第6章　LR分析

SLR(1) 分析

real a,b,…

例：(0)S'→S　　(1)S→rD

　　(2)D→D,i　　(3)D→i

LR(0)项目

1. S'→.S　　2. S'→S.　　3. S→.rD

4. S→r.D　　5. S→rD.　　6. D→.D,i

7. D→D.,i　　8. D→D,.i　　9. D→D,i.

10. D→.i　　11. D→i.

例：(0)S'→S　　(1)S→rD

　　(2)D→D,i　　(3)D→i

LR(0)项目集规范族

I_0: S'→.S
　　S→.r D

I_1: S'→S.

I_2: S→r.D
　　D→.D,i
　　D→.i

I_3: S→r D.
　　D→D.,i

I_4: D→i.

I_5: D→D,.i

I_6: D→D,i.

其中I_3中含有移进/归约冲突!

那么文法就不是LR(0)的，如何解决？

I_3: S→r D. D→D.,i

例:文法的LR(0)分析表有多重表项

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1				acc		
2			S_4			3
3	r_1	S_5, r_1	r_1	r_1		
4	r_3	r_3	r_3	r_3		
5			S_6			
6	r_2	r_2	r_2	r_2		

I_3: S→r D. D→D.,i

利用 FOLLOW(S) ∩ {, }=ϕ 解决冲突

SLR(1)分析表

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1				acc		
2			S_4			3
3	r_1	S_5	r_1	r_1		
4	r_3	r_3	r_3	r_3		
5			S_6			
6	r_2	r_2	r_2	r_2		

Forward

SLR(1)技术

问题描述：

如果 LR(0) 项目集规范族中某个项目集I_K含移进/归约，归约/归约 冲突：

I_K：{…, $A \rightarrow \alpha.b\beta$, $P \rightarrow \omega.$, $Q \rightarrow \gamma.$, …}

求解条件：

若FOLLOW(Q) ∩ FOLLOW(P) =ϕ

FOLLOW(P) ∩ { b } =ϕ

FOLLOW(Q) ∩ { b} =ϕ

解决冲突的SLR(1)技术：

action [k, b] = 移进；

对a ∈FOLLOW (P) 则action[k, a] =用 $P \rightarrow \omega$ 归约；

对a ∈FOLLOW (Q) 则action[k, a] =用 $Q \rightarrow \gamma$ 归约；

能用SLR(1)技术解决冲突的文法称为SLR(1)文法，SLR(1)文法无二义

例：表达式文法G的拓广文法

(0) S′→E

(1) E→E+T

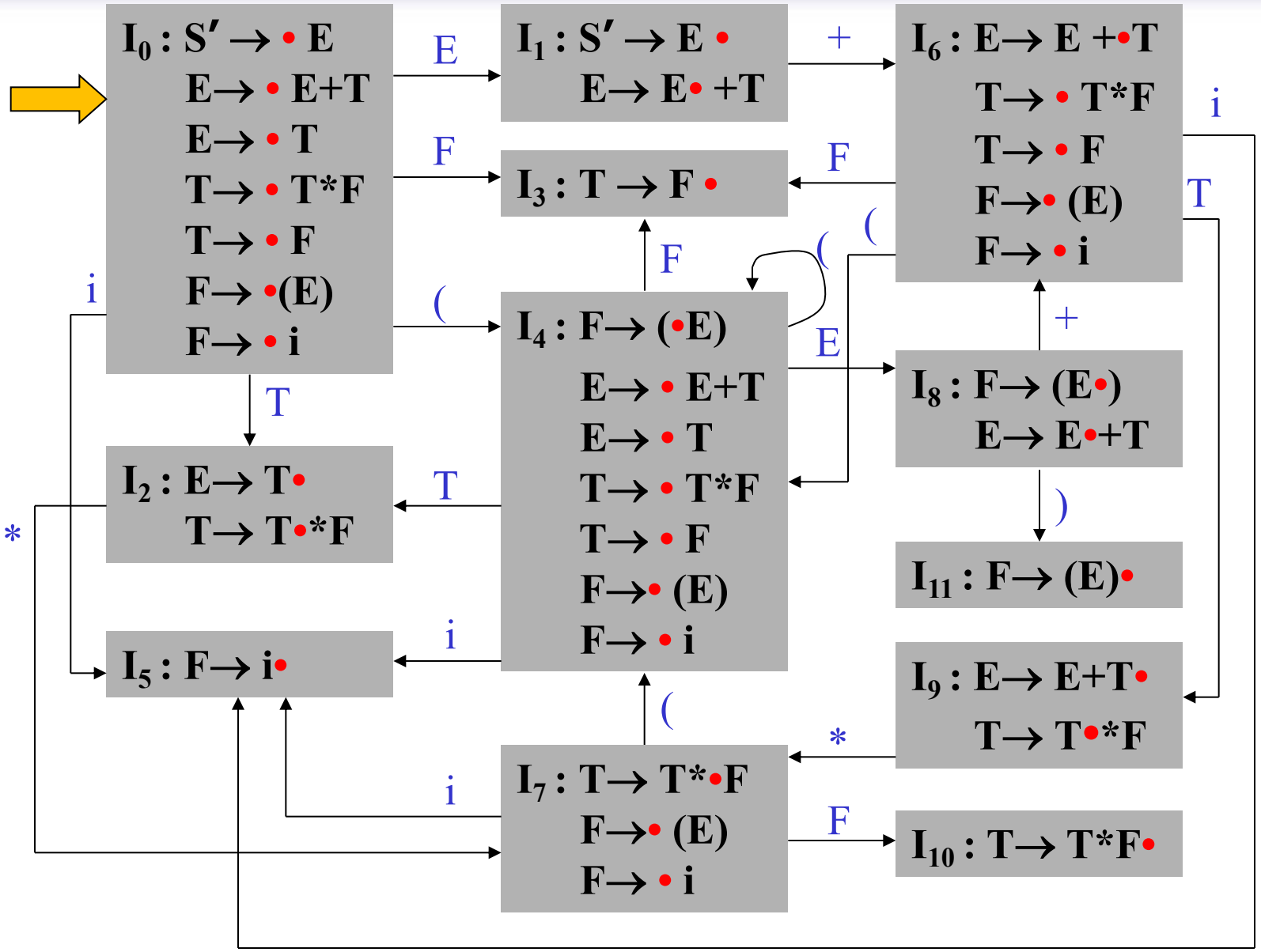
(2) E→T

(3) T→T*F

(4) T→F

(5) F→(E)

