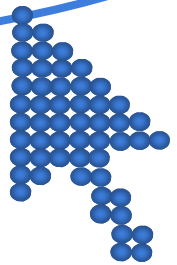
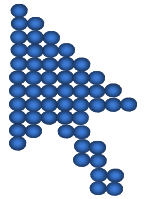
编 译 原 理

第五、六章　自底向上分析方法之LR分析



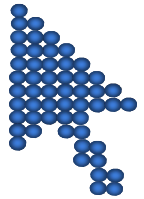
广东工业大学计算机学院

第一部分 预备知识（复习）



- 推导
  - 直接推导（应用文法的某一个规则）
  - 最左推导（每次替换句型中的最左非终结符）
  - 最右推导（规范推导，规范句型）
- 规约
- 短语
  - 对于文法**G[S]**，若有**S** $\Rightarrow$ α**A**δ且**A** $\Rightarrow$ β，则称β是句型αβδ相对于非终结符**A**的短语。
- 直接短语
  - 若有**A** $\Rightarrow$**β**则称**β**是句型αβδ相对于**A**或规则**A** → **β**的直接短语。
- 句柄
  - 一个句型的最左直接短语

第一部分 预备知识（复习）



例：给定文法G:

E→T | E+T | E－T

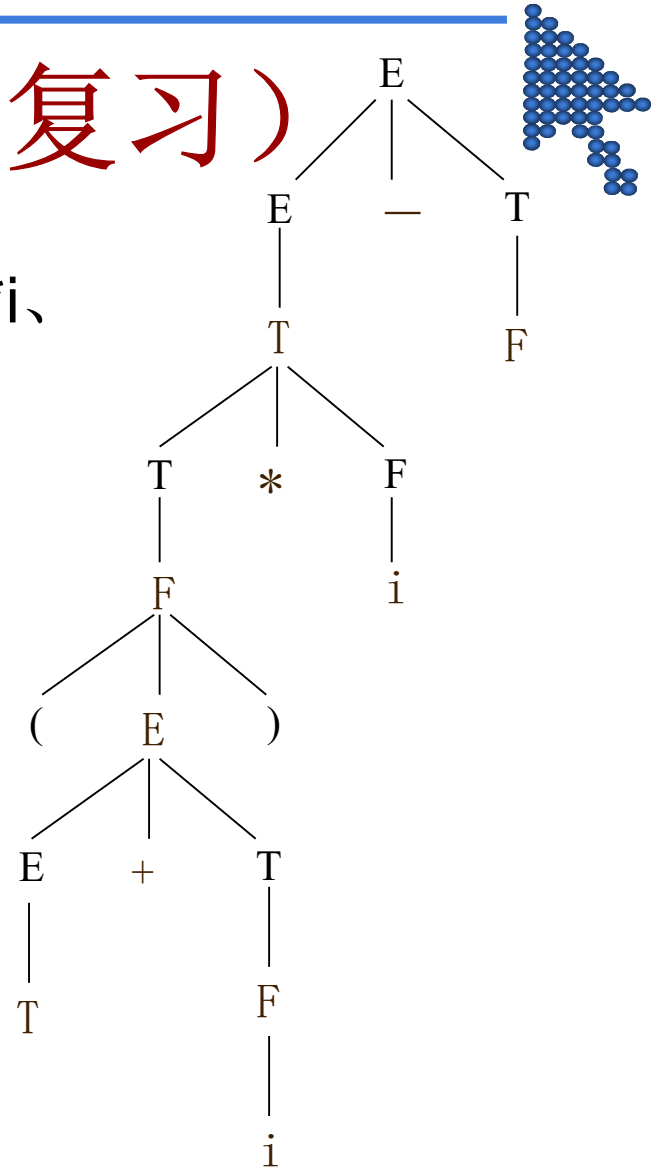
T→F | T*F | T / F

F→i | (E)

符号串（T+i）*i－F是文法G的一个句型，求其短语、直接短语和句柄。

第一部分 预备知识（复习）

- 短语有8个：（T+i）*i—F、（T+i）*i、（T+i）、……
- 直接短语：T、…..
- 句柄：…



用语法树求短语、直接短语和句柄的方法：

1）每个句型都有一棵语法树；

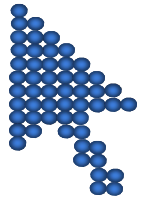
2）每棵语法树的叶（从左到右）组成一句型；

3）每个子树 的叶（从左到右）组成一短语；

4）每个直接子树 的叶（从左到右）组成一直接短语；

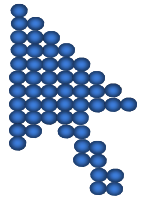
5）最左直接子树 的叶（从左到右）组成一句柄。

第二部分：自底向上分析方法



- 自底向上分析方法也称移进-归约分析法，它的实现思想是：
- **(1)** 对输入符号串自左向右进行扫描，将输入符逐个移入一个后进先出栈中，边移入边分析。
- **(2)** 一旦栈顶符号串形成某个句型的句柄时(该句柄对应某产生式的右部)，就用该产生式的左部非终结符代替此句柄，这称为一步归约。
- **(3)** 重复这一过程，直到归约到栈中只剩下文法的开始符号时，则认为分析成功，即确认输入符号串是文法的句子。

第二部分：自底向上分析方法



例6.1：设文法G[S]为：

(1) S→aAcBe　　(2) A→b

(3) A→Ab　　(4) B→d

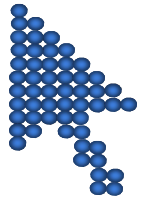
- 对输入串abbcde#进行分析，检查该符号串是否是G[S]的句子。
- 自底向上分析的移进-归约过程是自顶向下最右推导的逆过程，而最右推导为规范推导，自左向右的归约过程也称规范归约。
- 输入串abbcde的最右推导是：

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

- 规范规约

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

例子的移进-归约过程



- **(1) S→aAcBe　(2) A→b**
  (3) A→Ab　　(4) B→d
- 先设一个先进后出的符号栈，并把句子左括号"#"号放入栈底，其分析过程如表：

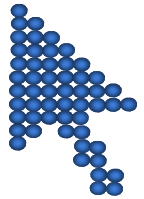
步　骤	符号栈	输入符号串	动　作
1)	#	abbcde#	移进
2)	#a	bbcde#	移进
3)	#ab	bcde#	归约(A→b)
4)	#aA	bcde#	移进
5)	#aAb	cde#	归约(A→Ab)
6)	#aA	cde#	移进
7)	#aAc	de#	移进

例子的移进-归约过程

- **(1) S→aAcBe　(2) A→b**
  (3) A→Ab　　(4) B→d
- 先设一个先进后出的符号栈，并把句子左括号"#"号放入栈底，其分析过程如表：

步　骤	符号栈	输入符号串	动　作
7)	#aAc	de#	移进
8)	#aAcd	e#	归约(B→b)
9)	#aAcB	e#	移进
10)	#aAcBe	#	归约(S→aAcBe
11)	#S	#	接受

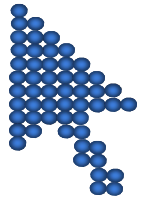
移进-归约的关键！



- 当一个文法无二义性时，那么它对一个句子的规范推导是唯一的，规范归约也必然是唯一的。
- *实际上，自底向上分析的关键问题是：在分析过程中如何确定句柄。*
- 也就是说，如何知道何时在栈顶符号串中已形成某句型的句柄，那么就可以确定何时可以进行归约。
- 常用的方法:

  优先分析（第**5**章）和**LR**分析（第**6**章）

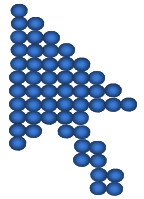
第三部分 LR分析法之概述



由上面的分析可见，自底向上分析法的关键问题是在分析过程中如何确定句柄。在移进－归约过程中，我们如何确定栈顶符号串是否已经形成相对于某个规则的句柄呢?

LR分析方法根据当前分析栈中的符号串（通常以状态表示）和向右顺序查看输入串的**K**（**K**大于等于**0**）个符号就可唯一地确定分析器的动作是移进还是归约以及用哪个产生式归约，因而也就能唯一地确定句柄.

第三部分 LR分析法之概述

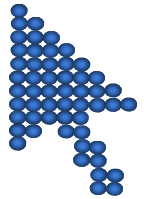


- **LR(K)的含义：**
  - **L表示从左到右扫描输入串**
  - **R表示最左规约（即最右推导的逆过程）**
  - **K表示向右查看输入串符号的个数**

当K=1时，能满足当前绝大多数高级语言编译程序的需要，所以着重介绍

LR(0), SLR(1), LR(1)方法

第三部分 LR分析法之概述



- **LR分析器的组成**

1) 总控程序(驱动程序)：所有LR分析器总控程序相同。
2) 分析表：
   - 不同文法有不同的分析表
   - 同一文法采用的LR分析器不同，分析表也不同
   - 包括ACTION表（动作表）和GOTO表（状态转换表）
   - 分析表是**LR**分析器的核心
3) 分析栈：

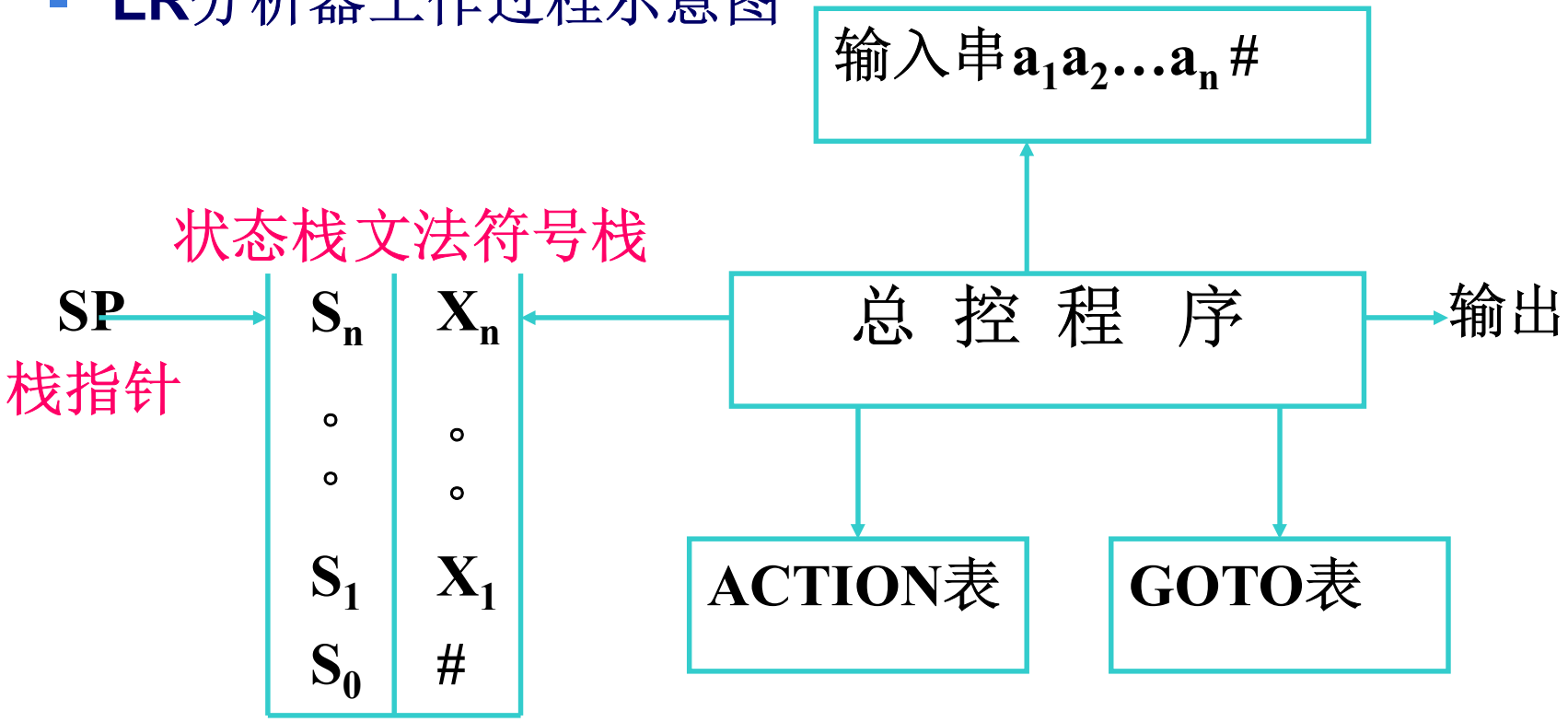
   包括状态栈S和文法符号栈X。

   分析器的动作由栈顶状态和当前输入符号所决定。

（若是**LR(0)**则不需要向前查看输入符号。）

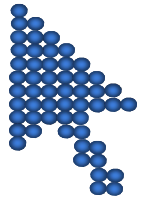
第三部分 LR分析法之概述

- LR分析器工作过程示意图



在分析的每一步，通用的总控程序按照状态栈栈顶状态Si和当前输入符号a查LR分析表，并执行其中ACTION和GOTO规定的操作。

第三部分 LR分析法之概述

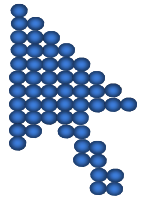


- LR分析表
- 行标题为状态，列标题为文法符号

状态	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S2						1		
1						acc			
2		S1		S3					
3	r2	r2	r2	r2	r2	r2			

- ACTION表示当前状态面临输入符号时应采取的动作。
- GOTO表示当前状态面临文法符号时应转向的下一个状态。

第三部分 LR分析法之概述



- ACTION[S_i,a]规定了栈顶状态为S_i时遇到输入符号a应执行的动作：（4个动作）

1. 移进(Sj)：

  当S_j=GOTO[S_i,a]成立，则把S_j移进状态栈，把a移进符号栈

⑩ 归约(rk)：

  用第k条产生式进行归约，此时栈顶形成了句柄β，文法中第k条产生式为A→β，且|β|=*l*，归约时从状态栈和符号栈中弹出*l*个符号，把A移入符号栈， 再把S_j=GOTO[S_i,A] 移入状态栈，其中状态i为修改指针后的栈顶状态.（特例，当*l*=*0*时）

⑩ 接受(acc)：

  当符号栈只剩文法开始符S，且当前输入符为'#'，则分析成功

⑩ 报错：

  当状态栈顶的状态遇到了不应该出现的文法符号，则报错，说明输入串不是该文法的句子

LR分析举例（以LR(0)为例）

- 下面以文法G[S]为例，说明LR(0)分析的一般过程：

  (1) S→aAcBe　(2) A→b　(3) A→Ab　(4) B→d

- 对输入串abbcde#用自底向上归约的方法进行分析。

- 已知该文法的LR(0)分析表：
- ACTION[S_0, a] = S_2
- GOTO[S_2, A] = S_3
- ACTION[S_4, #] = r_2

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)分析举例(续1)

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

S: shift，表示移进

r: reduce，表示规约

状态	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S2						1		
2				S4				3	
3		S5		S6					
4	r2	r2	r2	r2	r2	r2			
5					S8				7
6	r3	r3	r3	r3	r3	r3			

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	

LR(0)分析举例(续2)

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

状态	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S2						1		
2				S4				3	
3		S5		S6					
4	r2	r2	r2	r2	r2	r2			
5					S8				7
6	r3	r3	r3	r3	r3	r3			

步骤	符号栈	输入符号串	动 作	状态栈	ACTION	GOTO
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	

LR(0)分析举例(续3)

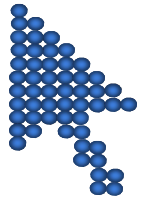
- G[S]：
- (1) S→aAcBe
- (2) A→b
- (3) A→Ab
- (4) B→d

