Assignment 2

Solutions

Compiler Design I (Kompilatorteknik I) 2012

1 Context-Free Grammars

Give the definition of a context free grammar over the alphabet $\Sigma = \{a, b, c\}$ where the amount of a's is double the amount of b's. The amount of c's is of no interest.

Answer:

$$\begin{array}{rcl} S & \rightarrow & SaSaSbS \mid SaSbSaS \mid SbSaSaS \mid C \\ C & \rightarrow & cS \mid \epsilon \end{array}$$

2 Parsing and Semantic Actions

The following grammar roughly resembles the syntax of some complex builtin datatypes in Python. The terminals are $\{\alpha, (,), [,], :, , \}$ and the initial symbol is A

(for more information see `http://docs.python.org/tutorial/datastructures.html`)

$$\begin{array}{rcl} A & \rightarrow & D \mid L \mid T \mid \alpha \\ D & \rightarrow & \{K\} \mid \{\} \\ K & \rightarrow & A : A \mid K, A : A \\ T & \rightarrow & (A, I) \mid () \\ L & \rightarrow & [I] \\ I & \rightarrow & J \mid \epsilon \\ J & \rightarrow & J, A \mid A \end{array}$$

and the string $\{((), (\alpha,)) : [\alpha, \alpha], [] : \{\alpha : [\alpha, (\alpha, \alpha)]\}\}$

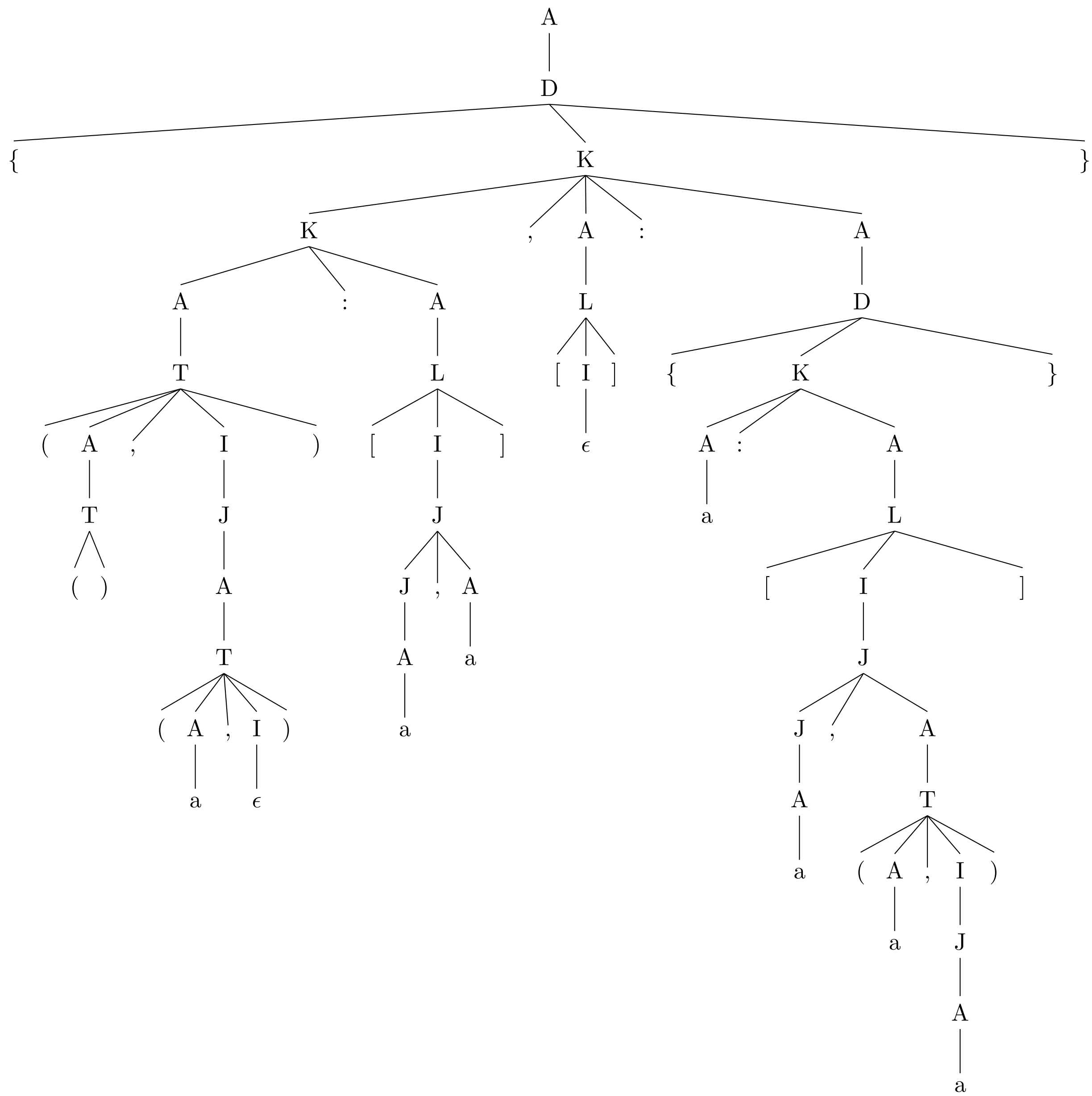
1. Give a leftmost derivation for the string.

