Assignment 2

Solutions

Compiler Design I (Kompilatorteknik I) 2011

1 Context-free grammars

Give the definition of a context free grammar over the alphabet $\Sigma = \{a, b\}$ that describes all strings that have a different number of 'a's and 'b's.

Answer:

$S \rightarrow U|V$
$U \rightarrow TaU|TaT$
$V \rightarrow TbV|TbT$
$T \rightarrow aTbT|bTaT|\epsilon$

The intuition is that the string will have either more 'a's (non-terminal U) or more 'b's (non-terminal V). Non-terminal T produces a string with balanced 'a's and 'b's.

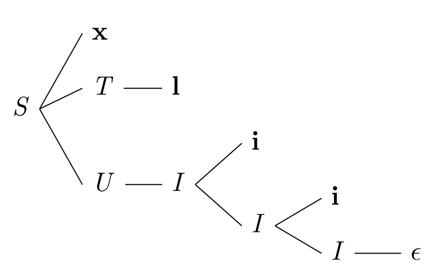
2 Parsing and semantic actions

The following context-free grammar can parse all the lowercase roman numerals from 1-99. The terminal symbols are { **c**, **l**, **x**, **v**, **i** } and the initial symbol is S. If you are unfamiliar with roman numerals, please have a look at `http://en.wikipedia.org/wiki/Roman_numerals`.

$$
\begin{array}{lcl}
S & \rightarrow & \mathbf{x}TU \mid \mathbf{l}X \mid X \\
T & \rightarrow & \mathbf{c} \mid \mathbf{l} \\
X & \rightarrow & \mathbf{x}X \mid U \\
U & \rightarrow & \mathbf{i}Y \mid \mathbf{v}I \mid I \\
Y & \rightarrow & \mathbf{x} \mid \mathbf{v} \\
I & \rightarrow & \mathbf{i}I \mid \epsilon
\end{array}
$$

1. Draw a parse tree for 42: "**xlii**"

   Answer:



2. Is this grammar ambiguous?

   Answer: No

3. Write semantic actions for each of the 14 rules in the grammar (remember $X \to A|B$ is short for $X \to A$ and $X \to B$) to calculate the decimal value of the input string. You can associate a synthesized attribute `val` to each of the non-terminals to store their value. The final value should be returned in S.`val`.

   Answer:

   Given $\mathbf{c}.val = 100$, $\mathbf{l}.val = 50$, $\mathbf{x}.val = 10$, $\mathbf{v}.val = 5$, $\mathbf{i}.val = 1$ and $\epsilon.val = 0$:

$$
\begin{array}{lcll}
S & \to & \mathbf{x}TU & \{S.val = T.val - \mathbf{x}.val + U.val\} \\
S & \to & \mathbf{l}X & \{S.val = \mathbf{l}.val + X.val\} \\
S & \to & X & \{S.val = X.val\} \\
T & \to & \mathbf{c} & \{T.val = \mathbf{c}.val\} \\
T & \to & \mathbf{l} & \{T.val = \mathbf{l}.val\} \\
X_1 & \to & \mathbf{x}X_2 & \{X_1.val = \mathbf{x}.val + X_2.val\} \\
X & \to & U & \{X.val = U.val\} \\
U & \to & \mathbf{i}Y & \{U.val = Y.val - \mathbf{i}.val\} \\
U & \to & \mathbf{v}I & \{U.val = \mathbf{v}.val + I.val\} \\
U & \to & I & \{U.val = I.val\} \\
Y & \to & \mathbf{x} & \{Y.val = \mathbf{x}.val\} \\
Y & \to & \mathbf{v} & \{Y.val = \mathbf{v}.val\} \\
I_1 & \to & \mathbf{i}I_2 & \{I_1.val = \mathbf{i}.val + I_2.val\} \\
I & \to & \epsilon & \{I.val = \epsilon.val\}
\end{array}
$$

3 LL(1) Parsers

In the following context-free grammar, the symbols **0**, **1**, **2** and **3** are terminals and S is the initial symbol.

