编译原理

第五章　语法分析——自下而上分析

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

语法分析的方法

- 自下而上分析法 (Bottom-up)
  - 基本思想
    - 从输入串开始，逐步进行归约，直到文法的开始符号
    - 从树末端开始，构造语法树
    - 核心问题：确定可归约串
  - 算符优先分析法
    - 按照算符的优先关系和结合性质进行语法分析
    - 适合分析表达式
  - LR 分析法
    - 规范归约：句柄作为可归约串

5.3 LR 分析法

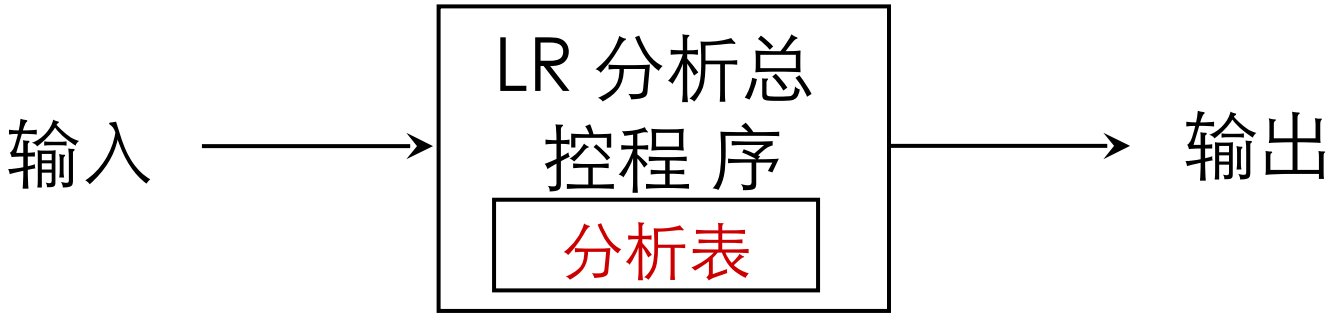
- 计算思维的典型方法
  - 知识与控制的分离
  - 自动化

- LR 分析法： 1965 年 由 Knuth 提出

Donald Ervin Knuth

For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to " the art of computer programming" through his well-known books in a continuous series by this title.

- LR 分析器工作



主要介绍

1. 总控程序 (LR 分析器) 的处理思想
2. LR 分析表的构造方法及原理

短语、直接短语和句柄

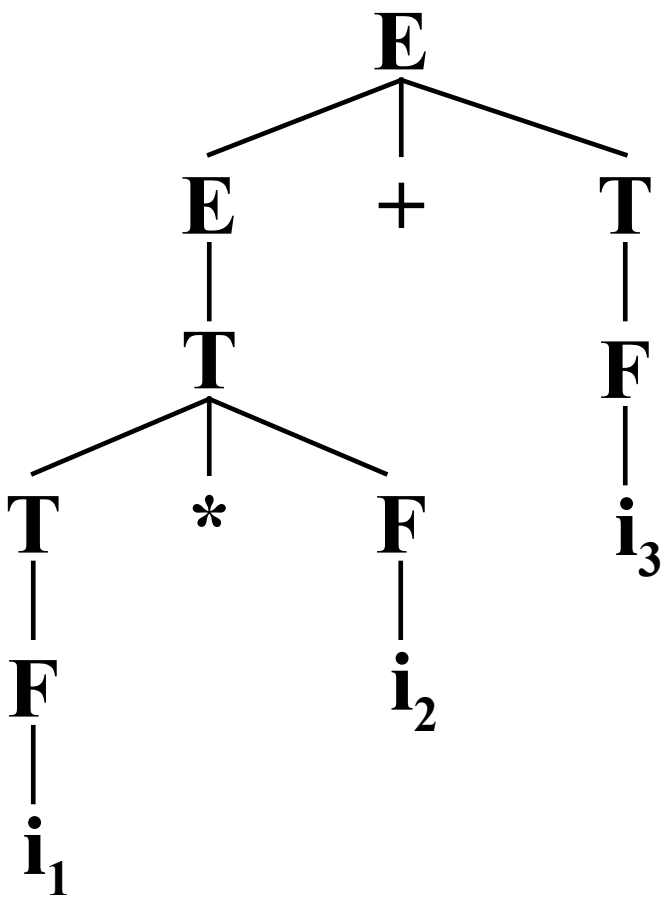
- 定义：令 G 是一个文法， S 是文法的开始符号，假定αβδ是文法 G 的一个句型，如果有 $S \overset{*}{\Rightarrow} \alpha A \delta$ 且 $A \overset{+}{\Rightarrow} \beta$

则β称是句型αβδ相对于非终结符 A 的短语。

特别是，如果有 A⇒β, 则称β是句型αβδ相对于规则 A→ β 的直接短语。

一个句型的最左直接短语称为该句型的句柄。

短语、直接短语和句柄



- 在一个句型对应的语法树中
  - 以某非终结符为根的两代以上的子树的所有末端结点从左到右排列就是相对于该非终结符的一个短语
  - 如果子树只有两代，则该短语就是直接短语

规范归约

- 定义：假定α是文法 G 的一个句子，我们称序列

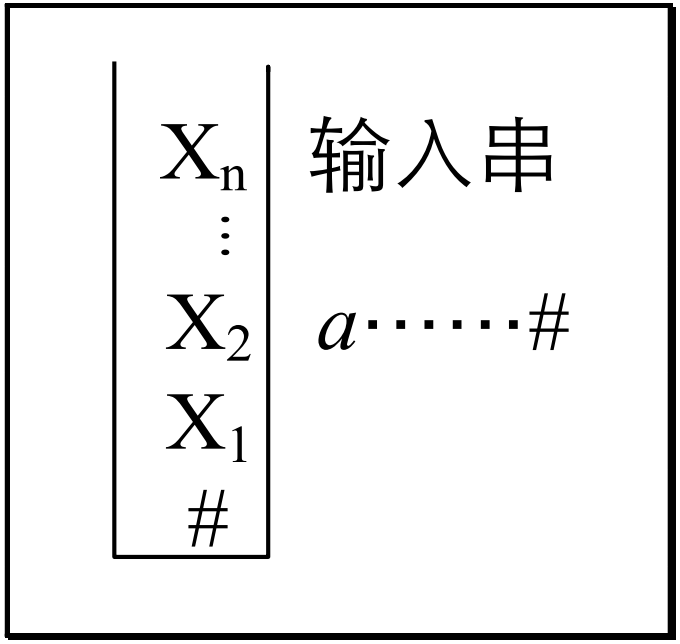
  α_n ， α_{n-1} ， … ， α_0

  是α的一个规范归约，如果此序列满足：

1. $\alpha_n = \alpha$
2. α_0 为文法的开始符号，即$\alpha_0 = S$
3. 对任何 i ， $0 \le i \le n$ ， α_{i-1} 是从α_i 经把句柄替换成为相应产生式左部符号而得到的

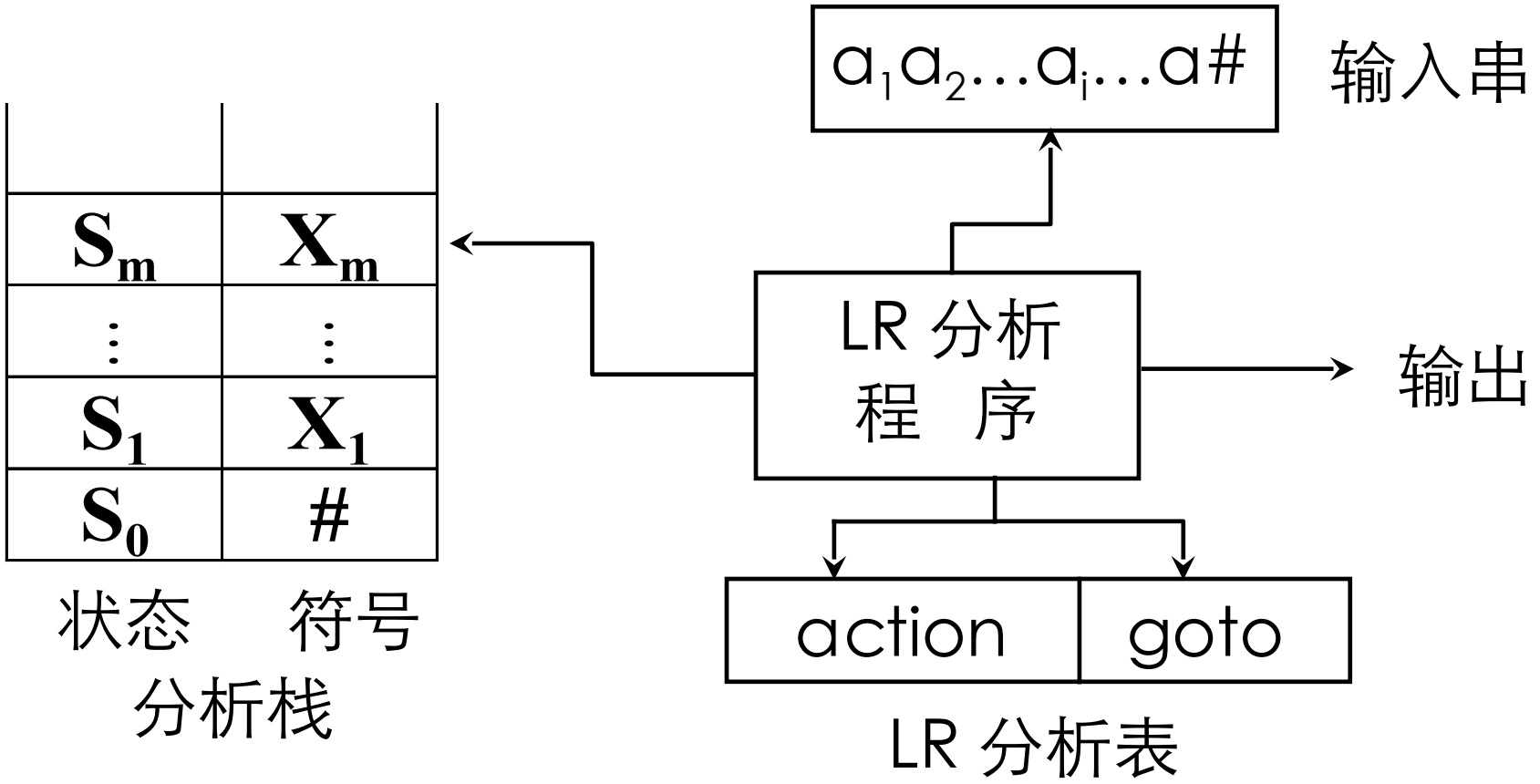
5.3.1 LR 分析器

- 规范归约的关键问题是寻找句柄 .
  - 历史：已移入符号栈的内容
  - 展望：根据产生式推测未来可能遇到的输入符号
  - 现实：当前的输入符号



- 计算思维的典型方法
  - 知识与控制的分离
  - 自动化

- LR 分析方法：把"历
  抽象成状态；由栈顶的状态和先行的输入
  符号唯一确定每一步工作



LR 分析器

- LR 分析器的核心是一张分析表
  - ACTION[s， a]： 当状态 s 面临输入符号 a 时，应采取什么动作 .
  - GOTO[s， X]： 状态 s 面对文法符号 X 时，下一状态是什么

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3

LR 分析器

移进 (shift)　把 (*s*，　*a*) 的下一状态 s’　和输入符号 *a* 推进栈，下一输入符号变成现行输入符号

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

LR 分析器

归约 (reduce)： 指用某产生式 $A \rightarrow \beta$ 进行归约． 假若β的长度为 r， 归约动作是， 去除栈顶 r 个项，使状态 s_{m-r} 变成栈顶状态，然后把 (s_{m-r}, A) 的下一状态 $s'=GOTO[s_{m-r}, A]$ 和文法符号 A 推进栈

状态	ACTION						GOTO		
	i	+	*	(		#	E	T	F
0	s5						1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

LR 分析器

接受 宣布分析成功，停止分析器工作

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

LR 分析器

报错

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

LR 分析器

- 分析开始时：

  状态　　已归约串　　输入串

  $(s_0$ ，　　# ，　　$a_1a_2 \ldots a_n$ #)

- 以后每步的结果可以表示为：

$(s_0\ s_1 \ldots s_m$ ，　# $X_1 \ldots X_m$ ，　$a_ia_{i+1} \ldots a_n$ #)

$(s_0\ s_1 \ldots s_m\ ,\ \#\ X_1 \ldots X_m\ ,\ a_i a_{i+1} \ldots a_n\ \#)$

$(s_0\ s_1 \ldots s_{m-r}\ s_{m-r+1} \cdots s_m\ ,\ \#\ X_1 \ldots X_{m-r}\ X_{m-r+1} \cdots X_m\ ,\ a_i a_{i+1} \ldots a_n\ \#)$

$(s_0\ s_1 \ldots s_{m-r}\ ,\ \#\ X_1 \ldots X_{m-r}\ ,\ a_i a_{i+1} \ldots a_n\ \#)$

$(s_0\ s_1 \ldots s_{m-r}\ ,\ \#\ X_1 \ldots X_{m-r} A\ ,\ a_i a_{i+1} \ldots a_n\ \#)$

$(s_0\ s_1 \ldots s_{m-r} s\ ,\ \#\ X_1 \ldots X_{m-r} A\ ,\ a_i a_{i+1} \ldots a_n\ \#)$

此处，s=GOTO(s_{m-r}, A), r 为β的长度，$\beta = X_{m-r+1} \cdots X_m$

3. 若 ACTION(s_m , a_i) 为 " 接受 " ，则三元式不再变化，变化过程终止，宣布分析成功 .

