Compilers

Spring 2009

Homework 3

Solution

Problem 1 [100 points]: LR Parsing Algorithm

Given the grammar below already augmented with the basic EOF production (0) answer the following questions:

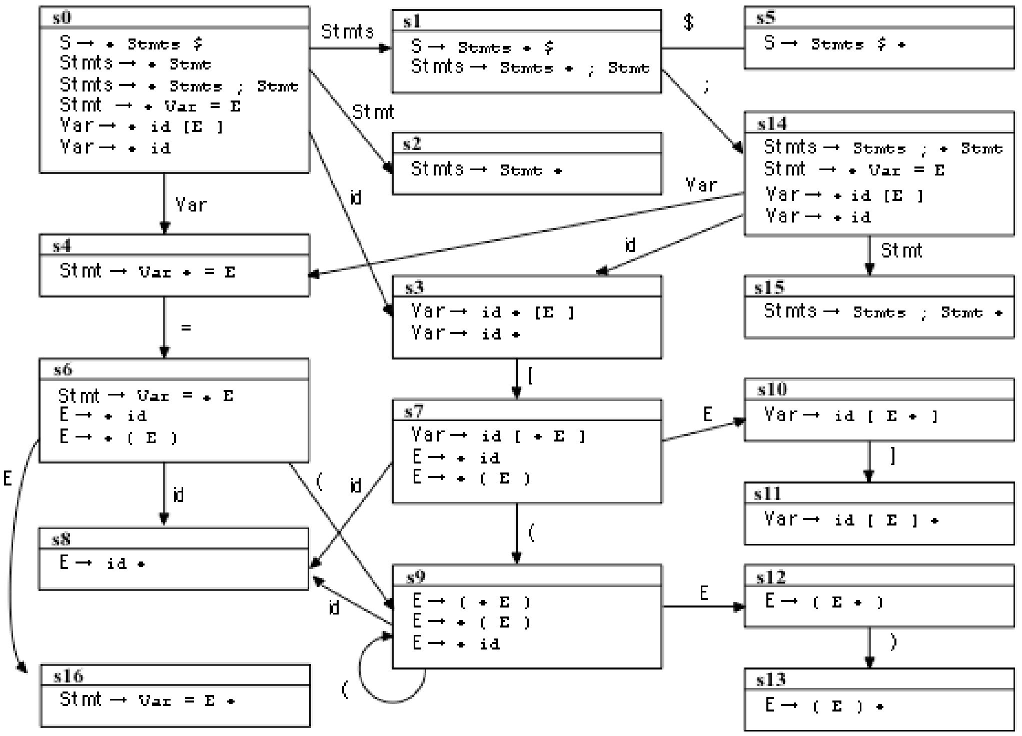
```
(0) S        → Stmts $
(1) Stmts    → Stmt
(2) Stmts    → Stmts ; Stmt
(3) Stmt     → Var = E
(4) Var      → id [E ]
(5) Var      → id
(6) E        → id
(7) E        → ( E )
```

a) [20 points] Construct the set of LR(0) items and the DFA capable of recognizing it.
b) [20 points] Construct the LR(0) parsing table and determine if this grammar is LR(0). Justify.
c) [10 points] Is the SLR(0) DFA for this grammar the same as the LR(0) DFA? Why?
d) [10 points] Is this grammar SLR(0)? Justify by constructing its table.
e) [20 points] Construct the set of LR(1) items and the DFA capable of recognizing it.
f) [10 points] Construct the LR(1) parsing table and determine if this grammar is LR(1). Justify.
g) [10 points] How would you derive the LALR(1) parsing table this grammar? What is the difference between this table and the table found in a) above?

Answers:

a) [20 points] Construct the set of LR(0) items and the DFA capable of recognizing it.

The figure below depicts the FA that recognizes the set of valid LR(0) items for this grammar.



b) [20 points] Construct the LR(0) parsing table and determine if this grammar is LR(0). Justify.

Based on the DFA above we derive the LR parsing table below where we noted a shift/reduce conflict in state 3. In this state the presence of a '[' indicates that the parse can either reduce using the production 5 or shift by advancing to state s6. Note that by reducing it would then be left in a state possible s0 where the presence of the '[' would lead to an error. Clearly, this grammar is not suitable for the LR(0) parsing method.

