

第6章 LR分析

6.1 自下而上分析及其LR分析概述

6.2 LR (0)分析

6.3 SLR(1)分析

6.4 LR(1)分析

6.5 LALR(1)分析

6.6 使用二义文法

6.7 语法分析程序的自动构造工具YACC

SLR(1) 分析

Real a,b,...

例: (0) $S' \rightarrow S$ (1) $S \rightarrow rD$
(2) $D \rightarrow D,i$ (3) $D \rightarrow i$

LR(0)项目

- | | | |
|-------------------------------|-------------------------------|------------------------------|
| 1. $S' \rightarrow \cdot S$ | 2. $S' \rightarrow S \cdot$ | 3. $S \rightarrow \cdot rD$ |
| 4. $S \rightarrow r \cdot D$ | 5. $S \rightarrow rD \cdot$ | 6. $D \rightarrow \cdot D,i$ |
| 7. $D \rightarrow D \cdot ,i$ | 8. $D \rightarrow D, \cdot i$ | 9. $D \rightarrow D,i \cdot$ |
| 10. $D \rightarrow \cdot i$ | 11. $D \rightarrow i \cdot$ | |

例: (0) $S' \rightarrow S$ (1) $S \rightarrow rD$
(2) $D \rightarrow D, i$ (3) $D \rightarrow i$

LR(0)项目集规范族

$I_0: S' \rightarrow .S$	$I_3: S \rightarrow r D.$
$S \rightarrow .r D$	$D \rightarrow D., i$
$I_1: S' \rightarrow S.$	$I_4: D \rightarrow i.$
$I_2: S \rightarrow r.D$	$I_5: D \rightarrow D., i$
$D \rightarrow .D, i$	$I_6: D \rightarrow D, i.$
$D \rightarrow .i$	

其中 I_3 中含有移进/归约冲突!

文法不是LR(0)的, 如何解决?

$$I_3: S \rightarrow r D. \quad D \rightarrow D., i$$

例:文法的LR(0)分析表有多重入口

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		

Forward

$I_3: S \rightarrow r D. \quad D \rightarrow D., i$

使用 $FOLLOW(S) \cap \{ , \} = \phi$ 信息解决冲突

例: 文法的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		

SLR(1)技术

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:

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

若 $\text{FOLLOW}(Q) \cap \text{FOLLOW}(P) = \phi$

$\text{FOLLOW}(P) \cap \{ b \} = \phi$

$\text{FOLLOW}(Q) \cap \{ b \} = \phi$

则解决冲突的SLR(1)技术:

action [k, b] = 移进

对 $a \in \text{FOLLOW}(P)$ 则 action [k, a] = 用 $P \rightarrow \omega$ 归约

对 $a \in \text{FOLLOW}(Q)$ 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)文法是无二义的。