4. 若 ACTION(s_m , a_i) 为 " 报错 " ，则三元式变化过程

LR 分析器示例

文法 G(E)：

(1) E→E + T
(2) E→T
(3) T→T*F
(4) T→F
(5) F→(E)
(6) F→i

其 LR 分析表为:

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

输入串为 i*i+i

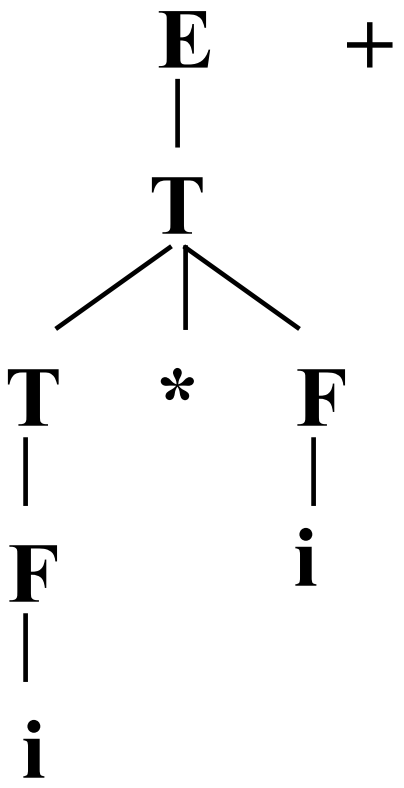
步骤	状态	符号	输入串
(1)	0	#	i*i+i#
(2)	05	#i	*i+i#
(3)	03	#F	*i+i#
(4)	02	#T	*i+i#
(5)	027	#T*	i+i#

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				

T * F i

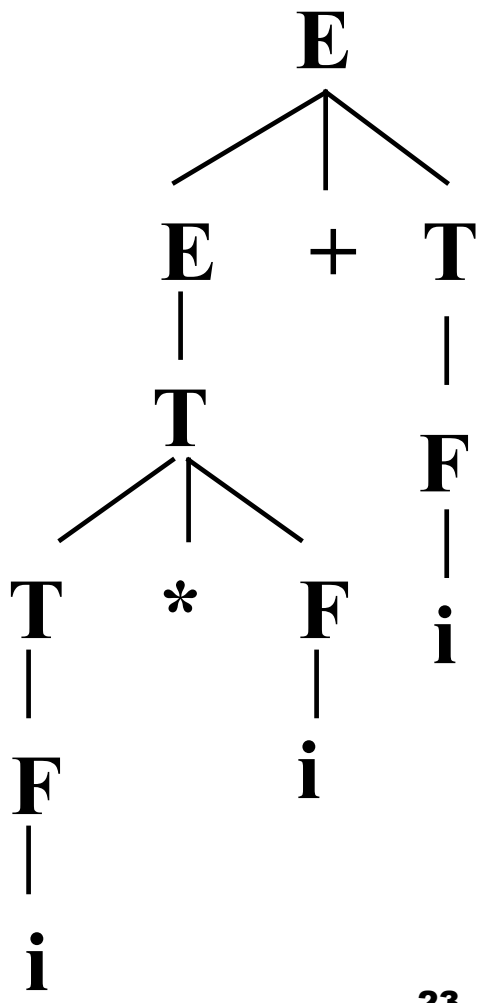
步骤	状态	符号	输入串
(5)	027	#T*	i+i#
(6)	0275	#T*i	+i#
(7)	02710	#T*F	+i#
(8)	02	#T	+i#
(9)	01	#E	+i#
(10)	016	#E+	i#

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			



步骤	状态	符号	输入串
(10)	016	#E+	i#
(11)	0165	#E+i	#
(12)	0163	#E+F	#
(13)	0169	#E+T	#
(14)	01	#E	#
(15)	接受		

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			



LR 文法

- 定义：对于一个文法，如果能够构造一张分析表，使得它的每个入口均是唯一确定的，则这个文法就称为 LR 文法。
- 定义：一个文法，如果能用一个每步顶多向前检查 k 个输入符号的 LR 分析器进行分析，则这个文法就称为 LR(k) 文法 .

LR 文法与二义文法

- LR 文法不是二义的，二义文法肯定不会是 LR 的
  - LR 文法⊂ 无二义文法
- 非 LR 结构

  $S \rightarrow iCtS \mid iCtSeS$

栈	输入
#…iCtS	e…#

小结

- LR 分析器的工作原理
- LR 分析器的性质
  - 栈内的符号串和扫描剩下的输入符号串构成了一个规范句型
  - 一旦栈的顶部出现可归约串 (句柄)，则进行归约

编译原理

第五章　语法分析——自下而上分析

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法
  - LR 分析器的工作原理
  - LR(0) 项目集规范族的构造

5.3.2 LR(0) 项目集族和 LR(0) 分析表的构造

- 假定α是文法 G 的一个句子，我们称序列

  α_n ， α_{n-1} ， … ， α_0

  是的一个规范归约，如果此序列满足:

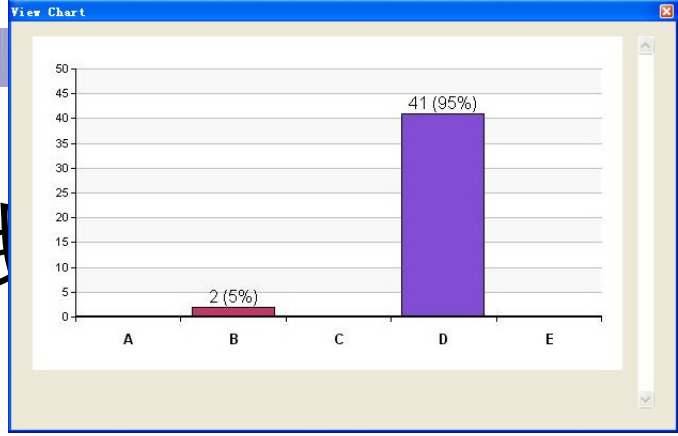
  1 $\alpha_n = \alpha$

  2 α_0 为文法的开始符号，即$\alpha_0 = S$

  3 对任何 i ， $0 \le i \le n$ ， α_{i-1} 是从α_i 经把句柄替换成为相应产生式左部符号而得到的。

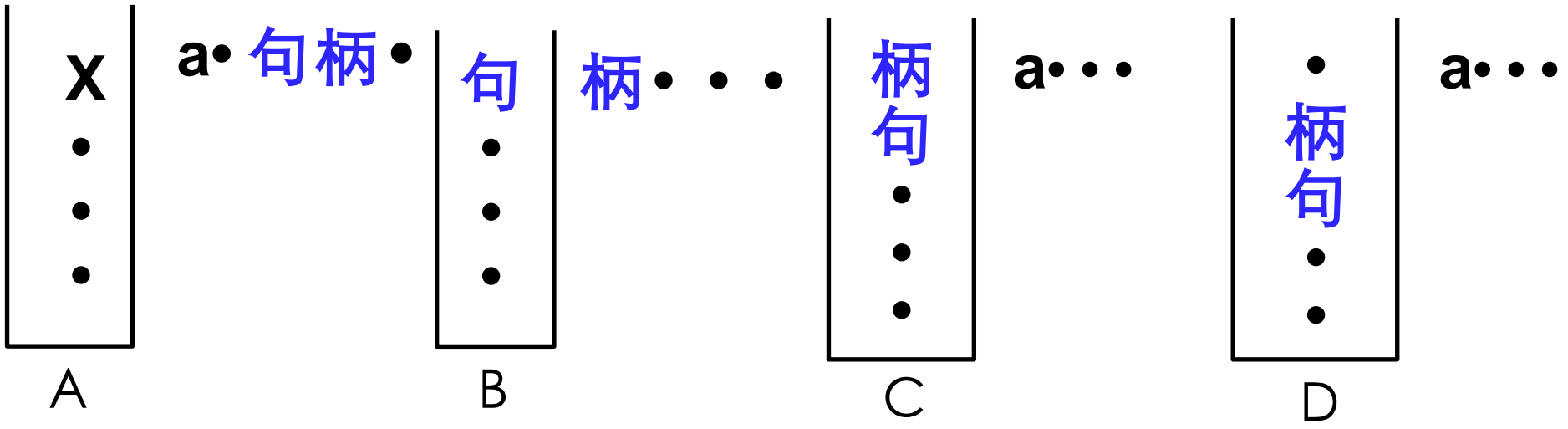
规范归约过程中栈内符号串

- 规范归约过程中
  - 栈内的符号串和扫描剩下的输入符号串构成了一个规范句型



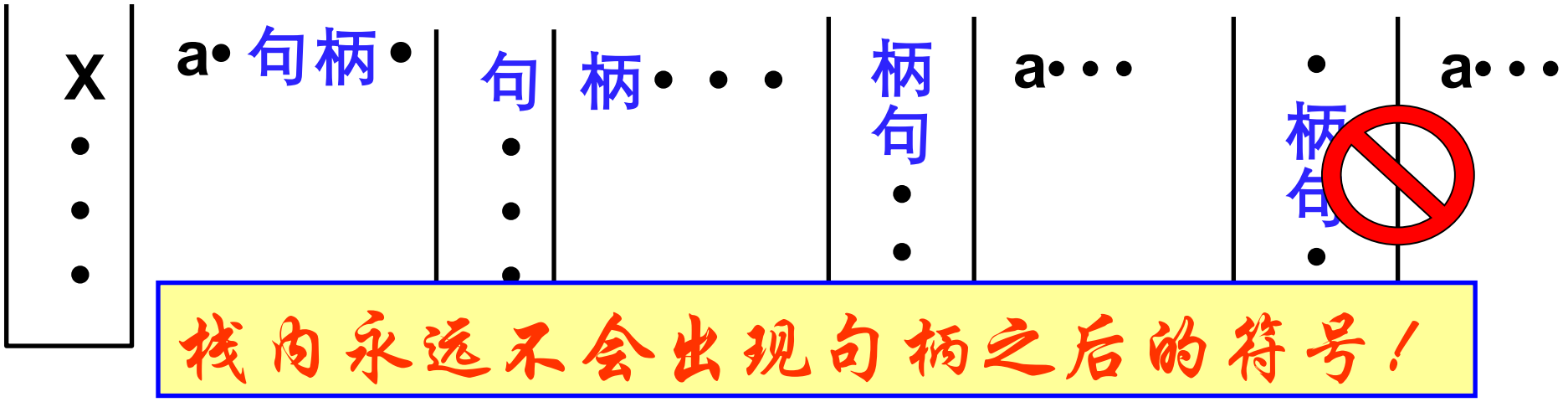
讨论：规范归约过程中栈

- 对于句子，在规范归约过程中，栈内的符号串和扫描剩下的输入符号串构成了一个规范句型，下面哪种格局不会出现：



规范归约过程中栈内符号串

- 规范归约过程中
  - 栈内的符号串和扫描剩下的输入符号串构成了一个规范句型
  - 栈内的如果出现句柄，句柄一定在栈的顶部



字的前缀、活前缀

- 字的前缀：是指字的任意首部，如字 *abc* 的前缀有ε， *a* ， *ab* ， *abc*
- 活前缀：是指规范句型的一个前缀，这种前缀不含句柄之后的任何符号。即，对于规范句型αβδ，β为句柄，如果αβ $=u_1u_2\cdots u_r$ ，则符号串 $u_1u_2\cdots u_i(1\le i\le r)$ 是αβδ的活前缀。（δ必为终结符串）

指导思想——目标驱动

- 踢足球

“如果你不知道怎样踢球，就往球门方向踢”

——施拉普纳

- LR分析

“如果你不知道怎样分析，就保证栈中总是活前缀”

- 哪些字符串是活前缀？
- 能不能构造一个DFA来识别活前缀？

回答：对于一个文法G，可以构造一个DFA，它能识别G的所有活前缀。

字的前缀、活前缀

- 字的前缀：是指字的任意首部，如字 *abc* 的前缀有ε， *a* ， *ab* ， *abc*
- 活前缀：是指规范句型的一个前缀，这种前缀不含句柄之后的任何符号。即，对于规范句型αβδ，β为句柄，如果αβ $=u_1u_2\cdots u_r$，则符号串 $u_1u_2\cdots u_i(1\le i\le r)$ 是αβδ的活前缀。（δ必为终结符串）
- 规范归约过程中，保证分析栈中总是活前缀，就说明分析采取的移进 / 归约动作是正确的

- 文法 G 的每个产生式的右部添加一个圆点称为 G 的 LR(0) 项目
- 如 :$A \rightarrow XYZ$ 有四个项目 :

$A \rightarrow \bullet XYZ$　$A \rightarrow X \bullet YZ$　$A \rightarrow XY \bullet Z$　$A \rightarrow XYZ \bullet$

  - $A \rightarrow \alpha \bullet$ 称为 " 归约项目 "
  - 归约项目 $S' \rightarrow \alpha \bullet$ 称为 " 接受项目 "
  - $A \rightarrow \alpha \bullet a\beta$ $(a \in V_T)$ 称为 " 移进项目 "
  - $A \rightarrow \alpha \bullet B\beta$ $(B \in V_N)$ 称为 " 待约项目 ".
- 项目表示我们在分析过程中看到了产生式多大部分

- 文法 G(S′)

  S′→E
  E→aA|bB
  A→cA|d
  B→cB|d

- 该文法的项目有:

1. S′→ · E 2. S′→E · 3. E→ · aA
4. E→a · A 5. E→aA · 6. A→ · cA
7. A→c · A 8. A→cA · 9. A→ · d
10. A→d · 11. E→ · bB 12. E→b · B
13. E→bB · 14. B→ · cB 15. B→c · B
16. B→cB · 17. B→ · d