(6) F→i


I0 : S′ → • E
E→ • E+T
E→ • T
T→ • T*F
T→ • F
F→ •(E)
F→ • i
I1 : S′ → E •
E→ E• +T
I2 : E→ T•
T→ T•*F
I3 : T → F •
I4 : F→ (•E)
E→ • E+T
E→ • T
T→ • T*F
T→ • F
F→• (E)
F→ • i
I5 : F→ i•
I6 : E→ E +•T
T→ • T*F
T→ • F
F→• (E)
F→ • i
I7 : T→ T*•F
F→• (E)
F→ • i
I8 : F→ (E•)
E→ E•+T
I9 : E→ E+T•
T→ T•*F
I10 : T→ T*F•
I11 : F→ (E)•
E
F
T
i
(
+
*
)

Q1：I_1, I_2, I_9中存在冲突！

I_1：S’ → E •，E→ E• +T

I_2：E→ T•，T→ T•*F

I_9：E→ E+T•，T→ T•*F

可用SLR(1)方法解决！

Q2：无用归约！

在状态I_3中，只有一个项目$T \rightarrow F \bullet$。

按SLR(1)方法，该项目中没有冲突，可保持原来LR(0)的处理方法：**不论当前输入符号是什么都进行归约；**

若当前输入符为“(”，进行归约是多余的（输入串不合法！）；

延迟了错误的发现。

Idea：**对所有归约项目，检查当前输入符号是否属于归约产生式左部非终结符的FOLLOW集：是则进行归约；否则报错。**

SLR(1)表的构造（改进）

假定C={I_0, I_1, …, I_n}，令每个项目集I_k的下标k为分析器的一个状态，则G'的SLR分析表含有状态0，1，…，n。

令那个含有项目S'→.S的I_k的下标k为初态。

ACTION表和GOTO表可按如下方法构造：

若项目A→α.aβ属于I_k且GO(I_k, a)=I_j, a为任意终结符，则置ACTION[k, a]为“把状态j和符号a移进栈”，简记为“sj”；

若项目A→α.属于I_k，那么对任何输入符号a，若a∈FOLLOW(A)，则置ACTION[k, a]为“用产生式A→α进行归约”，简记为“rj”；其中A→α为文法的第j个产生式；

若项目S'→S.属于I_k, 则置ACTION[k, #]为“接受”，简记为“acc”；

若GO (I_k, A)= I_j，A为非终结符，则置GOTO(k, A)=j；

分析表中凡不能用规则1至4填入的空白项均置“出错标志”。

- 按上述算法构造的含有ACTION和GOTO两部分的分析表，如果每个表项不含多重定义，则称它为文法G的一张SLR(1)分析表。

- 具有SLR(1)表的文法G称为一个SLR(1)文法。

例: 文法(0) S'→S　　(1)S→rD

(2) D→D,i　　(3)D→i

SLR(1)分析表

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S_2				1	
1			acc			
2			S_4			3
3		S_5		r_1		
4	r_3		r_3			
5		S_6				
6	r_2		r_2			

Backward

LR(1)分析

SLR(1)的局限

文法G:

(0) S'→S　　(1) S→aAd　　(2) S→bAc

(3) S→aec　　(4) S→bed　　(5) A→e

LR(0) 项目集规范族：

I_0: S'→.S
S →.aAd
S →.bAc
S →.aec
S →.bed

I_1: S'→S.

I_2: S→a.Ad
S→a.ec
A→.e

(0) S`→S (1) S→aAd (2) S→bAc
(3) S→aec (4) S→bed (5) A→e

I_3: S→b.Ac
S→b.ed
A→.e

I_4:
S→aA.d

I_5:
S→ae.c
A→e.

I_6:
S→bA.c

I_7:
S→be.d
A→e.

I_8:
S→aAd.

I_9: S→aec.

I_{10}: S→bAc.

I_{11}: S→bed.

(0) S`→S (1) S→aAd (2) S→bAc
(3) S→aec (4) S→bed (5) A→e

	ACTION						GOTO	
	a	c	e	b	d	#	S	A
0	S2			S3			1	
1						acc		
2			S5					4
3			S7					6
4					S8			
5	r5	**r5S9**	r5	r5	r5	r5		
6		S10						
7	r5	r5	r5	r5	**r5S11**	r5		
8	r1	r1	r1	r1	r1	r1		
9	r3	r3	r3	r3	r3	r3		
10	r2	r2	r2	r2	r2	r2		
11	r4	r4	r4	r4	r4	r4		

Follow(A)={c, d}

(0) S'→S　(1) S→aAd　(2) S→bAc

(3) S→aec　(4) S→bed　(5) A→e

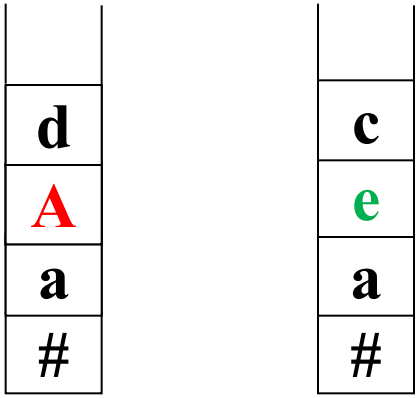
在一个规范推导中，哪些输入符号能跟在句柄之后?