例：表达式文法G的拓广文法：

(0) $S' \rightarrow E$

(1) $E \rightarrow E + T$

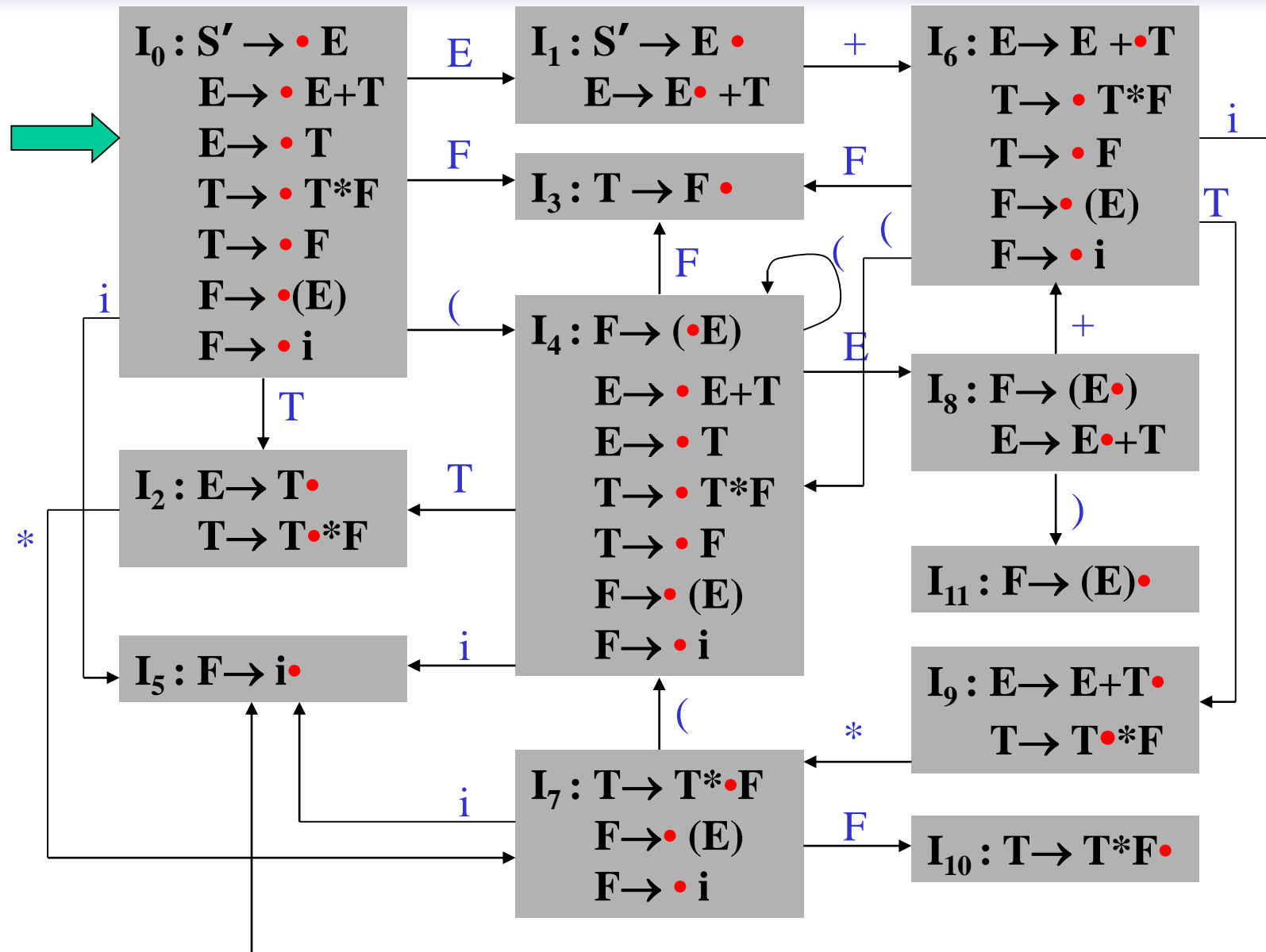
(2) $E \rightarrow T$

(3) $T \rightarrow T * F$

(4) $T \rightarrow F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow i$



Q1: I_1, I_2, I_9 中存在冲突!

$I_1: S' \rightarrow E \bullet, E \rightarrow E \bullet + T$

$I_2: E \rightarrow T \bullet, T \rightarrow T \bullet * F$

$I_9: E \rightarrow E + T \bullet, T \rightarrow T \bullet * F$

可用SLR(1)方法解决!

Q2: 无用规约!

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规约。**

若当前输入符为“(”，进行规约是多余的（输入串不合法！），**延迟了错误的发现。**

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

SLR(1)表的构造 (改进)

假定 $C = \{I_0, I_1, \dots, I_n\}$, 令每个项目集 I_k 的下标 k 为分析器的一个状态, 因此, G' 的SLR分析表含有状态 $0, 1, \dots, n$ 。令那个含有项目 $S' \rightarrow \cdot S$ 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 $A \rightarrow \alpha \cdot a\beta$ 属于 I_k 且 $GO(I_k, a) = I_j$, a 为终结符, 则置 $ACTION[k, a]$ 为 “把状态 j 和符号 a 移进栈”, 简记为 “sj”;

若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 那么对任何输入符号 a , $a \in FOLLOW(A)$, 置 $ACTION[k, a]$ 为 “用产生式 $A \rightarrow \alpha$ 进行规约”, 简记为 “rj”; 其中, 假定 $A \rightarrow \alpha$ 为文法 G' 的第 j 个产生式;

若项目 $S' \rightarrow S \cdot$ 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填入信息的空白格均置上 “出错标志”。

按上述算法构造的含有ACTION和GOTO两部分的分析表, 如果每个入口不含多重定义, 则称它为文法 G 的一张SLR表。具有SLR表的文法 G 称为一个SLR(1)文法。

(2) $D \rightarrow D, i$ (3) $D \rightarrow i$

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			

看看原来的

LR(1)分析

SLR(1)的局限

文法G:

- (0) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bAc$
(3) $S \rightarrow aec$ (4) $S \rightarrow bed$ (5) $A \rightarrow e$

LR(0) 项目集规范族:

- | | | |
|--------------------------|--------------------------|---------------------------|
| $I_0: S' \rightarrow .S$ | $I_1: S' \rightarrow S.$ | $I_2: S \rightarrow a.Ad$ |
| $S \rightarrow .aAd$ | | $S \rightarrow a.ec$ |
| $S \rightarrow .bAc$ | | $A \rightarrow .e$ |
| $S \rightarrow .aec$ | | |
| $S \rightarrow .bed$ | | |

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

$I_3: S \rightarrow b.Ac$
 $S \rightarrow b.ed$
 $A \rightarrow .e$

$I_4:$
 $S \rightarrow aA.d$

$I_5:$
 $S \rightarrow ae.c$
 $A \rightarrow e.$

$I_6:$
 $S \rightarrow bA.c$

$I_7:$
 $S \rightarrow be.d$
 $A \rightarrow e.$

$I_8:$
 $S \rightarrow aAd.$

$I_9: S \rightarrow aec.$

$I_{10}: S \rightarrow bAc.$

$I_{11}: S \rightarrow bed.$

- (0) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bAc$
 (3) $S \rightarrow aec$ (4) $S \rightarrow bed$ (5) $A \rightarrow e$

