第七章
L R分析程序及其自动构造

7.1 LR分析概述

7.2 LR (0) 分析

7.3 SLR(1) 分析

7.4 使用二义文法

7.5 LR(1)分析

7.1 L R分析概述

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

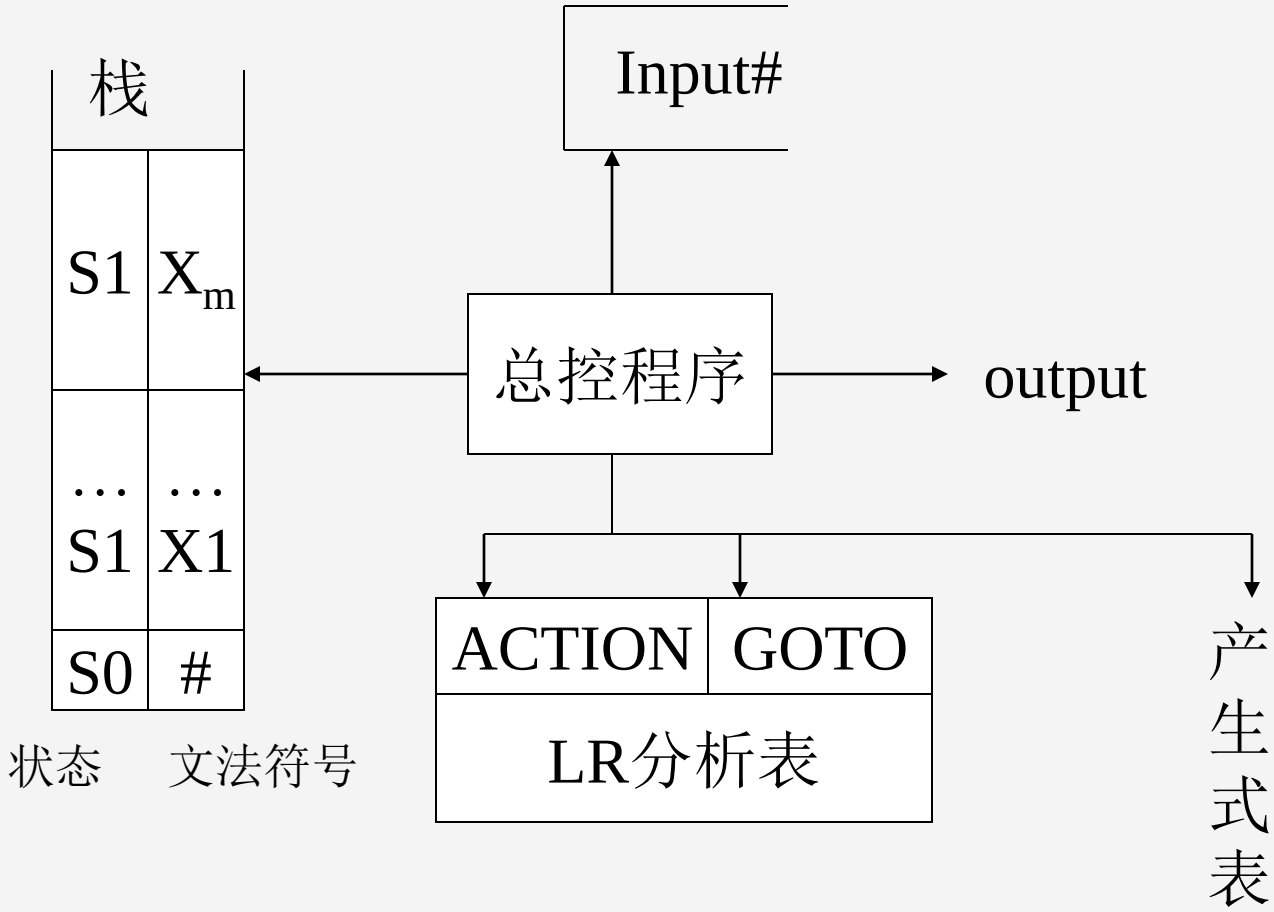
LR（0）:在分析过程中不需向右查看输入符号。

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


Input#
栈
S1
X_m
…
S1
…
X1
S0
#
状态
文法符号
总控程序
output
ACTION
GOTO
LR分析表
产生式表

例： G[S]: S →a A c B e [1] A →b [2]

A →Ab [3] B →d [4]

	ACTION						GOTO		
	a	c	e	b	d	#	S	A	B
0	S2						1		
1						acc			
2				S4				3	
3		S5		S6					
4	r2	r2	r2	r2	r2	r2			
5					S8				7
6	r3	r3	r3	r3	r3	r3			
7			S9						
8	r4	r4	r4	r4	r4	r4			
9	r1	r1	r1	r1	r1	r1			

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$A\rightarrow\beta$)

L R分析程序

» then begin

pop |β| 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=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

L R分析程序

例:

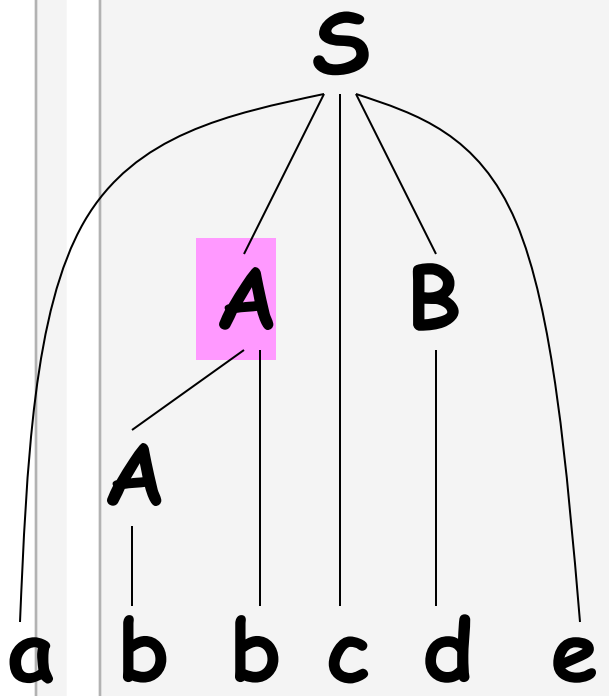
- G[S]: S →a A c B e [1]
- A →b [2]
- A →Ab [3]
- B →d [4]

w=abbcde#

Step	states.	Syms.	The rest of input	action goto
1	0	#	abbcde#	s2
2	02	#a	bbcde#	s4
3	024	#ab	bcde#	r2 goto(2,A)
4	023	#aA		s6
5	0236	#aAb	cde#	r3
6	023	#aA		s5
7	0235	#aAc	de#	s8
8	02358	#aAcd	e#	r4
9	02357	#aAcB		s9
10	023579	#aAcBe	#	r1
11	01	#S		acc

文法G[S]:

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



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
8)	# aAcd	e#	归约(B→d)
9)	#aAcB	e#	移进
10)	#aAcBe	#	归约
11)	#S	#	接受

对输入串abbcde#的移进-规约分析过程

符号串abbcde是否是G[S]的子

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

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

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

S_i:移进，将状态i和输入符进栈

r_i:归约，用第i个产生式归约，同时状态栈与符号栈退出相应个符号，并把GOTO表相应状态和第i个产生式的左部非终结符入栈。

	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			

构造LR分析表的预备知识

L R 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,**acc,**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。

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

LR分析

活前缀

G=(Vn,Vt,P,S),若有S’ $\overset{*}{\underset{R}{\Rightarrow}}$ **αAω** $\underset{R}{\Rightarrow}$ **αβω,**

γ是α β的前缀,则称是文法G的活前缀.

