编译原理（第3版）王生原课后习题答案

（若发现问题，请及时告知）

1 设有文法 G[S]：

$$S \rightarrow aSb \mid aab$$

若针对该文法设计一个自顶向下预测分析过程，则需要向前察看多少个输入符号？

参考解答：

需要向前察看 3 个单词。若向前察看 3 个单词是 aab 时，可选第 2 个分支；aaa 时，可选第 1 个分支。

2 设有文法 $G[E]$：

$$E \rightarrow iC$$
$$C \rightarrow EOC \mid \varepsilon$$
$$O \rightarrow + \mid *$$

试针对该文法填写下表

G 中的规则 r	$First(rhs(r))$	$Follow(lhs(r))$	$PS(r)$
$E \rightarrow iC$			
$C \rightarrow EOC$			
$C \rightarrow \varepsilon$		此处不填	
$O \rightarrow +$			
$O \rightarrow *$		此处不填	

其中，$rhs(r)$ 表示产生式 r 的右部，$lhs(r)$ 表示产生式 r 的左部

参考解答：

根据First集合定义，以及各产生式右端子串，不难计算各First(*rhs*(*r*))集合元素。例如，对于产生式

$$C \rightarrow EOC$$

First(EOC) = First($iCOC$) = { i }

根据Follow集合的计算过程，容易得到输入结束符 # 进Follow(E)。然后针对每一个产生式依次计算每一个非终结符的Follow集合元素。例如，对于产生式

$$C \rightarrow EOC$$

有，First(O)的元素进Follow(E)，故，Follow(E)={ #，+，*}。

根据产生式

$E \to iC$

Follow(E)的元素进Follow(C)，故，Follow(C)={ #，+，*}。

根据产生式

$C \to EOC$

First(C)的元素进Follow(O)，由于，ε ∈ First(C)，故，Follow(lhs(r))，即Follow(C) ，的元素进Follow(O)。所以，Follow(O)={ i,#,+,* }

最终计算结果如下表所示：

G 中的规则 r	First（rhs（r））	Follow（lhs（r））	PS （r）
E → iC	{ i }	{ #, +, * }	{ i }
C → EOC	{ i }	{ #, +, * }	{ i }
C → ε	{ ε }		{ #, +, * }
O → +	{ + }	{ i, #, +, * }	{ + }
O → *	{ * }		{ * }

3 设有文法 G[E]：

$E \to TA$
$A \to \vee TA \mid \varepsilon$
$T \to FB$
$B \to \wedge FB \mid \varepsilon$
$F \to (E) \mid i$

试针对该文法填写下表

G中的规则 r	first (rhs(r))	follow (lhs(r))	PS (r)
$E \to TA$			
$A \to \vee TA$			
$A \to \varepsilon$		此处不填	
$T \to FB$			
$B \to \wedge FB$			
$B \to \varepsilon$		此处不填	
$F \to (E)$			
F → i		此处不填	

其中, *rhs(r)* 表示产生式 *r* 的右部，*lhs(r)* 表示产生式 *r* 的左部

参考解答： 如下表。

G 中的规则 r	first (rhs(r))	follow (lhs(r))	PS (r)
E → TA	i，(	)，#	i，(
A → ∨TA	∨	)，#	∨
A → ε	ε	此处不填	)，#
T → FB	i，(	∨，)，#	i，(
B → ∧FB	∧	∨，)，#	∧
B → ε	ε	此处不填	∨，)，#
F → (E)	(	∧,∨，)，#	(
F → i	i	此处不填	i

4 试验证下列文法 G［S］是 LL(1) 文法：

$$S \to P \mid \varepsilon$$

$$P \to (\ P\ )\ P \mid a$$

其中 （，），以及 a 为终结符

参考解答：

先对于每条语法规则 r，计算 $PS(r)$。

G 中的规则 r	First (rhs(r))	Follow (lhs(r))	PS (r)
$S \to P$	a，(	#	a，(
$S \to \varepsilon$	ε		#
$P \to (P)\,P$	(	)，#	(
$P \to a$	a		a

因为 $PS(S \to P) \cap PS(S \to \varepsilon)=\phi$ 及 $PS(P \to (P)\ P) \cap PS(P \to a)=\phi$，所以文法 $G[S]$ 是 LL(1) 文法。