	Terminals								**Goto**			
State	**id**	**;**	**=**	**[**	**]**	**(**	**)**	**$**	**Stmts**	**Stmt**	**E**	**Var**
0	s3								g1	g2		g4
1		s14						s5				
2	r(1)	r(1)	r(1)	r(1)	r(1)	r(1)	r(1)	r(1)				
3	r(5)	r(5)	r(5)	s7/r(5)	r(5)	r(5)	r(5)	r(5)				
4			s6									
5								acc				
6	s8					s9					g16	
7	s8					s9					g10	
8	r(6)	r(6)	r(6)	r(6)	r(6)	r(6)	r(6)	r(6)				
9	s8					s9					g12	
10					s11							
11	r(4)	r(4)	r(4)	r(4)	r(4)	r(4)	r(4)	r(4)				
12							s13					
13	r(7)	r(7)	r(7)	r(7)	r(7)	r(7)	r(7)	r(7)				
14	s3									g15		g4
15	r(2)	r(2)	r(2)	r(2)	r(2)	r(2)	r(2)	r(2)				
16	r(3)	r(3)	r(3)	r(3)	r(3)	r(3)	r(3)	r(3)				

c) [10 points] Is the SLR(1) DFA for this grammar the same as the LR(0) DFA? Why?

The same. The states and transitions are the same as only the procedure to build the parse table is different. For this method of construction of the parsing table we include the production “reduce A → α” for all terminals ”a” in FOLLOW(A). The table below is the resulting parse table using the SLR table construction algorithm, also known as SLR(1) although it uses the DFA constructed using the LR(0) items.

For this specific grammar the FOLLOW set is as shown below:

FOLLOW(Stmts) = { $, ; }	FOLLOW(Stmt) = { $, ; }
FOLLOW(E) = { $, ; ,] ,) }	FOLLOW(Var) = { = }

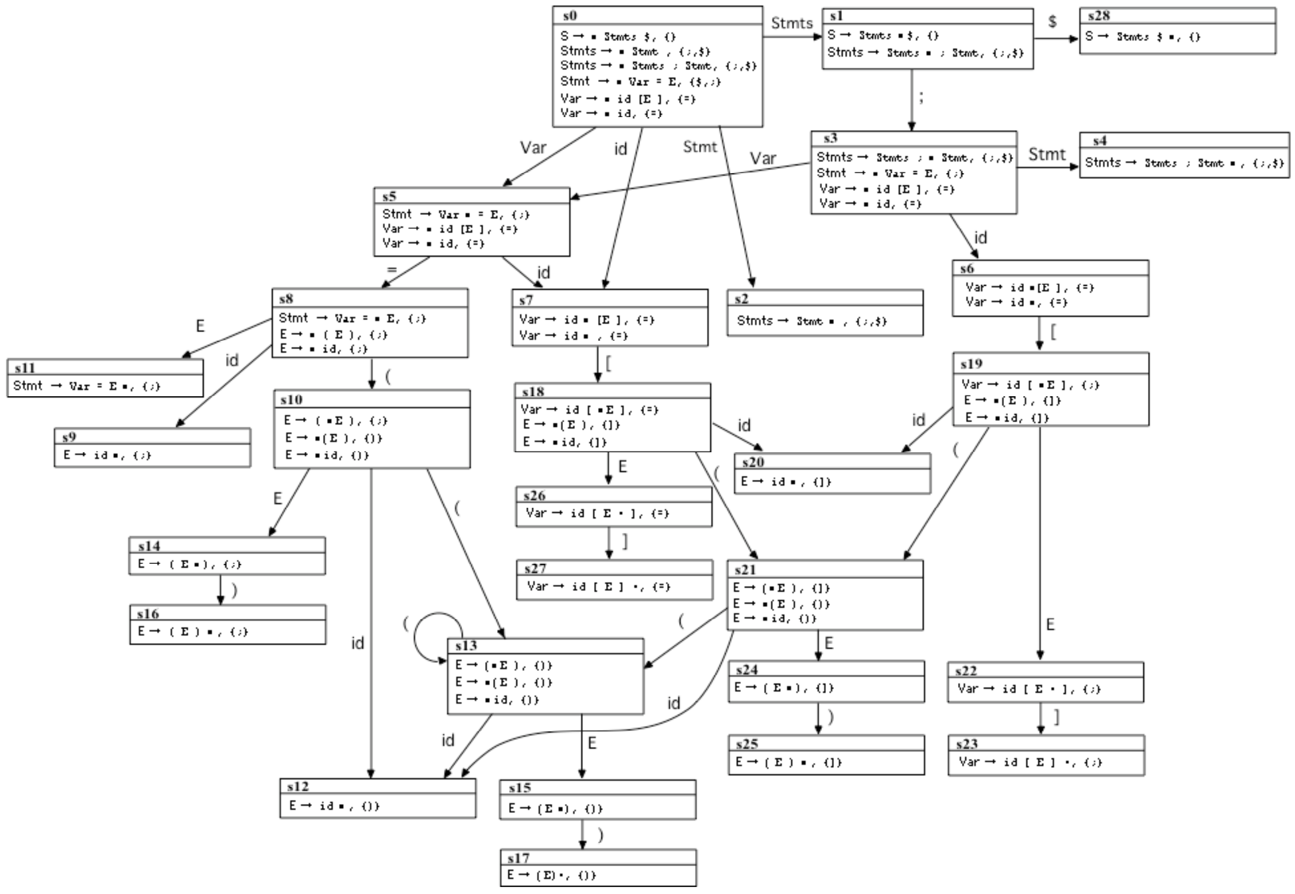
	Terminals								**Goto**			
State	**id**	**;**	**=**	**[**	**]**	**(**	**)**	**$**	**Stmts**	**Stmt**	**E**	**Var**
0	s3								g1	g2		g4
1		s14						s5				
2		r(1)						r(1)				
3			r(5)	s7								
4			s6									
5								acc				
6	s8					s9					g16	
7	s8					s9					g10	
8		r(6)			r(6)		r(6)	r(6)				
9	s8					s9					g12	
10					s11							
11			r(4)									
12							s13					
13		r(7)			r(7)		r(7)	r(7)				
14										g15		g4
15		r(2)						r(2)				
16		r(3)						r(3)				