	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			r5			
7	r7	r7	r7	r7	r5	r7		
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) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bAc$

(3) $S \rightarrow aec$ (4) $S \rightarrow bed$ (5) $A \rightarrow e$

$I_5: S \rightarrow ae.c$

$A \rightarrow e.$

$S' = \overline{R} > S \overline{R} > aAd \overline{R} > aed$

$S' = \overline{R} > S \overline{R} > aec$

$I_7: S \rightarrow be.d$

$A \rightarrow e.$

$S' = \overline{R} > S \overline{R} > bAc \overline{R} > bec$

$S' = \overline{R} > S \overline{R} > bed$

在某一规范推导中，哪些输入符号能跟在句柄之后？

注意到，并不是Follow(A)中的每个元素在含A的所有句型中在A之后都会出现。

$G[S]:$

$\text{若 } S \xrightarrow[\overline{R}]{*} \alpha A \omega \Rightarrow \alpha \beta \omega, \text{ r是}\alpha\beta\text{的前缀, 则称r是G的一个活前缀。}$

Forward

$$G[S]: (0) S' \rightarrow S$$

$$(3) L \rightarrow *R$$

$$(1) S \rightarrow L=R$$

$$(4) L \rightarrow id$$

$$(2) S \rightarrow R$$

$$(5) R \rightarrow L$$

LR(0)项目集规范族

$$I_0: S' \rightarrow \bullet S$$

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

$$S \rightarrow \bullet R$$

$$L \rightarrow \bullet *R$$

$$L \rightarrow \bullet id$$

$$R \rightarrow \bullet L$$

$$I_1: S' \rightarrow S \bullet$$

$$I_2: S \rightarrow L \bullet = R$$

$$R \rightarrow L \bullet$$

$$I_3: S \rightarrow R \bullet$$

$$I_4: L \rightarrow * \bullet R$$

$$R \rightarrow \bullet L$$

$$L \rightarrow \bullet *R$$

$$L \rightarrow \bullet id$$

$$I_5: L \rightarrow id \bullet$$

$$I_6: S \rightarrow L = \bullet R$$

$$R \rightarrow \bullet L$$

$$L \rightarrow \bullet *R$$

$$L \rightarrow \bullet id$$

$$I_7: L \rightarrow *R \bullet$$

$$I_8: R \rightarrow L \bullet$$

$$I_9: S \rightarrow L = R \bullet$$

Another example

I2: $S \rightarrow L \bullet = R$, $R \rightarrow L \bullet$

考虑分析表达式 $id = id$ 时, 在工作到 I2 处已经把第一个 id 归约到 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 $Follow(R)$ 的. 第二个项目要用 $R \rightarrow L$ 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 $R = \dots$ 开头 (有以 $*R = \dots$ 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, 必须用第一个, 即移进. 只有下一个符号是 $\#$ 时才能归约. 尽管 $=$ 属于 $\text{Follow}(R)$, 那是因为一个 R 可以出现在别的上下文中, 在现在这个特定的情况, 它不合适, 因为在用 $S \Rightarrow R \rightarrow L$ 推导句子时, $=$ 不能跟在 R 后.

不是LR(0)文法

$\because I_2: S \rightarrow L.=R, R \rightarrow L.$ 中存在移进/归约冲突

SLR能否解决 I_2 中的冲突?

$FOLLOW(R) = \{ \#, = \}$ 与 $\{ = \}$ 交不为空, 不是SLR(1)文法!

若用 $R \rightarrow L$ 归约, 则形成 $R = \dots$, 而 $R =$ 不是活前缀。

LR(1)方法

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

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

可考虑把FIRST(β)作为用产生式 $B \rightarrow \gamma$ 规约的搜索符，作为规约时查看的符号集用以代替SLR(1)分析中的FOLLOW集。

LR(1)项目(配置)的一般形式

$[A \rightarrow \alpha \bullet \beta, a]$

意味着处在栈顶是 α 的相应状态, 期望相应 β 在栈顶的状态, 然后只有当跟在 β 后的token是终结符 a 时进行归约。 a 称作该项目(配置)的向前搜索符(lookahead)。

向前搜索符(lookahead)只对圆点在最后的项目起作用。

$[A \rightarrow \alpha \beta \bullet, a]$

意味着处在栈中是 $\alpha \beta$ 的相应状态, 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, 比如 a, b, c 时, 可写作

$A \rightarrow u \bullet, a/b/c$

构造LR(1)项目集规范族和GO函数

closure(I)按如下方式构造:

- (1) I的任何项目属closure(I);
- (2) 若 $[A \rightarrow \beta_1 \cdot B \beta_2, a] \in \text{closure}(I)$, $B \rightarrow \delta$ 是一产生式, 那么对于 $\text{FIRST}(\beta_2 a)$ 中的每个终结符 b , 如果 $[B \rightarrow \cdot \delta, b]$ 不在 $\text{closure}(I)$ 中, 则把它加进去;
- (3) 重复(1)(2), 直至 $\text{closure}(I)$ 不再增大。

GO函数:

若I是一个项目集, X是一个文法符号,

$\text{GO}(I, X) = \text{closure}(J)$

其中 $J = \{ \text{任何形如 } [A \rightarrow \alpha X \cdot \beta, a] \text{ 的项目} \mid [A \rightarrow \alpha \cdot X \beta, a] \in I \}$ 。

LR(1)项目集规范族C的构造算法类同LR(0), 只是初始时:

$C = \{ \text{closure}(\{[S' \rightarrow \cdot S, \#]\}) \};$

例：LR(1)项目集规范族

$I_0: S' \rightarrow .S, \#$
 $S \rightarrow .aAd, \#$
 $S \rightarrow .bAc, \#$
 $S \rightarrow .aec, \#$
 $S \rightarrow .bed, \#$

$I_1: S' \rightarrow S., \#$

$I_2: S' \rightarrow a.Ad, \#$
 $S \rightarrow a.ec, \#$
 $A \rightarrow .e, d$

$I_3: S \rightarrow b.Ac, \#$
 $S \rightarrow b.ed, \#$
 $A \rightarrow .e, c$

$I_4: S \rightarrow aA.d, \#$

$I_5: S \rightarrow ae.c, \#$

$A \rightarrow e., d$

$I_6: S \rightarrow bA.c, \#$

$I_7: S \rightarrow be.d, \#$

$A \rightarrow e., c$

$I_8: S \rightarrow aAd., \#$

$I_9: S \rightarrow aec., \#$

$I_{10}: S \rightarrow bAc., \#$

$I_{11}: S \rightarrow bed., \#$

(0) $S' \rightarrow S$

(1) $S \rightarrow aAd$

(2) $S \rightarrow bAc$

(3) $S \rightarrow aec$

(4) $S \rightarrow bed$

(5) $A \rightarrow e$

规范的LR(1)分析表的构造

假定LR(1)项目集规范族 $C=\{I_0, I_1, \dots, I_n\}$, 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 $0, 1, \dots, n$.

1. 若项目 $[A \rightarrow \alpha.a\beta, b]$ 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 $[A \rightarrow \alpha., a]$ 属于 I_k , 那么置 $ACTION[k, a]$ 为“用产生式 $A \rightarrow \alpha$ 进行规约”, 简记为“ rj ”;其中, j 为产生式 $A \rightarrow \alpha$ 在文法 G 中的编号.
3. 若项目 $[S' \rightarrow S., \#]$ 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. 分析表分析中凡不能用规则1至5填入信息的空白格均置上“出错标志”。

按上述算法构造的含有ACTION和GOTO两部分的分析表, 如果每个入口不含多重定义, 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)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. 如果一个项目集里有项目 $[A \rightarrow u \bullet x v, a]$, x 是终结符, 那就不会有项目 $[B \rightarrow u \bullet, x]$;
2. 项目集里所有归约项目的向前搜索符不相交, 即不能同时含有项目 $[A \rightarrow u \bullet, a]$ 和 $[B \rightarrow v \bullet, a]$

LR(1)比SLR(1)能力强

例如文法 $G[S']$:

(0) $S' \rightarrow S$

(1) $S \rightarrow L = R$

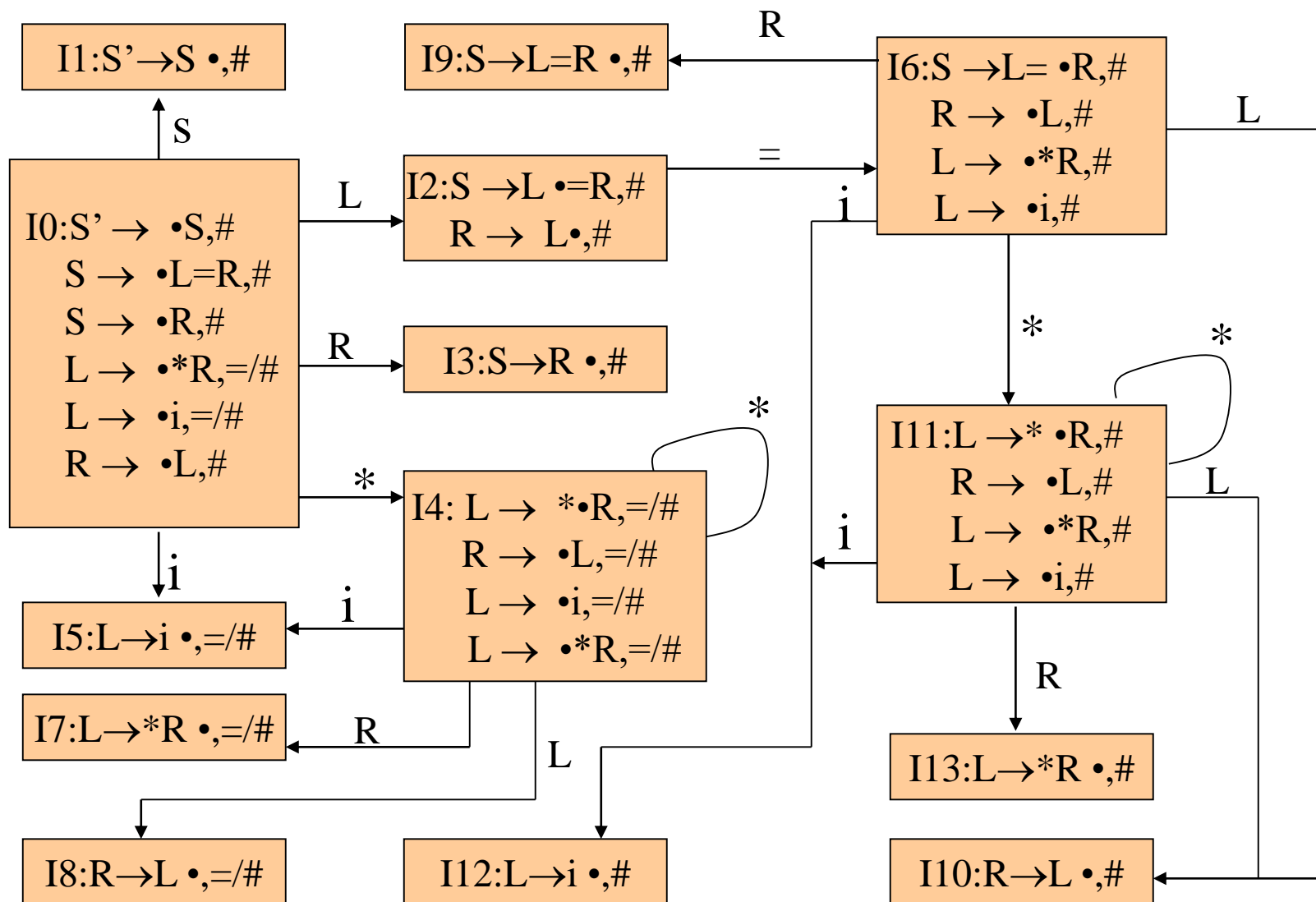
(2) $S \rightarrow R$

(3) $L \rightarrow *R$

(4) $L \rightarrow i$

(5) $R \rightarrow L$

不能用SLR(1)技术解决，但能用LR(1)技术解决。



每个SLR(1)文法都是LR(1)的，一个SLR(1)文法的LR(1)分析器比其SLR(1)分析器的状态要多。

例G(S): $S \rightarrow BB$ $B \rightarrow aB$ $B \rightarrow b$

$I_0: S' \rightarrow .S \quad S \rightarrow .BB \quad B \rightarrow .aB \quad B \rightarrow .b$