5 计算下列文法中每个非终结符的 First 集和 Follow 集，以及每个产生式的预测集合，并判断该文法是否 LL(1)文法（说明原因）：

（1）文法 $G_1[S]$：

$$S \to (S) \mid aT$$
$$T \to +ST \mid \varepsilon$$

（2）文法 $G_2[S]$：

$$S \to TP$$
$$T \to +PT \mid \varepsilon$$
$$P \to (S) \mid a$$

（3）文法 $G_3[S]$：

$S \to aSa \mid bSb \mid \varepsilon$

参考解答：

（1）计算非终结符的 FIRST 集和 FOLLOW 集，结果如下：

FIRST（S）= {（，a}　　FOLLOW（S）= {#，），+}

FIRST（T）= {+，ε}　　FOLLOW（T）= {#，），+}

PS($S \to (S)$)={（}

PS($S \to aT$)={a}

PS($T \to +ST$)={+}

PS($T \to \varepsilon$)={#，），+}

因为，PS($T \to +ST$)∩PS($T \to \varepsilon$)= {+}∩{#，），+}≠Φ，所以，G(S)不是 LL(1)文法。

（2）计算非终结符的 FIRST 集和 FOLLOW 集，结果如下：

FIRST（S）= {+，（，a}　　FOLLOW（S）= {#，）}

FIRST（T）= {+，ε}　　FOLLOW（T）= {（，a}

FIRST（P）= {（，a}　　FOLLOW（P）= {#，+，（，a，）}

PS($S \to TP$)={+，（，a}

PS($T \to +PT$)={+}

PS($T \to \varepsilon$)={（，a，+}

PS($P \to (S)$)={（}

PS($P \to a$)={a}

因为，PS($T \to +PT$)∩PS($T \to \varepsilon$)= {+}∩{（，a}=Φ，所以，G(S)是 LL(1)文法。

（3）计算非终结符的 FIRST 集和 FOLLOW 集，结果如下：

FIRST（S）= {a，b，ε}　　FOLLOW（S）= {#，a，b}

PS($S \to aSa$)={a}

PS($S \to bSb$)={b}

PS($S \to \varepsilon$)={#，a，b}

因为，PS($S \to aSa$)∩PS($S \to bSb$)∩PS($TS \to \varepsilon$)≠Φ，所以，G(S)不是 LL(1)文法。

6 验证如下文法是 LL(1)文法，并基于该文法构造递归下降分析程序：

（1）文法 $G[S]$：

$S \rightarrow AB$
$A \rightarrow aA$
$A \rightarrow \varepsilon$
$B \rightarrow bB$
$B \rightarrow \varepsilon$

（2）文法 $G'[E]$：

$E \rightarrow [\ F\ ]\ E'$
$E' \rightarrow E|\varepsilon$
$F \rightarrow aF'$
$F' \rightarrow aF'\ |\varepsilon$

参考解答：

（1）计算非终结符的 FIRST 集和 FOLLOW 集，结果如下：

FIRST（S）= {a, b}　　FOLLOW（S）= {# }

FIRST（A）= {a, ε }　　FOLLOW（A）= {b, # }

FIRST（B）= {b, ε }　　FOLLOW（B）= {# }

因为

FIRST（aA）∩ FOLLOW（A）= {a }∩{b, # } = Φ

FIRST（*bB*）∩ FOLLOW（B）= {b }∩{ # } = Φ

或

PS（$A \rightarrow aA$）∩ PS（$A \rightarrow \varepsilon$）= {a }∩{b, # } = Φ

PS（$B \rightarrow bB$）∩ PS（$B \rightarrow \varepsilon$）= {b }∩{ # } = Φ

所以，G(S)是LL(1)文法。

用类似C语言写出G[E]的递归子程序，其中yylex()为取下一单词过程，变量lookahead存放当前单词。不需要考虑太多编程语言相关的细节。程序如下：

```
void ParseS( )                    // 主函数
{
    ParseA( );
    ParseB( );
}
void ParseA( )
{
    switch (lookahead)            // lookahead为下一个输入符号
    {
        case ' a' :
```

```
                MatchToken( 'a' );
                ParseA();
                break;
            case ' b' , ' #' :
                break;
            default:
                printf("syntax error \n")
                exit(0);
        }
        return A_num;
    }
    void ParseB( )
    {
        switch (lookahead)  {
            case ' b' :
                MatchToken( 'b' );
                ParseB( );
                break;
            case ' #' :
                break;
            default:
                printf("syntax error \n");
                exit(0);
        }
    }
    void  Match_Token(int expected)
    {
        if (lookahead != expected)
        {
                printf("syntax error \n")
                exit(0);
        else
            lookahead = getToken();
    }
```

（2）观察文法规则可知，可能产生规则选择冲突的规则只能是 E' → E|ε 和F' → aF' |ε 。我们只需要求出这四条规则的 PS 集合（预测集合）即可。欲求这四个 PS 集合，我们需要先求出：

First (E) = { [}　　Follow (E') = {#}

First (aF') = { a }　　Follow (F') = {] }

从而

PS (E' → E) = First (E) = { [}

PS (E' → ε) = Follow (E') = {#}

PS (F' → aF') = First (aF') = { a }

PS (F' → ε) = Follow (F') = {] }

因为

对于E' → E|ε 有： $PS(E' \rightarrow E) \cap PS(E' \rightarrow \varepsilon) = \Phi$

对于F' → aF' |ε 有： $PS(F' \rightarrow aF') \cap PS(F' \rightarrow \varepsilon) = \Phi$