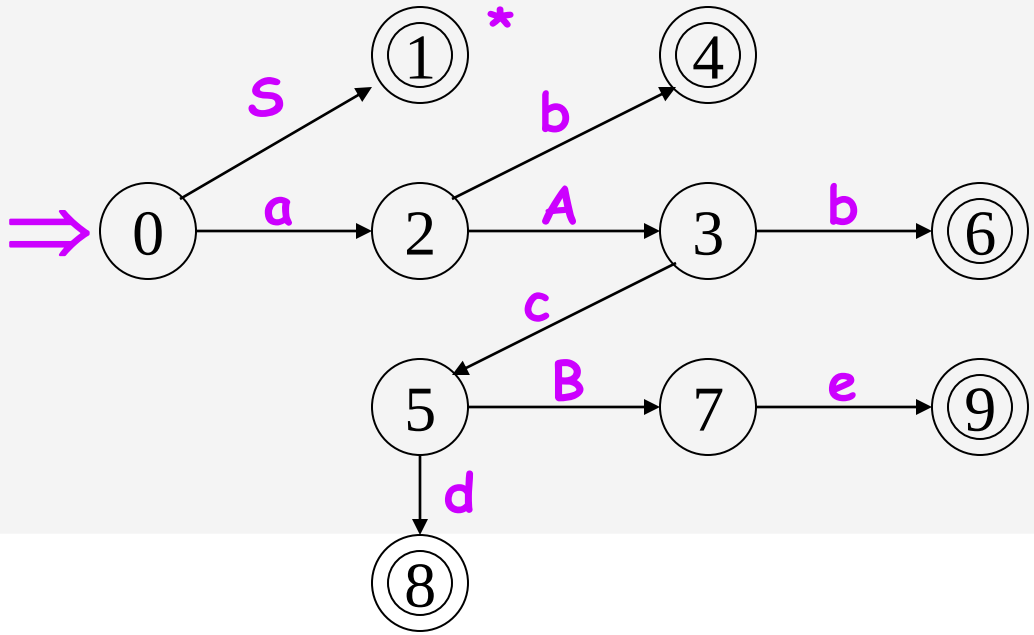
其中S’是对原文法扩充(S’→S)增加的非终结符.

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。

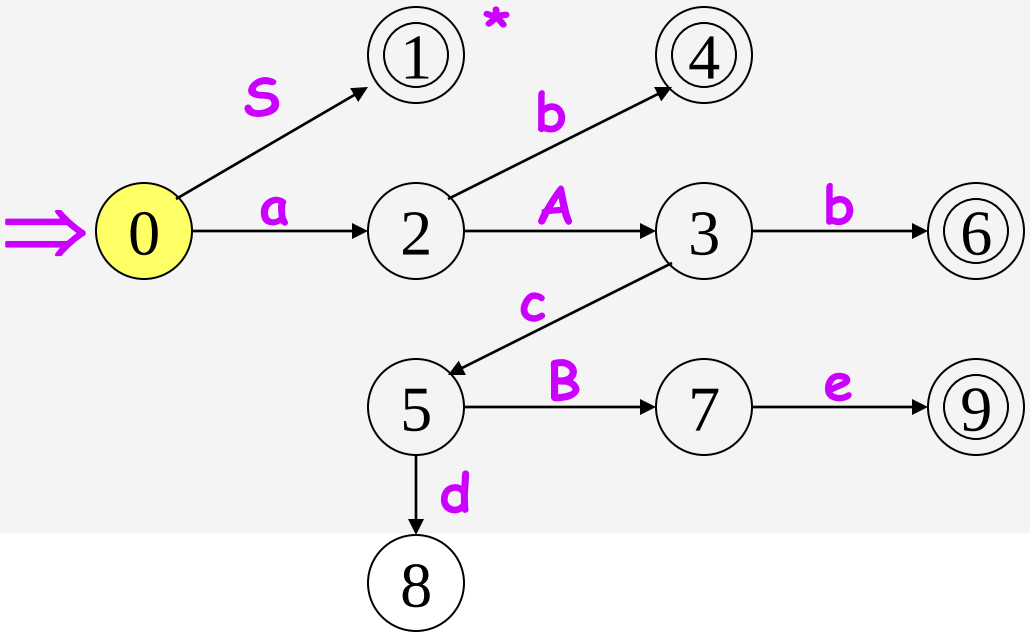
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	
5）	#aAb	cde#	归约(A→Ab)	0236	r_3	3
6）	#aA	cde#	移进	023	S_5	
7）	#aAc	de#	移进	0235	S_8	
8）	# aAcd	e#	归约(B→d)	02358	r_4	7
9）	#aAcB	e#	移进	02357	S_9	
10）	#aAcBe	#	归约(S→aAcBe)	023579	r_1	1
11）	#S	#	接受	01	acc	

对输入串abbcde#的LR分析过程



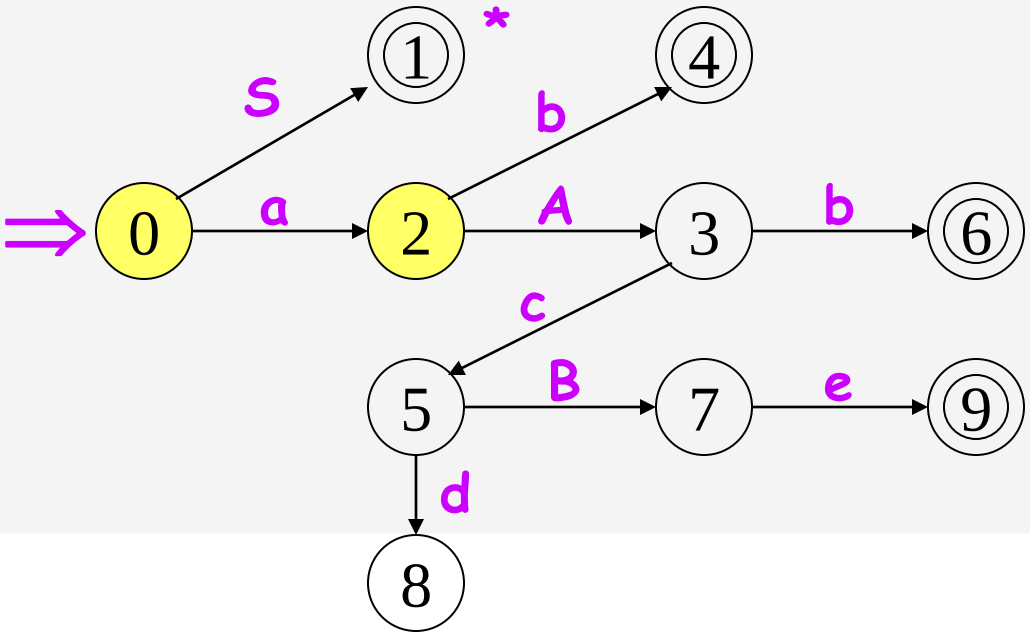
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	

对输入串abbcde#的LR分析过程



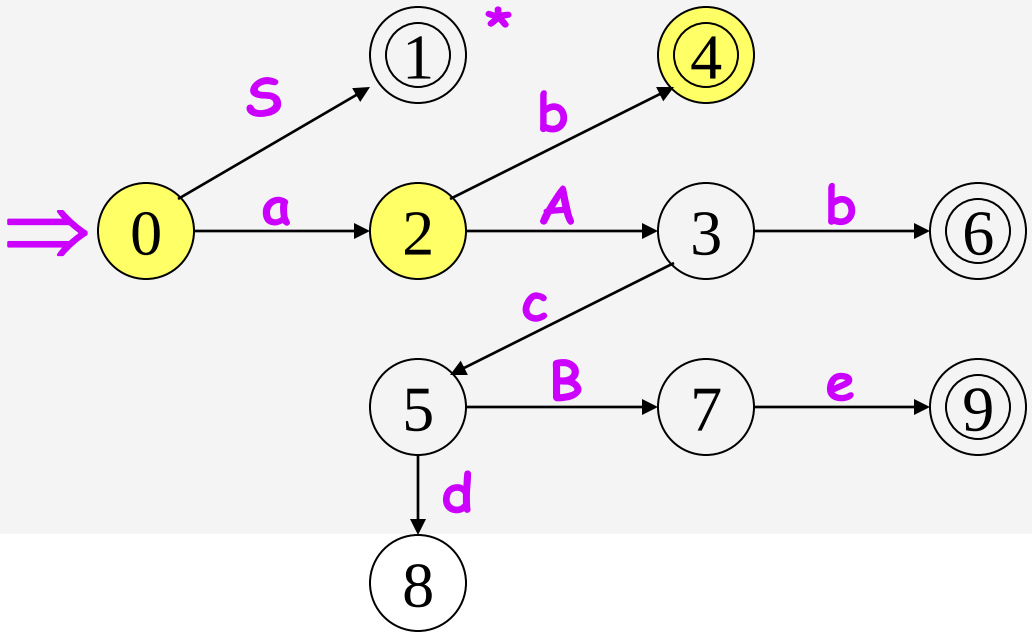
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	

