Individual Compiler Assignment - w2

Jonas Brunsgaard

December 9, 2012

1 Writing Context-Free Grammars

1.1 Define a CFG for $\{a^i b^j c^i | a, b, c \in \sum; i, j > 0\}$

Below I have defined the context free grammar.

$$
\begin{array}{lcl}
S & \rightarrow & aTc \\
T & \rightarrow & S \\
T & \rightarrow & R \\
R & \rightarrow & cR \\
R & \rightarrow & c
\end{array}
$$

1.2 Show the left-most derivation of the word 'aabbcc'

Below you find the leftmost derivation of the string

$$
\begin{array}{l}
\quad S \\
\Rightarrow a\underline{T}c \\
\Rightarrow aa\underline{T}cc \\
\Rightarrow aa\underline{R}cc \\
\Rightarrow aab\underline{R}cc \\
\Rightarrow aabbcc
\end{array}
$$

2 LL(1)-Parser Construction

2.1 Eliminating left-recursion and left-factorise the grammar

We eliminate left recursion

$$
\begin{array}{lcl}
S & \rightarrow & \text{id} \\
S & \rightarrow & \text{id}[E] \\
E & \rightarrow & SE^* \\
E^* & \rightarrow & \varepsilon \\
E^* & \rightarrow & ,E
\end{array}
$$

and follow up with left-factorisation

$$
\begin{array}{lcl}
S & \rightarrow & \mathrm{id}S^* \\
S^* & \rightarrow & \varepsilon \\
S^* & \rightarrow & [E] \\
E & \rightarrow & SE^* \\
E^* & \rightarrow & \varepsilon \\
E^* & \rightarrow & ,E
\end{array}
$$

2.2 Calculate First sets

First we find if terminals and nonterminals are *nullable*.

Right-hand side	Init	First Iter	Sec Iter
id	*false*	*false*	*false*
ε	*false*	*true*	*true*
$[E]$	*false*	*false*	*false*
SE^*	*false*	*false*	*false*
$,E$	*false*	*false*	*false*
Nonterminal			
S	*false*	*false*	*false*
S^*	*false*	*true*	*true*
E	*false*	*false*	*false*
E^*	*false*	*true*	*true*

Then we calculate $FIRST$:

Right-hand side	Init	First Iter	Sec Iter
id	$\emptyset$	$\{\mathrm{id}\}$	$\{\mathrm{id}\}$
ε	$\emptyset$	$\emptyset$	$\emptyset$
$[E]$	$\emptyset$	$\{[\}$	$\{[\}$
SE^*	$\emptyset$	$\{\mathrm{id}\}$	$\{\mathrm{id}\}$
$,E$	$\emptyset$	$\{,\}$	$\{,\}$
Nonterminal			
S	$\emptyset$	$\{\mathrm{id}\}$	$\{\mathrm{id}\}$
S^*	$\emptyset$	$\{[\}$	$\{[\}$
E	$\emptyset$	$\{\mathrm{id}\}$	$\{\mathrm{id}\}$
E^*	$\emptyset$	$\{,\}$	$\{,\}$

2.3 Calculate Follow sets for all nonterminals

By following the procedure on page 59 in the book, we find the following table.

Production	Constraints
$S' \to S\$$	$\{\$\} \subseteq FOLLOW(S)$
$S \to \mathrm{id}S^*$	$FOLLOW(S) \subseteq FOLLOW(S^*)$
$S^* \to \varepsilon$	
$S^* \to [E]$	$\{]\} \subseteq FOLLOW(E)$
$E \to SE^*$	$\{,\} \subseteq FOLLOW(S)$
	$FOLLOW(E) \subseteq FOLLOW(S)$
	$FOLLOW(E) \subseteq FOLLOW(E^*)$
$E^* \to \varepsilon$	
$E^* \to ,E$	$FOLLOW(E^*) \subseteq FOLLOW(E)$

In the table above I have used the $FIRST$-sets we calcualted earlier, and thus they are shown as explicit terminals.