所以，文法 G[E] 是 LL（1）文法。

用类似 C 语言写出 G[E]的递归子程序，其中 getToken()为取下一单词过程，变量 lookahead 为全局变量，存放当前单词。

```
void ParseE( ) {
    Match_Token ( [ );
    ParseF( );
    MatchToken ( ] );
    ParseE' ( );
}

void ParseE' ( ) {
    switch (lookahead)  {
        case [:
            ParseE( );
            break;
        case #:
            break;
        default:
            printf("syntax error \n" );
            exit(0);
    }
}

void ParseF( ) {
    MatchToken ( a );
    ParseF'  ( );
}

void ParseF' ( ) {
    switch (lookahead)  {
        case a:
            MatchToken ( a );
            ParseF'  ( );
            break;
        case ]:
            break;
        case:
            printf("syntax error \n" );
            exit(0);
    }
}

void MatchToken(int expected) {//判别当前单词是否与期望的终结符匹配
    if (lookahead != expected){
            printf("syntax error \n" );
```

```
            exit(0);
    else      // 若匹配,消费掉当前单词并调用词法分析器读入下一个单词
        lookahead = getToken();
}
```

7　给出习题 5 中所有文法的预测分析表，并根据分析表指出相应文法是否 LL(1)的，同时验证习题 5 的结果。

参考解答:

（1）文法 $G_1[S]$:

$S \to (S) \mid aT$
$T \to +ST \mid \varepsilon$

各产生式的预测集合为：

PS($S \to (S)$)={ (}

PS($S \to aT$)={ a }

PS($T \to +ST$)={ + }

PS($T \to \varepsilon$)={ #，)， + }

	(	)	a	+	#
S	$S \to (S)$		$S \to aT$		
T		$T \to \varepsilon$		$T \to +ST$ $T \to \varepsilon$	$T \to \varepsilon$

M[T，+]={ $T \to +ST$，$T \to \varepsilon$}，不是 LL(1)文法。

（2）文法 $G_2[S]$:

$S \to TP$
$T \to +PT \mid \varepsilon$
$P \to (S) \mid a$

PS($S \to TP$)={ +，(，a }

PS($T \to +PT$)={ + }

PS($T \to \varepsilon$)={ (，a}

PS($P \to (S)$)={ (}

PS($P \to a$)={ a }

	(	)	a	+	#
S	$S \to TP$		$S \to TP$	$S \to TP$	

T	$T \to \varepsilon$		$T \to \varepsilon$	$T \to +PT$	
P	$P \to (S)$		$P \to a$		

每一个表项唯一确定，所以，是LL(1)文法。

（3）文法 $G_3[S]$：

$S \to aSa \mid bSb \mid \varepsilon$

PS($S \to aSa$)={ a }

PS($S \to bSb$)={ b }

PS($S \to \varepsilon$)={ #，a，b }

	a	b	#
S	$S \to aSa$ $S \to \varepsilon$	$S \to bSb$ $S \to \varepsilon$	$S \to \varepsilon$

M[S，a]={ $S \to aSa$，$S \to \varepsilon$}，M[S，b]={ $S \to bSb$，$S \to \varepsilon$}，所以，不是LL(1)文法。

8 给出习题6中所有文法的LL(1)分析表。

参考解答:

（1）文法 $G[S]$：

$S \to AB$
$A \to aA$
$A \to \varepsilon$
$B \to bB$
$B \to \varepsilon$

的LL(1)分析表为：

	a	b	#
S	S --> AB	S --> AB	S --> AB
A	A --> aA	A --> ε	A --> ε
B		B --> bB	B --> ε

（2）文法 $G'[E]$：

$E \to [F]E'$
$E' \to E \mid \varepsilon$

$F \rightarrow aF'$

$F' \rightarrow aF' \mid \varepsilon$

的 LL(1)分析表为：

	a	[	]	#
E		E -> [F]E'		
E'		E' -> E		E' ->ε
F	F->aF'			
F'	F' ->aF'		F' -> ε	

9 基于图1中的LL(1)分析表，根据图2描述的方法，给出输入符号串*baacbd*的表驱动LL(1)分析过程。

参考解答:

步骤	下推栈	余留符号串	下一步动作
1	#*S*	*baacbd*#	应用产生式 *S*→*BbS*
2	# *SbB*	*baacbd*#	应用产生式 *B*→ε
3	# *Sb*	*baacbd*#	匹配栈顶和当前输入符号
4	# *S*	*aacbd*#	应用产生式 *S*→*AaS*
5	# *SaA*	*aacbd*#	应用产生式 *A*→*a*
6	# *Saa*	*aacbd*#	匹配栈顶和当前输入符号
7	#*Sa*	*acbd*#	匹配栈顶和当前输入符号
8	#*S*	*cbd*#	应用产生式 *S*→*BbS*
9	# *SbB*	*cbd*#	应用产生式 *B*→*c*
10	# *Sbc*	*cbd*#	匹配栈顶和当前输入符号
11	# *Sb*	*bd*#	匹配栈顶和当前输入符号
12	# *S*	*d*#	应用产生式 *S*→*d*
13	# *d*	*d*#	匹配栈顶和当前输入符号
14	#	#	返回，分析成功

