

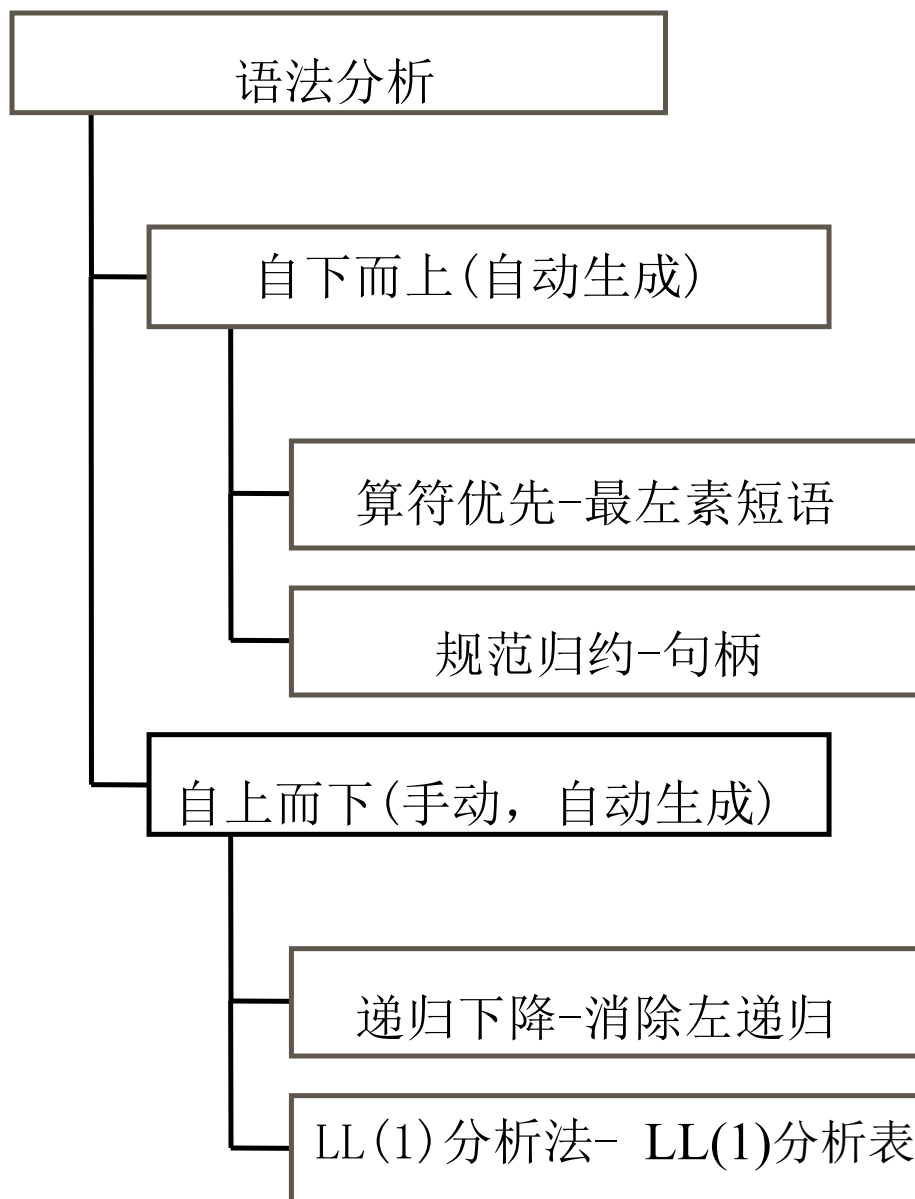
第六章

LR分析程序及其自动构造

- 6.1 LR分析概述
- 6.2 LR (0) 分析
- 6.3 SLR(1) 分析
- 6.4 LR(1) 分析



语法分析概述



LR分析法是一种自下而上进行规范归约的语法分析方法, L指自左向右扫描输入串, R指最右推导(规范归约)。

LR分析法比递归下降分析法、LL(1)分析法对文法的限制要少得多, 大多数无二义性CFG语言都可用LR分析器识别, 且速度快, 并能准确、指出输入串的语法错误及出错位置。

LR分析法的主要缺点:

手工构造工作量相当大, 必须求助自动产生工具。

- LR分析程序(器)：自左向右扫描，识别句柄，自下而上归约的语法分析程序。
- LR分析程序生成器：自动生成LR分析程序的程序。

LR分析器的工作原理

规范归约(最右推导逆过程)的关键是寻找句柄。

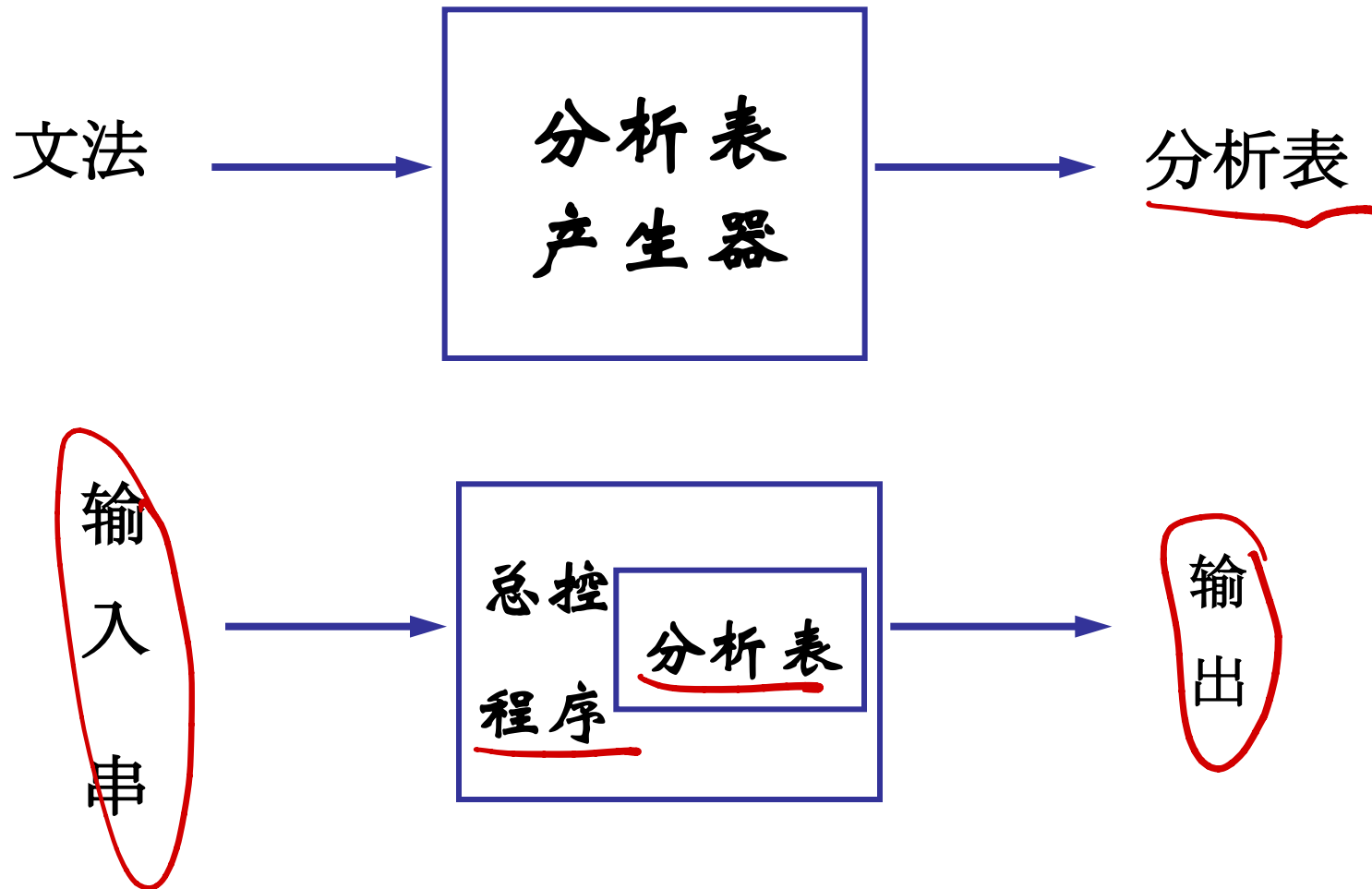
LR分析法的基本思想:

在规范归约过程中,一方面记住已移进和归约出的符号串,即记住“历史”(栈);

另一方面根据所用产生式推测未来可能遇到的输入符,即对未来进行“展望”。

当一串貌似句柄的符号串呈现于分析栈栈顶时,希望能根据所记载的“历史”、“展望”及“现实”材料来确定栈顶符号是否构成句柄。

LR分析程序结构



- 一个LR分析器实质上是一个带先进后出存储器（栈）的确定有限状态自动机。 类似词法分析
- 我们将把“历史”和“展望”材料综合地抽象成某些“状态”（自动机）。
- 分析栈（先进后出存储器）用来存放状态。栈里的每个状态概括了从分析开始直到某一归约阶段的全部“历史”和“展望”资料。
- 任何时候，栈顶的状态都代表了整个的历史和已推测出的展望。因此，在任何时候都可从栈顶状态得知所想了解的文法的一切信息，而没有必要从底而上翻阅整个栈。
- LR分析器的每一步工作都是由栈顶状态和现行输入符号所唯一决定的。



6.1 LR分析概述

概念

LR (K)

L 从左至右扫描输入符号串

R 构造一个最右推导的逆过程

K 向右顺序查看输入串的K个符号

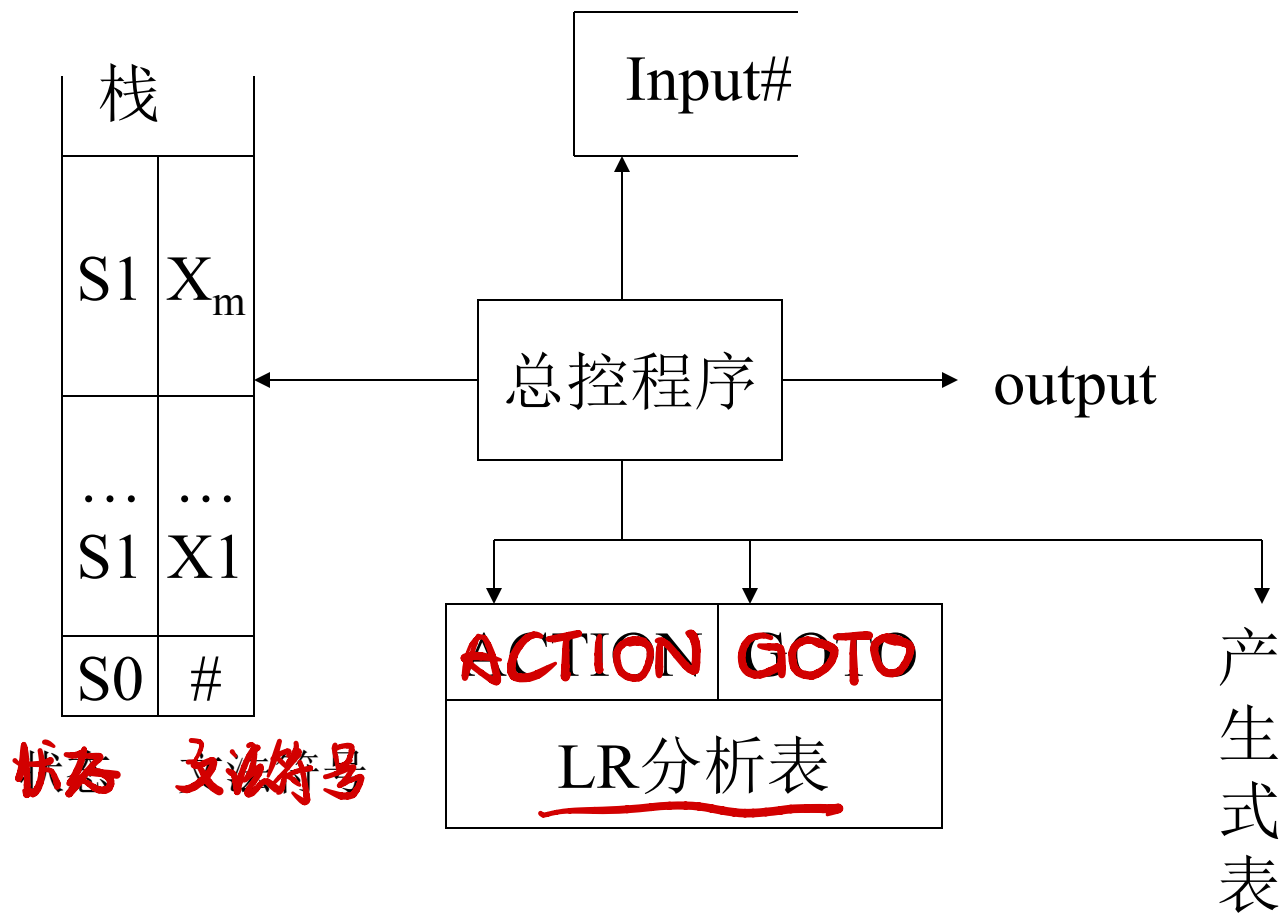
LR (0) :在分析过程中不需向右查看输入符号。

四种分析器: LR(0) SLR(1) LR(1) LALR(1)

SLR(1)和LALR(1)分别是LR(0)和LR(1)的一种改进。

- 分析器模型和分析算法

LR分析器模型



ACTION[k, a] 的动作:

(1) 移进: 设表中 $\text{ACTION}[k, a] = S_j$, 当 S 栈顶状态为 k , 现行输入符号为 a , 总控程序根据 “ k ” 和 “ a ” 查 LR(0) 分析表, 得:

$\text{ACTION}[k, a] = S_j$

书上这号不对

此时, S_j 表示 j 状态进 S 栈, a 进 T 栈 (文法符号栈)。

(2) 归约: 设LR(0)分析表中的 $\text{ACTION}[m_n, a] = r_j$, 其中 r_j 表示使用文法的第 j 个产生式 $A \rightarrow x_1x_2 \dots x_p$ 归约; m_n 表示LR分析表的一个状态。

假设总控程序按 S 栈顶状态 m_n 和现行输入符号 a 查LR分析表, 得

$$\text{ACTION}[m_n, a] = r_j$$

此时, S 栈的状态为:

$$m_1, \dots, m_{n-p+1}, \dots, m_n$$

文法符号 T 栈的符号为:

$$\dots x_1 x_2 \dots x_p$$

按 $\text{ACTION}[m_n, a] = r_j$ 的要求:

① S栈应删除栈顶p个状态:

$$m_{n-p+1}, \dots, m_n$$

删除后, S栈成为:

$$m_1, \dots, m_{n-p}$$

② T栈中 $x_1x_2\dots x_p$ 归约成A, 即T栈栈顶删除p个文法符号,
非终极符A进T栈;

③ 若 $GOTO[m_{n-p}, A]=j$, 则状态j进S栈。

(3) 接受: 宣布分析成功, 分析器停止工作。

当S栈顶状态为k, 现行输入符号为a, 总控程序根据“k”和“a”查LR分析表得:

$$\text{ACTION}[k, a] = \underline{\underline{\text{acc}}}$$

acc说明语法分析成功。

(4) 报错: 报告发现源程序有错, 调用出错处理程序。

总控程序若按“k”和“a”查表得:

$$\text{ACTION}[k, a] = \underline{\underline{\text{空白}}}$$

说明语法分析出错, 所给输入串不是本文法的句子。

LR分析器总控程序的工作十分简单,它的任一步只需按分析栈栈顶状态s和现行输入符a执行ACTION[s,a]所规定的动作。

LR分析器的工作过程可看成是栈中状态序列、已归约串和输入串构成的三元式的变化过程。



LR分析算法

- 置ip指向输入串w的第一个符号
 - 令S为栈顶状态
 - a是ip指向的符号
 - 重复 begin
 - if ACTION[S,a]=S_j
 - then begin PUSH j,a(进栈)
 - ip 前进(指向下一输入符号)
 - end
 - else if ACTION[S,a]=r_j (第j条产生式为A→β)



LR分析程序

- then begin
 - pop $|\beta|$ 项
 - 令当前栈顶状态为 S'
 - push GOTO[S' ,A]和A(进栈)
- end
- else if ACTION[s,a]=acc
 - then return (成功)
 - else error
- end.重复

其中, $S_j = \text{GOTO}[S_i, X]$ 表示当栈顶状态为 S_i 遇到当前文法符号为 X 时应转向状态 S_j

例: $G[S]: S \rightarrow a A c B e$ [1] $A \rightarrow b$ [2]
 $A \rightarrow Ab$ [3] $B \rightarrow d$ [4]

[illegible]

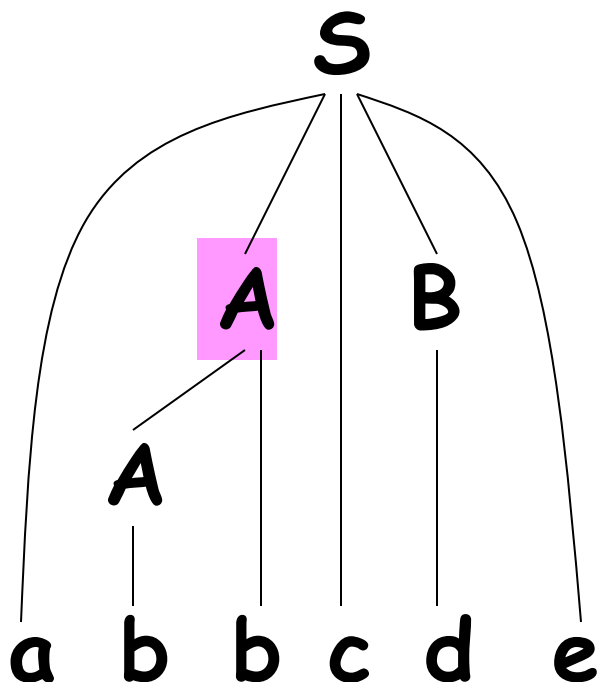
文法 $G[S]$:

(1) $S \rightarrow aAcBe$

(2) $A \rightarrow b$

(3) $A \rightarrow Ab$

(4) $B \rightarrow d$



$S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \Rightarrow abbcde$

步骤	符号栈	输入符号串	动作
1)	#	abbcde#	移进
2)	#a	bbcde#	移进
3)	#ab	bcde#	归约($A \rightarrow b$)
4)	#aA	bcde#	移进
5)	#aAb	cde#	归约($A \rightarrow Ab$)
6)	#aA	cde#	移进
7)	#aAc	de#	移进
8)	#aAcd	e#	归约($B \rightarrow d$)
9)	#aAcB	e#	移进
10)	#aAcBe	#	归约
11)	#S	#	接受

用(2)
用(3)?
用LR分析表
就没有
这种状况

最基本论

对输入串abbcde#的移进-规约分析过程

字符串abbcde是否是 $G[S]$ 的子



LR分析法的主要任务：构造一张LR分析表

首先讨论一种只概括“历史”资料而不包含推测性
“展望”材料的“状态”。

希望仅由这种简单状态就能识别呈现在栈顶
的某些句柄。

LR(0)项目集就是这样一种简单状态。



两种LR(0)分析表的构造算法：

方法一：先构造¹识别文法的活前缀非确定有限自动机^{NFA}，然后²确定化^{DFA}，再³构造LR(0)分析表；

方法二：是直接构造LR(0)项目集规范族，再构造LR(0)分析表。



6.2 LR(0) 分析

LR(0)文法

能力最弱，理论上最重要

- 存在FA 识别活前缀
- 识别活前缀的DFA如何构造
(LR(0)项目集规范族的构造)
- LR(0)分析表的构造

6.2 LR(0) 分析

构造LR分析表的预备知识

■ LR文法

- 对于一个上下文无关文法, 如果能够构造一张 LR 分析表, 使得它的每一个入口均是唯一的 $(S_j, r_j, \text{acc}, \text{空白})$, 则称该上下文无关是LR 文法.

