Computer Science 414

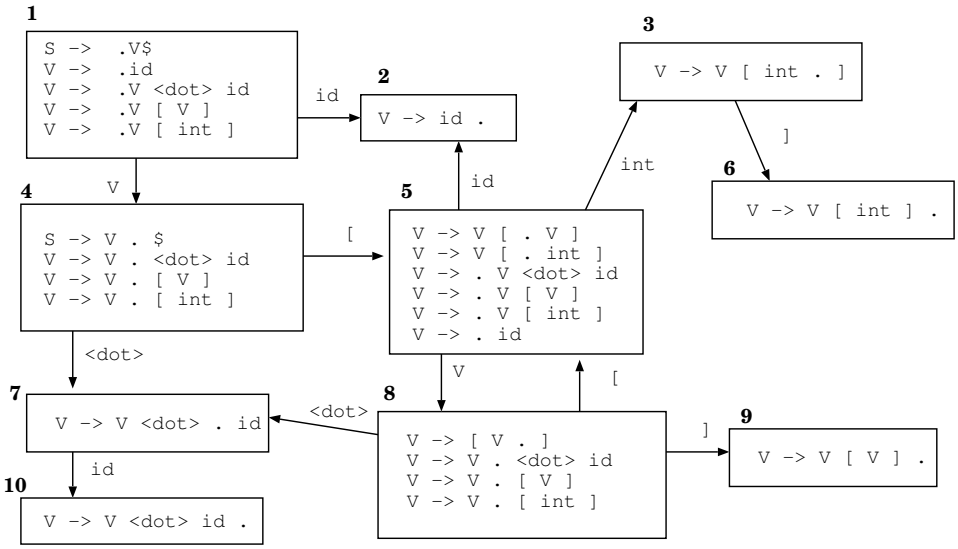
Midterm 2 Practice Problems

Solutions

1. Give the LR(0) states and transitions, as well as the LR(0) parse table, for the following grammars

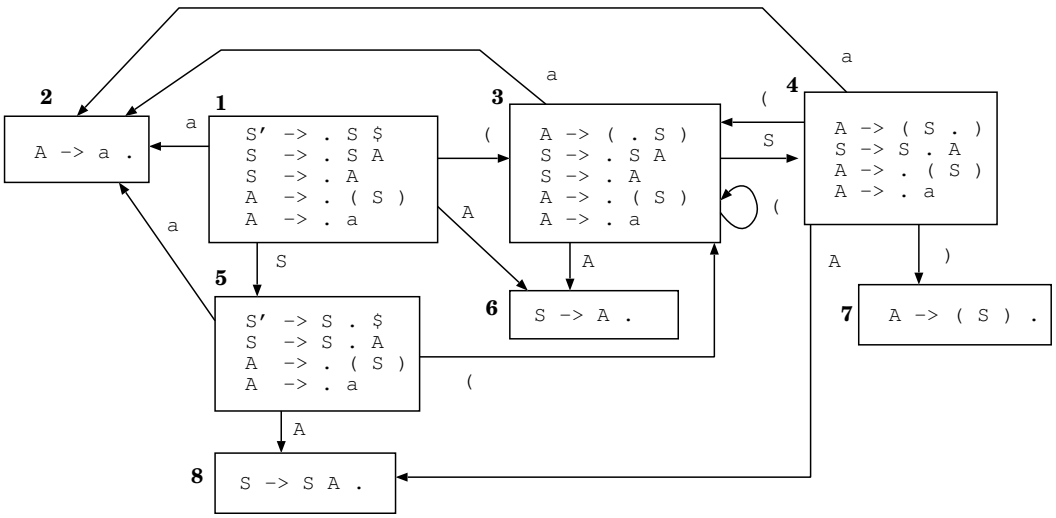
(a)

Terminals = {id, ., [,], int }
Non-Terminals = $\{S, V\}$
Rules = (0) $S \rightarrow V\$$
= (1) $V \rightarrow$ id
(2) $V \rightarrow V$. id
(3) $V \rightarrow V$ [V]
(4) $V \rightarrow V$ [int]
Start Symbol = S



	int	id	[	]	¡dot¿	$	S	V
1	s4	s6					g4	
2	r(1)	r(1)	r(1)	r(1)	r(1)	r(1)		
3				s6				
4			s5		s7	accept		
5	s3	s2					g4	
6	r(4)	r(4)	r(4)	r(4)	r(4)	r(4)		
7		s10						
8			s5	s9	s10			
9	r(3)	r(3)	r(3)	r(3)	r(3)	r(3)		
10	r(2)	r(2)	r(2)	r(2)	r(2)	r(2)		

