Homework 3

Dragon:
4.6.2, 4.6.3, 4.6.5, 4.6.6, 4.6.9
4.7.1, 4.7.4, 4.7.5
4.8.1, 4.8.2

Tiger:
3.8, 3.10, 3.11, 3.12, 3.13

Dragon Book

4.6.2: Construct the SLR sets of items for the (augmented) grammar of Exercise 4.2.1. Compute the GOTO function for these sets of items. Show the parsing table for this grammar. Is the grammar SLR?

$$S \rightarrow S\,S + \mid S\,S * \mid a$$

Solution:
Item Sets:

State 0
S->.SS+
S->.SS*
S->.a

State 1
S->S.S+
S->S.S*
S->.SS+
S->.SS*
S->.a

State 2
S->a.

State 3
S->SS.+
S->SS.*
S->S.S+
S->S.S*
S->.SS+
S->.SS*
S->.a

State 4
S->SS+.

State 5
S->SS*.

Goto Function:
Goto(0,a)=2 Goto(0,S)=1
Goto(1,a)=2 Goto(1,S)=3
Goto(3,*)=5 Goto(3,+)=4 Goto(3,a)=2 Goto(3,S)=3
Goto(4,$)=Accept

SLR Table:

	*	+	a	$	S
0			s2		1
1			s2	a	3
2	r3	r3	r3	r3	
3	s5	s4	s2		3
4	r1	r1	r1	r1	
5	r2	r2	r2	r2	

No Conflicts in SLR table. This **IS** a SLR grammar

4.6.3 Show the actions of your parsing table from Exercise 5.6.2 on the input aa*a+.

Solution:

	Stack	Input	Action
1	0	aa*a+$	Shift
2	0 2	a*a+$	Reduce S->a
3	0 1	a*a+$	Shift
4	0 1 2	*a+$	Reduce S->a
5	0 1 3	*a+$	Shift
6	0 1 3 5	a+$	Reduce S->SS*
7	0 1	a+$	Shift
8	0 1 2	+$	Reduce S->a
9	0 1 3	+$	Shift
10	0 1 3 4	$	Reduce S->SS+
11	0 1	$	Accept

4.6.5 Show that the following grammar:

$$\begin{array}{lcl} S & \rightarrow & A\ a\ A\ b \mid B\ b\ B\ a \\ A & \rightarrow & \epsilon \\ B & \rightarrow & \epsilon \end{array}$$

is LL(1) but not SLR(1).

Solution:
“ab” and “ba” can be determined by “a” and “b”, S is LL(1)

in SLR, consider State0: S->.AaAb S->.BbBa A->ε. B->ε. when we reduce “ε”, we can’t decide reduce to A or B, this is a reduce-reduce conflict, S is not SLR(1)

4.6.6 Show that the following grammar:

$$\begin{array}{lcl} S & \rightarrow & S\ A \mid A \\ A & \rightarrow & a \end{array}$$

is SLR(1) but not LL(1).

Solution:
S -> Sa | a
S is infinite sequence of “a”, we can’t determine how many “a” when we meet “a”, S is not LL(1)
We can construct an SLR table with no conflicts, S is SLR(1)

	a	S	S	A
0	s3		1	2
1	s3			4
2	r1	r1		
3	r2	r2		
4		a		

In 4.6.5 and 4.6.6, you can construct LL parsing table and SLR parsing table to show whether the grammar is LL(1) or SLR(1).

4.6.9 The following is an ambiguous grammar:

$$\begin{array}{lcl} S & \rightarrow & A\ S \mid b \\ A & \rightarrow & S\ A \mid a \end{array}$$

Construct for this grammar its collection of sets of LR(0) items. If we try to build an LR-parsing table, there are certain conflicting actions. What are they? Suppose we tried to use the parsing table by nondeterministically choosing a possible action whenever there is a conflict. Show all the possible sequences of actions on input abab.

Solution:
LR(0) items:

State 0
S->.AS
A->.SA
A->.a
S->.b

State 1
S->A.S
S->.AS
S->.b
A->.SA
A->.a

State 2
A->S.A
A->.SA
A->.a
S->.AS
S->.b

State 3
A->a.

State 4
S->b.