Notice that because we have used the FOLLOW of Var to limit the use of the reduction action for this table we have eliminated the shit/reduce conflict in this grammar.

d) [10 points] Is this grammar SLR(1)? Justify by constructing its table.

As can be seen in state 3 there is no longer a shift/reduce conflict. Essentially a single look-ahead symbol is enough to distinguish a single action to take in any context.

e) [20 points] Construct the set of LR(1) items and the DFA capable of recognizing them.



As can be seen there number of states in this new DFA is much larger when compared to the DFA that recognizes the LR(0) sets of items. There are many states with identical core items thus differing only in the look-ahead and can thus be merged as suggested by the procedure to construct LALR parsing tables. For instance states {s14, s15, s24} could be merged into a single state. The same is true for states in the sets {s6, s7}, {s16, s17, s29}, {s22, s26}, {s13, s21}, {s23, s27}, {s18, s19} and {s9, s12, s20} thus substantially reducing the number of states as it will be seen in the next point in this exercise.

f) [10 points] Construct the LR(1) parsing table and determine if this grammar is LR(1). Justify.

	Terminals								**Goto**			
State	**id**	**;**	**=**	**[**	**]**	**(**	**)**	**$**	**Stmts**	**Stmt**	**E**	**Var**
0	s7								g1	g2		g5
1		s3						s28				
2								r(1)				
3	s6									g4		g5
4		r(2)										
5	s7		s8									
6			r(5)	s19								
7			r(5)	s18								
8	s9					s10					g11	
9		r(6)										
10	s12					s13					g14	
11		r(3)										
12							r(6)					
13	s12					s13					g15	
14						s16						
15						s17						
16		r(7)										
17							r(7)					
18	s20					s21					g26	
19	s20					s21					g22	
20					r(6)							
21	s12					s13					g24	
22				s23								
23		r(4)										
24							s25					
25					r(7)							
26				s27								
27			r(4)									
28								acc				

Clearly, and as with the SLR(1) table construction method there are no conflicts in this parse table and the grammar is therefore LR(1).

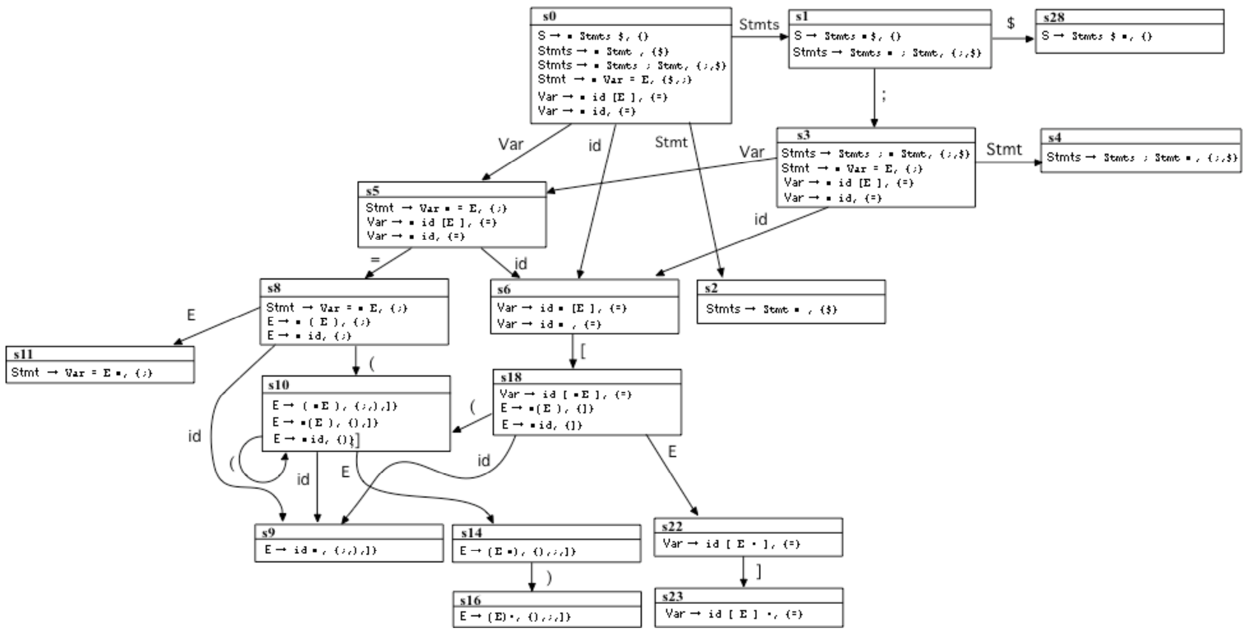
g) [10 points] How would you derive the LALR(1) parsing table this grammar? What is the difference between this table and the table found in a) above?

