Compiler Construction

Lecture 7 - LR(1), LALR and SLR(1) Parsing

Outline

I. SLR(1) State Machine
II. LR(1) State Machine
III. LALR State Machine

Shift-Reduce Conflicts

Let’s consider our original expression grammar:

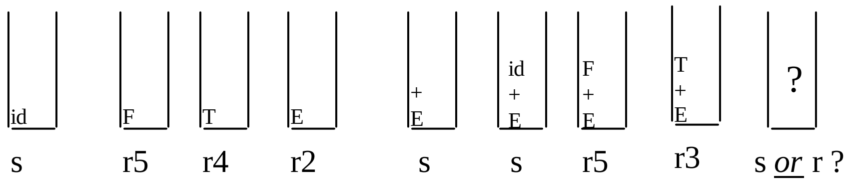
E ::= E **+** *T* | *T*

T ::= *T* ***** *F* | *F*

F ::= **id** | **(** *E* **)**

If we try to build an LR(0) parser, we will have a problem

Example: x + y * z



Shift-Reduce Conflicts (continued)

We have a conflict - do we shift or reduce? Looking at the state machine by itself does not tell us.

Answer: We must use a lookahead.

The simplest version of an LR(1) parser is called SLR(1) (The “***S***”is for “***S***implified”)

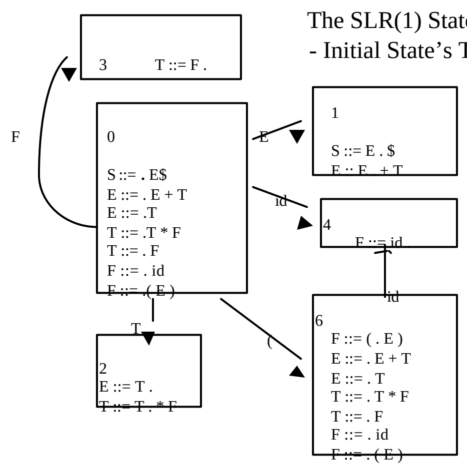
We will use the follows as a means of resolving the shift-reduce conflict.

The Initial State of the SLR(1) State Machine

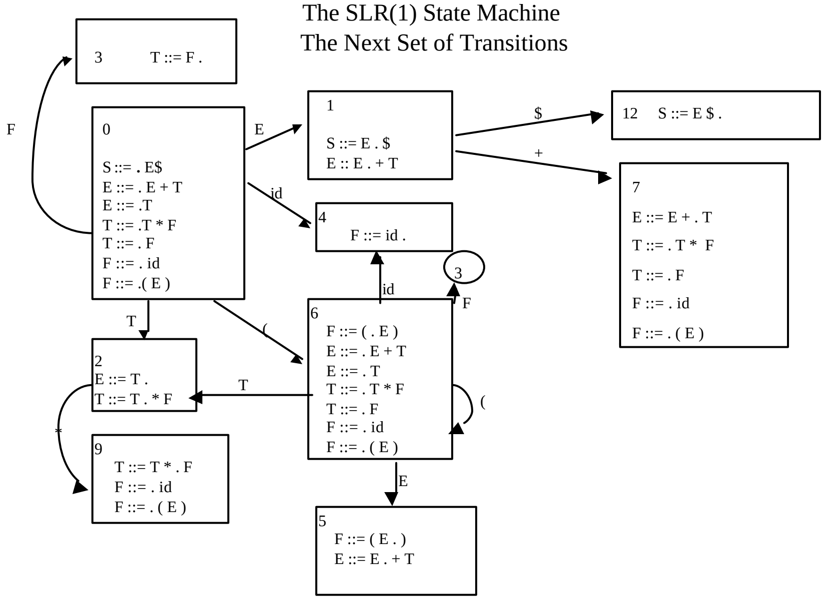
0

S ::= . E$
E ::= . E + T
E ::= .T
T ::= .T * F
T ::= . F
F ::= . id
F ::= .(E)