10 通过变换求出与下列文法 $G[S]$ 等价的一个文法，使其不含直接左递归：

$$S \to AbB$$
$$A \to Aa \mid a$$
$$B \to Ba \mid Bb \mid b$$

参考解答：

关于A、B的产生式含有直接左递归。根据转换方法，替换结果如下：

$$S \to A\,b\,B$$
$$A \to a\,A'$$
$$A' \to a\,A' \mid \varepsilon$$
$$B \to b\,B'$$
$$B' \to a\,B' \mid b\,B' \mid \varepsilon$$

11 按照本讲介绍的消除一般左递归算法消除下面文法 $G[S]$中的左递归（要求依非终结符的排序 S、Q、P 执行该算法）：

$$S \to PQ \mid a$$
$$P \to QS \mid b$$
$$Q \to SP \mid c$$

参考解答：

按照非终结符的特定顺序排列各规则：

S →PQ | a
Q →SP | c
P →QS | b

第一步，得：

S →PQ | a
Q →PQP | aP | c
P →QS | b

第二步，得：

S →PQ | a
Q →PQP | aP | c
P →PQPS | aPS | cS | b

消去 P →PQPS 的左递归得：

S →PQ | a
Q →PQP | aP | c
P →aPSR | cSR | bR
R →QPSR | ε

经检查，此时得到的文法已经不含左递归，可结束消除左递归过程。

12 按照本讲介绍的消除一般左递归算法消除下面文法 G[*P*]中的左递归（要求依非终结符的不同排序分别执行该算法）：

$$P \rightarrow Qa \mid a$$
$$Q \rightarrow Pb \mid b$$

参考解答:

按照非终结符的P、Q的顺序排列规则，考虑关于Q的产生式，替换为

$$Q \rightarrow Qab \mid ab \mid b$$

再消去关于Q的产生式的直接左递归，得到

$$P \rightarrow Qa \mid a$$
$$Q \rightarrow abR \mid bR$$
$$R \rightarrow abR \mid \varepsilon$$

按照非终结符Q、P的顺序排列规则，对于P的产生式，替换为

$$P \rightarrow baB \mid aB$$
$$B \rightarrow baB \mid \varepsilon$$

该文法的语言应等价于正规式 (ba) (ba)* | a 所描述的语言。如果题目未要求一定按照本讲介绍的算法，则可给出任意的等价文法，其语言和这个正规式的语言一致。

13 变换下列文法， 求出与其等价的一个文法，使变换后的文法不含左递归和左公因子：

（1）文法 *G*[*P*]：

$$P \rightarrow Pa \mid Pb \mid c$$

（2）文法 *G*'[*S*]：

$$S \rightarrow a\,S\,A\,c \mid a$$
$$A \rightarrow A\,b \mid d$$

参考解答:

（1）消除左递归后，G［P］变换为等价的 G'[P］如下：

$$P \rightarrow \quad c\,Q$$
$$Q \rightarrow a\,Q \mid b\,Q$$

（2）提取左公共因子和消除左递归后，G［S］变换为等价的 G'[S］如下：

$$S \rightarrow a\,B$$
$$B \rightarrow S\,A\,e \mid \varepsilon$$

$$A \rightarrow d\,C$$
$$C \rightarrow b\,C \mid \varepsilon$$

（若发现问题，请及时告知）

1. 下面的文法 G[S] 描述由命题变量 p、q，联结词 ∧（合取）、∨（析取）、¬（否定）构成的命题公式集合：

$$S \to S \vee T \mid T$$
$$T \to T \wedge F \mid F$$
$$F \to \neg F \mid p \mid q$$

试分别指出句型 ¬F ∨ ¬ q ∧ p 和 ¬F ∨ p ∧ ¬F ∨ T 的所有短语，直接短语。如果这些句型同时也是右句型，那么还要给出其句柄 . 请将结果填入下表中：

句型	短语	直接短语	句柄
¬F∨¬q∧p			
¬F∨p∧¬F∨T			

参考解答：

句型	短语	直接短语	句柄
¬F∨¬q∧p	¬F∨¬q∧p　¬F　¬q∧p ¬q　q　p	¬F　q　p	¬F
¬F∨p∧¬F∨T	¬F∨p∧¬F∨T　¬F∨p∧¬F ¬F　p∧¬F　p　¬F	¬F　p　¬F	无

