华中科技大学

编译原理

第6章 LR分析

2019年11月27日星期三

编译原理课程组

内容摘要

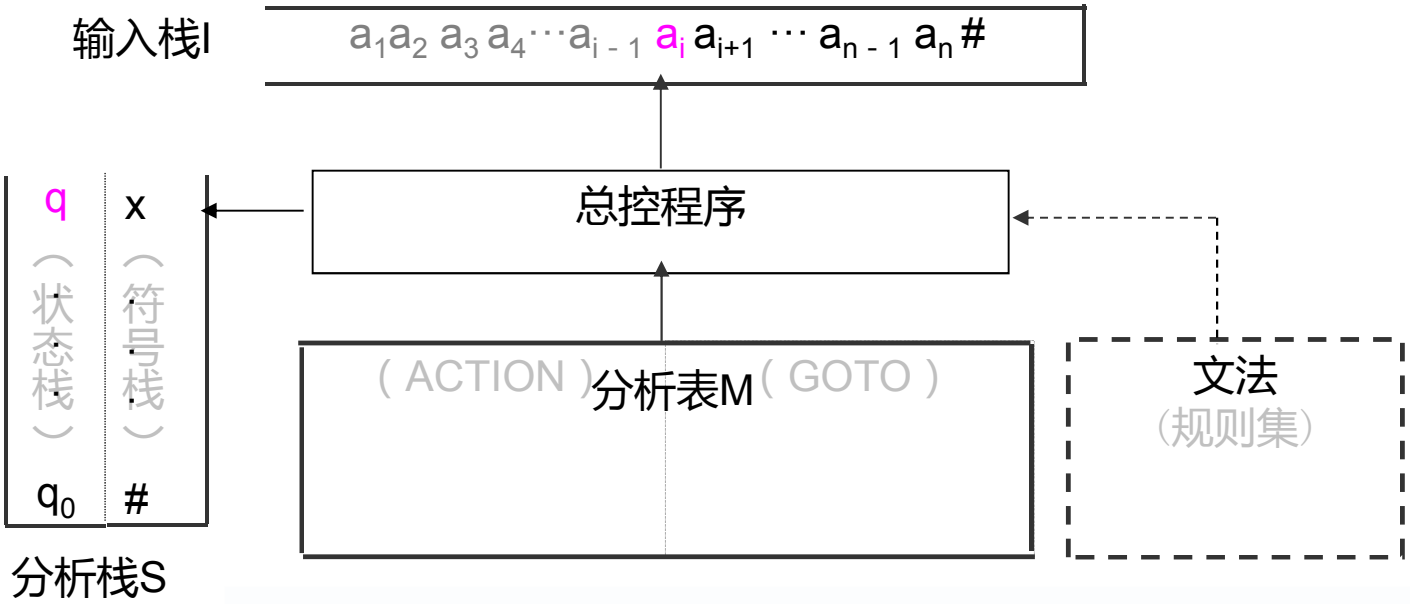
本章研究自底向上的LR分析法，LR分析法是一类归约法的统称，主要介绍其中最基本的LR(0)、SLR(1)、LR(1)和LALR(1)四种分析法，重点讨论可归约前缀的作用、识别活前缀DFA的构造、分析表的构造、分析法适用条件和语法分析程序结构及其分析算法。

重点讲解

6.1 LR分析概述

LR分析法也称为**LR(K)分析法**。这里，L表示从左到右扫描输入串，R表示最右推导之逆过程(即规范归约)，K表示向右查看输入串符号个数。

这类自底向上的语法分析法，其总体框架可以划分为总控程序、分析栈和分析表三个组成部分，如下图所示。



寻找句柄是根据向右查看输入串的K个符号，结合分析所处的“状态”，确定句柄是否出现在分析栈顶部。

假设文法G[S]和分析表M，状态0为开始状态，q为状态栈S.Q栈顶元素；a为输入栈I栈顶元素，则**总控程序的算法**如下：

(1) **初始化**：“0#”进栈S；

(2) **移进**：如果M.ACTION[q,a]为s_j，则 将“ja”进栈S，输入栈I出栈，转到步骤(2)；

(3) **归约**：如果M.ACTION[q,a]为r_i，则

(3.1) 令第i条规则为A→α。将｜α｜个状态和符号退出分析栈S；

(3.2) 令q′为此刻状态栈S.Q栈顶元素。如果M.GOTO[q′,A]为状态j，将“jA”进栈S，转到步骤(2)；否则M.GOTO[q′,A]为e_k，转入出错处理ERROR()；

(4) **报错**：如果M.ACTION[q,a]为e_k，则 转入出错处理ERROR()；

(5) **接受**：如果M.ACTION[q,a]为acc，则 输出“OK”，结束。

例6.1 设文法G[S]定义如右，并已知分析表M见表7.1，其中表中空白出表示e_k。试给出输入串abbcde的LR分析过程。

G[S]：
(1) S→aAcBe
(2) A→b
(3) A→Ab
(4) B→d

表 7.1 G[S] LR 分析表 M

$V_T \cup V_N$ A/G 状态	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S_2						1		
1						acc			
2				S_4				3	
3		S_5		S_6					
4	r_2	r_2	r_2	r_2	r_2	r_2			
5					S_8				7
6	r_3	r_3	r_3	r_3	r_3	r_3			
7			S_9						
8	r_4	r_4	r_4	r_4	r_4	r_4			
9	r_1	r_1	r_1	r_1	r_1	r_1			

6.2 LR(0)分析

6.2.1 可归前缀和活前缀

以例7.1定义文法G[S]为例，讨论LR分析法的基本原理，同时将提出可归前缀和活前缀重要概念。

G[S]:	(1) S→aAcBe[1]
	(2) A→b[2]
	(3) A→Ab[3]
	(4) B→d[4]

S⇒aAcBe[1]⇒aAcd[4]e[1]⇒aAb[3]cd[4]e[1]⇒ab[2]b[3]cd[4]e[1]

ab[2]b[3]cd[4]e[1]<=aAcd[4]e[1]<=aAcBe[1]<=S

如果使用分析栈实现这个过程，则方法也很简单。将输入串符号移进分析栈，直到遇到“**编号**”为止；这时，句柄出现在分析栈顶部，令编号代表的规则是A→α，将分析栈顶部｜α｜个符号出栈，A进栈便完成一次归约。重复这些步骤，直到归约出S。

6.2.1 可归前缀和活前缀

定义7.1 将符号串的任意含有头符号的子串称为前缀。特别地，空串ε为任意串的前缀。

定义7.2 设文法G[S]，如果$S \overset{*}{\underset{R}{\Rightarrow}} \alpha A\omega \underset{R}{\Rightarrow} \alpha\beta\omega$是句型αβω的规范推导，则αβ称为可归前缀，αβ的前缀称为活前缀。

即：尾符号恰好是句柄β尾符号的文法规范句型之前缀，称为可归前缀，可归约前缀之前缀称为活前缀。

例如文法G[S]，句型aAbcde的句柄为Ab，活前缀有：ε、a、aA和aAb，其中，aAb为可归前缀。

G[S]：(1) S→aAcBe
(2) A→b
(3) A→Ab
(4) B→d

假设事先知道文法所有规范句型可归前缀，使用分析栈实现分析的具体步骤修改为：

将输入串符号移进分析栈，直到分析栈出现“可归前缀”为止；这时，句柄出现在分析栈顶部，令可归前缀编号代表的规则是A→α，将分析栈顶部｜α｜个符号出栈，A进栈便完成一次归约。重复这些步骤，直到归约出S。显然不再需要输入串夹带着编号，也可以得到规范归约。

S⇒aAcBe[1]⇒aAcd[4]e[1]⇒aAb[3]cd[4]e[1]⇒ab[2]b[3]cd[4]e[1]

例7.1定义文法G[S]的可归前缀如下：ab[2]、aAb[3]、aAcd[4]、aAcBe[1]，使用分析栈实现abbcde的分析过程如下。

abbcde ⇐aAbcde ⇐aAcde ⇐aAcBe ⇐ S

6.2.2 识别活前缀DFA

识别活前缀DFA技术线路是根据文法G，构造识别活前缀NFA M。之后通过子集法，将NFA M确定化，得到识别活前缀DFA M'。

设文法G=(VN, VT, P, S)，构造识别活前缀NFA M的方法如下：

(1) 文法G等价改写成G'：

文法G' = (VN ∪ {S'}, VT, P ∪ {S' →S}, S')