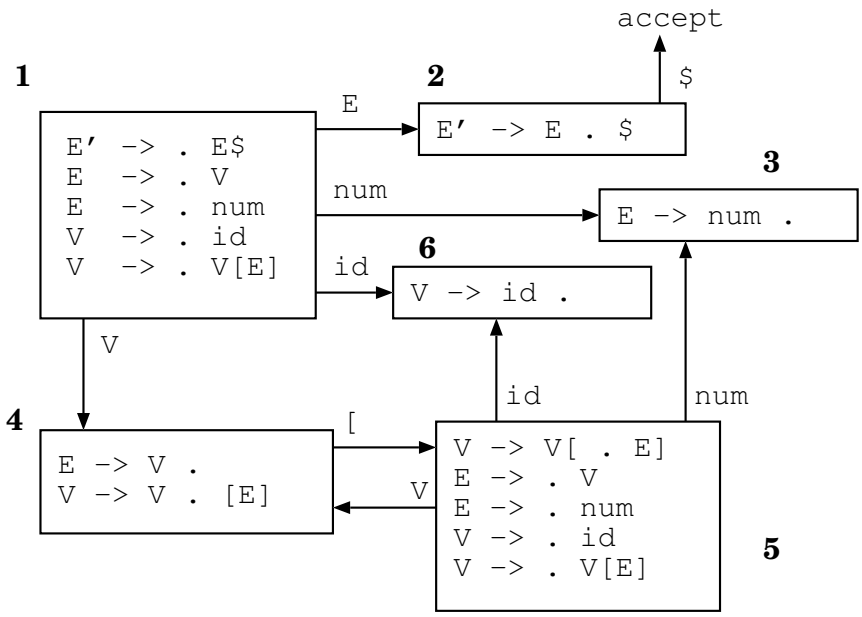
Terminals = {a, (,), \$}
Non-Terminals = $\{S', S, A\}$
Rules = (0) $S' \rightarrow S\$$
(1) $S \rightarrow SA$
(2) $S \rightarrow A$
(3) $A \rightarrow (S)$
(4) $A \rightarrow$a
Start Symbol = S'



	a	(	)	$	S	A
1	s2	s3			g5	g6
2	r(4)	r(4)	r(4)	r(4)		
3	s2	s3			g4	g6
4	s2	s3	s7			g8
5	s2	s3		accept		g8
6	r(2)	r(2)	r(2)	r(2)		
7	r(3)	r(3)	r(3)	r(3)		
7	r(1)	r(1)	r(1)	r(1)		

2. Create a set of LR(0) items and transitions for the following grammar. Show that the grammar is not LR(0). Give the SLR(1) parse table for the grammar.

Terminals = {num, id, [,], $}
Non-Terminals = $\{E', E, V\}$
Rules = (0) $E' \rightarrow E\$$
(1) $E \rightarrow V$
(2) $E \rightarrow$ num
(3) $V \rightarrow$ id
(4) $V \rightarrow V[E]$
Start Symbol = E'



The LR(0) parse table has duplicate entries:

	num	id	[	]	$	E	V
1	s3	s6				g2	g4
2					accept		
3	r(2)	r(2)	r(2)	r(2)	r(2)		
4	r(1)	r(1)	r(1),s5	r(1)	r(1)		
5	s3	s6					g4
6	r(3)	r(3)	r(3)	r(3)	r(3)		

First and follow sets:

Non-Terminal	First	Follow
E'	num, id	
E	num, id	$,]
V	id	$,], [

The SLR(1) parse table:

	num	id	[	]	$	E	V
1	s3	s6				g2	g4
2					accept		
3				r(2)	r(2)		
4			s5	r(1)	r(1)		
5	s3	s6					g4
6			r(3)	r(3)	r(3)		

3. Standard Java allows loop exits using the "break" and "continue" statements. A "break" statement leaves the current loop entirely, and a "continue" statement jumps to the end of the current loop. A break statement can only appear inside a loop or a switch statement, and a continue can only appear inside a loop. If we add "break" and "continue" to simpleJava, what changes must be made to the semantic analyzer to ensure that these statements only appear within loops?

   For the semantic analyzer, we only need to know if we are currently in a loop or not. Now, we could just try to use a global (to the iterator) boolean to denote if we were in the loop or not, but that would be difficult because of nesting loops. Instead, we couuld keep a count of how deeply nested we were in a loop. Every time a loop starts, increment the depth, and every time it ends, decrement the loop. A

break is legal if the nesting level is greater than zero. What about for creating the abstract assembly tree? We need to know what label to jump to on the break. We could keep a stack of "end loop labels" – every time we start a loop, push the end of the loop label onto the stack, and every time we end a loop, pop off the top. Any time we need to generate code for a break or continue, use the label at the top of the stack.

4. When creating the Abstract Assembly for a function definition, Do we *need* to store the Stack Pointer on the stack? What about the Frame Pointer and Return Register? Explain.