在符号栈形成句柄时，LR(0)文法对于任何当前输入符，都会进行规约动作

状态	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S2						1		
1						acc			
5					S8				7
7			S9						
8	r4	r4	r4	r4	r4	r4			
9	r1	r1	r1	r1	r1	r1			

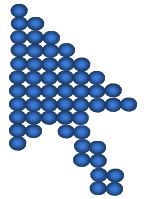
步骤	符号栈	输入符号串	动 作	状态栈	ACTION	GOTO
7)	#aAc	de#	移进	0235	S_8	
8)	#aAcd	e#	归约(B→d)	02358	r_4	7
9)	#aAcB	e#	移进	02357	S_9	
10)	#aAcBe	#	归约(S→aAcBe)	023579	r_1	1
11)	#S	#	接受	01	acc	

第四部分 LR分析法之LR(0)



- 从上分析，可见分析表的构造是**LR**分析的基础！
- 使用**LR(0)**分析表的**LR**分析器称为**LR(0)**分析器。
  - 构造**LR(0)**分析表的思想和方法是构造其他**LR**分析表的基础。
- **LR(0)**分析表的构造过程
  - 规范句型的可归前缀和活前缀**(6.2.1)**
  - 构造文法的识别活前缀及可归前缀的**DFA(6.2.2)**
  - 按**DFA**构造相应分析表——状态转换表和动作表**(6.2.4)**
  - 按分析表进行**LR(0)**分析**(6.2.4)**

第四部分 LR分析法之LR(0)



- 可归前缀和活前缀
- 对例6.1的文法G[S]的每条产生式编上序号（用[i]表示）并加在产生式的尾部，使产生式变为：

  S→aAcBe[1]

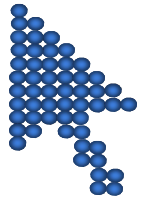
  A→b[2]

  A→Ab[3]

  B→d[4]

  但[i]不属产生式的文法符号

第四部分 LR分析法之LR(0)



- 对输入串 **abbcde** 进行推导时把序号也带入，则最右推导过程为：

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

- 它的逆过程,即最左归约(规范归约)则为：

  ab[2]b[3]cd[4]e[1] 用规则(2)归约
  ⇐ aAb[3]cd[4]e[1] 用规则(3)归约
  ⇐ aAcd[4]e[1] 用规则(4)归约
  ⇐ aAcBe[1] 用规则(1)归约
  ⇐ S

- 从这里可以看到每次归约时，归约前和归约后的已归约部分和剩余部分合起来构成文法的规范句型，而用哪个规则归约仅取决于当前句型的前部分内容。

例中每次归约前句型的前部分依次为：

ab[2]

aAb[3]

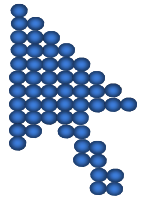
aAcd[4]

aAcBe[1]

这正是分析过程中每次采取归约动作前符号栈中的内容，即分别对应步骤 3、5、8、10 时符号栈中的符号串，我们把规范句型中句柄之前包括句柄在内的串称可归前缀。

步骤	状态	符号	输入串	ACTION	GOTO
(1)	0	#	abbcde#	S2	
(2)	02	#a	bbcde#	S4	
(3)	024	#ab	bcde#	r2	3
(4)	023	#aA	bcde#	S6	
(5)	0236	#aAb	cde#	r3	3
(6)	023	#aA	cde#	S5	
(7)	0235	#aAc	de#	S8	
(8)	02358	#aAcd	e#	r4	7
(9)	02357	#aAcB	e#	S9	
(10)	023579	#aAcBe	#	r1	1
(11)	01	#S	#	acc	

第四部分 LR分析法之LR(0)

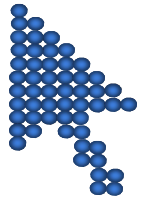


- 再来分析下列规范句型的前缀：

  ε,a,ab
  ε,a,aA,aAb
  ε,a,aA,aAc,aAcd
  ε,a,aA,aAc,aAcB,aAcBe

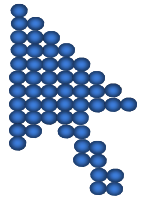
  不难发现前缀 **a,aA,aAc** 都不只是某一个规范句型的前缀，因此我们把形成可归前缀之前包括可归前缀在内所有规范句型的前缀都称为活前缀。
- 活前缀为一个或若干规范句型的前缀。在规范归约过程中的任何时刻只要已分析过的部分即在符号栈中的符号串均为规范句型的活前缀，则表明输入串已被分析过的部分是该文法某规范句型的一个正确部分。
- 对句柄的识别变成对规范句型活前缀的识别。

第四部分 LR分析法之LR(0)



- 活前缀的形式定义见书**P126**
  - *所谓活前缀是指规范句型的一个前缀，这种前缀不包含句柄之后的任何符号。（给个规范句型，可以找到句柄，那么句柄之前的（包括句柄）的前缀组成了活前缀）*

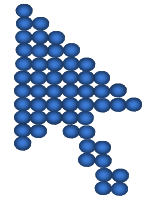
可归前缀和活前缀在LR分析中的作用



- 在LR分析过程中，实际上是把活前缀列出放在符号栈中；
- 一旦在栈中出现可归前缀，即句柄已经形成，就用相应的产生式进行归约；
- 在分析的过程中，只要符号栈中的符号串是一个活前缀，就可保证已被分析过的部分是该文法规范句型的正确部分。

因为可归前缀
包括句柄在内

第四部分 LR(0)--识别活前缀的有穷自动机

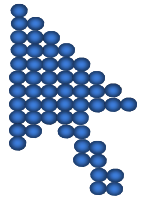


在 **LR** 方法实际分析过程中并不是去直接分析文法符号栈中的符号是否形成句柄，但它给我们一个启示：

- 把终结符和非终结符都看成一个有限自动机的输入符号；
- 每把一个符号进栈则看成已识别了该符号，而状态进行转换；
- 当识别到可归前缀时，表示栈中已经形成句柄，则认为到达了识别句柄的终态。

为了更好地构造活前缀的有穷自动机，我们首先应进行文法的拓广。

第四部分 LR(0)--识别活前缀的有穷自动机



- 拓广文法：若原文法G的开始符号为S，在G中加产生式S’→S后得新的文法G’，则称G’为G的拓广文法，S’为拓广后文法G’的开始符号。

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

- 用拓广文法则表示成：
  - (0) S′→S
  - (1) S→aAcBe
  - (2) A→b
  - (3) A→Ab
  - (4) B→d

- 对文法进行拓广的目的：对某些右部含有开始符号的文法，在归约过程中能分清是否已归约到文法的最初开始符，还是在文法右部出现的开始符号。

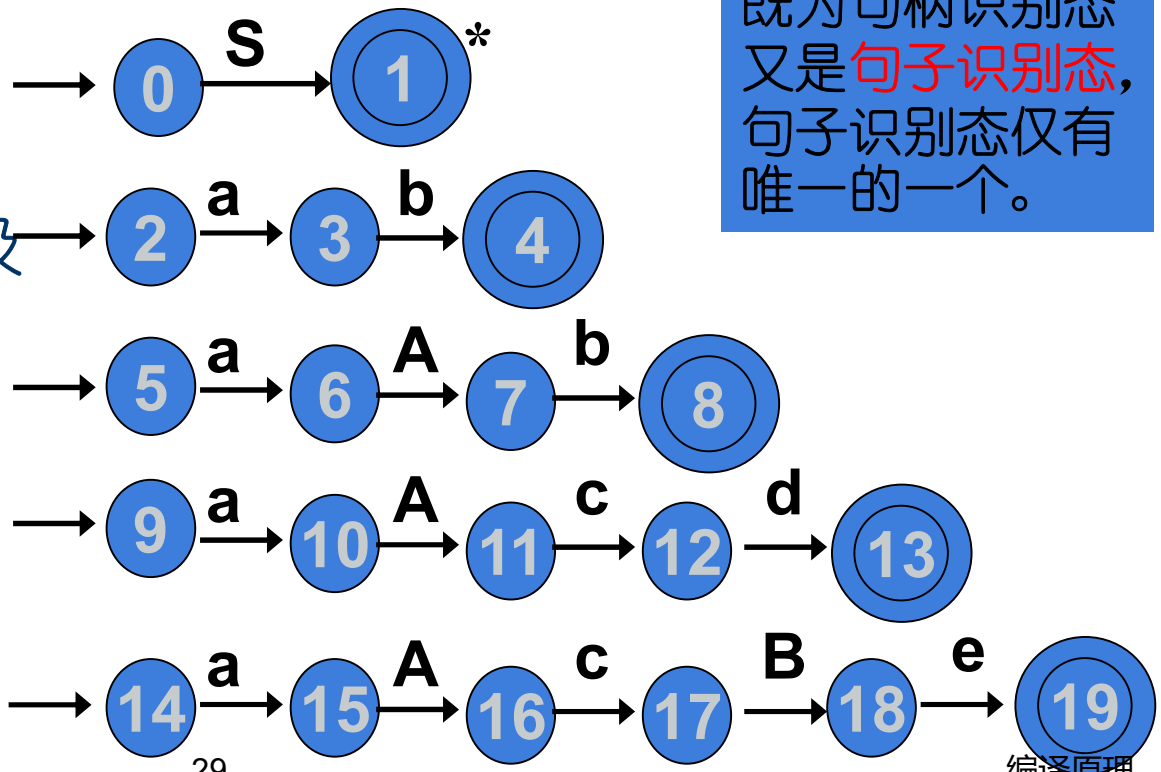
【例】

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

现对句子 **abbcde** 的可归前缀列出：

S[0]
ab[2]
aAb[3]
aAcd[4]
aAcBe[1]

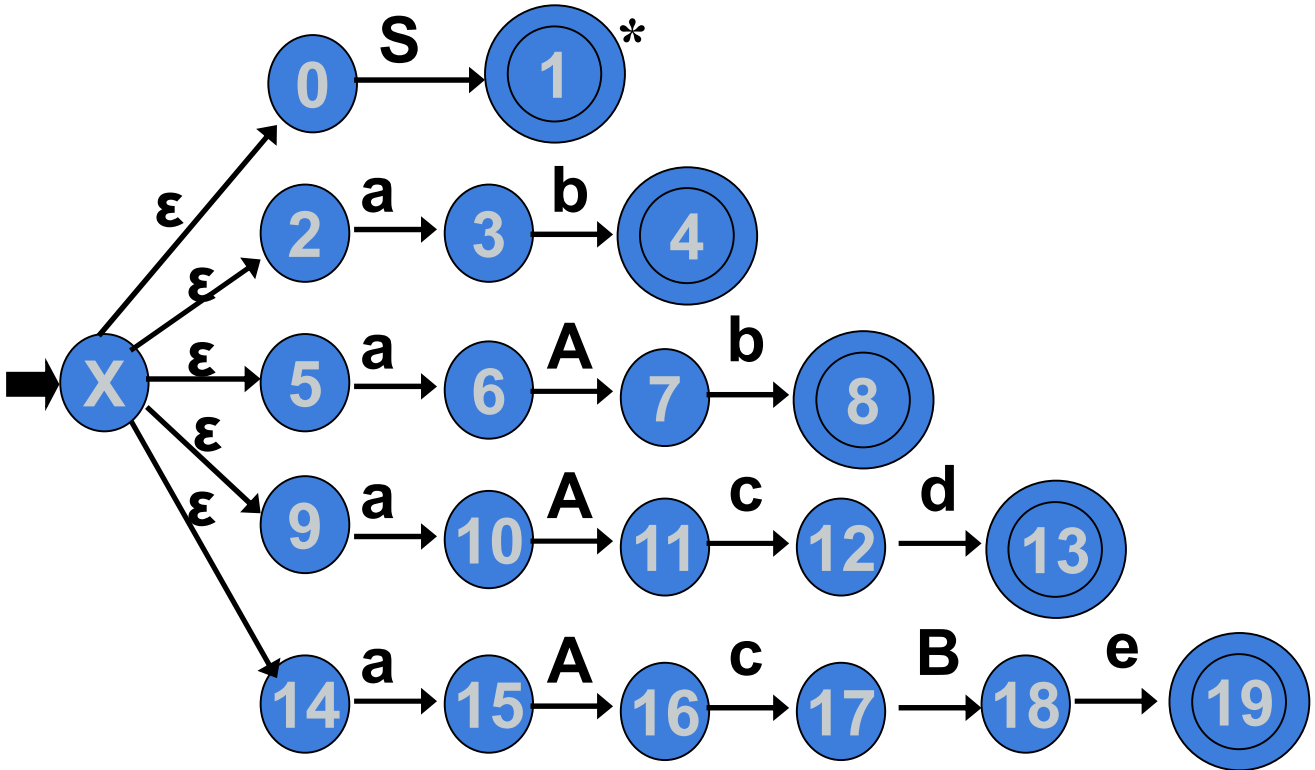
构造识别其活前缀及可归前缀的有限自动机如图



每一个终态都是句柄识别态，用双圈表示。
带"*"号的状态既为句柄识别态又是句子识别态，句子识别态仅有唯一的一个。

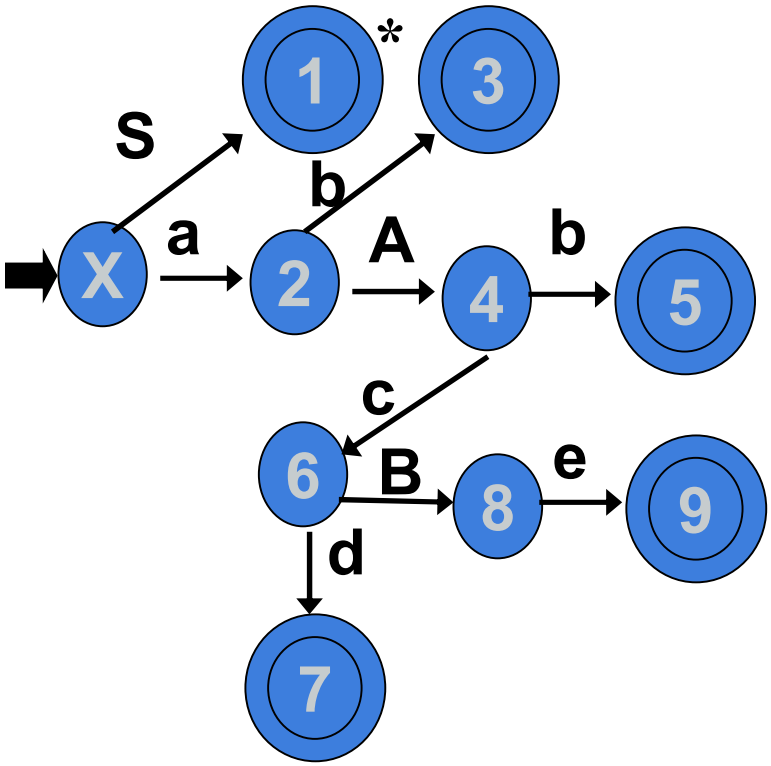
第四部分 LR(0)--识别活前缀的有穷自动机

- 如果加一个开始状态 X 并用ε弧和每个识别可归前缀的有限自动机连接，则可变为：



识别活前缀和可归前缀的NFA

将NFA确定化得到：
(确定化采用子集法)



识别活前缀和可归前缀的DFA

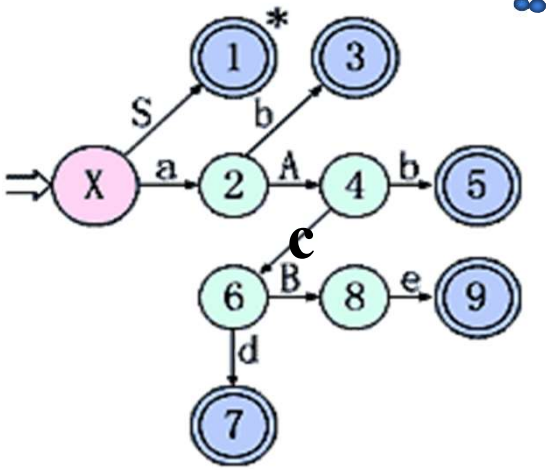
S[0]，ab[2]，aAb[3]，aAcd[4]，aAcBe[1]