这里：VN ∩ {S'}=Φ；

(2) 每个规则A→α构造等价一个NFAMA→α：

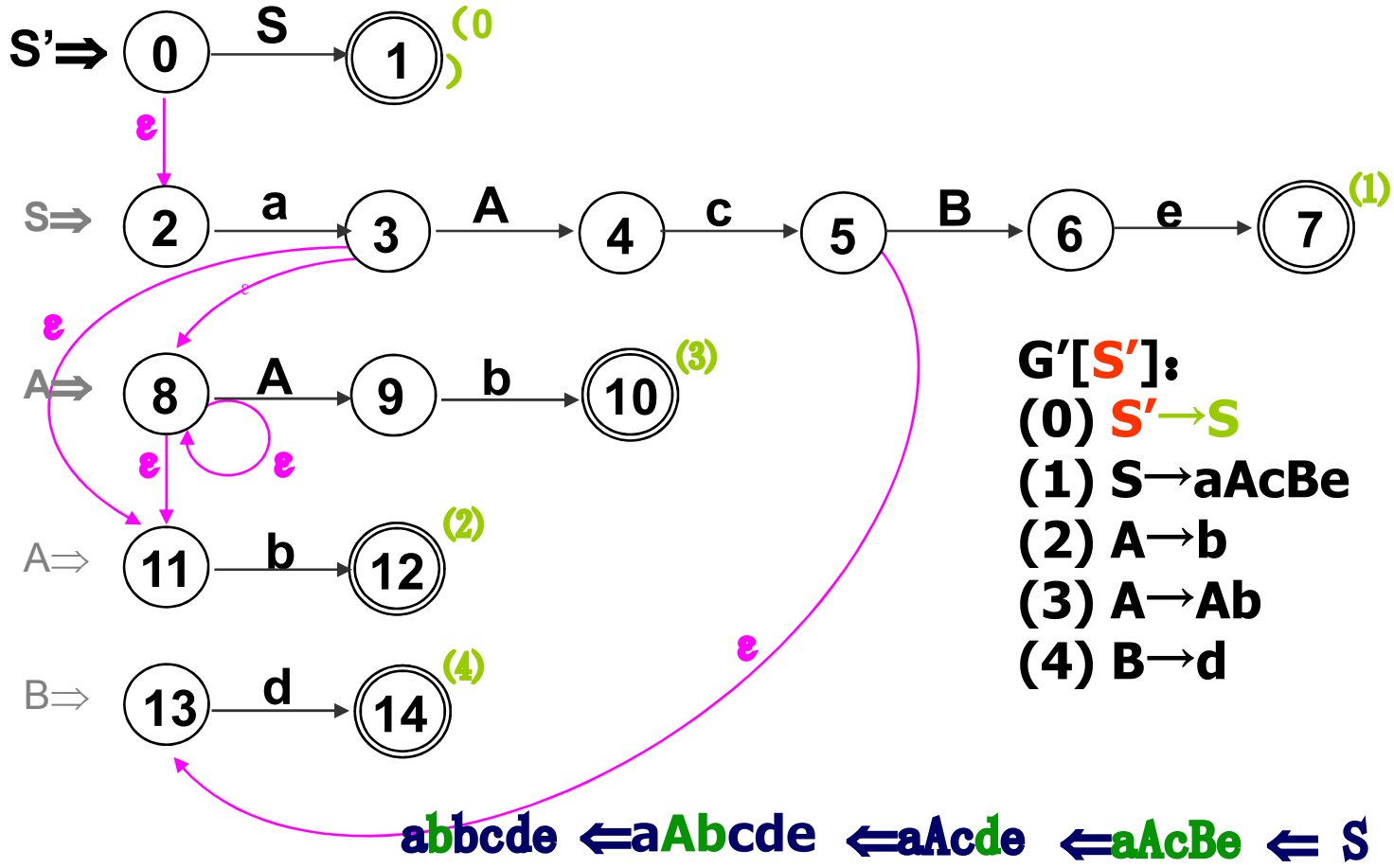
令α=x1x2・・・xn，增加n+1个状态q1、q2、q3、・・・、qn+1和转换f(qi, xi)={qi+1}(1≤i≤n)，q1为开始状态，qn+1为结束状态，Σ={ x1, x2, ・・・, xn }；

(3) 合并所有规则的NFAMA→α，构造成一个NFA M：

如果MA→α有f(q, B)={p}(B∈VN)，且NFA MB→β对应开始状态为q'，增加转换f(q, ε)⊇{ q' }；最后仅仅保留NFA MS'→S 的开始状态为NFA M的开始状态。

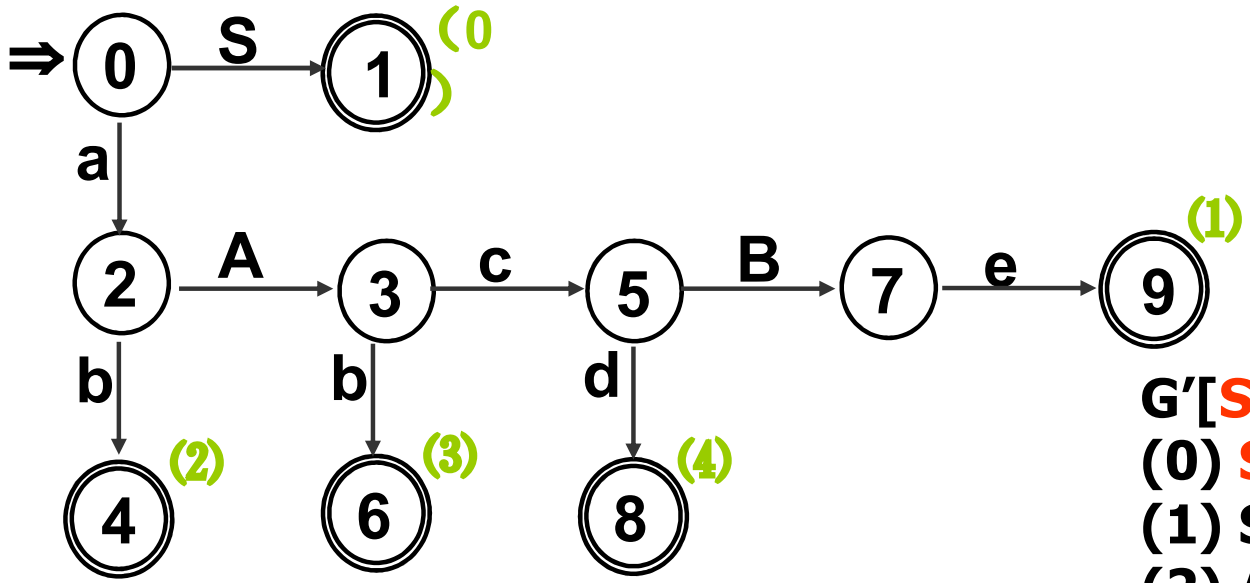
根据文法构造识别活前缀NFA

例7.1定义的文法G[S]，其等价文法G’[S’]如下，识别活前缀NFA M构造过程如下图所示。



确定化构造识别活前缀DFA

采用子集法，将NFA M确定化得到DFA M’，如下图所示。



G’[S’]：
(0) S’→S
(1) S→aAcBe
(2) A→b
(3) A→Ab
(4) B→d

根据得到DFA M’，输入串abbcde分析过程如下所示。

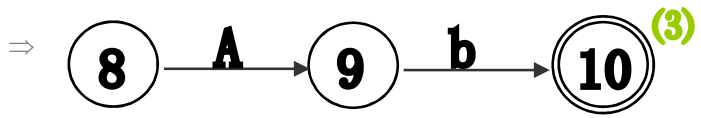
abbcde ⇐aAbcde ⇐aAcde ⇐aAcBe ⇐ S

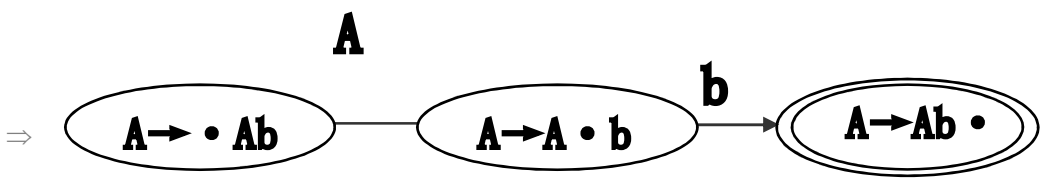
6.2.4 构造LR(0)项目集规范族

根据文法，可以直接构造识别活前缀DFA，其方法是将NFA构造和确定化合并一步完成。

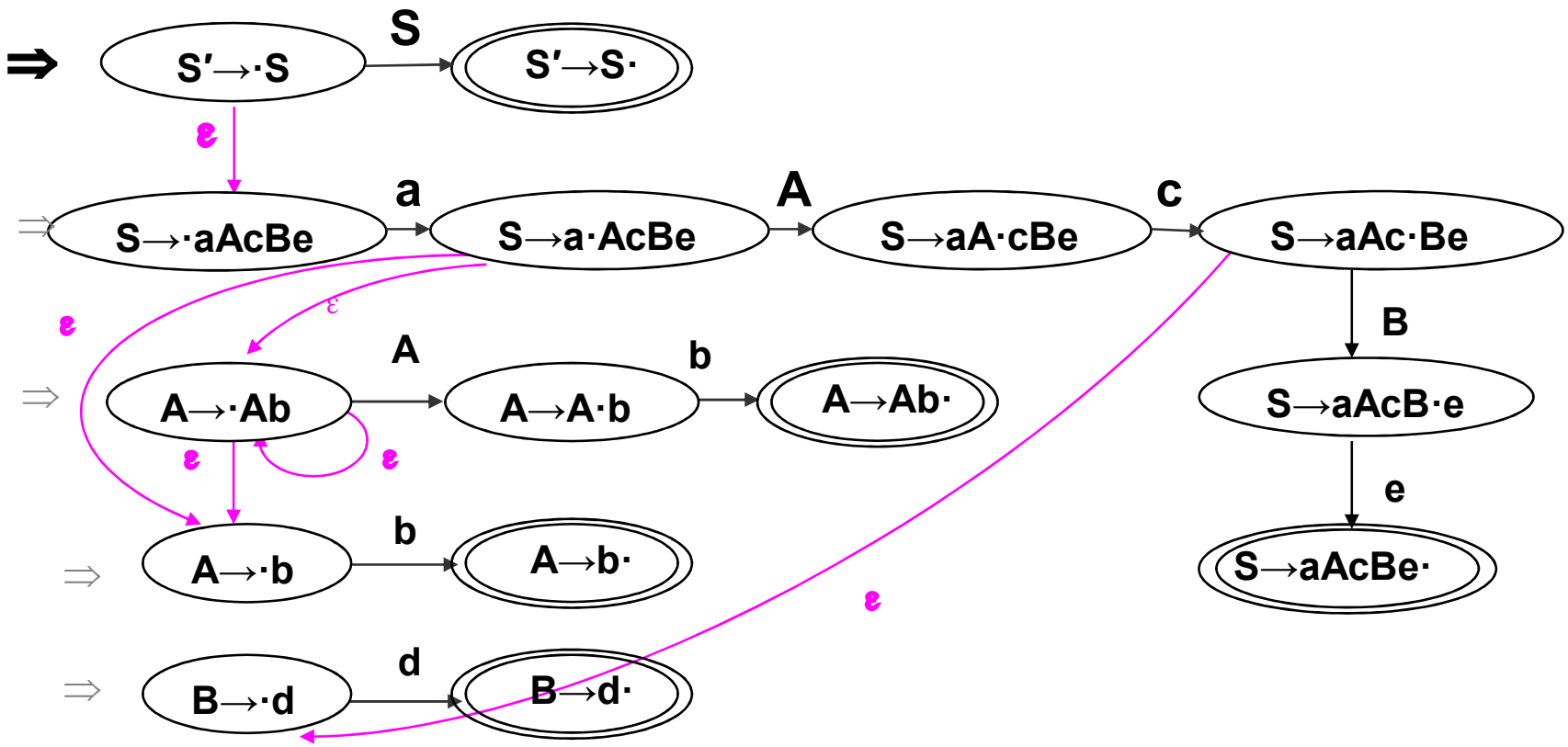
$M_{A\to\alpha}$的状态可以直接由A→α规则来命名：A→状态之前右部符号·状态之前右部符号。