   Answer:

$$
\begin{aligned}
A &\rightarrow D \\
&\rightarrow \{K\} \\
&\rightarrow \{K, A : A\} \\
&\rightarrow \{A : A, A : A\} \\
&\rightarrow \{T : A, A : A\} \\
&\rightarrow \{(A, I) : A, A : A\} \\
&\rightarrow \{(T, I) : A, A : A\} \\
&\rightarrow \{((), I) : A, A : A\} \\
&\rightarrow \{((), J) : A, A : A\} \\
&\rightarrow \{((), A) : A, A : A\} \\
&\rightarrow \{((), T) : A, A : A\} \\
&\rightarrow \{((), (A, I)) : A, A : A\} \\
&\rightarrow \{((), (a, I)) : A, A : A\} \\
&\rightarrow \{((), (a,)) : A, A : A\} \\
&\rightarrow \{((), (a,)) : L, A : A\} \\
&\rightarrow \{((), (a,)) : [I], A : A\} \\
&\rightarrow \{((), (a,)) : [J], A : A\} \\
&\rightarrow \{((), (a,)) : [J, A], A : A\} \\
&\rightarrow \{((), (a,)) : [A, A], A : A\} \\
&\rightarrow \{((), (a,)) : [a, A], A : A\} \\
&\rightarrow \{((), (a,)) : [a, a], A : A\} \\
&\rightarrow \{((), (a,)) : [a, a], L : A\} \\
&\rightarrow \{((), (a,)) : [a, a], [I] : A\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : A\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : D\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{K\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{A : A\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : A\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : L\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [I]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [J]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [J, A]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [A, A]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [a, A]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [a, T]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [a, (A, I)]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [a, (a, I)]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [a, (a, J)]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [a, (a, A)]\}\} \\
&\rightarrow \{((), (a,)) : [a, a], [] : \{a : [a, (a, a)]\}\}
\end{aligned}
$$

2. Give a parse tree for the rightmost derivation of the string

Answer:



3. Let us assume, that we use such a nested datastructure for calculations in the following way:

- α has a value of 1
- Lists (L) shall evaluate as the sum of their elements
- Tuples (T) shall evaluate as the negated sum of their elements
- Dictionaries (D) shall evaluate as the product of differences of key-value pairs (difference between a key and a value of a key-value-pair)
  (Examples: $\{\} = 0$, $X = \{[\alpha, \alpha, \alpha] : \alpha\} = 2$, $\{X : \{\}, \alpha : X\} = -2$

Write semantic actions to calculate the value of such a nested datastructure. You can associate a synthesized attribute `val` to each non-terminal symbol to store their value and you can read the values of the α's from α`.val`. The final value should be returned in the top-level `A.val`.

Answer:

A	$\rightarrow$	D	`{ A.val = D.val}`
A	$\rightarrow$	L	`{ A.val = L.val}`
A	$\rightarrow$	T	`{ A.val = T.val}`
A	$\rightarrow$	a	`{ A.val = 1}`
D	$\rightarrow$	$\{K\}$	`{ D.val = K.val}`
D	$\rightarrow$	$\{\}$	`{ D.val = 0}`
K	$\rightarrow$	$A_1 : A_2$	`{ K.val = A1.val - A2.val}`
K_1	$\rightarrow$	$K_2, A_1 : A_2$	`{K1.val = K2.val*(A1.val - A2.val)}`
T	$\rightarrow$	(A, I)	`{ T.val = -(A.val + I.val)}`
T	$\rightarrow$	$()$	`{ T.val = 0}`
L	$\rightarrow$	$[I]$	`{ L.val = I.val}`
I	$\rightarrow$	J	`{ I.val = J.val}`
I	$\rightarrow$	ϵ	`{ I.val = 0}`
J_1	$\rightarrow$	J_2, A	`{J1.val = J2.val + A.val}`
J	$\rightarrow$	A	`{ J.val = A.val}`

