Compiler Construction
WA04: Bottom-Up Parsing

Terence Henriod

October 2, 2015

Abstract

This assignment asks you to prepare written answers to questions on bottom-up parsing. The answers are longer but more mechanical than on previous assignments. You may discuss this assignment with other students and work the problems together. However, your writeup should be your own individual work. Remember written assignments are to be turned in in class on the date due.

One of the least loved and least understood aspects of the C programming language is its type declarators. We can improve on the latter problem by considering the following simplified declarator grammar.

$$
\begin{array}{lll}
S & \to & TP \\
T & \to & \texttt{int} \mid \texttt{char} \\
P & \to & *P \mid D \\
D & \to & \texttt{id} \mid D() \\
\end{array}
$$

In this grammar, S is the start symbol, T is a C type, P is a pointer declarator, and D is an ordinary declarator. The terminals are the punctuation marks, `int`, and `char`.

1. Give the LR(0) item sets, and the DFA of the LR(0) item sets for this grammar. Please show all of your work.

   Answer:

0:	$(start)$
	$Z \to .S$
	$S \to .TP$
	$T \to .\texttt{int}$
	$T \to .\texttt{char}$

1:	$0 \curvearrowright S$	
	$Z \to S.$	⊛

2:	$0 \curvearrowright T$
	$S \to T.P$
	$P \to . * P$
	$P \to .D$
	$D \to .\texttt{id}$
	$D \to .D()$

3:	$0 \curvearrowright \texttt{int}$	
	$T \to \texttt{int}.$	⊛

4:	$0 \curvearrowright \texttt{char}$	
	$T \to \texttt{char}.$	⊛

5:	$2 \curvearrowright P$	
	$S \to TP.$	⊛

6:	$2 \curvearrowright *$
	$P \to *.P$
	$P \to . * P$
	$P \to .D$
	$D \to .\texttt{id}$
	$D \to .D()$

7:	$2 \curvearrowright D$	
	$P \to D.$	⊛
	$D \to D.()$	

8: $2 \curvearrowright$ `id`
$D \rightarrow$ `id`. $\circledast$

9: $6 \curvearrowright P$
$P \rightarrow *P.$ $\circledast$

: $6 \curvearrowright *$
$=$ *State* 6

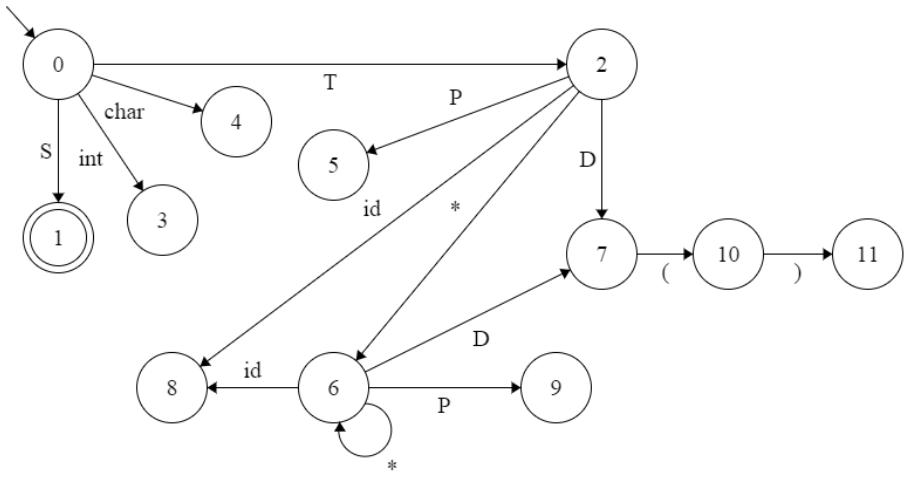
: $6 \curvearrowright D$
$=$ *State* 7

: $6 \curvearrowright$ `id`
$=$ *State* 8

10: $7 \curvearrowright ($
$D \rightarrow D(.)$

11: $10 \curvearrowright ($
$D \rightarrow D().$ $\circledast$

DFA:



2. Is this grammar SLR(1)? Why or why not?

Answer:

In order to find out if the grammar is SLR(1), we should find out if the grammar is ambiguous. We can do so by constructing the parse table. In order to construct the parse table, we will need to construct the *Follow* sets (which also requires constructing the *First* sets), and we will need to assign a numbering to the productions of the grammar.