例如规则：　A→·Ab

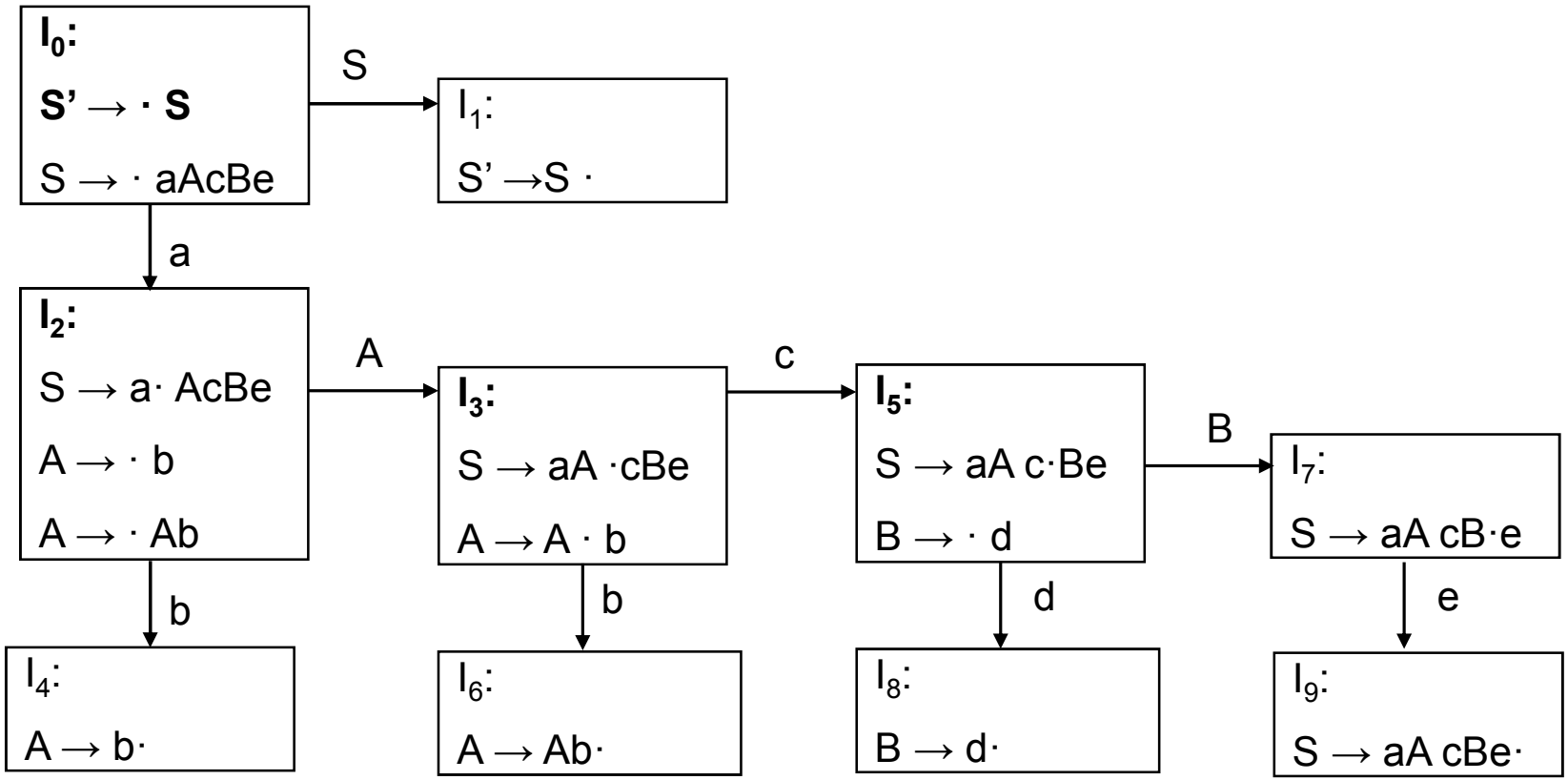




重新命名后的识别活前缀的NFA M



识别活前缀DFA



很容易地看出对应的DFA构造规律！

LR(0)项目

文法G[S’]之**LR(0)项目**是由规则右部最前、或最后、或两符号之间增加一个点符号“·”形成的。

可划分四类如下：

① **移进项目**：

形如A→α·aβ之项目称为移进项目。

② **待约项目**：

形如A→α·Xβ之项目称为待约项目。

③ **归约项目**：

形如A→α·之项目称为归约项目。

④ **接受项目**：

形如S′→α·之项目称为接受项目。

（其中α、$\beta \in (V_N \cup V_T)^*$，$a \in V_T$，$X \in V_N$）

特别地，空规则A→ε对应的LR(0)项目为A→·

LR(0)项目的MOVE运算定义

定义 7.4　设I是文法G的LR(0)项目子集，则Move(I，X)定义如下：

$$Move(I, X) = \{A \to \alpha X \cdot \beta \mid A \to \alpha \cdot X\beta \in I\}$$

例：

$I=\{S \to a \cdot AcBe,\ A \to \cdot Ab,\ A \to \cdot b\}$

$Move(I,A)=\{S \to aA \cdot cBe,\ A \to A \cdot b\}$

$Move(I,b)=\{A \to b \cdot\}$

LR(0)项目的CLOSURE运算定义

定义 7.5　设I是文法G的LR(0)项目子集，则closure(I)定义如下：

⑴ $I \subset closure(I)$

⑵ $\{B \rightarrow \cdot \gamma \mid A \rightarrow \alpha \cdot B\beta \in closure(I)\} \subseteq closure(I)$

⑶ 重复⑵，直到closure(I)，不再扩大为止。

例：I={S→a · AcBe}

closure(I)={ S→a · AcBe

A→·Ab

A→·b }

LR(0)识别活前缀DFA　M构造方法

设文法G＝(V$_N$，V$_T$，P，S)，且已等价改写成文法G，即G′＝(V$_N$∪{S′}，V$_T$，P∪{S′→S}，S′)，且V$_N$∩{S′}＝Φ，则识别活前缀DFA　M＝(K,Σ，f,S,Z)，其中：

(1) K ⊆ ρ (LR(0)项目集)

(2) Σ ＝ V$_N$∪V$_T$

(3) f(I,X) ＝ closure(Move(I，X)),I∈ K，X∈Σ

(4) S ＝ closure(S′ →·S)