■ 活前缀

- 规范句型的前缀, 若不含句柄以后的任何符号, 则称它为该规范句型的活前缀。

对原文法扩充 $(S \rightarrow S')$ 增加的终结符

$$G = (V_n, V_t, P, S), \text{ 若有 } S' \xRightarrow{*R} \alpha A w \xRightarrow{R} \alpha \beta w$$

最右推导

α 是 $\alpha\beta$ 的前缀, 则称是文法 G 的活前缀.



6.2.1 可归前缀和子前缀

- 最右推导和最左规约的关系
- 对于文法**G[S]**的每个产生式编上序号**i**
- **$S \rightarrow a A c B e[1]$**
- **$A \rightarrow b[2]$**
- **$A \rightarrow Ab[3]$**
- **$B \rightarrow d[4]$**

$G[S]: S \rightarrow a A c B e [1]$

$A \rightarrow b[2]$

$A \rightarrow Ab[3]$

$B \rightarrow d[4]$

对于合法的句子:

- 每次规范归约后得到的是文法的规范句型
- 用哪个产生式归约取决于当前句型的前部

符号串**abbcde**的最右推导过程

$S \Rightarrow aAcBe[1] \Rightarrow aAcd[4]e[1] \Rightarrow aAb[3]cd[4]e[1]$

$\Rightarrow ab[2]b[3]cd[4]e[1]$

符号串**abbcde**的最左归约过程

$ab[2]b[3]cd[4]e[1]$

$\leq aAb[3]cd[4]e[1]$

$\leq aAcd[4]e[1]$

$\leq aAcBe[1]$

$\leq S$

每次归约前句型的前部:

$ab[2]$

$aAb[3]$

$aAcd[4]$

$aAcBe[1]$

对应归约前符号栈中的符号串

可归前缀: 规范句型的这种前部



6.2.1 可归前缀和子前缀

- 活前缀：在规范句型中形成可归前缀之前包括可归前缀在内的所有前缀称为活前缀
- **aAb: ϵ , a, aA, aAb**
- 在规范归约的任何时刻只要已分析的部分为规范句型的活前缀，则该部分是某规范句型的一个正确部分。



6.2.1 可归前缀和子前缀

活前缀的形式定义：

若 $S' \Rightarrow \alpha A \omega \Rightarrow \alpha \beta \omega$ 是文法中的一个规范推导。

如果 γ 是 $\alpha \beta$ 的前缀，则称 γ 是 G 的一个活前缀。

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。

文法 $G[S]$:

$S \rightarrow aAd$

$A \rightarrow bB$

$B \rightarrow c$

6.2.2 识别活前缀的有限自动机

- **$G[S]$ 的拓广文法：**为使开始符号不出现在产生式的右部，对文法 **G** 进行扩展，增加 **$S' \rightarrow S$** ，其中 **S'** 是对原文法扩充而增加的非终结符。
- **$G[S] = (\{S\}, \{a\}, \{S \rightarrow Sa, S \rightarrow a\}, S)$**
- **$G'[S'] = (\{S, S'\}, \{a\}, \{S \rightarrow Sa, S \rightarrow a, S' \rightarrow S\}, S')$**



6.2.2 识别活前缀的有限自动机

- **LR分析过程:**
 - 不是直接分析文法符号栈中是否形成句柄
 - 而是识别符号栈中是否形成可归前缀，相当于形成句柄
 - 把终结符和非终结符看成是一个有限自动机的输入符号，每把一个符号入栈看成已识别过了该符号，而状态进行转换

G[S]: $S \rightarrow a A c B e$ [1]

$A \rightarrow b$ [2]

$A \rightarrow Ab$ [3]

$B \rightarrow d$ [4]

对该文法进行拓广:

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

$S \rightarrow a A c B e$ [1]

$A \rightarrow b$ [2]

$A \rightarrow Ab$ [3]

$B \rightarrow d$ [4]

现对句子**abbcde**的可归前缀列出:

ab[2]

aAb[3]

aAcd[4]

aAcBe[1]

S[0]

现对句子**abbcde**的可归前缀列出：

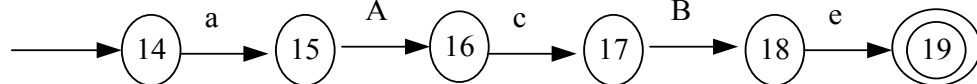
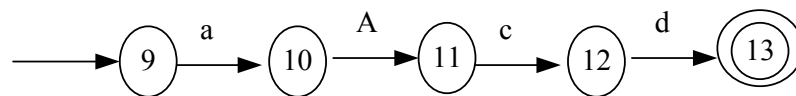
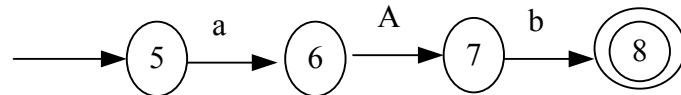
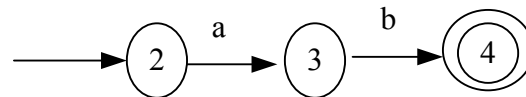
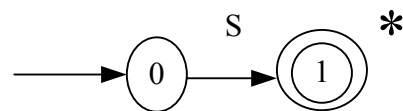
S[0]

ab[2]

aAb[3]

aAcd[4]

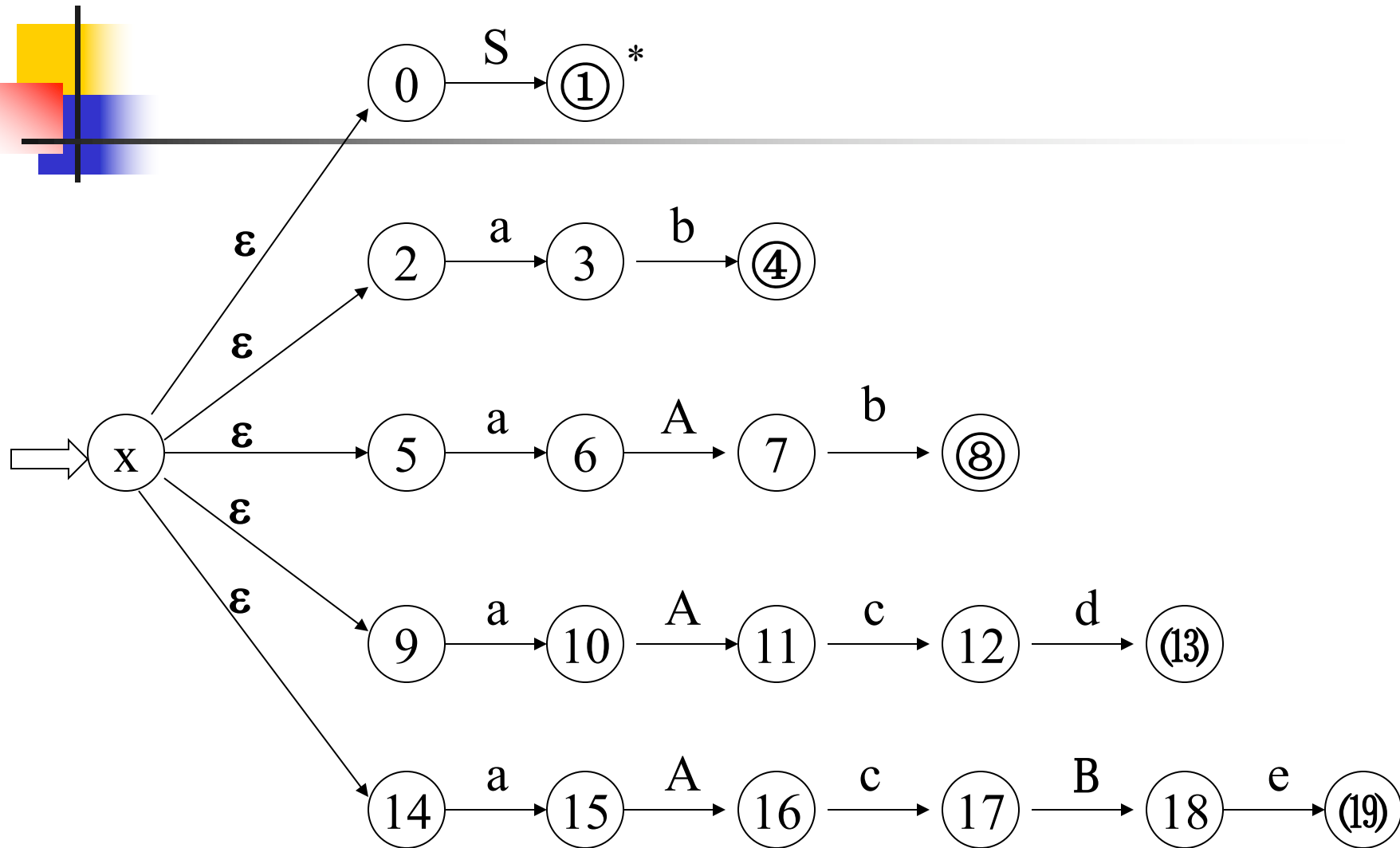
aAcBe[1]



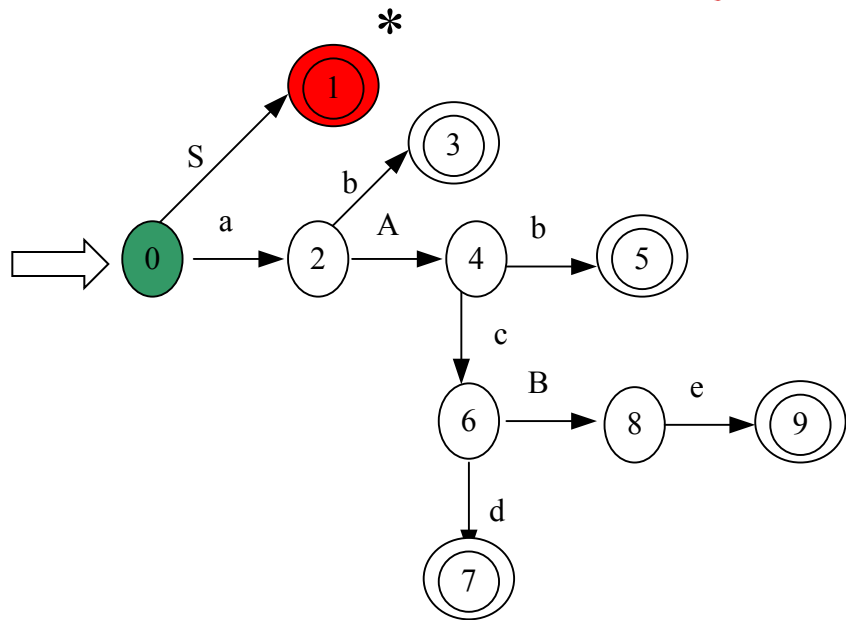
可构造识别其活前缀

及可归前缀的有限自动机

识别活前缀的有限自动机



➤将上面得到的**NFA**进行确定化： ?



识别活前缀的**DFA**

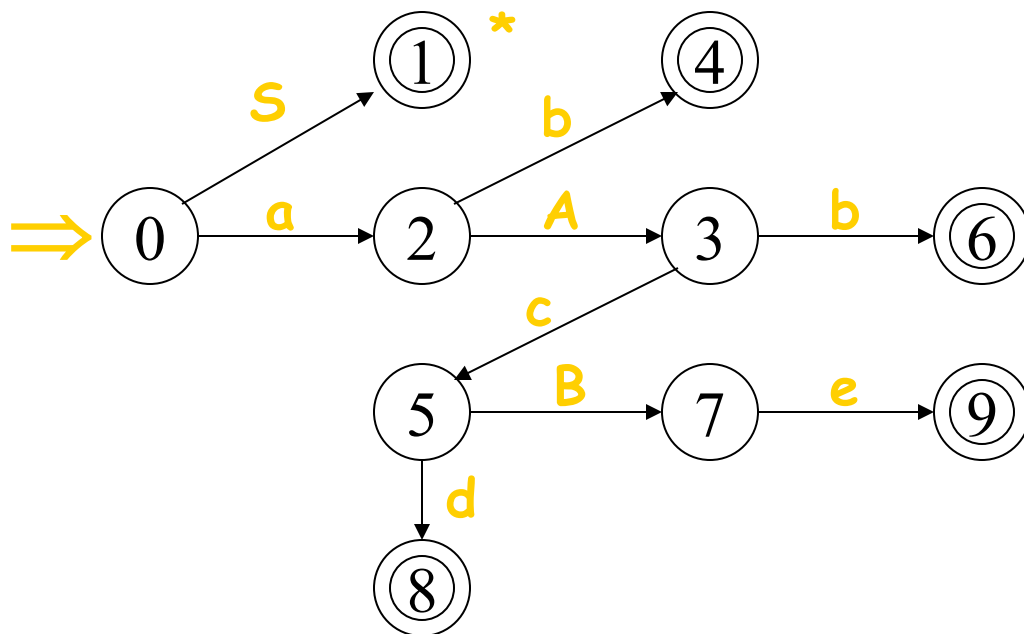


LR分析需要构造识别活前缀的有穷自动机

我们可以文法的终结符和非终结符都看成有穷自动机的输入符号，每次把一个符号进栈看成已识别过了该符号，同时状态进行转换，当识别到可归前缀时，相当于在栈中形成句柄，认为达到了识别句柄的终态。

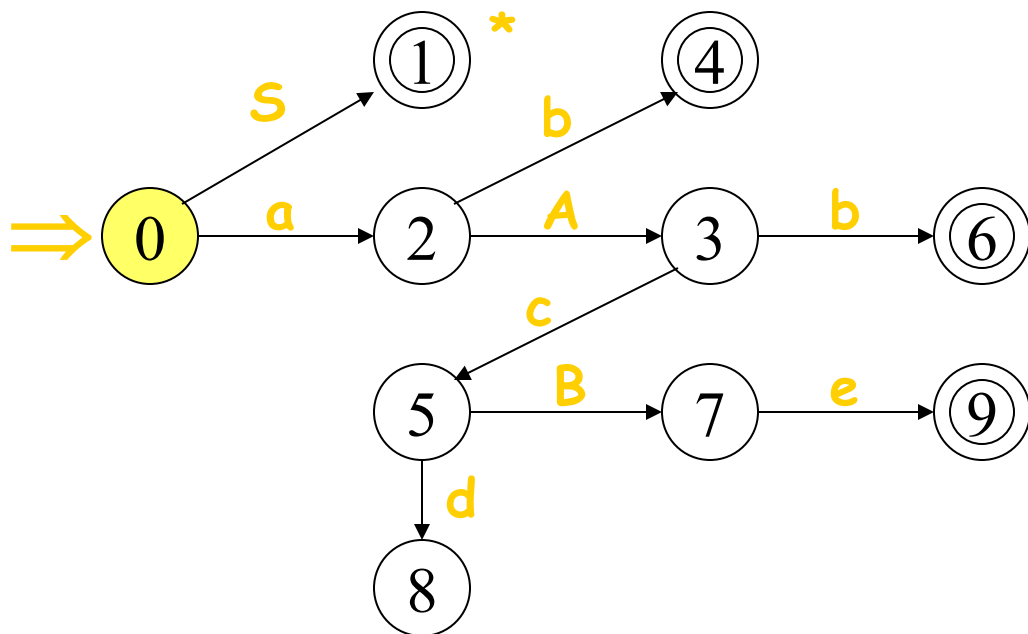
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3
4)	#aA	bcde#	移进	023	S ₆	
5)	#aAb	cde#	归约(A→Ab)	0236	r ₃	3
6)	#aA	cde#	移进	023	S ₅	
7)	#aAc	de#	移进	0235	S ₈	
8)	# aAcd	e#	归约(B→d)	02358	r ₄	7
9)	#aAcB	e#	移进	02357	S ₉	
10)	#aAcBe	#	归约(S→aAcBe)	023579	r ₁	1
11)	#S	#	接受	01	acc	

对输入串abbcde#的LR分析过程



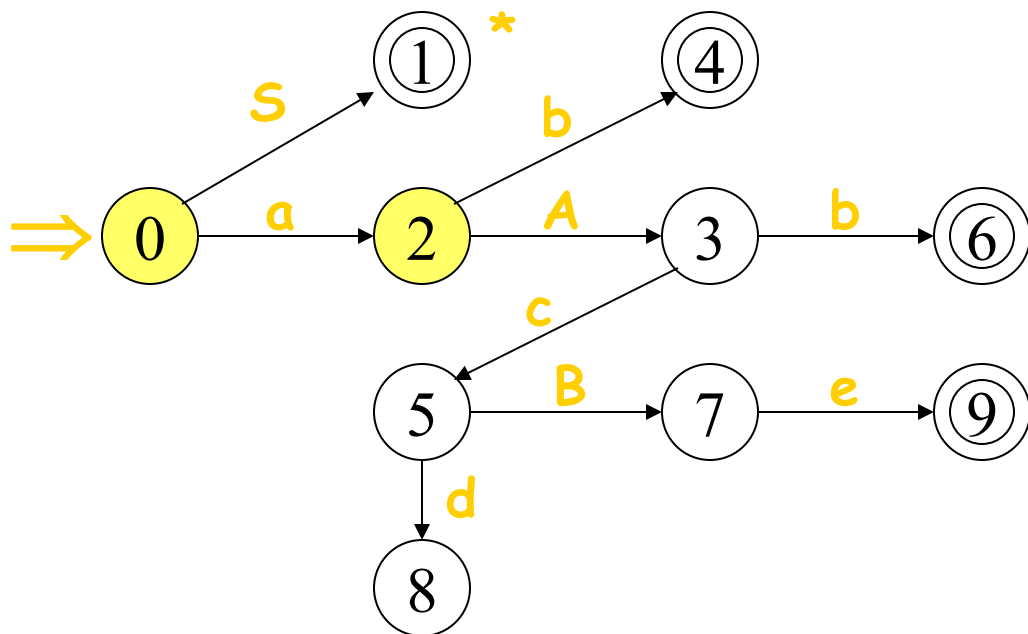
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S_2	

对输入串abbcde#的LR分析过程



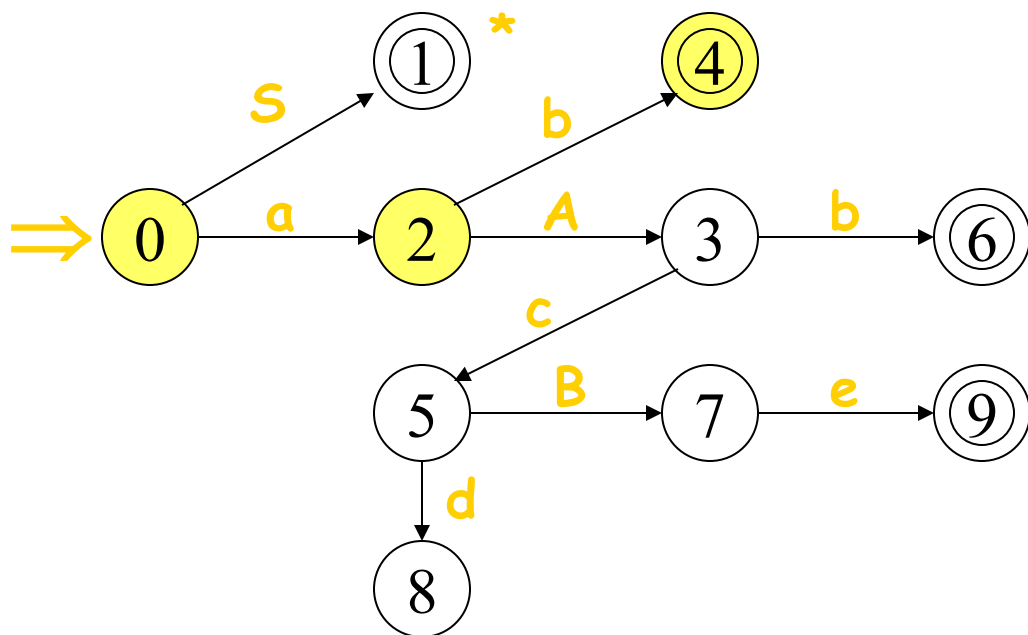
步骤	符号栈	输入字符串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	