理解识别活前缀和可归前缀的DFA和分析过程的对应

符号栈	输入串	动作
#	abbcde#	移进a
#a	bbcde#	移进b
#ab	bcde#	归约(A->b)
#aA	bcde#	移进b
#aAb	cde#	归约(A->Ab)
#aA	cde#	移进c
#aAc	de#	移进d
#aAcd	e#	归约(B->d)
#aAcB	e#	移进e
#aAcBe	#	归约(S->aAcBe)
#S	#	接受

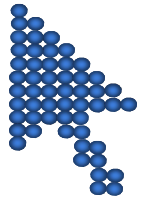
识别活前缀的有限自动机讨论

- 在上例中，用一个句子归约过程的所有活前缀和可归前缀构造出的有限自动机，刚好也是识别整个文法的活前缀及可归前缀的有限自动机。
- ——但这仅是一个特殊情况。



- 因此对一个上下文无关文法，只要能构造出它的识别活前缀及可归前缀的有限自动机，就可以构造其状态转换表和相应的**LR**分析表。
- ——虽然在理论上已经有严格的可归前缀计算方法，但对于一个复杂的文法，其可归前缀的计算过程也非常复杂。
- 我们下面介绍只一种由文法的产生式直接构造识别活前缀和可归前缀的有限自动机的实用方法。

LR(0)项目集规范族的构造（7.2.4）



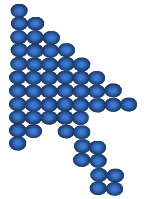
由活前缀的定义，活前缀和句柄之间的关系有三种：

1）活前缀中已经包含句柄的全部符号。
此时意味着某一产生式 $A\rightarrow\alpha$ 的右部符号串 α 已经出现在栈顶，相应的分析动作应该是按该产生式归约。

2）活前缀中只包含句柄的一部分。
此时意味着形如 $A\rightarrow\alpha_1\alpha_2$ 产生式的右部子串 α_1 已经出现在栈顶，正期待着从剩余的输入串中能移进或归约得 α_2。

3）活前缀中不包含句柄的任何符号。
此时意味着期待从剩余的输入串中能移进或归约得到某产生式 $A\rightarrow\alpha$ 的右部α。

LR(0)项目集规范族的构造（7.2.4）



- 为了刻画在分析过程中文法的一个产生式右部已经有多大一部分被识别，可在每个产生式右部某个位置上加一个圆点来表示。针对上述三种情况，标有圆点的产生式分别为：
- $A \rightarrow \alpha \cdot$
- $A \rightarrow \alpha 1 \cdot \alpha 2$
- $A \rightarrow \cdot \alpha$
- 我们把文法 **G** 中右部添加一个圆点的产生式称为文法 **G** 的一个 **LR(0)**项目。
- 特别地， $A \rightarrow \varepsilon$只有一个项目 $A \rightarrow \cdot$

【例1】

例如，产生式 **A→XYZ** 有四个 **LR(0)** 项目：

A→•XYZ

A→X•YZ

A→XY•Z

A→XYZ•

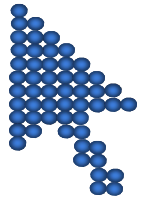
一个规则可对应的项目个数是它的右部符号长度加1。

每个项目的含义与圆点的位置有关：

概括地说，圆点的左部表示分析过程的某时刻用该产生式归约时句柄已识别过的部分，圆点右部表示待识别的部分。

只要已扫描过的部分保持可归约成一个规范句型的活前缀，那就意味着所扫描过的部分是正确的。为此，我们把构造识别文法所有活前缀的有限自动机 **NFA** 的每个状态由 **LR(0)** 项目构成。

【例】规则 S→aAcBe 对应有6个项目。



- 项目[0] 意味着希望用 S 的右部归约，当前输入串中符号应为 a；
- 项目[1] 表明用该规则归约已与第一个符号 a 匹配过了，需分析非终结符 A 的右部；
- 项目[2] 表明 A 的右部已分析完归约成 A，目前希望遇到输入串中的符号为 c；
- 以此类推直到 项目[5] 为 S 的右部都已分析完毕，则句柄已形成可以进行归约。

[0] S→·aAcBe
[1] S→a·AcBe
[2] S→aA·cBe
[3] S→aAc·Be
[4] S→aAcB·e
[5] S→aAcBe·

LR(0)项目进一步分类

一个 LR(0)项目的直观意义在于它指明了对文法规范句型活前缀的不同识别状态。

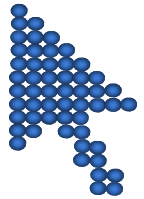
1）归约项目：形如 $A\rightarrow\alpha\bullet$ 的项目，其中，$\alpha\in(V_T|V_N)^*$。
它表示栈顶已形成句柄 α，下一步动作应该是归约。

2）移进项目：形如 $A\rightarrow\alpha\bullet a\beta$的项目。其中，$a\in V_T$，
$\alpha,\beta\in(V_T|V_N)^*$ 。
它表示期待从输入串中移进一个符号，已待形成句柄。

3）待约项目：形如 $A\rightarrow\alpha\bullet B\beta$的项目。其中，$B\in V_N$，
$\alpha,\beta\in(V_T|V_N)^*$。
它表示期待从输入串中进行归约而得到 B，然后进一步得到 A 的全部右部。

4）接受项目：形如 $S'\rightarrow\alpha\bullet$ 的项目。S′是文法开始符，以 S′为左部的规则只有一个（通过对一般文法拓广得到的），所以，该归约项目是一个特殊的归约项目，它表示整个句子已分析完毕，可以接受。

构造识别活前缀的有穷自动机



- 1) 构造识别活前缀的NFA
- a. 拓广文法G[S]为G'[S']，即加入产生式S'→S
- b.列出文法的每个项目：以G'[S']的每个项目为NFA的一个状态，S'→•S为初态，其余每个状态都为活前缀的识别态，所有归约项目为终态(句柄识别态)，S'→S•为接受态

以文法G'[S']为例：

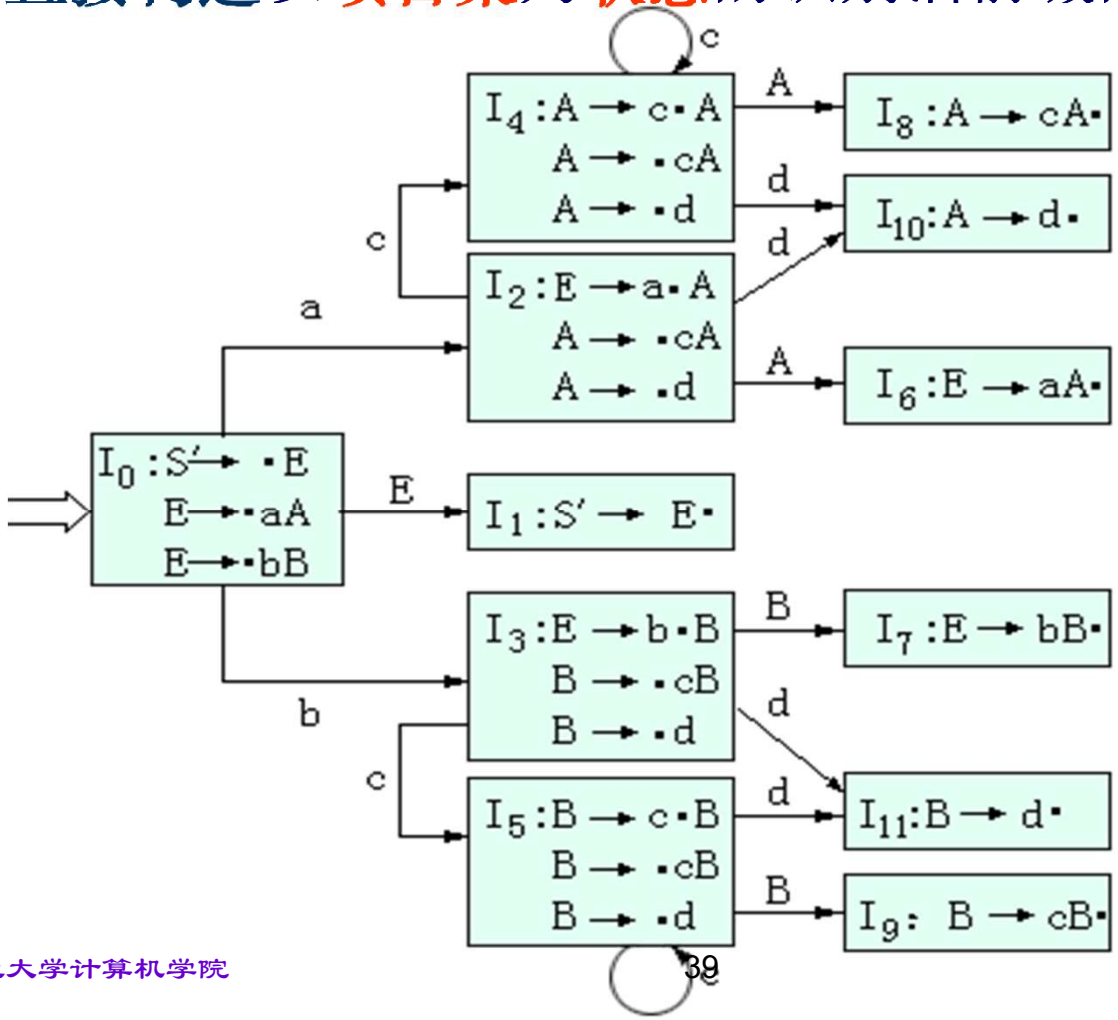
(1) S′→E (2) E→aA|bB (3) A→cA|d (4) B→cB|d

1. S′→•E	7. A→c•A	13. E→bB•
2. S′→E•	8. A→cA•	14. B→•cB
3. E→•aA	9. A→•d	15. B→c•B
4. E→a•A	10. A→d•	16. B→cB•
5. E→aA•	11. E→•bB	17. B→•d
6. A→•cA	12. E→b•B	18. B→d•

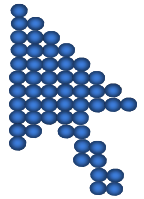
- 圆点在最后的项目为句柄识别态。
- 第一个产生式的句柄识别态为句子识别态。

构造识别活前缀的有穷自动机

2）直接构造以项目集为状态的识别活前缀的DFA



构造识别文法所有规范句型活前缀DFA



LR(0)项目集：识别文法活前缀的 DFA 的每一个状态都是由若干个 LR(0) 项目组成的集合，这个集合称为 LR(0) 项目集。（上图中一个DFA的状态就是一个项目集）

LR(0)项目集族：所有 LR(0)项目集组成的集合。

识别活前缀的DFA的构造

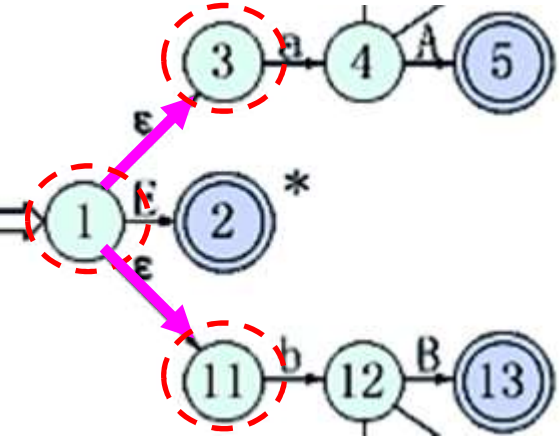
- 如何构造DFA的一个状态（项目集）——闭包函数
- 如何由DFA的一个状态求其他状态——状态转换函数

为了使接受项目唯一，对文法拓广，即对原文法G[S]增加S′→S，在G[S′]中，$S' \in V_N'$，$V_N' = V_N \cup \{ S' \}$。

项目集I的闭包CLOSURE(I)

- 对于拓广文法，设I是文法G'的项目集，
- 则定义和构造CLOSURE(I)：
- (1) 项目I ∈ CLOSURE(I)

1. S′→•E
3. E→•aA
11. E→•bB



- (2) 若X→α•Eβ ∈ CLOSURE(I)，则有每
- 一形如E→•γ的项目∈ CLOSURE(I)。

例如：令I = 项目1

CLOSURE(I) = {
S′→•E,
E→•aA,
E→•bB
}

- (3)重复(2)直到不出现新的项目为止。
- 即CLOSURE(I)不再扩大。
- CLOSURE(I)作为DFA的一个状态

- 实际上，CLOSURE(1)集是DFA的初态。
- 它含有文法开始符号为左部、圆点也在最左端的项目。

CLOSURE(I)练习

S′→S	S→A
S→B	
A→aAb	A→c
B→aBb	B→d

- 令 *I* = {S’→.S}
- CLOSURE(I)={S’→•S, S→•A, S→•B, A→•aAb,
- A→•c, B→•aBb, B→•d}
= I0

- I0 即为初态的项目集。

状态转换函数GO(I, X)

- 由DFA的一个状态求其他状态通过状态转换函数
- 设I为文法G的某一项目集(状态)，$X \in V_N \cup V_T$,则

GO(I, X) = CLOSURE(J)

其中J = {任何形如A→αX•β的项目 | A→α• Xβ∈I }称J为“核”

例 S’→E　E→aA | bB　A→cA|d　B→cB | d

I = { S’→•E，E→•aA，E→•bB }，则

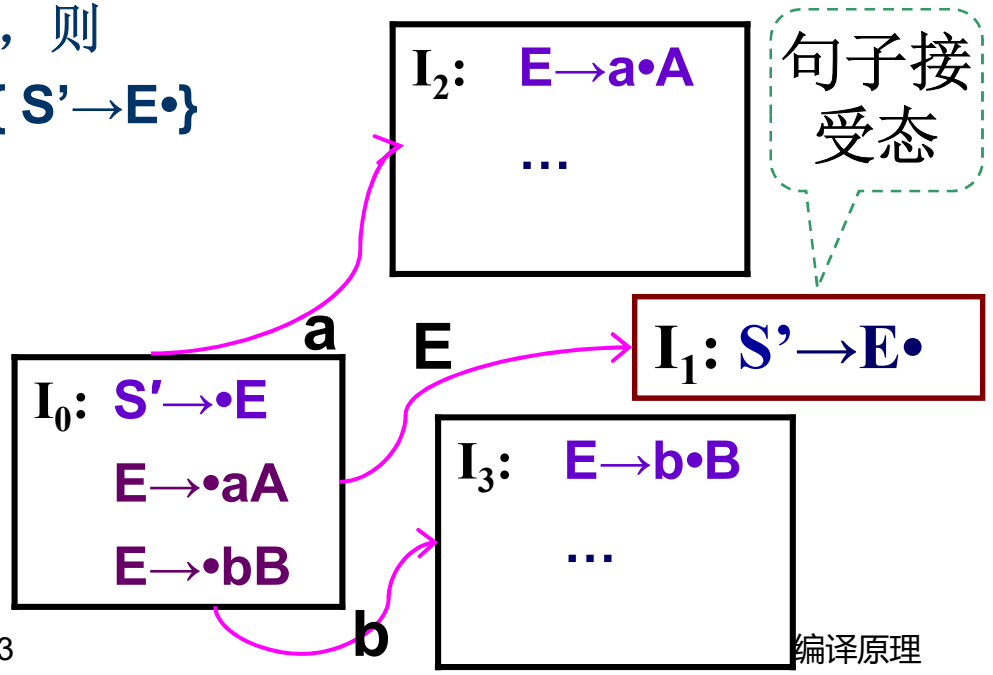
GO(I, E)=CLOSURE({ S’→E•})={ S’→E•}

GO(I,a)=CLOSURE({E→a•A})