I_5: S →ae.c

A →e.

$S' \Rightarrow_R S \Rightarrow_R aAd \Rightarrow_R aed$

$S' \Rightarrow_R S \Rightarrow_R aec$



w_1=a e d　w_2=a e c

并不是Follow(A)中的每个元素在包含A的每个句型中都会在A之后出现。

G[S]:

若$S \Rightarrow_R^* \alpha Aw \Rightarrow \alpha\beta w$，r是αβ的前缀，则称r是G的一个活前缀。

Forward

(0) S'→S　(1) S→aAd　(2) S→bAc

(3) S→aec　(4) S→bed　(5) A→e

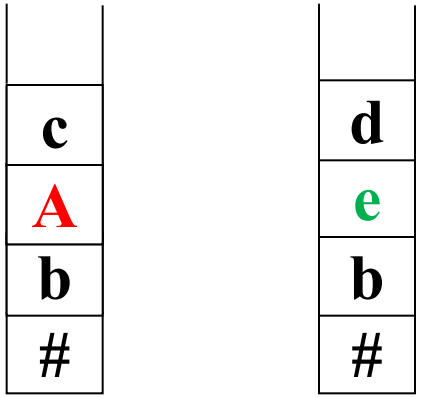
在一个规范推导中，哪些输入符号能跟在句柄之后?

I_7: S →be.d

A →e.

S' $\Rightarrow_R$ S $\Rightarrow_R$ bAc $\Rightarrow_R$ bec

S' $\Rightarrow_R$ S $\Rightarrow_R$ bed



w_1=b e c　　w_2=b e d

并不是Follow(A)中的每个元素在包含A的所有句型中都会在A之后出现。

G[S]:

若S $\Rightarrow_R^*$ αAw $\Rightarrow$ αβw，r是αβ的前缀，则称r是G的一个活前缀。

Forward

G[S]:(0) S'→ S (1) S→L=R (2) S→R
(3) L→ *R (4) L→id (5) R→L

LR(0)项目集规范族

I0: S' –> •S
S –> •L = R
S –> •R
L –> •*R
L –> •id
R –> •L

I1: S' –> S•

I2: S –> L• = R
R –> L•

I3: S –> R•

I4: L –> *•R
R –> •L
L –> •*R
L –> •id

I5: L –> id•

I6: S –> L =•R
R –> •L
L –> •*R
L –> •id

I7: L –> *R•

I8: R –> L•

I9: S –> L=R•

Another example

I2: S –> L• = R，R –> L•

考虑分析表达式 id = id时，在工作到 I2 处已经将第一个 id 归约成 L了, 看到下一个输入 = 要作决策，第一个项目设置 Action[2, =] 为S6, 即把赋值的其它部分找到. 但 = 也是属于 Follow(R), 第二个项目要用 R–>L归约. 出现 shift-reduce 冲突.

SLR 分析程序没有记住足够的左文以决定当见到一个能归约到L的串而在输入中碰上 = 时会发生什么. 虽然在栈顶的符号序列可以归约到 R, 但我们不能要这个选择, 因为不可能有规范句型以 R = …开头 (有以 *R = ... 开头的规范句型).

SLR(1)的局限

FOLLOW 集包含了在任何句型中跟在 R 后的符号, 但没有严格地指出在一个特定的推导里哪些符号跟在 R后. 所以需要扩充状态以包含更多的信息: FOLLOW 集的哪些部分才是进到该状态最恰当的归约依据.

处在状态 2时，试图构建句子有两条路:

1.S $\Rightarrow$L = R 或 2.S $\Rightarrow$ R $\rightarrow$ L. 如下一符号是 =, 那就不能用第二个选择, 必须用第一选择, 即移进. 只有下一个符号是#时才能归约. 尽管 = 属于 Follow(R), 但那是因为一个 R 可以出现在别的上下文中. 在当前这个特定情况下, 它不适合, 因为在用S $\Rightarrow$ R$\rightarrow$L推导句子时, = 不能跟在R后.

是否LR(0)文法？

∵ I_2: S→L.=R, R→L.中存在移进/归约冲突

不是LR(0)文法

SLR能否解决I_2中的冲突?

∵ FOLLOW(R)={#,=}与{=}交不为空

不是SLR(1)文法

LR(1)方法

若 $A \rightarrow \alpha . B \beta \in I$

则 $B \rightarrow . \gamma \in I$（$B \rightarrow \gamma$ 是一产生式）

Idea: 考虑将FIRST(β)作为用产生式B→γ归约的搜索符，作为归约时查看的符号集，用以代替SLR(1)分析中的FOLLOW集。

Backward

LR(1)项目(配置)的一般形式

[A →α•β, a]

意味着处在栈顶的为α，期望β出现在栈顶的状态；a 称作该项目(配置)的向前搜索符(lookahead)。

[A –> α β •, a]

意味着处在栈中的是αβ的状态，但只有当下一个输入符是a时才能进行归约；a 要么是一个终结符，要么是输入结束标记#。

有多个向前搜索符时，可写作 **A –> u•, a/b/c**

向前搜索符(lookahead)只对圆点在最后的项目起作用

LR(1)项目集规范族构造

Closure(I)按如下方式构造：

(1)**I** 的任何项目属于**closure(I)**;

(2)**若[A→β$_1$. Bβ$_2$, a]∈closure(I)，B→δ是一产生式，那么对于FIRST(β$_2$ a)中的每个终结符b，如果[B→.δ, b]不在closure(I)中, 则将它加入；**