对输入串abbcde#的LR分析过程



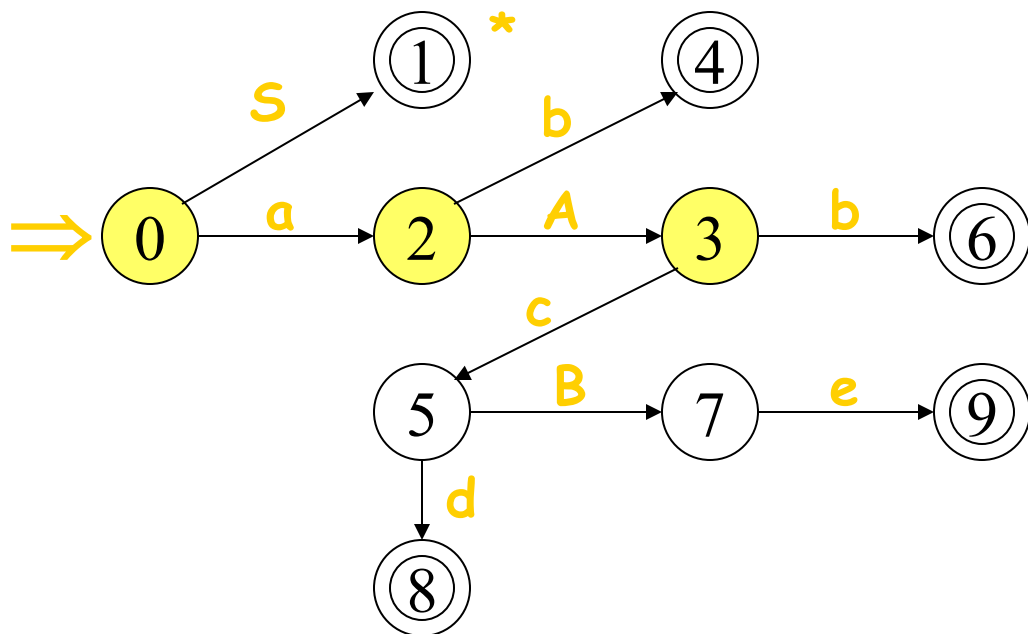
步骤	符号栈	输入字符串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3

对输入串abbcde#的LR分析过程



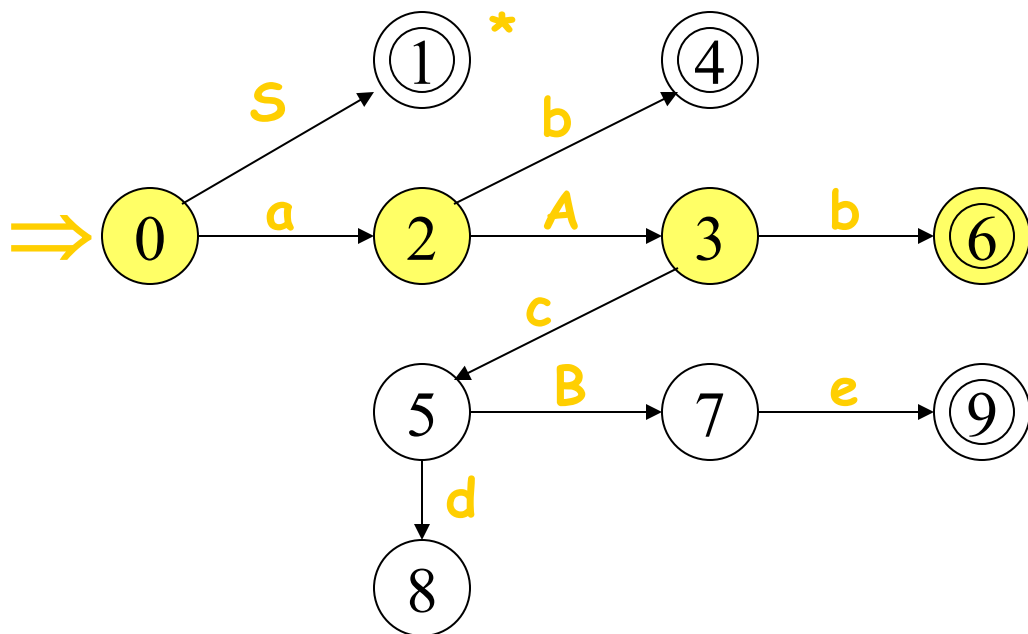
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3
4)	#aA	bcde#	移进	023	S ₆	

对输入串abbcde#的LR分析过程



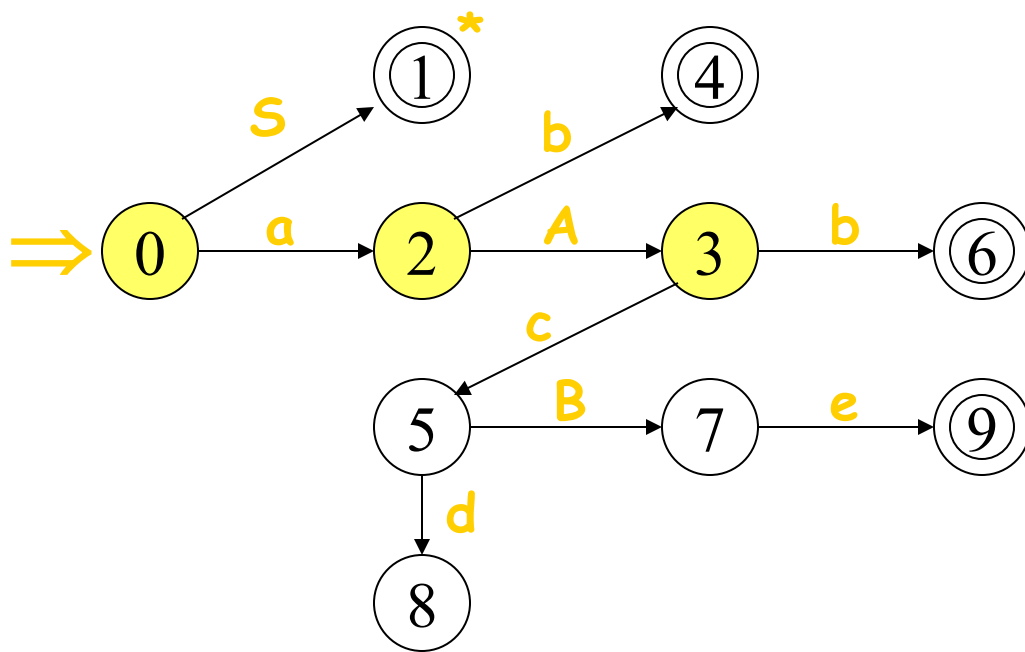
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3
4)	#aA	bcde#	移进	023	S ₆	
5)	#aAb	cde#	归约(A→Ab)	0236	r ₃	3

对输入串abbcde#的LR分析过程



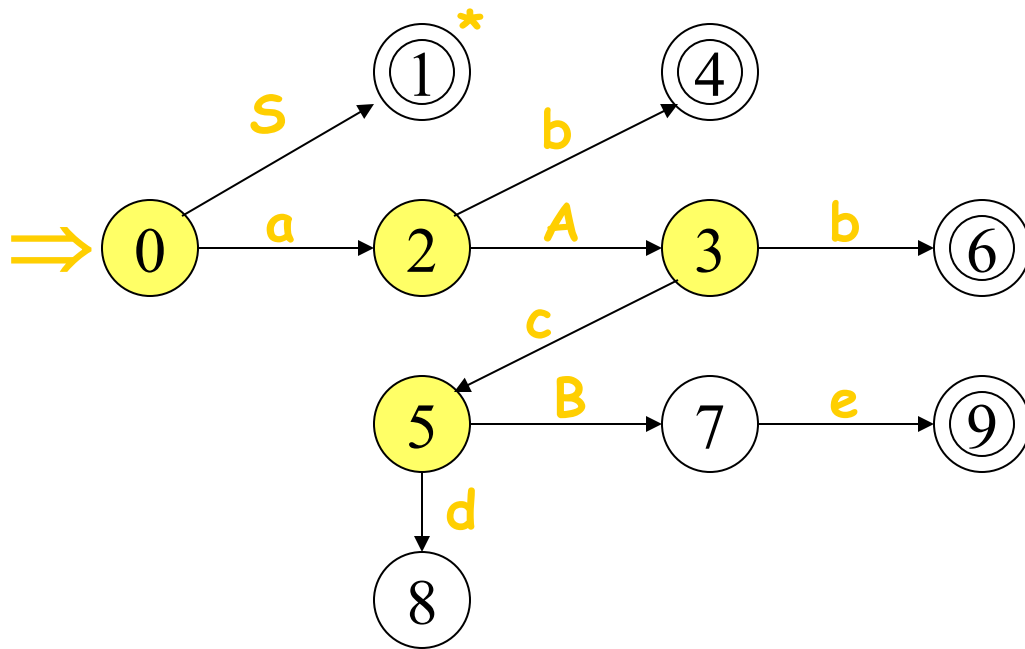
步骤	符号栈	输入字符串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3
4)	#aA	bcde#	移进	023	S ₆	
5)	#aAb	cde#	归约(A→Ab)	0236	r ₃	3
6)	#aA	cde#	移进	023	S ₅	

对输入串abbcde#的LR分析过程



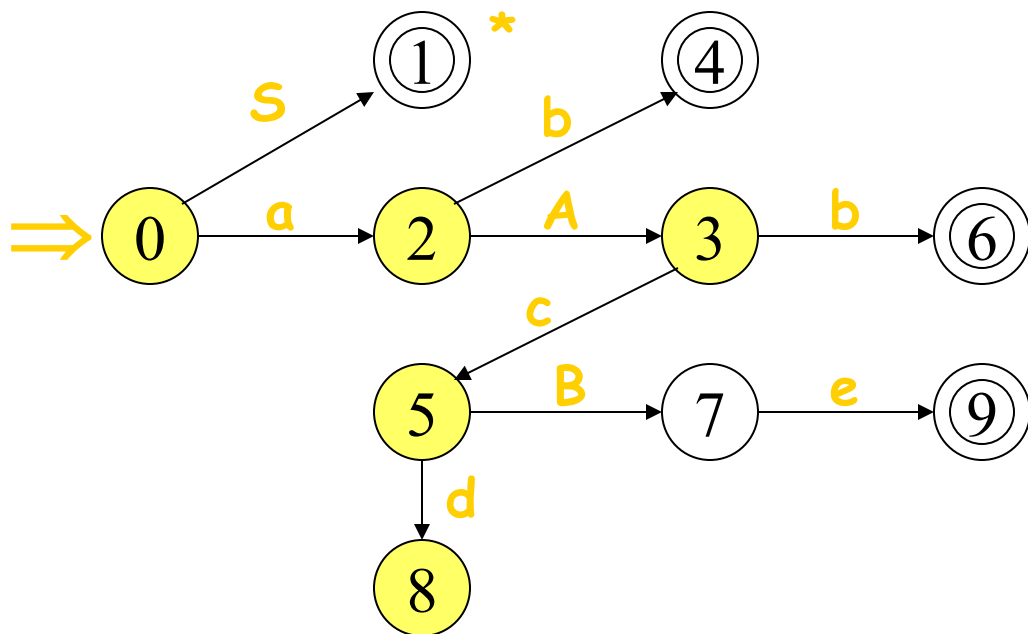
步骤	符号栈	输入字符串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3
4)	#aA	bcde#	移进	023	S ₆	
5)	#aAb	cde#	归约(A→Ab)	0236	r ₃	3
6)	#aA	cde#	移进	023	S ₅	
7)	#aAc	de#	移进	0235	S ₈	

对输入串abbcde#的LR分析过程



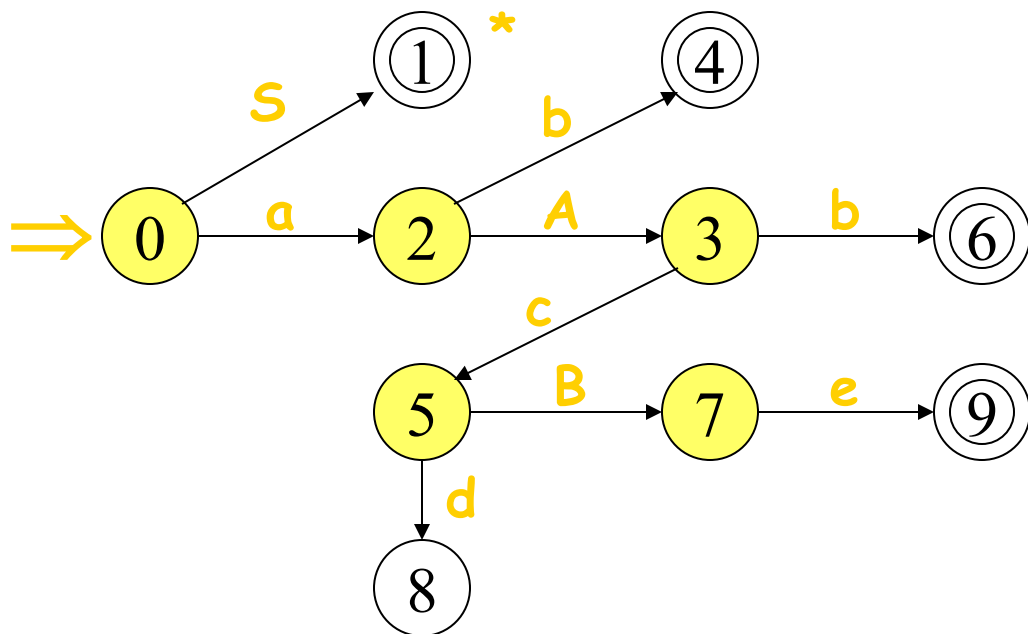
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3
4)	#aA	bcde#	移进	023	S ₆	
5)	#aAb	cde#	归约(A→Ab)	0236	r ₃	3
6)	#aA	cde#	移进	023	S ₅	
7)	#aAc	de#	移进	0235	S ₈	
8)	# aAcd	e#	归约(B→d)	02358	r ₄	7

对输入串abbcde#的LR分析过程



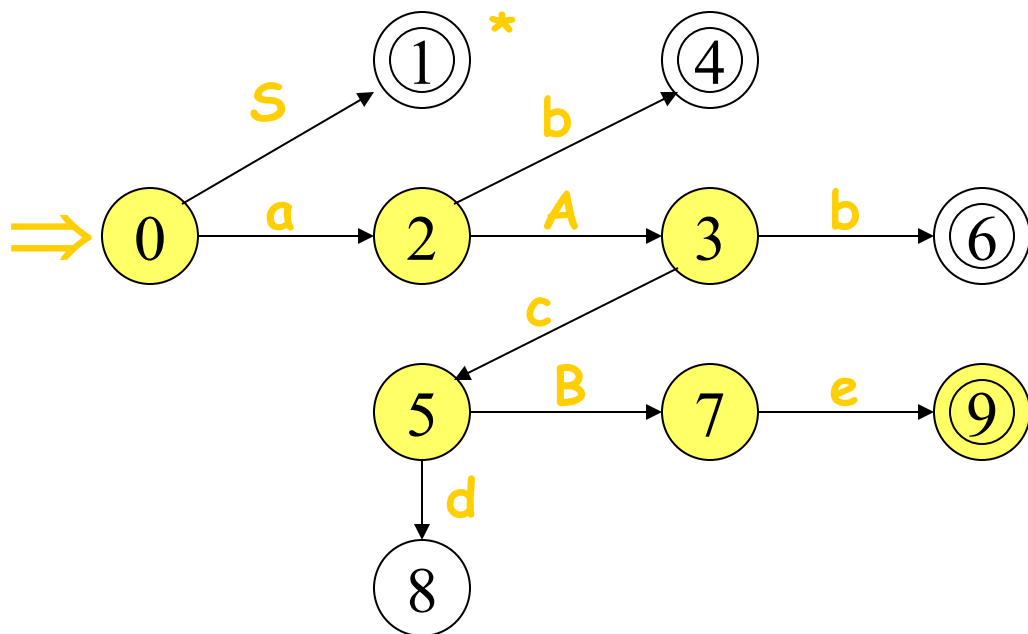
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S_2	
2)	#a	bbcde#	移进	02	S_4	
3)	#ab	bcde#	归约($A \rightarrow b$)	024	r_2	3
4)	#aA	bcde#	移进	023	S_6	
5)	#aAb	cde#	归约($A \rightarrow Ab$)	0236	r_3	3
6)	#aA	cde#	移进	023	S_5	
7)	#aAc	de#	移进	0235	S_8	
8)	# aAcd	e#	归约($B \rightarrow d$)	02358	r_4	7
9)	#aAcB	e#	移进	02357	S_9	

对输入串abbcde#的LR分析过程



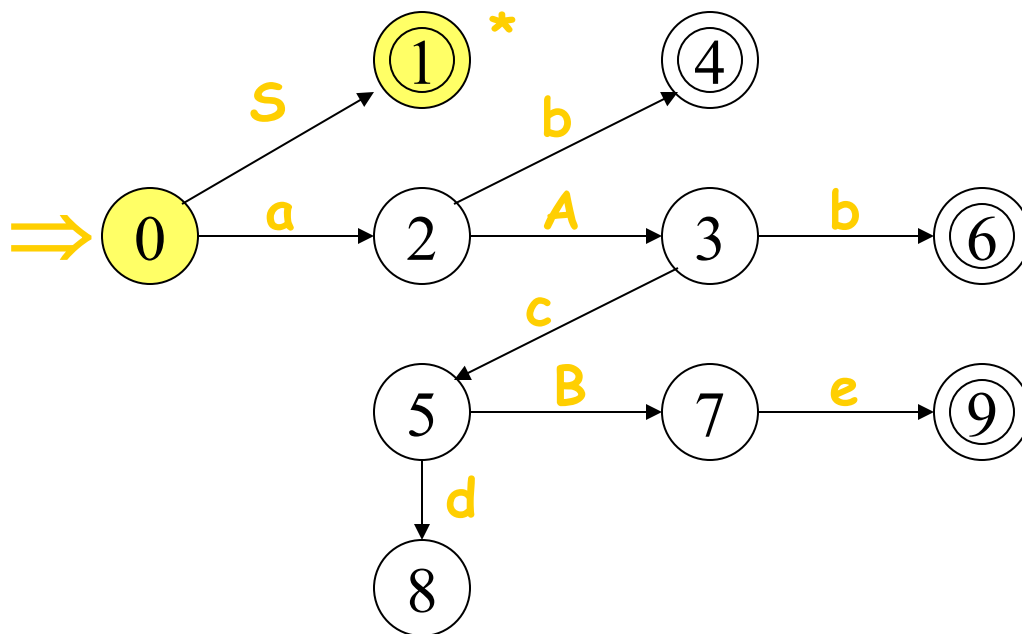
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S_2	
2)	#a	bbcde#	移进	02	S_4	
3)	#ab	bcde#	归约($A \rightarrow b$)	024	r_2	3
4)	#aA	bcde#	移进	023	S_6	
5)	#aAb	cde#	归约($A \rightarrow Ab$)	0236	r_3	3
6)	#aA	cde#	移进	023	S_5	
7)	#aAc	de#	移进	0235	S_8	
8)	# aAcd	e#	归约($B \rightarrow d$)	02358	r_4	7
9)	#aAcB	e#	移进	02357	S_9	
10)	#aAcBe	#	归约($S \rightarrow aAcBe$)	023579	r_1	1

