CSc 30400 Introduction to Theory of Computer Science

3rd Homework Set - Solutions

Due Date: 3/25

1 Practice Questions

P 1. Derivations:

(a) $E \Rightarrow T \Rightarrow F \Rightarrow N \Rightarrow 2$

(b) $E \Rightarrow E+T \Rightarrow T+F \Rightarrow F+N \Rightarrow N+3 \Rightarrow 5+3$

(c) $E \Rightarrow E+T \Rightarrow E+T+F \Rightarrow T+F+N \Rightarrow F+N+3 \Rightarrow N+7+5 \Rightarrow 3+7+5$

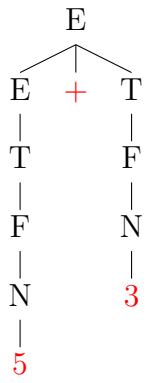
(d) $E \Rightarrow E+T \Rightarrow T+T\times F \Rightarrow F+F\times N \Rightarrow N+N\times 3 \Rightarrow 4+6\times 3$

(e) $E \Rightarrow T \Rightarrow T\times F \Rightarrow F\times N \Rightarrow (E)\times 6 \Rightarrow\Rightarrow (E+T)\times 6 \Rightarrow (T+F)\times 6 \Rightarrow (F+N)\times 6 \Rightarrow (N+5)\times 6 \Rightarrow (2+5)\times 6$

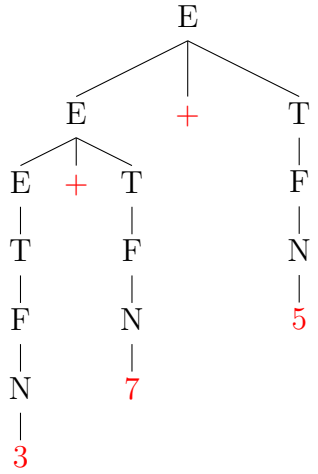
Parse Trees:

(a)

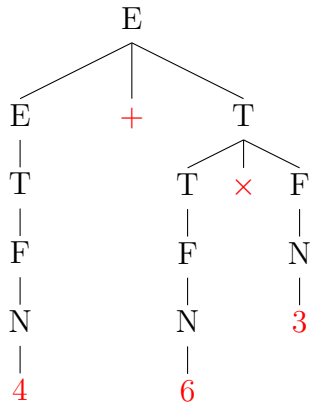
(b)



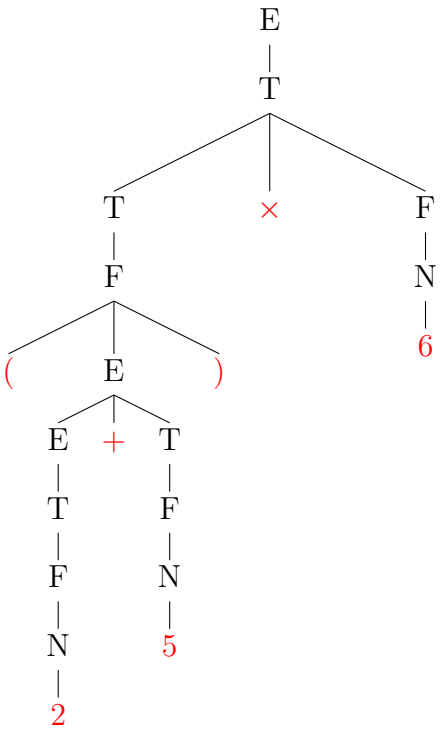
(c)



(d)



(e)



2 Easy Questions

E 1. (a) This is a regular language.

$$\begin{aligned} S &\rightarrow 0S \mid 1A \\ A &\rightarrow 0A \mid 1B \\ B &\rightarrow 0B \mid 1C \\ C &\rightarrow 0C \mid 1C \mid \varepsilon \end{aligned}$$

(b) This is not a regular language.

$$S \rightarrow 0S11 \mid \varepsilon$$

(c) This is a regular language.

$$\begin{aligned} S &\rightarrow 0Z \mid 1A \\ Z &\rightarrow 0Z \mid 1Z \mid 0 \\ A &\rightarrow 0A \mid 1A \mid 1 \end{aligned}$$

(d) This is not a regular language.

$$S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$

(e) This is not a regular language.

$$\begin{aligned} S &\rightarrow 0S0 \mid A \\ A &\rightarrow 111A \mid \varepsilon \end{aligned}$$

E 2. See figures 1 and 2.