(3)重复(1)(2), 直至**closure(I)**不再增大。

LR(1)项目集规范族C的构造算法类似于LR(0)：

在开始时，**C={closure({[S`→.S, #]})};**

GO函数：

若I是一个项目集, X是一个文法符号,

GO(I, X)= closure(J)

其中 **J={**任何形如**[A→αX.β, a]**的项目**|[A→α.Xβ,a]∈I}**。

例：LR(1)项目集的规范族

I_0: S'→.S, #
S →.aAd, #
S →.bAc, #
S →.aec, #
S →.bed, #

I_1: S'→S., #

I_2: S'→a.Ad, #
S →a.ec, #
A →.e, d

I_3: S →b.Ac, #
S →b.ed, #
A→.e,c

I_4: S →aA.d, #

I_5: S →ae.c, #
A →e., d

I_6: S →bA.c, #

I_7: S →be.d, #
A →e., c

I_8: S →aAd., #

I_9: S →aec., #

I_{10}: S →bAc., #

I_{11}: S →bed., #

(0) S'→S
(1) S→aAd
(2) S→bAc
(3) S→aec
(4) S→bed
(5) A→e

规范的LR(1)分析表构造

假定LR(1)项目集规范族C={I_0, I_1, …, I_n}, 令每个项目集I_k的下标k为分析器的一个状态，G'的LR(1)分析表含有状态0, 1, …, n.

1. 若项目**[A→α.aβ, b]**属于I_k且GO(I_k, a)= I_j, 则置**ACTION[k, a] = sj**, 含义是“把状态j和符号a移进栈”；
2. 若项目**[A→α., a]**属于I_k, 那么置**ACTION[k, a]**为“用产生式A→α进行归约”, 简记为“**rj**”；其中, j为产生式A→α在文法G'中的编号；
3. 若项目**[S`→S., #]**属于I_k, 则置**ACTION[k, #]**为“接受”, 简记为“**acc**”；
4. 若**GO(I_k, A)= I_j**, A为非终结符, 则置**GOTO(k, A)=j**；
5. 分析表中凡不能用规则1至4填入信息的**空白表项**，均置上“**出错标志**”。

- 按上述算法构造的含有ACTION和GOTO两部分的分析表，如果每个表项不含多重定义，则称它为文法G的一张规范的LR(1)分析表。

- 具有规范的LR(1)分析表的文法G称为一个LR(1)文法。

LR(1) 文法满足下面两个条件:

1. 如果一个项目集中有项目 **[A –> u•xv, a]**，x是终结符，则不会有项目**[B –> u•, x]**；

2. 项目集中所有归约项目的向前搜索符不相交，即不能同时含有项目：**[A –> u•, a]** 和 **[B –> v•, a]**

LR(1)比SLR(1)能力强

例如文法G[S']：

(0)S'→S

(1)S→L=R

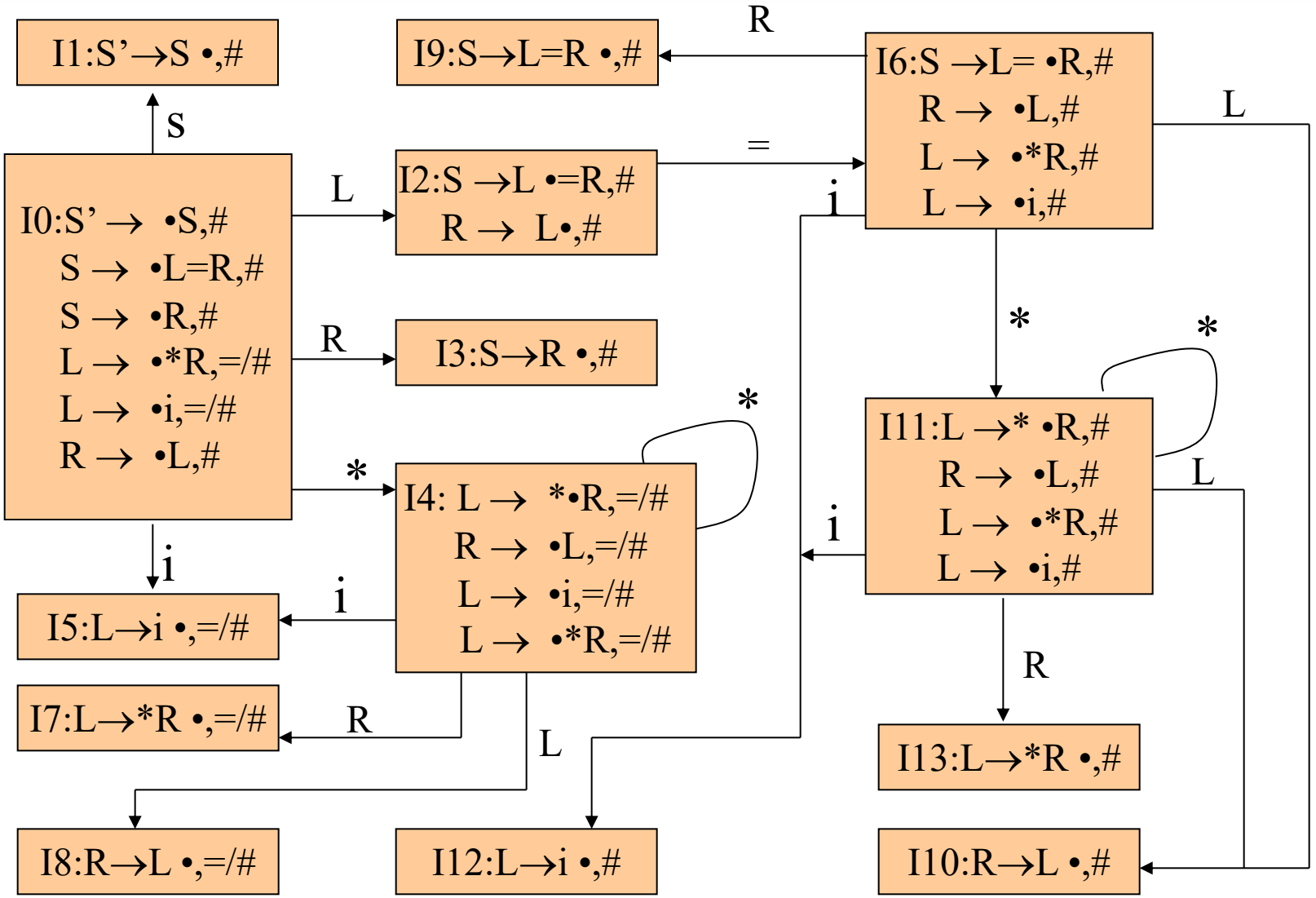
(2)S→R

(3)L→*R

(4)L→i

(5)R→L

不能用SLR(1)技术解决，
但能用LR(1)技术解决。


I1:S'→S •,#
I9:S→L=R •,#
R
I6:S →L= •R,#
R → •L,#
L → •*R,#
L → •i,#
L
S
=
I0:S' → •S,#
S → •L=R,#
S → •R,#
L → •*R,=/#
L → •i,=/#
R → •L,#
L
I2:S →L •=R,#
R → L•,#
i
*
*
R
I3:S→R •,#
*
I4: L → *•R,=/#
R → •L,=/#
L → •i,=/#
L → •*R,=/#
I11:L →* •R,#
R → •L,#
L → •*R,#
L → •i,#
L
i
i
i
I5:L→i •,=/#
I7:L→*R •,=/#
R
L
R
I13:L→*R •,#
I8:R→L •,=/#
I12:L→i •,#
I10:R→L •,#

每个SLR(1)文法都是LR(1)的，一个SLR(1)文法的LR(1)分析器往往比其SLR(1)分析器的状态数要多。

例G(S):S → BB　　B→aB　　B →b

I_0:S'→.S　S →.BB　B →.aB　B →.b

I_1:S'→S.

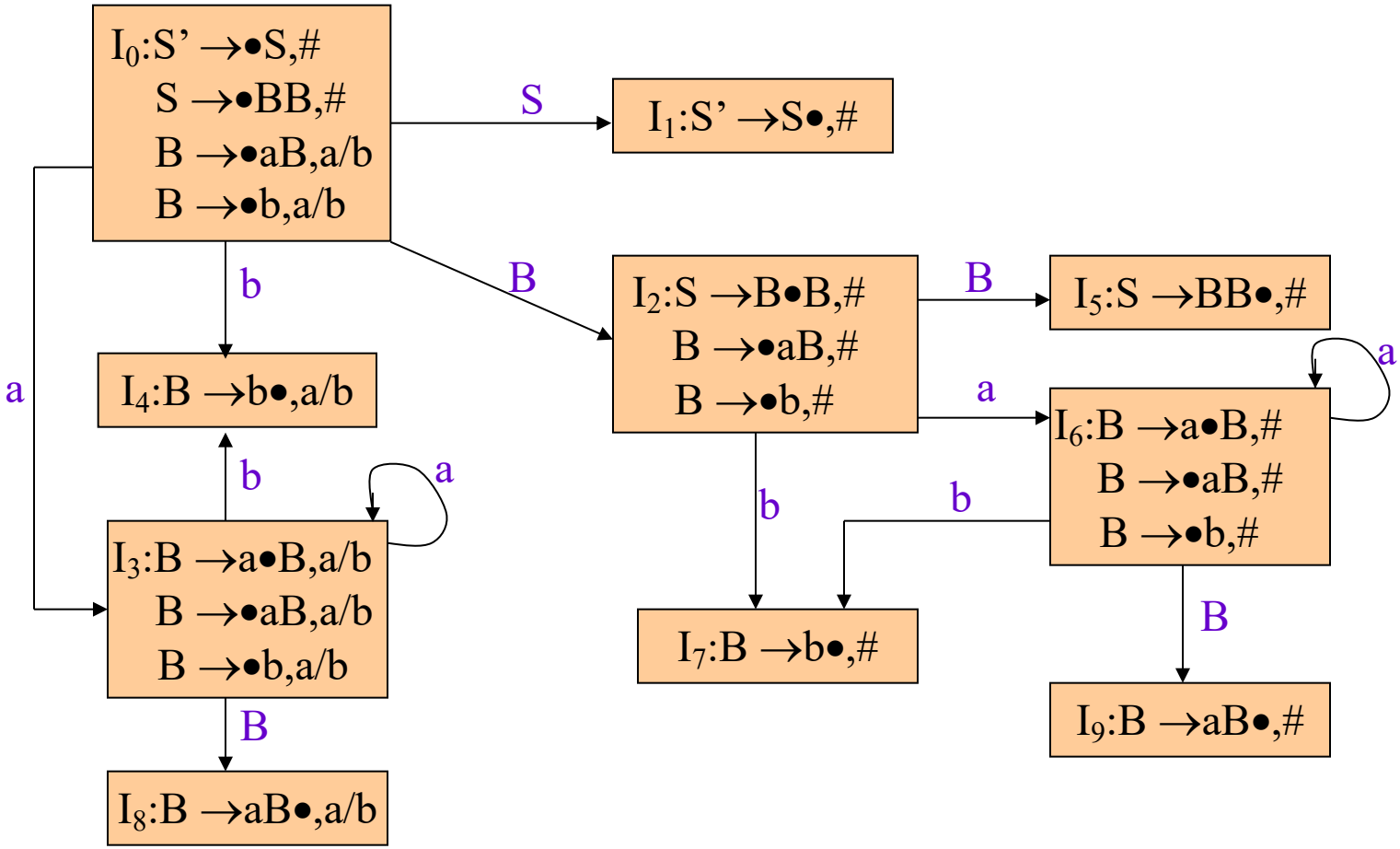
I_2:S →B.B　B→.aB　B→.b

I_3:B →a.B　B→.aB　B→.b

I_4:B→aB.