(5) Z ＝ {q | q∈K，q 含有归约项目}

定义 7.6　文法G的识别活前缀DFA　M的状态集称为文法G的LR(0)项目集规范族。

例6.2　对于例6.1定义的文法G[S]，直接构造识别活前缀DFA　M。

G[S]：(1)　S→aAcBe

(2)　A→b

(3)　A→Ab

(4) B→d

i）文法等价改写，并给规则编号，结果如下：

G′ [S′]：

(0)　S′→S

(1)　S→aAcBe

(2)　A→b

(3)　A→Ab

(4)　B→d

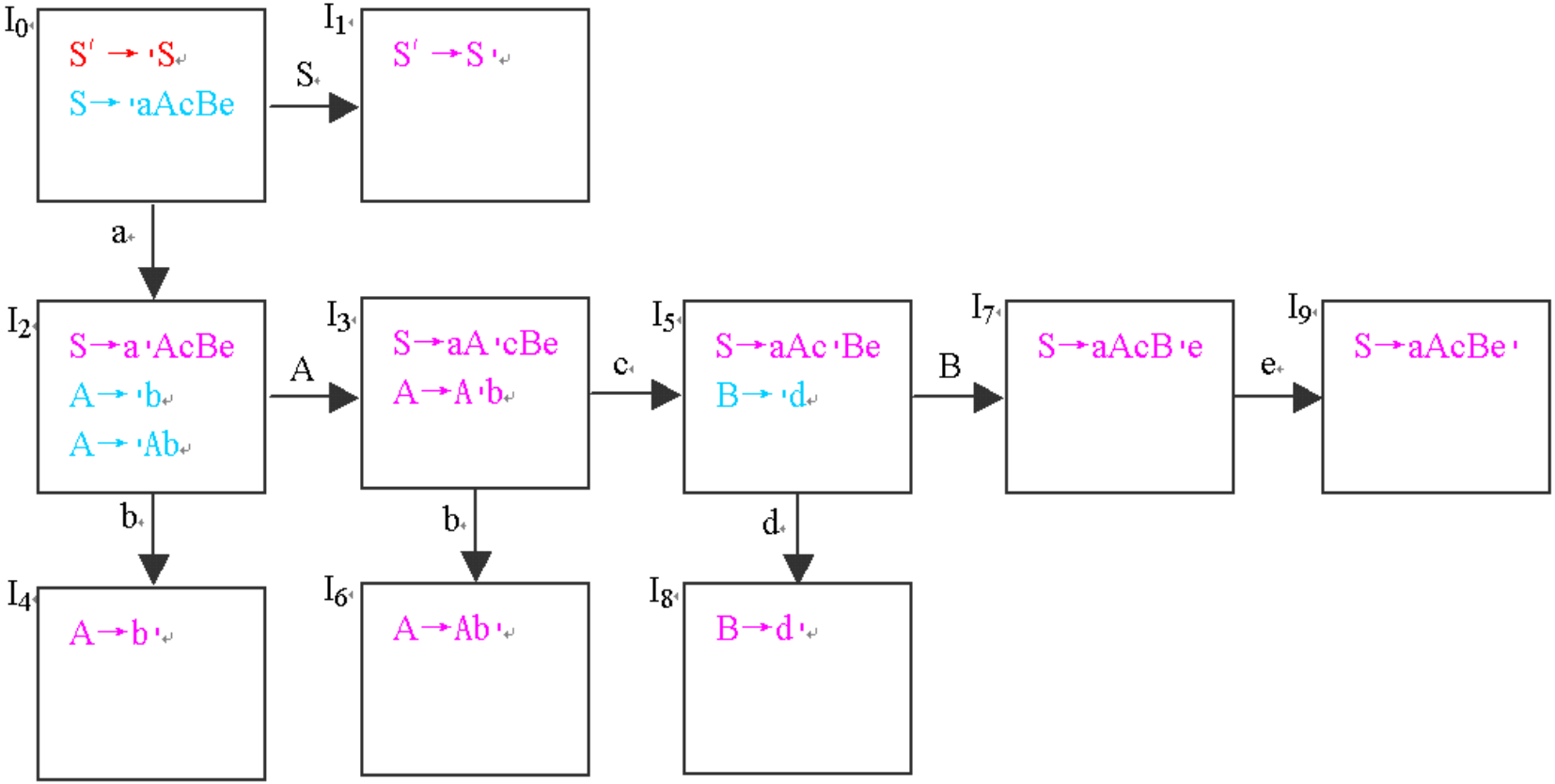
ii）根据规则，得LR(0)项目如下：

S′ →・S	S→aAc・Be	A→・Ab
S′ →S・	S→aAcB・e	A→A・b
S→・aAcBe	S→aAcBe・	A→Ab・
S→a・AcBe	A→・b	B→・d
S→aA・cBe	A→b・	B→d・

iii）根据LR(0)项目，得识别活前缀DFA　M如下。其中，矩形表示状态，粉红色是Move计算结果，天蓝色是closuer计算结果，I_k是状态名称，下表k是状态编号。I_0是开始状态，I_1、I_4、I_5、I_7和I_9是结束状态。

构造识别活前缀DFA　M过程演示。

LR(0)项目集规范族



iv）根据识别活前缀DFA M，得文法G的LR(0)项目集规范族C如下：

$$C=\{ I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9\}$$

LR(0)分析表的构造

设文法G的LR(0)项目集规范族C＝{ I_0，I_1，・・・，I_n}，且f为转换函数，则

(1) 对每一个LR(0)项目，依据下列情况分别填分析表：

如果移进项目A→α・ aβ ∈I_k，f(I_k，a)＝I_j，则

置M.ACTION[k,a]为S_j；

如果归约项目A→α・ ∈ I_k，A→α 标号为i，

∀a∈(V_T∪{#})，置M.ACTION[k,a]为r_i；

如果接受项目S′ → S・ ∈I_k，则

置M.ACTION[k,#]为acc；

如果f(I_k,A)＝I_j，A∈V_N，则

置M.GOTO[k,A]为j；

(2) 凡(1)没能填入分析表元素M.ACTION[k,a]和M.GOTO[k,a]置为e_t(t为错误编号)。

LR(0)分析表的构造

例如，对于例6.2构造的文法G[S] 识别活前缀DFA M，构造分析表如下。

A/G V 状态	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S_2						1		
1						acc			
2				S_4				3	
3		S_5		S_6					
4	r_2	r_2	r_2	r_2	r_2	r_2			
5					S_8				7
6	r_3	r_3	r_3	r_3	r_3	r_3			
7			S_9						
8	r_4	r_4	r_4	r_4	r_4	r_4			
9	r_1	r_1	r_1	r_1	r_1	r_1			

LR(0)文法的定义

如果同时含有移进项目和归约项目的项目集称为含有**移进-归约冲突**的项目集。如果同时含有一个以上的归约项目的项目集称为含有**归约-归约冲突**的项目集。

定义 7.7　如果文法G的LR(0)项目集规范族不存在移进-归约冲突或归约-归约冲突的项目集，则文法G称为**LR(0)文法**。

关于LR(0)文法，可以得出下列几个结论。

⑴如果文法G是LR(0)文法，则G可采用LR(0)分析法。

⑵如果文法G是LR(0)文法，则G是无二义性的。

如果文法G 是LR(0)文法，其分析ACTION表中每格仅会是移进、归约和报错3种动作之一。

6.3 SLR(1)分析

某些文法不是LR(0)文法时，可以采用简单地向右看一个输入符号的方法，解决文法LR(0)项目集规范族存在移进-归约冲突或归约-归约冲突的情况。

假设文法LR(0)项目集规范族有一个并存移进-归约冲突和归约-归约冲突的项目集I_k，

$I_k=\{A\rightarrow\alpha\cdot a\beta,\ A\rightarrow\gamma\cdot,\ B\rightarrow\delta\cdot,\cdots\}$

如果{a}、FOLLOW(A)和FOLLOW(B)没有相同的符号(即两两相交均为空集)，那么根据输入栈顶符号a_i属于这3个集合的哪个集合，就可以分别采用相应的分析动作了。显然，a_i不属于任何一个集合时，表明已经发现输入串的语法错误。

这样解决冲突问题思路的形成一般方法，这种分析方法称为SLR(1)分析法。

SLR(1)分析表M构造方法

(1) 对每一个LR(0)项目，依据下列情况分别填分析表：

如果移进项目$A\to\alpha\cdot a\beta\in I_k$，$f(I_k,a)=I_j$，则

置M.ACTION[k,a]为S_j；

如果归约项目$A\to\alpha\cdot\in I_k$，$A\to\alpha$标号为i，$a\in FOLLOW(A)$，则

置M.ACTION[k,a]为r_i；

如果接受项目$S'\to S\cdot\in I_k$，则

置M.ACTION[k,#]为acc；

如果$f(I_k,A)=I_j$，$A\in V_N$，则

置M.GOTO[k,A]为j；

(2) 凡(1)没能填入分析表元素M.ACTION[k,a]和M.GOTO[k,a]，

置为e_t（t为错误编号）。

例7.3 设文法G[S′]定义如下，试构造识别活前缀DFA和SLR(1)分析表M。

G[S′]：（0） S′→S （1） S→rD
（2） D→D，i （3） D→i

注： FOLLOW(S)＝{#}
移进符号集＝{，}

SLR(1)文法的定义

定义 7.8　设文法G的LR(0)项目集规范族C中任意含有m个移进项目和n个归约项目的冲突项目集I_k的一般形式为

$$I_k=\{A_1\to\alpha_1\cdot a_1\beta_1,\ A_2\to\alpha_2\cdot a_2\beta_2,\ \cdots,\ A_m\to\alpha_m\cdot a_m\beta_m,$$
$$B_1\to\gamma_1\cdot,\ B_2\to\gamma_2\cdot,\ \cdots,\ B_n\to\gamma_n\cdot,\ \cdots\}$$

（其中，A_i、$B_j\in V_N$，$a_i\in V_T$，α_i、βj、$\gamma_j\in(V_N\cup V_T)^*$，

$\cdots$表示剩下的待约项目），

如果移进符号集$\{a_1,\ a_2,\ \cdots,\ \alpha_m\}$和FOLLOW($B_1$)、FOLLOW($B_2$)、$\cdots$、FOLLOW($B_n$)两两相交均为空集，则文法G称为SLR(1)文法。

关于SLR(1)文法，可以得出下列几个结论。

(1)如果文法G是SLR(1)文法，则G可采用SLR(1)分析法。

(2)如果文法G是SLR(1)文法，则G是无二义性的。

(3)如果文法G是LR(0)文法，则G一定是SLR(1)。

例题：文法G[S]为：　S → AB

A → aBa | ε

B → bAb | ε

1. 该文法是SLR(1)的吗?

2. 若是请构造它的分析表;

3. 给出输入串baab#的分析过程.