对输入串abbcde#的LR分析过程



步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1)	#	abbcde#	移进	0	S ₂	
2)	#a	bbcde#	移进	02	S ₄	
3)	#ab	bcde#	归约(A→b)	024	r ₂	3
4)	#aA	bcde#	移进	023	S ₆	
5)	#aAb	cde#	归约(A→Ab)	0236	r ₃	3
6)	#aA	cde#	移进	023	S ₅	
7)	#aAc	de#	移进	0235	S ₈	
8)	# aAcd	e#	归约(B→d)	02358	r ₄	7
9)	#aAcB	e#	移进	02357	S ₉	
10)	#aAcBe	#	归约(S→aAcBe)	023579	r ₁	1
11)	#S	#	接受	01	acc	

对输入串abbcde#的LR分析过程



借助自动机
复现

构造识别文法活前缀DFA的三种方法

一、根据形式定义求出活前缀的正规表达式，然后由此正规表达式构造NFA再确定化为DFA 词法分析

二、求出文法的所有项目，按一定规则构造识别活前缀的NFA再确定化为DFA

三、使用闭包函数（CLOSURE）和转向函数（GOTO(I,X)）构造文法G'的LR(0)的[项目集规范族]，再由转换函数建立状态之间的连接关系得到识别活前缀的DFA

↓
构成识别一个文法的活前缀的DFA的状态的全体



6.2.4 LR(0)项目集规范族的构造

(1) LR(0)项目

- 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:
 $A \rightarrow .XYZ$ $A \rightarrow X.YZ$ $A \rightarrow XY.Z$ $A \rightarrow XYZ.$
- 圆点的左部表示分析过程的某一时刻欲用该产生式归约时已识别过的句柄部分，圆点的右部表示期待的后缀部分。

LR(0) 分析

活前缀与句柄的关系：

- $G[S]$:
- 若 $S \xRightarrow[R]{*} \alpha A \omega \xRightarrow[R]{} \alpha \beta \omega$ r 是 $\alpha\beta$ 的前缀，则
- 称 r 是 G 的一个活前缀
- 1. 活前缀已含有句柄的全部符号，表明产生式 $A \rightarrow \beta$ 的右部 β 已出现在栈顶
- 2. 活前缀只含句柄的一部分符号表明 $A \rightarrow \beta_1 \beta_2$ 的右部子串 β_1 已出现在栈顶，期待从输入串中看到 β_2 推出的符号
- 3. 活前缀不含有句柄的任何符号，此时期望 $A \rightarrow \beta$ 的右部所推出的符号串

LR(0)项目

根据圆点所在的位置和圆点后是终结符还是非终结符或为空把项目分为以下几种：

移进项目，形如 $A \rightarrow \alpha \cdot a \beta$ a 是终结符, $\alpha, \beta \in V^*$ 以下同

待约项目，形如 $A \rightarrow \alpha \cdot B \beta$

归约项目，形如 $A \rightarrow \alpha \cdot$

接受项目，形如 $S' \rightarrow S \cdot$

$A \rightarrow \epsilon$ 的LR(0)项目只有 $A \rightarrow \cdot$ 是归约项目



6.2.4 LR(0)项目集规范族的构造

- 根据圆点所在的位置和圆点后是终结符还是非终结符把项目分为：
 - ☞ $A \rightarrow \alpha \cdot$ 称为"归约项目"
 - ☞ 归约项目 $S' \rightarrow \alpha \cdot$ 称为"接受项目"
 - ☞ $A \rightarrow \alpha \cdot a\beta$ ($a \in V_T$) 称为"移进项目"
 - ☞ $A \rightarrow \alpha \cdot B\beta$ ($B \in V_N$) 称为"待约项目".



6.2.4 LR(0)项目集规范族的构造

- (2)构造识别活前缀的**NFA**
 - 把文法的所有产生式的项目都列出，并使每个项目都作为**NFA**的一个状态。
 - 方法：
 - ①若状态*i*为 $X \rightarrow X_1 \dots X_{i-1} \cdot X_i \dots X_n$ ，
状态*j*为 $X \rightarrow X_1 \dots X_{i-1} X_i \cdot X_{i+1} \dots X_n$ ，
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 $X \rightarrow \alpha \cdot A \beta$ ，**A**为非终结符，
则从状态*i*画一条 ϵ 边到所有状态 $A \rightarrow \cdot \gamma$ 。
- 把识别文法所有活前缀的**NFA**确定化。

LR(0)项目集规范族

- 定义1: 如果存在一个规范推导

$S \xRightarrow{*} \alpha A W \Rightarrow \alpha \beta_1 \beta_2 W$, 我们说项目 $A \rightarrow \beta_1 \bullet \beta_2$ 对活前缀 $\gamma = \alpha \beta_1$ 是有效的。

- 定义2: 若项目 $A \rightarrow \alpha \bullet B \beta_1$ 对活前缀 $\gamma = \delta \alpha$ 是有效的, 且 $B \rightarrow \eta$ 是一个产生式, 则项目 $B \rightarrow \bullet \eta$ 对 $\gamma = \delta \alpha$ 也是有效的。
- 定义3: 文法G的某个活前缀r的所有有效项目组成的集合成为r的有效项目集, 文法G的所有有效项目集组成的集合称为G的LR(0)项目集规范族。

■ 文法 $G(S')$

$S' \rightarrow E$

$E \rightarrow aA \mid bB$

$A \rightarrow cA \mid d$

$B \rightarrow cB \mid d$

- 项目1为初态
- 圆点在最后的项目为句柄识别态
- 第一个产生式的句柄识别态为句子识别态

■ 该文法的项目有:

1. $S' \rightarrow \cdot E$

2. $S' \rightarrow E \cdot$

3. $E \rightarrow \cdot aA$

4. $E \rightarrow a \cdot A$

5. $E \rightarrow aA \cdot$

6. $A \rightarrow \cdot cA$

7. $A \rightarrow c \cdot A$

8. $A \rightarrow cA \cdot$

9. $A \rightarrow \cdot d$

10. $A \rightarrow d \cdot$

11. $E \rightarrow \cdot bB$

12. $E \rightarrow b \cdot B$

13. $E \rightarrow bB \cdot$

14. $B \rightarrow \cdot cB$

15. $B \rightarrow c \cdot B$

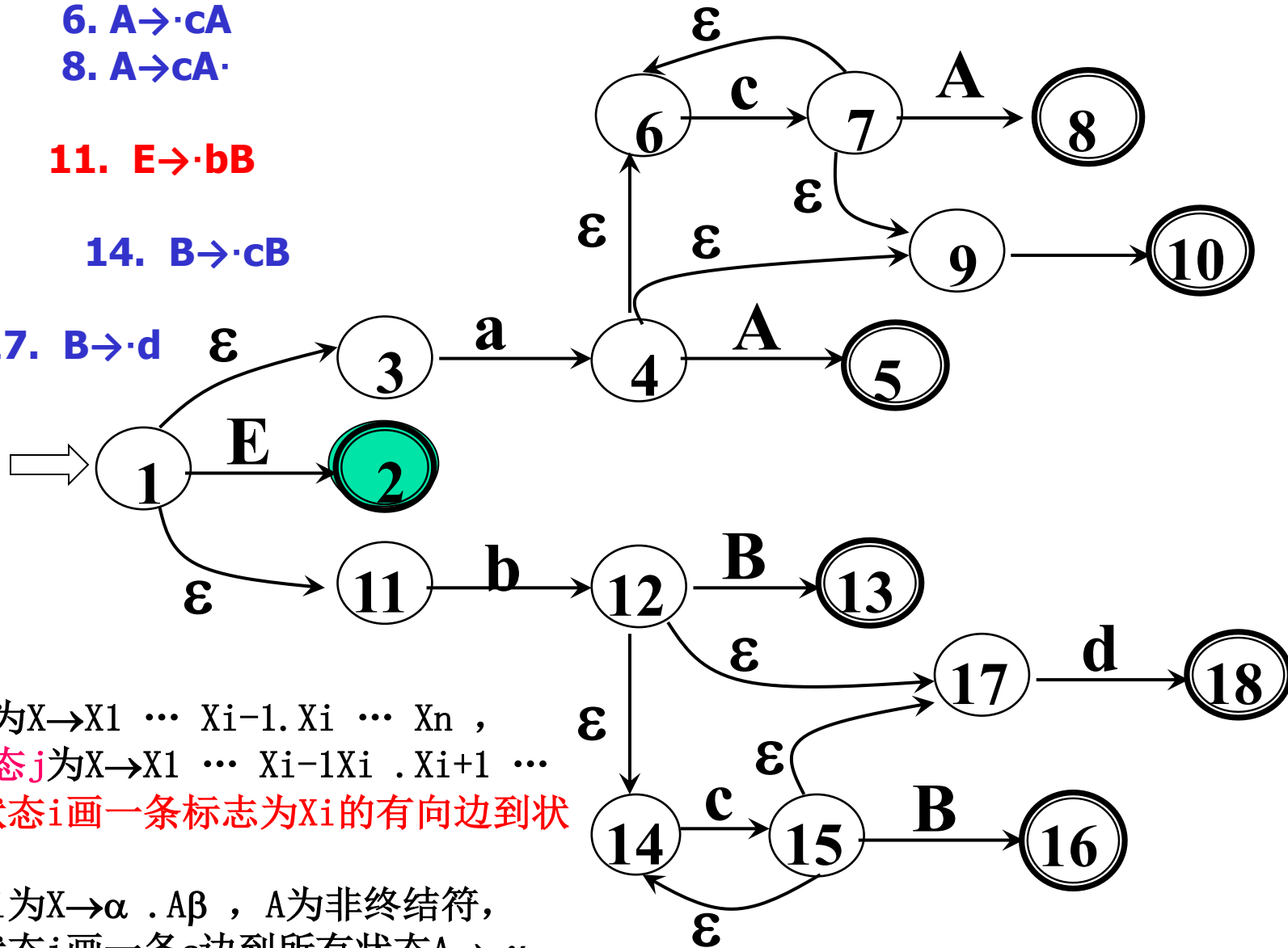
16. $B \rightarrow cB \cdot$

17. $B \rightarrow \cdot d$

18. $B \rightarrow d \cdot$

识别活前缀的NFA

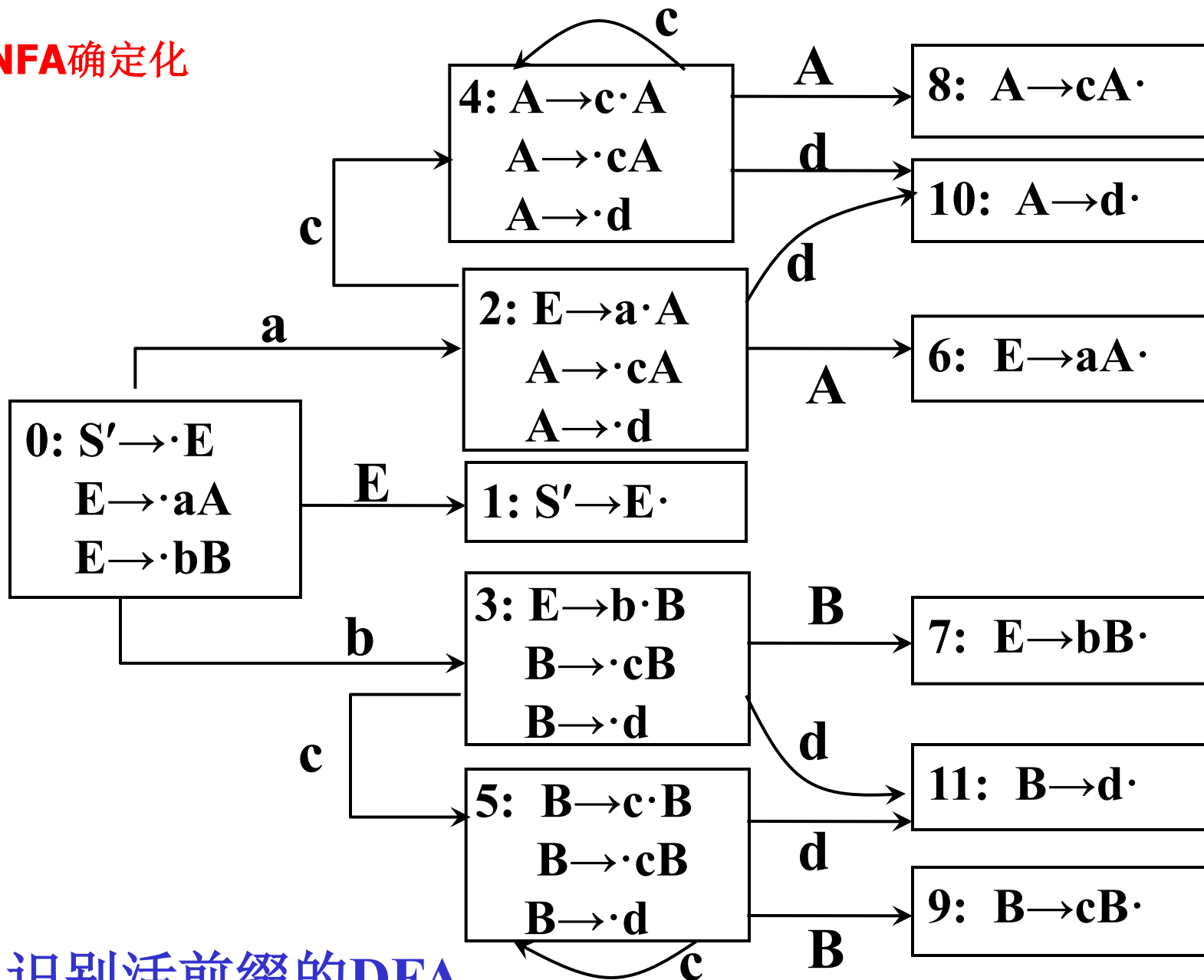
1. $S' \rightarrow \cdot E$
2. $S' \rightarrow E \cdot$
3. $E \rightarrow \cdot aA$
4. $E \rightarrow a \cdot A$
5. $E \rightarrow aA \cdot$
6. $A \rightarrow \cdot cA$
7. $A \rightarrow c \cdot A$
8. $A \rightarrow cA \cdot$
9. $A \rightarrow \cdot d$
10. $A \rightarrow d \cdot$
11. $E \rightarrow \cdot bB$
12. $E \rightarrow b \cdot B$
13. $E \rightarrow bB \cdot$
14. $B \rightarrow \cdot cB$
15. $B \rightarrow c \cdot B$
16. $B \rightarrow cB \cdot$
17. $B \rightarrow \cdot d$
18. $B \rightarrow d \cdot$



方法:

- ①若状态 i 为 $X \rightarrow X_1 \dots X_{i-1} \cdot X_i \dots X_n$,
状态 j 为 $X \rightarrow X_1 \dots X_{i-1} X_i \cdot X_{i+1} \dots X_n$,
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 $X \rightarrow \alpha \cdot A\beta$, A 为非终结符,
则从状态 i 画一条 ϵ 边到所有状态 $A \rightarrow \cdot \gamma$

将**NFA**确定化



识别活前缀的**DFA**

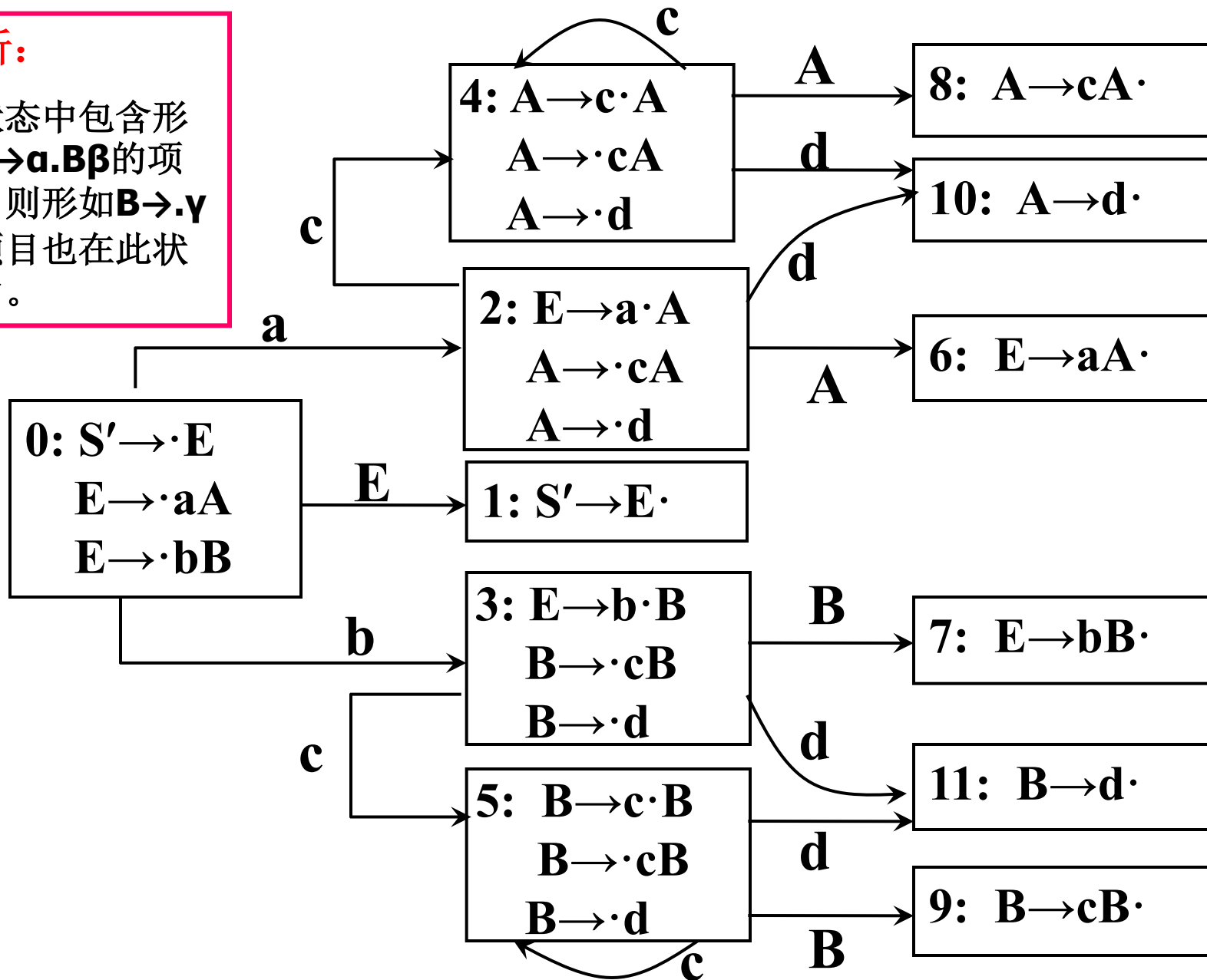


6.2.4 LR(0)项目集规范族的构造

- (3)LR(0)项目集规范族的构造
 - 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**。
 - 上面求**LR(0)项目集规范族**的方法:复杂
 - 列出拓广文法的所有项目
 - 构造**NFA**
 - 确定化