对输入串abbcde#的LR分析过程



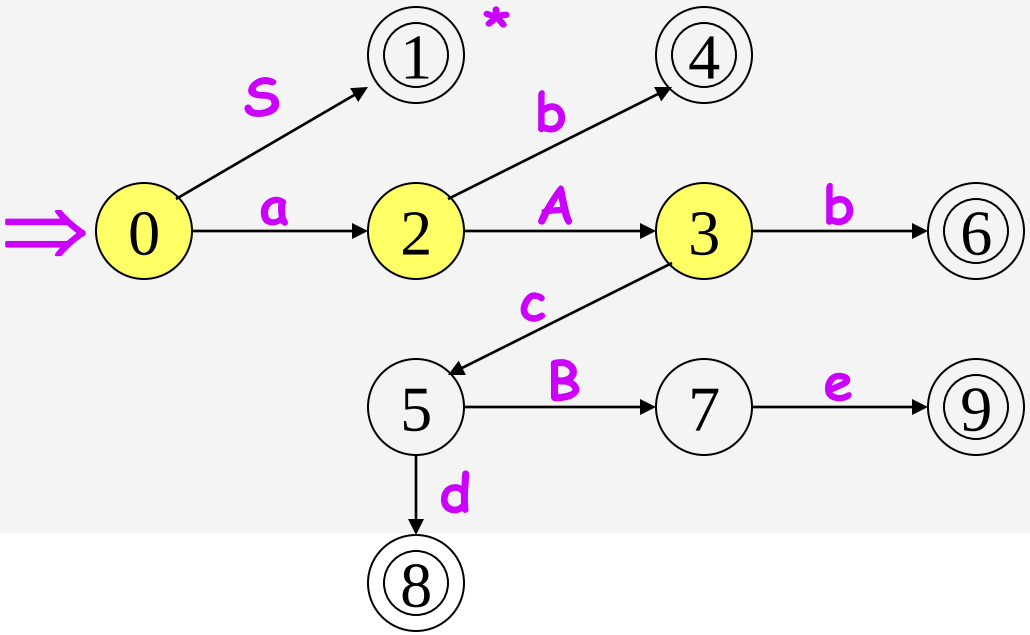
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3

对输入串abbcde#的LR分析过程



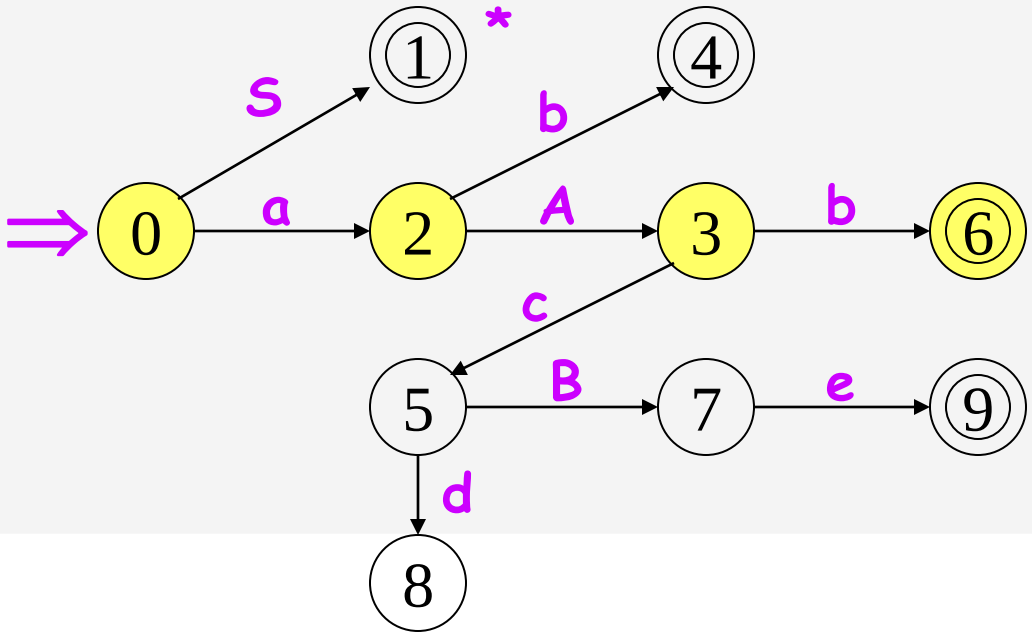
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	

对输入串abbcde#的LR分析过程



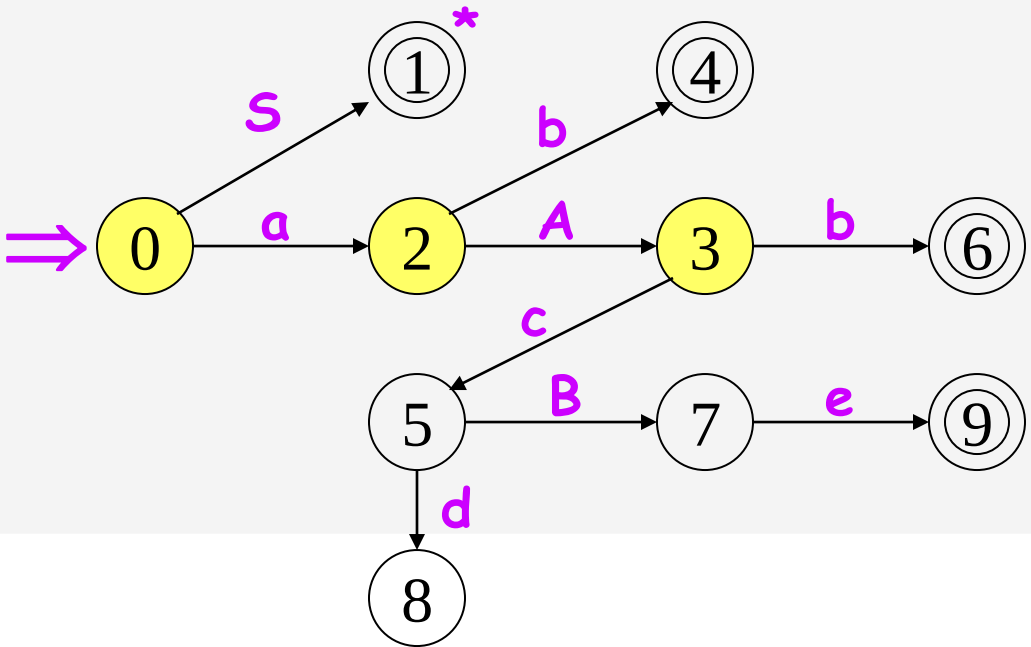
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	
5）	#aAb	cde#	归约(A→Ab)	0236	r_3	3

对输入串abbcde#的LR分析过程



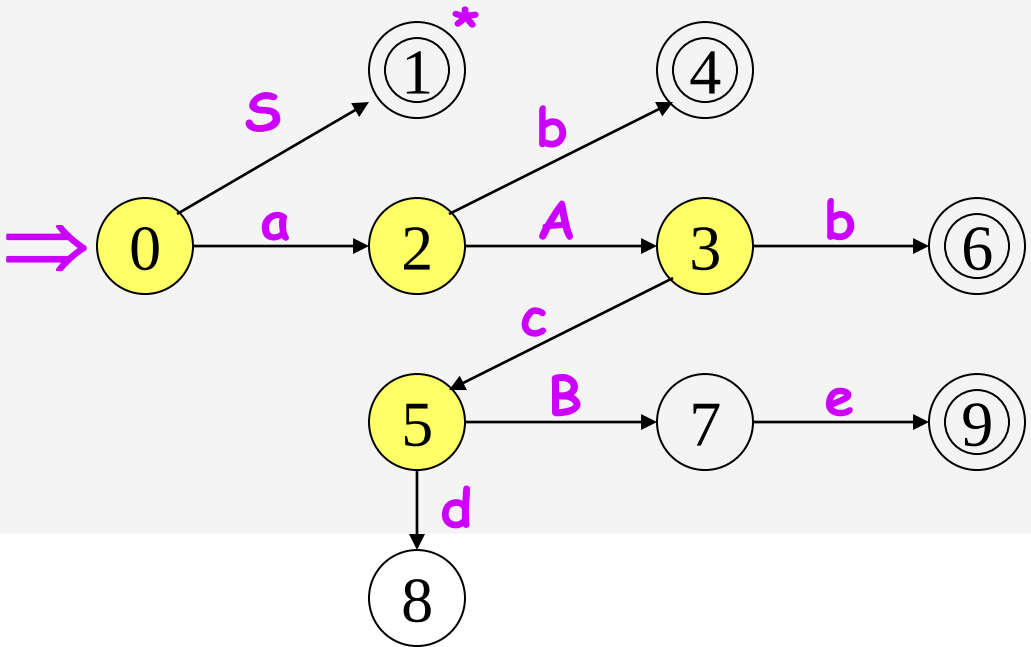
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	
5）	#aAb	cde#	归约(A→Ab)	0236	r_3	3
6）	#aA	cde#	移进	023	S_5	