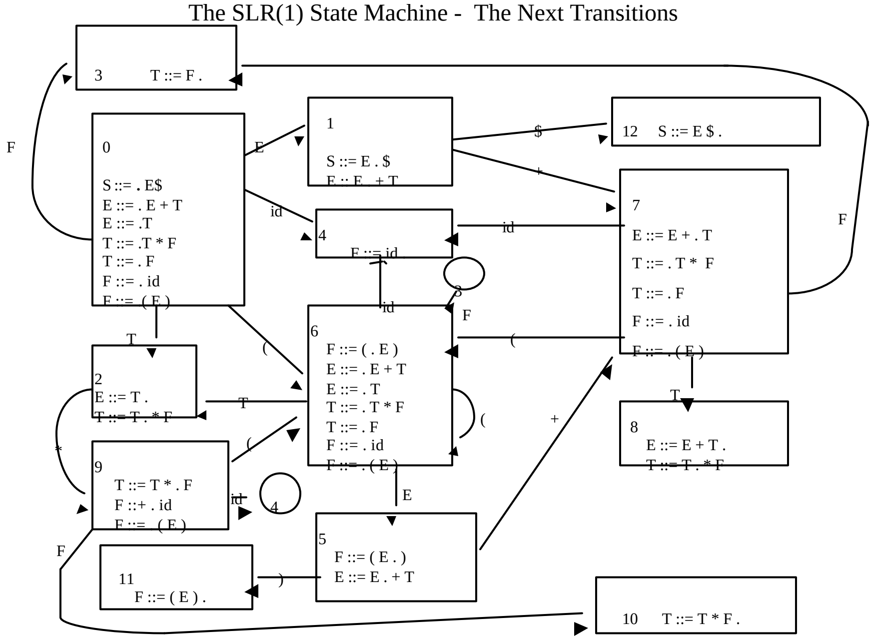
The SLR(1) State Machine
- Initial State's Transitions

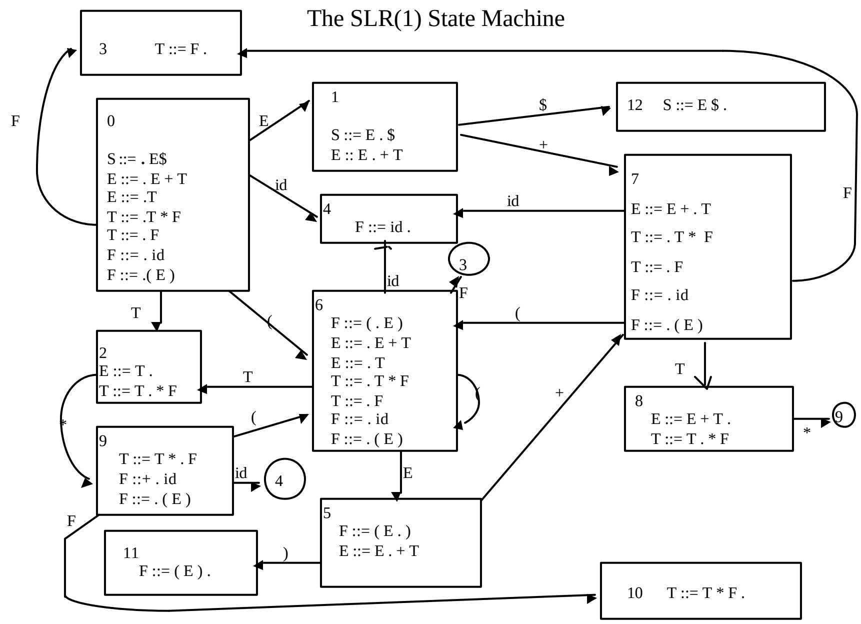


The SLR(1) State Machine
The Next Set of Transitions



The SLR(1) State Machine - The Next Transitions





The SLR(1) State Machine (continued)

To make the ACTION/GOTO table, we need the FOLLOW sets, finding the FIRST sets where necessary.

FIRST(*E*) = FIRST(*T*) = FIRST(*F*) = { **id**, **(** }

FIRST(*T*) = FIRST(*F*) = {**id**, **(** }

FIRST(*F*) = {**id**, **(** }

FOLLOW(*E*) = { **$**, **+**, **)** }

FOLLOW(*T*) = { **$**, **+**, *****, **)**}

FOLLOW(*F*) = { **$**, **+**, *****, **)**}

The SLR(1) Parse Table

- We will use the FOLLOW sets to determine which action we will take depending on which lookahead token.
- We will include the GOTOs for all terminals wherever the action is ***Shift***.
- Nonterminals will have only GOTOs.