={E→a•A ,A→•cA,A→•d }

GO(I,b)=CLOSURE({ E→b•B })

={ E→b•B ,B→•cB,B→•d }



Go函数练习

S′→S	S→A
S→B	
A→aAb	A→c
B→aBb	B→d

I_0 = {S′→•S, S→•A, S→•B, A→•aAb,
A→•c, B→•aBb, B→•d}

GO(I_0,S) = CLOSURE({S′→S•}) = {S′→S•} = I_1

GO(I_0,A) = CLOSURE({S→A•}) = {S→A•} = I_2

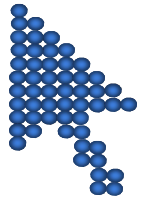
GO(I_0,B) = CLOSURE({S→B•}) = {S→B•} = I_3

GO(I_0,a) = CLOSURE({A→a•Ab, B→a•Bb})
= {A→a•Ab, B→a•Bb,
A→•aAb, A→•c,
B→•aBb, B→•d} = I_4

GO(I_0,c) = CLOSURE({A→c•}) = {A→c•} = I_5

GO(I_0,d) = CLOSURE({B→d•}) = {B→d•} = I_6

构造以LR(0)项目集为状态的DFA

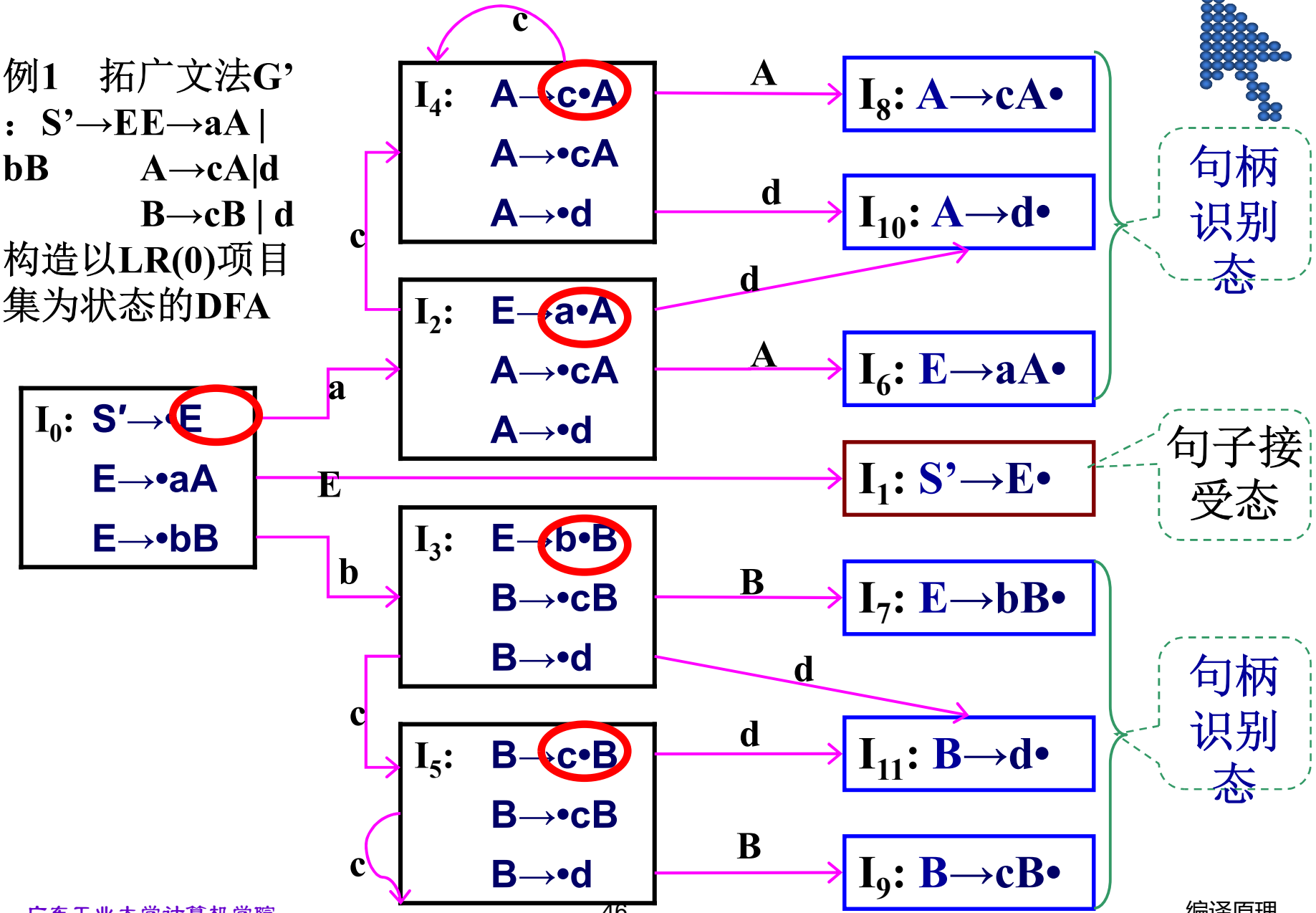


使用 GO 函数可以将拓广文法 G′的 LR(0)项目集规范族联结成一个识别文法活前缀的DFA。具体步骤如下:

- 求 Closure{S′→·S}，可得初态项目集 I0。
- 对已构造的项目集，应用状态转换函数 GO(I，X) 求它们的后继项目集（状态）。
- 重复第二步，直到不出现新的项目集为止。
- 状态转换函数 GO(I，X) 建立项目集之间的关系。

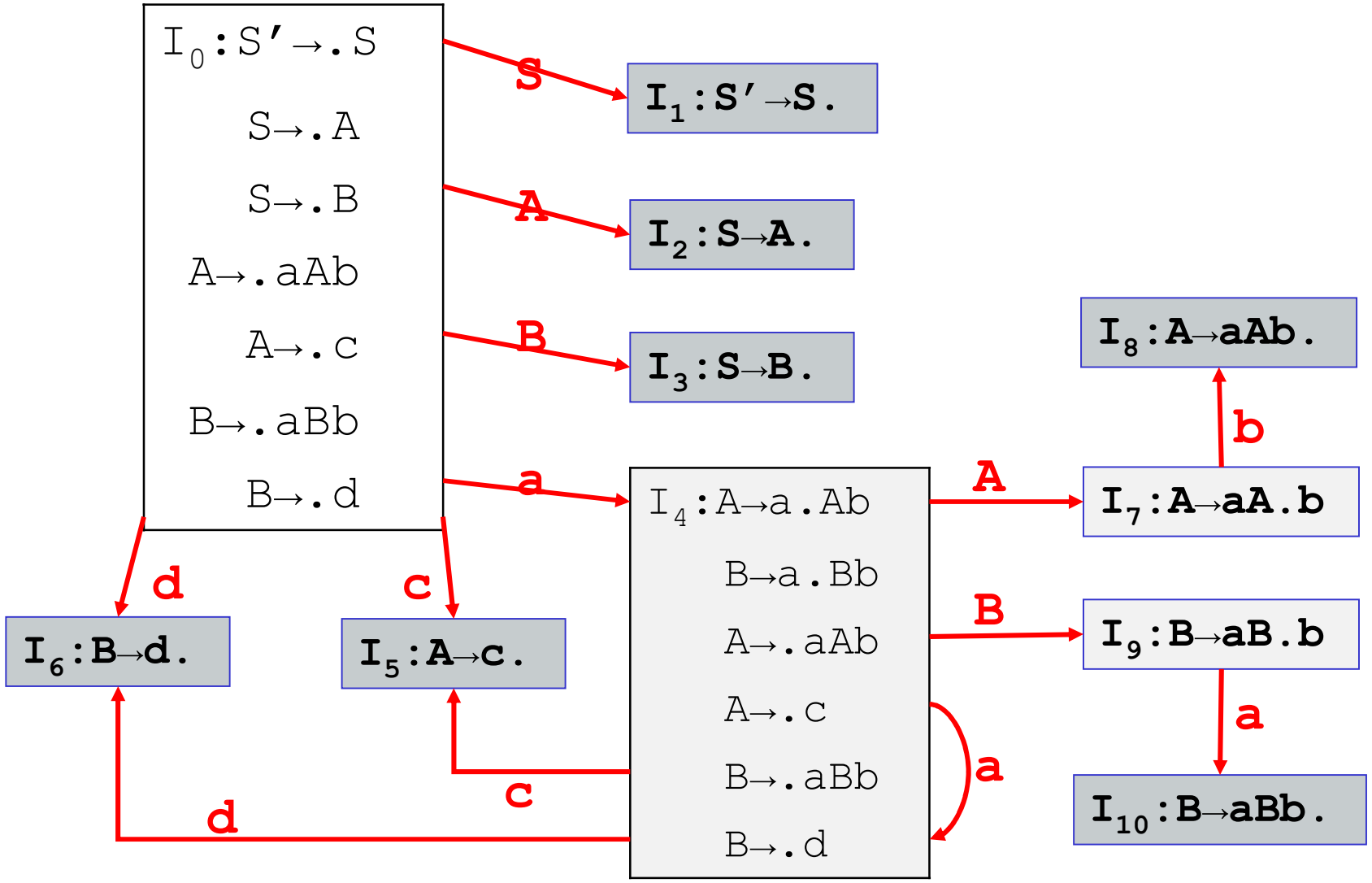
例1　拓广文法G’：S’→EE→aA | bB　　A→cA|d　　B→cB | d

构造以LR(0)项目集为状态的DFA



S′→S　　S→A　　S→B
A→aAb　　A→c　　B→aBb
B→d

【例】

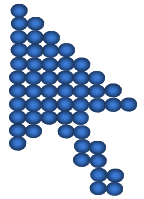


构造识别文法所有规范句型活前缀DFA

上述方法是把拓广文法的第一个项目**{S′→·S}**作为初态集的核，通过求核的闭包和转换函数，求出 **LR(0)**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**。

- 这种方法构造较简单不易出错

LR(0)项目集规范族的类型



- LR(0)项目集规范族的项目类型分为4种：

项目名称	定义	形如产生式	相应状态
移进项目	圆点后产生式为终结符	$A\rightarrow\alpha\bullet a\beta$ $\alpha, \beta \in V^*, a\in V_T$	移进状态
规约项目	圆点在产生式右部最后面	$A\rightarrow\beta\bullet$ $\beta \in V^*$	规约状态
待约项目	圆点后产生式为非终结符	$A\rightarrow\alpha\bullet B\beta$ $\alpha, \beta \in V^*, B\in V_N$	待约状态
接受项目	当规约项目为S’→S•时的项目	S’ →S•	接受状态

冲突类型

- 一个项目集中可能包含以上四种不同的项目，所以存在两种冲突：
- (1)移进-归约冲突：移进和归约项目同时存在

  形如：$A \rightarrow \alpha \cdot a\beta$

  $B \rightarrow \gamma \cdot$
- 面临输入符号为a时不能确定移进a还是把γ归约为B。

- (2)归约-归约冲突：归约和归约项目同时存在。

  形如：$A \rightarrow \beta \cdot$

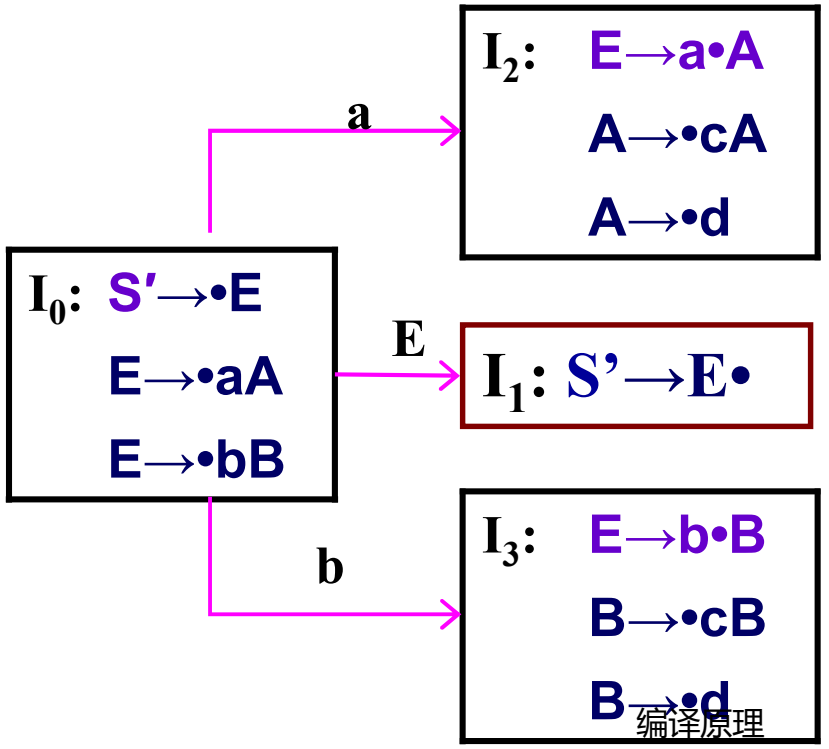
  $B \rightarrow \gamma \cdot$
- 在此项目集中，不能确定将句柄归约为A，还是归约为B。

- 对一个文法的LR(0)项目集规范族不存在移进-归约，或归约-归约冲突时，称这个文法为LR(0)文法。

LR(0)分析表的构造算法

- 假设已构造出LR(0)项目集规范族为：$C = \{I_0, I_1, \dots, I_n\}$，令项目集$I_k$对应的状态为k，含S'→•S项目的项目集对应的状态为初始状态
- 从初态开始，分析表的ACTION表和GOTO表构造步骤：
- (1)若项目$A \to \alpha \bullet a\beta \in I_k$，且$GO(I_k, a) = I_j$，则置ACTION[k,a]='$S_j$'，即将a和$S_j$移入分析符号栈和状态栈。移进
- .....
- (3)若项目$A \to \alpha \bullet B\beta \in I_k$，$B \in V_N$，且$GO(I_k,B)=I_j$，则置GOTO[k,B]='j'。
- (4) 若项目S' →S• $\in I_k$，则置
- ACTION[k, #]为 "acc".接受

状态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
0	S2	S3				1		
1					acc			
2	…	…	…	…	…	…	…	…