对输入串abbcde#的LR分析过程



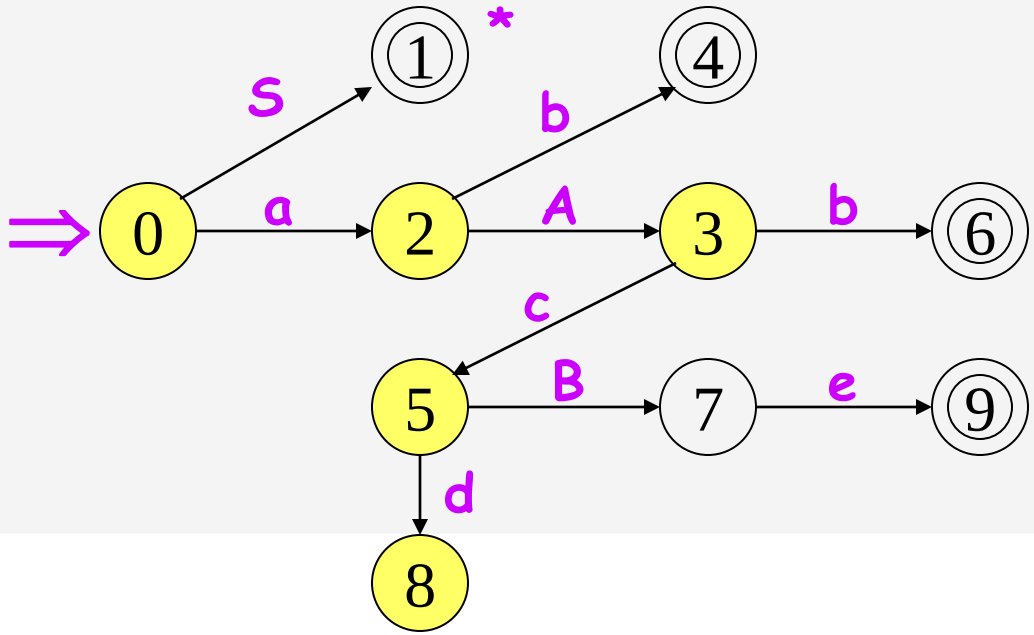
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	
5）	#aAb	cde#	归约(A→Ab)	0236	r_3	3
6）	#aA	cde#	移进	023	S_5	
7）	#aAc	de#	移进	0235	S_8	

对输入串abbcde#的LR分析过程



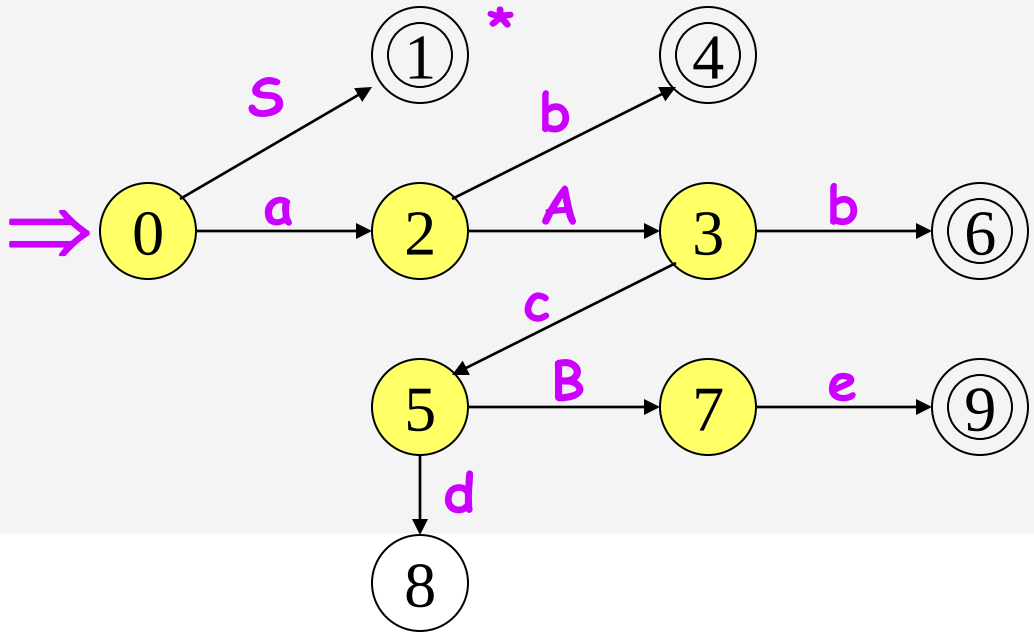
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	
5）	#aAb	cde#	归约(A→Ab)	0236	r_3	3
6）	#aA	cde#	移进	023	S_5	
7）	#aAc	de#	移进	0235	S_8	
8）	# aAcd	e#	归约(B→d)	02358	r_4	7

对输入串abbcde#的LR分析过程



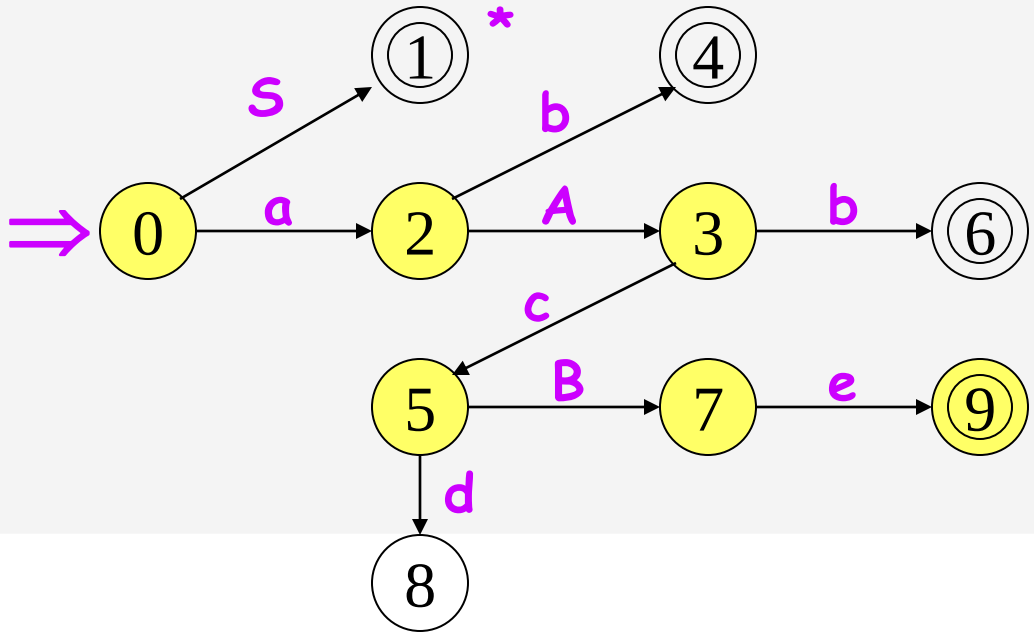
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	
5）	#aAb	cde#	归约(A→Ab)	0236	r_3	3
6）	#aA	cde#	移进	023	S_5	
7）	#aAc	de#	移进	0235	S_8	
8）	# aAcd	e#	归约(B→d)	02358	r_4	7
9）	#aAcB	e#	移进	02357	S_9	