2. （1）给定文法 *G*[S]：

$$S \to A\,B$$
$$A \to a\,A \mid \varepsilon$$
$$B \to B\,b \mid \varepsilon$$

（*a*）构造该文法 *G*[S] 的 LR（0）有限状态机。

（*b*）说明该文法不是 LR(0) 文法。

（*c*）该文法是否 SLR(1) 文法？为什么？

（2）给定文法 *G*[S]：

$$S \to S\,S \mid (\,S\,) \mid a$$

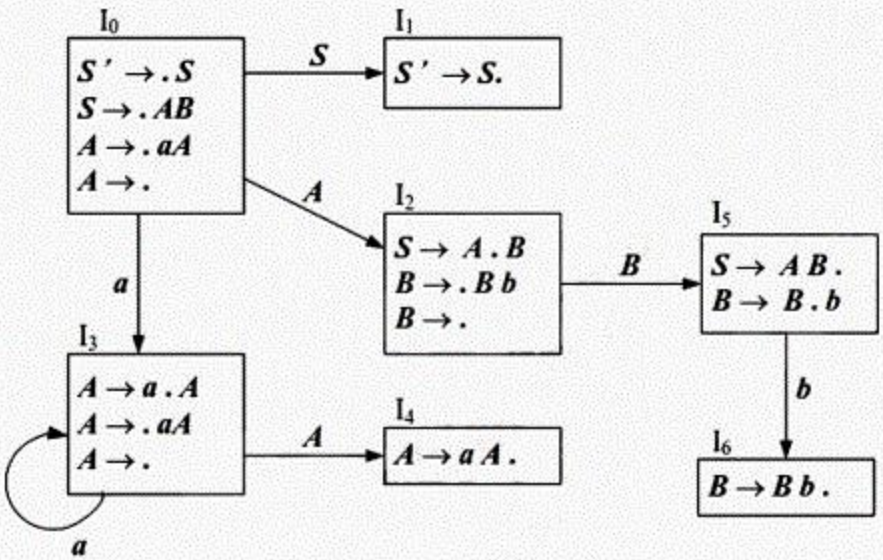
完成同（1）一样三个问题（*a*），（*b*），（*c*）。

参考解答:

（1）首先变换文法为增广文法：

$$S' \rightarrow S$$
$$S \rightarrow AB$$
$$A \rightarrow aA \mid \varepsilon$$
$$B \rightarrow Bb \mid \varepsilon$$

（a）该文法的LR（0）有限状态机状态转换图如下：



（b）状态I0、I3、I5有移进归-约冲突，所以不是LR(0)文法。

（c）对于状态I0、I3，由于FOLLOW(A)={b，#}，在面临第一个符号是#或b时，选择归约成A。面临符号a，则移进。

对于状态I5，由于FOLLOW(S)={#}，在面临第一个符号是#时，选择归约成S。面临符号b，则移进。

所以，该文法是SLR(1)文法。

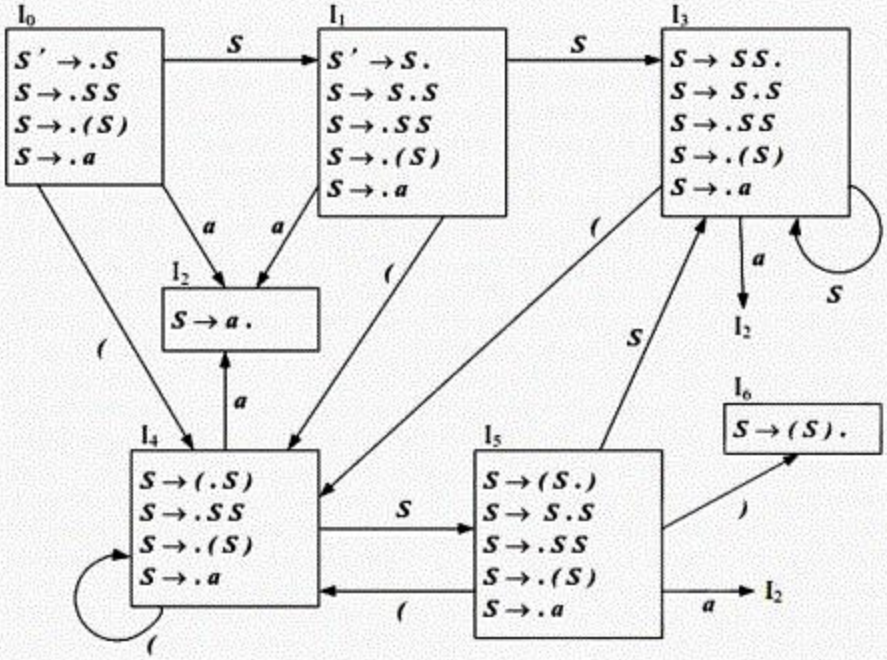
（2）首先变换文法为增广文法。增加如下产生式

$$S' \rightarrow S$$

得到增广文法如下

$$S' \rightarrow S$$
$$S \rightarrow SS$$
$$S \rightarrow (S)$$
$$S \rightarrow a$$

（a）该文法的LR（0）有限状态机状态转换图如下：



（b）状态I3中包含移进-归约冲突，所以G(S)不是LR(0)文法。

（c）对于状态I3，由于FOLLOW(S)={a，(，)，#}，在面临第一个符号是a时，不能选择是归约成S还是移进a，所以，不是SLR(1)文法。

3. 试构造下列文法的 LR（0）FSM，并判别它们是否 LR(0)或 SLR(1)文法:

a) 文法 G[E]:

$$E \rightarrow E + T \mid T$$
$$T \rightarrow (E) \mid id \mid id [E]$$

其中 E,T 为非终结符,其余符号其余符号 id，(，)，[，] 为终结符

b) 文法 G[S]:

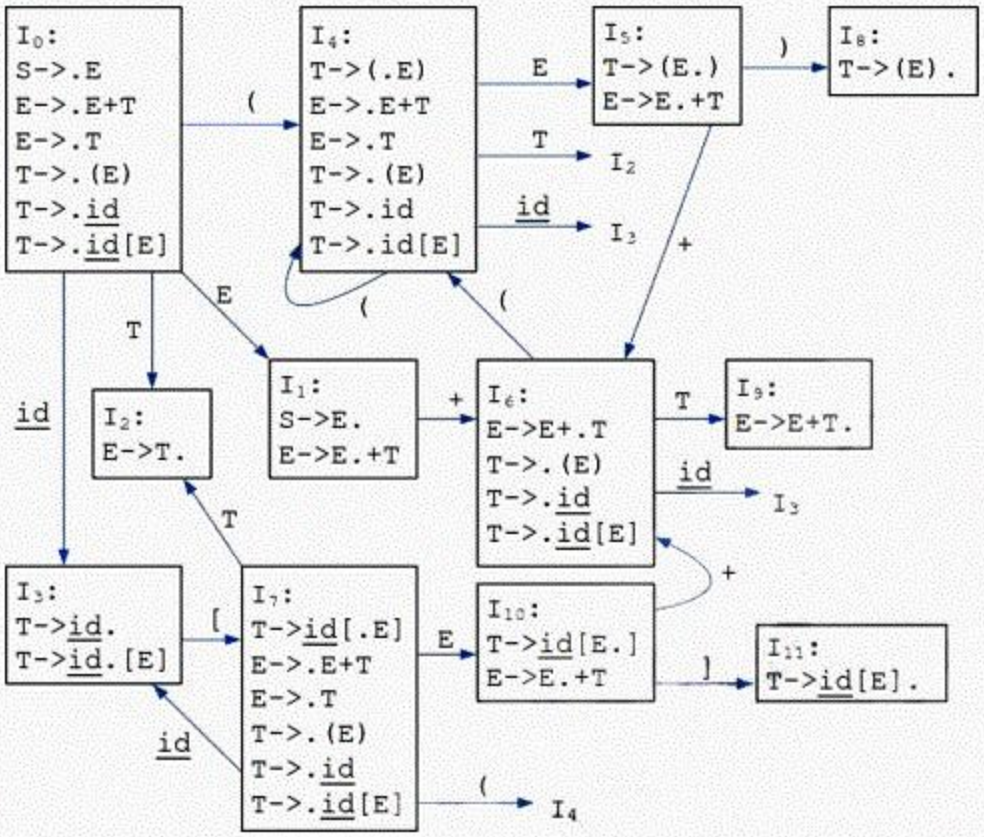
$$S \rightarrow Ab \mid ABc$$
$$A \rightarrow aA \mid a$$
$$B \rightarrow b$$

其中 S,A,B 为非终结符,其余符号为终结符

参考解答:

a) 增加产生式 S→E，得增广文法 G′[S]

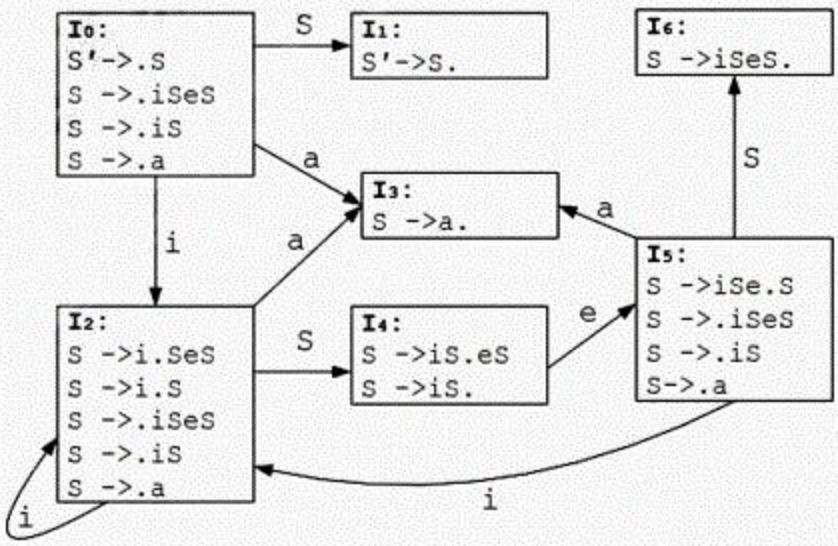
构造识别活前缀的 LR(0)FSM 如下：



可验证：状态 I_3 有移进-归约冲突，所以 G[E]不是 LR(0)文法；进一步，因 Follow(T) = { + ,) ,] , # }，不含[，所以 G[E]是 SLR(1)文法。

b) 增加产生式 S′→S，得增广文法 G′[S′]