State 5
S->AS.
A->S.A
A->.SA
A->.a
S->.AS
S->.b

State 6
A->SA.
S->A.S
S->.AS
A->.SA
A->.b

LR parsing table:

	a	b	$	S’	S	A
0	s4	s3			1	2
1	s4	s7	a		6	5
2	s4	s3			8	2
3	r2	r2	r2			
4	r4	r4				
5	r3|s4	r3|s7			9	10
6	s4	s7			6	5
7	r2	r2				
8	r1|s4	r1|s7	r1		6	5
9	r1|s4	r1|s7			6	5
10	s4	s7			9	10

4.7.1 Construct the
a) canonical LR and
b) LALR
sets of items for the grammar S->SS+|SS*|a

Solution:
a)

State 0
S->.SS+, $a
S->.SS*, a
S->.a, a

State 6
S->SS*..a

State 8
S->SS+.,+*a

State 1
S->S.S+, $a
S->S.S*, a
S->.SS+, +*a
S->.SS*, +*a
S->.a, +*a

State 9
S->SS*.,+*a

State 2
S->a., a

State 4
S->a.. +*a

State 5
S->SS+..$a

State 3
S->SS.+, $a
S->SS.*, a
S->S.S+, +*a
S->S.S*, +*a
S->.SS+, +*a
S->.SS*,+*a
S->.a,+*a

State 7
S->SS.+, +*a
S->SS.*, +*a
S->S.S+, +*a
S->S.S*, +*a
S->.SS+, +*a
S->.SS*, +*a
S->.a, +*a

b)

State 0
S->.SS+, $a
S->.SS*, a
S->.a, a

State 1
S->S.S+, $a
S->S.S*, a
S->.SS+, +*a
S->.SS*, +*a
S->.a, +*a

State 2
S->a., +*a

State 4
S->SS+.+*a$

State 5
S->SS*.+*a,

State 3
S->SS.+, +*a$
S->SS.*, +*a
S->S.S+, +*a
S->S.S*, +*a
S->.SS+, +*a
S->.SS*, +*a
S->.a, +*a

4.7.4 Show that the following grammar

$$\begin{array}{lcl} S & \rightarrow & A\,a \mid b\,A\,c \mid d\,c \mid b\,d\,a \\ A & \rightarrow & d \end{array}$$

is LALR(1) but not SLR(1).

Solution:

SLR Table:

	a	b	c	d	$	S’	S	A
0		s3		s4			1	2
1					a			
2	s5							
3				s7				6
4	r5		s8|r5					
5					r1			
6			s9					
7	s10|r5		r5					
8					r3			
9					r2			
					r4			

Shift-reduce conflicts, this grammar is not SLR(1).
Because when we meet “d”, we can’t decide whether reduce to A or shift(S->dc|bda).

LALR Table:

	a	b	c	d	$	S’	S	A
0		s3		s4			1	2
1					a			
2	s5							
3				s7				6
4	r5		s8					
5					r1			
6			s9					
7	s10		r5					
8					r3			
9					r2			
					r4			

No conflicts, this grammar is LALR(1).

4.7.5 Show that the following grammar

$$\begin{array}{lll} S & \rightarrow & A\ a \mid b\ A\ c \mid B\ c \mid b\ B\ a \\ A & \rightarrow & d \\ B & \rightarrow & d \end{array}$$

is LR(1) but not LALR(1)

Solution:

LR Table:

	a	b	c	d	$	S’	S	A	B
0		s3		s5			1	2	4
1					a				
2	s6								
3				s9				7	8
4			s10						
5	r5		r6						
6					r1				
7			s11						
8	s12								
9	r6	r5							
10				r3					
11				r2					
12				r4					

No conflicts, this grammar is LR(1)

LALR table:

	a	b	c	d	$	S’	S	A	B
0		s3		s5			1	2	4
1					a				
2	s6								
3				s5				7	8
4			s9						
5	r5|r6		r5|r6						
6					r1				
7			s10						
8	s11								
9				r3					
10				r2					
11				r4					

Reduce-reduce conflicts, this grammar is not LALR(1).
Because when we meet “bd”, we can’t determine whether reduce to “bA” or “bB”

4.8.1 The following is an ambiguous grammar for expressions with n binary, infix operators, at n different levels of precedence:

$$E \rightarrow E\ \theta_1\ E \mid E\ \theta_2\ E \mid \cdots\ E\ \theta_n\ E \mid (\ E\ ) \mid \textbf{id}$$

a) As a function of n, what are the SLR sets of items?
b) How would you resolve the conflicts in the SLR items so that all operators are left associative, and θ1 takes precedence over θ2, which takes precedence over θ3, and so on?
c) Show the SLR parsing table that results from your decisions in part (b).