拓广文法:

(0) S' → S

(1) S → AB

(2) A → aBa

(3) A → ε

(4) B → bAb

(5) B → ε

First(S')= {ε, a, b}

First(S)= {ε, a, b}

First(A)= {ε, a}

First(B)= {ε, b}

Follow(S')= {#}

Follow(S)= {#}

Follow(A)= {b, #}

Follow(B)= {a, #}

拓广文法:

(0) S' → S

(1) S → AB

(2) A → aBa

(3) A → ε

(4) B → bAb

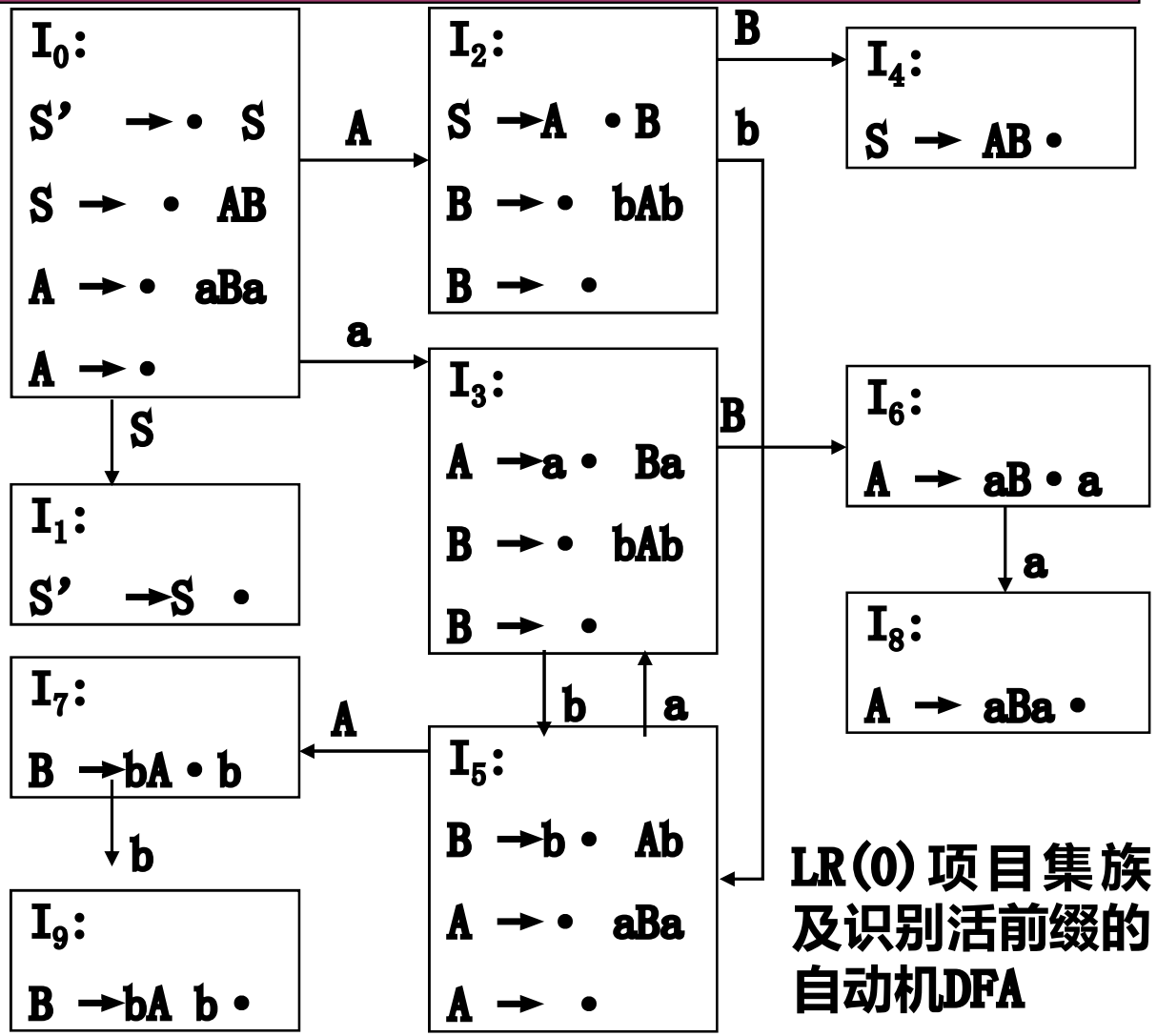
(5) B → ε

Follow(S')= {#}

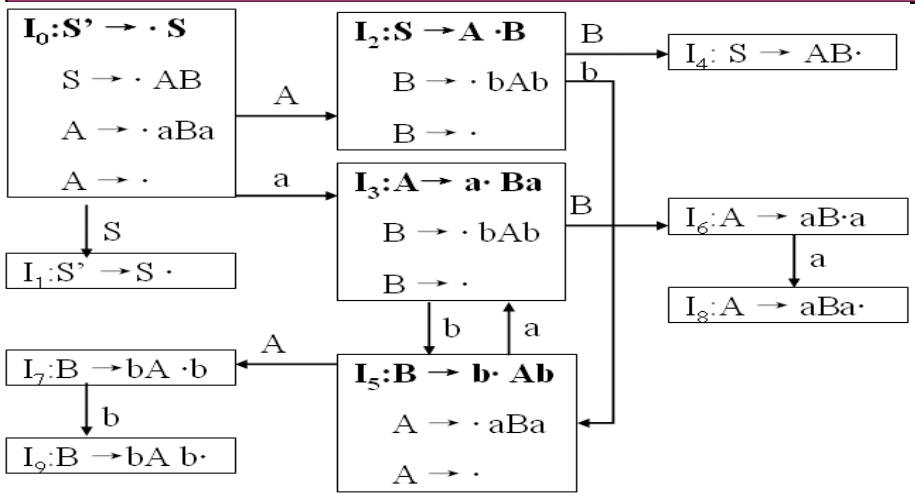
Follow(S)= {#}

Follow(A)= {b, #}

Follow(B)= {a, #}



LR(0)项目集族及识别活前缀的自动机DFA



(0) S' → S

(1) S → AB

(2) A → aBa

(3) A → ε

(4) B → bAb

(5) B→ ε

Follow(S')= {#}

Follow(S)= {#}

Follow(A)= {b, #}

Follow(B)= {a, #}

SLR(1)分析表

状态	ACTION			GOTO		
	a	b	#	S	A	B
0	S3	r3	r3	1	2	
1			acc			
2	R5	S5	r5			4
3	R5	S5	r5			6
4			r1			
5	S3	r3	r3		7	
6	S8					
7		S9				
8		r2	r2			
9	R4		r4			

SLR(1)分析举例（输入串baab#）

步骤	状态栈	符号栈	输 入 串	动 作
1	0	#	baab#	r3:A → ε GOTO 2
2	02	#A	baab#	S5:
3	025	#Ab	aab#	S3
4	0253	#Aba	ab#	r5: B → ε GOTO 6
5	02536	#AbaB	ab#	S8:
6	025368	#AbaBa	b#	r2:A → aBa GOTO 7
7	0257	#AbA	b#	S9:
8	02579	#AbAb	#	r4:B → bAb GOTO 4
9	024	#AB	#	r1: S → AB GOTO 1
10	01	#S	#	