分析:

若状态中包含形如 $A \rightarrow \alpha \cdot B \beta$ 的项目, 则形如 $B \rightarrow \cdot \gamma$ 的项目也在此状态中。





LR(0)项目集规范族的构造

- 假定文法**G**已拓广为**G'**，拓广后增加一个新产生式**S'→S**，而这个**S'**是**G'**的开始符号。如果**I**是文法**G'**的一个项目集，定义和构造**I**的闭包**CLOSURE(I)**如下：
 - **I**的任何项目都属于**CLOSURE(I)**；
 - 若**A→α·Bβ**属于**CLOSURE(I)**，那么 项目**B→·γ**也属于**CLOSURE(I)**；
 - 重复执行上述两步骤直至**CLOSURE(I)** 不再增大为止。
- 这样有了初态的项目集，其他状态的项目集如何求出？

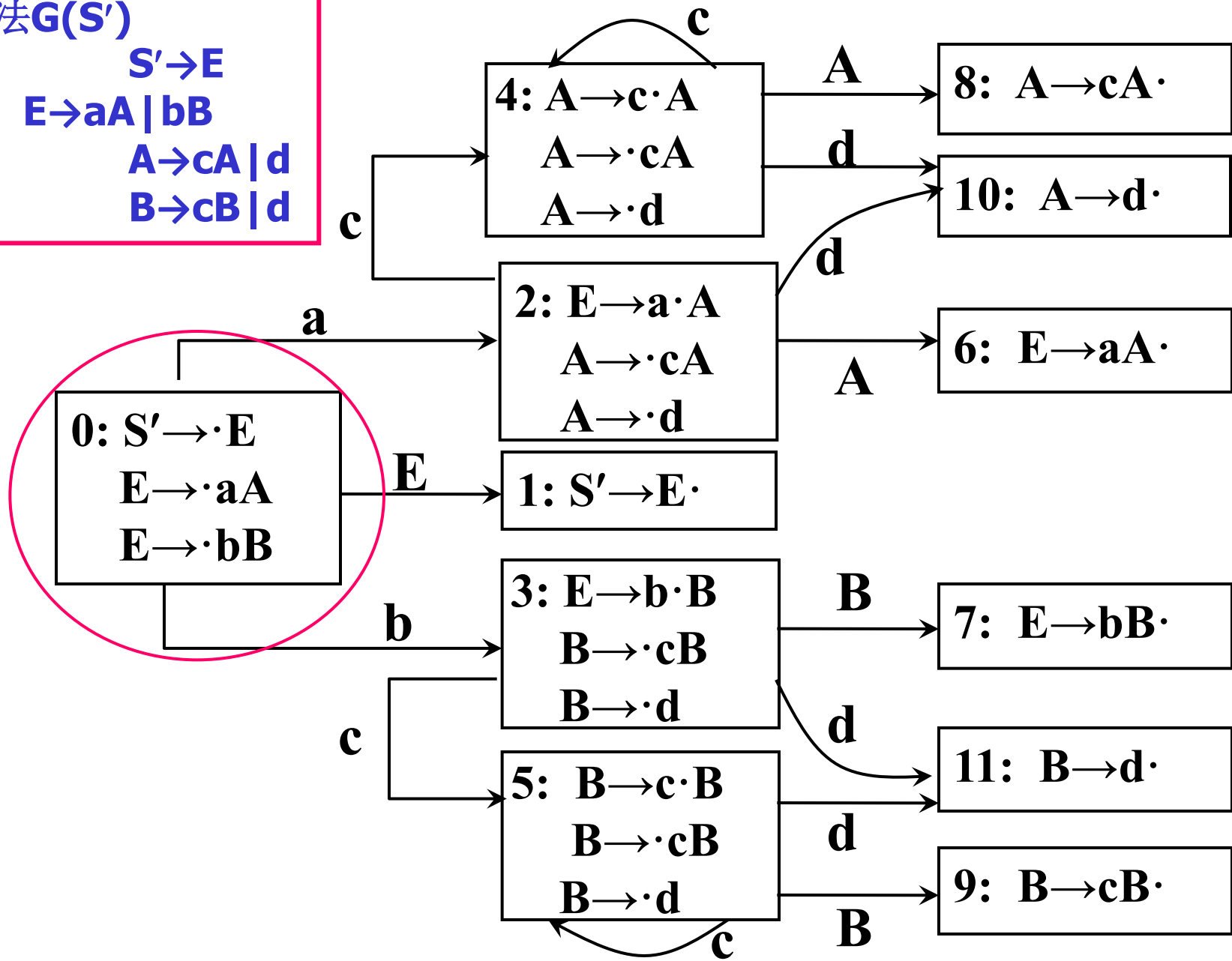
文法G(S')

$S' \rightarrow E$

$E \rightarrow aA \mid bB$

$A \rightarrow cA \mid d$

$B \rightarrow cB \mid d$



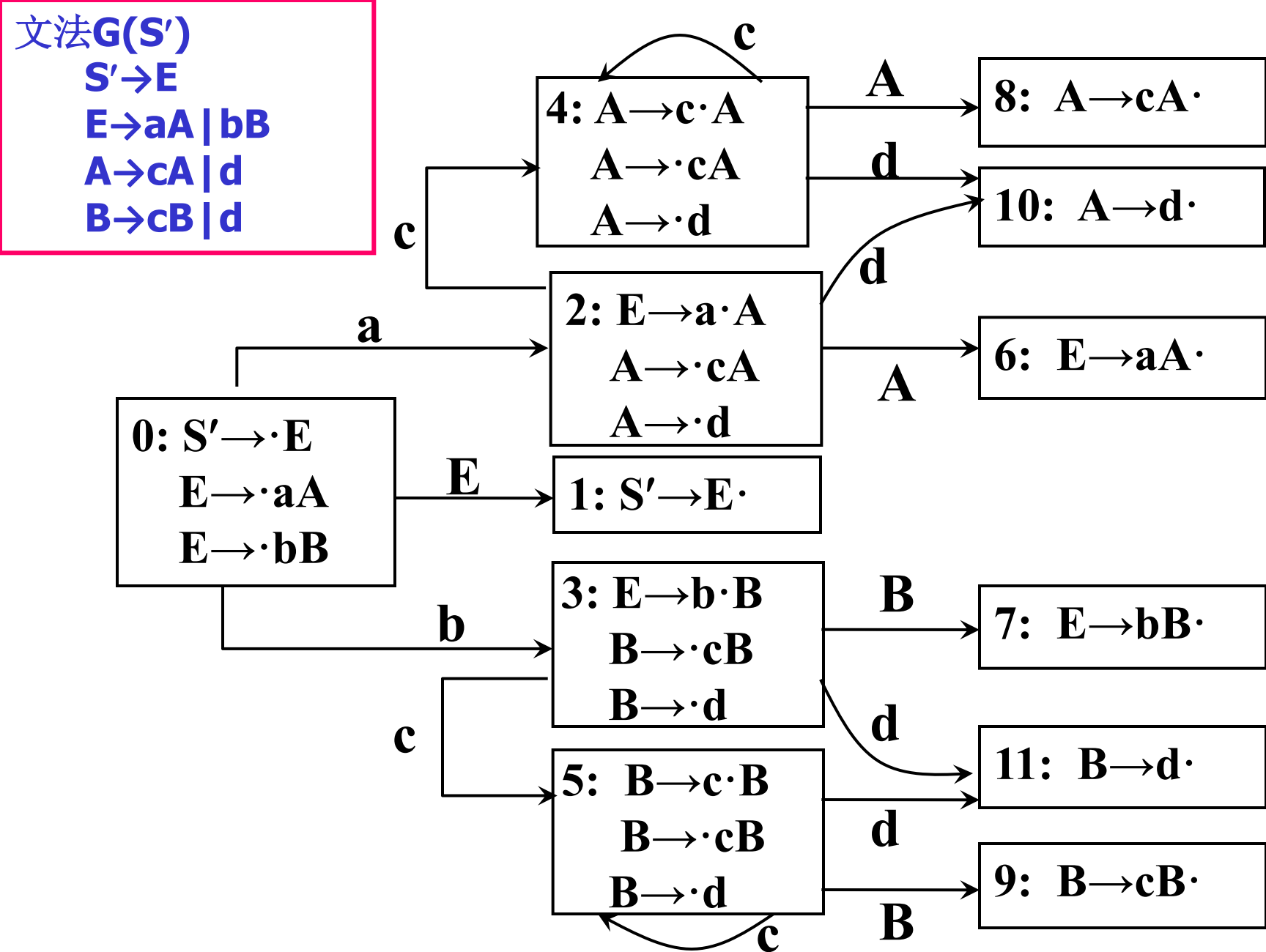
- 为了识别活前缀，我们定义一个状态转换函数GO。I是一个项目集，X是一个文法符号。函数值GO(I, X)定义为：

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

其中

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

- 直观上说，若I是对某个活前缀 γ 有效的项目集，那么，GO(I, X)便是对 γX 有效的项目集。



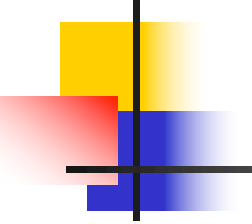


LR(0)文法定义

- 对于一个文法的**LR(0)**项目集规范族不存在移进-归约，或归约-归约冲突的文法称为**LR(0)**文法。
 - 移进-归约冲突：
 - 一个项目集中同时存在形如：
 - **$A \rightarrow \alpha.a\beta$** 和 **$B \rightarrow \gamma.$**
 - 归约-归约冲突：
 - 一个项目集中同时存在形如：
 - **$A \rightarrow \alpha.$** 和 **$B \rightarrow \gamma.$**

LR(0)分析表的构造

- 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 LR(0) 分析表含有状态 $0, 1, \dots, n$ 。令那个含有项目 $S' \rightarrow .S$ 的 I_k 的下标 k 为初态。
ACTION 和 GOTO 可按如下方法构造:
 - 1. 若项目 $A \rightarrow \alpha.$ 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 “ S_j ”;
 - 2. 若项目 $A \rightarrow \alpha.$ 属于 I_k , 那么, 对任何终结符 a 和 “ $\#$ ”, 置 ACTION[k, a] 为 “用产生式 $A \rightarrow \alpha$ 进行归约”, 简记为 “ r_j ”; 其中, 假定 $A \rightarrow \alpha$ 为文法 G' 的第 j 个产生式;



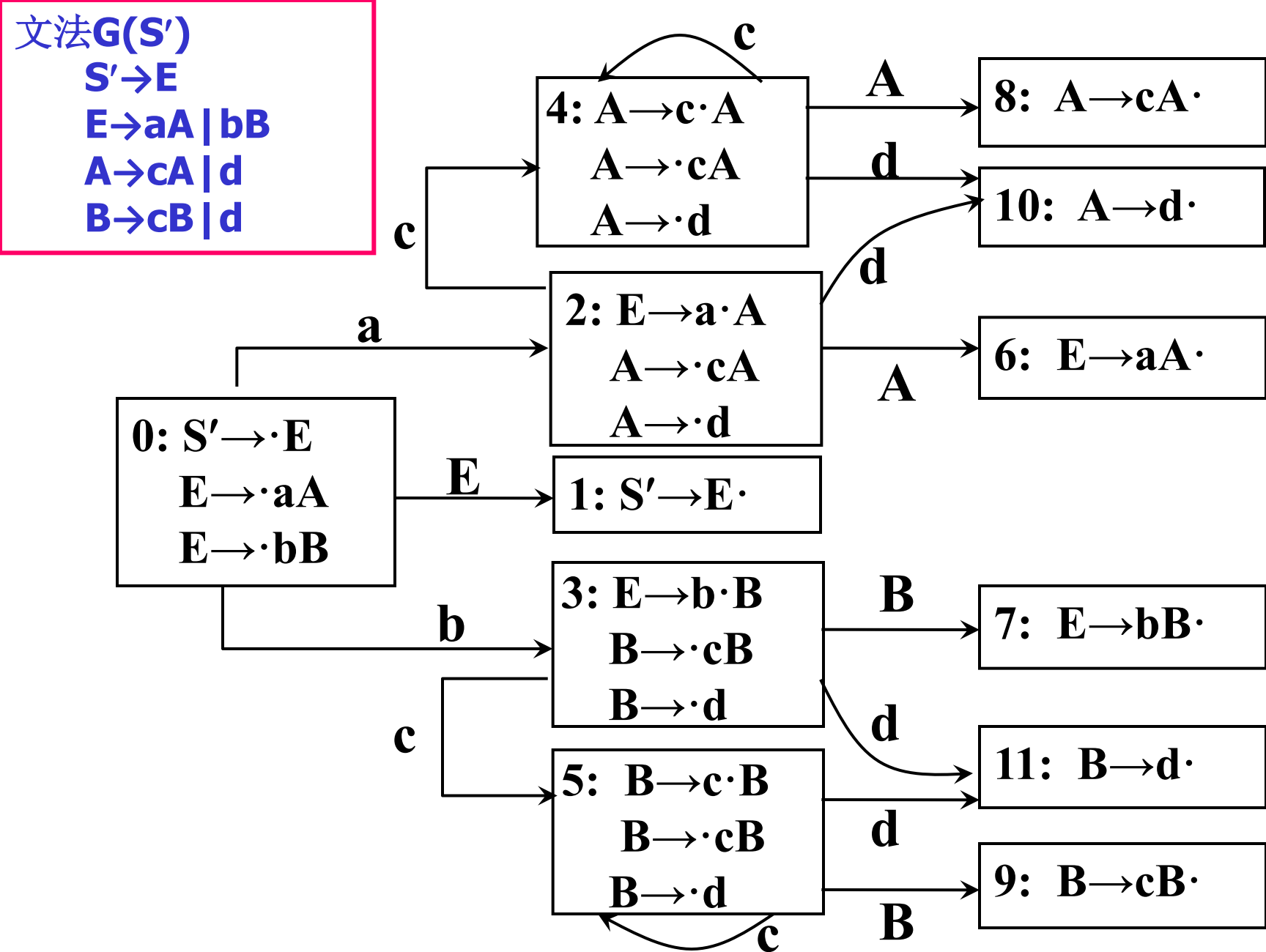
3.若项目 $S' \rightarrow S \cdot$ 属于 I_k , 则置 $ACTION[k, \#]$ 为“接受”, 简记为“acc”;

4.若 $GO(I_k, A) = I_j$, A 为非终结符, 则置 $GOTO[k, A] = j$;

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LR(0)表。

具有LR(0)表的文法G称为一个LR(0)文法。



■ LR(0)分析表为

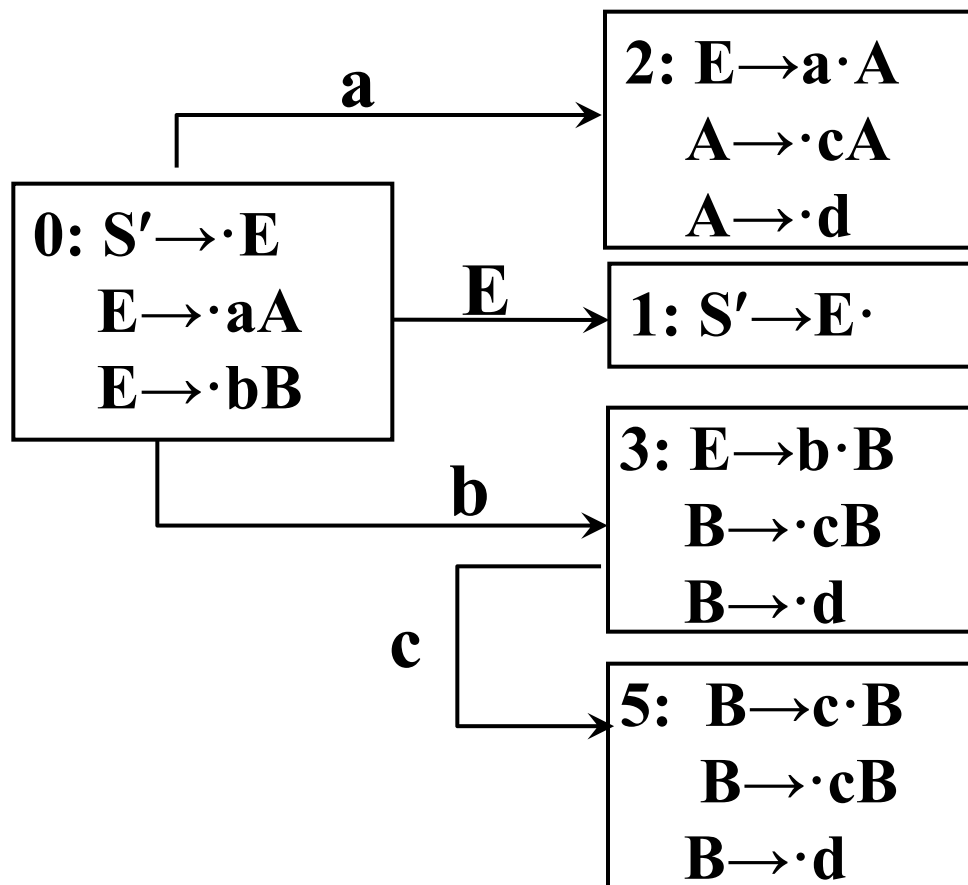
文法**G(S')**

$S' \rightarrow E(0)$

$E \rightarrow aA(1) \mid bB(2)$

$A \rightarrow cA(3) \mid d(4)$

$B \rightarrow cB(5) \mid d(6)$



状态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
0	s2	s3				1		
1					acc			
2			s4	s10			6	
3			s5	s11				7

文法**G(S')**

S'→E(0)

E→aA(1) | bB(2)

A→cA(3) | d(4)

B→cB(5) | d(6)

	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			

	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			

文法G(S')