$$\begin{array}{lcl} E_1 & \rightarrow & E_1\ \theta\ E_2 \mid E_2 \\ E_2 & \rightarrow & E_2\ \theta\ E_3 \mid E_3 \\ & \cdots & \\ E_n & \rightarrow & E_n\ \theta\ E_{n+1} \mid E_{n+1} \\ E_{n+1} & \rightarrow & (\ E_1\ ) \mid \textbf{id} \end{array}$$

Figure 4.55: Unambiguous grammar for n operators

d) Repeat parts (a) and (c) for the unambiguous grammar, which defines the same set of expressions, shown in Fig. 4.55.
e) How do the counts of the number of sets of items and the sizes of the tables for the two (ambiguous and unambiguous) grammars compare? What does that comparison tell you about the use of ambiguous expression grammars?

Solution:
(a) SLR sets: (θ stands for θ1 to θn) total:6+2n

State 0
E->.E θ E
E->.(E)
E->.id

State 1
E->E. θ E

State 2
E->.E θ E
E->(.E)
E->.(E)
E->.id

State 3
E->id.

State 4i
E->E θ i. E
E->.E θ E
E->.(E)
E->.id

State 5
E->(E.)
E->E. θ E

State 6i
E->E θ i E.
E->E. θ E

State 7
E->(E).

(b) If there are conflicts, then always shift the highest operator
(c)

	(	)	θ i	id	$	E

0	s2			s3		1
1			s4i			
2	s2			s3		5
3		r(E->id)	r(E->id)		r(E->id)	
4	s2			s3		6
5		s7	s4i			

6		r(E->E θ iE)
		for i>1
			s4i for i>1
			r(E->E θ iE)
			for i=1
					a	

7		r(E->(E))	r(E->(E))		r(E->(E))	

(d) total sets: 5+3n
table ……

(e) from (d) and (e), we can conclude that ambiguous grammar has much smaller items and parsing table, and can easily remove conflicts

Dragon Book

4.8.2 In Fig 4.56 is a grammar for certain statements, similar to that discussed in Ex 4.4.12. Again, e and s are terminals standing for conditional expressions and "other statements," respectively.

stmt -> if e then stmt
| if e then stmt else stmt
| while e do stmt
| begin list end
| s
list -> list; stmt
| stmt

a) Build an LR parsing table for this grammar, resolving conflicts in the usual way for the dangling-else problem.

b) Implement error correction by filling in the blank entries in the parsing table with extra reduce-actions or suitable error-recovery routines.

c) Show the behavior of your parser on the following inputs:
i) if e then s; if e then s end
ii) while e do begin s; if e then s; end

Answer:

Rewrite the grammar in order:

0: P -> stmt $
1: stmt -> if e then stmt
2: stmt -> if e then stmt else stmt
3: stmt -> while e do stmt
4: stmt -> begin list end
5: stmt -> s
6: list -> list; stmt

    7: list -> stmt

and a possible parser table is

	;	begin	e	do	else	if	end	s	then	while	$	P	stmt	list	E
0		s4				s2		s5		s3			1		
1											a				
2			s6												
3			s7												
4		s4				s2		s5		s3			9	8	
5	r5				s17	e1	r5				r5				4
6									s10						
7				s11											
8	s13						s12								
9	r7						r7								
10		s4				s2		s5		s3			14		
11		s4				s2		s5		s3			15		
12	r4				s17		r4				r4				
13		s4				s2	e2	s5		s3			16		
14	r1				s17		r1				r1				
15	r3				s17		r3				r3				
16	r6						r6								
17		s4				s2		s5		s3			18		
18	r2				s17		r2				r2				

e1: push State 4 (corresponding to begin); issue “missing begin”

e2: remove “;” from input; issue “redundant semicolon”

Tiger Book

3.8 Make up a tiny grammar containing left recursion, and use it to demonstrate that left recursion is not a problem for LR parsing. Then show a small example comparing growth of the LR parse stack with left recursion versus right recursion.

Answer：

Consider following grammar:

    0: S -> P $

    1: P -> P id

2: P -> id

and a program “a b c”, where “a”, “b” and “c” are id’s.

The progress of LR parsing is showed below:

Stack	Input	Action
|	a b c $	shift
| a	b c $	reduce P -> id
| P	b c $	shift
| P b	c $	reduce P -> P id
| P	c $	shift
| P c	$	reduce P -> P id
| P	$	accept

3.10 Diagram the LR(1) states for the grammar of Exercise 3.7 (without left-factoring), and construct the LR(1) parsing table. Indicate clearly any conflicts.