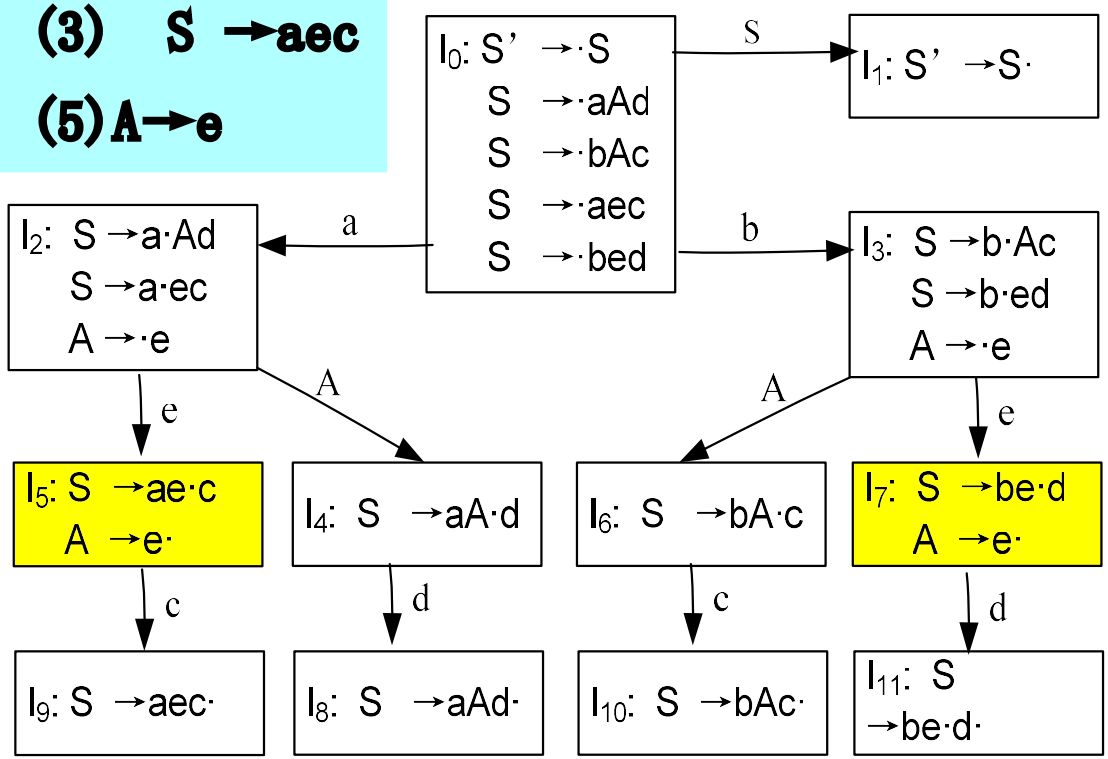
6.4 LR(1)分析

文法G'：(0) S'→S (1) S→aAd

(2) S→bAc (3) S→aec

(4) S→bed (5)A→e

I_5中，如果将e归约成A，得到错误句型aAc，或者说前缀为ae的句型中，遇到c时只能移进。



I5: FOLLOW (A) ∩ {c}={c, d} ∩ {c}={c}

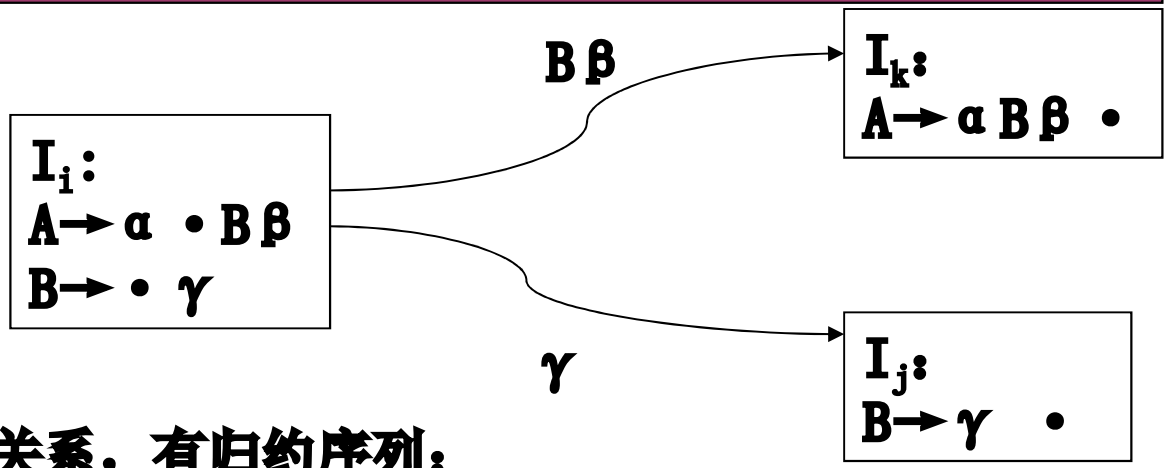
I7: FOLLOW (A) ∩ {d}={c, d} ∩ {d}={d}

一般情况：

若：$A \rightarrow \alpha \cdot B\beta \in I_i$

则：$B \rightarrow \cdot \gamma \in I_i$

对应DFA：



状态I_j和状态I_k的关系，有归约序列：

$$a_1 \cdots\cdots a_n \underset{R}{\overset{*}{\Leftarrow}} \delta\ \alpha\ \gamma\ a_p \cdots\cdots a_n \underset{R}{\overset{*}{\Leftarrow}} \delta\ \alpha\ B\ a_p \cdots\cdots a_n$$

$$\underset{R}{\overset{*}{\Leftarrow}} \delta\ \alpha\ B\ \beta\ a_q \cdots\cdots a_n$$

在状态I_j处SLR（1）直接按搜索符是否属于B的FOLLOW（B）集合来确定是否归约，而I_i的项目$A \rightarrow \alpha \cdot B\beta$表示要接收$B\beta$到达状态$I_k$，这样当输入栈当前符$a_p$是属于FOLLOW（B）但不属于FIRST（$\beta$）时，就会在状态$I_j$进行一次错误的归约。不可能由上面的第三个规范句型归约到第四个规范句型。

LR(1)项目& MOVE运算

定义 7.9　文法的附加搜索符($\in V_t \cup \{\#\}$)的LR(0)项目称为LR(1)项目。记为[LR(0)项目，搜索符]。LR(1)项目中LR(0)项目部分称为LR(1)项目的心。对于同心的LR(1)项目简记为 [LR(0)项目，搜索符$_1$|搜索符$_2$|・・・|搜索符$_m$]。“搜索符$_1$|搜索符$_2$|・・・|搜索符$_m$”称为搜索集。

定义 7.10　设I是文法G的LR(1)项目子集，则Move1(I，X)定义如下：　Move1(I，X) ＝ {[A→αX・β,a]|[A→α・Xβ,a]∈I}

定义 7.11　设I是文法G的LR(1)项目子集，closure1(I)定义如下：

(1) I ⊂ closure1(I)

(2) {[B→・γ,b]|[A→α・Bβ,a]∈closure1(I),

b∈FIRST(βa)} ⊂ closure1(I)

(3) 重复(2)，直到closure1(I)，不再扩大为止。

LR(1)识别活前缀DFA M构造方法

设文法G=(V_N, V_T, P, S)，等价改写成文法G′：G′ = (V_N∪{S′}, V_T, P∪{S′ →S}, S′)，其中V_N∪{S′}=Φ，则识别活前缀DFA M=(K,Σ, f,S,Z)，其中

(1) K ⊆ ρ (LR(1)项目集)

(2) Σ = V_N∪V_T

(3) f(I,X) = closure(Move1(I, X)),I∈ K, X∈Σ

(4) S = closure1([S′ →·S,#])

(5) Z = {q | q∈K, q 含有归约项目}

定义 7.12 文法G的LR(1)识别活前缀DFA M的状态集称为文法G的LR(1)项目集规范族。

LR(1)分析表构造方法

设文法G的LR(1)项目集规范族C＝{ I_0，I_1，・・・，I_n}，且f为转换函数，则

(1) 对每一个LR(0)项目，依据下列情况分别填分析表：

如果移进项目[A→α・ aβ,b]∈I_k，f(I_k，a)＝I_j，则

置M.ACTION[k,a]为S_j；

如果归约项目[A→α・,b]∈ I_k，A→α标号为i，则

置M.ACTION[k,b]为r_i；

如果接受项目[S′ → S・,#]∈I_k，则

置M.ACTION[k,#]为acc；

如果f(I_k,A)＝I_j，A∈V_N，则

置M.GOTO[k,A]为j；

(2) 凡(1)没能填入分析表元素M.ACTION[k,a]和M.GOTO[k,a]

置为e_t（t为错误编号）。

LR(1)分析表构造举例

例7.4 设文法G[S′]定义如右，试构造LR(1)分析表M。

G[S′]：
(0) S′→S
(1) S→aAd
(2) S→bAc
(3) S→aec
(4) S→bed
(5) A→e

构造识别LR(1)活前缀DFA和LR(1)分析表之过程演示：

$V_T \cup V_N$ A/G 状态	ACTION						GOTO	
	a	b	c	d	e	#	S	A
0	S_2	S_3					1	
1						acc		
2					S_5			4
3					S_7			6
4				S_8				
5			S_9	r_5				
6			S_{10}					
7			r_5	S_{11}				
8						r_1		
9						r_3		
10					r_2			
11				r_4				

LR(1)文法的定义

定义 7.13　设文法G的LR(1)项目集规范族C中任意含有m个移进项目和 n个归约项目的冲突项目集 I_k 的一般形式为

$$I_k=\{[A_1\rightarrow\alpha_1\cdot a_1\beta_1,S'_1],\cdots,[A_m\rightarrow\alpha_m\cdot a_m\beta_m,S'_m],$$
$$[B_1\rightarrow\gamma_2\cdot,S_1],\cdots,[B_n\rightarrow\gamma_n\cdot,S_n],\cdots\}$$

(A_i、$B_j\in V_N$，$a_i\in V_T$，α_i、βj、$\gamma_j\in(V_N\cup V_T)^*$，$S'_i$、$S_j$为搜索集，最后的$\cdots$表示剩下的待约项目)。

如果移进符号集$\{a_1,a_2,\cdots,\alpha_m\}$和搜索集$S_1$、$S_2$、$\cdots$、$S_n$两两相交均为空集，则文法G称为**LR(1)文法**。