$I_1: S' \rightarrow S.$

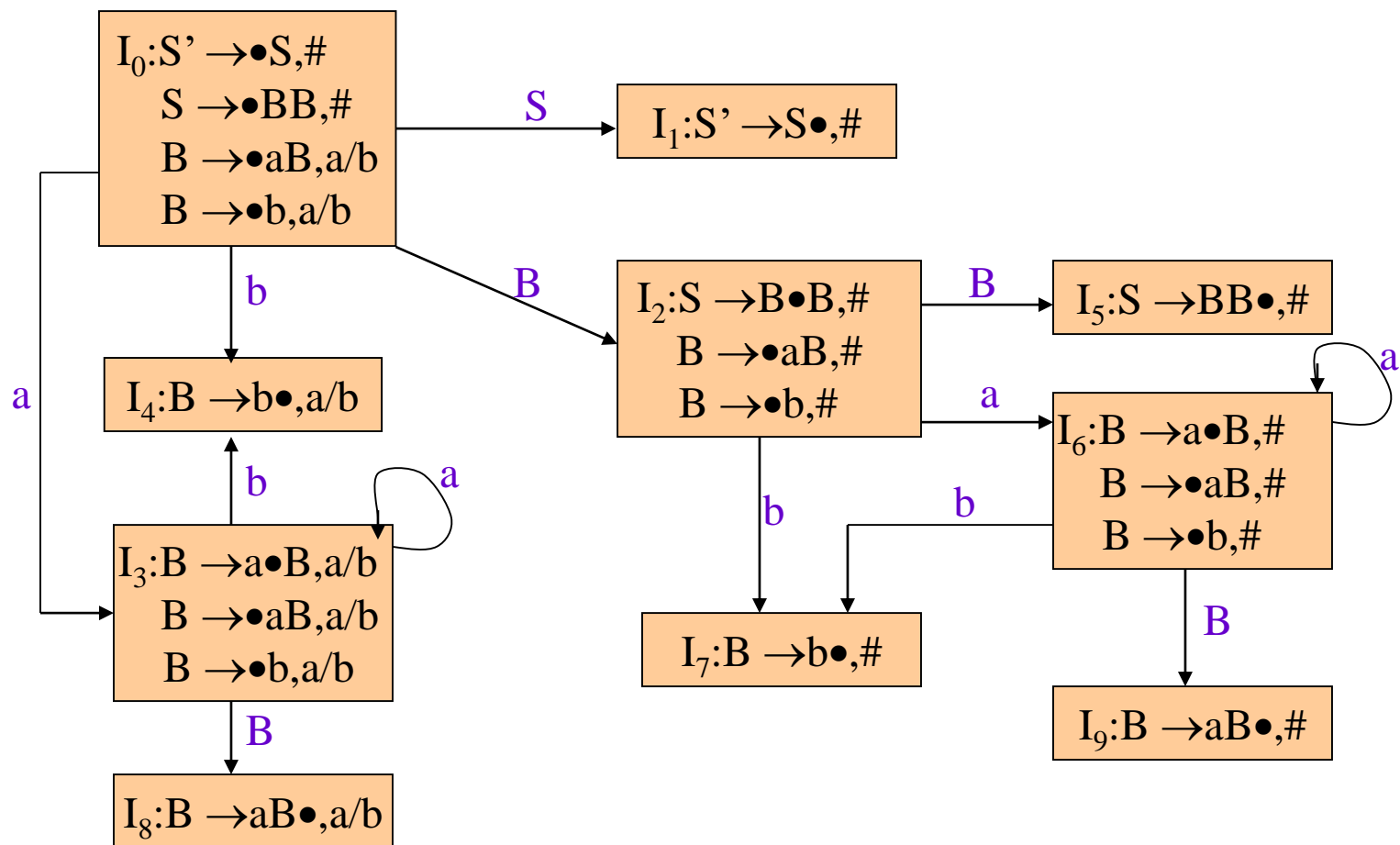
$I_2: S \rightarrow B.B \quad B \rightarrow .aB \quad B \rightarrow .b$

$I_3: B \rightarrow a.B \quad B \rightarrow .aB \quad B \rightarrow .b$

$I_4: B \rightarrow aB.$

$I_5: B \rightarrow b.$

$I_6: S \rightarrow BB.$



LR(1)项目集和转换函数

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*)

$$\begin{array}{l} I_3: B \rightarrow a \bullet B, a/b \\ B \rightarrow \bullet a B, a/b \\ B \rightarrow \bullet b, a/b \end{array}$$

$$I_4: B \rightarrow b \bullet, a/b$$

$$I_8: B \rightarrow a B \bullet, a/b$$

$$I_7: B \rightarrow b \bullet, \#$$

$$I_9: B \rightarrow a B \bullet, \#$$

$$\begin{array}{l} I_6: S \rightarrow a \bullet B, \# \\ B \rightarrow \bullet a B, \# \\ B \rightarrow \bullet b, \# \end{array}$$

合并同心集 $I_3 I_6$ $I_4 I_7$ $I_8 I_9$ 。

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规约/规约冲突，而不可能产生移进/规约冲突。
- 4、合并同心集后对某些错误发现的时间会产生推迟现象。

合并同心集后若有冲突也只可能是规约/规约冲突，而不可能产生移进/规约冲突。

若LR(1)文法的项目集 I_k 与 I_j 为同心集，其中，

I_k : $[A \rightarrow \alpha., u_1]$

$[B \rightarrow \beta.a\gamma, b]$

I_j : $[A \rightarrow \alpha., u_2]$

$[B \rightarrow \beta.a\gamma, c]$

合并后 I_{kj} :