First Sets:

$First(S : S \to TP)$	$= First(TP)$ $= First(T) - \{\epsilon\}$ $= \{\texttt{int}, \texttt{char}\}$
$First(S)$	$= \{\texttt{int}, \texttt{char}\}$

$First(T : T \to \texttt{int})$	$= First(\texttt{int})$ $= \{\texttt{int}\}$
$First(T : T \to \texttt{char})$	$= First(\texttt{char})$ $= \{\texttt{char}\}$
$First(T)$	$= \{\texttt{int}\} \cup \{\texttt{char}\}$ $= \{\texttt{int}, \texttt{char}\}$

$First(P : P \to *P)$	$= First(*P)$ $= First(*)$ $= \{*\}$
$First(P : P \to D$	$= First(D)$ $= \{\texttt{id}\}$
$First(P)$	$= \{*\} \cup \{\texttt{id}\}$ $= \{*, \texttt{id}\}$

$First(D : D \to \texttt{id})$	$= First(\texttt{id})$ $= \{\texttt{id}\}$
$First(D : D \to D())$	$= First(D())$ $= First(D)$ $= \texttt{TBD}$
$First(D)$	$= \{\texttt{id}\}$

Follow Sets:

$Follow(S : \texttt{Start Symbol})$	$= \{\$\}$
$Follow(S)$	$= \{\$\}$

$Follow(T : S \to TP)$	$= First(P) - \{\epsilon\}$ $= \{*, \texttt{id}\}$
$Follow(T)$	$= \{*, \texttt{id}\}$

$Follow(P : S \to TP)$	$= Follow(S)$
	$= \{\$\}$
$Follow(P : P \to *P)$	$= Follow(P)$
	$=$ `TBD`
$Follow(P)$	$= \{\$\}$

$Follow(D : P \to D)$	$= Follow(P)$
	$= \{\$\}$
$Follow(D : D \to D())$	$= First(())$
	$= First(($
	$= \{($
$Follow(D)$	$= \{\$\} \cup \{($
	$= \{(, \$\}$

Numbered Productions:

0	Z	$\to$	S
1	S	$\to$	TP
2	T	$\to$	`int`
3	T	$\to$	`char`
4	P	$\to$	$*P$
5	P	$\to$	D
6	D	$\to$	`id`
7	D	$\to$	$D()$

Action Table:

	Terminals							Non-Terminals			
State	`int`	`char`	`*`	`id`	`(`	`)`	$	S	T	P	D
0	s3	s4						1	2		
1							acc				
2			s6	s8						5	7
3			r2	r2							
4			r3	r3							
5							r1				
6			s6	s8						9	7
7					s10		r5				
8					r6		r6				
9							r4				
10						s11					
11					r7		r7				

There were no conflicts in creating the parser table, the grammar is not ambiguous, the grammar is an SLR(1) grammar.

3. Show the sequence of moves of an SLR(1) parser on the following input. Resolve any shift-reduce conflicts in favor of shifting.
`char * id()`

Answer:

Note: There are no conflicts to resolve.

Stack	Input	Entry	Action/Production
[(start), 0]	`char * id()`$	s4	Shift; Enter State 4
[(start), 0] [`char`, 4]	`* id()`$	r3	(3) $T \rightarrow$ `char`
[(start), 0] [T, 2]	`* id()`$	s6	Shift; Enter State 6
[(start), 0] [T, 2] [$*$, 6]	`id()`$	s8	Shift; Enter State 8
[(start), 0] [T, 2] [$*$, 6] [`id`, 8]	`()`$	r6	(6) $D \rightarrow$ `id`
[(start), 0] [T, 2] [$*$, 6] [D, 7]	`()`$	s10	Shift; Enter State 10
[(start), 0] [T, 2] [$*$, 6] [D, 7] [`(`, 10]	`)`$	s11	Shift; Enter State 11
[(start), 0] [T, 2] [$*$, 6] [D, 7] [`(`, 10] [`)`, 11]	$	r7	(7) $D \rightarrow D$`()`
[(start), 0] [T, 2] [$*$, 6] [D, 7]	$	r5	(5) $P \rightarrow D$
[(start), 0] [T, 2] [$*$, 6] [P, 9]	$	r4	(4) $P \rightarrow *P$
[(start), 0] [T, 2] [P, 5]	$	r1	(1) $S \rightarrow TP$
[(start), 0] [S, 1]	$	acc	**Successful Parse**