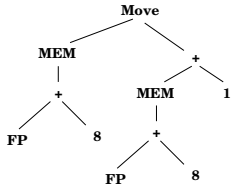
   We do not need to store the old Stack Pointer on the stack, since we know the size of our stack frame. Instead, we could just subtract (or add a negative) appropriate offest to the stack pointer at the beginning of the function call, and then add (or subtract, if negative) the same offset at the end of the function call. We do, however, need to store the old Frame Pointer and Return register, since we do not know who called us at compile time.

5. Given the following class definitions, and the local variable declarations in the function foo:

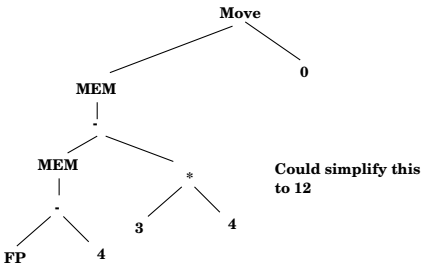
```
class c1 {
   int w;
   boolean z;
}

class c2 {
   c1 y[];
   int z;
}

int foo(int x,y) {
   boolean a;
   boolean b[];
   c1 C;
   c2 D[];

  /*  Body of foo  */

}
```

   Give the abstract assembly tree for each of the following statements, if they appeared in the body of foo:

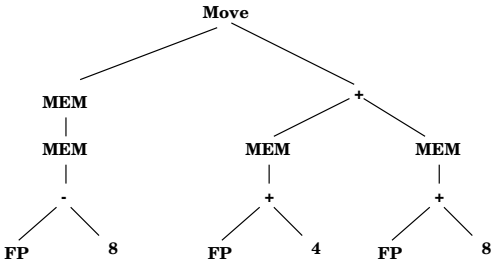
   (a) `y++;`



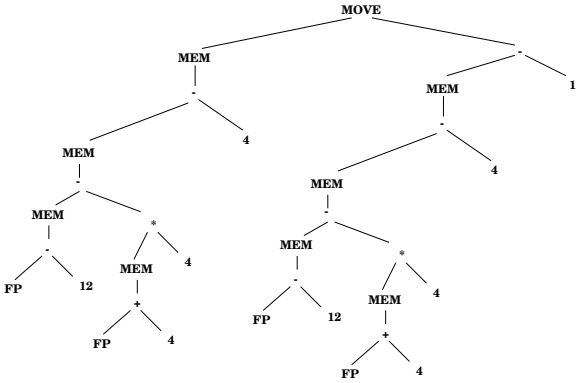
   (b) `b[3] = false;`



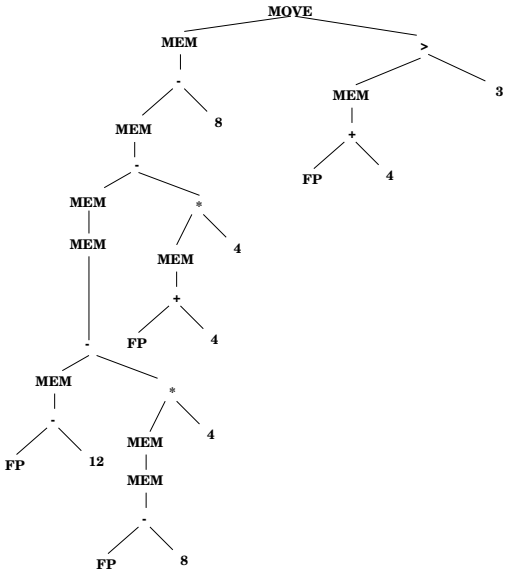
(c) `C.w = x + y;`



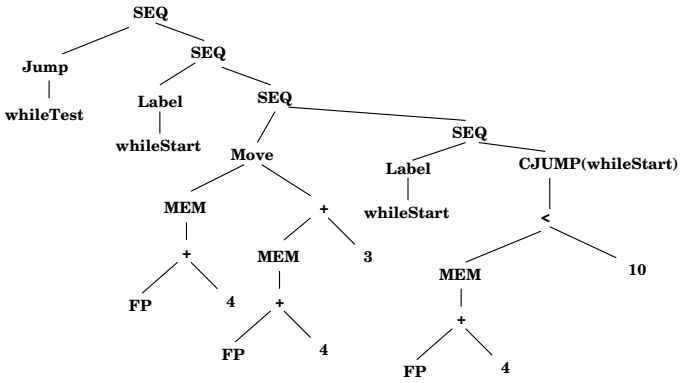
(d) `D[x].z--;`



(e) `D[C.w].y[x].z = x > 3;`



(f) `while (x < 10) x = x + 3;`


SEQ
Jump
SEQ
whileTest
Label
SEQ
whileStart
Move
SEQ
Label
CJUMP(whileStart)
MEM
+
whileStart
<
+
MEM
3
MEM
10
FP
4
+
+
FP
4
FP
4