E 3. (a) The string 0100 cannot be produced by the grammar.

0	1	0	0
A	B	A	A
-	S	B	
-	A		
B			

(b) The string 1100 can be produced by the grammar.

1	1	0	0
B	B	A	A
A	S	B	
B	A		
S			

A derivation is $S \Rightarrow BA \Rightarrow AAA \Rightarrow BBAA \Rightarrow 1100$.

E 4. The grammar is:

$$S \rightarrow aSb \mid bSa \mid \varepsilon$$

i Add new start symbol

$$\begin{aligned} S_0 &\rightarrow S \\ S &\rightarrow aSb \mid bSa \mid \varepsilon \end{aligned}$$

ii Remove ε productions.

$$\begin{aligned} S_0 &\rightarrow S \mid \varepsilon \\ S &\rightarrow aSb \mid bSa \mid ab \mid ba \end{aligned}$$

iii Remove unary productions.

$$\begin{aligned} S_0 &\rightarrow aSb \mid bSa \mid ab \mid ba \mid \varepsilon \\ S &\rightarrow aSb \mid bSa \mid ab \mid ba \end{aligned}$$

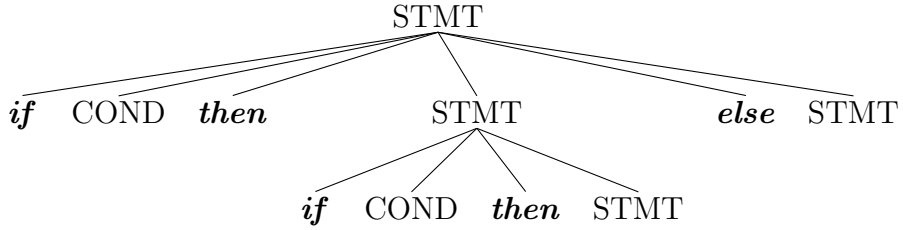
iv Add new variable for every terminal

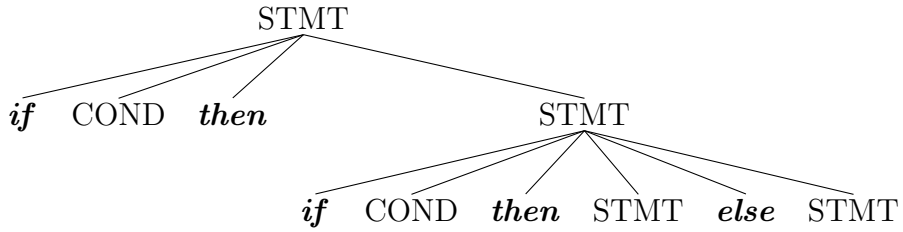
$$\begin{aligned} S_0 &\rightarrow ASB \mid BSA \mid AB \mid BA \mid \varepsilon \\ S &\rightarrow ASB \mid BSA \mid AB \mid BA \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

v Fix large rules

$$\begin{aligned} S_0 &\rightarrow CB \mid DA \mid AB \mid BA \mid \varepsilon \\ S &\rightarrow CB \mid DA \mid AB \mid BA \\ A &\rightarrow a \\ B &\rightarrow b \\ C &\rightarrow AS \\ D &\rightarrow BS \end{aligned}$$

E 5. The grammar is ambiguous. Consider the following two parts of parse trees:





It is obvious that those two trees are different and they can eventually produce the same string. In order to solve this problem we generally consider changing the language by adding terminals denoting where an instruction (the ***if*** in this example) starts and ends. We could add for example a new reserved word (terminal) ***endif*** which is going to match the corresponding ***if***.

$$\begin{aligned} < STMT > \;\; \rightarrow \;\; & if < COND > then < STMT > endif \mid \\ & if < COND > then < STMT > else < STMT > endif \end{aligned}$$

3 Hard Questions

H 1.