关于LR(1)文法，可以得出下列几个结论。

(1) 如果文法G是LR(1)文法，则G可采用LR(1)分析法。

(2) 如果文法G是LR(1)文法，则G是无二义性的。

(3) 如果文法G是SLR(1)文法，则G一定是LR(1) 。

6.5 LALR(1)分析

定义 7.14 如果采用同心项目集合并方法，进行合并后的文法G的LR(1)项目集规范族，没有LR(1)项目冲突，则称文法G为**LALR(1)文法**。

关于LALR(1)文法，可以得出下列几个结论。

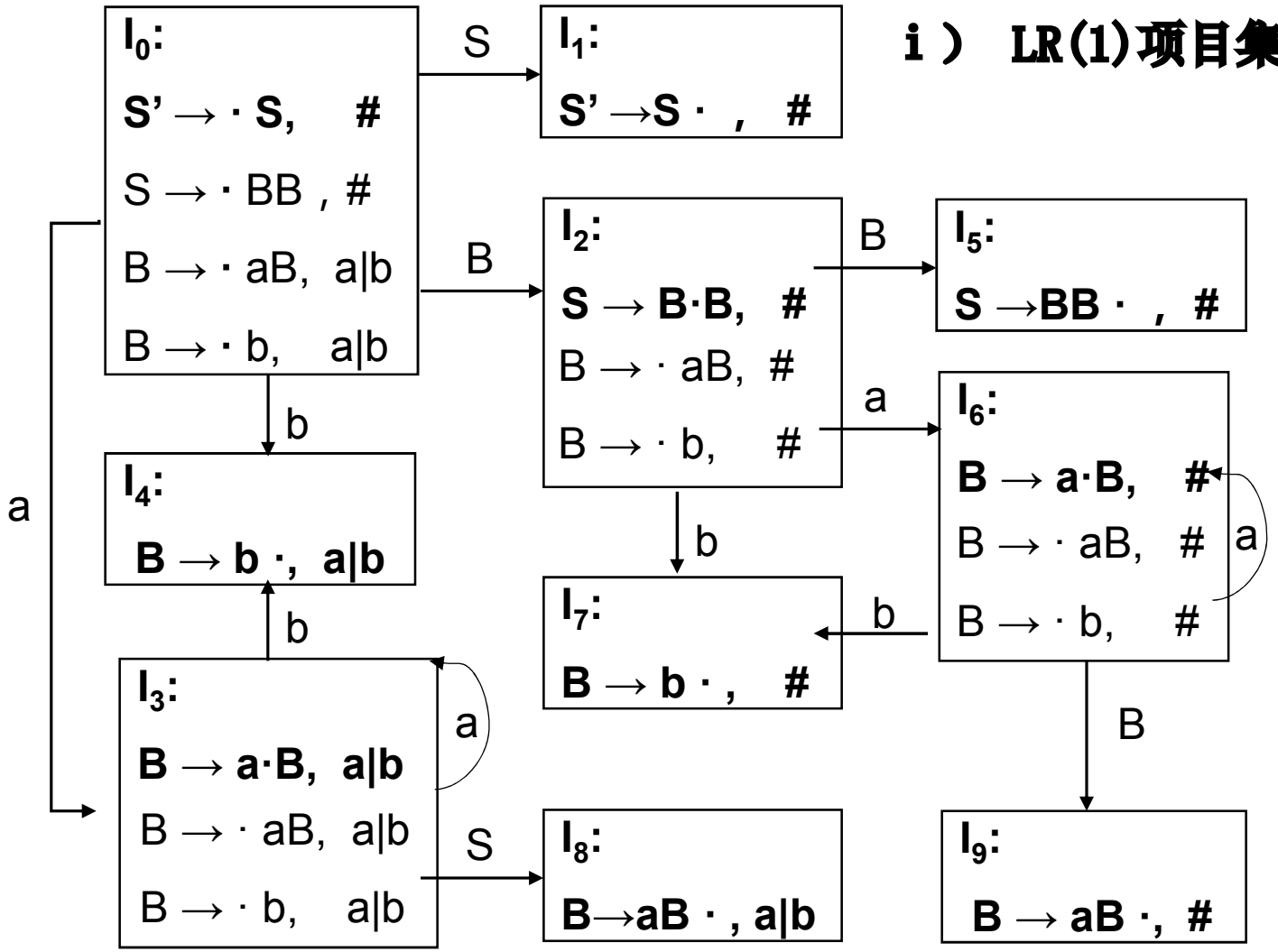
⑴如果文法G是LALR(1)文法，则G可采用LALR(1)分析法。

⑵如果文法G是LALR(1)文法，则G是无二义性的。

⑶如果文法G是LALR(1)文法，则G一定是LR(1)。

例7.5 设文法G[S′]定义如下，试构造LALR(1)分析表M。

G[S′]：（0） S′→S
（1） S→BB
（2） B→aB
（3） B→b



i） LR(1)项目集族和转换函数

拓广文法:

(0) S' → S

(1) S → BB

(2) B → aB

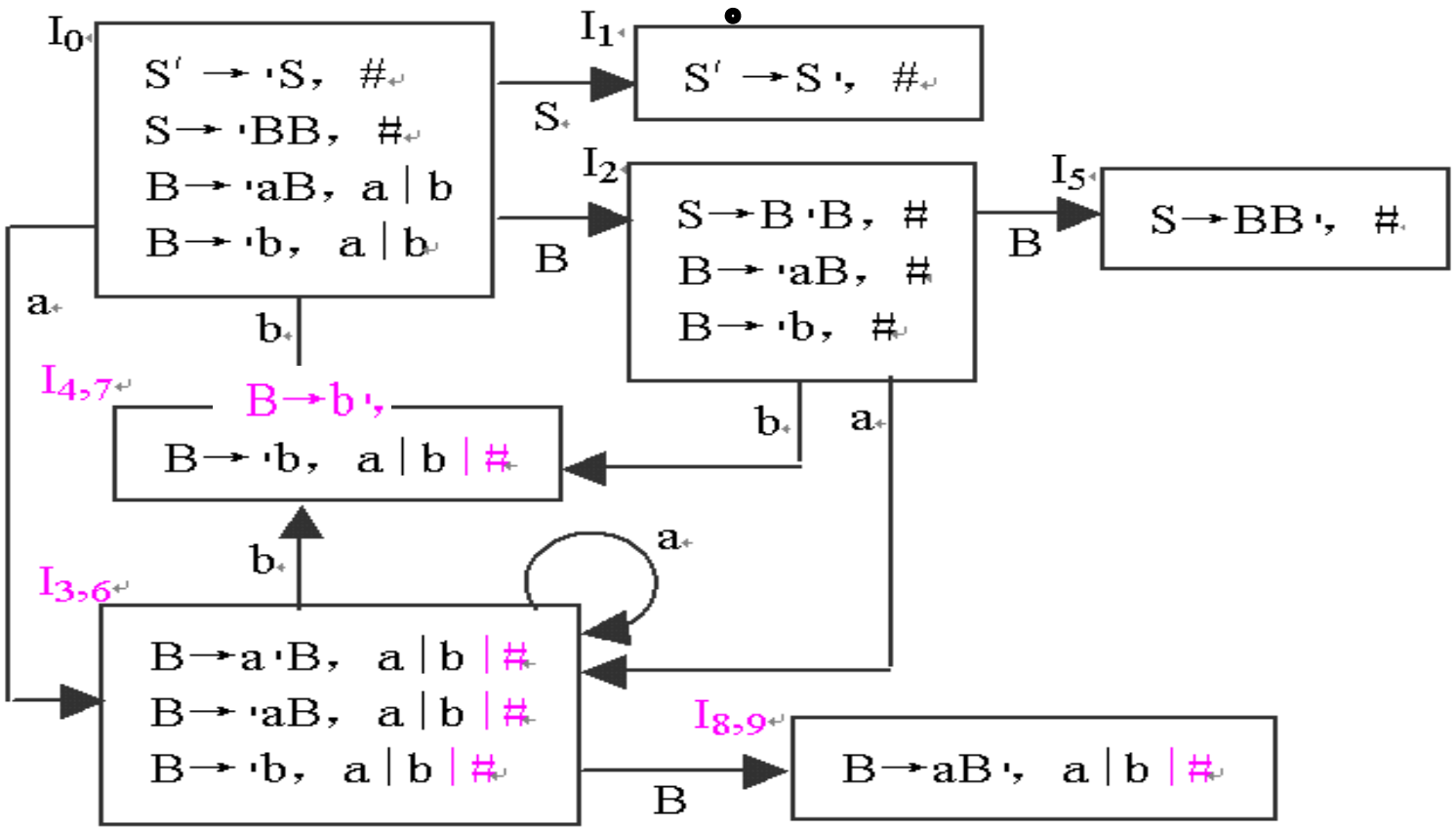
(3) B → b

I_3与I_6同心集

I_4与I_7同心集

I_8与I_9同心集

ii）同心项目集I_3 和I_6、I_4 和I_7 、I_8 和I_9合并后分别用$I_{3,6}$、$I_{4,7}$和$I_{8,9}$表示，得识别活前缀DFA M′如下，没有新冲突情况出现



iii）构造文法G[S']的LALR(1)分析表M如下。

$V_T \cup V_N$ A/G 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		

(0) S' → S

(1) S → L=R

(2) S → R

(3) L → *R

(4) L → i

(5) R → L

FOLLOW(R)

= { # , = }

非LR(0),非SLR(1)

I_0: S' → ·S

S → · L=R

S → · R

L → · *R

L → · i

R → · L

I_1: S' → S ·

I_2: S → L · =R

R → L·

I_3: S → R ·

I_4: L → * · R

R → · L

L → · *R

L → · i

I_5: L → i ·

I_6: S → L= · R

R → · L

L → · *R

L → · i

I_7: L → *R ·

I_8: R → L ·

I_9: S → L=R ·

LR（0）项目及规范族

LR（1）项目及规范族

I_0: S' →•S #
S →•L=R #
S →•R #
L →•*R =|#
L →•I =|#
R →•L #

I_1: S' →S• #

I_2: S →L• =R #
R →L• #

I_3: S →R• #

I_4: L →*•R =|#
R →•L =|#
L →•*R =|#
L →•i =|#

I_5: L →i• =|#

I_6: S →L=•R #
R →•L #
L →•*R #
L →•I #

I_8: R →L• =|#

I_9: S →L=R• #

I_{10}: R →L• #

I_{11}: L →*•R #
R →•L #
L →•*R #
L →•I #

I_{12}: L →i• #

• #

是LR(1)文法，合并同心集无冲突，
也是LALR（1）文法

LR（1）项目及规范族

(0) S' → S
(1) S → aAd
(2) S → bBd
(3) S → aBe
(4) S → bAe
(5) S → c
(6) S → c

I_0: S' → •S #
S → •aAd #
S → •bBd #
S → •aBe #
S → •bAe #

I_1: S' → S• #

I_2: S → a•Ad #
S → a•Be #
A → •c #
B → •c #

I_3: S → b•Bd #
S → b•Ae #
B → •c #

I_4: S → a A•d #

I_5: S → a B•e #

I_6: A → c • d
B → c • e

I_7: S → bB•d #

I_8: S → b A•e #

I_9: A → c • e
B → c • d

I_{10}: S → a Ad• #

I_{11}: S → a B e• #

I_{12}: S → b B d• #

I_{13}: S → b A e• #

是LR(1)文法，合并同心集有冲突，不是LALR（1）文法

6.6 二义性文法的应用

任何一个二义性文法决不是LR类文法，与其相应的LR分析表一定含有多重定义的元素。但是对某些二义性文法，在含多重定义的LR分析表中加进足够的无二义性规则，从而可以构造出比相应非二义性文法更优越的LR分析器。