We first use the constraints $\{\$\} \subseteq FOLLOW(S)$ and constraints of the form $FIRST(\dots) \subseteq FOLLOW(\dots)$ to get the initial sets. Note, that in the following i will use the word "komma" to refer to the terminal ",". The reason be that the symbol "," are used as seperator, when describing a set.

$$\begin{aligned}
FOLLOW(S) &\supseteq \{\text{komma}, \$\} \\
FOLLOW(S^*) &\supseteq \{\emptyset\} \\
FOLLOW(E) &\supseteq \{]\} \\
FOLLOW(E^*) &\supseteq \{\emptyset\}
\end{aligned}$$

and then use the constrains on the form $FOLLOW(\dots) \subseteq FOLLOW(\dots)$. After first iteration:

$$\begin{aligned}
FOLLOW(S) &\supseteq \{\text{komma}, \$]\} \\
FOLLOW(S^*) &\supseteq \{\text{komma}, \$\} \\
FOLLOW(E) &\supseteq \{]\} \\
FOLLOW(E^*) &\supseteq \{]\}
\end{aligned}$$

second iteration:

$$\begin{aligned}
FOLLOW(S) &\supseteq \{\text{komma}, \$]\} \\
FOLLOW(S^*) &\supseteq \{\text{komma}, \$]\} \\
FOLLOW(E) &\supseteq \{]\} \\
FOLLOW(E^*) &\supseteq \{]\}
\end{aligned}$$

and the third iteration, nothing has changed, so the final result is

$$\begin{aligned}
FOLLOW(S) &= \{\text{komma}, \$]\} \\
FOLLOW(S^*) &= \{\text{komma}, \$]\} \\
FOLLOW(E) &= \{]\} \\
FOLLOW(E^*) &= \{]\}
\end{aligned}$$

2.4 Look-aheads sets and pseudo code

From the lecture slides the look ahead set is defined as

$$la(X \to \alpha) = \begin{cases} FIRST(\alpha) \cup FOLLOW(X) & \text{, if } NULLABLE(\alpha) \\ FIRST(\alpha) & \text{, otherwise} \end{cases}$$

Below the lookahead sets for our productions are shown.

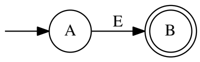
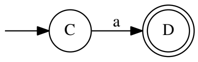
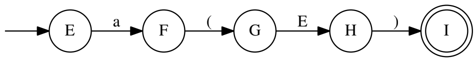
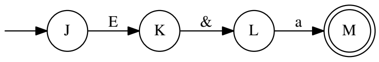
$$\begin{array}{lcl} LA(S' \to S\$) & = & \{\text{id}\} \\ LA(S \to \text{id}S^*) & = & \{\text{id}\} \\ LA(S^* \to \varepsilon) & = & \{\emptyset\} \\ LA(S^* \to [E]) & = & \{[\} \\ LA(E \to SE^*) & = & \{\text{id}\} \\ LA(E^* \to \varepsilon) & = & \{\emptyset\} \\ LA(E^* \to \text{komma}E) & = & \{\text{komma}\} \end{array}$$

We now build our pseudo code for the parser:

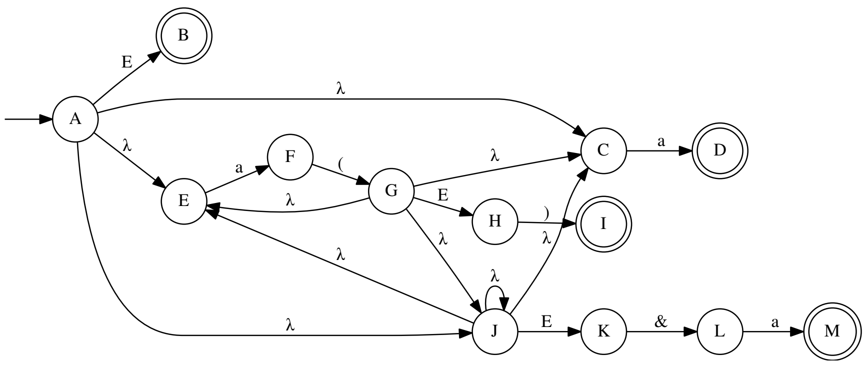
```
function parseS' () =
    if next = 'id'
    then parseT(); match('\$')
    else reportError()

function parseS () =
    if next = 'id'
    then match('id'); parseS*()
    else reportError()

function parseS* () =
    if next = '\$'
    then (* do nothing *)
    else if next = '['
         then match('['); parseE(); match(']')
         else reportError()

function parseE () =
    if next = 'id'
    then parseS(); parseE*()
    else reportError()

function parseE* () =
    if next = '\$'
    then (* do nothing *)
    else if next = ','
         then match(','); parseE();
         else reportError()
```

3 SLR Parser Construction

First we produce NFA's for the right hand side of every production.