- A right linear grammar has rules of the form:

$$\begin{aligned} A &\rightarrow \varepsilon \\ A &\rightarrow aB \\ A &\rightarrow a \end{aligned}$$

  where $A, B \in V$ and $a \in \Sigma$. Since a can be considered as a string of just one terminal, $a \in \Sigma^*$, a right linear grammar is also an extended right linear grammar.

- In order to transform an extended right linear grammar to a right linear one we should take care of the productions of the second and the third form.

  For every production of the third form $A \rightarrow w$ with $w = w_1 w_2 \dots w_n$ and $w_1, w_2, \dots, w_n \in \Sigma$ we create the following rules:

$$\begin{aligned} A &\rightarrow w_1 A_1 \\ A_1 &\rightarrow w_2 A_2 \\ &\vdots \\ A_{n-1} &\rightarrow w_n \end{aligned}$$

For productions of the second form we distinguish between two cases:

- $w = w_1 w_2 \dots w_n$, with $w_1, w_2, \dots, w_n \in \Sigma$. Following a similar technique as we did above we eliminate the long rule $A \to w_1 w_2 \dots w_n B$ and substitute it with the rules:

$$\begin{aligned} A &\to w_1 A_1 \\ A_1 &\to w_2 A_2 \\ &\vdots \\ A_{n-1} &\to w_n B \end{aligned}$$

- $w = \varepsilon$. This rule is a unary production $A \to B$. We can eliminate unary productions by substituting B with the right part of the rules having B in the left part.

H 2. The 2-PDAs for L_1 and L_2 are shown in figures 3 and 4.

- In order to be able to recognize L_1 we should use the first stack to save the as and the second one to save the bs. Then we can match the cs with the as and bs to see if they are all equal.
- We first discuss the idea behind a 2-PDA for L_2' and then we provide a non-deterministic version of this that recognizes L_2. We use the first stack to save the first part of the input until # is reached. Then we save the second part in the second stack. We finally compare the two stacks to see if the contents are identical.

  The 2-NPDA for L_2 works in a very similar way: it guesses where the # is and divides the input in the two stacks which finally compares. If the string is of the form ww there is going to be one partition which makes the 2-PDA accept. On the other hand, for strings that are not in the language there is no partition that is going to make the two stacks match.

H 3. (a) We give a context free grammar for L_a:

$$\begin{aligned} S &\to AB \\ A &\to aA \mid \varepsilon \\ B &\to bBc \mid \varepsilon \end{aligned}$$

A context free grammar for L_c is:

$$\begin{aligned} S &\rightarrow BC \\ B &\rightarrow aBb \mid \varepsilon \\ C &\rightarrow Cc \mid \varepsilon \end{aligned}$$

Thus both L_a, L_c are context free languages.

(b) The language $L_1 = \{a^n b^n c^n : n, \geq 0\}$ is not a context free language as it was stated in H2. However $L_1 = \{a^k b^n c^n : n, k \geq 0\} \cap \{a^n b^n c^k : n, k \geq 0\} = L_a \cap L_c$. So we found an example where the intersection of two context free languages is not a context free language. Thus we proved that the context free languages are not closed under intersection.

(c) To obtain a contradiction suppose that the set of context free languages was closed under complement. Then L_a^c and L_c^c would both be context free languages. Since context free languages are closed under union we have that $L_a^c \cup L_c^c$ is a context free language. By De Morgan's law we have that $L_a^c \cup L_c^c = (L_a \cap L_c)^c = L_1^c$. Since L_1^c is context free it should be that L_1 is also context free. But question H2 states that this is not the case. Thus our initial assumption that context free languages are closed under complement doesn't hold.