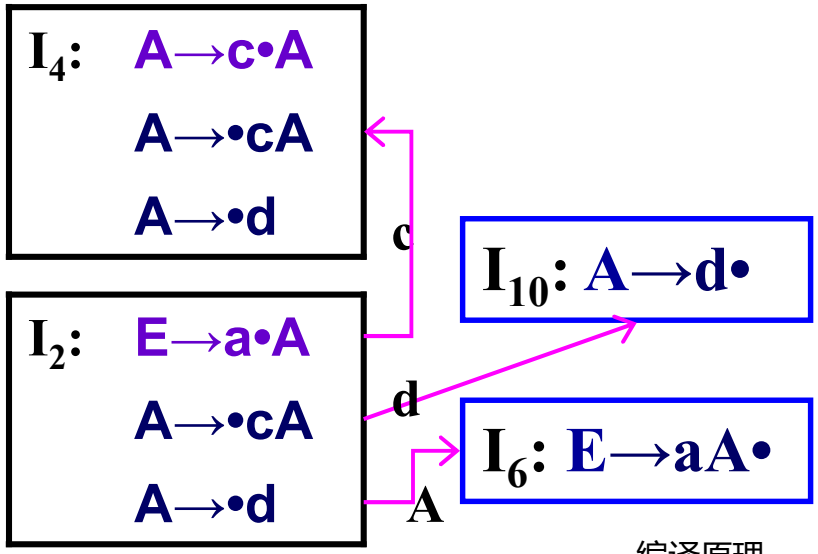


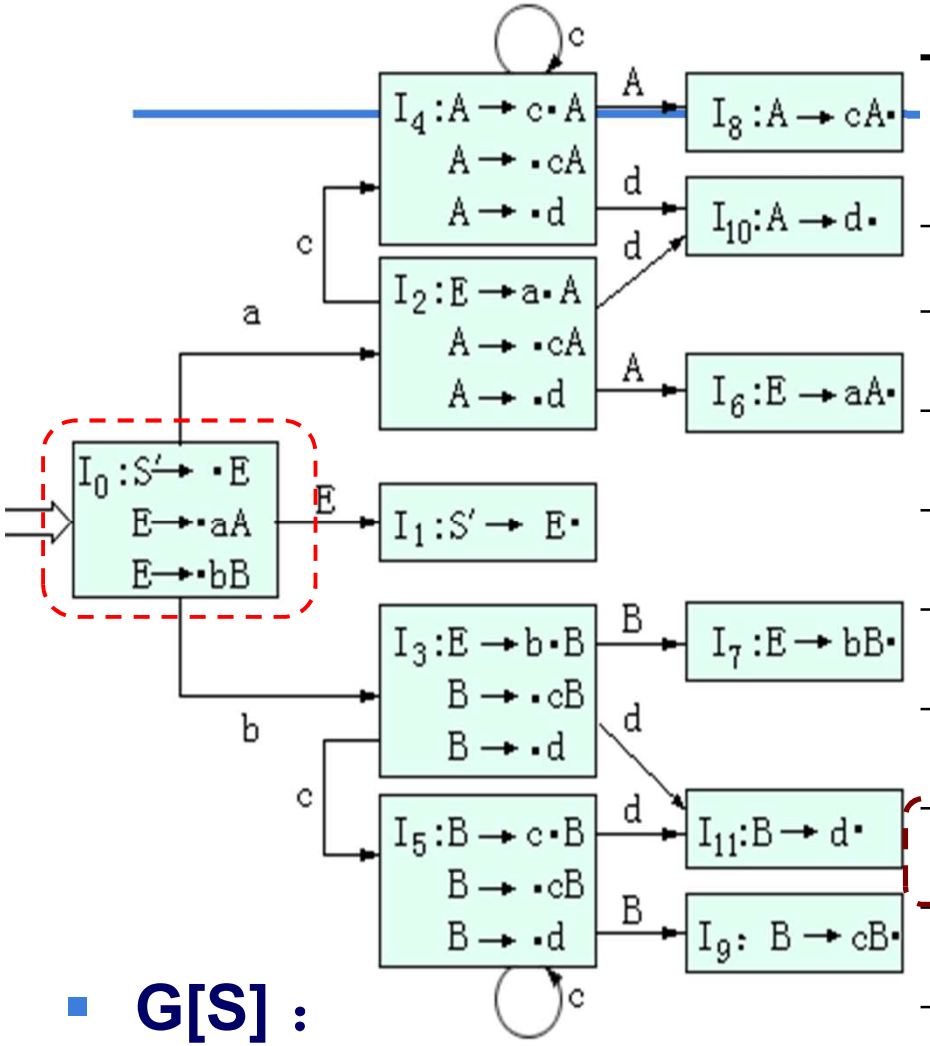
LR(0)分析表的构

- 算法步骤：
- (1) 若$A\to\alpha\bullet a\beta \in I_k$，且$GO(I_k$
- (2) 若$A\to\alpha\bullet \in I_k$，则对任意终结符x和“#”号，均令ACTION[k, x] = r_j，j为产生式$A\to\alpha$的序号。归约
- (3) 若$GO(I_k, A) = I_j$，在填写GOTO表$GOTO[I_k, A] = I_j$。
- .....

- G[S]：
- (0) S’→E　(1) E→aA
- (2) E→bB　(3) A→cA
- (4) A→d　(5) B→cB
- (6) B→d

状态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
2			S4	S10			6	
6	r1	r1	r1	r1	r1			
10	r4	r4	r4	r4	r4			

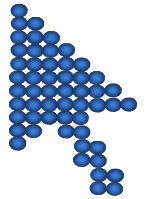




状态	ACTION					GOTO		
	a	b	c	d	#	E	A	B
0	S2	S3				1		
1					acc			
2			S4	S10			6	
3			S5	S11				7
4			S4	S10			8	
5			S5	S11				9
6	r1	r1	r1	r1	r1			
7	r2	r2	r2	r2	r2			
8	r3	r3	r3	r3	r3			
9	r5	r5	r5	r5	r5			
10	r4	r4	r4	r4	r4			
11	r6	r6	r6	r6	r6			

- G[S]：
- (0) S'→E (1) E→aA
- (2) E→bB (3) A→cA
- (4) A→d (5) B→cB
- (6) B→d

【综合实例】考虑文法G[S]： S→(S)|a



(1)构造识别文法规范句型活前缀的 **DFA**。

(2)判断该文法是否 **LR(0)** 文法，若是，构造 **LR(0)** 分析表；若不是，请说明理由。

第一步、文法拓广，并给出每条规则编号。

0. **S′→S**
1. **S→(S)**
2. **S→a**

第二步、构造识别文法规范句型活前缀的 DFA。

I_0=CLOSURE({S′→.S})={S′→.S,S→.(S),S→.a}

GO(I_0,S) = CLOSURE({S′→S•}) = {S′→S•} = I_1

GO(I_0,() = CLOSURE({S→(•S)})= {S→(•S),S→•(S), S→•a} = I_2

GO(I_0,a) = CLOSURE({S→a•}) = {S→a•} = I_3

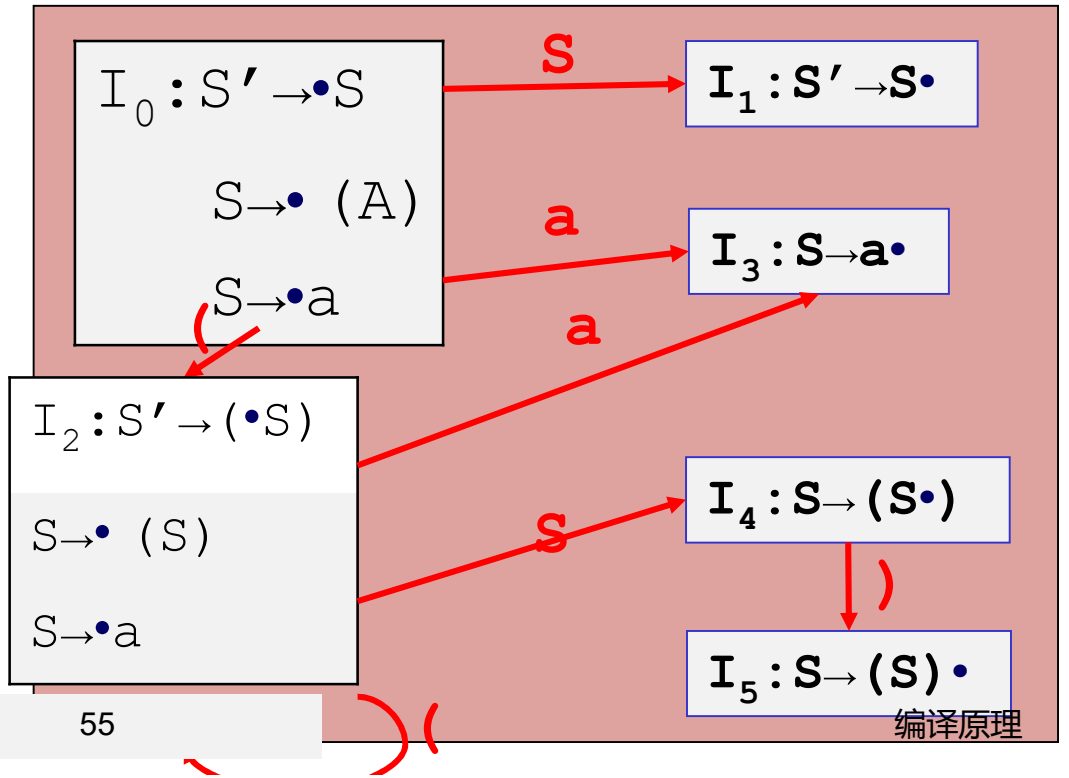
GO(I_2,a) = CLOSURE({S→a•}) = {S→a•} = I_3

GO(I_2,S) = CLOSURE({S→(S•)})= {S→(S•)} = I_4

GO(I_2,() = CLOSURE({S→(•S)})= {S→(•S),S→•(S), S→•a} = I_2

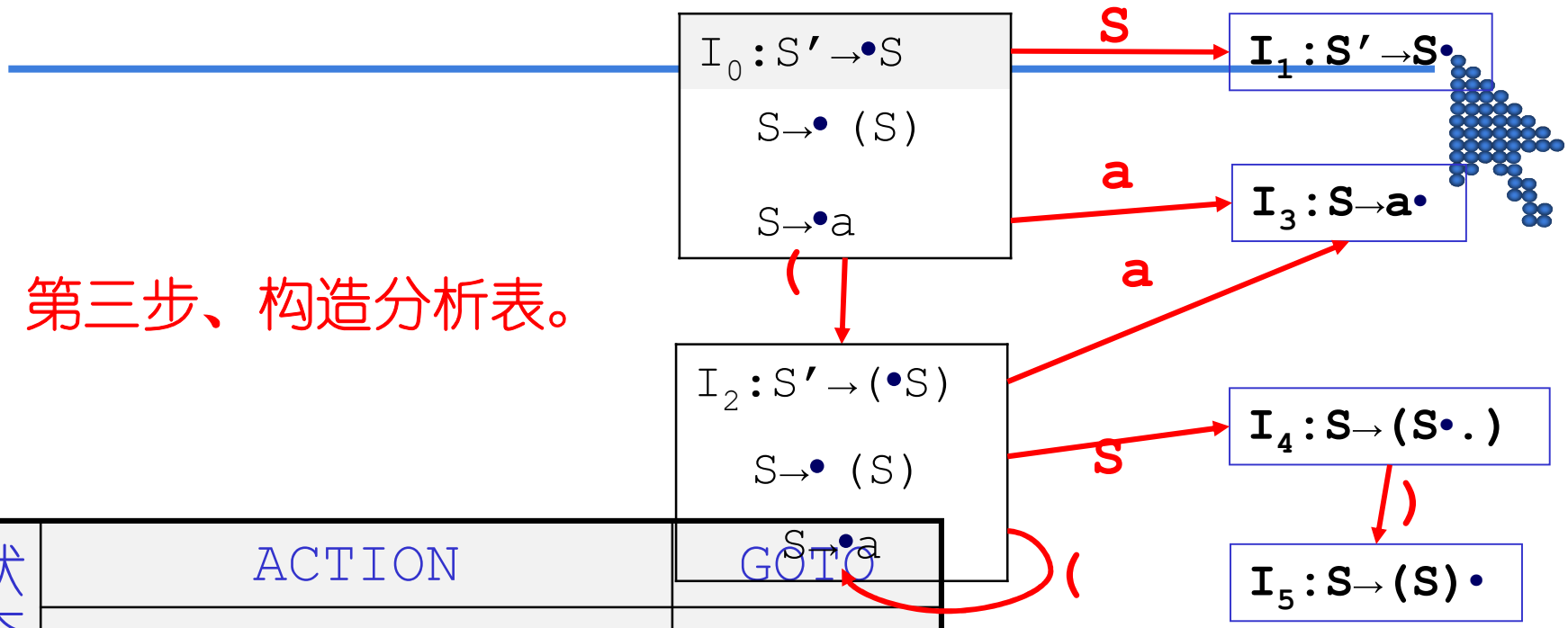
GO(I_4,)) = CLOSURE({S→(S)•})= {S→(S)•} = I_5

- 求 Closure{S′→•S}，可得初态项目集 I_0
- 对已构造的项目集，应用状态转换函数 GO(I,X) 求它们的后继项目集。



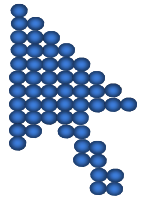
该文法是 LR(0) 文法。因为它的 LR(0) 项目集中均不含有冲突项目，即不存在移进和归约项目并存或多个归约项目并存的情况。

第三步、构造分析表。



状态	ACTION				GOTO
	a	(	)	#	S
0	S3	S2			1
1				acc	
2	S3	S2			4
3	r2	r2	r2	r2	
4			S5		
5	r1	r1	r1	r1	

LR(0)分析器特点



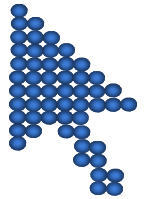
根据 LR(0)分析表的构造过程可以看出，LR(0)分析器的特点是不需要向前查看任何输入符号就能决定归约。

构造方法第二条

若 $A\rightarrow\alpha.$ 属于 I_k，那么对任何终结符a（或结束符#），置 $ACTION[k,a]=r_j$，“用第j条规则$A\rightarrow\alpha$进行归约”。

说明状态 k 为栈顶状态时，不管输入符号是什么，都按规则 j 归约，而不会发生错误。

第五部分 LR分析法之SLR(1)



- SLR(1)引入原因
  - LR(0)局限性：

    1）LR(0)方法对文法的要求严格。

    2）LR(0)方法容易出现冲突状态（归约-归约，归约-移进冲突 ）。

- 比如某个**LR(0)**规范族中有项目集：

  I={X→α·bβ，A→α·，B→β·}

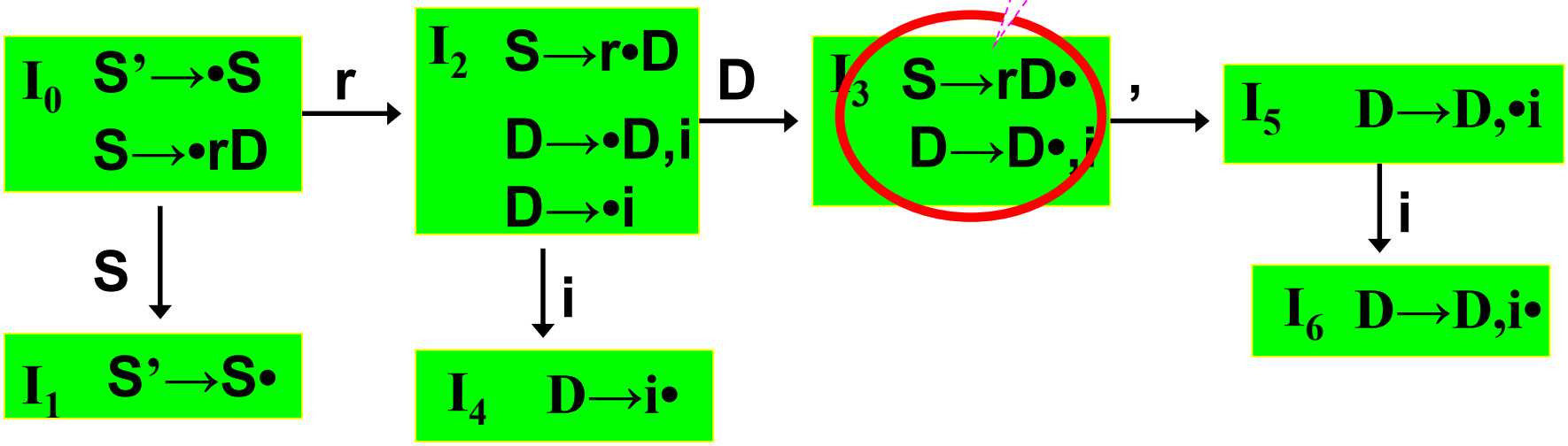
  则第一个项目和后两个归约项目产生移进－归约冲突，后两个归约项目产生归约－归约冲突。此时**LR(0)**分析法是无能为力的。
  - LR(0)受限原因：它要求每个状态只做一种分析动作，而且不考虑输入流信息。