构造识别活前缀的 LR(0)FSM 如下：



I4 存在归约/归约冲突， I3 存在归约/移进冲突.因此不是 LR(0)文法。

考察能否使用SLR(1)方法解决冲突：I4 中，因为 Follow(S) = {#} 而 Follow(B) = {c}. 所以可以解决。 I3中，因Follow(A) = {b}，不含 a，因此该移进/归约冲突也可解决. 文法是

SLR(1)文法

4. 给定 SLR(1) 文法 G[S]：

（1）$S \rightarrow A\ b$

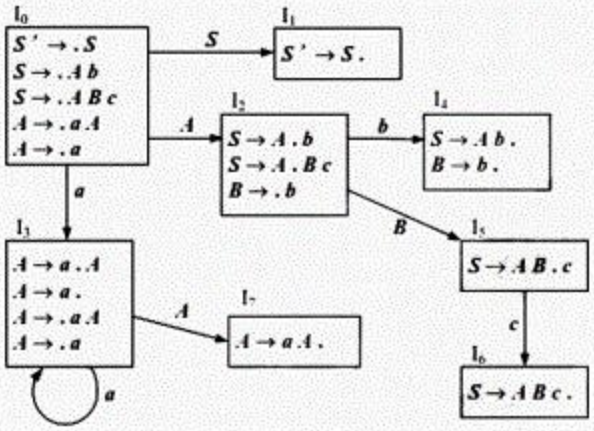
（2）$S \rightarrow A\ B\ c$

（3）$A \rightarrow a\ A$

（4）$A \rightarrow a$

（5）$B \rightarrow b$

其 LR(0) 有限状态机如下图所示：



（*a*）构造该文法的 SLR(1) 分析表。

（*b*）若采用 SLR(1) 方法对于 *L*(*G*) 中的某一输入串进行分析，当栈顶出现句柄 *a* 时，余留输入符号串中的第一个符号是什么？

参考解答：

（*a*）SLR(1) 分析表：

状态	ACTION				GOTO		
	a	b	c	#	S	A	B
0	s_3				1	2	
1				acc			
2		s_4					5
3	s_3	r_4				7	
4			r_5	r_1			
5			s_6				
6				r_2			
7		r_3					

（*b*） *b*

5. 给定 SLR(1) 文法 G[S]：

（1）$S \to aSa$

（2）$S \to bSb$

（3）$S \to c$

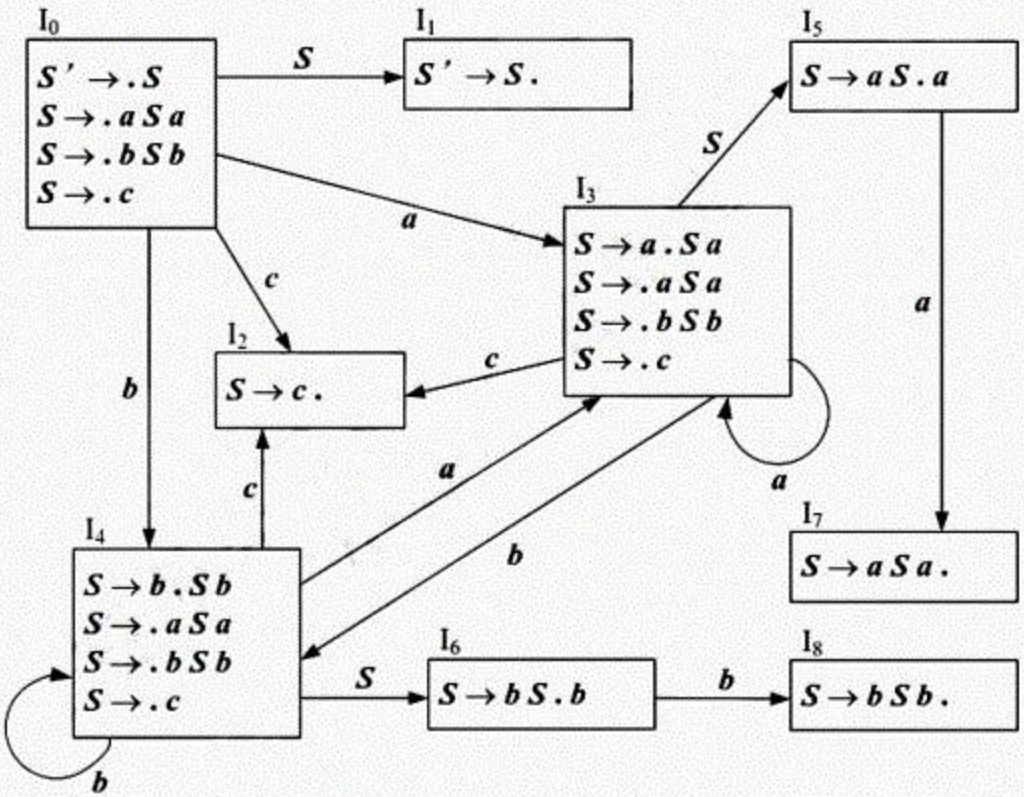
（a）构造该文法的 LR(0) 有限状态机

（b）构造该文法的 SLR(1) 分析表。

（c）若根据以上 SLR(1)分析表对于 $L(G)$ 中的某一输入串执行 SLR(1) 分析过程，初始时符号栈存放符号 #。当扫描过串 *abbcb* 后，分析栈中的符号串是什么（以进栈先后次序给出）？当前可规约的句柄是什么？