- 构造识别文法所有活前缀的 NFA 方法

1. 若状态 i 为 $X \to X_1 \cdots X_{i-1} \bullet X_i \cdots X_n$ ，

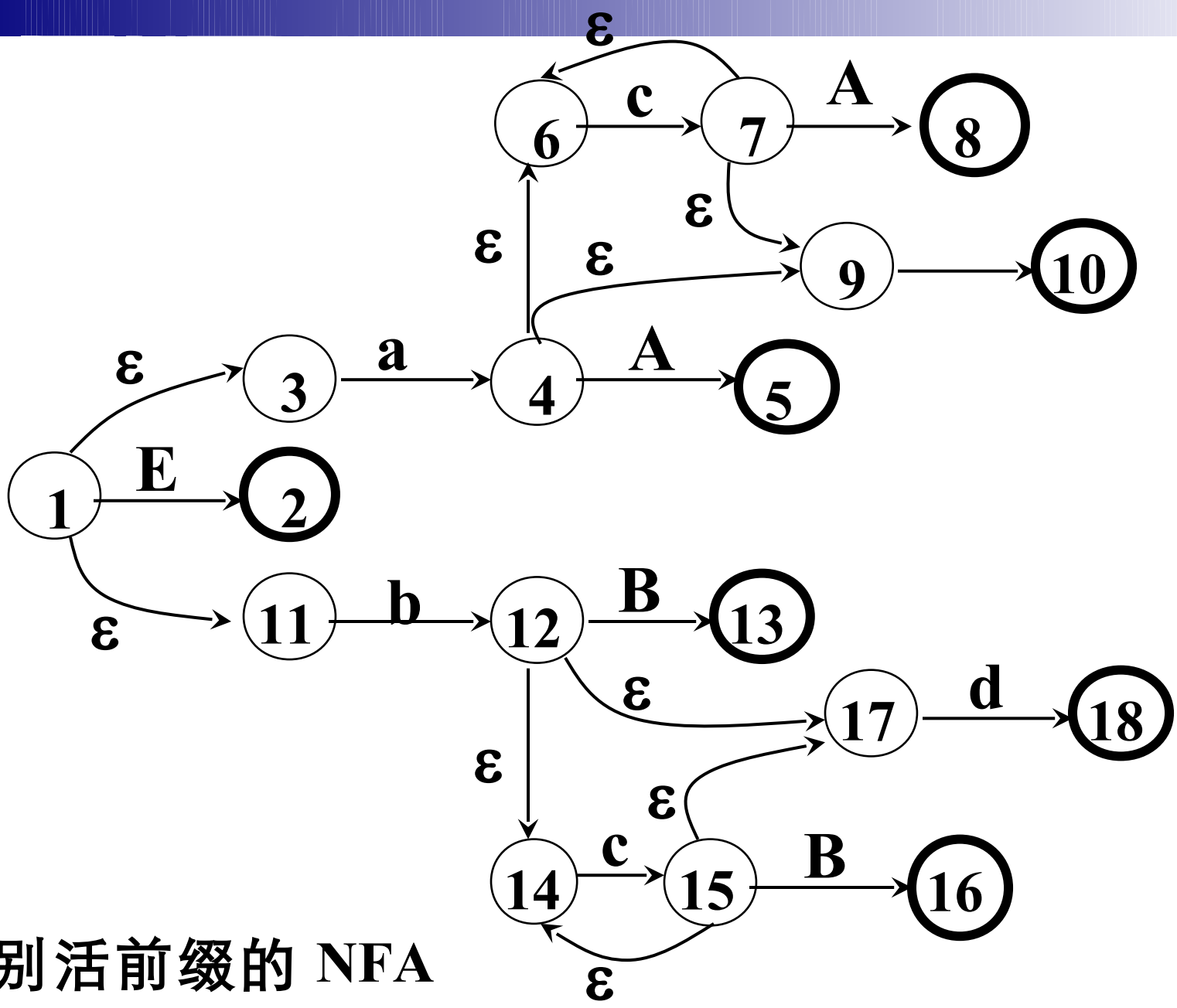
   状态 j 为 $X \to X_1 \cdots X_{i-1} X_i \bullet X_{i+1} \cdots X_n$ ，

则从状态 i 画一条标志为 X_i 的有向边到状态 j ；

2. 若状态 i 为 $X \to \alpha \bullet A\beta$ ， A 为非终结符，

   则从状态 i 画一条ε边到所有状态 $A \to \bullet\gamma$

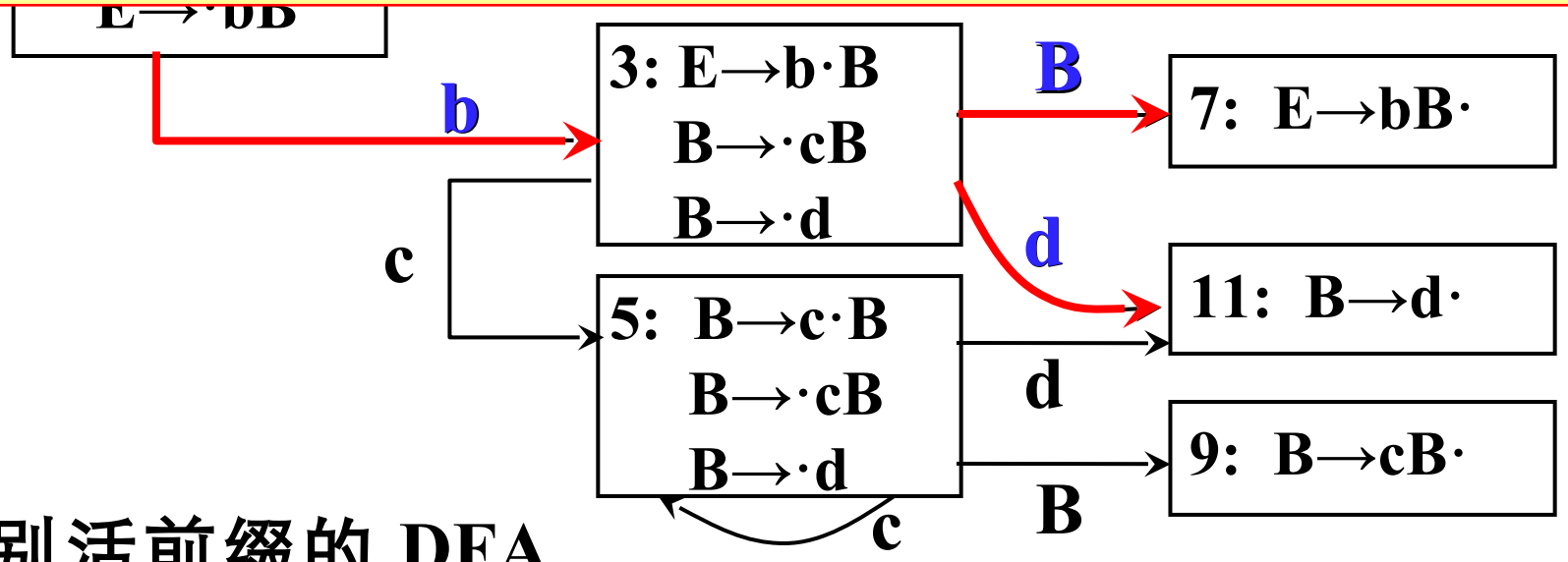
- 把识别文法所有活前缀的 NFA 确定化



识别活前缀的 NFA

对比 DFA ：

```
curState = 初态
GetChar();
while( stateTrans[curState][ch] 有定义 ){
    //存在后继状态，读入、拼接
    Concat();
    //进入下一状态，读入下一字符
   curState= stateTrans[curState][ch];
    if cur_state 是终态 then 返回 strToken 中的单词
    GetChar( );
}
```



识别活前缀的 DFA

LR(0) 项目集规范族

- 构成识别一个文法活前缀的 DFA 的项目集 (状态) 的全体称为文法的 LR(0) 项目集规范族。

有效项目

- 我们

  效的

  - 在任

X_m 的

的初

项目

- 结论：若项目 A→α•Bβ 对活前缀η =δα 是有效的且 B →γ 是一个产生式，则项目 B → •γ 对η =δα 也是有效的。

$$S' \underset{R}{\overset{*}{\Longrightarrow}} \delta A\omega \underset{R}{\Rightarrow} \delta\alpha B\beta\omega$$

设 $\beta\omega \underset{R}{\overset{*}{\Rightarrow}} \varphi\omega$ ，那么

$$S \underset{R}{\overset{*}{\Longrightarrow}} \delta A\omega \underset{R}{\Rightarrow} \delta\alpha B\beta\omega \underset{R}{\overset{*}{\Rightarrow}} \delta\alpha B\varphi\omega \underset{R}{\Rightarrow} \delta\alpha\gamma\varphi\omega$$

所以， B → •γ 对 η =δα 也是有效的。

项目 $A\rightarrow \beta_1 \bullet \beta_2$ 对活前缀$\alpha\beta_1$是有效的，其条件是存在规范推

- 文法 G(S′)

  S′→E

- 考虑

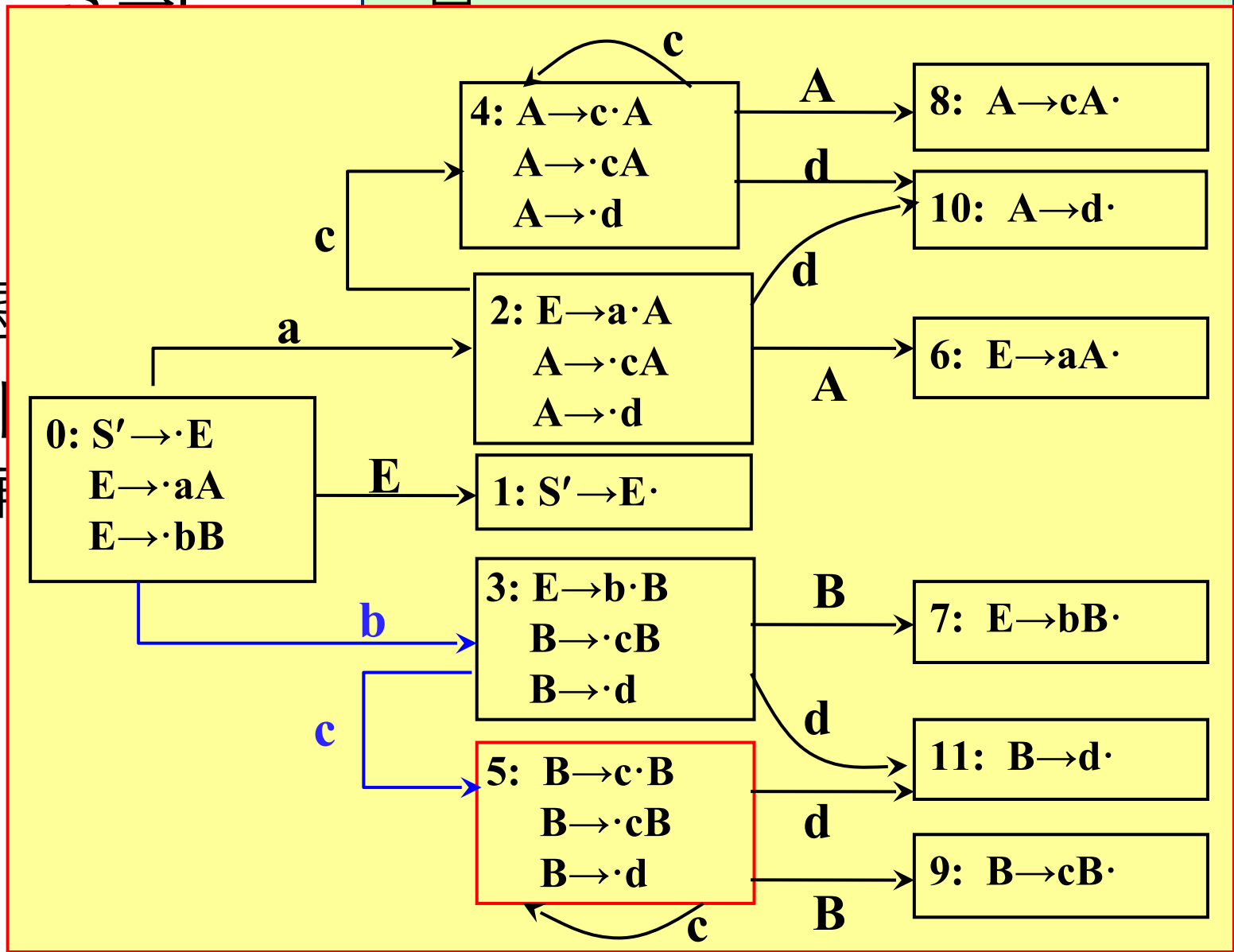
  项

  活

  S’

  S’

  S’



LR(0) 项目集规范族的构造

- 假定文法 G，以 S 为开始符号
- 构造一个 G′，它包含了整个 G，但它引进了一个不出现在 G 中的非终结符 S′，并加进一个新产生式 S′→S，S′ 是 G′ 的开始符号
- 称 G′ 是 G 的拓广文法
- G′ 唯一的“接受”态：仅含项目 S′→S• 的状态

闭包 CLOSURE

- 假定 I 是文法 G′ 的任一项目集，定义和构造 I 的闭包 CLOSURE(I) 如下：

2. 若状态 i 为 X→α•Aβ ， A 为非终结符，
则从状态 i 画一条ε边到所有状态 A→•γ 。

2 若 A→α·Bβ 属于 CLOSURE(I) ，那么

P50： NFA 确定化

设 I 是的状态集的一个子集，定义 I 的ε - 闭包ε -closure(I) 为：

i) 若 s∈I，则 s∈ε-closure(I)；

ii) 若 s∈I，则从 s 出发经过任意条ε弧而能到达的任何状态 s’ 都属于ε -closure(I)

ε-closure(I)=I∪{s’| 从某个 s∈I 出发经过任意条ε弧能到达

- 为了识别活前缀，我们定义一个状态转换函数 GO 是一个状态转换函数。I 是一个项目集，X 是一个文法符号。函数值 GO(I，X) 定义为：

$$GO(I, X) = CLOSURE(J)$$

其中

J = { 任何形如 $A \rightarrow \alpha X \cdot \beta$ 的项目 | $A \rightarrow \alpha \cdot X\beta$ 属于 I} 。

- **P50：设 a 是Σ中的一个字符，定义**

$$\mathbf{I_a = \varepsilon\text{-closure}(J)}$$

其中，J 为 I 中的某个状态出发经过一条 a 弧而到达的状态集合。

- 文法 G(S′)

  S′→E
  E→aA|bB
  A→cA|d
  B→cB|d

- I_0={S′→ · E, E→ · aA, E→ · bB}

GO(I_0, E)= closure(J)=closure({S’→E · })
= {S’→E · } = I_1

GO(I_0, a)= closure(J)=closure({E→a · A})
={ E→a · A, A→ · cA, A→ · d})=I_2

GO(I_0, b)= closure(J)=closure ({E →b · B})
={E →b · B, B→ · cB, B→ · d}= I_3

- 构造文法 G 的拓广文法 G′ 的 LR(0) 项目集规范族算法:

```
PROCEDURE  ITEMSETS(G′) ;
BEGIN
  C:={CLOSURE({S′→ · S})} ;
  REPEAT
     FOR  C 中每个项目集 I 和 G′ 的每个符号 X
  DO
          IF  GO(I ,  X) 非空且不属于 C   THEN
             把 GO(I ,  X) 放入 C 族中 ;
  UNTIL C   不再增大
END
```

- 转换函数 GO 把项目集连接成一个 DFA 转换图 .