$S' \rightarrow E(0)$

$E \rightarrow aA(1) \mid bB(2)$

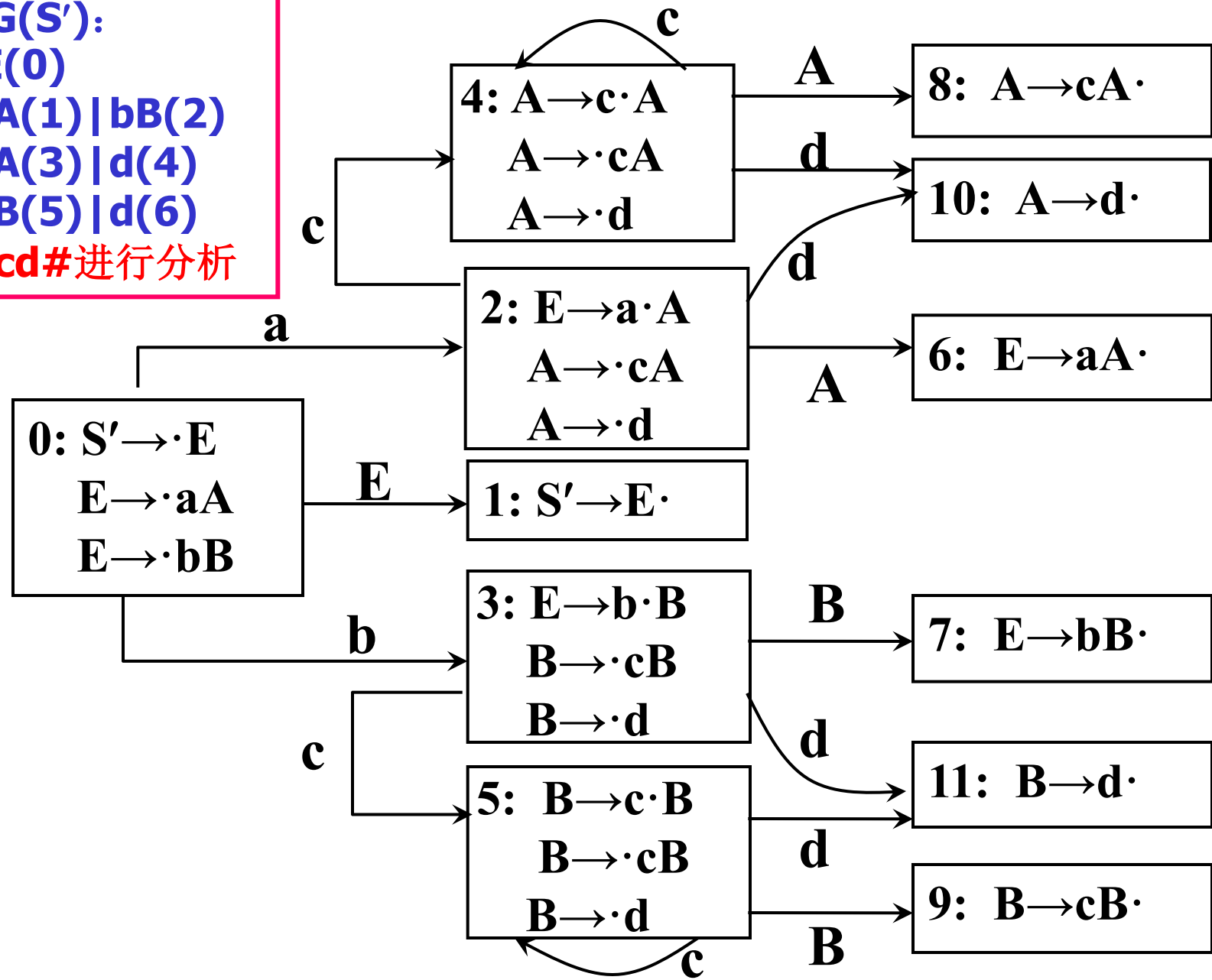
$A \rightarrow cA(3) \mid d(4)$

$B \rightarrow cB(5) \mid d(6)$

对**bccd#**进行分析

步骤	状态栈	符号栈	输入串	ACTION	GOTO
1	0	#	bccd#	S3	
2	03	#b	ccd#	S5	
3	035	#bc	cd#	S5	
4	0355	#bcc	d#	S11	
5	0355(11)	#bccd	#	r6	9
6	03559	#bccB	#	r5	9
7	0359	#bcB	#	r5	7
8	037	#bB	#	r2	1
9	01	#E	#	acc	

文法G(S'):
S'→E(0)
E→aA(1)|bB(2)
A→cA(3)|d(4)
B→cB(5)|d(6)
对**bcccd#**进行分析

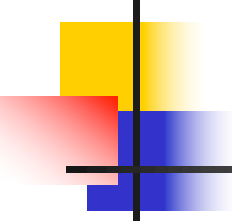




回顾：

LR(0)项目集规范族的构造

- 假定文法**G**已拓广为**G'**，拓广后增加一个新产生式**S'→S**，而这个**S'**是**G'**的开始符号。如果**I**是文法**G'**的一个项目集，定义和构造**I**的闭包**CLOSURE(I)**如下：
 - **I**的任何项目都属于**CLOSURE(I)**；
 - 若**A→α·Bβ**属于**CLOSURE(I)**，那么 项目 **B→·γ**也属于**CLOSURE(I)**；
 - 重复执行上述两步骤直至**CLOSURE(I)** 不再增大为止。

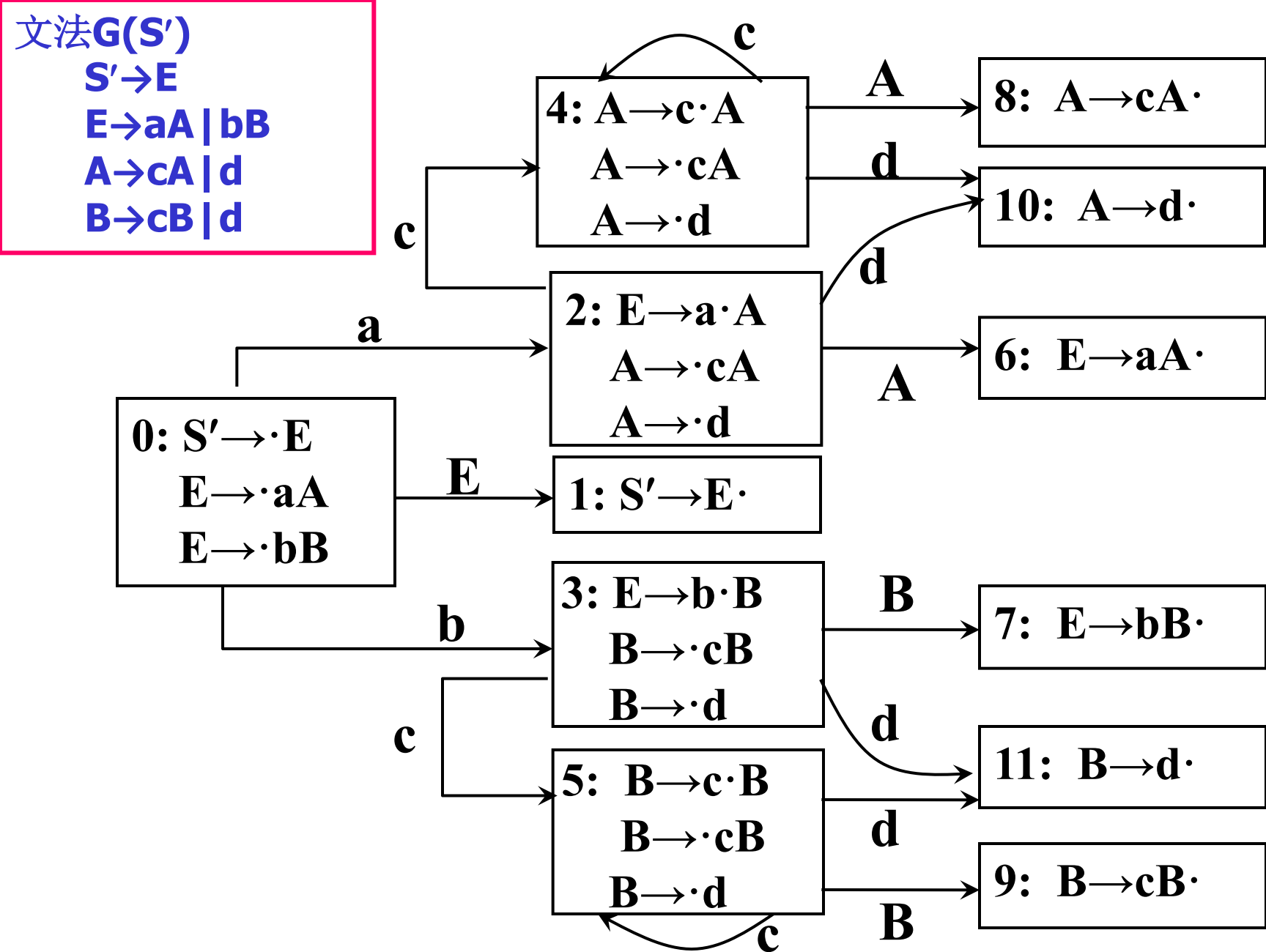
- 
- 为了识别活前缀，我们定义一个状态转换函数GO。I是一个项目集，X是一个文法符号。函数值GO(I, X)定义为：

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

其中

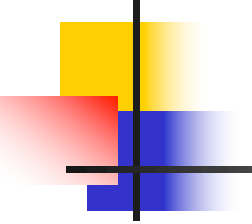
$J = \{\text{任何形如 } A \rightarrow \alpha X \cdot \beta \text{ 的项目} \mid A \rightarrow \alpha \cdot X \beta \text{ 属于 } I\}$ 。

- 直观上说，若I是对某个活前缀 γ 有效的项目集，那么，GO(I, X)便是对 γX 有效的项目集。



LR(0)分析表的构造

- 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 LR(0) 分析表含有状态 $0, 1, \dots, n$ 。令那个含有项目 $S' \rightarrow .S$ 的 I_k 的下标 k 为初态。
ACTION 和 GOTO 可按如下方法构造:
 - 1. 若项目 $A \rightarrow \alpha.$ 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 “ S_j ”;
 - 2. 若项目 $A \rightarrow \alpha.$ 属于 I_k , 那么, 对任何终结符 a 和 “ $\#$ ”, 置 ACTION[k, a] 为 “用产生式 $A \rightarrow \alpha$ 进行归约”, 简记为 “ r_j ”; 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“接受”, 简记为“acc”;

4.若 $GO(I_k, A) = I_j$, A 为非终结符, 则置 $GOTO[k, A] = j$;

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LR(0)表。

具有LR(0)表的文法G称为一个LR(0)文法。

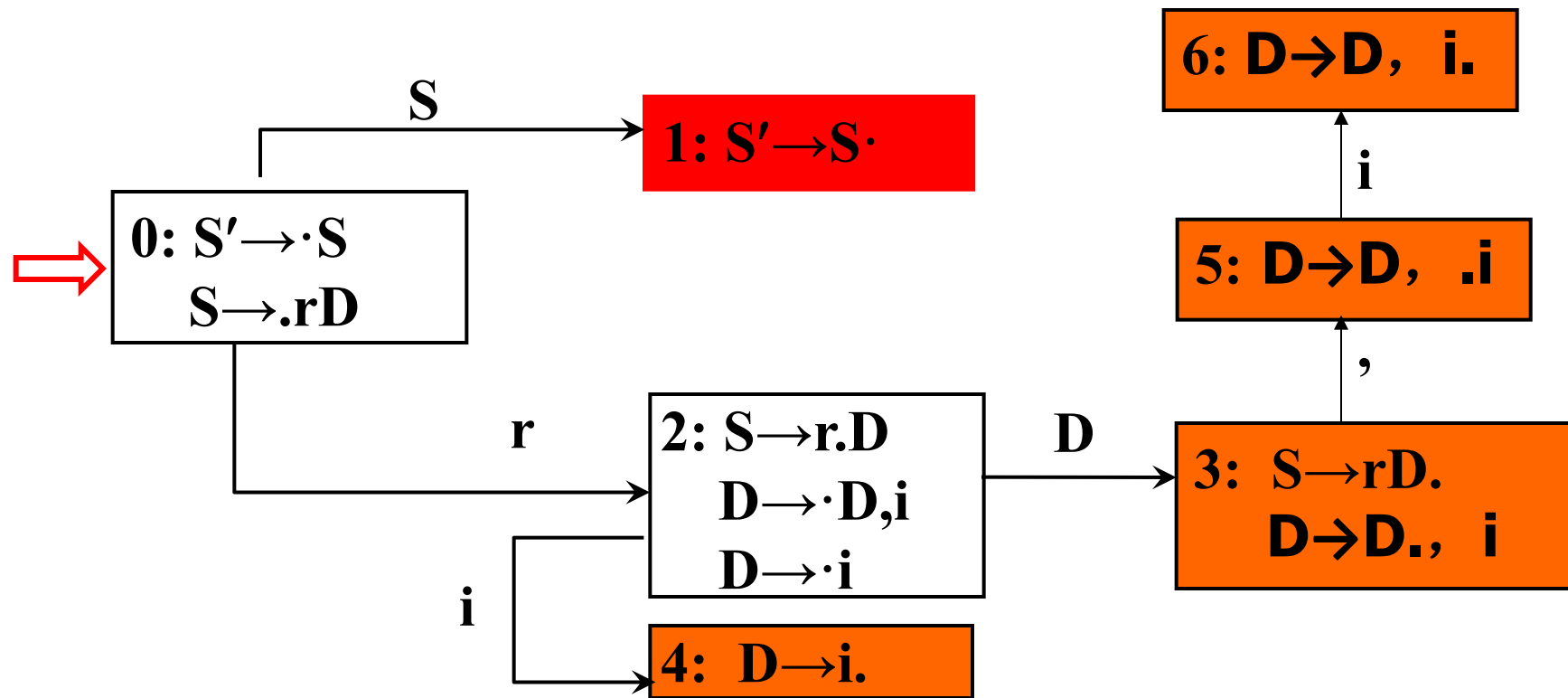


LR(0)文法定义

- 对于一个文法的**LR(0)**项目集规范族不存在移进-归约，或归约-归约冲突的文法称为**LR(0)**文法。
 - 移进-归约冲突：
 - 一个项目集中同时存在形如：
 - **$A \rightarrow \alpha.a\beta$** 和 **$B \rightarrow \gamma.$**
 - 归约-归约冲突：
 - 一个项目集中同时存在形如：
 - **$A \rightarrow \alpha.$** 和 **$B \rightarrow \gamma.$**

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

构造识别此文法活前缀的**DFA**





该文法的项目集规范族如下：

0: $S' \rightarrow \cdot S$
 $S \rightarrow \cdot rD$

1: $S' \rightarrow S \cdot$

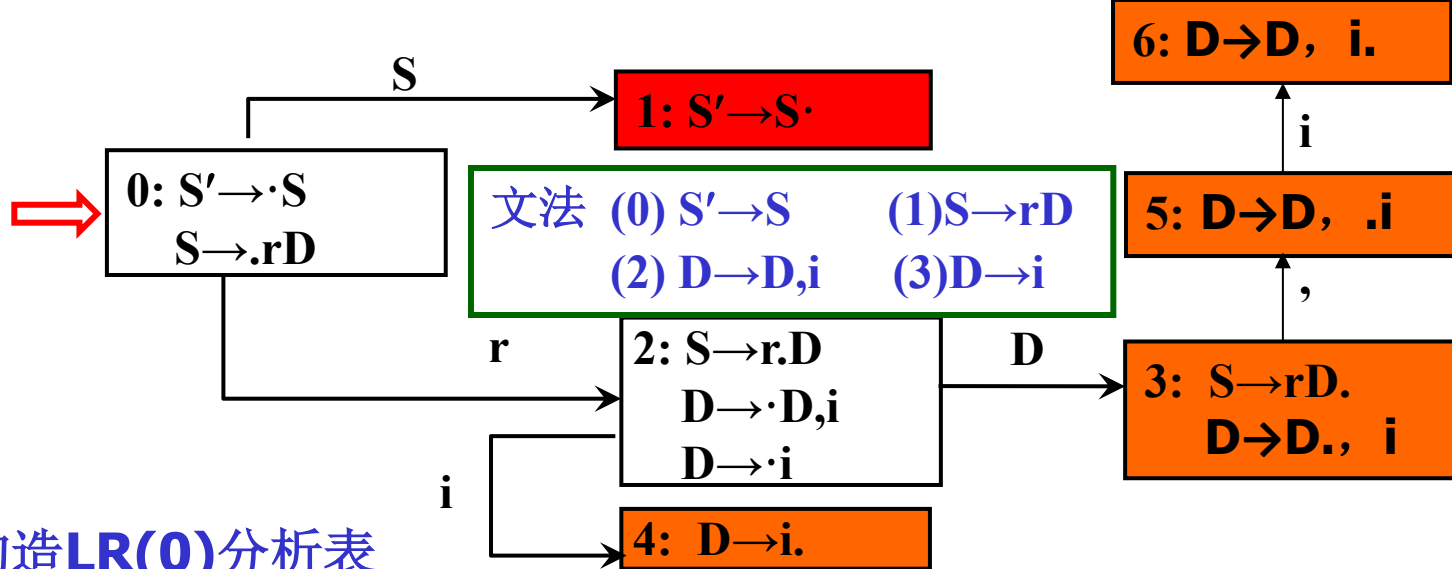
2: $S \rightarrow r \cdot D$
 $D \rightarrow \cdot D, i$
 $D \rightarrow \cdot i$

4: $D \rightarrow i \cdot$

6: $D \rightarrow D, i \cdot$

5: $D \rightarrow D, \cdot i$

3: $S \rightarrow rD \cdot$
 $D \rightarrow D \cdot, i$



构造LR(0)分析表

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S2				1	
1				acc		
2			S4			3
3	r1	<u>S5 r1</u>	r1	r1		
4	r3	r3	r3	r3		
5			S6			
6	r2	r2	r2	r2		

6.3 SLR(1)分析

3: $S \rightarrow rD.$
 $D \rightarrow D., i$

- **SLR(1)分析**: 对**LR(0)**规范族中有冲突的项目集状态用向前看一个符号的办法进行处理, 以解决冲突
- 只需要考查当用**rD**进行归约成**S**时, **S**的后跟符号集合中不包含当前所有移进项目的移进符号的集合, 则移进-规约冲突便可解决。

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

6.3 SLR(1)分析

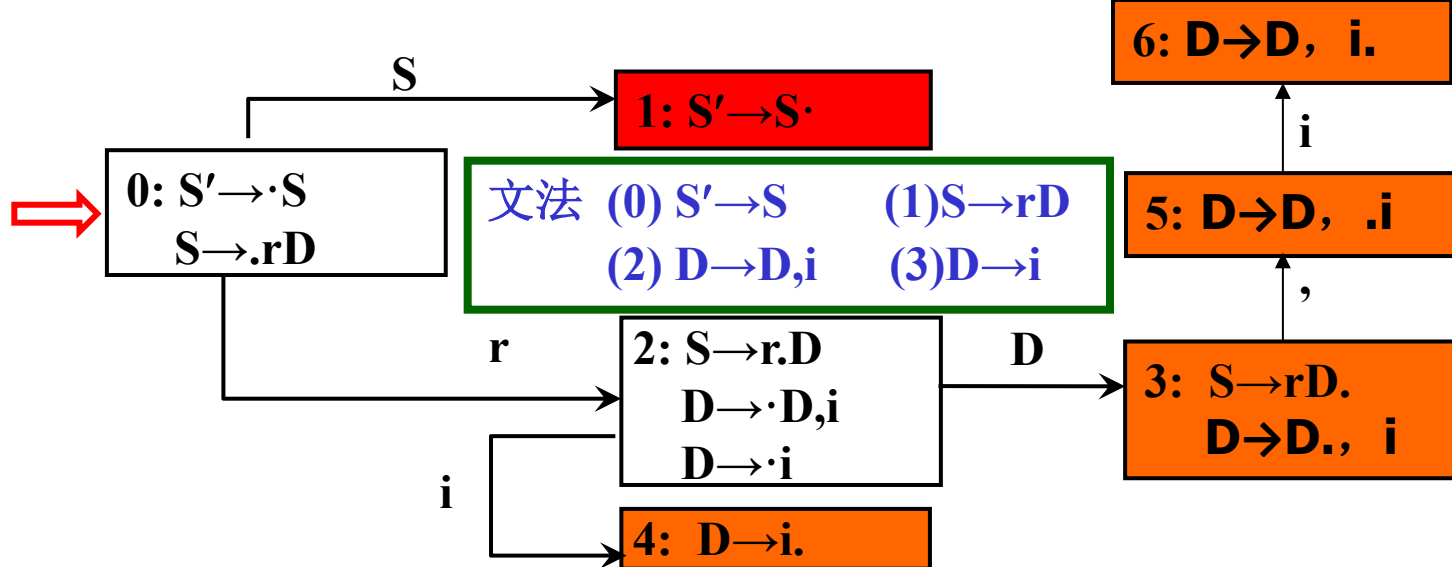
3: $S \rightarrow rD.$
 $D \rightarrow \cdot D, i$

SLR (1) 分析表

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S2				1	
1				acc		
2			S4			3
3		S5		r1		
4	r3	r3	r3	r3		
5			S6			
6	r2	r2	r2	r2		