Production	NFA
$S \rightarrow E$	
$E \rightarrow$ a	
$E \rightarrow$ a(E)	
$E \rightarrow E$&a	

If an NFA state s has an outgoing transition on nonterminal N we add a lambda transition from s to the starting states of the NFAs for the rhs of the productions for N.



Then we convert the combined NFA to a DFA.

$$
\begin{array}{lcl}
\hat{\lambda}(A) & = & \{A, C, E, J\} = s_0 \\
move(s_0, a) & = & \hat{\lambda}(\{D, F\}) = \{D, F\} = s_1 \\
move(s_0, E) & = & \hat{\lambda}(\{B, K\}) = \{B, K\} = s_2 \\
move(s_1, () & = & \hat{\lambda}(\{G\}) = \{G, C, E, J\} = s_3 \\
move(s_2, \&) & = & \hat{\lambda}(\{L\}) = \{L\} = s_4 \\
move(s_3, a) & = & \hat{\lambda}(\{D, F\}) = \{D, F\} = s_1 \\
move(s_3, E) & = & \hat{\lambda}(\{K, H\}) = \{K, H\} = s_5 \\
move(s_4, a) & = & \hat{\lambda}(\{M\}) = \{M\} = s_6 \\
move(s_5,)) & = & \hat{\lambda}(\{I\}) = \{I\} = s_7 \\
move(s_5, \&) & = & \hat{\lambda}(\{L\}) = \{L\} = s_4
\end{array}
$$

Now we are able to build the initial cross index table

DFA state	NFA states	a	(	)	&	E
0	A,C,E,J	s1				g2
1	D,F		s3			
2	K,B				s4	
3	G,C,E,J	s1				g5
4	L	s6				
5	H,K			s7	s4	
6	M					
7	I					

The follow set for the production E and S, has been computed

$$\begin{aligned} FOLLOW(S) &= \{\$\} \\ FOLLOW(E) &= \{\$,), \&\} \end{aligned}$$

and used to determine wheather we shift or reduce. Below the finale table can be seen.

DFA state	a	(	)	&	$	E
0	s1					g2
1		s3	r1	r1	r1	
2				s4	a	
3	s1					g5
4	s6					
5			s7	s4		
6			r3	r3	r3	
7			r2	r2	r2	

3.1 Parsing the input "a(a&a)"

Input	stack	action
a(a&a)	0	s1
(a&a)	01	s3
a&a)	0131	s1
&a)	0135	r1($E \rightarrow a$); g5
&a)	01354	s4
)	013546	r3($E \rightarrow E$&a); g5
)	0135	s7
$	10357	r2($E \rightarrow$ a(E)); g2
$	02	a

3.2 Build a parser in mosmlyac

After compiling the .grm file, it is clear from the output file, that mosmlyac are using SLR parsing.

DFA state	a	(	)	&	'\001'	EOF	*%end*	*%entry*	*S*	*E*
0					s1			g2		
1	s3								g4	g5
2							a			
3		s6	r2	r2		r2				
4	r5	r5	r5	r5	r5	r5	r5			
5				s7		s8				
6	s3									g9
7	s10									
8	r1	r1	r1	r1	r1	r1	r1			
9			s11	s7						
10	r4	r4	r4	r4	r4	r4	r4			
11	r3	r3	r3	r3	r3	r3	r3			

We can see that the two tables are not that different and i try to parse the string '\001a(a&a)EOF'

Input	stack	action
'\001'a(a&a)	0	s1
a(a&a)	0 1	s3
(a&a)	0 1 3	s6
a&a)	0 1 3 6	s3
&a)	0 1 3 6 3	r2; g9
&a)	0 1 3 6 9	s7
a)	0 1 3 6 9 7	s10
)	0 1 3 6 9 7 10	r4; g9
)	0 1 3 6 9	s11
EOF	0 1 3 6 9 11	r3; g5
EOF	0 1 5	s8
$	0 1 5 8	r1; g4
$	0 1 4	r5; g2
$	0 2	1

The parser generator introduces some new terminals and some nonterminale e.g. %entry and %end, but overall the handmade SLR table and the parsergenerated SLR table are not that different.