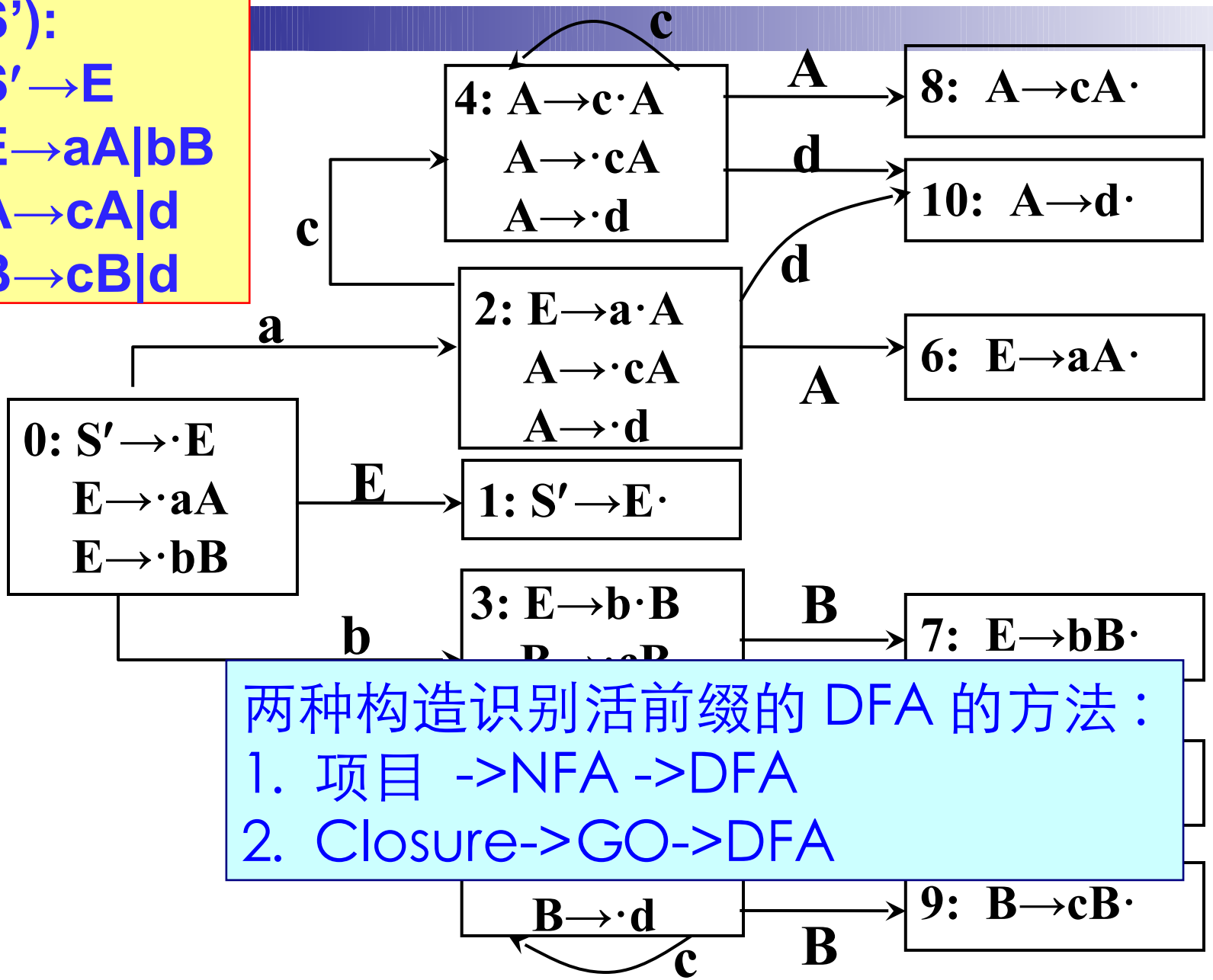
G'(S'):

$S' \rightarrow E$

$E \rightarrow aA|bB$

$A \rightarrow cA|d$

$B \rightarrow cB|d$



两种构造识别活前缀的 DFA 的方法：

1. 项目 ->NFA ->DFA
2. Closure->GO->DFA

小结

- 规范归约过程中，只要保证分析栈中总是活前缀，就说明分析采取的移进 / 归约动作是正确的
- 哪些字符串是活前缀？能不能构造一个 DFA 来识别活前缀？
- 两种构造识别活前缀的 DFA 的方法
  - 项目 ->NFA ->DFA
  - Closure->GO->DFA

编译原理

第五章　语法分析——自下而上分析

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法
  - LR(0) 项目集规范族和 LR(0) 分析表的构造
  - SLR 分析表的构造

LR(0) 分析表的构造

- 假若一个文法 G 的拓广文法 G′ 的活前缀识别自动机中的每个状态 (项目集) 不存在下述情况:

  1) 既含移进项目又含归约项目,

  2) 含有多个归约项目

  则称 G 是一个 LR(0) 文法。

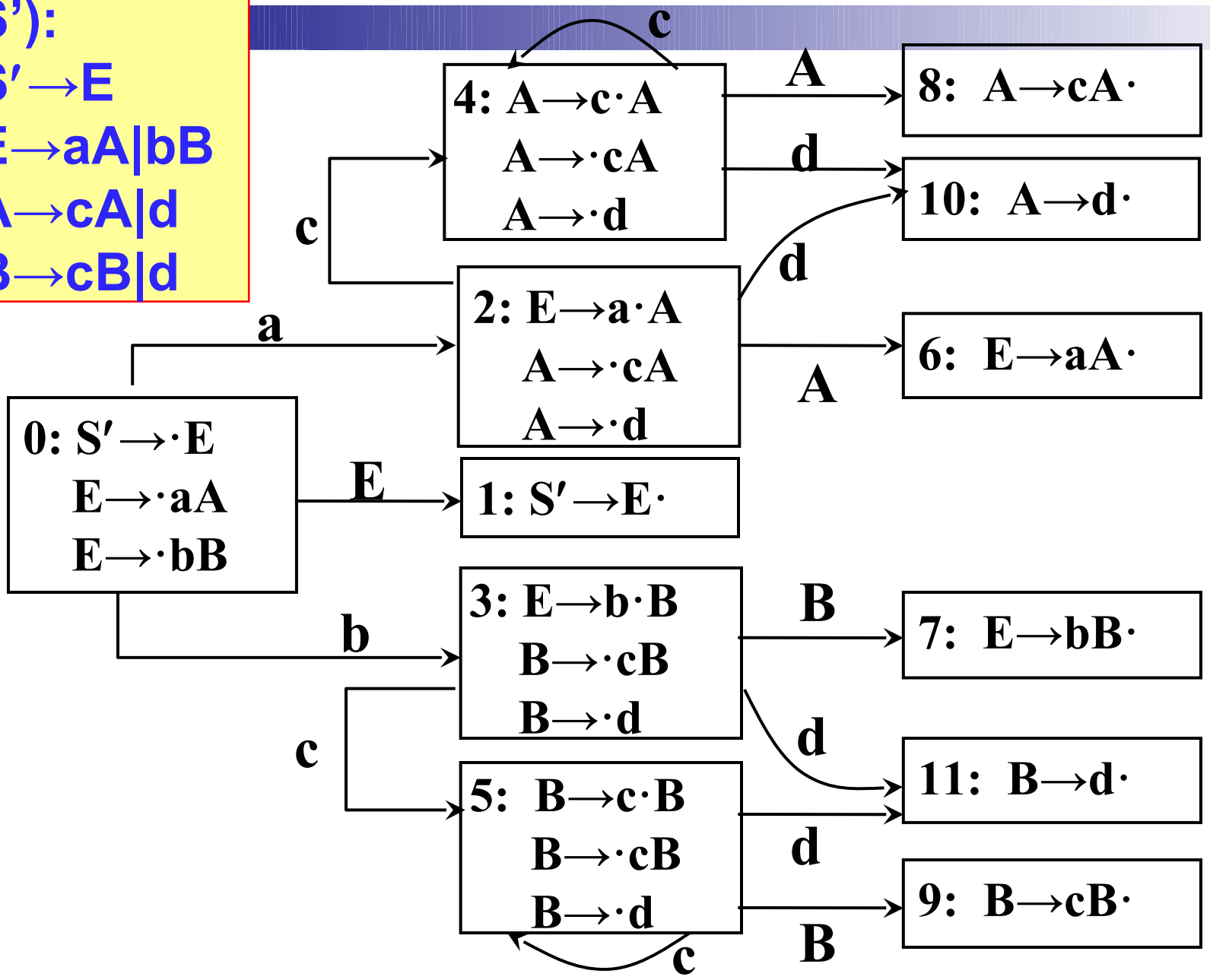
G'(S'):

$S' \rightarrow E$

$E \rightarrow aA|bB$

$A \rightarrow cA|d$

$B \rightarrow cB|d$



构造 LR(0) 分析表的算法

- 令每个项目集 I_k 的下标 k 作为分析器的状态，包含项目 $S' \rightarrow \cdot S$ 的集合 I_k 的下标 k 为分析器的初态。

  - 在任何时候，分析栈中的活前缀 $X_1X_2 \cdots X_m$ 的有效项目也正是从识别活前缀的 DFA 的初态出发，读出 $X_1X_2 \cdots X_m$ 后到达的那个项目集 (状态)
  - 也正是栈顶状态 S_m 所代表的那个集合

LR(0) 分析表的 ACTION 和 GOTO 子表构造

1. 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“ sj”。
2. 若项目 $A\rightarrow\alpha\cdot$ 属于 I_k，那么，对任何终结符 a(或结束符 #)，置 ACTION[k,a] 为 “ rj”(假定产生式 $A\rightarrow\alpha$ 是文法 G′ 的第 j 个产生式)。
3. 若项目 $S'\rightarrow S\cdot$ 属于 I_k，则置 ACTION[k,#] 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 填入信息的空白格均

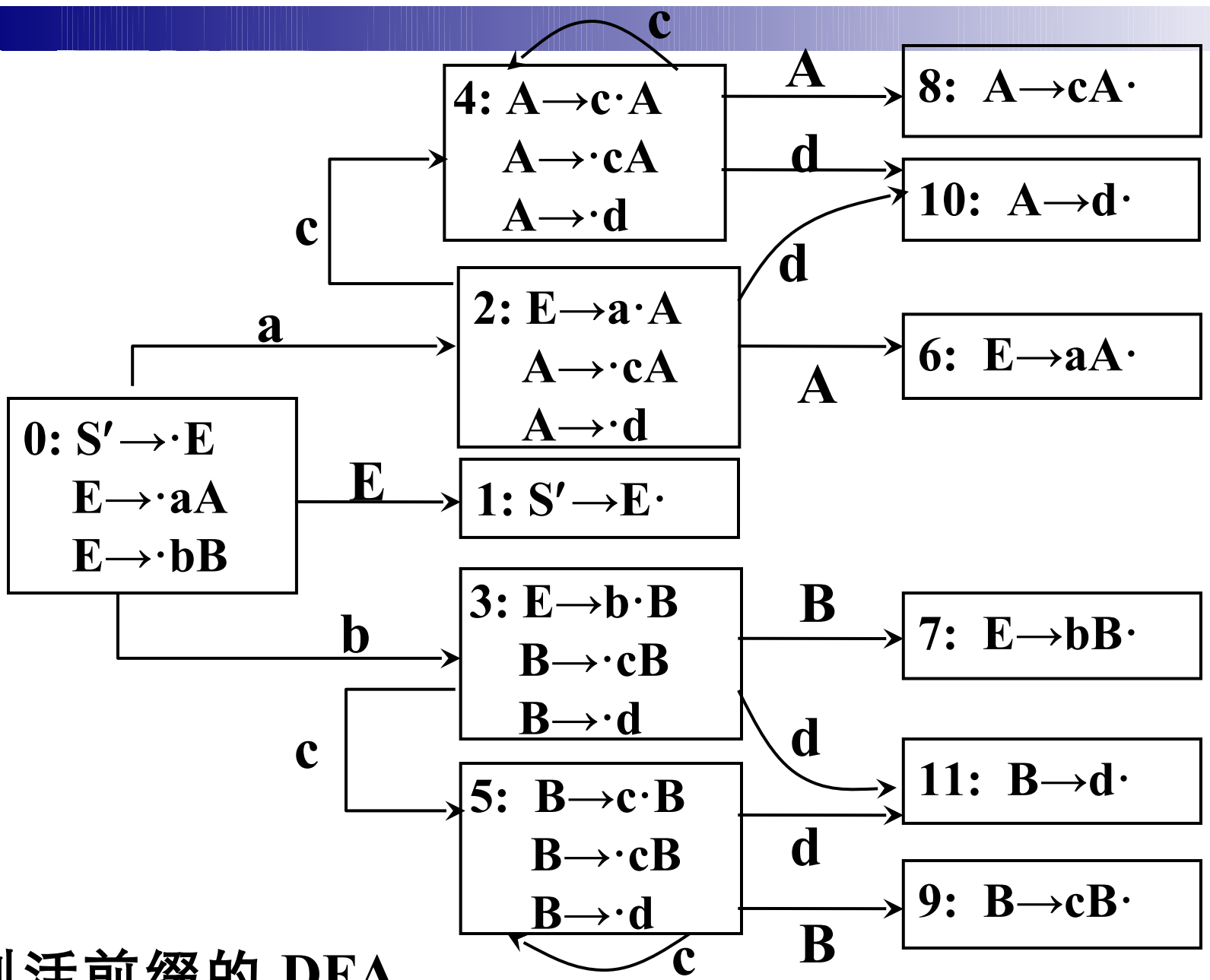
- 文法 G(S′)