I_5:B→b.

I_6:S→BB.



LR(1)项目集及其之间转换关系

LALR(1) 分析

LALR—SLR(1)和LR(1)之间的折衷办法（状态数目、分析能力）

LALR (*lookahead LR*)

I_3:B →a•B,a/b
B →•aB,a/b
B →•b,a/b

I_4:B →b•,a/b

I_8:B →aB•,a/b

I_7:B →b•,#

I_9:B →aB•,#

I_6:S →a•B,#
B →•aB,#
B →•b,#

合并同心集I_3I_6，I_4I_7，I_8I_9。

同心集合并的几个问题：

1. 同心集合并后所得项目集的心不变，但搜索符为各同心集的搜索符集的并。
2. 合并同心集后的项目集经转换函数到达的仍为同心集。
3. 合并同心集后若有冲突也只可能是归约/归约冲突，而不可能产生移进/归约冲突。
4. 合并同心集后对某些错误发现时间会产生推迟现象。

合并同心集后若有冲突也只可能是归约/归约冲突，而不可能产生移进/归约冲突。

若LR(1)文法的项目集I_k与I_j为同心集，其中：

I_k：[A →α., u_1]

[B →β.aγ, b]

I_j：[A →α., u_2]

[B →β.aγ, c]

合并后I_{kj}：

[A →α., u_1/u_2]

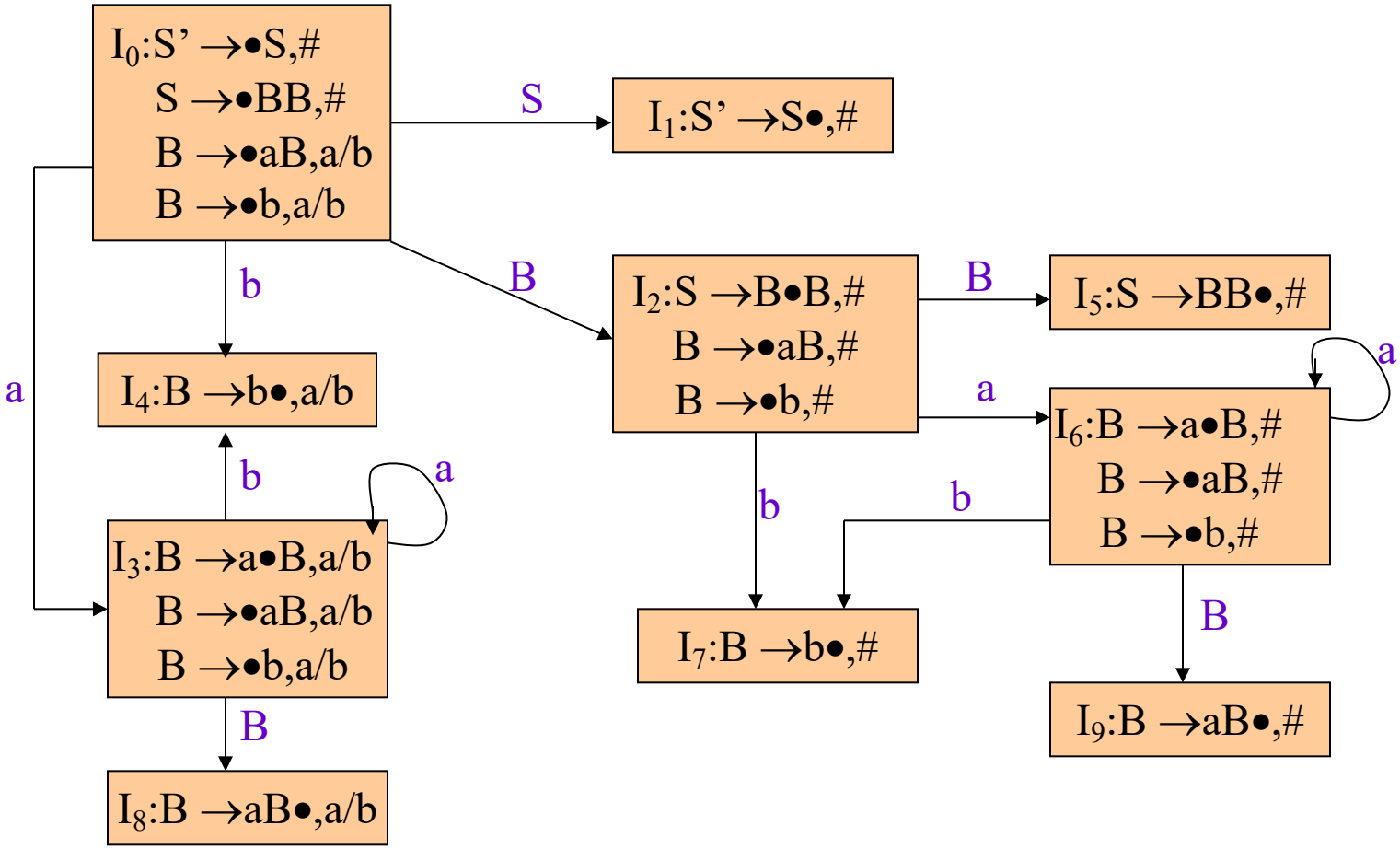
[B →β.aγ, b/c]

构造 LALR(1)分析表

1. 构造文法G的规范 LR(1) 状态集.
2. 合并同心集（除搜索符外两个集合是相同的）的状态.
3. 新 LALR(1) 状态的GO函数是合并的同心集状态的GO函数的并.
4. LALR(1)分析表的ACTION 和 GOTO 构造方法与LR(1) 一样.