$$
\begin{array}{lcl}
S & \to & \mathbf{0} \mid \mathbf{1}\ S\ \mathbf{2}\ S\ \mathbf{3} \mid \mathbf{1}\ A\ \mathbf{3} \\
A & \to & S \mid A\ S
\end{array}
$$

1. Explain briefly why this grammar is not LL(1).

   Answer:

   This grammar cannot be parsed by a recursive descent parser. This can be shown by the following two examples:

   - If the parser has to expand an S non-terminal and the next token is **1**, it is not possible to choose between the 2 productions from S that start with **1** with just this information. However LL(1) languages allow for just one look-ahead symbol.
   - If the parser were to make use of the $A \to AS$ production, for some look-ahead symbol, then in the new state it would still have to expand the new A with the same look-ahead, leading to an infinite loop.

2. Convert this grammar to an equivalent that is LL(1).

   Answer:

   - Factorize the S productions and eliminate immediate left recursion from the A productions:

$$
\begin{array}{lcl}
S & \to & \mathbf{0} \mid \mathbf{1}\ S' \\
S' & \to & S\ \mathbf{2}\ S\ \mathbf{3} \mid A\ \mathbf{3} \\
A & \to & S\ A' \\
A' & \to & S\ A' \mid \epsilon
\end{array}
$$

- Inline singular A production rule to uncover another possible factorization:

$$\begin{array}{lll} S & \rightarrow & \mathbf{0} \mid \mathbf{1}\ S' \\ S' & \rightarrow & S\ \mathbf{2}\ S\ \mathbf{3} \mid S\ A'\ \mathbf{3} \\ A' & \rightarrow & S\ A' \mid \epsilon \end{array}$$

- Factorize the S' production and inline the new singular S' it in S's production:

$$\begin{array}{lll} S & \rightarrow & \mathbf{0} \mid \mathbf{1}\ S\ S'' \\ S'' & \rightarrow & \mathbf{2}\ S\ \mathbf{3} \mid A'\ \mathbf{3} \\ A' & \rightarrow & S\ A' \mid \epsilon \end{array}$$

3. For the grammar of the previous subtask, construct the complete LL(1) parsing table.

Answer:

$$\begin{array}{ll|ll} \mathit{First}(S) & = \{\mathbf{0}, \mathbf{1}\} & \mathit{Follow}(S) & = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \$\} \\ \mathit{First}(S'') & = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}\} & \mathit{Follow}(S'') & = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \$\} \\ \mathit{First}(A') & = \{\mathbf{0}, \mathbf{1}, \epsilon\} & \mathit{Follow}(A') & = \{\mathbf{3}\} \end{array}$$

	0	**1**	**2**	**3**	$
S	$S \rightarrow \mathbf{0}$	$S \rightarrow \mathbf{1}\ S\ S''$			
S''	$S'' \rightarrow A'\ \mathbf{3}$	$S'' \rightarrow A'\ \mathbf{3}$	$S'' \rightarrow \mathbf{2}\ S\ \mathbf{3}$	$S'' \rightarrow A'\ \mathbf{3}$	
A'	$A' \rightarrow S\ A'$	$A' \rightarrow S\ A'$		$A' \rightarrow \epsilon$	

4. Show all the steps required to parse the input string: **1 1 0 2 0 3 0 1 0 3 3**

Answer:

