Formal Languages and Compilers
Exercises on Syntax Analysis II with Solutions

MSc in Computer Science, University of Camerino
prof. Luca Tesei

Exercise 1

Consider the language of expressions formed by identifiers (tokens), parentheses and two **binary** operators $\oplus$ and $\otimes$. Define, illustrating all the steps, a context-free grammar that generates the expressions of this language and that encapsulates the following precedence and associativity assumptions:

- Operator $\oplus$ has precedence over $\otimes$
- Operator $\oplus$ is right-associative
- Operator $\otimes$ is left-associative

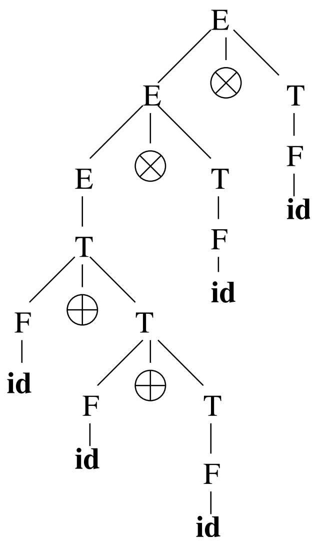
Then, draw a derivation tree for the string $\mathbf{id} \oplus \mathbf{id} \oplus \mathbf{id} \otimes \mathbf{id} \otimes \mathbf{id}$ according to the defined grammar and explain the structure of the expression according to the rules of precedence and associativity.

Solution

Let's create a non-terminal symbol for each precedence level: F for the higher level, which encompasses the operands and the parenthesised expressions, T for the level of $\oplus$ and E for the level of lowest precedence, that is the one of operator $\otimes$. E is the starting symbol. Regarding the associativity, it is sufficient to use left recursion for left associativity and right recursion for right associativity. The grammar is the following:

$$\begin{array}{lcl} E & \rightarrow & E \otimes T \mid T \\ T & \rightarrow & F \oplus T \mid F \\ F & \rightarrow & \mathbf{id} \mid (E) \end{array}$$

The derivation tree for the given string is the following:



The structure of the string according to the rules and reflected in the derivation tree can be explicitly represented using square brackets as follows:

$$[[\mathbf{id} \oplus [\mathbf{id} \oplus \mathbf{id}]] \otimes \mathbf{id}] \otimes \mathbf{id}$$

Exercise 2

Consider the following language:

$$\{a^n\, u\, b^{k-1}\, v\, c^m \mid n, k, m > 0 \text{ e } m = n + k\}$$

1. Define a context-free grammar for the language.
2. Is the language LR? Justify your answer illustrating all the steps.
3. If the language is LR then give a parsing table for a bottom-up shift-reduce parser.

Solution

The language can be equivalently expressed as $\{a^n\, u\, b^k\, v\, c\, c^k\, c^n \mid n > 0, k \geq 0\}$. Using this formulation it is easy to obtain a grammar using two left-right recursion schemes:

$$\begin{array}{lcl} S & \rightarrow & aSc \mid auBc \\ B & \rightarrow & bBc \mid vc \end{array}$$

Let's check if this grammar is SLR. If this is the case, it follows that the grammar is also LR(1). If the grammar is LR(1) then also the language is LR(1) because there exists an LR(1) grammar for it.
The collection of LR(0) items is the following:

$I_0 =$	$S' \to \cdot S$ $S \to \cdot aSc$ $S \to \cdot auBc$	$I_1 = goto(I_0, S) =$	$S' \to S\cdot$
$I_2 = goto(I_0, a) =$	$S \to a \cdot Sc$ $S \to a \cdot uBc$ $S \to \cdot aSc$ $S \to \cdot auBc$	$I_3 = goto(I_2, S) =$	$S \to aS \cdot c$
$I_4 = goto(I_2, u) =$	$S \to au \cdot Bc$ $B \to \cdot bBc$ $B \to \cdot vc$	$goto(I_2, a) = I_2$	
$I_5 = goto(I_3, a) =$	$S \to aSc\cdot$	$I_6 = goto(I_4, B) =$	$S \to auB \cdot c$
$I_7 = goto(I_4, b) =$	$B \to b \cdot Bc$ $B \to \cdot bBc$ $B \to \cdot vc$	$I_8 = goto(I_4, v) =$	$B \to v \cdot c$
$I_9 = goto(I_6, c) =$	$S \to auBc\cdot$	$I_{10} = goto(I_7, B) =$	$B \to bB \cdot c$
$goto(I_7, b) = I_7$		$goto(I_7, v) = I_8$	
$I_{11} = goto(I_8, c) =$	$B \to vc\cdot$	$I_{12} = goto(I_{10}, c) =$	$B \to bBc\cdot$

There are no conflicts in the states, thus the grammar is SLR.
Let's calculate FOLLOW(S') = $\{\$\}$, FOLLOW($S$) = $\{c, \$\}$ e FOLLOW(B) = $\{c\}$.
The parsing table is the following (the productions are numbered starting from 0):

	c	a	b	u	v	$\$$	S	B
0		s2					1	
1						acc		
2		s2		s4			3	
3	s5							
4			s7		s8			6
5	r1					r1		
6	s9							
7			s7		s8			10
8	s11							
9	r2					r2		
10	s12							
11	r4							
12	r3							

Exercise 3

Consider the language:

$$\{a^n\, b\, c \mid n > 0\} \cup \{b^n\, c\, b \mid n \geq 0\} \cup \{c\, a^n \mid n > 0\}$$

1. Define a context-free grammar for the language.
2. Is the language LL(1)? Justify your answer illustrating all the steps.
3. If the language is LL(1) give a table for a top-down predictive parser and execute the parsing of the strings cb and caa.

Solution

Let's try to write directly an LL(1) grammar. We can follow the structure of the language that presents naturally three different cases (the three sets in union). An approach could be that of choosing a different non-terminal symbol for each case. Following this idea we notice that strings starting with c could belong to both the second and the third case. However, after this first c, by looking at the following character we can resolve the ambiguity: if the following character is a b then the string belongs to case 2, while if the following character is an a then the string belongs to case 3. The resulting grammar is the following:

$$
\begin{array}{lcl}
S & \to & aA \mid bB \mid cC \\
A & \to & aA \mid bc \\
B & \to & bB \mid cb \\
C & \to & b \mid aD \\
D & \to & aD \mid \epsilon
\end{array}
$$

The FIRST sets of S, A, B, C are all disjoint and by calculating $\text{FOLLOW}(D) = \{\$\}$ we see that also for D there are no conflicts. We can conclude that the grammar is LL(1) and thus also the language is LL(1).
The table for the predictive parser is the following one:

	a	b	c	$\$$
S	$S \to aA$	$S \to bB$	$S \to cC$	
A	$A \to aA$	$A \to bc$		
B		$B \to bB$	$B \to cb$	
C	$C \to aD$	$C \to b$		
D	$D \to aD$			$D \to \epsilon$

Let's parse cb and caa:

STACK	INPUT	ACTION
$\$S$	$cb\$$	$S \to cC$
$\$Cc$	$cb\$$	match
$\$C$	$b\$$	$C \to b$
$\$b$	$b\$$	match
$	$	accept

STACK	INPUT	ACTION
$\$S$	$caa\$$	$S \to cC$
$\$Cc$	$caa\$$	match
$\$C$	$aa\$$	$C \to aD$
$\$Da$	$aa\$$	match
$\$D$	$a\$$	$D \to aD$
$\$Da$	$a\$$	match
$\$D$	$	$D \to \epsilon$
$	$	accept