  S′→E

  E→aA|bB

  A→cA|d

  B→cB|d

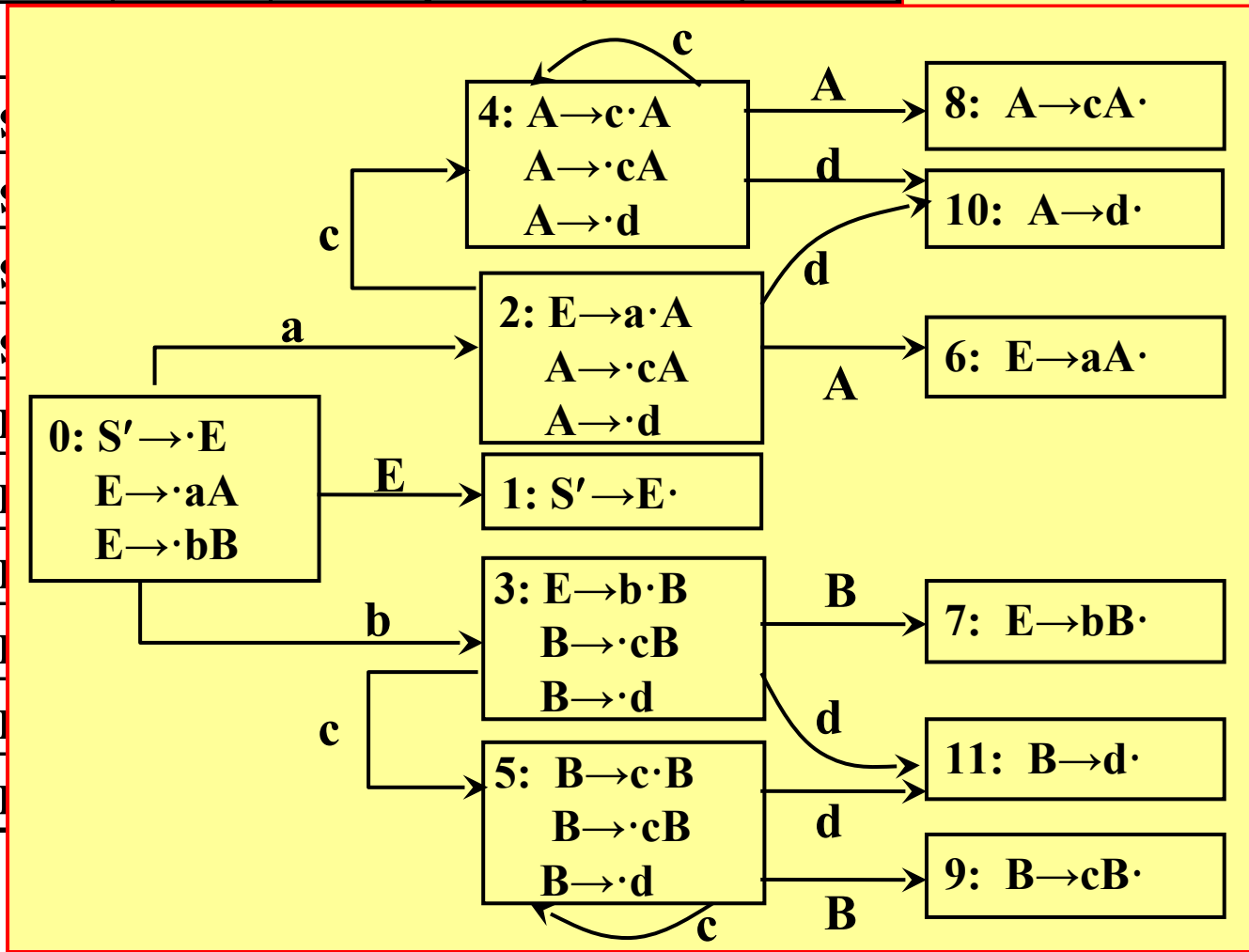

0: S′→·E
E→·aA
E→·bB
1: S′→E·
2: E→a·A
A→·cA
A→·d
3: E→b·B
B→·cB
B→·d
4: A→c·A
A→·cA
A→·d
5: B→c·B
B→·cB
B→·d
6: E→aA·
7: E→bB·
8: A→cA·
9: B→cB·
10: A→d·
11: B→d·
a
b
c
d
E
A
B

识别活前缀的 DFA

- LR(0) 分析表为

	ACTION					GOTO		
状态	a	b	c	d	#	E	A	B
0	s2	s3				1		
1								
2								
3								
4								
5								
6	r1	r1						
7	r2	r2						
8	r3	r3						
9	r5	r5						
10	r4	r4						
11	r6	r6						

0:S′→E
1:E→aA
2:E→ bB
3:A→cA
4:A→ d
5:B→cB
6:B→ d



- 例： 按上表对 bcd 进行分析

步骤	状态	符号	输入串
1	0	#	bcd#
2	03	#b	cd#
3	035	#bc	d#

0:S′→E
1:E→aA
2:E→ bB
3:A→cA
4:A→ d
5:B→cB
6:B→ d

	ACTION					GOTO		
状态	a	b	c	d	#	E	A	B
0	s2	s3				1		
1					acc			
2			s4	s10			6	
3			s5	s11				7
4			s4	s10			8	
5			s5	s11				9
6	r1	r1	r1	r1	r1			
7	r2	r2	r2	r2	r2			
8	r3	r3	r3	r3	r3			

- 例： 按上表对 bccd 进行分析

步骤	状态	符号	输入串
3	035	#bc	d#
4	03511	#bcd	#
5	0359	#bcB	#

0:S′→E
1:E→aA
2:E→ bB
3:A→cA
4:A→ d
5:B→cB
6:B→ d

	ACTION					GOTO		
状态	a	b	c	d	#	E	A	B
0	s2	s3				1		
1					acc			
2			s4	s10			6	
3			s5	s11				7
4			s4	s10			8	
5			s5	s11				9
6	r1	r1	r1	r1	r1			
7	r2	r2	r2	r2	r2			
8	r3	r3	r3	r3	r3			
9	r5	r5	r5	r5	r5			
10	r4	r4	r4	r4	r4			
11	r6	r6	r6	r6	r6			

- 例： 按上表对 bccd 进行分析

步骤	状态	符号	输入串
5	0359	#bcB	#
6	037	#bB	#
7	01	#E	#
8	接受		

0:S′→E
1:E→aA
2:E→ bB
3:A→cA
4:A→ d
5:B→cB
6:B→ d

	ACTION					GOTO		
状态	a	b	c	d	#	E	A	B
0	s2	s3				1		
1					acc			
2			s4	s10			6	
3			s5	s11				7
4			s4	s10			8	
5			s5	s11				9
6	r1	r1	r1	r1	r1			
7	r2	r2	r2	r2	r2			
8	r3	r3	r3	r3	r3			
9	r5	r5	r5	r5	r5			
10	r4	r4	r4	r4	r4			

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法
  - LR(0) 项目集族和 LR(0) 分析表的构造
  - SLR 分析表的构造

5.3.3　SLR 分析表的构造

- LR(0) 文法太简单，没有实用价值
- 例 5.11　考察下面的拓广文法：

  (0) S′→E
  (1) E→E+T
  (2) E→T
  (3) T→T*F
  (4) T→F
  (5) F→(E)
  (6) F→i

- 这个文法的 LR(0) 项目集规范族为：

I_0: $S' \to \cdot E$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_1: $S' \to E \cdot$
$E \to E \cdot +T$

I_2: $E \to T \cdot$
$T \to T \cdot *F$

I_3: $T \to F \cdot$

I_4: $F \to (\cdot E)$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_5: $F \to i \cdot$

I_6: $E \to E+\cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

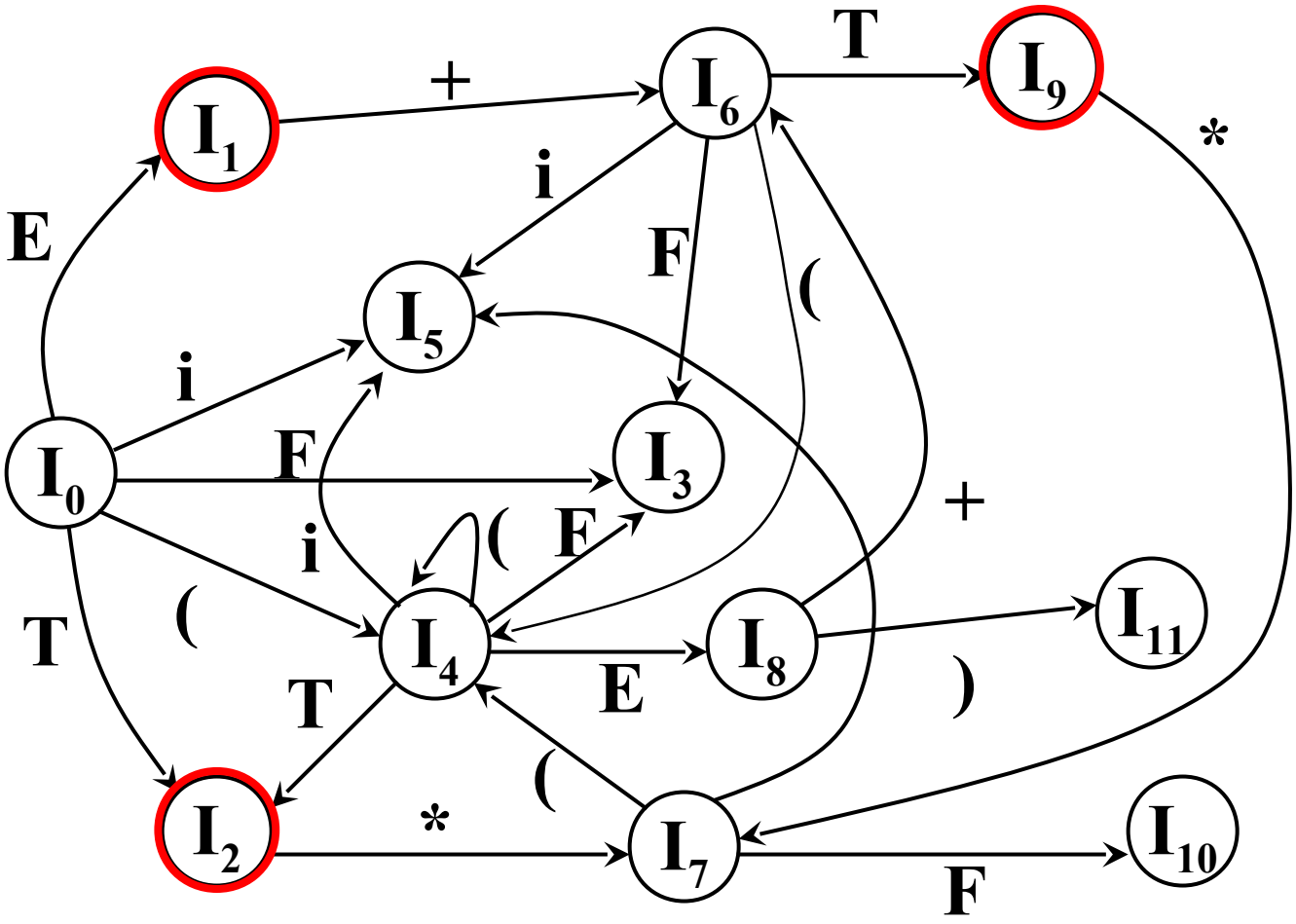
I_7: $T \to T*\cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_8: $F \to (E \cdot)$
$E \to E \cdot +T$

I_9: $E \to E+T \cdot$
$T \to T \cdot *F$

I_{10}: $T \to T*F \cdot$

I_{11}: $F \to (E) \cdot$


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

- 这个文法的 LR(0) 项目集规范族为：

I_0: $S' \to \cdot E$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_1: $S' \to E \cdot$
$E \to E \cdot +T$

I_2: $E \to T \cdot$
$T \to T \cdot *F$

I_3: $T \to F \cdot$

I_4: $F \to (\cdot E)$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_5: $F \to i \cdot$

I_6: $E \to E+ \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_7: $T \to T* \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_8: $F \to (E \cdot)$
$E \to E \cdot +T$

I_9: $E \to E+T \cdot$
$T \to T \cdot *F$

I_{10}: $T \to T*F \cdot$

I_{11}: $F \to (E) \cdot$

- I_1、I_2 和 I_9 都含有“移进 – 归约”冲突

I_1: S′→E·	I_2: E→T·	I_9: E→E+T·
E→E·+T	T→T·*F	T→T·*F

讨论：有没有办法消除冲突?

$$FOLLOW(A)=\{a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T\}$$

FOLLOW(S′) = {#}

FOLLOW(E) = {#,), +}

5.3.3 SLR 分析表的构造

- LR(0) 文法太简单，没有实用价值
- 假定一个 LR(0) 规范族中含有如下的一个项目集 (状态)I = {X→α · bβ， A→α · ， B→α · } 。 FOLLOW(A) 和 FOLLOW(B) 的交集为∅，且不包含 b ，那么 ，当状态 I 面临任何输入符号 *a* 时，可以：
  1. 若 a=b ，则移进；
  2. 若 a∈FOLLOW(A) ，用产生式 A→α 进行归约；
  3. 若 a∈FOLLOW(B) ，用产生式 B→α 进行归约；
  4. 此外，报错。

- 假定 LR(0) 规范族的一个项目集 I={$A_1 \to \alpha \cdot a_1\beta_1$，$A_2 \to \alpha \cdot a_2\beta_2$，…，$A_m \to \alpha \cdot a_m\beta_m$，$B_1 \to \alpha \cdot$，$B_2 \to \alpha \cdot$，…，$B_n \to \alpha \cdot$ } 如果集合 {a_1，…，a_m}，FOLLOW(B_1)，…，FOLLOW(B_n) 两两不相交 (包括不得有两个 FOLLOW 集合有 #)，则：