例如，考虑算术表达式的二义性文法

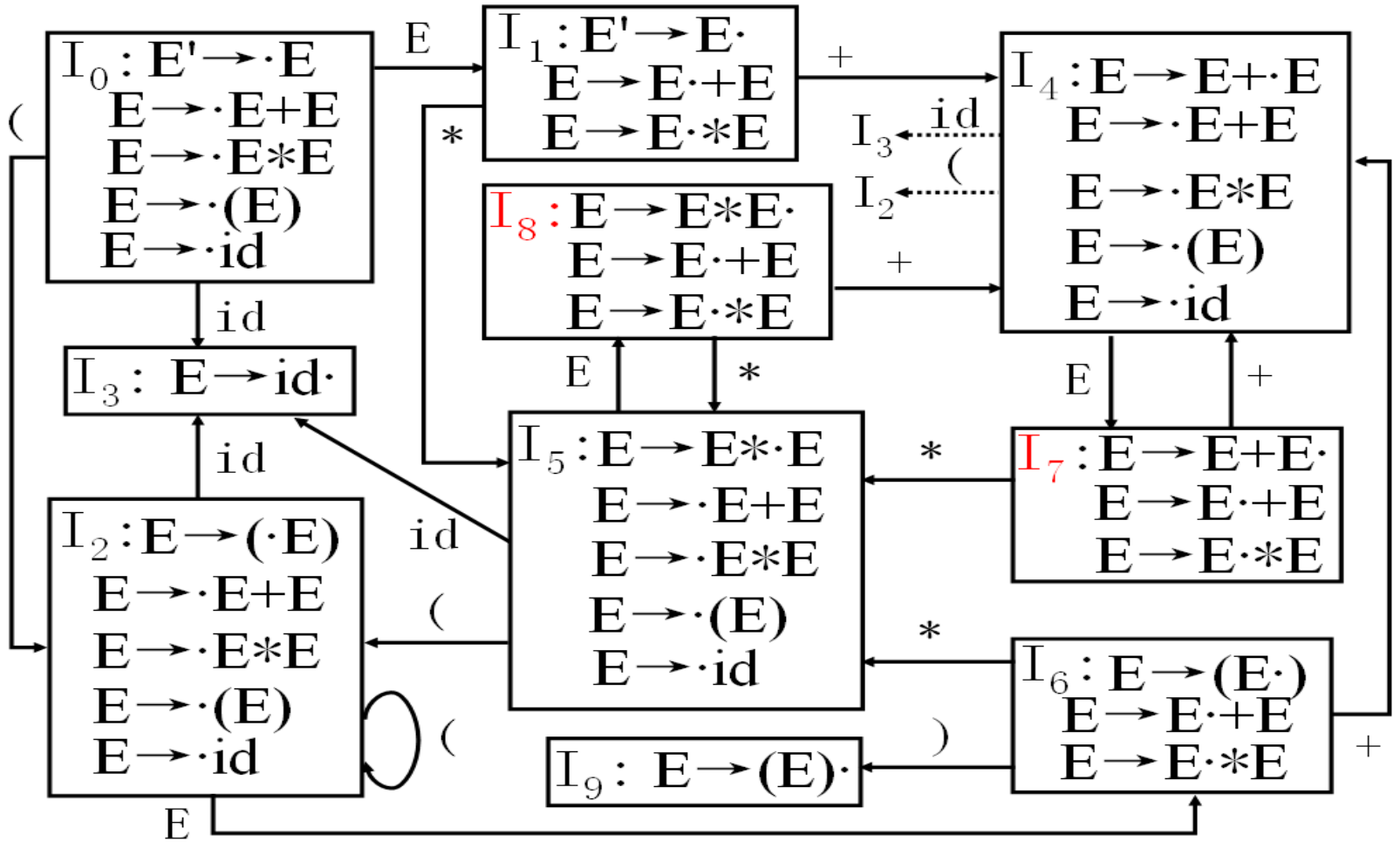
$$E \rightarrow E+E \mid E*E \mid (E) \mid id$$

相应的非二义性文法为：

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

$$T \rightarrow T*F \mid F$$

$$F \rightarrow (E) \mid id$$


I0 : E'→·E
E→·E+E
E→·E*E
E→·(E)
E→·id
E
I1 : E'→E·
E→E·+E
E→E·*E
+
I4 : E→E+·E
E→·E+E
E→·E*E
E→·(E)
E→·id
id
I3
(
I2
I8 : E→E*E·
E→E·+E
E→E·*E
+
E
*
I3 : E→id·
id
I5 : E→E*·E
E→·E+E
E→·E*E
E→·(E)
E→·id
*
I7 : E→E+E·
E→E·+E
E→E·*E
I2 : E→(·E)
E→·E+E
E→·E*E
E→·(E)
E→·id
id
(
(
*
I6 : E→(E·)
E→E·+E
E→E·*E
)
I9 : E→(E)·
E
+

I_1：移进—归约冲突，I_1中冲突可用SLR(1)方法解决。因为FOLLOW(E ‘)∩{+,*}=Φ，即遇到输入符号为‘#’时则接受，遇到‘＋’或‘*’时则移进。

对I7和I8而有：FOLLOW(E)∩{+,*}={#,+,*,)}∩{+,*}≠Φ

因而I7和I8中冲突不能用SLR(1)方法解决，也不能用其它LR(K)方法解决，但是我们用+，*的优先级和结合性可以解决这类冲突。

I7：由于‘*’优先级高于‘＋’，所以状态7面临‘*’移进，又因‘＋’服从左结合，所以状态7面临‘＋’则用 E→E＋E归约。

I8：由于‘*’优先于‘＋’且‘*’服从左结合，因此状态8面临‘＋’或‘*’都应用E→E*E归约。

表 7.17 对二义性表达式文法的 LR 分析表

状态	ACTION						GOTO
	+	*	(	)	i	#	E
0			S_2		S_3		1
1	S_4	S_5				acc	
2			S_2		S_8		6
3	r_4	r_4		r_4		r_4	
4			S_2		S_3		7
5			S_2		S_3		8
6	S_4	S_5		S_9			
7	r_1	S_5		r_1		r_1	
8	r_2	r_2		r_2		r_2	
9	r_3	r_3		r_3		r_3	

本章研究自底向上的LR分析法，LR分析法是一类归约法的统称，主要介绍其中的、也是最基本的LR(0)、SLR(1)、LR(1)和LALR(1)四种分析法，重点讨论可归约前缀的作用、识别活前缀DFA的构造、分析表的构造、分析法适用条件和语法分析程序结构及其分析算法。

提出的基本概念是可归前缀、活前缀、LR(0)项目、移进项目、待约项目、归约项目、接受项目、移进-归约冲突的项目集、归约-归约冲突的项目集、LR(0)项目集、LR(0)项目集规范族、LR(0)文法、SLR(1)文法、搜索符、搜索集、LR(1)文法、LR(1)项目集规范族、同心项目、同心项目集和LALR(1)文法。

LR(0)分析方法、SLR(1)分析方法、LR(1)分析方法和LALR(1)分析方法，构造语法分析程序，其语法分析算法是一致的、通用的。

本 章 小 结

采用LR(0)分析方法构造语法分析程序的技术线路是：依据给定的源语言，设计其上下文无关文法，并构造识别LR(0)活前缀DFA，判定文法是否是LR(0)文法；如果LR(0)文法，则根据LR(0)活前缀DFA，构造LR(0)分析表。

采用SLR(1)分析方法构造语法分析程序的技术线路是：依据给定的源语言，设计其上下文无关文法，并构造识别LR(0)活前缀DFA以及FOLLOW集，判定文法是否是SLR(1)文法；如果是SLR(1)文法，则根据LR(0)活前缀DFA以及FOLLOW集，构造SLR(1)分析表。

采用LR(1)分析方法构造语法分析程序的技术线路是：依据给定的源语言，设计其上下文无关文法，并构造识别LR(1)活前缀DFA，判定文法是否是LR(1)文法；如果是LR(1)文法，则根据LR(1)活前缀DFA，构造LR(1)分析表。

本 章 小 结

采用LALR(1)分析方法构造语法分析程序的技术线路是：依据给定的源语言，设计其上下文无关文法，并构造识别LR(1)活前缀DFA，判定文法是否是LR(1)文法；如果是LR(1)文法，则根据LR(1)活前缀DFA，合并同心项目集后，判定文法是否是LALR(1)文法；如果是LALR(1)文法，则构造LALR(1)分析表。

如果$S_{LR(0)}$、$S_{SLR(1)}$、$S_{LALR(1)}$和$S_{LR(1)}$分别表示LR(0)文法集、SLR(1) 文法集、LALR(1) 文法集和LR(1)文法集，则$S_{LR(0)} \subsetneq S_{SLR(1)} \subsetneq S_{LALR(1)} \subsetneq S_{LR(1)}$。

重点掌握的内容是：

①构造识别LR(0)活前缀DFA、构造识别LR(1)活前缀DFA和合并识别LR(1)活前缀DFA的同心项目集；

②LR(0)、SLR(1)、LR(1)和LALR(1)文法判别；

③LR(0)、SLR(1)、LR(1)和LALR(1)文法优先分析表；

④LR分析算法。