经上述步骤构造的表若不存在冲突，则称它为G的**LALR(1)分析表**。

存在这种分析表的文法称为**LALR(1)文法**。



LR(1)项目集及其之间转换关系

LALR(1)分析表

LR(1)分析表

状态	ACTION			GOTO	
	a	b	#	S	B
0	S3	S4		1	2
1			acc		
2	S6	S7			5
3	S3	S4			8
4	r3	r3			
5			r1		
6	S6	S7			9
7			r3		
8	r2	r2			
9			r2		

分析过程1

状态	ACTION			GOTO	
	a	b	#	S	B
0	$S_{3,6}$	$S_{4,7}$		1	2
1			acc		
2	$S_{3,6}$	$S_{4,7}$			5
3,6	$S_{3,6}$	$S_{4,7}$			8,9
4,7	r_3	r_3	r_3		
5			r_1		
8,9	r_2	r_2	r_2		

对输入串ab#用LR(1)分析的过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	ab#	S_3	
2	03	#a	b#	S_4	
3	034	#ab	#	出错	

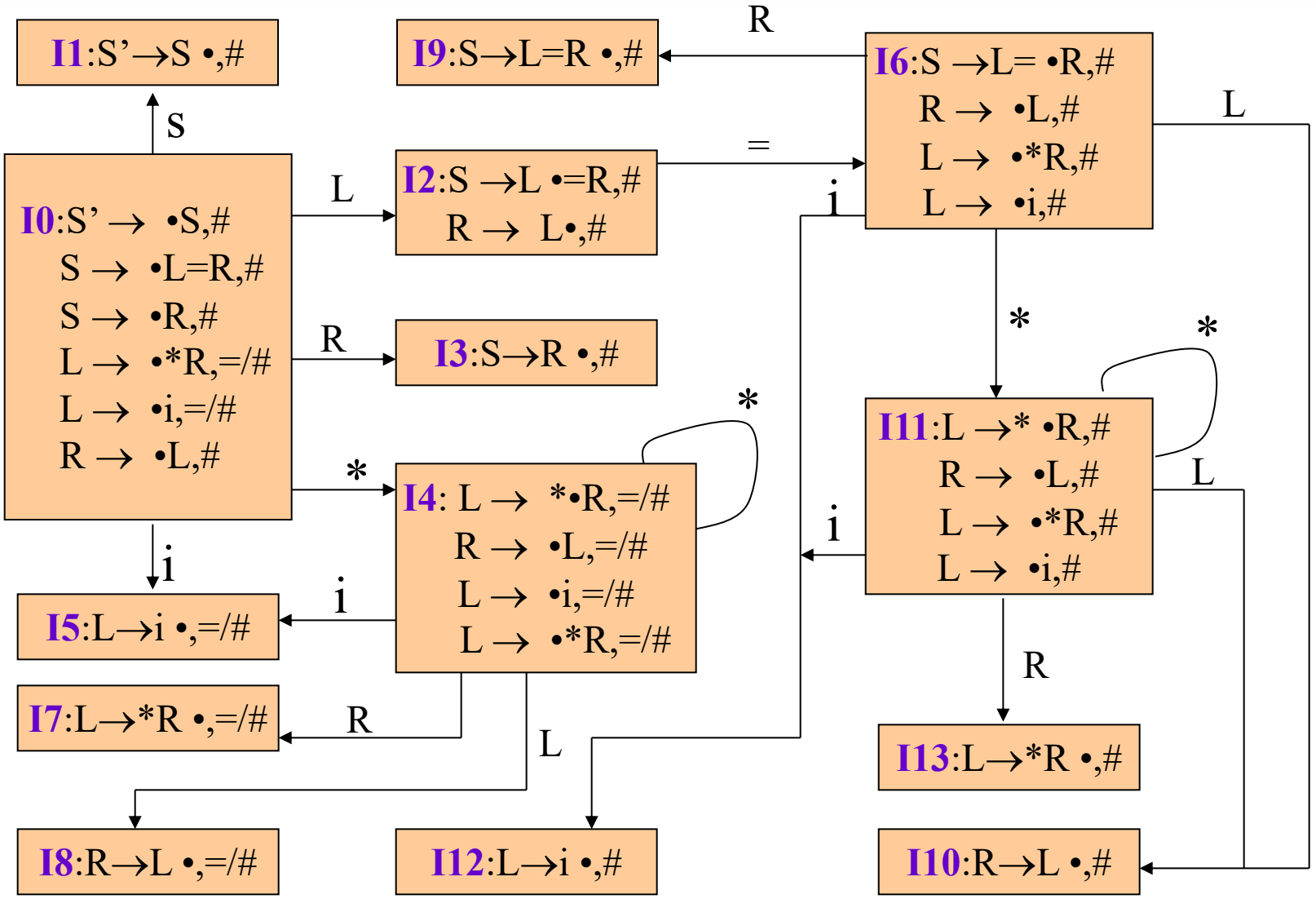
分析表

对输入串ab#用LALR(1)分析的过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	ab#	$S_{3,6}$	
2	0(3,6)	#a	b#	$S_{4,7}$	
3	0(3,6)(4,7)	#ab	#	r_3	(8,9)
4	0(3,6)(8,9)	#aB	#	r_2	2
5	02	#B	#	出错	

分析表

例：(0)S'→S (1)S→L=R (2)S→R (3)L→*R (4)L→I (5)R→L



LALR分析表

考察项目集I_2：

$$S \rightarrow L \bullet = R, \#$$

$$R \rightarrow L \bullet, \#$$

1. 存在移进与归约项目，非LR(0)；
2. FOLLOW(R)={#, =}，与{=}相交不为空，非SLR(1)；
3. 是LR(1)和LALR(1)的。

合并：I_4与I_{11}，I_5与I_{12}，I_7与I_{13}，I_8与I_{10}。

状态	ACTION				GOTO		
	=	*	i	#	S	L	R
0		S4	S6		1	2	3
1				acc			
2	S6			r5			
3				r2			
4		S4	S5			8	7
5	r4			r4			
6		S4	S5			8	9
7	r3			r3			
8	r5			r5			
9				r1			