The SLR(1) Parse Table

	ACTION						GOTO		
state	+	*	id	(	)	$	E	T	F
0			s4	s6			1	2	3
1	s7					acc			
2	r3	s9			r3	r3			
3	r5	r5			r5	r5			
4	r6	r6			r6	r6			
5	s7				s11				
6			s4	s6			5	2	3
7			s4	s6				8	3
8	r2	s9			r2	r2			
9			s4	s6					10
10	r4	r4			r4	r4			
11	r7	r7			r7	r7			

LR(1) Parsing

- SLR(1) grammars are somewhat limited in terms of the languages that they can process.
- LR(1) use a lookahead and place the lookahead within the term:

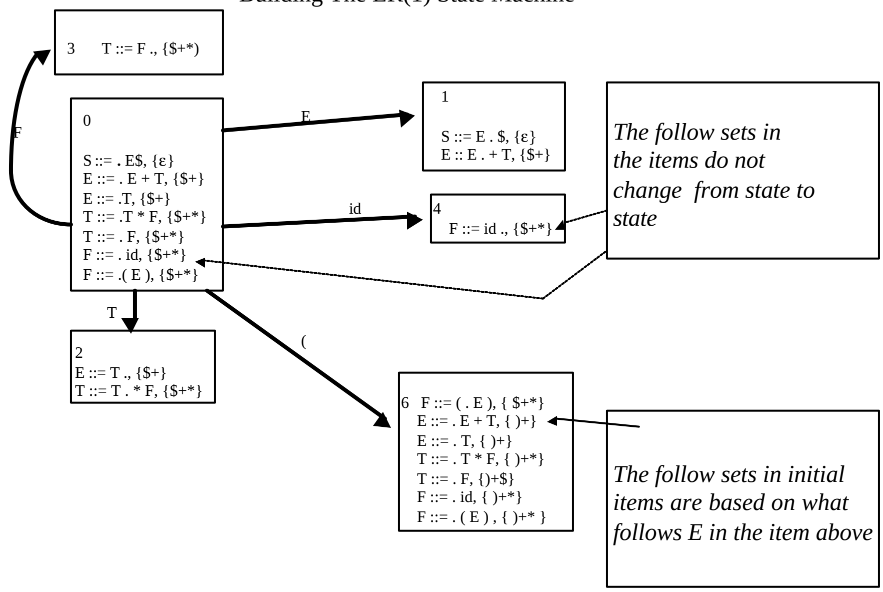
1. S ::= . E $, { ε } *Nothing can follow S*
2. E ::= . E + T, { $ + } *$ comes from prod. 1*
3. E ::= T, { $ +} *+ comes from prod. 2*

Building The LR(1) State Machine

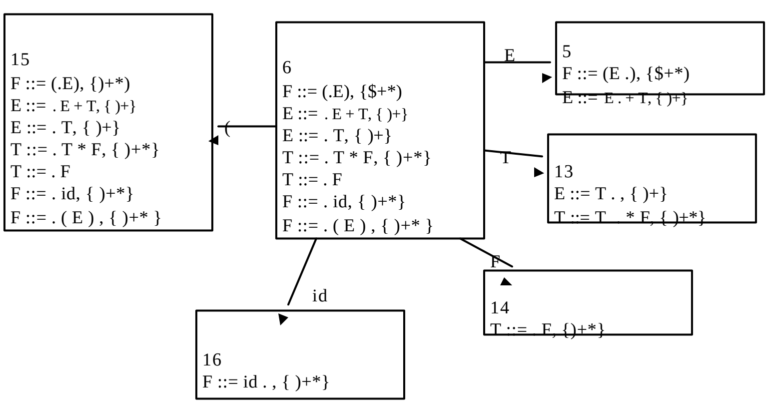
0

S ::= . E$, {ε}
E ::= . E + T, {$+}
E ::= .T, {$+}
T ::= .T * F, {$+*} *comes from production 4*
T ::= . F, {$+*}
F ::= . id, {$+*} *comes from production 4*
F ::= .(E), {$+*}