0. S → G $

1. G → P

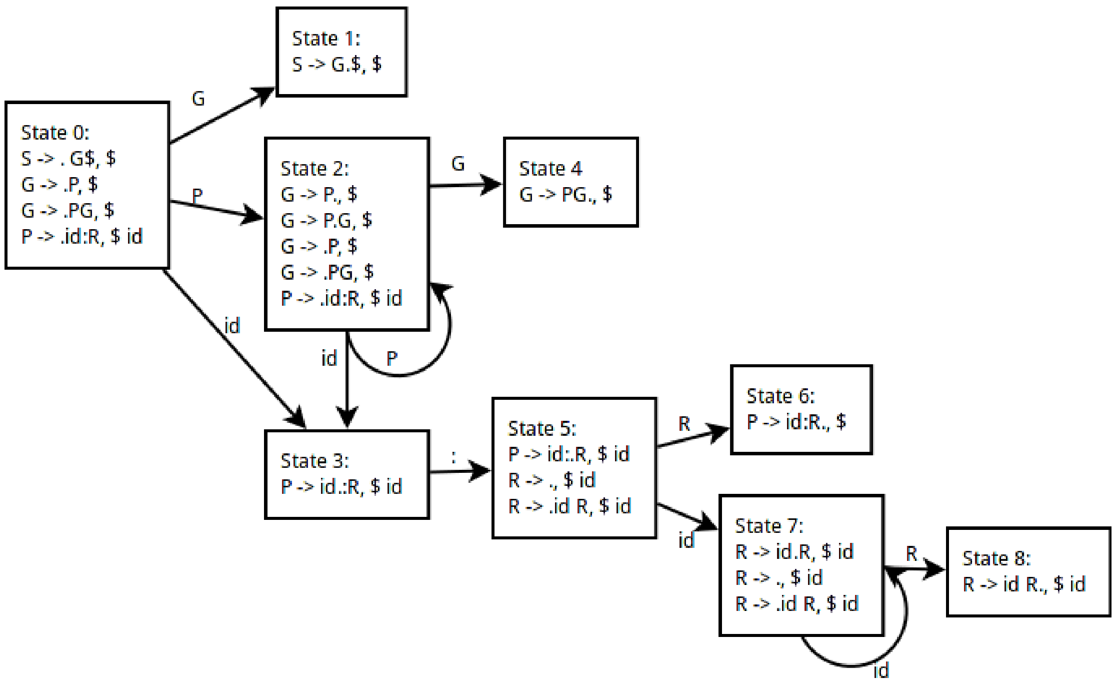
2. G → PG

3. P → id : R

4. R →

5. R → id R

Answer:



	:	ε	id	$	S	G	P	R
0			s3			1	2	
1				a				
2			s3	r1		4	2	
3	s5							
4				r2				

5			r4 | s7	r4				6
6			r3	r3				
7			r4 | s7	r4				8
8			r5	r5				

对于(5,id)和(7,id)中的冲突，考虑对程序 id:id:的分析：

Stack	Input	Actio
|	id:id:	s3
| id	:id:	s5
| id :	id:	either r4 or s7 is ok; in the first case, the program is in the form of “PG”, where P is “id:” and G is “id:”; in the later case ,the program is in the form of “P”, where P is “id:id:”.

3.11 Construct the LR(0) states for this grammar, and then determine whether it is an SLR grammar.

0. S → B $

1. B → id P

2. B → id *(E]

3. P →

4. P → (E)

5. E → B

6. E → B, E

Answer:

State 0:

S -> •B$

B -> •id P

B -> •id*(E]

State 1:

S -> B•$

State 2:

B -> id•P

B -> id•*(E]

P -> •

P -> •(E)

State 3:

B -> id P•

State 4:

B -> id*•(E]

State 5:

P -> (•E)

E -> •B

E -> •B,E

B -> •id P

B -> •id*(E]

State 6:

B->i*(•E]

E->•B

E->•B,E

B->•iP

B->•i*(E]

State 7:

P->(E•)

State 8:

E->B•

E->B•,E

State 9:

B->i*(E•]

State 10:

P->(E)•

State 11:

E->B,•E

E->•B

E->•B,E

B->•iP

B->•i*(E]

State 12:

B->i*(E]•

State 13:

E->B,E•

是 SLR 文法。

3.12

1. Build the LR(0) DFA for this grammar:

0. S → E $

1. E → id

2. E → id (E)

3. E → E + id

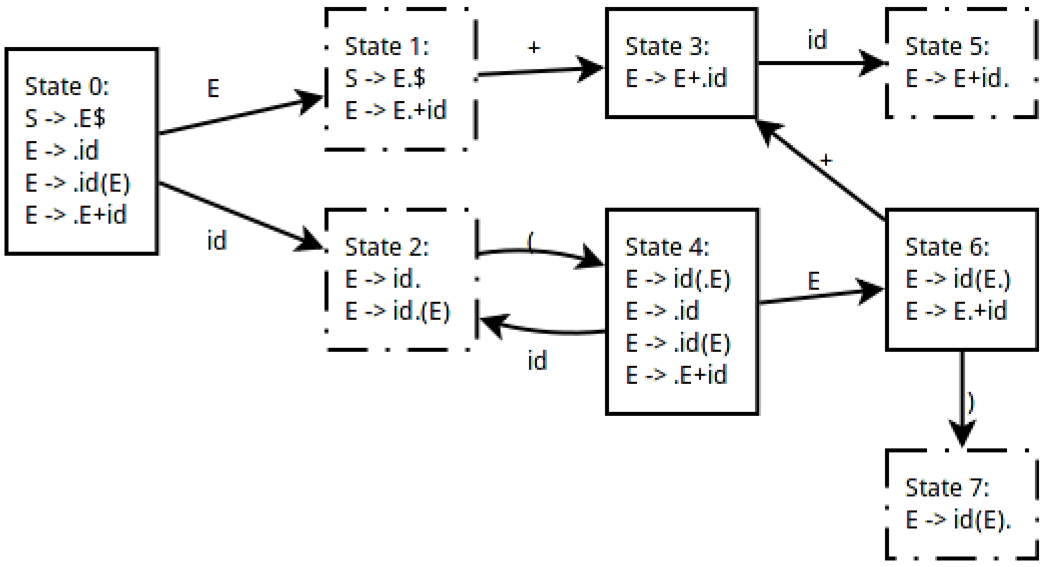
2. Is this an LR(0) grammar? Give evidence.

3. Is this an SLR grammar? Give evidence.

4. Is this an LR(1) grammar? Give evidence.

Answer

1. 虚线框为接受状态



2. 不是 LR(0)文法。State2 存在移进归约冲突

3. 是 SLR 文法.

4. 是 LR(1)文法，因为可以构造无冲突的 LR(1)分析表（此处略）。

3.13 Show that this grammar is LALR(1) but not SLR:

0. S → X $

1. X → Ma

2. X → bMc

3. X → dc

4. X → bda

5. M → d

Answer：

1. 可以构造如下闭包和无冲突的 LALR(1)分析表，所以该文法是 LALR(1)文法

State 0:	**State 3:**	**State 6:**
S -> •X$, $	X -> b•Mx, $	X -> bM•c, $
X -> • Ma, $	X -> b•da, $	**State 7:**
X -> •bMs, $	M -> •d, c	X -> bd•a, $

X -> •dc, $

X -> •bda, $

M -> •d, a

State 1:

S -> X•, $

State 2:

X -> M•a, $

State 4:

X -> d•c, $

M -> d•, a

State 5:

X -> Ma•, $

M -> d•, c

State 8:

X -> dc•, $

State 9:

X -> bMc•, $

State 10:

X -> bda•, $

	a	b	c	d	$	S	X	M
0		s3		s4			1	2
1					a			
2	s5							
3				s7				6
4	r5		s8					
5					r1			
6			s9					
7	s10		r5					
8					r3			
9					r2			
10					r4			

2. 尝试构造如下闭包和 SLR 分析表。因为 SLR 分析表中存在冲突，所以该文法不是 SLR 文法。

State 0:

S -> •X$

X -> • Ma

X -> •bMs

X -> •dc

X -> •bda

M -> •d

State 1:

S -> X•

State 2:

X -> M•a

State 3:

X -> b•Mx

X -> b•da

M -> •d

State 4:

X -> d•c

M -> d•

State 5:

X -> Ma•

State 6:

X -> bM•c

State 7:

X -> bd•a

M -> d•

State 8:

X -> dc•

State 9:

X -> bMc•

State 10:

X -> bda•

	a	b	c	d	$	S	X	M
0		s3		s4			1	2
1					a			
2	s5							
3				s7				6
4	r5		s8 | r5					
5					r1			
6			s9					
7	s10 | r5		r5					
8					r3			

9					r2			
10					r4			