【例】

产生移进-规约冲突

- 例如有实数说明文法的拓广文法为G'[S'，如下：

  (0) S′→S　(1) S→rD　(2) D→D,i　(3) D→i



【例】(续)

不管面临什么输入符号，都归约

当面临输入为","时，移进

- 对G'[S']:
  (0) S′→S　(1) S→rD　(2) D→D,i　(3) D→i
- 在I_3 = { S→ rD•　D→ D•,i }中存在移进规-约冲突，则其LR(0)分析表如下：
- (1) 对I_3中的移进项目D→ D•,i，期望当前输入字符是“,”。
- (2) 对I_3中的规约项目S→ rD•，会将当前分析栈栈顶的句柄“rD”规约为“S”。

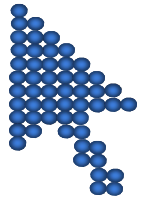
文法符号栈

#	r	D	

#	S	

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S2				1	
1				acc		
2			S4			3
3	r1	r1 S5	r1	r1		

第五部分 LR分析法之SLR(1)



- 解决冲突的SLR(1)方法

➢ 基本思想：

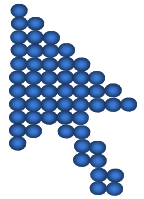
对于LR(0)有冲突的项目集用向前查看输入符号串的一个符号的办法加以解决

➢ 解决方法：

对归约项目A→r•，只有当输入符号a∈FOLLOW(A)才进行归约，缩小归约范围，有可能解决冲突。

- 注意：SLR(1)分析方法是简单的LR(1)分析方法，只有发生冲突时，才会向前查看一个输入符号。

SLR(1)分析举例



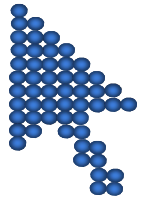
- 例如有实数说明文法为G'如下：
  (0) S'→S　(1) S→rD　(2) D→D,i　(3) D→i
- 在其LR(0)项目集规范族中有：I_3 = { S→ rD•　D→ D•,i }
- 例如输入串是：ri,i#。
- FOLLOW(S) = {#}；　FOLLOW(S) ∩{,} = ∅
- (1) 当前输入字符是“,”时，应当移进。
- (2) 当前输入字符是”#“时，应当规约。

移进项目的当前输入符号集

文法符号栈				
#	r	D		
#	r	D		

输入串当前状态			
	,	i	#
			#

解决冲突的办法



- 上面的例子给出了一种解决冲突的办法：假定一个**LR(0)**规范族中含有如下的项目集(状态)**I**

  I = {X→α•bβ，A→γ•，B→δ•}

- 其中**α,β,γ,δ**为文法符号串，**b**为终结符。只要满足

  (1) FOLLOW(A)∩FOLLOW(B) = ∅

  (2) FOLLOW(A)∩{b} = ∅

  (3) FOLLOW(B)∩{b} = ∅

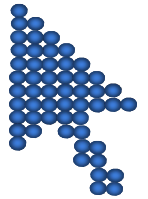
此式解决了什么问题？

- 那么，当**DFA**处于状态**I**，面临某个输入符号**a**时，则动作可由下规定决策：

回顾例子：I_3 = { S→ rD•　　D→ D•,i }

- **(1)** 若**a = b**，则移进。
- **(2)** 若**a∈FOLLOW(A)**，则用**A→γ**进行归约。
- **(3)** 若**a∈FOLLOW(B)**，则用**B→δ**进行归约。
- **(4)** 此外，报错。

解决冲突的办法扩展讨论



- 假设项目集I(状态)中有：
- m个移进项目：$A_1 \rightarrow \alpha_1 \bullet a_1 \beta_1$, $A_2 \rightarrow \alpha_2 \bullet a_2 \beta_2$, …, $A_m \rightarrow \alpha_m \bullet a_m \beta_m$
- n个归约项目：$B_1 \rightarrow \gamma_1 \bullet$，$B_2 \rightarrow \gamma_2 \bullet$，…,$B_n \rightarrow \gamma_n \bullet$

- 只要下列等式均成立：

  (1) $\{a_1, a_2, \ldots, a_m\} \cap FOLLOW(B_1) = \varnothing$

  (2) $\{a_1, a_2, \ldots, a_m\} \cap FOLLOW(B_2) = \varnothing$

  ……,

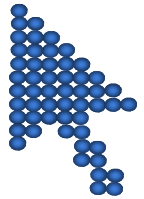
  (n) $\{a_1, a_2, \ldots, a_m\} \cap FOLLOW(B_n) = \varnothing$

  (n+1) $FOLLOW(B_1) \cap FOLLOW(B_2) \cap \ldots\ldots \cap FOLLOW(B_n) = \varnothing$

- 则仍可用根据当前输入符号a决定动作：
- (1) 若$a \in \{a_1, a_2, \ldots, a_m\}$，则移进。
- (2) 若$a \in FOLLOW(B_i)$，i = 1,2…n，则用$B_i \rightarrow \gamma_i$进行归约。

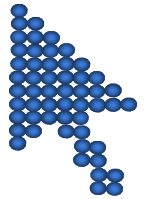
(3) 此外，报错。

SLR(1)分析



- 上述解决项目集(状态)中冲突的方法称为SLR(1)方法（Simple因为只对有冲突的状态才向前查看一个符号，以确定动作）
- 如果一个文法的LR(0)项目集规范族中某些项目集所含有的动作冲突都能用SLR(1)方法解决，则称这个文法为SLR(1)文法

SLR(1)判别



- 判断是否为SLR(1)文法：

1. 按照LR(0)方法把DFA构造出来；
2. 看该DFA的状态里面，是否存在冲突；
3. 若存在冲突，则看能否用SLR(1)的方法解决冲突。

例如：$I=\{X\rightarrow\alpha\bullet b\beta, A\rightarrow r\bullet, B\rightarrow\delta\bullet\}$,其中$b\in V_T$,

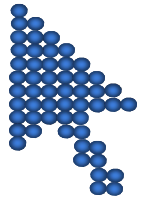
则要求： $FOLLOW(A)\cap FOLLOW(B)=\Phi$

$FOLLOW(A)\cap b=\Phi$

$FOLLOW(B)\cap b=\Phi$

为SLR(1)文法

SLR(1)分析表的构造



- SLR(1)分析表的构造与LR(0)分析表的构造类似，仅在含有冲突的项目集中分别进行处理。

- 实数说明文法为G′:
  (0) S'→S　(1) S→rD　(2) D→D,i　(3) D→i
- 在I_3 = { S→ rD•　D→ D•,i }中存在移进规约冲突，其LR(0)分析表:

状态	ACTION				GOTO	
	r	,	i	#	S	D
0	S2				1	
1				acc		
2			S4			3
3	r1	r1 S5	r1	r1		

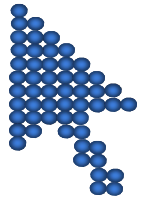
- 在I_3中:
- (1) 当前输入字符为“,”:
- 移进;
- (2) 当前输入字符为“#”:
- 规约

【例】设文法G[S']:

0. S′→S
1. S→Sb
2. S→bAa
3. A→aSc
4. A→aSb
5. A→a

1)构造识别文法规范句型活前缀的 **DFA**。
2)判断该文法是否 **SLR(1)** 文法。
若是，构造 **SLR(1)** 分析表；
若不是，请说明理由。

第一步、构造识别文法规范句型活前缀的 DFA。



I_0 = CLOSURE({S′→.S})={S′→.S, S→.Sb, S→.bAa}

GO(I_0,S) = CLOSURE({S′→S., S→S.b}) = {S′→S., S→S.b} = I_1

GO(I_0,b) = CLOSURE({S→b.Aa)})

= {S→b.Aa, A→.aSc, A→.aSb, A→a.} = I_2

GO(I_1,b) = CLOSURE({S→Sb.}) = {S→Sb.} = I_3

GO(I_2,A) = CLOSURE({S→bA.a}) = {S→bA.a} = I_4

GO(I_2,a) = CLOSURE({A→a.Sc, A→a.Sb, A→a.})

= {A→a.Sc, A→a.Sb, A→a., S→.Sb, S→.bAa} = I_5

GO(I_4,a) = CLOSURE({S→bAa.)})= {S→bAa.} = I_6

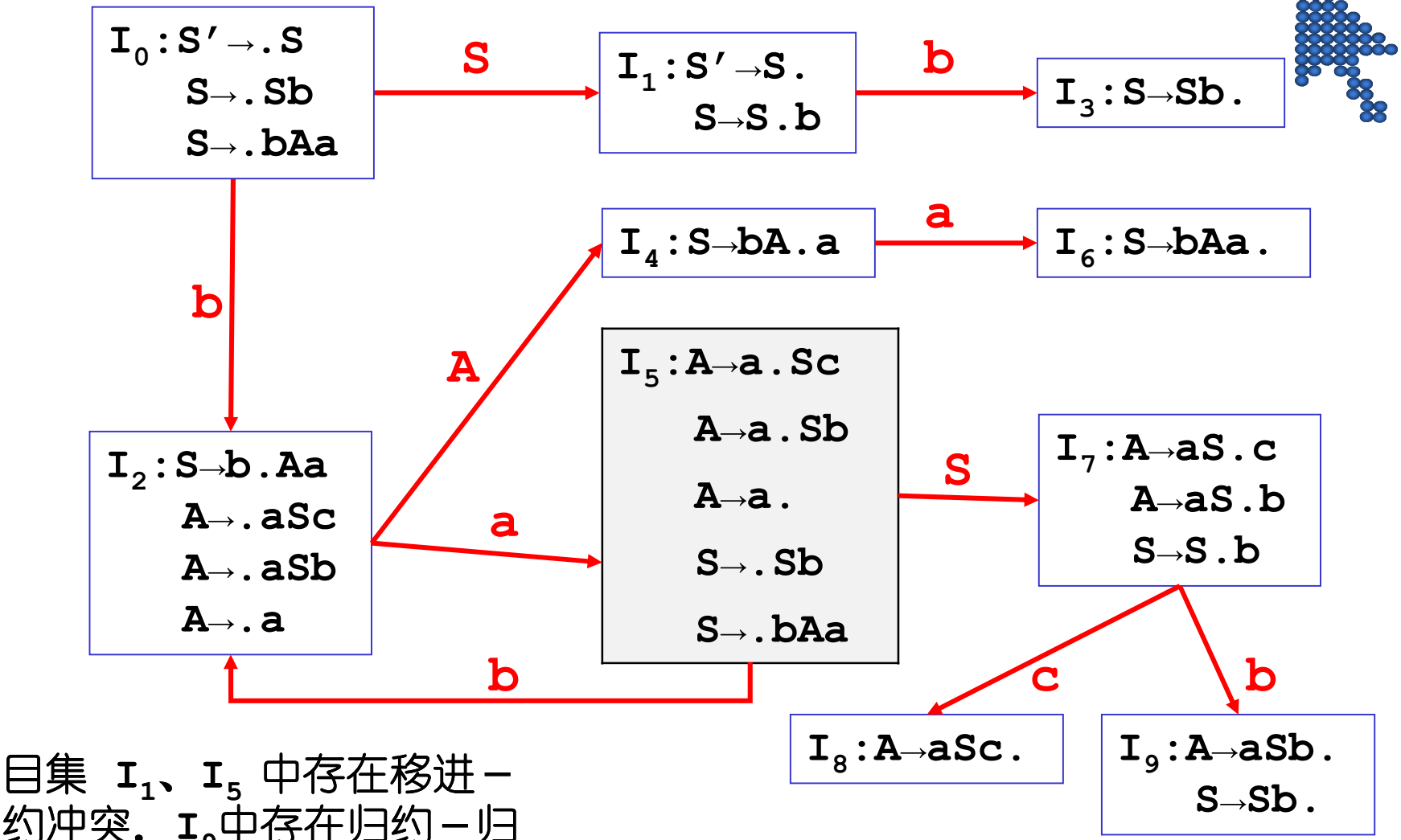
GO(I_5,S) = CLOSURE({A→aS.c, A→aS.b, S→S.b})

= {A→aS.c, A→aS.b, S→S.b} = I_7

GO(I_7,c) = CLOSURE({A→aSc.) = {A→aSc.} = I_8

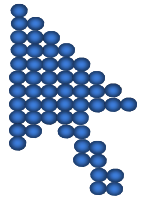
GO(I_7,b) = CLOSURE({A→aSb.) = {A→aSb.} = I_9

- 求 Closure{S′→·S}，可得初态项目集 I_0。
- 对已构造的项目集，应用状态转换函数 GO(I,X)求它们的后继项目集。



项目集 I_1、I_5 中存在移进－归约冲突，I_9中存在归约－归约冲突，因此该文法不是LR(0)文法。

第二步、考虑含冲突的项目集能否用 SLR(1) 方法解决。



1、 I_1 = {S′→S., S→S.b}

由于: FOLLOW(S′) ∩ {b} = {$} ∩ {b} = Φ

移进－归约冲突可以解决。

2、 I_5 = {A→a.Sc, A→a.Sb, A→a., S→.Sb,S→.bAa}

由于: FOLLOW(A) ∩ {b} = {a} ∩ {b} = Φ

移进－归约冲突可以解决。

3、 I_9 = {A→aSb., S→Sb.}

由于: FOLLOW(A) ∩ FOLLOW(S)

= {a} ∩ {b,c,$} = Φ

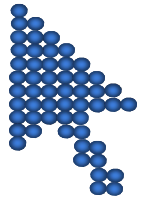
归约－归约冲突可以解决。

所以该文法是 SLR(1) 文法。

第三步、构造相应的 SLR(1) 分析表。

状态	ACTION				GOTO	
	a	b	c	#	S	A
0		S2			1	
1		S3		acc		
2	S5					4
3	r1	r1	r1	r1		
4	S6					
5	r5	S2			7	
6	r2	r2	r2	r2		
7		S9	S8			
8	r3	r3	r3	r3		
9	r4	r1	r1	r1		

SLR(1)的一种改进

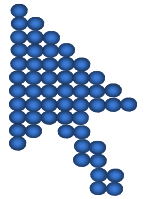


- 在**SLR(1)**文法中，仅在含有冲突的项目集中分别进行处理。还是会遇到无效规约的情况。

- 因此，可以对**SLR(1)**进行改进。
  - 若**A→ γ • ∈ I_k**，则对**x**为任何终结符或**'#'**，且满足**x∈ FOLLOW(A)**时，才有**ACTION[k, x] = r_j**。

  （对所有的归约项目仅对当前输入符号包含在该归约项目左部非终结符的**FOLLOW**集中，才采取归约动作。）

SLR(1) 小结

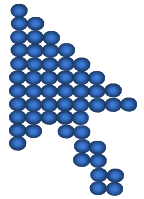


- 对于一个文法的**LR(0)**项目集规范族，如果所有的动作冲突都能用“向前看一个字符”的方法解决，则称这个文法是**SLR(1)**文法。

- 改进的**SLR(1)** 通过“规约时必须向前看一个字符”的方法，能够解决延迟发现错误的问题。

- **SLR(1)**分析方法对文法的要求比**LR(0)**宽松，允许文法中存在某些冲突的情况，但它并未能解决所有文法中可能出现的冲突。

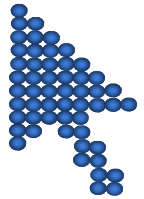
LR(1)分析法



⑩ 仔细分析一下 **SLR(1)** 方法，它在解决冲突时，仅孤立地考察对于归约项目 **A→α.** ,只要当前面临输入符号 **a∈FOLLOW(A)** 时，就确定使用规则 **A→α** 进行归约，而没有考虑符号串 α 所在规范句型的环境。