Stack	Input	Action
S \$	**1 1 0 2 0 3 0 1 0 3 3** \$	$S \rightarrow \mathbf{1}\ S\ S''$
1 $S\ S''$ \$	**1 1 0 2 0 3 0 1 0 3 3** \$	terminal
$S\ S''$ \$	**1 0 2 0 3 0 1 0 3 3** \$	$S \rightarrow \mathbf{1}\ S\ S''$
1 $S\ S''\ S''$ \$	**1 0 2 0 3 0 1 0 3 3** \$	terminal
$S\ S''\ S''$ \$	**0 2 0 3 0 1 0 3 3** \$	$S \rightarrow \mathbf{0}$
0 $S''\ S''$ \$	**0 2 0 3 0 1 0 3 3** \$	terminal
$S''\ S''$ \$	**2 0 3 0 1 0 3 3** \$	$S'' \rightarrow \mathbf{2}\ S\ \mathbf{3}$
2 S **3** S'' \$	**2 0 3 0 1 0 3 3** \$	terminal
S **3** S'' \$	**0 3 0 1 0 3 3** \$	$S \rightarrow \mathbf{0}$
0 3 S'' \$	**0 3 0 1 0 3 3** \$	terminal
3 S'' \$	**3 0 1 0 3 3** \$	terminal
S'' \$	**0 1 0 3 3** \$	$S'' \rightarrow A'\ \mathbf{3}$
A' **3** \$	**0 1 0 3 3** \$	$A' \rightarrow S\ A'$
$S\ A'$ **3** \$	**0 1 0 3 3** \$	$S \rightarrow \mathbf{0}$
0 A' **3** \$	**0 1 0 3 3** \$	terminal
A' **3** \$	**1 0 3 3** \$	$A' \rightarrow S\ A'$
$S\ A'$ **3** \$	**1 0 3 3** \$	$S \rightarrow \mathbf{1}\ S\ S''$
1 $S\ S''\ A'$ **3** \$	**1 0 3 3** \$	terminal
$S\ S''\ A'$ **3** \$	**0 3 3** \$	$S \rightarrow \mathbf{0}$
0 $S''\ A'$ **3** \$	**0 3 3** \$	terminal
$S''\ A'$ **3** \$	**3 3** \$	$S'' \rightarrow A'\ \mathbf{3}$
A' **3** A' **3** \$	**3 3** \$	$A' \rightarrow \epsilon$
3 A' **3** \$	**3 3** \$	terminal
A' **3** \$	**3** \$	$A' \rightarrow \epsilon$
3 \$	**3** \$	terminal
\$	\$	ACCEPT

4 LR(1) Parsers

In the following context-free grammar, the symbols **(**, **a**, **)** and **,** are terminals. and S is the initial symbol.

(1) $S \rightarrow$ **(** L **)**
(2) $S \rightarrow$ **a**
(3) $L \rightarrow L$ **,** S
(4) $L \rightarrow S$

Because **,** is a symbol of the language we are going to use | as a separator between the core of the LR(1) items and the lookahead symbols. Lookaheads with the same core can be separated as usual with /.

1. Calculate the closure of the LR(1) item [$S \rightarrow$ **(** $\cdot$ L **)** | \$]

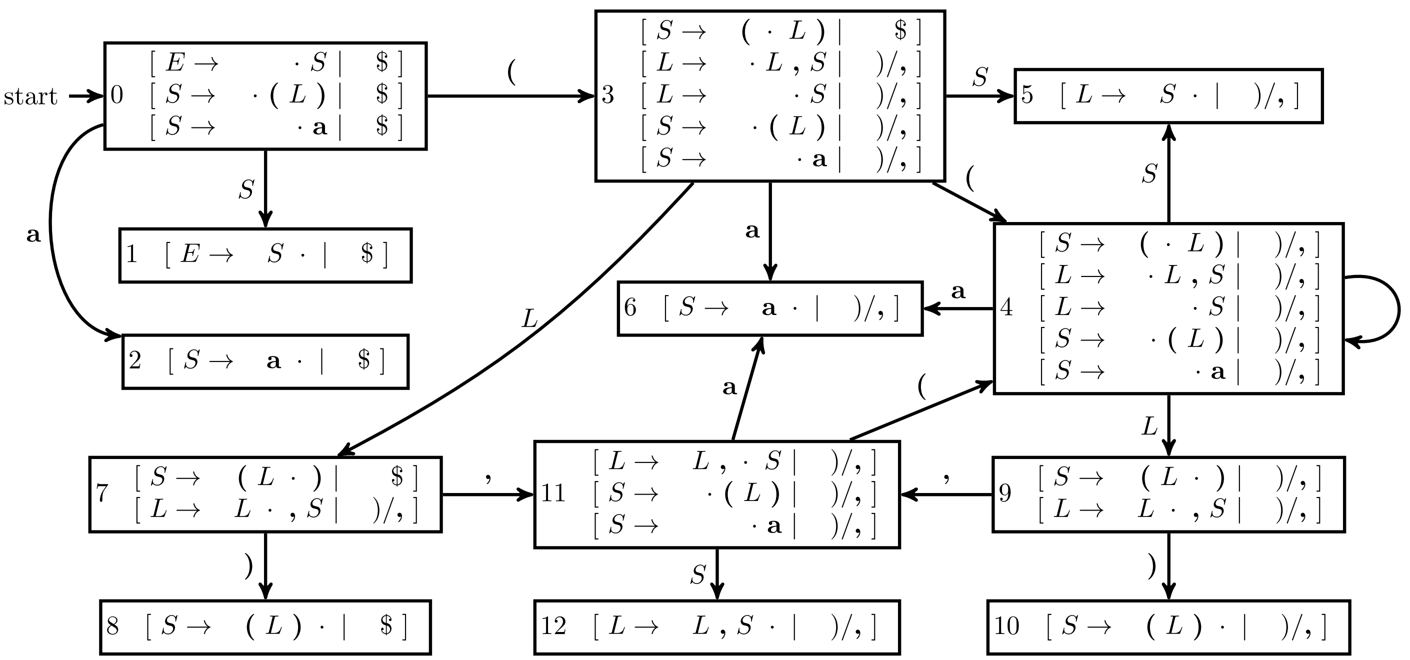
   Answer:

   [$S \rightarrow$ **(** $\cdot$ L **)** | \$]
   [$L \rightarrow$ $\cdot$ L **,** S |)/,]
   [$L \rightarrow$ $\cdot$ S |)/,]
   [$S \rightarrow$ $\cdot$ **(** L **)** |)/,]
   [$S \rightarrow$ $\cdot$ **a** |)/,]

2. Construct the full LR(1) DFA, showing all items in each state.

   Answer:

   New unique initial production: (0) $E \rightarrow S$



3. Construct the LR(1) parsing table using the DFA. For the reduce actions, please use the provided enumeration of the productions in the grammar.

Answer:

State	(	**a**	)	,	$	S	L
0	s3	s2				1	
1					ACCEPT		
2					r2		
3	s4	s6				5	7
4	s4	s6				5	9
5			r4	r4			
6			r2	r2			
7			s8	s11			
8					r1		
9			s10	s11			
10			r1	r1			
11	s4	s6				12	
12			r3	r3			

4. Show all the steps required to parse the input string: **((a , a) , a , a)**

Answer:

Stack	Symbols	Input	Action
0		((**a** , **a**) , **a** , **a**)$	shift
0,3	(	(**a** , **a**) , **a** , **a**)$	shift
0,3,4	((	**a** , **a**) , **a** , **a**)$	shift
0,3,4,6	((**a**	, **a**) , **a** , **a**)$	reduce
0,3,4,5	((S	, **a**) , **a** , **a**)$	reduce
0,3,4,9	((L	, **a**) , **a** , **a**)$	shift
0,3,4,9,11	((L ,	**a**) , **a** , **a**)$	shift
0,3,4,9,11,6	((L , **a**	) , **a** , **a**)$	reduce
0,3,4,9,11,12	((L , S	) , **a** , **a**)$	reduce
0,3,4,9	((L	) , **a** , **a**)$	shift
0,3,4,9,10	((L)	, **a** , **a**)$	reduce
0,3,5	(S	, **a** , **a**)$	reduce
0,3,7	(L	, **a** , **a**)$	shift
0,3,7,11	(L ,	**a** , **a**)$	shift
0,3,7,11,6	(L , **a**	, **a**)$	reduce
0,3,7,11,12	(L , S	, **a**)$	reduce
0,3,7	(L	, **a**)$	shift
0,3,7,11	(L ,	**a**)$	shift
0,3,7,11,6	(L , **a**	)$	reduce
0,3,7,11,12	(L , S	)$	reduce
0,3,7	(L	)$	shift
0,3,7,8	(L)	$	reduce
0,1	S	$	ACCEPT!