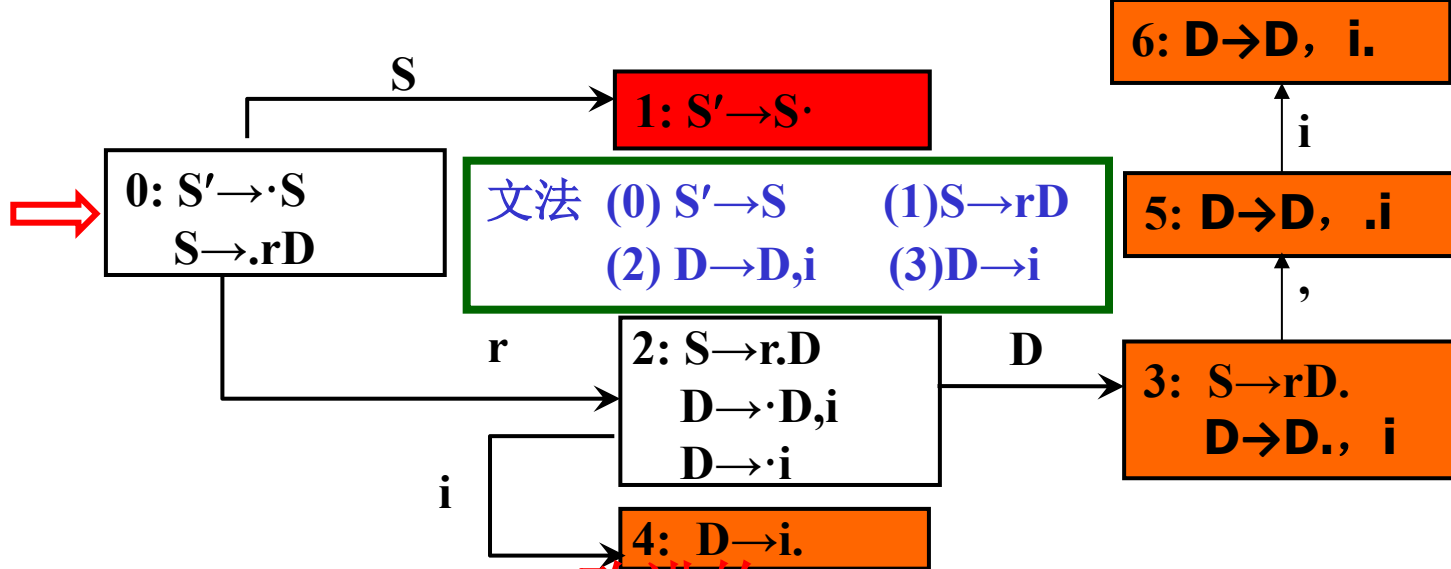
- 如果 **LR(0)** 项目集规范族中某个项目集 I_k 含 **移进/归约和归约/归约冲突**:
- $I_k: \{ \dots A \rightarrow \alpha . b \beta, P \rightarrow \omega ., Q \rightarrow \gamma ., \dots \}$
- 若 $\text{FOLLOW}(Q) \cap \text{FOLLOW}(P) = \phi$
- $\text{FOLLOW}(P) \cap \{ b \} = \phi$
- $\text{FOLLOW}(Q) \cap \{ b \} = \phi$
- 则解决冲突的 **SLR(1)** 技术:
- $\text{action} [k, b] = \text{移进}$
- 对 $a \in \text{FOLLOW}(P)$ 则 $\text{action} [k, a] = \text{用 } P \rightarrow \omega \text{ 归约}$
- 对 $a \in \text{FOLLOW}(Q)$ 则 $\text{action} [k, a] = \text{用 } Q \rightarrow \gamma \text{ 归约}$

- 通常对于**LR(0)**项目集规范族的一个项目**I**中可能含有多个移进项目和多个归约项目。
- 假定**LR(0)**规范族的一个项目集**I**= $\{A_1 \rightarrow \alpha \cdot a_1 \beta_1, A_2 \rightarrow \alpha \cdot a_2 \beta_2, \dots, A_m \rightarrow \alpha \cdot a_m \beta_m, B_1 \rightarrow \alpha_1 \cdot, B_2 \rightarrow \alpha_2 \cdot, \dots, B_n \rightarrow \alpha_n \cdot\}$
- 如果集合 $\{a_1, \dots, a_m\}, FOLLOW(B_1), \dots, FOLLOW(B_n)$ 两两不相交, 则:
 - 若 $a \in \{a_1, \dots, a_m\}$, 则移进
 - 若 $a \in FOLLOW(B_i), i=1,2,\dots,n$, 则用产生式 $B_i \rightarrow \alpha_i$ 进行归约;
 - 此外, 报错。
- 如果一个文法的**LR(0)**分析表中所含有的动作冲突都能用上述方法解决, 则这个文法是**SLR (1)** 文法



SLR (1) 分析表

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S2				1	
1				acc		
2			S4			3
3		S5		r1		
4	r3	r3	r3	r3		
5			S6			
6	r2	r2	r2	r2		



改进的 SLR (1) 分析表

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S2				1	
1				acc		
2			S4			3
3		S5		r1		
4		r3		r3		
5			S6			
6		r2		r2		

构造改进的**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**和识别活前缀的有限自动机.
- 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**I_k**的下标**k**为分析器的初态。

■ 分析表的**ACTION**和**GOTO**子表构造方法:

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 “ s_j ”;
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 “ r_j ”; 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置上“出错标志”。

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**。

- 
- 例： 考察下面的拓广文法：
-

(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) $F \rightarrow i$

■ 这个文法的**LR(0)**项目集规范族为：

I₀: $S' \rightarrow \cdot E$
 $E \rightarrow \cdot E + T$
 $E \rightarrow \cdot T$
 $T \rightarrow \cdot T * F$
 $T \rightarrow \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot i$

I₄: $F \rightarrow (\cdot E)$
 $E \rightarrow \cdot E + T$
 $E \rightarrow \cdot T$
 $T \rightarrow \cdot T * F$
 $T \rightarrow \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot i$

I₇: $T \rightarrow T * \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot i$

I₈: $F \rightarrow (E \cdot)$
 $E \rightarrow E \cdot + T$

I₁: $S' \rightarrow E \cdot$
 $E \rightarrow E \cdot + T$

I₅: $F \rightarrow i \cdot$

I₉: $E \rightarrow E + T \cdot$
 $T \rightarrow T \cdot * F$

I₂: $E \rightarrow T \cdot$
 $T \rightarrow T \cdot * F$

I₆: $E \rightarrow E + \cdot T$
 $T \rightarrow \cdot T * F$
 $T \rightarrow \cdot F$
 $F \rightarrow \cdot (E)$
 $F \rightarrow \cdot i$

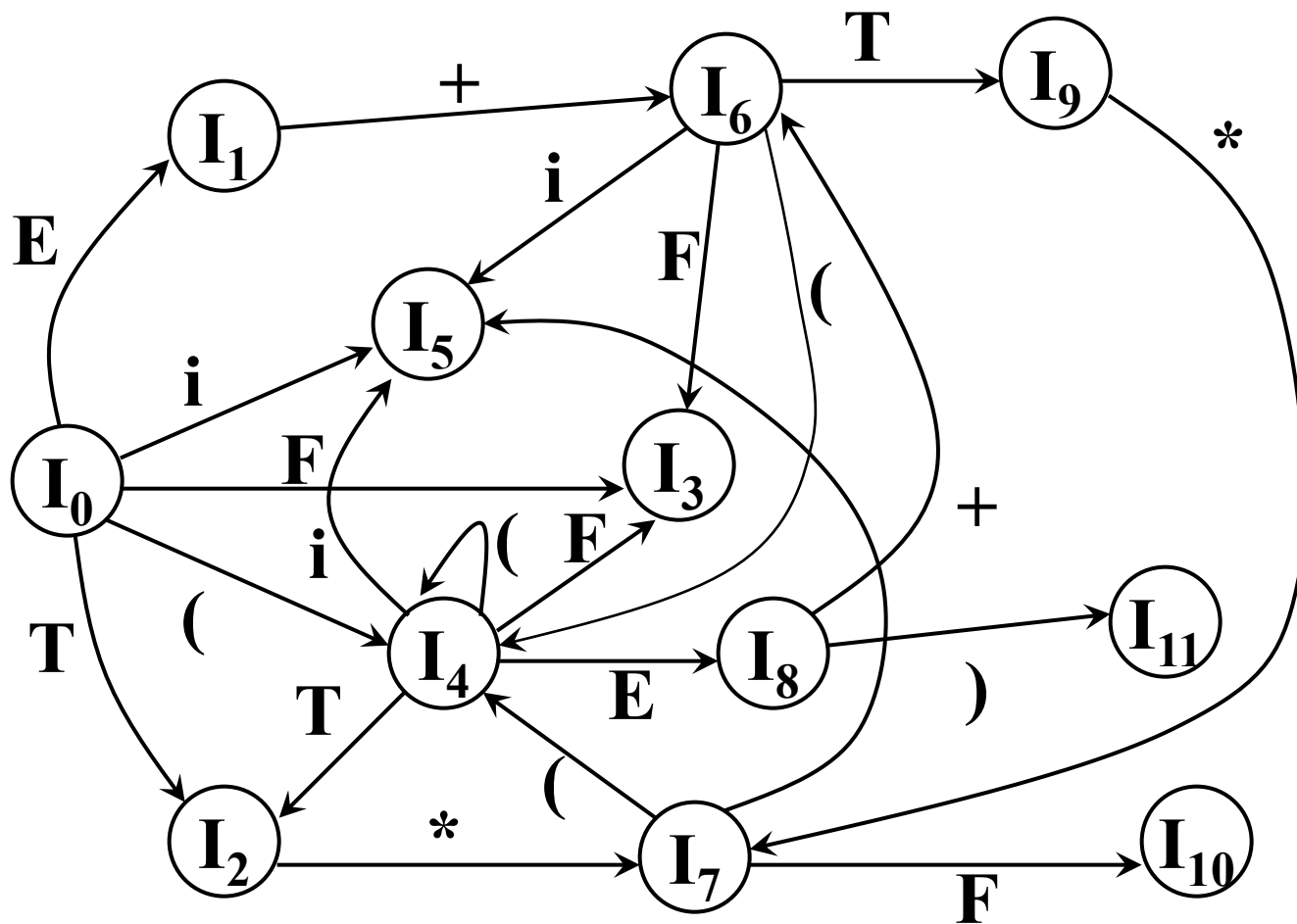
I₁₀: $T \rightarrow T * F \cdot$

I₃: $T \rightarrow F \cdot$

I₁₁: $F \rightarrow (E) \cdot$

(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) $F \rightarrow i$

■ 识别该文法活前缀的**DFA**



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

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

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

I_1 、 I_2 、 I_9 存在移进、归约冲突

解决?

$I_1: S' \rightarrow E \cdot$ FOLLOW(S')={#} 遇#接受
 $E \rightarrow E \cdot + T$ 遇+移进

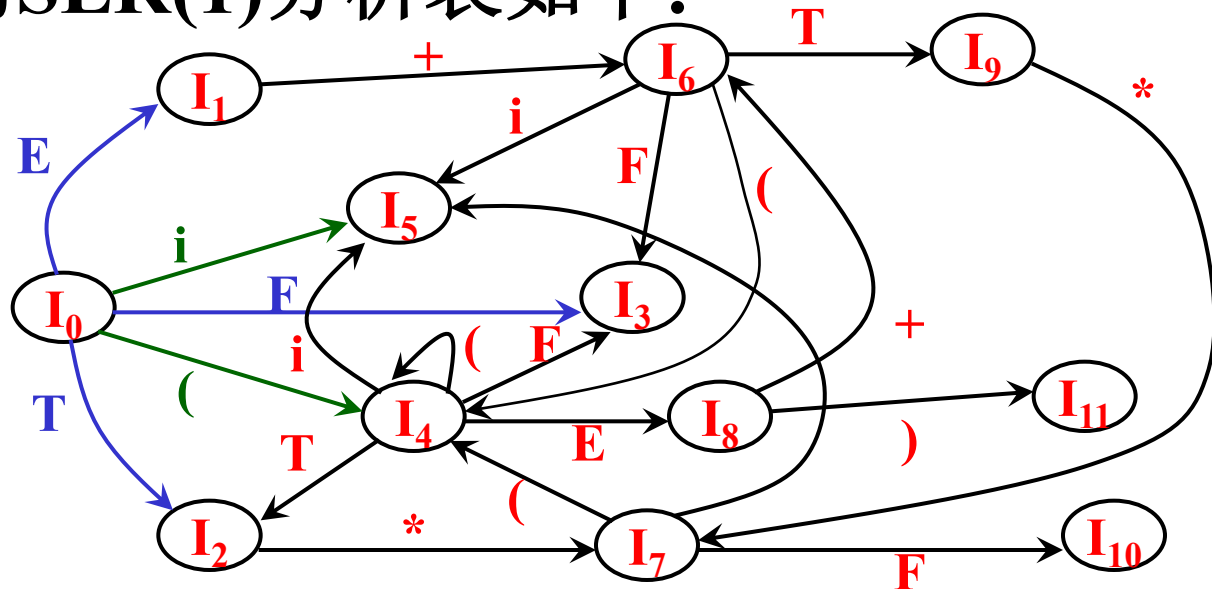
$I_2: E \rightarrow T \cdot$ 遇FOLLOW(E)={+,),#}归约
 $T \rightarrow T \cdot * F$ 遇* 移进

$I_9: E \rightarrow E + T \cdot$ 遇FOLLOW(E)={+,),#}归约
 $T \rightarrow T \cdot * F$ 遇* 移进

- (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) $F \rightarrow i$

改进的SLR(1)分析表如下:

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

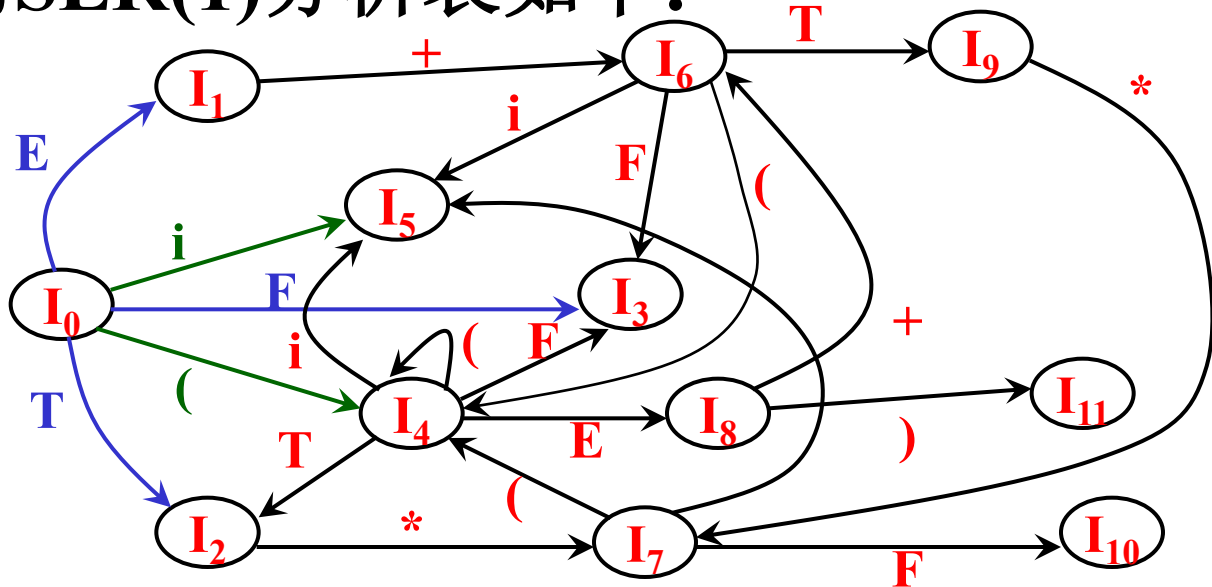


	ACTION						GOTO		
	i	+	*	()	#	E	T	F
0	S5			S4			1	2	3
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

改进的SLR(1)分析表如下:

$$\begin{array}{l} \mathbf{I}_1: \quad \mathbf{S}' \rightarrow \mathbf{E} \cdot \\ \quad \quad \mathbf{E} \rightarrow \mathbf{E} \cdot + \mathbf{T} \end{array}$$
$$\begin{array}{l} \mathbf{I_2: \quad E \rightarrow T \cdot} \\ \quad \mathbf{T \rightarrow T \cdot * F} \end{array}$$

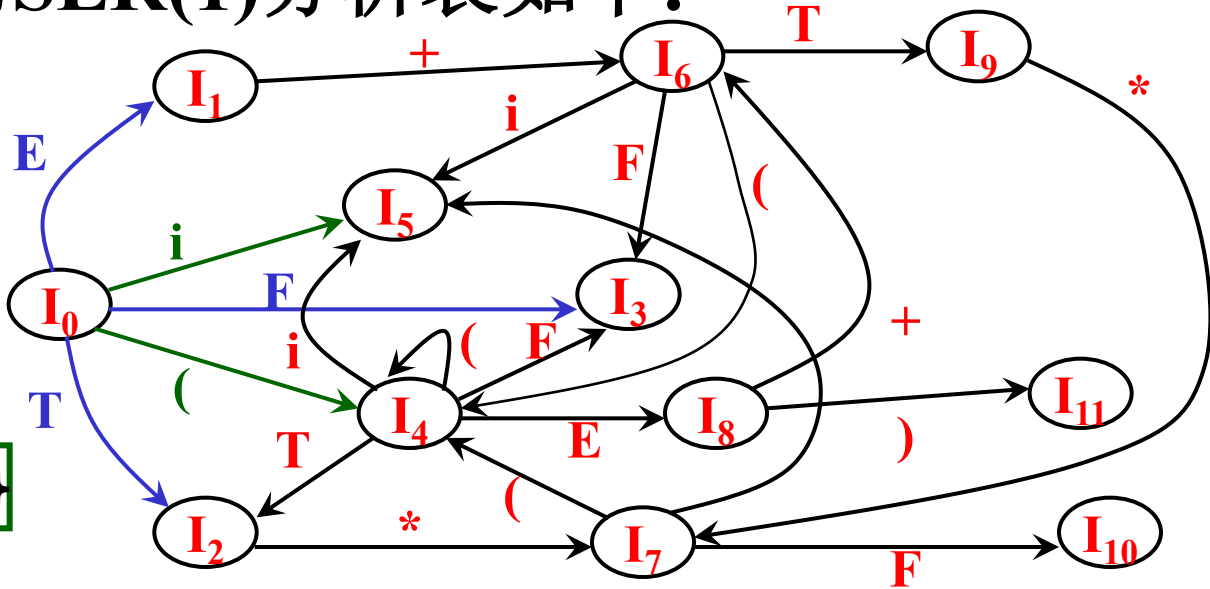
FOLLOW(E)={+,),#}

[illegible]

改进的SLR(1)分析表如下:

$$\mathbf{I}_3: \quad \mathbf{T} \rightarrow \mathbf{F}.$$

FOLLOW(T)={+,),#, *}

[illegible]