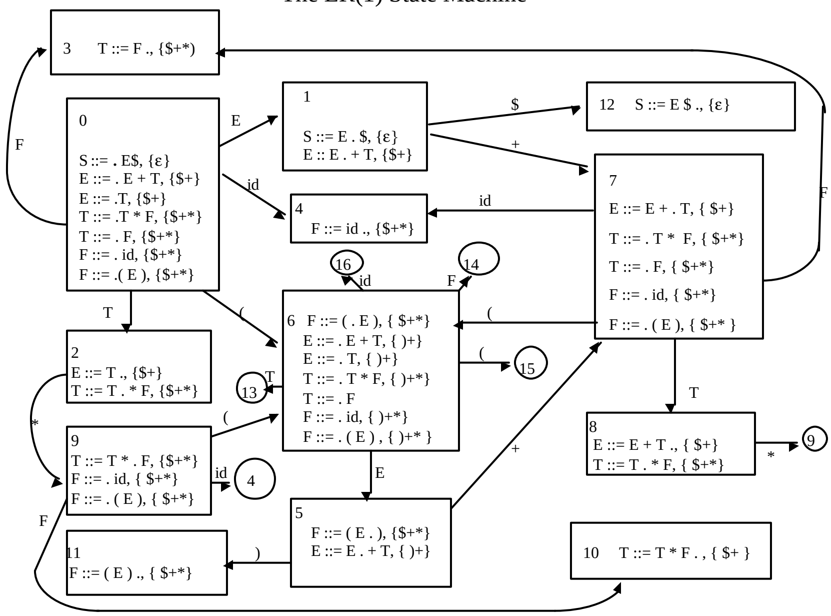
Building The LR(1) State Machine



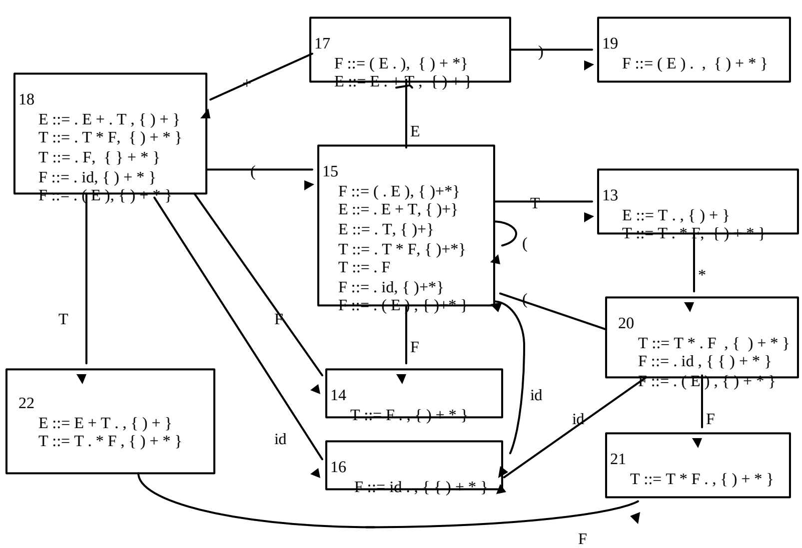
Building The LR(1) State Machine



The LR(1) State Machine



The LR(1) State Machine (continued)



The LR(1) Parse Table

	ACTION						GOTO		
state	+	*	id	(	)	$	E	T	F
0			s4	s6			1	2	3
1	s7					acc			
2	r3	s9				r3			
3	r5	r5				r5			
4	r6	r6				r6			
5	s7				s11				
6			s16	s15			5	13	14
7			s4	s6				8	3
8	r2	s9				r2			
9			s4	s6					10
10	r4	r4				r4			
11	r7	r7				r7			

The LR(1) Parse Table (continued)