对输入串abbcde#的LR分析过程



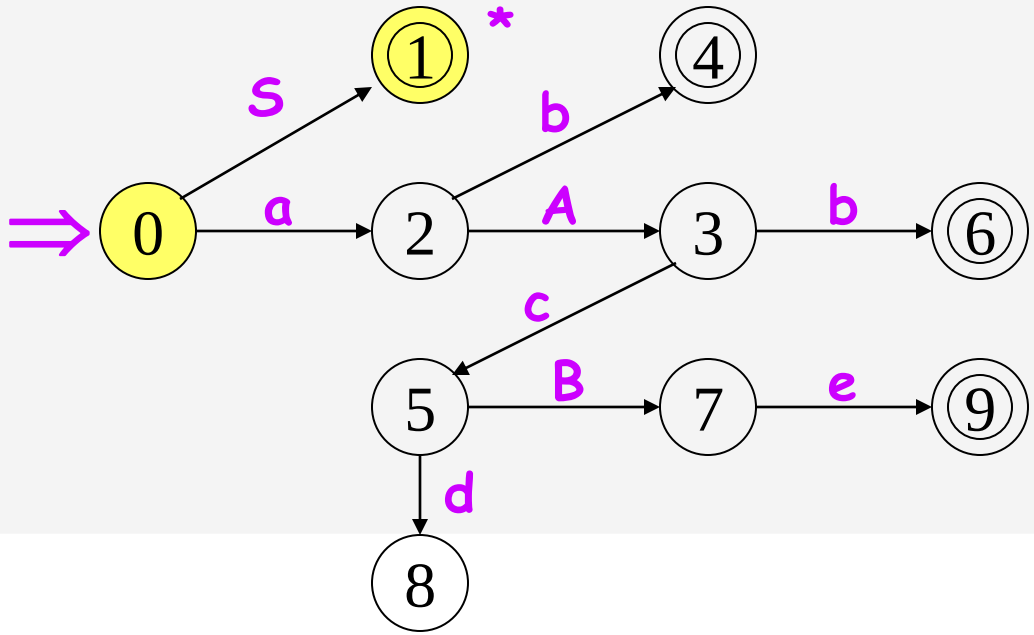
步骤	符号栈	输入符号串	动作	状态栈	ACTION	GOTO
1）	#	abbcde#	移进	0	S_2	
2）	#a	bbcde#	移进	02	S_4	
3）	#ab	bcde#	归约(A→b)	024	r_2	3
4）	#aA	bcde#	移进	023	S_6	
5）	#aAb	cde#	归约(A→Ab)	0236	r_3	3
6）	#aA	cde#	移进	023	S_5	
7）	#aAc	de#	移进	0235	S_8	
8）	# aAcd	e#	归约(B→d)	02358	r_4	7
9）	#aAcB	e#	移进	02357	S_9	
10）	#aAcBe	#	归约(S→aAcBe)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

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



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

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**

构造LR(0)项目集规范族

LR(0)项目集规范族(构成识别一个文法的活前缀的DFA的状态的全体)。

LR（0）项目或配置（ *item* or *configuration*）.

---在右端某一位置有圆点的G的产生式

A→ xyz　　A→.xyz

A→x.yz

A→xy.z

A→xyz.

如：S→aAd

S→.aAd　S→a .Ad　S→aA .d　S→aAd .

活前缀与句柄的关系：

G[S]:

若$S \overset{*}{\underset{R}{\Rightarrow}} \alpha A \omega \underset{R}{\Rightarrow} \alpha \beta \omega$ r是α β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→β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→$\beta_1\beta_2$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→β的右部所推出的符号串

活前缀,与句柄 ,与 LR(0)项目

为刻划这种分析过程中的文法G的每一个产生式的右部符号已有多大一部分被识别（出现在栈顶）的情况，分别用标有圆点的产生式来指示位置。

A→β．刻划产生式A→β的 右部β已出现在栈顶

A→β_1．β_2 刻划A→$\beta_1\beta_2$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

A→．β 刻划没有句柄的任何符号在栈顶，此时期望A→β的右部所推出的符号串

对于A→ε 的LR(0)项目只有A→．

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，α ,$\beta \in V^*$ 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 $S' \rightarrow S \cdot$

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

作用?

LR(0)项目集规范族

定义1：如果存在一个规范推导 $S \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta_1\beta_2 w$，我们说项目 $A \to \beta_1 \bullet \beta_2$ 对活前缀 $\gamma = \alpha\beta_1$ 是有效的。

定义2：若项目 $A \to \alpha \bullet B\beta_1$ 对活前缀 $\gamma = \delta\alpha$ 是有效的，且 $B \to \eta$ 是一个产生式，则项目 $B \to \bullet\eta$ 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。

LR（0） 项目集的闭包 CLOSURE

若当前处于A –> X•YZ刻划的情况，期望移进First(Y)中的某些符号，假如有产生式

Y –> u | w . 那么Y –> •u和Y –> •w这两个项目便是刻划期望移进 First(Y)中的某些符号的情况.

A –> X•YZ

Y –> •u

Y –> •w

这三个项目对应移进归约分析的同一个状态,这三个项目构成一个项目集，对应每个项目集，分析表将有一个状态.

构造识别活前缀的DFA

用闭包函数CLOSURE求DFA一个状态的项目集

若文法G已拓广为G’，而S为文法G的开始符号，拓广后增加产生式S’→S。如果I是G’的一个项目集，定义和构造I的闭包CLOSURE（I）如下：

1、I的项目均在CLOSURE（I）中。

2、若$A \to \alpha \bullet B\beta$属于CLOSURE（I）。则每一形如$B \to \bullet\eta$的项目也属于CLOSURE（I）。

3、重复2直到不出现新的项目为止。即CLOSURE（I）不再扩大。

转换函数 GO (I,x)= CLOSURE(J) ;

其中， I:项目集，x: 文法符号，

J={任何形如A→α x. β 的项目|A→α .x β ∈I}

LR(0)项目集规范族

计算LR（0）项目集规范族

$C=\{I_0 , I_1 , \ldots I_n\}$

Procedure itemsets(G');

Begin C := { CLOSURE ({S'→.S})}

Repeat

For C 中每一项目集I和每一文法符号x

Do if GO(I,x) 非空且不属于C

Then 把 GO(I,x) 放入C中

Until C 不再增大

End;

例：文法G:

（0）S`→E　　(1) E→aA　　(2) E→bB

(3) A→cA　　(4) A→d　　(5) B→cB

(7) B→d

LR(0) 项目集规范族（识别G的活前缀的DFA)：

I_0: S`→. E　　$I_1$: S`→E.　I_2: E→a. A

E→. aA　　A→.cA

E→. bB　　A→. d

I_3: E→b. B
B→. cB
B→. d

I_4: A→c. A
A→. cA
A →. d

I_5: B→c. B
B→. cB
B→. d

I_6: **E→aA.**

I_7: **E→bB.**

I_8: **A→cA.**

I_9: **B→cB.**

I_{10}: A→d.

I_{11}: B→cB.

DFA

LR(0)分析表的构造

假定C={I_0, I_1,……，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，……，n。令那个含有项目S`→．S的I_k的下标k为初态。ACTION和GOTO可按如下方法构造：

- 若项目A→α．aβ属于I_k且GO (I_k, a)= I_j, a为终结符，则置ACTION[k, a]为“把状态j和符号a移进栈”，简记为“sj”;
- 若项目A→α．属于I_k，那么，对任何终结符a，置ACTION[k, a]为“用产生式A→α进行规约”，简记为“rj”;其中，假定A→α为文法G`的第j个产生式;
- 若项目S`→S．属于I_k, 则置ACTION[k, #]为“接受”，简记为“acc”;
- 若GO (I_k, A)= I_j, A为非终结符，则置GOTO(k, A)=j;
- 分析表中凡不能用规则1至4填入信息的空白格均置上“出错标志”。

按上述算法构造的含有ACTION和GOTO两部分的分析表，如果每个入口不含多重定义，则称它为文法G的一张LR(0)表。具有LR(0)表的文法G称为一个LR（0）文法。

LR(0)文法是无二义的。

	ACTION					GOTO		
	a	c	b	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：（0）S`→S　　（1） S→rD

（2）D→D,i　　（3）D→i

LR（0）项目

1. S`→. S	2. S`→S.	3. S→. rD
4. S→r. D	5. S→rD.	6. D→. D,i
7. D→D. ,i	8. D→D,. i	9. D→D,i.
10. D→. i	11. D→i.	

例：（0）S`→S　　(1)　S→rD

　　(2) D→D,i　　(3)　D→i

LR（0）项目集规范族

I_0:　S`→. S　　I_3:　S→r D.

　　S→. r D　　D→D. , i

I_1:　S`→S.　　I_4:　D→i.

I_2:　S→r. D　　I_5:　D→D, . i

　　D→. D , i　　I_6:　D→D, i.