LR(1)项目集不存在动作冲突，合并同心集后会不会产生新的冲突（移进-归约、归约-归约）？

例：S→ aAd|bBd|aBe|bAe

A→ c

B→ c

不会产生新的移进-归约冲突；

会产生新的归约-归约冲突。

可见，该文法是LR(1)的但不是LALR(1)的。

非LR的上下文无关文法

语言$L=\{ww^R|w\in(a|b)^*\}$的文法：

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

对其中的任一句子，如abaaba，扫描前一半字符时应压栈，扫描后一半字符时先做空归约，然后将剩余字符和栈中字符通过归约进行比较，以保证后一半是前一半的逆。

二义性文法在LR分析中的应用

例：表达式文法

E′→E

E →E+E

E →E*E

E →(E)

E →i

二义性文法不是LR文法，但对某些二义性文法，根据背景语义人为地给出优先性和结合性，则可能构造出更有效的LR分析器

如：规定在表达式文法中

i 的优先性最高

‘*’ > ‘+’

‘*’ 和 ‘+’ 服从左结合

算术表达式二义性文法的LR(0)项目集及状态转换矩阵

状态 \ 当前符号	+	*	(	)	i	#	E
I_0: E'→•E E→•E+E E→•E*E E→•(E) E→•i			I_2: E→(•E) E→•E+E E→•E*E E→•(E) E→•i		I_3: E→i•		I_1: E'→E• E→E•+E E→E•*E
I_1:	I_4: E→E+•E E→•E+E E→•E*E E→•(E) E→•i	I_5: E→E*•E E→•E+E E→•E*E E→•(E) E→•i				acc	

当前符号 / 状态	+	*	(	)	i	#	E
I_2:			I_2:		I_3:		I_6: E→(E•) E→E•+E E→E•*E
I_3:							
I_4:			I_2:		I_3:		I_7: E→E+E• E→E•+E E→E•*E
I_5:			I_2:		I_3:		I_8: E→E*E• E→E•+E E→E•*E
I_6:	I_4:	I_5:		I_9: E→(E) •			
I_7:	I_4:	I_5:					
I_8:	I_4:	I_5:					
I_9:							

在I_1, I_7, I_8中 存在移进-归约冲突

在I_1，I_7，I_8中 存在移进 -归约冲突

I_1: $E' \rightarrow E\bullet$
$E \rightarrow E\bullet+E$
$E \rightarrow E\bullet*E$

I_7: $E \rightarrow E+E\bullet$
$E \rightarrow E\bullet+E$
$E \rightarrow E\bullet*E$

I_8: $E \rightarrow E*E\bullet$
$E \rightarrow E\bullet+E$
$E \rightarrow E\bullet*E$

I_1: 移进和接受无冲突

I_7，I_8利用优先关系和结合性解决冲突，规定：‘*’优先于‘+’，都服从左结合。

I_7: 遇‘*’移进，
遇‘+’归约。

I_8: 遇‘+’，‘*’都归约。

二义性表达式文法的LR分析表

状态	ACTION						GOTO
	+	*	(	)	i	#	E
0			S2		S3		1
1	S4	S5				acc	
2			S2		S2		6
3	r4	r4		r4		r4	
4			S2		S3		7
5			S2		S3		8
6	S4	S5		S9			
7	r1	S5		r1		r1	
8	r2	r2		r2		r2	
9	r3	r3		r3		r3	

对输入串i+i*i的分析过程

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	i+i*i#	S3	
2	03	#i	+i*i#	r4	1
3	01	#E	+i*i#	S4	
4	014	#E+	i*i#	S3	
5	0143	#E+i	*i#	r4	
6	0147	#E+E	*i#	S5	
7	01475	#E+E*	i#	S3	
8	014753	#E+E*i	#	r4	8
9	014758	#E+E*E	#	r2	
10	0147	#E+E	#	r1	1
11	01	#E	#	acc	

LR分析的错误处理---在表中插入处理程序

E –> E + E | E * E | (E) | id

State on top of stack	Action						Goto
	id	+	*	(	)	$	E
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	e3	r4	r4	e3	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	e3	r1	s5	e3	r1	r1	
8	e3	r2	r2	e3	r2	r2	
9	e3	r3	r3	e3	r3	r3	

e1: 状态 0, 2, 4, 5 调用，这些状态期望输入一个表达式的首符，而遇到的是算符，错误信息是“缺少运算量”，可将id置于栈，并盖以状态3

e2: 状态 0, 1, 2, 4, 5 调用，这些状态期望一个新表达式的开头或者是状态1的输入结束，而见到的是右括号，可能的修复：删除右括号，错误信息是“不匹配的右括号”

e3: 状态 1, 3, 6, 7, 8, 9 调用,见到的是左括号或id，期望输入的是算符，可能的修复：将+置于栈，并盖以状态4，错误信息是“缺少运算符”

e4: 状态 6 调用, 见到的是输入结束符 $. 期望输入的是算符或右括号，可能的修复：可将“）”置于栈，并盖以状态9，错误信息是“缺少右括号”

习 题

3 (1)(2)

7

8

9

补充习题

1. 下列文法是LR(0)吗？是SLR(1)吗？

a) E –> E + T | T

T –> (E) | id | id[E]

b) S –> Ab | ABc

A –> aA | a

B –> b

2. 为下列文法构造LR(1)分析表，并分析串 baab。

0) S' –> S

1) S –> XX

2) X –> aX

3) X –> b