- 因为栈里符号串所构成的活前缀 **$δα** 未必允许把 α 归约到 **A**，因为可能没有一个规范句型含有活前缀 **$δAa**。
- 因此，这种情况下，把 α 归约到 **A** 未必有效。
- 也就是说，并不是 **FOLLOW(A)** 中的每个元素在含 **A** 的句型中都会出现在 **A** 的后面。某一个符号可能只在某一规范句型中 **A** 的后面出现。

SLR(1)存在多余规约



- 算术表达式文法：(0) S’ →E　(1) E→E+T　(2) E→T
  (3) T→T*F　(4) T→F　(5) F→(E)　(6) F→i
- 在项目集I_2 = {E→T•, T→T•*F}中存在移进-归约冲突：

文法符号栈　输入串当前状态　状态栈

#	E		

	)	#

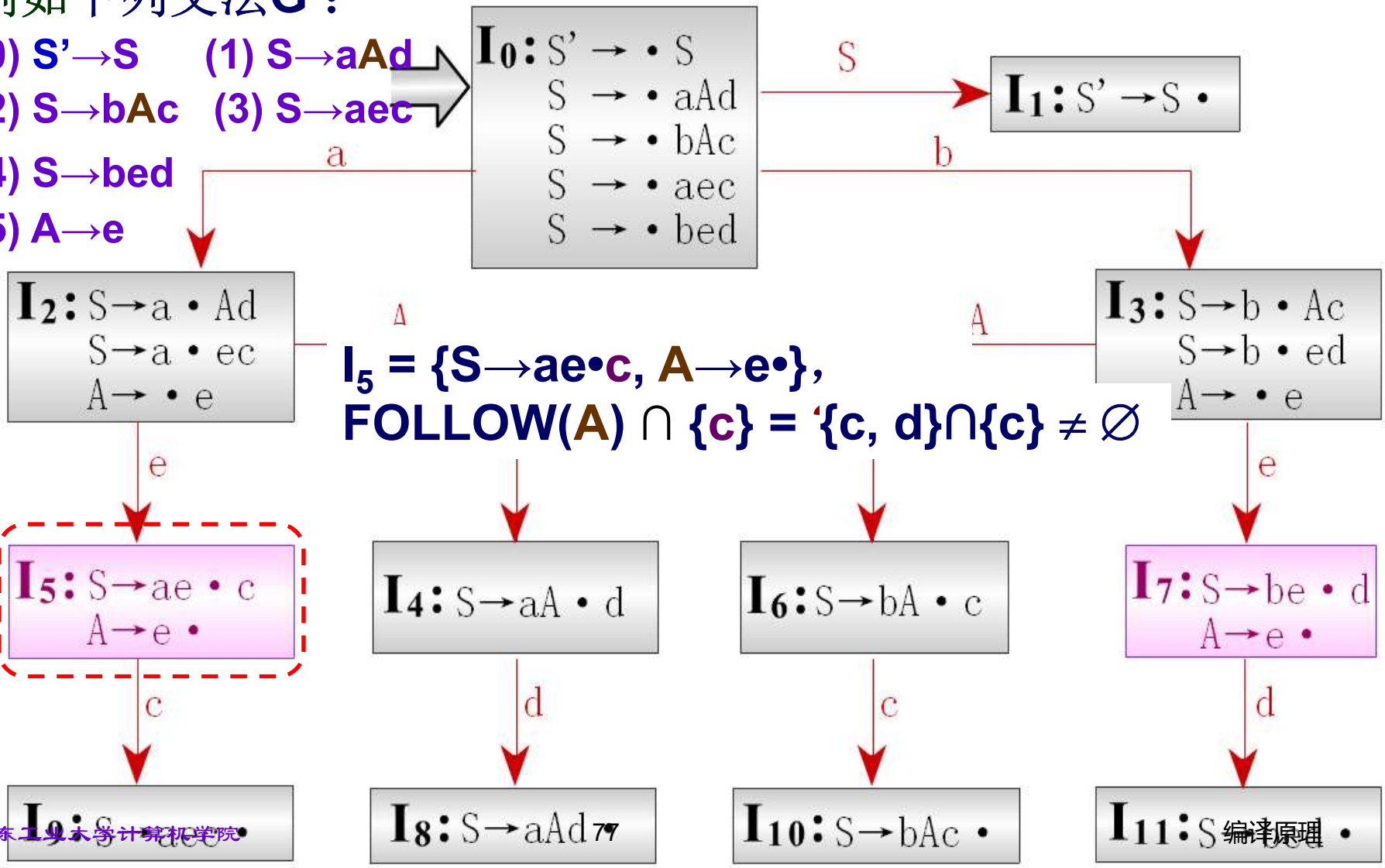
0	2		

- ∵ ‘)’ ∈ FOLLOW(E)
- ∴采用E→T规约，得到的句型#E)并非规范句型。
- 进而产生了一系列的多余规约。

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	S5			S4			1	2	
1		S6			出错	acc			
2		r2	S7		r2	r2		3	
3		r4	r4		r4	r4			

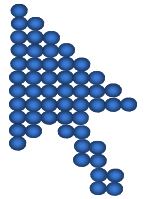
不能用SLR(1)分析的文法举例

- 例如下列文法G’：
- (0) S’→S　(1) S→aAd
- (2) S→bAc　(3) S→aec
- (4) S→bed
- (5) A→e



I_5 = {S→ae•c, A→e•},
FOLLOW(A) ∩ {c} = ‘{c, d}∩{c} ≠ ∅

举例讨论



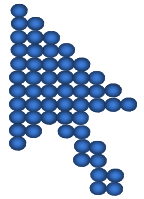
- 说明：含有A的句型中，不是FOLLOW(A)的所有元素都会在A的后面出现。

- 对下列文法G'：(0) S'→S　(1) S→aAd　(2) S→bAc
- 　　(3) S→aec　(4) S→bed　(5) A→e
- 进一步考察I_5 = {S→ae•c, A→e•}，FOLLOW(A) = {c, d}，有两个最右推导：
  S' ⇒ S ⇒ aAd ⇒ aed
  S' ⇒ S ⇒ aec

改写I_5 = { S→ae•c, #
A→e•, d }

- 对于活前缀ae，遇到输入符号c时，应该移进；遇到输入符号d时，应该将e规约为A。

LR(1)基本思想



- 在分析过程中，当试图用规则 **A→α** 归约栈顶的符号串 **α**时，不仅应该查看栈中符号串 **δα**，还应向前扫描一个输入符号**a(**称为向前搜索符**)**，只有当 **δAa** 的确构成文法某一句型的前缀时，才能用此规则进行归约。

- 换句话说， **LR(1)**方法按每个具体的句型决定采用的方式。

如果存在如下的一些句型

…αAa…，**…βAb…**，**…γAc…**，则**FOLLOW(A)={a,b,c}**

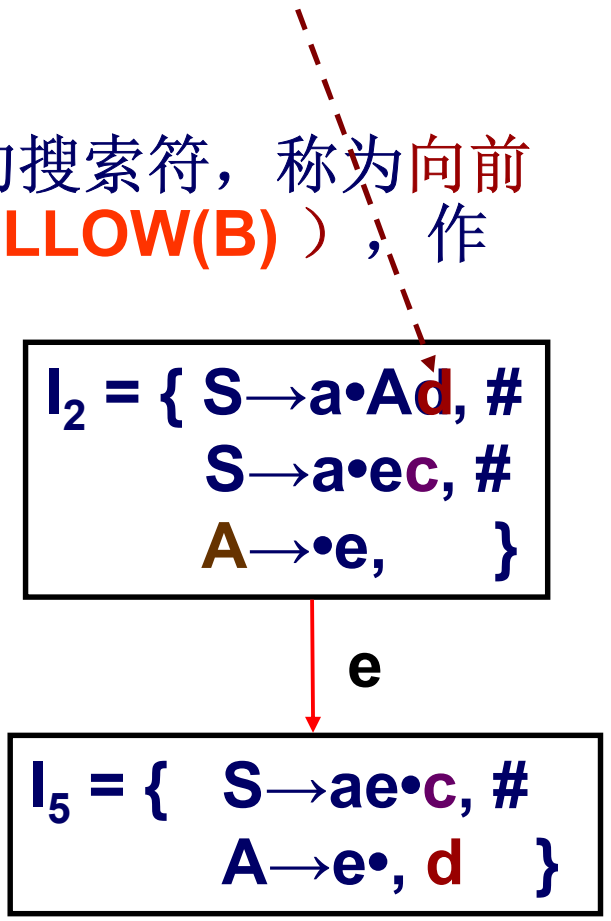
处理到句型**…αA**，只当输入符号为**a**时归约；

处理到句型**…βA**，只当输入符号为**b**时归约；

处理到句型**…γA**，只当输入符号为**c**时归约；

向前搜索符

- 在构造SLR(1)项目集规范族时，若(1) A→α•Bβ∈项目集I，则B→•γ也包含在I中。
- 则把FIRST(β)作为用(2) B→γ作归约的搜索符，称为向前搜索符（用以代替SLR(1)分析中的FOLLOW(B)），作为归约时查看的符号集合。
- 向前搜索符放在相应项目的后面。
- 特别地，如果β为空，则(1) 的向前
- 搜索符作为(2)的向前搜索符。
- 这种处理方法即为LR(1)方法。



采用向前搜索符和用 FOLLOW 集作为归约时向前查看的符号集合有什么区别?

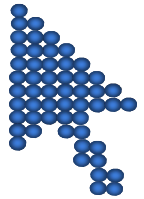
- **FIRST(β)⊆ FOLLOW（B）（子集合）**

- 原因是：一个非终结符号的 **FOLLOW** 集合，包含了所有含该非终结符的任一句型中在该非终结符后的向前搜索符集合。

- 而对于**LR(1)**分析法来说，如在项目集 **I** 中有项目：**[A→α • Bβ]，[B→ • γ]**。

  当分析经过若干步后在项目集 **J** 中含有项目**[B→γ·]**需要用产生式 **B→γ** 归约，这时向前查看的符号集合是 **FIRST(β)**，而 **FIRST(β)⊆ FOLLOW（B）**。

1、LR(1)项目集 I 的Closure(I)构造方法

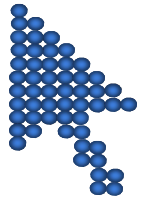


1) I 的任何项目都属于 Closure(I)
2) 若项目 [A→α•Bβ, a] 属于 Closure(I), 且 B→γ 是一个规则，对于每个 b∈First(βa)，[B→•γ, b] 属于Closure(I)。
3) 重复上述步骤，直到Closure(I)不再增大为止。

对项目[A→α•Bβ,a]，计算 B 的向前搜索符时，应为 FIRST(βa)，这是因为

β∈V*，即β可能为ε，而 a 是用产生式 A→αBβ 归约时的向前搜索符，而现在 β 为 ε，就等于用 A→αB 归约，向前搜索符为 a，那么用 A→αB 归约前，必须先用产生式 B→γ 归约成 B，因此，B 的向前搜索符时也应为 a。

LR(1) 项目集族的 GO 函数



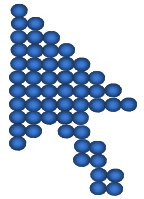
令 I 是一个 LR(1) 项目集，X 是一个文法符号($V_N \cup V_T$)，函数 GO (I,X) 定义为：

GO(I,X)=Closure(J)

其中：

J = {[A→αX•β, a] | [A→α•Xβ, a]∈I}

LR(1)项目集族的构造实例



- 对下列文法G'： (0) S'→S (1) S→aAd (2) S→bAc
- (3) S→aec (4) S→bed (5) A→e
- 构造它的LR(1)项目集规范族，令S'→ •S, # ∈ I_0。
- I_0: S'→ •S, #
- S→ •aAd, #
- S→ •bAc, #
- S→ •aec, #
- S→ •bed, #
- I_1: S'→ S•, #

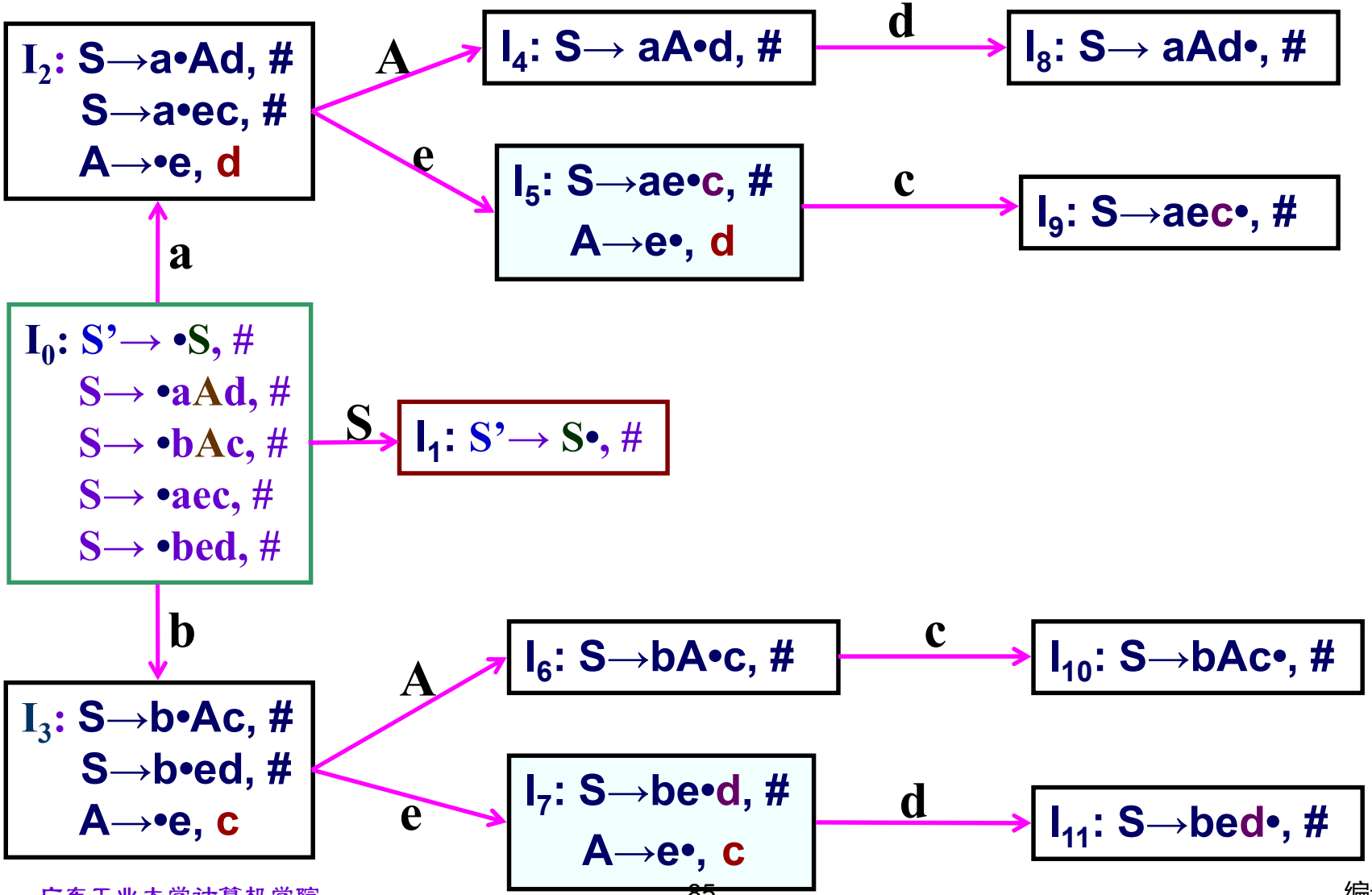
#表示：活前缀γ在面临输入符#时，才可规约成S

I_2: S→a•Ad, #
S→a•ec, #
A→•e, d

I_3: S→b•Ac, #
S→b•ed, #
A→•e, c

- [A→α•Bβ, a] ∈I_k，则B→•γ ∈I_k。
- B→•γ的向前搜索符 = FIRST(βa)