There are many states with very similar core items that differ only on the look-ahead and can thus be merged as suggested by the procedure to construct LALR parsing tables. For instance states {s14, s15, s24} could be merged into a single state. The same is true for states {s16, s17, s25} and the pairs of states {s26, s22} and {s27, s23}, {s9, s12, s20} thus substantially reducing the number of states. We further note that {s18, s19} are two identical states which also means that we can merge states {s6, s7} and {s10, s13, s21}.

The table below reflects these merge operations resulting in a much smaller table which is LALR as there are no conflicts due to the merging of states with the same core items.

	Terminals								Goto			
State	id	;	=	[	]	(	)	$	Stmts	Stmt	E	Var
0	s6								g1	g2		g5
1		s3						s28				
2								r(1)				
3	s6									g4		g5
4		r(2)										
5	s6		s8									
6			r(5)	s18								
7	---	---	---	---	---	---	---	---	---	---	---	---
8	s9					s10					g11	
9		r(6)			r(6)		r(6)					
10	s9					s10					g14	
11		r(3)										
12	---	---	---	---	---	---	---	---	---	---	---	---
13	---	---	---	---	---	---	---	---	---	---	---	---
14						s16	s16					
15	---	---	---	---	---	---	---	---	---	---	---	---
16		r(7)			r(7)		r(7)					
17	---	---	---	---	---	---	---	---	---	---	---	---
18	s9					s10					g22	
19	---	---	---	---	---	---	---	---	---	---	---	---
20	---	---	---	---	---	---	---	---	---	---	---	---
21	---	---	---	---	---	---	---	---	---	---	---	---
22				s23								
23		r(4)	r(4)									
24	---	---	---	---	---	---	---	---	---	---	---	---
25	---	---	---	---	---	---	---	---	---	---	---	---
26	---	---	---	---	---	---	---	---	---	---	---	---
27	---	---	---	---	---	---	---	---	---	---	---	---
28								acc				

After this state simplification we get the DFA below. This DFA is identical to the first DFA found using the LR(0) items but the additional information on the look-ahead token allows for a better table parsing construction method that does not have shift/reduce conflicts.


s0
S → ▪ Stmts $, {}
Stmts → ▪ Stmt , {$}
Stmts → ▪ Stmts ; Stmt, {;,$}
Stmt → ▪ Var = E, {$,;}
Var → ▪ id [E], {=}
Var → ▪ id, {=}
Stmts
s1
S → Stmts ▪ $, {}
Stmts → Stmts ▪ ; Stmt, {;,$}
$
s28
S → Stmts $ ▪, {}
;
s3
Stmts → Stmts ; ▪ Stmt, {;,$}
Stmt → ▪ Var = E, {;}
Var → ▪ id [E], {=}
Var → ▪ id, {=}
Stmt
s4
Stmts → Stmts ; Stmt ▪, {;,$}
Var
id
Stmt
Var
id
s5
Stmt → Var ▪ = E, {;}
Var → ▪ id [E], {=}
Var → ▪ id, {=}
=
id
s8
Stmt → Var = ▪ E, {;}
E → ▪ (E), {;}
E → ▪ id, {;}
s6
Var → id ▪ [E], {=}
Var → id ▪ , {=}
s2
Stmts → Stmt ▪ , {$}
E
s11
Stmt → Var = E ▪, {;}
(
[
s10
E → (▪E), {;,),]}
E → ▪(E), {),]}
E → ▪id, {)}]
s18
Var → id [▪E], {=}
E → ▪(E), {]}
E → ▪id, {]}
(
id
(
id
E
id
E
s9
E → id ▪, {;,),]}
s14
E → (E ▪), {),;,]}
)
s16
E → (E)▪, {),;,]}
s22
Var → id [E ▪], {=}
]
s23
Var → id [E] ▪, {=}