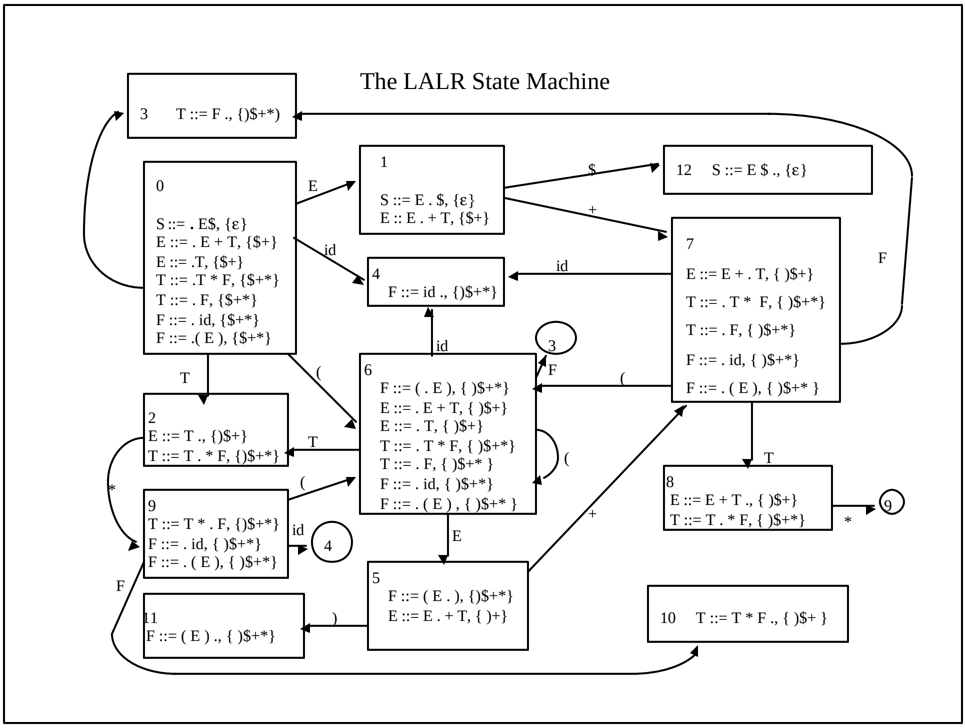
	ACTION						GOTO		
state	+	*	id	(	)	$	E	T	F
12	r2	s21			r2				
13	r3	s20			r3				
14	r5	r5			r5				
15			s16	s15			17	13	14
16	r6	r6			r6				
17	s18				s19				
18			s16	s15				12	14
19	r7	r7			r7				
20	r2		s16	s15					21
21	r4	r4							

LALR Parsing

- We seek to simplify LR(1) without losing all its power. We do this by combining states that match in all regards other than the lookahead.
- Because this process can be cumbersome when there are a large number of states, it is often done by building something that is very similar to the SLR(1) machine.

LALR Parsing (continued)

LALR States	LR(1) States combined to form this LALR State
0	0
1	1
2	2 & 13
3	3 & 14
4	4 & 16
5	5 & 17
6	6 & 15
7	7 & 18
8	8 & 12
9	9 & 20
10	10 & 21
11	11 & 19
12	22


The LALR State Machine
3 T ::= F ., {)$+*)
0
S ::= . E$, {ε}
E ::= . E + T, {$+}
E ::= .T, {$+}
T ::= .T * F, {$+*}
T ::= . F, {$+*}
F ::= . id, {$+*}
F ::= .(E), {$+*}
1
S ::= E . $, {ε}
E :: E . + T, {$+}
12 S ::= E $., {ε}
4
F ::= id ., {)$+*}
7
E ::= E + . T, {)$+}
T ::= . T * F, {)$+*}
T ::= . F, {)$+*}
F ::= . id, {)$+*}
F ::= . (E), {)$+* }
6
F ::= (. E), {)$+*}
E ::= . E + T, {)$+}
E ::= . T, {)$+}
T ::= . T * F, {)$+*}
T ::= . F, {)$+* }
F ::= . id, {)$+*}
F ::= . (E) , {)$+* }
2
E ::= T ., {)$+}
T ::= T . * F, {)$+*}
9
T ::= T * . F, {)$+*}
F ::= . id, {)$+*}
F ::= . (E), {)$+*}
8
E ::= E + T ., {)$+}
T ::= T . * F, {)$+*}
5
F ::= (E .), {)$+*}
E ::= E . + T, {)+}
11
F ::= (E) ., {)$+*}
10 T ::= T * F ., {)$+ }
E
id
$
+
F
T
(
*
)