　　D→. i

其中I_3中含有移进/归约冲突

文法不是LR(0)的，如何解决？

SLR(1)技术

若 LR(0) 项目集规范族中有项目集I_K含 移进/归约、归约/归约冲突：

I_K :{ ...A→ α . b β , P → α . , Q → α . ,...}

存在“移进-归约”和“归约-归约”冲突。

解决冲突的方法是分析含P和Q的句型即考察FOLLOW(P)和FOLLOW(Q)

则解决冲突的SLR(1)技术：

当状态K面临当前输入符号a时：

若a=b，则**移进**

对a ∈FOLLOW (P) 则 action [K,a] =用 **P → α 归约**

对a ∈FOLLOW (Q) 则 action [K,a] =用 Q → **α 归约**

能用SLR(1)技术解决冲突的文法称为SLR(1)文法。

SLR(1)文法是无二义的。

SLR表

假定C={I_0, I_1,……，I_n}，令每个项目集I_k的下标k 为分析器的一个状态，因此，G` 的SLR分析表含有状态0，1，……，n。令那个含有项目S`→．S的I_k的下标k为初态。函数ACTION和GOTO可按如下方法构造：

- 若项目A→α．aβ 属于I_k且GO (I_k, a)= I_j, a为终结符，则置ACTION[k, a]为“把状态j和符号a移进栈”，简记为“sj”;
- 若项目A→α．属于I_k，那么，对任何输入符号a, a∈FOLLOW(A), 置ACTION[k, a]为“用产生式A→α 进行规约”，简记为“rj”;其中，假定A→α 为文法G`的第j个产生式;
- 若项目S`→S．属于I_k，则置ACTION[k, #]为“接受”，简记为“acc”;
- 若GO (I_k, A)= I_j, A为非终结符，则置GOTO(k, A)=j;
- 分析表中凡不能用规则1至4填入信息的空白格均置上“出错标志”。

按上述算法构造的含有ACTION和GOTO两部分的分析表，如果每个入口不含多重定义，则称它为文法G的一张SLR表。具有SLR表的文法G称为一个SLR（1）文法。数字1的意思是，在分析过程中顶多只要向前看一个符号。

实例说明文法的SLR（1）分析表

状态	ACTION				GOTO	
	r	,	i	#	S	D.
0	S_2				1	
1				acc		
2			S_4			3
3		S_5		r_1		
4		r_3		r_3		
5			S_6			
6		r_2		r_2		

二义文法在LR分析中的应用

例：

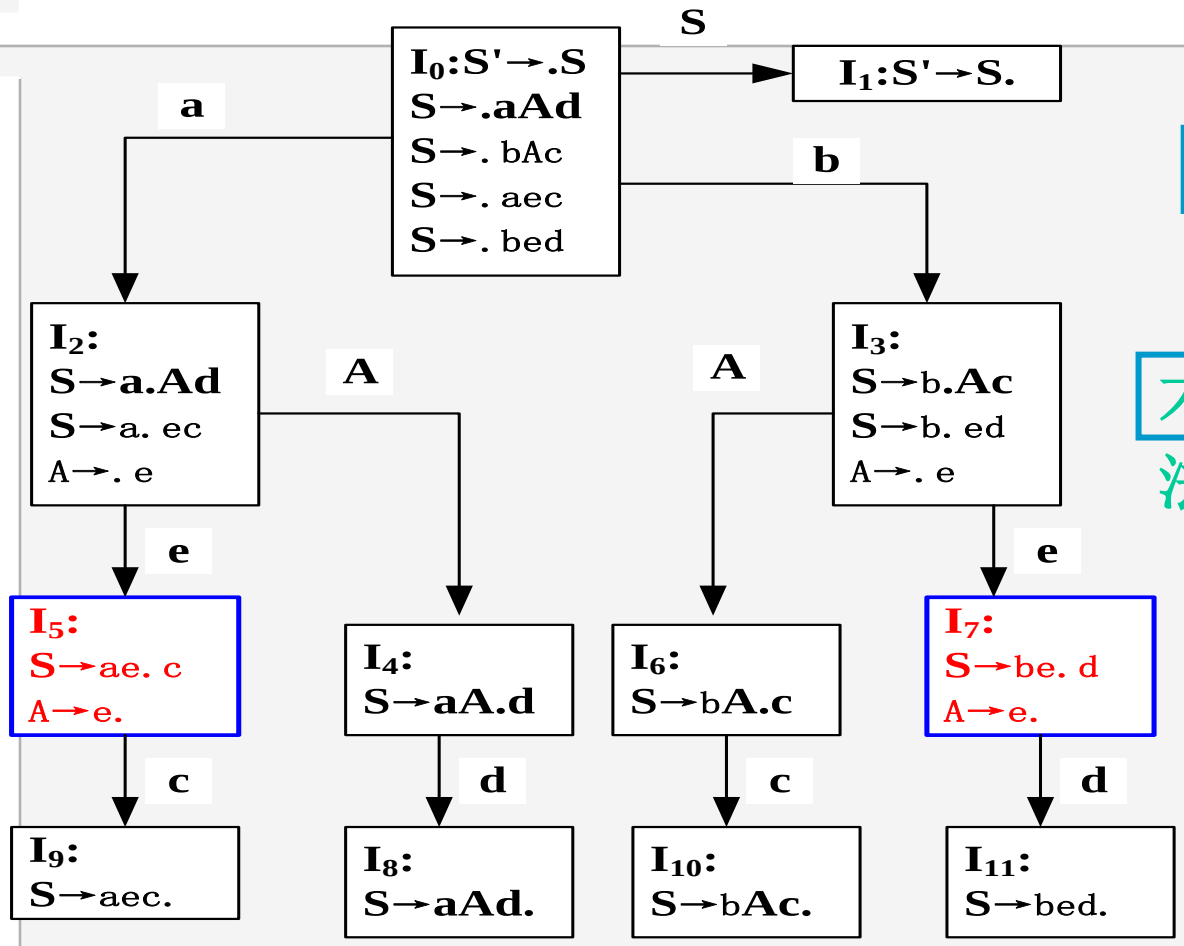
$E \rightarrow E+E|E*E|(E)|id$

$S \rightarrow iSeS|iS|a$

文法G:

（0）S′→S　(1) S→aAd　(2) S→bAc

(3) S→aec　(4) S→bed　(5) A→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→ae. c
A→e.

I_7:
S→be. d
A→e.

文法G:

- （0）S′→S　(1) S→aAd　(2) S→bAc
- (3) S→aec　(4) S→bed　(5) A→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′=>S=>aAd=>aed　　S′=>S=>bAc=>bec

S′=>S=>aec　　S′=>S=>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 $A \to \alpha\ .B\ \beta \quad \in I$

则 $B \to .\gamma$ （ $B \to \gamma$ 是一产生式） $\in I$

不防考虑，把FIRST（ β ）中的符号作为用 $B \to \gamma$ 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 →α .β， 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 –> α β •, a]:意味着处在栈中是α β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 –> u•, 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′→．S, #]})};

LR（1）项目集的构造：

- 对初始项目[S′→．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→β$_1$．Bβ$_2$，a]∈closure(I)，

B→δ是一产生式，那么对于FIRST(β$_2$a)中的每个终结符b，如果[B→．δ，b]不在closure(I)中，则把它加进去；

③重复①②，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)= closure(J)

其中

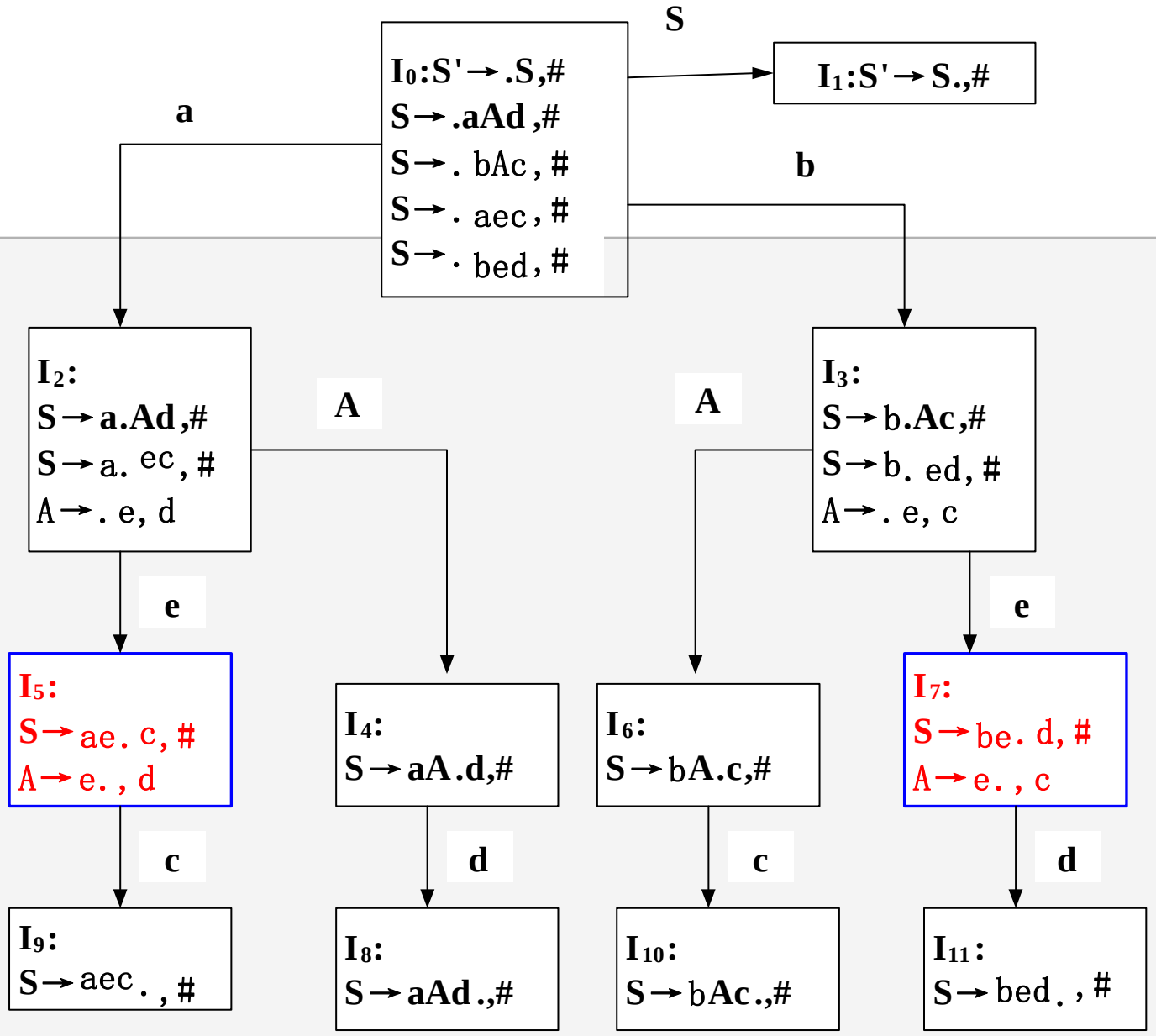
J={ 任何形如[A→ α X. β , a]的项目 | [A→ α . X β , a] ∈ I}