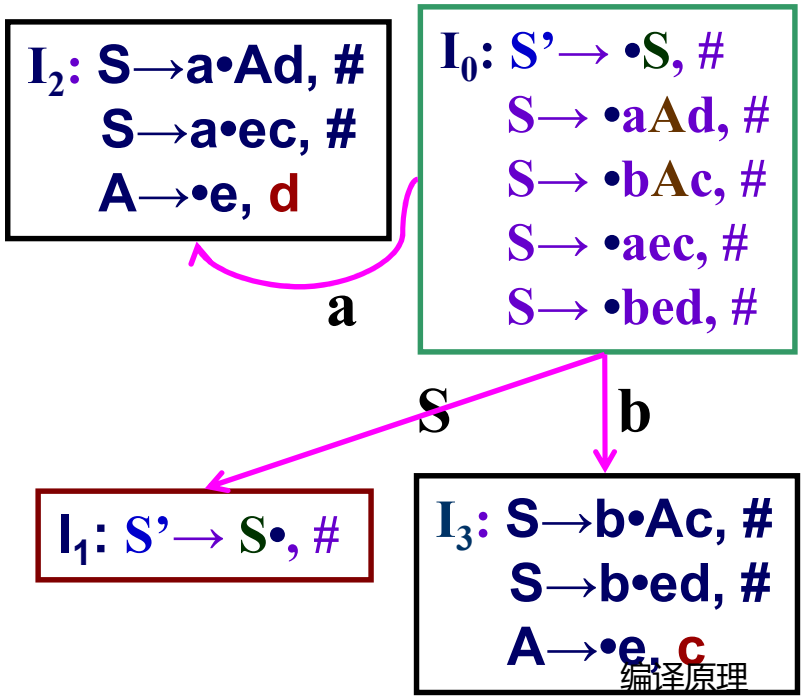
LR(1)项目集规范族构造举例(续)



LR(1)分析表的构造

- 假设已构造出LR(0)项目集规范族为：$C = \{I_0, I_1, \dots, I_n\}$，
- (1) 若$[A\rightarrow\alpha\bullet a\beta, b] \in I_k$，且$GO(I_k, a) = I_j$，则置$ACTION[k, a] = S_j$，即将a和$S_j$移入分析符号栈和状态栈。
- ……
- (3) 若$[S'\rightarrow S\bullet, \#] \in I_k$，则
- $ACTION[k, \#] = \text{“acc”}$。
- (4) 若$GO(I_k, A) = I_j$，其中$A \in V_N$，
- 则置$GOTO[k, A] = j$。

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



LR(1)分析表的构造(续)

- 假设已构造出LR(0)项目集规范族为：$C = \{I_0, I_1, \dots, I_n\}$，
- (1) 若$[A\rightarrow\alpha\bullet a\beta, b] \in I_k$，且$GO(I_k, a) = I_j$，则置$ACTION[k, a] = s_j$。
- (2) 若$[A\rightarrow\alpha\bullet, x] \in I_k$，$x \in VT$，则置$ACTION[k, x] = r_j$。j为在文法中对产生式$A\rightarrow\alpha$的编号
- ……

状态	ACTION						GOTO	
	a	b	c	d	e	#	S	A
2					S5			4
3					S7			6
4				S8				
5			S9	r5				

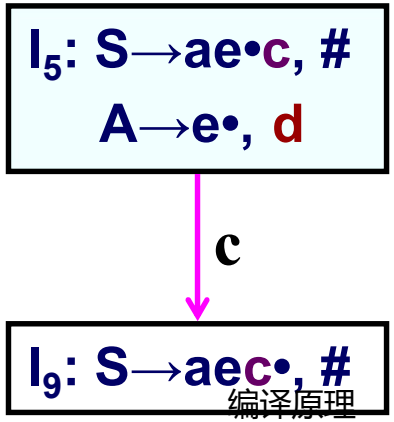
文法G'[S']:

(0) S'→S (1) S→aAd

(2) S→bAc (3) S→aec

(4) S→bed

(5) A→e

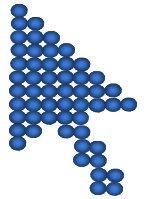


LR(1)分析表

- 由LR(1)分析表可以看出，对LR(1)的规约项目不存在任何无效规约。
- 如果一个文法的LR(1)分析表不含多重入口 (即任一LR(1)项目集中无移进-归约冲突或归约-归约冲突)，则称该文法为LR(1)文法。

状态	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				S10				
7				r5	S11			
8						r1		
9						r3		
10						r2		
11						r4		

【例】文法G[S]: S→(S)|a



试构造它的 **LR(1)** 项目集合的 **DFA** 和**LR(1)** 分析表。

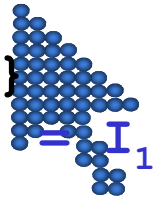
第一步、文法拓广，并给出每条规则编号。

0. S′→S

1. S→(S)

2. S→a

第二步、构造 LR(1) 项目集合的 DFA。



I_0=CLOSURE({[S′→.S,$]})={[S′→.S,$],[S→.(S),#],[S→.a,$]}

GO(I_0,S) = CLOSURE({[S′→S.,$]}) = {[S′→S.,$]} = I_1

GO(I_0,() = CLOSURE({[S→(.S),$]})

= {[S→(.S),$],[S→.(S),)],[S→.a,)]} = I_2

GO(I_0,a) = CLOSURE({[S→a.,$]}) = {[S→a.,$]} = I_3

GO(I_2,S) = CLOSURE({[S→(S.),$}) = {[S→(S.),$]} = I_4

GO(I_2,() = CLOSURE({[S→(.S),)]})

= {[S→(.S),)],[S→.(S),)],[S→.a,)]} = I_5

GO(I_2,a) = CLOSURE({[S→a.,)]}) = {[S→a.,)]} = I_6

GO(I_4,)) = CLOSURE({[S→(S),$]}) = {[S→(S).$]} = I_7

GO(I_5,S) = CLOSURE({[S→(S.),)]})= {[S→(S.),)]} = I_8

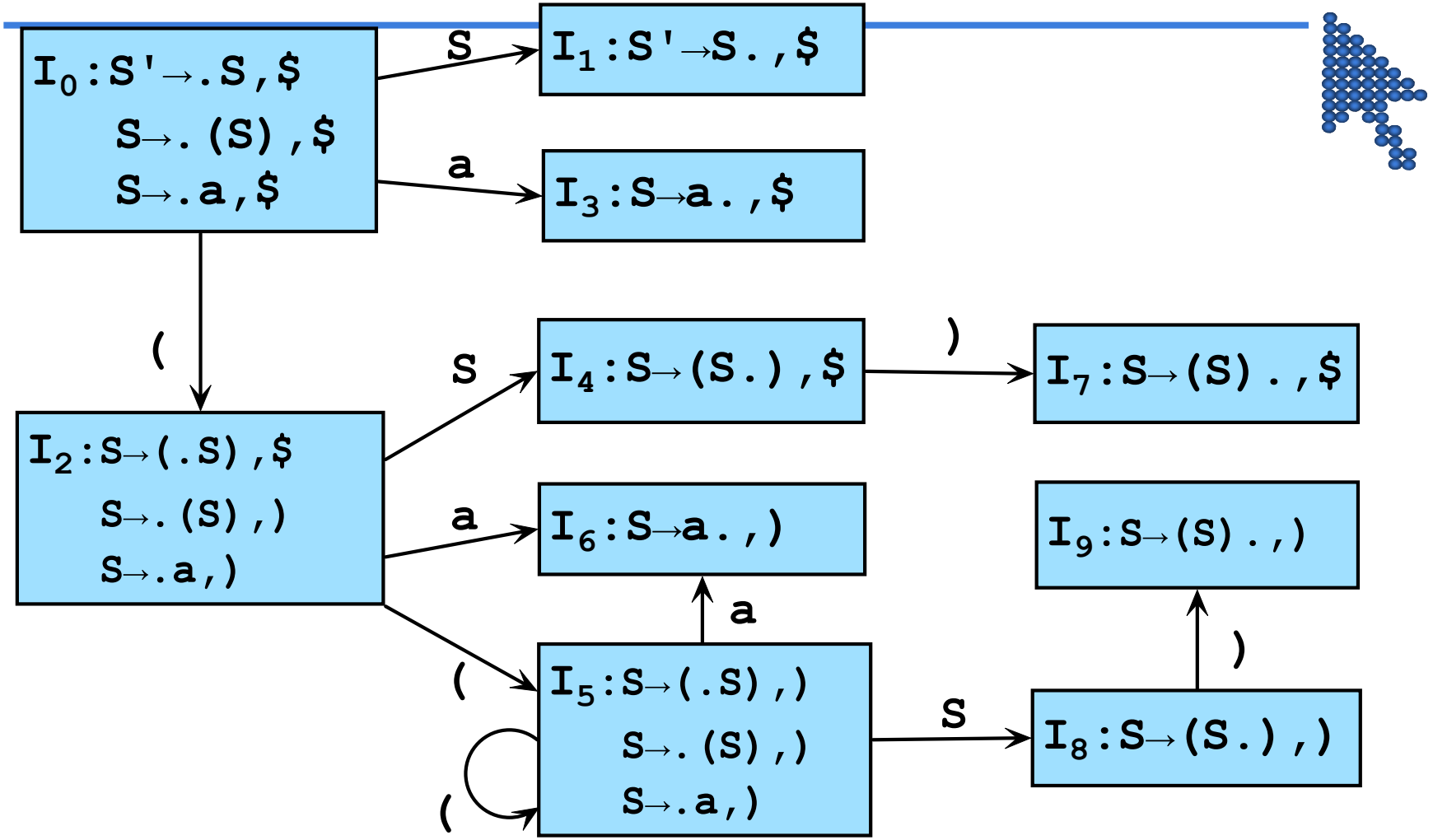
GO(I_5,() = CLOSURE({[S→(.S),)]})

= {[S→(.S),)],[S→.(S),)],[S→.a,)]} = I_5

GO(I_5,a) = CLOSURE({[S→a.,)]}) = {[S→a.,)]} = I_6

GO(I_8,)) = CLOSURE({[S→(S).,)]})= {[S→(S).,)]} = I_9

- 求 Closure{S′→·S,$}，可得初态项目集 I_0
- 对已构造的项目集，应用状态转换函数 GO(I,X) 求它们的后继项目集。

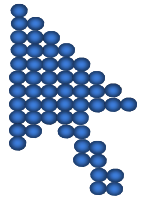


该文法是 LR(1) 文法。因为它的 LR(1) 项目集中均不含有冲突项目。

第三步、构造分析表。

状态	ACTION				GOTO
	a	(	)	$	S
0	S3	S2			1
1				acc	
2	S6	S5			4
3				r2	
4			S7		
5	S6	S5			8
6			r2		
7					

LR(1)与LR(0)及SLR(1)方法的区别



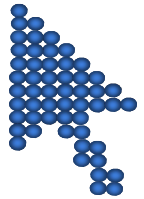
区别体现在构造分析表算法的步骤(2)上。

若项目 $A\rightarrow\alpha\cdot$ 属于 I_k，则当用产生式 $A\rightarrow\alpha$ 归约时：

1. LR(0)无论面临什么输入符号都进行归约动作；
2. SLR(1)则是仅当面临的输入符号$a\in FOLLOW(A)$时进行归约动作，而不判断栈里的符号串所构成的活前缀$\beta\alpha$是否存在着把α归约为A的规范句型——其前缀是βAa；
3. LR(1)则明确指出了当 α 后跟终结符 a （即存在规范句型其前缀为 βAa ）时，才容许把 α 归约为 A。

因此，LR(1)比SLR(1)更精确，解决的冲突也多于SLR(1)。

作业



- 作业格式：
- **(1)** 在每一次的作业开头，需要写上日期：

- **(2)** 每道题目的题号要写清楚

主观题 10分

设置

文法G[S]及其LR分析表如下，请给出串baba#的LR分析过程。

(1) S → DbB　　(2) D → d　　(3) D → ε

(4) B → a　　(5) B → Bba　　(6) B → ε

状态	ACTION				GOTO		
	b	D	a	#	S	B	D
0	r3	s3			1		2
1				acc			
2	S4						
3	r2						
4	r6		S5	r6		6	
5	r4			r4			
6	S7			r1			
7			S8				
8	r5			r5			

(注：答案格式为 步骤　　状态栈　符号栈　输入串 ACTION　GOTO)

作答

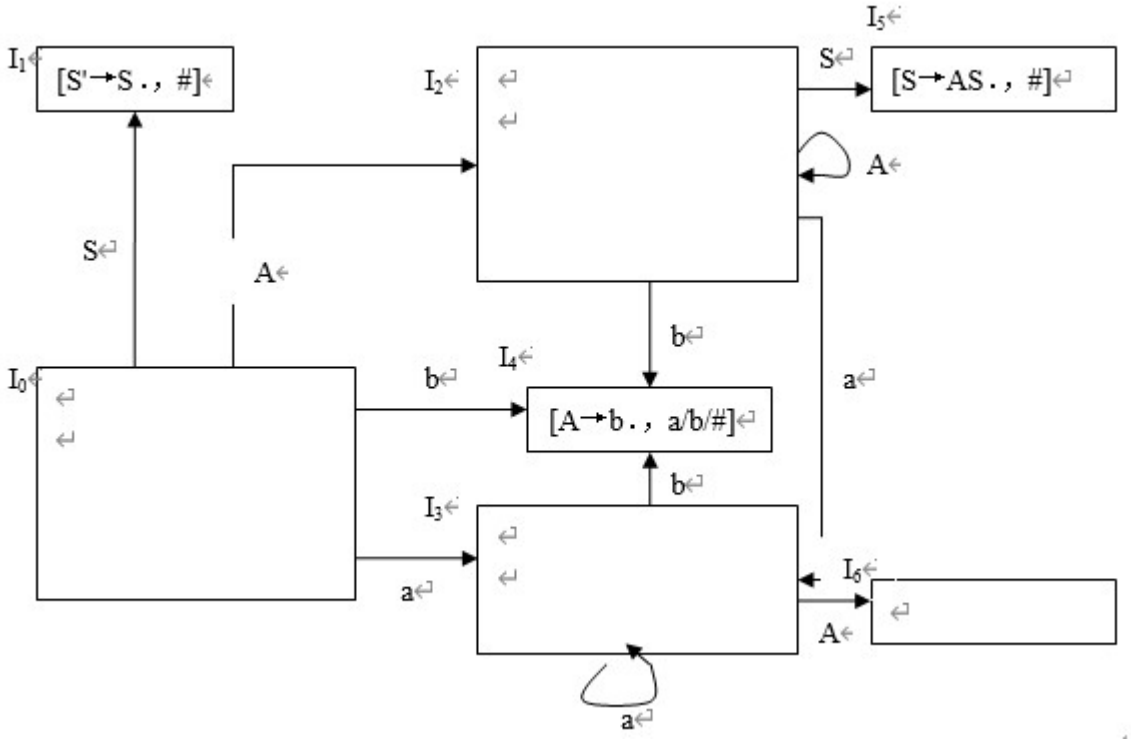
主观题 10分

设置

已知拓广文法G[S']：S'→S　S→AS|ε　A→aA|b

（1）试构造以LR（1）项目集为状态的识别活前缀的有穷自动机；

（2）试判断文法是否是LR(1)文法，并说明理由。



正常使用主观题需2.0以上版本雨课堂

作答


Thank You !