参考解答:

（a）



（b）

状态	ACTION				GOTO
	a	b	c	#	S
0	s_3	s_4	s_2		1
1				acc	
2	r_3	r_3		r_3	
3	s_3	s_4	s_2		5
4	s_3	s_4	s_2		6
5	s_7				
6		s_8			
7	r_1	r_1		r_1	
8	r_2	r_2		r_2	

（*c*）当输入扫描过串 *abbcb* 后，分析栈中的符号串是什么 #abbSb （或 abbSb）。当前可规约的句柄是 bSb。

6. 给定文法 *G*[S]:

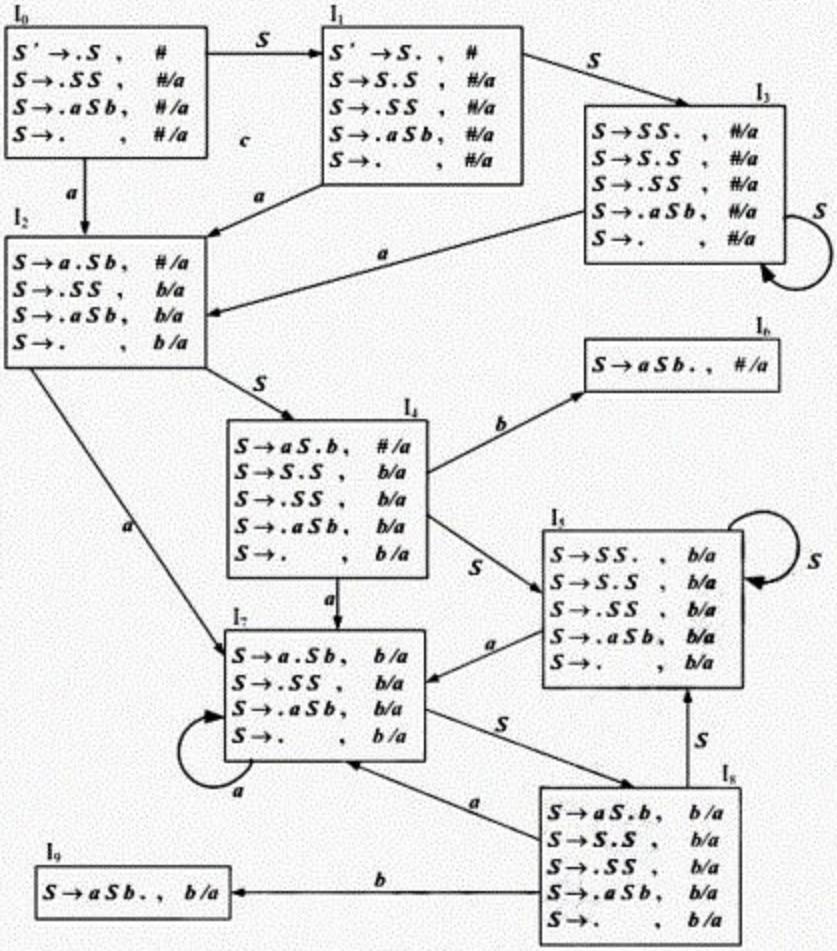
$$S \rightarrow SS \mid aSb \mid \varepsilon$$

（*a*）构造该文法的 LR（1）有限状态机。

（*b*）该文法是否 LR(1) 文法？为什么？

参考解答:

（a） 完整的 LR(1) 自动机：



（b）该文法不是 LR(1) 文法。存在“移进/归约冲突”的状态：I_0, I_2, I_2, I_3, I_4, I_5, I_7, I_8

存在“归约/归约冲突”的状态：I_3, I_5

7. 对于下列文法 $G(S)$：

$$S \to Aa \mid cAb \mid Bb \mid cBa$$
$$A \to d$$
$$B \to d$$

试验证：该文法是一个 LR（1）文法，但不是 LALR（1） 文法。

参考解答:

本题考察 LR(1)有限状态机的构造。LR(1) 有限状态机比 SLR(1) 有限状态机有更多的状态，所以，LR(1)分析法比 SLR(1)分析法有着更强的解决冲突的能力。对某些文法的 LR(1)有限状态机，用合并同心集方法还可以构造出和 SLR(1)状态相同的 LALR(1) 有限状态机。LR(1)解决冲突能力强的原因是用向前搜索符代替了 SLR(1)所用的非终结符的后跟符。

根据 LR(1)有限状态机构造步骤，

（1） 对文法 G(S)增加产生式

$$S' \to S$$