is defined as follows. For each  $(r_1, r_2) \in Q$ , each  $a \in \Sigma$ , and each  $x \in \Gamma \cup \{\epsilon\}$ , let

$$\delta((r_1, r_2), a, x) = \{((r'_1, r'_2), x') : r'_1 = \delta_1(r_1, a) \text{ and } (r'_2, x') \in \delta_2(r_2, a, x)\}.$$

Note  $\delta$  gets a state  $(r_1, r_2)$  of  $P$  (which is a pair of states from  $M$  and  $N$ ,  $r_1 \in Q_1$  and  $r_2 \in Q_2$ ), together with an input symbol  $a \in \Sigma$  and a stack symbol  $x \in \Gamma$ , and returns a set of all possible next state and stack symbol pairs  $((r'_1, r'_2), x')$ . When the input symbol is  $\epsilon$ , for which a DFA does not have a transition function defined, its next transition state  $r'_1$  for the DFA on input  $\epsilon$  remains the same as the current state  $r_1$  (i.e.  $\delta_1(r_1, \epsilon) = r_1$ ). Hence, we define the transition function on input  $\epsilon$  as follows.

$$\delta((r_1, r_2), \epsilon, x) = \{((r_1, r'_2), x') : (r'_2, x') \in \delta_2(r_2, a, x)\}.$$

5.  $s \in Q$ , the start state, is the pair  $(s_1, s_2)$ .
6.  $F$ , the set of accept states, is the set of pairs in which each member is an accept state of  $M$  and  $N$ . We can write it as

$$F = \{(r_1, r_2) : r_1 \in F_1 \text{ and } r_2 \in F_2\}.$$

This expression is the same as  $F = F_1 \times F_2$ .

## Problem 4

In the previous homework assignment, you proved that the language  $L_4 = \{a^n b^m c^k \mid \min(n, m) \leq k\}$  is context-free. In this problem we examine the language

$$L = \{a^n b^m c^k \mid \max(n, m) \leq k\}.$$

Despite the similarity between the two definitions, it turns out that  $L$  is not context-free. Prove that  $L$  is not a context-free language using the pumping lemma for context-free languages.

**Solution:** Assume by contradiction that  $L$  is context free (CF). Then, by PL4CFL, we know that there exists a pumping length  $p > 0$  such that any word  $w \in L$ ,  $|w| \geq p$ , can be partitioned as  $uvxyz = w$  (with  $|vxy| \leq p$  and  $|uy| > 0$ ) in such a way that, for any  $i \geq 0$ , the word  $uv^i xy^i z$  also belongs to  $L$ .

Now, consider the string  $w = a^p b^p c^p$ . Clearly  $w \in L$  and  $|w| \geq p$ . Moreover, because the condition  $|vxy| \leq p$ , any partition of  $w$  into  $uvxyz$  must fall into exactly one of the following cases :

1.  $v$  and  $y$  both contain only the same symbol, that is, either (A)  $v = a^k$  and  $y = a^j$ , or (B)  $v = b^k$  and  $y = b^j$ , or (C)  $v = c^k$  and  $y = c^j$ .

First, consider the sub-cases (A) and (B). By pumping up twice this word (that is,  $i = 2$ ) we get the word  $w' = a^{p+\alpha}b^{p+\beta}c^p$ , where either  $\alpha$  or  $\beta$  are equal to  $k + j > 0$ ; hence, there is more  $a$ 's or  $b$ 's than  $c$ 's. This is a contradiction since the exponent of  $c$  is equal to  $p$ .

Now, consider the sub-case (C). By pumping down this word (which means  $i = 0$ ) into  $w' = uxz$  we get that  $w' = a^p b^p c^\alpha$  where  $\alpha = p - (k + j) < p$ . Therefore, string  $w'$  cannot belong to the language  $L$ . We get a contradiction for this sub-case too.

2.  $v = a^k$  and  $y = b^j$  By pumping up twice this word (that is,  $i = 2$ ) we get the word  $w' = a^{p+\alpha}b^{p+\beta}c^p$ , where  $\alpha = p + k \geq p$  and  $\beta = p + j \geq p$ . Since  $k + j > 0$  either  $\alpha > p$  or  $\beta > p$ . This is a contradiction since the exponent of  $c$  is exactly  $p$ .
3.  $v = b^k$  and  $y = c^j$ . If  $j = 0$  then  $k > 0$  (since  $|vy| > 0$ ). By pumping up twice we get a word  $w'$  of a form  $a^p b^{p+k} c^p$  which does not belong to  $L$ .

If  $j > 0$  we consider two sub-cases: (A)  $k \leq j$  and (B)  $k > j$ . If  $k \leq j$ , then by pumping down ( $i = 0$ )  $w$  onto  $w' = uxz$  we get that  $w' = a^p b^{p-k} c^{p-j} \notin L$  since  $p - j \leq p - k$ . If  $k > j$ , then by pumping up ( $i = 2$ ) we get  $w' = a^p b^{p+k} c^{p+j} \notin L$  since  $p + k > p + j$ . Both of them are contradictions.

4. Either  $v$  and  $y$  contains two different symbols, that is, one of the following sub-cases hold: (A)  $v = a^k$  and  $y = a^j b^l$ , or (B)  $v = a^l b^j$  and  $y = b^k$ , or (C)  $v = b^k$  and  $y = b^j c^l$ , or (D)  $v = b^l c^j$  and  $y = c^k$ . First of all, if  $j + l = 0$  then there must be the case that  $k > 0$  because the condition  $|vy| = k + j + l > 0$ . If that happens, we can get a contradiction for (A),(B) and (C) by pumping up twice, and for (D) by pumping down once. Now, if  $j + l > 0$  but  $j = 0$  then sub-cases (A),(B),(C) and (D) boil down to the case 2 and 3 shown above, and therefore they lead into a contradiction.

If  $j, l > 0$  then by pumping up this word (say  $i = 2$ ) into  $w' = uv^2xy^2z$  we get symbols out of order in every case. This implies that string  $w'$  cannot belong to language  $L$ . We get a contradiction for this case.

Since for any possible partition of  $w$  into  $uvxyz$  we obtain a contradiction, we have that  $L$  is not context free.

---