文法的LR(1)项目集:反复利用（1）（2），直到项目集不在扩大

LR(1)的项目集和转换函数

文法G:

（0）S′→S
(1) S→aAd
(2) S→bAc
(3) S→aec
(4) S→bed
(5) A→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,……， 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，……，n。

1. 令含有项目[S′→．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→α．,b]属于I_k，那么置ACTION[k, b]为 “r_j”;其中，假定A→α为文法G′的第j个产生式

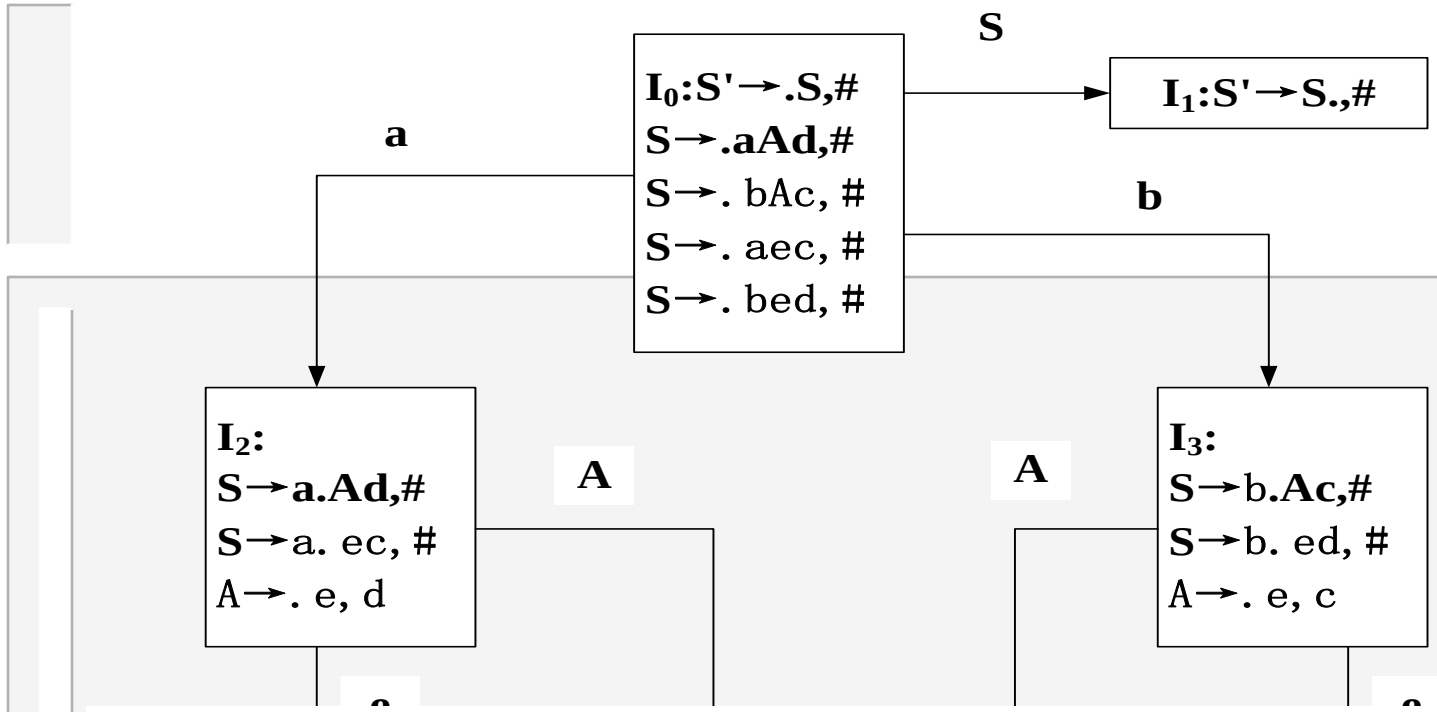
3. 若项目[A→α．aβ，b]属于I_k且GO（I_k，a)= I_j,则置ACTION[k, a]为“s_j”;