  1. 若 a 是某个 a_i，i=1,2,…,m，则移进；
  2. 若 a∈FOLLOW(B_i)，i=1,2,…,n，则用产生式 $B_i \to \alpha$ 进行归约；
  3. 此外，报错。

- 冲突性动作的这种解决办法叫做 SLR(1) 解决

构造 SLR(1) 分析表方法

- 首先把 G 拓广为 G′，对 G′ 构造 LR(0) 项目集规范族 C 和活前缀识别自动机的状态转换函数 GO
- 然后使用 C 和 GO，按下面的算法构造 SLR 分析表
  - 令每个项目集 I_k 的下标 k 作为分析器的状态，包含项目 S′→ · S 的集合 Ik 的下标 k 为分析器的初态。

SLR(1) 分析表的 ACTION 和 GOTO 子表构造

1. 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 “ sj” ；

2. 若项目 $A\rightarrow\alpha\cdot$ 属于 I_k，那么，对任何终结符 a， $a\in FOLLOW(A)$，置 ACTION[k,a] 为 “ rj” ；其中，假定 $A\rightarrow\alpha$ 为文法 G′ 的第 j 个产生式;

3. 若项目 $S'\rightarrow S\cdot$ 属于 I_k，则置 ACTION[k,#] 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 填入信息的空白格均

LR(0) 分析表的 ACTION 和 GOTO 子表构造

1. 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“ sj”。
2. 若项目 $A\rightarrow\alpha\cdot$ 属于 I_k，那么，对任何终结符 a(或结束符 #)，置 ACTION[k,a] 为“ rj”(假定产生式 $A\rightarrow\alpha$ 是文法 G′ 的第 j 个产生式)。
3. 若项目 $S'\rightarrow S\cdot$ 属于 I_k，则置 ACTION[k,#] 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 填入信息的空白格均置上“报错标志”

SLR(1) 分析表的 ACTION 和 GOTO 子表构造

1. 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 “ sj” ；

2. 若项目 $A\rightarrow\alpha\cdot$ 属于 I_k，那么，对任何终结符 a， $a\in FOLLOW(A)$，置 ACTION[k,a] 为 “ rj” ；其中，假定 $A\rightarrow\alpha$ 为文法 G′ 的第 j 个产生式;

3. 若项目 $S'\rightarrow S\cdot$ 属于 I_k，则置 ACTION[k,#] 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 填入信息的空白格均

SLR(1) 文法

- 按上述方法构造出的 ACTION 与 GOTO 表如果不含多重入口，则称该文法为 SLR(1) 文法
- 使用 SLR 表的分析器叫做一个 SLR 分析器
- 每个 SLR(1) 文法都是无二义的。但也存在许多无二义文法不是 SLR(1) 的
- LR(0) ⊂ SLR(1) ⊂ 无二义文法

- 例 5.11　考察下面的拓广文法:

  (0) S′→E

  (1) E→E+T

  (2) E→T

  (3) T→T*F

  (4) T→F

  (5) F→(E)

  (6) F→i

- 这个文法的 LR(0) 项目集规范族为：

I_0: $S' \to \cdot E$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_1: $S' \to E \cdot$
$E \to E \cdot +T$

I_2: $E \to T \cdot$
$T \to T \cdot *F$

I_3: $T \to F \cdot$

I_4: $F \to (\cdot E)$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_5: $F \to i \cdot$

I_6: $E \to E+\cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

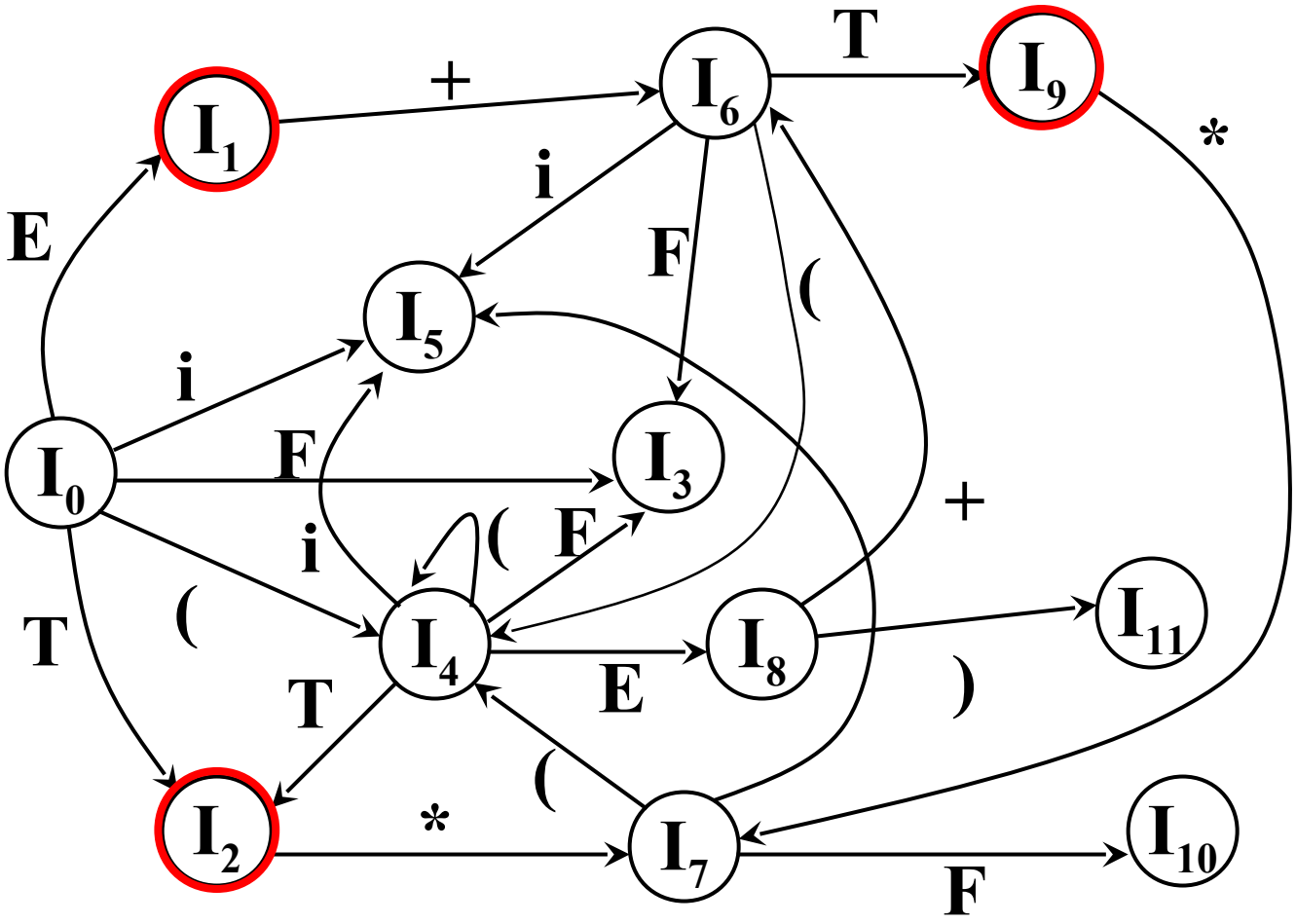
I_7: $T \to T*\cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_8: $F \to (E \cdot)$
$E \to E \cdot +T$

I_9: $E \to E+T \cdot$
$T \to T \cdot *F$

I_{10}: $T \to T*F \cdot$

I_{11}: $F \to (E) \cdot$


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

- 这个文法的 LR(0) 项目集规范族为：

I_0: $S' \to \cdot E$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_1: $S' \to E \cdot$
$E \to E \cdot +T$

I_2: $E \to T \cdot$
$T \to T \cdot *F$

I_3: $T \to F \cdot$

I_4: $F \to (\cdot E)$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_5: $F \to i \cdot$

I_6: $E \to E+\cdot T$
$T \to \cdot T*F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_7: $T \to T*\cdot F$
$F \to \cdot (E)$
$F \to \cdot i$

I_8: $F \to (E \cdot)$
$E \to E \cdot +T$

I_9: $E \to E+T \cdot$
$T \to T \cdot *F$

I_{10}: $T \to T*F \cdot$

I_{11}: $F \to (E) \cdot$

- I_1、I_2 和 I_9 都含有“移进 – 归约”冲突

I_1: $\dot{S}' \rightarrow E\cdot$ $E \rightarrow E\cdot + T$	I_2: $E \rightarrow T\cdot$ $T \rightarrow T\cdot * F$	I_9: $E \rightarrow E + T\cdot$ $T \rightarrow T\cdot * F$

采取 SLR （1）冲突消解

FOLLOW(S′) = {#}

I_1: S′→E·
E→E·+T

其分析表如下：

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

FOLLOW(E) = {#,), +}

I_2: E→T·
T→T·*F

I_9: E→E+T·
T→T·*F

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

小结

- 根据识别活前缀的 DFA 、 LR(0) 项目集规范族
  - 构造 LR(0) 分析表
  - 构造 SLR(1) 分析表

作业

- P134—5 (1) ， (2) ， (3), (4)

编译原理

第五章　语法分析——自下而上分析

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法
  - LR(0) 项目集规范族
  - LR(0) 分析表的构造
  - SLR 分析表的构造
  - LR(1) 分析表的构造

SLR 冲突消解存在的问题

- 计算 FOLLOW 集合所得到的超前符号集合可能大于实际能出现的超前符号集。

$$FOLLOW(A) = \{a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots, a \in V_T\}$$

- 非 SLR 文法示例：
  - (0) S′→S
  - (1) S→L=R
  - (2) S→R
  - (3) L→*R
  - (4) L→i
  - (5) R→L

这个文法的 LR(0) 项目集规范族为

I_0 : $S' \rightarrow \cdot S$

$S \rightarrow \cdot L=R$

$S \rightarrow \cdot R$

$L \rightarrow \cdot *R$

$L \rightarrow \cdot i$

$R \rightarrow \cdot L$

I_1 : $S' \rightarrow S \cdot$

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

$R \rightarrow L \cdot$

I_3 : $S \rightarrow R \cdot$

I_4 : $L \rightarrow * \cdot R$

$R \rightarrow \cdot L$

$L \rightarrow \cdot *R$

$L \rightarrow \cdot i$

I_5 : $L \rightarrow i \cdot$

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

$R \rightarrow \cdot L$

$L \rightarrow \cdot *R$

$L \rightarrow \cdot i$

I_7 : $L \rightarrow *R \cdot$

I_8 : $R \rightarrow L \cdot$

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

I_2：

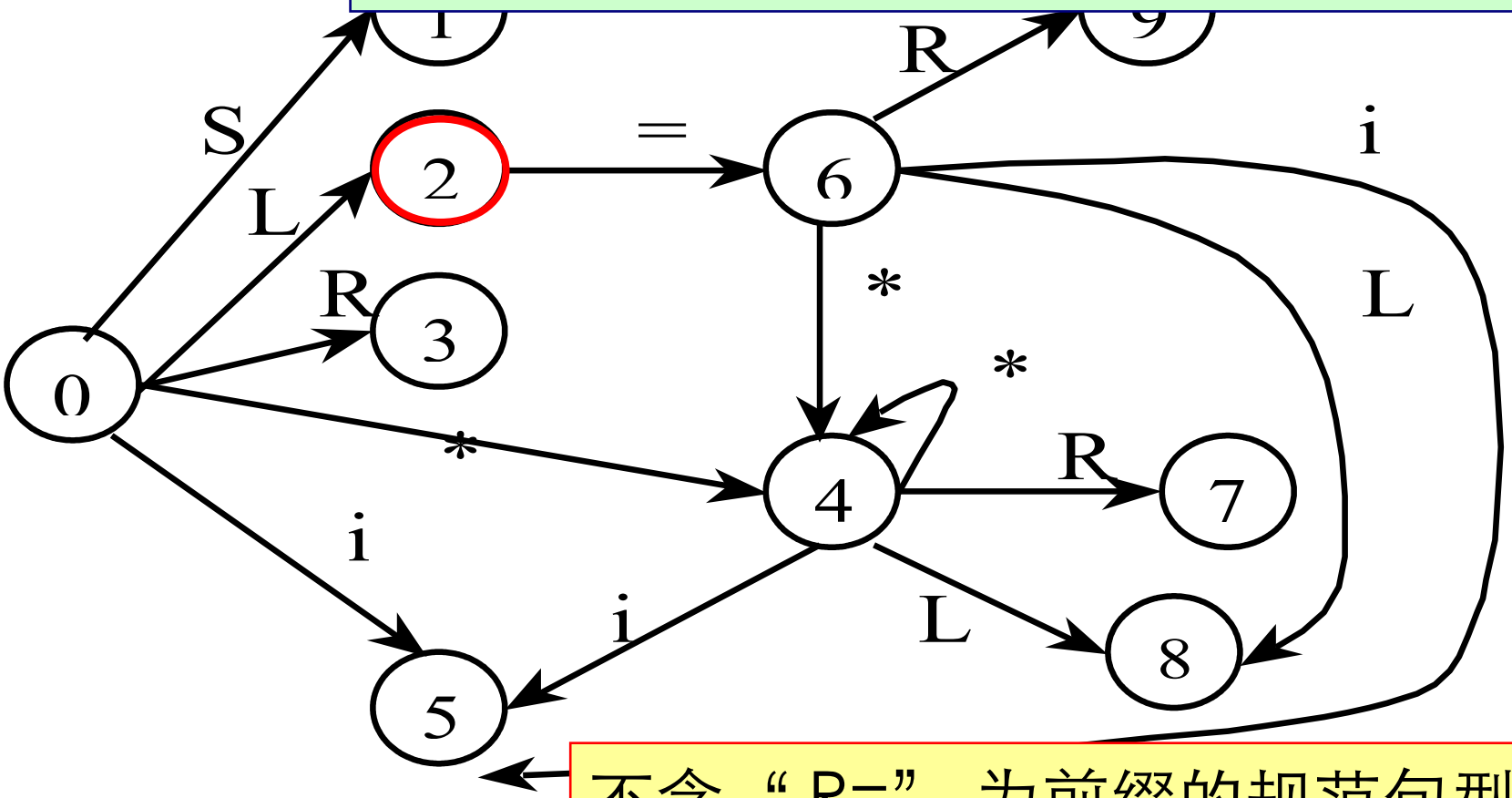
$S \rightarrow L \cdot = R$

$R \rightarrow L \cdot$

含有“移进 – 归约”冲突

FOLLOW(R) = {#, =}

当状态 2 显现于栈顶而且面临输入符号为‘ =’ 时，实际上不能用对栈顶 L 进行归约。



不含“ R=” 为前缀的规范句型

有含“ *R=” 为前缀的规范句型

SLR 冲突消解存在的问题

- SLR 在方法中，如果项目集 I_i 含项目 $A\rightarrow\alpha\cdot$ 而且下一输入符号 $a\in FOLLOW(A)$，则状态 i 面临 a 时，可选用“用 $A\rightarrow\alpha$ 归约”动作
- 但在有些情况下，当状态 i 显现于栈顶时，栈里的活前缀未必允许把α归约为 A，因为可能根本就不存在一个形如“β Aa” 的规范句型
- 在这种情况下，用“ $A\rightarrow\alpha$ ” 归约不一定合适
- FOLLOW 集合提供的信息太泛!