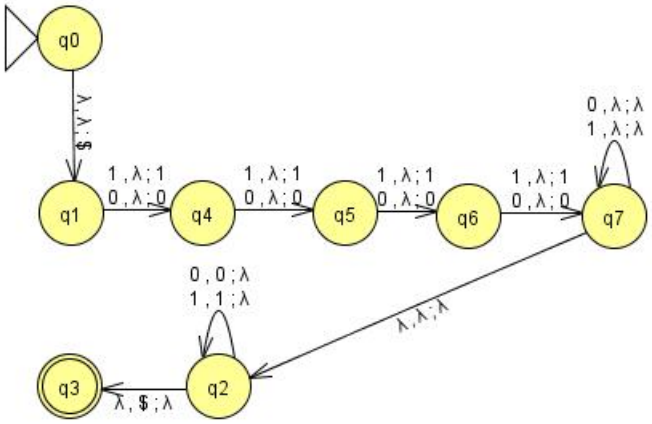


Figure 1: A PDA for L_1 of question E 2.

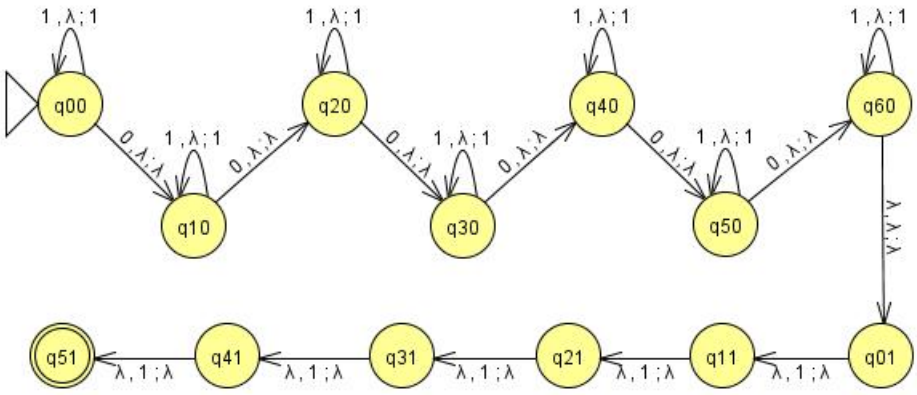


Figure 2: An PDA for L_2 of question E 2.

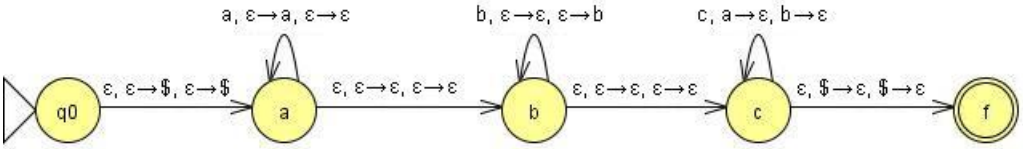


Figure 3: A 2-PDA for L_1 of question H 2.

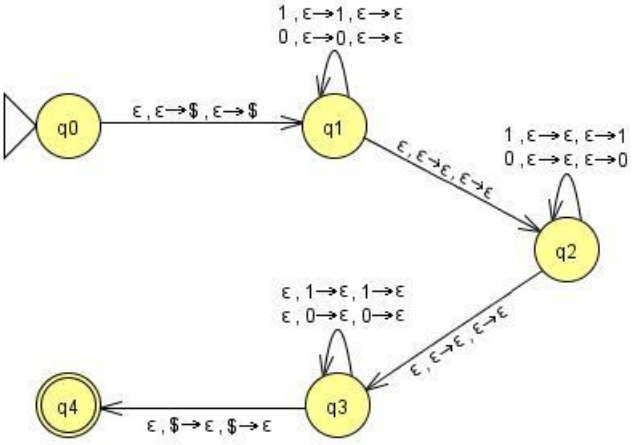


Figure 4: An 2-PDA for L_2 of question H 2.