3 LL(1)

Consider the following grammar, which describes lists of words. Terminal symbols in this grammar are $\{word, and, ,\}$.

$$\begin{array}{rcl} S & \rightarrow & word \\ & | & word\ and\ word \\ & | & M,\ word\ and\ word \\ M & \rightarrow & M, word \\ & | & word \end{array}$$

Examples:

- `word`
- `word and word`
- `word, word, word and word`

Tasks:

1. Identify and explain all the reasons 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 *word*, it is not possible to choose between the 3 productions from S that start with *word* (M also starts with *word*) with just this information. However LL(1) languages allow for just one look-ahead symbol.
   - If the parser were to make use of the $M \rightarrow M, word$ production, for some look-ahead symbol, then in the new state it would still have to expand the new M with the same look-ahead, leading to an infinite loop.

2. Rewrite the grammar so that it is LL(1).

   Answer:

   - Eliminate immediate left recursion from the M productions:

$$\begin{array}{rcl} S & \rightarrow & word \\ & | & word\ and\ word \\ & | & M,\ word\ and\ word \\ M & \rightarrow & word\ M' \\ M' & \rightarrow & , word\ M' \\ & | & \epsilon \end{array}$$

- Inline singular M production rule to factorize S in one go:

$$\begin{array}{rcl} S & \rightarrow & word \\ & | & word\ and\ word \\ & | & word\ M',\ word\ and\ word \\ M' & \rightarrow & ,word\ M' \\ & | & \epsilon \end{array}$$

- Factorize S. Notice that "$,word$" can be produced from M', so remove it from the 3rd production of S':

$$\begin{array}{rcl} S & \rightarrow & word\ S' \\ S' & \rightarrow & \epsilon \\ & | & and\ word \\ & | & M'\ and\ word \\ M' & \rightarrow & ,word\ M' \\ & | & \epsilon \end{array}$$

- "*and word*" can be produced from S', through the 3rd rule if M' expands to ϵ, so remove the direct production and reorder:

$$\begin{array}{rcl} S & \rightarrow & word\ S' \\ S' & \rightarrow & M'\ and\ word \\ & | & \epsilon \\ M' & \rightarrow & ,word\ M' \\ & | & \epsilon \end{array}$$

3. Give the FIRST and FOLLOW sets for the non-terminals in the new grammar.

   Answer:

$$\begin{array}{ll|ll} First(S) & = \{word\} & Follow(S) & = \{\$\} \\ First(S') & = \{and, ',' , \epsilon\} & Follow(S') & = \{\$\} \\ First(M') & = \{',' , \epsilon\} & Follow(M') & = \{and\} \end{array}$$

4. To prove that your grammar is LL(1), construct an LL(1) parsing table for it.

	word	*and*	,	\$
S	$S \rightarrow word\ S'$			
S'		$S' \rightarrow M'\ and\ word$	$S' \rightarrow M'\ and\ word$	$S' \rightarrow \epsilon$
M'		$M' \rightarrow \epsilon$	$M' \rightarrow ,\ word\ M'$	

4 LR(1)

Again, consider the following grammar, where $\{a, b, c\}$ are terminal symbols:

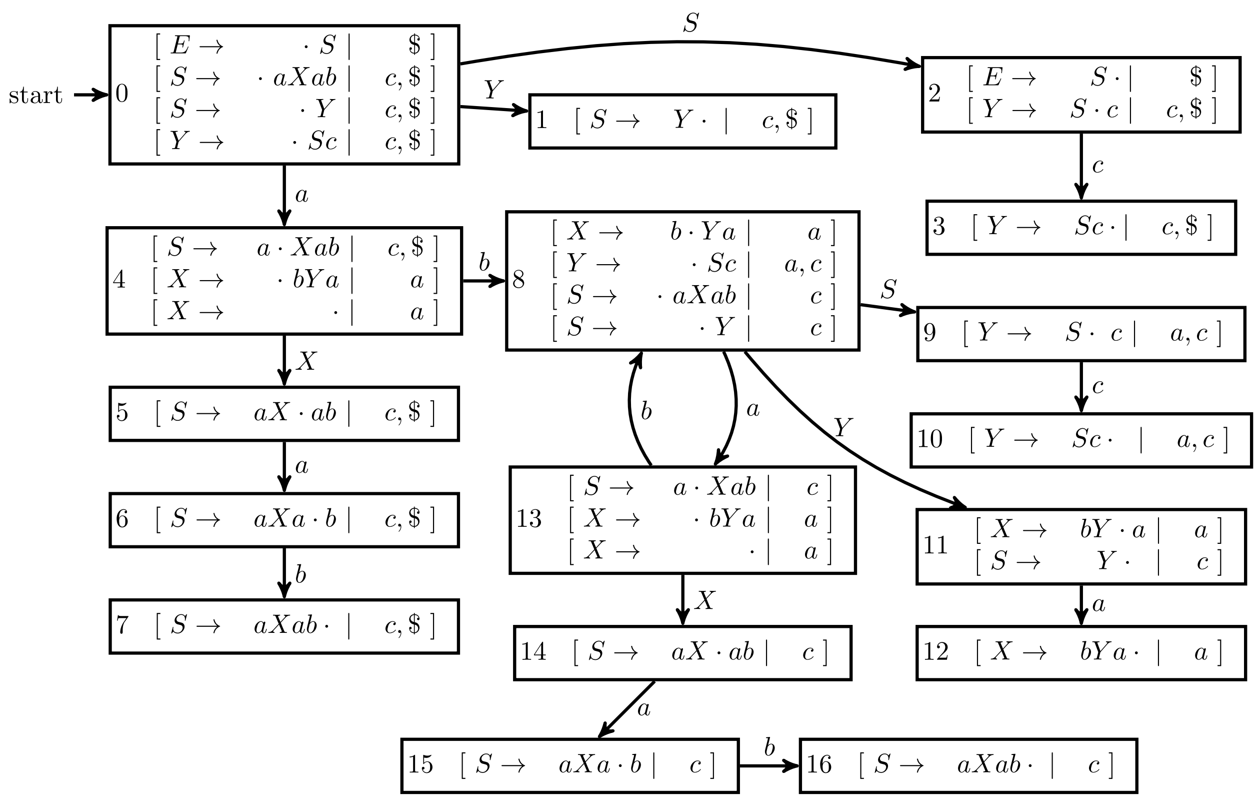
$$\begin{aligned} S &\rightarrow aXab && (1)\\ &\mid Y && (2)\\ X &\rightarrow bYa && (3)\\ &\mid \epsilon && (4)\\ Y &\rightarrow Sc && (5) \end{aligned}$$

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

   Answer:

   We use $\cdot$ to mark the position within each LR(1) item and $|$ as a separator between the core of the LR(1) items and the lookahead symbols.

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



2. 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	b	c	$	S	X	Y
0	s4				g2		g1
1			r2	r2			
2			s3	ACCEPT			
3			r5	r5			
4	r4	s8				g5	
5	s6						
6		s7					
7			r1	r1			
8	s13				g9		g11
9			s10				
10	r5		r5				
11	s12		r2				
12	r3						
13	r4					g14	
14	s15						
15		s16					
16			r1				

3. Show all steps required to parse the following string: *abaabccaab*

Answer:

Stack	Symbols	Input	Action
0		*abaabccaab*$	shift
0,4	*a*	*baabccaab*$	shift
0,4,8	*ab*	*aabccaab*$	shift
0,4,8,13	*aba*	*abccaab*$	reduce 4
0,4,8,13,14	*abaX*	*abccaab*$	shift
0,4,8,13,14,15	*abaXa*	*bccaab*$	shift
0,4,8,13,14,15,16	*abaXab*	*ccaab*$	reduce 1
0,4,8,9	*abS*	*ccaab*$	shift
0,4,8,9,10	*abSc*	*caab*$	reduce 5
0,4,8,11	*abY*	*caab*$	reduce 2
0,4,8,9	*abS*	*caab*$	shift
0,4,8,9,10	*abSc*	*aab*$	reduce 5
0,4,8,11	*abY*	*aab*$	shift
0,4,8,11,12	*abYa*	*ab*$	reduce 3
0,4,5	*aX*	*ab*$	shift
0,4,5,6	*aXa*	*b*$	shift
0,4,5,6,7	*aXab*	$	reduce 1
0,2	*S*	$	ACCEPT!