4.若项目[S′→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

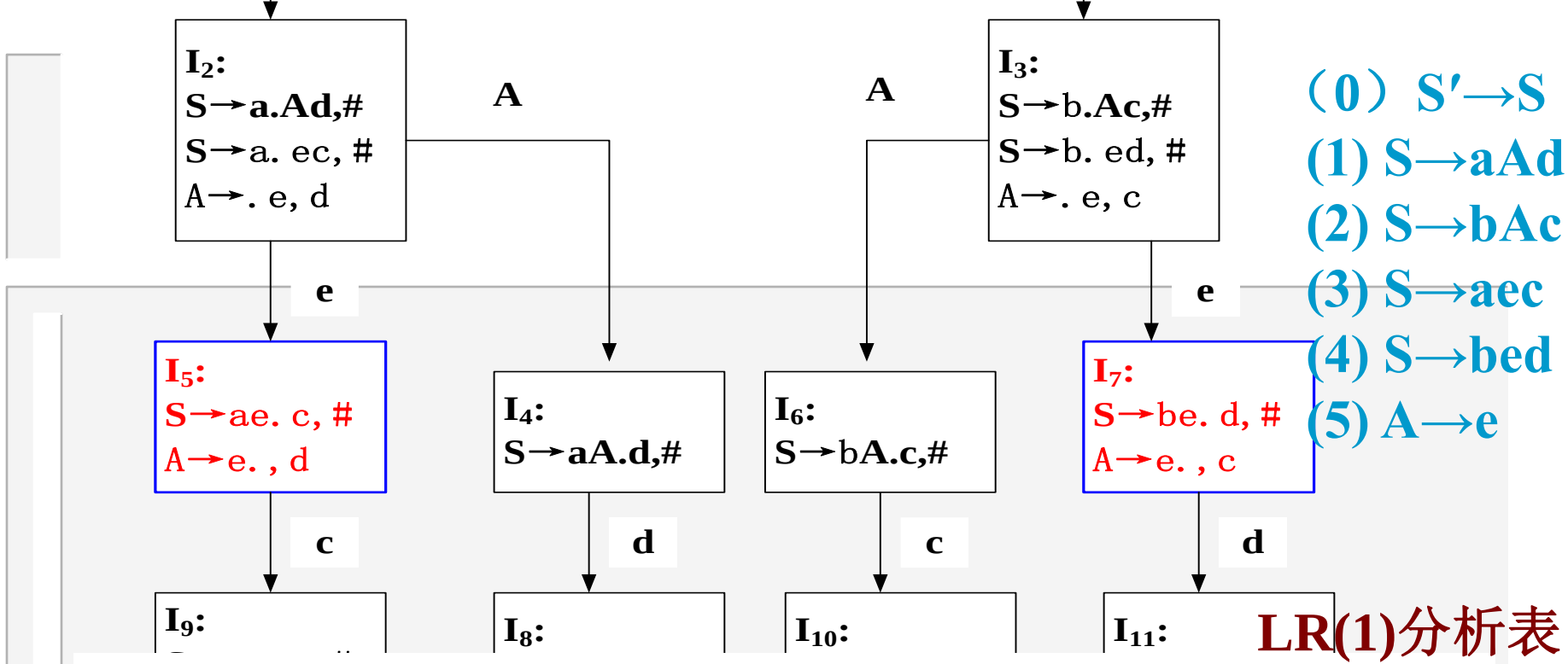
6.分析表分析中凡不能用规则1至5填入信息的空白格均置上“出错标志”。

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		



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

LR(1)分析表

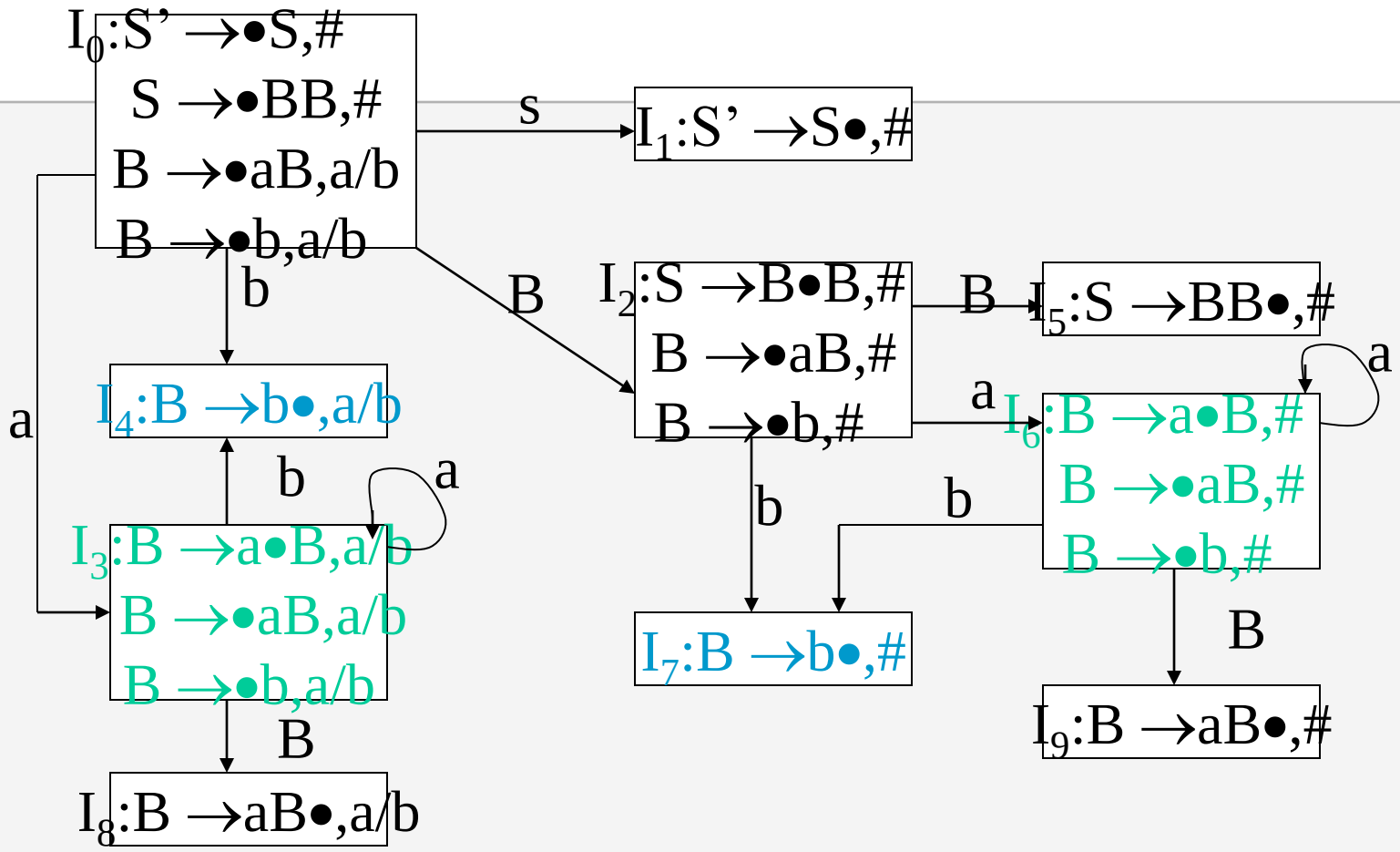
	ACTION						GOTO	
	a	b	c	d	e	#	S	A
0	S2	S3					1	
1						acc		
2					S5			4
3					S7			6
4				S8				
5			S9	r5				
6								

每个SLR（1）文法都是LR（1）文法。

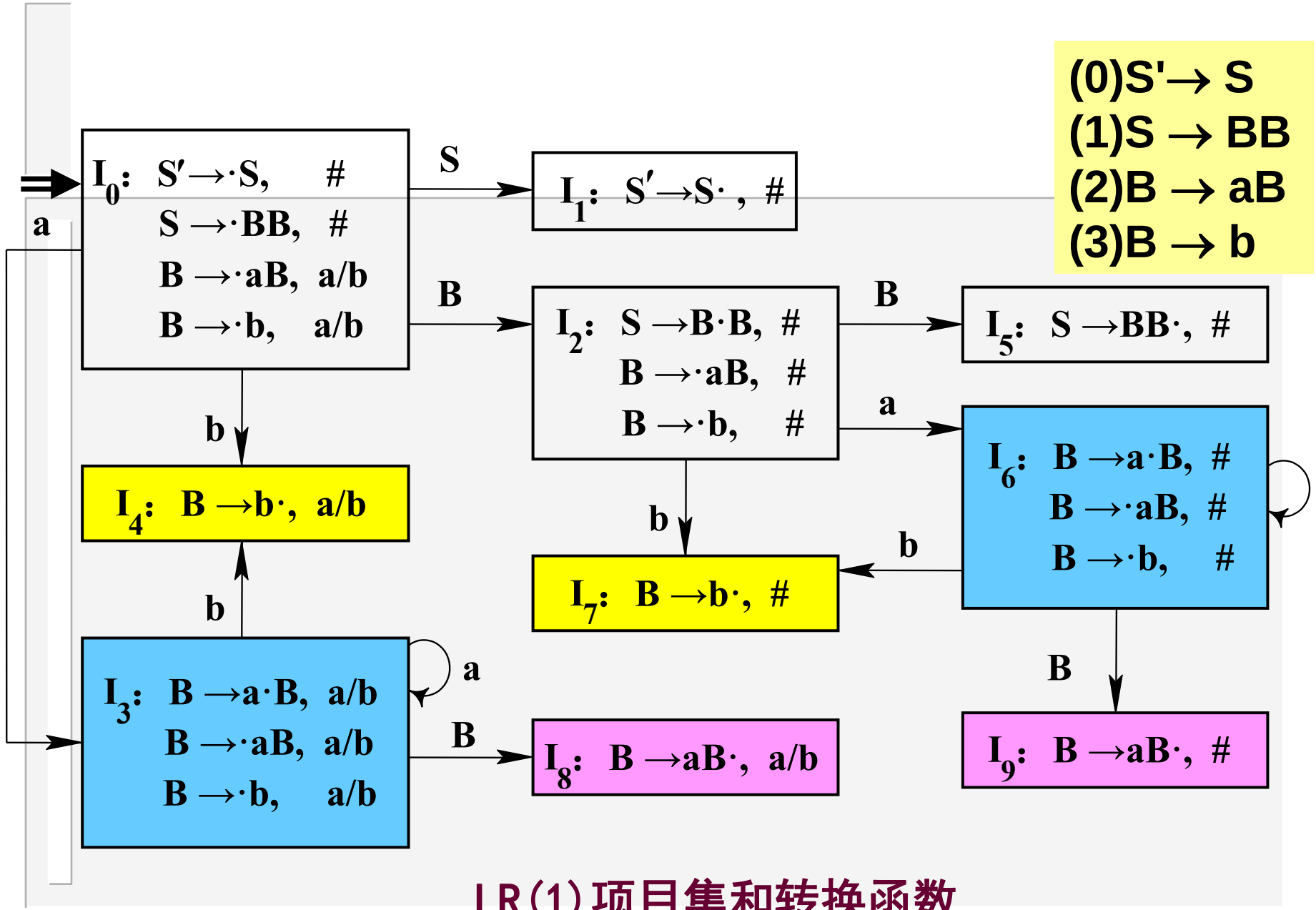
LR(1)比SLR（1）能力强

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’→S (1)S → BB (2) B→aB (3) B →b



LR(1)项目集和转换函数



LR(1)项目集和转换函数