$[A \rightarrow \alpha., u_1/u_2]$

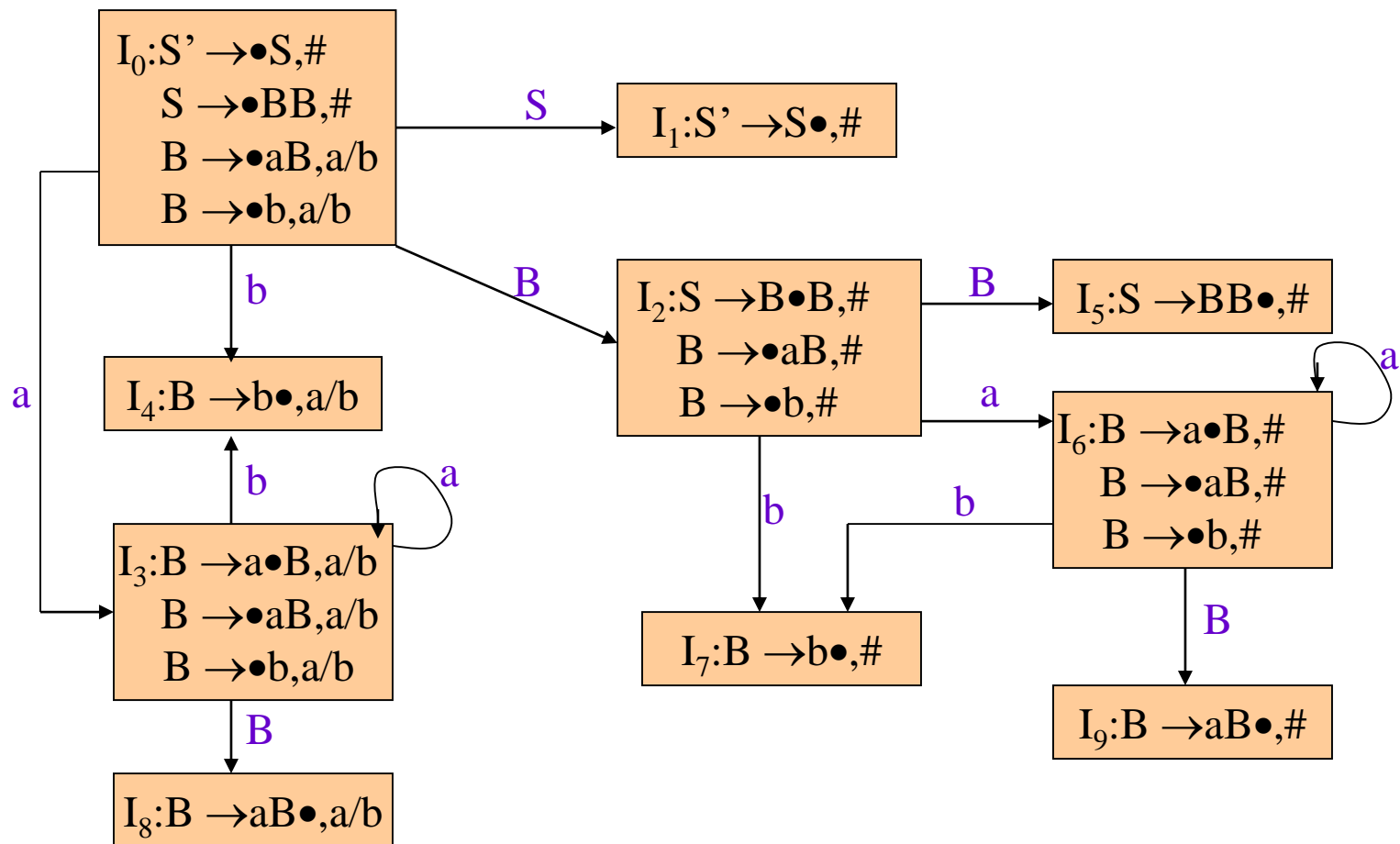
$[B \rightarrow \beta.a\gamma, b/c]$

构造 LALR(1)分析表

- 1.构造文法G的规范 LR(1) 状态.
- 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)项目集和转换函数 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		

状态	ACTION			GOTO	
	a	b	#	S	B
0	$S_{3,6}$	$S_{4,7}$	acc	1	2
1					
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