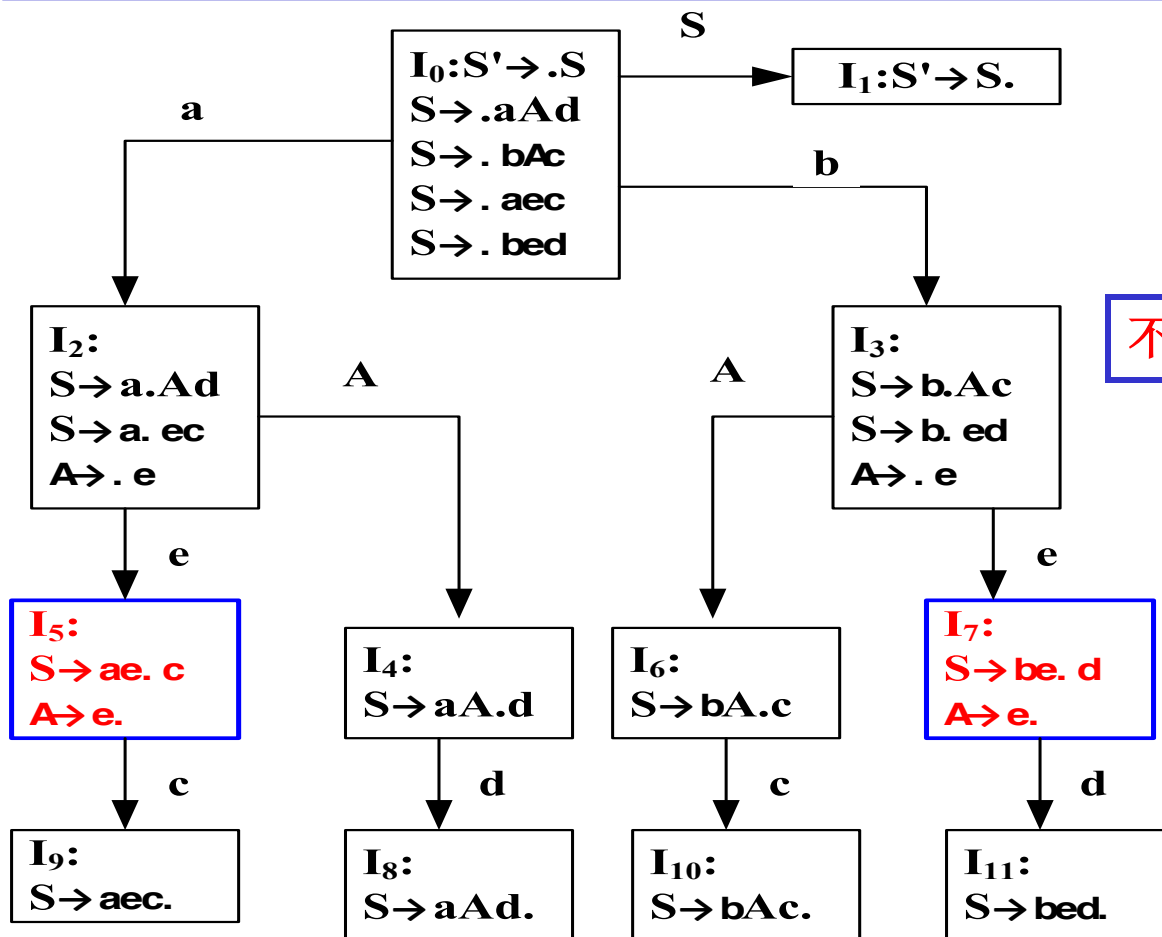
利用**SLR(1)**分析表 对i) #进行分析

	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			

- (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) $F \rightarrow i$

文法G:

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



$FOLLOW(A) = \{c, d\}$

不能用SLR (1) 方法解决

LR(0)识别G的活前缀的DFA

7.4 LR(1)分析

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

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

- 文法**G**:
 - (0) $S' \rightarrow S$ (1) $S \rightarrow aAd$ (2) $S \rightarrow bAc$
 - (3) $S \rightarrow aec$ (4) $S \rightarrow bed$ (5) $A \rightarrow e$
- I_5 、 I_7 存在移进-归约冲突，不能用**SLR (1)**方法解决
- **FOLLOW(A)={c,d}**
 - 在**5**状态，遇见**c**，移进(**S9**)、归约(**r5**)
 - 在**7**状态，遇见**d**，移进(**S11**)、归约(**r5**)

分析:

- $S' \Rightarrow S \Rightarrow aAd \Rightarrow aed$ $S' \Rightarrow S \Rightarrow bAc \Rightarrow bec$
- $S' \Rightarrow S \Rightarrow aec$ $S' \Rightarrow S \Rightarrow bed$
- 状态**5**，遇见**d**归约，遇见**c**移进；状态**7**，遇见**c**归约，遇见**d**移进；



7.4 LR(1)分析

- **FOLLOW(A)** 包含了在**任何句型**中跟在 **A** 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**A**后。
- **FOLLOW**集合提供的信息太泛！
- 根据项目集的构造原则有：
- 若 $A \rightarrow \alpha . B \beta \in I$
则 $B \rightarrow . \gamma$ ($B \rightarrow \gamma$ 是一产生式) $\in I$
不防考虑，把**FIRST** (β) 中的符号作为用 $B \rightarrow \gamma$ 归约的搜索符，**向前搜索符**

7.4.1 LR(1)项目集族的构造

- **LR (1) 项目的一般形式:** $[A \rightarrow \alpha \cdot \beta, a]$
 - 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 后的符号是终结符 a 时进行归约
- **a 称作该项目的向前搜索符**
- 向前搜索符只对**归约项目**起作用, 对于任何**移进或待约项目**不起作用
- **$[A \rightarrow \alpha \beta \cdot, a]$:**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 $A \rightarrow u \bullet, a/b/c$



7.4.1 LR(1)项目集族的构造

- 为构造有效的**LR(1)**项目集族我们需要两个函数 **CLOSURE**（闭包）和**GO**（转换）。

- 初始时：

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

LR (1) 项目集的构造：

- 对初始项目 $[S' \rightarrow \cdot S, \#]$ 求闭包后再用转换函数逐步求出整个文法的**LR(1)**项目集



(1) 构造LR(1)项目集的闭包函数

- 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**FIRST**($\beta_2 a$)中的每个终结符**b**, 如果 $[B \rightarrow \cdot \delta, b]$ 不在**closure(I)**中, 则把它加进去;
- ③ 重复① ②, 直至**closure(I)**不再增大。

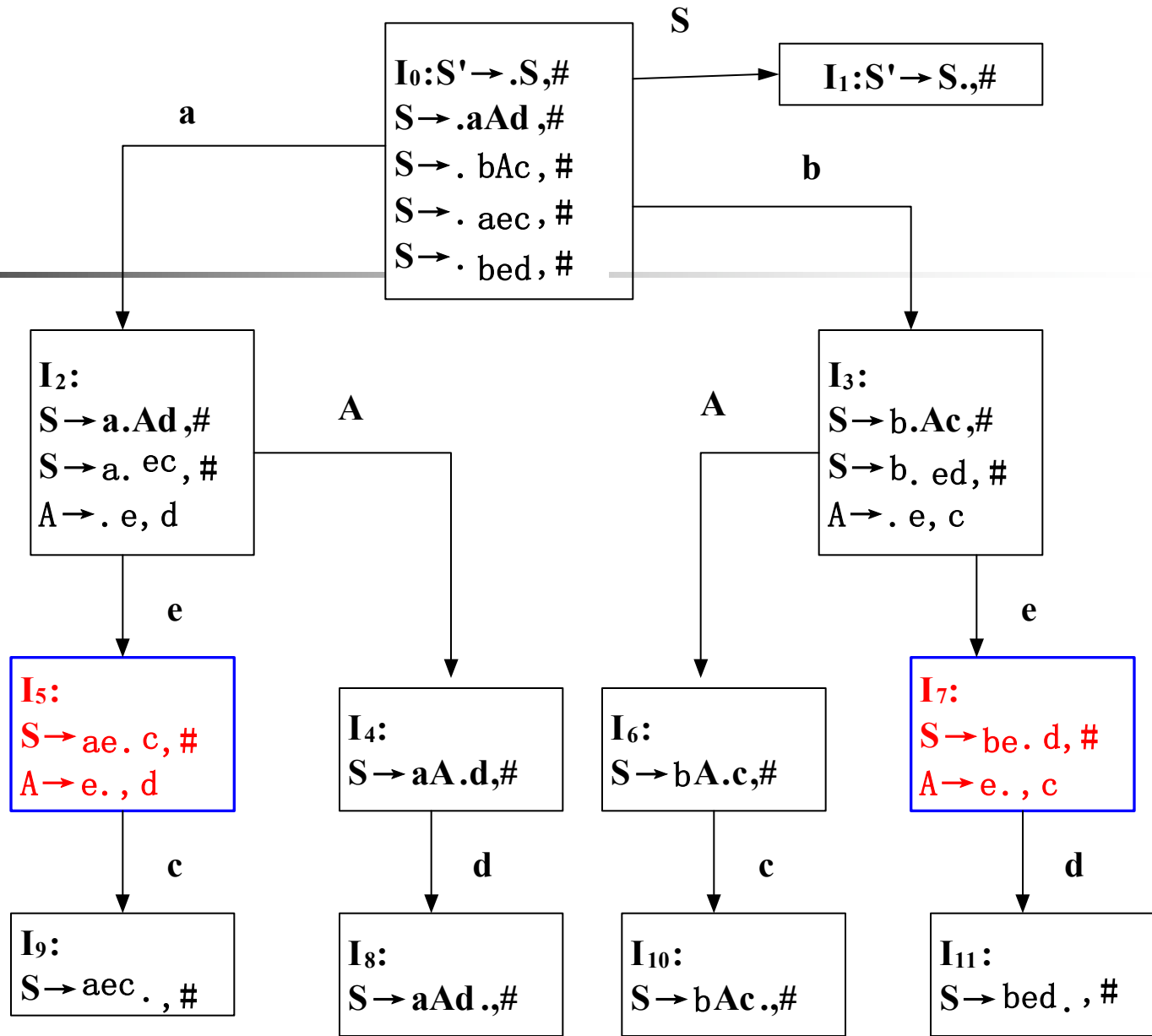


(2) 构造转换函数

- 若 I 是一个项目集， X 是一个文法符号
 $GO(I, X) = \text{closure}(J)$
- 其中
 $J = \{ \text{任何形如 } [A \rightarrow \alpha X. \beta, a] \text{ 的项目} \mid [A \rightarrow \alpha . X \beta, a] \in I \}$
- 文法的LR(1)项目集: 反复利用 (1) (2), 直到项目集不在扩大

文法G:

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

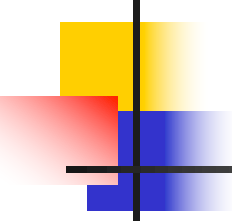




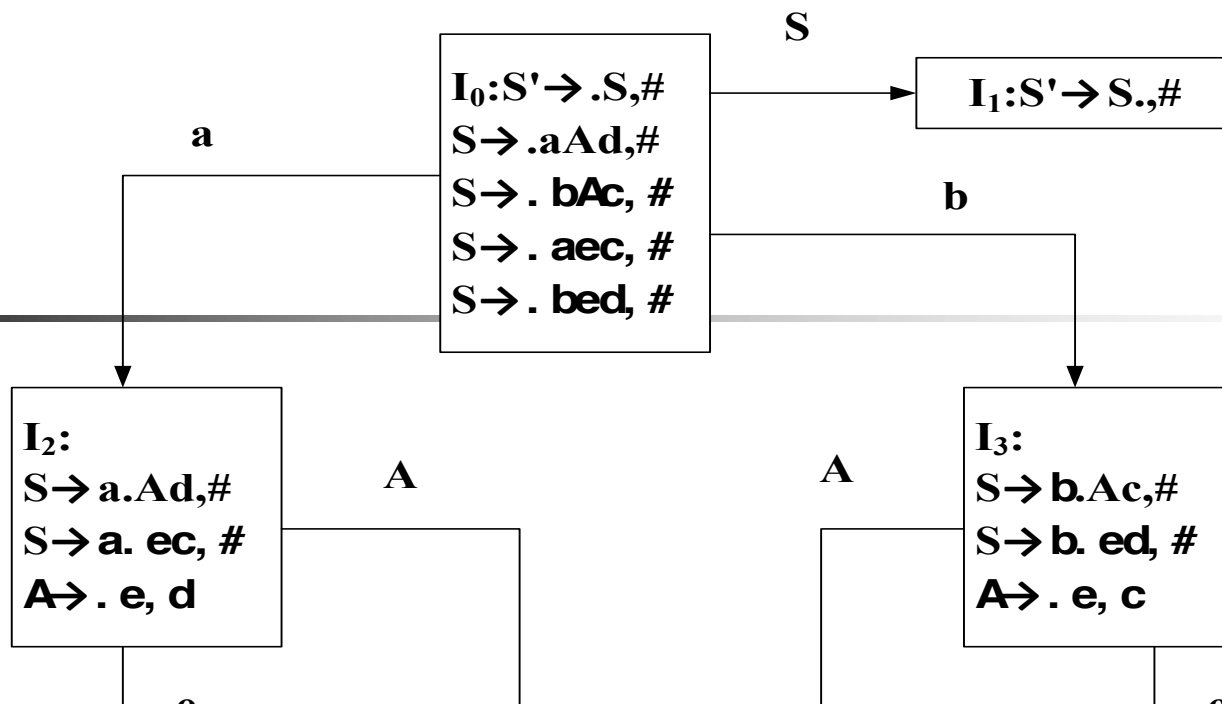
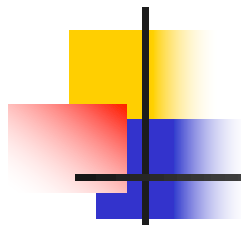
7.4.2 LR(1)分析表的构造

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. 令含有项目 $[S' \rightarrow \cdot S, \#]$ 的 I_k 的下标 k 为状态 0 (初态)。
ACTION表和GOTO表可按如下方法构造:
2. 若项目 $[A \rightarrow \alpha \cdot, b]$ 属于 I_k , 那么置ACTION[k, b]为“ r_j ”;其中, 假定 $A \rightarrow \alpha$ 为文法 G' 的第 j 个产生式

- 
3. 若项目 $[A \rightarrow \alpha . a \beta, b]$ 属于 I_k 且 $GO(I_k, a) = I_j$, 则置 $ACTION[k, a]$ 为 “ s_j ”;
 - ~~4. 若项目 $[S' \rightarrow S . , \#]$ 属于 I_k , 则置 $ACTION[k, \#]$ 为 “acc”;~~
 5. 若 $GO(I_k, A) = I_j$, A 为非终结符, 则置 $GOTO(k, A) = j$
 6. 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，则称该文法**G** 为一个**LR (1)** 文法。



LR(1)分析表

	ACTION						GOTO	
	a	b	c	d	e	#	S	A
0	S2	S3					1	
1						acc		

I₂:
 $S \rightarrow a.Ad, \#$
 $S \rightarrow a. ec, \#$
 $A \rightarrow . e, d$

I₃:
 $S \rightarrow b.Ac, \#$
 $S \rightarrow b. ed, \#$
 $A \rightarrow . e, c$

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

I₅:
 $S \rightarrow ae. c, \#$
 $A \rightarrow e. , d$

I₄:
 $S \rightarrow aA.d, \#$

I₆:
 $S \rightarrow bA.c, \#$

I₇:
 $S \rightarrow be. d, \#$
 $A \rightarrow e. , c$

I₉:

I₈:

I₁₀:

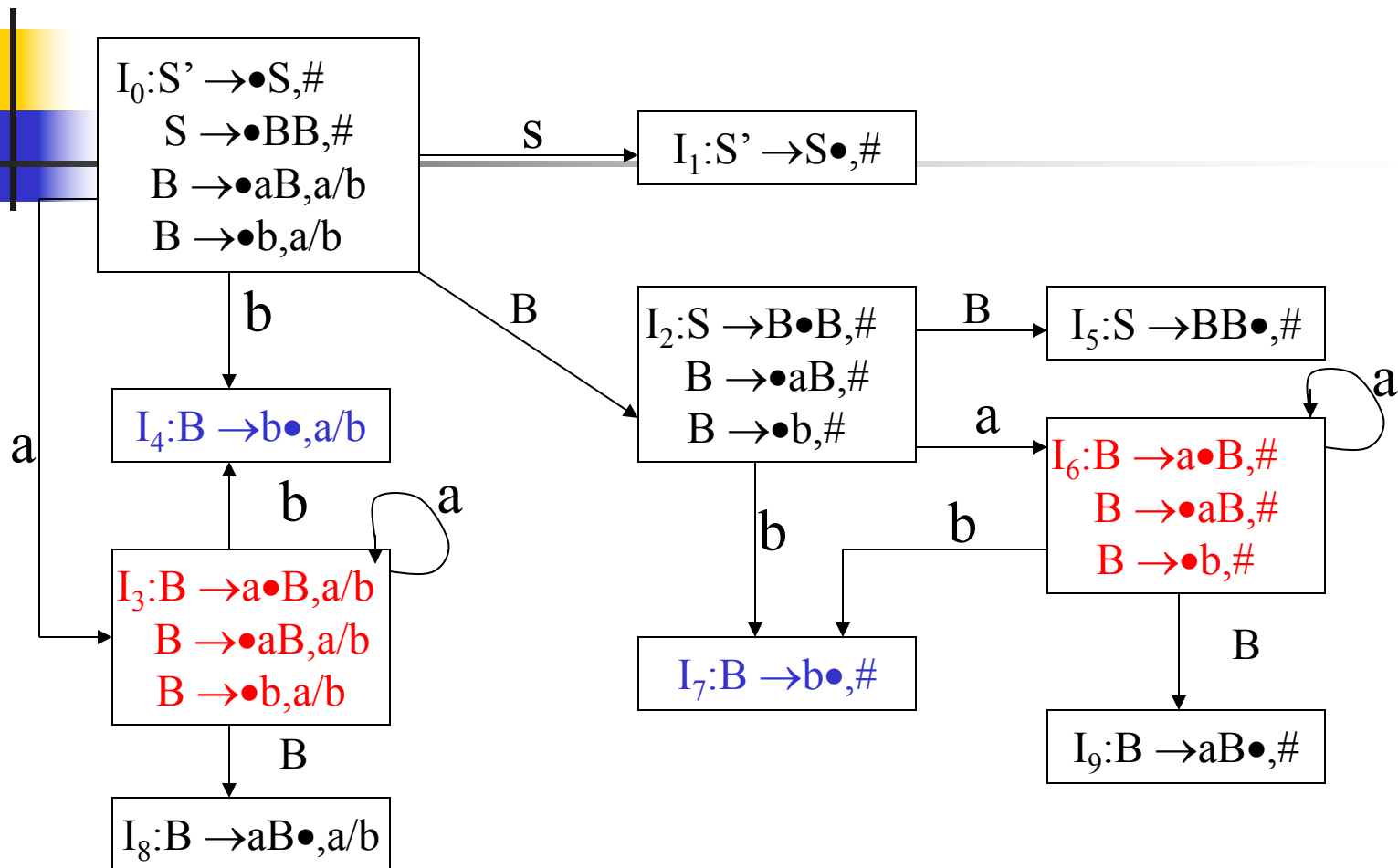
I₁₁:

LR(1)分析表

	ACTION						GOTO	
	a	b	c	d	e	#	S	A
0	S2	S3					1	
1						acc		
2					S5			4
3					S7			6
4				S8				
5			S9	r5				
6								
7								

- 
-
- 每个**SLR (1)** 文法都是**LR (1)** 文法。
 - **LR(1)**比**SLR (1)** 能力强
 - 一般情况下，一个**LR (1)** 文法的项目集的个数比其**SLR (1)** 的状态要多。

G(S): (0) $S' \rightarrow S$ (1) $S \rightarrow BB$ (2) $B \rightarrow aB$ (3) $B \rightarrow b$



LR(1)项目集和转换函数



本章小结

- **6.1 LR分析概述**
- **6.2 LR (0) 分析**
- **6.3 SLR(1) 分析**
- **6.4 LR(1) 分析**