$$FOLLOW(A)=\{a \mid S \overset{*}{\Rightarrow} \ldots Aa \ldots, a\in V_T\}$$

5.3.4 规范 LR 分析表的构造

- 我们需要重新定义项目，使得每个项目都附带有 k 个终结符
- 每个项目的一般形式是

$$[A\rightarrow\alpha\cdot\beta, a_1a_2\cdots a_k]$$

这样的一个项目称为一个 LR(k) 项目。项目中的 $a_1a_2\cdots a_k$ 称为它的向前搜索符串 (或展望串)。

- 向前搜索符串仅对归约项目 $[A\rightarrow\alpha\cdot\ ,\ a_1a_2\cdots a_k]$ 有意义
- 对于任何移进或待约项目 $[A\rightarrow\alpha\cdot\beta, a_1a_2\cdots a_k]$, $\beta\neq\varepsilon$，搜索符串 $a_1a_2\cdots a_k$ 没有直接作用

归约项目

- 归约项目 $[A\rightarrow\alpha\cdot, a_1a_2\cdots a_k]$ 意味着：当它所属的状态呈现在栈顶且后续的 k 个输入符号为 $a_1a_2\cdots a_k$ 时，才可以把栈顶上的α归约为 A
- 我们只对 $k\leq 1$ 的情形感兴趣，向前搜索 (展望) 一个符号就多半可以确定“移进”或“归约”

有效项目

- 形式上我们说一个 LR(1) 项目 [A→α · β, a] 对于活前缀γ是有效的，如果存在规范推导

$$S \underset{R}{\overset{*}{\Rightarrow}} \delta A \omega \underset{R}{\Rightarrow} \delta \alpha \beta \omega$$

其中， 1) γ = δα； 2) a 是ω的 第一个符号，或者 a 为 # 而ω为ε。

LR(1) 项目集规范族

- 为构造有效的 LR(1) 项目集规范族我们需要两个函数 CLOSURE 和 GO 。

- [A→α·Bβ, *a*] 对活前缀γ = δα是有效的，则对于每个形如 B→ξ 的产生式， 对任何 b∈FIRST(β*a*) ， [B→·ξ, b] 对γ也是有效的。

- 证明：若项目 [A→α·Bβ, *a*] 对γ = δα有效，则有规范推导

$$S \underset{R}{\overset{*}{\Rightarrow}} \delta Aa\chi \underset{R}{\Rightarrow} \delta\alpha B\beta a\chi \qquad \gamma = \delta\alpha$$

$$\because\ b \in FIRST(\beta a) \qquad \therefore \beta a\chi \underset{R}{\overset{*}{\Rightarrow}} b\omega$$

若$B \to \xi$是产生式

则$S \underset{R}{\overset{*}{\Rightarrow}} \delta\alpha B\beta a\chi \underset{R}{\Rightarrow} \gamma Bb\omega \underset{R}{\Rightarrow} \gamma\xi b\omega$

$\therefore$ 项目$[B \to .\xi, b]$对于γ是有效的

项目集的闭包 CLOSURE(I)

1. I 的任何项目都属于 CLOSURE(I) 。

2. 若项目 [A→α · Bβ, *a*] 属于 CLOSURE(I) ， B→ξ 是一个产生式，那么，对于 FIRST(β*a*) 中的每个终结符 *b* ，如果 [B→ · ξ, *b*] 原来不在 CLOSURE(I) 中，则把它加进去。

3. 重复执行步骤 2 ，直至 CLOSURE(I) 不再增大为止。

项目集的转换函数 GO

- 令 I 是一个项目集， X 是一个文法符号，函数 GO(I ， X) 定义为：

$$GO(I, X) = CLOSURE(J)$$

其中

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

LR(1) 项目集规范族的构造算法

- 计算 LR(1) 项目集规范族 C

```
BEGIN
    C:={CLOSURE({[S′→ · S ， #]})};
    REPEAT
      FOR C 中每个项目集 I 和 G′ 的每个符号 X
      DO
          IF GO(I ， X) 非空且不属于 C ， THEN
                把 GO(I ， X) 加入 C 中
    UNTIL C 不再增大
 END
```

LR(1) 分析表的构造算法

- 构造 LR(1) 分析表的算法
  - 令每个 I_k 的下标 k 为分析表的状态，令含有 [S′→ · S, #] 的 I_k 的 k 为分析器的初态。

LR(1) 分析表的 ACTION 和 GOTO 子表构造

1. 若项目 [A→α · aβ, b] 属于 I_k 且 GO(I_k, a) = I_j，a 为终结符，则置 ACTION[k, a] 为 “ sj” 。
2. 若项目 [A→α · ， a] 属于 I_k，则置 ACTION[k, a] 为 “ rj” ；其中假定 A→α 为文法 G′ 的第 j 个产生式。
3. 若项目 [S′→S · , #] 属于 I_k，则置 ACTION[k, #] 为 “ acc” 。
4. 若 GO(I_k， A) = I_j，则置 GOTO[k, A]=j 。
5. 分析表中凡不能用规则 1 至 4 填入信息的空白栏均填上“出错标志”。

LR(1) 分析表和 LR(1) 文法

- 按上述算法构造的分析表，若不存在多重定义的入口 (即，动作冲突) 的情形，则称它是文法 G 的一张规范的 LR(1) 分析表。
- 使用这种分析表的分析器叫做一个规范的 LR 分析器。
- 具有规范的 LR(1) 分析表的文法称为一个 LR(1) 文法。
- LR(1) 状态比 SLR 多，

$$LR(0) \subset SLR \subset LR(1) \subset \text{无二义文法}$$

示例： LR(1) 分析表构造

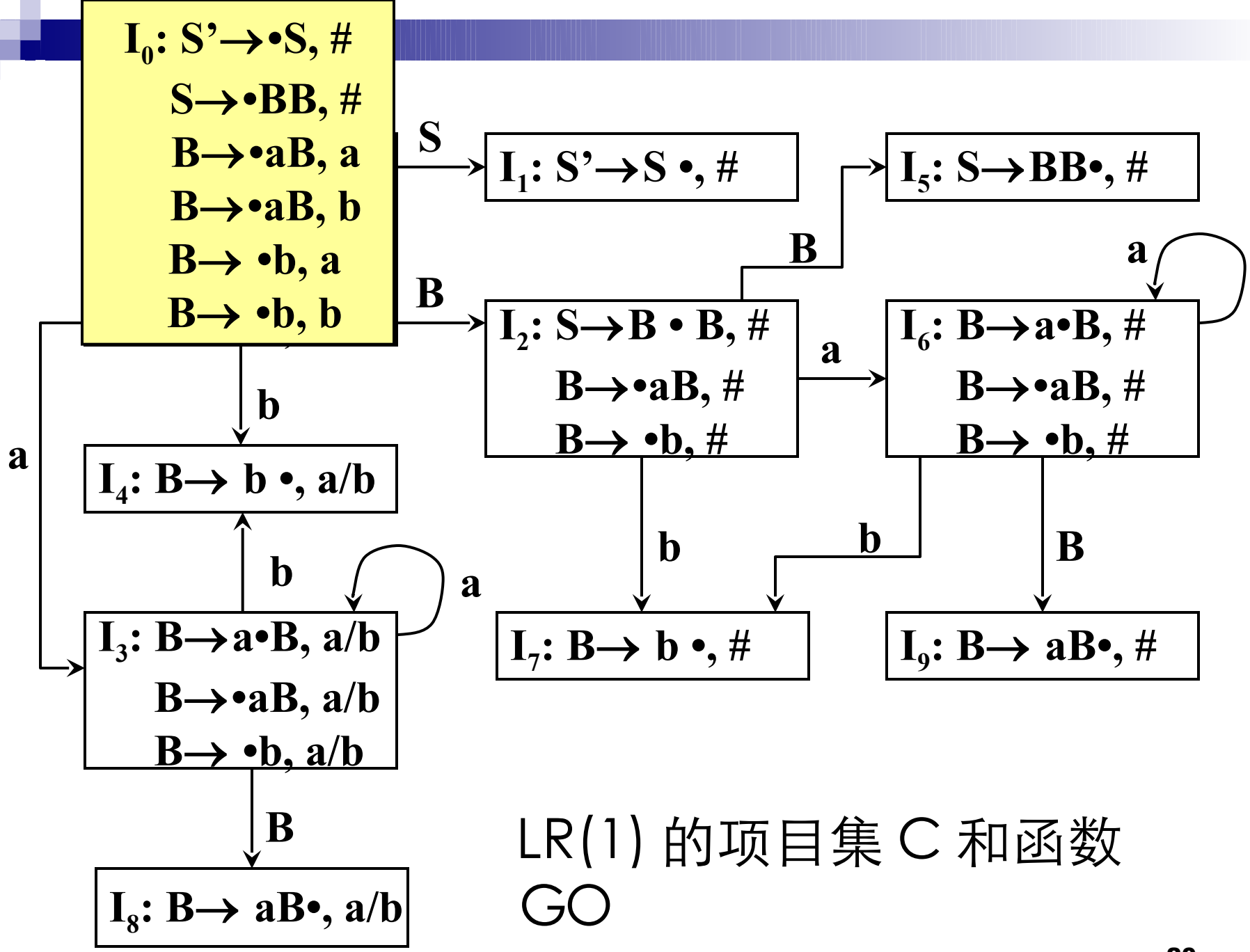
- 例 5.13　(5.10) 的拓广文法 G(S′)

  (0) S′→S

  (1) S→BB

  (2) B→aB

  (3) B→b



LR(1) 的项目集 C 和函数 GO

LR(1) 分析表为：

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		

- 例： 按上表对 aabab 进行分析

步骤	状态	符号	输入串
0	0	#	aabab#
1	03	#*a*	abab#
2	033	#aa	bab#
3	0334	#aab	ab#
4	0338	#aaB	ab#

(0) S′→S

(1) S→BB

(2) B→aB

(3) B→b

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					

例：按上表对 aabab 进行分析

步骤	状态	符号	输入串
4	0338	#aaB	ab#
5	038	#aB	ab#
6	02	#B	ab#
7	026	#Ba	b#
8	0267	#Bab	#

(0) S′→S

(1) S→BB

(2) B→aB

(3) B→b

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			

- 例：按上表对 aabab 进行分析

步骤	状态	符号	输入串	
8	0267	#Bab	#	
9	0269	#BaB	#	
10	025	#BB	#	
11	01	#S	#	acc

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		

(0) S′→S

(1) S→BB

(2) B→aB

(3) B→b

- 例：按上表对 abab 进行分析

步骤	状态	符号	输入串	
0	0	#	abab#	
1	03	#*a*	bab#	
2	034	#ab	ab#	
3	038	#aB	ab#	
4	02	#B	ab#	
5	026	#Ba	b#	
6	0267	#Bab	#	
7	0269	#BaB	#	
8	025	#BB	#	
9	01	#*S*	#	acc

一些分析器产生器

- YACC 工具实习
  - 阅读资料
    - Yacc 与 Lex 快速入门 .pdf
    - Bison Manual
      - 2.1 Reverse Polish Notation Calculator

一些分析器产生器

- YACC 工具实习
  - 用 Bison 构建一个 Reverse Polish Notation Calculator
    - 编写 rpcalc.y
    - 用 Bison 编译 rpcalc.y，生成 rpcalc.tab.c
    - 用 C++ 编译器编译 rpcalc.tab.c ，生成 rpcalc
    - 运行 rpcalc，输入若干后缀式，观察运行结果
  - 改进
    - 用 Flex 生成词法分析器

一些分析器产生器

- YACC 工具实习
  - 提交文档 .zip
    - rpcalc.y
    - rpcalc.tab.c
    - rpcalc
    - Word 文件 Run.doc: 运行 rpcalc，输入若干后缀式，运行结果的截图
    - rpcalc 的 LEX 源程序： rpcalc.lex
    - rpcalc 词法分析程序 C 源程序： rpcalc.yy.c

一些分析器产生器

- The Lex & Yacc Page
  - http://dinosaur.compilertools.net/
- YACC——Yet Another Compiler Compiler
  - Stephen C. Johnson. YACC: Yet Another Compiler-Compiler. *Unix Programmer's Manual* Vol 2b, 1979.
  - GNU bison: 基本兼容 Yacc，与 flex 一起使用
  - Berkeley Yacc : 目前最好的 Yacc 变种
  - LALR(1) 分析

小结

- LR(1) 项目集规范族
- LR(1) 分析表的构造

作业

- P134—8(选作)

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

1.试简述语法分析程序自动生成的基本思想。
(南京大学 1999 年试题)

解答:

语法分析程序自动生成的基本思想是根据用户提供的源语言的语法描述规格说明，基于语法分析的原理，如 LR，LL(1)，算符优先分析法等，自动构造一个该语言的语法分析器。通常这类语法分析器由控制程序和分析表组成，控制程序是通用的，而分析表则根据用户提供的源语言的语法描述规格说明自动生成。

2. 已知文法 G(S)：S → aS | bS | a

1) 构造该文法的 LR(0)项目集规范族；
2) 构造识别该文法所产生的活前缀的 DFA；
3) 构造其 SLR 分析表，并判断该文法是否是 SLR(1)文法。

(北京航空航天大学 1998 年试题)

解题思路：构造 LR(0)项目集规范族，有两种方法：一种是利用有限自动机来构造，另一种是利用函数 CLOSURE 和 GO 来构造。本题采取第二种方法，通过计算 CLOSURE 和 GO 得到文法的 LR(0)项目集规范族，而 GO 函数则把 LR(0)项目集规范族连成一个识别该文法所产生的活前缀的 DFA。

解答:

1) 将文法 G(S)拓广为 G(S')：

(0) S' → S
(1) S → aS
(2) S → bS
(3) S → a

构造该文法的 LR(0)项目集规范族：

I0 = CLOSURE({ S' → •S }) = { S' → •S，S → •aS，S → •bS，S → •a }
I1 = GO(I0，a) = CLOSURE({S → a•S，S → a• }) = {S → a•S,
S → a•，S → •aS，S → •bS，S → •a }
I2 = GO(I0，b) = CLOSURE({S → b•S }) = {S → b•S， S → •aS， S → •bS，S → •a }
I3 = GO(I0，S) = CLOSURE({S' → S• }) = { S' → S•}
GO(I1，a) = CLOSURE({S → a•S，S → a• }) = I1
GO(I1，b) = CLOSURE({S → b•S }) = I2
I4 = GO(I1，S) = CLOSURE({S → aS• }) = {S → aS• }
GO(I2，a) = CLOSURE({S → a•S，S → a• }) = I1
GO(I2，b) = CLOSURE({S → b•S }) = I2
I5 = GO(I2，S) = CLOSURE({S → bS• }) = {S → bS• }

所以，项目集 I0，I1，I2，I3，I4 和 I5 构成了该文法的 LR(0)项目集规范族。

2) 我们用 GO 函数则把 LR(0)项目集规范族连成一个识别该文法所产生的活前缀的 DFA 如图 4.2。

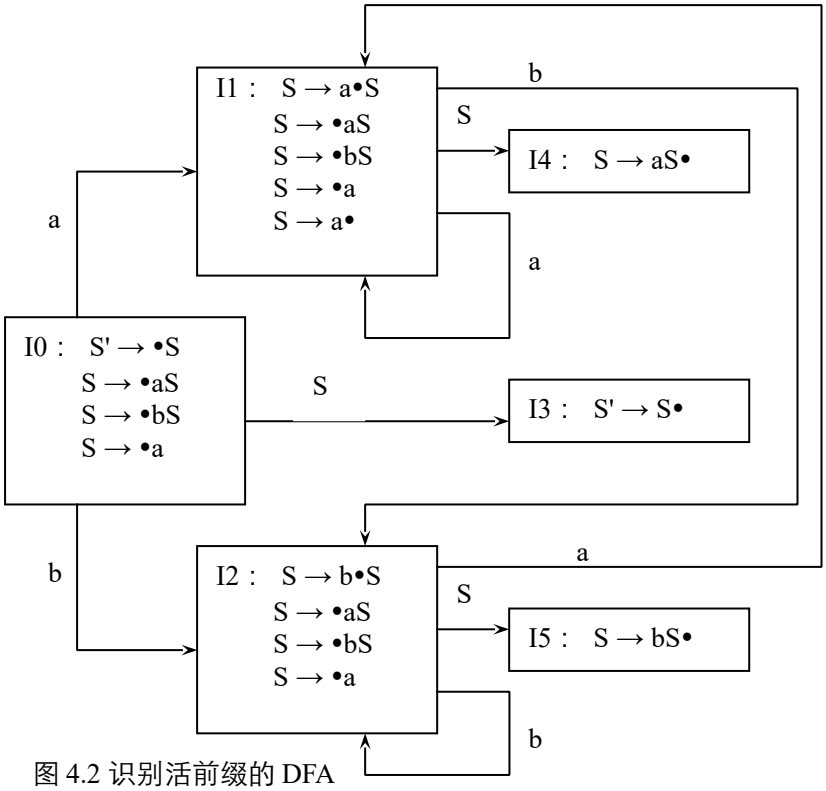


图 4.2 识别活前缀的 DFA

3) 构造其 SLR 分析表：

注意到状态 I1 存在移进-归约冲突，计算 S 的 FOLLOW 集合：

FOLLOW(S)={#}

可以采用 SLR 冲突消解法，得到下面的 SLR 分析表：

状态	ACTION			GOTO
	a	b	#	S
0	s1	s2		3
1	s1	s2	r3	4
2	s1	s2		5
3			acc	
4			r1	
5			r2	

从分析表可以看出，表中没有冲突项，所以该文法是 SLR(1)文法。

对于一个文法，如果能够构造一张分析表，使得它的每个入口均是唯一确定的，则这个文法就称为 **LR 文法**。

一个文法，如果能用一个每步顶多向前检查 k 个输入符号的 LR 分析器进行分析，则这个文法就称为 **LR(k)文法**。

字的前缀是指字的任意首部，如字 *abc* 的前缀有 ε，*a*，*ab*，*abc*。

活前缀是指规范句型的一个前缀，这种前缀不含句柄之后的任何符号。即，对于规范句型 $\alpha\beta\delta$，β 为句柄，如果 $\alpha\beta=u_1u_2\ldots u_r$，则符号串 $u_1u_2\ldots u_i(1\le i\le r)$是 $\alpha\beta\delta$ 的活前缀(δ 必为终结符串)。

假若一个文法 G 的拓广文法 G′的活前缀识别自动机中的每个状态(项目集)不存在下述情况：

1) 既含移进项目又含归约项目，

2) 含有多个归约项目

则称 G 是一个 **LR(0)文法**。

每个项目的一般形式是$[A\rightarrow\alpha\cdot\beta, a_1a_2\ldots a_k]$，这样的一个项目称为一个 **LR(k)项目**。项目中的 $a_1a_2\ldots a_k$ 称为它的**向前搜索符串**(或展望串)。

1965 年，Knuth 提出了 LR 分析法。

高德纳最早开始自行编写 TEX 的原因是当时十分粗糙的排版水平已经影响到他的巨著《计算机程序设计艺术》（The Art of Computer Programming）的印刷质量。他以典型的黑客思维模式，最终决定自行编写一个排版软件：TEX。他原本以为他只需要半年时间，在 1978 年下半年就能完成，但最终他用了超过十年时间，直到 1989 年 TEX 才最终停止修改。

Guy Steele 在高德纳编写其第一版的 TEX 程序时正好在高德纳所工作的斯坦福大学，当他回到 MIT 时他在 ITS 下重写了 TEX 的输出输入系统。

TEX 的第一版是用 SAIL 编程语言写成的，运行于 PDP-10 型计算机，操作系统是斯坦福大学的 WAITS 操作系统。在之后的 TeX 版本中，Knuth 发明了所谓的“文学编程”（literate programming），一种从同一源文件自动生成可编译的源代码和高质量的文档的编程方法。这种语言被称为 WEB，它所生成的源代码是使用 Pascal 编程语言。

TEX 的版本号码也十分有趣。从 TEX 第三版开始，之后的升级是在小数点后加入一个新数位，使之越来越接近圆周率 π 的值。TEX 目前的版本是

3.1415926。这显示了 TEX 已经十分稳定，任何的升级都十分细微。高德纳曾表示“最后一次升级是（于我过世后）将版本数改为 π，那时任何余下的漏洞将被看作程序的功能。”

TEX 允许自由的再发布及修改，但禁止任何修改版本以 TEX 或任何其他相似的名字命名。TEX 是非常稳定的程序，高德纳悬赏奖励任何能够在 TEX 中发现程序漏洞（bug）的人。每一个漏洞的奖励金额从 1 美分开始，并每年翻倍，直到目前的 327.68 美元封顶。然而高德纳从未因此而损失大笔金钱，因为 TEX 中的漏洞少之又少，而真正发现漏洞的人在获得支票后，宁愿将其裱起来留作纪念也不愿拿去兑现！

到目前为止，关于 TEX 的最后一个 bug 是被高德纳自己发现的。

CTeX 是 LaTeX 排版软件。是一个文本编辑器，特别适合小文本文件。它支持拖放、支持 AppleScript 等等。LaTeX（LATEX）是一种基于 TeX 的排版系统，由美国计算机学家 Leslie Lamport 在 20 世纪 80 年代初期开发，利用这种格式，即使使用者没有排版和程序设计的知识也可以充分发挥由 TeX 所提供的强大功能，能在几天，甚至几小时内生成很多具有书籍质量的印刷品。对于生成复杂表格和数学公式，这一点表现得尤为突出。因此它非常适用于生成高印刷质量的科技和数学类文档。这个系统同样适用于生成从简单的信件到完整书籍的所有其他种类的文档。

Leslie Lamport 开发的 LATEX 是当今世界上最流行和使用最为广泛的 TEX 宏集。它构筑在 Plain TEX ~~ 的基础之上，并加进了很多的功能以使得使用者可以更为方便的利用 TEX 的强大功能.使用 LATEX 基本上不需要使用者自己设计命令和宏等，因为 LATEX 已经替你做好了。因此，即使使用者并不是很了解

TEX,也可以在短短的时间内生成高质量的文档。对于生成复杂的数学公式,LATEX 表现的更为出色。LATEX 自从八十年代初问世以来，也在不断的发展.最初的正式版本为 2.09，在经过几年的发展之后，许多新的功能，机制被引入到 LATEX 中。在享受这些新功能带来的便利的同时，它所伴随的副作用也开始显现，这就是不兼容性.标准的 LATEX 2.09 引入了“新字体选择框架”(NFSS)的 LATEX，SLiTEX，AMSLATEX 等等，相互之间并不兼容.这给使用者和维护者都带来很大的麻烦。为结束这中糟糕的状况，FrankMittelbach 等人成立了 ATEX3 项目小组，目标是建立一个最优的，有效的，统一的，标准的命令集合。即得到 LATEX 的一个新版本 3。这是一个长期目标，向这个目标迈出第一步就是在 1994 年发布的 LATEX2e。LATEX2e 采用了 NFSS 作为标准，加入了很多新的功能，同时还兼容旧 LATEX 2.09。LATEX2e 每 6 个月更新一次，修正发现的错误并加入前，LATEX2e 将是标准的。

1.有文法 G=({S}，{a}，{ S → SaS，S→ ε}，S)：

该文法是。

A. LL(1)文法　　B. 二义性文法

C. 算符优先文法　　D. SLR(1)文法

答案： B

2.给出文法 G(P)

P → aPb | Q

Q → bQc | bSc

S → Sa | a

请构造它的 SLR 分析表，以说明它是不是 SLR 文法。

答案：

将文法 G(P)拓广为 G(S')：

(0) S'→ P

(1) P → aPb

(2) P → Q

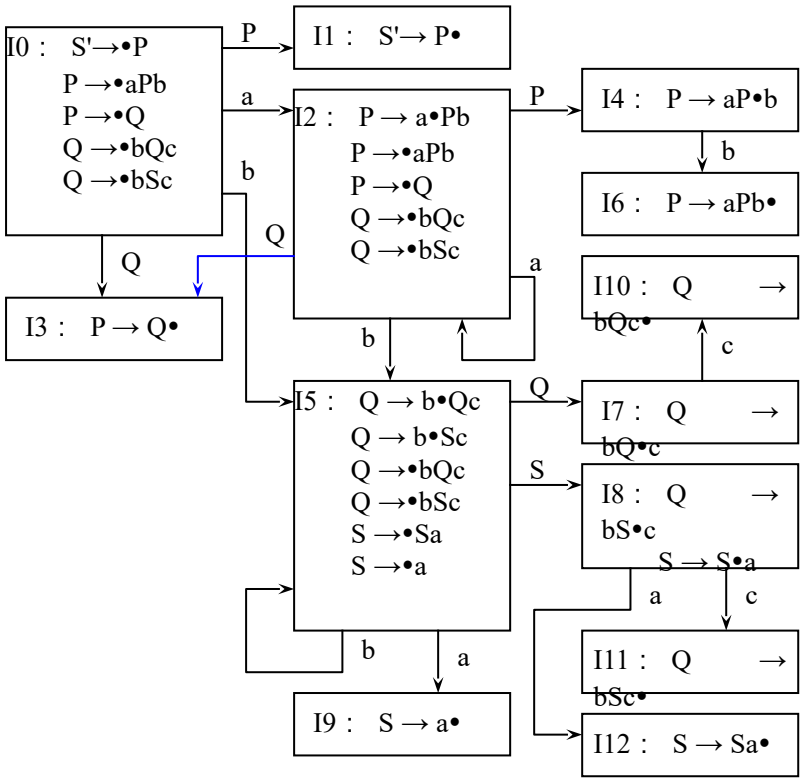
(3) Q → bQc

(4) Q → bSc

(5) S → Sa

(6) S → a

识别该文法所产生的活前缀的 DFA 如图 4.11



计算 FOLLOW 集合：

FOLLOW(S') = {#}

FOLLOW(P) = {b，#}

FOLLOW(Q) = {b，c，#}

FOLLOW(S) = {a，c}

构造它的 SLR 分析表如下表

SLR 分析表

	ACTION				GOTO		
	a	b	c	#	P	Q	S
0	s2	s5			1	3	
1				acc			
2	s2	s5			4	3	
3		r2		r2			
4		s6					
5	s9	s5				7	8
6		r1		r1			
7			s10				
8	s12		s11				
9	r6		r6				
10		r3	r3	r3			
11		r4	r4	r4			
12	r5		r5				

从它的 SLR 分析表可以看出，表中没有冲突，所以文法 G 是 SLR 文法。