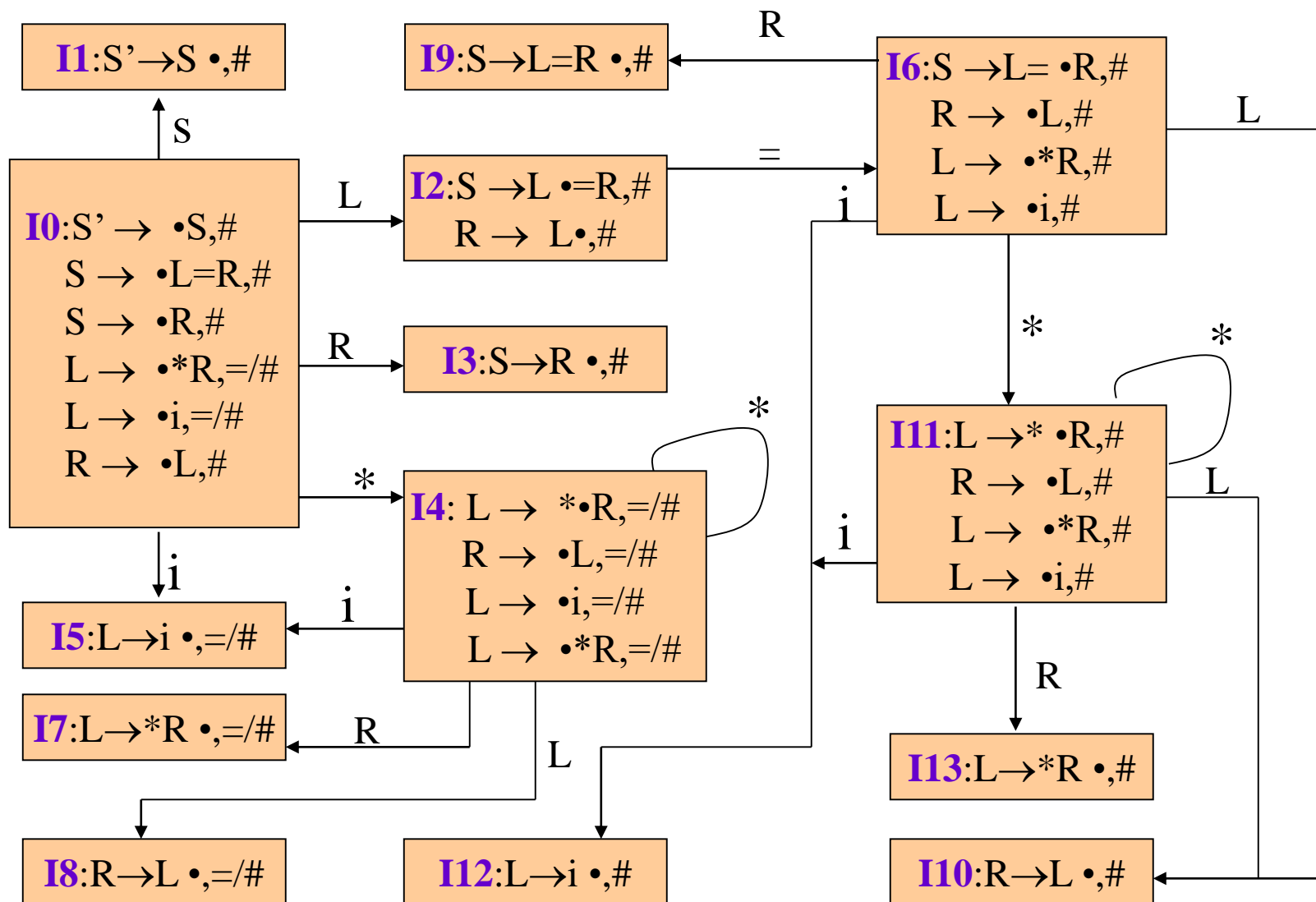
步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	ab#	S ₃	
2	03	#a	b#	S ₄	
3	034	#ab	#	出错	

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) $S' \rightarrow S$ (1) $S \rightarrow L=R$ (2) $S \rightarrow R$ (3) $L \rightarrow *R$ (4) $L \rightarrow I$ (5) $R \rightarrow L$



LALR分析表

考察项目集 I_2 :

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

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

- 1、存在移进与规约项目, 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： $S \rightarrow aAd|bBd|aBe|bAe$

$A \rightarrow c$

$B \rightarrow c$

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 $L = \{ ww^R \mid w \in (a|b)^* \}$ 的文法:

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

对其中的任一句子，如abaaba，扫描前半一半字符时应压栈，扫描后半一半字符时先做空规约，然后将剩余字符和栈中字符通过规约进行比较，以保证后半一半是前半一半的逆。

二义性文法在LR分析中的应用

例如表达式文法:

$$E' \rightarrow E$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow (E)$$
$$E \rightarrow i$$

二义性文法不是LR文法, 但是对某些二义性文法, 人为地给出优先性和结合性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

$'*' > '+'$

$'*' & '+'$ 都服从左结合

算术表达式二义性文法的 LR(0)项目集及状态转换矩阵

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

当前 符号 状态	+	*	()	i	#	E
$I_2:$			$I_2:$		$I_3:$		$I_6:$ $E \rightarrow (E \bullet)$ $E \rightarrow E \bullet + E$ $E \rightarrow E \bullet * E$
$I_3:$							
$I_4:$			$I_2:$		$I_3:$		$I_7:$ $E \rightarrow E + E \bullet$ $E \rightarrow E \bullet + E$ $E \rightarrow E \bullet * E$
$I_5:$			$I_2:$		$I_3:$		$I_8:$ $E \rightarrow E * E \bullet$ $E \rightarrow E \bullet + E$ $E \rightarrow E \bullet * E$
$I_6:$	$I_4:$	$I_5:$		$I_9:$ $E \rightarrow (E) \bullet$			
$I_7:$	$I_4:$	$I_5:$					
$I_8:$	$I_4:$	$I_5:$					
$I_9:$							

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 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

$E \rightarrow E + E \mid E * E \mid (E) \mid id$

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) $E \rightarrow E + T \mid T$

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

b) $S \rightarrow Ab \mid ABc$

$A \rightarrow aA \mid a$

$B \rightarrow b$

2. 为下列文法构造LR(1)分析表, 并分析串 baab。

0) $S' \rightarrow S$

1) $S \rightarrow XX$

2) $X \rightarrow aX$

3) $X \rightarrow b$